به نام خدا

## عنوان

بخش سوم از تکلیف اول درس پردازش تصویر رقمی

## استاد

دکتر منصوری

## دانشجو

محمدعلی مجتهدسلیمانی

۴۰۳۳۹۰۴۵۰۴

## تاریخ

۱۴۰۴/۰۲/۵

Table of Contents

# سوال سوم

برای این سوال ۳ 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 ۳ 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 ۰ 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 ( ۰ که سیاه است و معنی فلفل را میدهد) و یا بیشینه شدت ممکن (۲۵۵ 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 ۱ منهای تعداد نمک و فلفل ها است و همین حالت برای احتمال سفید شدن و اگر هیچکدوم از این ۲ 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:

```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from skimage import data

HAS_SKIMAGE = True
%matplotlib inline
```

✓ 0.0s Python

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.

```
def add_gaussian_noise(image, mean=0, sigma=25):
    row, col = image.shape
    dtype = image.dtype
    gauss = np.random.normal(mean, sigma, (row, col))
    noisy_image = image.astype(np.float32) + gauss
    noisy_image = np.clip(noisy_image, 0, 255)
    noisy_image = noisy_image.astype(dtype)
    return noisy_image

def add_salt_and_pepper_noise(image, amount=0.04, salt_vs_pepper=0.5):
    noisy_image = np.copy(image)
    row, col = image.shape
    num_pixels = row * col
    num_salt = int(np.ceil(amount * num_pixels * salt_vs_pepper))
    coords_salt = [np.random.randint(0, i - 1, num_salt) for i in image.shape]
    noisy_image[coords_salt[0], coords_salt[1]] = 255 # White
    num_pepper = int(np.ceil(amount * num_pixels * (1.0 - salt_vs_pepper)))
    coords_pepper = [np.random.randint(0, i - 1, num_pepper) for i in image.shape]
    noisy_image[coords_pepper[0], coords_pepper[1]] = 0   # Black
    return noisy_image
```

```python
def add_speckle_noise(image, sigma=0.1):
    dtype = image.dtype
    row, col = image.shape
    noise = np.random.normal(0, sigma, (row, col))
    noisy_image = image.astype(np.float32) + image.astype(np.float32) * noise
    noisy_image = np.clip(noisy_image, 0, 255)
    noisy_image = noisy_image.astype(dtype)
    return noisy_image
```

✓ 0.0s Python

۳ تابع مختلف را برای ۳ 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 ۰ تا ۲۵۵ محدود کردیم با کمک clip و در نهایت تصویر را به dtype اصلی برگرداندیم.

در تابع salt_and_pepper ۳ 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.

```python
gaussian_sigma = 25
sp_amount = 0.05
sp_ratio = 0.5
speckle_sigma = 0.15

gray_image = None
image_source = "N/A"
image_path = "."

if HAS_SKIMAGE:
    try:
        gray_image = data.camera()
        print("Loaded standard 'cameraman' image using scikit-image.")
        image_source = 'scikit-image cameraman'
    except Exception as e:
        print(f"Error loading 'cameraman' from scikit-image: {e}")
        print("Falling back to loading from file.")
        HAS_SKIMAGE = False
```

```python
if gray_image is None and not HAS_SKIMAGE:
    if not os.path.exists(image_path):
        print(f"Error: Image file not found at '{image_path}'")
    else:
        gray_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        image_source = image_path
        if gray_image is None:
            print(f"Error: Could not load image from file '{image_path}' using OpenCV.")
        else:
            print(f"Loaded image from file: '{image_path}'")


if gray_image is None:
    print("\nERROR: Image loading failed. Cannot proceed.")

else:
    print(f"Image loaded successfully. Shape: {gray_image.shape}, Type: {gray_image.dtype}")

    plt.figure(figsize=(5, 5))
    plt.imshow(gray_image, cmap='gray')
    plt.title(f'Original Image ({image_source})')
    plt.axis('off')
    plt.show()
```

در این قسمت ابتدا پارامترهای نویز را تنظیم کردیم برای هر کدام از ۳ 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:

```
Loaded standard 'cameraman' image using scikit-image.
Image loaded successfully. Shape: (512, 512), Type: uint8
```


```python
if gray_image is not None:
    gaussian_noisy = add_gaussian_noise(gray_image, sigma=gaussian_sigma)
    salt_pepper_noisy = add_salt_and_pepper_noise(gray_image, amount=sp_amount, salt_vs_pepper=sp_ratio)
    speckle_noisy = add_speckle_noise(gray_image, sigma=speckle_sigma)
    print("Noise added successfully.")
else:
    print("Skipping noise addition because the image was not loaded.")
    gaussian_noisy, salt_pepper_noisy, speckle_noisy = None, None, None
```

Python

```
Noise added successfully.
```

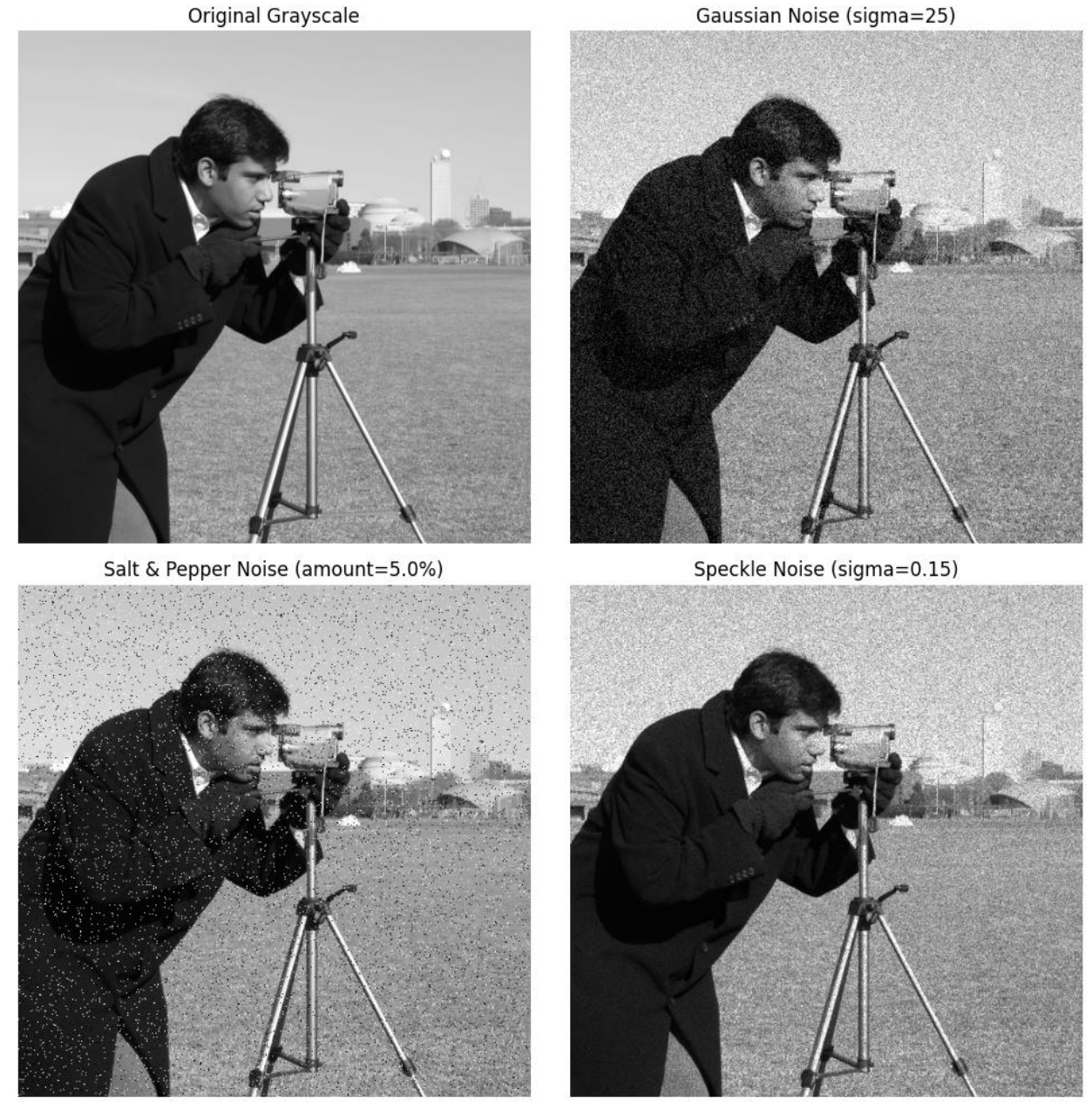
همانطور که در تصویر بالا مشخص است نویز را بر روی تصویر دریافت شده اعمال کردیم.

```python
if gray_image is not None and gaussian_noisy is not None:
    fig, axes = plt.subplots(2, 2, figsize=(10, 10))

    cmap = 'gray'

    axes[0, 0].imshow(gray_image, cmap=cmap)
    axes[0, 0].set_title('Original Grayscale')
    axes[0, 0].axis('off')
    axes[0, 1].imshow(gaussian_noisy, cmap=cmap)
    axes[0, 1].set_title(f'Gaussian Noise (sigma={gaussian_sigma})')
    axes[0, 1].axis('off')
    axes[1, 0].imshow(salt_pepper_noisy, cmap=cmap)
    axes[1, 0].set_title(f'Salt & Pepper Noise (amount={sp_amount*100:.1f}%)')
    axes[1, 0].axis('off')

    axes[1, 1].imshow(speckle_noisy, cmap=cmap)
    axes[1, 1].set_title(f'Speckle Noise (sigma={speckle_sigma})')
    axes[1, 1].axis('off')

    plt.tight_layout()
    plt.show()
```

Python

در نهایت خروجی نویزی شده را به نمایش میگذاریم.

**خروجی**



# بخش ب

در این بخش ما ابتدا ۳ 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. ۲ 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 ۲ 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 ۳ 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 ۲ 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:

```
gaussian_ksize = (5, 5)
median_ksize = 5
bilateral_d = 9
bilateral_sigma_color = 75
bilateral_sigma_space = 75
```

✓ 0.0s Python

سپس

```
if gray_image is not None:
    print("Applying filters (Gaussian, Median, Bilateral)...")
    gauss_filtered_g = cv2.GaussianBlur(gaussian_noisy, gaussian_ksize, 0)
    median_filtered_g = cv2.medianBlur(gaussian_noisy, median_ksize)
    bilateral_filtered_g = cv2.bilateralFilter(gaussian_noisy, bilateral_d, bilateral_sigma_color, bilateral_sigma_space)

    gauss_filtered_sp = cv2.GaussianBlur(salt_pepper_noisy, gaussian_ksize, 0)
    median_filtered_sp = cv2.medianBlur(salt_pepper_noisy, median_ksize)
    bilateral_filtered_sp = cv2.bilateralFilter(salt_pepper_noisy, bilateral_d, bilateral_sigma_color, bilateral_sigma_space)

    gauss_filtered_s = cv2.GaussianBlur(speckle_noisy, gaussian_ksize, 0)
    median_filtered_s = cv2.medianBlur(speckle_noisy, median_ksize)
    bilateral_filtered_s = cv2.bilateralFilter(speckle_noisy, bilateral_d, bilateral_sigma_color, bilateral_sigma_space)
    print("Filtering complete.")
```

[7] ✓ 0.0s Python

```
Applying filters (Gaussian, Median, Bilateral)...
Filtering complete.
```

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


## تحلیل

در ردیف اول تصویر نویز گاوسی به عنوان ورودی داده شده است. اول فیلتر گاوسی اعمال شده است که باعث شده است به طور چشمگیری تصویر هموار تر شوند و همچنین نواحی مانند آسمان و کت شخص واضح تر شوند. البته که باعث شده است تصویر تا حدودی blur شود و لبه های تیز در قسمت پا یا خطوط ساختمان نرم تر شوند. همچنین texture ها مانند ناحیه زمین که ظاهرا چمن است از دست رفته اند. به طور کلی کاهش نویز خوب بوده اما در جزئیات و لبه ها ضعیف عمل شده است. دوم فیلتر میانه اعمال شده است که تا حدودی نویز را کاهش داده است اما نسبت

به فیلتر گاوسی ضعیف تر بوده است. البته باعث شده لبه ها نسبت به فیلتر گاوسی بهتر حفظ شوند. مقدار قابل توجهی blurring اضافه نشده اس. در کل در لبه ها بهتر عمل شده اما نسبت به کاهش نویز و هموار سازی آن نسبت به گاوسی ضعیف تر عمل کرده. سوم فیلتر bilateral اعمال شده است که به صورت واضح نویز گاوسی را هموار تر کرده مانند خود فیلتر گاوسی عمل کرده است. همچنین به طور خیلی خوبی لبه ها را توانسته نگه دارد نسبت به فیلتر گاوسی، به صورت کلی میتوان گفت این فیلتر عملکرد خیلی خوبی در هر ۲ زمینه هموار سازی نویز یا کاهش نویز و همچنین حفظ لبه ها داشته است.

در ردیف دوم نویز نمک و فلفل به عنوان ورودی داده شده است. اول فیلتر گوسی اعمال شده است که به وضوح مشخص است نتوانسته نویز نمک و فلفل را حذف بکند و با نقاط سیاه/سفید مانند هر نقطه دیگری رفتار کرده است. تصویر کمی blur شده در نقاط سیاه یا سفید و بعضی قسمت ها که نیازی نبوده است. به طور کلی این فیلتر کارآمد نبوده است. دوم فیلتر میانه اعمال شده است که میتوان گفت به طور کامل تمام نقاط سیاه و سفید را حذف کرده است و مقادیر سیاه و سفید را تشخیص داده است و توانسته با مقادیر درست که میانه همسایه ها هست جایگزین کند همچنین لبه ها خوب حفظ شده اند، میتوان گفت بهترین انتخاب برای این نویز همین فیلتر است. سوم فیلتر bilateral اعمال شده است که نتوانسته نقاط سیاه و سفید را حذف کند در حالی که سعی کرده لبه ها را به خوبی حفظ کند در نتیجه به طور کلی این فیلتر خوبی برای این نویز نیست.

در ردیف سوم نویز speckle به عنوان ورودی داده شده است. اول فیلتر گاوسی اعمال شده است که توانسبته نویز را تا حدودی کاهش دهد اما تیزی و لبه ها از دست رفته اند. دوم فیلتر میانه اعمال شده است که تاثیر خیلی کمی روی نویز داشته است و تقریبا دانه‌دانه های نویز باقی مانده اند البته در حفظ لبه ها بهتر عمل کرده است به طور کلی این فیلتر هم برای این نویز مناسب نیست. سوم فیلتر bilateral اعمال شده است که توانسته نویز speckle را به خوبی کاهش دهد و ظاهر

هموار تری تصویر بگیرد همچنین توانسته است لبه ها را به خوبی حفظ کند و مرز های المان تیز مانده است. به طور کلی این فیلتر برای این نوع نویز بسیار مناسب است نسبت به بقیه.

## بخش ج

برای تشخیص نوع نویز و انتخاب فیلتر مناسب بر اساس آنچه در بخش ب از ویژگی‌های فیلترها گفتیم روشی که در ادامه ارائه میکنیم به همراه پیاده‌سازی میتواند به هدف برسد. توضیحات پیاده‌سازی در قسمت گزارش کار آمده است.

در گام اول باید نوع نویز را مشخص کنیم برای اینکار میتوانیم از هیورستیک‌ها یا روش های ابتکاری کمک بگیریم. سعی میکنیم از ویژگی‌های آماری استفاده کنیم تا نویز اعمال شده را پیدا کنیم. اگر نویز از نوع **salt& pepper** باشد باید به دنبال تعداد مشخص و زیادی پیکسل هایی با مقادیر کمینه و بیشینه یعنی ۰ یا ۲۵۵ روبرو بشیم. اگر نویز از این نوع نباشد، باید یک تفاوت هایی بین نویز افزایشی که برای گاوسی است و نویز ضربی که برای **speckle** است پیدا کنیم. یک تفاوت کلیدی بین این دو اینکه واریانس نویز **speckle** با کمیت/شدت تصویر افزایش پیدا میکند در حالی که واریانس نویز گوسی تقریبا ثابت است. اینکار را میتوانیم با محاسبه واریانس در دسته های محلی تصویر انجام بدهیم و ارتباط بین دسته‌های مختلف را ببینیم.

در گام دوم باید به سراغ انتخاب فیلتر برویم همانطور که قبلا توضیح دادیم اگر نویز نمک و فلفل مشاهده شد فیلتر میانه بهترین گزینه است. اگر نویز **speckle** مشاهده شد یعنی واریانس با میانگین ارتباط داشت فیلتر **bilateral** گزینه بهتری است زیرا لبه ها را بهتر حفظ میکند در حالی که دارد نویز ضربی را هموار میکند البته در این حالت میتوانیم از فیلتر گاوسی نیز استفاده کنیم. اگر نویز گاوسی تشخیص داده شد ( یعنی واریانس ارتباطی با میانگین ندارد همچنین **salt and pepper** نیست) فیلتر گاوسی بهترین گزینه است البته که میتوان از فیلتر **bilateral** نیز استفاده کرد اگر میخواهیم لبه ها را نگه داریم.

در گام سوم این موارد گفته شده را پیاده سازی میکنیم که به عنوان ورودی یک تصویر نویزی میگیریم و موارد گفته شده را اعمال میکنیم و نویز را تشخیص میدهیم و بعد فیلتر را تخصیص میدهیم.

## گزارش کار

بخش های عمده از کد مانند قسمت قبلی است و از توضیح آن موارد صرف نظر میکنیم.

```python
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from skimage import data
from scipy import stats

HAS_SKIMAGE = True
%matplotlib inline
```

✓ 5.6s Python

تمام این قسمت مانند قبل است اما از کتابخانه **scipy** استفاده کردیم تا بتوانیم رابطه **pearson** را محاسبه کنیم که برای موارد ابتکاری گفته شده در بالا قرار است مورد استفاده قرار بگیرد.

### تابع تشخیص نویز و اعمال فیلتر

```python
def detect_noise_and_filter(noisy_image,
                            sp_threshold=0.01,
                            speckle_corr_threshold=0.5,
                            block_size=16,
                            gaussian_ksize=(5,5),
                            median_ksize=5,
                            bilateral_d=9,
                            bilateral_sigma_color=75,
                            bilateral_sigma_space=75):
    img_h, img_w = noisy_image.shape
    num_pixels = img_h * img_w

    num_salt = np.sum(noisy_image == 255)
    num_pepper = np.sum(noisy_image == 0)
    sp_percentage = (num_salt + num_pepper) / num_pixels

    if sp_percentage > sp_threshold:
        noise_type = "Salt & Pepper"
        filter_name = "Median Filter"
        print(f"Detected {noise_type} ({(sp_percentage*100):.2f}% extreme pixels). Applying {filter_name}...")
        filtered_image = cv2.medianBlur(noisy_image, median_ksize)
        return noise_type, filter_name, filtered_image

    local_means = []
    local_vars = []
    block_size = min(block_size, img_h, img_w)
```

```
    for r in range(0, img_h - block_size + 1, block_size):
        for c in range(0, img_w - block_size + 1, block_size):
            block = noisy_image[r:r+block_size, c:c+block_size]
            mean = np.mean(block)
            var = np.var(block)
            if var > 1e-4:
                local_means.append(mean)
                local_vars.append(var)

    if not local_means:
        print("Could not compute local statistics. Defaulting to Gaussian Filter.")
        noise_type = "Undetermined (Defaulting Gaussian)"
        filter_name = "Gaussian Filter"
        filtered_image = cv2.GaussianBlur(noisy_image, gaussian_ksize, 0)
        return noise_type, filter_name, filtered_image

    if np.std(local_means) > 1e-4 and np.std(local_vars) > 1e-4:
        correlation, _ = stats.pearsonr(local_means, local_vars)
    else:
        correlation = 0
```

```
    if correlation > speckle_corr_threshold:
        noise_type = "Speckle (Likely)"
        filter_name = "Bilateral Filter"
        print(f"Detected {noise_type} (Mean/Var Correlation: {correlation:.3f}). Applying {filter_name}...")
        filtered_image = cv2.bilateralFilter(noisy_image, bilateral_d, bilateral_sigma_color, bilateral_sigma_space)
    else:
        noise_type = "Gaussian (Likely)"
        filter_name = "Gaussian Filter"
        print(f"Detected {noise_type} (Mean/Var Correlation: {correlation:.3f}). Applying {filter_name}...")
        filtered_image = cv2.GaussianBlur(noisy_image, gaussian_ksize, 0)

    return noise_type, filter_name, filtered_image
```

✓ 0.0s Python

آرگومان های تابع تعریف شده به این شکل خواهند بود: از **sp_threshold** برای درصد کمینه پیکسل ها برای تشخیص نویز نمک و فلفل استفاده میکنیم. از **speckle_corr_threshold** برای کمینه رابطه بین پیکسل ها برای تشخیص **speckle** استفاده میکنیم همانطور که در بالا توضیح دادیم. از **block_size** برای اندازه سایز های بلاک برای گرفتن آمار های محلی استفاده میکنیم که گفتیم قرار است بر اساس اون دسته ها که با هم ارتباط داشتند فرق بین **speckle** و گوسی را تشخیص بدهیم. **gaussian_ksize** و ... پارامتر های فیلتر ها هستند که در بخش ب به مفصل توضیح دادیم.

در این تابع ورودی نویزی را میگیریم و ابتدا با چک کردن مقادیر ۲۵۵ و ۰ سعی میکنیم ببینیم آیا نویز اعمال شده **salt & pepper** است یا خیر اگر میزان بیشتر از آستانه معرفی شده در آستانه باشد نویز نمک و فلفل تشخیص داده میشود و فیلتر میانه/**median** برای آن در نظر گرفته میشود.

اگر نویز نمک و فلفل نباشد باید میانگین محلی و واریانس را در هر بلاک که در آرگومان مشخص کردیم محاسبه بکنیم و بعد در ادامه با کمک کتابخانه **scipy** میایم و رابطه **pearson** را بین میانگین محلی و واریانس محلی محاسبه میکنیم و بر اساس آن بین دو نویز گاوسی و **speckle** تصمیم گیری میکنیم. اگر نویز **speckle** بیشتر یعنی رابطه محاسبه شده از آستانه تعریف شده در آرگومان بیشتر باشد از فیلتر **bilateral** استفاده میکنیم.

اگر نباشد همانطور که گفتیم نویز گاوسی است پس بهتر از فیلتر گاوسی استفاده بکنیم.

بقیه قسمت ها مانند قبل است.

### پارامترهای جدید

```
sp_detect_threshold = 0.01
speckle_detect_corr = 0.4
local_stat_block_size = 16
```

✓ 0.0s Python

پارامتر های جدید را برای آرگومان تابع تعریف شده، تعریف میکنیم.

### اعمال تابع روی تصاویر نویزی

```
if gray_image is not None:
    print("\nApplying adaptive filtering (Detection + Selection)...")
    print("--- Processing Gaussian Noisy Image ---")
    g_detected_noise, g_selected_filter, g_adaptive_filtered = detect_noise_and_filter(
        gaussian_noisy, sp_detect_threshold, speckle_detect_corr, local_stat_block_size,
        gaussian_ksize, median_ksize, bilateral_d, bilateral_sigma_color, bilateral_sigma_space
    )
    print("\n--- Processing Salt & Pepper Noisy Image ---")
    sp_detected_noise, sp_selected_filter, sp_adaptive_filtered = detect_noise_and_filter(
        salt_pepper_noisy, sp_detect_threshold, speckle_detect_corr, local_stat_block_size,
        gaussian_ksize, median_ksize, bilateral_d, bilateral_sigma_color, bilateral_sigma_space
    )
    print("\n--- Processing Speckle Noisy Image ---")
    s_detected_noise, s_selected_filter, s_adaptive_filtered = detect_noise_and_filter(
        speckle_noisy, sp_detect_threshold, speckle_detect_corr, local_stat_block_size,
        gaussian_ksize, median_ksize, bilateral_d, bilateral_sigma_color, bilateral_sigma_space
    )
    print("\nAdaptive filtering complete.")
```

✓ 0.0s Python

```
...
    Applying adaptive filtering (Detection + Selection)...
    --- Processing Gaussian Noisy Image ---
    Detected Salt & Pepper (6.85% extreme pixels). Applying Median Filter...

    --- Processing Salt & Pepper Noisy Image ---
    Detected Salt & Pepper (4.98% extreme pixels). Applying Median Filter...

    --- Processing Speckle Noisy Image ---
    Detected Salt & Pepper (2.15% extreme pixels). Applying Median Filter...

    Adaptive filtering complete.
```

در نهایت خروجی را نمایش میدهیم.

# خروجی


خروجی آزمایشی برای ورودی تصویر نویزی speckle

Adaptive Filtering Result for Speckle Noise Input

Input: Speckle Noisy

Detected: Salt & Pepper
Selected: Median Filter