EE7204 – COMPUTER VISION AND IMAGE PROCESSING

TAKE HOME ASSIGNMENT 2

Name : Hashika W.D.E.

RegNo : EG/2019/3598

Semester : 07

Date : 21/04/2024

GitHub Link :

Task 1

Consider an image with 2 objects and a total of 3-pixel values (1 for each object and one for the background). Add Gaussian noise to the image. Implement and test Otsu's algorithm with this image.

1. **Generating an image with 3 pixels, where there are two objects (circle and traingle) and the background**

<u>Code</u>

```
# Create a blank white image
image = np.ones((height, width), dtype=np.uint8) * 255

# Draw a circle
cv2.circle(image, (height//3, width//2), 50, (0, 0, 0), -1)  # Circle with center (height//3, width//2) and radius 50

# Draw a triangle
pts = np.array([[2*height//3, width//3], [2*height//3 + 60, width//3 + 100], [2*height//3 - 60, width//3 + 100]], np.int32)
cv2.fillPoly(image, [pts], (155,155,155))  # Triangle with vertices defined in pts list
```

Figure 1 . Code of generating an image with 3 pixels, where there are two objects (circle and riangle) and the background

<u>Output</u>

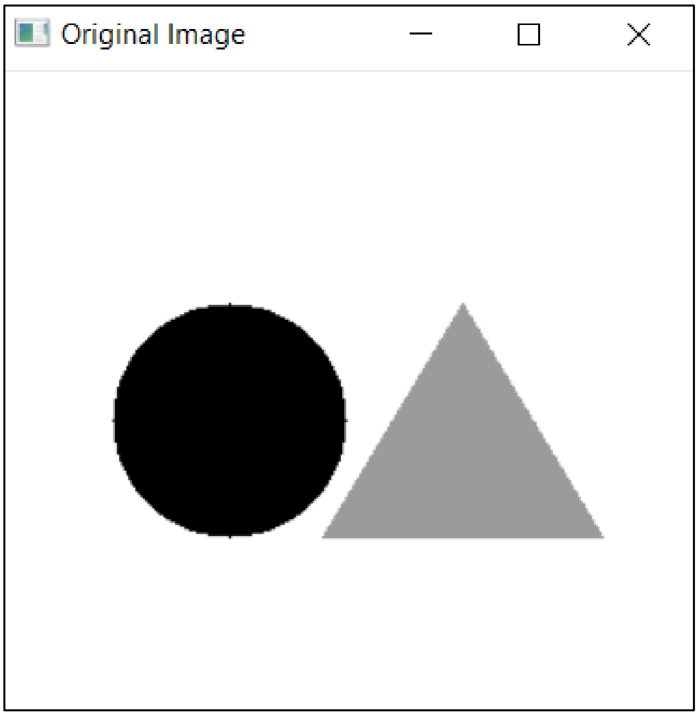

Figure 2. Generated image

2. Defining the Otsu's Algorithm

Code

```
# Function to implement Otsu's algorithm for thresholding
def otsu_threshold(image):
    # Calculate histogram
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])
    hist_norm = hist.ravel() / hist.sum()
    best_threshold = -1
    best_variance = 0

    # Iterate over all possible threshold values
    for threshold in range(256):
        # Calculate background and foreground probabilities and means
        w0 = np.sum(hist_norm[:threshold])/np.sum(hist_norm[:256])
        if w0 == 0:
            continue
        mean0 = np.sum(np.arange(threshold) * hist_norm[:threshold]) / np.sum(hist_norm[:threshold])

        w1 = np.sum(hist_norm[threshold:])/np.sum(hist_norm[:256])
        if w1 == 0:
            continue
        mean1 = np.sum(np.arange(threshold, 256) * hist_norm[threshold:]) / np.sum(hist_norm[threshold:])

        # Calculate within class variance
        within_class_variance = w0 * w1 *(mean0-mean1)**2

        # Update best threshold if variance is improved
        if within_class_variance > best_variance:
            best_threshold = threshold
            best_variance = within_class_variance

    # Apply thresholding using the best threshold value
    _, otsu_image = cv2.threshold(image, best_threshold, 255, cv2.THRESH_BINARY)
    return otsu_image
```

Figure 3. Code of defining the Otsu's Algorithm

3. Defining a Gaussian noise to the image(with mean 0 and variance 30)

Code

```
# Function to add Gaussian noise to an image
def add_gaussian_noise(image, mean=0, sigma=30):
    # Generate Gaussian noise with specified mean and sigma
    row, col = image.shape
    gauss = np.random.normal(mean, sigma, (row, col))
    # Add the noise to the image
    noisy_image = image + gauss
    # Clip values to ensure they are within 0-255 range
    noisy_image = np.clip(noisy_image, 0, 255)
    # Convert back to uint8
    noisy_image = noisy_image.astype(np.uint8)
    return noisy_image
```

Figure 4. Code for adding the Gaussian noise to the image

Output

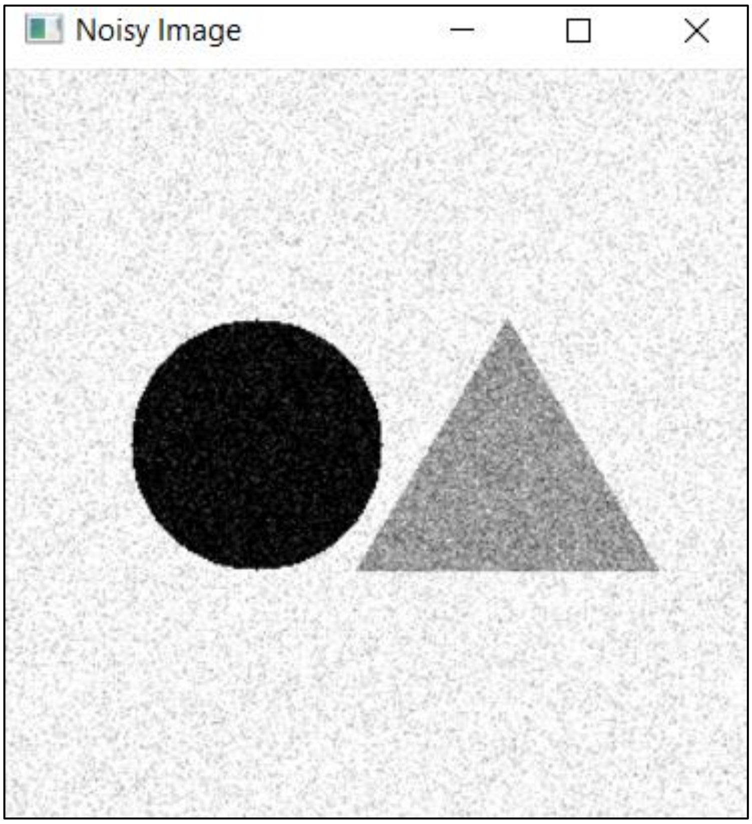

Figure 5. Image after adding the Gaussian noise

4. Main Code setup to add noise, apply Otsu's algorithm, and output the images

Code

```
# Add Gaussian noise to the image
noisy_image = add_gaussian_noise(image)

# Implement Otsu's algorithm for thresholding
otsu_image = otsu_threshold(noisy_image)

# Di (function) imshow: Any , noisy image, and Otsu thresholded image
cv2.imshow('Original Image', image)
cv2.imshow('Noisy Image', noisy_image)
cv2.imshow('Otsu Thresholded Image', otsu_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 6. Main Code setup to add noise, apply Otsu's algorithm, and output the images

Output after Applying Otsu's algorithm for noisy image

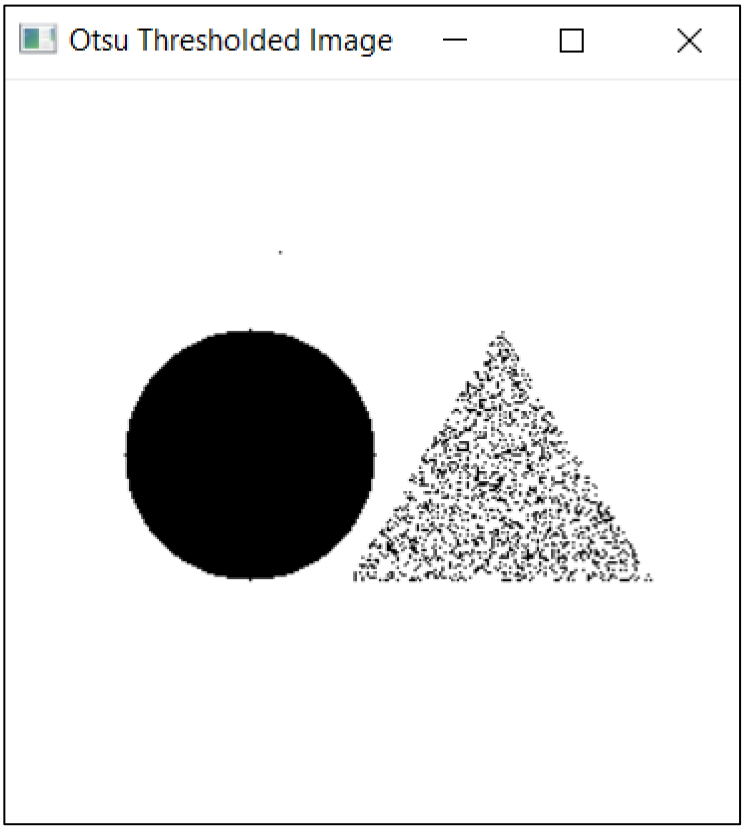

Figure 7. Image after applying Otsu's algorithm for noisy image

Task 2

Implement a region-growing technique for image segmentation. The basic idea is to start from a set of points inside the object of interest (foreground), denoted as seeds, and recursively add neighboring pixels as long as they are in a pre-defined range of the pixel values of the seeds.

1. **Initially checking the intensity levels of input image using a pixel histogram**

<u>Histogram</u>

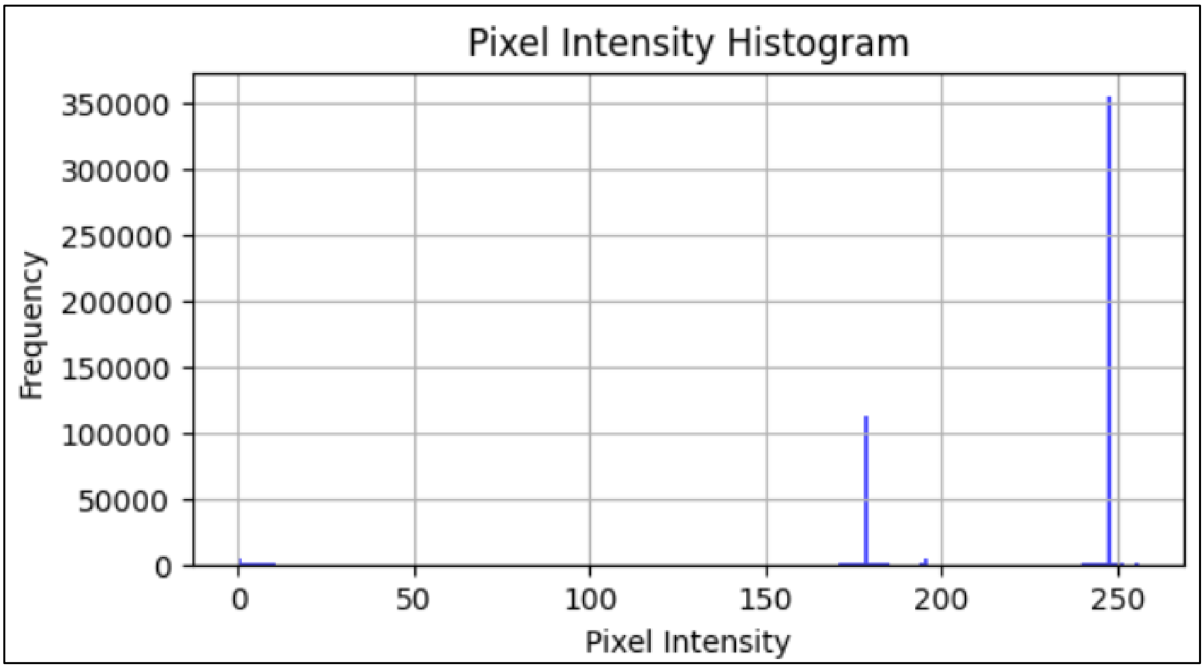

Figure 8. Pixel intensity histogram of the input image

<u>Input Image</u>

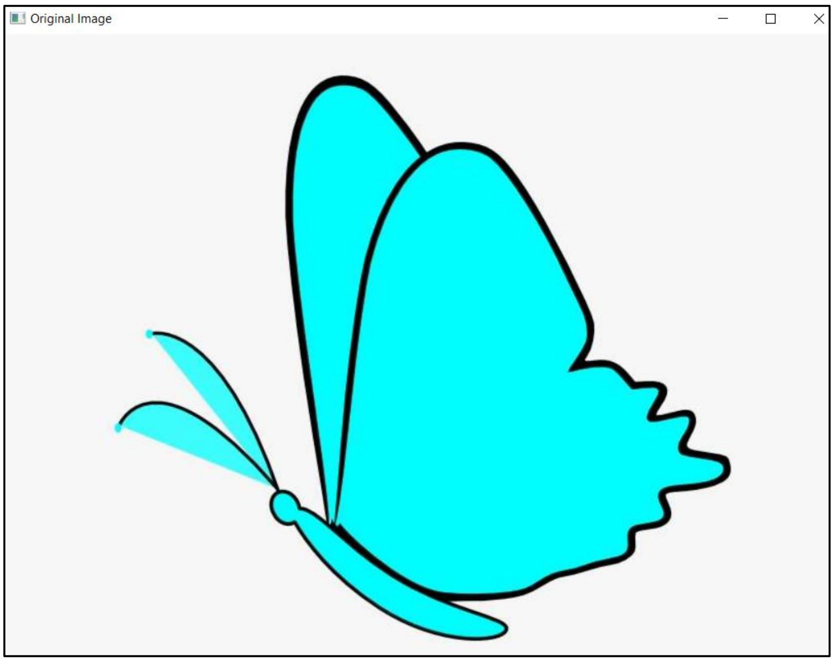

Figure 9. Input Image

2. Defining the Region's growing algorithm

Code

```python
def region_growing(image, seed, threshold):
    # Create an empty region of the same size as the input image
    region = np.zeros_like(image, dtype=np.uint8)
    # Initialize a list to store the seed point
    region_points = [seed]
    # While there are points in the region_points list
    while region_points:
        # Pop the first point from the list
        x, y = region_points.pop(0)
        # Check if the point is not already part of the region and meets the threshold condition
        if region[x, y] == 0 and abs(int(image[x, y]) - int(image[seed])) < threshold:
            # Mark the point as part of the region
            region[x, y] = 255
            # Add neighboring points to the region_points list
            region_points.extend([(x-1, y), (x+1, y), (x, y-1), (x, y+1)])
    return region
```

Figure 10. Code for defining the Region's growing algorithm

3. Importing the image and converting to grayscale

Code

```python
# Load the image
image = cv2.imread('butterfly.png')

# Convert to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Figure 11. Code for importing the image and converting it to grayscale

4. Main code(Defining the seed points, applying the threshold and output the images)

Code

```python
# Define seed point and threshold for region growing
seed_point = (150, 150)
threshold_value = 30

# Apply region growing segmentation
segmented_region = region_growing(gray_image, seed_point, threshold_value)

# Display original and segmented images
cv2.imshow('Original Image', image)
cv2.imshow('Segmented Region', segmented_region)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 12. Main code(Defining the seed points, applying the threshold, and output the images)

Output of the Segmented Image

Figure 13. Output of the image after applying region growing