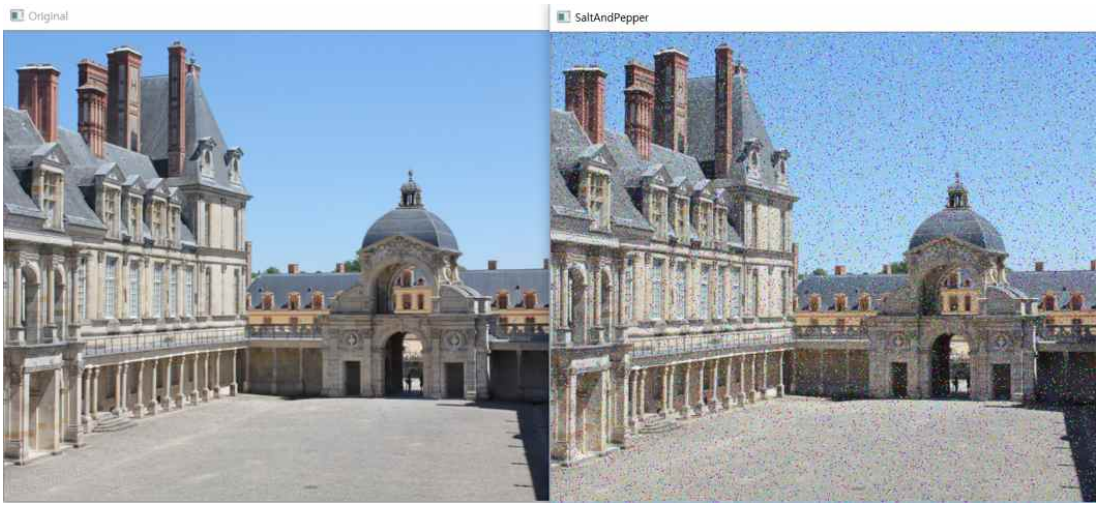
OpenCV 2차시

ㅇ Salt-and-Pepper 이미지 만들기

이미지에 일부러 잡음을 넣어, 마치 '소금'과 '후추가루'가 사진 위에 흩뿌려진 듯한 효과를 내보고자 한다. 아래 왼쪽은 noise가 없는 정상 사진인데, 임의 값을 넣어서 노이즈를 발생시킨다.




```
31  int main(int argc, char** argv)
32  {
33
34      if (argc != 2)
35      {
36          cout << " Provide image name to read" << endl;
37          return -1;
38      }
39
40      Mat inputImg;
41      Mat spImg;
42
43      inputImg = imread(argv[1], IMREAD_UNCHANGED);// 이미지 자체 그대로 받음
44      resize(inputImg, inputImg, Size(), 0.3, 0.3, CV_INTER_AREA);// 인풋이미지 사이즈 조정
45                                                                  // 줄이는거 옵션  세로,가로 0.3배씩
46      spImg = inputImg.clone();
47      addSaltAndPepperNoise(spImg, 0.05);// 5%(=0.05) 픽셀을 노이즈로 , 레벨커질수록 노이즈 커짐
48
49      imshow("Original", inputImg);
50      imshow("SaltAndPepper", spImg);
51
52      waitKey(0);
53      return 0;
54  }
```

Mat 클래스는 그림을 담기 위해 사용되는 클래스

* clone?
똑같은 사진을 복사 한다.

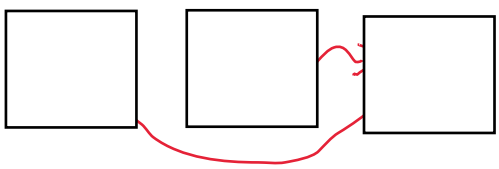
```
/*
    img의 전체 픽셀중 noise_ratio 퍼센트 만큼의
    픽셀을 salt & pepper noise로 바꾼다.
*/
void addSaltAndPepperNoise(Mat& img, double noise_ratio)// 레퍼런스로 받아온다. 레퍼런스로 안 하면? call by value 메모리 커진다
{
    int rows = img.rows;// 이미지 행
    int cols = img.cols;// 이미지 열
    int ch = img.channels(); // channel : 픽셀 (B, G, R)
    int num_of_noise_pixels = (int)((double)(rows * cols * ch)*noise_ratio);

    for (int i = 0; i < num_of_noise_pixels; i++)
    {
        int r = rand() % rows;  // noise로 바꿀 행을 임의로 선택
        int c = rand() % cols;  // noise로 바꿀 열을 임의로 선택
        int _ch = rand() % ch;  // noise로 바꿀 채널의 임의로 선택

        // img.ptr<uchar>(r)은 r번째 행의 첫번째 픽셀, 첫번째 채널에 대한 주소값을 반환한다.
        uchar* pixel = img.ptr<uchar>(r) +(c*ch) + _ch; // noise로 바꿀 정확한 위치를 계산
                                                        // 어느 행,열에가서 어떤 채널을 바꿀껀지 위치
        *pixel = (rand() % 2 == 1) ? 255 : 0; // black(0) 혹은 white(255)로 교체
    }
}
```

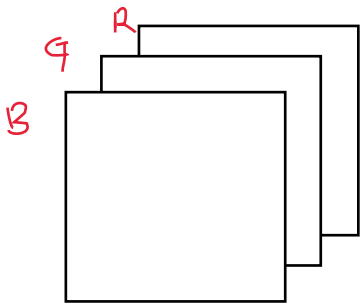
*ptr
이미지 안에 있는 포인터
*uchar
unsigned char

실험

다운로드 받은 파일에서 ‘saltAndPepper.jpg’를 이용하여,

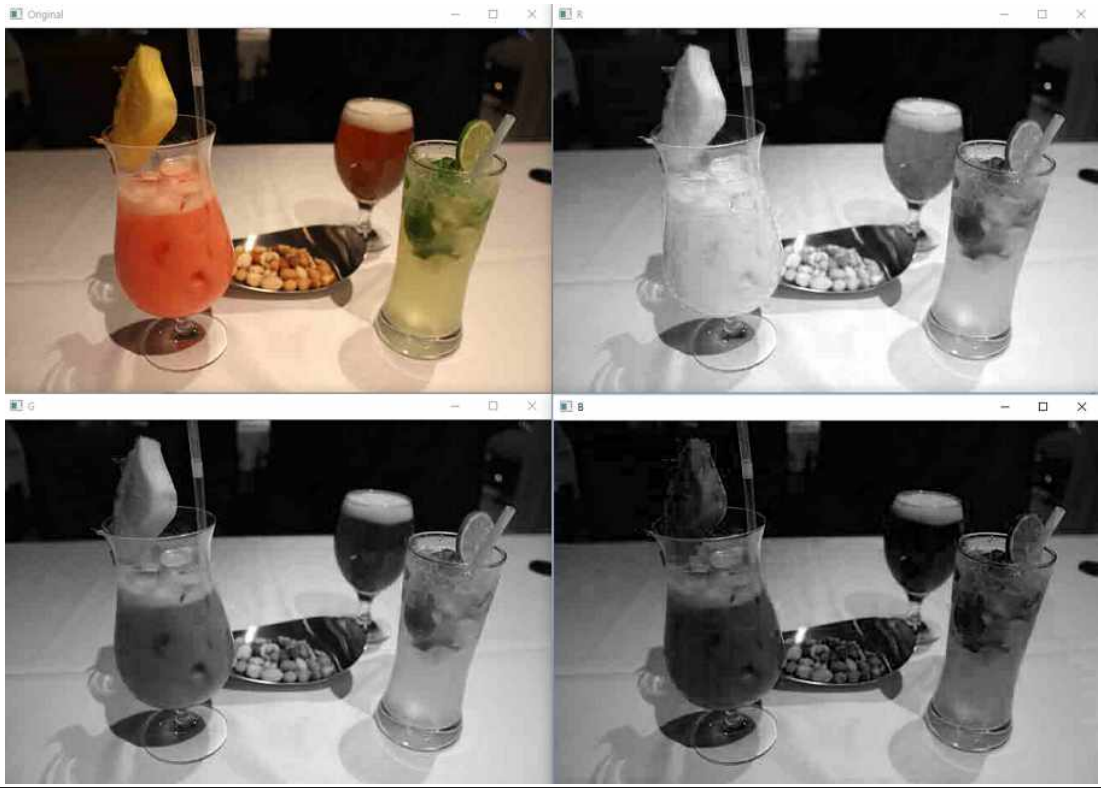
1. line 47의 두 번째 argument를 0.05 --> 0.35로 변경한 후 결과를 관찰해 보자
2. line 46을 ‘spImg = inputImg;’와 같이 수정한 후 결과를 관찰해 보자
   // shallow copy가 되서 똑같은 사진이 된다.





o 이미지를 채널별로 구분해서 보기

컬러이미지의 한 픽셀은 B-G-R 값을 가지고 있다. 이들을 각각 채널이라고 하는데, 채널별로 이미지를 추출해서 볼 수 있다. 아래 사진에서는 잔에 담긴 음료가 '빨간' 색인데, R-channel을 보여주는 이미지에서 음료잔이 '흰색 - 높은 값' 으로 표시된 것을 볼 수 있다.



```cpp
int main(int argc, char** argv)
{


    Mat inputImg;
    Mat hlsImg;

    inputImg = imread("C:/OpenCV/opencv_projects/YUV_example/dog.jpg", IMREAD_UNCHANGED);

    if (inputImg.empty() == true)
    {
        cout << "Unable to open the image" << endl;
        return -1;
    }

    vector<Mat> bgr_images(3);
    split(inputImg, bgr_images); // original 이미지를 r,b,g 채널을 나눠서 bgr_image에 넣어 준다.

    imshow("Original", inputImg);
    imshow("B", bgr_images[0]);
    imshow("G", bgr_images[1]);
    imshow("R", bgr_images[2]);

    waitKey(0);
    return 0;
}
```

*vector
array가 된다.

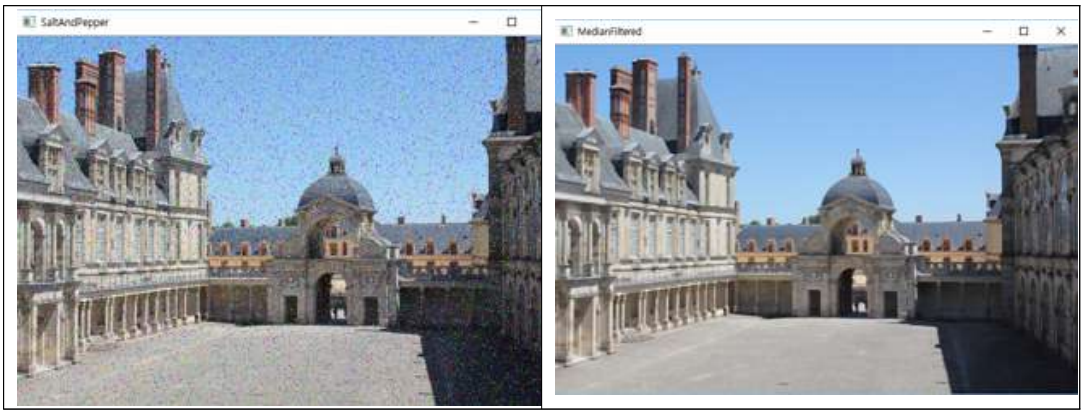
* split 해서 영상처리하면?
연산량이 줄어든다

o 실험

glasses.jpg를 이용해서 결과를 확인해 보자

ㅇ 사진의 노이즈를 없애 보자

Salt-and-Pepper noise가 들어간 사진에서 노이즈를 없애보자.




```
31  int main(int argc, char** argv)
32  {
33
34      Mat inputImg;
35      Mat spImg;
36
37      inputImg = imread(argv[1], IMREAD_UNCHANGED);
38      resize(inputImg, inputImg, Size(), 0.25, 0.25, CV_INTER_AREA);
39
40      spImg = inputImg.clone();
41      addSaltAndPepperNoise(spImg, 0.05);
42
43      Mat localAvgImg;
44      blur(spImg, localAvgImg, Size(5, 5));
45
46      Mat gaussianSmoothedImg;
47      GaussianBlur(spImg, gaussianSmoothedImg, Size(5, 5), 1.5);
48
49      Mat medianFilteredImg;
50      medianBlur(spImg, medianFilteredImg, 3);
51
52      imshow("Original", inputImg);
53      imshow("SaltAndPepper", spImg);
54      imshow("LocalAveraging", localAvgImg);
55      imshow("GaussianSmoothing", gaussianSmoothedImg);
56      imshow("MedianFiltered", medianFilteredImg);
57
58      waitKey(0);
59      return 0;
60  }
```

// 3?
내주변3*3 몇 개 중 가운데 값?

1. localAvg
   픽셀 5 x 5 뽑아서 애들의 평균 값으로 Img가 흐려진다 = blur

2. gaussianSmoothed
   가운데 픽셀에 가중치를 주는것
   가운데에 있는 애가 가장 원본이랑 가까운 색이라고 가정한다.
   = GaussianBlur
   왜? 원래색에 더 가깝게 유지하면서 가중치가 된다.

3. medianFiltered
   픽셀들 중 가운데 값을 대치

blur는 평균을 내는 방식,
GaussianBlur는 가중평균을 내는 방식,
MedianBlur는 주변 피셀들의 중간값으로 대치

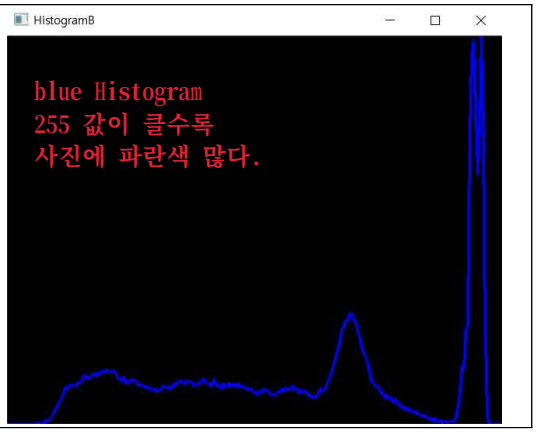
블러링은 사진을 멀리서 본게 윤곽을 정확하게 보기위해서

saltAndPepper.jpg 이미지를 가지고 실험
line 44의 Size(5,5)를 Size(10,10)으로 할 경우,
line 47의 Size(5,5)를 Size(10,10), 1.5 (분산)를 더 큰 숫자로 변경
line 50의 마지막 인수를 3 이외의 더 큰 숫자로 입력

ㅇ 히스토그램 보기

사진의 B-G-R 채널별 히스토그램을 살펴보면, 사진의 색상적 특성을 알 수 있다.

아래 왼쪽사진은 전체적으로 '푸른' 색이기 때문에, B-channel의 histogram을 보면 '강한 푸른' 색이 많음을 알 수 있다. 각 값의 분포도를 뜻한다.



픽셀 0 255

```
9   int main(int argc, char** argv)
10  {
11      Mat inputImg;
12
13      inputImg = imread(argv[1], CV_LOAD_IMAGE_COLOR); // 이미지 읽기
14      resize(inputImg, inputImg, Size(), SCALE, SCALE, CV_INTER_AREA); // 사진크기 줄이기
15
16      MatND histogramB, histogramG, histogramR;// 빈도수를 저장할 배열
17      const int channel_numbersB[] = { 0 };  // Blue
18      const int channel_numbersG[] = { 1 };  // Green
19      const int channel_numbersR[] = { 2 };  // Red
20      float channel_range[] = { 0.0, 255.0 };
21      const float* channel_ranges = channel_range;
22      int number_bins = 255;
23
24      // R, G, B별로 각각 히스토그램을 계산한다.
25      calcHist(&inputImg, 1, channel_numbersB, Mat(), histogramB, 1, &number_bins, &channel_ranges);
26      calcHist(&inputImg, 1, channel_numbersG, Mat(), histogramG, 1, &number_bins, &channel_ranges);
27      calcHist(&inputImg, 1, channel_numbersR, Mat(), histogramR, 1, &number_bins, &channel_ranges);
```

```
29      // Plot the histogram
30      int hist_w = 512; int hist_h = 400;
31      int bin_w = cvRound((double)hist_w / number_bins);
32
33      Mat histImageB(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0)); // B channel
34      normalize(histogramB, histogramB, 0, histImageB.rows, NORM_MINMAX, -1, Mat());
35
36      Mat histImageG(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));
37      normalize(histogramG, histogramG, 0, histImageG.rows, NORM_MINMAX, -1, Mat());
38
39      Mat histImageR(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));
40      normalize(histogramR, histogramR, 0, histImageR.rows, NORM_MINMAX, -1, Mat());
41
42      for (int i = 1; i < number_bins; i++) // 선으로 빈도수 그린다.
43      {
44
45          line(histImageB, Point(bin_w*(i - 1), hist_h - cvRound(histogramB.at<float>(i - 1))),
46              Point(bin_w*(i), hist_h - cvRound(histogramB.at<float>(i))),
47              Scalar(255, 0, 0), 2, 8, 0);
48          line(histImageG, Point(bin_w*(i - 1), hist_h - cvRound(histogramG.at<float>(i - 1))),
49              Point(bin_w*(i), hist_h - cvRound(histogramG.at<float>(i))),
50              Scalar(0, 255, 0), 2, 8, 0);
51
52          line(histImageR, Point(bin_w*(i - 1), hist_h - cvRound(histogramR.at<float>(i - 1))),
53              Point(bin_w*(i), hist_h - cvRound(histogramR.at<float>(i))),
54              Scalar(0, 0, 255), 2, 8, 0);
55
56      }
57
58      imshow("Original", inputImg);
59      imshow("HistogramB", histImageB);
60      imshow("HistogramG", histImageG);
61      imshow("HistogramR", histImageR);
62
63      waitKey(0);
```