디지털 영상처리

화소처리

학습목표

- 6.1 영상 화소의 접근
- 6.2 화소 밝기 변환
- 6.3 히스토그램
- 6.4 컬러 공간 변환

Section 01 화소 점 처리의 개념

■ 화소 점 처리

- 원 화소의 값이나 위치를 바탕으로 단일 화소 값을 변경하는 기술
- 다른 화소의 영향을 받지 않고 단순히 화소 점의 값만 변경하므로 포인트 처리(Point Processing)라고도 함
- 산술연산, 논리연산, 반전, 광도 보정, 히스토그램 평활화, 명암 대비 스트레칭 등의 기법이 있음
- 디지털 영상의 산술연산은 디지털 영상의 각 화소 값에서 임의의 상수 값으로 덧셈, 뺄셈, 곱셈, 나눗셈을 수행하는 것
- 그레이 레벨 영상에서 화소 값이 작으면 영상이 어둡고, 화소의 값이 크면 밝음

119	119	121	121	130	139	114	71	74	74	73
119	121	121	119	129	139	114	72	75	75	74
120	120	119	119	128	139	112	72	75	76	75
121	120	119	122	130	138	109	73	76	77	76
120	121	119	122	133	136	105	71	78	77	76
121	121	118	122	133	133	102	72	77	76	77
122	122	117	123	132	133	101	70	75	77	79
123	123	117	122	133	135	99	70	77	78	80
122	123	117	123	133	135	97	70	76	77	79
121	123	118	123	131	133	95	70	75	77	79
120	122	117	122	131	133	94	72	76	78	78

Section 01 화소 점 처리의 개념

■ 8비트 그레이 레벨 영상 데이터의 밝기 정도

화소의 십진수 값	화소의 이진수 값	표현되는 밝기
0	0000 0000	검정색
⋮	⋮	⋮
63	0011 1111	어두운 회색
⋮	⋮	⋮
127	0111 1111	회색
⋮	⋮	⋮
191	1011 1111	밝은 회색
⋮	⋮	⋮
255	1111 1111	흰색

6.1 영상 화소의 접근

- 영상처리
  - 2차원 데이터에 대한 행렬 연산

- 영상처리 프로그래밍을 한다는 것
  - 영상이라는 2차원 데이터의 원소값을 개발자가 원하는 방향으로 변경하는 것
  - 영상을 다루려면 기본적으로 영상의 화소 접근, 값 수정, 새로 만들 수 있어야 함

- 6.1.1 화소(행렬 원소) 접근

6.1 영상 화소의 접근

예제 6.1.1 행렬 원소 접근 방법 - 01.mat_access.py

```
01  import numpy as np
02
03  def mat_access1(mat):                          # 원소 직접 접근 방법
04      for i in range(mat.shape[0]):
05          for j in range(mat.shape[1]):
06              k = mat[i, j]                      # 원소 접근– mat[i][j] 방식도 가능
07              mat[i, j] = k * 2                  # 원소 할당
08
09  def mat_access2(mat):                          # item() , itemset() 함수 사용방식
10      for i in range(mat.shape[0]):
11          for j in range(mat.shape[1]):
12              k = mat.item(i, j)                 # 원소 접근
13              mat.itemset((i, j), k * 2)         # 원소 할당
14
15  mat1 = np.arange(10).reshape(2, 5)             # 0~10 사이 원소 생성
16  mat2 = np.arange(10).reshape(2, 5)
17
18  print("원소 처리 전: \n%s\n" % mat1)
19  mat_access1(mat1)
20  print("원소 처리 후: \n%s\n" % mat1)
21
22  print("원소 처리 전: \n%s\n" % mat2)
23  mat_access2(mat2)
24  print("원소 처리 후: \n%s" % mat2)
```

6.1 영상 화소의 접근

예제 6.1.1 행렬 원소 접근 방법 - 01.mat_access.py

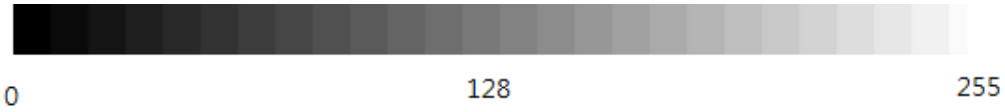
```
01  import numpy as np
02
03  def mat_access1(mat):                          # 원소 직접 접근 방법
04      for i in range(mat.shape[0]):
05          for j in range(mat.shape[1]):
06              k = mat[i, j]                      # 원소 접근– mat[i][j] 방식도 가능
07              mat[i, j] = k * 2                  # 원소 할당
08
09  def mat_access2(mat):                          # item() , itemset() 함수 사용방식
10      for i in range(mat.shape[0]):
11          for j in range(mat.shape[1]):
12              k = mat.item(i, j)                 # 원소 접근
13              mat.itemset((i, j), k * 2)         # 원소 할당
14
15  mat1 = np.arange(10).reshape(2, 5)             # 0~10 사이 원소 생성
16  mat2 = np.arange(10).reshape(2, 5)
17
18  print("원소 처리 전: \n%s\n" % mat1)
19  mat_access1(mat1)
20  print("원소 처리 후: \n%s\n" % mat1)
21
22  print("원소 처리 전: \n%s\n" % mat2)
23  mat_access2(mat2)
24  print("원소 처리 후: \n%s" % mat2)
```

```
Run:  01.mat_access
  D:/source/chap06/01.mat_access.py
원소 처리 전:
[[0 1 2 3 4]
 [5 6 7 8 9]]

원소 처리 후:
[[ 0  2  4  6  8]
 [10 12 14 16 18]]

원소 처리 전:
[[0 1 2 3 4]
 [5 6 7 8 9]]

원소 처리 후:
[[ 0  2  4  6  8]
 [10 12 14 16 18]]
```

6.2.1 그레이 스케일(명암도) 영상

- 흑백 영상 ?
  - 단어 자체의 의미: 검은색과 흰색의 영상, 의미 안맞음

- 그레이 스케일(gray-scale) 영상 , 명암도 영상
  - 화소값은 0~255의 값을 가지는데 0은 검은색을, 255는 흰색을 의미
  - 0~255 사이의 값들은 다음과 같이 진한 회색에서 연한 회색



디지털 영상의 산술연산

■ 화소의 밝기 값

- 밝기의 단계 수는 화소를 표현하는 양자화 비트 수가 결정
- 그레이 레벨 영상에서는 색은 없고 밝기만 있음
- 보통, 화소는 밝기를 나타내는데, 주로 양자화 비트 수를 8비트로 표현

■ 명암 대비

- 대비(Contrast): 영상 내에 있는 가장 밝은 값과 가장 어두운 값의 차이로, 영상의 품질을 결정하는 중요한 요소임
- 높은 대비를 보이는 디지털 영상: 어두운 명도와 밝은 명도의 차이가 너무 커서 시각적으로 좀더 명확하게 보임
- 낮은 대비를 보이는 디지털 영상: 밝기의 차이가 크지 않아 시각적으로 명확하지 못함

디지털 영상의 산술연산

- 화소 값의 덧셈연산
- 화소의 밝기 값에 특정한 상수 값을 더해 화소의 밝기 값을 증가시켜 영상을 밝게 하는 처리 기술

> 화소 + α : 영상의 밝기 증가 = 밝아짐

- 화소의 값에 임의의 상수를 더할 때 화소의 최대값을 넘기도 함
- 최대값인 255를 넘는 값은 모두 255로 처리

> (화소 값 + α) 〉 255이면, (화소 값 + α) = 255

디지털 영상의 산술연산

(a) 원본 영상

(b) 상수 값 10을 더한 영상

(c) 상수 값 50을 더한 영상

(d) 상수 값 100을 더한 영상

[그림 4-5] 덧셈 상수의 변화에 따른 디지털 영상의 밝기 증가

6.2.2 영상의 화소 표현

- modulo 방식과 saturation 방식 비교

예제 6.2.3 행렬 가감 연산 통한 영상 밝기 변경 – 04.bright_dark.py

```
01  import cv2
02
03  image = cv2.imread("images/bright.jpg", cv2.IMREAD_GRAYSCALE)    # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  ## OpenCV 함수 이용(saturation 방식)
07  dst1 = cv2.add(image, 100)                                       # 영상 밝게
08  dst2 = cv2.subtract(image, 100)                                  # 영상 어둡게
09
10  ## numpy.ndarray 이용(modulo 방식)
11  dst3 = image + 100                                               # 영상 밝게
12  dst4 = image - 100                                               # 영상 어둡게
13
14  cv2.imshow("original image", image)
15  cv2.imshow("dst1- bright:OpenCV", dst1)
16  cv2.imshow("dst2- dark:OpenCV", dst2)
17  cv2.imshow("dst3- bright:numpy", dst3)
18  cv2.imshow("dst4- dark:numpy", dst4)
19  cv2.waitKey(0)
```

6.2.2 영상의 화소 표현

- modulo 방식과 saturation 방식 비교

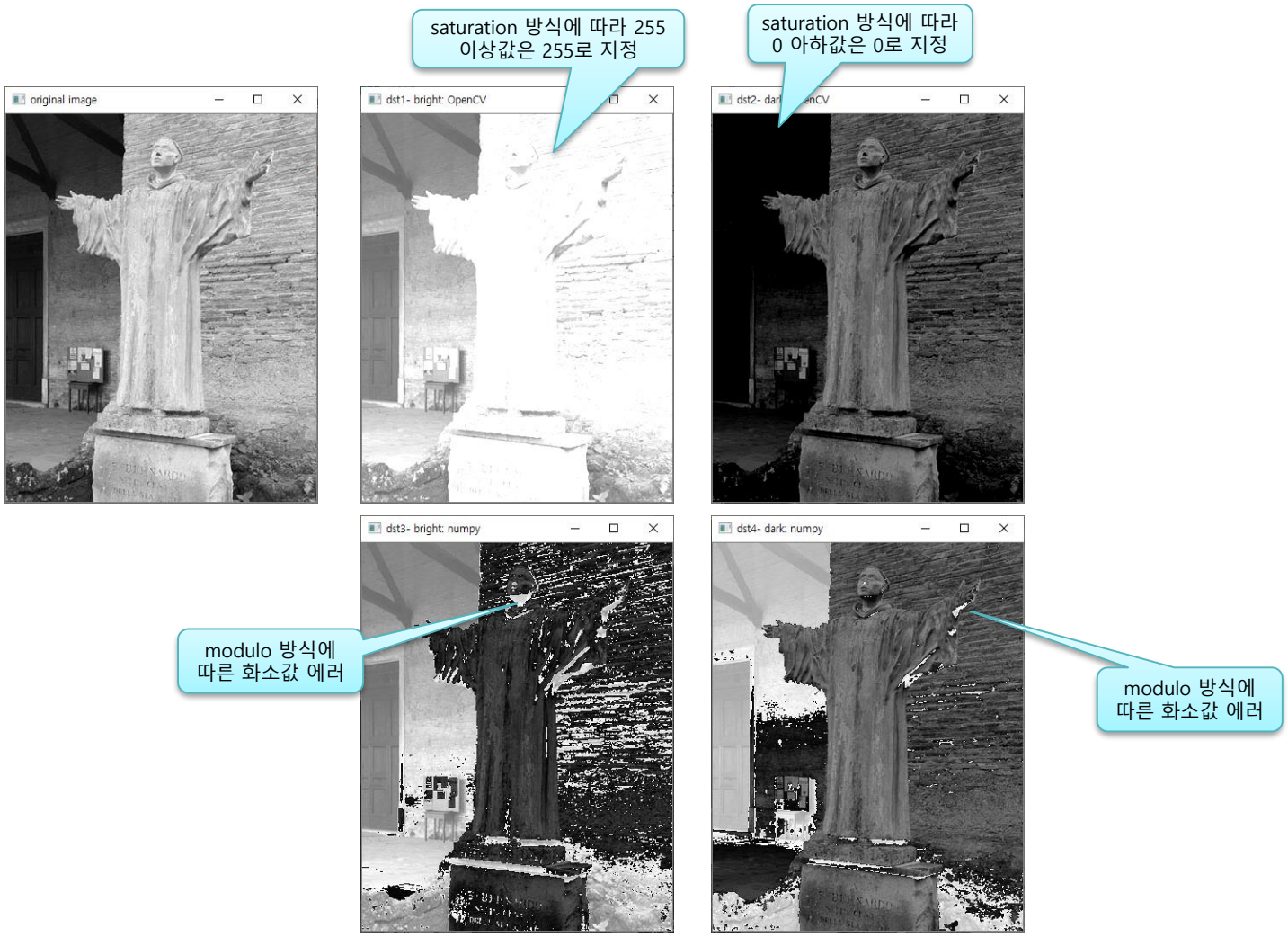
예제 6.2.3 행렬 가감 연산 통한 영상 밝기 변경 – 04.bright_dark.py

```
01  import cv2
02
03  image = cv2.imread("images/bright.jpg", cv2.IMREAD_GRAYSCALE)    # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  ## OpenCV 함수 이용(saturation 방식)
07  dst1 = cv2.add(image, 100)                                       # 영상 밝게
08  dst2 = cv2.subtract(image, 100)                                  # 영상 어둡게
09
10  ## numpy.ndarray 이용(modulo 방식)
11  dst3 = image + 100                                               # 영상 밝게
12  dst4 = image - 100                                               # 영상 어둡게
13
14  cv2.imshow("original image", image)
15  cv2.imshow("dst1- bright:OpenCV", dst1)
16  cv2.imshow("dst2- dark:OpenCV", dst2)
17  cv2.imshow("dst3- bright:numpy", dst3)
18  cv2.imshow("dst4- dark:numpy", dst4)
19  cv2.waitKey(0)
```

OpenCV와 numpy의 0 미만과 255 이상의 화소값 처리 방식이 다름에 주의
- OpenCV : 250 + 100 = 360 → 255 (saturation 방식)
- numpy : 250 + 100 = 350 % 256 → 104 (modulo 방식)

영상의 화소 표현

- 실행 결과



디지털 영상의 산술연산

■ 화소 값의 곱셈 연산

- 화소의 밝기 값에 특정 상수 값을 곱해 전체적으로 화소의 밝기 값이 증가해 더 밝아짐

> 화소 $* \alpha$: 영상의 밝기 차이 증가 = 뚜렷해짐

- 밝은 부분은 더욱 밝아지고, 어두운 부분은 약간 밝아져 영상 내의 밝기에 커다란 차이가 생기는 것

■ Ex

- 하나의 pixel이 '10'이고, 다른 하나는 '50' 이라고 가정할 때,
- 영상을 2배 곱셈을 하면
  - 10 x 2 = 20 (밝기), 50 x 2 = 100 (밝기)
  - 처음에는 40(50-10)정도의 밝기 차이가 있었으나, 곱셈 연산을 통해 2배 정도의 차이로 벌어졌음

■ 밝기의 차이가 커지므로 영상의 선명도 증가함

디지털 영상의 산술연산

(a) 원본 영상 (b) 상수 1.3을 곱한 영상

(c) 상수 1.5를 곱한 영상 (d) 상수 1.7을 곱한 영상

[그림 4-7] 곱셈 상수 변화에 따른 디지털 영상의 명도 대비 향상

6.2.4 행렬 덧셈 및 곱셈을 이용한 영상 합성

심화예제 6.2.4 행렬 합과 곱 연산을 통한 영상 합성 – 05.image_synthesis.py

```
01  import numpy as np, cv2
02
03  image1 = cv2.imread("images/add1.jpg", cv2.IMREAD_GRAYSCALE)     # 영상 읽기
04  image2 = cv2.imread("images/add2.jpg", cv2.IMREAD_GRAYSCALE)
05  if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")
06
07  ## 영상 합성 방법
08  alpha, beta = 0.6, 0.7                                          # 곱셈 비율
09  add_img1 = cv2.add(image1, image2)                              # 두 영상 단순 더하기
10  add_img2 = cv2.add(image1 * alpha, image2 * beta)               # 두 영상 비율에 따른 더하기
11  add_img2 = np.clip(add_img2, 0, 255).astype('uint8')            # saturation 처리
12  add_img3 = cv2.addWeighted(image1, alpha, image2, beta, 0) # 두 영상 비율에 따른 더하기
13
14  titles = ['image1', 'image2', 'add_img1', 'add_img2', 'add_img3'] # 윈도우 이름
15  for t in titles: cv2.imshow(t, eval(t))                         # 영상 표시
16  cv2.waitKey(0)
```

6.2.4 행렬 덧셈 및 곱셈을 이용한 영상 합성

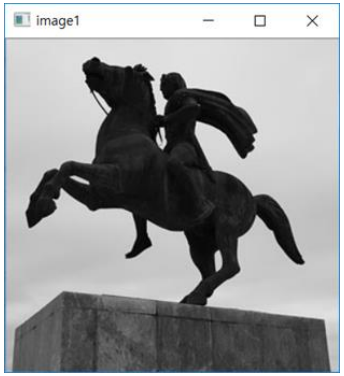
심화예제 6.2.4 행렬 합과 곱 연산을 통한 영상 합성 – 05.image_synthesis.py

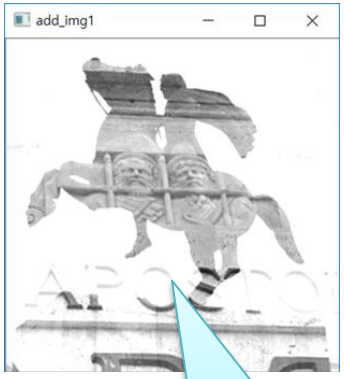
```
01  import numpy as np, cv2
02
03  image1 = cv2.imread("images/add1.jpg", cv2.IMREAD_GRAYSCALE)     # 영상 읽기
04  image2 = cv2.imread("images/add2.jpg", cv2.IMREAD_GRAYSCALE)
05  if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")
06
07  ## 영상 합성 방법
08  alpha, beta = 0.6, 0.7                                          # 곱셈 비율
09  add_img1 = cv2.add(image1, image2)                              # 두 영상 단순 더하기
10  add_img2 = cv2.add(image1 * alpha, image2 * beta)               # 두 영상 비율에 따른 더하기
11  add_img2 = np.clip(add_img2, 0, 255).astype('uint8')            # saturation 처리
12  add_img3 = cv2.addWeighted(image1, alpha, image2, beta, 0) # 두 영상 비율에 따른 더하기
13
14  titles = ['image1', 'image2', 'add_img1', 'add_img2', 'add_img3'] # 윈도우 이름
15  for t in titles: cv2.imshow(t, eval(t))                         # 영상 표시
16  cv2.waitKey(0)
```

1) dst(y,x) = image1(y,x) + image2(y,x) ;
2) dst(y,x) = image1(y,x) * alpha + image2(y,x) * beta
3) dst(y,x) = image1(y,x) * alpha + image2(y,x) * beta + gamma

6.2.1 그레이 스케일(명암도) 영상

- 실행결과





1대1 합성 – 화소값이 255가 넘는 경우들이 생겨 밝은 값으로 나타남

비율 조정하여 합성

6.2.5 명암 대비

- 대비
  - 같은 색도 인접한 색의 밝기에 따라서 다르게 보임

〈그림 6.2.2〉 밝기 대비 예시

- 대비 영상의 예

6.2.5 명암 대비

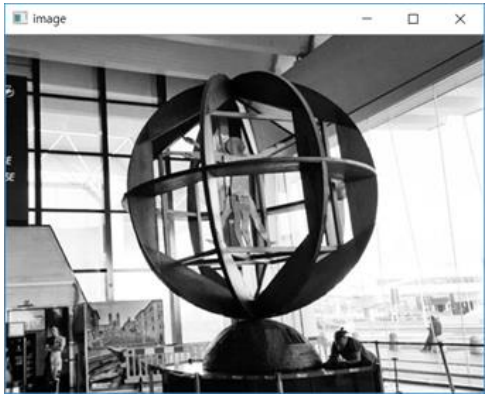
예제 6.2.5 영상 대비 변경 – 06.contrast.py

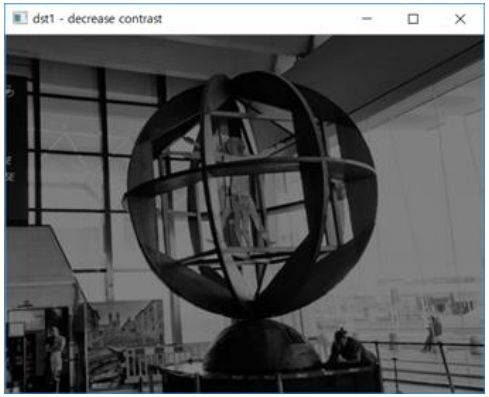
```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/contrast.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  noimage = np.zeros(image.shape[:2], image.dtype)                # 더미 영상
07  avg = cv2.mean(image)[0]/2.0                                    # 영상 화소 평균의 절반
08
09  dst1 = cv2.scaleAdd(image, 0.5, noimage)                        # 명암 대비 감소
10  dst2 = cv2.scaleAdd(image, 2.0, noimage)                        # 명암 대비 증가
11  dst3 = cv2.addWeighted(image, 0.5, noimage, 0, avg)             # 명암 대비 감소
12  dst4 = cv2.addWeighted(image, 2.0, noimage, 0, -avg)            # 명암 대비 증가
13
14  cv2.imshow("image", image)                                      # 영상 띄우기
15  cv2.imshow("dst1 - decrease contrast", dst1)
16  cv2.imshow("dst2 - increase contrast", dst2)
17  cv2.imshow("dst3 - decrease contrast using average", dst3)
18  cv2.imshow("dst4 - increase contrast using average", dst4)
19  cv2.waitKey(0)
```

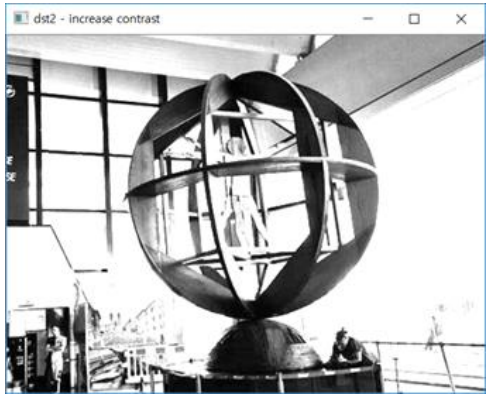
6.2.5 명암 대비

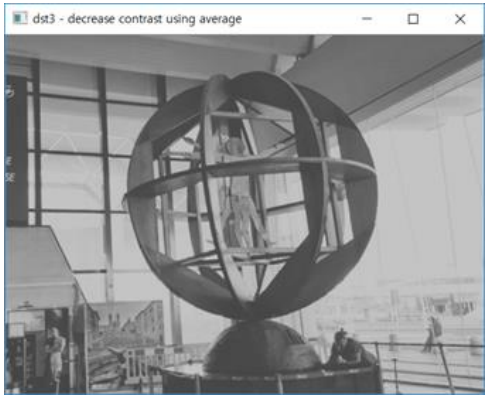
실행결과

곱셈으로 영상 대비 변경(감소 및 증가)









영상 평균값을 활용하여
대배 변경시 화질 개선

연습문제1

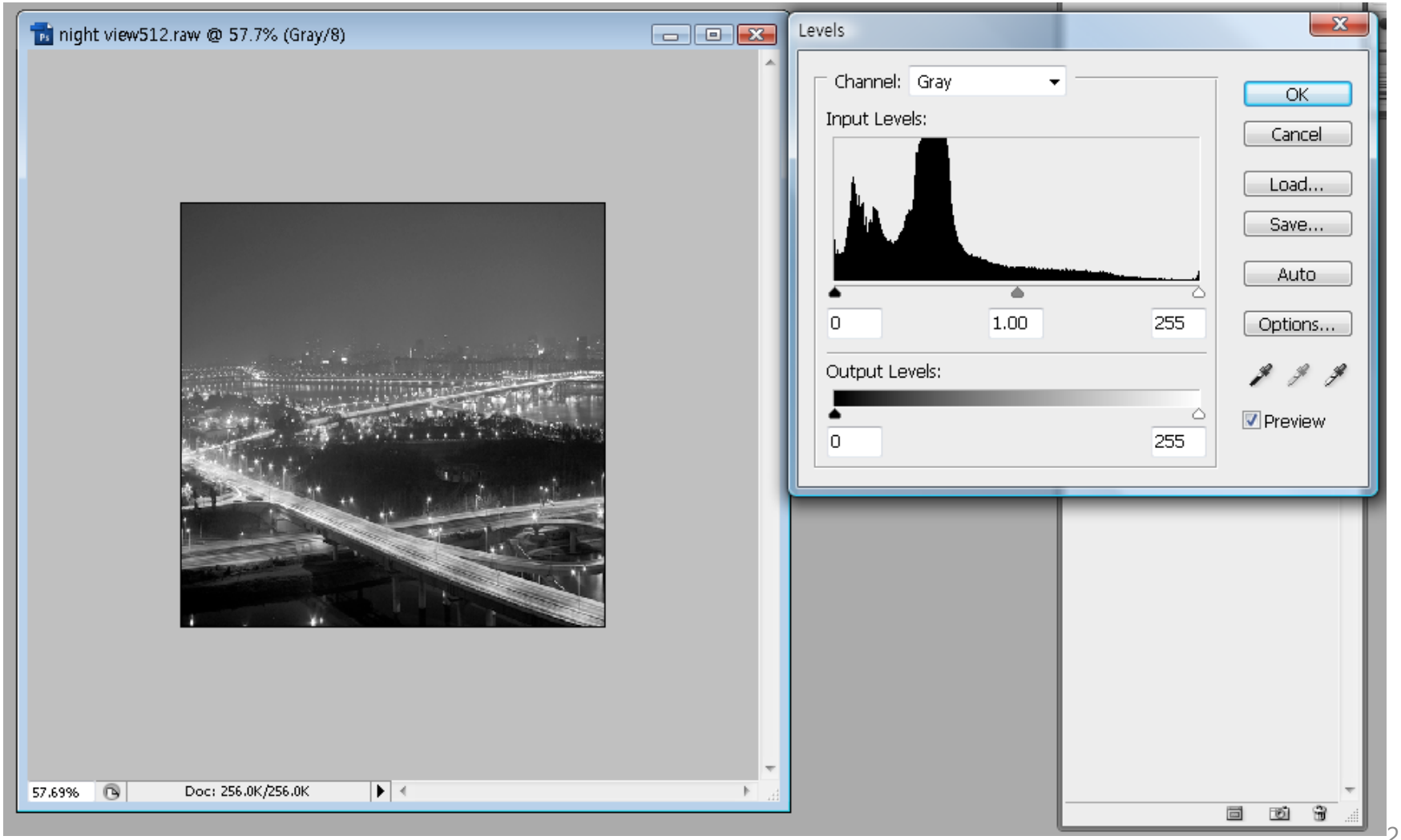
- OpenCV함수 중에서 cv2.addWeighted() 함수를 사용해서 두 영상을 합성하는 프로그램을 작성하시오

6.3 히스토그램

- 6.3.1 히스토그램 개념
- 6.3.2 히스토그램 계산
- 6.3.3 OpenCV 함수 활용
- 6.3.4 히스토그램 스트레칭

5장. 히스토그램을 이용한 화소

■ 포토샵에서 히스토그램

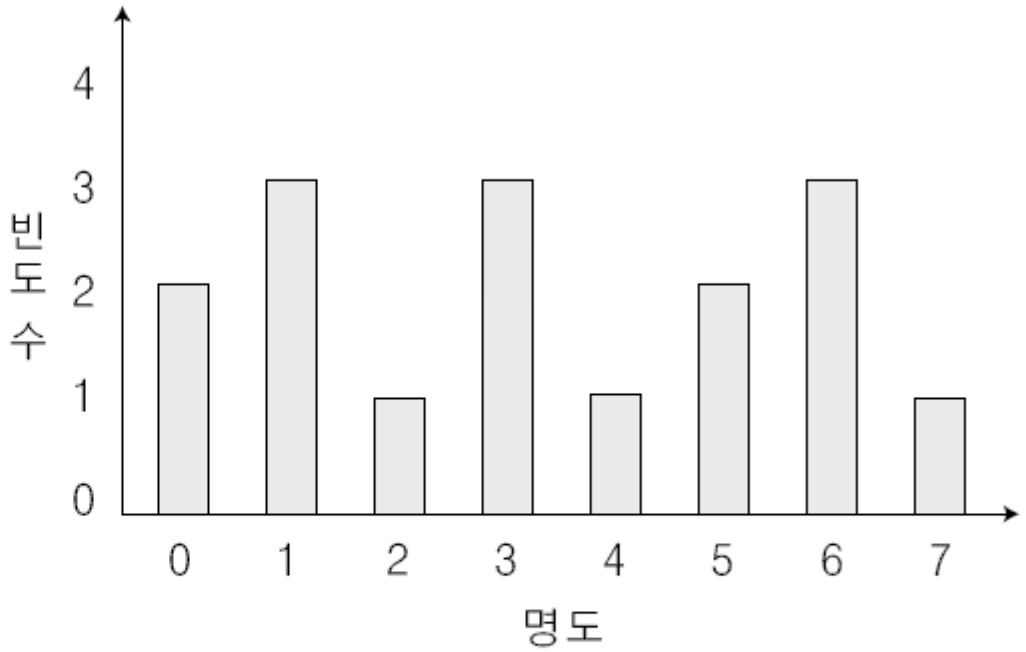


Section 01 디지털 영상의 히스토그램

■ 디지털 영상의 히스토그램

- 관찰한 데이터의 특징을 한눈에 알아볼 수 있도록 데이터를 막대그래프 모양으로 나타낸 것
- 디지털 영상에 대한 많은 정보를 제공함

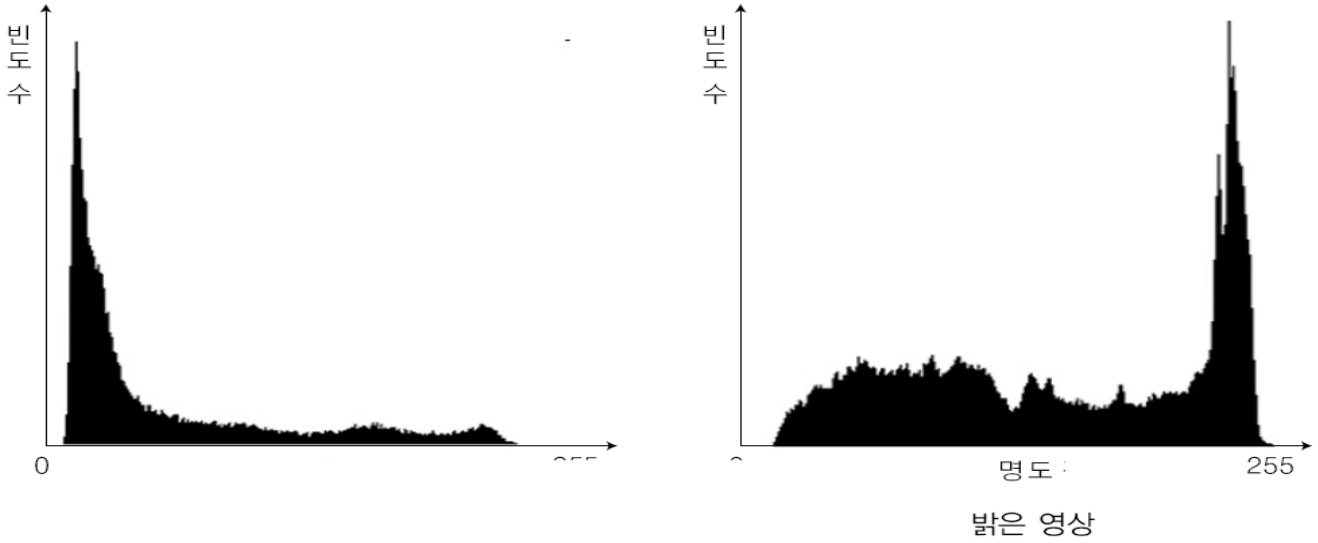
6	6	6	7
4	5	5	3
2	1	1	3
0	0	1	3



(a) 입력 영상 (b) 히스토그램

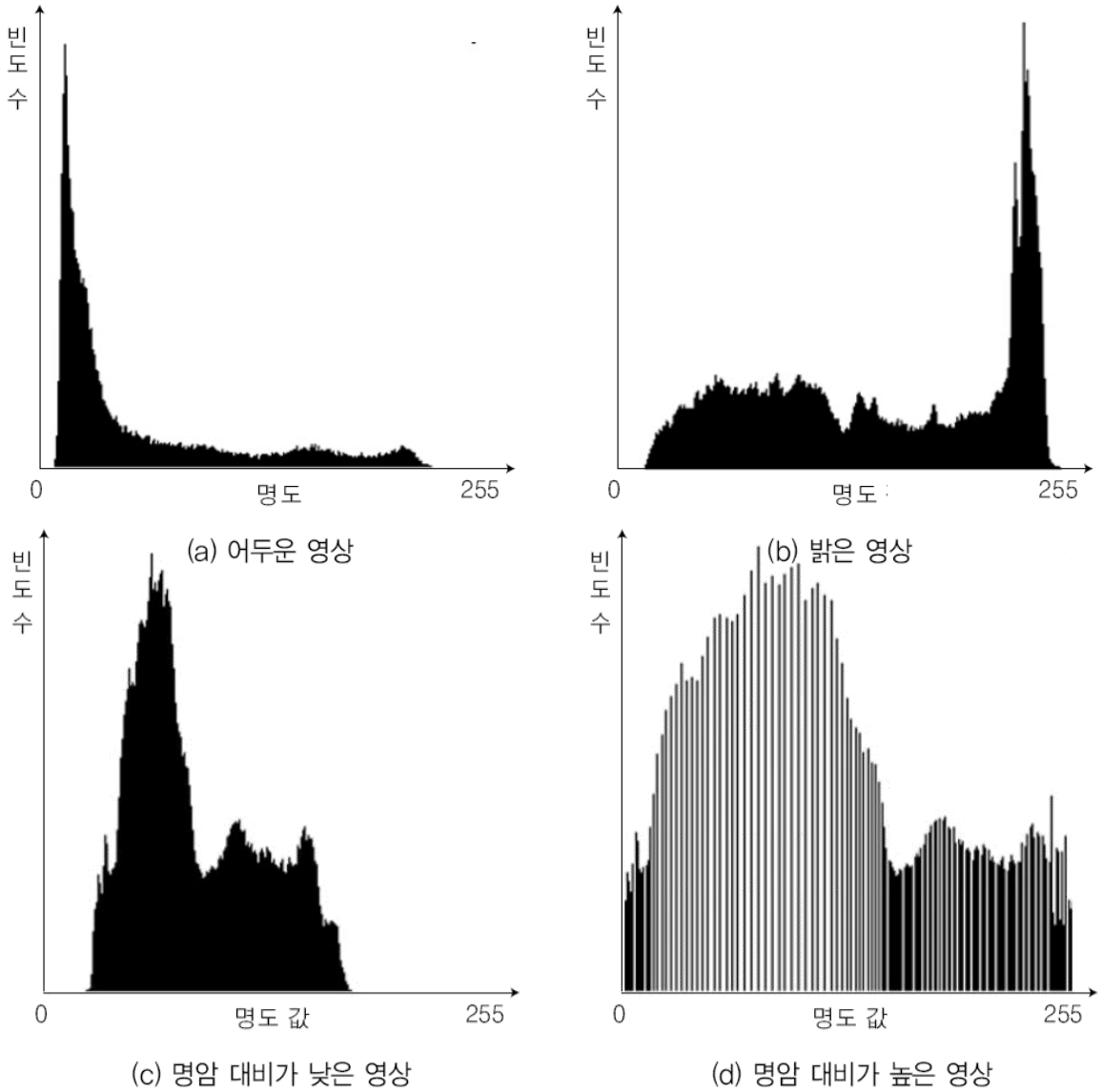
[그림 5-1] 이상적인 영상의 히스토그램

영상의 특성에 따른 히스토그램



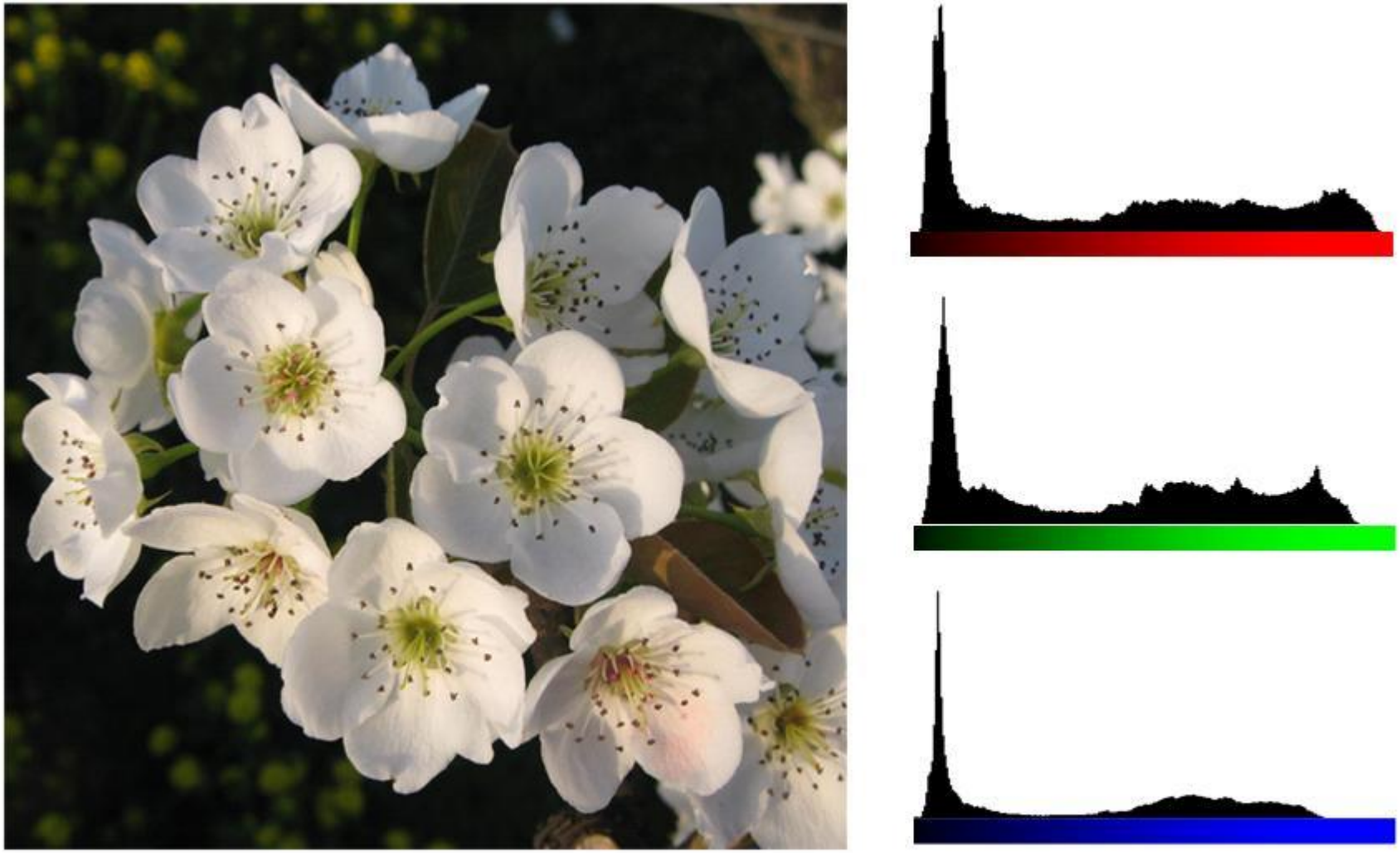
왼쪽 그림과 오른쪽 그림의 차이를 설명해 보자

영상의 특성에 따른 히스토그램



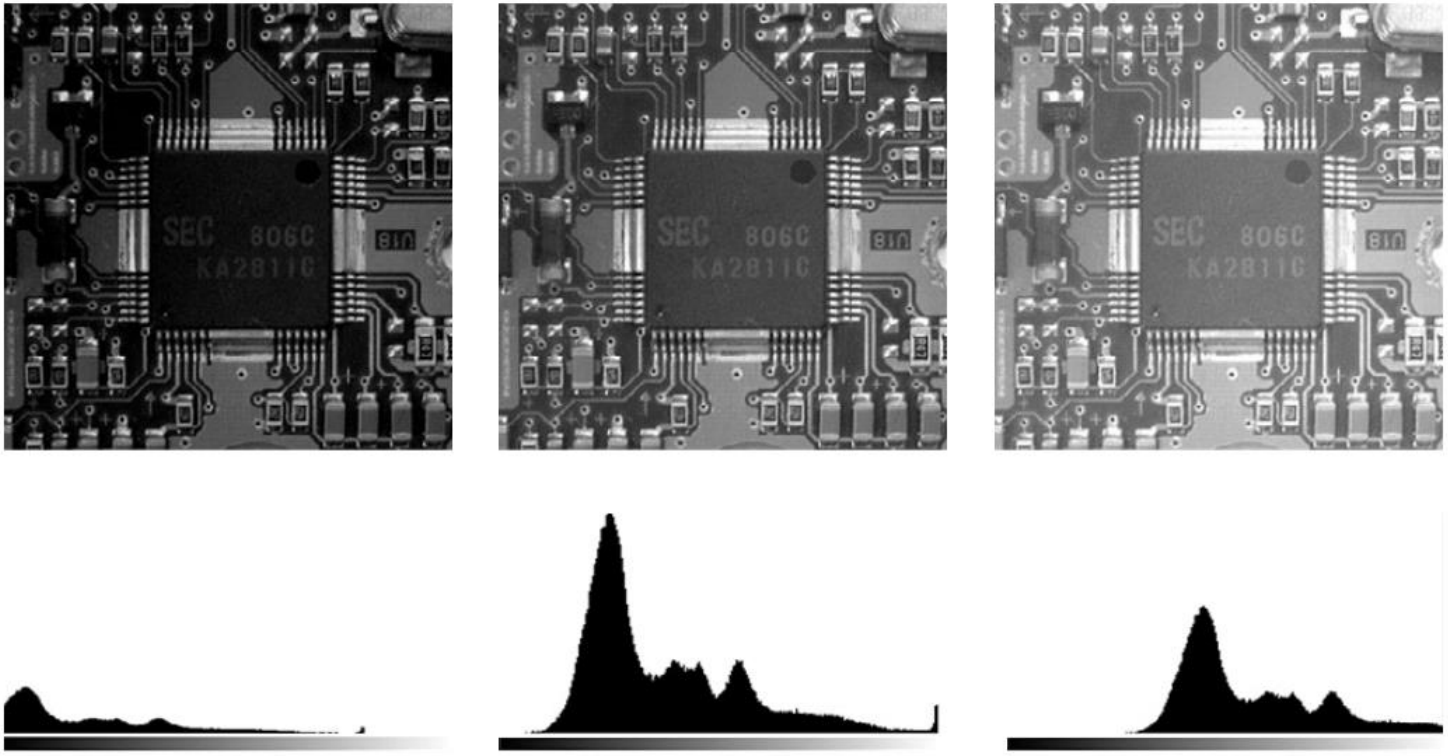
[그림 5-2] 영상의 특성에 따른 히스토그램

RGB 컬러 영상의 히스토그램



[그림 5-3] RGB 컬러 영상의 히스토그램

Section 02 산술 연산을 이용한 히스토그램 이동



(a) 50을 뺄셈한 영상 (b) 원본 영상 (c) 50을 덧셈한 영상

[그림 5-4] 덧셈과 뺄셈연산으로 히스토그램 이동하기

- 덧셈연산: 명도 값을 증가시켜 밝게, 히스토그램의 기둥이 오른쪽으로 이동
- 뺄셈연산: 명도 값을 감소시켜 어둡게, 히스토그램의 기둥이 왼쪽으로 이동

히스토그램 스트레칭

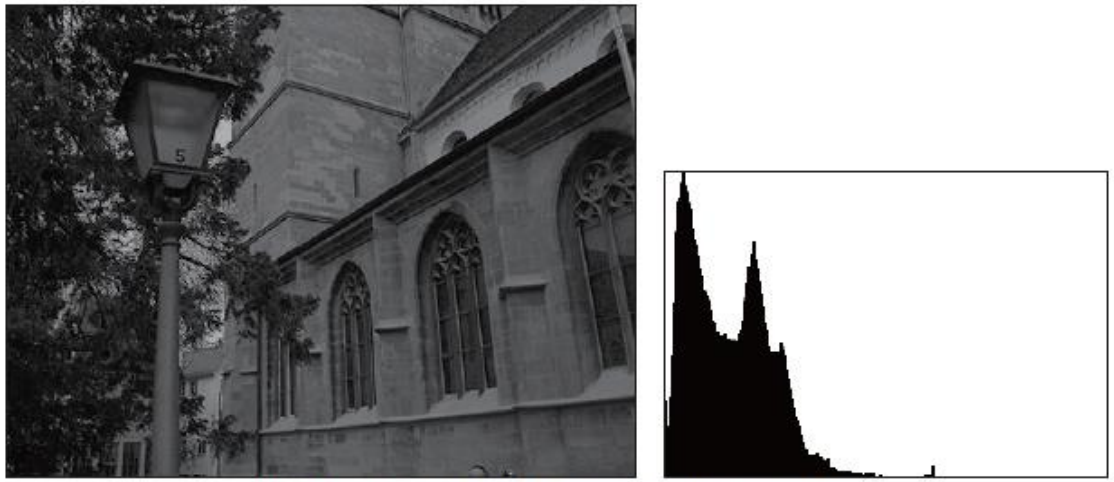
■ 히스토그램 스트레칭(Histogram Stretching)

- 명암 대비를 향상시키는 연산으로, 낮은 명암 대비를 보이는 영상의 화질을 향상시키는 방법
- 명암 대비 스트레칭이라고도 함
- 히스토그램이 모든 범위의 화소 값을 포함하도록 히스토그램의 분포를 넓힘
- 기본 명암 대비 스트레칭과 앤드-인 탐색 기법이 대표적

6.3.4 히스토그램 스트레칭

- 히스토그램의 분포가 좁아서 영상의 대비가 좋지 않은 영상

(a) 밝은 부분을 많이 분포하는 영상과 히스토그램



(b) 어두운 부분을 많이 분포하는 영상과 히스토그램

기본 명암 대비 스트레칭

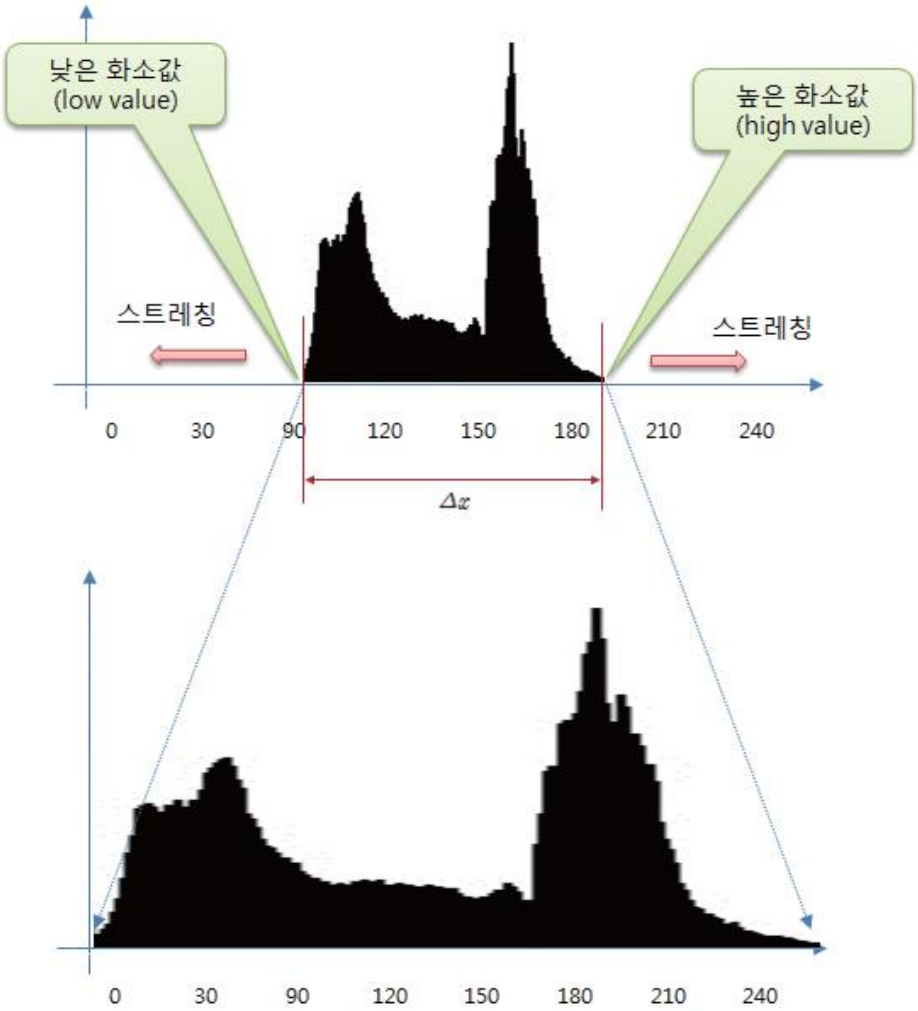
- 이상적이지 못한 히스토그램 분포 중에서 **명암 대비**가 낮은 디지털 영상의 품질을 향상시키는 기술
- 특정 부분이나 가운데에 집중된 히스토그램을 모든 영역으로 확장 시켜서 디지털 영상이 모든 범위의 화소 값을 포함하게 함
- **기본 명암 대비 스트레칭 수행 공식**

$$new\ pixel = \frac{old\ \ pixel - low}{high - low} \times 255$$

  - old pixel은 원 영상 화소의 명도 값 (현재 화소 값)
  - new pixel은 결과 영상 화소의 명도 값
  - low는 히스토그램의 최저 명도 값
  - high는 히스토그램의 최고 명도 값

6.3.4 히스토그램 스트레칭

- 히스토그램 스트레칭
  - 명암 분포가 좁은 히스토그램을 좌우로 잡아당겨(스트레칭해서) 고른 명암 분포를 가진 히스토그램이 되게 하는 것



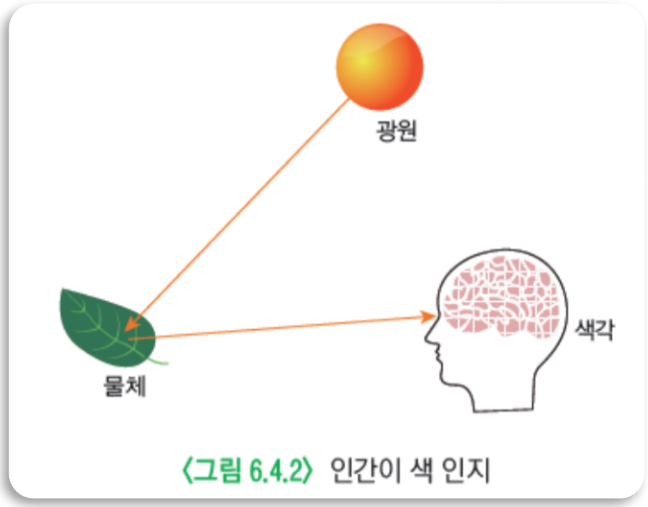
$$\text{새 화소값} = \frac{(\text{화소값} - low)}{high - low} * 255$$

6.4 컬러 공간 변환

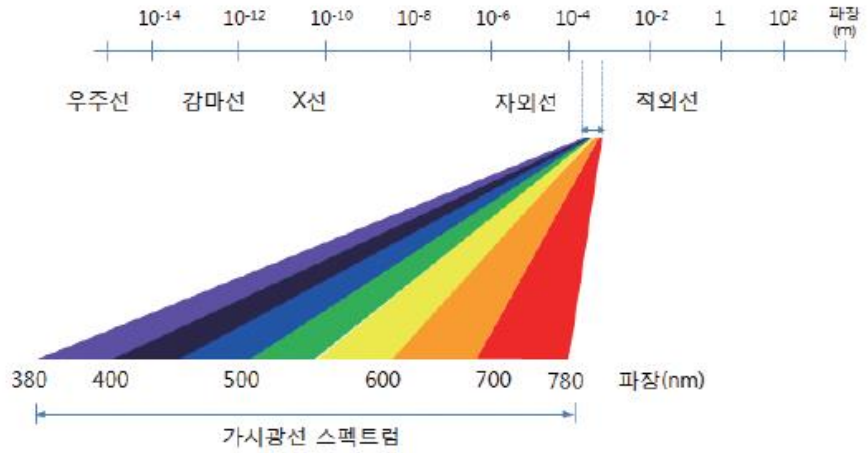
- 6.4.1 컬러 및 컬러 공간
- 6.4.2 RGB 컬러 공간
- 6.4.3 CMY(K) 컬러 공간
- 6.4.4 HSI 컬러 공간
- 6.4.5 기타 컬러 공간

6.4.1 컬러 및 컬러 공간

- 색
  - 색각으로 느낀 빛에서 주파수(파장)의 차이에 따라 다르게 느껴지는 색상



〈그림 6.4.2〉 인간이 색 인지



〈그림 6.4.3〉 가시광선의 분포

- 가시 광선
  - 전체 전자기파 중에서 인간이 인지할 수 있는 약 380 nm~780 nm사이의 파장

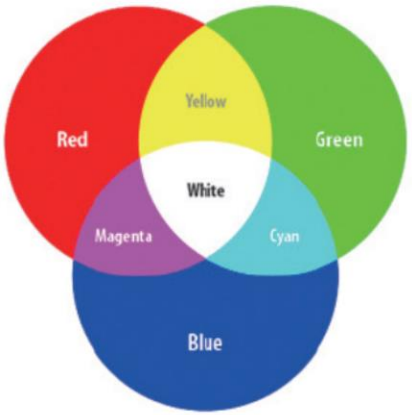
6.4.1 컬러 및 컬러 공간

- 컬러 공간(color space)
  - 색 표시계(color system)의 모든 색들을 색 공간에서 3차원 좌표로 표현한 것
    - 색 표시계 - RGB, CMY, HSV, LAB, YUV 등의 색 체계
  - 공간 상의 좌표로 표현 → 어떤 컬러와 다른 컬러들과의 관계를 표현하는 논리적인 방법 제공

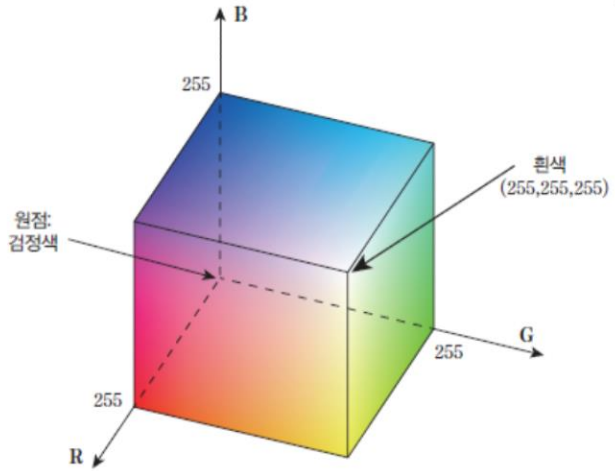
- 영상처리에서 컬러 공간 다양하게 활용
  - 어떤 영상에서 각각의 색상이 다른 객체를 분리하기 위해서 적절한 컬러 공간을 이용해 전체 영상을 컬러 영역별 분리
  - 전선의 연결 오류 검사를 위해 기준 색상과 비슷한 색상의 전선인지를 체크 위해 사용
  - 내용 기반 영상 검색에서 색상 정보를 이용하여 원하는 물체를 검색하기 위해 특정 컬러 공간 이용

6.4.2 RGB 컬러 공간

- RGB 컬러 공간
  - 가산 혼합(additive mixture) – 빛을 섞을 수록 밝아짐
  - 빛을 이용해서 색 생성
  - 빛의 삼원색(빨강 빛, 초록 빛, 파랑 빛) 사용
  - 모니터, 텔레비전, 빔 프로젝터와 같은 디스플레이 장비들에서 기본 컬러 공간으로 사용



〈그림 6.4.5〉 가산 혼합



〈그림 6.4.6〉 RGB 컬러 공간

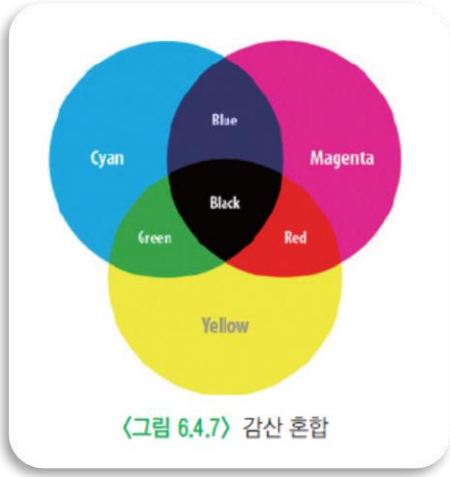
6.4.2 RGB 컬러 공간

〈표 6.4.1〉 대표적인 색상에 대한 RGB 표현

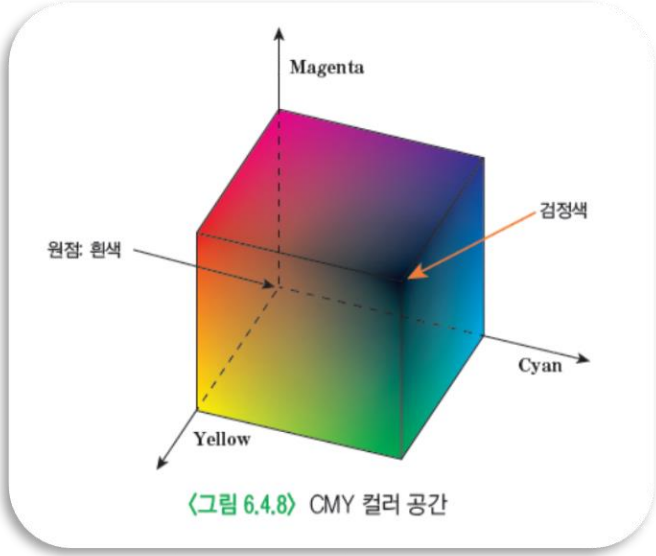
RGB 화소값	색상	RGB 화소값	색상
0, 0, 0	white	240, 230, 140	khaki
255, 255, 255	black	238, 130, 238	violet
128, 128, 128	gray	255, 165, 0	orange
192, 192, 192	silver	255, 215, 0	gold
255, 0, 0	red	0, 0, 128	navy
0, 255, 0	green	160, 32, 240	purple
0, 0, 255	blue	0, 128, 128	olive
255, 255, 0	yellow	75, 0, 130	indigo
255, 0, 255	magenta	255, 192, 203	pink
0, 255, 255	cyan	135, 206, 235	skyblue

6.4.3 CMY(K) 컬러 공간

- CMY 컬러 공간
  - 감산 혼합(subtractive mixture) – 섞을 수록 어두워지는 방식
  - 색의 삼원색(청록색(Cyan), 자홍색(Magenta), 노랑색(Yellow))으로 색 만듦
  - RGB 컬러 공간과 보색 관계



〈그림 6.4.7〉 감산 혼합



〈그림 6.4.8〉 CMY 컬러 공간

C = 255 − R	R = 255 − C
M = 255 − G	G = 255 − M
Y = 255 − R	B = 255 − Y

6.4.3 CMY(K) 컬러 공간

- CMYK 컬러공간
  - 순수한 검정색 사용
    - 뛰어난 대비를 제공, 검정 잉크가 컬러 잉크보다 비용이 적은 장점

blacK	= min(Cyan, Magenta, Yellow)
Cyan	= Cyan - blacK
Magenta	= Magenta- blacK
Yellow	= Yellow - blacK

6.4.3 CMY(K) 컬러 공간

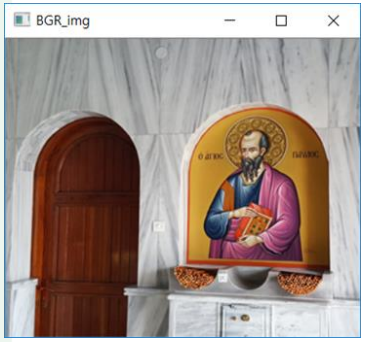
예제 6.4.1 컬러 공간 변환(BGR→CMY) – convert_CMY.py

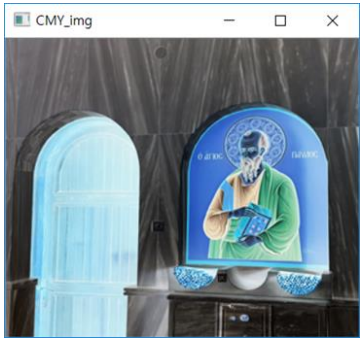
```
01  import numpy as np, cv2
02
03  BGR_img = cv2.imread("images/color_model.jpg", cv2.IMREAD_COLOR) # 컬러 영상 읽기
04  if BGR_img is None: raise Exception("영상파일 읽기 오류")
05
06  white = np.array([255, 255, 255], np.uint8)
07  CMY_img = white - BGR_img
08  Yellow, Magenta, Cyan = cv2.split(CMY_img)                    # 채널 분리
09
10  titles = ['BGR_img', 'CMY_img', 'Yellow', 'Magenta', 'Cyan']
11  for t in titles: cv2.imshow(t, eval(t))
12  cv2.waitKey(0)
```

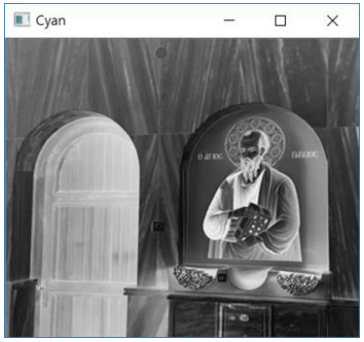
6.4.3 CMY(K) 컬러 공간

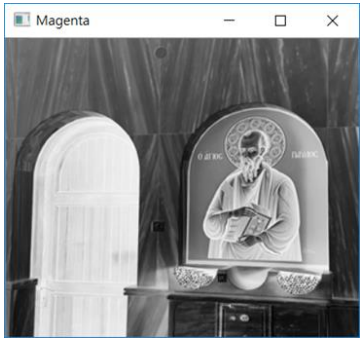
예제 6.4.1 컬러 공간 변환(BGR→CMY) - convert_CMY.py

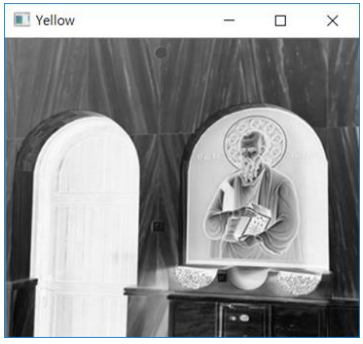
```
01  import numpy as np, cv2
02
03  BGR_img = cv2.imread("images/color_model.jpg", cv2.IMREAD_COLOR) # 컬러 영상 읽기
04  if BGR_img is None: raise Exception("영상파일 읽기 오류")
05
06  white = np.array([255, 255, 255], np.uint8)
07  CMY_img = white - BGR_img
08  Yellow, Magenta, Cyan = cv2.split(CMY_img)                    # 채널 분리
09
10  titles = ['BGR_img', 'CMY_img', 'Yellow', 'Magenta', 'Cyan']
11  for t in titles: cv2.imshow(t, eval(t))
12  cv2.waitKey(0)
```











6.4.5 기타 컬러 공간

cv2.cvtColor() 함수

〈표 6.4.2〉 컬러 공간 변환을 위한 옵션 상수(cv2. 생략)

옵션 상수	값	옵션 상수	값	옵션 상수	값
COLOR_BGR2BGRA	0	COLOR_BGR2YCrCb	36	COLOR_HSV2BGR	54
COLOR_BGRA2BGR	1	COLOR_RGB2YCrCb	37	COLOR_HSV2RGB	55
COLOR_BGRA2RGB	2	COLOR_YCrCb2BGR	38	COLOR_LAB2BGR	56
COLOR_RGBA2BGR	3	COLOR_YCrCb2RGB	39	COLOR_LAB2RGB	57
COLOR_BGR2RGB ,	4	COLOR_BGR2HSV	40	COLOR_LUV2BGR	58
COLOR_BGRA2RGBA	5	COLOR_RGB2HSV	41	COLOR_LUV2RGB	59
COLOR_BGR2GRAY	6	COLOR_BGR2LAB	44	COLOR_HLS2BGR	60
COLOR_RGB2GRAY	7	COLOR_RGB2LAB	45	COLOR_HLS2RGB	61
COLOR_GRAY2BGR	8	COLOR_BayerBG2BGR	46	COLOR_BGR2YUV	82
COLOR_GRAY2BGRA	9	COLOR_BayerGB2BGR	47	COLOR_RGB2YUV	83
COLOR_BGRA2GRAY	10	COLOR_BayerRG2BGR	48	COLOR_YUV2BGR	84
COLOR_RGBA2GRAY	11	COLOR_BayerGR2BGR	49	COLOR_YUV2RGB	85
COLOR_BGR2XYZ	32	COLOR_BGR2LUV	50	COLOR_BayerBG2GRAY	86
COLOR_RGB2XYZ	33	COLOR_RGB2LUV	51	COLOR_BayerGB2GRAY	87
COLOR_XYZ2BGR	34	COLOR_BGR2HLS	52	COLOR_BayerRG2GRAY	88
COLOR_XYZ2RGB	35	COLOR_RGB2HLS	53	COLOR_BayerGR2GRAY	89

6.4.5 기타 컬러 공간

예제 6.4.4 다양한 컬러 공간 변환 – convert_others.py

```
01  import cv2
02
03  BGR_img = cv2.imread("images/color_space.jpg", cv2.IMREAD_COLOR) # 컬러 영상 읽기
04  if BGR_img is None: raise Exception("영상파일 읽기 오류")
05
06  Gray_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2GRAY)          # 명암도 영상 변환
07  YCC_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2YCrCb)          # YCbCr 컬러 공간 변환
08  YUV_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2YUV)            # YUV 컬러 공간 변환
09  LAB_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2LAB)            # La*b* 컬러 공간 변환
10
11  YCC_ch = cv2.split(YCC_img)                                   # 채널 분리
12  YUV_ch = cv2.split(YUV_img)
13  Lab_ch = cv2.split(LAB_img)
14
15  cv2.imshow("BGR_img", BGR_img)
16  cv2.imshow("Gray_img", Gray_img)
17
18  sp1, sp2, sp3 = ['Y', 'Cr', 'Cb'] , ['Y', 'U', 'V'], ['L', 'A', 'B']
19  for i in range(len(ch1)):
20        cv2.imshow("YCC_img[%d]-%s" %(i, sp1[i]), YCC_ch[i])
21        cv2.imshow("YUV_img[%d]-%s" %(i, sp2[i]), YUV_ch[i])
22        cv2.imshow("LAB_img[%d]-%s" %(i, sp3[i]), Lab_ch[i])
23  cv2.waitKey(0)
```

임계치

- cv2.threshold(src, thresh, maxval, type, dst=None) -> retval, dst

  – src: 입력 영상. 다채널, 8비트 또는 32비트 실수형

  – thresh: 사용자 지정 임계값

  – maxval: cv2.THRESH_BINARY 또는 cv2.THRESH_BINARY_INV 방법 사용 시 최댓값. 보통 255로 지정

  - cv2.THRESH_TOZERO : pixel's value > threshold --> pixel's value, or 0
  - cv2.THRESH_TOZERO_INV : 위의 반대

- cv2.THRESH_BINARY - 조건을 만족하는 픽셀값을 255로 만족하지 않는 픽셀값을 0으로 설정
- cv2.THRESH_BINARY_INV - 조건을 만족하는 픽셀값을 0로 만족하지 않는 픽셀값을 255으로 설정
- cv2.THRESH_TRUNC - 조건을 만족하는 픽셀값을 255로 만족하지 않는 픽셀값을 원본값으로 설정
- cv2.THRESH_TOZERO - 조건을 만족하는 픽셀값을 원본값으로 만족하지 않는 픽셀값을 0으로 설정
  cv2.THRESH_TOZERO_INV - 조건을 만족하는 픽셀값을 0로 만족하지 않는 픽셀값을 원본값으로 설정

–type: cv2.THRESH_ 로 시작하는 플래그. 임계값 함수 동작 지정 또는 자동 임계값 결정 방법 지정

–retval: 사용된 임계값

–dst: 출력 영상. src와 동일 크기, 동일 타입, 같은 채널 수

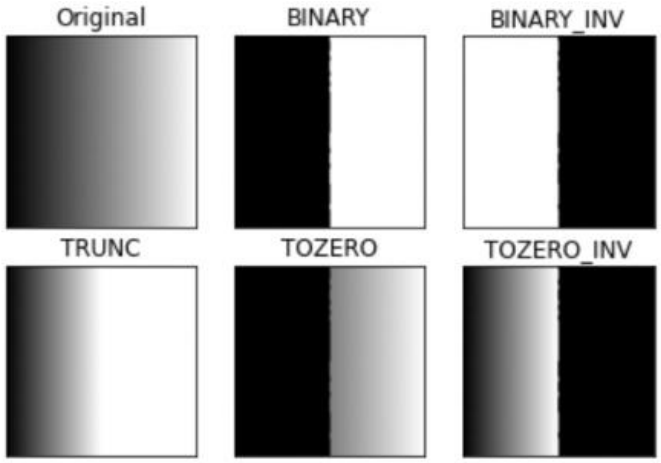


fig3. output of sample code

임계치

```
import cv2

def onThreshold(value):
    result  = cv2.threshold(image, value, 255, cv2.THRESH_BINARY)[1]
    cv2.imshow("result", result)


image = cv2.imread("images/color_space.jpg",
cv2.IMREAD_GRAYSCALE) # 컬러 영상 읽기
if image is None: raise Exception("영상 파일 읽기 오류")

cv2.namedWindow("result")
cv2.createTrackbar("threshold", "result", 128, 255, onThreshold)
onThreshold(128)                                    # 이진화 수행
cv2.imshow("image", image)
cv2.waitKey(0)
```

임계치

```
import cv2

def onThreshold(value):
    result  = cv2.threshold(image, value, 255, cv2.THRESH_BINARY)[1]
    cv2.imshow("result", result)


image = cv2.imread("images/color_space.jpg", cv2.IMREAD_GRAYSCALE) # 컬러 영상 읽기
if image is None: raise Exception("영상 파일 읽기 오류")

cv2.namedWindow("result")
cv2.createTrackbar("threshold", "result", 128, 255, onThreshold)
onThreshold(128)                                    # 이진화 수행
cv2.imshow("image", image)
cv2.waitKey(0)
```

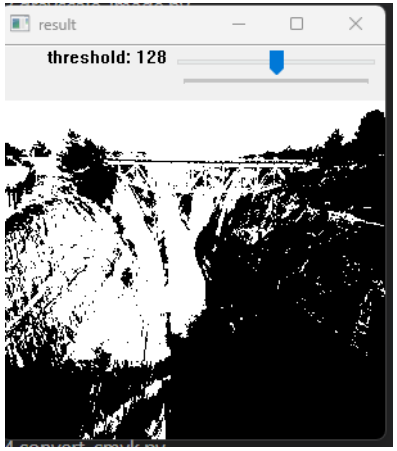
임계치 (테스트)

```
import cv2

def onThreshold(value):
    result  = cv2.threshold(image, value, 255, cv2.THRESH_BINARY)[1]
    cv2.imshow("result", result)


image = cv2.imread("images/color_space.jpg", cv2.IMREAD_GRAYSCALE) # 컬러 영상 읽기
if image is None: raise Exception("영상 파일 읽기 오류")

cv2.namedWindow("result")
cv2.createTrackbar("threshold", "result", 128, 255, onThreshold)
onThreshold(128)                                    # 이진화 수행
cv2.imshow("image", image)
cv2.waitKey(0)
```

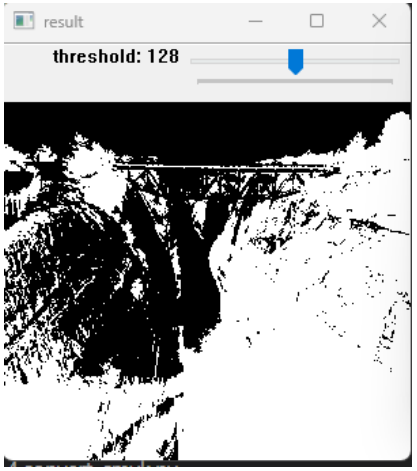
- 다음의 옵션을 넣어 테스트해 보자
  - cv2.THRESH_BINARY
  - cv2.THRESH_BINARY_INV
  - cv2.THRESH_TOZERO
  - cv2.THRESH_TOZERO_INV

6.4.5 기타 컬러 공간

- 실행 결과

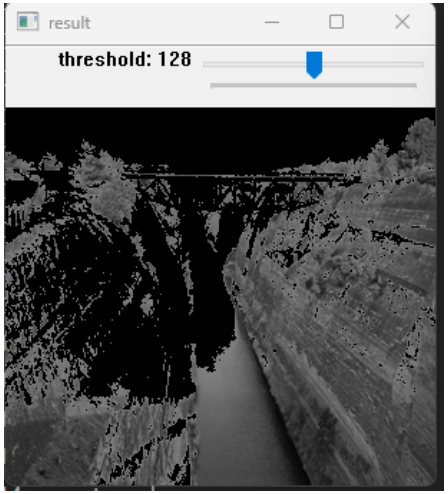


cv2.THRESH_BINARY



cv2.THRESH_BINARY_INV

cv2.THRESH_TOZERO



cv2.THRESH_TOZERO_INV

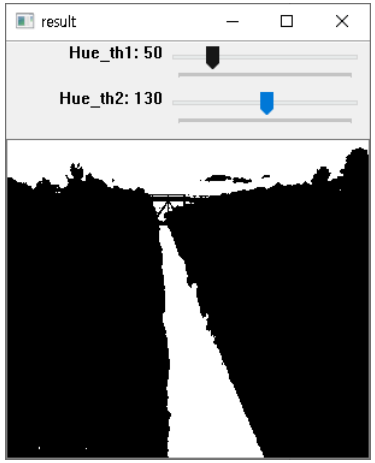
6.4.5 기타 컬러 공간

심화예제 6.4.5 Hue 채널을 이용한 객체 검출 – 17.hue_threshold.py

```
01  import numpy as np, cv2
02
03  def onThreshold(value):
04      th[0] = cv2.getTrackbarPos("Hue_th1", "result")
05      th[1] = cv2.getTrackbarPos("Hue_th2", "result")
06
07      ## 이진화– 화소 직접 접근 방법
08      # result = np.zeros(hue.shape, np.uint8)
09      # for i in range(result.shape[0]):
10      #     for j in range(result.shape[1]):
11      #         if th[0] <= hue[i, j] < th[1] : result[i, j] = 255
12
13      ## 이진화– 넘파이 함수 활용 방식
14      # result = np.logical_and(hue < th[1], hue >= th[0])
15      # result = result.astype('uint8') * 255
16
17      ## OpenCV 이진화 함수 이용– 상위 값과 하위 값 제거
18      _, result = cv2.threshold(hue, th[1], 255, cv2.THRESH_TOZERO_INV)
19      cv2.threshold(result, th[0], 255, cv2.THRESH_BINARY, result)
20      cv2.imshow("result", result)
21
```

6.4.5 기타 컬러 공간

```
22  BGR_img = cv2.imread("images/color_space.jpg", cv2.IMREAD_COLOR)      # 컬러 영상 읽기
23  if BGR_img is None: raise Exception("영상파일 읽기 오류")
24
25  HSV_img = cv2.cvtColor(BGR_img, cv2.COLOR_BGR2HSV)      # 컬러 공간 변환
26  hue = np.copy(HSV_img[:, :, 0])                         # hue 행렬에 색상 채널 복사
27
28  th = [50, 100]                                          # 트랙바로 선택할 범위 변수
29  cv2.namedWindow("result")
30  cv2.createTrackbar("Hue_th1", "result", th[0], 255, onThreshold)
31  cv2.createTrackbar("Hue_th2", "result", th[1], 255, onThreshold)
32  onThreshold(th[0])                                      # 이진화 수행
33  cv2.imshow("BGR_img", BGR_img)
34  cv2.waitKey(0)
```



연습문제2

■ 지난 문제에서 두 개의 트랙바를 추가해서 각 영상의 반영 비율을 조절할 수 있도록 수정하시오.