동양미래대학교

파이썬을 활용한 컴퓨터 비전 입문

Chapter 05. 영상의 밝기와 명암비 조절

동양미래대학교
인공지능소프트웨어학과
권 범

본 강의자료는 길벗 출판사의『OpenCV 4로 배우는 컴퓨터 비전과 머신 러닝』
교재 내용을 토대로 작성되었습니다.

목차

5.1 영상의 밝기 조절

5.2 영상의 명암비 조절

5.3 히스토그램 분석

❖ 그레이스케일 영상 다루기 (1/3)

- OpenCV에서 영상 파일을 그레이스케일 형태로 불러오려면 `imread()` 함수의 두 번째 인자에 `IMREAD_GRAYSCALE` 플래그를 설정해야 함
- 예를 들어 lenna.bmp 파일로부터 레나 영상을 그레이스케일 영상 형태로 불러오려면 다음과 같이 코드를 작성함

```
img1 = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
```

❖ 그레이스케일 영상 다루기 (2/3)

- 만약 이미 3채널 컬러 영상을 가지고 있고, 이 영상을 그레이스케일 영상으로 변환하려면 cvtColor() 함수를 사용함
- 다음 코드는 3채널 컬러 영상을 1채널 그레이스케일 영상으로 변환하는 코드임

```
img_color = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2COLOR)
```

❖ 그레이스케일 영상 다루기 (3/3)

- 이 예제 코드에서 `img_color` 변수에는 3채널 BGR 컬러 형식의 레나 영상이 저장됨
- BGR이라는 것은 색상 채널 순서가 파란색(Blue), 녹색(Green), 빨간색(Red) 순서로 설정되어 있음을 의미함
- `cvtColor()` 함수는 NumPy ndarray 객체에 저장된 색상 정보를 변경할 때 사용하는 함수임
- `cvtColor()` 함수에 전달하는 인자는 차례대로 입력 영상, 컬러 변환 코드임
- 앞 소스 코드에서 사용한 컬러 변환 코드 `COLOR_BGR2GRAY`는 BGR 3채널 컬러 영상을 1채널 그레이스케일 영상으로 변환할 때 사용함
- 3채널 컬러 영상 `img_color`을 그레이스케일 영상으로 변환하여 `img_gray`에 저장됨

❖ 영상의 밝기 조절 (1/8)

- **영상의 밝기**(**brightness**) 조절이란 영상의 전체적인 밝기를 조절하여 좀 더 밝거나 어두운 영상을 만드는 작업임
- 영상의 밝기를 조절하려면 입력 영상의 모든 픽셀에 일정 값을 더하거나 빼는 작업을 수행함
- 입력 영상의 모든 픽셀에 양수 값을 더하면 영상이 밝아지고, 반대로 양수 값을 빼면 영상이 어두워짐

▼ **그림 5-2 영상의 밝기 조절**

(a) 입력 영상 (b) 밝기 −50 조절 (c) 밝기 +50 조절

❖ 영상의 밝기 조절 (2/8)

- 영상의 밝기 조절을 수식으로 표현하면 다음과 같음

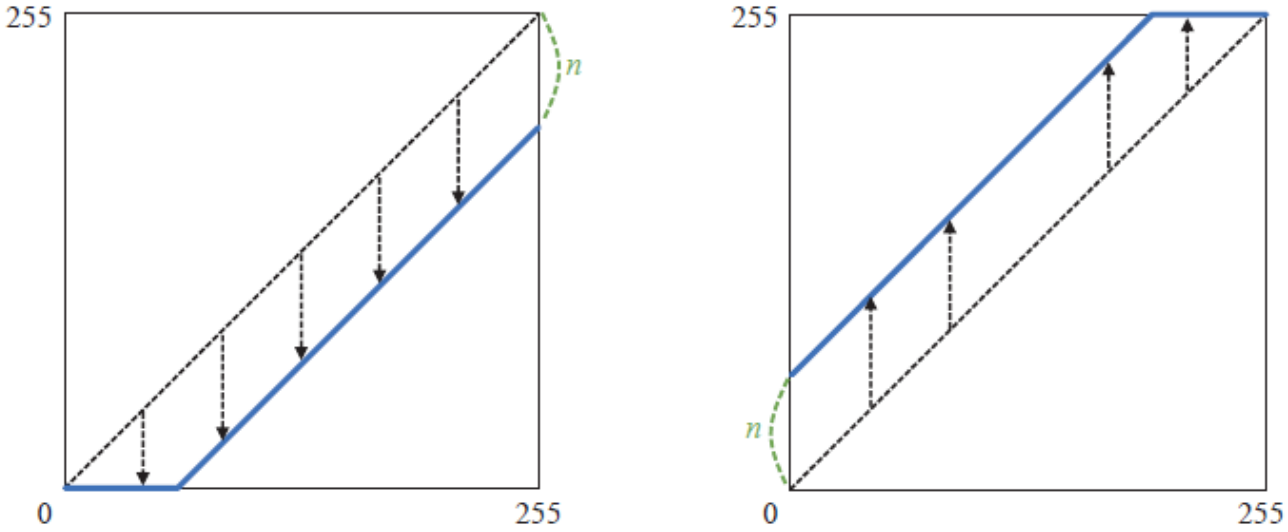
$$\mathrm{dst}(x, y) = \mathrm{src}(x, y) + n$$

- 이 수식에서 src는 입력 영상, dst는 출력 영상, n은 조절할 밝기 값을 나타냄
- n이 양수이면 출력 영상 dst의 전체적인 밝기가 증가하고,
  n이 음수이면 밝기가 감소하여 어두워짐

❖ 영상의 밝기 조절 (3/8)

- 이 그래프에서 가로축은 입력 영상 src의 그레이스케일 값을 나타냄
- 세로축은 출력 영상 dst의 그레이스케일임
- 그림 5-3(a)는 n이 음수인 경우인 경우이며 밝기가 어두워진 결과 영상이 생성됨
- 그림 5-3(b)는 n이 양수인 경우이며 결과 영상의 밝기가 밝아지는 그래프임

▼ **그림 5-3 영상의 밝기 조절 함수의 그래프**



(a) 밝기 감소 (b) 밝기 증가

❖ 영상의 밝기 조절 (4/8)

- 행렬의 원소 값을 설정할 때, 원소 자료형이 가질 수 있는 값의 범위를 벗어나는 경우 해당 자료형의 최솟값 또는 최댓값으로 원소 값을 설정하는 연산을 OpenCV에서 **포화**(**saturate**) **연산**이라고 부름
- grayscale 영상에 대해 포화 연산을 수식으로 나타내면 다음과 같음

$$\text{saturate}(x) = \begin{cases} 0 & x < 0 \text{ 일 때} \\ 255 & x > 255 \text{ 일 때} \\ x & \text{그 외} \end{cases}$$

- 실제로 영상의 밝기 조절을 구현할 때에는 다음과 같이 포화 연산을 함께 고려한 수식을 사용해야 함

$$\text{dst}(x, y) = \text{saturate}(\text{src}(x, y) + n)$$

❖ 영상의 밝기 조절 (5/8)

- OpenCV 라이브러리는 잘 설계되어 만들어졌기 때문에 훨씬 직관적으로 밝기를 조절할 수 있음

코드 5-1 영상의 밝기를 100만큼 증가하기 (brightness.py)

```
1   import cv2
2
3   def brightness1():
4       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
5
6       if src is None:
7           print('Image load failed!')
8           return
9
10      dst = cv2.add(src, 100)
11
12      cv2.imshow('src', src)
13      cv2.imshow('dst', dst)
14      cv2.waitKey()
15      cv2.destroyAllWindows()
16
17  if __name__ == '__main__':
18      brightness1()
```

cv2.add(): 값의 범위를 0~255까지 제한함

❖ 영상의 밝기 조절 (6/8)

- brightness.py 소스 코드 설명
  - 4행 lenna.bmp 레나 영상을 grayscale 형식으로 불러와 src에 저장합니다.
  - 6~8행 lenna.bmp 파일 불러오기가 실패하면 에러 메시지를 출력하고 종료합니다.
  - 10행 src 영상의 모든 픽셀 값을 100만큼 증가시킨 결과 영상을 dst에 저장합니다.
  - 12~14행 src와 dst 영상을 각각 새 창에 출력하고 키 입력이 있을 때까지 기다립니다.
  - 15행 영상 출력 창을 모두 닫습니다.

❖ 영상의 밝기 조절 (7/8)

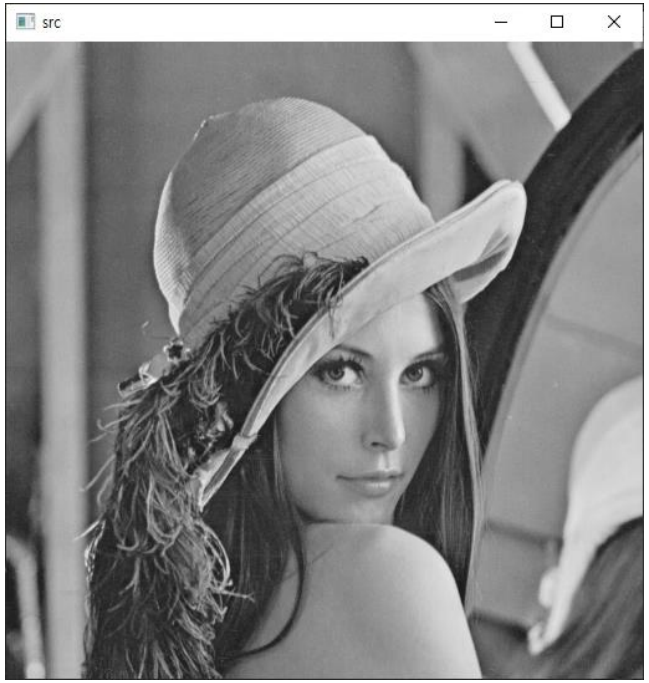
- src 창은 입력 레나 영상이고, dst 창은 밝기가 100만큼 증가된 결과 영상을 나타냄
- 입력 영상보다 결과 영상이 전체적으로 밝아진 것을 확인할 수 있음

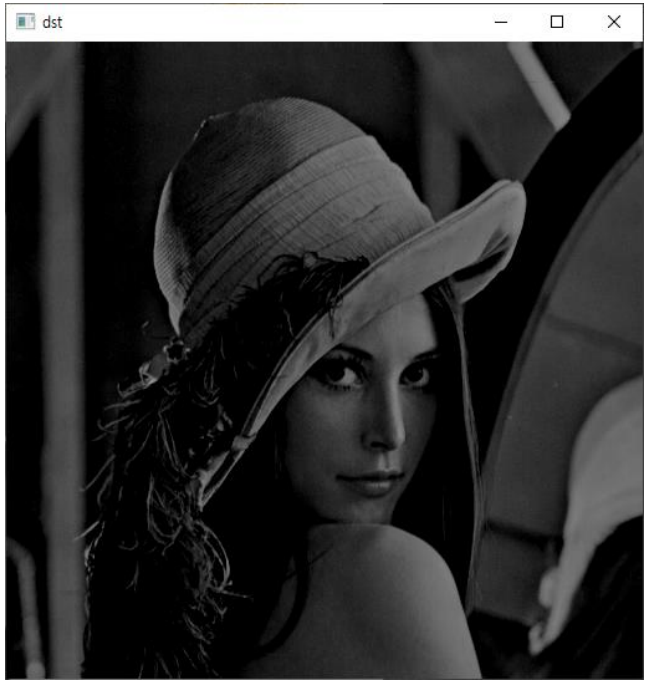
▼ **그림 5-4 영상의 밝기를 100만큼 증가하기 예제 실행 결과**

❖ 영상의 밝기 조절 (8/8)

- 만약 영상을 전체적으로 어둡게 만들고 싶다면 덧셈 대신 뺄셈을 하면 됨
- 다음은 입력 영상 src보다 픽셀 값이 100만큼 어두운 결과 영상 dst를 생성하는 코드임

```
dst = cv2.subtract(src, 100)
```





❖ 영상의 밝기 조절 직접 구현하기 (1/13)

- NumPy ndarray 행렬의 원소 값 참조 방법을 사용하면 어렵지 않게 밝기 조절을 직접 구현할 수 있음
- 입력 영상의 모든 픽셀을 방문하면서 픽셀 값에 일정한 상수를 더하거나 빼면 밝기 조절이 적용됨

❖ 영상의 밝기 조절 직접 구현하기 (2/13)

코드 5-2 포화 연산을 고려하지 않은 영상의 밝기 증가 직접 구현 (brightness.py)

```
1   import numpy as np
2   import cv2
3
4   def brightness2():
5       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
6
7       if src is None:
8           print('Image load failed!')
9           return
10
11      dst = np.empty(shape=src.shape, dtype=src.dtype)
12      for y in range(src.shape[0]):
13          for x in range(src.shape[1]):
14              dst[y, x] = src[y, x] + 100
15
16      cv2.imshow('src', src)
17      cv2.imshow('dst', dst)
18      cv2.waitKey()
19      cv2.destroyAllWindows()
20
21  if __name__ == '__main__':
22      brightness2()
```

print(src[y, x].dtype) → uint8

❖ 영상의 밝기 조절 직접 구현하기 (3/13)

- brightness.py 소스 코드 설명
  - 5행 lenna.bmp 레나 영상을 grayscale 형식으로 불러와 src에 저장합니다.
  - 11행 입력 영상 src와 크기, 타입이 같은 결과 영상 dst를 생성합니다.
  - 12~14행 영상 전체를 스캔하면서 입력 영상의 픽셀 값에 100을 더하여 결과 영상 픽셀 값으로 설정합니다.

❖ 영상의 밝기 조절 직접 구현하기 (4/13)

- 입력 영상 src와 크기 및 타입이 같은 dst 객체를 미리 생성함

```
dst = np.empty(shape=src.shape, dtype=src.dtype)
```

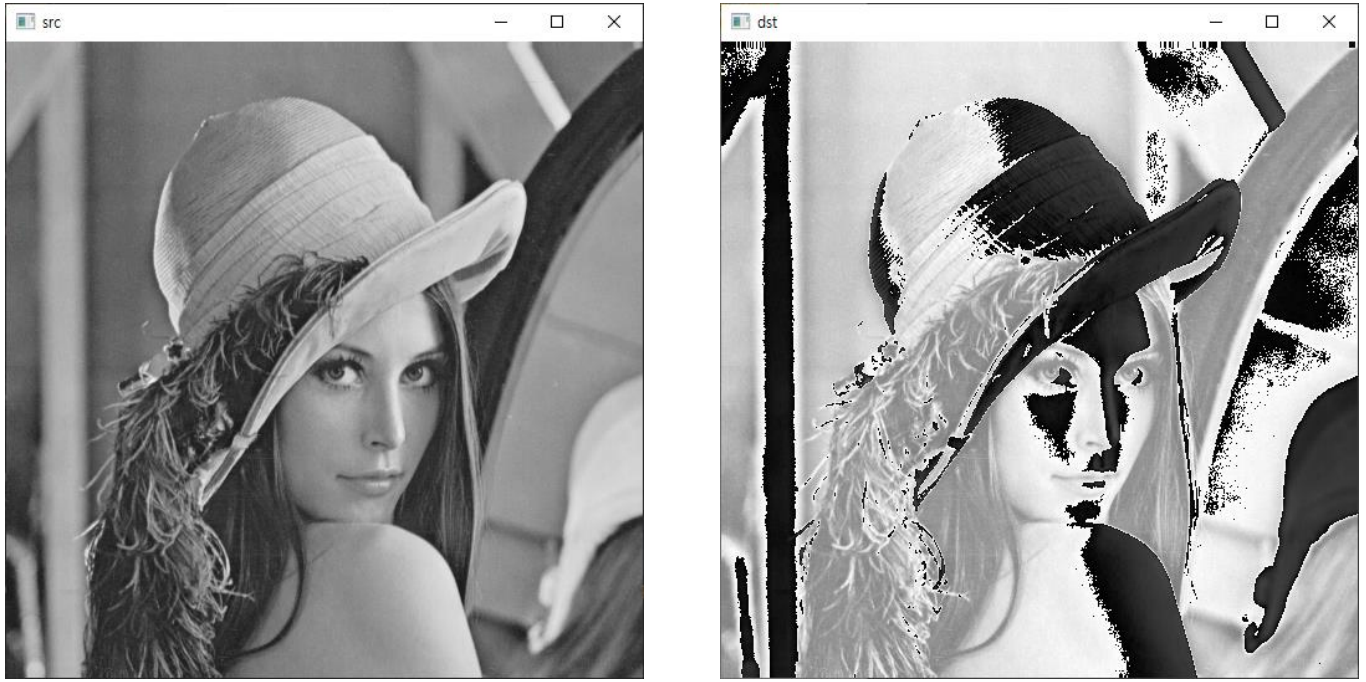
- 밝기 조절을 직접 구현하기 위해 이중 for 반복문을 이용하여 영상 전체를 스캔함
- for 반복문 안에서 다음 코드를 이용하여 결과 영상의 픽셀 값을 설정함

```
dst[y, x] = src[y, x] + 100
```

❖ 영상의 밝기 조절 직접 구현하기 (5/13)

- 그렇다면 코드 5-2의 `brightness2()` 함수를 실행하면 어떻게 될까요?
- 실제로 `brightness2()` 함수를 실행한 결과를 그림 5-5에 나타냄

▼ **그림 5-5 포화 연산을 고려하지 않은 영상의 밝기 증가 직접 구현 실행 결과**



❖ 영상의 밝기 조절 직접 구현하기 (6/13)

- 밝은 픽셀 주변에서 급격하게 어두운 픽셀이 나타나는 것은 앞에서 설명했던 **포화 연산을 수행하지 않았기 때문에 발생하는 현상**임
- 그림 5-5와 같은 결과가 나타난 원인을 알아보기 위해 다음 코드를 추가해 보자

```
print(dst[y, x].dtype)
```

- 위 코드의 출력 결과는 uint8임

❖ 영상의 밝기 조절 직접 구현하기 (7/13)

- 정수 256을 16진수로 표현하면 0x100이고, 이 값을 uint8 자료형에 대입하려고 하면 하위 1바이트(byte)만 대입되기 때문에 변수에는 0x00이 저장됨
- 마찬가지 방식으로 uint8 자료형의 변수에 257을 대입하려고 하면 실제로는 1이 저장됨
- 입력 영상의 픽셀 중에서 밝기 100을 더하여 255보다 큰 값이 되는 픽셀은 오히려 픽셀 값이 0에 가까운 어두운 픽셀로 바뀌게 되는 것임
- 위 내용이 사실인지, 이중 for 반복문 안에 다음 코드를 추가해서 확인해 보자

```
if (src[y, x] + 100 > 255):
    print(src[y, x] + 100)
    print(dst[y, x])
```

❖ 영상의 밝기 조절 직접 구현하기 (8/13)

- 그렇다면 어떻게 해야 정상적인 밝기 조절 결과 영상을 얻을 수 있을까요?
- 코드 5-2에서 12~14행 이중 for 반복문을 다음과 같이 변경하면 정상적인 밝기 조절 결과 영상을 만들 수 있음

```
for y in range(src.shape[0]):
    for x in range(src.shape[1]):
        if src[y, x] + 100 >= 255:
            dst[y, x] = 255
        elif src[y, x] + 100 < 0:
            dst[y, x] = 0
        else:
            dst[y, x] = src[y, x] + 100
```

❖ 영상의 밝기 조절 직접 구현하기 (9/13)

- grayscale 값 범위에 맞게끔 결과 영상 픽셀 값을 설정하는 작업은 컴퓨터 비전 프로그래밍에서는 매우 빈번하게 일어남
- 함수로 만들어서 사용하는 것이 함수의 재사용 측면에서 좋은 방법임

❖ 영상의 밝기 조절 직접 구현하기 (10/13)

- saturated() 함수를 정의하여 영상의 밝기를 직접 조절하는 소스 코드를 코드 5-3에 나타냄
- 코드 5-3에 나타난 brightness3() 함수는 앞서 설명한 brightness2() 함수에서 포화 연산이 추가된 코드임

코드 5-3 포화 연산을 고려한 영상의 밝기 증가 직접 구현 (brightness.py)

```
1   import numpy as np
2   import cv2
3
4   def saturated(value):
5       if value > 255:
6           value = 255
7       elif value < 0:
8           value = 0
9
10      return value
11
```

❖ 영상의 밝기 조절 직접 구현하기 (11/13)

코드 5-3 포화 연산을 고려한 영상의 밝기 증가 직접 구현 (brightness.py)

```
12  def brightness3():
13      src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
14
15      if src is None:
16          print('Image load failed!')
17          return
18
19      dst = np.empty(shape=src.shape, dtype=src.dtype)
20      for y in range(src.shape[0]):
21          for x in range(src.shape[1]):
22              dst[y, x] = saturated(src[y, x] + 100)
23
24      cv2.imshow('src', src)
25      cv2.imshow('dst', dst)
26      cv2.waitKey()
27      cv2.destroyAllWindows()
28
29  if __name__ == '__main__':
30      brightness3()
```

❖ 영상의 밝기 조절 직접 구현하기 (12/13)

- brightness.py 소스 코드 설명

  - 22행 밝기 조절된 픽셀 값에 `saturated()` 함수를 이용하여 포화 연산을 수행한 후 결과 영상 픽셀 값으로 설정합니다.

❖ 영상의 밝기 조절 직접 구현하기 (13/13)

- 일반적으로 밝기 조절과 같은 작업을 수행할 때 OpenCV에서 제공하는 `add()` 또는 `subtract()` 함수를 사용하여 코드를 작성하는 것이 더욱 빠르고 간편함
- OpenCV에서 제공하는 함수를 사용하면 CPU 최적화 및 병렬 처리를 수행하기 때문에 빠르게 동작하고, 소스 코드 가독성도 높은 편임
- 다만 컴퓨터 비전 프로젝트를 수행하다 보면 OpenCV에서 지원하지 않는 새로운 기능을 직접 구현해야 하는 경우가 발생하기 때문에 영상의 픽셀 값을 직접 참조하고 변경하는 방법은 반드시 기억하고 있어야 함
- 그리고 이 경우, 포화 연산을 위한 `saturated()` 함수 필요성을 인지해 두기 바람

❖ 트랙바를 이용한 영상의 밝기 조절 (1/4)

- OpenCV 라이브러리를 사용하면 영상의 밝기를 변경하기 위해서 소스 코드 수정을 여러 번 하는 것이 불편하다면 밝기 조절 프로그램에 트랙바를 부착함
- 프로그램 동작 중 트랙바로 밝기를 조정하여 곧바로 그 결과를 확인할 수 있어 좋음

❖ 트랙바를 이용한 영상의 밝기 조절 (2/4)

- 결과 영상 출력 창에 트랙바를 부착하고, 트랙바가 움직일 때 트랙바 위치만큼의 밝기를 조절하는 소스 코드를 코드 5-4에 나타냄

코드 5-4 트랙바를 이용한 영상의 밝기 조절하기 (brightness.py)

```
1   import cv2
2
3   def brightness4():
4       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
5
6       if src is None:
7           print('Image load failed!')
8           return
9
10      def update(pos):
11          dst = cv2.add(src, pos)
12          cv2.imshow('dst', dst)
13
14      cv2.namedWindow('dst')
15      cv2.createTrackbar('Brightness', 'dst', 0, 100, update)
16      update(0)
17
18      cv2.waitKey()
19      cv2.destroyAllWindows()
```

cv2.createTrackbar('트랙바 이름', '창 이름', 최솟값, 최댓값, 콜백 함수)

❖ 트랙바를 이용한 영상의 밝기 조절 (3/4)

코드 5-4 트랙바를 이용한 영상의 밝기 조절하기 (brightness.py)

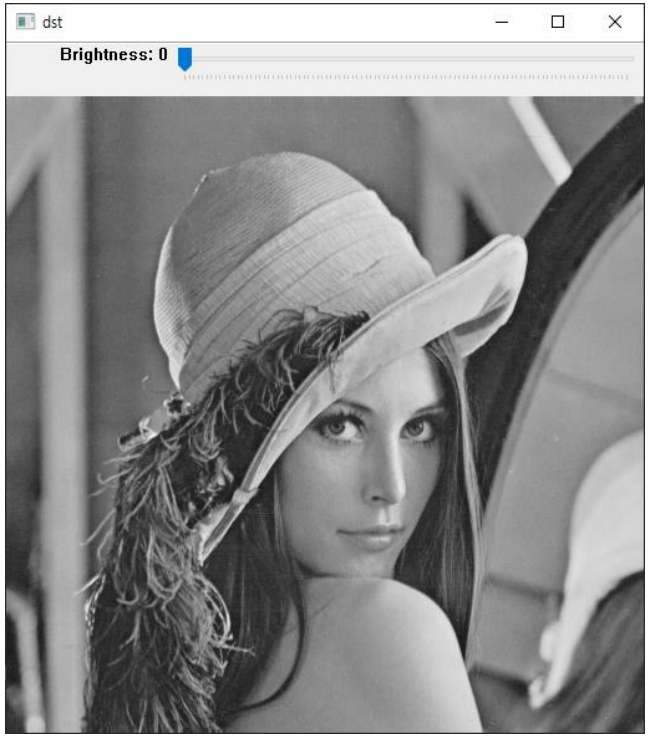
```
20
21  if __name__ == '__main__':
22      copy_method()
```

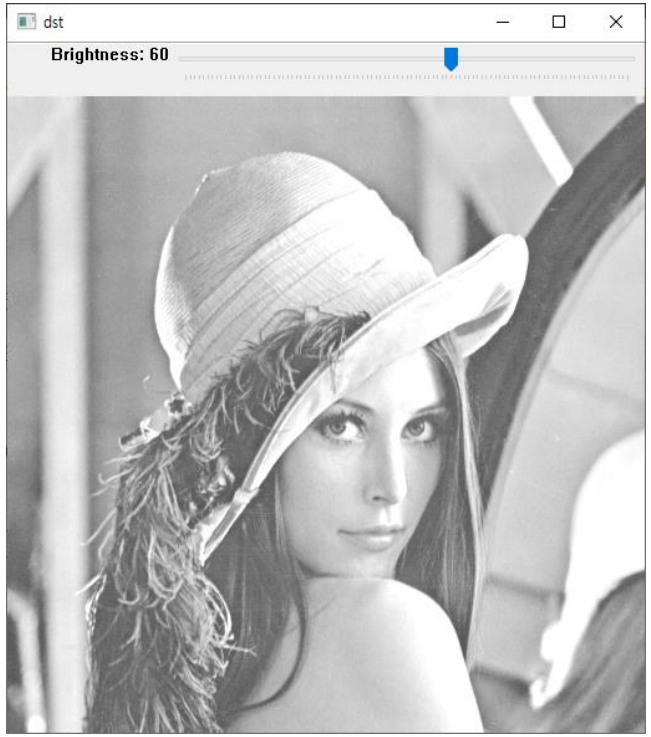
- brightness.py 소스 코드 설명
  - 14행 결과 영상을 출력하고 트랙바를 부착할 dst 창을 미리 생성합니다.
  - 10~12행 트랙바 콜백 함수에서 밝기 조절된 결과 영상 dst를 화면에 출력합니다.
  - 15행 dst 창에 트랙바를 부착하고 콜백 함수 update를 등록합니다.
    `cv2.createTrackbar('트랙바 이름', '창 이름', 최솟값, 최댓값, 콜백 함수)`
  - 16행 프로그램 실행 시 dst 창에 레나 영상이 정상적으로 표시되도록 강제로 `update()` 함수를 호출합니다.

❖ 트랙바를 이용한 영상의 밝기 조절 (4/4)

- 만약 마우스를 이용하여 트랙바의 위치를 변경하면 그림 5-6의 오른쪽 그림처럼 트랙바 위치에 해당하는 밝기만큼 밝아진 결과 영상이 dst 창에 나타나게 됨

▼ **그림 5-6 트랙바를 이용한 영상의 밝기 조절 결과**





5.2 영상의 명암비 조절

5.1 영상의 밝기 조절

5.3 히스토그램 분석

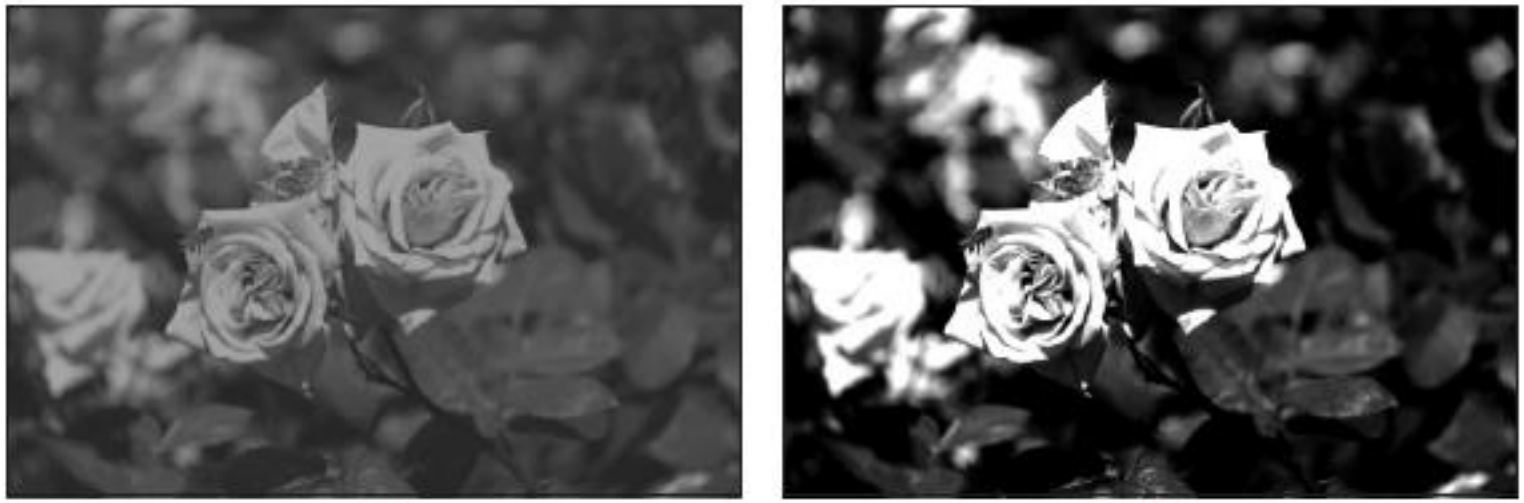
❖ 기본적인 명암비 조절 방법 (1/6)

- **명암비**란 영상에서 밝은 영역과 어두운 영역 사이에 드러나는 **밝기 차이의 강도를 의미**함
- **명암 대비** 또는 **콘트라스트**(**contrast**)라고도 함
- 영상이 전반적으로 어둡거나 또는 전반적으로 밝은 픽셀로만 구성된 경우, 명암비가 낮다고 표현함
- 반면에 밝은 영역과 어두운 영역이 골고루 섞여 있는 영상은 명암비가 높다고 말함
- 일반적으로 명암비가 낮은 영상은 객체 간의 구분이 잘 되지 않아서 전반적으로 흐릿하게 느껴짐
- **명암비가 높은 영상은 사물의 구분이 잘 되며 선명한 느낌을 줌**

❖ 기본적인 명암비 조절 방법 (2/6)

- 그림 5-7은 서로 다른 명암비를 갖는 영상의 예임

▼ **그림 5-7 서로 다른 명암비를 갖는 영상**



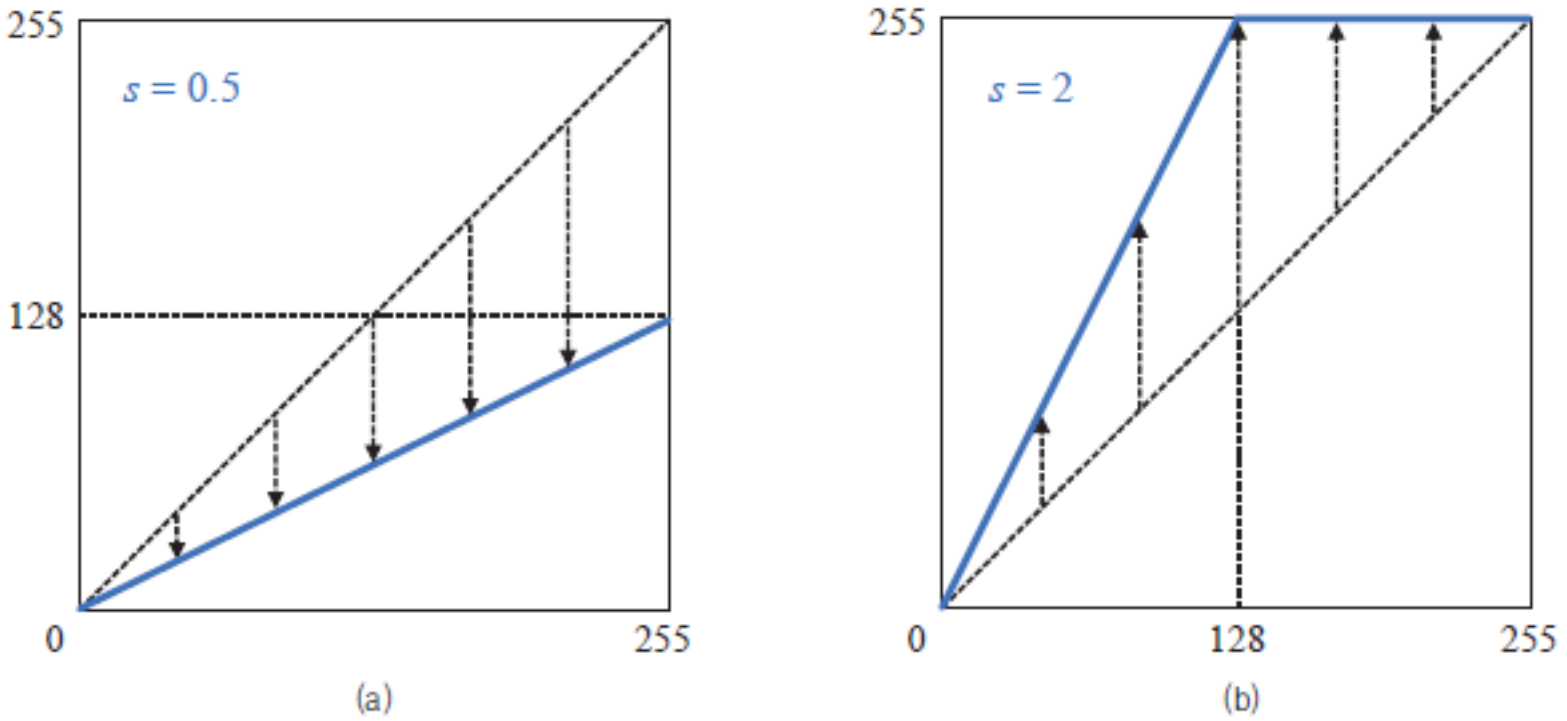
(a) (b)

❖ 기본적인 명암비 조절 방법 (3/6)

$$\mathrm{dst}(x,y) = \mathrm{saturate}(s \times \mathrm{src}(x,y))$$

- $s = 0.5$인 경우와 $s = 2$인 경우의 그래프를 그림 5-8에 나타냄

▼ **그림 5-8 기본적인 영상의 명암비 조절 함수 그래프**



❖ 기본적인 명암비 조절 방법 (4/6)

- 코드 5-5의 contrast1() 함수는 입력 영상의 모든 픽셀 값에 2를 곱하여 결과 영상을 생성하고 화면에 출력함

코드 5-5 기본적인 영상의 명암비 증가 예제 (contrast.py)

```
1   import cv2
2
3   def contrast1():
4       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
5
6       if src is None:
7           print('Image load failed!')
8           return
9
10      s = 2.0
11      dst = cv2.multiply(src, s)
12
13      cv2.imshow('src', src)
14      cv2.imshow('dst', dst)
15      cv2.waitKey()
16      cv2.destroyAllWindows()
17
18  if __name__ == '__main__':
19      contrast1()
```

cv2.multiply(): 값의 범위를 0~255까지 제한함

❖ 기본적인 명암비 조절 방법 (5/6)

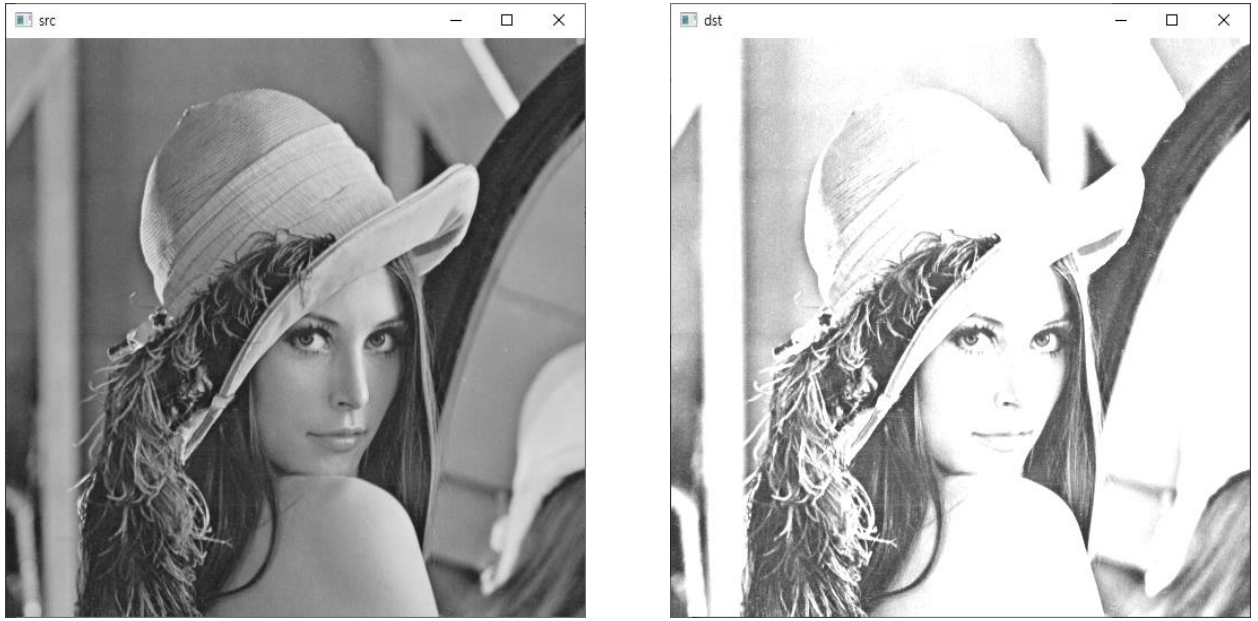
- contrast.py 소스 코드 설명

  - 10~11행 입력 영상 `src`의 모든 픽셀 값에 2.0을 곱하여 결과 영상 `dst`를 생성합니다. 이때 포화 연산도 함께 수행합니다.

❖ 기본적인 명암비 조절 방법 (6/6)

- dst 영상을 보면 전체적으로 픽셀 값이 포화되어 흰색으로 나타나는 영역이 너무 많으며, 이로 인해 사물의 윤곽 구분이 더 어려워짐
- 사실상 픽셀 값에 일정 상수를 단순히 곱하여 명암비를 조절하는 방식은 실전에서는 잘 사용되지 않음

▼ **그림 5-9 기본적인 영상의 명암비 증가 예제 실행 결과**



❖ 효과적인 명암비 조절 방법 (1/8)

- 명암비를 효과적으로 높이기 위해서는 밝은 픽셀은 더욱 밝게, 어두운 픽셀은 더욱 어두워지게 변경해야 함
- 이때 픽셀 값이 밝고 어둡다는 기준을 어떻게 설정할 것인지가 명암비 조절 결과 영상의 품질 차이를 가져올 수 있음
- grayscale 범위 중간값인 128을 기준으로 설정할 수도 있고, 입력 영상의 평균 밝기를 구하여 기준으로 삼을 수도 있음

❖ 효과적인 명암비 조절 방법 (2/8)

- grayscale 범위 중간값인 128을 기준으로 명암비를 조절하는 방법을 구현해 보자
- 입력 영상의 픽셀 값이 128보다 크면 더욱 밝게 만들고, 128보다 작으면 픽셀 값을 더 작게 만드는 방식임
- 반대로 명암비를 감소시키려면 128보다 큰 입력 영상 픽셀 값은 좀 더 작게 만들고, 128보다 작은 픽셀 값은 오히려 128에 가깝게 증가시킴

❖ 효과적인 명암비 조절 방법 (3/8)

- 픽셀 값 변경 방식을 수식으로 정리하면 다음과 같음

$$\mathrm{dst}(x, y) = \mathrm{src}(x, y) + (\mathrm{src}(x, y) - 128) \cdot \alpha$$

α는 -1보다 같거나 큰 실수입니다.

- 이 수식은 항상 (128, 128) 좌표를 지나가고, α에 의해 기울기가 변경되는 직선의 방정식임
- α의 범위가 $0 \leq \alpha \leq 1$이면 기울기가 0부터 1 사이의 직선이 되며, 이는 명암비를 감소시키는 변환 함수임
- 반면에 α의 범위가 $0 < \alpha$이면 기울기가 1보다 큰 직선의 방정식이며, 이는 명암비를 증가시키는 변환 함수임

❖ 효과적인 명암비 조절 방법 (4/8)

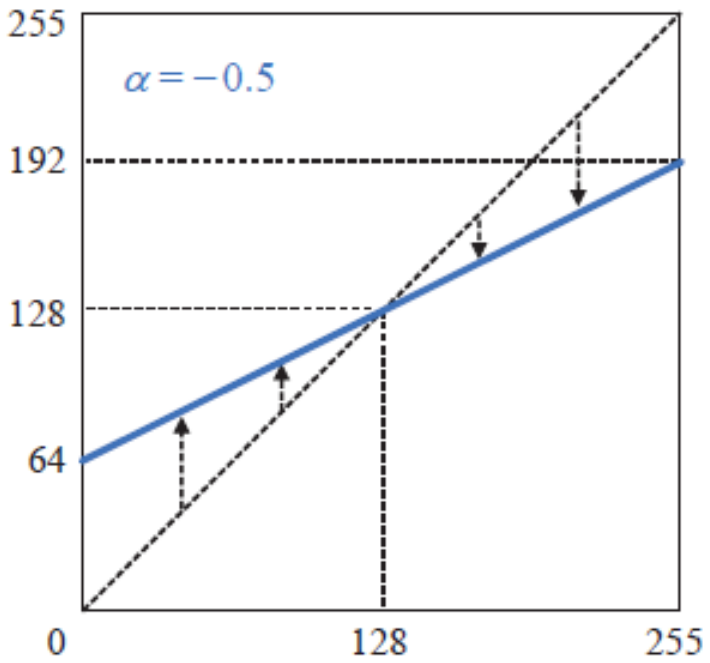
- 앞 수식에 의해 계산되는 결과 영상의 픽셀 값은 0보다 작거나 255보다 커지는 경우가 발생할 수 있으므로 포화 연산도 함께 수행해야 함
- 포화 연산까지 포함한 효과적인 명암비 조절 수식은 다음과 같음

$$\mathrm{dst}(x, y) = \mathrm{saturate}(\mathrm{src}(x, y) + (\mathrm{src}(x, y) - 128) \cdot \alpha)$$

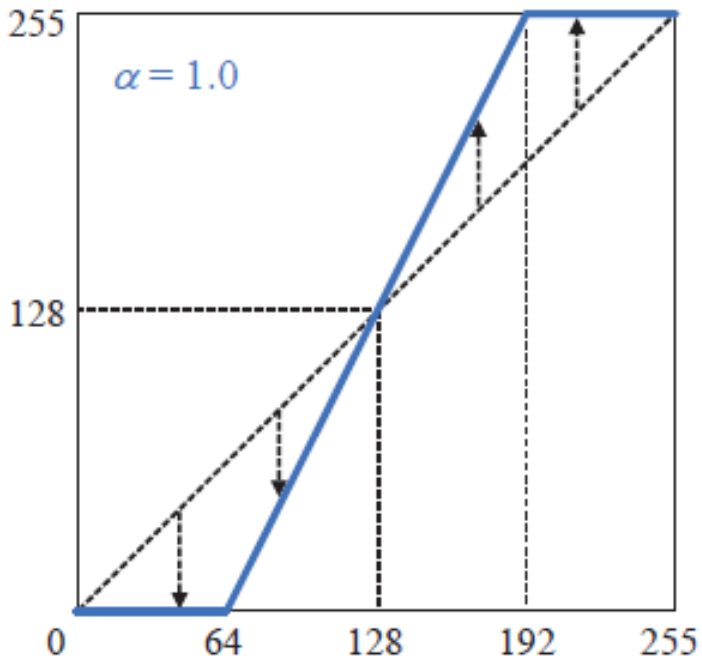
❖ 효과적인 명암비 조절 방법 (5/8)

- 이 수식에서 $\alpha = -0.5$인 경우와 $\alpha = 1.0$인 경우의 함수 그래프를 그림 5-10에 나타냄

▼ 그림 5-10 효과적인 영상의 명암비 조절 함수 그래프



(a) 명암비 감소(α = −0.5)



(b) 명암비 증가(α = 1.0)

❖ 효과적인 명암비 조절 방법 (6/8)

- 코드 5-6의 contrast2() 함수에서는 α 값 1.0을 사용하여 레나 영상의 명암비를 증가시킴

코드 5-6 효과적인 영상의 명암비 조절 방법 (contrast.py)

```
1   import numpy as np
2   import cv2
3
4   def contrast2():
5       src = cv2.imread('lenna.bmp', cv2.IMREAD_GRAYSCALE)
6
7       if src is None:
8           print('Image load failed!')
9           return
10
11      # dst(x,y) = src(x,y) + (src(x,y) - 128)*alpha
12      alpha = 1.0
13      dst = np.clip(src + (src - 128.)*alpha, 0, 255).astype(np.uint8)
14
15      cv2.imshow('src', src)
16      cv2.imshow('dst', dst)
17      cv2.waitKey()
18      cv2.destroyAllWindows()
```

❖ 효과적인 명암비 조절 방법 (7/8)

코드 5-6 효과적인 영상의 명암비 조절 방법 (contrast.py)

```
19
20  if __name__ == '__main__':
21      contrast2()
```

- contrast.py 소스 코드 설명

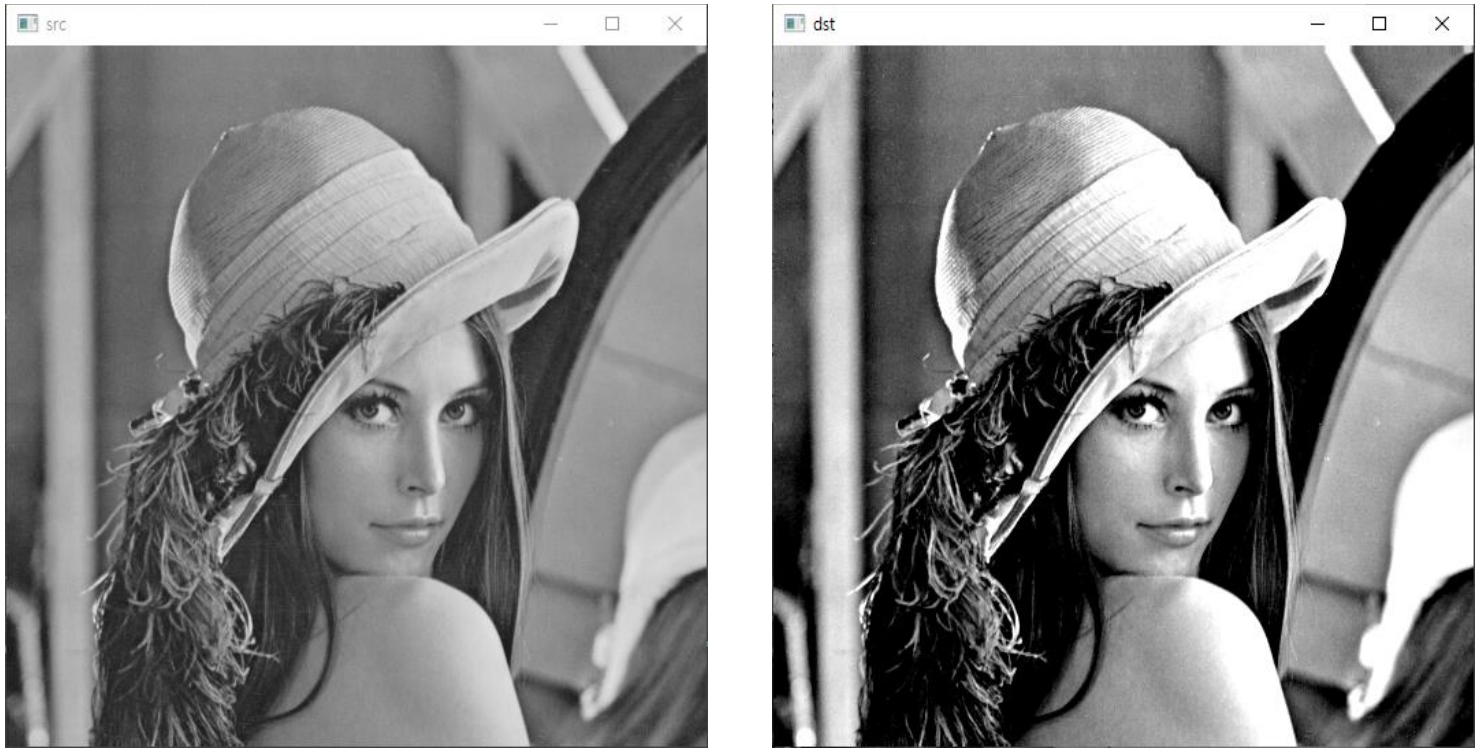
  - 12~13행 효과적인 영상의 명암비 조절 수식을 그대로 소스 코드 형태로 변환한 것이며, 입력 영상 src로부터 명암비가 증가된 결과 영상 dst를 생성합니다.

    np.clip(array, min, max)
    array 내 원소들에 대해서 min 값보다 작은 원소들은 min 값으로 변경하고,
    max 값보다 큰 원소들은 max 값으로 변경합니다.

❖ 효과적인 명암비 조절 방법 (8/8)

- 앞서 그림 5-9에서 보았던 기본적인 명암비 증가 예제보다 그림 5-11에 나타난 결과 화면이 좀 더 자연스럽게 명암비가 증가되었음을 확인할 수 있음

▼ 그림 5-11 효과적인 영상의 명암비 증가 예제 실행 결과



5.3 히스토그램 분석

5.1 영상의 밝기 조절
5.2 영상의 명암비 조절

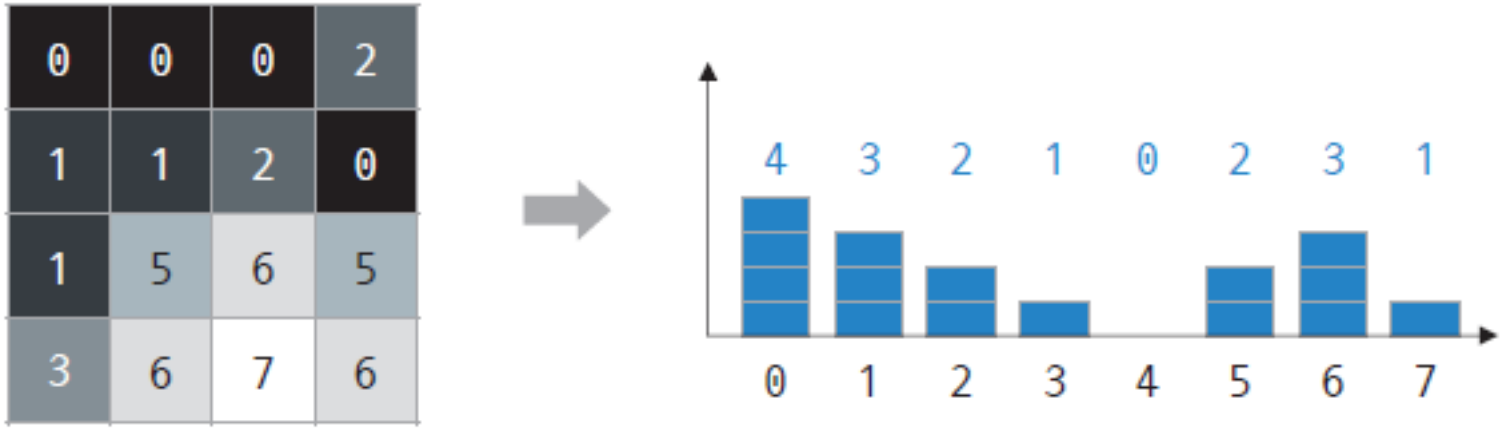
❖ 히스토그램 구하기 (1/19)

- 영상의 **히스토그램**(**histogram**)이란 영상의 픽셀 값 분포를 그래프 형태로 표현한 것을 의미함
- grayscale 영상의 경우, 각 grayscale 값에 해당하는 픽셀의 개수를 구하고 이를 막대그래프 형태로 표현함으로써 히스토그램을 구할 수 있음
- 컬러 영상에 대해서도 세 개의 색상 성분 조합에 따른 픽셀 개수를 계산하여 히스토그램을 구할 수 있음

❖ 히스토그램 구하기 (2/19)

- 그림 5-12 왼쪽에 나타난 4×4 입력 영상은 각 픽셀이 0부터 7 사이의 밝기를 가질 수 있는 단순한 형태의 영상임
- 각각의 밝기에 해당하는 픽셀 개수를 세어서 막대그래프 형태로 표현한 히스토그램을 그림 5-12 오른쪽에 나타냄

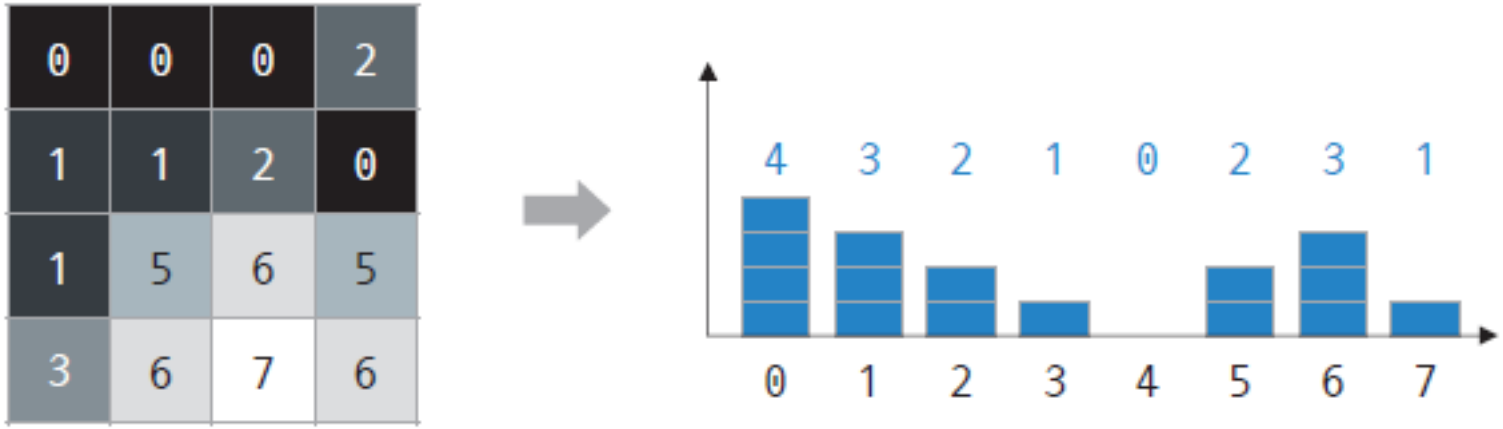
▼ 그림 5-12 단순한 영상에서 히스토그램 구하기



❖ 히스토그램 구하기 (3/19)

- 히스토그램 그래프에서 가로축을 히스토그램의 **빈**(**bin**)이라고 함
- 그림 5-12에 나타난 히스토그램에서 빈 개수는 8임
- 그림 5-12에서 사용된 영상이 0부터 7 사이의 픽셀 값을 가질 수 있기 때문에 여덟 개의 빈으로 구성된 히스토그램을 생성한 것임
- grayscale 영상의 경우에는 256개의 빈을 갖는 히스토그램을 구하는 것이 일반적임
- 경우에 따라서 히스토그램의 빈 개수를 픽셀 값 범위보다 작게 설정할 수도 있음

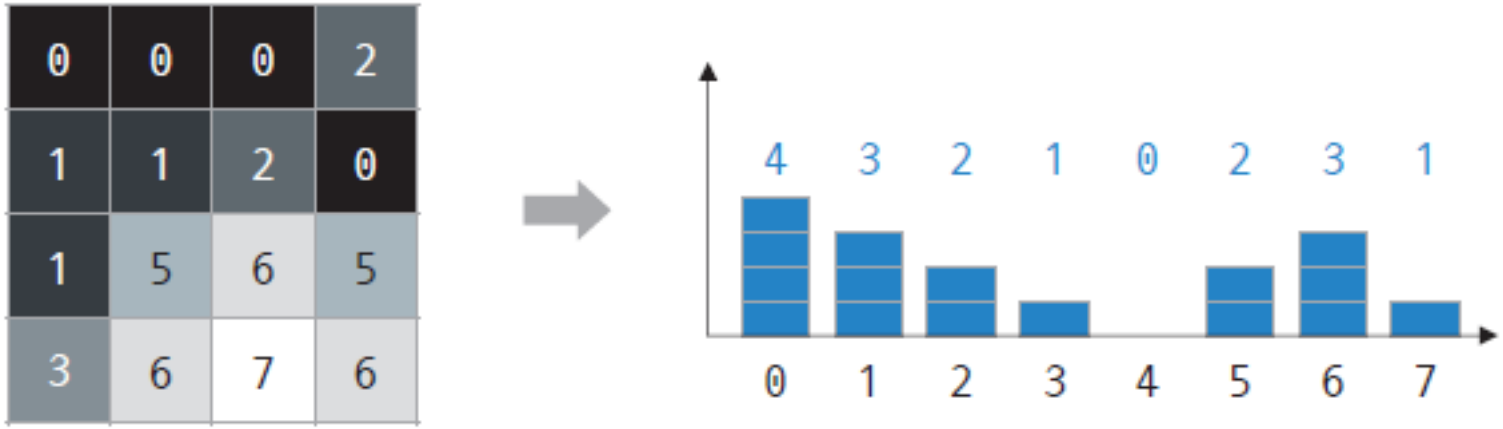
▼ 그림 5-12 단순한 영상에서 히스토그램 구하기



❖ 히스토그램 구하기 (4/19)

- 일반적으로 히스토그램의 빈 개수가 줄어들면 히스토그램이 표현하는 영상의 픽셀 값 분포 모양이 좀 더 대략적인 형태로 바뀜
- 반대로 **빈 개수가 많으면 세밀한 픽셀 값 분포 표현이 가능함**

▼ 그림 5-12 단순한 영상에서 히스토그램 구하기



❖ 히스토그램 구하기 (5/19)

- OpenCV에서 영상의 히스토그램을 구하려면 `calcHist()` 함수를 사용함
- `calcHist()` 함수는 한 장의 영상뿐만 아니라 여러 장의 영상으로부터 히스토그램을 구할 수 있음
- 여러 채널로부터 히스토그램을 구할 수도 있음
- 히스토그램 빈 개수도 조절할 수 있음

❖ 히스토그램 구하기 (6/19)

- `calcHist()` 함수 원형과 인자에 대한 설명은 다음과 같음

```
cv2.calcHist(images, channels, mask, histSize, ranges)
```

`images`	입력 영상의 배열
`channels`	히스토그램을 구할 채널을 나타내는 리스트 grayscale 영상인 경우: [0] color 영상에서는 Blue는 0, Green은 1, Red는 2를 의미함 B 영상인 경우: [0], G 영상인 경우: [1], R 영상인 경우 [2]라고 적음
`mask`	영상의 분석 영역. None이면 영상 전체 영역.
`histSize`	각 차원의 히스토그램 빈(Bin) 개수를 나타냄
`ranges`	각 차원의 히스토그램 범위

❖ 히스토그램 구하기 (7/19)

- grayscale 입력 영상으로부터 256개의 빈(bins)으로 구성된 히스토그램을 생성하는 사용자 정의 함수 `calcGrayHist()`를 코드 5-7에 나타냄
- `calcGrayHist()` 함수는 내부적으로 `calcHist()` 함수를 호출하여 히스토그램을 계산함
- 이때 `calcHist()` 함수 인자를 어떻게 설정하는지를 주의 깊게 살펴보기 바람

❖ 히스토그램 구하기 (8/19)

코드 5-7 grayscale 영상의 히스토그램 구하기 (histogram.py)

```
1   import cv2
2
3   def calcGrayHist(img):
4       channels = [0]
5       histSize = [256]
6       histRange = [0, 256]
7
8       hist = cv2.calcHist([img], channels, None, histSize, histRange)
9
10      return hist
```

❖ 히스토그램 구하기 (9/19)

- histogram.py 소스 코드 설명

  - 4~6행 8행에서 호출하는 `calcHist()` 함수에 전달할 인자를 생성하는 구문입니다.
  - 4행 히스토그램을 구할 채널 번호를 담은 `channels` 리스트를 생성합니다.
    grayscale 영상은 한 개의 채널을 가지고 있고, 채널 번호는 0부터 시작하므로 `channels` 리스트는 0 하나만 원소로 가집니다.
  - 5행 `histSize` 배열 원소에 256을 하나 지정한다는 것은 입력 영상의 첫 번째 채널 값의 범위를 256개 빈(bins)으로 나누어 히스토그램을 구하겠다는 의미입니다.
  - 6행 `ranges` 리스트에는 grayscale 값의 최솟값과 최댓값인 0과 256을 차례대로 지정합니다.
  - 8행 `calcHist()` 함수를 이용하여 `img` 영상의 히스토그램을 구하고, 그 결과를 `hist` 변수에 저장합니다.
  - 10행 구해진 히스토그램 `hist`를 반환합니다.

❖ 히스토그램 구하기 (10/19)

- `calcGrayHist()` 함수는 내부에서 OpenCV 함수 `calcHist()`를 이용하여 grayscale 영상의 히스토그램을 표현하는 행렬 `hist`를 구하여 반환함
- 이때 반환되는 `hist`는 256×1 크기의 행렬임
- 즉 `hist` 행렬의 행 개수는 256이고, 열 개수는 1임

❖ 히스토그램 구하기 (11/19)

- `calcGrayHist()` 함수로 구한 히스토그램 행렬을 막대그래프 형태로 나타내려면 직접 `hist` 행렬을 참조하여 막대그래프 영상을 생성해야 함
- `getGrayHistImage()` 함수를 만들어서 히스토그램 그래프에서 최대 빈도수를 표현하는 막대그래프 길이가 100픽셀이 되도록 그래프를 그리자

❖ 히스토그램 구하기 (12/19)

코드 5-8 grayscale 영상의 히스토그램 그래프 그리기 (histogram.py)

```
1   import numpy as np
2   import cv2
3
4   def getGrayHistImage(hist):
5       histMax = np.max(hist)
6
7       imgHist = np.full((100, 256), 255, dtype=np.uint8)
8       for x in range(256):
9           pt1 = (x, 100)
10          pt2 = (x, 100 - int(hist[x, 0] * 100 / histMax))
11          cv2.line(imgHist, pt1, pt2, 0)
12
13      return imgHist
```

cv2.line(영상, 시작점 좌표, 종료점 좌표, 색상)

❖ 히스토그램 구하기 (13/19)

- histogram.py 소스 코드 설명

  - 5행 `hist` 행렬 원소의 최댓값을 `histMax` 변수에 저장합니다.
  - 7행 흰색으로 초기화된 100×256 크기의 새 영상 `imgHist`를 생성합니다.
  - 8~11행 for 반복문과 `line()` 함수를 이용하여 각 빈(bin)에 대한 히스토그램 그래프를 그립니다.
  - 13행 `hist` 행렬로부터 구한 100×256 크기의 히스토그램 영상 `imgHist`를 반환합니다.

❖ 히스토그램 구하기 (14/19)

- 히스토그램 행렬의 최댓값 histMax는 히스토그램 막대그래프를 그릴 때 사용함
- 히스토그램 행렬의 최댓값 위치에서 100픽셀에 해당하는 검은색 직선을 그림
- 나머지 히스토그램 막대그래프는 100픽셀보다 짧은 길이의 직선으로 표현됨

❖ 히스토그램 구하기 (15/19)

- calcGrayHist() 함수와 getGrayHistImage() 함수를 사용하여 hawkes.bmp 영상의 히스토그램을 화면에 출력하려면 다음과 같이 코드를 작성함

코드 5-8 grayscale 영상의 히스토그램 그래프 그리기 (histogram.py)

```
1  if __name__ == '__main__':
2      src = cv2.imread('hawkes.bmp', cv2.IMREAD_GRAYSCALE)
3
4      hist = calcGrayHist(src)
5      cv2.imshow('src', src)
6      cv2.imshow('srcHist', getGrayHistImage(hist))
7
8      cv2.waitKey()
9      cv2.destroyAllWindows()
```

❖ 히스토그램 구하기 (16/19)

- 앞 예제 코드에서 hist는 히스토그램 정보를 담고 있는 행렬임
- 만약 단순히 히스토그램 그래프 영상을 화면에 출력하는 것이 목적이라면 hist 변수를 선언할 필요 없이 다음과 같이 코드를 작성할 수 있음

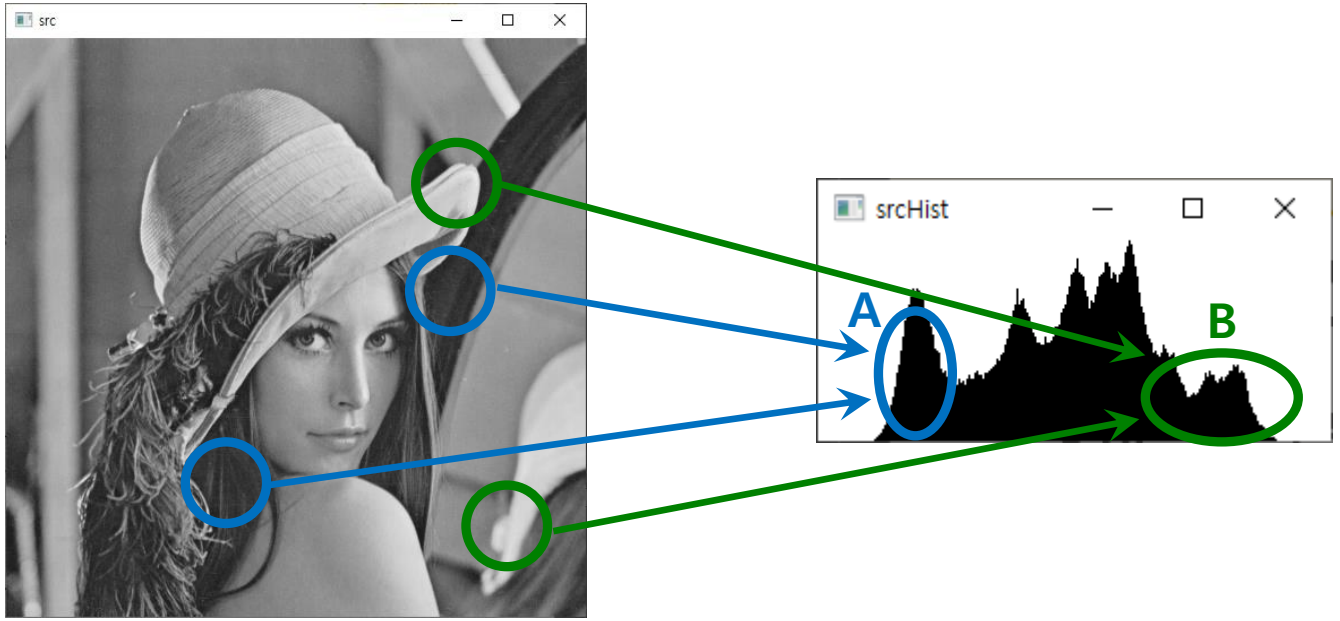
코드 5-8 grayscale 영상의 히스토그램 그래프 그리기 (histogram.py)

```
1  if __name__ == '__main__':
2      src = cv2.imread('hawkes.bmp', cv2.IMREAD_GRAYSCALE)
3
4      # hist = calcGrayHist(src)
5      cv2.imshow('src', src)
6      cv2.imshow('srcHist', getGrayHistImage(calGrayHist(src)))
7
8      cv2.waitKey()
9      cv2.destroyAllWindows()
```

❖ 히스토그램 구하기 (17/19)

- srcHist 창에 나타난 히스토그램에서 A 영역은 주로 레나의 머리카락 또는 거울 테두리 픽셀로부터 만들어진 어두운 픽셀 분포를 표현함
- 상대적으로 밝은 회색을 표현하는 B 영역은 영상에서 모자 또는 거울의 픽셀로부터 생성되었음을 가늠할 수 있음

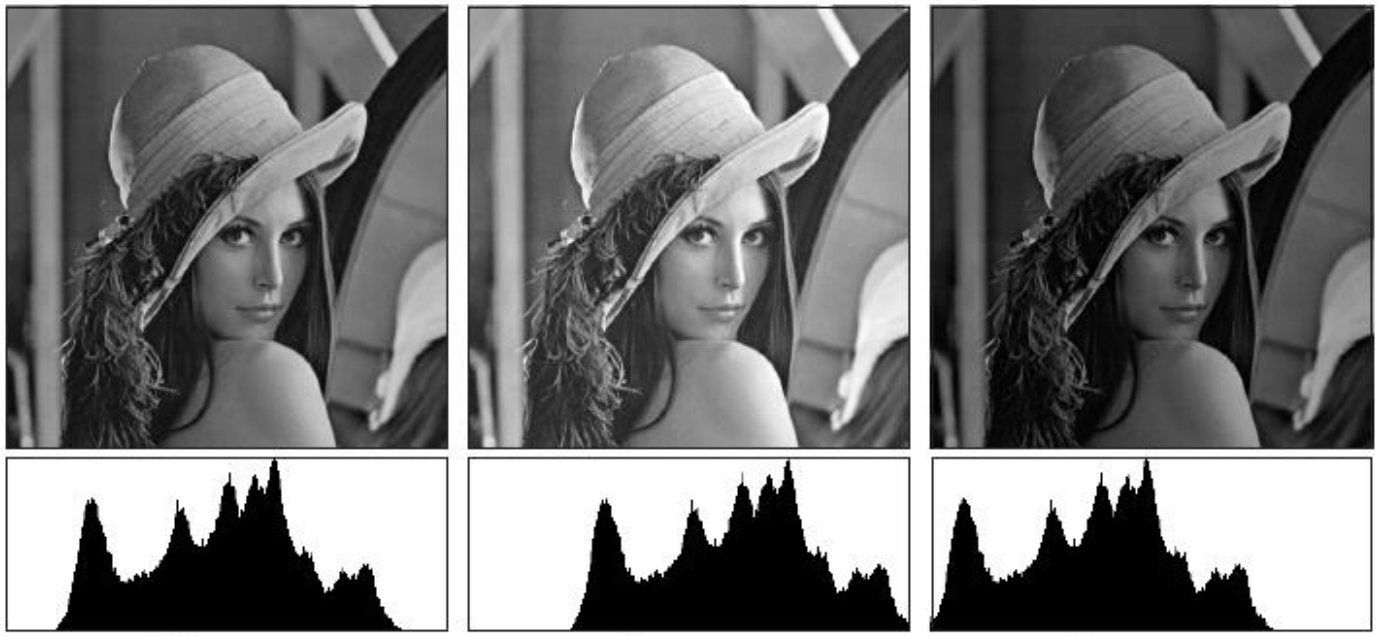
▼ 그림 5-14 영상의 히스토그램 분석



❖ 히스토그램 구하기 (18/19)

- 히스토그램의 픽셀 분포 그래프는 영상의 밝기와 명암비를 가늠할 수 있는 유용한 도구로 사용될 수 있음

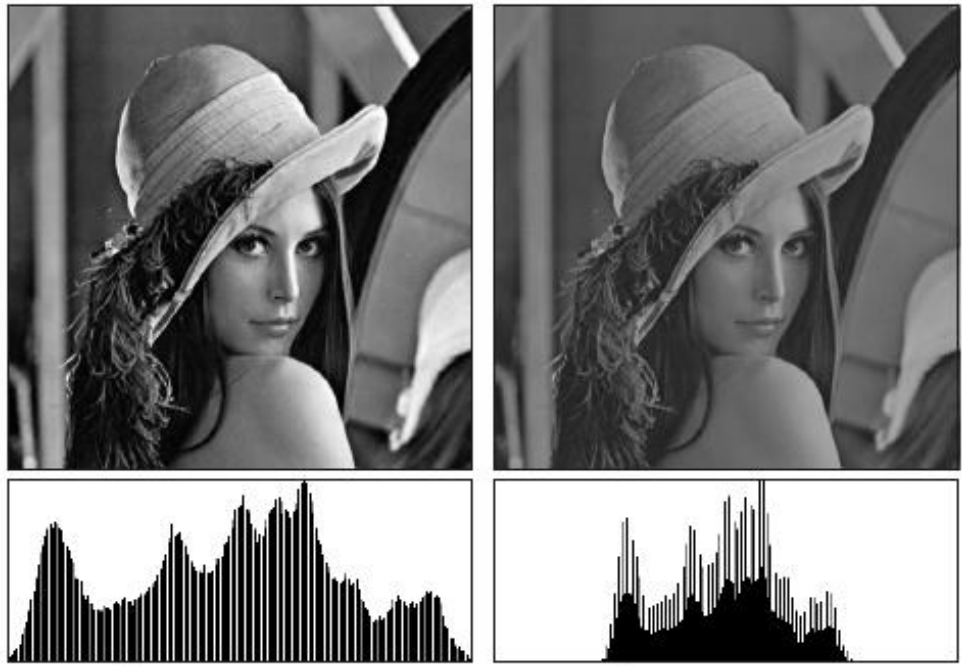
▼ 그림 5-15 영상 특성에 따른 히스토그램 분석



(a) 입력 영상 (b) 밝은 영상 (c) 어두운 영상

❖ 히스토그램 구하기 (19/19)

▼ 그림 5-15 영상 특성에 따른 히스토그램 분석



(d) 명암비가 높은 영상　　(e) 명암비가 낮은 영상

❖ 히스토그램 스트레칭 (1/8)

- **히스토그램 스트레칭**(**histogram stretching**)은 영상의 히스토그램이 그레이스케일 전 구간에 걸쳐서 나타나도록 변경하는 선형 변환 기법임
- 보통 명암비가 낮은 영상은 히스토그램이 특정 구간에 집중되어 나타나게 됨
- 히스토그램을 마치 고무줄을 잡아 늘이듯이 펼쳐서 히스토그램 그래프가 그레이스케일 전 구간에서 나타나도록 변환하는 기법임
- **히스토그램 스트레칭을 수행한 영상은 명암비가 높아지기 때문에 대체로 보기 좋은 사진으로 바뀌게 됨**

❖ 히스토그램 스트레칭 (2/8)

- 히스토그램 스트레칭을 수식으로 표현하면 다음과 같음

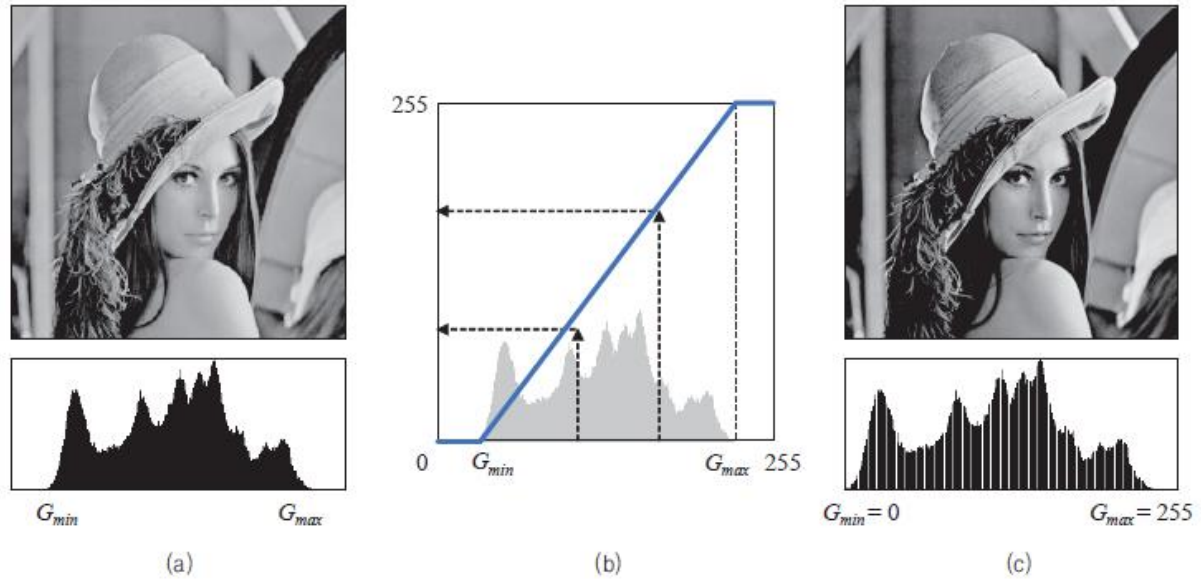
$$\mathrm{dst}(x,y) = \frac{\mathrm{src}(x,y) - G_{\min}}{G_{\max} - G_{\min}} \times 255$$

- 이 식에서 src와 dst는 각각 입력 영상과 출력 영상을 나타내고, $G_{\min}$과 $G_{\max}$는 입력 영상의 픽셀 값 중에서 가장 큰 grayscale 값과 가장 작은 grayscale 값을 나타냄

❖ 히스토그램 스트레칭 (3/8)

- 그림 5-16(a)는 레나 영상과 레나 영상의 히스토그램임
- $G_{\min}$과 $G_{\max}$는 레나 영상에서의 최소 픽셀 값과 최대 픽셀 값임
- 레나 영상의 히스토그램 분포는 $G_{\min}$과 $G_{\max}$ 사이에서만 나타남
- 히스토그램 그래프를 grayscale 양방향으로 늘려서 $G_{\min}$은 0이 되게 함
- $G_{\max}$는 255가 되도록 변환하면 히스토그램이 grayscale 전체 구간에 대해 나타나게 됨

▼ 그림 5-16 히스토그램 스트레칭과 변환 함수 그래프



❖ 히스토그램 스트레칭 (4/8)

- 그림 5-16(a) 히스토그램을 그림 5-16(c) 히스토그램처럼 만들려면 어떻게 해야 할까요?
- 그 해답이 바로 그림 5-16(b)에 나타난 변환 함수 그래프임
- 그림 5-16(b)와 같이 ($G_{\min}$, 0)과 ($G_{\max}$, 255)를 지나가는 직선의 방정식을 구해서 이를 변환 함수로 사용하면 히스토그램 스트레칭이 수행됨
- 이 직선의 방정식을 구하려면 직선의 기울기와 y 절편을 구하면 됨
- 직선의 기울기는 255 / ($G_{\max} - G_{\min}$)이고, y 절편은 비례식을 이용하여 구하면 $-255 \cdot G_{\min}$ / ($G_{\max} - G_{\min}$)이 됨
- 직선의 방정식은 다음과 같이 결정됨

$$\mathrm{dst}(x,y) = \frac{255}{G_{\max} - G_{\min}} \times \mathrm{src}(x,y) - \frac{255 \cdot G_{\min}}{G_{\max} - G_{\min}}$$

$$= \frac{\mathrm{src}(x,y) - G_{\min}}{G_{\max} - G_{\min}} \times 255$$

❖ 히스토그램 스트레칭 (5/8)

- 히스토그램 스트레칭을 위한 함수는 OpenCV에서 따로 제공하지는 않음
- OpenCV가 기본적인 산술 연산에 대한 연산자 재정의를 지원하기 때문에 앞 수식을 소스 코드로 변경하는 것은 그리 어려운 일이 아님
- 앞 수식에서 $G_{\min}$과 $G_{\max}$ 값은 `np.min()` 함수와 `np.max()` 함수를 사용하면 쉽게 구할 수 있음

❖ 히스토그램 스트레칭 (6/8)

코드 5-9 히스토그램 스트레칭 (histogram.py)

```
1   def histgoram_stretching():
2       src = cv2.imread('hawkes.bmp', cv2.IMREAD_GRAYSCALE)
3
4       if src is None:
5           print('Image load failed!')
6           return
7
8       gmin = float(np.min(src))
9       gmax = float(np.max(src))
10
11      dst = ((src - gmin) * 255. / (gmax - gmin)).astype(np.uint8)
12
13      cv2.imshow('src', src)
14      cv2.imshow('srcHist', getGrayHistImage(calcGrayHist(src)))
15
16      cv2.imshow('dst', dst)
17      cv2.imshow('dstHist', getGrayHistImage(calcGrayHist(dst)))
18
19      cv2.waitKey()
20      cv2.destroyAllWindows()
```

❖ 히스토그램 스트레칭 (7/8)

- histogram.py 소스 코드 설명
  - 2행 hawkes.bmp 파일을 grayscale 형식으로 불러와서 `src`에 저장합니다.
  - 8~9행 입력 영상 `src`에서 grayscale 최솟값과 최댓값을 구하여 `gmin`과 `gmax`에 저장합니다.
  - 11행 히스토그램 스트레칭 수식을 그대로 적용하여 결과 영상 `dst`를 생성합니다.
  - 13~17행 입력 영상과 히스토그램 스트레칭 결과 영상, 그리고 각각의 히스토그램을 화면에 출력합니다.

❖ 히스토그램 스트레칭 (8/8)

- 입력 영상 src가 전체적으로 뿌옇게 보이는 것과 달리, 히스토그램 스트레칭이 수행된 결과 영상 dst는 어두운 영역과 밝은 영역이 골고루 분포하는, 명암비가 높은 영상으로 바뀜
- 히스토그램 그래프를 살펴보면, dstHist 창에 나타난 결과 영상의 히스토그램은 입력 영상의 히스토그램이 양 옆으로 늘어난 듯한 형태로 변경된 것을 확인할 수 있음

▼ 그림 5-17 히스토그램 스트레칭 예제 실행 결과

❖ 히스토그램 평활화 (1/8)

- **히스토그램 평활화**(**histogram equalization**)는 히스토그램 스트레칭과 더불어 영상의 픽셀 값 분포가 grayscale 전체 영역에서 골고루 나타나도록 변경하는 알고리즘의 하나임
- 히스토그램 평활화는 히스토그램 그래프에서 특정 grayscale 값 근방에서 픽셀 분포가 너무 많이 뭉쳐 있는 경우 이를 넓게 펼쳐 주는 방식으로 픽셀 값 분포를 조절함
- 히스토그램 평활화는 **히스토그램 균등화** 또는 **히스토그램 평탄화**라는 용어로도 번역되어 사용되고 있음

❖ 히스토그램 평활화 (2/8)

- 히스토그램 평활화를 구현하기 위해서는 먼저 히스토그램을 구해야 함
- $h(\mathrm{g})$는 영상에서 grayscale 값이 g인 픽셀 개수를 나타냄
- 히스토그램 평활화를 계산하기 위해서는 $h(\mathrm{g})$로부터 히스토그램 누적 함수 $H(\mathrm{g})$를 구해야 함
- 히스토그램 누적 함수 $H(\mathrm{g})$는 다음 수식으로 정의됨

$$H(g) = \sum_{0 \le i \le g} h(i)$$

❖ 히스토그램 평활화 (3/8)

- 히스토그램 평활화는 이 히스토그램 누적 함수 $H(\mathrm{g})$를 픽셀 값 변환 함수로 사용함
- 다만 $H(\mathrm{g})$ 값의 범위가 보통 grayscale 값의 범위(0~255)보다 훨씬 크기 때문에 $H(\mathrm{g})$ 함수의 최댓값이 255가 되도록 정규화 과정을 거쳐야 함
- 만약 입력 영상의 픽셀 개수를 N이라고 표기하면 히스토그램 평활화는 다음과 같은 형태로 정의됨

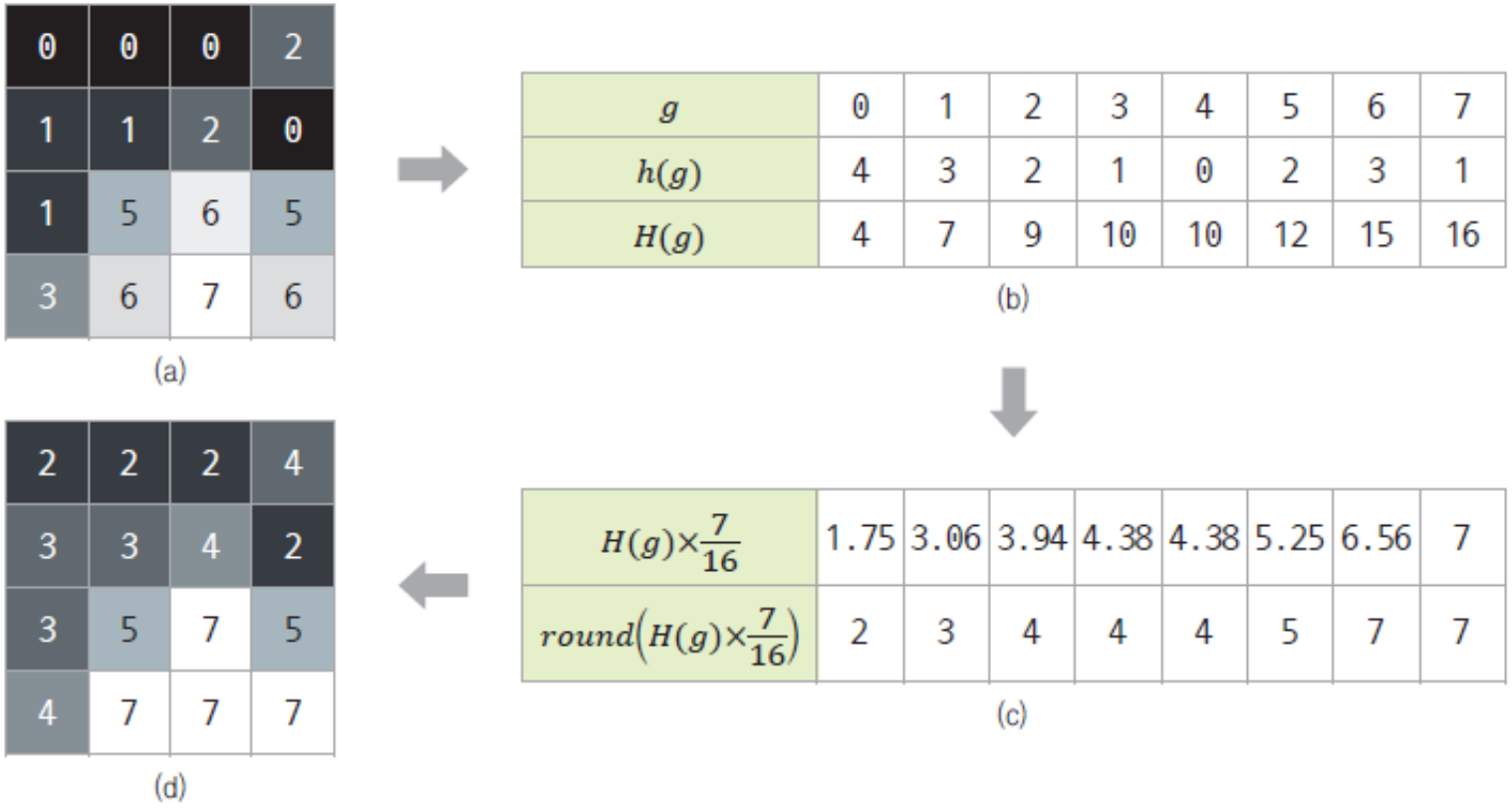
$$\mathrm{dst}(x,y) = \mathrm{round}\left(H(\mathrm{src}(x,y)) \times \frac{L_{\max}}{N} \right)$$

- 이 수식에서 $L_{\max}$는 영상이 가질 수 있는 최대 밝기 값을 의미하며 일반적인 grayscale 영상의 경우 $L_{\max} = 255$임
- round()는 반올림 함수를 나타냄

❖ 히스토그램 평활화 (4/8)

- 단순한 형태의 영상을 대상으로 실제 히스토그램 평활화가 동작하는 방식을 살펴보자

▼ 그림 5-18 단순한 영상에서 히스토그램 평활화 구현하기



❖ 히스토그램 평활화 (5/8)

- OpenCV는 grayscale 영상의 히스토그램 평활화를 수행하는 `equalizeHist()` 함수를 제공함
- `equalizeHist()` 함수 원형은 다음과 같음

`cv2.equalizeHist(src)`

`src`	입력 영상. 8비트 1채널
반환값	출력 영상. `src`와 크기와 타입이 같습니다.

- `equalizeHist()` 함수는 그레이스케일 영상만 입력으로 받음
- 3채널로 구성된 컬러 영상을 `equalizeHist()` 함수 입력으로 전달하면 에러가 발생하므로 주의해야 함

❖ 히스토그램 평활화 (6/8)

코드 5-10 히스토그램 평활화 (histogram.py)

```
1   def histogram_equalization():
2       src = cv2.imread('hawkes.bmp', cv2.IMREAD_GRAYSCALE)
3
4       if src is None:
5           print('Image load failed!')
6           return
7
8       dst = cv2.equalizeHist(src)
9
10      cv2.imshow('src', src)
11      cv2.imshow('srcHist', getGrayHistImage(calcGrayHist(src)))
12
13      cv2.imshow('dst', dst)
14      cv2.imshow('dstHist', getGrayHistImage(calcGrayHist(dst)))
15
16      cv2.waitKey()
17      cv2.destroyAllWindows()
18
19  if __name__ == '__main__':
20      histogram_equalization()
```

❖ 히스토그램 평활화 (7/8)

- histogram.py 소스 코드 설명
  - 2행 hawkes.bmp 파일을 grayscale 형식으로 불러와서 src에 저장합니다.
  - 8행 히스토그램 평활화를 수행한 결과를 dst에 저장합니다.
  - 10~14행 입력 영상과 히스토그램 평활화 결과 영상, 그리고 각각의 히스토그램을 화면에 출력합니다.

❖ 히스토그램 평활화 (8/8)

- 코드 5-10의 histgoram_equalization() 함수 실행 결과를 그림 5-19에 나타냄

▼ 그림 5-19 히스토그램 평활화 예제 실행 결과

THANK YOU!
Q & A

- Name: 권범
- Office: 동양미래대학교 2호관 704호 (02-2610-5238)
- E-mail: bkwon@dongyang.ac.kr
- Homepage: https://sites.google.com/view/beomkwon/home