동양미래대학교

파이썬을 활용한 컴퓨터 비전 입문

Chapter 07. 필터링

동양미래대학교
인공지능소프트웨어학과
권 범

본 강의자료는 길벗 출판사의『OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝』
교재 내용을 토대로 작성되었습니다.

목차

❖ 7장 필터링
- 7.1 영상의 필터링
- 7.2 블러링: 영상 부드럽게 하기
- 7.3 샤프닝: 영상 날카롭게 하기
- 7.4 잡음 제거 필터링

7.1 영상의 필터링

7.2 블러링: 영상 부드럽게 하기

7.3 샤프닝: 영상 날카롭게 하기

7.4 잡음 제거 필터링

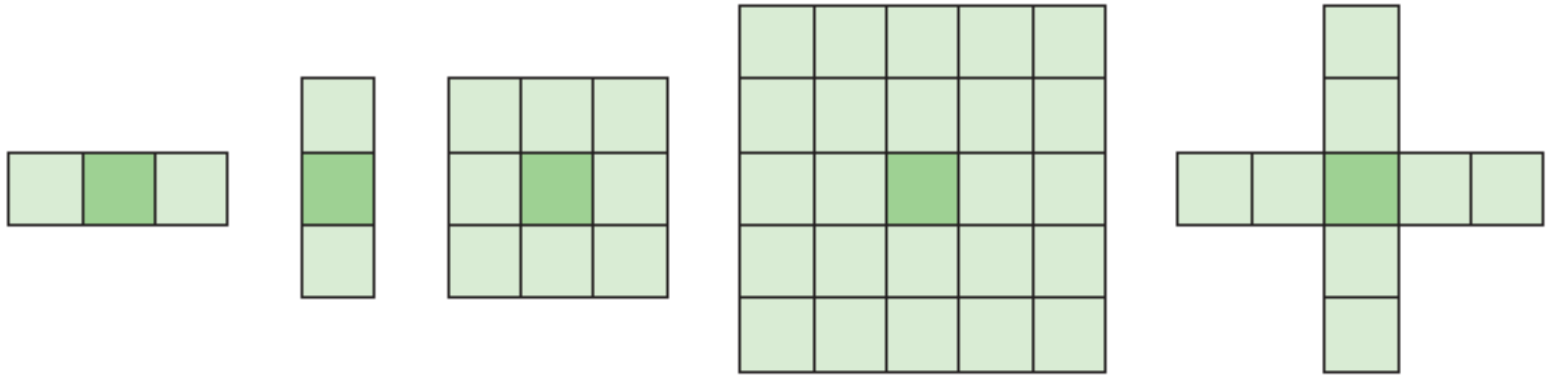
❖ 필터링 연산 방법 (1/14)

- 영상 처리에서 **필터링**(**filtering**)이란 **영상에서 원하는 정보만 통과시키고 원치 않는 정보는 걸러 내는 작업임**
- 영상에서 지저분한 잡음(noise)을 걸러 내어 영상을 깔끔하게 만드는 필터가 있음
- 부드러운 느낌의 성분을 제거함으로써 영상을 좀 더 날카로운 느낌이 나도록 만들 수도 있음
- 영상의 필터링은 보통 **마스크**(**mask**)라고 부르는 **작은 크기의 행렬을 이용함**
- 마스크는 필터링의 성격을 정의하는 행렬이며 **커널**(**kernel**), **윈도우**(**window**)라고도 부르며, 경우에 따라서는 마스크 자체를 **필터**라고 부르기도 함
- 마스크는 다양한 크기와 모양으로 정의할 수 있으며, 마스크 행렬의 원소는 보통 실수로 구성됨

❖ 필터링 연산 방법 (2/14)

- 1×3 또는 3×1 형태의 직사각형 행렬을 사용하기도 하고 3×3, 5×5 등 정방형 행렬을 사용하기도 함
- 필요하다면 십자가 모양의 마스크를 사용할 수도 있음
- 여러 가지 모양의 필터 마스크 중에서 3×3 정방형 행렬이 다양한 필터링 연산에서 가장 널리 사용되고 있음

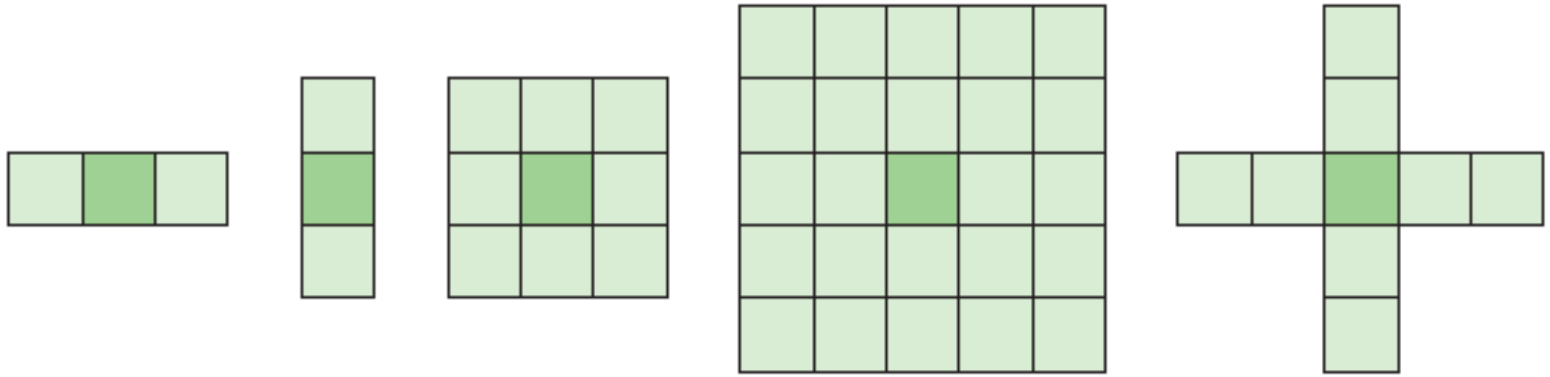
▼ **그림 7-1 다양한 필터 마스크 모양**



❖ 필터링 연산 방법 (3/14)

- 그림 7-1에 표시한 다양한 필터 마스크에서 진한 색으로 표시한 위치는 **고정점**(**anchor point**)을 나타냄
- 고정점은 현재 필터링 작업을 수행하고 있는 기준 픽셀 위치를 나타내고, 대부분의 경우 마스크 행렬 정중앙을 고정점으로 사용함

▼ **그림 7-1 다양한 필터 마스크 모양**

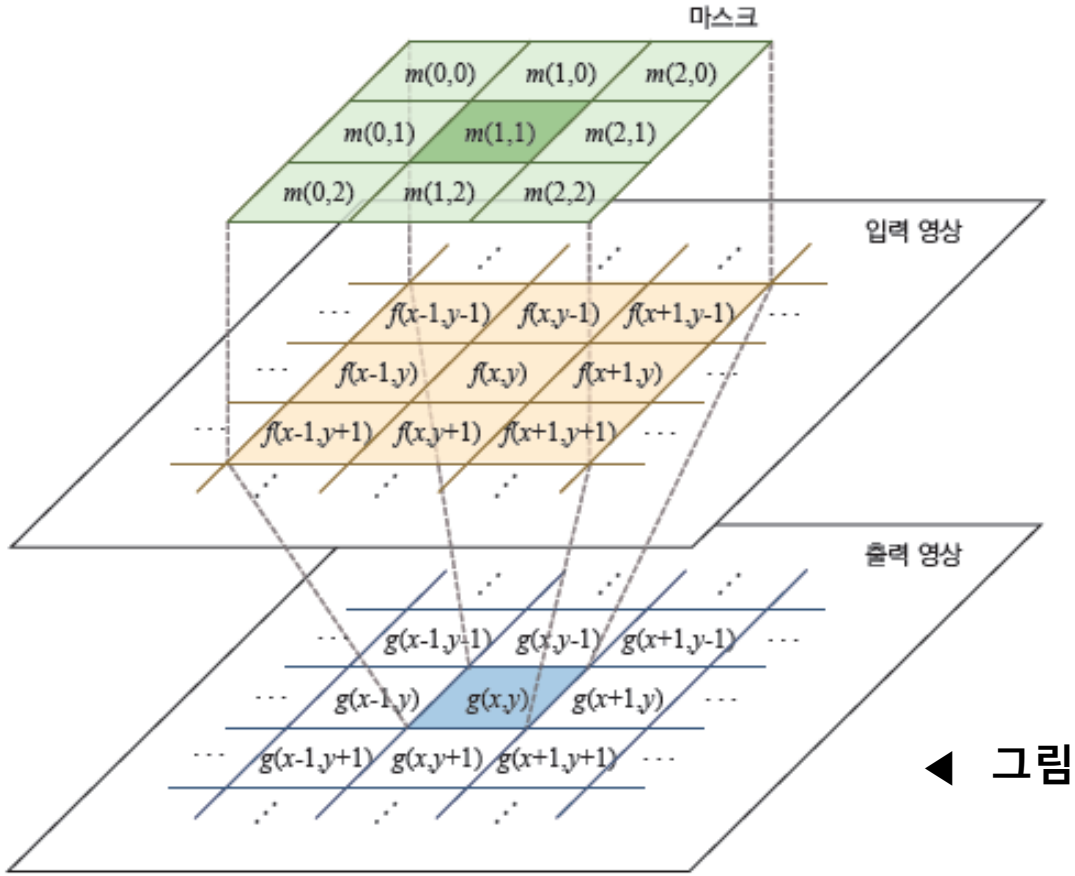


❖ 필터링 연산 방법 (4/14)

- 필터링 연산의 결과는 마스크 행렬의 모양과 원소 값에 의해 결정됨
- 마스크 행렬을 어떻게 정의하는가에 따라 영상을 전반적으로 부드럽게 만들 수도 있고, 반대로 날카롭게 만들 수도 있음
- 영상에서 잡음을 제거하거나 에지(edge) 성분만 나타나도록 만들 수도 있음
- 마스크를 이용한 필터링은 입력 영상의 모든 픽셀 위로 마스크 행렬을 이동시키면서 마스크 연산을 수행하는 방식으로 이루어짐
- **마스크 연산**이란 **마스크 행렬의 모든 원소에 대하여 마스크 행렬 원소 값과 같은 위치에 있는 입력 영상 픽셀 값을 서로 곱한 후, 그 결과를 모두 더하는 연산임**
- 마스크 연산의 결과를 출력 영상에서 고정점 위치에 대응되는 픽셀 값으로 설정함

❖ 필터링 연산 방법 (5/14)

- 마스크 행렬 m의 중심이 입력 영상의 (x, y) 좌표 위에 위치했을 때 필터링 결과 영상의 픽셀 값 $g(x, y)$는 다음과 같이 계산됨



$$
\begin{aligned}
g(x,y) = {} & m(0,0)f(x-1,y-1) + m(1,0)f(x,y-1) + m(2,0)f(x+1,y-1) \\
& + m(0,1)f(x-1,y) + m(1,1)f(x,y) + m(2,1)f(x+1,y) \\
& + m(0,2)f(x-1,y+1) + m(1,2)f(x,y+1) + m(2,2)f(x+1,y+1)
\end{aligned}
$$

◀ 그림 7-2 마스크를 이용한 필터링 연산 방법

❖ 필터링 연산 방법 (6/14)

- 영상의 가장자리 픽셀이란 영상에서 가장 왼쪽 또는 오른쪽 열, 가장 위쪽 또는 아래쪽 행에 있는 픽셀을 의미함
- 예를 들어 $(x, y) = (0,1)$ 위치에서 3×3 크기의 마스크 연산을 수행하는 경우, 결과 영상의 픽셀 값 $g(0,1)$은 다음과 같이 계산됨

$$\begin{aligned} g(0,1) &= m(0,0)f(-1,0) + m(1,0)f(0,0) + m(2,0)f(1,0) \\ &+ m(0,1)f(-1,1) + m(1,1)f(0,1) + m(2,1)f(1,1) \\ &+ m(0,2)f(-1,2) + m(1,2)f(0,2) + m(2,2)f(1,2) \end{aligned}$$

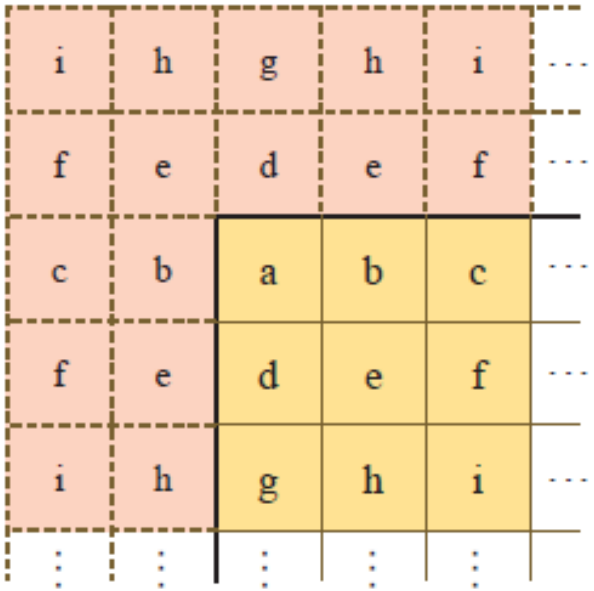
- 앞 수식에서 $x = -1$인 위치에서의 픽셀 값, 즉 $f(-1,0)$, $f(-1,1)$, $f(-1,2)$ 세 픽셀은 실제 영상에 존재하지 않음
- 이 수식은 적용할 수 없으며, **영상의 가장자리 픽셀에 대해 필터링을 수행할 때에는 특별한 처리를 해야 함**

❖ 필터링 연산 방법 (7/14)

- OpenCV는 영상의 필터링을 수행할 때, **영상의 가장자리 픽셀을 확장하여 영상 바깥쪽에 가상의 픽셀을 만듦**
- 영상의 바깥쪽 가상의 픽셀 값을 어떻게 설정하는가에 따라 필터링 연산 결과가 달라짐

❖ 필터링 연산 방법 (8/14)

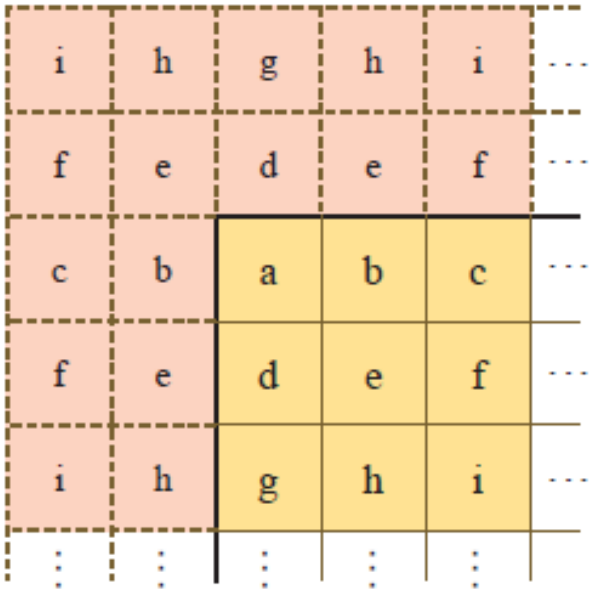
- 그림 7-3은 입력 영상의 좌측 상단 부분을 확대하여 나타낸 것으로 각각의 사각형은 픽셀을 표현함
- 실선으로 그려진 노란색 픽셀은 영상에 실제 존재하는 픽셀이고, 점선으로 표현된 바깥쪽 분홍색 픽셀은 필터링 연산 시 사용할 가상의 픽셀임



◀ 그림 7-3 필터링 연산을 위한 기본적인 가장자리 픽셀 확장 방법

❖ 필터링 연산 방법 (9/14)

- 이 그림에서는 5×5 크기의 필터 마스크를 사용하는 필터링을 고려하여 영상 바깥쪽에 두 개씩의 가상 픽셀을 표현함
- 각각의 픽셀에 쓰여진 영문자는 픽셀 값을 나타내며, 가상의 픽셀 위치에는 실제 영상의 픽셀 값이 대칭 형태로 나타나도록 설정되어 있음
- OpenCV는 이러한 가장자리 픽셀 확장 방법을 이용하여 영상의 가장자리 픽셀에 대해서도 문제없이 필터링 연산을 수행함



◀ **그림 7-3 필터링 연산을 위한 기본적인 가장자리 픽셀 확장 방법**

❖ 필터링 연산 방법 (10/14)

- 표 7-1에 나타난 상수는 BorderTypes라는 이름의 열거형 상수 중 일부임

▼ 표 7-1 OpenCV 필터링에서 가장자리 픽셀 처리 방법

BorderTypes 열거형 상수	설명
BORDER_CONSTANT	0 0 0 a b c d e f g h 0 0 0
BORDER_REPLICATE	a a a a b c d e f g h h h h
BORDER_REFLECT	c b a a b c d e f g h h g f
BORDER_REFLECT_101	d c b a b c d e f g h g f e
BORDER_REFLECT101	BORDER_REFLECT_101과 같음
BORDER_DEFAULT	BORDER_REFLECT_101과 같음

❖ 필터링 연산 방법 (11/14)

- OpenCV에서 필터 마스크를 사용하는 일반적인 필터링은 `filter2D()` 함수를 이용하여 수행함
- `filter2D()` 함수 원형은 다음과 같음

```
dst = cv2.filter2D(src, ddepth, kernel, delta, borderType)
```

`src`	입력 영상
`ddepth`	출력 영상의 dtype, -1로 지정하면 `src`의 dtype과 같아짐
`kernel`	필터링 커널, 1채널 실수형 행렬
`delta`	필터링 연산 후 추가적으로 더할 값
`borderType`	가장자리 픽셀 확장 방식

❖ 필터링 연산 방법 (12/14)

- `filter2D()` 함수는 `src` 영상에 `kernel` 필터를 이용하여 필터링을 수행하고, 그 결과를 `dst`에 저장함
- `filter2D()` 함수가 수행하는 연산을 수식으로 표현하면 다음과 같음

$$\mathrm{dst}(x, y) = \sum_j \sum_i \mathrm{kernel}(i, j) \cdot \mathrm{src}(x + i - \mathrm{anchor.x}, y + j - \mathrm{anchor.y}) + \mathrm{delta}$$

❖ 필터링 연산 방법 (13/14)

- filter2D() 함수 인자 중에서 ddepth는 결과 영상의 깊이를 지정하는 용도로 사용함
- 만약 ddepth에 -1을 지정하면 출력 영상의 깊이는 입력 영상과 같게 설정됨

여기서 영상의 깊이는 데이터 타입을 의미합니다.

▼ 표 7-2 입력 영상의 깊이에 따라 지정 가능한 ddepth 값

입력 영상의 깊이(src.depth())	지정 가능한 ddepth 값
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

❖ 필터링 연산 방법 (14/14)

- `filter2D()` 함수에서 `anchor`, `delta`, `borderType` 인자는 기본값을 가지고 있기 때문에 생략할 수 있음
- `anchor` 인자는 커널 행렬에서 고정점으로 사용할 좌표이며, 기본값인 Point(−1, −1)을 지정하면 커널 행렬 중심 좌표를 고정점으로 사용함
- `delta` 인자에는 필터링 연산 후 결과 영상에 추가적으로 더할 값을 지정할 수 있으며, 기본값은 0임
- `borderType` 인자에는 앞서 표 7-1에 나타낸 `borderType` 열거형 상수 중 하나를 지정할 수 있음

❖ 엠보싱 필터링 (1/5)

- 엠보싱 필터는 입력 영상을 엠보싱 느낌이 나도록 변환하는 필터임
- 보통 입력 영상에서 픽셀 값 변화가 적은 평탄한 영역은 회색으로 설정함
- 객체의 경계 부분은 좀 더 밝거나 어둡게 설정하면 엠보싱 느낌이 남

emboss: v. 부각하다. 돋을새김하다.

❖ 엠보싱 필터링 (2/5)

- 이 필터 마스크를 사용하여 필터링을 수행하면 대각선 방향으로 픽셀 값이 급격하게 변하는 부분에서 결과 영상 픽셀 값이 0보다 훨씬 크거나 또는 0보다 훨씬 작은 값을 가지게 됨
- 입력 영상에서 픽셀 값이 크게 바뀌지 않는 평탄한 영역에서는 결과 영상의 픽셀 값이 0에 가까운 값을 가지게 됨
- 이렇게 구한 결과 영상을 그대로 화면에 나타내면 음수 값은 모두 포화 연산에 의해 0이 되어 버리기 때문에 입체감이 크게 줄어들게 됨
- 엠보싱 필터를 구현할 때에는 결과 영상에 128을 더하는 것이 보기에 좋음 (`delta`로 지정)

▼ **그림 7-4 엠보싱 필터 마스크**

-1	-1	0
-1	0	1
0	1	1

❖ 엠보싱 필터링 (3/5)

- 코드 7-1의 `filter_embossing()` 함수는 rose.bmp 장미 영상에 엠보싱 필터링을 수행하고 그 결과를 화면에 출력함

코드 7-1 엠보싱 필터링 예제 (emboss.py)

```
1   import sys
2   import numpy as np
3   import cv2
4
5   src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)
6
7   if src is None:
8         print('Image load failed!')
9         sys.exit()
10
11  emboss = np.array([[-1, -1, 0],
12                     [-1, 0, 1],
13                     [0, 1, 1]], np.float32)
14
15  dst = cv2.filter2D(src, -1, emboss, delta=128)
16
17  cv2.imshow('src', src)
18  cv2.imshow('dst', dst)
19
20  cv2.waitKey()
21  cv2.destroyAllWindows()
```

❖ 엠보싱 필터링 (4/5)

- emboss.py 소스 코드 설명

  - 5행 rose.bmp 레나 영상을 `grayscale` 형식으로 불러와 `src`에 저장합니다.

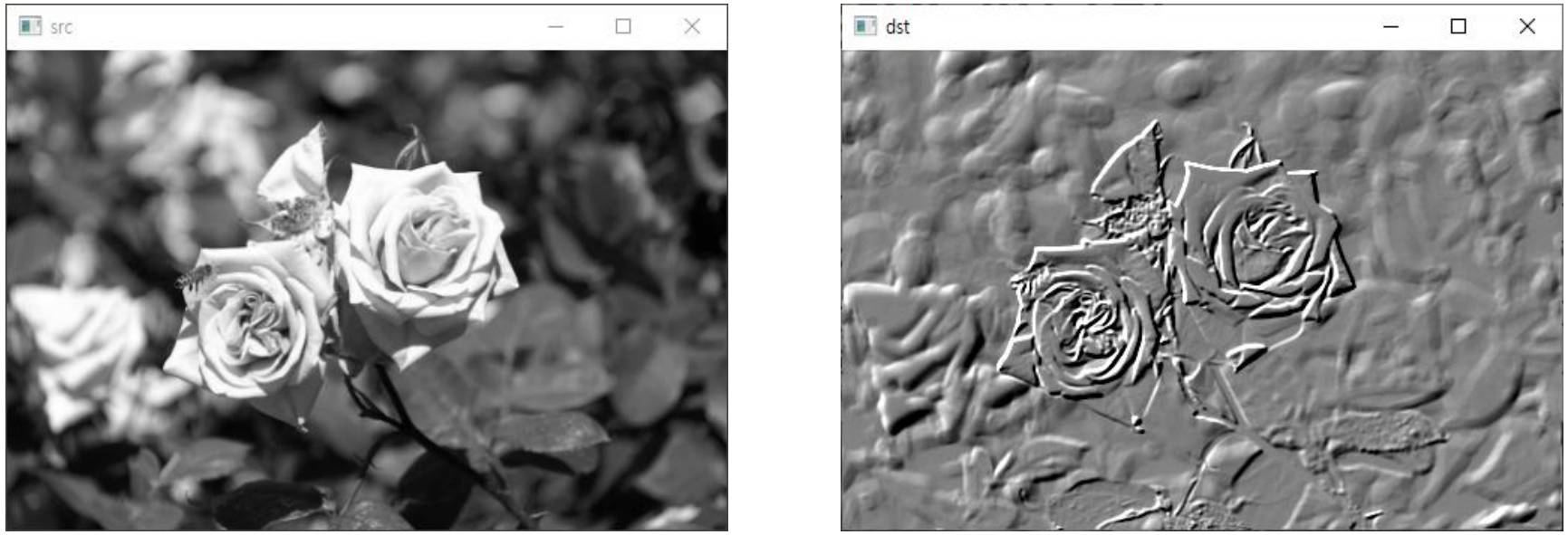
  - 11~13행 numpy.ndarray 배열을 이용하여 3×3 크기의 엠보싱 필터 마스크 행렬 `emboss`를 생성합니다.

  - 15행 `filter2D()` 함수를 이용하여 엠보싱 필터링을 수행합니다.
    이때 `filter2D()` 함수 `delta` 인자에 128을 지정하여 필터링 결과 영상에 128을 더합니다.

❖ 엠보싱 필터링 (5/5)

- 그림 7-5에서 src는 rose.bmp 장미 입력 영상이고, dst는 엠보싱 필터링이 적용된 결과 영상임
- dst 영상에서 장미꽃 경계 부분이 입체감 있게 표현된 것을 확인할 수 있음
- 픽셀 값이 완만하게 바뀌는 부분에서는 필터링 결과 영상이 대체로 밝기 값 128에 가까운 회색으로 표현됨

▼ **그림 7-5 엠보싱 필터링 예제 실행 결과**



7.2 블러링: 영상 부드럽게 하기

7.1 영상의 필터링

7.3 샤프닝: 영상 날카롭게 하기

7.4 잡음 제거 필터링

❖ 블러링: 영상 부드럽게 하기

- **블러링**(**blurring**)은 **마치 초점이 맞지 않은 사진처럼 영상을 부드럽게 만드는 필터링 기법**이며 **스무딩**(**smoothing**)이라고도 함
- 영상에서 인접한 픽셀 간의 픽셀 값 변화가 크지 않은 경우 부드러운 느낌을 받을 수 있음
- 블러링은 거친 느낌의 입력 영상을 부드럽게 만드는 용도로 사용되기도 하고,
  혹은 입력 영상에 존재하는 잡음의 영향을 제거하는 전처리 과정으로도 사용됨

❖ 블러링: ① 평균값 필터 (1/8)

- 평균값 필터는 입력 영상에서 특정 픽셀과 주변 픽셀들의 산술 평균을 결과 영상 픽셀 값으로 설정하는 필터임
- 평균값 필터에 의해 생성되는 결과 영상은 픽셀 값의 급격한 변화가 줄어들어 날카로운 에지가 무뎌지고 잡음의 영향이 크게 사라지는 효과가 있음
- 평균값 필터를 너무 과도하게 사용할 경우 사물의 경계가 흐릿해지고 사물의 구분이 어려워질 수 있음

❖ 블러링: ① 평균값 필터 (2/8)

- 각각의 행렬은 모두 원소 값이 1로 설정되어 있고, 행렬의 전체 원소 개수로 각 행렬 원소 값을 나누는 형태로 표현되어 있음
- 평균값 필터는 마스크의 크기가 커지면 커질수록 더욱 부드러운 느낌의 결과 영상을 생성하며, 그 대신 연산량이 크게 증가할 수 있음

▼ **그림 7-6 다양한 크기의 평균값 필터 마스크**

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

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

❖ 블러링: ① 평균값 필터 (3/8)

- OpenCV에서는 `blur()` 함수를 이용하여 평균값 필터링을 수행할 수 있음
- `blur()` 함수의 사용법은 다음과 같음

```
dst = cv2.blur(src, (ksize, ksize), borderType)
```

`src`	입력 영상. 다채널 영상은 각 채널별로 블러링을 수행합니다.
`ksize`	블러링 커널 크기
`borderType`	가장자리 픽셀 확장 방식

❖ 블러링: ① 평균값 필터 (4/8)

- `blur()` 함수는 `src` 영상에 `ksize` 크기의 평균값 필터 마스크를 사용하여 `dst` 출력 영상을 생성함
- `anchor` 인자와 `borderType` 인자는 기본값을 가지고 있으므로 함수 호출 시 생략할 수 있음
- `blur()` 함수에서 사용하는 커널은 다음과 같은 형태를 가짐

$$\text{kernel} = \frac{1}{\text{ksize.width} \times \text{ksize.height}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

❖ 블러링: ① 평균값 필터 (5/8)

- 코드 7-2의 blurring_mean() 함수는 3×3, 5×5, 7×7 크기의 평균값 필터를 이용하여 rose.bmp 장미 영상을 부드럽게 변환하고 그 결과를 화면에 출력함

코드 7-2 평균값 필터를 이용한 블러링 (blurring.py)

```
1   import cv2
2
3   def blurring_mean():
4       src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)
5
6       if src is None:
7           print('Image load failed!')
8           return
9
10      cv2.imshow('src', src)
11
```

❖ 블러링: ① 평균값 필터 (6/8)

코드 7-2 평균값 필터를 이용한 블러링 (blurring.py)

```
12      for ksize in (3, 5, 7):
13          dst = cv2.blur(src, (ksize, ksize))
14
15          desc = "Mean: %dx%d" % (ksize, ksize)
16          cv2.putText(dst, desc, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
17                      1.0, 255, 1, cv2.LINE_AA)
18
19          cv2.imshow('dst', dst)
20          cv2.waitKey()
21
22      cv2.destroyAllWindows()
23
24  if __name__ == '__main__':
25      blurring_mean()
```

cv2.putText(창 이름, 문자열, 좌표, 폰트, 폰트 스케일, 색상, 선 타입)

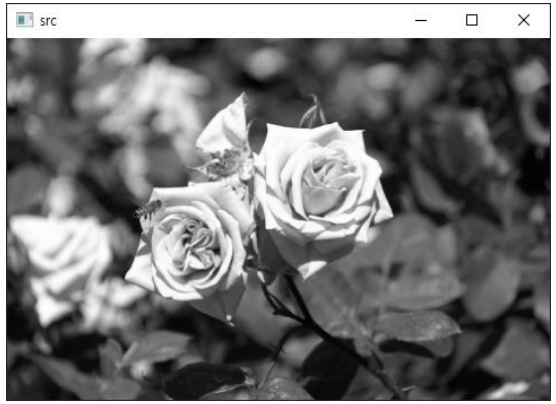
❖ 블러링: ① 평균값 필터 (7/8)

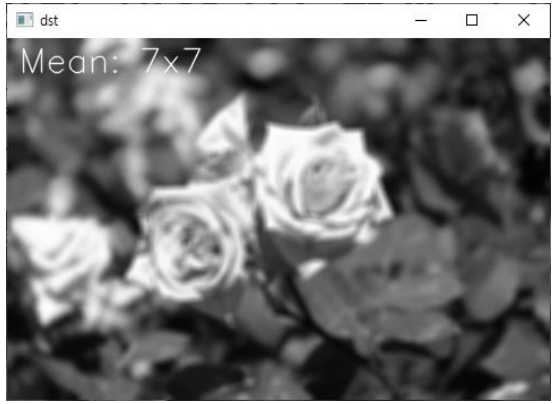
- blurring.py 소스 코드 설명
  - 12행 `ksize` 값이 3, 5, 7이 되도록 for 반복문을 설정합니다.
  - 13행 `ksize×ksize` 크기의 평균값 필터 마스크를 이용하여 블러링을 수행합니다.
  - 15~7행 사용된 평균값 필터의 크기를 문자열 형태로 결과 영상 `dst` 위에 출력합니다.

❖ 블러링: ① 평균값 필터 (8/8)

- 그림 7-7에서 src는 입력 영상인 rose.bmp 파일이고, dst는 blur() 함수에 의해 생성된 블러링 결과 영상임
- 평균 값 필터의 크기가 커질수록 결과 영상이 더욱 부드럽게 변경되는 것을 확인할 수 있음

▼ **그림 7-7 평균값 필터를 이용한 블러링 실행 결과**





❖ 블러링: ② 가우시안 필터 (1/19)

- **가우시안 분포**는 평균을 중심으로 좌우 대칭의 종 모양(bell shape)을 갖는 확률 분포를 말하며 **정규 분포**(**normal distribution**)라고도 함
- 자연계에서 발생하는 대부분의 사건은 가우시안 분포를 따르는 것으로 알려져 있음
- 가우시안 분포는 평균과 표준 편차에 따라 분포 모양이 결정됨
- 다만 영상의 가우시안 필터에서는 주로 평균이 0인 가우시안 분포 함수를 사용함
- 평균이 0이고 표준 편차가 σ인 1차원 가우시안 분포를 함수식으로 나타내면 다음과 같음

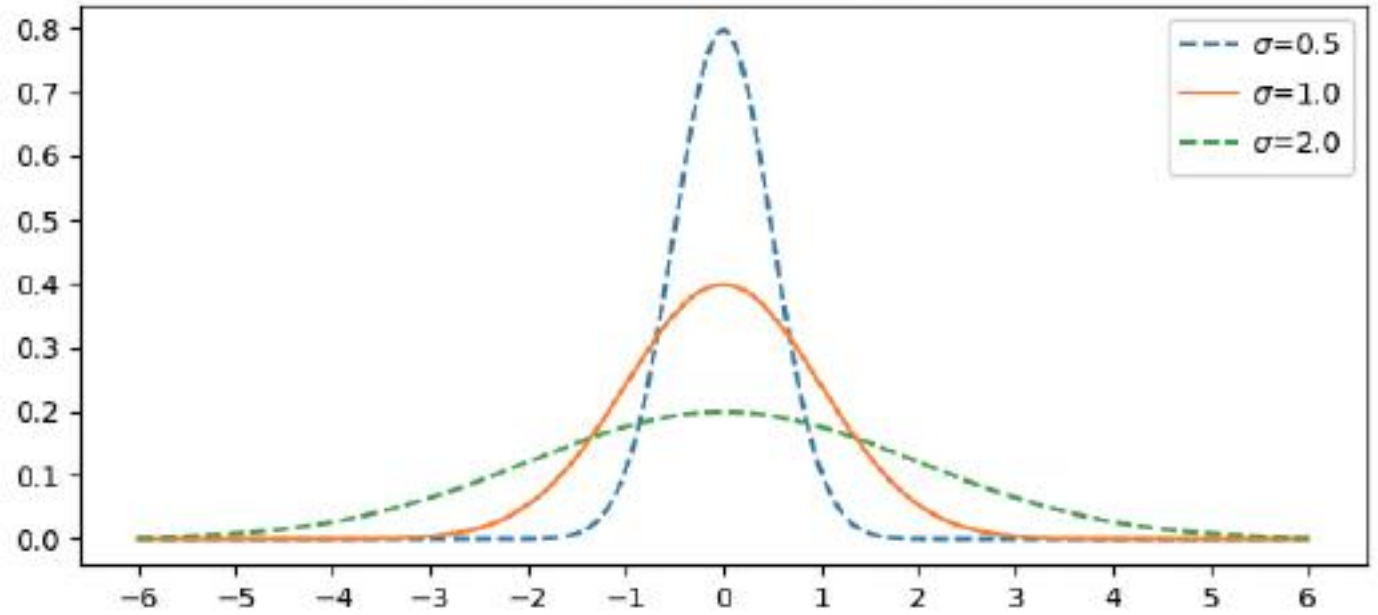
$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}$$

❖ 블러링: ② 가우시안 필터 (2/19)

- 세 개의 그래프가 모두 평균이 0이므로 $x = 0$에서 최댓값을 가지며, x가 0에서 멀어질수록 함수 값이 감소함
- 표준 편차 σ가 작으면 가우시안 분포 함수 그래프가 뾰족한 형태가 됨
- 반대로 표준 편차 σ가 크면 그래프가 넓게 퍼지면서 완만한 형태를 따름
- 가우시안 분포 함수 값은 특정 x가 발생할 수 있는 확률의 개념을 가지며, 그래프 아래 면적을 모두 더하면 1이 됨

❖ 블러링: ② 가우시안 필터 (3/19)

▼ 그림 7-8 평균이 0인 1차원 가우시안 분포 함수 그래프



❖ 블러링: ② 가우시안 필터 (4/19)

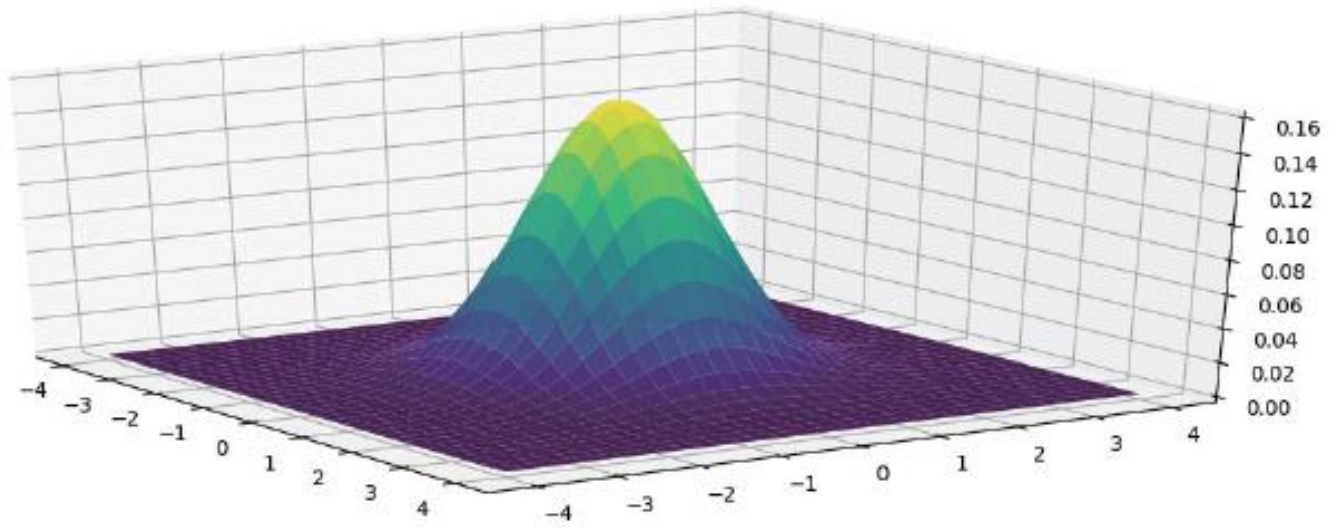
- 가우시안 분포를 따르는 2차원 필터 마스크 행렬을 생성하려면 2차원 가우시안 분포 함수를 근사해야 함
- 2차원 가우시안 분포 함수는 x와 y 두 개의 변수를 사용하고, 분포의 모양을 결정하는 평균과 표준 편차도 x축과 y축 방향에 따라 따로 설정함
- 평균이 $(0, 0)$이고 x축과 y축 방향의 표준 편차가 각각 σ_x, σ_y인 2차원 가우시안 분포 함수는 다음과 같이 정의됨

$$G_{\sigma_x \sigma_y}(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)}$$

❖ 블러링: ② 가우시안 필터 (5/19)

- 2차원 가우시안 분포 함수 그래프는 1차원 가우시안 분포 함수 그래프의 차원을 확장한 형태임
- 평균이 (0, 0)이므로 그림 7-9 그래프는 (0, 0)에서 최댓값을 갖고, 평균에서 멀어질수록 함수가 감소함
- 2차원 가우시안 분포 함수의 경우, 함수 그래프 아래의 부피를 구하면 1이 됨

▼ **그림 7-9 평균이 (0, 0)인 2차원 가우시안 함수 그래프($\sigma_x = \sigma_y = 1.0$)**



❖ 블러링: ② 가우시안 필터 (6/19)

- 가우시안 필터는 이러한 2차원 가우시안 분포 함수로부터 구한 마스크 행렬을 사용함
- 가우시안 분포 함수는 연속 함수이지만 이산형의 마스크를 만들기 위해서 x와 y 값이 정수인 위치에서만 가우시안 분포 함수 값을 추출하여 마스크를 생성함
- 평균이 0이고 표준 편차가 σ인 가우시안 분포는 x가 -4σ부터 4σ 사이인 구간에서 그 값의 대부분이 존재하기 때문에 가우시안 필터 마스크의 크기는 보통 $(8\sigma + 1)$로 결정함

❖ 블러링: ② 가우시안 필터 (7/19)

▼ 그림 7-10 $\sigma_x = \sigma_y = 1.0$인 경우의 가우시안 필터 마스크

$$\mathbf{G} = \begin{pmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0001 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0011 & 0.0018 & 0.0011 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0002 & 0.0029 & 0.0131 & 0.0215 & 0.0131 & 0.0029 & 0.0002 & 0.0000 \\ 0.0000 & 0.0011 & 0.0131 & 0.0586 & 0.0965 & 0.0586 & 0.0131 & 0.0011 & 0.0000 \\ 0.0001 & 0.0018 & 0.0215 & 0.0965 & 0.1592 & 0.0965 & 0.0215 & 0.0018 & 0.0001 \\ 0.0000 & 0.0011 & 0.0131 & 0.0586 & 0.0965 & 0.0586 & 0.0131 & 0.0011 & 0.0000 \\ 0.0000 & 0.0002 & 0.0029 & 0.0131 & 0.0215 & 0.0131 & 0.0029 & 0.0002 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0011 & 0.0018 & 0.0011 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0001 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{pmatrix}$$

❖ 블러링: ② 가우시안 필터 (8/19)

- 이 필터 마스크를 이용하여 마스크 연산을 수행한다는 것은 **필터링 대상 픽셀 근처에는 가중치를 크게 줌**
- 필터링 대상 픽셀과 **멀리 떨어져 있는 주변부에는 가중치를 조금만 주어서 가중 평균(weighted average)을 구하는 것과 같음**
- 가우시안 필터 마스크가 가중 평균을 구하기 위한 가중치 행렬 역할을 하는 것임

❖ 블러링: ② 가우시안 필터 (9/19)

- 마스크 연산에 의한 영상 필터링은 마스크 크기가 커짐에 따라 연산량도 함께 증가함
- 그림 7-10의 경우, 마스크 행렬 크기가 9×9이기 때문에 한 번의 마스크 연산 시 81번의 곱셈 연산이 필요함
- 큰 표준 편차 값을 사용하면 마스크 크기도 함께 커지므로 연산 속도 측면에서 부담이 될 수 있음
- 다행히도 **2차원 가우시안 분포 함수는 1차원 가우시안 분포 함수의 곱으로 분리할 수 있으며, 이러한 특성을 이용하면 가우시안 필터 연산량을 크게 줄일 수 있음**

❖ 블러링: ② 가우시안 필터 (10/19)

- 2차원 가우시안 분포 함수 수식은 다음과 같이 분리하여 작성할 수 있음

$$G_{\sigma_x\sigma_y}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)}$$

$$= \frac{1}{\sqrt{2\pi}\sigma_x} e^{-\frac{x^2}{2\sigma_x^2}} \times \frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{y^2}{2\sigma_y^2}} = G_{\sigma_x}(x)\cdot G_{\sigma_y}(y)$$

- 앞 수식에서 2차원 가우시안 분포 함수가 x축과 y축 방향의 1차원 가우시안 분포 함수의 곱으로 분리되는 것을 볼 수 있음
- 2차원 필터 마스크 생성 함수를 x축 방향으로의 함수와 y축 방향으로의 함수로 각각 분리할 수 있을 경우, 입력 영상을 x축 방향으로의 함수와 y축 방향으로의 함수로 각각 1차원 마스크 연산을 수행함으로써 필터링 결과 영상을 얻을 수 있음

❖ 블러링: ② 가우시안 필터 (11/19)

- 실제로 $\sigma = 1.0$인 1차원 가우시안 함수로부터 1×9 가우시안 마스크 행렬은 다음과 같음

$$\mathbf{g} = \begin{pmatrix} 0.0001 & 0.0044 & 0.0540 & 0.2420 & 0.3989 & 0.2420 & 0.0540 & 0.0044 & 0.0001 \end{pmatrix}$$

- 행렬 $\mathbf{g}$를 이용하여 필터링을 한 번 수행함
- 그 결과를 다시 $\mathbf{g}$의 전치 행렬인 $\mathbf{g}^T$를 이용하여 필터링하는 것은 2차원 가우시안 필터 마스크로 한 번 필터링하는 것과 같은 결과를 얻을 수 있음
- 이 경우 픽셀 하나에 대해 필요한 곱셈 연산 횟수가 18번으로 감소하며 연산량이 크게 줄어듬

❖ 블러링: ② 가우시안 필터 (12/19)

- OpenCV에서 가우시안 필터링을 수행하려면 `GaussianBlur()` 함수를 사용함
- `GaussianBlur()` 함수 원형은 다음과 같음

```
dst = cv2.GaussianBlur(src, (ksize, ksize), sigmaX, sigmaY, borderType)
```

`src`	입력 영상. 다채널 영상은 각 채널별로 블러링을 수행합니다.
`ksize`	가우시안 커널 크기. `ksize.width`와 `ksize.height`는 0보다 큰 홀수이어야 합니다. (0, 0)을 지정하면 표준 편차 값에 의해 자동으로 결정됩니다.
`sigmaX`	x 방향으로의 가우시안 커널 표준 편차
`sigmaY`	y 방향으로의 가우시안 커널 표준 편차
`borderType`	가장자리 픽셀 확장 방식

❖ 블러링: ② 가우시안 필터 (13/19)

- 코드 7-3의 blurring_gaussian() 함수는 가우시안 표준 편차를 1부터 5까지 정수 단위로 증가시키면서 rose.bmp 장미 영상에 대해 가우시안 필터링을 수행함

코드 7-3 가우시안 필터링 (blurring.py)

```
1   import cv2
2
3   def blurring_gaussian():
4       src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)
5
6       if src is None:
7           print('Image load failed!')
8           return
9
10      cv2.imshow('src', src)
11
```

❖ 블러링: ② 가우시안 필터 (14/19)

코드 7-3 가우시안 필터링 (blurring.py)

```
12      for sigma in range(1, 6):
13          dst = cv2.GaussianBlur(src, (0, 0), sigma)
14
15          desc = "Gaussian: sigma = %d" % (sigma)
16          cv2.putText(dst, desc, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
17                      1.0, 255, 1, cv2.LINE_AA)
18
19          cv2.imshow('dst', dst)
20          cv2.waitKey()
21
22      cv2.destroyAllWindows()
23
24  if __name__ == '__main__':
25      blurring_gaussian()
```

cv2.putText(창 이름, 문자열, 좌표, 폰트, 폰트 스케일, 색상, 선 타입)

❖ 블러링: ② 가우시안 필터 (15/19)

- blurring.py 소스 코드 설명
  - 12~20행 `sigma` 값을 1부터 5까지 증가시키면서 가우시안 블러링을 수행하고 그 결과를 화면에 나타냅니다.
  - 13행 `src` 영상에 가우시안 표준 편차가 `sigma`인 가우시안 블러링을 수행하고 그 결과를 `dst`에 저장합니다.
  - 15~17행 사용한 가우시안 표준 편차(`sigma`) 값을 결과 영상 `dst` 위에 출력합니다.

❖ 블러링: ② 가우시안 필터 (16/19)

- 그림 7-11에서 src는 입력 영상인 rose.bmp 파일이고, dst는 GaussianBlur() 함수에 의해 생성된 블러링 결과 영상임
- 표준 편차 값이 커질수록 결과 영상이 더욱 부드럽게 변경되는 것을 확인할 수 있음

▼ **그림 7-11 가우시안 필터링 예제 실행 화면**

❖ 블러링: ② 가우시안 필터 (17/19)

- 실제로 `GaussianBlur()` 함수 내부에서 가우시안 필터링을 구현할 때에도 x축 방향과 y축 방향에 따라 1차원 가우시안 필터 마스크를 각각 생성하여 필터링을 수행함
- 이때 1차원 가우시안 필터 마스크를 생성하기 위해 OpenCV에서 제공하는 `getGaussianKernel()` 함수를 사용함

❖ 블러링: ② 가우시안 필터 (18/19)

- getGaussianKernel() 함수는 사용자가 지정한 표준 편차를 따르는 1차원 가우시안 필터 마스크 행렬을 생성하여 반환함
- getGaussianKernel() 함수 원형은 다음과 같음

cv2.getGaussianKernel(ksize, sigma, ktype)	
ksize	커널 크기. ksize는 0보다 큰 홀수이어야 합니다.
sigma	가우시안 표준 편차. 0 또는 음수를 지정하면 sigma=0.3*((ksize-1)*0.5-1)+0.8 형태로 sigma를 계산합니다.
ktype	필터의 타입
반환값	ksize×1 크기의 가우시안 필터 커널

❖ 블러링: ② 가우시안 필터 (19/19)

- `getGaussianKernel()` 함수는 표준 편차가 `sigma`인 1차원 가우시안 분포 함수로부터 `ksize`×1 크기의 필터 마스크 행렬을 생성하여 반환함
- `ksize`는 (8*`sigma`+1)보다 같거나 크게 지정하는 것이 좋음
- 이 행렬의 원소에 저장되는 값은 다음 수식을 따름

$$G_i = \alpha \cdot e^{-\frac{(i-(\text{ksize}-1)/2)^2}{2\cdot\text{sigma}^2}}$$

- 위 수식에서 $i = 0, \ldots, \text{ksize} - 1$의 범위를 가지며, α는 $\sum_i G_i = 1$이 되도록 만드는 상수임

7.3 샤프닝: 영상 날카롭게 하기

7.1 영상의 필터링
7.2 블러링: 영상 부드럽게 하기
7.4 잡음 제거 필터링

❖ 언샤프 마스크 필터 (1/11)

- **샤프닝**(**sharpening**)이란 영상을 **날카로운 느낌이 나도록 변경하는 필터링 기법임**
- 날카로운 느낌의 영상이란 초점이 잘 맞은 사진처럼 **객체의 윤곽이 뚜렷하게 구분되는 영상**을 의미함
- 이미 촬영된 사진을 초점이 잘 맞은 사진처럼 보이게끔 변경하려면 **영상 에지 근방에서 픽셀 값의 명암비가 커지도록 수정**해야 함

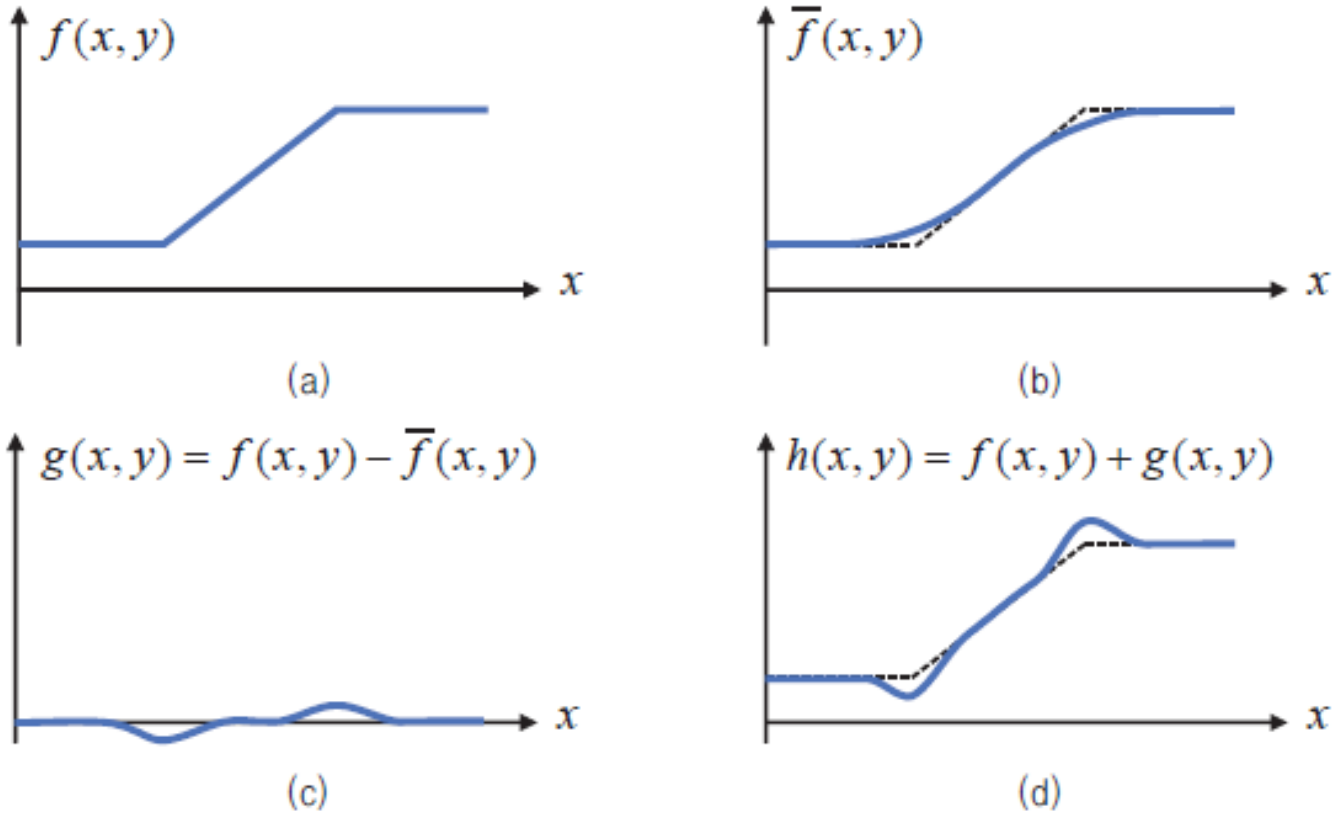
❖ 언샤프 마스크 필터 (2/11)

- 샤프닝 기법과 관련해서 흥미로운 사실은 샤프닝을 구현하기 위해 블러링된 영상을 사용한다는 점임
- 블러링이 적용되어 **부드러워진 영상을 활용하여 반대로 날카로운 영상을 생성**함
- 여기서 블러링이 적용된 영상, 즉 날카롭지 않은 영상을 언샤프(unsharp)하다고 말하기도 함
- 언샤프한 영상을 이용하여 역으로 날카로운 영상을 생성하는 필터를 **언샤프 마스크 필터**(**unsharp mask filter**)라고 함

❖ 언샤프 마스크 필터 (3/11)

- 그림 7-12에서 가로축은 픽셀 좌표의 이동을 나타내며, 세로축은 픽셀 값을 나타냄

▼ **그림 7-12 언샤프 마스크 필터의 동작 방식**



❖ 언샤프 마스크 필터 (4/11)

- 그림 7-12에서 $g(x,y)$는 입력 영상에서 블러링된 영상을 뺀 결과이므로 $g(x,y)$는 입력 영상에서 오직 날카로운 성분만을 가지고 있는 함수라고 할 수 있음
- 입력 $f(x,y)$에 $g(x,y)$를 더함으로써 날카로운 성분이 강조된 최종 영상 $h(x,y)$가 얻어지는 것으로 해석할 수 있음
- $f(x,y)$에 $g(x,y)$를 단순하게 더하는 것이 아니라 실수 가중치를 곱한 후 더하면 날카로운 정도를 사용자가 조절할 수 있음
- 샤프닝이 적용된 결과 영상 $h(x,y)$ 수식을 다음과 같이 수정할 수 있음

$$h(x,y) = f(x,y) + \alpha \cdot g(x,y)$$

❖ 언샤프 마스크 필터 (5/11)

- 앞 수식에서 α는 샤프닝 결과 영상의 날카로운 정도를 조절할 수 있는 파라미터임
- 즉, α에 1.0을 지정하면 날카로운 성분을 그대로 한 번 더하는 셈이고,
  α에 1보다 작은 값을 지정하면 조금 덜 날카로운 영상을 만들 수 있음
- 앞 수식에서 $g(x,y)$ 대신 $f(x,y)-\bar{f}(x,y)$ 수식을 대입하고 식을 정리하면 다음과 같음

$$\begin{aligned} h(x,y) &= f(x,y)+\alpha(f(x,y)-\bar{f}(x,y)) \\ &= (1+\alpha)f(x,y)-\alpha\cdot\bar{f}(x,y) \end{aligned}$$

❖ 언샤프 마스크 필터 (6/11)

- OpenCV는 언샤프 마스크 필터 함수를 따로 제공하지 않음
- 다만 앞 수식을 그대로 소스 코드 형태로 작성하면 어렵지 않게 샤프닝 결과 영상을 얻을 수 있음
- 이 수식에서 $\bar{f}(x,y)$는 입력 영상에 블러링이 적용된 영상이며, 이때 블러링 영상을 구하기 위해 평균값 필터를 사용해도 되고 가우시안 필터를 사용해도 됨
- 가우시안 필터로 $\bar{f}(x,y)$ 영상을 생성할 경우, 가우시안 분포의 표준 편차를 어떻게 지정하느냐가 샤프닝 결과에 영향을 줄 수 있음

❖ 언샤프 마스크 필터 (7/11)

- 코드 7-4의 unsharp_mask() 함수는 rose.bmp 장미 영상을 다양한 표준 편차 값으로 가우시안 필터를 적용하고, 블러링된 영상을 이용하여 샤프닝 결과 영상을 생성함

코드 7-4 언샤프 마스크 필터링 예제 코드 (sharpen.py)

```
1   import sys
2   import cv2
3
4   src = cv2.imread('rose.bmp', cv2.IMREAD_GRAYSCALE)
5
6   if src is None:
7       print('Image load failed!')
8       return
9
10  cv2.imshow('src', src)
11
```

❖ 언샤프 마스크 필터 (8/11)

코드 7-4 언샤프 마스크 필터링 예제 코드 (sharpen.py)

```
12  for sigma in range(1, 6):
13      blurred = cv2.GaussianBlur(src, (0, 0), sigma)
14
15      alpha = 1.0
16      dst = cv2.addWeighted(src, 1 + alpha, blurred, -alpha, 0.0)
17
18      desc = "sigma: %d" % sigma
19      cv2.putText(dst, desc, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
20                  1.0, 255, 1, cv2.LINE_AA)
21
22      cv2.imshow('dst', dst)
23      cv2.waitKey()
24
25  cv2.destroyAllWindows()
```

cv2.putText(창 이름, 문자열, 좌표, 폰트, 폰트 스케일, 색상, 선 타입)

❖ 언샤프 마스크 필터 (9/11)

- sharpen.py 소스 코드 설명
  - 12~16행 가우시안 필터의 표준 편차 `sigma` 값을 1부터 5까지 증가시키면서 언샤프 마스크 필터링을 수행합니다.
  - 13행 가우시안 필터를 이용한 블러링 영상을 blurred에 저장합니다.
  - 15~16행 언샤프 마스크 필터링을 수행합니다.
  - 18~20행 샤프닝 결과 영상 `dst`에 사용된 `sigma` 값을 출력합니다.

❖ 언샤프 마스크 필터 (10/11)

- src 창에 나타난 장미 영상은 입력 영상이고, dst 창에 나타난 영상은 다양한 `sigma` 값에 의해 생성된 언샤프 마스크 필터링 결과 영상임
- src 영상보다 dst 영상이 장미꽃 경계 부분이 좀 더 뚜렷하게 구분이 되는 것을 확인할 수 있음
- 다만 `sigma` 값이 커짐에 따라 다소 과장된 느낌의 샤프닝 결과 영상이 만들어질 수도 있으니 주의해야 함
- 코드 7-4에서는 날카로운 성분에 대한 가중치 `alpha` 값을 1.0으로 고정하여 사용하였지만, 소스 코드를 변경하여 다양한 `alpha` 값에 대해서도 샤프닝 결과를 확인해 보기 바람

❖ 언샤프 마스크 필터 (11/11)

▼ 그림 7-13 언샤프 마스크 필터링 예제 실행 결과

7.4 잡음 제거 필터링

7.1 영상의 필터링
7.2 블러링: 영상 부드럽게 하기
7.3 샤프닝: 영상 날카롭게 하기

❖ 영상과 잡음 모델 (1/9)

- 신호 처리 관점에서 **잡음**(**noise**)이란 원본 신호에 추가된 원치 않은 신호를 의미함
- 영상에서 잡음은 주로 영상을 획득하는 과정에서 발생함
- 디지털 카메라에서 사진을 촬영하는 경우에는 광학적 신호를 전기적 신호로 변환하는 **센서**(**sensor**)에서 주로 잡음이 추가됨
- 디지털 카메라에서 카메라 렌즈가 바라보는 장면을 원본 신호 $s(x, y)$라고 함
- 여기에 추가되는 잡음을 $n(x, y)$라고 표현한다면 실제로 카메라에서 획득되는 영상 $f(x, y)$는 보통 다음과 같이 표현함

$$f(x, y) = s(x, y) + n(x, y)$$

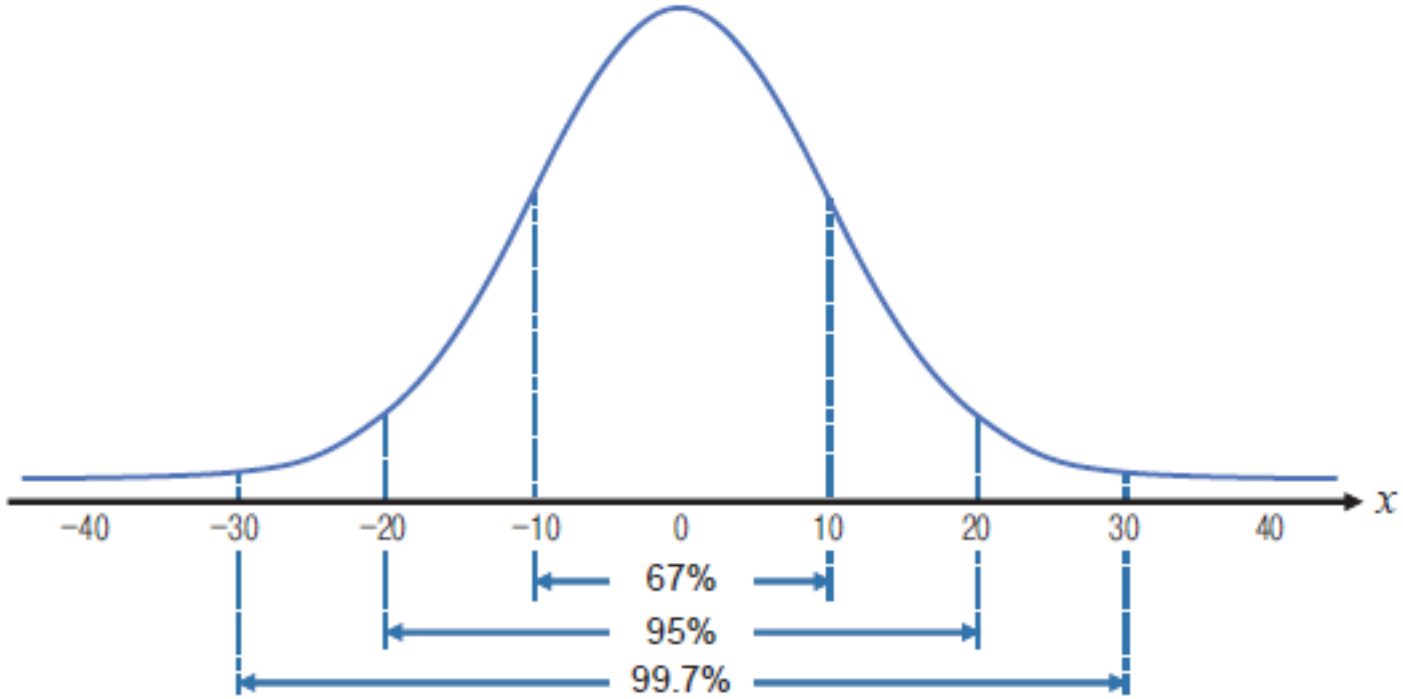
❖ 영상과 잡음 모델 (2/9)

- 잡음이 생성되는 방식을 **잡음 모델**(**noise model**)이라고 함
- 다양한 잡음 모델 중에서 가장 대표적인 잡음 모델은 **가우시안 잡음 모델**(**Gaussian noise model**)임
- 가우시안 잡음 모델은 보통 평균이 0인 가우시안 분포를 따르는 잡음을 의미함

❖ 영상과 잡음 모델 (3/9)

- 표준 편차가 작은 가우시안 잡음 모델일수록 잡음에 의한 픽셀 값 변화가 적다고 생각할 수 있음

▼ **그림 7-14 평균이 0이고 표준 편차가 10인 가우시안 분포 그래프**



❖ 영상과 잡음 모델 (4/9)

- OpenCV 함수를 이용하여 영상에 가우시안 모델을 따르는 잡음을 인위적으로 추가할 수 있음
- randn() 함수는 가우시안 잡음으로 구성된 행렬을 생성하여 반환합니다.
- randn() 함수 원형은 다음과 같음

cv2.randn(dst, mean, stddev)	
dst	가우시안 난수로 채워질 행렬. dst 행렬은 미리 할당되어 있어야 합니다.
mean	가우시안 분포 평균
stddev	가우시안 분포 표준 편차

❖ 영상과 잡음 모델 (5/9)

- randn() 함수에 전달되는 dst 영상은 미리 적절한 타입으로 생성되어 있어야 하며, randn() 함수에 의해 생성된 난수는 dst 행렬의 자료형에 맞게끔 포화 연산이 수행됨
- 평균이 0인 가우시안 잡음을 생성할 경우 양수와 음수가 섞여 있는 난수가 발생하므로 부호 있는 자료형 행렬을 사용해야 함

❖ 영상과 잡음 모델 (6/9)

- 코드 7-5의 noise_gaussian() 함수는 레나 영상에 평균이 0이고 표준 편차가 각각 10, 20, 30인 가우시안 잡음을 추가하여 화면에 나타냄

코드 7-5 가우시안 잡음 추가 예제 코드 (noise.py)

```
1   import numpy as np
2   import cv2
3   import random
4
5   def noise_gaussian():
6       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
7
8       if src is None:
9           print('Image load failed!')
10          return
11
12      cv2.imshow('src', src)
13
```

❖ 영상과 잡음 모델 (7/9)

코드 7-5 가우시안 잡음 추가 예제 코드 (noise.py)

```
14      for stddev in [10, 20, 30]:
15          noise = np.zeros(src.shape, np.int32)
16          cv2.randn(noise, 0, stddev)
17
18          dst = cv2.add(src, noise, dtype=cv2.CV_8UC1)
19
20          desc = 'stddev = %d' % stddev
21          cv2.putText(dst, desc, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
22                      1.0, 255, 1, cv2.LINE_AA)
23          cv2.imshow('dst', dst)
24          cv2.waitKey()
25
26      cv2.destroyAllWindows()
27
28  if __name__ == '__main__':
29      noise_gaussian()
```

cv2.putText(창 이름, 문자열, 좌표, 폰트, 폰트 스케일, 색상, 선 타입)

❖ 영상과 잡음 모델 (8/9)

- noise.py 소스 코드 설명

  - 6행 lenna.bmp 파일을 grayscale 형식으로 불러와 src에 저장합니다.
  - 14행 표준 편차 stddev 값이 10, 20, 30이 되도록 for 반복문을 수행합니다.
  - 15~16행 평균이 0이고 표준 편차가 stddev인 가우시안 잡음을 생성하여 noise 행렬에 저장합니다.
    이때 noise 행렬은 부호 있는 정수형(np.int32)을 사용하도록 미리 생성하여 randn() 함수에 전달합니다.
  - 18행 입력 영상 src에 가우시안 잡음 noise를 더하여 결과 영상 dst를 생성합니다.
    dst 영상의 깊이는 cv2.8UC1으로 설정합니다.

❖ 영상과 잡음 모델 (9/9)

- 원본 영상 src에 비해 가우시안 잡음이 추가된 결과 영상 dst가 거칠고 지저분해 보이는 것을 확인할 수 있음
- 특히 표준 편차 stddev 값이 증가함에 따라 잡음의 영향이 커지므로 결과 영상이 더욱 지저분해지는 것을 볼 수 있음

▼ **그림 7-15 가우시안 잡음 추가 예제 실행 결과**

❖ 잡음 제거: ① 양방향 필터 (1/12)

- 대부분의 영상에는 가우시안 잡음이 포함되어 있으며 많은 컴퓨터 비전 시스템이 가우시안 잡음을 제거하기 위해 가우시안 필터를 사용함
- 입력 영상에서 픽셀 값이 크게 변하지 않는 평탄한 영역에 가우시안 필터가 적용될 경우, 주변 픽셀 값이 부드럽게 블러링되면서 잡음의 영향도 크게 줄어듦
- 픽셀 값이 급격하게 변경되는 에지(edge) 근방에 동일한 가우시안 필터가 적용되면 잡음뿐만 아니라 에지 성분까지 함께 감소하게 됨
- 잡음이 줄어들면서 함께 에지도 무뎌지기 때문에 객체의 윤곽이 흐릿하게 바뀜

❖ 잡음 제거: ① 양방향 필터 (2/12)

- 이러한 단점을 보완하기 위해 많은 사람들이 에지 정보는 그대로 유지하면서 잡음만 제거하는 **에지 보전 잡음 제거 필터**(**edge-preserving noise removal filter**)에 대해 연구함
- 특히 1998년 토마시(C. Tomasi)가 제안한 **양방향 필터**(**bilateral filter**)는 에지 성분은 그대로 유지하면서 가우시안 잡음을 효과적으로 제거하는 알고리즘[Tomasi98]으로 알려져 있음
- 양방향 필터 기능은 OpenCV 라이브러리 초기 버전부터 포함되어 있어서 많은 사람들이 사용하고 있음

[Tomasi98] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, Jan. 1998, pp. 839-846.

❖ 잡음 제거: ① 양방향 필터 (3/12)

- 양방향 필터는 다음 공식을 사용하여 필터링을 수행함

$$g_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}\left(\|\mathbf{p}-\mathbf{q}\|\right) G_{\sigma_r}\left(|f_{\mathbf{p}} - f_{\mathbf{q}}|\right) f_{\mathbf{q}}$$

- 앞 수식에서 f는 입력 영상, g는 출력 영상, 그리고 $\mathbf{p}$와 $\mathbf{q}$는 픽셀의 좌표를 나타냄
- $f_{\mathbf{p}}$와 $f_{\mathbf{q}}$는 각각 $\mathbf{p}$ 점과 $\mathbf{q}$ 점에서의 입력 영상 픽셀 값이고, $g_{\mathbf{p}}$는 $\mathbf{p}$ 점에서의 출력 영상 픽셀 값임
- G_{σ_s}와 G_{σ_r}는 각각 표준 편차가 σ_s와 σ_r인 가우시안 분포 함수임
- S는 필터 크기를 나타내고, W_p는 양방향 필터 마스크 합이 1이 되도록 만드는 정규화 상수임
- 양방향 필터 수식은 매우 복잡해 보이지만 가만히 살펴보면 **두 개의 가우시안 함수 곱으로 구성된 필터임**

❖ 잡음 제거: ① 양방향 필터 (4/12)

- 먼저 $G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)$ 함수는 두 점 사이의 거리에 대한 가우시안 함수로서, 가우시안 필터와 완전히 같은 의미로 동작함
- 반면에 $G_{\sigma_r}(|f_\mathbf{p}-f_\mathbf{q}|)$ 함수는 두 점의 픽셀 값 차이에 의한 가우시안 함수임
- $G_{\sigma_r}(|f_\mathbf{p}-f_\mathbf{q}|)$ 함수는 두 점의 픽셀 밝기 값의 차이가 적은 평탄한 영역에서는 큰 가중치를 갖게 만듦
- 반면에 에지를 사이에 두고 있는 두 픽셀에 대해서는 $|f_\mathbf{p}-f_\mathbf{q}|$ 값이 크게 나타나므로 상대적으로 $G_{\sigma_r}(|f_\mathbf{p}-f_\mathbf{q}|)$는 거의 0에 가까운 값이 됨

❖ 잡음 제거: ① 양방향 필터 (5/12)

- 양방향 필터 수식이 픽셀 값의 차이에 의존적이기 때문에 양방향 필터 마스크는 영상의 모든 픽셀에서 서로 다른 형태를 갖게 됨
- 모든 픽셀 위치에서 주변 픽셀과의 밝기 차이에 의한 고유의 필터 마스크 행렬을 만들어서 마스크 연산을 수행해야 함
- 일반적인 가우시안 블러링이 모든 위치에서 일정한 가우시안 마스크 행렬을 사용하는 것과 차이가 있음
- **양방향 필터는 가우시안 블러링보다 훨씬 많은 연산량을 필요로 함**

❖ 잡음 제거: ① 양방향 필터 (6/12)

- OpenCV에서는 `bilateralFilter()` 함수를 이용하여 양방향 필터를 수행할 수 있음
- `bilateralFilter()` 함수 원형은 다음과 같음

```
dst = cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace, borderType)
```

src	입력 영상. 8비트 또는 실수형, 1채널 또는 3채널 영상
dst	출력 영상. src와 같은 크기, 같은 타입을 갖습니다.
d	필터링에 사용할 이웃 픽셀과의 거리(지름). 양수가 아닌 값(예를 들어 -1)을 지정하면 sigmaSpace로부터 자동 계산됩니다.
sigmaColor	색 공간에서의 가우시안 필터 표준 편차
sigmaSpace	좌표 공간에서의 가우시안 필터 표준 편차
borderType	가장자리 픽셀 확장 방식

❖ 잡음 제거: ① 양방향 필터 (7/12)

- `bilateralFilter()` 함수에서 `sigmaSpace` 값은 일반적인 가우시안 필터링에서 사용하는 표준 편차와 같은 개념
- 즉, 값이 클수록 더 많은 주변 픽셀을 고려하여 블러링을 수행함
- `sigmaColor` 값은 주변 픽셀과의 밝기 차이에 관한 표준 편차임
- `sigmaColor` 값을 작게 지정할 경우, 픽셀 값 차이가 큰 주변 픽셀과는 블러링이 적용되지 않음
- 반면에 `sigmaColor` 값을 크게 지정하면 픽셀 값 차이가 조금 크더라도 블러링이 적용됨
- 즉, `sigmaColor` 값을 이용하여 어느 정도 밝기 차를 갖는 에지를 보존할 것인지를 조정할 수 있음

❖ 잡음 제거: ① 양방향 필터 (8/12)

- 코드 7-6의 `filter_bilateral()` 함수는 레나 영상에 가우시안 잡음을 추가하고, 가우시안 블러와 양방향 필터를 각각 적용하여 그 결과를 비교함

코드 7-6 양방향 필터링 예제 코드 (noise.py)

```
1   import numpy as np
2   import cv2
3   import random
4
5   def filter_bilateral():
6       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
7
8       if src is None:
9           print('Image load failed!')
10          return
11
```

❖ 잡음 제거: ① 양방향 필터 (9/12)

코드 7-6 양방향 필터링 예제 코드 (noise.py)

```
12      noise = np.zeros(src.shape, np.int32)
13      cv2.randn(noise, 0, 5)
14      cv2.add(src, noise, src, dtype=cv2.CV_8UC1)
15
16      dst1 = cv2.GaussianBlur(src, (0, 0), 5)
17      dst2 = cv2.bilateralFilter(src, -1, 10, 5)
18
19      cv2.imshow('src', src)
20      cv2.imshow('dst1', dst1)
21      cv2.imshow('dst2', dst2)
22      cv2.waitKey()
23      cv2.destroyAllWindows()
24
25  if __name__ == '__main__':
26      filter_bilateral()
```

❖ 잡음 제거: ① 양방향 필터 (10/12)

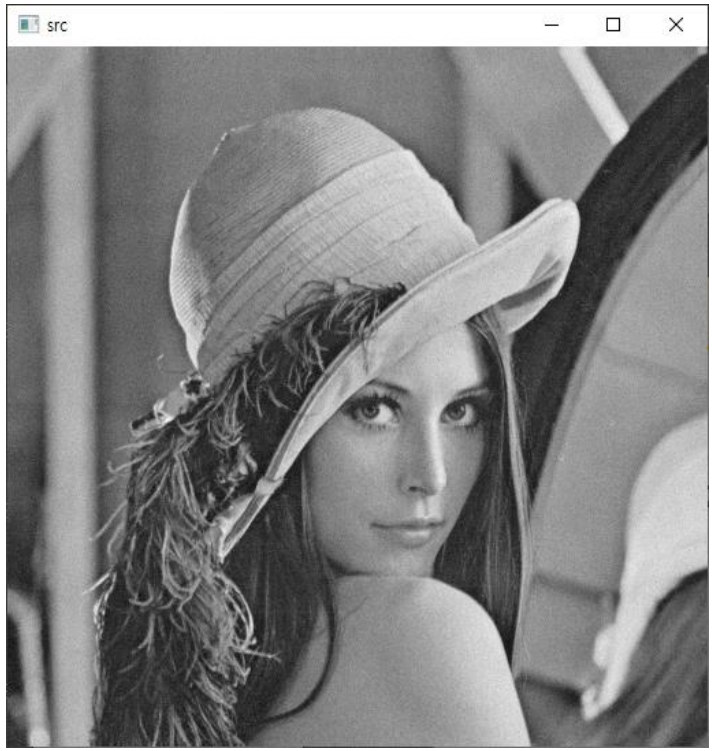
- noise.py 소스 코드 설명
  - 12~14행 grayscale 레나 영상 `src`에 평균이 0이고 표준 편차가 5인 가우시안 잡음을 추가합니다.
  - 16행 표준 편차가 5인 가우시안 필터링을 수행하여 `dst1`에 저장합니다.
  - 17행 색 공간의 표준 편차는 10, 좌표 공간의 표준 편차는 5를 사용하는 양방향 필터링을 수행하여 `dst2`에 저장합니다.
  - 19~21행 `src`, `dst1`, `dst2` 영상을 모두 화면에 출력합니다.

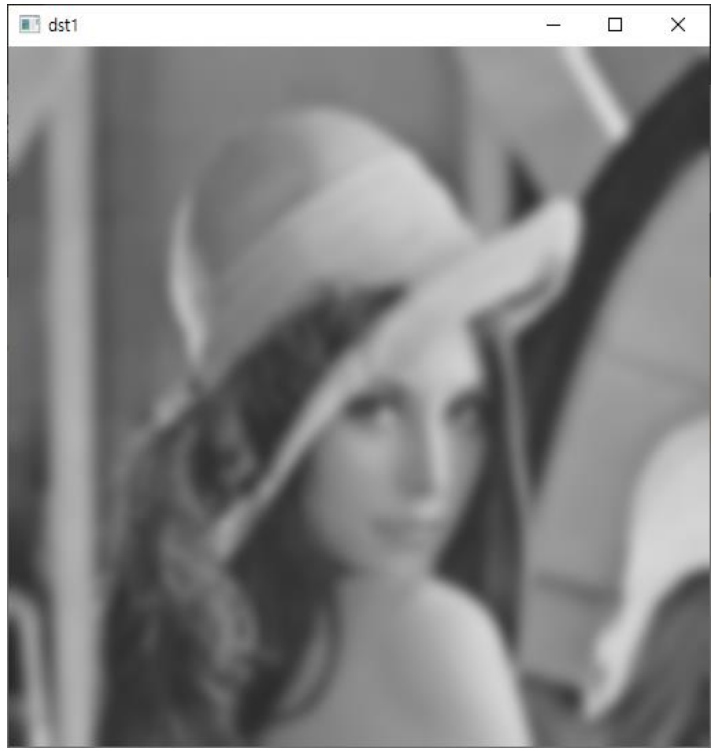
❖ 잡음 제거: ① 양방향 필터 (11/12)

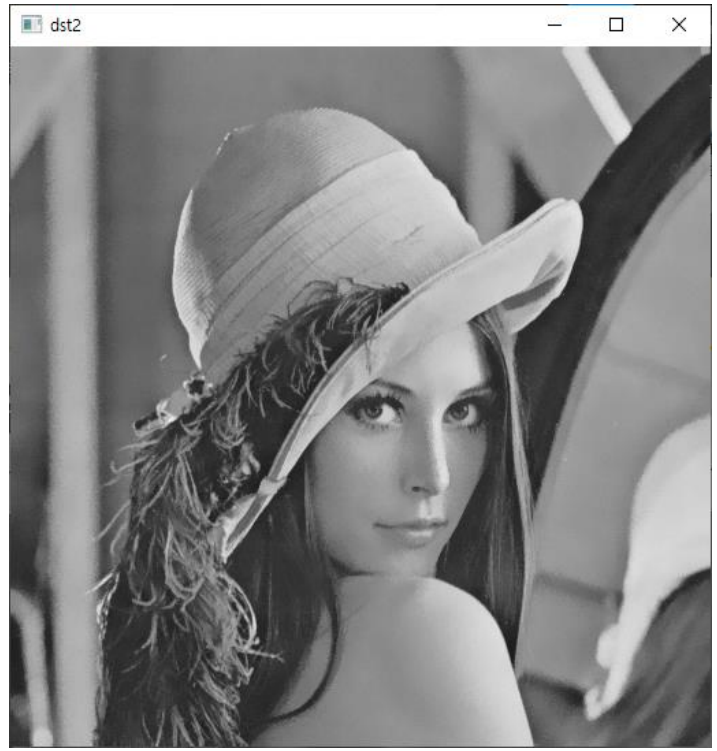
- src 창의 영상은 lenna.bmp 영상에 평균이 0이고 표준 편차가 5인 가우시안 잡음이 추가된 영상임
- 이 영상에 대해 표준 편차가 5인 가우시안 필터링을 수행한 결과가 dst1 영상임
- 입력 영상 src에 비해 지글거리는 잡음의 영향은 크게 줄었지만, 머리카락, 모자, 배경 사물의 경계 부분이 함께 블러링되어 흐릿하게 변경됨
- 반면에 양방향 필터가 적용된 src2 영상은 머리카락, 모자, 배경 사물의 경계는 그대로 유지됨
- 평탄한 영역의 잡음은 크게 줄어들어 눈으로 보기에 매우 깔끔한 느낌을 주는 것을 확인할 수 있음

❖ 잡음 제거: ① 양방향 필터 (12/12)

▼ 그림 7-16 양방향 필터링 예제 코드 실행 결과







❖ 잡음 제거: ② 미디언 필터 (1/8)

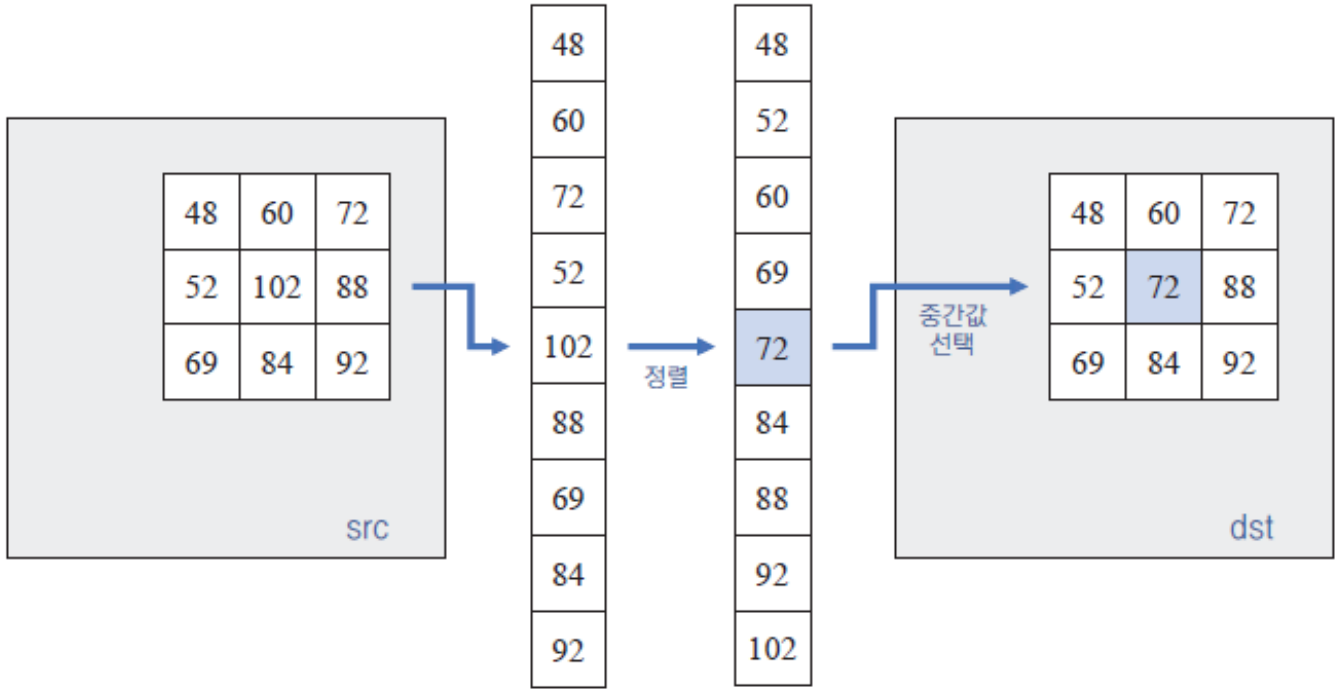
- **미디언 필터**(**median filter**)는 입력 영상에서 자기 자신 픽셀과 주변 픽셀 값 중에서 중간값(median)을 선택하여 결과 영상 픽셀 값으로 설정하는 필터링 기법임
- 미디언 필터는 마스크 행렬과 입력 영상 픽셀 값을 서로 곱한 후 모두 더하는 형태의 연산을 사용하지 않음
- 미디언 필터는 주변 픽셀 값들의 중간값을 선택하기 위해 내부에서 픽셀 값 정렬 과정이 사용됨
- 미디언 필터는 특히 잡음 픽셀 값이 주변 픽셀 값과 큰 차이가 있는 경우에 효과적으로 동작함

❖ 잡음 제거: ② 미디언 필터 (2/8)

- 영상에 추가되는 잡음 중에 소금&후추 잡음(salt & pepper noise)은 픽셀 값이 일정 확률로 0 또는255로 변경되는 형태의 잡음임
- '소금&후추'라는 다소 재미있는 이름이 붙은 이유는 잡음이 마치 소금과 후추처럼 흰색 또는 검은색으로 구성되기 때문임
- 소금&후추 잡음이 추가된 영상에 미디언 필터를 적용하면 대부분 소금&후추 잡음이 아닌 원본 영상에 존재하는 픽셀 값이 중간값으로 선택되기 때문에 잡음은 효과적으로 제거됨

❖ 잡음 제거: ② 미디언 필터 (3/8)

- 그림 7-17에서 가장 왼쪽 그림은 입력 영상 특정 위치에서의 3×3 주변 픽셀 값 배열을 나타냄
- 이 영역의 픽셀 값을 일렬로 늘여 세운 후 픽셀 값 크기 순으로 정렬함
- 정렬된 데이터에서 중앙에 있는 픽셀 값인 72를 선택하여, 결과 영상의 픽셀 값으로 설정함
- 이와 같은 과정을 영상 전체 픽셀에 대하여 수행하면 미디언 필터 결과 영상이 만들어짐



◀ 그림 7-17 미디언 필터링 수행 과정

❖ 잡음 제거: ② 미디언 필터 (4/8)

- OpenCV에서는 `medianBlur()` 함수를 이용하여 미디언 필터링을 수행할 수 있음
- `medianBlur()` 함수 원형은 다음과 같음

```
dst = cv2.medianBlur(src, ksize)
```

`src`	입력 영상. 1, 3, 4채널 영상.
`dst`	출력 영상. `src`와 같은 크기, 같은 타입을 갖습니다.
`ksize`	필터 크기. 3보다 같거나 큰 홀수를 지정합니다.

- `medianBlur()` 함수는 `ksize×ksize` 필터 크기를 이용하여 미디언 필터링을 수행함
- 다채널 영상인 경우 각 채널별로 필터링을 수행함
- `medianBlur()` 함수는 내부적으로 BORDER_REPLICATE 방식으로 가장자리 외곽 픽셀 값을 설정하여 필터링을 수행함

❖ 잡음 제거: ② 미디언 필터 (5/8)

- 코드 7-7의 `filter_median()` 함수는 입력 영상 전체 크기의 10%에 해당하는 픽셀에 소금&후추 잡음을 추가하고, 가우시안 필터와 미디언 필터를 수행한 결과 영상을 화면에 출력함

코드 7-7 미디언 필터링 예제 코드 (noise.py)

```
1   import numpy as np
2   import sys
3   import cv2
4
5   def filter_median():
6       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
7
8       if src is None:
9           print('Image load failed!')
10          return
11
```

❖ 잡음 제거: ② 미디언 필터 (6/8)

코드 7-7 미디언 필터링 예제 코드 (noise.py)

```
12      for j in range(0, int(src.size / 10)):
13          x = random.randint(0, src.shape[1] - 1)
14          y = random.randint(0, src.shape[0] - 1)
15          src[x, y] = (j % 2) * 255
16
17      dst1 = cv2.GaussianBlur(src, (0, 0), 1)
18      dst2 = cv2.medianBlur(src, 3)
19
20      cv2.imshow('src', src)
21      cv2.imshow('dst1', dst1)
22      cv2.imshow('dst2', dst2)
23      cv2.waitKey()
24      cv2.destroyAllWindows()
25
26  if __name__ == '__main__':
27      filter_median()
```

❖ 잡음 제거: ② 미디언 필터 (7/8)

- noise.py 소스 코드 파일
  - 12~15행 `src` 영상에서 10%에 해당하는 픽셀 값을 0 또는 255로 설정합니다.
  - 17행 표준 편차가 1인 가우시안 필터링을 수행하여 `dst1`에 저장합니다.
  - 18행 크기가 3인 미디언 필터를 실행하여 `dst2`에 저장합니다.
  - 20~22행 `src`, `dst1`, `dst2` 영상을 모두 화면에 출력합니다.

❖ 잡음 제거: ② 미디언 필터 (8/8)

- src 영상은 레나 영상에 10%의 확률로 소금&후추 잡음이 추가된 영상임
- 이 영상에 대해 가우시안 필터를 적용한 결과가 dst1 영상임
- 소금&후추 잡음에 대해서는 가우시안 블러링을 적용하여도 여전히 영상이 지저분하게 보이는 것을 확인할 수 있음
- 반면에 미디언 필터를 적용한 dst2 영상에서는 잡음에 의해 추가된 흰색 또는 검은색 픽셀이 효과적으로 제거된 것을 확인할 수 있음

▼ **그림 7-18 미디언 필터링 실행 화면**

THANK YOU!
Q & A

- Name: 권범
- Office: 동양미래대학교 2호관 704호 (02-2610-5238)
- E-mail: bkwon@dongyang.ac.kr
- Homepage: https://sites.google.com/view/beomkwon/home