EECE5639-HW2

October 6, 2021

##Computer Vision Homework #2 #Noise, filters, corners

[110]:
```
# import libraries
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
```

#Problem #1 Use a program of your choice (Matlab, your own, etc) to generate ten 256x256 images with gray level 128 corrupted with additive, zero-mean Gaussian noise with standard deviation 2.0. Use the procedures EST NOISE in Chapter 2 of Trucco and Verri (see attached pdf), to estimate the noise in the images. Discuss your results. Filter the generated noisy images generated with a 3x3 box filter. Use EST NOISE to estimate the noise of the output images. Discuss your results.

[111]:
```
# Create 10 images with grayscale value 128
num = 10
images = 128*np.ones([256,256,num])

# Add noise to the images
sigma = 2
mean = 0
noise = sigma*np.random.randn(256,256,num) + mean
images = images+noise
```

[112]:
```
# Signal to noise ratio
avg_image = np.mean(images,axis=2)
avg_images = np.repeat(avg_image[:, :, np.newaxis], num, axis=2)

EST_NOISE = np.sqrt( 1/(num-1)*np.sum( (avg_images-images)**2, axis=2) )
avg_EST_NOISE = np.mean(EST_NOISE)
max_EST_NOISE = np.amax(EST_NOISE)

print('The average noise is:', np.round(avg_EST_NOISE,3))
print('The max noise is:', np.round(max_EST_NOISE,3))
```

```
The average noise is: 1.945
The max noise is: 4.099
```

[113]:
```
# Smooth images with 3x3 box filter
images_smoothed = cv.blur(images, (3,3))
```

```
avg_image_smoothed = np.mean(images_smoothed,axis=2)
avg_images_smoothed = np.repeat(avg_image_smoothed[:, :, np.newaxis], num,␣
 ↪axis=2)

EST_NOISE_smoothed = np.sqrt( 1/(num-1)*np.sum(␣
 ↪(avg_images_smoothed-images_smoothed)**2, axis=2) )
avg_EST_NOISE_smoothed = np.mean(EST_NOISE_smoothed)
max_EST_NOISE_smoothed = np.amax(EST_NOISE_smoothed)

print('The average noise is:', np.round(avg_EST_NOISE_smoothed,3))
print('The max noise is:', np.round(max_EST_NOISE_smoothed,3))
```

```
The average noise is: 0.651
The max noise is: 1.481
```

#Problem #2 Generate a 2D Gaussian filter mask with standard deviation 1.4. Find two equivalent 1-D masks so that they can be used to smooth images applying the separability property.

[114]:
```
sigma = 1.4
factor = 5
window = np.int(np.floor(sigma*factor))
gauss2d = np.ones([window,window])
offset = np.floor(window/2)
for i in range(window):
    for j in range(window):
        gauss2d[i,j] = np.exp( -1/(2*sigma**2)*( (i-offset)**2 + (j-offset)**2␣
 ↪) )
gauss2d = gauss2d/np.sum(gauss2d)

diag = gauss2d*np.identity(gauss2d.shape[0])
diag = diag[diag>0]
gauss1d = np.sqrt(diag)
print('The 1D gaussian filter is:', np.around(gauss1d, 3) )
```

```
The 1D gaussian filter is: [0.029 0.104 0.223 0.288 0.223 0.104 0.029]
```

#Problem #3 Consider the following two averaging filters used to clean images:

[115]:
```
img = np.array([10.,10.,10.,10.,10.,40.,40.,40.,40.,40.])
pad = np.hstack([0.,0.,img,0.,0.])
avg_const = 1/5*np.array([1,1,1,1,1])

output = np.zeros(img.shape)
for i in range(img.size):
  output[i] = np.dot(pad[i:i+avg_const.size], avg_const)
print(output)

avg_gauss = 1/10*np.array([1,2,4,2,1])
output = np.zeros(img.shape)
```

```
for i in range(img.size):
  output[i] = np.dot(pad[i:i+avg_gauss.size], avg_gauss)
print(output)
```

```
[ 6.  8. 10. 16. 22. 28. 34. 40. 32. 24.]
[ 7.  9. 10. 13. 19. 31. 37. 40. 36. 28.]
```

#Problem #4 An image contains a thin vertical line, one pixel thick. It has a gray level of 50 and lies on a background of salt and pepper noise having gray values of 100 and 0, respectively, where the probability(pepper) = 0.3 and the probability(salt)=0.7, and the probability of salt and pepper are independent of each other. A horizontal 1x3 operator with values (-1,2,-1) is applied to this image. Find the probability distribution for the output values when the operator is centered at a pixel: on the line adjacent to the line
Probability Distribution
-100 —> 9%
0 —> 42%
100 —> 49%

[116]:
```
r = 10000
c = 4
salt_pepper = 100*np.random.choice(np.arange(0, 2), r*c, p=[0.3, 0.7])
salt_pepper = np.reshape(salt_pepper, [r,c])

image = np.hstack([salt_pepper, 50*np.ones([r,1]), salt_pepper])
#plt.imshow(image)
#plt.show()

operator = np.array([-1,2,-1])
operator = np.reshape(operator,[1,3])
output = cv.filter2D(image, -1, operator)
#plt.imshow(output)
#plt.show()

line = output[:,c]
#plt.hist(line, [-100, 0, 100, 200])
#plt.show()

# Operator input possibilities
options = np.array([[0, 50, 0],
                    [0, 50, 100],
                    [100, 50, 0],
                    [100, 50, 100]])
print(options)

# Operator output possibilities
print(np.sum(operator*options,axis=1))

# Probabilities
```

```
chances = np.array([0.3**2, 0.3*0.7, 0.7*0.3, 0.7**2])

# Total Prob
total = np.sum(0.3**2+2*0.3*0.7+0.7**2)

one = sum(line==100)/line.size;
zero = sum(line==0)/line.size;
negone = sum(line==-100)/line.size;
print("Chances of 100 are: ", one)
print("Chances of 0 are: ", zero)
print("Chances of -100 are: ", negone)
```

```
[[  0  50   0]
 [  0  50 100]
 [100  50   0]
 [100  50 100]]
[ 100    0    0 -100]
Chances of 100 are:  0.093
Chances of 0 are:  0.4222
Chances of -100 are:  0.4848
```

#Problem #5 An 8x8 image I(i, j) has pixel values following the equation I(i, j) = |i − j| with i, j = 0, 1, . . . , 7 Find the output image obtained by applying a 3 × 3 median filter on the image I. (Assume that the border pixels are not changed)

[117]:
```
fxy = np.zeros([8,8])
for i in range(8):
  for j in range(8):
    fxy[i,j] = abs(i-j)

fxy = np.hstack((np.zeros([fxy.shape[0],2]), fxy, np.zeros([fxy.shape[0],2])))
fxy = np.vstack((np.zeros([2,fxy.shape[1]]), fxy, np.zeros([2,fxy.shape[1]])))
fxy = np.uint8(fxy)
fxy_med = cv.medianBlur(fxy,3)

print(fxy_med[2:-2,2:-2])
```

```
[[0 0 1 2 3 4 5 0]
 [0 1 1 2 3 4 5 5]
 [1 1 1 1 2 3 4 4]
 [2 2 1 1 1 2 3 3]
 [3 3 2 1 1 1 2 2]
 [4 4 3 2 1 1 1 1]
 [5 5 4 3 2 1 1 0]
 [0 5 4 3 2 1 0 0]]
```

#Problem #6 Consider a 1D step profile f (i) = 4 i [0, 3] 8 i [4, 7] Work out the result of applying the median filtering with n = 3 and compare the result with the output of filtering with the averaging mask 1/4[1 2 1].

[118]:
```
img = np.array([4.,4.,4.,4.,8.,8.,8.,8.])
pad = np.hstack([0.,img,0.])
avg_const = 1/5*np.array([1,1,1])

pad = np.uint8(pad)
output = cv.medianBlur(pad,3)
output = np.transpose(output[1:-1])
print(output)

avg_gauss = 1/4*np.array([1,2,1])
output = np.zeros(img.shape)
for i in range(img.size):
  output[i] = np.dot(pad[i:i+avg_gauss.size], avg_gauss)
print(output)
```

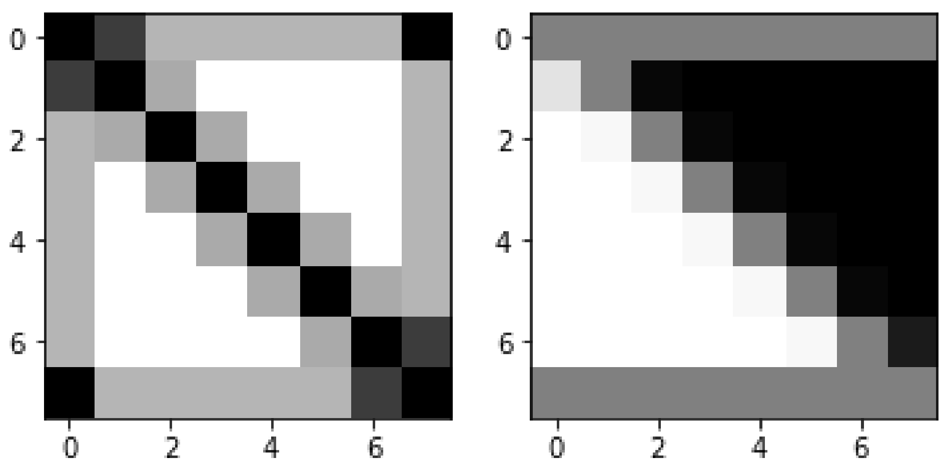
```
[[4 4 4 4 8 8 8 8]]
[3. 4. 4. 5. 7. 8. 8. 6.]
```

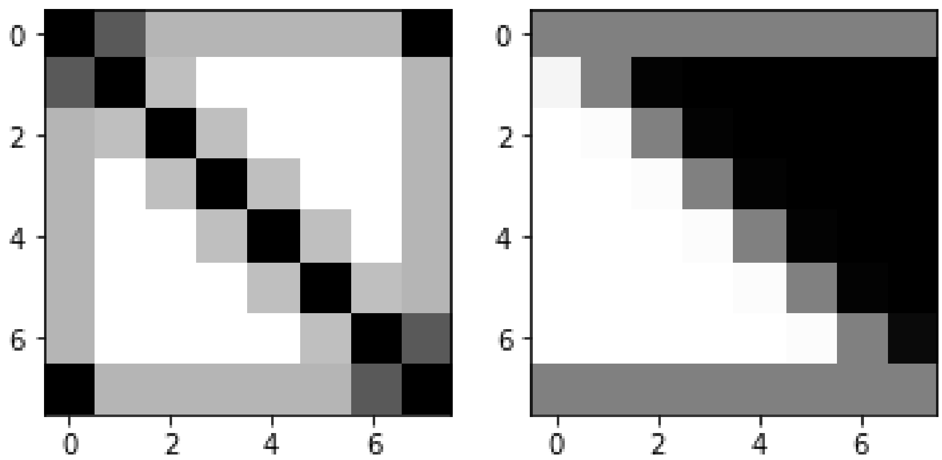
#Problem #7 Consider a 8x8 image f(x,y) such that f (x, y) = |x − y| for x, y = 0, 1, . . . , 7 . (a) Find the magnitude and orientation of the gradient by using the Prewitt maks. (Disregard boundaries) (b) Repeat using the Sobel masks.

[119]:
```
idx = 8
fxy = np.zeros([idx,idx])
for i in range(idx):
  for j in range(idx):
    fxy[i,j] = i-j
fxy = np.abs(fxy)
print(fxy)

Prewittx = np.array([[-1,0,1],
                     [-1,0,1],
                     [-1,0,1]])
Prewitty = np.array([[-1,-1,-1],
                     [0,0,0],
                     [1,1,1]])
Px = cv.filter2D(fxy,-1,Prewittx)
Py = cv.filter2D(fxy,-1,Prewitty)

mag = np.sqrt(Px**2+Py**2)
theta = np.arctan(Py,Px)

plt.subplot(1,2,1)
plt.imshow(mag, cmap='gray')
plt.subplot(1,2,2)
plt.imshow(theta, cmap='gray')
plt.show()
```

```
Sx = cv.Sobel(fxy,cv.CV_64F,1,0,ksize=3)
Sy = cv.Sobel(fxy,cv.CV_64F,0,1,ksize=3)

mag = np.sqrt(Sx**2+Sy**2)
theta = np.arctan(Sy,Sx)

plt.subplot(1,2,1)
plt.imshow(mag, cmap='gray')
plt.subplot(1,2,2)
plt.imshow(theta, cmap='gray')
plt.show()

print(np.around(Sx,3))
print(np.around(Sy,3))
```

```
[[0. 1. 2. 3. 4. 5. 6. 7.]
 [1. 0. 1. 2. 3. 4. 5. 6.]
 [2. 1. 0. 1. 2. 3. 4. 5.]
 [3. 2. 1. 0. 1. 2. 3. 4.]
 [4. 3. 2. 1. 0. 1. 2. 3.]
 [5. 4. 3. 2. 1. 0. 1. 2.]
 [6. 5. 4. 3. 2. 1. 0. 1.]
 [7. 6. 5. 4. 3. 2. 1. 0.]]
```



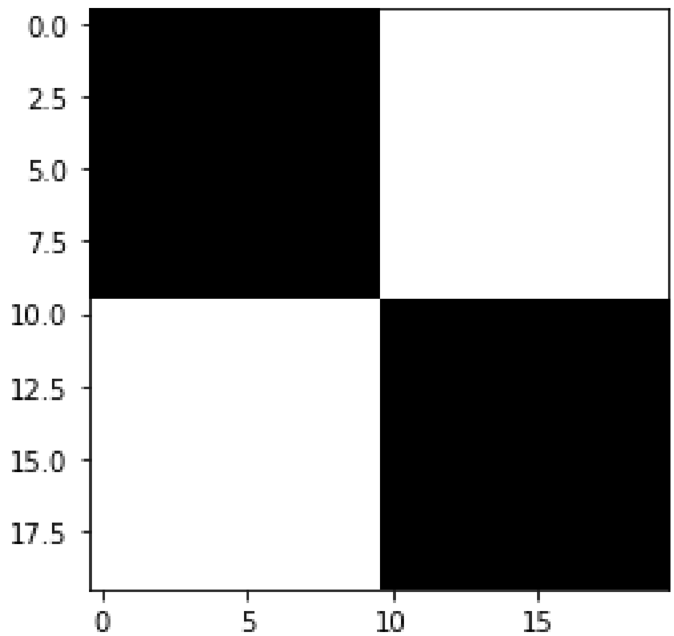


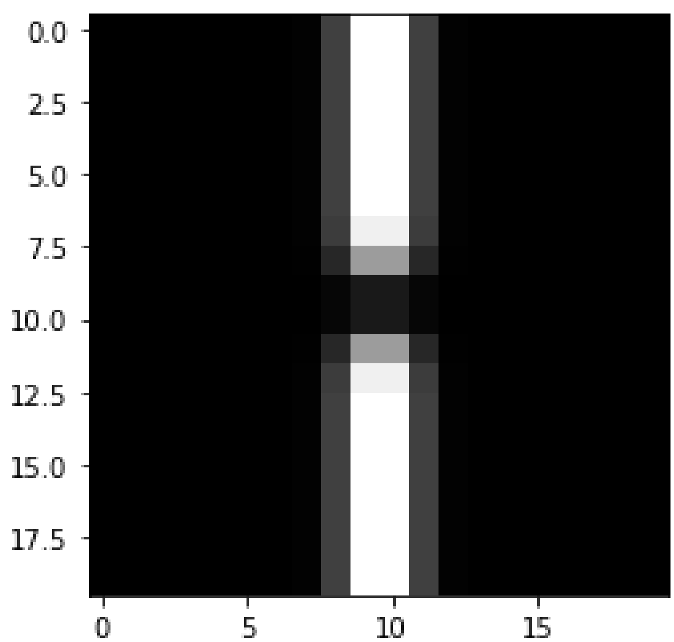
```
[[ 0.     0.     0.     0.     0.     0.     0.     0.   ]
 [ 1.326  0.    -1.406 -1.446 -1.446 -1.446 -1.446 -1.446]
 [ 1.446  1.406  0.    -1.406 -1.446 -1.446 -1.446 -1.446]
 [ 1.446  1.446  1.406  0.    -1.406 -1.446 -1.446 -1.446]
 [ 1.446  1.446  1.446  1.406  0.    -1.406 -1.446 -1.446]
 [ 1.446  1.446  1.446  1.446  1.406  0.    -1.406 -1.446]
 [ 1.446  1.446  1.446  1.446  1.446  1.406  0.    -1.326]
 [ 0.     0.     0.     0.     0.     0.     0.     0.   ]]
[[ 0.  0.  0.  0.  0.  0.  0.  0.]
 [ 4.  0. -6. -8. -8. -8. -8. -8.]
 [ 8.  6.  0. -6. -8. -8. -8. -8.]
 [ 8.  8.  6.  0. -6. -8. -8. -8.]
 [ 8.  8.  8.  6.  0. -6. -8. -8.]
 [ 8.  8.  8.  8.  6.  0. -6. -8.]
 [ 8.  8.  8.  8.  8.  6.  0. -4.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.]]
```

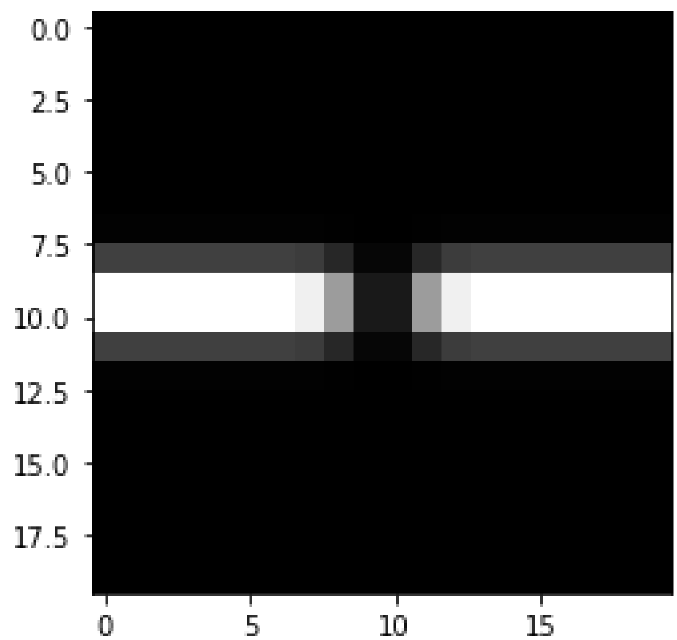
#Problem #8 Compute the C matrix and find its eigenvalues, to detect the corner in the image given in Figure 1, when using a neighborhood of 7×7.

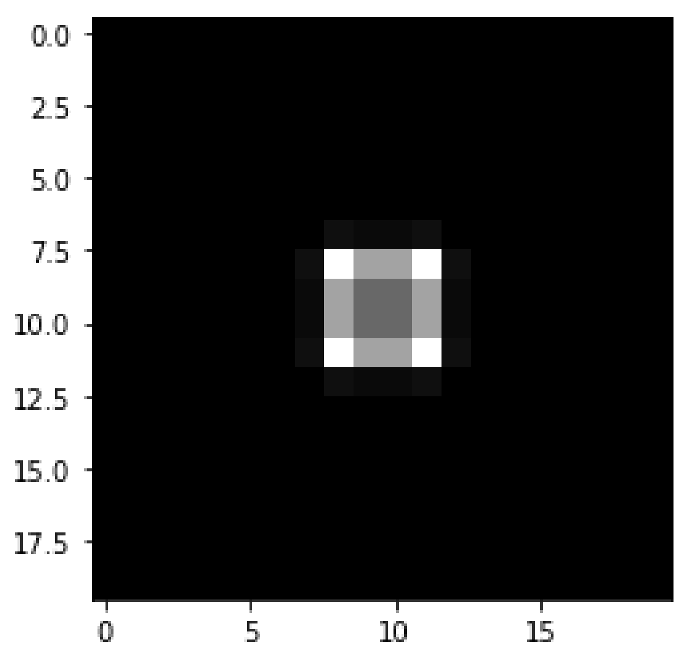
[120]:
```
image = np.hstack([np.zeros([10,10]), 40*np.ones([10,10])])
checker = np.vstack([image, np.fliplr(image)])
plt.imshow(checker, cmap='gray', vmin=0, vmax=40)
plt.show()

Sx = cv.Sobel(checker,cv.CV_64F,1,0,ksize=7)
Sy = cv.Sobel(checker,cv.CV_64F,0,1,ksize=7)

R = (Sx**2 * Sy**2) - 0.05*(Sx**2 + Sy**2)
plt.imshow(Sx**2, cmap='gray')
plt.show()
```

```
plt.imshow(Sy**2, cmap='gray')
plt.show()
plt.imshow(R, cmap='gray')
plt.show()
```






0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15


0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5
10
15