영역 기반 영상 처리
Region-based Image Processing

목 차

1. 회선(convolution)

❖공간 영역의 개념과 회선

- 화소 기반 처리 (6장)
  - 화소값 각각에 대해 여러 가지 연산 수행

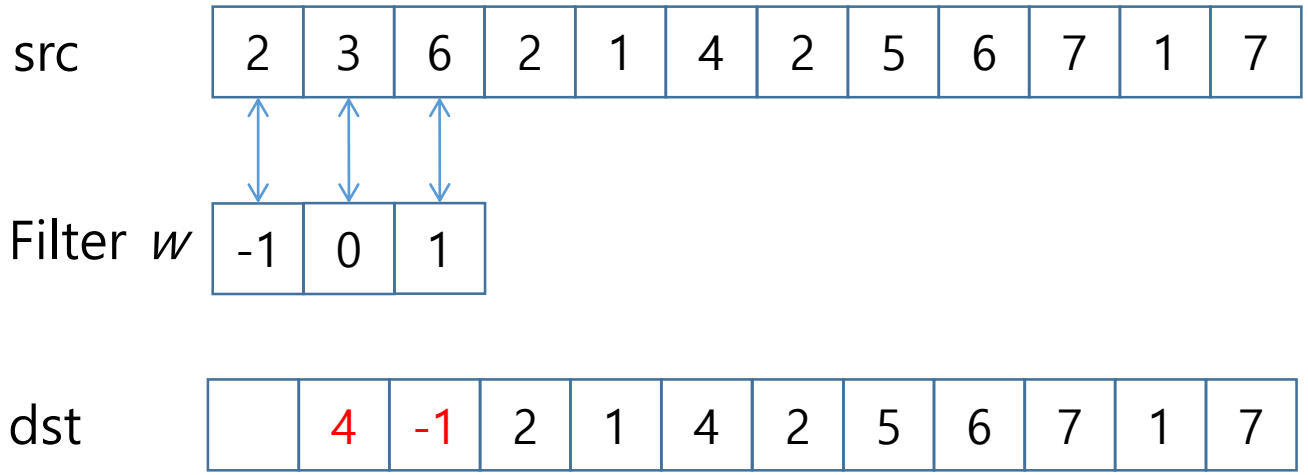
- 영역 기반 처리 (7장)
  - 마스크(mask)라 불리는 규정된 영역을 기반으로 연산 수행

    마스크(mask) = 커널(kernel) = 윈도우(window) = 필터(filter)

Correlation vs. Convolution

❖ 1D correlation

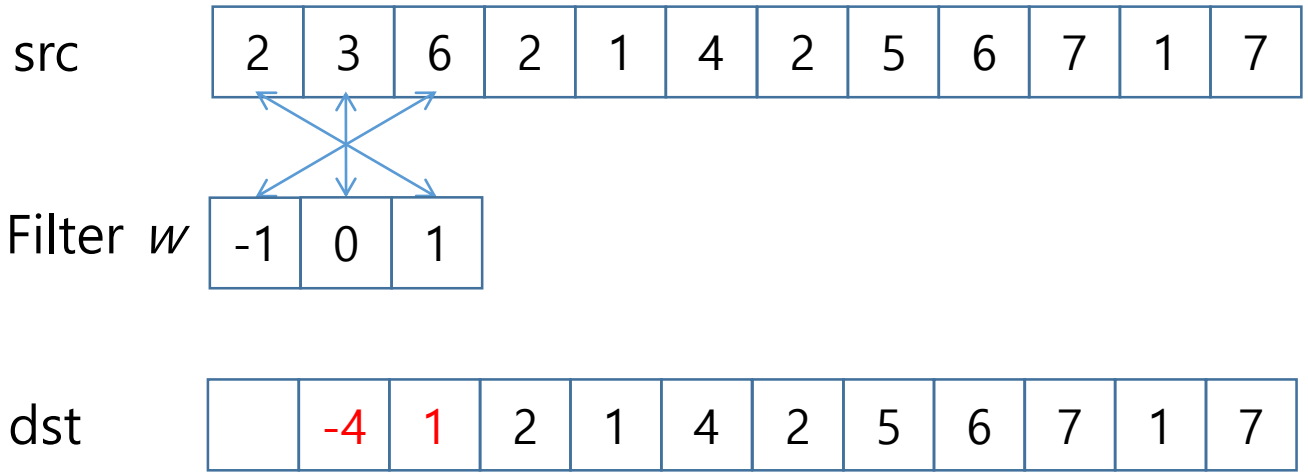
$$dst(x) = \sum_{s=-size}^{size} w(s)src(x+s)$$



Correlation vs. Convolution

❖ 1D convolution

$$dst(x) = \sum_{s=-size}^{size} w(s)src(x-s)$$



Correlation vs. Convolution

❖ 2D correlation

$$dst(x,y) = \sum_{s=-size}^{size} \sum_{t=-size}^{size} w(s,t)src(x+s, y+t)$$

src

1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1

Filter w

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

dst

1	2	4	5	2	1
1	12	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1

Correlation vs. Convolution

❖ 2D convolution

$$dst(x,y) = \sum_{s=-size}^{size} \sum_{t=-size}^{size} w(s,t)src(x-s,y-t)$$

src

1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1

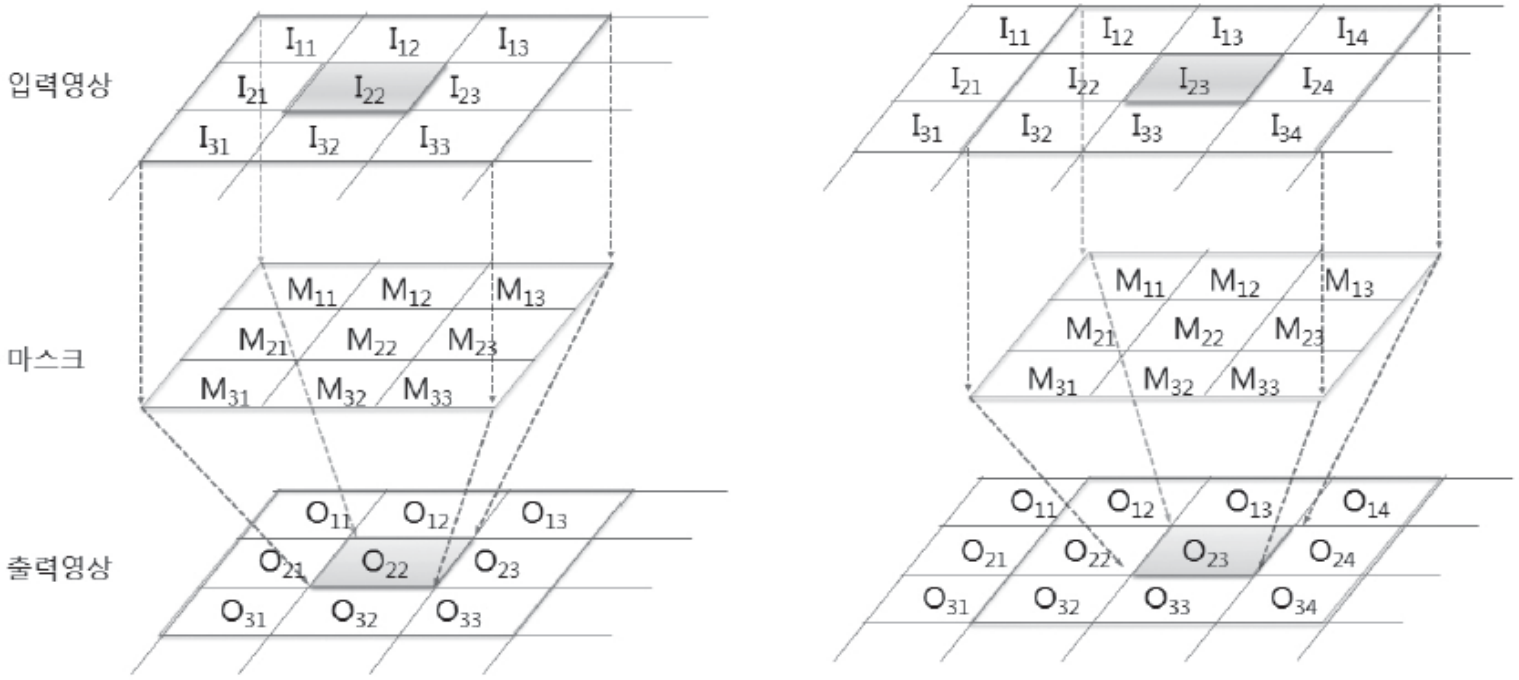
Filter w

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

dst

1	2	4	5	2	1
1	-12	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1
1	2	4	5	2	1

회선의 개념



$$O_{22} = \sum_{i=1}^{3}\sum_{j=1}^{3} I_{ij} \cdot M_{ij}$$
$$= I_{11} \cdot M_{11} + I_{12} \cdot M_{12} + I_{13} \cdot M_{13}$$
$$+ I_{21} \cdot M_{21} + I_{22} \cdot M_{22} + I_{23} \cdot M_{23}$$
$$+ I_{31} \cdot M_{31} + I_{32} \cdot M_{32} + I_{33} \cdot M_{33}$$

$$O_{23} = \sum_{i=1}^{3}\sum_{j=1}^{3} I_{ij} \cdot M_{ij}$$
$$= I_{12} \cdot M_{11} + I_{13} \cdot M_{12} + I_{14} \cdot M_{13}$$
$$+ I_{22} \cdot M_{21} + I_{23} \cdot M_{22} + I_{24} \cdot M_{23}$$
$$+ I_{32} \cdot M_{31} + I_{33} \cdot M_{32} + I_{34} \cdot M_{33}$$

〈그림 7.1.1〉 회선의 과정에 대한 이해

회선의 개념

입력 영상

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

마스크

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

출력 영상

75	98	90	87	58
83	99	104	102	88
98	116	127	114	97
87	94	116	118	122
80	92	107	110	108

블러링 (blurring)

❖블러링 현상

- 디지털카메라로 사진 찍을 때, 초점이 맞지 않으면

  → 사진 흐려짐

  → 이러한 현상을 이용해서 영상의 디테일한 부분을 제거하는 아웃 포커싱(out focusing) 기법으로 사진을 촬영하기도 함.

❖블러링(blurring)

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

블러링 (blurring)

- 화소값이 급격이 변화하는 것을 점진적으로 변하게 하는 방법
  → 블러링 마크스로 회선 수행

- 블러링 마스크
  → 원리: 평균값으로 변경

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

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

〈그림 7.1.2〉 블러링 마스크의 예

예제 7.1.1 회선이용 블러링 - 01.bluring.py

```python
import numpy as np, cv2

## 회선 수행 함수 - 행렬 처리 방식(속도 면에서 유리)
def filter(image, mask):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.float32)          # 회선 결과 저장 행렬
    ycenter, xcenter = rows//2, cols//2               # 마스크 중심 좌표

    for i in range(ycenter, rows - ycenter):          # 입
        for j in range(xcenter, cols - xcenter):
            y1, y2 = i - ycenter, i + ycenter + 1     # 관
            x1, x2 = j - xcenter, j + xcenter + 1     # 관
            roi = image[y1:y2, x1:x2].astype('float32')
            tmp = cv2.multiply(roi, mask)             # 회
            dst[i, j] = cv2.sumElems(tmp)[0] p.190    # 출
    return dst                                        # 자

```

```python
## 회선  수행 함수- 화소 직접 근접
def filter2(image, mask):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.float32)          # 회선 결과 저장 행렬
    ycenter, xcenter = rows//2, cols//2               # 마스크 중심 좌표

    for i in range(ycenter, rows - ycenter):          # 입력 행렬 반복 순회
        for j in range(xcenter, cols - xcenter):
            sum = 0.0
            for u in range(mask.shape[0]):            # 마스크 원소 순회
                for v in range(mask.shape[1]):
                    y, x = i + u - ycenter, j + v - xcenter
                    sum += image[y, x] * mask[u, v]   # 회선 수식
            dst[i, j] = sum
    return dst
```

```
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)  p.184   # 절대값을 구한 후 uint8로 변환
45
46  cv2.imshow("image", image)
47  cv2.imshow("blur1", blur1)
48  cv2.imshow("blur2", blur2)
49  cv2.waitKey(0)
```

```
void cv::convertScaleAbs ( InputArray  src,
                           OutputArray dst,
                           double      alpha = 1,
                           double      beta = 0
                         )
```

$$\mathtt{dst}(I) = \mathtt{saturate\_cast<uchar>}(|\mathtt{src}(I) * \mathtt{alpha} + \mathtt{beta}|)$$

cv2.filter2D

```
filter2D (InputArray src, OutputArray dst,
          int  ddepth[1],
          InputArray  kernel,
          Point anchor = Point(-1, -1) [2],
          double delta = 0,
          int  borderType=BORDER_DEFAULT
)
```

1) ddepth=-1 이면, dst.depth()=src.depth()
2) anchor=Point(-1, -1) 이면, kernel의 중심 위치

샤프닝 (sharping)

❖샤프닝(sharpening)

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

❖사프닝 마스크

- 마스크 원소 전체 합이 1이 되어야 입력 영상 밝기가 손실 없이 출력 영상에 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

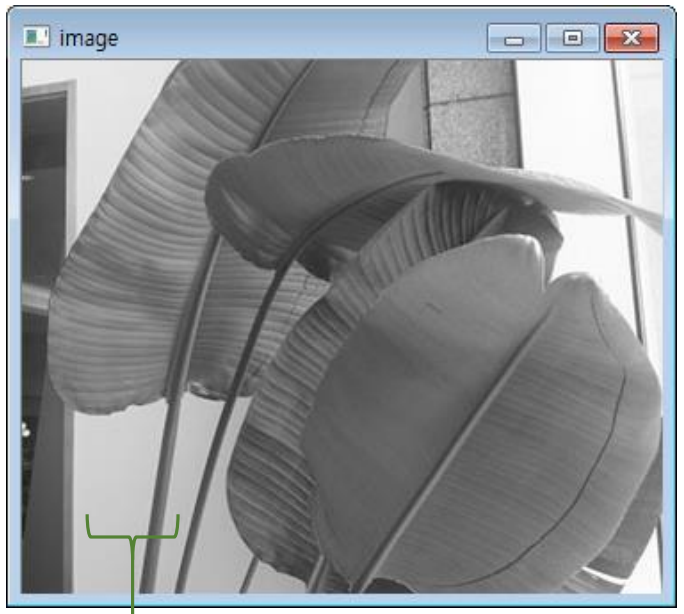
예제 7.1.2 회선이용 샤프닝 – 02.sharpening.py

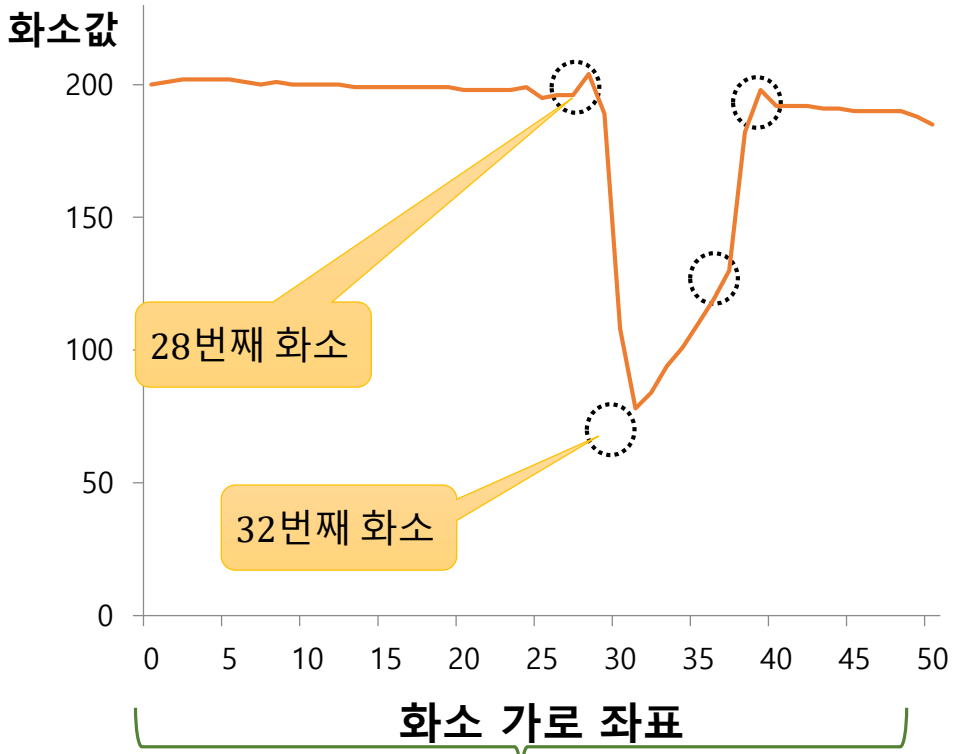
```
01  import numpy as np, cv2
02  from Common.filters import filter                    # filters 모듈의 filter() 함수 임포
03
04  image = cv2.imread("images/filter_sharpen.jpg", cv2.IMREAD_GRAYSCALE)
05  if image is None: raise Exception("영상파일 읽기 오류")
06
07  ## 샤프닝 마스크 원소 지정
08  data1 = [ 0, -1, 0,                                  # 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)
```

2. 에지 검출

❖에지(edge)

- 명도 값의 변화가 크게 나타나는 경계





범위 안 한 행의 화소값을 그래프로 표현

1차 미분을 이용한 에지 검출

❖1차 미분 연산

- 미분: 변화율을 측정
- $f'(x) = \frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = f(x+1) - f(x)$
- Gradient $\nabla f(x,y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$

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

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

1차 미분을 이용한 에지 검출

❖1차 미분 연산

- Gradient의 크기와 방향

$$\mathrm{mag}\left(\nabla f(x,y)\right) = \sqrt{\left({G_x}^2 + {G_y}^2\right)}$$

$$\mathrm{angle}\left(\nabla f(x,y)\right) = \mathrm{atan}\left(\frac{G_y}{G_x}\right)$$

차분 연산을 이용한 에지 검출

- 유사 연산자
  - 중심화소와 이웃화소 간 차의 최대값

$$\text{유사 연산자 출력화소} = \max(|c-m_0|, |c-m_1|, |c-m_2|, |c-m_3| \\ |c-m_5|, |c-m_6|, |c-m_7|, |c-m_8|)$$

m_0	m_1	m_2
m_3	C	m_5
m_6	m_7	m_8

- 차 연산자
  - 중심화소 주위의 마주보는 두 화소 간 차의 최대값

$$\text{차 연산자 출력화소} = \max(|m_0-m_8|, |m_1-m_7|, |m_2-m_6|, |m_3-m_5|$$

로버츠(Roberts) 마스크

❖Roberts filter

- 두 방향의 미분에 대한 크기(magnitude)로 에지 검출

-1	0	0
0	1	0
0	0	0

G_x

0	0	-1
0	1	0
0	0	0

G_y

$$\mathrm{Mag}(\mathrm{x},\mathrm{y}) = \sqrt{G_x^2 + G_y^2}$$

- 장점: 처리 속도가 빠르다.
- 단점: 잡음에 매우 민감하다.

예제 7.2.3 로버츠 에지 검출 – 03.edge_roberts.py

```python
import numpy as np, cv2
from Common.filters import filter                    # filte

def differential(image, data1, data2):
    mask1 = np.array(data1, np.float32).reshape(3, 3)  # 입
    mask2 = np.array(data2, np.float32).reshape(3, 3)

    dst1 = filter(image, mask1)                      # 저
    dst2 = filter(image, mask2)
    dst1, dst2 = np.abs(dst1), np.abs(dst2)          # 회
    dst = cv2.magnitude(dst1, dst2)                  # 두

    # Saturate 후 uint8로 변환
    dst = np.clip(dst, 0, 255).astype('uint8')       # 윈
    dst1 = np.clip(dst1, 0, 255).astype('uint8')     # 형
    dst2 = np.clip(dst2, 0, 255).astype('uint8')
    return dst, dst1, dst2                           # 3개 결과 행렬 반환

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

data1 = [ -1, 0, 0,                                  # 마스
          0, 1, 0,
          0, 0, 0]
data2 = [ 0, 0, -1,
          0, 1, 0,
          0, 0, 0]
dst, dst1, dst2 = differential(image, data1, data2)  # 회선

cv2.imshow("image", image)
cv2.imshow("roberts edge", dst)
cv2.imshow("dst1", dst1)
cv2.imshow("dst2", dst2)
cv2.waitKey(0)
```

p.174

G_x

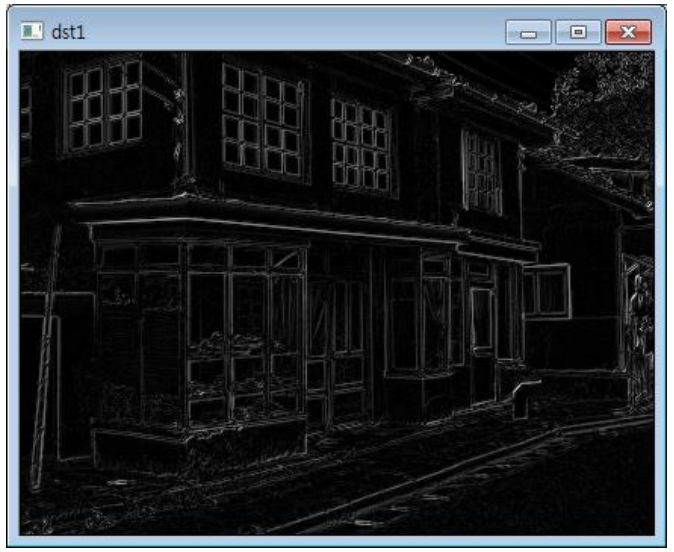
G_y

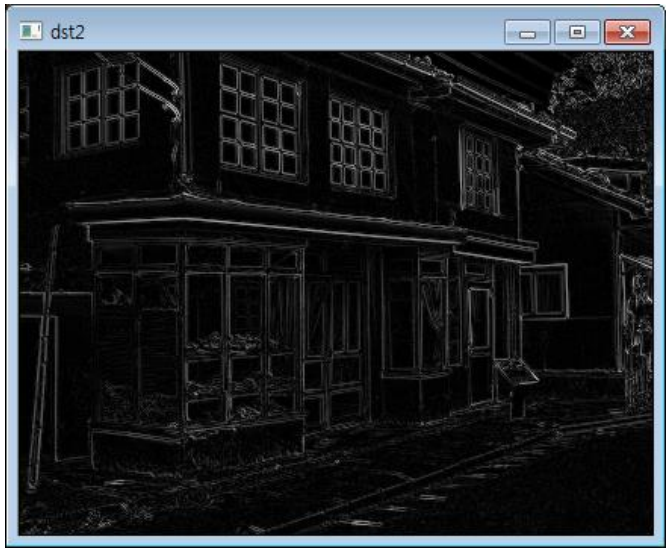
G_x

-1	0	0
0	1	0
0	0	0

G_y

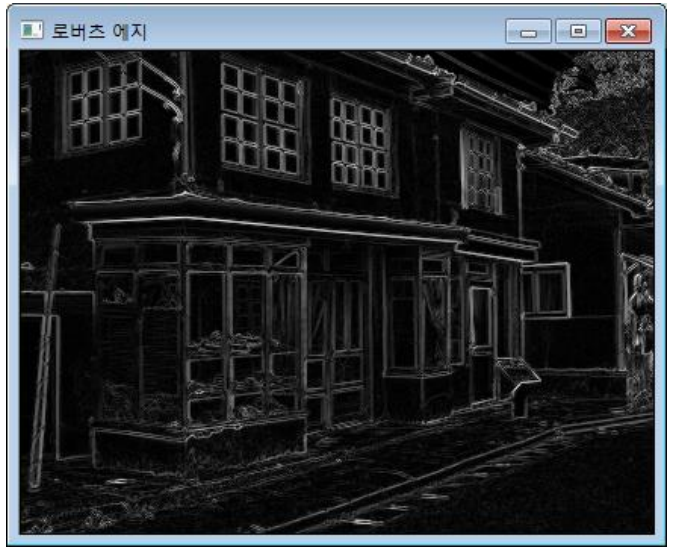
0	0	-1
0	1	0
0	0	0





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

대각선 방향 에지의 크기로 에지 강도 검출



프리윗(Prewitt) 마스크

❖Prewitt filter

- 수직/수평 두 방향의 미분에 대한 크기(magnitude)로 에지 검출

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

G_x

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

G_y

$$\mathrm{Mag}(x, y) = \sqrt{G_x^2 + G_y^2}$$

- 장점1: 로버츠에 비하여 에지의 강도가 강하다.
- 장점2: 수직과 수평 에지를 독립적으로 검출 가능

예제 7.2.4 프리윗 에지 검출 – 04.edge_prewitt.py

```python
import numpy as np, cv2
from Common.filters import filter                    #

def differential(image, data1, data2):
    mask1 = np.array(data1, np.float32).reshape(3, 3)
    mask2 = np.array(data2, np.float32).reshape(3, 3)

    dst1 = filter(image, mask1)                      #
    dst2 = filter(image, mask2)
    dst = cv2.magnitude(dst1, dst2)                  #

    dst = cv2.convertScaleAbs(dst)                   #
    dst1 = cv2.convertScaleAbs(dst1)
    dst2 = cv2.convertScaleAbs(dst2)
    return dst, dst1, dst2

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

data1 = [ -1, 0, 1,                                  # 프리윗 수직 마스크
          -1, 0, 1,          Gx
          -1, 0, 1]
data2 = [ -1,-1,-1,                                  # 프리윗 수평 마스크
           0, 0, 0,          Gy
           1, 1, 1]
dst, dst1, dst2 = differential(image, data1, data2)

cv2.imshow("image", image)
cv2.imshow("prewitt edge", dst)
cv2.imshow("dst1 - vertical mask", dst1)
cv2.imshow("dst2 - horizontal mask", dst2)
cv2.waitKey(0)
```

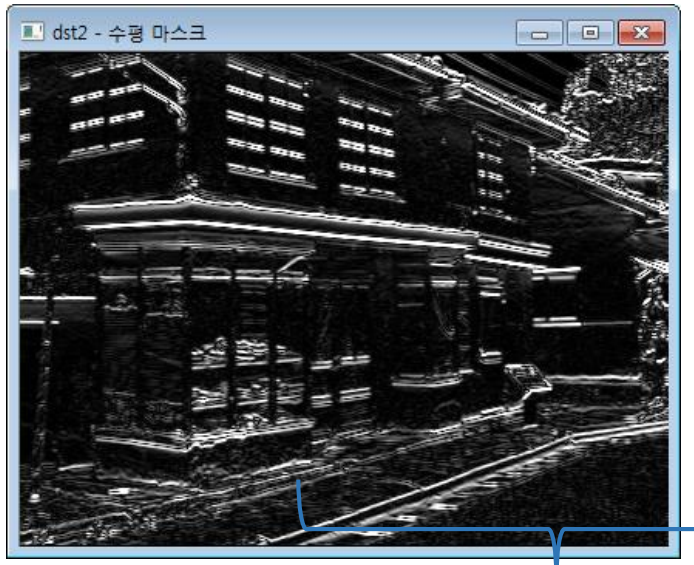
입력영상

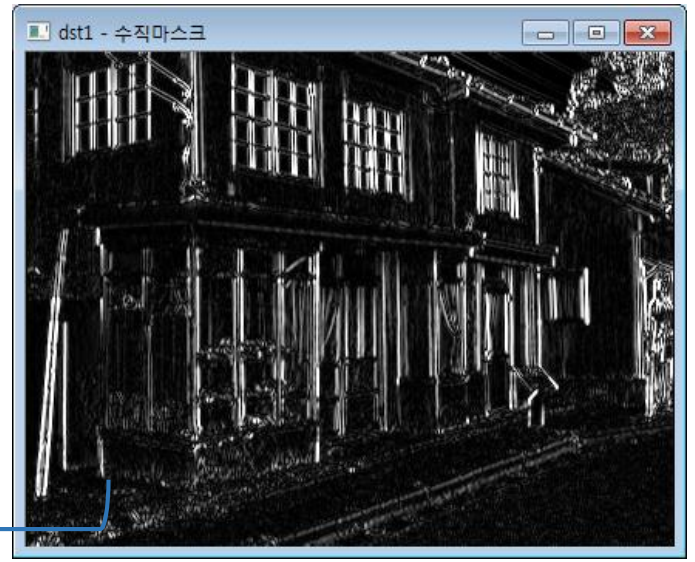
G_x

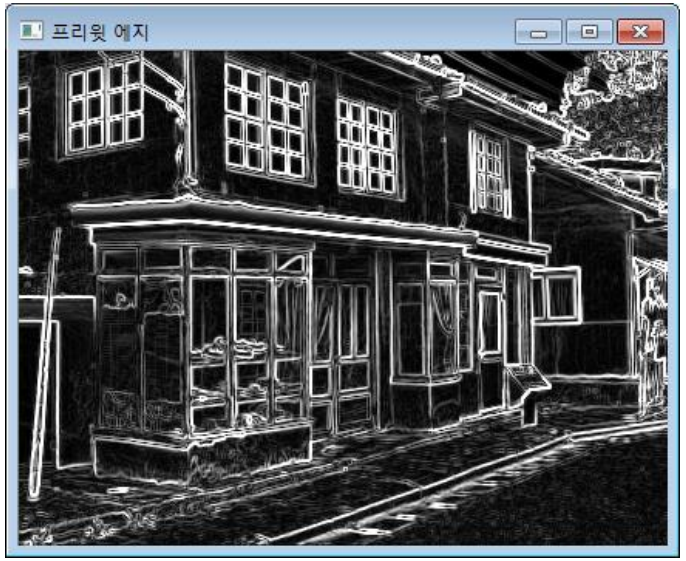
-1	0	1
-1	0	1
-1	0	1

G_y

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







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

수평, 수직 방향 에지의 크기로
에지 강도 검출

소벨(Sobel) 마스크

❖Sobel filter

- 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 7.2.7〉 3x3 크기의 소벨 마스크

예제 7.2.5 소벨 에지 검출 – 05.edge_sobel.py

```
01  import numpy as np, cv2
02  from  Common.filters import differential          # filters 모듈의 differential() 함수 임포트
03
04  image = cv2.imread("images/edge.jpg", cv2.IMREAD_GRAYSCALE)
05  if image is None: raise Exception("영상파일 읽기 오류")
06
07  data1 = [ -1, 0, 1,
08            -2, 0, 2,
09            -1, 0, 1]
10  data2 = [ -1,-2,-1,
11            0, 0, 0,
12            1, 2, 1]
13  dst, dst1, dst2 = differential(image, data1, data2)
```

G_x

G_y

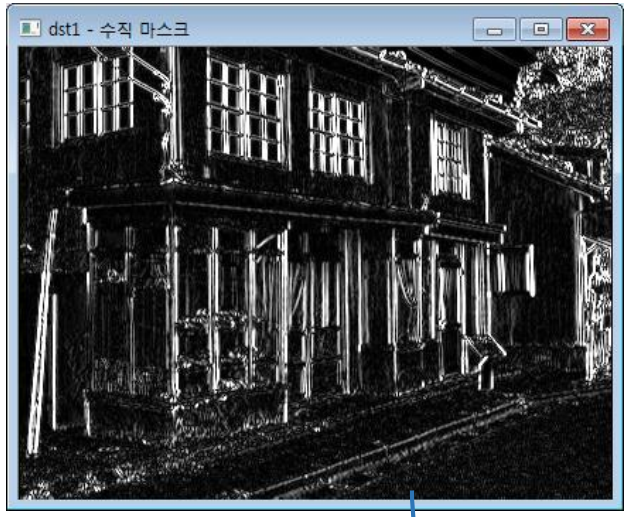
```
15  ## OpenCV 제공 소벨 에지 계산
16  dst3 = cv2.Sobel(np.float32(image), cv2.CV_32F, 1, 0, 3) # x
17  dst4 = cv2.Sobel(np.float32(image), cv2.CV_32F, 0, 1, 3) # y
18  dst3 = cv2.convertScaleAbs(dst3)                          # 절
19  dst4 = cv2.convertScaleAbs(dst4)
20
21  cv2.imshow("dst1- vertical_mask", dst1)
22  cv2.imshow("dst2- horizontal_mask", dst2)
23  cv2.imshow("dst3- vertical_OpenCV", dst3)
24  cv2.imshow("dst4- horizontal_OpenCV", dst4)
25  cv2.waitKey(0)
```

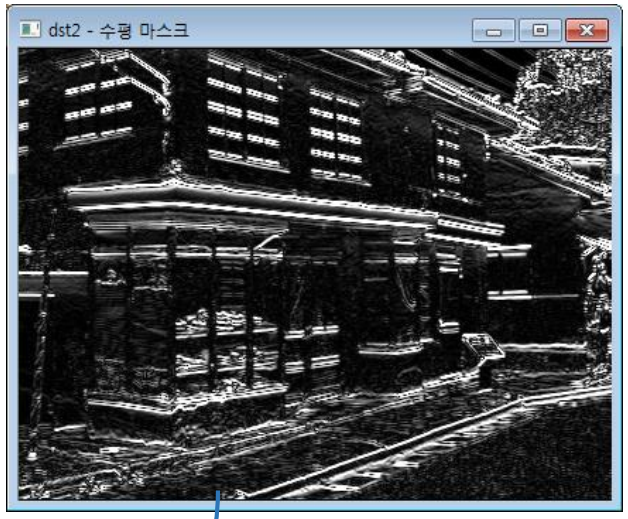
G_x

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

G_y

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





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

수평, 수직 방향 에지의 크기로
에지 강도 검출

cv2.Sobel

```
Sobel (InputArray src,  OutputArray dst,
        int  ddepth,
        int  dx,  int dy,     // x방향, y방향의 미분 차수
        int ksize[1]=3,   // 커널의 크기
        double scale[2]=1, double delta[3]=0,
        int borderType=BORDER_DEFAULT
)
```

Scharr 3x3 커널

$$\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} \qquad \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

1) ksize=-1 이면, Scharr 3x3 커널이 생성되고,
   ksize=1, 3, 5, 7 이면, Sobel 커널이 생성됨
   (ksize=1이면 3x1 or 1x3 커널 생성, ksize≠1이면 ksize x ksize 커널 생성)
2) scale: dst 영상에 저장하기 전에 곱해지는 값
3) delta: dst 영상에 저장하기 전에 더해지는 값

Summary

- ❖회선(convoluttion):
  - 마스크 내의 원소 값과 대응되는 입력 영상의 화소 값들의 곱의 합을 계산하여 출력 화소를 결정
    - 마스크(mask) = 커널(kernel) = 윈도우(window) = 필터(filter)
- ❖블러링(bluring)
  - 부드러운 영상이 되며, 흐려지는 결과가 발생한다.
    - 마스크 원소들의 총합 = 1
- ❖사프닝(sharpening)
  - 선명하고 날까로운 영상을 만드는 방법
    - 마스크 원소들의 총합 = 1

Summary

- 에지(edge)
  - 화소값이 급격하게 변화하는 부분
    - 객체에서 크기, 위치, 모양을 인지할 수 있는 중요한 정보
  - 이웃하는 두 화소의 차분으로 구할 수 있음(미분과 유사)
    - 따라서 미분 마스크로 회선을 수행하여 에지를 검출
  - 1차 미분 마스크
    - 로버츠(Roberts), 프리윗(Prewitt), 소벨(Sobel)