

Spatial Filtering of Gray Images

전자공학과 20161453 김규래

1 Overview

이번 과제의 목적은 spatial filtering 을 구현해보고, padding 을 적용한 다음 각각의 특성을 특성을 이해하는 것이다. 또한 Gaussian smoothing, box smoothing, sobel edge detection, laplacian edge enhancement 와 같은 광범위하게 사용되는 필터들을 사용해봄으로서 이들의 특성 또한 파악하는 것이 목표이다.

구현한 함수들은 다음과 같으며, 순서대로 spatial filtering 함수, gaussian kernel 생성 함수, laplacian kernel 생성 함수, sobel kernel 생성 함수, box kernel 생성 함수이다.

```
inline cv::Mat conv2d(cv::Mat const& src, kernel_t const& kernel, pad_type pad);
inline kernel_t kernel::gaussian_filter(float sigma, size_t n);
inline kernel_t kernel::laplacian_filter(bool is_4_direction)
inline kernel_t kernel::sobel_filter(bool is_x_direction)
inline kernel_t kernel::box_filter(size_t n)
```

1.1 padding

conv2d 는 기본적으로 'same' 모드로, 입력되는 src 영상과 동일한 크기의 영상을 반환한다. Finite discrete domain 에서는 spatial filtering 을 수행할 경우 정보의 손실이 발생하면서 영상의 크기가 줄어들기 때문에 같은 크기를 유지하기 위해서는 보정을 해야한다. 이 보정을 padding 이라고 하며, 일반적으로는 다음과 같은 3가지를 사용한다

1. **Zero padding** 영상의 바깥을 전부 0 으로 취급.
2. **Extension padding** 영상의 바깥부분의 픽셀들을 영상에서 가장 가까운 픽셀로 대체.
3. **Mirror padding** 영상의 바깥 부분에 영상을 대칭으로 연장.

Zero padding 의 경우에는 접근 인덱스가 영상의 테두리를 벗어나는 경우 영상의 값을 0 으로 대체하면 되고, extension padding 의 경우 접근 인덱스를 clamping 해서 구현할 수 있다. 마지막으로 mirror padding 은 인덱스가 영상을 벗어나는 만큼 영상의 방향으로 인덱스를 변환하는 것으로 구현할 수 있다. 본 구현에서는 zero padding 과 extension padding 을 구현하였으며 conv2d 함수에 인자를 전달해서 선택을 할 수 있게 하였다.

2 Implementation

언어는 C++14 를 사용하였으며 빌드 시스템은 CMake 로 구성하였다. 리눅스 환경에 맞게 작성이 됐기 때문에 리눅스에서 빌드할 것을 권장한다.

2.1 External Dependencies

외부 라이브러리로는 OpenCV¹를 사용하였다. OpenCV에서는 다양한 영상처리 관련 루틴들을 제공하고 있으나, 본 과제에서는 `cv::imshow()`와 `cv::imread()`, `cv::imwrite()`와 같은 입출력 관련 함수들만 사용하였다.

2.2 Installation

시스템에 OpenCV가 표준 경로에 설치돼있을 경우 CMake에서 자동으로 인식을 할 것이다. 별도의 경로에 설치돼 있을 경우 `OpenCV_ROOT_DIR` 변수를 CMake에 플래그로 넘김으로써 경로를 알려줄 수 있다. Blaze의 경우에는 함께 첨부돼 있기 때문에 별도의 작업이 필요하지 않다. 다음과 같이 입력하면 빌드를 할 수 있다.

```
mkdir build && cd build
cmake -G ‘‘Unix Makefiles’’ ..
make
```

2.3 구현1

구현1에서는 box kernel과 gaussian kernel을 이용하여 smoothing 연산을 수행하는 것이 목표이다. Gaussian kernel의 경우에는 Eq.1를 이용하여 생성하였다. 이 때 k 는 normalization constant로, 생성되는 커널의 모든 coefficient들의 합의 역수이다. Eq.2는 $\sigma = 1.0, n = 7$ 일 때 생성되는 gaussian blur 커널이다.

$$K_{\text{gauss}}(x, y, \sigma) = \frac{1}{k} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (1)$$

$$K_{\text{gauss}} = \begin{pmatrix} 0.000036 & 0.000363 & 0.001446 & 0.002291 & 0.001446 & 0.000363 & 0.000036 \\ 0.000363 & 0.003676 & 0.014662 & 0.023226 & 0.014662 & 0.003676 & 0.000363 \\ 0.001446 & 0.014662 & 0.058488 & 0.092651 & 0.058488 & 0.014662 & 0.001446 \\ 0.002291 & 0.023226 & 0.092651 & 0.146768 & 0.092651 & 0.023226 & 0.002291 \\ 0.001446 & 0.014662 & 0.058488 & 0.092651 & 0.058488 & 0.014662 & 0.001446 \\ 0.000363 & 0.003676 & 0.014662 & 0.023226 & 0.014662 & 0.003676 & 0.000363 \\ 0.000036 & 0.000363 & 0.001446 & 0.002291 & 0.001446 & 0.000363 & 0.000036 \end{pmatrix} \quad (2)$$

Box kernel은 Eq.3와 같이 생성하였으며 이를 통해서 $n = 3$ 일 때 생성되는 커널은 Eq.4와 같다. 이 때 n 은 box kernel의 가로, 세로 크기이다.

$$K_{\text{box}}(x, y, n) = \frac{1}{n^2} \quad (3)$$

$$K_{\text{gauss}} = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix} \quad (4)$$

다음은 제시된 영상 `Test_pattern.tif`에 gaussian smoothing을 적용했을 때의 결과이다.

¹ <https://opencv.org/>

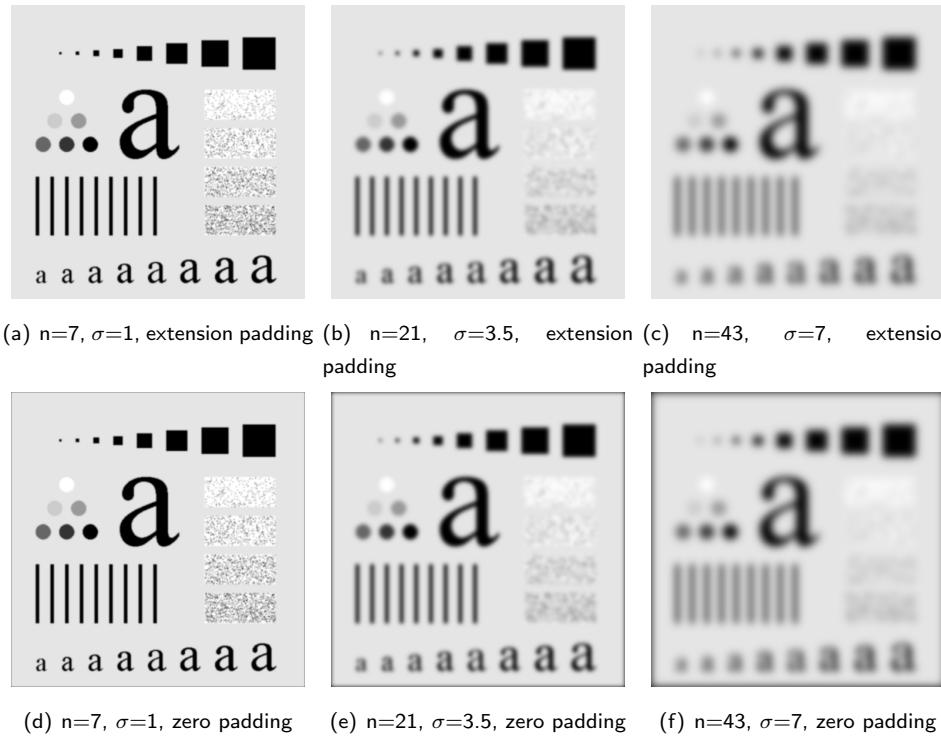


Fig. 1: Gaussian filtering 을 적용했을 때의 모습. (열) kernel size 와 σ 의 값에 따른 차이. (행) extension padding 과 zero padding 여부에 따른 차이

Fig.1 를 보면 gaussian filtering 을 적용했을 때의 효과를 볼 수 있다. Kernel 의 사이즈, σ 가 증가함에 따라 영상의 선명도가 확연히 감소하는 것을 확인할 수 있다. 또한 zero padding 을 사용할 경우 영상의 테두리 부분에 검은색으로 왜곡이 발생하는 것을 볼 수 있다.

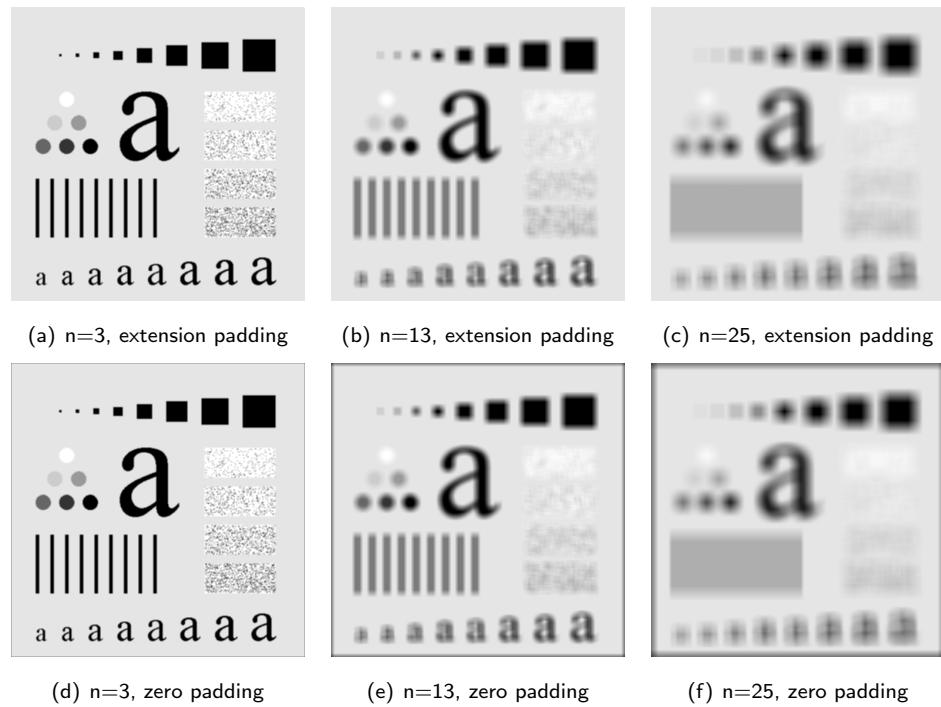


Fig. 2: Box filtering 을 적용했을 때의 모습. (열) kernel size 와 σ 의 값에 따른 차이. (행) extension padding 과 zero padding 여부에 따른 차이

Box filtering 을 사용할 때의 결과는 Fig.2 에서 볼 수 있다. 전체적으로는 gaussian filtering 과 비슷하게 영상의 선명도 떨어지는 효과를 얻는 것을 볼 수 있다. Kernel 의 크기가 증가할수록 선명도가 감소한다. 또한 gaussian filtering 에서와 비슷하게 zero padding 을 사용하는 경우에 태두리에 검은색 왜곡이 발생한다.

검토사항 Box kernel 은 x축과 y축에 대해서 moving average filter 와 유사한 형태를 띤다. Moving average filter는 high frequency 신호의 강도가 약해지는 low pass filter 이기 때문에, 주어진 영상의 수직선과 같이 x 축을 따라서 변화가 크게 발생하는(주파수가 큰) 신호의 경우 필터링돼 버린다. 그 효과로 수직선들 사이의 경계가 흐려진 것을 시각적으로 확인할 수 있다. Spatial domain 에서 생각을 할 경우 box kernel 효과는 kernel 의 크기만큼의 픽셀들을 평균 내버리는 것이기 때문에, 수직선의 경계가 뭉개져 버린 것으로 생각을 할 수 있다.

2.4 구현2

구현2에서는 laplacian kernel 을 구현해서 edge enhancement, image sharpening 효과를 확인하는 것이 목적이다. 사용하는 kernel 은 2가지 종류이며, 각각 4-direction laplacian kernel 5 과 8-direction laplacian kernel 6 이다. 이 필터의 경우 제시된 Boy.tif 영상을 이용하였다.

$$K_{\text{laplace-4}} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad (5)$$

$$K_{\text{laplace-8}} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \quad (6)$$

Fig. 3 를 보면 laplacian filter 를 이용해서 추출한 edge response 를 볼 수 있다.

시각적으로 볼 때는 4-direction 과 8-direction 의 추출효과는 큰 차이가 없다. 다만 이는 Fig. 3의 영상들이 normalization 과 8-bit quantization 을 거쳤다는 점을 고려해야 한다. Normalization 과 quantization 을 거치지 않았을 때는 magnitude 가 차이를 보였을 가능성이 있다. 여기서 눈여겨볼 수 있는 점은 이전 Section 의 smoothing filter 들과 다르게 zero padding 과 extension padding 이 큰 차이를 보이지 않는다는 점이다. 이는 사용되는 kernel 의 사이즈가 크지 않고, 사용된 영상의 테두리 근처가 어두운 색이라서 edge 특성이 강하지 않기 때문으로 보인다. 아래는 원본 영상과 laplacian kernel 로 추출한 edge response 를 더했을 때의 결과물이다.

Fig. 4에서 edge enhancement 효과를 확인할 수 있다. Zero padding 과 extension padding 이 큰 차이를 보이지 않는 만큼, 여백이 부족하여 zero padding 을 적용한 영상은 제시하지 않았다. 다만 제출한 소스코드에서는 zero padding 을 적용한 영상 또한 생성을 한다.

결과를 보면 원본 영상에 비해 edge enhancement 를 적용한 영상들이 더 선명하다는 것을 시각적으로 확인할 수 있다. 또한 8-direction laplacian kernel 을 사용했을 때가 4-direction kernel 을 사용했을 때보다 선명도가 높은 것으로 나타난다. 이를 통해서 단순히 추출된 edge response 영상에서는 확인하기 힘든 두 kernel 간의 magnitude 차이가 존재함을 추론할 수 있다.

2.5 구현3

마지막으로 구현3에서는 sobel kernel 을 사용하여 edge detection 을 수행하였다. 사용한 kernel 은 7, 8 로 각각 x축과 y축 방향으로 central differentiation 을 수행한다.

$$K_{\text{sobel}_x} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad (7)$$

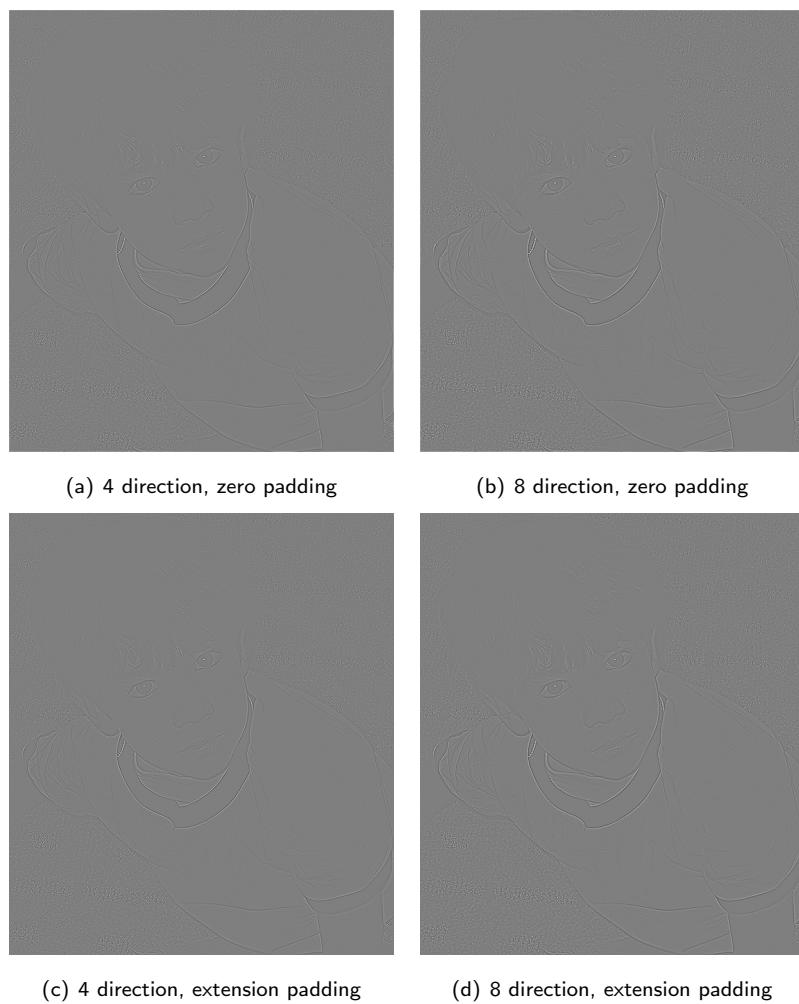


Fig. 3: Laplacian filter을 적용한 값. (열) direction에 따른 차이. (행) extension padding과 zero padding 여부에 따른 차이

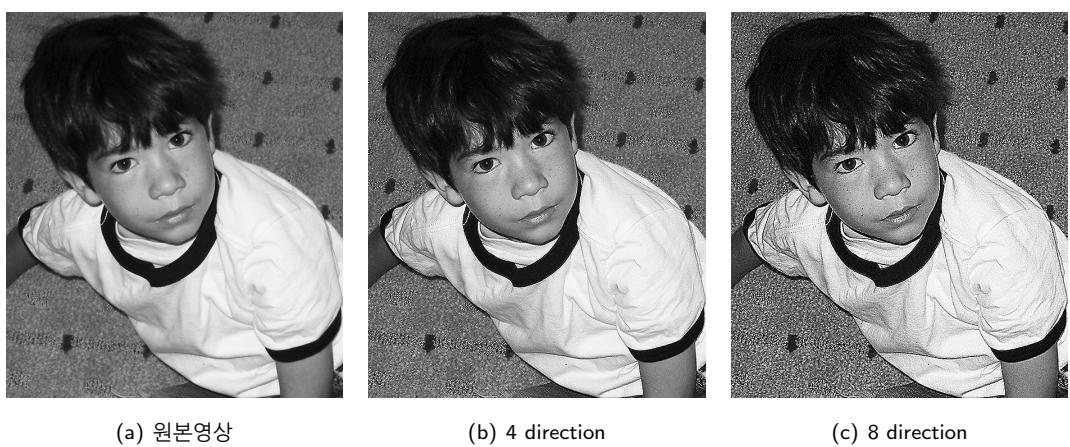


Fig. 4: Laplacian filter를 이용한 edge enhancement 적용 효과

$$K_{\text{sobel}_y} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 2 \end{pmatrix} \quad (8)$$

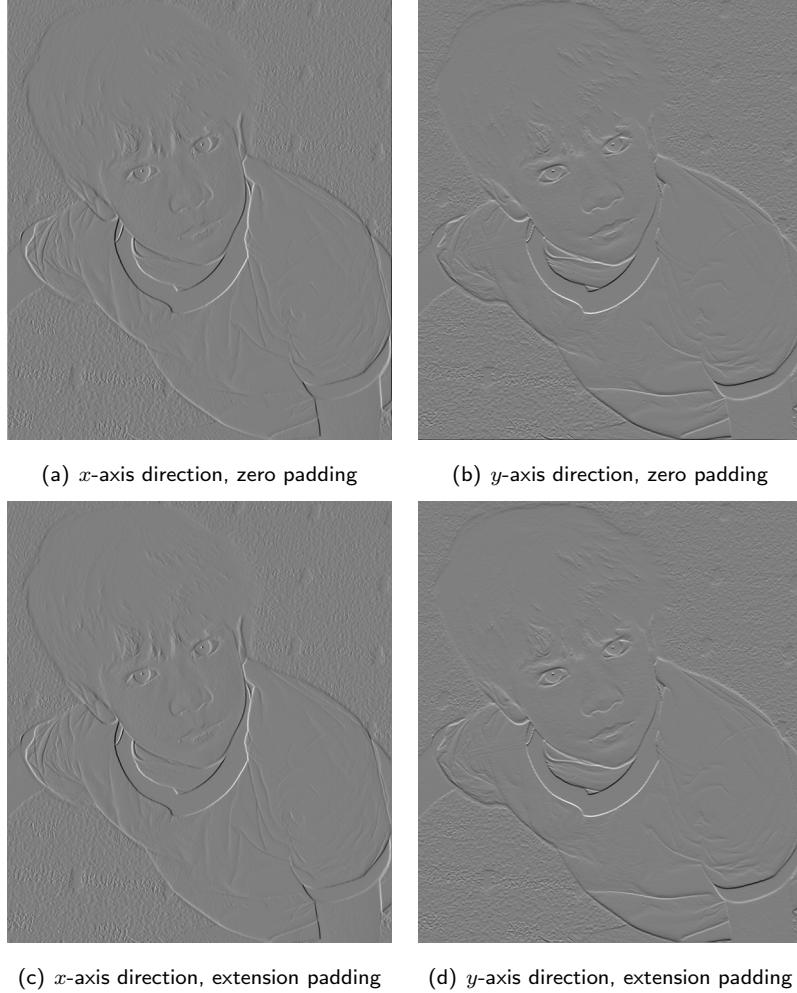


Fig. 5: Sobel operator를 적용한 값. (열) direction에 따른 차이. (행) extension padding과 zero padding 여부에 따른 차이

Sobel operator를 적용했을 때의 결과는 Fig. 5에서 볼 수 있다. x -axis로 sobel operator를 적용했을 때는 세로로 filter response가 크게 나타나고, y -axis로 적용했을 때는 가로로 filter response가 크게 나타난 것을 볼 수 있다. 또한 기준으로 한 방향에 따라서 비슷한 형태의 엣지라도 특정 영역에서는 response가 양수로 나타났는데 반해 특정 영역에서는 음수로 나타난 것을 알 수 있다. 따라서 의미 있는 값을 얻어내기 위해서는 별도로 Norm 연산을 취해야 하나, 본 과제에서는 요구되지 않았으므로 생략하였다. 마지막으로, laplacian filter를 적용했을 때와 비슷하게 zero padding과 extension padding의 차이가 크게 나타나지 않았다.

3 conclusion

이번 과제에서는 2D convolution을 이용한 linear spatial filtering을 C++로 구현하였고, gaussian blur, box blur, laplacian edge enhancement, sobel edge detection 등의 고전적인 image operation들을 직접 수행하였다. 이들의 효과를 실제 영상에 적용하여 확인하였고, extension padding과 zero padding 두 종의 padding을 구현하여 이들의 차이 또한 분석하였다.