**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

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

**Subject Name**

**Digital Image Processing**

**Lab Number**

**6**

**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

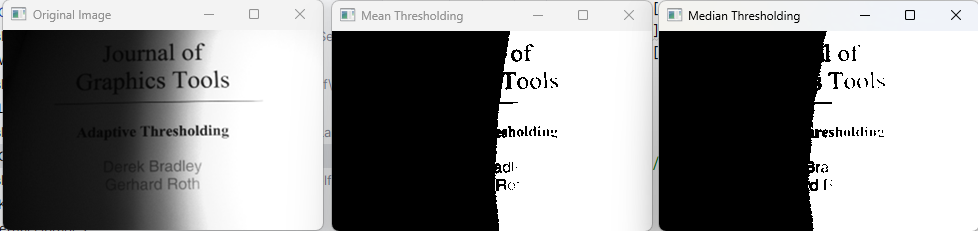
**Task 1:**

**Write a program that thresholds the provided image (Threshold\_Image.png) using global mean and median.**

**Solution:**

import numpy as np  
import cv2 as cv  
  
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  
  
*#Main*image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lab/Lab 6/Lab 6/Threshold\_Image.png", 0)  
mean = np.mean(image)  
median = np.median(image)  
  
mean\_thresh = lower\_by\_x(image, mean)  
median\_thresh = lower\_by\_x(image, median)  
  
cv.imshow("Original Image", image)  
cv.imshow("Mean Thresholding", mean\_thresh)  
cv.imshow("Median Thresholding", median\_thresh)  
cv.waitKey()

**Output:**



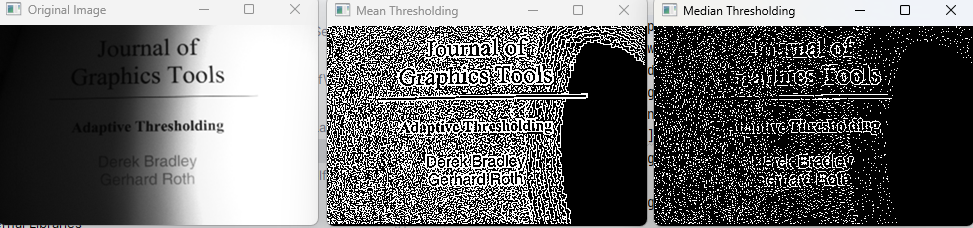
**Task 2:**

**Now threshold the same image by taking threshold value mean and median of 3x3 block locally.**

**Solution**

import cv2 as cv  
import numpy as np  
  
def padding(pad, orig):  
 rows, cols = orig.shape  
 padded\_arr = np.ones((rows+ 2 \* pad, cols+ 2 \* pad), dtype = np.uint8)\*255  
  
 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 local\_thresh\_mean(image, filter\_size):  
 pad = filter\_size//2  
 rows, cols = image.shape  
 filtered\_img = np.zeros((rows, cols), dtype = np.uint8)  
  
 padded\_img = padding(pad, filtered\_img)  
 for i in range(pad, rows-pad):  
 for j in range(pad, cols-pad):  
 sub\_img = image[i-pad:i+pad+1, j-pad:j+pad+1]  
 mean\_val = np.mean(sub\_img)  
 if(image[i][j] <= mean\_val):  
 padded\_img[i][j] = 0  
 else:  
 padded\_img[i][j] = 255  
  
 filtered\_img = remove\_padding(padded\_img, pad)  
  
 return filtered\_img  
  
def local\_thresh\_median(image, filter\_size):  
 pad = filter\_size//2  
 rows, cols = image.shape  
 filtered\_img = np.zeros((rows, cols), dtype = np.uint8)  
  
 padded\_img = padding(pad, filtered\_img)  
 for i in range(pad,rows-pad):  
 for j in range(pad, cols-pad):  
 sub\_img = image[i-pad:i+pad+1, j-pad:j+pad+1]  
 median\_val = np.median(sub\_img)  
 if(image[i][j] <= median\_val):  
 padded\_img[i][j] = 0  
 else:  
 padded\_img[i][j] = 255  
  
 filtered\_img = remove\_padding(padded\_img, pad)  
  
 return filtered\_img  
  
*#Main*image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lab/Lab 6/Lab 6/Threshold\_Image.png", 0)  
mean\_img = local\_thresh\_mean(image, 3)  
median\_img = local\_thresh\_median(image, 3)  
  
cv.imshow("Original Image", image)  
cv.imshow("Mean Thresholding", mean\_img)  
cv.imshow("Median Thresholding", median\_img)  
cv.waitKey()

**Output:**



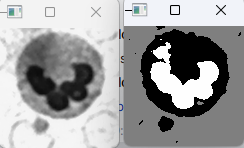
**Task 3:**

**Write a function that takes hematological image as input and segments the image using K mean clustering with and k=2. Write the function such that the k value can be passed as a parameter. Display the resulting image**

**Solution**

import cv2 as cv  
import numpy as np  
import math  
  
  
def initial\_centroids(image, k):  
 unique\_values = np.unique(image)  
  
 init\_centroids = np.random.choice(unique\_values, k, replace=False)  
 return np.array(init\_centroids, dtype=np.uint8)  
  
  
def assign\_cluster(image, centroids):  
 rows, cols = image.shape  
 labels = np.zeros((rows, cols), dtype=np.uint8)  
  
 for i in range(rows):  
 for j in range(cols):  
 min\_dist = float('inf')  
 close\_cluster = -1  
  
 for index, centroid\_intensity in enumerate(centroids):  
 distance = abs(int(centroid\_intensity) - int(image[i][j]))  
  
 if distance < min\_dist:  
 min\_dist = distance  
 close\_cluster = index  
  
 labels[i][j] = close\_cluster  
  
 return labels  
  
  
def update\_centroids(image, labels, k):  
 new\_centroids = []  
  
 for cluster in range(k):  
 cluster\_points = image[labels == cluster]  
  
 if len(cluster\_points) == 0:  
 new\_centroids.append(0)  
 else:  
 new\_centroid = np.mean(cluster\_points)  
 new\_centroids.append(new\_centroid)  
  
 return np.array(new\_centroids, dtype=np.uint8)  
  
  
def k\_clustering(image, k, tolerance):  
 centroids = initial\_centroids(image, k)  
 rows, cols = image.shape  
  
 while True:  
 labels = assign\_cluster(image, centroids)  
 new\_centroids = update\_centroids(image, labels, k)  
 print("In this loop")  
  
 converge = True  
 for i in range(len(centroids)):  
 distance = abs(centroids[i] - new\_centroids[i])  
 if distance > tolerance:  
 converge = False  
 break  
  
 if converge:  
 break  
  
 centroids = new\_centroids  
  
 segmented\_image = np.zeros((rows, cols), dtype = np.uint8)  
 for cluster in range(k):  
 segmented\_image[labels == cluster] = int(255 / (k - 1)) \* cluster  
  
 print(np.unique(segmented\_image))  
  
 return segmented\_image  
  
  
*# Main*image = cv.imread("D:/Uni/Semester 6/DIP/Self/Lec/Assignment 1/dataset\_DIP\_assignment/train/images/003.bmp", 0)  
segmented\_image = k\_clustering(image, 3, 0)  
  
cv.imshow("Original Image", image)  
cv.imshow("K Means Segmentation", segmented\_image)  
cv.waitKey()

**Output:**

****