**Computer Vision**
**Dr. Mohammadi**
**Fall 2022**
**Hoorieh Sabzevari - 98412004**
**HW4**

دانشگاه علم و صنعت ایران

.۱

1) $F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$ $\quad \begin{array}{|c|c|} \hline 2 & 3 \\ \hline 1 & 4 \\ \hline \end{array}$

$v=0$, $u=0$ : $F(u,v) = \sum_{x=0}^{1} \sum_{y=0}^{1} f(x,y) \times 1 = 2+3+1+4 = \boxed{10}$

$v=0$, $u=1$ : $F(u,v) = \sum_{x=0}^{1} \sum_{y=0}^{1} f(x,y)\, e^{-j\pi x} = 2\times 1 + 3 \times e^{-j\pi}$

$+ 1 \times 1 + 4 \times e^{-j\pi} = 2-3+1-4 = \boxed{-4}$

$v=1$, $u=0$ : $F(u,v) = \sum_{x=0}^{1} \sum_{y=0}^{1} f(x,y)\, e^{-j\pi y} = 2\times 1 + 3\times 1 + 1 \times e^{-j\pi}$

$+ 4 \times e^{-j\pi} = 2+3-1-4 = \boxed{0}$

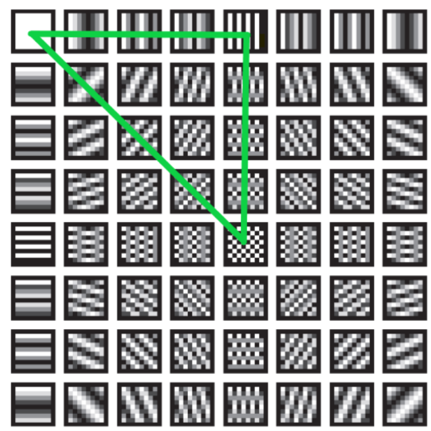
$v=1$, $u=1$ : $F(u,v) = \sum_{x=0}^{1} \sum_{y=0}^{1} f(x,y)\, e^{-j\pi(x+y)} = 2\times 1 + 3 \times e^{-j\pi}$

$+ 1 \times e^{-j\pi} + 4 \times e^{-2\pi j} = 2-3-1+4 = \boxed{2}$

$F(u,v) = \begin{array}{|c|c|} \hline 10 & -4 \\ \hline 0 & 2 \\ \hline \end{array}$

۲.

الف) طبق مثالی که در اسلایدها داشتیم می‌دانیم که تبدیل فوریه‌ی ماتریس $n \times n$ در بخش حقیقی یک ماتریس متقارن با چهار خط تقارن (دو خط قطر اصلی و فرعی و دو خط افقی و عمودی) است، لذا مثلا در همان مثال اسلاید تعداد مولفه‌های آزاد ماتریس مجموع خانه‌های مشخص‌شده در تصویر زیر یعنی ۱۵ می‌شود.



زیرا بقیه‌ی مقادیر با استفاده از تقارن بدست می‌آیند.

برای $n$ های زوج فرمول زیر برقرار است:

$$\frac{(\frac{n}{2}+1)(\frac{n}{2}+2)}{2}$$

و برای $n$ های فرد نیز فرمول زیر برقرار است:

$$\frac{(\frac{n+1}{2})(\frac{n+1}{2}+1)}{2}$$

ب) طبق رابطه‌ی زیر اگر $v$ و $u$ را صفر بگذاریم توان نمایی صفر شده و کل عبارت برابر با حاصل جمع شدت روشنایی‌های تمام نقاط می‌شود. جمع تمام فرکانس‌ها هم روشن‌تر از بقیه‌ی نقاط است.

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

این سوال همراه با همفکری آقای امیرحسین سماوات حل شده است.

.۳

الف) در این بخش ابتدا به تصویر اولیه به اندازه‌ی نصف ابعاد کرنل صفر اضافه کرده (zero-padding) و سپس برای هر پیکسل در دو حلقه‌ی تو در تو کانولوشن را حساب می‌کنیم و در تصویر نهایی به جای مقدار اولیه‌ی آن قرار می‌دهیم. برای پیاده‌سازی سیگما هم می‌توان از تابع آماده‌ی جمع در کتابخانه‌ی numpy استفاده کرد.

```
# https://medium.com/analytics-vidhya/2d-convolution-using-python-numpy-
43442ff5f381
def filter_2d(image, kernel):
    result = np.zeros(image.shape)
    ########################
    # Your code goes here. #

    x = image.shape[0]
    y = image.shape[1]
    kernelSize = kernel.shape[0]
    padding = int(kernelSize/2)
    new_x = x + 2*padding
    new_y = y + 2*padding

    zpImg = np.zeros((new_x, new_y))
    zpImg[int(padding):int(-1 * padding), int(padding):int(-
1 * padding)] = image

    for c in range(x):
      for r in range(y):
        result[c,r] = np.sum((kernel * zpImg[c: c + kernelSize, r: r + ker
nelSize]))
```

```
    ########################
    return result
```

ب) مقادیر هر خانه در فیلتر میانگین‌گیر یک تقسیم بر تعداد خانه‌های کرنل است یعنی مساحت آن.

```
def averaging_kernel(size):
    ########################
    # Your code goes here. #
    s = size * size
    result = np.ones((size,size),np.float32)/s
    ########################
    return result
```

پس از اعمال آن بر روی تصویر دارای نویز نمک و فلفل خروجی زیر را داریم:

```
[12] im = cv2.imread('salt_and_pepper_low.jpeg',0)
     plt.imshow(im)
     plt.axis('off')
```

(-0.5, 224.5, 224.5, -0.5)

```
kernel = averaging_kernel(3)
im_smoothed = filter_2d(im, kernel)
plt.imshow(im_smoothed)
plt.axis('off')
```

(-0.5, 224.5, 224.5, -0.5)

همانطور که مشاهده می‌کنید تصویر کمی تار شده و جزئیات کم‌اهمیت آن از بین رفته است. در واقع تصویر صاف و هموار شده است.

پ) در این بخش دقیقا مانند بخش الف عمل می‌کنیم و فقط در قسمت محاسبه بجای استفاده از تابع جمع، از تابع میانه استفاده می‌کنیم.

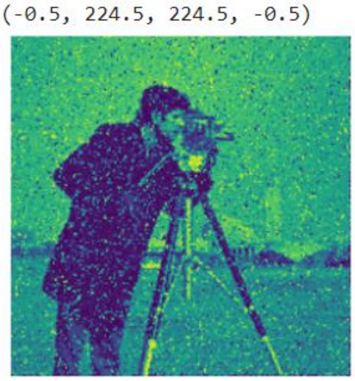
```
def median_filter(image, size):
    result = np.zeros(image.shape)
    ########################
    # Your code goes here. #
```

```
    x = image.shape[0]
    y = image.shape[1]
    kernelSize = size
    padding = int(kernelSize/2)
    new_x = x + 2*padding
    new_y = y + 2*padding

    zpImg = np.zeros((new_x, new_y))
    zpImg[int(padding):int(-1 * padding), int(padding):int(-
1 * padding)] = image

    for c in range(x):
      for r in range(y):
        result[c,r] = np.median(zpImg[c: c + kernelSize, r: r + kernelSize
])
    ########################
    return result
```

پس از اعمال این فیلتر روی تصویر خروجی زیر را داریم:

```
im = cv2.imread('salt_and_pepper_high.jpeg',0)
plt.imshow(im)
plt.axis('off')
```

```
(-0.5, 224.5, 224.5, -0.5)
```



```
im_smoothed = median_filter(im, size=3)
plt.imshow(im_smoothed)
plt.axis('off')
```

```
(-0.5, 224.5, 224.5, -0.5)
```

همانطور که می‌بینیم این فیلتر روی تصویری با نویز نمک و فلفل شدیدتری اعمال شده و تا حد خوبی توانسته نویز را کاهش دهد. فیلتر میانه‌گیر برای نویزهای نمک و فلفل که در یک نقطه شدت روشنایی خیلی زیاد و خیلی تیره می‌شود موثر است. چرا که معمولا میانه‌ی شدت روشنایی در یک شبکه‌ی کرنل خیلی نزدیک به رنگ اصلی آن پیکسل بدون نویز است و پس از مرتب‌سازی پیکسل‌ها نویز جزو پیکسل‌های بالایی یا پایینی قرار می‌گیرد. اما در فیلتر میانگین‌گیر، یک نقطه‌ی خیلی

روشن ممکن است میانگین را کاملا عوض کند و به همین علت باعث تاری می‌شود. نویز نمک و فلفل چون نویز جمع‌شونده نیست با فیلترهای خطی به خوبی رفع نویز نمی‌شود.

ت) طبق فرمول مشتق در اسلایدها، مشتق در راستای $y$ به صورت زیر است که ضریب ایگرگ بالایی ۰.۵- و ایگرگ پایینی ۰.۵ است. بقیه‌ی ضرایب هم صفر است.

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1) - f(x,y-1)}{2}$$

خروجی به صورت زیر است که لبه‌ها کاملا مشخص شده‌اند:

```
im_smoothed = filter_2d(im, derivative_kernel)
plt.imshow(im_smoothed)
plt.axis('off')
```

```
(-0.5, 255.5, 255.5, -0.5)
```

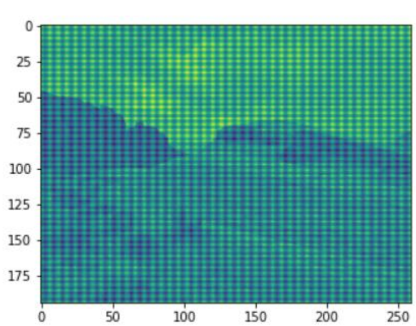
برای کم‌تر شدن تاثیر نویز می‌توان بسته به نوع نویز از روش‌های رفع نویز استفاده کرد. مثلا برای نویز نمک و فلفل از فیلتر میانه‌گیر استفاده کرد. همچنین می‌توانیم نویز را پس از تبدیل فوریه حذف کنیم که روش بسیار راحت‌تر و موثرتری است.

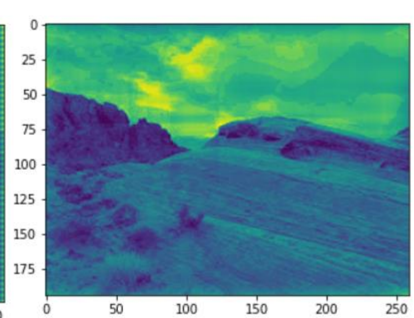
۴.

الف) در این بخش، ابتدا از تصویر تبدیل فوریه گرفته و سپس با دستور fftshift پیکسل‌ها را جوری شیفت می‌دهیم که مبدا روی مرکز تصویر بیفتد. در فضای فرکانسی، نویزها به صورت دو خط افقی و عمودی که از مرکز رد شده‌اند، نمایش داده می‌شوند. برای رفع نویز کافیست که این دو خط را رفع نویز کنیم. سپس دوباره شیفت معکوس داده و تبدیل فوریه‌ی معکوس می‌گیریم.

```
# https://towardsdatascience.com/image-processing-with-python-application-
of-fourier-transformation-5a8584dc175b
def denoise_image(image):
    denoised = image.copy()
    ########################
    # Your code goes here. #
    x = image.shape[0]
    y = image.shape[1]
    xCenter = x//2
    yCenter = y//2
    xLeft = xCenter - 2
    xRight = xCenter + 2
    yTop = yCenter - 2
    yBottom = yCenter + 2
    denoised = np.fft.fftshift(np.fft.fft2(image))
    denoised[xLeft:xRight, :yCenter - 8] = 0
    denoised[xLeft:xRight, yCenter + 8:] = 0
    denoised[:xCenter - 8, yTop:yBottom] = 0
    denoised[xCenter + 8:, yTop:yBottom] = 0
    denoised = abs(np.fft.ifft2(np.fft.fftshift(denoised)))
    ########################

    return denoised
```

خروجی این الگوریتم به صورت زیر است:



همانطور که می‌بینیم تصویر رفع نویز شده است.

ب) در سیستم‌های پردازش تصویر نسبت سیگنال به نویز تعریف متفاوتی دارد. در این حالت، صورت کسر برابر با مربع مقدار پیک سیگنال است و در مخرج کسر، توان نویز یا واریانس آن قرار می‌گیرد. به عنوان مثال، یک تصویر ۸ بیتی دارای مقادیر در بازه ۰ تا ۲۵۵ است. در نتیجه برای محاسبه پیک نسبت سیگنال به نویز یا PSNR، صورت در تمام موارد برابر با $۲۵۵^۲$ است. با انجام رفع نویز مخرج کسر کوچک شده و مقدار PSNR افزایش می‌یابد. (منبع)

```
PSNR between noisy image and original image =  8.122255096865844
PSNR between denoised image and original image =  27.96136067049111
```

پ) در کد موجود برای ساخت تصویر نویزی، نویز با تصویر اصلی جمع شده پس نویز جمع‌شونده است.

در کل دو نوع نویز داریم، نویز جمع‌شونده و نویز ضرب‌شونده. در نویز جمع‌شونده یک سیگنال نویز با مقدار اصلی تصویر جمع می‌شود اما در نویز ضرب‌شونده یک سیگنال نویز در مقدار اصلی تصویر ضرب می‌شود. حذف نویز ضرب‌شونده کار دشواری است زیرا راه‌حل‌های بازیابی تصویر کمی برای حذف این نوع نویز وجود دارد. نویز ضرب‌شونده دارای توزیع گاما و نویز جمع‌شونده دارای توزیع نرمال است. نویز جمع‌شونده تاثیر کمتری نسبت به نویز ضرب‌شونده دارد. لذا رفع آن آسان‌تر است. نویز گاوسی بهترین مدل نویز جمع‌شونده است. (منبع)