Assignment 1 Report

Digital Image Processing

By:

Danyal Faheem

19I-2014 CS-F

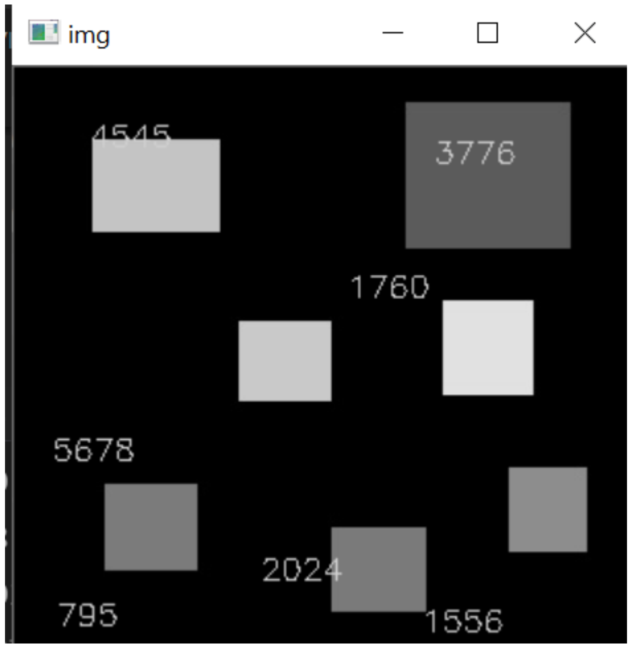
Question 1:

Part a.

First of all I read the image into a variable and converted it to grayscale using the OpenCV imread function. To perform intensity slicing, I realized that each object would have it's own unique range of intensities. An assumption that I made was that the range of intensities of each object would not exceed 10. Therefore, I used python sets to append each pixel into a set which would only take up unique values discarding duplicates itself. As the objects have a range of intensities, I had some extra values to be discarded. Therefore I ran a loop on the set excluding any values that are in the range of 10 of each other. This way, I had 7 unique intensities for 7 unique objects.

After this step, I traversed the image once again and if there was a pixel that lied in the range of 10 of the intensity in the set, I would increase the pixel count for that object. Traversing the entire image for 7 objects, I had the pixel count for each object. Then, I used the putText() function in

OpenCV to draw the text on the objects themselves. Later, using the OpenCV imshow() function to show the image in a new window. The output was as follows:



As I did not have the exact locations to print the text due to making the program as generic as I could, therefore, the printing of the text had been misaligned due to the different range of intensities each object had.

Code for Q1:

```
#Importing our libraries for use

import cv2

import numpy as np

import matplotlib.pyplot as plt

#Reading our image and then converting it to grayscale

path = "..\I192014 Danyal Faheem - Assignment#1\Assignment#1\img.jpg"
```

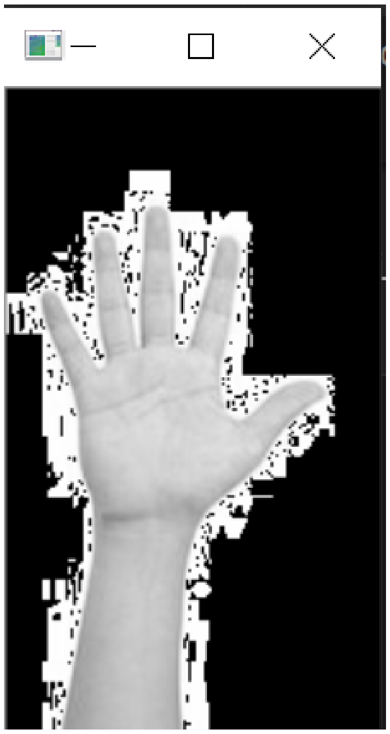
```
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

#Since we have multiple objects, grabbing the unique intensity values from
the image in a python set

pixel_set = {0}

counter = 0

count = 0

for x in img:

    for y in x:

        pixel_set.add(y)

# print(pixel_set)

#Converting the set to a list to apply operations on it

arr = list(pixel_set)

arr.sort()

#print(arr)

#There are a range of intensity values in a single object

#To decide on one intensity for each object, remove similar intensities in
the range of 10 of each intensity

for x in arr:

    for y in arr:

        if (y > x - 10 and y < x + 10):

            arr.remove(y)

#Running the same loop again to make sure only 1 intensity per object
remains

for x in arr:

    for y in arr:
        if (y > x - 5 and y < x + 5):

            arr.remove(y)

pixel_count = [0] * len(arr)

#print(pixel_count)

#Now that we have the intensity of each object, counting the number of
pixels that are in a range of 10 of the selected pixel

counter = 0

for i in arr:

    for x in img:

        for y in x:

                if (y > i - 10 and y < i + 10 and y != 0):

                    pixel_count[counter] += 1

    counter += 1

counter = 0

count = 0

index = 0

#Writing the counts of the pixel on the location of each pixel we choose
to use as our range pixel

for i in arr:

    counter = 0

    for x in img:

        count = 0

        for y in x:

            if y == i and y in pixel_set:
                print((counter, count))
                cv2.putText(img, str(pixel_count[index]), (counter,
count), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (209, 80, 0, 255), 1)
                pixel_set.remove(y)
            count += 1
        counter += 1
    index += 1


#Displaying the new image
print(arr)
print(pixel_count)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Part c.

Using OpenCV, we would automate many of the tasks we had to manually do such as looping through each individual pixel ourselves. The code would be shortened and therefore, the time taken would be less as well. For example, in part b where I had done the traversing manually, using Jupyter Notebook, it took around 4 secs for the program to execute which is relatively very slow for such a small program and OpenCV would increase the speed further making quick executions of such small algorithms.

Q.2

a. First of all, I had read the image into a variable and converted it to grayscale using the OpenCV function imread(). To get only hand, I had applied thresholding. I traversed the entire image and took a negative of every white pixel on the image which would make all the white pixels black thus leaving only the hand behind. The output was as follows:



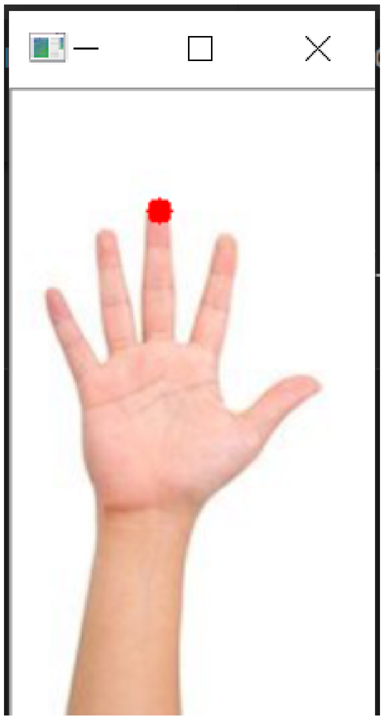
The uneven pixelation shows that the image itself was not uniform in the intensities and that around the hand, the pixels were not white but of ranging intensities.

Code for part a.

```
#Importing our libraries for use
```

```python
import cv2

import numpy as np

import matplotlib.pyplot as plt

path = "..\I192014 Danyal Faheem - Assignment#1\Assignment#1\img2.jpg"

img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

#Traversing the image array and taking the negative of any white pixels in the array

counter = 0

for x in img:

    count = 0

    for y in x:

        if y > 254:

            img[counter][count] -= 255 #Taking negative by minus 255

        count += 1

    counter += 1

# Displaying the image

cv2.imshow('img',img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

b. For the second part, I once again read the image but not in grayscale this time as we had to print a red dot and that could not be done in a grayscale image. After this, I traversed the image and the first location I found that would not be the white pixel, I had drawn a red pixel there using the circle() function in OpenCV. Then, I would return in the function so as to stop the traversing. After this, I would display the image using the imshow() function in OpenCV. The output is as follows:



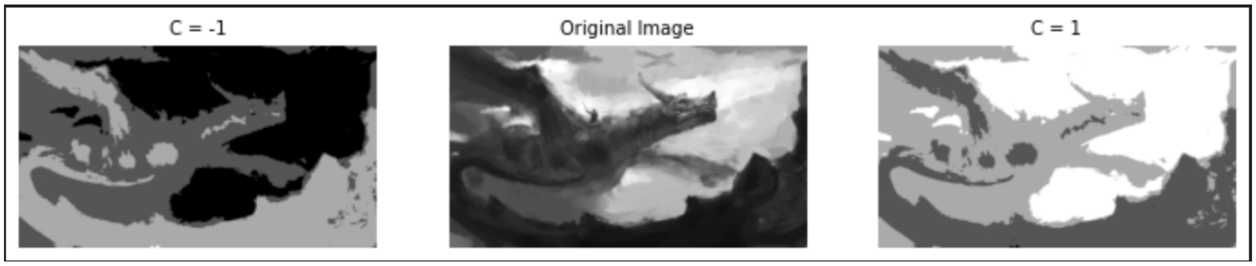
Code for Part b:

```
#Reading the image again but not in grayscale mode to print red dot

img = cv2.imread(path)
```

```
#Using the first non white pixel we find, drawing a filled red circle at
that spot

def draw():

    counter = 0

    for x in img:

        count = 0

        for y in x:

            count2 = 0

            for i in y:

                if i < 254:

                    cv2.circle(img, (counter + 27, count), 5, (0,0,255),
-1)

                    return

                count2 += 1

            count += 1

        counter += 1


draw()


#Displaying the image

cv2.imshow('img',img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Q.3

A. For the first part, of log transformations, I used simple mathematical formulas by using the np.log() methods provided in the numpy python library. Then I performed the actions for different values of c in the case of logarithmic and displayed the output using the matplotlib library in Python. The results are as follows:

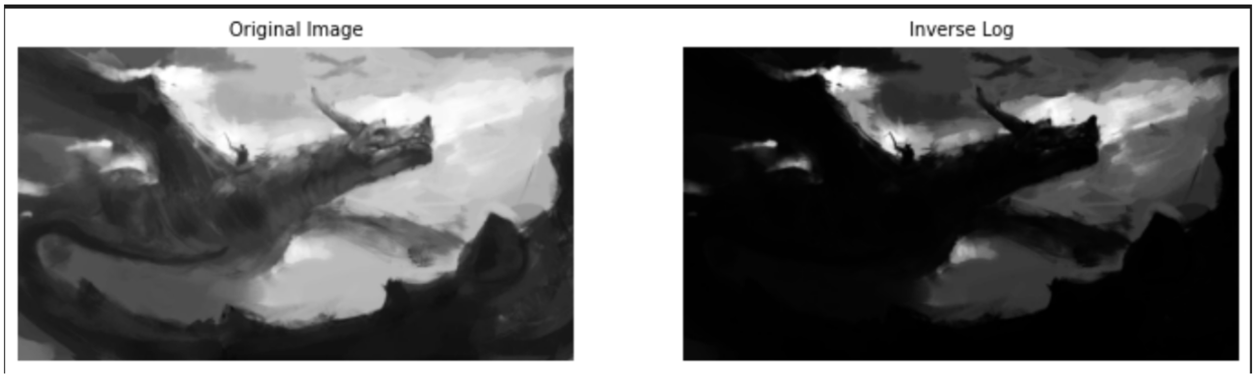


Logarithmic Transformation

The higher amplitude of gray levels is compressed while the lower amplitude of gray levels is expanded. By increasing C, we are making the brighter regions brighter and less brighter regions into a shade of bright. The code for logarithmic transformation is:

```
# Logarithmic Transformation

# Using the formula s = c * log(1 + r)

#keeping c = 1

c = 1

img1 = c * (np.log(1.1 + img))

#keeping c = -1

c = 10
```

```
img2 = c * (np.log(1.1 + img))


#Converting into arrys to display as image

img1 = np.array(img1, dtype = np.uint8)

img2 = np.array(img2, dtype = np.uint8)




fig, axs = plt.subplots(1,3,figsize=(12,12))

#Display the C= -1 image on the left

axs[0].imshow(img2,cmap='gray')

axs[0].axis('off')

axs[0].set_title('C = -1',fontsize='medium')


#Displaying the Original Image in the middle

axs[1].imshow(img,cmap='gray')

axs[1].axis('off')

axs[1].set_title('Original Image',fontsize='medium')


#Displaying the C = 1 middle on the right

axs[2].imshow(img1,cmap='gray')

axs[2].axis('off')

axs[2].set_title('C = 1',fontsize='medium')

plt.show()
```



Inverse logarithmic Transformation

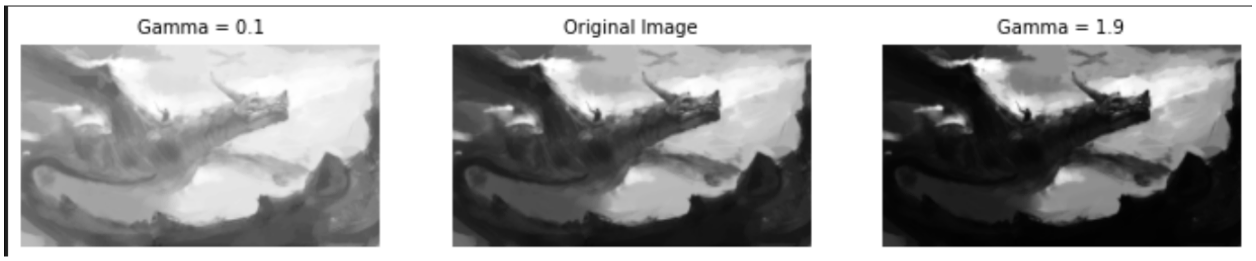
For inverse logarithmic, the opposite happens where the lower amplitude are expanded and the higher amplitude are compressed, meaning darker regions get darker and the lighter regions get a shade of dark. The code of inverse logarithmic is as follows:

```
# Inverse Logarithmic Transformation

c = 255 / np.log(1 + np.max(img))

img1 = np.exp(img ** 1 / c) - 1




#Converting into arrys to display as image

img1 = np.array(img1, dtype = np.uint8)




fig, axs = plt.subplots(1,2,figsize=(12,12))



#Displaying the Original Image in the middle
```

```
axs[0].imshow(img,cmap='gray')

axs[0].axis('off')

axs[0].set_title('Original Image',fontsize='medium')


#Displaying the Inverse Log image on the right

axs[1].imshow(img1,cmap='gray')

axs[1].axis('off')

axs[1].set_title('Inverse Log',fontsize='medium')

plt.show()
```

B. Power Law nth Power:

For Power law nth Power, I again used the inbuilt power() function present in the numpy library. There, I performed the transformation for different values of gamma to observe the result using the matplotlib library in Python. The output it as follows:



Power Law Nth Power Transformation

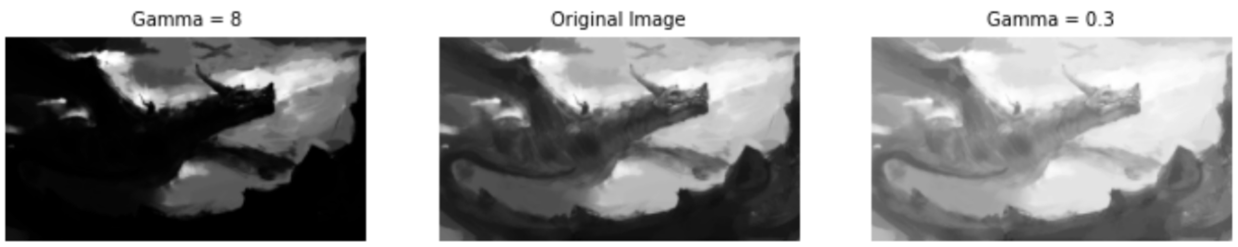
By increasing the gamma value, we can observe that it is decreasing the intensity values of the darker regions and by decreasing the gamma value below 1, it is increasing the intensity of the darker regions. The code for Power Law Nth Power Transformation is as follows:

```
#Power law nth power

#Using the formula s = c * r^γ

c = 1

#Using gammae = 0.1

Gamma = 0.1

img1 = c * (img ** Gamma)

#Using gammae = 1.9

Gamma = 1.9
```

```
img2 = c * (img ** Gamma)


fig, axs = plt.subplots(1,3,figsize=(12,12))
#Display the Gamma = 1.9 image on the right
axs[2].imshow(img2,cmap='gray')
axs[2].axis('off')
axs[2].set_title('Gamma = 1.9',fontsize='medium')


#Displaying the Original Image in the middle
axs[1].imshow(img,cmap='gray')
axs[1].axis('off')
axs[1].set_title('Original Image',fontsize='medium')


#Displaying the gamma = 0.2 image on the left
axs[0].imshow(img1,cmap='gray')
axs[0].axis('off')
axs[0].set_title('Gamma = 0.1',fontsize='medium')
plt.show()
```

C. Power Law Nth Root:

For Power Law Nth root, I again used the built in power function in numpy with a few modifications to the parameters. The output is as follows:



Power Law nth Root Transformation

Here, we can observe that the opposite from nth Power transformation happens as we increase the gamma value, the regions with darker intensities become darker whereas decreasing the gamma value makes the darker regions more brighter. The code for nth Root Transformation is:

```
#Power law nth root

#Using the formula s = c * y/r

c = 1

#Using gammae = 0.1

Gamma = 0.3

img1 = c * np.power(img, 1/Gamma)

#Using gammae = 1.9

Gamma = 8

img2 = c * np.power(img, 1/Gamma)



fig, axs = plt.subplots(1,3,figsize=(12,12))

#Display the Gamma = 0.3 image on the right

axs[2].imshow(img2,cmap='gray')

axs[2].axis('off')
```

```
axs[2].set_title('Gamma = 0.3',fontsize='medium')


#Displaying the Original Image in the middle

axs[1].imshow(img,cmap='gray')

axs[1].axis('off')

axs[1].set_title('Original Image',fontsize='medium')


#Displaying the gamma = 08 image on the left

axs[0].imshow(img1,cmap='gray')

axs[0].axis('off')

axs[0   ].set_title('Gamma = 8',fontsize='medium')

plt.show()
```

Thank you!