HW8

- **A discription of your homework**

  Programming language used: Python 2.7

  Library used: Numpy, PIL, Scipy.misc

- **Your parameters**

  i: row

  j: column

  tem, temp: 用於儲存每個像素的灰階數值 0~255

  sum: 以(i,j)為中心周圍 3x3 格或 5x5 格內的灰階數值總和

  list: 儲存以(i,j)為中心周圍 3x3 格或 5x5 格內的灰階數值

  kernel: octogonal 3-5-5-5-3 kernel

- **Functions**

  gua: Guassian noise

  salt: salt-and-pepper noise

  box: box filter

  med: median filter

  dil: dilation

  ero: erosion

  opn: opening

  clo: closing

  oc: opening followed by closing

  co: closing followed by opening

- **The algorithm you used**

1. Generate additive white Gaussian noise

   使用 np.random.normal(0,1)來產生 Gaussian noise，震幅分別為 10 和 30。

   $\mathrm{I(i,j) = I(i,j) + amplitude \times N(0,1)}$

   $\mathrm{N(0,1)}$: Guassian random variable with mean 0 and standard deviation 1

2. Generate salt-and-pepper noise

   使用 np.random.uniform(0,1)來產生 salt-and-pepper noise，threshold 分別為 0.05 和 0.1。

   $\mathrm{I(i,j) = 0, if\ uniform(0,1) < threshold}$

   $\mathrm{I(i,j) = 255, if\ uniform(0,1) > threshold}$

   $\mathrm{uniform(0,1)}$: random variable uniformly distributed over [0,1]

3. Run box filter (3X3, 5X5) on all noisy images

   以(i,j)為中心，周圍 3x3 格內的灰階值總和再平均，取代原圖的灰階值。

$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \times \frac{1}{9}$$

以(i,j)為中心，周圍 5x5 格內的灰階值總和再平均，取代原圖的灰階值。

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

$$\times \frac{1}{25}$$

註：邊界沒有處理，放處理前的灰階值。

4. Run median filter (3X3, 5X5) on all noisy images

以(i,j)為中心，周圍 3x3 格內的灰階值的中位數(list.sort 完標號為 4 者)，取代原圖的灰階值。

以(i,j)為中心，周圍 5x5 格內的灰階值的中位數(list.sort 完標號為 12 者)，取代原圖的灰階值。

註：邊界沒有處理，放處理前的灰階值。

5. Run opening followed by closing and closing followed by opening

(1) Dilation

kernel 區域內灰階數值最大者存入該像素。

(2) Erosion

kernel 區域內灰階數值最小者存入該像素。

(3) Opening

$\mathrm{B} \circ \mathrm{K} = (\mathrm{B} \ominus \mathrm{K}) \oplus \mathrm{K}$

先做 erosion，再做 dilation：把 erosion 後的結果丟到 dilation 跑一遍。

(4) Closing

$\mathrm{B} \cdot \mathrm{K} = (\mathrm{B} \oplus \mathrm{K}) \ominus \mathrm{K}$

先做 dilation，再做 erosion：把 dilation 後的結果丟到 erosion 跑一遍。

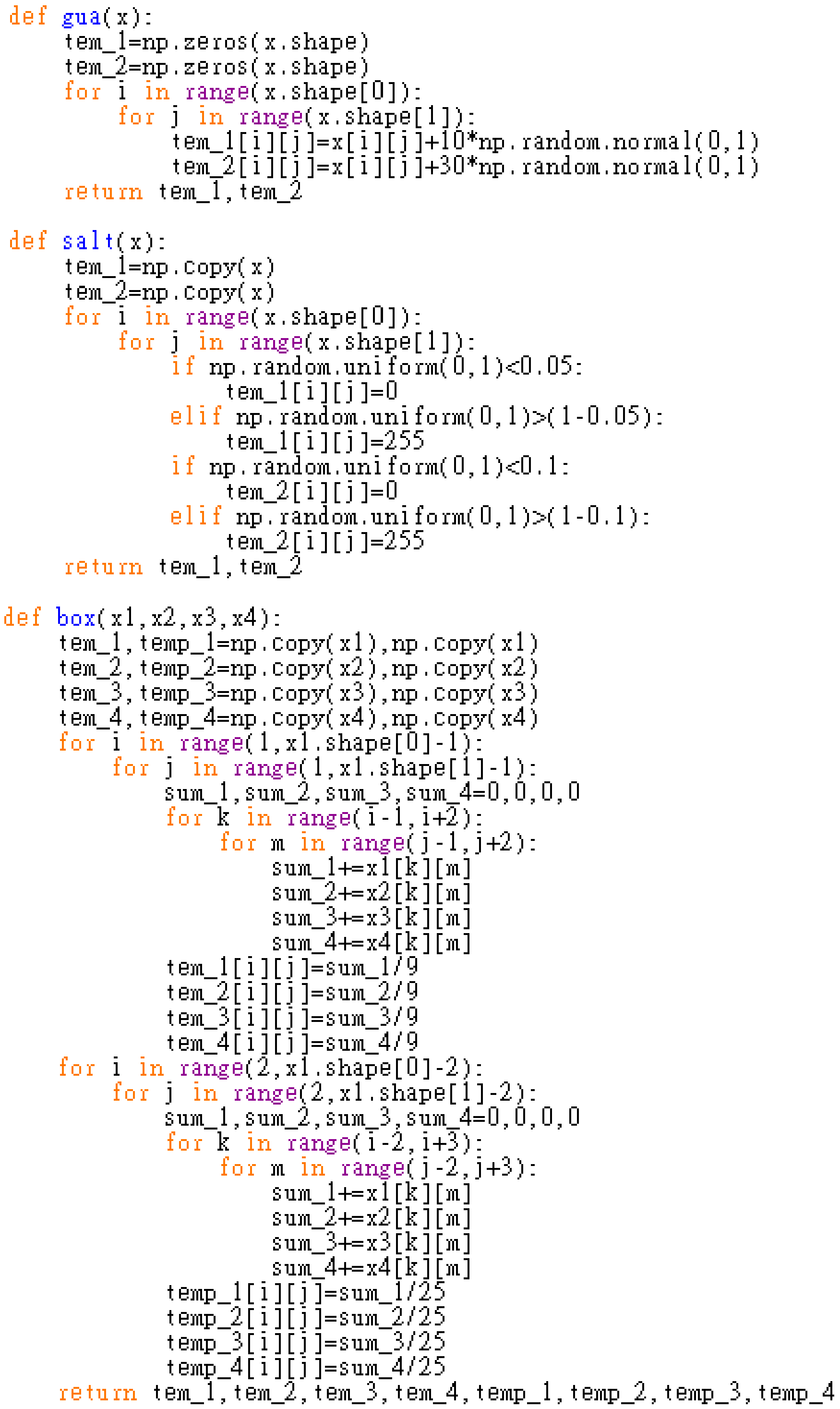
(5) Closing followed by opening

先做 closing，再做 opening：把 closing 後的結果丟到 opening 跑一遍。

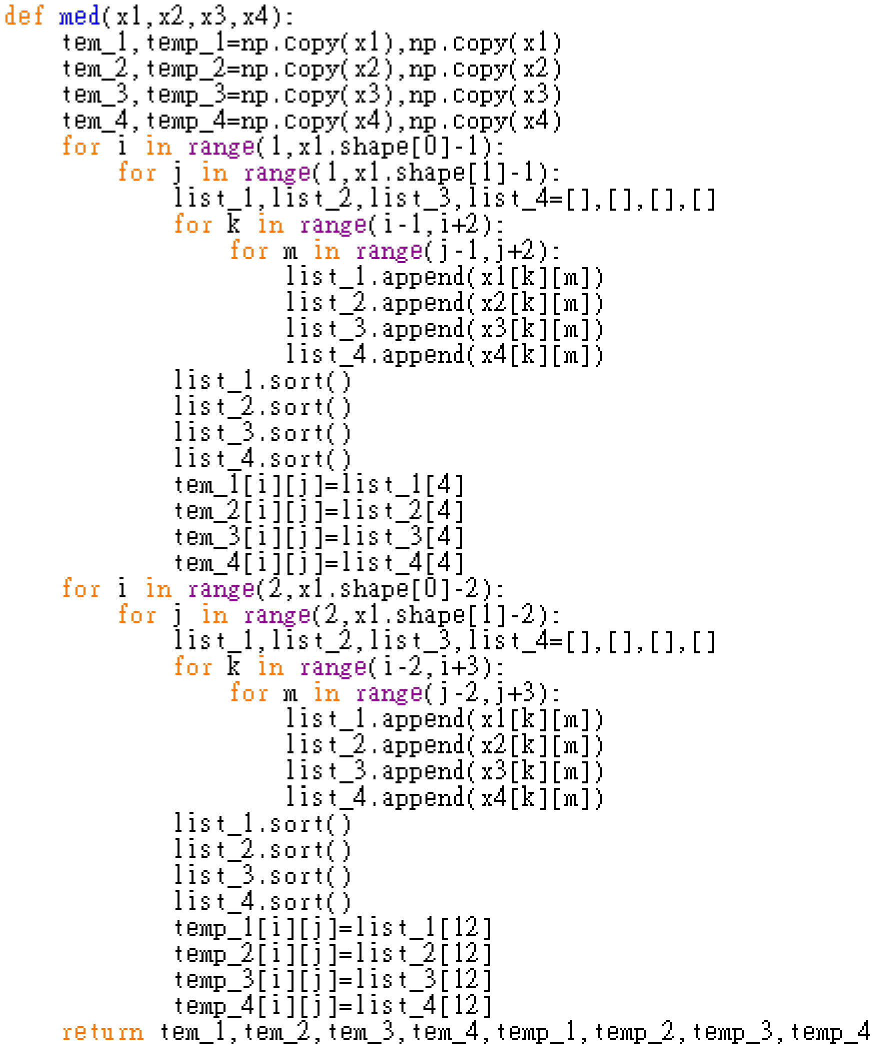
(6) Opening followed by closing

先做 opening，再做 closing：把 opening 後的結果丟到 closing 跑一遍。

- **Principal code fragment**



```
def gua(x):
    tem_1=np.zeros(x.shape)
    tem_2=np.zeros(x.shape)
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            tem_1[i][j]=x[i][j]+10*np.random.normal(0,1)
            tem_2[i][j]=x[i][j]+30*np.random.normal(0,1)
    return tem_1,tem_2

def salt(x):
    tem_1=np.copy(x)
    tem_2=np.copy(x)
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            if np.random.uniform(0,1)<0.05:
                tem_1[i][j]=0
            elif np.random.uniform(0,1)>(1-0.05):
                tem_1[i][j]=255
            if np.random.uniform(0,1)<0.1:
                tem_2[i][j]=0
            elif np.random.uniform(0,1)>(1-0.1):
                tem_2[i][j]=255
    return tem_1,tem_2

def box(x1,x2,x3,x4):
    tem_1,temp_1=np.copy(x1),np.copy(x1)
    tem_2,temp_2=np.copy(x2),np.copy(x2)
    tem_3,temp_3=np.copy(x3),np.copy(x3)
    tem_4,temp_4=np.copy(x4),np.copy(x4)
    for i in range(1,x1.shape[0]-1):
        for j in range(1,x1.shape[1]-1):
            sum_1,sum_2,sum_3,sum_4=0,0,0,0
            for k in range(i-1,i+2):
                for m in range(j-1,j+2):
                    sum_1+=x1[k][m]
                    sum_2+=x2[k][m]
                    sum_3+=x3[k][m]
                    sum_4+=x4[k][m]
            tem_1[i][j]=sum_1/9
            tem_2[i][j]=sum_2/9
            tem_3[i][j]=sum_3/9
            tem_4[i][j]=sum_4/9
    for i in range(2,x1.shape[0]-2):
        for j in range(2,x1.shape[1]-2):
            sum_1,sum_2,sum_3,sum_4=0,0,0,0
            for k in range(i-2,i+3):
                for m in range(j-2,j+3):
                    sum_1+=x1[k][m]
                    sum_2+=x2[k][m]
                    sum_3+=x3[k][m]
                    sum_4+=x4[k][m]
            temp_1[i][j]=sum_1/25
            temp_2[i][j]=sum_2/25
            temp_3[i][j]=sum_3/25
            temp_4[i][j]=sum_4/25
    return tem_1,tem_2,tem_3,tem_4,temp_1,temp_2,temp_3,temp_4
```



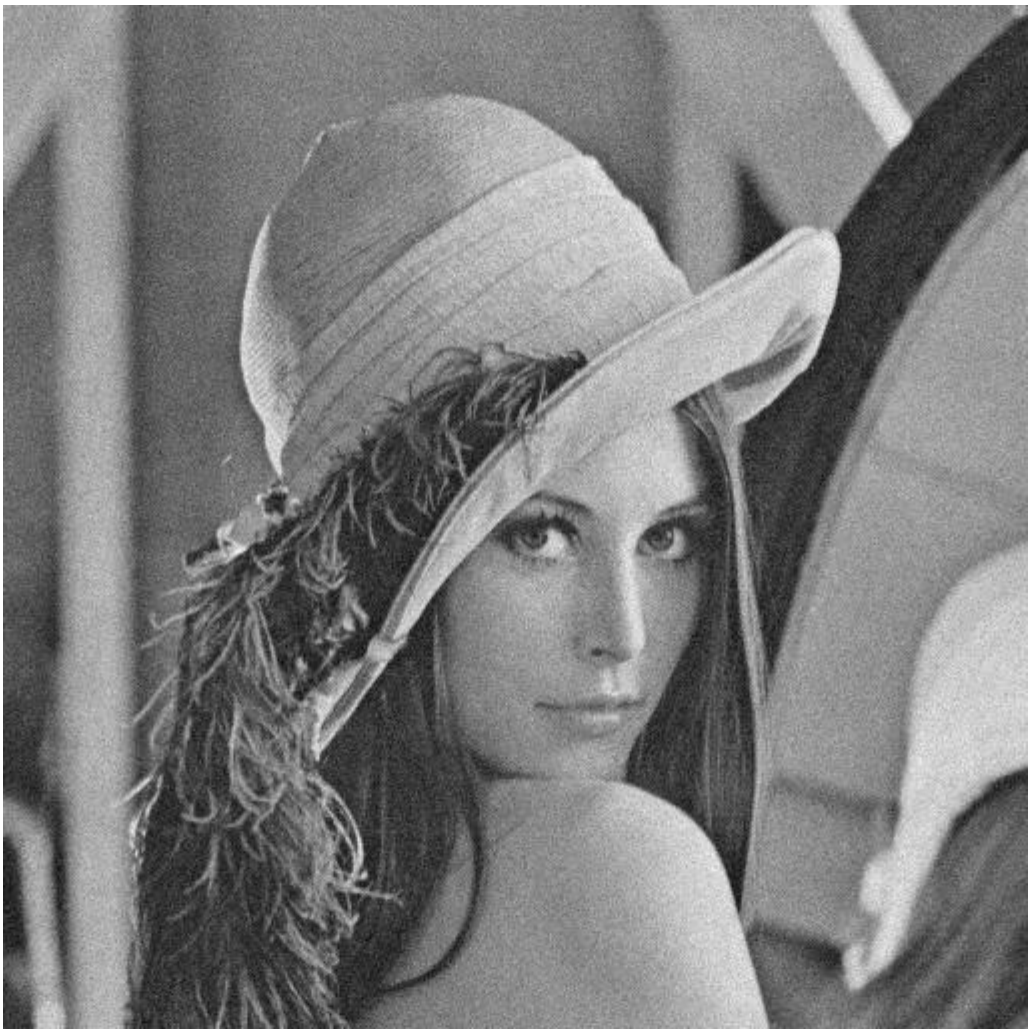
```python
def med(x1,x2,x3,x4):
    tem_1,temp_1=np.copy(x1),np.copy(x1)
    tem_2,temp_2=np.copy(x2),np.copy(x2)
    tem_3,temp_3=np.copy(x3),np.copy(x3)
    tem_4,temp_4=np.copy(x4),np.copy(x4)
    for i in range(1,x1.shape[0]-1):
        for j in range(1,x1.shape[1]-1):
            list_1,list_2,list_3,list_4=[],[],[],[]
            for k in range(i-1,i+2):
                for m in range(j-1,j+2):
                    list_1.append(x1[k][m])
                    list_2.append(x2[k][m])
                    list_3.append(x3[k][m])
                    list_4.append(x4[k][m])
            list_1.sort()
            list_2.sort()
            list_3.sort()
            list_4.sort()
            tem_1[i][j]=list_1[4]
            tem_2[i][j]=list_2[4]
            tem_3[i][j]=list_3[4]
            tem_4[i][j]=list_4[4]
    for i in range(2,x1.shape[0]-2):
        for j in range(2,x1.shape[1]-2):
            list_1,list_2,list_3,list_4=[],[],[],[]
            for k in range(i-2,i+3):
                for m in range(j-2,j+3):
                    list_1.append(x1[k][m])
                    list_2.append(x2[k][m])
                    list_3.append(x3[k][m])
                    list_4.append(x4[k][m])
            list_1.sort()
            list_2.sort()
            list_3.sort()
            list_4.sort()
            temp_1[i][j]=list_1[12]
            temp_2[i][j]=list_2[12]
            temp_3[i][j]=list_3[12]
            temp_4[i][j]=list_4[12]
    return tem_1,tem_2,tem_3,tem_4,temp_1,temp_2,temp_3,temp_4
```

```python
kernel = []
for k in range(-2,3):
    for m in range(-2,3):
        if (k!=-2 or m!=-2) and (k!=-2 or m!=2) and (k!=2 or m!=-2) and (k!=2 or m!=2):
            kernel.append([k,m])

def dil(x):
    tem = np.copy(x)
    for i in range(2,x.shape[0]-2):
        for j in range(2,x.shape[1]-2):
            maxi = 0
            for k in range(len(kernel)):
                if x[i+kernel[k][0]][j+kernel[k][1]]> maxi:
                    maxi = x[i+kernel[k][0]][j+kernel[k][1]]
            tem[i][j] = maxi
    return tem

def ero(x):
    tem = np.copy(x)
    for i in range(2,x.shape[0]-2):
        for j in range(2,x.shape[1]-2):
            mini = 255
            for k in range(len(kernel)):
                if x[i+kernel[k][0]][j+kernel[k][1]]< mini:
                    mini = x[i+kernel[k][0]][j+kernel[k][1]]
            tem[i][j] = mini
    return tem

def opn(x):
    tem = np.copy(dil(ero(x)))
    return tem

def clo(x):
    tem = np.copy(ero(dil(x)))
    return tem

def co(x1,x2,x3,x4):
    tem_1 = np.copy(opn(clo(x1)))
    tem_2 = np.copy(opn(clo(x2)))
    tem_3 = np.copy(opn(clo(x3)))
    tem_4 = np.copy(opn(clo(x4)))
    return tem_1,tem_2,tem_3,tem_4

def oc(x1,x2,x3,x4):
    tem_1 = np.copy(clo(opn(x1)))
    tem_2 = np.copy(clo(opn(x2)))
    tem_3 = np.copy(clo(opn(x3)))
    tem_4 = np.copy(clo(opn(x4)))
    return tem_1,tem_2,tem_3,tem_4
```

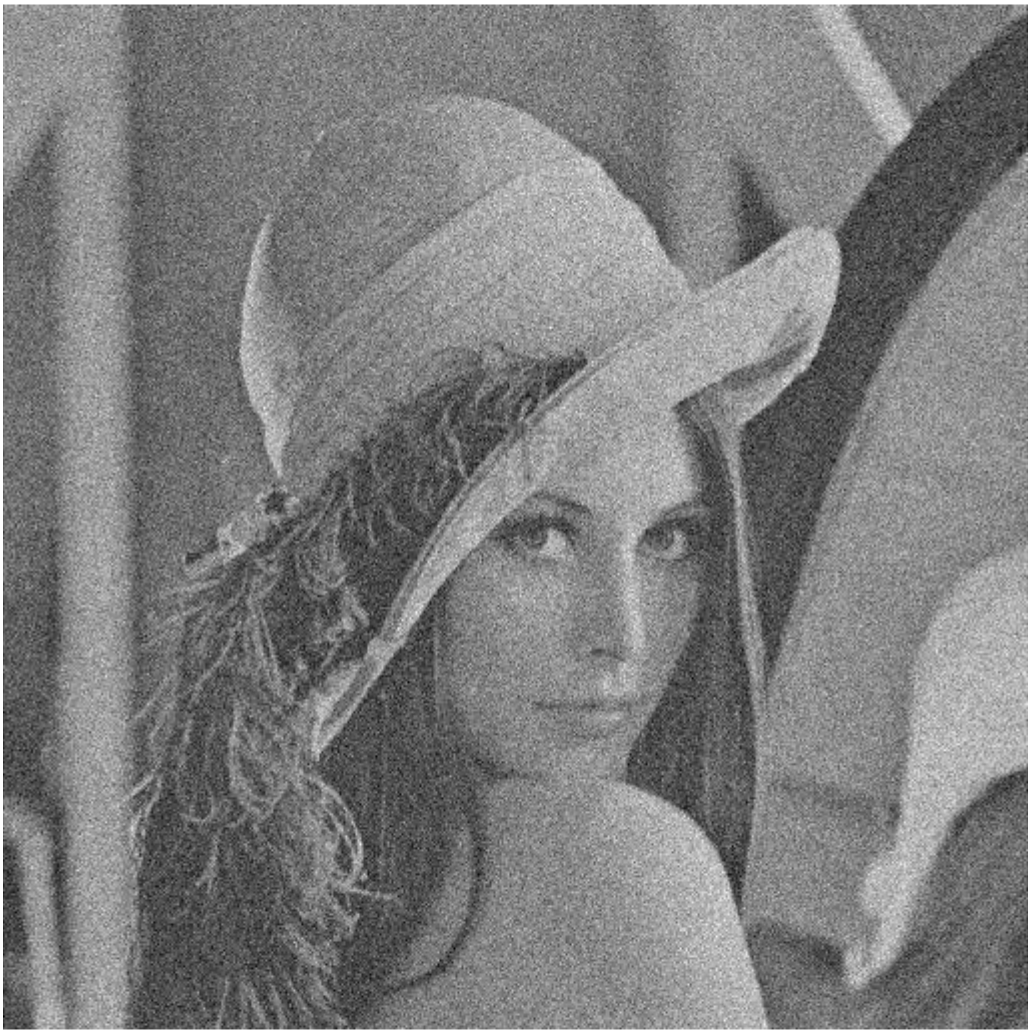
- **Resulting images**

1. Generate additive white Gaussian noise

(1) Amplitude=10

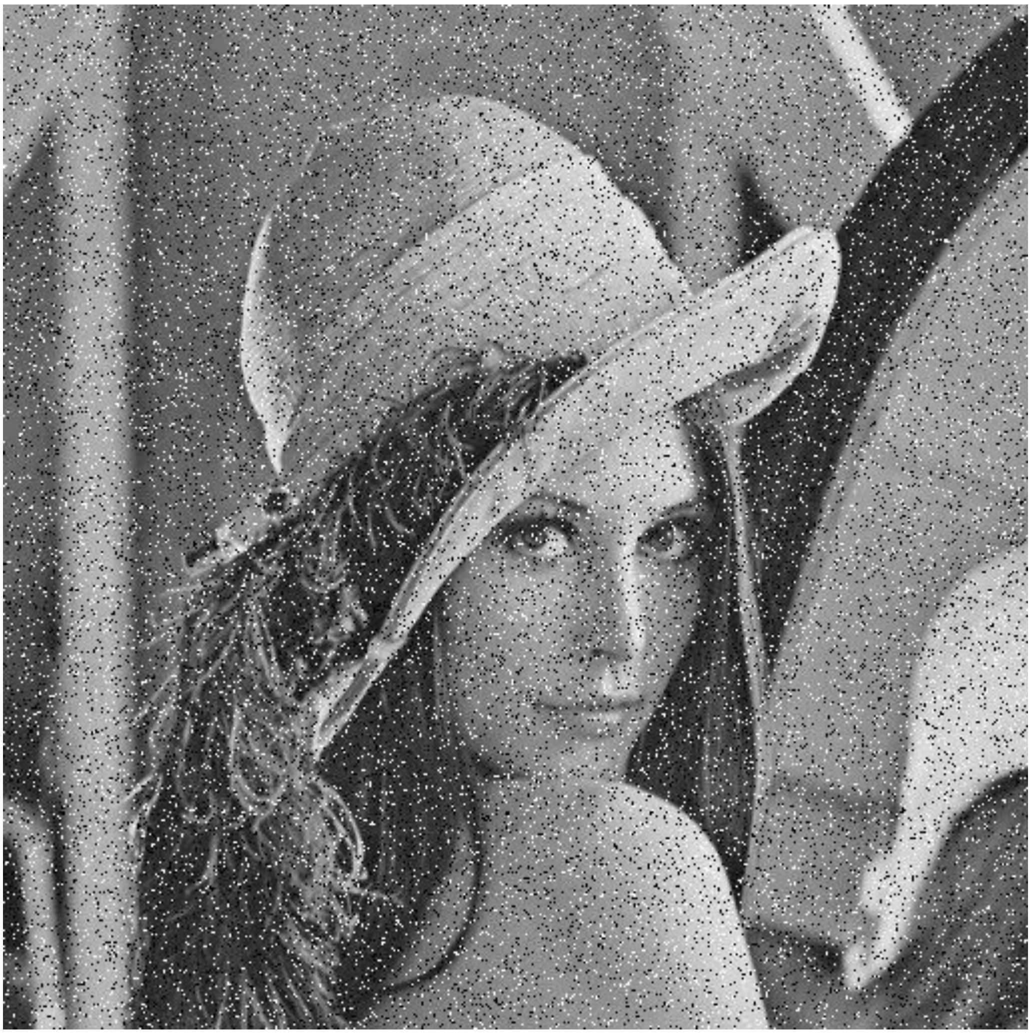


(2) Amplitude=30

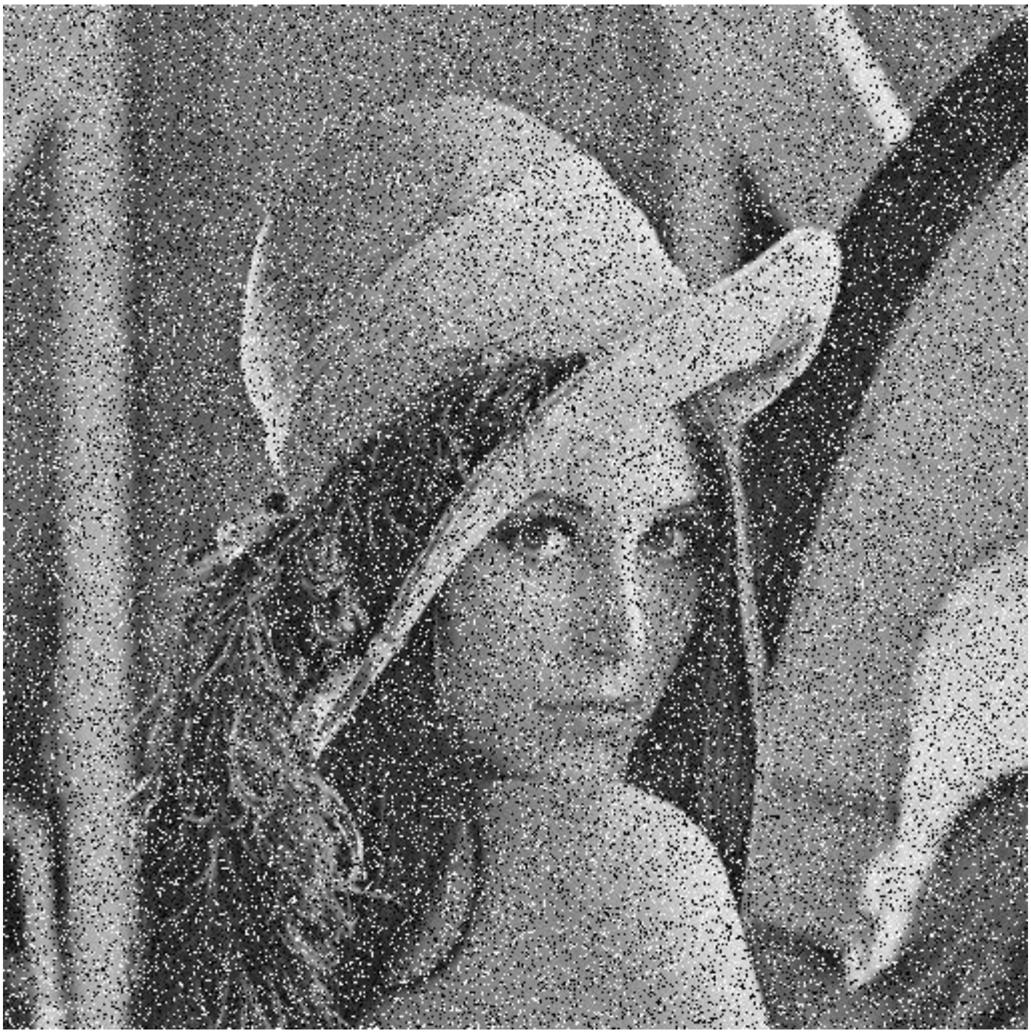


2. Generate salt-and-pepper noise

(1) Threshold=0.05



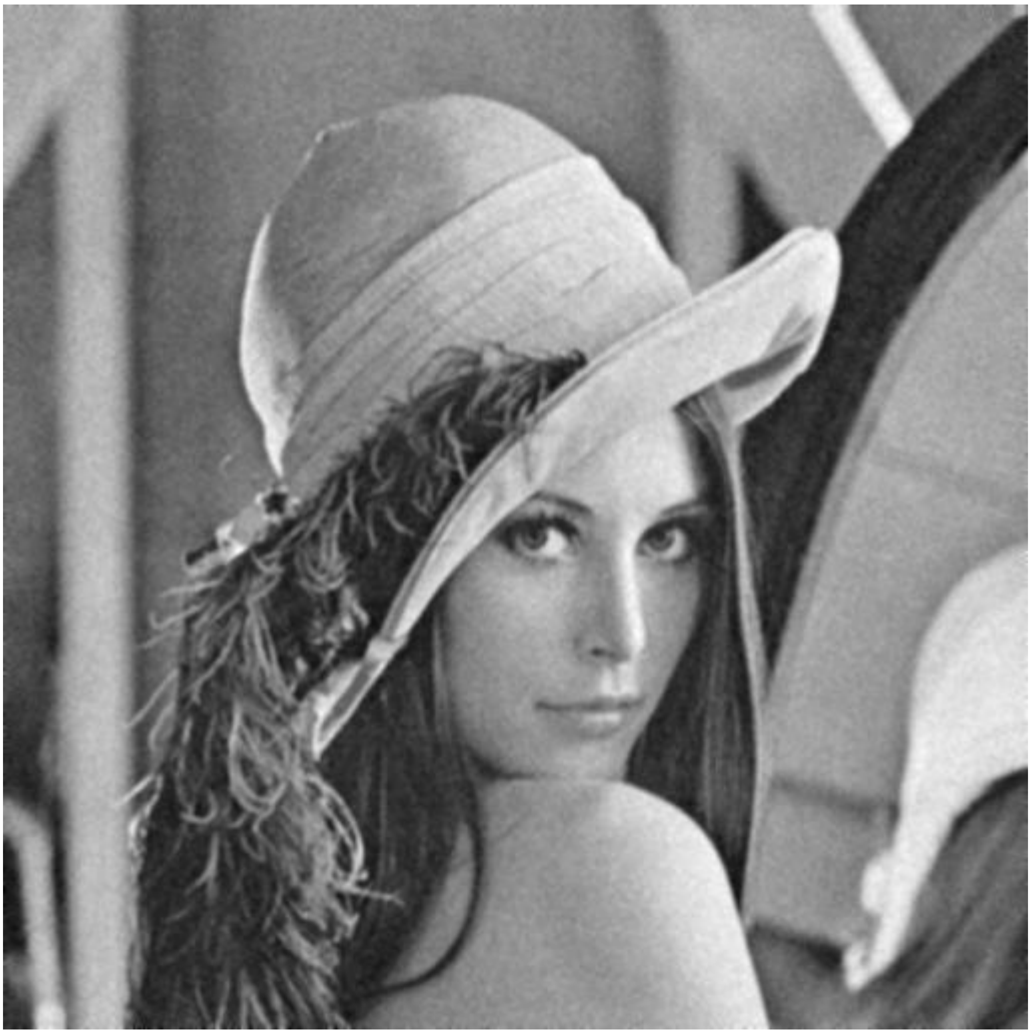
(2) Threshold=0.1



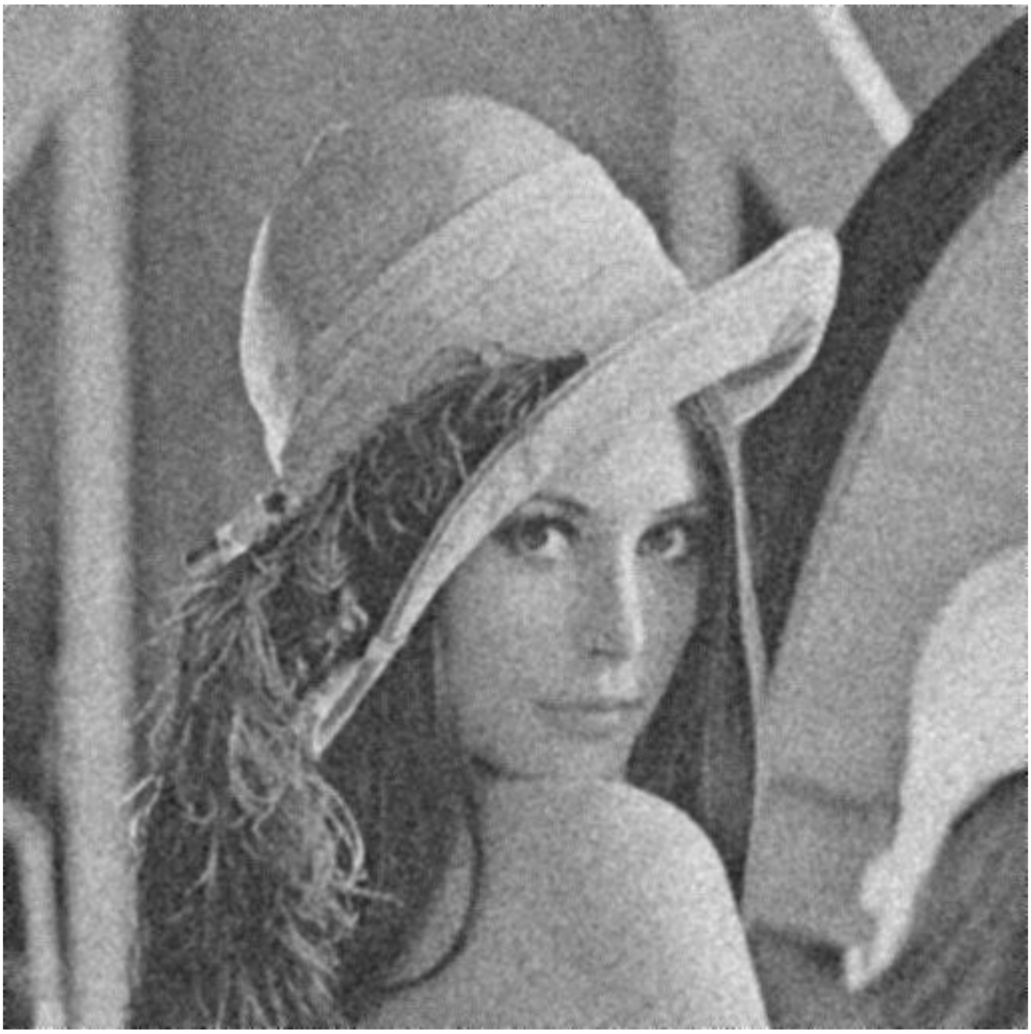
3. Run box filter (3X3, 5X5) on all noisy images

(1) 3x3

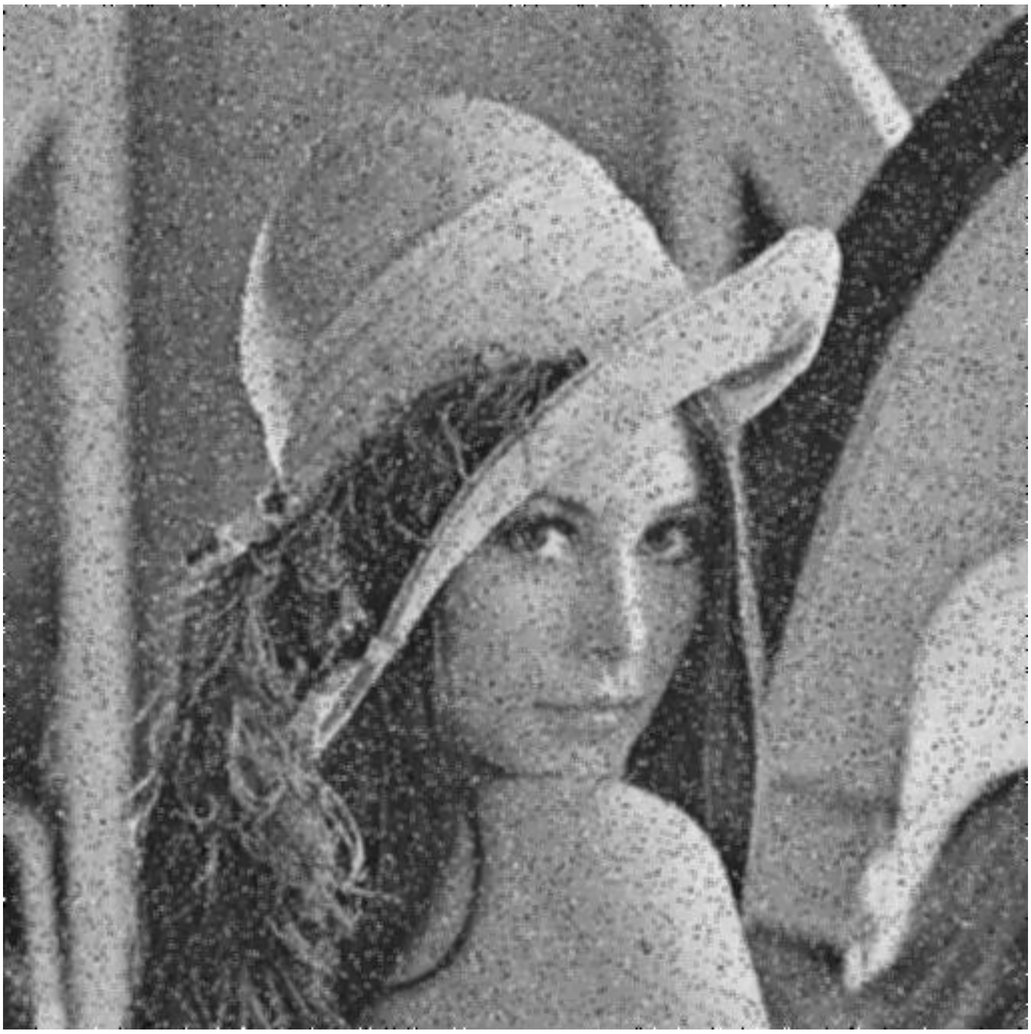
- Guassian(10)



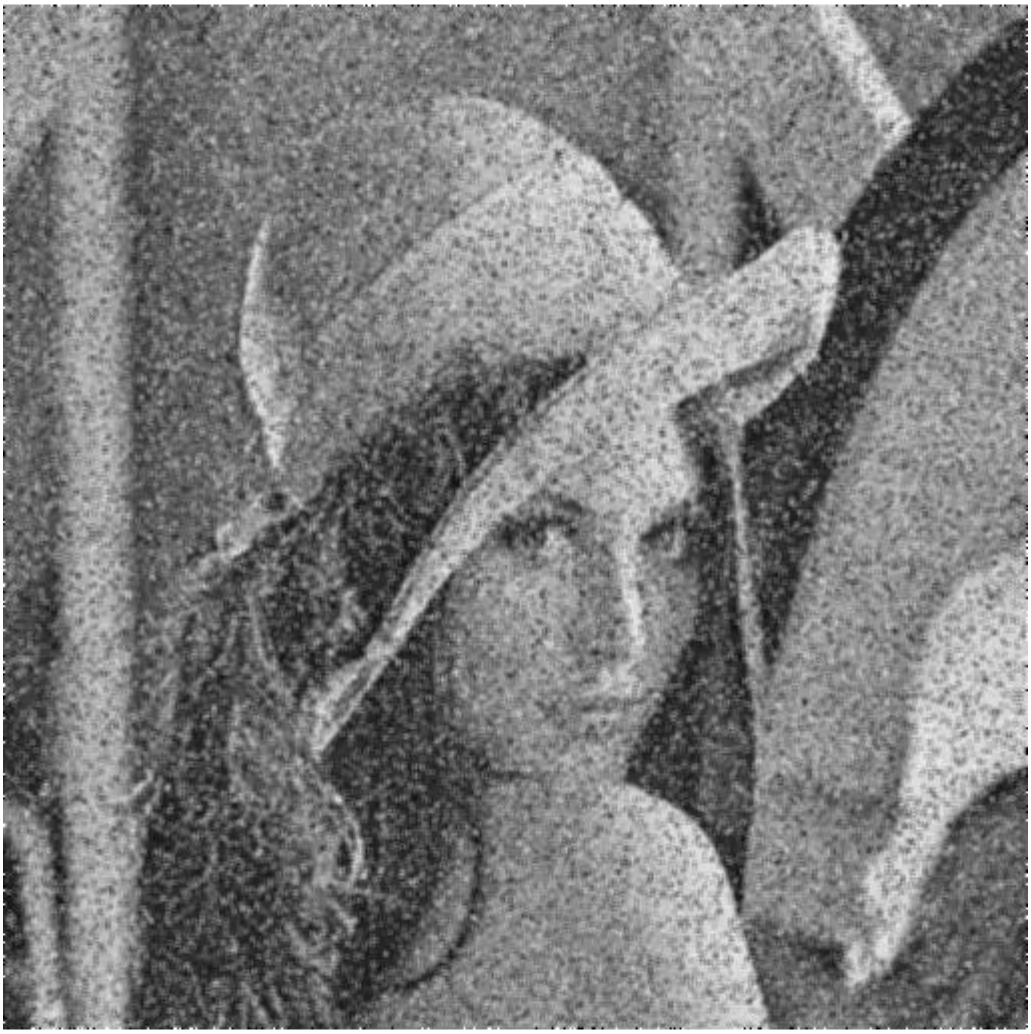
- Guassian(30)



- Salt-and-pepper(0.05)

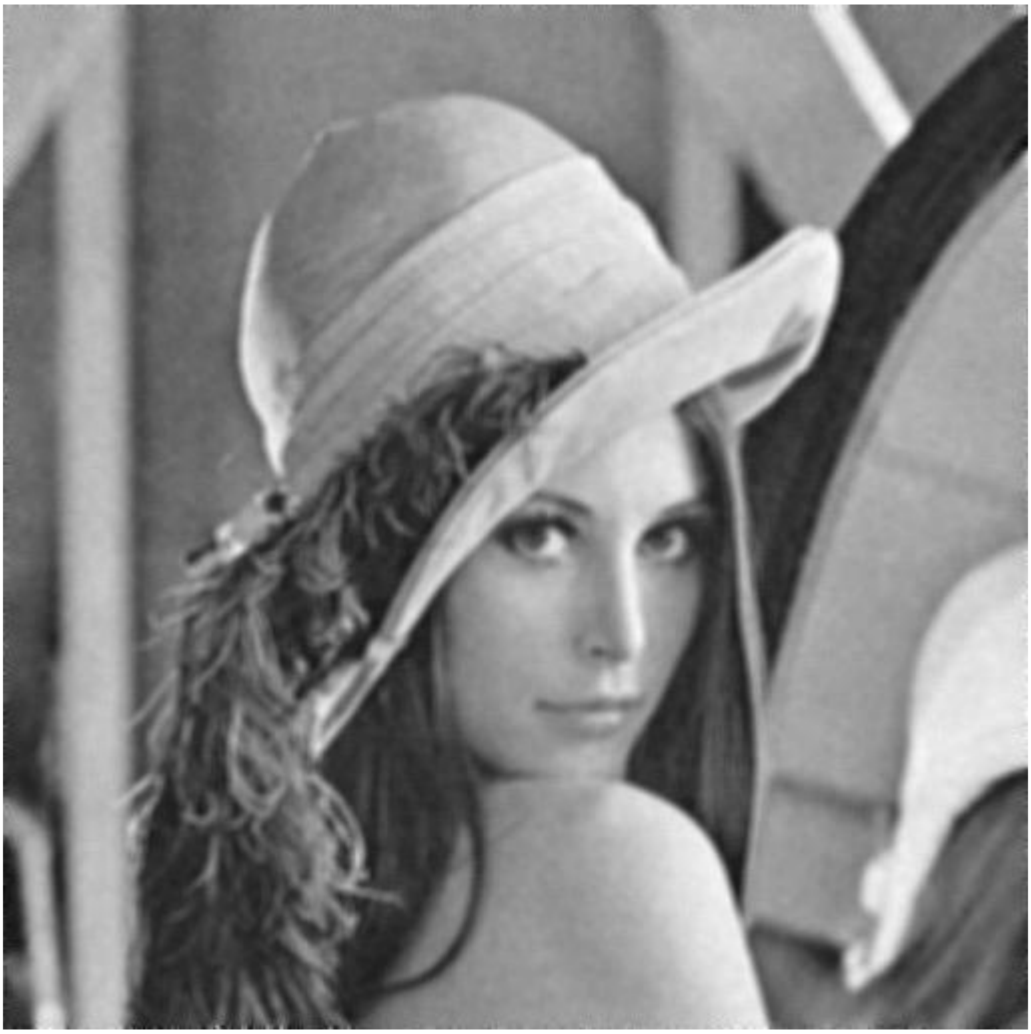


- Salt-and-pepper(0.1)

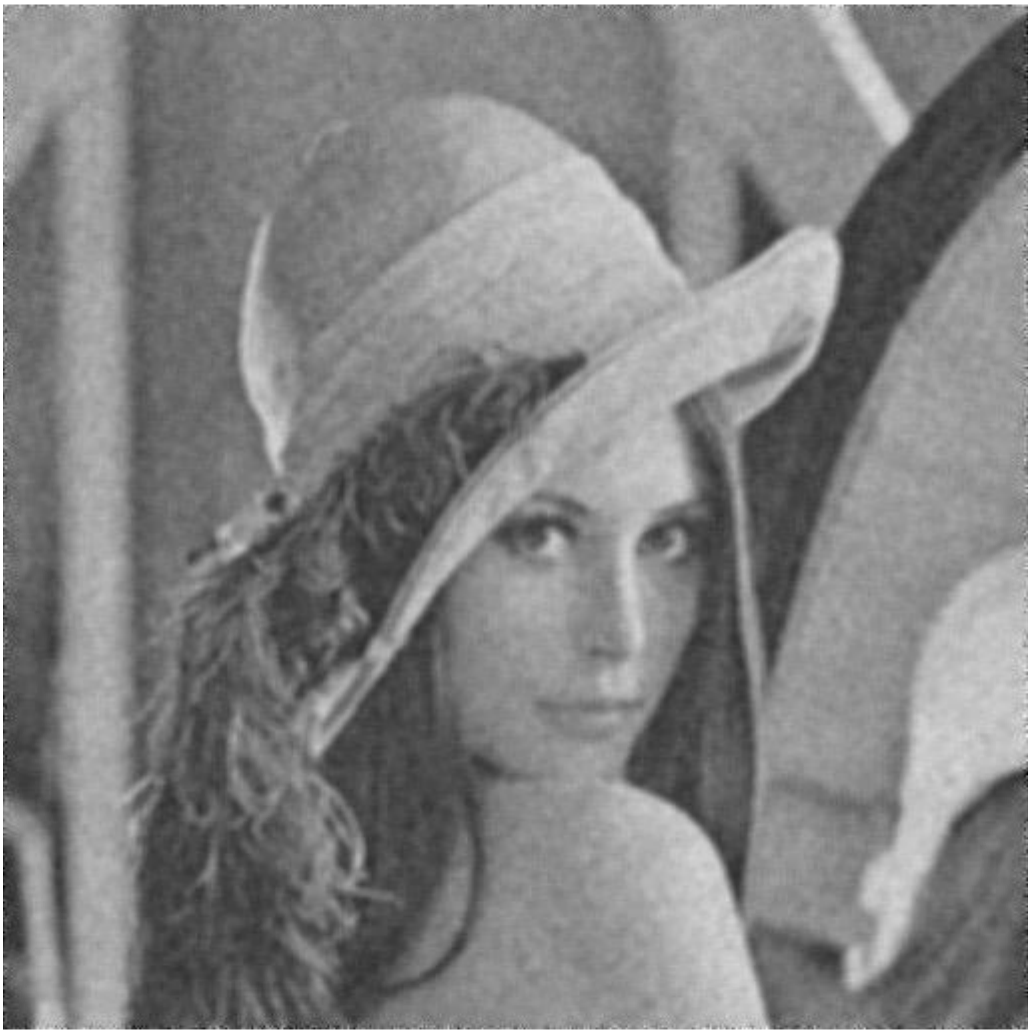


(2) 5x5

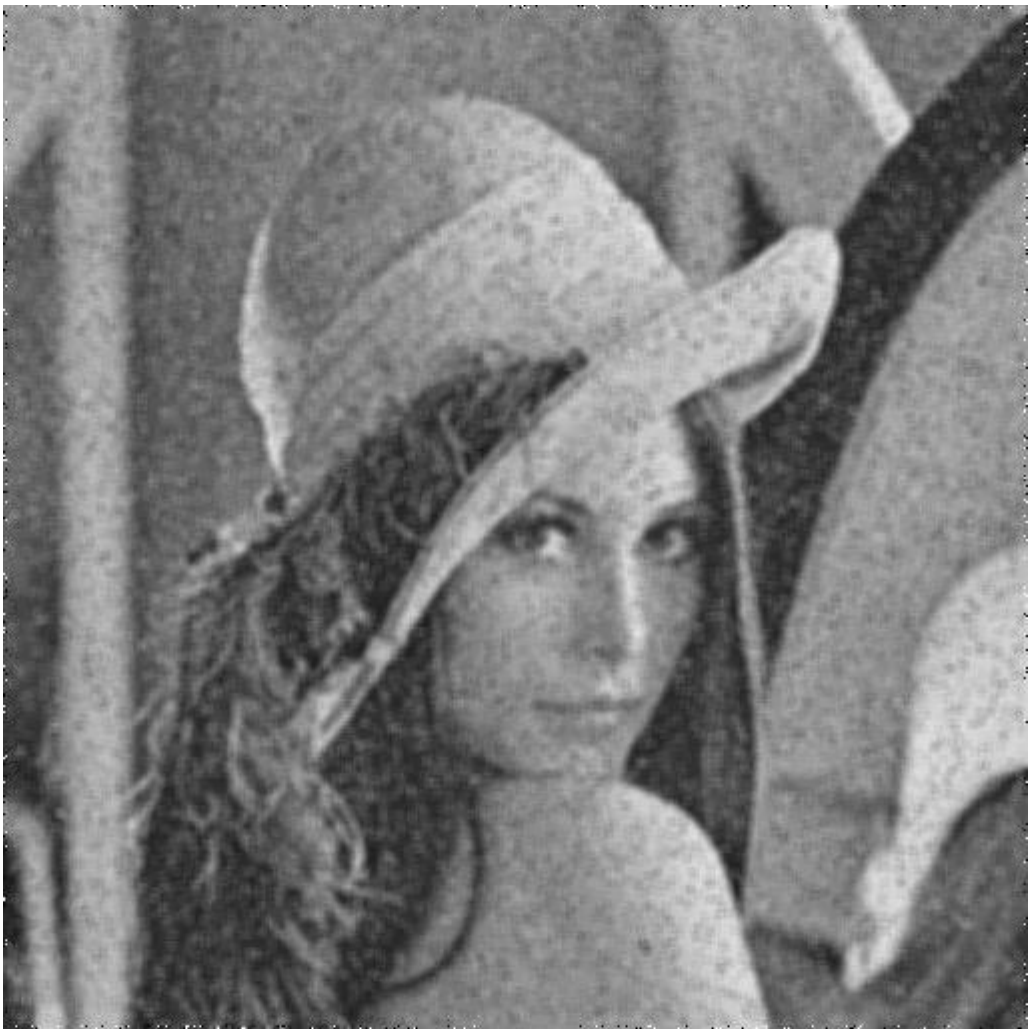
- Guassian(10)



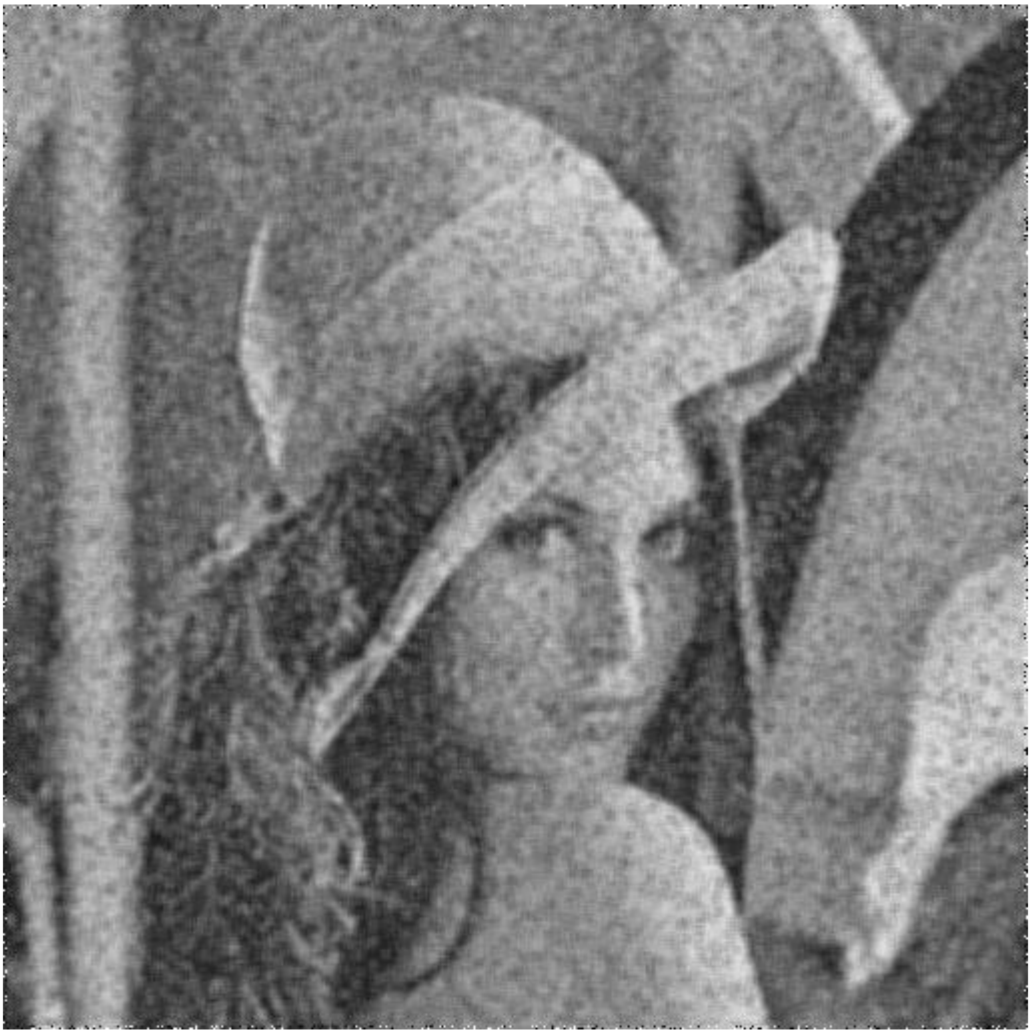
- Guassian(30)



- Salt-and-pepper(0.05)



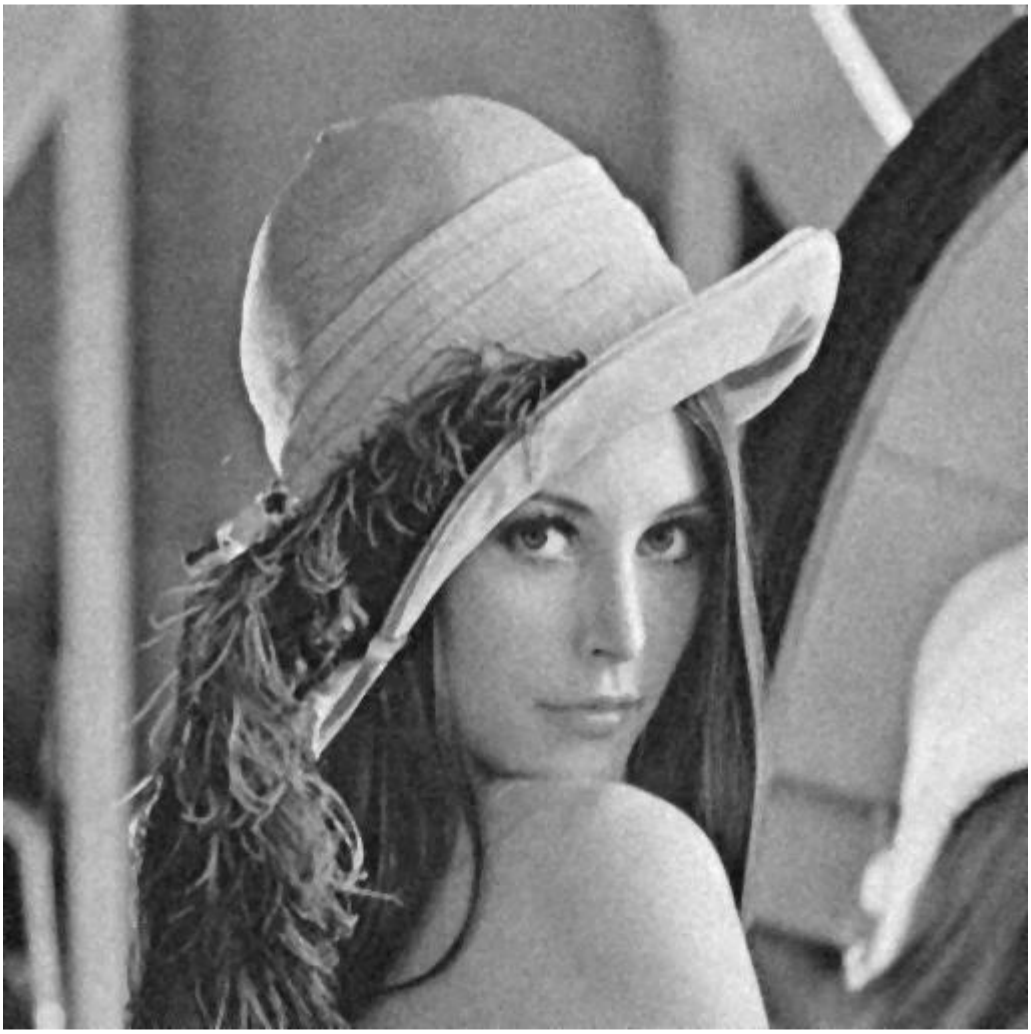
- Salt-and-pepper(0.1)



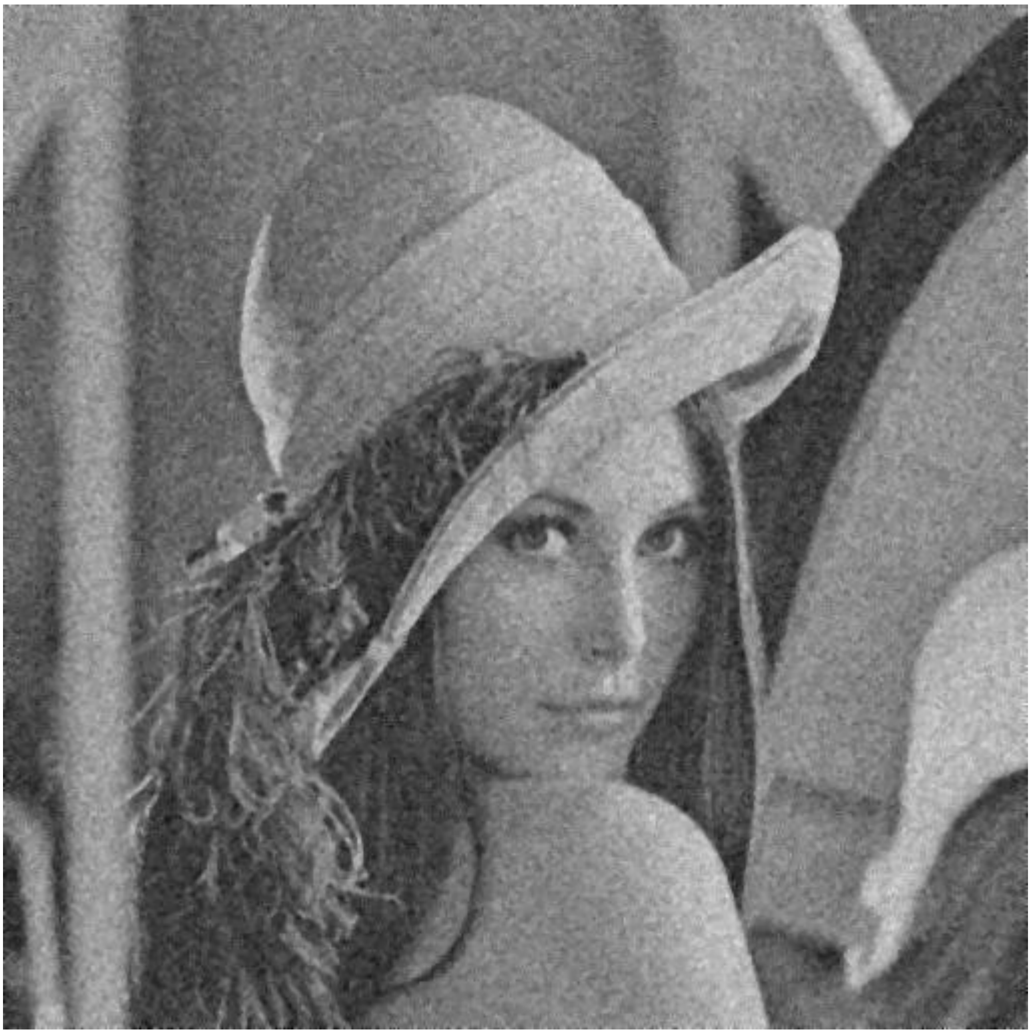
4. Run median filter (3X3, 5X5) on all noisy images

(1) 3x3

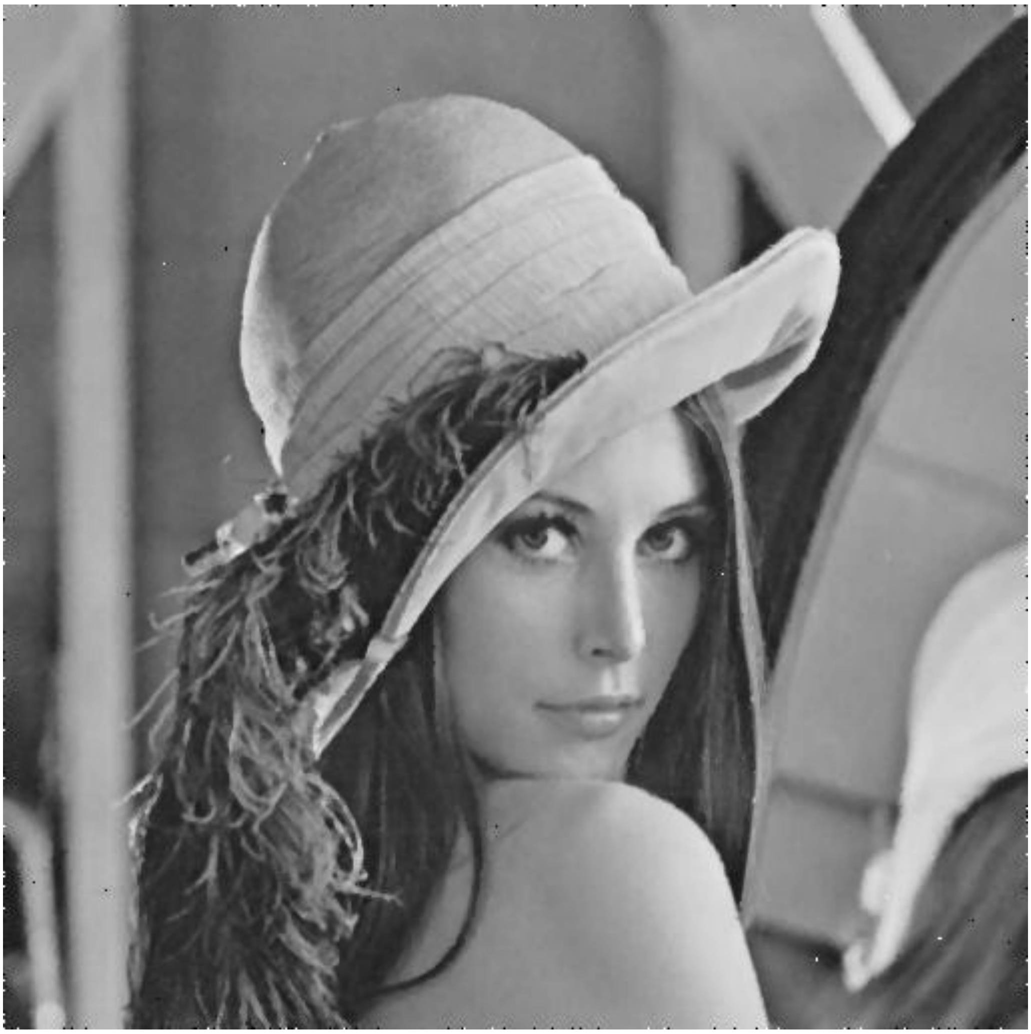
- Guassian(10)



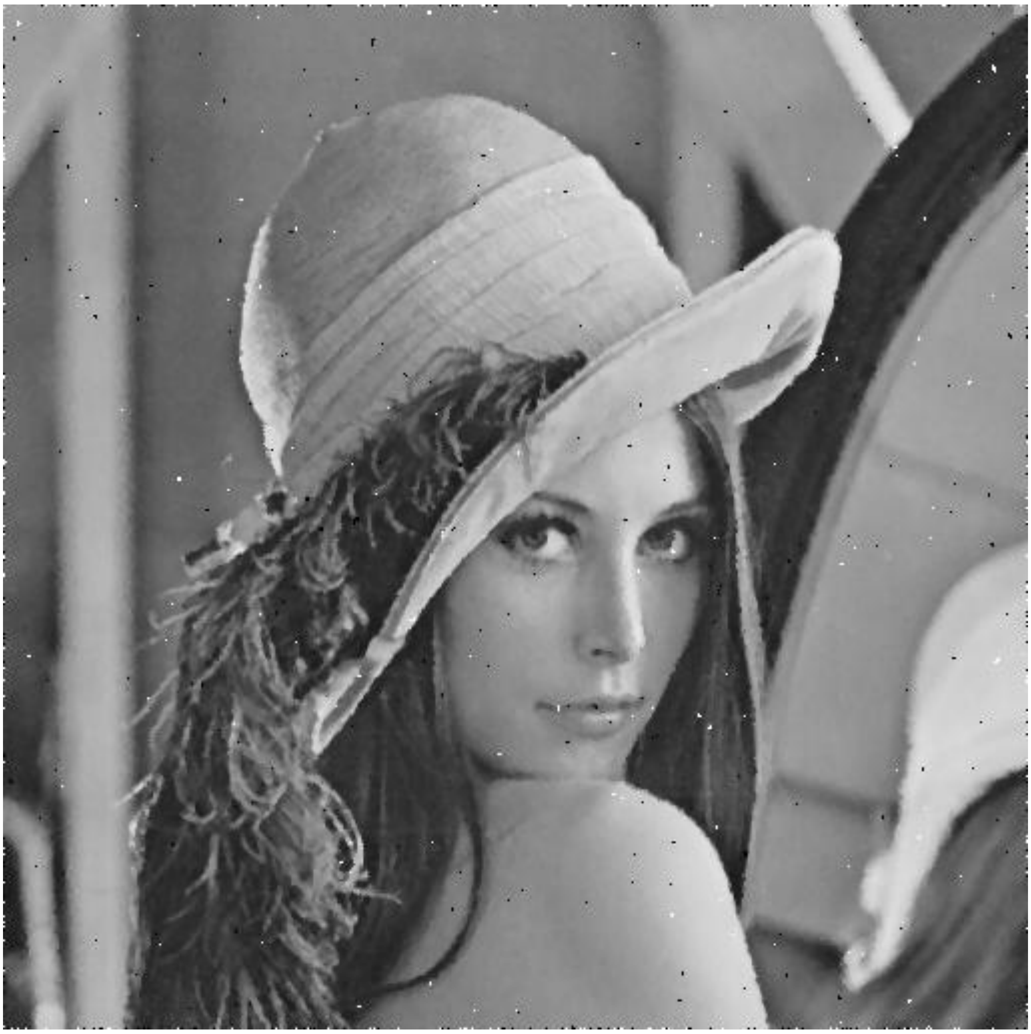
- Guassian(30)



- Salt-and-pepper(0.05)

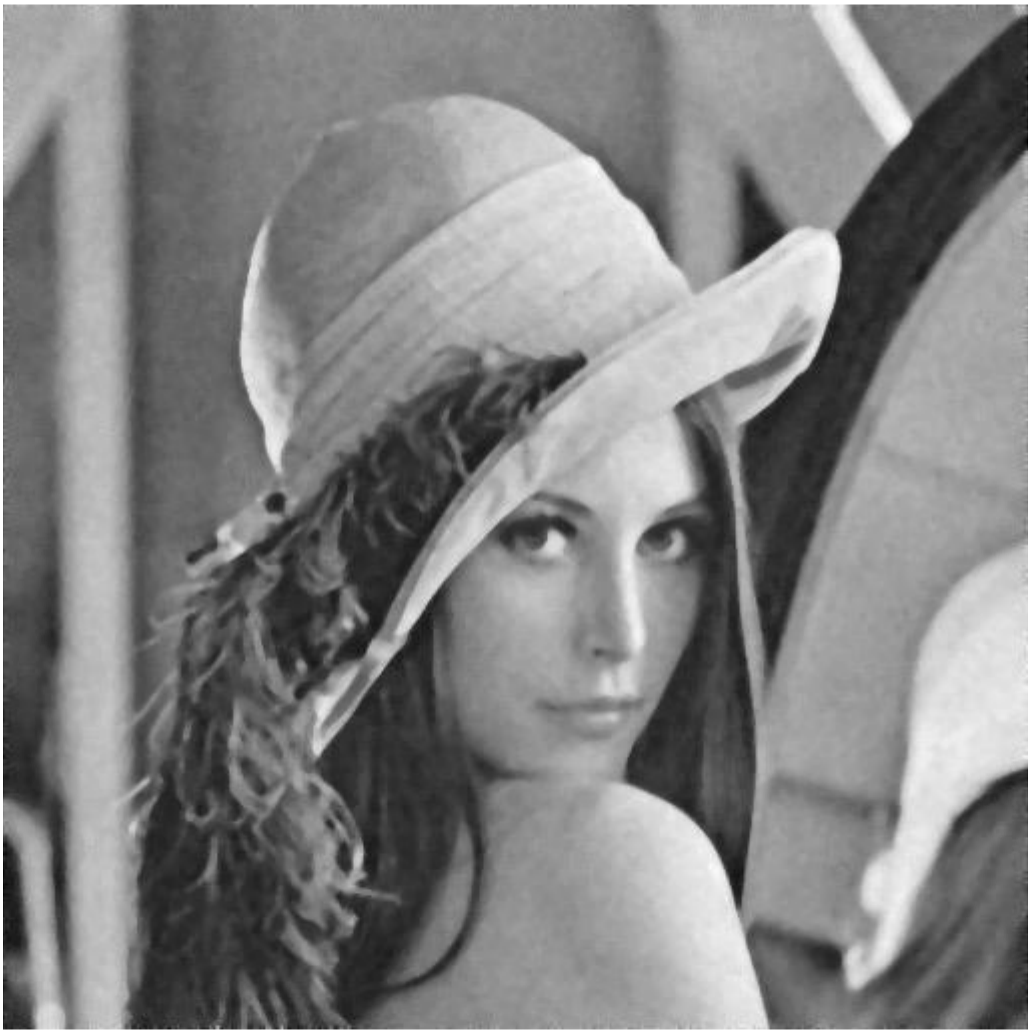


- Salt-and-pepper(0.1)

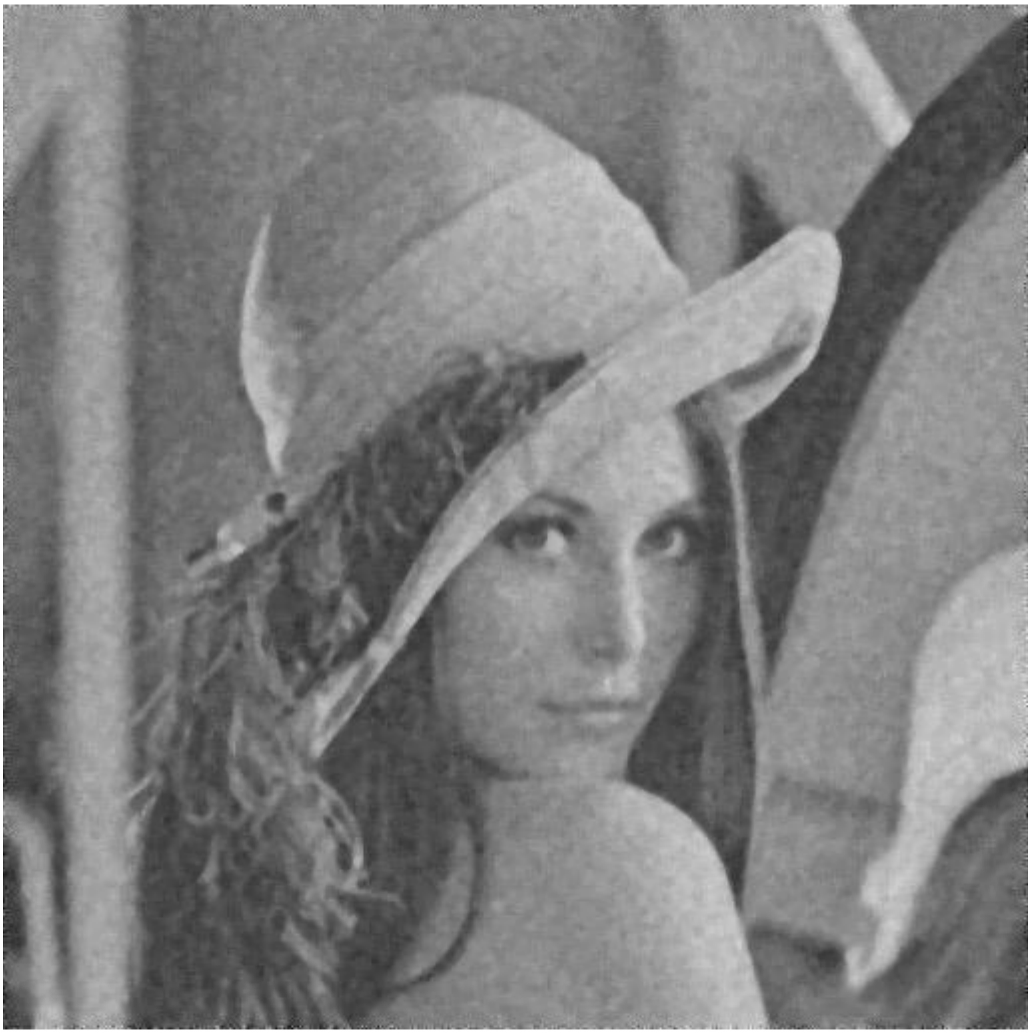


(2) 5x5

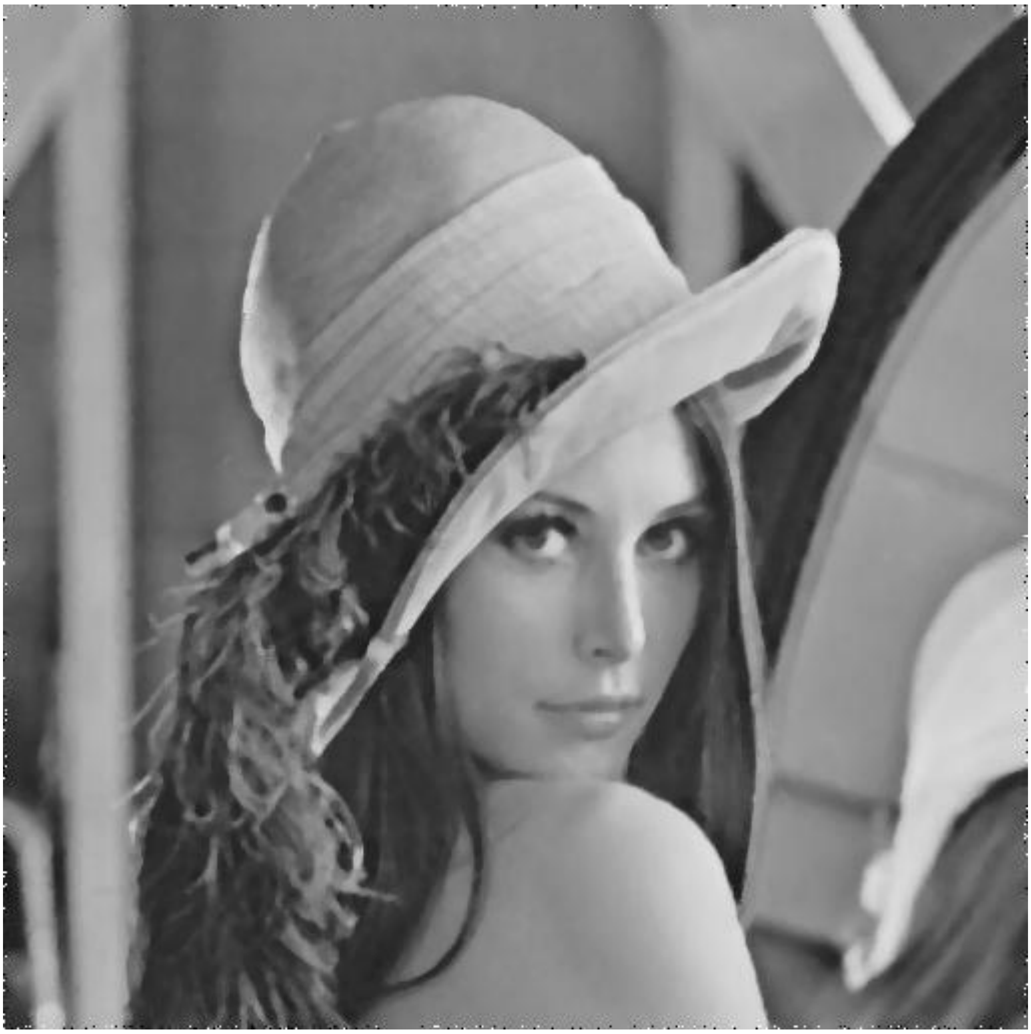
- Guassian(10)



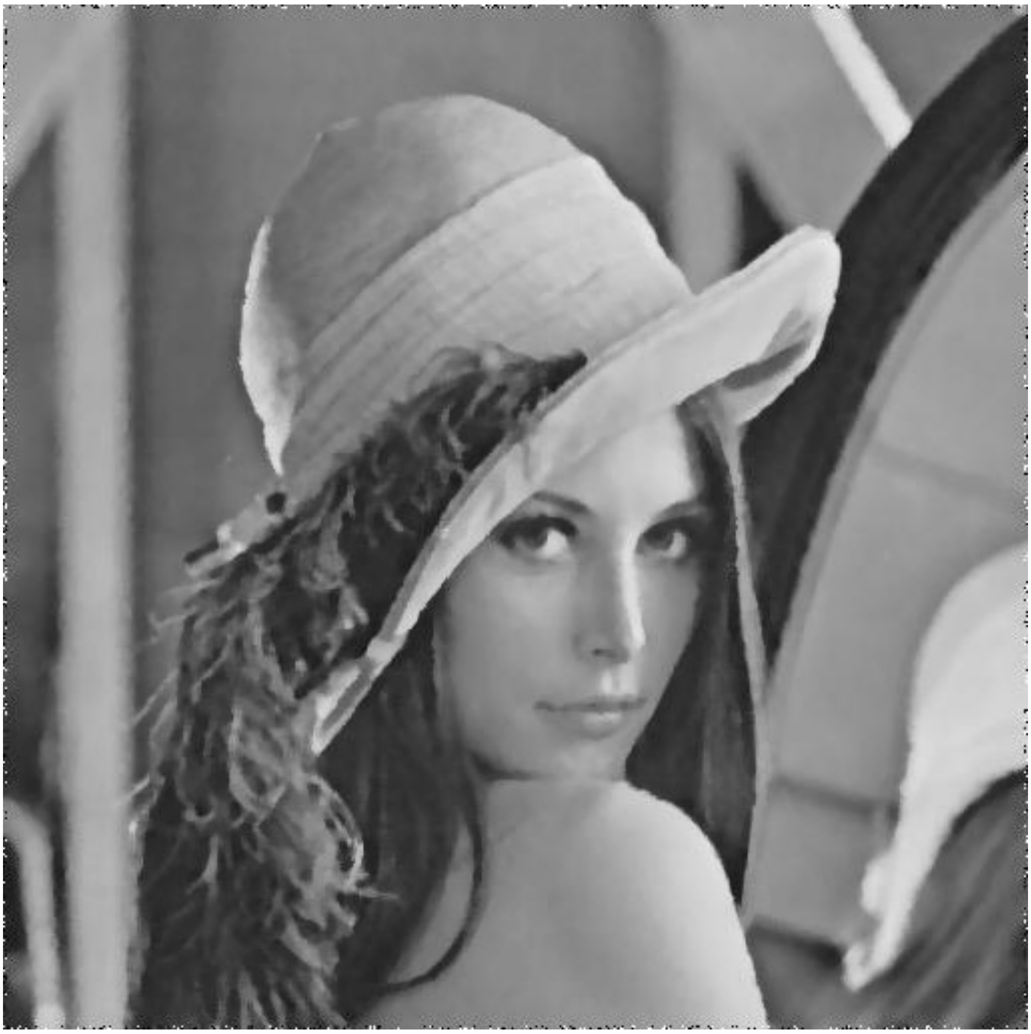
- Guassian(30)



- Salt-and-pepper(0.05)



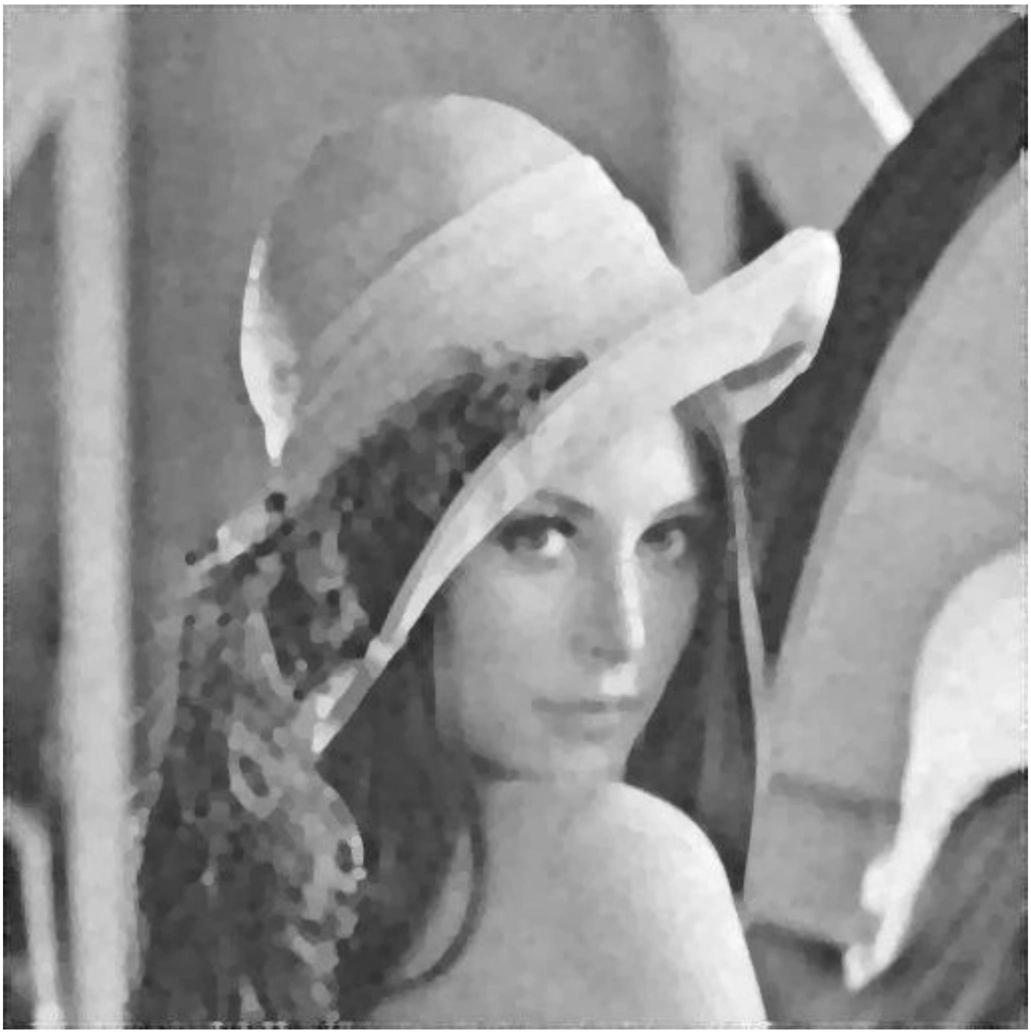
- Salt-and-pepper(0.1)



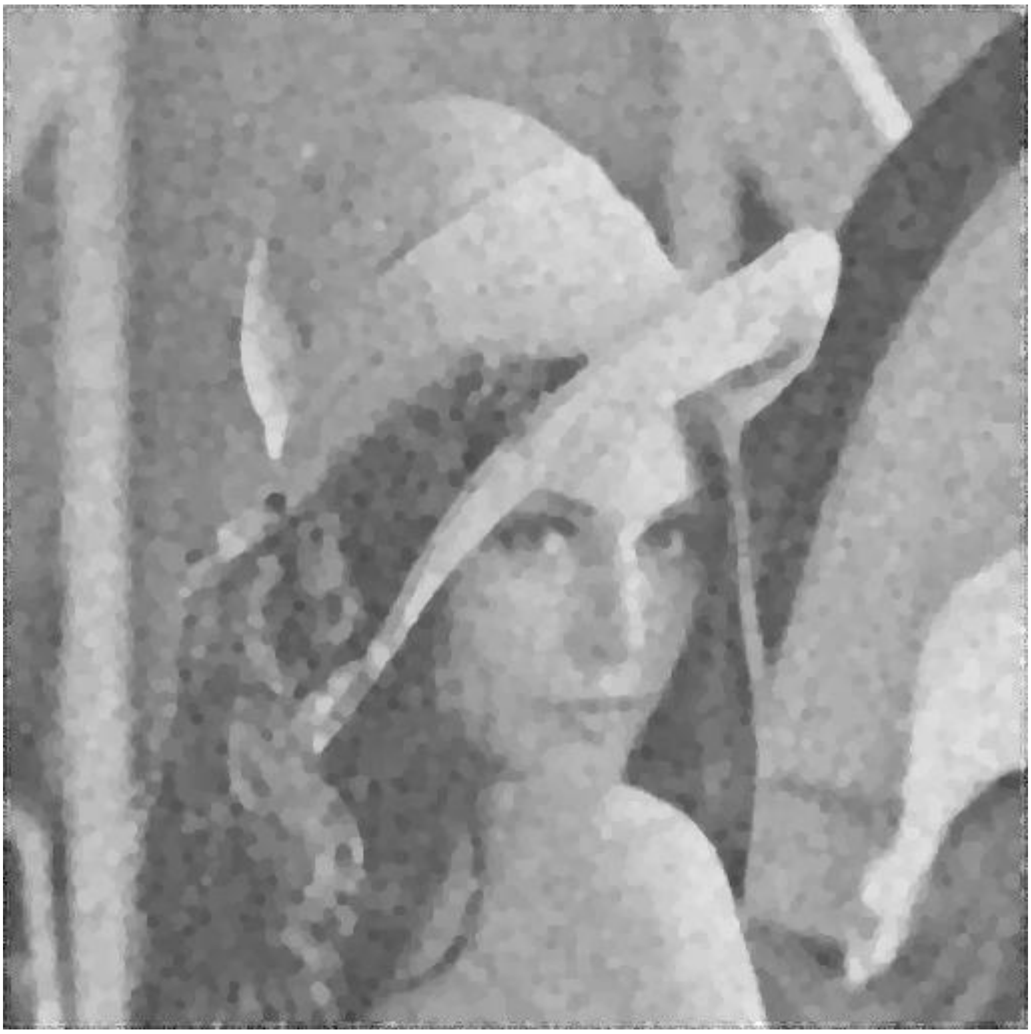
5. Run opening followed by closing or closing followed by opening

(1) closing followed by opening

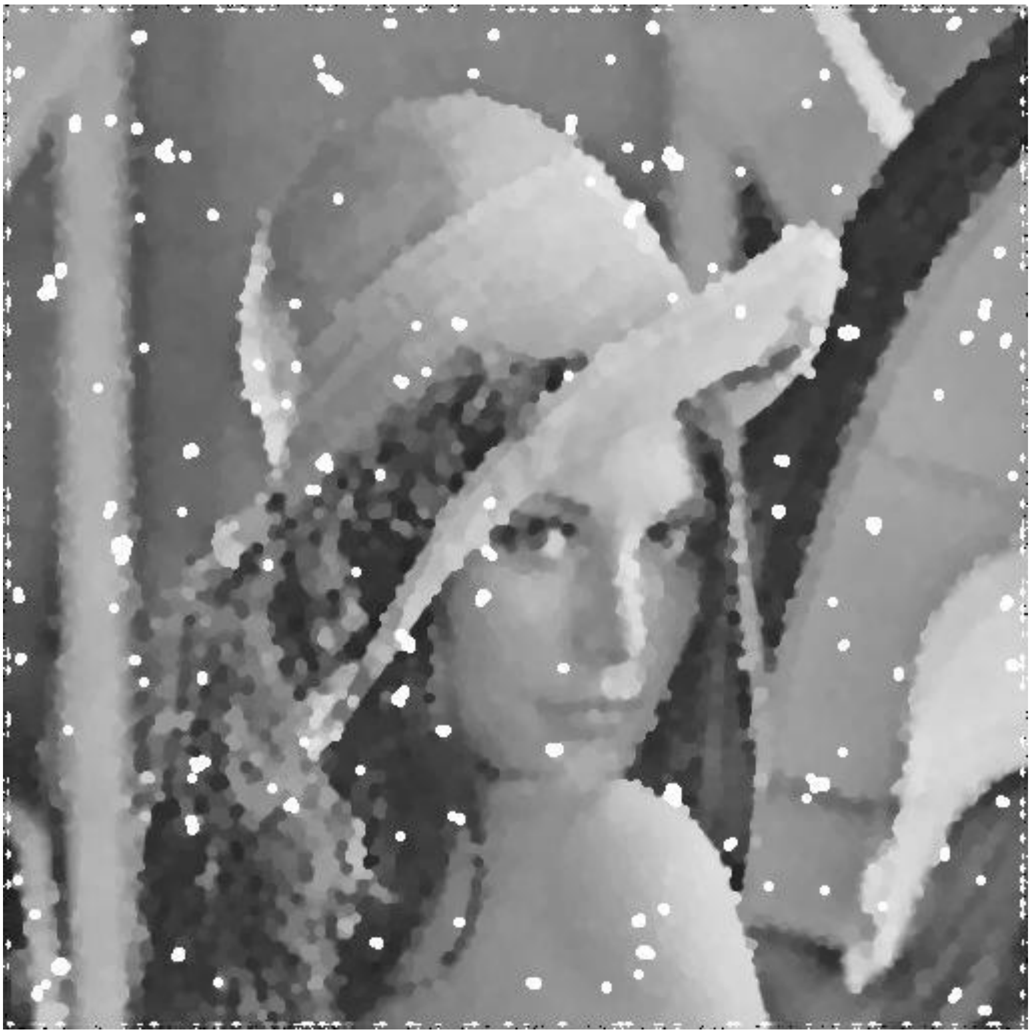
- Guassian(10)



- Guassian(30)



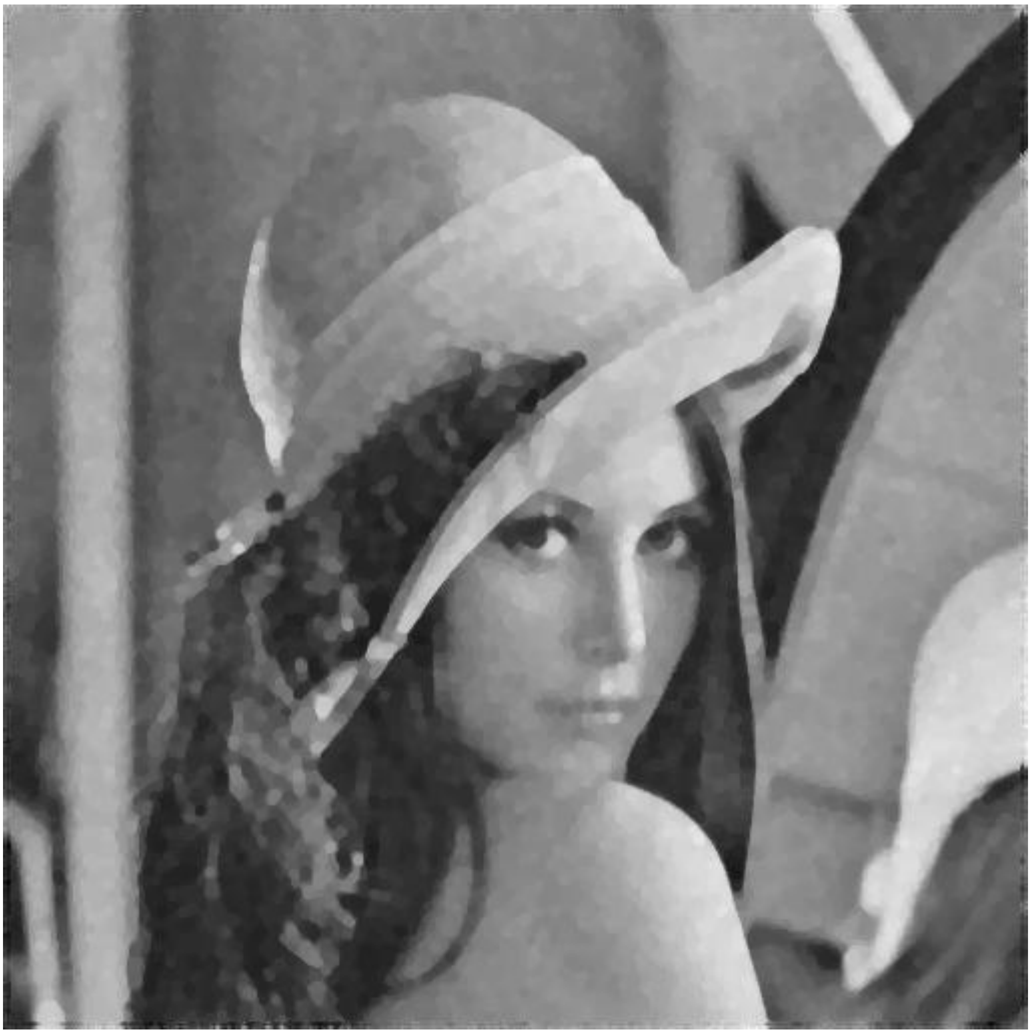
- Salt-and-pepper(0.05)



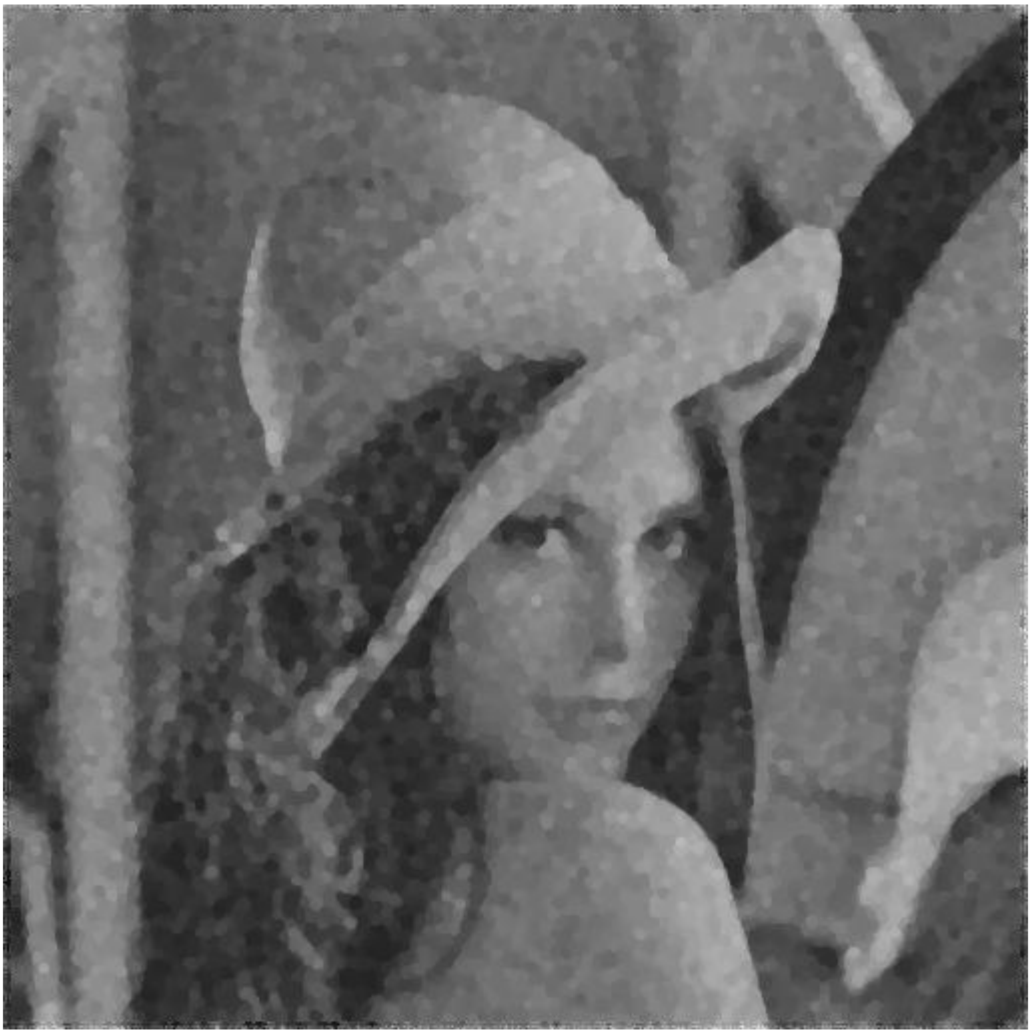
- Salt-and-pepper(0.1)

(2) opening followed by closing

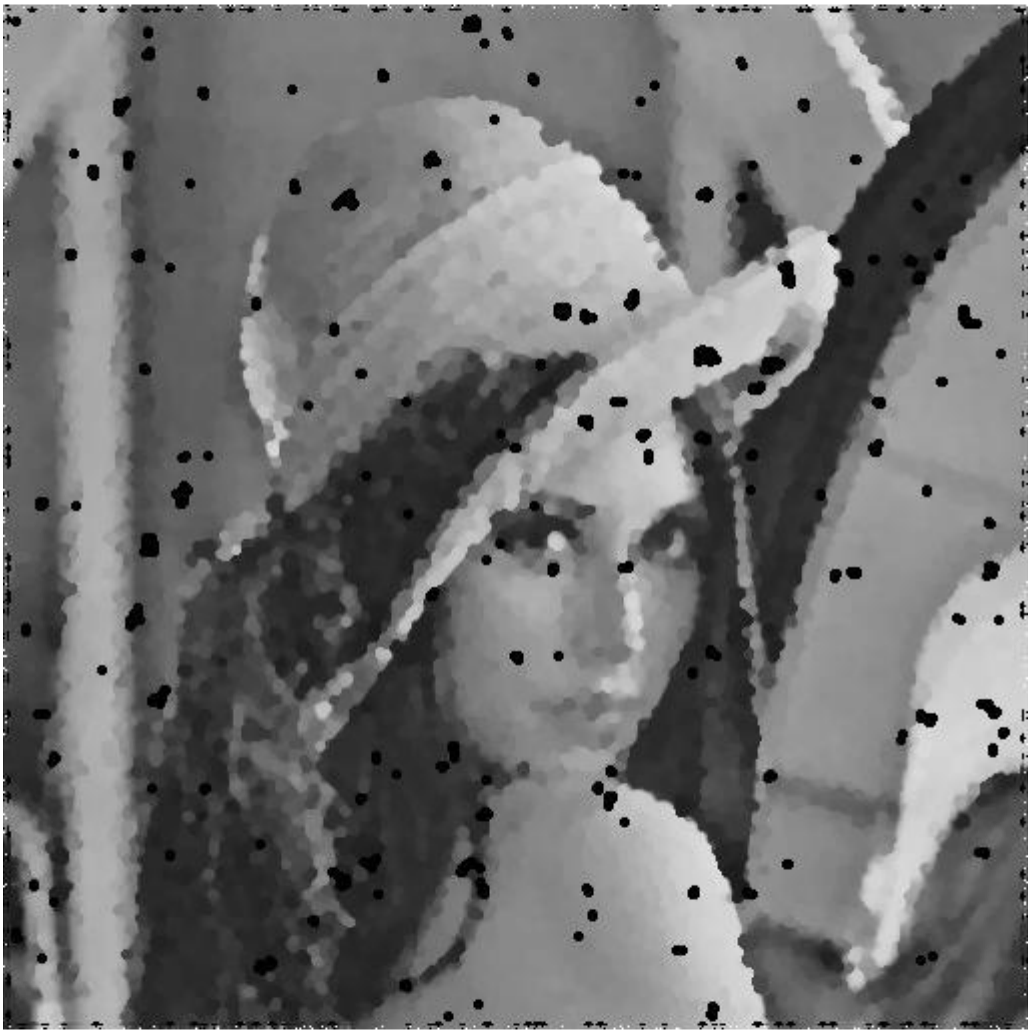
- Guassian(10)



- Guassian(30)



- Salt-and-pepper(0.05)



- Salt-and-pepper(0.1)