**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

**Subject Name**

**Digital Image Processing**

**Lab Mid**

**SUBMITTED TO:**

**Dr. Asad Khan**

**LE Sundas Ashraf**

**SUBMITTED BY:**

**Student Name**

1. Wahaaj Nasir

**Reg#413238**

**DE- 44 Dept C&SE**

**Objectives:**

Filtering on Images and Histogram Calculations

**Related Topic/Chapter in theory class:**

Spatial Filtering

**Hardware/Software required:**

Hardware: PC

Software Tool: Pycharm

**Task 1:**

**Solution:**

import numpy as np  
import cv2 as cv  
import matplotlib.pyplot as plt  
from pandas.compat.numpy.function import validate\_argsort\_kind  
  
  
def padding(pad, orig):  
 rows, cols = orig.shape  
 padded\_arr = np.ones((rows+ 2 \* pad, cols+ 2 \* pad), dtype = np.uint8)\*0  
  
 for i in range(rows):  
 for j in range(cols):  
 padded\_arr[i+pad][j+pad] = orig[i][j]  
  
 return padded\_arr  
  
def remove\_padding(padded\_img, pad):  
 rows, cols = padded\_img.shape  
 return padded\_img[pad:rows-pad, pad:cols-pad]  
  
def lower\_by\_x(image, thresh):  
 rows, cols = image.shape  
 new\_image = np.ones((rows, cols), dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if (image[i, j] >= 0 and image[i, j] <= thresh):  
 new\_image[i, j] = 0  
 elif (image[i, j] >= thresh+1 and image[i, j] <= 255):  
 new\_image[i, j] = 255  
  
 return new\_image  
  
def lower\_by\_2(image):  
 rows, cols = image.shape  
 new\_image = np.ones((rows, cols), dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if (image[i, j] >= 0 and image[i, j] <= 127):  
 new\_image[i, j] = 0  
 elif (image[i, j] >= 128 and image[i, j] <= 255):  
 new\_image[i, j] = 255  
  
 return new\_image  
  
def cc(orig, lower\_bound, upper\_bound):  
 rows, cols = orig.shape  
 new\_img = np.zeros((rows, cols), dtype=np.int32)  
 my\_dict = {}  
 count = 1  
  
 for i in range(1, rows):  
 for j in range(1, cols):  
 if ((orig[i][j] >= lower\_bound) & (orig[i][j] <= upper\_bound)) :  
 neighbors = [] # Store nonzero neighboring labels  
  
 # Check all 8-connected neighbors  
 if ((orig[i - 1][j] >= lower\_bound) & (orig[i - 1][j] <= upper\_bound)):  
 neighbors.append(new\_img[i - 1][j])  
 if ((orig[i][j-1] >= lower\_bound) & (orig[i][j-1] <= upper\_bound)):  
 neighbors.append(new\_img[i][j - 1])  
 if ((orig[i-1][j-1] >= lower\_bound) & (orig[i-1][j-1] <= upper\_bound)):  
 neighbors.append(new\_img[i - 1][j - 1])  
 if ((j + 1 < cols) and (lower\_bound <= orig[i - 1][j + 1] <= upper\_bound)):  
 neighbors.append(new\_img[i - 1][j + 1])  
  
 if not neighbors: # No connected neighbors, assign new label  
 new\_img[i][j] = count  
 my\_dict[count] = count  
 count += 1  
 else:  
 min\_label = min(neighbors)  
 new\_img[i][j] = min\_label  
  
 # Merge equivalence classes  
 for label in neighbors:  
 root1 = find\_root(my\_dict, min\_label)  
 root2 = find\_root(my\_dict, label)  
 if root1 != root2:  
 my\_dict[max(root1, root2)] = min(root1, root2)  
  
 for i in range(1, rows):  
 for j in range(1, cols):  
 if new\_img[i][j] > 0:  
 new\_img[i][j] = find\_root(my\_dict, new\_img[i][j])  
  
 return new\_img, my\_dict  
  
  
# Path compression to find root label  
def find\_root(my\_dict, x):  
 #Added to avoid that the background coming in the dictionaries  
 if x == 0:  
 return 0  
 if x not in my\_dict:  
 my\_dict[x] = x  
 return x  
 while my\_dict[x] != x:  
 my\_dict[x] = my\_dict[my\_dict[x]] # Path compression  
 x = my\_dict[x]  
 return x  
  
def histogram\_creating(image):  
 rows, cols = image.shape  
 histogram = np.zeros(256, dtype = int)  
  
 for i in range(rows):  
 for j in range(cols):  
 val = image[i][j]  
 histogram[val] += 1  
  
 return histogram  
  
def hist\_cumsum(histogram):  
 cumsum = np.zeros(len(histogram), dtype = int)  
 cumsum[0] = histogram[0]  
 for i in range(1, len(histogram)):  
 cumsum[i] = cumsum[i-1] + histogram[i]  
  
 return cumsum  
  
  
  
def merge\_for\_mask(cyto\_img, nuclei\_img):  
 rows, cols = cyto\_img.shape  
 mask = np.zeros((rows, cols), dtype = np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if((cyto\_img[i][j] == 128) & (nuclei\_img[i][j] != 255)):  
 mask[i][j] = 128  
 elif((cyto\_img[i][j] == 128) & (nuclei\_img[i][j] == 255)):  
 mask[i][j] = 255  
 elif((cyto\_img[i][j] != 128) & (nuclei\_img[i][j] == 255)):  
 mask[i][j] = 0 #Reduce false positives of nucleus as nuclei should only be inside cyto  
  
 return mask  
  
#D.C = 2 \* (X ∩ Y) / X + Y  
#X is Predicted Pixels  
#Y is Actual Pixels  
#X ∩ Y is true Positives  
def calculate\_dice\_coefficient(true\_mask, own\_mask, label):  
 rows, cols = true\_mask.shape  
 X = 0  
 Y = 0  
 TP = 0  
  
 for i in range(rows):  
 for j in range(cols):  
 if(own\_mask[i][j] == label):  
 X += 1  
  
 for i in range(rows):  
 for j in range(cols):  
 if (true\_mask[i][j] == label):  
 Y += 1  
  
 for i in range(rows):  
 for j in range(cols):  
 if ((own\_mask[i][j] == label) & (true\_mask[i][j] == label)):  
 TP += 1  
  
 DC = (2 \* TP) / (X+Y)  
  
 return DC  
  
#Purely for checking purposes  
def neg\_img(image):  
 l = 256  
 rows, cols = image.shape  
 new\_img = np.zeros((rows, cols), dtype = np.uint8)  
 for i in range(rows):  
 for j in range(cols):  
 r = int(image[i][j])  
 s = (256-1)-r  
 new\_img[i][j] = np.uint8(s)  
  
 return new\_img  
  
def contrast\_stretch(image):  
 im\_min\_5 = np.percentile(image, 5)  
 im\_max\_95 = np.percentile(image, 95)  
 rows,cols = image.shape  
 new\_img = np.zeros((rows, cols), dtype = np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 if(image[i][j] < im\_min\_5):  
 new\_img[i][j] = 0  
 elif(image[i][j] > im\_max\_95):  
 new\_img[i][j] = 255  
 else:  
 new\_img[i][j] = 255 \* ((image[i][j] - im\_min\_5) / (im\_max\_95 - im\_min\_5))  
  
 return new\_img  
  
def transformation\_fun(cum\_pdf):  
 return np.uint8((cum\_pdf\*255))  
  
def apply\_trans(image, trans\_fun):  
 rows, cols = image.shape  
 new\_img = np.zeros((rows, cols), dtype = np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 new\_img[i][j] = trans\_fun[image[i][j]]  
  
 return new\_img  
  
def count\_num(image, dict):  
 rows, cols = image.shape  
 count = 1  
 max\_val = 0  
  
 new\_img = np.zeros((rows, cols), dtype = np.uint8)  
  
 value\_arr = np.zeros(max\_val, dtype= np.int32)  
 for i in range(rows):  
 for j in range(cols):  
 val = image[i][j]  
 value\_arr[val] = value\_arr[val]+1  
  
 for i in range(len(value\_arr)):  
 val = value\_arr[i]  
 if val > 65:  
 if val >= 200:  
 for i in range(len(value\_arr)):  
 if(i == val):  
 temp = i  
 break  
 dict[temp] = 255  
 else:  
 for i in range(len(value\_arr)):  
 if(i == val):  
 temp = i  
 break  
 dict[temp] = 127  
 else:  
 dict[i] = 0  
  
   
  
  
  
  
  
# Main  
image = cv.imread(r"D:\Uni\Semester 6\DIP\Self\Lab\Lab Mid\Open Lab with Instructions\dataset\images\render0358.png", 0)  
  
image = contrast\_stretch(image)  
histogram = histogram\_creating(image)  
cumsum = hist\_cumsum(histogram)  
cdf = cumsum/max(cumsum)  
thresh\_rocks = (np.where(cdf >= 0.9)[0][0])  
lowered\_x = lower\_by\_x(image, thresh\_rocks)  
  
image\_padded = padding(1, lowered\_x)  
img\_cc\_rocks, img\_cc\_rocks\_dict = cc(image\_padded, 255, 255)  
count\_num(img\_cc\_rocks, img\_cc\_rocks\_dict)  
  
cv.imshow("Original Image", image)  
cv.imshow("Lowered", lowered\_x)  
cv.waitKey()

**Output:**

**No output obtained as time finished.  
Code will give error.  
Runs till calculation of what small and big rocks should be, but was not able to update dictionary for images according to those values. Dictionary indexes have been calculated for small and big rocks, but not updated in image.**