화소 처리
Pixel-based Image Processing

- http://opencv.org

  Documentation: https://docs.opencv.org/4.7.0/

  Tutorial: https://docs.opencv.org/4.x/d7/da8/tutorial_table_of_content_imgproc.html

목차

1. 영상 화소의 접근

❖영상 처리

- 2차원 데이터(숫자)에 대한 연산

- 영상 처리는 영상을 구성하는 화소의 값을 목적에 맞게 변경하는 작업이다.
- 영상 처리를 위해서는 영상의 화소 접근과 수정이 필수

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]]
```

각 방식의 처리속도 비교

예제 6.1.2 Mat::ptr()을 통한 행렬 원소 접근 – 02.image_access.py

```
01  import numpy as np, cv2, time                  # 수행시간 계산 위해 time 모듈 임포트
02
03  def pixel_access1(image):                      # 화소 직접 접근 방법
04      image1 = np.zeros(image.shape[:2], image.dtype)
05      for i in range(image.shape[0]):
06          for j in range(image.shape[1]):
07              pixel = image[i,j]                 # 화소 접근
08              image1[i, j] = 255 - pixel         # 화소 할당
09  return image1
10
11  def pixel_access2(image):                      # item() 함수 접근 방법
12      image2 = np.zeros(image.shape[:2], image.dtype)
13      for i in range(image.shape[0]):
14          for j in range(image.shape[1]):
15              pixel = image.item(i, j)           # 화소 접근
16              image2.itemset((i, j), 255 - pixel)  # 화소 할당
17      return image2
18
19  def pixel_access3(image):                      # 룩업테이블 이용 방법
20      lut = [255 - i for i in range(256)]        # 룩업테이블 생성
21      lut = np.array(lut, np.uint8)
22      image3 = lut[image]
23      return image3
24
```

각 방식의 처리속도 비교(계속)

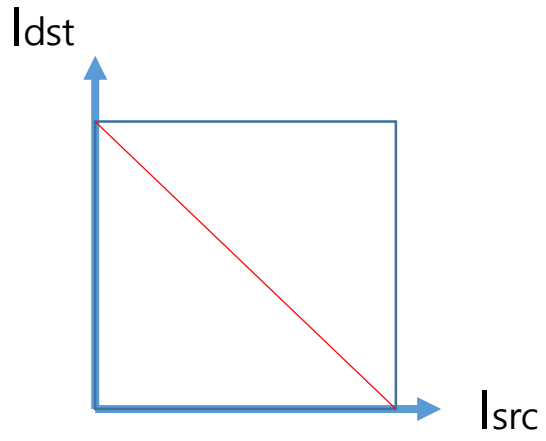
```
25  def pixel_access4(image):                          # OpenCV 함수 이용 방법
26      image4 = cv2.subtract(255, image)
27      return image4
28
29  def pixel_access5(image):                          # ndarray 산술 연산 방법
30      image5 = 255 - image
31      return image5
32
33  image = cv2.imread("images/bright.jpg", cv2.IMREAD_GRAYSCALE)
34  if image is None: raise Exception("영상파일 읽기 오류")
35
36  ## 수행시간 체크 함수
37  def time_check(func, msg):
38      start_time = time.perf_counter()
39      ret_img = func(image)
40      elapsed = (time.perf_counter() - start_time) * 1000
41      print(mag, "수행시간 : %0.2f ms" % elapsed )
42      return ret_img
43
44  image1 = time_check(pixel_access1, "[방법1] 직접 접근 방식")
45  image2 = time_check(pixel_access2, "[방법2] item() 함수 방식")
46  image3 = time_check(pixel_access3, "[방법3] 룩업테이블 방식")
47  image4 = time_check(pixel_access4, "[방법4] OpenCV 함수 방식")
48  image5 = time_check(pixel_access5, "[방법5] ndarray 연산 방식")
```

```
Run: 02.image_access
C:\Python\python.exe D:/source/chap06/02.image_access.py
[방법 1] 직접 접근 방식 수행시간 : 586.93 ms
[방법 2] item() 함수 방식 수행시간 : 65.21 ms
[방법 3] 룩업 테이블 방식 수행시간 : 0.68 ms
[방법 4] OpenCV 함수 방식 수행시간 : 0.19 ms
[방법 5] ndarray 연산 방식 수행시간 : 0.16 ms
```

반전 영상 만들기

❖Negative Image

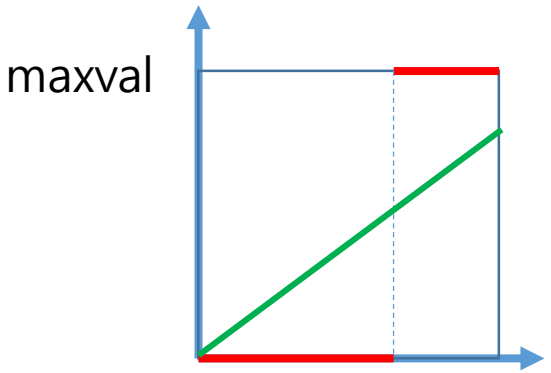
- dstImage(x, y) = 255 - srcImage(x, y)



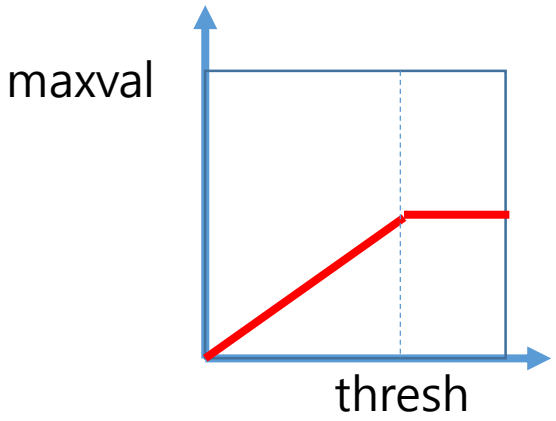
임계값 분할 영상 만들기

❖Thresholding

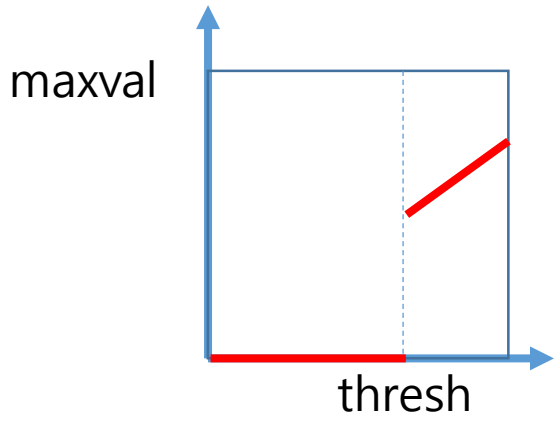
cv2.threshold (src, thresh, maxval, type);



THRESH_BINARY



THRESH_TRUNC



THRESH_TO_ZERO

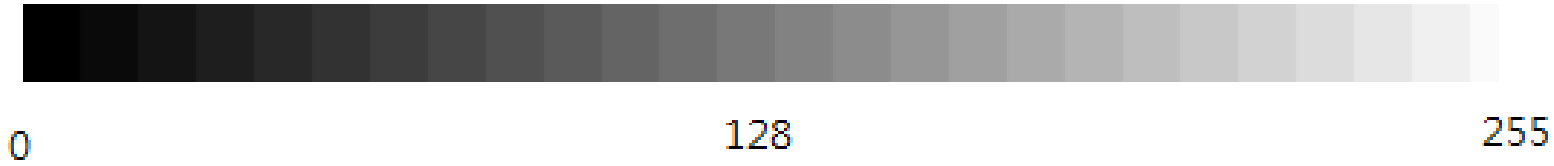
2. 화소 밝기 변환

- ❖명도 영상 (Gray-scale Image)
- ❖영상의 화소 표현
- ❖영상 밝기의 가감 연산
- ❖행렬 덧셈 및 곱셈을 이용한 영상 합성
- ❖명암 대비

명도 영상(Gray-scale Image)

단일 채널 영상을 명도(gray-scale) 영상이라 부름

- 밝기 차이만 존재 (색상 정보는 없음)
- 일반적으로 지칭하는 흑백 영상은 의미에 맞지 않음
- 그레이 스케일(gray-scale)
  - 0은 검은색을, 255는 흰색, 그 사이의 값들은 진한 회색에서 연한 회색까지 표현
  - 화소값이 밝기를 표현, 0~255의 값을 가지는 화소들로 구성



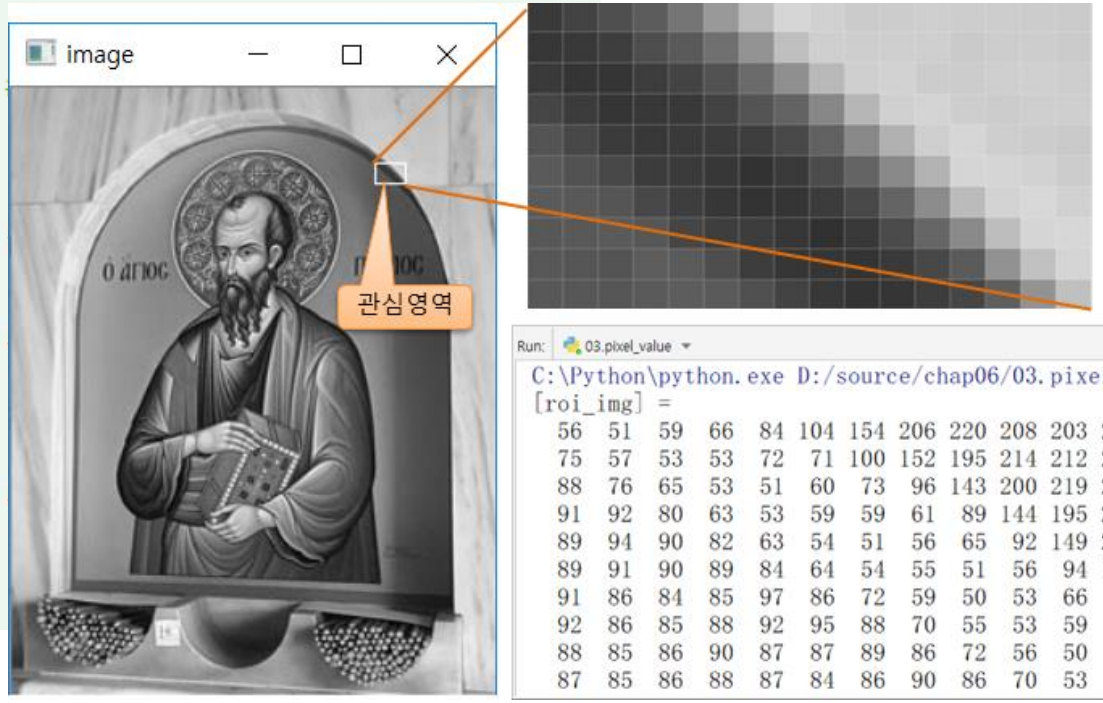
예제 6.2.1 명암도 영상 생성 – grayscale_image.py

```
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)
```

영상의 화소 표현

예제 6.2.2 영상 화소값 확인 – 03.pixel_value.py

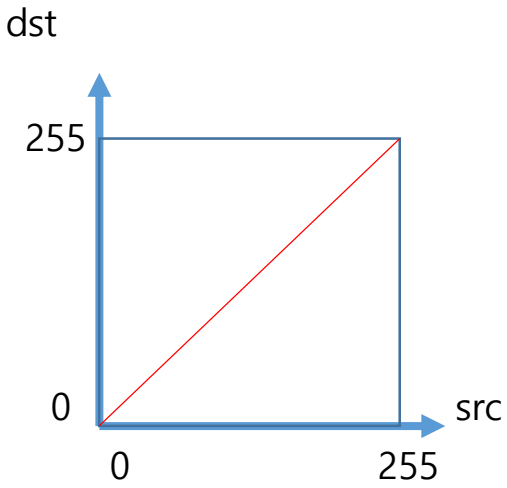
```
01  import cv2
02
03  image = cv2.imread("images/pixel.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  (x, y), (w, h) = (180, 37), (15, 10)                    # 좌표는 x, y
07  roi_img = image[y:y+h, x:x+w]                           # 행렬 접근은 y, x
08
09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



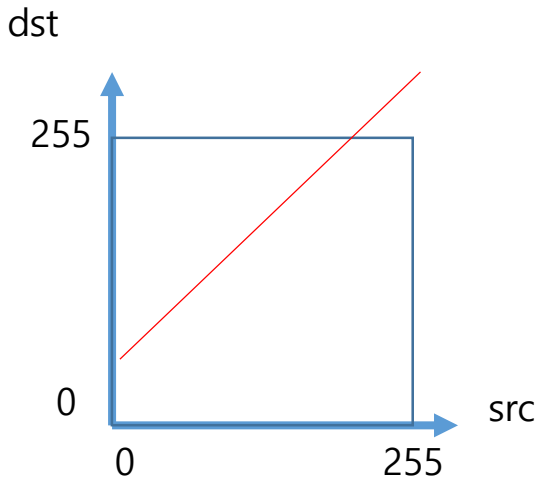

```
Run: 03.pixel_value
C:\Python\python.exe D:/source/chap06/03.pixel_value.py
[roi_img] =
  56  51  59  66  84 104 154 206 220 208 203 207 205 204 204
  75  57  53  53  72  71 100 152 195 214 212 201 209 207 205
  88  76  65  53  51  60  73  96 143 200 219 200 206 204 202
  91  92  80  63  53  59  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
```

영상의 밝기의 가감 연산

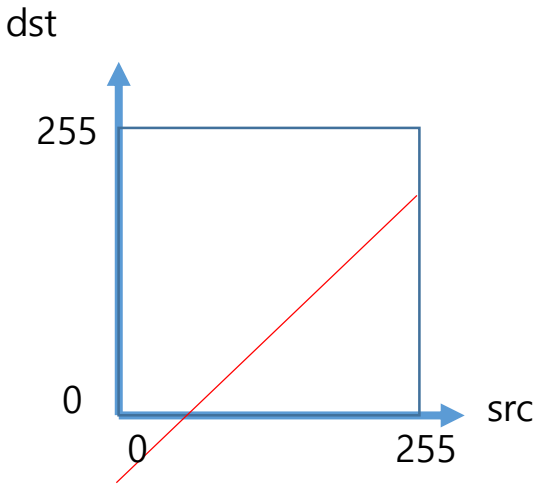
Q. 영상 내 화소 값을 증가(덧셈) or 감소(뺄셈)시키면
영상에 어떤 변화가 나타날까?



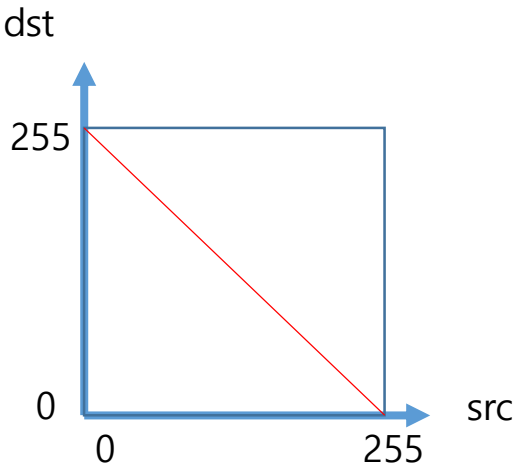
dst = src + 0



dst = src + delta



dst = src - delta



dst = 255 - src

Saturation 방식과 Modulo 방식

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)
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)

다른 부분을 찾아보자.

How can we find the difference between the two images?

행렬 덧셈 및 곱셈을 이용한 영상 합성

❖영상 합성

- 두 영상의 합 / 두 영상의 차 (차 영상)

❖영상을 더하는 다양한 방법

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

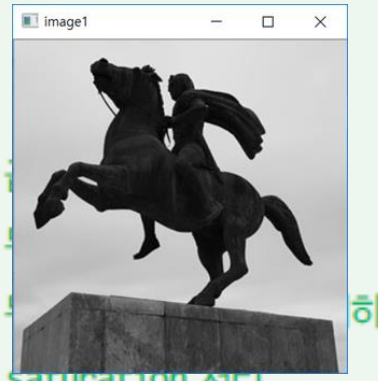
심화예제 6.2.4 행렬 합과 곱 연산을 통한 영상 합성 – 05.image_synthesis.py

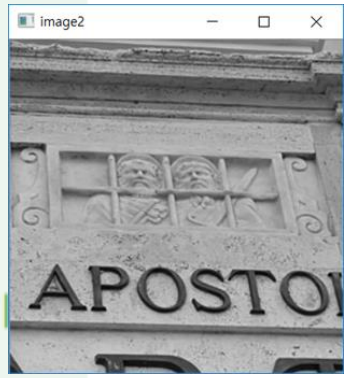
```
01  import numpy as np, cv2
02
03  image1 = cv2.imread("images/add1.jpg", cv2.IMREAD_GRAYSCALE)
04  image2 = cv2.imread("images/add2.jpg", cv2.IMREAD_GRAYSCALE)
05  if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")
06
07  ## 영상 합성 방법
08  alpha, beta = 0.6, 0.7                   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

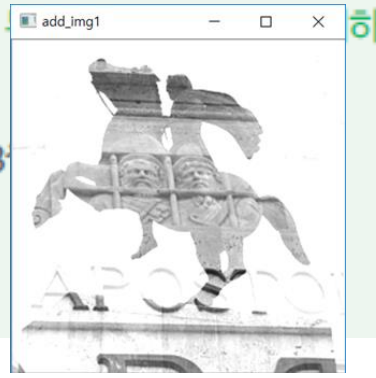
1) dst(y,x) = image1(y,x) * 0.5 + image2(y,x) * 0.5 ;

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





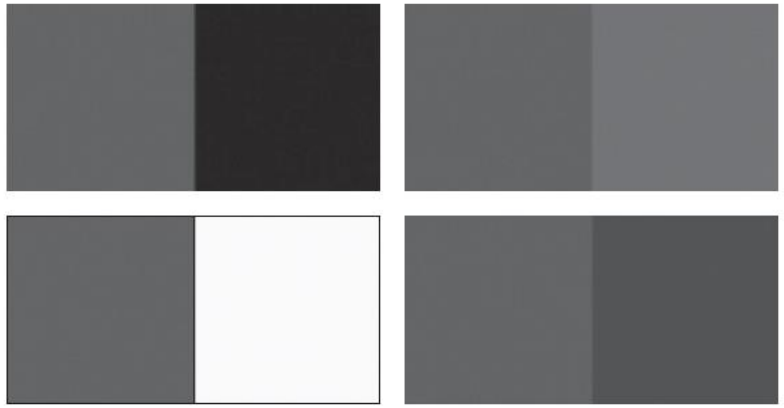


np.clip: 0~255 범위로 saturate

명암 대비

❖명암 대비(contrast)

- 상이한 두 가지 색(밝기)이 경계에서 서로 영향을 미쳐 그 차이가 강조되어 나타나는 현상



〈그림 6.2.2〉 밝기 대비 예시

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
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)
```