EE7204 – COMPUTER VISION AND IMAGE PROCESSING

TAKE HOME ASSIGNMENT 2

Name : SADEEPA P.M.A.S

RegNo : EG/2019/3726

Semester : 07

Date : 21/04/2024

GitHub URL: https://github.com/ssadeepa/Computer_Vision_Take_Home_Assignment_2.git

Task 01: 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.

01. Generating an image with 3 pixels, where there are two objects (circle and a square) and the background

Code :

```python
def generateImage(width, height):
    # Creating an empty grayscale image
    image = np.zeros( shape: (height, width), dtype=np.uint8)

    # Setting pixel values for distinct image regions (Background: Black)
    image[:, :] = 0

    # Square (Gray)
    square_size = width // 2
    square_x = 0
    square_y = (height - square_size) // 2
    image[square_y:square_y+square_size, square_x:square_x+square_size] = 128  # color is gray

    # Circle (White)
    circle_radius = width // 5
    circle_center = (width - circle_radius, height // 2)
    cv2.circle(image, circle_center, circle_radius, color: 255, -1)  # color is white

    return image
```

Figure 1 : Image generation using OpenCV

Code Output :

Figure 2 : Generated image with 3-pixel values

2. Defining a Gaussian noise to the image

```
def addGaussianNoise(image):
    # Converting image to floating-point data type
    image_float = image.astype(np.float32)
    # Generate Gaussian noise with specified mean and standard deviation
    # Passing mean and stddev
    noise = np.random.normal( loc: 0,  scale: 50, size=image.shape).astype(np.float32)
    # Adding noise to the image
    img_noised = image_float + noise
    # Ensure that the pixel values are limited to the acceptable range of [0, 255].
    noisy_img = np.clip(img_noised,  a_min: 0,  a_max: 255).astype(np.uint8)
    return noisy_img
```

Figure 3 : Gaussian noise addition to the image

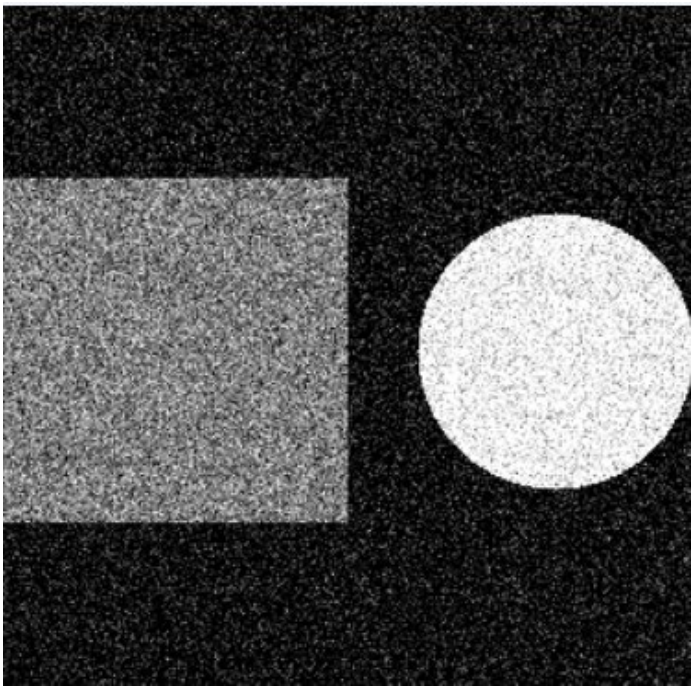

Figure 4 : Noise added image

3. Applying Otsu's Algorithm

Code :

```
# Applying Otsu's thresholding
_, otsuThreshold = cv2.threshold(noisyImage,  thresh: 0,  maxval: 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

Figure 5 : Application of otsu's algorithm for the noisy image

Output code after Applying Otsu’s algorithm for noisy image :

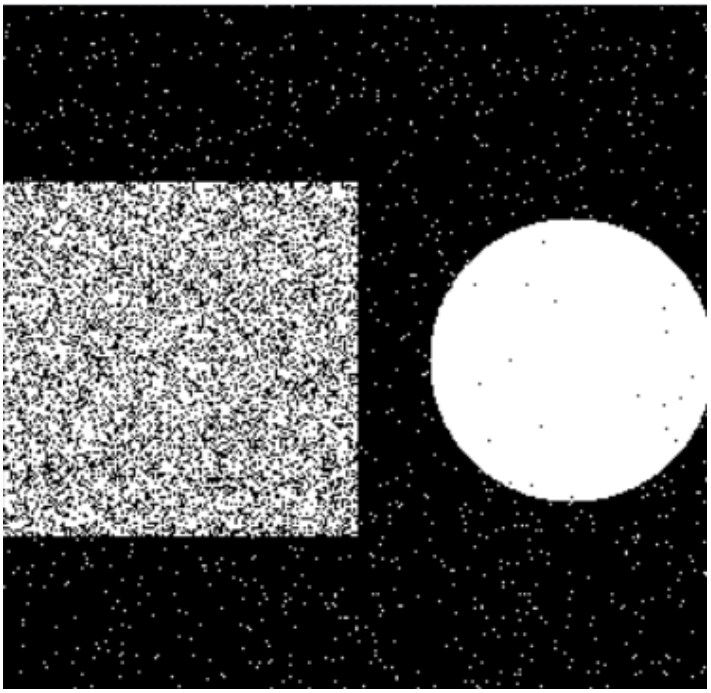

Figure 6 : Otsu's threshold applied image

Task 02 : 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**

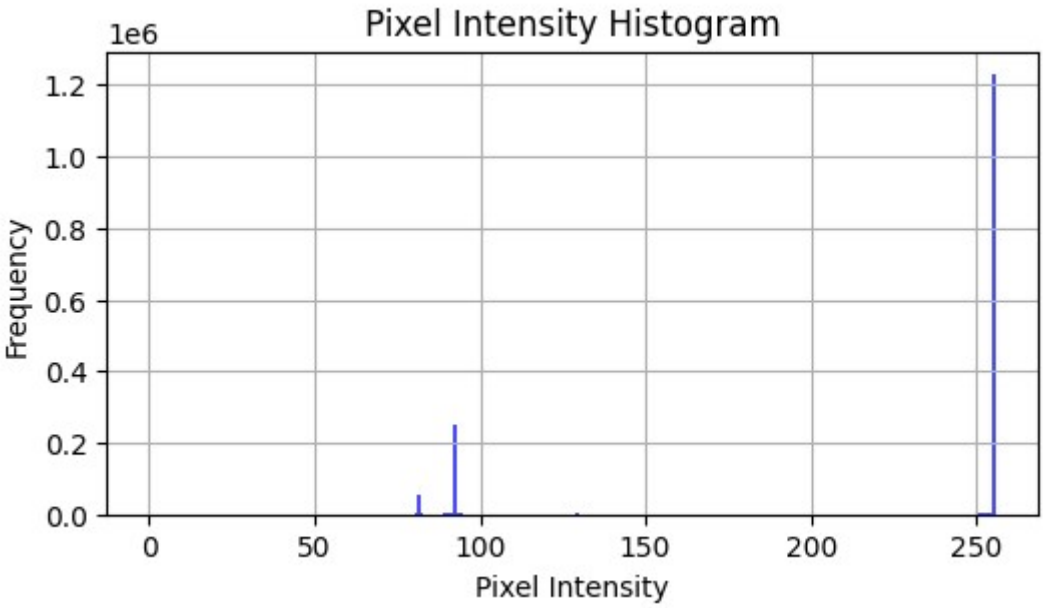

Figure 7 : Pixel Intensity Histogram

Input image :

Figure 8 : Original image

2. Defining the Region's growing algorithm

Code :

```
def region_growing(image, seed, threshold):
    # Create a mask to store the segmented region
    region = np.zeros_like(image, dtype=np.uint8)
    region_points = [seed]
    while region_points:
        x, y = region_points.pop(0)
        if region[x, y] == 0 and abs(int(image[x, y]) - int(image[seed])) < threshold:
            region[x, y] = 255
            region_points.extend([(x-1, y), (x+1, y), (x, y-1), (x, y+1)])
    return region
```

Figure 9. Defining the Region's growing algorithm

3. Importing the image and converting to grayscale

Code :

```
# Loading the image
image = cv2.imread('sample_img.jpg')

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

Figure 10 : Importing the image and converting to grayscale

4. Main code(Defining the seed points and threshold, apply region growing function and output the images)

Code :

```
# Defining seed point and threshold for region growing
seed_point = (50, 50)
threshold_value = 30
# Apply region growing segmentation
segmented_region = region_growing(gray_image, seed_point, threshold_value)

# Display original and segmented images
cv2.namedWindow( winname: 'Original Image', cv2.WINDOW_NORMAL)
cv2.namedWindow( winname: 'Segmented Region', cv2.WINDOW_NORMAL)

cv2.imshow( winname: 'Original Image', image)
cv2.imshow( winname: 'Segmented Region', segmented_region)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 11 : Main Code

Output of the Segmented Image :

Figure 12. Image after applying the region growing algorithm