

## فاطمه باغخانی

اطلاعات گزارش	Filters چکیده
تاریخ: 99/1/30	فیلترهای مکانی یا <i>spatial filter</i> ، به گونه ای روی تصویر اعمال می شود که مقدار پیکسل با توجه به مقدار پیکسل های همسایگی آن تغییر می کند. مثلا می توان مقدار یک پیکسل را با اعمال یک فیلتر میانگین، برابر مقدار میانگین هشت پیکسل همسایگی آن قرار داد. فیلترها به دو نوع خطی و غیرخطی تقسیم می شوند، نوع خطی را کانولوشن مکانی هم می گویند.
<b>واژگان کلیدی:</b> Box Filter Salt and pepper noise gaussian filter Median Filter Sharping and Noise removal Edge Detection Unsharp Masking highboostfiltering	

### 1-مقدمه

فیلتر کردن یک تکنیک است که به وسیله ی آن می توان یک عکس را اصلاح کرد و یا کیفیت آن را افزایش داد. بعنوان مثال شما می توانید یک عکس را فیلتر کنید تا برخی از ویژگی هایش را مهمتر جلوه دهید و یا اینکه برخی از ویژگی های آن را حذف کنید.

فیلتر کردن، درواقع یک عملیات تقریبی و محدوده ای است به طوری که مقدار به دست آمده برای هر پیکسل از عکس خروجی به وسیله ی اعمال یک یا چند الگوریتم بر پیکسل هایی که در همسایگی پیکسل ورودی قرار دارند، به دست می آید.

محدوده یا همسایگی یک پیکسل، درواقع یک مجموعه از پیکسل ها است که موقعیت آنها نسبت به این پیکسل تعریف شده است

### 2-شرح تکنیکال

**BoxFilter** • هموار سازی یا مات کردن، یک عمل ساده پردازش تصویری است که بسیار مورد استفاده قرار می گیرد. دلایل زیادی برای هموار کردن یک تصویر وجود دارد. مانند کاهش نویز و جزئیات نامرتبطو از آثار بد آن میتوان به تارشدگی لبه ها اشاره کرد. در این بخش از هموار کردن برای کاهش نویز استفاده می کنیم (یا کاربردهای دیگر آن در بخش های دیگر آشنا خواهید شد).

به منظور هموار کردن تصویر باید از فیلترها استفاده کرد که عموماً از فیلترهای بودن و اینکه هرچه از مرکز دور شویم خطی استفاده می‌شود؛ در این فیلترها وزن کم شده و تاثیر کم تری در میانگین مقدار یک پیکسل خروجی (یعنی  $g(i,j)$ ) به دارد. جمع وزن دار پیکسل‌های ورودی بستگی دارد (یعنی  $f(i+k,j+l)$ ). نمودار کرنل گوسی یک بعدی به صورت زیر است:

$$g(i,j) = \sum_{k,l} f(i+k,j+l)h(k,l)$$

به  $h(k,l)$  کرنل می‌گویند. کرنل همان ضرایب فیلتر است.

معادله بالا به ما نشان می‌دهد که فیلتر، یک پنجره لغزنده از ضرایب است که روی تصویر حرکت می‌کند که **BoxFilter** نمونه ای از این فیلترهاست:

اگر فرض کنیم عکس یک بُعدی باشد، با توجه به شکل بالا پیکسلی که در مرکز قرار گرفته بزرگترین وزن را خواهد داشت. با افزایش فاصله از پیکسل مرکزی وزن‌ها نیز کمتر می‌شوند.

ساده‌ترین فیلتر موجود است که هر پیکسل خروجی، میانگین همسایه‌های آن است.

• کرنل این فیلتر به صورت زیر است:

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$$

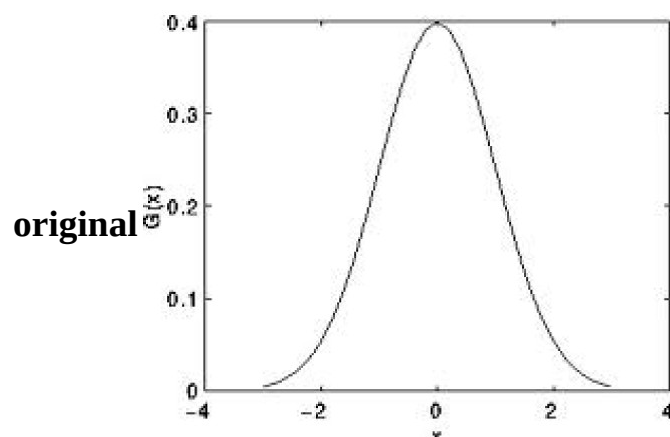
گوسی دو بعدی به شکل زیر است:

$$G_0(x, y) = A e^{-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2}}$$

که  $\mu$  نشان دهنده میانگین و  $\sigma$  نشان دهنده انحراف معیار (به ازای متغیرهای  $x$  و  $y$ ) است.

### GaussianFilter•

یکی دیگر از فیلترهای خطی و احتمالاً کاربردی‌ترین فیلتر (اما نه سریع‌ترین) است. فیلتر کردن به روش گوسی از کانوال هر نقطه موجود در آرایه ورودی با یک کرنل گوسی و سپس جمع همه آنها برای به دست آوردن خروجی، انجام می‌شود.



دارای چند مرحله است ابتدا عناصر داخل پنجره مرتب شده و سپس عنصر میانی به عنوان مقدار پیکسل در نظر گرفته میشود و از ویژگی های آن میتوان به حذف نویز بدون تارشدگی و مقاوم در برابر نویز فلفل نمک است

### salt and pepper noise

عمدتا در فرآیند انتقال اطلاعات رخ می دهند.

احتمال رخ دادن این نویز فقط در دو مقدار بوده، یا صفر یا ۲۵۵ (تصویر ۸ بیتی)، یا یک سیگنال را نابود می کند (صفر می کند)، یا یک سیگنال را کاملاً یک می کند و چیزی بین آن وجود ندارد. (مقدار نویز را با مقدار سیگنال جایگزین می کند) مانند شکل زیر :

### salt and pepper noise



### GaussianFilter•

$$\sigma=2.8$$

### Median Filter•

•از فیلترهای غیر خطی میتوان به min &max &median اشاره کرد

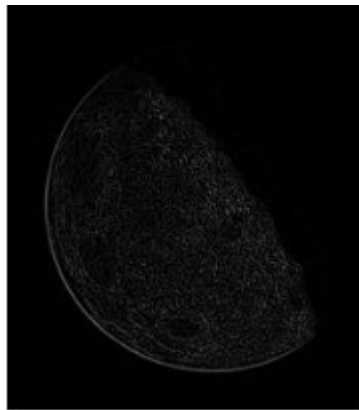
یکی از انواع فیلترهای غیرخطی میتوان به این فیلتر اشاره کرد که در این جا از محاسبات آماری استفاده میشود  
فیلتر میانه گیر روی پیکسل های عکس عبور می کند و هر پیکسل را با میانه پیکسل های همسایه اش جایگزین می کند.



removing by median filter

### Sharpening (فیلترهای شارپ کننده)

هدف عمده این فیلترها برجسته کردن جزئیات در یک تصویر یا بهبود جزئیاتی است که تار شده است. این فیلتر تغییر ناپذیر با چرخش است (isotropic)



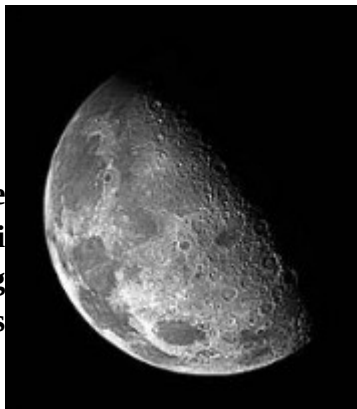
برای تیز کردن و تقویت کردن از مشتق گیری استفاده میشود.

مشتق اول و مشتق دوم در تابع گسسته:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

use  
sharpening  
filters



مشتق اول لبه های ضخیم تر به وجود می آورد اما مشتق دوم به جزئیات بهتر و بیشتر اشاره میکند.

یکی از این فیلترها فیلتر Laplacian است

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

و از انواع دیگر آن میتوان به Roberts و Sobel نیز اشاره کرد.

## highboostfiltering

### &Unsharp Masking

انواع لبه:

highboostfiltering نوعی روش تیز

کردن که تا مدت‌ها در چاپگرها مورد استفاده بوده است. و مراحل آن به شکل

زیر است:

#### ۱- الگوریتم soble

1- ایجاد تصویر تار از تصویر اصلی

2- تفریق و کم کردن تصویر تار از تصویر این متد لبه‌ها را با استفاده از تخمین زدن مشتق پیدا می‌کند، که لبه‌ها را در آن

نقاطی بر می‌گرداند که گرادیان تصویر،  $\max$  است. در فیلتر سوبل دو ماسک به

3- اضافه کردن ماسک به تصویر اصلی

صورت زیر وجود دارد:

$$g_{\text{mask}}(x, y) = f(x, y) - \bar{f}(x, y)$$

$$g(x, y) = f(x, y) + k * g_{\text{mask}}(x, y)$$

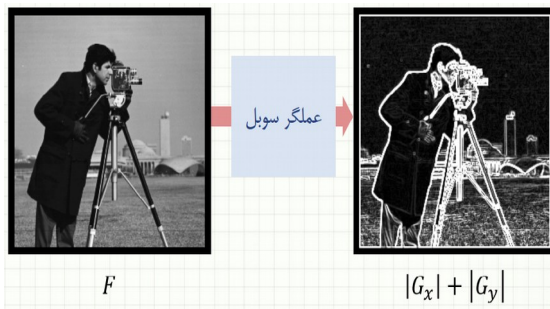
$$G_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

ماسک سوبل افقی

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

ماسک سوبل عمودی

و در این تصویر از تأثیرات آن را مشاهده میکنید

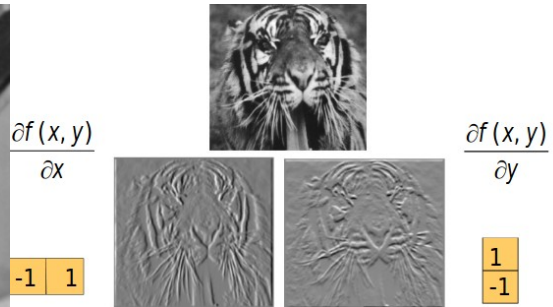


علاوه بر این فیلتر فیلترهای دیگه مانند Robert و.. ماسک‌های برای انواع لبه یابی به صورت افقی یا عمودی وجود دارد-

### فیلترهای لبه یاب:

لبه یابی در واقع مجموعه عملیات ریاضی می باشد که به کمک آنها می توان نقاطی از تصویر که در آنها روشنایی بطور شدید تغییر می کند را شناسایی کرد. لبه ها معمولا بصورت خطوطی که دارای انحنای هستند مشخص می شوند.

از لبه یابی می توان برای تشخیص تغییرات شدید در روشنایی که معمولا نشانه رویدادی مهم یا تغییر در محیط است، استفاده کرد. همچنین می توان از لبه یابی در object recognition (تشخیص اشیا) و segmentation (جدا سازی عکس و تبدیل آن به چند عکس) و بینایی ماشین استفاده کرد.



سمت راست در جهت  $y$  سمت چپ در جهت  $x$  خواهد بود

### 3-شرح نتایج

در سوال اول: این فیلترها در جهت کاهش نویز و هموار کردن  $\text{false contour}$  ها استفاده می شود اما یکی از بدی های آن باعث تار شدگی لبه ها و همین طور گاهی بعد از دو مرحله



از دست دادن بعضی جزئیات که این بسته به کاربرد آن میتواند مفید و یا مضر باشد هنگامی هدف به دست آوردن کلیتی از تصویر است مناسب اما زمانی که جزئیات میخواهیم نامناسب میباشد.

3.1.2: اگر بارها این فیلتر را روی تصویر تأثیر دهیم برای کاهش نویز اما کم کم تصویر تار شده تمام اطلاعات تصویر را از دست داده و زیاد از حد آن سبب تار شدن تصویر می شود و مات شدن آن میشود.

3.1.3: اجرای  $\text{boxfilter}$  به تعداد زیاد:

بعد از یک مرحله اجرا

بعد از ۴ مرحله



300 بار



بعد از 10 مرحله

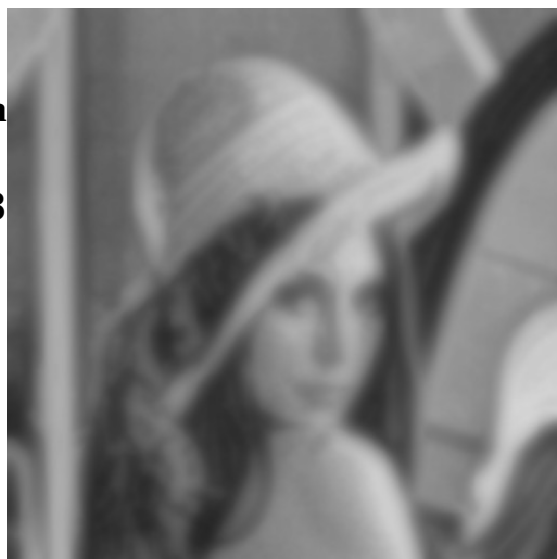


500 بار





media  
m  
3\*3



و وقتی به سمت بی نهایت رود کم  
مات میشود

:3.2.1

0.1 نویز نمک فلفل

median5\*5 روی تصویر  
0.1







**median 7\*7**



**salt pepper nois 0.2**



**median 9\*9**

**median 3\*3**

**9\*9**



**5\*5**



**salt and pepper 0.05**

**7\*7**





**median 7\*7**

**\*3  
3**



**median 3\*3**

**median 5\*5**



median 9\*9



حال به جدول زیر توجه کنید:mse

mse	3*3	5*5	7*7	9*9
0.05	18.40	25.25	29.63	33.02
0.1	21.29	26.71	30.53	33.85
0.2	29.96	30.48	33.21	35.82

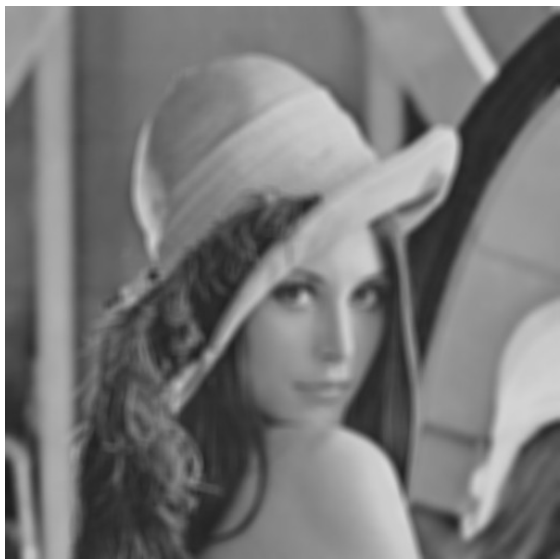
و با توجه به جدول همان طور که میبینید هرچه اندازه پنجره بزرگ تر  $5 \times 5$  boxfilter تصویر از تصویر اصلی دورتر می شود و تفاوت ها فاحش تر همان طور که در MSE ها دیده میشود.

### 3.2.2:

ابتدا گوسین را روی عکس ها زده و سپس از فیلترهای median و boxfilter استفاده میکنیم.

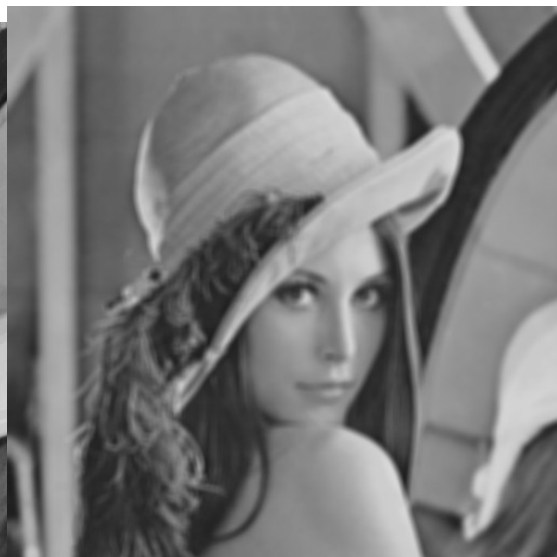
0.01 گوسین

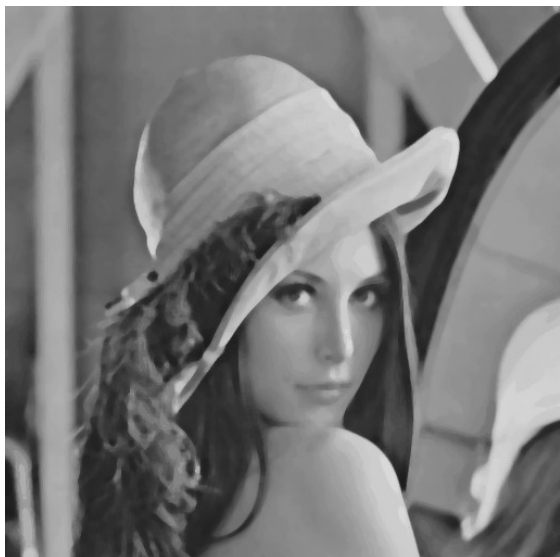
**boxfilter9\*9**



**7box\*7**

**median3\*3**



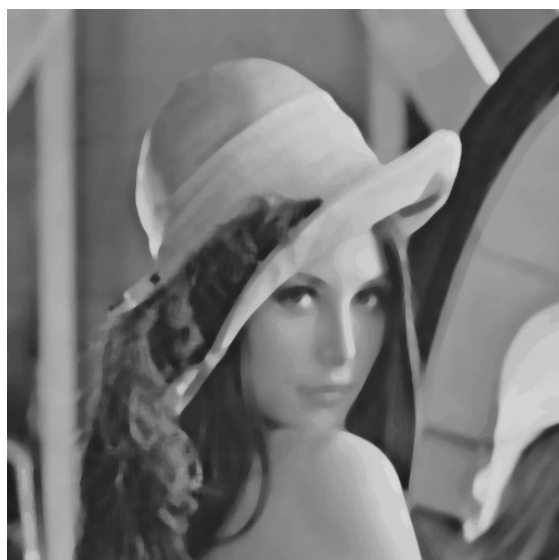


**median5\*5**

**median9\*9**



**median7\*7**



**0.0  
5**

**gussian**



**box5\*5**



**boxfilter3\*3**

**7\*7  
boxfilter**





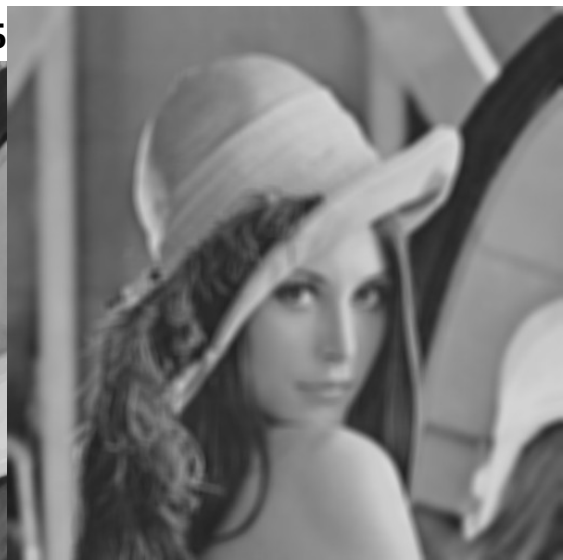


**median3\*3**

**box9\*9**

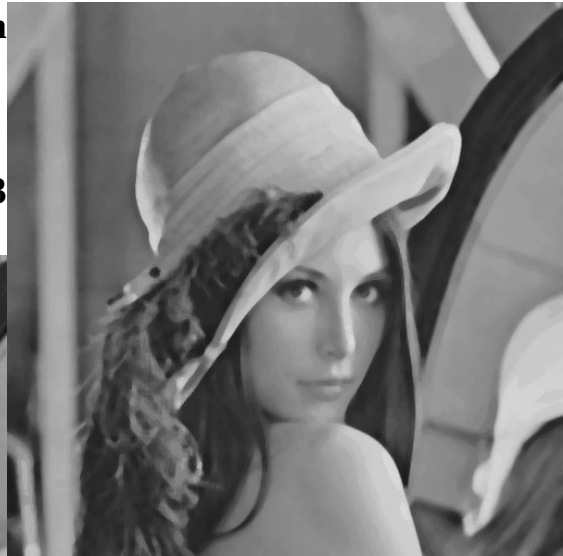


**median5\*5**



**gussian**

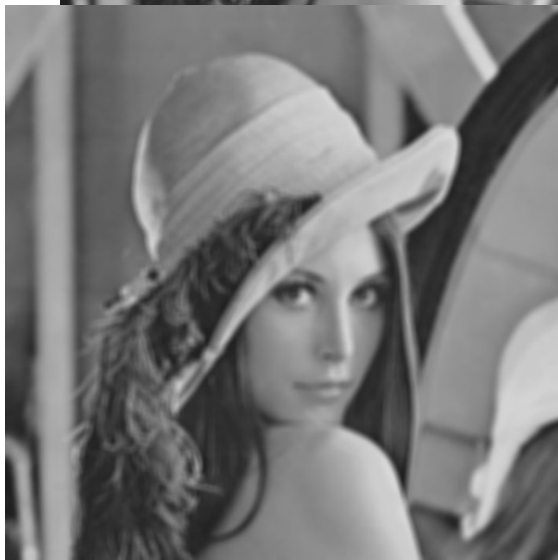
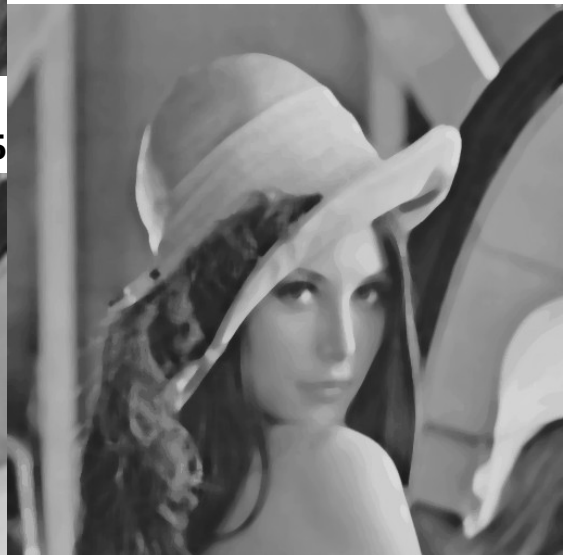
**boxfilter3\*3**



**median7\*7**

**median9\*9**

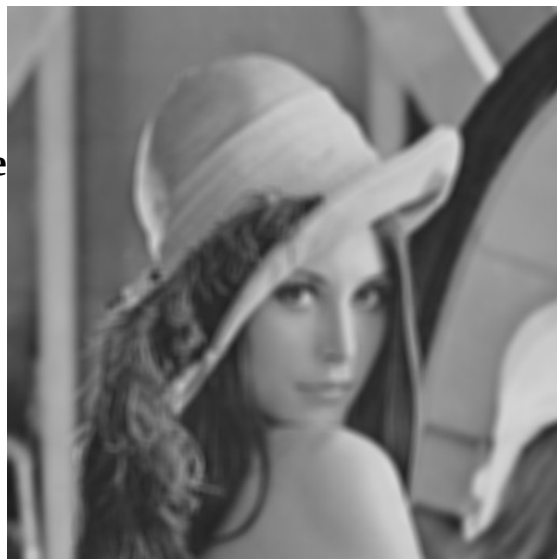
**box5\*5**



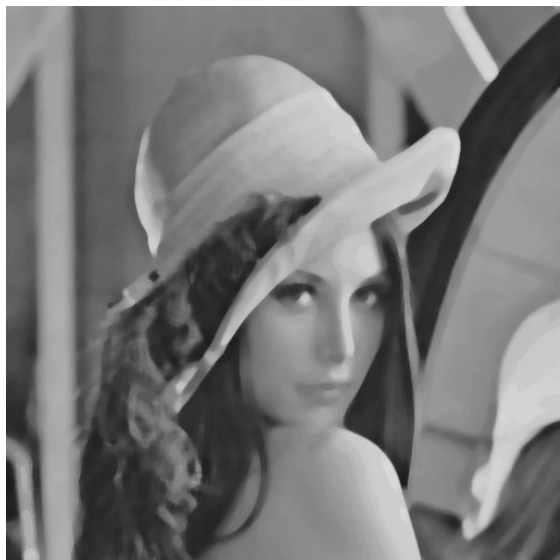
**0.  
1**

**boxfilter7\*7**

**box9\*9**

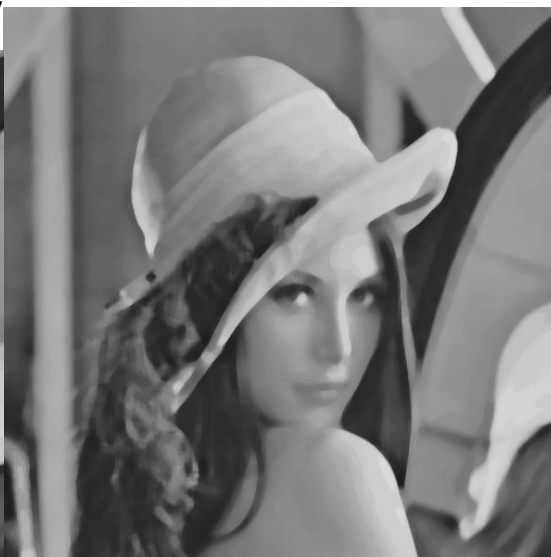
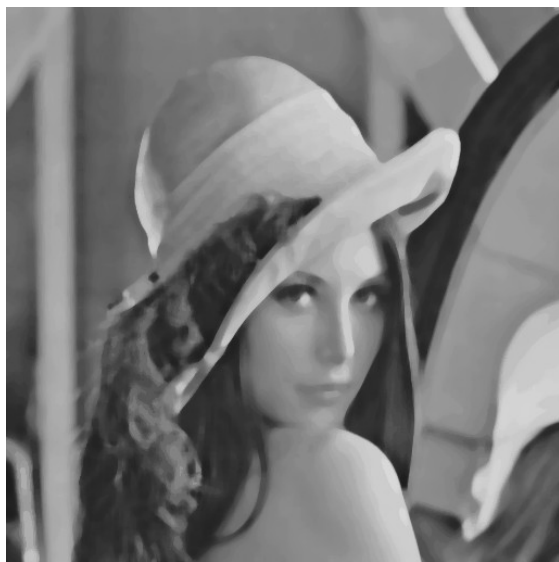


**median5\*5**



**median3\*3**

**median7\*7**

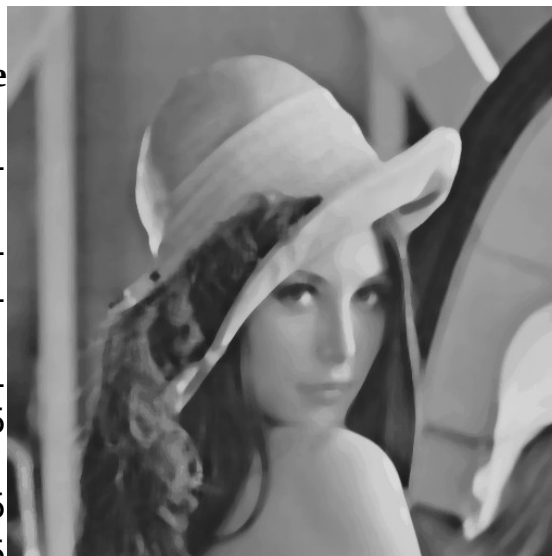


27.96 :7\*7&0.1  
 31.75 :9\*9&0.1

median9\*9

boxfiltermse

20.79 :3\*3&0.01  
 29.91 :5\*5&0.01  
 36.05 :7\*7&0.01  
 40.90 :9\*9&0.01  
 20.79 :3\*3&0.05  
 29.91 :5\*5&0.05  
 36.05 :7\*7&0.05  
 40.90 :9\*9&0.05



20.79 :3\*3&0.1  
 29.91 :5\*5&0.1

36.05 :7\*7&0.1  
 40.90 :9\*9&0.1

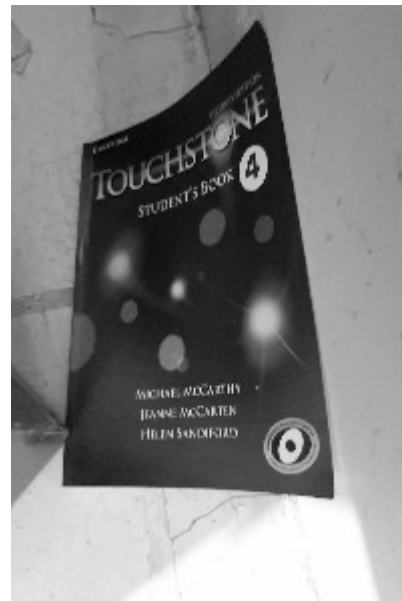
median mse

15.62 :3\*3&0.01  
 23.24 :5\*5&0.01  
 28.00 :7\*7&0.01  
 31.73 :9\*9&0.01  
 15.64 :3\*3&0.05  
 23.26 :5\*5&0.05  
 27.98 :7\*7&0.05  
 31.79 :9\*9&0.05  
 15.6414 :3\*3&0.1  
 23.25 :5\*5&0.1

در کل باکس فیلتر در 0.1 و 0.05 تقریباً مانند هم رفتار کردند و اینکه median فیلتر بهتری نسبت به باکس فیلتر است و نویز بیشتری میبرد

### 3.3.1: تصویر اصلی خوب

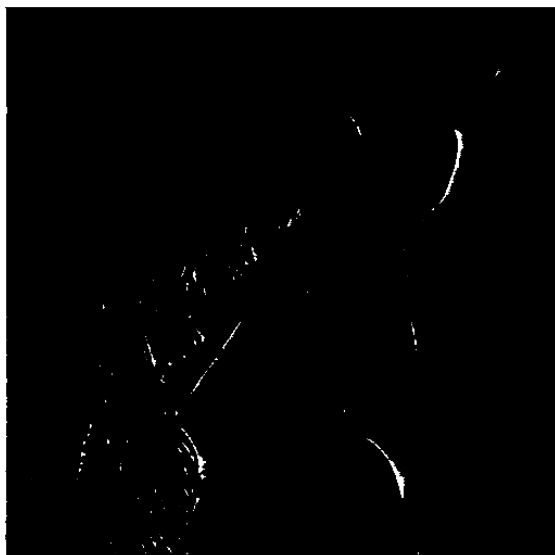
تصویر شارپ: با highboost



تصویر بد بی کیفیت blur

sharp and smooth





3.4:  
برای b



a



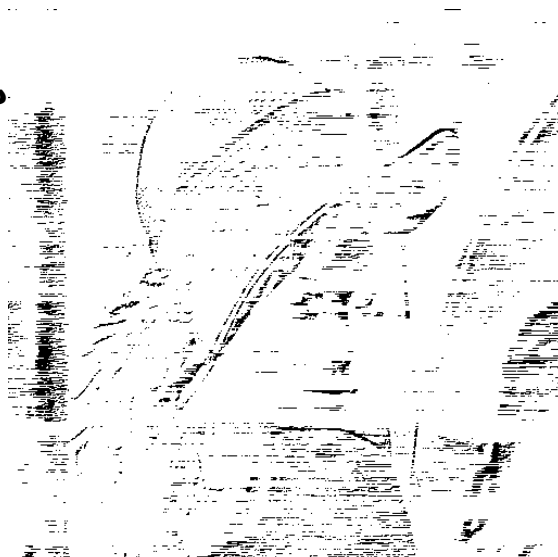
این تصویر حاصل می‌شود که لبه ها را  
نشان میدهد

این زیر هم برای قسمت c

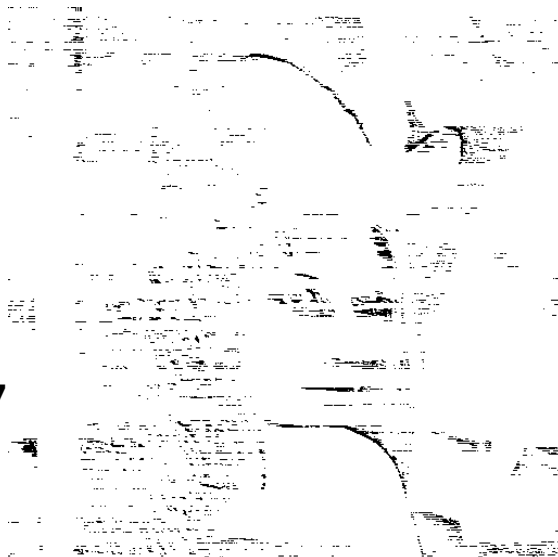
a در این لبه عمودی نشان میدهد

### 3.4.2:

a:



b:



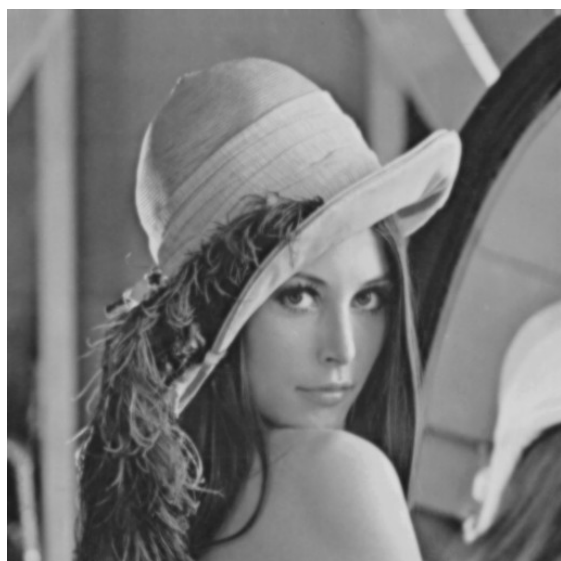
در  $a$  ,  $b$  لبه های مورب را نمایش  
میدهد

3.5: به چند حالت زیر توجه  
کنید:

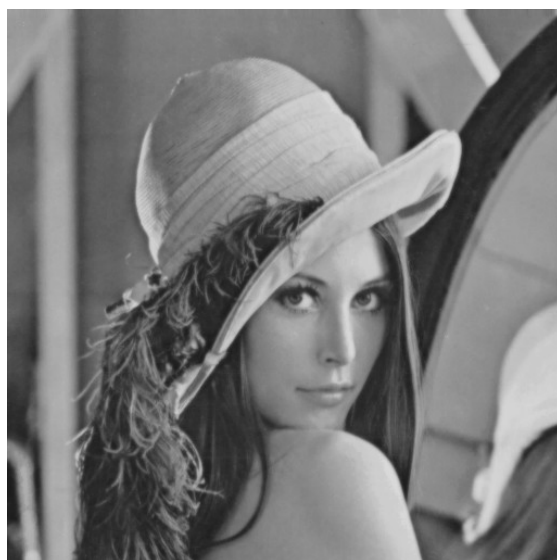


$a=0.099$  7\*7

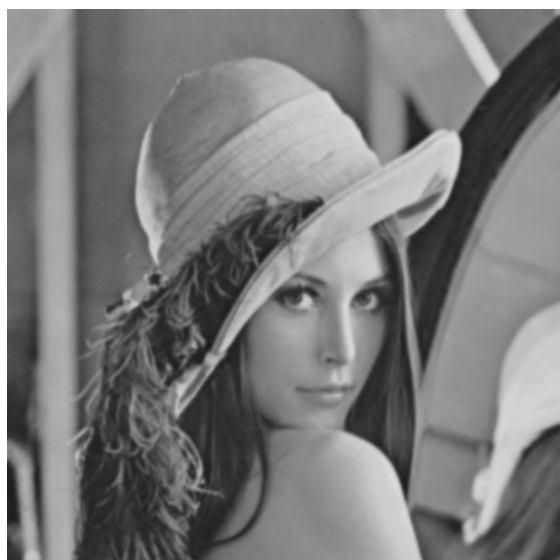




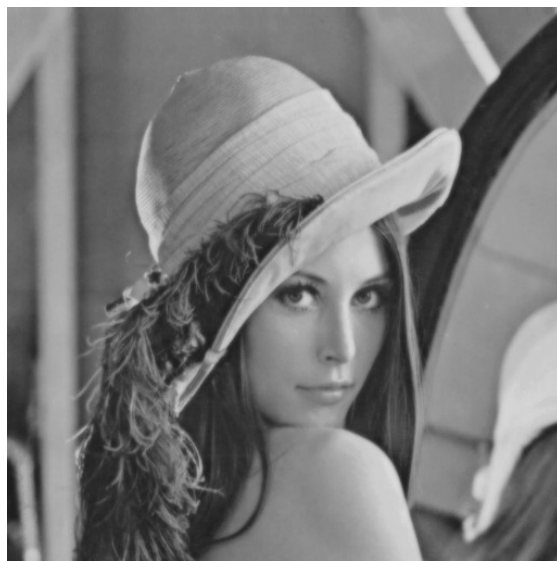
**a=0.86 7\*7 ,**



**a=0.6333& 7\*7**



**a=0.90 ,7\*7**



**a=0.6333 /9\*9**



**0.80a , 3\*3**

**a=0.25 3\*3**



**5\*5&0.2**

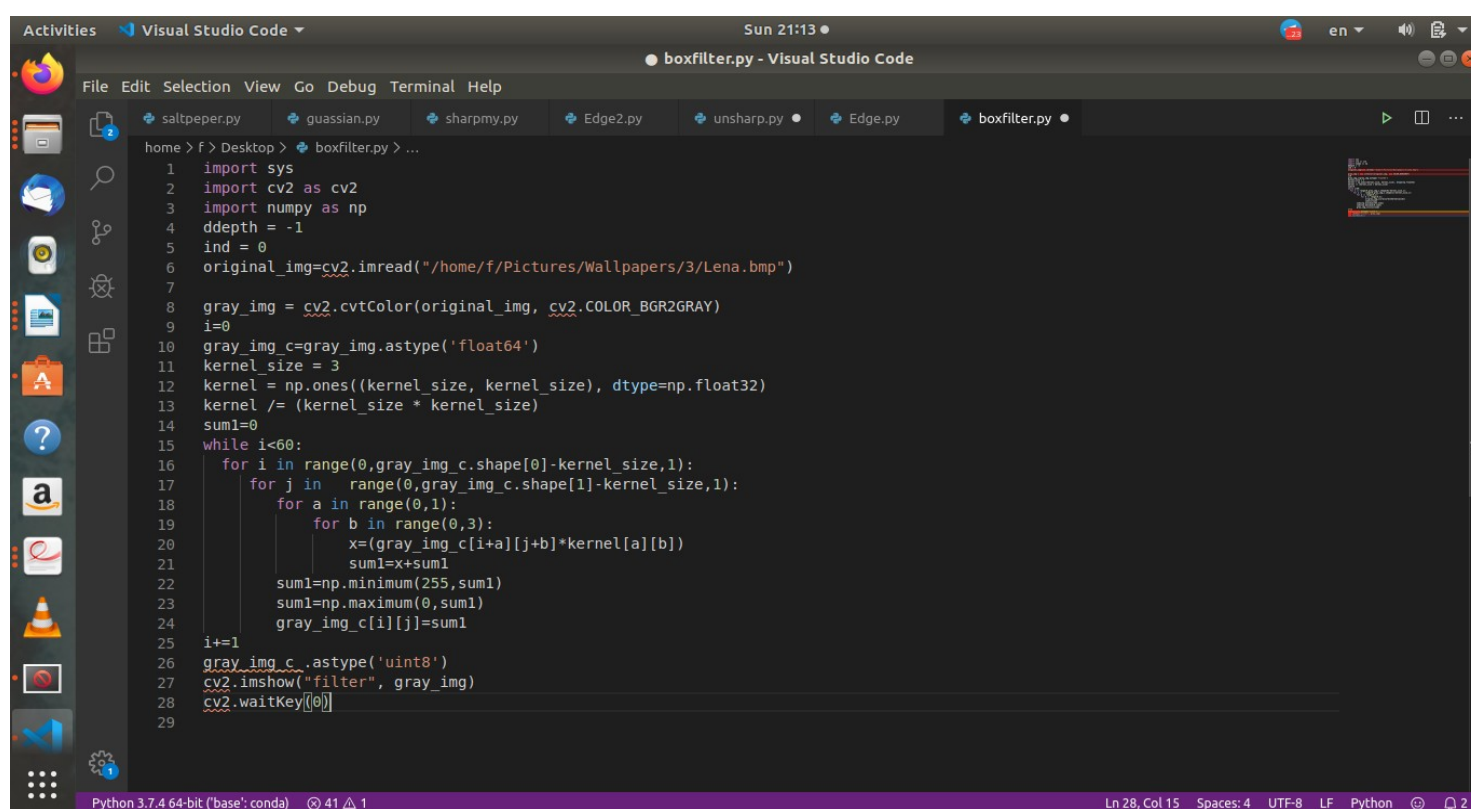


**0.80**  
**5&**  
**5\***

به طور کلی هرچه  $a$  کوچکتر بهتر است  
تصویر

#### 4- پیوست کد:

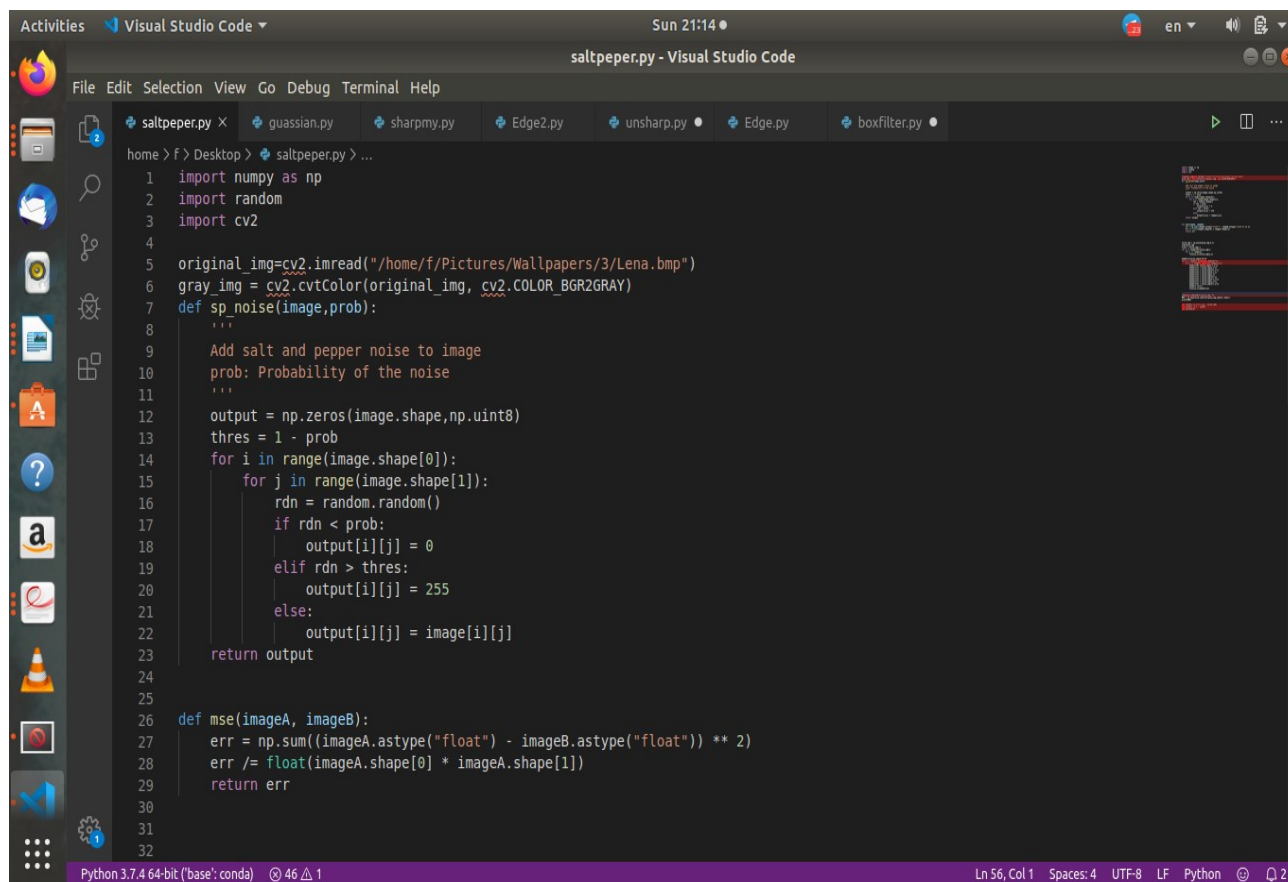
tamrin1



```
home > f > Desktop > boxfilter.py > ...
1 import sys
2 import cv2 as cv2
3 import numpy as np
4 ddepth = -1
5 ind = 0
6 original_img=cv2.imread("/home/f/Pictures/Wallpapers/3/Lena.bmp")
7
8 gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
9 i=0
10 gray_img_c=gray_img.astype('float64')
11 kernel_size = 3
12 kernel = np.ones((kernel_size, kernel_size), dtype=np.float32)
13 kernel /= (kernel_size * kernel_size)
14 sum1=0
15 while i<60:
16     for i in range(0,gray_img_c.shape[0]-kernel_size,1):
17         for j in range(0,gray_img_c.shape[1]-kernel_size,1):
18             for a in range(0,1):
19                 for b in range(0,3):
20                     x=(gray_img_c[i+a][j+b]*kernel[a][b])
21                     sum1=x+sum1
22                 sum1=np.minimum(255,sum1)
23                 sum1=np.maximum(0,sum1)
24                 gray_img_c[i][j]=sum1
25 i+=1
26 gray_img_c.astype('uint8')
27 cv2.imshow("filter", gray_img)
28 cv2.waitKey([0])
29
```

Python 3.7.4 64-bit ('base': conda) 41 1 Ln 28, Col 15 Spaces: 4 UTF-8 LF Python 2

## tamrin2 salt and pepper and median filter and gussian noise

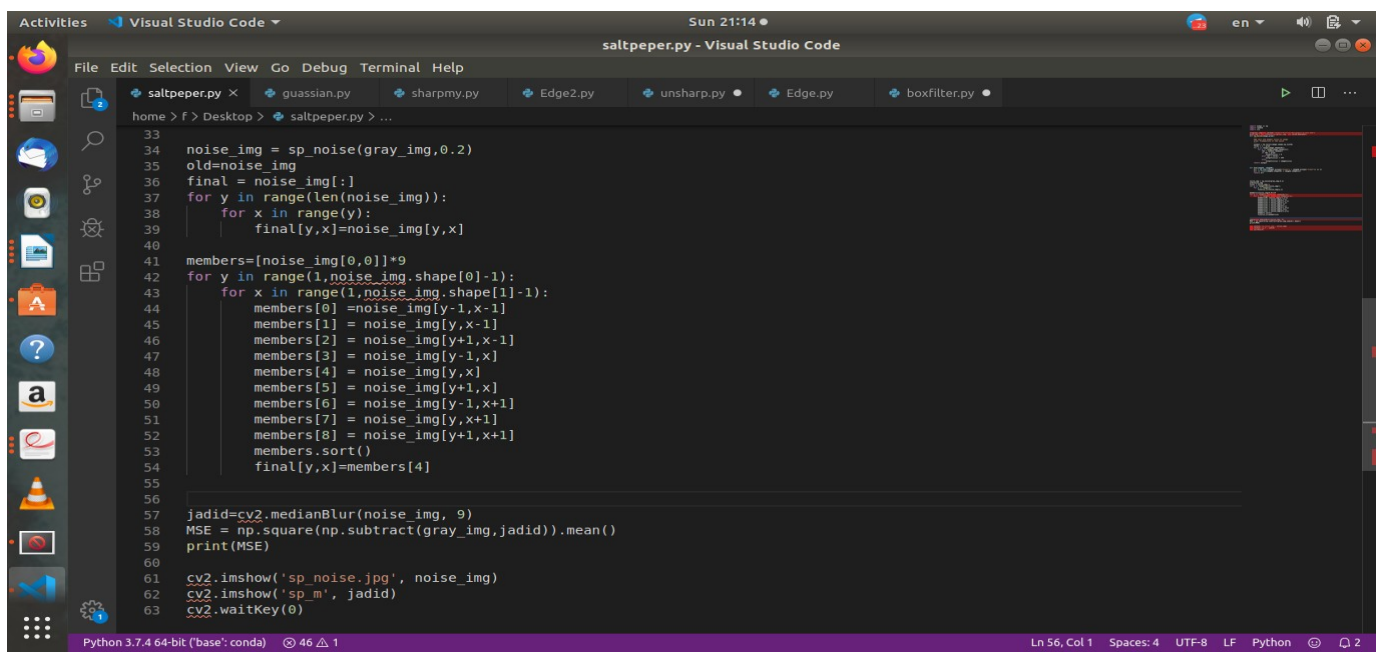


The screenshot shows the Visual Studio Code interface with a Python file named `saltpeper.py` open. The code implements a function `sp_noise` to add salt and pepper noise to an image. It also includes a function `mse` to calculate the Mean Squared Error between two images. The code is as follows:

```
1 import numpy as np
2 import random
3 import cv2
4
5 original_img=cv2.imread("/home/f/Pictures/Wallpapers/3/Lena.bmp")
6 gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
7 def sp_noise(image,prob):
8     '''
9     Add salt and pepper noise to image
10    prob: Probability of the noise
11    '''
12    output = np.zeros(image.shape,np.uint8)
13    thres = 1 - prob
14    for i in range(image.shape[0]):
15        for j in range(image.shape[1]):
16            rdn = random.random()
17            if rdn < prob:
18                output[i][j] = 0
19            elif rdn > thres:
20                output[i][j] = 255
21            else:
22                output[i][j] = image[i][j]
23    return output
24
25
26 def mse(imageA, imageB):
27     err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
28     err /= float(imageA.shape[0] * imageA.shape[1])
29     return err
30
31
32
```

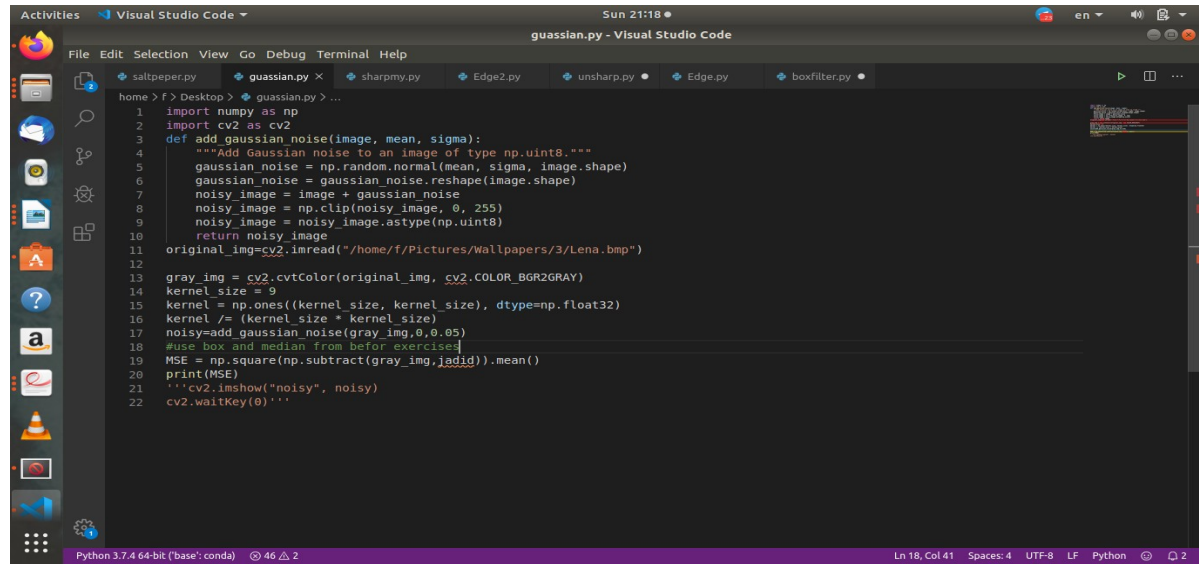
The status bar at the bottom indicates the Python 3.7.4 64-bit environment is active, and the cursor is at line 56, column 1.

## ادامه 2



```
33
34 noise_img = sp_noise(gray_img,0.2)
35 old=noise_img
36 final = noise_img[:]
37 for y in range(len(noise_img)):
38     for x in range(y):
39         final[y,x]=noise_img[y,x]
40
41 members=[noise_img[0,0]]*9
42 for y in range(1,noise_img.shape[0]-1):
43     for x in range(1,noise_img.shape[1]-1):
44         members[0]=noise_img[y-1,x-1]
45         members[1] = noise_img[y,x-1]
46         members[2] = noise_img[y+1,x-1]
47         members[3] = noise_img[y-1,x]
48         members[4] = noise_img[y,x]
49         members[5] = noise_img[y+1,x]
50         members[6] = noise_img[y-1,x+1]
51         members[7] = noise_img[y,x+1]
52         members[8] = noise_img[y+1,x+1]
53         members.sort()
54         final[y,x]=members[4]
55
56
57 jadid=cv2.medianBlur(noise_img, 9)
58 MSE = np.square(np.subtract(gray_img,jadid)).mean()
59 print(MSE)
60
61 cv2.imshow('sp_noise.jpg', noise_img)
62 cv2.imshow('sp_m', jadid)
63 cv2.waitKey(0)
```

gussian



```
1 import numpy as np
2 import cv2 as cv2
3 def add_gaussian_noise(image, mean, sigma):
4     """Add Gaussian noise to an image of type np.uint8."""
5     gaussian_noise = np.random.normal(mean, sigma, image.shape)
6     gaussian_noise = gaussian_noise.reshape(image.shape)
7     noisy_image = image + gaussian_noise
8     noisy_image = np.clip(noisy_image, 0, 255)
9     noisy_image = noisy_image.astype(np.uint8)
10    return noisy_image
11    original_img=cv2.imread("/home/f/Pictures/Wallpapers/3/Lena.bmp")
12
13    gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
14    kernel_size = 9
15    kernel = np.ones((kernel_size, kernel_size), dtype=np.float32)
16    kernel /= (kernel_size * kernel_size)
17    noisy=add_gaussian_noise(gray_img,0,0.05)
18    #use box and median from befor exercise
19    MSE = np.square(np.subtract(gray_img,noisy)).mean()
20    print(MSE)
21    cv2.imshow("noisy", noisy)
22    cv2.waitKey(0)'''
```



Activities Visual Studio Code Sun 21:18

sharpmy.py - Visual Studio Code

File Edit Selection View Go Debug Terminal Help

sharpmy.py X Edge2.py unsharp.py Edge.py boxfilter.py

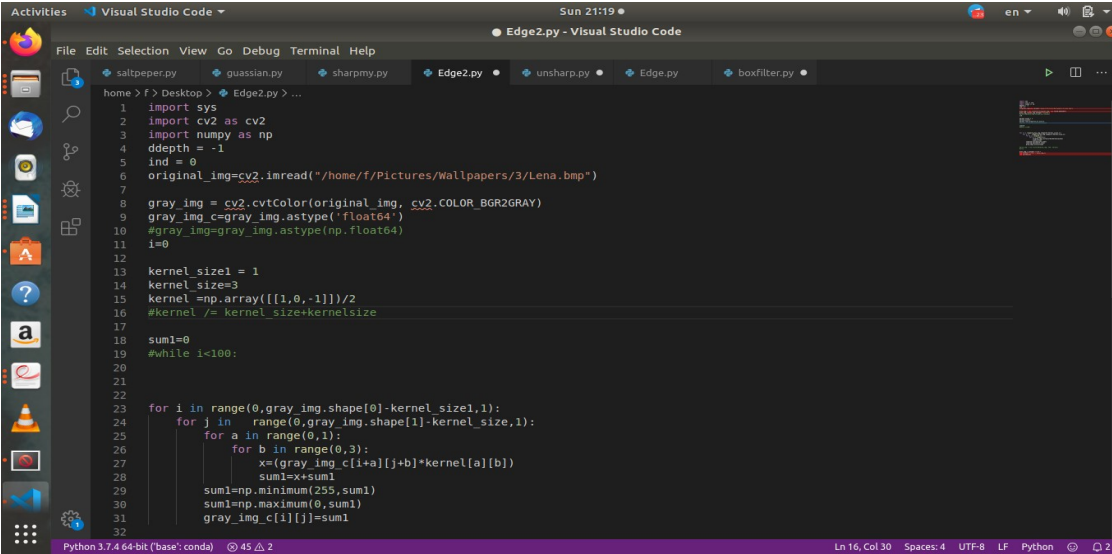
```
home > F > Desktop > sharpmy.py > ...
1 import numpy as np
2 import random
3 import cv2
4
5 original_img=cv2.imread("/home/f/Pictures/Wallpapers/3/me.bmp")
6 gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
7 original_img1=cv2.imread("/home/f/Pictures/Wallpapers/3/measl.jpg")
8 gray_img1 = cv2.cvtColor(original_img1, cv2.COLOR_BGR2GRAY)
9
10
11 gray_img = cv2.resize(gray_img, (200, 300))
12 gray_img1 = cv2.resize(gray_img1, (200, 300))
13 mask=gray_img1-gray_img
14 jadid=gray_img1+(mask)*3
15 #cv2.imshow('sp_noise.jpg', jadid)
16 cv2.imshow('sp_m', gray_img1)
17 cv2.waitKey(0)
```

Python 3.7.4 64-bit ('base': conda) 45 2 Ln 14, Col 25 Spaces: 4 UTF-8 LF Python 2

tamrin3

کم کردن  
تصویر  
کدر از  
تصویر  
اصلی و  
تولید  
ماسک

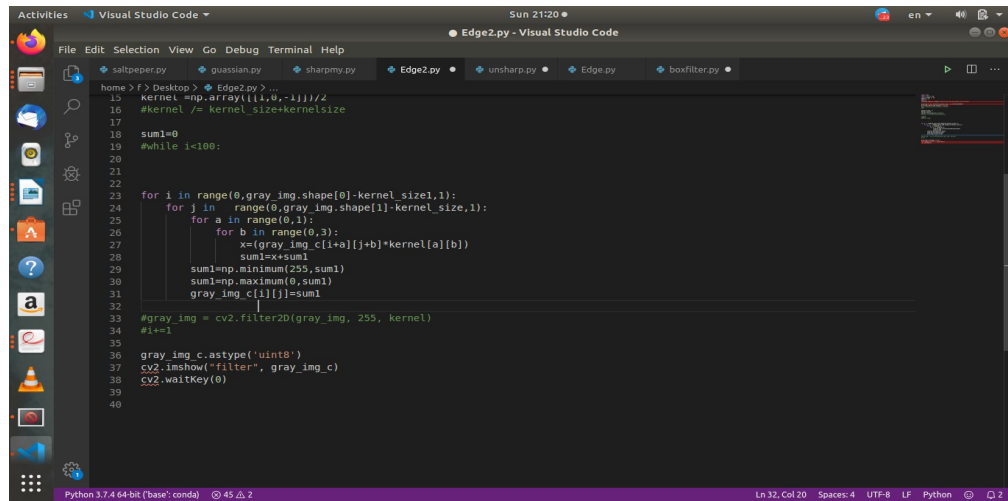
tamrin4



```
File Edit Selection View Go Debug Terminal Help
Edge2.py - Visual Studio Code
Sun 21:19
en
Edge2.py • unsharp.py • Edge.py • boxfilter.py
home > f > Desktop > Edge2.py > ...
1 import sys
2 import cv2 as cv2
3 import numpy as np
4 ddepth = -1
5 ind = 0
6 original_img=cv2.imread("/home/f/Pictures/Wallpapers/3/Lena.bmp")
7
8 gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
9 gray_img_c=gray_img.astype('float64')
10 #gray_img=gray_img.astype(np.float64)
11 i=0
12
13 kernel_size1 = 1
14 kernel_size=3
15 kernel =np.array([[1,0,-1]])/2
16 #kernel /= kernel_size+kernel_size
17
18 sum1=0
19 #while i<100:
20
21
22
23 for i in range(0,gray_img.shape[0]-kernel_size1,1):
24     for j in range(0,gray_img.shape[1]-kernel_size,1):
25         for a in range(0,1):
26             for b in range(0,3):
27                 x=(gray_img_c[i+a][j+b]*kernel[a][b])
28                 sum1=x+sum1
29             sum1=np.minimum(255,sum1)
30             sum1=np.maximum(0,sum1)
31             gray_img_c[i][j]=sum1
32
```

Python 3.7.4 64-bit (base: conda) 45 2 Ln 16, Col 30 Spaces: 4 UTF-8 LF Python 2

## تمرین 4 ادامه



```
15 kernel = np.array([[-1,0,-1]]//2)
16 #kernel /= kernel_size*kernel_size
17
18 sum1=0
19 #while i<100:
20
21
22
23 for i in range(0,gray_img.shape[0]-kernel_size,1):
24     for j in range(0,gray_img.shape[1]-kernel_size,1):
25         for a in range(0,1):
26             for b in range(0,3):
27                 x=(gray_img_c[i+a][j+b]*kernel[a][b])
28                 sum1=x+sum1
29             sum1=np.minimum(255,sum1)
30             sum1=np.maximum(0,sum1)
31             gray_img_c[i][j]=sum1
32
33 #gray_img = cv2.filter2D(gray_img, 255, kernel)
34 #i+=1
35
36 gray_img_c.astype('uint8')
37 cv2.imshow("filter", gray_img_c)
38 cv2.waitKey(0)
39
40
```

لبه یابی و ضرب ماتریس ها در  
عکس

```
home > f > Desktop > unsharp.py > ...
1  import cv2 as cv2
2  import numpy as np
3  original_img=cv2.imread("/home/f/Pictures/Wallpapers/3/Lena.bmp")
4
5  gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
6  gray_img_c=gray_img.astype('float64')
7  iprime= cv2.GaussianBlur(gray_img,(5,5),0)
8  iprime=iprime.astype('float64')
9  a=0.89
10
11  sharp=float(1-a)*gray_img_c+float(a)*iprime
12  sharp = np.maximum(sharp, np.zeros(sharp.shape))
13  sharp = np.minimum(sharp, 255 * np.ones(sharp.shape))
14  sharp = sharp.round().astype(np.uint8)
15
16  #gray_img_c.astype('uint8')
17  #iprime.astype('uint8')
18  cv2.imshow("filter", sharp)
19  cv2.waitKey(0)
20
21
```

Python 3.7.4 64-bit ('base': conda) 45 2 Ln 9, Col 7 Spaces: 4 UTF-8 LF Python 2

تمرین 5

استفاده از فرمول داده شده  
جایگذاری