컴퓨터비젼 기말 프로젝트 보고서

2018110658 윤재성

Edge Detector

선택한 언어 : Python

상세한 코드 설명은 소스 코드 파일(.ipynb)의 주석으로 하였습니다.

나는 파이썬을 이용해서 Edge Detector를 구현하는 프로젝트를 하였다.

Edge는 이미지 내에서, 특징이 단절되는, 두 영역이 분리되는 영역, 컬러 또는 밝기값들이 변화가 발생하는 부분이다. 밝기에 따라 지형으로 표현하고 지형으로 표현된 컬러값에서 밝기값들의 변화가 발생한다는 것은 경사가 심한 부분이고, 그 경사가 그래디언트를 의미한다. 따라서 이 그래디언트를 계산하면 Edge를 검출할 수 있다. 2차원영상 픽셀 값들은 discrete한 값으로 표현 되어있기 때문에 실제로 gradient를 계산하려면 미분을 할 수는 없고, approximation을 해야한다. 이 때 Tailor Series를 이용해서 approximation 을 하는데, 이 approximation 을 나타내는 derivative filter는 [-1,0,1] 이다. 여기서 양쪽 픽셀을 이용해서 approximation 을 하는 이유는 한 쪽 픽셀만 사용할 때보다 에러가 더 작아지기 때문이다. 이제 이 derivative filter를 이용해서 Gradient를 구하고 그 Gradient는 벡터이므로 magnitude를 구하여 적절한 threshold를 정해 edge를 검출한다. 그러나 Gradient로만 edge를 검출하게 되면 noise가 많기 때문에 원하는 edge들만 얻어지는 것이 아니다. 따라서 나는 noise를 제거하기 위해 스무딩을 해야하는데, 이 때 사용하는 필터가 Box filter 혹은 Gaussian filter이다. Box filter는 주변의 픽셀 color 값들이 비슷할 때 사용하는데, 모든 픽셀의 weight 값들을 평균 값으로 넣는 것이다. Gaussian filter는 자기 픽셀과 주변 픽셀들의 weight 값들을 가우시안 분포에 따라 부여하는 것이다. 그런데 자기 픽셀과 주변 픽셀들의 weight 값들이 같은 것이 문제가 있고, 주변 픽셀 color 값들이 언제나 비슷하지는 않으므로,Box Filter보다는 Gaussian filter가 자연현상의 상황들을 더 잘 표현한다. 따라서 나는 Box filter도 써보았지만, Gaussian filter도 사용해서 edge를 검출해보았다. Gaussian filter를 이용해서 edge를 검출하는 방법으로는, derivative filter와 convolution property를 이용하여 이미지에 convolution 하기 전에 먼저 convolution 을 하고 그 뒤에 이미지에 convolution 을 하면 노이즈가 어느정도 제거가 된 edge detected 이미지를 얻을 수 있다. 나는 이것에 더해서 두번 미분하고 가우시안 필터를 적용하는 Laplacian of Gaussian을 구현하였는데, Laplacian of Gaussian이 미분을 2번하기 때문에 노이즈도 증폭되고, 계산량도 많아지기 때문에, Laplacian of Gaussian과 비슷한 결과를 내는서로 다른 sigma 값을 가진 두 Gaussian 분포의 차이를 나타내는 Difference of Gaussian으로도 edge detector를 구현해보았다. Laplacian of Gaussian과 Difference of Gaussian은 Derivative of Gaussian 과 다르게 threshold 값을 지정해서 threshold보다 큰 부분을 edge를 검출하는 것이 아니라, zero-crossing이 일어난 부분을 edge로 검출한다. 나는 이렇게 5가지 방법으로 edge detector를 구현하였다.

프로젝트에 사용한 기법 및 방법, 함수 구현

- 흑백 이미지를 사용할 경우를 위한 gray scaling 함수 구현
- x, y 방향 convolution 함수 구현 (2차원,3차원(rgb 포함))
- 이미지의 출력 크기가 입력보다 줄어들지 않기 위해 입력 이미지에 zero-padding 기법 적용
- 가우시안 커널의 Sigma 값을 다르게 하여 스무딩 정도 관측
- LoG와 Difference of Gaussian의 경우, 0~255의 중간값인 128을 zero crossing의 기준으로 적용
- Laplacian, Gaussian, Derivative filter 생성 함수

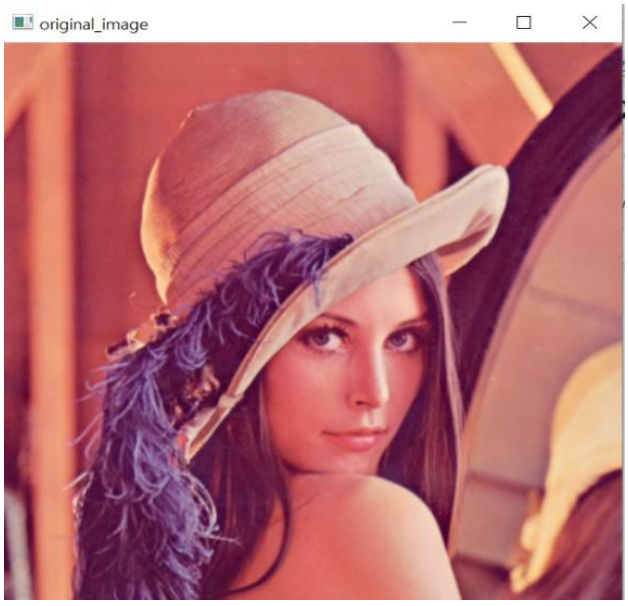
  ➔ 가우시안 필터 생성시 가우시안 분포 공식 적용

$$G_\sigma \equiv \frac{1}{2\pi\sigma^2}\exp\{-\frac{x^2+y^2}{2\sigma^2}\}$$

- 외부 라이브러리는 numpy, cv2 사용

  ➔ Numpy는 배열을 다루는데 사용, cv2는 이미지 로드 및 저장, 보여주기 용도로만 사용

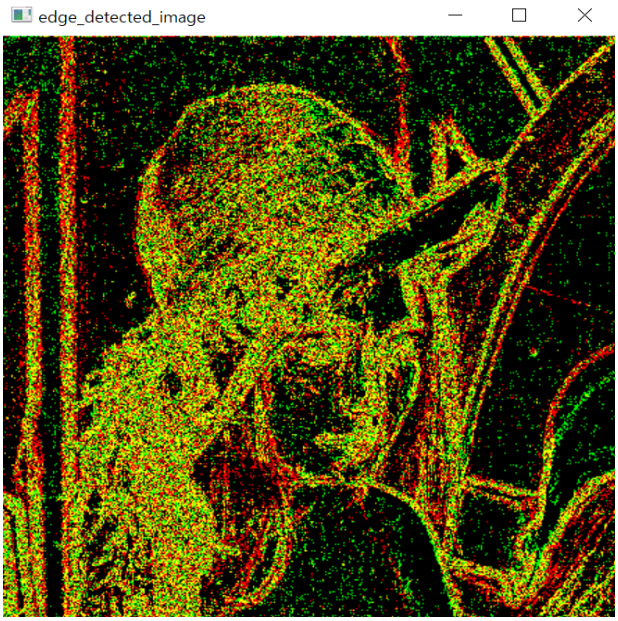
Original Image



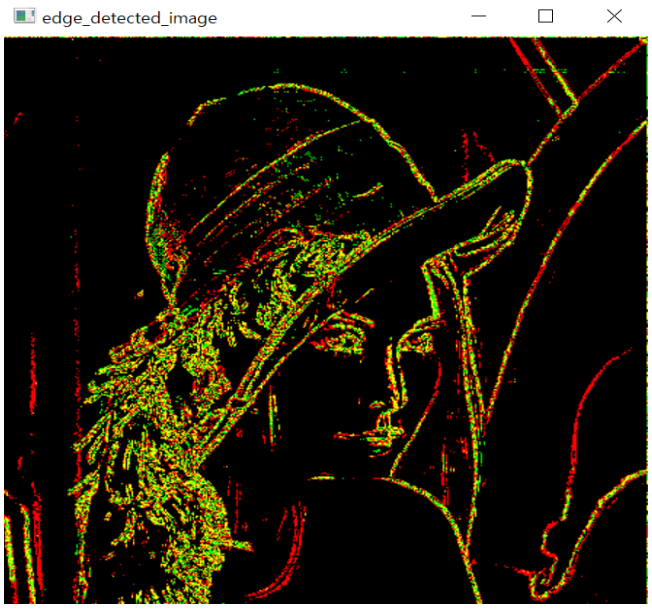
** 동일한 threshold 를 사용하여 성능 비교 진행 (최적화된 threshold 값은 아님)

Kernel = 3x3, threshold = 10

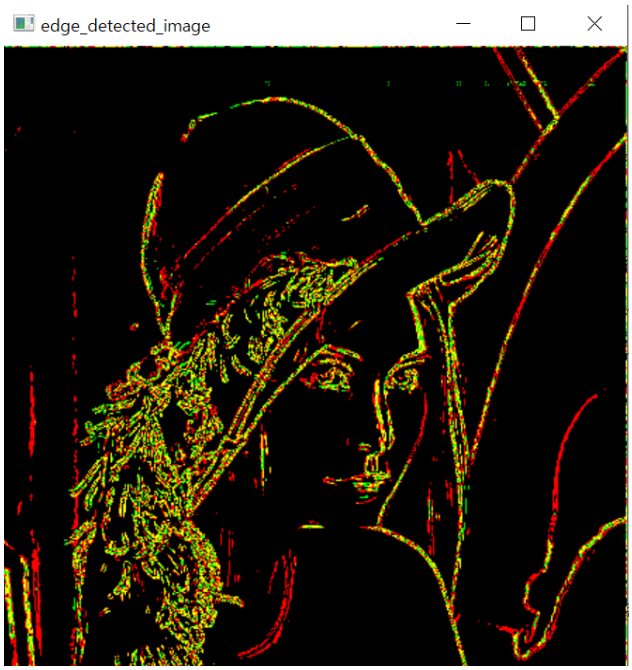
Gradient



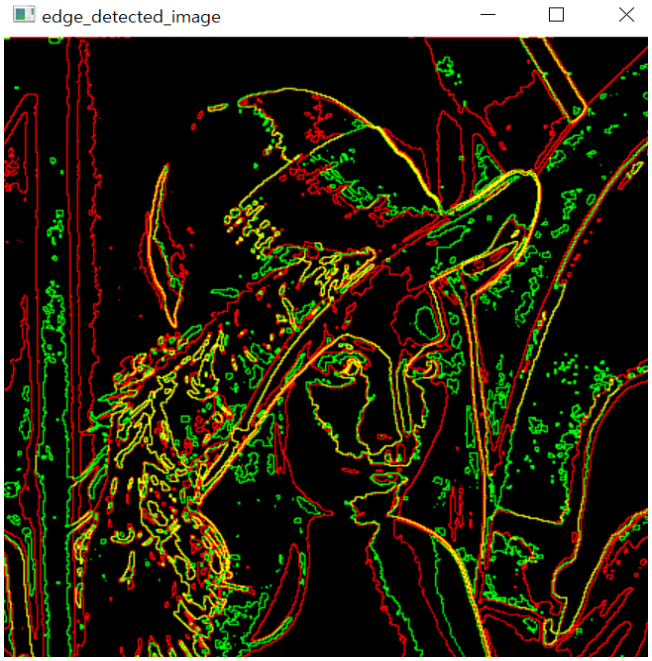
Derivative of Box Filter



Derivative of Gaussian(DoG)



Laplacian of Gaussian(LoG)



Difference of Gaussian (sigma1 = 1, sigma2 =2)

Gradient 만 사용해서 edge를 검출한 결과, 스무딩을 하지 않았기 때문에 노이즈가 상당히 많은 것을 확인할 수 있다. 따라서, 스무딩이 필요한데, Box filter를 적용한 결과 어느정도 노이즈가 제거 되긴 했지만, Gaussian filter를 적용했을 때가 좀 더 깔끔하게 noise가 제거 되고, 이미지의 눈동자 같은 부위에서 boxfilter보다 Gaussian filter가 더 눈동자 의 edge 부분을 더 잘 나타내고 있음을 확인할 수 있다.

Laplacian of Gaussian filter와 sigma1,sigam2 로 한 Difference of Gaussian 도 적용해보았는데, 둘 다 edge가 어느정도 검출이 된 것을 확인할 수 있다. 다만 LoG의 경우는, 일반적으로 미분을 하면 노이즈가 증폭이 되는데 Second Derivative 즉, 두 번 미분을 하는 것을 approximation 했기 때문에 노이즈가 한 번 미분할 때보다 더 증폭된다. 따라서 Gaussian filter를 적용하더라도 노이즈가 어느정도 있는 것을 볼 수 있다. 그런데, LoG의 경우, 이미지의 모자 부분이나, 어깨부분처럼 edge부분이 아니어도 빛을 받는쪽 과 빛을 덜 받는 쪽이 어느정도 경계가 된 부분도 표현하고 있는 것을 확인할 수 있다.

Derivative of Gaussian filter 에서 가우시안 커널의 sigma 값이 스무딩 및 edge 검출에 어떤 영향을 미치는지 확인하기 위해 threshold 값은 같게 하고 igma 값을 다르게 하여 Gaussian filter를 만들고 convolution 및 edge 검출을 진행하였다.

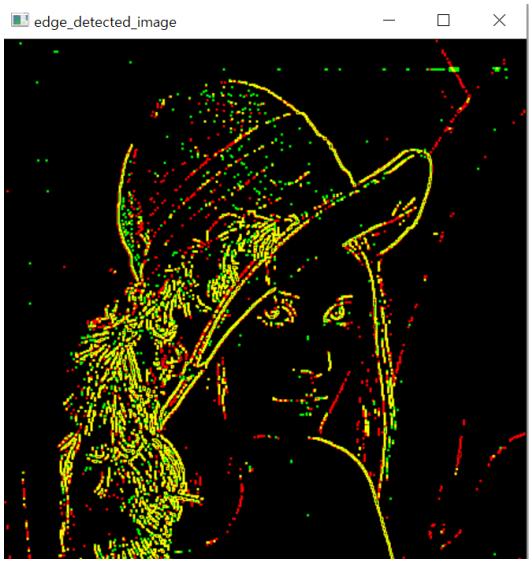
Sigma = 1

Sigma = 3

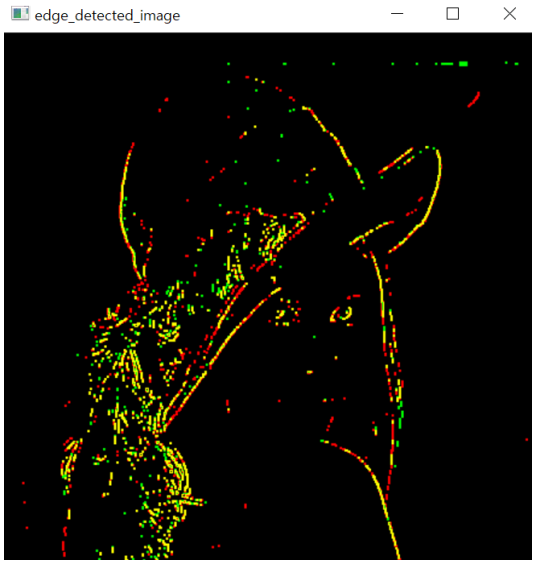
Sigma = 1 일 때보다 Sigma = 3 일 때 Edge 부분에서 조금 끊어지는 부분이 더 있지만, 확실히 노이즈가 줄어든 것을 확인할 수 있다. 따라서 Sigma 값이 커지면, 스무딩이 더 많이 된다. Sigma 값을 적절하게 조절하여 Edge들을 잘 검출하면서도, 노이즈를 제거하는 것이 중요하다.

Difference of Gaussian filter에서 가우시안 커널의 Sigma 값이 스무딩 및 edge 검출에 어떤 영향을 미치는지 확인해보기 위해서 sigma 값을 다르게 하여 convolution을 진행하고 edge를 검출해보았다.

(sigma1 = 1, sigma2 =2)



(sigma1 = 1.25, sigma2 = 2.5)



➔ Sigma 값이 더 커짐에 따라 커널에서 더 높은 가중치가 부여가 되기 때문에 스무딩이 더 많이됨을 알 수 있었다.

1D Gaussian과 1D derivative filter 를 각각 x,y방향으로 convolution 하는 것과

2D Gaussian과 2D derivative filter 를 convolution 하는 것이 edge 검출 성능과 연산 시간에 있어 어떤 차이가 있는지 비교해보았다.

1D x(1x3) , y(3x1)

2D (3x3)

연산 시간

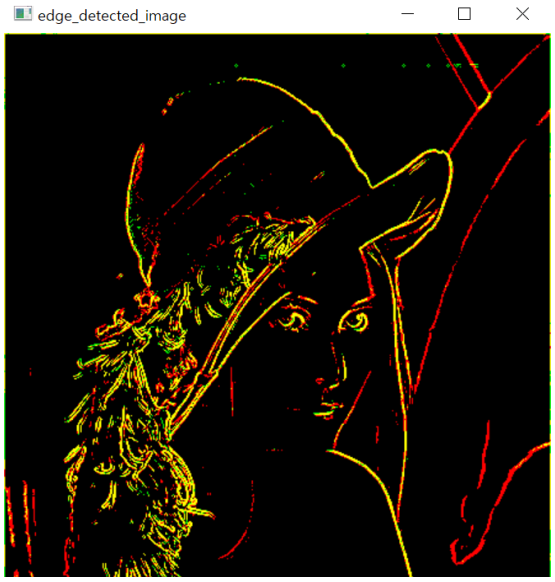
```
1D: 8.91267549999975
2D: 4.237443899997743
```

경과시간을 측정하기 위해서 time 라이브러리를 import하여서 사용하였습니다. (비젼과 관련 없다고 판단)

연산시간과 edge 검출 모두 2D convolution을 한 번에 하였을 때가 더 시간이 적게 걸리고, noise가 더 적은 것을 확인할 수 있었다.

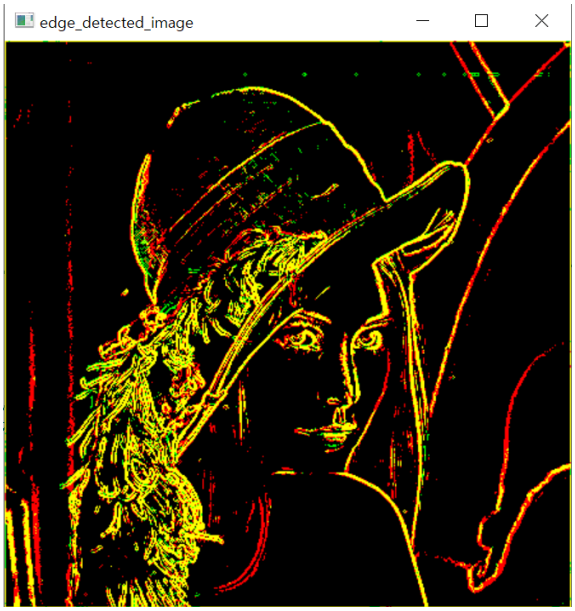
성능 향상을 위해 어떤 방법이 있을까 고민해보다가 가우시안 필터와 Derivative 필터를 Convolution 하는 것이 아닌, Multiply를 해보는 것은 어떨까 하고 두 필터의 각 요소를 곱해준 값을 이미지의 필터로 적용하여 edge를 검출해보고 convolution 과 어떤 차이가 있는지 비교해보았다.

multiply (threshold =5)



convolution (threshold = 5)

multiply (threshold = 3)



convolution (threshold = 3)

같은 threshold를 적용하였을 때, multiply가 noise 측면에서는 noise가 더 제거되는 장점이 있었지만, threshold가낮지 않음에도, edge로 검출되어야 할 것이 잘 검출 되지 않은 단점이 있었다. 따라서 convolution 방법을 선택하고 threshold 값을 잘 조절하여 노이즈를 최적으로 제거하는 방법이 더 좋은 방법이라고 생각된다.