



# Day67; 20221212

날짜	@2022년 12월 12일
유형	@2022년 12월 12일
태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<https://github.com/u8yes/ai>

u8yes/AI



Contributor 1 Issues 0 Stars 0 Forks 0

## 선형 공간 필터링 - filter(mask)

다 합쳐서 총 1이 되도록 setting 해준다.

## 블러링(blurring)

- ✓ 블러링은 영상에서 픽셀값이 급격하게 변하는 부분들을 감소시켜 점진적으로 변하게 함으로써 영상이 전체적으로 부드러운 느낌이 나게 하는 기술인데 스무딩(smoothing)이라 하기도 함
- ✓ 급격히 변화하는 것을 점진적으로 변하게 하려면 커널 마스크를 이용해서 회선을 수행하면 됨

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

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

## 01\_blurring.py

```
01_blurring.py ×
import numpy as np, cv2, time

def blur_convolution(image, filter):
    rows, cols = image.shape[:2] # 시작은 0부터 # 행과 열
    # height, width로 생각하는 게 더 쉽다.
    # 이미지는 300x270
    # rows = 270, cols = 300 # 270행, 300열
    dst = np.zeros((rows, cols), np.float32) # 270 by 300
    xcenter, ycenter = filter.shape[1]//2, filter.shape[0] // 2
    # filter는 3 by 3이다. 그래서 3/2 하면 몫 1이 나온다.
    # xcenter, ycenter는 둘 다 1이 됨.
```

```

for i in range(ycenter, rows - ycenter): # 행만큼 반복하겠다.
    # ycenter를 만든 것은 다음으로 넘어가기 위함.
    # 270이 아닌 269에서 끝내야 하기 때문에 -1을 추가해준 것.
    # 1부터 시작하려고 center라고 한 것임.
    for j in range(xcenter, cols - xcenter):
        y1, y2 = i - ycenter, i + ycenter + 1 # 0, 3 저장 돼있음
        x1, x2 = j - xcenter, j + xcenter + 1 # 0, 3 저장 돼있음
        roi = image[y1:y2, x1:x2].astype("float32")
        # y1=0 ~ y2=3-1 까지, x도 동일하게.
        # 정수값을 실수값으로 바꿔줌.

        tmp = cv2.multiply(roi, filter)
        # 관심영역과 mask를 각각 곱해줌.
        dst[i, j] = cv2.sumElems(tmp)[0]
        # sumElems은 채널 별로 다 더해줌
        # 흑백이라서 첫번째 채널의 값만 읽어옴.

return dst

```

```

image = cv2.imread("images/filter_blur.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류 발생")

# 블러링 마스크 원소 지정
filter = [1/9, 1/9, 1/9,
          1/9, 1/9, 1/9,
          1/9, 1/9, 1/9]
# filter를 총 1로 만들어줘야함.

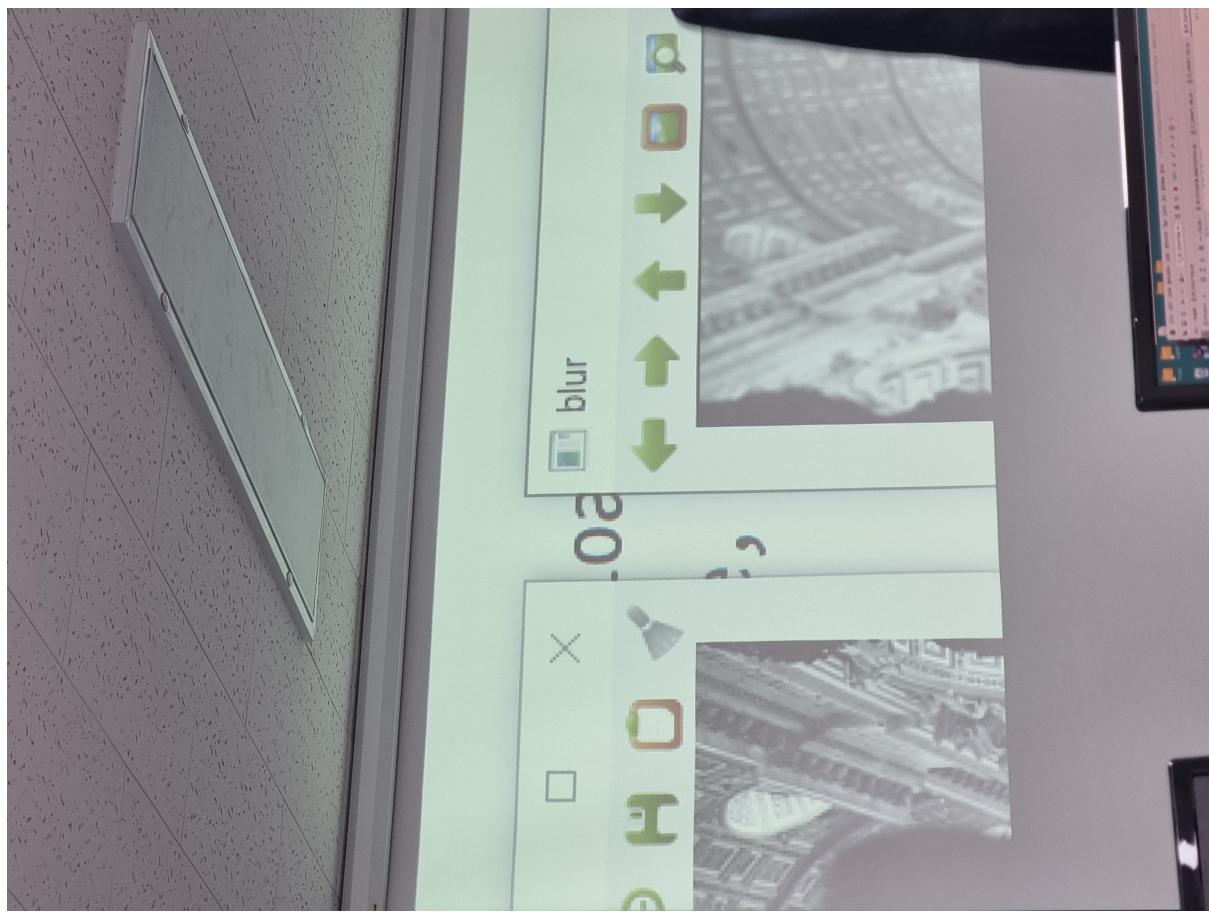
mask = np.array(filter, np.float32).reshape(3, 3)
blur = blur_convolution(image, mask)

cv2.imshow("image", image)
cv2.imshow("blur", blur.astype("uint8"))
# 0이지는 정수값이어야 하기 때문에 실수값을 정수로 바꿔줌.

cv2.waitKey(0)

```

높은 차이를 완만하게 만들어줘서 blur를 만들어줌.



blur는 테두리가 검은색으로 삭제된 상태.

$y = 1$ ,  $x = 1$  부분

## 샤프닝

## 샤프닝(Sharpening)



입력 픽셀에서 이웃 픽셀끼리 차이를 크게 되도록 출력 픽셀를 만들어서 날카로운 느낌이 나게 만드는 것으로 영상의 세세한 부분을 강조할 수 있으며 경계 부분에서 명암 대비가 증가되는 효과를 낼 수 있음

샤프닝에서는 마스크 원소들의 값 차이가 커지도록 구성하는 것

입력 영상의 픽셀과 출력 영상의 픽셀가 마스크의 중심 위치에서 대응되는데 이 마스크의 중심 위치의 계수를 중심 계수라 하는데 마스크 중심 계수의 비중이 크면 출력 영상은 입력 영상의 형태를 유지하게 되므로 주변 계수들을 중심 계수와 값의 차이를 크게 만들면 샤프닝이 수행됨

마스크 원소의 전체 합이 1이 되어야 입력 영상의 밝기가 손실 없이 출력 영상의 밝기로 유지됨

마스크 원소의 합이 1보다 작으면 출력 영상의 밝기가 입력 영상보다 어두워지며 1보다 크면 입력 영상보다 더 밝아짐

전체 합이 0이 되면 대부분의 출력 픽셀가 0이 되어 출력 영상이 검은색을 나타내게 되는데 에지 검출에서 주로 사용



중심 계수값을 1보다 크면 밝아지고, 1보다 작으면 입력영상보다 밝기가 어두워짐

마스크 원소의 합이 1보다 작으면 출력 영상의 밝기가 입력 영상보다 어두워지며 1보다 크면 입력 영상보다 더 밝아짐

## 02\_sharpening.py

```

import numpy as np, cv2
from Common.filters import filter
# 함수를 직접 사용 가능해짐.

image = cv2.imread("images/filter_sharpen.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류 발생")

# 샤프닝 필터(마스크) 원소 지정
# 전체가 1이 되게끔 # 중심값을 크게 해줌.
data1 = [0, -1, 0,
          -1, 5, -1,
          0, -1, 0]

data2 = [[-1, -1, -1], # 다이렉트로 직접 처리가 가능하다.
          [-1, 9, -1],
          [-1, -1, -1]]

mask1 = np.array(data1, np.float32).reshape(3,3)
# filter에서는 실수로.
mask2 = np.array(data2, np.float32)
# reshape 필요없이 바로 2차원으로 만든다.

```

```

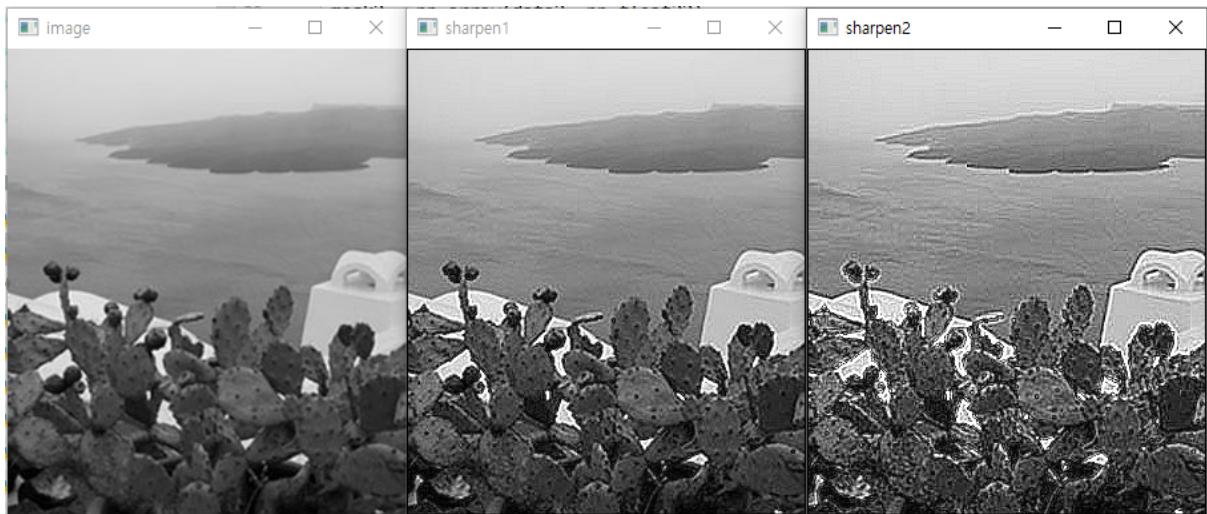
# 합성곱으로 filter해주는 함수를 사용.
sharpen1 = filter(image, mask1)
sharpen2 = filter(image, mask2)

sharpen1 = cv2.convertScaleAbs(sharpen1)
# scaleAbsolute - 음수로 나오는 것을 절대값을 써움
sharpen2 = cv2.convertScaleAbs(sharpen2)

cv2.imshow("image", image)
cv2.imshow("sharpen1", sharpen1)
cv2.imshow("sharpen2", sharpen2)

cv2.waitKey(0)

```

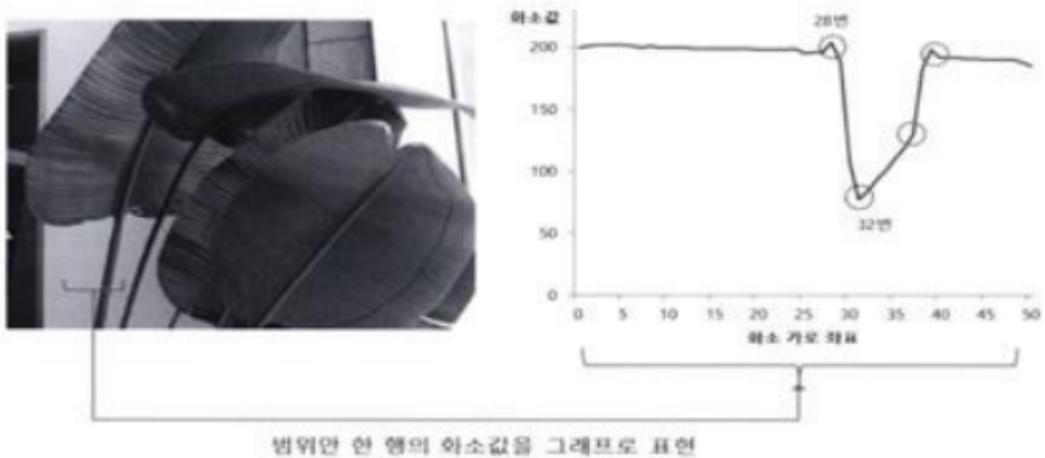


sharpen1 - 중심계수가 5

sharpen2 - 중심계수가 9

---

## 에지 검출(edge detection)이란 에지에 해당하는 픽셀들을 찾는 과정



0에 가까운 부분은 어둡기 때문에 떨어짐.

급격하게 변화하는 부분으로 정의

---

픽셀의 차분 - 이전값에서 현재값을 뺌셈

차분으로 하면 수행속도가 느려짐.

---

그래서 미분 마스크 사용함.

- ▣ 마스크를 이용하여 에지를 계산할 수도 있는데 1차 미분 마스크나 2차 미분 마스크를 사용하여 회선을 수행
- 

수식을 증명할 필요가 없다. 원리만 이해하면 된다.

$$G[f(x, y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$
$$G_x = \frac{f(x+dx, y) - f(x, y)}{dx} \doteq f(x+1, y) - f(x, y), \quad dx = 1$$
$$G_y = \frac{f(x, y+dy) - f(x, y)}{dy} \doteq f(x, y+1) - f(x, y), \quad dy = 1$$
$$G[f(x, y)] \doteq \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$
$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

---

### 미분 - 접선의 기울기

미분은 함수의 순간 변화율을 구하는 계산 과정을 의미하는데 에지가 픽셀의 밝기가 급격히 변하는 부분이기 때문에 함수의 변화율을 취하는 미분 연산을 이용해서 에지를 검출할 수 있음

흡사한 것이 GDA(**Gradient Descent Algorithm**)와 비슷하다.

---

연속형

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

이산형이 되게 만들어줌.  $dx$ ,  $dy$ 는 1로 만들어줌(분모)

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

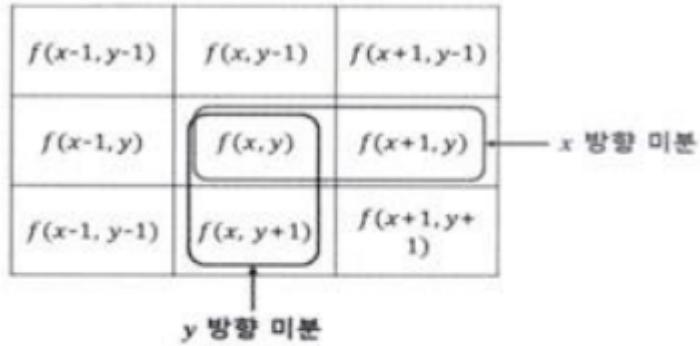
매그니튜드

$$G[f(x, y)] \doteq \sqrt{G_x^2 + G_y^2} \approx |G_x| + |G_y|$$
$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

---

### 미분 필터

- ✓ 2차원 공간상의 한 픽셀에서 수평 방향과 수직 방향으로 각각 미분하는데 이것을 보통 편미분(partial derivative)이라 하고 각 방향의 편미분을 한 픽셀 단위 ( $dx = 1$ ,  $dy = 1$ )의 차분으로 근사
- ✓ 다음으로 각 방향의 차분을 이용해서 기울기의 크기를 계산하는데 이 크기(magnitude)가 에지의 강도가 되는데 계산 복잡도를 줄이기 위해서 제곱과 제곱근 대신에 절댓값을 사용하기도 하며 또한 역탄젠트(arctan) 함수에 가로 방향과 세로 방향 차분을 적용하면 에지의 방향을 계산할 수도 있음
- ✓ 1차 미분 공식을 영상에 구현하는 쉬운 방법이 1차 미분 마스크로 회선을 적용하는 것
- ✓  $3 \times 3$  마스크에서 입력 픽셀의 위치를 생각해 보면 마스크의 중심 위치의 입력 픽셀가  $f(x, y)$ 일 때 주변 픽셀의 위치를 보면 다음 그림과 같음



0	0	0
0	-1	0
0	1	0

(a)  $y$  방향 마스크

0	0	0
0	-1	1
0	0	0

(b)  $x$  방향 마스크

$$f(x+1, y) - f(x, y),$$

$$f(x, y+1) - f(x, y),$$

연속형일 경우가 편미분(partial derivative).

정수값에서 가장 작은 값이 1.

## 예지 검출

### ❖ 미분 필터

- ✓ 마스크 원소를 (a) 와 같이  $f(x, y)$ ,  $f(x, y+1)$  위치에 -1 과 1을 구성하여 회선을 수행하면 회선의 내부 계산 수식이  $f(x, y+1) - f(x, y)$  이 되어서  $y$  방향 미분인  $G_y$  와 같은 결과가 되고 오른쪽의 (b)와 같이  $f(x, y)$ ,  $f(x+1, y)$  위치에 -1과 1을 구성하여 회선을 수행하면 회선 수식이  $f(x+1, y) - f(x, y)$ 이 되어서  $x$  방향 미분인  $G_x$ 가 적용됨

## Roberts Mask

## Roberts Mask

- 로버트 마스크는 대각선 방향으로 1과 -1을 배치하여 구성
- 나머지 원소의 값이 모두 0이어서 다른 1차 미분 마스크에 비해서 계산이 단순
- 한 번만 차분을 계산하기 때문에 차분의 크기가 작고 이로 인해서 경계가 확실한 에지만을 추출하며 잡음에 매우 민감

$$G_x =$$

-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

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

## 03\_edge\_RobertsMask.py

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

제곱을 해서 부호를 없애주고 루트 씌워줌.

### magnitude

미국식[ˌmægnɪtʃd] 영국식[ˌmægnɪtju:d]

(명사)

1 (엄청난) 규모[중요도]

We did not realize the magnitude of the problem.

우리는 그 문제의 규모[중요도]를 깨닫지 못했다.

2 (별의) 광도

The star varies in brightness by about three magnitudes.

그 별은 밝기가 세 광도 정도로 달라진다.

3 지진 규모

Magnitude of an earthquake

```

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) # x분의 증가량
    dst2 = filter(image, mask2) # y분의 증가량

    dst = cv2.magnitude(dst1, dst2) # 제곱해서 루트씌우는 식
    dst1, dst2 = np.abs(dst1), np.abs(dst2) # 음수값이 반영되도록

    # edge 검출에 대한 시각화
    dst = np.clip(dst, 0, 255).astype("uint8")
    # 최대 255가 넘어가는 값은 255로, 0 이하는 0으로
    dst1 = np.clip(dst1, 0, 255).astype("uint8")
    dst2 = np.clip(dst2, 0, 255).astype("uint8")

    return dst, dst1, dst2

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

data1 = [-1, 0, 0,
          0, 1, 0,
          0, 0, 0]

data2 = [0, 0, -1,
          0, 1, 0,
          0, 0, 0]

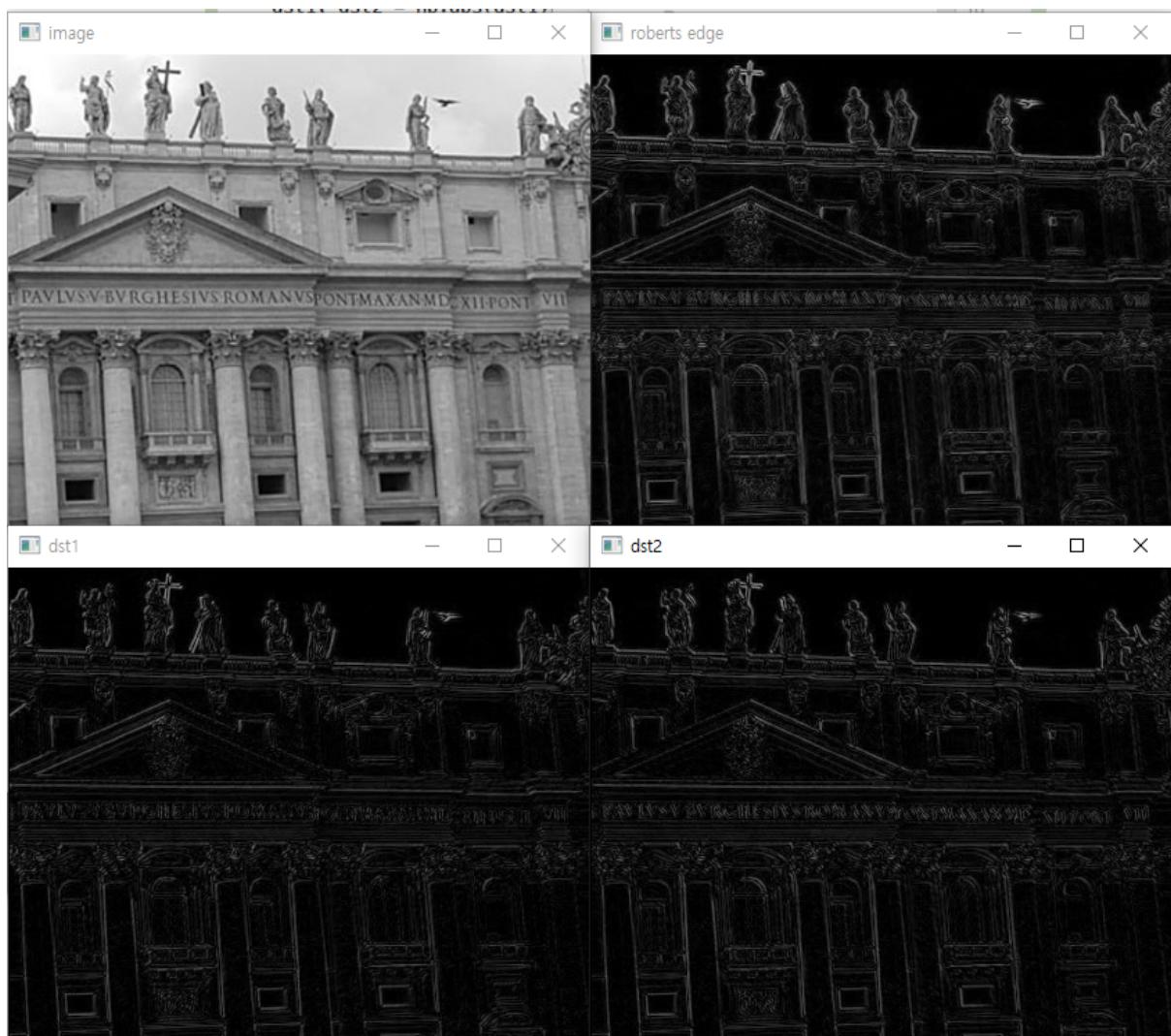
```

```

dst, dst1, dst2 = differential(image, data1, data2) # 1차 미분 수행

cv2.imshow("image", image)
cv2.imshow("roberts edge", dst)
cv2.imshow("dst1", dst1) # 좌측 대각선으로 적용
cv2.imshow("dst2", dst2) # 우측 대각선으로 적용
cv2.waitKey(0)

```



### Roberts Mask

- 로버트 마스크는 대각선 방향으로 1 과 -1을 배치하여 구성
- 나머지 원소의 값이 모두 0이어서 다른 1차 미분 마스크에 비해서 계산이 단순
- 한 번만 차분을 계산하기 때문에 차분의 크기가 작고 이로 인해서 경계가 확실한  
에지만을 추출하며 잡음에 매우 민감

## 04\_edge\_prewitt.py

```
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) # x분의 증가량
    dst2 = filter(image, mask2) # y분의 증가량

    dst = cv2.magnitude(dst1, dst2) # 제곱해서 루트씌우는 식
    dst1, dst2 = np.abs(dst1), np.abs(dst2) # 음수값이 반영되도록

    # edge 검출에 대한 시각화
    dst = np.clip(dst, 0, 255).astype("uint8")
    # 최대 255가 넘어가는 값은 255로, 0 이하는 0으로
    dst1 = np.clip(dst1, 0, 255).astype("uint8")
    dst2 = np.clip(dst2, 0, 255).astype("uint8")

    return dst, dst1, dst2

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

# 수직 방향
data1 = [-1, 0, 1, # 프리윗 수직 마스크
          -1, 0, 1,
          -1, 0, 1]
# 수평 방향
data2 = [-1, -1, -1, # 프리윗 수평 마스크
          0, 0, 0,
          1, 1, 1]
```

```

dst, dst1, dst2 = differential(image, data1, data2) # 1차 미분 수행

cv2.imshow("image", image)
cv2.imshow("prewitt edge", dst)
cv2.imshow("dst1- vertical", dst1) # 좌측 대각선으로 적용
cv2.imshow("dst2- horizontal", dst2) # 우측 대각선으로 적용
cv2.waitKey(0)

```

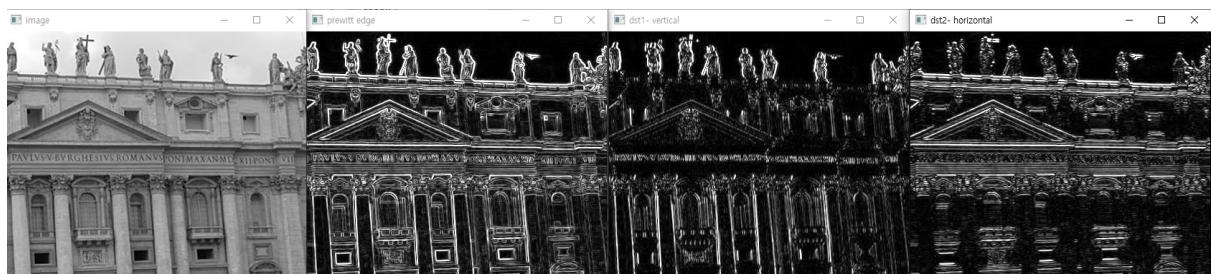
$$G_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

수직 마스크

$$G_y = \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

수평 마스크

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



수직, 수평

아래만 변경한 상태

```

# 수직 방향
data1 = [-1, 0, 1, # 프리윗 수직 마스크
          -1, 0, 1,
          -1, 0, 1]

# 수평 방향
data2 = [-1, -1, -1, # 프리윗 수평 마스크
          0, 0, 0,
          1, 1, 1]

```

---

실행 후 아래처럼 변경 가능

```
# edge 검출에 대한 시각화  
dst = cv2.convertScaleAbs(dst)  
dst1 = cv2.convertScaleAbs(dst1)  
dst2 = cv2.convertScaleAbs(dst2)
```

---

## Sobel Mask

$G_x =$

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

수직마스크

$G_y =$

1	-2	1
0	0	0
1	2	1

수평마스크

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

소벨 마스크도 수직 마스크의 회선 결과와 수평 마스크의 회선 결과의 크기로 구성되는데 수직, 수평 방향의 에지도 잘 추출하며 특히 중심 계수의 차분 비중을 높였기 때문에 대각선 방향의 에지도 잘 검출

---

가장 흔히 쓰이는 mask

**05\_edge\_Sobel\_Mask.py**

```

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) # x분의 증가량
    dst2 = filter(image, mask2) # y분의 증가량

    dst = cv2.magnitude(dst1, dst2) # 제곱해서 루트쓰는 식

    # edge 검출에 대한 시각화
    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("영상 파일 읽기 오류")

# 수직 방향
data1 = [-1, 0, 1, # 프리윗 수평 마스크
          -2, 0, 2,
          -1, 0, 1]
# 수평 방향
data2 = [-1, -2, -1, # 프리윗 수직 마스크
          0, 0, 0,
          1, 2, 1]

dst, dst1, dst2 = differential(image, data1, data2) # 1차 미분 수행

#OpenCV 제공 소벨 엣지 계산
dst3 = cv2.Sobel(np.float32(image), cv2.CV_32F, 1, 0) # x를 편미분
dst4 = cv2.Sobel(np.float32(image), cv2.CV_32F, 0, 1) # y를 편미분
dst3 = cv2.convertScaleAbs(dst3) # 0~255, 정수형으로 모두 변환
dst4 = cv2.convertScaleAbs(dst4)

```

```
cv2.imshow("image", image)

cv2.imshow("sobel edge", dst)
cv2.imshow("dst1- vertical", dst1)
cv2.imshow("dst2- horizontal", dst2)
cv2.imshow("dst3- vertical_OpenCV", dst3)
cv2.imshow("dst4- horizontal_OpenCV", dst4)
cv2.waitKey(0)
```



2차 미분 마스크는 밝기가 점진적으로 변화하는 것에는 반응이 보이지 않는다.

## 2차 미분 마스크

- ✓ 1차 미분 연산자는 밝기가 급격하게 변화하는 영역뿐 아니라 점진적으로 변화하는 부분까지 민감하게 에지를 검출하여 너무 많은 에지가 나타날 수 있는데 이를 보완하는 방법으로 1차 미분에서 한 번 더 미분하는 방법인 2차 미분(second order derivative) 연산이 있음
- ✓ 2차 미분 연산자는 변화하는 영역의 중심에 위치한 에지만을 검출하며 밝기가 점진적으로 변화되는 영역에 대해서는 반응을 보이지 않음
- ✓ 대표적인 방법으로는 라플라시안(Laplacian), LoG(Laplacian of Gaussian), DoG(Difference of Gaussian) 등이 있음

---

## Laplacian Mask

라플라시안은 피에르시몽 라플라스(Pierre-Simon Laplace)

### 2차 미분 마스크

- ✓ Laplacian Mask
  - 라플라시안은 피에르시몽 라플라스(Pierre-Simon Laplace)라는 프랑스의 수학자 이름을 따서 지은 것
  - 라플라시안은 함수  $f$ 에 대한 그래디언트의 발산으로 정의되며 수식으로 표현하면 다음과 같음

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$$

- 영상은 2차원 좌표계이기 때문에 2차원 직교좌표계에서 라플라시안의 수식은 다음과 같음

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- 두 항을 더하면 다음과 같이 라플라시안 마스크의 공식이 완성

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

---

엣지 부분만 0이 되게끔 어둡게 만들어야 한다는 것.

## 2차 미분 마스크

### ✓ Laplacian Mask

- $3 \times 3$  크기의 마스크를 예로 라플라시안 마스크 공식에 적용하면 중심 계수를 4 배로 하고 상하좌우 픽셀을 중심 계수와 반대 부호를 갖게 구성하고 마스크 원소의 전체 합은 0이 되어야 함
- 이런 방법으로 그림 (a)와 같이 두 개의 마스크를 구성할 수 있으며 4방향을 가지는 마스크가 되고 경우에 따라서는 그림 (b)와 같이 8방향으로 늘려 보면 모든 방향의 에지를 검출하고자 할 때도 있는데 이 경우에는 중심 계수의 값을 더 크게 하고 8방향의 모든 값을 반대 부호가 되게 하면 됨

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

0	1	0
1	-4	1
0	1	0

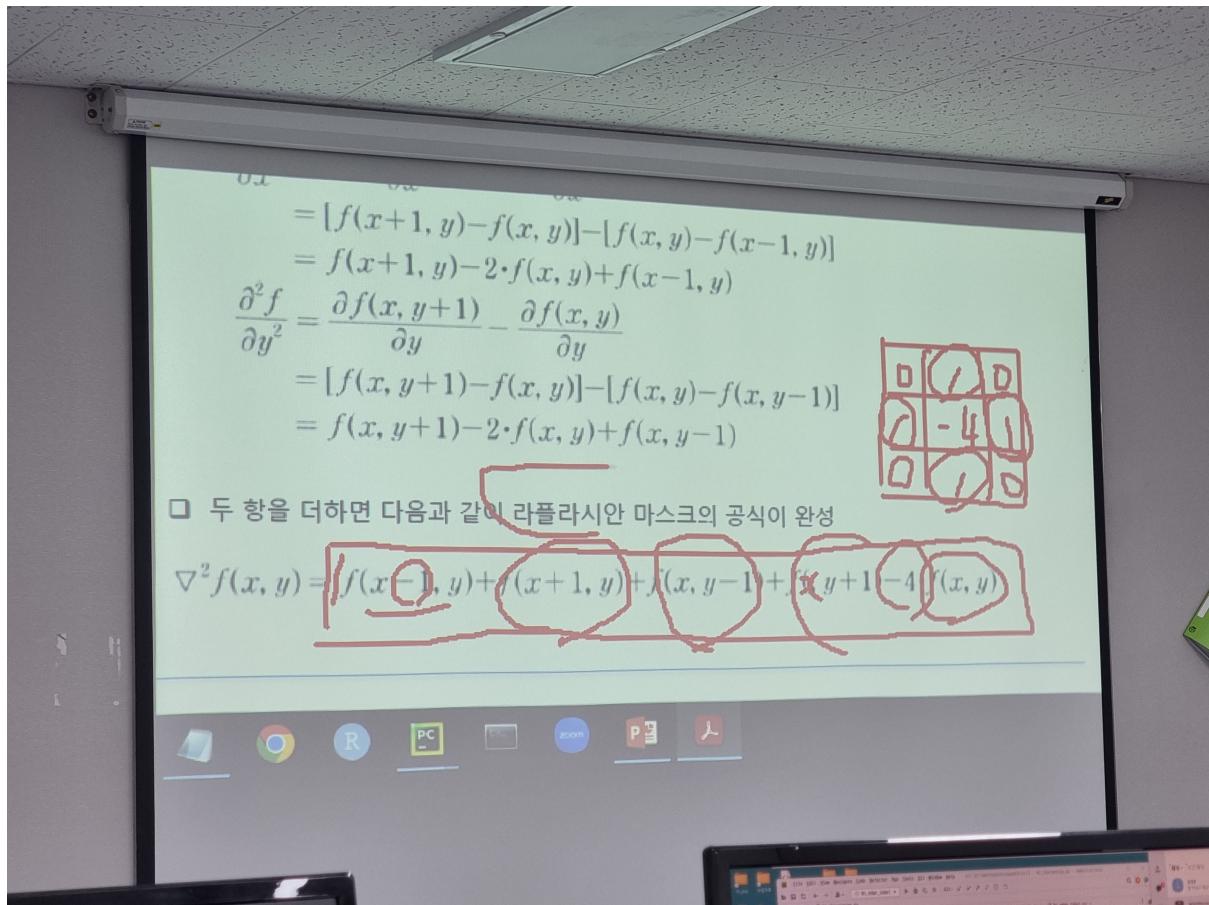
-1	-1	-1
-1	8	-1
-1	-1	-1

1	1	1
1	-8	1
1	1	1

(a) 4방향 마스크

(b) 8방향 마스크

- 라플라시안은 중심 계수와 4방향 혹은 8방향의 주변 픽셀과 차분을 합하여 에지를 검출하기 때문에 주변 픽셀에 잡음 성분이 있으면 잡음 성분에 매우 민감하여 실제보다 더 많은 에지를 검출하는 경향이 있음



## 06\_edge\_Laplacian.py

```

import numpy as np, cv2

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

data1 = [[0, 1, 0],
          [1, -4, 1],
          [0, 1, 0]]

data2 = [[-1, -1, -1], # 전체를 합했을 때 0이 되게끔
          [-1, 8, -1], # 모든 방향성을 고려하게 됨
          [-1, -1, -1]] # 2차 미분

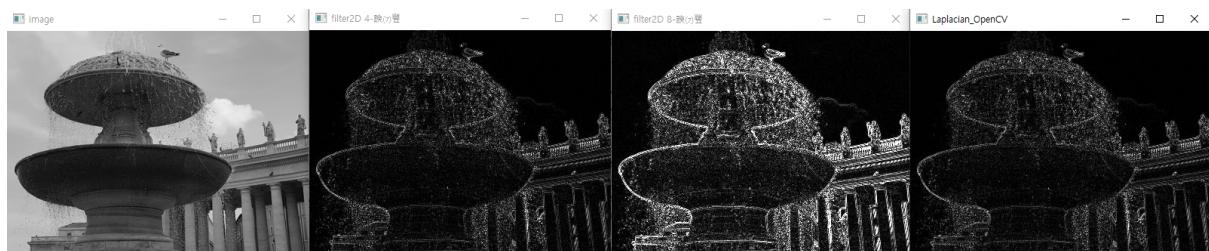
mask1 = np.array(data1, np.int16)
mask2 = np.array(data2, np.int16)

# OpenCV 함수 cv2.filter2D()를 통한 라플라시안 수행
dst1 = cv2.filter2D(image, cv2.CV_16S, mask1)
# CV_16S의 Sign은 -을수까지 넣어줌
dst2 = cv2.filter2D(image, cv2.CV_16S, mask2)
dst3 = cv2.Laplacian(image, cv2.CV_16S)

cv2.imshow("image", image)
cv2.imshow("filter2D 4-방향", cv2.convertScaleAbs(dst1))
cv2.imshow("filter2D 8-방향", cv2.convertScaleAbs(dst2))
cv2.imshow("Laplacian_OpenCV", cv2.convertScaleAbs(dst3))

cv2.waitKey(0)

```



이름에서 한글이 좀 깨져서 나옴

## LoG & DoG

LoG는 복잡한 공식에 의해 마스크를 생성해야 하며 그에 따라서 수행 시간도 많이 걸리게 되는데 이런 단점을 보완하여 LoG와 유사한 기능을 하면서 단순한 방법으로 구현하는 알고리즘이 있는데 이것이 바로 DoG(Difference of Gaussian)

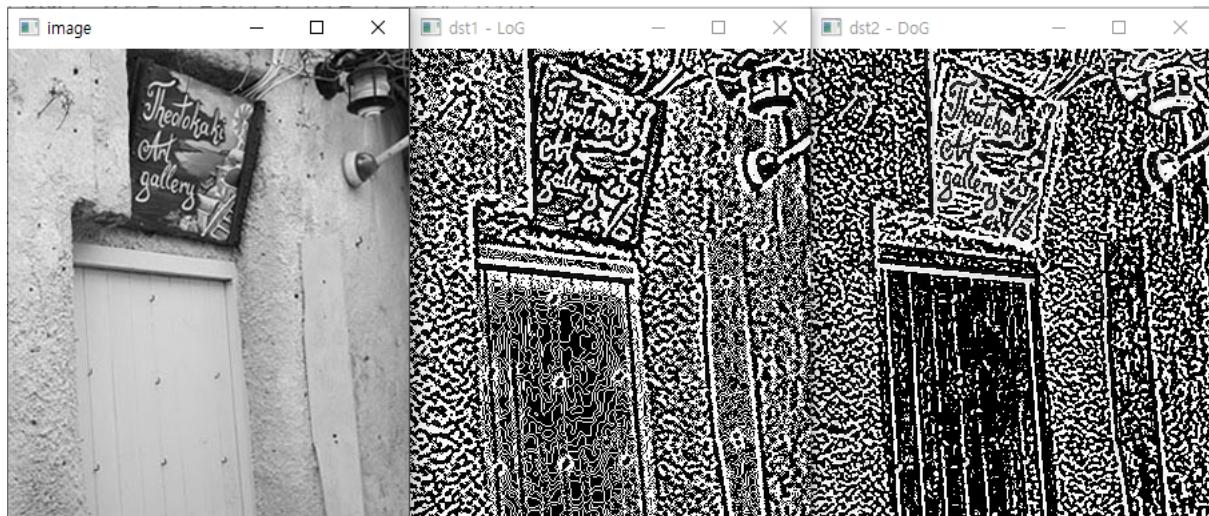
## 07\_edge\_DoG.py

```
import numpy as np, cv2

image = cv2.imread("images/dog.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류")
|
gaus = cv2.GaussianBlur(image, (7, 7), 0) # 가우시안 마스크 적용
# default는 9 by 9가 일반적 # 표준편차는 0으로 가우시안 값을 적용
# Blur는 평균을 만들어서 이미지를 부드럽게 가져감.
dst1 = cv2.Laplacian(gaus, cv2.CV_16S, 7)

gaus1 = cv2.GaussianBlur(image, (3, 3), 0)
gaus2 = cv2.GaussianBlur(image, (9, 9), 0)
dst2 = gaus1 - gaus2 # DoG 수행

cv2.imshow("image", image)
cv2.imshow("dst1 - LoG", dst1.astype("uint8"))
cv2.imshow("dst2 - DoG", dst2)
cv2.waitKey(0)
```



## □ Canny Edge

캐니 에지 알고리즘은 일반적으로 다음의 네 단계의 알고리즘으로 구성

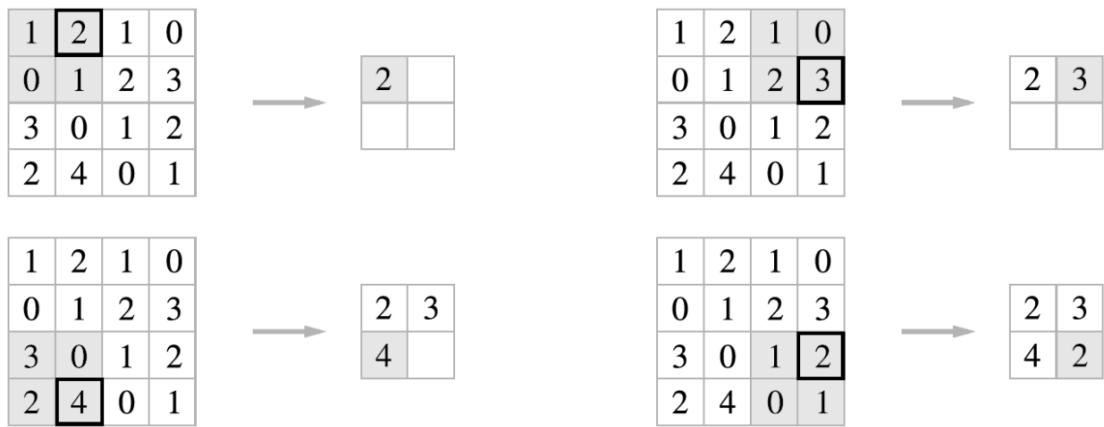
- 블러링을 통한 노이즈 제거(가우시안 블러링)
- 픽셀 기울기(gradiant)의 강도와 방향 검출(소벨 마스크)
- 비최대치 억제(non-maximum suppression)
- 이력 임계값(hysteresis threshold)으로 에지 결정

## 비선형 공간 필터링

mask - 출력픽셀로 결정.

pooling과 흡사

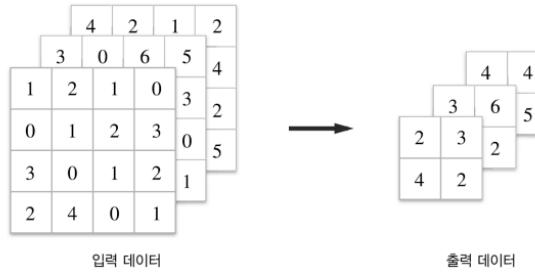
풀링 계층



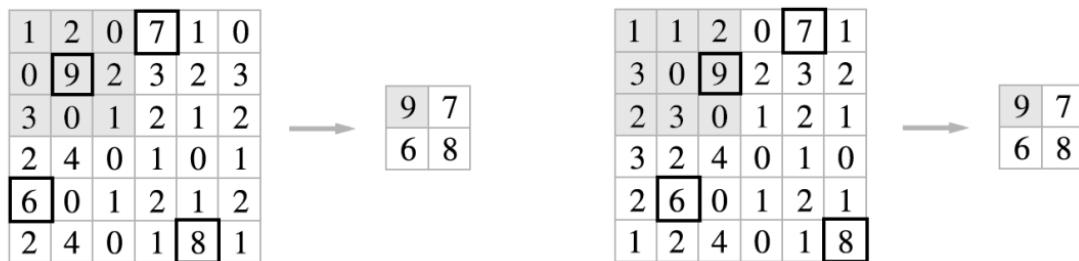
- 세로 · 가로 방향의 공간을 줄이는 연산.
  - 위의 예 경우  $2 \times 2$  최대 풀링(max pooling)을 스트라이드 2로 처리하는 순서.
  - 풀링의 윈도우 크기와 스트라이드는 같은 값으로 설정하는 것이 일반적.

## 풀링 계층의 특징

- 학습해야 할 매개변수가 없다.
  - 채널 수가 변하지 않는다.



- 입력의 변화에 영향을 적게 받는다(강건하다).



## CNN 의 네트워크 구조



## 08\_filter\_minmax

```

import numpy as np, cv2

def minmax_filter(image, ksize, mode):
    # mode - min인지 max인지 체크해줌
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2

    for i in range(center, rows - center):
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1
            x1, x2 = j - center, j + center + 1
            mask = image[y1:y2, x1:x2]
            dst[i, j] = cv2.minMaxLoc(mask)[mode]

    return dst

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

minfilter_img = minmax_filter(image, 3, 0) # 3 by 3으로 찾아줌 # 최소값은 0
maxfilter_img = minmax_filter(image, 3, 1)
# 행렬을 3 3 다 넣어줄 필요 없다.
# 최대값은 1

cv2.imshow("image", image)
cv2.imshow("minfilter_img", minfilter_img)
cv2.imshow("maxfilter_img", maxfilter_img)

cv2.waitKey(0)

```



## 09\_filter\_average

```
import numpy as np, cv2

def average_filter(image, ksize):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2

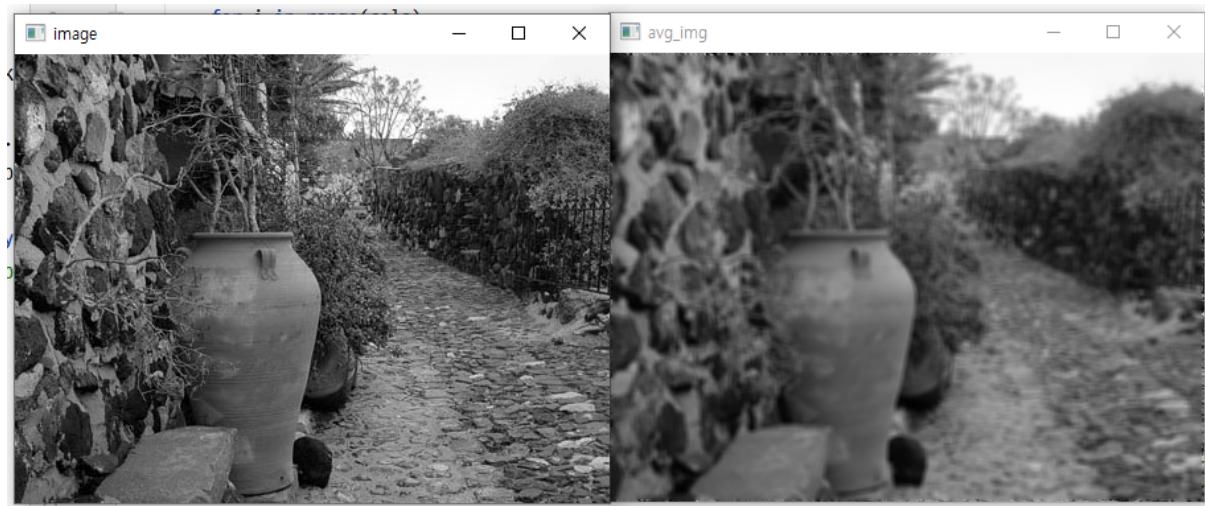
    for i in range(rows):
        for j in range(cols):
            y1, y2 = i - center, i + center + 1
            x1, x2 = j - center, j + center + 1
            if y1 < 0 or y2 > rows or x1 < 0 or x2 > cols:
                dst[i, j] = image[i, j]
            else:
                mask = image[y1:y2, x1:x2]
                dst[i, j] = cv2.mean(mask)[0]

    return dst

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

avg_img = average_filter(image, 5) # 5 by 5

cv2.imshow("image", image)
cv2.imshow("avg_img", avg_img)
cv2.waitKey(0)
```



평균은 부드럽게.

### 평균값 필터링

- ✓ 실행 결과
- ❑ borderType

옵션 상수	값	설명
cv2.BORDER_CONSTANT	0	특정 상수값으로 대체 70 70 70 20 25 35 45 50 60 70 70 70
cv2.BORDER_REPLICATE	1	계산 가능한 경계의 출력 화소 하나만으로 대체 대체 화소 20 20 20 20 25 35 45 50 60 60 60 60 대체 화소
cv2.BORDER_REFLECT	2	계산 가능한 경계의 출력 화소로부터 대칭되어 한 화소씩 지정 대체 화소 35 25 20 20 25 35 45 50 60 60 50 45 대체 화소
cv2.BORDER_WRAP	3	영상의 왼쪽 끝과 오른쪽 끝이 연결되어 있다고 가정하여 한 화소씩 가져와서 지정 대체 화소 45 50 60 20 25 35 45 50 60 20 25 35 대체 화소

10\_filter\_median.py

```
import numpy as np, cv2

def median_filter(image, ksize):
    rows, cols = image.shape[:2]
    dst = np.zeros((rows, cols), np.uint8)
    center = ksize // 2

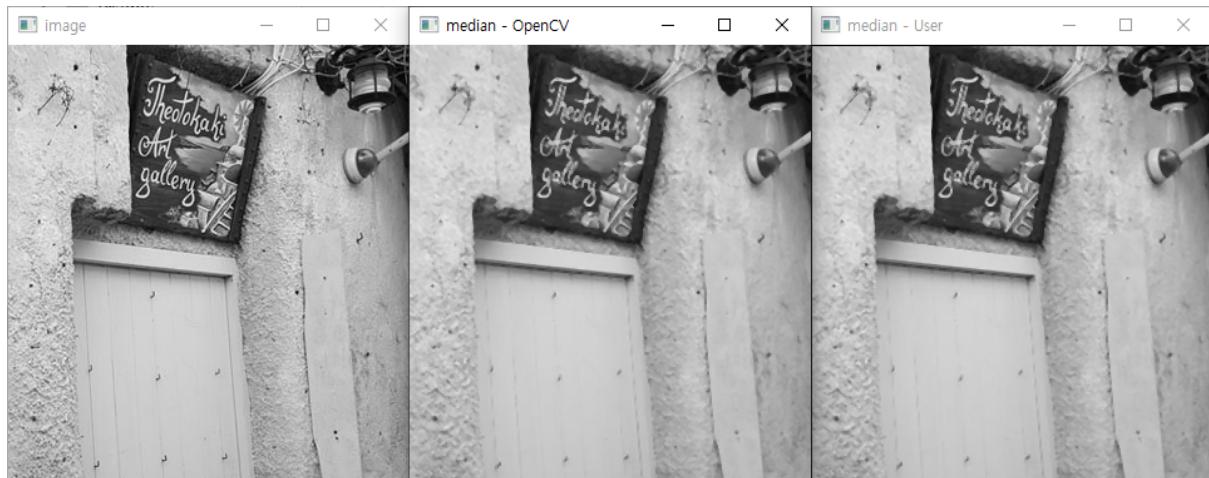
    for i in range(center, rows - center):
        for j in range(center, cols - center):
            y1, y2 = i - center, i + center + 1
            x1, x2 = j - center, j + center + 1
            mask = image[y1:y2, x1:x2].flatten()

            sort_mask = cv2.sort(mask, cv2.SORT_EVERY_COLUMN)
            dst[i, j] = sort_mask[sort_mask.size//2]
    return dst

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

med_img1 = median_filter(image, 3)                                # 사용자 정의 함수
med_img2 = cv2.medianBlur(image, 3)                                 # OpenCV 제공 함수

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



## 모폴로지(Morphology)

침식은 아주 점같은 것들은 오브젝트로 인식되기 때문에 작은 크기는 없앰.

- 객체의 크기는 축소되고 배경은 확장됨
- 객체의 크기가 축소되기 때문에 영상 내에 존재하는 잡음 같은 작은 크기의 객체들은 사라질 수도 있음

## 11\_erosion

```
import numpy as np, cv2

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

data = [0, 1, 0,
        1, 1, 1,
        0, 1, 0]

mask = np.array(data, np.uint8).reshape(3,3)
th_img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)[1]
# 진짜 흑백영상을 만들어서 담아줌
# 128보다 작은 값은 0으로 셋팅, 255보다 큰 값은 255

dst = cv2.erode(th_img, mask)

cv2.imshow("image", image)
cv2.imshow("erode", dst)

cv2.waitKey(0)
```



## 모폴리지 연산

### ✓ 팽창 연산

- ❑ 객체의 가장 외곽 픽셀을 확장시키기 때문에 객체의 크기는 확대되고 배경은 축소되며 객체의 팽창으로 인해 서 객체 내부에 있는 빈 공간도 메워지게 됨

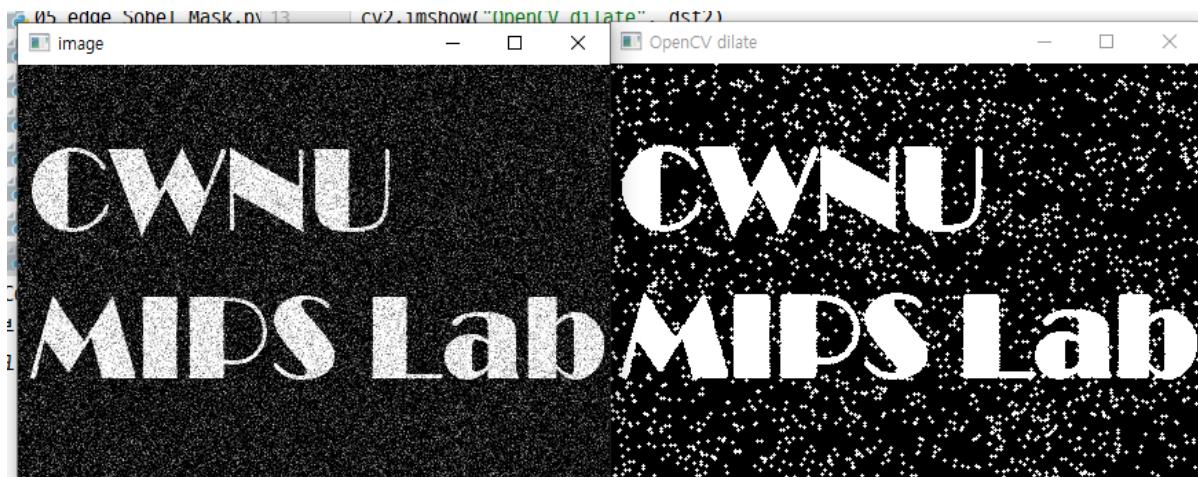
## 12\_dilate.py

```
12_dilate.py ✘
import numpy as np, cv2

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

mask = np.array([[0, 1, 0],
                 [1, 1, 1],
                 [0, 1, 0]]).astype("uint8")
th_img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)[1]
dst2 = cv2.morphologyEx(th_img, cv2.MORPH_DILATE, mask) # 팽창 수행

cv2.imshow("image", image)
cv2.imshow("OpenCV dilate", dst2)
cv2.waitKey(0)
```



## 열림 연산(opening operation)

- 침식 연산을 먼저 수행하고 바로 팽창 연산을 수행하는데 침식 연산으로 인해서 객체는 축소되고 배경 부분의 미세한 잡음들은 제거되고 다음으로 축소되었던 객체들이 팽창 연산으로 인해서 다시 원래 크기로 돌아감

## 닫힘 연산(closing operation)

- 팽창 연산을 먼저 수행하고 다음으로 침식 연산을 수행

### 12\_dilate.py

```
import numpy as np, cv2

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

mask = np.array([[0, 1, 0],
                 [1, 1, 1],
                 [0, 1, 0]]).astype("uint8")
th_img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)[1]
dst2 = cv2.morphologyEx(th_img, cv2.MORPH_DILATE, mask) # 팽창 수행
dst3 = cv2.morphologyEx(th_img, cv2.MORPH_OPEN, mask) # 열림 연산
dst4 = cv2.morphologyEx(th_img, cv2.MORPH_CLOSE, mask) # 닫힘 연산

cv2.imshow("image", image)
cv2.imshow("OpenCV dilate", dst2)
cv2.imshow("OpenCV opening", dst3)
cv2.imshow("OpenCV closing", dst4)
cv2.waitKey(0)
```

