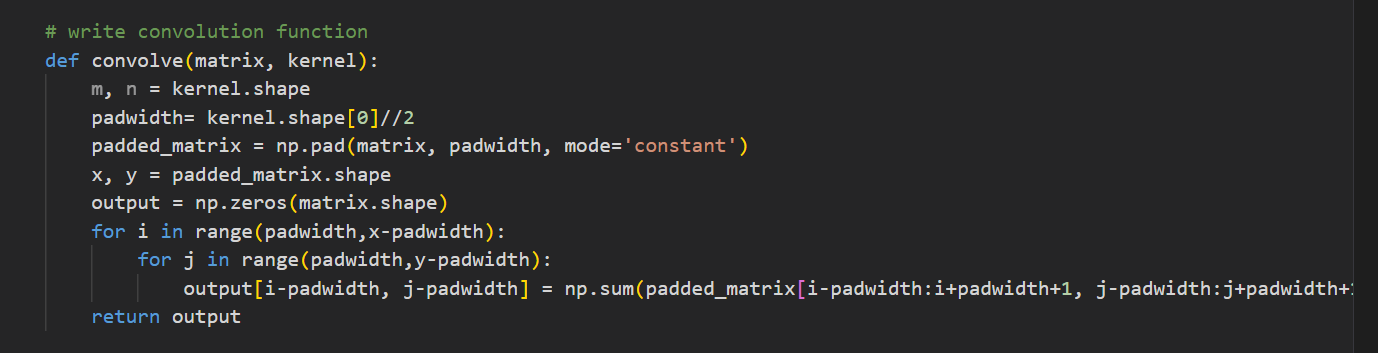
|  |
| --- |
| **Amirreza Vishteh** |
| HW2 26/1/1402  *Operation System* |

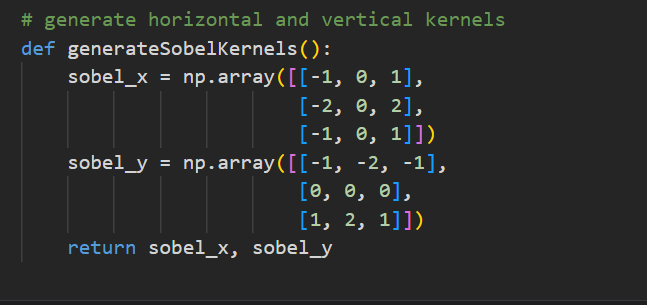
(1

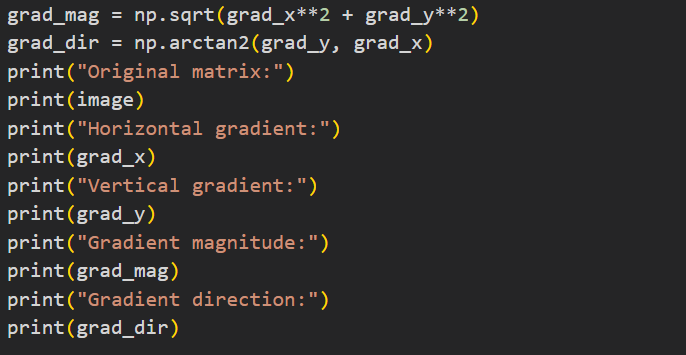
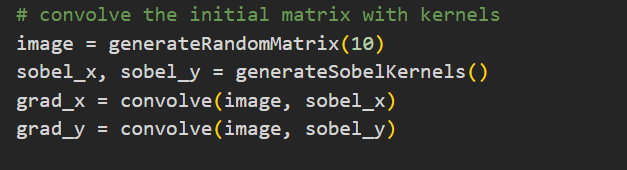
الف) برای این بخش یک ماتریس n\*n با استفاده از تابع random.randintساختیم

در بخش پیاده سازی تابع convolve مقداری برای به دست اوردن ایندکس در حلقه ها به علت داشتن padding به مششکل خوردم اما در نهایت با رسم شکل حل شد

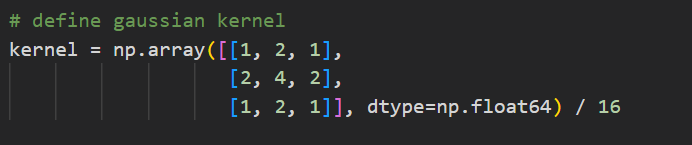
در اینجا ما ابتدا برای ماتریس padding در نظر گرفتیم که در واقع zero padding است و بعد با دو حلقه انرا پیمایش و در کرنل مد نظر ضرب کردیم

سپس کرنل های sobel را تعریف کردم و با استفاده از تابع convolve ماتریس اولیه را با ان ضرب میکنیم تا بتوانیم اندازه و جهت گرادیان را با استفاده از مشتق ها به دست بیاوریم در واقع حاصل convolve ماتریس و هر یک از sobelها یکس از مشتق های ما است

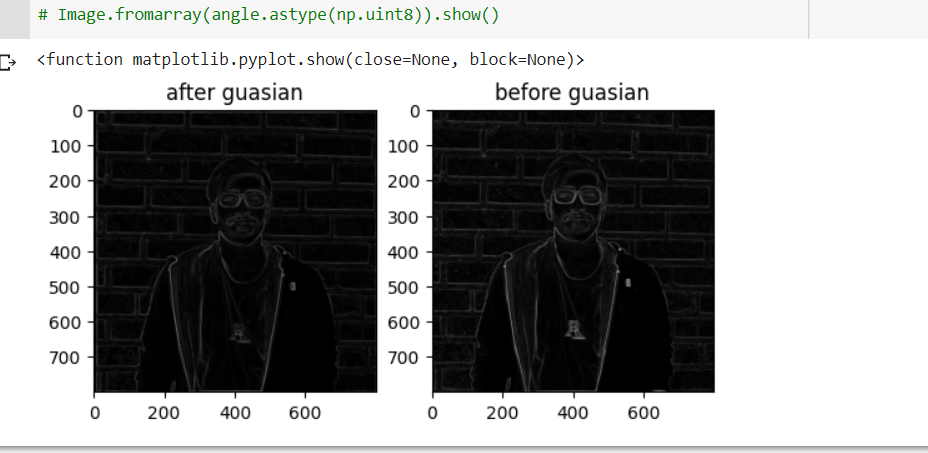




برای بخش بعد ابتدا برای کمتر کردن نویز انرا با یک فیلتر گاووسی کانوالو میکنیم و بعد با کرنل های سوبل مشتق ها را محاسبه و اندازه و جهت گرادیان را نمایش میدهیم من کرنل گاووسین را ابتدا به صورت 3\*3 تعریف کردم ولی بعد از انجام سوال 6 و زدن تابع به دست اوردن فیلتر کاووسین با اندازه های دیگر هم امتحان کردم در نهایت فیلتر25\*25 را انتخاب کردم که تصویر کمی واضح تر شود







همانطور که مشاهده میشود در عکس با فیلتر گاووسی لبه ها کم رنگ تر شده و خیلی از جا ها مثل داخل کاپشن دیگر لبه واضحی ندارد

تابع های اماده هم به صورت زیر ورودی میگیرند:  
ورودی هایcv2.gausian :

src: تصویر ورودی

ksize: اندازه هسته فیلتر گاوسی. باید یک عدد صحیح فرد مثبت یا دو عدد صحیح فرد مثبت باشد که عرض و ارتفاع هسته را نشان می دهد.

sigmaX: انحراف استاندارد گاوسی در جهت x

Cv2.sobel:

src: تصویر ورودی

ddepth: نوع تصویر خروجی. اگر -1 باشد، تصویر خروجی همان نوع تصویر ورودی را خواهد داشت.

یا اگر مثل انچه مازدیمcv2.cv\_64 باشد float 64 bitاست

dx: ترتیب مشتق در جهت x

dy: ترتیب مشتق در جهت y

ksize: اندازه هسته سوبل. باید یک عدد صحیح فرد مثبت باشد.

(2

تبدیل فوریه سریع (FFT) یک الگوریتم کارآمد برای محاسبه تبدیل فوریه گسسته (DFT) یک دنباله از مقادیر است. DFT یک عملیات ریاضی است که سیگنال حوزه زمان را به نمایش دامنه فرکانس آن تبدیل می کند. الگوریتم FFT از ویژگی های تقارن ماتریس DFT برای کاهش تعداد ضرب های مورد نیاز برای محاسبه DFT از O(n^2) به O(n log n) استفاده می کند که در آن n طول سیگنال ورودی است.

توابع :

Fft(): این تابع برای محاسبه تبدیل فوریه گسسته یک بعدی n نقطه ای (DFT) یک دنباله واقعی یا مختلط استفاده می شود.

fft2(): این تابع برای محاسبه تبدیل فوریه گسسته 2 بعدی n1 x n2 (DFT) یک دنباله واقعی یا مختلط استفاده می شود.

Fftshift(): این تابع برای انتقال مولفه فرکانس صفر به مرکز طیف استفاده می شود.

Ifft(): این تابع برای محاسبه تبدیل فوریه گسسته معکوس یک بعدی n نقطه ای (IDFT) یک دنباله واقعی یا مختلط استفاده می شود.

ifft2(): این تابع برای محاسبه تبدیل فوریه گسسته معکوس دو بعدی n1 x n2 (IDFT) یک دنباله واقعی یا پیچیده استفاده می شود.

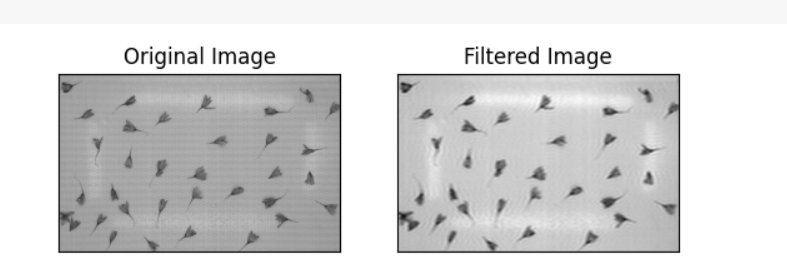
Ifftshift(): این تابع برای خنثی سازی شیفت اعمال شده توسط fftshift() استفاده می شود.

np.abs() یک تابع NumPy است که قabsolute value یک عدد مختلط یا آرایه ای از اعداد مختلط را برمی گرداند. در زمینه پردازش تصویر، معمولاً پس از انجام تبدیل فوریه معکوس بر روی یک تصویر تبدیل فوریه برای به دست آوردن تصویر حوزه فضایی مربوط به آن استفاده می شود. این تابع بزرگی مقادیر مختلط حاصل از تبدیل فوریه معکوس را برمی گرداند.

cv2.convertScaleAbs() یک تابع OpenCV است که یک آرایه ورودی را به فرمت عدد صحیح بدون علامت 8 بیتی تبدیل می کند. یک آرگومان واحد می گیرد که آرایه ورودی را نشان می دهد و یک آرایه خروجی با همان اندازه و نوع را برمی گرداند، اما هر عنصر مقیاس شده و به یک عدد صحیح 8 بیتی بدون علامت تبدیل شده است. این تابع معمولاً برای تبدیل یک تصویر ممیز شناور به فرمت 8 بیتی استفاده می شود تا بتوان آن را به عنوان یک فایل تصویری در فرمت هایی که فقط از تصاویر 8 بیتی پشتیبانی می کنند مانند JPEG یا PNG نمایش داده یا ذخیره کرد.

برای این سوال ما ابتدا با تبدیل فوریه عکس را به فضای فوریه میبریم و سپس فرکانس های پایین را به مرکز میاوریم و یک mask میسازیم که فقط سیگنال های فرکانس پایینش 1 و فرکانس بالا هایش صفر است سپس با ضرب کردن در فوریه تصویر سیگنال های فرکانس بالا که اغلب نویز اند را حذف میکنیم (در تصاویر طبیعی اغلب سیگنال ها فرکانس پایین اند)

در اخر هم فوریه معکوس میزنیم و عکس را نشان میدهیم



همانطور که مشاهده میشود تصویر حاصل دارای نویز کمتری است

بخش 2:

در این بخش با استفاده از تابع cv2.canny که الگوریتم لبه یاب canny را بر روی تصویر اعمال میکند لبه ها را میابیم:

ورودی:

تصویر: تصویر ورودی که باید یک تصویر خاکستری باشد.

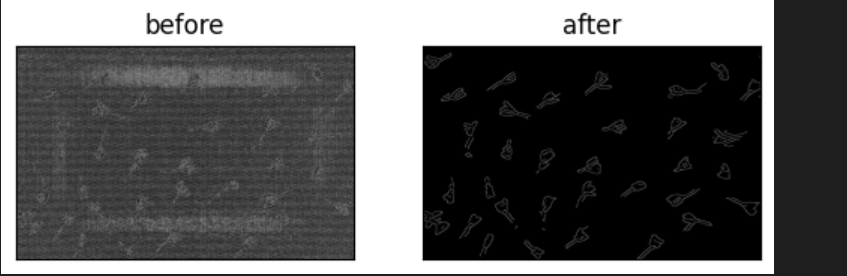
Double Thresholding:

threshold1: اولین آستانه ای که برای تشخیص لبه های قوی استفاده می شود.->

threshold2: آستانه دوم که برای تشخیص لبه های ضعیفی که به لبه های قوی متصل هستند استفاده می شود.

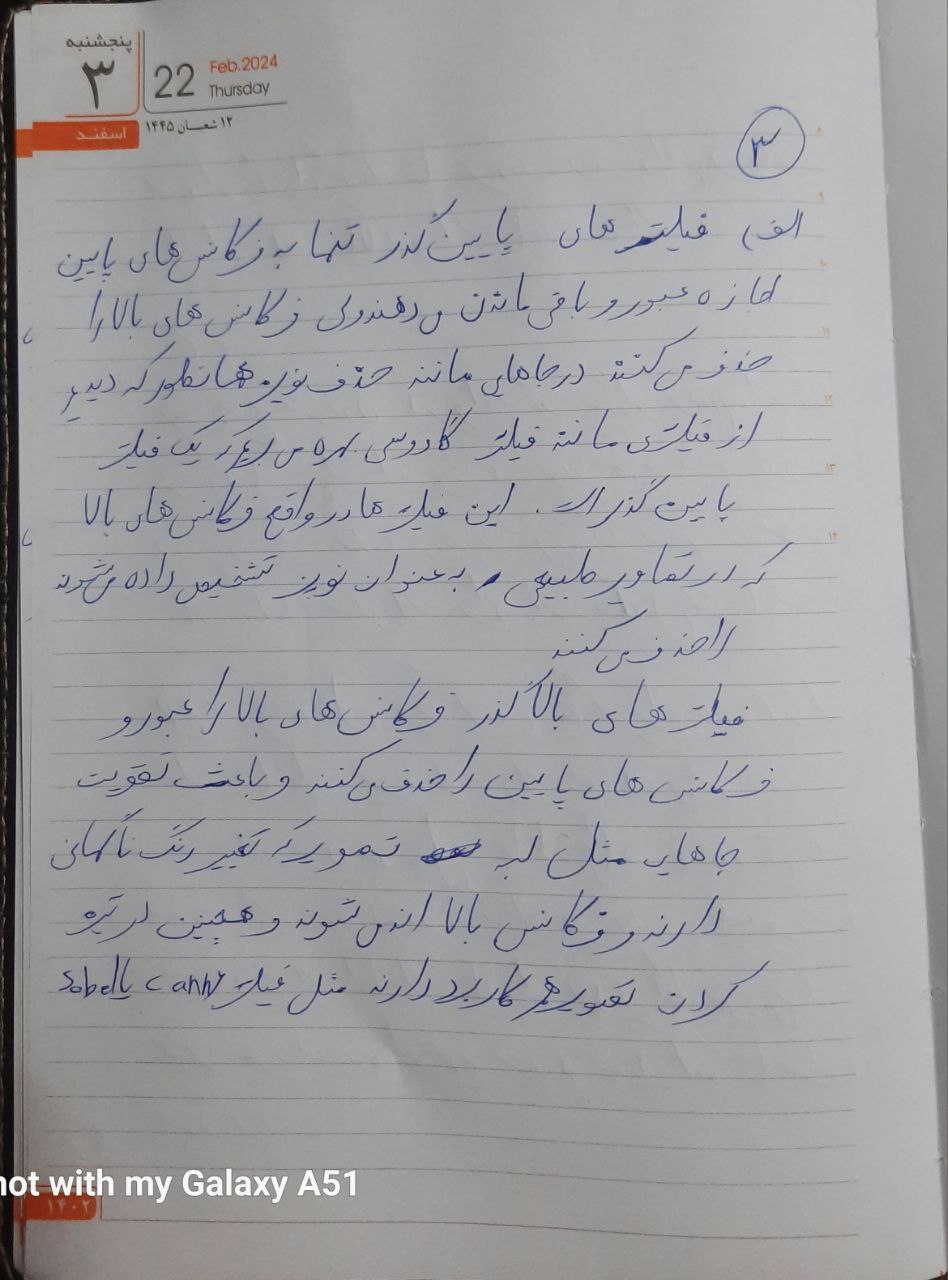
apertureSize: اندازه هسته Sobel که برای محاسبه گرادیان استفاده می شود. به طور پیش فرض، apertureSize=3. هر چه بزرگتر باشد اسموسینگ بیشتر است

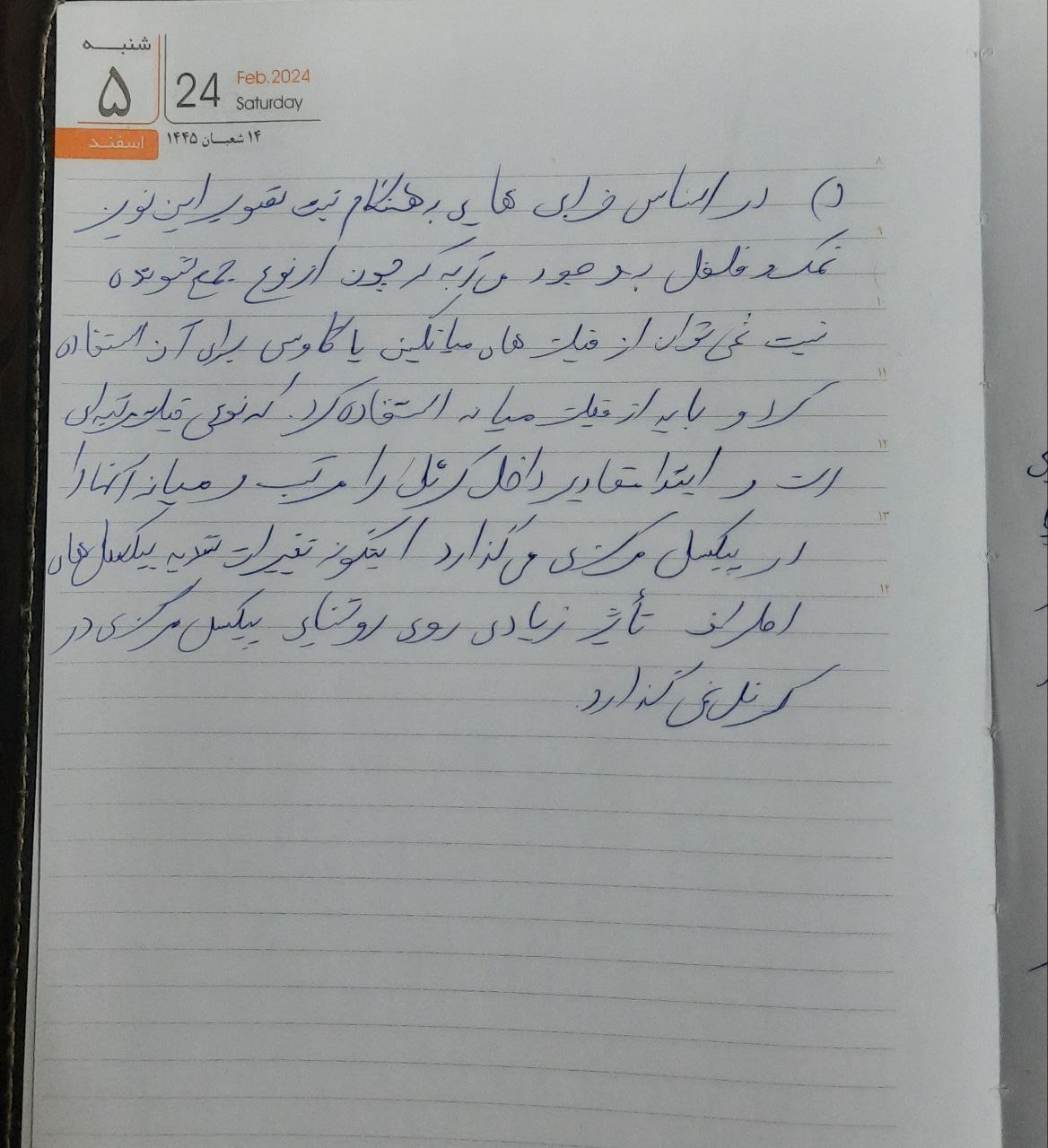
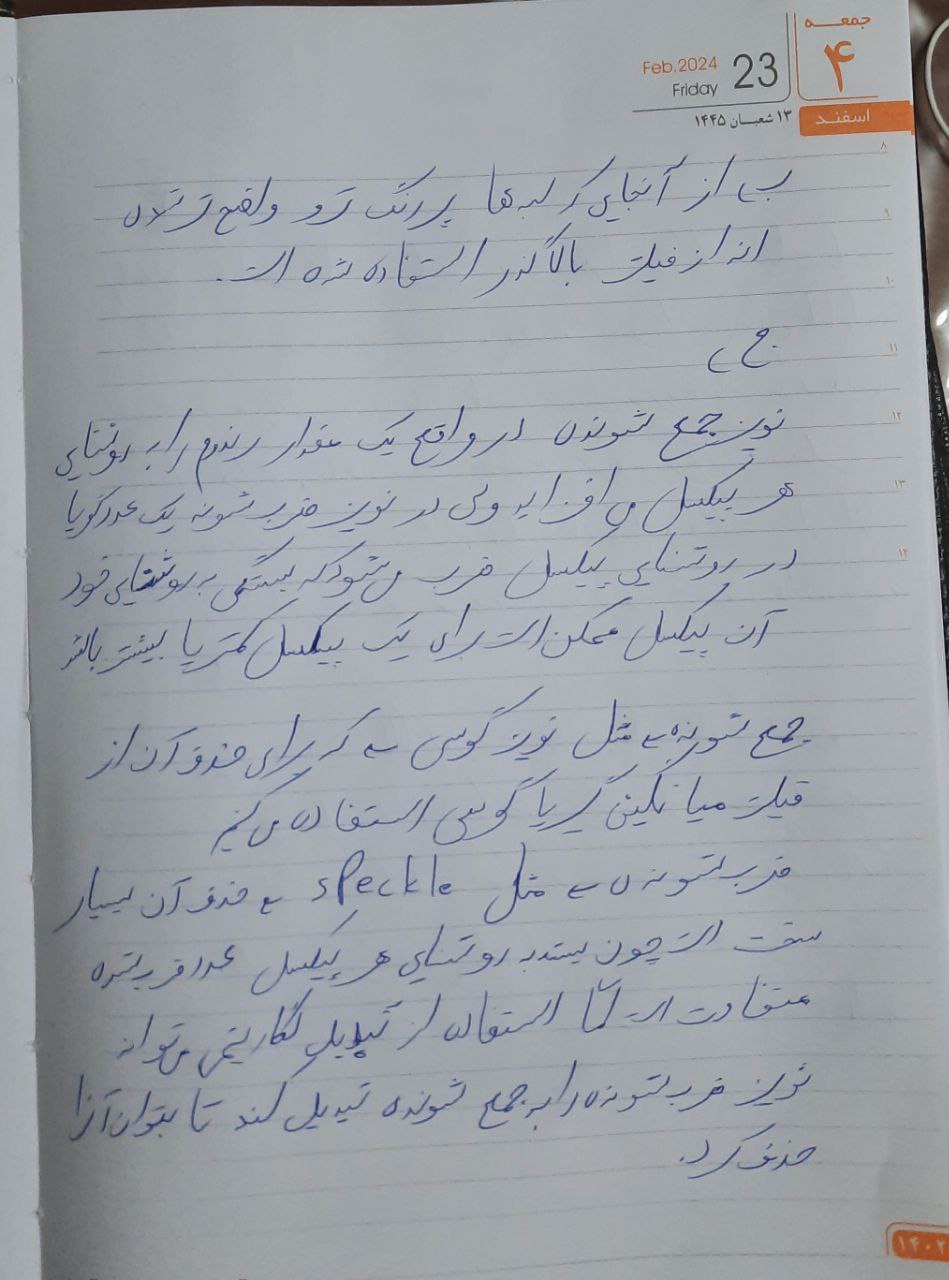
تابع یک تصویر باینری را برمی‌گرداند که در آن لبه‌ها با پیکسل‌های سفید و غیر لبه‌ها با پیکسل‌های سیاه مشخص شده‌اند.



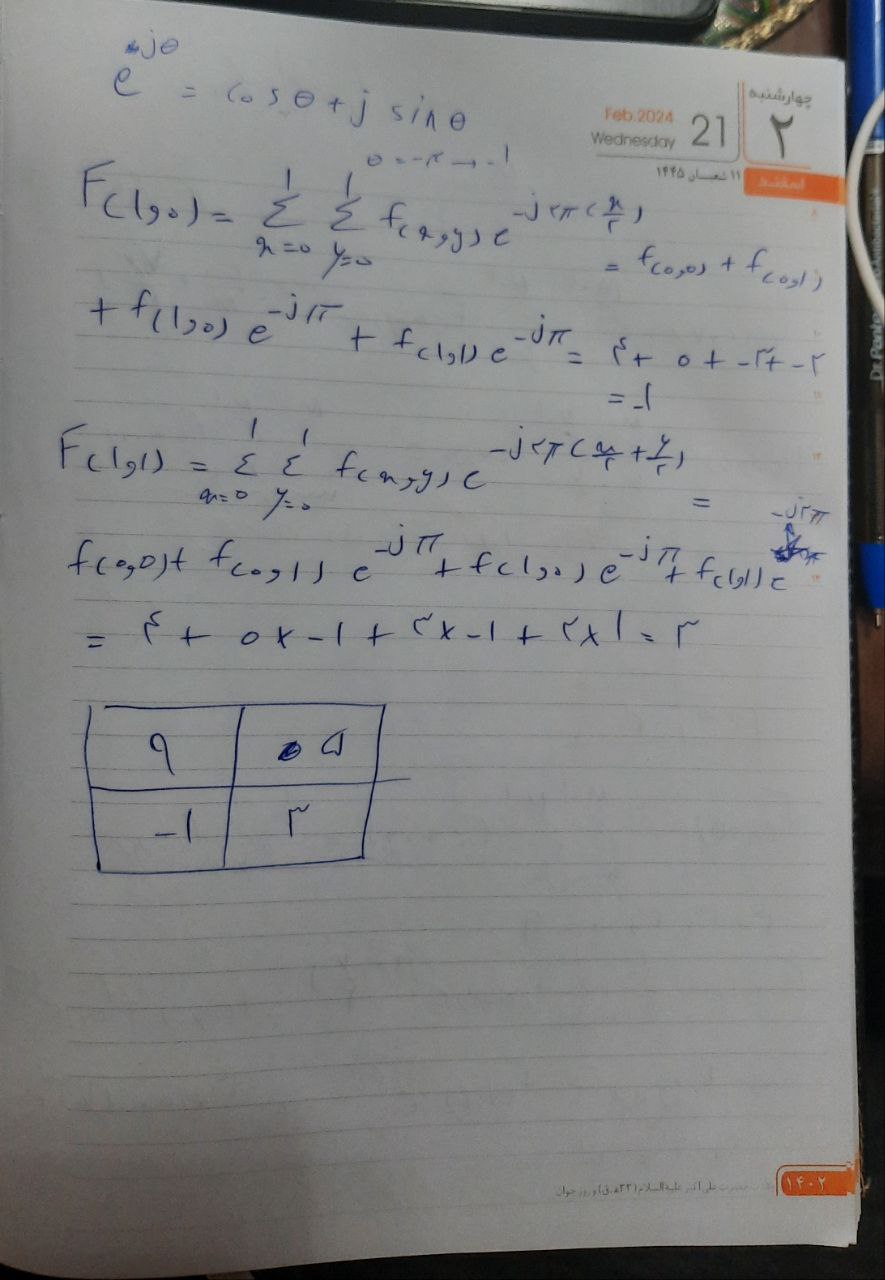
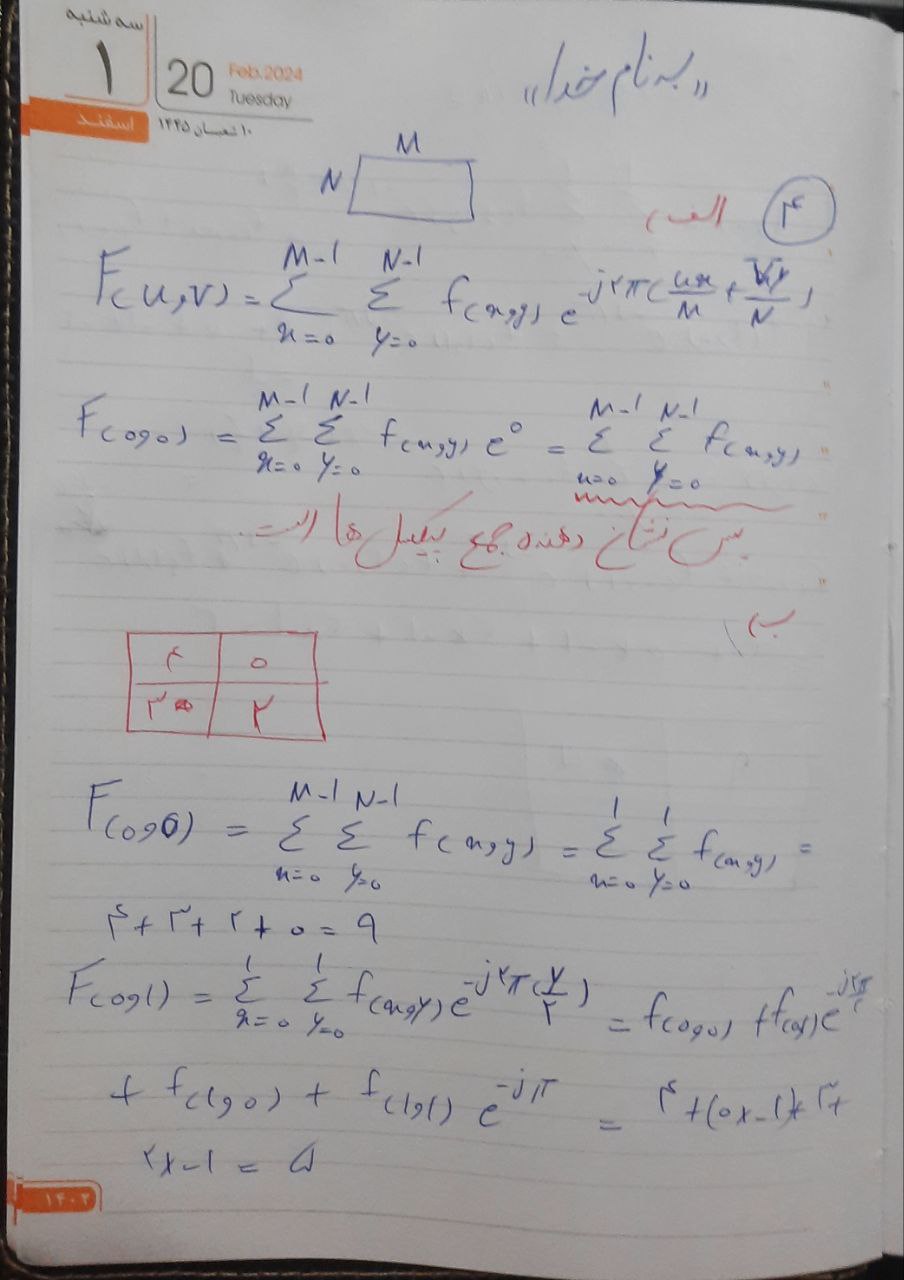
همانطور که مشاهده میشود در عکس که با مسک مد نظر ما نویز اش حذف شده نویز کمتر و لبه ها واضح تر اند

(3



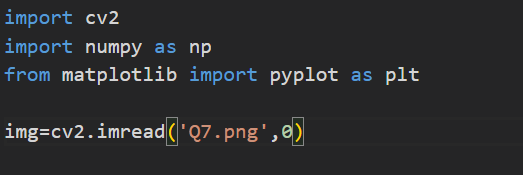


(4



(5

ابتدا عکس را میخوانیم



سپس با استفاده از canny لبه ها را می یابیم



تابع findContours:

تابعی در کتابخانه OpenCV است که برای تشخیص خطوط در یک تصویر باینری استفاده می شود. تصویر باینری ورودی و حالت بازیابی کانتور و روش تقریب کانتور را به عنوان پارامترهای ورودی می گیرد و لیستی از خطوط شناسایی شده را برمی گرداند. خطوط، فهرستی از آرایه های Numpy هستند، که در آن هر آرایه حاوی مختصات (x، y) نقاط مرزی شی است.

این تابع در واقع یک ورودی باینری میگیرد و تعددادی contour به عنوان خروجی برمیگرداند هر کدام در واقع مجموعه نقاط یک شکل هندسی در عکس اند

ورودی ها:

تصویر ورودی: تصویر باینری که می خواهیم خطوط را در آن تشخیص دهیم.

Contour Retrieval mode: حالت بازیابی را مشخص می کند. چهار گزینه دارد:

cv2.RETR\_EXTERNAL: فقط خطوط خارجی را بازیابی می کند -> 4 شکل را شناسایی میکند که برای ما مطلوب است

cv2.RETR\_LIST: تمام خطوط را بدون هیچ سلسله مراتبی بازیابی می کند

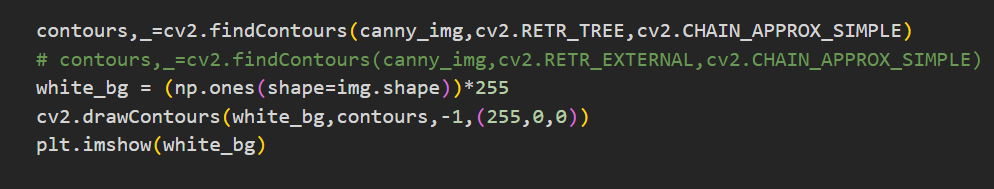
cv2.RETR\_CCOMP: همه خطوط را بازیابی می کند و آنها را در یک سلسله مراتب دو سطحی سازماندهی می کند. در سطح بالایی، مرزهای خارجی اجزا قرار دارد. در سطح دوم مرزهای سوراخ ها قرار دارد.

cv2.RETR\_TREE: همه خطوط را بازیابی می کند و یک سلسله مراتب کامل از خطوط تودرتو را بازسازی می کند.<- 16 برمیگرداند یعنی 16 شکل را شناسایی میکند که مطلوب ما نیست

روش تقریب Contour: روش تقریب را مشخص می کند. سه گزینه دارد:

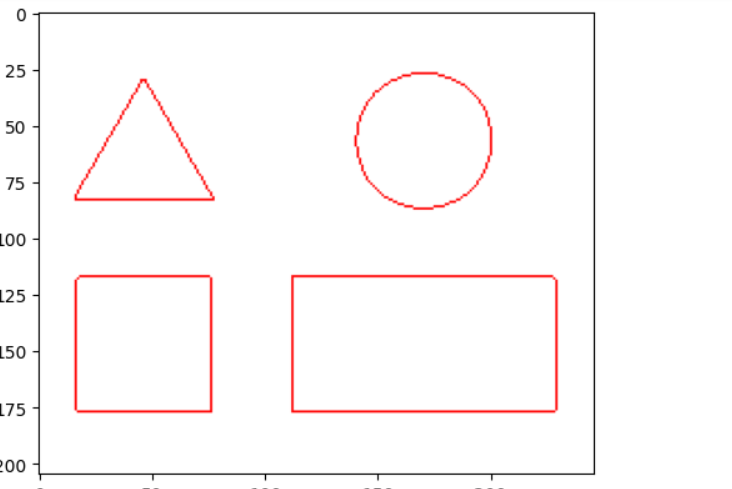
cv2.CHAIN\_APPROX\_NONE: تمام نقاط کانتور را ذخیره می کند.

cv2.CHAIN\_APPROX\_SIMPLE: بخش‌های افقی، عمودی و مورب را فشرده می‌کند و فقط نقاط انتهایی آن‌ها را باقی می‌گذارد.

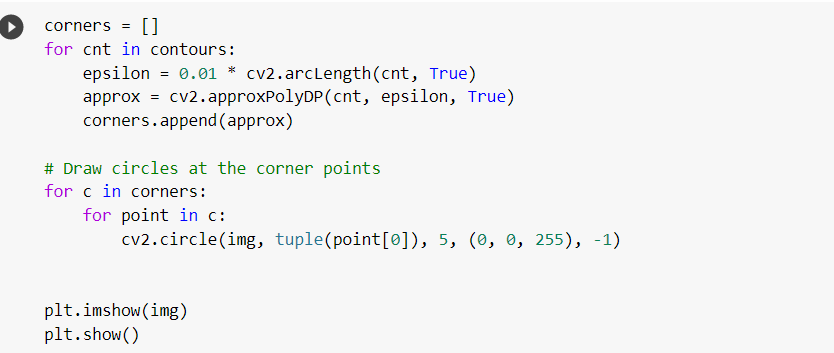


White back ground هم درست کردم که روی اون اشکال رو نشان بدهد

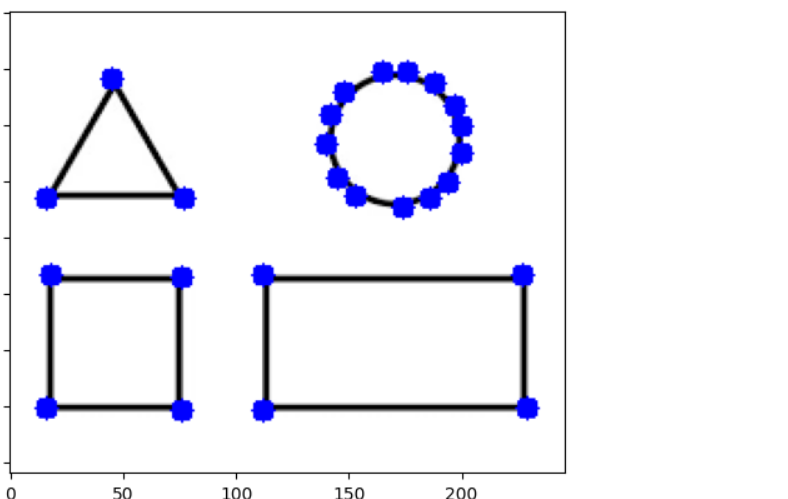
drawContours هم در واقع میاد نقاطی که پیدا کردیم و در یک contour به عنوان معرف شکل گذاشتیم رو به هم وصل میکنه



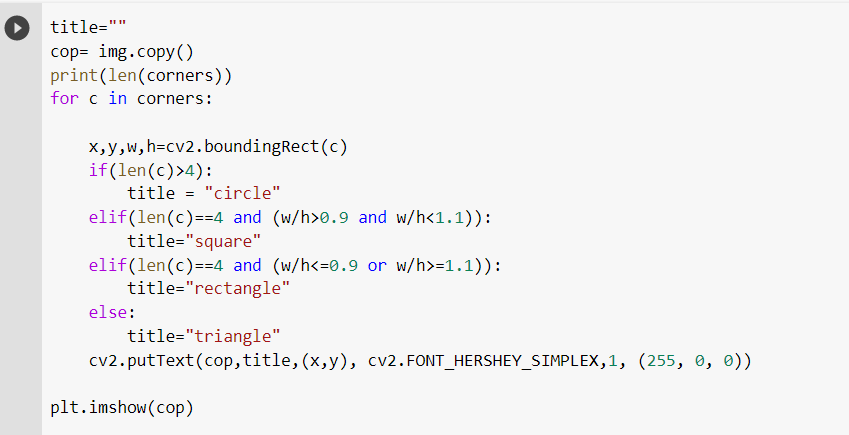
برای یافتن گوشه ها از تابع approxPolyDP استفاده میکنیم که میاد حداقل نقاط لازم برای شناسایی تصویر رو با دقتی که ما تعیین میکنیم به دست میاورد من ابتدا دقت را 0.1 گذاشتم که برای دایره هم 4 تقطه گذاشت وقتی رسمش کردم ولی دقت رو که بالا بردم و عدد 0.01 گذاشتم درست گوشه ها را تشخیص داد



Epsilon اون خطایی است که در نظر میگیریم approx. در واقع اون نقاطی است که با توجه به خطایی که در نظر میگیریم متفاوته و در واقع از بین خروجی های تابع findContours انتخاب میشه برای هر شکل در دو حلقه اخر هم گوشه ها را علامت زدم



حال برای شناسایی اشکال درواقع از همین تعداد گوشه ها و از تابع boundingrect که به ما اندازه اضلاع و نقطه گوشه بالا سمت چپ را میدهد استفاده میکنیم تا بتوانیم مستطیل و مربع را از هم تشخیص دهیم و بدانیم کجا باید با ایتفاده از تابع put text متن مد نظ را بنویسیم



putText:

image

text متن

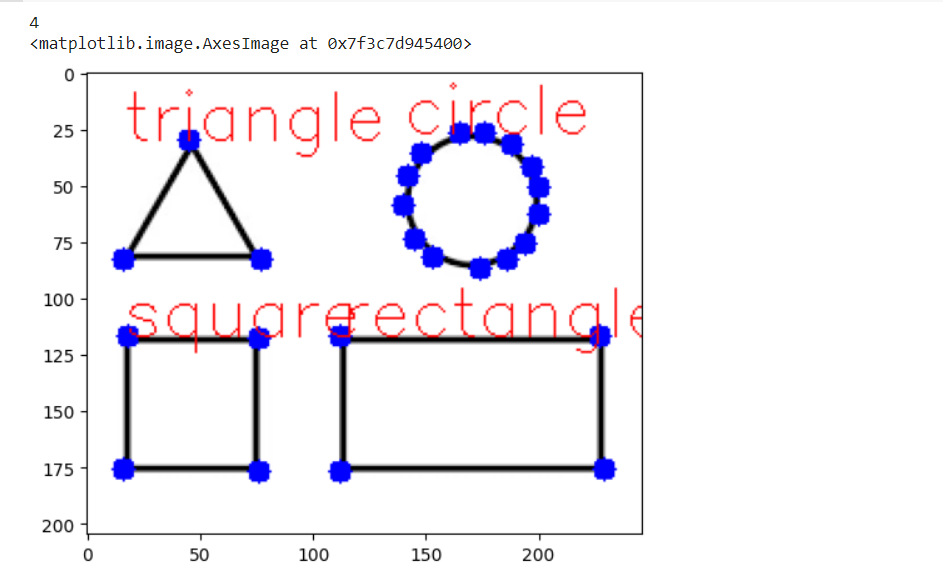
location موقعیت در عکس

font

font scale

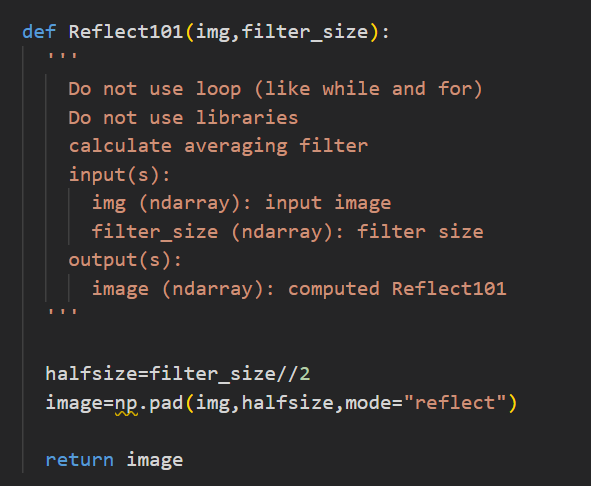
color رنگ

thickness ضخامت متن

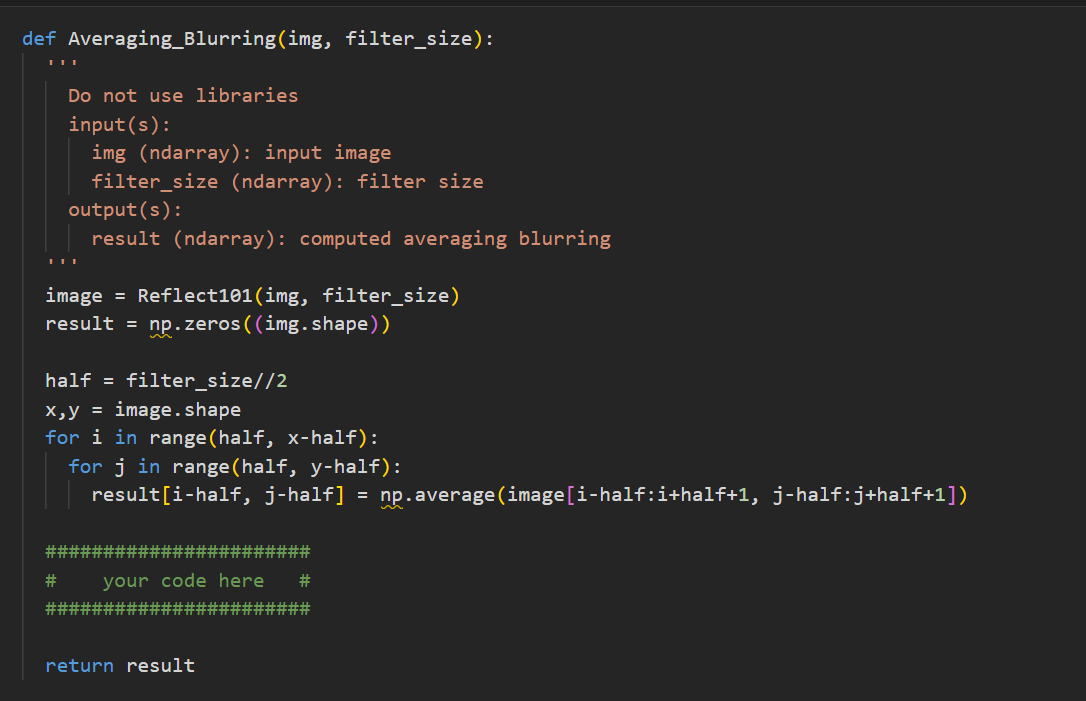


6)

Reflect101:



Mode تابع pad را reflect میگذاریم و از ان استفاده میکنیم



Average Bluring:

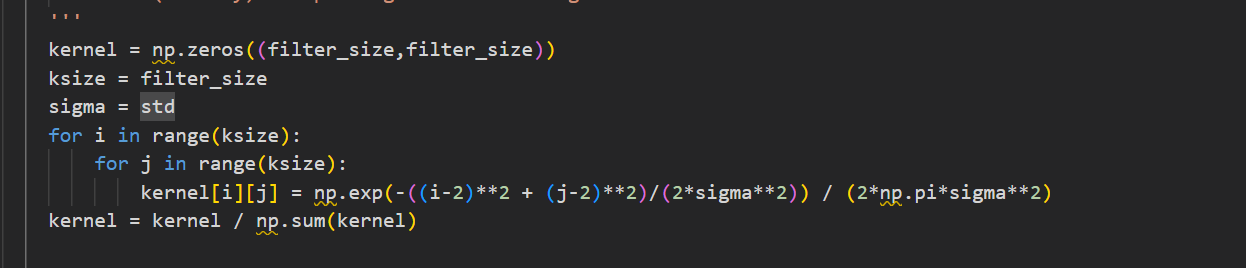
بعد از اعمال padding با np.average میانگین گیری میکنیم

Median Blurring:

Np.median استفاده میکنیم:



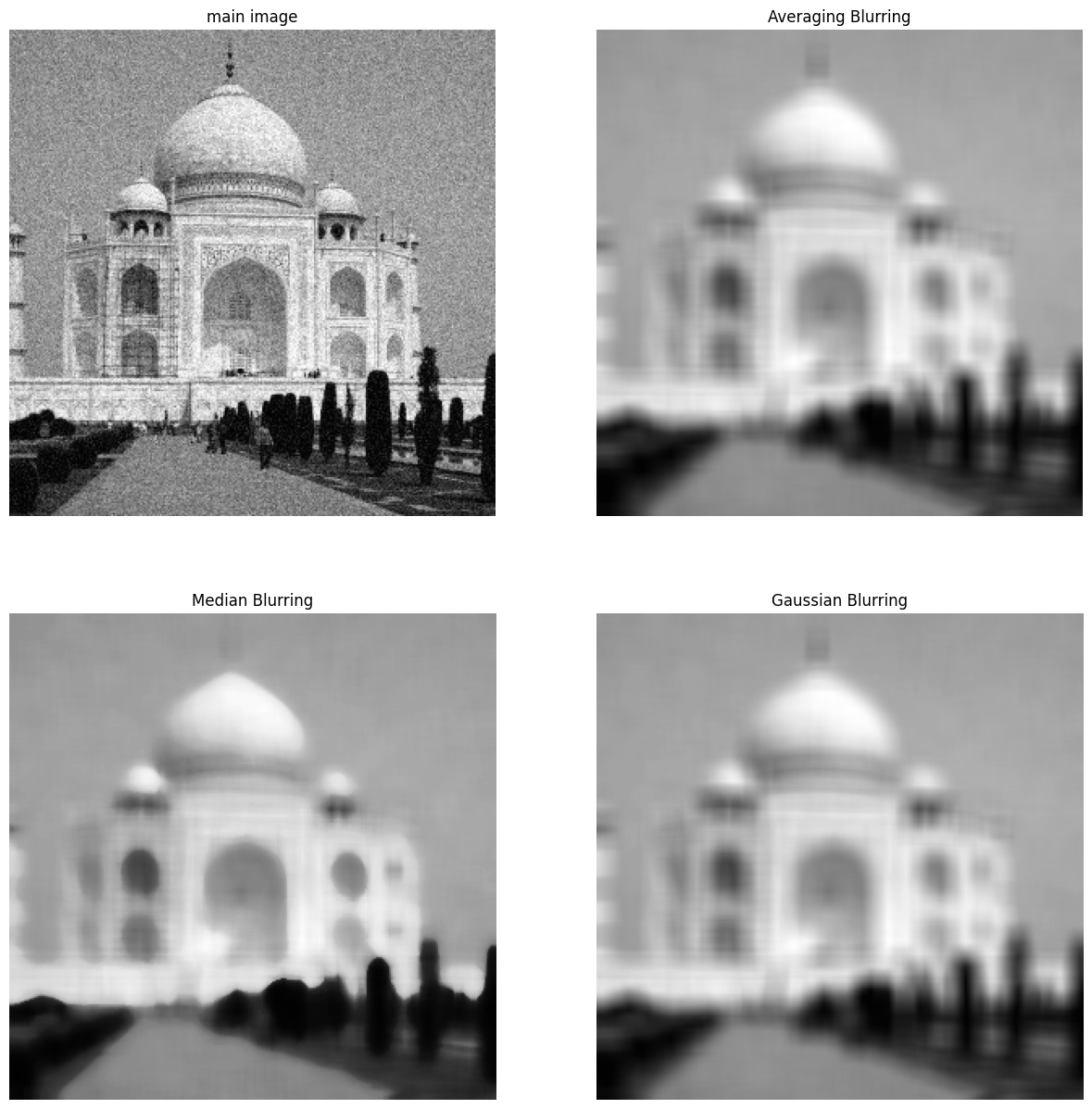
در پیاده سازی گاووسین هم فقط از فرمول مرنل را میسازیم:



سایز کرنل اینگونه تاثیر میگذارد که هرچه بزرکتر باشد تصویر تار باشد و نویز هم کمتر ولی اگر کوچکتر باشد تصویر کمتر تار شده و نویز هم کمتر حدف میشود

زیرا در حالت بزرگتر کرنل از داده ها تصویر هم بیشتر کم میشود

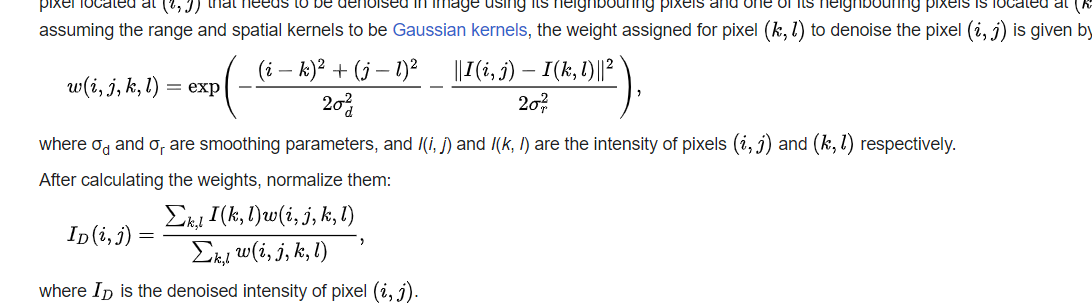
خروجی ها:



با سایز 15 همانطور که دیده مشود تصویر خیلی تار مشود

ب)

ما ابتدا فرمول این فیلتر را میابیم

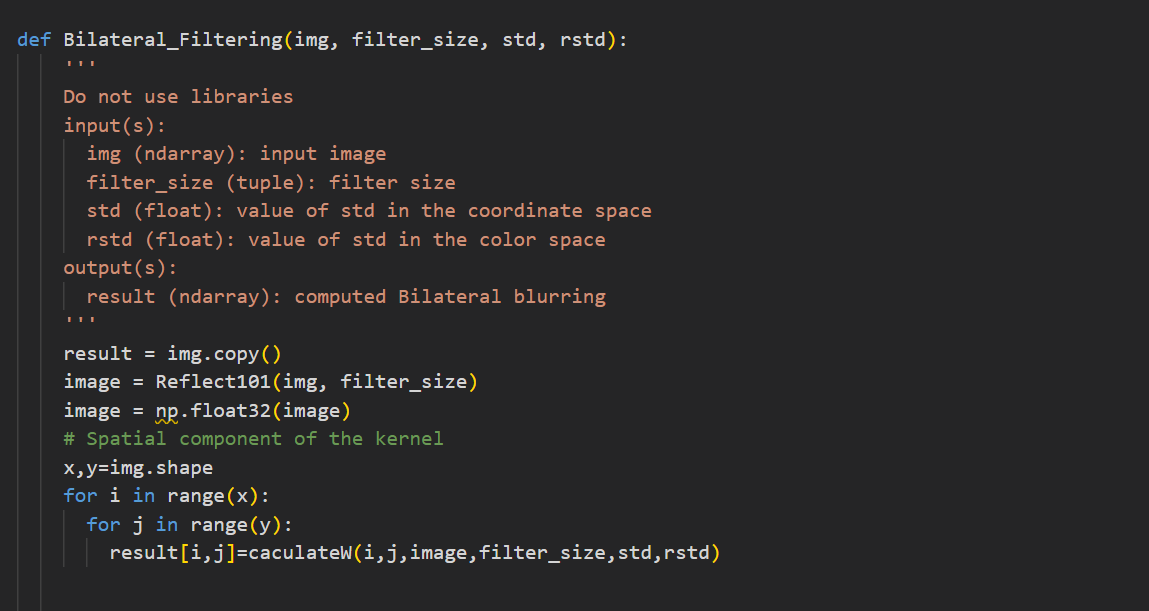


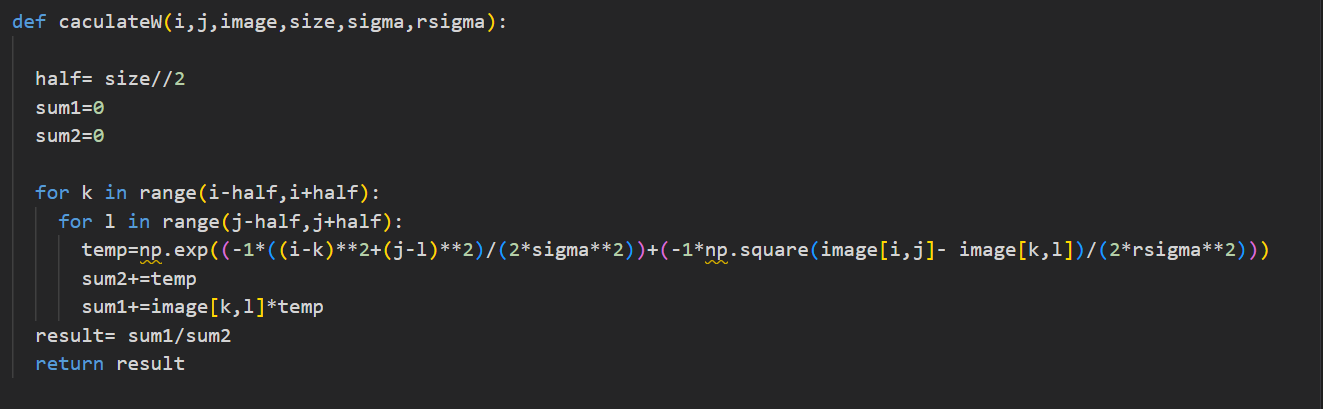
من از این دو فرمول استفاده کردم که ابتدا وزن ها را یافته و بعد روشنایی متناسب را می یابد با فیلتر گاووسی در این متفاوت است که علاوه بر تاثیر فاصله از مرکز میزان تفاوت روشنایی را نیز دخیل میکند و به این شکل برای هر پیکسل جدا روشنایی را به دست میاورد

کاربرد این فیلتر در این است که علاوه بر حذف نویز تصویر را تار نمیکند زیرا از انجایی که تفاوت روشنایی را در وزن دادن دخیل میکند لبه ها را حذف نمیکند در واقع در حالت لبه بودن به پیکسل مرکزی وزن بیشتری میدهد تا ان را تار نکند

در مورد سیگما اگر به بینهایت برود میانگین گیری معمولی میشود و اگر به صفر برود تغییری در تصویر رخ نمیدهد

در مورد انحراف معیار روشنایی خیلی زیاد شود به فیلتر گاووسی نزدیک میشویم و اگر مکانی زیاد شود فیلتری میشود که بر اساس تفاوت روشنایی کار میکند





در تابع calculateW در واقع روشنایی جدید ان پیکسل طبق فرمول محاسبه میشود

خروجی تصویر طبق انچه انتظار داریم بسیار واضح تر از قبلی ها است  
