Chapter

7

영역처리(1)

1

컨볼루션

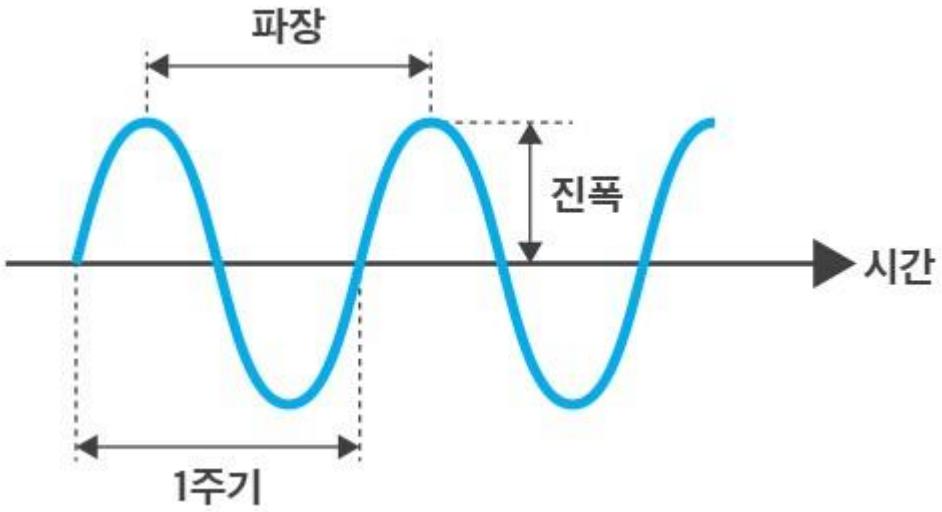
공간영역과 주파수영역

❖ 시간 영역(time domain)

- 신호를 시간에 의해 다루는 영역

❖ 공간 영역(spatial domain)

- x, y축의 2차원 공간
- 화소값으로 영상(이미지)를 표현



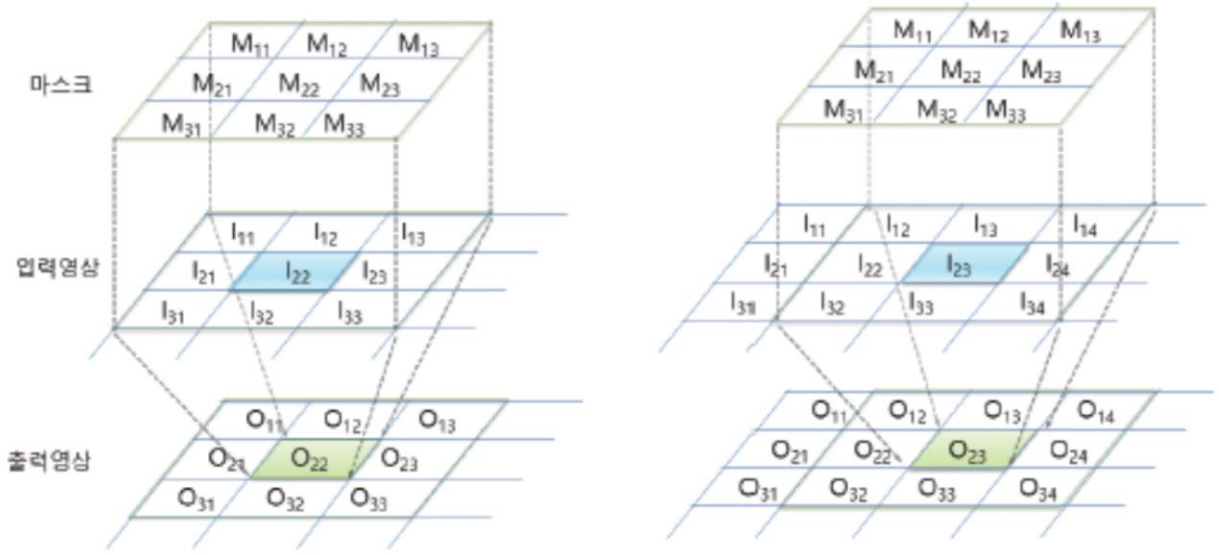
❖ 주파수 영역(frequency domain)

- 주파수(frequency)
  - 진동이나 파동 현상에서 단위시간(보통 1초) 동안에 똑같은 상태가 반복되는 횟수
  - 단위로 헤르츠(Hz)를 사용
- 신호에 포함된 주파수 성분(진폭,위상)에 의해 다루는 변환영역

영역처리와 회선(convolution)

❖ 마스크(mask)라 불리는 규정된 영역을 기반으로 연산 수행

- 커널(kernel), 윈도우(window), 필터(filter)



$$O_{22} = \sum\sum M_{ij} \cdot I_{ij}$$
$$= M_{11} \cdot I_{11} + M_{12} \cdot I_{12} + M_{13} \cdot I_{13}$$
$$+ M_{21} \cdot I_{21} + M_{22} \cdot I_{22} + M_{23} \cdot I_{23}$$
$$+ M_{31} \cdot I_{31} + M_{32} \cdot I_{32} + M_{33} \cdot I_{33}$$

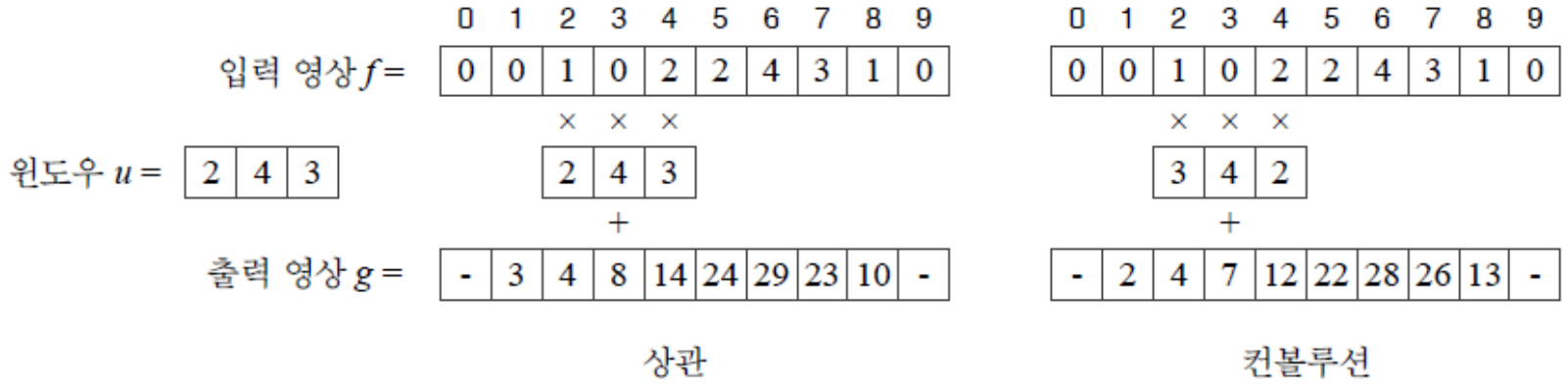
$$O_{23} = \sum\sum M_{ij} \cdot I_{ij}$$
$$= M_{11} \cdot I_{12} + M_{12} \cdot I_{13} + M_{13} \cdot I_{14}$$
$$+ M_{21} \cdot I_{22} + M_{22} \cdot I_{23} + M_{23} \cdot I_{24}$$
$$+ M_{31} \cdot I_{32} + M_{32} \cdot I_{33} + M_{33} \cdot I_{34}$$

〈그림 7.1.1〉 회선의 과정에 대한 이해

1차원 상관과 회선

❖ 상관(correlation)

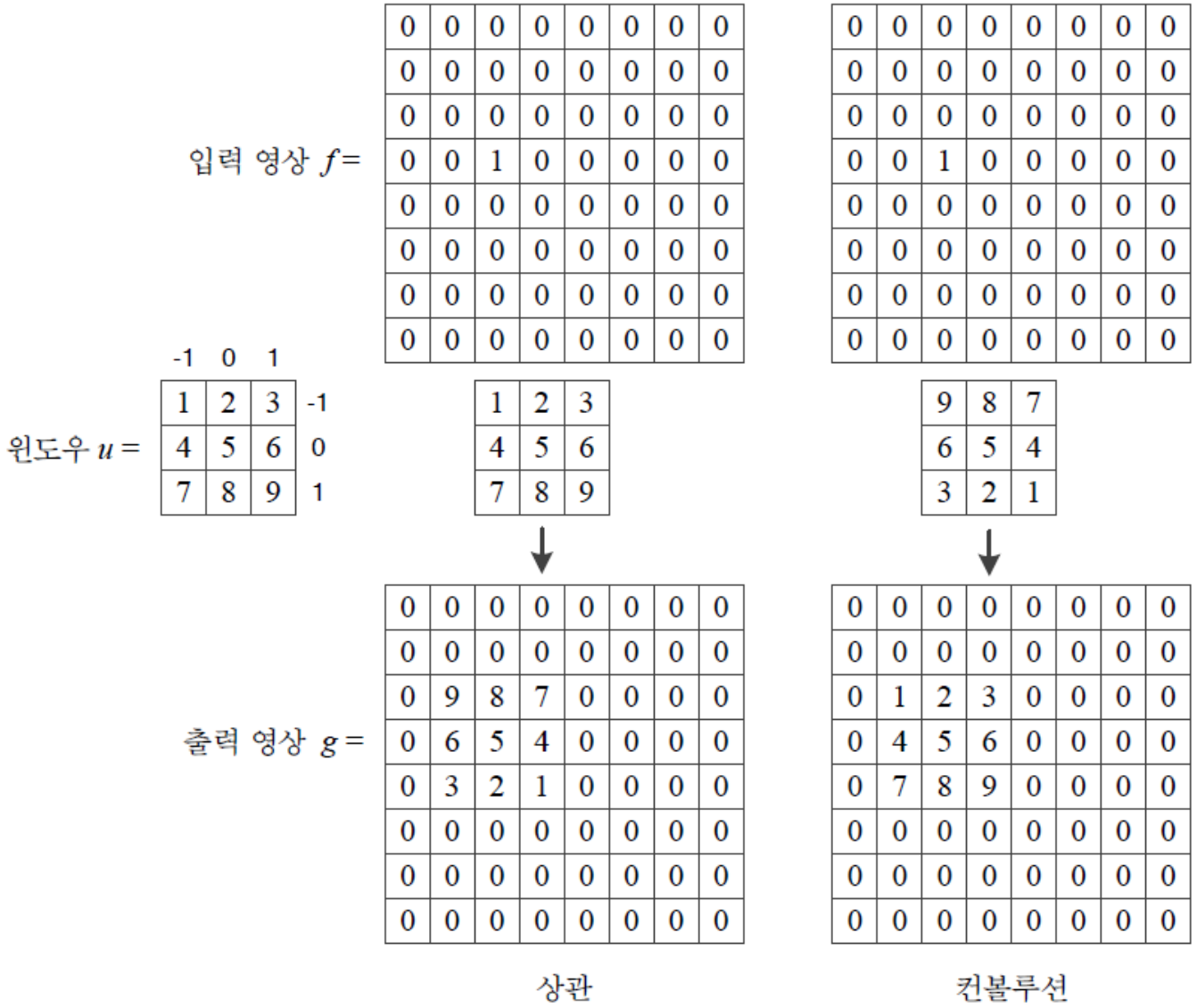
- 원시적인 매칭 연산 (물체를 윈도우 형태로 표현하고 물체를 검출)
- 아래 예에서는 최대값 29를 갖는 위치 6에서 물체가 검출됨



❖ 회선(convolution)

- 윈도우를 뒤집은 후 상관 적용

2차원 상관 및 회선



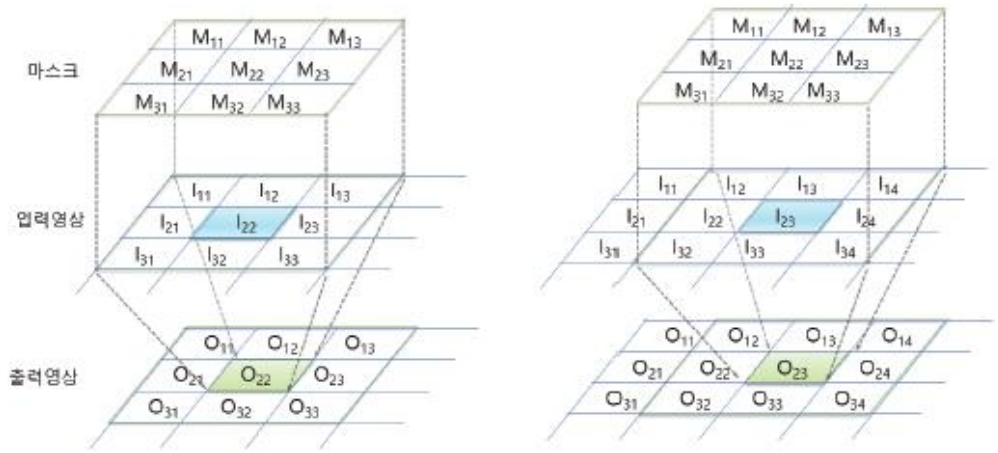
필터(마크스)

❖ 필터(마스크) 설계

- 가로 세로의 크기가 같고 홀수여야 한다.
- 마스크 가중치의 합은 1이어야 한다.
  - low-pass / median / high-pass filter

❖ 회선(convolution)의 적용

- 좌측상단픽셀로 부터 오른쪽 아래방향으로 진행
- 이미지의 경계부분 처리 방법(예)
  - 바깥쪽에 모두 0을 붙여 처리
  - 2번째줄 2번째 칸부터 시작하여 마지막 2번째 줄 2번째 칸에서 끝나도록 처리

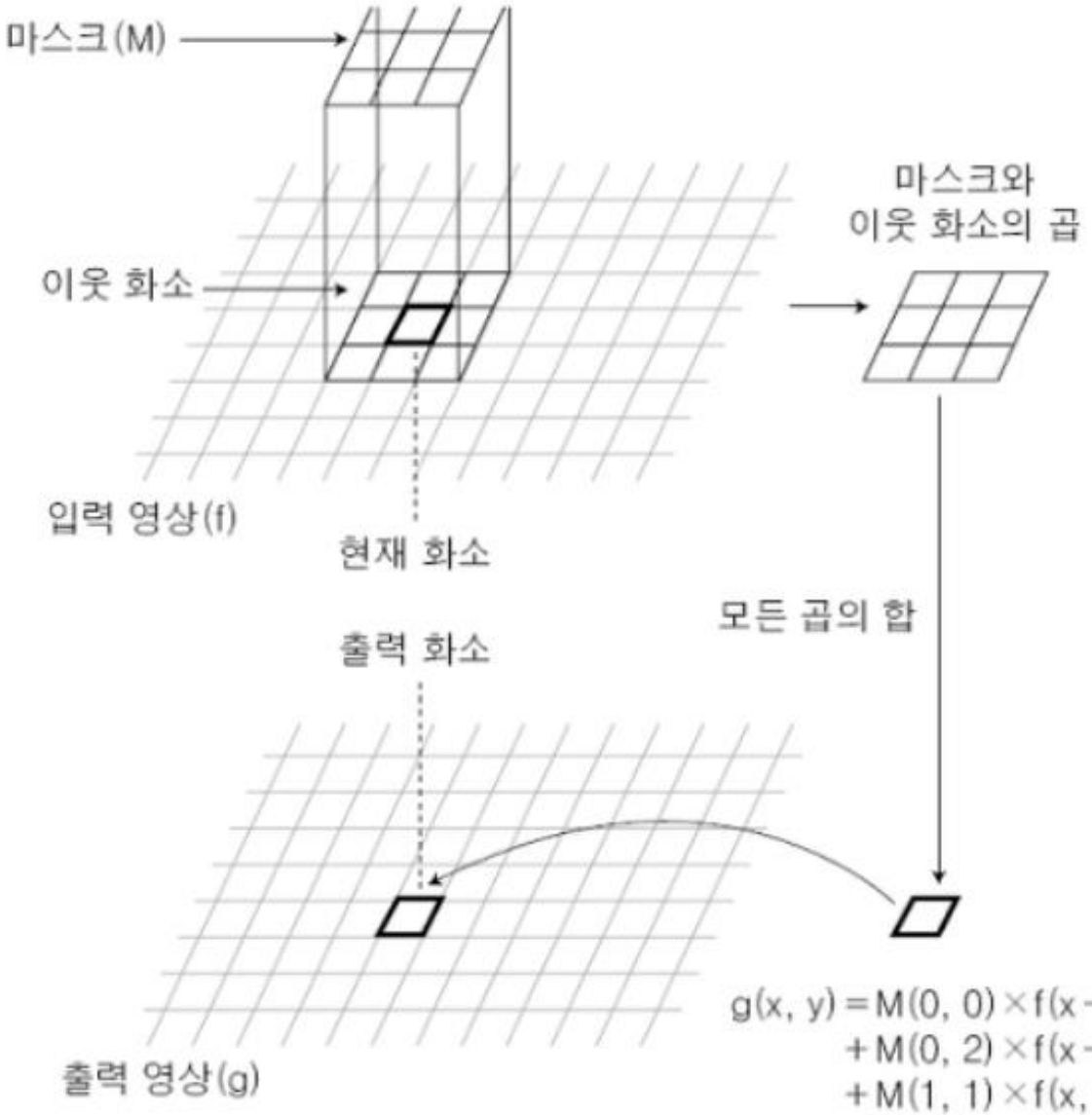


$$O_{22} = \sum\sum M_{ij} \cdot I_{ij}$$
$$= M_{11} \cdot I_{11} + M_{12} \cdot I_{12} + M_{13} \cdot I_{13}$$
$$+ M_{21} \cdot I_{21} + M_{22} \cdot I_{22} + M_{23} \cdot I_{23}$$
$$+ M_{31} \cdot I_{31} + M_{32} \cdot I_{32} + M_{33} \cdot I_{33}$$

$$O_{23} = \sum\sum M_{ij} \cdot I_{ij}$$
$$= M_{11} \cdot I_{12} + M_{12} \cdot I_{13} + M_{13} \cdot I_{14}$$
$$+ M_{21} \cdot I_{22} + M_{22} \cdot I_{23} + M_{23} \cdot I_{24}$$
$$+ M_{31} \cdot I_{32} + M_{32} \cdot I_{33} + M_{33} \cdot I_{34}$$

〈그림 7.1.1〉 회선의 과정에 대한 이해

출처: https://deep-learning-study.tistory.com/142



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

필터(마스크)의 종류

❖ low-pass filter

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

❖ median filter

- 중심픽셀과 주변픽셀의 중간값으로 중심픽셀값을 대체하는 필터
- 잡음제거에 효과가 있음

❖ high-pass filter

0	−1	0
−1	5	−1
0	−1	0

−1	−1	−1
−1	9	−1
−1	−1	−1

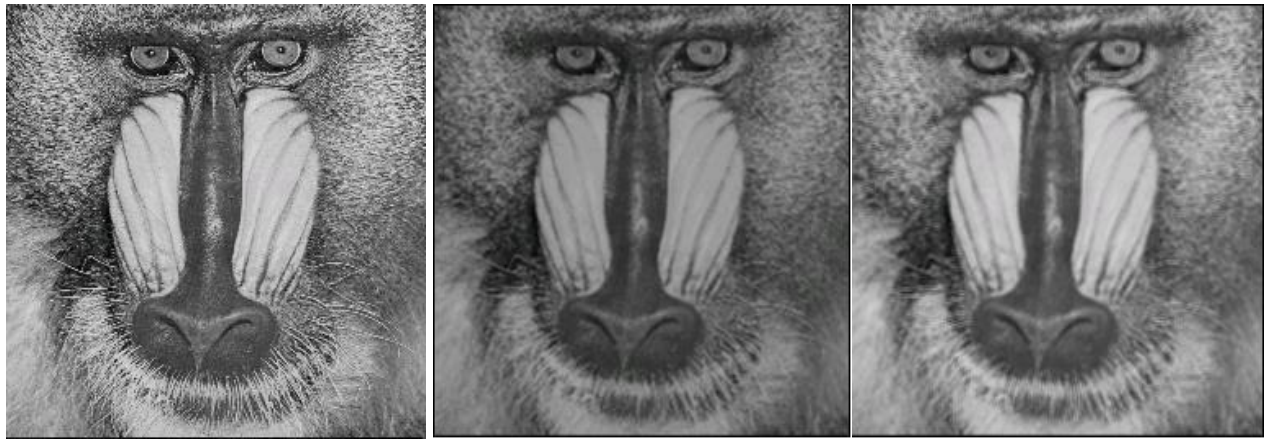
1	−2	1
−2	5	−2
1	−2	1

❖ embossing

−1	0	0
0	0	0
0	0	1

0	−1	0
0	0	0
0	1	0

0	0	−1
0	0	0
1	0	0



(a)원 이미지 (b) 박스 평활화 (c)가우시안 평활화

Low-pass filter 마스크를 적용한 예

박스

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

가우시안

.0000	.0000	.0002	.0000	.0000
.0000	.0113	.0837	.0113	.0000
.0002	.0837	.6187	.0837	.0002
.0000	.0113	.0837	.0113	.0000
.0000	.0000	.0002	.0000	.0000

샤프닝

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

수평 에지

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

수직 에지

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

모션

.0304	.0501	0	0	0
.0501	.1771	.0519	0	0
0	.0519	.1771	.0519	0
0	0	.0519	.1771	.0501
0	0	0	.0501	.0304

(a) 원래 영상과 여러 가지 마스크들

> 박스

> 가우시안

> 샤프닝

> 수평 에지

> 수직 에지

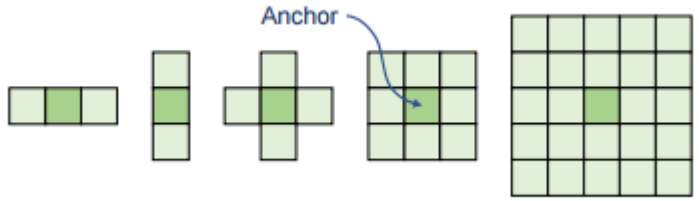
> 모션

(b) 다양한 마스크로 컨볼루션한 영상들

cv2.filter2D()

❖ **cv.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) -> dst**

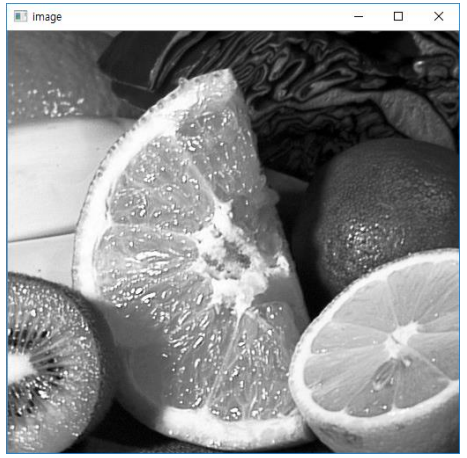
- **src** : 입력 영상
- **ddepth** : 출력 영상 데이터 타입
  - cv2.CV_8U, cv2.CV_32F, cv2.CV_64F
  - -1을 지정하면 src와 같은 타입의 dst 영상을 생성
- **kernel**: 필터 마스크 행렬. 실수형.
- **dst**: 출력 영상
- anchor: 고정점 위치
  - (-1, -1)이면 필터 중앙을 고정점으로 사용
- delta: 추가적으로 더할 값
- borderType: 가장자리 픽셀 확장 방식



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

cv2.filter2D()

```
import cv2
import numpy as np
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

    # high-pass filter
    kernel = np.array([[-1, -1, -1],\
                       [-1,  9, -1],\
                       [-1, -1, -1]])
    result = cv2.filter2D(img, -1, kernel)

    cv2.imshow('image', img)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```



2

블러링(blurring)
샤프닝(sharpening)

블러링

❖ 블러링 현상

- 디지털카메라로 사진 찍을 때, 초점이 맞지 않으면 → 사진 흐려짐

→ 이러한 현상을 이용해서 영상의 디테일한 부분을 제거하는 아웃 포커싱 (out focusing) 기법

  - 포토샵을 이용한 '뽀샵'
  - 사진 편집앱의 '뽀샤시' 기능

❖ 블러링(blurring)

- 영상에서 화소값이 급격하게 변하는 부분들을 감소시켜 점진적으로 변하게 함으로써 영상이 전체적으로 부드러운 느낌이 나게 하는 기술

❖ 화소값이 급격이 변화하는 것을 점진적으로 변하게 하는 방법

- → 블러링 마스크로 회선 수행

❖ 블러링 마스크

$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$
$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$	$\frac{1}{25}$

〈그림 7.1.2〉 블러링 마스크의 예

공통부분의 합
$= 60 + 90 + 90 + 200 + 100 + 100$

O_{23}=(50+50+200 + 공통부분의 합) / 9

50	60	90	50	100
100	90	200	50	30
100	100	100	200	100
100	100	150	150	50
30	90	80	70	160

입력 영상

75	98	90	87	58
83	99	104	102	88
98	116	127	114	97
87	94	116	[illegible]	122
80	92	107	110	[illegible]

출력 영상

O_{22}=(50+100+100 + 공통부분의 합) / 9

두화소 차이 줄어듬

예제 7.1.1 회선이용 블러링 – 01.bluring.py

```
01  import numpy as np, cv2
02
03  ## 회선 수행 함수 – 행렬 처리 방식(속도 면에서 유리)
04  def filter(image, mask):
05      rows, cols = image.shape[:2]
06      dst = np.zeros((rows, cols), np.float32)          # 회선 결과 저장 행렬
07      ycenter, xcenter = rows//2, cols//2               # 마스크 중심 좌표
08
09      for i in range(ycenter, rows - ycenter):          # 입력 행렬 반복 순회
10          for j in range(xcenter, cols - xcenter):
11              y1, y2 = i - ycenter, i + ycenter + 1     # 관심 영역 높이 범위
12              x1, x2 = j - xcenter, j + xcenter + 1     # 관심 영역 너비 범위
13              roi = image[y1:y2, x1:x2].astype('float32')      # 관심 영역 형변환
14              tmp = cv2.multiply(roi, mask)             # 회선 적용–원소간 곱셈
15              dst[i, j] = cv2.sumElems(tmp)[0]          # 출력 화소 저장
16      return dst                                        # 자료형 변환하여 반환
17
```

사프닝

❖ 샤프닝(sharpening)

- 출력화소에서 이웃 화소끼리 차이를 크게 해서 날카로운 느낌이 나게 만드는 것
- 영상의 세세한 부분을 강조할 수 있으며, 경계 부분에서 명암대비가 증가되는 효과

❖ 사프닝 마스크

- 마스크 원소들의 값 차이가 커지도록 구성
- 마스크 원소 전체합이 1이 되어야 입력영상 밝기가 손실 없이 출력영상 밝기로 유지

0	−1	0
−1	5	−1
0	−1	0

−1	−1	−1
−1	9	−1
−1	−1	−1

1	−2	1
−2	5	−2
1	−2	1

〈그림 7.1.4〉 샤프닝 마스크의 예

예제 7.1.2 회선이용 샤프닝 – 02.sharpening.py

```
01  import numpy as np, cv2
02  from Common.filters import filter                    # filters 모듈의 filter() 함수 임포트
03
04  image = cv2.imread("images/filter_sharpen.jpg", cv2.IMREAD_GRAYSCALE)
05  if image is None: raise Exception("영상파일 읽기 오류")
06
07  ## 샤프닝 마스크 원소 지정
08  data1 = [ 0, -1, 0,                                  # 1차원 리스트
09            -1, 5, -1,
10            0, -1, 0]
11  data2 = [[ -1, -1, -1],                              # 2차원 리스트
12           [-1, 9, -1],
13           [-1, -1, -1]]
14  mask1 = np.array(data1, np.float32).reshape(3, 3)    # ndarray 객체 생성 및 형태 변경
15  mask2 = np.array(data2, np.float32)
16
17  sharpen1 = filter(image, mask1)                      # 회선 수행 – 저자 구현 함수
18  sharpen2 = filter(image, mask2)
19  sharpen1 = cv2.convertScaleAbs(sharpen1)             # 윈도우 표시 위한 형변환
20  sharpen2 = cv2.convertScaleAbs(sharpen2)
21
22  cv2.imshow("image", image)                           # 결과 행렬을 윈도우에 표시
23  cv2.imshow("sharpen1", sharpen1)
24  cv2.imshow("sharpen2", sharpen2)
25  cv2.waitKey(0)
```

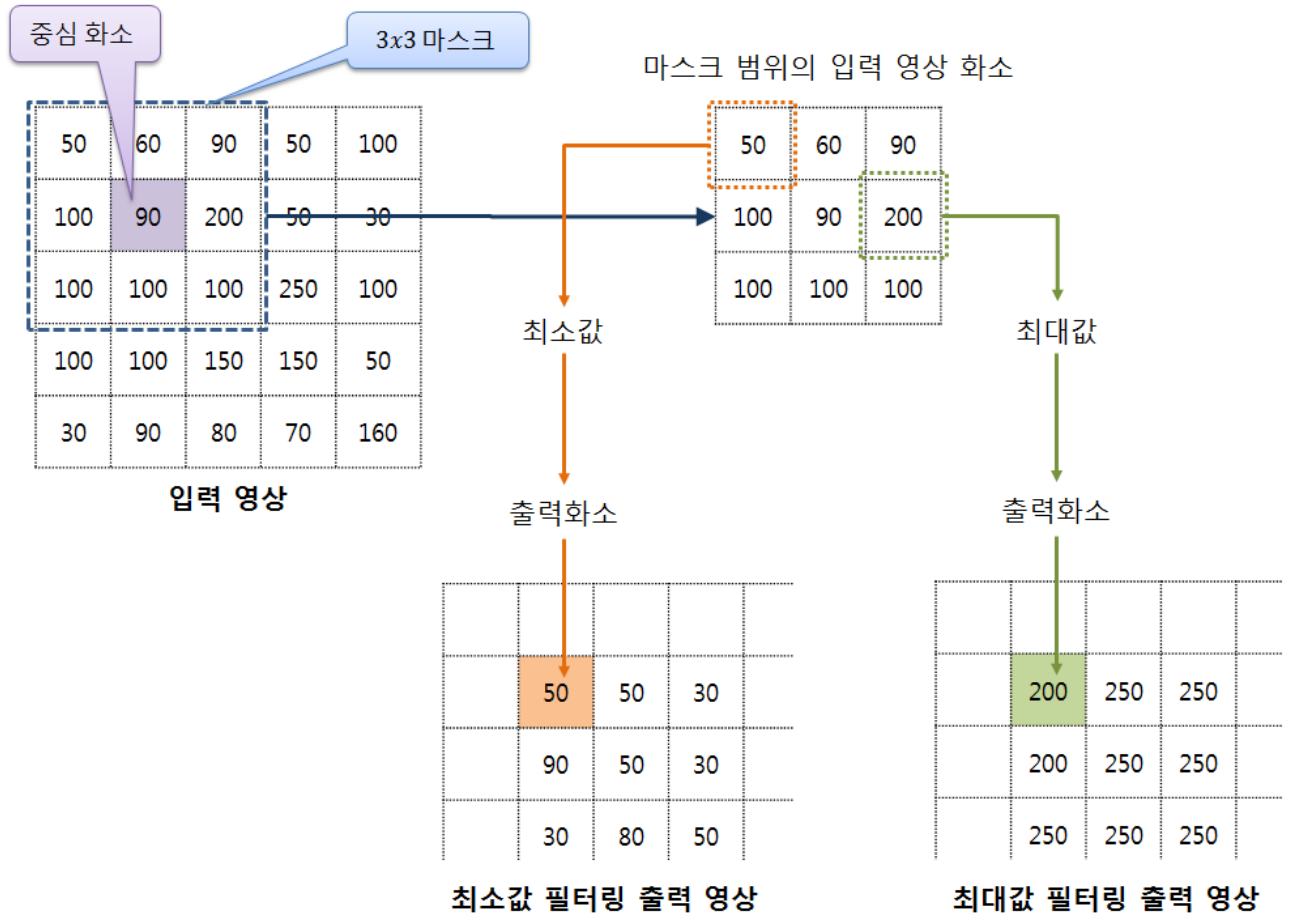
너무 강한 샤프닝 마스크를 적용하여 결과 영상이 날카롭고 거친 느낌

3

기타 필터링

최대값/최소값 필터링

❖ 입력 영상의 해당 화소(중심화소)에서 마스크로 씌워진 영역의 입력화소들을 가져와서 그 중에 최댓값/최솟값을 출력화소로 결정하는 방법



❖ 최댓값 필터링

- 가장 큰 값인 밝은 색들로 출력화소가 구성
- 돌출되는 어두운 값이 제거 전체적으로 밝은 영상이 됨

❖ 최솟값 필터링

- 가장 작은 값들인 어두운 색들로 출력화소가 구성
- 돌출되는 밝은 값들이 제거되며, 전체적으로 어두운 영상 됨

예제 7.3.1 최솟값-최댓값 필터링 – 09.filter_minMax.py

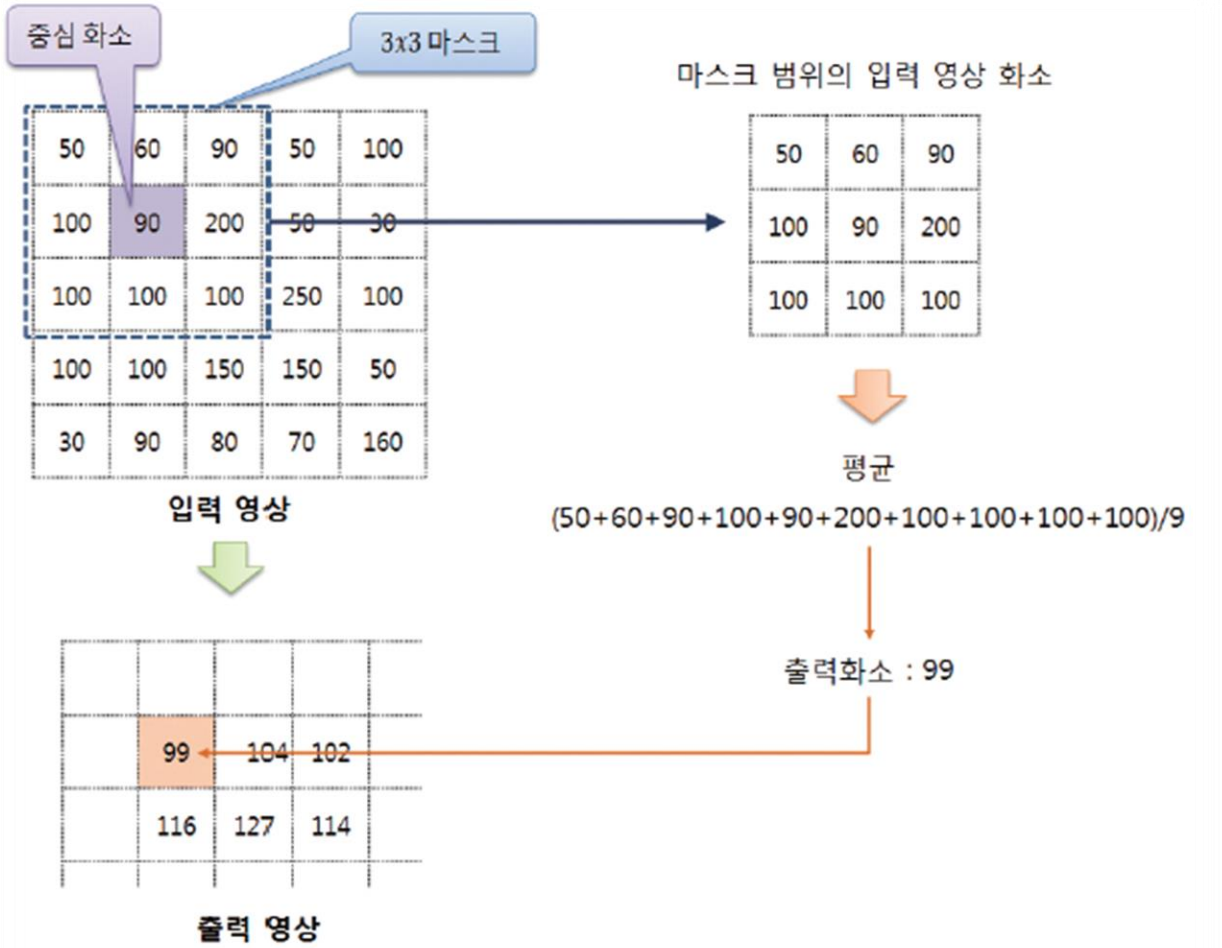
```
01  import numpy as np, cv2
02
03  def minmax_filter(image, ksize, mode):                  # 최솟값&최댓값 필터링 함수
04      rows, cols = image.shape[:2]
05      dst = np.zeros((rows, cols), np.uint8)
06      center = ksize // 2                                 # 마스크 절반 크기
07
08      for i in range(center, rows - center):              # 입력 영상 순회
09          for j in range(center, cols - center):
10              ## 마스크 영역 행렬 처리 방식
11              y1, y2 = i - center, i + center + 1         # 마스크 높이 범위
12              x1, x2 = j - center, j + center + 1         # 마스크 너비 범위
13              mask = image[y1:y2, x1:x2]                  # 마스크 영역
14              dst[i, j] = cv2.minMaxLoc(mask)[mode]       # 최소 or 최대
15      return dst
16
17  image = cv2.imread("images/min_max.jpg", cv2.IMREAD_GRAYSCALE)
18  if image is None: raise Exception("영상파일 읽기 오류")
19
20  minfilter_img = minmax_filter(image, 3, 0)              # 3×3 마스크 최솟값 필터링
21  maxfilter_img = minmax_filter(image, 3, 1)              # 3×3 마스크 최댓값 필터링
22
23  cv2.imshow("image", image)
24  cv2.imshow("minfilter_img", minfilter_img)
25  cv2.imshow("maxfilter_img", maxfilter_img)
26  cv2.waitKey(0)
```

1이면 최댓값 필터링 수행,
0이면 최솟값 필터링 수행

입력영상 끝단 마스크
절반 크기 조회 안함

평균값 필터링

❖ 마스크 영역 입력화소들의 평균을 출력화소로 지정하는 방법



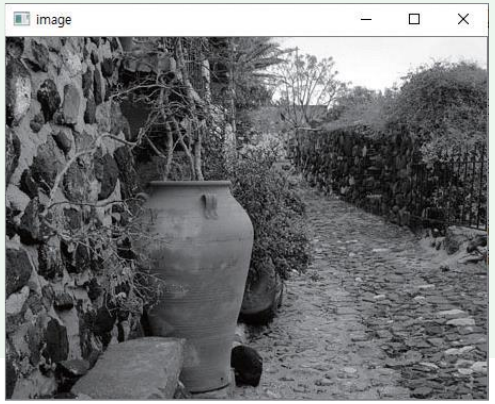
〈그림 7.3.2〉 평균값 필터링의 과정

예제 7.3.2 평균값 필터링 – 10.filter_average.py

```
01  import numpy as np, cv2
02
03  def average_filter(image, ksize):                    # 평균값 필터링 함수
04      rows, cols = image.shape[:2]
05      dst = np.zeros((rows, cols), np.uint8)
06      center = ksize // 2                              # 마스크 절반 크기
07
08      for i in range(rows):                            # 입력 영상 순회
09          for j in range(cols):
10              y1, y2 = i - center, i + center + 1      # 마스크 높이 범위
11              x1, x2 = j - center, j + center + 1      # 마스크 너비 범위
12              if y1 < 0 or y2 > rows or x1 < 0 or x2 > cols:    # 입력 영상 벗어남
13                  dst[i, j] = image[i, j]              # cv2.BORDER_CONSTANT 방식
14              else:
15                  mask = image[y1:y2, x1:x2]           # 범위 지정
16                  dst[i, j] = cv2.mean(mask)[0]        # np.mean(mask) 사용 가능
17      return dst
18
```

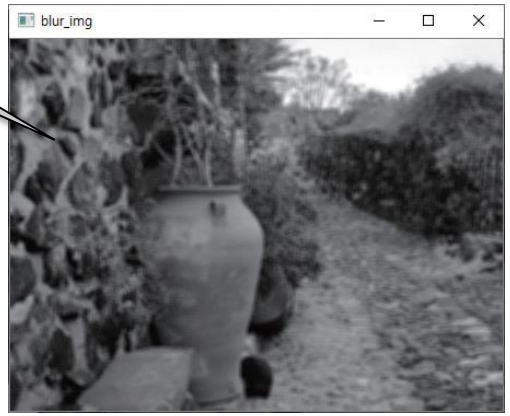
```
19  image = cv2.imread("images/avg_filter.jpg", cv2.IMREAD_GRAYSCALE)
20  if image is None: raise Exception("영상파일 읽기 오류")
21
22  avg_img  = average_filter(image, 5)                    # 사용자 정의 평균값 필터 함수
23  blur_img = cv2.blur(image, (5, 5), (-1,-1), cv2.BORDER_REFLECT)    # OpenCV의 블러링
24  box_img  = cv2.boxFilter(image, ddepth=-1, ksize=(5, 5) )    # OpenCV의 박스 필터 함수
25
26  cv2.imshow("image", image)
27  cv2.imshow("avg_img", avg_img)
28  cv2.imshow("blur_img", box_img)
29  cv2.imshow("box_img", box_img)
30  cv2.waitKey(0)
```

같은 기능을 수행하는 함수들



평균값 필터링

블러링

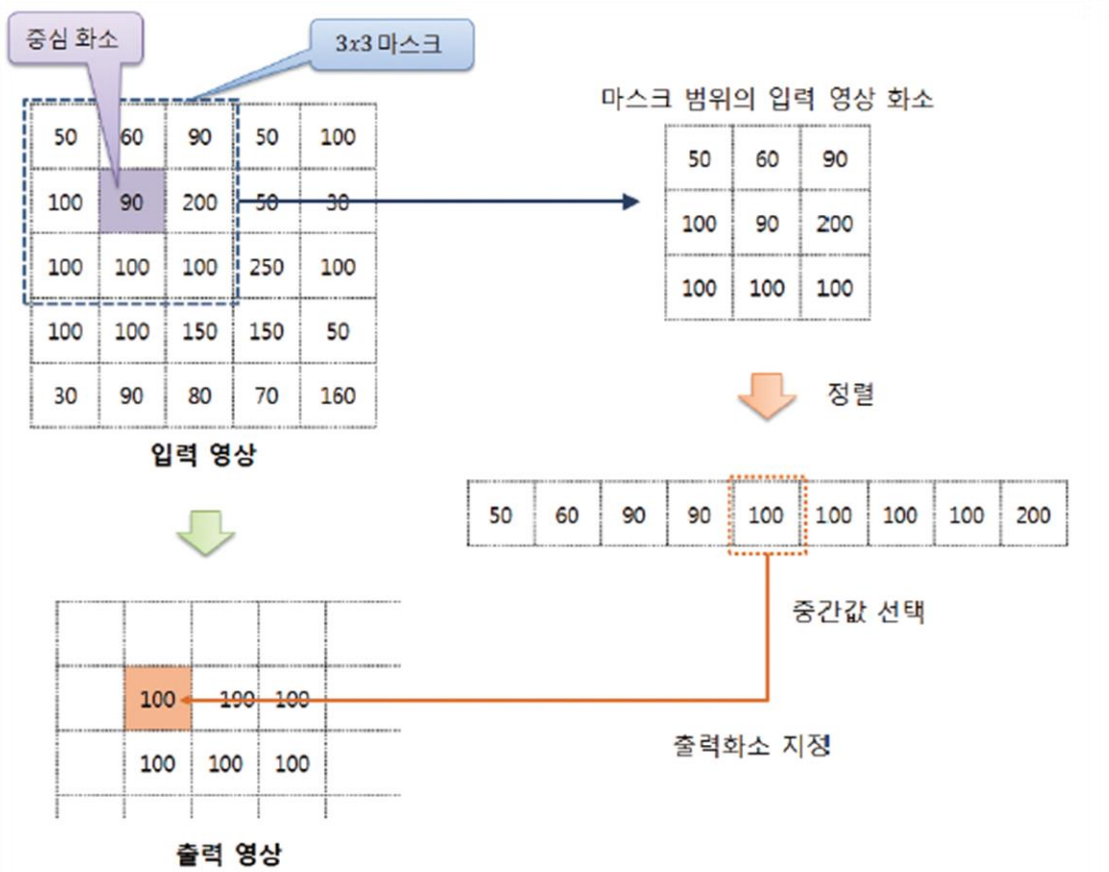


박스 필터링

미디언 필터링

❖ 마스크 범위 원소 중 중간값 취하여 출력화소로 결정하는 방식

- 마스크 범위내에 있는 화소값 정렬 필요
- 임펄스 잡음, 소금-후추 잡음 제거
- RGB 컬러
  - 3개채널 간의 상호 의존도가 커서
  - 잡음이 많아 질 수 있음

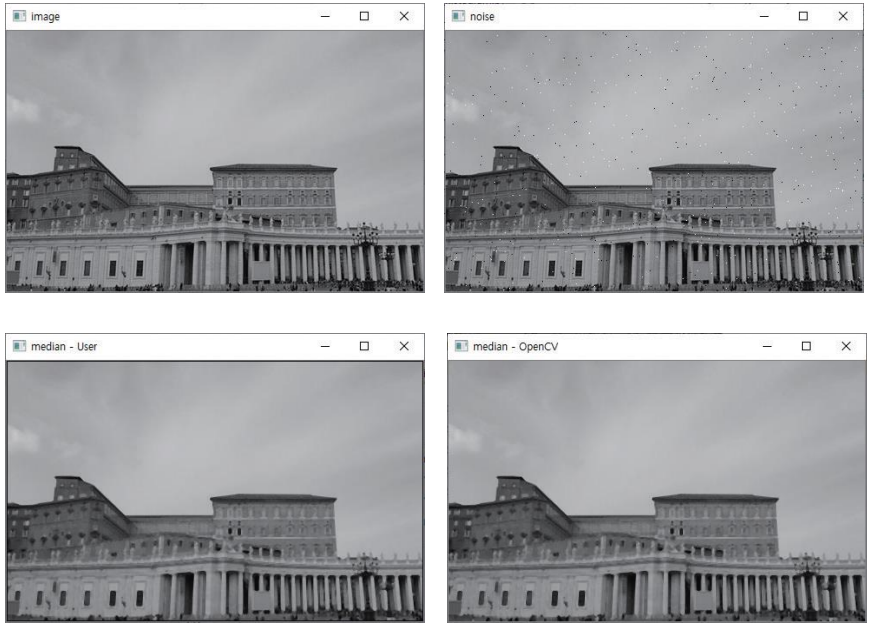


〈그림 7.3.3〉 미디언 필터링의 과정

예제 7.3.3 미디언 필터링 – 11.filter_median.py

```python
import numpy as np, cv2

def median_filter(image, size):                          # 미디언 필터링 함수
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2                                  # 마스크 절반 크기

    for i in range(center, rows - center):               # 입력 영상 순회
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1          # 마스크 높이 범위
            x1, x2 = j - center, j + center + 1          # 마스크 너비 범위
            mask = image[y1:y2, x1:x2].flatten()         # 관심 영역 지정 및 벡터 변환

            sort_mask = cv2.sort(mask, cv2.SORT_EVERY_COLUMN)          # 정렬 수행
            dst[i, j] = sort_mask[sort_mask.size//2] # 출력 화소로 지정
    return dst

def salt_pepper_noise(img, n):                           # 소금 후추 잡음 생성 함수
    h, w = img.shape[:2]
    x, y = np.random.randint(0, w, n), np.random.randint(0, h, n)
    noise = img.copy()
    for (x, y) in zip(x, y):
        noise[y, x] = 0 if np.random.rand() < 0.5 else 255
    return noise
```

```python
image = cv2.imread("images/median.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류")

noise = salt_pepper_noise(image, 500)                    # 소금-후추 잡음 영상 생성
med_img1 = median_filter(noise, 5)                       # 사용자 정의 함수
med_img2 = cv2.medianBlur(noise, 5)                      # OpenCV 제공 함수

cv2.imshow("image", image),
cv2.imshow("noise", noise),
cv2.imshow("median - User", med_img1)
cv2.imshow("median - OpenCV", med_img2)
cv2.waitKey(0)
```



가우시안 스무딩

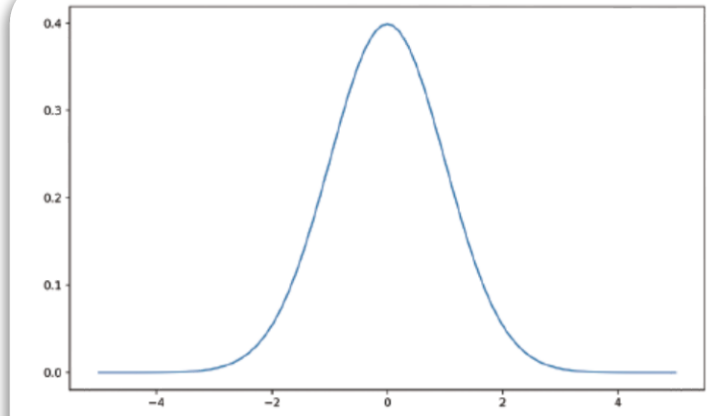
❖ 스무딩

- 회선을 통해서 영상의 세세한 부분을 부드럽게 하는 기법
- 대표적인 방법 - 가우시안 필터링

$$N(\mu,\sigma)(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

❖ 가우시안 분포(정규 분포)

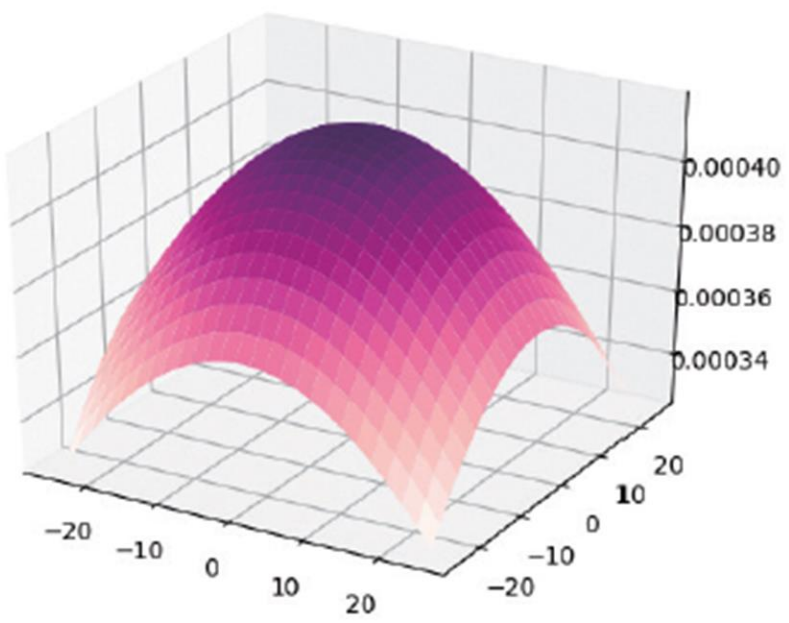
- 특정 값의 출현 비율을 그래프로 그렸을 때, 평균에서 가장 큰 수치 가짐
- 평균을 기준으로 좌우 대칭 형태
- 양끝으로 갈수록 수치가 낮아지는 종 모양
  - 평균과 표준 편차로 그래프 모양 변경



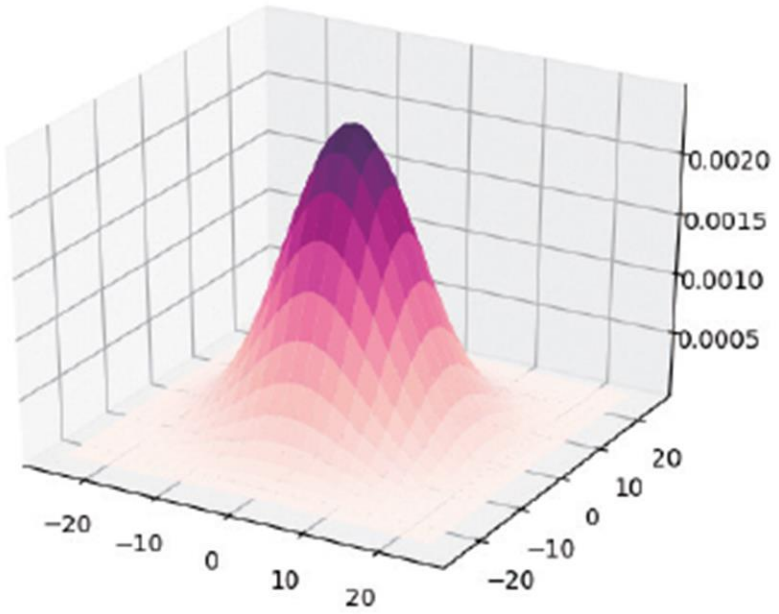
〈그림 7.3.4〉 정규 분포 그래프

❖ 2차원 가우시안 분포

$$N(\mu,\ \sigma_x,\ \sigma_y)(x,\ y) = \frac{1}{\sigma_x \sigma_y 2\pi} \exp\left[-\left(\frac{(x-\mu)^2}{2\sigma_x^2} + \frac{(y-\mu)^2}{2\sigma_y^2}\right)\right]$$



ksize = (50, 50), $\sigma_x, \sigma_y = 50, 50$



ksize = (50, 50), $\sigma_x, \sigma_y = 8.8$

〈그림 7.3.5〉 2차원 정규 분포 그래프의 예

예제 7.3.4 가우시안 필터링 – 12.filter_gausian.py

```
01  import numpy as np, cv2
02
03  def getGaussianMask(ksize, sigmaX, sigmaY):          # 가우시안 마스크 생성 함수
04      sigma = 0.3 * ((np.array(ksize) - 1.0) * 0.5 - 1.0) + 0.8
05      if sigmaX <= 0: sigmaX = sigma[0]                # 표준편차 양수 아닐 때,
06      if sigmaY <= 0: sigmaY = sigma[1]                # ksize로 기본 표준편차 계산
07
08      u = np.array(ksize)//2                           # 커널 크기 절반
09      x = np.arange(-u[0], u[0]+1, 1)                  # x 방향 범위
10      y = np.arange(-u[1], u[1]+1, 1)                  # y 방향 범위
11      x, y = np.meshgrid(x, y)                         # 정방 행렬 생성
12
13      ratio = 1 / (sigmaX * sigmaX * 2 * np.pi)
14      v1 = x ** 2 / (2 * sigmaX ** 2)
15      v2 = y ** 2 / (2 * sigmaY ** 2)
16      mask = ratio * np.exp(-(v1+v2))                  # 2차원 정규분포 수식
17      return mask / np.sum(mask)                       # 원소 전체 합 1 유지
```

```
Run: 12.filter_gaussian
C:\Python\python.exe D:/source/chap07/12.filter_gaussian.py
x=
[[-8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8]
 [-8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8]
 [-8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8]
 [-8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8]
 [-8 -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7  8]]
y=
[[-2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2]
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
 [ 2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2]]
```

```
18
19  image = cv2.imread("images/smoothing.jpg", cv2.IMREAD_GRAYSCALE)
20  if image is None: raise Exception("영상파일 읽기 오류")
21
22  ksize = (17, 5)                                    # 커널 크기: 가로×세로
23  gaussian_2d = getGaussianMask(ksize, 0, 0)
24  gaussian_1dX = cv2.getGaussianKernel(ksize[0], 0, cv2.CV_32F)    # 가로 방향 마스크
25  gaussian_1dY = cv2.getGaussianKernel(ksize[1], 0, cv2.CV_32F)    # 세로 방향 마스크
26
27  gauss_img1 = cv2.filter2D(image, -1, gaussian_2d)       # 사용자 생성 마스크 적용
28  gauss_img2 = cv2.GaussianBlur(image, size, 0)           # OepnCV 제공1–가우시안 블러링
29  gauss_img3 = cv2.sepFilter2D(image, -1, gaussian_1dX, gaussian_1dY)    # OpenCV 제공2
30
31  titles = ['image', 'gauss_img1', 'gauss_img2', 'gauss_img3']
32  for t in titles: cv2.imshow(t, eval(t))         # 문자열 리스트로 행렬들을 윈도우 표시
33  cv2.waitKey(0)
```

가로 방향이 큰 값

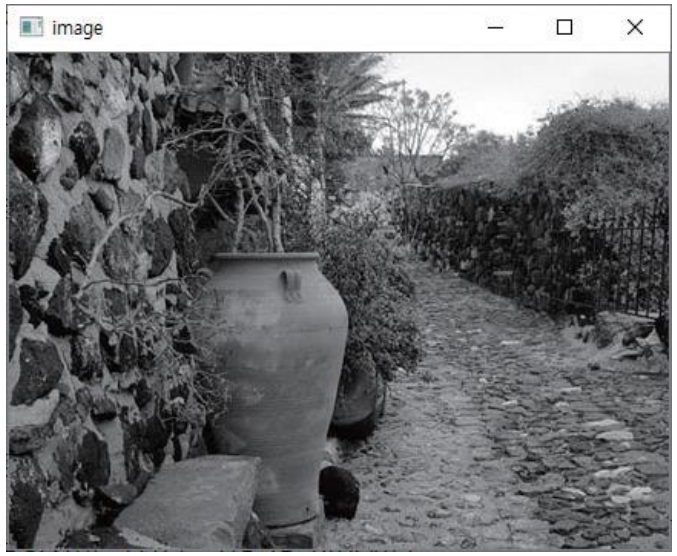
2차원 가우시안 마스크 생성

1차원 가우시안 마스크 생성
– x, y 방향

가우시안 블러링으로 수행

1차원 가우시안 마스크로 cv2.setFilter2D() 함수에 적용

❖ 실행결과



가로 방향으로 흐림

심화예제 7.3.5 블러링과 캐니 에지를 이용한 컬러 에지 검출 – 13.edge_color_canny.py

```
01  import cv2
02
03  def onTrackbar(th):                                          # 트랙바 콜백 함수
04      rep_edge = cv2.GaussianBlur(rep_gray, (5, 5), 0)         # 가우시안 블러링
05      rep_edge = cv2.Canny(rep_edge, th, th*2, 5)              # 캐니 에지 검출
06      h, w = image.shape[:2]
07      cv2.rectangle(rep_edge, (0, 0, w, h), 255, -1)           # 흰색 사각형 그리기
08      color_edge = cv2.bitwise_and(rep_image, rep_image, mask=rep_edge)
09      cv2.imshow("color edge", color_edge)
10
11  image = cv2.imread("images/color_edge.jpg", cv2.IMREAD_COLOR)
12  if image is None: raise Exception("영상파일 읽기 오류")
13
14  th = 50
15  rep_image = cv2.repeat(image, 1, 2)                          # 가로 반복 복사
16  rep_gray = cv2.cvtColor(rep_image, cv2.COLOR_BGR2GRAY)       # 명암도 영상 변환
17
18  cv2.namedWindow("color edge", cv2.WINDOW_AUTOSIZE)           # 윈도우 생성
19  cv2.createTrackbar("Canny th", "color edge", th, 100, onTrackbar) # 콜백 함수 등록
20  onTrackbar(th)                                               # 콜백 함수 첫 실행
21  cv2.waitKey(0)
```

에지만 검출위해 명암도 영상에서 부드러운 부분 제거

낮은 임계값

높은 임계값

복사시 에지 영상을 마스크로 사용 – 컬러 영상의 에지 부분만 복사