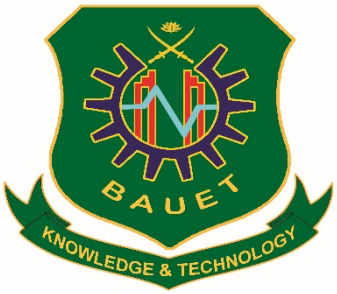
Bangladesh Army University of Engineering & Technology (BAUET)
Qadirabad, Natore-6431



Department of Computer Science and Engineering (CSE)

Course Code: CSE-3216

Course Title: Digital Image Processing Sessional

SUBMITTED BY

Name: Md. Arik Rayhan
ID: 20104033
Batch: 11th
Year: 3rd
Semester: 2nd
Section: Odd
Session: 2020-2021

SUBMITTED TO

Md. Nazmus Salehin
Lecturer, Dept. of CSE, BAUET

Md. Abul Ala Walid
Lecturer, Dept. of CSE, BAUET

INDEX

Task-01: Take three float values in three variables by user input and find out the sum of three values.

Code:

```
a = float(input("Enter the first number: "))
b = float(input("Enter the second number: "))
c = float(input("Enter the third number: "))
sum = a + b + c
print("The sum of three numbers is: ", sum)
```

Input/Output:

Enter the first number: 8
Enter the second number: 12
Enter the third number: 12
The sum of three numbers is: 32

Task-02: Take three lists, print them and find out the sum of three lists.

Code:

```
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8, 9, 10]
list3 = [11, 12, 13, 14, 15]
print("List 1: ", list1)
print("List 2: ", list2)
print("List 3: ", list3)
#sum of all elements in the list
sum = 0
for i in list1:
    sum = sum + i
for i in list2:
    sum = sum + i
for i in list3:
    sum = sum + i
print("Sum of all elements in the list: ", sum)
```

Input/Output:

List 1: [1, 2, 3, 4, 5]
List 2: [6, 7, 8, 9, 10]
List 3: [11, 12, 13, 14, 15]
Sum of all elements in the list: 120

Task-03: Print the sum of first 100 numbers using for loop, while loop and do while loop

Code:

```
# For loop
sum = 0
for i in range (1, 101):
    sum += i
print ("The sum of first 100 numbers using for loop is: ",
sum)

# While loop
sum = 0
i = 1
```

```
while i <= 100:
    sum += i
    i += 1
print ("The sum of first 100 numbers using while loop is: ",
sum)

# Do while loop
sum = 0
i = 1
while True:
    sum += i
    i += 1
    if i > 100:
        break
print ("The sum of first 100 numbers using do while loop is:
", sum)
```

Input/Output:

The sum of first 100 numbers using for loop: 5050
The sum of first 100 numbers using while loop is: 5050
The sum of first 100 numbers using do while loop is: 5050

Task-04: Print the Fibonacci series using for loop.

Code:

```
a = 0
b = 1
print("Fibonacci series using for loop: ")
for i in range(1, 20):
    print(a, end = " ")
    c = a + b
    a = b
    b = c
```

Input/Output:

Fibonacci series using for loop:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584

Task-05: Take three strings as user input, print them and concatenate them.

Code:

```
a = input("Enter the first string: ")
b = input("Enter the second string: ")
c = input("Enter the third string: ")
print("The three strings are: ", a, b, c)
print("The concatenated string is: ", a + b + c)
```

Input/Output:

The three strings are: My name is User
The concatenated string is: My name isUser

Task-06: Perform String Slicing operation: Slice from the Start, Slice to the End and slicing using Negative Indexing.

Code:

```
a = "Digital Image"
print ("The string is: ", a)
print ("Slicing from the start: ", a[:7])
print ("Slicing to the end: ", a[7:])
print ("Slicing using negative indexes: ", a[-5:-1])
```

Input/Output:

The string is: Bangladesh Army
Slicing from the start: Bangl
Slicing to the end: desh Army
Slicing using negative indexes: Arm

Task-07: Determine the grade of a student using if..else logic if subject number is taken as user input

Code:

```
marks = int(input("Enter the marks: "))
if marks >= 90:
    print("Grade: A")
elif marks >= 80:
    print("Grade: B")
elif marks >= 70:
    print("Grade: C")
elif marks >= 60:
    print("Grade: D")
elif marks >= 50:
    print("Grade: E")
else:
    print("Grade: F")
```

Input/Output:

Enter the marks: 81
Grade: B

Task-08: Determine the grade of a student using switch..case logic if subject number is taken as user input.

Code:

```
grade_dict = {
    range(90, 101): "A",
    range(80, 90): "B",
    range(70, 80): "C",
    range(60, 70): "D",
    range(50, 60): "E",
    range(0, 50): "F"
}
marks = int(input("Enter the marks: "))
for key in grade_dict:
    if marks in key:
        print("Grade:", grade_dict[key])
        break
```

Input/Output:

Enter the marks: 98
Grade: A

Task-09: Take four NumPy arrays (nparray1, nparray2, nparray3, nparray4). Define nparray1= [[1,2,3,4], [5,6,7,8]]), nparray2 = [5,6,7,8], nparray3 as zeros of dimension (3,4) and nparray4 as ones of dimension (3,7). Now perform NumPy array slicing: selecting 1st 2 rows and 1 st two columns, first element, middle element, and last element individually.

Code:

```
import numpy as np
nparray1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
nparray2 = np.array([5, 6, 7, 8])
nparray3 = np.zeros((3, 4))
nparray4 = np.ones((3, 7))
print("nparray1: ", nparray1)
print("nparray2: ", nparray2)
print("nparray3: ", nparray3)
print("nparray4: ", nparray4)
print("Slicing nparray1: ", nparray1[:2, :2])
print("Slicing nparray2: ", nparray2[0])
print("Slicing nparray3: ", nparray3[1, 2])
print("Slicing nparray4: ", nparray4[2, 6])
```

Input/Output:

```
nparray1: [[1 2 3 4]
 [5 6 7 8]]
nparray2: [5 6 7 8]
nparray3: [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
nparray4: [[1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1.]]
Slicing nparray1: [[1 2]
 [5 6]]
Slicing nparray2:  5
Slicing nparray3:  0.0
Slicing nparray4:  1.0
```

Task-10: Take a 5x6 NumPy array and flip the 5x6 NumPy array.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

array = np.array([[255, 0, 1, 100, 25, 101], [10, 200, 70, 80,
 120, 150], [95, 30, 30, 81, 96, 77], [87, 89, 220, 250, 100,
10], [18, 7, 221, 21, 8, 15]])
temp = array.copy()
```

```
array = np.flip(array, 0)#vertical
array = np.flip(array, 1)#horizontal
print(temp)
print(array)
```

Input/Output:

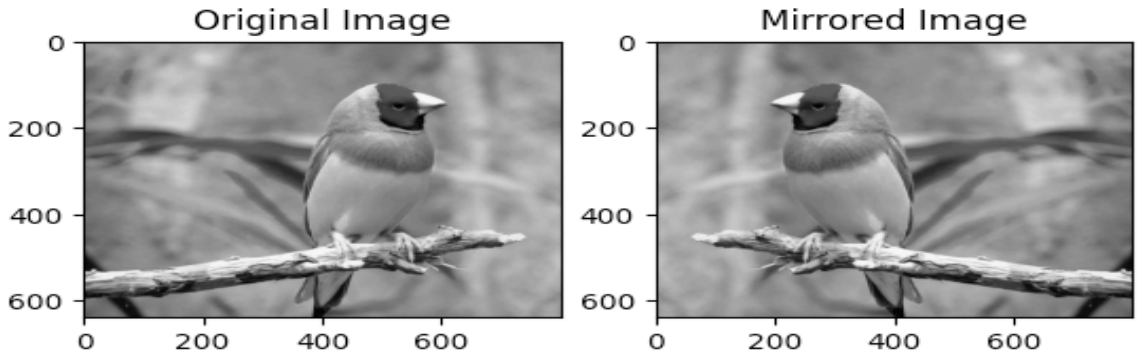
```
[[255   0   1 100  25 101]
 [ 10 200  70  80 120 150]
 [ 95  30  30  81  96  77]
 [ 87  89 220 250 100  10]
 [ 18   7 221  21   8  15]]
[[ 15   8  21 221   7  18]
 [ 10 100 250 220  89  87]
 [ 77  96  81  30  30  95]
 [150 120  80  70 200  10]
 [101  25 100   1   0 255]]
```

Task-11: Take a grayscale image and generate the mirror image of the original image. Show input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
#step-3: Create another matrix with same dimentions
img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
height=img_gray.shape[0]#no of rows
width=img_gray.shape[1]#no of colums
img_flip=np.zeros((height,width))

#step-4: flip img_gray on y-axis
for i in range(height):
    for j in range(width):
        img_flip[i][j]=img_gray[i][width-j-1]

#show the image in subplot
plt.subplot(1,2,1)
plt.imshow(img_gray,cmap='gray')
plt.title('Original Image')
plt.subplot(1,2,2)
plt.imshow(img_flip,cmap='gray')
plt.title('Mirrored Image')
```

Input/Output:

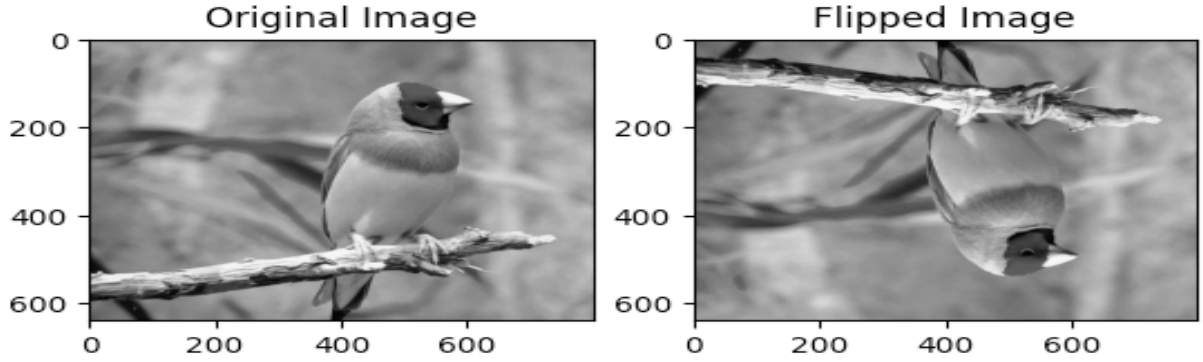


Task-12: Take a grayscale image and generate the flipped image of the original image. Show input and output image side by side in a subplot with a title.

Code:

```
import cv2
img = cv2.imread("photo.jpg")

array = np.flip(array, 0)#vertical
array = np.flip(array, 1)#horizontal
img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
height=img_gray.shape[0]#no of rows
width=img_gray.shape[1]#no of colums
img_flip=np.zeros((height,width))

#step-4: flip img_gray on y-axis
for i in range(height):
    for j in range(width):
        img_flip[i][j]=img_gray[i][width-j-1]
img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#flip the image
img_flip=np.flip(img_gray,0)
#show the image in subplot
plt.subplot(1,2,1)
plt.imshow(img_gray,cmap='gray')
plt.title('Original Image')
plt.subplot(1,2,2)
plt.imshow(img_flip,cmap='gray')
plt.title('Flipped Image')
```

Input/Output:



Task-13: Take an 8-bit grayscale image and apply Thresholding such that the pixel having intensity values below 150 will be replaced by 0 and pixels having intensity value equal or above 150 will be replaced by 255. Display input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
array = np.array([[9,5,6,7,2],[1,1,0,1,2],[50,21,25,99,100],[1
50,255,250,130,200],[145,135,139,190,180]])
temp = array.copy()
#thresholding the array
for i in range(array.shape[0]):
    for j in range(array.shape[1]):
        if array[i][j] < 150:
            array[i][j] = 0
        else:
            array[i][j] = 255
print(temp)
print(array)
```

Input/Output:

```
[[  9   5   6   7   2]
 [  1   1   0   1   2]
 [ 50  21  25  99 100]
 [150 255 250 130 200]
 [145 135 139 190 180]]
[[  0   0   0   0   0]
 [  0   0   0   0   0]
 [  0   0   0   0   0]
 [255 255 255   0 255]
 [  0   0   0 255 255]]
```

Task-14: Take an 8-bit grayscale image and apply Thresholding such that the pixel having intensity values below 50 will be replaced by 0 and pixels having intensity value above 150 will be replaced by 255. Display input and output image side by side in a subplot with a title

Code:

```
import cv2
import numpy as np
array = np.array([[9,5,6,7,2],[1,1,0,1,2],[50,21,25,99,100],[1
50,255,250,130,200],[145,135,139,190,180]])
temp = array.copy()
#thresholding the array
for i in range(array.shape[0]):
    for j in range(array.shape[1]):
        if array[i][j] < 50:
            array[i][j] = 0
        if array[i][j] > 150:
            array[i][j] = 255
print(temp)
print(array)
```

Input/Output:

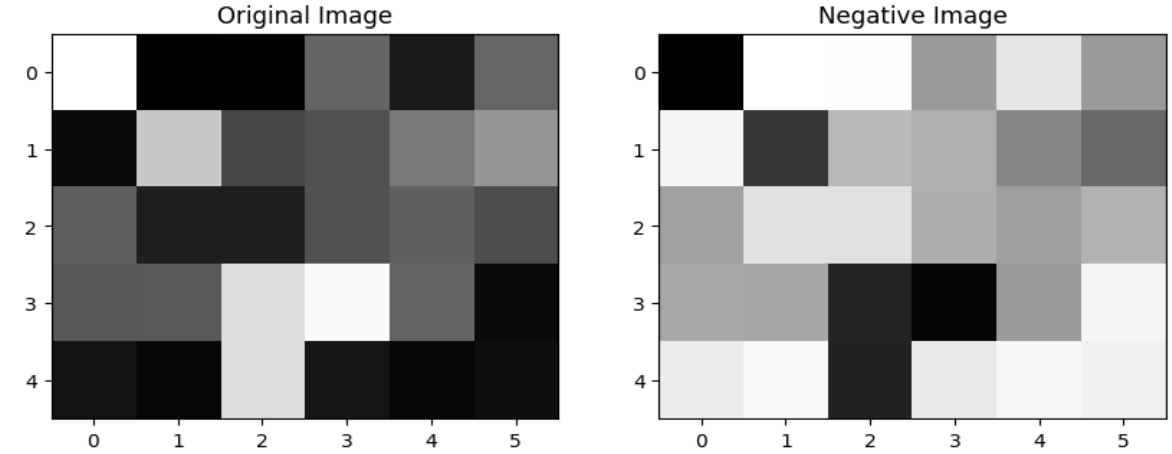
```
[[  9   5   6   7   2]
 [  1   1   0   1   2]
 [ 50  21  25  99 100]
 [150 255 250 130 200]
 [145 135 139 190 180]]
[[  0   0   0   0   0]
 [  0   0   0   0   0]
 [ 50   0   0  99 100]
 [150 255 255 130 255]
 [145 135 139 255 255]]
```

Task-15: Take a 5x6 NumPy array and apply negative image operations (calculate image bit dynamically). Convert the input and output NumPy array into an image and show them side by side in a subplot with a title

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
array = np.array([[255, 0, 1, 100, 25, 101], [10, 200, 70, 80,
 120, 150], [95, 30, 30, 81, 96, 77], [87, 89, 220, 250, 100,
10], [18, 7, 221, 21, 8, 15]])
temp = array.copy()
max_pixel = np.max(array)
max_pixel = (2**(np.ceil(np.log2(max_pixel))))-1
for i in range(0, 5):
    for j in range(0, 6):
        array[i][j] = max_pixel-array[i][j]
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(temp, cmap='gray')
plt.subplot(1, 2, 2)
plt.imshow(array, cmap='gray')
plt.title('Negative Image')
```

Input/Output:

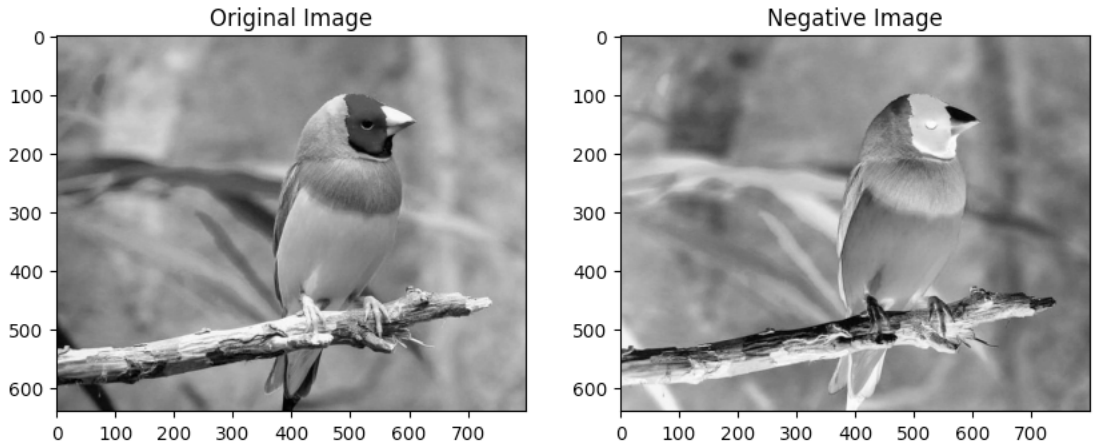


Task-16: Take a grayscale image and apply negative image operations (calculate image bit dynamically). Show input and output image side by side in a subplot with a title. Use PyPlot to plot the negative transformation curve (r on the x-axis and s on the y-axis).

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

#negative of an image
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
temp = img_gray.copy()

max_pixel = np.max(img_gray)
max_pixel = (2**(np.ceil(np.log2(max_pixel))))-1 #L-1 = 255

height = img_gray.shape[0]
width = img_gray.shape[1]

for i in range(0, height):
    for j in range(0, width):
        img_gray[i][j] = max_pixel-img_gray[i][j]

plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(temp, cmap='gray')
plt.subplot(1, 2, 2)
plt.imshow(img_gray, cmap='gray')
plt.title('Negative Image')
```

Input/Output:

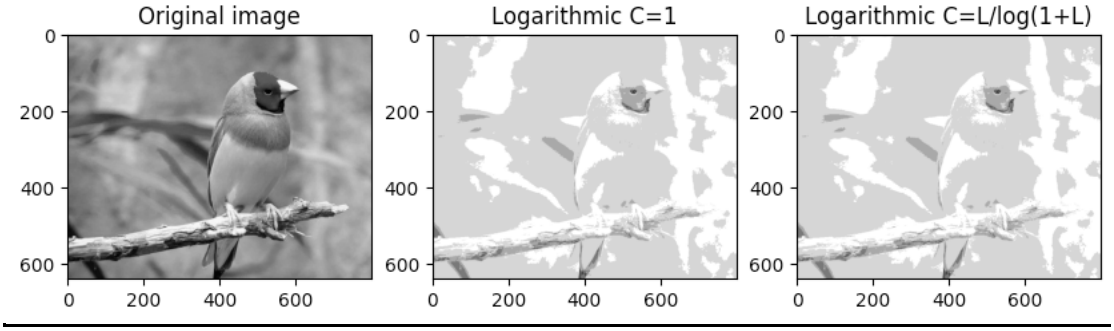


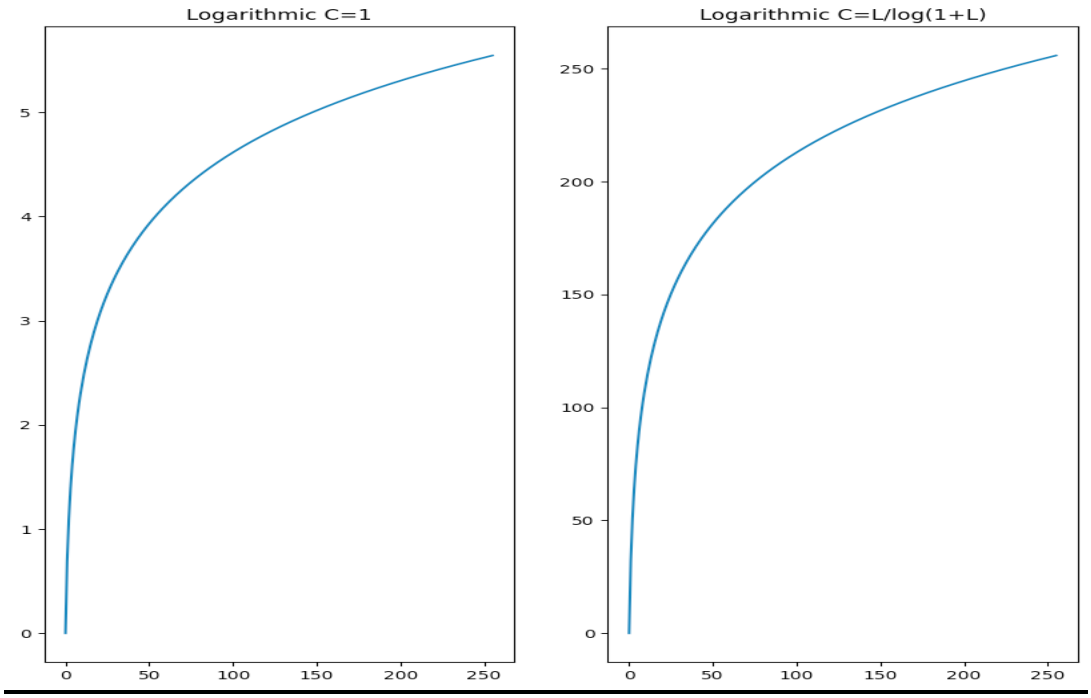
Task-17: Take a grayscale image and apply log transformation image operations (C = 1 and C = L/ (log10 (1+r))). Show input and output image side by side in a subplot with title. Use PyPlot to plot log transformation curve (r in x-axis and s in y-axis).

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
# logarithmic transformation
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gray_CL=img_gray.copy()
temp = img_gray.copy()

max_pixel = np.max(array)
L = 2**(np.ceil(np.log2(max_pixel)))
C=L/np.log(1+L)

height = img_gray.shape[0]
width = img_gray.shape[1]

for i in range(0, height):
    for j in range(0, width):
        img_gray[i][j] = np.log(1+img_gray[i][j])
        img_gray_CL[i][j]=C*img_gray[i][j]
plt.figure(figsize=(10, 10))
plt.subplot(1, 3, 1)
plt.imshow(temp, cmap='gray')
plt.title('Original image')
plt.subplot(1, 3, 2)
plt.imshow(img_gray, cmap='gray')
plt.title('Logarithmic C=1')
plt.subplot(1, 3, 3)
plt.imshow(img_gray_CL, cmap='gray')
plt.title('Logarithmic C=L/log(1+L)')
# plot logarithmic plotting
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
height = img_gray.shape[0]
width = img_gray.shape[1]
r = np.unique(img_gray)
s1 = np.zeros(len(r))
s2 = np.zeros(len(r))
for i in range(0, len(r)):
    s1[i] = np.log(1+r[i])
    s2[i] = C*s1[i]
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.plot(r, s1)
plt.title('Logarithmic C=1')
plt.subplot(1, 2, 2)
plt.plot(r, s2)
plt.title('Logarithmic C=L/log(1+L)')
```

Input/Output:





Task-18: Take a grayscale image and apply gamma transformation image operations (γ = 0.2, γ = 0.4, γ = 0.6, γ= 1.0, γ = 1.5, γ = 2.5, γ = 5.0). Show input and output image side by side in a subplot with a title. Use PyPlot to plot the gamma transformation curve for each γ value individually (r on the x-axis and s on the y-axis).

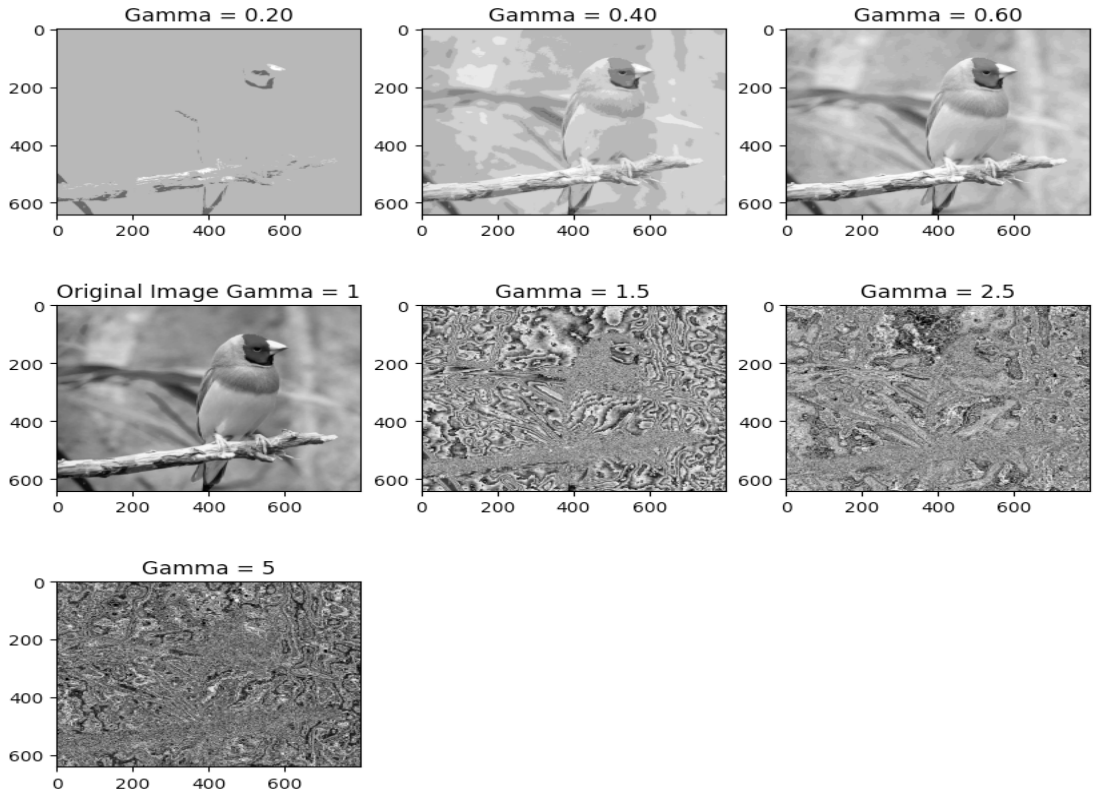
Code:

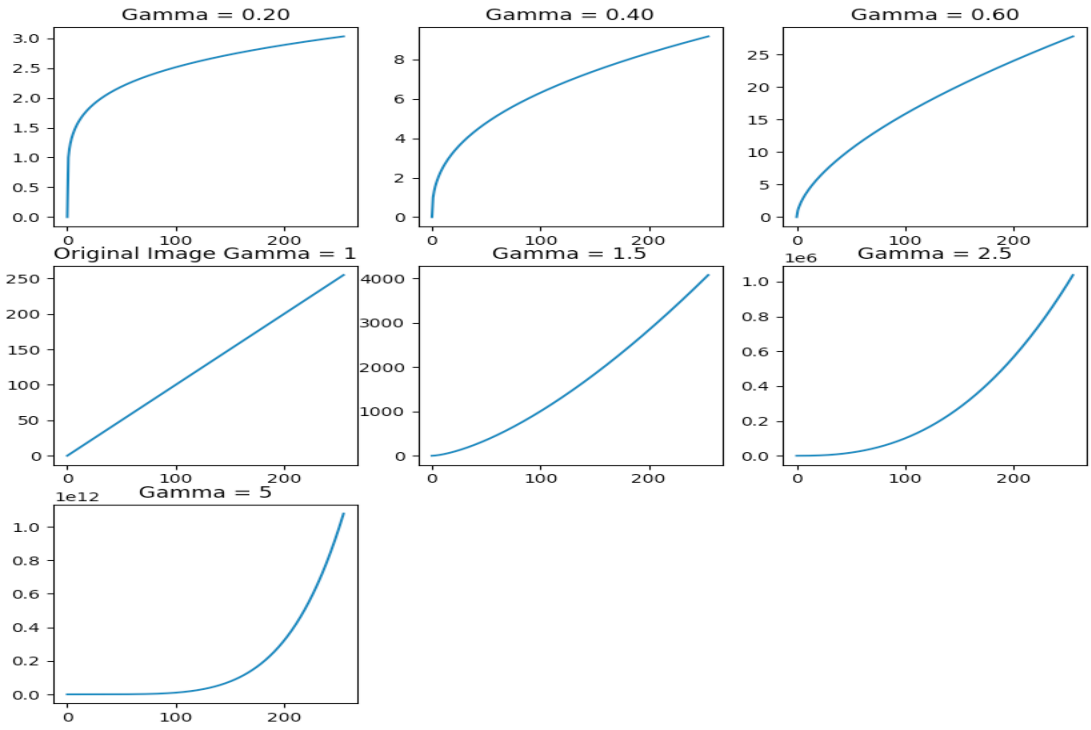
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
img_gray= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gray_20 = img_gray.copy()
img_gray_40 = img_gray.copy()
img_gray_60 = img_gray.copy()
img_gray15 = img_gray.copy()
img_gray25 = img_gray.copy()
img_gray50 = img_gray.copy()

height = img_gray.shape[0]
```

```
width = img_gray.shape[1]

for i in range(0, height):
    for j in range(0, width):
        img_gray_20[i][j] = np.power(img_gray[i][j], 0.20)
        img_gray_40[i][j] = np.power(img_gray[i][j], 0.40)
        img_gray_60[i][j] = np.power(img_gray[i][j], 0.60)
        img_gray15[i][j] = np.power(img_gray[i][j], 1.5)
        img_gray25[i][j] = np.power(img_gray[i][j], 2.5)
        img_gray50[i][j] = np.power(img_gray[i][j], 5)

plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1)
plt.imshow(img_gray_20, cmap='gray')
plt.title('Gamma = 0.20')
plt.subplot(3, 3, 2)
plt.imshow(img_gray_40, cmap='gray')
plt.title('Gamma = 0.40')
plt.subplot(3, 3, 3)
plt.imshow(img_gray_60, cmap='gray')
plt.title('Gamma = 0.60')
plt.subplot(3, 3, 4)
plt.imshow(img_gray, cmap='gray')
plt.title('Original Image Gamma = 1')
plt.subplot(3, 3, 5)
plt.imshow(img_gray15, cmap='gray')
plt.title('Gamma = 1.5')
plt.subplot(3, 3, 6)
plt.imshow(img_gray25, cmap='gray')
plt.title('Gamma = 2.5')
plt.subplot(3, 3, 7)
plt.imshow(img_gray50, cmap='gray')
plt.title('Gamma = 5')

# plot power-law plotting
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
r = np.unique(img_gray)

s= np.zeros(len(r))
s_20 = np.zeros(len(r))
s_40 = np.zeros(len(r))
s_60 = np.zeros(len(r))
s_15 = np.zeros(len(r))
s_25 = np.zeros(len(r))
s_50 = np.zeros(len(r))

for i in range(0, len(r)):
    s[i] = np.power(r[i], 1)
    s_20[i] = np.power(r[i], 0.20)
    s_40[i] = np.power(r[i], 0.40)
    s_60[i] = np.power(r[i], 0.60)
```

```
    s_15[i] = np.power(r[i], 1.5)
    s_25[i] = np.power(r[i], 2.5)
    s_50[i] = np.power(r[i], 5)

plt.figure(figsize=(10, 10))
plt.subplot(3, 3, 1)
plt.plot(r, s_20)
plt.title('Gamma = 0.20')
plt.subplot(3, 3, 2)
plt.plot(r, s_40)
plt.title('Gamma = 0.40')
plt.subplot(3, 3, 3)
plt.plot(r, s_60)
plt.title('Gamma = 0.60')
plt.subplot(3, 3, 4)
plt.plot(r, s)
plt.title('Original Image Gamma = 1')
plt.subplot(3, 3, 5)
plt.plot(r, s_15)
plt.title('Gamma = 1.5')
plt.subplot(3, 3, 6)
plt.plot(r, s_25)
plt.title('Gamma = 2.5')
plt.subplot(3, 3, 7)
plt.plot(r, s_50)
plt.title('Gamma = 5')
```

Input/Output:





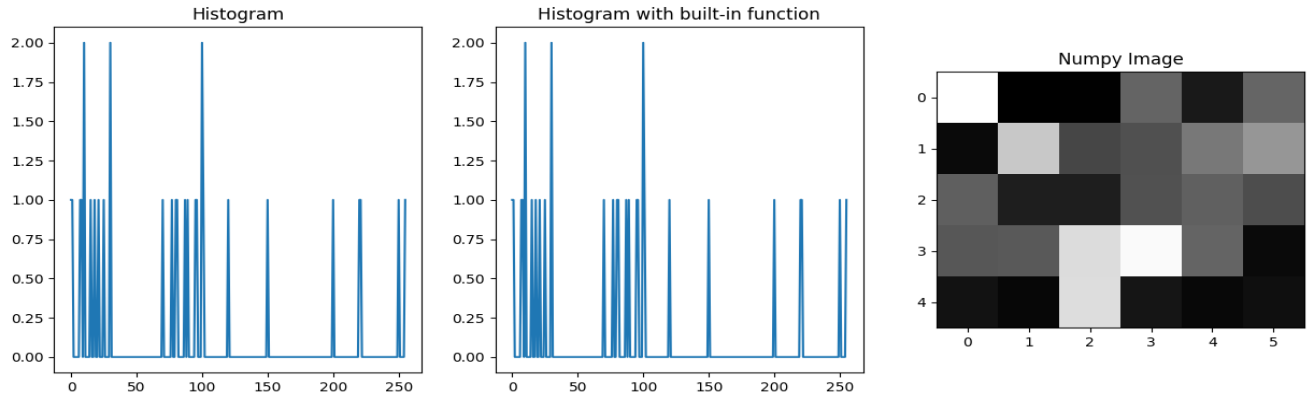
Task-19: Take a 5x6 NumPy array and plot the histogram of the array with proper labels (with and without built in histogram function). Show both histograms side by side in a subplot with a title. Show both input and output NumPy array into an image and show them side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

array = np.array([[255, 0, 1, 100, 25, 101], [10, 200, 70, 80,
 120, 150], [95, 30, 30, 81, 96, 77], [87, 89, 220, 250, 100,
10], [18, 7, 221, 21, 8, 15]])

hist = np.zeros(256)
for i in range(array.shape[0]):
    for j in range(array.shape[1]):
        hist[array[i, j]] += 1

hist2, bins = np.histogram(array, 256, [0, 256])

plt.figure(figsize=(15,5))
plt.subplot(1,3,1)
plt.plot(hist)
plt.title('Histogram')
plt.subplot(1,3,2)
```

```
plt.plot(hist2)
plt.title('Histogram with built-in function')
plt.subplot(1,3,3)
plt.imshow(array, cmap='gray')
plt.title('Numpy Image')
```

Input/Output:



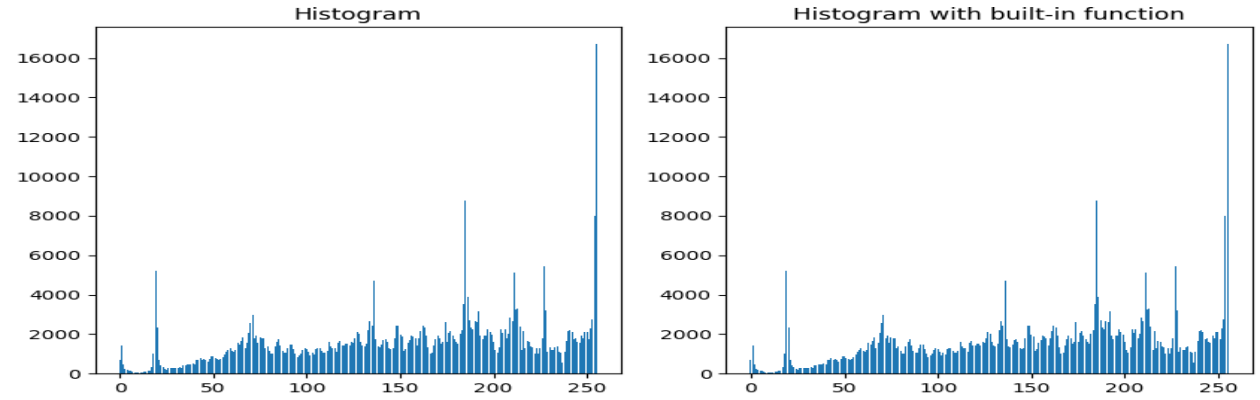
Task-20: Take an 8-bit grayscale image and plot the histogram of the image with proper labels (with and without built in histogram function). Show both histograms side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

array = np.array([[255, 0, 1, 100, 25, 101], [10, 200, 70, 80,
 120, 150], [
                  95, 30, 30, 81, 96, 77], [87, 89, 220, 250, 1
00, 10], [18, 7, 221, 21, 8, 15]])

hist = np.zeros(256)
for i in range(array.shape[0]):
    for j in range(array.shape[1]):
        hist[array[i, j]] += 1

hist2, bins = np.histogram(array, 256, [0, 256])
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
hist = np.zeros(256)
for i in range(img_gray.shape[0]):
    for j in range(img_gray.shape[1]):
        hist[img_gray[i, j]] += 1

hist2, bins = np.histogram(img_gray, 256, [0, 256])
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.bar(range(256), hist)
```

```
plt.title('Histogram')
plt.subplot(1,2,2)
plt.bar(range(256), hist2)
plt.title('Histogram with built-in function')
```

Input/Output:



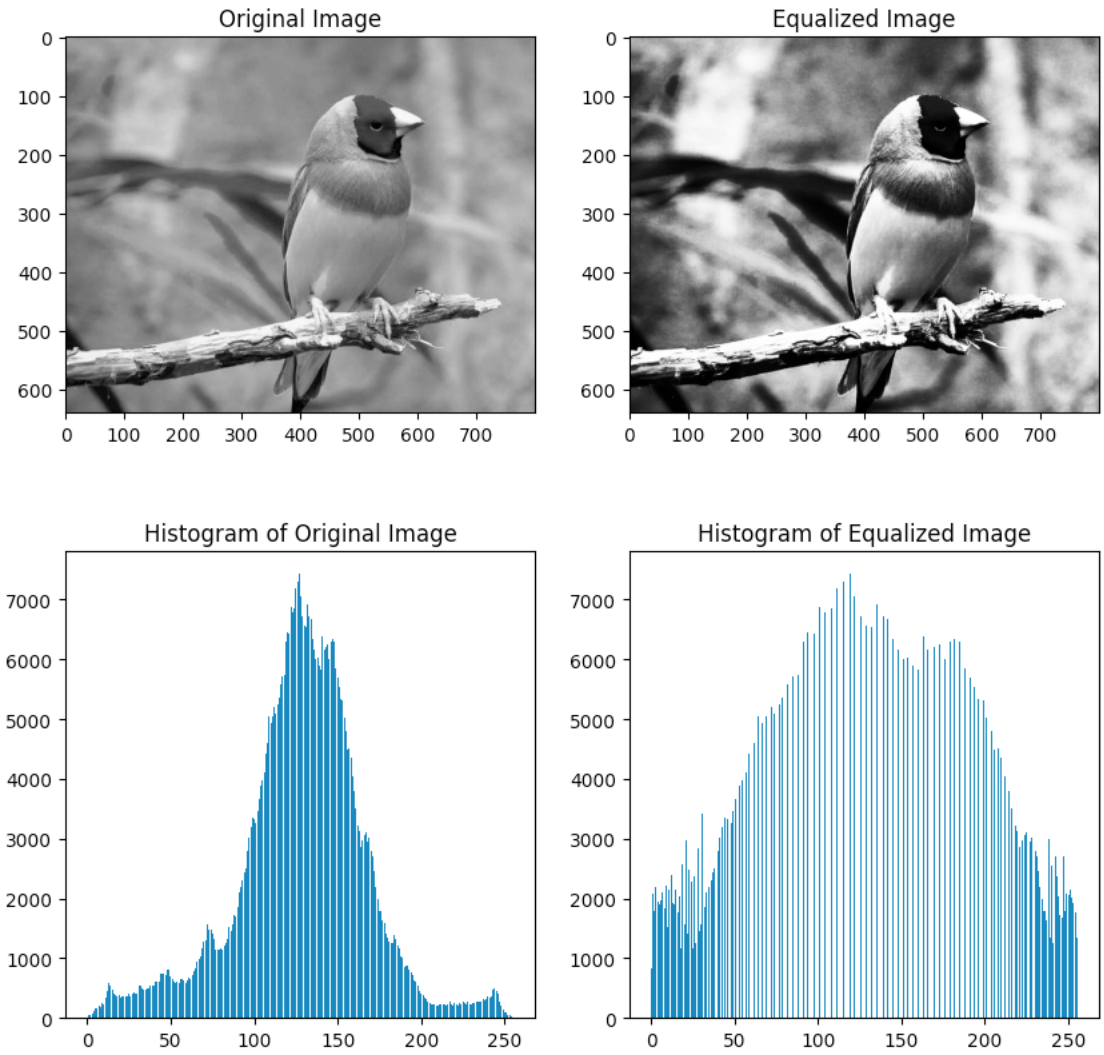
Task-21: Take an 8-bit grayscale image and plot the histogram equalized image as well as its histogram with proper labels (with and without built in histogram function). Show both histograms side by side in a subplot with a title. Show both equalized image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
hist = np.zeros(256)
for i in range(img_gray.shape[0]):
    for j in range(img_gray.shape[1]):
        hist[img_gray[i, j]] += 1

# Histogram equalization
cdf = np.zeros(256)
cdf[0] = hist[0]
for i in range(1, 256):
    cdf[i] = cdf[i-1] + hist[i]

cdf = cdf * 255 / cdf[255]

img_eq = np.zeros(img_gray.shape)
for i in range(img_gray.shape[0]):
    for j in range(img_gray.shape[1]):
        img_eq[i, j] = cdf[img_gray[i, j]]

#histogram of the equalized image
hist1 = np.zeros(256)
for i in range(img_eq.shape[0]):
```

```
    for j in range(img_eq.shape[1]):
        pixel = round(img_eq[i, j])
        hist1[pixel] += 1

plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.imshow(img_gray, cmap='gray')
plt.title('Original Image')
plt.subplot(2,2,2)
plt.imshow(img_eq, cmap='gray')
plt.title('Equalized Image')
plt.subplot(2,2,3)
plt.bar(range(256), hist)
plt.title('Histogram of Original Image')
plt.subplot(2,2,4)
plt.bar(range(256), hist1)
plt.title('Histogram of Equalized Image')
```

Input/Output:



Task-22: Take a 6x6 NumPy array apply convolution operation of the array. Use zero padding and consider a 3x3 mask/kernel for convolution (with and without built in function). Show both input and output array. Show both input and output NumPy array into an image and show them side by side in a subplot with a title.

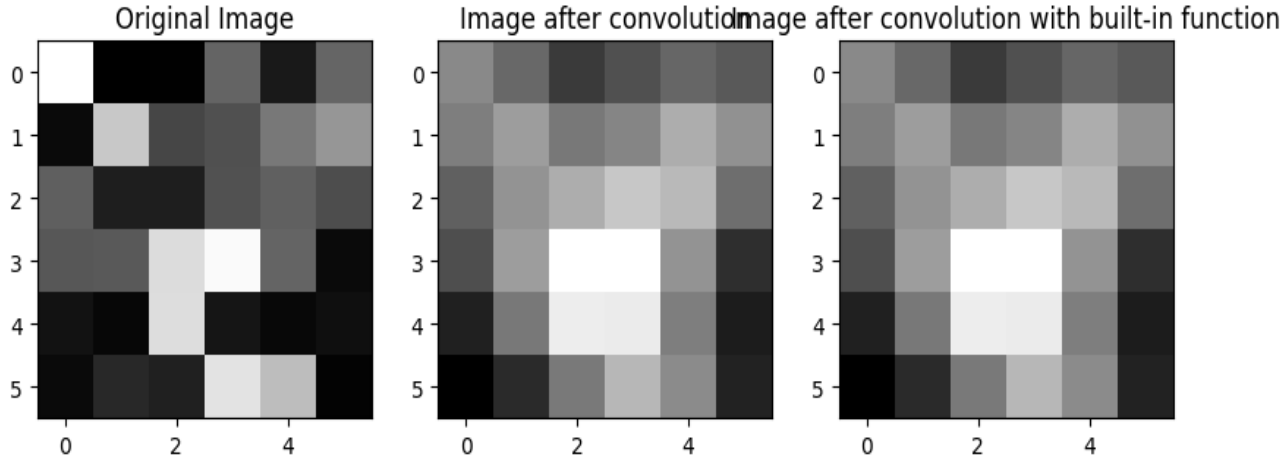
Code:

```
arr = np.array([[255, 0, 1, 100, 25, 101], [10, 200, 70, 80, 1
20, 150], [95, 30, 30, 81, 96, 77], [87, 89, 220, 250, 100, 10
], [18, 7, 221, 21, 8, 15],[10,40,33,227,189,3]],np.uint8)

temp = np.array([[255, 0, 1, 100, 25, 101], [10, 200, 70, 80,
120, 150], [95, 30, 30, 81, 96, 77], [87, 89, 220, 250, 100, 1
0], [18, 7, 221, 21, 8, 15],[10,40,33,227,189,3]],np.uint8)

#use zero padding on array
arr = np.pad(arr, 1, 'constant', constant_values=0)

#use 3x3 gaussian filter manually
filter = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])/16
ans=arr.copy()
for i in range(1, arr.shape[0]-1):
    for j in range(1, arr.shape[1]-1):
        ans[i, j] = round(np.sum(arr[i-1:i+2, j-
1:j+2]*filter))
#remove zero padding
ans = ans[1:ans.shape[0]-1, 1:ans.shape[1]-1]
print(ans)
print()

#using built-in function
conv2 = cv2.filter2D(arr, -1, filter)
#remove zero padding conv2
conv2 = conv2[1:conv2.shape[0]-1, 1:conv2.shape[1]-1]
print(conv2)
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(temp, cmap='gray')
plt.title('Original Image')
plt.subplot(1,3,2)
plt.imshow(ans, cmap='gray')
plt.title('Image after convolution')
plt.subplot(1,3,3)
plt.imshow(conv2, cmap='gray')
plt.title('Image after convolution with built-in function')
```

Input/Output:

```
[[ 78  62  39  50  61  55]
 [ 73  88  70  76  96  82]
 [ 58  83  96 109 102  65]
 [ 49  88 137 137  83  33]
 [ 26  70 128 127  73  24]
 [ 10  31  71 101  79  27]]
```

```
[[ 78  62  39  50  61  55]
 [ 73  88  70  76  96  82]
 [ 58  83  96 109 102  65]
 [ 49  88 137 137  83  33]
 [ 26  70 128 127  73  24]
 [ 10  31  71 101  79  27]]
```



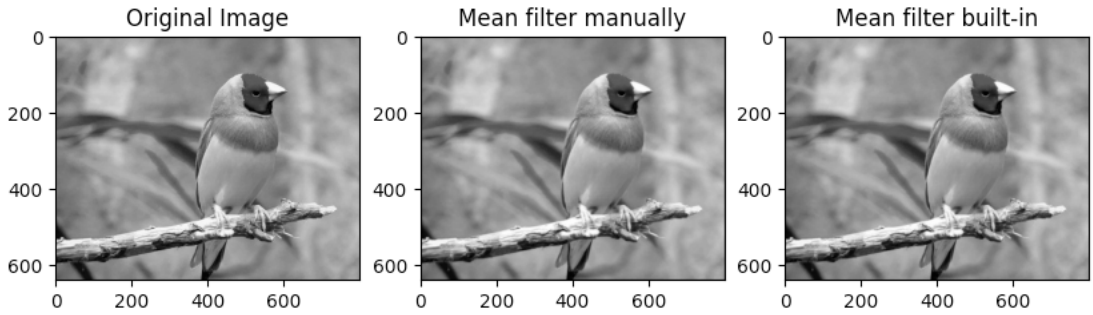
Task-23: Take an 8-bit grayscale image and apply smoothing operation by using mean filter (with and without built in image smoothing function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#apply mean filter with 3x3 kernel size manually
filter = np.ones((3,3))/9
ans = img_gray.copy()
#use zero padding on ans
ans = np.pad(ans, 1, 'constant', constant_values=0)
for i in range(1, ans.shape[0]-1):
    for j in range(1, ans.shape[1]-1):
        ans[i, j] = round(np.sum(ans[i-1:i+2, j-
1:j+2]*filter))
#remove zero padding
ans = ans[1:ans.shape[0]-1, 1:ans.shape[1]-1]

#apply mean filter with 3x3 kernel size with built-in function
conv2 = cv2.filter2D(img_gray, -1, filter)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(img_gray, cmap='gray')
```

```
plt.title('Original Image')
plt.subplot(1,3,2)
plt.imshow(ans, cmap='gray')
plt.title('Mean filter manually')
plt.subplot(1,3,3)
plt.imshow(conv2, cmap='gray')
plt.title('Mean filter built-in')
```

Input/Output:



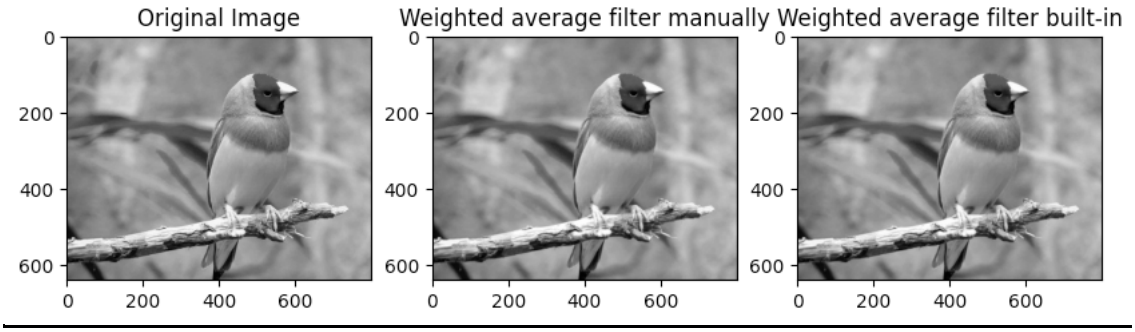
Task-24: Take an 8-bit grayscale image and apply smoothing operation by using weighted averaging filter (with and without built in image smoothing function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#apply weighted average filter with 3x3 kernel size manually
filter = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])/16
ans = img_gray.copy()
#use zero padding on ans
ans = np.pad(ans, 1, 'constant', constant_values=0)
for i in range(1, ans.shape[0]-1):
    for j in range(1, ans.shape[1]-1):
        ans[i, j] = round(np.sum(ans[i-1:i+2, j-
1:j+2]*filter))
#remove zero padding
ans = ans[1:ans.shape[0]-1, 1:ans.shape[1]-1]

#apply weighted average filter with 3x3 kernel size with built
-in function
conv2 = cv2.filter2D(img_gray, -1, filter)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(img_gray, cmap='gray')
```

```
plt.title('Original Image')
plt.subplot(1,3,2)
plt.imshow(ans, cmap='gray')
plt.title('Weighted average filter manually')
plt.subplot(1,3,3)
plt.imshow(conv2, cmap='gray')
plt.title('Weighted average filter built-in')
```

Input/Output:



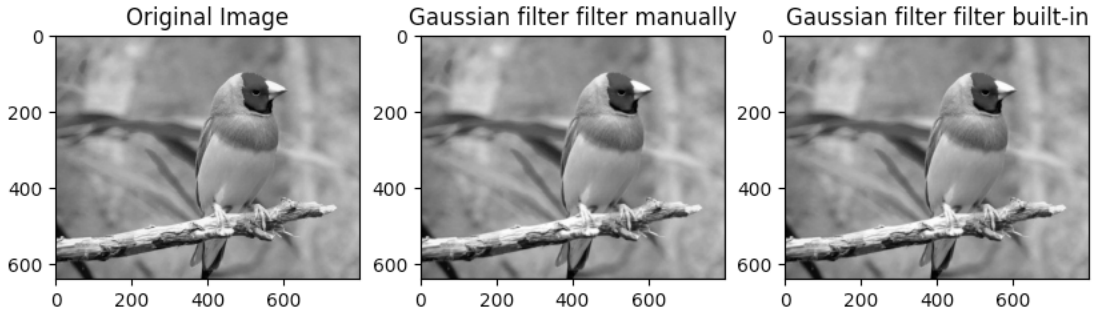
Task-25: Take an 8-bit grayscale image and apply smoothing operation by using gaussian filter (with and without built in image smoothing function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#apply gaussian filter filter with 3x3 kernel size manually
filter = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])/16
ans = img_gray.copy()
#use zero padding on ans
ans = np.pad(ans, 1, 'constant', constant_values=0)
for i in range(1, ans.shape[0]-1):
    for j in range(1, ans.shape[1]-1):
        ans[i, j] = round(np.sum(ans[i-1:i+2, j-
1:j+2]*filter))
#remove zero padding
ans = ans[1:ans.shape[0]-1, 1:ans.shape[1]-1]

#apply gaussian filter filter with 3x3 kernel size with built-
in function
conv2 = cv2.filter2D(img_gray, -1, filter)

plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(img_gray, cmap='gray')
plt.title('Original Image')
```

```
plt.subplot(1,3,2)
plt.imshow(ans, cmap='gray')
plt.title('Gaussian filter filter manually')
plt.subplot(1,3,3)
plt.imshow(conv2, cmap='gray')
plt.title('Gaussian filter filter built-in')
```

Input/Output:



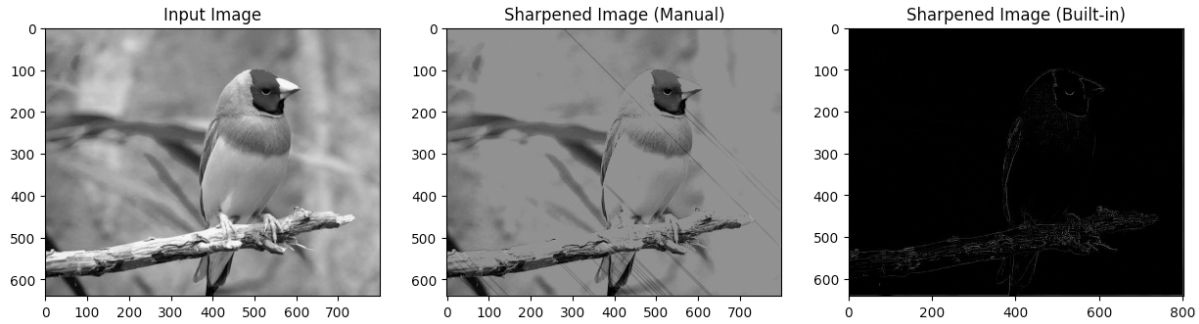
Task-26: Take an 8-bit grayscale image and apply sharpening operation by using laplacian filter (with and without built in image sharpening function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the input image
img = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)

# Define the composite Laplacian filter
filter = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
# Apply the composite Laplacian filter manually
ans = img.copy()
ans = np.pad(ans, 1, 'constant', constant_values=0)
for i in range(1, ans.shape[0]-1):
    for j in range(1, ans.shape[1]-1):
        ans[i, j] = np.clip(round(np.sum(ans[i-1:i+2, j-
1:j+2]*filter)), 0, 255)
ans = ans[1:ans.shape[0]-1, 1:ans.shape[1]-1]

# Apply the composite Laplacian filter using built-in function
img =np.pad(img, 1, 'constant', constant_values=0)
conv2 = cv2.filter2D(img, -1, filter)
img = img[1:img.shape[0]-1, 1:img.shape[1]-1]
# Display the input and output images side by side
plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
plt.imshow(img, cmap='gray')
plt.title('Input Image')
plt.subplot(1,3,2)
plt.imshow(ans, cmap='gray')
```

```
plt.title('Sharpened Image (Manual)')
plt.subplot(1,3,3)
plt.imshow(conv2, cmap='gray')
plt.title('Sharpened Image (Built-in)')
plt.show()
```

Input/Output:

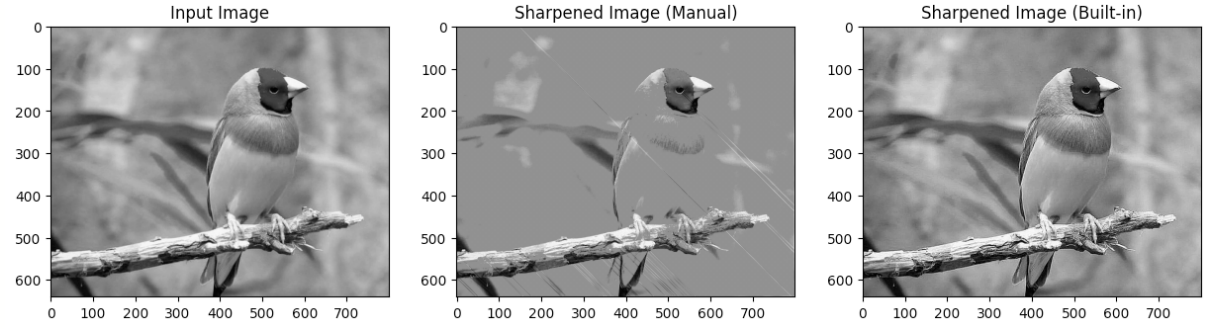


Task-27: Take an 8-bit grayscale image and apply sharpening operation by using composite laplacian filter (with and without built in image sharpening function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the input image
img = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)
# Define the composite Laplacian filter
filter = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
# Apply the composite Laplacian filter manually
ans = img.copy()
ans = np.pad(ans, 1, 'constant', constant_values=0)
for i in range(1, ans.shape[0]-1):
    for j in range(1, ans.shape[1]-1):
        ans[i, j] = np.clip(round(np.sum(ans[i-1:i+2, j-
1:j+2]*filter)), 0, 255)
ans = ans[1:ans.shape[0]-1, 1:ans.shape[1]-1]
# Apply the composite Laplacian filter using built-in function
conv2 = cv2.filter2D(img, -1, filter)
# Display the input and output images side by side
plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
plt.imshow(img, cmap='gray')
plt.title('Input Image')
plt.subplot(1,3,2)
plt.imshow(ans, cmap='gray')
plt.title('Sharpened Image (Manual)')
plt.subplot(1,3,3)
plt.imshow(conv2, cmap='gray')
plt.title('Sharpened Image (Built-in)')
plt.show()
```

Input/Output:

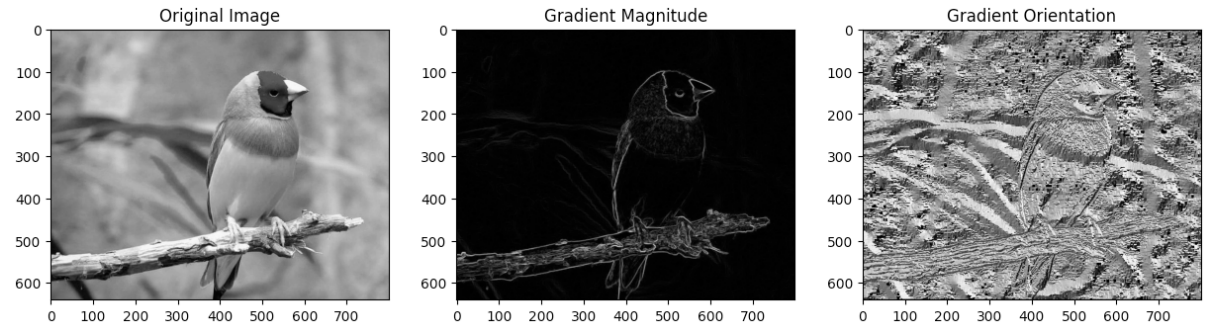


Task-28: Take an 8-bit grayscale image and calculate the gradient of image as well as gradient orientation and gradient magnitude.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# read the image in grayscale format
img_gray = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)
# calculate the x and y gradients using Sobel operator
sobelx = cv2.Sobel(img_gray, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(img_gray, cv2.CV_64F, 0, 1, ksize=3)
# calculate gradient magnitude and orientation
mag, angle = cv2.cartToPolar(sobelx, sobely, angleInDegrees=Tr
ue)
plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
plt.imshow(img_gray, cmap='gray')
plt.title('Original Image')
plt.subplot(1,3,2)
plt.imshow(mag, cmap='gray')
plt.title('Gradient Magnitude')
plt.subplot(1,3,3)
plt.imshow(angle, cmap='gray')
plt.title('Gradient Orientation')
plt.show()
```

Input/Output:

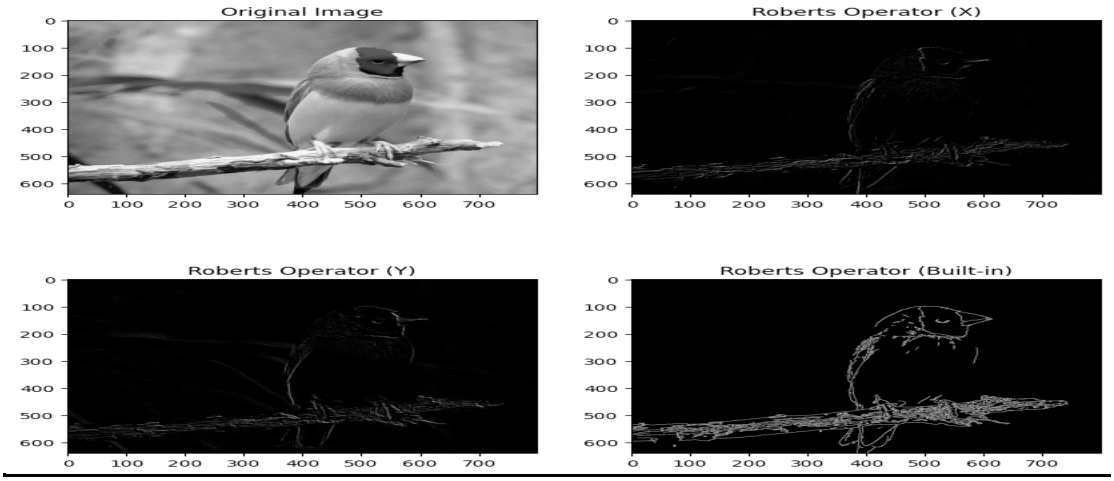


Task-29: Take an 8-bit grayscale image and apply edge detection by using roberts operator (with and without built in function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# read the grayscale image
img = cv2.imread('photo.jpg', 0
# create the Roberts operator kernels
roberts_x = np.array([[1, 0], [0, -1]])
roberts_y = np.array([[0, 1], [-1, 0]])
# apply the Roberts operator using convolution
roberts_x_img = cv2.filter2D(img, -1, roberts_x)
roberts_y_img = cv2.filter2D(img, -1, roberts_y)
# calculate the magnitude and orientation of the gradient
gradient_magnitude = np.sqrt(np.square(roberts_x_img) + np.squ
are(roberts_y_img))
gradient_orientation = np.arctan2(roberts_y_img, roberts_x_img
)
# apply the Roberts operator using built-in function
roberts_img = cv2.Canny(img, 100, 200)
# plot the images side by side
plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.subplot(2,2,2)
plt.imshow(roberts_x_img, cmap='gray')
plt.title('Roberts Operator (X)')
plt.subplot(2,2,3)
plt.imshow(roberts_y_img, cmap='gray')
plt.title('Roberts Operator (Y)')
plt.subplot(2,2,4)
plt.imshow(roberts_img, cmap='gray')
plt.title('Roberts Operator (Built-in)')
plt.show()
```

Input/Output:

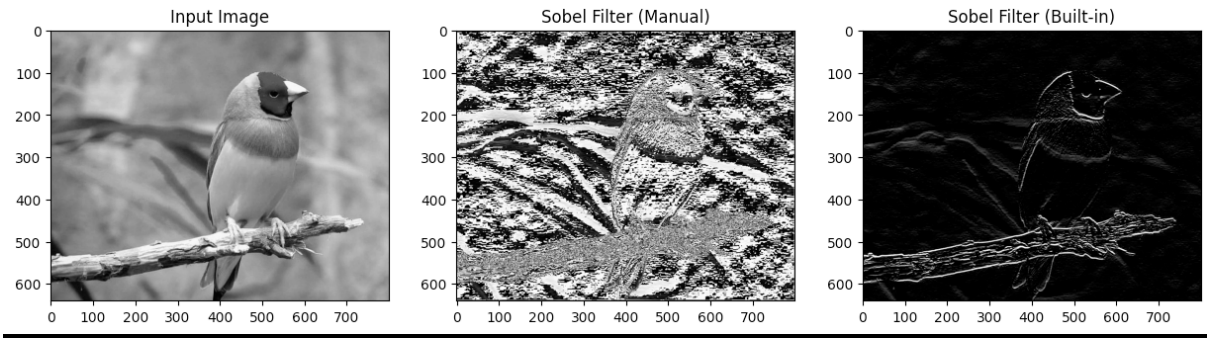


Task-30: Take an 8-bit grayscale image and apply edge detection by using sobel operator (with and without built in function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read in the image in grayscale
img = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)
sobel = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
sobel_out = img.copy()
sobel_out = np.pad(sobel_out,((1,1),(1,1)),'constant')
for i in range(sobel_out.shape[0]-2):
    for j in range(sobel_out.shape[1]-2):
        sobel_out[i,j] = np.sum(sobel*sobel_out[i:i+3,j:j+3])

sobel_out = sobel_out[1:-1,1:-1]

#using built in function
sobel_out2 = cv2.filter2D(img,-1,sobel)

plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
plt.imshow(img, cmap='gray')
plt.title('Input Image')
plt.subplot(1,3,2)
plt.imshow(sobel_out, cmap='gray')
plt.title('Sobel Filter (Manual)')
plt.subplot(1,3,3)
plt.imshow(sobel_out2, cmap='gray')
plt.title('Sobel Filter (Built-in)')
plt.show()
```

Input/Output:

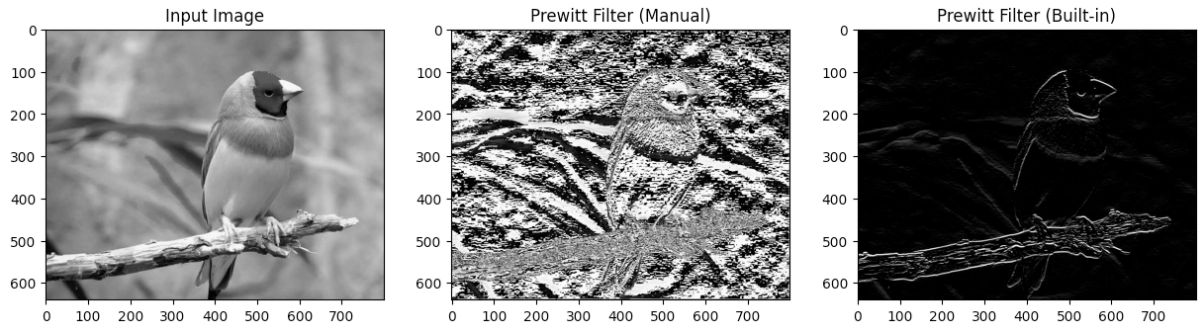


Task-31: Take an 8-bit grayscale image and apply edge detection by using prewitt operator (with and without built in function). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read in the image in grayscale
img = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)
prewitt = np.array([[-1,-1,-1],[0,0,0],[1,1,1]])

prewitt_out = img.copy()
prewitt_out = np.pad(prewitt_out,((1,1),(1,1)),'constant')
for i in range(prewitt_out.shape[0]-2):
    for j in range(prewitt_out.shape[1]-2):
        prewitt_out[i,j] = np.sum(prewitt*prewitt_out[i:i+3,j:
j+3])

prewitt_out = prewitt_out[1:-1,1:-1]

#using built in function
prewitt_out2 = cv2.filter2D(img,-1,prewitt)

plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
plt.imshow(img, cmap='gray')
plt.title('Input Image')
plt.subplot(1,3,2)
plt.imshow(prewitt_out, cmap='gray')
plt.title('Prewitt Filter (Manual)')
plt.subplot(1,3,3)
plt.imshow(prewitt_out2, cmap='gray')
plt.title('Prewitt Filter (Built-in)')
plt.show()
```

Input/Output:

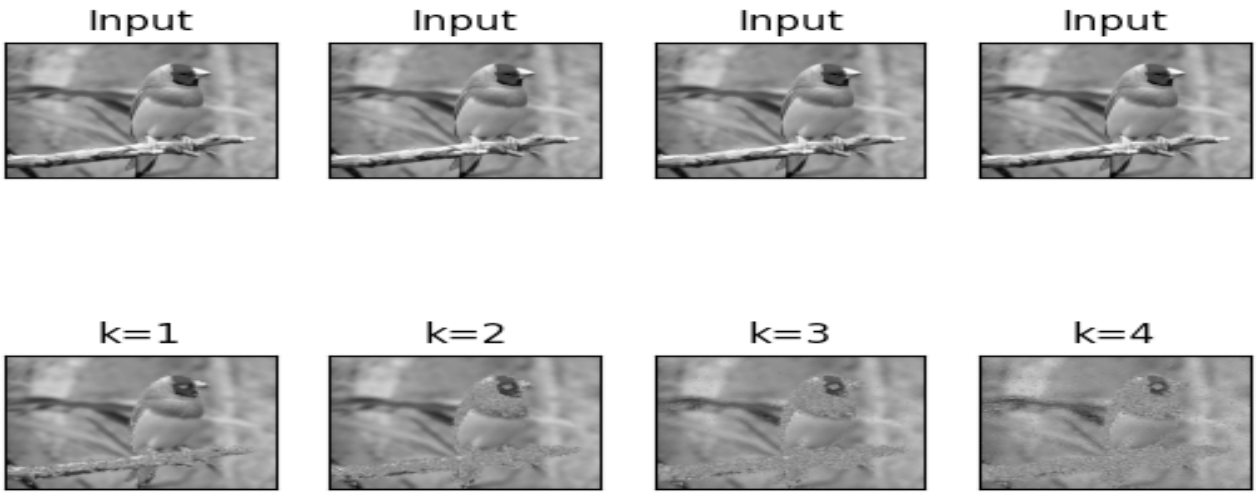


Task-32: Take an 8-bit grayscale image and apply sharpening operation by using unsharp masking and high boost filtering (with k = 1, 2, 3, 4). Show both input and output image side by side in a subplot with a title.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the grayscale image
img = cv2.imread('photo.jpg', cv2.IMREAD_GRAYSCALE)
# Convert to floating-point format and normalize
img = np.float32(img) / 255.0
# Apply unsharp masking
blur = cv2.GaussianBlur(img, (5, 5), 3)
sharpened = cv2.addWeighted(img, 2, blur, -1, 0)
# Apply high boost filtering
k_values = [1, 2, 3, 4]
for i, k in enumerate(k_values):
    # Create high pass filter mask
    mask = cv2.Laplacian(img, cv2.CV_32F, ksize=3)
    # Add scaled mask to original image
    filtered = cv2.add(img, k * mask)
    # Convert back to 8-bit grayscale range
    filtered = np.uint8(filtered * 255.0)
    # Display input and output images side by side
    plt.subplot(2, 4, i+1), plt.imshow(img, cmap='gray')
    plt.title('Input'), plt.xticks([]), plt.yticks([])
    plt.subplot(2, 4, i+5), plt.imshow(filtered, cmap='gray')
    plt.title(f'k={k}'), plt.xticks([]), plt.yticks([])
plt.show()
```

Input/Output:



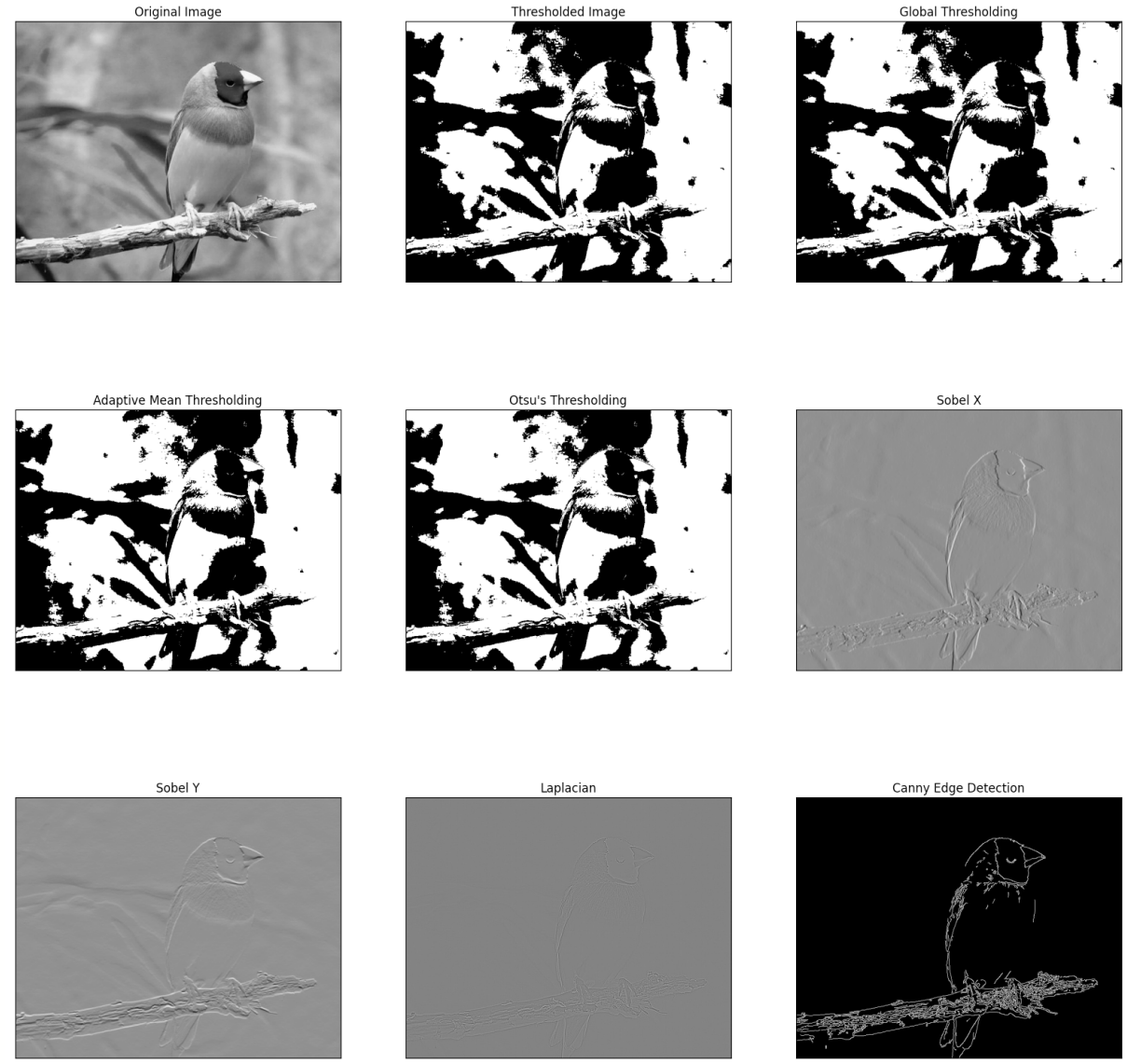
Task-33: Using OpenCV, perform Thresholding, Global Thresholding, Adaptive Thresholding, Otsu Thresholding, Edge Detection (Sobel Edge Detection, Laplacian Edge Detection, Canny Edge Detector), and Hough Line Transformation. Show results in separate plots.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
# Thresholding
threshold_value=127
ret,thresh1=cv2.threshold(img_gray,threshold_value,255,cv2.THR
ESH_BINARY)

#global thresholding
ret1,th1=cv2.threshold(img_gray,127,255,cv2.THRESH_BINARY)
#adaptive thresholding
th2=cv2.adaptiveThreshold(img_gray,255,cv2.ADAPTIVE_THRESH_MEA
N_C,cv2.THRESH_BINARY,11,2)
#Otsu's thresholding
ret2,th2=cv2.threshold(img_gray,0,255,cv2.THRESH_BINARY+cv2.TH
RESH_OTSU)

#sobel edge detection
sobelx=cv2.Sobel(img_gray,cv2.CV_64F,1,0,ksize=5)
sobely=cv2.Sobel(img_gray,cv2.CV_64F,0,1,ksize=5)
#laplacian edge detection
laplacian=cv2.Laplacian(img_gray,cv2.CV_64F)
#Canny edge detection
edges=cv2.Canny(img_gray,100,200)
#hougline transform
minLineLength=100
maxLineGap=10
lines=cv2.HoughLinesP(edges,1,np.pi/180,100,minLineLength,maxL
ineGap)

#plot all the images in single figure
plt.figure(figsize=(20,20))
plt.subplot(331),plt.imshow(img_gray,cmap='gray')
plt.title('Original Image'),plt.xticks([]),plt.yticks([])
plt.subplot(332),plt.imshow(thresh1,cmap='gray')
plt.title('Thresholded Image'),plt.xticks([]),plt.yticks([])
plt.subplot(333),plt.imshow(th1,cmap='gray')
plt.title('Global Thresholding'),plt.xticks([]),plt.yticks([])
plt.subplot(334),plt.imshow(th2,cmap='gray')
plt.title('Adaptive Mean Thresholding'),plt.xticks([]),plt.yti
cks([])
plt.subplot(335),plt.imshow(th2,cmap='gray')
plt.title("Otsu's Thresholding"),plt.xticks([]),plt.yticks([])
plt.subplot(336),plt.imshow(sobelx,cmap='gray')
plt.title('Sobel X'),plt.xticks([]),plt.yticks([])
```

```
plt.subplot(337),plt.imshow(sobely,cmap='gray')
plt.title('Sobel Y'),plt.xticks([]),plt.yticks([])
plt.subplot(338),plt.imshow(laplacian,cmap='gray')
plt.title('Laplacian'),plt.xticks([]),plt.yticks([])
plt.subplot(339),plt.imshow(edges,cmap='gray')
plt.title('Canny Edge Detection'),plt.xticks([]),plt.yticks([]
)
plt.show()
#print huoghline transform
for x1,y1,x2,y2 in lines[0]:
    cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
plt.imshow(img,cmap='gray')
plt.show()
```

Input/Output:

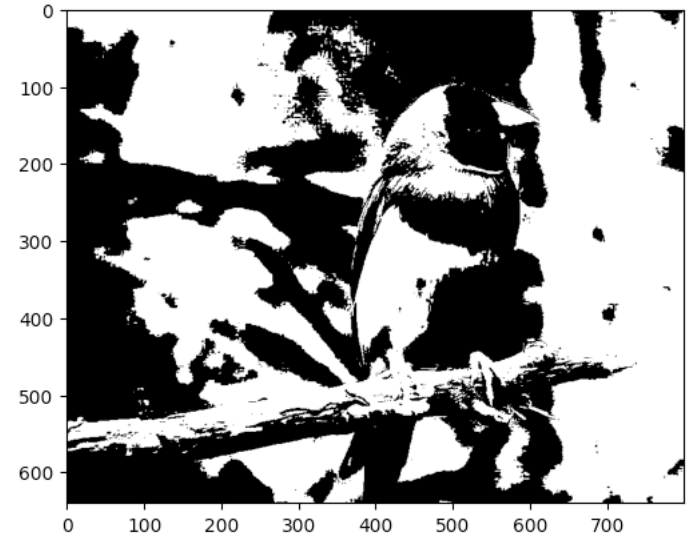


Task-34: Manually perform Thresholding on a gray scale image.

Code:

```
from google.colab import drive
drive.mount('/content/drive')
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')
img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
#mannual thresholding
threshold_value=127
for i in range(img_gray.shape[0]):
    for j in range(img_gray.shape[1]):
        if img_gray[i,j]>threshold_value:
            img_gray[i,j]=255
        else:
            img_gray[i,j]=0
plt.imshow(img_gray,cmap='gray')
```

Input/Output:



Task-35: Manually perform Sobel Edge Detection with first-order derivative in the X-direction and the Y- direction. Show all the resultant figures in a single plot.

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('photo.jpg')

img_gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
# Get user input for kernel size
kernel_size = int(input("Enter kernel size (odd number): "))

# Check if kernel size is odd
if kernel_size % 2 == 0:
    kernel_size += 1

# Generate Sobel filter kernels
gx = np.zeros((kernel_size, kernel_size))
gy = np.zeros((kernel_size, kernel_size))

value = kernel_size // 2
center = value

#for gx kernel
for i in range(kernel_size):
    for j in range(kernel_size):
        if j == center and i != center:
            gx[i, j] = value*2
        else:

            gx[i, j] = value

    value -= 1

# for gy kernel
value = kernel_size // 2
for i in range(kernel_size):
    for j in range(kernel_size):
        if j == center and i != center:
            gy[j, i] = value*2
        else:
            gy[j, i] = value

    value -= 1

#apply sobel filter on image using gx and gy kernel manually
img_sobelx = np.zeros(img_gray.shape)
img_sobely = np.zeros(img_gray.shape)
```

```
for i in range(img_gray.shape[0]-kernel_size):
    for j in range(img_gray.shape[1]-kernel_size):
        img_sobelx[i, j] = np.sum(img_gray[i:i+kernel_size, j:
j+kernel_size] * gx)
        img_sobely[i, j] = np.sum(img_gray[i:i+kernel_size, j:
j+kernel_size] * gy)

#plot all the images in single figure
plt.figure(figsize=(20,20))
plt.subplot(131),plt.imshow(img_gray,cmap='gray')
plt.title('Original Image'),plt.xticks([]),plt.yticks([])
plt.subplot(132),plt.imshow(img_sobelx,cmap='gray')
plt.title('Sobel X'),plt.xticks([]),plt.yticks([])
plt.subplot(133),plt.imshow(img_sobely,cmap='gray')
plt.title('Sobel Y'),plt.xticks([]),plt.yticks([])
plt.show()
```

Input/Output:

Enter kernel size (odd number): 3