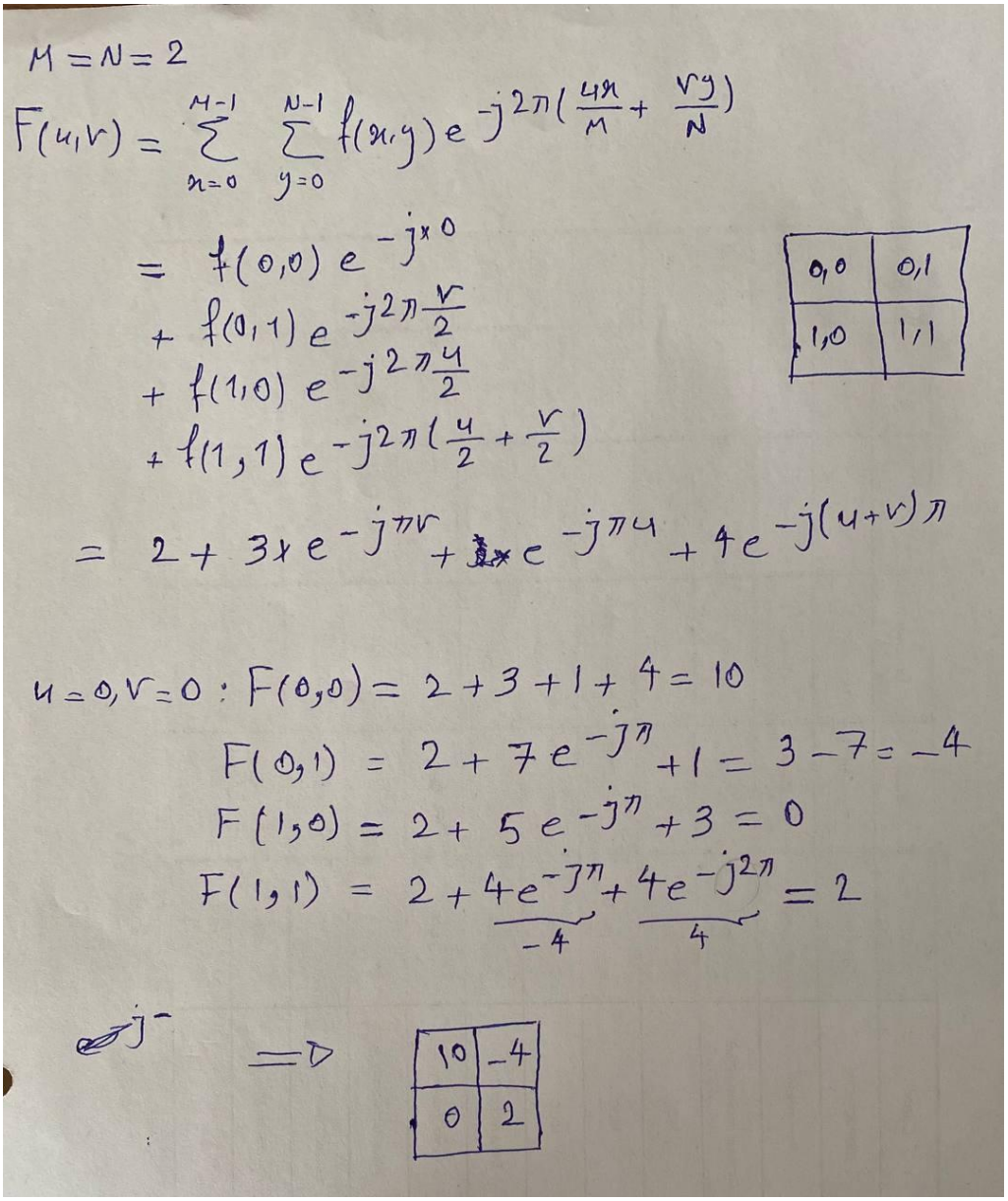
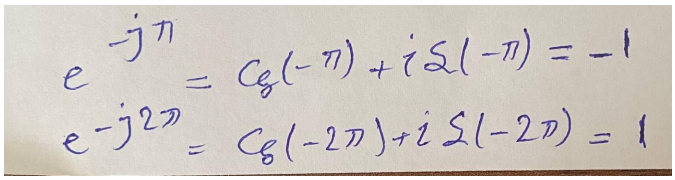
Computer vision

Homework 4

Parsa Eissazadeh 97412364

سوال 1





سوال 2

الف)

ب)

فرمول تبدیل فوریه بدین شکل بود :

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

اگر u و v صفر باشد ، مقدار $e^{-j2\pi(ux/M+vy/N)}$ برابر با 1 می شود . در نتیجه تبدیل فوریه در آن نقطه برابر است با :

$$\mathrm{F(u,v)} = \sum_{y=0}^{M-1}\sum_{x=0}^{N-1} f(x,y)$$

که برابر است با مجموع مقدار تابع f در همه نقاط . در نتیجه تبدیل فوریه در خانه 0،0 برابر است با مجموع روشنایی تمام پیکسل های عکس .

سوال 3

ساخت کرنل متوسط گیر : در کرنل متوسط گیر همه خانه ها مقدار 1 دارند (تا جمع بدون وزن روی شدت روشنایی پیکسل ها زده شود) سپس تقسیم بر تعداد پیکسل ها می شود تا به میانگین برسیم . تعداد پیکسل ها می شود سایز کرنل به توان 2 (در اینجا طول و عرض کرنل برابر است .)

کد بدین صورت است :

```
def averaging_kernel(size):
    """
    Returns an averaging kernel with the specified size.

    Parameters
    ----------
    size: int
        Width and height of the kernel.

    Returns
    -------
    ndarray
        The averaging kernel.
    """
    result = np.ones((size, size))
    ########################
    # Your code goes here. #
    ########################
    pixel_counts = size**2
    return result / pixel_counts
```

همانطور که مشاهده می شود تنها یک خط به متد اضافه شد .

برای convulsion ، در ابتدا باید یک padding به عکس اضافه کنیم (تا پیکسل های مرزی عکس همسایه داشته باشند) از متد np.pad استفاده می کنیم .

متد convolve را نوشتم که در آن کل خانه های کرنل در خانه های همسایگی یک پیکسل مشخص ضرب می شود و در نهایت با هم جمع می شوند .

```
def convolve (image , center_index , kernel , kernel_size):

    x_domain = int(np.floor(kernel_size[0]/2))
    y_domain = int(np.floor(kernel_size[1]/2))

    sum = np.multiply(image[
        center_index[0]- x_domain:center_index[0]+ x_domain+ 1 ,
        center_index[1]- y_domain:center_index[1]+ y_domain+ 1
    ],kernel).sum()

    return sum
```

Center_index موقعیت پیکسلی است که میخواهیم خانه های کرنل را دانه دانه در پیکسل های همسایه اش ضرب کنیم .

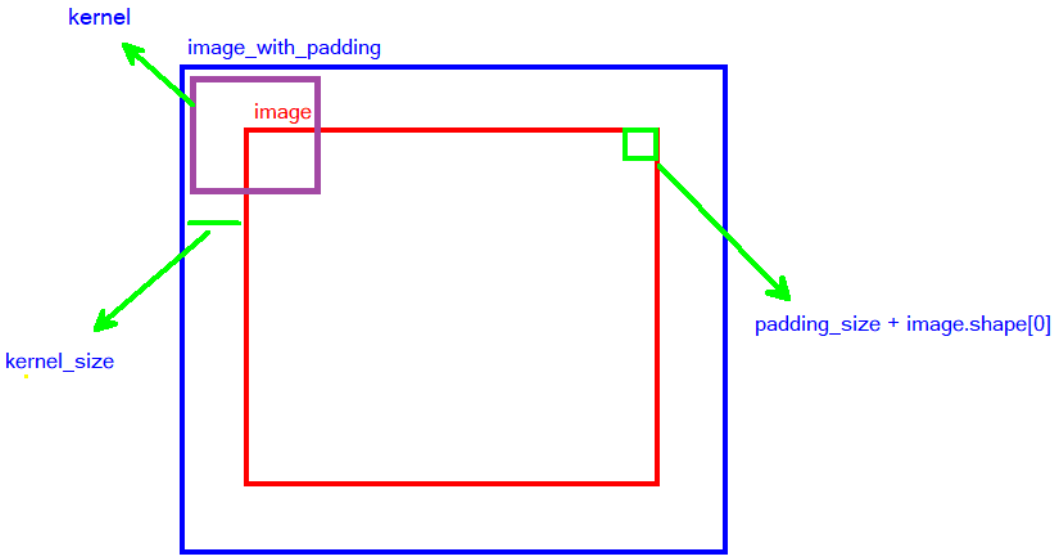
متد np.multiply دو آرایه را به عنوان ورودی می گیرد و آرایه ای را به عنوان خروجی باز میگرداند که در هر اندیس خانه اش ضرب خانه همان اندیس در آرایه های ورودی است . سپس با متد sum این ها را با هم جمع می زنیم و باز میگردانیم .

متد filter_2d :

```
def filter_2d(image, kernel):
    """
    Convolves an image with the kernel, applying zero-padding to maintain the size of the image.

    Parameters
    ----------
    image: ndarray
        2D array, representing a grayscale image.
    kernel: ndarray
        2D array, representing a linear kernel.
    Returns
    -------
    ndarray
        The result of convolving `image` with `kernel`.
    """
    result = np.zeros(image.shape)
    ########################
    # Your code goes here. #
    ########################

    # Adding padding
    padding_size = int(np.floor(kernel.shape[0] / 2))
    image_with_padding = np.pad(image , ((padding_size,padding_size) , (padding_size,padding_size)) , 'edge')
    print(image.shape)
    print(image_with_padding.shape)

    print(padding_size+image.shape[0])

    # Convolution
    for i in range(padding_size , padding_size+image.shape[0] -1  ):
      for j in range(padding_size , padding_size+image.shape[1] -1 ):
        result[i-1][j-1] = convolve(image_with_padding ,(i,j) ,kernel ,kernel.shape )

    return result
```

کار اصلی در این متد انجام می شود :

همانطور که گفته شد ابتدا padding را به عکس اضافه میکنیم . طول padding باید به اندازه kernel باشد :



به اندازه اندازه کرنل تقسیم بر ۲ ، از پیکسل مورد نظر ، بالا ، پایین ، چپ و راست میرویم .

خروجی بدین شکل شد :

```
[41] im = cv2.imread('/content/salt_and_pepper_low.jpeg', cv2.IMREAD_GRAYSCALE)
     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)
```

عکس اندکی تار تر شد ولی نویز هم بسیار کمتر شد . اگر اندازه پنجره را بزرگتر کنیم مشاهده می شود که عکس بیشتر تار می شود و در نتیجه نویز بیشتری از دست میرود :

برای مثلا برای اندازه کرنل 5 :

```
kernel = averaging_kernel(5)
im_smoothed = filter_2d(im, kernel)
plt.imshow(im_smoothed)
plt.axis('off')
```

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

برای حذف نویز نمک و فلفل گرفتن میانه راه حل بهتری است . همانند قسمت قبل padding اضافه میکنیم ،

```
def median_filter(image, size):
    """
    Applies the median filter to the image with the given window size.

    Parameters
    ----------
    image: ndarray
        2D array, representing a grayscale image.
    size: int
        Size of the window for median calculation.
    Returns
    -------
    ndarray
        The result of convolving `image` with `kernel`.
    """
    result = np.zeros(image.shape)
    ########################
    # Your code goes here. #
    ########################

    # add padding
    padding_size = int(np.floor(kernel.shape[0] / 2))
    image_with_padding = np.pad(image , ((padding_size,padding_size) , (padding_size,padding_size)) , 'edge')

    for i in range(padding_size, padding_size + image.shape[0] - 1):
      for j in range(padding_size, padding_size + image.shape[1] - 1 ):
        result[i-1][j-1] = np.median(image[
          i - padding_size:i+ padding_size+ 1 ,
          j- padding_size:j+ padding_size+ 1
        ])

    return result
```

پیمایش عکس به کمک ماتریس های هم اندازه کرنل همانند بخش قبل است . نتیجه :

```
im = cv2.imread('/content/salt_and_pepper_high.jpeg',cv2.IMREAD_GRAYSCALE)
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)
```

نویز عکس به مقدار زیادی کاهش یافته .

برای اینکه در یک راستا مشتق بگیریم ، باید از معادله زیر استفاده کنیم :

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

برای اینکه این رفتار را شبیه سازی کنیم ، کرنل باید به صورت زیر باشد :

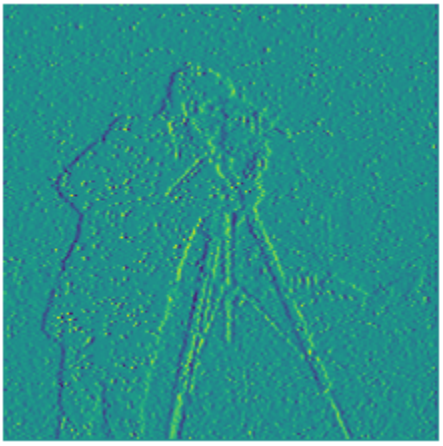
0	0	0
1/2-	0	1/2
0	0	0

بعد از اعمال کردن این فیلتر در هر پیکسل ½ * (f(x-1) - f(x+1)) باقی می ماند که مشتق تصویر است .
متد کانوالو کردن یک کرنل در یک عکس در بخش های قبلی پیاده سازی شده .

کد :

```
derivative_kernel = np.array([
    [0, 0, 0],
    [-.5, 0, .5],
    [0, 0, 0],
])

########################
# Your code goes here. #
########################
im_smoothed = filter_2d(im, kernel)
plt.imshow(im_smoothed)
plt.axis('off')
```

نتیجه :



در مکان هایی که رنگ پیکسل ها تغییر می کنند ، مشتق عکس مقداری غیر صفر دارد . در لبه های اشیا رنگ پیکسل ها تغییر می کنند . طبق این نکته عکس بالا به درستی نشان دهنده ی مشتق عکس است زیرا لبه های مرد فیلم بردار و پایه های دوربینش نمایان است .

این فیلتر مشتق گیر در عکس بدون نویز عملکرد خیلی بهتری داشت :

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

```
(256, 256)
(258, 258)
257
(-0.5, 255.5, 255.5, -0.5)
```

سوال 4

```
def denoise_image(image):
    """
    Denoises the input image.
    ----------
    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result denoised image.
    """

    denoised = image.copy()
    ########################
    # Your code goes here. #
    ########################

    dft = np.fft.fft2(denoised)
    dft_shifted = np.fft.fftshift(dft)

    dft_shifted[:,134:] = 0
    dft_shifted[:,:90] = 0
    dft_shifted[134:,:] = 0
    dft_shifted[:90,:] = 0

    idft_shift = np.fft.ifftshift(dft_shifted)
    idft = np.fft.ifft2(idft_shift)

    denoised = np.real(idft)
    return denoised
```

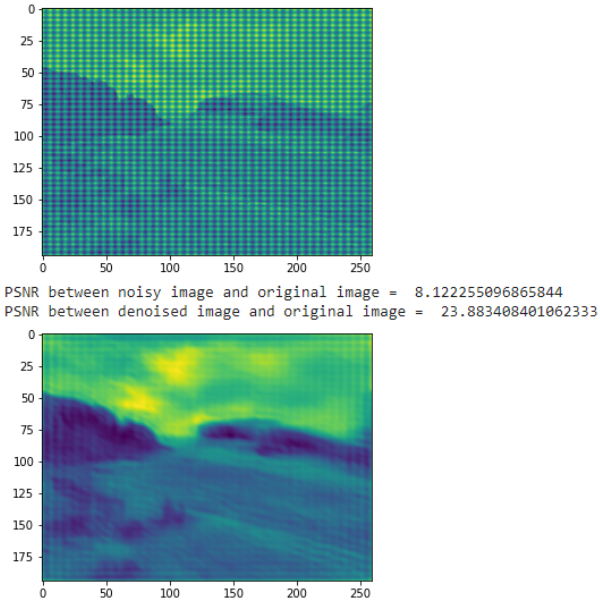
در ابتدا ، تبدیل فوریه میگیریم و شیفتش می دهیم به گونه ای که نقطه قوی در وسط تصویر بیفتد .

از آنجایی که ابعاد هر عکس 255 در 255 است و نقاط مرکزی که فرکانس صفر دارند عکس ما هستند ، ما پیکسل هایی که فاصله شان از مرکز از حدی بیشتر است حذف می کنیم .

این نقاط در راستای x ها کمتر از 90 و بیشتر از 134 هستند . (اعداد مختلفی را امتحان کردم ولی در این نقاط نویز کمترین مقدار و عکس بیشترین وضوح را داشت .)

Mask کردن به صورت دستی انجام شده است .

بقیه توابع از قبل نوشته شده بودند . خروجی :



هر چه مقدار PSNR بیشتر باشد مقدار efficiency تبدیل بیشتر است . در اینجا در ابتدا PSNR مقدار 8 داشته اما بعد از حذف کردن نویز به مقدار 23.8 (حدودا سه برابر رسیده است)

اگر از متد f + g تبدیل فوریه بگیریم به H می رسیم که برابر خواهد بود با F + G که F تبدیل فوریه f و G تبدیل فوریه g است .

اما تبدیل فوریه f*g پیچیده است . ما در اینجا به راحتی توانستیم با صفر کردن مقدار تبدیل فوریه در جاهایی که از مرکز دورند ، تبدیل فوریه نویز را حذف کنیم . این بدین معنی است که نویز جمع شونده بوده است . و ما بعد از صفر کردن G به F رسیدیم .