بنفشه قلی نژاد

تمرین شماره دو.

98522328

سوال یک)

الف)

1- تعریف ماتریس رندم
2- تعریف کرنل های سوبل. حساس به لبه های افقی و عمودی

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

3- نوشتن تابع کانولوشن. (ضرب کرنل در تصویر و جمع همه ی مقادیر ناحیه کرنل)
4- اعمال کانولوشن بین کرنل و ماتریس :

```
matrix = generateRandomMatrix(10)
sobelx = convolve( matrix, gx)
sobely = convolve(matrix , gy)
```

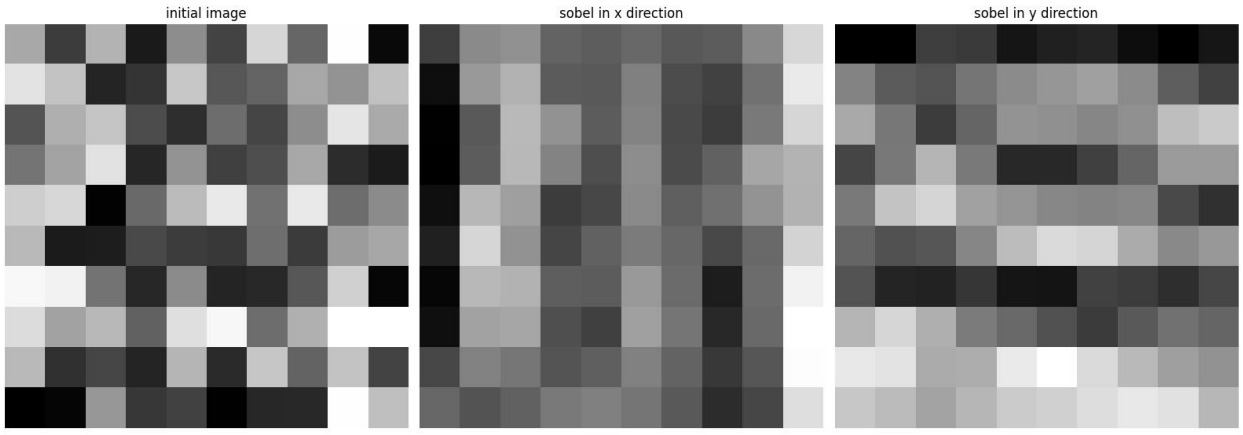
5- محاسبه ی جهت گرادیان:

```
direction = np.arctan2(sobelx , sobely)
```

6- محاسبه اندازه گرادیان.

```
value = np.power(sobelx, 2) + np.power(sobely , 2)

magnitude = np.power(value , 0.5)
```

نتیجه:



ب)

1- تعریف کرنل گاوسی 3*3 با کمک cv2.guassiankernel

```
cguassian = cv2.getGaussianKernel(3, 1)

array = np.array([[cguassian[0][0] , cguassian[1][0] , cguassian[2][0] ] ,

                  [cguassian[1][0]  , 1.0 , cguassian[1][0] ],

                  [cguassian[0][0] , cguassian[1][0] , cguassian[2][0]]])

coafficient = 1/np.sum(array)

kernel = coafficient * array
```

2- اعمال عملگر سوبل بر روی تصویر و مقایسه با حالت اعمال فیلتر گاوسی :

همان طور که مشخص است به علت نرم تر کردن تصویر، نویز را کم تر می کند و در نتیجه عملگر سوبل نتیجه بهتری خواهد داشت.

ج)

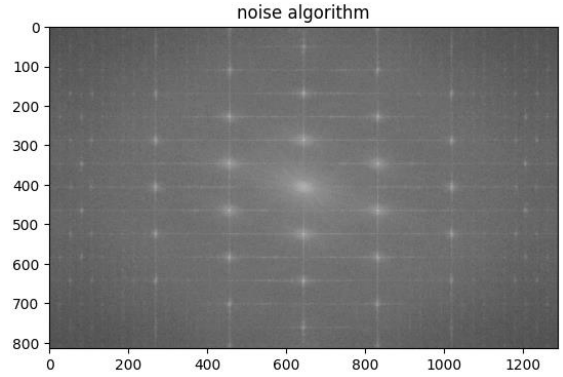
```
# do the operations in part b with OpenCV Sobel method and describe its parameters

sobelx = cv2.Sobel(src=image,  ddepth=cv2.CV_64F, dx=1, dy=0, ksize=3) # Sobel Edge Detection on the X axis

sobely = cv2.Sobel(src=image, ddepth=cv2.CV_64F , dx=0, dy=1, ksize=3) # Sobel Edge Detection on the Y axis

cv2im = sobelx+ sobely

combined = cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)
```

پارامتر ها به ترتیب تصویر ورودی ، عمق تصویر، که دراینجا برابر cv_64U است. به علت مقادیر منفی تصویر خاکسری استو صفر سیاه نیست. dy , dxبرای لبه های افقی و عمودی و مقدار اخر مقدار کرنل سایز است.

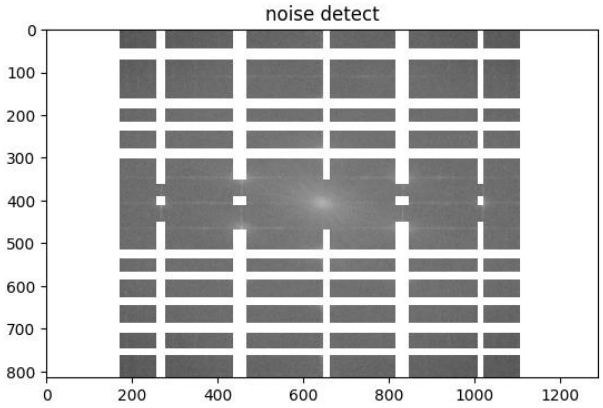
سوال دو)

1- تبدیل فضای مکانی به فضای فرکانسی با کمک تابع fft، اسفاده از log برای بهتر نشان داده شدن مقادیر انداره و شیفت دادن عکس به وسط تصویر .

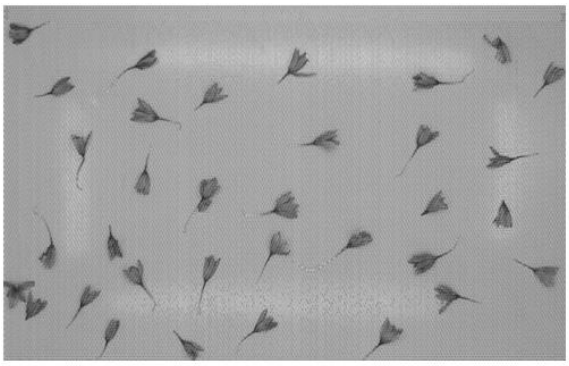
```
f = np.fft.fft2(image)
f_shift = np.fft.fftshift(f)
magnitude_spectrum = 20 * np.log(np.abs(f_shift))
```



2- تلاش برای کم تر کردن نویز های متناوب با فرکانس بالا



نتیجه ی نهایی:



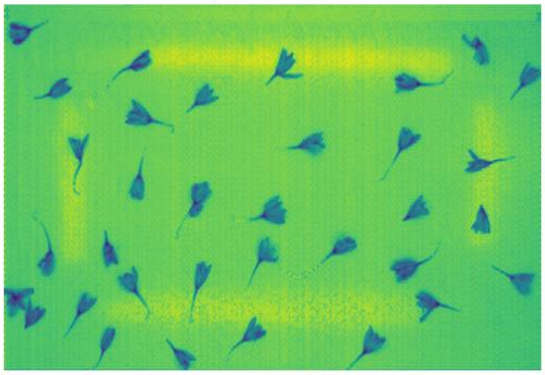
نویز ها روزنه های عمودی و افقی هستند که به صورت متناوب تکرار شده اند و وجود هر یک وابسته به دیگری است.خطوط سفید متناوب در قسمت هایی ک به خصوص بسیار روشن است نشان دهنده وجود نویز است.در مرکز تصویر به طور قطع جزییات اصلی تصویر است پس آن را خراب نمی کنیم.
پس بهتر است که خطوط عمودی و افقی به خصوص در قسمت خطوط پررنگ تر احتمال نویز وجود دارد.

ب)

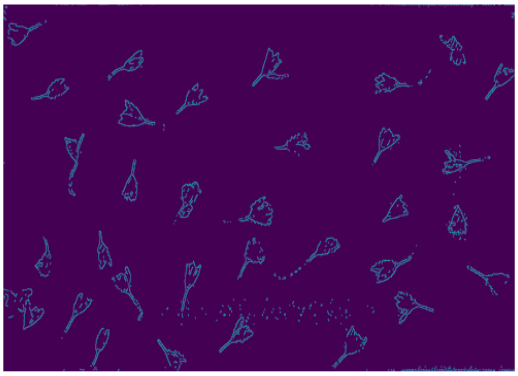
1- اعمال فیلتر گاوسی

2-اعمال لبه یاب کنی

gaussian filter



canny edge detector



```
edges = cv2.Canny(slice1,90,110)
```

پارامتر اول تصویر ، پارامتر دوم حد استانه گذاری اول و پارامتر دوم حد استانه گذاری دوم است.

مراحل انحام عملگر کنی:

-1 فیلتر گاوسی

2- گرادیان تصویر

3-حذف مقادیر غیر بیشینه

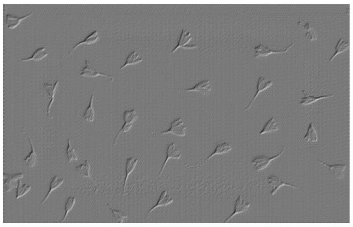
4-آستانه گذاری دو مرحه ای : دو آستانه تعریف میشود. اگر کم تر از استانه ی اول بااشد که لبه نیست. اگر بیشتر باشد به طور قطع لبه است و اگر بین این دو باشد بستگی به پیکسل های همسایه وضعیت متفاوتی پیدا می کند .

ج)

با کمک عملگر سوبل گرادیان در جهت ایکس و ایگرگ را محاسبه نموده و با فرمول اندازه 0,5**(gx**2 + gy**2) و فرمول arctan2(gx , gy) اندازه و جهت گرادیان را محاسبه می کنیم.

```
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=3) # Sobel Edge Detection on the X axis
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=3) # Sobel Edge Detection on the Y axis
answer = np.arctan2(sobelx , sobely)*180 /np.pi
gradiantvalue =  cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)
```

Gradiant magnitude :



جهت گرادیان نیز یک ماتریس می سازد که در نوت بوک other_Qsنتیجه مشخص است.

د) می توان با توجه به مقادیر جهت گرادیان ، مقادیری که کنار هم هستند و جهت گرادیان یکسان دارند(روی یک خط) آن ها را تاجایی که نزدیک به هم هستند حذف کرد. یعنی ترکیبی از تشخیص خطوط با کمک گرادیان تصویر و همچنین با کمک گرفتن از کنی.

سوال 3)

الف) فیلتر های پایین گذر:

فیلترهای پایین گذر (LPF) آن دسته از فیلترهای فضایی هستند که تأثیر آن ها بر روی تصویر خروجی معادل کاهش مولفه های فرکانس بالا (جزئیات ظریف در تصویر) و حفظ مولفه های فرکانس پایین (کلیات تر و نواحی همگن در تصویر) است. این فیلترها معمولاً برای مات کردن تصویر یا کاهش میزان نویز موجود در تصویر بکار گرفته می شوند. معمولا فیلترهای پایین گذر خطی را می توان با استفاده از ماسک های کانولوشن دو بعدی با ضرایب غیر منفی پیاده سازی کرد.از این فیلتر ها برای از بین بردن نویز به کار برده میشود و تصویر ارام تر و نرم تر میشود

فیلترهای بالاگذر (HPF) به طور مکمل و در نقطه مقابل فیلتر های پایین گذر (LPF) عمل می کنند، یعنی این فیلترها مولفه های فرکانس بالا را با اثرات احتمالی افزایش پیکسل های نویزی نیز حفظ یا بهبود می بخشند. اجزای فرکانس بالا شامل جزئیات ظریف، نقاط، خطوط و لبه ها هستند. به عبارت دیگر، این کار، تغییرات شدت روشنایی تصویر را برجسته می کند

این فیلتربرای برجسته کردن لبه های تصویر به کار برده میشود.

ب) فیلتر بالاگذر: بر فرکانس های پایین تاثیر گذاشته و تصویر برجسته تر است.

ج)نویز های افزایشی ، نویز هایی هستند که به تصویریا سیگنال اصلی اضافه میشوند یعنی :

$$x[n]=s[n]+vnoise[n]$$

مانند نویز سفید و یا نویز گاوسی که اکثرا با روش های فیلتر کردن می توان آن ها رااز بین برد .

نویز ضربی نویز ضرب شونده است که از ضرب نویز در سیگنال به دست می آید در نتیجه از بین بردن آن سخت تر است.

$X[n] = s[n]*vnoise[n]$و از روش های از بین بردن آن می توان اشاره کرد که آن را به نویز خطی تبدیل کنیم.

د) اگر تصویری دارای نویز نمک و فلفل یا Salt and Pepper باشد،(مقادیر 0 یا 1) آنگاه نقاط سیاه و سفیدی در اکثر نقاط تصویر خواهد افتاد. این نقاط سیاه و سفید بر روی پیکسل های تصویر اصلی می افتند و کیفیت تصویر اصلی را پایین می آورند. بنا بر درصد نویزی که روی تصویر افتاده است، پراکندگی این نقاط سیاه و سفید کم یا زیاد می شود. با کمک فیلتر median می توان این نویز را از بین برد.

سوال 4)

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

N=2, M=2

F(0.0) = f(0,0) + f(0,1) + f(1,0) + f(1,1) = 4 + 0 + 3 + 2 = 9

الف) چون U و V هر دو صفر هستند، مقدار F(U,V) مجموع مقدار های f(x,y) است که ۹ شده است.

ب)

F(0,1) = f(0,0) + f(0,1) * (e ^ $-2\pi j(0,5)$) + f(1,0) * (e ^ $-2\pi j(0)$) + f(1,1) * (e ^ $-2\pi j(0+0,5)$) = 4 + 0 + 3 - 2 = 5

F(1,0) = f(0,0) + 0 + f(1,0) * (e ^ $-2\pi j(0,5)$) + f(1,1) * (e ^ $-2\pi j(0,5+0)$) = -1

F(1,1)=f(0,0) + 0 + f(1,0) * (e ^ $-2\pi j(0,5)$) + f(1,1) * (e ^ $-2\pi j(1)$) = 3

سوال 5) کانتور به عنوان خطی تعریف می شوند که تمام نقاطی را در امتداد مرز یک تصویر که شدت یکسانی دارند به هم می پیوندد. کانتورها در تجزیه و تحلیل شکل، یافتن اندازه شی

مورد نظر و تشخیص شی مفید هستند. OpenCV دارای تابع findContour() است که به استخراج خطوط از تصویر کمک می کند.

برای استفاده از کانتور باید تصویر باینری باشد در نتیجه تکنیک های استانه گذاری و تبدیل عکس به باینری باید اعمال شود درنتیجه
1-تبدیل عکس به سیاه سفید
2- استانه گذاری و اعمال لبه یاب کنی برای تصویر:

```
im = cv2.Canny(image, 100, 140)
   ret, thresh = cv2.threshold(im, 127, 255, 0)
   contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

پارامتر سوم مشخص می کند که به دنبال چه هستیم. برای مثال پارامتر سوم نقاط مری اشکال را مشخص می کند. در صورت استفاده از cv.approx_Noneمرز اشکال مشخص می شود.

1- صدا کردن تابع findcontours برای پیدا کردن گوشه ی نقاط

2- یک لوپ برای کانتور

با توجه به تعداد و مرز نقاط . برای هر شکل چهار وریبل تعریف میکنیم و در صورتی که تعداد نقاط کامنور برابر تعداد نقاط هر شکل باشد. آن را به if else ifاستفاده از چند if نوع شکل را مینویسد. همچنینی اولین نقطه ی هر شکلی که پیدا میکنیم شروغ کننده برای نوشتن اسم شکل است. cv2.puttext()لیست اصلی اضافه می کند و با کمک

```
len(approxpoly) == 3 :
        if(triangle is None):
           write_text(image , 'triangle', (point_x, point_y))
        triangle = approxpoly;
```

3- برای تشخیص مربع از مستطیل از تابع cv2.boundingrectاستفاده می کنیم. در صورتی که نسبت اضلاع 1 باشد مربع باشد. برای پیدا کردن مستطیل این شرط لازم نیست

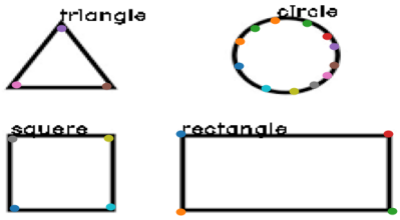
```
if len(approxpoly) == 4 and area > max_area:
            if(approx_contour is None):
               write_text(image , 'rectangle', (point_x, point_y))
            biggest = approxpoly
            max_area = area
            approx_contour = biggest
```

4- برای دایره در صورتی که تعداد نقاط از 6 بیشتر باشد، شکل دایره است.

```
if(len(approxpoly) >= 6):
        if(circle is None):
            write_text(image , 'circle', (point_x, point_y))
        circle = approxpoly
```

لیست هر شکل را داخل یک لیست کلی میریزیم و روی آن فور میزنیم. با کمک plt.scatter، (یا می توان از drawcontour) استفاده کرد.

نتیجه کلی:



ج) یکی از وجه های تمایز هر شکل از شکل دیگر ، تعداد گوشه های آن است.می توانیم برای شناخت کلی هر شکل (چند گوشه ای بودن) از روش بالا بهره بگیریم.

همچنین برای تمایز هر شکل چند گوشه ای میتوان از ویژگی هایی مثل اندازه اضلاع و یا زوایا که با داشتن نقاط گوشه قابل محاسبه است از هم متمایز کرد.

سوال 6)

الف) برای زدن تابع رفلکت101 .gfedcb|abcdefg|fedbcaیعنی برای گسترش تصویر از هر طرف یک ردیف قبلی آن برعکس میشود.

1- ساختن تصویر به اندازه imgsize + (kernel//2)*2

2- گسترش سمت راست و چپ تصویر به اندازه size//2 و به صورت تکرار قرینه از یک ردیف قبلی

3- گسترش بالا و پایین تصویر به اندازه size//2 و به صورت تکرار و قرینه از ردیف های بالاو پایین.

4- پر کردن گوشه های تصویر و به صورت قرینه ی قطری آن.

```
padding = filter_size //2
 image = img.copy()
 w, h = img.shape
  leftside_padding = np.fliplr(img[ :  ,h-padding -1  :h -1])
 rightside_padding = np.fliplr(image[: , 1:padding+1])
 upside_padding = np.flipud(image[1:padding+1 , :])
 downside_padding =np.flipud( image[w-padding-1:w-1, : ])
 new_image = np.zeros((w+(2*padding) , h+(2*padding)));
 w,h = new_image.shape
 new_image[padding:w-padding , padding : h-padding] = image
 new_image[padding:w-padding ,h-padding:h ] = leftside_padding
```

```
new_image[padding:w-padding ,0:padding ] = rightside_padding
new_image[0:padding,  padding:h-padding ] = upside_padding
new_image[w-padding:w, padding : h-padding ] = downside_padding
new_image[w-padding:w , 0:padding] = np.flipud(np.fliplr(image[image.shape[0] - padding -1:image.shape[0]-1 ,1:padding+1]))
new_image[0:padding , 0:padding] = np.flipud(np.fliplr(image[1:padding+1 ,1:padding+1]))
new_image[0:padding ,h-padding:h] = np.flipud(np.fliplr(image[1:padding+1 ,image.shape[1] - padding -1:image.shape[1]-1]))
new_image[w-padding:w ,h-padding:h] = np.flipud(np.fliplr(image[image.shape[0] - padding -1:image.shape[0]-1 ,image.shape[1] -
padding -1:image.shape[1]-1]))
```

فیلتر میانگین گیر:

1- تعیین کرنل میانه گیر (برای جمع همه ی مقادیر در بازه ی کرنل و تقسیم بر تعداد) : عنصری با اندازه ی filter_size با عناصر 1

2- اعمال کانولوشن بر روی کرنل. به طوری که در هر بازه ی x-x+filtersize مقادیر تصویر و کرنل را ضرب می کند و جمع میکند. نتیجه ی تصویر یک تصویر تار شده است

فیلتر میانه گیر:

این فیلتر یک فیلتر غیر خطی است و در آن از سورت استفاده می شود. در هر بازه به اندازه یک پنجره به طول filter_sizeمیانه ی همه ی مقادیر در آن را به جای پیکس متناظر پاسخ قرار میدهیم.

فیلتر گوسی:

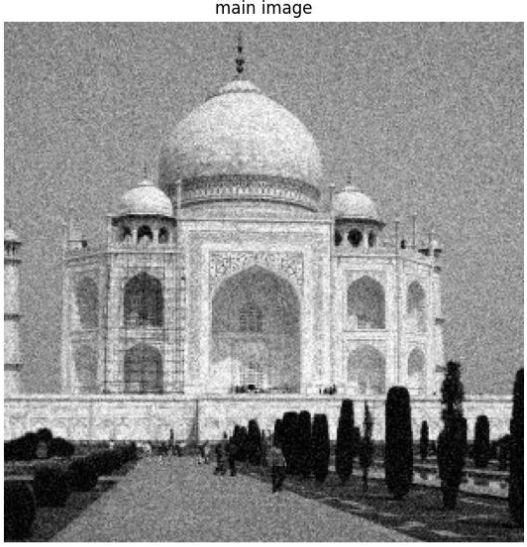
فیلتر گوسی یک فیلتر میانگیر گیر با ضرایب است که هر چقدر از مرکز کرنل دور میشود مقادیر داخل کرنل کم رنگ تر شده و از فرمول گاوسی پیروی می کند.

با کمک np.linspace، فواصل هر نقطه را از مرکز با کمک فرمول فاصله محاسبه کرده و آن را در قرمول گاوسی جا گداری میکنیم تا کرنل به اندازه خواسته شده اماده گردد و آن را با filter2dکانوالو می کنیم.

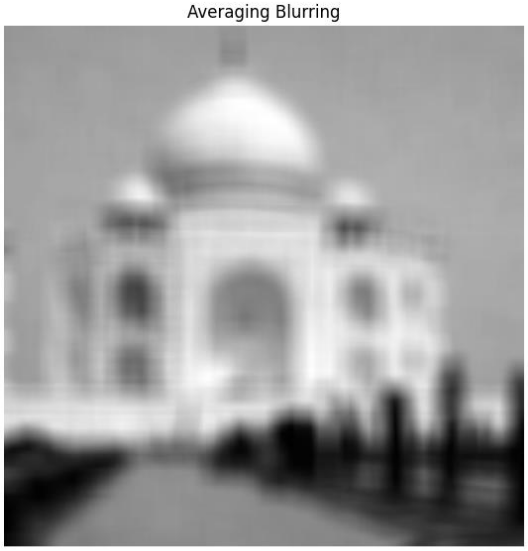
```
x, y = np.meshgrid(np.linspace(-1, 1, kernel_size),np.linspace(-1, 1, kernel_size))
    dst = np.sqrt(x**2+y**2)
    normalize = 1/(2 * np.pi * sigma**2)
  return np.exp(-((dst-muu)**2 / (2.0 * sigma**2))) * normalize
```

با بزرگ تر شدن کرنل. به علت این که در هر ایترشین برای هر پیکسل ، بازه ی پیکسل های اطراف گسترش یافته اند، کرنل ها تاثیر بیشتری بر روی هر پیکس دارند در نتیجه تصویر تار تر میشود ، نویز ها بیشتر از بین میروند اما تصویر تار خواهد شد و جزییات آن از بین میرود.

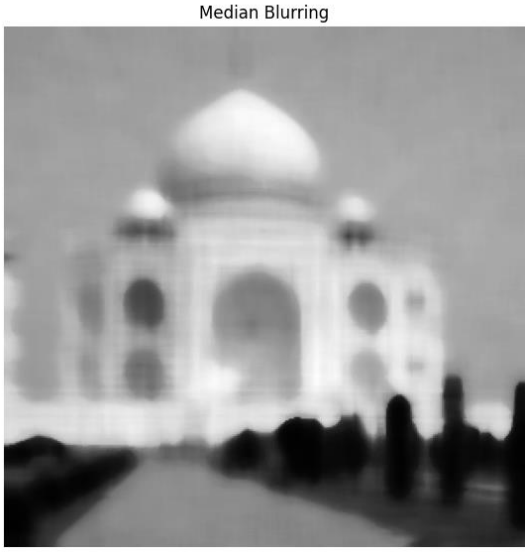
main image



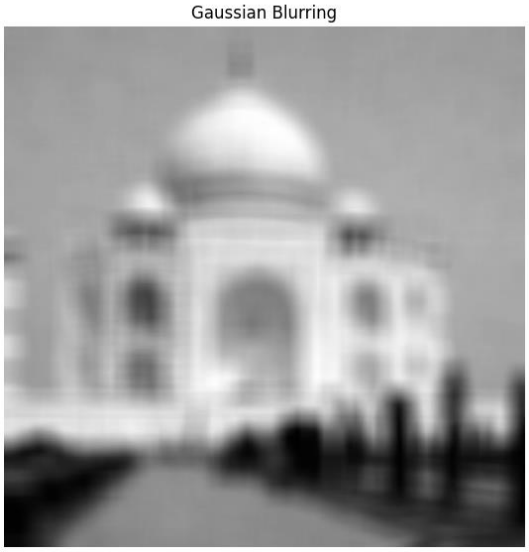
Averaging Blurring



Median Blurring



Gaussian Blurring



ب) فیلتر bileterial فیلتری شبیه به فیلتر گوسی است. اما با این تفاوت که علاوه بر صاف کردن تصویر و از بین بردن نویز لبه ها تا حند خوبی مشخص هستند.فرمول آن به شکل زیر لست.

The bilateral filter is defined as[1][2]

$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$

and normalization term, W_p, is defined as

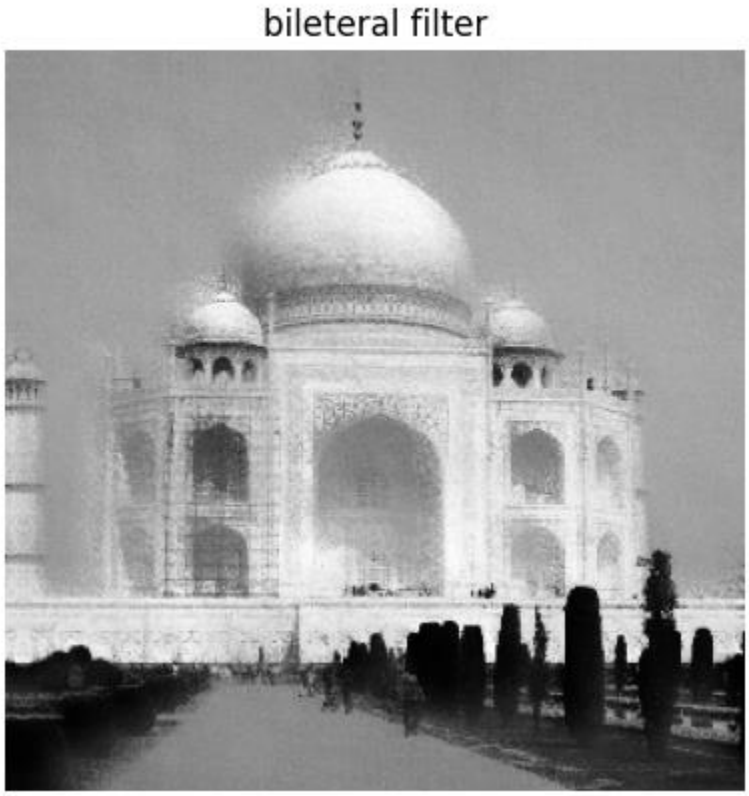
$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

where

I^{filtered} is the filtered image;
I is the original input image to be filtered;
x are the coordinates of the current pixel to be filtered;
Ω is the window centered in x, so $x_i \in \Omega$ is another pixel;
f_r is the range kernel for smoothing differences in intensities (this function can be a Gaussian function);

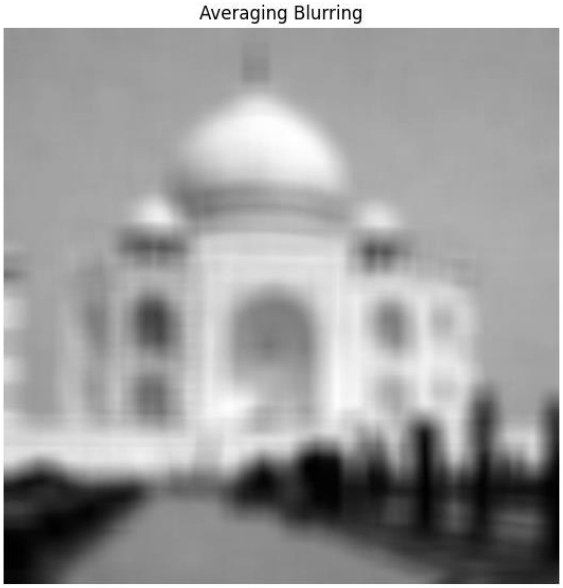
این فیلتر شدت هر پیکسل را با میانگین وزنی مقادیر شدت پیکسل های مجاور جایگزین می کند. با افزایش پارامتر دامنه σr، فیلتر دو طرفه به تدریج به پیچیدگی گاوسی نزدیک‌تر می‌شود، زیرا محدوده گاوسی گسترده و مسطح می‌شود، به این معنی که در بازه شدت تصویر تقریباً ثابت می‌شود.ا افزایش پارامتر فضایی σd، ویژگی های بزرگتر صاف می شوند.

پیاده سازی این فیلتر بر اساس فرمول بالا است. به طوری که در یک بازه به اندازه filter_size * filter_size، عدد گوسی اختلاف شدت روشنایی هر پیکسل و پیکس های همسایه با ضریب sigmar و فاصله هر نقطه از نقطه همسایه با فرمول گوسی و سیگمای d محاسبه میشود. این دو مقداردر هم ضرب میشوند و این مقادیر نیز تحت عنوان ضرایب جمع شده و در مقدار جمع شده ی ضرب این مقدار در هر پیکسل آن بازه تقسیم میشود.

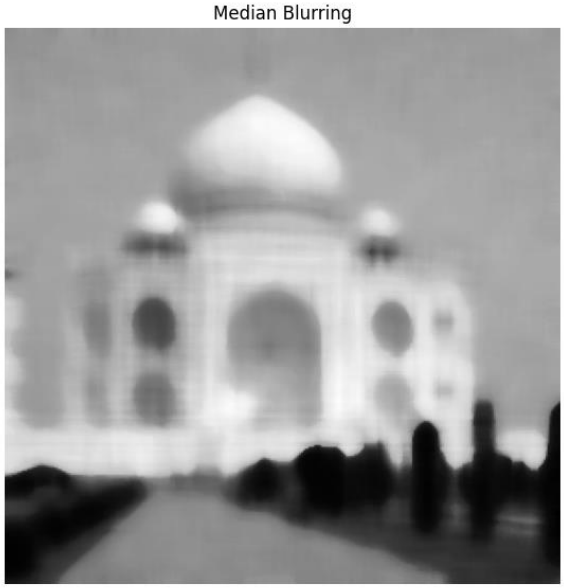


د) مقایسه opencv و نتیجه اصلی

Averaging Blurring



Median Blurring



Gaussian Blurring

Bilateral