OPEN CV SEMINAR

Chap 6

2019/07/24

박소민

OPEN CV

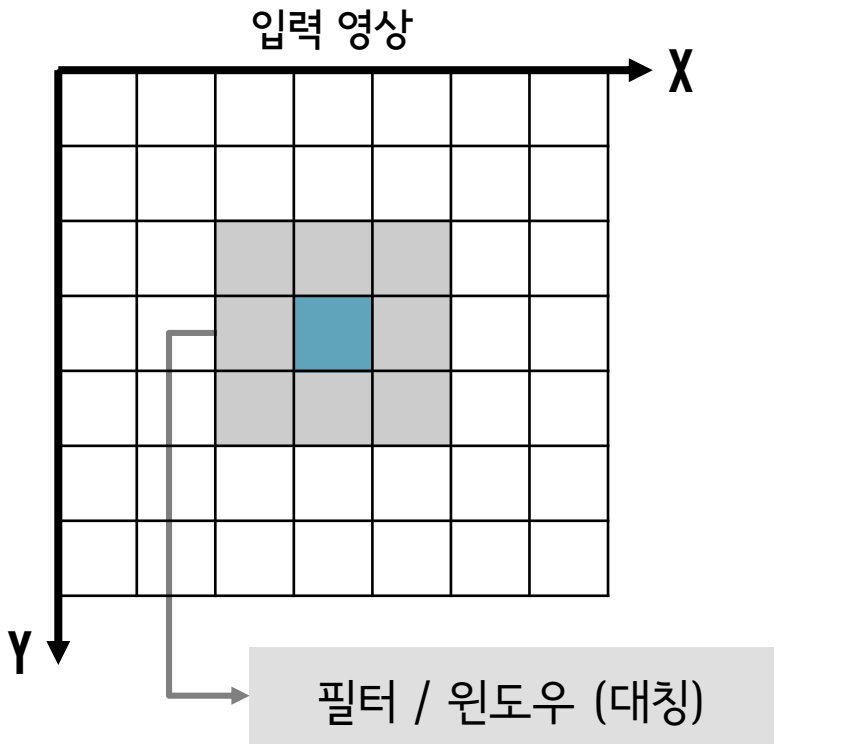
CHAP 6 이웃을 고려한 공간 영역 필터링

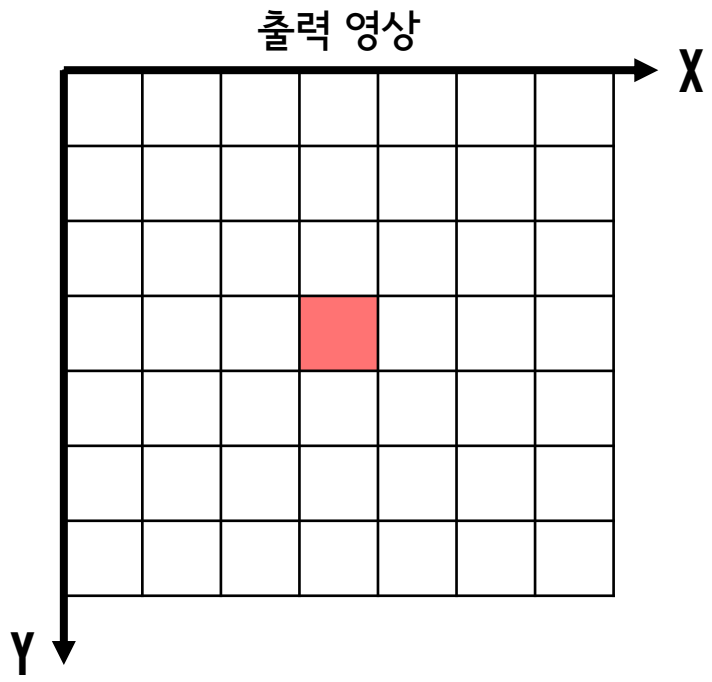
포인트 프로세싱

주위의 이웃 화소 고려하지 않음

공간 영역 필터링

입력 화소 주변 이웃 고려

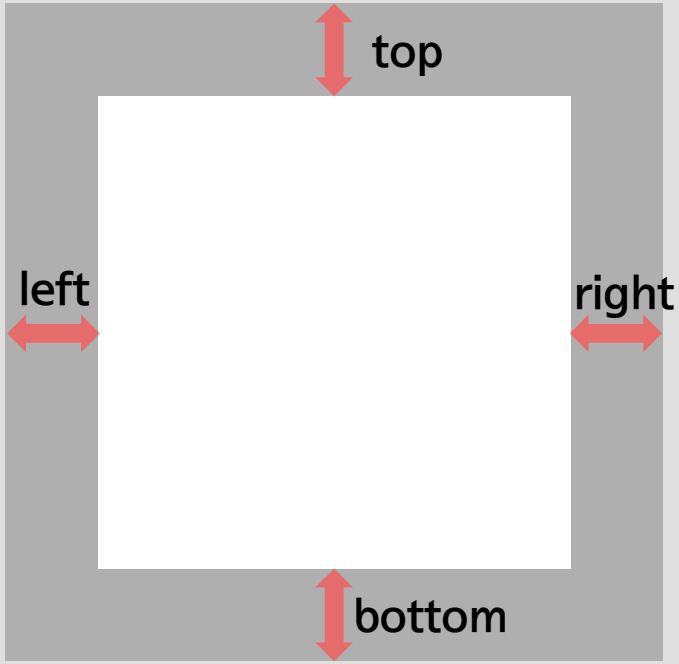




CHAP 6 01 경계값 채우기, boxFilter

경계값 채우기

```
void copyMakeBorder(InputArray src, OutputArray dst, int top, int bottom, int left,
                    int right, int borderType, const Scalar& value = Scalar())
```



borderType

- BORDER_CONSTANT, value — 00 123 00
- BORDER_REPLICATE — 11 123 33
- BORDER_REFLECT — 21 123 32
- BORDER_REFLECT101 — 32 123 21
- BORDER_WRAP — 23 123 12

CHAP 6 01 경계값 채우기, boxFilter

경계값 채우기

int borderInterpolate(int p, int len, int borderType)

borderType에 따라, p 위치에 대한 원본 영상의 좌표 계산

len은 행렬의 행 또는 열의 길이

주어진 행렬 밖(virtual)의 값을 알고 싶을 때

img 행렬의 Point(-5,100) 값

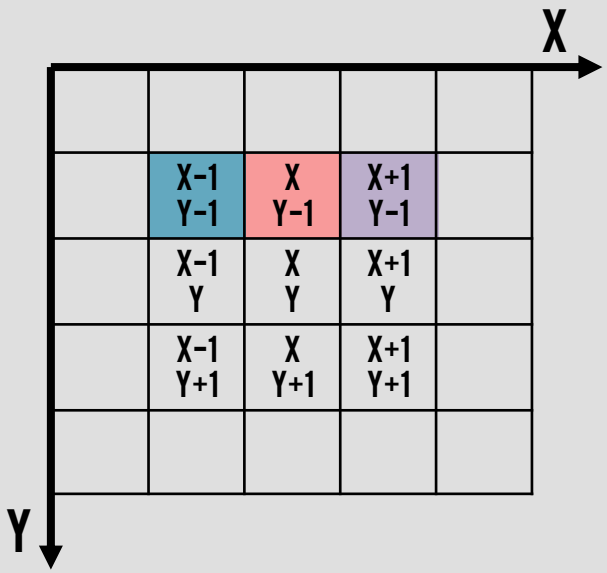
```
img.at<float>( borderInterpolate(100, img.rows, BORDER_REPLICATE),
               borderInterpolate(-5, img.cols, BORDER_REPLICATE));
```

CHAP 6 02 2D 필터 연산

2D 필터 연산

상관관계 (correlation)

$$\sum_{s=0}^{kernel.cols-1} \sum_{t=0}^{kernel.rows-1} w(s,t)\, src(x+s-anchor.x, y+t-anchor.y)$$



W(0,0)	W(1,0)	W(2,0)
W(0,1)	W(1,1)	W(2,1)
W(0,2)	W(1,2)	W(2,2)

필터 커널

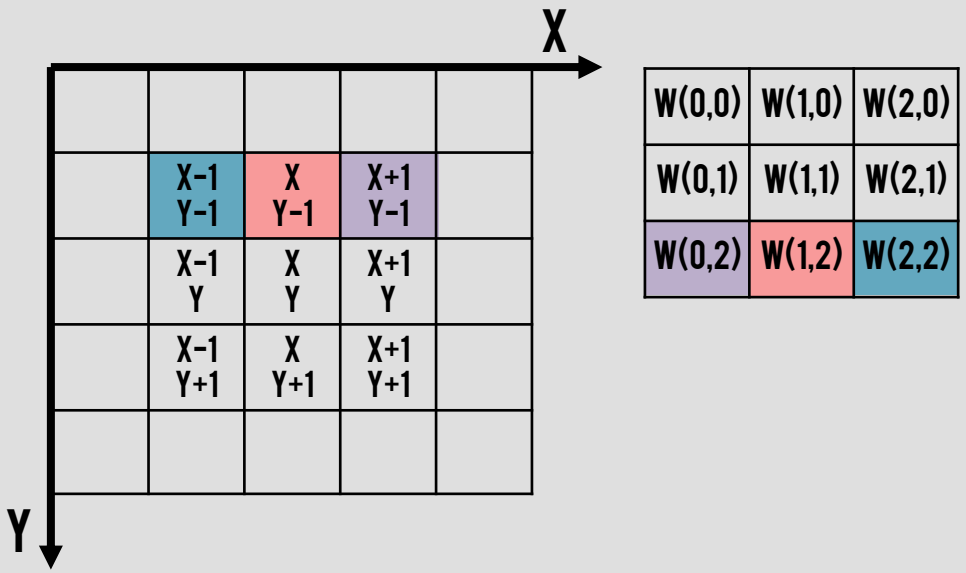
- 이웃에 속하는 대응되는 위치의 값을 곱셈하여 합계를 출력영상에 저장
- 통계에서의 상관관계와 다름

CHAP 6 02 2D 필터 연산

2D 필터 연산

회선 (convolution) → DFT로 계산

$$\sum_{s=0}^{kernel.cols-1}\sum_{t=0}^{kernel.rows-1} w(s,t)\, src(x+s+(kernel.cols-1)-anchor.x,\; y+t+(kernel.rows-1)-anchor.y)$$



- 커널을 상하좌우로 뒤집어 회전시켜 계산

$$w = w_1 * w_2$$

위 식을 만족하면

- 분리 가능한 필터(separable filter)
- 각각 행, 열 필터면 분리 가능한 선형 필터 (separable linear filter)

CHAP 6 02 2D 필터 연산

2D 필터링 함수

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

- filter2D 함수는 **상관관계**로 필터링
  대부분의 윈도우 커널이 **대칭**이므로
  "회선한다"고 말함
- **kernel**에 따라 스무딩 필터링, 선명한 샤프닝 필터링을 할 수 있음

- **ddepth** = -1이면 src.depth()=dst.depth()
  src.depth()보다 큰 bit depth인 dst.depth() 사용 가능
- **kernel** : 1-채널 float 행렬
- **anchor** : kernel의 중심점으로 kernel 내의 위치
  Point(-1,-1)은 kernel의 중심점 계산
- **delta** : 필터링 결과에 더해지는 값
- BORDER_DEFAULT = BORDER_REFLECT101

CHAP 6 02 2D 필터 연산

2D 필터링 함수

void sepFilter2D(InputArray src, OutputArray dst, int ddepth, InputArray kernelX, InputArray kernel, Point anchor = Point(-1,-1), double delta = 0, int borderType = BORDER_DEFAULT)

- 분리가능한 선형 필터

  각 행에 kernelX를 적용시킨 후,

  각 열에 kernelY를 적용하여 계산

- 연산 속도를 높여줌

CHAP 6 02 2D 필터 연산

2D 필터링 함수

https://bskyvision.com/42

예제 6-5 : 영상에서 filter2D, sepFilter2D를 이용한 평균 필터링

0.1111	0.1111	0.1111
0.1111	0.1111	0.1111
0.1111	0.1111	0.1111

그림 1. normalized box filter를 위한 커널

100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200
100	100	100	100	100	200	200	200	200	200

그림 2. 단순한 엣지가 있는 이미지와 대응하는 픽셀 값

100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000
100.0000	100.0000	100.0000	133.3333	166.6667	200.0000	200.0000	200.0000

그림 5. 필터 처리된 이미지와 대응하는 픽셀값들.

CHAP 6 03 영상을 부드럽게 하는 스무딩 연산

박스 필터와 양방향 필터

```
void boxFilter(InputArray src, OutputArray dst, int ddepth, Size ksize,
               Point anchor = Point(-1,-1), bool normalize = true, int borderType)
```

- 박스 커널의 커널 K는 영상을 부드럽게 함
- 예제 6-5와 동일

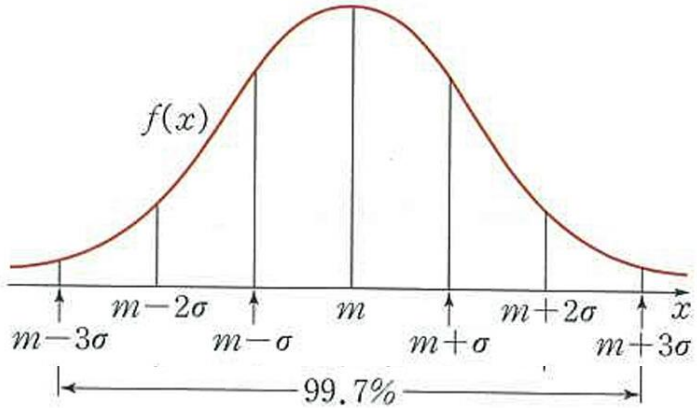
- **dst :** src와 같은 크기 같은 자료형의 ddepth 깊이의 필터링된 출력 영상
- **borderType :** BORDER_REFLECT101

- **normalize :** true면 커널 크기로 정규화 평균 필터와 같음

$$K = \alpha \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

$$\alpha = \begin{cases} \dfrac{1}{ksize.width * ksize.height} & \text{if true} \\ 1 & \text{o.w} \end{cases}$$

CHAP 6 03 영상을 부드럽게 하는 스무딩 연산



박스 필터와 양방향 필터

```
void bilateralFilter(InputArray src, OutputArray dst, int d, double sigmaColor,
                     double sigmaSpace, int borderType = BORDER_DEFAULT)
```

- 가우시안 함수를 사용하여 에지를 덜 약화시킴

- d : 필터링 동안 사용될 각 화소의 이웃을 결정한 지름 (필터의 크기)

  d = 5 : 실시간 처리에 적합

  d < 0 : sigmaSpace에 의해 계산

  d = 2 * 3 * sigmaSpace + 1

- **sigmaColor : 컬러 공간에서 필터 표준편차**

  값이 클수록, 이웃화소 내(signalSpace)의 화소 중에서 색상 공간에서 **더 멀리 떨어져 있는 색상들을 혼합하여** 더 큰 영역으로 만듦

- **sigmaSpace : 좌표 공간에서 필터 표준편차**

  값이 클수록, sigmaColor에 의한 색상이 충분히 가까우면서, **위치가 더 멀리 떨어진 이웃화소에 영향**

https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=bilateralfilter#bilateralfilter

CHAP 6 03 영상을 부드럽게 하는 스무딩 연산

블러 필터 (blur filter)

void medianBlur(InputArray src, OutputArray dst, int ksize)

- 점잡음(salt and pepper noise)에 효과적

- **ksize :** ksize * ksize의 필터 윈도우 사용
  중위수 계산하여 dst에 저장
- **dst :** src와 같은 크기, 같은 자료형
- **borderType :** BORDER_REPLICATE

- **src :** 1, 3, 4 채널

  ksize = 3 or 5
  src 깊이 - CV_8U, CV_16U, CV_32F

  ksize 크면
  src 깊이 - CV_8U

CHAP 6 03 영상을 부드럽게 하는 스무딩 연산

블러 필터 (blur filter)

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

- 정규화된 박스 필터

  boxFilter(src, dst, -1, ksize, true, BORDER_REFLECT)와 동일

- **src** : 모든 채널

  CV_8U, CV_16U, CV_32F, CV_64F
- **dst** : src와 같은 크기, 같은 자료형
- **borderType** : BORDER_REFLECT101

CHAP 6 03 영상을 부드럽게 하는 스무딩 연산

블러 필터 (blur filter)

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

- ksize 크기의 2차원 가우시안 커널과 회선을 수행

Mat getGaussianKernel(int ksize, double sigma, int ktype = CV_64F)으로 (CV_32F, CV_64F)

ksize * 1 의 1D 가우시안 커널을 행렬로 반환 (ksize = 1, 3, 5, 7 는 미리 계산되어 있음)

sigma : 가우시안 표준편차
sigma <= 0 : sigma = 0.3 * ((ksize - 1) * 0.5 - 1) + 0.8

- **sigmaX, sigmaY** : X축과 Y축 방향으로의 가우시안 커널 표준편차

sigmaX ≠ 0 : sigmaY = sigmaX　　**sigmaX, sigmaY = 0** : ksize 가지고 계산

- **BORDER_DEFAULT** : BORDER_REFLECT101

CHAP 6 04 영상을 날카롭게 하는 샤프닝 연산

1차 미분 필터링

```
void getDeivKernels(OutputArray kx, OutputArray ky, int dx, int dy, int ksize,
                    bool normalize = false, int ktype = CV_32F)
```

CV_32F, CV_64F

- 영상에서 미분을 계산하기 위한 1D 선형 필터 반환
- sepFilter2D나 ky * kx.t()에 전달하여 사용

- **ksize** : 커널의 크기
  CV_SHARR(= -1)이면 Scharr 3 * 3 커널
  1, 3, 4, 7이면 Sobal 커널
- **borderType** : BORDER_REFLECT101

CHAP 6 04 영상을 날카롭게 하는 샤프닝 연산

1차 미분 필터링

```
void Sobel(InputArray src, OutputArray ky, int ddepth, int dx, int dy, int ksize=3,
           double scale=1, double delta=0, int borderType=BORDER_DEFAULT)
```

8비트 영상을 8비트의 출력 영상에 저장하면 결과가 잘리는(truncation) 주의

- dx, dy : 각각 x축, y축 미분 차수
- ksize = 1 : 3 * 1 or 1 * 3 커널 적용
  ≠ 1 : ksize * ksize
  = -1 : scharr 3 * 3

```
void Scharr(InputArray src, OutputArray ky, int ddepth, int dx, int dy,
            double scale=1, double delta=0, int borderType=BORDER_DEFAULT)
```

Soble 함수에서 ksize = -1일 때와 동일한 결과

CHAP 6 04 영상을 날카롭게 하는 샤프닝 연산

2차 미분 필터링

2차원 함수 $f(x, y)$ 의 **라플라시안(Laplacian)** $\nabla^2 f(x, y)$은 2차 편미분의 합으로 정의

edge 화소 = 라플라시안 필터링에서 부호가 바뀌는 곳 = 0-교차(zero-crossing)

→ void Laplacian(InputArray src, OutputArray dst, int ddepth, int ksize=1,
double scale = 1, double delta=0, int borderType BORDER_DEFAULT)

잡음에 민감 => ① 가우시안 필터링 후 라플라시안
② 가우시안 함수에 대한 라플라시안 계산하여 윈도우 필터 커널 생성

→ LoG(Laplacian of Gaussian)

CHAP 6 05 모폴로지 연산

구조 요소를 이용하여 반복적으로 **영역을 확장**시켜 떨어진 부분이나 구멍을 채우거나
잡음을 축소하여 제거하는 연산

Mat getStructuringElement(int shape, Size ksize, Point anchor=Point(-1,-1))

- MORPH_RECT
- MORPH_ELLIPSE
- MORPH_CROSS

void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1,
int borderType = BORDER_CONSTANT, const Scalar & borderValu= morphologyDefaultBorderValue())

void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1,
int borderType = BORDER_CONSTANT, const Scalar & borderValu= morphologyDefaultBorderValue())

CHAP 6 05 모폴로지 연산

MORPH_CROSS size=(3,3) iter=3

erode : 반복적인 min filtering

dilate : 반복적인 max filtering

CHAP 6 05 모폴로지 연산

```
void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor=Point(-1,-1),
int iterations=1, int borderType = BORDER_CONSTANT, const Scalar & borderValu= morphologyDefaultBorderValue())
```

op는 모폴로지 연산 방식

- MORPH_OPEN : erode -> dilate
- MORPH_CLOSE : dilate -> erode
- MORPH_GRADIENT : dilate - erode
- MORPH_TOPHAT : src - open
- MORPH_BLACKHAT : close - src

CHAP 6 06 템플릿 매칭

참조 영상에서 템플릿 영상과 매치되는 위치를 탐색하는 방법

물체 인식, 스테레오 영상에서 대응점 검출 -> **이동의 문제 해결**

템플릿을 영상 I(x,y)에서 이동시키며 결과를 R(x,y)에 저장

R(x,y)를 탐색하여 템플릿 위치 찾음

- **상관관계**(correlation) : 최대값의 위치
- SAD(Sum of absolute difference) : 최소값의 위치

→ void matchTemplate(InputArray src, InputArray templ, OutputArray result, int method)

- TM_SQDIFF
- TM_SQDIFF_NORMED

⟷

- TM_CCORR
- TM_ CCORR _NORMED
- TM_CCOEFF
- TM_CCOEFF_NORMED

CHAP 6 06 템플릿 매칭