슈티어링 3주차

목차

- **영상의 산술 및 논리 연산**
  영상의 산술 연산, 영상의 논리 연산
- **필터링**
  영상의 필터링(필터링 연산 방법, 엠보싱 필터링)
- **실습**

영상의 산술 연산

영상의 산술 연산

- 영상 ≒ 2차원 행렬 → 행렬의 산술 연산 그대로 적용 가능
  - 서로 더하거나 빼는 연산을 수행하여 새로운 결과 영상 생성
    (곱셈 나눗셈은 거의 사용 X)

덧셈 연산

- 두 개의 입력 영상에서
- 같은 위치 픽셀 값을
- 서로 더하여
- 결과 영상 픽셀 값으로 설정하는 연산

$$dst(x,y) = src1(x,y) + src2(x,y)$$

결과 영상 입력 영상

결과값 그레이스케일 255(최댓값)보다 커지는 경우 발생

$$dst(x,y) = saturate(src1(x,y)) + src2(x,y)$$

결과 영상 픽셀 값을 255로 설정하는 포화 연산

덧셈 연산 = add()함수 사용 가능

void add(InputArray src1, InputArray src2, OutputArray dst,
InputArray mask = noArray(), int dtype = -1);

src1	첫번째 입력 행렬 또는 스칼라
src2	두번째 입력 행렬 또는 스칼라
dst	• 입력 행렬과 같은 크기, 같은 채널 수를 갖는 출력 행렬 • dst의 깊이는 src1, src2의 깊이와 같거나 또는 dtype 인자에 의해 결정
mask	• 8비트 1채널 마스크 영상 • mask 행렬 원소 값이 0이 아닌 위치에서만 덧셈 연산을 수행
dtype	• 출력 행렬의 깊이 • src1과 src2의 깊이가 같은 경우: dtype에 -1을 지정할 수 있음 → dst의 깊이= src1, src2 • src1과 src2의 깊이가 서로 다른 경우: dtype을 반드시 지정

덧셈 연산

- add() 함수 예시

```
Mat src1 = imread("aero2.bmp", IMREAD_GRAYSCALE);
Mat src2 = imread("camera.bmp", IMREAD_GRAYSCALE);
//두 입력 영상의 타입 모두 CV_8UC1

Mat dst;
add(src1, src2, dst);
//두 입력 영상의 타입이 서로 같음 → dtype 인자 따로 지정하지 않아도 됨 (결과도 타입 동일)
//mask 인자를 따로 지정하지 않았음 → 두 영상의 모든 픽셀 위치에서 덧셈 연산 수행
```

덧셈 연산

- 덧셈 연산의 두 입력 영상 타입이 같다면 add() 함수 대신 + 연산자 재정의를 사용 가능

```
Mat src1 = imread("aero2.bmp", IMREAD_GRAYSCALE);
Mat src2 = imread("camera.bmp", IMREAD_GRAYSCALE);

Mat dst = src1 + src2;
```

덧셈 연산 결과

- 특징
  - 두 입력 영상의 윤곽을 조금씩 포함
  - 전반적으로 밝게 포화되는 부분이 많음

가중치를 부여하여 덧셈 연산

$$dst(x,y) = saturate(\alpha \cdot src1(x,y)) + \beta \cdot src2(x,y)$$

가중치

- (대부분) α+β=1 : 결과 영상에서 포화되는 픽셀이 발생 X
  Ex) α=0.1, β=0.9 : src1 영상의 윤곽↓, src2 영상의 윤곽 ↑나타나는 결과 영상 생성

- A=β=0.5 : 두 입력 영상의 윤곽을 골고루 가지는 평균 영상 생성

- α+β>1 : 결과 영상이 두 입력 영상보다 밝아지고, 포화 현상 (덧셈의 결과 > 255) 발생할 수 있음

- α+β<1 : dst 영상은 두 입력 영상의 평균 밝기보다 어두운 결과 영상이 생성

두 영상의 가중치 합 = addWeighted() 함수

```
void addWeighted(InputArray src1, double alpha, InputArray src2, double beta,
                 double gamma, OutputArray dst, int dtype = -1);
```

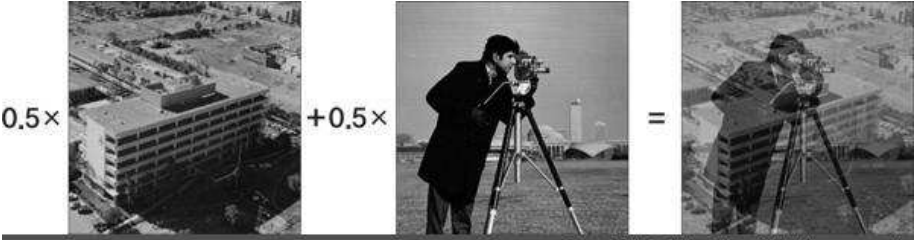
src1	첫 번째 입력 행렬
alpha	src1 행렬의 가중치
src2	두 번째 입력 행렬. src1과 크기와 채널 수가 같아야 함
beta	src2 행렬의 가중치
gamma	가중합 결과에 추가적으로 더할 값
dst	• 출력 행렬 • 입력 행렬과 같은 크기, 같은 채널 수의 행렬이 생성
dtype	출력 행렬의 깊이 (이하 add()와 동일)

addWeighted() 함수에 의해 생성되는 dst

$$dst(x,y) = saturate(srcl(x,y)*alpha + srcl2(x,y)*beta + gamma)$$

평균 연산 결과

- 특징
  - 입력 영상의 윤곽 골고루 포함
  - 평균 밝기가 그대로 유지



뺄셈 연산

- 두 개의 입력 영상에서
- 같은 위치 픽셀 값을
- 빼기 연산 수행

=> 뺄셈의 대상이 되는 영상 순서에 따라 결과가 달라짐

$$dst(x,y) = src1(x,y) - src2(x,y)$$

결과 영상 입력 영상

결과값 그레이스케일 0(최솟값)보다 작아지는 경우 발생

$$dst(x,y) = saturate(src1(x,y)) - src2(x,y)$$

결과 영상 픽셀 값을 0으로 설정하는 포화 연산

뺄셈 연산 = subtract ()함수 사용 가능

void subtract (InputArray src1, InputArray src2, OutputArray dst, InputArray mask = noArray(), int dtype = -1);

src1	첫번째 입력 행렬 또는 스칼라
src2	두번째 입력 행렬 또는 스칼라
dst	• 입력 행렬과 같은 크기, 같은 채널 수를 갖는 출력 행렬 • dst의 깊이는 src1, src2의 깊이와 같거나 또는 dtype 인자에 의해 결정
mask	• 8비트 1채널 마스크 영상 • mask 행렬 원소 값이 0이 아닌 위치에서만 덧셈 연산을 수행
dtype	• 출력 행렬의 깊이 • src1과 src2의 깊이가 같은 경우: dtype에 -1을 지정할 수 있음 → dst의 깊이= src1, src2 • src1과 src2의 깊이가 서로 다른 경우: dtype을 반드시 지정

*add()함수와 동일

두 입력 영상의 평균 영상 생성

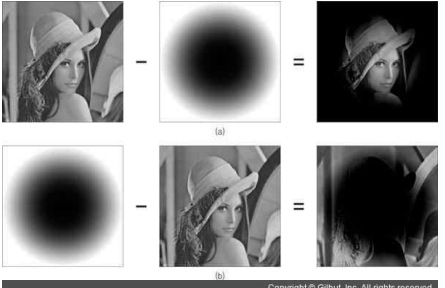
- addWeighted() 함수 예시

```
Mat src1 = imread("aero2.bmp", IMREAD_GRAYSCALE);
Mat src2 = imread("camera.bmp", IMREAD_GRAYSCALE);

Mat dst; addWeighted(src1, 0.5, src2, 0.5, 0, dst);
//addWeighted() 함수에 가중치 모두 0.5로 지정, 추가적으로 더하는 값 0으로 지정
```

차이 연산

$$dst(x,y) = |src1(x,y) + src2(x,y)|$$

- 뺄셈 연산 결과에 절댓값을 취하는 연산
- 결과 영상= 차영상(difference image)



차이 연산 = absdiff ()함수 사용 가능

```
void absdiff (InputArray src1, InputArray src2, OutputArray dst);
```

src1	첫번째 입력 행렬 또는 스칼라
src2	두번째 입력 행렬 또는 스칼라
dst	• 출력 행렬 • 입력 행렬과 같은 크기, 같은 채널 수의 행렬 생성

차이 연산의 이용

- 두 개의 영상에서 변화가 있는 영역을 쉽게 찾을 수 있음

정적인 배경 영상

동정적인 배경 영상

픽셀 값 차이 두드러지게 나타남
(두 입력 영상에서 큰 변화가 없는 영역
=픽셀 값이 0에 가까운 검은색)

곱셈 연산 = multiply ()함수 사용 가능

void multiply (InputArray src1, InputArray src2, OutputArray dst, InputArray scale = 1, int dtype = -1);

src1	첫번째 입력 행렬
src2	두번째 입력 행렬 (src1과 크기와 타입이 같아야함)
dst	• src1과 같은 크기, 같은 타입인 출력 행렬 • dst(x, y) = saturate(scale·src1(x, y)·src2(x, y))
scale	추가적으로 확대/축소할 비율
dtype	출력 행렬의 깊이

*add()함수와 동일

나눗셈 연산 = divide ()함수 사용 가능

void divide (InputArray src1, InputArray src2, OutputArray dst,
InputArray scale = 1, int dtype = -1);

src1	첫번째 입력 행렬
src2	두번째 입력 행렬 (src1과 크기와 타입이 같아야함)
dst	• src1과 같은 크기, 같은 타입인 출력 행렬 • dst(x, y) = saturate(scale·src1(x, y)·src2(x, y))
scale	추가적으로 확대/축소할 비율
dtype	출력 행렬의 깊이

*multiply()함수와 동일

영상의 논리 연산

영상의 논리 연산

- 픽셀 값을 이진수로 표현하여 각 비트(bit) 단위 논리 연산을 수행하는 것

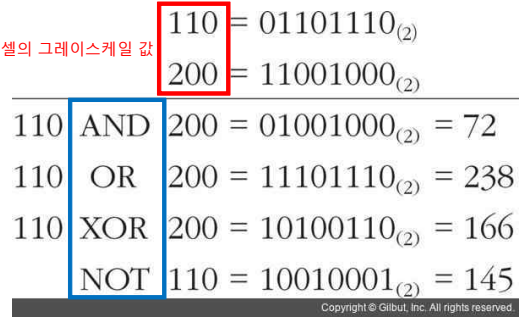
<OpenCV에서 제공하는 논리 연산 진리표>

입력 비트		논리 연산 결과			
a	b	a AND b	a OR b	a XOR b	NOT a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

영상의 논리 연산

- 각 픽셀 값에 대하여 비트 단위로 이루어짐
- 그레이스케일 영상 :한 픽셀을 구성하는 여덟 개의 비트에 모두 논리 연산이 이루어짐

특정 좌표에 있는 픽셀의 그레이스케일 값



- 비트 단위 논리곱= bitwise_and ()함수
- 비트 단위 논리합= bitwise_or ()함수
- 비트 단위 배타적 논리합= bitwise_xor ()함수

```
void ~ (InputArray src1, InputArray src2,
        OutputArray dst, InputArray mask = noArray());
```

*2개의 인자

- 비트 단위 부정 연산= bitwise_not ()함수

```
void bitwise_not (InputArray src1, OutputArray dst,
                  InputArray mask = noArray());
```

*1개의 인자

```
void ~ (InputArray src1, InputArray src2,
        OutputArray dst, InputArray mask = noArray());
```

src1	첫번째 입력 행렬 또는 스칼라
src2	두번째 입력 행렬 또는 스칼라 (src1과 크기와 타입이 같아야함)
dst	• src1과 같은 크기, 같은 타입인 출력 행렬 • dst 행렬 원소 값은 논리 연산 종류에 의해 각각 다르게 결정
mask	• 마스크 영상 • mask 영상의 픽셀 값≠0 : 위치에서만 논리 연산을 수행하도록 설정 가능 • mask 인자 따로 지정X / noArray() 또는 Mat() = mask인자 설정 : 영상 전체에 대해 논리 연산 수행

필터링

필터링 연산 방법

영상 처리에서 필터링

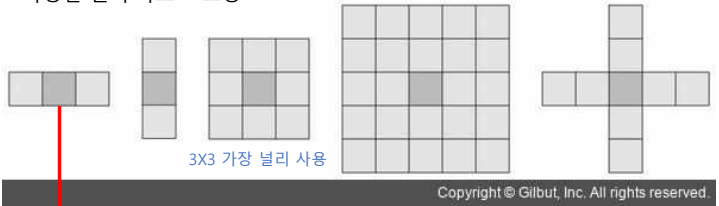
- 영상에서 원하는 정보만 통과시키고 원치 않는 정보는 걸러 내는 작업
  Ex) 지저분한 잡음(noise)을 걸러 내어 영상을 깔끔하게 만드는 필터
  Ex) 부드러운 느낌의 성분을 제거함으로써 영상을 좀 더 날카로운 느낌으로 변경

- 마스크(mask)라고 부르는 작은 크기의 행렬을 이용

마스크

- 필터링의 성격을 정의하는 행렬
- 커널(kernel), 윈도우(window)라고도 부름
- 마스크 자체를 필터라고 부르기도 함

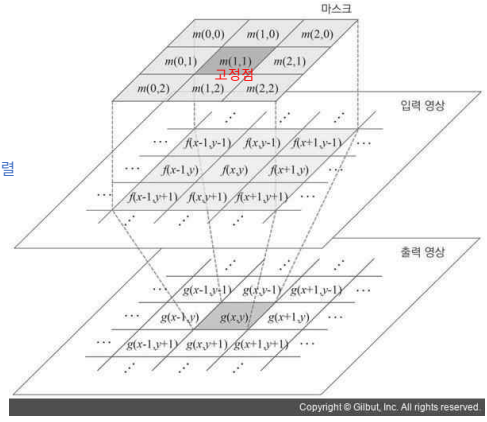
<다양한 필터 마스크 모양>



*고정점
:현재 필터링 작업을 수행 하고 있는 기준 픽셀 위치를 나타냄
:대부분의 경우 마스크 행렬 정중앙을 고정점으로 사용

3×3 정방형 마스크를 이용한 필터링 수행 방법

- M : 마스크 행렬
- F : 입력 영상
- G : 출력 영상



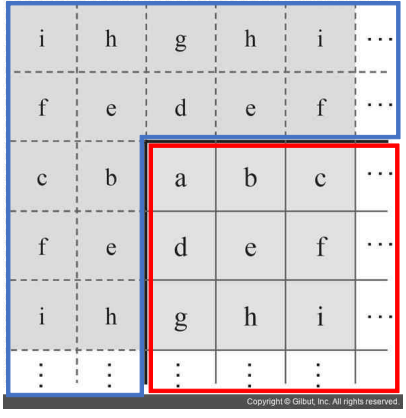
- 입력 영상의 모든 픽셀 위로
- 마스크 행렬을 이동시키면서
- 마스크 연산을 수행하는 방식

*마스크 연산
: (마스크 행렬 원소 값 X 같은 위치에 있는 입력 영상 픽셀)
마스크 행렬의 모든 원소에 대해 실행 후 결과 모두 더함

- 마스크 연산의 결과= 출력 영상에서 고정점 위치에 대응되는 픽셀 값으로 설정
- 그러므로 마스크 행렬 m의 중심이 입력 영상의 (x, y) 좌표 위에 위치했을 때 필터링 결과 영상의 픽셀 값 g(x, y)은 아래의 식과 같이 계산

$$g(x,y) = m(0,0)f(x-1,y-1) + m(1,0)f(x,y-1) + m(2,0)f(x+1,y-1) + m(0,1)f(x-1,y) + m(1,1)f(x,y) + m(2,1)f(x+1,y) + m(0,2)f(x-1,y+1) + m(1,2)f(x,y+1) + m(2,2)f(x+1,y+1)$$

가장자리 픽셀 확장 방법



- OpenCV는 영상의 필터링을 수행할 때, 영상의 가장자리 픽셀을 확장하여 영상 바깥쪽에 가상의 픽셀을 만듦

- 각각의 픽셀에 쓰여진 영문자 = 픽셀값
- 가상의 픽셀 위치에는 실제 영상의 픽셀 값이 대칭 형태로 나타나도록 설정

=>영상의 가장자리 픽셀에 대해서도 문제없이 필터링 연산을 수행

OpenCV 필터링에서 가장자리 픽셀 처리 방법

BorderTypes 열거형 상수	설명
BORDER_CONSTANT	
BORDER_REPLICATE	
BORDER_REFLECT	
BORDER_REFLECT_101	
BORDER_REFLECT101	BORDER_REFLECT_101과 같음
BORDER_DEFAULT	BORDER_REFLECT_101과 같음

필터링= filter2D ()함수 사용 가능

```
void divide (InputArray src1, OutputArray dst, int ddepth,
             InputArray kernel, Point anchor = Point(-1,-1),
             double delta = 0, int borderType = BORDER_DEFAULT);
```

src1	첫번째 입력 행렬
dst	출력 영상(src와 같은 크기, 같은 채널 수)
ddepth	결과 영상의 깊이
kernel	필터링 커널(1채널 실수형 행렬)
anchor	고정점 좌표 //Point(-1, -1)을 지정하면 커널 중심을 고정점으로 사용
delta	필터링 연산 후 추가적으로 더할 값
borderType	가장자리 픽셀 확장 방식

filter2D() 함수

- src 영상에 kernel 필터를 이용하여 필터링을 수행
- 결과 dst에 저장
- src 인자& dst 인자에 같은 변수 지정 : 필터링 결과 입력 영상에 덮어씀

<filter2D() 함수가 수행하는 연산>

$$\mathrm{dst}(x,y) = \sum_{j}\sum_{i} \mathrm{kernel}(i,j)\cdot \mathrm{src}(x+i-\mathrm{anchor.x}, y+j-\mathrm{anchor.y}) + \mathrm{delta}$$

입력 영상의 깊이에 따라 지정 가능한 ddepth 값

입력 영상의 깊이(src.depth())	지정 가능한 ddepth 값
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

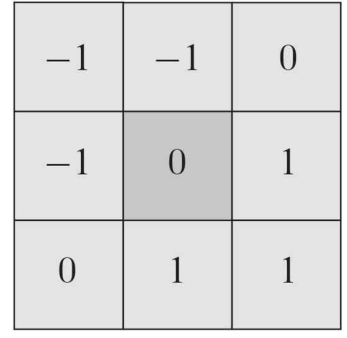
엠보싱 필터링

엠보싱 필터

*엠보싱 : 직물이나 종이, 금속판 등에 올록볼록한 형태로 만든 객체의 윤곽 또는 무늬

- 입력 영상을 엠보싱 느낌이 나도록 변환하는 필터
- 입력 영상에서 픽셀 값 변화가 적은 평탄한 영역은 회색으로 설정하고, 객체의 경계 부분은 좀 더 밝거나 어둡게 설정하면 엠보싱 느낌

엠보싱 필터 마스크

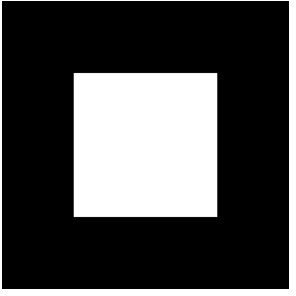


- 대각선 방향으로 +1 또는 -1의 값이 지정되어 있는 3×3 행렬
- 대각선 방향으로 픽셀 값이 급격하게 변하는 부분에서 결과 영상 픽셀 값이 0보다 훨씬 크거나 또는 0보다 훨씬 작은 값을 가짐
- 입력 영상에서 픽셀 값이 크게 바뀌지 않는 평탄한 영역에서는 결과 영상의 픽셀 값이 0에 가까운 값을 가짐
- 이렇게 구한 결과 영상을 그대로 화면에 나타내면 음수 값은 모두 포화 연산에 의해 0이 되어 버리기 때문에 입체감이 크게 줄어듬

=> 엠보싱 필터를 구현할 때에는 결과 영상에 128을 더하는 것이 보기에 좋음

슈티어링 4주차 실습 1

영상의 산술 연산 수행



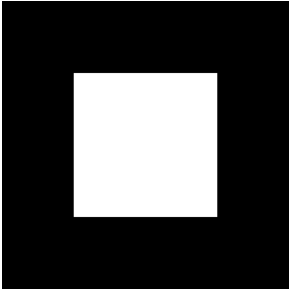
```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(void)
{
	Mat src1 = imread("lenna256.bmp", IMREAD_GRAYSCALE);
	Mat src2 = imread("square.bmp", IMREAD_GRAYSCALE);

	if (src1.empty() || src2.empty()) {
		cerr << "Image load failed!" << endl;
		return -1;
	}

	imshow("src1", src1);
	imshow("src2", src2);

	Mat dst1, dst2, dst3, dst4;
```

```
//덧셈 연산
add(src1, src2, dst1); //사각형 바깥 영역=레나 영상의 픽셀 값 + 0 : 변화 X, 사각형 내부+255:무조건 포화가 발생 -> 픽셀 값이 모두 255로 설정
//평균 연산
addWeighted(src1, 0.5, src2, 0.5, 0, dst2); // 사각형 바깥 영역= 레나 영상과 밝기 값 0의 평균:어둠, 사각형 안쪽= 레나 영상과 밝기 값 255의 평균:밝아짐
//뺄셈 연산
subtract(src1, src2, dst3); // 사각형 바깥 영역=레나 영상의 픽셀 값- 0: 변화 X, 사각형 내부=레나 영상- 255:포화 연산에 의해 무조건 0으로 설정
//차 연산
absdiff(src1, src2, dst4); // 사각형 안쪽 영역에서만 반전이 되는 효과

imshow("dst1", dst1);
imshow("dst2", dst2);
imshow("dst3", dst3);
imshow("dst4", dst4);
waitKey();

return 0;
}
```


src1
src2
dst1
dst2
dst3
dst4
Copyright © Gilbut, Inc. All rights reserved.

슈티어링 4주차 실습 2

영상의 논리 연산 수행



```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(void)
{
	Mat src1 = imread("lenna256.bmp", IMREAD_GRAYSCALE);
	Mat src2 = imread("square.bmp", IMREAD_GRAYSCALE);

	if (src1.empty() || src2.empty()) {
		cerr << "Image load failed!" << endl;
		return -1;
	}

	imshow("src1", src1);
	imshow("src2", src2);

	Mat dst1, dst2, dst3, dst4;
```

```
//비트 단위 논리곱
bitwise_and(src1, src2, dst1); //dst1 = src1 & src2;
//비트 단위 논리합
bitwise_or(src1, src2, dst2); // dst2 = src1 | src2;
//비트 단위 배타적 논리합
bitwise_xor(src1, src2, dst3); // dst3 = src1 ^ src2;
//비트 단위 부정 연산
bitwise_not(src1, dst4); // dst4 = ~src1;
/*연산자 재정의 사용 가능*/

imshow("dst1", dst1);
imshow("dst2", dst2);
imshow("dst3", dst3);
imshow("dst4", dst4);
waitKey();

return 0;
}
```

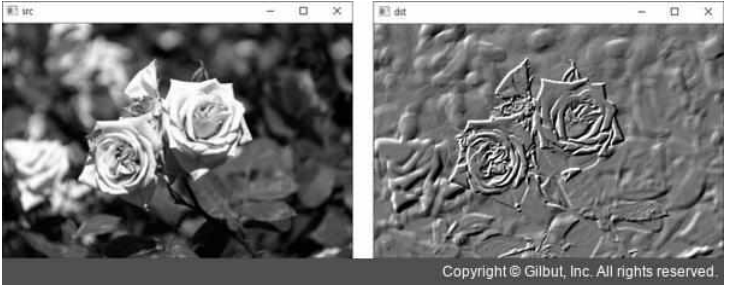

src1
src2
dst1
dst2
dst3
dst4
Copyright © Gilbut, Inc. All rights reserved.

슈티어링 4주차 실습 3

실제 영상에 대해 엠보싱 필터링 수행

```
#include "opencv2/opencv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

void filter_embossing();

int main(void)
{
	filter_embossing();

	return 0;
}

bitwise_and(src1, src2, dst1);
```

```
void filter_embossing()
{
	Mat src = imread("rose.bmp", IMREAD_GRAYSCALE);

	if (src.empty()) {
		cerr << "Image load failed!" << endl;
		return;
	}

	float data[] = { -1, -1, 0, -1, 0, 1, 0, 1, 1 };
	Mat emboss(3, 3, CV_32FC1, data);

	Mat dst;
	filter2D(src, dst, -1, emboss, Point(-1, -1), 128); //더욱 입체감 있는 엠보싱 결과 영상을 얻기 위해 filter2D() 함수의 여섯 번째 인자에 128을 지정, 필터 마스크 중앙을 고정점으로 사용(기본값)

	imshow("src", src);
	imshow("dst", dst);

	waitKey();
	destroyAllWindows();
}
```


src
dst
Copyright © Gilbut, Inc. All rights reserved.

7.2 블러링 : 영상 부드럽게 하기

- 정의
  - 영상을 부드럽게 만드는 필터링 기법 (스무딩)

- 사용
  - 거친 입력 영상 -> 부드럽게 만드는 용도
  - 입력 영상 내 잡음 영향 제거 위한 전처리

7.2.1 평균값 필터

- 정의
  - 입력 영상에서 특정 픽셀과 주변 픽셀들의 산술 평균을 결과 영상 픽셀 값으로 설정하는 필터

- 효과
  - 날카로운 에지가 무뎌짐
  - 잡음의 영향이 크게 사라짐

- 한계
  - 과도하게 사용 시 사물의 경계 흐릿
  - 사물의 구분 어려워짐

평균값 필터 마스크

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$$\frac{1}{25}\times\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

$$\text{kernel}=\frac{1}{\text{ksize.width}\times\text{ksize.height}}\begin{bmatrix}1&1&\cdots&1\\1&1&\cdots&1\\\vdots&\vdots&\ddots&\vdots\\1&1&\cdots&1\end{bmatrix}$$

-> 각 행렬 원소 값 / 행렬의 전체 원소 개수
마스크의 크기가 커질수록 부드러운 느낌의 결과 영상 생성
& 연산량 크게 증가

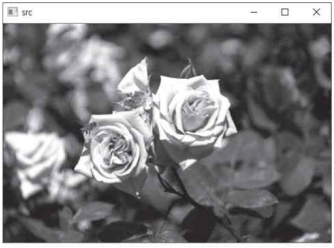
blur() 함수를 이용한 평균값 필터링

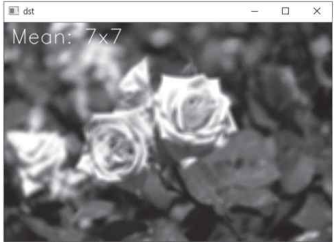
void blur(InputArray src, OutputArray dst, Size ksize,
Point anchor = Point(-1,-1), int borderType = BORDER_DEFAULT);

- src : 입력 영상
- dst : 출력 영상. src와 같은 크기, 같은 채널 수
- ksize : 블러링 커널 크기
- anchor : 고정점 좌표. Point(-1, -1)을 지정하면 커널 중심을 고정점으로 사용
- borderType : 가장자리 픽셀 확장 방식

```
// 평균값 필터
void blurring_mean()
{
    // 입력 영상 src : rose.bmp 파일
    Mat src = imread("rose.bmp", IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    imshow("src", src);
```

```
    Mat dst;

    // ksize 값이 3, 5, 7이 되도록 for 반복문을 설정
    for (int ksize = 3; ksize <= 7; ksize += 2) {
        // ksize×ksize 크기의 평균값 필터 마스크를 이용하여 블러링을 수행
        blur(src, dst, Size(ksize, ksize));

        // 사용된 평균값 필터의 크기를 문자열 형태로 결과 영상 dst 위에 출력
        String desc = format("Mean: %dx%d", ksize, ksize);
        putText(dst, desc, Point(10, 30), FONT_HERSHEY_SIMPLEX, 1.0,
            Scalar(255), 1, LINE_AA);

        imshow("dst", dst);
        waitKey();
    }

    destroyAllWindows();
}
```





평균값 필터의 크기가 커질수록 결과 영상이 더욱 부드럽게 변경되는 것을 확인 가능

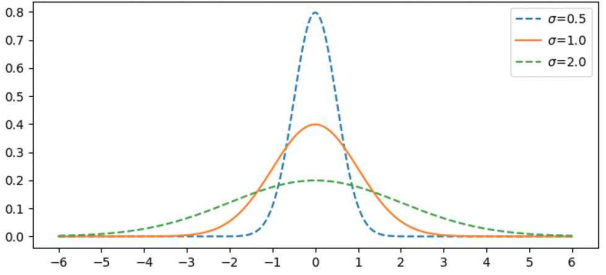
7.2.2 가우시안 필터

- 정의
  - 가우시안 분포 함수를 근사하여 생성한 필터 마스크를 사용 하는 필터링 기법

- 가우시안 분포
  - 평균이 0인 가우시안 분포 함수를 주로 사용
  - 평균 : 0, 표준 편차 : 1

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

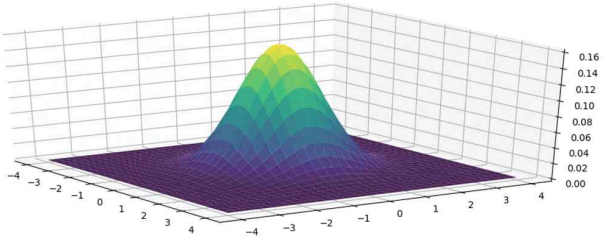
1차원 가우시안 분포



- 평균이 0이고 표준 편차 σ가 각각 0.5, 1.0, 2.0인 가우시안 분포 그래프
- 평균이 0이므로 x=0에서 최댓값을 가짐
- 가우시안 분포 함수 값은 특정 x가 발생할 수 있는 확률의 개념을 가짐

-> 그래프 아래 면적을 모두 더하면 1이 됨

2차원 가우시안 분포



$$G_{\sigma_x \sigma_y}(x, y) = \frac{1}{2\pi\sigma_x \sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)}$$

- 평균이 (0, 0)이고 x축과 y축 방향의 표준 편차가 각각 σx = σy = 1인 2차원 가우시안 분포 함수
- 함수 그래프 아래의 부피를 구하면 1이 됨

9 x 9 가우시안 필터

- 평균이 0이고 표준 편차가 σ인 가우시안 분포는 x가 -4σ부터 4σ 사이인 구간에서 그 값의 대부분이 존재하기 때문에 가우시안 필터 마스크의 크기는 보통 (8σ+1)로 결정함
- > σx=σy=1.0인 가우시안 함수를 사용할 경우, x={-4, -3, -2, -1, 0, 1, 2, 3, 4}, y={-4, -3, -2, -1, 0, 1, 2, 3, 4}
- > 9 x 9 크기의 가우시안 필터 사용됨

$$\mathbf{G} = \begin{pmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0001 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0011 & 0.0018 & 0.0011 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0002 & 0.0029 & 0.0131 & 0.0215 & 0.0131 & 0.0029 & 0.0002 & 0.0000 \\ 0.0000 & 0.0011 & 0.0131 & 0.0586 & 0.0965 & 0.0586 & 0.0131 & 0.0011 & 0.0000 \\ 0.0001 & 0.0018 & 0.0215 & 0.0965 & 0.1592 & 0.0965 & 0.0215 & 0.0018 & 0.0001 \\ 0.0000 & 0.0011 & 0.0131 & 0.0586 & 0.0965 & 0.0586 & 0.0131 & 0.0011 & 0.0000 \\ 0.0000 & 0.0002 & 0.0029 & 0.0131 & 0.0215 & 0.0131 & 0.0029 & 0.0002 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0011 & 0.0018 & 0.0011 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0001 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \end{pmatrix}$$

필터링 대상 픽셀 근처에는 가중치를 크게 주고,
필터링 대상 픽셀과 멀리 떨어져 있는 주변부에는 가중치를 조금만 주어서
가중 평균(weighted average)을 구함

-> 가우시안 필터 마스크가 가중 평균을 구하기 위한 가중치 행렬 역할을 함
But. 81번의 많은 연산량 필요

$$G_{\sigma_x\sigma_y}(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\left(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2}\right)}$$

$$= \frac{1}{\sqrt{2\pi}\sigma_x} e^{-\frac{x^2}{2\sigma_x^2}} \times \frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{y^2}{2\sigma_y^2}} = G_{\sigma_x}(x)\cdot G_{\sigma_y}(y)$$

2차원 가우시안 분포 함수는
1차원 가우시안 분포 함수의 곱으로 분리할 수 있음

$$\mathbf{g} = \begin{pmatrix} 0.0001 & 0.0044 & 0.0540 & 0.2420 & 0.3989 & 0.2420 & 0.0540 & 0.0044 & 0.0001 \end{pmatrix}$$

σ=1.0인 1차원 가우시안 함수 -> 1×9 가우시안 마스크 행렬

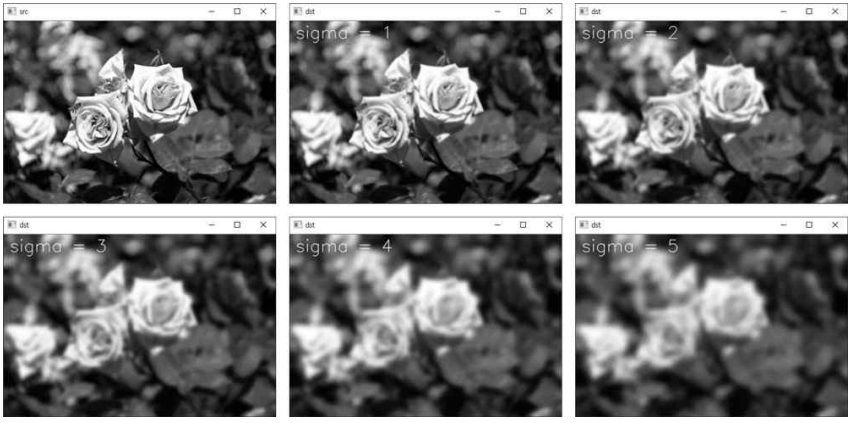
행렬 g를 이용하여 필터링을 한 번 수행
그 결과를 다시 g의 전치 행렬인 gT를 이용하여 필터링하는 것
= 2차원 가우시안 필터 마스크로 한 번 필터링하는 것

```
void GaussianBlur(InputArray src, OutputArray dst, Size ksize,
                  double sigmaX, double sigmaY = 0,
                  int borderType = BORDER_DEFAULT);
```

- src : 입력 영상. 다채널 영상은 각 채널별로 블러링을 수행합니다.
- dst : 출력 영상. src와 같은 크기, 같은 타입을 갖습니다.
- ksize : 가우시안 커널 크기. ksize.width와 ksize.height는 0보다 큰 홀수여야 함. ksize에 Size()를 지정하면 표준 편차로부터 커널 크기를 자동으로 결정.
- sigmaX: x 방향으로의 가우시안 커널 표준 편차
- sigmaY : y 방향으로의 가우시안 커널 표준 편차. 만약 sigmaY = 0이면 sigmaX와 같은 값을 사용
- borderType : 가장자리 픽셀 확장 방식

```
void blurring_gaussian()
{
    Mat src = imread("../../rose.bmp", IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    imshow("src", src);

    Mat dst;
```

```
    // sigma 값을 1부터 5까지 증가시키면서 가우시안 블러링을 수행하고 그 결과를 화면에 나타냄
    for (int sigma = 1; sigma <= 5; sigma++) {

        // src 영상에 가우시안 표준 편차가 sigma인 가우시안 블러링을 수행하고 그 결과를 dst에 저장
        GaussianBlur(src, dst, Size(0, 0), (double)sigma);

        // 사용한 가우시안 표준 편차(sigma) 값을 결과 영상 dst 위에 출력
        String desc = format("Gaussian: sigma = %d", sigma);
        putText(dst, desc, Point(10, 30), FONT_HERSHEY_SIMPLEX, 1.0,
            Scalar(255), 1, LINE_AA);

        imshow("dst", dst);
        waitKey();
    }

    destroyAllWindows();
}
```



표준 편차 값이 커질수록
결과 영상이 더욱 부드럽게 변경되는 것을 확인할 수 있음

7.3 샤프닝 : 영상 날카롭게 하기

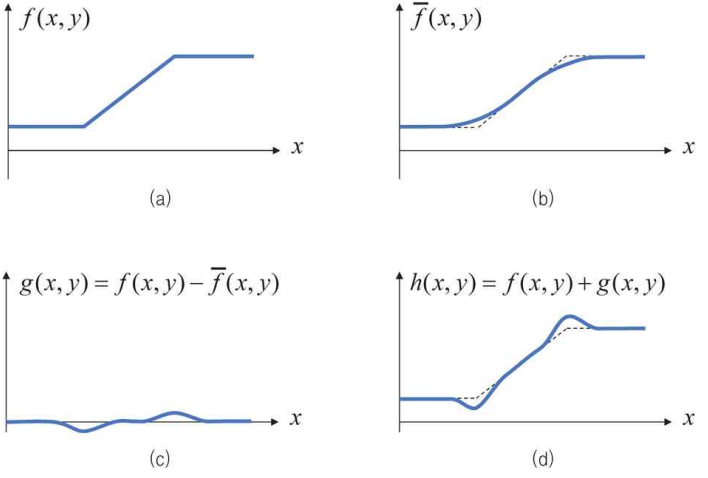
- 정의
  - 사물의 윤곽이 뚜렷하고 선명한 느낌이 나도록 영상을 변경하는 필터링 기법

- 방법
  - 영상 에지 근방에서 픽셀 값의 명암비가 커지도록 수정

7.3.1 언샤프 마스크 필터

- 언샤프
  - 날카롭지 않은 영상

- 언샤프 마스크 필터
  - 언샤프한 영상을 이용하여 역으로 날카로운 영상을 생성하는 필터



원본 영상에서 블러링 된 영상을 뺀 함수를 원본 영상에 더하면
날카로운 엣지 성분이 강조된 최종 영상이 결과로 나옴

$$h(x,y) = f(x,y) + \alpha \cdot g(x,y)$$

$$h(x,y) = f(x,y) + \alpha(f(x,y) - \bar{f}(x,y))$$
$$= (1+\alpha) f(x,y) - \alpha \cdot \bar{f}(x,y)$$

α는 샤프닝 결과 영상의 날카로운 정도를 조절할 수 있는 파라미터
α에 1.0을 지정하면 날카로운 성분을 그대로 한 번 더하는 것
α에 1보다 작은 값을 지정하면 조금 덜 날카로운 영상을 만들 수 있음

```
void unsharp_mask()
{
    Mat src = imread("../../rose.bmp", IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    imshow("src", src);

    // 가우시안 필터의 표준 편차 sigma 값을 1부터 5까지 증가시키면서 언샤프 마스크 필터링 수행
    for (int sigma = 1; sigma <= 5; sigma++) {

        // 가우시안 필터를 이용한 블러링 영상을 blurred에 저장
        Mat blurred;
        GaussianBlur(src, blurred, Size(), sigma);
```

```
        // 언샤프 마스크 필터링을 수행
        float alpha = 1.f;
        Mat dst = (1 + alpha) * src - alpha * blurred;

        // 샤프닝 결과 영상 dst에 사용된 sigma 값 출력
        String desc = format("sigma: %d", sigma);
        putText(dst, desc, Point(10, 30), FONT_HERSHEY_SIMPLEX, 1.0,
            Scalar(255), 1, LINE_AA);

        imshow("dst", dst);
        waitKey();
    }

    destroyAllWindows();
}
```

Alpha 값이 1.0f 로 고정되어 있음
-> 변경하며 무엇이 달라지는지 확인해보기

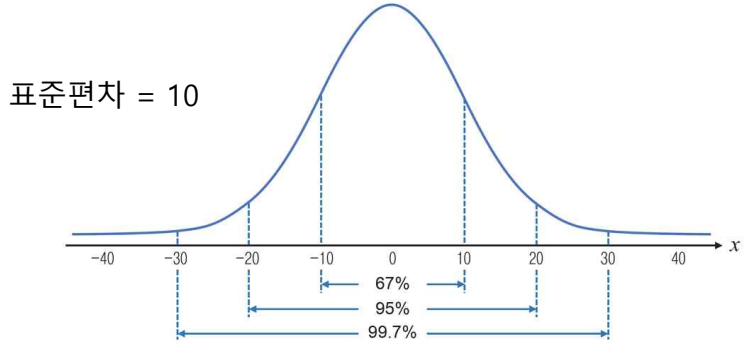
Sigma : gaussian blurring의 표준 편차 -> 클수록 부드러운 영상
-> sigma 값이 커질수록 경계가 뚜렷해지는 것을 알 수 있음

7.4.1 영상과 잡음 모델

- **잡음**
  - 원본 신호에 추가된 원치 않은 신호
  - 사진의 경우 광학적 신호 -> 전기적 신호로 변환하는 센서에서 주로 잡음이 추가 됨
  - 영상 신호 = 원본 신호 + 추가되는 잡음

$$f(x,y) = s(x,y) + n(x,y)$$

- **잡음 모델**
  - 잡음이 생성되는 방식



void randn(InputOutputArray dst, InputArray mean, InputArray stddev);

- Dst : 가우시안 난수로 채워질 행렬
- Mean : 가우시안 분포 평균
- Stddev : 가우시안 분포 표준 편차

```
void noise_gaussian()
{
    // lenna.bmp 파일을 그레이스케일 형식으로 불러와 src에 저장
    Mat src = imread("../../../lenna.bmp", IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    imshow("src", src);

    // 표준 편차 stddev 값이 10, 20, 30이 되도록 for 반복문을 수행
    for (int stddev = 10; stddev <= 30; stddev += 10) {

        // 평균이 0이고 표준 편차가 stddev인 가우시안 잡음을 생성하여 noise 행렬에 저장
        // noise 행렬은 부호 있는 정수형(CV_32SC1)을 사용하도록 미리 생성하여 randn() 함수에 전달
        Mat noise(src.size(), CV_32SC1);
        randn(noise, 0, stddev);
```

```
        // 입력 영상 src에 가우시안 잡음 noise를 더하여 결과 영상 dst를 생성
        // dst 영상의 깊이는 CV_8U로 설정
        Mat dst;
        add(src, noise, dst, Mat(), CV_8U);

        String desc = format("stddev = %d", stddev);
        putText(dst, desc, Point(10, 30), FONT_HERSHEY_SIMPLEX, 1.0, Scalar(255), 1, LINE_AA);
        imshow("dst", dst);
        waitKey();
    }

    destroyAllWindows();
}
```

표준 편차 stddev 값이
증가함에 따라
잡음의 영향이 커지므로
결과 영상이 더욱 지저분해짐

7.4.2 양방향 필터

- 가우시안 잡음 제거를 위해 가우시안 필터 사용 시
  - 픽셀 값이 크게 변하지 않는 평탄한 영역
    - 주변 픽셀 값이 부드럽게 블러링되면서 잡음의 영향도 크게 줄어듦
  - 픽셀 값이 급격하게 변경되는 에지 근방
    - 에지 성분까지 감소
    - 객체 윤곽 흐릿

- 에지 보전 잡음 제거 필터
  - 양방향 필터 : 에지 성분 유지, 가우시안 잡음 효과적으로 제거

$$g_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}\left(\|\mathbf{p}-\mathbf{q}\|\right) G_{\sigma_r}\left(|f_{\mathbf{p}} - f_{\mathbf{q}}|\right) f_{\mathbf{q}}$$

- 두 개의 가우시안 함수 곱으로 구성된 필터
- F : 입력 영상, g : 출력 영상, 그리고 p, q : 픽셀의 좌표
- Fp, fq : p점과 q점에서의 입력 영상 픽셀 값
- gp : p점에서의 출력 영상 픽셀 값
- Gσs, Gσr : 표준 편차가 σs와 σr인 가우시안 분포 함수
- S : 필터 크기를 나타내고
- Wp : 양방향 필터 마스크 합이 1이 되도록 만드는 정규화 상수

$$g_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}\left(\|\mathbf{p}-\mathbf{q}\|\right) G_{\sigma_r}\left(|f_{\mathbf{p}} - f_{\mathbf{q}}|\right) f_{\mathbf{q}}$$

- Gσs(||p-q||) 함수
  - 두 점 사이의 거리에 대한 가우시안 함수
  - 앞의 가우시안 함수와 동일
- Gσr(|fp-fq|) 함수
  - 두 점의 픽셀 값 차이에 의한 가우시안 함수
  - 두 점의 픽셀 밝기 값의 차이가 적은 평탄한 영역에서는 큰 가중치를 가짐
  - 두 점의 픽셀 밝기 값의 차이가 큰 에지 영역에서는 0에 가까운 값 -> 블러링 효과 거의 나타나지 않음

```
void filter_bilateral()
{
    Mat src = imread("../../../lenna.bmp", IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    // 그레이스케일 레나 영상 src에 평균이 0이고 표준 편차가 5인 가우시안 잡음 추가
    Mat noise(src.size(), CV_32SC1);
    randn(noise, 0, 5);
    add(src, noise, src, Mat(), CV_8U);

    // 표준 편차가 5인 가우시안 필터링을 수행하여 dst1에 저장
    Mat dst1;
    GaussianBlur(src, dst1, Size(), 5);
```

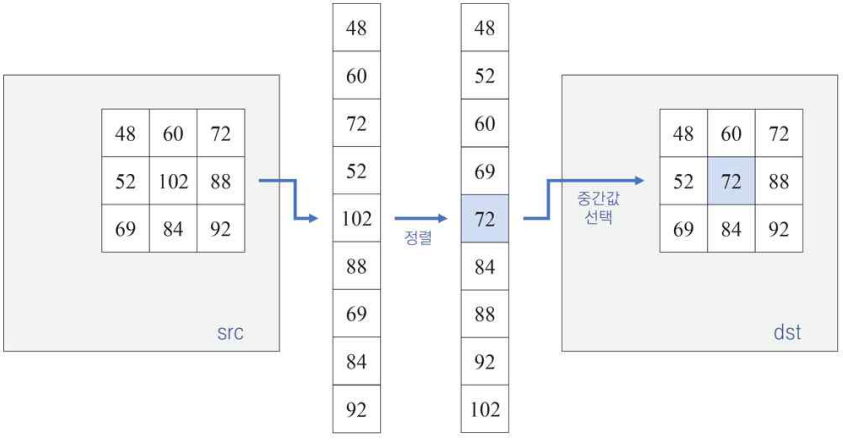
```
    // 색 공간의 표준 편차는 10, 좌표 공간의 표준 편차는 5를 사용하는 양방향 필터링을 수행하여 dst2에 저장
    Mat dst2;
    bilateralFilter(src, dst2, -1, 10, 5);

    // src, dst1, dst2 영상을 모두 화면에 출력
    imshow("src", src);
    imshow("dst1", dst1);
    imshow("dst2", dst2);

    waitKey();
    destroyAllWindows();
}
```

- src 창의 영상
  - lenna 영상에 평균이 0이고 표준 편차가 5인 가우시안 잡음이 추가
- dst1 영상
  - Src 에 표준 편차가 5인 가우시안 필터링을 수행한 결과
  - 머리카락, 모자, 배경 사물의 경계 부분이 함께 블러링되어 흐릿하게 변경
- dst2 영상
  - 머리카락, 모자, 배경 사물의 경계는 그대로 유지
  - 평탄한 영역의 잡음은 크게 줄어듦

7.4.3 미디언 필터

- 미디언 필터(median filter)
  - 입력 영상에서 자기 자신 픽셀과 주변 픽셀 값 중에서 중간값(median)을 선택하여 결과 영상 픽셀 값으로 설정하는 필터링 기법
  - 내부에서 픽셀 값 정렬 과정이 사용됨
  - 잡음 픽셀 값이 주변 픽셀 값과 큰 차이가 있는 경우 효과적

- 소금&후추 잡음(salt & pepper noise)
  - 픽셀 값이 일정 확률로 0 또는 255로 변경되는 형태의 잡음


48 60 72
52 102 88
69 84 92
src
48
60
72
52
102
88
69
84
92
정렬
48
52
60
69
72
84
88
92
102
중간값
선택
48 60 72
52 72 88
69 84 92
dst

```
void filter_median()
{
    Mat src = imread("../../../lenna.bmp", IMREAD_GRAYSCALE);

    if (src.empty()) {
        cerr << "Image load failed!" << endl;
        return;
    }

    // src 영상에서 10%에 해당하는 픽셀 값을 0 또는 255로 설정
    int num = (int)(src.total() * 0.1);
    for (int i = 0; i < num; i++) {
        int x = rand() % src.cols;
        int y = rand() % src.rows;
        src.at<uchar>(y, x) = (i % 2) * 255;
    }
```

```
    // 표준 편차가 1인 가우시안 필터링을 수행하여 dst1에 저장
    Mat dst1;
    GaussianBlur(src, dst1, Size(), 1);

    // 크기가 3인 미디언 필터를 실행하여 dst2에 저장
    Mat dst2;
    medianBlur(src, dst2, 3);

    // src, dst1, dst2 영상을 모두 화면에 출력
    imshow("src", src);
    imshow("dst1", dst1);
    imshow("dst2", dst2);

    waitKey();
    destroyAllWindows();
}
```

- src 영상
  - Lenna.bmp 에10%의 확률로 소금&후추 잡음이 추가된 영상
- dst1 영상
  - 가우시안 필터를 적용한 결과
  - 가우시안 블러링을 적용하여도 여전히 영상이 지저분하게 보임
- dst2 영상
  - 미디언 필터를 적용
  - 잡음에 의해 추가된 흰색 또는 검은색 픽셀이 효과적으로 제거됨