EE7204: COMPUTER VISION AND IMAGE PROCESSING

TAKE HOME ASSIGNMENT - 02

NAME	:	LAKSHAN D.P.W.K.
REG. NO	:	EG/2019/3649
SEMESTER	:	07
DATE	:	21/04/2024

Contents

1 GitHub Link 4

2 Coding Answers 4

2.1 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. . 4

2.2 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 o the pixel values of the seeds. 6

3 Results 8

3.1 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. . 8

3.2 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 o the pixel values of the seeds. 9

List of Figures

FIGURE 3.1: ORIGINAL IMAGE, GAUSSIAN IMAGES FOR DIFFERENT VARIANCE VALUES AND IMAGE AFTER PERFORMING OTSU'S ALGORITHM 8

FIGURE 3.2: ORIGINAL IMAGE AND THE REGIONAL-GROWN IMAGE 9

1 GitHub Link

https://github.com/KalinduLakshan/EE7204_EG_2019_3649_TakeHomeAssignment_02.git

2 Coding Answers

2.1 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.

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

# Create a blank image with a gray background
image = np.full((550, 550), 128, dtype=np.uint8)

# Define pixel values for the objects
black_pixel = 0
white_pixel = 255

# Draw the original image objects
cv2.rectangle(image, (100, 100), (250, 250), black_pixel, -1)  # Black object
cv2.rectangle(image, (300, 300), (375, 375), white_pixel, -1)  # White object


# Adding Gaussian noise to the original image
def add_gaussian_noise(img, mean, sigma):
    if len(img.shape) == 2:
        h, w = img.shape  # h-height, w-width for grayscale images

        '''
        Generate Gaussian noise with a specified mean (mean) and standard
deviation (sigma)
        using NumPy's random normal function
        '''
        noise = np.random.normal(mean, sigma, (h, w))  # sigma represents the
variance

        '''
        Add the noise to the original image and then clips the resulting
pixel values to be within the range [0, 255].
        This ensures that no pixel values exceed the valid range for uint8
images.
        '''
        noisy_img = np.clip(img + noise, 0, 255).astype(np.uint8)

    else:
        h, w, c = img.shape  # h-height, w-width, c-number of channels used
for RGB images
        noise = np.random.normal(mean, sigma, (h, w, c))
        noisy_img = np.clip(img + noise, 0, 255).astype(np.uint8)
    return noisy_img
```

```python
# Apply Gaussian noise with different parameters
noisy_image_1 = add_gaussian_noise(image, 0, 25)
noisy_image_2 = add_gaussian_noise(image, 0, 50)
noisy_image_3 = add_gaussian_noise(image, 0, 75)
noisy_image_4 = add_gaussian_noise(image, 0, 100)


# Apply Otsu's thresholding algorithm
def otsu_threshold(img):
    _, otsu_threshold_image = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY
+ cv2.THRESH_OTSU)
    return otsu_threshold_image


# Call the otsu_threshold function and assign its value to otsu_image
variable
otsu_image = otsu_threshold(noisy_image_4)

# Plotting the images
fig, axs = plt.subplots(2, 3, figsize=(15, 10))

# Original image
axs[0, 0].imshow(image, cmap='gray')
axs[0, 0].set_title('Original Image')

# Noisy images for different sigma values
axs[0, 1].imshow(noisy_image_1, cmap='gray')
axs[0, 1].set_title('Gaussian Noise: sigma=25')
axs[0, 2].imshow(noisy_image_2, cmap='gray')
axs[0, 2].set_title('Gaussian Noise: sigma=50')
axs[1, 0].imshow(noisy_image_3, cmap='gray')
axs[1, 0].set_title('Gaussian Noise: sigma=75')
axs[1, 1].imshow(noisy_image_4, cmap='gray')
axs[1, 1].set_title('Gaussian Noise: sigma=100')

# Otsu's image
axs[1, 2].imshow(otsu_image, cmap='gray')
axs[1, 2].set_title("Otsu's Algorithm")

# Plot all the images
plt.show()
```

2.2 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.

```
# Import the necessary libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt


def get_eight_neighbour(x, y, shape):
    # Returns the coordinates of the 8-neighbors of a point within a given
shape
    output = []
    max_x = shape[1] - 1
    max_y = shape[0] - 1

    # Define the eight possible neighbors and constrain them within the image
boundaries.
    neighbors = [
        (x - 1, y - 1), (x, y - 1), (x + 1, y - 1),  # Top row
        (x - 1, y),                 (x + 1, y),       # Middle row (left and
right)
        (x - 1, y + 1), (x, y + 1), (x + 1, y + 1)  # Bottom row
    ]

    for nx, ny in neighbors:
        # Constrain the coordinates within the image boundaries
        output_x = min(max(nx, 0), max_x)
        output_y = min(max(ny, 0), max_y)
        output.append((output_x, output_y))

    return output


def region_growing(im, seed):
    # Performs region growing from a given seed point
    seed_points = [seed]  # Start with the initial seed point
    output_img = np.zeros_like(im)  # Output image initialized to zeros
(black)
    processed = set()  # A set to track processed points

    while seed_points:
        pix = seed_points.pop(0)  # Get the next seed point to process
        if pix in processed:
            continue  # If already processed, skip

        output_img[pix[0], pix[1]] = 255  # Mark the point as white in the
output image
        processed.add(pix)  # Add to the set of processed points
```

```python
        # Add neighbors to the seed points if they haven't been processed
        for coord in get_eight_neighbour(pix[0], pix[1], im.shape):
            if im[coord[0], coord[1]] == 255 and coord not in processed:
                seed_points.append(coord)

    return output_img


# Define the predefined seed point
predefined_seed = (100, 10)  # (y, x) coordinate for the seed point

# Load the image and convert to grayscale
image = cv2.imread('apple.jpg', 0)

# Apply a binary threshold to create a binary image
ret, img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)

# Use the predefined seed point for region growing
rg_out = region_growing(img, predefined_seed)

# Create subplots to display the original and the region-grown image
plt.figure(figsize=(10, 5))  # Set the figure size

# Original image subplot
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.imshow(image, cmap='gray')  # Display the original image in grayscale
plt.title('Original Image')
plt.axis('off')  # Hide axis

# Region growing result subplot
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
plt.imshow(rg_out, cmap='gray')  # Display the region-grown image in
grayscale
plt.title('Region-Grown Image')
plt.axis('off')  # Hide axis

# Show the plot
plt.show()
```

3 Results

3.1 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.

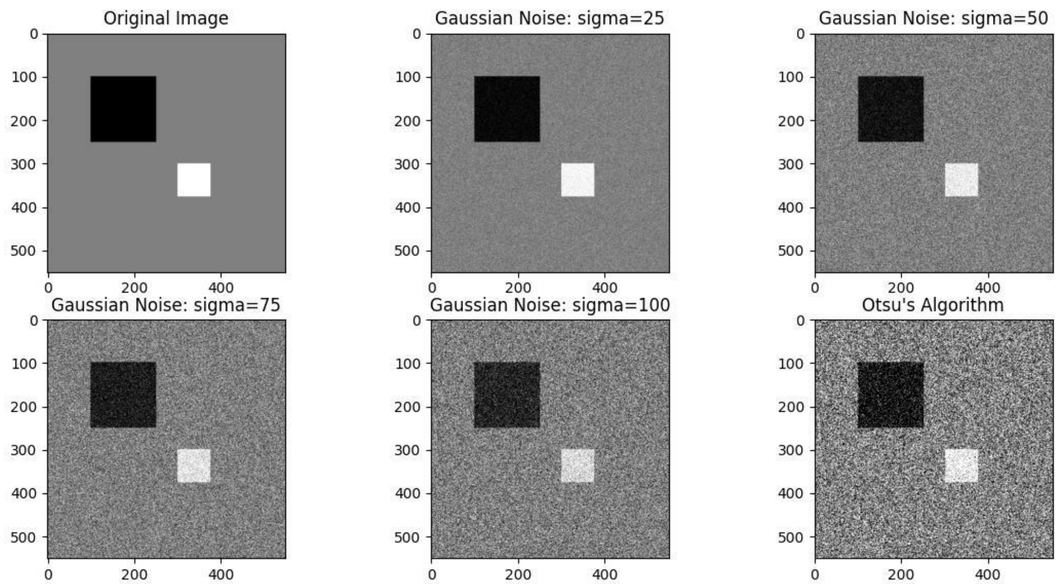

FIGURE 3.1: ORIGINAL IMAGE, GAUSSIAN IMAGES FOR DIFFERENT VARIANCE VALUES AND IMAGE AFTER PERFORMING OTSU'S ALGORITHM

3.2 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.

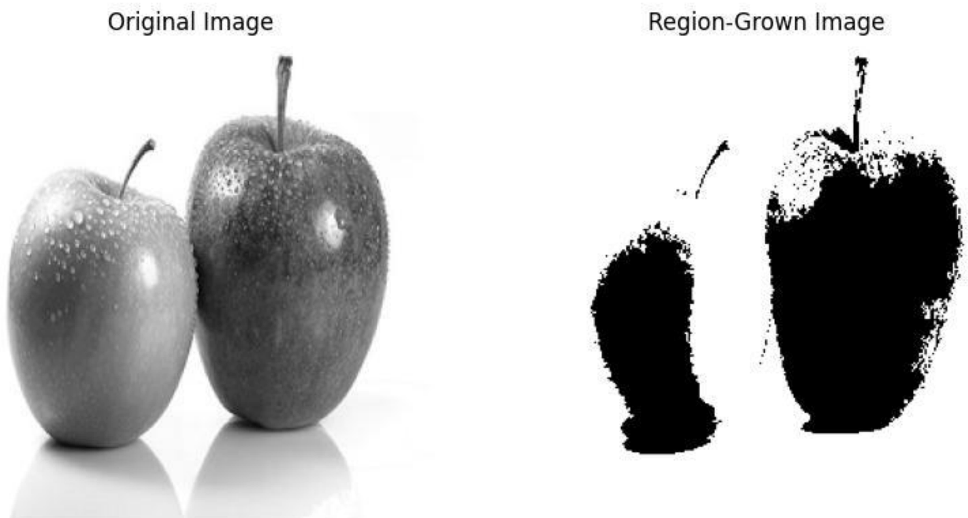

FIGURE 3.2: ORIGINAL IMAGE AND THE REGIONAL-GROWN IMAGE