**Assignment 6 (50)**

**Goutham Selvakumar – 2092286**

**Mathematical Morphology.**

**A MatLab reference:** <https://www.mathworks.com/help/images/morphological-filtering.html>

**A Python tutorial:** <https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html>

**Problem 1: Erosion and Dilation (20)**

**Choose a gray scale image (or convert a color image to grayscale). Convert to binary based on a mean-intensity threshold. Apply erosion twice and then dilation twice with a structuring element of your choice. (Defaults are fine.) Then, starting with the original binary image, dilate twice and erode twice. Show the two resulting images and contrast them.**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from skimage import filters, color

img = cv2.imread('C:/Users/admin/Desktop/abcd.jpg')

color\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

gray\_img = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)

**#Plot images**

plt.clf()

fig = plt.figure()

fig.set\_size\_inches(20, 20)

ax1 = fig.add\_subplot(1,2,1)

ax1.imshow(color\_img)

ax1.set\_title('Color Image')

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

ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(gray\_img, cmap = 'gray')

ax2.set\_title('Grey Image')

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

plt.show()



**#Convert to binary**

thresh = np.mean(gray\_img)

**#Binary image**

binary\_img = cv2.threshold(gray\_img, thresh, 255, cv2.THRESH\_BINARY)[1]

**#Plot image**

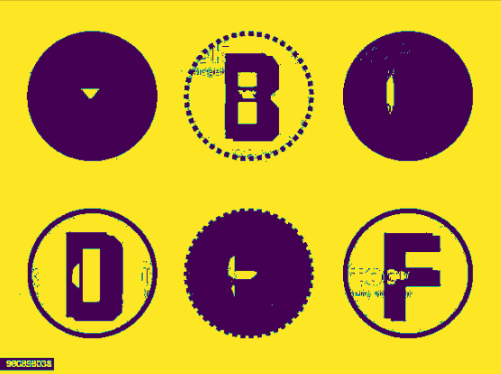
fig = plt.figure()

plt.imshow(binary\_img)

fig.set\_size\_inches(10, 10)

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

plt.show()



**#Erode > Dialate binary\_image**

**#Define kernel**

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

**#Erode image**

eroded\_dial\_img = cv2.erode(binary\_img, kernel,iterations = 2)

**#Dilate image**

eroded\_dial\_img = cv2.dilate(eroded\_dial\_img, kernel, iterations = 2)

**#Define kernel**

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

**#Dilate image**

dialated\_erode\_img = cv2.dilate(binary\_img, kernel, iterations = 2)

**#Erode image**

dialated\_erode\_img = cv2.erode(dialated\_erode\_img, kernel,iterations = 2)

**#Plot images**

plt.clf()

fig = plt.figure()

fig.set\_size\_inches(20, 20)

ax1 = fig.add\_subplot(1,2,1)

ax1.imshow(eroded\_dial\_img,cmap='gray')

ax1.set\_title('Erosion 1st Dilate 2nd')

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

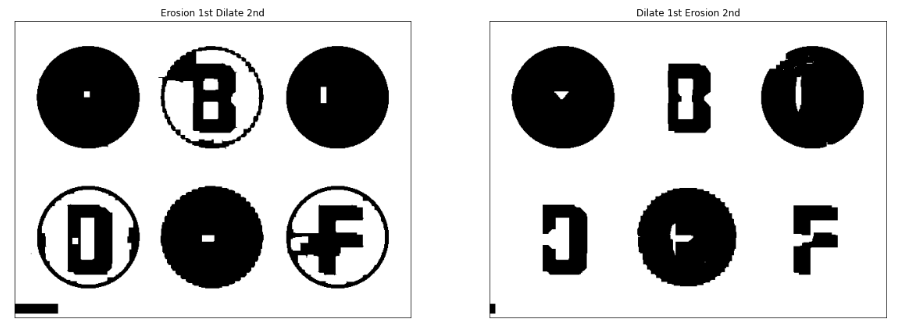
ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(dialated\_erode\_img,cmap='gray')

ax2.set\_title('Dilate 1st Erosion 2nd')

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

plt.show()



* Erosion primarily involves eroding the outer surface of the image. As binary images only contain 0-255, it primarily involves eroding the foreground of the image and it is suggested to have the foreground as white.
* Dilation involves dilating the outer surface of the image. As binary images only contain 0-255, it primarily involves expanding the foreground of the image and it is suggested to have the foreground as white.
* In contrasting between erosion and dilation with the original image in the middle, we can see the difference in there layers. The erosion removed the edges and details of the foreground with an iteration of 2 times, whereas the dilation kept the details by increasing the intensity same with an iteration of 2 times.

**Problem 2: Opening and Closing (20)**

**Using the binary image from Problem 1, apply opening twice and then closing twice with a structuring element of your choice. (Defaults are fine.) Then, starting with the original binary image, close twice and open twice. Show the two resulting images and contrast them. Then, compare how your two images from Problem 2 visually differ from the two images of Problem 1.**

**#Define kernel**

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

**#Opening image x2**

opening\_closed\_img = cv2.morphologyEx(binary\_img, cv2.MORPH\_OPEN, kernel)

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

**#Dilate image x2**

opening\_closed\_img = cv2.morphologyEx(opening\_closed\_img, cv2.MORPH\_CLOSE, kernel)

opening\_closed\_img = cv2.morphologyEx(opening\_closed\_img, cv2.MORPH\_CLOSE, kernel)

**#Closing > Opening binary\_image**

**#Define kernel**

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

**#Closing image x2**

closing\_openned\_img = cv2.morphologyEx(binary\_img, cv2.MORPH\_CLOSE, kernel)

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

**#Opening image x2**

closing\_openned\_img = cv2.morphologyEx(closing\_openned\_img, cv2.MORPH\_OPEN, kernel)

closing\_openned\_img = cv2.morphologyEx(closing\_openned\_img, cv2.MORPH\_OPEN, kernel)

**# Plot images**

plt.clf()

fig = plt.figure()

fig.set\_size\_inches(20, 20)

ax1 = fig.add\_subplot(1,2,1)

ax1.imshow(opening\_closed\_img,cmap='gray')

ax1.set\_title('Open 1st Close 2nd')

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

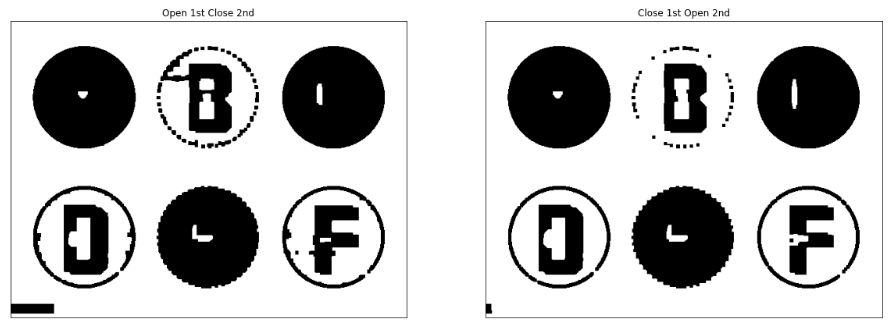
ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(closing\_openned\_img,cmap='gray')

ax2.set\_title('Close 1st Open 2nd')

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

plt.show()



generalized\_kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (15, 15))

**#Opening image x2**

opening\_closed\_img = cv2.morphologyEx(binary\_img, cv2.MORPH\_OPEN, generalized\_kernel)

opening\_closed\_img = cv2.morphologyEx(opening\_closed\_img, cv2.MORPH\_OPEN, generalized\_kernel)

**#Dilate image x2**

opening\_closed\_img = cv2.morphologyEx(opening\_closed\_img, cv2.MORPH\_CLOSE, generalized\_kernel)

opening\_closed\_img = cv2.morphologyEx(opening\_closed\_img, cv2.MORPH\_CLOSE, generalized\_kernel)

**#Closing image x2**

closing\_openned\_img = cv2.morphologyEx(binary\_img, cv2.MORPH\_CLOSE, generalized\_kernel)

closing\_openned\_img = cv2.morphologyEx(closing\_openned\_img, cv2.MORPH\_CLOSE, generalized\_kernel)

**#Opening image x2**

closing\_openned\_img = cv2.morphologyEx(closing\_openned\_img, cv2.MORPH\_OPEN, generalized\_kernel)

closing\_openned\_img = cv2.morphologyEx(closing\_openned\_img, cv2.MORPH\_OPEN, generalized\_kernel)

**# Plot images**

plt.clf()

fig = plt.figure()

fig.set\_size\_inches(20, 20)

ax1 = fig.add\_subplot(1,2,1)

ax1.imshow(opening\_closed\_img,cmap='gray')

ax1.set\_title('Open 1st Close 2nd')

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

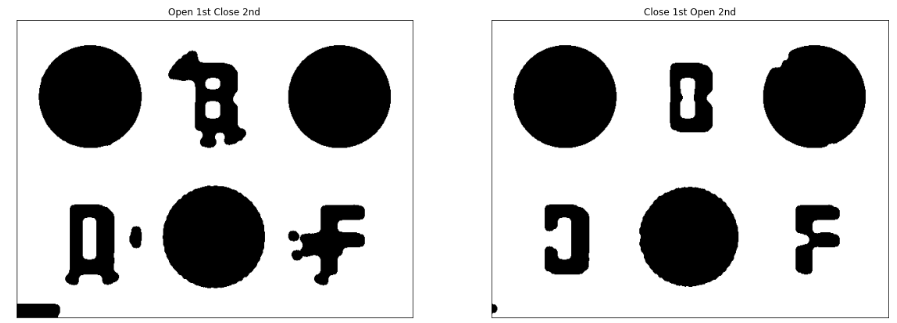
ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(closing\_openned\_img,cmap='gray')

ax2.set\_title('Close 1st Open 2nd')

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

plt.show()



* Opening involves erosion followed by dilation in the outer surface of the image and is generally used for removing the noise in the image.
* Closing involves dilation followed by erosion in the outer surface of the image and is also used for removing the noise in the image.
* So, in comparing the opening and closing of the image we can notice difference in the foreground of the image. Opening has higher detail of the image since the dilation is done at the last whereas in the Closing method we can see detail missing since the eroding is done at the last.
* In comparison of the Problem 1, erosion and dilation method, we can notice the background being kept as it is, and since the process in problem 2 is opening and closing (combining erosion and dilation [vice-versa for closing]), the difference is of high as the wordings are getting noticeable in Problem 2.

**Problem 3: Boundary Extraction (10)**

**Boundary extraction of an image I, β(I), can be performed by eroding I with a structuring element B and then subtracting the eroded image from I:**

**β(I) = I – (I ɵ B)**

**Using your binary image from the two problems above, extract the boundaries. Then, extract Canny edges from the original grayscale image. Show the two images and compare them.**

import cv2

**#Generate structured kernel**

struct\_kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (5, 5))

**#Erode binary image**

eroded\_img = cv2.erode(binr, struct\_kernel, iterations = 1)

**#I - B(I)**

eroded\_edges = binr - eroded\_img

**#Canny edges**

canny\_edges = cv2.Canny(img, 100, 200)

**#Plot images**

plt.clf()

fig = plt.figure()

fig.set\_size\_inches(20, 20)

ax1 = fig.add\_subplot(1,2,1)

ax1.imshow(eroded\_edges)

ax1.set\_title('Boundary Extraction')

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

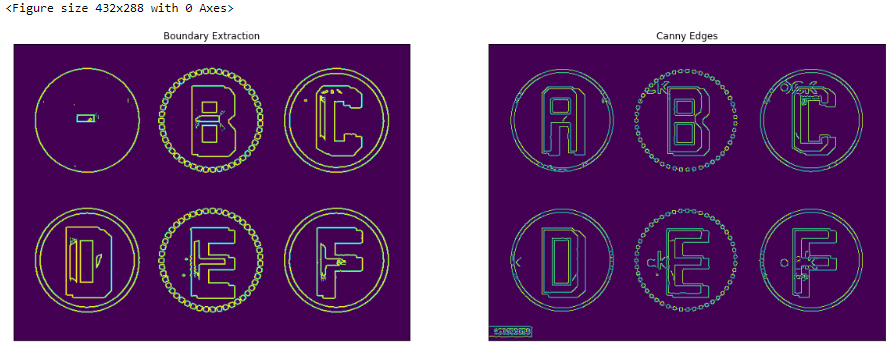
ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(canny\_edges)

ax2.set\_title('Canny Edges')

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

plt.show()



We can see the Boundary Extraction and Canny Edge image, in which the boundary extraction has reduced details in it because if A is an image and structuring element is B then Boundary Extraction can be given as, Boundary **β(I) = I – (I ɵ B)**. It means subtracting the erode image of A from the original image. Moreover, we can see that the Canny Edge on the second one has higher details and edges present in them.