به نام خدا

درس مبانی بینایی کامپیوتر

تمرین سری چهارم

مدرس درس:
جناب آقای دکتر محمدی

تهیه شده توسط:
الناز رضایی ۹۸۴۱۱۳۸۷

تاریخ ارسال: ۱۴۰۱/۰۸/۰۶

سوال ۱:

مقدار پیکسل‌های تصویر ۲ در ۲ به صورت زیر است. تبدیل فوریه این تصویر را حساب کنید.(۱۵ نمره)

۳	۲
۴	۱

پاسخ ۱:

در اینجا تبدیل فوریه دو بعدی داریم، بنابراین باید از فرمول زیر استفاده کنیم:

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

f(x,y) مطابق ماتریس زیر می‌باشد:

$$f(x,y) = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

حال، با توجه به f(x,y) داده شده، به محاسبه تبدیل فوریه می‌پردازیم:

$$F(0,0) = \sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)e^{-j2\pi(0+0)} = \sum_{x=0}^{1} f(x,0) + f(x,1) = f(0,0)$$
$$+ f(0,1) + f(1,0) + f(1,1) = 2+3+1+4 = 10$$

$$F(0,1) = \sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)e^{-j2\pi(0+\frac{y}{2})} = \sum_{x=0}^{1} f(x,0) + f(x,1)e^{-j\pi}$$
$$= f(0,0) + f(0,1)e^{-j\pi} + f(1,0) + f(1,1)e^{-j\pi} = 2 + (3*(-1)$$
$$+1 + (4*(-1)) = -4$$

$$F(1,0) = \sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)e^{-j2\pi(\frac{x}{2}+0)} = \sum_{x=0}^{1} f(x,0)e^{-j\pi x} + f(x,1)e^{-j\pi x}$$
$$= f(0,0) + f(0,1) + f(1,0)e^{-j\pi} + f(1,1)e^{-j\pi} = 2+3+(1*(-1))$$
$$+(4*(-1)) = 0$$

$$F(1,1) = \sum_{x=0}^{1}\sum_{y=0}^{1} f(x,y)e^{-j2\pi(\frac{x}{2}+\frac{y}{2})} = \sum_{x=0}^{1} f(x,0)e^{-j\pi x} + f(x,1)e^{-j\pi(x+1)}$$
$$= (f(0,0) + f(0,1)e^{-j\pi} + f(1,0)e^{-j\pi} + f(1,1)e^{-j\pi 2}) = 2 + (3*(-1))$$
$$+(1*(-1)) + (4*1) = 2$$

بنابراین با توجه به مقادیر به دست آمده از این بخش، تبدیل فوریه این تصویر، به شرح زیر می‌باشد:

$$F(u,v) = \begin{bmatrix} 10 & -4 \\ 0 & 2 \end{bmatrix}$$

سوال ۲:

- الف) تصویری با ابعاد n × n داریم حاصل این تصویر در بخش حقیقی دامنه‌ی فرکانسی آن که ماتریسی n × n است در چه ابعادی در فضا قرار دارد؟ چرا؟ (تعداد مولفه‌های آزاد یک تصویر، ابعاد آن در فضا را نشان می‌دهد. برای مثال ابعاد یک ماتریس n × n قطری که فقط در قطر اصلی خود مقدار دارد، برابر n، و ابعاد یک ماتریس متقارن، $n * \frac{n-1}{2} + n = n * \frac{n+1}{2}$ است؛ زیرا n مولفه آزاد در قطر اصلی، و $\frac{n^2-n}{2}$ مولفه آزاد در بقیه ماتریس‌ها دارد.) (۵ نمره)
- ب) نقطه‌ی مبدا (۰،۰)تبدیل فوریه تصویر چه رابطه‌ای با مقادیر تصویر دارد؟ با روابط موجود جوابتان را ثابت کنید. (۱۰ نمره)

پاسخ ۲:

- ۲ـ الف) چون در سوال در مورد بخش حقیقی خواسته شده است، بنابراین فقط cos داریم و می‌دانیم $cos(-\theta) = cos(\theta)$. بنابراین ماتربس متفارن است. پس طبق توضیحات داده شده در صورت سوال، ابعاد ماتریس، $n * \frac{n+1}{2}$ می‌شود.
- ۲ـ ب)

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

در نتیجه همانطور که در رابطه بالا نیز مشخص است، F(0,0) مجموع همه نقاط تصویر می‌باشد.

سوال ۳:

برای این تمرین می‌بایست بخش Q۳ در نوتبوک ضمیمه را مطابق توضیحات داخل نوتبوک تکمیل کرده و به پیوست پاسخ‌های کتبی خود ارسال کنید؛ (۵۰ نمره)

- الف) در این بخش از سوال می‌بایست تابعی را تکمیل کنید که برای محاسبه نتیجه اعمال یک کرنل به یک تصویر سیاه‌وسفید استفاده خواهد شد. ورودی‌ها و خروجی‌های این تابع در کامنت‌ها شرح داده شده‌اند.
- ب) در بخش دوم، هدف پیاده‌سازی تابعی برای تولید کرنل‌های میانگین‌گیر با ابعاد دلخواه است. بعد از آن از این کرنل‌ها برای صاف‌کردن تصویری با نویز نمک و فلفل استفاده خواهید کرد.

- پ) در جریان بخش سوم سوال با تغییر پیاده‌سازی اولیه‌تان برای بخش A تابعی طراحی خواهید کرد که فیلتر میانه‌گیر را روی یک تصویر اعمال خواهد کرد. با اعمال این فیلتر به تصویری با نویز نمک و فلفل شدیدتر، مزایا و معایب این فیلتر نسبت به فیلترهای میانگین‌گیر را بررسی کنید.

- ت) در بخش نهایی سوال، شما می‌بایست فیلتری طراحی کنید که مشتق یک تصویر را در راستای افقی یا عمودی (نه هر دو) محاسبه کند. بدین منظور به رابطه‌های معرفی‌شده در اسلاید ۱۸ از جلسه پنجم مراجعه کنید. علاوه بر این، پیشنهادات خود برای کاهش اثر نویز تصویر در خروجی نهایی را نیز شرح دهید.

پاسخ ۳:

- ۳ـالف) می‌دانیم در فیلتر خطی، حاصل کانولوشن تصویر و کرنل محاسبه می‌شود که از فرمول زیر به دست می‌آید:

$$g(x,y) = \sum_{s=-1}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

بنابراین تنها کافیست کد مربوط به تابع بالا را، پیاده‌سازی کنیم.

```python
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
    -------
    result: ndarray
        The result of convolving `image` with `kernel`.
    """
    result = np.zeros(image.shape)
    ########################
    # Your code goes here. #
    ########################
    h, w = image.shape
    a = int(kernel.shape[0]/2)
    # copyMakeBorder(src, top, bottom, left, right, borderType, None, value)
    # we used this function to add extra padding for image
    image = cv2.copyMakeBorder(image, a, a, a, a, cv2.BORDER_DEFAULT)
    for j in range(a, h + a):
        for i in range(a, w + a):
            result[j - a, i - a] = (image[j - a:j + a + 1, i - a:i + a + 1] * kernel).sum()
    return result
```
✓ 0.2s

نکته: برای حل این بخش، از لینک زیر، کمک گرفته شد.

https://pyimagesearch.com/2016/07/25/convolutions-with-opencv-and-python/

- ۳ـ ب) برای پیاده‌سازی کرنل میانگین‌گیر، باید همه پیکسل‌ها را بر اندازه کرنل تقسیم کنیم. بنابراین برای این بخش، از کد زیر استفاده می‌کنیم.

```python
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. #
    ########################
    for i in range(size):
        for j in range(size):
            result[i,j] = result[i,j]/(size*size)
    return result
```
✓ 0.1s

در شکل زیر، تصویر را قبل و بعد از اعمال این فیلترها مشاهده می‌کنید. (تصویر سمت چپ، قبل از اعمال فیلتر و تصویر سمت راست، مربوط به بعد از آن می‌باشد.)

- ۳ـپ) در این بخش، باید مقادیر خانه‌ها در هر کرنل را sort کرده و سپس میانه آن را پیدا کنیم. برای انجام این بخش، از کد زیر استفاده می‌کنیم.

```python
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. #
    ########################
    h, w = image.shape[:2]
    a = size//2
    temp=[]
    # copyMakeBorder(src, top, bottom, left, right, borderType, None, value)
    # we used this function to add extra padding for image
    image = cv2.copyMakeBorder(image, a, a, a, a, cv2.BORDER_REPLICATE)
    for j in range(a, h + a):
        for i in range(a, w + a):
            temp = np.array(image[j - a:j + a + 1, i - a:i + a + 1])
            temp = temp.flatten()
            temp.sort()
            result[j - a, i - a] = temp[len(temp)//2]
            temp =[]
    return result
```
✓ 0.2s

در شکل زیر، تصویر را قبل و بعد از اعمال این فیلتر مشاهده می‌کنید. (تصویر سمت چپ مربوط به قبل از اعمال فیلتر، و تصویر سمت راست مربوط به بعد از آن می‌باشد.)

همانطور که از نتایج مربوط به دو بخش قبل نیز مشخص است، فیلتر میانه‌گیر نسبت به فیلتر میانگین‌گیر در این تصویر، عملکرد بهتری داشت. دلیل این اتفاق، این است که در تصاویری که دارای نویز نمک و فلفل هستند، یک نقطه ممکن است به علت نویز، سفید شده باشد و باقی نقاط همسایه‌اش، سیاه باشند، در این صورت وقتی از فیلتر میانگین‌گیر استفاده کنیم، یک نقطه نسبتا طوسی به ما می‌دهد که باز هم رفع نویز نمی‌شود. اما وقتی از فیلتر میانه‌گیر استفاده کنیم، این مشکل حل می‌شود. همچنین فیلتر میانگین‌گیر، تصویر را بیشتر تار می‌کرد. به طور کلی فیلترهای هموارساز خطی برای نویز نمک و فلفل، مناسب نیستند و فیلتر میانه‌گیر که جمع شونده نیست و غیرخطی است، بهتر است.

- ۳ــت) در این بخش، با استفاده از فرمول زیر، ضرایب را به‌گونه‌ای تغییر می‌دهیم که مشتق حساب شود.

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

تغییرات درایه‌ها، به شرح زیر است:

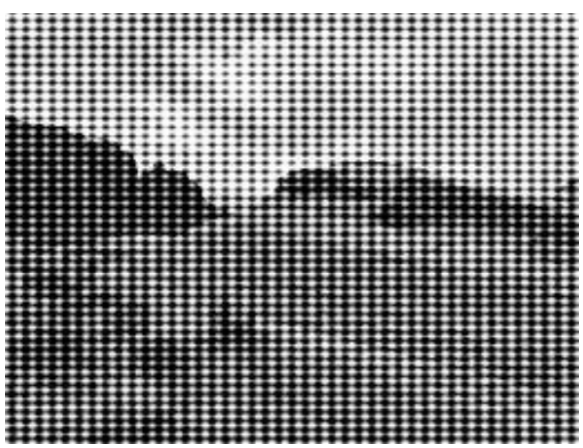
```
derivative_kernel = np.array([
    [0, 0, 0],
    [0.5, 0, -0.5],
    [0, 0, 0],
])

########################
# Your code goes here. #
########################
✓ 0.2s
```

همچنین تغییرات تصویر قبل و بعد از مشتق گرفتن، در تصویر زیر آورده شده است. (تصویر سمت چپ مربوط به قبل از مشتق‌گیری و تصویر سمت راست، بعد از مشتق گرفتن است.)

سوال ۴:

در بخش Q4 نوتبوک ضمیمه شده، به تصویر “png.image_original“ نویز اضافه شده است. (۴۰ نمره)

- الف) نویز تصویر حاصل را در حوزه فرکانسی(با استفاده از توابع موجود ماژول FFT از کتابخانه numpy)حذف کنید ؛ تصویر حاصل را ذخیره کنید و مراحل الگوریتم خود را شرح دهید.
- ب) در انتها با استفاده از تابع PSNR بهبود حاصل در تصویر را شرح دهید.
- پ) نویز اضافه شده به تصویر از چه نوعی است؟ (ضرب‌شونده یا جمع شونده)، تفاوت این دو نوع را توضیح دهید.

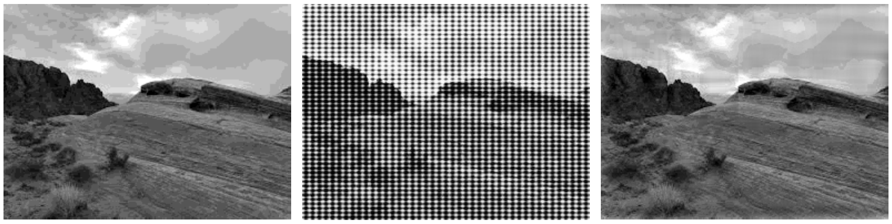


پاسخ ۴:

- ۴ــالف) در این بخش، ابتدا با استفاده از fft2، تبدیل فوریه تصویر را محاسبه کرده و سپس با تابع fftshift، صفرها را از گوشه‌ها به وسط انتقال می‌دهیم. در ادامه برای حذف نویزها، خط وسط عرضی و طولی را به دست آورده و به اندازه k که تقریبا عدد کوچکی است، مقدار خانه‌های آن را برابر صفر قرار می‌دهیم. چرا که نویز ها روی خط وسط عرضی و طولی می‌افتند و ما برای حذف آن، مقادیر این نقاط را برابر با صفر قرار می‌دهیم. کد مربوط به این بخش در عکس زیر آورده شده است.

```
from turtle import ontimer

from scipy import fftpack


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. #
    ########################
    denoised = fftpack.fftshift(fftpack.fft2(denoised))
    w, h = denoised.shape[0]//2 , denoised.shape[1]//2
    k = 3
    denoised[w - k: w + k, :h - k * k] = 0
    denoised[w - k: w + k, h + k * k:] = 0
    denoised[:w - k - k * k, h - k: h + k] = 0
    denoised[w + k + k * k:, h - k: h + k] = 0
    denoised = np.real(fftpack.ifft2(fftpack.ifftshift(denoised)))
    return denoised
```
✓ 0.4s

در بخش زیر نیز، به ترتیب از چپ به راست، تصویر اصلی، تصویر نویزی و تصویر رفع نویز شده مشخص شده‌اند.



- ۴ـ ب) نسبت سیگنال به نویز پیک، PSNR نسبت بین حداکثر توان ممکن یک تصویر و توان نویز مخرب است که بر کیفیت نمایش آن، تاثیر می‌گذارد. برای تخمین PSNR، باید تصویر را با یک تصویر تمیز با حداکثر توان ممکن مقایسه کرد. مقدار PSNR از رابطه زیر محاسبه می‌شود:

$$PSNR = 20\log_{10}^{(\frac{L-1}{RMSE})}$$

در فرمول بالا، L نشان‌دهنده تعداد حداکثر سطوح شدت ممکن و MSE به معنای mean squar error می‌باشد.
با توجه به توضیحات داده شده، هر چه مقدار PSNR بیشتر باشد، یعنی در رفع نویز، عملکرد بهتری داشته‌ایم. به بیان ساده‌تر، هر چه این مقدار بزرگتر باشد (مخرج به صفر میل کند) یعنی تصویر نهایی به تصویر اصلی نزدیکتر بوده و کیفیت بهتری ارائه می کند. و هر چه به صفر نزدیک باشد نشان دهنده این است که تصویر نهایی اطلاعات خیلی بیشتری را از دست داده است و خیلی کم شبیه تصویر اصلی است. همانطور که در تصویر زیر مشاهده می‌شود، مقدار PSNR تصویر نویزی و تصویر اولیه، 8.12 است، در حالی‌که این مقدار برای تصویر رفع نویز شده و تصویر اصلی، برابر 28.47 است.

```
PSNR between noisy image and original image =  8.122255096865844
PSNR between denoised image and original image =  28.477611106832704
```

- ۴ـ پ) نویز اضافه شده به تصویر، از نوع جمع‌شونده می‌باشد.
تفاوت‌های نویز جمع‌شونده و ضرب‌شونده:

۱. نویز جمع‌شونده، سیگنال ناگهانی ناخواسته‌ای است که به برخی از سیگنال‌های original اضافه می‌شود؛ اما نویز ضرب‌شونده، سیگنال ناگهانی ناخواسته‌ای است که در سیگنال original ضرب می‌شود.

۲. مدل نویز جمع‌شونده، به شرح زیر است:

Noisy_image[t] = original_image[t] + noise[t]

در حالی‌که مدل نویز ضرب‌شونده به این شکل است:

Noisy_image[t] = original_image[t] * noise[t]

۳. حذف نویز جمع‌شونده از ضرب‌شونده آسان‌تر است؛ زیرا مدل‌های بازیابی تصویر زیادی برای این نوع وجود دارد.
۴. منابع اصلی نویز جمع‌شونده، در زمان استفاده از عکس، و منابع اصلی نویز ضرب‌شونده، در حین ضبط، انتقال یا پردازش‌های دیگر است.
۵. تاثیر نویز بر تصویر در نویز جمع‌شونده، کمتر از ضرب‌شونده است؛ زیرا در نویز جمع‌شونده، سیگنال نویز به سیگنال اصلی اضافه می‌شود، ولی در نویز ضرب‌شونده، نویز چند برابر می‌شود.
۶. نویز جمع‌شونده، توزیع نرمال، و نویز ضرب‌شونده، توزیع گاما دارد.
۷. نویز لکه‌ای، یک مثال خوب برای نویز ضرب‌شونده است.
۸. نویز ضرب‌شونده، عمدتا در تصاویر رادار و ultrasound یافت می‌شود.