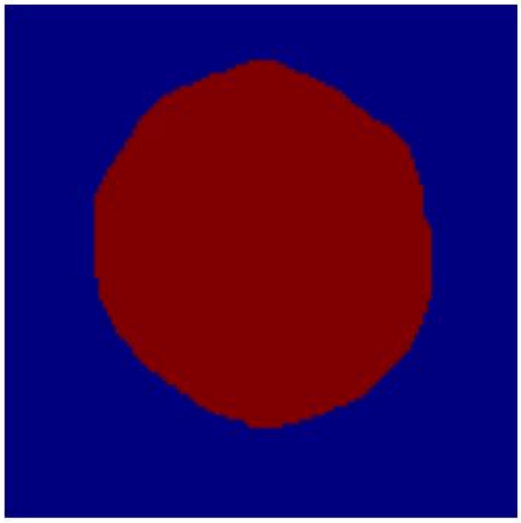
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os


def load_image(image_path):
    return cv2.imread(image_path)


def preprocess_mask(mask):

    binary = np.where(mask[..., 0] > 127, 255, 0).astype(np.uint8)  # Using
only the Red channel
    return binary


def connected_component_labeling(binary_mask):
    num_labels, labels = cv2.connectedComponents(binary_mask, connectivity=8)
    return num_labels, labels


def visualize_segmentation(original, labels, title="Segmented WBC"):
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(labels, cmap='jet')
    plt.title(title)
    plt.axis("off")

    plt.show()


def process_images(image_folder, mask_folder):
    image_files = {os.path.splitext(f)[0]: f for f in
os.listdir(image_folder) if f.endswith('.bmp')}
    mask_files = {os.path.splitext(f)[0]: f for f in os.listdir(mask_folder)
if f.endswith('.png')}

    common_files = sorted(set(image_files.keys()) & set(mask_files.keys()))

    for filename in common_files:
        image_path = os.path.join(image_folder, image_files[filename])
        mask_path = os.path.join(mask_folder, mask_files[filename])

        image = load_image(image_path)
        mask = load_image(mask_path)
        binary_mask = preprocess_mask(mask)
        num_labels, labels = connected_component_labeling(binary_mask)
```

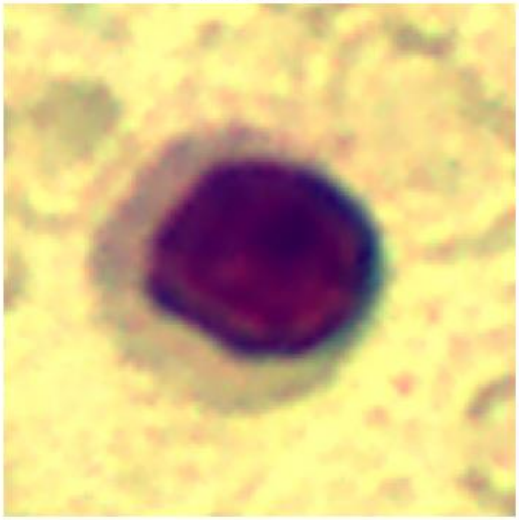
```
        visualize_segmentation(image, labels, title=filename)
        print(f"{filename}: {num_labels} connected components found.")


colorful_images_folder = r"C:\Users\immad\Downloads\cell_images"
grayscale_masks_folder = r"C:\Users\immad\Downloads\masks"

process_images(colorful_images_folder, grayscale_masks_folder)
```

Original Image

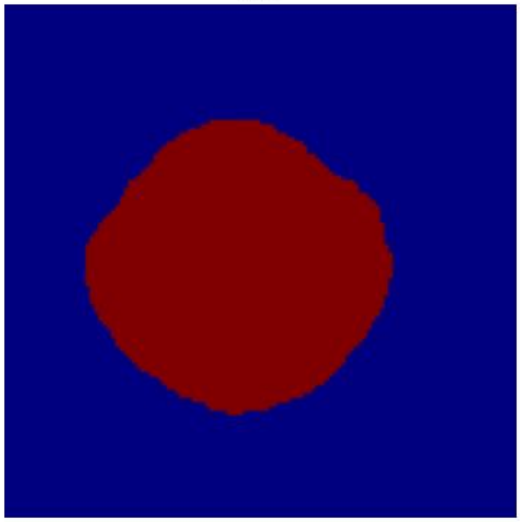
006

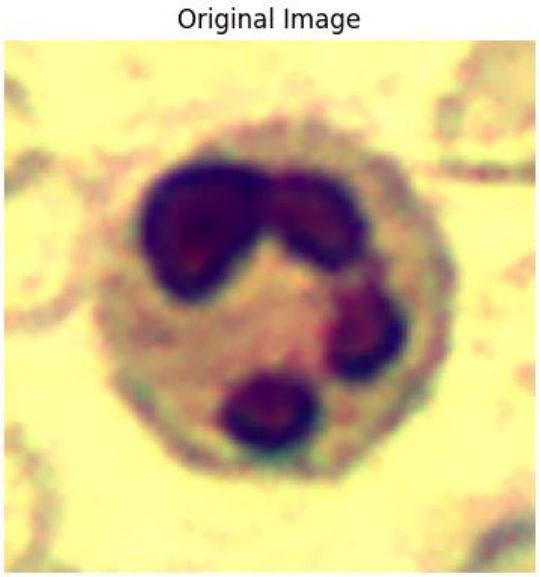


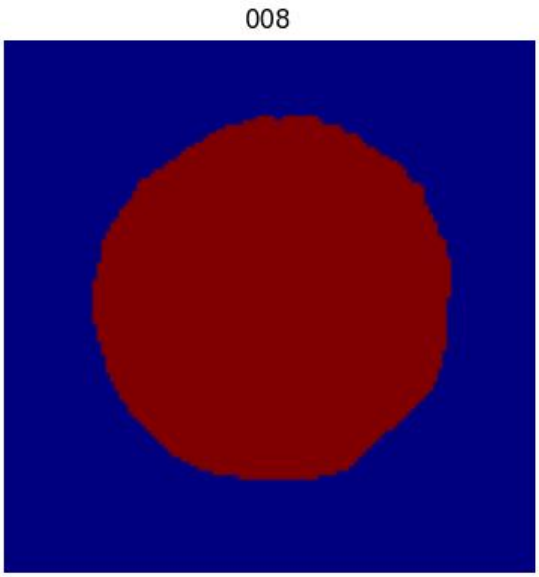
Original Image



007







V Set

1. Images and Masks
2. Morphological Processing
3. Dice Coefficient Calculation
4. File Processing

Steps

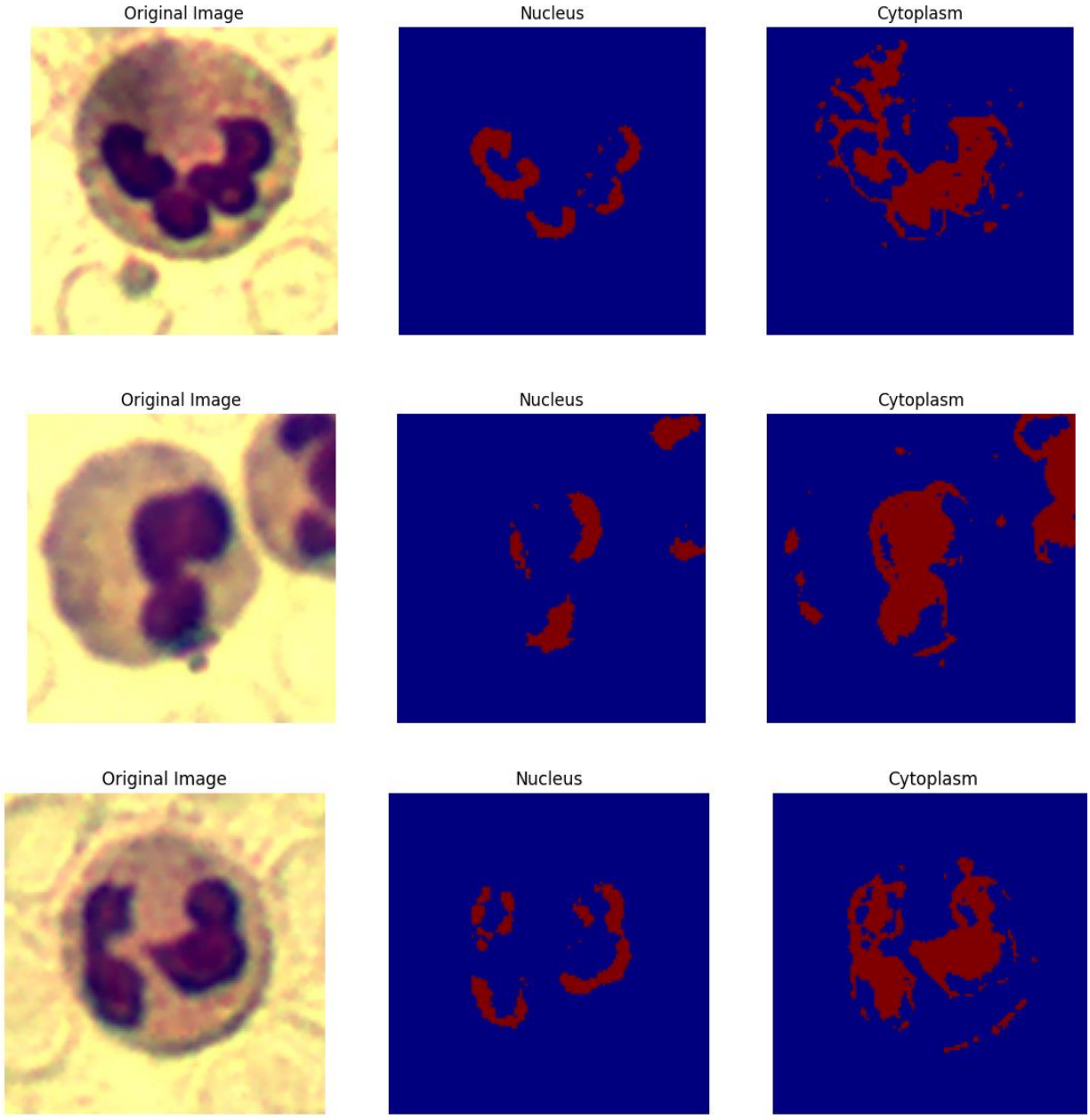
1. Loading the Data
2. Preprocessing the Mask
3. Applying Morphological Operations
4. Computing Dice Coefficients

TASK 2:

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

```
def load_image(image_path):
    return cv2.imread(image_path)

def preprocess_mask(mask):
    binary = np.zeros_like(mask, dtype=np.uint8)
    binary[mask > 127] = 255  # Manually thresholding based on a fixed value
    return binary

def apply_morphology(binary_mask):
    kernel = np.ones((3, 3), np.uint8)
    cleaned_mask = cv2.morphologyEx(binary_mask, cv2.MORPH_OPEN, kernel,
iterations=2)
    return cleaned_mask

def extract_nucleus_cytoplasm(image):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    lower_nucleus = np.array([100, 50, 50])
    upper_nucleus = np.array([140, 255, 255])
    nucleus_mask = cv2.inRange(hsv, lower_nucleus, upper_nucleus)

    lower_cytoplasm = np.array([140, 20, 20])
    upper_cytoplasm = np.array([180, 255, 255])
    cytoplasm_mask = cv2.inRange(hsv, lower_cytoplasm, upper_cytoplasm)

    return nucleus_mask, cytoplasm_mask

def kmeans_segmentation(image, k=3):
    pixel_values = image.reshape((-1, 3))
    pixel_values = np.float32(pixel_values)

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
    _, labels, centers = cv2.kmeans(pixel_values, k, None, criteria, 10,
cv2.KMEANS_RANDOM_CENTERS)
    centers = np.uint8(centers)
    segmented_image = centers[labels.flatten()].reshape(image.shape)
    return segmented_image, labels.reshape(image.shape[:2])

def visualize_segmentation(original, nucleus, cytoplasm, title="Segmented
WBC"):
    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(cv2.cvtColor(original, cv2.COLOR_BGR2RGB))
    plt.title("Original Image")
    plt.axis("off")

    plt.subplot(1, 3, 2)
    plt.imshow(nucleus, cmap='jet')
    plt.title("Nucleus")
    plt.axis("off")

    plt.subplot(1, 3, 3)
    plt.imshow(cytoplasm, cmap='jet')
    plt.title("Cytoplasm")
    plt.axis("off")
```

```
    plt.show()

def process_images(image_folder, mask_folder):
    image_files = {os.path.splitext(f)[0]: f for f in
os.listdir(image_folder) if f.endswith('.bmp')}
    mask_files = {os.path.splitext(f)[0]: f for f in os.listdir(mask_folder)
if f.endswith('.png')}

    common_files = sorted(set(image_files.keys()) & set(mask_files.keys()))

    for filename in common_files:
        image_path = os.path.join(image_folder, image_files[filename])
        image = load_image(image_path)

        nucleus_mask, cytoplasm_mask = extract_nucleus_cytoplasm(image)
        visualize_segmentation(image, nucleus_mask, cytoplasm_mask,
title=filename)
        print(f"{filename}: Segmentation completed.")

colorful_images_folder = r"C:\Users\immad\Downloads\cell_images"
grayscale_masks_folder = r"C:\Users\immad\Downloads\masks"

process_images(colorful_images_folder, grayscale_masks_folder)
```



V values

- **Background:** Low V values (dark regions).
- **Nucleus:** Medium V values (moderate intensity regions).
- **Cytoplasm:** High V values (brighter regions).

Steps:

Convert Image to HSV Color Space

Apply Histogram Analysis on V-Channel

Segmentation Using Thresholding

Morphological Operations

K-Means Clustering for Refinement

TASK 3:

V Set

1. Grayscale Intensity Distribution.
2. HSV Color Space
3. Morphological Features
4. Clustering

Steps

1. Data Loading & Preprocessing

2. Segmentation of Nucleus and Cytoplasm

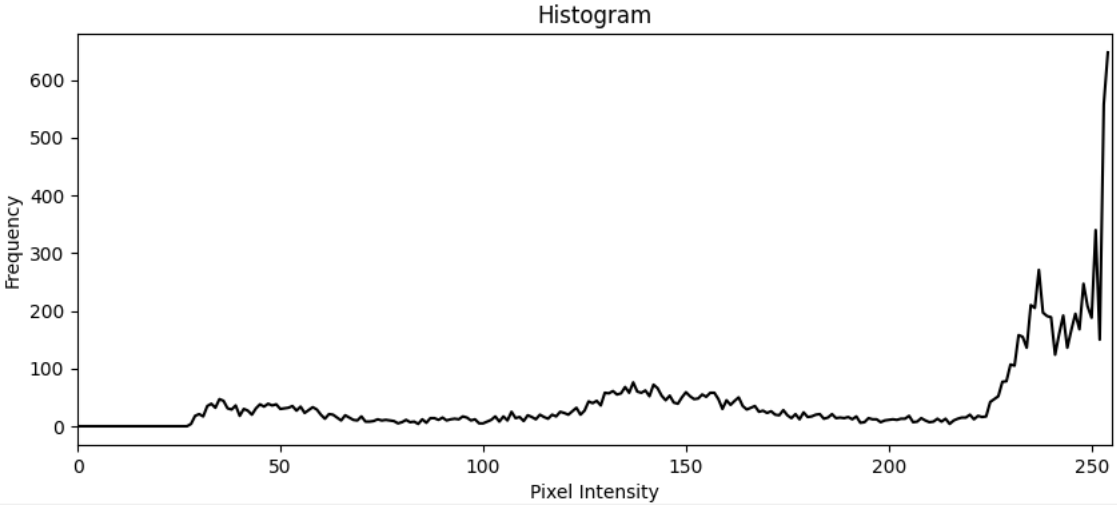
3. Refinement Using Morphological Operations

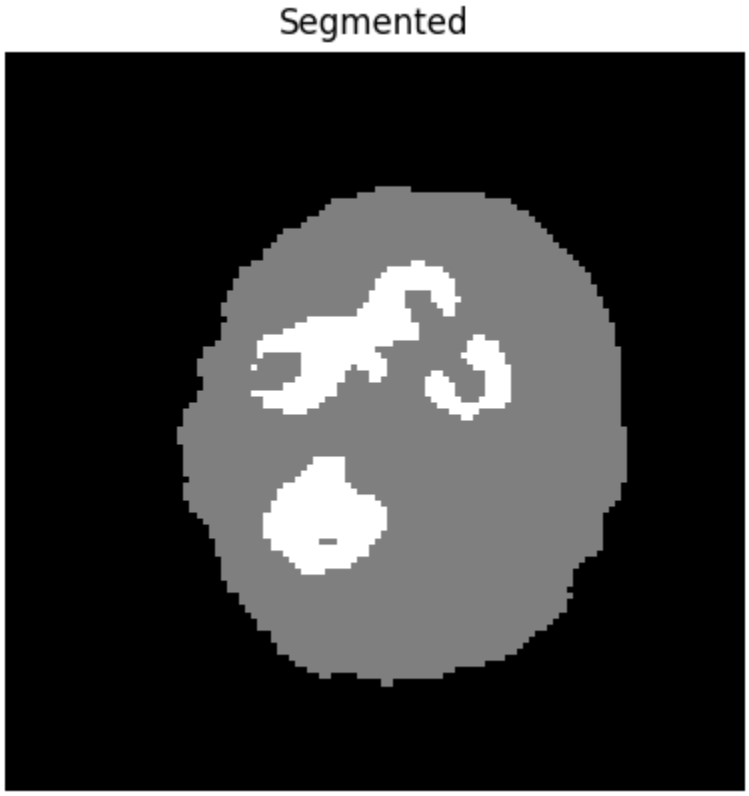
4. K-Means Clustering for Further Segmentation

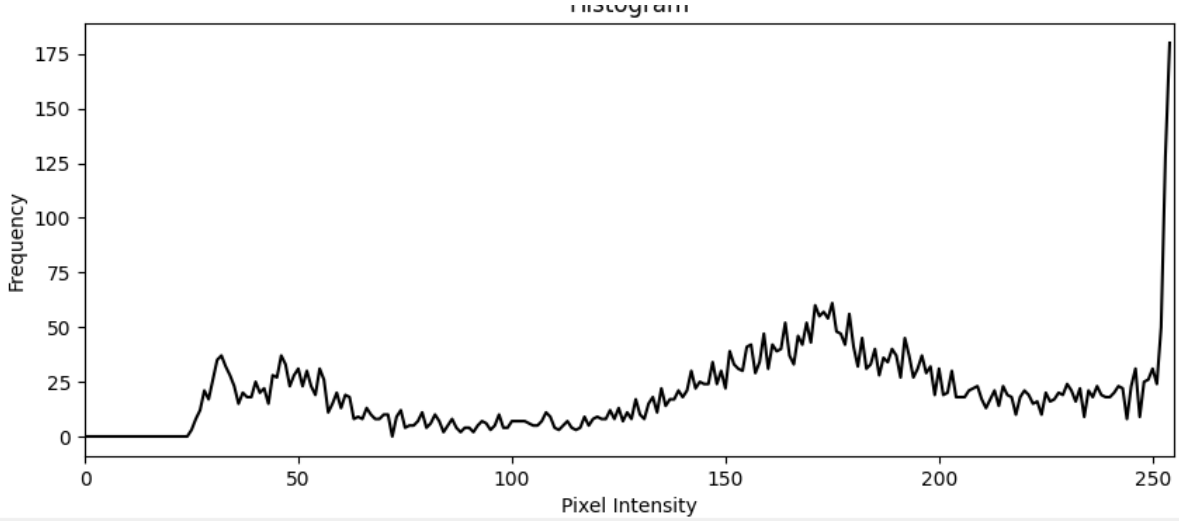
5. Compute Dice Coefficient for Each Class

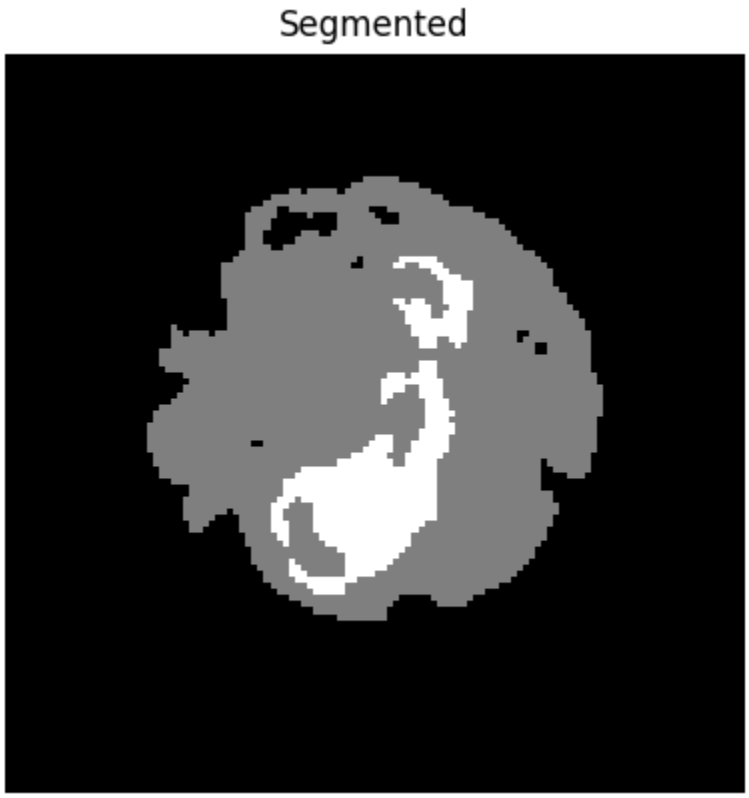
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def plot_histograms(image, title):
    plt.figure(figsize=(10, 4))
    hist = cv2.calcHist([image], [0], None, [255], [0, 255])
    plt.plot(hist, color='black')
    plt.xlim([0, 255])
    plt.title(title)
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.show()

def compute_weighted_dice_coefficient(gen_mask, orig_mask):

    gen_mask = (gen_mask > 0).astype(np.float32)
    orig_mask = (orig_mask > 0).astype(np.float32)


    intersection = np.sum(gen_mask * orig_mask)


    sum_masks = np.sum(gen_mask) + np.sum(orig_mask)

    dice_weighted = (2.0 * intersection) / sum_masks if sum_masks > 0 else
```

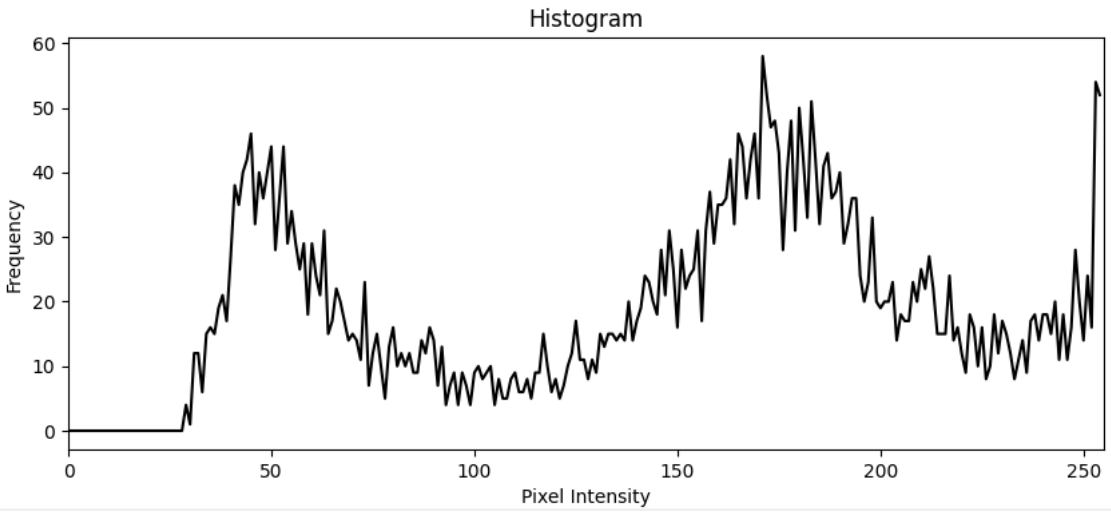
```
1.0

    return dice_weighted


image = cv2.imread(r"C:\Users\immad\Downloads\test_images\243.bmp",
cv2.IMREAD_GRAYSCALE)
original_mask = cv2.imread(r"C:\Users\immad
mazhar\Downloads\test_mask\243.png", cv2.IMREAD_GRAYSCALE)

plot_histograms(image, "Histogram")

low_threshold = 50
high_threshold = 200


mask1 = image < low_threshold
mask2 = image > high_threshold
mask3 = (~mask1) & (~mask2)


segmented = np.zeros_like(image)
segmented[mask1] = 255
segmented[mask2] = 0
segmented[mask3] = 127



dice_weighted = compute_weighted_dice_coefficient(segmented, original_mask)
print(f"Weighted Dice Coefficient: {dice_weighted:.4f}")


plt.imshow(segmented, cmap='gray')
plt.title("Segmented")
plt.axis("off")
plt.show()
```

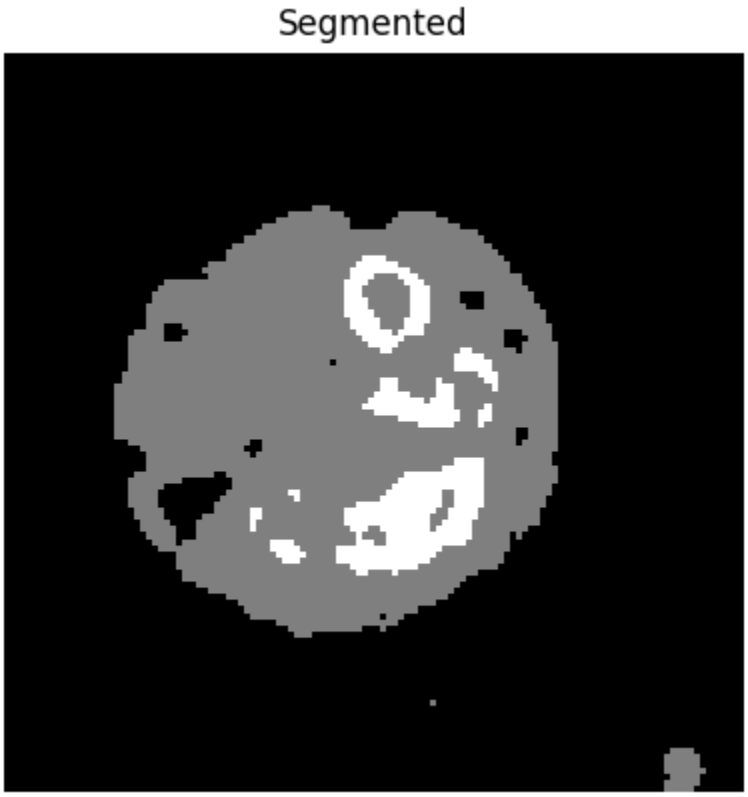

Histogram
Frequency
Pixel Intensity
0
100
200
300
400
500
600
0
50
100
150
200
250


Segmented


Histogram
Frequency
Pixel Intensity
0
25
50
75
100
125
150
175
0
50
100
150
200
250


Segmented






```
Weighted Dice Coefficient: 0.9472

Process finished with exit code 0
```