Visvesvaraya National Institute of Technology (VNIT), Nagpur

Digital Image Processing (ECP411)

Lab Report

Submitted by :
Pranav Kathar (BT19ECE058)
Semester VI

Submitted to :
Prof. Kishor M. Bhurchandi, Dr. Deep Gupta
(Lab Instructor)
Department of Electronics and Communication Engineering,
VNIT Nagpur

Contents

Experiment 1

1.1 Problem Statement: Write a generalised function for any coloured or grey image and custom image dimensions to evaluate the mean and standard deviation of the pixels of an image. Also, return the histogram of the pixel values that will be a 3D vector for coloured images and 2D vector for grey images. Name of the function - calculate_metrics Parameters to be returned - mean, standard deviation, histogram The data type of all the input and output arguments has to be a numpy ndarray.

1.2 Python Code:

```
#importing libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2

def calculate_metrics(ndarray_image):
    ndarray_image = np.asarray(Image.open(ndarray_image))
    #print(ndarray_image.shape)
    res = np.shape(ndarray_image)
    total_pixels=res[0]*res[1]
    h=res[0]
    w=res[1]
    #for gray image dimensions will be 2s
    if ndarray_image.ndim == 2:

        #mean
        sum=np.array([0.0])
        for i in range(h):
            for j in range(w):
                sum[0]+=ndarray_image[i,j]
        mean= sum/total_pixels

        #standard_deviation
        std=np.array([0.0])
        for i in range(h):
            for j in range(w):
                std[0]+=np.square(ndarray_image[i,j]-mean[0])
        standard_deviation= np.sqrt(std/total_pixels)

        #histogram
        hist=np.zeros((1,256),dtype=int)
        for i in range(h):
```

```
34              for j in range(w):
35                  hist[0,(ndarray_image[i,j])]+=1
36          # plt.bar(range(256),hist[0])
37          # plt.show()
38
39
40      #for color image dimensions will be 3
41      elif (ndarray_image.ndim==3):
42          #mean
43          sum=np.array([0.0,0.0,0.0])
44          for i in range(h):
45              for j in range(w):
46                  for k in range(3):
47                      sum[k]+=ndarray_image[i,j,k]
48          mean= sum/total_pixels
49
50          #standard_deviation
51          std=np.array([0.0,0.0,0.0])
52          for i in range(h):
53              for j in range(w):
54                  for k in range(3):
55                      std[k]+=np.square(ndarray_image[i,j,k]-mean[k])
56          temp=std/total_pixels
57          standard_deviation= np.sqrt(temp)
58
59          #histogram
60          hist=np.zeros((1,256,3),dtype=int)
61          for i in range(w):
62              for j in range(h):
63                  for k in range(3):
64                      hist[0][ndarray_image[j,i,k]][k]+=1
65          # plt.subplot(1,3,1)
66          # plt.bar(np.arange(0,256,1),hist[0][:,0].reshape(256))
67
68          # plt.subplot(1,3,2)
69          # plt.bar(np.arange(0,256,1),hist[0][:,1].reshape(256))
70
71          # plt.subplot(1,3,3)
72          # plt.bar(np.arange(0,256,1),hist[0][:,2].reshape(256))
73          # plt.show()
74
75      return mean,standard_deviation,hist
76
77  if __name__ == '__main__':
78      ndarray_image= "Lena.jpg" # color image
79      #ndarray_image= "cameraman.png" #gray image
80      print(calculate_metrics(ndarray_image))
81      # calculate_metrics(ndarray_image)
```

1.3 Results:

Input

Figure 1:

Figure 2:

Output

```
(256, 256)
(array([118.7244873]), array([62.34123969]), array([[   0,    0,    0,    0,    0,    0,    0,    4,  423, 1477, 1259,
        1175, 1456, 1529, 1685, 1338,  968,  471,  261,  187,  169,  140,
         107,  109,  104,   96,   99,  114,   86,  108,   91,   76,   99,
          92,   83,  105,   85,  100,   98,   97,   81,   86,   80,  106,
          68,   59,   72,   77,   65,   70,   61,   67,   50,   64,   74,
          67,   59,   70,   85,   80,   68,   72,   74,   86,   87,   76,
```

Figure 3: Mean , Std, and Hist values for Gray image (Cameraman.jpg)

1.3.0

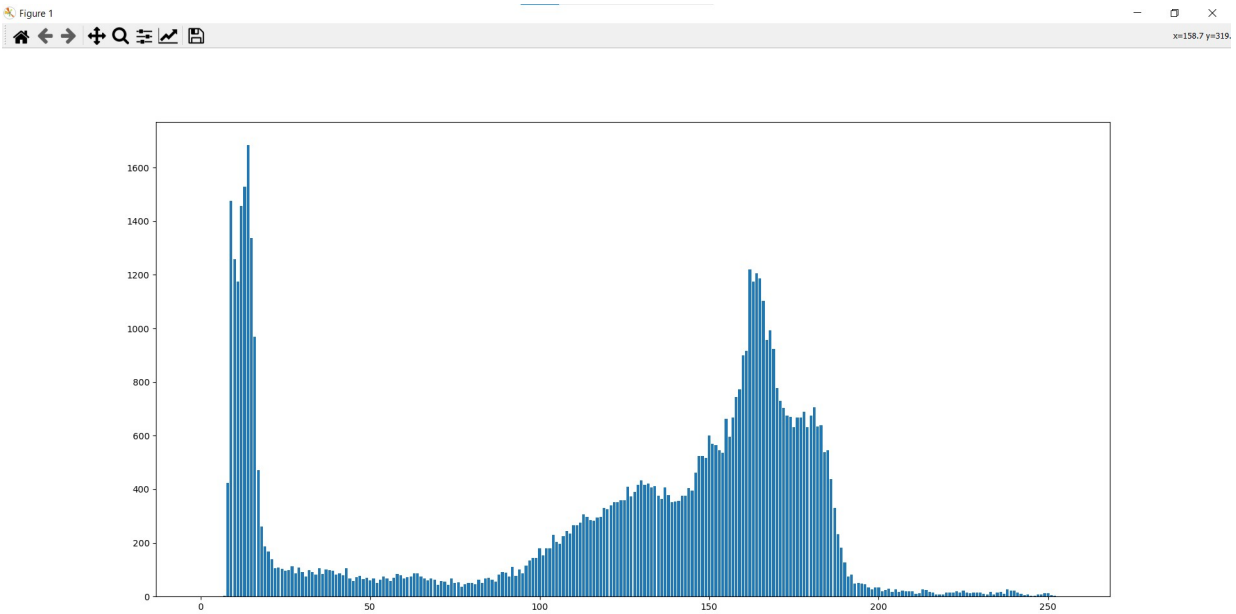

Figure 4: Histogram for Gray image (Cameraman.jpg)

```
C:\Users\91726\Desktop\DIP Assigns New>python -u "c:\Users\91726\Desktop\DIP Assigns New\A1.py"
(225, 225, 3)
(array([180.11085432,  99.07109136, 105.59751111]), array([49.54624285, 52.26583241, 35.41664947]), array([[[  0, 183,   3],
       [  0,  27,   0],
       [  0,  34,   0],
       [  0,  40,   1],
       [  0,  54,   1],
       [  0,  41,   0],
```

Figure 5: Mean , Std, and Hist values for RGB image (Lena.jpg)

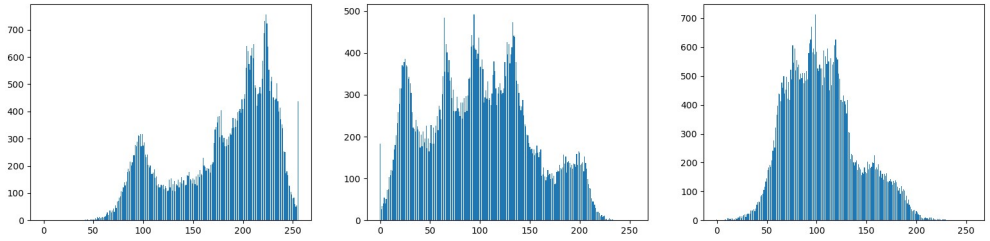

Figure 6: Histogram for RGB image (Lena.jpg)

1.4 Conclusion: From the output results and histograms obtained, We successfully performed metrics operations asked to calculate. In case of RGB image, Histograms of all the 3 channels were plotted. Also, mean and standard deviation has also been correctly calculated.

Experiment 2

2.1 Problem Statement: Write a generalised function called 'normalisation' for any coloured or grey image to perform image normalisation of the following types : a. Pixel Normalization: scale pixel values to the range 0-1. b. Pixel Centering: scale pixel values to have a zero mean. c. Pixel Standardization: scale pixel values to have a zero mean and unit variance.

2.2 Python Code:

```
#importing libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2

def normalization(Input_img,pn=True,pc=False,ps=False):

    if pn or pc or ps: #normalization centering or standardization
        Input_img= Input_img/255
        mean, var = np.mean(Input_img), np.var(Input_img)
        range= [np.min(Input_img),np.max(Input_img)]
        plt.title( "Normalised  Mean" )
        plt.imshow( Input_img)
        plt.show()

    if pc or ps: #centering or standardization
        Input_img=Input_img-mean
        mean, var = np.mean(Input_img), np.var(Input_img)
        range= [np.min(Input_img),np.max(Input_img)]
        plt.title( "Zero Mean Mean" )
        plt.imshow( Input_img)
        plt.show()

    if ps: # pixel standardization
        Input_img= Input_img/np.sqrt(var)
        mean, var = np.mean(Input_img), np.var(Input_img)
        range= [np.min(Input_img),np.max(Input_img)]
        plt.title( "Unity variance Mean" )
        plt.imshow( Input_img)
        plt.show()

    return Input_img, mean, range ,var

if __name__=='__main__':
    #reading image
```

```
119     Input_img = cv2.imread('8bit gray image.jpg')
120     #calling function
121     Input_img, mean, range ,var = ...
            normalization(Input_img,False,False,True)
122     #showing image for 1 second
123     cv2.imshow("",Input_img)
124     cv2.waitKey(0) # waiting 1 second for output
125     print(mean, range ,var)
```

2.3 Results:

Input

Figure 7:

Output

Figure 8:

Figure 9:

Figure 10:

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
6.422604560668086e-16 [-2.5777073578088165, 2.7125364240812755] 1.0000000000000004

(base) C:\Users\91726\Desktop\DIP Assigns New>
```

Figure 11:

2.4 Conclusion: We were asked to perform operations like normalization, centering and standardizaton on pixels. Hence firstly we normalised the input image, shifted it to zero mean and then scaled to get unity variance. All the respective plots are shown in output section

Experiment 3

3.1 Problem Statement: A) Linear Contrast Stretching
B) Gamma Transform

3.2 Python Code:

```
#importing libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2


def contrast_streching(image_path,r1,r2,s1,s2):
    #calculating slopes
    m1=s1/r1
    m2=(s2-s1)/(r2-r1)
    m3=(255-s2)/(255-r2)
    #reading image
    img=cv2.imread(image_path)#use opencv for reading image
    #shape of the image
    shape=np.shape(img)
    h,w=shape[0],shape[1]

    #func for different conditions according to the values given
    def update(val):
        if val<r1:
            return m1*val
        if val<r2:
            return m2*(val-r1)+s1
        return m3*(val-r2)+s2
    op=np.zeros(shape)
    for i in range(h):
        for j in range(w):
            for k in range(3):
                op[i,j,k]=update(img[i,j,k])

    op=op.astype(np.uint8)
    print(op,op.shape)
    cv2.imshow('original',img)
    cv2.imshow('image_after_contrast_streching',op)
    cv2.waitKey(0)
    return op

```

```
165 def gamma_transform(image_path,gamma):
166     img= (cv2.imread(image_path))/255
167     op=255*(np.power(img,gamma))
168     op=op.astype(np.uint8)
169 
170     print(op,op.shape)
171     cv2.imshow('original',img)
172     cv2.imshow("gamma transform",op)
173     cv2.waitKey(0)
174     return op
175 
176 if __name__ == '__main__':
177     path="cameraman.png"
178     #uncomment following if running manually
179     contrast_streching(path,50,100,80,200)
180     # gamma_transform(path,1.2)
```

3.3 Results:

Input

Figure 12:

Output

Figure 13: Contrast stretched image

Figure 14: Gamma transform image

3.4 Conclusion: Contrast stretching (often called normalization) is a simple image enhancement technique that attempts to improve the contrast in an image by 'stretching' the range of intensity values it contains to span a desired range of values, e.g. the the full range of pixel values that the image type concerned allows. It differs from the more sophisticated histogram equalization in that it can only apply a linear scaling function to the image pixel values. As a result the 'enhancement' is less harsh. (Most implementations accept a graylevel image as input and produce another graylevel image as output.)

And regarding gamma transform, Gamma correction or gamma is a nonlinear operation used to encode and decode luminance or tristimulus values in video or still image systems

Experiment 4

4.1 Problem Statement: a. Gray Level Slicing (Intensity Slicing) with and without background
b. Histogram Equalization

4.2 Python Code:

```
#importing libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2

def gray_slicing(input_image, thr1, thr2):
    image=Image.open(input_image)
    image=np.array(image)
    x,y=image.shape[0],image.shape[1]
    # plt.imshow(image,cmap='gray') #input image
    # plt.show()
    if (image.ndim==2):
        z1=np.zeros((x,y))
        z2=np.zeros((x,y))
        for i in range(x):
            for j in range(y):
                if(image[i,j]>thr1 and image[i,j]<thr2):
                    z1[i,j]=255
                    z2[i,j]=255
                else:
                    z1[i,j]=image[i,j]
                    z2[i,j]=0

    else:
        z1=np.zeros((x,y,3))
        z2=np.zeros((x,y,3))
        for i in range(x):
            for j in range(y):
                for k in range(3):
                    if(image[i,j,k]>thr1 and image[i,j,k]<thr2):
                        z1[i,j,k]=255
                        z2[i,j,k]=255
                    else:
                        z1[i,j,k]=image[i,j,k]
                        z2[i,j,k]=0

```

```
219         z1=z1.astype(np.uint8)
220         z2=z2.astype(np.uint8)
221     output_image_with_background=z1
222     output_image_without_background=z2
223     plt.title( "output_image_with_background" )
224     plt.imshow(z1)#output_image_with_background
225     plt.show()
226     plt.title( "output_image_without_background" )
227     plt.imshow(z2)#output_image_without_background
228     plt.show()
229 
230     return(output_image_with_background, ...
            output_image_without_background)
231 
232 def histogram_equalization(input_image):
233     image=Image.open(input_image)
234     image=np.array(image)
235     x,y=image.shape[0],image.shape[1]
236     # cv2.imshow("ip",image)
237     # cv2.waitKey(0)
238     output_image = np.zeros_like(image)
239 
240     if (image.ndim==2):
241         h,bins= np.histogram(image,bins=256)
242         pdf=h/(x*y)
243         cdf=(255*(np.cumsum(pdf))).astype(np.uint8)
244         for i in range(x):
245             for j in range(y):
246                 output_image[i,j]=cdf[image[i,j]]
247 
248     else:
249         for k in range(3):
250             h,bins= np.histogram(image[:,:,k],bins=256)
251 
252             pdf=h/(x*y)
253             cdf=(255*(np.cumsum(pdf))).astype(np.uint8)
254             for i in range(x):
255                 for j in range(y):
256                     output_image[i,j,k]=cdf[image[i,j,k]]
257     plt.title("equalized image")
258     plt.imshow(output_image)
259     plt.show()
260     return(output_image)
261 
262 if __name__ == '__main__':
263     # input_image="cameraman.png"
264     input_image="Lena.jpg"
265     # print(gray_slicing(input_image, 50, 150))
266     print(histogram_equalization(input_image))
```

4.3.0

4.3 Results:

Input

Figure 15:

Output

Figure 16:

Figure 17:


output_image_without_background
0
50
100
150
200
250
0
50
100
150
200
250

Figure 18:

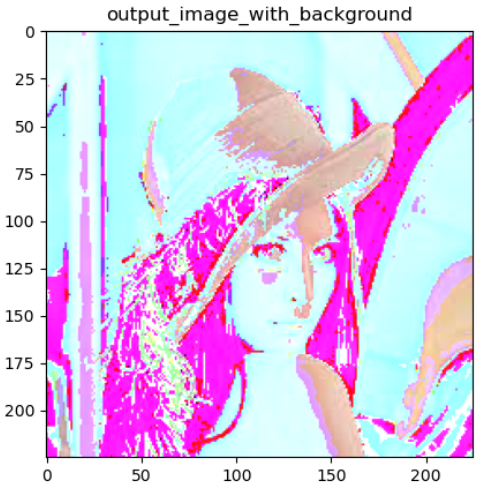

Figure 19:

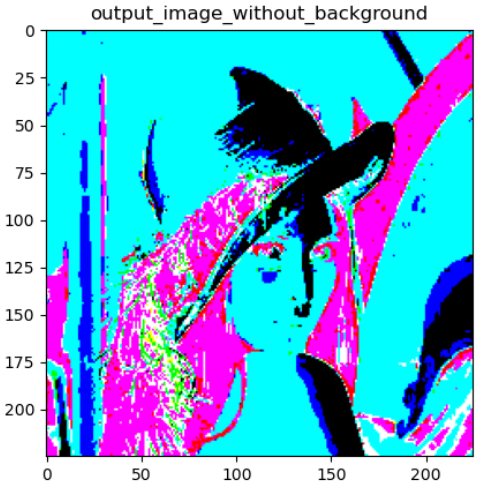

Figure 20:

Figure 21:

Figure 22:

4.4 Conclusion: Grey level slicing is equivalent to band pass filtering. It manipulates group of intensity levels in an image up to specific range by diminishing rest or by leaving them alone.

- 1) Grey level slicing without background: It displays high values in the specific region of an image and low value to other regions by ignoring background.
- 2) Grey level slicing with background: displays high values in specific region of an image and original grey level to other region by preserving background

Histogram equalization can make an image fill the available gray-scale range, and be uniformly distributed over that range.

Experiment 5

5.1 Problem Statement: a. Scaling of an image with sacling factor
x = × v
y = × w
where (v, w) are pixel coordinates in input image and (x, y) are the pixel coordinates in output image.
b. Rotation of an image with angle = 450
x = v × cos() − w × sin()
y = v × sin() + w × cos()
where (v, w) are pixel coordinates in input image and (x, y) are the pixel coordinates in output image.

5.2 Python Code:

```
268 import numpy as np
269 import cv2
270 import matplotlib.pyplot as plt
271 from PIL import Image
272 from numpy import cos,sin
273
274 def scale_image(input_image, scaling_factor):
275     a=scaling_factor
276     image=Image.open(input_image)
277     image=np.array(image)
278     x,y=image.shape[0],image.shape[1]
279
280
281     output_image = np.zeros((a*x,a*y))
282     for i in range(a*x):
283         for j in range(y*a):
284             output_image[i,j]=image[i//a,j//a]
285     plt.subplot(1,2,1)
286     plt.title("input image")
287     plt.imshow(image,cmap='gray')#input image
288
289     plt.subplot(1,2,2)
290     plt.title('scaled output image')
291     plt.imshow(output_image,cmap='gray')#output image
292     plt.show()
293     return(output_image)
294
295 def rotate_image(input_image, angle):
296     theta=np.radians(angle)
```

```
297     image=Image.open(input_image)
298     image=np.array(image)
299     x,y=image.shape[0],image.shape[1]
300     c1,c2=x//2,y//2
301
302     X=x*cos(theta)+y*sin(theta)
303     Y=x*sin(theta)+y*cos(theta)
304     X,Y=abs(int((X+1))),abs(int((Y+1)))
305     C1,C2=X//2,Y//2
306     output_image=np.zeros((X,Y))
307
308     for i in range(x):
309         for j in range(y):
310             var1,var2=i-c1,j-c2
311             var3,var4=var1*cos(theta)-var2*(sin(theta)),
312             (var1)*sin(theta)+var2*(cos(theta))
313             var3,var4=var3+C1,var4+C2
314             var3,var4=int(var3),int(var4)
315             output_image[var3,var4]=image[i,j]
316
317
318
319     for i in range(X):
320         for j in range(Y):
321             if output_image[i,j]==0:
322                 try:
323                     output_image[i,j]=output_image[i,j+1]
324                 except:
325                     output_image[i,j]=output_image[i,j-1]
326
327     output_image=output_image.astype(np.uint8)
328
329     plt.subplot(1,3,1)
330     plt.title("input image")
331     plt.imshow(image,cmap='gray')#input image
332
333     plt.subplot(1,3,2)
334     plt.title('output image before interpolation')
335     plt.imshow(output_image,cmap='gray')#output image before ...
        interpolation
336
337     plt.subplot(1,3,3)
338     plt.title('output_image after interpolation')
339     plt.imshow(output_image,cmap='gray')#output image after ...
        interpolation
340     plt.show()
341
342     return(output_image)
343
```

```
344 if __name__ == '__main__':
345     input_image="cameraman.png"
346     #print(scale_image(input_image, 2))
347     print(rotate_image(input_image, 45))
```

5.3 Results:

Input

Figure 23:

Output

Figure 24:

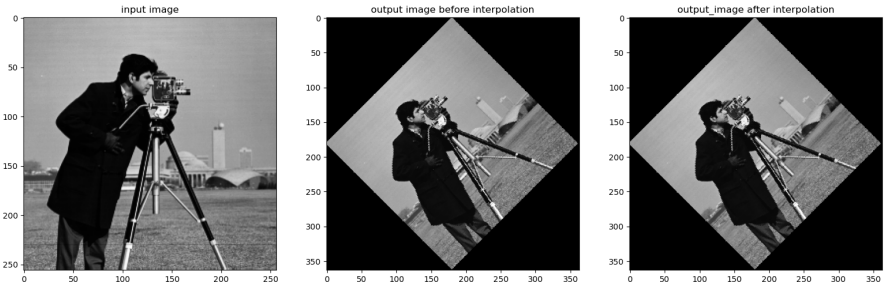

Figure 25:

5.4 Conclusion: IMAGE SCALING (USING NEAREST NEIGHBOR INTERPOLATION): Nearest neighbor interpolation — this is the easiest way to interpolate. This method simply determines the "closest" neighboring pixel and takes a value for its intensity.
For IMAGE ROTATION, we used the formulae provided and successfully rotated the image followed by interpolation. The input to an image rotation routine is an image, the rotation angle , we can decide point about which rotation is done.

Experiment 6

6.1 Problem Statement: Write a generalised function named "convolution2d" to perform 2D spatial convolution that has 4 input arguments, the input of dimensions M1 × M2 × 3 or M1 × M2, a kernel of dimensions of N1 × N2, a non zero stride and padding. DO not use any predefined python functions.

The data type of input image of size M1×M2×3 or M1×M2 and kernel/filter of size N1×N2 has to be numpy ndarray, padding and stride will also be numpy arrays of size [2 × 1] indicating stride and padding in x and y direction respectively. (input image, kernel, stride, padding) The output has to be convolved image having the datatype of a numpy ndarray of dimensions M1 × M2 × 3 or M1 × M2.

6.2 Python Code:

```
349 import numpy as np
350 import matplotlib.pyplot as plt
351 from PIL import Image
352 import cv2
353
354 def convolution2d(image,ker,stride,padding):
355     print(ker)
356     def invertkernel(k):
357         hk,wk=np.shape(k)
358         new=np.zeros([hk,wk])
359         new2=np.zeros([hk,wk])
360         for i in range(hk):
361             new[i,:]=ker[hk-i-1,:]
362
363         for j in range(wk):
364             new2[:,j]=new[:,wk-j-1]
365         #print (new2)
366         return new2
367
368     kernel=invertkernel(ker)
369     k1,k2=np.shape(kernel)#kernel size
370     h=np.shape(image)[0]
371     w=np.shape(image)[1]
372     x_pad,y_pad=padding[0,0],padding[1,0]#padding size
373     H,W=h+2*x_pad,w+2*y_pad#size of padded img
374
375     if(image.ndim==2):
376         new=np.zeros((H,W))#zero matrix (+padded )
377         c_img=np.zeros((H-k1,W-k2))#output img zero matrix
378         for i in range(H):
```

```
379             for j in range(W):
380                 if i<x_pad or j<y_pad or i≥(h+x_pad) or j≥(w+y_pad):
381                    new[i,j]=0
382                 else:
383                     new[i,j]=image[i-x_pad,j-y_pad]
384         s1,s2=stride[0,0],stride[1,0]
385         for i in range(k1//2,H-k1//2,s1):
386             for j in range(k2//2,W-k2//2,s2):
387                 temp=0
388                 for p in range(k1):
389                     for q in range(k2):
390                         temp+=kernel[p,q]*new[i+p-k1//2][j+q-k2//2]
391                 if temp>255:
392                     temp=255
393                 if temp<0:
394                     temp=0
395                 c_img[i-k1//2-1,j-k2//2-1]=temp
396     else:
397         new=np.zeros((H,W,3))#zero matrix (+padded )
398         c_img=np.zeros((H-k1,W-k2,3))#output img zero matrix
399         for k in range(3):
400             for i in range(H):
401                 for j in range(W):
402                         if i<x_pad or j<y_pad or i≥(h+x_pad) or ...
                                j≥(w+y_pad):
403                             new[i,j,k]=0
404                         else:
405                             new[i,j,k]=image[i-x_pad,j-y_pad,k]
406         s1,s2=stride[0,0],stride[1,0]
407         for k in range(3):
408             for i in range(k1//2,H-k1//2,s1):
409                 for j in range(k2//2,W-k2//2,s2):
410                         temp=0
411                         for p in range(k1):
412                             for q in range(k2):
413                                 temp+=kernel[p,q]*new[i+p-k1//2,j+q-k2//2,k]
414                         if temp>255:
415                             temp=255
416                         if temp<0:
417                             temp=0
418                         c_img[i-k1//2-1,j-k2//2-1,k]=temp
419
420     c_img=c_img.astype(np.uint8)
421
422     plt.subplot(1,2,1)
423     plt.title("input image")
424     plt.imshow(image,cmap='gray')#input image
425
426     plt.subplot(1,2,2)
```

```
427     plt.title('convolved output image (blurring filters)')
428     plt.imshow(c_img,cmap='gray')#convolved output image
429     plt.show()
430 
431     # cv2.imshow("Input_image", image)
432     # cv2.imshow("convolution",c_img)
433     # cv2.waitKey(0)
434     # cv2.destroyAllWindows()
435 
436     return c_img.astype(np.uint8)
437 
438 if __name__ == '__main__':
439     # K= np.array([[1,0,-1],[0,0,0],[-1,0,1]])
440     #K= np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])
441     #K = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
442     #K= np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
443     K= np.array([[1,1,1],[1,1,1],[1,1,1]])/9
444     #K = np.array([[1,2,1], [2,4,2], [1,2,1]])/16
445     #K = ...
            np.array([[1,4,6,4,1],[4,16,24,16,4],[6,24,36,24,6],[4,16,24,16,4],[1,4,6,4,
446 
447     stride=np.array([[1],[1]])
448     padding=np.array([[1],[1]])
449     # stride= np.array([[2], [3]])
450     # padding= np.array([[1], [1]])
451     #image = cv2.imread("cameraman.png",0)
452     #img= cv2.imread("Lena.png")
453     #image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY )
454 
455     # img=Image.open("Lena.jpg")
456     img=Image.open("cameraman.png")
457     image=np.asarray(img)
458     convolution2d(image,K,stride,padding)
459     #print(convolution2d(image,K,stride,padding))
```

6.3 Results:

Input

Output

Figure 26:

Figure 27:

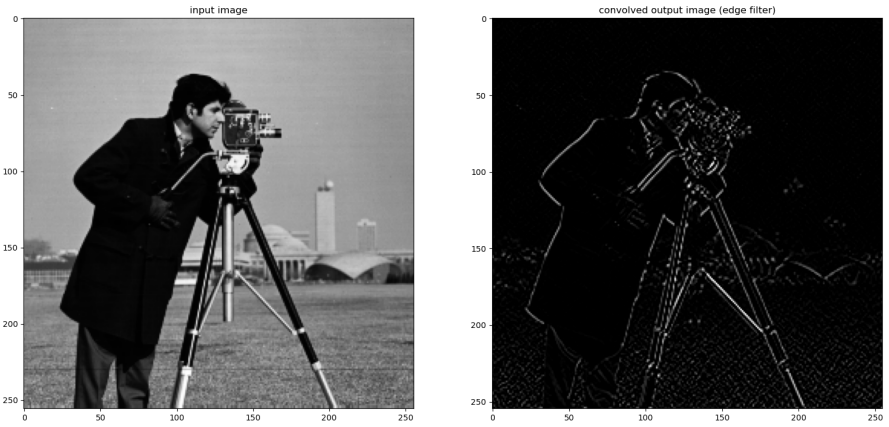

Figure 28:

Figure 29:

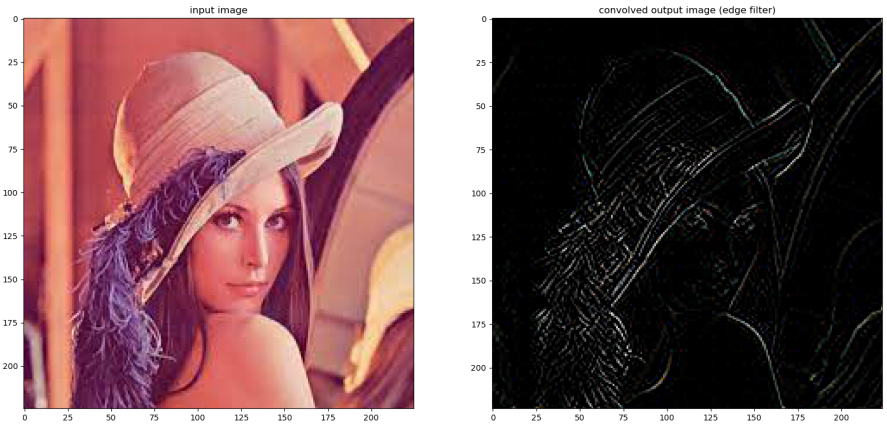

Figure 30:

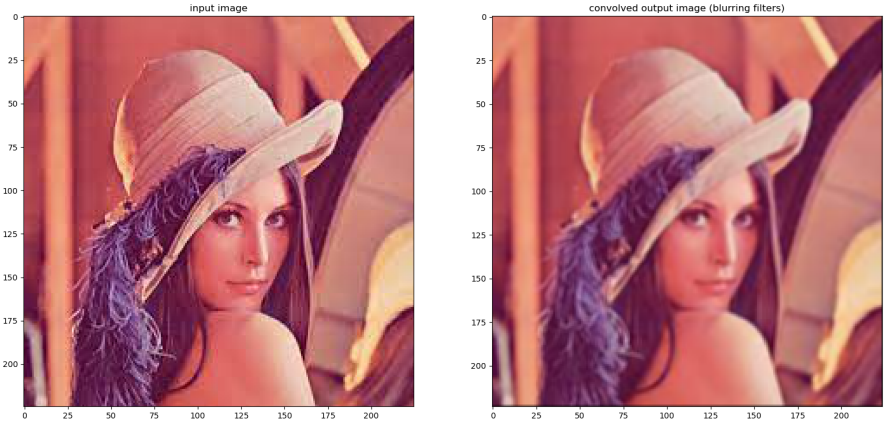

Figure 31:

6.4 Conclusion: As we know, convolution is the process of transforming an image by applying a kernel over each pixel and its local neighbors across the entire image. we did the same using various kernel matrices refering to edge filters , blurring filters, etc.

Experiment 7

7.1 Problem Statement: Write a generalised function named "correlation2d" to perform 2D spatial correlation that has 4 input arguments, the input of dimensions M1 × M2 × 3 or M1 × M2, a kernel of dimensions of N1 × N2, a non zero stride and padding. DO not use any predefined python functions.

The data type of input image of size M1×M2×3 or M1×M2 and kernel/filter of size N1×N2 has to be numpy ndarray, padding and stride will also be numpy arrays of size [2 × 1] indicating stride and padding in x and y direction respectively. (input image, kernel, stride, padding) The output has to be correlation image having the datatype of a numpy ndarray of dimensions M1 × M2 × 3 or M1 × M2.

7.2 Python Code:

```
460 import numpy as np
461 import matplotlib.pyplot as plt
462 from PIL import Image
463 import cv2
464
465 def correlation2d(image,kernel,stride,padding):
466     k1,k2=np.shape(kernel)#kernel size
467     h=np.shape(image)[0]#ip image size
468     w=np.shape(image)[1]
469
470     x_pad,y_pad=padding[0,0],padding[1,0]#padding size
471     H,W=h+2*x_pad,w+2*y_pad#size of padded img
472
473     #gray img
474     if(image.ndim==2):
475         new=np.zeros((H,W))#zero matrix (+padded )
476         c_img=np.zeros((H-k1,W-k2))#output img zero matrix
477         for i in range(H):
478             for j in range(W):
479                 if i<x_pad or j<y_pad or i≥(h+x_pad) or j≥(w+y_pad):
480                    new[i,j]=0
481                 else:
482                     new[i,j]=image[i-x_pad,j-y_pad]
483         s1,s2=stride[0,0],stride[1,0]
484         for i in range(k1//2,H-k1//2,s1):
485             for j in range(k2//2,W-k2//2,s2):
486                 temp=0
487                 for p in range(k1):
488                     for q in range(k2):
489                         temp+=kernel[p,q]*new[i+p-k1//2][j+q-k2//2]
```

```
490                 if temp>255:
491                     temp=255
492                 if temp<0:
493                     temp=0
494                 c_img[i-k1//2-1,j-k2//2-1]=temp
495     #color img
496     else:
497         new=np.zeros((H,W,3))#zero matrix (+padded )
498         c_img=np.zeros((H-k1,W-k2,3))#output img zero matrix
499         for k in range(3):
500             for i in range(H):
501                 for j in range(W):
502                         if i<x_pad or j<y_pad or i≥(h+x_pad) or ...
                                j≥(w+y_pad):
503                             new[i,j,k]=0
504                         else:
505                             new[i,j,k]=image[i-x_pad,j-y_pad,k]
506         s1,s2=stride[0,0],stride[1,0]
507         for k in range(3):
508             for i in range(k1//2,H-k1//2,s1):
509                 for j in range(k2//2,W-k2//2,s2):
510                         temp=0
511                         for p in range(k1):
512                             for q in range(k2):
513                                 temp+=kernel[p,q]*new[i+p-k1//2,j+q-k2//2,k]
514                         if temp>255:
515                             temp=255
516                         if temp<0:
517                             temp=0
518                         c_img[i-k1//2-1,j-k2//2-1,k]=temp
519     #print(image.shape)#ip
520     #print(kernel.shape)
521     #print(padding.shape)
522     #print(c_img.shape)
523     c_img=c_img.astype(np.uint8)
524     # cv2.imshow("Input_image", image)
525     # cv2.imshow("convolution",c_img)
526     # cv2.waitKey(0)
527     # cv2.destroyAllWindows()
528
529     plt.subplot(1,2,1)
530     plt.title("input image")
531     plt.imshow(image,cmap='gray')#input image
532
533     plt.subplot(1,2,2)
534     plt.title('correlated output image (blurring filters)')
535     plt.imshow(c_img,cmap='gray')#convolved output image
536     plt.show()
537
```

```
538     return c_img.astype(np.uint8)
539
540 if __name__ == '__main__':
541
542
543     # K= np.array([[1,0,-1],[0,0,0],[-1,0,1]])
544     #K= np.array([[0,-1,0],[-1,4,-1],[0,-1,0]])
545     #K = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
546     #K= np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
547     K= np.array([[1,1,1],[1,1,1],[1,1,1]])/9
548     #K = np.array([[1,2,1], [2,4,2], [1,2,1]])/16
549     #K = ...
            np.array([[1,4,6,4,1],[4,16,24,16,4],[6,24,36,24,6],[4,16,24,16,4],[1,4,6,4,
550
551     stride=np.array([[1],[1]])
552     padding=np.array([[1],[1]])
553     # stride= np.array([[2], [3]])
554     # padding= np.array([[1], [1]])
555     #image = cv2.imread("cameraman.png",0)
556     #img= cv2.imread("Lena.png")
557     #image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY )
558
559
560
561     # img=Image.open("Lena.jpg")
562     img=Image.open("cameraman.png")
563     image=np.asarray(img)
564
565     print(correlation2d(image, K,stride,padding))
```

7.3 Results:

Input

Output

Figure 32:

Figure 33:

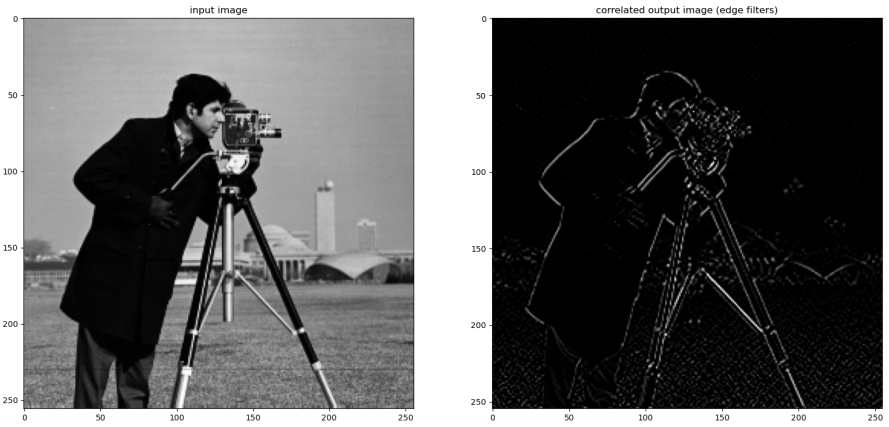

Figure 34:

Figure 35:

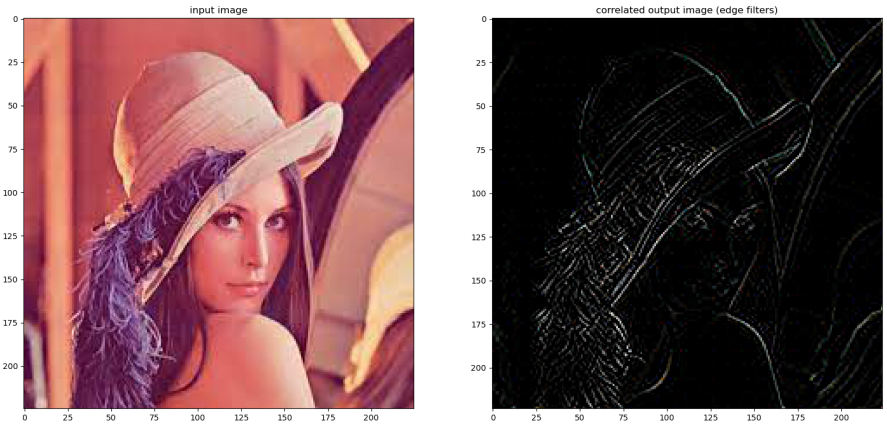

Figure 36:

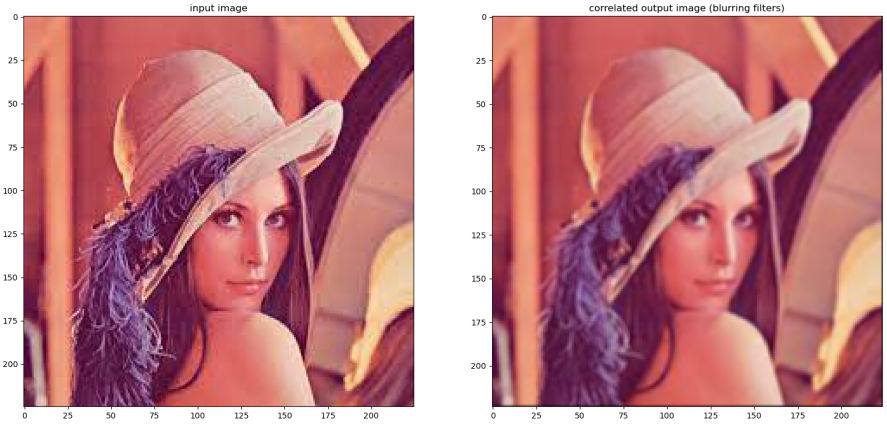

Figure 37:

7.4 Conclusion: Correlation is same as convolution, with a small differnce that in correlation kernel matrix is not transposed. The output plots obtained are same for convolution and correlation are same as we took symmetric kernel matrices.

Experiment 8

8.1 Problem Statement: a. Discrete Fourier Transform (DFT) of a gray image using conventional method.
b. Discrete Fourier Transform (DFT) of a gray image using Row-Column method.
c. Discrete Fourier Transform (DFT) of a gray image using Kernel method.
Compare required computational time for the three functions.

8.2 Python Code:

```
568 import numpy as np
569 from PIL import Image
570 import matplotlib.pyplot as plt
571
572
573 def DFT_Conventional():
574     # img = Image.open( path)
575     # img = np.array(img)
576     img = np.array([ [1,1,1] , [1,1,1] , [1,1,1]])
577     img = img.astype(np.uint8)
578
579     print(img)
580     h,w = np.shape( img)
581     new_img = np.zeros_like(img)
582
583
584     for u in range(h):
585         for v in range(w):
586             s = 0
587             for x in range(h):
588                 for y in range(w):
589                     s += img[x,y] * np.exp( -1 * 1j*2*np.pi *( ...
                            (u*x/h) + (v*y/w)))
590             s = round(s , 3)
591             new_img[u,v] = s
592             print(s)
593
594     new_img = new_img.astype(np.uint8)
595     plt.imshow( new_img,cmap='gray')
596     plt.show()
597
598     print(new_img)
599
600     return(new_img)
601
```

```
602 def DFT_Row_Column(path):
603 
604     img = Image.open( path)
605     img = np.array(img)
606     # img = np.array([ [1,1,1] , [1,1,1] , [1,1,1]   ])
607     shape = np.shape( img)
608     h,w = shape[0] , shape[1]
609 
610     new_img = np.zeros_like(img)
611     new_img = new_img.astype(np.float64).view(np.complex64)
612 
613     for u in range(h):
614         for v in range(w):
615             s = 0
616             for y in range(w):
617                 s += img[u,y] * np.exp( -1 * 1j*2*np.pi *( (v*y/w)))
618             s = round(s , 3)
619             new_img[u,v] = s
620 
621     img = new_img
622     new_img = np.zeros_like(img)
623 
624     for v in range(h):
625         for u in range(w):
626             s = 0
627             for x in range(w):
628                 s += img[ x, v] * np.exp( -1 * 1j*2*np.pi *( ...
                        (u*x/h)))
629             s = round(s , 3)
630             new_img[u,v] = s
631             # print(s)
632     plt.title("new image")
633     plt.imshow( new_img)
634     plt.show()
635 
636     return(new_img)
637 
638 def DFT_Kernel(path):
639     img = Image.open( path)
640     img = np.array(img)
641     # img = np.array([ [1,1,1] , [1,1,1] , [1,1,1]   ])
642     shape = np.shape( img)
643     h,w = shape[0] , shape[1]
644 
645     A = np.zeros_like(img)
646     A = A.astype( np.csingle)
647 
648     for u in range(h):
649         for v in range(w):
```

```
650             s = 0
651             s = np.exp( -1 * 1j*2*np.pi *( (v*u/w)))
652             # s = round(s , 3)
653             A[u,v] = s
654 
655     img = np.dot( img , A)
656 
657 
658     for u in range(h):
659         for v in range(w):
660             s = 0
661             s = np.exp( -1 * 1j*2*np.pi *( (u*v/h)))
662             # s = round(s , 3)
663             A[u,v] = s
664 
665     img = np.dot( img.T , A)
666     # print( img)
667     # print( img)
668     # plt.imshow( img)
669     # plt.show()
670 
671     # return(new_img)
672     return(img)
673 
674 # if __name__ == '__main__':
675 #     # path = "8bit gray image.jpg"
676 
677 #     DFT_Conventional()
678 #     # img = DFT_Conventional( path)
679 #     # img = DFT_Row_Column( path)
680 #     # img = DFT_Kernel(path)
681 #     # print(img)
682 
683 #     # plt.imshow( np.abs(img))
684 #     # plt.show()
685 
686 if __name__ == '__main__':
687     path = "cameraman_new_32 (1).png"
688     # img = DFT_Conventional( path)
689     DFT_Row_Column(path)
690     # img = DFT_Kernel(path)
691     # print( (np.abs(img) *255 / np.max( np.abs(img) ) ...
          ).astype(np.uint8) )
692 
693     # plt.imshow( (np.abs(img) *255 / np.max( np.abs(img) ) ...
          ).astype(np.uint8) )
694     # plt.show()
```

8.3 Results:

Input

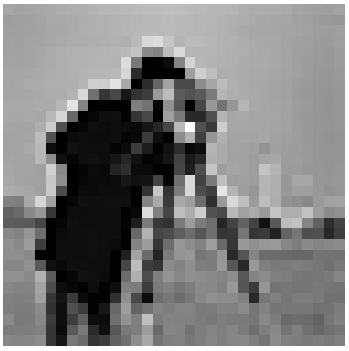

Figure 38: 32*32 cameraman gray image

Output

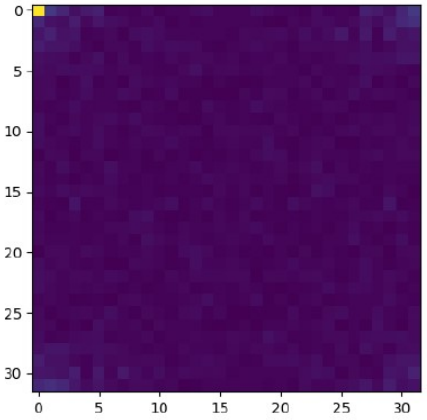

Figure 39: Output of conventional DFT

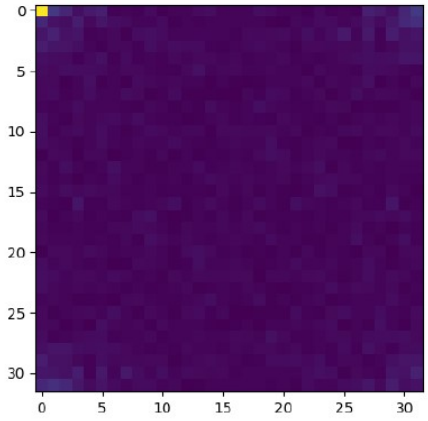

Figure 40: Output of Row-Column DFT

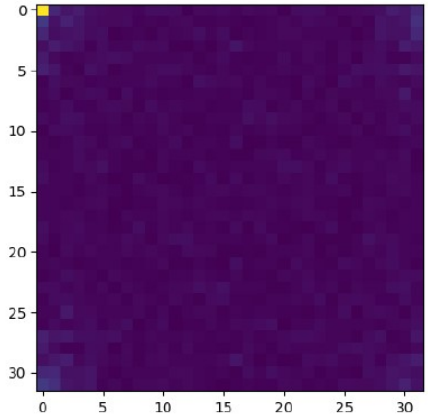

Figure 41: Output of Kernel DFT

8.4 Conclusion: We couldn't get output for 256 *256 image even after waiting for more than an hour.128x128 runs in 5-10 seconds with Row-Column, almost instantly with Kernel and around 15-20 minutes with Conventional. 32x32 runs in a few seconds, 64x64 takes around a minute (with Conventional)

Experiment 9

9.1 Problem Statement: Write a generalized function to perform Radon transform of a 3* 3 matrix.

9.2 Python Code:

```
#importing libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2

def Radon_Transform(img):
    height,width = np.shape(img)
    H=2*height-1
    W= 2*width-1
    new_image = np.zeros( (H ,W))

    new_image[1:4 ,0 ] = np.sum( img , axis=0)
    image  =  np.concatenate( (img , np.zeros( (2,3)) ), axis=0)
    for i in range( 0 , 5):
        new_image[ i , 1] = image[ i][0]+ image[i-1][1] + ...
            image[i-2][2]
    new_image[1:4 ,2 ] = np.sum( img , axis=1)
    for i in range( 0 , 5):
        new_image[ i , 3] = image[ i][2]+ image[i-1][1] + ...
            image[i-2][0]
    new_image[1:4 ,4 ] = np.sum( img , axis=0)

    return new_image


if __name__ == '__main__':
    img = np.array([
            [1,1,1],
            [2,0,2] ,
            [3,3,6]
    ])

print(Radon_Transform(img))
```

```
26 if __name__ == '__main__':
27     img = np.array([
28             [1,1,1],
29             [2,0,2] ,
30             [3,3,6]
31     ])
32
33 print(Radon_Transform(img))
34
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
C:\Users\91726\Desktop\review later\OpenCV>python -u "c:\Users\91726\Desktop\review later\OpenCV\A9.py"
[[ 0.  1.  0.  1.  0.]
 [ 6.  3.  3.  3.  6.]
 [ 4.  4.  4.  7.  4.]
 [ 9.  5. 12.  5.  9.]
 [ 0.  6.  0.  3.  0.]]
```

Figure 42:

9.3 Results:

9.4 Conclusion: The Radon transform is an integral transform whose inverse is used to reconstruct images from medical CT scans. To represent an image, the radon function takes multiple, parallel-beam projections of the image from different angles by rotating the source around the center of the image.

Experiment 10

10.1 Problem Statement: Write a generalized function to perform two-level Discrete Wavelet Transform (DWT) of a gray image with Haar wavelet.

10.2 Python Code:

```
729 from PIL import Image
730 import numpy as np
731 import matplotlib.pyplot as plt
732 import cv2
733
734
735 def DWT_Haar( path):
736     img = Image.open(path)
737     img = np.array( img).astype( np.float64)
738
739     h , w = np.shape( img)[0] , np.shape( img)[1]
740     L = np.zeros( (h, w//2) )
741     H = np.zeros( (h, w//2) )
742
743     for i in range( 0,h ,1 ):
744         for j in range( 0 ,w,2):
745             L[ i , j//2] = img[i,j] + img[ i ,j+1]
746             H[ i , j//2] = img[i,j] - img[ i ,j+1]
747
748     img_ = np.concatenate( (L,H) , axis=1)
749     img = img_.T
750     for i in range( 0,h ,1 ):
751         for j in range( 0 ,w,2):
752             L[ i , j//2] = img[i,j] + img[ i ,j+1]
753             H[ i , j//2] = img[i,j] - img[ i ,j+1]
754
755     img_ = np.concatenate( (L,H) , axis=1)
756     img_ = img_.T.astype( np.float64)
757
758     return img_[ 0: i//2+1 , 0:j//2 +1] ,  img_[(i//2) +1 :, ...
        :j//2+1 ], img_[ 0: i//2 +1 , (j//2) +1 :]  ,   ...
        img_[(i//2) +1 :, (j//2) +1: ]
759
760 if __name__ == "__main__":
761
762     img = "cameraman.png"
763     LL,LH,HL,HH = DWT_Haar(img)
764
765     plt.subplot(2,2,1)
```

```
766     plt.title("LL")
767     plt.imshow(LL,cmap='gray')#input image
768     plt.subplot(2,2,2)
769     plt.title("HL")
770     plt.imshow(HL,cmap='gray')#input image
771     plt.subplot(2,2,3)
772     plt.title("LH")
773     plt.imshow(LH,cmap='gray')#input image
774     plt.subplot(2,2,4)
775     plt.title("HH")
776     plt.imshow(HH,cmap='gray')#input image
777     plt.show()
```

10.3 Results:

Input

Figure 43:

Output

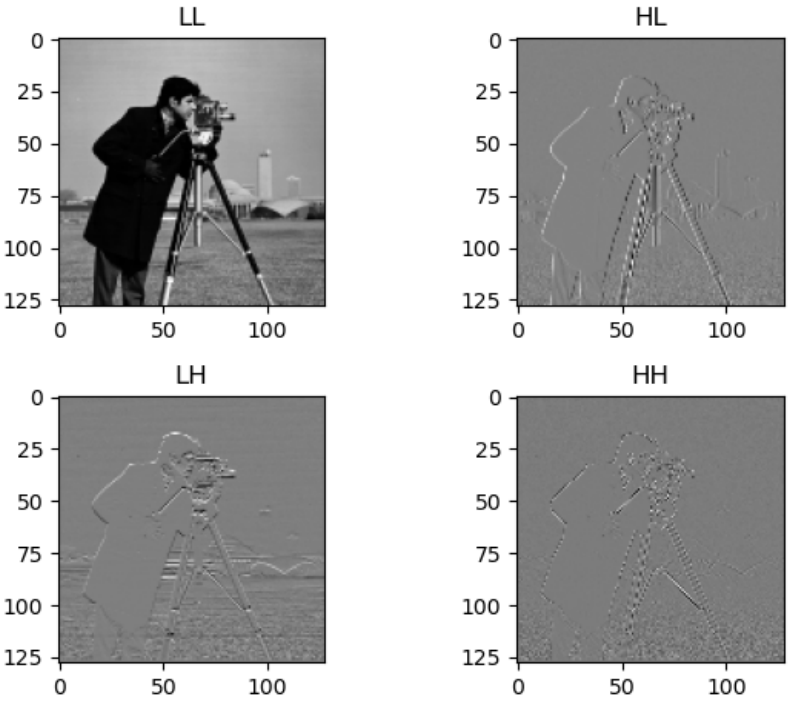

Figure 44:

10.4 Conclusion: Haar wavelet compression is an efficient way to perform both lossless and lossy image compression. The advantages of using DWT over the DFT lies in the fact that the DWT projects high-detail image components onto shorter basis functions with higher resolution, while lower detail components are projected onto larger basis functions.