영상처리 실습 보고서

4주차: filtering

학번	201802170
이름	하 상 호

1. 과제의 내용

• average filter 및 sharpening filter 구현

•1차 Gaussian filter 완성 및 2차 Gaussian filter와 시간 비교

2. 과제의 해결 방법

과제의 내용을 해결하기 위해 어떠한 방법을 사용했는지, 자세하게 기술한다.

과제 1

■ 2개의 mask 작성 함수 및 filtering 함수 작성하여 작성 방법

```
def get_average_mask(fshape):
    print('get average filter')
    ########################################################
    # TODO                                                 #
    # mask 완성                                            #
    ########################################################
    (k_h, k_w) =fshape
    k_size = k_h * k_w
    mask = np.full((k_h, k_w), 1 / k_size)

    # mask 확인
    print(mask)

    return mask
```

average_mask

fshape를 통해 h , w 값을 받고 , mask(kernel) size를 선언

mask에서 np.full을 통해 k_h , k_w 사이즈에 1/k_size를 삽입

전체 사이즈를 나눈값을 모든 pixel 에 삽입하여 합이 1이 되도록 만든다.

```python
def get_sharpening_mask(fshape):
    print('get sharpening filter')
    ##################################################
    # TODO                                           #
    # mask 완성                                      #
    ##################################################

    (k_h, k_w) = fshape
    k_size = k_h * k_w
    mask = np.full((k_h, k_w), 1 / k_size)

    k_sh = np.zeros((k_h, k_w))
    k_sh[k_h // 2][k_w // 2] = 2
    mask = k_sh - mask
    print(fshape)


    #mask 확인
    print(mask)

    return mask
```

sharpening_mask

본 mask 는 이론 수업에 나온 것을 토대로 구현

image를 선명하게 해주는 효과를 기대할 수 있다.

0	0	0
0	2	0
0	0	0

−

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

그림으로 설명 하자면 위와 같다.

k_sh 는 sharpening mask를 만들기 위한 변수선언

k_sh 는 한가운데값을 2로 삽입
average_mask 를 빼주고 해당 mask를 return 한다.

```python
def my_filtering(src, mask, pad_type='zero'):
    (h, w) = src.shape
    src_pad = my_padding(src, (mask.shape[0]//2, mask.shape[1]//2), pad_type)
    dst = np.zeros((h, w))
    ############################################################
    # TODO                                                     #
    # dst 완성                                                 #
    # dst : filtering 결과 image                               #
    ############################################################

    #if stride 1
    (i_h, i_w) = src_pad.shape
    # print("srcpad : " , i_h , i_w)

    (k_h, k_w) = mask.shape
    (p_size_h, p_size_w) = (mask.shape[0]//2, mask.shape[1]//2)

    o_h = (h - k_h + 2 * p_size_h) // 1 + 1
    o_w = (w - k_w + 2 * p_size_w) // 1 + 1

    #print("out : " , o_h , o_w)

    for i in range(o_h):
        for j in range (o_w):
            cut_img = src_pad[i:i + k_h, j:j + k_w]
            dst[i, j] = np.sum(mask * cut_img)

    #pixel save
    dst = np.where(dst > 255, 255, dst)
    dst = np.where(dst < 0, 0, dst)


    dst = (dst+0.5).astype(np.uint8)

    return dst
```

filterring 작업을 수행

먼저 입력 받은 이미지에 padding을 수행한다.
커널의 h,w 패딩을 얼마나 해줬는지 확인하여 변수에 저장한다.

$$O = \frac{I - K + 2P}{S} + 1$$

해당 함수식을 사용하여 output image 의 사이즈를 알아낸다.
I 는 인풋 사이즈, k 는 커널 사이즈 p 는 패딩 사이즈 s 는 stride 다.

2중 포문을 사용하여 필터링 과정을 진행한다.
이전 실습에서 배웠던 차원을 나누는 개념을 사용하여 필터링 진행
곱한 값을 더해주며 dst 변수에 삽입한다.

3x3 average_filter 3x3 sharpening filter

두 필터를 비교해 보았을 때 aver 필터는 전체적으로 흐리게 나오고,
sharpening 필터는 굉장히 선명하게 나온 것을 확인 할 수 있다.

11x13 average filter, 11x13 sharpening filter

위 두 필터 역시 3x3 필터와 느낀점은 비슷하나, 흐린정도 와 선명도가 확실히 달랐다. 위 두 필터가 더 전체적으로 흐리거나 엄청나게 선명해진 것을 확인 할 수 있었다.

2d gaussian mask와 1d gaussian mask를 만드는 함수 작성 및 방법

본 과제는 이론, 실습에서 배운 내용을 가지고 구현하였다.

- **2D Gaussian filter**

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

$x : -n \sim n$ 범위의 mask에서의 x좌표(열)
$y : -n \sim n$ 범위의 mask에서의 y좌표(행)
σ : Gaussian 분포의 표준편차

n = mask의 행or열 길이// 2
ex) mask의 크기가 5이면
n = 5//2 = 2

위 해당 내용을 그대로 사용하여 구현

```
ksize = msize // 2
y, x = np.mgrid[-ksize:ksize + 1, -ksize:ksize + 1]
```

k_size = n
y,x를 통해 차원을 차원을 나누고 metrix를 구현한다.

```
#2차 gaussian mask 생성
gaus2D = (1 / (2 * np.pi * sigma ** 2)) * (np.e ** (-((x**2 + y**2) / 2 * sigma ** 2)))
#mask의 총 합 = 1
gaus2D /= np.sum(gaus2D)
```

위 이론 내용을 그대로 구현하면 된다.

1D gaussian filter 역시 그대로 구현하면 된다.

$$G(x) = (\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}})$$

```
ksize = msize // 2
x =np.mgrid[-ksize:ksize + 1, -ksize:ksize + 1]
# print(x)
#print(x[1,0])
#print(x[1, 1])
#print(x[0, 0])
#print(x[0, 1])
#print(x[0, 2])
#print(x[0, 3])
'''

x = np.full((1, 3), [-1, 0, 1])
x = [[ -1, 0, 1]]

x = np.array([[-1, 0, 1]])
x = [[ -1, 0, 1]]
'''
gaus1D = (1 / (((2 * np.pi)**(1/2)) * sigma)) * (np.e ** (-((x[1, 0] ** 2) / 2 * sigma ** 2)))

#mask의 총 합 = 1
gaus1D /= np.sum(gaus1D)
return gaus1D

return gaus1D
```

1D gaussian filter vs 2D gaussian filter

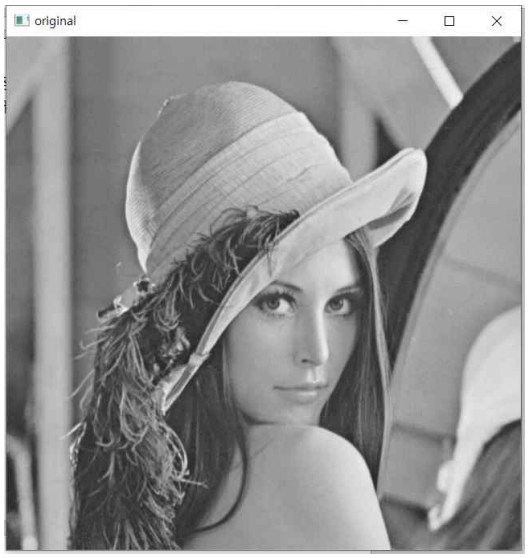
사진을 확인 하였을 때 솔직한 감상으로는 차이를 잘 모르겠다.

```
C:\Users\USER\anaconda3\python.e
mask size :  13
1D gaussian filter
1D time :  0.04849060000000005
2D gaussian filter
2D time :  0.32071289999999997
```

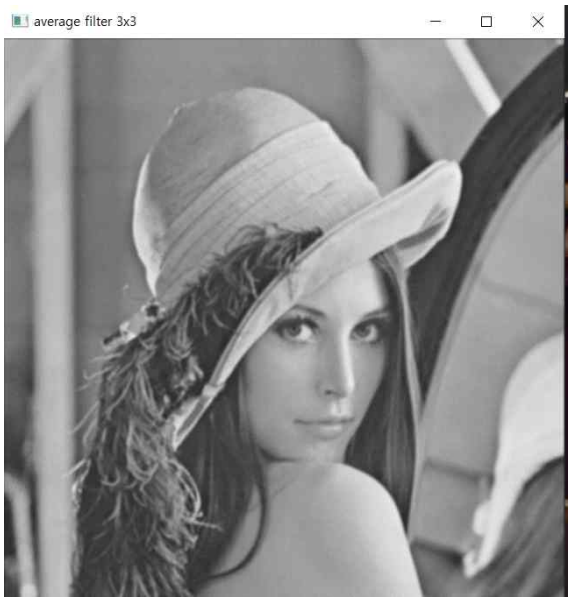
하지만 속도의 차이를 보았을 때 2d 가 더 오래걸린 것을 알 수 있다.

3. 결과물

결과물이 잘 보이도록 화면을 캡처해 보고서에 올린다.

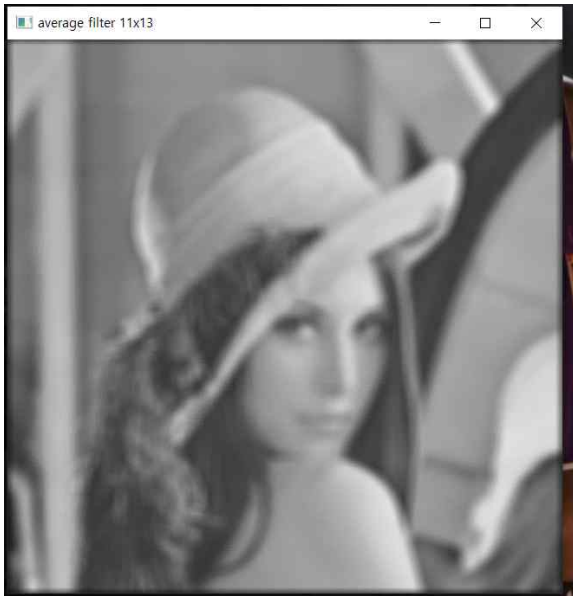


average_mask 3x3



sharpening_mask 3x3

average_mask 11x13



sharpening_mask 11x13

1D gaussian filter

2D gaussian filter