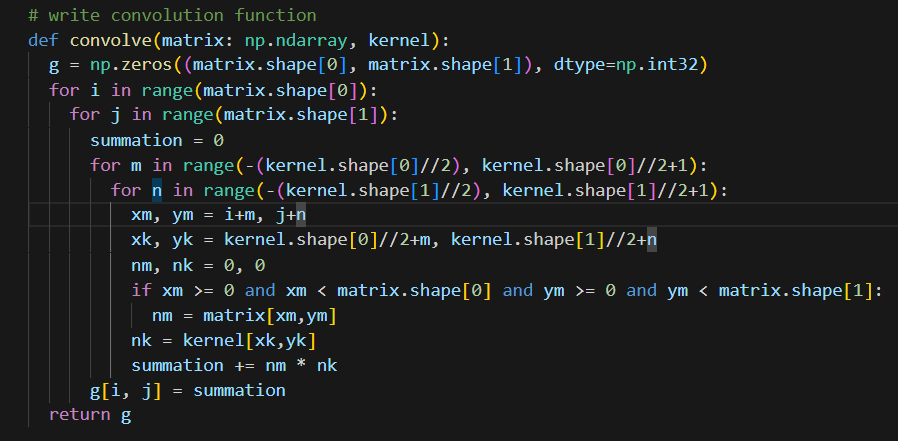
به نام خدا

گزارش تمرین دوم

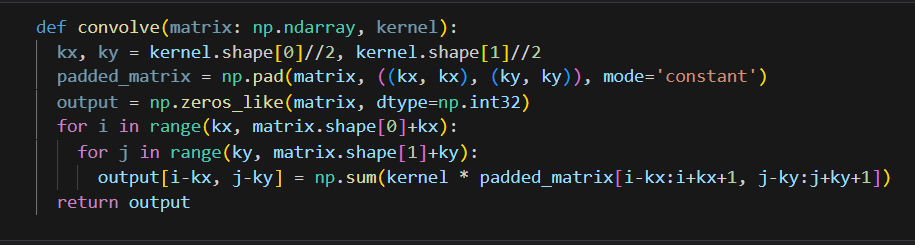
سینا علی‌نژاد

سوال 1

الف) برای این بخش ابتدا یک ماتریس n\*n رندوم درست کردم و سپس تابع convolve را پیاده سازی کردم. اولین پیاده‌ سازی که از چهارتا حلقه تشکیل شده بود، حتی برای عکسهای با رزولوشن کم خیلی طول میکشید. پس از یک پیاده سازی دیگه که فقط از دو حلقه تشکیل میشد استفاده کردم و نسبتا بهتر شد.



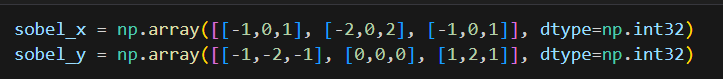
Naïve



Improved

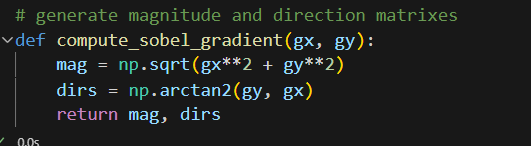
لازم به ذکر است که اینجا از zero padding استفاده کردم و برای آن از تابع np.pad با مود constant استفاده کردم.

سپس کرنل های افقی و عمودی سوبل را طبق اسلاید تعریف کردم.

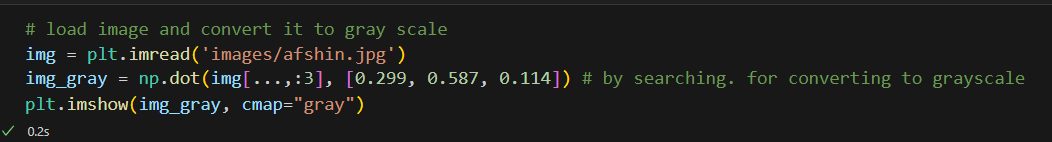


سپس عملیات کانولوشن را بین ماتریس و هر کدام از این کرنل‌ها انجام دادم.

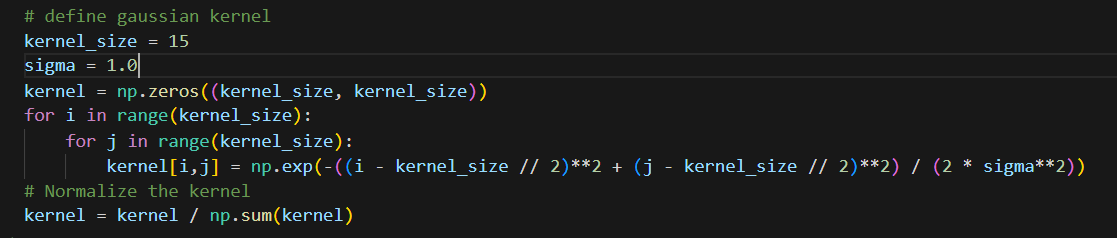
اکنون که سوبل افقی و عمودی اعمال شد، باید اندازه و جهت گرادیان را به کمک آن بدست آوریم.



ب) ابتدا تصویر را خواندم و به کمک فرمولی که با سرچ پیدا کردم، grayscale کردم.

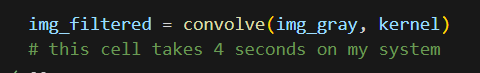


سپس کرنل گاوسی را طبق فرمول اسلاید بدست آوردم.

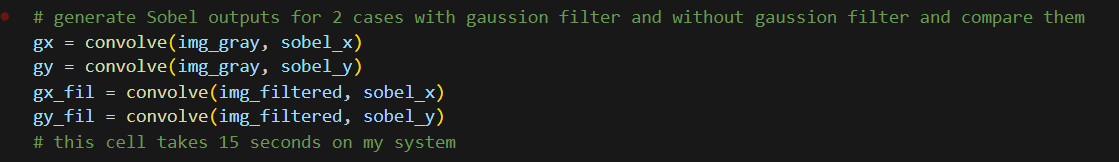


در خط آخر نرمالایز کردم. سایز کرنل را 15 در 15 گرفتم زیرا 3 \* 3 تاثیر زیادی روی خروجی عکس نگذاشت. مقدار سیگما را نیز برابر با 1 گذاشتم تا میانگین گیر معمولی نشود. در اینجا از دو حلقه استفاده کردم، میتونستم از حلقه استفاده نکنم و با نامپای بزنم. مشابه این کار رو توی سوال 6 برای فیلتر bilateral انجام دادم که اومدم کرنل مربوط به اون رو بدون هیچ حلقه‎‌ای بدست آوردم، در آنجا بدست آوردن کرنل گاوسی هم بخشی از بدست آوردن کرنل bilateral بود.

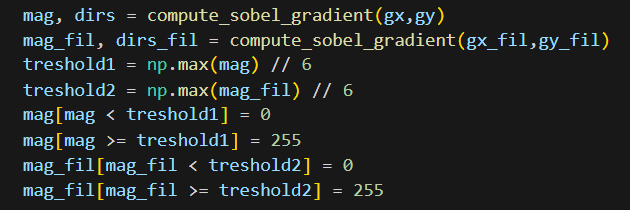
سپس عکس را با این کرنل کانوالو کردم.



و عکس حاصل از این فیلتر را با سوبل عمودی و افقی کانوالو کردم.



بعد از آن، اندازه گرادیان و جهت آن را برای هر پیکسل بدست آوردم.



اینجا مقادیر اندازه گرادیان ماکزیمم به 800 میرسید، یک مرزی که یک ششم ماکزیمم باشد در نظر گرفتم و پیکسل های روشن تر از آن را لبه و بقیه را غیرلبه در نظر گرفتم. این کار را برای عکسی که فیلتر گاوسی شده بود هم انجام دادم.



در نتیجه نهایی میتونیم مشاهده کنیم که در عکسی که فیلتر گاوسی قبل از لبه یابی سوبل روی آن اجرا شد، بعضی لبه‌های جزئی تر تشخیص داده نشده‌اند یا قوی تشخیص داده نشده اند. برای مثل آویز گردنبند در عکس سمت راست به اندازه عکس سمت چپ قوی نیست. البته مطمئن نیستم اینجا تبدیل گاوسی مفید واقع شده باشه. چون بعضی از لبه‌های مربوط به آجرها را تشخیص نداده است.

ج) پارامترهای تابع Sobel در کتابخانه OpenCV به شرح زیر است:

1- src : عکس ورودی که قرار است لبه های آن به کمک Sobel تشخیص داده شود.

2- ddepth : این پارامتر دیتا تایپ خروجی را مشخص میکند. اگر مقدار آن 1- باشد، به معنای این است که نوع دیتای خروجی همانند نوع دیتای ورودی باشد. بعضی از مقادیری که این پارامتر میتواند بگیرد، به شرح زیر است:

Cv2.CV\_8U : 8-bit unsigned integer

Cv2.CV\_8S : 8-bit signed integer

Cv2.CV\_64F : 64-bit floating point

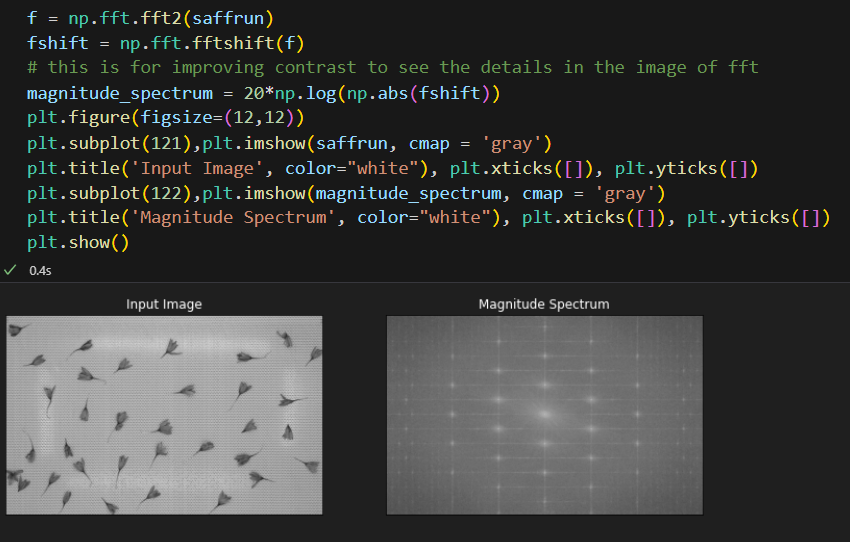
3- dx , dy : این پارامترها مرتبه مشتق گیری در هر جهت را مشخص میکند. مقدار آن ها برای فرمولی که در اسلاید درس بود، برابر با dx=1, dy=1 می‌باشد. اگر داشته باشیم: dx = 2, dy = 2 به عملیات لاپلاسین تبدیل می‌شود.

4- ksize : سایز کرنل سوبل را مشخص میکند. معمولا مقدار آن برابر با 3 است. هرچه سایز کرنل بزرگتر باشد، تصویرحاصل smooth تر است.

سوال 2

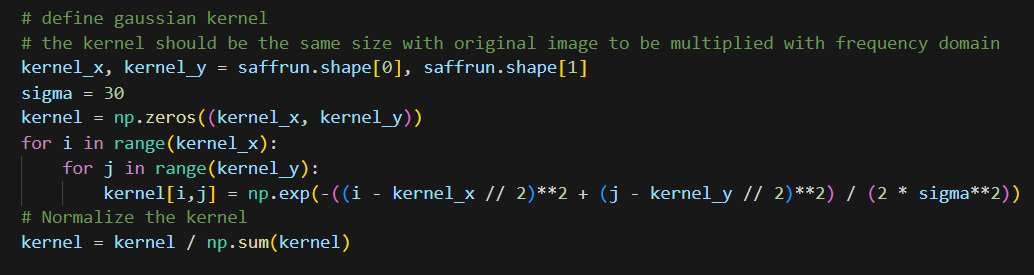
الف) ابتدا عکس را به صورت grayscale خواندم.

سپس با استفاده از پکیج fft در نامپای و تابع fft2 تبدیل فوریه تصویر را بدست آوردم.

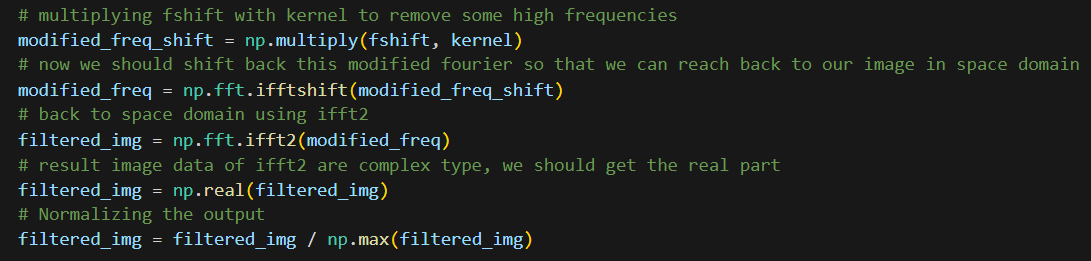


برای مشاهده بهتر، نقطه بالا چپ را به مرکز شیفت میدهیم. این کار به کمک تابع np.fft.fftshift قابل انجام است. از آنجا که نقطه مرکز نسبت به نقاط دیگر اندازه خیلی بیشتری دارد، باعث میشود روشنایی نقاط دیگر دیده نشود، پس کاری که باید بکنیم افزایش کنتراست است که با یک تابع لگاریتمی در اینجا قابل انجام است. نکته‌ای که در تصویر حوزه فرکانس عکس ما هست اینه که نقاط زیادی از آن به رنگ خاکستری هستند و این به دلیل وجود نویز است. همچنین شاهد روشنایی بعضی فرکانسهای بالا هستیم که برای یک تصویر نرمال غیرمعمول است.

کاری که کردم، اومدم از فیلتر گاوسی استفاده کردم که بیاد فرکانس‌های بالا را تضعیف کنه و فرکانسهای پایین رو نگه داره. هرچند بخشی از دیتای تصویر اصلی هم به همراه نویز از بین میرود ولی مقدار نویزی که از بین میرود به مراتب بیشتر است.



کرنل گاوسی رو هم سایز با تصویر انتخاب کردم، چون در فیلتر مربوط به حوزه فرکانس طبق چیزی که در اسلاید ها بود ، دو تصویر در هم ضرب می‌شدند و کانولوشن نداشتیم. کاری که فیلتر گاوسی برای انجام میدهد، این است که به پیکسل مرکزی حوزه فرکانس ضریب بیشتری میدهد و هرچه از مرکز دورتر میشود، این ضریب به صفر نزدیکتر می‌شود.



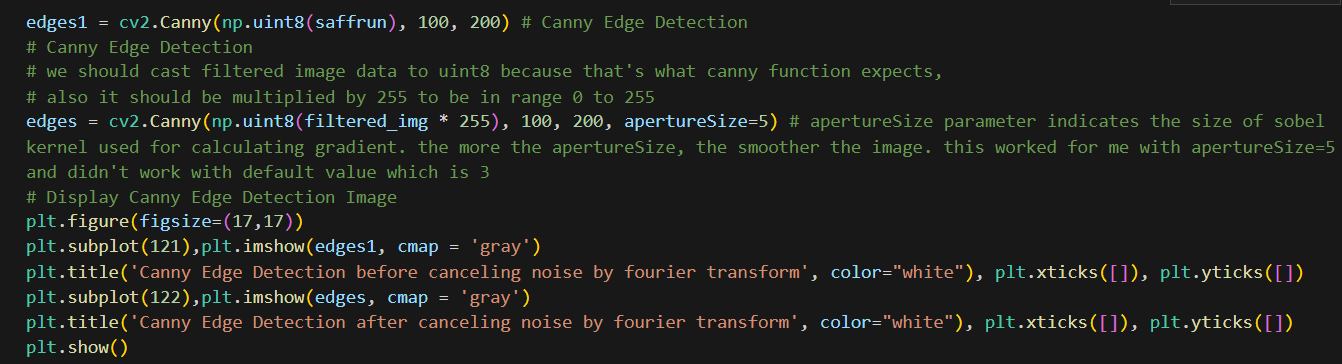
حال باید تصویر حوزه فرکانس که شیفت داده بودیم را در کرنل بدست آمده ضرب کنیم. سپس تصویر حاصل را شیفت معکوس داده تا بتوانیم تبدیل فوریه معکوس را روی آن اجرا کنیم. در آخر تصویری که بدست میاید حاوی دیتای مختلط است پس بخش حقیقی آن را میگیریم و در آخر آن را نرمالایز میکنیم.

چند نکته: در اول اومده بودم تصویری که با تابع لگاریتمی کنتراستش رو افزایش داده بودم رو به عنوان ورودی تابع multiply با کرنل ضرب کردم که نتیجه درست نبود. پس افزایش کنتراست صرفا برای نمایش هست و نباید تو قدم های بعدی بیاد

مشکل دیگه ای که پیش اومد این بود که سعی میکردم مثل سوال یک بیام یه کرنل 15 \* 15 بسازم و با تصویر کانوالو کنم که اینم اصلا نتیجه درستی نمیداد. پس ضرب یک صفحه با تصویر درست است.



میبینیم که تصویر حاصل به مقدار زیادی نویز آن حذف شده، هرچند گل‌های زعفران هم تار شده اند.



حال لبه یاب canny را روی عکسی که نویز داشت و عکسی که نویزش حذف شد، میزنیم.

توضیح پارامترهای تابع canny :

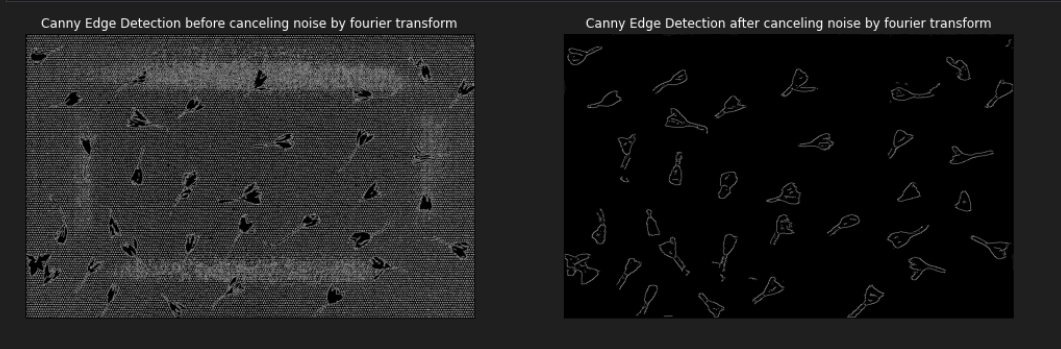
Image : تصویری که میخواهیم لبه یابی را روی آن اجرا کنیم.

Treshold1 : همان مرز اولیه در آستانه گذاری دو مرحله ای است که در اسلایدها خواندیم.

Treshold2: همان مرز دوم در آستانه گذاری دو مرحله ای که در اسلایدها خواندیم.

apertureSize : از آنجا که یکی از مراحل لبه یابی canny استفاده از کرنل سوبل برای بدست آوردن گرادیان است، این پارامتر تعیین میکند اندازه کرنل سوبلی که استفاده میشود چند باشد و این عدد باید فرد باشد وگرنه خطا میدهد. هرچه این عدد بزرگتر باشد، تصویر smooth تر خواهد شد.

چند نکته درباره تابع canny در cv2 : عکس فیلترشده‌ای که بدست آوردیم و معکوس فوریه به ما داد، مقادیر پیکسلها در بازه 0 تا 1 است ولی تابع کنی مقادیر 0 تا 255 رو ورودی میخواهد پس ابتدا در 255 ضرب میکنیم. همچنین تابع کنی دیتایی با تایپ uint8 را ورودی میگیرد پس باید تایپ دیتا را به این تایپ تبدیل کنیم وگرنه ارور میدهد.



در خروجی نهایی میبینیم که عکسی که حاوی نویز بود، مقادیر روشن زیادی پس از لبه یابی دارد که مطلوب ما نیست و مربوط به نویز است ولی در تصویر راست گل های زعفران مشخص هستند.

د) امتیازی: برای تشخیص نقطه‌ای از ساقه که باید برش اتفاق بیفتد میتوان کار زیر را انجام داد:

یک پنجره با سایز مشخص در نظر میگیریم و روی کل صفحه می‌غلطانیم. توجه کنید که اینجا صفحه ما در واقع صفحه ای است که دیتای هر پیکسل برابر با اندازه جهت گرادیان آن پیکسل در عکس اصلی است. حال هرجا که پنجره ما قرار گرفت، در آن ناحیه هیستوگرام را بدست میاریم. فرض میکنیم نقاطی که لبه تشخیص داده نشدند و اندازه گرادیان خیلی کم بود، در صفحه مربوط به جهت گرادیان مقدار -1 دارند.(به هر حال باید اندازه بزرگی داشته باشد تا پارامتر جهت را برای آن در نظر بگیریم) یا اینکه میتوانیم برای هر پیکسل به پیکسل متناظر در ماتریس اندازه ها برویم و اگر اندازه آن در حد یک لبه نبود، آن را در هیستوگرام دخیل نکنیم)

نکته دیگر انتخاب سایز پنجره است. سایز پنجره باید از اندازه ساقه زعفران کوچکتر باشد پس باید اندازه تقریبی ساقه زعفران را بدانیم تا بتوانیم کرنل خوبی انتخاب کنیم.

برای هر قسمت که هیستوگرام گرفتیم، چهار حالت داریم:

1. مجموع تعداد مقدارهای هیستوگرام نزدیک صفر است. این بدین معنی است که اصلا روی زعفران قرار نداریم و در نتیجه لبه ای نداریم برای همین در محاسبه هیستوگرام، آن پیکسلها دخیل نشدند.
2. یک پیک قابل توجه روی یکی از مقادیر داریم و مقادیر دیگر تقریبا صفر هستند. در این صورت روی ساقه قرار داریم ولی نه روی قسمتی از ساقه که به زعفران متصل است.
3. یک پیک قابل توجه روی یکی از مقادیر داریم، و بقیه مقادیر به طور پخش آمده‌اند. اینجا میتوان نتیجه گرفت که در نقطه اتصال ساقه و زعفران هستیم و باید برش بزنیم.
4. مقادیر هیستوگرام در یک بازه پخش هستند. در این صورت روی خود گل زعفران قرار داریم.

سوال 3

الف) فیلترهای پایین گذر یا low-pass filters به فرکانس‌های پایین اجازه عبور داده یعنی آنها را از تصویر فرکانس حاصل از تبدیل فوریه حذف نمیکنند ولی فرکانس‌های بالا را حذف کرده و عبور نمی‌دهند. محل کاربرد این نوع فیلترها در noise cancellation یا کاهش سایز عکس می‌باشد. زیرا فرکانس‌های بالا برای یک عکس نرمال غیرمعمول است و بیشتر این فرکانس‌ها را نویز تشکیل می‌دهد و با حذف این فرکانس‌ها قسمت بیشتری از نویز و قسمت کمتری از عکس‌ ما از بین می‌رود.

فیلترهای بالاگذر یا high-pass filters برعکس قبلی کار می‌کند. فرکانس‌های بالا را عبور داده و فرکانس‌های پایین را حذف میکند و باعث می‌شود لبه‌ها و جزئیات تقویت شوند. با توجه به عملکرد آن، درتشخیص لبه، تیز کردن عکس یا sharpening و استخراج ویژگی از عکس بدرد می‌خورد.

ب) حاصل اعمال فیلتر بالاگذر است زیرا مشاهده می‌کنیم لبه‌ها، حتی لبه‌های جزئی که ضعیف بودند، تقویت شده‌اند.

ج) نویز جمع شونده بدین صورت است که اگر یک پیکسل مقدار روشنایی 100 دارد، سنسور ما مقدار روشنایی 103 ثبت میکند، یعنی مقداری خطا دارد و این برای بقیه پیکسل ها هم تکرار می‌شود. در مقابل نویز ضرب شونده، بدین صورت است که مقدار واقعی روشنایی یک پیکسل را در یک مقدار رندوم ضرب میکند. یک تفاوت دیگر این دو نویز این است که نویز جمع شونده به عکس یا سیگنال وابسته نیست در حالیکه نویز ضرب شونده وابسته به سیگنال است، زیرا در نویز جمع شونده مقداری که به مقدار واقعی اضافه میشود رندوم است و به مقدار خود پیکسل وابسته نیست ولی در نویز ضرب شونده، هرچه مقدار یک پیکسل بزرگتر باشد، پس از اعمال نویز، افزایش بیشتری نسبت به پیکسل های با مقدار کوچکتر دارد. به همین دلیل حذف نویز جمع شونده از نویز ضرب شونده راحت تر است. برای مثال میتوان از یک کرنل میانگین‌گیر استفاده کرد، یعنی مقدار هر پیکسل برابر می‌شود با میانگین مقادیر پیکسل‌های اطراف. راه دیگر استفاده از فیلتر گاوسی است که آن هم نوعی میانگین گیری است، بدین صورت که عکس را با یک کرنل گاوسی کانوالو می‌کنیم.

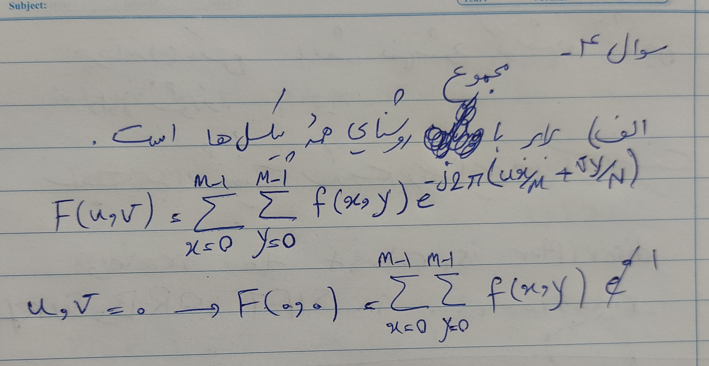
د) نویز نمک و فلفل زمانی پیش می‌آید که بعضی از سنسورها خراب باشند، مثلا فقط مقدار صفر برگردانند یا یک مقدار دیگر. این نوع نویز جمع‌شونده نیست و نمی‌توان برای آن از فیلتر میانگین گیر استفاده کرد، برای مثال اگر مقدار یک پیکسل 100 باشد و در واقع مقدار درست آن باشد و فرض کنیم که در پنجره اطراف آن پیکسلی باشد که مقدار آن صفر ثبت شده، در این صورت این مقدار صفر در میانگین تاثیر زیادی گذاشته و باعث میشود، مقدار درستی برای این پیکسل در نظر نگیریم.

راهکار حل آن استفاده از فیلتر میانه است که نوعی فیلتر مرتبه‌ای است. در فیلتر میانه داده‌های پرت تاثیر داده نمیشوند.

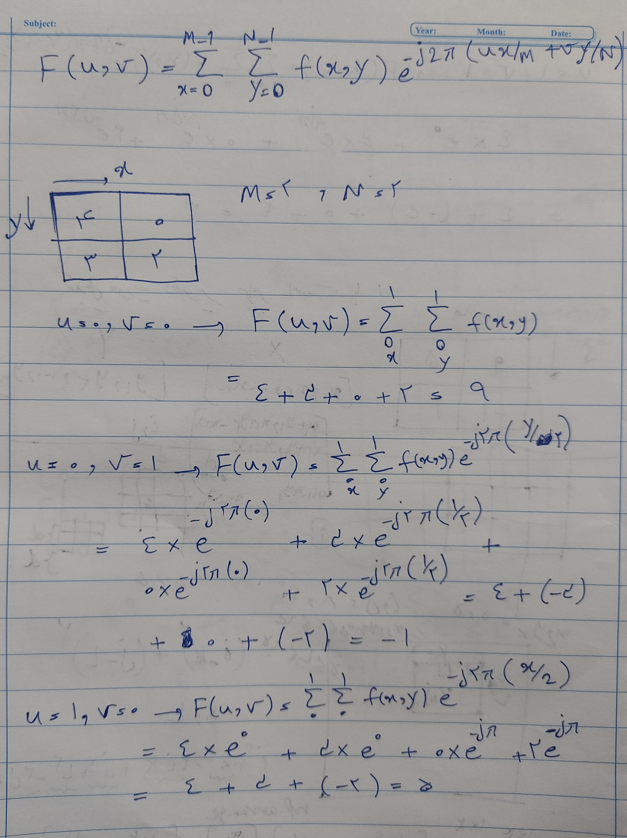
منظور از فیلتر میانه این است که مقدار هر پیکسل را برابر با میانه‌ی مقدار پیکسل های اطراف آن قرار دهیم.

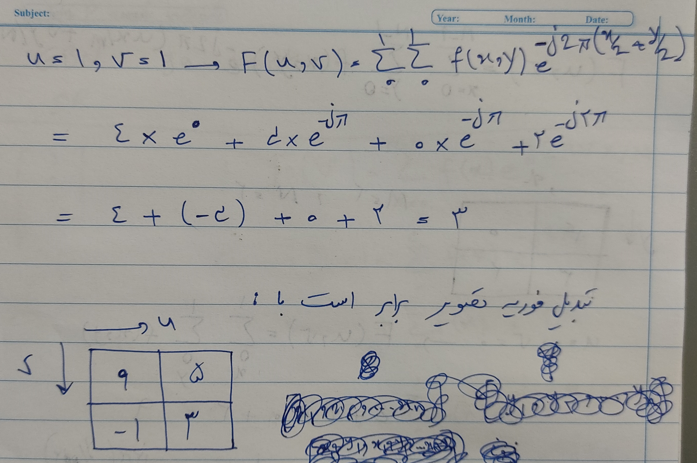
سوال 4-

الف)



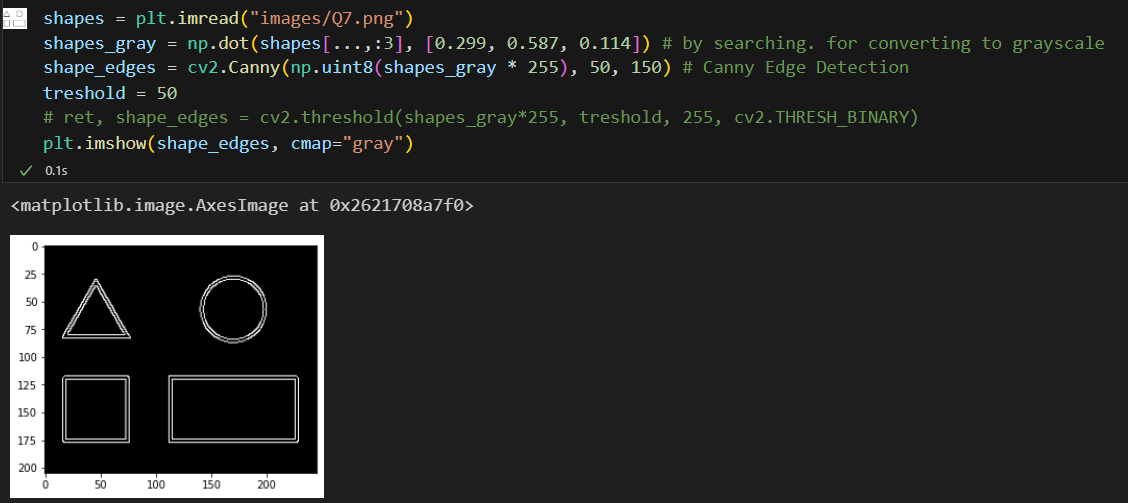
ب)





سوال 5-

الف) ابتدا تصویر را میخوانیم و grayscale میکنیم.

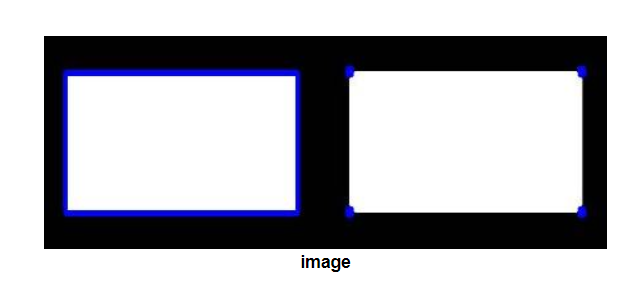


ب) تعریف ساده: تابع findContours به عنوان ورودی یک تصویر باینری گرفته و به عنوان خروجی لیستی از مجموعه نقاط میدهد که هر کدام از این مجموعه نقاط، نقاط مرزی یک شکل هندسی هستند.

برای اینکه تصویری که به این تابع میدهیم، باینری باشد، لازم است، روی عکس پردازشی انجام دهیم، بدین صورت که نقاطی که از یک حدی بالاتر هستند را 1 در نظر گرفته و آنهایی که پایین تر از آن مرز هستند، صفر در نظر بگیریم. می‌توان از تابع cv2.treshhold استفاده کرد ولی یک راه دیگر آن خروجی حاصل از لبه‌یابی canny است که من همین کار را کردم. خروجی حاصل از canny دارای مقادیر صفر یا 255 می‌باشد، با تقسیم این مقادیر به 255 ، به دیتای باینری صفر و یک می‌رسیم.

پارامترهای تابع findContours :

Img : عکسی که میخواهیم نقاط مرزی آن را بدست آوریم.

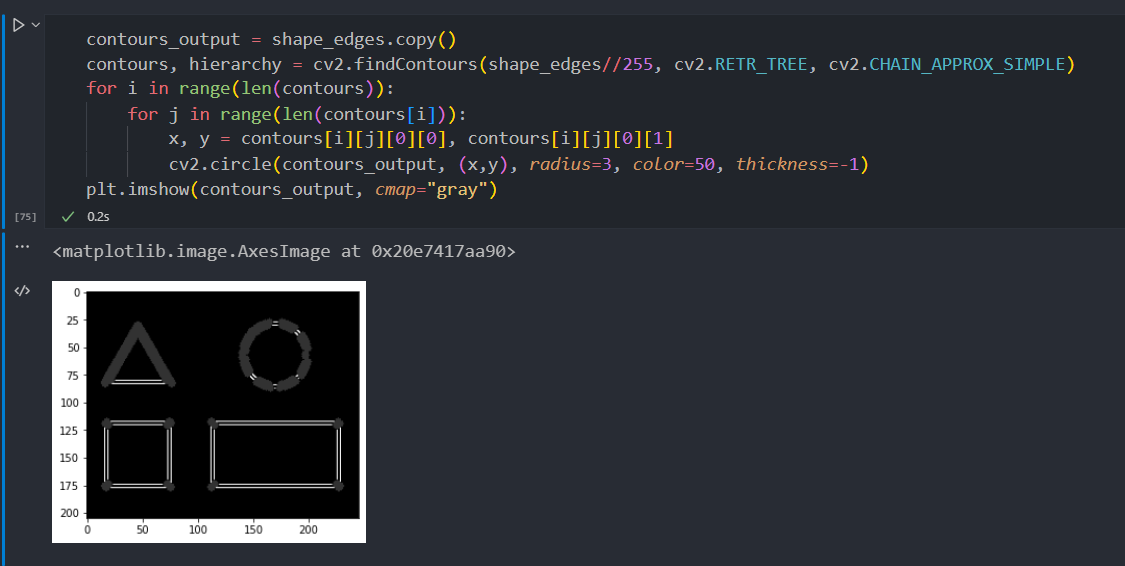
Approximation method : گفتیم که این تابع، نقاطی که روی یک شکل هندسی هستند را خروجی میدهد، ولی آیا همه این نقاط را خروجی میدهد؟ این توسط این پارامتر تعیین میشود. برای این پارامتر، متود هایی از پیش تعریف شده توسط خود opencv قرار داده شده است. برای مثال اگر مقدار cv2.CHAIN\_APPROX\_NONE را برای آن قرار دهیم ، همه نقاط مرزی اشکال را برمیگرداند. اما آیا نیاز به این کار است؟ برای مثال در اشکال ساده مثل خط صاف یا مربع و مستطیل تعداد محدودی نقطه کافی است. برای یک خط صاف دو نقطه کافی است، یا برای مربع و مستطیل چهار نقطه کافی است، بدین صورت مقدار حافظه مصرفی خیلی کم میشود، برای این کار میتوان مقدار این پارامتر را برابر با cv2. CHAIN\_APPROX\_SIMPLE قرار داد. 

این عکس رو از داک رسمی OpenCV برداشتم. سمت چپ مربوط به cv2.CHAIN\_APPROX\_NONE است و سمت راست مربوط به متود cv2.CHAIN\_APPROX\_SIMPLE میباشد.

چند نکته:

از تابع cv2.treshold استفاده نکردم چون میومد اون خطوط نمودار matplotlib رو توی عکس در نظر میگرفت و مقدارشون رو مشکی میکرد بعد خود اون رو یه شکل مربع حساب میکرد ☺

از تابع canny استفاده کردم ولی لبه یابش میومد دو تا مرز برای شکلها در نظر میگرفت همانطور که انتظار میرفت و باعث میشد تابع findContours تعداد 16 تا شکل تشخیص دهد که کار را سخت میکرد پس پارامتر retrieval mode رو توی تابع findContours تغییر دادم تا فقط اشکال بیرونی رو بگیره. پایینتر درباره این پارامتر صحبت کردم.

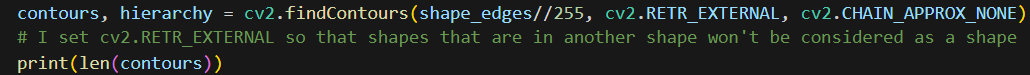


همانطور که مشخص است برای مستطیل و مربع تنها چهار نقطه خروجی داده است ولی برای دایره، این نقاط خیلی بیشتر است. در مورد م باید سه تا خروجی میداد ولی حدس من اینه که اضلاع مثلث توسط canny به اندازه کافی صاف نیستن و نویز دارن، برای همین یه خط صاف در نظر نگرفته اونا رو، پس اینجا میایم از یه تابعی به اسم approxPolyDP استفاده میکنیم که میاد به طور تخمینی و با یه دقتی که خودمون بهش میدیم، کمترین نقاط لازم برای تشخیص اون شکل رو میده و اینجوری تاثیر نویز کم میشه.

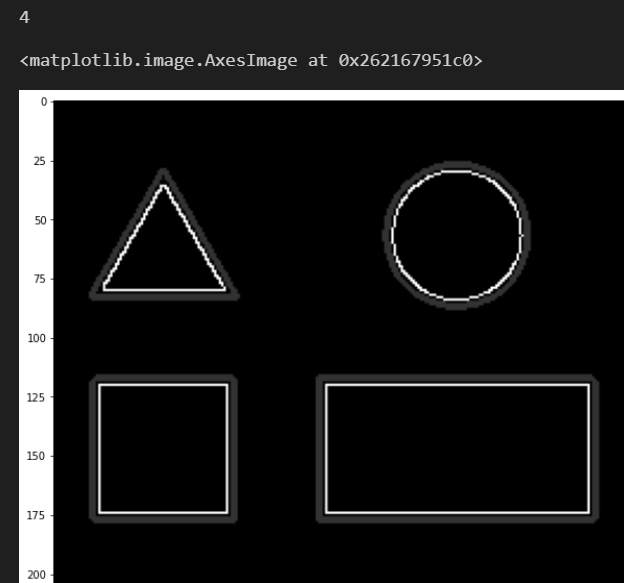
Contour retrieval mode : وقتی از تابع findContours استفاده میکنیم، این تابع یک درخت از کنتورها تشکیل میدهد. هر کنتور میتواند یک parent و صفر یا چند فرزند داشته باشد. اگر کنتوری در یک کنتور دیگر محدود شده باشد، فرزند آن محسوب میشود. حال با استفاده از این پارامتر میتوان مشخص کرد از این درخت کدام کنتورها را به ما بدهد.

اگر مقدار cv2.RETR\_EXTERNAL را بدهیم، فقط کنتورهایی که هیچ parent ی ندارند ، برگردانده میشوند. اگر مقدار cv2.RETR\_LIST بدهیم ، همه کنتورها را بدون توجه به درخت برمیگرداند.

در اینجا ما از EXTERNAL استفاده میکنیم زیرا تصویر حاصل از CANNY به صورت دو شکل تو در تو است.

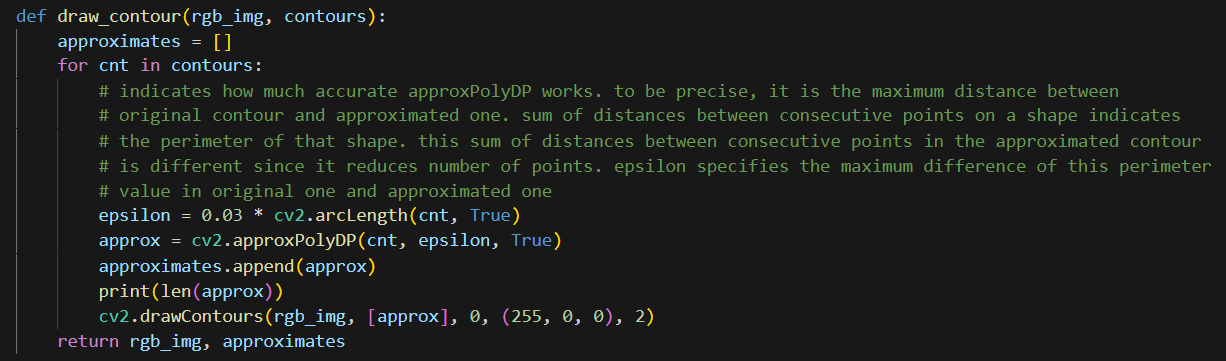


خروجی کد بالا یک لیست با اندازه چهار خواهد بود و فقط اشکال بیرونی را در نظر میگیرد.

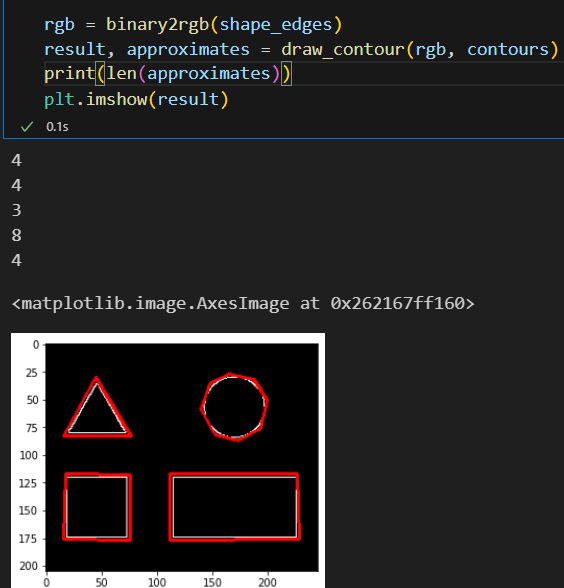


خطوط خاکستری نقاط تشخیص داده شده توسط تابع findContours میباشد.

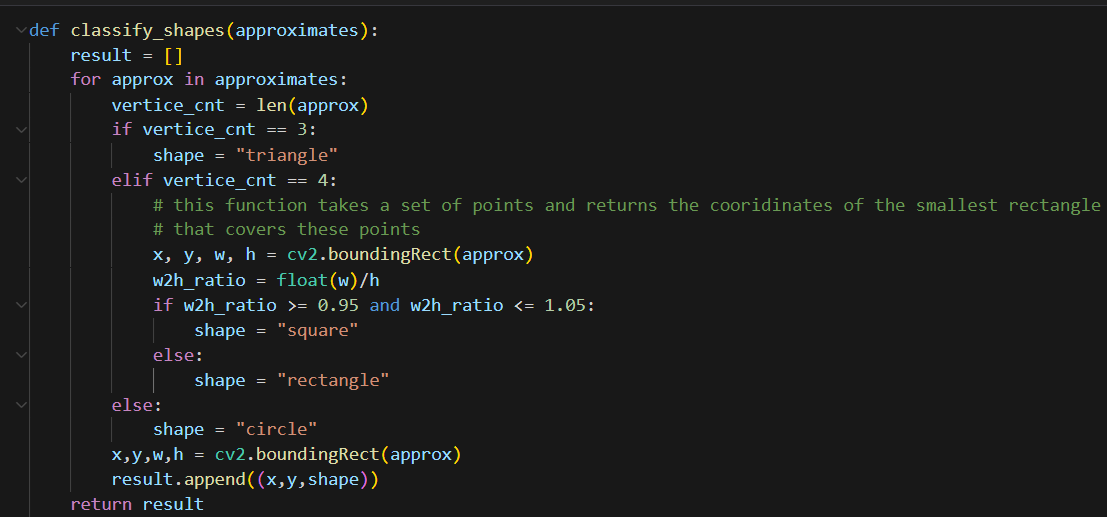
ج) حال همانطور که گفتیم، باید از تابع approxPolyDP استفاده کنیم تا حداقل تعداد نقاط ممکن بر اساس دقتی که به آن میدهیم تعیین شود، ولی چرا میخواهیم این کار را کنیم؟ چون بتوانیم بر اساس آن تعداد نقاط تشخیص دهیم شکل ما دایره است یا مربع یا ...



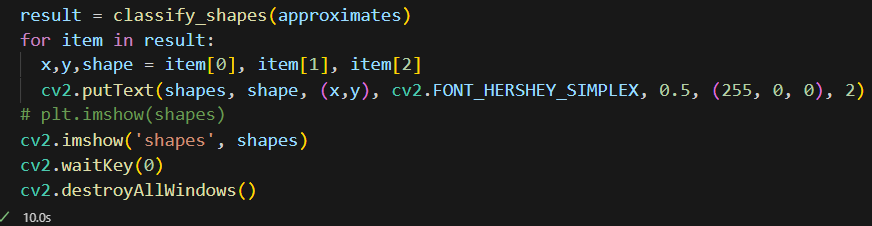
این تابع یک عکس و کنتورهای تشخیص داده شده را میگیرد و برای هر کنتور یا شکل کمترین تعداد نقاط را با استفاده از تابع approxPolyDP خروجی میدهد. پارامتر اول این تابع ، کنتور یا شکل ما است و پارامتر دوم اپسیلون یا دقت است که در کامنتی در تصویر به انگلیسی نوشتم که این پارامتر در واقع چه چیزی را نشان میدهد. هرچه این پارامتر کمتر باشد، دقت بیشتر خواهد بود و تعداد نقاط برگشتی نیز بیشتر خواهد بود. پس باید این پارامتر را به گونه‌ای تنظیم کرد که برای مربع و مستطیل عدد 4 ، برای مثلث عدد 3 و برای دایره عددی بزرگتر از اینها خروجی دهد تا بتوان آن ها را تشخیص داد. البته خروجی این تابع یک عدد نیست بلکه مجموعه آن نقاط است که ما منظورمان تعداد اینها است. یک common practice که در نت برای این پارامتر پیدا کردم استفاده از محیط شکلی است که تابع findContours برگردانده زیرا پارامتر اپسیلون ماکزیمم اختلاف محیط شکل تخمین زده شده و محیط شکل اصلی که تابع findContours برگردانده، میباشد. تابع arcLength همین کار میکند، یک کنتور که شامل مجموعه ای از نقاط است گرفته و مجموع فواصل نقاط پشت سر هم که همان محیط شکل را تشکیل میدهد بدست می‌آورد. پارامتر دوم که true قرار داده شده است یعنی یک شکل هندسی بسته در نظر بگیرد و نقطه آخر را به نقطه اول وصل کند. حال ضریبی از این مقدار را برابر با اپسیلون قرار میدهیم که من این ضریب را با آزمون و خطا بدست آوردم به طوری که با تکرار این کار به حالتی رسیدم که برای دایره تعداد نقاط برابر با 8 ، برای مثلث 3 و برای مستطیل و مربع برابر با 4 شد. حال عکس حاصل را به همراه اشکال تخمینی برگرداندم.



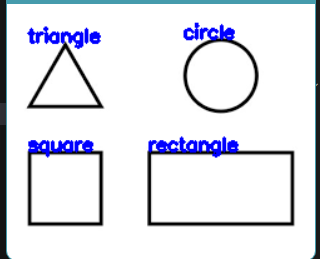
همانطور که میبینید 4 تا شکل خروجی داده که مقادیر 4و4و3و8 دارند. اینها نقاط گوشه‌ی شکلها هستند. برای دایره هشت تا گوشه در نظر گرفته و تابع drawContours اومده این 8 تا رو به هم وصل کرده، برای بقیه اشکال هم همینطور. پس خروجی تابع approxPolyDP در واقع نقاط گوشه اشکال است.



حال نوبت به مرحله نهایی یعنی تشخیص اشکال میرسد. در اینجا اشکال تخمینی را ورودی میگیریم. سپس با توجه به تعداد رئوس، شکل را تشخیص میدهیم. اگر تعداد رئوس 3 تا باشد، مثلث است و اگر بیش از 4 تا باشد، دایره است و اگر 4 تا باشد یا مستطیل است یا مربع پس باید این را تشخیص دهیم. از تابع boundingRect استفاده میکنیم. این تابع به عنوان ورودی یک مجموعه از نقاط گرفته و یک مستطیل با حداقل طول و عرض به طوری که همه این نقاط را پوشش دهد، می‌کشد، سپس مختصات نقطه بالا چپ مستطیل و طول و عرض آن را خروجی میدهد. حال اگر این طول و عرض تقریبا برابر باشد، مربع داریم و اگر اختلاف داشته باشند، مستطیل داریم. اینجا نسبت طول به عرض رو گرفتم و برای یه خطای 5 درصد هم گذاشتم. در آخر دوباره از تابع boundingRect استفاده کردم و مکان حدودی هر شکل را بدست بیاورم و متنی را که برای آن شکل تشخیص دادم هم به همراه آن مختصات به لیست result اضافه کردم و برگرداندم.



سپس روی تصویر آن نوشته را به کمک تابع putText در cv2 قرار دادم. نتیجه به صورت زیر شد:



د) امتیازی: بعضی از ویژگی هایی که میتوانیم از آنها برای کلاس بندی اشکال استفاده کنیم به شرح زیر است:

1- تعداد اضلاع: به کمک تابع findContours و approxPolyDP میتوان به تعداد گوشه ها و در نتیجه تعداد اضلاع یک شکل رسید. برای تعداد اضلاع کم مثل 3 یا 4 یا 5 تا میتوان شکل را به مثلث، چهارضلعی یا پنج ضلعی کاهش داد و سپس با کمک ویژگیهای دیگر به طور دقیقتر نوع شکل را حدس زد.

2- نسبت محیط شکل به مساحت شکل: به کمک توابع cv2.arcLength و cv2.contourArea میتوان محیط و مساحت یک کانتور را بدست آورد. از نسبت این محیط به مساحت میتوان ویژگیهایی استخراج کرد. فرمول زیر را برای فشردگی داریم:

Compactness = perimeter\*\*2 / (4 \* np.pi \* area)

مقدار این فرمول برای دایره عدد 1 است و برای دیگر اشکال کمتر از 1 است. برای مثال مربع یا مستطیل فشردگی نزدیک به یک دارند در حالیکه مثلث فشردگی کمتری از اینها دارد.

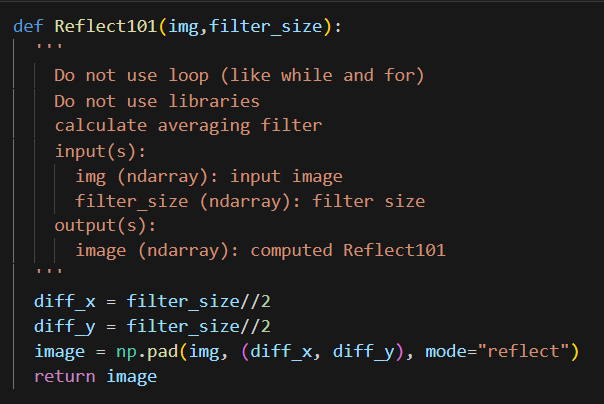
3- وضعیت توازی و برابری اضلاع: پس از اینکه با استفاده از approxPolyDP نقاط گوشه و در نتیجه اضلاع را پیدا کردیم ، میتوانیم فرمول خط هر کدام را به کمک الگوریتم hough محاسبه کنیم و وضعیت توازی بین اضلاع را بسنجیم و همچنین میتوان به کمک الگوریتم LSD پاره خط مربوط به آن ضلع را بدست آورد و به کمک آن به اندازه آن ضلع رسید و برابری اضلاع را بررسی کرد. برای مثال اگر یک چهارضلعی داشته باشیم، و دو جفت ضلع موازی داشته و دو ضلع موازی آن هم‌اندازه هم باشند، یک متوازی الاضلاع را قطعا خواهیم داشت و به کمک ویژگیهای دیگر میتوان بطور دقیقتر تعیین کرد که این متوازی الاضلاع، مربع یا مستطیل یا ... هم هست یا نه.

4-زاویه بین اضلاع: وقتی با Hough خطوط اضلاع را یافتیم، با استفاده از فرمولهایی که در هندسه تحلیلی داشتیم میتوانیم زاویه بین اضلاع را بیابیم تا به دسته بندی ما بیشتر کمک کند.

با استفاده از ویژگی های بالا و یک مدل که از آنها استفاده کند، میتوان خیلی از اشکال معمول را تشخیص داد.

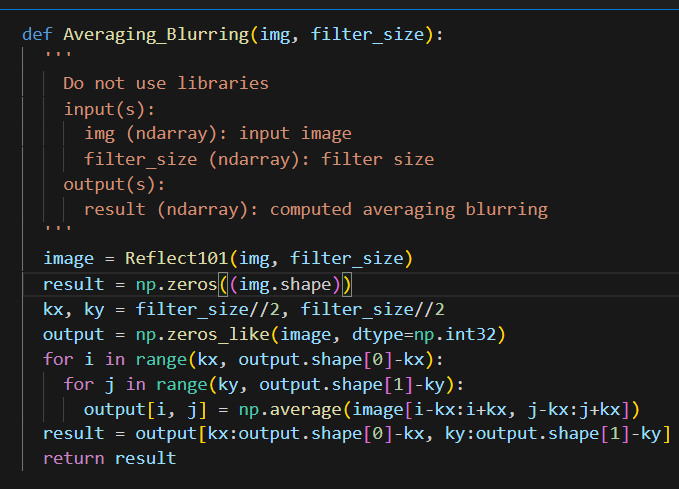
سوال 6

پیاده‌سازی تابع Reflect101 :



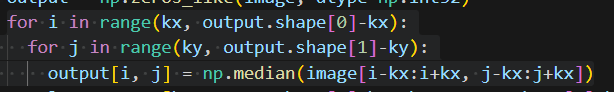
از تابع np.pad استفاده کردم و mode رو برابر با reflect قرار دادم، سایز پدینگ رو هم نصف سایز کرنل میگذاریم.

پیاده‌سازی تابع AverageBlurring :



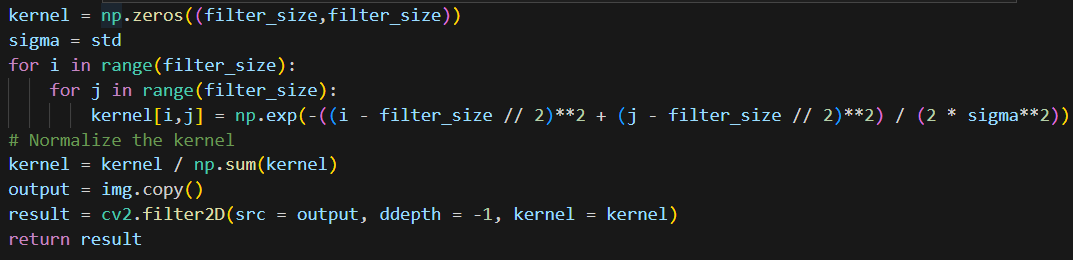
ابتدا پدینگ را روی تصویر اعمال کرده و سپس روی پیکسلهای تصویر حلقه زده و برای میانگین گیری از آن پنجره از تابع np.average استفاده میکنیم. نکته خاص دیگری ندارد.

پیاده‌سازی تابع MedianBlurring :



همانند تابع قبلی، فقط به جای np.average از np.median استفاده میکنیم.

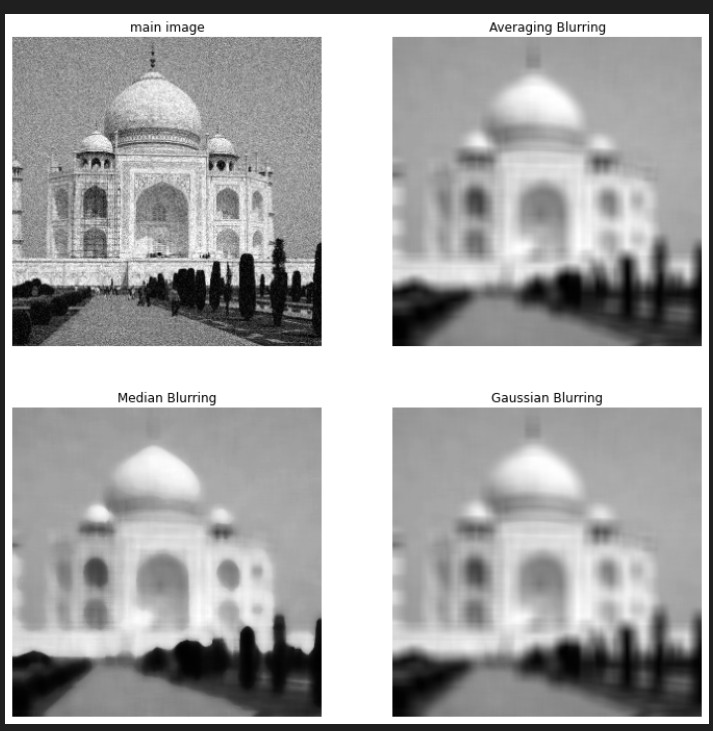
پیاده‌سازی تابع GaussianBlurring :



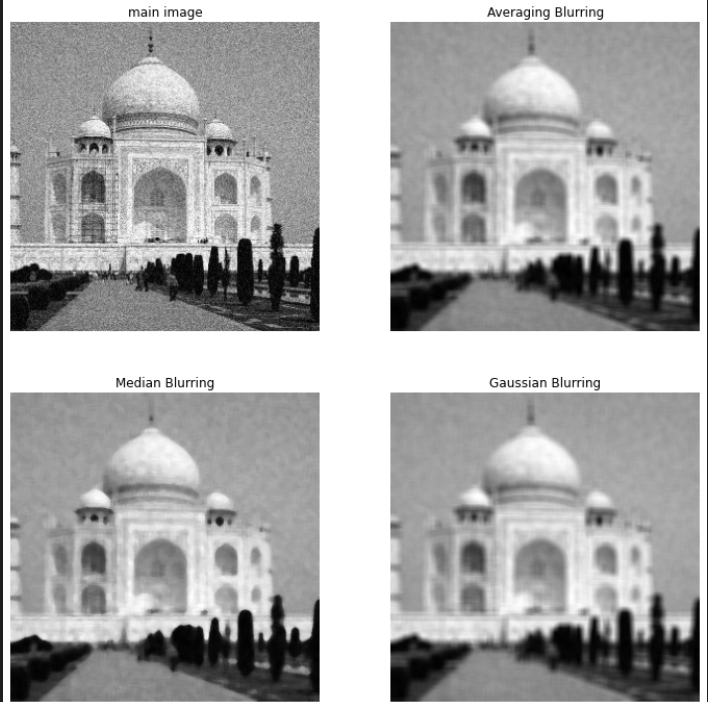
مشابه سوال یک تمرین، ابتدا کرنل گاوسی را ساخته و سپس با تصویر کانوالو میکنیم که اینجا برای کانولوشن از تابع cv2.filter2D استفاده شده است.

**تاثیر سایز کرنل بر روی خروجی:**

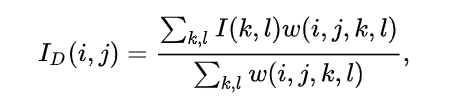
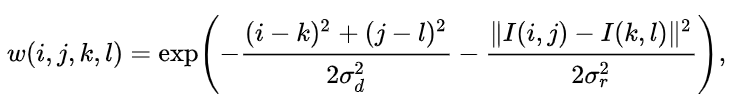
هرچه سایز کرنل بزرگتر شود، تصویر تارتر میشود، نویز بیشتری از بین میرود ولی به تبع دیتای اصلی هم بیشتر از حالت کرنل با سایز کوچکتر خراب میشود. هر چه کرنل را کوچک کنیم تصویر از تار بودن خارج شده و به نویز بیشتری میرسد. در تصویر زیر سایز کرنل برابر با 15 است.



و در تصویر زیر سایز کرنل برابر با 7 است.



ب) می‌خواهیم فیلتر bilateral را پیاده‌سازی کنیم. ابتدا باید ببینیم فرمول آن و طرز کار آن به چه شکل است.



فرمول رو از ویکیپدیا برداشتم. در اینجا مشاهده میکنید که تفاوتی که با فیلترهای قبلی دارد، این است که از یک کرنل ثابت برای فیلتر کردن استفاده نمیکند بلکه برای هر پیکسل کرنل متفاوتی را طبق این فرمول ساخته و از آن استفاده برای آن پیکسل استفاده میکند. این کرنل بر اساس فاصله نقاط اطراف و مقدار روشنایی نقاط اطراف نسبت به نقطه مرکزی بدست می‌آید. در واقع قسمت فاصله نقاط اطراف تا نقطه مرکزی را در فیلتر گاوسی داشتیم ولی به دلیل یکسان بودن سایز کرنل، این مقدار برای پیکسلهای اطراف هر پیکسل مرکزی یکسان میشد، برای همین نیازی نبود که دوباره برای پیکسلهای بعدی این کرنل محاسبه شود ولی اینجا مقدار روشنایی پیکسلهای اطراف نیز مهم است که این برای پیکسل های مرکزی متفاوت یکسان نیست. دو پارامتر سیگما داریم که یکی برای فاصله و دیگری برای روشنایی است. اگر پارامتر سیگمای روشنایی خیلی زیاد شود، کسر دوم در فرمول w صفر شده و به فیلتر گاوسی تبدیل میشد.

فرمول مربوط به ID(i,j) مقدار روشنایی جدید پیکسل i, j است که برابر است با :

ابتدا روشنایی هر پیکسل اطراف پیکسل مرکزی را در پنجره‌ای که سایزی برابر با سایز کرنل ما دارد، در وزن حاصل از تابع w برای آن نقطه k,l بدست می‌آوریم که بعدا این تابع را نیز بررسی میکنیم. سپس حاصل این ضربها برای همه پیکسلهای اطراف را با یکدیگر جمع کرده و در صورت قرار میدهیم. مقدار مخرج هم مجموع وزنهای در نظر گرفته شده برای هر نقطه k,l است.

تابع w(i, j, k, l) :

وزن هر خانه از کرنل را با توجه به اختلاف روشنایی آن خانه با پیکسل i, j و اختلاف مکانی آن با خانه i, j بدست می‌آورد.

کاربرد و دلیل استفاده از این فیلتر این است که میخواهیم نویز تصویر حذف شود ولی لبه‌های تصویر خراب نشوند و تصویر تار نشود. از آنجا که این فیلتر کرنل را بر اساس هر نقطه و برای آن پیکسل به طور جداگانه حساب میکند، این را نیز در نظر میگیرد که اگر لبه بود، به پیکسل مرکزی وزن خیلی بیشتری نسبت به اطرافش بده. لبه بودن رو هم بر اساس اختلاف سطح روشنایی و فاصله متوجه میشه.

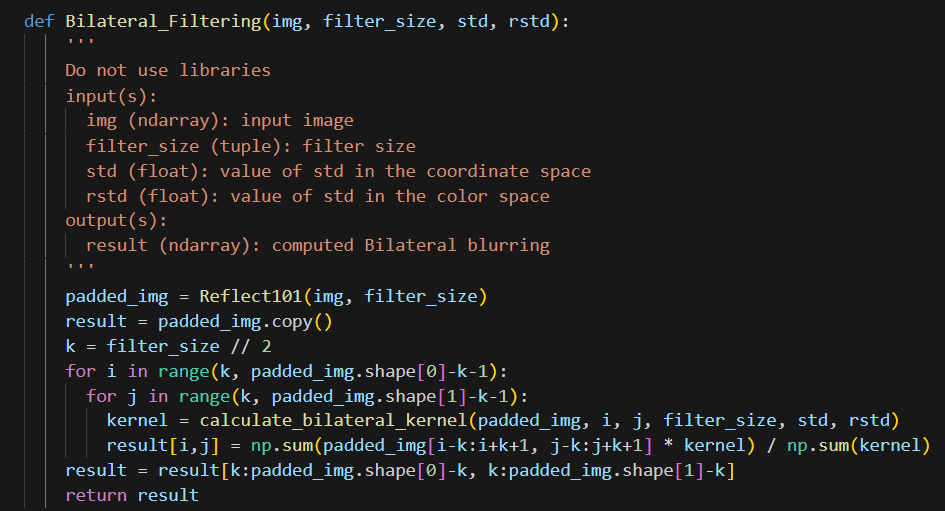
پارامترهای سیگما:

اگر هر دو سیگما به بینهایت بروند، به همان میانگین گیری تبدیل میشود.

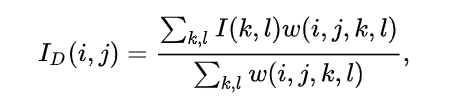
اگر هر دو خیلی کوچک در حد صفر باشند، تصویر تغییری نمیکند.

اگر انحراف معیار روشنایی خیلی زیاد شود، به همان فیلتر گاوسی میرسیم و اگر انحراف معیار مکانی خیلی زیاد شود، به یک فرمول جدید میرسیم که فقط بر اساس تفاوت روشنایی کار میکند که این رو تست کردم خیلی اتفاق خاصی نیفتاد و تصویر بهم نخورد و خوب افتاد.

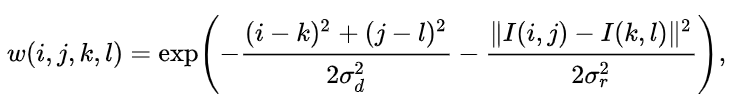
پیاده‌سازی:



همانند فیلترهای قبلی ابتدا پدینگ را اعمال کرده و سپس روی هر پیکسل تصویر حلقه زده و از اینجا به بعد با آنها فرق میکند. برای هر پیکسل تابع w را روی تک تک نقاط اطراف آن زده و کرنل وزنی نهایی برای آن پیکسل را بدست می‌آوریم. این کار توسط تابع calculate\_bilateral\_filtering انجام میشود که جلوتر توضیح میدهم. سپس با استفاده از این کرنل و خط بعدی که طبق فرمول زیر است، مقدار جدید روشنایی آن پیکسل را جایگزین میکنیم.

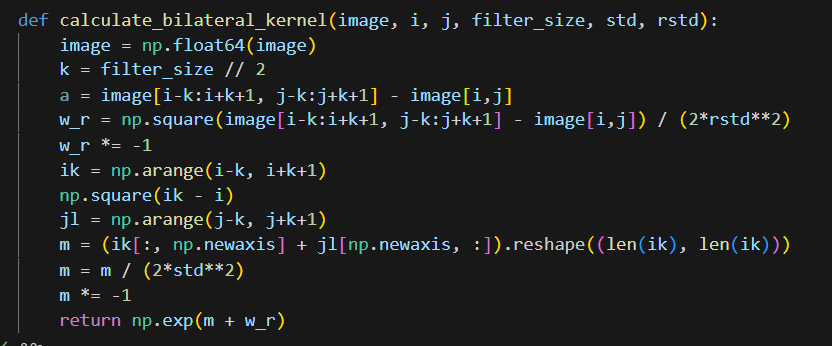


از توابع نامپای استفاده میکنیم تا سرعت بهتری بدست آوریم، چون همینطوریش هم دو تا حلقه داریم پس نباید از حلقه دیگری استفاده کنیم. از حلقه سوم که استفاده میکردم سرعت خیلی پایین میومد. توی تابع calculate\_bilateral\_filtering هم از هیچ حلقه ای استفاده نکردم و تنها از توابع نامپای استفاده کردم.



باید طبق این فرمول تابع calculate\_bilateral\_filtering را پیاده سازی کنیم.

تابع calculate\_bilateral\_filtering :

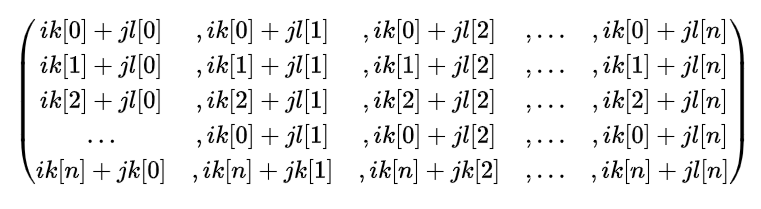


متغیر w\_r مقدار کسر دوم در فرمول است که مربوط به اختلاف سطح روشنایی است.

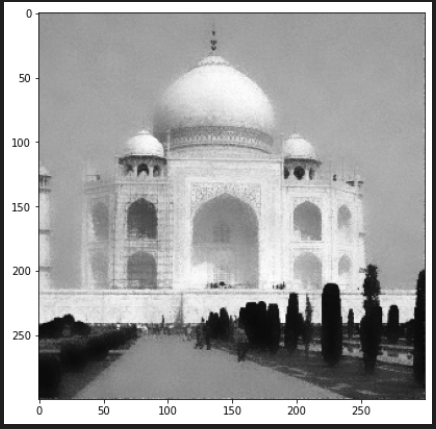
متغیر m مربوط به کسر اول است. تقریبا هر خط مشخص هست که چیکار میکنه به غیر از خط زیر:



در اینجا ما میخواهیم دو آرایه ik , jl را گرفته و ماتریسی بدست بیاوریم که مقادیر آن به صورت زیر است:

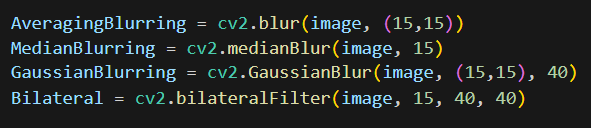


برای این کار از Chatgpt کمک گرفتم چون خیلی با توابع نامپای آشنا نبودم و این تیکه کدی بود که بهم داد و درست هم کار کرد.

خروجی نهایی: 

ج)

در اینجا از توابع آماده opncCV استفاده کردیم.



و خروجی زیر را بدست آوردیم.

