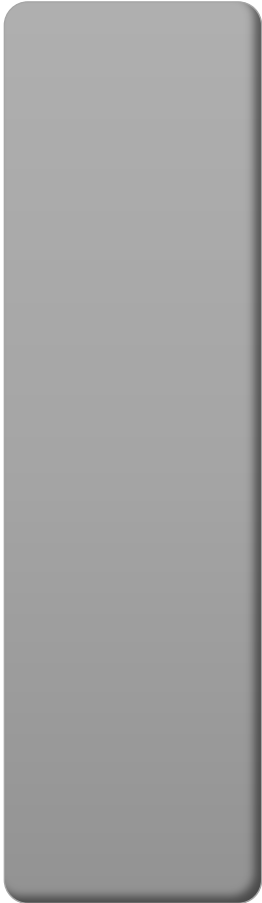


# **CHAPTER 07**

## **컨볼루션(Convolution)\_1**

# contents

- 
- 7.1 컨볼루션(convolution)
  - 7.2 에지 검출
  - 7.3 기타 필터링
  - 7.4 모폴로지(morphology)

## 7.1 컨볼루션(convolution)

- 7.1.1 공간 영역의 개념과 컨볼루션
- 7.1.2 블러링
- 7.1.3 샤프닝

## 7.1.1 공간 영역의 개념과 컨볼루션

- 화소 기반 처리
  - 화소값 각각에 대해 여러 가지 연산 수행
- 역 기반 처리
  - 마스크(mask)라 불리는 규정된 영역을 기반으로 연산 수행
    - 커널(kernel), 윈도우(window), 필터(filter)

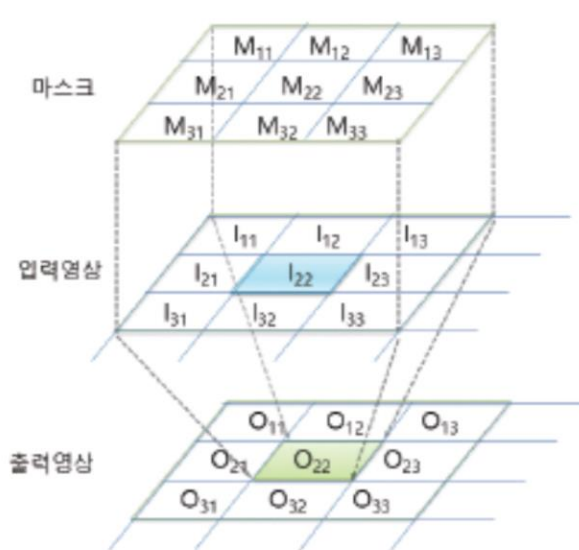
## 7.1.1 공간 영역의 개념과 회선(convolution)

The general expression of a convolution is

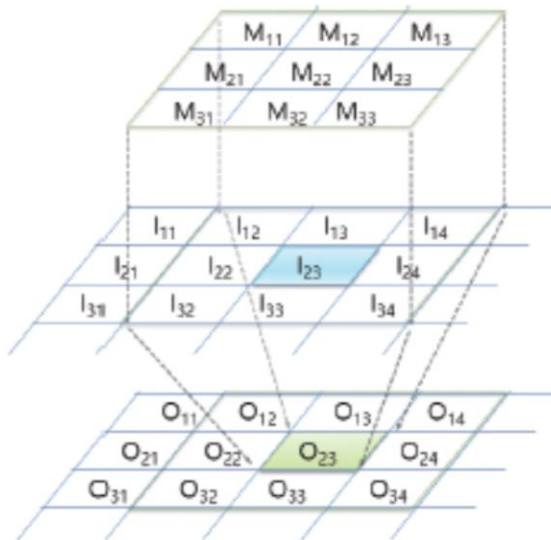
$$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^a \sum_{dy=-b}^b \omega(dx, dy) f(x + dx, y + dy),$$

where  $g(x, y)$  is the filtered image,  $f(x, y)$  is the original image,  $\omega$  is the filter kernel. Every element of the filter kernel is considered by  $-a \leq dx \leq a$  and  $-b \leq dy \leq b$ .

## 7.1.1 공간 영역의 개념과 회선(convolution)



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



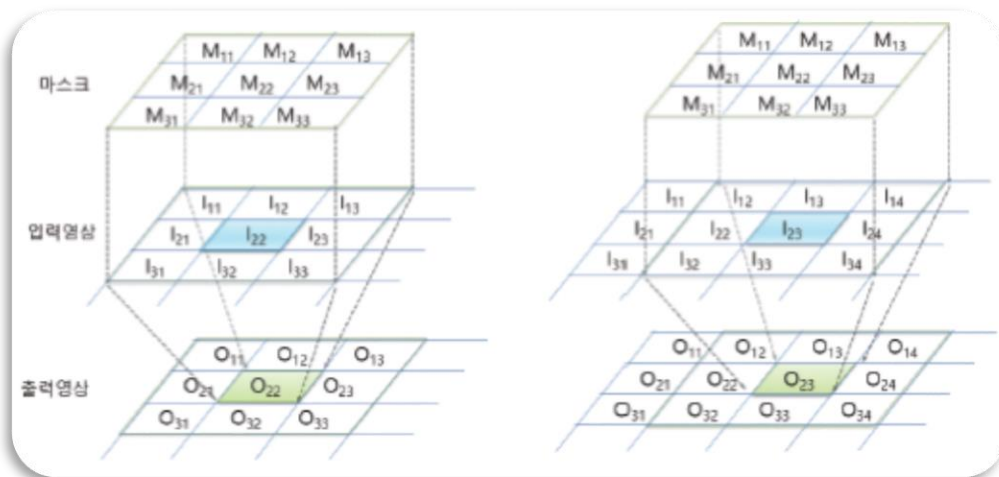
$$\begin{aligned}O_{23} &= \sum \sum M_{ij} \cdot I_{ij} \\&= M_{11} \cdot I_{12} + M_{12} \cdot I_{13} + M_{13} \cdot I_{14} \\&\quad + M_{21} \cdot I_{22} + M_{22} \cdot I_{23} + M_{23} \cdot I_{24} \\&\quad + M_{31} \cdot I_{32} + M_{32} \cdot I_{33} + M_{33} \cdot I_{34}\end{aligned}$$


〈그림 7.1.1〉 회선의 과정에 대한 이해

## 7.1.1 공간 영역의 개념과 회선(convolution)

```
for each image row in input image:  
  for each pixel in image row:  
  
    set accumulator to zero  
  
    for each kernel row in kernel:  
      for each element in kernel row:  
  
        if element position corresponding* to pixel position then  
          multiply element value corresponding* to pixel value  
          add result to accumulator  
        endif  
  
    set output image pixel to accumulator
```

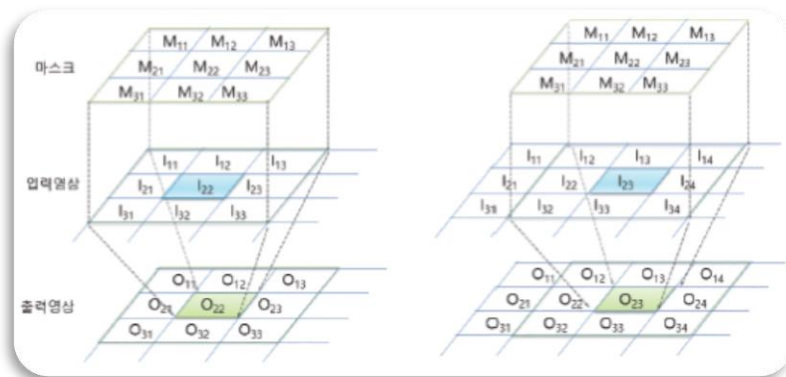
## 7.1.1 공간 영역의 개념과 회선(convolution)



Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	



## 7.1.1 공간 영역의 개념과 회산(convolution)

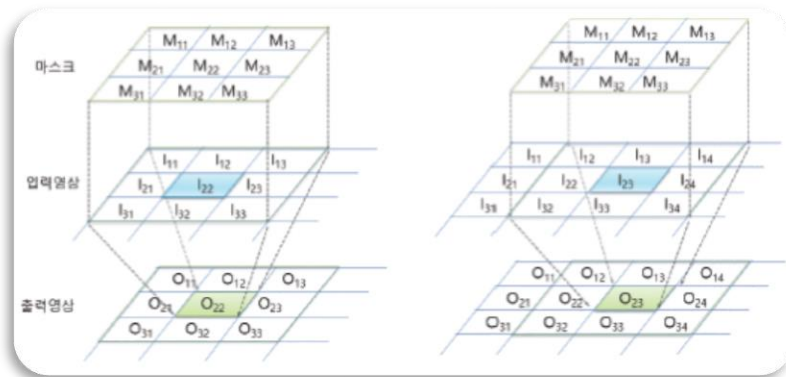


Original

0	0	0
0	1	0
0	0	0

?

## 7.1.1 공간 영역의 개념과 회산(convolution)



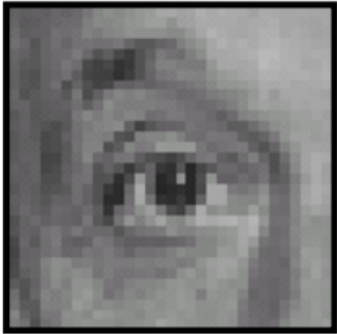
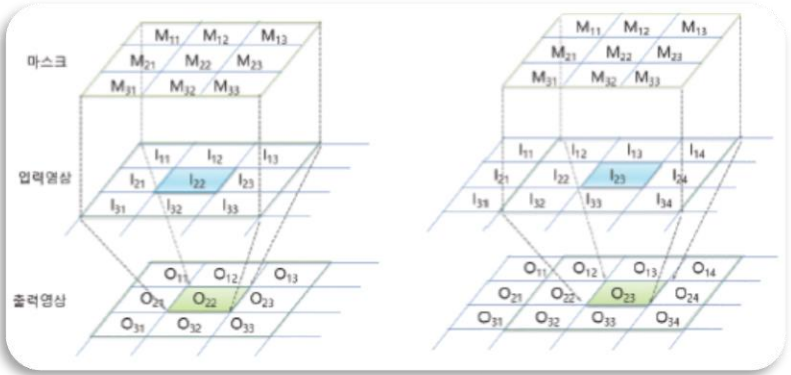
Original

0	0	0
0	1	0
0	0	0



Filtered  
(no change)

# 7.1.1 공간 영역의 개념과 회산(convolution)

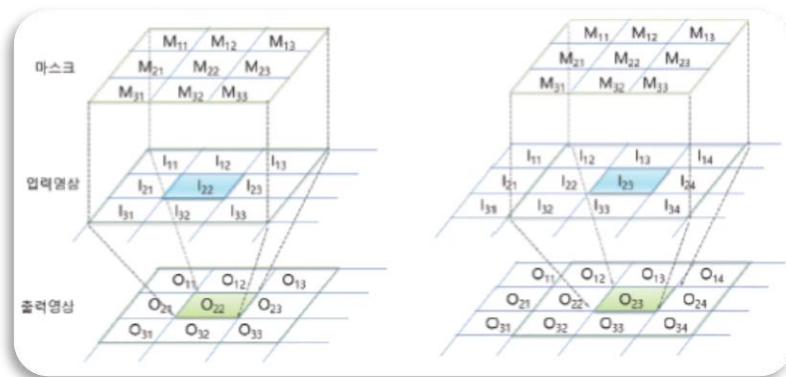


Original

0	0	0
0	0	1
0	0	0

?

## 7.1.1 공간 영역의 개념과 회산(convolution)



Original

0	0	0
0	0	1
0	0	0



Shifted **left**  
By 1 pixel

# 선형 필터의 특성

- 선형성(Linearity)
  - $\text{imfilter}(I, f1 + f2) = \text{imfilter}(I, f1) + \text{imfilter}(I, f2)$
- 이동 불변성(shift invariance)
  - $\text{Imfilter}(I, \text{shift}(f)) = \text{shift}(\text{imfilter}(I, f))$
- 교환 법칙(commutative):  $a*b=b*a$ 
  - 개념적으로는 필터와 이미지의 차이는 없으나, 실질적인 구현에서는 영향을 줄 수 있음
- 결합 법칙(associative):  $a*(b*c) = (a*b)*c$ 
  - 이미지에 여러 개의 필터를 여러 번 적용할 경우:  $((a*b)*c)*d$
  - 필터끼리 먼저 처리 후 이미지에 적용해도 된다:  $a*(b*c*d)$

## 7.1.2 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

## 7.1.2 블러링

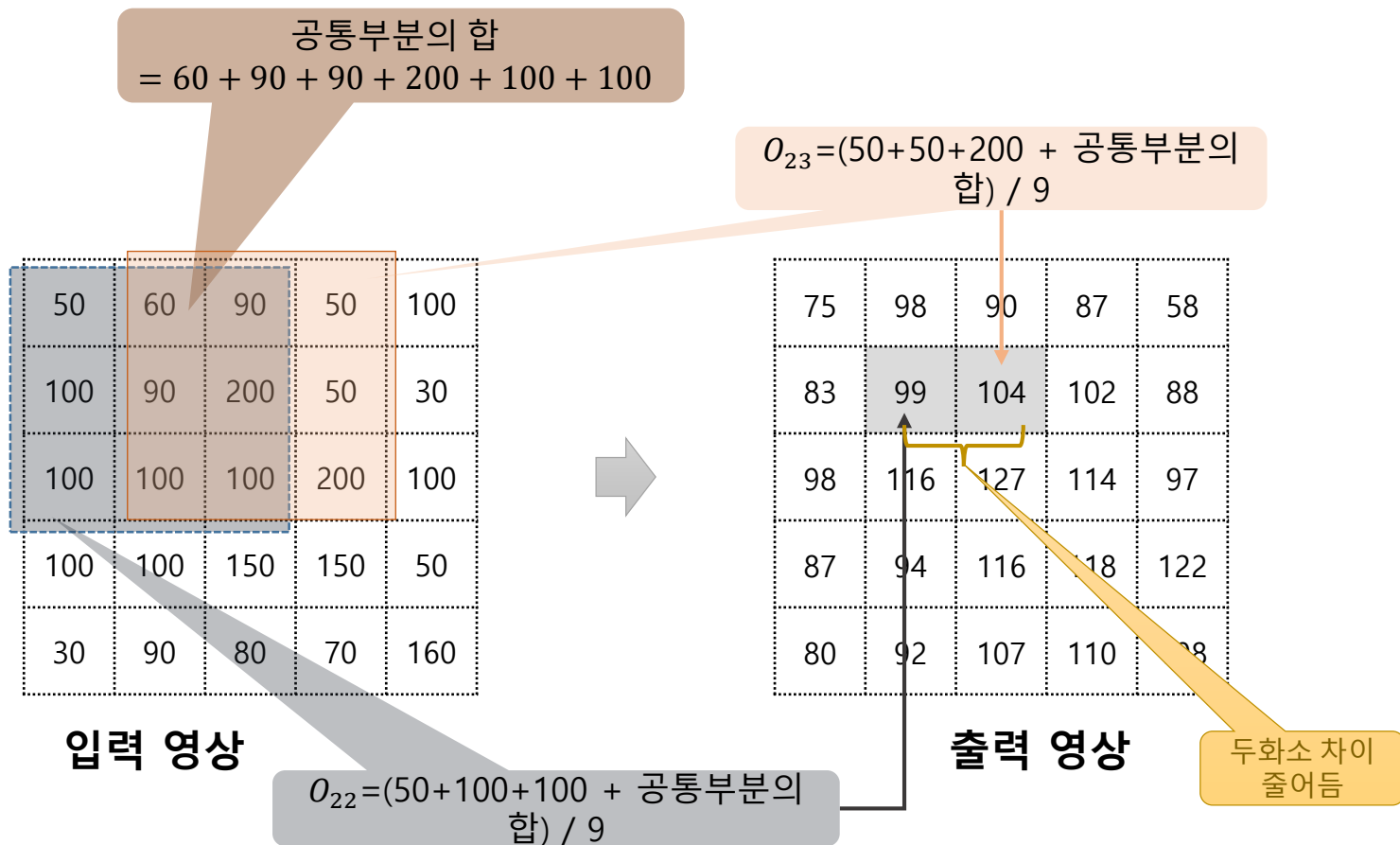
- 화소값이 급격히 변화하는 것을 점진적으로 변하게 하는 방법
  - → 블러링 마스크로 Convolution 수행
- 블러링 마스크

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

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

〈그림 7.1.2〉 블러링 마스크의 예

## 7.1.2 블러링





## 7.1.2 블러링

예제 7.1.1

회선이용 블러링 - 01.bluring.py

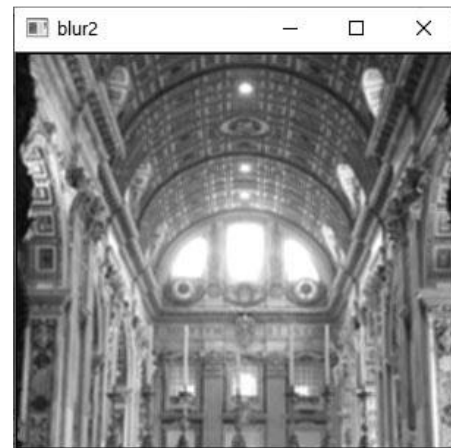
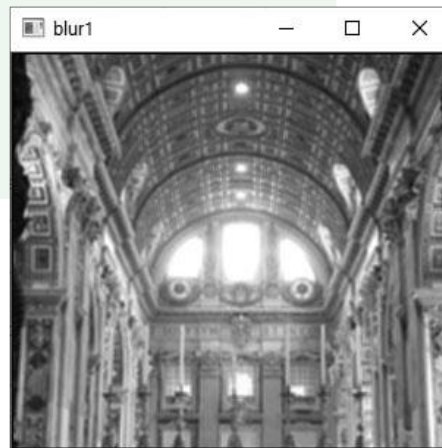
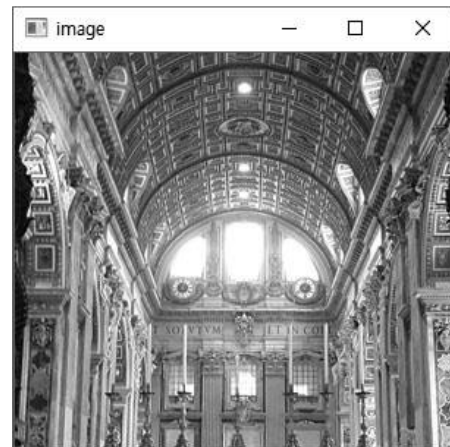
```
01 import numpy as np, cv2
02
03 ## 회선 수행 함수 - 행렬 처리 방식(속도 면에서 유리)
04 def filter(image, mask):
05     rows, cols = image.shape[:2]
06     dst = np.zeros((rows, cols), np.float32) # 회선 결과 저장 행렬
07     ycenter, xcenter = mask.shape[0]//2, mask.shape[1]//2 # 마스크 중심 좌표
08
09     for i in range(ycenter, 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
```

## 7.1.2 블러링

```
18 ## 회선 수행 함수- 화소 직접 근접
19 def filter2(image, mask):
20     rows, cols = image.shape[:2]
21     dst = np.zeros((rows, cols), np.float32)           # 회선 결과 저장 행렬
22     ycenter, xcenter = mask.shape[0]//2, mask.shape[1]//2 # 마스크 중심 좌표
23
24     for i in range(ycenter, rows - ycenter):           # 입력 행렬 반복 순회
25         for j in range(xcenter, cols - xcenter):
26             sum = 0.0
27             for u in range(mask.shape[0]):              # 마스크 원소 순회
28                 for v in range(mask.shape[1]):
29                     y, x = i + u - ycenter, j + v - xcenter
30                     sum += image[y, x] * mask[u, v]      # 회선 수식
31             dst[i, j] = sum
32     return dst
```

## 7.1.2 블러링

```
34 image = cv2.imread("images/filter_blur.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
35 if image is None: raise Exception("영상파일 읽기 오류")
36
37 data = [ 1/9, 1/9, 1/9,                                # 블러링 마스크 원소 지정
38         1/9, 1/9, 1/9,
39         1/9, 1/9, 1/9]
40 mask = np.array(data, np.float32).reshape(3, 3)          # 마스크 행렬 생성
41 blur1 = filter(image, mask)                               # 회선 수행- 행렬 처리 방식
42 blur2 = filter2(image, mask)                              # 회선 수행- 화소 직접 접근
43 blur1 = blur1.astype('uint8')                            # 행렬 표시위해 uint8형 변환
44 blur2 = cv2.convertScaleAbs(blur2)
45
46 cv2.imshow("image", image)
47 cv2.imshow("blur1", blur1)
48 cv2.imshow("blur2", blur2)
49 cv2.waitKey(0)
```



## 7.1.3 샤프닝

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

마스크 중심계수

〈그림 7.1.4〉 샤프닝 마스크의 예

## 7.1.3 샤프닝

### 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 импорт
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)
```

convolution

## 7.1.3 샤프닝

- 실행결과



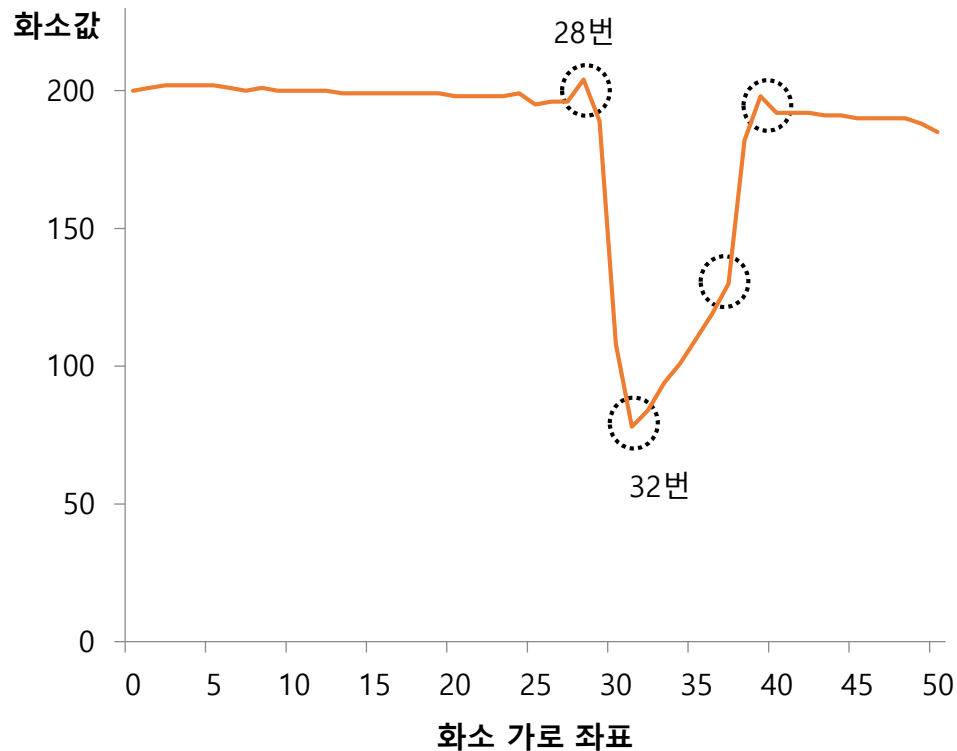
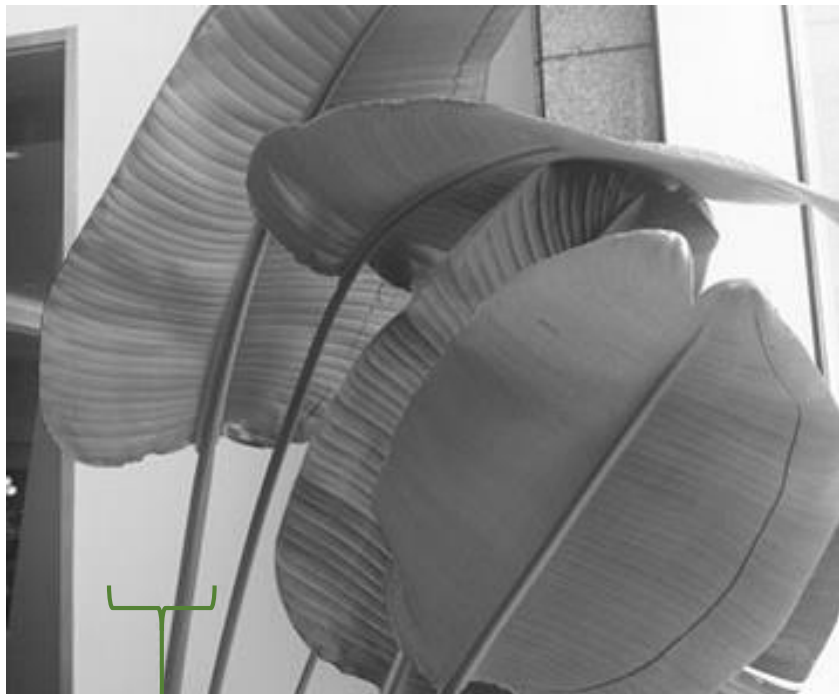
너무 강한 샤프닝 마스크를 적용하여 결과 영상이 날카롭고 거친 느낌

## 7.2 에지 검출

- 7.2.1 차분 연산을 통한 에지 검출
- 7.2.2 1차 미분 마스크
- 7.2.4 2차 미분 마스크

## 7.2 에지 검출

- 화소값의 그래프 표현



범위안 한 행의 화소값을 그래프로 표현



## 7.2.2 1차 미분 마스크

### • 미분

- 함수의 순간 변화율을 구하는 계산 과정을 의미
- 에지가 화소의 밝기가 급격히 변하는 부분이기 때문에 함수의 변화율을 취하는 미분 연산을 이용해서 에지 검출 가능

### • 밝기의 변화율을 검출하는 방법

- 밝기에 대한 기울기(gradient)를 계산

$$G[f(x, y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

$$G_x = \frac{f(x+dx, y) - f(x, y)}{dx} \doteq f(x+1, y) - f(x, y), \quad dx = 1$$

$$G_y = \frac{f(x, y+dy) - f(x, y)}{dy} \doteq f(x, y+1) - f(x, y), \quad dy = 1$$

$$G[f(x, y)] \doteq \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

디지털 영상은 1픽셀  
씩으로 근사

밝기 변화율 : 에지 강도

기울기 : 에지의 방향

# 1) 로버츠(Roberts) 마스크

- 대각선 방향으로 1과 -1을 배치하여 구성

 $G_x =$ 

-1	0	0
0	1	0
0	0	0

대각방향 마스크 1

 $G_y =$ 

0	0	-1
0	1	0
0	0	0

대각방향 마스크 2

$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

〈그림 7.2.3〉 3×3 크기의 로버츠 마스크

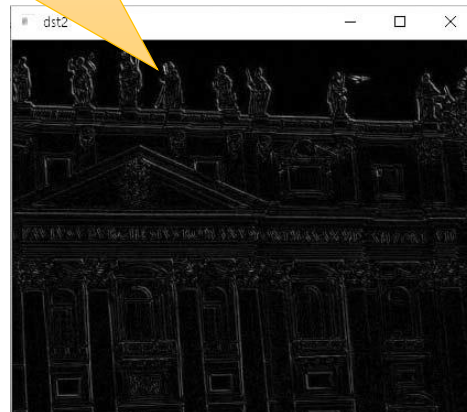
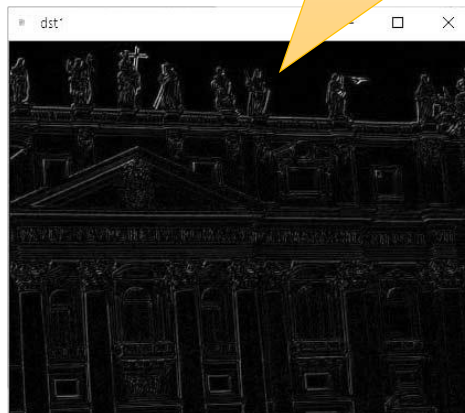
# 1) 로버츠(Roberts) 마스크

## • 실행결과



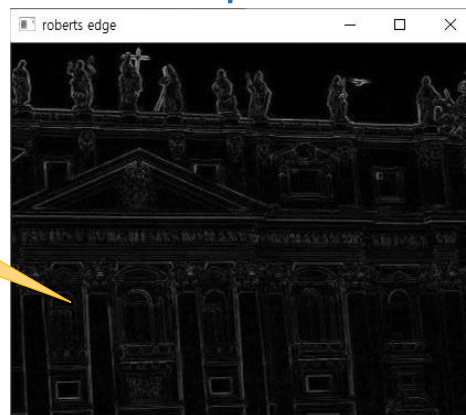
입력영상

각 대각선 방향 에지 검출 영상



$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

각 대각선 방향 에지의 크기로  
에지 강도 검출



## 2) 프리윗(Prewitt) 마스크

- 로버츠 마스크의 단점을 보완하기 위해 고안
  - 수직 마스크 - 원소의 배치가 수직 방향으로 구성, 에지의 방향도 수직
  - 수평 마스크 - 원소의 배치가 수평 방향으로 구성, 에지의 방향도 수평

$$G_x =$$

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

수직 마스크

$$G_y =$$

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

수평 마스크

$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

〈그림 7.2.4〉 3×3 크기의 로버츠 마스크

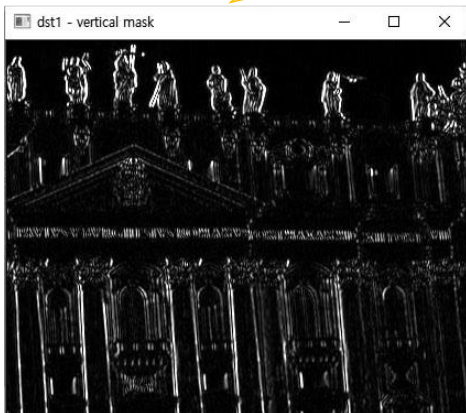
## 2) 프리윗(Prewitt) 마스크

- 실행결과

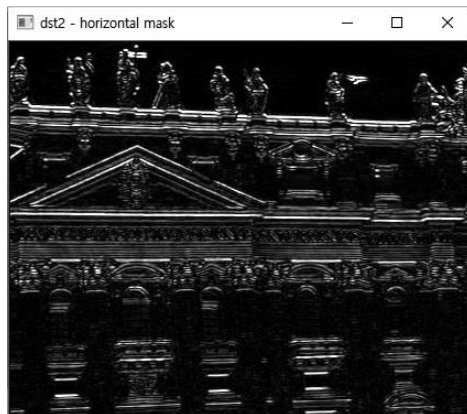


입력영상

수직방향 에지 검출 영상

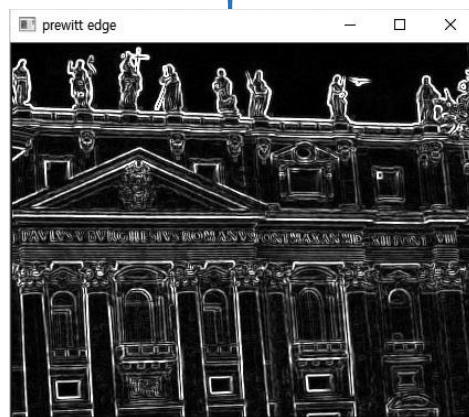


수평방향 에지 검출 영상



$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

두 방향 에지의 크기로  
에지 강도 검출



### 3) 소벨(Sobel) 마스크

- 프리윗 마스크와 유사, 중심화소의 차분에 대한 비중을 2배 키운 것이 특징
  - 수직, 수평 방향 에지 추출
  - 특히, 중심화소의 차분 비중을 높였기 때문에 대각선 방향 에지 검출

$$G_x =$$

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

수직마스크

$$G_y =$$

1	-2	1
0	0	0
1	2	1

수평마스크

$$O(i, j) = \sqrt{G_x^2 + G_y^2}$$

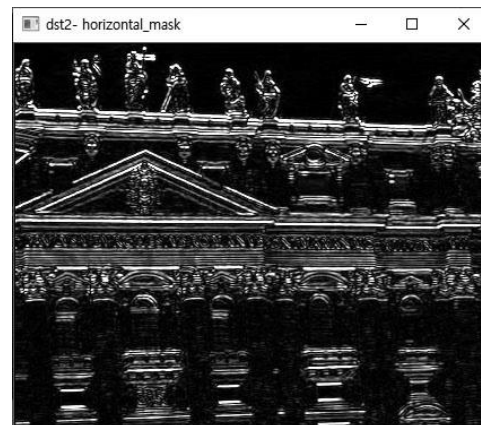
〈그림 7.2.5〉 3×3 크기의 소벨 마스크

### 3) 소벨(Sobel) 마스크

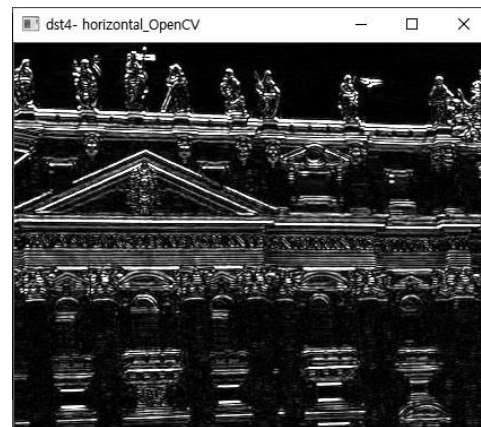
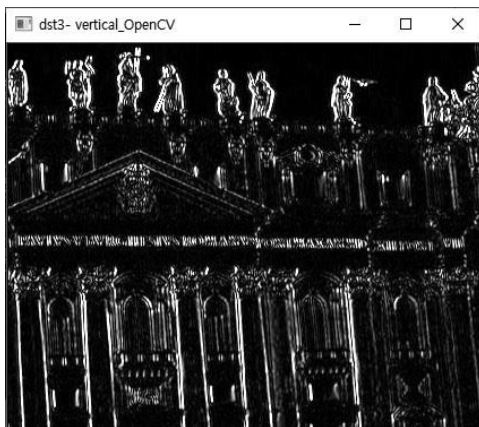
- 실행결과



입력영상



저자 구현 함수 적용



OpenCV 제공 함수 적용



## 7.2.4 2차 미분 마스크

- 1) 라플라시안 에지 검출

- 피에르시몽 라플라스라는 프랑스의 수학자 이름을 따서 지은 것
- 함수  $f$  에 대한 그래디언트의 발산으로 정의

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f \quad \Rightarrow \quad \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

2차원 좌표계

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} &= \frac{\partial f(x+1, y)}{\partial x} - \frac{\partial f(x, y)}{\partial x} \\ &= [f(x+1, y) - f(x, y)] - [f(x, y) - f(x-1, y)] \\ &= f(x+1, y) - 2 \cdot f(x, y) + f(x-1, y) \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 f}{\partial y^2} &= \frac{\partial f(x, y+1)}{\partial y} - \frac{\partial f(x, y)}{\partial y} \\ &= [f(x, y+1) - f(x, y)] - [f(x, y) - f(x, y-1)] \\ &= f(x, y+1) - 2 \cdot f(x, y) + f(x, y-1) \end{aligned}$$

$$\nabla^2 f(x, y) = f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4 \cdot f(x, y)$$



## 7.2.4 2차 미분 마스크

- 최종 수식

$$\nabla^2 f(x, y) = f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4 \cdot f(x, y)$$

0	-1	0
-1	4	-1
0	-1	0

0	1	0
1	-4	1
0	1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

1	1	1
1	-8	1
1	1	1

(a) 4방향 마스크

(b) 8방향 마스크

〈그림 7.2.6〉 라플라시안 마스크의 예

# 1) 라플라시안 에지 검출

예제 7.2.6

라플라시안 에지 검출 - 06.edge\_laplacian.py

```
01 import numpy as np, cv2
02
03 image = cv2.imread("images/laplacian.jpg", cv2.IMREAD_GRAYSCALE)
04 if image is None: raise Exception("영상파일 읽기 오류")
05
06 data1 = [ [0, 1, 0],
07           [1, -4, 1],
08           [0, 1, 0]]
09 data2 = [ [-1, -1, -1],
10           [-1, 8, -1],
11           [-1, -1, -1]]
12 mask4 = np.array(data1, np.int16)
13 mask8 = np.array(data2, np.int16)
14
15 dst1 = cv2.filter2D(image, |cv2.CV_16S, mask4)
16 dst2 = cv2.filter2D(image, cv2.CV_16S, mask8)
17 dst3 = cv2.Laplacian(image, cv2.CV_16S, 1)
18
19 cv2.imshow("image", image)
20 cv2.imshow("filter2D 4-direction", cv2.convertScaleAbs(dst1))
21 cv2.imshow("filter2D 8-direction", cv2.convertScaleAbs(dst2))
22 cv2.imshow("Laplacian_OpenCV", cv2.convertScaleAbs(dst3))
23 cv2.waitKey(0)
```

4방향 라플라시안 마스크 원소

# 4 방향 필터

8방향 라플라시안 마스크 원소

# 8 방향 필터

# 음수로 인해 int16형 행렬 선언

# OpenCV 회선 함수 호출

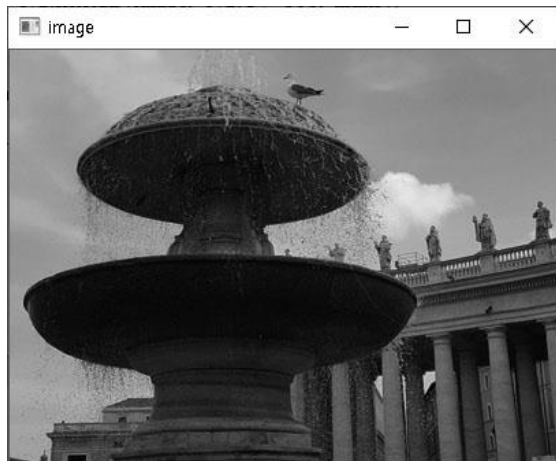
# OpenCV 라플라시안 수행 함수

# 형변환 후 영상 표시

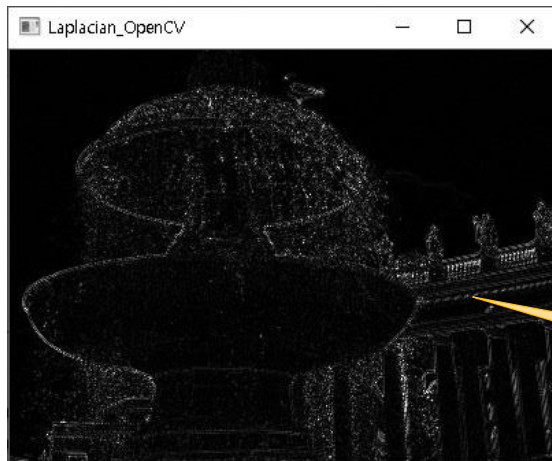
음수값 정리를 위해 절대값  
및 uchar 변환 한번에 수행

# 1) 라플라시안 에지 검출

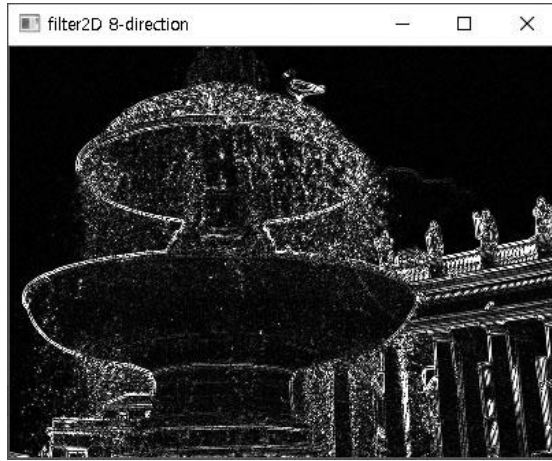
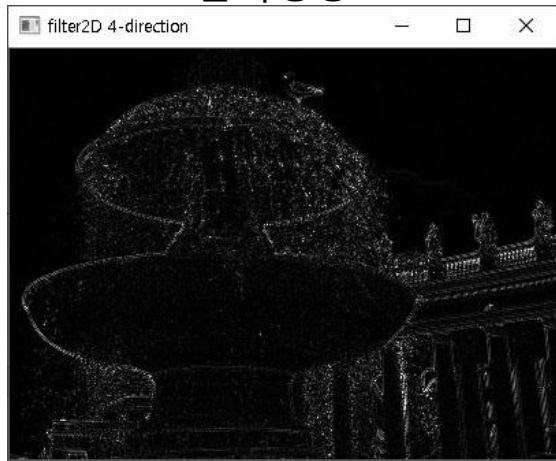
- 실행결과



입력영상



OpenCV 라플라시안



## 2) LoG와 DoG

- LoG(Laplacian of Gaussian)
  - 라플라시안은 잡음에 민감한 단점
  - 먼저 잡음 제거후 라플라시안 수행
    - 가우시안 필터링(블러) → 잡음에 강한 에지 검출 가능
  - 다양한 잡음 제거 방법 있음
    - 비선형 필터링은 계산에서 속도 저하문제 발생
    - 선형 필터링으로 단일 마스크 생성

$$\Delta[G_\sigma(x, y) * f(x, y)] = [\Delta G_\sigma(x, y)] * f(x, y) = LoG * f(x, y)$$

$$LoG(x, y) = \frac{1}{\pi\sigma^4} \left[ 1 - \frac{x^2 + y^2}{2\sigma^2} \right] \cdot e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

자세한 내용은:

<http://fourier.eng.hmc.edu/e161/lectures/gradient/node8.html>

## 2) LoG와 DoG

- DoG(Difference of Gaussian)
  - 단순한 방법으로 2차 미분 계산
  - 가우시안 스무딩 필터링의 차이를 이용해서 에지를 검출하는 방법

$$DoG(x, y) = \left( \frac{1}{2\pi\sigma_1^2} \cdot e^{-\frac{(x^2+y^2)}{2\sigma_1^2}} \right) - \left( \frac{1}{2\pi\sigma_2^2} \cdot e^{-\frac{(x^2+y^2)}{2\sigma_2^2}} \right) \quad (\text{단, } \sigma_1 < \sigma_2)$$

## 2) LoG와 DoG

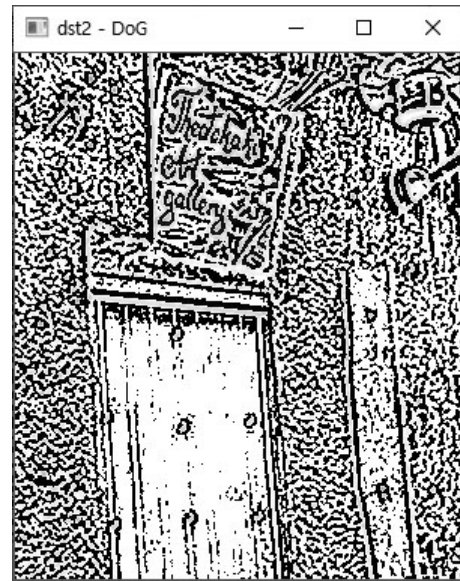
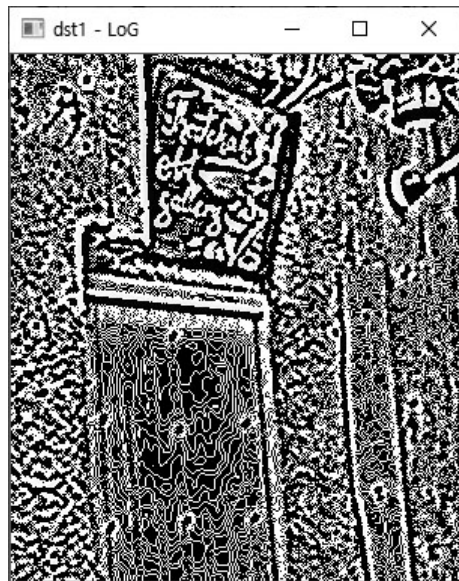
예제 7.2.7

LoG/DoG 에지 검출 -- 07.edge\_DOG.py

```
01 import numpy as np, cv2
02
03 image = cv2.imread("images/dog.jpg", cv2.IMREAD_GRAYSCALE)
04 if image is None: raise Exception("영상파일 읽기 오류")
05
06 gaus = cv2.GaussianBlur(image, (7, 7), 0, 0)          # 가우시안 마스크 적용
07 dst1 = cv2.Laplacian(gaus, cv2.CV_16S, 7)            # 라플라시안 수행
08
09 gaus1 = cv2.GaussianBlur(image, (3, 3), 0)           # 가우시안 블러링
10 gaus2 = cv2.GaussianBlur(image, (9, 9), 0)
11 dst2 = gaus1 - gaus2                                  # DoG 수행
12
13 cv2.imshow("image", image)
14 cv2.imshow("dst1- LoG", dst1.astype('uint8'))        # 형변환 후 영상 표시
15 cv2.imshow("dst2- DoG", dst2)
16 cv2.waitKey(0)
```

## 2) LoG와 DoG

- 실행결과



입력영상

## 단원 요약

- 컨볼루션(회선, convolution)은 마스크 내의 원소값과 공간 영역에 있는 입력 영상의 화소값들을 대응되게 곱하여 출력 화소값을 계산하는 것을 말한다.
  - 이때, 입력 영상에 곱해지는 이 마스크를 커널(kernel), 윈도우(window), 필터(filter) 등 용어로 부른다.



# 실습 과제

- (과제) 연습문제 10. (p.352)

- `Common.filters.filter()`는 명암도 영상에서 필터링을 수행한다. `Filter()` 함수를 이용해서 컬러 영상에서 블러링과 샤프닝을 구현해보자.
  - 컬러 영상의 채널을 분리해서 각 채널에 블러링과 샤프닝을 수행 후 합쳐보자.
  - 컬러 영상에서 바로 OpenCV 함수인 `cv2.filter2D()`를 적용해 보자.

- (보너스) 2차원 컨볼루션 구현

- 아래 수식/설명을 토대로 컨볼루션을 구현하고(교과서와 PPT를 참고하지 않고), 입력 이미지(gray-scale)에 각종 컨볼루션 연산을 적용해 보시오.

The general expression of a convolution is

$$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^a \sum_{dy=-b}^b \omega(dx, dy) f(x + dx, y + dy),$$

where  $g(x, y)$  is the filtered image,  $f(x, y)$  is the original image,  $\omega$  is the filter kernel.

Every element of the filter kernel is considered by  $-a \leq dx \leq a$  and  $-b \leq dy \leq b$ .

## 7. 실습 규칙

- 실습 과제는 실습 시간내로 해결해야 합니다.
  - 해결 못한경우 실습 포인트를 얻지 못합니다.
  - -> 집에서 미리 연습하고 오길 권장합니다.
- 코드 공유/보여주기 금지. 의논 가능.
- 보너스문제까지 해결한 학생은 조기 퇴실 가능