**IMAGE PROCESSING INDEX**

|  |  |  |
| --- | --- | --- |
| **CLASS:** BSc CSIT  **ROLL NO**.: 23608 | | **SEMESTER:** V |
| **SR. NO** | **TITLE OF THE EXPERIMENT.** | |
| 1 | Point processing in spatial domain a. Negation of an  image  b. Thresholding of an image  c. Contrast Stretching of an image | |
| 2 | Bit Plane Slicing | |
| 3 | Histogram Equalization | |
| 4 | Histogram Specification | |
| 5 | Zooming by interpolation and replication | |
| 6 | Filtering in spatial domain a. Low Pass  Filtering  b. High Pass Filtering | |
| 7 | Edge Detection | |
| 8 | Data compression using Huffman coding | |

|  |  |
| --- | --- |
| ***Experiment No. 1A*** | ***Negation of an image*** |
| Aim | To study image negative |
| Tool | PYTHON |
| Theory | The negative of an image with gray levels in the range [ 0, L-1] is obtained by using the negative transformation given by the expression  S= L – 1 – r (1)  This is according to the transformation S = T ( r ) In above transformation ( 1 ) , the intensity of the output image decreases as the intensity of the input increases. The type of processing is particularly suited for enhancing white or gray detail embedded in dark regions of an image especially when black areas are dominants in site. |
| Algorithm | 1. Read i/p image  2. Read maximum gray level pixel of i/p image  3. Replace input image by ( maximum – i/p ) = o/p 4. Display o/p image |

**SOURCE CODE:**

import numpy as np

import cv2

img=cv2.imread('kabina.jpg',0)

w,h=img.shape

neg\_img=np.array(img)

for x in range(w):

    for y in range(h):

        neg\_img[x,y]=255-img[x,y]

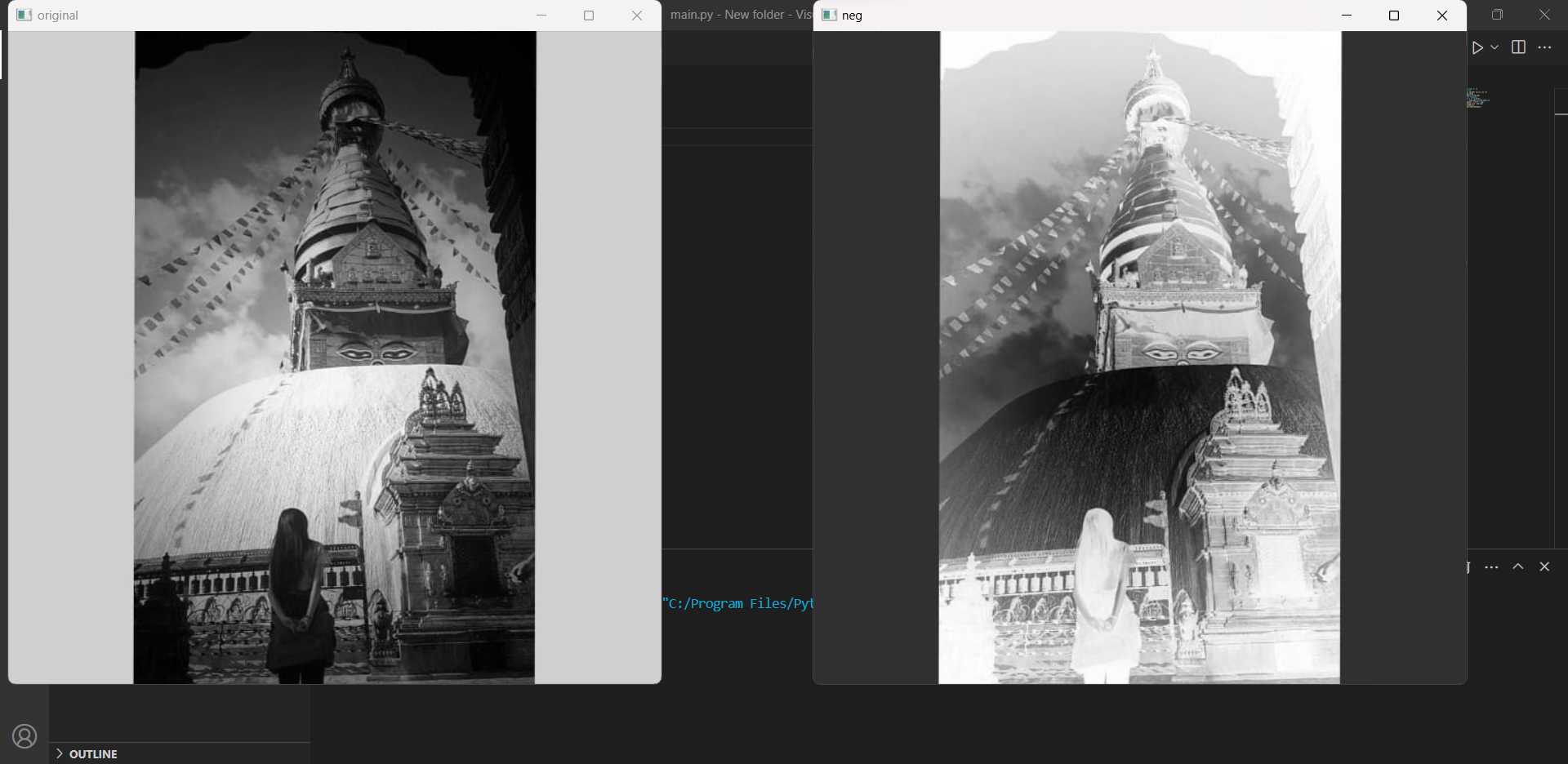
cv2.imshow('original',img)

cv2.imshow('neg',neg\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**

****

**CONCLUSION:**

The negative of the image was obtained in this labwork.

|  |  |
| --- | --- |
| ***Experiment No. 1B*** | ***Thresholding of an Image*** |
| Aim | To study thresholding of the image |
| Tool | PYTHON |
| Theory | Thresholding is a simple  S  process to separate the  interested object from the  background. It gives the  binary image. The formula  L-1  for achieving thresholding  is as follows  s = 0 if r <= t s = L-1 if r > t  t L-1 r |
| Algorithm | 1. Read input image  2. Enter thresholding value t  3. If image pixel is less than t replace it by zero.  4. If image pixel is > t replace it by 255  5. Display input image  6. Display threshold image  7. Write input image  8. Write threshold image |

**SOURCE CODE:**

import cv2

import cv2 as cv

import numpy as np

img=cv.imread('kabina.jpg',0)

th=80

w,h=img.shape

thr\_img=np.array(img)

for x in range (w):

    for y in range (h):

        if thr\_img[x,y]<=th:

            thr\_img[x,y]=0

        else:

            thr\_img[x,y]=255

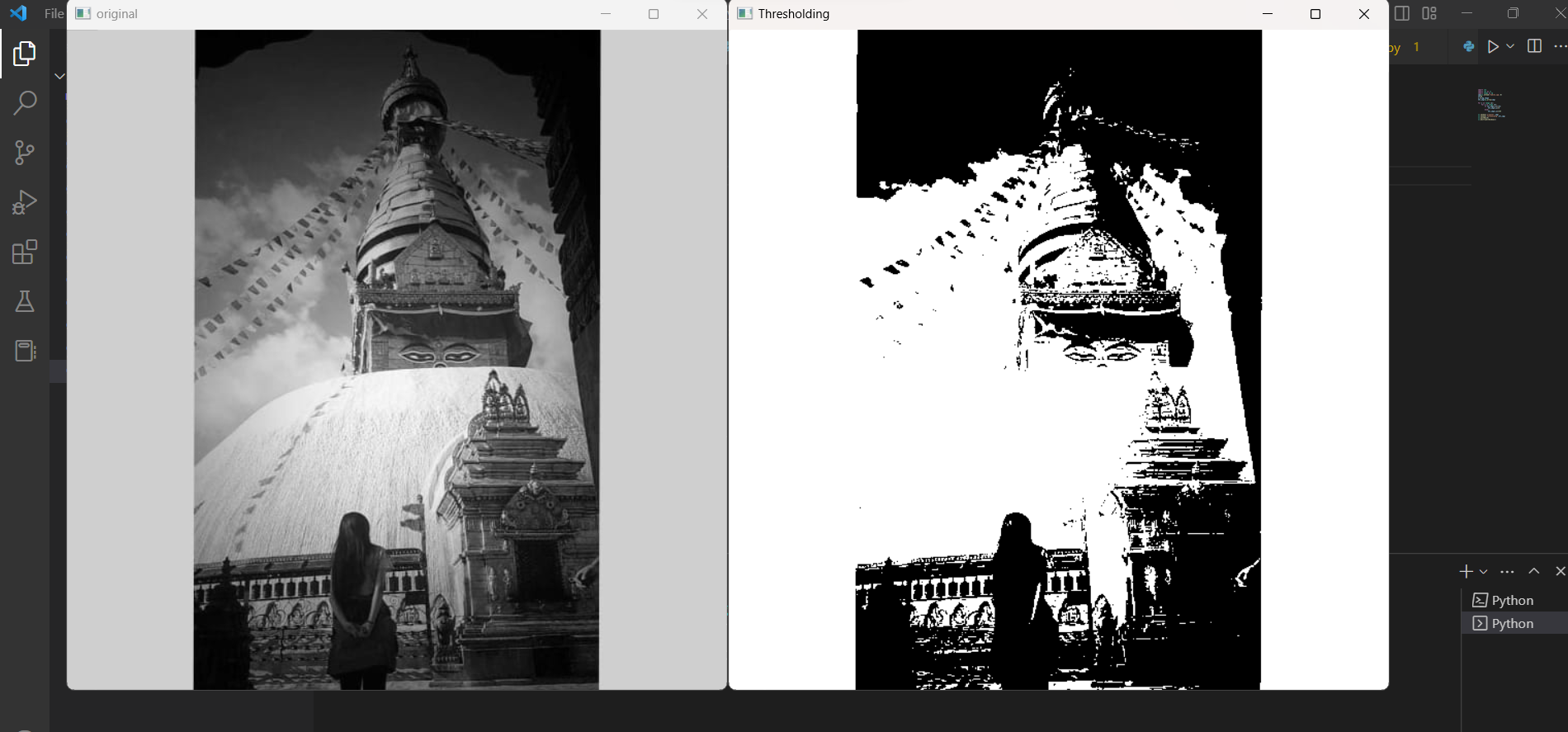
cv.imshow('original',img)

cv.imshow('Thresholding',thr\_img)

cv.waitKey(0)

cv.destroyAllWindows()

**OUTPUT:**

****

**CONCLUSION:**

Thresholding separates out the object from the background which is proved in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 1C*** | ***Contrast Stretching of an Image*** |
| Aim | To study Contrast Stretching of an image |
| Tool | PYTHON |
| Theory | Low contrast images can result from poor illumination, lack of dynamic range in the imaging sensor etc. The idea behind contrast stretching is to increase the dynamic range of the gray levels in the image being processed. The transformation function for contrast stretching is given by r 0 r r1  =  (r-r1)+s1 r1 r r2    (r-r2 )+ s2 r2 r L-1    The location of the points (r1 , s1) & (r2 , s2) control the shape of the transformation function. |
| Algorithm | 1. Read input image  2. Enter values r1,r2,s1,s2  3. Calculate alpha,beta and gamma slopes.  3. if input pixel value is <= r1 then o/p = alpha x input  5. If input pixel is > r1and <=r2 then o/p = beta x (r-r1)+s1 6. otherwise o/p = gamma x (r-r2)+s2  7. Display i/p image  8. Display o/p image. |

**SOURCE CODE:**

import cv2

import numpy as np

# Function to map each intensity level to output intensity level.

def pixelVal(pix, r1, s1, r2, s2):

    if (0 <= pix and pix <= r1):

        return (s1 / r1) \* pix

    elif (r1 < pix and pix <= r2):

        return ((s2 - s1) / (r2 - r1)) \* (pix - r1) + s1

    else:

        return ((255 - s2) / (255 - r2)) \* (pix - r2) + s2

img = cv2.imread('kabina.jpg')

cv2.imshow('original',img)

r1 = 70

s1 = 0

r2 = 140

s2 = 255

# Vectorize the function to apply it to each value in the Numpy array.

pixelVal\_vec = np.vectorize(pixelVal)

contrast\_stretched = pixelVal\_vec(img, r1, s1, r2, s2)

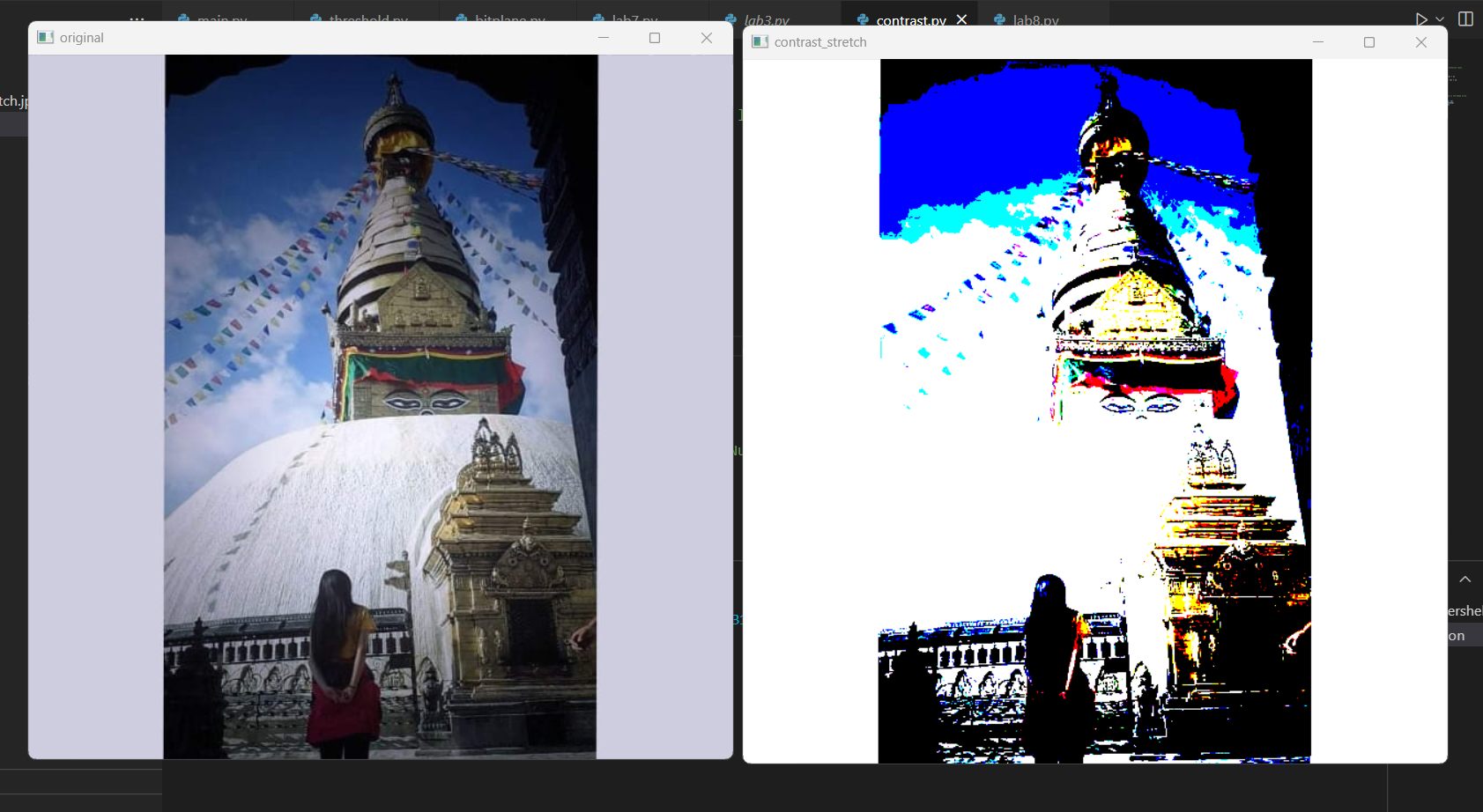
cv2.imwrite('contrast\_stretch.jpg', contrast\_stretched)

cv2.imshow('contrast\_stretch', contrast\_stretched)

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**

****

**CONCLUSION:**

Contrast stretching increases the contrast of the image which is done in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 2*** | ***Bit Plane Slicing*** |
| Aim | To study Bit Plane Slicing |
| Tool | PYTHON |
| Theory | This transformation involves determining the number of usually significant bits in an image. In case of a 8 bit image each pixel is represented by 8 bits. Imagine that the image is composed of eight 1 bit planes ranging from bit plane 0 for the least significant bit to bit plane 7 for the most significant bit. Plane 0 contains all the lowest order bits in the bytes comprising the pixels in the image & plane 7 contains all the high order bits. The higher order bits contain usually significant data and the other bit planes contribute to more subtle details in the iamge. Separating a digital image into its bit planes is useful for analyzing the relative importance played by each bit of the image. |
| Algorithm | 1.Read i/p image  2. Use bitand operation to extract each bit  3. Do the step 2 for every pixel.  4. Display the original image and the biplanes formed by bits extracted |

**SOURCE CODE:**

import numpy as np

import cv2

img=cv2.imread('kabina.jpg',0)

cv2.imshow('original image',img)

cv2.waitKey(0)

lst=[]

for i in range(img.shape[0]):

    for j in range(img.shape[1]):

        lst.append(np.binary\_repr(img[i][j],width=8))

eight\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*128).reshape(img.shape[0],img.shape[1])

seven\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*64).reshape(img.shape[0],img.shape[1])

six\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*32).reshape(img.shape[0],img.shape[1])

five\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*16).reshape(img.shape[0],img.shape[1])

four\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*8).reshape(img.shape[0],img.shape[1])

three\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*4).reshape(img.shape[0],img.shape[1])

two\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*2).reshape(img.shape[0],img.shape[1])

one\_bit\_img=(np.array([int(i[0]) for i in lst],dtype=np.uint8)\*1).reshape(img.shape[0],img.shape[1])

cv2.imshow('bitplane7',cv2.normalize(eight\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imshow('bitplane6',cv2.normalize(seven\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imshow('bitplane5',cv2.normalize(six\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imshow('bitplane4',cv2.normalize(five\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imshow('bitplane3',cv2.normalize(four\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

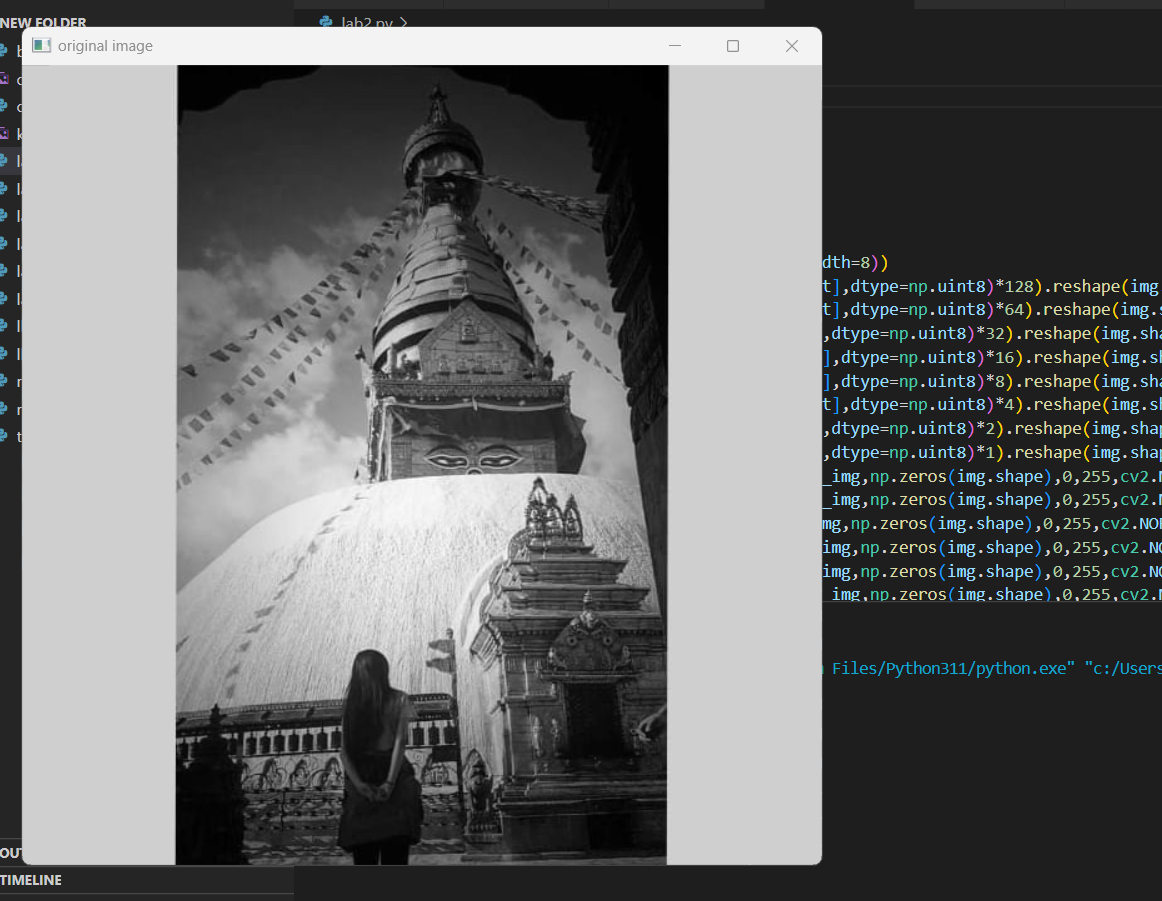
cv2.imshow('bitplane2',cv2.normalize(three\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

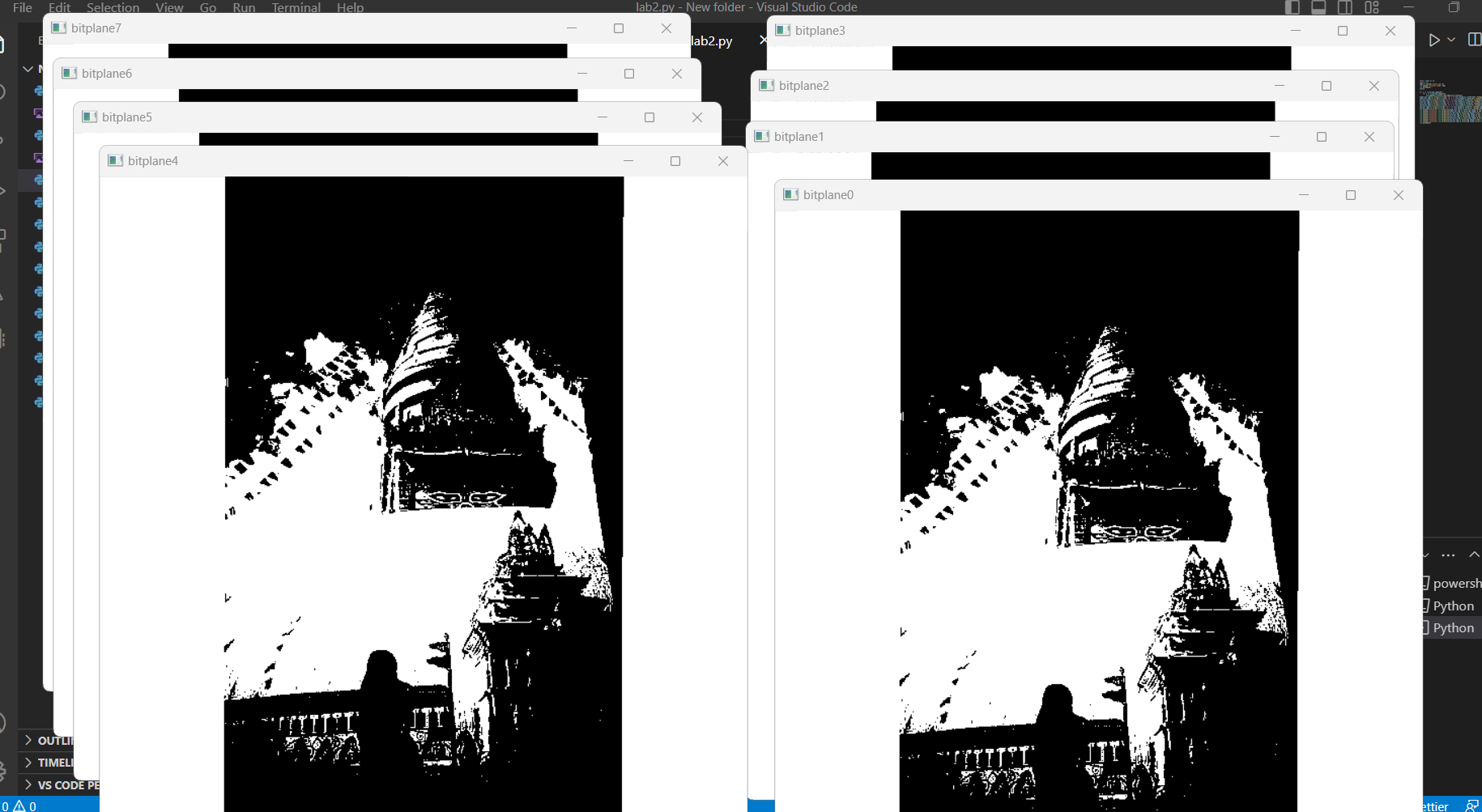
cv2.imshow('bitplane1',cv2.normalize(two\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.imshow('bitplane0',cv2.normalize(one\_bit\_img,np.zeros(img.shape),0,255,cv2.NORM\_MINMAX))

cv2.waitKey(0)

**OUTPUT:**

****

****

**CONCLUSION:**

Higher order bit planes carry maximum visual information which is proved in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 3*** | ***Histogram Equalization*** |
| Aim | To implement histogram equalization. |
| Tool | PYTHON |
| Theory | Histogram of a digital image with gray levels in range [0,L-1] is a discrete function h( k) = nk where k kth gray level and nk = no. of pixels of an image having gray level rk In histogram there are 3 possibilities as follows, 1. For a dark image the components of histogram on the low (dark) side. 2. For a bright image the component are on high ( bright ) side & 3. For an image with low contrast they are in the middle of gray side.  Histogram equalization is done to spread there component uniformly over the gray scale as far as possible.  This is obtained by function Sk = ∑ (limit k to i=0) hi /n; k  = 0,1,2,3,...i-l  Thus processed image is obtained by mapping each pixel with level rk into a corresponding pixel with level sk in o/p image. This transformation is called Histogram equalization |
| Algorithm | 1. Read the i/p image & its size.  2. Obtain the gray level values of each pixel & divide them by total number of gray level values.  3. Implement the function Sk  4. Plot the equalized histogram and original histogram.  5. Display the original and the new image. |

**SOURCE CODE:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread('kabina.jpg',0)

plt.hist(img.ravel(),256,[0,256])

plt.show()

plt.savefig('hist1.png')

equ = cv2.equalizeHist(img)

res = np.hstack((img,equ))

cv2.imshow('Equalized Image',res)

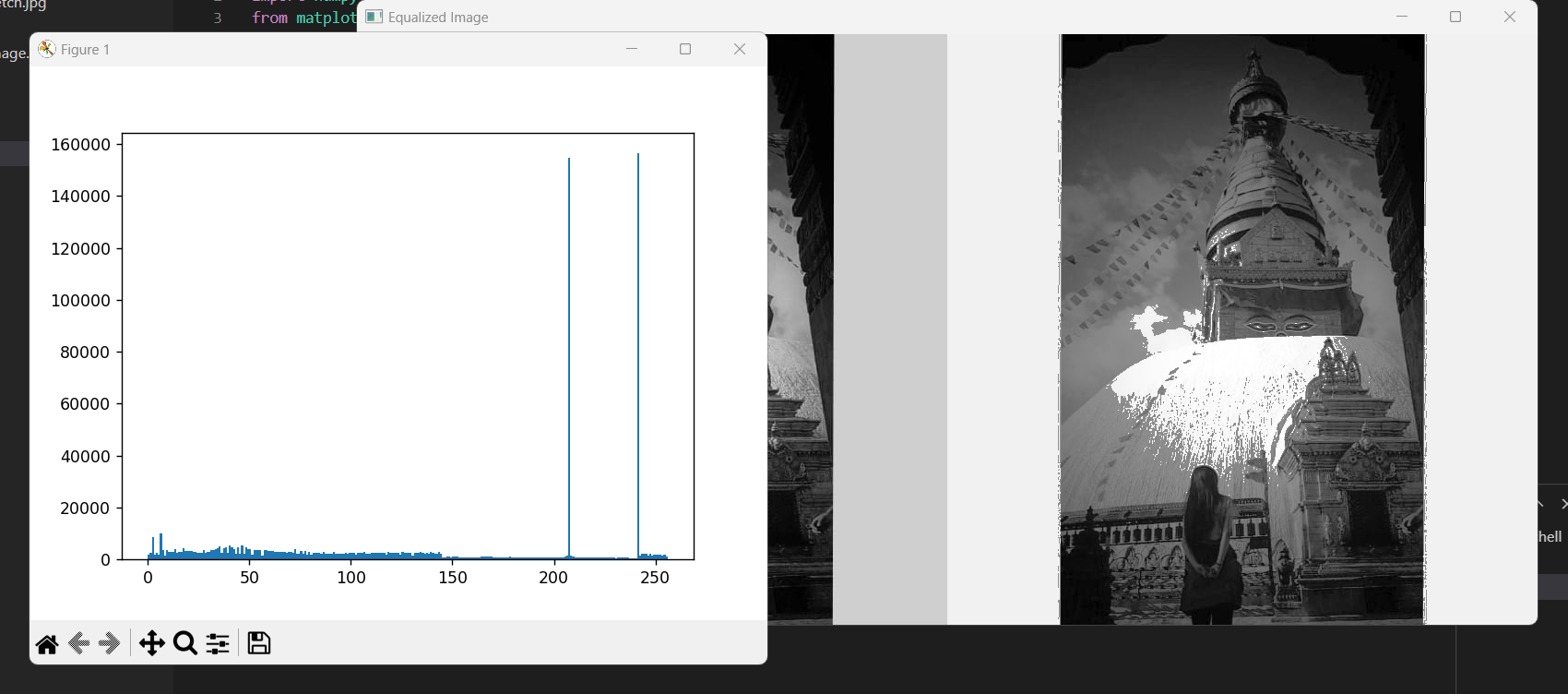
cv2.imwrite('Equalized Image.png',res)

plt.hist(res.ravel(),256,[0,256])

plt.show()

plt.savefig('equal-hist1.png')

**OUTPUT:**

****

**CONCLUSION:**

Digital Histogram enhances image but it does not generate a flat histogram which is proved in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 4*** | ***Histogram Specification*** |
| Aim | To implement histogram specification |
| Tool | PYTHON |
| Theory | Histogram equalization automatically determines a transformation function that seeks to produce an output image that has a uniform histogram. But it is useful sometimes to be able specify the shape of the histogram that we wish the processed image to have. The method used to generate a processed image that has a specified histogram is called histogram specification.  Sk = T(rk)=∑ Pr(rj ) k = 0,1,2,3,...L-l  Vk= G(zk) = ∑ Pz(zj ) k = 0,1,2,3,...L-l  Zk = G-1(T(rk)) k = 0,1,2,3,...L-l  Map each pixel with level rk into a corresponding pixel with level sk . Obtain the transformation function G from a given histogram Pz(z). For any Zq this transformation function yields a corresponding value Vq. We would find the corresponding value Zq from G-1. |
| Algorithm | . Obtain the histogram of the given image.  2. Map each level rk to sk  3. Obtain the transformation function G from the given Pz(z)  4. Calculate zk for each value of sk  5. For each pixel in the original image, if the value of that pixel is rk , map this value to its corresponding level sk ,then map level sk into the final value zk 6. Display the modified image and its histogram |
| Questions | Explain the histogram specification in continuous domain. |

**SOURCE CODE:**

import matplotlib.pyplot as plt

from skimage import exposure

from skimage.exposure import match\_histograms

import cv2

# reading main image

img1 = cv2.imread("kabina.jpg")

# checking the number of channels

print('No of Channel is: ' + str(img1.ndim))

# reading reference image

img2 = cv2.imread("hist1.png")

# checking the number of channels

print('No of Channel is: ' + str(img2.ndim))

image = img1

reference = img2

matched = match\_histograms(image, reference)

fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3,

                                    figsize=(8, 3),

                                    sharex=True, sharey=True)

for aa in (ax1, ax2, ax3):

    aa.set\_axis\_off()

ax1.imshow(image)

ax1.set\_title('Source')

ax2.imshow(reference)

ax2.set\_title('Reference')

ax3.imshow(matched)

ax3.set\_title('Matched')

plt.tight\_layout()

plt.show()

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(8, 8))

for i, img in enumerate((image, reference, matched)):

    for c, c\_color in enumerate(('red', 'green', 'blue')):

        img\_hist, bins = exposure.histogram(img[..., c],  source\_range='dtype')

        axes[c, i].plot(bins, img\_hist / img\_hist.max())

        img\_cdf, bins = exposure.cumulative\_distribution(img[..., c])

        axes[c, i].plot(bins, img\_cdf)

        axes[c, 0].set\_ylabel(c\_color)

axes[0, 0].set\_title('Source')

axes[0, 1].set\_title('Reference')

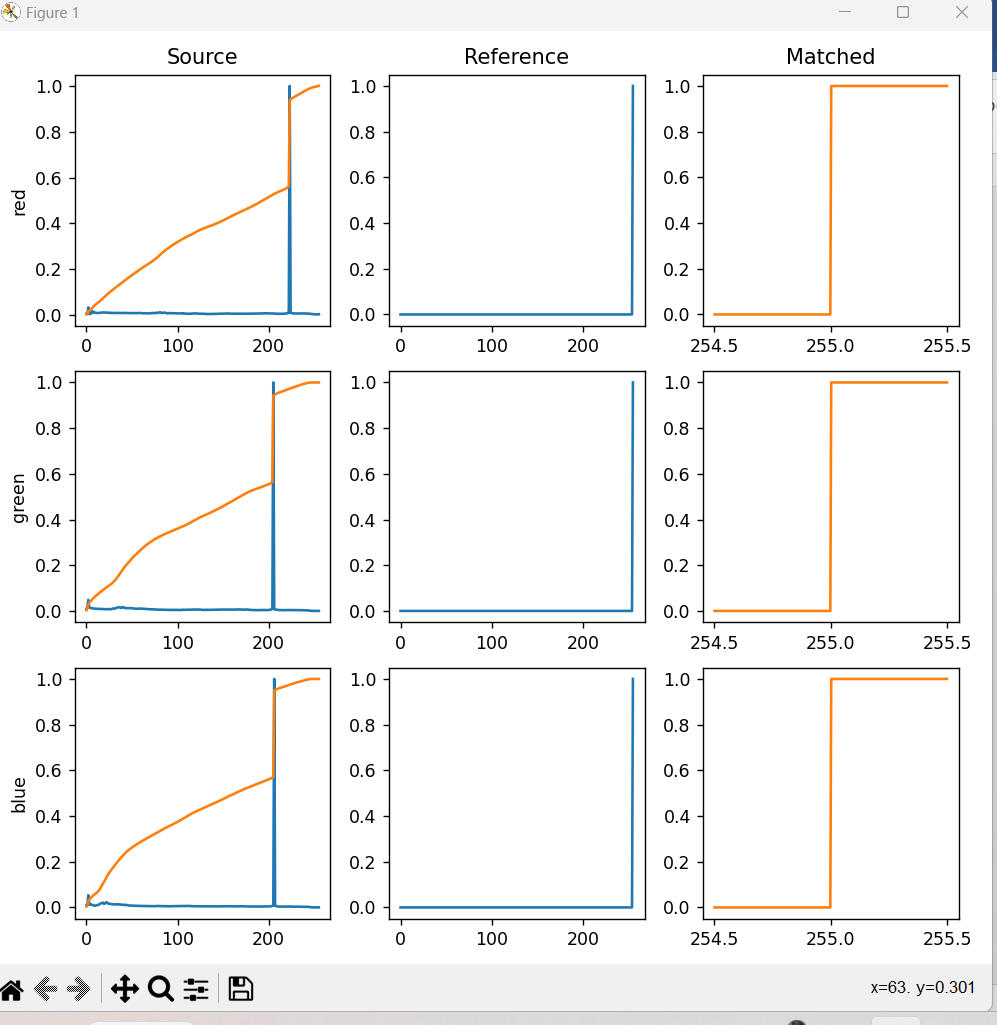
axes[0, 2].set\_title('Matched')

plt.tight\_layout()

plt.show()

**OUTPUT:**

****

****

**CONCLUSION:**

Digital Histogram enhances image but it does not generate a flat histogram which is proved in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 5*** | ***Zooming by interpolation and replication*** |
| Aim | To implement the magnification by replication and interpolation |

|  |  |
| --- | --- |
| Tool | PYTHON |
| Theory | Zooming can be done in two ways.  1)Replication : In replication we simply replicate each pixel and then replicate each row. Hence image of size n x n is zoomed to 2n x2n. Zooming by replication gives the final image a patchy look since clusters of grey levels are formed. This can be substantially reduced by using a better method of zooming known as interpolation.  2) Interpolation : In this method instead of replicating each pixel, average of two adjacent pixels along the rows is taken and placed between two pixels. The same operation is then performed along the columns. The patchiness that was present in the replicated image is much less in the interpolated image. |
| Algorithm | Replication:  1. Read i/p image.  2. Replicate each pixel  3. Replicate each row  4. Display o/p image  Interpolation  1. Read i/p image  2. Average of two adjacent pixels along the rows is taken and placed between two pixels.  3. Do the same along columns  4. Display o/p image. |

**SOURCE CODE:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

image = cv2.imread("kabina.jpg", 1)

# Loading the image

half = cv2.resize(image, (0, 0), fx=0.1, fy=0.1)

bigger = cv2.resize(image, (1050, 1610))

stretch\_near = cv2.resize(image, (780, 540),

                          interpolation=cv2.INTER\_NEAREST)

Titles = ["Original", "Half", "Bigger", "Interpolation Nearest"]

images = [image, half, bigger, stretch\_near]

count = 4

for i in range(count):

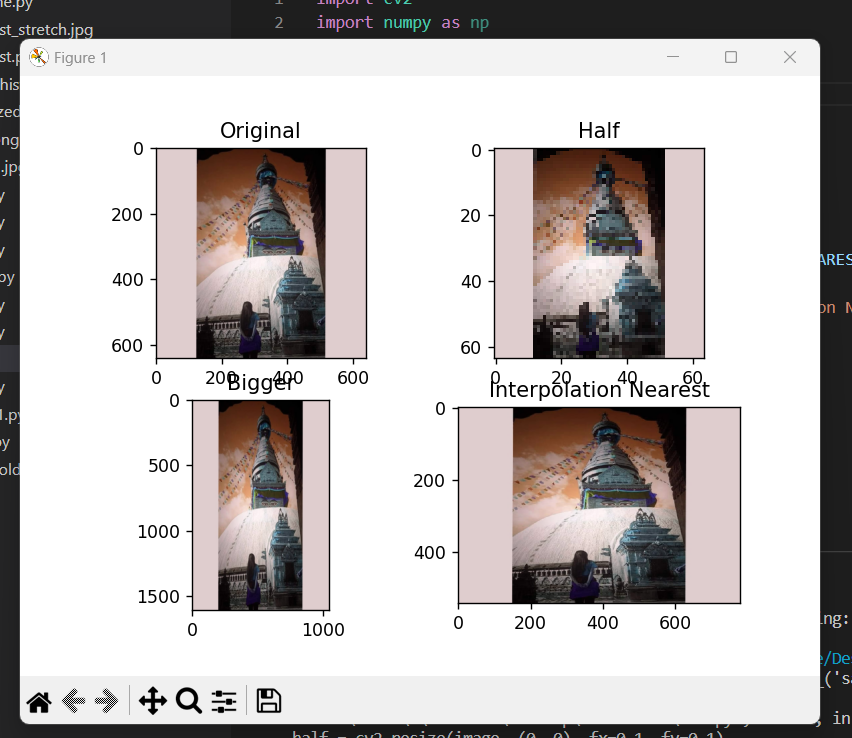
    plt.subplot(2, 2, i + 1)

    plt.title(Titles[i])

    plt.imshow(images[i])

plt.show()

**OUTPUT:**

****

**CONCLUSION:**

Zooming by interpolation is more effective than zooming by replication which is verified in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 6A*** | ***Filtering in spatial domain: Low pass filtering*** |
| Aim | To implement low pass filtering in spatial domain |

|  |  |
| --- | --- |
| Tool | PYTHON |
| Theory | Low pass filtering as the name suggests removes the high frequency content from the image. It is used to remove noise present in the image. Mask for the low pass filter is :  1/9 1/9 1/9  1/9 1/9 1/9  1/9 1/9 1/9  One important thing to note from the spatial response is that all the coefficients are positive. We could also use 5 x 5 or 7 x 7 mask as per our requirement. We place a 3 x 3 mask on the image . We start from the left hand top corner. We cannot work with the borders and hence are normally left as they are. We then multiply each component of the image with the corresponding value of the mask. Add these values to get the response. Replace the center pixel of the o/p image with these response. We now shift the mask towards the right till we reach the end of the line and then move it downwards. |
| Algorithm | 1. Read I/p image  2. Ignore the border pixel  3.Apply low pass mask to each and every pixel.  4. Display the o/p image |

**SOURCE CODE:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

def lowPassFiltering(img,size):#Transfer parameters are Fourier transform spectrogram and filter size

    h, w = img.shape[0:2]#Getting image properties

    h1,w1 = int(h/2), int(w/2)#Find the center point of the Fourier spectrum

    img2 = np.zeros((h, w), np.uint8)#Define a blank black image with the same size as the Fourier Transform Transfer

    img2[h1-int(size/2):h1+int(size/2), w1-int(size/2):w1+int(size/2)] = 1#Center point plus or minus half of the filter size,

forming a filter size that defines the size, then set to 1, preserving the low frequency part

    img3=img2\*img #A low-pass filter is obtained by multiplying the defined low-pass filter with the incoming Fourier

spectrogram one-to-one.

    return img3

def cv2\_imread(file\_path, flag=1):

    # Read Picture Data

    return cv2.imdecode(np.fromfile(file\_path, dtype=np.uint8), flag)

gray = cv2\_imread("kabina.jpg", 1)

gray = cv2.cvtColor(gray, cv2.COLOR\_BGR2GRAY)

gray = cv2.resize(gray, (1280, 720))

h,w =gray.shape

for i in range(3000):    #Add 3000 Noise Points

    x = np.random.randint(0, h)

    y = np.random.randint(0, w)

    gray[x,y] = 255

# Fourier transform

img\_dft = np.fft.fft2(gray)

dft\_shift = np.fft.fftshift(img\_dft)  # Move frequency domain from upper left to middle

# Low-pass filter

dft\_shift = lowPassFiltering(dft\_shift, 200)

res = np.log(np.abs(dft\_shift))

# Inverse Fourier Transform

idft\_shift = np.fft.ifftshift(dft\_shift)  # Move the frequency domain from the middle to the upper left corner

ifimg = np.fft.ifft2(idft\_shift)  # Fourier library function call

ifimg = np.abs(ifimg)

cv2.imshow("ifimg", np.int8(ifimg))

cv2.imshow("gray", gray)

# Draw pictures

plt.subplot(131), plt.imshow(gray, 'gray'), plt.title('Original Image')

plt.axis('off')

plt.subplot(132), plt.imshow(res, 'gray'), plt.title('Low-pass filter')

plt.axis('off')

plt.subplot(133), plt.imshow(np.int8(ifimg), 'gray'), plt.title('Effect after filtering')

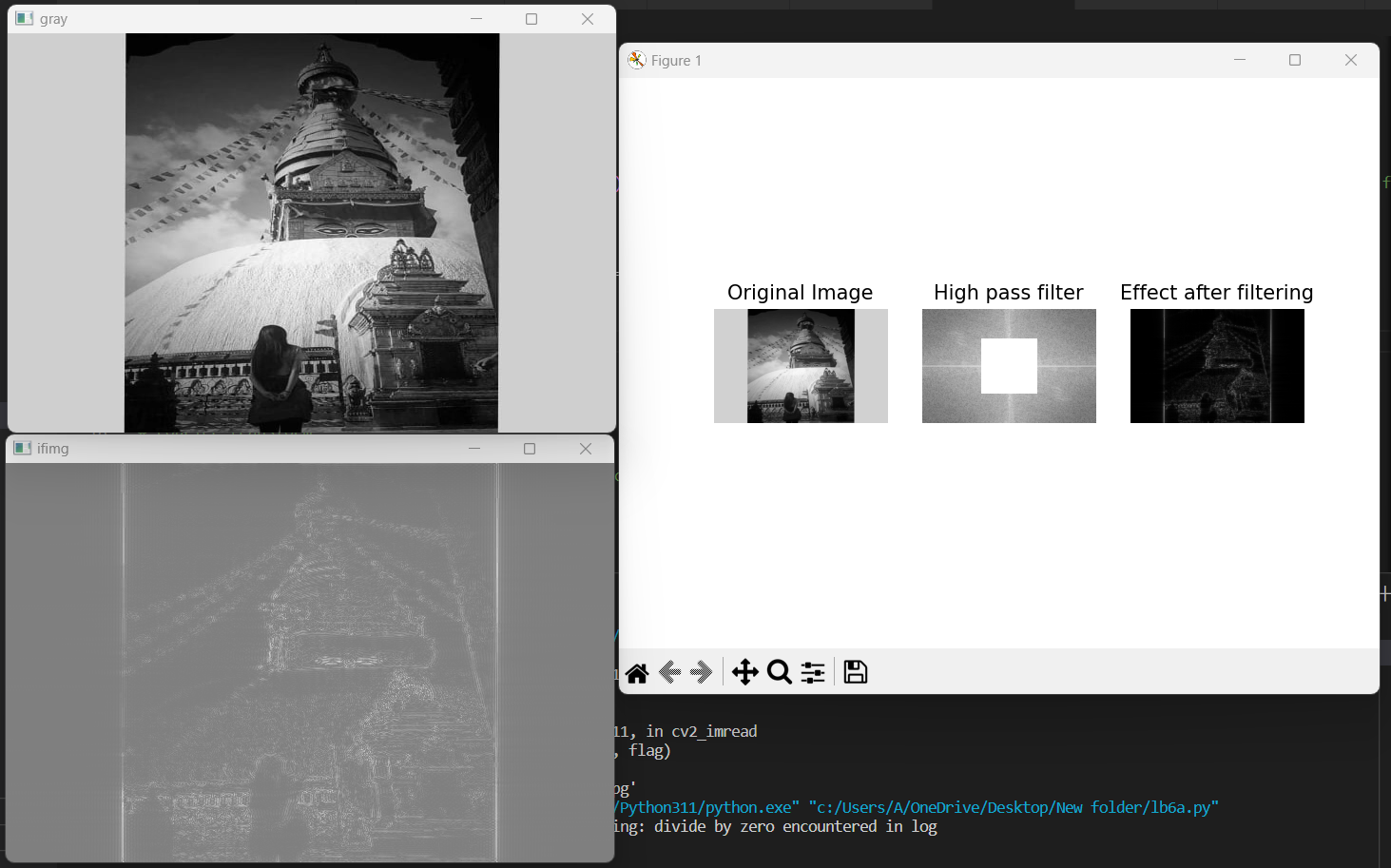
plt.axis('off')

plt.show()

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**

****

**CONCLUSION:**

Low pass filtering makes the image blurred which is verified in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 6B*** | ***Filtering in spatial domain: High pass filtering*** |
| Aim | To implement high pass filtering in spatial domain |
| Tool | PYTHON |

|  |  |
| --- | --- |
| Theory | High pass filtering as the name suggests removes the low frequency content from the image. It is used to highlight fine detail in an image or to enhance detail that has been blurred. Mask for the high pass filter is :  -1/9 -1/9 -1/9  -1/9 8/9 -1/9  -1/9 -1/9 -1/9  One important thing to note from the spatial response is that sum of all the coefficients is zero. We could also use 5 x 5 or 7 x 7 mask as per our requirement. We place a 3 x 3 mask on the image . We start from the left hand top corner. We cannot work with the borders and hence are normally left as they are. We then multiply each component of the image with the corresponding value of the mask. Add these values to get the response. Replace the center pixel of the o/p image with these response. We now shift the mask towards the right till we reach the end of the line and then move it downwards. |
| Algorithm | 1. Read I/p image  2. Ignore the border pixel  3.Apply high pass mask to each and every pixel.  4. Display the o/p image |

**SOURCE CODE:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

def highPassFiltering(img,size):

#Transfer parameters are Fourier transform spectrogram and filter size

    h, w = img.shape[0:2]#Getting image properties

    h1,w1 = int(h/2), int(w/2)#Find the center point of the Fourier spectrum

    img[h1-int(size/2):h1+int(size/2), w1-int(size/2):w1+int(size/2)] = 0#Center point plus or minus half of the filter size,

forming a filter size that defines the size, then set to 0

    return img

def cv2\_imread(file\_path, flag=1):

    # Read Picture Data

    return cv2.imdecode(np.fromfile(file\_path, dtype=np.uint8), flag)

gray = cv2\_imread("kabina.jpg", 1)

gray = cv2.cvtColor(gray, cv2.COLOR\_BGR2GRAY)

gray = cv2.resize(gray, (640, 420))

# Fourier transform

img\_dft = np.fft.fft2(gray)

dft\_shift = np.fft.fftshift(img\_dft)  # Move frequency domain from upper left to middle

#High pass filter

dft\_shift=highPassFiltering(dft\_shift,200)

res = np.log(np.abs(dft\_shift))

# Inverse Fourier Transform

idft\_shift = np.fft.ifftshift(dft\_shift)  #Move the frequency domain from the middle to the upper left corner

ifimg = np.fft.ifft2(idft\_shift)  # Fourier library function call

ifimg = np.abs(ifimg)

cv2.imshow("ifimg",np.int8(ifimg))

cv2.imshow("gray",gray)

# Draw pictures

plt.subplot(131), plt.imshow(gray, 'gray'), plt.title('Original Image')

plt.axis('off')

plt.subplot(132), plt.imshow(res, 'gray'), plt.title('High pass filter')

plt.axis('off')

plt.subplot(133), plt.imshow(np.int8(ifimg), 'gray'), plt.title('Effect after filtering')

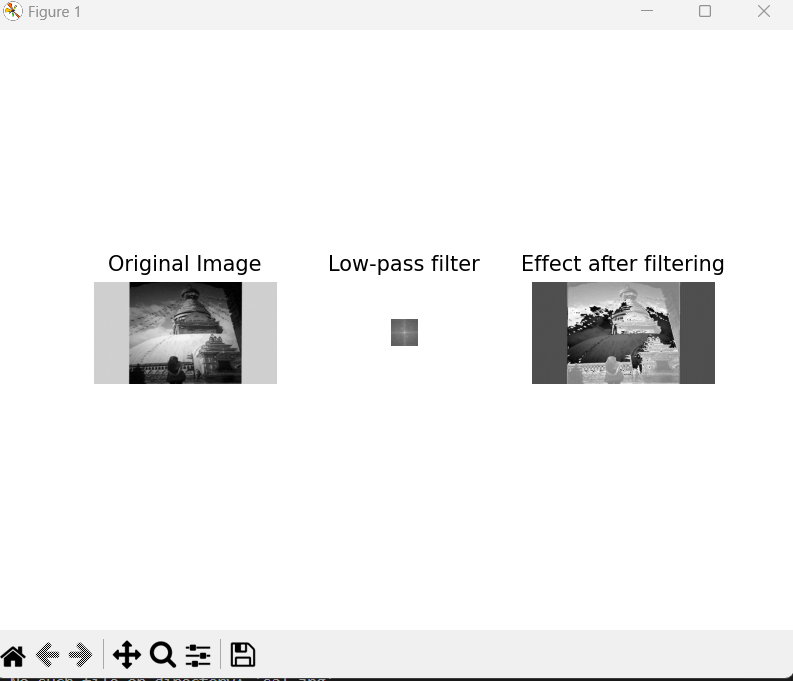
plt.axis('off')

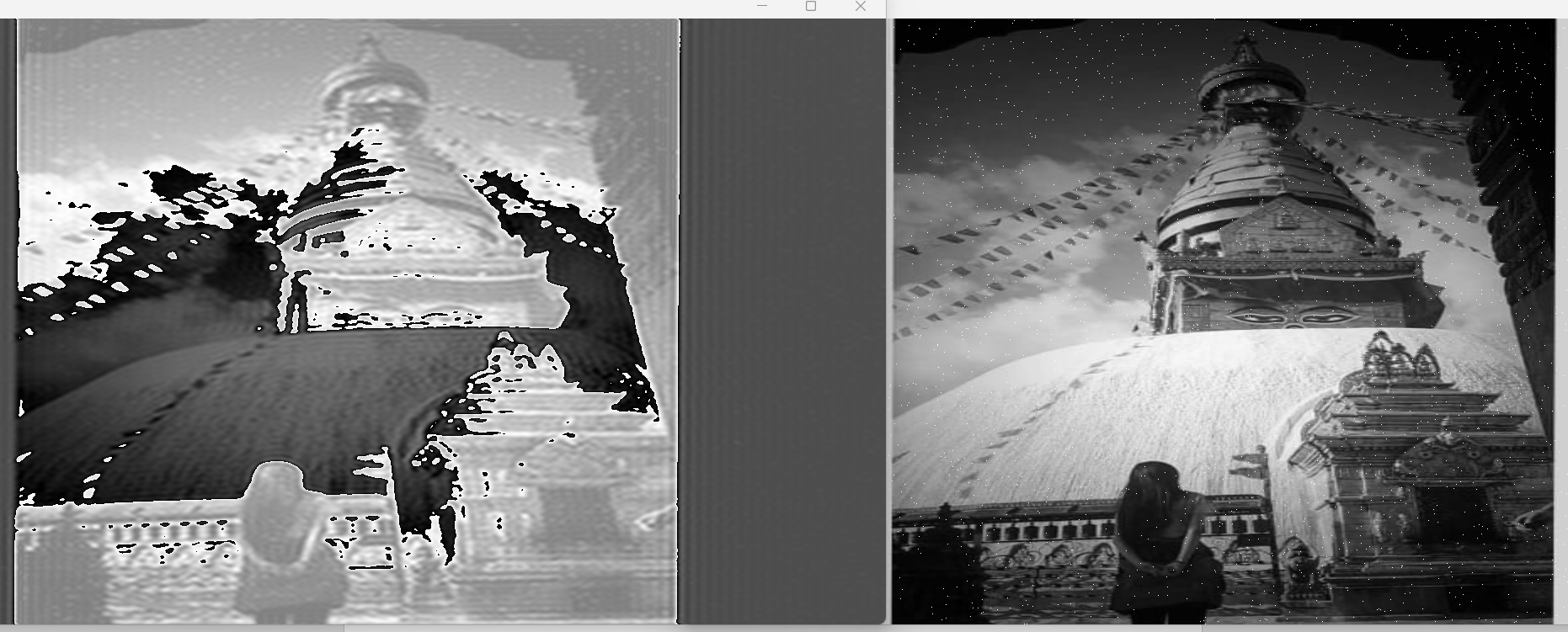
plt.show()

cv2.waitKey(0)

cv2.destroyAllWindows()

**OUTPUT:**

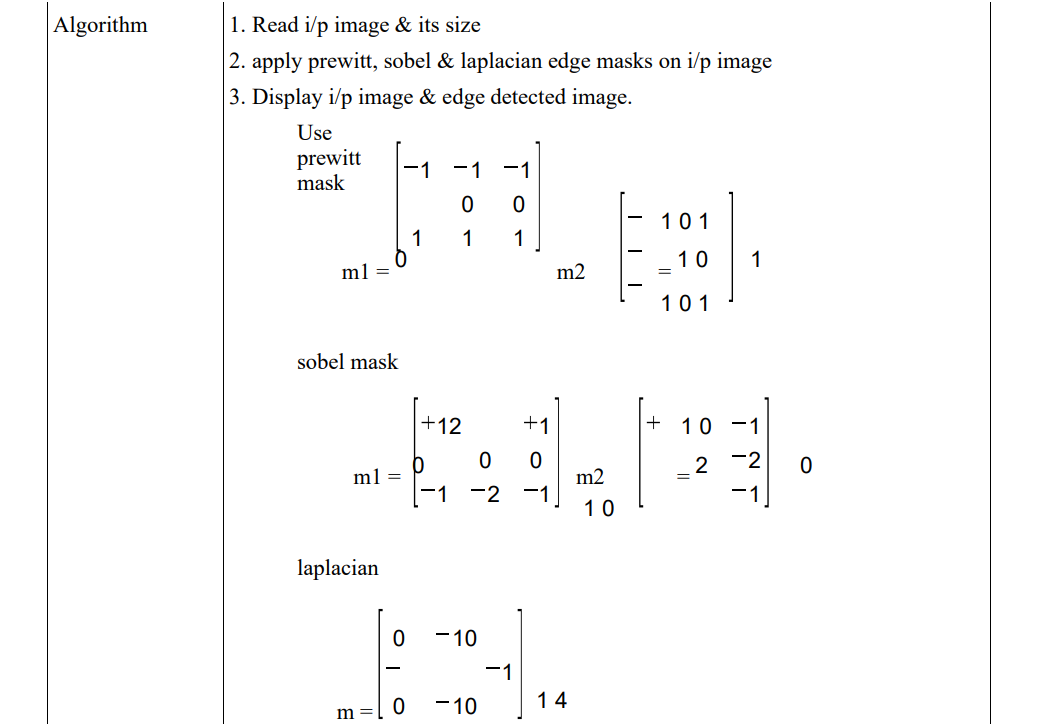
****

****

**CONCLUSION:**

High pass filtering makes the image sharpened which is verified in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 7*** | ***Edge Detection*** |
| Aim | To implement Image segmentation using edge detection technique. |
| Tool | PYTHON |
| Theory |  |



**SOURCE CODE:**

import numpy as np

import cv2

cap = cv2.VideoCapture(0)

while True:

    \_, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

    edges = cv2.Canny(gray, 30, 100)

    cv2.imshow("edges", edges)

    cv2.imshow("gray", gray)

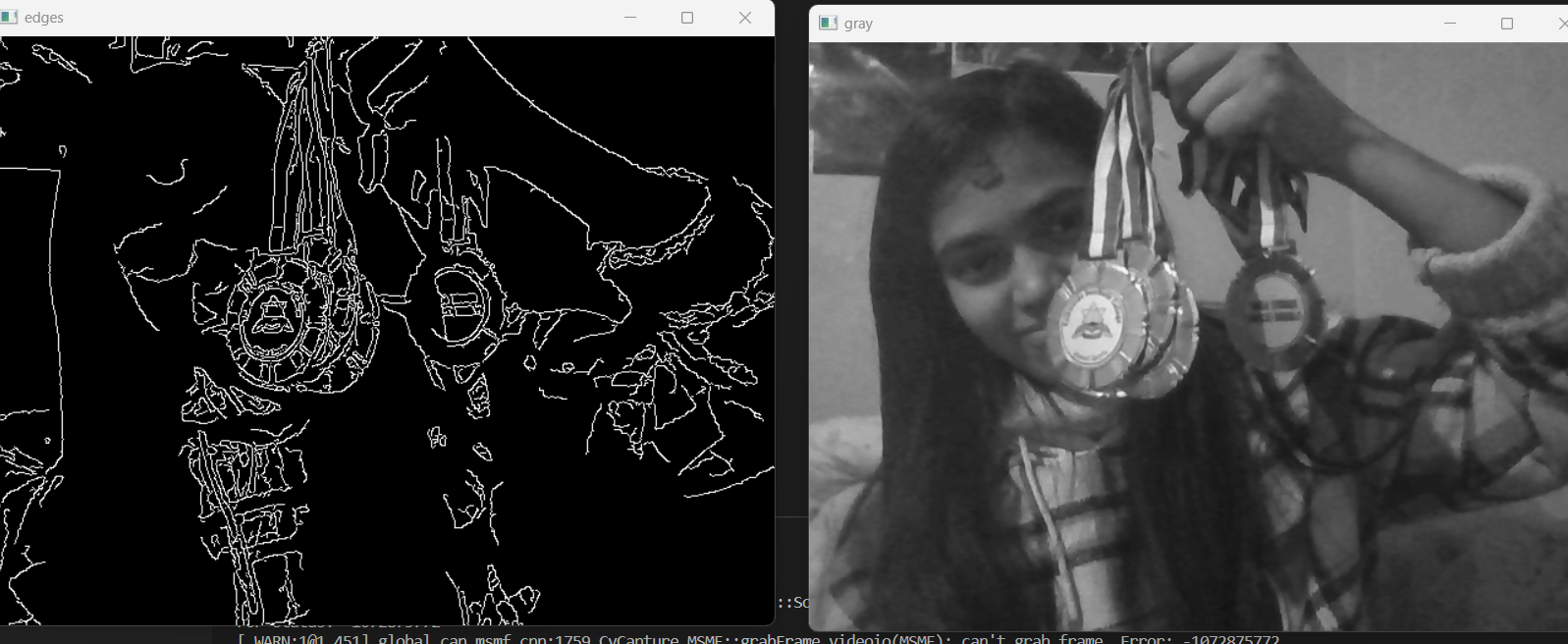
    if cv2.waitKey(1) == ord("q"):

        break

cap.release()

cv2.destroyAllWindows()

**OUTPUT:**

****

**CONCLUSION:**

Different edge detection methods were tested in this lab work.

|  |  |
| --- | --- |
| ***Experiment No. 8*** | ***Data compression using Huffman coding*** |
| Aim | To implement data compression using Huffman coding |
| Tool | PYTHON |
| Theory | It is used to reduce the space that an image uses on disk or in transit. It is the most popular technique to remove the coding redundancy. When coding the symbols of an information source individually Huffman coding yields the smallest possible number of code symbols per source symbol. It is lossless coding technique. |
| Algorithm | 1. Order the gray levels according to their frequency of use, most frequent first 2. Combine the two least used gray levels into one group, combine their frequencies and reorder the gray levels  3. Continue to do this until only two gray levels are left  4. Now allocate a '0' to one of these gray level groups and '1' to the other 5. Work back through the groupings so that where two groups have been combined to form a new , larger, group which is currently coded as 'ccc' , code one of the smaller groups as 'ccc0' and the other as 'cccc1'. |

**SOURCE CODE:**

string = ' ABABEBABEBABEC'

# Creating tree nodes

class NodeTree(object):

def \_\_init\_\_(self, left=None, right=None):

        self.left = left

        self.right = right

    def children(self):

        return (self.left, self.right)

    def nodes(self):

        return (self.left, self.right)

    def \_\_str\_\_(self):

        return '%s\_%s' % (self.left, self.right)

# Main function implementing huffman coding

def huffman\_code\_tree(node, left=True, binString=''):

    if type(node) is str:

        return {node: binString}

    (l, r) = node.children()

    d = dict()

    d.update(huffman\_code\_tree(l, True, binString + '0'))

    d.update(huffman\_code\_tree(r, False, binString + '1'))

    return d

# Calculating frequency

freq = {}

for c in string:

    if c in freq:

        freq[c] += 1

    else:

        freq[c] = 1

freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)

nodes = freq

while len(nodes) > 1:

    (key1, c1) = nodes[-1]

    (key2, c2) = nodes[-2]

    nodes = nodes[:-2]

    node = NodeTree(key1, key2)

    nodes.append((node, c1 + c2))

    nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman\_code\_tree(nodes[0][0])

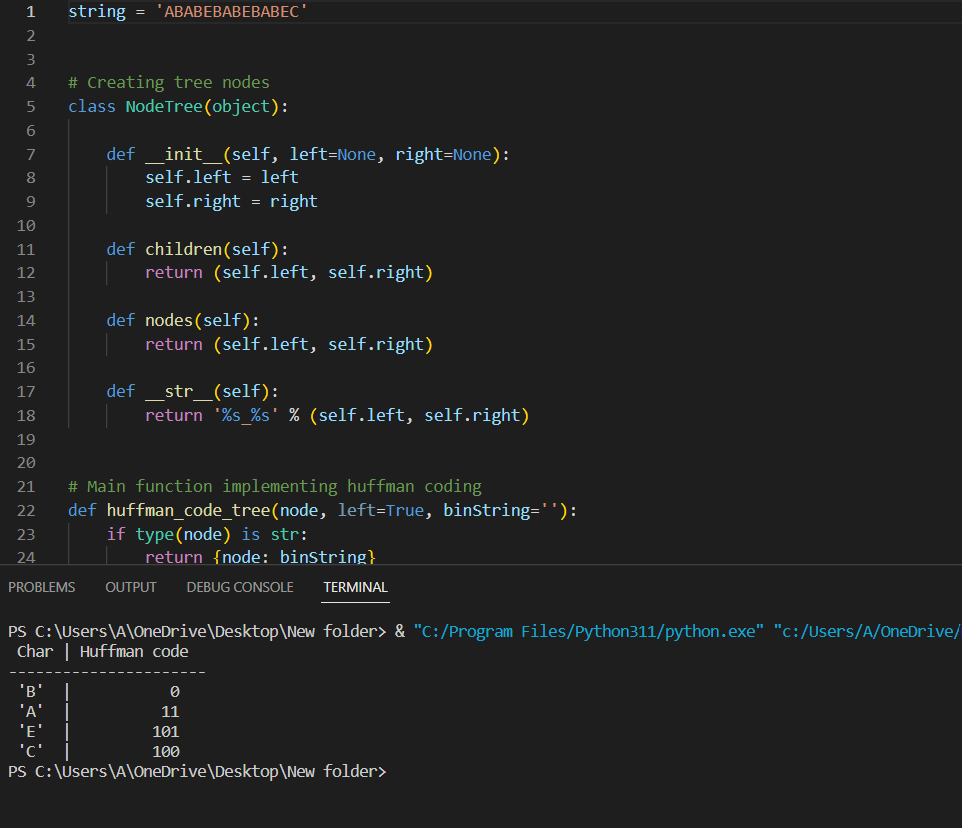
print(' Char | Huffman code ')

print('----------------------')

for (char, frequency) in freq:

    print(' %-4r |%12s' % (char, huffmanCode[char]))

**OUTPUT:**

****

**CONCLUSION:**

Huffman code is an instantaneous uniquely decodable block code which was implemented and the result were displayed at the console.