Image Processing
실습 4주차

신 동 헌

Department of Computer Science and Engineering

Chungnam National University, Korea

실습 소개

- **과목 홈페이지**
  - 충남대학교 사이버 캠퍼스 (http://e-learn.cnu.ac.kr)

- **TA 연락처**
  - 공대 5호관 531호 컴퓨터비전 연구실
  - 과제 질문은 [IP]를 제목에 붙여 메일로 주세요.
  - 00반
    - 안준혁
    - ajh99345@gmail.com
  - 01반
    - 신동헌
    - doghon85@naver.com

목차

- **2주차 과제 리뷰**

- **실습**
  - Image filtering
  - Average filter
  - Padding
  - Filter 총합에 따른 밝기
  - Sharpening filter

- **과제**
  - Filtering 구현
  - Gaussian filter 분석

2주차 과제

- **myVideo_bgr2gray.py**
  - 컬러 비디오를 흑백 비디오로 전환하여 저장하는 과제

```python
def calc_bgr2gray(src):
    h, w, c = src.shape
    dst = np.zeros((h, w))

    # B * 0.0721 + G * 0.7154 + R * 0.2125
    for row in tqdm(range(h)):
        for col in range(w):
            dst[row, col] = src[row, col, 0] * 0.0721 + src[row, col, 1] * 0.7154 + src[row, col, 2] * 0.2125

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

```python
if cap.isOpened(): # 객체 초기화 정상
    fps = 30.0  # FPS, 초당 프레임 수
    fourcc = cv2.VideoWriter_fourcc(*'DIVX')  # 인코딩 포맷 문자
    h, w = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)), int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

    # 영상 저장을 위핸 객체 생성
    out = cv2.VideoWriter('./record_gray.avi', fourcc, fps, (w, h), isColor=False)
```

Image filtering

- **Image filtering**
  - 이미지의 각 지점에서 지역적인 영역을 상정하고, 그 영역내에 있는 픽셀 값들과 kernel(혹은 fliter)값들의 내적 연산(dot product) 또는 가중합(weighted sum)
  - 이미지 픽셀의 값을 변화시킬 때 주변 이웃한 픽셀들의 값도 고려
  - 이미지로부터 중요한 정보(질감, 엣지 등)를 추출 가능

0	0	0	0	0
15	16	0	0	0
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4

X

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

=

3	3	2	0	0
6	7	6	4	3
7	9	8	7	5
4	6	7	8	6
1	2	3	4	3

Image filtering

- **Image filtering 원리**
  - $g(x,\ y) = \sum_{s=-k}^{k}\sum_{t=-k}^{k} w(s,t) f(x+s, y+t)$
    - $g(x,\ y)$: output
    - $w(s,t)$: kernel
    - $f(x+s, y+t)$: input

1/9	1/9	1/9			
1/9	1/9	1/9	0	0	0
1/9	1/9	1/9	0	0	0
	10	11	12	13	14
	5	6	7	8	9
	0	1	2	3	4

0×1/9 +
0×1/9 +
… +
15×1/9 +
16×1/9 = 3
(마지막 filtering이면 반올림)

3	3	2	0	0
6	7	6	4	3
7	9	8	7	5
4	6	7	8	6
1	2	3	4	3

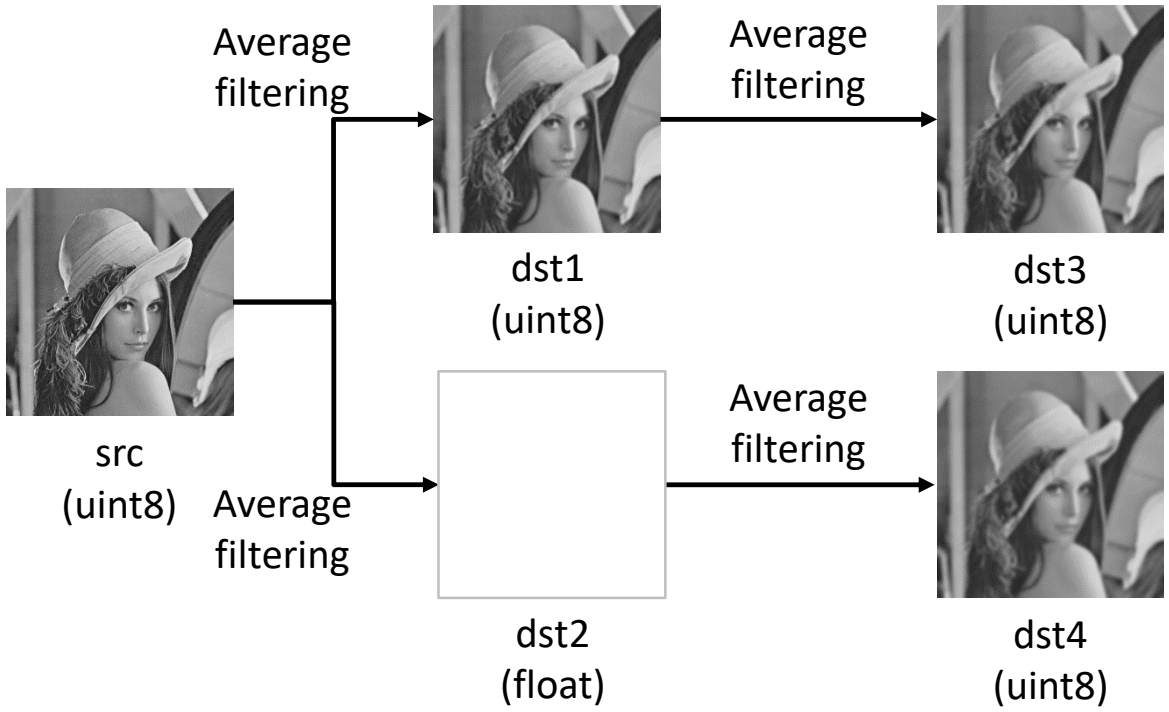
Image filtering

- **Filtering 실습**
  - cv2.filter2D(src, ddepth, kernel)
    - src: 이미지
    - ddepth: 출력 영상의 데이터 타입을 지정
      -1 이면 입력과 동일
      cv2.CV_32F이면 float32 형태로 출력
    - kernel: filtering에 사용될 kernel

```
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena.png', cv2.IMREAD_GRAYSCALE)
    kernel = np.ones((7, 7))
    kernel = kernel / np.sum(kernel)
    print(np.sum(kernel))

    dst1 = cv2.filter2D(src, -1, kernel)

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst1', dst1)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

Image filtering

- **Filtering 중 데이터 타입의 중요성**
  - dst1: 데이터 타입을 uint8로 유지
  - dst2: 데이터 타입을 적절하게 변환



Filtering 과정에서 데이터 타입을 잘 명시하지 않으면 값의 차이가 발생

```python
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena.png', cv2.IMREAD_GRAYSCALE)
    kernel = np.ones((7, 7))
    kernel = kernel / np.sum(kernel)
    print(np.sum(kernel))

    dst1 = cv2.filter2D(src, -1, kernel)
    dst2 = cv2.filter2D(src, cv2.CV_32F, kernel)

    dst3 = cv2.filter2D(dst1, -1, kernel)
    dst4 = cv2.filter2D(dst2, -1, kernel)

    dst4 = np.clip(dst4+0.5,  a_min: 0,  a_max: 255).astype(np.uint8)

    print('dst3:', dst3[0, 30], dst3[0, 42], dst3[0, 52])
    print('dst4:', dst4[0, 30], dst4[0, 42], dst4[0, 52])

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst1', dst1)
    cv2.imshow( winname: 'dst2', dst2)
    cv2.imshow( winname: 'dst3', dst3)
    cv2.imshow( winname: 'dst4', dst4)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

```
dst3: 154 168 166
dst4: 155 167 165
```

출력 결과

Average filter

- **Average filtering**
  - 이미지를 부드럽게 해주는 효과
  - 잡음을 제거하는데 사용

Original

Noisy image

3×3
average filter

9×9
average filter

15×15
average filter

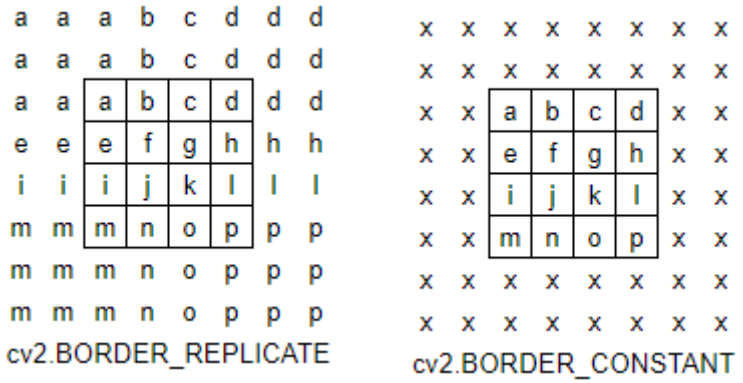
Average filter

- **Average filtering 실습**
  - cv2.filter2D(src, ddepth, kernel, borderType)
    - src: 이미지
    - ddepth: 출력 영상 데이터 타입을 지정
      -1 이면 입력과 동일
      cv2.CV_32F이면 float32 형태로 출력
    - kernel: filtering에 사용될 kernel
    - borderType: 이미지 가장자리 픽셀 확장 방식

```python
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena.png', cv2.IMREAD_GRAYSCALE)
    kernel = np.ones((3, 3))
    kernel = kernel / np.sum(kernel)
    print(np.sum(kernel))
    dst1 = cv2.filter2D(src, -1, kernel)
    dst2 = cv2.filter2D(src, -1, kernel, borderType=cv2.BORDER_REPLICATE)
    dst3 = cv2.filter2D(src, -1, kernel, borderType=cv2.BORDER_CONSTANT)

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst1', dst1)
    cv2.imshow( winname: 'dst2', dst2)
    cv2.imshow( winname: 'dst3', dst3)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

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

3 × 3 kernel



borderType 종류

Average filter

- **Average filtering noise 제거 실습**
  - Filter 크기에 따른 noise 제거 확인

3×3
average filter

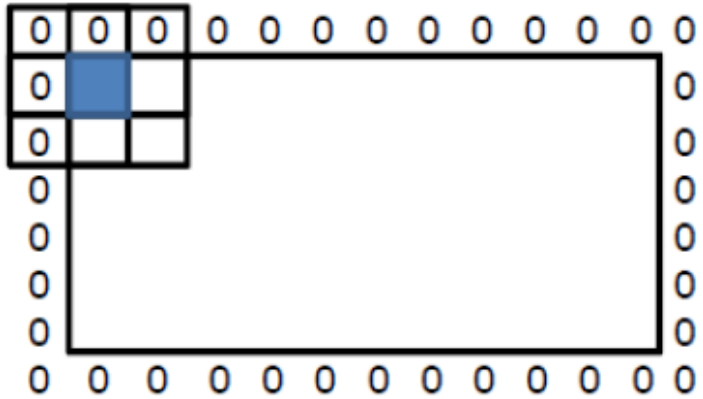
9×9
average filter

15×15
average filter

```python
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena_noise.png', cv2.IMREAD_GRAYSCALE)

    kernel1 = np.ones((3, 3))
    kernel1 = kernel1 / np.sum(kernel1)
    dst1 = cv2.filter2D(src, -1, kernel1,
                        borderType=cv2.BORDER_CONSTANT)

    kernel2 = np.ones((9, 9))
    kernel2 = kernel2 / np.sum(kernel2)
    dst2 = cv2.filter2D(src, -1, kernel2,
                        borderType=cv2.BORDER_CONSTANT)

    kernel3 = np.ones((15, 15))
    kernel3 = kernel3 / np.sum(kernel3)
    dst3 = cv2.filter2D(src, -1, kernel3,
                        borderType=cv2.BORDER_CONSTANT)

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst1', dst1)
    cv2.imshow( winname: 'dst2', dst2)
    cv2.imshow( winname: 'dst3', dst3)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

Padding

- **Zero padding**
  - 실제 이미지에 없는 가장자리 부분을 0으로 채움



Zero padding 이미지

Padding

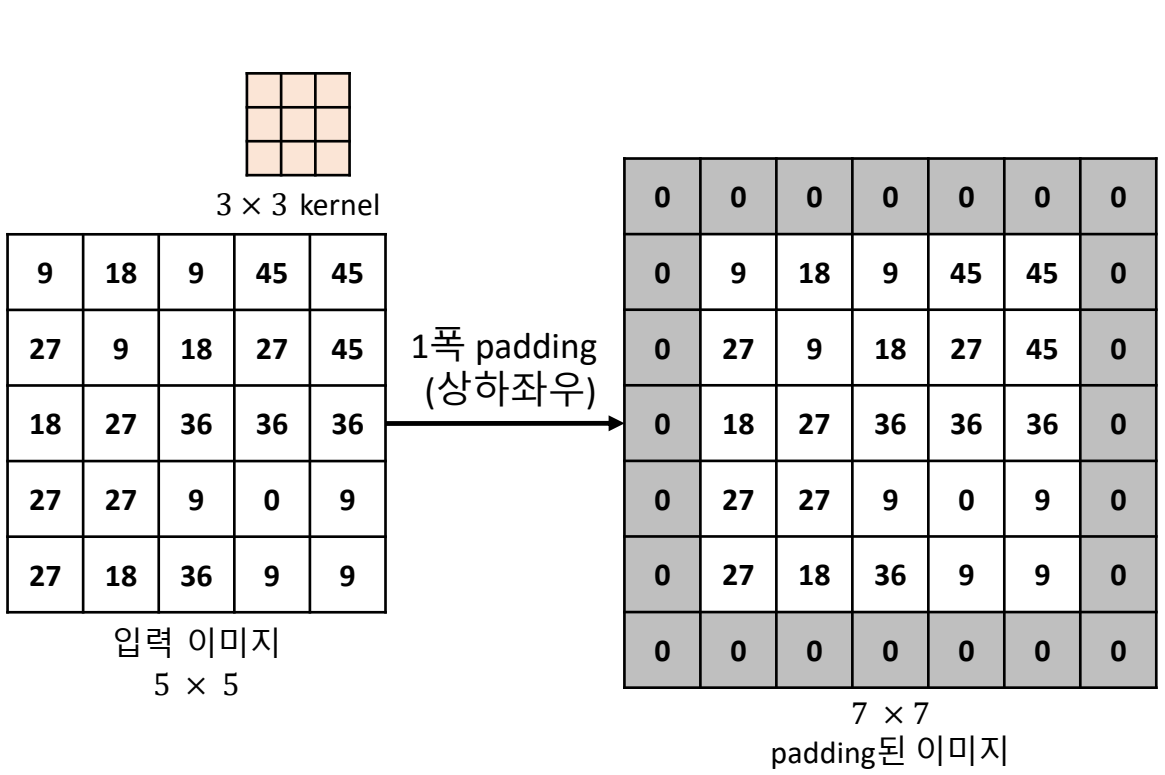
- **Repetition padding**
  - 실제 이미지에 없는 가장자리 부분을 가장자리의 값으로 채움

2	2	3	4	5	4	3	2	1	2	3	4	4
2	2	3	4	5	4	3	2	1	2	3	4	4
3	3										3	3
4	4										4	4
5	5										5	5
6	6										6	6
7	7	6	5	4	3	2	1	2	3	4	5	5
7	7	6	5	4	3	2	1	2	3	4	5	5

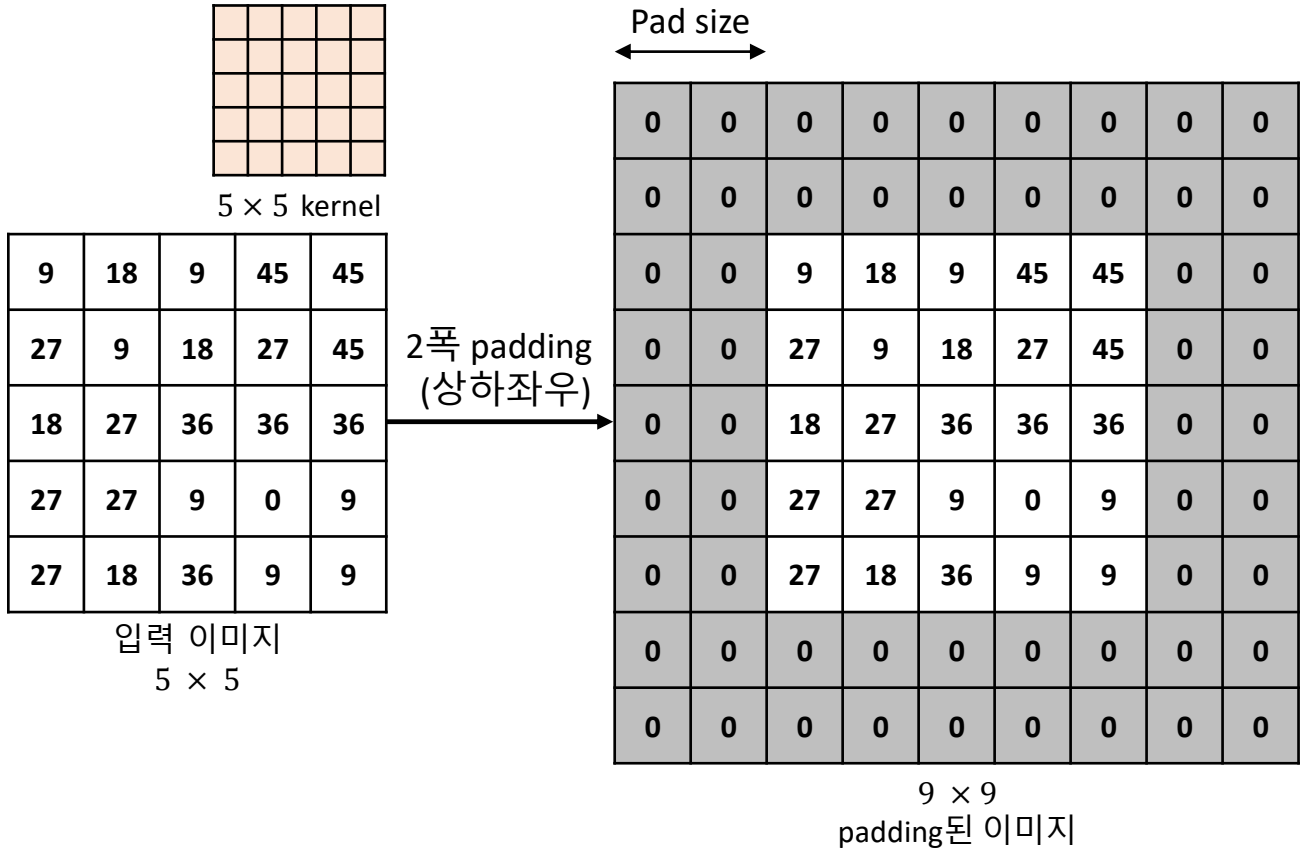
Repetition padding 이미지

Padding

- **Padding size 설정**
  - Padding은 filter의 (높이-1)/2만큼 상하에, (너비-1)/2 만큼 좌우에 진행



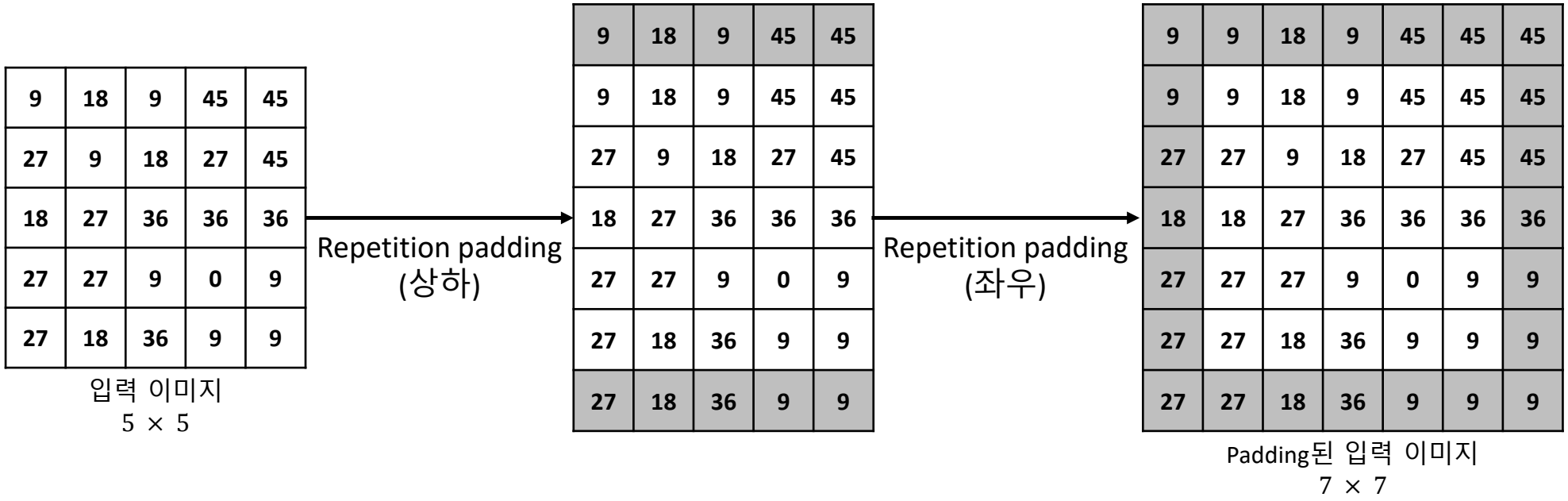
3 × 3 filter 사용할 때



5 × 5 filter 사용할 때

Padding

- **Padding 과정 (Repetition)**
  - (5, 5) 형태의 이미지에 상하좌우 1픽셀만큼 repetition padding 진행
  - 상하좌우 순서로 padding 진행



Padding

- **Padding 실습**

```python
if __name__ == '__main__':
    src = cv2.imread('Lena.png',cv2.IMREAD_GRAYSCALE)

    my_zero_pad_img = my_padding(src,(20,20))
    my_repetition_pad_img = my_padding(src, (20,20),pad_type='repetition')

    cv2.imshow('original',src)
    cv2.imshow('my zero pad img', my_zero_pad_img)
    cv2.imshow('my repetition pad img', my_repetition_pad_img)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

```python
import cv2
import numpy as np

def my_padding(src,pad_shape,pad_type = 'zero'):

    # default - zero padding으로 셋팅
    (h,w) = src.shape
    (p_h, p_w) = pad_shape
    pad_img = np.zeros((h + 2 * p_h, w + 2 * p_w), dtype=np.uint8)
    pad_img[p_h:h + p_h, p_w: w + p_w] = src


    if pad_type == 'repetition':
        print('repetition padding')
        #up
        pad_img[:p_h, p_w:p_w + w] = src[0,:]

        #down
        pad_img[p_h + h:, p_w:p_w + w] = src[h-1,:]

        #left
        pad_img[:,:p_w] = pad_img[:,p_w:p_w + 1]

        #right
        pad_img[:,p_w + w:] = pad_img[:,p_w + w -1: p_w + w]

    else:
        # else is zero padding
        print('zero padding')
    return pad_img
```

Filter 총합에 따른 밝기

- **Filter 총합에 따른 결과 이미지 분석**

Original

총합 1

$$\frac{1}{25}\cdot\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

Filtering 결과 이미지

Original

총합 1 미만

$$\frac{1}{35}\cdot\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

Filtering 결과 이미지

Original

총합 1 초과

$$\frac{1}{15}\cdot\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

Filtering 결과 이미지

Filter 종합에 따른 밝기

- 실습

```python
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena.png', cv2.IMREAD_GRAYSCALE)
    kernel1 = np.ones((5, 5))
    kernel1 = kernel1 / 25
    print(np.sum(kernel1))
    dst1 = cv2.filter2D(src, -1, kernel1,
                        borderType=cv2.BORDER_REPLICATE)

    kernel2 = np.ones((5, 5))
    kernel2 = kernel2 / 40
    print(np.sum(kernel2))
    dst2 = cv2.filter2D(src, -1, kernel2,
                        borderType=cv2.BORDER_REPLICATE)
```

```python
    kernel3 = np.ones((5, 5))
    kernel3 = kernel3 / 10
    print(np.sum(kernel3))
    dst3 = cv2.filter2D(src, -1, kernel3,
                        borderType=cv2.BORDER_REPLICATE)

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst1', dst1)
    cv2.imshow( winname: 'dst2', dst2)
    cv2.imshow( winname: 'dst3', dst3)

    cv2.waitKey()
    cv2.destroyAllWindows()
```

Sharpening filter

- **Sharpening filter**
  - 이미지를 선명하게 해주는 효과
  - Noisy한 이미지는 Noise 강화

Original

3 × 3 sharpening filter

Noisy image

3 × 3 sharpening filter

Sharpening filter

- **Sharpening filter 실습**
  - Sharpening filter = kernel 1 – kernel 2

Case 1

*

0	0	0
0	2	0
0	0	0

kernel 1

*

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

kernel 2

```python
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena.png', cv2.IMREAD_GRAYSCALE)

    ###################################
    #  sharpening filter case 1 구현  #
    ###################################

    dst3 = (np.clip(dst3 + 0.5,  a_min: 0,  a_max: 255)).astype(np.uint8)

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst3', dst3)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

Case 2

0	0	0
0	2	0
0	0	0

kernel 1

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

kernel 2

*

-1/9	-1/9	-1/9
-1/9	17/9	-1/9
-1/9	-1/9	-1/9

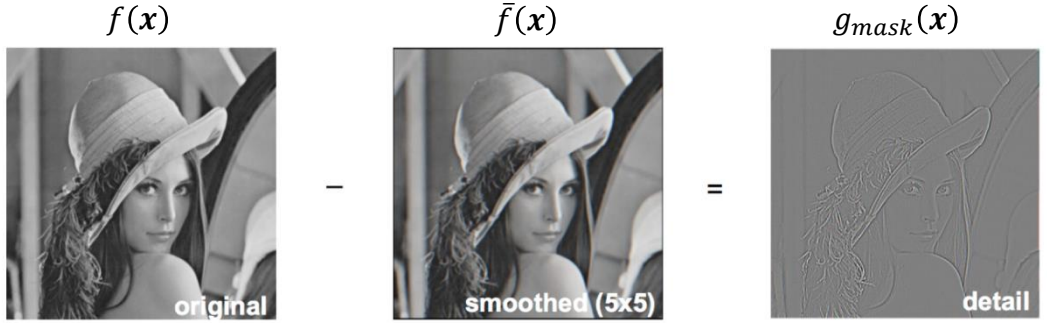
Sharpening filter

```python
if __name__ == '__main__':
    src = cv2.imread( filename: 'Lena.png', cv2.IMREAD_GRAYSCALE)

    ###################################
    #  sharpening filter case 2 구현  #
    ###################################

    print(f'sharpening_filter 총합: {np.sum(sharpening_filter)}')

    cv2.imshow( winname: 'original', src)
    cv2.imshow( winname: 'dst4', dst4)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

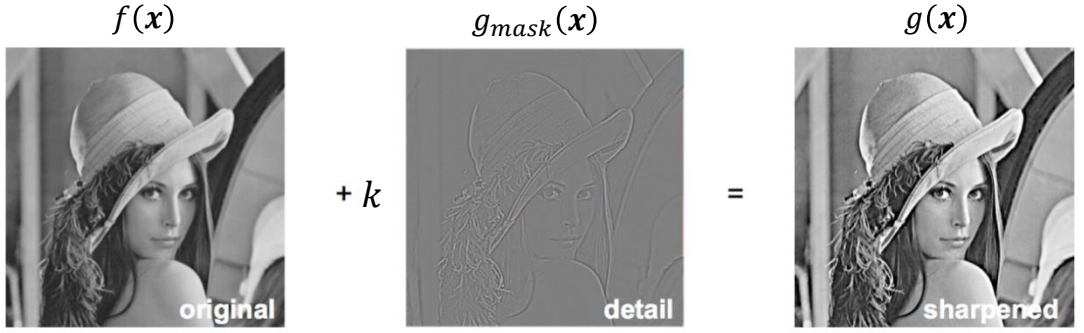
Sharpening filter

- **Sharpening filter 작동 원리**
  - Unsharp masking
    - $f(\boldsymbol{x})$: original 이미지, $\bar{f}(\boldsymbol{x})$: smoothed original 이미지, $g(\boldsymbol{x})$: sharpening 이미지
    - $g_{mask}(\boldsymbol{x}) = f(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})$
    - $g(\boldsymbol{x}) = f(\boldsymbol{x}) + k \cdot g_{mask}(\boldsymbol{x})$
      $= (k+1)f(\boldsymbol{x}) - k\bar{f}(\boldsymbol{x})$

1. Original – Smoothed = Details



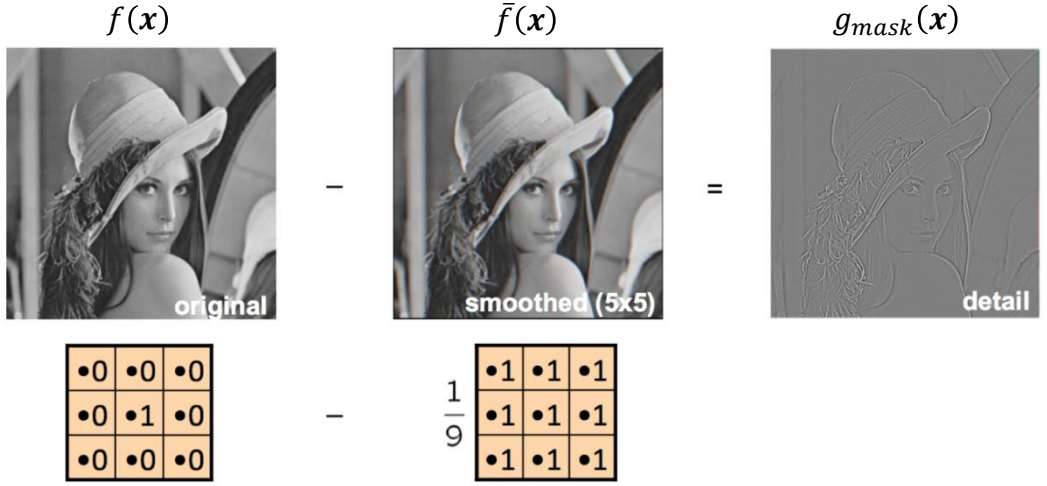
2. Original + Details = Sharpened



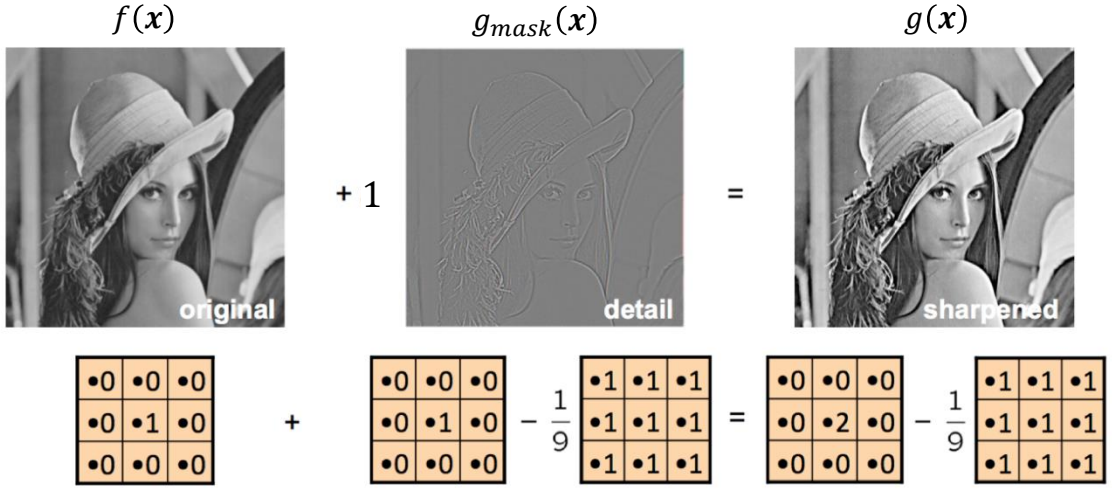
Sharpening filter

- **Sharpening filter 작동 원리**
  - Unsharp masking
    - $f(\boldsymbol{x})$: original 이미지, $\bar{f}(\boldsymbol{x})$: smoothed original 이미지, $g(\boldsymbol{x})$: sharpening 이미지
    - $g_{mask}(\boldsymbol{x}) = f(\boldsymbol{x}) - \bar{f}(\boldsymbol{x})$
    - $g(\boldsymbol{x}) = f(\boldsymbol{x}) + k \cdot g_{mask}(\boldsymbol{x})$
      $= (k+1)f(\boldsymbol{x}) - k\bar{f}(\boldsymbol{x})$

1. Original – Smoothed = Details



2. Original + Details = Sharpened



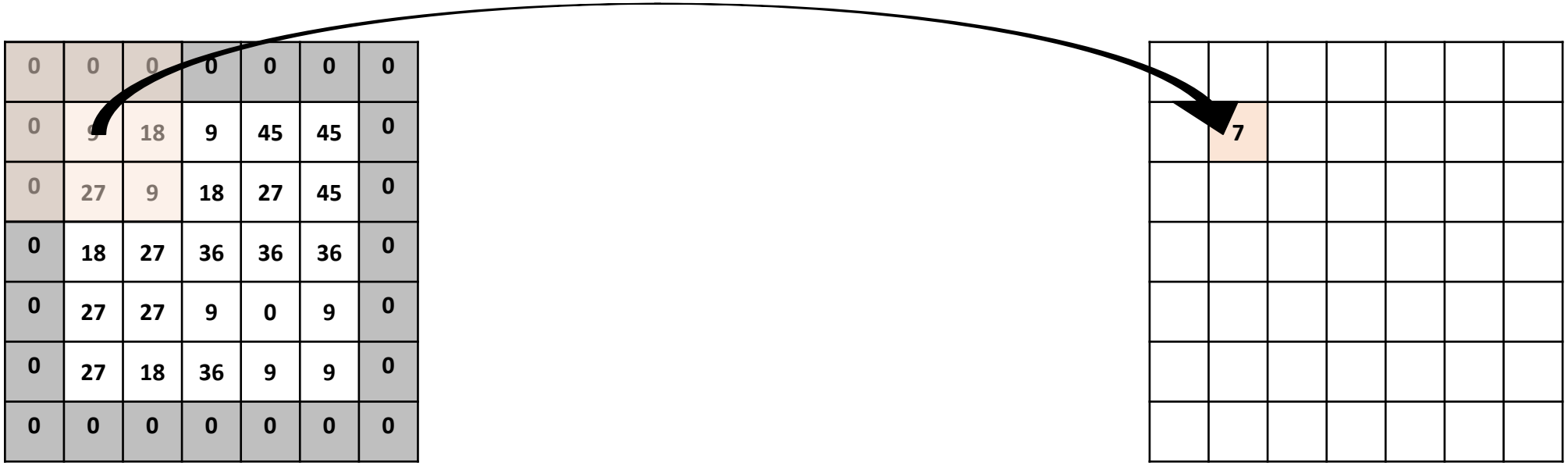
<$k = 1$ 일때 예시>

과제 1 my_filtering.py

- **입력과 출력의 크기가 동일한 my_filtering.py 완성**
  - my_filtering(src, kernel, pad_type)
    - src: 입력 이미지
    - kernel: filtering에 사용되는 kernel
      - Kernel은 (홀수, 홀수) 형태만 들어온다고 가정
    - pad_type: padding의 타입 지정
    - 출력: filtering된 결과 이미지
    - 짝수를 제외한 임의의 kernel의 size가 들어와도 filtering 되어야 함
  - 보고서에 포함될 내용
    - Average filtering된 결과 이미지
  - 사용 금지 함수(아래 함수 제외하고 모두 사용 가능)
    - cv2.filte2D()
  - $g(x,\ y) = \sum_{s=-k}^{k}\sum_{t=-k}^{k} \underbrace{w(s,t)}_{\text{kernel}} \underbrace{f(x+s, y+t)}_{\text{input}}$

과제 1 my_filtering.py

- **Filtering 과정**

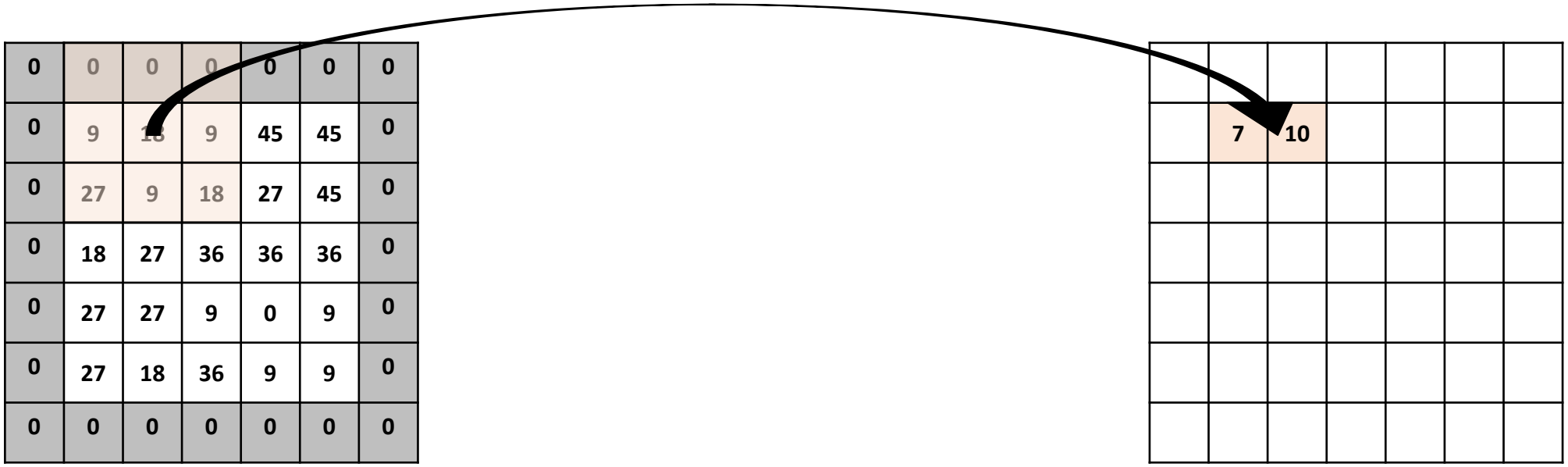


Padding 입력 이미지
7 × 7

결과 이미지

과제 1 my_filtering.py

- **Filtering 과정**

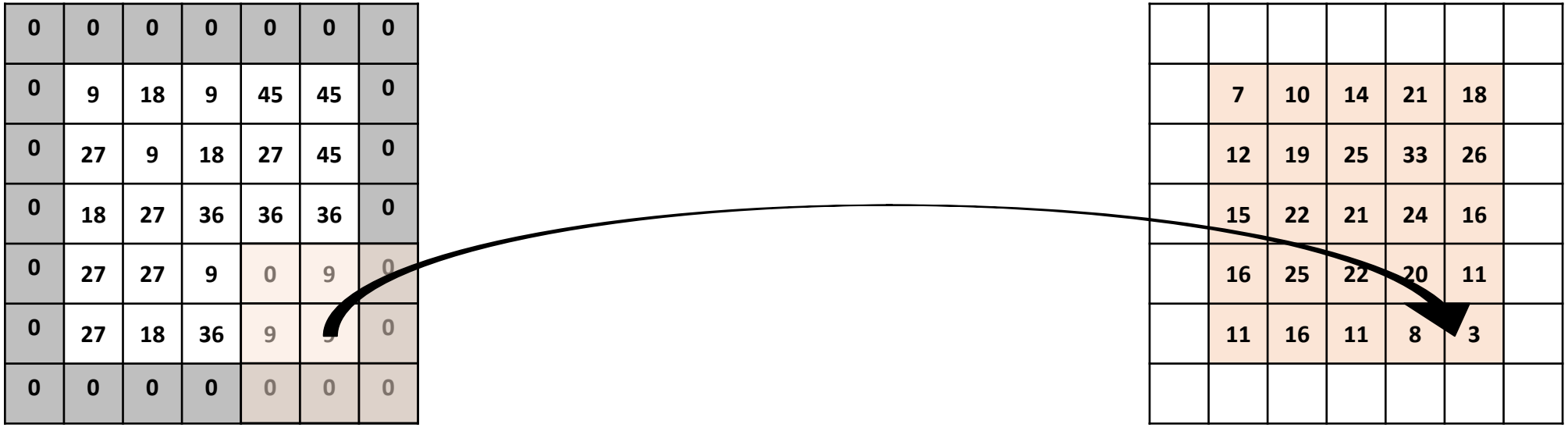


Padding 입력 이미지
7 × 7

결과 이미지

과제 1 my_filtering.py

- **Filtering 과정**

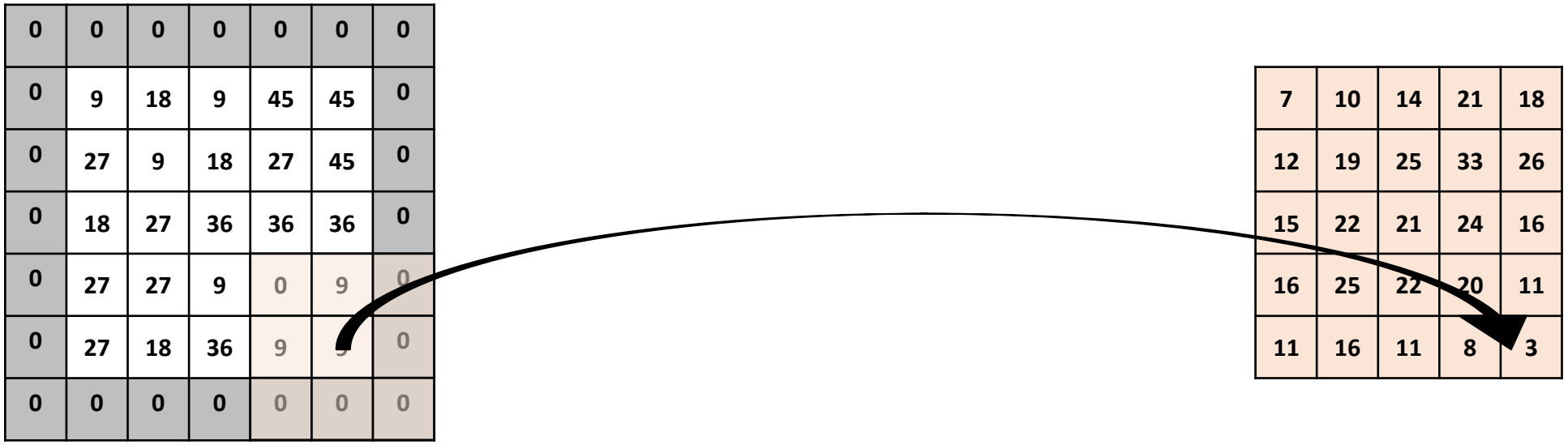


Padding 입력 이미지
7 × 7

결과 이미지

과제 1 my_filtering.py

- **Filtering 과정**

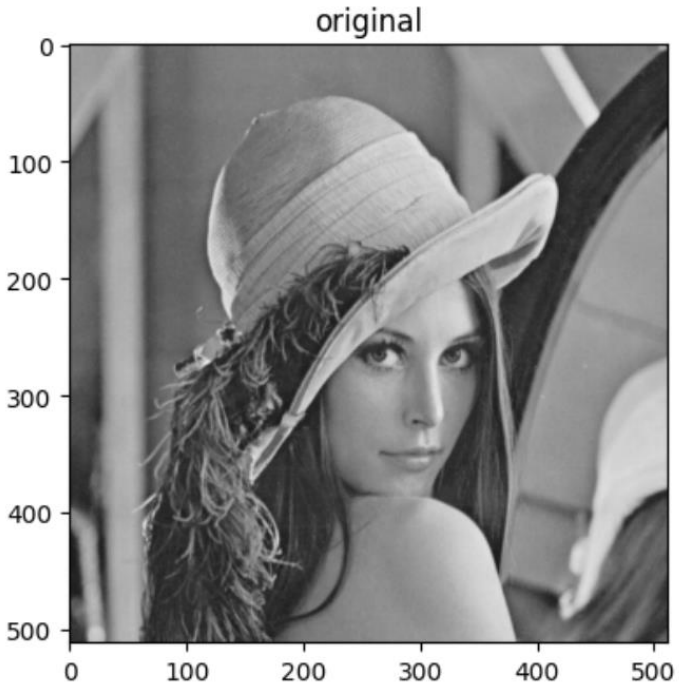


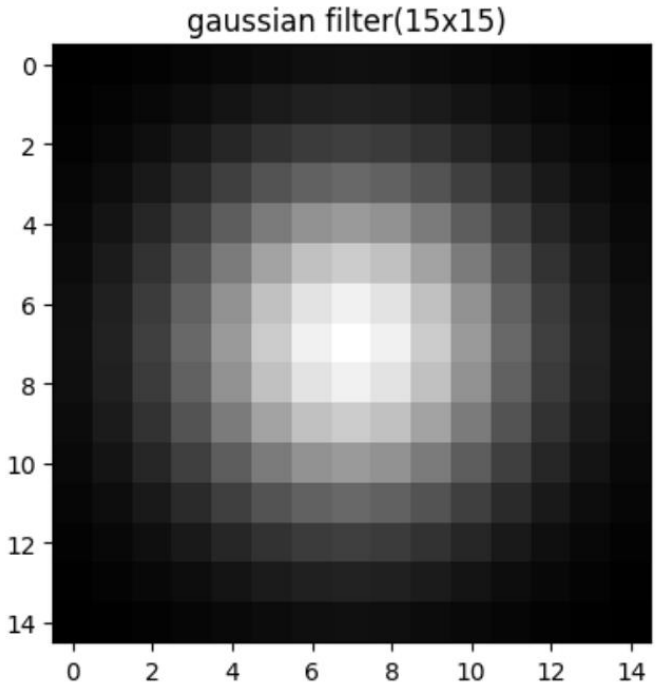
Padding 입력 이미지
7 × 7

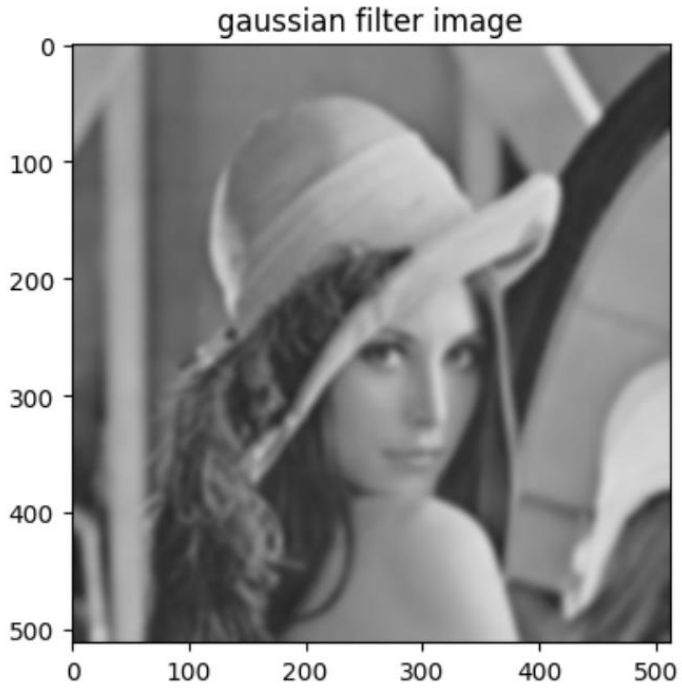
결과 이미지

Gaussian filter

- **2D Gaussian filter**
  - 잡음을 줄이고, 이미지를 부드럽게 만드는데 사용
  - 중심에서 멀어질수록 값이 감소하는 Gaussian 함수를 통해 생성







Gaussian filter

- **2D Gaussian filter**

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

- 임의의 $m \times n$ 가우시안 필터 생성시 $k = (n-1)/2$
- x : $-k \sim k$ 범위의 kernel에서 x좌표(열)
- y : $-k \sim k$ 범위의 kernel에서 y좌표(행)
- σ (sigma) : Gaussian 분포의 표준편차

(-2, -2)	(-1, -2)	(0, -2)	(1, -2)	(2, -2)
(-2, -1)	(-1, -1)	(0, -1)	(1, -1)	(2, -1)
(-2, 0)	(-1, 0)	(0, 0)	(1, 0)	(2, 0)
(-2, 1)	(-1, 1)	(0, 1)	(1, 1)	(2, 1)
(-2, 2)	(-1, 2)	(0, 2)	(1, 2)	(2, 2)

5×5 Gaussian filter의 $(x,\ y)$

$$\frac{1}{sum} \cdot$$

$\frac{1}{2\pi}e^{-\frac{4+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{0+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{4+4}{2}}$
$\frac{1}{2\pi}e^{-\frac{4+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{0+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{4+1}{2}}$
$\frac{1}{2\pi}e^{-\frac{4+0}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+0}{2}}$	$\frac{1}{2\pi}e^{0}$	$\frac{1}{2\pi}e^{-\frac{1+0}{2}}$	$\frac{1}{2\pi}e^{-\frac{4+0}{2}}$
$\frac{1}{2\pi}e^{-\frac{4+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{0+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+1}{2}}$	$\frac{1}{2\pi}e^{-\frac{4+1}{2}}$
$\frac{1}{2\pi}e^{-\frac{4+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{0+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{1+4}{2}}$	$\frac{1}{2\pi}e^{-\frac{4+4}{2}}$

5×5 Gaussian filter, $\sigma = 1$

과제 2 my_gaussian.py

- **my_gaussian 2D 구현**
  - my_get_Gaussian2D_kernel(ksize, sigma)
    - ksize: Gaussian filter의 크기
    - sigma: Gaussian filter 의 표준편차
    - 출력: (ksize, ksize)인 Gaussian filter
  - 보고서에 포함될 내용
    - 구현한 my_get_Gaussian2D_kernel() 함수 설명
  - 사용 금지 함수 (아래 함수 제외하고 모두 사용 가능)
    - cv2.getGaussianKernel()
    - cv2.filter2D()

과제 2 my_gaussian.py

- σ에 따른 최적의 kernel 크기 선정
  - print_kernel(kernel)
    - kernel: 출력하고 싶은 kernel
  - save_kernel_img(kernel, ksize, sigma) – 참고용
    - kernel: 시각화 하고 싶은 kernel
    - ksize: kernel의 크기
    - sigma: gaussian filter에 사용된 sigma
    - kernel_scaled는 0 ~ 1로 정규화(min-max scaling)된 kernel을 의미
    - 함수의 결과로 kernel의 시각화 된 이미지 저장
  - 보고서에 포함될 내용
    - $\sigma = 0.5,\ \ 2,\ \ 3$
    - 위 σ별 최적의 kernel 크기를 선정하고, 그에 대한 근거 작성
      - Gaussian filter 영상이 아닌 값을 보고 근거 작성

과제 2 보조 자료

- $\sigma = 1$일때, 최적의 filter size 5인 경우 값 출력
  - 가장자리의 값이 0이 아닌, 0에 가장 근접할 때가 최적의 filter size를 가짐

```
kernel_size = 5
sigma = 1
gaus2D = my_get_Gaussian2D_kernel(kernel_size, sigma)
```

```
def print_kernel(kernel):
    k_h, k_w = kernel.shape

    for row in range(k_h):
        for col in range(k_w):
            print(round(kernel[row, col], 4), end="\t")
        print()
```

kernel을 순회하는 반복문

row를 소수점 넷째 자리까지 출력

```
0.003   0.0133  0.0219  0.0133  0.003
0.0133  0.0596  0.0983  0.0596  0.0133
0.0219  0.0983  0.1621  0.0983  0.0219
0.0133  0.0596  0.0983  0.0596  0.0133
0.003   0.0133  0.0219  0.0133  0.003
```

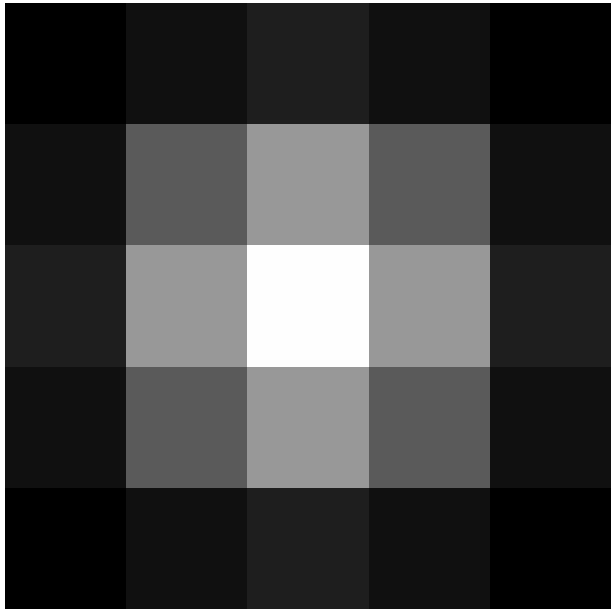
과제 2 보조 자료

- $\sigma = 1$일때, 최적의 filter size 5인 경우 영상 출력
  - 영상은 값을 보는 것 보다 부정확하므로 참고로 사용
  - 저장된 영상은 매우 작으므로 확대해서 사용

```python
def save_kernel_img(kernel, ksize, sigma):
    kernel_scaled = (kernel - np.min(kernel)) / ((np.max(kernel) - np.min(kernel)) + 1e-10)
    kernel_scaled = (kernel_scaled * 255).astype(np.uint8) # float -> uint8
    cv2.imwrite( filename: f'gaussian_img_{ksize}_{sigma}.png', kernel_scaled)
```

0 ~ 1 정규화

kernel 크기, sigma 명시해서 Gaussian filter 저장



과제

- **보고서**
  - 내용
    - 학과, 학번, 이름
    - 구현 코드: 구현한 코드에 대한 간단한 설명
    - 이미지: 언급한 이미지 모두 첨부
    - 느낀 점: 구현 결과를 보고 느낀 점, 혹은 어려운 점 등
    - 과제 난이도: 개인적으로 느낀 난이도 및 이유(과제가 쉽다, 어렵다 등)
  - .pdf 파일로 제출(이외의 파일 형식일 경우 감점)
  - 보고서 명
    - [IP]20xxxxxxx_이름_x주차_과제.pdf

과제

- **과제 안내**
  - 채점 기준
    - 구현을 못하거나 잘못 구현한 경우
    - 보고서 내용이 빠진 경우
    - 다른 사람의 코드 copy 적발시 보여준 사람, copy한 사람 둘 다 0점
    - 내장 함수 사용시 감점(내장 함수를 사용해도 된다고 한 것 제외)
  - 제출 파일
    - 아래의 파일을 압축해서 [IP]20XXXXXXX_이름_X주차_과제.zip 으로 제출
      - .py 파일 (과제에 해당하는 모든 .py파일)
      - .pdf 보고서 파일
  - 제출 기한
    - 2024년 4월 11일 23시 59분까지

Q & A