6장. 영상 필터

세부목차

Convolution Filter & Blurring

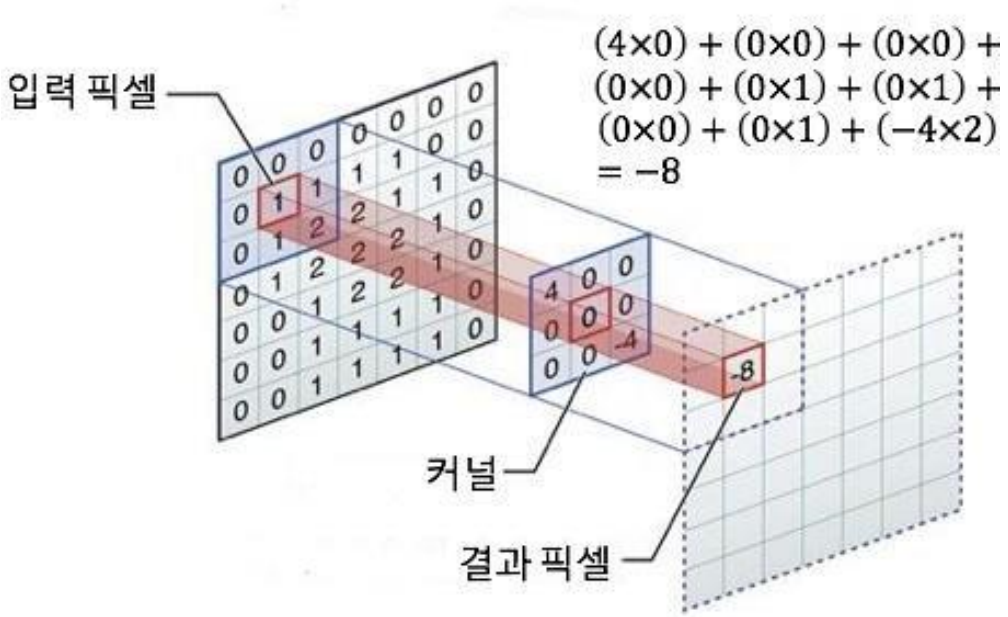
❖ Filter

- 원치 않는 값은 걸러내고 원하는 값만 얻는다.
  - LPF(Low Pass Filter) : Blurring, Noise 제거
  - HPF(High Pass Filter) : Sharpening, Edge 검출
- Domain
  - 공간 영역 필터(Spacial Domain Filter)
    - 픽셀 연산 대상을 하나가 아닌 주변 픽셀 값을 활용해서 연산
    - 컨볼루션(Convolution) 연산
  - 주파수 영역 필터(Frequency Domain Filter)
    - 픽셀 값들의 차이를 주파수로 변환해서 연산
    - 푸리에 변환(Fourier Transform)

Convolution Filter

❖ Convolution

- 공간 영역 필터의 핵심 연산
- 커널의 각 요소와 대응하는 입력 픽셀 값을 곱해서 모두 합하는 것
- Kernel : 연산에 활용할 주변 픽셀 대상 선정
  - window, mask, filter 등의 이름으로 혼용
  - 커널 크기: n x n
  - n : 일반적으로 홀수



[그림 6-1] 컨볼루션 연산

Convolution Filter

❖ Convolution 연산 적용 함수

- dst = cv2.filter2D(src, ddepth, kernel[, dst, anchor, delta, borderType])
  - src : 입력 영상, NumPy 배열
  - ddepth : 출력 영상의 dtype,
    - -1 : 입력 영상과 동일
    - CV_8U, CV16U/CV16S, CV_32F, CV_64F
  - kernel : 컨볼루션 커, float32
  - dst : 결과 영상, NumPy 배열
  - anchor : 커널의 기준점, default: 중심점(-1,-1)
  - delta : 필터 적용된 결과에 추가할 값
  - borderType : 외곽 픽셀 보정 방법 지정

세부목차

Blurring

❖ Blurring

- Filter를 이용해서 영상을 흐릿하게 만든다
- Noise 제거에 탁월
- Blur 종류
  - Averaging Blur
    - cv2.blur, cv2.boxFilter
  - Gassian Blur
    - cv2.GaussianBlur
  - Median Blur
    - cv2.medianBlur
  - Bilateral Filter
    - cv2.bilateralFilter

Blurring

❖ Averaging Blurring

- 각 픽셀의 값을 주변 요소들과 평균으로 변경
- 주변 요소 영역 결정
  - Kernel matrix 생성
  - Kernel이 클수록 흐릿해짐
- 방법
  - cv2.filter2D()
  - cv2.boxFilter()
  - cv2.blur()

Blurring

❖ Averaging Blurring

- 주변 픽셀의 평균을 대상 픽셀에 적용
- 고유의 값을 평균화 해서 흐릿한 효과
- 5 X 5 평균 필터

$$k = \begin{bmatrix} 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 & 0.4 & 0.4 \end{bmatrix} \qquad k = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Blurring

❖ Averaging Blurring

- cv2.filter2D Example

```
import cv2
import numpy as np
img = cv2.imread('../img/girl.jpg')
'''
#5x5 평균 필터 커널 생성 ---①
kernel = np.array([[0.04, 0.04, 0.04, 0.04, 0.04],
                   [0.04, 0.04, 0.04, 0.04, 0.04],
                   [0.04, 0.04, 0.04, 0.04, 0.04],
                   [0.04, 0.04, 0.04, 0.04, 0.04],
                   [0.04, 0.04, 0.04, 0.04, 0.04]])
'''
# 5x5 평균 필터 커널 생성 ---②
kernel = np.ones((5,5))/5**2
# 필터 적용 ---③
blured = cv2.filter2D(img, -1, kernel)
# 결과 출력
cv2.imshow('origin', img)
cv2.imshow('avrg blur', blured)
cv2.waitKey()
cv2.destroyAllWindows()
```

[예제 6-2] 평균 필터를 생성해서불러 적용(blur_avg_kernel.py)

Blurring

❖ Averaging Blurring

- Example <결과>

[그림 6-1] [예제 6-1]의 실행 결과

Blurring

❖ Averaging Blurring

- 커널 정의 없이 블러링 적용
- dst = cv2.blur(src, ksize[, dst, anchor, borderType])
  - src : 입력 영상, NumPy 배열
  - ksize : 커널의 크기
  - 나머지 인자는 cv2.filter2D()와 동일
- dst = cv2.boxFilter(src, ddepth, ksize[, dst, anchor, normalize, borderType])
  - src : 입력 영상, NumPy 배열
  - ddepth : 출력 영상의 dtype, -1 : 입력 영상과 동일
  - normalize : 커널 크기로 정규화 지정 여부 , 불리언
  - 나머지 인자는 cv2.filter2D()와 동일

Blurring

❖ Averaging Blurring

- 블러링 전용 함수 Example

```
import cv2
import numpy as np

file_name = '../img/taekwonv1.jpg'
img = cv2.imread(file_name)

# blur() 함수로 블러링 ---①
blur1 = cv2.blur(img, (10,10))
# boxFilter() 함수로 블러링 적용 ---②
blur2 = cv2.boxFilter(img, -1, (10,10))

# 결과 출력
merged = np.hstack( (img, blur1, blur2))
cv2.imshow('blur', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-2] 블러 전용 함수로 블러링 적용(blur_avg_api.py)

Blurring

❖ Averaging Blurring

- 블러링 전용 함수 결과

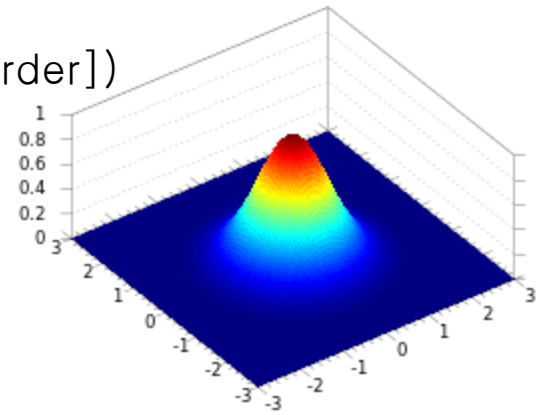
[그림 6-3] [예제 6-2]의 실행 결과

Blurring

❖ Gaussian Blurring

- 가우시안 함수를 적용한 커널 사용
- 백색 노이즈에 효과적
- cv2.GaussianBlur(src, ksize, sigmaX [, sigmaY, border])
  - src : 입력 이미지
  - ksize : 커널 크기
  - sigmaX : X 방향 표준편차, 0=auto
  - sigmaY : Y 방향 표준편차, default = sigmaX
  - border : 테두리 보정 방식
- $\sigma = 0.3((\text{ksize}-1)0.5-1)+0.8$
- kernel = cv2.getGaussianKernel(ksize, sigma)
  - ksize : 커널 크기
  - sigma : 표준 편차
  - kernel * kernel.T

[그림 6-1] 관심영역 표시



$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Blurring

❖ Gaussian Blurring

- Example

```
import cv2
import numpy as np
img = cv2.imread('../img/gaussian_noise.jpg')
# 가우시안 커널을 직접 생성해서 블러링 ---①
k1 = np.array([[1, 2, 1],
               [2, 4, 2],
               [1, 2, 1]]) *(1/16)
blur1 = cv2.filter2D(img, -1, k1)
# 가우시안 커널을 API로 얻어서 블러링 ---②
k2 = cv2.getGaussianKernel(3, 0)
blur2 = cv2.filter2D(img, -1, k2*k2.T)
# 가우시안 블러 API로 블러링 ---③
blur3 = cv2.GaussianBlur(img, (3, 3), 0)
# 결과 출력
print('k1:', k1)
print('k2:', k2*k2.T)
merged = np.hstack((img, blur1, blur2, blur3))
cv2.imshow('gaussian blur', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

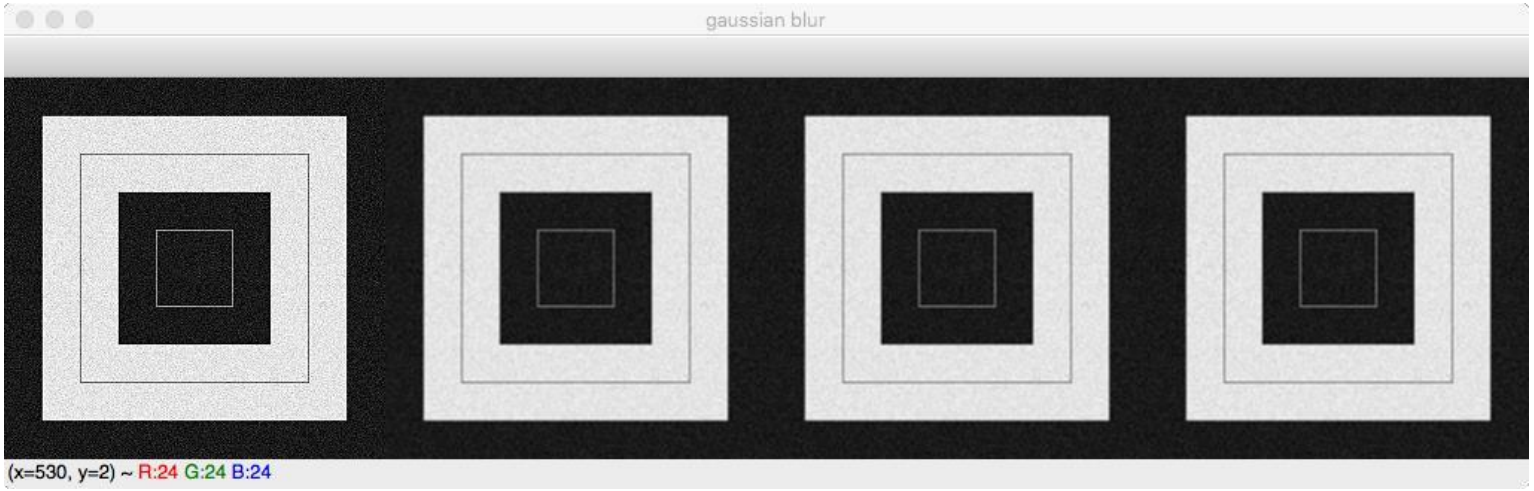
[예제 6-3] 가우시안 블러(blur_gaussian.py)

Blurring

❖ Gaussian Blurring

- Example <실행결과>

```
k1: [[0.0625  0.125   0.0625]
 [0.125   0.25    0.125 ]
 [0.0625  0.125   0.0625]]
k2: [[0.0625  0.125   0.0625]
 [0.125   0.25    0.125 ]
 [0.0625  0.125   0.0625]]
```



[그림 6-5] [예제 6-3]의 실행 결과

Blurring

❖ Median Blurring

- 커널 영역 픽셀 값중 중간 값 적용
- salt-and-pepper noise(점잡음)에 효과적
- 새로운 값이 아닌 기존 픽셀의 값을 재활용
- dst = cv2.medianBlur(src, ksize)
  - src : 입력 영상, NumPy 배열
  - ksize : 커널 크기

Blurring

❖ Median Blurring

[예제 6-4] 미디언 블러링(blur_median.py)

```
import cv2
import numpy as np

img = cv2.imread("../img/salt_pepper_noise.jpg")

# 미디언 블러 적용 --- ①
blur = cv2.medianBlur(img, 5)

# 결과 출력
merged = np.hstack((img,blur))
cv2.imshow('media', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[그림 6-6] [예제 6-4]의 실행 결과

Blurring

❖ Bilateral Filter Blurring

- 양방향 필터, 2개의 필터 사용
  - 가우시안 필터 + 경계 필터
- Blur 결과는 Edge도 흐리게 만드는데, 이것을 개선
- 연산 시간 길어서 속도 느림
- dst = cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[,dst, borderType])
  - src : 입력 영상, NumPy 배열
  - d : 필터의 직경(diameter), 5보다 크면 매우 느림
  - sigmaColor : 색공간 필터의 시그마 값, **값이 클 수록 이웃한 픽셀과 기준색상의 영향이 커진다.**
  - sigmaSpace : 좌표 공간의 시그마 값, **값이 클 수록 주변 픽셀에 미치는 영향이 커진다.**
    - 단순한 사용을 위해 sigmaColor와 sigmaSpace 에 같은 값을 사용 권장
    - 10 ~ 150 범위의 값 사용을 권장

Blurring

❖ **Bilateral Filter Blurring**

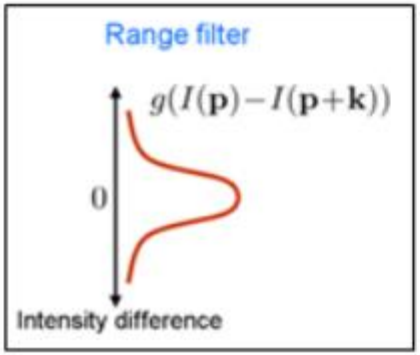
W_p : normalization factor, $W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$

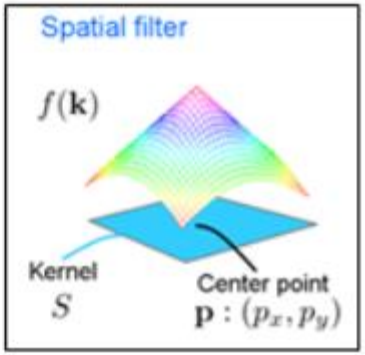
$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|),$$

$I(x_i)$: pixel value

226	240	239	65	53
230	254	240	74	64
215	220	240	77	53
220	243	255	76	36
211	222	240	72	54

Kernel center(Ω) & neighbor pixels(x_i)





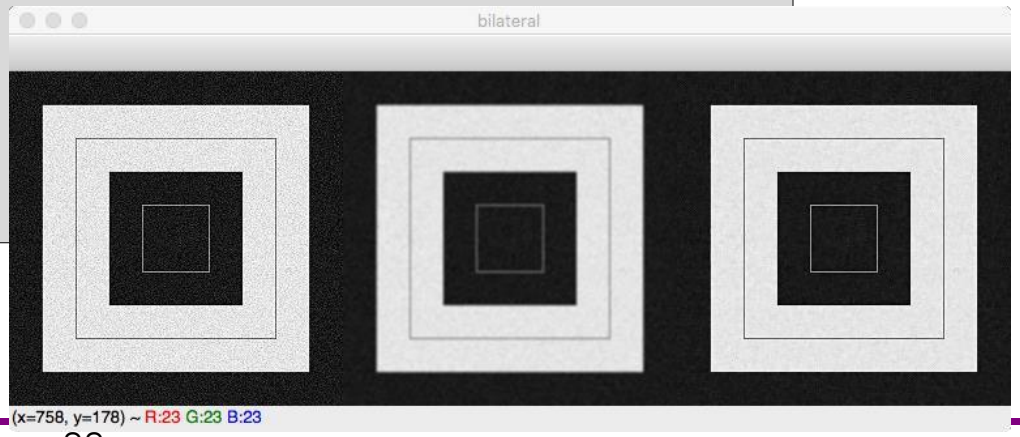
Blurring

❖ Bilateral Filter Blurring

- Example

[예제 6-5] 바이레터럴 필터와 가우시안 필터 비교(blur_bilateral.py)

```
import cv2
import numpy as np
img = cv2.imread("../img/gaussian_noise.jpg")
# 가우시안 필터 적용 ---①
blur1 = cv2.GaussianBlur(img, (5,5), 0)
# 바이레터럴 필터 적용 ---②
blur2 = cv2.bilateralFilter(img, 5, 75, 75)
# 결과 출력
merged = np.hstack((img, blur1, blur2))
cv2.imshow('bilateral', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



[그림 6-7] [예제 6-5]의 실행 결과

세부목차

1. Convolution Filter
2. Blurring
3. **Edge Detection**
4. Morphology
5. Image Pyramids
6. Workshop

Edge Detecting

❖ 경계 검출

- 전경과 배경을 분리하는 기초적인 작업
- Sharping : 경계를 검출해서 강조하는 것
- 경계 : 픽셀 값의 변화가 큰 지점
- 미분 연산
  - 화소의 변화율
  - 이미지 경사도

Continuous function:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h}$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{h \to 0} \frac{f(x,y+h) - f(x,y)}{h}$$

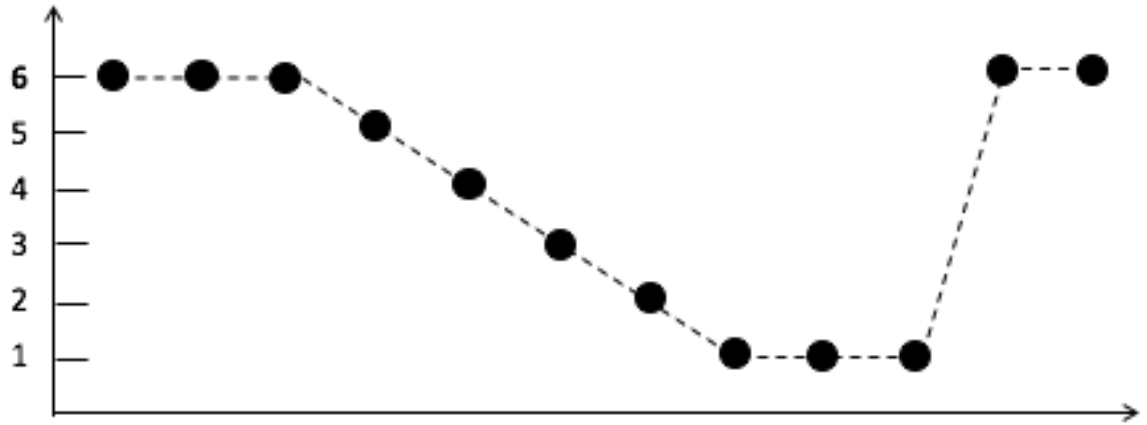
Edge Detecting

❖ 기본 미분 필터

- 미분 공식 간소화
  - 연속된 공간이 아니므로
  - 이산화 후 근사 계산
  - 다음 픽셀 – 현재 픽셀
  - 2방향의 컨볼루션 커널

- $Gx = \frac{\partial f(x,y)}{\partial x} \approx f_{x+1,y} - f_{x,y}$
- $Gy = \frac{\partial f(x,y)}{\partial y} \approx f_{x,y+1} - f_{x,y}$

- $Gx = \begin{bmatrix} -1 & 1 \end{bmatrix}$
- $Gy = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

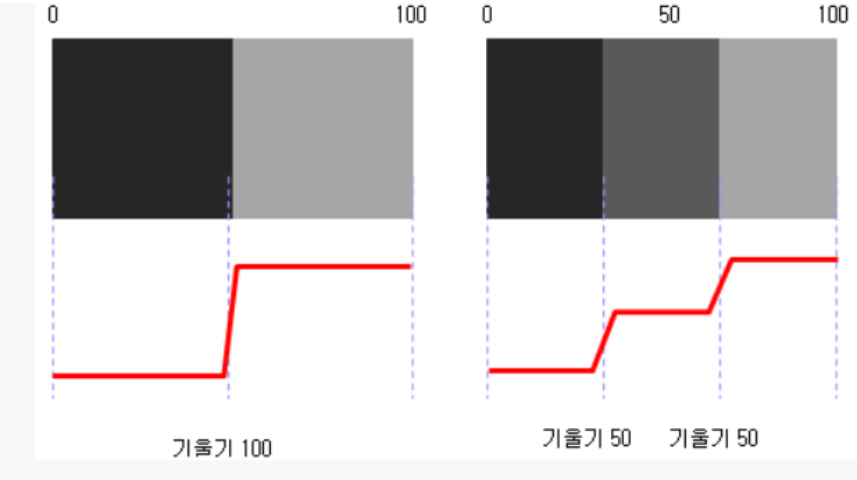


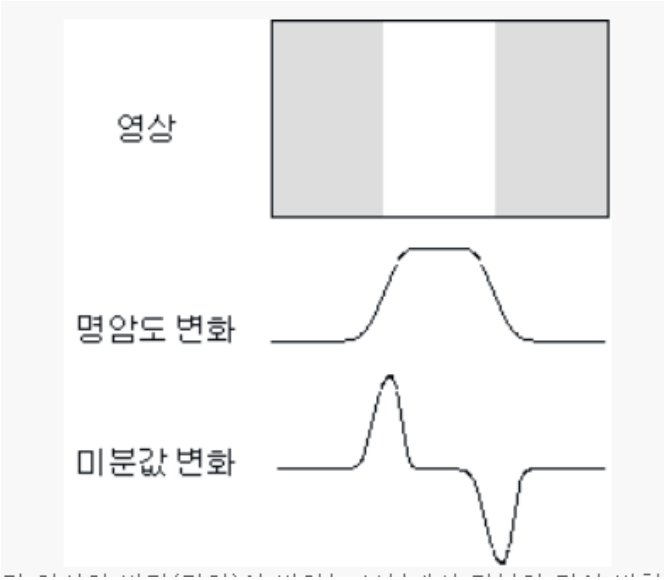
픽셀 값	6	6	6	5	4	3	2	1	1	1	6	6
1차 미분	0	0	0	-1	-1	-1	-1	0	0	5	0	
2차 미분	0	0	0	-1	0	0	1	0	5	-5	0	

[그림 6-8] 픽셀에 대한 미분 연산

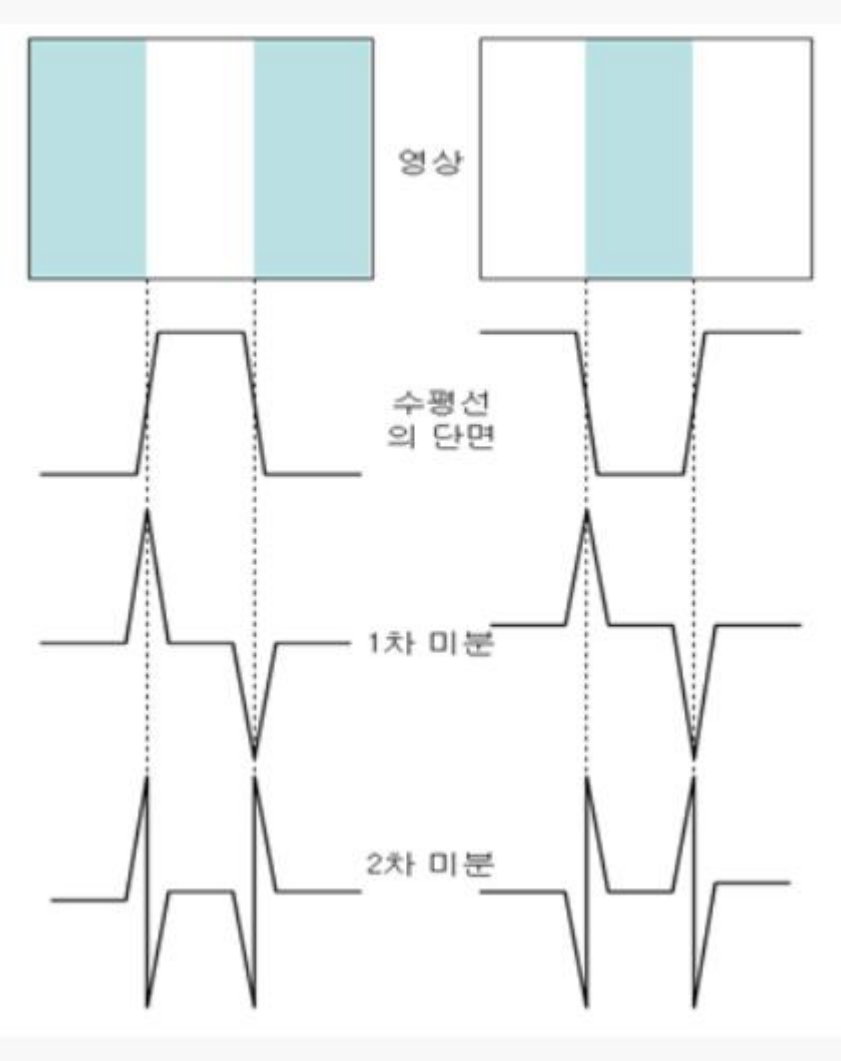
Edge Detecting

1차 미분





Edge Detecting



Edge Detecting

❖ 기본 미분 필터

- kernel [-1, 1] Example

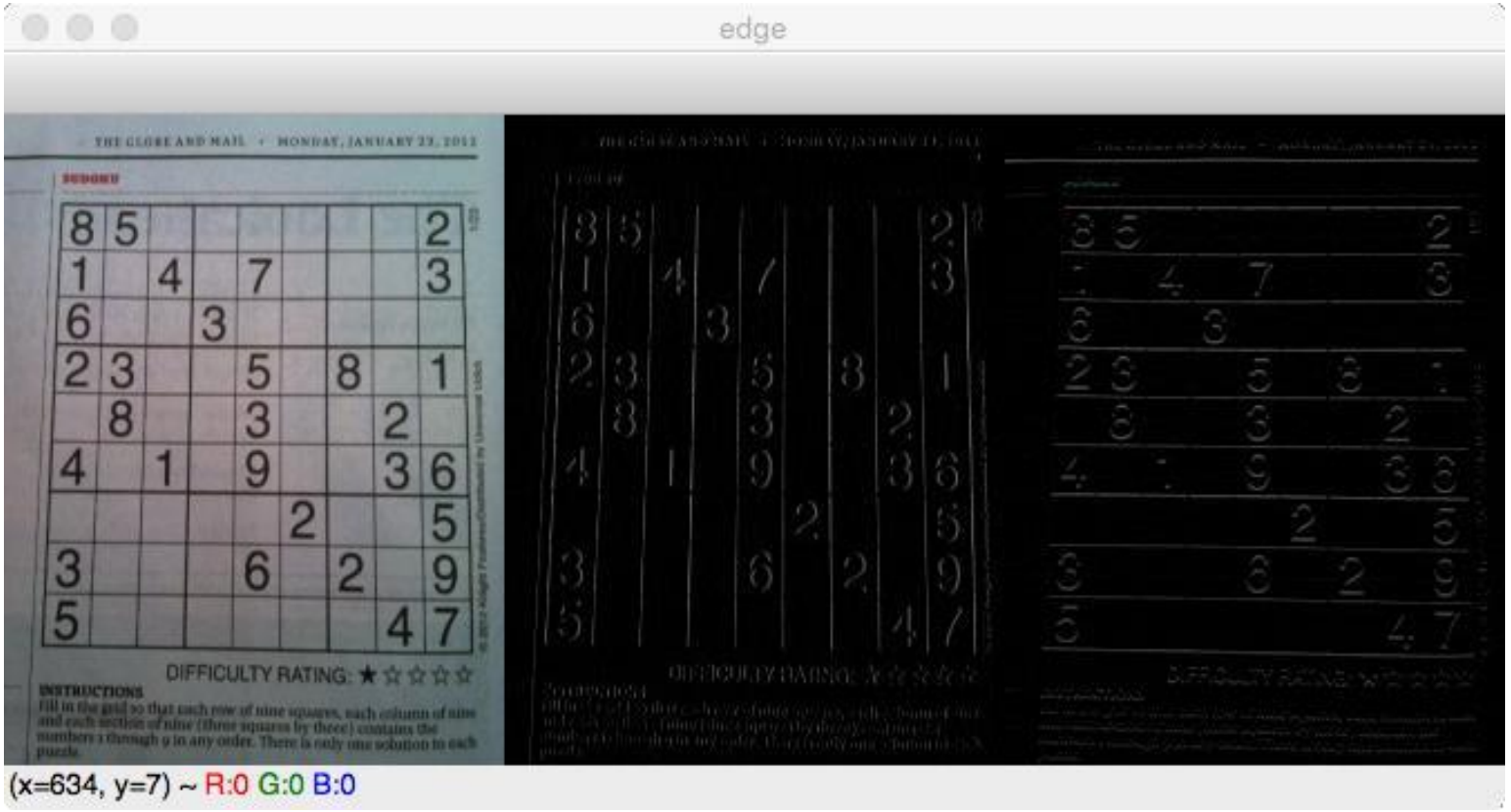
```
import cv2
import numpy as np
img = cv2.imread("../img/sudoku.jpg")
#미분 커널 생성 ---①
gx_kernel = np.array([[ -1, 1]])
gy_kernel = np.array([[ -1],[ 1]])
# 필터 적용 ---②
edge_gx = cv2.filter2D(img, -1, gx_kernel)
edge_gy = cv2.filter2D(img, -1, gy_kernel)
# 결과 출력
merged = np.hstack((img, edge_gx, edge_gy))
cv2.imshow('edge', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-6] 미분 커널로 경계 검출(edge _differential.py)

Edge Detecting

❖ 기본 미분 필터

- kernel [−1, 1] Example < 결과 >



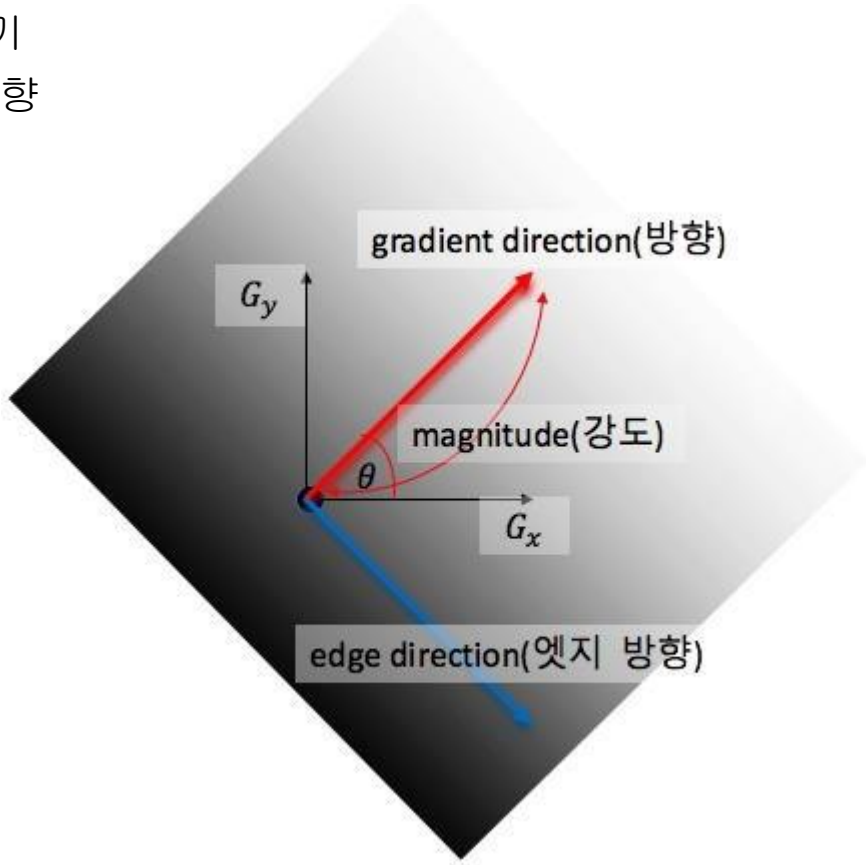
[그림 6-9] [예제 6-6]의 실행 결과

Edge Detecting

❖ 엣지의 방향과 강도

- 그레이디언트(Gradient)
  - 강도(Magnitude) : 변화의 세기
  - 방향(Direction) : 값의 변화 방향
    - 엣지 방향과 수직
- 영상 특징 벡터화에 사용

- $\text{magnitude} = \sqrt{G_x^2 + G_y^2}$
- $\text{direction}(\theta) = \arctan(\frac{G_y}{G_x})$



[그림 6-10] 그레이디언트의 강도와 방향

Edge Detecting

❖ Roberts cross operator

- Lawrence Roberts 제안(1963)
- 기본 미분 커널 개선한 것 중 가장 오래된 것
- 대각선 방향으로 1과 -1 배치
- 기본 미분 마스크에 비해 사선 검출 효과
- 노이즈 민감
- 엣지 강도 약함

- $Gx = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}$

- $Gx = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}$

Edge Detecting

❖ Roberts cross operator

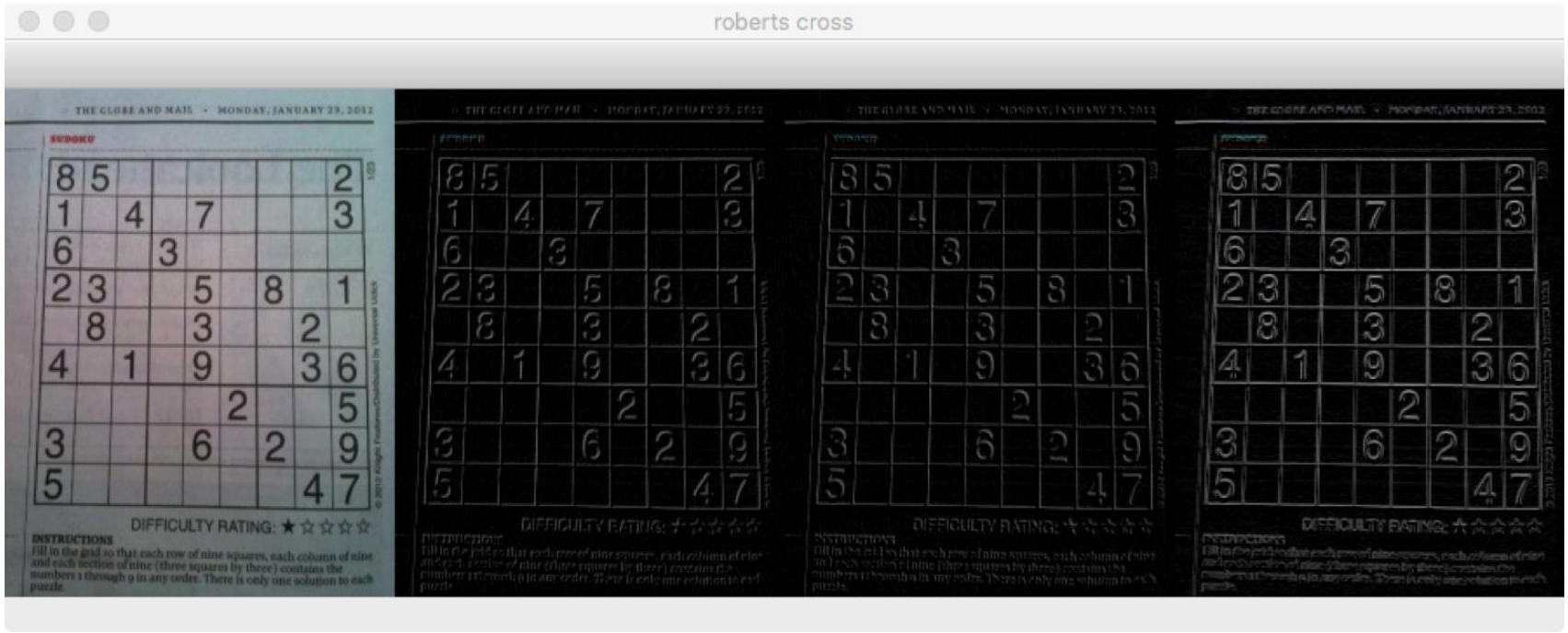
- Example

```
import cv2
import numpy as np
img = cv2.imread("../img/sudoku.jpg")
# 로버츠 커널 생성 ---①
gx_kernel = np.array([[1,0], [0,-1]])
gy_kernel = np.array([[0, 1],[-1,0]])
# 커널 적용 ---②
edge_gx = cv2.filter2D(img, -1, gx_kernel)
edge_gy = cv2.filter2D(img, -1, gy_kernel)
# 결과 출력
merged = np.hstack((img, edge_gx, edge_gy, edge_gx+edge_gy))
cv2.imshow('roberts cross', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-7] 로버츠 마스크를 적용한 경계 검출(edge_roberts.py)

Edge Detecting

- **Roberts cross operator**
  - Example < 결과 >



[그림 6-11] [예제 6-7]의 실행 결과

Edge Detecting

❖ Prewitt Operator

- Judith M. S. Prewitt 제안
- 각 방향 3번의 차분
- 엣지 강도 강함
- 수직과 수평 엣지 동등
- 대각선 검출 약함

- $Gx = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$

- $Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$

Edge Detecting

❖ Prewitt Operator

- Example

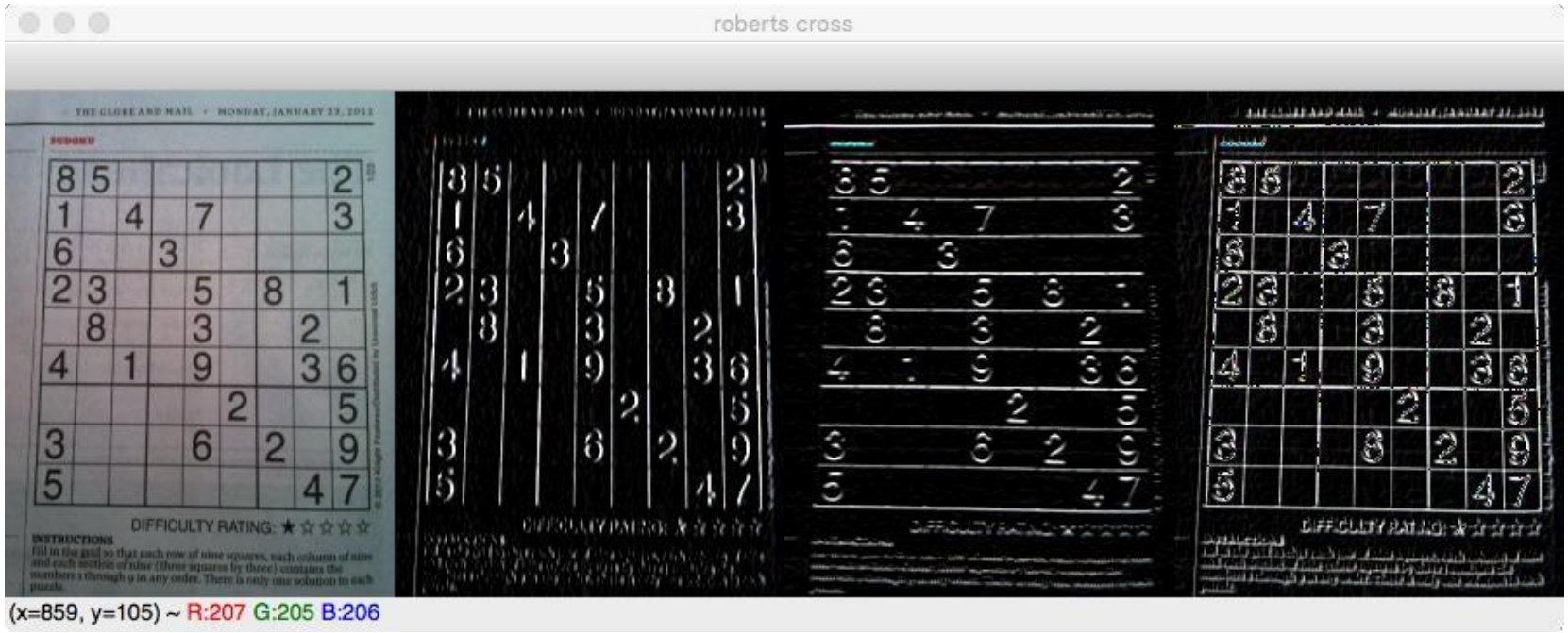
```
import cv2
import numpy as np
file_name = "../img/sudoku.jpg"
img = cv2.imread(file_name)
# 프리윗 커널 생성
gx_k = np.array([[-1,0,1], [-1,0,1],[-1,0,1]])
gy_k = np.array([[-1,-1,-1],[0,0,0], [1,1,1]])
# 프리윗 커널 필터 적용
edge_gx = cv2.filter2D(img, -1, gx_k)
edge_gy = cv2.filter2D(img, -1, gy_k)
# 결과 출력
merged = np.hstack((img, edge_gx, edge_gy, edge_gx+edge_gy))
cv2.imshow('prewitt', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-7] 프리위 마스크를 적용한 경계 검출(edge_prewitt.py)

Edge Detecting

❖ Prewitt Operator

- Example <결과>



[그림 6-12] [예제 6-8]의 실행 결과

Edge Detecting

❖ Sobel Operator

- Irwin Sobel 제안(1968)
- 1차 미분 오퍼레이터 중 가장 대표적
- 중심화소의 차분 비중 2배
- 수평, 수직, 대각선 엣지 모두 강함

- $Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$
- $Gx = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$

- dst = cv2.Sobel(src, ddepth, dx, dy[, dst, ksize, scale, delta, borderType])
  - src : 입력 영상, NumPy 배열
  - ddepth : 출력 영상의 dtype, -1 : 입력영상과 동일
  - dx, dy : 미분 차수, 0,1,2 중 선택, 둘 다 0일 수 없슴
  - ksize : 커널의 크기, 1,3,5,7 중 선택
  - scale : 미분에 사용할 계수
  - delta : 연산 결과에 가산 할 값

Edge Detecting

❖ Sobel Operator

- Example

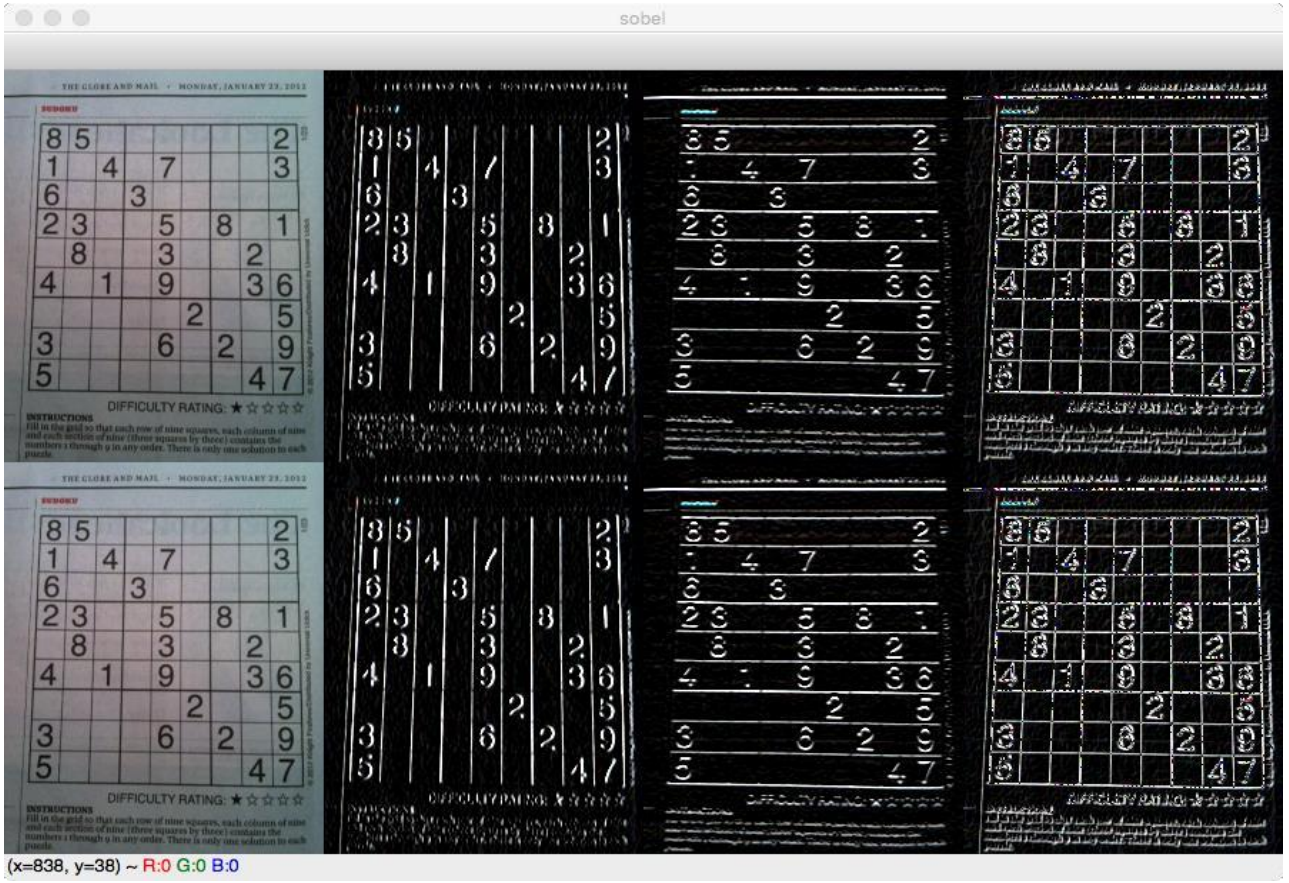
```
import cv2
import numpy as np
img = cv2.imread("../img/sudoku.jpg")
# 소벨 커널을 직접 생성해서 엣지 검출 ---①
## 소벨 커널 생성
gx_k = np.array([[-1,0,1], [-2,0,2],[-1,0,1]])
gy_k = np.array([[-1,-2,-1],[0,0,0], [1,2,1]])
## 소벨 필터 적용
edge_gx = cv2.filter2D(img, -1, gx_k)
edge_gy = cv2.filter2D(img, -1, gy_k)
# 소벨 API를 생성해서 엣지 검출
sobelx = cv2.Sobel(img, -1, 1, 0, ksize=3)
sobely = cv2.Sobel(img, -1, 0, 1, ksize=3)
# 결과 출력
merged1 = np.hstack((img, edge_gx, edge_gy, edge_gx+edge_gy))
merged2 = np.hstack((img, sobelx, sobely, sobelx+sobely))
merged = np.vstack((merged1, merged2))
cv2.imshow('sobel', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-9] 소벨 마스크를 적용한 경계 검출(edge_sobel .py)

Edge Detecting

❖ Sobel Operator

- Example <결과>



[그림 6-13] [예제 6-9]의 실행 결과

Edge Detecting

❖ Scharr Operator

- Sobel의 문제점
  - 커널 크기가 작거나 크더라도 중심에서 멀어 질 수록 엣지 방향성의 정확도 감소
- Sobel의 문제 개선

$$\bullet\ Gx = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$

$$\bullet\ Gx = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

- dst = cv2.Scharr(src, ddepth, dx, dy[, dst, scale, delta, borderType])
  - 함수의 인자는 ksize가 없다는 것을 제외하면 cv2.Sobel() 과 동일

Edge Detecting

❖ Scharr Operator

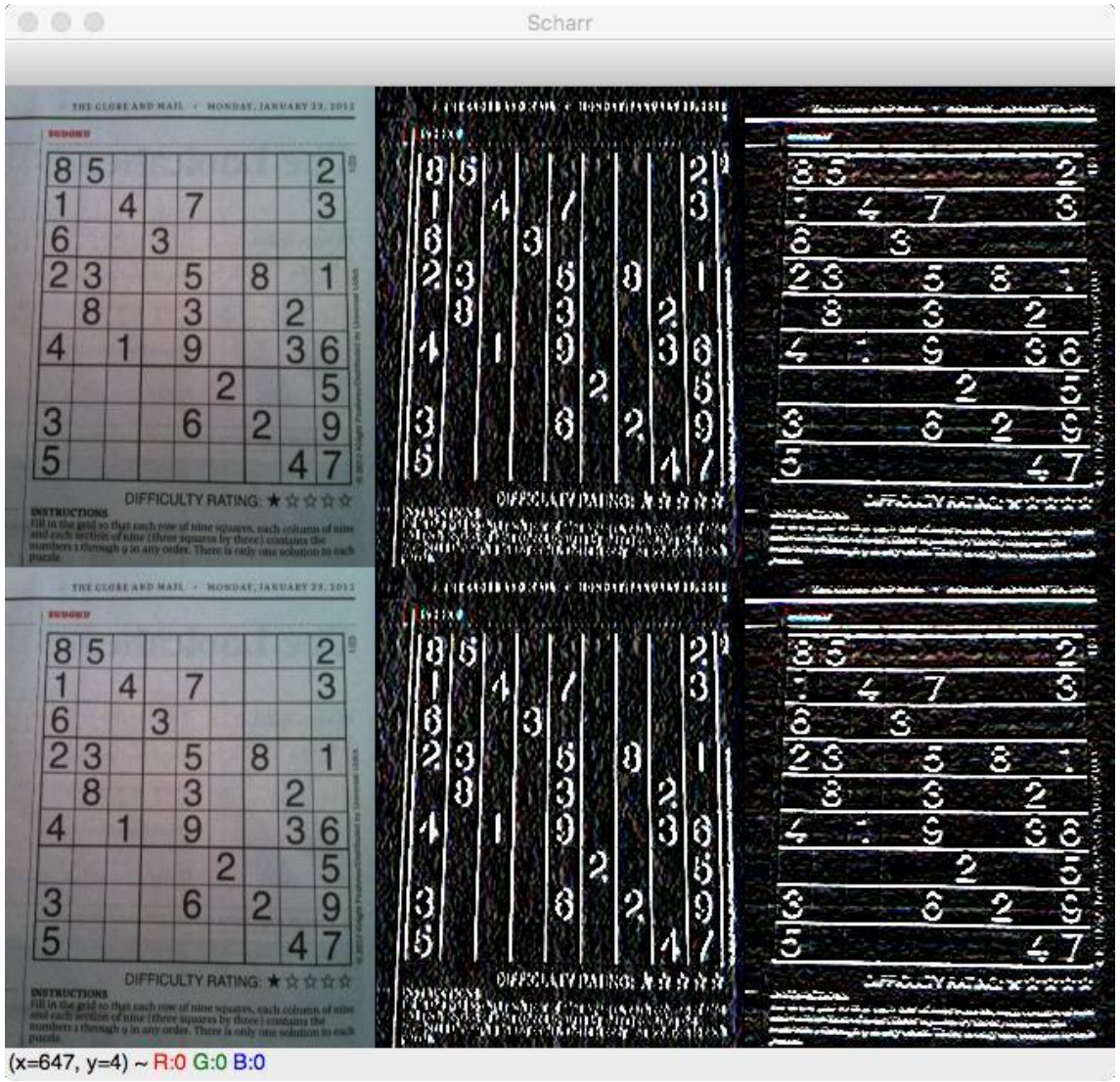
- Example

```
mport cv2
import numpy as np
img = cv2.imread("../img/sudoku.jpg")
# 샤르 커널을 직접 생성해서 엣지 검출 ---①
gx_k = np.array([[-3,0,3], [-10,0,10],[-3,0,3]])
gy_k = np.array([[-3,-10,-3],[0,0,0], [3,10,3]])
edge_gx = cv2.filter2D(img, -1, gx_k)
edge_gy = cv2.filter2D(img, -1, gy_k)
# 샤르 API로 엣지 검출 ---②
scharrx = cv2.Scharr(img, -1, 1, 0)
scharry = cv2.Scharr(img, -1, 0, 1)
# 결과 출력
merged1 = np.hstack((img, edge_gx, edge_gy))
merged2 = np.hstack((img, scharrx, scharry))
merged = np.vstack((merged1, merged2))
cv2.imshow('Scharr', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-10] 샤르 마스크를 적용한 검출(edge_scharr.py)

Edge Detecting

- **Scharr Operator**
  - Example <결과>



[그림 6-14] [예제 6-10]의 실행 결과

Edge Detecting

❖ Laplacian Operator

- Pierre-Simon de Laplace(1749-1827)
- 대표적 2차 미분 오퍼레이터
- 1차 미분의 점진적 변화 엣지 검출 문제 해결

$$\frac{d^2 f}{dy^2} = \frac{df(x, y+1)}{dy} - \frac{df(x, y+1)}{dy}$$
$$= [f(x, y+1) - f(x, y)] - [f(x, y) - f(x, y-1)]$$
$$= f(x, y+1) - 2f(x, y) + f(x, y-1)$$

$$kernel = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- cv2.Laplacian(src, ddepth)
  - 함수인자는 cv2.Sobel() 과 동일

Edge Detecting

❖ Laplacian Operator

- Example

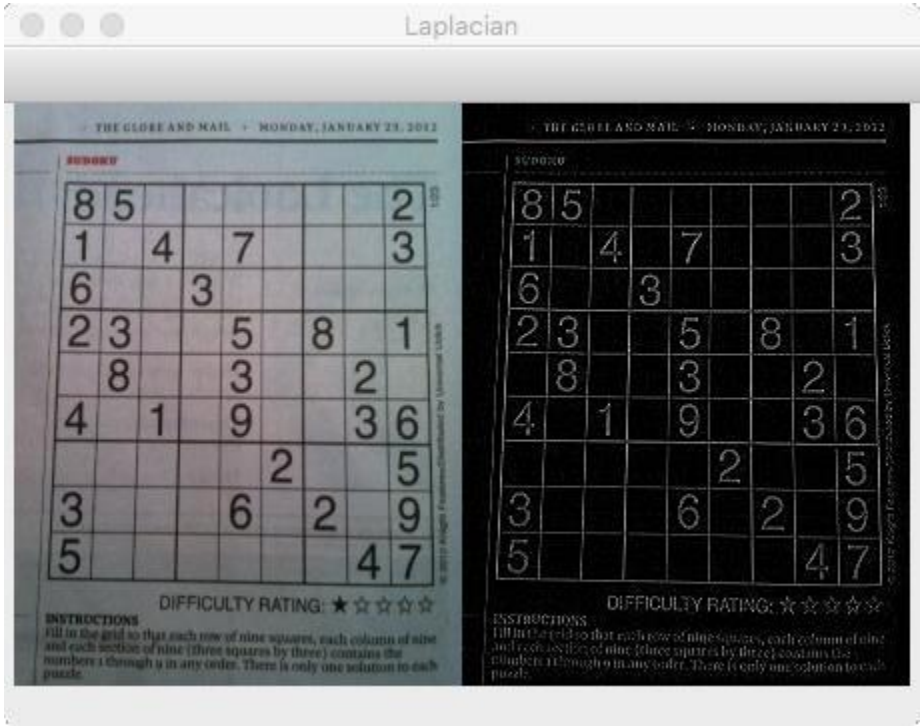
```
import cv2
import numpy as np

img = cv2.imread("../img/sudoku.jpg")

# 라플라시안 필터 적용 ---①
edge = cv2.Laplacian(img, -1)

# 결과 출력
merged = np.hstack((img, edge))
cv2.imshow('Laplacian', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-2] 라플라시안 마스크를 적용한 경계 검출(edge_laplacian.py)

Edge Detecting

❖ Laplacian Operator

- Example <결과>

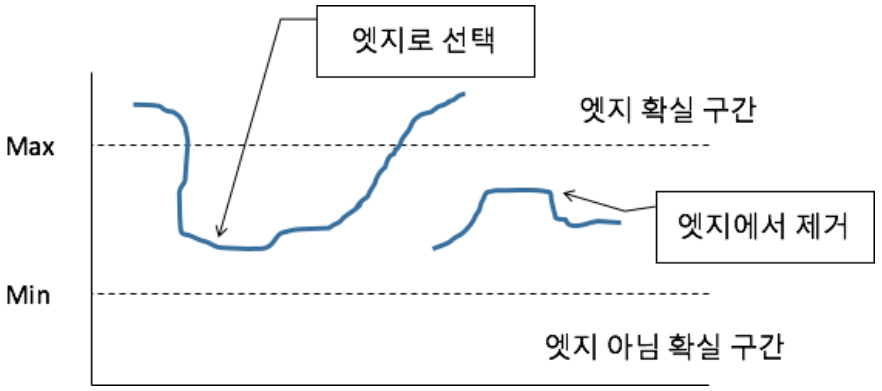


[그림 6-15] [예제 6-11]의 실행 결과

Edge Detecting

❖ Canny Edge Dectector

- John F Canny 제안(1986)
- 1개 아닌 4단계 알고리즘
  1. Noise Reduction
     - 5x5 Gaussian Filter
  2. Edge Gradient Detection
     - Sobel로 Gradient의 방향 각도 계산
  3. Non-Maximum Suppression
     - Gradient 방향에서 검출된 엣지 중에 가장 큰 값만 선택 나머지 제거
  4. Hysteresis Thresholding
     - 임계값 Max와 Min 설정
     - 임계값 범위 엣지 검사
     - Max 보다 큰 값과 연결 없으면 버림



[그림 6-16] 이력 스레시홀딩 사례

Edge Detecting

❖ Canny Edge Dectector

- edges = cv2.Canny(img, threshold1, threshold2 [,edges, apertureSize, L2gardient])
  - img : 입력 영상, NumPy 배열
  - threshold1, threshold2 : 이력 쓰레시홀딩에 사용할 최소, 최대 값
  - apertureSize : 소벨 마스크에 사용할 커널 크기
  - L2gradient : 그레이디언트 강도를 구할 방식 지정 플래그
    - True : $\sqrt{G_x^2 + G_y^2}$
    - False : $|G_x| + |G_y|$
  - egges : 엣지 결과값을 같는 2차원 배열

Edge Detecting

❖ Canny Edge Detector

- Example

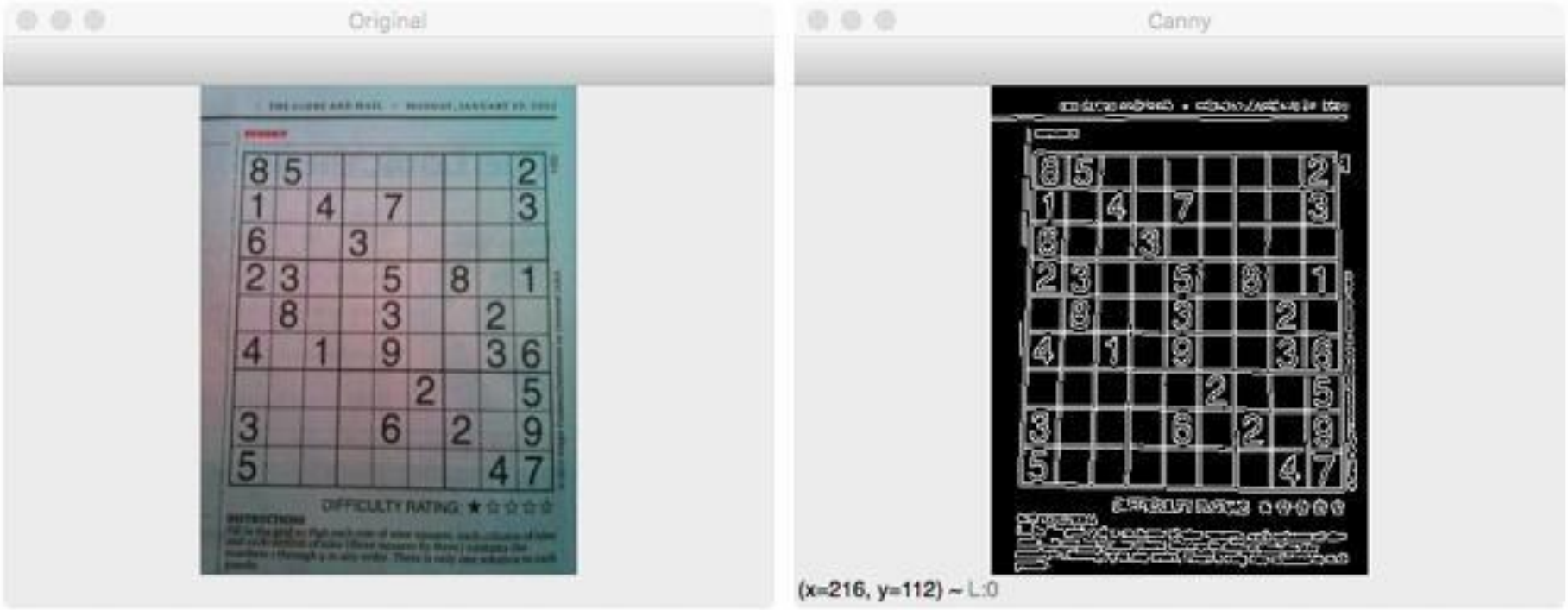
```
import cv2, time
import numpy as np

img = cv2.imread("../img/sudoku.jpg")

# 케니 엣지 적용
edges = cv2.Canny(img,100,200)

# 결과 출력
cv2.imshow('Original', img)
cv2.imshow('Canny', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-12] 캐니 엣지 검출(edge_canny.py)

Edge Detecting

❖ Canny Edge Detector

- Example <결과>



[그림 6-17] [예제 6-12]의 실행 결과

세부목차

Morphology

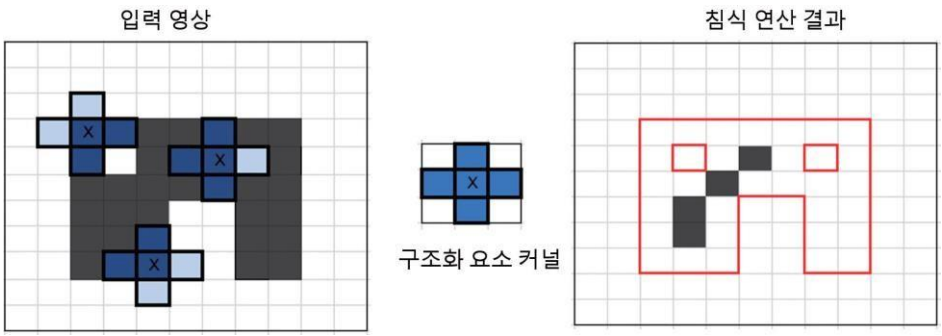
❖ Morphological Operation

- 형태학적 관점의 영상 연산
- Gray-Scaled 영상 대상
- 노이즈 제거, 단순화, 떨어진 부분 이나 구멍 채우기 등에 활용
- Structuring Element
  - 원본 이미지에 적용할 kernel
  - 사각형, 타원형, 십자형
  - np.ones( ) : 사각형
  - cv2.getStructuringElements(shape, ksize[, anchor]) : 사각형 이외
    - shape: cv2.MORPH_*
      - RET, ELLIPSE, CROSS
    - ksize : 커널크기
    - anchor : 구조화 요소의 기준점, 십자형 이외 무의미
- 연산 종류
  - Erosion (침식)
  - Dilation (팽창)
  - Opening (열기)
  - Closing (닫기)

Morphology

❖ Erosion

- 침식
- Structuring Element를 적용하여 한 픽셀이라도 0이 있으면 제거
  - 고스란히 커널을 올려 놓을 수 없으면 제거
- 작은 객체(주로 노이즈) 제거에 효과적
- cv2.erode(src, kernel [, anchor, iteration])
  - src : 입력 이미지
  - kernel : Structuring Element
  - anchor : Kernel의 중심점, default(-1, -1)
  - iteration : 반복 횟수, default(1)



[그림 6-18] 침식 연산 개념도

Morphology

❖ Erosion

- Example

[예제 6-13] 침식 연산(morph_erode.py)

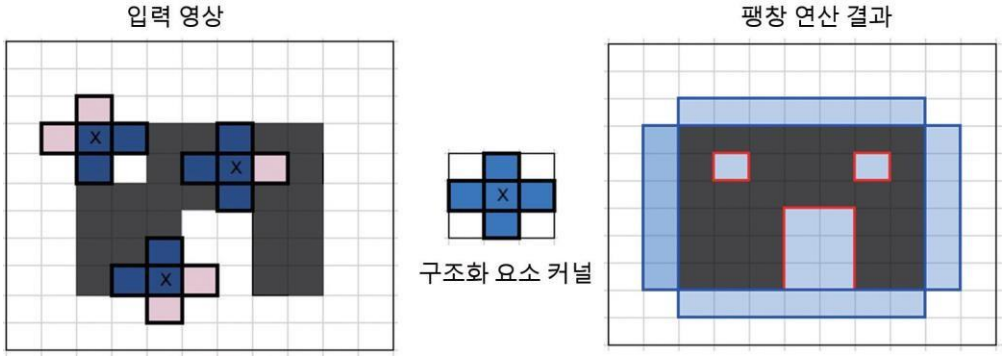
```
import cv2
import numpy as np
img = cv2.imread('../img/morph_dot.png')
# 구조화 요소 커널, 사각형 (3x3) 생성 ---①
k = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
# 침식 연산 적용 ---②
erosion = cv2.erode(img, k)
# 결과 출력
merged = np.hstack((img, erosion))
cv2.imshow('Erode', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[그림 6-19] [예제 6-13]의 실행 결과

Morphology

❖ Dilation

- 팽창
- Structuring Element를 적용하여 0인 픽셀을 채운다.
  - 커널을 올려 놓고 덮히지 않는 부분 채우기
- 끊어짐 연결, 구멍 채우기에 효과적
- cv2.dilation(src, kenel [, anchor, iteration)
  - src : 입력 이미지
  - kernel : Structuring Element
  - anchor : Kernel의 중심점, default(-1, -1)
  - iteration : 반복 횟수, default(1)



[그림 6-20] 팽창 연산 개념도

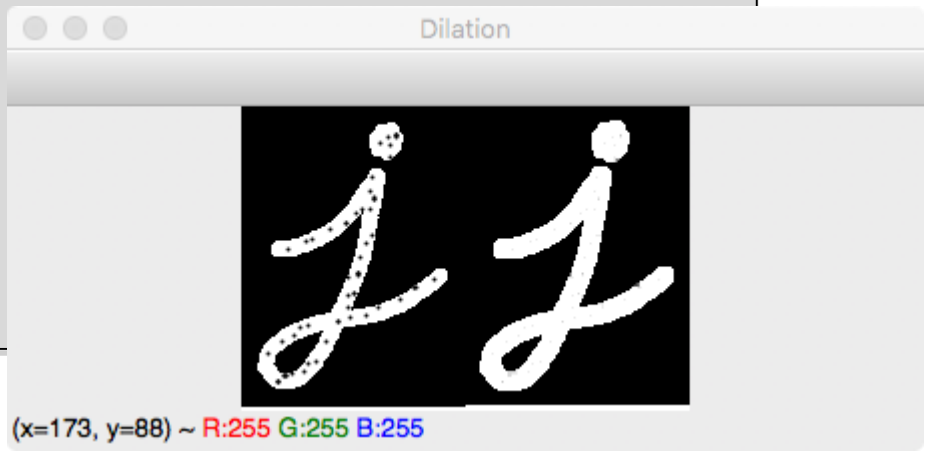
Morphology

❖ Dilation

- Example

[예제 6-13] 팽창 연산(morph_dilate.py)

```
import cv2
import numpy as np
img = cv2.imread('../img/morph_hole.png')
# 구조화 요소 커널, 사각형 (3x3) 생성 ---①
k = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
# 팽창 연산 적용 ---②
dst = cv2.dilate(img, k)
# 결과 출력
merged = np.hstack((img, dst))
cv2.imshow('Dilation', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

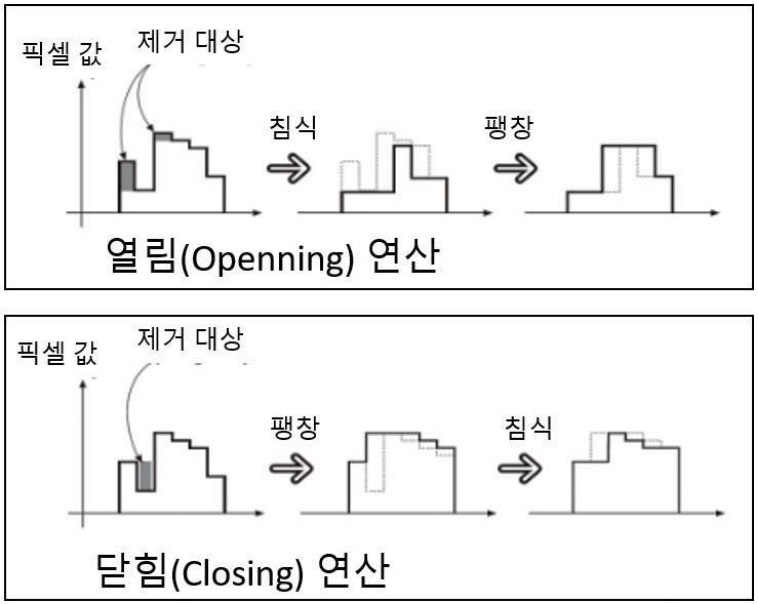


[그림 6-21] [예제 6-14]의 실행 결과

Morphology

❖ Opening & Closing

- Erosion과 Dilation의 조합
- Opening : Erosion + Dilation
  - 주변보다 밝은 노이즈 제거에 효과적
  - 맞닿아 있는 독립적 개체 분리
- Closing : Dilation + Erosion
  - 주변보다 어두운 노이즈 제거에 효과적
  - 끊어짐 연결
- cv2.morphologyEx(src, op, kernel)
  - src : 입력 영상
  - op : 연산 타입
    - cv2.MORPH_OPEN
    - cv2.MORPH_CLOSE
    - cv2.MORPH_GRADIENT : dilation과 erosion의 차이
    - cv2.MORPH_TOPHAT : opening과 원본의 차이
    - cv2.MORPH_BLACKHAT : closing과 원본의 차이



[그림 6-22] 열림 연산과 닫힘 연산 개념도

❖ Opening & Closing

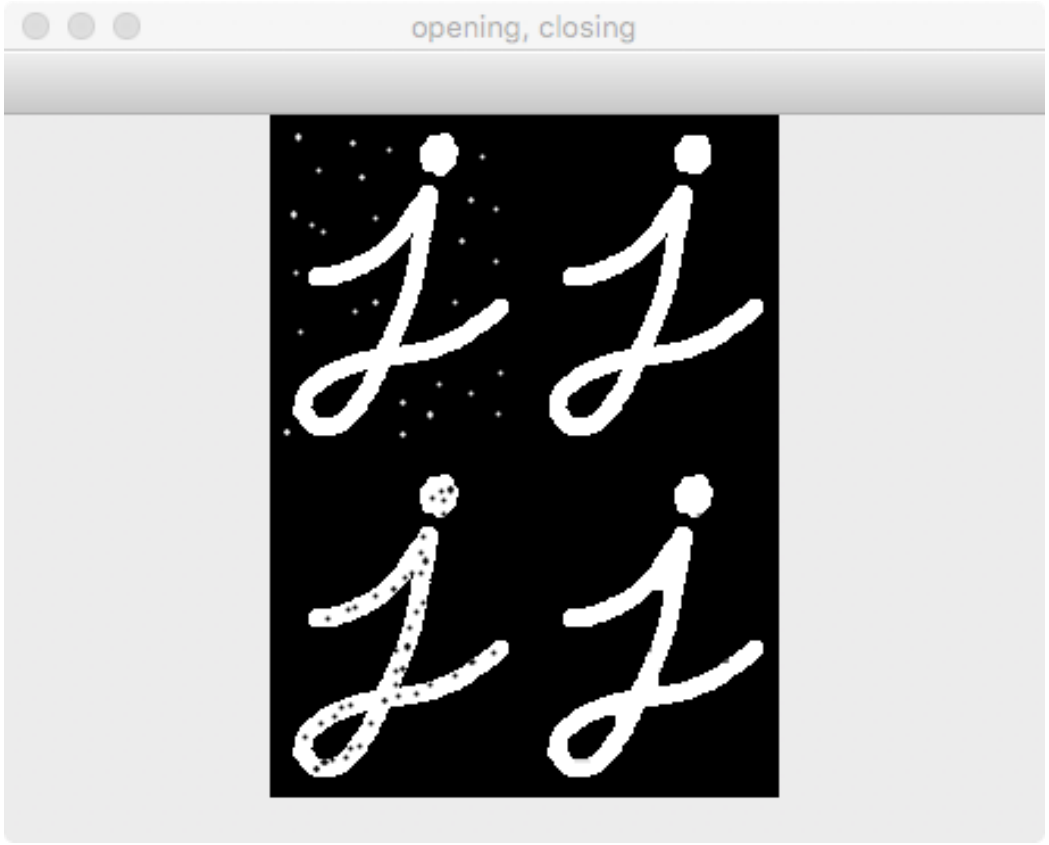
- Example

```
import cv2
import numpy as np
img1 = cv2.imread('../img/morph_dot.png', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('../img/morph_hole.png', cv2.IMREAD_GRAYSCALE)
# 구조화 요소 커널, 사각형 (5x5) 생성 ---①
k = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
# 열림 연산 적용 ---②
opening = cv2.morphologyEx(img1, cv2.MORPH_OPEN, k)
# 닫힘 연산 적용 ---③
closing = cv2.morphologyEx(img2, cv2.MORPH_CLOSE, k)
# 결과 출력
merged1 = np.hstack((img1, opening))
merged2 = np.hstack((img2, closing))
merged3 = np.vstack((merged1, merged2))
cv2.imshow('opening, closing', merged3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-15] 열림과 닫힘 연산으로 노이즈 제거(morph_open_close.py)

❖ Opening & Closing

- Example

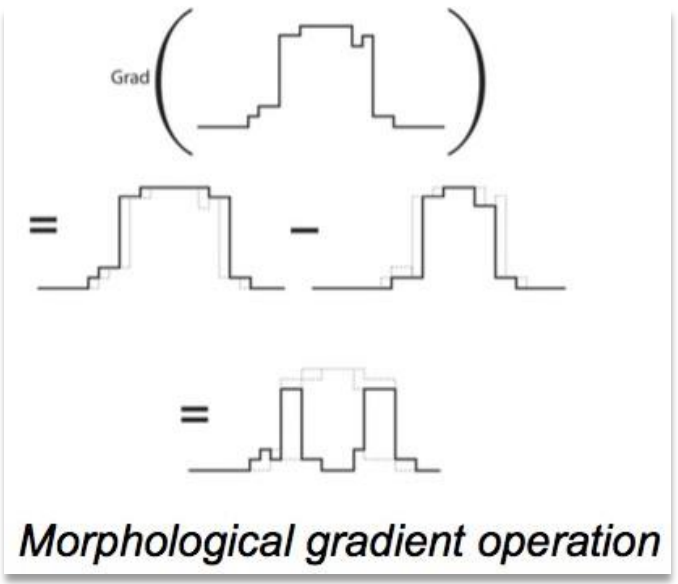


[그림 6-23] [예제 6-15]의 실행 결과

Morphology

❖ Gradient, Top-Hat, Black-Hat

- Gradient = dilate − erode
  - 외곽선만 남기는 효과
- TopHat = src − opening
  - 밝기 값이 크게 튀는 부분 제거
- BlackHat = closing − src
  - 어두운 부분 강조



Morphological gradient operation

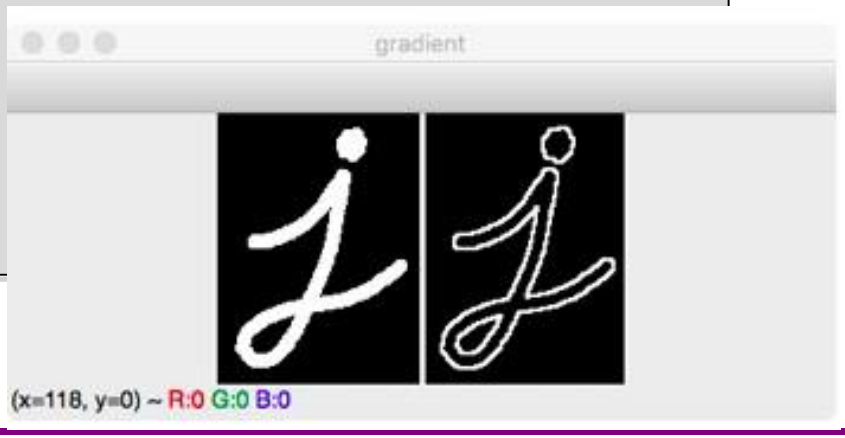
Morphology

❖ Gradient

- Example

[예제 6-16] 모폴로지 그레이디언트(morph_gradient.py)

```
import cv2
import numpy as np
img = cv2.imread('../img/morphological.png')
# 구조화 요소 커널, 사각형 (3x3) 생성 ---①
k = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
# 열림 연산 적용 ---②
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT,
k)
# 결과 출력
merged = np.hstack((img, gradient))
cv2.imshow('gradient', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



[그림 6-24] [예제 6-16]의 실행 결과

❖ Top-Hat, Black-Hat

- Example

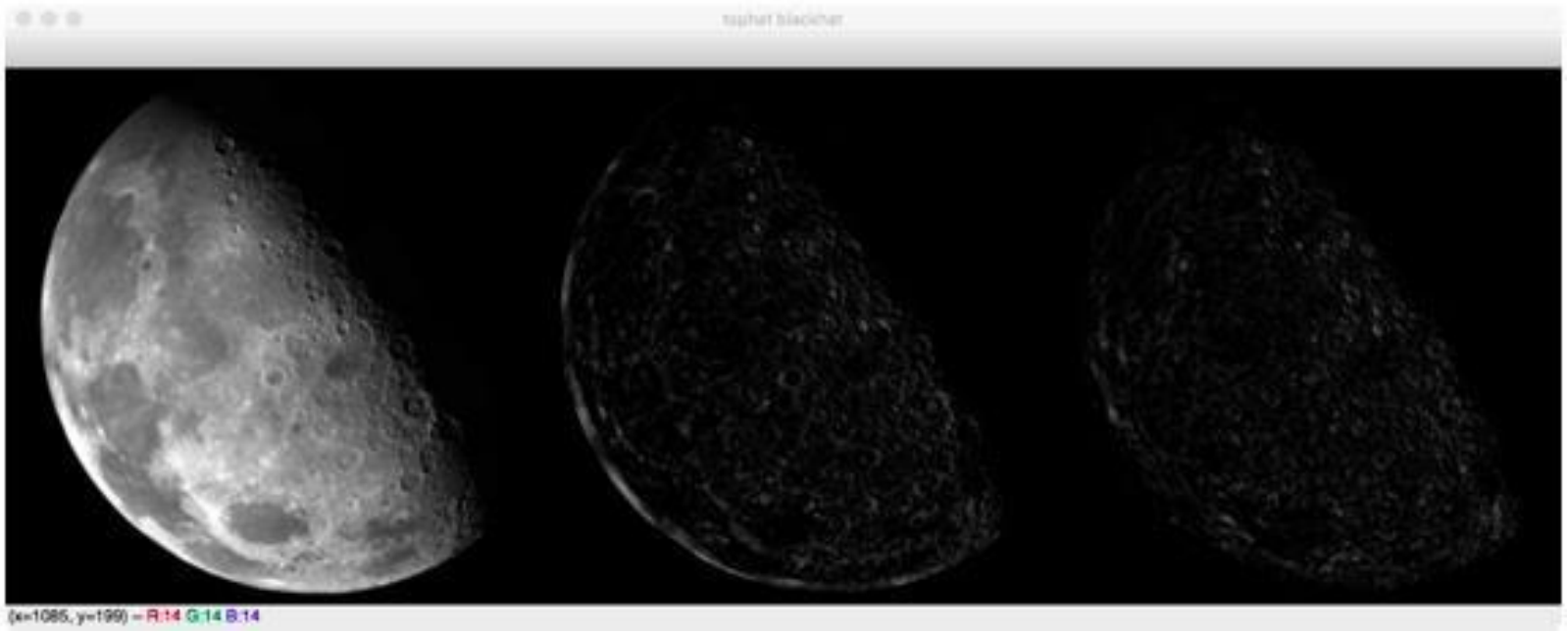
```
import cv2
import numpy as np
img = cv2.imread('../img/moon_gray.jpg')
# 구조화 요소 커널, 사각형 (5x5) 생성 ---①
k = cv2.getStructuringElement(cv2.MORPH_RECT, (9,9))
# 탑햇 연산 적용 ---②
tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, k)
# 블랫햇 연산 적용 ---③
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, k)
# 결과 출력
merged = np.hstack((img, tophat, blackhat))
cv2.imshow('tophat blackhat', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-17] 모폴로지 탑햇, 블랙햇 연산(morph_hat.py)

Morphology

❖ Top-Hat, Black-Hat

- Example



[그림 6-25] [예제 6-17]의 실행 결과

세부목차

Image Pyramids

❖ Image Pyramids

- 영상의 크기를 단계적으로 축소 및 확대해서 피라미드 처럼 쌓은 것
- 속도와 정확도
  - 작은 이미지로 빠르게 분석 후 다음 단계 영상으로 자세히 분석
- 크기에 따른 분석 결과 차이 보정
- Gaussian Pyramids
  - Down Sampling/Scaling
  - Shrinking Images
- Laplacian Pyramids
  - Up Sampling/Scaling
  - Enlarging Images

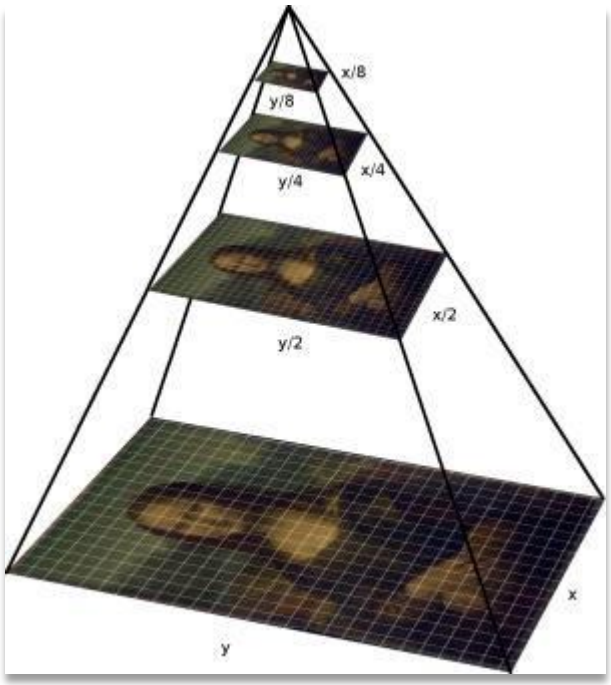


Image Pyramids

❖ Gaussian Pyramids

- cv2.pyrDown(src [, dstsize, borderType])
  - src : 원본 이미지
  - dtsize : 출력 이미지 크기
  - 피라미드 윗단계 영상 생성
  - 가우시안 커널 적용 후 짝수번째 열과 행 삭제
  - ¼ 크기로 작아진다
- cv2.pyrUp()
  - 피라미드 아랫단계 영상 생성
  - 새로 생성한 행과 열을 0으로 채우고, 지정된 필터로 컨볼루션 연산 후 설정
  - 4배 크기로 커진다.
  - 흐려진다

❖ Gaussian Pyramids

- Example

[예제 6-18] 가우시안 이미지 피라미드(pyramid_gaussian.py)

```
import cv2
img = cv2.imread('../img/girl.jpg')
# 가우시안 이미지 피라미드 축소
smaller = cv2.pyrDown(img) # img x 1/4
# 가우시안 이미지 피라미드 확대
bigger = cv2.pyrUp(img) # img x 4
# 결과 출력
cv2.imshow('img', img)
cv2.imshow('pyrDown', smaller)
cv2.imshow('pyrUp', bigger)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[그림 6-26] [예제 6-18]의 실행 결과

Image Pyramids

LaplacianPyramids

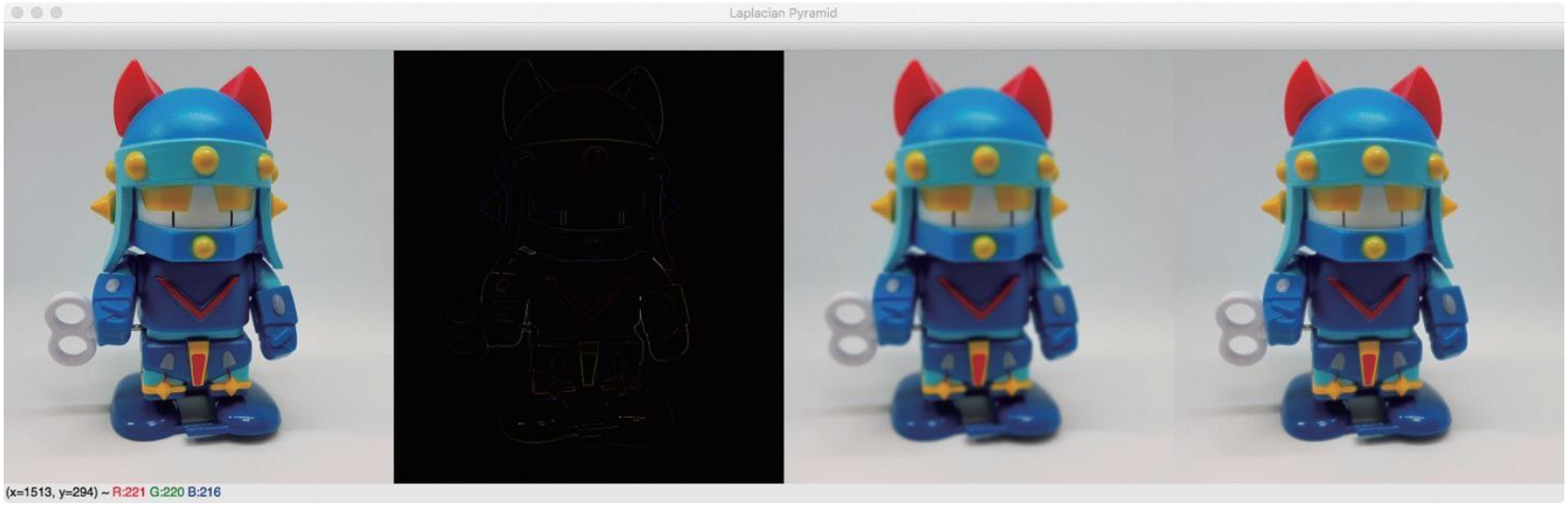
- 원본과 가우시안 피라미드의 작은 레벨 확대의 차이
- $L_i = G_i - pyrUp(G_{i+1})$
- 작은 피라미드에서 원본 복원에 사용
- 엣지를 제외 대부분 픽셀 0(zero)

```
import cv2
import numpy as np
img = cv2.imread('../img/taekwonv1.jpg')
# 원본 영상을 가우시안 피라미드로 축소
smaller = cv2.pyrDown(img)
# 축소한 영상을 가우시안 피라미드로 확대
bigger = cv2.pyrUp(smaller)
# 원본에서 확대한 영상 빼기
laplacian = cv2.subtract(img, bigger)
# 확대 한 영상에 라플라시안 영상 더해서 복원
restored = bigger + laplacian
# 결과 출력 (원본 영상, 라플라시안, 확대 영상, 복원 영상)
merged = np.hstack((img, laplacian, bigger, restored))
cv2.imshow('Laplacian Pyramid', merged)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[예제 6-19] 라플라시안 피라미드로 영상 복원(pyramid_laplacian.py)

Image Pyramids

❖ **LaplacianPyramids**



[그림 6-27] [예제 6-19]의 실행 결과

세부목차

Workshop

❖ Mosaic2

- 마우스로 선택한 영역을 필터를 이용해서 모자이크 처리 하세요
  .
- 결과 예시

[그림 6-28] 모자이크 처리 예시

- 힌트
  - 선택한 영역을 평균 블러 처리

Workshop

❖ Cartoonizing Camera

- 카레라 영상이 만화나 수채화 같은 느낌이 나게 만들어 보세요.
- 스케치 영상과 물감 그림 영상을 함께 출력 하세요.
- 결과 예시

[그림 6-29] 그림처럼 찍어주는 카메라 예시

Workshop

❖ Cartoonizing Camera

- 힌트
  - 스케치 영상: 그레이 스케일로 바꾸어서 엣지를 얻어서 반전
    - cv2.Laplacian()
    - cv2.GauusianBlur()
  - 물감 그림 : 컬러 스케일 영상을 블러 필터 적용하고 스케치 영상과 합성
    - cv2.blur()
    - cv2.bitwise_and()