

Course Title (과목명)	디지털영상처리개론 (01)
HW Number (HW 번호)	HW02
Submit Date (제출일)	2021-04-19
Grade (학년)	4 <sup>th</sup> Grade
ID (학번)	20161482
Name (이름)	박준용

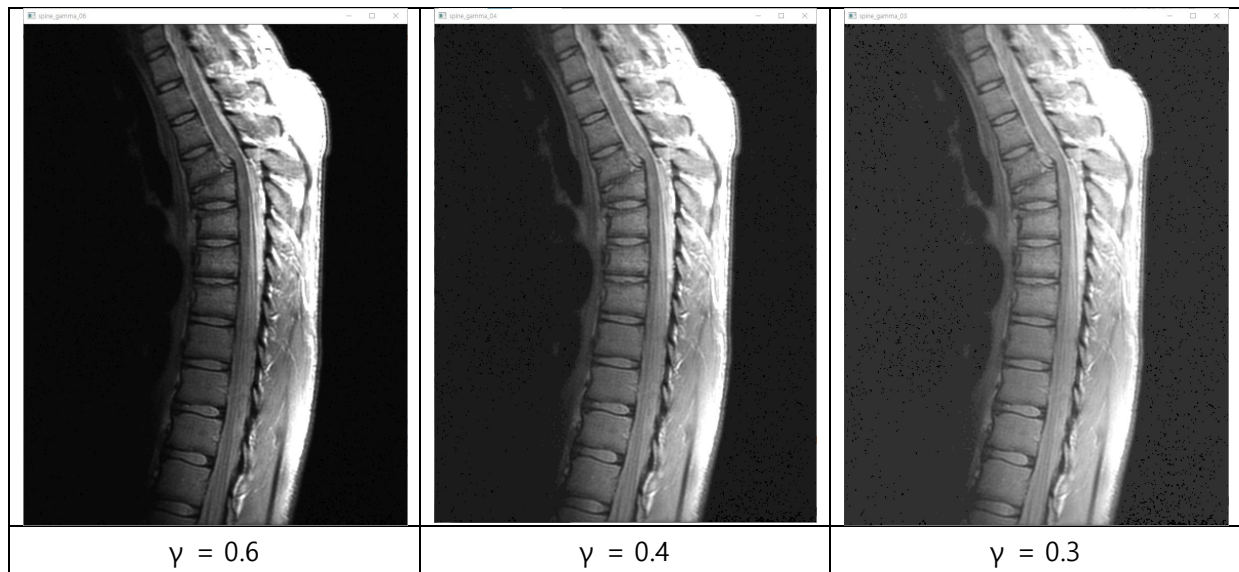
## Intensity Transformations and Spatial Filtering

- Intensity transform과 histogram modification을 이용한 contrast enhancement를 구현
- Spatial kernel과 convolution을 이용한 spatial filtering을 구현
  - ✓ OpenCV의 Mat class method 및 입/출력 함수(imread와 imwrite)만을 사용함.
  - ✓ 제공된 HW02.cpp 및 utils.h파일은 수정하지 않고, utils.cpp파일을 작성하여 제출함.

### (1) 구현 1

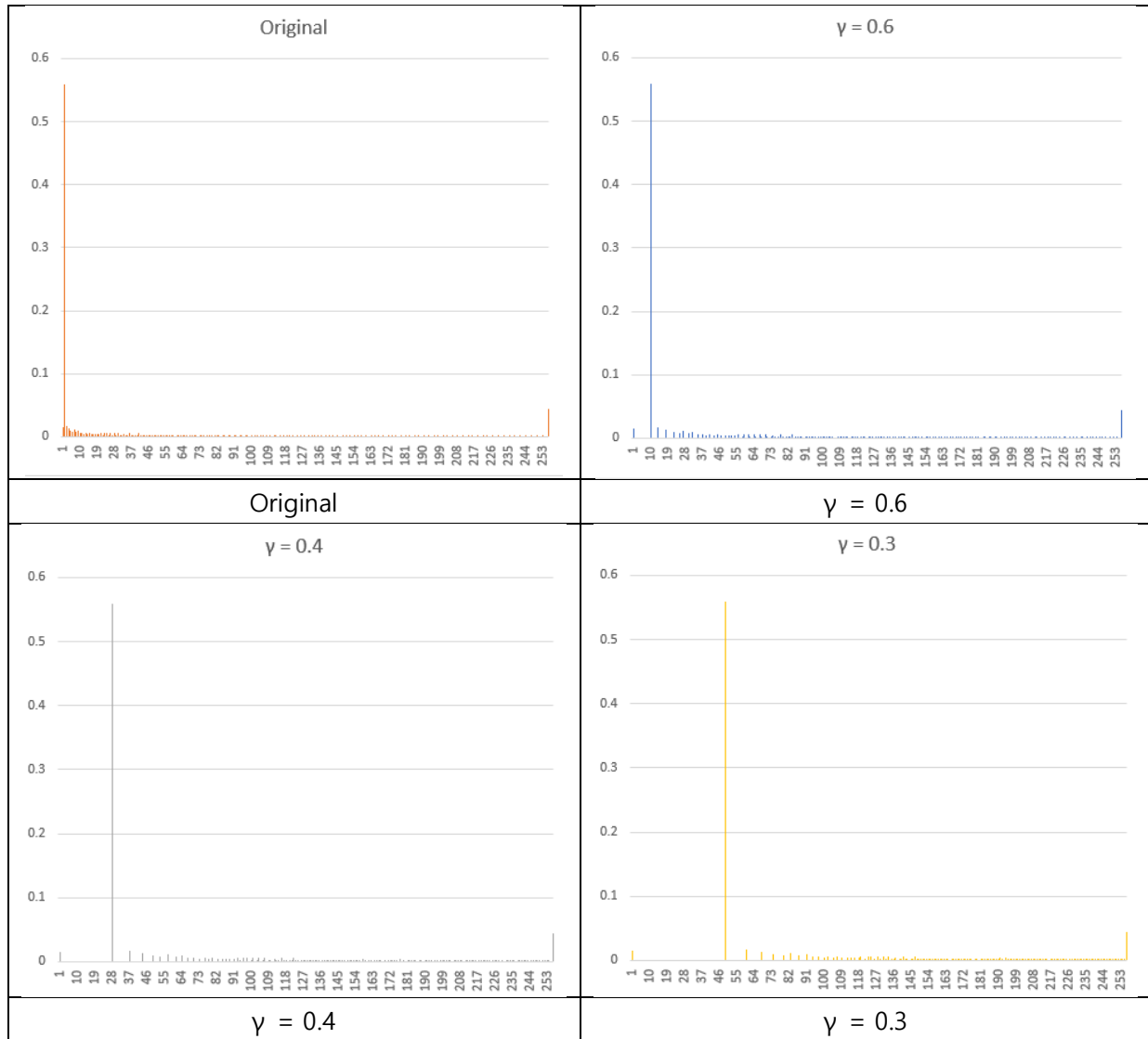
a) \*.zip에 포함된 "spine.tif"에 **gamma transform**을 적용하시오.

- ✓ 3가지  $\gamma$  값에 대하여 적용하시오. ( $\gamma = 0.6$ ,  $\gamma = 0.4$ ,  $\gamma = 0.3$ )



b) 입력영상과 출력영상(3가지)의 **histogram**을 출력하시오

- ✓ histogram 분포는 excel등을 이용하여 plot함.



입력 영상과 각각의  $\gamma$ 값에 대한 출력 영상의 histogram을 비교해볼 수 있다. 우선, 원본 영상은 0에 치우쳐 있고 분산이 큰 pixel intensity를 가진 histogram이 출력되는 반면, gamma transformation을 적용한 경우 분산이 줄어들며 치우친 pixel intensity 값이 원본 영상보다 밝은 값으로 출력되는 것을 histogram을 통하여 알 수 있다. Gamma value가 작아질수록 영상이 밝아지며, 분산이 작아지는 것을 확인할 수 있다.

c) 아래 6.3)를 참고하여 `gammaTransform()`, `getHist()`을 작성하시오.

```
float* getHist(Mat src) {
    float* hist = (float*)calloc(256, sizeof(hist));
    for (int y = 0; y < src.cols; y++) {
        for (int x = 0; x < src.rows; x++) {
            hist[src.at<uchar>(x, y)]++;
        }
    }
    for (int i = 0; i < 256; i++) {
        hist[i] = hist[i]/(src.cols * src.rows);
    }
}
```

```

        return hist;
    }

    Mat gammaTransform(Mat src, float gamma) {
        Mat dst = Mat::zeros(src.rows, src.cols, CV_8U);
        for (int y = 0; y < dst.cols; y++) {
            for (int x = 0; x < dst.rows; x++) {
                dst.at<uchar>(x, y) = (int) (pow((float)src.at<uchar>(x, y) / 255,
gamma) * 255);
            }
        }

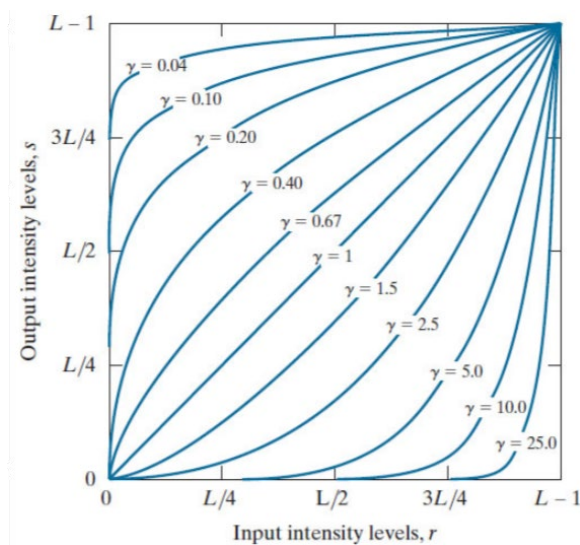
        std::ofstream gammaTransform;
        float* src_hist = getHist(src);
        float* dst_hist = getHist(dst);

        gammaTransform.open("gammaTransform.csv");
        for (int i = 0; i < 256; i++) {
            gammaTransform << i << "," << src_hist[i] << "," << dst_hist[i] << "\n";
        }
        gammaTransform.close();
        return dst;
    }
}

```

우선, gammaTransform() 함수를 구현하기 전에 function identify를 위하여 getHist() 함수를 먼저 작성하였다. getHist() 함수는 image Matrix만을 input으로 받는다.

먼저 histogram 값을 저장하기 위하여 float array를 만들어 memory 할당을 한 뒤 변수를 초기화 시킨다. 이후, input image의 pixel 개수를 count하여  $h(r_k)$ 를 얻은 뒤, 이를 input image의 height와 width를 곱한 값을 통하여 normalized histogram  $p(r_k) = \frac{h(r_k)}{MN}$ 를 구할 수 있도록 code를 작성하여 구현하였다.

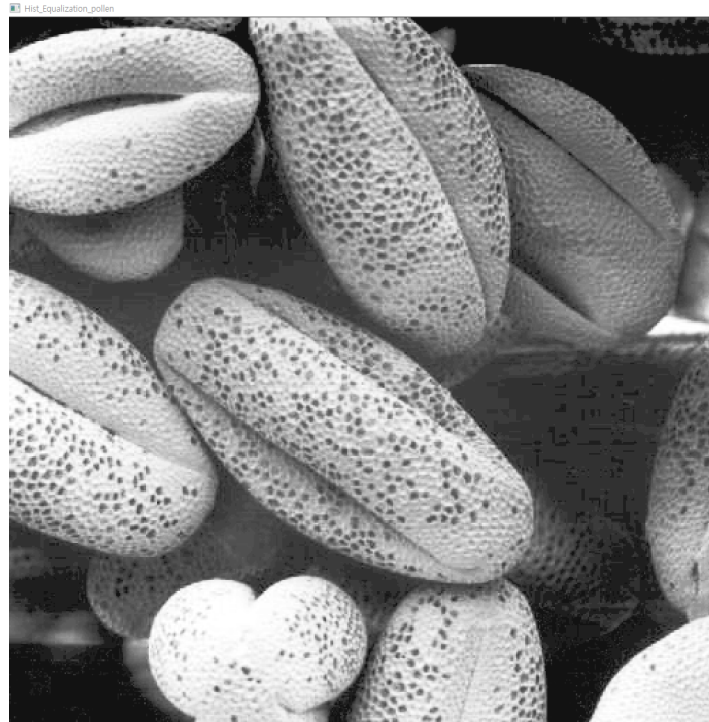


그 다음으로, gammaTransform() 함수의 경우 image Matrix와 gamma float value를 입력으로 받으며, input image의 pixel 개수를 count한 뒤 cmath library의 pow함수를 사용하여

normalized pixel intensity  $r$ 과 gamma value를 제공한다. 이후  $c$ 값을 곱하여 gamma transformation인  $s = cr^\gamma$ 를 반환할 수 있다. 또한, 제작된 histogram을 파일로 저장하기 위하여 fstream library의 ofstream 함수를 사용하였다. Float array에 저장된 값을 작성하여 csv 형식으로 export한 뒤 엑셀 기능을 사용하여 histogram을 작성할 수 있었다.

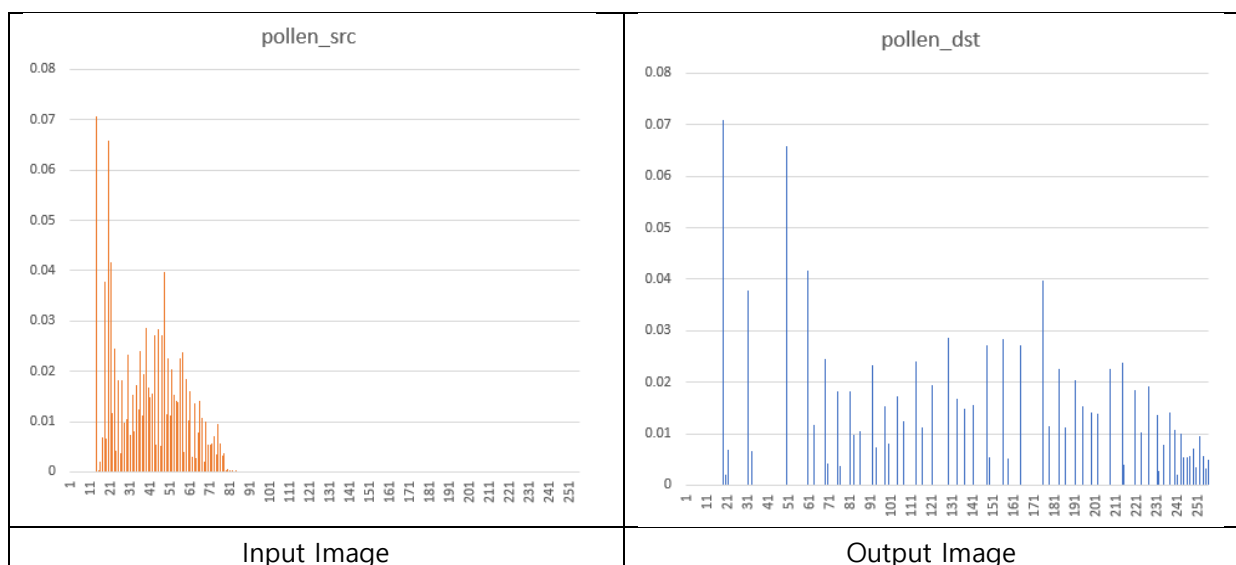
## (2) 구현 2

a) \*.zip에 포함된 "pollen.tif"에 histogram equalization을 적용하시오.



b) 입력영상과 출력영상의 histogram을 출력하시오.

✓ histogram 분포는 excel등을 이용하여 plot함.



입력 영상과 출력 영상의 histogram을 비교해볼 수 있다. 우선, 원본 영상은 어두운 값에 pixel intensity가 집중되어 있는 반면, histogram equalization을 적용한 경우 histogram의 전반적인 형태를 유지한 채로 치우친 pixel intensity 값이 전체적으로 고르게 분포하는 것을 histogram을 통하여 알 수 있다.

c) 아래 6.3)를 참고하여 histEqualization()을 작성하시오.

```
Mat histEqualization(Mat src) {
    Mat dst = Mat::zeros(src.rows, src.cols, CV_8U);
    float* LUT = (float*)calloc(256, sizeof(LUT));
    float cdf = NULL;

    for (int i = 0; i < 256; i++) {
        cdf = cdf + getHist(src)[i];
        LUT[i] = cdf * 255;
    }
    for (int y = 0; y < src.cols; y++) {
        for (int x = 0; x < src.rows; x++) {
            dst.at<uchar>(x, y) = round(LUT[src.at<uchar>(x, y)]);
        }
    }
    std::ofstream histEqualization;
    float* src_hist = getHist(src);
    float* dst_hist = getHist(dst);

    histEqualization.open("histEqualization.csv");
    for (int i = 0; i < 256; i++) {
        histEqualization << i << "," << src_hist[i] << ",";
        histEqualization << i << "," << dst_hist[i] << "\n";
    }
    histEqualization.close();
    return dst;
}
```

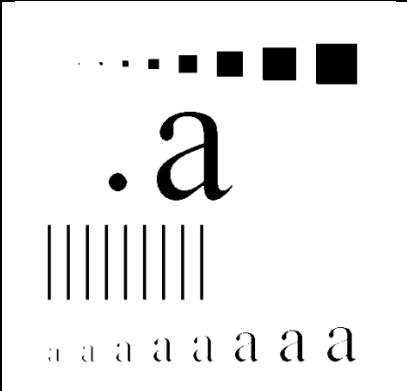




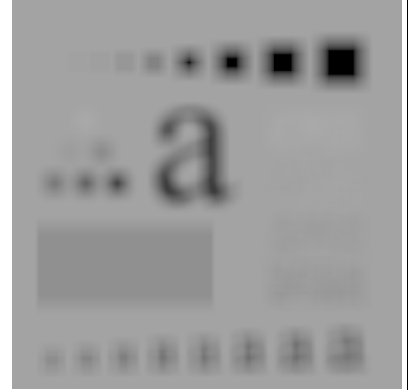
histEqualization() 함수의 경우 image Matrix를 입력으로 받는다. 우선, 각 intensity마다 getHist() 함수에서 출력되는 histogram floating array를 value에 더하여 cdf 값을 얻을 수 있다. 또한,  $s^k = (L - 1) \sum_{j=0}^k p_r(r_j)$  식을 계산하기 위해 look-up table (LUT) array를 사용하여 input image에서의  $r^k$  intensity 값을  $s^k$  intensity 값으로 intensity transform을 할 수 있다. 이렇게 얻은 값을 round하여 equalized histogram을 얻을 수 있다. 이 역시 gammaTransform() 함수에서 사용한 방법으로 file export한 뒤 엑셀 기능을 사용하여 histogram을 작성할 수 있었다.

### (3) 구현 3

a) \*.zip에 포함된 "test\_pattern.tif"에 box kernel를 적용하시오.

✓ Box kernel :  $3 \times 3$ ,  $11 \times 11$ ,  $25 \times 25$

✓ Padding type : Zero padding and mirror padding.

		
3x3 kernel, Zero padding	11x11 kernel, Zero padding	25x25 kernel, Zero padding
		
3x3 kernel, Mirror padding	11x11 kernel, Mirror padding	25x25 kernel, Mirror padding

b) 아래 6.3)를 참고하여 `boxFilter()`, `imFilter()`을 작성하시오.

```

Mat boxFilter(Size size) {
    Mat box = Mat::zeros(size.height, size.width, CV_32F);
    for (int y = 0; y < box.cols; y++) {
        for (int x = 0; x < box.rows; x++) {
            box.at<float>(x, y) = 1 / (float)(size.height * size.width);
        }
    }
    return box;
}

Mat imFilter(Mat src, Mat kernel, string pad_type) {
    Mat padded = Mat::zeros (src.rows + kernel.rows - 1, src.cols + kernel.cols - 1,
CV_32F);
    int colsDiv = (kernel.cols - 1) / 2;
    int rowsDiv = (kernel.rows - 1) / 2;

    if (pad_type == "zero") {
        for (int y = 0; y < src.cols; y++) {
            for (int x = 0; x < src.rows; x++) {
                padded.at<float>(x + rowsDiv, y + colsDiv) =
(float)src.at<uchar>(x, y);
            }
        }
    }
}

```

```

else if (pad_type == "mirror") {
    for (int x = rowsDiv; x < padded.rows - rowsDiv; x++) {
        for (int y = colsDiv; y < padded.cols - colsDiv; y++) {
            if (padded.rows - rowsDiv <= x) {
                padded.at<float>(x, y) = padded.at<float>(2 *
(padded.rows - 1 - rowsDiv - x, y));
            }
            else {
                padded.at<float>(x, y) = (float)src.at<uchar>(abs(x
- rowsDiv), y - colsDiv);
            }
        }
        for (int y = 0; y < colsDiv; y++) {
            padded.at<float>(x, y) = padded.at<float>(x, kernel.cols - 1
- y);
        }
        for (int y = padded.cols - colsDiv; y < padded.cols; y++) {
            padded.at<float>(x, y) = padded.at<float>(x, 2 * (padded.cols
- 1 - colsDiv) - y);
        }
    }
    for (int x = 0; x < rowsDiv; x++) {
        for (int y = colsDiv; y < padded.cols - colsDiv; y++) {
            if (padded.rows - rowsDiv <= x) {
                padded.at<float>(x, y) = padded.at<float>(2 *
(padded.rows - 1 - rowsDiv - x, y));
            }
            else {
                padded.at<float>(x, y) = (float)src.at<uchar>(abs(x
- rowsDiv), y - colsDiv);
            }
        }
        for (int y = 0; y < colsDiv; y++) {
            padded.at<float>(x, y) = padded.at<float>(x, kernel.cols - 1
- y);
        }
        for (int y = padded.cols - colsDiv; y < padded.cols; y++) {
            padded.at<float>(x, y) = padded.at<float>(x, 2 * (padded.cols
- 1 - colsDiv) - y);
        }
    }
    for (int x = padded.rows - rowsDiv; x < padded.rows; x++) {
        for (int y = colsDiv; y < padded.cols - colsDiv; y++) {
            if (padded.rows - rowsDiv <= x) {
                padded.at<float>(x, y) = padded.at<float>(2 *
(padded.rows - 1 - rowsDiv - x, y));
            }
            else {
                padded.at<float>(x, y) = (float)src.at<uchar>(abs(x
- rowsDiv), y - colsDiv);
            }
        }
        for (int y = 0; y < colsDiv; y++) {
            padded.at<float>(x, y) = padded.at<float>(x, kernel.cols - 1
- y);
        }
    }
}

```

```

        for (int y = padded.cols - colsDiv; y < padded.cols; y++) {
            padded.at<float>(x, y) = padded.at<float>(x, 2 * (padded.cols
- 1 - colsDiv) - y);
        }
    }
    Mat dst(src.rows, src.cols, CV_32F);
    for (int x = 0; x < src.rows; x++) {
        for (int y = 0; y < src.cols; y++) {
            float conv = 0;
            for (int x_kernel = 0; x_kernel < kernel.rows; x_kernel++) {
                for (int y_kernel = 0; y_kernel < kernel.cols; y_kernel++) {
                    conv = conv + padded.at<float>(x + x_kernel, y +
y_kernel) * kernel.at<float>(kernel.rows - 1 - x_kernel, kernel.cols - 1 - y_kernel);
                }
            }
            dst.at<float>(x, y) = conv / (kernel.rows * kernel.cols);
        }
    }
    return dst;
}

```

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

boxFilter() 함수의 경우 matrix size를 입력으로 받는다. 위와 같은 간단한 3x3 box filter 예제에서 확인할 수 있듯, box filter를 생성하기 위해서 1-filled matrix인 kernel을 kernel의 height와 width를 곱한 값으로 나눠줄 수 있다.

이를 사용하여 ImFilter 함수를 작성할 수 있다. ImFilter는 원본 영상, kernel (box filter) matrix, padding 방법을 input으로 한다. 우선, padding을 위해서  $\text{rowsDiv} = \frac{m-1}{2}$ ,  $\text{colsDiv} = \frac{n-1}{2}$ 의 상수를 사용하였으며, output matrix로 옮기기 전 input matrix보다 m+1, n+1만큼 큰 size를 가진 padding matrix를 통하여 영상 처리를 할 수 있다.

zero padding의 경우에는 간단히 padding matrix에서 (x+ rowsDiv, y+ colsDiv) 의 좌표를 가진 경우 원본 image의 (x, y) intensity 값을 가지도록 하며, 이외의 좌표를 가진 경우 값을 모두 0으로 처리할 수 있다. (padding matrix가 원래 zero matrix이므로 값을 변화하지 않아도 얻을 수 있다) 이로 인하여 edge에 약간의 검은색 영역이 생긴 것을 확인할 수 있다. (작은 kernel size의 경우 smoothing의 효과로 blur 처리되어 거의 보이지 않지만, 큰 kernel size에서는 영역을 확인할 수 있다)



반면, mirror padding의 경우 padding matrix를 9가지의 경우로 나누어서 볼 수 있다.

	$0 \leq x < \text{rowsDiv}$	$\text{rowsDiv} \leq x < M - \text{rowsDiv}$	$M - \text{rowsDiv} \leq x < M$
$0 \leq y < \text{colsDiv}$	$\text{pad}(x, n - 1 - y)$	$\text{pad}(x, n - 1 - y)$	$\text{pad}(x, n - 1 - y)$
$\text{colsDiv} \leq y < N - \text{colsDiv}$	$\text{pad}(m - 1 - x, y)$	$\text{src}( x - \text{rowsDiv} , y - \text{colsDiv})$	$\text{pad}(2(M - 1 - \text{rowsDiv} - x), y)$
$N - \text{colsDiv} \leq y < N$	$\text{pad}(x, 2(N - 1 - \text{colsDiv} - y))$	$\text{pad}(x, 2(N - 1 - \text{colsDiv} - y))$	$\text{pad}(x, 2(N - 1 - \text{colsDiv} - y))$

이때 src는 원본 image의 좌표, pad는 padding matrix의 좌표를 의미한다. 위와 같은 형태로 중앙 좌표의 경우 그대로 input image value를 사용하며 가장자리의 경우 padding matrix의 값을 mirror reflecting하여 해당하는 값으로 채우게 된다. 이 때문에 zero padding과 다르게 edge에 검은색 영역이 생기지 않으며 원본 border의 형태를 띄는 것을 확인할 수 있다.

이를 통해서 output 영상을 얻기 위하여 convolution process를 거친다. 2D convolution의 경우 input image의 pixel에서, 각 pixel마다 kernel size 만큼의 convolution을 누적하여 반환하고, 이를 kernel의 height와 width를 곱한 값으로 나눠주어 최종 convolution된 값을 구한다. 이를 통하여 kernel size에 따라 blur된 output image를 획득할 수 있다.

#### 4. 검토사항

- 1) 구현 1, 2, 3에 대한 결과를 분석하여 보고서에 첨부하시오.
- 2) Spatial domain에서 filtering된 결과가 원본 영상과 같은 크기를 갖기 위해서는 원본 영상을 padding해야 한다. 이때 사용할 수 있는 padding의 종류에 대하여 적어도 3가지 이상 나열하고 각각에 대해 설명하시오.



##### (1) Zero Padding

원본 영상보다 size가 큰 부분을 0으로 padding하는 방법이다. Greyscale image에서 0은 검은색이므로 이 padding 기법을 사용할 경우 filtered result의 가장자리에 dark borders가 생기며 kernel size가 클수록 그 border size 역시 커진다.

### (2) Mirror (symmetric) padding

원본 영상보다 size가 큰 부분을 원본 영상의 가장자리로부터 대칭되도록 intensity를 가져와 padding하는 방법이며 kernel size가 클수록 이러한 가장자리로부터 가져오는 영상의 size가 커진다. 예를 들어 32x32 영상에 5\*5 box kernel을 적용하여 mirror padding을 할 경우 입력  $g(31, y)$ 는 출력  $f(28, y)$ 로부터,  $g(30, y)$ 는  $f(29, y)$ 로부터 등의 형태로 intensity를 반환할 수 있다.

### (3) Replicate padding

원본 영상보다 size가 큰 부분을 원본 영상과 가장 가까운 부분을 복제하여 padding하는 방법이다. 위와 같은 예로 32x32 영상에 5\*5 box kernel을 적용하여 replicate padding을 할 경우  $f(31, y)$ ,  $f(30, y)$  모두  $g(29, y)$ 로부터 intensity를 반환하게 된다.

- 3) Filtering의 수행시간을 빠르게 하기 위한 방법 중 하나인 **separable filter**를 이용한 convolution에 대해서 설명하시오.

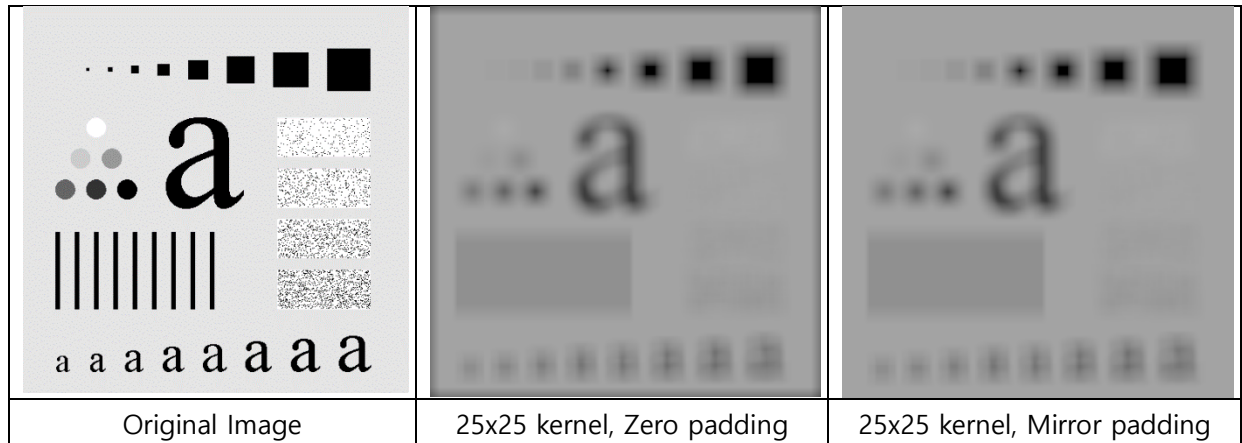
$$g(x, y) = w(x, y) * f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x-s, y-t)$$

2D function에 대한 convolution은 다음과 같으며, 이 때 2D function  $g(x, y)$ 가 1D function 2개의 product  $g_1(x)g_2(y)$ 의 형태로 표현 가능할 경우 Separable 2D function이라고 한다. 이를 바탕으로, convolution의 commutative 성질을 이용하여 다음과 같이 작성할 수 있다.

$$\begin{aligned} g(x, y) = w(x, y) * f(x, y) &= \sum_{s=-a}^a g_2(y) \left[ \sum_{t=-b}^b g_1(x) \times f(x-s, y-t) \right] \\ &= g_2(y) * [g_1(x) * f(x, y)] \end{aligned}$$

일반적인 경우, image size가  $M \times N$ , kernel size  $m \times n$ 일 때 매 픽셀마다  $m \times n$ 번의 곱셈과 덧셈이 필요하여 총  $(M \times N) \times (m \times n)$  번의 연산량이 필요하지만, 이러한 kernel을 2개의 1차원 filter로 구분하게 된다면 한 픽셀당  $m+n$ 번의 곱셈과 덧셈만이 필요하며 총  $(M \times N) \times (m+n)$ 으로 연산횟수를 줄일 수 있다.

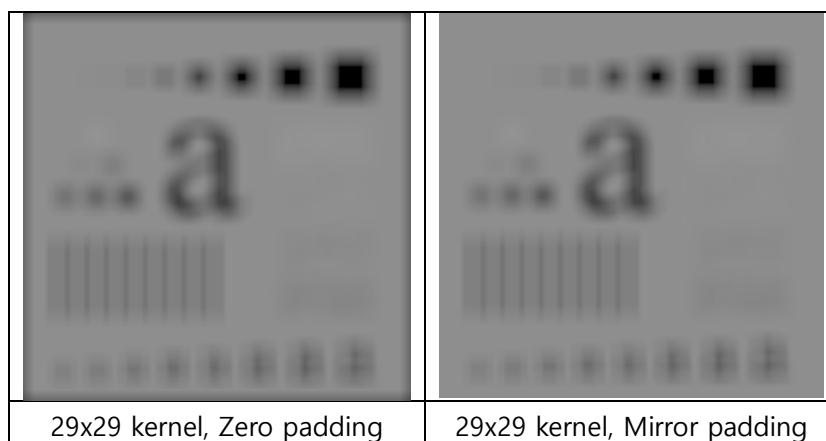
- 4) 구현 3에서는 box kernel의 size에 따라 “test\_pattern.tif”가 수직선이 구분되지 않는 경우가 발생한다. 이러한 결과가 나타나는 이유를 LPF와 연관지어 설명하시오.



test\_pattern.tif 이미지의 경우, kernel size가 25x25일 경우 결과로부터 수직선이 구분되지 않는 경우가 발생한다. 이는 box filter kernel이 기본적인 separable lowpass spatial filter이기 때문이다. Box kernel의 경우 normalization을 통하여 bias가 없이 같은 average value를 가질 수 있도록 하며, rank가 1이므로 separable한 성질 역시 가진다.

이 때, 수직선과 같은 edges and sharp intensity transitions와 같은 image의 속성은 high frequency로 특징지어질 수 있기 때문에, box filter는 이와 같은 high frequency는 통과시키지 않는 반면 smooth regions인 low frequency는 pass하는 성질을 보인다. 이 경우, input image의 intensity 변화가 줄어드는 것으로 blur된 output을 보일 수 있다.

Windows 내장 image viewer를 사용하여 수직선의 폭은 100px\*6px, 수직선 사이의 간격의 폭은 100px\*19px임을 알 수 있었다. 이는 25x25의 kernel size에서 수직선을 완전히 구분하지 못하는 이유를 설명할 수 있다. Mask의 수평 stride마다 수직선의 폭과 수직선 사이의 간격만큼의 pixel (6+19=25)의 intensity average가 항상 동일하므로, image를 kernel에 통과시켰을 때 해당 부분의 output의 intensity가 동일하다.



25x25보다 더욱 큰 kernel size로 image를 process한 결과 수직선의 detail을 약간이나마 확인할 수 있었다. 이를 토대로 3x3, 11x11 등의 kernel size에서는 수직선을 구분할 수 있었지만 25x25의 kernel size에서 수직선을 구분하지 못하는 이유 역시 알 수 있다.

## 참조문헌

DIP\_Homework2.pdf, 서강대학교 MMI Lab

OpenCV Documentation (<https://docs.opencv.org/>), Intel Corporation

Gonzalez, Rafael C. Woods, Richard E. - Digital image processing, 4<sup>th</sup> Edition, Pearson, 2018.