GrayScale Image Processing

과정: [Intel] 엣지 AI SW 과정
과목명:절차지향 프로그래밍(C)
조원진
블로그:https://blog.naver.com/jowonjin6o1

1. 프로젝트 개요

- 목표 : 파이썬을 기반으로 4가지 영상처리를 구현
  1.화소점 처리, 2.기하학 처리, 3.히스토그램 처리, 4.화소영역 처리

- 기능
  - 기본 기능: 윈도우창, 영상처리버튼
  - 영상처리기능 : 1.화소점 처리 2.기하학 처리
    3.히스토그램 처리 4.화소영역처리

- 개발 환경 OS: Window 10(64bit) / pycham / python

- 전체코드: https://blog.naver.com/jowonjin0601/223388017993

2. 영상처리에 대한 이해

INPUT	PROCESSING	OUTPUT
데이터를 받아와 전역변수로 저장	INPUT 이미지를 받아오고 Main함수에서 원하는 영상처리기법을 택한 후 영상처리	영상처리한 이미지를 출력 또는 저장

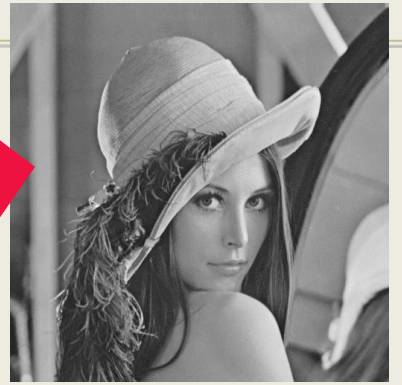
3.화소점 처리

- 원 화소의 값,위치로 단일 화소 값을 변경
- 다른 화소 영향 X
- 화소의 점의 값만 변경 (Point Processing)

3.화소점 처리

흑백(F)

동일(A)



반전(E)

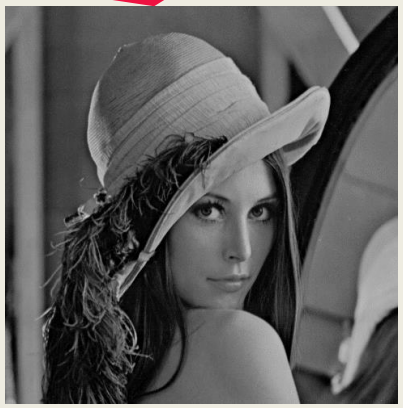
밝게(40)(B)

곱(3)(D)

어둡게(40)(C)

```
outImage[i][k] = inImage[i][k]
```

(B) + value
(C) - value
(D) * value
(E) 255 – inImage[i][k]
(F)outImage= 255 or 0



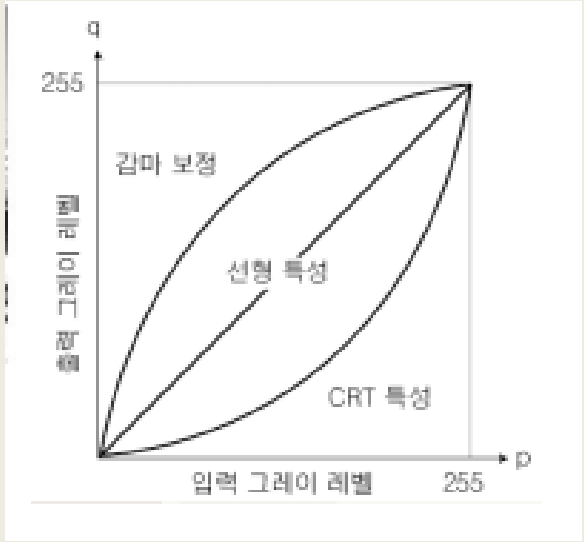
3.화소점 처리

❖ 감마보정

감마 보정 함수: Output(q) = [Input(p)](1/r)
r<1이면 어두워지고 r>1 밝아진다.

핵심코드 $Output(q) = [Input(p)](1/\gamma)$

```
outImage[i][k] = int(((inImage[i][k]/255)**(1/value))*255)
```



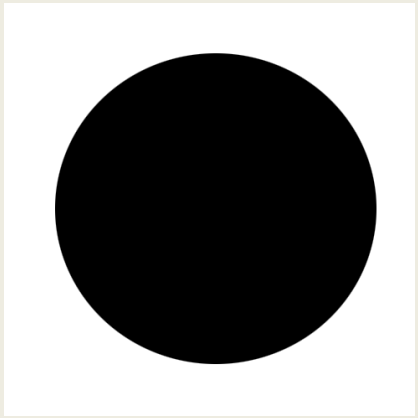
r=0.8

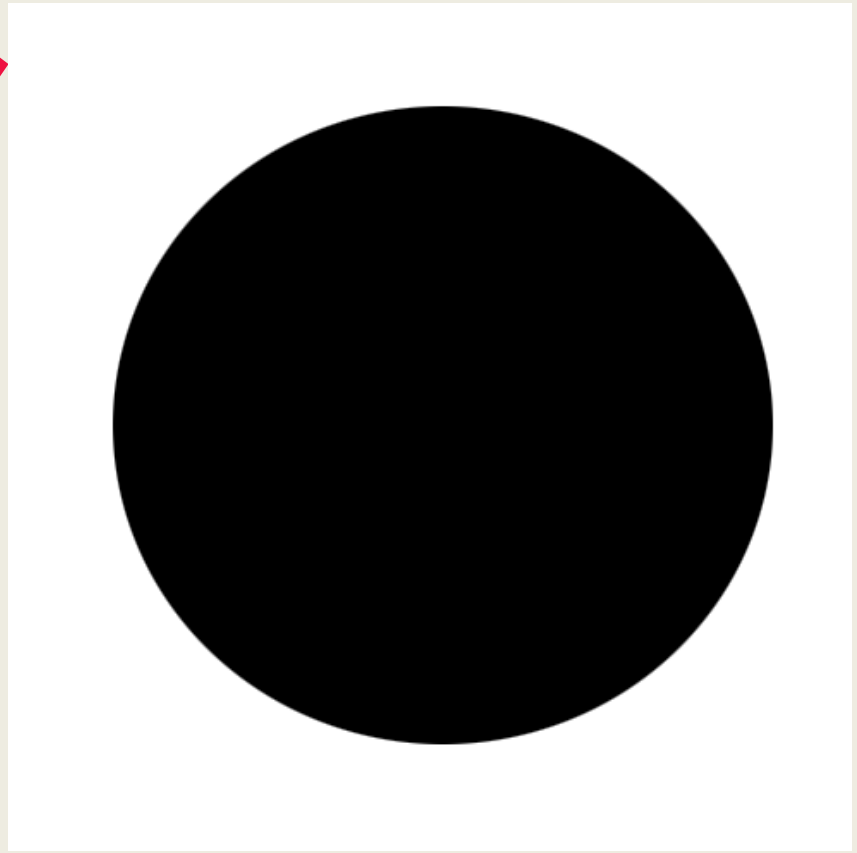
r=2

4.기하학 처리(축소, 확대)

확대

```
outH = inH // scale
outW = inW // scale
outImage[i // scale][k // scale] = inImage[i][k]
```





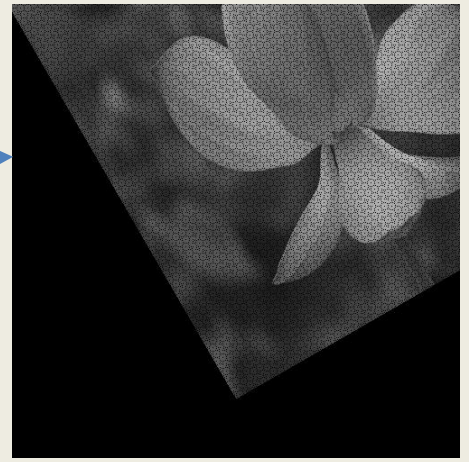
축소

```
outH = inH * scale
outW = inW * scale
outImage[(i * scale)][(k * scale)] = inImage[i][k]
```

4.기하학 처리(회전)

회전 중앙백워딩

회전 포워딩



```
cx = inH//2
cy = inW//2

for i in range(outH):
    for k in range(outW):
        xd = i
        yd = k
        xs = math.cos(radian) * (xd -cx) + math.sin(radian) * (yd - cy)
        ys = -math.sin(radian) * (xd - cx) + math.cos(radian) * (yd - cy)
        xs +=cx
        ys +=cy

        if (0 <= xs and xs < outH and 0 <= ys and ys < outW):
            outImage[int(xd)][int(yd)] = inImage[int(xs)][int(ys)]
```

중앙 기점으로 회전

```
xs = i
ys = k
xd = int(math.cos(radian) * xs - math.sin(radian) * ys)
yd = int(math.sin(radian) * xs + math.cos(radian) * ys)

if (0 <= xd and xd < outH) and (0 <= yd and yd <
outW):
    outImage[xd][yd] = inImage[xs][ys]
```

[o][o]기준으로 회전

4.기하학 처리(반전)

좌우반전

```
outImage[outH-i-1][k] = inImage[i][k]
```

상하반전

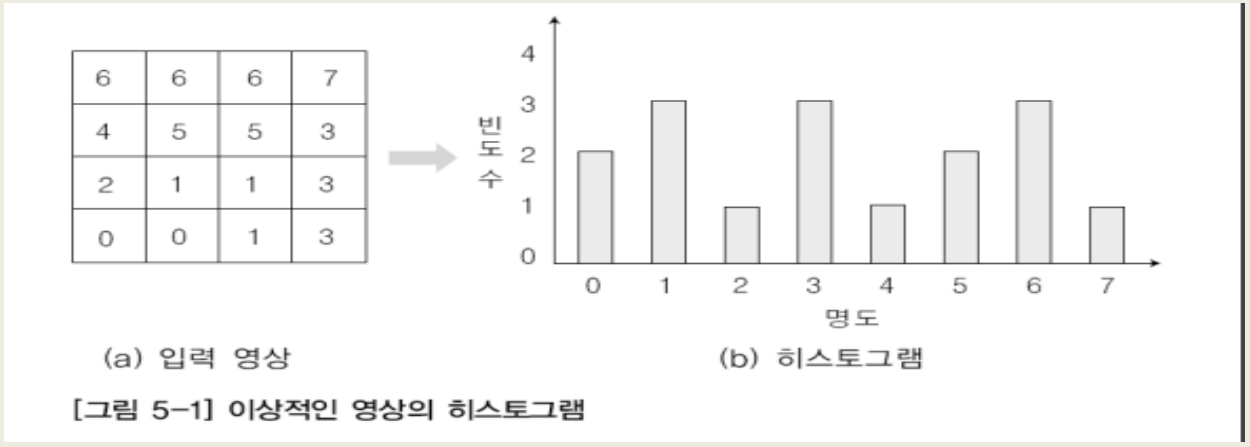
```
outImage[i][outH-k-1] = inImage[i][k]
```

5.히스토그램 처리

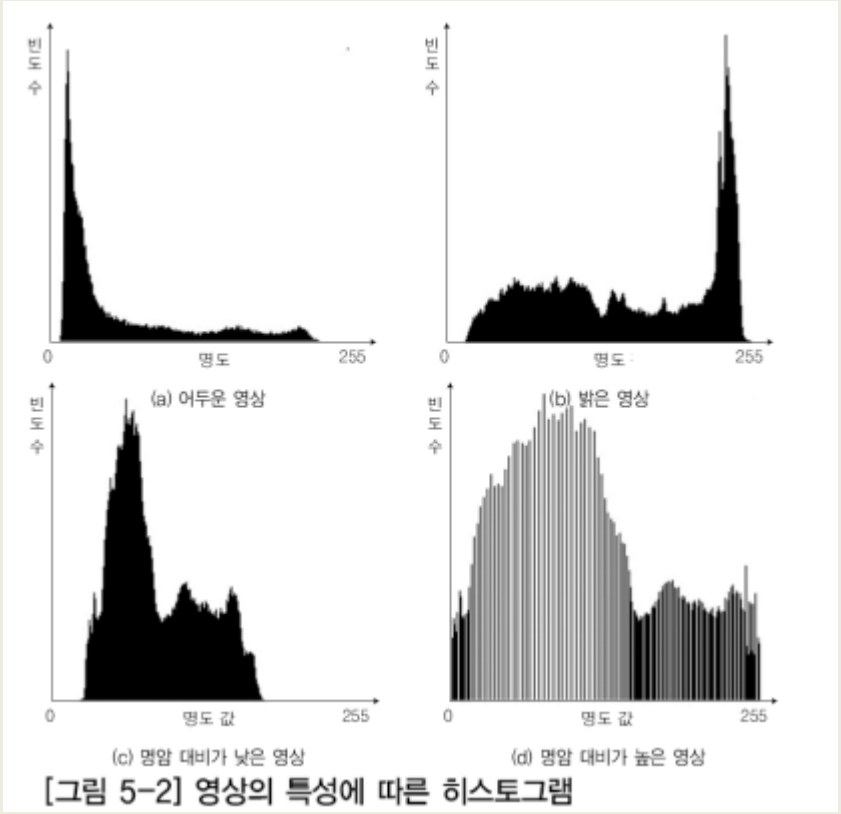
- 디지털 영상의 히스토그램에 관하여

- 관찰한 데이터의 특징을 한눈에 알아볼 수 있도록 데이터를 막대그래프 모양으로 나타낸 것

-디지털 영상에 대한 많은 정보 제공



[그림 5-1] 이상적인 영상의 히스토그램



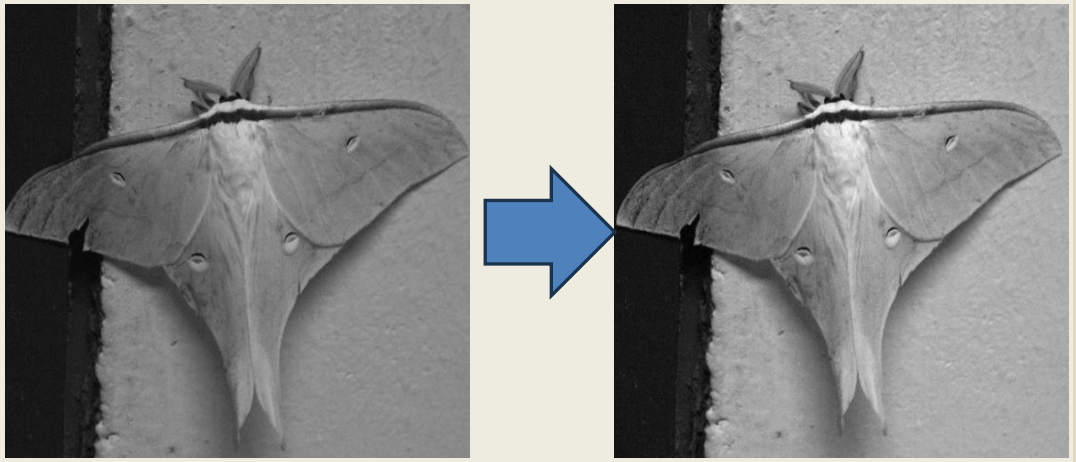
[그림 5-2] 영상의 특성에 따른 히스토그램

5.히스토그램 처리 (스트레칭)

1. 히스토그램 스트레칭

-이상적이지 못한 히스토그램 분포 중에 명암 대비가 낮은 디지털 영상의 품질을 향상시키는 기술

-특정 부분 집중된 히스토그램을 모든 영역으로 확장

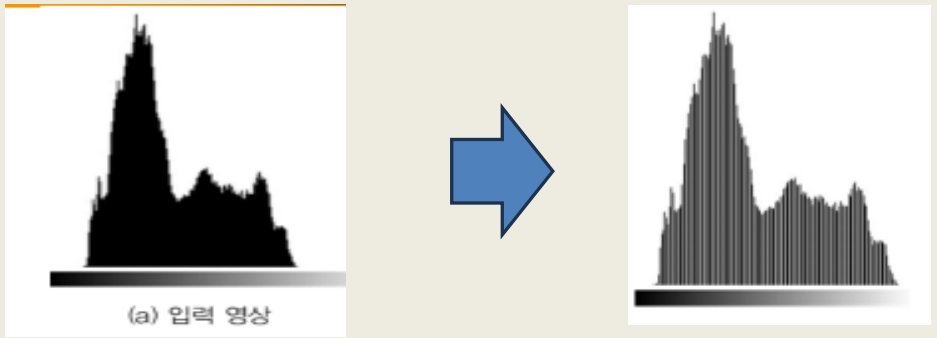


기본 명암 대비 스트레칭 수행 공식

$$new\ pixel = \frac{old\ \ pixel - low}{high - low} \times 255$$

코딩 부분

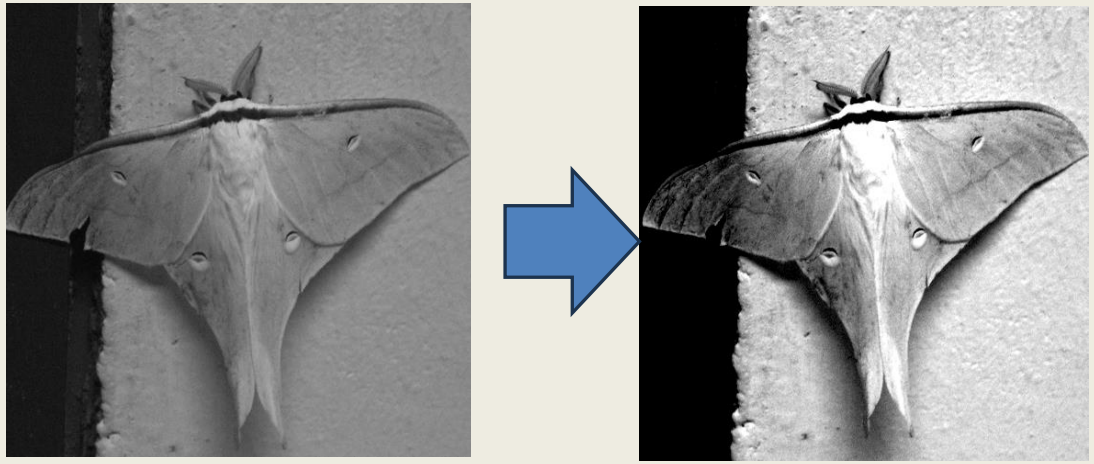
```
old = inImage[i][k]
new = int((old-low)/(high - low)*255.0)
if(new > 255):
    new = 255
if(new< 0):
    new = 0
outImage[i][k] = new
```



5.히스토그램 처리 (엔드인)

2. 엔드인

-일정한양의 화소를 흰색이나 검정색으로 지정하여 히스토그램의 분포를 좀 더 균일하게 만듬



앤드-인 탐색 수행 공식

$$new\ pixel = \begin{cases} 0 & old\ pixel \le low \\ \dfrac{old\ pixel - low}{high - low} \times 255 & low \le old\ pixel \le high \\ 255 & high \le old\ pixel \end{cases}$$

코딩 부분

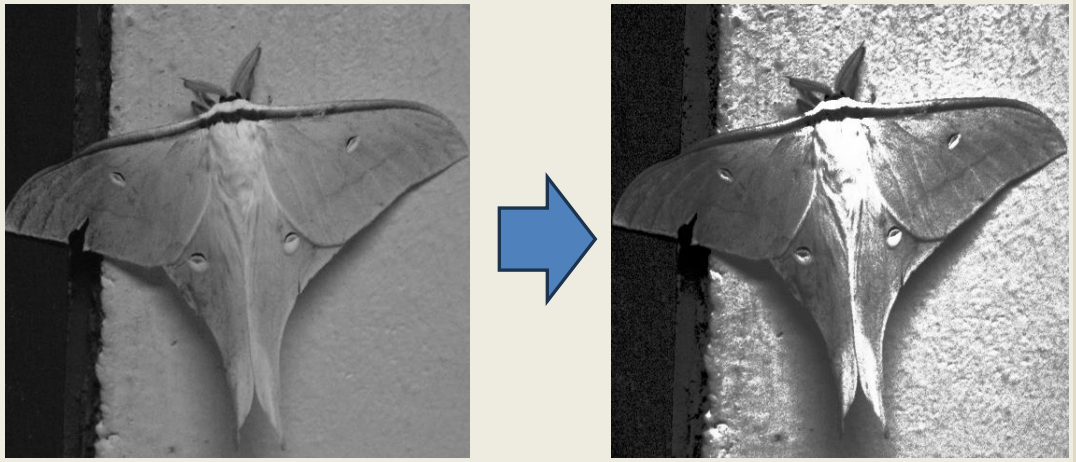
```
high -= 50
low += 50
new, old = [0]*2
for i in range(inH):
    for k in range(inW):
        old = inImage[i][k]
        new = int((old-low)/(high - low)*255.0)
        if(new > 255):
            new = 255
        if(new< 0):
            new = 0
        outImage[i][k] = new
```

5.히스토그램 처리 (평활화)

3. 히스토그램 평활화

-명암 분포가 빈약한 영상을 균일하게 만들고 영상의 밝기 분포 재분배하여 명암 대비를 최대화
-명암 대비 조정을 자동으로 수행
-검출 특성이 좋은 영상만 출력하지는 않지만 영상 검출 특성을 증가시킴

결과값 도출



히스토그램 평활화의 4단계

1.빈도수 세기

```
histo[inImage[i][k]] += 1
```

2.누적히스토그램 생성

```
sumHisto[i] =sumHisto[i-1] + histo[i]
```

3.정규화된 히스토그램 생성

```
normalHisto[i] = sumHisto[i] * (1.0 / (inH * inW)) * 255.0
```

4.inImage를 정규화된 값으로 치환

```
outImage[i][k] = int(normalHisto[inImage[i][k]])
```

6.화소영역 처리

화소 영역 처리

- 주위의 화소 값도 함께 고려하는 공간 영역 연산

-원시 화소와 이웃한 각 화소에 가중치를 곱한 합을 출력 화소로 생성한다.

$$Output\_pixel[x,y] = \sum_{m=(x-k)}^{x+k} \sum_{n=(y-k)}^{y+k} (I[m,n] \times M[m,n])$$

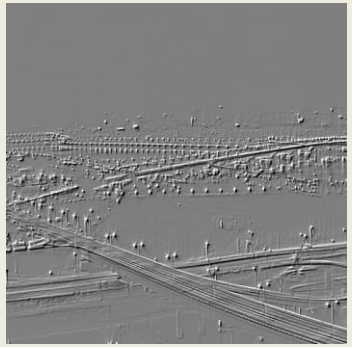
- 엠보싱, 블러링, 샤프닝, 경계선 검출, 잡음 제거 등등 많다.

- 공통 회선 연산 공식:

```
S += tmpInImage[i + m][k + n] * mask[m][n]

tmpOutImage[i][k] = S
```

6.화소영역 처리(엠보싱과 블러링)



엠보싱

블러링 3X3

mask = [[1./9,1./9, 1./9], [1./9, 1./9, 1./9], [1./9, 1./9,1./9]]

블러링 5X5

블러링:
이미지의 세밀한 부분 제거
뿌옇게됨

엠보싱:
입력 영상을
양각으로 표현

mask = [[-1.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 1]]

```
mask = [[1. / 25, 1. / 25, 1. / 25,1. / 25, 1. / 25, ],
        [1. / 25, 1. / 25, 1. / 25,1. / 25, 1. / 25, ],
        [1. / 25, 1. / 25, 1. / 25,1. / 25, 1. / 25, ],
        [1. / 25, 1. / 25, 1. / 25,1. / 25, 1. / 25, ],
        [1. / 25, 1. / 25, 1. / 25,1. / 25, 1. / 25, ]]
```

6.화소영역 처리(가우스, 고주파 샤프닝)

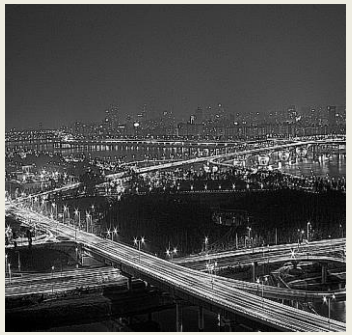
1

```
mask = [[-1,-1, -1], [-1,9, -1], [-1,-1, -1]]
```

고주파샤프닝

-블러링와 반대되는 효과 -> 흐린 영상을 개선
-고주파에 해당하는 상세한 부분을 강조

2



```
mask = [[0,-1, 0], [-1,5, -1], [0,-1, 0]]
```

가우시안 스무딩:
가우시안 필터를
이용한 블러링

```
mask = [[1./16,1./8, 1./16], [1./8,1./4, 1./8], [1./16,1./8, 1./16]]
```

$$G[x,y] = \frac{e^{\frac{-(x^2+y^2)}{2\sigma^2}}}{2\pi\sigma^2}$$

6.화소영역 처리(엣지 검출)

- 엣지

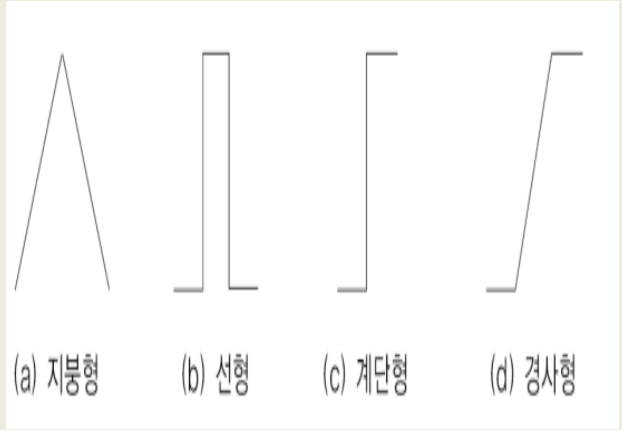
-디지털 영상의 밝기가 낮은 값에서 높은 값으로 혹은 반대로 변하는 지점
-디지털 영상을 구성하는 객체 간의 경계 (경계선)
-디지털 영상의 엣지: 물체 식별, 위치/모양/크기/방향성을 탐지가능성 정보 제공

- 간단한 엣지 추출 기법:

-연산 자체가 간단하고 빠름.
- 유사 연산자와 차 연산자가 있음.
-엣지를 강화하거나 약화시키는 추가적인 임계값을 처리하는 방법이 있음.

- 미분을 이용한 엣지 검출 방법:

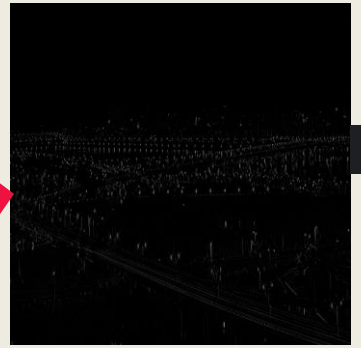
- 엣지가 화소의 밝기 변화율에 관여한다는 것
- 1차 미분을 이용한 검출 방법과 2차 미분을 이용한 검출 방법 있음
-2차 미분을 이용한 검출 방법: 1차 미분으로 얻은 결과에 미분을 한 번 더 추 가하여 에지 검출의 성능을 향상시킨 것



- 다양한 엣지 패턴

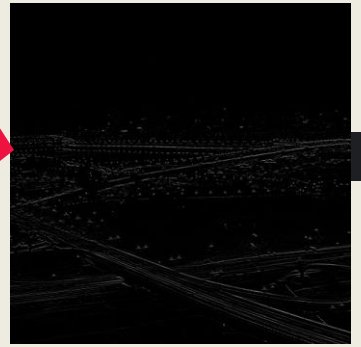
6.화소영역 처리(수평 수직 엣지)

유사연산자



수평엣지

mask = [[0.0,0.0, 0.0], [-1.0,1.0,0.0], [0.0,0.0,0.0]]



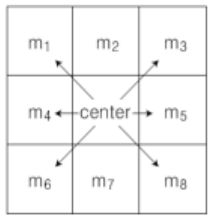
수직엣지

mask = [[0.0,-1.0, 0.0], [0.0,1.0,0.0], [0.0,0.0,0.0]]

유사 연산자 엣지 :

간단한 연산으로 화소를
감산한 값에서 최대값을
구해 엣지를 검출하는 방식

최대값 구하는 방법:



New Pixel = max(|center − m1| ··· |center − m8|)
총 8번 계산

6.화소영역 처리(1차 미분 엣지 검출)

- 1차 미분 엣지 검출

  -에지는 화소의 밝기 변화가 급격히 변하는 부분

  -미분 연산을 이용하여 급격한 변화 부분을 탐지

종류: (1)로버트 마스크

(2)프리윗 마스크

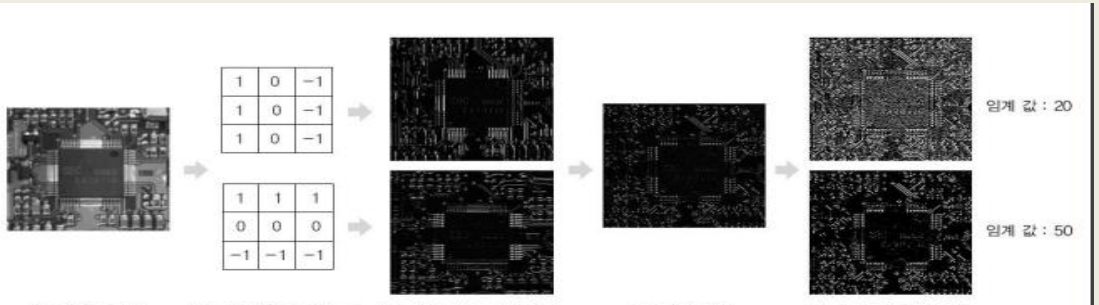
(3)소벨 마스크

좌표(x,y)에서 각 방향으로의 편미분

$$\nabla H(x,\ y)=\begin{bmatrix}H_r(x,\ y)\\ H_c(x,\ y)\end{bmatrix}=\begin{bmatrix}\frac{\partial f(x,\ y)}{\partial x}\\ \frac{\partial f(x,\ y)}{\partial y}\end{bmatrix}=\begin{bmatrix}f(x+1,\ y)-f(x,\ y)\\ f(x,\ y+1)-f(x,\ y)\end{bmatrix}$$

영상의 전체 변화 분의 크기 계산

$$H(x,\ y)\approx|H_r(x,\ y)|+|H_c(x,\ y)|=\sqrt{H_r^2(x,\ y)+H_c^2(x,\ y)}$$

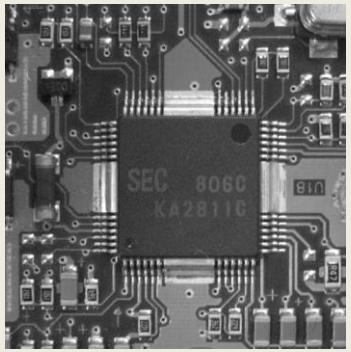


[그림 7-9] 1차 미분 회선 마스크를 이용한 에지 검출 영상

6.화소영역 처리(로버츠)

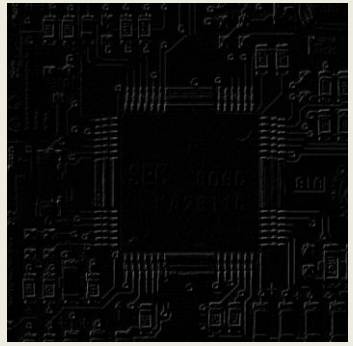
-크기가 작아 빠른 속도로 작동

-돌출된 값을 잘 평균할 수 없으며 잡음에 민감



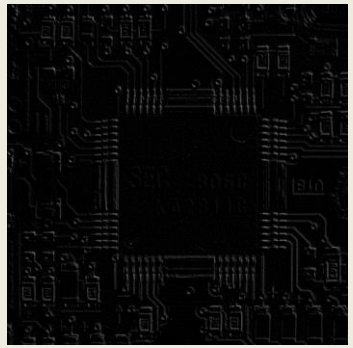
로버츠 행 검출

mask = [[-1,0, 0], [0,1, 0], [0,0,0]]



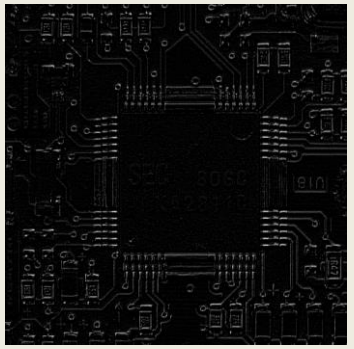
로버츠 열 검출

mask = [[0,0,-1], [0,1, 0], [0,0,0]]



엣지 합성

mask = [[-1,0,-1], [0,2, 0], [0,0,0]]

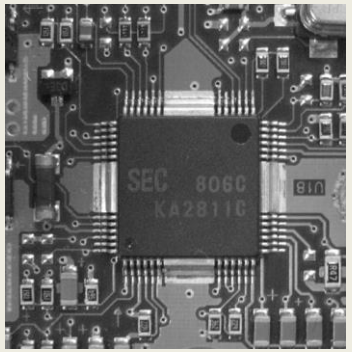


임계값 강조

6.화소영역 처리(프리윗)

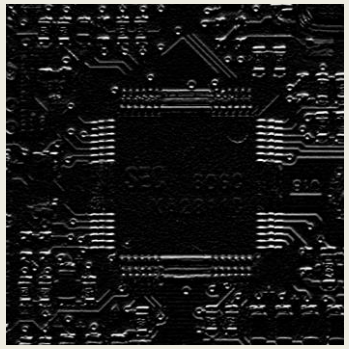
-돌출된 값을 잘 평균화함

-대각선보다 수평, 수직 에지에 더 민감



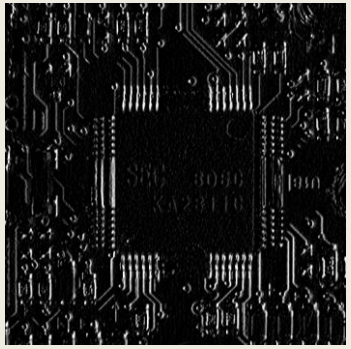
프리윗 행 검출

mask = [[-1,-1, -1], [0,0,0], [1,1,1]]



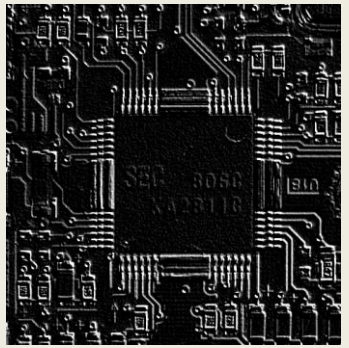
mask = [[1,0,-1], [1,0, -1], [1,0,-1]]

프리윗 열 검출



엣지 합성

mask = [[0,-1,-2], [1,0, -1], [2,1,0]]

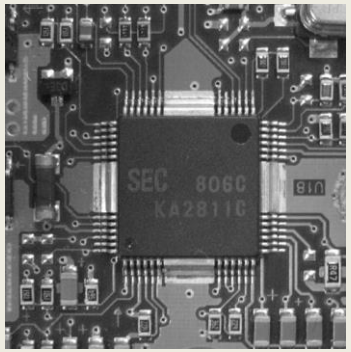


임계값 강조

6.화소영역 처리(소벨)

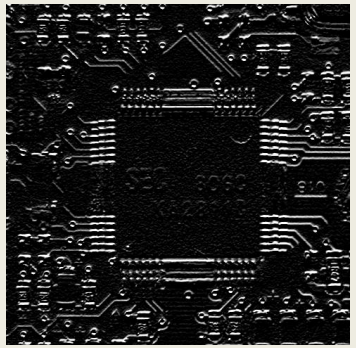
-돌출된 값을 비교적 잘 평균화

-대각선 방향 엣지에 더 민감



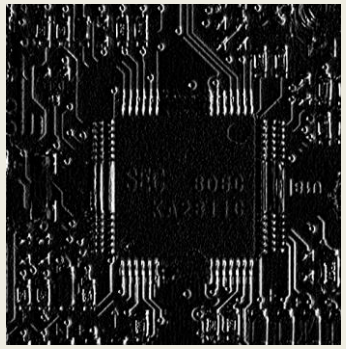
소벨 행 검출

mask = [[-1,-2,-1], [0,0, 0], [1,2,1]]



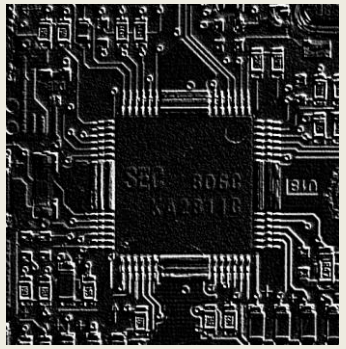
mask = [[1,0,-1], [2,0,-2], [1,0,-1]]

소벨 열 검출



엣지 합성

mask = [[0,-2,-2], [2,0, -2], [2,2,0]]



임계값 강조

6.화소영역 처리(2차 미분 엣지 검출)

10. 라플라시안 검출

대표적인 2차 미분 연산자로, 모든 방향의 엣지를 강조함. 임계값 이상 엣지만 검출

라플라시안의 공식

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

$$\frac{\partial^2 f(x, y)}{\partial x^2} = f(x+1, y) - 2f(x, y) + f(x-1, y)$$

$$\frac{\partial^2 f(x, y)}{\partial y^2} = f(x, y+1) - 2f(x, y) + f(x, y-1)$$

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

대표적인 라플라시안 회선 마스크

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

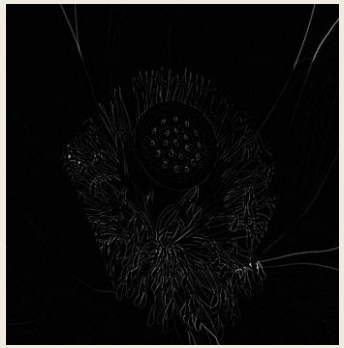
(a) (b) (c)

사진 원본

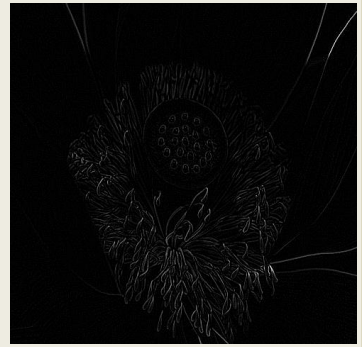
결과값 도출

(a)

(b)



(c)



6.화소영역 처리(2차 미분 엣지 검출)

10. LoG(Laplacian of Gaussian) 연산자

-잡음에 매우 민감한 라플라시안 마스크를 이용한 에지 검출기의 문제점을 해결 위해 만듬

-가우시안 스무딩을 수행하여 잡음 제거 한 뒤에 에지 강조하기 위해 라플라시안을 이용함

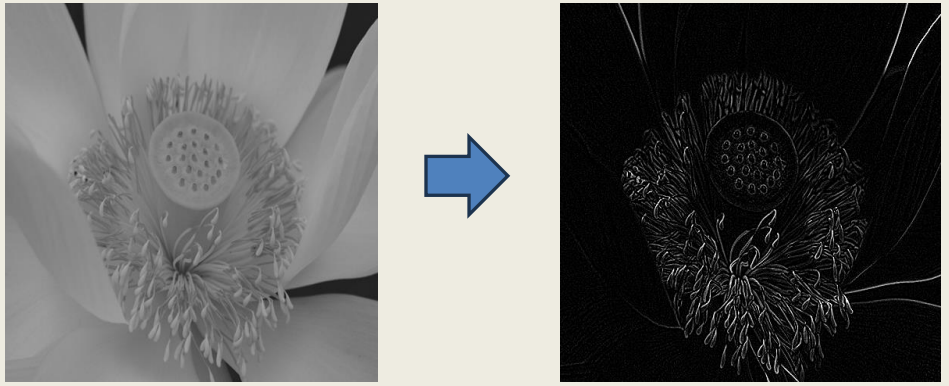
LoG 연산자 공식

$$\mathrm{LoG}(x,\ y) = \frac{1}{\pi\sigma^4}\left[1 - \frac{(x^2+y^2)}{2\sigma^2}\right] - e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

대표적인 log 회선 마스크

```
{ 0, 0, -1, 0, 0}, //
{ 0, -1, -2, -1, 0},
{ -1, -2, 16, -2, -1},
{ 0,-1,-2,-1,0 },
{ 0,0,-1,0,0 }
```

결과값 도출



6.화소영역 처리(2차 미분 엣지 검출)

10. DoG(Laplacian of Gaussian) 연산자

-계산 시간이 많이 소요되는 LoG 연산자의 단점을 보안

-각 가우시안 연산에 분산 값을 서로 다르게 주어 이차를 이용해 에지 맵을 구함

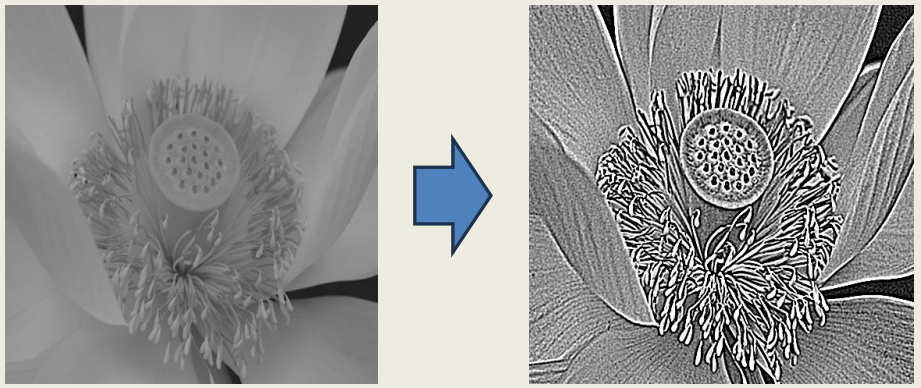
DoG 연산자 공식

$$DoG(x,y) = \frac{e^{\frac{-(x^2+y^2)}{2\sigma_1^2}}}{2\pi\sigma_1^2} - \frac{e^{\frac{-(x^2+y^2)}{2\sigma_2^2}}}{2\pi\sigma_2^2}$$

대표적인 dog 회선 마스크

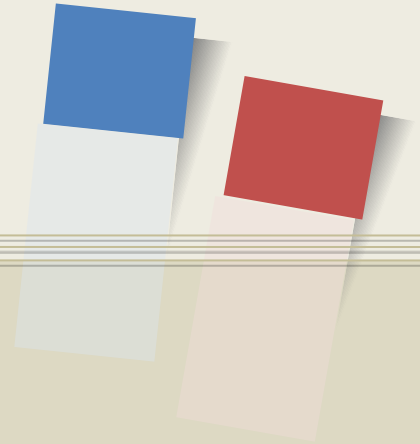
```
{0,0,-1,-1,-1,0,0},
{0,-2,-3,-3,-3,-2,0},
{0,-3,5,5,5,-3,-1},
{-1,-3,5,16,5,-3,-1},
{-1,-3,5,5,5,-3,-1},
{0,-2,-3,-3,-3,-2,0},
{0,0,-1,-1,-1,0,0},
```

결과값 도출



7. 프로젝트 마치며

- 파이썬을 사용하면서 느낀점:
  1. 사용을 안 해본 python을 사용하면서 여러가지 문법을 익힘
  2. C언어보다 더욱 간결하고 직관적인 문법을 사용하면서 코드 줄 수를 획기적으로 줄임
  3. C언어와 다르게 작은 실수들도 그대로 돌아가 오류를 일으킴
- 한계점:
  1. 여러 사진을 동시에 편집을 할 수 없음
  2. 파일 처리에 대해 더욱 많은 함수들을 이해할 수 있어야 함
  3. 익숙지 않은 문법으로 많은 시간이 소요됨
- 향후 발전 방향
  1. 여러 사진을 동시에 편집할 수 있도록 코드 추가
  2. 색상 있는 파일로도 영상처리 할 수 있도록 코드 추가
  3. python에서만 사용가능했던 기능들을 C언어에서도 적용



Thank you

과정: [Intel] 엣지 AI SW 과정
과목명:절차지향 프로그래밍(C)
조원진
블로그:https://blog.naver.com/jowonjin6o1