Chapter

6

화소처리(1)

1

화소처리와 영역처리

영상처리의 기본연산

❖ 점 연산(Point Processing)

- 오직 자신의 명암값에 따라 새로운 값을 결정

❖ 영역 연산(Region Processing)

- 이웃 화소의 명암값에 따라 새로운 값 결정

❖ 기하 연산(Geometric Processing)

- 일정한 기하 연산으로 결정된 화소의 명암값에 따라 새로운 값 결정

화소처리 vs. 영역처리

20	20	20	20	20	20	20
50	50	60	60	60	60	60
50	50	60	70	70	70	60
60	60	60	70	70	80	70
60	70	70	70	80	80	70
70	80	70	80	80	90	80
80	80	80	80	90	90	90

<Point Processing>

70을 변화시키기 위해 (4,4)의 자신만의 데이터를 사용한 경우

vs.

<Region Processing>

(4,4)위치 인근의 여러 점들을 사용하는 경우

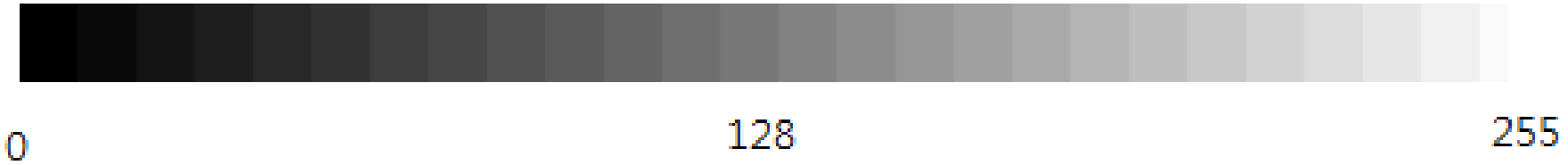
그레이(Gray-scale) 영상

❖ 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



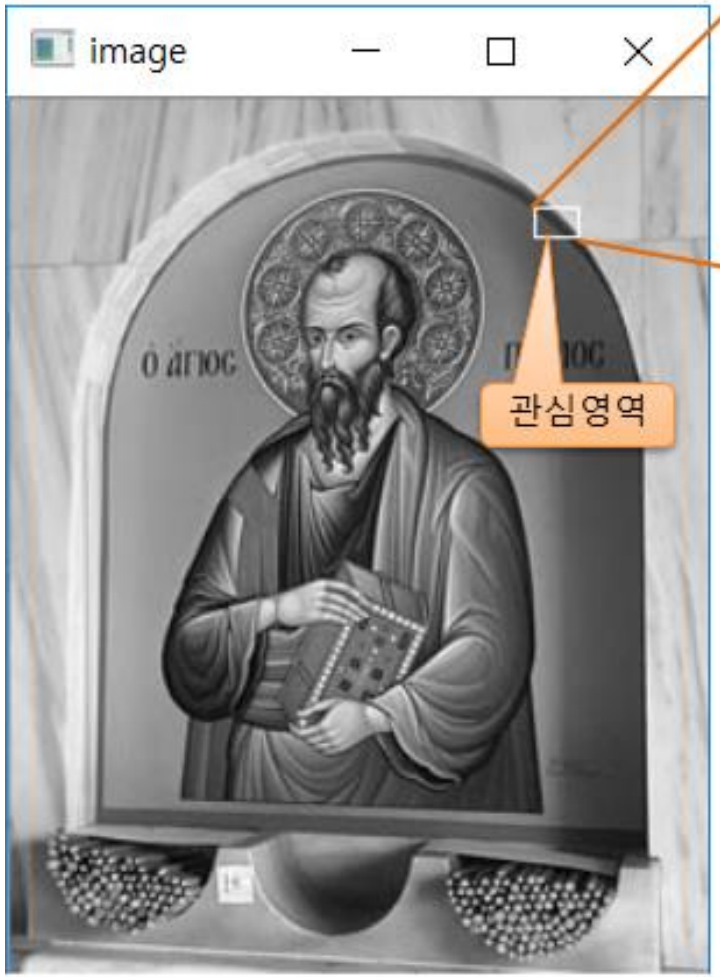
예제 6.2.2 영상 화소값 확인 – 03.pixel_value.py

```
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

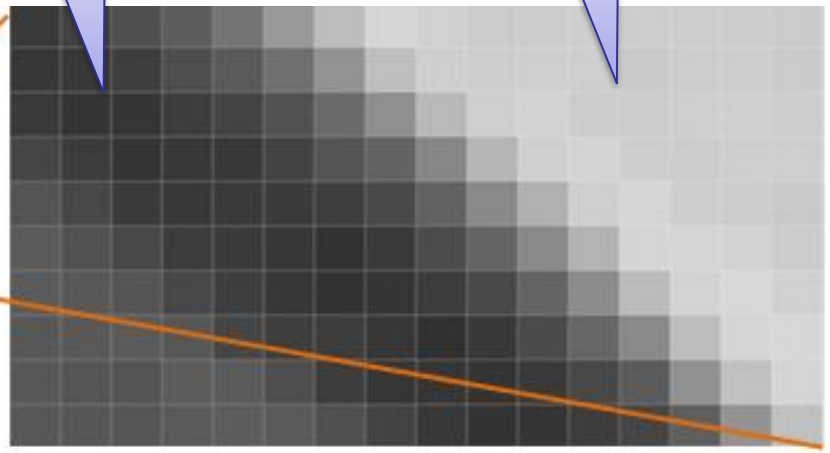
사각형 튜플

```
Run: 03.pixel_value
C:\Python\python.exe D:/source/chap06/03.pixel_value.py
Traceback (most recent call last):
  File "D:/source/chap06/03.pixel_value.py", line 4, in <module>
    if image is None: raise Exception("영상 파일 읽기 오류")
Exception: 영상 파일 읽기 오류
```



어두운 화소

밝은 화소



밝은 화소의 값

```
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  82  72  56  50  53  59  88 175
  87  85  86  88  87  84  86  86  70  53  44  51  56 111
```

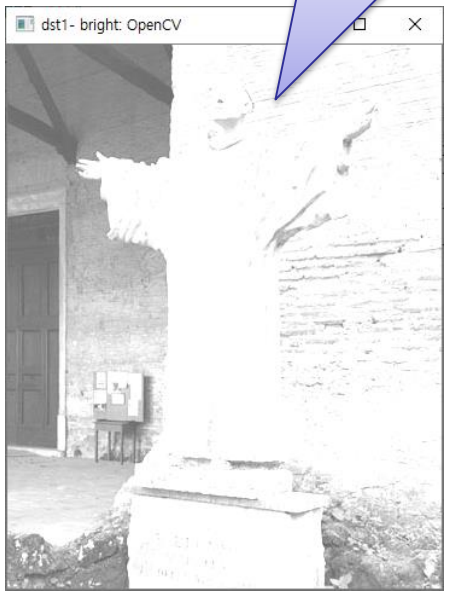
어두운 화소의 값

modulo방식과 saturation방식

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)
```

- OpenCV
  - saturation 방식
    - 250 + 100 = 360
    - 360 -> 255
- numpy
  - modulo 방식
    - 250 + 100 = 360
    - 360 % 256 -> 104



saturation 방식에 따라 255 이상값은 255로 지정

saturation 방식에 따라 0 아하 값은 0로 지정

modulo 방식에 따른 화소값 에러

modulo 방식에 따른 화소값 에러

이미지 픽셀에 접근하기

❖ 픽셀의 값에 직접접근

- img[340, 200] = [100, 150, 200]
  - 340, 200위치의 픽셀을 B=100, G=150, R=200의 값으로 변경
- 상대적으로 느림

❖ numpy의 item(), itemset()함수로 접근

- 최적화되어 있어 빠르다
- B, G, R 각각의 값에 개별적으로 접근해야 함
- (예)
  - `b = img.item(340, 200, 0)`
  - `g = img.item(340, 200, 1)`
  - `r = img.item(340, 200, 2)`

❖ itemset()함수로 픽셀값 변경하기

- B, G, R 각각의 변경해야 함
- (예)
  - `img.itemset((340, 200, 0), 100) # B = 100`
  - `img.itemset((340, 200, 1), 150) # G = 150`
  - `img.itemset((340, 200, 2), 200) # R = 200`

이미지의 속성 얻기

❖ 주요 이미지 속성

- img.shape
  - 이미지의 (높이, 너비, 채널수)
- img.size
  - 이미지의 크기(바이트)
- img.dtype
  - 이미지 픽셀의 데이터타입

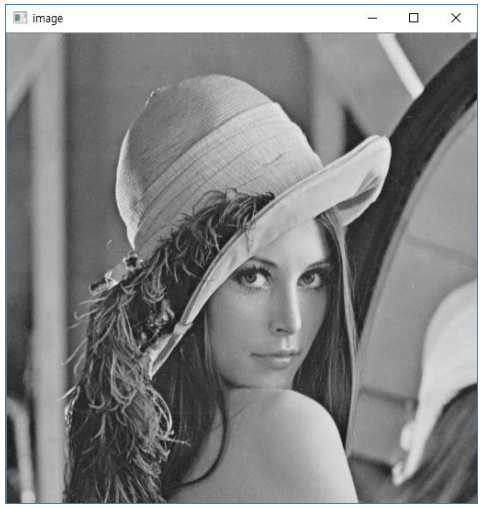
❖ (예)

```
print(img.shape)
print(img.size)
print(img.dtype)
```

```
(512, 512, 3)
786432
uint8
```

이치화(픽셀처리)

```
import cv2
import numpy as np

def showImage():
    filename = "lena.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    ysize = img.shape[0]
    xsize = img.shape[1]
    for y in range(ysize):
        for x in range(xsize):
            if img.item(y, x) < 128:
                img.itemset((y, x), 0)
            else:
                img.itemset((y, x), 255)
    cv2.imshow('bin-image', img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```



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]]
```

화소접근 방법들

❖ 개별화소에 접근하는 방법들

- 인덱싱에 의한 직접접근
- np.item()을 통해 접근
- lookup table(LUT)를 통해 접근
  - LUT를 생성한 후, LUT를 이용하여 연산
- OpenCV의 함수를 통해 접근
  - OpenCV에 구현되어 있는 함수를 사용하는 방법
- ndarray의 산술연산을 통해 접근
  - numpy의 ndarray에 대한 산술연산

❖ 결론(실행시간)

- **인덱싱 >> np.item() >> LUT >> OpenCV >> ndarray**
- **가급적 numpy 또는 OpenCV의 함수를 이용하자!!!!**

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

```python
import numpy as np, cv2, time                    # 수행시간 계산 위해 time 모듈 임
                                                 # 화소 직접 접근 방법
def pixel_access1(image):
    image1 = np.zeros(image.shape[:2], image.dtype)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            pixel = image[i,j]                   # 화소 접근
            image1[i, j] = 255 - pixel           # 화소 할당
return image1

                                                 # item() 함수 접근 방법
def pixel_access2(image):
    image2 = np.zeros(image.shape[:2], image.dtype)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            pixel = image.item(i, j)             # 화소 접근
            image2.itemset((i, j), 255 - pixel)  # 화소 할당
    return image2

                                                 # 룩업테이블 이용 방법
def pixel_access3(image):
    lut = [255 - i for i in range(256)]          # 룩업테이블 생성
    lut = np.array(lut, np.uint8)
    image3 = lut[image]
    return image3

def pixel_access4(image):                        # OpenCV 함수 이용 방법
    image4 = cv2.subtract(255, image)
    return image4

def pixel_access5(image):                        # ndarray 산술 연산 방법
    image5 = 255 - image
    return image5

image = cv2.imread("images/bright.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상파일 읽기 오류")

## 수행시간 체크 함수
def time_check(func, msg):
    start_time = time.perf_counter()
    ret_img = func(image)
    elapsed = (time.perf_counter() - start_time) * 1000
    print(mag, "수행시간 : %0.2f ms" % elapsed )
    return ret_img

image1 = time_check(pixel_access1, "[방법1] 직접 접근 방식")
image2 = time_check(pixel_access2, "[방법2] item() 함수 방식")
image3 = time_check(pixel_access3, "[방법3] 룩업테이블 방식")
image4 = time_check(pixel_access4, "[방법4] OpenCV 함수 방식")
image5 = time_check(pixel_access5, "[방법5] ndarray 연산 방식")
```

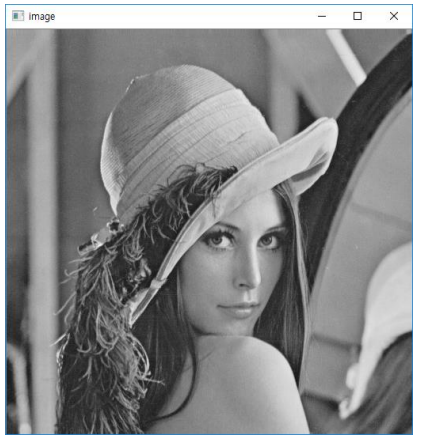
함수명

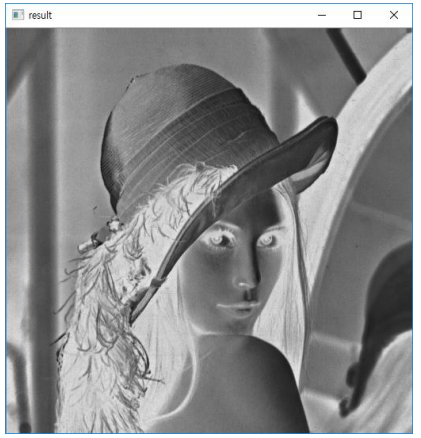
수행할 함수명

```
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
```

참고: LUT 적용의 예

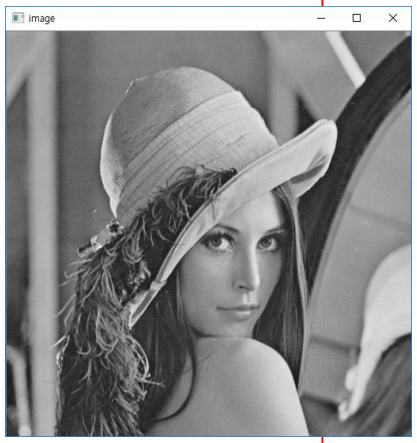
```python
import cv2
import numpy as np

def main():
    filename = "lena.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    # Create Lookup Table
    lut = np.arange(255, -1, -1, dtype="uint8")

    ysize = img.shape[0]
    xsize = img.shape[1]
    for y in range(ysize):
        for x in range(xsize):
            img.itemset((y, x), lut[img.item(y, x)])

    cv2.imshow('result', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

main()
```

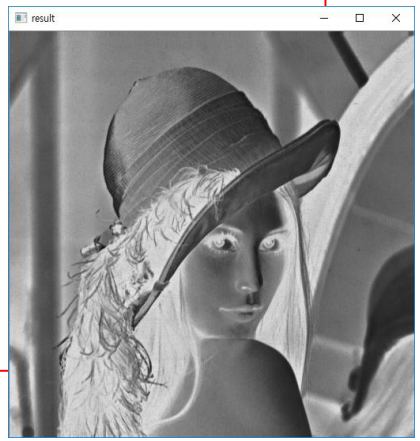




cv2.LUT()

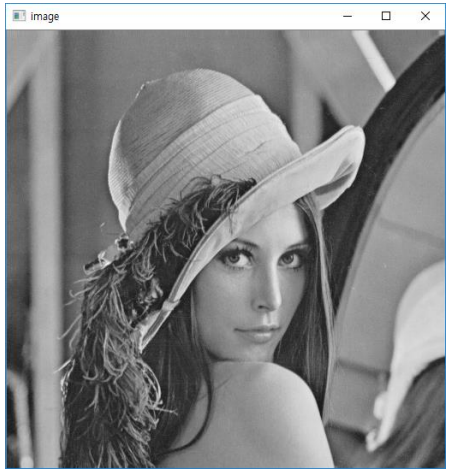
```python
import cv2
import numpy as np

def showImage():
    filename = "lena.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    # Create Lookup Table
    lut = np.arange(255, -1, -1, dtype='uint8')

    result = cv2.LUT(img, lut)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```

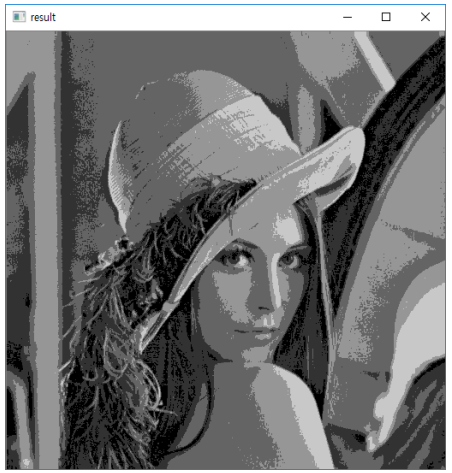




참고: 포스터라이징(Posterizing)

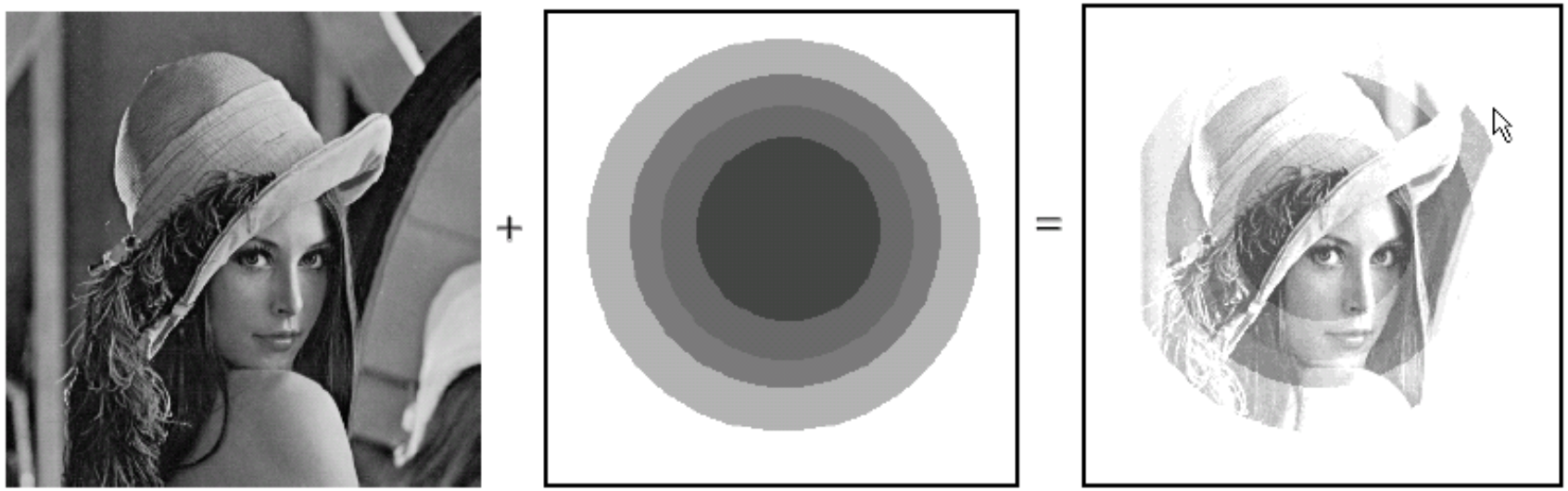
```python
import cv2
import numpy as np

def CreateLUT():
    LUT = np.arange(256)
    for i in range(256):
        LUT[i] = LUT[i] // 50 * 50
    return LUT

def main():
    filename = "lena.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    # Create Lookup Table
    lut = CreateLUT()

    ysize = img.shape[0]
    xsize = img.shape[1]
    for y in range(ysize):
        for x in range(xsize):
            img.itemset((y, x), lut[img.item(y, x)])

    cv2.imshow('result', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

main()
```

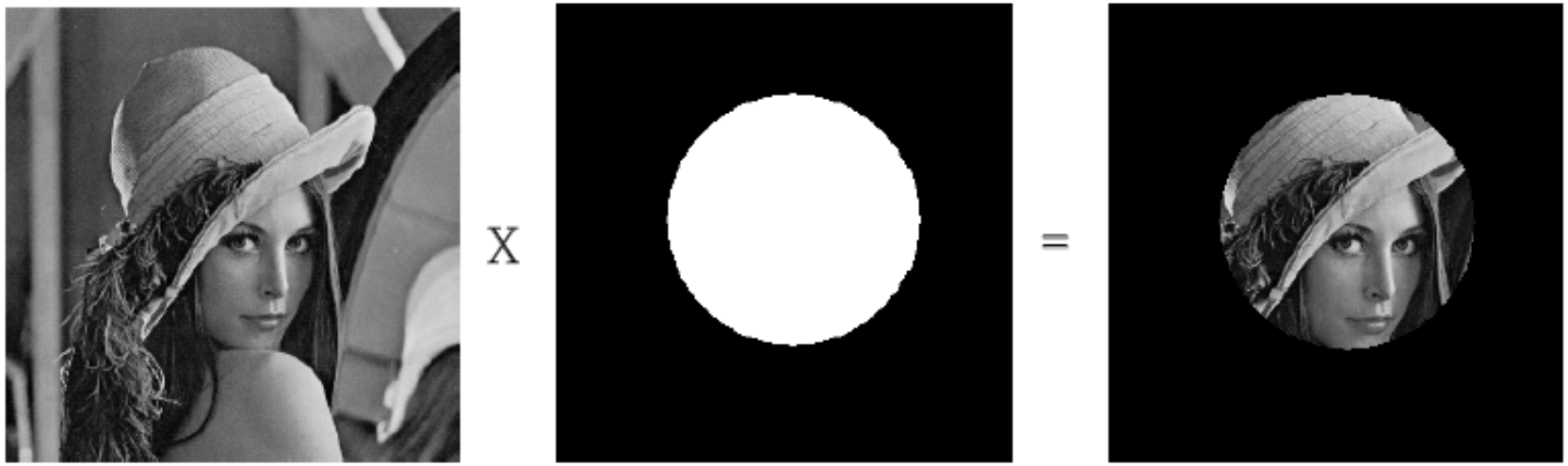




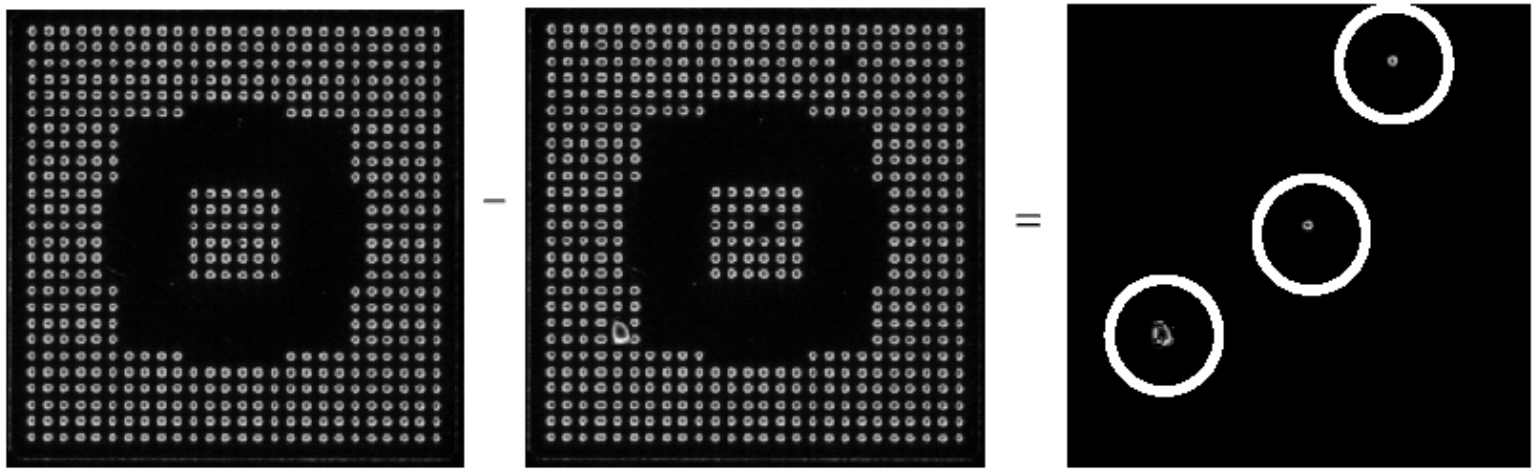
2 산술연산

두 영상간의 산술연산





두 영상간 뺄셈 연산의 예

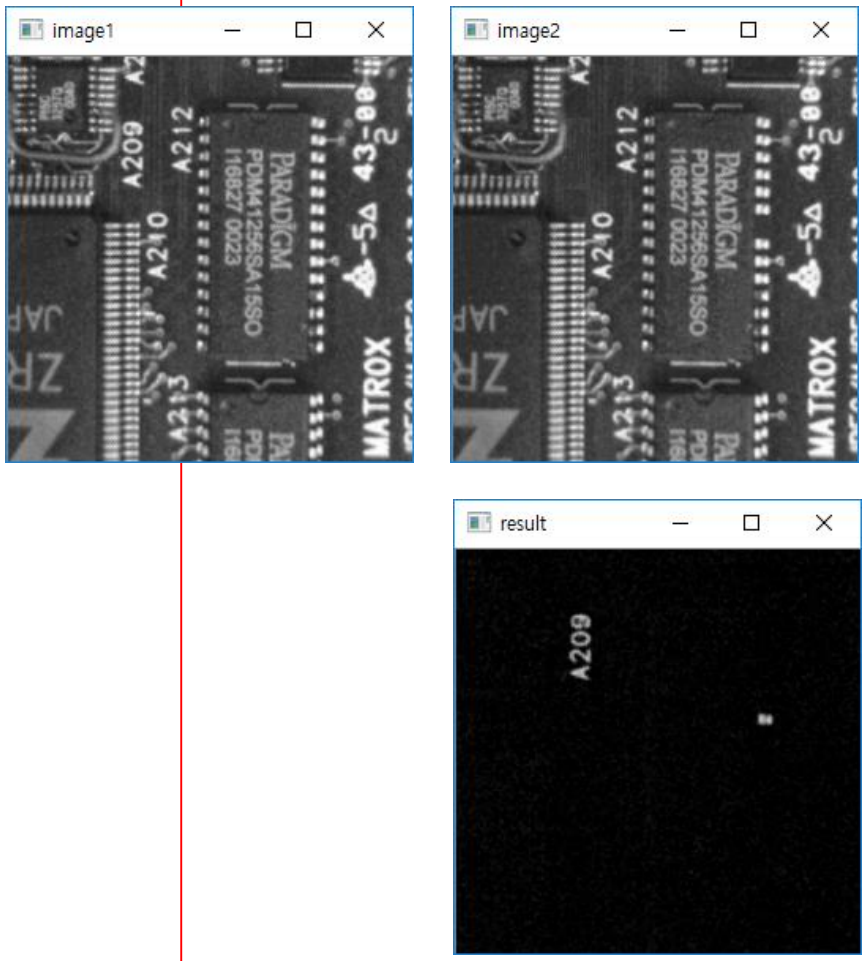


BGA모델영상 검사할 결함영상 발견된 결함(빼기영상)

배경(background)영상 입력영상 차분 영상

cv2.subtract()

```
import cv2
import numpy as np

def showImage():
    file1 = "ic_ref.jpg"
    file2 = "ic_test.jpg"

    img1 = cv2.imread(file1, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image1', img1)
    img2 = cv2.imread(file2, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image2', img2)

    result = cv2.subtract(img1, img2)
    cv2.imshow('result', result)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```

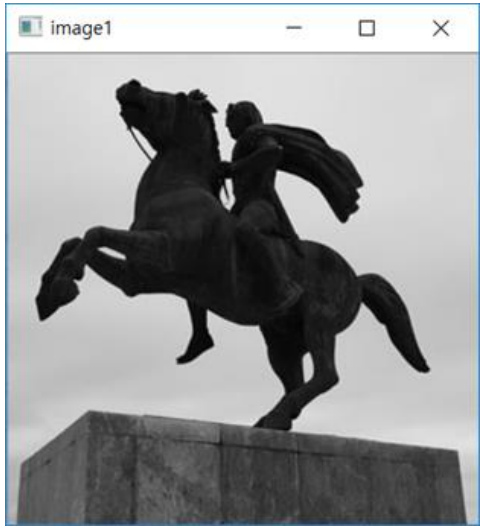


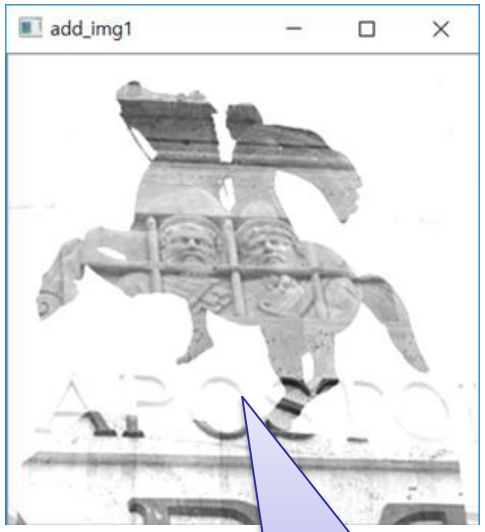
영상합성

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





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

비율 조정하여 합성

명암 대비

❖ 대비(contrast)

- 같은 색도 인접한 색의 밝기에 따라서 다르게 보임

〈그림 6.2.2〉 밝기 대비 예시

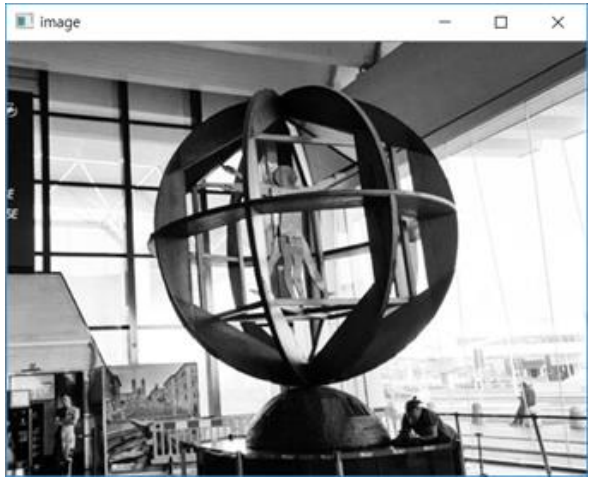
- (예) 낮은 대비영상과 높은 대비의 영상

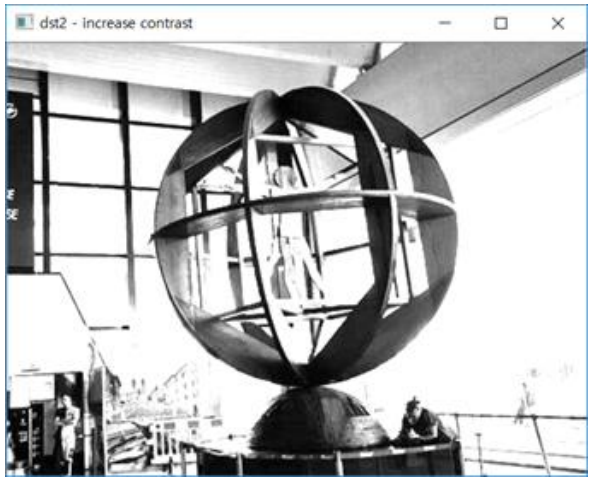
예제 6.2.5 영상 대비 변경 – 06.contrast.py

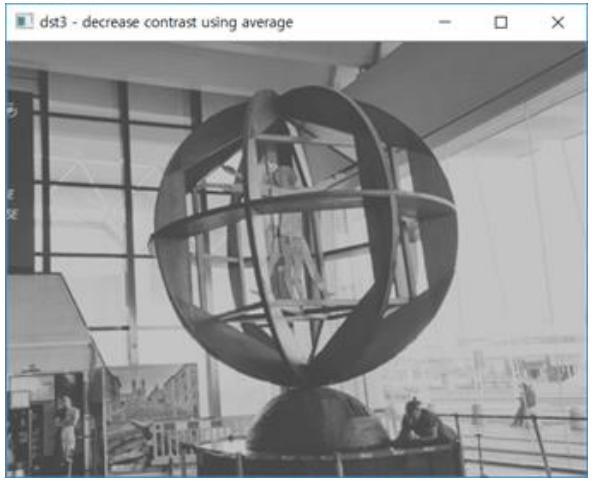
```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/contrast.jpg", cv2.IMREAD_GRAYSCALE) # 영상 읽기
04  if image is None: raise Exception("영상파일 읽기 오류")
05
06  noimage = np.zeros(image.shape[:2], image.dtype)                # 더미 영상
07  avg = cv2.mean(image)[0]/2.0                                    # 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)
```

❖ 실행결과

곱셈으로 영상 대비 변경(감소 및 증가)







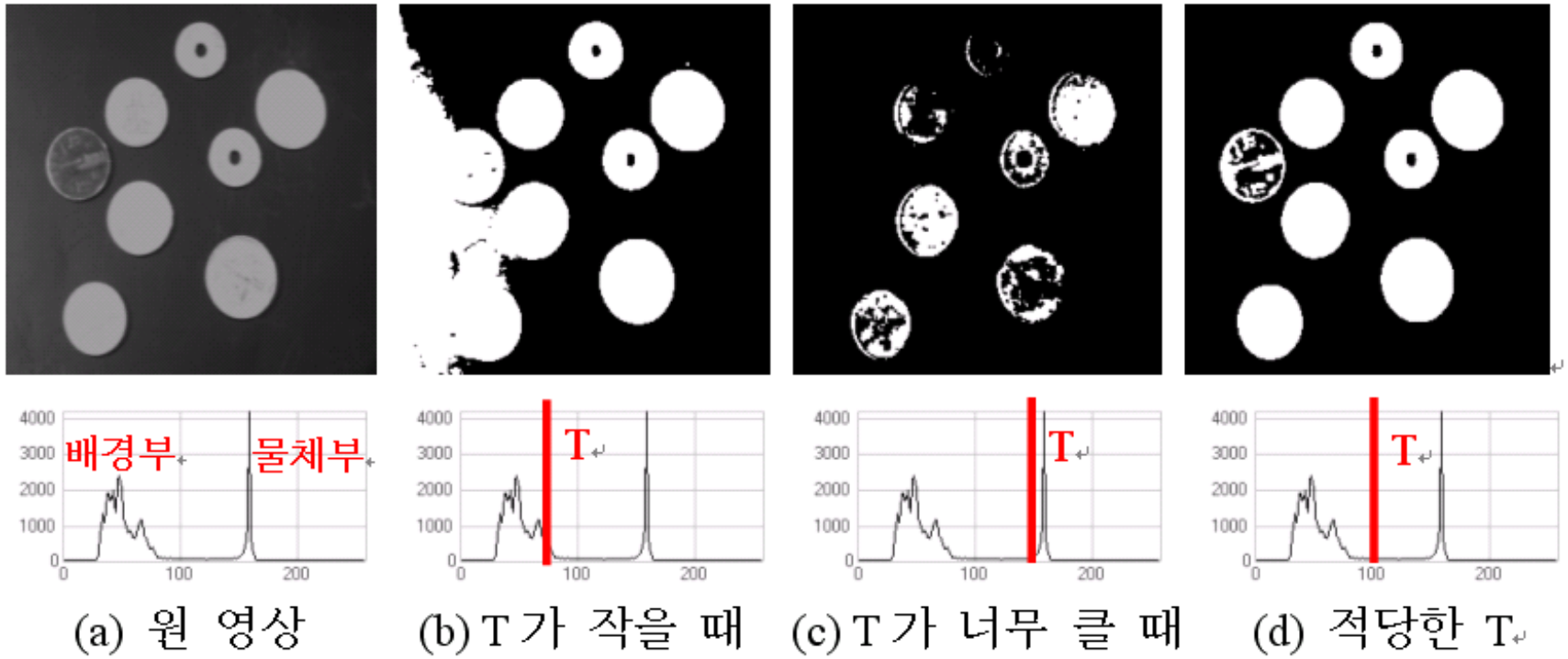
영상 평균값을 활용하여
대배 변경시 화질 개선

3

이치화

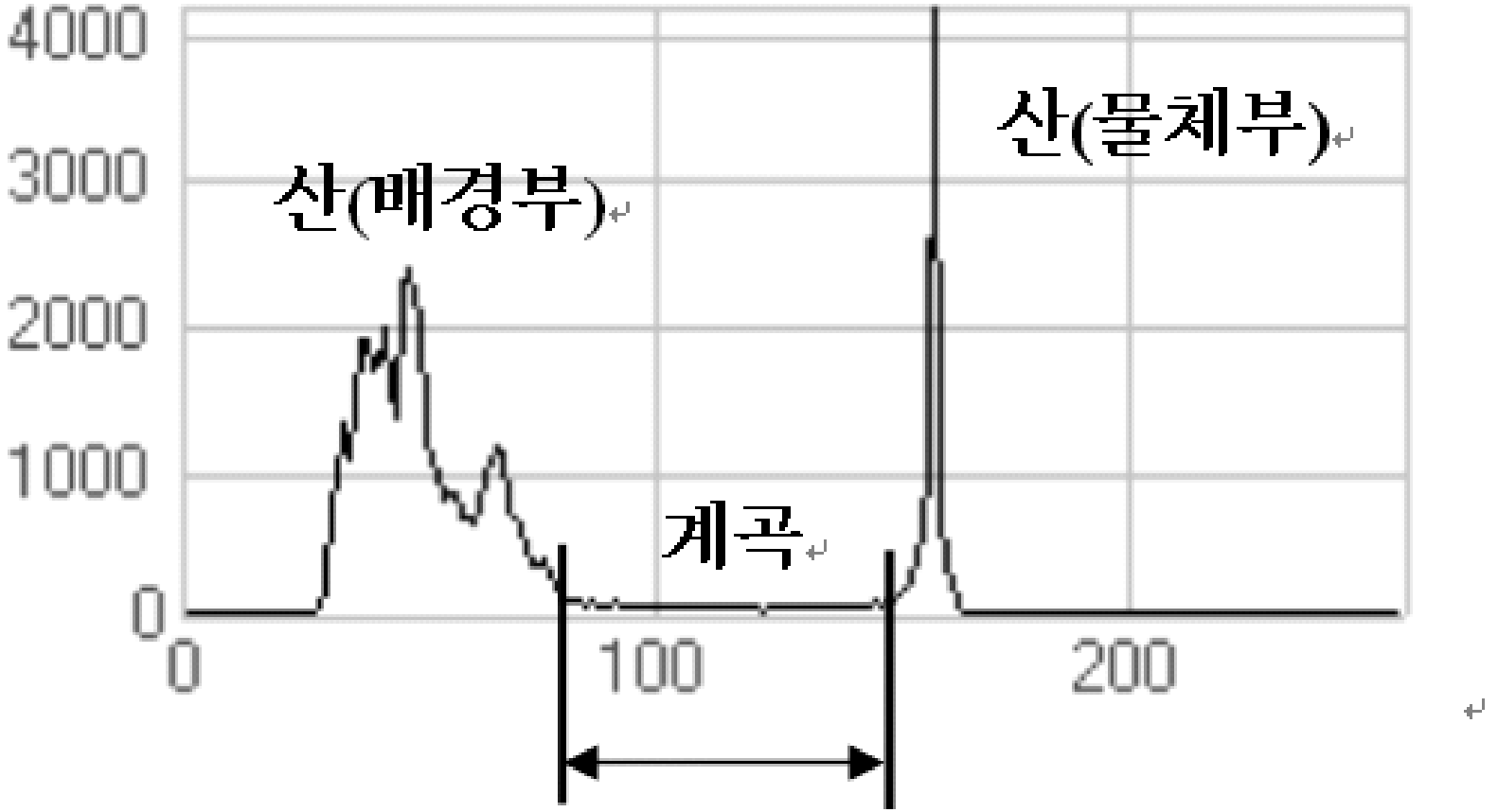
이치화(Binarization)

❖ 영상을 0과 255의 두 개의 값으로 표현



이치화 영상: 픽셀의 밝기값이 0 아니면 255의 두 값(binary values) 중 하나를 가짐

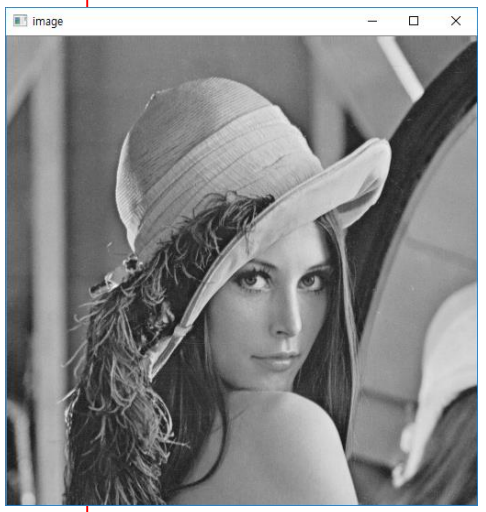
임계치(Threshold)



임계치(T)가 존재해야 하는 범위

continue…

```python
import cv2
import numpy as np

def main():
    filename = "lena.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    ysize = img.shape[0]
    xsize = img.shape[1]
    for y in range(ysize):
        for x in range(xsize):
            if img.item(y, x) < 128:
                img.itemset((y, x), 0)
            else:
                img.itemset((y, x), 255)
    cv2.imshow('bin-image', img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

main()
```



자동이치화(Otsu 알고리즘)

- 임계치를 자동으로 결정하는 알고리즘
- 높은 유사성(high homogeneity)을 갖는 픽셀들은 낮은 분산값 (Low Variance)을 가진다는 점을 이용
- 영상에는 대부분 배경과 물체가 있으며, 배경과 물체를 효과적으로 분리할 수 있는 임계치 t를 결정
- 즉, 배경부와 물체부의 두 그룹이 가능한 낮은 분산값을 가지도록 임계치 t를 결정

- 구현방법
  - 어떤 t를 0~255로 바꾸면서 분산값이 가장 작은 t를 찾음

Otsu 알고리즘

❖ 다음의 분산값을 최소화하는 t를 결정

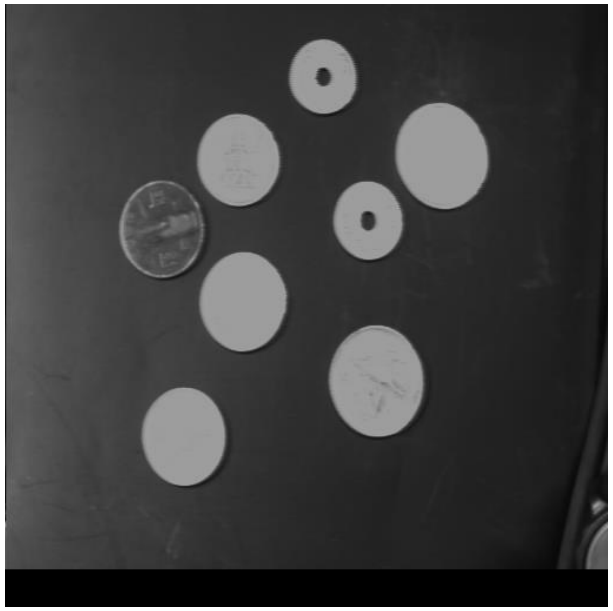
$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

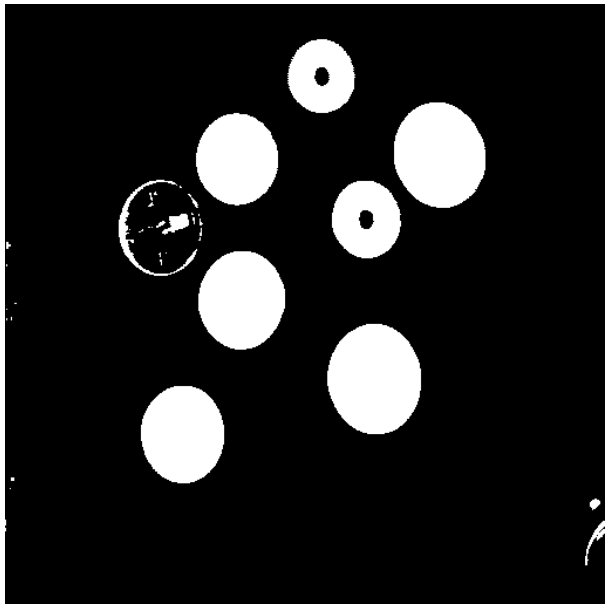
$P(i) = \#\{(r,c) \mid \text{Image}(r,c) == i\} / (\#R \times C)$: 밝기 i의 히스토그램 확률

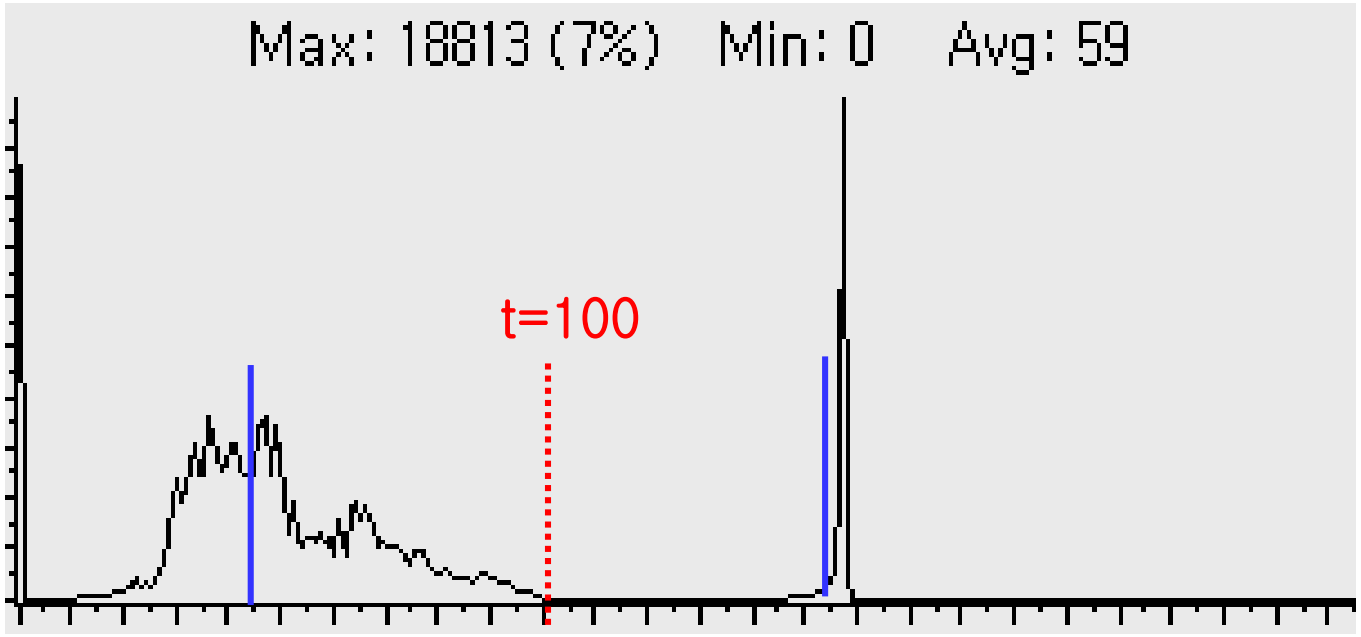
$$\begin{cases} q_1(t) = \sum_{i=1}^{t} P(i) & \text{: t보다 작은 픽셀그룹의 확률} \\ q_2(t) = \sum_{i=t+1}^{I} P(i) & \text{: t보다 큰 픽셀그룹의 확률} \end{cases}$$

$$q_1(t) + q_2(t) = 1$$

$$\begin{cases} \mu_1(t) = \sum_{i=1}^{t} iP(i)/q_1(t) & \sigma_1(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 P(i)/q_1(t) \\ \mu_2(t) = \sum_{i=t+1}^{I} iP(i)/q_2(t) & \sigma_2(t) = \sum_{i=t+1}^{I} [i-\mu_2(t)]^2 P(i)/q_2(t) \end{cases}$$







m1
(분산 큼)

M2
(분산 작음)

실험결과

- Optimal threshold: 100
- Group 1 Mean: 46.575
  Group 2 Mean: 152.622
- Group 1 Variance: 448.173
  Group 2 Variance: 133.493

continue…

```
import cv2
import numpy as np

def main():
    filename = "coin.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    cv2.imshow('image', img)

    # Create Histogram
    Hist = np.zeros((256)) # 0으로 초기화
    ysize = img.shape[0]
    xsize = img.shape[1]
    for y in range(ysize):
        for x in range(xsize):
            Hist[img.item(y,x)] = Hist[img.item(y,x)] + 1

    # Calculate Probability
    Prob = np.empty((256), dtype="float32")
    for i in range(256):
        Prob[i] = Hist[i] / img.size

    # Calculate threshold value: wsv_t
    wsv_min = float('inf') # float의 최대값, 최소값은 float('-inf')
    wsv_t = 0
```

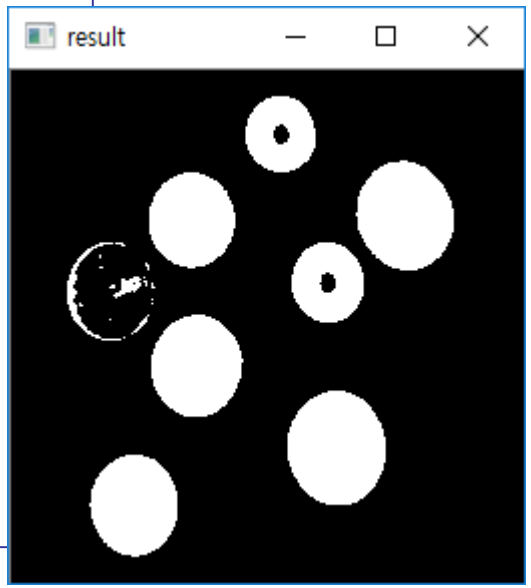
```
for t in range(256):
    # 두 집단의 확률 q1, q2 계산
    q1 = q2 = 0.0
    for i in range(t):
        q1 = q1 + Prob[i]
    for i in range(t, 256):
        q2 = q2 + Prob[i]
    if q1 == 0 or q2 == 0:
        continue

    # 두 집단의 평균 u1, u2 계산
    u1 = u2 = 0.0
    for i in range(t):
        u1 = u1 + i * Prob[i]
    u1 = u1 / q1
    for i in range(t, 256):
        u2 = u2 + i * Prob[i]
    u2 = u2 / q2

    # 두 집단의 분산 s1, s2 계산
    s1 = s2 = 0.0
    for i in range(t):
        s1 = s1 + pow(i-u1, 2) * Prob[i]
    s1 = s1 / q1
    for i in range(t, 256):
        s2 = s2 + pow(i-u2, 2) * Prob[i]
    s2 = s2 / q2
```

$$\begin{cases} q_1(t) = \sum_{i=1}^{t} P(i) \\ q_2(t) = \sum_{i=t+1}^{I} P(i) \end{cases}$$

$$\begin{cases} \mu_1(t) = \sum_{i=1}^{t} iP(i)/q_1(t) \\ \mu_2(t) = \sum_{i=t+1}^{I} iP(i)/q_2(t) \end{cases}$$

$$\begin{cases} \sigma_1(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 P(i)/q_1(t) \\ \sigma_2(t) = \sum_{i=t+1}^{I} [i-\mu_2(t)]^2 P(i)/q_2(t) \end{cases}$$

```
            # 수식 wsv를 계산하고 임계치 wsv_t를 결정
            wsv = q1 * s1 + q2 * s2
            if wsv < wsv_min:
                wsv_min = wsv
                wsv_t = t
        # end-for t

        # Thresholding by wsv_t
        for y in range(ysize):
            for x in range(xsize):
                if img.item(y, x) < wsv_t:
                    img.itemset((y, x), 0)
                else:
                    img.itemset((y, x), 255)
            # end-for x
        # end-for y

        cv2.imshow('result', img)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

main()
```



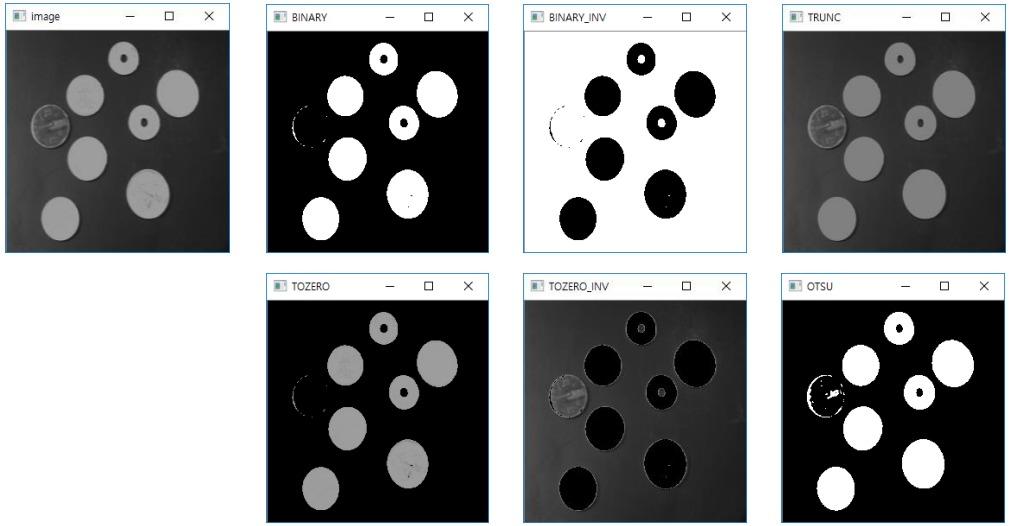
cv2.threshold() 함수

❖ cv2.threshold(img, th, maxValue, flag)

- img : Grayscale 이미지
- th : 임계값
- maxValue : 임계값보다 클 때 적용되는 값
- flag
  - cv2.THRESH_BINARY # 0 < th < maxValue
  - cv2.THRESH_BINARY_INV # maxValue < th < 0
  - cv2.THRESH_TRUNC # value < th < th
  - cv2.THRESH_TOZERO # 0 < th < value
  - cv2.THRESH_TOZERO_INV # value < th < 0
  - cv2.THRESH_OTSU # Otsu알고리즘적용

continue…

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def showImage():
    filename = "coin.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

    ret, result1 = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
    ret, result2 = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY_INV)
    ret, result3 = cv2.threshold(img, 128, 255, cv2.THRESH_TRUNC)
    ret, result4 = cv2.threshold(img, 128, 255, cv2.THRESH_TOZERO)
    ret, result5 = cv2.threshold(img, 128, 255, cv2.THRESH_TOZERO_INV)
    ret, result6 = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    cv2.imshow('BINARY', result1)
    cv2.imshow('BINARY_INV', result2)
    cv2.imshow('TRUNC', result3)
    cv2.imshow('TOZERO', result4)
    cv2.imshow('TOZERO_INV', result5)
    cv2.imshow('OTSU', result6)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```



적응이치화

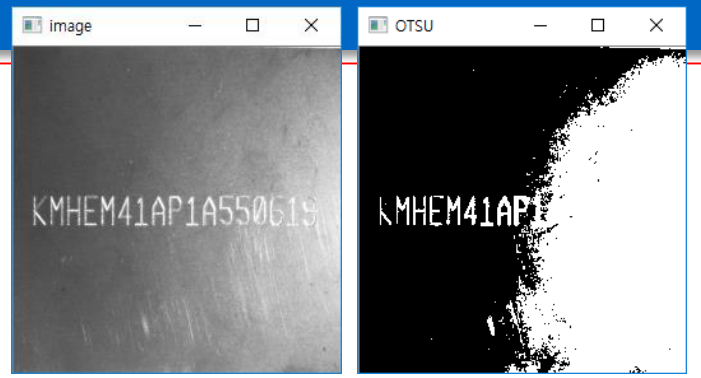
❖ 자동이치화(전역이치화)의 문제점

입력 영상

전역이치화

❖ 하나의 임계치(threshold)만으로 이치화를 하기 때문에 조명 등에 의해 밝기의 변화가 점진적으로 발생하는 경우에는 만족할만한 결과를 얻기 힘듦

cv2.adaptiveThreshold()




```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename

def showImage():
    filename = askopenfilename()    # filename = "VIN0.jpg"
    img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)

    ret, result = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    result1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 5, 2)
    result2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 5, 2)

    cv2.imshow('image', img)
    cv2.imshow('OTSU', result)
    cv2.imshow('MEAN_C', result1)
    cv2.imshow('GAUSSIAN_C', result2)

    cv2.waitKey(0)
    cv2.destroyAllWindows()

showImage()
```