Computer Vision HW8

R07921010 余奇安

1.Add Gaussian noise

Gaussian noise with amp10	Gaussian noise with amp30
	
SNR 13.61	SNR 4.049

2.Add salt&pepper noise

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
SNR 1.0278	SNR -1.8038

3. 3x3 box filter to deal with 4 noise image

Gaussian noise with amp10	Gaussian noise with amp30
	
SNR 16.37	SNR 12.07

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
SNR 9.32	SNR 6.5

3. 5x5 box filter to deal with each case

Gaussian noise with amp10	Gaussian noise with amp30
	
SNR 13.59	12.41

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
SNR 10.62	SNR 8.40

4. 3x3 median filter

Gaussian noise with amp10	Gaussian noise with amp30
	
10.10	8.14

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
10.17	9.47

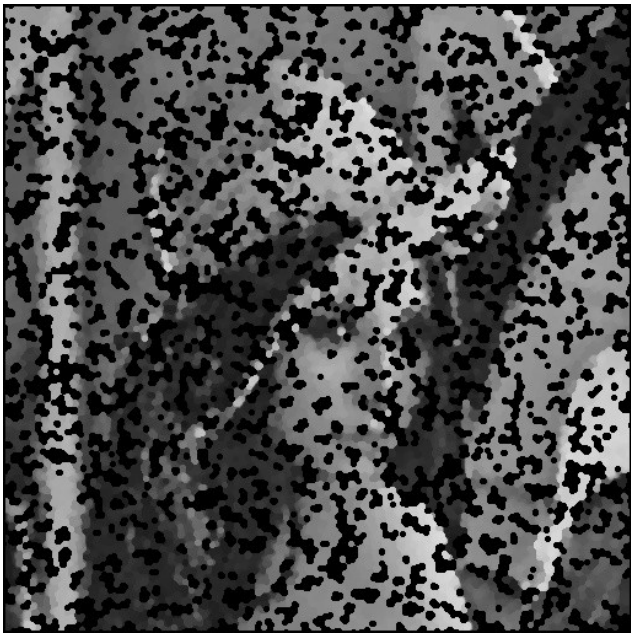
5. 5x5 median filter

Gaussian noise with amp10	Gaussian noise with amp30
	
15.85	12.55

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
15.87	14.16

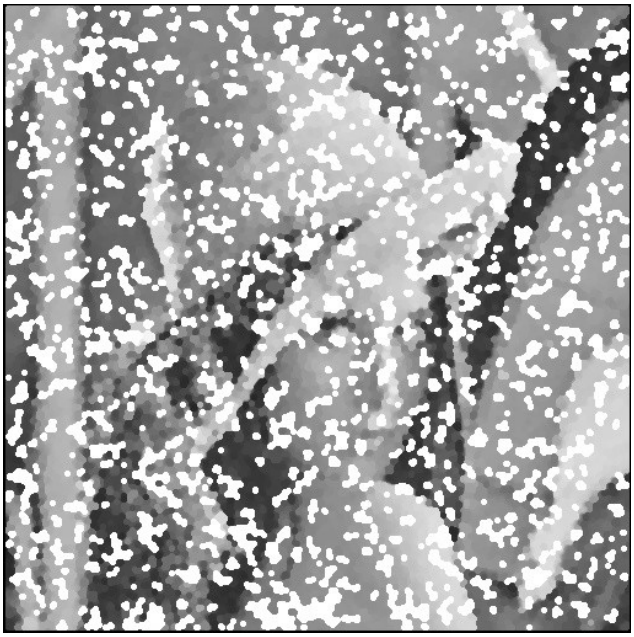
6.opening closing filter

Gaussian noise with amp10	Gaussian noise with amp30
	
8.61	8.63

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
3.99	-2.22

7. closing and opening filter

Gaussian noise with amp10	Gaussian noise with amp30
	
7.65	6.09

Salt & Pepper with threshold 0.05	Salt & Pepper with threshold 0.1
	
4.35	-2.22

Discussion:
We can find that some SNR becomes negative value. It is because the noise bigger than signal.
Opening closing / Closing opening filter can deal with small noise both on SP and Gaussian, however, it will damage the image if the noise is too big.
Generally speaking, the 5x5 median filter can deal with the noise the best. It is because the median filter is good at dealing with the out peak value.

Code:
Use np.random.noraml to generate Gaussian noise
Use np.random.uniform to generate Salt&pepper noise
both with average 0 ,stand deviation 1

```
def Gaussian_noise(img,amp):
    return img+amp*np.random.normal(0,1,(x,y))

def SP_noise(img,threshold):
    row,col=img.shape
    new=np.zeros(shape=(row,col))
    for i in range(row):
        for j in range(col):
            if np.random.uniform(0,1)<threshold:
                new[i][j]=0
            elif np.random.uniform(0,1)>1-threshold:
                new[i][j]=255
            else:
                new[i][j]=img[i][j]
    return new
```

SNR function:

```
def SNR(c1,n2):
    row,col=c1.shape
    cs=0
    ns=0
    for i in range(row):
        for j in range(col):
            cs+=c1[i][j]
            ns+=(n2[i][j]-c1[i][j])
    ms=cs/(row*col)
    mn=ns/(row*col)
    #print(ms,mn)
    VS=0
    VN=0
    for i in range(row):
        for j in range(col):
            VS+=(c1[i][j]-ms)**2
            VN+=(n2[i][j]-c1[i][j]-mn)**2
    #print(VS,VN)
    VS=VS/(row*col)
    VN=VN/(row*col)

    #print(VS,VN)
    ans=20*math.log10((VS**0.5)/(VN**0.5))
    return ans
```

Filters

```
def box_filter3x3(img):
    row,col=img.shape
    new=np.zeros(shape=(row+2,col+2))
    res=np.zeros(shape=(row+2,col+2))
    fres=np.zeros(shape=(row,col))
    for i in range(row):
        for j in range(col):
            new[i+1][j+1]=img[i][j]
    box=[(0,0),(0,1),(0,-1),(-1,-1),(-1,0),(-1,1),(1,-1),(1,0),(1,1)]
    for i in range(1,row+1):
        for j in range(1,col+1):
            s=0
            for m,n in box:
                #print(i+m,j+n)
                s+=new[i+m][j+n]
            res[i][j]=s/9
    for i in range(row):
        for j in range(col):
            fres[i][j]=res[i+1][j+1]
    return fres
```

```
def box_filter5x5(img):
    row,col=img.shape
    new=np.zeros(shape=(row+4,col+4))
    res=np.zeros(shape=(row+4,col+4))
    fres=np.zeros(shape=(row,col))
    for i in range(row):
        for j in range(col):
            new[i+2][j+2]=img[i][j]
    box=[]
    for i in range(-2,3):
        for j in range(-2,3):
            box.append((i,j))
    #print box
    #box=[(0,0),(0,1),(0,-1),(0,2),(0,-2),(-1,-2),(-1,-1),(-1,0),(-1,1),(1,
    for i in range(2,row+2):
        for j in range(2,col+2):
            s=0
            for m,n in box:
                #print(i+m,j+n)
                s+=new[i+m][j+n]
            res[i][j]=s/25

    for i in range(row):
        for j in range(col):
            fres[i][j]=res[i+2][j+2]
    return fres
```

```python
def median_filter3x3(img):
    row,col=img.shape
    new=np.zeros(shape=(row+2,col+2))
    res=np.zeros(shape=(row+2,col+2))
    fres=np.zeros(shape=(row,col))
    for i in range(row):
        for j in range(col):
            new[i+1][j+1]=img[i][j]
    box=[(0,0),(0,1),(0,-1),(-1,-1),(-1,0),(-1,1),(1,-1),(1,0),(1,1)]
    for i in range(1,row+1):
        for j in range(1,col+1):
            s=[]
            for m,n in box:
                s.append(new[i+m][j+n])
            s.sort()
            res[i+1][j+1]=s[4]
    for i in range(row):
        for j in range(col):
            fres[i][j]=res[i+1][j+1]
    return fres
```

```python
def median_filter5x5(img):
    row,col=img.shape
    new=np.zeros(shape=(row+4,col+4))
    res=np.zeros(shape=(row+4,col+4))
    fres=np.zeros(shape=(row,col))
    for i in range(row):
        for j in range(col):
            new[i+2][j+2]=img[i][j]
    box=[]
    for i in range(-2,3):
        for j in range(-2,3):
            box.append((i,j))

    for i in range(2,row+2):
        for j in range(2,col+2):
            s=[]
            for m,n in box:
                s.append(new[i+m][j+n])
            s.sort()
            res[i][j]=s[12]
    for i in range(row):
        for j in range(col):
            fres[i][j]=res[i+2][j+2]
    return fres
```

Base on dilation and erosion, implement the opening-closing / closing-opening filter

```
def OC_filter(img):
    op=opening(img)
    cl=closing(op)
    return cl
def CO_filter(img):
    cl=closing(img)
    op=opening(cl)
    return op
```