Name: **Muhammad Aqeel Afzal**

Roll. No: **i190650**

Section: **8A**

Assignment: **Report 2st**

Submitted To: **Dr. Akhtar Jamil**

Report

Question 01 part(a):

Description:

- *Separate_teath(path)* function takes an image as input.
- Read the image using the python library open cv.
- Then modify the pixel intensity if the intensity is less than 165 out of 255 then replace it with zero to separate the teeth.
- After then it converts back to BGR.
- In the end it will display the final image

Code:

```python
#Q1 a
import cv2  #importing open cv
import sys  #importing sys
import numpy as np  #importing numpy
import matplotlib.pyplot as plt
np.set_printoptions(threshold=np.inf)
def separate_teeth(path):   #function toseparate teeths
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    for i in range(0,len(img1)):  #setting red color intensity to zero of the require path of img
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] <165:
                    img1[i][j][k]=0


    img1 = cv2.cvtColor(img1,cv2.COLOR_RGB2BGR)   #converting back to bgr
    cv2.imshow('final img', img1)  #dispaly img
    cv2.waitKey()



path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\dental_xray.tif' #img path

separate_teeth(path)  #function call
```

Input image:

Output image:

Question 01 part(b):

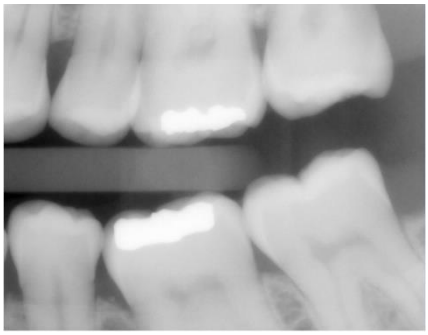
Description:

- *color_teath(path)* function takes an image as input.
- Read the image using the python library open cv.
- Then modify the pixel intensity if the intensity is greater than 230 out of 255 then replace it with 230 of red color to separate the teeth.
- After then it converts back to BGR.
- In the end it will display the final image

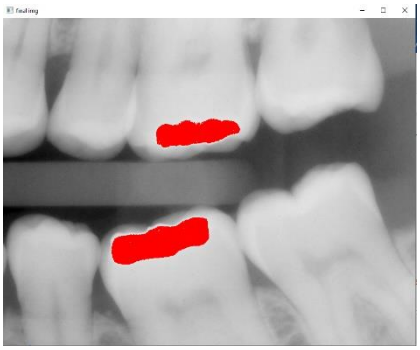
Code:

```python
#Q1 b
import cv2  #importing open cv
import sys  #importing sys
import numpy as np  #importing numpy
def color_teeth(path):   #function to color teath
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    for i in range(0,len(img1)):  #setting red color intensity to zero of the require path of img
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] >230:
                    img1[i][j][0]=255
                    img1[i][j][1]=0
                    img1[i][j][2]=0



    img1 = cv2.cvtColor(img1,cv2.COLOR_RGB2BGR)   #converting back to bgr
    cv2.imshow('final img', img1)  #dispaly img
    cv2.waitKey()



path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\dental_xray.tif' #img path

color_teeth(path)  #function call
```

Input image:



Output image:



Question 01 part(c) :

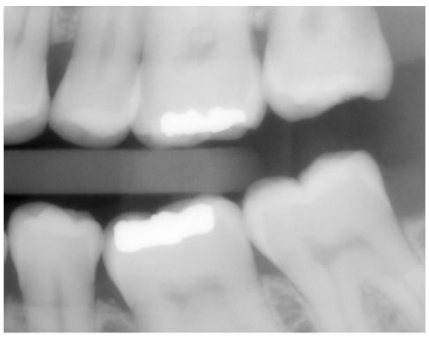
Description:

- *Cal_percentage (path)* function takes an image as input.
- Read the image using the python library open cv.
- Then modify the pixel intensity if the intensity is greater than 230 out of 255 with 230 of it will count it as effected pixels else as not be effected.
- Then calculate the percentage of the effected pixels.
- After then it converts back to BGR.
- In the end it will print the percentage

Code:

```
#Q1 c
import cv2  #importing open cv
import sys  #importing sys
import numpy as np  #importing numpy
def cal_percentage(path):   #function to remove a part of an image
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    count1=0
    count2=0
    for i in range(0,len(img1)):  #setting red color intensity to zero of the require path of img
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] >230:
                    count1+=1
                else:
                    count2+=1
    percentage=(count1*100)/(count1+count2)
    print(percentage,"%")

path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\dental_xray.tif' #img path

cal_percentage(path)  #function call
```

Input image:



Output:

```
3.204545913320818 %
```

Question 02 part(a):

Description:

- *Get_white_part (path)* function takes an image as input.
- Read the image using the python library open cv.
- And make two copies one to compare with the final image.
- Then modify the pixel intensity of the second image so that if the intensity is greater than 210 out of 255 set the pixel value to 255 else zero.
- After then it converts back to BGR.
- In the end, it will display both the image final as well as original.

Code:

```python
#Q2 a
def get_white_part(path):   #function to remove a part of an image
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img2 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image

    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    for i in range(0,len(img1)):  #setting red color intensity to zero of the require path of img
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] >210:
                    img1[i][j][k]=255
                else:
                    img1[i][j][k]=0



    img1 = cv2.cvtColor(img1,cv2.COLOR_RGB2BGR)   #converting back to bgr
    cv2.imshow('final img', img1)  #dispaly img

    cv2.imshow('original img', img2)  #dispaly img
    cv2.waitKey()


path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\brain.tif' #img path

get_white_part(path)  #function call
```

Input image:

Output image:

Question 02 part(b):

Description:

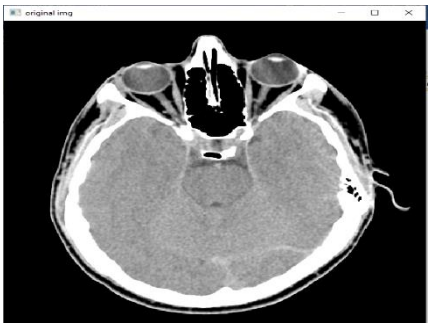
- *Row_wise_pixel_count (path)* function takes an image as input.

- Read the image using the python library open cv.
- And convert the image from BGR to RGB
- Then modify the pixel intensity of the second image so that if the intensity is greater than 200 out of 255 set the pixel value to 255 else zero.
- Then sum pixels with the intensity of 255 in each row.
- In the end, it will plot the histogram of all the intensities.

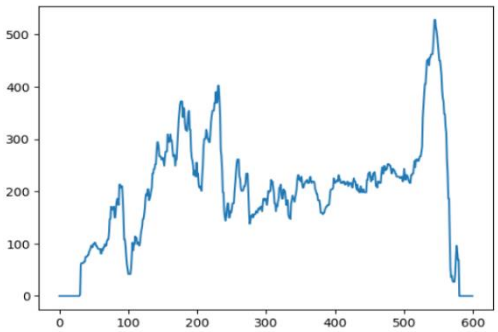
Code:

```python
#Q2 b
import cv2  #importing open cv
import sys  #importing sys
import numpy as np  #importing numpy
def row_wise_pixal_count(path):   #function to remove a part of an image
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img2 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    for i in range(0,len(img1)):  #setting red color intensity to zero of the require path of img
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] >200:
                    img1[i][j][k]=255
                else:
                    img1[i][j][k]=0


    white_pixels = []    #array to store white pixal row wise
    for row in img1:
        white_pixels.append(np.sum(row == 255)) #summing the pixals
    plt.plot(white_pixels)  #ploting the graph
    plt.show()

path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\brain.tif' #img path

row_wise_pixal_count(path)  #function call
```

Input image:



Output graph:



Question 02 part(c):

Description:

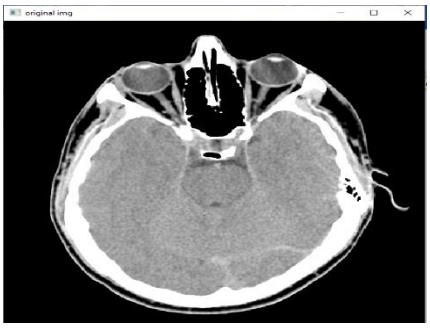
- *col_wise_pixel_count (path)* function takes an image as input.
- Read the image using the python library open cv.

- And convert the image from BGR to RGB
- Then modify the pixel intensity of the second image so that if the intensity is greater than 200 out of 255 set the pixel value to 255 else zero.
- Then sum pixels with the intensity of 255 in each column.
- In the end, it will plot the histogram of all the intensities.

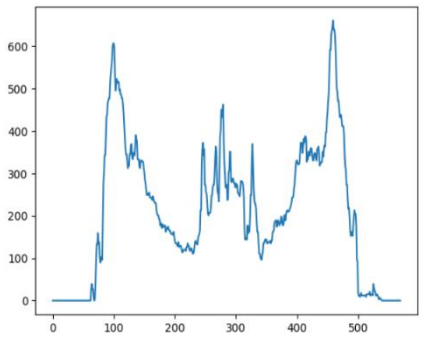
Code:

```python
#Q2 c
def col_wise_pixal_count(path):   #function to remove a part of an image
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img2 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    for i in range(0,len(img1)):  #setting red color intensity to zero of the require path of img
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] >200:
                    img1[i][j][k]=255
                else:
                    img1[i][j][k]=0


                        # Counting white pixals in each col
    white_pixels = []
    for i in range(img1.shape[1]):
        col = img1[:, i]
        white_pixels.append(np.sum(col == 255)) #summing the pixals

    plt.plot(white_pixels)  #ploting the graph
    plt.show()

path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\brain.tif' #img path

col_wise_pixal_count(path)  #function call
```

Input image:



Output graph:



Question 03:

Description:

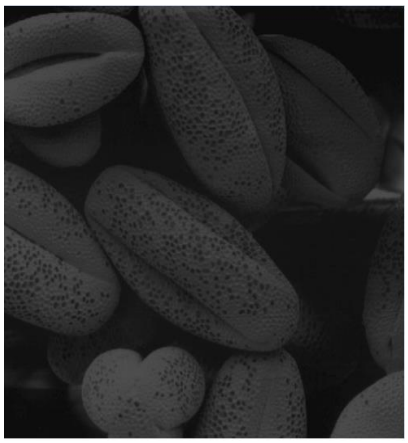
- *Log_transfermation (path)* function takes an image as input.
- Read the image using the python library open cv.

- Initialize an array of different values of c.
- Then use the plt library for plotting the final images
- Then apply the log transformation on the image with all different values of c.
- Then convert the image into 8-bit.
- In the end, it will plot the graphs of all the images at different values of c.
- According to my observation c = 100, gives the best result.

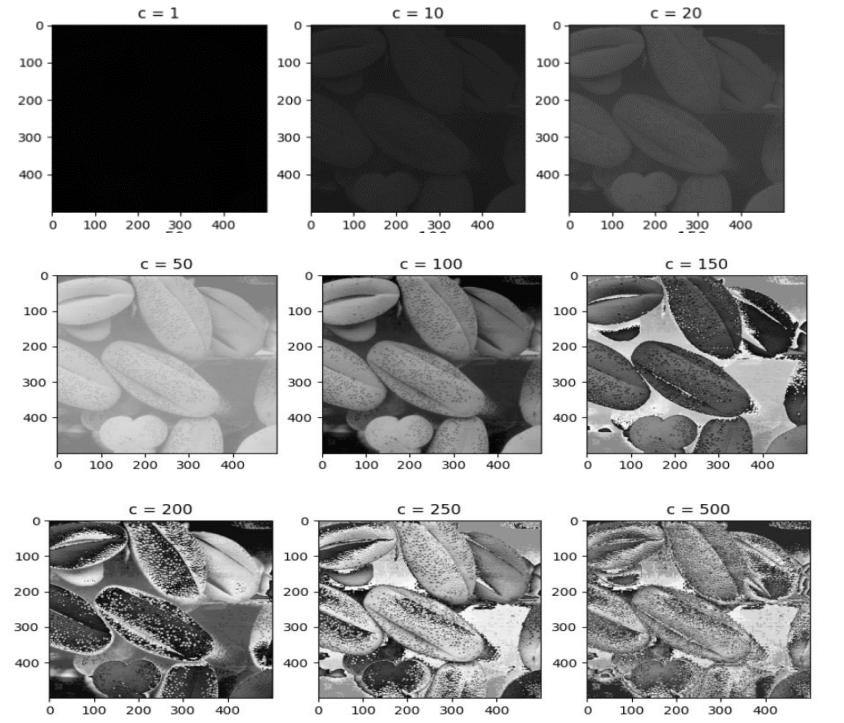
Code:

```python
# Q3
import cv2
import numpy as np
import matplotlib.pyplot as plt
def log_tranfermation(path):
    img1 = cv2.imread(r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\grain3.tif',
    c_values = [1, 10, 20, 50,100,150,200,250,500]  # array for scaling fector

    image, img_axs = plt.subplots(3,3, figsize=(10, 10)) #creating a grid of 9 images
    img_axs = img_axs.ravel()
    for i, c in enumerate(c_values):
        output = c * np.log(1 + img1)  #appling log transfarmation

        img=np.uint8(output)  #resizing inage to 8int
        img_axs[i].imshow(img, cmap='gray')  #setting gray image to show
        img_axs[i].set_title('c = {}'.format(c))  #setting title
    plt.show() # ploting
path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\grain3.tif' #img path
log_tranfermation(path)  #function call

```

Input image:



Output image:



Question 04 part(a):

Description:

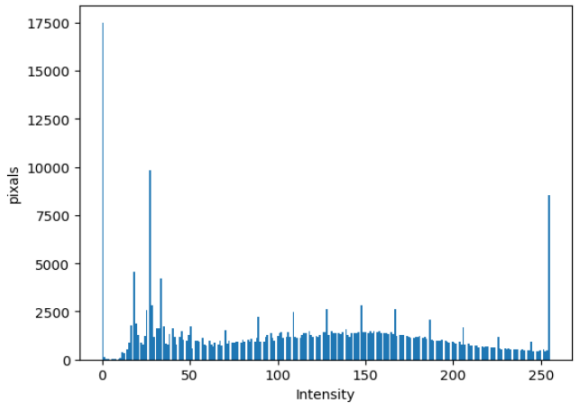
- *Count_intensity (path)* function takes an image as input.
- Read the image using the python library open cv in grayscale.
- Initialize the count array for each pixel count.
- Then count the number of pixels at each intensity value.
- Return the count.
- In the end, it will plot the histogram of counts of intensities.

Code:

```python
#Q4 a
def count_intensity(path):
    img1 = cv2.imread(path, cv2.IMREAD_GRAYSCALE) # reading image
    counts = [0] * 256  #array initalization for counting the intensity value

    for row in img1: #reading img rows one by one
        for i in row: #reading each of row
            counts[i] += 1  # increment the count of pixals

    return counts  #returning the count
path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\4.grain.tif' #path
counts = count_intensity(path)  #function call

plt.hist(range(256), bins=256, weights=counts) #ploting the intensity count by histogram
plt.xlabel('Intensity')
plt.ylabel('pixals')
plt.show()
```

Input image:

Output image:



Question 04 part(b):

Description:

- *Box_filter (path)* function takes an image as input and two values for box filters.
- Initialize the kernels one is 7x7 and the other is 3x3.
- Add the padding for kernel 7x7.
- Then apply the kernel 7x7.
- Then remove the padding for kernel 7x7.
- Add the padding for kernel 3x3.
- Then apply the kernel 3x3.
- Then remove the padding for kernel 3x3.
- Take the difference between both kernels.
- Take the absolute to make the values positive.
- Convert the image into grayscale.
- In the end subtract the output image from original image.
- Display the image.

Code:

```
#Q4 b
def box_filter(img,v1,v2):
    # defining the kernals
    kernel1 = [[1/49]*v1 for _ in range(7)]   #kernal 7
    kernel2 = [[1/9]*v2 for _ in range(3)]   #jernal 3

    padding1 = (len(kernel1) - 1) // 2    #size of kernal 1
    img_padded = np.zeros((img.shape[0] + padding1 * 2, img.shape[1] + padding1 * 2, 3), np.uint8)  #padding the image

    for i in range(img.shape[0]):    #padding adding
        for j in range(img.shape[1]):
            img_padded[i+padding1, j+padding1] = img[i,j]

    filter_img = np.zeros_like(img_padded)   #appling the kernal 1
    for i in range(padding1, img.shape[0] + padding1):
        for j in range(padding1, img.shape[1] + padding1):
            for k in range(3):
                filter_img[i,j,k] = np.sum(kernel1 * img_padded[i-padding1:i+padding1+1, j-padding1:j+padding1+1, k])


    filter_img = filter_img[padding1:img.shape[0]+padding1, padding1:img.shape[1]+padding1] # removing the padding
    padding2 = (len(kernel2) - 1) // 2     # size of kernal 2
    img_padded1 = np.zeros((img.shape[0] + padding2 * 2, img.shape[1] + padding2 * 2, 3), np.uint8)

```

```
filter_img = filter_img[padding1:img.shape[0]+padding1, padding1:img.shape[1]+padding1] # removing the padding
padding2 = (len(kernel2) - 1) // 2     # size of kernal 2
img_padded1 = np.zeros((img.shape[0] + padding2 * 2, img.shape[1] + padding2 * 2, 3), np.uint8)

for i in range(img.shape[0]):    #padding adding
    for j in range(img.shape[1]):
        img_padded1[i+padding2, j+padding2] = img[i,j]


filter_img1 = np.zeros_like(img_padded)    #appling the kernal 2
for i in range(padding2, img.shape[0] + padding2):
    for j in range(padding2, img.shape[1] + padding2):
        for k in range(3):
            filter_img1[i,j,k] = np.sum(kernel2 * img_padded1[i-padding2:i+padding2+1, j-padding2:j+padding2+1, k])

# Removeing padding
filter_img1 = filter_img1[padding2:img.shape[0]+padding2, padding2:img.shape[1]+padding2]

for i in range(filter_img.shape[0]):   #abs negative to positive
    for j in range(filter_img.shape[1]):
        for k in range(filter_img.shape[2]):
            filter_img[i][j][k]=abs(filter_img[i][j][k]-filter_img1[i][j][k])
```

```
44
45      img1_abs=cv2.cvtColor(filter_img, cv2.COLOR_BGR2GRAY)   #converting to gray scale
46
47      for i in range(img1_abs.shape[0]):   #subtracting the original image
48          for j in range(img1_abs.shape[1]):
49              img[i][j]=img[i][j]-img1_abs[i][j]
50      return img
51
52  # Load the image
53  path = r'E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\4.grain.tif'
54  img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
55  img = box_filter(img, 7,3)
56  cv2.imshow('Filtered Image', img)   #display the image
57  cv2.waitKey(0)
58  cv2.destroyAllWindows()
```

Input image:

Output image:

Question 05:

Description:

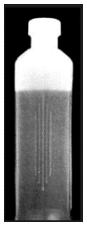
- *Filled_bottle (path)* function takes an image as input and threshold value.

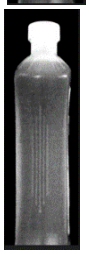
- Read the image using the python library open cv.
- Then check the pixel intensity if the intensity is greater than 200 out of 255 then add 1 to the filled counter else to not filled the counter.
- In the end calculate the percentage.
- Check the threshold and display the massage with percentage
- In the end, display the image also.

Code:

```python
#Q5
import cv2  #importing open cv
import sys  #importing sys
import numpy as np  #importing numpy
def filled_bottle(path, threshold=90 ):   #function to remove a part of an image
    img1 = cv2.imread(path, cv2.IMREAD_COLOR) # reading image
    img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)  #coverting into rgb
    count1=0
    count2=0
    for i in range(0,len(img1)):  #counting the pixals for the filled part and not filled part of the bottle
        for j in range(0,len(img1[0])):
            for k in range(0,len(img1[0][1])):
                if img1[i][j][k] <200:
                    count1+=1       #filled pivxals (black)
                else:
                    count2+=1    #not filled pixals(while)


    percentage=(count1*100)/(count1+count2)  #percentage calculating
    print("Filled percentage: {:.2f}%".format(percentage))
    if percentage < threshold:
        print("Bottle is not properly filled")
    else:
        print("Bottle is properly filled")
    img1 = cv2.cvtColor(img1,cv2.COLOR_RGB2BGR)   #converting back to bgr
    cv2.imshow('final img', img1)  #dispaly img
    cv2.waitKey()



path = r"E:\Samester 08\Digital Image Processing\Assignments\assignment01\Assignment#1-(B)\data\bottle1.jpg" #img path

filled_bottle(path)  #function call
```

Input images:





Output percentage:

```
Filled percentage: 80.02%
Bottle is not properly filled
```

```
Filled percentage: 92.94%
Bottle is properly filled
```