

**과목명 | 디지털영상처리**

**실습 | 3**

**담당교수 | 정진우**

**학과 | 컴퓨터공학과**

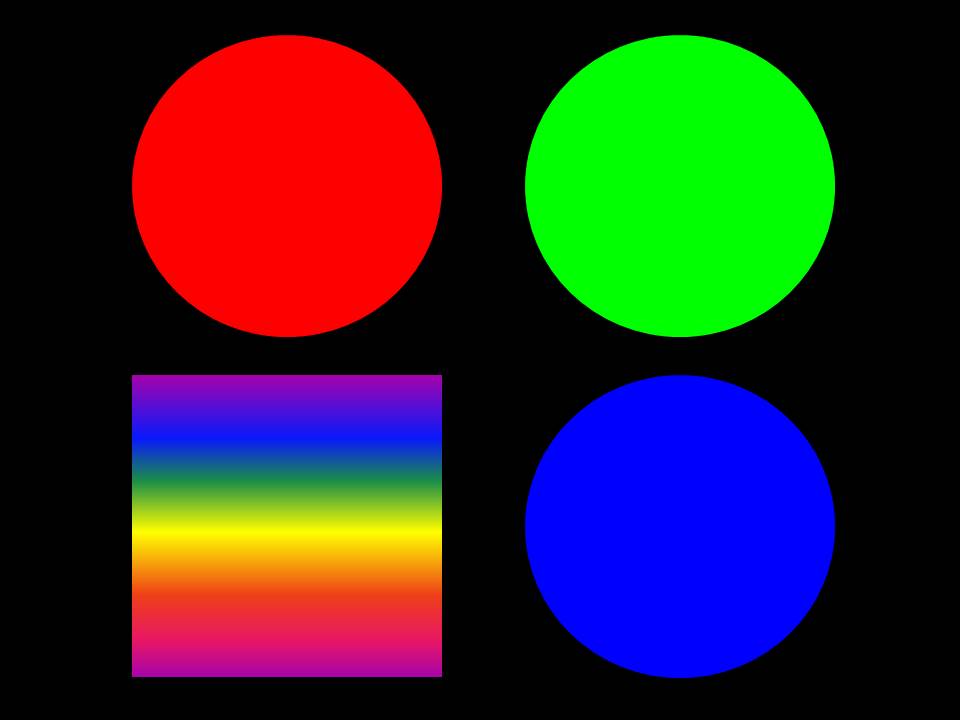
**학년 | 3**

**학번 | 2010112406**

**이름 | 박도령**

**제출일 | 2016. 3. 31**

**실습에 사용한 이미지)**

** **

**mylove.jpg stuff\_color,jpg**

**실습 1)**

♋ **동작 순서 및 원리**

**1. Make histograms of each input image(Input : A any image, b = stuff\_color.jpg file in given materials)**

**(A\_originalHist, B\_originalHist)**

**Image를 매트릭스 객체 형태로 로드하기 위하여 imread 함수를 사용한다. 이미지를 로드한 뒤, Input Image에 대한 히스토그램을 만들기 위해서는 우선 이미지를 Grayscale Image로 변환해야 한다. Grayscale Image로 변환하기 위해서 Opencv에서 지원하는 cvtColor 함수를 사용한다. Grayscale로 변환하기 위하여 CV\_BGR2RAY 옵션을 매개변수로 설정한다.**

//그레이스케일 변환

cvtColor(img1, img1, CV\_BGR2GRAY);

cvtColor(img2, img2, CV\_BGR2GRAY);

**이 코드를 사용하면 img1과 img2를 Grayscale로 변경 가능하다.**

**Grayscale로 변환한 이후에는 히스토그램을 간단히 구현해 보았다. 두 이미지를 위한 0~255 인덱스를 가진 1차원 Mat 객체를 생성한다.**

//히스토그램 생성(1 ~ 255의 배열<usigned short>)

Mat img1\_histo{ 1, 256, CV\_16U };

Mat img2\_histo{ 1, 256, CV\_16U };

**위 코드는 사용할 히스토그램 객체를 생성한 코드이다.**

//히스토그램에 영상의 값 대입

for (int i = 0; i < img1\_row; ++i) {

for (int j = 0; j < img1\_col; ++j) {

auto black = static\_cast<int>(img1.at<uchar>(i, j));//한 픽셀의 값 추출

img1\_histo.at<ushort>(black)++;//해당 픽셀의 인덱스의 값 증가

}

}

**위 코드는 이미지1의 매트릭스 객체의 한 픽셀의 값을 변수 black에 담아 그 값에 해당하는 인덱스의 값을 증가시킴으로써 히스토그램을 만드는 과정을 코드로 구현하였다. 만든 히스토그램을 콘솔 창에 출력함으로써 정확한 값을 확인해 보았다.**

**이미지의 히스토그램을 그래프 이미지 형태로 표현하기 위하여 직접 함수를 만들어 보았다. 이미지 매트릭스와 이미지 이름을 매개변수로 하는 사용자 정의 함수 make\_histogram\_window를 다음과 같이 정의하였다.**

**해당 함수는 우선 이미지를 Opencv에서 지원하는 calcHist 함수를 사용하여 히스토그램을 나타내는 MatND형 객체를 생성한다. 히스토그램을 전에 사용하였던 방법과는 다른 방법이라고 할 수 있다. 그 뒤 calcHist이미지 매트릭스 histImage를 적절한 사이즈로 만들고 line함수를 사용하여 매트릭스의 해당 픽셀의 개수에 따라 그에 사상되는 구역의 점을 위로 올려주면서 이어준다. 그 뒤 완성된 히스토그램 이미지를 화면에 띄어주고 main 함수에서 해당 이미지를 저장하기 위해 이미지를 반환한다. 다음 함수를 이용하여 2개의 Original Image들의 히스토그램을 띄우고 저장할 수 있다.**

//해당 이미지의 히스토그램을 창으로 출력하고 히스토그램 이미지를 반환한다.

Mat make\_histogram\_window(Mat& img, string s) {

int histSize = 256;//히스토그램의 가로 사이즈

double total;

total = img.rows \* img.cols;

int hist\_w = 512;//히스토그램의 가로 길이

int hist\_h = 500;//히스토그램의 높이

int bin\_w = cvRound((double)hist\_w / histSize);//실수형 변수를 반올림한다.

Mat histImage(hist\_h, hist\_w, CV\_8U, Scalar(0, 0, 0));//히스토그램을 표현할 이미지

for (int i = 0; i < histSize; i++) {//세로로 막대그래프를 그려나간다. (색깔 수를 100으로 나누어 표현)

line(histImage, Point(bin\_w \* (i + 1), hist\_h - img.at<ushort>(i)/100),

Point(bin\_w \* (i + 1), hist\_h), CV\_RGB(255, 255, 255));

}

imshow(s, histImage);//완성된 히스토그램 이미지를 보인다.

return histImage;//히스토그램의 이미지를 리턴한다.

}

**2. Make equalized images of each input images after converting to gray scale images (hint: use GS\_histeq functinon)**

**두 이미지를 Histogram Equalization 하기 위하여 equalizedHist 함수를 사용한다. 다음 함수를 사용하면 한번에 Equalize된 이미지를 얻을 수 있다.**

//Histogram Equalization

equalizeHist(img1, img1\_hist\_equalized);

equalizeHist(img2, img2\_hist\_equalized);

**3. Make equalized histograms of each equalized image (A\_equalizedHist, B\_equalizedHist)**

**위에서 만든 사용자 함수 make\_histogram\_window를 사용하여 Equalize된 이미지들의 히스토그램 이미지를 띄우고 저장한다.**

//이미지1, 이미지2 히스토그램 창으로 출력 및 생성

Mat img1\_eq\_hist\_img = make\_histogram\_window(img1\_hist\_equalized, "mylove.jpg - Histogram equalized");

Mat img2\_eq\_hist\_img = make\_histogram\_window(img2\_hist\_equalized, "stuff\_color.jpg - Histogram equalized");

**4. Open the project folder to check output images**

**다음 생성된 모든 이미지를 파일 형태로 저장한다.**

//Save original images

imwrite("A\_originalImg.jpg", img1);

imwrite("B\_originalImg.jpg", img2);

//Save equalized images

imwrite("A\_equalizedImg.jpg", img1\_hist\_equalized);

imwrite("B\_equalizedImg.jpg", img2\_hist\_equalized);

//Save original histogram

imwrite("A\_originalHist.jpg", img1\_og\_hist\_img);

imwrite("B\_originalHist.jpg", img2\_og\_hist\_img);

//Save equalized histogram

imwrite("A\_equalizedHist.jpg", img1\_eq\_hist\_img);

imwrite("B\_equalizedHist.jpg", img2\_eq\_hist\_img);

♋ **실행 결과 및 분석(ReportCode4-1.cpp)**

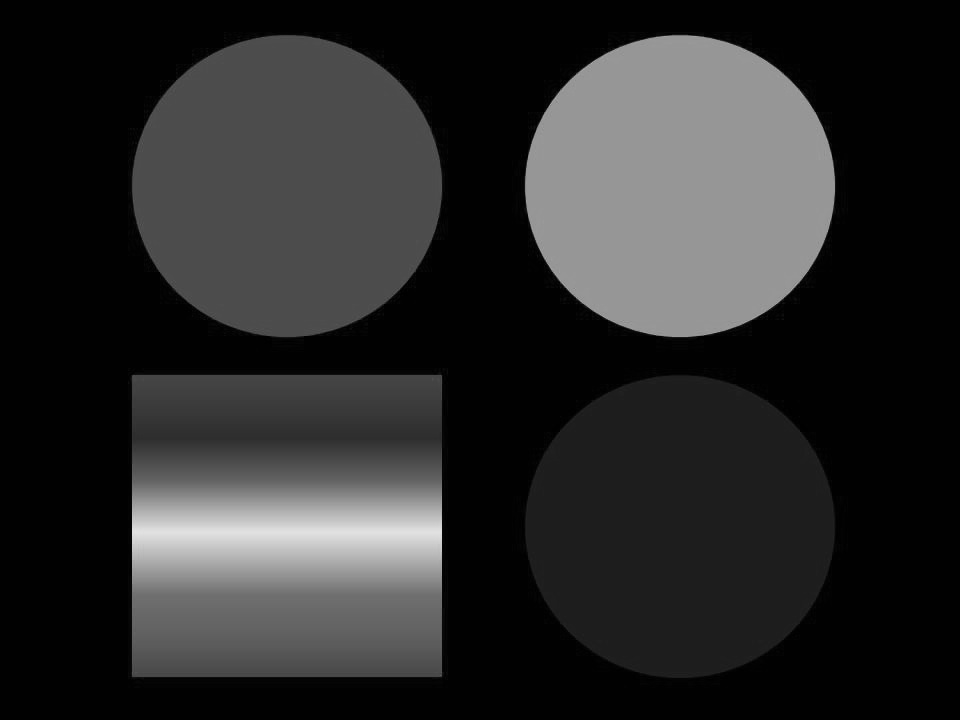
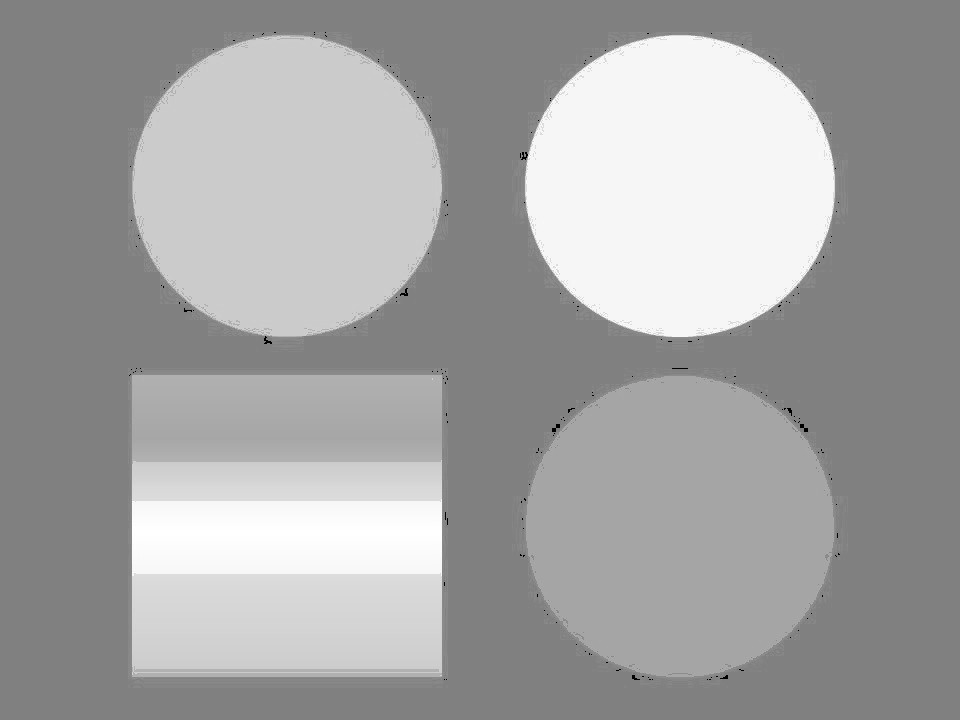
**- Compare original image and equalized image**

**(A\_originalImg vs A\_equalizedImg, B\_originalImg vs B\_equalizedImg)**

** **

**A\_originalImg A\_equalizedImg**

**흰색 부분과 검정색 부분이 조금 더 명확하게 구분 된 것을 확인할 수 있다. 특히 동그라미 부분은 검정색 부분이 더욱 더 도드라지게 변했음을 확인할 수 있다.**

** **

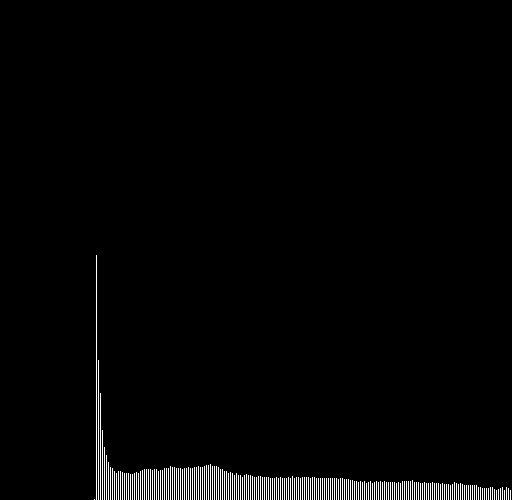
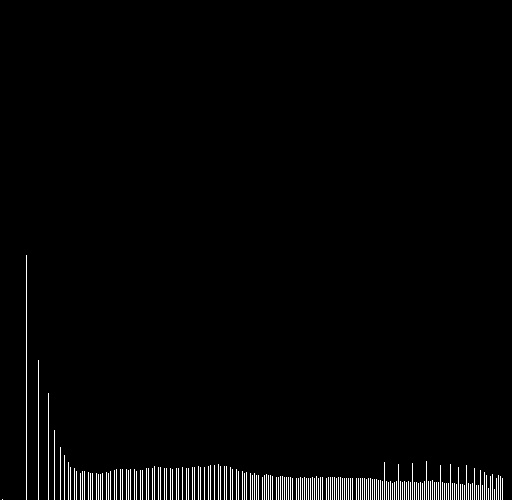
**B\_originalImg B\_equalizedImg**

**Stuff\_color.jpg를 Equalization 하였더니 다음 결과 이미지가 나왔다. 전체적으로 색이 옅어졌으며, 특히 동그라미 부분의 변화가 인상적이다. 색의 그라데이션 효과가 나타났던 원본 이미지와 다르게 Equalization 이후 소수의 단층으로 색깔의 영역이 나누어 졌다. Equalization 과정에서 각 픽셀의 간격이 조정되면서 몇 개의 특정 색깔로 픽셀로 몰리는 현상이 나타났음을 추론할 수 있다.**

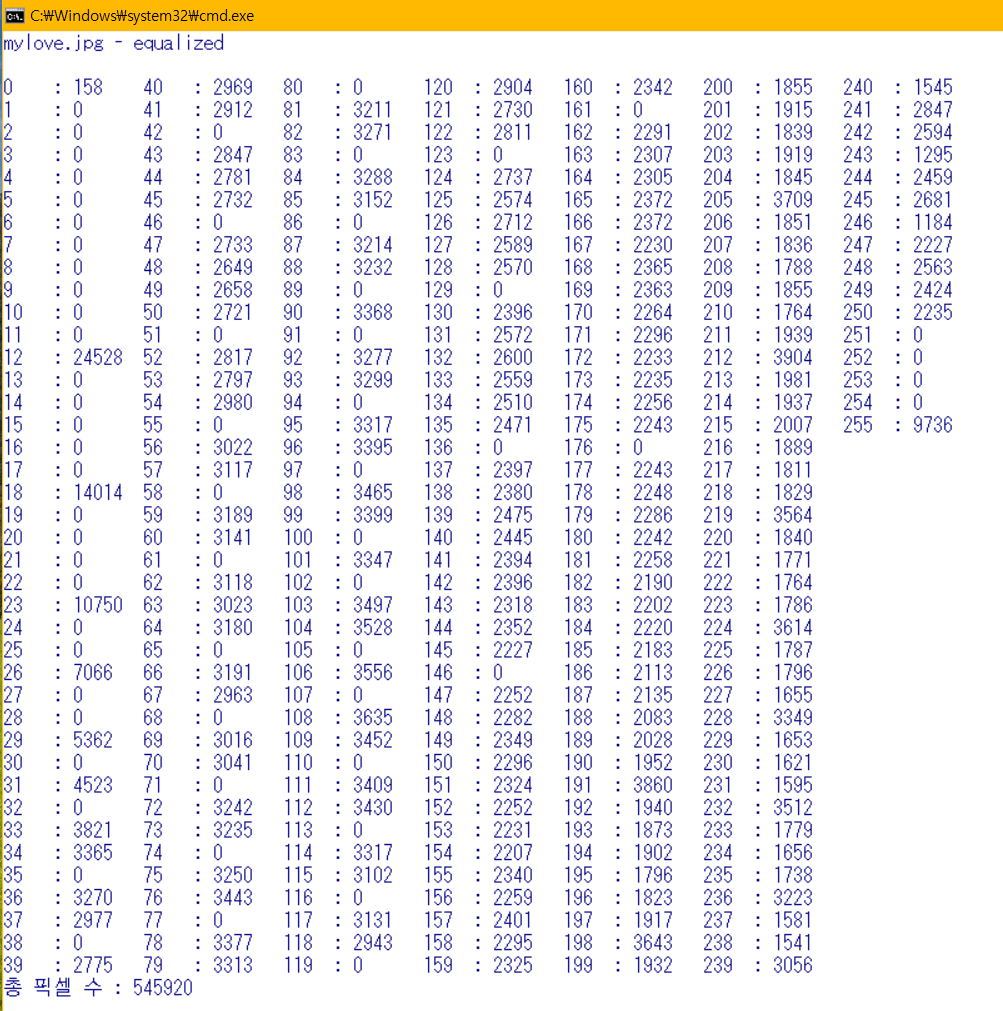
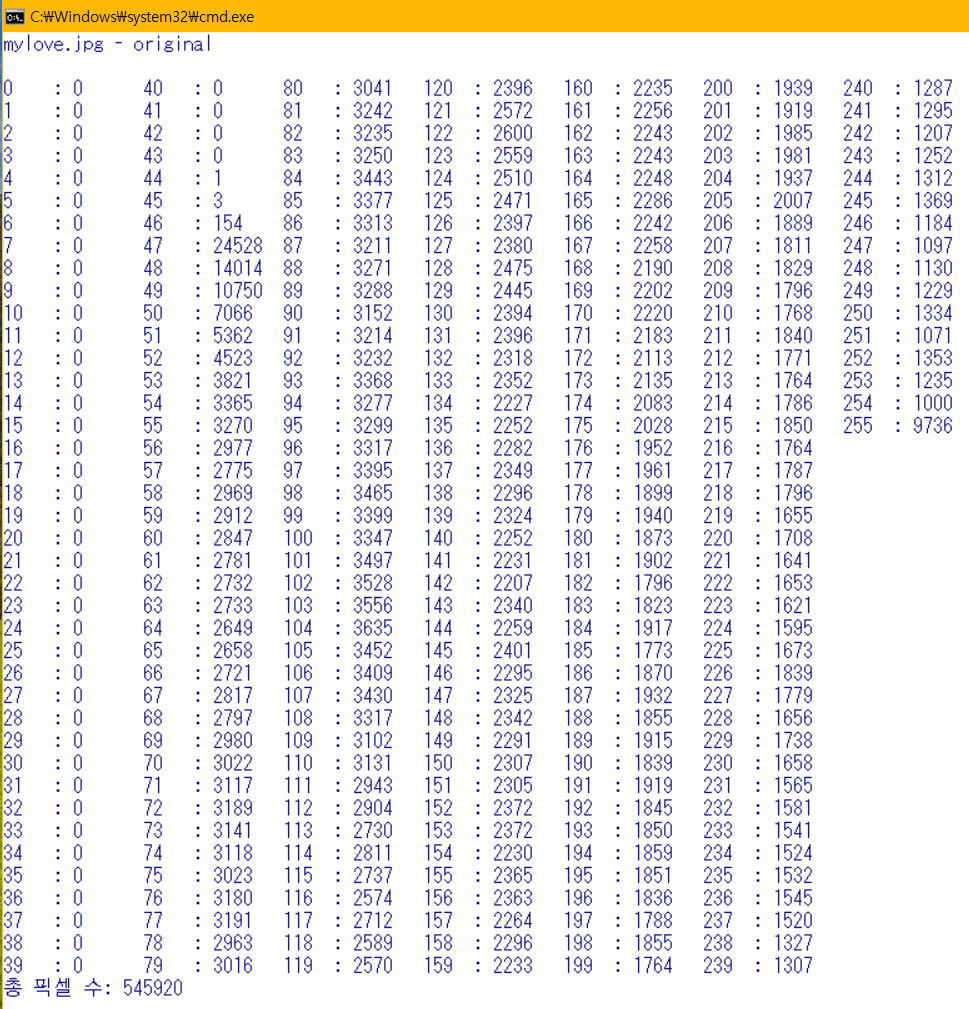
**- Compare original histogram and equalized histogram**

**(A\_originalHist vs A\_equalizedHist, B\_originalHist vs B\_equalizedHist)**

**수치적인 변화를 관찰하기 위하여 각 그림의 히스토그램 이미지와 콘솔 창에 출력한 색깔 당 픽셀 수를 출력해보았다.**

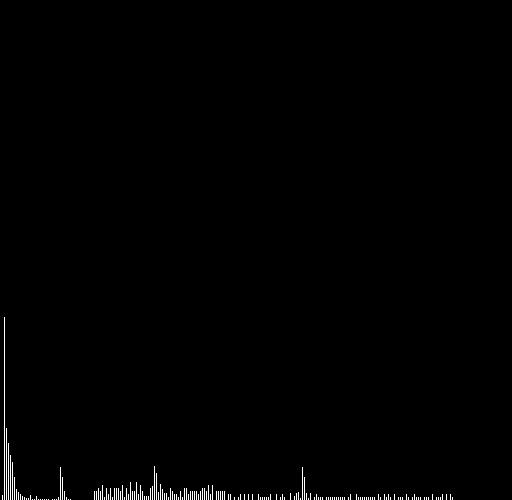
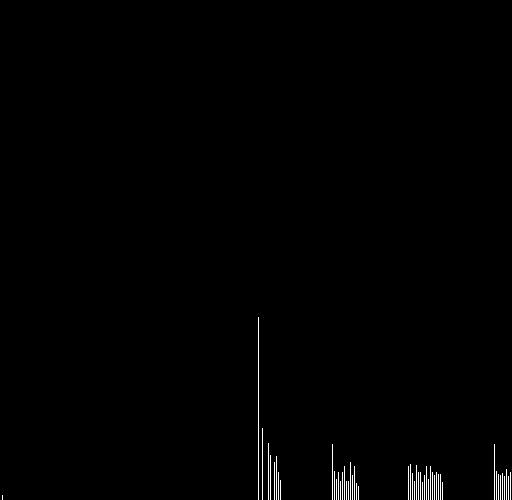
** **

**A\_originalHist A\_equalizedHist**

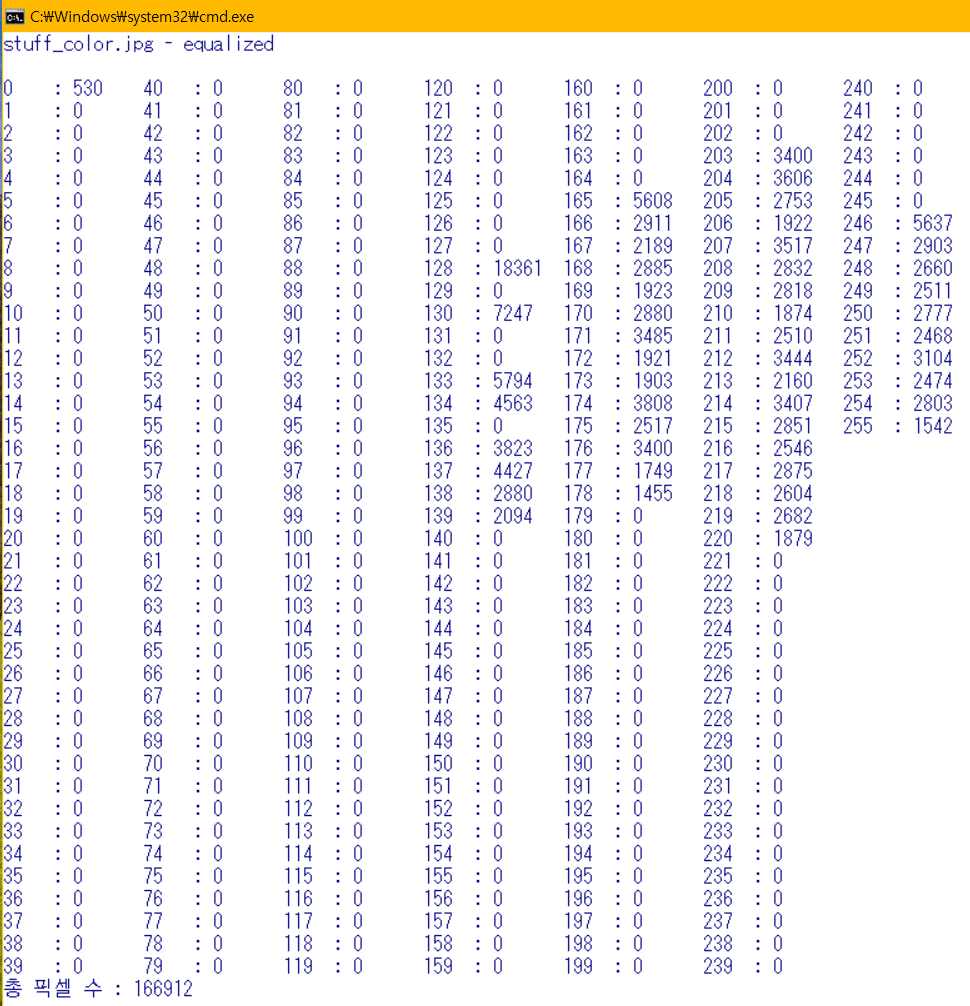
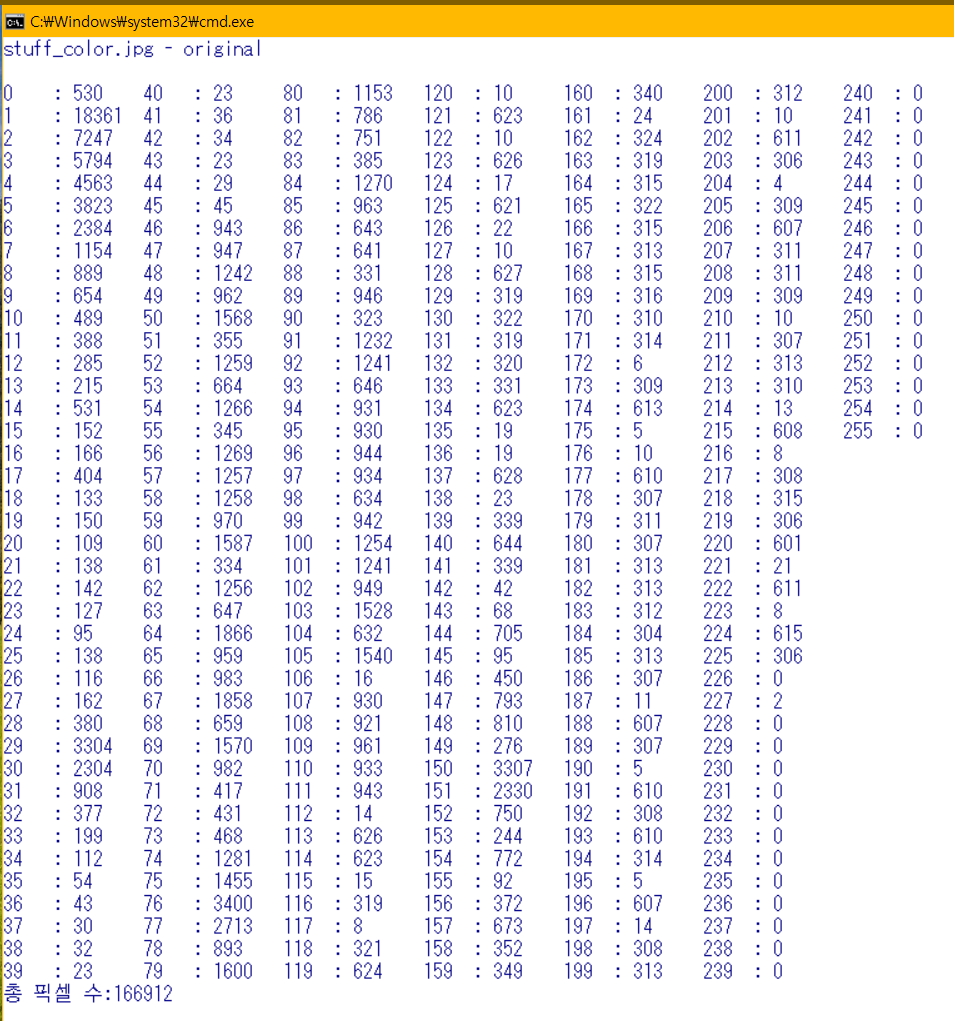


**A\_originalHist A\_equalizedHist**

**두 그래프를 관찰해보면 확실히 Equalized Image가 Original Image보다 넓게 퍼져 있는 것을 확인할 수 있다. 아래는 콘솔 창으로 위의 그래프를 표현한 것이다. 원본 이미지는 0 ~ 45까지 색깔이 거의 존재하지 않지만 변환된 이미지는 0 ~ 45까지도 골고루 퍼져 있는 것을 확인할 수 있다.**

** **

**B\_originalHist B\_equalizedHist**



**B\_originalHist B\_equalizedHist**

**이미지를 Equalize 변환한 뒤 특정 픽셀로 색이 몰리는 것을 확인할 수 있다. 왜냐하면 원본 이미지는 왼쪽의 특정 색깔로 몰려 있기 때문이다. 색깔이 전체적으로 밝아졌지만, 색깔이 특정 픽셀들로 몰리는 현상을 관찰할 수 있다.**