스마트홈 IoT가전기기 제어를 위한 객체인식 모델구현

사전

딥러닝을 위한 파이썬 프로그래밍과 영상 처리 개념 이해(10)

KETI 한국전자기술연구원
Korea Electronics Technology Institute

학습내용

- OpenCV를 이용한 영상 처리 방법
  - 산술 연산 함수: 사칙/절대값/최대/최소 연산
  - 행렬 연산 함수: 행렬곱 연산
- 히스토그램 소개 및 실습

학습목표

- OpenCV를 이용한 영상 처리 방법을 설명할 수 있다.
- 히스토그램을 소개하고 실습할 수 있다.

OpenCV를 이용한 영상 처리 방법

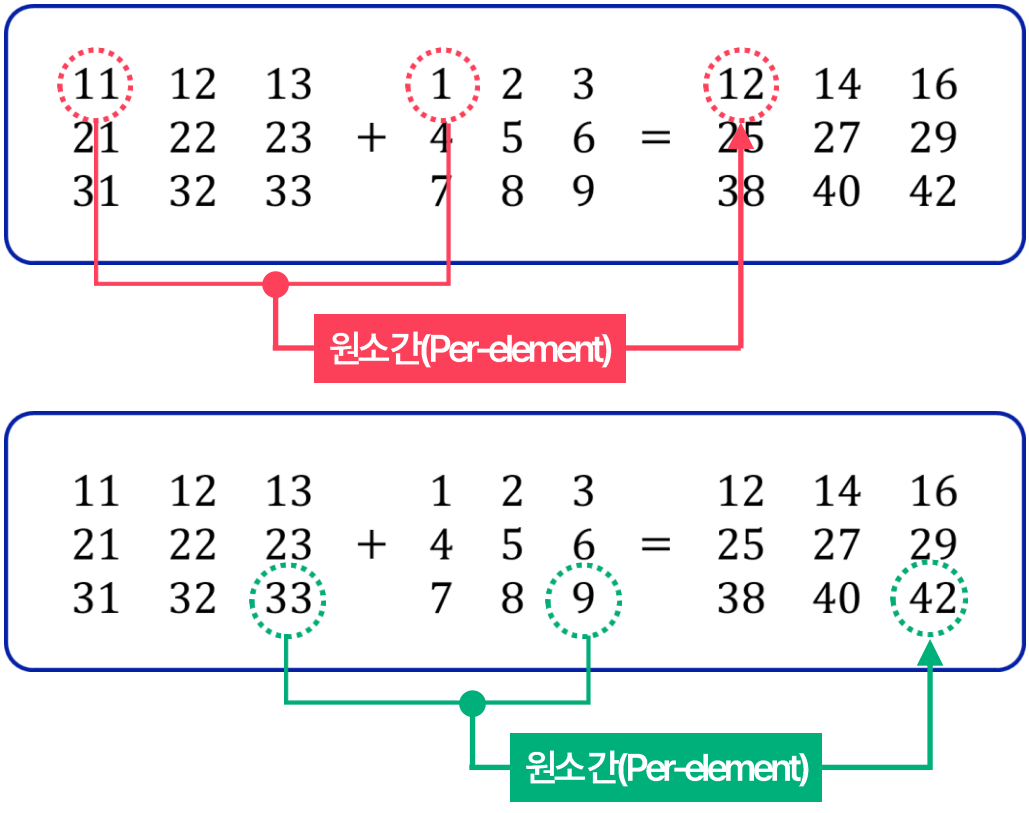
OpenCV를 이용한 영상처리 방법

◆ 산술 연산 함수

→ 영상 처리를 위해 행렬 간 산술 연산 진행

$$\begin{matrix}11 & 12 & 13\\ 21 & 22 & 23\\ 31 & 32 & 33\end{matrix} + \begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{matrix} = \begin{matrix}12 & 14 & 16\\ 25 & 27 & 29\\ 38 & 40 & 42\end{matrix}$$

- 원소 간(Per-element, Element-wise) 연산



→ 기존에는 행렬의 사칙연산을 위해 반복문을 통해 연산을 구현
→ OpenCV를 통해 행렬의 사칙연산 구현

OpenCV를 이용한 영상 처리 방법

사칙 연산 덧셈

- add 함수 사용

→ cv2.add(src1, src2[, dst[, mask[, dtype]]]) → dst

$$dst(i) = saturate\big(src1(i) + src2(i)\big) \quad if\ mask(i) \neq 0$$
$$dst(i) = saturate\big(src1 \quad + src2(i)\big) \quad if\ mask(i) \neq 0$$
$$dst(i) = saturate\big(src1(i) + src2 \quad\big) \quad if\ mask(i) \neq 0$$

연산을위한옵션적용

사칙 연산 뺄셈

- subtract 함수 사용

→ cv2.add(src1, src2[, dst[, mask[, dtype]]]) → dst

$$dst(i) = saturate\big(src1(i) - src2(i)\big) \quad if\ mask(i) \neq 0$$
$$dst(i) = saturate\big(src1 \quad - src2(i)\big) \quad if\ mask(i) \neq 0$$
$$dst(i) = saturate\big(src1(i) - src2 \quad\big) \quad if\ mask(i) \neq 0$$

OpenCV를 이용한 영상처리 방법

사칙 연산 곱셈

- multiply 함수 사용

→ cv2.multiply(src1, src2[, dst[, scale[, dtype]]]) → dst

$$dst(i) = saturate(scale \cdot src1(i)) \cdot src2(i)$$

- addWeighted 함수 사용

→ cv2.addWeighted(src1, alpha, src2, beta, gamma
[, dst[, dtype]]]) → dst

$$dst(i) \\ = saturate(src1(i) \cdot alpha + src2(i) \cdot beta + gamma$$

OpenCV를 이용한 영상처리 방법

◆ 행렬 산술 연산

예제 5.3.1

```
01  import numpy as np, cv2
02
03  m1 = np.full((3, 6), 10, np.uint8)
04  m2 = np.full((3, 6), 50, np.uint8)
05  m_mask = np.zeros(m1.shape, np.uint8)
06  m_mask[ :, 3: ] = 1
```

→ m1과 m2는 3, 6으로 동일한 사이즈
→ np.uint8을 통해 0~255까지 범위를 갖는 행렬로 정의
→ mask 옵션을 사용하여 특별한 행렬 연산 가능

예제 5.3.1

```
01  import numpy as np, cv2
02
03  m1 = np.full((3, 6), 10, np.uint8)
04  m2 = np.full((3, 6), 50, np.uint8)
05  m_mask = np.zeros(m1.shape, np.uint8)
06  m_mask[ :, 3: ] = 1
07
08  m_add1 = cv2.add(m1, m2)
09  m_add2 = cv2.add(m1, m2, mask=m_mask)
```

mask가 1인 영역만 연산수행

→ add 함수를 이용하여 m1과 m2 행렬을 더함
→ m1과 m2를 더한 결과는 변수 m_add1
→ mask 옵션을 통해 행렬의 앞부분을 0으로 초기화하고, 3라인 뒤쪽으로부터 1로 정의함

OpenCV를 이용한 영상처리 방법

◆ 행렬 산술 연산

예제 5.3.1

```
11  ## 행렬 나눗셈 수행
12  m_div1 = cv2.divide(m1, m2)
13  m1 = m1.astype(npfloat32)
14  m2 = np.float32(m2)
15  m_div2 = cv2.divide(m1, m2)
16
17  titles = ['m1', 'm2', 'm_mask', 'm_add1', 'm_add2', 'm_div1', 'm_div2']
18  for title in titles:
19      print("[%s] = \n%s \n" % (title, eval(title)))
```

→ divide 함수를 이용하여 m1과 m2 행렬을 나눔
→ m1과 m2를 나눈 결과는 변수 m_div1
→ divide 함수를 두 번 사용하여 구현
→ divide 함수

- 모든 값을 정수형으로 처리하는 함수
- 실수형으로 나오는 값을 정수형으로 보존하기 위해 사용

Run:

```
[m1] =
[[10. 10. 10. 10. 10. 10. ]
 [10. 10. 10. 10. 10. 10. ]
 [10. 10. 10. 10. 10. 10. ]]

[m2] =
[[50. 50. 50. 50. 50. 50. ]
 [50. 50. 50. 50. 50. 50. ]
 [50. 50. 50. 50. 50. 50. ]]

[m_mask] =
[[0 0 0 1 1 1]
 [0 0 0 1 1 1]
 [0 0 0 1 1 1]]

[m_add1] =
[[60 60 60 60 60 60]
 [60 60 60 60 60 60]
 [60 60 60 60 60 60]]

[m_add2] =
[[0 0 0 60 60 60]
 [0 0 0 60 60 60]
 [0 0 0 60 60 60]]

[m_div1] =
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]

[m_div2] =
[[0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]]
```

OpenCV를 이용한 영상 처리 방법

◆ 행렬 산술 연산

Run:

```
[m1] =
[[10. 10. 10. 10. 10. 10. ]
 [10. 10. 10. 10. 10. 10. ]
 [10. 10. 10. 10. 10. 10. ]]

[m2] =
[[50. 50. 50. 50. 50. 50. ]
 [50. 50. 50. 50. 50. 50. ]
 [50. 50. 50. 50. 50. 50. ]]

[m_mask] =
[[0 0 0 1 1 1]
 [0 0 0 1 1 1]
 [0 0 0 1 1 1]]

[m_add1] =
[[60 60 60 60 60 60]
 [60 60 60 60 60 60]
 [60 60 60 60 60 60]]

[m_add2] =
[[0 0 0 60 60 60]
 [0 0 0 60 60 60]
 [0 0 0 60 60 60]]

[m_div1] =
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]

[m_div2] =
[[0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]]
```

mask가 1인 영역만 연산수행

관심영역 [:,3:]

예제 5.3.1

```
01  import numpy as np, cv2
02
03  m1 = np.full((3, 6), 10, np.uint8)
04  m2 = np.full((3, 6), 50, np.uint8)
05  m_mask = np.zeros(m1.shape, np.uint8)
06  m_mask[ :, 3: ] = 1
07
08  m_add1 = cv2.add(m1, m2)
09  m_add2 = cv2.add(m1, m2, mask=m_mask)
10
11  ## 행렬 나눗셈 수행
12  m_div1 = cv2.divide(m1, m2)
13  m1 = m1.astype(npfloat32)
14  m2 = np.float32(m2)
15  m_div2 = cv2.divide(m1, m2)
16
17  titles = ['m1', 'm2', 'm_mask', 'm_add1', 'm_add2', 'm_div1', 'm_div2']
18  for title in titles:
19      print("[%s] = \n%s \n" % (title, eval(title)))
```

OpenCV를 이용한 영상 처리 방법

◆ 행렬 산술 연산

Run:

```
[m1] =
[[10. 10. 10. 10. 10. 10. ]
 [10. 10. 10. 10. 10. 10. ]
 [10. 10. 10. 10. 10. 10. ]]

[m2] =
[[50. 50. 50. 50. 50. 50. ]
 [50. 50. 50. 50. 50. 50. ]
 [50. 50. 50. 50. 50. 50. ]]

[m_mask] =
[[0 0 0 1 1 1]
 [0 0 0 1 1 1]
 [0 0 0 1 1 1]]

[m_add1] =
[[60 60 60 60 60 60]
 [60 60 60 60 60 60]
 [60 60 60 60 60 60]]

[m_add2] =
[[0 0 0 60 60 60]
 [0 0 0 60 60 60]
 [0 0 0 60 60 60]]

[m_div1] =
[[0 0 0 0 0 0]
 [0 0 0 0 0 0]
 [0 0 0 0 0 0]]

[m_div2] =
[[0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2 0.2]]
```

모든 값을 정수형으로 처리

데이터를 실수형으로 변환

예제 5.3.1

```
01  import numpy as np, cv2
02
03  m1 = np.full((3, 6), 10, np.uint8)
04  m2 = np.full((3, 6), 50, np.uint8)
05  m_mask = np.zeros(m1.shape, np.uint8)
06  m_mask[ :, 3: ] = 1
07
08  m_add1 = cv2.add(m1, m2)
09  m_add2 = cv2.add(m1, m2, mask=m_mask)
10
11  ## 행렬 나눗셈 수행
12  m_div1 = cv2.divide(m1, m2)
13  m1 = m1.astype(npfloat32)
14  m2 = np.float32(m2)
15  m_div2 = cv2.divide(m1, m2)
16
17  titles = ['m1', 'm2', 'm_mask', 'm_add1', 'm_add2', 'm_div1', 'm_div2']
18  for title in titles:
19      print("[%s] = \n%s \n" % (title, eval(title)))
```

OpenCV를 이용한 영상처리 방법

◆ 원소의 절대값 연산

- absdiff 함수

→ 두 배열간 각 원소 간(per-element) 차분 절대값을 계산

- 차분은 + 혹은 – 값이 나올 수 있음

$$dst(i) = saturate|src1(i) - src2(i)|$$
$$dst(i) = saturate|src1(i) - src2 \quad |$$
$$dst(i) = saturate|src1 \quad - src2(i)|$$

절대값적용

소스1 **소스2** **dst 변수**

→ cv2.absdiff(src1, src2[, dst]) → dst
→ 영상은 0~255 범위 내에서 표현
→ 영상 내에서 마이너스 값은 존재할 수 없음

$$dst(i) = saturate|src1(i) - src2(i)|$$
$$dst(i) = saturate|src1(i) - src2 \quad |$$
$$dst(i) = saturate|src1 \quad - src2(i)|$$

원하지않는마이너스값이
나오는것을방지하기위함

OpenCV를 이용한 영상처리 방법

◆ 원소의 절대값 연산

예제 5.4.1

```
01    import numpy as np, cv2
02
03    image1 = cv2.imread("images/abs_test1.jpg", cv2.IMREAD_GRAYSCALE)
04    image2 = cv2.imread("images/abs_test2.jpg", cv2.IMREAD_GRAYSCALE)
05    if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")
06
07    dif_img1 = cv2.subtract(image1, image2)
08    dif_img2 = cv2.subtract(np.int16(image1), np.int16(image2))
09    abs_dif1 = np.absolute(dif_img2).astype('uint8')
10    abs_dif2 = cv2.absdiff(image1, image2)
11
12    x, y, w, h = 100, 150, 7, 3
13    print("[dif_img1(roi) uint8] = \n%s\n" % dif_img1[y:y+h, x:x+w])
14    print("[dif_img2(roi) int16]  = \n%s\n" % dif_img2[y:y+h, x:x+w])
15    print("[dif_img1(roi)] = \n%s\n" % abs_dif1[y:y+h, x:x+w])
16    print("[dif_img2(roi)] = \n%s\n" % abs_dif2[y:y+h, x:x+w])
17
18    titles = ['image1', 'image2', 'dif_img1', 'abs_dif1', 'abs_dif2']
19    for title in titles:
20        cv2.imshow(title, eval(title))
21    cv2.waitKey(0)
```

→ subtract 함수를 사용해 image1에서 image2를 빼기
→ 빼기를 적용할 경우 마이너스 값이 발생할 수 있음
→ int16을 통해 – 값과 + 값이 모두 포함된 결과 저장
→ absolute 함수를 적용하여 – 값을 + 값으로 변환
→ absdiff에 image1, image2를 파라미터로 사용

OpenCV를 이용한 영상 처리 방법

◆ 원소의 절대값 연산

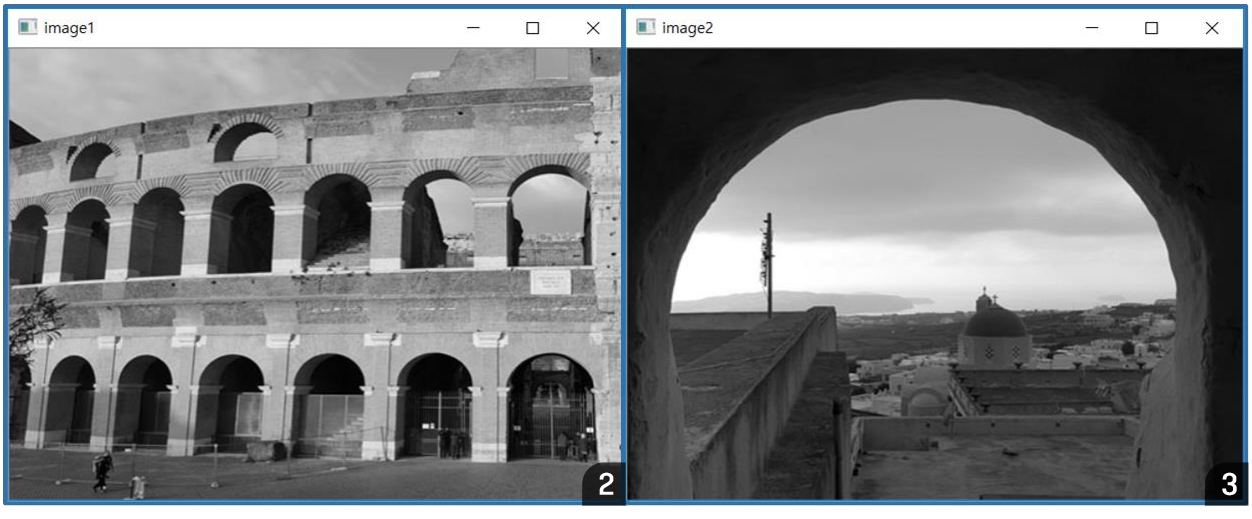
Run:

```
[dif_img1(roi) uint8] =
[[  0   0   0   0   9 12   7]
 [  0   0   0   0   4   9   3]
 [  0   0   0 15   0   4   0]]

[dif_img2(roi) int16] =
[[ -100 -106   -80    -6     9    12     7]
 [ -105 -109   -72    -4     4     9     3]
 [ -106 -109   -58    15    -1     4     0]]

[abs_dif1(roi)] =
[[ 100  106   80    6    9   12    7]
 [ 105  109   72    4    4    9    3]
 [ 106  109   58   15    1    4    0]]

[abs_dif2(roi)] =
[[ 100  106   80    6    9   12    7]
 [ 105  109   72    4    4    9    3]
 [ 106  109   58   15    1    4    0]]
```



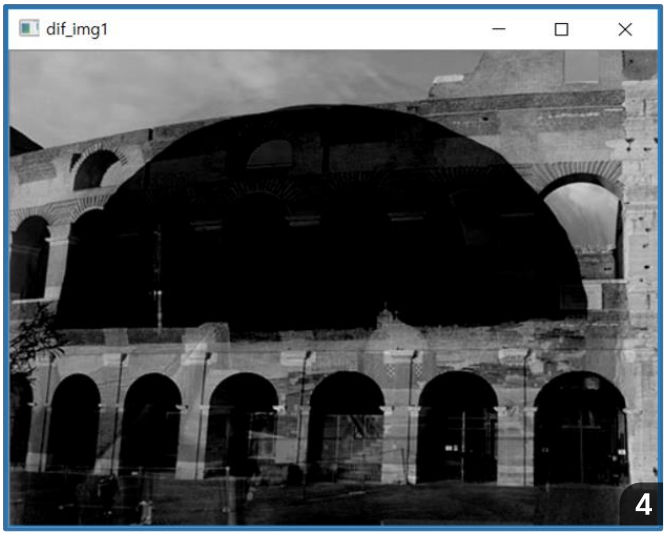
OpenCV를 이용한 영상처리 방법

◆ 원소의 절대값 연산

Run:

```
[dif_img1(roi) uint8] =
[[  0   0   0   0   9 12   7]
 [  0   0   0   0   4   9   3]
 [  0   0   0 15   0   4   0]]
[dif_img2(roi) int16] =
[[ -100 -106   -80    -6     9    12     7]
 [ -105 -109   -72    -4     4     9     3]
 [ -106 -109   -58    15    -1     4     0]]
[abs_dif1(roi)] =
[[ 100  106   80    6    9   12    7]
 [ 105  109   72    4    4    9    3]
 [ 106  109   58   15    1    4    0]]
[abs_dif2(roi)] =
[[ 100  106   80    6    9   12    7]
 [ 105  109   72    4    4    9    3]
 [ 106  109   58   15    1    4    0]]
```

→ 정수를 포함하는 변수를 사용해서 – 값이 모두 0으로 변함



OpenCV를 이용한 영상 처리 방법

◆ 원소의 절대값 연산

Run:

```
[dif_img1(roi) uint8] =
[[  0   0   0   0   9 12   7]
 [  0   0   0   0   4   9   3]
 [  0   0   0 15   0   4   0]]

[dif_img2(roi) int16] =
[[ -100 -106   -80    -6     9    12     7]
 [ -105 -109   -72    -4     4     9     3]
 [ -106 -109   -58    15    -1     4     0]]

[abs_dif1(roi)] =
[[ 100  106   80    6    9   12    7]
 [ 105  109   72    4    4    9    3]
 [ 106  109   58   15    1    4    0]]

[abs_dif2(roi)] =
[[ 100  106   80    6    9   12    7]
 [ 105  109   72    4    4    9    3]
 [ 106  109   58   15    1    4    0]]
```

absolute결과와 absdiff결과는동일

→ int16 데이터를 사용해 저장된 – 값을 확인할 수 있음

OpenCV를 이용한 영상처리 방법

원소의 최소값과 최대값

- 영상 안에 포함되어 있는 데이터 중 최소값을 갖는 영역과 최대값을 갖는 영역을 찾는 방법

예제 5.4.2

```
01  import numpy as np, cv2
02
03  daga = [10, 200, 5, 7, 9
04          15, 35, 60, 80, 170,
05          100, 2, 55, 37, 70 ]
06  m1 = np.reshape(data, (3, 5))
07  m2 = np.full((3, 5), 50)
08
09  m_min = cv2.min(m1, 30)
10  m_max = cv2.max(m1, m2)
11
12  ## 행렬의 최소값/최대값과 그 좌표들을 반환
13  min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(m1)
14
15  print("[m1] = \n%s\n" % m1)
16  print("[m_min] = \n%s\n" % m_min)
17  print("[m_max] = \n%s\n" % m_max)
18
19  ## min_loc와 max_loc 좌표는(y, x)이므로 행렬의 좌표 위치와 반대임
20  print("m1 행렬 최소값 좌표%s, 최소값: %d" %(min_loc, min_val) )
21  print("m1 행렬 최대값 좌표%s, 최대값: %d" %(max_loc, max_val) )
```

두개의행렬정의
m1과m2는3,5행렬로동일하지만
m2는50이라는값설정

min, max함수에서는
특정값을사용할수있음

행렬안에서최소값의위치와값,
최대값의위치와값을알려줌

→ min 함수: m1 행렬과 30을 비교하여 작은값을 출력
→ max 함수: m1행렬과 m2행렬을 비교하여 둘 중 큰 값 출력

OpenCV를 이용한 영상 처리 방법

◆ 원소의 최소값과 최대값

Run:

```
[m1] =
[[  10  200    5    7    9]
 [  15   35   60   80  170]
 [ 100    2   55   37   70]]
[m_min] =
[[  10   30    5    7    9]
 [  15   30   30   30   30]
 [  30    2   30   30   30]]
[m_max] =
[[  50  200   50   50   50]
 [  50   50   60   80  170]
 [ 100   50   55   50   70]]
m1 행렬 최소값 좌표 (1, 2), 최소값: 2
m1 행렬 최대값 좌표 (1, 0), 최대값: 200
```

```
[m2] =
[[  50   50   50   50   50]
 [  50   50   50   50   50]
 [  50   50   50   50   50]]
```

m1행렬의요소와
30을비교하여
30보다작은값출력

Run:

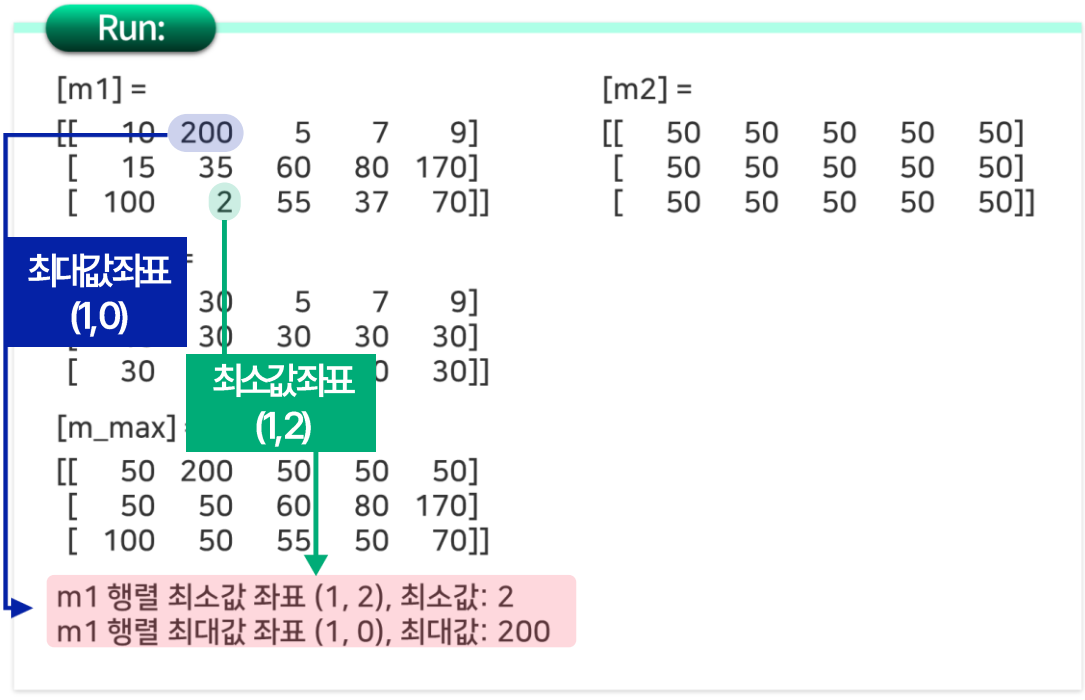
```
[m1] =
[[  10  200    5    7    9]
 [  15   35   60   80  170]
 [ 100    2   55   37   70]]
[m_min] =
[[  10   30    5    7    9]
 [  15   30   30   30   30]
 [  30    2   30   30   30]]
[m_max] =
[[  50  200   50   50   50]
 [  50   50   60   80  170]
 [ 100   50   55   50   70]]
m1 행렬 최소값 좌표 (1, 2), 최소값: 2
m1 행렬 최대값 좌표 (1, 0), 최대값: 200
```

```
[m2] =
[[  50   50   50   50   50]
 [  50   50   50   50   50]
 [  50   50   50   50   50]]
```

m1행렬의요소와
m2행렬의요소를비교하여
보다큰값출력

OpenCV를 이용한 영상 처리 방법

◆ 원소의 최소값과 최대값



OpenCV를 이용한 영상처리 방법

◆ 행렬 연산 함수

- Gemm(General Matrix Multiply) 함수

→ 일반화된 행렬 곱셈을 수행하는 함수

→ cv2.gemm(src1, src2, alpha, src3, beta[, dst[, flages]]) → dst

- alpha: 행렬 곱($src1$ ^T)에 대한 가중치
- beta: src3 행렬에 곱해지는 가중치

- Gemm(General Matrix Multiply) 함수

옵션	값	설명
cv2.GEMM_1_T	1	src1을 전치
cv2.GEMM_2_T	2	src2를 전치
cv2.GEMM_3_T	4	src3을 전치

$$dst = alpha \cdot src1^T \cdot src2 + beta \cdot src3^T$$

OpenCV를 이용한 영상처리 방법

◆ 행렬 연산 함수

예제 5.4.2

```
01    import numpy as np, cv2
02
03    src1 = np.array([1, 2, 3, 1, 2, 3], np.float32).reshape(2, 3)
04    src2 = np.array([1, 2, 3, 4, 5, 6], np.float32).reshape(2, 3)
05    src3 = np.array([1, 2, 1, 2, 1, 2], np.float32).reshape(3, 2)
06    alpha, beta = 1.0, 1.0
07
08    dst1 = cv2.gemm(src1, src2, alpha, None, beta, flags=cv2.GEMM_1_T)
09    dst2 = cv2.gemm(src1, src2, alpha, None, beta, flags=cv2.GEMM_2_T)
10    dst3 = cv2.gemm(src1, src2, alpha, None, beta)
11
12    titles = ['src1', 'src2', 'src3', 'dst1', 'dst2', 'dst3']
13    for title in titles:
14        print("[%s] = \n%s\n" % (title, eval(title)))
```

cv2.GEMM_1_T 옵션을 사용하여 src1을 전치하여 src1과 src2를 곱셈

$$src1^T \cdot src2 = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix}^T \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}1 & 1\\ 2 & 2\\ 3 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}5 & 7 & 9\\ 10 & 14 & 18\\ 15 & 21 & 27\end{bmatrix}$$

$$src1 \cdot src2^T = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}^T = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix} = \begin{bmatrix}14 & 32\\ 14 & 32\end{bmatrix}$$

$$src1 \cdot src3 = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2\\ 1 & 2\\ 1 & 2\end{bmatrix} = \begin{bmatrix}6 & 12\\ 6 & 12\end{bmatrix}$$

OpenCV를 이용한 영상처리 방법

◆ 행렬 연산 함수

Run:

```
[src1] =
[[ 1.  2.  3.]
 [ 1.  2.  3.]]
[src2] =
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
[src3] =
[[ 1.  2.]
 [ 1.  2.]
 [ 1.  2.]]
[dst1] =
[[ 5.  7.  9.]
 [10. 14. 18.]
 [15. 21. 27.]]
[dst2] =
[[14. 32.]
 [14. 32.]]
[dst3] =
[[ 6. 12.]
 [ 6. 12.]]
```

$$src1^T \cdot src2 = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix}^T \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}1 & 1\\ 2 & 2\\ 3 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}5 & 7 & 9\\ 10 & 14 & 18\\ 15 & 21 & 27\end{bmatrix}$$

$$src1 \cdot src2^T = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}^T = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix} = \begin{bmatrix}14 & 32\\ 14 & 32\end{bmatrix}$$

$$src1 \cdot src3 = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2\\ 1 & 2\\ 1 & 2\end{bmatrix} = \begin{bmatrix}6 & 12\\ 6 & 12\end{bmatrix}$$

히스토그램 소개 및 실습

히스토그램 소개 및 실습

◆ 히스토그램

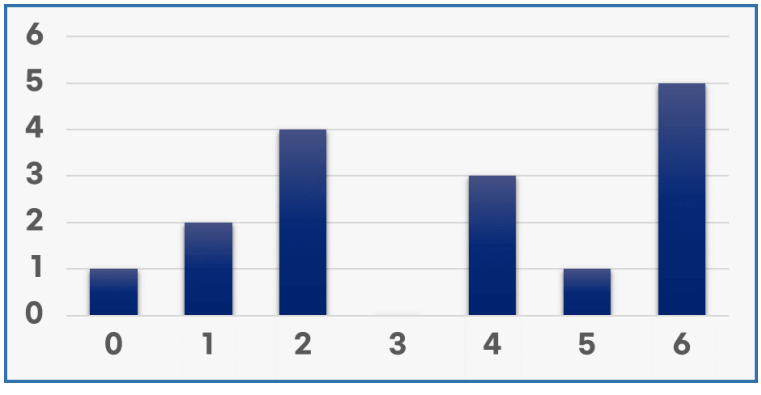
- 영상의 픽셀에 대한 명암값의 분포를 나타낸 것
- 어떤 데이터가 많은지를 나타내는 도수 분포표를 그래프로 나타낸 것

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

(a) 입력 영상

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

(a) 입력 영상



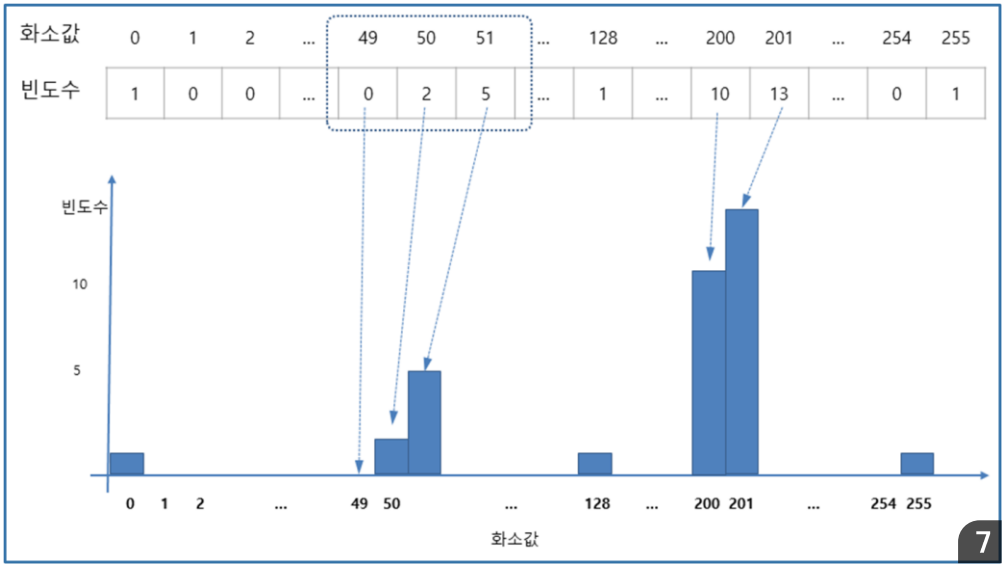
(b) 히스토그램

히스토그램 소개 및 실습

◆ 히스토그램

실행 결과

```
[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
```



히스토그램 소개 및 실습

◆ 히스토그램 계산

- 단일 채널 히스토그램 구현 함수

예제 6.3.1

calc_histo함수를정의하여입력이미지와 히스토그램사이즈,요소값의범위를설정

```
01    import numpy as np, cv2
02
03    def calc_histo(image, histSize, ranges=[0, 256] ):
04            hist = np.zeros((histSize, 1), np.float32)
05            gap = ranges[1] / histSize
06
07            for row in image:
08                for pix in row:
09                    idx = int(pix/gap)
10                    hist[idx] += 1
11            return hist
```

히스토그램 소개 및 실습

◆ 히스토그램 OpenCV 함수 활용

- calcHist 함수

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

예제 6.3.3

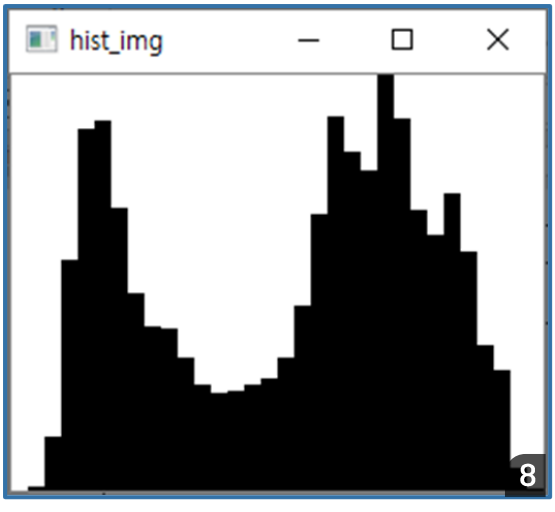
```
01  import numpy as np, cv2
02
03  def draw_histo(hist, shape=(200, 256)):
04          hist_img = np.full(shape, 255, np.uint8)
05          cv2.normalize(hist, hist, 0, shape[0], cv2.NORM_MINMAX)
06          gap = hist_img.shape[1]/hist.shape[0]
07
08          for i, h in enumerate(hist):
09                  x = int(round(i * gap))
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)
14
```

히스토그램 소개 및 실습

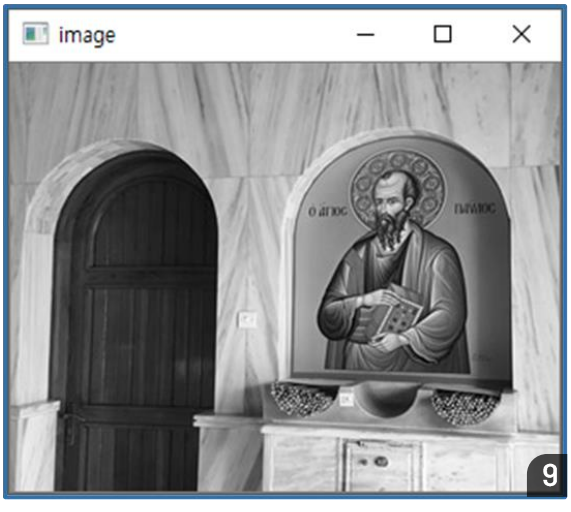
◆ 히스토그램 OpenCV 함수 활용

예제 6.3.3

```
15    image = cv2.imread("images/pixel.jpg", cv2.IMREAD_GRAYSCALE)
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", hisg_img)
23    cv2.waitKey(0)
```



8



9

정리하기

- **OpenCV를 이용한 영상 처리 방법**
  - → 산술 연산 함수
  - → 원소의 절대값 연산
  - → 행렬 연산 함수
- **히스토그램 소개 및 실습**
  - → 화소 밝기 변환
  - → 히스토그램 계산
  - → OpenCV 함수 활용