초급 영상처리

(나만의 Opencv 구현하기)

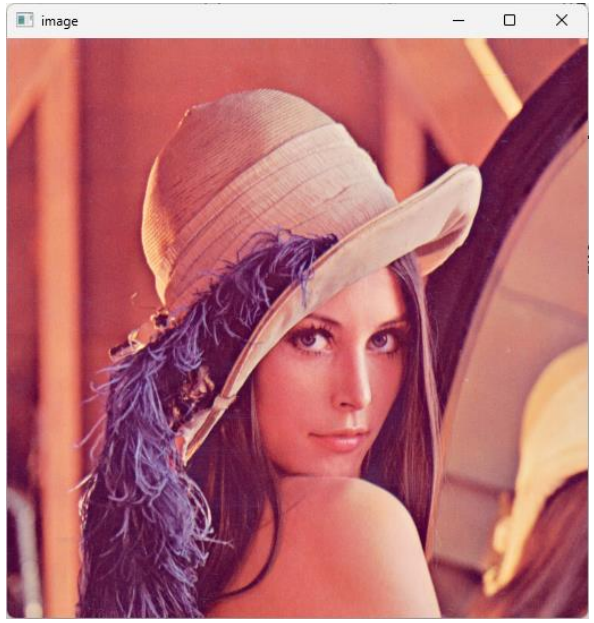
박화종

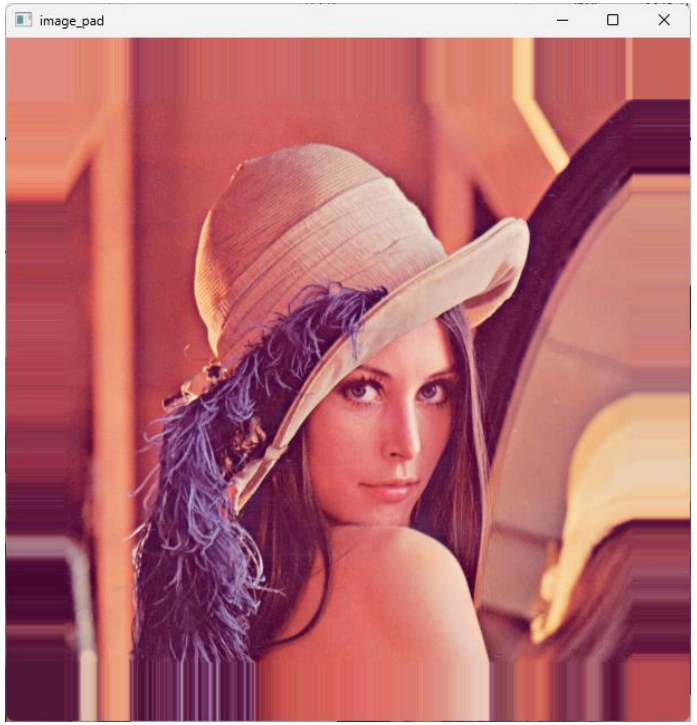
INDEX

- 저번 주 과제 정답
- Filtering
- Filter
- 실습
- 과제

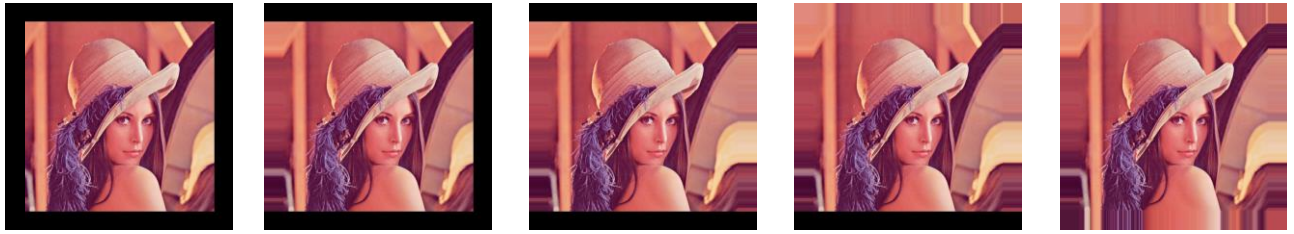
저번 주 과제(IP6_test1)

- Replicate padding 구현하기



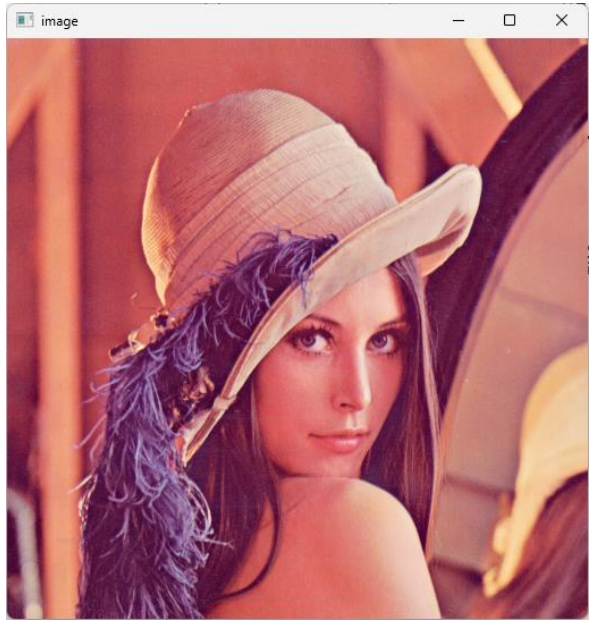


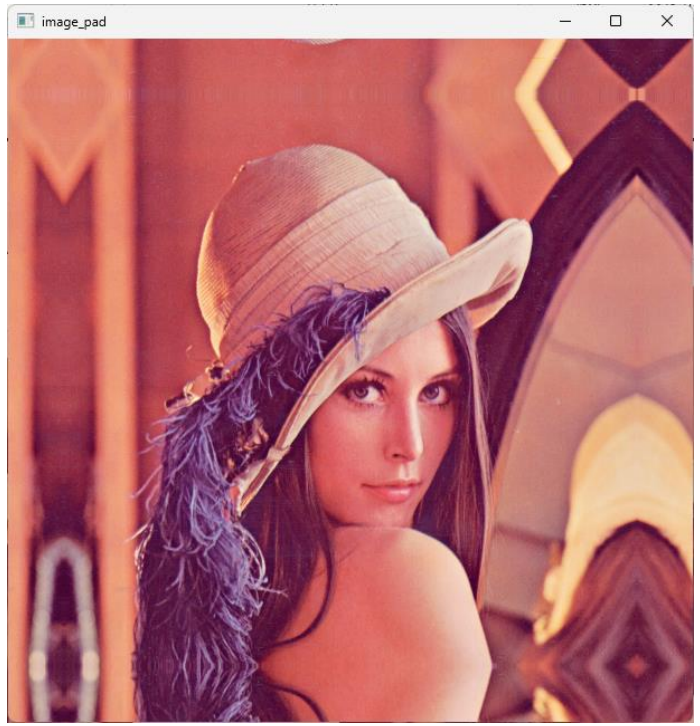

```python
def replicate_padding(img, pad):
    h, w, c = img.shape
    img_pad = np.zeros((h+pad*2, w+pad*2, c), dtype=img.dtype)

    img_pad[pad:pad+h, pad:pad+w] = img

    # left
    img_pad[:, :pad] = np.tile(img_pad[:, pad:pad+1], (1, pad, 1))
    # img_pad[:, :pad] = img_pad[:, pad:pad+1]

    # right
    img_pad[:, pad+w:] = img_pad[:, pad+w-1:pad+w]

    # top
    img_pad[:pad] = img_pad[pad:pad+1]

    # bottom
    img_pad[pad+h:] = img_pad[pad+h-1:pad+h]

    return img_pad
```



저번 주 과제(IP6_test2)

- Mirror padding 구현하기

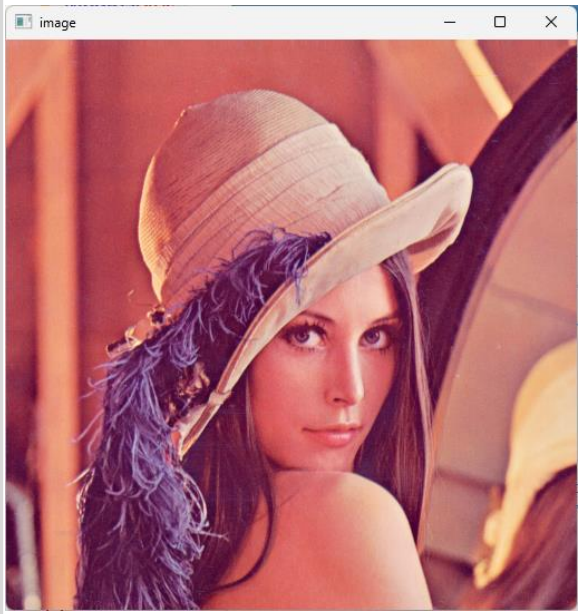




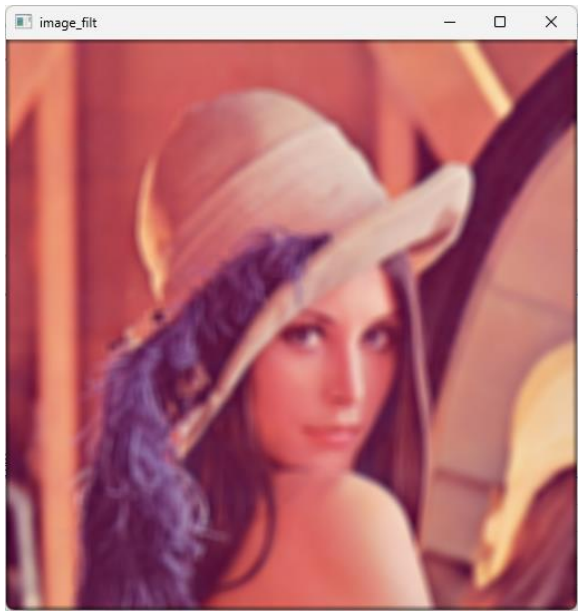

```python
def mirror_padding(img, pad):
    h, w, c = img.shape
    img_pad = np.zeros((h+pad*2, w+pad*2, c), dtype=img.dtype)

    img_pad[pad:pad+h, pad:pad+w] = img

    # left
    img_pad[:, :pad] = img_pad[:, pad:pad+pad][:, ::-1]

    # right
    img_pad[:, pad+w:] = img_pad[:, w:pad+w][:, ::-1]

    # top
    img_pad[:pad] = img_pad[pad:pad+pad][::-1]

    # bottom
    img_pad[pad+h:] = img_pad[h:pad+h][::-1]

    return img_pad
```

Filtering

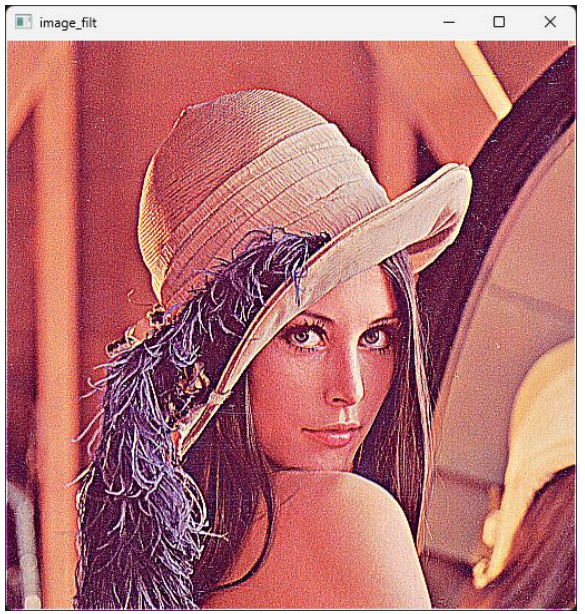
- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정



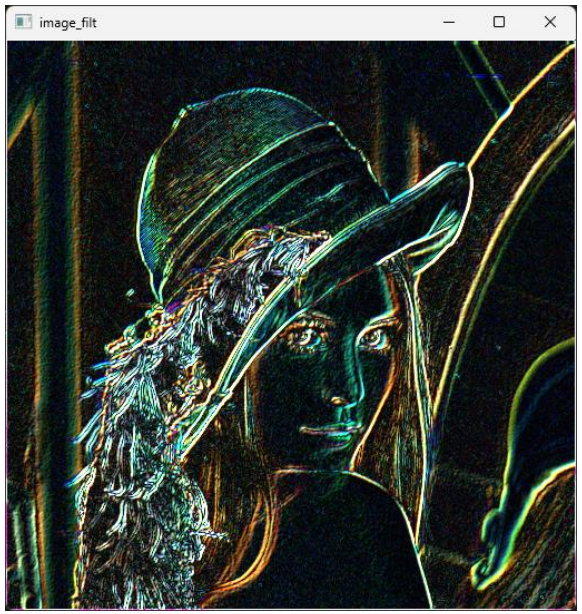
Original



Blur



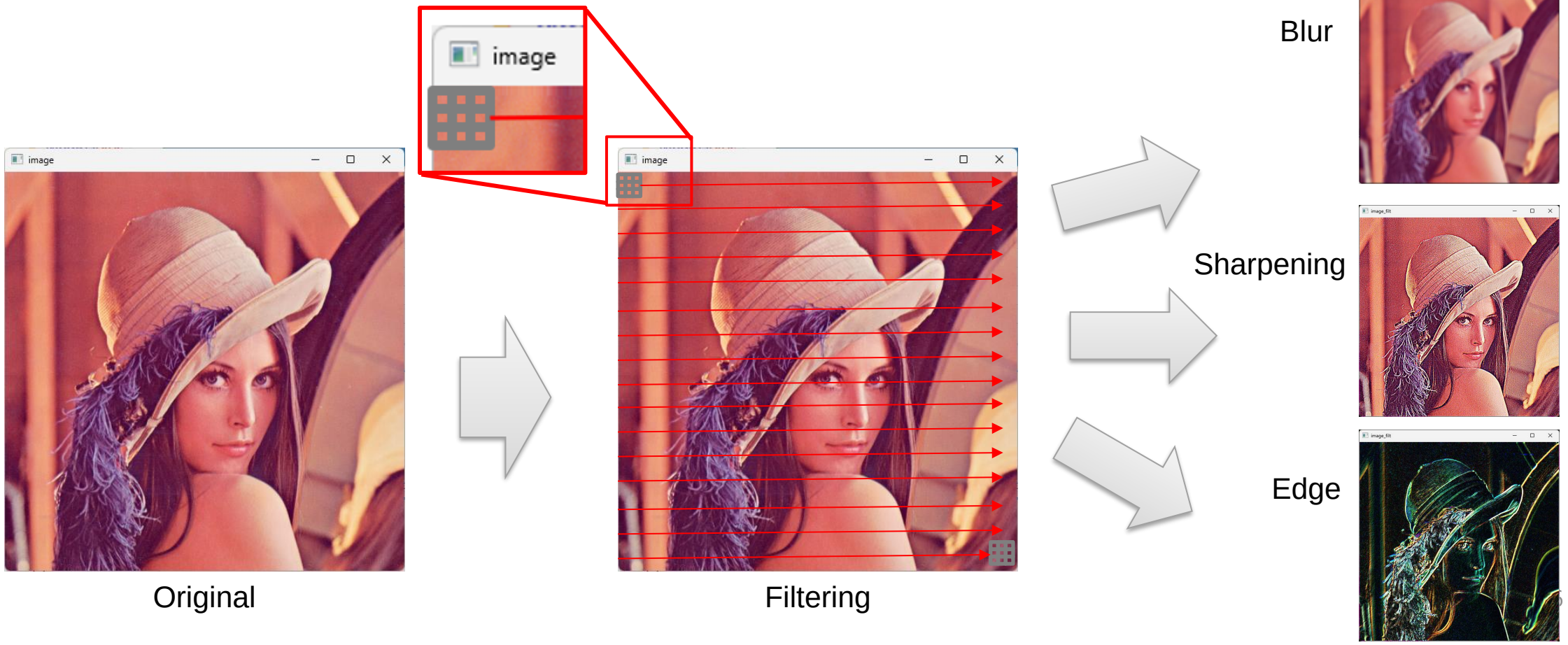
Sharpening



Edge

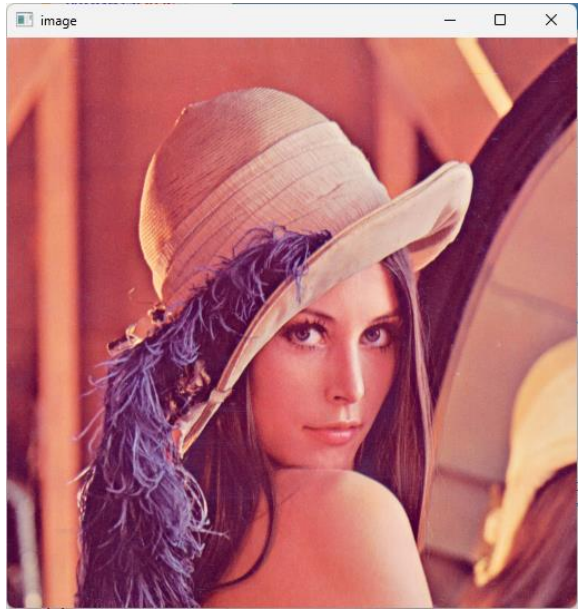
Filtering

- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - **어떤 Kernel(filter)을 사용할 건지?**

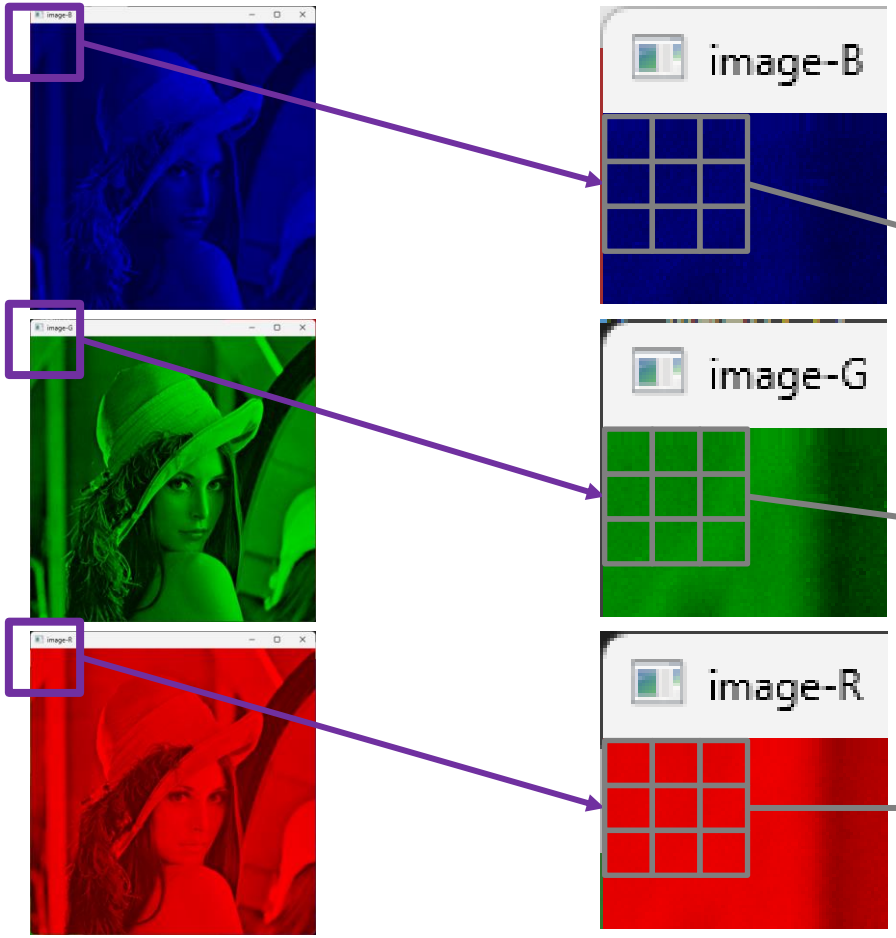


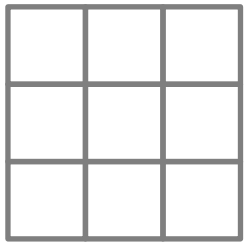
Filtering

- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - 어떤 Kernel(filter)을 사용할 건지?



(H, W, 3)

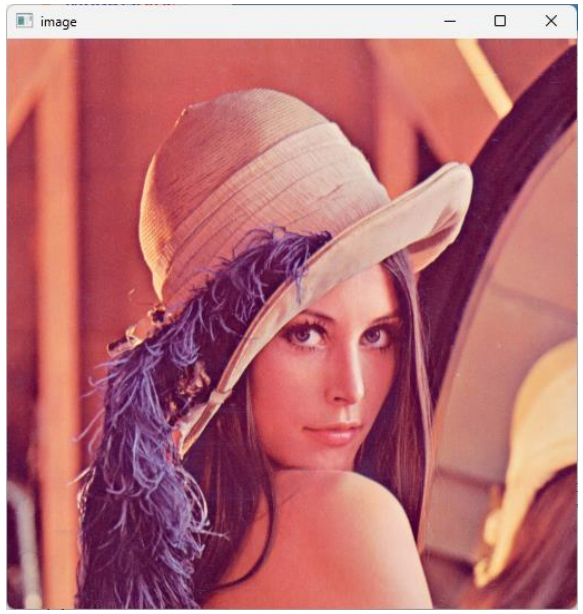




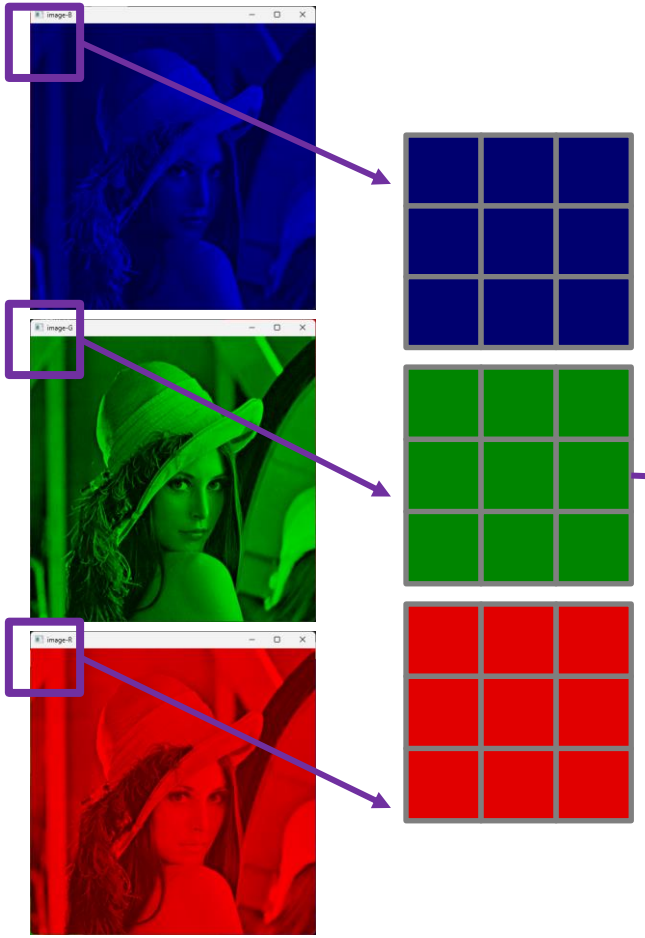
(kH, kW)

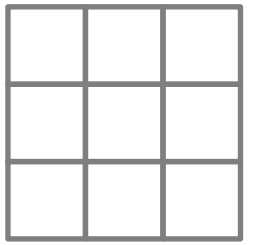
Filtering

- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - 어떤 Kernel(filter)을 사용할 건지?

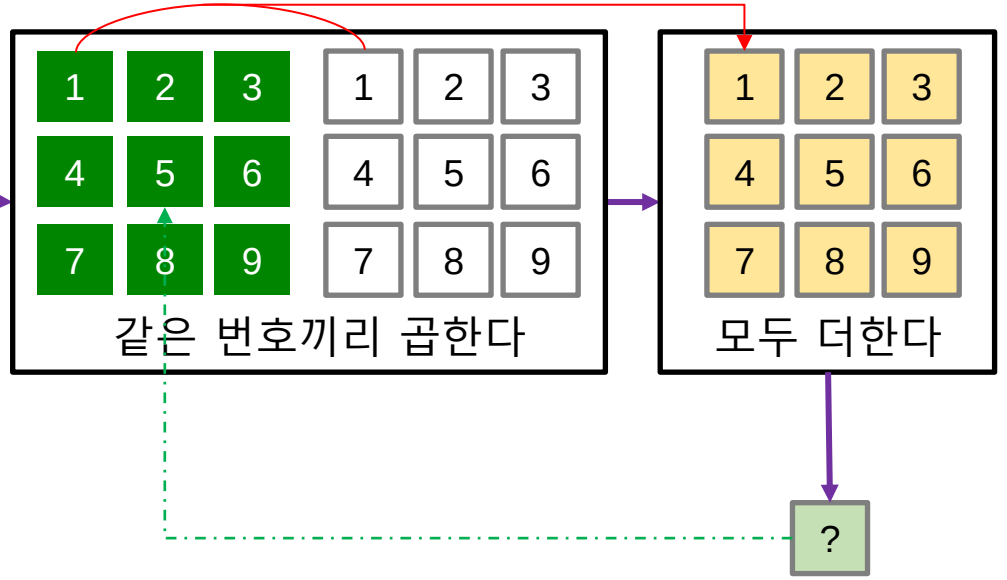


(H, W, 3)





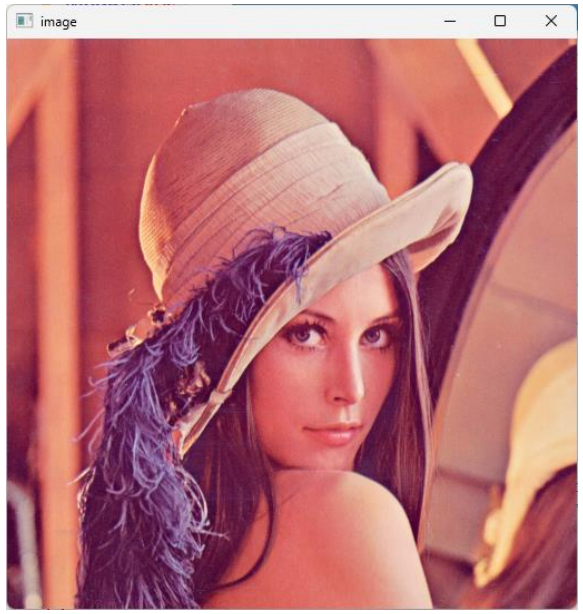
(kH, kW)



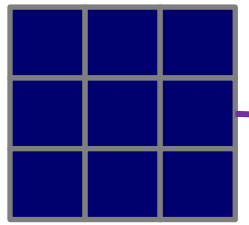
5번 위치에 사용될 값

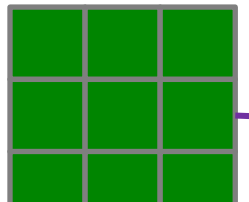
Filtering

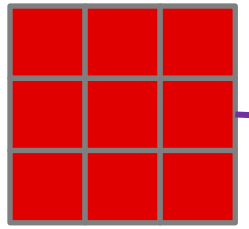
- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - 어떤 Kernel(filter)을 사용할 건지?

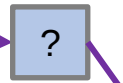


(H, W, 3)

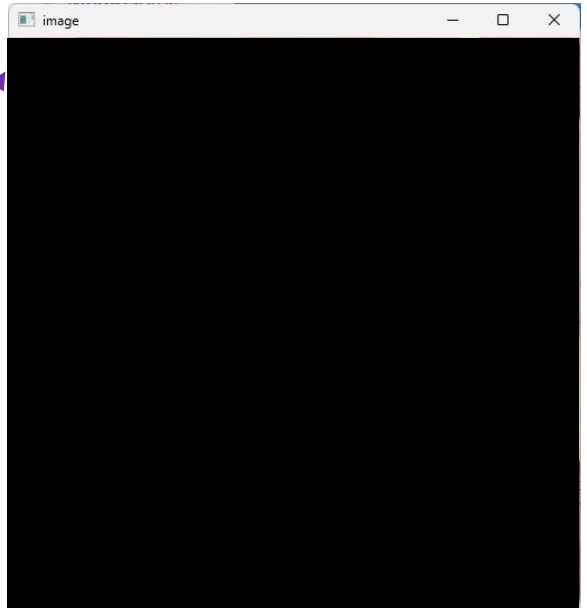




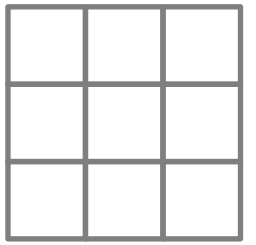




(1, 1, 3)



(H, W, 3)

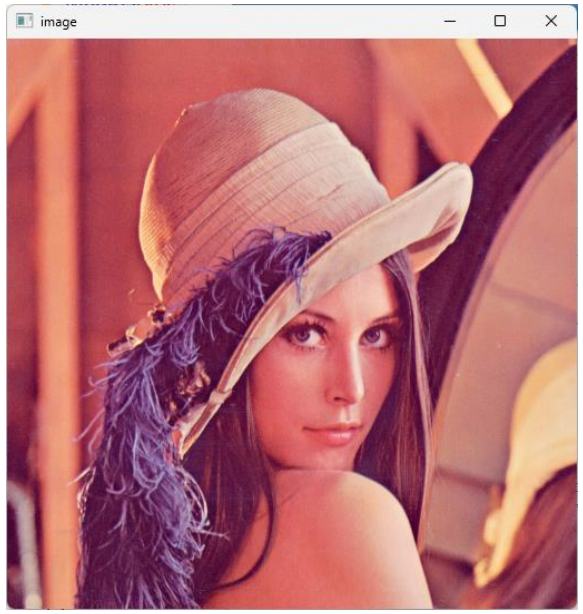


(kH, kW)

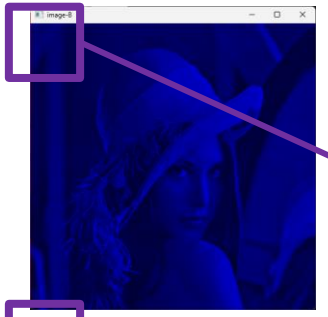
이 과정을 H*W만큼 반복하면 끝

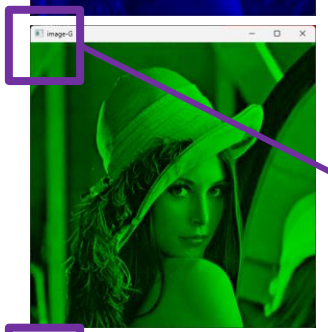
Filtering

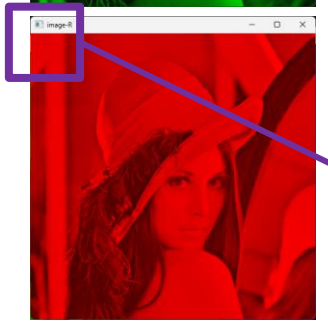
- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - 어떤 Kernel(filter)을 사용할 건지?

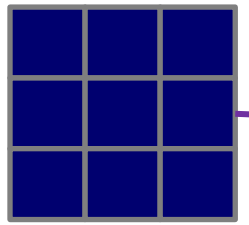


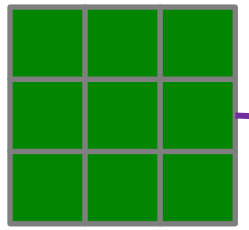
(H, W, 3)

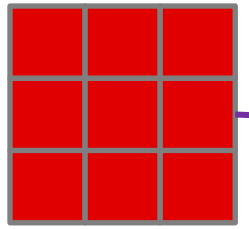




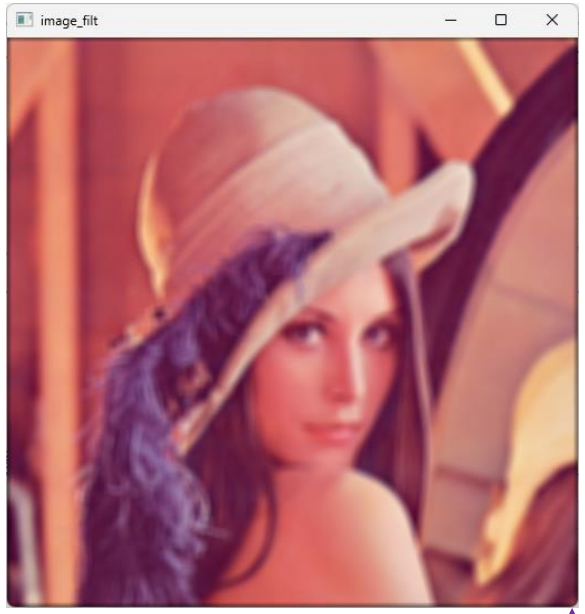






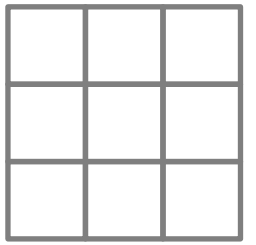


(1, 1, 3)



(H, W, 3)

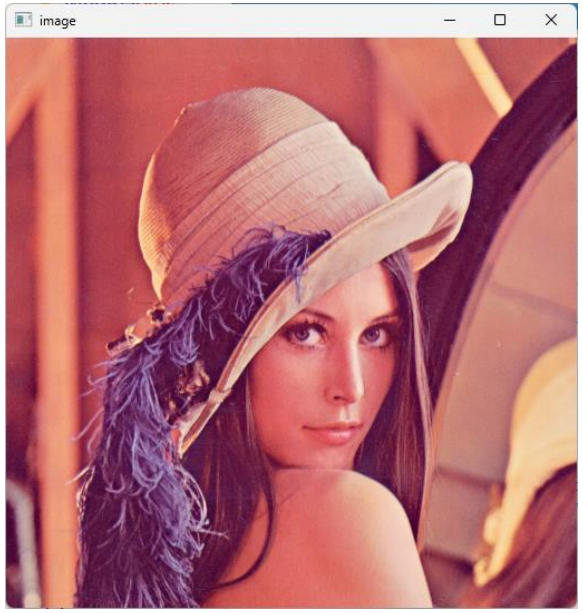
H*W만큼 반복 후



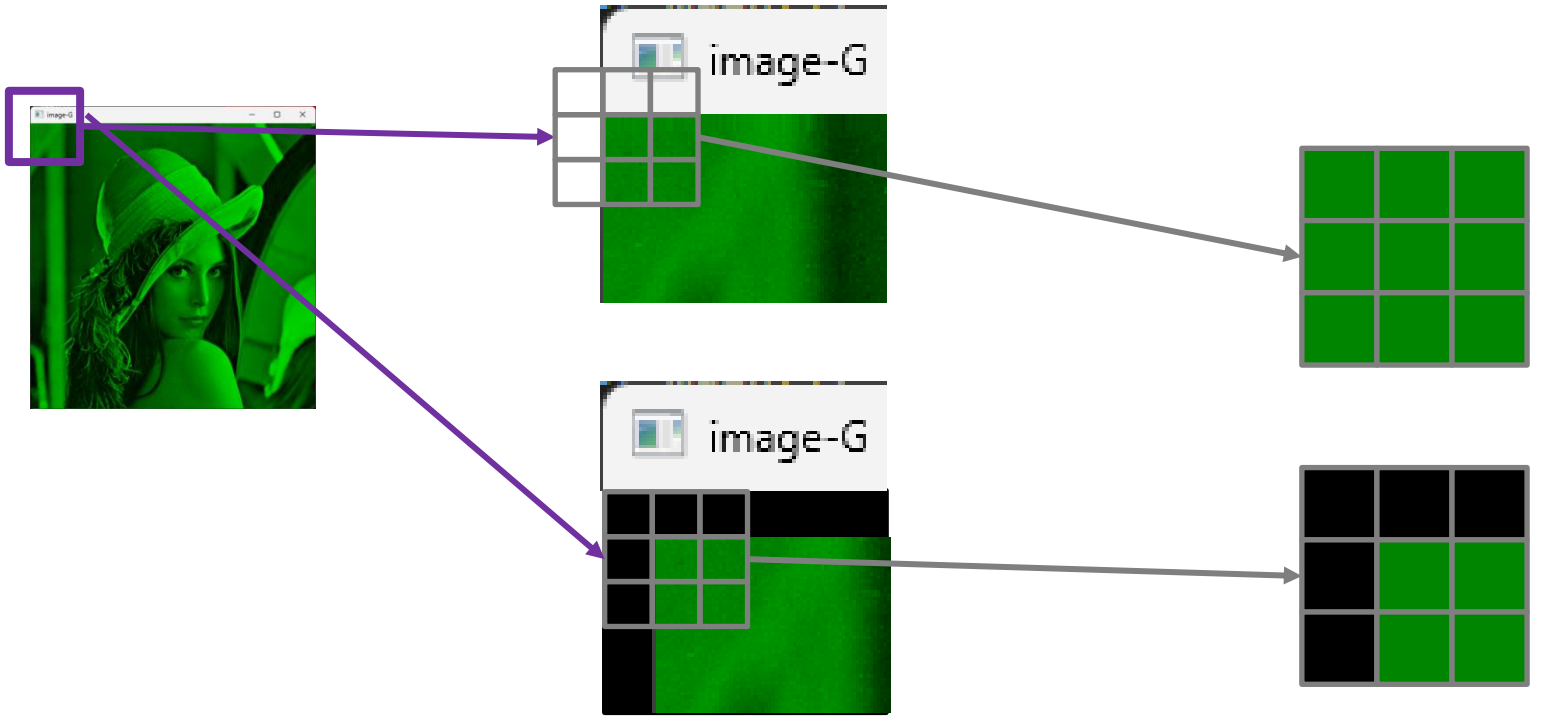
(kH, kW)

Filtering

- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - 어떤 Kernel(filter)을 사용할 건지?
  - **가장자리는 어떻게 하는 건가요? H, W가 변하는 거 아닌가요?**
    - 저번 주에 배운 padding을 사용한다.

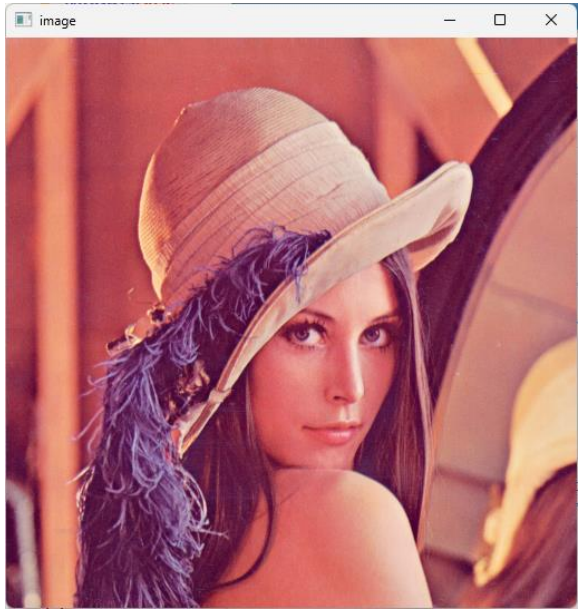


(H, W, 3)

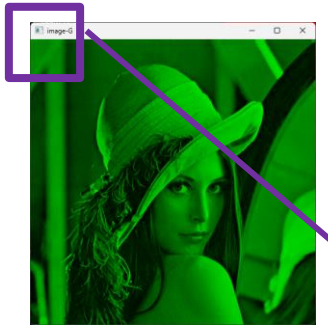


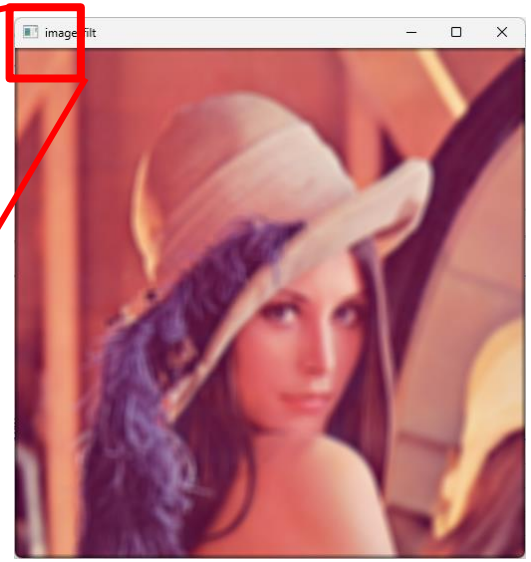
Filtering

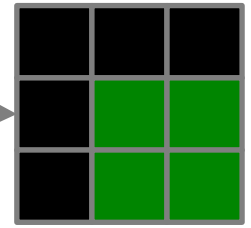
- Filtering
  - 영상의 특정 특징이나 성질을 강조하거나 제거하는 과정
  - 어떤 Kernel(filter)을 사용할 건지?
  - **가장자리는 어떻게 하는 건가요? H, W가 변하는 거 아닌가요?**
    - 저번 주에 배운 padding을 사용한다.



(H, W, 3)

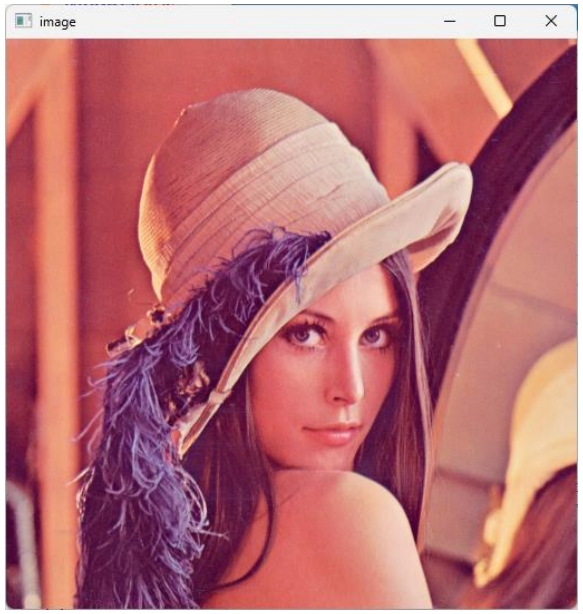




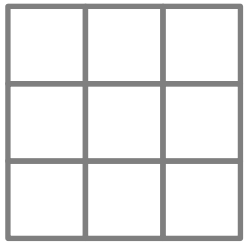


Filter

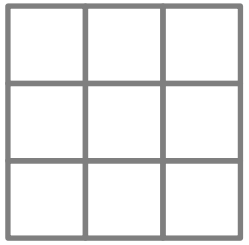
- Filter
  - 영상의 특징을 바꾸거나 강조하는 역할을 하는 **일종의 연산 도구**



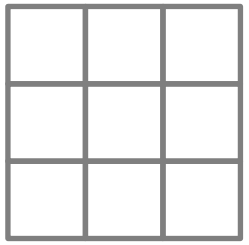
Original



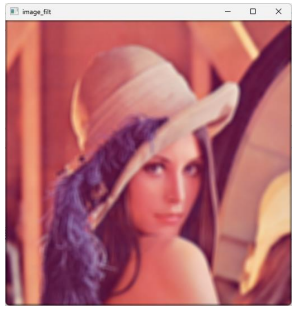
Blur filter

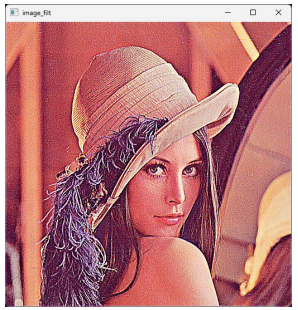


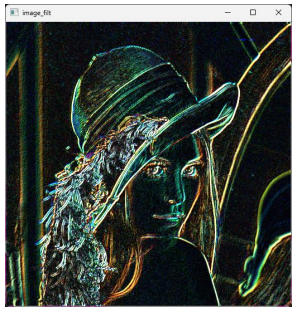
Sharpening filter



Edge filter

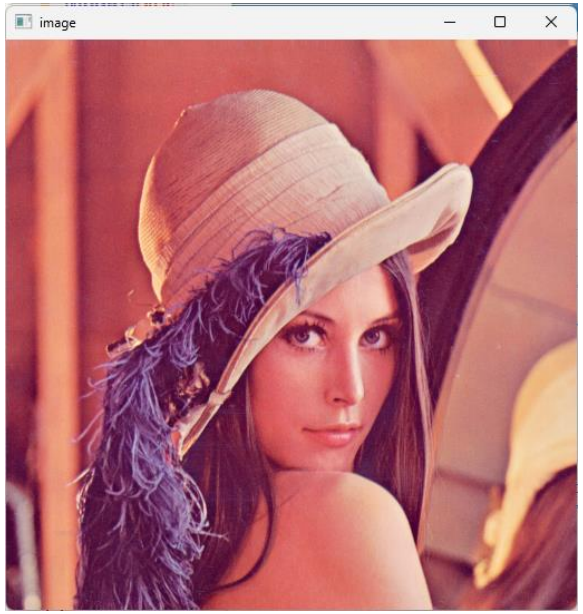




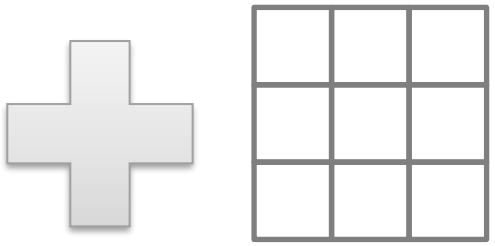


Filter

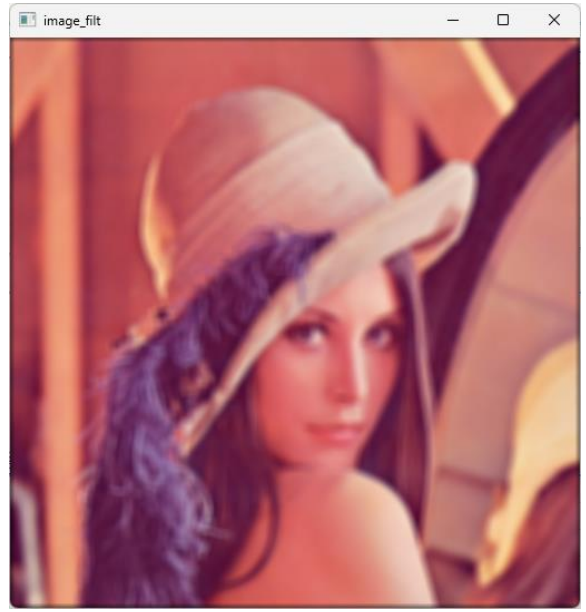
- Blur Filter
  - 세부 정보를 흐릿하게 만들어 부드럽고 매끄러운 이미지를 생성하는 필터
    - Mean filter
    - Gaussian filter



Original



Blur filter



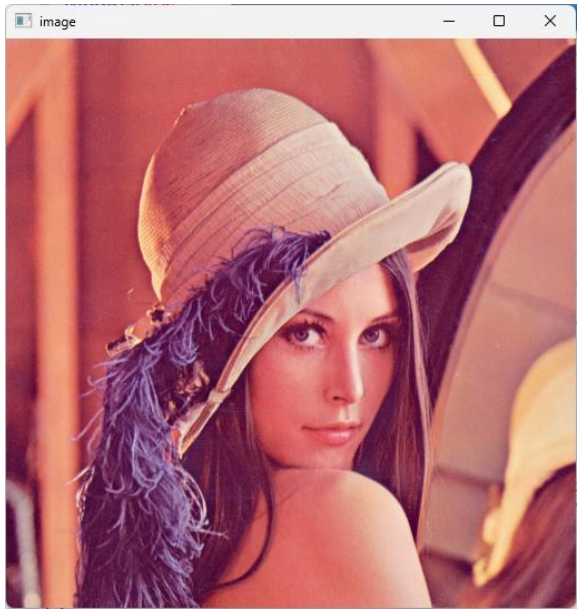
Blur

Filter

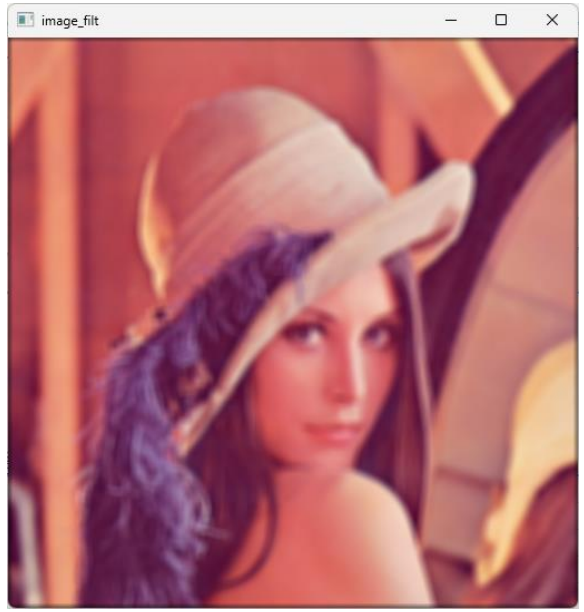
- Blur Filter – Mean filter
  - 필터의 모든 값은 동일한 값
  - 필터의 총 합은 1

$$1/9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$1/16 * \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$



Original



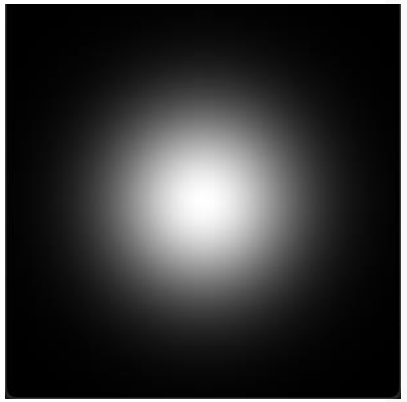
Blur

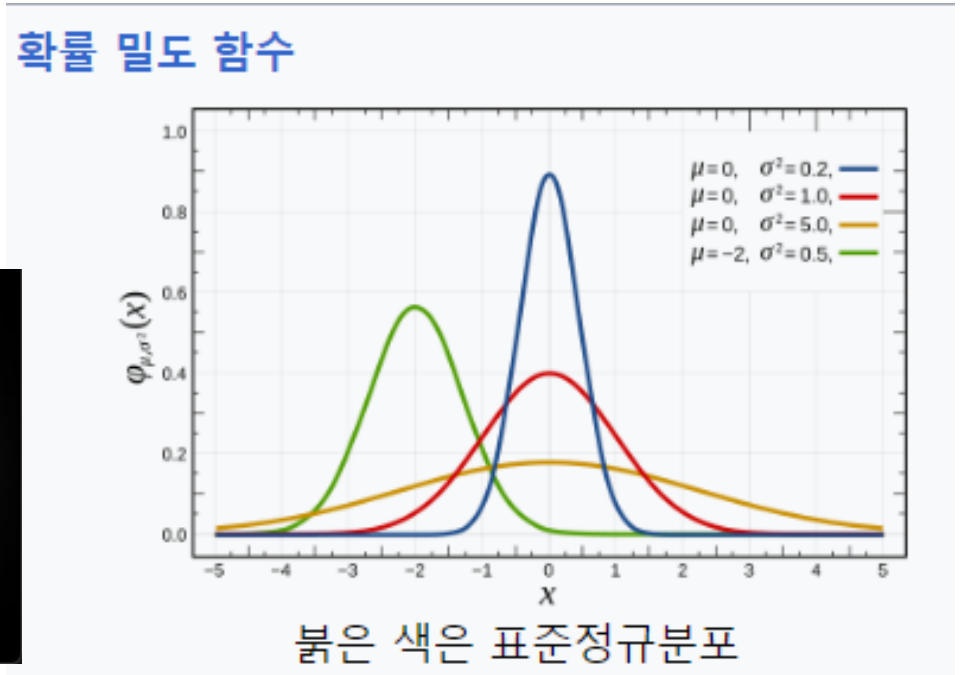
Filter

- Blur Filter – Gaussian filter
  - 가우시안 분포 함수를 근사하여 생성한 필터
  - 필터의 총 합은 1

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

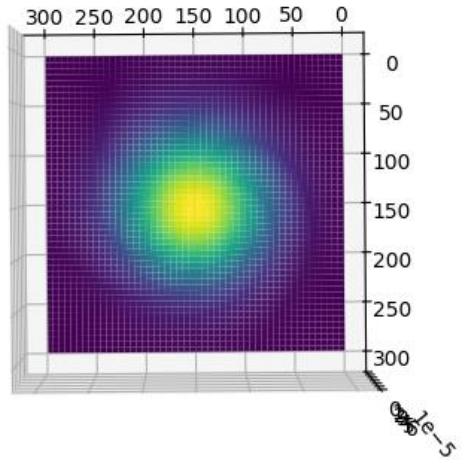
$$f(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

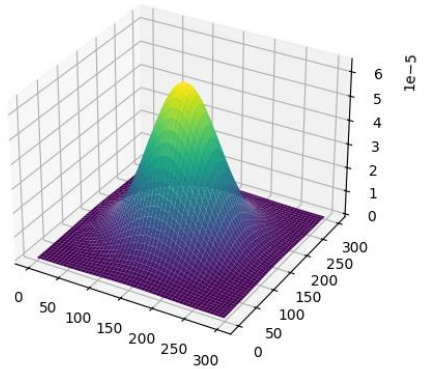





```
[0.07511361, 0.1238414 , 0.07511361]
[0.1238414 , 0.20417996, 0.1238414 ]
[0.07511361, 0.1238414 , 0.07511361]
```

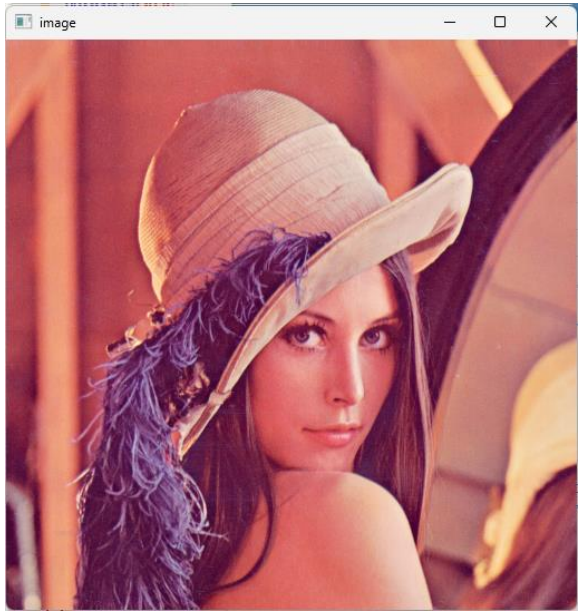
3x3 Gaussian filter



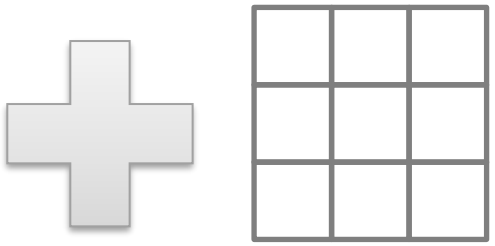


Filter

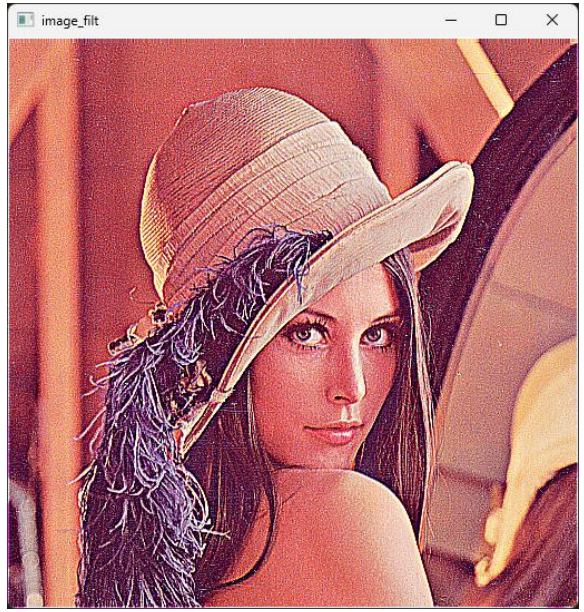
- Sharpening Filter
  - 세부 정보를 강조하여 이미지의 경계선을 뚜렷하고 선명하게 만드는 필터



Original

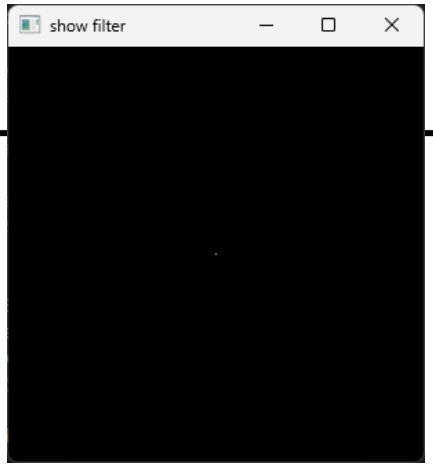


Sharpening filter



Blur

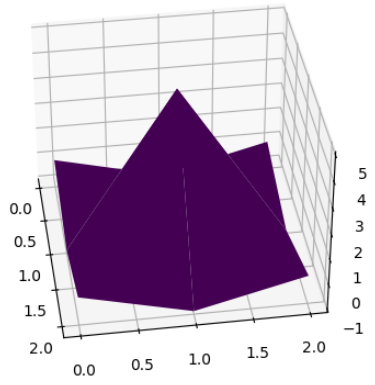
Filter

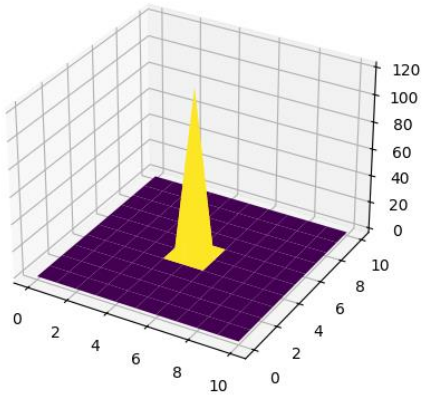


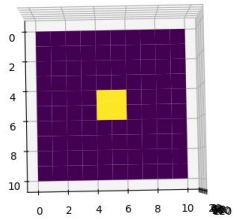
- Sharpening Filter
  - 필터의 중앙은 양수, 주변은 음수의 값을 가지는 필터
  - 필터의 총 합은 1

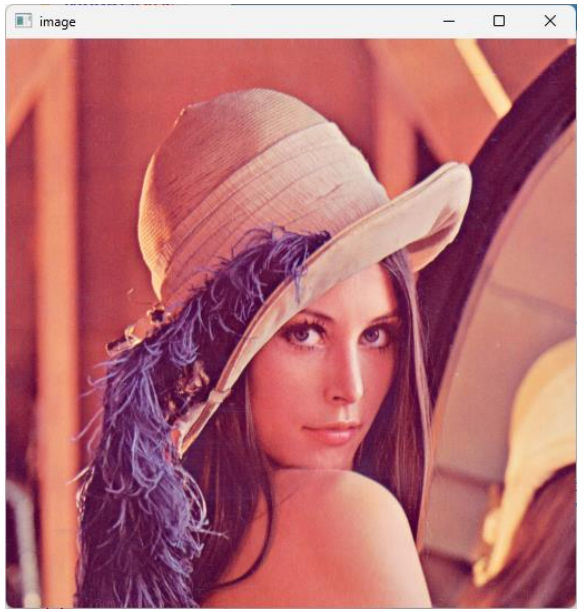
0	-1	0
-1	5	-1
0	-1	0

-1	-1	-1
-1	9	-1
-1	-1	-1

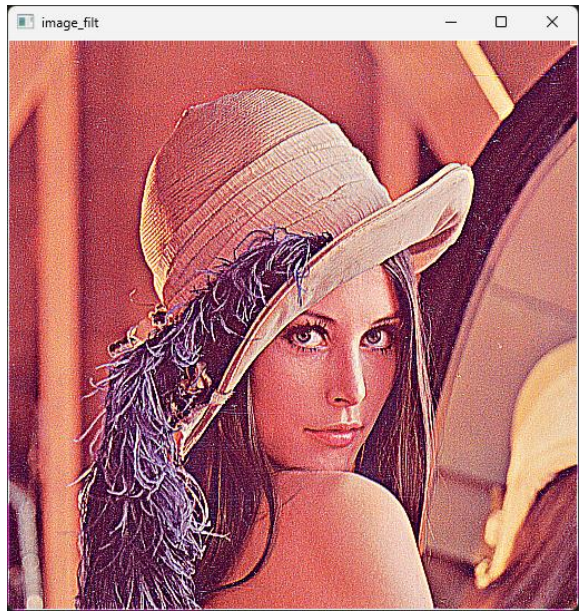








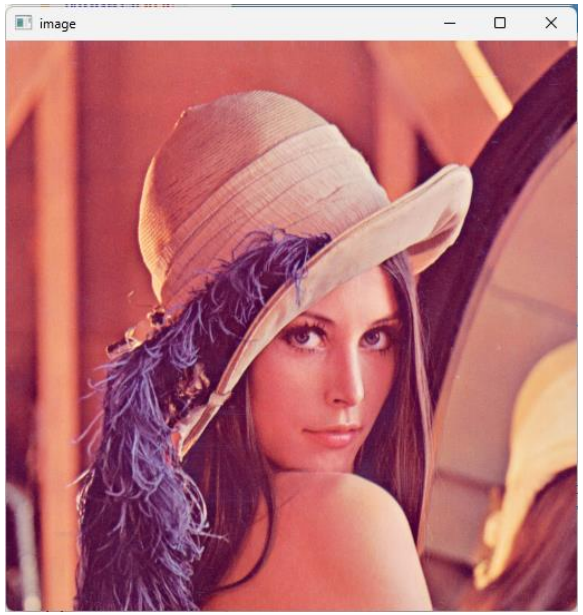
Original



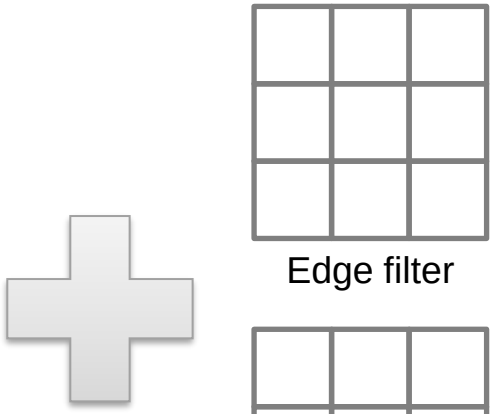
Sharpening

Filter

- Edge Filter
  - 이미지에서 밝기 변화가 급격하게 일어나는 경계 부분을 감지하여, 물체의 윤곽이나 경계를 추출하는 필터

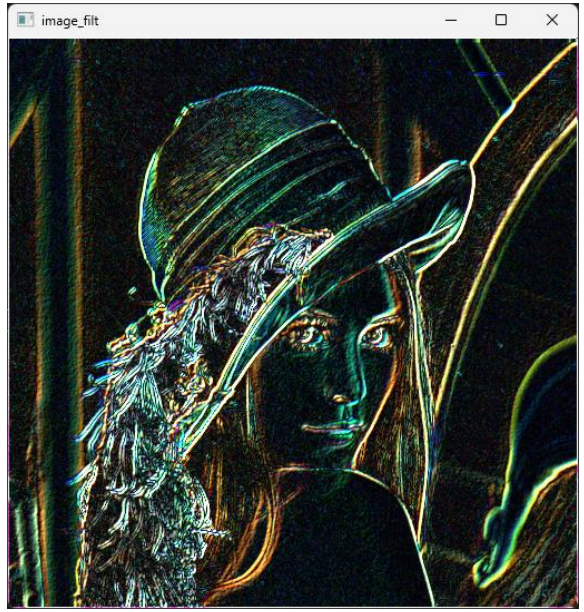


Original



Edge filter

Edge filter



Edge

Filter

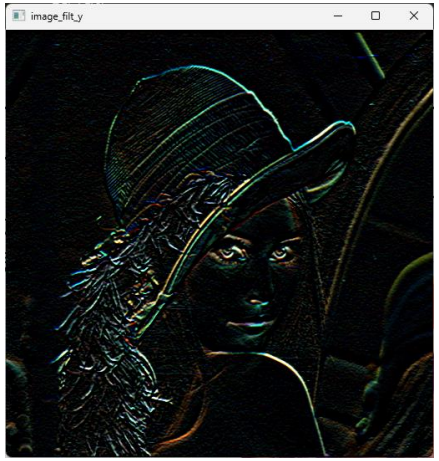
- Edge Filter – Sobel filter
  - 이미지의 수평 및 수직 방향의 경계를 감지하는 필터
  - x축 변화량 & y축 변화량 결과를 합하여 최종 결과를 얻을 수 있음
  - 필터의 총 합은 0

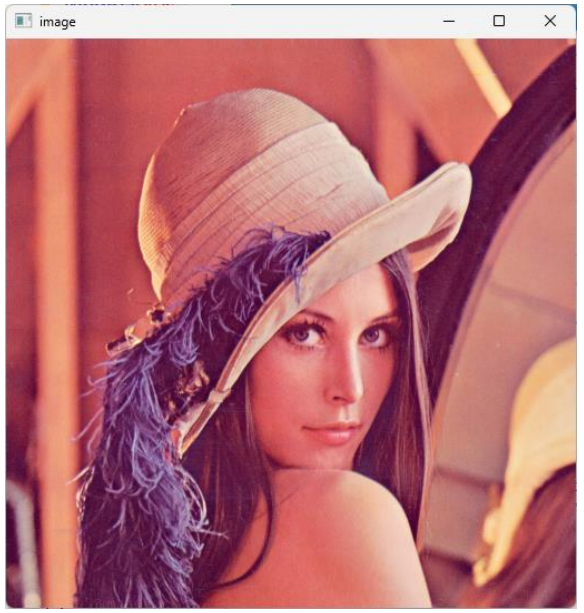
-1	0	1
-2	0	2
-1	0	1

gx

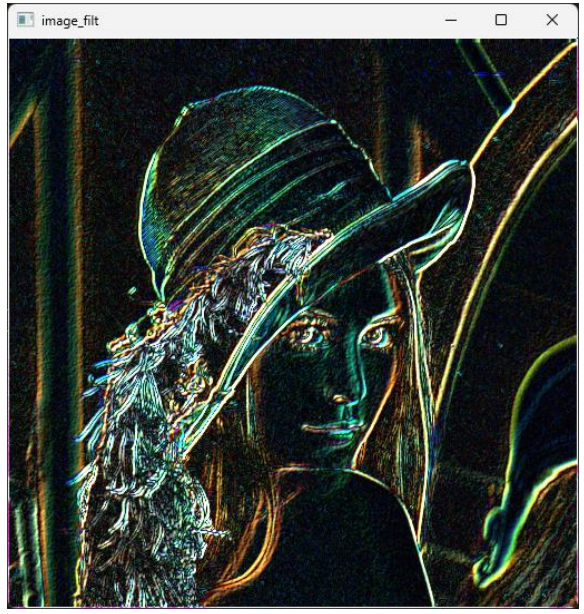
-1	-2	-1
0	0	0
1	2	1

gy





Original



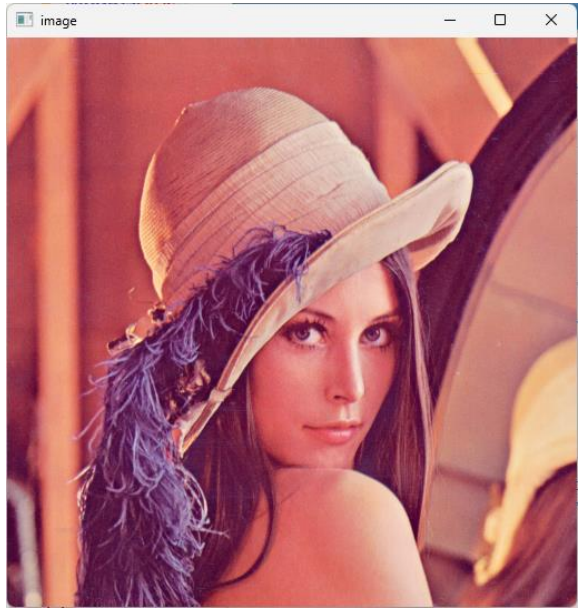
Edge

실습 및 과제

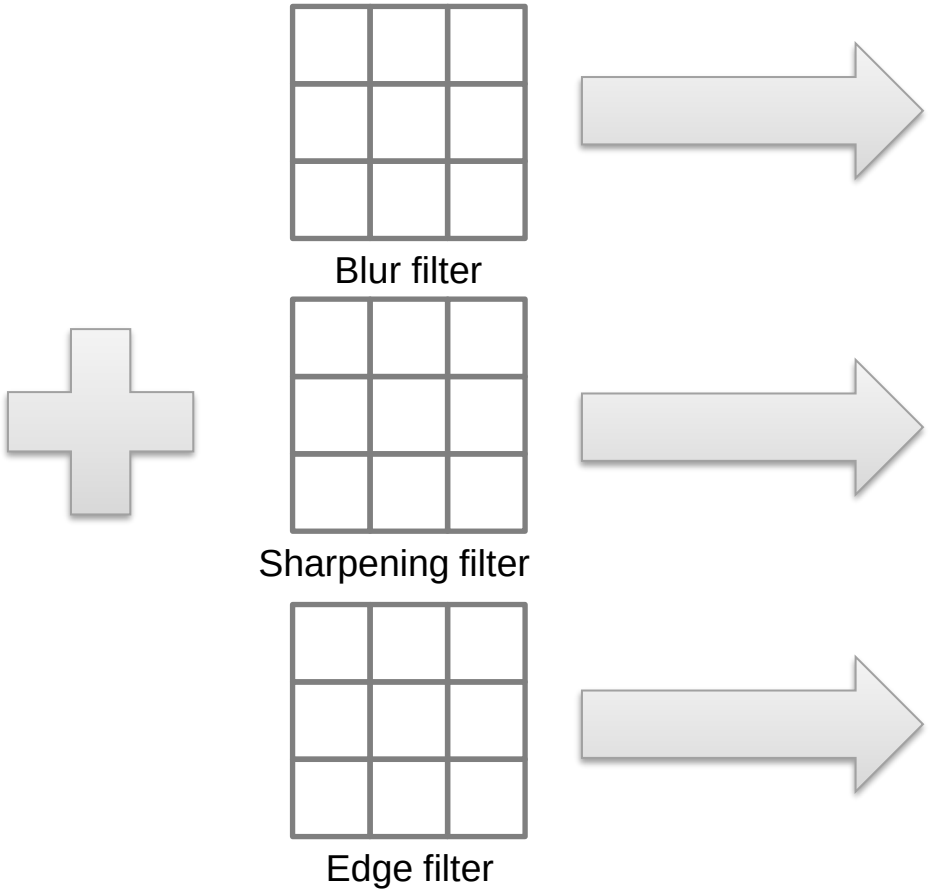
- Github : [Hwa-Jong/MyOpenCV: study Opencv (github.com)]

실습(IP6_2)

- Filtering



Original



Blur filter

Sharpening filter

Edge filter

실습(IP6_2)

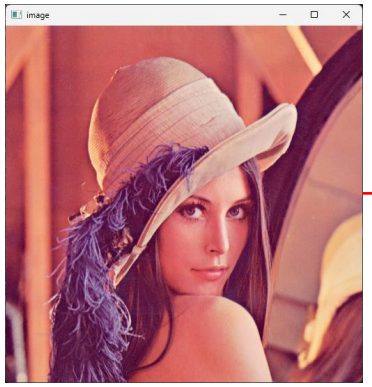
- Filtering
  - 현재 실습의 필터링 코드는 매우 비효율적인 코드
  - Padding의 종류는 변경 가능
  - Filter는 가장 간단한 mean filter를 사용

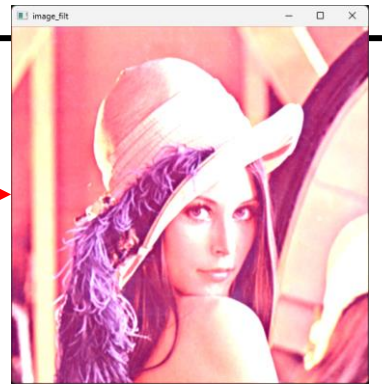
```python
def filtering(img, kernel, padding=0, last_norm=True):
    h_org, w_org = img.shape[:2]

    # padding
    if padding > 0:
        img = zero_padding(img, padding)

    # filtering
    img = img.astype(np.float32) / 255.0
    img_filter = np.zeros_like(img, dtype=np.float32)

    h, w, c = img_filter.shape
    kh, kw = kernel.shape
    kh_half, kw_half = kh//2, kw//2
```

```python
    print('filtering...')
    for row in range(padding, h-padding):
        for col in range(padding, w-padding):
            for ch in range(c):
                roi = img[row-kh_half:row+kw_half+1, col-kh_half:col+kw_half+1, ch]
                value = 0
                for row_k in range(kh):
                    for col_k in range(kw):
                        value += roi[row_k, col_k] * kernel[row_k, col_k]

                img_filter[row, col, ch] = value

    img_filter = img_filter[padding:padding+h_org, padding:padding+w_org]
    if last_norm:
        img_filter = np.clip(img_filter, 0, 1)
        img_filter = (img_filter * 255).astype(np.uint8)

    return img_filter
```

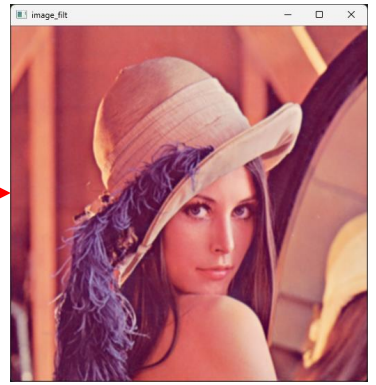
실습(IP6_2)

- Filtering
  - 왜 필터의 총 합을 1로 했을까?

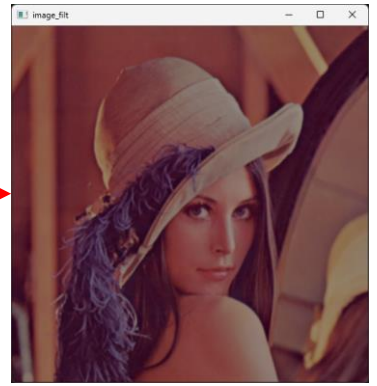




1/5



1/9



1/15

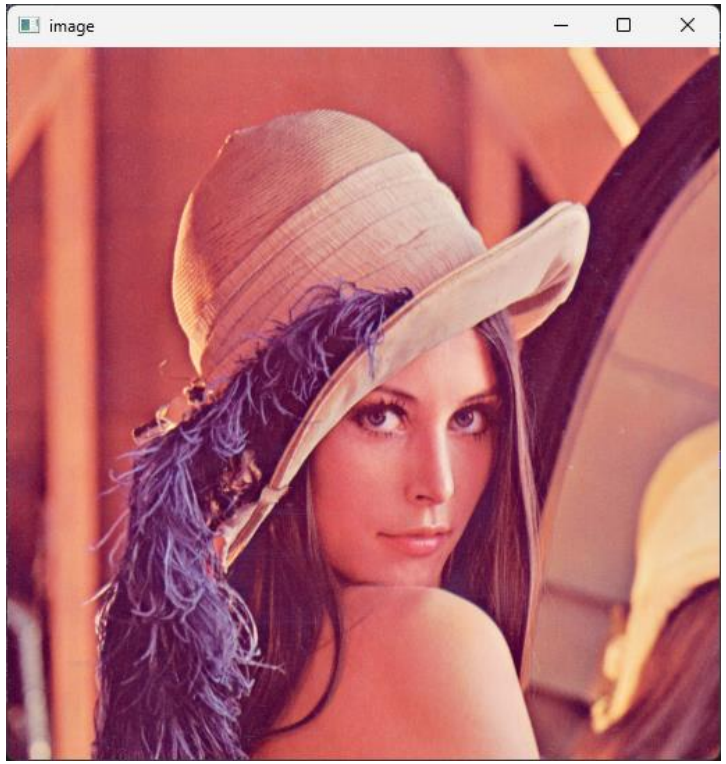
$$1/5 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$1/9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

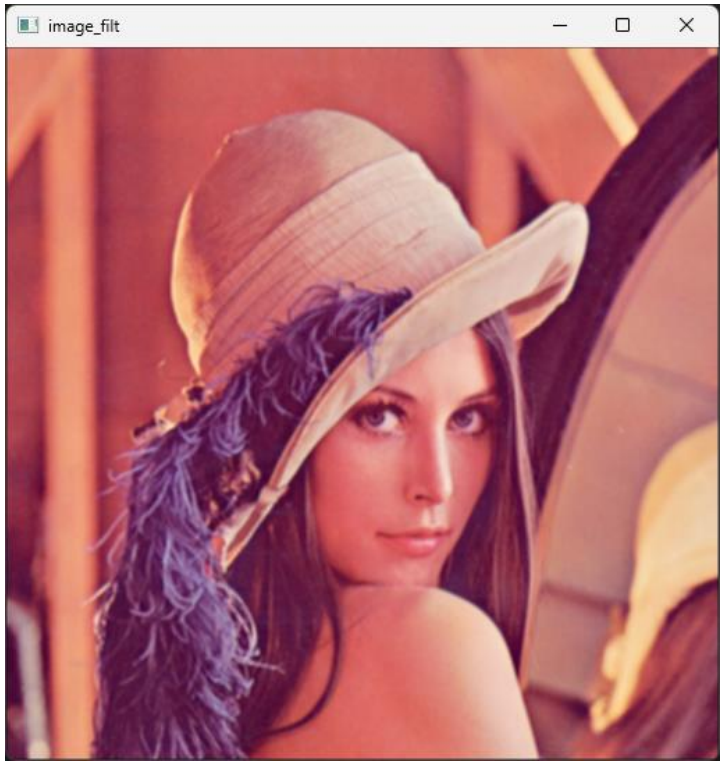
$$1/15 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

실습(IP6_2)

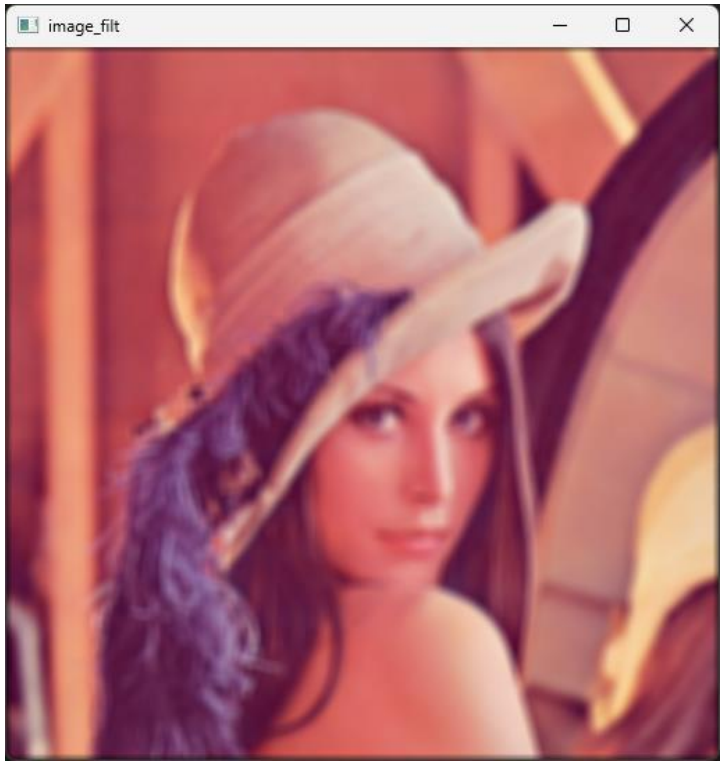
- Filtering
  - Kernel의 size는 어떤 역할을 할까?



Original



Size : 3



Size : 9

실습(IP6_3)

- Filtering
  - Mean filter
  - Gaussian filter
  - Sharpening filter
  - Sobel filter

```python
def get_box_kernel(size):
    kernel = np.full((size,size), 1/(size**2))
    return kernel
```

```python
def calc_gaussian_value(x, y, sigma):
    return 1/(2*np.pi*(sigma*sigma)) * np.exp( - ((x*x + y*y) / (2*(sigma*sigma))) )

def get_gaussian_kernel(size, sigma):
    kernel = np.zeros((size, size))

    h, w = kernel.shape

    for row in range(h):
        for col in range(w):
            y = row - h//2
            x = col - w//2
            kernel[row, col] = calc_gaussian_value(x, y, sigma)

    kernel = kernel / kernel.sum()

    return kernel
```

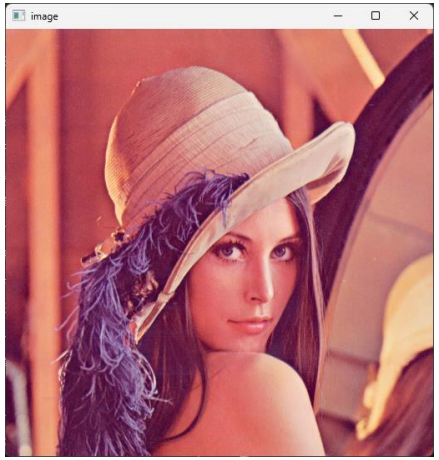
```python
def get_sharpening_kernel(size):
    kernel = np.full((size, size), -1.0)
    kernel[size//2, size//2] = (size * size)
    return kernel
```

```python
def get_sobel_filter():
    gx = np.array([ [-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1] ])

    gy = np.array([ [-1, -2, -1],
                    [ 0,  0,  0],
                    [ 1,  2,  1] ])

    return gx, gy
```

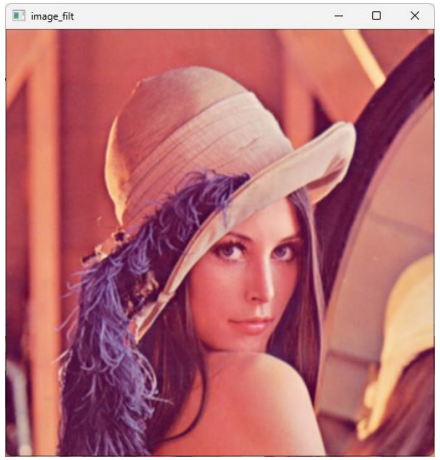
실습(IP6_3)

- Filtering
  - Mean filter
  - Gaussian filter
  - Sharpening filter
  - Sobel filter

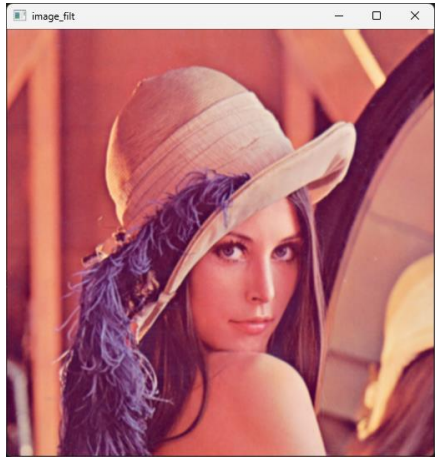
직접 코드를 돌려보면 된다



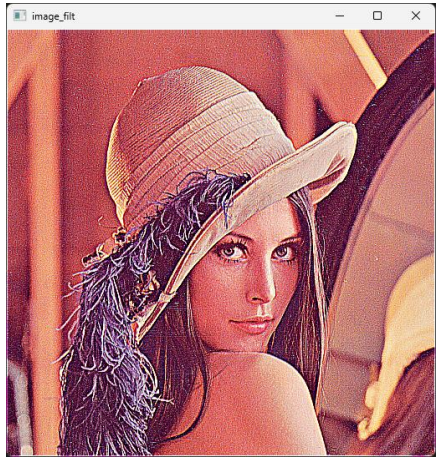
Original



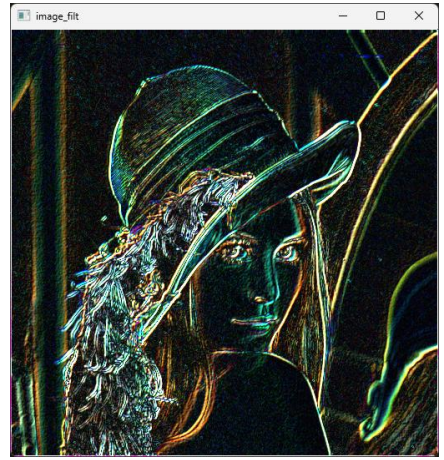
Mean



Gaussian



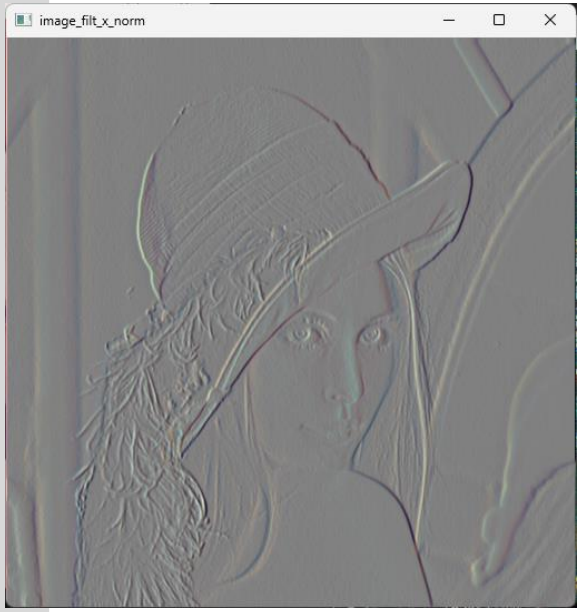
Sharpening



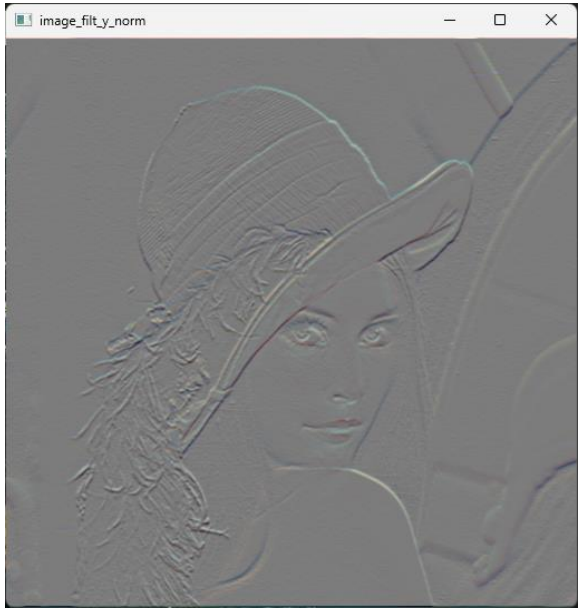
Sobel

실습(IP6_3)

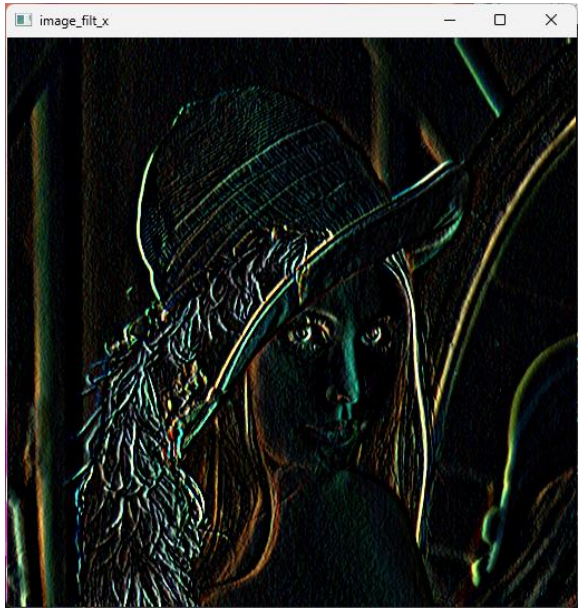
- Filtering
  - Sobel filter
    - 아래 이미지의 차이는 뭘까?



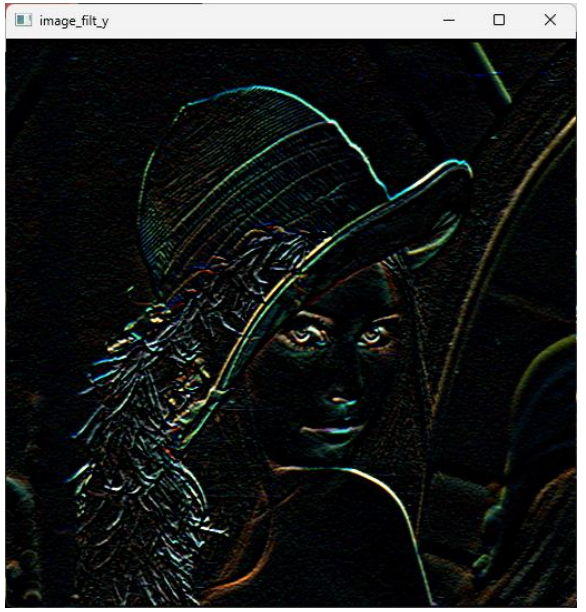
gx



gy



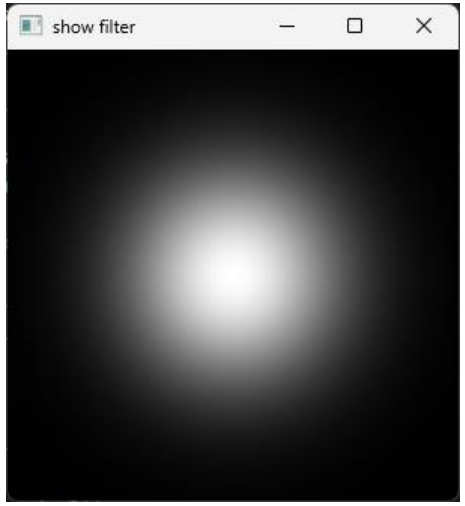
gx



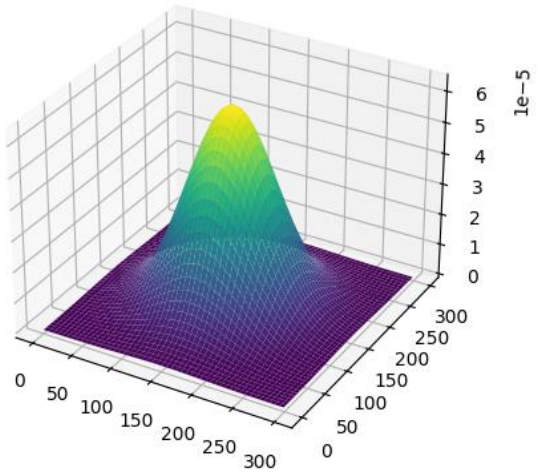
gy

실습(IP6_4)

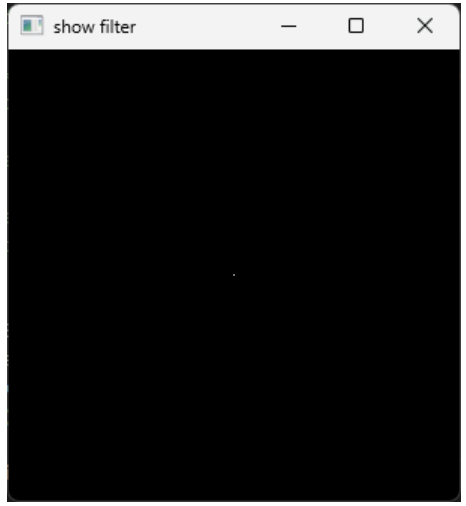
- Filter
  - 필터의 모양을 시각적으로 표현



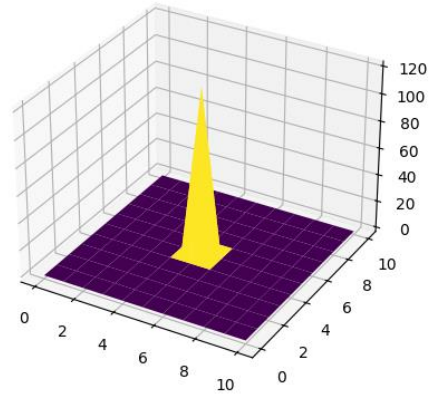
Gaussian filter 2D



Gaussian filter 3D



sharpening filter 2D

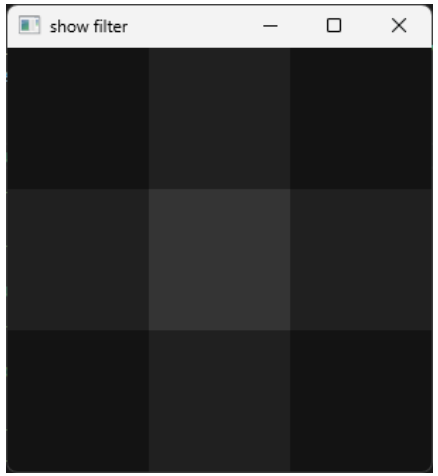


sharpening filter 3D

실습(IP6_4)

- Filter
  - 필터의 모양을 시각적으로 표현
    - Gaussian filter 실습

```
kernel_size = 3
kernel = get_gaussian_kernel(kernel_size, 1.0)
show_filter_2d(kernel)
```



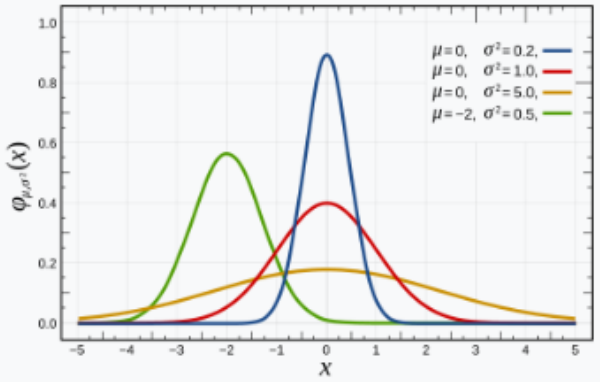

```
kernel_size = 3
kernel = get_gaussian_kernel(kernel_size, 1.0)
# show_filter_2d(kernel)

kernel_resize = cv2.resize(kernel, (300, 300), interpolation=cv2.INTER_NEAREST)
show_filter_2d(kernel_resize)
```

실습(IP6_4)

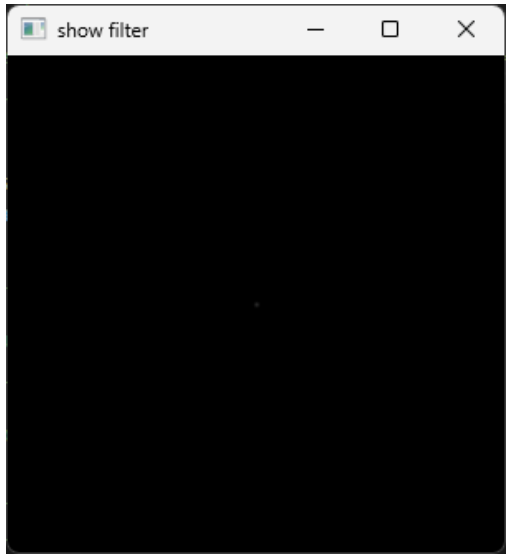
- Filter
  - 필터의 모양을 시각적으로 표현
    - Gaussian filter 실습

```python
def normalize(filter):
    filter -= filter.min()
    filter /= filter.max()
    filter *= 1
    return filter

def show_filter_2d(filter, norm=False):
    if norm:
        filter = normalize(filter)
    cv2.imshow('show filter', filter)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

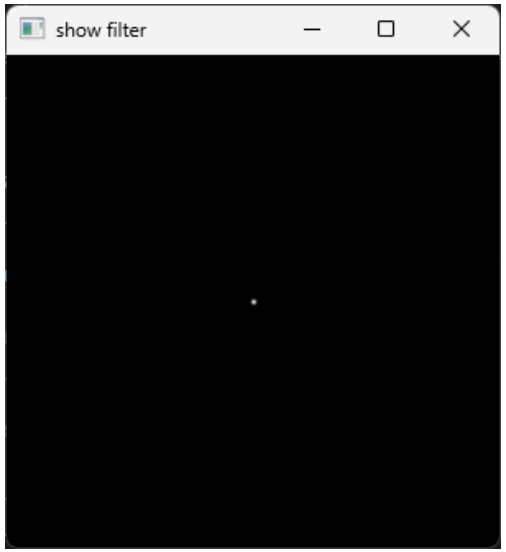
확률 밀도 함수



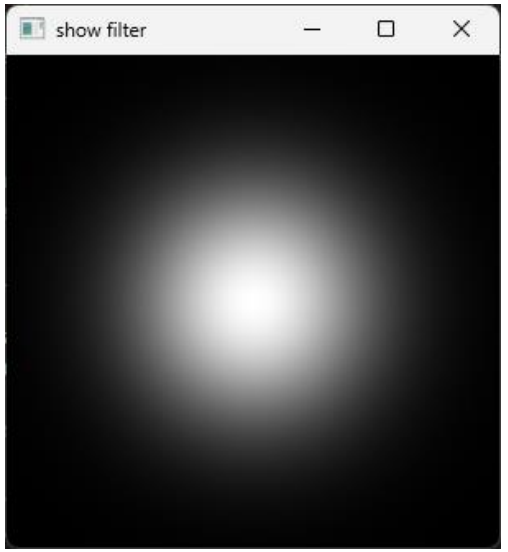
붉은 색은 표준정규분포




```python
kernel_size = 300
kernel_large = get_gaussian_kernel(kernel_size, 1.0)
show_filter_2d(kernel_large)
```



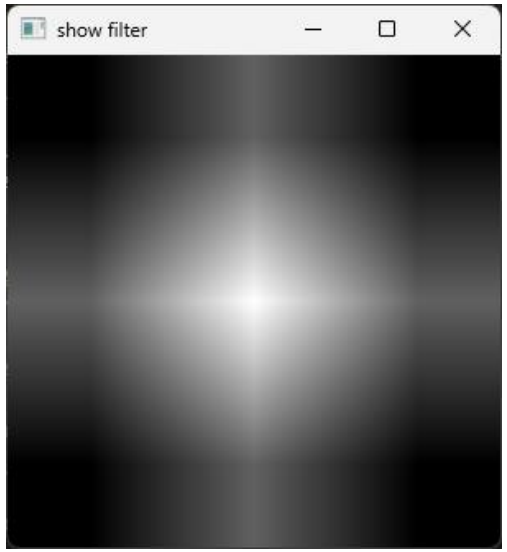

```python
kernel_size = 300
kernel_large = get_gaussian_kernel(kernel_size, 1.0)
# show_filter_2d(kernel_large)

show_filter_2d(kernel_large, norm=True)
```



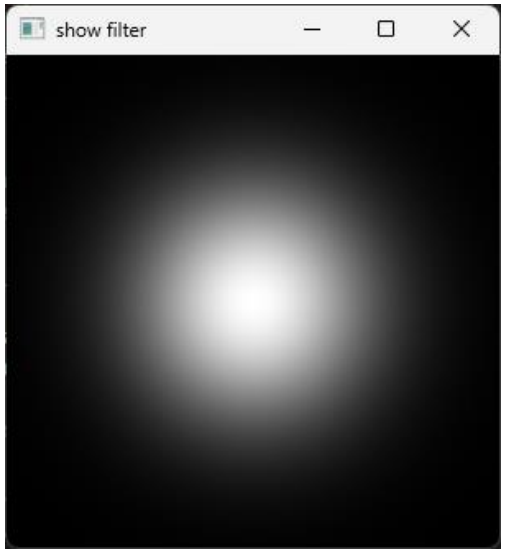

```python
kernel_size = 300
# kernel_large = get_gaussian_kernel(kernel_size, 1.0)
# # show_filter_2d(kernel_large)

# show_filter_2d(kernel_large, norm=True)

kernel_large = get_gaussian_kernel(kernel_size, 50.0)
show_filter_2d(kernel_large, norm=True)
```

실습(IP6_4)

- Filter
  - 필터의 모양을 시각적으로 표현
    - Gaussian filter 실습




```
kernel_size = 3
kernel = get_gaussian_kernel(kernel_size, 1.0)
# show_filter_2d(kernel)

# kernel_resize = cv2.resize(kernel, (300, 300), interpolation=cv2.INTER_NEAREST)
# show_filter_2d(kernel_resize)

kernel_resize = cv2.resize(kernel, (300, 300))
show_filter_2d(kernel_resize, norm=True)
```




```
kernel_size = 300
# kernel_large = get_gaussian_kernel(kernel_size, 1.0)
# # show_filter_2d(kernel_large)

# show_filter_2d(kernel_large, norm=True)

kernel_large = get_gaussian_kernel(kernel_size, 50.0)
show_filter_2d(kernel_large, norm=True)
```

과제(IP6_test3)

- Filtering 효율적으로 재구현하기
  - 5중 for문은 설명을 위해 구현한 코드
  - 속도를 빠르게 할 수 있는 방법을 찾아보기(numpy 활용)
  - 꼭 1줄로 작성할 필요는 없음

```
print('filtering...')
tmr = time.time()
for row in range(padding, h-padding):
    for col in range(padding, w-padding):
        for ch in range(c):
            roi = img[row-kh_half:row+kw_half+1, col-kh_half:col+kw_half+1, ch]
            value = 0
            for row_k in range(kh):
                for col_k in range(kw):
                    value += roi[row_k, col_k] * kernel[row_k, col_k]

            img_filter[row, col, ch] = value
print('filtering time : %f'%(time.time() - tmr))
```

```
filtering...
filtering time : 4.986633
```

```
print('filtering...')
for row in range(padding, h-padding):
    for col in range(padding, w-padding):
        img_filter[row, col] =
```

```
filtering...
filtering time : 2.709040
```

과제(IP6_test4)

- 2차 가우시안 필터 구현하기
  - numpy에 익숙해지기 위함(어려우면 구현하지 않아도 상관 없음)
  - 기존에는 x, y 각 좌표에 gaussian값을 계산하여 필터를 생성

```python
def calc_gaussian_value(x, y, sigma):
    return 1/(2*np.pi*(sigma*sigma)) * np.exp( - ((x*x + y*y) / (2*(sigma*sigma))) )

def get_gaussian_kernel(size, sigma):
    kernel = np.zeros((size, size))

    h, w = kernel.shape

    for row in range(h):
        for col in range(w):
            y = row - h//2
            x = col - w//2
            kernel[row, col] = calc_gaussian_value(x, y, sigma)

    kernel = kernel / kernel.sum()

    return kernel
```

```python
def get_gaussian_kernel(size, sigma):
    # hint : np.arange, np.reshape, np.tile
    x = 
    y = 

    kernel = 
    kernel = kernel / kernel.sum()

    return kernel
```

QnA