assignment05

October 18, 2018

This script demonstrates Computation of Image Features using Convolution

Name : Ji-Su Lee

Student ID : 20141718

github link : https://github.com/Jisu-Lee/HII

import packages for plotting graphs and manipulating data:

```
In [30]: import matplotlib.pyplot as plt
         import numpy as np
         from numpy import linalg as LA
         from scipy import signal
         from skimage import io, color
         from skimage import exposure
```

define image gradient kernel in x-direction

$$gradKerX = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{1}$$

```
In [31]: gradKerX = np.array([[1, 0, -1],
                               [1, 0, -1],
                               [1, 0, -1]])
```

define image gradient kernel in y-direction

$$gradKerY = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2}$$

```
In [88]: gradKerY = np.array([[1, 1, 1],
                               [0, 0, 0],
                               [-1, -1, -1]])
```

define smoothing kernel

$$smoothKer = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

```
In [33]: smoothKer = np.array([[1/9, 1/9, 1/9],
                               [1/9, 1/9, 1/9],
                               [1/9, 1/9, 1/9]])
```

define my own kernel

$$myKer = \begin{bmatrix} -1 & 2 & 3 \\ 4 & 5 & -6 \\ 7 & 8 & -9 \end{bmatrix} \tag{4}$$

```
In [89]: myKer = np.array([[-1, 2, 3],
                           [4, 5, -6],
                           [7, 8, -9]])
```

define function to compute magnitude of gradient

$$G = \sqrt{G_x^2 + G_y^2} \tag{5}$$

```
In [90]: def computeMagnitude(gx, gy):
             res = []
             for i in range(len(gx)):
                 res.append([])
                 for j in range(len(gx[0])):
                     t = [gx[i][j], gy[i][j]]
                     res[i].append(LA.norm(t))
             return res
```

define function to compute direction of gradient(returned value is radian)

$$\Theta = atan(\frac{G_y}{G_x}) \tag{6}$$

Note that there are error handlings for x = 0 either $\frac{\pi}{2}$ or $-\frac{\pi}{2}$ is stored at **res**

```
In [57]: def computeDirection(gx, gy):
             res = []
             for i in range(len(gx)):
                 res.append([])
                 for j in range(len(gx[0])):
                     x = gx[i][j]
                     y = gy[i][j]
                     # for this case, append pi/2 radian as a result
                     if(x == 0 and y >= 0):
```

```
                    res[i].append(1.57079633)
                # append -(pi/2) radian as a result
                elif(x == 0 and y < 0):
                    res[i].append(-1.57079633)
                else:
                    res[i].append(np.arctan(gy[i][j]/gx[i][j]))
        return res
```

load image and convert to grayscale image

```
In [48]: file_image        = 'cau.jpg'

         im_color         = io.imread(file_image)
         im_gray          = color.rgb2gray(im_color)
```

First, let's apply derivative kernerls, both x-direction and y-direction

```
In [49]: gradX = signal.convolve2d(im_gray, gradKerX, boundary='symm', mode='same')
         gradY = signal.convolve2d(im_gray, gradKerY, boundary='symm', mode='same')
```

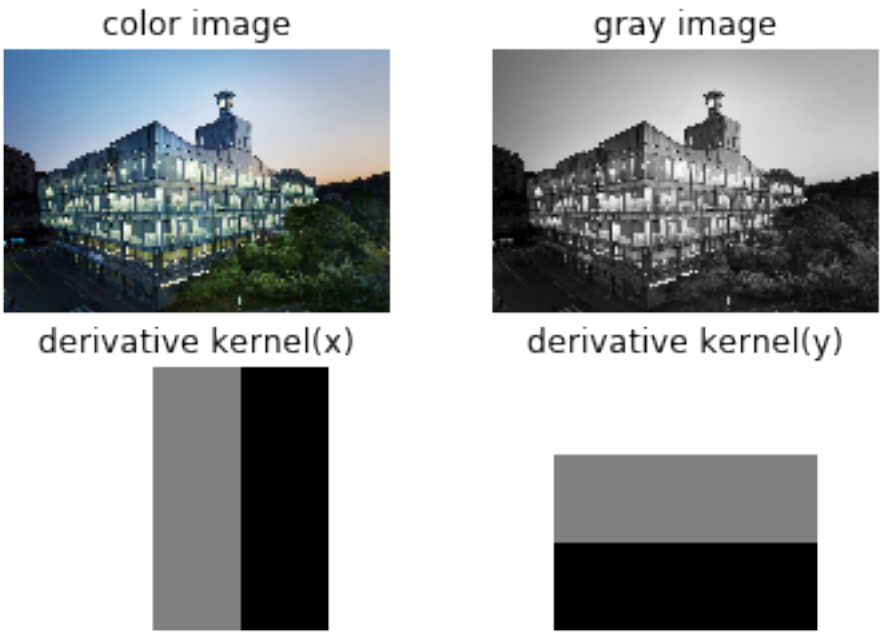
Let's calculate magnitude and direction

```
In [58]: gradMag = computeMagnitude(gradX, gradY)
         gradDir = computeDirection(gradX, gradY)
```

Now let's see results...

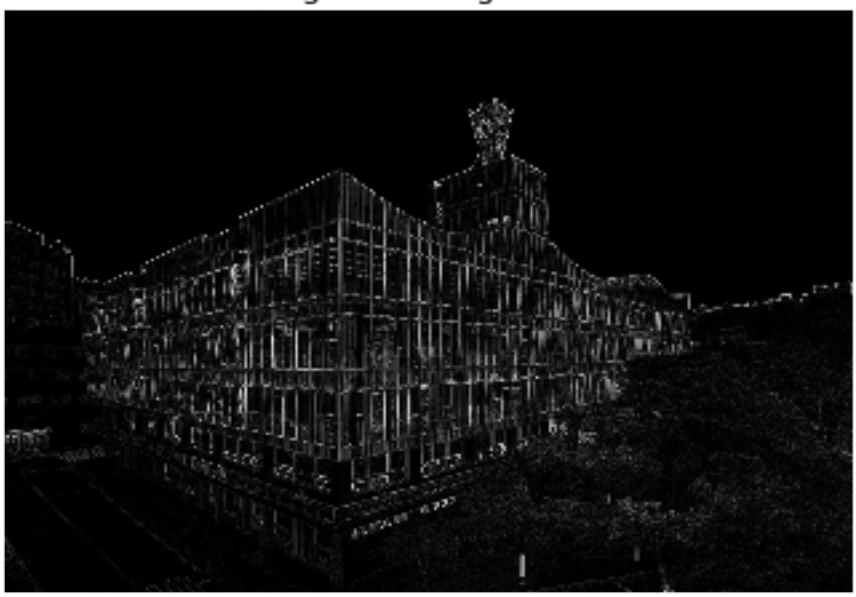
```
In [71]: # input color image
         p1 = plt.subplot(2,2,1)
         p1.set_title('color image')
         plt.imshow(im_color)
         plt.axis('off')

         # input gray image
         p2 = plt.subplot(2,2,2)
         p2.set_title('gray image')
         plt.imshow(im_gray, cmap='gray')
         plt.axis('off')

         # derivative in x-direction
         p3 = plt.subplot(2,2,3)
         p3.set_title('derivative kernel(x)')
         plt.imshow(gradKerX, cmap='gray')
         plt.axis('off')

         # derivative in y-direction
         p4 = plt.subplot(2,2,4)
         p4.set_title('derivative kernel(y)')
```

```
plt.imshow(gradKerY, cmap='gray')
plt.axis('off')

plt.show()
```

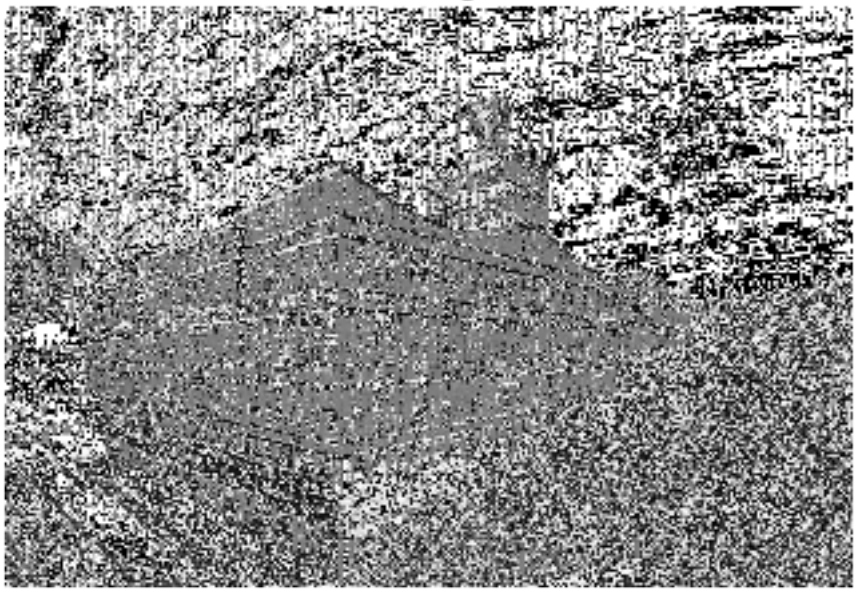


In [72]:
```
# magnitude of gradient
p5 = plt.subplot()
p5.set_title('magnitude of gradient')
plt.imshow(gradMag, cmap='gray')
plt.axis('off')

plt.show()

# direction of gradient
p6 = plt.subplot()
p6.set_title('direction of gradient')
plt.imshow(gradDir, cmap='gray')
plt.axis('off')

plt.show()
```

magnitude of gradient



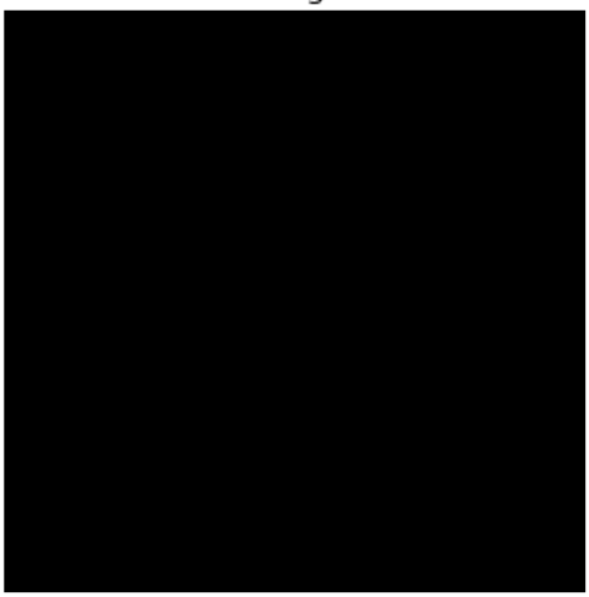
direction of gradient



Next, let's smooth out the image

In [74]:
```
# smoothing image
smoothImg = signal.convolve2d(im_gray, smoothKer, boundary='symm', mode='same')

# shows smoothing kernel
p8 = plt.subplot()
p8.set_title('smoothing kernel')
plt.imshow(smoothKer, cmap='gray')
plt.axis('off')

plt.show()

# shows original image
p7 = plt.subplot()
p7.set_title('original image')
plt.imshow(im_gray, cmap='gray')
plt.axis('off')

plt.show()

# shows smoothed image
p8 = plt.subplot()
p8.set_title('smoothed image')
plt.imshow(smoothImg, cmap='gray')
plt.axis('off')

plt.show()
```

smoothing kernel

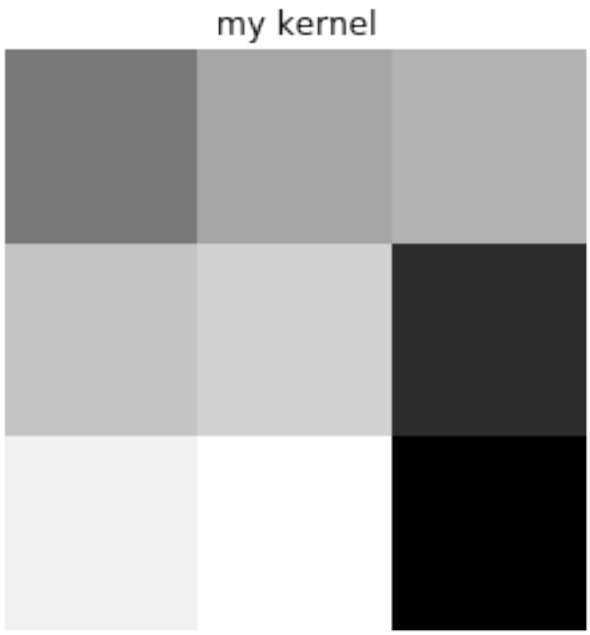


original image

This is the image with my own kernel

```
In [87]: # apply my kernel to image
         myImg = signal.convolve2d(im_gray, myKer, boundary='symm', mode='same')

         # shows smoothing kernel
         p9 = plt.subplot()
         p9.set_title('my kernel')
         plt.imshow(myKer, cmap='gray')
         plt.axis('off')

         plt.show()

         # shows original image
         p10 = plt.subplot()
         p10.set_title('original image')
         plt.imshow(im_gray, cmap='gray')
         plt.axis('off')

         plt.show()

         # shows smoothed image
         p11 = plt.subplot()
         p11.set_title('my image')
```

```
plt.imshow(myImg, cmap='gray')
plt.axis('off')

plt.show()
```



The picture seems quite old...