

ECE 6258 Digital Image Processing

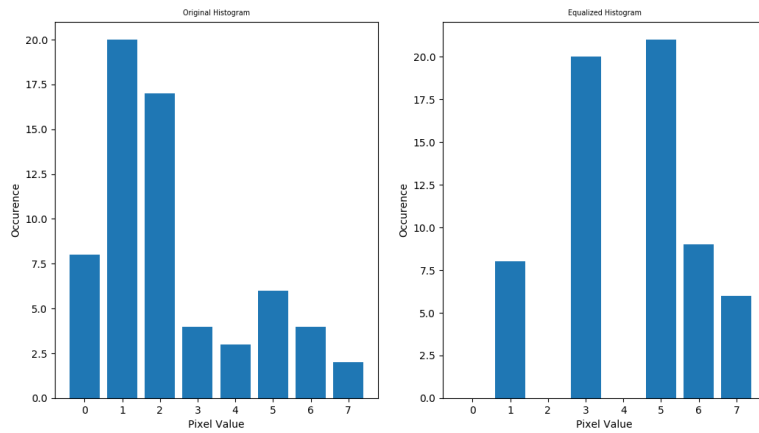
Problem Set 7

Harsh Bhate
Mail: bhate@gatech.edu

Problem 2.

r_k	n_k	$p_r(r_k) = \frac{n_k}{MN}$	$s_k = 7 \sum_k p_r(r_k)$	Rounded s_k	$p_s(\tilde{s}_k)$
0	8	0.125	0.875	1	0.125
1	20	0.3125	3.0625	3	0.3125
2	17	0.265625	4.921875	5	0.328125
3	4	0.0625	5.359375	5	0.328125
4	3	0.046875	5.6875	6	0.140625
5	6	0.09375	6.34375	6	0.140625
6	4	0.0625	6.78125	7	0.09375
7	2	0.03125	7	7	0.09375

Histograms



Problem 4.

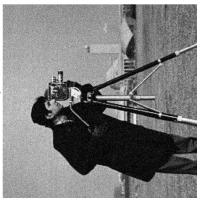
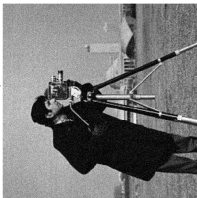
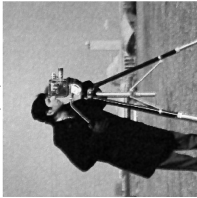
MSE Values and output

```
The MSE for Gaussian Transform is 368.47.
```

```
The MSE for Sigma Transform is 355.20.
```

```
The MSE for Bilateral Transform is 368.47397.
```

Upon observing the MSE error and Visual Results, it is evident that the Sigma transform did the best job at denoising. The Sigma filter smoothened a lot of edges while significantly reducing the noise. The Gaussian Filter and Bilateral Filter ensured good edges but also contained lot of noise as evident in the north half of the image. The Gaussian filter does the best job at maintaining details as evident by the grass patches on the images.



Code

```
1 import dippykit as dip
2 import numpy as np
3 from skimage.restoration import denoise_bilateral as bilateral
4 from skimage.filters import gaussian
5
6 #Loading the image and assigning to f
7 f = dip.im_read("cameraman.tif")
8 f = dip.im_to_float(f)
9
10 #Assigning the noise to g
11 SIGMA = 20/255
12 g = dip.image_noise(f, mode='gaussian', mean=0, var=(SIGMA**2))
13
14
15 #Finding the Gaussian Noise
16 LEN, WIDTH = g.shape
17 SIZE = 6
18 sigma = 1/255
19 TEENY = 0.000000000000001
20 gaussianImage = np.zeros(g.shape)
21 # gaussianImage = gaussian(g,\
22 #                             sigma=1.0,\
23 #                             mode='wrap',\
24 #                             multichannel=False)
25 for n in range(LEN):
26     for m in range(WIDTH):
27         coeffSum = 0
28         outputSum = 0
29         for k in range(SIZE):
30             for l in range(SIZE):
31                 xIndex = n - k
32                 yIndex = m - l
33                 if (xIndex < 0):
34                     xIndex = LEN + xIndex
35                 if (yIndex < 0):
36                     yIndex = WIDTH + yIndex
37                 coeff = (k**2+l**2)/(2*(sigma**2))
38                 h_kl = np.exp(-coeff)
39                 coeffSum = coeffSum + h_kl
40                 outputSum = outputSum + h_kl*g[xIndex,yIndex]
41             gaussianImage[n,m] = (outputSum+TEENY)/(coeffSum+TEENY)
42 #Displaying the Images
43 dip.figure()
44 dip.subplot(1, 4, 1)
45 dip.imshow(f, 'gray')
46 dip.title('Original Image', fontsize='x-small')
47 dip.subplot(1, 4, 2)
48 dip.imshow(gaussianImage, 'gray')
49 dip.title('Gaussian Filtering', fontsize='x-small')
50 #Printing the MSE
51 mseGaussian = dip.metrics.MSE(f*255, gaussianImage*255)
52 print("\n\tThe MSE for Gaussian Transform is %.2f.\n"%mseGaussian)
53
54 #Computing the Sigma Filter
55 SIZE = 6
```

```

56 p = 40/255
57 TEENY = 0.00000000000001
58 sigmaImage = np.zeros(g.shape)
59 for n in range(LEN):
60     for m in range(WIDTH):
61         coeffSum = 0
62         outputSum = 0
63         for k in range(SIZE):
64             for l in range(SIZE):
65                 xIndex = n - k
66                 yIndex = m - l
67                 if (xIndex < 0):
68                     xIndex = LEN + xIndex
69                 if (yIndex < 0):
70                     yIndex = WIDTH + yIndex
71                 coeff = (g[xIndex,yIndex]**2 - g[n,m])**2
72                 coeff = coeff/(2*(p**2))
73                 h_kl = np.exp(-coeff)
74                 coeffSum = coeffSum + h_kl
75                 outputSum = outputSum + h_kl*g[xIndex,yIndex]
76             sigmaImage[n,m] = (outputSum+TEENY)/(coeffSum+TEENY)
77 # Displaying the Images
78 dip.subplot(1, 4, 3)
79 dip.imshow(sigmaImage, 'gray')
80 dip.title('Sigma Filtering', fontsize='x-small')
81 #Printing the MSE
82 mseSigma = dip.metrics.MSE(f*255, sigmaImage*255)
83 print ("\n\tThe MSE for Sigma Transform is %.2f.\n"%mseSigma)
84
85
86
87 #Computing the Bilateral Filter
88 SIZE = 12
89 p = 50.0/255
90 sigma = 2/255
91 TEENY = 0.00000000000001
92 bilateralImage = np.zeros(g.shape)
93
94 # bilateralImage = bilateral(g,\
95 #                             win_size=SIZE, \
96 #                             sigma_spatial=1.0,\
97 #                             mode='symmetric',\
98 #                             multichannel=False)
99
100 for n in range(LEN):
101     for m in range(WIDTH):
102         coeffSum = 0
103         outputSum = 0
104         for k in range(SIZE):
105             for l in range(SIZE):
106                 xIndex = n - k
107                 yIndex = m - l
108                 if (xIndex < 0):
109                     xIndex = LEN + xIndex
110                 if (yIndex < 0):
111                     yIndex = WIDTH + yIndex
112                 coeff_1 = (g[xIndex,yIndex]**2 - g[n,m])**2

```

```

113         coeff_1 = coeff_1/(2*(p**2))
114         coeff_2 = (k**2+l**2)/(2*(sigma**2))
115         h_kl = np.exp(-coeff_1)*np.exp(-coeff_2)
116         coeffSum = coeffSum + h_kl
117         outputSum = outputSum + h_kl*g[xIndex,yIndex]
118         bilateralImage[n,m] = (outputSum+TEENY)/(coeffSum+TEENY)
119 #Printing the MSE
120 mseBilateral = dip.metrics.MSE(f*255, bilateralImage*255)
121 print ("\n\tThe MSE for Bilateral Transform is %.5f.\n"%
        mseBilateral)
122 # Displaying the Images
123 dip.subplot(1, 4, 4)
124 dip.imshow(bilateralImage, 'gray')
125 dip.title('Bilateral Filtering', fontsize='x-small')
126 dip.show()

```

Problem 5.

Code

```
1 import dipykit as dip
2 import numpy as np
3
4 #Resolution Guide
5 #4CIF Resolution = 704 x 480
6 #CIF Resolution = 352 x 240
7
8 X = dip.im_read("coatOfArms.png")
9 X = dip.im_to_float(X)
10 X = X*255
11
12 #Filter
13 SIZE = 25
14 filterSize = (SIZE, SIZE)
15 minor = 5**2
16 mild = 25**2
17 severe = 100**2
18
19 minorWindow = dip.window_2d (filterSize , window_type='gaussian' ,
20                               variance = minor )
21 mildWindow = dip.window_2d (filterSize , window_type='gaussian' ,
22                               variance = mild )
23 severeWindow = dip.window_2d (filterSize , window_type='gaussian' ,
24                               variance = severe )
25
26 #Output of the Filter
27 minorX = dip.convolve2d(X, minorWindow, mode = 'same', boundary = '
28 wrap')
29 mildX = dip.convolve2d(X, mildWindow, mode = 'same', boundary = '
30 wrap')
31 severeX = dip.convolve2d(X, severeWindow, mode = 'same', boundary = '
32 wrap')
33
34 #Laplacian Filtering
35 LaplaceKernel = np.array ([[0,1,0] ,
36                             [1,-4,1] ,
37                             [0,1,0]] ,
38                             dtype=np.float)
39
40 edgeMinorX = dip.convolve2d(X, LaplaceKernel, mode = 'same',
41                               boundary = 'wrap')
42 edgeMildX = dip.convolve2d(X, LaplaceKernel, mode = 'same',
43                               boundary = 'wrap')
44 edgeSevereX = dip.convolve2d(X, LaplaceKernel, mode = 'same',
45                               boundary = 'wrap')
46
47 #PSNR Calculations
48
49 PSNR_minor = dip.metrics.PSNR(dip.float_to_im(X/255), dip.
50                               float_to_im(edgeMinorX/255))
51 PSNR_mild = dip.metrics.PSNR(dip.float_to_im(X/255), dip.
52                               float_to_im(edgeMildX/255))
53 PSNR_severe = dip.metrics.PSNR(dip.float_to_im(X/255), dip.
54                               float_to_im(edgeSevereX/255))
```

```

43
44 #Displaying PSNR Output
45 print ("\n\tPSNR for Laplacian with minor blurring is %.2f \n" %
    PSNR_minor)
46 print ("\n\tPSNR for Laplacian with mild blurring is %.2f \n" %
    PSNR_mild)
47 print ("\n\tPSNR for Laplacian with severe blurring is %.2f \n" %
    PSNR_severe)
48
49 #Extended Laplacian Filtering
50 extLaplaceKernel = np.array([[1,1,1],
51                               [1,-8,1],
52                               [1,1,1]],
53                               dtype=np.float)
54
55 extEdgeMinorX = dip.convolve2d(X, extLaplaceKernel, mode = 'same',
    boundary = 'wrap')
56 extEdgeMildX = dip.convolve2d(X, extLaplaceKernel, mode = 'same',
    boundary = 'wrap')
57 extEdgeSevereX = dip.convolve2d(X, extLaplaceKernel, mode = 'same',
    boundary = 'wrap')
58
59 #PSNR Calculations
60 PSNR_minor = dip.metrics.PSNR(dip.float_to_im(X/255), dip.
    float_to_im(extEdgeMinorX/255))
61 PSNR_mild = dip.metrics.PSNR(dip.float_to_im(X/255), dip.
    float_to_im(extEdgeMildX/255))
62 PSNR_severe = dip.metrics.PSNR(dip.float_to_im(X/255), dip.
    float_to_im(extEdgeSevereX/255))
63
64 #Displaying PSNR Output
65 print ("\n\tPSNR for extended Laplacian with minor blurring is %.2f
    \n" %PSNR_minor)
66 print ("\n\tPSNR for extended Laplacian with mild blurring is %.2f
    \n" %PSNR_mild)
67 print ("\n\tPSNR for extended Laplacian with severe blurring is %.2
    f \n" %PSNR_severe)
68
69 #Image Output
70 dip.figure()
71 dip.subplot(3, 4, 1)
72 dip.imshow(X, 'gray')
73 dip.title('Original Image', fontsize='x-small')
74 dip.subplot(3, 4, 2)
75 dip.imshow(minorX, 'gray')
76 dip.title('Minor Blurring', fontsize='x-small')
77 dip.subplot(3, 4, 3)
78 dip.imshow(mildX, 'gray')
79 dip.title('Mild Blurring', fontsize='x-small')
80 dip.subplot(3, 4, 4)
81 dip.imshow(severeX, 'gray')
82 dip.title('Severe Blurring', fontsize='x-small')
83 dip.subplot(3, 4, 5)
84 dip.imshow(edgeMinorX, 'gray')
85 dip.title('Laplacian on Minor Blurring', fontsize='x-small')
86 dip.subplot(3, 4, 6)
87 dip.imshow(edgeMildX, 'gray')

```



```

88 dip.title('Laplacian on Mild Blurring', fontsize='x-small')
89 dip.subplot(3, 4, 7)
90 dip.imshow(edgeSevereX, 'gray')
91 dip.title('Laplacian on Severe Blurring', fontsize='x-small')
92 dip.subplot(3, 4, 8)
93 dip.imshow(extEdgeMinorX, 'gray')
94 dip.title('Extended Laplacian on Minor Blurring', fontsize='x-small')
95 dip.subplot(3, 4, 9)
96 dip.imshow(extEdgeMildX, 'gray')
97 dip.title('Extended Laplacian on Mild Blurring', fontsize='x-small')
98 dip.subplot(3, 4, 10)
99 dip.imshow(extEdgeSevereX, 'gray')
100 dip.title('Extended Laplacian on Severe Blurring', fontsize='x-small')
101 dip.show()

```

Outputs

```

PSNR for Laplacian with minor blurring is 27.73
PSNR for Laplacian with mild blurring is 27.73
PSNR for Laplacian with severe blurring is 27.73

PSNR for extended Laplacian with minor blurring is 27.78
PSNR for extended Laplacian with mild blurring is 27.78
PSNR for extended Laplacian with severe blurring is 27.78

```

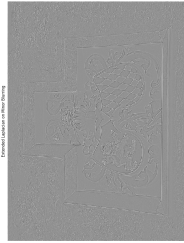
Observations

It can be observed that as blurring increases, the Laplacian filter loses track of fine edge details while maintaining structural edge details. Often, weakly blurred images leave a lot of loose edge details. At the same time, a strongly blurred image loses a lot of finer details.

It was also observed that an extended Laplacian filter provides a much finer edge details as compared to regular Laplacian filter.



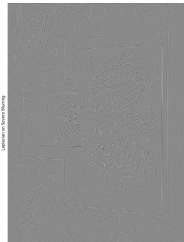
Stadler, 1870/71



Stadler, 1870/71



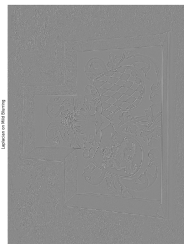
Stadler, 1870/71



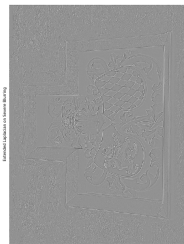
Stadler, 1870/71



Stadler, 1870/71



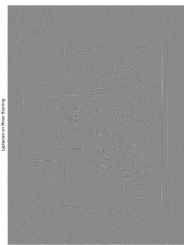
Stadler, 1870/71



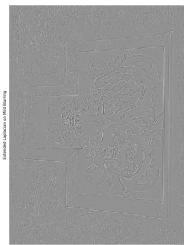
Stadler, 1870/71



Stadler, 1870/71



Stadler, 1870/71



Stadler, 1870/71

Problem 7.

The edge case of the matrix are the elements a and i . The operations on these pixels will appear as follows:

$$a = \frac{1}{4} \cdot i + \frac{1}{2} \cdot a + \frac{1}{4} \cdot b$$

$$i = \frac{1}{4} \cdot j + \frac{1}{2} \cdot i + \frac{1}{4} \cdot a$$

For all other cases, three successive elements will be added. Thus, each row of a column will have circular symmetry at the edge cases. Thus, the filter H is:

$$\mathbf{H} = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & \frac{1}{2} \end{bmatrix}$$

The output of the filter for a 3×3 image f vectorized as 9×1 is given by:

$$\hat{f} = H \cdot f$$

where $\hat{f} \in R^{9 \times 1}$, $H \in R^{9 \times 9}$ and $f \in R^{9 \times 1}$.

Problem 9.

f. True and Estimated Variance Values

Image Number	1	2	3	4	5
σ^2	0.018	0.027	0.032	0.036	0.059
$\widehat{\sigma_f^2}$	0.00125	0.01031	0.015029	0.01898	0.04368
$\widehat{\sigma_t^2}$	0.0104	0.0191	0.02411	0.02862	0.0437

Code

```

1
2 import numpy as np
3 import dippykit as dip
4
5 def noise_estimation(im: np.ndarray):
6     """
7     Given an image, this function determines the variance in its
8     pixel
9     values and displays a histogram of the image along with a
10    fitted normal
11    distribution.
12    """
13    # Calculate histogram
14    im_hist, bin_edges = np.histogram(im, 30)
15    bin_centers = bin_edges[:-1] + (np.diff(bin_edges) / 2)
16    # Normalize histogram to make a PDF
17    im_hist = im_hist.astype(float)
18    im_hist /= np.sum(im_hist)
19    # Calculate the mean and variance values
20    mean = np.sum(im_hist * bin_centers)
21    var = np.sum(im_hist * (bin_centers ** 2)) - (mean ** 2)
22    # Calculate the values on the normal distribution PDF
23    norm_vals = np.exp(-((bin_centers - mean) ** 2) / (2 * var)) \
24                / np.sqrt(2 * np.pi * var)
25    # Normalize the norm_vals
26    norm_vals /= np.sum(norm_vals)
27    # Rescale variance
28    var /= (255 ** 2)
29    print('Variance: {}'.format(var))
30    dip.figure()
31    dip.bar(bin_centers, im_hist)
32    dip.plot(bin_centers, norm_vals, 'r')
33    dip.legend(['Fitted Gaussian PDF', 'Histogram of image'])
34    dip.xlabel('Pixel value')
35    dip.ylabel('Occurrence')
36    dip.show()
37
38 def main():
39     ##### PART (a): EDIT HERE #####
40     img = img = dip.im_read("WiseonRocks_noise_1.png")
41     # img = dip.im_to_float(img)
42     # img = img*255
43     print("The shape of IMG is %s \n" %str(img.shape))
44     ##### PART (b): EDIT HERE #####

```

```

43     noise_estimation(img)
44
45     ##### PART (c)-(e): EDIT HERE #####
46     SIZE = 100
47     print ("Image 1")
48     flatRegion = img[0:SIZE, 100:(100+SIZE)]
49     notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
50     noise_estimation(flatRegion)
51     noise_estimation(notFlatRegion)
52
53     print ("Image 2")
54     img = img = dip.im_read("WiseonRocks_noise_2.png")
55     flatRegion = img[0:SIZE, 100:(100+SIZE)]
56     noise_estimation(img)
57     notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
58     noise_estimation(flatRegion)
59     noise_estimation(notFlatRegion)
60
61     print ("Image 3")
62     img = img = dip.im_read("WiseonRocks_noise_3.png")
63     flatRegion = img[0:SIZE, 100:(100+SIZE)]
64     noise_estimation(img)
65     notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
66     noise_estimation(flatRegion)
67     noise_estimation(notFlatRegion)
68
69     print ("Image 4")
70     img = img = dip.im_read("WiseonRocks_noise_4.png")
71     flatRegion = img[0:SIZE, 100:(100+SIZE)]
72     noise_estimation(img)
73     notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
74     noise_estimation(flatRegion)
75     noise_estimation(notFlatRegion)
76
77     print ("Image 5")
78     img = img = dip.im_read("WiseonRocks_noise_5.png")
79     flatRegion = img[0:SIZE, 100:(100+SIZE)]
80     noise_estimation(img)
81     notFlatRegion = img[250:(250+SIZE), 150:(150+SIZE)]
82     noise_estimation(flatRegion)
83     noise_estimation(notFlatRegion)
84
85 if __name__ == '__main__':
86     main()

```