REPORT

Lab01 : Point Operations

과목명	지능형영상처리
담당교수	유 훈 교수님
학과	융합전자공학과
학년	3학년
학번	201910906
이름	이학민
제출일	2023.05.02

상명대학교
SANGMYUNG UNIVERSITY

I. 서론

지능형영상처리 9주차 수업에서 다룬 Lab01은 아래의 'lena.png'를 테스트 이미지로 사용하여 Point Operations를 수행하는 실습이다. Point Operation은 이미지의 품질을 개선하기 위한 과정으로 Contrast Enhancement가 그 중 하나이다. 보통 Linear Operation을 통해 Contrast Enhancement가 가능한데, 각 계수를 구하기 위해 xmax와 xmin의 값을 얻어야 하고 그 방법은 다음과 같다.

① xmax와 xmin에 단순히 최대·최소의 값을 이용하기
② 히스토그램에서 상위 n%, 하위 n%의 값을 찾아 이용하기 (단, n = 5, 10, 20)
③ 히스토그램 평활화를 이용하기

Matlab에서 3가지 경우에 대한 Point Operations를 수행하였을 때 얻어진 각각의 결과와 작성한 코드 및 설명을 첨부하였다. 마지막으로 서로 다른 결과를 한눈에 비교하기 위하여 R/G/B 색상 채널별로 재분류하여 결과를 분석해 보았다.

테스트 이미지 - 'lena.png'

II. 본론

1) Point Operation 미처리

왼쪽부터 imgR / imgG / imgB

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
figure(1);
imshow(img);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
figure(2);
imshow([imgR, imgG, imgB]);
	이 코드는 이미지를 불러와서 RGB 채널 별로 나누고, 이를 하나의 figure 안에 서브플롯으로 띄우는 코드이다.
	각각의 색상 채널(Red, Green, Blue)에 대해 imgR, imgG, imgB로 분리하기 위해 img(:,:,1), img(:,:,2), img(:,:,3)과 같이 인덱싱을 사용한다.
	즉, img(:,:,1)은 원본 이미지의 빨간색(Red) 채널에 해당하는 배열을 추출하고, img(:,:,2)은 녹색(Green) 채널, img(:,:,3)은 파란색(Blue) 채널에 해당하는 배열을 각각 추출한다. 이를 통해 세 가지 색상 채널을 독립적으로 조작할 수 있게 된다.
	imshow() 함수는 입력된 배열을 이미지로 보여주는 함수이다. imshow() 함수에 RGB 이미지를 입력하면 자동으로 컬러 이미지로 출력된다. imshow() 함수에 여러 개의 입력 이미지를 넣고자 할 때는, 입력 이미지를 [imgR, imgG, imgB]와 같이 하나의 배열로 만들어서 입력하면 된다.

2) 최대·최소의 값을 이용한 Point Operation 처리

① imgR

imgR

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Stretching
imgX = double(imgR);
xmin = min(imgX(:));
xmax = max(imgX(:));
ymin = 0;
ymax = 255;
a = (ymax-ymin)/(xmax-xmin);
b = -a*xmin;
imgY = a*imgX+b;
figure(1);
imshow([imgX, imgY]/255);
	imgX = double(imgR); : 빨간색(Red) 채널 이미지 데이터를 double 형태로 변환하여 imgX 변수에 저장한다.
	xmin = min(imgX(:)); : imgX 변수의 최소값을 구한다.
	xmax = max(imgX(:)); : imgX 변수의 최대값을 구한다.
	ymin = 0; : 출력 이미지의 최소값을 지정한다.
	ymax = 255; : 출력 이미지의 최대값을 지정한다.
	a = (ymax-ymin)/(xmax-xmin); : 스트레칭 함수의 기울기를 계산한다.
	b = -a*xmin; : 스트레칭 함수의 y절편을 계산한다.
	imgY = a*imgX+b; : 스트레칭 함수를 적용하여 출력 이미지를 계산한다.
	figure(1); imshow([imgX, imgY]/255); : 입력 이미지와 출력 이미지를 옆으로 붙여서 보여준다. 여기서 255로 나누는 이유는 imshow() 함수가 0~1 사이의 값을 입력으로 받기 때문이다.

② imgG

imgG

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Stretching
imgX = double(imgG);
xmin = min(imgX(:));
xmax = max(imgX(:));
ymin = 0;
ymax = 255;
a = (ymax-ymin)/(xmax-xmin);
b = -a*xmin;
imgY = a*imgX+b;
figure(1);
imshow([imgX, imgY]/255);
	imgX = double(imgG); : 녹색(Green) 채널 이미지 데이터를 double 형태로 변환하여 imgX 변수에 저장한다.
	xmin = min(imgX(:)); : imgX 변수의 최소값을 구한다.
	xmax = max(imgX(:)); : imgX 변수의 최대값을 구한다.
	ymin = 0; : 출력 이미지의 최소값을 지정한다.
	ymax = 255; : 출력 이미지의 최대값을 지정한다.
	a = (ymax-ymin)/(xmax-xmin); : 스트레칭 함수의 기울기를 계산한다.
	b = -a*xmin; : 스트레칭 함수의 y절편을 계산한다.
	imgY = a*imgX+b; : 스트레칭 함수를 적용하여 출력 이미지를 계산한다.
	figure(1); imshow([imgX, imgY]/255); : 입력 이미지와 출력 이미지를 옆으로 붙여서 보여준다. 여기서 255로 나누는 이유는 imshow() 함수가 0~1 사이의 값을 입력으로 받기 때문이다.

③ imgB

imgB

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Stretching
imgX = double(imgB);
xmin = min(imgX(:));
xmax = max(imgX(:));
ymin = 0;
ymax = 255;
a = (ymax-ymin)/(xmax-xmin);
b = -a*xmin;
imgY = a*imgX+b;
figure(1);
imshow([imgX, imgY]/255);
	imgX = double(imgB); : 파란색(Blue) 채널 이미지 데이터를 double 형태로 변환하여 imgX 변수에 저장한다.
	xmin = min(imgX(:)); : imgX 변수의 최소값을 구한다.
	xmax = max(imgX(:)); : imgX 변수의 최대값을 구한다.
	ymin = 0; : 출력 이미지의 최소값을 지정한다.
	ymax = 255; : 출력 이미지의 최대값을 지정한다.
	a = (ymax-ymin)/(xmax-xmin); : 스트레칭 함수의 기울기를 계산한다.
	b = -a*xmin; : 스트레칭 함수의 y절편을 계산한다.
	imgY = a*imgX+b; : 스트레칭 함수를 적용하여 출력 이미지를 계산한다.
	figure(1); imshow([imgX, imgY]/255); : 입력 이미지와 출력 이미지를 옆으로 붙여서 보여준다. 여기서 255로 나누는 이유는 imshow() 함수가 0~1 사이의 값을 입력으로 받기 때문이다.

3) 히스토그램에서 n%의 값을 이용한 Point Operation 처리

① n=5

imgR

imgG

imgB

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Stretching
imgX_R = double(imgR);
imgX_G = double(imgG);
imgX_B = double(imgB);
[hdata_R, ~] = histcounts(imgX_R(:), 256);
cumhist_R = cumsum(hdata_R) / numel(imgX_R);
[hdata_G, ~] = histcounts(imgX_G(:), 256);
cumhist_G = cumsum(hdata_G) / numel(imgX_G);
[hdata_B, ~] = histcounts(imgX_B(:), 256);
cumhist_B = cumsum(hdata_B) / numel(imgX_B);
xmin = find(cumhist_R > 0.05, 1, 'first');
xmax = find(cumhist_R >= 0.95, 1, 'first') - 1;
ymin = 0;
ymax = 255;
a = (ymax - ymin) / (xmax - xmin);
b = -a * xmin;
imgY_R = a * imgX_R + b;
imgY_G = a * imgX_G + b;
imgY_B = a * imgX_B + b;
figure(1);
imshow([imgX_R, imgY_R] / 255);
figure(2);
imshow([imgX_G, imgY_G] / 255);
figure(3);
imshow([imgX_B, imgY_B] / 255);
	이 코드는 RGB 이미지의 각 채널에 대한 히스토그램 스트레칭을 수행하는 코드이다.
	
	우선, 이미지 파일을 불러와서 R, G, B 채널로 분리하고, 이들을 double 자료형으로 변환한다.
	
	그 다음, R, G, B 채널 각각에 대해 256개의 bin으로 나누어 히스토그램을 계산한다. 그리고 각 bin의 값에 해당하는 누적 분포 함수(cumulative distribution function, CDF)를 계산한다. 이렇게 함으로써 각 픽셀 값이 해당 픽셀 값 이하의 값을 가진 픽셀의 비율을 계산할 수 있다.
	
	CDF를 이용해서 히스토그램 스트레칭의 범위를 결정한다. 여기서는 CDF가 0.05 이상인 픽셀 값 중 가장 작은 값을 xmin으로, CDF가 0.95 이상인 픽셀 값 중 가장 작은 값을 xmax로 설정한다. 이 범위 내에서 히스토그램이 골고루 분포하도록 하는 변환 함수를 계산한다.
	
	마지막으로, R, G, B 채널 각각에 대해 변환 함수를 적용하여 히스토그램 스트레칭 된 이미지를 얻는다. 그리고 이를 각각의 색 채널에 대해 따로 시각화한다.

② n=10

imgR

imgG

imgB

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Stretching
imgX_R = double(imgR);
imgX_G = double(imgG);
imgX_B = double(imgB);
[hdata_R, ~] = histcounts(imgX_R(:), 256);
cumhist_R = cumsum(hdata_R) / numel(imgX_R);
[hdata_G, ~] = histcounts(imgX_G(:), 256);
cumhist_G = cumsum(hdata_G) / numel(imgX_G);
[hdata_B, ~] = histcounts(imgX_B(:), 256);
cumhist_B = cumsum(hdata_B) / numel(imgX_B);
xmin = find(cumhist_R > 0.10, 1, 'first');
xmax = find(cumhist_R >= 0.90, 1, 'first') - 1;
ymin = 0;
ymax = 255;
a = (ymax - ymin) / (xmax - xmin);
b = -a * xmin;
imgY_R = a * imgX_R + b;
imgY_G = a * imgX_G + b;
imgY_B = a * imgX_B + b;
figure(1);
imshow([imgX_R, imgY_R] / 255);
figure(2);
imshow([imgX_G, imgY_G] / 255);
figure(3);
imshow([imgX_B, imgY_B] / 255);
	이 코드는 RGB 이미지의 각 채널에 대한 히스토그램 스트레칭을 수행하는 코드이다.
	
	우선, 이미지 파일을 불러와서 R, G, B 채널로 분리하고, 이들을 double 자료형으로 변환한다.
	
	그 다음, R, G, B 채널 각각에 대해 256개의 bin으로 나누어 히스토그램을 계산한다. 그리고 각 bin의 값에 해당하는 누적 분포 함수(cumulative distribution function, CDF)를 계산한다. 이렇게 함으로써 각 픽셀 값이 해당 픽셀 값 이하의 값을 가진 픽셀의 비율을 계산할 수 있다.
	
	CDF를 이용해서 히스토그램 스트레칭의 범위를 결정한다. 여기서는 CDF가 0.10 이상인 픽셀 값 중 가장 작은 값을 xmin으로, CDF가 0.90 이상인 픽셀 값 중 가장 작은 값을 xmax로 설정한다. 이 범위 내에서 히스토그램이 골고루 분포하도록 하는 변환 함수를 계산한다.
	
	마지막으로, R, G, B 채널 각각에 대해 변환 함수를 적용하여 히스토그램 스트레칭 된 이미지를 얻는다. 그리고 이를 각각의 색 채널에 대해 따로 시각화한다.

③ n=20

imgR

imgG

imgB

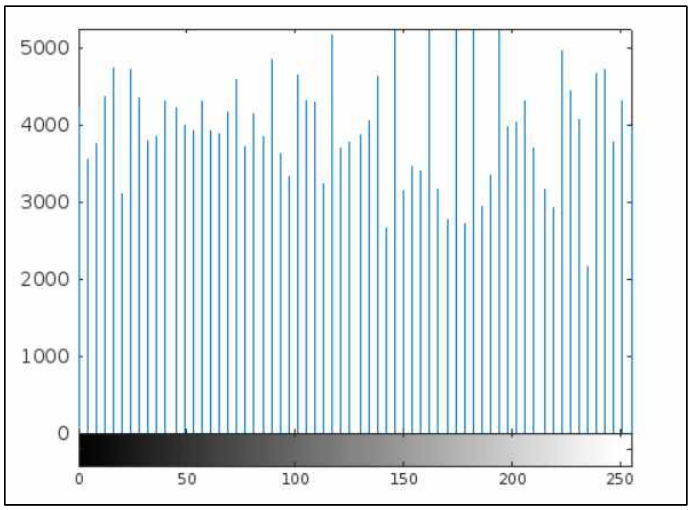
Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Stretching
imgX_R = double(imgR);
imgX_G = double(imgG);
imgX_B = double(imgB);
[hdata_R, ~] = histcounts(imgX_R(:), 256);
cumhist_R = cumsum(hdata_R) / numel(imgX_R);
[hdata_G, ~] = histcounts(imgX_G(:), 256);
cumhist_G = cumsum(hdata_G) / numel(imgX_G);
[hdata_B, ~] = histcounts(imgX_B(:), 256);
cumhist_B = cumsum(hdata_B) / numel(imgX_B);
xmin = find(cumhist_R > 0.20, 1, 'first');
xmax = find(cumhist_R >= 0.80, 1, 'first') - 1;
ymin = 0;
ymax = 255;
a = (ymax - ymin) / (xmax - xmin);
b = -a * xmin;
imgY_R = a * imgX_R + b;
imgY_G = a * imgX_G + b;
imgY_B = a * imgX_B + b;
figure(1);
imshow([imgX_R, imgY_R] / 255);
figure(2);
imshow([imgX_G, imgY_G] / 255);
figure(3);
imshow([imgX_B, imgY_B] / 255);
	이 코드는 RGB 이미지의 각 채널에 대한 히스토그램 스트레칭을 수행하는 코드이다.
	
	우선, 이미지 파일을 불러와서 R, G, B 채널로 분리하고, 이들을 double 자료형으로 변환한다.
	
	그 다음, R, G, B 채널 각각에 대해 256개의 bin으로 나누어 히스토그램을 계산한다. 그리고 각 bin의 값에 해당하는 누적 분포 함수(cumulative distribution function, CDF)를 계산한다. 이렇게 함으로써 각 픽셀 값이 해당 픽셀 값 이하의 값을 가진 픽셀의 비율을 계산할 수 있다.
	
	CDF를 이용해서 히스토그램 스트레칭의 범위를 결정한다. 여기서는 CDF가 0.20 이상인 픽셀 값 중 가장 작은 값을 xmin으로, CDF가 0.80 이상인 픽셀 값 중 가장 작은 값을 xmax로 설정한다. 이 범위 내에서 히스토그램이 골고루 분포하도록 하는 변환 함수를 계산한다.
	
	마지막으로, R, G, B 채널 각각에 대해 변환 함수를 적용하여 히스토그램 스트레칭 된 이미지를 얻는다. 그리고 이를 각각의 색 채널에 대해 따로 시각화한다.

4) 히스토그램 평활화를 이용한 Point Operation 처리

① imgR

imgR



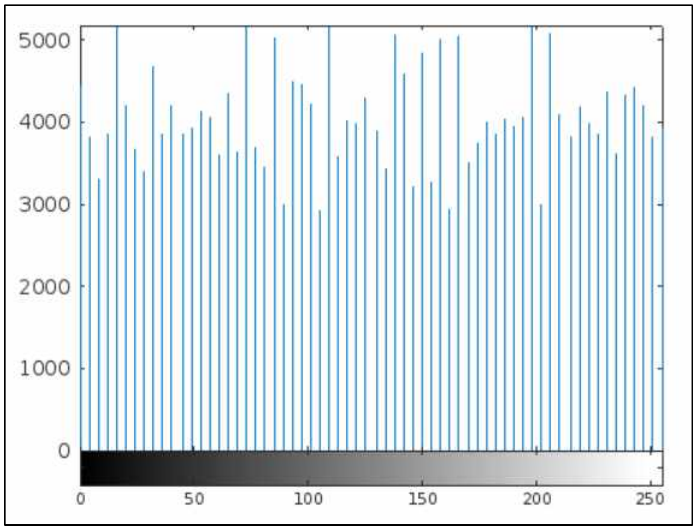
equalized histogram of imgR

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Equalization
imgEQ = histeq(imgR);
figure(1);
imshow([imgR, imgEQ]);
figure(2);
imhist(imgEQ);
	히스토그램 평활화는 이미지의 대비를 개선하여 더 선명한 이미지를 생성하는 데 사용된다.
	histeq 함수는 입력 이미지의 히스토그램을 분석하고, 각 픽셀 값의 빈도수를 동일하게 만드는 새로운 픽셀값으로 이미지를 변경한다.
	그 다음 imshow 함수를 사용하여 원본 이미지와 변환된 이미지를 옆으로 나란히 출력하고, imhist 함수를 사용하여 변환된 이미지의 히스토그램을 출력한다.

② imgG

imgG



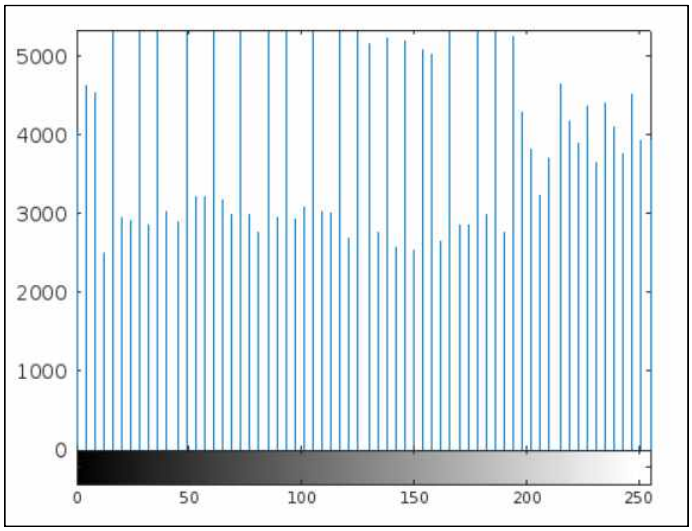
equalized histogram of imgG

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Equalization
imgEQ = histeq(imgG);
figure(1);
imshow([imgG, imgEQ]);
figure(2);
imhist(imgEQ);
	히스토그램 평활화는 이미지의 대비를 개선하여 더 선명한 이미지를 생성하는 데 사용된다.
	histeq 함수는 입력 이미지의 히스토그램을 분석하고, 각 픽셀 값의 빈도수를 동일하게 만드는 새로운 픽셀값으로 이미지를 변경한다.
	그 다음 imshow 함수를 사용하여 원본 이미지와 변환된 이미지를 옆으로 나란히 출력하고, imhist 함수를 사용하여 변환된 이미지의 히스토그램을 출력한다.

③ imgB

imgB



equalized histogram of imgB

Matlab code	

fname = 'lena.png';
img = imread(fname);
 	
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);
 	
% Histogram Equalization
imgEQ = histeq(imgB);
figure(1);
imshow([imgB, imgEQ]);
figure(2);
imhist(imgEQ);
	히스토그램 평활화는 이미지의 대비를 개선하여 더 선명한 이미지를 생성하는 데 사용된다.
	histeq 함수는 입력 이미지의 히스토그램을 분석하고, 각 픽셀 값의 빈도수를 동일하게 만드는 새로운 픽셀값으로 이미지를 변경한다.
	그 다음 imshow 함수를 사용하여 원본 이미지와 변환된 이미지를 옆으로 나란히 출력하고, imhist 함수를 사용하여 변환된 이미지의 히스토그램을 출력한다.

5) 결과 비교 및 분석

서로 다른 Point Operation 방법에 따른 이미지 결과를 비교하기 위해 색상 채널 별로 묶어 나열하였다. 아래의 색상별 이미지 나열 순서는 다음을 따른다.

(원본 → 단순최대최소 → n=5 → n=10 → n=20 → 히스토그램 평활화)

① imgR

② imgG

③ imgB

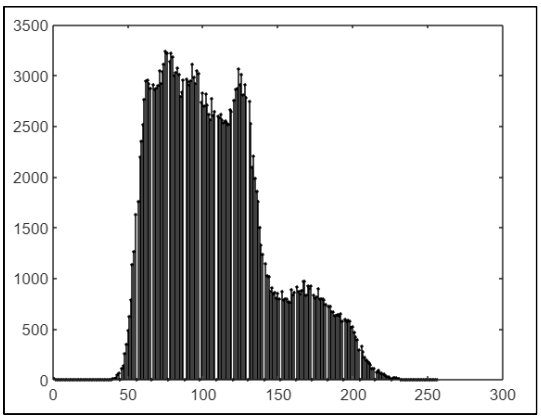
Point Operation을 수행한 뒤의 이미지 결과를 원본과 비교해 보면, 원본에 비해 명암 대비가 더 뚜렷해져 원본보다 더 선명한 품질이 얻어졌음을 시각적으로 알 수 있다. 특히 상위 n%의 값을 xmax, 하위 n%의 값을 xmin으로 사용하는 방법의 경우 n의 값에 따라 개선된 이미지 결과의 차이가 확연히 드러난다. R/G/B 모두 n=5 또는 n=10 일 때 준수한 품질을 보인다. 하지만 n=20과 같이 n의 값이 너무 커져버리면 imgR과 같이 이미지가 너무 밝아지거나 imgG 또는 imgB와 같이 이미지가 과도하게 어두워지는 현상이 나타난다. 색상별로 나열된 6장의 이미지를 비교해 보았을 때 히스토그램 평활화를 이용한 Point Operation 결과가 가장 뛰어남을 알 수 있는데 이에 대한 이유는 결론에서 다루도록 한다.

III. 결론

앞서 다양한 방법의 Point Operations를 Matlab을 통해 구현하여 이미지 결과를 얻었다. Matlab 환경이 아직 익숙하지 않아 코드를 짜고 이해하는데 인공지능 ChatGPT의 도움을 받았고, 원활하게 실습을 진행할 수 있었다.
실습을 통해 얻은 이미지를 한 줄에 나열해 놓고 비교했을 때 가장 좋은 결과를 보이는 것은 히스토그램 평활화를 이용한 방법이었다. 히스토그램 평활화란 원본 이미지 PDF의 CDF를 이용하여 Point Operation을 수행한 것으로, 본론 4)의 equalized histogram 그림들과 같이 Uniform한 형태의 히스토그램을 얻을 수 있다. 이는 이미지가 줄 수 있는 정보량이 최대임을 의미하고, 이미지의 품질이 가장 높게 개선됐음을 뜻한다.
단순 최대·최소값을 이용하여 xmax, xmin의 값을 결정하는 것 보다는 히스토그램의 상·하위 n%의 값을 이용하여 Point Operation을 수행하는 것이 더 나은 이미지를 얻을 수 있다. 하지만 n의 값이 과도하게 커져 버리면 이미지의 밝기가 너무 밝거나 어두워지는 현상이 나타났다.(n=20) 따라서 위 실습을 통해 상·하위 5%의 값을 xmax와 xmin에 이용하는 것이 이미지의 품질을 개선하는데 있어 가장 합리적인 선택임을 알 수 있었다.

cf) 수업의 Matlab 실습 中 histogram count figure

Matlab code




```
fname = 'lena.png';
img = imread(fname);
imgR = img(:,:,1);
imgG = img(:,:,2);
imgB = img(:,:,3);

hdata=histogram(imgB, 256);
histcnt = hdata.BinCounts;
figure(3);
stem(histcnt, 'k.');
```

이 코드는 'lena.png'라는 파일을 불러와서, 해당 이미지의 파란색 채널을 선택하여 히스토그램을 구하고, 구해진 히스토그램의 빈도수를 가시화하는 코드이다.
이미지 처리에서 히스토그램은 해당 이미지에서 각 픽셀 값이 몇 번 등장하는지 빈도를 보여주는 그래프이다. histogram() 함수는 입력된 데이터에서 히스토그램을 구하는 함수이며, 이 함수를 사용하여 imgB에서 256개의 구간으로 나누어 히스토그램 데이터를 구한다. 이 구간의 수를 더 적게 하면 더 큰 값으로 픽셀 값을 표현할 수 있지만, 그만큼 정밀도가 떨어지게 된다.
stem() 함수는 히스토그램 데이터를 막대 그래프 대신 점으로 나타낸다. 이 코드에서는 막대 그래프 대신 점으로 나타내는 이유는, 이미지에서 픽셀 값의 분포가 적은 부분과 많은 부분이 있을 수 있기 때문이다.