

تمرين ۳ سوال ۳.۱

چکیده

هدف از اين تمرين آشنايی کلي با نحوه احرائي فيلتر و همچنين انواع فيلتر هاي Laplacian Filter و Box Filter است.

مقدمه

در اين تمرين ابتدا با نحوه اعمال فيلتر ها آشنا مي شويم و در ادامه به بررسی فيلتر هاي Box Filter و Laplacian Filter پرداخته و به پاسخ سوالات مطرح شده مي پردازيم.
برای اعمال انواع فیلتر ها روی یک تصویر، می توان (با توجه به نوع فیلتر) از روش هایی مثل تغییر تک به تک پیکسل ها و یا محاسبه حاصل کانولوشن ماتریس فیلتر در تصویر اصلی استفاده کرد. در این تمرين برای Box Filter و Laplacian Filter از روش کانولوشن استفاده کرده (برای اعمال کانولوشن ازتابع آماده پیدا شده در اینترنت استفاده کردم)، و در تمرين های آينده با نحوه تغییر تک به تک پیکسل ها برای Box Filter آشنا مي شويم.

:Box Filter

Blur Filter یا Average Filter یا Box Filter است که باعث تار و Blur شدن تصویر می شود. برای اعمال این فیلتر می توان از ماتریس یک مانند $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ استفاده کرد. (باید ماتریس نرمالایز این فیلتر را بر تصویر اعمال کرد چراکه این کار برای نگه داشتن مقادیر پیکسل ها بین ۰ تا ۲۵۵ مورد نیاز است) این فیلتر که یک نوع فیلتر smooth کننده تصویر است، برای حذف برخی از نویز ها مورد استفاده قرار گرفته که باعث نزدیک کردن intensity پیکسل های همسایه می شود و با این روش نویز های خاصی را می تواند کاهش داده یا حذف کند. (به علت اعمال فیلتر به صورت پنجره ای، نیاز به zero padding داشت).

- پاسخ به سوال ۳.۱.۱: از مشکلات Box Filter می توان به این مورد اشاره کرد که اگر پیکسلی در اطراف همسایگی پیکسل های موجود در پنجره تعیین شده باشد که مقدار آن با مقادیر بقیه پیکسل های داخل پنجره بسیار متفاوت باشد، به علت محاسبه میانگین برای اعمال این فیلتر، مقدار نماینده این پنجره بسیار متفاوت تر از مقادیر پیکسل های عادی بدست می آید چراکه از این پیکسل متفاوت می تواند نویز پذیری بسیار بالایی داشته باشد.
مشکل دیگری که می توان به آن اشاره کرد این است که در اطراف لبه های sharp توان انتظار لبه های sharp را داشت (که به علت خاصیت اصلی آن یعنی Smoothing Filter بودن این مورد بدیهی است اما در مواردی که edge sharp مورد نیاز است نمی توان به این فیلتر اعتماد کرد).
(برای پاسخ به این سوال از [این لینک](#) استفاده شده است.)

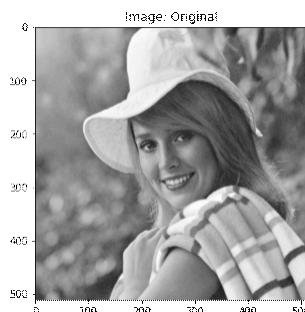
- پاسخ به سوال ۳.۱.۲: ویژگی های بد این فیلتر با اعمال فیلتر به تعداد دفعات ممتد می تواند بهبود یابد. همانطور که گفتیم این فیلتر از مقادیر بسیار پرت خیلی نویز پذیر است اما با اعمال چند باره فیلتر روی نتیجه قبلی می توان مشاهده کرد که این تفاوت مقدار پیکسل های همسایه با پیکسلی که مقدار خیلی پرت داشت کمتر و کمتر شده و باعث بهبود تصویر می شود. (می توان نتیجه این سوال را در قسمت نتیجه گیری به صورت واضح تری مشاهده کرد).

: این فیلتر از انواع فیلتر های Sharp کننده بوده که با اعمال آن می توان لبه های واضح تر و sharp تری را انتظار داشت اما با اعمال این فیلتر به صورت متداوم می توان به تصویر پر از نویز رسید. برای اعمال این فیلتر نیز می توان از

$$\text{ماتریس استفاده کرد.} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

شرح نتایج

تصویر اصلی به صورت زیر است و در ادامه به بررسی نتیجه هر سوال به صورت جداگانه می پردازیم:



- مشاهده نتیجه سوال ۳.۱.۳: در این سوال به بررسی اعمال فیلتر Box Filter به دفعات متعدد می پردازیم.

Image: Box Filtered (WindowSize = 3, rounds = 1) Image: Box Filtered (WindowSize = 3, rounds = 2)

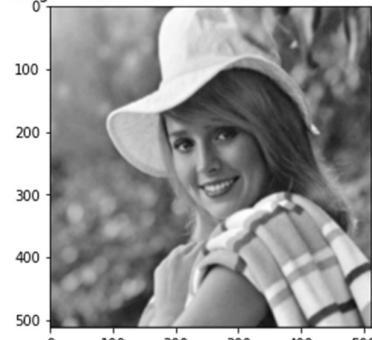
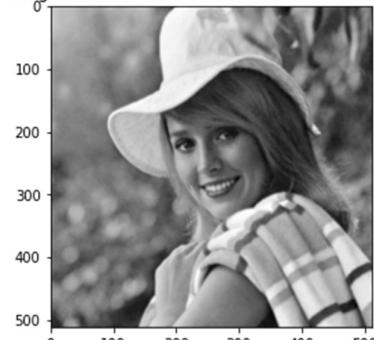
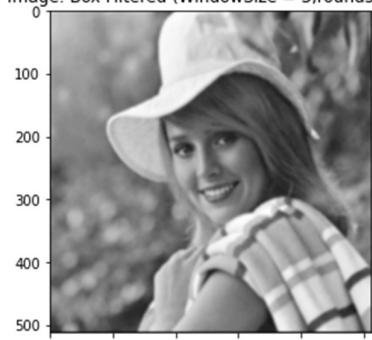
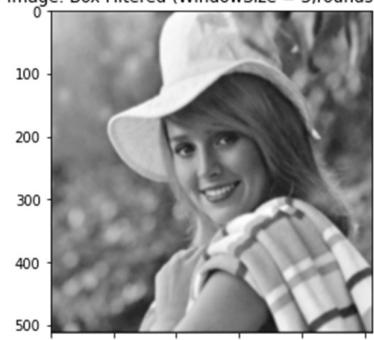


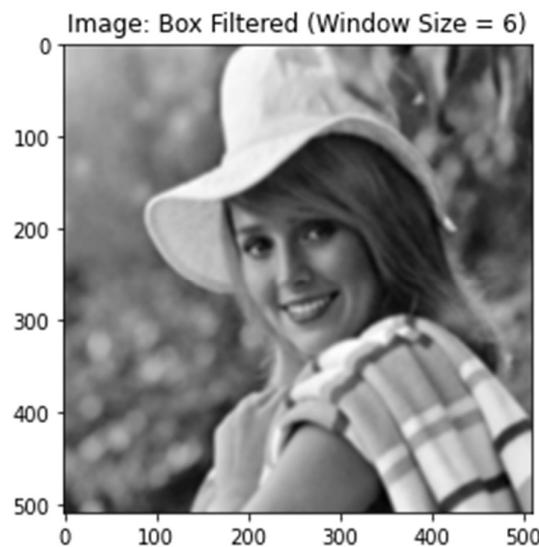
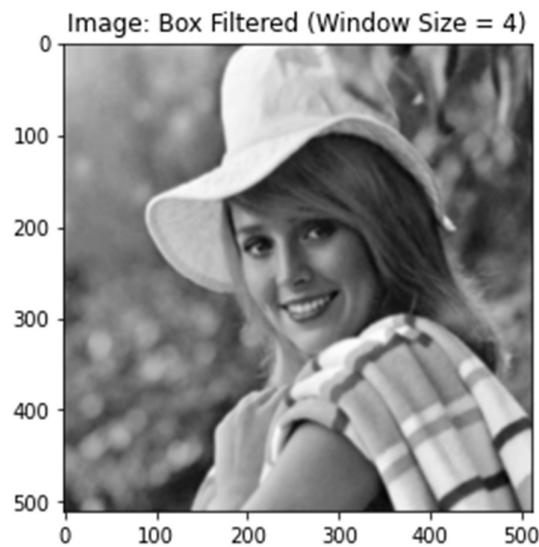
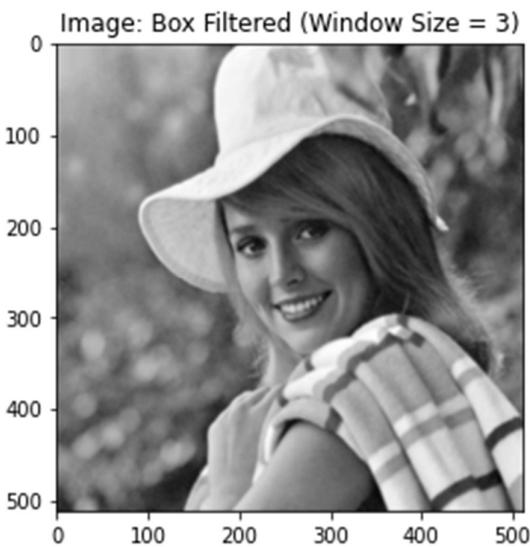
Image: Box Filtered (WindowSize = 3, rounds = 3) Image: Box Filtered (WindowSize = 3, rounds = 4)

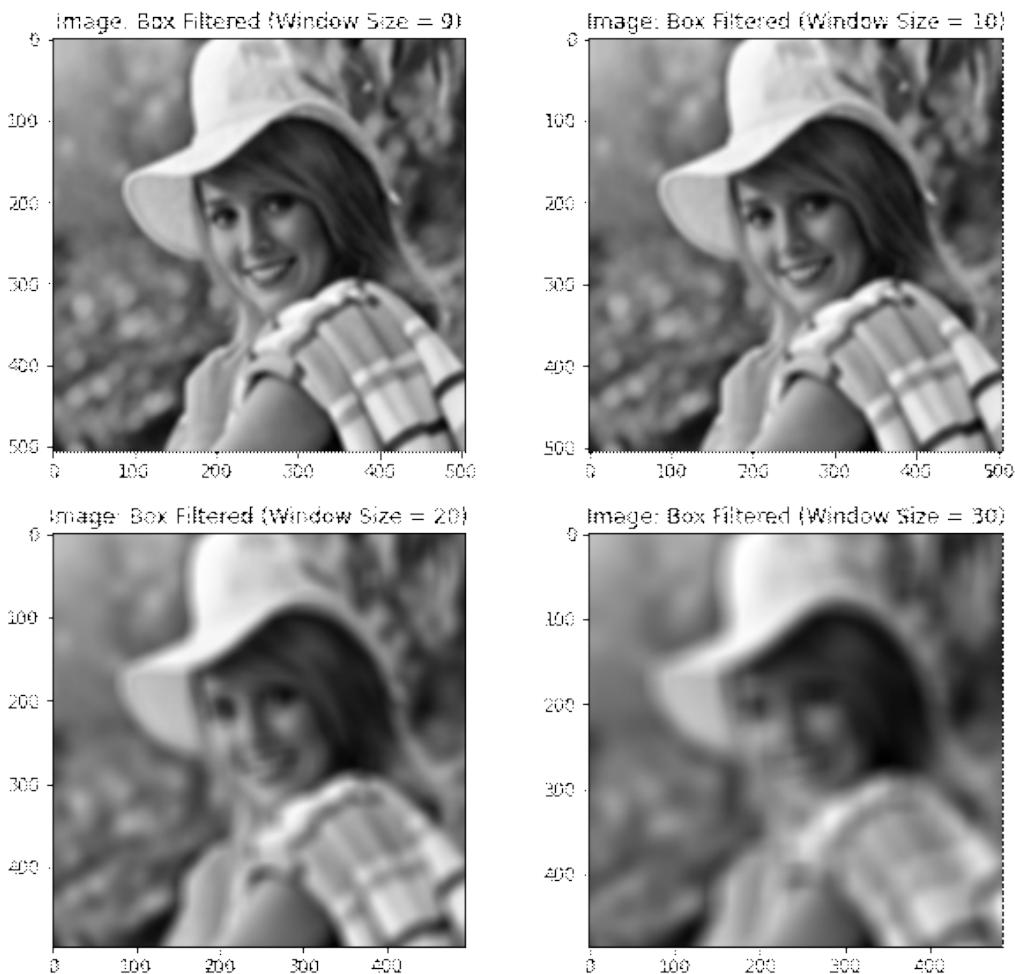




می توان مشاهده کرد که در اطراف قسمت هایی که تفاوت رنگ مشهودی دارند، مانند سایه موها روی گونه، پس از اعمال فیلتر به صورت متعدد رنگ ها نسبت به مرتبه قبلی متعادل تر شده اند.

- مشاهده نتیجه سوال ۳.۱.۴: در این سوال به بررسی اعمال فیلتر Box Filter با سایز پنجره متفاوت می پردازیم.

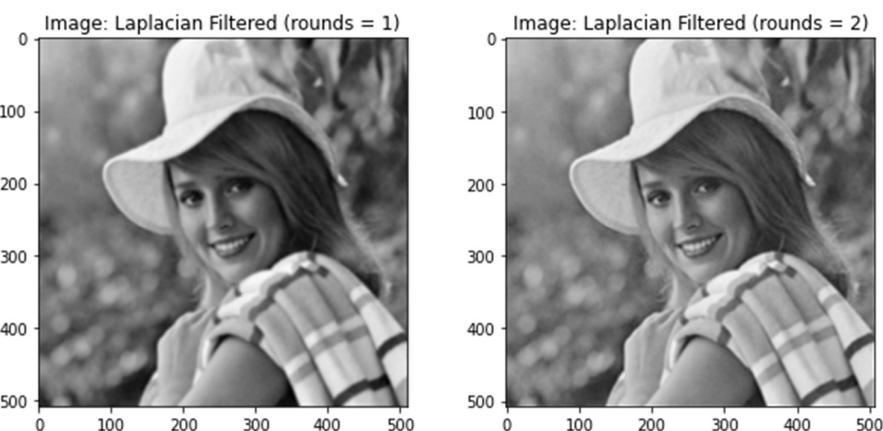


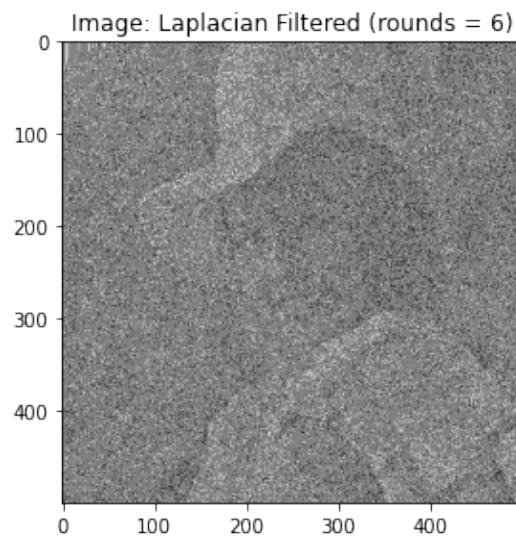
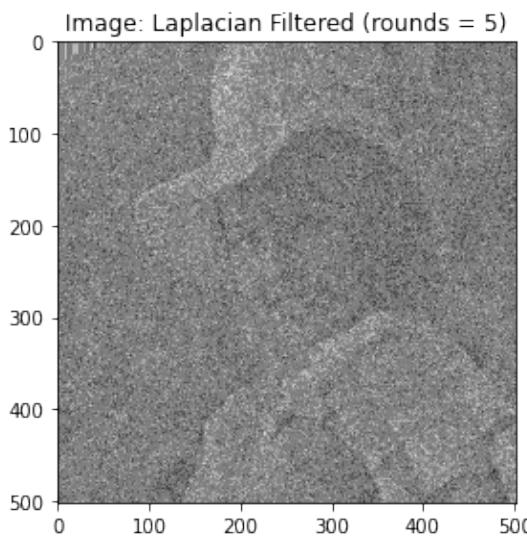
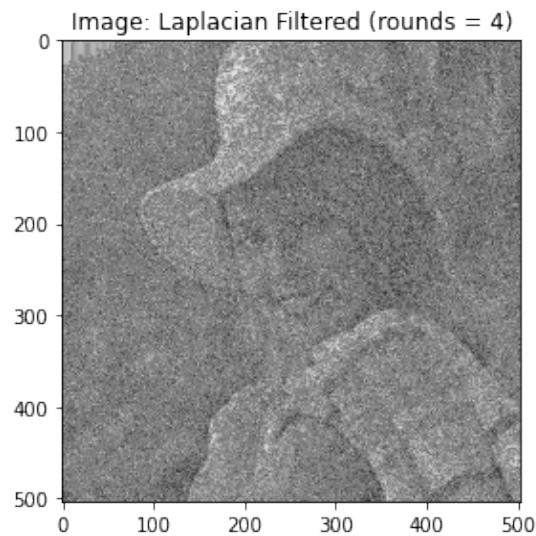
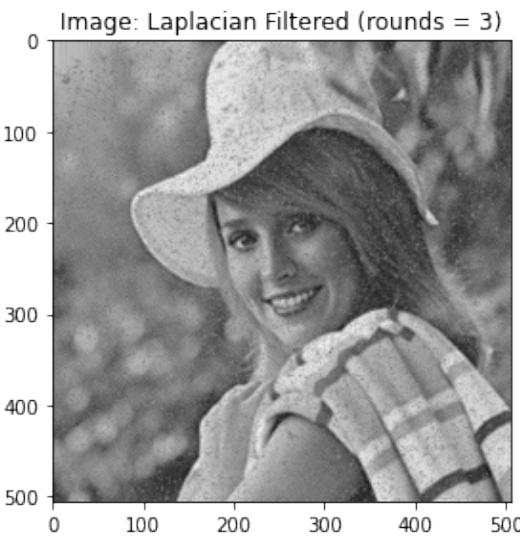


می توان مشاهده کرد که هرچه سایز پنجره را بزرگتر در نظر بگیریم، به علت دخیل بودن تعداد پیکسل های بیشتر در محاسبه میانگین، تصویر تار تر شده و پس از مدتی غیر قابل تشخیص می شود.

- مشاهده نتیجه سوال ۳.۱.۵: به نظر من بهترین میزان Smooth شدگی نسبت به سایز پنجره، پنجره با سایز ۴ است چرا که نسبت به پنجره سایز ۳ edge های خیلی sharp ندارد و رنگ ها حالت طبیعی تر و نزدیکتری دارند و همزمان نسبت به پنجره سایز ۵ خیلی از جزئیات مثل خطوط نشان دهنده دندان ها حفظ شده اند.

- مشاهده نتیجه سوال ۳.۱.۶: در این سوال به بررسی اعمال فیلتر Laplacian Filter به دفعات متعدد می پردازیم.





این فیلتر در تلاش است تا لبه های sharp تری در اختیارمان قرار دهد اما همانطور که دیده می شود، پس از مدتی تصویر غیر قابل تشخیص شده و پر از نویز می شود که علت آن sharp شدن نویز هاست. (همزمان تنها قسمتی که از تصویر باقی مانده و قابل مشاهده است edge های آن هستند).

ضمیمه

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
from sklearn.metrics import mean_squared_error

img = cv2.imread('Elaine.bmp', 0)

def convolve2D(image, kernel, padding=0, strides=1):
    kernel = np.flipud(np.fliplr(kernel))
```

```

xKernShape = kernel.shape[0]
yKernShape = kernel.shape[1]
xImgShape = image.shape[0]
yImgShape = image.shape[1]

xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
output = np.zeros((xOutput, yOutput))

if padding != 0:
    imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
    imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
else:
    imagePadded = image

for y in range(imagePadded.shape[1]):
    if y > imagePadded.shape[1] - yKernShape:
        break
    if y % strides == 0:
        for x in range(imagePadded.shape[0]):
            if x > imagePadded.shape[0] - xKernShape:
                break
            try:
                if x % strides == 0:
                    output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
            except:
                break

output[output>255]=255
output[output<0]=0

return output

```

```

def box_filter (image, window_size, many_times=0):
    filter_matrix = np.ones((window_size, window_size))
    filter_matrix = filter_matrix / (window_size * window_size)

    if many_times:
        box_filtered_image = image

        for round in range(many_times):
            box_filtered_image = convolve2D(box_filtered_image, filter_matrix, 1)
    else:
        box_filtered_image = convolve2D(image, filter_matrix, 1)
    return box_filtered_image.astype(int)

```

```

# 3.1.3

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

filter_rounds = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

fig, plot = plt.subplots(int(len(filter_rounds) / 2), 2, figsize = (10,
5 * int(len(filter_rounds) / 2)))

window_size = 3

for filter_round in range(0, len(filter_rounds), 2):
    plot[int(filter_round / 2)][0].imshow(box_filter(img, window_size, fi
lter_rounds[filter_round]), cmap='gray')
    plot[int(filter_round / 2)][0].set_title("Image: Box Filtered (Window
Size = " + str(window_size) + ", rounds = " + str(filter_rounds[filter_r
ound]) + ")")

    plot[int(filter_round / 2)][1].imshow(box_filter(img, window_size, fi
lter_rounds[filter_round + 1]), cmap='gray')
    plot[int(filter_round / 2)][1].set_title("Image: Box Filtered (Window
Size = " + str(window_size) + ", rounds = " + str(filter_rounds[filter_r
ound + 1]) + ")")

q 3 1 3 output = box filter(img, 3, 5)

```

```

# 3.1.4

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

window_sizes = [3, 4, 5, 6, 7, 8, 9, 10, 20, 30]

fig, plot = plt.subplots(int(len(window_sizes) / 2), 2, figsize = (10,
5 * int(len(window_sizes) / 2)))

for window_size in range(0, len(window_sizes), 2):
    plot[int(window_size / 2)][0].imshow(box_filter(img, window_sizes[win
dow_size]), cmap='gray')
    plot[int(window_size / 2)][0].set_title("Image: Box Filtered (Window
Size = " + str(window_sizes[window_size]) + ")")

```

```
    plot[int(window_size / 2)][1].imshow(box_filter(img, window_sizes[window_size + 1]), cmap='gray')
    plot[int(window_size / 2)][1].set_title("Image: Box Filtered (Window Size = " + str(window_sizes[window_size + 1]) + ")")
```

```
def laplacian_filter (image, many_times=0):
    filter_matrix = np.array([(0, -1, 0), (-1, 5, -1), (0, -1, 0)])

    if many_times:
        laplacian_filtered_image = image

        for round in range(many_times):
            laplacian_filtered_image = convolve2D(laplacian_filtered_image, filter_matrix)
    else:
        laplacian_filtered_image = convolve2D(image, filter_matrix)

    return laplacian_filtered_image.astype(int)
```

```
# 3.1.6
fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

filter_rounds = [1, 2, 3, 4, 5, 6]

fig, plot = plt.subplots(int(len(filter_rounds) / 2), 2, figsize = (10, 5 * int(len(filter_rounds) / 2)))

for filter_round in range(0, len(filter_rounds), 2):
    plot[int(filter_round / 2)][0].imshow(laplacian_filter(q_3_1_3_output, filter_rounds[filter_round]), cmap='gray')
    plot[int(filter_round / 2)][0].set_title("Image: Laplacian Filtered (rounds = " + str(filter_rounds[filter_round]) + ")")

    plot[int(filter_round / 2)][1].imshow(laplacian_filter(q_3_1_3_output, filter_rounds[filter_round + 1]), cmap='gray')
    plot[int(filter_round / 2)][1].set_title("Image: Laplacian Filtered (rounds = " + str(filter_rounds[filter_round + 1]) + ")")
```

تموین ۳ سوال ۲.۲.۱

چکیده

هدف از این تمرین آشنایی با نویز Salt and Pepper و فیلتر Median و میزان تأثیر آن برای خنثی کردن این نویز است.

مقدمه

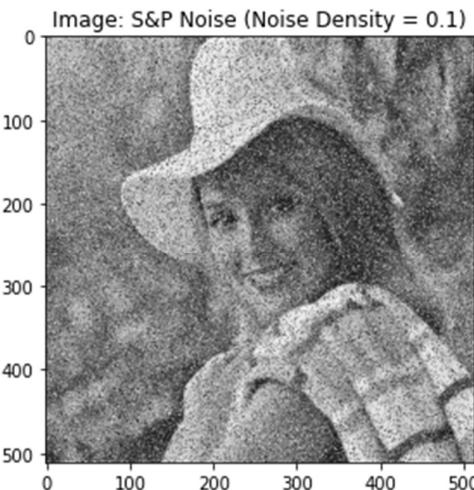
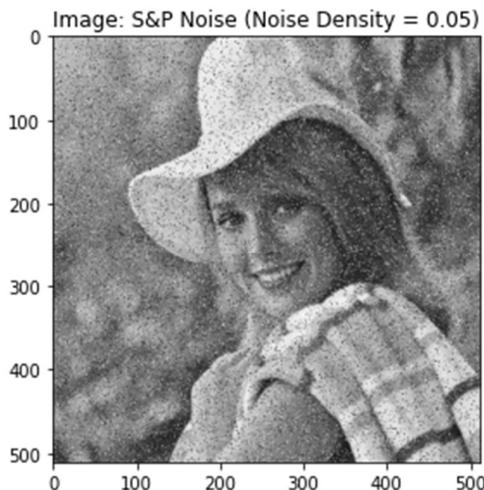
در این تمرین با نویز Salt and Pepper آشنا شده و نتیجه را در Density های مختلف بررسی می کنیم. سپس با استفاده از فیلتر Median و سایز پنجره های متفاوت، نتایج را با هم مقایسه و بهترین حالت را انتخاب می کنیم.

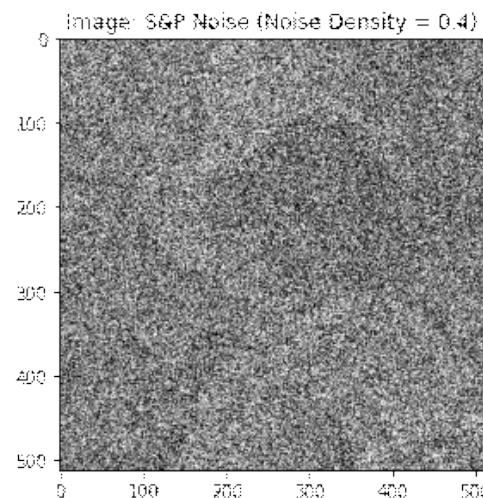
برای ساخت نویز Salt and Pepper که نویزی است که به صورت رندوم مقدار پیکسل ها را سیاه و سفید می کند، می توان از توابع آماده استفاده کرد اما برای پیاده سازیتابع می توان از تولید یک مقدار رندوم استفاده کرد و با بررسی آن و ایجاد احتمال رخداد هر مقدار بر اساس density، پیکسل ها را مقدار دهی می کنیم.

در استفاده از فیلتر Median نیز با استفاده از سایز پنجره قسمتی از تصویر را انتخاب کرده و پس از sort کردن پیکسل های داخل پنجره می توانیم Median آن پنجره را انتخاب کرده و به عنوان اولین پیکسل از بالا و سمت چپ (پیکسل [0][0]) (پنجره) قرار دهیم و تصویر را دوباره بازسازی کنیم. این فیلتر به علت اینکه نسبت به پیکسل هایی که خیلی با پیکسل های همسایه مقاومت داشته و مقدار پیکسل نماینده پنجره را نسبت به مقدار مقاومت تغییر نمی دهد برای نویز های رندوم و اتفاقی مثل Salt and Pepper مناسبند.

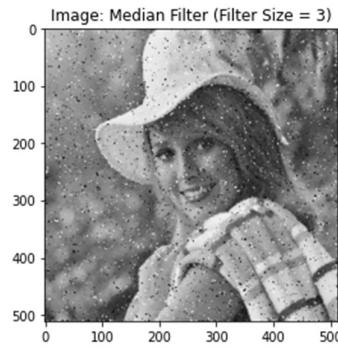
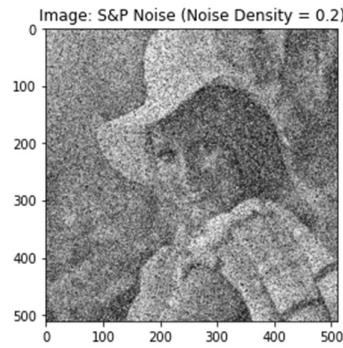
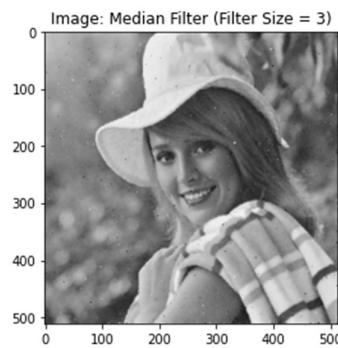
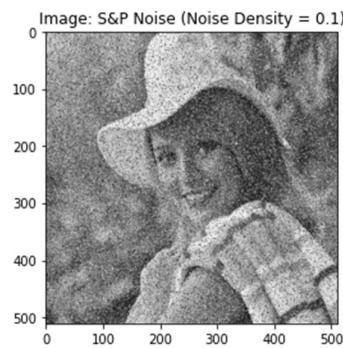
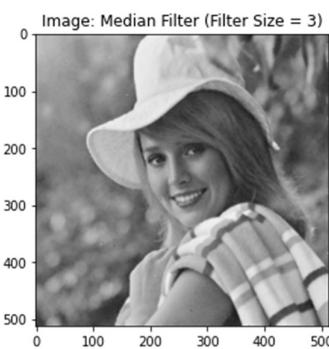
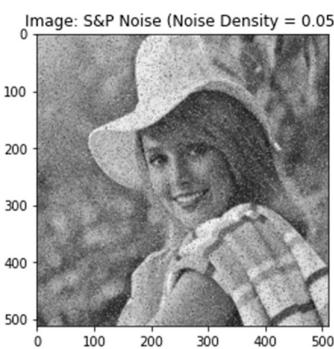
شرح نتایج

در قسمت نتایج ابتدا نتیجه ایجاد نویز با Density های مختلف را مشاهده می کنیم.





همانطور که مشاهده می شود، تصویر با افزایش میزان Noise Density غیر قابل تشخیص تر و متفاوت تر از تصویر اصلی می شود. همچنین نقاط جدید سیاه و سفید ایجاد شده در تصویر قابل مشاهده هستند.
در ادامه فیلتر Median با پنجره های متفاوت را بر این نتایج نویز اعمال می کنیم.



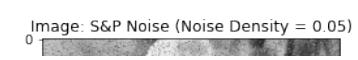
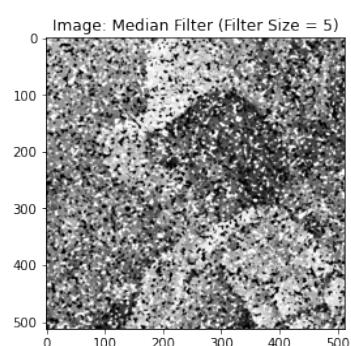
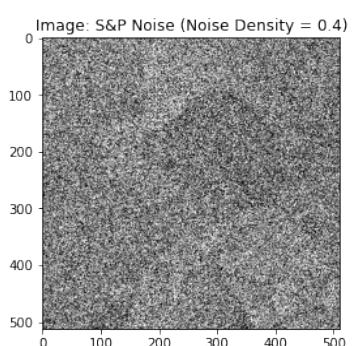
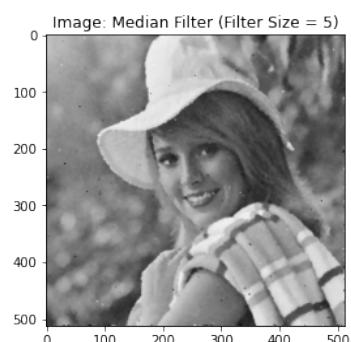
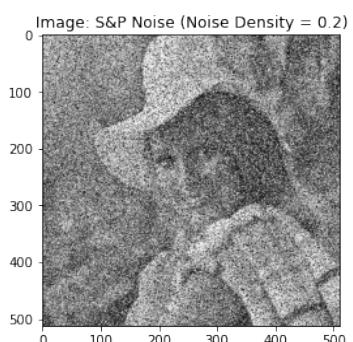
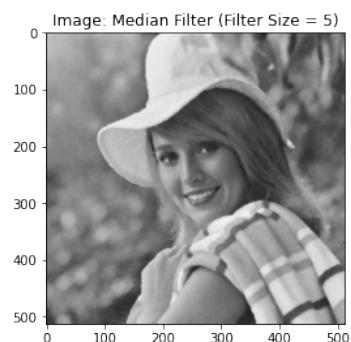
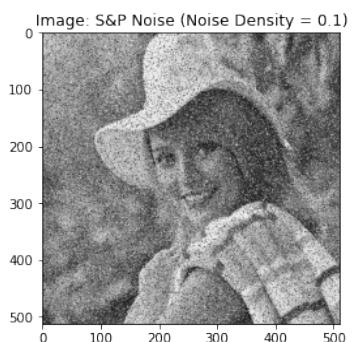
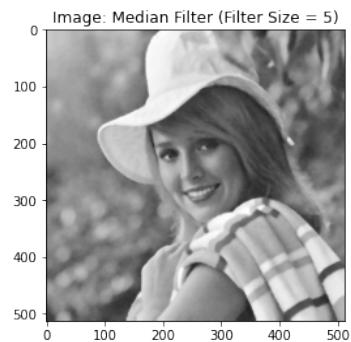
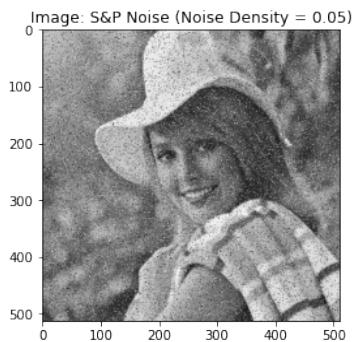
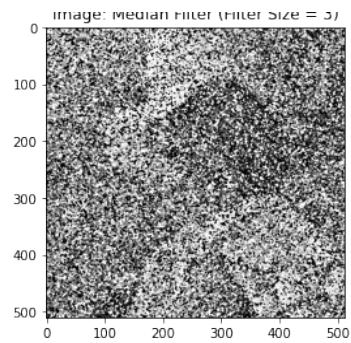
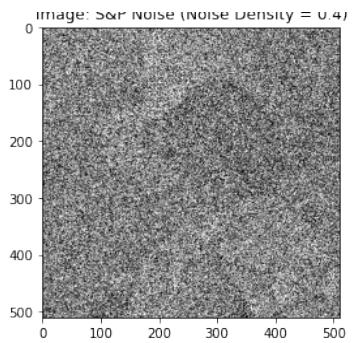


Image: S&P Noise (Noise Density = 0.05)

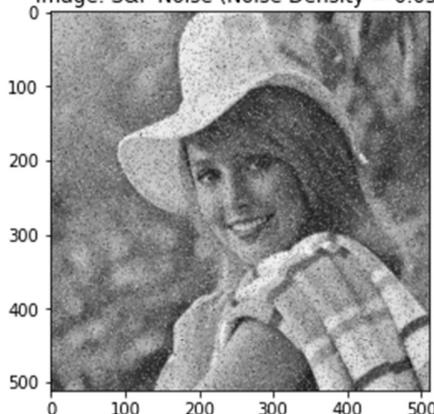


Image: Median Filter (Filter Size = 7)

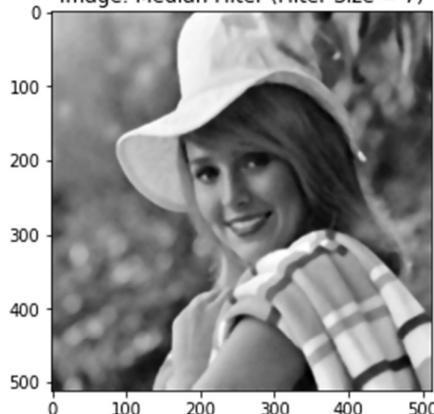


Image: S&P Noise (Noise Density = 0.1)

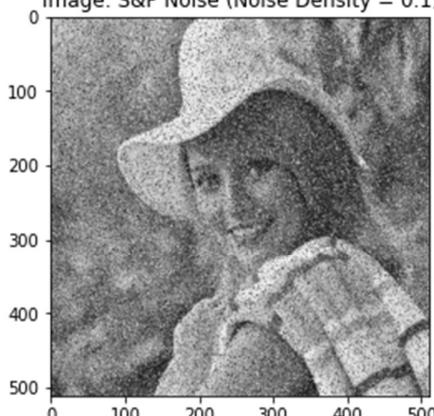


Image: Median Filter (Filter Size = 7)

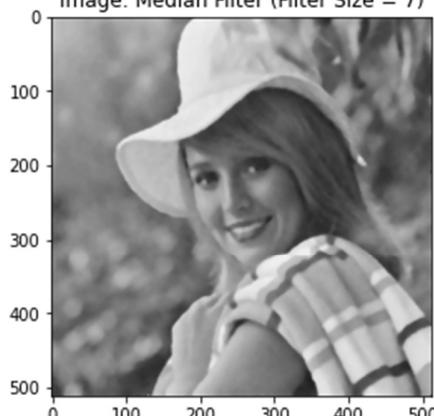


Image: S&P Noise (Noise Density = 0.2)

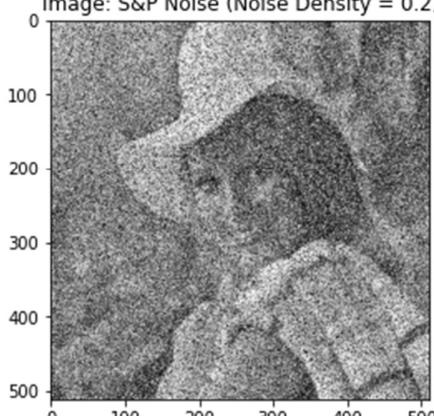


Image: Median Filter (Filter Size = 7)

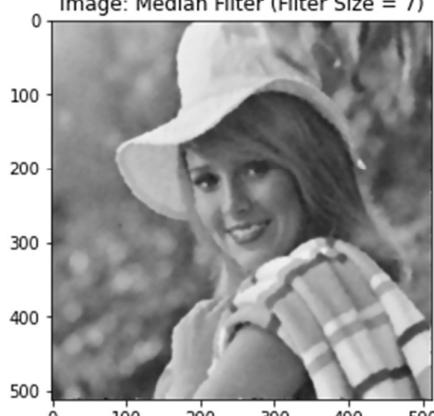


Image: S&P Noise (Noise Density = 0.4)

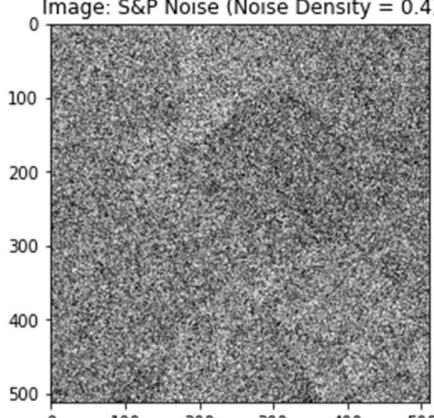


Image: Median Filter (Filter Size = 7)

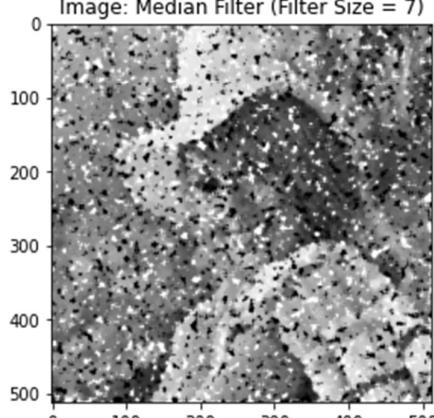


Image: S&P Noise (Noise Density = 0.05)

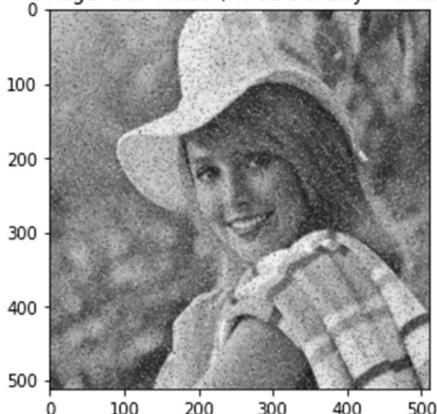


Image: Median Filter (Filter Size = 9)

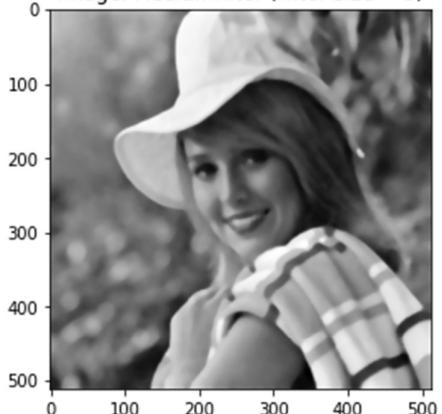


Image: S&P Noise (Noise Density = 0.1)

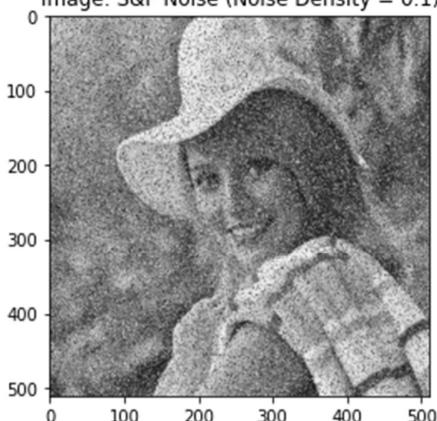


Image: Median Filter (Filter Size = 9)

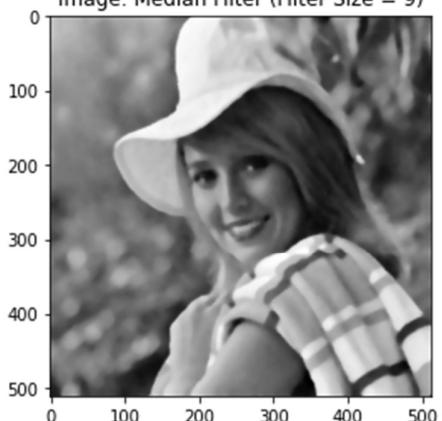


Image: S&P Noise (Noise Density = 0.2)

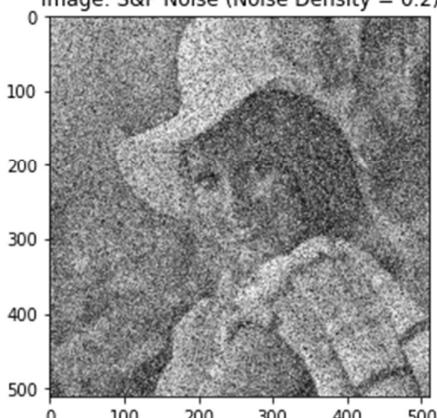


Image: Median Filter (Filter Size = 9)

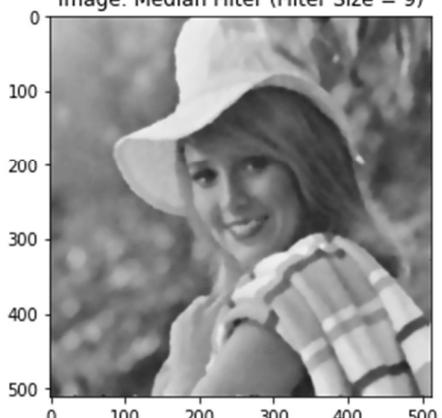


Image: S&P Noise (Noise Density = 0.4)

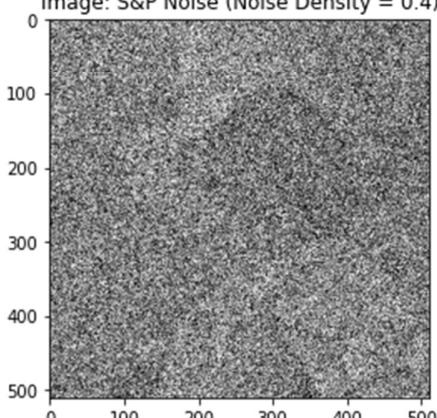
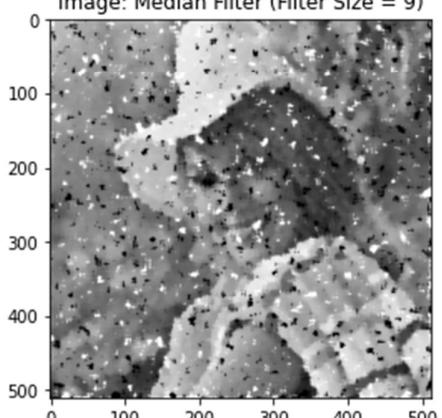
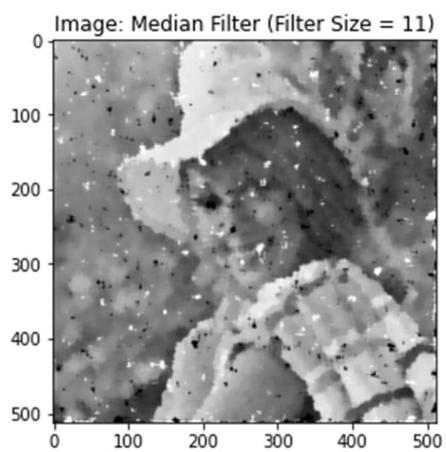
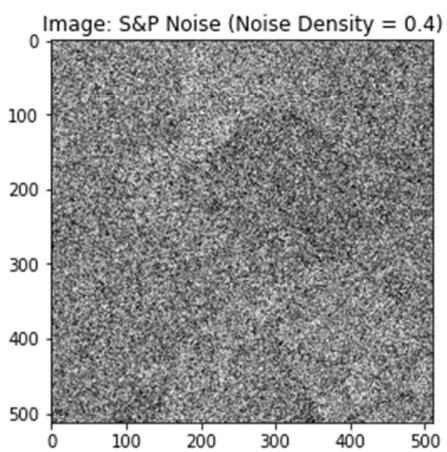
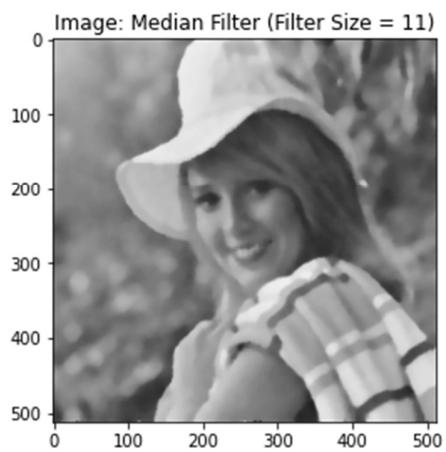
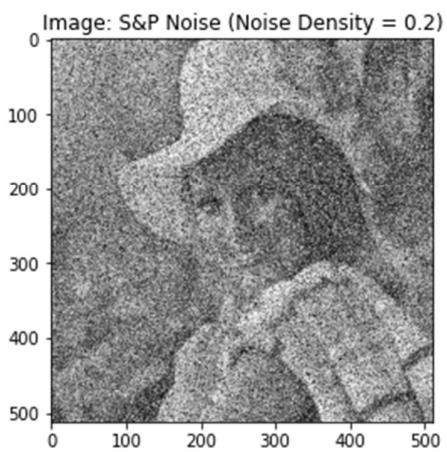
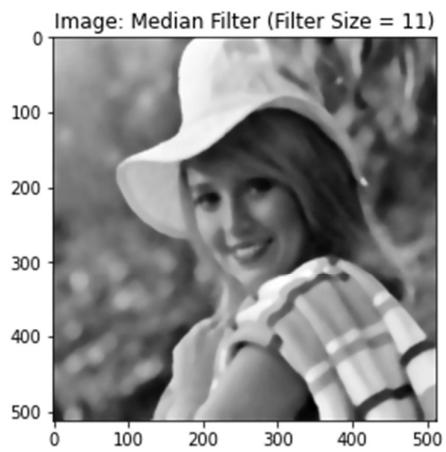
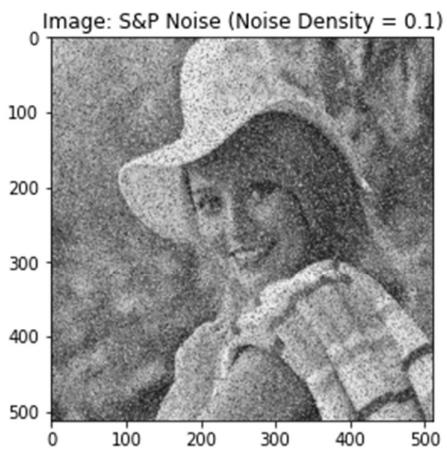
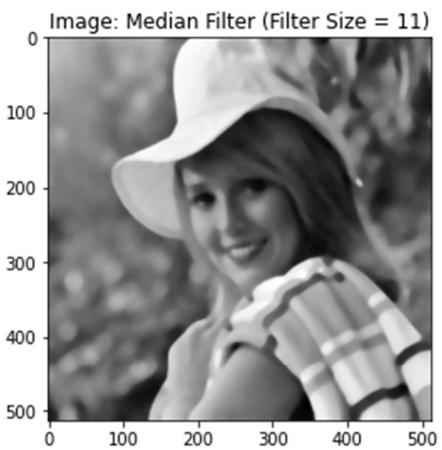
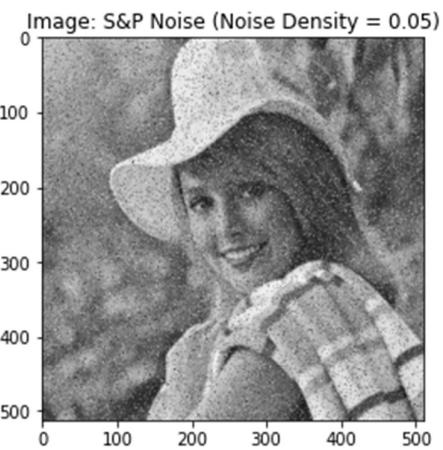


Image: Median Filter (Filter Size = 9)





همانطور که در نتایج نیز قابل مشاهده است، در یک فیلتر با سایز یکسان، هرچه میزان نویز کمتر باشد، نتیجه واضح تر و به تصویر اصلی نزدیک تر است. از طرفی در مقدار density های یکسان، هرچه سایز پنجره بزرگتر شود، تصویر حدودی تر شده و جزئیات خود را از دست می دهد اما همین موضوع در تصاویری که density های خیلی بالاتر دارد، باعث دیده شدن بهتر تصویر اصلی می شود چراکه تعداد نویز های بیشتری حذف شده و تصویر واضح تر از سایز های کوچکتر پنجره است.

برای انتخاب بهترین سایز پنجره می توان مقادیر MSE را با تصویر اصلی مقایسه کرد و بهترین سایز را کوچکترین سایز در تصاویر با نویز کمتر و بزرگترین سایز در تصاویر با نویز بیشتر است.

مقادیر MSE نیز در زیر قابل مشاهده هستند.

```
Median Filter: Filter Size = 3
----S&P Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images = 1916.5606021881104
MSE Between Original and Filtered Images = 104.17545127868652
----S&P Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images = 3664.6183824539185
MSE Between Original and Filtered Images = 145.54156589508057
----S&P Noise: Noise Density = 0.2----
MSE Between Noisy and Filtered Images = 7242.852326393127
MSE Between Original and Filtered Images = 845.3797235488892
----S&P Noise: Noise Density = 0.4----
MSE Between Noisy and Filtered Images = 16430.27618789673
MSE Between Original and Filtered Images = 9887.767101287842
```

```
Median Filter: Filter Size = 5
----S&P Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images = 2049.350136756897
MSE Between Original and Filtered Images = 230.50567722320557
----S&P Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images = 3811.2201404571533
MSE Between Original and Filtered Images = 234.4759693145752
----S&P Noise: Noise Density = 0.2----
MSE Between Noisy and Filtered Images = 7392.586150169373
MSE Between Original and Filtered Images = 270.77004528045654
----S&P Noise: Noise Density = 0.4----
MSE Between Noisy and Filtered Images = 16622.05848789215
MSE Between Original and Filtered Images = 5933.481854438782
```

```
Median Filter: Filter Size = 7
----S&P Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images = 2185.22069644928
MSE Between Original and Filtered Images = 377.9258508682251
----S&P Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images = 3937.1881647109985
MSE Between Original and Filtered Images = 380.9727144241333
----S&P Noise: Noise Density = 0.2----
MSE Between Noisy and Filtered Images = 7500.632660865784
MSE Between Original and Filtered Images = 395.19392108917236
----S&P Noise: Noise Density = 0.4----
MSE Between Noisy and Filtered Images = 15831.817255020142
MSE Between Original and Filtered Images = 3336.2706050872803
```

```
Median Filter: Filter Size = 9
```

```

----S&P Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images = 2321.177876472473
MSE Between Original and Filtered Images = 528.3937501907349
----S&P Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images = 4058.975785255432
MSE Between Original and Filtered Images = 530.1811532974243
----S&P Noise: Noise Density = 0.2----
MSE Between Noisy and Filtered Images = 7593.407160758972
MSE Between Original and Filtered Images = 541.7863416671753
----S&P Noise: Noise Density = 0.4----
MSE Between Noisy and Filtered Images = 15233.083855628967
MSE Between Original and Filtered Images = 1941.0869646072388

```

```

Median Filter: Filter Size = 11
----S&P Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images = 2451.2088441848755
MSE Between Original and Filtered Images = 673.4227304458618
----S&P Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images = 4173.283555030823
MSE Between Original and Filtered Images = 674.6237344741821
----S&P Noise: Noise Density = 0.2----
MSE Between Noisy and Filtered Images = 7673.956993103027
MSE Between Original and Filtered Images = 684.7884941101074
----S&P Noise: Noise Density = 0.4----
MSE Between Noisy and Filtered Images = 14916.336122512817
MSE Between Original and Filtered Images = 1304.396104812622

```

ضعيه

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
import random
from sklearn.metrics import mean_squared_error
from statistics import median

img = cv2.imread('Elaine.bmp',0)

@jit(nopython=True)
def salt_and_pepper_noise(image, noise_density):
    R, C = image.shape
    salt_and_pepper_noise_added_image = np.zeros((R, C))
    for r in range (R):
        for c in range (C):
            rand = random.random()
            if rand < noise_density:
                salt_and_pepper_noise_added_image[r][c] = 0
            elif rand > 1 - noise_density:
                salt_and_pepper_noise_added_image[r][c] = 255
            else:
                salt_and_pepper_noise_added_image[r][c] = image[r][c]
    return salt_and_pepper_noise_added_image

```

```

# 3.2.1 - Salt and Pepper Noise Added Images

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

noise_densities = [0.05, 0.1, 0.2, 0.4]

R, C = img.shape
salt_and_pepper_noise_added_images = np.zeros((len(noise_densities), R,
C))

fig, plot = plt.subplots(int(len(noise_densities) / 2), 2, figsize = (10, 5 * int(len(noise_densities) / 2)))

for noise_density in range(0, len(noise_densities), 2):
    salt_and_pepper_noise_added_image_1 = salt_and_pepper_noise(img, noise_densities[noise_density])
    plot[int(noise_density / 2)][0].imshow(salt_and_pepper_noise_added_image_1, cmap='gray')
    plot[int(noise_density / 2)][0].set_title("Image: S&P Noise (Noise Density = " + str(noise_densities[noise_density]) + ")")

    salt_and_pepper_noise_added_image_2 = salt_and_pepper_noise(img, noise_densities[noise_density + 1])
    plot[int(noise_density / 2)][1].imshow(salt_and_pepper_noise_added_image_2, cmap='gray')
    plot[int(noise_density / 2)][1].set_title("Image: S&P Noise (Noise Density = " + str(noise_densities[noise_density + 1]) + ")")

    salt_and_pepper_noise_added_images[noise_density, 0:R, 0:C] = salt_and_pepper_noise_added_image_1
    salt_and_pepper_noise_added_images[noise_density + 1, 0:R, 0:C] = salt_and_pepper_noise_added_image_2

def median_filter(image, filter_size):
    R, C = image.shape

    median_filtered_image = np.zeros((R, C))

    for r in range (R):
        for c in range (C):
            median_filtered_image[r][c] = median(image[r:r + filter_size, c:c + filter_size].flatten())

    return median_filtered_image

```

```

# 3.2.1 - Median Filtered Images
fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

filter_sizes = [3, 5, 7, 9, 11]

mse_noisy_and_filtered = np.zeros(len(filter_sizes) * len(noise_densities))
mse_original_and_filtered = np.zeros(len(filter_sizes) * len(noise_densities))

fig, plot = plt.subplots(len(filter_sizes) * len(noise_densities), 2, figsize = (10, 5 * len(filter_sizes) * len(noise_densities)))

for filter_size in range(len(filter_sizes)):
    for noise_density in range(len(noise_densities)):
        salt_and_pepper_noise_added_image = salt_and_pepper_noise_added_images[noise_density, 0:R, 0:C]
        plot[filter_size * len(noise_densities) + noise_density][0].imshow(salt_and_pepper_noise_added_image, cmap='gray')
        plot[filter_size * len(noise_densities) + noise_density][0].set_title("Image: S&P Noise (Noise Density = " + str(noise_densities[noise_density]) + ")")

        median_filtered_image = median_filter(salt_and_pepper_noise_added_image, filter_sizes[filter_size])
        plot[filter_size * len(noise_densities) + noise_density][1].imshow(median_filtered_image, cmap='gray')
        plot[filter_size * len(noise_densities) + noise_density][1].set_title("Image: Median Filter (Filter Size = " + str(filter_sizes[filter_size]) + ")")

        mse_noisy_and_filtered[filter_size * len(noise_densities) + noise_density] = mean_squared_error(salt_and_pepper_noise_added_image, median_filtered_image)
        mse_original_and_filtered[filter_size * len(noise_densities) + noise_density] = mean_squared_error(img, median_filtered_image)

```

```

# 3.2.1 - MSE Report

for filter_size in range(len(filter_sizes)):
    print("Median Filter: Filter Size = " + str(filter_sizes[filter_size]))
    for noise_density in range(len(noise_densities)):

```

```
    print("----\nS&P Noise: Noise Density = " + str(noise_densities[noise_density]) + "----")\n    print("MSE Between Noisy and Filtered Images = " + str(mse_noisy_and_filtered[filter_size * 4 + noise_density]))\n    print("MSE Between Original and Filtered Images = " + str(mse_original_and_filtered[filter_size * 4 + noise_density]))\n    print("\n")
```

تمرين ۳ سوال ۳.۲.۲

چکیده

در این تمرين با نویز Gaussian آشنا شده و سعی می کنیم تصویر را توسط فیلتر های Mean و Median از این نویز پاک کنیم.

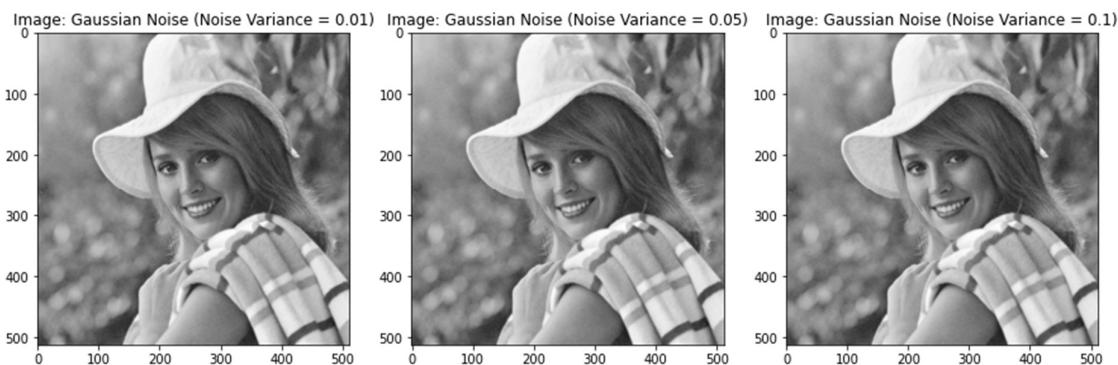
مقدمه

نویز Gaussian نویزی است که با توجه به میزان واریانس نویز که در آن تعریف می شود، مقادیر رندومی در این بازه تعریف کرده و یک آرایه به اندازه سایز تصویر از مقادیر رندوم می سازد و در نهایت با تصویر اصلی جمع می کند. در این حالت، بر خلاف نویز Salt and Pepper که اختلاف یک پیکسل با پیکسل های همسایه می توانست به صورت ناگهانی تغییرات خیلی زیادی داشته باشد، تغییرات پیکسل ها نسبت به همسایه ها به علت جمع شدن یا پیکسل اصلی و نه جایگزینی آن، کمتر است و بنابراین فیلتر میانگین می تواند تأثیر مناسبی داشته باشد.

فیلتر Average یا Box فیلتری است که در سوال اول نیز بررسی کردیم و در اینجا حالت تغییر دادن پیکسل به پیکسل را بررسی می کنیم. در این حالت با سایز پنجره انتخابی، پیکسل ها را پنجره پنجره انتخاب کرده و میانگین آن ها را محاسبه کرده و در نهایت به جای پیکسل بالا سمت چپ پنجره (پیکسل [0][0]) جایگزین کرده و ادامه را بررسی می کنیم. این فیلتر باعث blur و smooth شدن تصویر شده و برای تصاویری که نویز در آنها به صورت هموار تری پخش شده و اختلاف بین پیکسل های همسایه خیلی زیاد نیست بهتر عمل می کند.

شرح نتایج

در ابتداء نویز Gaussian را بر تصویر اعمال می کنیم. (به علت کوچک بودن مقادیر واریانس تغییرات نسبت به تصویر اصلی زیاد مشهود نیست).



نتیجه اعمال فیلتر میانگین را می توان در ادامه مشاهده کرد.

Image: Gaussian Noise (Noise Variance = 0.01)

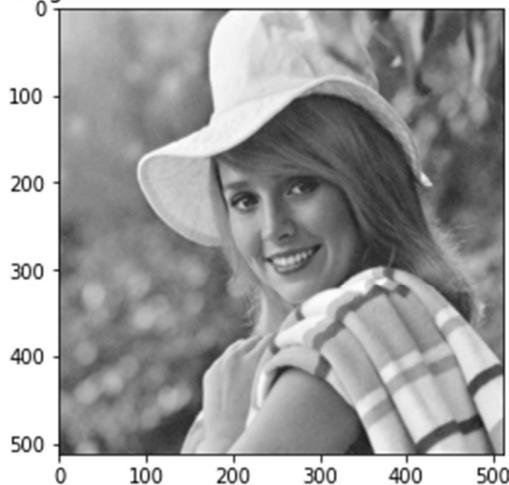


Image: Mean Filter (Filter Size = 3)

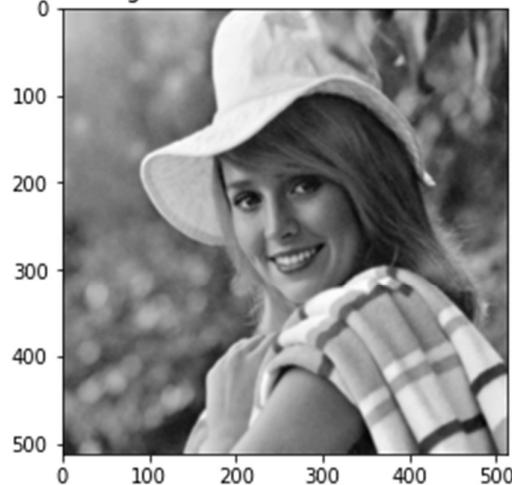


Image: Gaussian Noise (Noise Variance = 0.05)

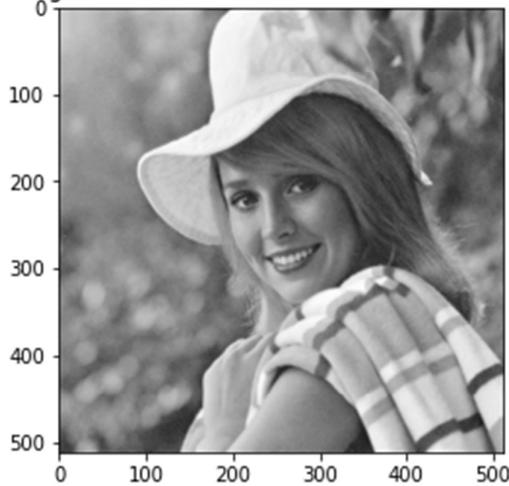


Image: Mean Filter (Filter Size = 3)

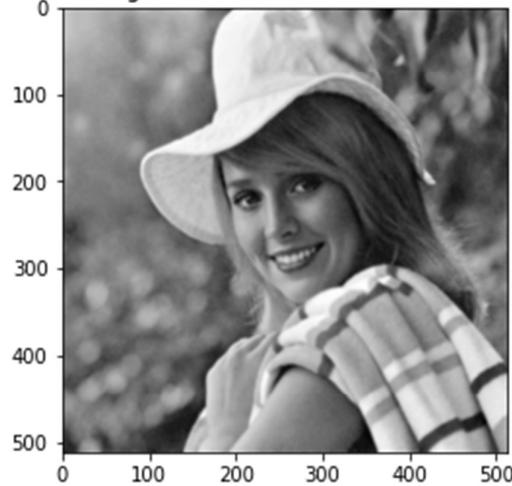


Image: Gaussian Noise (Noise Variance = 0.1)

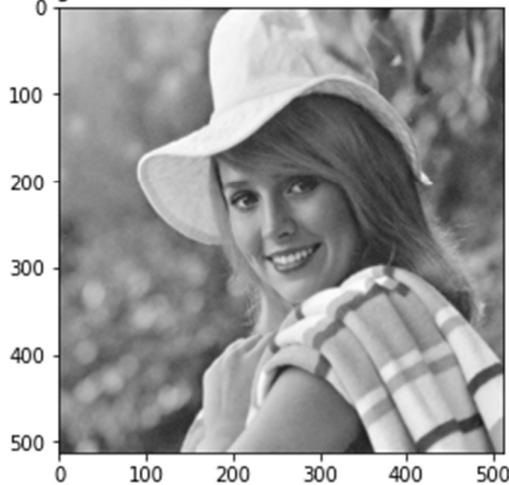
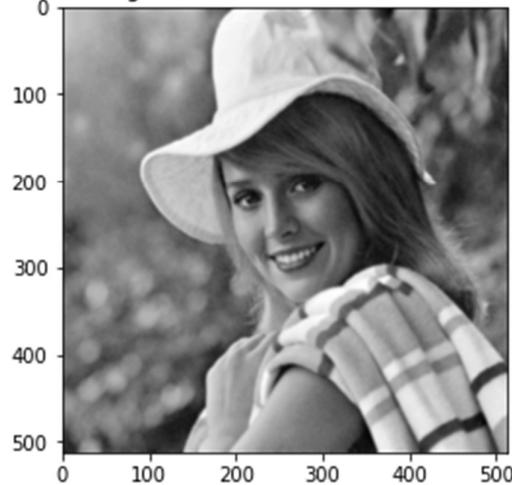


Image: Mean Filter (Filter Size = 3)



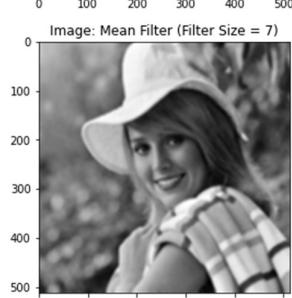
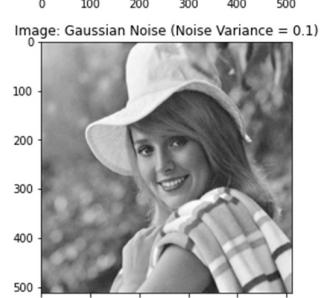
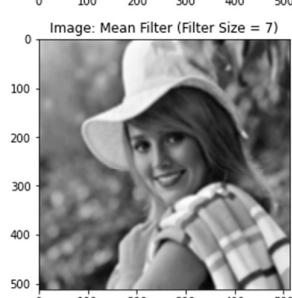
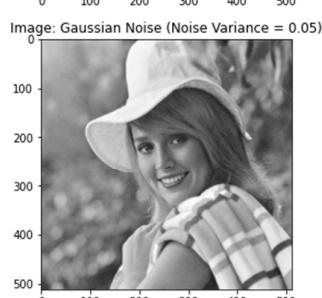
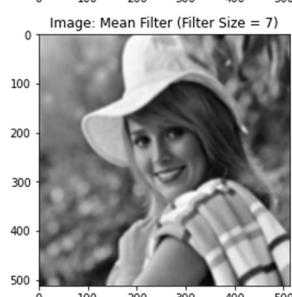
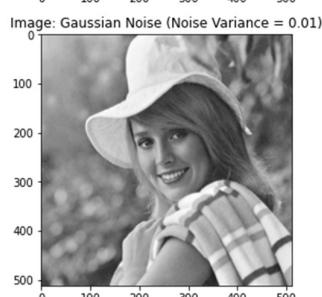
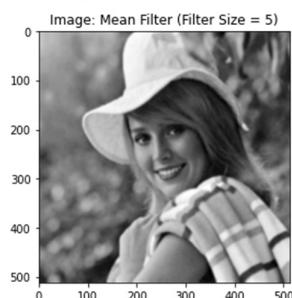
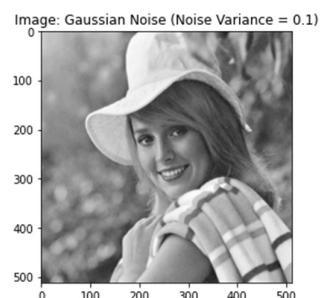
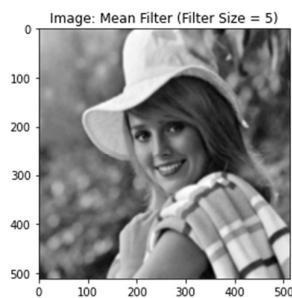
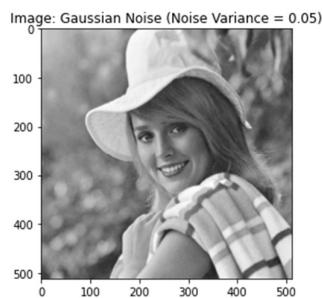
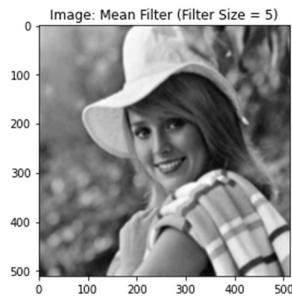
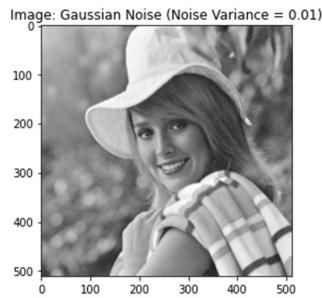


Image: Gaussian Noise (Noise Variance = 0.01)

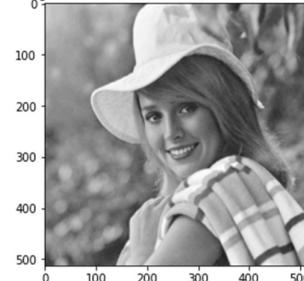


Image: Mean Filter (Filter Size = 9)

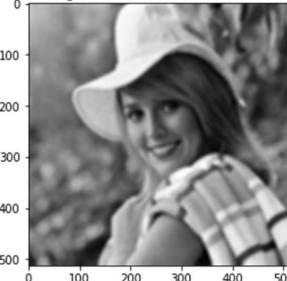


Image: Gaussian Noise (Noise Variance = 0.05)

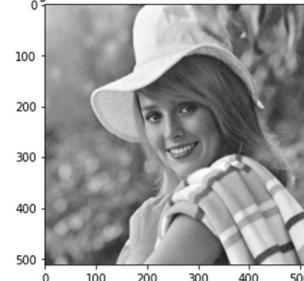


Image: Mean Filter (Filter Size = 9)

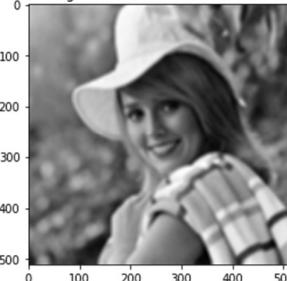


Image: Gaussian Noise (Noise Variance = 0.1)

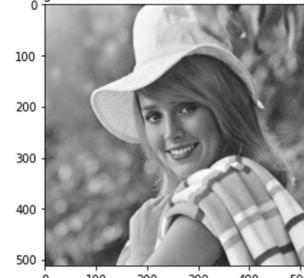


Image: Mean Filter (Filter Size = 9)

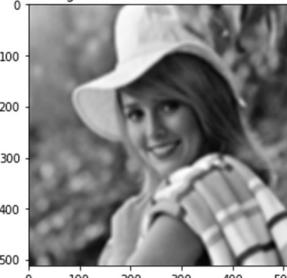


Image: Gaussian Noise (Noise Variance = 0.01)

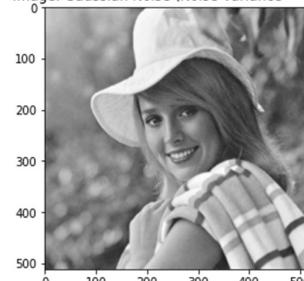


Image: Mean Filter (Filter Size = 11)

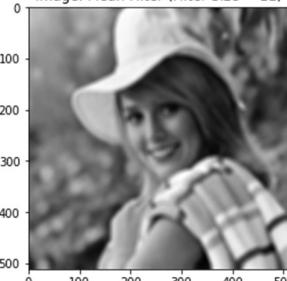


Image: Gaussian Noise (Noise Variance = 0.05)

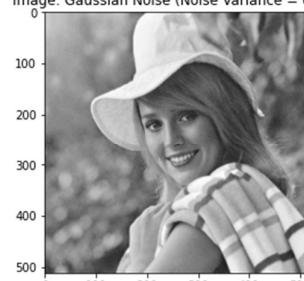


Image: Mean Filter (Filter Size = 11)

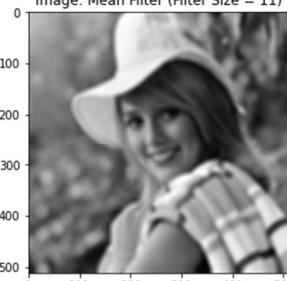


Image: Gaussian Noise (Noise Variance = 0.1)

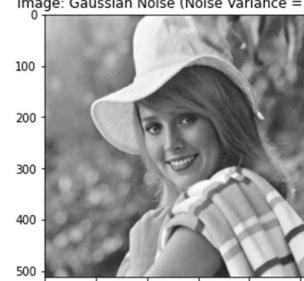
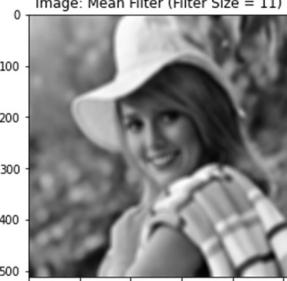


Image: Mean Filter (Filter Size = 11)



همانطور که مشاهده می شود با افزایش سایز پنجره، تصویر حدودی تر شده و پس از مدتی جزئیات خیلی زیادی از تصویر از بین می رود.

حال بر روی تصویر نویز دار، این بار فیلتر میانه را اعمال می کنیم.

Image: Gaussian Noise (Noise Variance = 0.01)

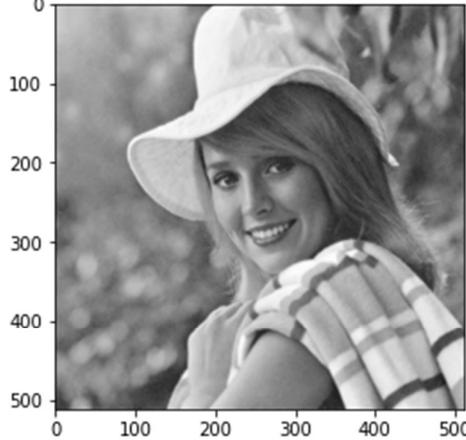


Image: Median Filter (Filter Size = 3)

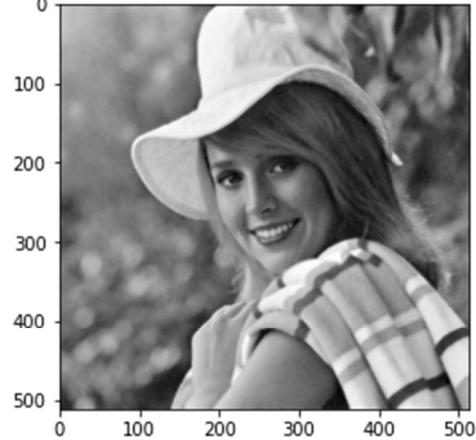


Image: Gaussian Noise (Noise Variance = 0.05)

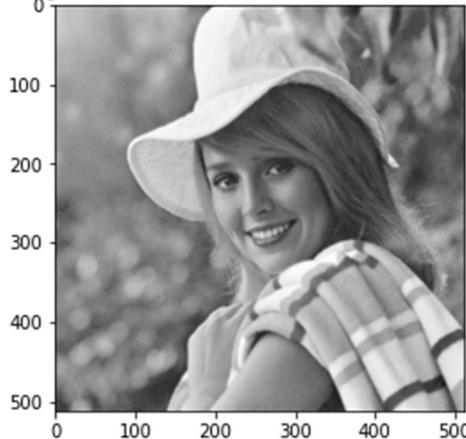


Image: Median Filter (Filter Size = 3)

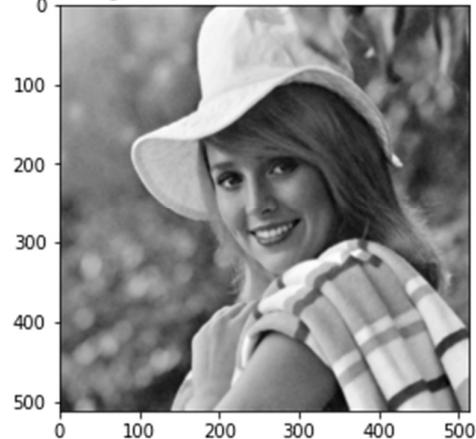


Image: Gaussian Noise (Noise Variance = 0.1)

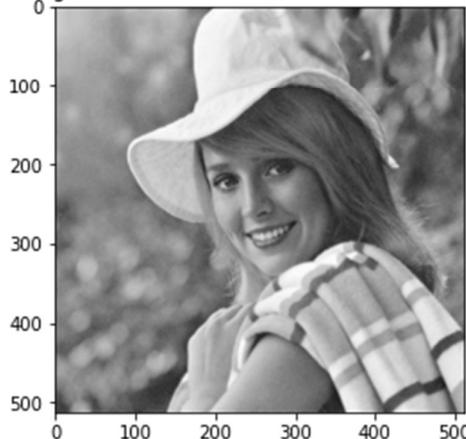
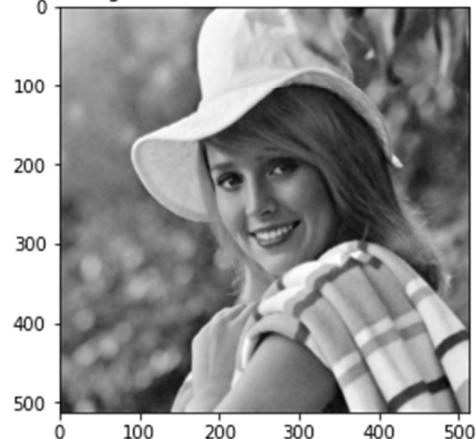
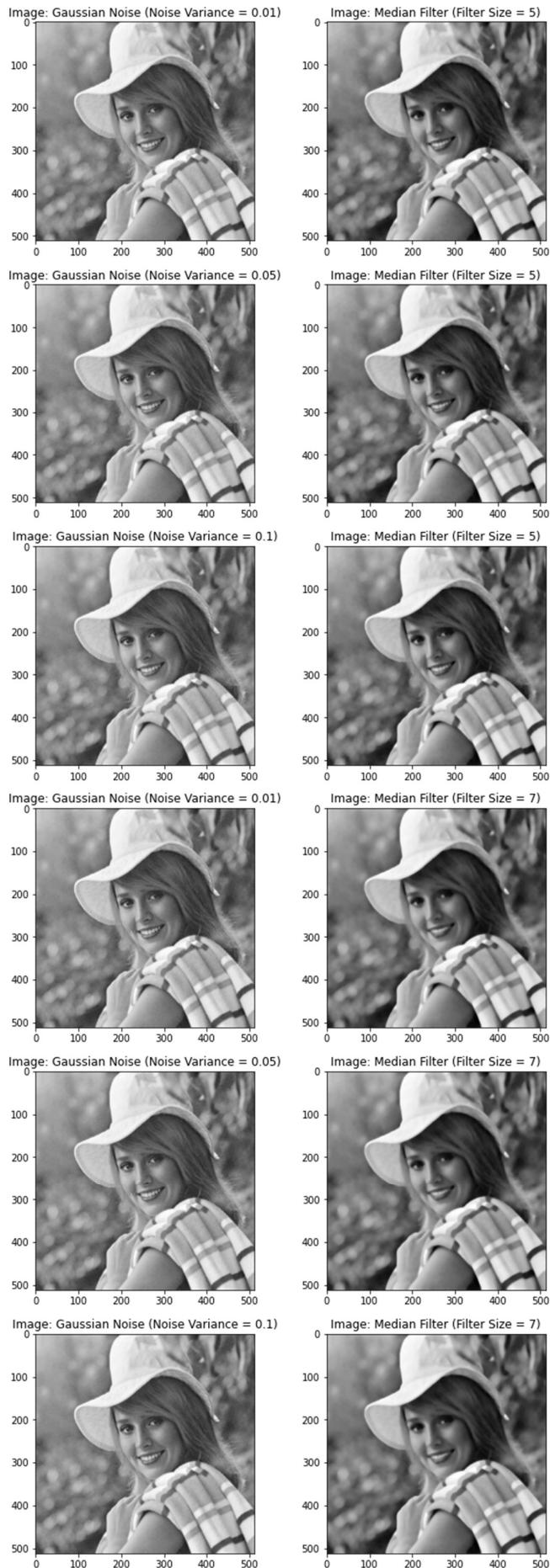
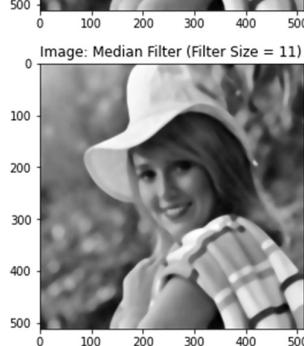
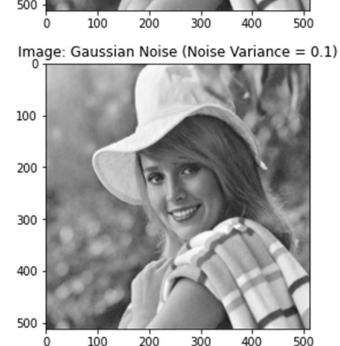
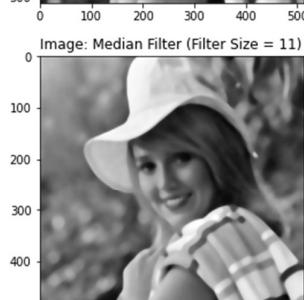
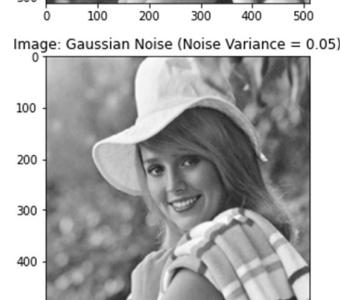
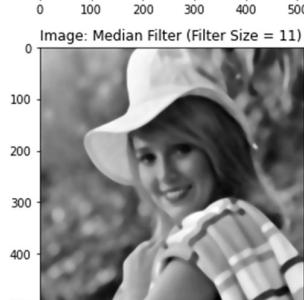
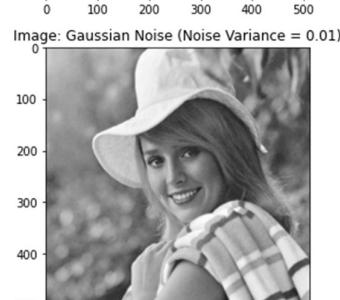
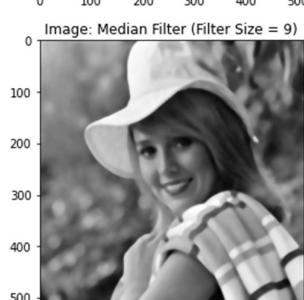
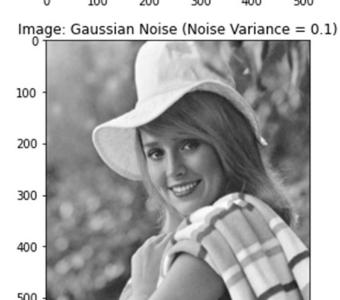
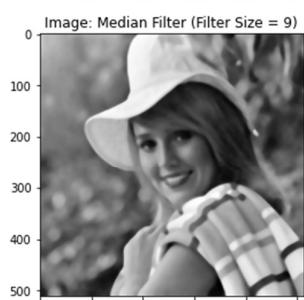
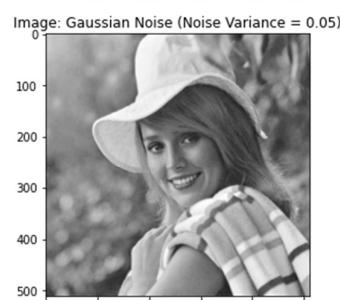
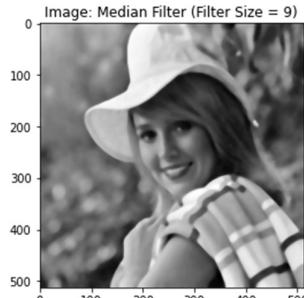
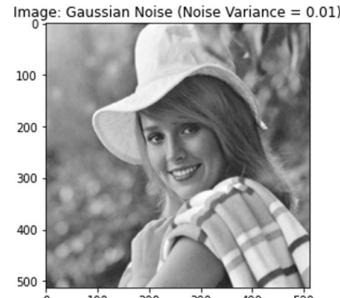


Image: Median Filter (Filter Size = 3)







در مقایسه با فیلتر میانگین، می توان مشاهده کرد که نتایج مخصوصا در پنجره های با سایز بالاتر، بسیار متفاوت تر از تصویر اصلی بوده و جزئیات خیلی بیشتری از دست می رود که این مورد نیز به علت انتخاب میانه پنجره به جای میانگین آن است که جزئیات خیلی زیادی از بین می ورد و مقدار یک پیکسل کمتر به پیکسل های همسایه وابسته است. می توان در مقادیر MSE نیز این مورد را مشاهده کرد.

```
Mean and Median Filter: Filter Size = 3
----Gaussian Noise: Noise Density = 0.01----
MSE Between Noisy and Filtered Images with Mean Filter =
90.00556256687074
MSE Between Noisy and Filtered Images with Median Filter =
98.90275190197457
MSE Between Original and Filtered Images with Mean Filter =
90.00524514991355
MSE Between Original and Filtered Images with Median Filter =
98.9024634746639
----Gaussian Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images with Mean Filter =
90.00672912883186
MSE Between Noisy and Filtered Images with Median Filter =
98.88818536104512
MSE Between Original and Filtered Images with Mean Filter =
90.00259170367107
MSE Between Original and Filtered Images with Median Filter =
98.88390931071024
----Gaussian Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images with Mean Filter =
90.01515620268582
MSE Between Noisy and Filtered Images with Median Filter =
98.87109326875053
MSE Between Original and Filtered Images with Mean Filter =
90.0075884193866
MSE Between Original and Filtered Images with Median Filter =
98.86443602307185
```

```
Mean and Median Filter: Filter Size = 5
----Gaussian Noise: Noise Density = 0.01----
MSE Between Noisy and Filtered Images with Mean Filter =
203.16145243560115
MSE Between Noisy and Filtered Images with Median Filter =
229.19191751271268
MSE Between Original and Filtered Images with Mean Filter =
203.1609240268779
MSE Between Original and Filtered Images with Median Filter =
229.19140526637256
----Gaussian Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images with Mean Filter =
203.16544987241141
MSE Between Noisy and Filtered Images with Median Filter =
229.1685446387392
MSE Between Original and Filtered Images with Mean Filter =
203.1572206518431
MSE Between Original and Filtered Images with Median Filter =
229.1602601841064
```

```
----Gaussian Noise: Noise Density = 0.1----  
MSE Between Noisy and Filtered Images with Mean Filter =  
203.17725672173722  
MSE Between Noisy and Filtered Images with Median Filter =  
229.16683104608293  
MSE Between Original and Filtered Images with Mean Filter =  
203.16460074770345  
MSE Between Original and Filtered Images with Median Filter =  
229.1542229997082
```

```
Mean and Median Filter: Filter Size = 7  
----Gaussian Noise: Noise Density = 0.01----  
MSE Between Noisy and Filtered Images with Mean Filter =  
327.9431437821788  
MSE Between Noisy and Filtered Images with Median Filter =  
376.9060274613272  
MSE Between Original and Filtered Images with Mean Filter =  
327.94237327716723  
MSE Between Original and Filtered Images with Median Filter =  
376.9051867845667  
----Gaussian Noise: Noise Density = 0.05----  
MSE Between Noisy and Filtered Images with Mean Filter =  
327.9485523099498  
MSE Between Noisy and Filtered Images with Median Filter =  
376.8893751298531  
MSE Between Original and Filtered Images with Mean Filter =  
327.93810699832443  
MSE Between Original and Filtered Images with Median Filter =  
376.87779631568526  
----Gaussian Noise: Noise Density = 0.1----  
MSE Between Noisy and Filtered Images with Mean Filter =  
327.96114533991357  
MSE Between Noisy and Filtered Images with Median Filter =  
376.86723461910253  
MSE Between Original and Filtered Images with Mean Filter =  
327.9456333833757  
MSE Between Original and Filtered Images with Median Filter =  
376.8524385214979
```

```
Mean and Median Filter: Filter Size = 9  
----Gaussian Noise: Noise Density = 0.01----  
MSE Between Noisy and Filtered Images with Mean Filter =  
453.222776700816  
MSE Between Noisy and Filtered Images with Median Filter =  
527.7321480313507  
MSE Between Original and Filtered Images with Mean Filter =  
453.22179586705715  
MSE Between Original and Filtered Images with Median Filter =  
527.7313131475777  
----Gaussian Noise: Noise Density = 0.05----  
MSE Between Noisy and Filtered Images with Mean Filter =  
453.22794466984965  
MSE Between Noisy and Filtered Images with Median Filter =  
527.7123609308992  
MSE Between Original and Filtered Images with Mean Filter =  
453.2162877409753
```

```

MSE Between Original and Filtered Images with Median Filter =
527.6991752099457
----Gaussian Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images with Mean Filter =
453.2421115952384
MSE Between Noisy and Filtered Images with Median Filter =
527.687886693051
MSE Between Original and Filtered Images with Mean Filter =
453.2247237452714
MSE Between Original and Filtered Images with Median Filter =
527.6709699492362

```

```

Mean and Median Filter: Filter Size = 11
----Gaussian Noise: Noise Density = 0.01----
MSE Between Noisy and Filtered Images with Mean Filter =
572.8107154530405
MSE Between Noisy and Filtered Images with Median Filter =
672.8927217443819
MSE Between Original and Filtered Images with Mean Filter =
572.8094863987421
MSE Between Original and Filtered Images with Median Filter =
672.8915314136229
----Gaussian Noise: Noise Density = 0.05----
MSE Between Noisy and Filtered Images with Mean Filter =
572.8152059097115
MSE Between Noisy and Filtered Images with Median Filter =
672.8711039764207
MSE Between Original and Filtered Images with Mean Filter =
572.802786515552
MSE Between Original and Filtered Images with Median Filter =
672.8572289490464
----Gaussian Noise: Noise Density = 0.1----
MSE Between Noisy and Filtered Images with Mean Filter =
572.8300649959764
MSE Between Noisy and Filtered Images with Median Filter =
672.8566781457664
MSE Between Original and Filtered Images with Mean Filter =
572.8110889420533
MSE Between Original and Filtered Images with Median Filter =
672.8377864859701

```

مهم

```

@jit(nopython=True)
def gaussian_noise(image, filter_variance):
    R, C = image.shape

    gaussian_noise_vector = np.random.normal(0, filter_variance, (R, C, 1))
    gaussian_noise_vector = gaussian_noise_vector.reshape(R, C)

    guassian_noise_added_image = image + gaussian_noise_vector

    return guassian_noise_added_image

```

```

# 3.2.2 - Guassian Noise Added Images

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

noise_variances = [0.01, 0.05, 0.1]

R, C = img.shape
gaussian_noise_added_images = np.zeros((len(noise_variances), R, C))

fig, plot = plt.subplots(1, len(noise_variances), figsize = (5 * len(noise_variances), 5))

for noise_variance in range(len(noise_variances)):
    gaussian_noise_added_image = gaussian_noise(img, noise_variances[noise_variance])
    plot[noise_variance].imshow(gaussian_noise_added_image, cmap='gray')
    plot[noise_variance].set_title("Image: Gaussian Noise (Noise Variance = " + str(noise_variances[noise_variance]) + ")")

    gaussian_noise_added_images[noise_variance, 0:R, 0:C] = gaussian_noise_added_image

```

```

def mean_filter(image, filter_size):
    R, C = image.shape

    average_filtered_image = np.zeros((R, C))

    for r in range (R):
        for c in range (C):
            average_filtered_image[r][c] = np.mean(image[r:r + filter_size, c:c + filter_size].flatten())

    return average_filtered_image

```

```

# 3.2.2 - Mean Filtered Images

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

filter_sizes = [3, 5, 7, 9, 11]

mse_noisy_and_filtered_mean = np.zeros(len(filter_sizes) * len(noise_variances))
mse_original_and_filtered_mean = np.zeros(len(filter_sizes) * len(noise_variances))

```

```

fig, plot = plt.subplots(len(filter_sizes) * len(noise_variances), 2, figsize = (10, 5 * len(filter_sizes) * len(noise_variances)))

for filter_size in range(len(filter_sizes)):
    for noise_variance in range(len(noise_variances)):
        gaussian_noise_added_image = gaussian_noise_added_images[noise_variance, 0:R, 0:C]
        plot[filter_size * len(noise_variances) + noise_variance][0].imshow(gaussian_noise_added_image, cmap='gray')
        plot[filter_size * len(noise_variances) + noise_variance][0].set_title("Image: Gaussian Noise (Noise Variance = " + str(noise_variances[noise_variance]) + ")")

        mean_filtered_image = mean_filter(gaussian_noise_added_image, filter_sizes[filter_size])
        plot[filter_size * len(noise_variances) + noise_variance][1].imshow(mean_filtered_image, cmap='gray')
        plot[filter_size * len(noise_variances) + noise_variance][1].set_title("Image: Mean Filter (Filter Size = " + str(filter_sizes[filter_size]) + ")")

        mse_noisy_and_filtered_mean[filter_size * len(noise_variances) + noise_variance] = mean_squared_error(gaussian_noise_added_image, mean_filtered_image)
        mse_original_and_filtered_mean[filter_size * len(noise_variances) + noise_variance] = mean_squared_error(img, mean_filtered_image)

```

```

# 3.2.2 - Median Filtered Images
fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

filter_sizes = [3, 5, 7, 9, 11]

mse_noisy_and_filtered_median = np.zeros(len(filter_sizes) * len(noise_variances))
mse_original_and_filtered_median = np.zeros(len(filter_sizes) * len(noise_variances))

fig, plot = plt.subplots(len(filter_sizes) * len(noise_variances), 2, figsize = (10, 5 * len(filter_sizes) * len(noise_variances)))

for filter_size in range(len(filter_sizes)):
    for noise_variance in range(len(noise_variances)):
        gaussian_noise_added_image = gaussian_noise_added_images[noise_variance, 0:R, 0:C]

```

```

    plot[filter_size * len(noise_variances) + noise_variance][0].imshow
(gaussian_noise_added_image, cmap='gray')
    plot[filter_size * len(noise_variances) + noise_variance][0].set_tit
le("Image: Gaussian Noise (Noise Variance = " + str(noise_variances[no
ise_variance]) + ")")

    median_filtered_image = median_filter(gaussian_noise_added_image, f
ilter_sizes[filter_size])
    plot[filter_size * len(noise_variances) + noise_variance][1].imshow
(median_filtered_image, cmap='gray')
    plot[filter_size * len(noise_variances) + noise_variance][1].set_tit
le("Image: Median Filter (Filter Size = " + str(filter_sizes[filter_si
ze]) + ")")

    mse_noisy_and_filtered_mean[filter_size * len(noise_variances) +
noise_variance] = mean_squared_error(gaussian_noise_added_image, median
_filtered_image)
    mse_original_and_filtered_mean[filter_size * len(noise_variances) +
noise_variance] = mean_squared_error(img, median_filtered_image)

```

```

# 3.2.2 - MSE Report

for filter_size in range(len(filter_sizes)):
    print("Mean and Median Filter: Filter Size = " + str(filter_sizes[fil
ter_size]))
    for noise_variance in range(len(noise_variances)):
        print("----"
Gaussian Noise: Noise Density = " + str(noise_variances[noise_variance]
) + "----")
        print("MSE Between Noisy and Filtered Images with Mean Filter = " +
str(mse_noisy_and_filtered_mean[filter_size * len(noise_variances) + n
oise_variance]))
        print("MSE Between Noisy and Filtered Images with Median Filter = " +
str(mse_noisy_and_filtered_median[filter_size * len(noise_variances) +
noise_variance]))
        print("MSE Between Original and Filtered Images with Mean Filter =
" + str(mse_original_and_filtered_mean[filter_size * len(noise_varianc
es) + noise_variance]))
        print("MSE Between Original and Filtered Images with Median Filter
= " + str(mse_original_and_filtered_median[filter_size * len(noise_vari
ances) + noise_variance]))
        print("\n")

```

تمرین ۳ سوال ۳

چکیده

در این تمرین به بررسی نتیجه اعمال فیلتر بر تصویری که دارای هر دو نویز Salt and Pepper و Gaussian است می پردازیم.

مقدمه

با توجه به اینکه در قسمت های قبلی تمرین آموختیم که فیلتر Median برای نویز Salt and Pepper مناسب تر بوده و فیلتر Mean برای نویز های Gaussian مناسب تر است، هر دو فیلتر را بر تصویر اعمال کرده و نتیجه را بررسی می کنیم. با توجه به ماهیت فیلتر ها که بر تمام پیکسل ها با توجه به مقادیر پیکسل های دیگر (پردازش ناحیه ای) تاثیر گذارند، ترتیب اعمال آنها اهمیت داشته و بنابراین هر دو حالت را محاسبه کرده و نتیجه بهتر را انتخاب می کنیم. همچنین در نویز Gaussian به علت تاثیر گذاری مقدار پیکسل های تصویر قبل از اعمال فیلتر بر تصویر پس از اعمال فیلتر، بار دیگر هر دو نویز را در ترتیب های متفاوت بررسی می کنیم.

بنابراین ۴ حالت زیر را خواهیم داشت:

۱. Median – Mean – Gaussian – Salt and Pepper.

۲. Mean – Median – Gaussian – Salt and Pepper.

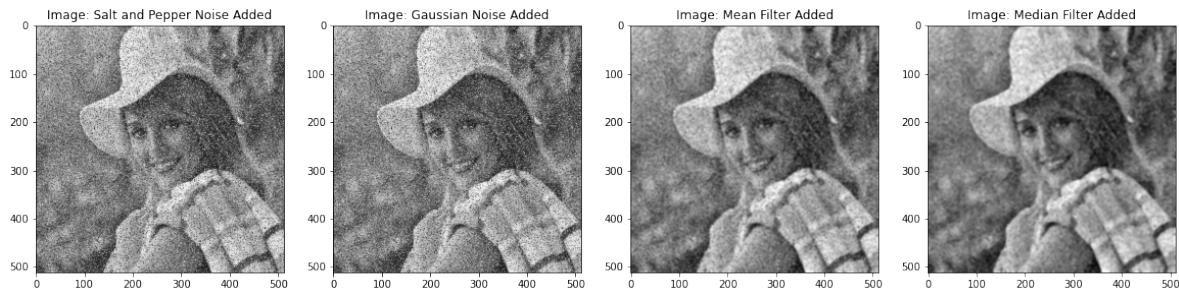
۳. Median – Mean – Salt and Pepper – Gaussian.

۴. Mean – Median – Salt and Pepper – Gaussian.

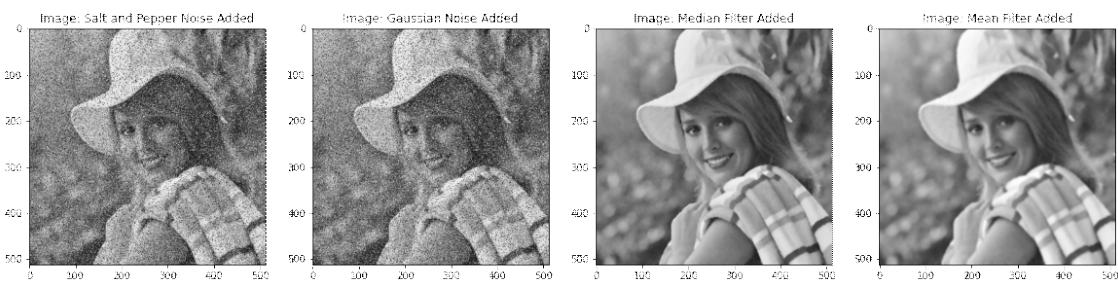
در قسمت مشاهده نتایج این حالت ها را بررسی کرده و با یکدیگر مقایسه می کنیم.

شرح نتایج

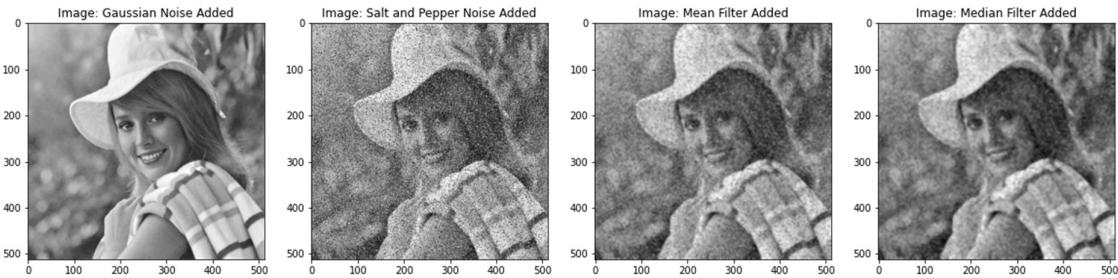
Median – Mean – Gaussian – Salt and Pepper .۱



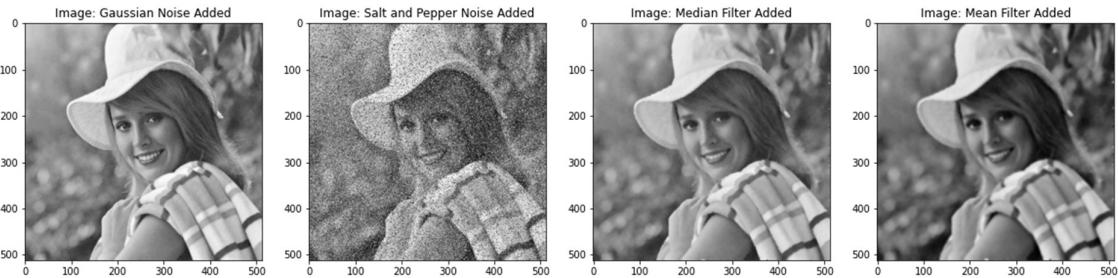
Mean – Median – Gaussian – Salt and Pepper .۲



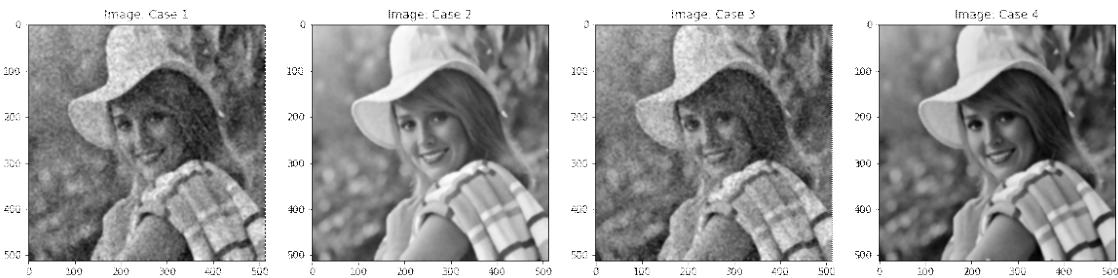
Median – Mean – Salt and Pepper – Gaussian .۳



Mean – Median – Salt and Pepper – Gaussian .۴



همه حالات در کنار هم



همانطور که مشاهده می شود در حالاتی که ابتدا فیلتر میانه اعمال شده و سپس فیلتر میانگین اعمال می شود، نتیجه بسیار بهتر است و علت آن است که مقادیر پیکسل ها نسبت به پیکسل های بسیار متفاوت سفید و سیاه اضافه شده توسط نویز Salt and Pepper توسط فیلتر میانگین بسیار تغییر نکرده و ابتدا نویز Salt and Pepper حذف می شود و پس از آن تصویر توسط فیلتر میانگین smooth می شود.

در بررسی ترتیب نویز ها نیز می بینیم که در حالتی که ابتدا نویز Salt and Pepper اعمال شده است، نتیجه بهتری داریم که به نظر علت آن smooth شدن مقادیر پیکسل ها تا حدودی توسط اضافه شدن نویز Gaussian است که مقادیر را کمی بیشتر به هم نزدیک می کند.

این تحلیل ها را نیز می توان در مقادیر MSE مشاهده کرد.

MSE Report

```
Case 1: Salt and Pepper Noise(0.1) -> Gaussian Noise(0.1) -> Mean  
Filter(5) -> Median Filter(5) MSE = 589.9504425609473
```

```
Case 2: Salt and Pepper Noise(0.1) -> Gaussian Noise(0.1) -> Median  
Filter(5) -> Mean Filter(5) MSE = 512.6706167255204
```

```
Case 3: Gaussian Noise(0.1) -> Salt and Pepper Noise(0.1) -> Mean  
Filter(5) -> Median Filter(5) MSE = 586.4087135701509
```

```
Case 4: Gaussian Noise(0.1) -> Salt and Pepper Noise(0.1) -> Median  
Filter(5) -> Mean Filter(5) -> Median Filter(5) MSE = 513.5038190121368
```

ضمیمه

```
# 3.2.3 - Case 1: Salt and Pepper Noise(0.1) -> Gaussian Noise(0.1) -  
> Mean Filter(5) -> Median Filter(5)

salt_and_pepper_noise_density = 0.1
gaussian_noise_variance = 0.1
mean_filter_size = 5
median_filter_size = 5

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

salt_and_pepper_noise_added_image = salt_and_pepper_noise(img, salt_and_
_pepper_noise_density)
gaussian_noise_added_image = gaussian_noise(salt_and_pepper_noise_added_
_image, gaussian_noise_variance)
mean_filtered_image = mean_filter(gaussian_noise_added_image, mean_filt_
er_size)
median_filtered_image = median_filter(mean_filtered_image, median_filt_
r_size)

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(salt_and_pepper_noise_added_image, cmap='gray')
plot[0].set_title("Image: Salt and Pepper Noise Added")

plot[1].imshow(gaussian_noise_added_image, cmap='gray')
plot[1].set_title("Image: Gaussian Noise Added")

plot[2].imshow(mean_filtered_image, cmap='gray')
plot[2].set_title("Image: Mean Filter Added")
```

```

plot[3].imshow(median_filtered_image, cmap='gray')
plot[3].set_title("Image: Median Filter Added")

case_1_result = median_filtered_image
case_1_mse = mean_squared_error(case_1_result, img)

# 3.2.3 - Case 2: Salt and Pepper Noise(0.1) -> Gaussian Noise(0.1) -
> Median Filter(5) -> Mean Filter(5)

salt_and_pepper_noise_density = 0.1
gaussian_noise_variance = 0.1
median_filter_size = 5
mean_filter_size = 5

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

salt_and_pepper_noise_added_image = salt_and_pepper_noise(img, salt_and_
pepper_noise_density)
gaussian_noise_added_image = gaussian_noise(salt_and_pepper_noise_added_
image, gaussian_noise_variance)
median_filtered_image = median_filter(gaussian_noise_added_image, media_
n_filter_size)
mean_filtered_image = mean_filter(median_filtered_image, mean_filter_si_
ze)

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(salt_and_pepper_noise_added_image, cmap='gray')
plot[0].set_title("Image: Salt and Pepper Noise Added")

plot[1].imshow(gaussian_noise_added_image, cmap='gray')
plot[1].set_title("Image: Gaussian Noise Added")

plot[2].imshow(median_filtered_image, cmap='gray')
plot[2].set_title("Image: Median Filter Added")

plot[3].imshow(mean_filtered_image, cmap='gray')
plot[3].set_title("Image: Mean Filter Added")

case_2_result = mean_filtered_image
case_2_mse = mean_squared_error(case_2_result, img)

# 3.2.3 - Case 3: Gaussian Noise(0.1) -> Salt and Pepper Noise(0.1) -
> Mean Filter(5) -> Median Filter(5)

gaussian_noise_variance = 0.1

```

```

salt_and_pepper_noise_density = 0.1
mean_filter_size = 5
median_filter_size = 5

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

gaussian_noise_added_image = gaussian_noise(img, gaussian_noise_variance)
salt_and_pepper_noise_added_image = salt_and_pepper_noise(gaussian_noise_added_image, salt_and_pepper_noise_density)
mean_filtered_image = mean_filter(salt_and_pepper_noise_added_image, mean_filter_size)
median_filtered_image = median_filter(mean_filtered_image, median_filter_size)

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(gaussian_noise_added_image, cmap='gray')
plot[0].set_title("Image: Gaussian Noise Added")

plot[1].imshow(salt_and_pepper_noise_added_image, cmap='gray')
plot[1].set_title("Image: Salt and Pepper Noise Added")

plot[2].imshow(mean_filtered_image, cmap='gray')
plot[2].set_title("Image: Mean Filter Added")

plot[3].imshow(median_filtered_image, cmap='gray')
plot[3].set_title("Image: Median Filter Added")

case_3_result = median_filtered_image
case_3_mse = mean_squared_error(case_3_result, img)

```

```

# 3.2.3 - Case 4: Gaussian Noise(0.1) -> Salt and Pepper Noise(0.1) -
> Median Filter(5) -> Mean Filter(5)

gaussian_noise_variance = 0.1
salt_and_pepper_noise_density = 0.1
median_filter_size = 5
mean_filter_size = 5

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

```

```

gaussian_noise_added_image = gaussian_noise(img, gaussian_noise_variance)
salt_and_pepper_noise_added_image = salt_and_pepper_noise(gaussian_noise_added_image, salt_and_pepper_noise_density)
median_filtered_image = median_filter(salt_and_pepper_noise_added_image, median_filter_size)
mean_filtered_image = mean_filter(median_filtered_image, mean_filter_size)

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(gaussian_noise_added_image, cmap='gray')
plot[0].set_title("Image: Gaussian Noise Added")

plot[1].imshow(salt_and_pepper_noise_added_image, cmap='gray')
plot[1].set_title("Image: Salt and Pepper Noise Added")

plot[2].imshow(median_filtered_image, cmap='gray')
plot[2].set_title("Image: Median Filter Added")

plot[3].imshow(mean_filtered_image, cmap='gray')
plot[3].set_title("Image: Mean Filter Added")

case_4_result = mean_filtered_image
case_4_mse = mean_squared_error(case_4_result, img)

```

```

# 3.2.3 - All Cases Together

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(case_1_result, cmap='gray')
plot[0].set_title("Image: Case 1")

plot[1].imshow(case_2_result, cmap='gray')
plot[1].set_title("Image: Case 2")

plot[2].imshow(case_3_result, cmap='gray')
plot[2].set_title("Image: Case 3")

plot[3].imshow(case_4_result, cmap='gray')
plot[3].set_title("Image: Case 4")

print("MSE Report")

```

```
print("Case 1: Salt and Pepper Noise(0.1) -> Gaussian Noise(0.1) -  
> Mean Filter(5) -> Median Filter(5) MSE = " + str(case_1_mse))  
print("Case 2: Salt and Pepper Noise(0.1) -> Gaussian Noise(0.1) -  
> Median Filter(5) -> Mean Filter(5) MSE = " + str(case_2_mse))  
print("Case 3: Gaussian Noise(0.1) -> Salt and Pepper Noise(0.1) -  
> Mean Filter(5) -> Median Filter(5) MSE = " + str(case_3_mse))  
print("Case 4: Gaussian Noise(0.1) -> Salt and Pepper Noise(0.1) -  
> Median Filter(5) -> Mean Filter(5) -  
> Median Filter(5) MSE = " + str(case_4_mse))
```

تمرین ۳ سوال ۳.۳

چکیده

هدف از این تمرین آشنازی با بهبود تصویر در شرایط واقعی و بدون اعمال دستی فیلتر است.

مقدمه

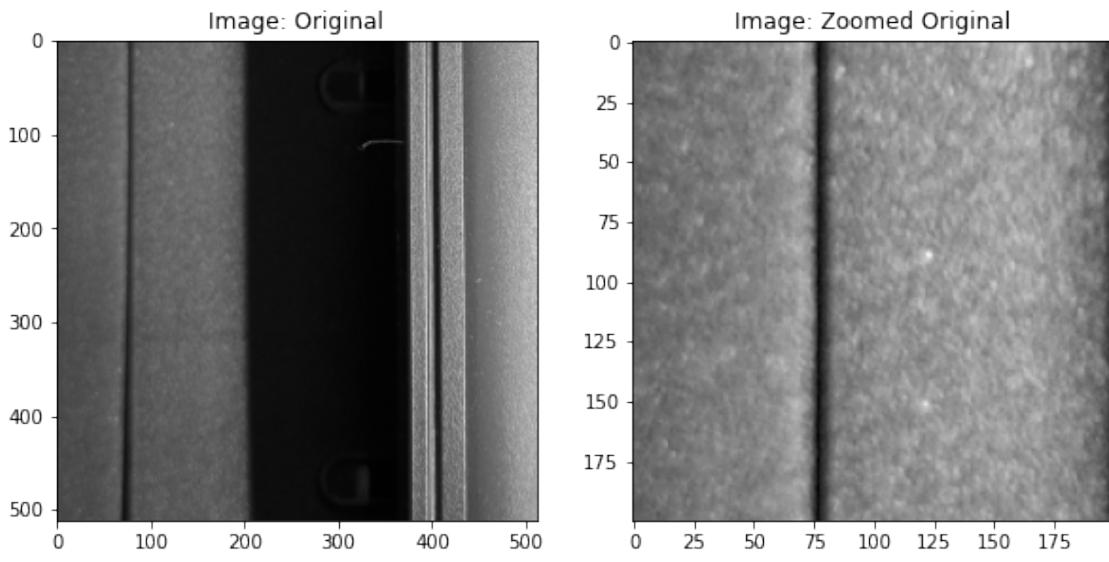
با ۳ نوع فیلتری که تا کنون آموختیم و حالت های مختلف آنها، نتایج زیر را تولید کردم. با توجه به اینکه تصویر گرفته شده هم دارای نویز و هم Blury است، باید برای رفع نویز از یکی یا هردو فیلتر های Median Filter و Mean Filter استفاده کرده و برای رفع مشکل Blur هم از Laplacian Filter برای Sharp کردن تصویر استفاده کرد.

در قسمت شرح نتایج، ۴ حالت زیر را به ترتیب اعمال فیلتر های گفته شده بررسی می کنیم:

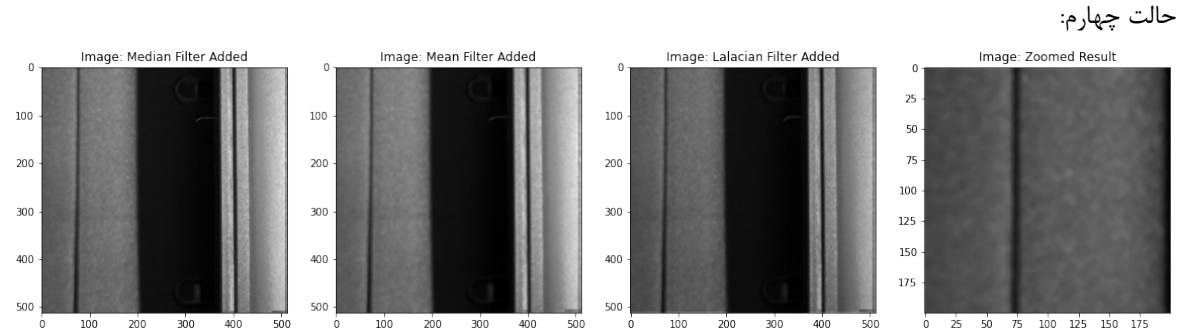
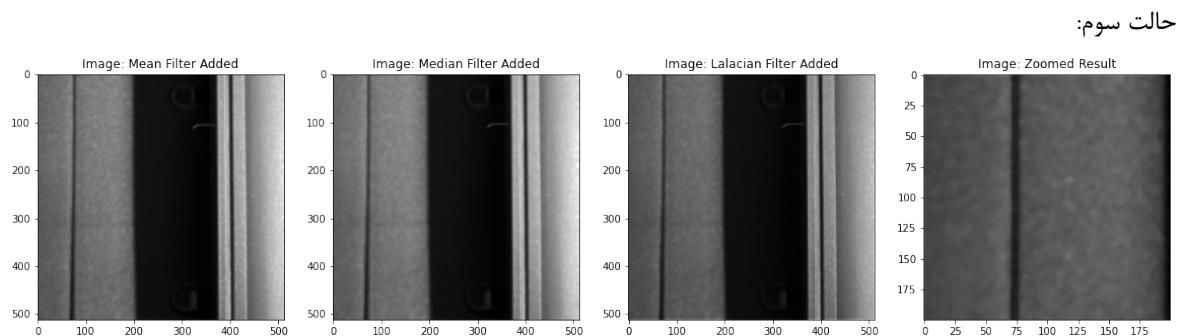
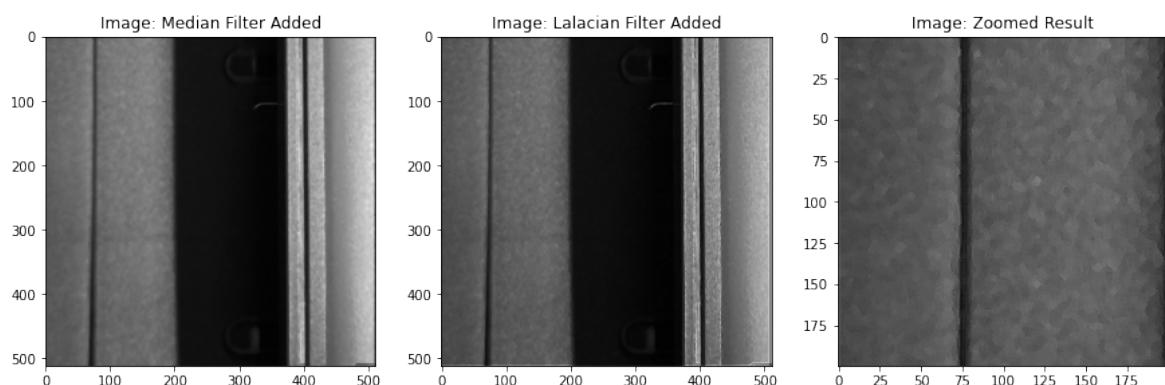
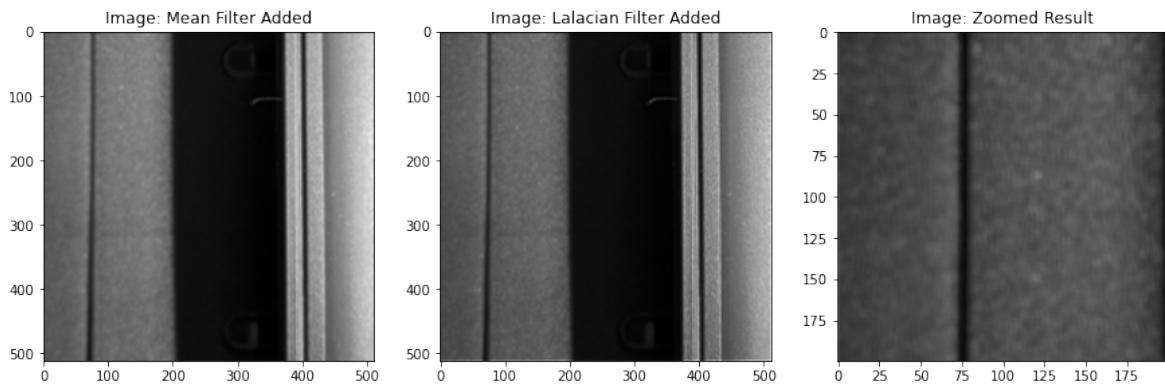
۱. فیلتر Mean و Laplacian
۲. فیلتر Median و Laplacian
۳. فیلتر Mean و Median و Laplacian
۴. فیلتر Median و Mean و Laplacian

شرح نتایج

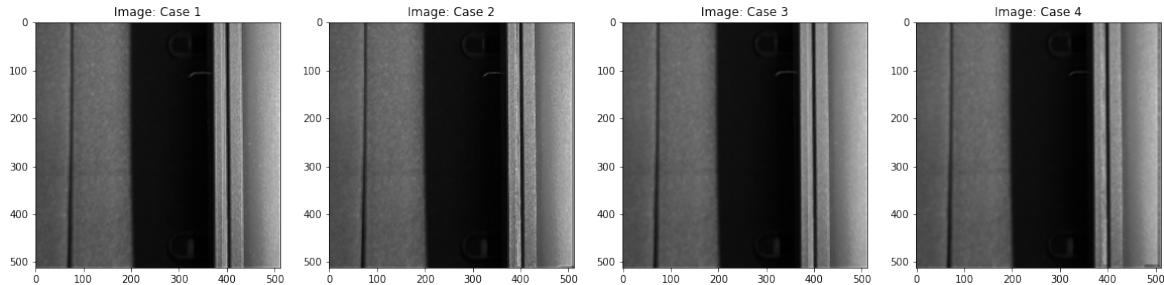
تصویر اصلی:



حالات اول:



همه حالات در کنار هم:



علاوه بر تصاویر می توان به مقادیر MSE نیز توجه داشت:

MSE Report

```
Case 1: Mean Filter -> Laplacian Filter MSE = 261.7553291320801
Case 2: Median Filter -> Laplacian Filter MSE = 543.8545913696289
Case 3: Mean Filter -> Median Filter -> Laplacian Filter MSE = 523.1673278808594
Case 4: Median Filter -> Mean Filter -> Laplacian Filter MSE = 573.9383583068848
```

همانطور که مشاهده می شود، حالت دوم یعنی حالتی که تنها فیلتر Median را بر تصویر اعمال کرده و پس از آن blur اعمال می کنیم دارای نتیجه بهتری بوده و edge ها را بسیار بهتر نمایش می دهد و مانند حالات سوم و چهارم تصویر نمی شود. (به علت عدم اعمال فیلتر میانگین).

ضمیمه

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
import random
from sklearn.metrics import mean_squared_error
from statistics import median

img = cv2.imread('image.bmp', 0)

def convolve2D(image, kernel, padding=0, strides=1):
    kernel = np.flipud(np.fliplr(kernel))

    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
```

```

        imagePadded[int(padding):int(-1 * padding), int(padding):int(-
1 * padding)] = image
    else:
        imagePadded = image

    for y in range(imagePadded.shape[1]):
        if y > imagePadded.shape[1] - yKernShape:
            break
        if y % strides == 0:
            for x in range(imagePadded.shape[0]):
                if x > imagePadded.shape[0] - xKernShape:
                    break
                try:
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y:
y + yKernShape]).sum()
                except:
                    break

    output[output>255]=255
    output[output<0]=0

    return output

```

```

def laplacian_filter (image):
    filter_matrix = np.array([(0, -1, 0), (-1, 5, -1), (0, -1, 0)])

    laplacian_filtered_image = convolve2D(image, filter_matrix, 1)

    return laplacian_filtered_image.astype(int)

```

```

def median_filter(image):
    R, C = image.shape

    filter_size = 5

    median_filtered_image = np.zeros((R, C))

    for r in range (R):
        for c in range (C):
            median_filtered_image[r][c] = median(image[r:r + filter_size, c:c
+ filter_size].flatten())

    return median_filtered_image

```

```

def mean_filter(image):
    R, C = image.shape

    filter_size = 5

```

```

average_filtered_image = np.zeros((R, C))

for r in range (R):
    for c in range (C):
        average_filtered_image[r][c] = np.mean(image[r:r + filter_size, c:c + filter_size].flatten())

return average_filtered_image

```

```

R, C = img.shape

```

```

case_result = np.zeros((4, R, C))
case_mse = np.zeros(4)

```

```

# Case 1: Mean Filter -> Laplacian Filter

case_number = 1

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(img[0:200, 0:200], cmap='gray')
plot[1].set_title("Image: Zoomed Original")

mean_filtered_image = mean_filter(img)
laplacian_filtered_image = laplacian_filter(mean_filtered_image)

fig, plot = plt.subplots(1, 3, figsize = (15, 5))

plot[0].imshow(mean_filtered_image, cmap='gray')
plot[0].set_title("Image: Mean Filter Added")

plot[1].imshow(laplacian_filtered_image, cmap='gray')
plot[1].set_title("Image: Lalacian Filter Added")

plot[2].imshow(laplacian_filtered_image[0:200, 0:200], cmap='gray')
plot[2].set_title("Image: Zoomed Result")

case_result[case_number - 1, 0:R, 0:C] = laplacian_filtered_image
case_mse[case_number - 1] = mean_squared_error(case_result[case_number - 1, 0:R, 0:C], img)

```

```

# Case 2: Median Filter -> Laplacian Filter
case_number = 2

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

```

```

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(img[0:200, 0:200], cmap='gray')
plot[1].set_title("Image: Zoomed Original")

median_filtered_image = median_filter(img)
laplacian_filtered_image = laplacian_filter(median_filtered_image)

fig, plot = plt.subplots(1, 3, figsize = (15, 5))

plot[0].imshow(median_filtered_image, cmap='gray')
plot[0].set_title("Image: Median Filter Added")

plot[1].imshow(laplacian_filtered_image, cmap='gray')
plot[1].set_title("Image: Laplacian Filter Added")

plot[2].imshow(laplacian_filtered_image[0:200, 0:200], cmap='gray')
plot[2].set_title("Image: Zoomed Result")

case_result[case_number - 1, 0:R, 0:C] = laplacian_filtered_image
case_mse[case_number - 1] = mean_squared_error(case_result[case_number - 1, 0:R, 0:C], img)

```

```

# Case 3: Mean Filter -> Median Filter -> Laplacian Filter

case_number = 3

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(img[0:200, 0:200], cmap='gray')
plot[1].set_title("Image: Zoomed Original")

mean_filtered_image = mean_filter(img)
median_filtered_image = median_filter(mean_filtered_image)
laplacian_filtered_image = laplacian_filter(median_filtered_image)

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(mean_filtered_image, cmap='gray')
plot[0].set_title("Image: Mean Filter Added")

plot[1].imshow(median_filtered_image, cmap='gray')
plot[1].set_title("Image: Median Filter Added")

```

```

plot[2].imshow(laplacian_filtered_image, cmap='gray')
plot[2].set_title("Image: Laplacian Filter Added")

plot[3].imshow(laplacian_filtered_image[0:200, 0:200], cmap='gray')
plot[3].set_title("Image: Zoomed Result")

case_result[case_number - 1, 0:R, 0:C] = laplacian_filtered_image
case_mse[case_number - 1] = mean_squared_error(case_result[case_number - 1, 0:R, 0:C], img)

```

```

# Case 4: Median Filter -> Mean Filter -> Laplacian Filter

case_number = 4

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(img[0:200, 0:200], cmap='gray')
plot[1].set_title("Image: Zoomed Original")

median_filtered_image = median_filter(img)
mean_filtered_image = mean_filter(median_filtered_image)
laplacian_filtered_image = laplacian_filter(mean_filtered_image)

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(median_filtered_image, cmap='gray')
plot[0].set_title("Image: Median Filter Added")

plot[1].imshow(mean_filtered_image, cmap='gray')
plot[1].set_title("Image: Mean Filter Added")

plot[2].imshow(laplacian_filtered_image, cmap='gray')
plot[2].set_title("Image: Laplacian Filter Added")

plot[3].imshow(laplacian_filtered_image[0:200, 0:200], cmap='gray')
plot[3].set_title("Image: Zoomed Result")

case_result[case_number - 1, 0:R, 0:C] = laplacian_filtered_image
case_mse[case_number - 1] = mean_squared_error(case_result[case_number - 1, 0:R, 0:C], img)

```

```

# All Cases Results + MSE Report

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')

```

```
plot.set_title("Image: Original")

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

for case in range (len(case_result)):
    plot[case].imshow(case_result[case], cmap='gray')
    plot[case].set_title("Image: Case " + str(case + 1))

print("MSE Report")
print("Case 1: Mean Filter -
> Laplacian Filter MSE = " + str(case_mse[0]))
print("Case 2: Median Filter -
> Laplacian Filter MSE = " + str(case_mse[1]))
print("Case 3: Mean Filter -> Median Filter -
> Laplacian Filter MSE = " + str(case_mse[2]))
print("Case 4: Median Filter -> Mean Filter -
> Laplacian Filter MSE = " + str(case_mse[3]))
```

تمرین ۳ سوال ۳.۴

چکیده

هدف از این تمرین آشنایی با برخی فیلتر های Edge Detection و میزان کارآمدی آنها و نحوه استفاده از آنهاست.

مقدمه

در این تمرین از دو نوع فیلتر مخصوص Edge Detection استفاده کرده و با مفاهیمی مثل 2D Gradiant آشنا می شویم. فیلتر های Edge Detection به صورتی عمل کرده که تمام قسمت های تصویر را تاریک و تنها قسمت هایی که دارای لبه هستند را روشن می کند. اما لبه ها در انواع حالات و زوايا هستند و نمی توان از یک فیلتر انتظار داشت تمام لبه ها در تمام جهات را تشخیص دهد. همچنین تصاویری که نیاز به استخراج ویژگی هستند، همیشه خالی از نویز نبوده و این نویز ها نیز لبه هایی دارند که از فیلتر ها انتظار می رود تا حد زیادی این لبه ها را تشخیص ندهند.

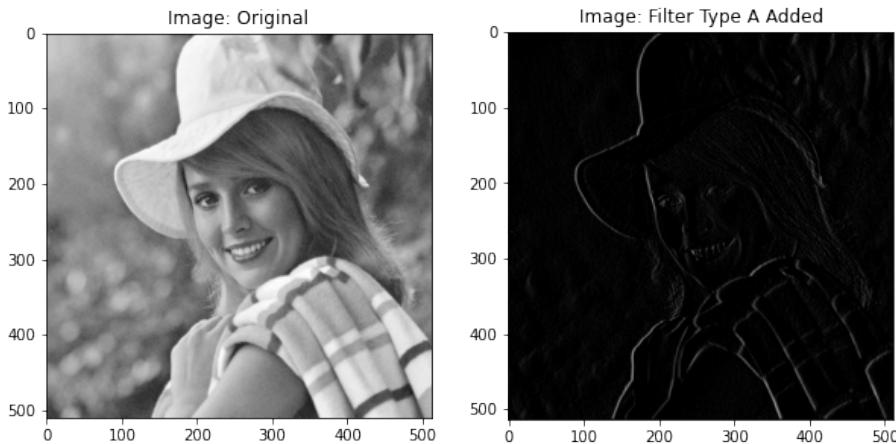
فیلتر First Order از نوع فیلتر هایی است که تا حد زیادی نویز پذیر است و می توان در تصاویر نتیجه خصوصا در ماتریس a نویز های زیادی را مشاهده کرد اما همزمان برای تشخیص لبه های عمودی بسیار مناسب هستند.

فیلتر های Robert نویز ها را در حد خوبی از بین می برد ولی فقط در جهات خاصی مثل 45° درجه در a و 135° درجه در b قابل استفاده است.

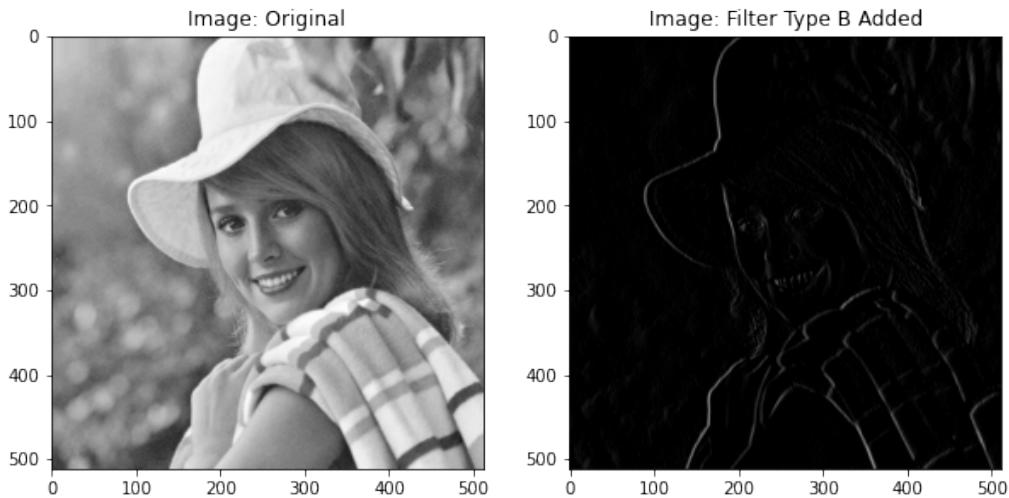
مفهوم 2D Gradiant به استخراج ویژگی ها مانند لبه ها از تصاویر اشاره می کند که می توان از فیلتر های بالا برای آنها استفاده کرد.

شرح نتایج

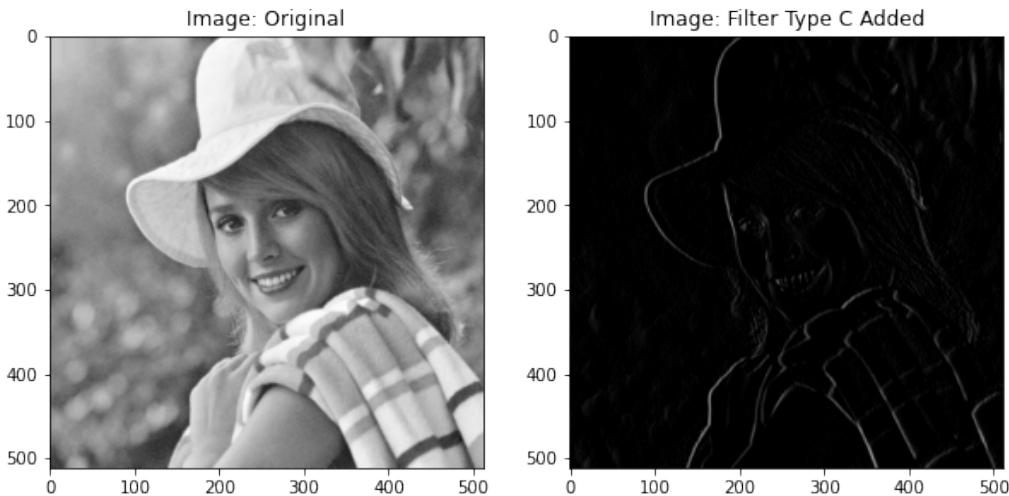
فیلتر First Order حالت



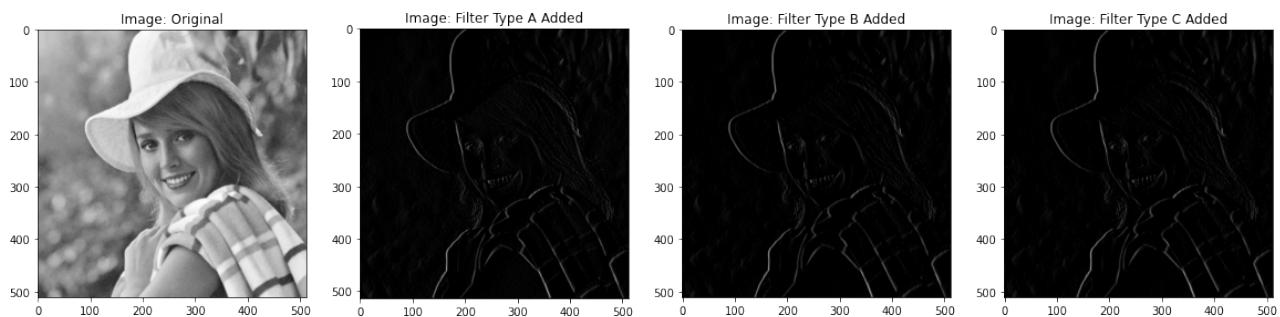
فیلتر First Order حالت B



فیلتر First Order حالت B

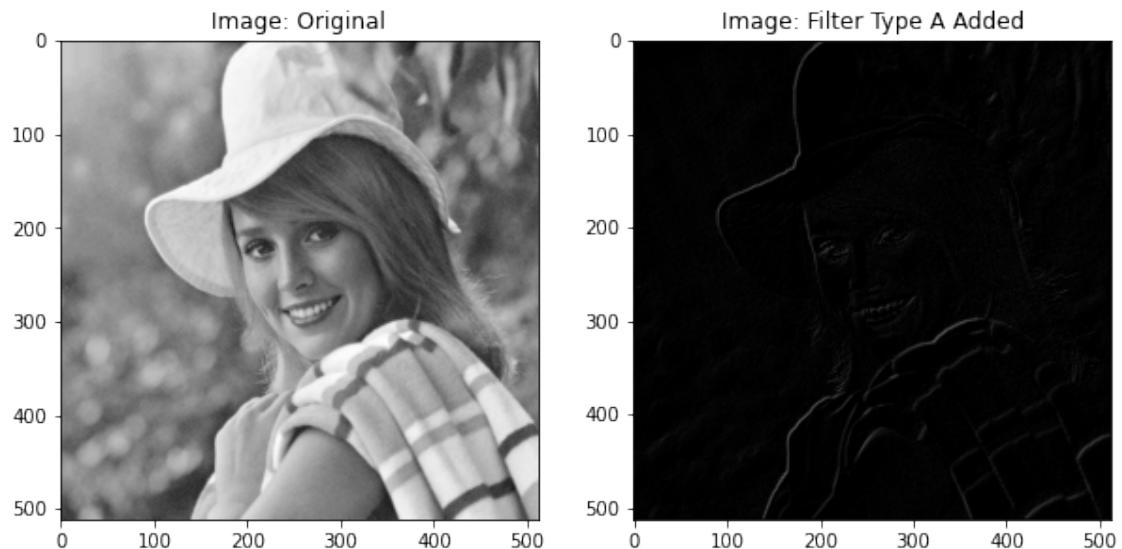


فیلتر First Order حالت A و B و C در کنار هم

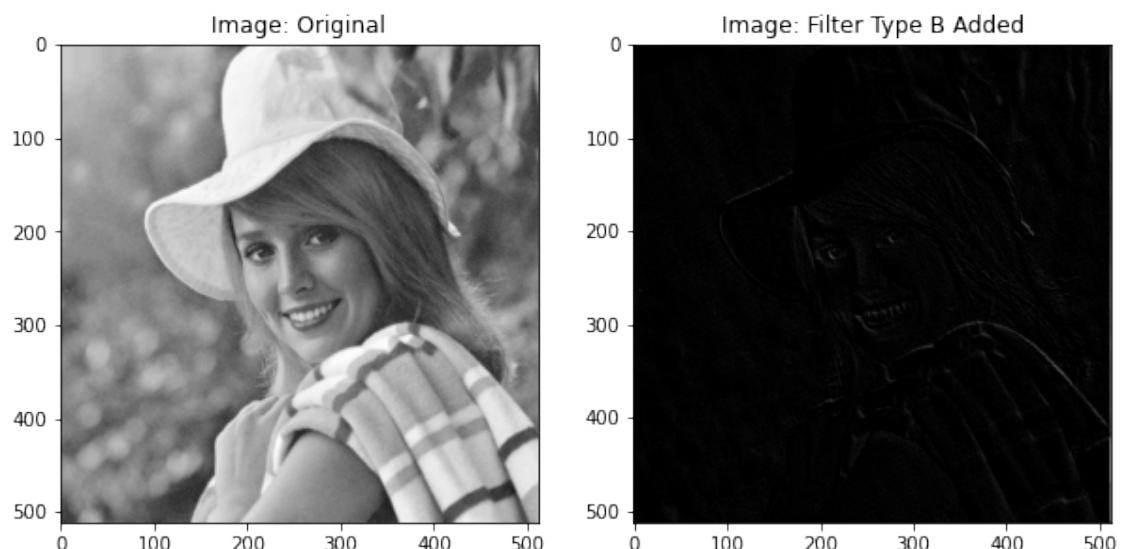


همانطور که مشاهده می شود در فیلتر نوع A و در قسمت های پیش زمینه (مانند پایین سمت چپ) نویز های زیادی دیده می شوند که در فیلتر های B و C از بین رفته اند.

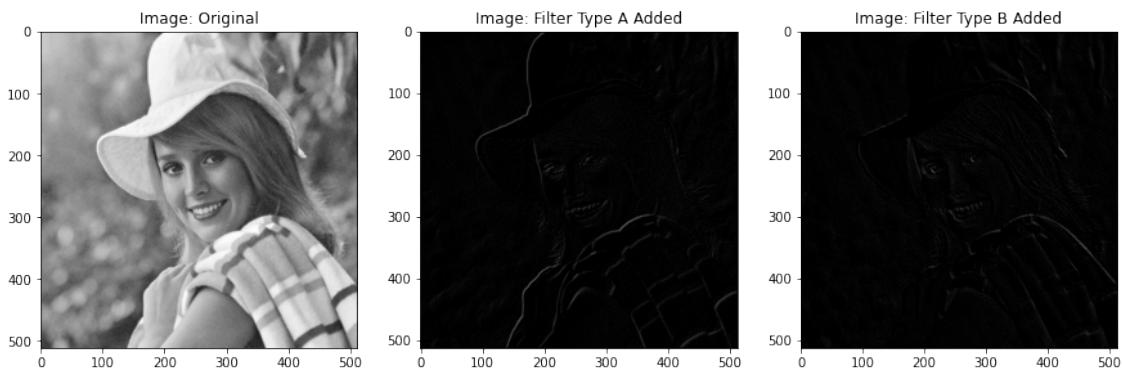
فیلتر Robert حالت A



فیلتر Robert حالت B



فیلتر Robert حالات A و B در کنار هم



همانطور که مشاهده می شود در فیلتر A لبه ها در زوایای ۴۵ درجه و در فیلتر B در زوایه ۱۳۵ درجه قابل تشخیص است.

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
import random
from sklearn.metrics import mean_squared_error
from statistics import median

img = cv2.imread('Elaine.bmp',0)
R, C = img.shape

def convolve2D(image, kernel, padding=0, strides=1):
    kernel = np.flipud(np.fliplr(kernel))

    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-(1 * padding))] = image
    else:
        imagePadded = image

    for y in range(imagePadded.shape[1]):
        if y > imagePadded.shape[1] - yKernShape:
            break
        if y % strides == 0:
            for x in range(imagePadded.shape[0]):
                if x > imagePadded.shape[0] - xKernShape:
                    break
                try:
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
                except:
                    break
            output[output>255]=255
            output[output<0]=0
    return output

```

```

def first_order_difference_filters (image, filter_type):
    if filter_type == 'a':
        filter_matrix = np.array([[1, 0, -1]])
        filter_matrix = filter_matrix / 2
    elif filter_type == 'b':
        filter_matrix = np.array([(1, 0, -1), (1, 0, -1), (1, 0, -1)])
        filter_matrix = filter_matrix / 6
    elif filter_type == 'c':
        filter_matrix = np.array([(1, 0, -1), (2, 0, -2), (1, 0, -1)])
        filter_matrix = filter_matrix / 8
    else:
        return

    first_order_difference_filtered_image = convolve2D(image, filter_matrix, 1)

    return first_order_difference_filtered_image.astype(int)

```

```

# 3.4.1 - Filter A

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

filtered_image_type_a = first_order_difference_filters(img, 'a')
filtered_mse_type_a = mean_squared_error(filtered_image_type_a[0: R, 0: C], img)

plot[1].imshow(filtered_image_type_a, cmap='gray')
plot[1].set_title("Image: Filter Type A Added")

```

```

# 3.4.1 - Filter B

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

filtered_image_type_b = first_order_difference_filters(img, 'b')
filtered_mse_type_b = mean_squared_error(filtered_image_type_b[0: R, 0: C], img)

plot[1].imshow(filtered_image_type_b, cmap='gray')
plot[1].set_title("Image: Filter Type B Added")

```

```

# 3.4.1 - Filter C

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

```

```

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

filtered_image_type_c = first_order_difference_filters(img, 'c')
filtered_mse_type_c = mean_squared_error(filtered_image_type_c[0: R, 0: C], img)

plot[1].imshow(filtered_image_type_c, cmap='gray')
plot[1].set_title("Image: Filter Type C Added")

```

```

# 3.4.1 - All Filters Together

fig, plot = plt.subplots(1, 4, figsize = (20, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(filtered_image_type_a, cmap='gray')
plot[1].set_title("Image: Filter Type A Added")

plot[2].imshow(filtered_image_type_b, cmap='gray')
plot[2].set_title("Image: Filter Type B Added")

plot[3].imshow(filtered_image_type_c, cmap='gray')
plot[3].set_title("Image: Filter Type C Added")

print("MSE Report")
print("Filter A = " + str(filtered_mse_type_a))
print("Filter B = " + str(filtered_mse_type_b))
print("Filter C = " + str(filtered_mse_type_c))

```

```

def robert_filters (image, filter_type):
    if filter_type == 'a':
        filter_matrix = np.array([(1, 0), (0, -1)])
    elif filter_type == 'b':
        filter_matrix = np.array([(0, 1), (-1, 0)])
    else:
        return

    robert_filtered_image = convolve2D(image, filter_matrix, 1)

    return robert_filtered_image.astype(int)

```

```

# 3.4.2 - Filter A

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

```

```
filtered_image_type_a = robert_filters(img, 'a')
filtered_mse_type_a = mean_squared_error(filtered_image_type_a[0: R, 0: C], img)

plot[1].imshow(filtered_image_type_a, cmap='gray')
plot[1].set_title("Image: Filter Type A Added")
```

```
# 3.4.2 - Filter B

fig, plot = plt.subplots(1, 2, figsize = (10, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

filtered_image_type_b = robert_filters(img, 'b')
filtered_mse_type_b = mean_squared_error(filtered_image_type_b[0: R, 0: C], img)

plot[1].imshow(filtered_image_type_b, cmap='gray')
plot[1].set_title("Image: Filter Type B Added")
```

```
# 3.4.2 - All Filters Together

fig, plot = plt.subplots(1, 3, figsize = (15, 5))

plot[0].imshow(img, cmap='gray')
plot[0].set_title("Image: Original")

plot[1].imshow(filtered_image_type_a, cmap='gray')
plot[1].set_title("Image: Filter Type A Added")

plot[2].imshow(filtered_image_type_b, cmap='gray')
plot[2].set_title("Image: Filter Type B Added")

print("MSE Report")
print("Filter A = " + str(filtered_mse_type_a))
print("Filter B = " + str(filtered_mse_type_b))
```

تمرین ۳ سوال ۳.۵

چکیده

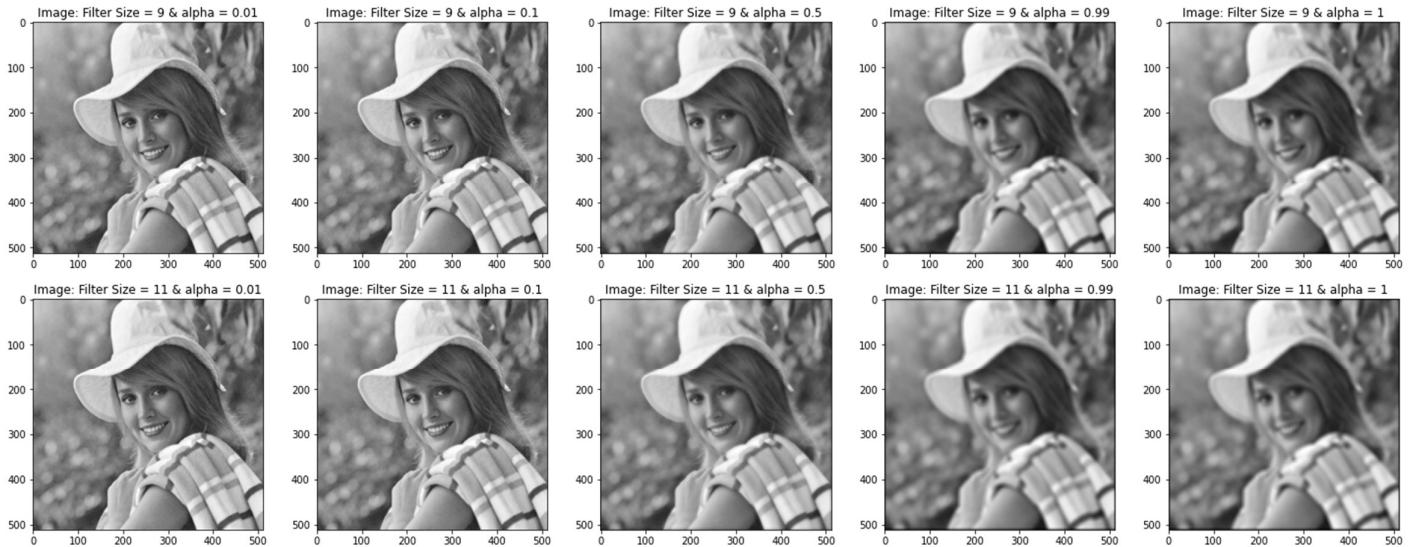
هدف این تمرین آشنایی با Unsharp Masking و نحوه کار کردن آن با تغییر مقادیر ورودی است.

مقدمه

در این تمرین ابتدا با استفاده از یک کرنل گوسی (کد کرنل را از اینترنت پیدا کردم)، تصویر را smooth کرده و سپس با اعمال فرمول گفته شده تصویر را Unsharp می کنیم.
پس از بدست آوردن Gaussian Kernel با سایز فیلتر مناسب، ماتریس آن را در تصویر اصلی کانوالو کرده و نتیجه را به عنوان I' و تصویر اصلی را به جای I در فرمول قرار داده و به جای مقادیر مختلف آلفا نتیجه را تست می کنیم.

شرح نتایج





همانطور که در تصویر دیده می شود، هرچه سایز فیلتر را بزرگتر کنیم، لبه های Sharp تری دریافت خواهیم کرد. از طرفی با تغییر آلفا، مشاهده می شود که هرچه مقدار آن بزرگتر باشد، لبه ها Sharp تر شده و همزمان نویز ها هم بیشتر قابل مشاهده است (چراکه لبه های نویز ها نیز Sharp تر می شوند). و به صورت برعکس هرچه مقدار آن کمتر باشد لبه ها به تیزی مقادیر بالاتر نبوده ولی همزمان نویز های کمتری در تصویر دیده می شود. به نظر من میزان Optimum آلفا را می توان مقدار وسط یعنی 0.5 در نظر گرفت چراکه لبه های شارپ تری نسبت به حالات قبل داشته و همزمان نویز کمتری نسبت به حالات بعدی دارد.

می توان مقادیر MSE را نیز در تصویر زیر مشاهده کرد.

```

Filter Size = 3 Alpha = 0.01 MSE = 0.005269786258163034
Filter Size = 3 Alpha = 0.1 MSE = 0.5269786258163036
Filter Size = 3 Alpha = 0.5 MSE = 13.174465645407587
Filter Size = 3 Alpha = 0.99 MSE = 51.64917511625592
Filter Size = 3 Alpha = 1 MSE = 52.697862581630346

Filter Size = 5 Alpha = 0.01 MSE = 0.008644496260631997
Filter Size = 5 Alpha = 0.1 MSE = 0.8644496260632
Filter Size = 5 Alpha = 0.5 MSE = 21.611240651579998
Filter Size = 5 Alpha = 0.99 MSE = 84.72470785045422
Filter Size = 5 Alpha = 1 MSE = 86.44496260631999

Filter Size = 7 Alpha = 0.01 MSE = 0.012066303495930807
Filter Size = 7 Alpha = 0.1 MSE = 1.20663034959308
Filter Size = 7 Alpha = 0.5 MSE = 30.165758739826998
Filter Size = 7 Alpha = 0.99 MSE = 118.26184056361777
Filter Size = 7 Alpha = 1 MSE = 120.66303495930799

Filter Size = 9 Alpha = 0.01 MSE = 0.016101318244163026
Filter Size = 9 Alpha = 0.1 MSE = 1.6101318244163028
Filter Size = 9 Alpha = 0.5 MSE = 40.25329561040757
Filter Size = 9 Alpha = 0.99 MSE = 157.80902011104183
Filter Size = 9 Alpha = 1 MSE = 161.01318244163028

Filter Size = 11 Alpha = 0.01 MSE = 0.02029618066633429
Filter Size = 11 Alpha = 0.1 MSE = 2.0296180666334283
Filter Size = 11 Alpha = 0.5 MSE = 50.740451665835714
Filter Size = 11 Alpha = 0.99 MSE = 198.9228667107423
Filter Size = 11 Alpha = 1 MSE = 202.96180666334286
  
```

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
import random
from sklearn.metrics import mean_squared_error
from statistics import median

img = cv2.imread('Elaine.bmp',0)

def convolve2D(image, kernel, padding=0, strides=1):
    kernel = np.flipud(np.fliplr(kernel))

    xKernShape = kernel.shape[0]
    yKernShape = kernel.shape[1]
    xImgShape = image.shape[0]
    yImgShape = image.shape[1]

    xOutput = int(((xImgShape - xKernShape + 2 * padding) / strides) + 1)
    yOutput = int(((yImgShape - yKernShape + 2 * padding) / strides) + 1)
    output = np.zeros((xOutput, yOutput))

    if padding != 0:
        imagePadded = np.zeros((image.shape[0] + padding*2, image.shape[1] + padding*2))
        imagePadded[int(padding):int(-1 * padding), int(padding):int(-1 * padding)] = image
    else:
        imagePadded = image

    for y in range(imagePadded.shape[1]):
        if y > imagePadded.shape[1] - yKernShape:
            break
        if y % strides == 0:
            for x in range(imagePadded.shape[0]):
                if x > imagePadded.shape[0] - xKernShape:
                    break
                try:
                    if x % strides == 0:
                        output[x, y] = (kernel * imagePadded[x: x + xKernShape, y: y + yKernShape]).sum()
                except:
                    break
    output[output>255]=255
    output[output<0]=0

    return output

```

```

def guassian_kernel(filter_size):
    ax = np.linspace(-
(filter_size - 1) / 2., (filter_size - 1) / 2., filter_size)
    xx, yy = np.meshgrid(ax, ax)

    sigma = 100
    kernel = np.exp(-
0.5 * (np.square(xx) + np.square(yy)) / np.square(sigma))

    return kernel / np.sum(kernel)

def sharpenning(image, filter_size):
    sharpened_image = convolve2D(image, guassian_kernel(filter_size), filter_size//2)
    return sharpened_image

def unsharpenning(image, filter_size, alpha):
    unsharpened_image = image + (alpha * (sharpenning(image, filter_size) - image))
    return unsharpened_image

fig, plot = plt.subplots(1, 1, figsize = (5, 5))

plot.imshow(img, cmap='gray')
plot.set_title("Image: Original")

filter_sizes = [3, 5, 7, 9, 11]
alphas = [0.01, 0.1, 0.5, 0.99, 1]

R, C = img.shape
unsharpened_mse = np.zeros((len(filter_sizes), len(alphas)))

fig, plot = plt.subplots(len(filter_sizes), len(alphas), figsize = (5 * len(alphas), 5 * len(filter_sizes)))

for filter_size in range (len(filter_sizes)):
    for alpha in range (len(alphas)):
        sharpened_image = unsharpenning(img, filter_sizes[filter_size], alphas[alpha])
        plot[filter_size][alpha].imshow(sharpened_image, cmap='gray')
        plot[filter_size][alpha].set_title("Image: Filter Size = " + str(filter_sizes[filter_size]) + " & alpha = " + str(alphas[alpha]))

        unsharpened_mse[filter_size][alpha] = mean_squared_error(sharpened_image, img)

# MSE Report
for filter_size in range (len(filter_sizes)):
```

```
for alpha in range (len(alphas)):
    print("Filter Size = " + str(filter_sizes[filter_size]) + " Alpha =
" + str(alphas[alpha]) + " MSE = " + str(unsharpened_mse[filter_size][
alpha]))
    print("\n")
```