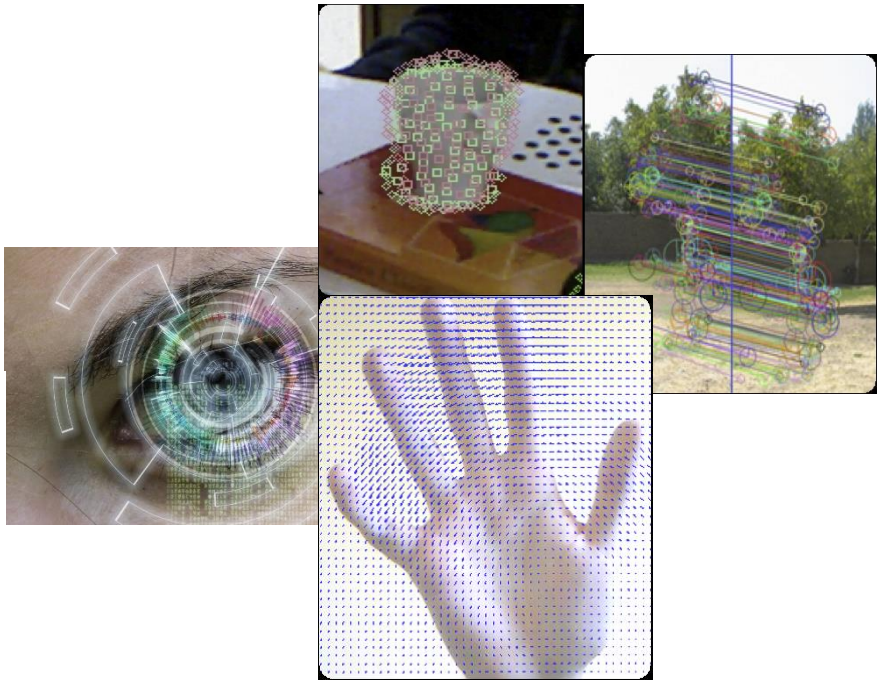
IVPL
Intelligent Vision Processing Lab

2023 Fall

COMPUTER VISION 비젼 프로그래밍



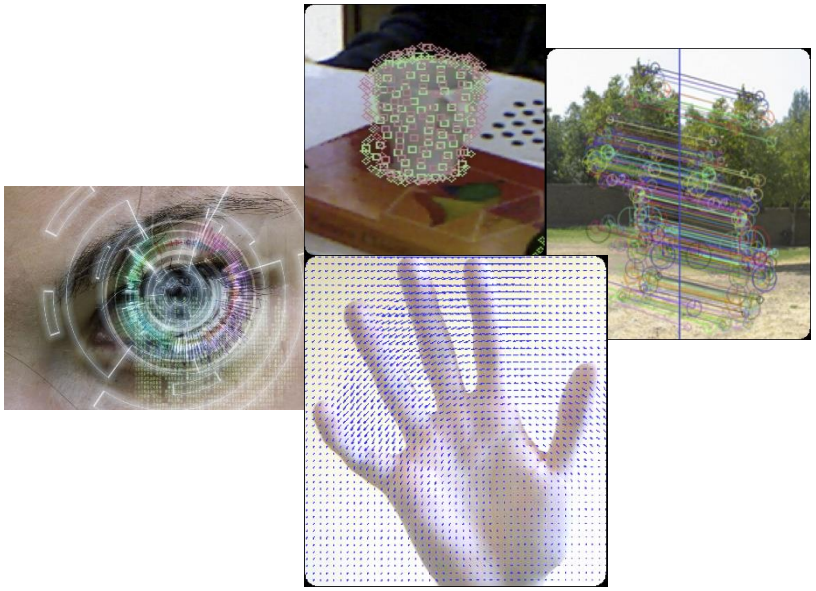
4장. 히스토그램 처리 (histogram processing)(Theory)

지난 강의 요약 : 화소 점 처리 (pixel point processing)

- 사람의 시각적 데이터 센싱 구조
- 화소(pixel)의 정의
- 화소 점처리(pixel point processing)에 대한 소개
  - 현재 처리하고자 하는 화소의 값만을 처리에 고려하는 구조
- 다양한 화소 점처리 (pixel point processing) 기법 소개
  - 산술/논리 연산
  - 반전, 이진화, 향상, 대역통과/제거 등등
  - 색상 대비, 감마 보정 기법, 비트 슬라이싱 기법

COMPUTER VISION 비젼 프로그래밍

Histogram Processing



IVPL
Intelligent Vision Processing Lab

학습 목표

- 히스토그램(histogram)이란?
- 히스토그램(histogram)을 어떻게 활용하는가?
- 히스토그램 처리(histogram processing)의 종류는?
- 실제 활용 방안 고민해 보자.

확률의 개념

확률이란?

- 도수이론(frequentist view)
  - 확률은 한 시행을 동일한 조건 하에서 독립적으로 반복할 때 그 사건이 일어날 것으로 예측되는 횟수의 전체 시행횟수에 대한 백분율 (상대 빈도)

- 주관적 견해(subjective view)
  - 사건에 대한 주관적 확신의 정도. 이는 반복시행 여부와 관계없이 정의 가능함

- 하나의 사상(事象) 혹은 사건이 일어날 수 있는 가능성의 정도. 또는, 그 수치

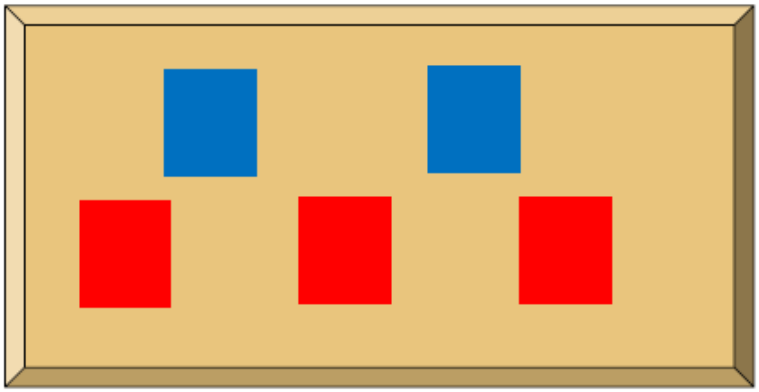
확률 값에 대한 계산

- 동전 토스할 때: 한 쪽면이 나올 확률 = 1/2

- 주사위를 던질 때
  - 임의의 숫자가 나올 확률은? 1/6

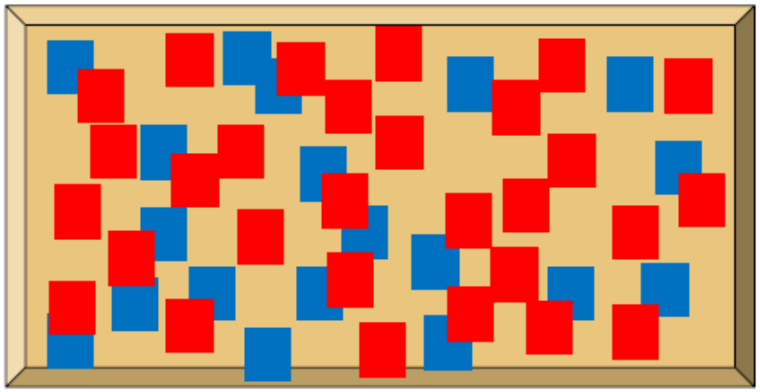
확률 값에 대한 계산

- 상자 A와 상자 B로부터 붉은 카드를 꺼낼 확률 P(R)은?



상자 A

(붉은 카드 3장, 푸른 카드 2장)



상자 B

(붉은 카드 30장, 푸른 카드 20장)

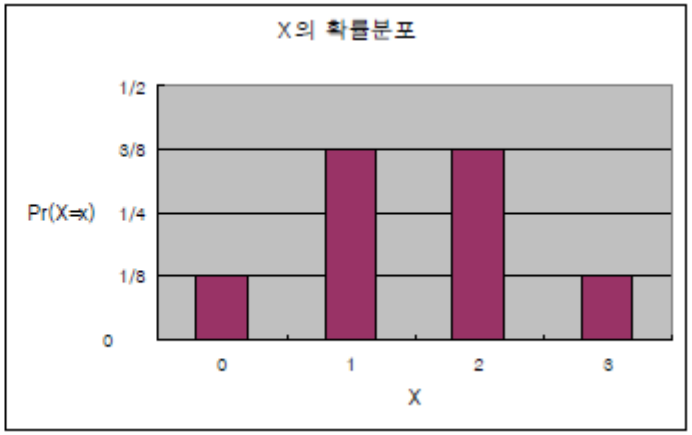
$$P(R) = \frac{\text{붉은 카드의 수}}{\text{전체 카드의 수}} = \frac{3}{5}$$

확률 분포 (Probability Distribution)

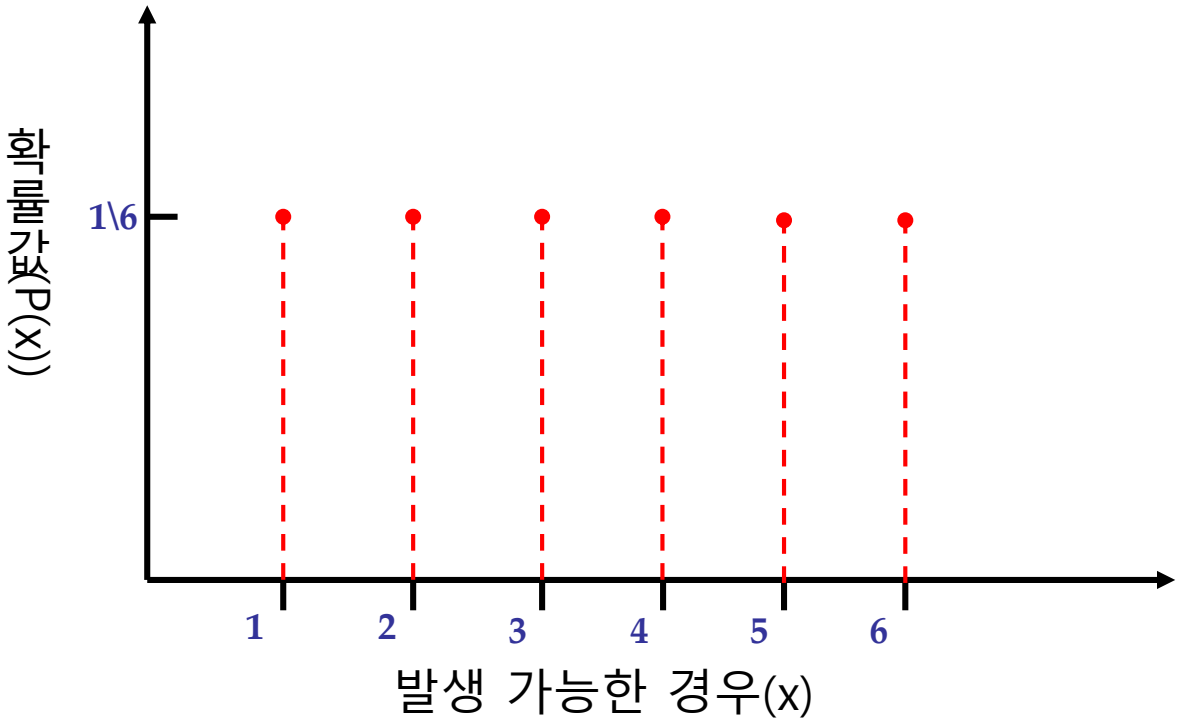
- 확률 분포란?
  - *확률분포*(確率分布)는 확률변수가 특정한 값을 가질 확률을 나타내는 함수
  - 확률 변수가 취하는 범위와 해당 확률을 대응시켜 표현한 것
- 확률 변수
  - 일정한 확률을 가지고 발생하는 사건에 수치를 부여하기 위해 변수화 시킨 것

<표 2.1> X의 확률분포표

x	0	1	2	3
Pr(X=x)	1/8	3/8	3/8	1/8



확률 분포 (Probability Distribution)



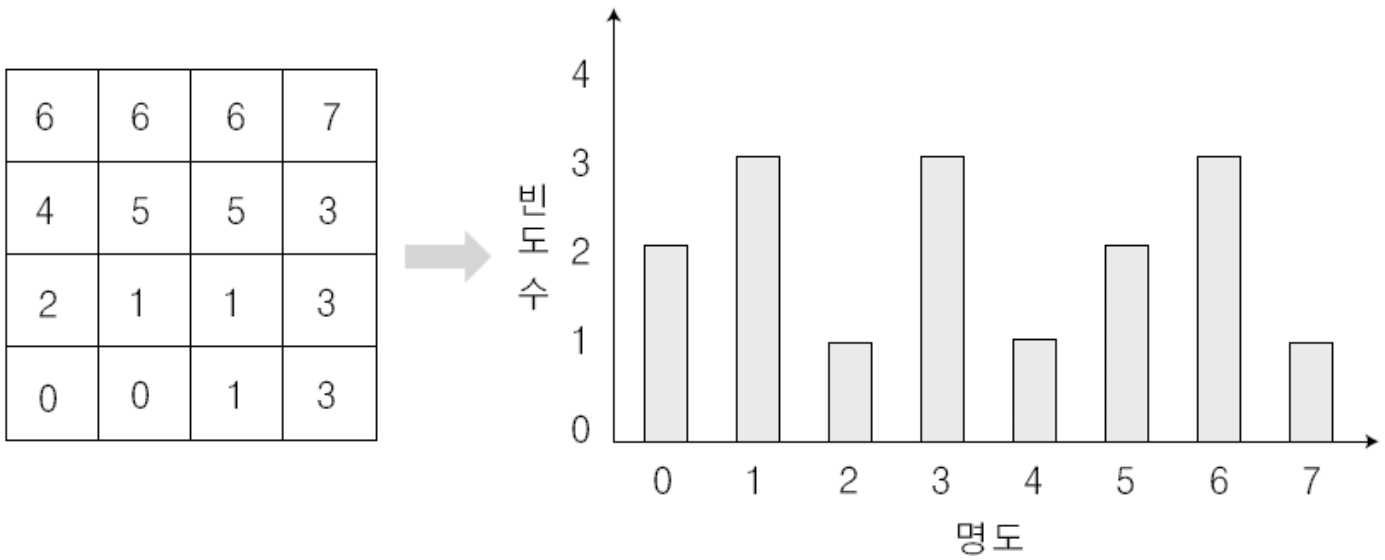
영상에서의 확률변수 및 확률 분포

- 확률 변수(x): 밝기 값 ([0~255])
- 밝기(x)에 대한 확률 분포(p(x))
  - 개별 밝기 값에 해당하는 발생 확률을 함수로 표현 (그래프로 표현)

디지털 영상의 히스토그램

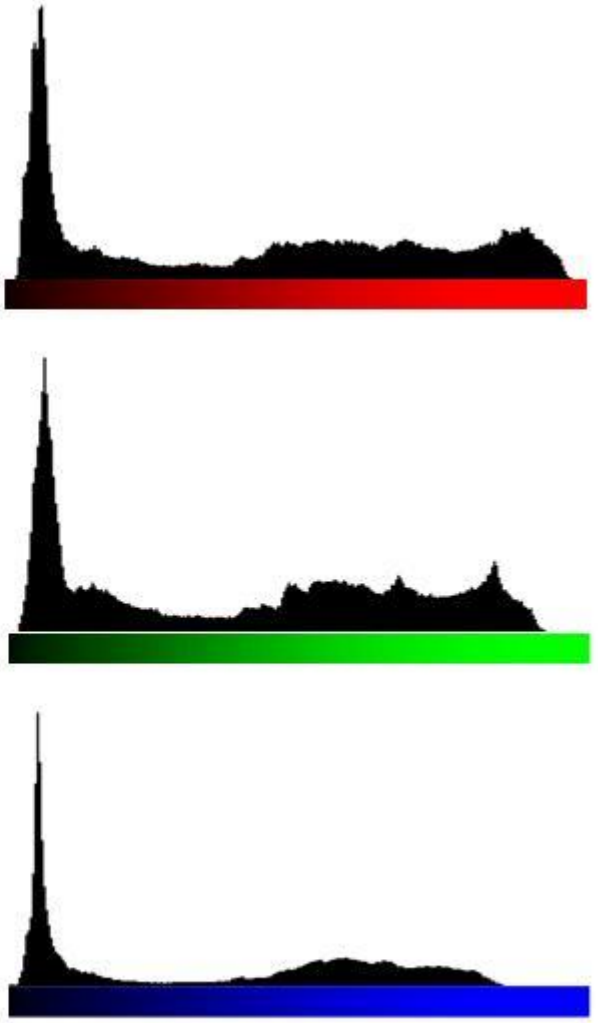
디지털 영상의 히스토그램

- 개별 밝기 값에 대한 발생 확률을 함수로 표현한 것
- 관찰한 데이터의 특징을 한눈에 알아볼 수 있도록 데이터를 막대그래프 모양으로 나타낸 것
- 디지털 영상에 대한 많은 정보를 제공함.



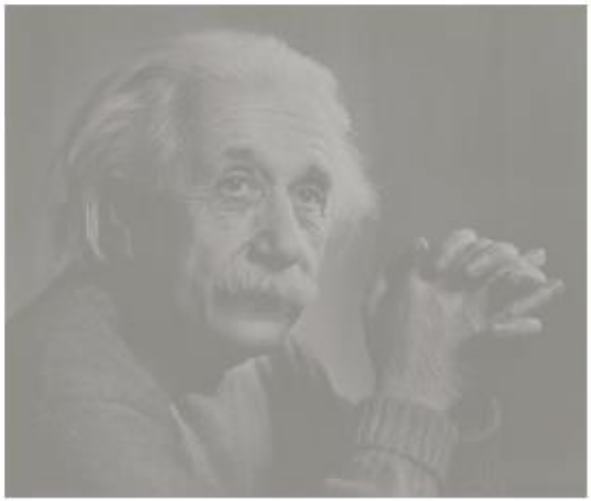
[그림 5-1] 이상적인 영상의 히스토그램

RGB 컬러 영상의 히스토그램

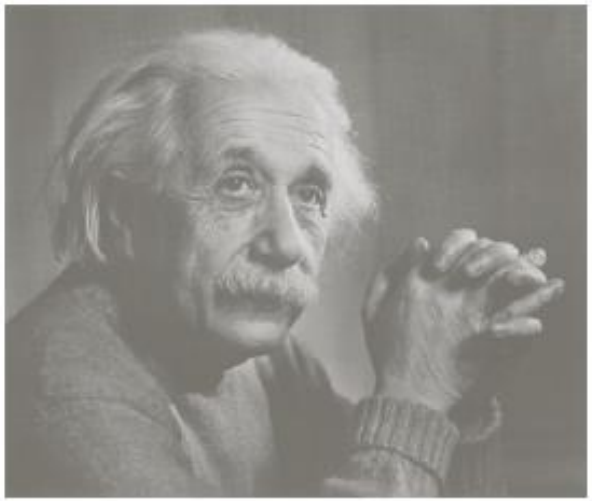


[RGB 컬러 히스토그램- 각 밴드 별로 밝기 분포 존재]

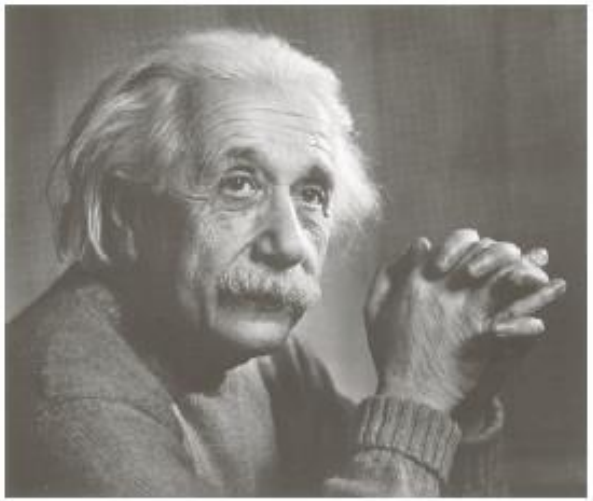
예시: Comparison of Standard Deviation Values



$\sigma = 14.3$

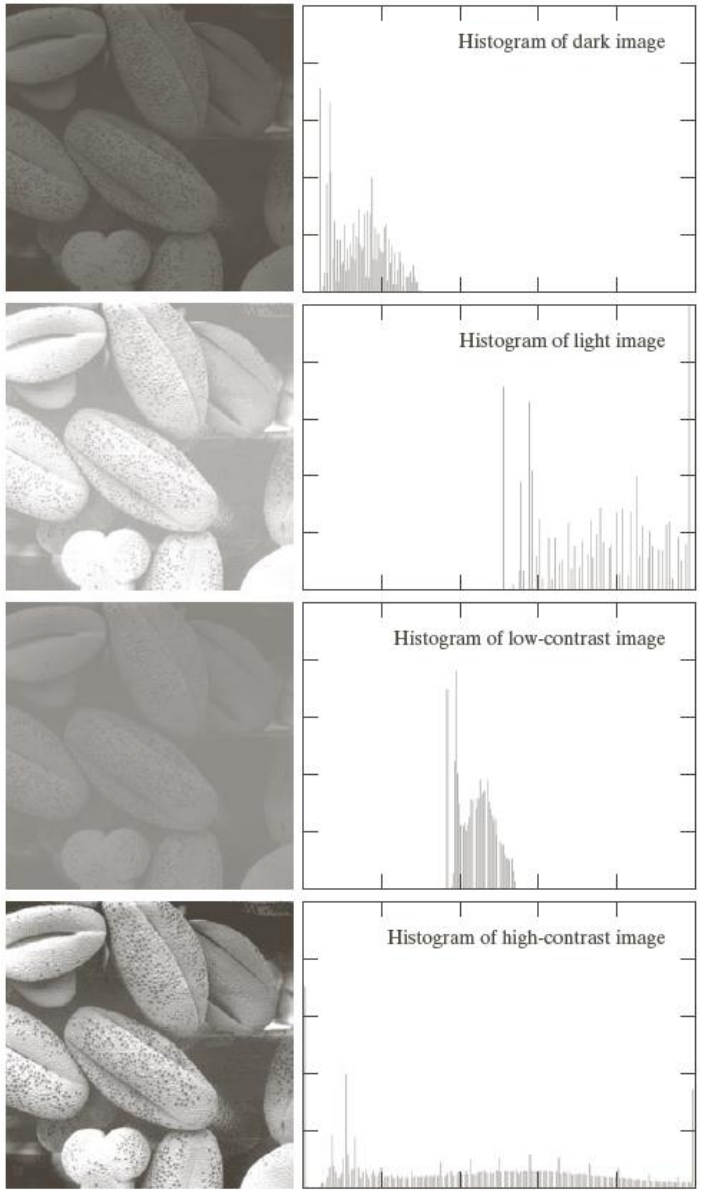


$\sigma = 31.6$



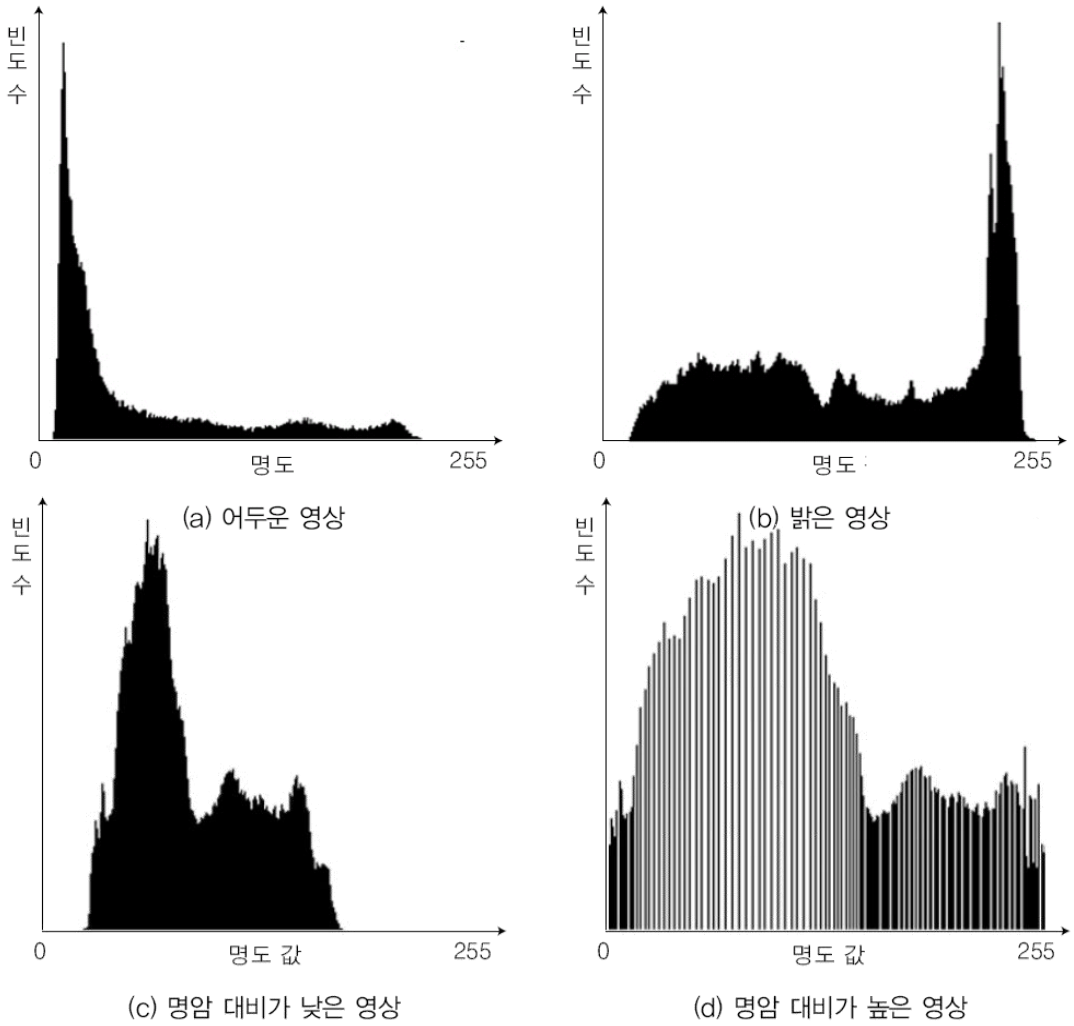
$\sigma = 49.2$

예시: Comparison of histogram shape (distribution)



Uniform distribution can give a good visual information.

영상의 특성에 따른 히스토그램



[영상 특성에 따른 히스토그램의 변화]

How to improve the visual quality? (품질 개선 방법)

- For low-contrasted image (as shown in the below)

Thresholding(이진화) ?

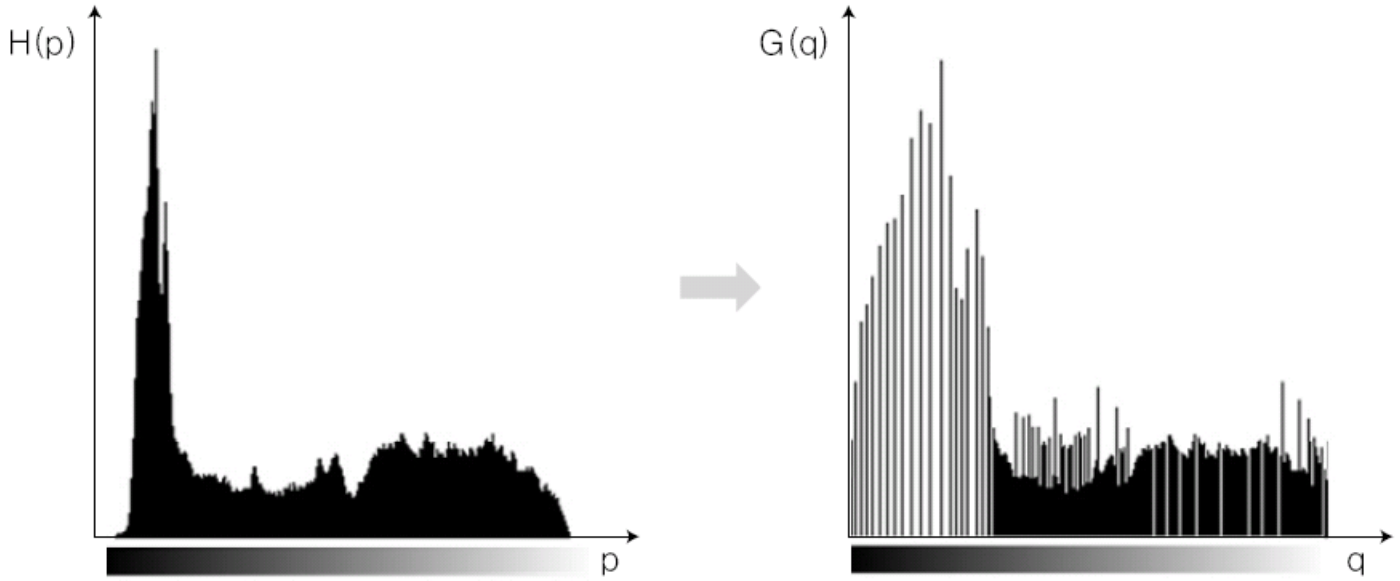
Enhancement(개선)

- Histogram Stretching
- Histogram equalization

? Thresholding(이진화)

Histogram Equalization (히스토그램 평활화)

- 히스토그램 평활화 기법(Histogram Equalization)
  - 명암 분포가 빈약한 영상을 균일하게 만들어 줌.
  - **영상의 밝기 분포를 재분배하여 명암 대비를 최대화**
  - 명암 대비 조정을 자동으로 수행
  - 검출 특성이 좋은 영상만 출력하지는 않지만 영상의 검출 특성을 증가시킴



[histogram equalization 후 균등화 된 Histogram shape]

Histogram Equalization(히스토그램 평활화)의 3단계

- 1단계
  - 명암 값 j의 빈도 수 hist[j]를 계산해 입력 영상의 히스토그램 생성
- 2단계
  - 각 명암 값 i에서 0~i까지의 누적 빈도 수(누적합)를 계산

$$sum[i] = \sum_{j=0}^{i} hist[j]$$

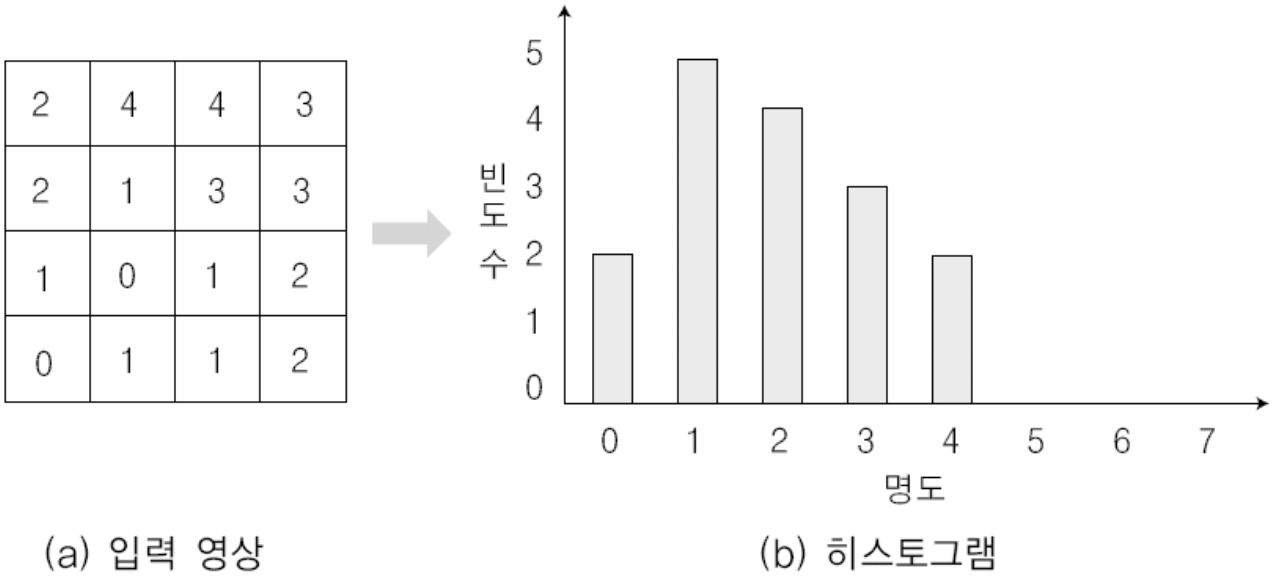
- 3단계
  - 2단계에서 구한 누적 빈도 수를 정규화(정규화 누적합)

$$n[i] = sum[i] \times \frac{1}{N} \times I_{\max}$$

    - N은 화소의 총 수, Imax는 최대 명도 값
    - 3단계에서 얻은 정규화된 값 n[i]로 입력 영상의 화소 값 i를 변환하면 평활화된 결과 영상 생성

Histogram Equalization (히스토그램 평활화): 1단계

- 1단계
  - 빈도 수 hist[j]에서의 히스토그램 생성

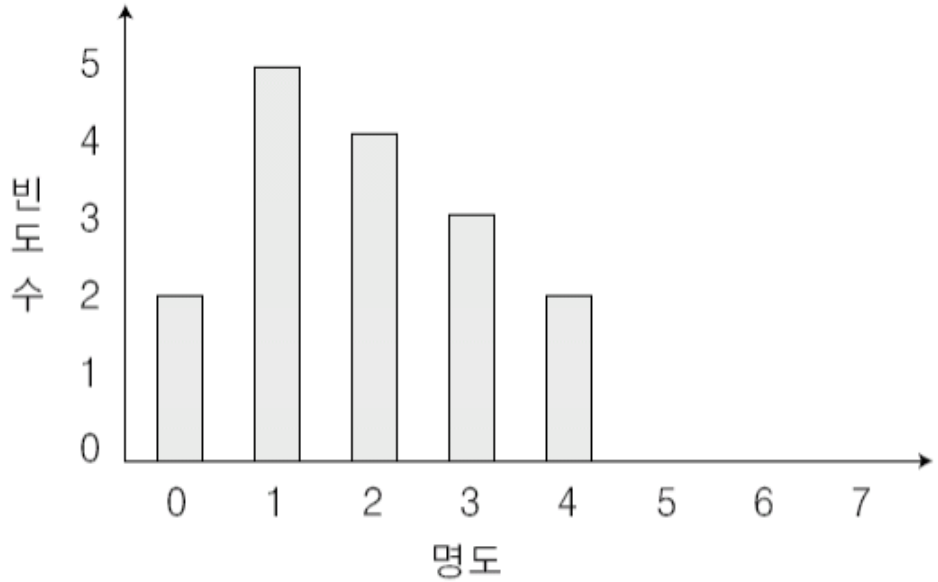


[histogram 생성]

- 화소의 명도 값 0은 2개, 1은 5개, 2는 4개, 3은 3개, 4는 2개
- 가장 큰 명도 값이 4이므로 전체적으로 왼쪽으로 치우침.

Histogram Equalization (히스토그램 평활화): 2단계

- 2단계
  - 누적합 sum[i] 생성



(a) 히스토그램

명도	누적합
0	2
1	7
2	11
3	14
4	16
5	16
6	16
7	16

(b) 누적합

[Histogram 누적 합 계산]

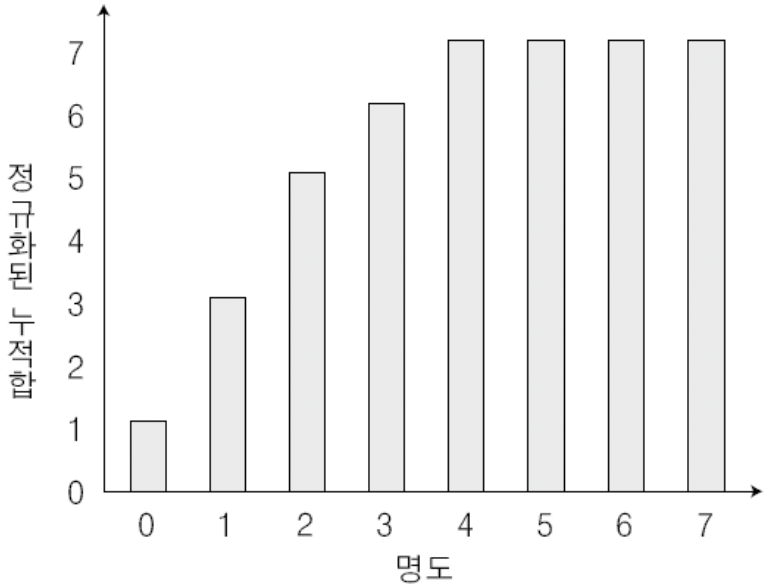
- 화소의 명도 0번까지의 누적합은 2, 1번까지는 2+5=7, 2번까지는 2+5+4=11, 3번까지는 2+5+4+3=14, 4번까지는 2+5+4+3+2=16
- 나머지 명도 값은 영상에는 없으므로 누적합은 16

Histogram Equalization (히스토그램 평활화): 3단계

- 3단계
  - n[i]=sum[i]*(1/16)*7

명도 (i)	누적합 (sum[i])	정규화된누적합 (n[i])
0	2	0.875
1	7	3.0625
2	11	4.8125
3	14	6.125
4	16	7
5	16	7
6	16	7
7	16	7

(a) 정규화



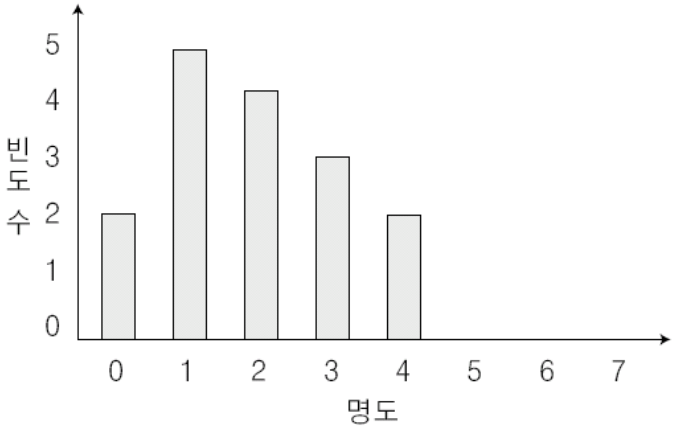
(b) 정규화된 히스토그램

[Histogram 누적 합에서의 정규화 과정]

- n[0]은 2*(1/16)*7=0.875, n[1]은 7*(1/16)*7=3.0625
- n[2]는 11*(1/16)*7=4.8125, n[3]은 14*(1/16)*7=6.125
- n[4]와 n[5], n[6], n[7]은 16*(1/16)*7=7

Histogram Equalization (히스토그램 평활화): 3단계

2	4	4	3
2	1	3	3
1	0	1	2
0	1	1	2

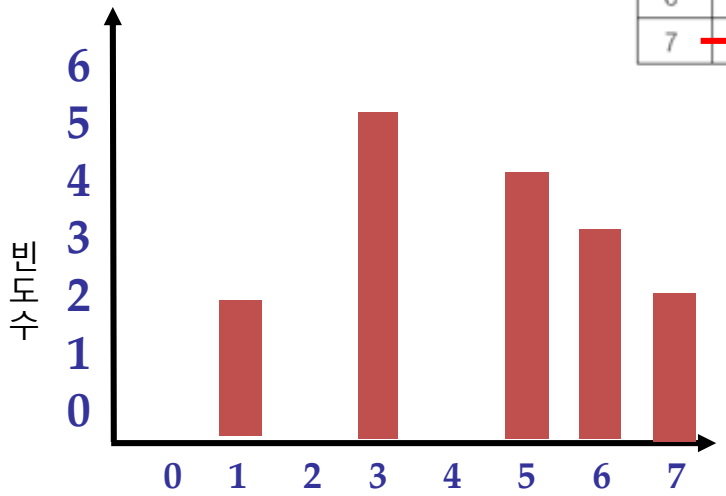


(a) 입력 영상

(b) 히스토그램

명도 (i)	누적합 (sum[i])	정규화된누적합 (n[i])	
0	2	0.875	1
1	7	3.0625	3
2	11	4.8125	5
3	14	6.125	6
4	16	7	7
5	16	7	7
6	16	7	7
7	16	7	7

5	7	7	6
5	3	6	6
3	1	3	5
1	3	3	5

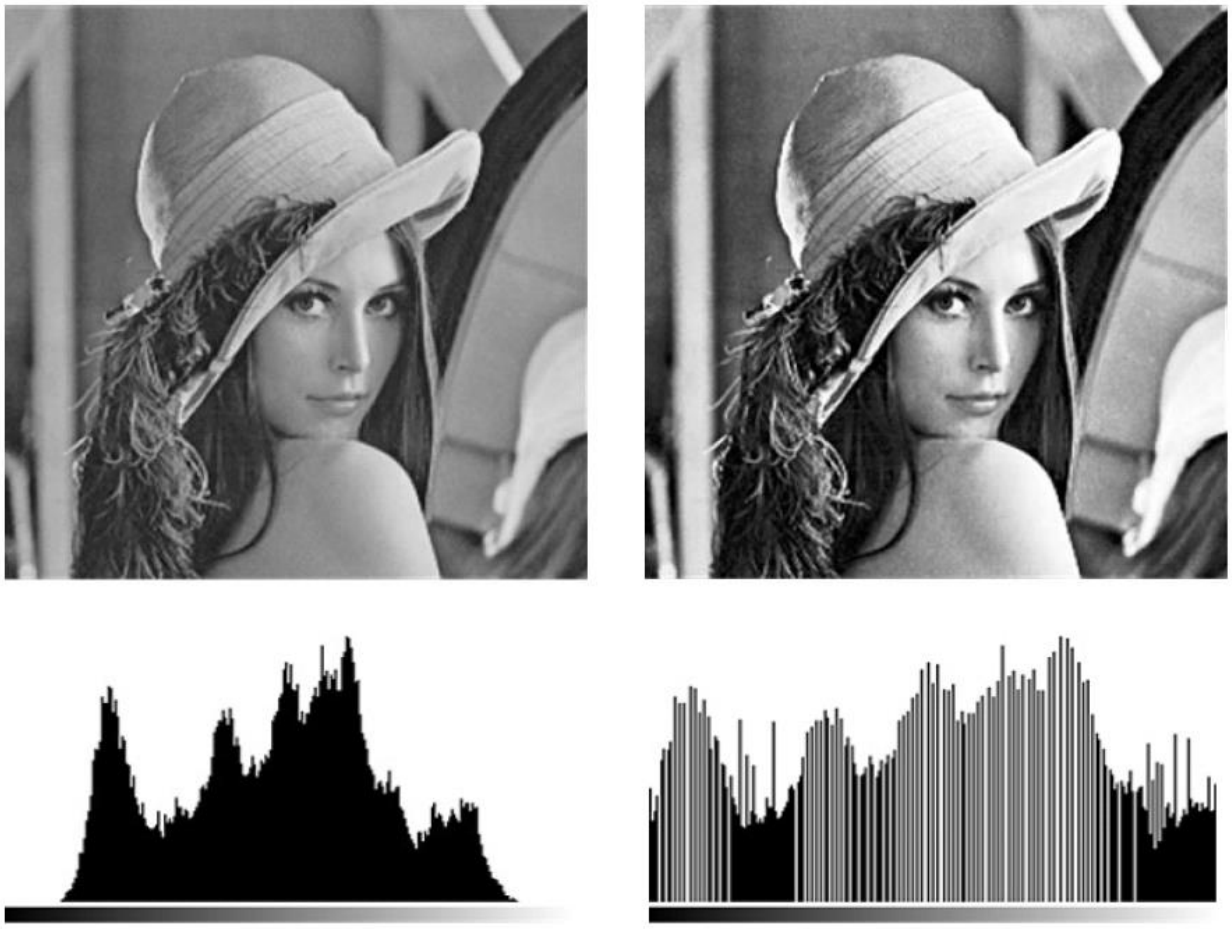


(a) 결과 영상

(b) 평활화 된 히스토그램

[Histogram 누적 합에서의 정규화 후 명도 맵핑을 통한 균등화된 Histogram 생성]

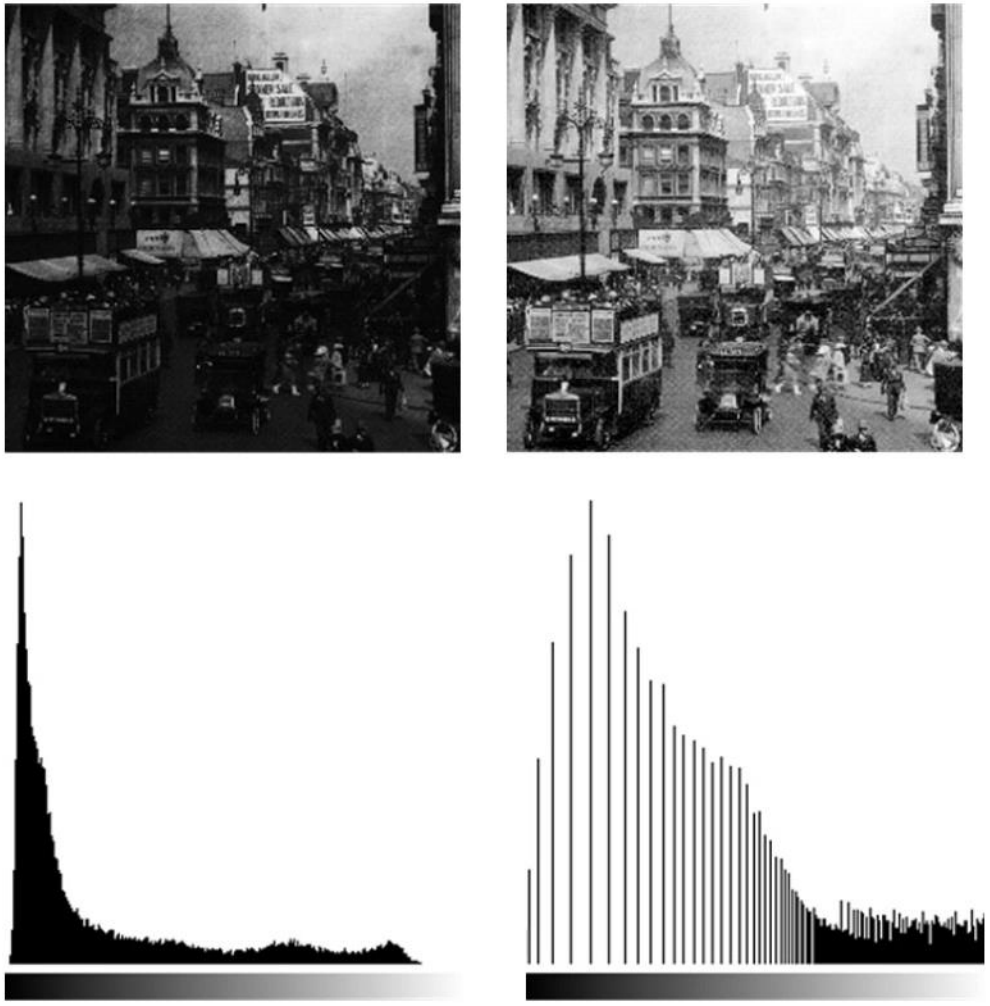
Histogram Equalization (히스토그램 평활화) 적용 결과



(a) 원본 영상 (b) 평활화 영상

[Histogram equalization을 통한 결과 영상 예시]

Histogram Equalization (히스토그램 평활화) 적용 결과(계속)



(a) 원본 영상 (b) 평활화 영상

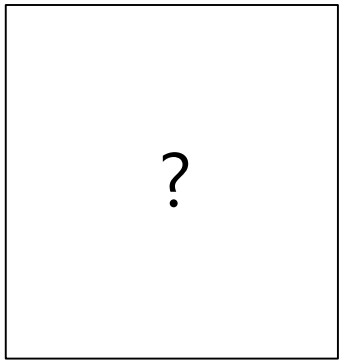
[Histogram equalization을 통한 결과 영상 예시]

One question???

- Histogram equalization (히스토그램 평활화)를 두 번 이상적용을 하게 되면, 그 결과는?

Hist. 평활화

Hist. 평활화



  - Histogram equalization is idempotent…!!

Histogram Specification(히스토그램 명세화)

Histogram Specification(히스토그램 명세화)

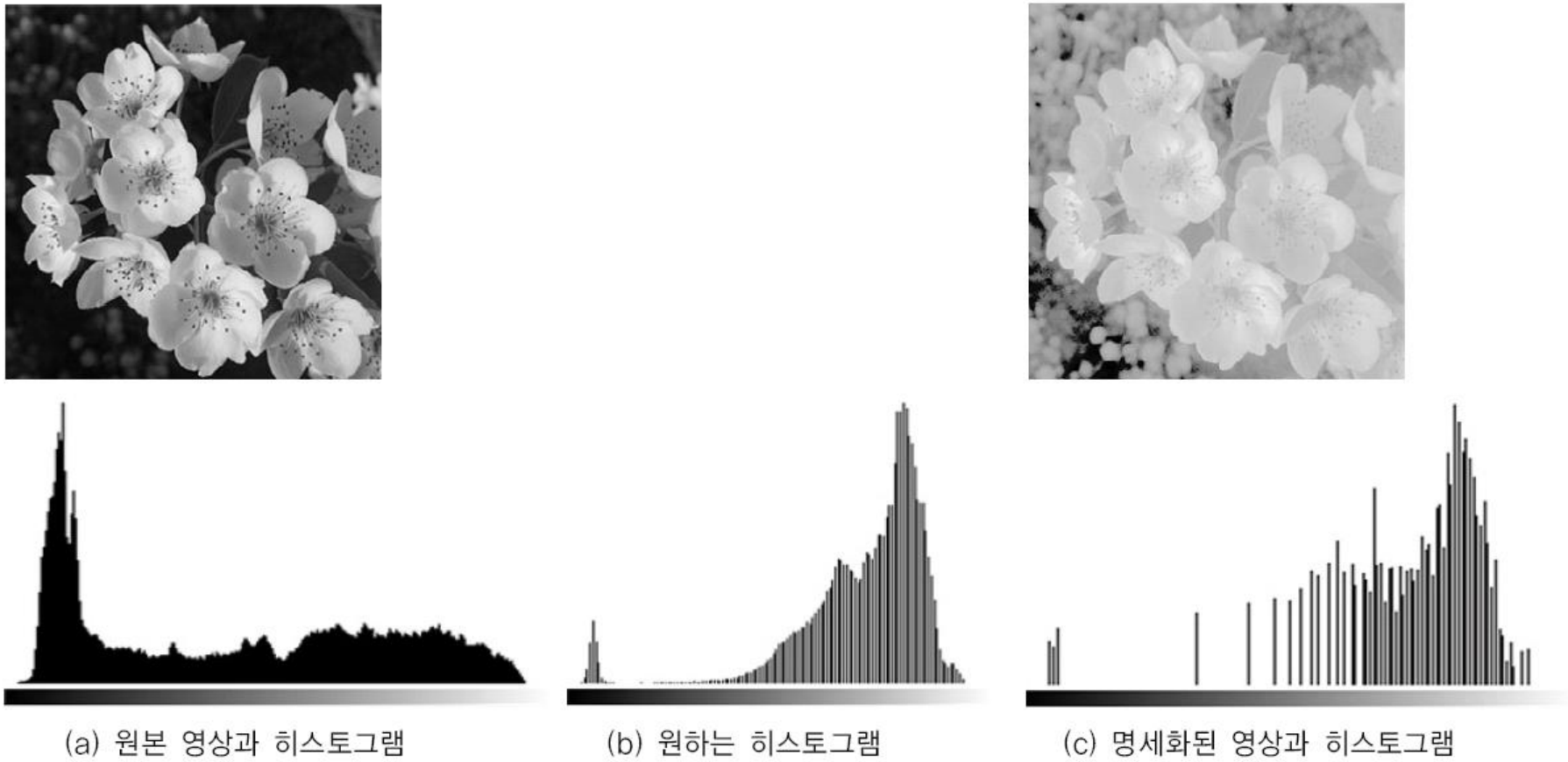
- 특정 모양의 히스토그램을 생성된 디지털 영상의 히스토그램에 포함하여 영상의 일부 영역의 명암 대비(콘트라스트)를 개선할 가능

- 입력 영상의 히스토그램을 원하는 히스토그램으로 변환한다고 해서 Histogram Matching (히스토그램 정합) 기법

- 명암 대비를 개선하는 것은 히스토그램 평활화와 같지만 특정 부분을 향상시키려고 원하는 히스토그램을 이용한 정합으로 일부 영역에서만 명암 대비를 개선한다는 점이 다름.

- 기본적으로 입력 영상을 원하는 히스토그램으로 평활화하고 역 히스토그램 평활화 수행 → 룩업테이블(lookup table)을 생성하고 평활화된 원 영상을 역 변환하여 원하는 히스토그램을 얻음.

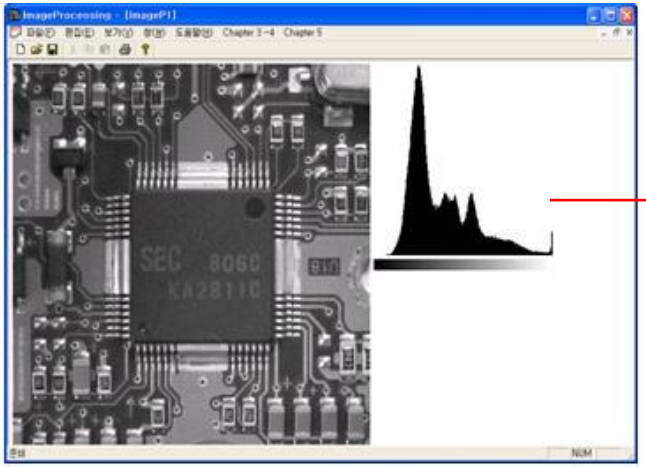
Histogram Specification(히스토그램 명세화) 개념



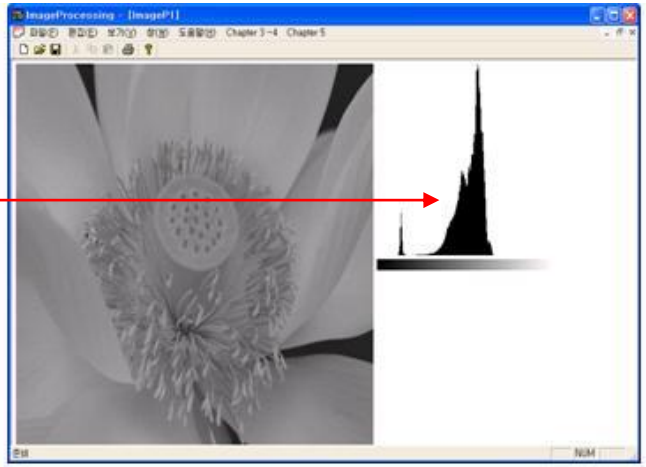
(a) 원본 영상과 히스토그램 (b) 원하는 히스토그램 (c) 명세화된 영상과 히스토그램

[Histogram Specification(히스토그램 명세화)의 개념]

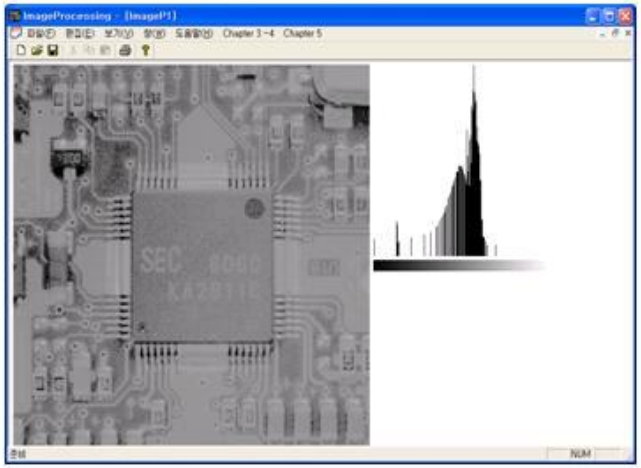
Histogram Specification(히스토그램 명세화) : 수행 결과 예시



[원본 영상 및 histogram]



[목적(target) 영상 및 histogram]



[명세화된 histogram 및 영상 결과]

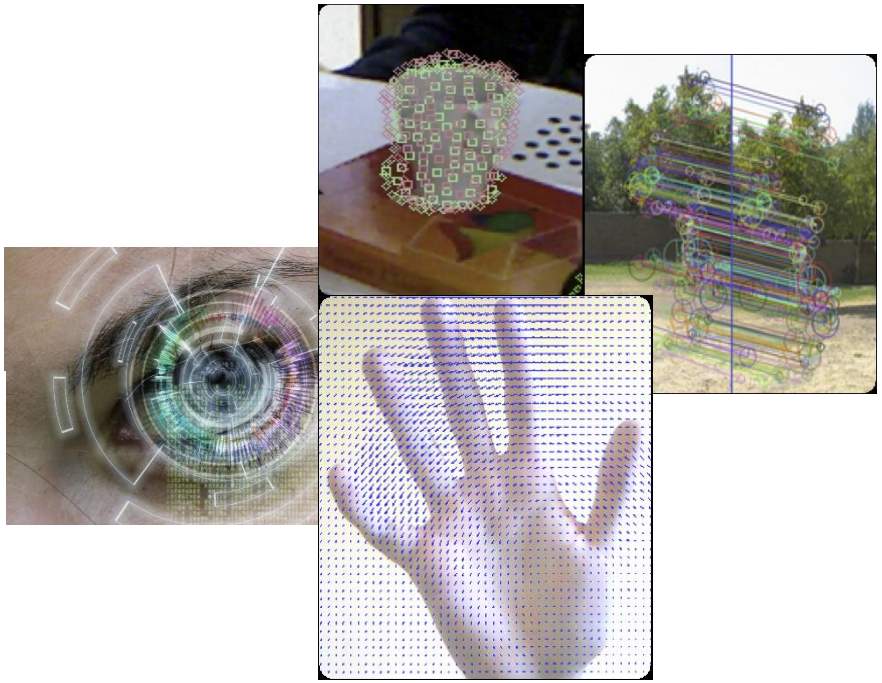
요 약

- 확률 및 확률 분포
  - 확률
  - 확률분포: 확률 변수가 취하는 범위와 해당 확률을 대응시켜 표현한 것
- 히스토그램
  - 영상의 밝기 분포
  - 밝기(x)에 대한 확률 분포(p(x))
- 히스토그램 처리
  - 생성된 히스토그램 정보를 기반으로 영상의 밝기 분포를 변화시키는 작업
  - 기본 처리기법
    - Equalization(평활화): 균일 분포(uniform distribution)로 펼쳐 주는 것
    - Specification(명세화): 원하는 분포 형태로 영상의 밝기는 변화시기는 것

IVPL
Intelligent Vision Processing Lab

2023, Fall

COMPUTER VISION 비젼 프로그래밍

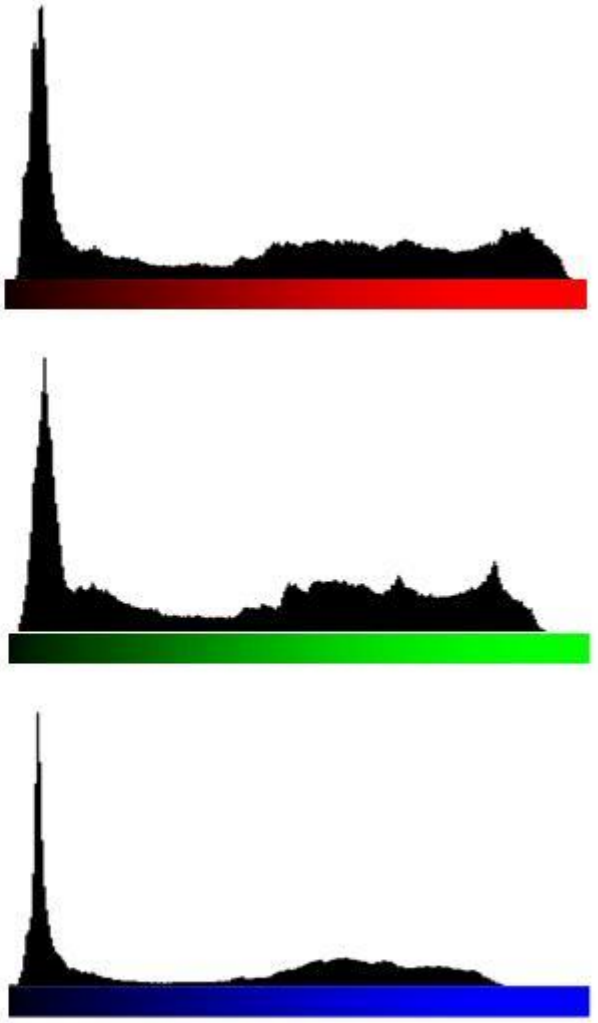


4장. 히스토그램 처리 (Histogram processing)-실습

영상에서의 확률변수 및 확률 분포: Histogram(히스토그램)

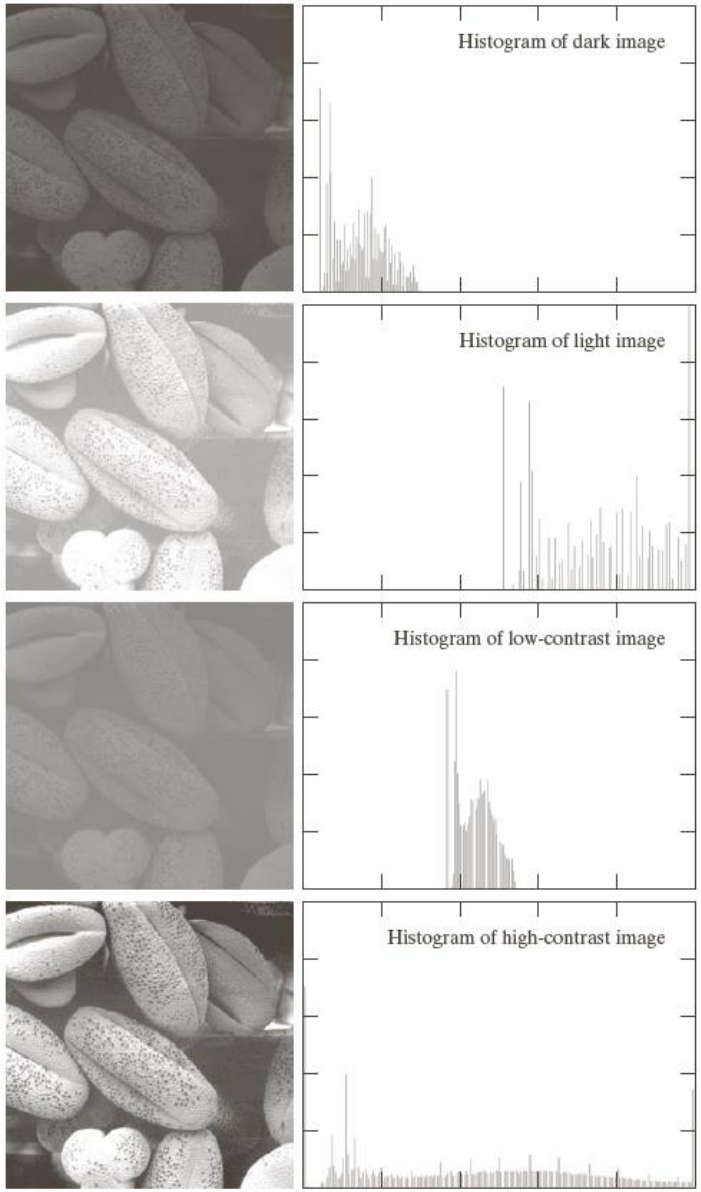
- 확률 변수(x): 밝기 값 ([0~255])
- 밝기(x)에 대한 확률 분포(p(x))
  - 개별 밝기 값에 해당하는 발생 확률을 함수로 표현 (그래프로 표현)

RGB 컬러 영상의 히스토그램



[RGB 컬러 히스토그램- 각 밴드 별로 밝기 분포 존재]

예시: Comparison of histogram shape (distribution)



Uniform distribution can give a good visual information.

Histogram processing(히스토그램 처리)-Practical Exercise:

- Histogram 처리 관련 API
  - **calcHist**
    - Calculates a histogram of a set of arrays.
    - void calcHist(const Mat **images**, int **nimages**, const int* **channels**, InputArray **mask**, OutputArray **hist**, int **dims**, const int* **histSize**, const float** **ranges**, bool **uniform**=true, bool **accumulate**=false)

Parameters:	•images – Source arrays. They all should have the same depth, CV_8U or CV_32F , and the same size. Each of the m can have an arbitrary number of channels. •nimages – Number of source images. •channels – List of the dims channels used to compute the histogram. The first array channels are numerated fro m 0 to images[0].channels()-1 , the second array channels are counted from images[0].channels() to images[0].ch annels() + images[1].channels()-1, and so on. •mask – Optional mask. If the matrix is not empty, it must be an 8-bit array of the same size as images[i] . The n on-zero mask elements mark the array elements counted in the histogram. •hist – Output histogram, which is a dense or sparse dims -dimensional array. •dims – Histogram dimensionality that must be positive and not greater than CV_MAX_DIMS (equal to 32 in the current OpenCV version). •histSize – Array of histogram sizes in each dimension. •ranges – Array of the dims arrays of the histogram bin boundaries in each dimension. When the histogram is un iform (uniform =true), then for each dimension i it is enough to specify the lower (inclusive) boundary of the 0-t h histogram bin and the upper (exclusive) boundary for the last histogram bin histSize[i]-1 . That is, in case of a uniform histogram each of ranges[i] is an array of 2 elements. When the histogram is not uniform (uniform=fals e), then each of ranges[i] contains histSize[i]+1 elements: . The array elements, that are not between and , are n ot counted in the histogram. •uniform – Flag indicating whether the histogram is uniform or not (see above). •accumulate – Accumulation flag. If it is set, the histogram is not cleared in the beginning when it is allocated. T his feature enables you to compute a single histogram from several sets of arrays, or to update the histogram in time.

Histogram processing(히스토그램 처리)-Practical Exercise :

- **compareHist**
  - Compares two histograms.
  - double compareHist(InputArray **H1**, InputArray **H2**, int **method**)

Parameters:

- **H1** – First compared histogram.
- **H2** – Second compared histogram of the same size as H1 .
- **method** – Comparison method that could be one of the following:
  - **CV_COMP_CORREL** Correlation
  - **CV_COMP_CHISQR** Chi-Square
  - **CV_COMP_INTERSECT** Intersection
  - **CV_COMP_BHATTACHARYYA** Bhattacharyya distance
  - **CV_COMP_HELLINGER** Synonym for CV_COMP_BHATTACHARYYA

Histogram processing(히스토그램 처리) -Practical Exercise:

- **equalizeHist**
  - Equalizes the histogram of a grayscale image.
  - void **equalizeHist**(InputArray **src**, OutputArray **dst)**

Parameters:

- **src** – Source 8-bit single channel image.
- **dst** – Destination image of the same size and type as src.

Histogram processing(히스토그램 처리) -Practical Exercise:

- Histogram 계산 및 화면에 보여주기
  - Histogram 계산 및 그래프 생성 클래스 구현

```
//--- 히스토그램 과련 클래스 구현 --//
class Histogram1D{
        private:
            int histSize[1]; // 히스토그램 빈도수
            float hranges[2]; // 히스토그램 최소/최대 화소값
            const float* ranges[1];
            int channels[1]; // 1채널만 사용

        public:
            Histogram1D(){ // 1차원 히스토그램을 위한 인자 준비
                histSize[0] = 256;
                hranges[0] = 0.0;
                hranges[1] = 255.0;
                ranges[0] = hranges;
                channels[0] = 0;
            }

            // 1차원 히스토그램 계산
            MatND getHistogram(const Mat &image){
                MatND hist;
                // 이미지의 히스토그램 계산
                calcHist(&image, 1, channels, Mat(), hist, 1, histSize, ranges);
                // 인자 값 : 이미지, 단일영상, 대상채널, 마스크 사용안함, 결과히스토그램,
                // 1차원 히스토그램, 빈도수, 화소값 범위
                return hist;
            }
    (계속)
```

Histogram processing(히스토그램 처리) -Practical Exercise:

(계속)

```
// 히스토그램을 위한 바 그래프 사용

    Mat getHistogramImage(const Mat &image){
        MatND hist = getHistogram(image); // 히스토그램 계산

        double maxVal = 0; // 최대 빈도수
        double minVal = 0; // 최소 빈도수
        minMaxLoc(hist, &minVal, &maxVal, 0, 0);

        // 히스토그램을 출력하기 위한 영상
        Mat histImg(histSize[0], histSize[0], CV_8U, Scalar(255));

        // nbins의 90%를 최대점으로 설정
        int hpt = static_cast<int>(0.9*histSize[0]);

        for (int h = 0; h < histSize[0]; h++){
            float binVal = hist.at<float>(h);
            int intensity = static_cast<int> ( binVal*hpt / maxVal);

            // 두 점간의 거리를 그림
            line(histImg, Point(h, histSize[0]), Point(h, histSize[0] - intensity), Scalar::all(0));
        }
        return histImg;
    }
};
```

Histogram processing(히스토그램 처리) -Practical Exercise:

- main 함수 구현

From documentation, the MatND actually is Mat

```
int main( int argc, char** argv )
{
  Mat image, dst;

  /// Load image
  image = imread( "test.jpg", 0 );   // Read the file  as grayscale image

   if (image.empty()){                              // Check for invalid input
   cout << "Could not open or find the image" << std::endl;
   return -1;
   }
   namedWindow("Display window", WINDOW_AUTOSIZE);  // Create a window for display.
   imshow("Display window", image );

  Histogram1D h; // 히스토그램을 위한 객체
 MatND histo = h.getHistogram(image); // 히스토그램 계산

    for (int i = 0; i < 256; i++) // 히스토그램의 빈도를 조회
        std::cout << "Value" << i << "=" << histo.at<float>(i) << std::endl;

    // 히스토그램을 영상으로 띄우기
    namedWindow("Histogram");
    imshow("Histogram", h.getHistogramImage(image));

    //영상을 두 그룹으로 나누는 부분을 경계값으로 처리해 확인
    Mat thresholdedImage; // 경계값으로 이진 영상 생성
    threshold(image, thresholdedImage, 60, 255, THRESH_BINARY);

    (계속)
```

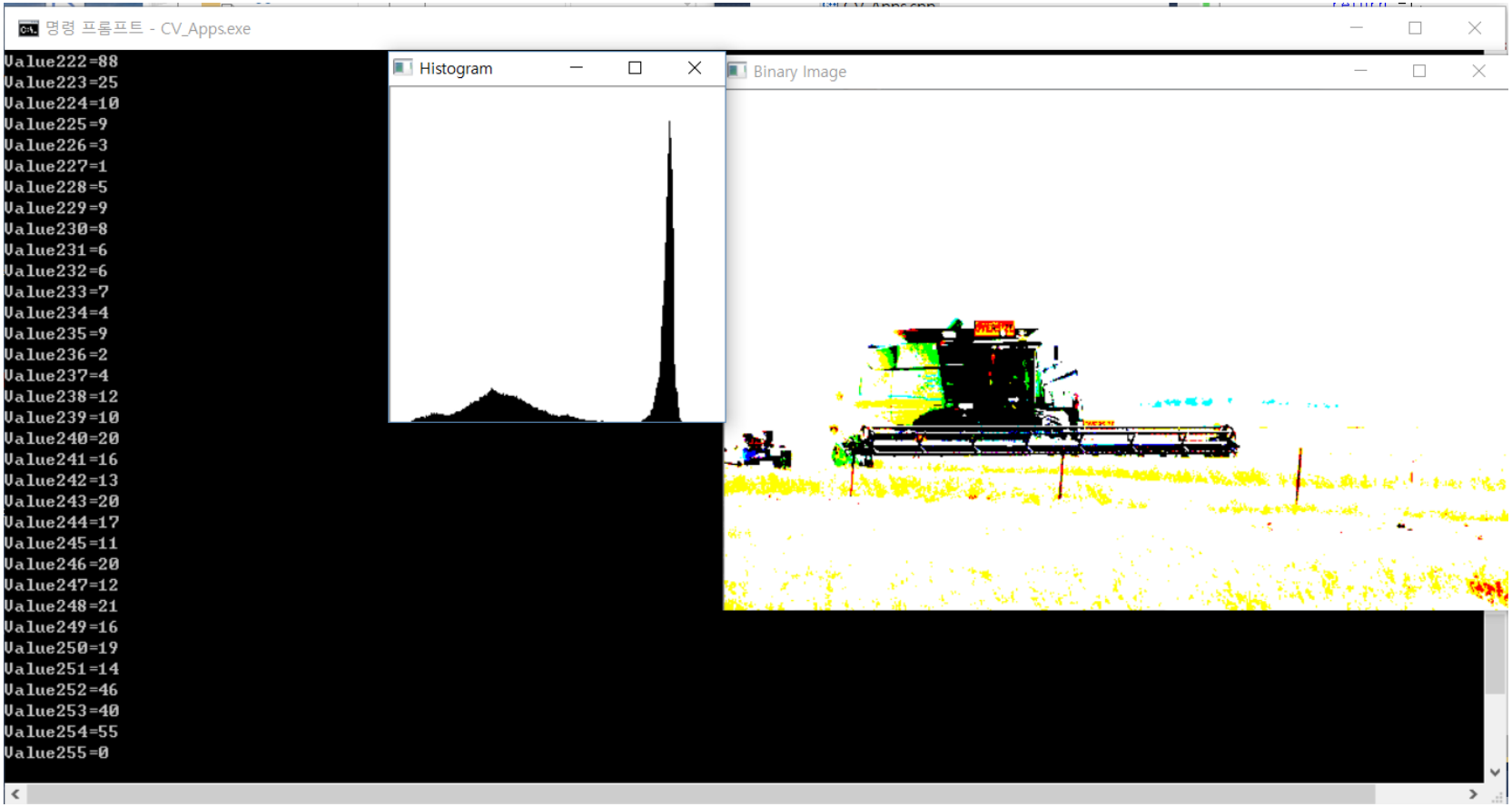
Histogram processing(히스토그램 처리) -Practical Exercise:

(계속)

```
// 영상을 경계화 하기 위해 히스토그램의 높은 봉우리(60) 방향으로 증가하기 직전인 최소값으로 정함
    namedWindow("Binary Image");
    imshow("Binary Image", thresholdedImage);


  waitKey(0);

  return 0;
}
```

Histogram processing(히스토그램 처리) -Practical Exercise:

- Histogram 계산 및 화면에 보여주기: 처리결과
  - 개별 bin의 값을 보여주고 화면에 히스토그램 출력, 이진영상 출력



Histogram processing(히스토그램 처리) -Practical Exercise:

- Histogram equalization (히스토그램 평활화)
  - Histogram 계산 및 그래프 생성 클래스 구현

```
//--- 히스토그램 과련 클래스 구현 --//
class Histogram1D{
private:
    int histSize[1]; // 히스토그램 빈도수
    float hranges[2]; // 히스토그램 최소/최대 화소값
    const float* ranges[1];
    int channels[1]; // 1채널만 사용

public:
    Histogram1D(){ // 1차원 히스토그램을 위한 인자 준비
        histSize[0] = 256;
        hranges[0] = 0.0;
        hranges[1] = 255.0;
        ranges[0] = hranges;
        channels[0] = 0;
    }

    // 1차원 히스토그램 계산
    MatND getHistogram(const Mat &image){
        MatND hist;
        // 이미지의 히스토그램 계산
        calcHist(&image, 1, channels, Mat(), hist, 1, histSize, ranges);
        // 인자 값 : 이미지, 단일영상, 대상채널, 마스크 사용안함, 결과히스토그램,
        // 1차원 히스토그램, 빈도수, 화소값 범위
        return hist;
    }
    (계속)
```

Histogram processing(히스토그램 처리) -Practical Exercise:

- Histogram equalization (히스토그램 평활화)
  - Histogram equalization 함수 구현

(계속)

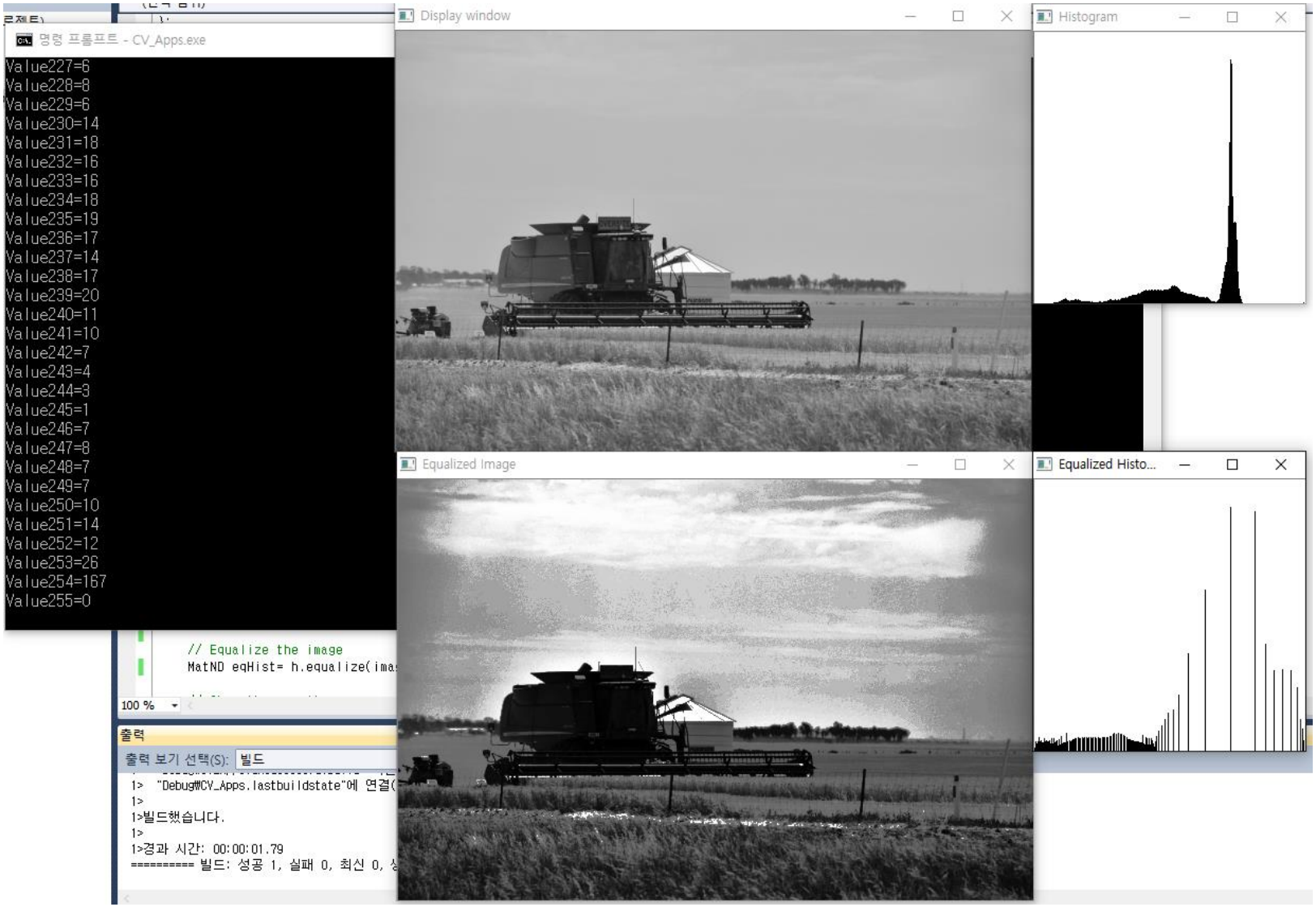
```
// Equalizes the source image.
MatND equalize(const cv::Mat &image) {

cv::Mat result;
cv::equalizeHist(image,result);

return result;
}
```

}; //클래스 선언 종료 부부

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

    Histogram1D h; // 히스토그램을 위한 객체

     ~~~~~~

     // Equalize the image
      MatND eqHist= h.equalize(image);

      // Show the result
      namedWindow("Equalized Image");
      imshow("Equalized Image",eqHist);

      // Show the new histogram
      namedWindow("Equalized Histogram");
      imshow("Equalized Histogram",
                  h.getHistogramImage(eqHist));

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

Histogram processing(히스토그램 처리) 실습:

- 수행 결과: 전 영역에 고루 분포된 영상으로 변환



Histogram processing(히스토그램 처리) -Practical Exercise:

Color Histogram 생성 및 처리

Color histogram 클래스에 draw 기능 추가

```
cv::MatND getHistogramCImage(const Mat &image){

    /// Separate the image in 3 places ( B, G and R )
    Mat bgr_planes[3];
    split( image, bgr_planes );

    /// Establish the number of bins
    int histSize = 256;

    /// Set the ranges ( for B,G,R) )
    float range[] = { 0, 256 } ;
    const float* histRange = { range };

    bool uniform = true; bool accumulate = false;

    Mat b_hist, g_hist, r_hist;

    /// Compute the histograms:
    calcHist( &bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize, &histRange, uniform, accumulate );
    calcHist( &bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize, &histRange, uniform, accumulate );
    calcHist( &bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize, &histRange, uniform, accumulate );

    // Draw the histograms for B, G and R
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound( (double) hist_w/histSize );
```

Histogram processing(히스토그램 처리) -Practical Exercise:

(계속)

```
    Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 255,255,255) );

    /// Normalize the result to [ 0, histImage.rows ]
    normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
    normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
    normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );

    /// Draw for each channel
    for( int i = 1; i < histSize; i++ )
    {
        line( histImage, Point( bin_w*(i-1), hist_h - cvRound(b_hist.at<float>(i-1)) ) ,
         Point( bin_w*(i), hist_h - cvRound(b_hist.at<float>(i)) ),
         Scalar( 255, 0, 0), 2, 8, 0  );
        line( histImage, Point( bin_w*(i-1), hist_h - cvRound(g_hist.at<float>(i-1)) ) ,
         Point( bin_w*(i), hist_h - cvRound(g_hist.at<float>(i)) ),
         Scalar( 0, 255, 0), 2, 8, 0  );
        line( histImage, Point( bin_w*(i-1), hist_h - cvRound(r_hist.at<float>(i-1)) ) ,
         Point( bin_w*(i), hist_h - cvRound(r_hist.at<float>(i)) ),
         Scalar( 0, 0, 255), 2, 8, 0  );
    }

    return histImage;
}
```

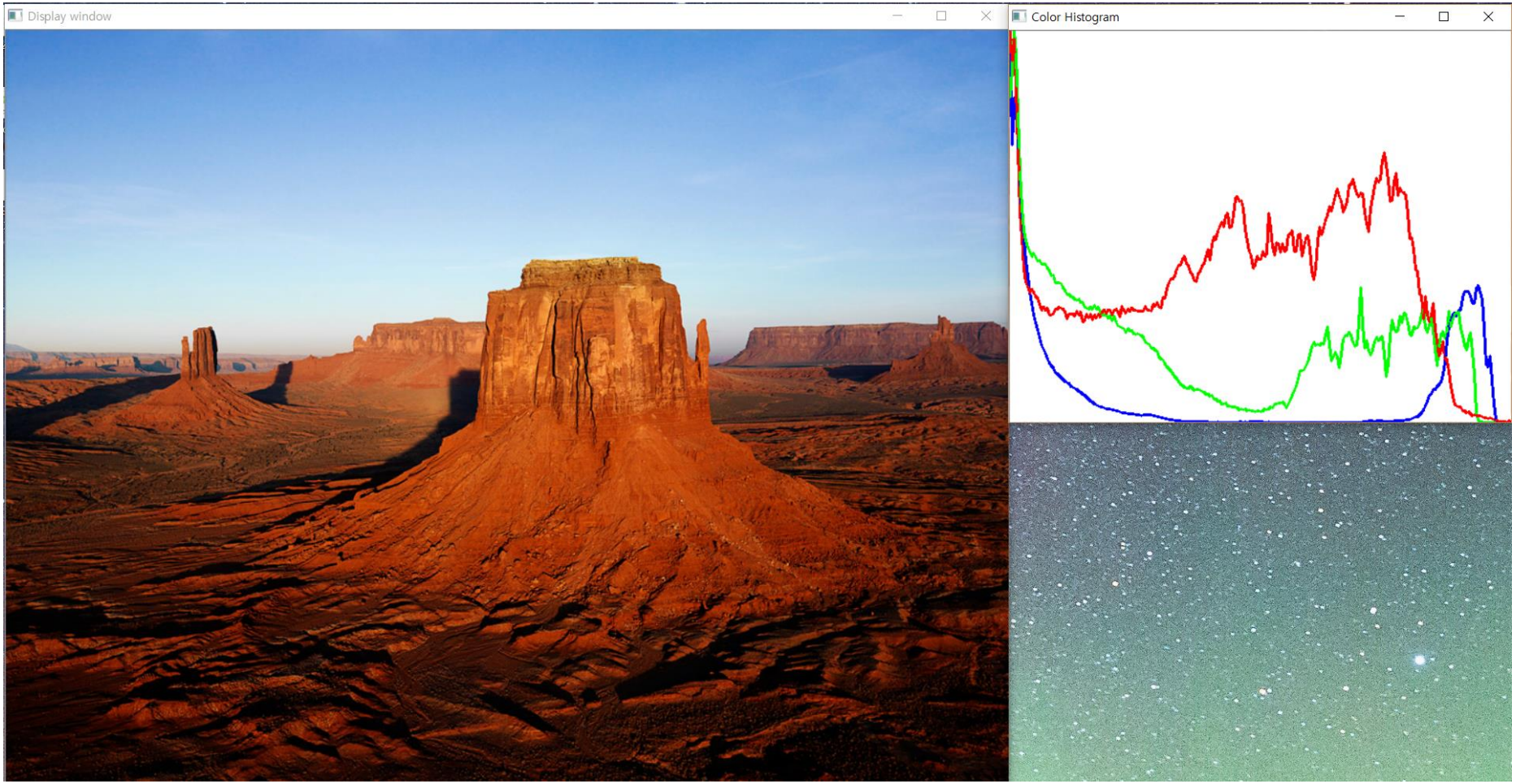
Histogram processing(히스토그램 처리) -Practical Exercise:

- main 함수 내 실행 코드

```
int main( int argc, char** argv )
{
        Mat image, dst;

        /// Load image
        image = imread( "Desert.bmp", 1);    // Read the file

         if (image.empty()){                              // Check for invalid input
         cout << "Could not open or find the image" << std::endl;
         return -1;
         }
         namedWindow("Display window", WINDOW_AUTOSIZE);  // Create a window for display.
         imshow("Display window", image );

        ColorHistogram h;

         MatND histo = h.getHistogram(image); // 히스토그램 계산
         MatND Chist = h.getHistogramCImage(image);

        // 히스토그램을 영상으로 띄우기
        namedWindow("Color Histogram");
        imshow("Color Histogram", Chist);

        waitKey(0);

       return 0;
}
```

Histogram processing(히스토그램 처리) -Practical Exercise:

- 수행 결과



Histogram processing(히스토그램 처리) -Practical Exercise:

- 예제 코드 이해
  - 기본적으로 개별 컬러 밴드를 각각 처리하여 다시 합쳐 주는 방식

```
int main( int argc, char** argv )
{
        Mat image, dst;

    /// Load image
     image = imread( "Desert.bmp", 1);    // Read the file

      if (image.empty()){                                  // Check for invalid input
       cout << "Could not open or find the image" << std::endl;
       return -1;
       }
       namedWindow("Display window", WINDOW_AUTOSIZE);
       imshow("Display window", image );

        //--컬러 영상 평활화 --//
        Mat channels[3];

        // 개별 채널 분리
        split(image,channels);
```

Histogram processing(히스토그램 처리) -Practical Exercise:

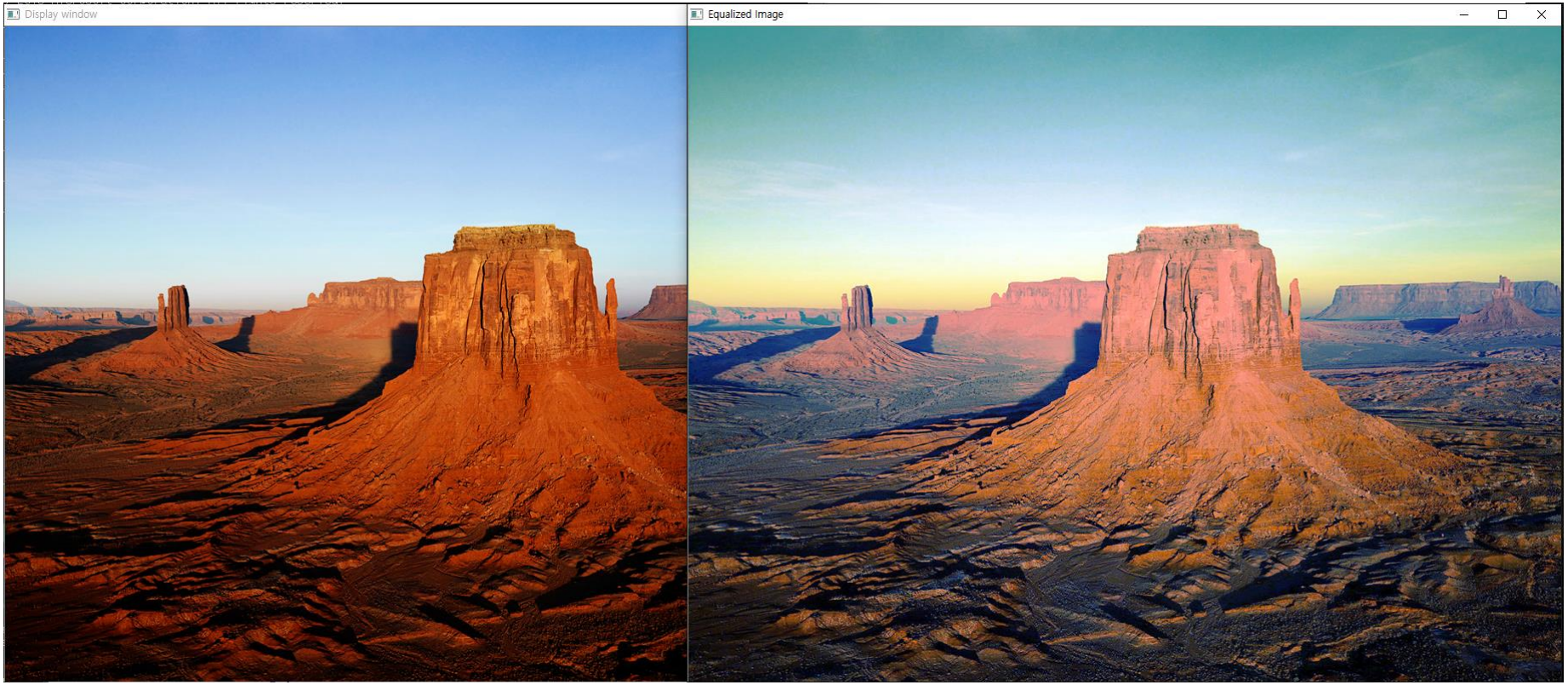
- 예제 코드 이해

(계속)

```
//equalize histogram on the each channel
equalizeHist(channels[0], channels[0]);
equalizeHist(channels[1], channels[1]);
equalizeHist(channels[2], channels[2]);

//merge 3 channels including the modified 1st channel into one image
merge(channels, 3, image);

namedWindow("Equalized Image", WINDOW_AUTOSIZE);
                                        // Create a window for display.
imshow("Equalized Image", image );


waitKey(0);

return 0;
}
```

Histogram processing(히스토그램 처리) -Practical Exercise:

- 수행 결과: 컬러 평활화가 수행 됨..... 조금 색상이 변화가 있죠???



Histogram processing(히스토그램 처리) -Practical Exercise:

사용 API 확인

- **split()**
  - Divides a multi-channel array into several single-channel arrays.
- void split(const Mat& **src**, Mat* **mvbegin**) 또는 void split(InputArray **m**, OutputArrayOfArrays **mv**)

Parameters:	•src – input multi-channel array. •mv – output array or vector of arrays

- **cvtColor()**
  - Convert color space.
- void cvtColor(putArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)
  - RGB -> GRAY (CV_BGR2GRAY, CV_RGB2GRAY, CV_GRAY2BGR, CV_GRAY2RGB)
  - RGB -> YCrCb JPEG (or YCC) (CV_BGR2YCrCb, CV_RGB2YCrCb, CV_YCrCb2BGR, CV_YCrCb2RGB)
  - RGB->HSV (CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB)
  - RGB->CIE XYZ.Rec 709 with D65 white point (CV_BGR2XYZ, CV_RGB2XYZ, CV_XYZ2BGR, CV_XYZ2RGB)

Histogram processing(히스토그램 처리) -Practical Exercise:

- **merge()**
  - Creates one multichannel array out of several single-channel ones.
- void merge(const Mat* **mv**, size_t **count**, OutputArray **dst**)

Parameters:	•**mv** – input array or vector of matrices to be merged; all the matrices in mv must have the same size and the same depth. •**count** – number of input matrices when mv is a plain C array; it mus t be greater than zero. •**dst** – output array of the same size and the same depth as mv[0];

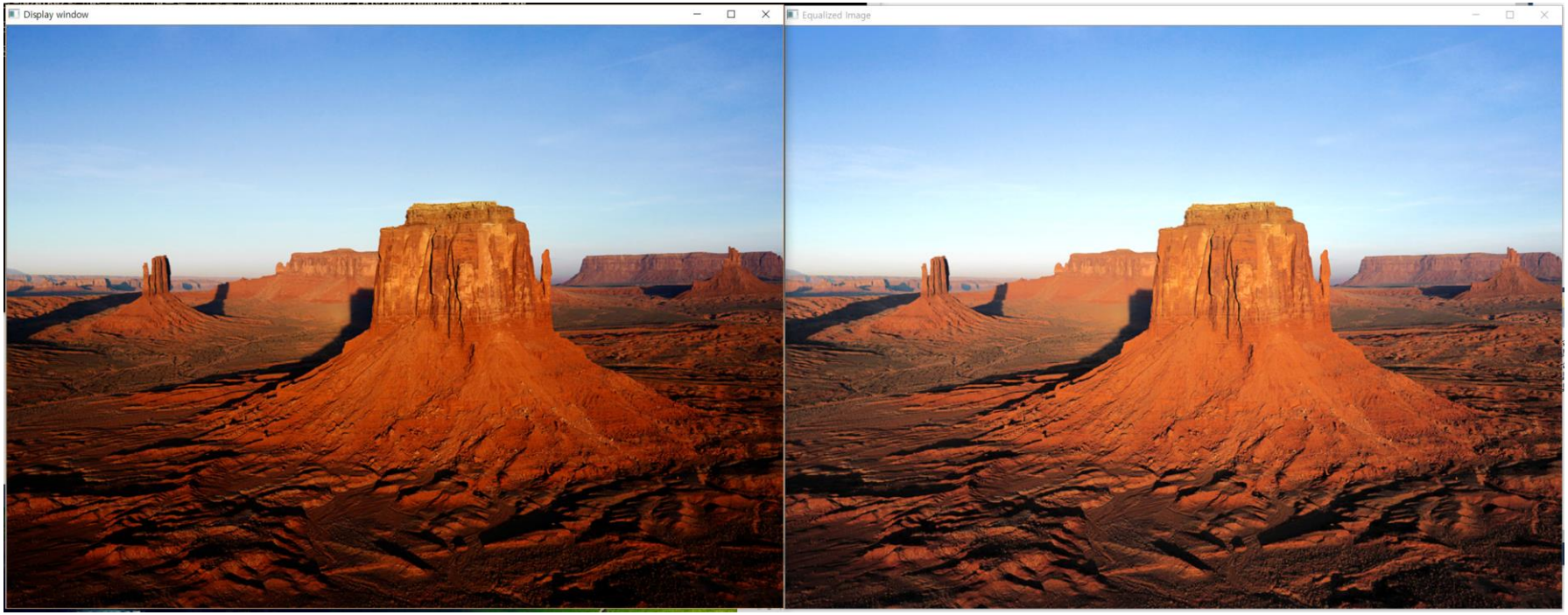
Histogram processing(히스토그램 처리) -Practical Exercise:

- 일반적인 컬러 histogram equalization
  - 개별 밴드 별로 수행하지 않고 YCbCr로 변환 후 **Y성분만 평활화** 한다.

```
int main( int argc, char** argv )
{
        Mat image, dst;

        /// Load image
        image = imread( "Desert.bmp", 1);   // Read the file
          (중략) ~~~
      //--컬러 영상 평활화 --//
      Mat channels[3];

      //컬러변환
      cvtColor(image, dst, CV_BGR2YCrCb);

      // 개별 채널 분리
      split(dst,channels);

      //equalize histogram on the 1st channel (Y)
      equalizeHist(channels[0], channels[0]);

      //merge 3 channels including the modified 1st channel into one image
      merge(channels, 3, dst);

      //컬러변환 to BGR
      cvtColor(dst, image, CV_YCrCb2BGR);

      namedWindow("Equalized Image", WINDOW_AUTOSIZE);  // Create a window for display.
      imshow("Equalized Image", image );

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

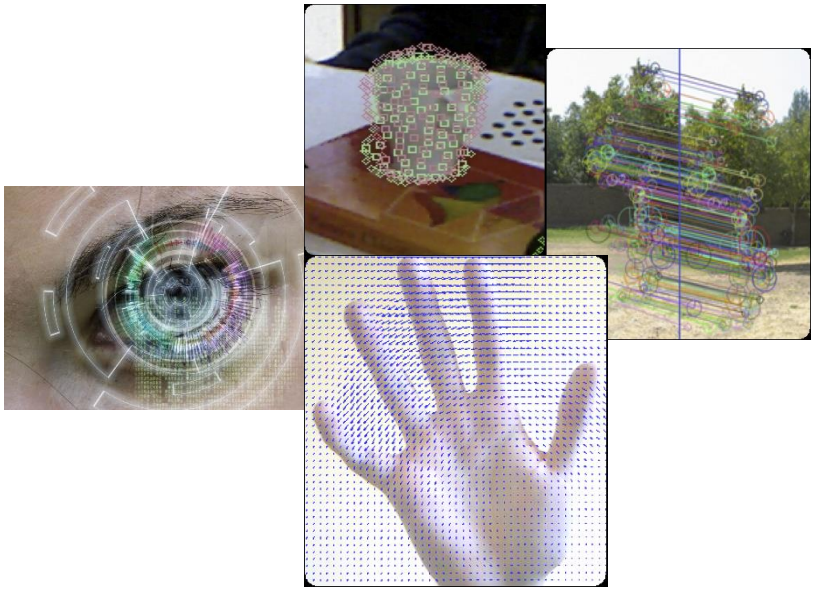
Histogram processing(히스토그램 처리) -Practical Exercise:

- 수행 결과: 컬러 평활화가 정상적으로 수행 됨



COMPUTER VISION 비젼 프로그래밍

Thank you and Question?



IVPL
Intelligent Vision Processing Lab