به نام خدا

Table of Contents

[**سوال سوم** 3](#_Toc196499680)

[**بخش الف** 3](#_Toc196499681)

[**انواع نویز** 3](#_Toc196499682)

[**گزارش کار** 5](#_Toc196499683)

[**بخش ب** 11](#_Toc196499684)

[**انواع فیلتر** 11](#_Toc196499685)

# **سوال سوم**

برای این سوال 3 فایل notebook پیاده‌سازی شده است که نتایج خروجی آنها هم به صورت فایل تصویر در همین پوشه قرار دارد (تحت عنوان PART?\_output) هم در خود فایل ها قرار دارد و هم در قسمت گزارش کار وجود دارد. توضیحات مربوط به کد در قسمت گزارش کار بخش مرتبط با آن قرار داده شده است. در هر بخش توضیحات مورد نیاز نیز داده شده است.

## **بخش الف**

در این قسمت ما از یک تصویر grayscale به عنوان ورودی برای اعمال نویزهای مختلف ورود کردیم. برای محاسبه یک تصویر grayscale از روش Luminosity استفاده کردیم که یک جمع وزن دار از کانال‌های RGB است و بازتاب دهنده حساسیت چشم انسان است.

در این پروژه ما 3 نوع نویز را پیاده‌سازی کردیم که در ادامه هر کدام را توضیح مختصر میدهم.

### **انواع نویز**

#### **نویز Gaussian**

نویز گوسی یک نویز آماری است که در آن تغییراتی که به هر پیکسل اضافه میشود از یک توزیع گوسی پیروی میکند که مانند bell curve است. به این نویز معمولا نویز گوسی سفید افزایشی نیز گفته میشود. افزایشی به این معنا است که مقدار نویز به مقدار اولیه پیکسل‌ها اضافه میشود. سفید یعنی اینکه نویز مستقل هست بین پیکسل‌های مختلف و توان آن در تمام فرکانس‌های مختلف برابر است. مقدار نویز در این روش از یک توزیع گوسی به صورت تصادفی نمونه برداری میشود. میانگین معمولا 0 است. و با پارامتر سیگما شدت نویز را کنترل میکنیم هر چه پارامتر سیگما بالاتر باشد نویز قوی تر است که همه این موارد در پیاده‌سازی نیز قابل مشاهده است.

#### **نویز فلفل و نمک**

این نویز یک نویز ضربه‌ای است که به جای اینکه یک مقدار تصادفی را به مقادیر پیکسل اضافه کنیم، مقدار پیکسل اولیه را یا با کمینه شدت ممکن ( 0 که سیاه است و معنی فلفل را میدهد) و یا بیشینه شدت ممکن (255 که همان سفید است و معنی نمک را میدهد) جایگزین میکنیم. در این روش فقط درصد مشخصی از پیکسل‌ها تحت تاثیر قرار میگیرند. همانطور که خروجی مشاهده خواهد شد شبیه این است که یک سری نقاط سفید و سیاه به صورت تصادفی در سراسر عکس پخش شده اند شبیه دانه‌های نمک و فلفل.

برای انتخاب تعدادی از پیکسل‌ها ما یک احتمالی را در نظر میگیریم به این صورت که اگر اون پیکسل انتخاب شده احتمال سیاه شدن برابر با تعداد آنها ضربدر 1 منهای تعداد نمک و فلفل ها است و همین حالت برای احتمال سفید شدن و اگر هیچکدوم از این 2 نشد مقدار آن پیکسل دست نمیخورد. به صورت دقیق تر:

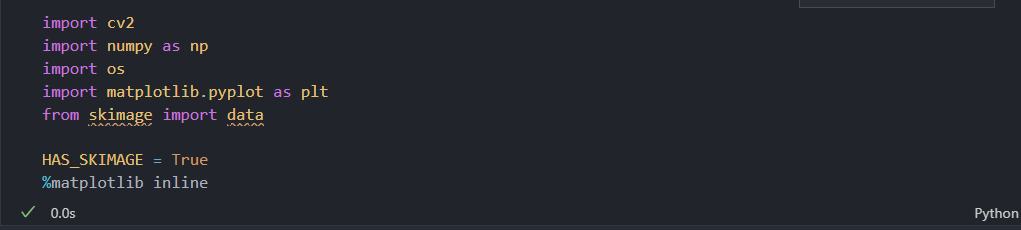
* Noisy\_Pixel = 0 (Pepper) with probability amount \* (1 - salt\_vs\_pepper)
* Noisy\_Pixel = 255 (Salt) with probability amount \* salt\_vs\_pepper
* Noisy\_Pixel = Original\_Pixel otherwise.

#### **نویز speckle**

این نویز ضربی است به جای افزایشی بودن به این معنی که مقدار پیکسل‌ها به نوعی مقیاس میشود (scale میشود). اغلب شبیه یک بافت دانه‌دانه است که روی تصویر قرار گرفته که نواحی روشن‌تر در تصویر اصلی، معمولا تغییرات نویز شدیدتری نسبت به نواحی تاریک تر دارند. نکته‌ای که مطرح است اینکه مقدار نویز در اینجا نیز یک مقدار متغیر تصادفی با میانگین صفر است مانند توزیع گوسی و تنها فرق آن ضرب شدن نویز در مقدار پیکسل اولیه است.

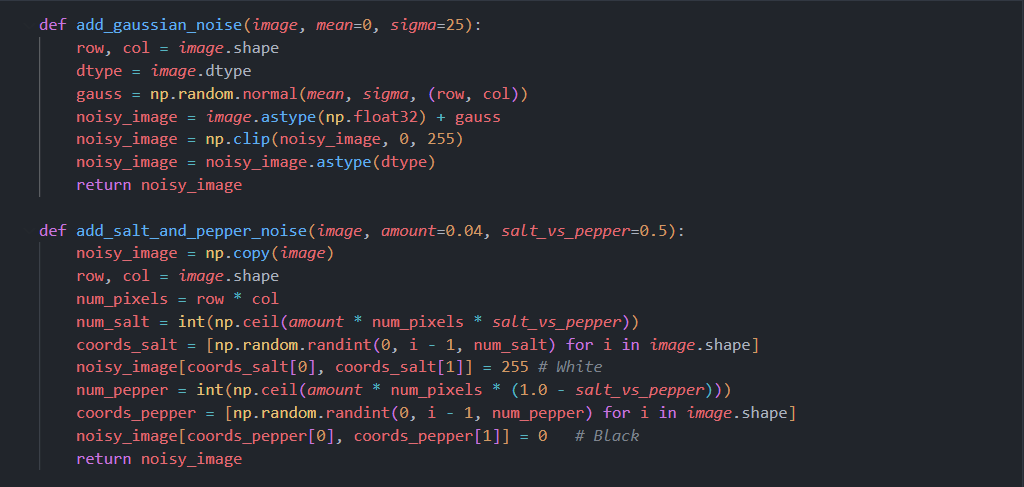
### **گزارش کار**

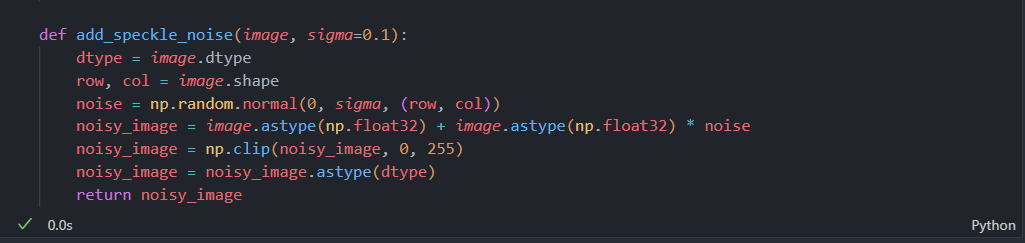
در این قسمت کد پیاده‌سازی را بخش به بخش توضیح میدهیم:



از کتابخانه cv2 برای اعمال عملیات ها مختلف مانند خواندن عکس‌ها استفاده میکنیم.

برای نشان دادن خروجی از matplotlib استفاده کرده‌ایم. همچنین تصاویر مورد نیاز را از scikit-image تحت عنوان skimage بارگذاری کرده‌ایم. که در این پروژه از تصویر معیار cameraman استفاده کردیم. خط آخر هم صرفا برای این است که تمام خروجی به صورت inline داده شود.

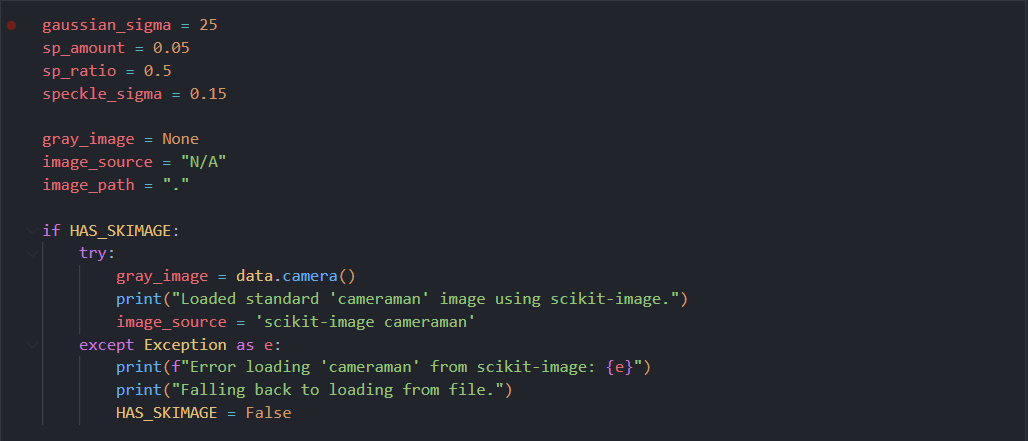


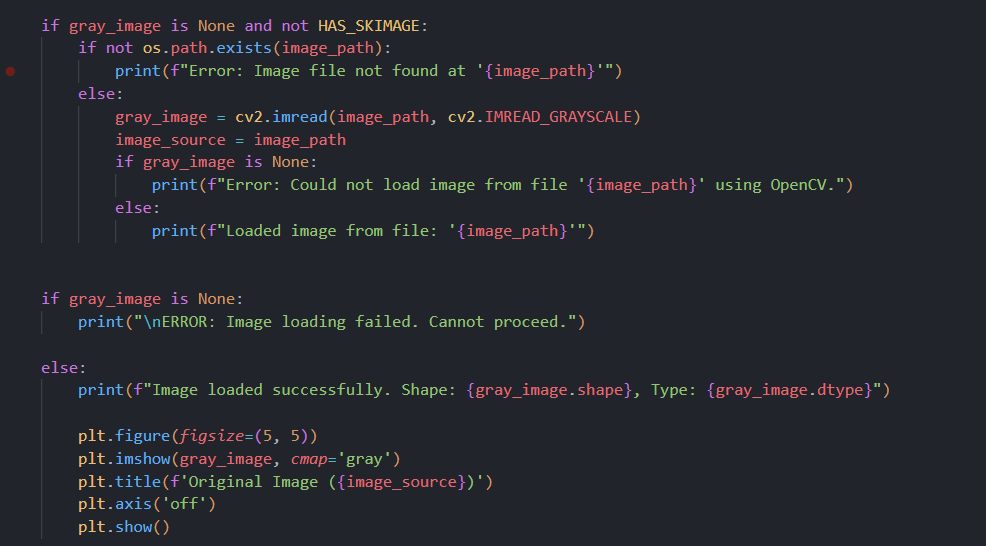


3 تابع مختلف را برای 3 نویزی که قبل توضیح دادیم پیاده‌سازی کردیم، در تابع gaussian ابتدا مطمئن شدیم اعداد float هستند همراه با dtype اولیه بعد از آن با کمک noisy\_image به تصویر نویز اضافه کردیم. در مرحله بعدی مقادیر را بین 0 تا 255 محدود کردیم با کمک clip و در نهایت تصویر را به dtype اصلی برگرداندیم.

در تابع salt\_and\_pepper 3 تا آرگومان داریم. آرگومان اول که تصویر grayscale است. آرگومان دوم درصدی از پیکسل ها هستند که قرار است با نویز جایگزین شوند(amount). آرگومان سوم درصدی از نویزها هستند که قرار است یا سفید شوند و یا سیاه شوند همانطور که در بالا توضیح دادیم(salt\_vs\_pepper). و در نهایت خروجی نویزی شده را برگرداندیم. در ادامه همین تابع یک سری مختصات تصادفی برای نویزهای سفید و سیاه تولید کردیم و نویزهای سفید و سیاه را به تصویر اضافه کردیم.

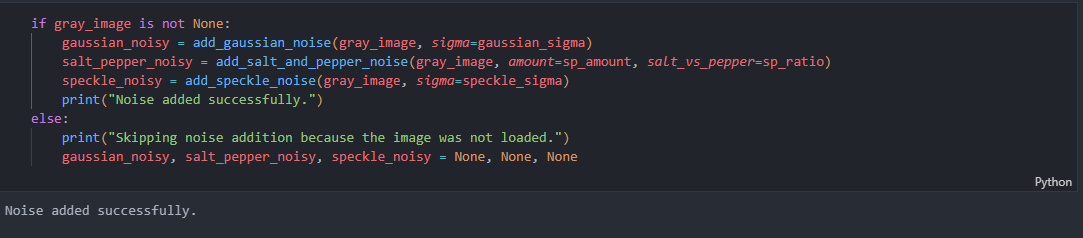
در تابع speckle\_noise نیز همانطور که توضیح دادیم قرار است مقادیر پیکسل را در مقدار نویز ضرب کنیم. چون قرار است به یک توزیع نیاز داشته باشیم پس از پارامتر سیگما در آرگومان استفاده کرده‌ایم در این قسمت از توزیع گوسی استفاده کردیم.

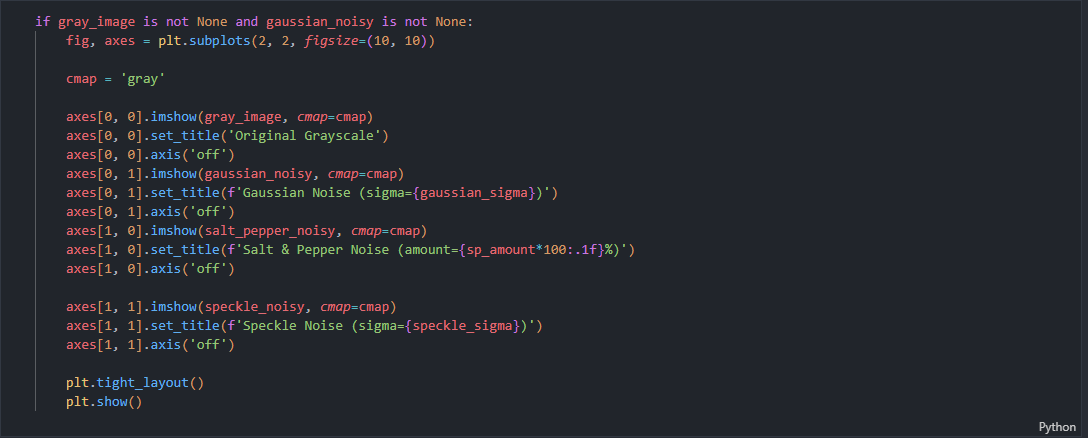




در این قسمت ابتدا پارامترهای نویز را تنظیم کردیم برای هر کدام از 3 تابعی داریم. Gaussian\_sigma برای میزان شدت نویز در تابع گوسی، sp\_amount و sp\_ration برای تابع salt and pepper است که میزان پیکسل‌های که قرار است توسط این تابع تحت تاثیر قرار بگیرند را مشخص میکنیم و نسبت سیاه به سفید را مشخص میکنیم. متغیر speckle\_sigma نیز برای شدت نویز تابع speckle است. بعد در ادامه از کتابخانه scikit-image کمک میگیریم و تصویر cameraman را بارگذاری میکنیم و بررسی میکنیم عکس به درستی در دسترس قرار گرفته باشد و در نهایت خروجی اولیه را که عکس اصلی cameraman هست به درستی دریافت کرده و نمایش میدهیم:

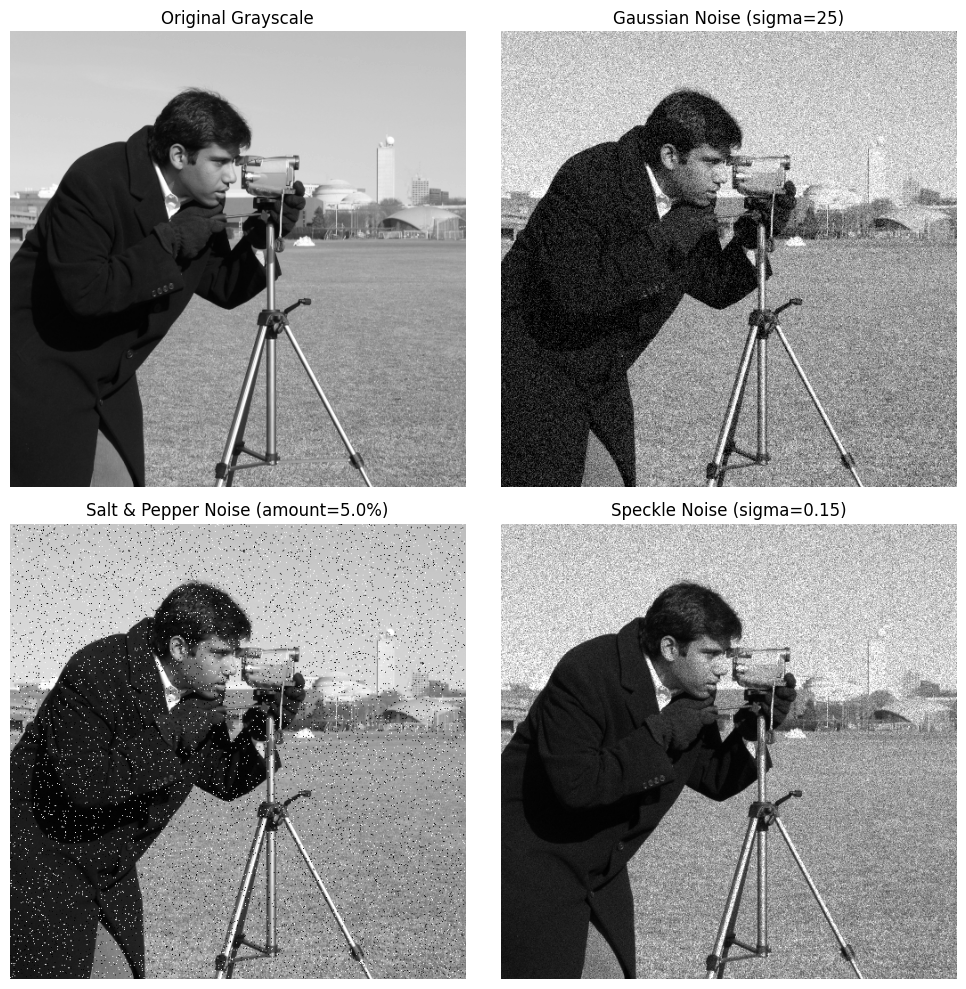


همانطور که در تصویر بالا مشخص است نویز را بر روی تصویر دریافت شده اعمال کردیم.



در نهایت خروجی نویزی شده را به نمایش میگذاریم.

خروجی:



## **بخش ب**

در این بخش ما ابتدا 3 فیلتر گفته شده را توضیح میدهیم و بعد در قسمت گزارش کار خود کدها را توضیح میدهیم و نتایج بدست آمده را تحلیل میکنیم.

### **انواع فیلتر**

#### **فیلتر گاوسی**

این فیلتر کمک تا نویز در تصویر کمتر شود و تصویر هموار شود به ویژه وقتی تصویر دارای نویز گوسی است. در واقع فیلتر گاوسی یک low-pass فیلتر است به معنی اینکه مولفه‌های high-frequency را تضعیف میکند مانند لبه‌های تیز. نحوه کارکردن این فیلتر به اینگونه است که هر مقدار پیکسل را با ید میانگین وزن دار از مقادیر پیکسل همسایه‌اش جایگزین میکند. وزن ها توسط یک تابع دو بعدی گاوسی که شبیه به bell curve است مشخص میشوند. پیکسل‌هایی که به مرکز همسایگی نزدیک هستند وزن بیشتری دارند و هر چه قدر دورتر میشویم پیکسل‌های دور تر وزن کمتری دارند. 2 پارامتر ksize و sigmaX داریم که اولی برای تعریف سایز همسایگی استفاده میشود که باید عدد فرد و مثبت باشد. هر چه همسایگی بزرگتر باشد تصویر blur تر میشود، پارامتر دوم برای مشخص کردن انحراف معیار استفاده میشود که پراکندگی وزن‌ها را مشخص میکند، طبیعتا مقادیر بزرگتر باعث تصویری هموارتر و blurتر میشود. این فیلتر برای کاهش نویز گوسی بسیار مناسب است اما blur شدن یکی از مشکلات آن است و در برابر نویز salt & pepper کارآمد نیست.

#### **فیلتر median**

این فیلتر به صورت عمده برای حذف نویز ضربه‌ مانند salt & pepper استفاده میشود و نسبت به لبه‌ها عملکرد بسیار بهتری نسبت به فیلترهای خطی مثل گاوسی دارد. نحوه کارکرد این فیلتر به اینگونه است که یک پنجره یا یک کرنل بر روی تصویر حرکت میدهد و برای هر موقعیت، همه مقادیر پیکسل داخل پنجره را جمع‌آوری میکند و به صورت عددی آنها را مرتب میکند و بعد مقدار پیکسل مرکزی را با median یا میانه مقادیر لیست مرتب شده جایگزین میکند. مانند نویز گاوسی یک پارامتر به نام ksize دارد که سایز همسایگی که اندازه پنجره یا کرنل بر اساس تعریف میشود را مشخص میکند. همانطور که گفته شد برای نویزهای نمک و فلفل بسیار خوب عمل میکند زیرا مقادیر نمک و فلفل که همان سیاه سفید هستند مقادیر خیلی بالا یا پایینی هستند و در میانه قرار نمیگیرند در همسایگی در نظر گرفته شده همچنین نسبت به لبه‌ها بسیار بهتر از گاوسی عمل میکند به خاطر اینکه median یا میانه نسبت به داده های پرت (outliers) مقاوم تر هستند. البته اگر سایز کرنل کوچک در نظر بگیریم ممکن است گاهی اوقات خطوط درست را حذف کند، اگر عکس فقط دچار نویز گاوسی شده باشد خیلی ضعیف تر از فیلتر گاوسی عمل میکند و همچنین از فیلتر گاوسی کند تر است.

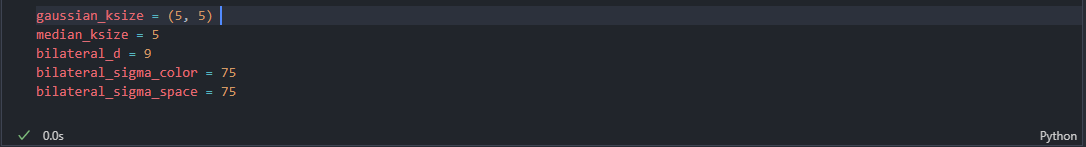
#### **فیلتر bilateral**

وظیفه این فیلتر کاهش نویزهای هست که لبه‌ها را شدیدا تحت تاثیر قرار میدهند. یک فیلتر غیرخطی است و سعی میکند لبه‌ها را هموار کند. نحوه کارکرد این فیلتر به اینگونه است که مانند فیلتر گوسی یک میانگین وزن دار از همسایگی را محاسبه میکند اما وزنی به هر پیکسل در همسایگی میدهد به 2 عامل بستگی دارد: اول به فاصله مکانی (spatial) یعنی اینکه یک همسایه چه قدر با پیکسل مرکزی فاصله دارد (مانند فیلتر گوسی پیکسل های نزدیک به مرکز وزن بیشتری میگیرند) و دوم تفاوت شدت/کمیت یعنی اینکه همسایه‌های مختلف مقدار کمیت آنها چه قدر فرق میکند نسبت به کمیت پیکسل مرکزی ( یعنی پیکسل‌هایی که کمیت یا شدت مشابه داشته باشند وزن بیشتری میگیرند). این فیلتر 3 پارامتر دارد: اول d که بیانگر قطر همسایگی پیکسل‌ها هست، طبیعتا هر چه مقدار آن بزرگتر باشد، پیکسل های بیشتری شامل محاسبات میشوند و سرعت آهسته‌تر میشود. دوم sigmaColor این پارامتر سیگما را در فضای کمیت فیلتر میکند یعنی هر چه مقدار آن بیشتر باشد پیکسل‌هایی که شدت/کمیت تفاوت بزرگتری دارند شامل میانگین گیری میشوند ( یعنی لبه‌ها خیلی هموار تر میشوند و کم میشوند اگر مقدار خیلی زیاد باشد و تصویر خیلی blur میشود). سوم sigmaSpace که سیگما را در فضای مختصات فیلتر میکند، یعنی هر چه قدر مقدار آن بزرگتر باشد پیکسل‌هایی که فاصله دورتری دارند تاثیر بیشتری دارند.

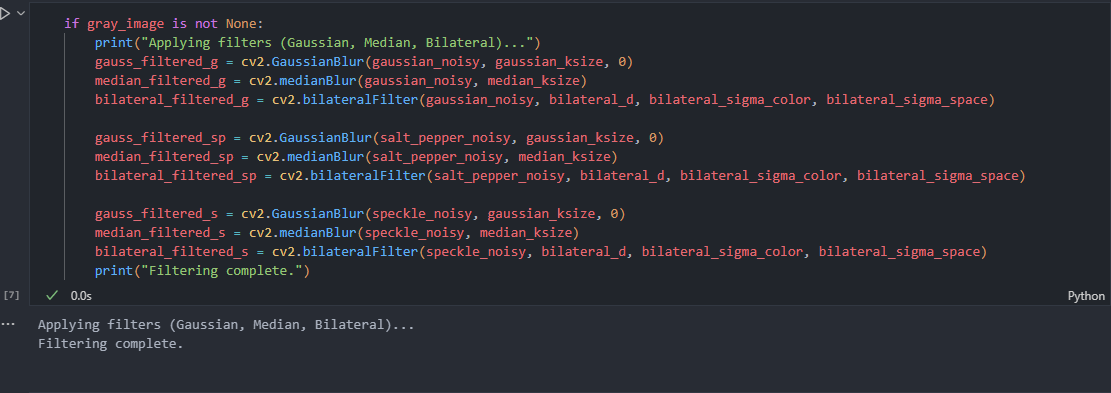
این فیلتر معمولا میتواند یک تعادل خوبی داشته باشد بین هر 2 هدف یعنی هم بتواند به خوبی نویز را کاهش دهد هم لبه‌ها را به خوبی نگه دارد. البته عیب این فیلتر این است که به طور چشمگیر از فیلتر گاوسی و میانه کند تر است. تنظیم کردن پارامتر آن میتواند خروجی را بهینه کند، همچنین در برابر نویز نمک و فلفل نسبت به فیلتر میانه کمتر کارآمد هست.

گزارش کار

تا قبل از اعمال فیلتر بقیه کد مانند قبل است و توضیحات آنها در بالاتر آمده است فقط در بخش پارامترها، پارامترهای فیلترهای گفته شده را که بالاتر توضیح دادیم اضافه میکنیم:



سپس



در این قسمت با کمک کتابخانه cv2 فیلترهایی که بالاتر توضیح دادیم را اعمال میکنیم و بعد در ادامه کد از matplotlib استفاده میکنیم و خروجی را نمایش میدهیم. هر سطر خروجی نمایش داده شده شامل یک نویز مشخص و اعمال فیلترهای مختلف بر روی آن است.

##### **خروجی**



بخش ج