**Exercise-7**

**Task 1: To implement basic Morphological operations on images such as erosion, dilation, opening and closing.**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('alpha.jpg', 0)

img2=cv2.imread('noise.png',0)

kernel = np.ones((5,5), np.uint8)

binr = cv2.threshold(image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]

invert = cv2.bitwise\_not(binr)

erosion = cv2.erode(invert, kernel, iterations = 1)

dilation = cv2.dilate(invert, kernel, iterations = 1)

binr1 = cv2.threshold(img2, 0, 255,

                     cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]

opening = cv2.morphologyEx(binr1, cv2.MORPH\_OPEN, kernel)

closing = cv2.morphologyEx(binr1, cv2.MORPH\_CLOSE, kernel)

titles = ['Original Image', 'Erosion', 'Dilation', 'Opening', 'Closing']

images = [invert, erosion, dilation, opening, closing]

for i in range(5):

    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray')

    plt.title(titles[i])

    plt.xticks([]), plt.yticks([])

plt.show()

**Task 2: To implement region-based image segmentation techniques such as Region Growing, and Region Splitting and Merging.**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def region\_growing(img, seed, threshold):

    h, w = img.shape

    output = np.zeros\_like(img)

    pixel\_list = [seed]

    while pixel\_list:

        x, y = pixel\_list.pop(0)

        if abs(int(img[x, y]) - int(img[seed])) < threshold and output[x, y] == 0:

            output[x, y] = 255

            # Check neighbors (4-connected)

            if x > 0: pixel\_list.append((x-1, y))

            if x < h-1: pixel\_list.append((x+1, y))

            if y > 0: pixel\_list.append((x, y-1))

            if y < w-1: pixel\_list.append((x, y+1))

    return output

image = cv2.imread('brain.jpeg', 0)

binr = cv2.threshold(image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]

invert = cv2.bitwise\_not(binr)

seed\_point = (130, 100)

threshold = 10

seed\_image = invert.copy()

cv2.circle(seed\_image, seed\_point, 5, (255, 0, 0), -1)  # Highlight the seed point with a circle

result = region\_growing(invert, seed\_point, threshold)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

plt.imshow(image, cmap='gray')

plt.title('Original Image')

plt.xticks([]), plt.yticks([])

plt.subplot(1, 3, 2)

plt.imshow(seed\_image, cmap='gray')

plt.title('Seed Image (with Seed Point)')

plt.xticks([]), plt.yticks([])

plt.subplot(1, 3, 3)

plt.imshow(result, cmap='gray')

plt.title('Region Growing Result')

plt.xticks([]), plt.yticks([])

plt.show()

-------------------🡪

import cv2

import numpy as np

import matplotlib.pyplot as plt

def split(image, thresh):

    h, w = image.shape

    regions = [(0, h, 0, w)]

    def is\_homogeneous(region):

        x1, x2, y1, y2 = region

        region\_intensity = image[x1:x2, y1:y2]

        return np.std(region\_intensity) < thresh

    def recursive\_split(region):

        if is\_homogeneous(region):

            return [region]

        else:

            x1, x2, y1, y2 = region

            mid\_x = (x1 + x2) // 2

            mid\_y = (y1 + y2) // 2

            return (recursive\_split((x1, mid\_x, y1, mid\_y)) +  # Top-left

                    recursive\_split((mid\_x, x2, y1, mid\_y)) +  # Bottom-left

                    recursive\_split((x1, mid\_x, mid\_y, y2)) +  # Top-right

                    recursive\_split((mid\_x, x2, mid\_y, y2)))   # Bottom-right

    all\_regions = []

    for region in regions:

        all\_regions.extend(recursive\_split(region))

    return all\_regions

def merge\_regions(regions):

    merged\_image = np.zeros\_like(image)

    for region in regions:

        x1, x2, y1, y2 = region

        merged\_image[x1:x2, y1:y2] = np.mean(image[x1:x2, y1:y2])

    return merged\_image

image = cv2.imread('kP0u2.png', 0)

regions = split(image, thresh=20)

merged\_image = merge\_regions(regions)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.imshow(image, cmap='gray')

plt.title('Original Image')

plt.xticks([]), plt.yticks([])

plt.subplot(1, 2, 2)

plt.imshow(merged\_image, cmap='gray')

plt.title('Splitting and Merging Result')

plt.xticks([]), plt.yticks([])

plt.show()

**Task 3: To perform image segmentation using Watershed Segmentation algorithm.**

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('tree.jpeg')

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

\_, binary = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

kernel = np.ones((3, 3), np.uint8)

opening = cv2.morphologyEx(binary, cv2.MORPH\_OPEN, kernel, iterations=2)

sure\_bg = cv2.dilate(opening, kernel, iterations=3)

dist\_transform = cv2.distanceTransform(opening, cv2.DIST\_L2, 5)

\_, sure\_fg = cv2.threshold(dist\_transform, 0.7 \* dist\_transform.max(), 255, 0)

sure\_fg = np.uint8(sure\_fg)

unknown = cv2.subtract(sure\_bg, sure\_fg)

\_, markers = cv2.connectedComponents(sure\_fg)

markers = markers + 1

markers[unknown == 255] = 0

markers = cv2.watershed(image, markers)

image[markers == -1] = [255, 0, 0]

plt.subplot(1, 2, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.title('Segmented Image'), plt.xticks([]), plt.yticks([])

plt.subplot(1, 2, 2), plt.imshow(markers, cmap='gray')

plt.title('Watershed Markers'), plt.xticks([]), plt.yticks([])

plt.show()