스마트홈 IoT가전기기 제어를 위한 객체인식 모델구현

사전

딥러닝을 위한 파이썬 프로그래밍과 영상 처리 개념 이해(11)

KETI 한국전자기술연구원
Korea Electronics Technology Institute

학습내용

- Convolution 연산
- 경계선(Edge) 검출
- 노이즈 제거

학습목표

- 경계선 검출 알고리즘의 종류와 특징을 설명할 수 있다.

Convolution 연산

Convolution 연산

화소 기반 처리

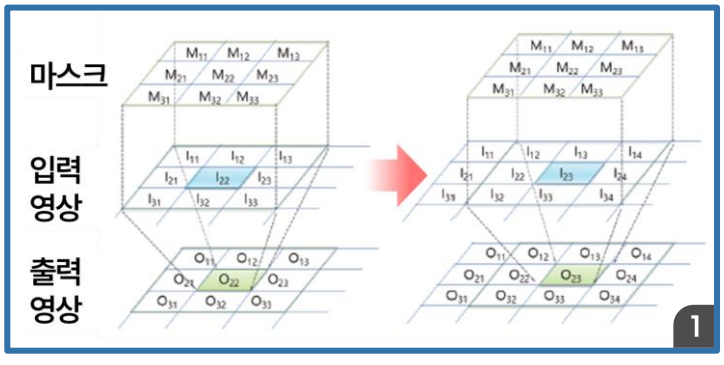
- 화소 값 각각에 대해 여러 가지 연산 수행

영역 기반 처리

- 마스크(Mask)라 불리는 규정된 영역을 기반으로 연산 수행
  → 커널(Kernel), 윈도우(Window), 필터(Filter)

Convolution 연산

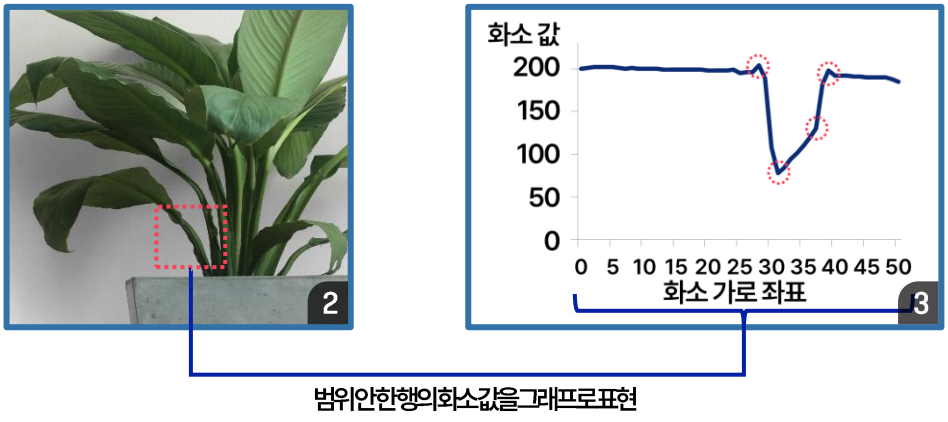
- 마스크(Mask)와 입력 영상을 산술 연산하는 방법
  → 동일한 위치의 마스크와 입력 영상을 곱함
- 연산 한 번에 스칼라 값 하나가 만들어짐
- 마스크를 옮겨서 영상 전체에 적용



경계선(Edge) 검출

경계선(Edge) 검출

에지 검출



→ 경계선 검출 알고리즘

1차 미분 마스크

- 미분
  - → 함수의 순간 변화율을 구하는 계산 과정
  - → 에지가 화소의 밝기가 급격히 변하는 부분
  - → 함수의 변화율을 취하는 미분 연산을 이용해서 에지 검출 가능

프리윗(Prewitt) 마스크

- 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

수직마스크

$G_y =$

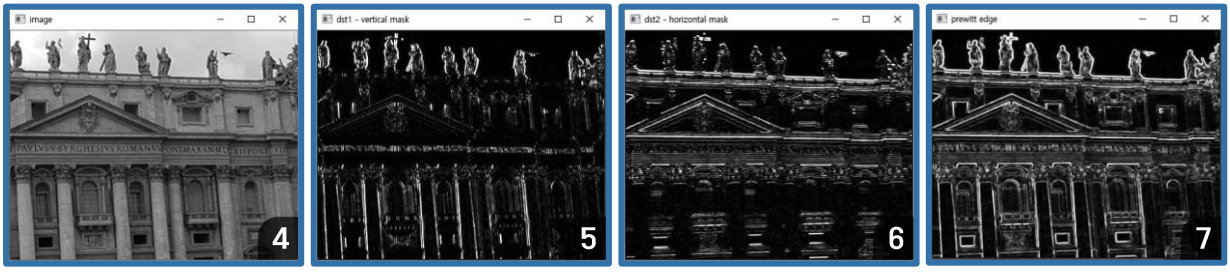
-1	-1	-1
0	0	0
1	1	1

수평마스크

경계선(Edge) 검출

프리윗(Prewitt) 마스크

- 실행 결과



입력영상 수직방향에지검출영상 수평방향에지검출영상 프리윗마스크출력영상

소벨(Sobel) 마스크

- 프리윗 마스크와 유사
- 중심 화소의 차분에 대한 비중을 2배 키운 것이 특징

  → 수직, 수평 방향 에지 추출
  → 중심 화소의 차분 비중을 높여 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

- OpenCV 라이브러리

  → Sobel 함수
  → 매개변수(Parameter)를 이용하여 X축, Y축 방향 결정

예제 7.2.5

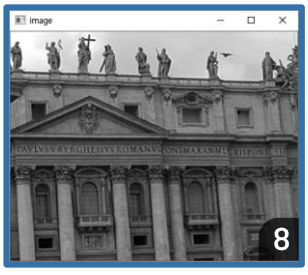
```
01  Import numpy as np, cv2
02  Form Commonfilters import differential

16  dst3 = cv2.Sobel(np.float32(image), cv2.CV_32F, 1, 0, 3)
17  dst4 = cv2.Sobel(np.float32(image), cv2.CV_32F, 0, 1, 3)
18  dst3 = cv2.convertScaleAbs(dst3)
19  dst4 = cv2.convertScaleAbs(dst4)
```

두방향에지검출함수
소벨수행
X방향미분
Y방향미분

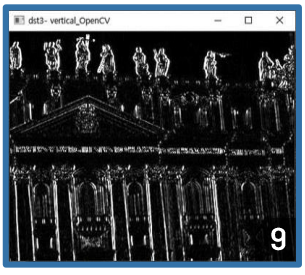
경계선(Edge) 검출

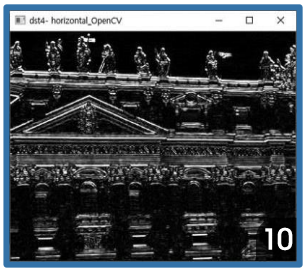
소벨(Sobel) 마스크

- 실행 결과



입력영상





소벨마스크출력영상

라플라시안 마스크

- 최종 수식

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

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

0	1	0
1	-4	1
0	1	0

4방향마스크

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

1	1	1
1	-8	1
1	1	1

8방향마스크

- OpenCV 라이브러리

→ Laplacian 함수 이용

예제 7.2.6

4방향라플라시안마스크원소

8방향라플라시안 마스크원소

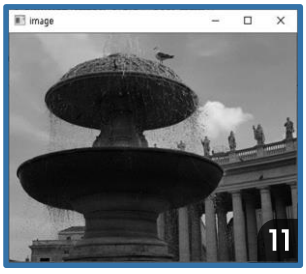
라플라시안수행함수

```
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)
```

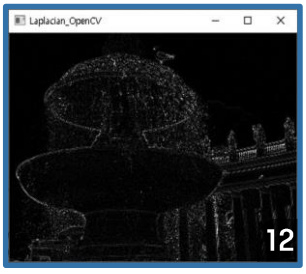
경계선(Edge) 검출

◆ 라플라시안 마스크

- 실행 결과



입력영상



출력영상

◆ 캐니 에지 검출

- 최근 가장 많이 사용하는 검출 방법

① 블러링을 통한 노이즈 제거(가우시안 블러링)
② 화소 기울기(Gradiant)의 강도와 방향 검출(소벨 마스크)
③ 비최대치 억제(Non-maximum Suppression)
④ 이력 임계값(Hysteresis Threshold)으로 에지 결정

- 잡음은 다른 부분과 경계를 이루는 경우 많음

→ 대부분의 에지 검출 방법이 잡음을 에지로 검출
→ 이런 문제를 보안하는 방법

① 블러링을 통한 노이즈 제거(가우시안 블러링)

→ 5×5 크기의 가우시안 필터 적용

- 불필요한 잡음 제거
- 필터 크기는 변경 가능

② 화소 기울기(Gradiant)의 강도와 방향 검출(소벨 마스크)

→ 가로 방향과 세로 방향의 소벨 마스크로 회선 적용
→ 화소 기울기의 크기(magnitude)와 방향(direction) 계산
→ 기울기 방향은 4개 방향(0, 45, 90, 135)으로 근사하여 단순화

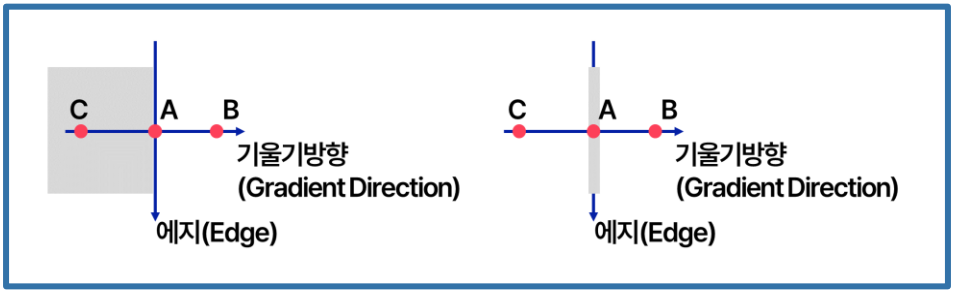
- 반대편 180 ° 포함

경계선(Edge) 검출

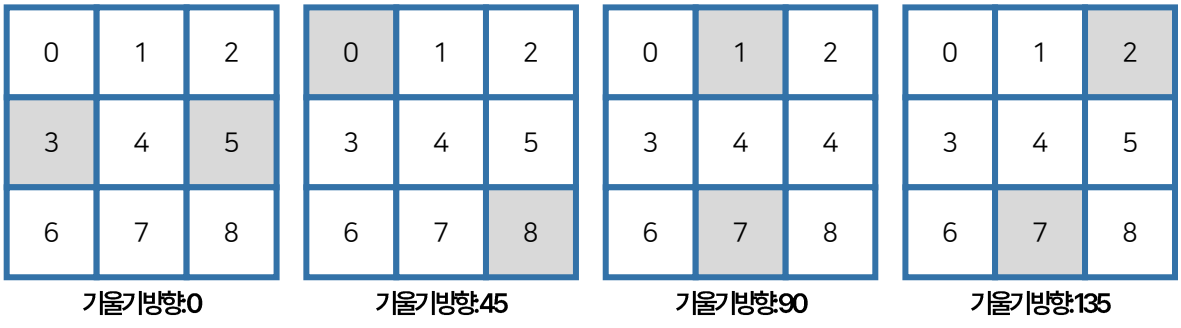
◆ 캐니 에지 검출

③ 비최대치 억제(Non-maximum Suppression)

→ 기울기의 방향과 에지의 방향은 수직
→ 최대치만 검출



→ 기울기의 방향에 따른 이웃 화소 선택



④ 이력 임계값(Hysteresis Threshold)으로 에지 결정

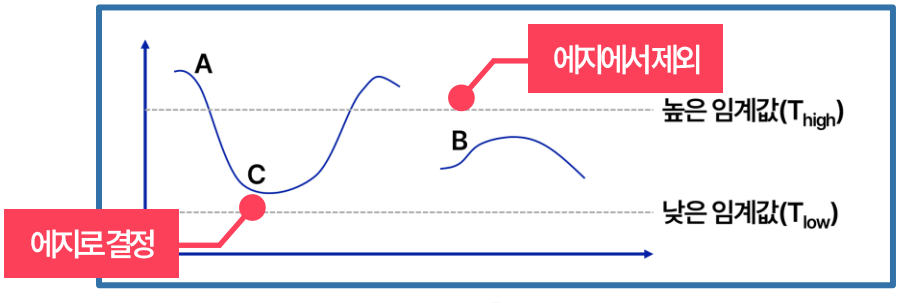
→ 임계값

- 값의 사용 여부를 판단하는 기준이 되는 값

→ 하나의 임계값으로 에지를 검출하면 잡음 발생
→ 임계값을 너무 높게 잡으면 에지가 검출되지 않을 수 있음
→ 두 개의 임계값(Thigh, Tlow)을 사용해 에지 이력 추적
→ 각 화소에서 높은 임계값보다 크면 에지 추적 시작

- 추적하면 추적하지 않은 이웃 화소로 낮은 임계값보다 큰 화소를 에지로 결정

→ 임계값 사이에 있는 값은 어떻게 판단할까?



경계선(Edge) 검출

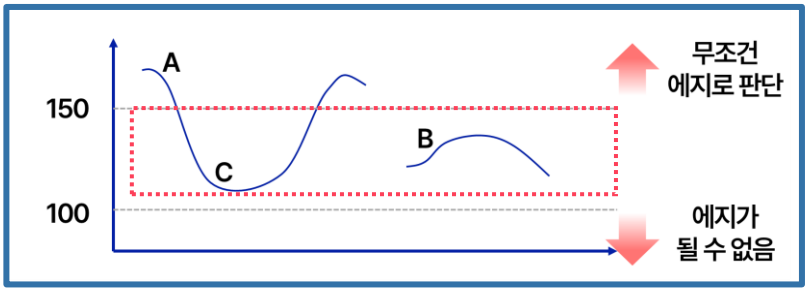
캐니 에지 검출

- OpenCV 라이브러리

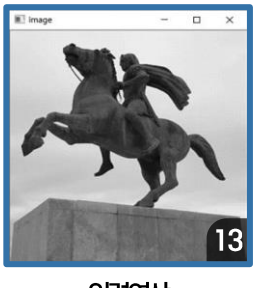
  → Canny 함수 이용
  → 매개변수(Parameter)에 낮은 임계값과 높은 임계값 입력

```
49   image = cv2.imread("images/canny.jpg", cv2.IMREAD_GRAYSCALE)

56   gaus_img = cv2.GaussianBlur(image, (5, 5), 0.3)
57   Gx = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 1, 0, 3)
58   Gy = cv2.Sobel(np.float32(gaus_img), cv2.CV_32F, 0, 1, 3)
59   soble = cv2.magnitude(Gx, Gy)

66   canny2 = cv2.Canny(image, 100, 150)
```

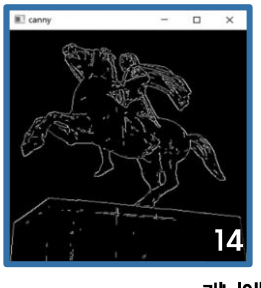
낮은임계값 높은임계값

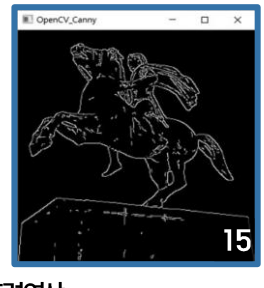


- 실행 결과



입력영상

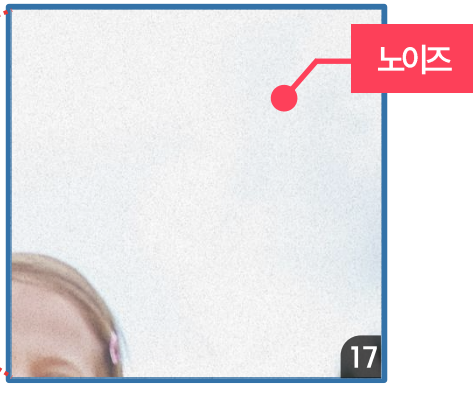




캐니에지출력영상

- 노이즈 제거 필요

  → 가우시안 필터 사용

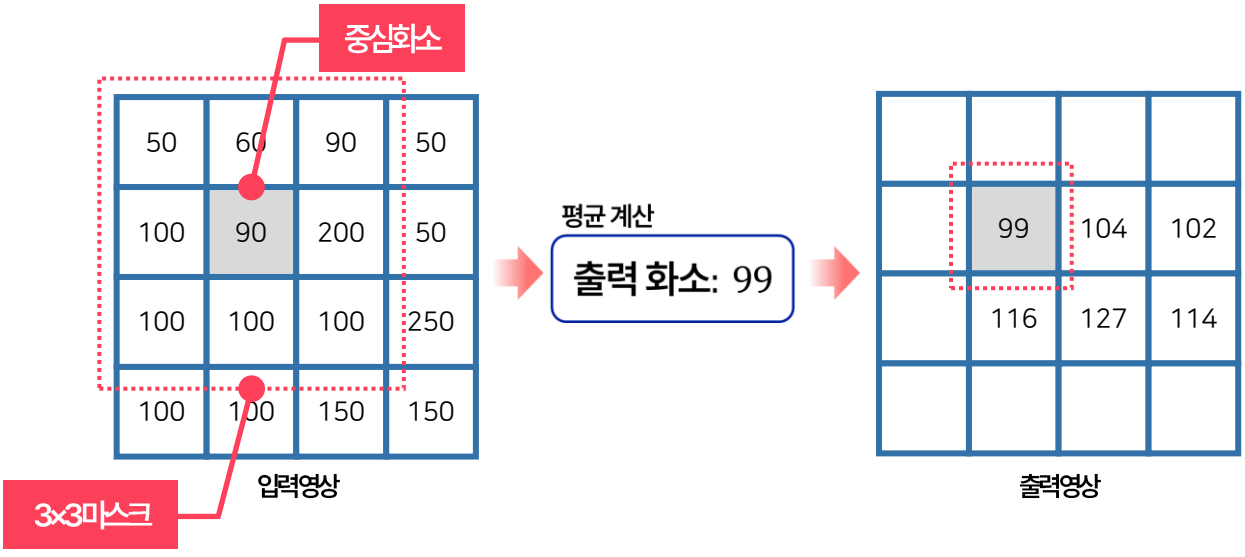


노이즈 제거

노이즈제거

◆ 평균값 필터링

- 마스크 영역 입력 화소들의 평균을 구하여 출력 화소로 지정하는 방법

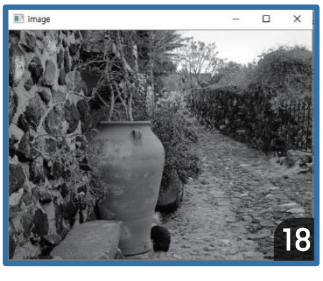


- OpenCV 라이브러리

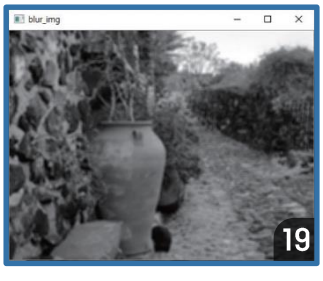
  → blur 함수 이용

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

22  avg_img = average_filter(image, 5)
23  blur_img = cv2.blur(image, (5, 5))
24  box_img = cv2.boxFilter(image, ddepth = -1, ksize = (5, 5))
```

- 실행 결과



입력영상

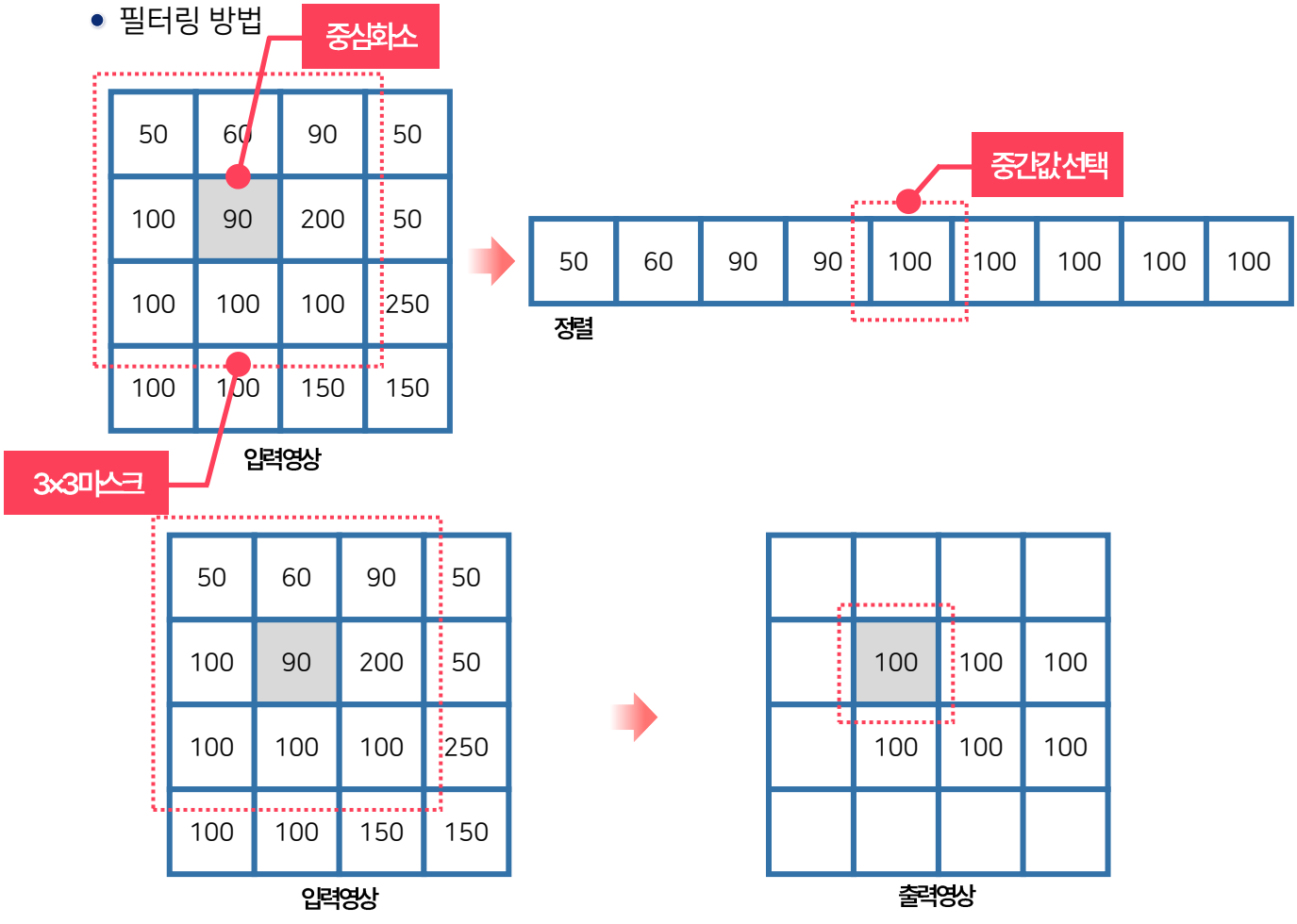


평균값필터링출력영상

노이즈제거

미디언 필터링

- 마스크 범위 원소 중 중간값 취하여 출력 화소로 결정하는 방식
  - → 마스크 범위 내에 있는 화소 값 정렬 필요
  - → 임펄스 잡음, 소금-후추 잡음 제거
- 필터링 방법



- OpenCV 라이브러리
  - → medianBlur 함수 이용

예제 7.3.3

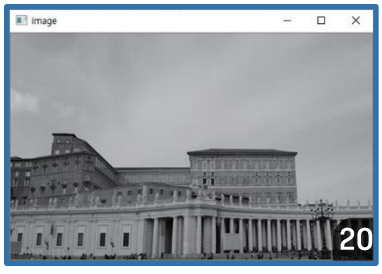
```
25  image = cv2.imread("images/median.jpg", cv2.IMREAD_GRAYSCALE)
26  if image is None: raise Exception("영상파일 읽기 오류")

29  noise = salt_papper_noise(image, 500)
30  med_img1 = median_filter(noise, 5)
31  med_img2 = cv2.medianBlur(noise, 5)
```

5x5 크기 미디언 필터링

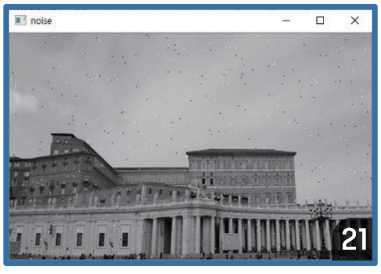
노이즈제거

미디언 필터링

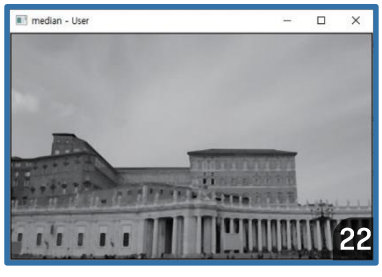
- 실험 결과



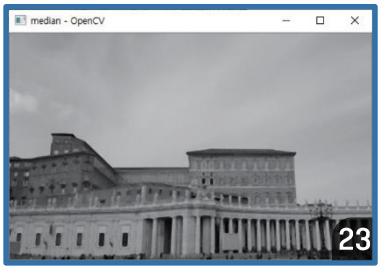
입력영상



임의노이즈추가영상



소금-후추잡음제거



미디언필터링결과영상

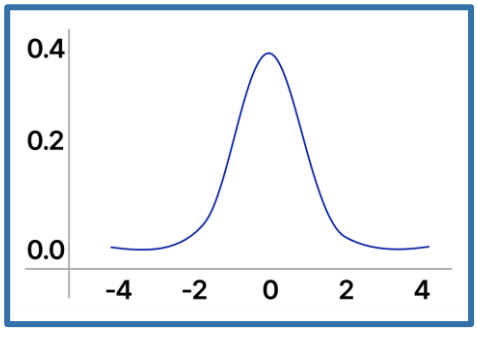
가우시안 스무딩 필터링

- 스무딩

  → 회선을 통해 영상의 세세한 부분을 부드럽게 하는 기법
  → 가우시안 필터링이 대표적인 방법

- 가우시안 분포(정규 분포)

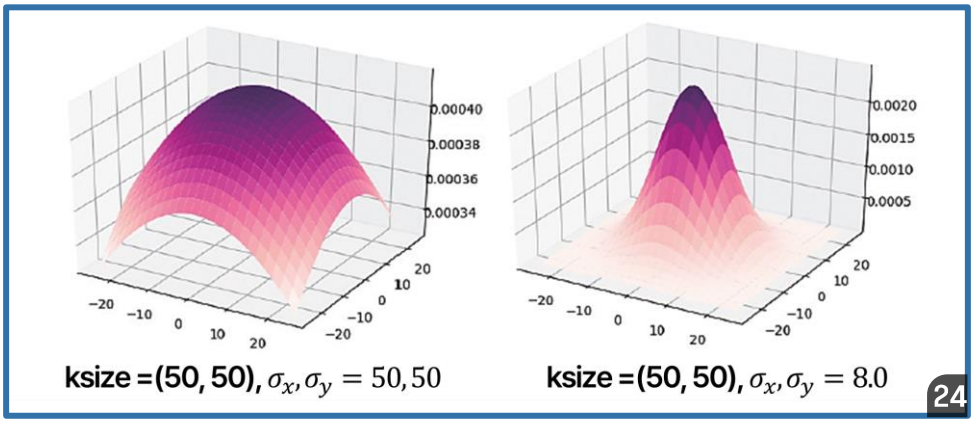
  → 특정 값의 출현 비율을 그래프로 그렸을 때, 평균에서 가장 큰 수치 가짐
  → 평균을 기준으로 좌우 대칭 형태
  → 양 끝으로 갈수록 수치가 낮아지는 종 모양



노이즈제거

◆ 가우시안 스무딩 필터링

- 2차원 가우시안 분포

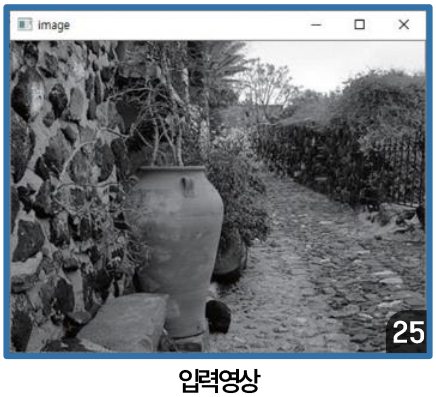


- OpenCV 라이브러리

→ GaussianBlur 함수 이용

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

22  ksize = (17, 5)
23  gaussian_2d = getGaussianMask(ksize, 0, 0)
24  gaussian_1dX = cv2.getGaussianKernel(ksize[0], 0, cv2.CV_32F)
25  gaussian_1dY = cv2.getGaussianKernel(ksize[1], 0, cv2.CV_32F)

27  gauss_img1 = cv2.filter2D(image, -1, gaussian_2d)
28  gauss_img2 = cv2.GaussianBlur(image, size, 0)
29  gauss_img3 = cv2.sepFilter2D(image, -1, gaussian_1dX, gaussian_1dY)
```

- 실험 결과



입력영상

출력영상

정리하기

- **Convolution 영상**
  - → 마스크(Mask)와 입력 영상을 산술 연산하는 방법
- **경계선(Edge) 검출**
  - → 프리윗(Prewitt) 마스크
  - → 소벨(Sobel) 마스크
  - → 라플라시안 마스크
  - → 캐니 에지 검출
- **노이즈 제거**
  - → 평균값 필터링
  - → 미디언 필터링
  - → 가우시안 스무딩 필터링