디지털 영상처리 설계(hw3)

12211827 임지혜 2분반

example 1)

9x9 Gaussian filter를 구현하고 결과를 확인할 것, 9x9 Gaussian filter를 적용했을 때 히스토그램이 어떻게 변하는지 확인할 것

- source code

(1) myKernelConv9x9 와 myGaussianFilter 구현

```
//9x9 마스크에 대한 convolution
int myKernerConv9x9(uchar* arr, double kernel[][9], int x, int y, int width, int height) // uchar 그레이 이미지에 접근
{
    int sum = 0;
    int sumKernel = 0;

    //특정 화소의 모든 이웃화소에 대해 계산하도록 반복문 구성
    //9x9 kernel을 생성
    for (int j = -4; j <= 4; j++) { // 9x9 니까 행렬 각각이 9개씩 -4~4
        for (int i = -4; i <= 4; i++) {
            if ((y + j) >= 0 && (y + j) < height && (x + i) >= 0 && (x + i) < width) {
                //영상 가장자리에서 영상 밖의 화소를 읽지 않도록 하는 조건문
                sum += arr[(y + j) * width + (x + i)] * kernel[i + 4][j + 4];
                sumKernel += kernel[i + 4][j + 4];
            }
        }
    }

    if (sumKernel != 0) { return sum / sumKernel; } //합이 1로 정규화되도록 하여, 영상의 밝기 변화를 방지******
    else return sum;
}
```

```
//Gaussian filter
Mat myGaussianFilter(Mat srcImg)
{
    int width = srcImg.cols;
    int height = srcImg.rows;
    double kernel[9][9];    //9x9 형태의 Gaussian 마스크 배열 // 9x9 형태의 마스크 배열을 모두 넣을 수 없으니 함수로 따로 작성
    double sigma = 2.0; //9x9 Gaussian mask의 크기 -> sigma = 2 // sigma가 커질 수록 블러가 많아지기에 1은 티가 별로 없어 2로 설정
    double r;
    double s = 2 * sigma * sigma;
    const int PI = 3.14;

    // 가우시안 필터에 대한 공식을 아래식으로 표현
    for (int j = -4; j <= 4; j++) {
        for (int i = -4; i <= 4; i++) {
            r = sqrt(i * i + j * j); // 제곱은 sqrt이용
            kernel[i + 4][j + 4] = (exp(-(r * r) / s)) / (PI * s);
        }
    }

    Mat dstImg(srcImg.size(), CV_8UC1);
    //CV_8UC1 : 2^8bits(0~255), one(gray) channel
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++)
            dstData[y * width + x] = myKernerConv9x9(srcData, kernel, x, y, width, height);
        //앞서 구현한 convolution에 마스크 배열을 입력해 사용
    }
    return dstImg;
}
```

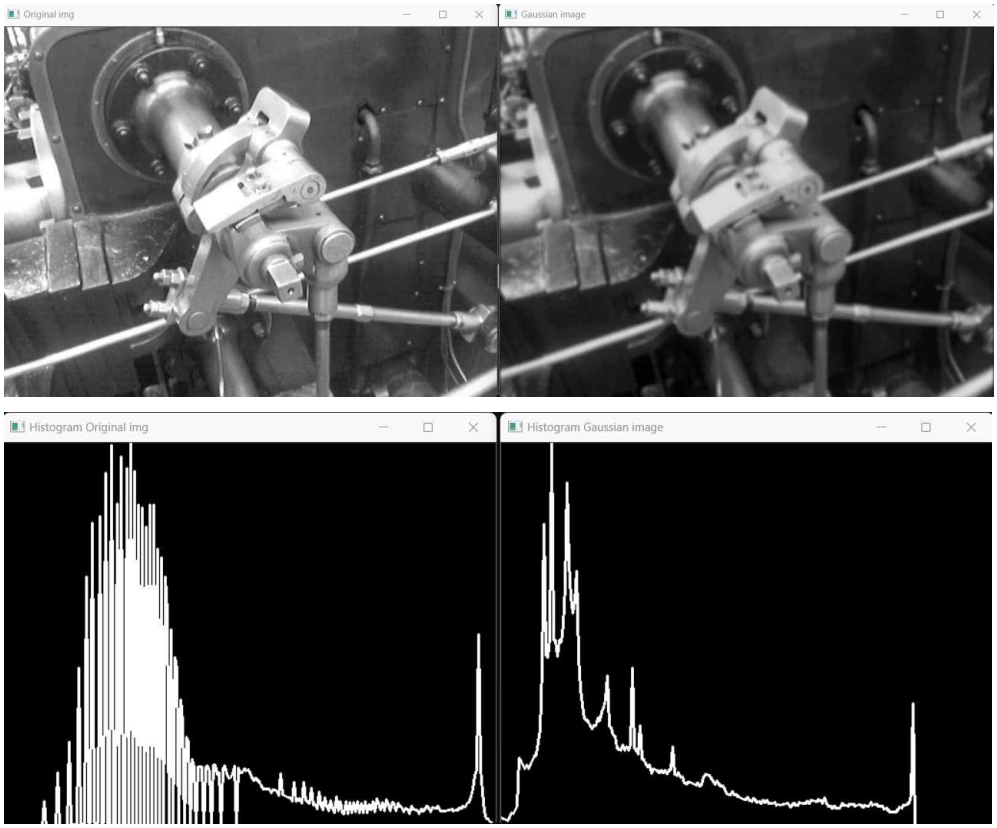
comment)

9x9 마스크에 대한 convoulution를 실행하기 위해서는 총 9x9의 metrix가 적용되는 것을 이해하면 중앙의 양 옆에 대칭적으로 4개의 이웃 화소가 존재하는 것을 알 수 있다. 따라서

for문을 작성할 때 -4~4까지의 범위를 지정해주고 kernel의 index 또한 [i+4][j+4]로 해주었다.
3x3 가우시안 필터는 총 9개의 값만 넣어주면 되기 때문에 강의노트의 코드와 같이 공식에 각 pixel에 맞춰 값을 대입해줄 수 있지만 9x9 가우시안 필터는 총 81개의 값을 넣어주어야 하기 때문에 코드가 복잡하는 어려움이 있다.

따라서 가우시안 필터에 대한 공식을 강의노트에 있는 $G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ 를 이용하여 작성하였다.

- 실행 결과



original 이미지의 histogram과 gaussian filter를 적용한 histogram을 비교해보았을 때, 적용한 후의 분포가 더욱 넓게 퍼진 것을 확인할 수 있다. 이를 통해 고주파수를 낮추어 노이즈를 줄여주었다. 코드를 작성하는 과정에서 sigma의 값을 강의 노트와 다르게 2로 설정해주었는데 1은 티가 별로 나지 않아 실험 결과를 판별하기에 어려움이 있었다.
그렇다고 sigma의 값을 높이면 블러가 많아지는 현상이 있어 적당한 값을 넣어줄 수 있었다.

example 2)

1) 영상에 salt and pepper noise를 주고, 구현한 9x9 Gaussian filter를 적용해라.

- source code

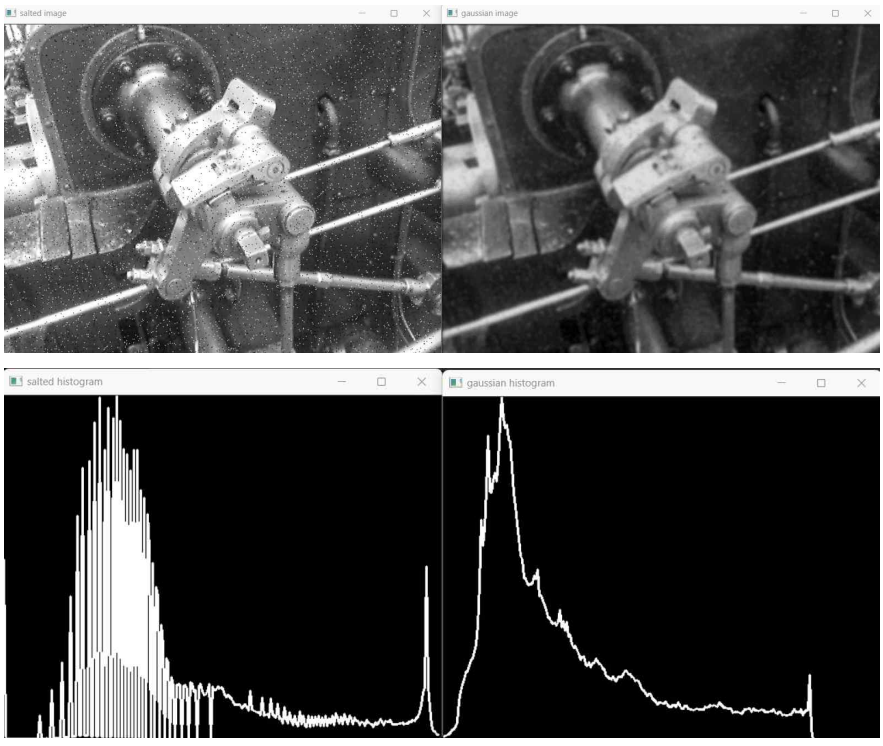
```cpp
//Generate Salts and Pepper noise
Mat salt_and_pepper_noise(Mat img) {
    srand(time(NULL));
    int black, white;
    int height = img.rows;
    int width = img.cols;
    uchar Black = 0;
    uchar White = 255;
    Mat dst_img = img;

    cout << "Enter how many black dots you wanna mask: "; cin >> black;
    cout << "Enter how many white dots you wanna mask: "; cin >> white;

    for (int i = 0; i < black; i++) {
        img.at<uchar>(rand() % height, rand() % width) = Black;
    }

    for (int i = 0; i < white; i++) {
        img.at<uchar>(rand() % height, rand() % width) = White;
    }

    return dst_img;
}
```

comment)

무작위로 salt와 pepper를 이미지에 뿌리기 위해서 random하게 time에 따라서 뿌려주었다. 이를 위한 함수로 srand를 사용하였고 salt의 픽셀값을 225로 pepper를 0으로 설정하였다. salt의 값과 pepper의 값을 얼마나 입력할 것인지 입력받고 조건에 맞게 이미지에 뿌려주었다.

- 실행 결과

입력 값을 각각 5000, 5000으로 입력해주고 실행하였을 때,



salt와 pepper의 점을 찍고 가우시안 필터를 적용한 사진을 볼 때, 사진 상으로도 histogram 상으로도 노이즈가 거의 보이지 않는 것을 확인하였다.

example 3)

1) 45도와 135도의 대각 edge를 검출하는 Sobel filter를 구현하고 결과를 확인하라.

- source code

```
//45도와 135도의 대각 edge를 검출하는 Sobel filter
Mat mySobelFilter(Mat srcImg) {
    int kernelX[3][3] = { 0, 1, 2,
                         -1, 0, 1,
                         -2, -1, 0 };    //45도에 대한 Sobel kernel 마스크(가로방향)
    int kernelY[3][3] = { -2, -1, 0,
                           -1, 0, 1,
                           0, 1, 2 };//135도에 대한 Sobel kernel 마스크(세로방향)

    //마스크 합이 0이 되므로 1로 정규화하는 과정은 필요 X
    Mat dstImg(srcImg.size(), CV_8UC1);
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;
    int width = srcImg.cols;
    int height = srcImg.rows;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++)
            dstData[y * width + x] = (abs(myKernerConv3x3(srcData, kernelX, x, y, width, height))
                + abs(myKernerConv3x3(srcData, kernelY, x, y, width, height))) / 2;
        //두 edge 결과의 절대값의 합 형태로 최종결과 도출
    }

    return dstImg;
}
```

commet)

45도와 135도에 대각 edge를 검출하는 sobel filter는 강의노트를 보면 kernel 배열을 3x3으로 입력한다. 나머지의 코드 또한 강의노트를 참고하였고 마스크의 합은 0이므로 1로 정규화하는 과정을 따로 작성하지 않았다. 다음 출력 결과는 고주파수의 값이 출력된 것을 확인할 수 있다.

- 실행 결과

example 4)

1) 컬러영상에 대한 Gaussian pyramid를 구축하고 결과를 확인하라

- source code

```
//Gaussian filter about color image
Mat myGaussianFilter_color(Mat srcImg)
{
    int width = srcImg.cols;
    int height = srcImg.rows;
    double sigma = 2.0; //9x9 Gaussian mask의 크기 -> sigma = 2
    double r;
    double s = 2 * sigma * sigma;
    const int PI = 3.14;
    double kernel[9][9];    //9x9 형태의 Gaussian 마스크 배열

    for (int j = -4; j <= 4; j++) {
        for (int i = -4; i <= 4; i++) {
            r = sqrt(i * i + j * j);
            kernel[i + 4][j + 4] = (exp(-(r * r) / s)) / (PI * s);
        }
    }

    Mat dstImg(srcImg.size(), CV_8UC3);
    //CV_8UC1 : 2^8bits(0~255), one(gray) channel
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;

    Mat RGB[3];
    split(srcImg, RGB); //split()함수 : 이미지를 채널별(R,G,B)로 분리
    uchar* BlueData = RGB[0].data; // 흑백 이미지이기에 uchar을 이용하여 사용
    uchar* GreenData = RGB[1].data;
    uchar* RedData = RGB[2].data;
```

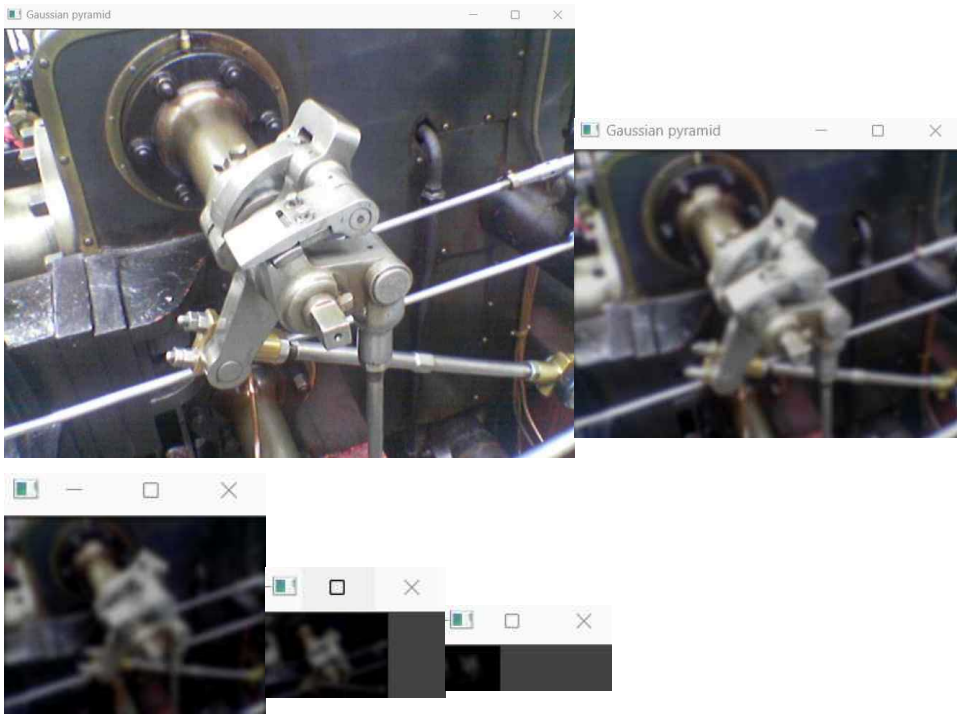
```
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = (y * width + x) * 3;
            dstData[index + 0] = myKernerConv9x9(BlueData, kernel, x, y, width, height);
            dstData[index + 1] = myKernerConv9x9(GreenData, kernel, x, y, width, height);
            dstData[index + 2] = myKernerConv9x9(RedData, kernel, x, y, width, height);
            //앞서 구현한 convolution에 마스크 배열을 입력해 사용
        }
    }
    return dstImg;
}
```

comment) 컬러 이미지를 따로 함수를 구현하여 진행하였는데 CV_8UC3으로 컬러 채널의 영상을 dstImg로 선언해주었고, RGB 값을 저장할 인덱스를 생성하여 원본 이미지는 흑백이기 때문에 uchar을 이용하여 data를 만들어 주었다. 앞서 구현한 convolution에 마스크 배열을 활용하여 결과가 저장될 수 있도록 해주었다.

```
Mat mySampling(Mat srcImg) {
    int width = srcImg.cols / 2;
    int height = srcImg.rows / 2;
    //가로 세로가 입력 영상의 절반인 영상을 먼저 생성

    Mat dstImg(height, width, CV_8UC3);
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            dstImg.at<Vec3b>(y, x) = srcImg.at<Vec3b>(y * 2, x * 2); // 2배 간격으로 인덱싱 해 큰 영상을 작은 영상에 대입, 여기선 컬러영상을 받으니까 vec3b로
        }
    }

    return dstImg;
}
```

down sampling을 수행하는 위 함수는 RGB에 대해 이미지를 작은 영상에 대입해주었다. 각 픽셀에 접근하여 2배 간격으로 인덱싱하고 큰 영상을 작은 영상에 대입해주었다.

-실행 결과



출력 결과 상 이미지가 점차적으로 줄어들었고 gaussian filter가 적용되어 블러가 점점 심해지는 것을 확인하였다.

example 5)

1) 컬러영상에 대한 Laplacian pyramid를 구축하고 복원을 수행한 결과를 확인해라.

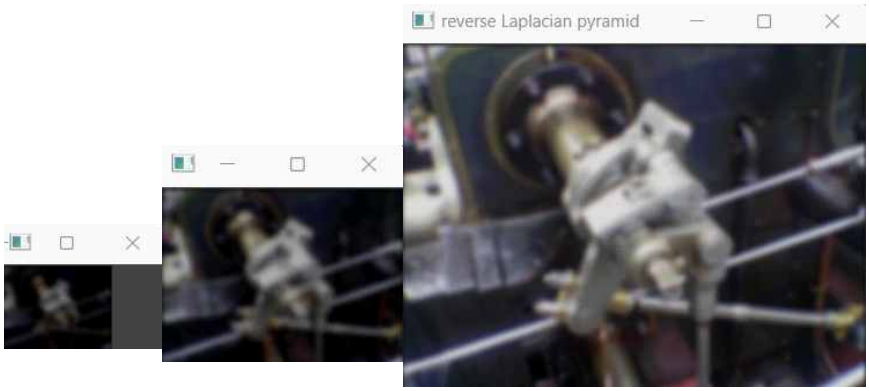
- source code

```
//Laplacian pyramid
//높은 해상도의 영상과 작아진 영상 간의 차영상을 저장하는 방식
// up sampling하고 차 영상을 더하여 높은 해상도의 영상 복원
vector<Mat> myLaplacianPyramid(Mat srcImg) {
    vector<Mat> Vec;

    for (int i = 0; i < 4; i++) {
        if (i != 3) {
            Mat highImg = srcImg;   //수행하기 이전 영상을 백업

            srcImg = mySampling(srcImg);
            srcImg = myGaussianFilter_color(srcImg);

            Mat lowImg = srcImg;
            resize(lowImg, lowImg, highImg.size());
            //작아진 영상을 백업한 영상의 크기로 확대
            Vec.push_back(highImg - lowImg + 128);
            //차 영상을 컨테이너에 삽입
            //128 더해준 것은 차 영상에서 오버플로우를 방지하기 위함
        }
        else
            Vec.push_back(srcImg);
    }

    return Vec;
}
```

comment)

laplacian pyramid는 높은 해상도의 영상과 작아진 영상 간의 차영상을 저장하는 방식으로 up sampling하고 차 영상을 더하여 높은 해상도의 영상을 복원하는 것이다.
따라서 컬러영상에 대한 함수를 구현하여 gray scale에서 overflow를 방지하기 위해서 128을 더해 변경을 해주었다.
또한 복원을 위해서 overflow를 방지하기 위한 128를 더한 값을 다시 빼주어 main함수를 구성하였다.

-실행 결과



출력 결과를 확인해보면 크기가 점차적으로 커지는 것을 확인할 수 있었지만 복원하는 과정에서 이미지가 훼손된 것을 확인할 수 있었다.

각 example에 대한 main 실행 함수)

main함수로 작성하여 1~5까지 ex를 수행할 때, 한 가지 main 함수만 두고 나머지는 다 주석으로 처리하며 실행하였다.

```
//9x9 Gaussian filter를 구현하고 결과를 확인할 것
//9x9 Gaussian filter를 적용했을 때 히스토그램이 어떻게 변하는지 확인할 것
//1번
int main() {
    Mat img = imread("gear.jpg", 0);
    Mat GaussianImg = myGaussianFilter(img);

    imshow("Original img", img);
    imshow("Gaussian image", GaussianImg);

    Mat hist_img = GetHistogram(img);
    Mat hist_GaussianImg = GetHistogram(GaussianImg);
    imshow("Histogram Original img", hist_img);
    imshow("Histogram Gaussian image", hist_GaussianImg);

    waitKey(0);
    destroyAllWindows();
}
```

```
//2번
//영상에 Salt and peppter noise를 주고 구현한 9x9 Gaussian filter를 적용해볼 것
int main() {
    Mat src_img = imread("gear.jpg", 0);
    Mat dst_img = salt_and_pepper_noise(src_img);

    imshow("salted image", dst_img);
    imshow("salted histogram", GetHistogram(dst_img));
    dst_img = myGaussianFilter(dst_img);
    imshow("gaussian image", dst_img);
    imshow("gaussian histogram", GetHistogram(dst_img));
    waitKey(0);
    destroyAllWindows();
}


//3번
//45도와 135도의 대각 edge를 검출하는 Sobel filter를 구현하고 결과를 확인할 것
int main() {
    Mat img = imread("gear.jpg", 0);
    Mat SobelImg = mySobelFilter(img);

    imshow("Sobel image", SobelImg);
    waitKey(0);
    destroyAllWindows();

}
```

```
//4번
//컬러영상에 대한 Gaussain pyramid를 구축하고 결과를 확인할 것
int main() {
    Mat img = imread("gear.jpg", 1);
    vector<Mat> GaussianPyramid = myGaussianPyramid(img);

    for (int i = 0; i < GaussianPyramid.size(); i++)
    {
        imshow("Gaussian pyramid", GaussianPyramid[i]);
        waitKey(0);
        destroyAllWindows();
    }
}
```

```cpp
//5번
int main() {
    //컬러영상에 대한 Laplacian pyramid를 구축하고
    //복원을 수행한 결과를 확인할 것
    Mat src_img = imread("gear.jpg", 1);
    Mat dst_img;
    vector<Mat> VecLap = myLaplacianPyramid(src_img);

    reverse(VecLap.begin(), VecLap.end()); //작은 영상부터 처리하기 위해 vector의 순서를 반대로 해줌

    for (int i = 0; i < VecLap.size(); i++) {
        //vector의 크기만큼 반복
        if (i == 0) {
            dst_img = VecLap[i];    //가장 작은 영상
            //가장 작은 영상 != 차 영상이면 바로 불러옴
        }
        else {
            resize(dst_img, dst_img, VecLap[i].size()); //작은 영상을 확대(up-sampling)
            dst_img = dst_img + VecLap[i] - 128;
            //차 영상을 다시 더하여 큰 영상으로 복원
            //overflow 방지용으로 더했던 128을 다시 빼줌
        }

        string fname = "lap_pyr" + to_string(i) + ".png";
        imwrite(fname, dst_img);
        imshow("reverse Laplacian pyramid", dst_img);
        waitKey(0);
        destroyAllWindows();
    }
}
```