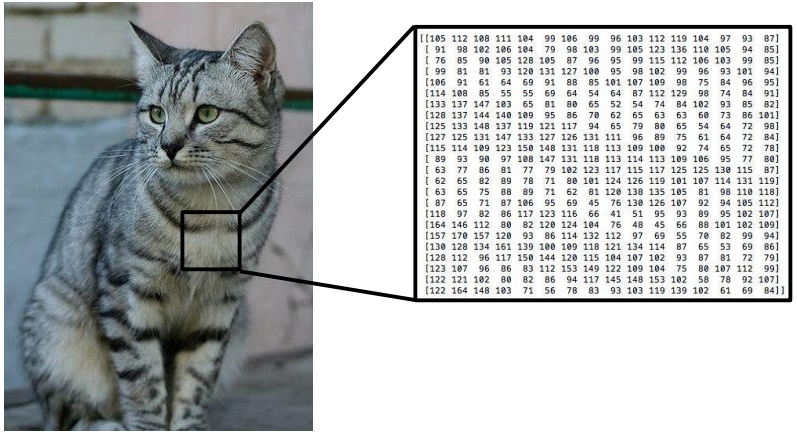
CHAPTER 06
화소처리 (1)

contents

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 영상 화소의 접근 (실습)

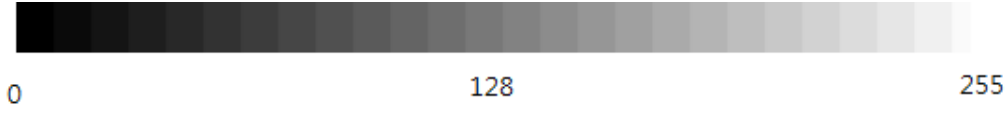
예제 6.1.1 행렬 원소 접근 방법 - 01.mat_access.py

```
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 화소 밝기 변환

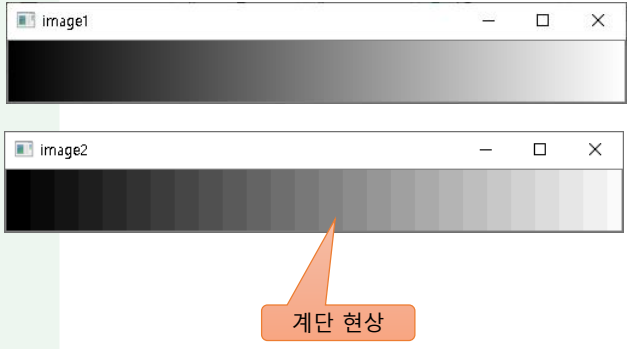
- 6.2.1 그레이 스케일(명암도) 영상
  - 흑백 영상
    - 단어 자체의 의미: 검은색과 흰색의 영상, 의미 안 맞음
  - 그레이 스케일(gray-scale) 영상 , 명암도 영상
    - 화소값은 0~255의 값을 가지는데 0은 검은색을, 255는 흰색을 의미
    - 0~255 사이의 값들은 다음과 같이 진한 회색에서 연한 회색



6.2.1 그레이 스케일(명암도) 영상 (실습)

예제 6.2.1 명암도 영상 생성 – grayscale_image.py

```
01  import numpy as np
02  import cv2
03
04  image1 = np.zeros((50, 512), np.uint8)              # 50 x 512 영상 생성
05  image2 = np.zeros((50, 512), np.uint8)
06  rows, cols = image1.shape[:2]
07
08  for i in range(rows):                               # 행렬 전체 조회
09      for j in range(cols):
10          image1.itemset((i, j), j // 2)              # 화소값 점진적 증가
11          image2.itemset((i, j), j // 20*10)          # 계단 현상 증가
12
13  cv2.imshow("image1", image1)
14  cv2.imshow("image2", image2)
15  cv2.waitKey(0)
```



// : 몫 연산자

6.2.2 영상의 화소 표현 (실습)

예제 6.2.2 영상 화소값 확인 – 03.pixel_value.py

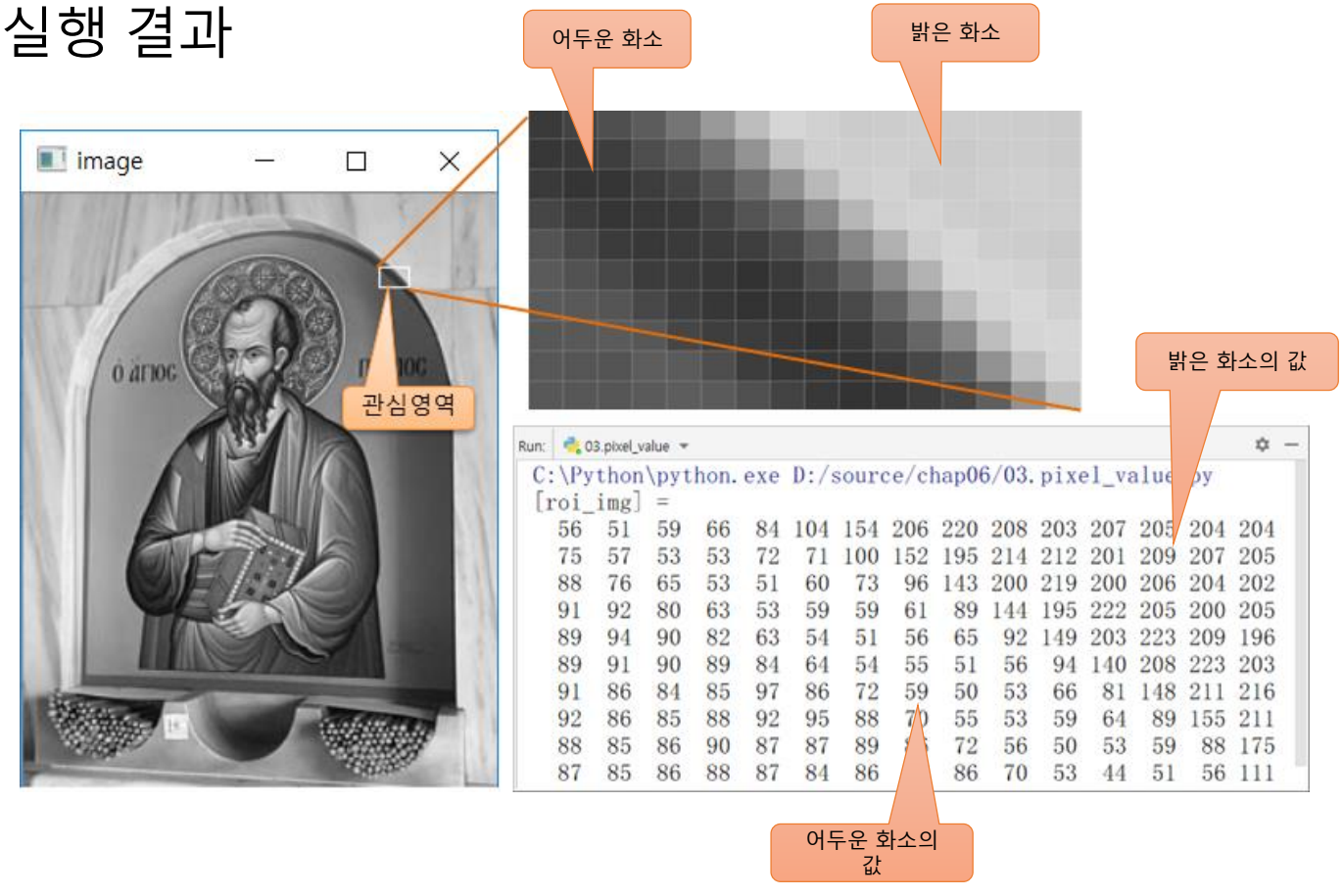
```
01  import cv2
02
03  image = cv2.imread("images/pixel.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  (x, y), (w, h) = (180, 37), (15, 10)                    # 좌표는 x, y
07  roi_img = image[y:y+h, x:x+w]                           # 행렬 접근은 y, x
08
09  #print("[roi img] =\n", roi img)                         # 행렬 원소 바로 출력 가능
10
11  print("[roi_img] =")
12  for row in roi_img:                                     # 원소 순회 방식 출력
13      for p in row:
14          print("%4d" % p, end=" ")                       # 순회 원소 하나씩 출력
15  print()
16
17  cv2.rectangle(image, (x, y, w, h), 255, 1)              # 관심 영역에 사각형 표시
18  cv2.imshow("image", image)
19  cv2.waitKey(0)
```

슬라이스 연산자 통한 관심영역 지정

사각형 튜플

6.2.2 영상의 화소 표현

- 실행 결과



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)

6.2.2 영상의 화소 표현

- 실행 결과

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)

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)
```

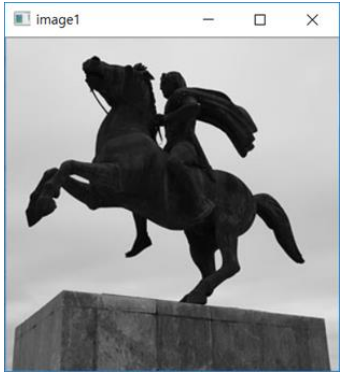
1) dst(y,x) = image1(y,x) * 0.5 + image2(y,x) * 0.5 ;

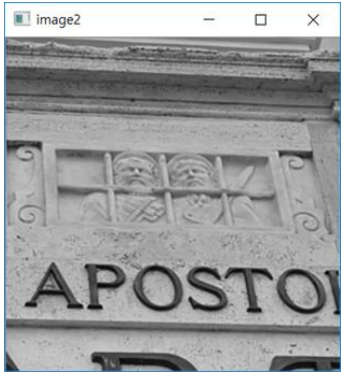
2) dst(y,x) = image1(y,x) * alpha + image2(y,x) * (1-alpha)

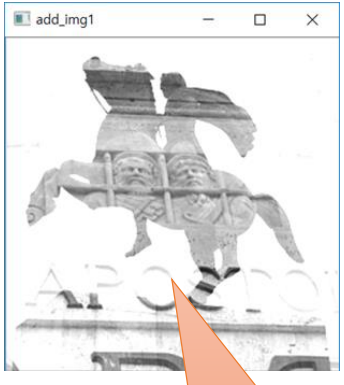
3) dst(y,x) = image1(y,x) * alpha + image2(y,x) * beta

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

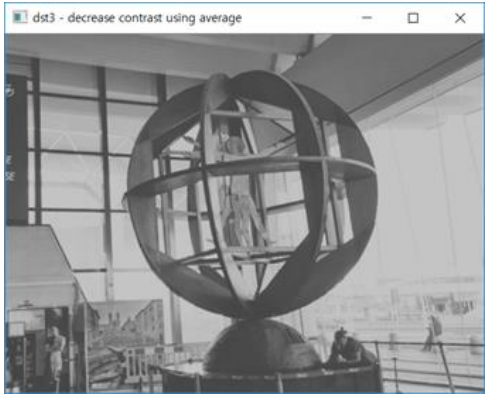
예제 6.2.5 영상 대비 변경 – 06.contrast.py

```
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

6.3 히스토그램

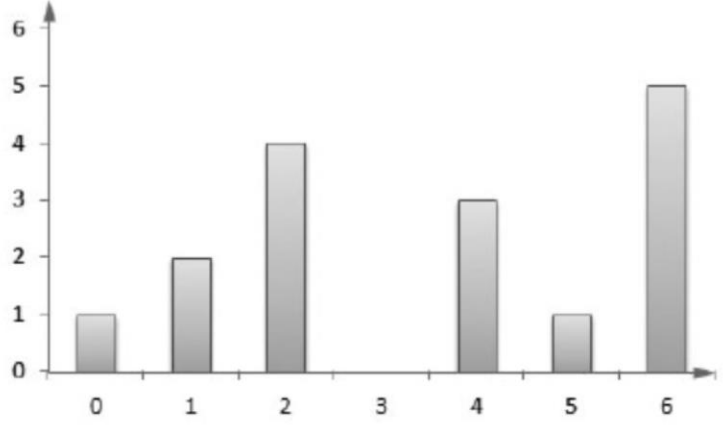
- 6.3.1 히스토그램 개념
- 6.3.2 히스토그램 계산
- 6.3.3 OpenCV 함수 활용
- 6.3.4 히스토그램 스트레칭
- 6.3.5 히스토그램 평활화

6.3.1 히스토그램 개념

- 히스토그램
  - “관측값의 개수를 겹치지 않는 다양한 계급으로 표시하는 것”
  - 어떤 데이터가 얼마나 많은지를 나타내는 도수 분포표를 그래프로 나타낸 것

5	4	6	6
2	1	6	4
2	2	4	6
1	6	0	2

(a) 입력 영상



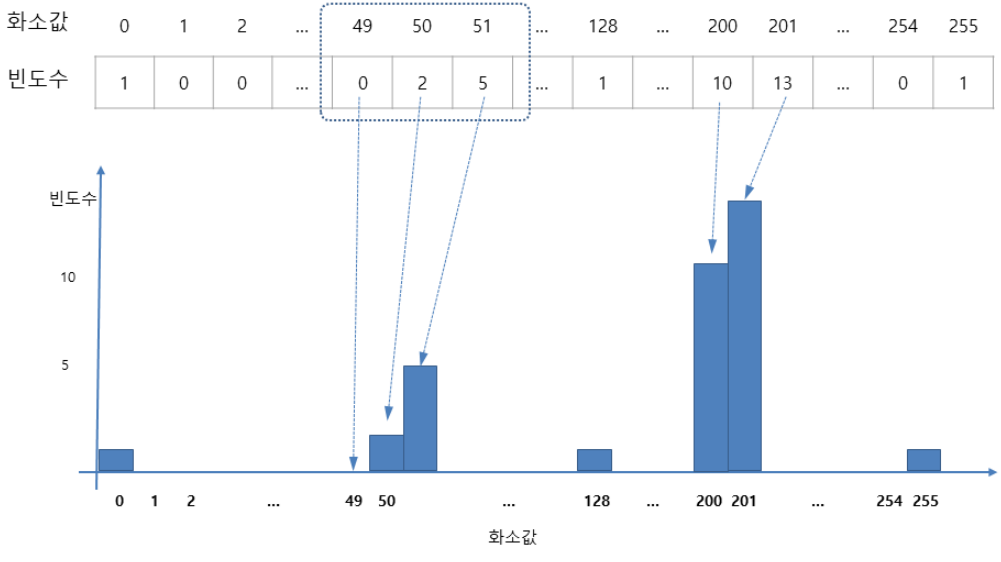
(b) 히스토그램

〈그림 6.3.1〉 히스토그램의 개념

6.3.2 히스토그램 계산

- 히스토그램 계산 및 그래프 그리기 예시

```
Run:  03.pixel_value
C:\Python\python.exe D:/source/chap06/03.pixel_value.py
[roi_img] =
  56  51  59  66  84 104 154 206 220 208 203 207 205 204 204
  75  57  53  53  72  71 100 152 195 214 212 201 209 207 205
  88  76  65  53  51  60  73  96 143 200 219 200 206 204 202
  91  92  80  63  53  59  59  61  89 144 195 222 205 200 205
  89  94  90  82  63  54  51  56  65  92 149 203 223 209 196
  89  91  90  89  84  64  54  55  51  56  94 140 208 223 203
  91  86  84  85  97  86  72  59  50  53  66  81 148 211 216
  92  86  85  88  92  95  88  70  55  53  59  64  89 155 211
  88  85  86  90  87  87  89  86  72  56  50  53  59  88 175
  87  85  86  88  87  84  86  90  86  70  53  44  51  56 111
```



6.3.2 히스토그램 계산 (실습)

- 단일 채널 히스토그램 구현 함수

예제 6.3.1 영상 히스토그램 계산 – 08.calc_histogrm.opencv.py(일부)

```
01  import numpy as np, cv2
02
03  def calc_histo(image, histSize, ranges=[0, 256] ):      # 행렬 원소의 1차원 히스토그램 계산
04      hist = np.zeros((histSize, 1), np.float32)          # 히스토그램 누적 행렬
05      gap = ranges[1] / histSize                          # 계급 간격
06
07      for row in image:                                   # 2차원 행렬 순회 방식
08          for pix in row:
09              idx = int(pix/gap)
10              hist[idx] += 1
11      return hist
```

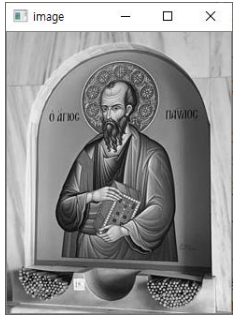
6.3.3 OpenCV 함수 활용

함수 및 인수 구조		
cv2.calcHist (images , channels , mask , histSize , ranges [, hist [, accumulate]]) → ret ■설명: 행렬의 원속값의 빈도를 계산한다.		
인수 설명	■images	원본 배열들 – CV_8U 혹은 CV_32F 형으로 크기가 같아야 함
	■channels	히스토그램 계산에 사용되는 차원 목록
	■mask	특정 영역만 계산하기 위한 마스크 행렬 – 입력 영상과 같은 크기의 8비트 배열
	■histSize	각 차원의 히스토그램 배열 크기 – 계급(bin)의 개수
	■ranges	각 차원의 히스토그램의 범위
	■accumulate	누적 플래그– 여러 배열에서 단일 히스토그램을 구할 때 사용

6.3.3 OpenCV 함수 활용

예제 6.3.1 영상 히스토그램 계산 – 08.calc_histogram_opencv.py

```
01  import numpy as np, import cv2
02
03  def calc_histo(image, histSize, ranges=[0, 256]): …              # 소스 내용 생략
12
13  image = cv2.imread("images/pixel_test.jpg", cv2.IMREAD_GRAYSCALE)          # 영상 읽기
14  if image is None: raise Exception("영상파일 읽기 오류")
15
16  histSize, ranges = [32], [0, 256]                                # 히스토그램 간격수, 값 범위
17  gap = ranges[1]/histSize[0]                                      # 계급 간격
18  ranges_gap = np.arange(0, ranges[1]+1, gap)                      # 넘파이 계급범위&간격
19  hist1 = calc_histo(image, histSize, ranges)                      # User 함수
20  hist2 = cv2.calcHist([image], [0], None, histSize, ranges)       # OpenCV 함수
21  hist3, bins = np.histogram(image, ranges_gap )                   # numpy 모듈 함수
22
23  print("User 함수: \n", hist1.flatten())                          # 1차원 행렬 1행 표시
24  print("OpenCV 함수: \n", hist2.flatten())
25  print("numpy 함수: \n", hist3)
```

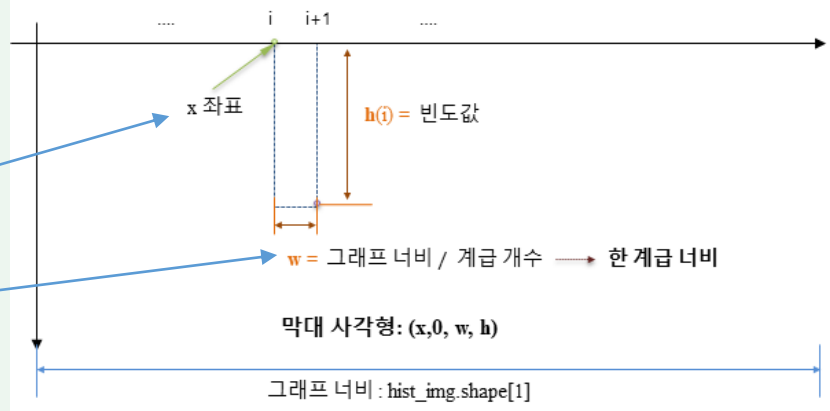


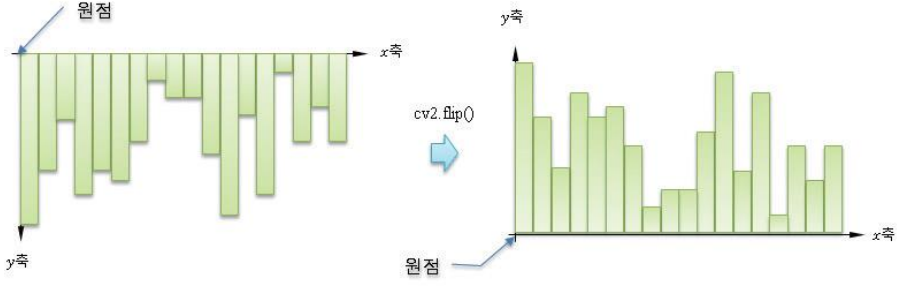

```
Run: 08.calc_histogram_opencv
C:\Python\python.exe D:/source/chap06/08.calc_histogram_opencv.py
User 함수:
 [  97.  247.  563. 1001. 1401. 1575. 1724. 1951. 2853. 3939. 3250. 2549.
 2467. 2507. 2402. 2418. 2727. 3203. 3410. 3161. 2985. 2590. 3384. 4312.
 4764. 3489. 2802. 2238. 1127.  628.  199.   37.]
OpenCV 함수:
 [  97.  247.  563. 1001. 1401. 1575. 1724. 1951. 2853. 3939. 3250. 2549.
 2467. 2507. 2402. 2418. 2727. 3203. 3410. 3161. 2985. 2590. 3384. 4312.
 4764. 3489. 2802. 2238. 1127.  628.  199.   37.]
numpy 함수:
 [  97  247  563 1001 1401 1575 1724 1951 2853 3939 3250 2549 2467 2507
 2402 2418 2727 3203 3410 3161 2985 2590 3384 4312 4764 3489 2802 2238
 1127  628  199   37]
```

6.3.3 OpenCV 함수 활용 (실습)

예제 6.3.3 히스토그램 그래프 그리기 – 09.draw_histogram.py

```
01  import numpy as np, cv2
02
03  def draw_histo(hist, shape=(200, 256)):
04      hist_img = np.full(shape, 255, np.uint8)
05      cv2.normalize(hist, hist, 0, shape[0], cv2.NORM_MINMAX)     # 정규화
06      gap = hist_img.shape[1]/hist.shape[0]                       # 한 계급 너비
07
08      for i, h in enumerate(hist):
09          x = int(round(i * gap))                                 # 막대 사각형 시작 x 좌표
10          w = int(round(gap))
11          cv2.rectangle(hist_img, (x, 0, w, int(h)), 0, cv2.FILLED)
12
13      return cv2.flip(hist_img, 0)                                # 영상 상하 뒤집기 후 반환
14
```

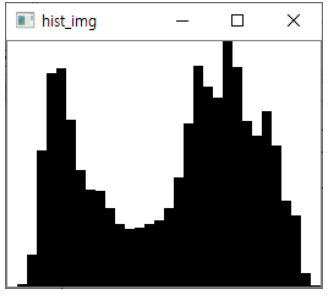




6.3.3 OpenCV 함수 활용

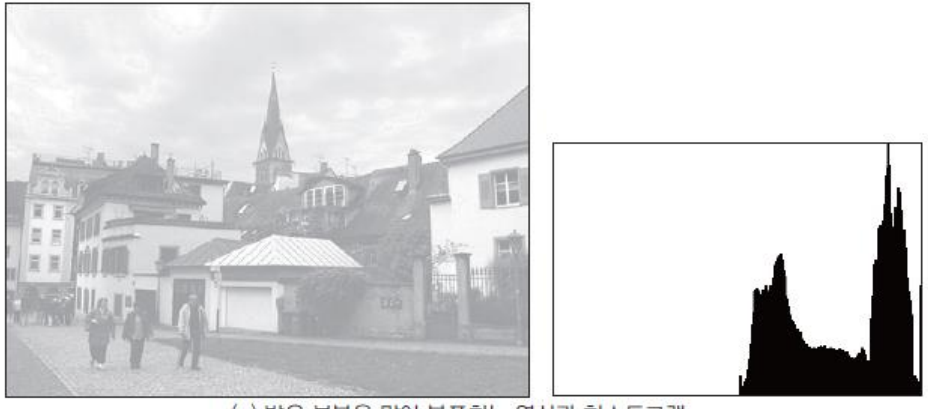
```
15  image = cv2.imread("images/pixel.jpg", cv2.IMREAD_GRAYSCALE)  # 영상 읽기
16  if image is None: raise Exception("영상파일 읽기 오류")
17
18  hist = cv2.calcHist([image], [0], None, [32], [0, 256])
19  hist_img = draw_histo(hist)
20
21  cv2.imshow("image", image)
22  cv2.imshow("hist_img", hist_img)
23  cv2.waitKey(0)
```

- 실행결과

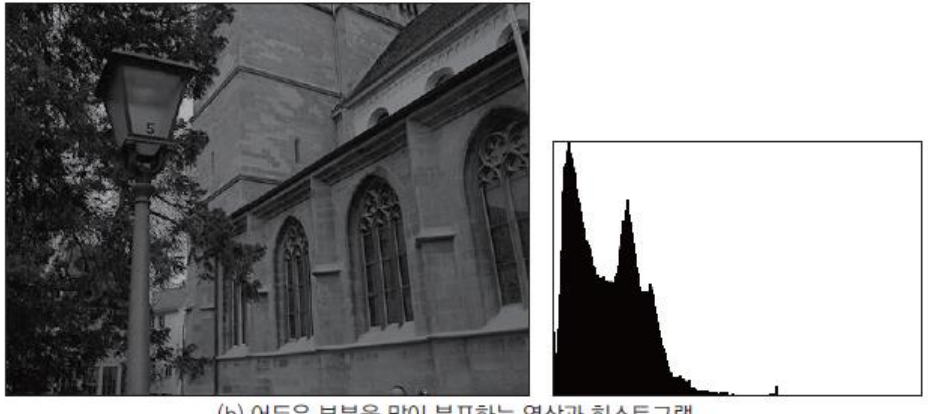


6.3.4 히스토그램 스트레칭

- 히스토그램의 분포가 좁아서 영상의 대비가 좋지 않은 영상



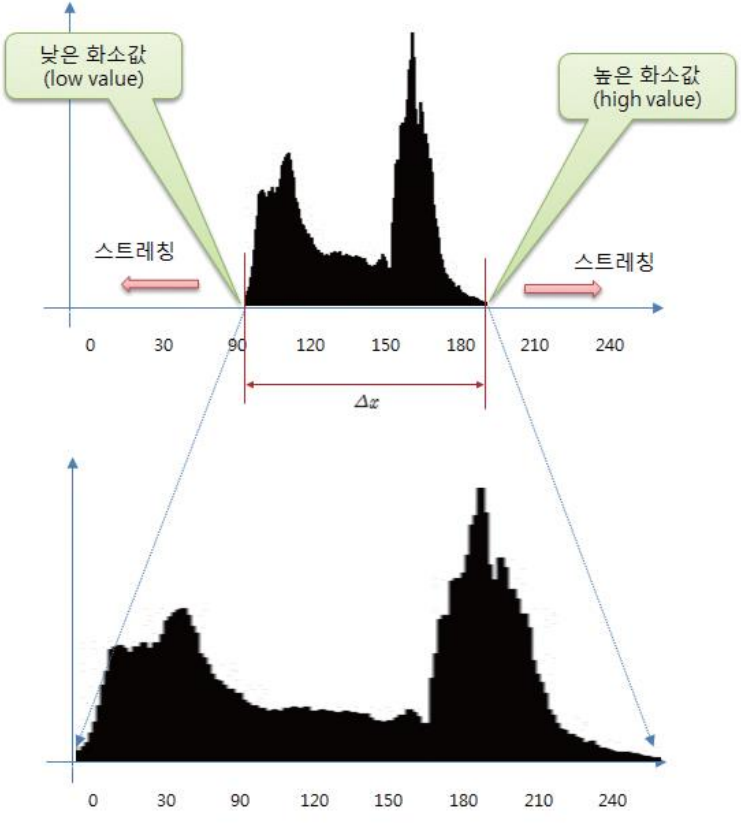
(a) 밝은 부분을 많이 분포하는 영상과 히스토그램



(b) 어두운 부분을 많이 분포하는 영상과 히스토그램

6.3.4 히스토그램 스트레칭

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

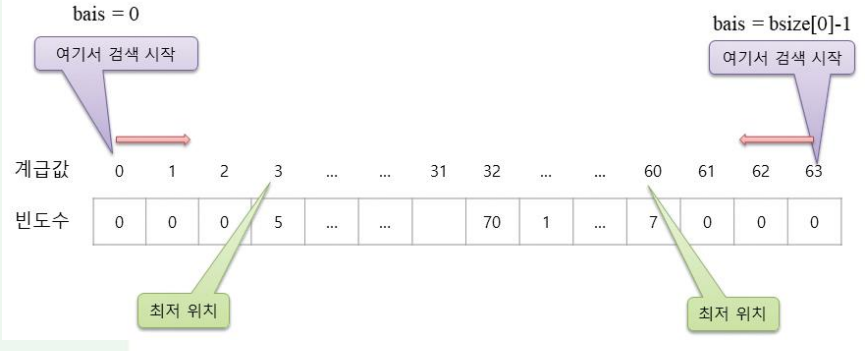


$$\text{새 화소값} = \frac{(\text{화소값} - low)}{high - low} * 255$$

6.3.4 히스토그램 스트레칭

예제 6.3.5 히스토그램 스트래칭 – histogram_stretching.py

```
01  import numpy as np, cv2
02  from Common.histogram import draw_histo              # 함수 재사용 위한 임포트
03
04  def search_value_idx(hist, bias=0):                  # 값있는 첫 계급 검색 함수
05      for i in range(hist.shape[0]):
06          idx = np.abs(bias - i)                       # 검색 위치(처음 또는 마지막)
07          if hist[idx] > 0: return idx                 # 위치 반환
08      return -1                                        # 대상 없으면 반환
09
10  image = cv2.imread("images/hist_stretch.jpg", cv2.IMREAD_GRAYSCALE)
11  if image is None: raise Exception("영상파일 읽기 오류")
12
13  bsize, ranges = [64], [0,256]                        # 계급 개수 및 화소 범위
14  hist = cv2.calcHist([image], [0], None, bsize, ranges)
15
16  bin_width = ranges[1]/bsize[0]                       # 한 계급 너비
17  low = search_value_idx(hist, 0) * bin_width          # 최저 화소값
18  high= search_value_idx(hist, bsize[0] - 1) * bin_width   # 최고 화소값
19
20  idx = np.arange(0, 256)                              # 룩업 인덱스(0~255) 생성
21  idx = (idx - low)/(high- low) * 255                  # 수식 적용하여 룩업 인덱스 완성
22  idx[0:int(low)] = 0                                  # 히스토그램 하위 부분
23  idx[int(high+1):] = 255                              # 히스토그램 상위 부분
24
```



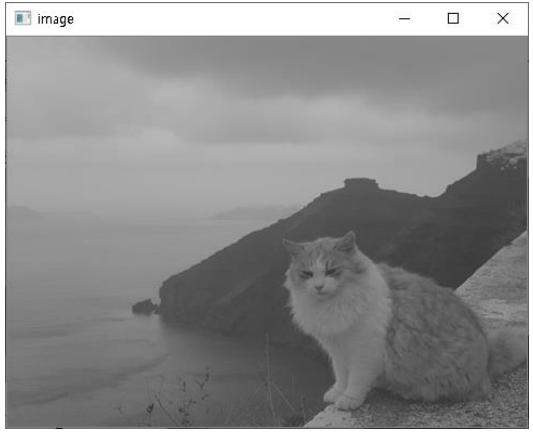
6.3.4 히스토그램 스트레칭

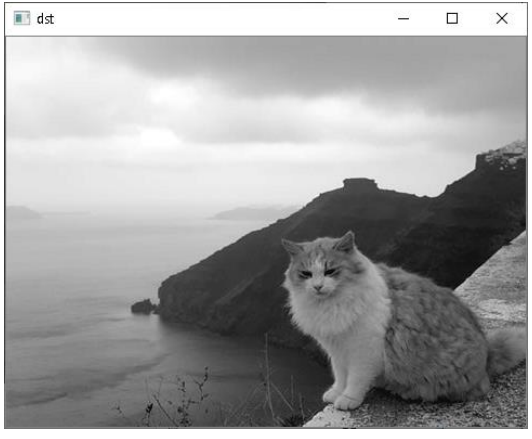
```
25  dst = cv2.LUT(image, idx.astype('uint8'))                    # 룩업 테이블 사용
26  ## 룩업 테이블 사용하지 않고 직접 구현
27  # dst = np.zeros(image.shape, dtype=image.dtype)
28  # for i in range(dst.shape[0]):
29  #         for j in range(dst.shape[1]):
30  #               dst[i,j] = idx[image[i,j]]
31
32  hist_dst = cv2.calcHist([dst], [0], None, bsize, ranges)# 결과 영상 히스토그램 재계산
33  hist_img = draw_histo(hist, (200,360))                       # 원본 영상 히스토그램 그리기
34  hist_dst_img = draw_histo(hist_dst, (200,360))               # 결과 영상 히스토그램 그리기
35
36  print("high_vlue =", high)
37  print("low_vlue =", low)
38  cv2.imshow("image", image);     cv2.imshow("hist_img", hist_img)
39  cv2.imshow("dst", dst);         cv2.imshow("hist_dst_img", hist_dst_img)
40  cv2.waitKey(0)
```

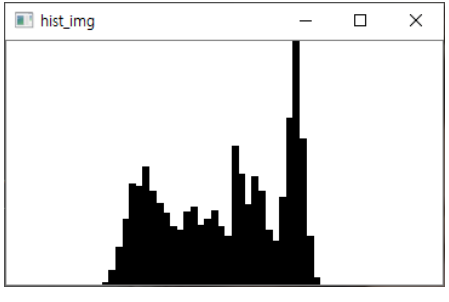
6.3.4 히스토그램 스트레칭

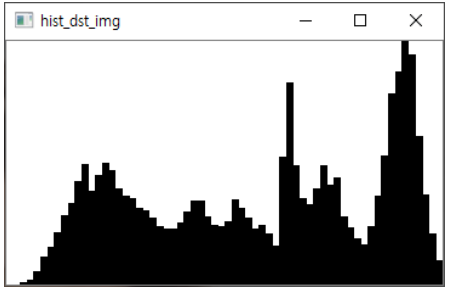
- 실행결과

```
Run: 11.histogram_stretching
C:\Python\python.exe D:/source/chap06/11.histogram_stretching.py
high_value =  180.0
low_value =  52.0
```









단원 핵심 요약

- 1. 디지털 영상은 화소들로 구성되며, 하나의 화소값은 0~255의 값을 가진다. 화소값 0은 검은색을, 255는 흰색을 의미한다.

- 2. 행렬(ndarray 객체)의 모든 원소에 스칼라 값을 더하면 영상의 밝기를 밝게 하며, 스칼라 값을 빼면 영상 밝기를 어둡게 한다.

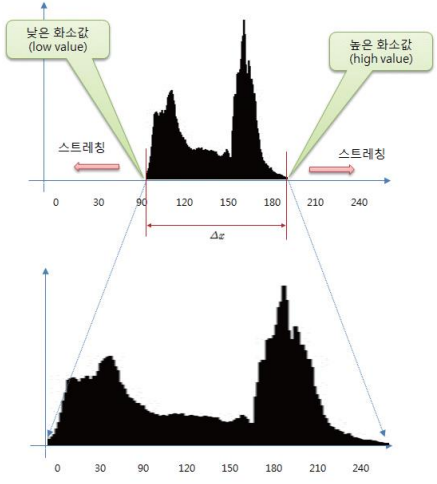
- 3. 히스토그램은 어떤 데이터가 얼마나 많은지를 나타내는 도수 분포표를 그래프로 나타낸 것이다. 가로축이 계급, 세로축이 도수(빈도수)를 뜻한다.

- 4. 히스토그램의 분포가 한쪽으로 치우쳐서 분포가 좁아서 영상의 대비가 좋지 않은 영상의 화질을 개선할 수 있는 알고리즘이 히스토그램 스트레칭(histogram stretching)이다.

6. 실습 과제

- (과제)
  - 1. 히스토그램 구현
  - 2. 히스토그램 스트레칭 구현 (코드 6.1.1 기반)
    - 피피티와 교과서에 나온 코드를 참조하지 않고 스스로 구현할 것

    - 함수 사용 범위
      - Common.histogram의 drawhist함수를 사용하여 히스토그램 출력
        - **From Common.histogram import drawhist**



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

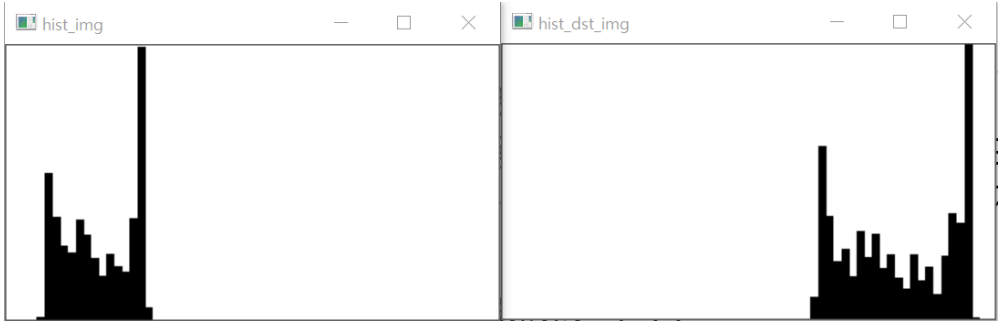
```
01  import numpy as np
02
03  def mat_access1(mat):
04      for i in range(mat.shape[0]):
05          for j in range(mat.shape[1]):
06              k = mat[i, j]
07              mat[i, j] = k * 2
08
```

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

high: 히스토그램 최대 값
low: 히스토그램 최소 값

6. 실습 과제

- (과제) 앞의 과제 코드 수정
  - 히스토그램 초기구간 [c, d]에 대해서 [a, b]로 사상(mapping) 시키는 히스토그램 스트레칭 기능을 구현하시오.
  - 예: [12, 76] -> [150,250] 으로 이동한 결과



6. 실습 규칙

- 실습 과제는 실습 시간내로 해결해야 합니다.
  - 해결 못한경우 실습 포인트를 얻지 못합니다.

- 상호간 코드 공유/보여주기 금지

- 교수/조교를 통한 질문 가능