

CHAPTER 05

OPENCV 기본행렬 연산

5.1 기본 배열(ARRAY) 처리 함수

5.2 채널 처리 함수

5.3 산술 연산 함수

5.4 원소의 절댓값 연산

5.5 통계 관련 함수

5.6 행렬 연산 함수

1

5.1 기본 배열(Array) 처리 함수

◆파이썬에서는 배열을 처리하기 위한 자료형

❖ 열거형(sequence) 객체 - 리스트, 튜플, 사전(dictionary)

◆명칭 표현

❖ 1차원 데이터 - 벡터

❖ 2차원 데이터 - 행렬

❖ 1차원과 2차원 데이터 통칭해서 배열

2

5.1 기본 배열(Array) 처리 함수

◆기본 배열 처리 함수

함수 설명	
<code>cv2.flip(src, flipCode[, dst]) → dst</code> ■ 설명: 입력된 2차원 배열을 수직, 수평, 양측으로 뒤집는다.	
인수 설명	■ <code>src, dst</code> 입력 배열, 출력 배열 ■ <code>flipCode</code> 배열을 뒤집는 축 - 0: x축을 기준으로 위아래로 뒤집는다. - 1: y축을 기준으로 좌우로 뒤집는다. - -1: 양축(x축, y축 모두)을 기준으로 뒤집는다.
<code>cv2.repeat(src, ny, nx[, dst]) → dst</code> ■ 설명: 입력 배열의 반복된 복사본으로 출력 배열을 채운다.	
인수 설명	■ <code>src, dst</code> 입력 배열, 출력 배열 ■ <code>ny, nx</code> 수직 방향, 수평방향 반복 횟수
<code>cv2.transpose(src[, dst]) → dst</code> ■ 설명: 입력 행렬의 전치 행렬을 출력으로 반환한다.	
인수 설명	■ <code>src, dst</code> 입력 배열, 출력 배열

3

5.1 기본 배열(Array) 처리 함수

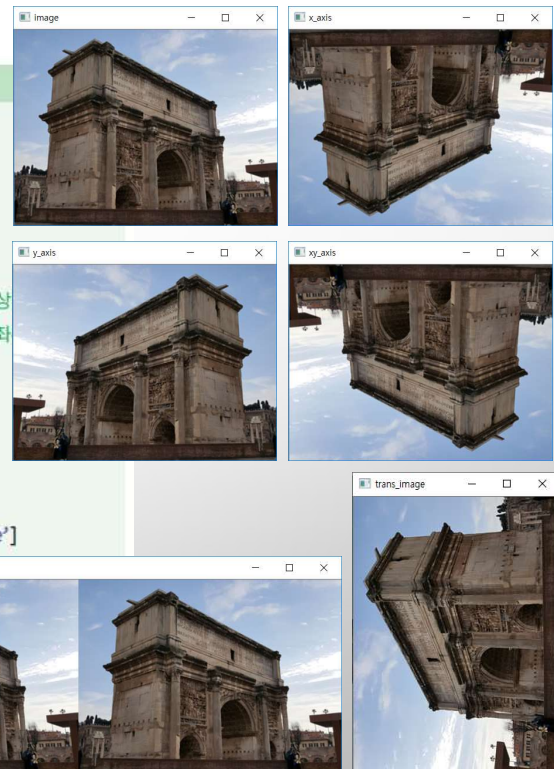
예제 5.1.1 행렬 처리 함수 - 01.mat_array.py

```

01 import cv2
02
03 image = cv2.imread("images/flip_test.jpg", cv2.IMREAD_COLOR)
04 if image is None: raise Exception("영상파일 읽기 오류 발생")    # 예외 처리
05
06 x_axis = cv2.flip(image, 0)                                     # x축 기준 상
07 y_axis = cv2.flip(image, 1)                                     # y축 기준 좌
08 xy_axis = cv2.flip(image, -1)
09 rep_image = cv2.repeat(image, 1, 2)                             # 반복 복사
10 trans_image = cv2.transpose(image)                               # 행렬 전치
11
12 ## 각 행렬을 영상으로 표시
13 titles = ['image', 'x_axis', 'y_axis', 'xy_axis', 'rep_image', 'trans_image']
14 for title in titles:
15     cv2.imshow(title, eval(title))
16 cv2.waitKey(0)

```

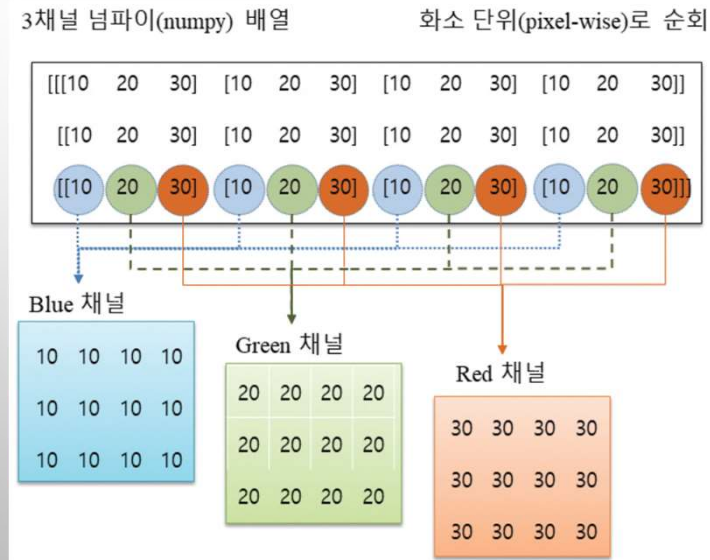
문자열을 명령어 만들
- 행렬 변수로 적용



4

5.2 채널 처리 함수

◆채널 개념



5

5.2 채널 처리 함수

◆채널 관련 함수

함수 설명		
<code>cv2.merge(mv[, dst]) → dst</code>		
■ 설명: 여러 개의 단일채널 배열을 다채널 배열로 합성한다.		
인수	■ mv	합성될 입력 배열 혹은 벡터, 합성될 단일채널 배열들의 크기와 깊이(depth)가 동일해야 함
설명	■ dst	입력 배열과 같은 크기와 같은 깊이의 출력 배열
<code>cv2.split(m[, mv]) → mv</code>		
■ 설명: 다채널 배열을 여러 개의 단일채널 배열로 분리한다.		
인수	■ m	입력되는 다채널 배열
설명	■ mv	분리되어 반환되는 단일채널 배열들의 벡터

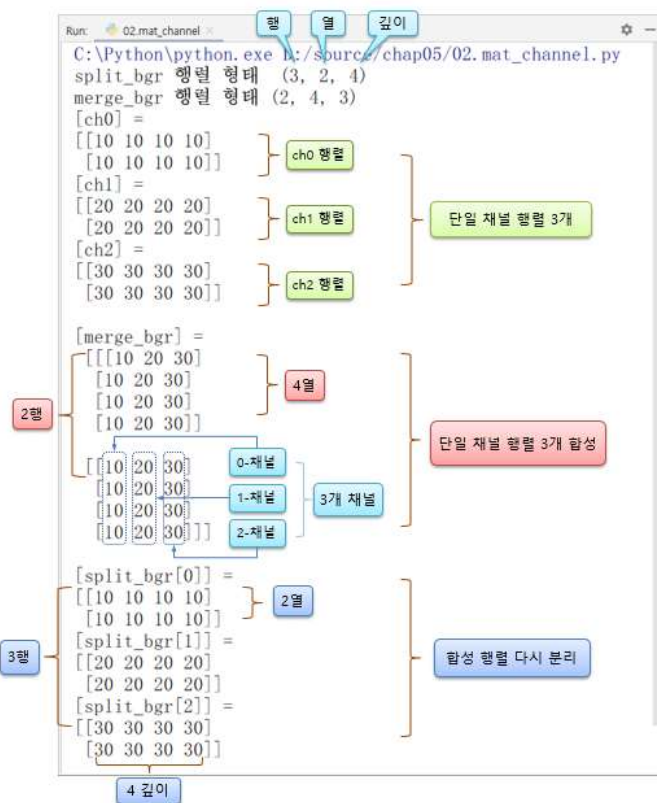
6

```

01 import numpy as np
02 import cv2
03
04 ## numpy.ndarray를 이용해 행렬 생성 및 초기화 방법
05 ch0 = np.zeros((2, 4), np.uint8) + 10          # 0 원소 행렬 선언 후 10 더하기
06 ch1 = np.ones((2, 4), np.uint8) * 20          # 1 원소 행렬 선언 후 20 곱하기
07 ch2 = np.full((2, 4), 30, np.uint8)           # 행렬을 생성하며 30으로 초기화
08
09 list_bgr = [ch0, ch1, ch2]                     # 단일채널 행렬들을 모아 리스트 구성
10 merge_bgr = cv2.merge(list_bgr)               # 채널 합성
11 split_bgr = cv2.split(merge_bgr)              # 채널 분리: 컬러 영상 → 3채널 분리
12
13
14 print("split_bgr 행렬 형태", np.array(split_bgr).shape)  # 행렬의 형태 확인
15 print("merge_bgr 행렬 형태", merge_bgr.shape)
16 print("[ch0] = \n%s" % ch0)                    # 단일채널 원소 출력
17 print("[ch1] = \n%s" % ch1)
18 print("[ch2] = \n%s\n" % ch2)
19 print("[merge_bgr] = \n %s\n" % merge_bgr)      # 다채널 원소 출력
20
21 print("[split_bgr[0]] =\n%s " % split_bgr[0])    # 분리 채널 결과 출력
22 print("[split_bgr[1]] =\n%s " % split_bgr[1])
23 print("[split_bgr[2]] =\n%s " % split_bgr[2])

```

7



8

5.2 채널 처리 함수

◆영상 채널 분리

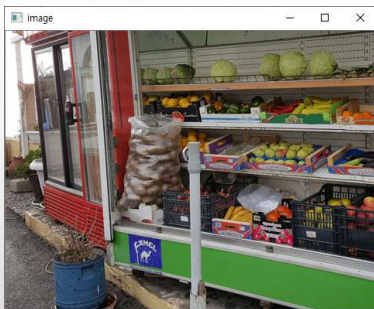
예제 5.2.2 컬러 채널 분리 - 03.image_channels.py

```
01 import cv2
02
03 image = cv2.imread("images/color.jpg", cv2.IMREAD_COLOR) # 영상 읽기
04 if image is None: raise Exception("영상파일 읽기 오류") # 예외 처리
05 if image.ndim != 3: raise Exception("컬러 영상 아님") # 예외 처리-컬러 영상 확인
06
07 bgr = cv2.split(image) # 채널 분리: 컬러 영상 → 3채널 분리
08 # blue, green, red = cv2.split(image) # 3개 변수로 반환받기 가능
09 print("bgr 자료형:", type(bgr), type(bgr[0]), type(bgr[0][0][0]) )
10 print("bgr 원소개수:" len(bgr))
11
12 ## 각 채널을 윈도우에 띄우기
13 cv2.imshow("image", image)
14 cv2.imshow("Blue channel", bgr[0]) # Blue 채널
15 cv2.imshow("Green channel", bgr[1]) # Green 채널
16 cv2.imshow("Red channel", bgr[2]) # Red 채널
17 # cv2.imshow("Blue channel", image[:, :, 0]) # 넘파이 객체 인덱싱 방식
18 # cv2.imshow("Green channel", image[:, :, 1])
19 # cv2.imshow("Red channel", image[:, :, 2])
20 cv2.waitKey(0)
```

9

5.2 채널 처리 함수

◆실행결과

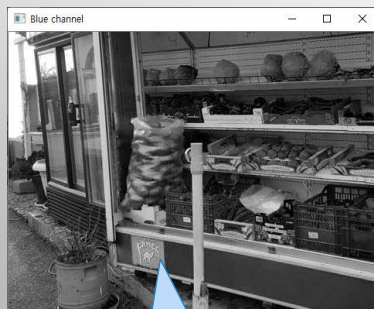


```
Run: 03.image_channels
C:\Python\python.exe D:/source/chap05/03.image_channels.py
bgr 자료형: <class 'list'> <class 'numpy.ndarray'> <class 'numpy.uint8'>
bgr 원소개수: 3
```

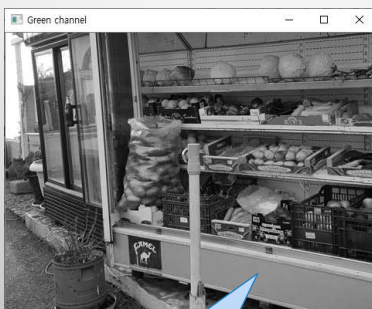
bgr 자료형

bgr 원소(단일채널)
자료형

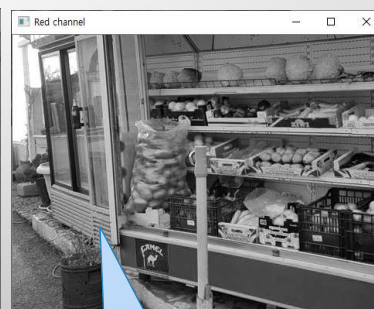
단일채널의 원
소 자료형



파란색 - 밝은색



녹색 - 밝은색



붉은색 - 밝은색

10

5.3 산술 연산 함수

◆ 원소간(per-element, element-wise) 연산 : 크기가 같은 두 입력영상으로 원소간 연산 후, 입력영상과 크기가 같은 결과영상을 생성

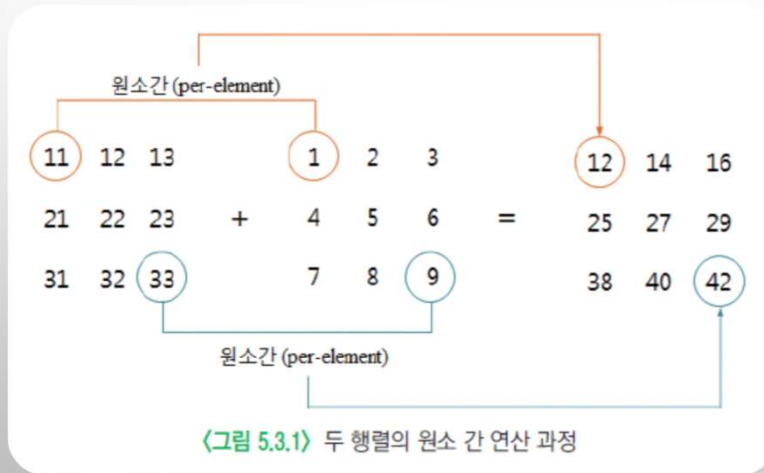
원소간 (per-element)

11	12	13		1	2	3		12	14	16
21	22	23	+	4	5	6	=	25	27	29
31	32	33		7	8	9		38	40	42

원소간 (per-element)

〈그림 5.3.1〉 두 행렬의 원소 간 연산 과정

◆ 원소간(per-element, element-wise) 연산 : 크기가 같은 두 입력영상으로 원소간 연산 후, 입력영상과 크기가 같은 결과영상을 생성



11

함수 설명

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

- 설명: 두 개의 배열 혹은 배열과 스칼라의 각 원소 간 합을 계산한다. 입력 인수 src1, src2 중 하나는 스칼라값일 수 있다.
- 수식:
$$\begin{aligned} dst(i) &= saturate(src1(i) + src2(i)) && \text{if } mask(i) \neq 0 \\ dst(i) &= saturate(src1 + src2(i)) && \text{if } mask(i) \neq 0 \\ dst(i) &= saturate(src1(i) + src2) && \text{if } mask(i) \neq 0 \end{aligned}$$

인수 설명	■src1	첫 번째 입력 배열 혹은 스칼라
	■src2	두 번째 입력 배열 혹은 스칼라
	■dst	계산된 결과의 출력 배열
	■mask	연산 마스크: 0이 아닌 마스크 원소의 위치만 연산 수행(8비트 단일채널)
	■dtype	출력 배열의 길이

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

- 설명: 두 개의 배열 혹은 배열과 스칼라의 각 원소 간 차분을 계산한다. add() 함수의 인수와 동일하다.
- 수식:

$$\begin{aligned} dst(i) &= saturate(src1(i) - src2(i)) && \text{if } mask(i) \neq 0 \\ dst(i) &= saturate(src1 - src2(i)) && \text{if } mask(i) \neq 0 \\ dst(i) &= saturate(src1(i) - src2) && \text{if } mask(i) \neq 0 \end{aligned}$$

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

- 설명: 두 배열의 각 원소 간 곱을 계산한다.
- 수식: $dst(i) = saturate(scale \cdot src1(i)) \cdot src2(i)$

인수 설명	<ul style="list-style-type: none"> scale : 두 배열의 원소 간 곱할 때 추가로 곱해주는 배율
-------	---

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

- 설명: 두 배열의 각 원소 간 나눗셈을 수행한다.
- 수식: $dst(i) = saturate(scale \cdot src1(i)/src2(i))$

```
cv2.divide(scale, src2[, dst[, dtype]]) → dst
```

- 설명: 스칼라값과 행렬원소간 나눗셈을 수행한다.
- 수식: $dst(i) = scale/src2(i)$

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

- 설명: 두 배열의 각 원소에 가중치를 곱한 후에 각 원소 간 합 즉, 가중된(weighted) 합을 계산한다.
- 수식: $dst(i) = saturate(src1(i) \cdot alpha + src2(i) \cdot beta + gamma)$

인수 설명	<ul style="list-style-type: none"> ■ alpha 첫 번째 배열의 모든 원소에 대한 가중치 ■ beta 두 번째 배열의 모든 원소에 대한 가중치 ■ gamma 두 배열의 원소 간 해를 추가로 더해주는 스칼라
----------	---

5.3.1 사칙 연산

12

5.3.1 사칙 연산

예제 5.3.1 행렬 산술 연산 - 04.arithmetic_op.py

```
01 import numpy as np, cv2
02
03 m1 = np.full((3, 6), 10, np.uint8)           # 단일채널 생성 및 초기화
04 m2 = np.full((3, 6), 50, np.uint8)
05 m_mask = np.zeros(m1.shape, np.uint8)        # 마스크 생성
06 m_mask[:, 3:] = 1                             # 관심 영역을 지정한 후
07
08 m_add1 = cv2.add(m1, m2)                      # 행렬 덧셈
09 m_add2 = cv2.add(m1, m2, mask=m_mask)        # 관심 영역만 덧셈 수행
10
11 ## 행렬 나눗셈 수행
12 m_div1 = cv2.divide(m1, m2)
13 m1 = m1.astype(np.float32)                   # 소수 부분 보존위해 행
14 m2 = np.float32(m2)                          # 형변환 방법2
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 = %s\n" % (title, eval(title)))
```

```
[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]]
```

관심 영역
[:, 3:]

mask 원소가
1인 위치만 연산 수행

```
[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]]
```

나눗셈으로 인한
소수점 이하 값이 소실

```
[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]]
```

형변환으로
소수점 이하 값 유지

13

함수 설명

cv2.exp(src[, dst]) → dst

- 설명: 모든 배열 원소의 지수(exponent)를 계산한다.
- 수식: $dst(i) = e^{src(i)}$

인수	■ src, dst	입력 배열, 입력 배열과 같은 크기와 타입의 출력 배열
----	------------	--------------------------------

cv2.log(src[, dst]) → dst

- 설명: 모든 배열 원소의 절대값에 대한 자연 로그를 계산한다.
- 수식: $dst(i) = \begin{cases} \log|src(i)| & \text{if } src(i) \neq 0 \\ c & \text{otherwise} \end{cases}$

cv2.sqrt(src[, dst]) → dst

- 설명: 모든 배열 원소에 대해 제곱근을 계산한다.
- 수식: $dst(i) = \sqrt{src(i)}$

cv2.pow(src, power[, dst]) → dst

- 설명: 모든 배열 원소에 대해서 제곱 승수를 계산한다.
- 수식: $dst(i) = \begin{cases} src(i)^{power} & \text{if } power \text{ is integer} \\ |src(i)|^{power} & \text{otherwise} \end{cases}$

인수	■ power	제곱 승수
----	---------	-------

cv2.magnitude(x, y[, magnitude]) → magnitude

- 설명: 2차원 배열들의 크기(magnitude)를 계산한다.
- 수식: $magnitude(i) = \sqrt{x(i)^2 + y(i)^2}$

인수	■ x, y	x, y 좌표들의 입력 배열
설명	■ magnitude	입력 배열과 같은 크기의 출력 배열

cv2.phase(x, y[, angle[, angleInDegrees]]) → angle

- 설명: 2차원 배열의 회전 각도를 계산한다.
- 수식: $angle(i) = \tan^{-1}(y(i), x(i)) \cdot [180/\pi]$

인수	■ angle	각도들의 출력 배열
설명	■ angleInDegrees	True: 각을 도(degree)로 측정, False: 각을 라디안(radian)으로 측정

cv2.cartToPolar(x, y[, magnitude[, angle[, angleInDegrees]]]) → magnitude, angle

- 설명: 2차원 배열들의 크기(magnitude)와 각도를 계산한다.
- 수식: $magnitude(i) = \sqrt{x(i)^2 + y(i)^2}$
 $angle(i) = \tan^{-1}(y(i), x(i)) \cdot [180/\pi]$

cv2.polarToCart(magnitude, angle[, x[, y[, angleInDegrees]]]) → x, y

- 설명: 각도와 크기(magnitude)로부터 2차원 배열들의 좌표를 계산한다.
- 수식: $x(i) = magnitude(i) \cdot \cos(angle(i))$
 $y(i) = magnitude(i) \cdot \sin(angle(i))$

5.3.2 지수, 로그, 제곱근 관련 함수

14

5.3.2 지수, 로그, 제곱근 관련 함수

```

01 import numpy as np, cv2
02
03 ## ndarray 생성 예시
04 v1 = np.array([1, 2, 3], np.float32)           # 1차원 리스트로 행렬 생성
05 v2 = np.array([[1], [2], [3]], np.float32)     # 2차원 리스트(3행, 1열) - 열벡터
06 v3 = np.array([[1, 2, 3]], np.float32)         # 2차원 리스트(1행, 3열) - 행벡터
07
08 ## OpenCV 산술 연산 함수 - 입력 인수로 ndarray 객체만 가능한
09 v_exp = cv2.exp(v1)                           # 1차원 행렬에 대한 지수
10 m_exp = cv2.exp(v2)                           # 행벡터(1x3)에 대한 지수 계산
11 m_exp = cv2.exp(v3)                           # 열벡터(3x1)에 대한 지수 계산
12 v_log = cv2.log(v1)                           # 로그 계산
13 m_sqrt = cv2.sqrt(v2)                         # 제곱근 계산
14 m_pow = cv2.pow(v3, 3)                        # 3의 거듭제곱 계산
15
16 ## 행렬 정보 결과 출력
17 print("[v1] 형태: %s 원소: %s" % (v1.shape, v1))
18 print("[v2] 형태: %s 원소: %s" % (v2.shape, v2))
19 print("[v3] 형태: %s 원소: %s" % (v3.shape, v3))
20 print()
21
22 ## 행렬 정보 출력 - OpenCV 결과는 행렬로 반환됨
23 print("[v1_exp] 자료형: %s 형태: %s" % (type(v1_exp), v1_exp.shape))
24 print("[v2_exp] 자료형: %s 형태: %s" % (type(v2_exp), v2_exp.shape))
25 print("[v3_exp] 자료형: %s 형태: %s" % (type(v3_exp), v3_exp.shape))
26 print()
27
28 ## 열벡터를 1행에 출력하는 예시
29 print("[log] =", log.T)                       # 전치하여 행벡터(1행, n열)로 변경
30 print("[sqrt] =", np.ravel(sqrt))             # 전개하여 1차원 행렬로 변경
31 print("[pow] =", pow.flatten())               # 전개하여 1차원 행렬로 변경

```

15

5.3.2 지수, 로그, 제곱근 관련 함수

◆ 실행 결과

3행, 1열 → 열벡터

3열 → 1차원 행렬

1행, 3열 → 행벡터

OpenCV 함수에서 행렬로 반환하는 것은 대부분 ndarray 객체임

2차원 행렬로 반환

```

Run: 05.exp_log
C:\Python\python.exe D:/source/chap05/05.exp_log.py
[v1] 형태: (3,) 원소: [1. 2. 3.]
[v2] 형태: (3, 1) 원소:
[[1.]
 [2.]
 [3.]]
[v3] 형태: (1, 3) 원소: [[1. 2. 3.]]

[v1_exp] 자료형: <class 'numpy.ndarray'> 형태: (3, 1)
[v2_exp] 자료형: <class 'numpy.ndarray'> 형태: (3, 1)
[v3_exp] 자료형: <class 'numpy.ndarray'> 형태: (1, 3)

[log] = [[0.          0.6931472  1.0986123]]
[sqrt] = [1.          1.4142135  1.7320508]
[pow] = [ 1.   8.  27.]

```

16

5.3.2 지수, 로그, 제곱근 관련 함수

예제 5.3.3 행렬 크기 및 위상 연산 - 06.magnitude.py

```
01 import numpy as np, cv2
02
03 x = np.array([1, 2, 3, 5, 10], np.float32)           # 리스트로 ndarray 객체 생성
04 y = np.array([2, 5, 7, 2, 9]).astype("float32")      # 행렬 생성 후 실수형 변환
05
06 mag = cv2.magnitude(x, y)                           # 크기 계산
07 ang = cv2.phase(x, y)                               # 각도(방향) 계산
08 p_mag, p_ang = cv2.cartToPolar(x, y)                # 극 좌표로 변환
09 x2, y2 = cv2.polarToCart(p_mag, p_ang)              # 직교좌표로 변환
10
11 print("[x] 형태: %s 원소: %s" % (x.shape, x) )      # 1차원 행렬
12 print("[mag] 형태: %s 원소: %s" % (mag.shape, mag) ) # 2차원 열벡터
13
14 print(">>>열벡터를 1행에 출력하는 방법")
15 print("[m_mag] = %s" % mag.T)                       # 행렬 전치
16 print("[p_mag] = %s" % np.ravel(p_mag))              # ravel() 함수로 전개
17 print("[p_ang] = %s" % np.ravel(p_ang))
18 print("[x_mat2] = %s" % x2.flatten())                 # 2차원 행렬 전개
19 print("[y_mat2] = %s" % y2.flatten())
```

17

5.3.2 지수, 로그, 제곱근 관련 함수

◆실행 결과

```
Run: 06.magnitude
C:\Python\python.exe D:/source/chap05/06.magnitude.py
[x] 형태: (5,) 원소: [ 1.  2.  3.  5. 10.]
[mag] 형태: (5, 1) 원소: [[ 2.236068 ]
 [ 5.3851647 ]
 [ 7.615773 ]
 [ 5.3851647 ]
 [13.453624 ]]
>>>열벡터를 1행에 출력하는 방법
[m_mag] = [[ 2.236068  5.3851647  7.615773  5.3851647 13.453624 ]]
[p_mag] = [ 2.236068  5.3851647  7.615773  5.3851647 13.453624 ]
[p_ang] = [1.1071129 1.1902124 1.1658309 0.3805839 0.7329612]
[x_mat2] = [1.0000718 2.000388 3.0005162 4.999845 9.998685 ]
[y_mat2] = [1.9999641 4.9998446 6.999779 2.0003877 9.001461 ]
```

18

5.3.3 논리(비트) 연산 함수

함수 설명	
cv2.bitwise_and(src1, src2[, dst[, mask]]) → dst	
<ul style="list-style-type: none"> ■ 설명: 두 배열의 원소 간 혹은 배열 원소와 스칼라 간의 비트별(bit-wise) 논리곱(AND) 연산을 수행한다. 입력 인수 src1, src2 중 하나는 스칼라값일 수 있다. ■ 수식: $dst(i) = src1(i) \wedge src2(i)$ if $mask(i) \neq 0$ $dst(i) = src1(i) \wedge src2$ if $mask(i) = 0$ $dst(i) = src1 \wedge src2(i)$ if $mask(i) = 0$ 	
cv2.bitwise_or(src1, src2[, dst[, mask]]) → dst	
<ul style="list-style-type: none"> ■ 설명: 두 개의 배열 원소 간 혹은 배열 원소와 스칼라 간의 비트별 논리합(OR) 연산을 수행한다. ■ 수식: $dst(i) = src1(i) \vee src2(i)$ if $mask(i) \neq 0$ $dst(i) = src1(i) \vee src2$ if $mask(i) = 0$ $dst(i) = src1 \vee src2(i)$ if $mask(i) = 0$ 	
cv2.bitwise_xor(src1, src2[, dst[, mask]]) → dst	
<ul style="list-style-type: none"> ■ 설명: 두 개의 배열 원소 간 혹은 배열 원소와 스칼라 간의 비트별 배타적 논리합(XOR) 연산을 수행한다. 	
cv2.bitwise_not(src[, dst[, mask]]) → dst	
<ul style="list-style-type: none"> ■ 설명: 입력 배열의 모든 원소마다 비트 보수 연산을 한다. 쉽게 말하자면 반전시킨다. ■ 수식: $dst(i) = \sim src(i)$ 	
인수 설명	<ul style="list-style-type: none"> ■ src1 첫 번째 입력 배열 혹은 스칼라값 ■ src2 두 번째 입력 배열 혹은 스칼라값 ■ dst 입력 배열과 같은 크기의 출력 배열 ■ mask 마스크 연산 수행(8비트 단일채널 배열) - 마스크 배열의 원소가 0이 아닌 좌표만 계산을 수행

19

5.3.3 논리(비트) 연산 함수

예제 5.3.4 행렬 비트 연산 - 07.bitwise_op.py

```

01 import numpy as np, cv2
02
03 image1 = np.zeros((300, 300), np.uint8)           # 300행, 300열 검은색(0) 영상 생성
04 image2 = image1.copy()                             # image1 복사
05
06 h, w = image1.shape[:2]
07 cx, cy = w//2, h//2                                내부 채움
08 cv2.circle(image1, (cx, cy), 100, 255, -1)         # 중심 좌표
09 cv2.rectangle(image2, (0, 0, cx, h), 255, -1)      # 중심에 원 그리기
10                                                    # 영상의 가로 절반
11 image3 = cv2.bitwise_or(image1, image2)            # 원소 간 논리합
12 image4 = cv2.bitwise_and(image1, image2)           # 원소 간 논리곱
13 image5 = cv2.bitwise_xor(image1, image2)           # 원소 간 배타적 논리합
14 image6 = cv2.bitwise_not(image1)                   # 행렬 반전
15
16 cv2.imshow("image1", image1); cv2.imshow("image2", image2)
17 cv2.imshow("bitwise_or", image3); cv2.imshow("bitwise_and", image4)
18 cv2.imshow("bitwise_xor", image5); cv2.imshow("bitwise_not", image6)
19 cv2.waitKey(0)

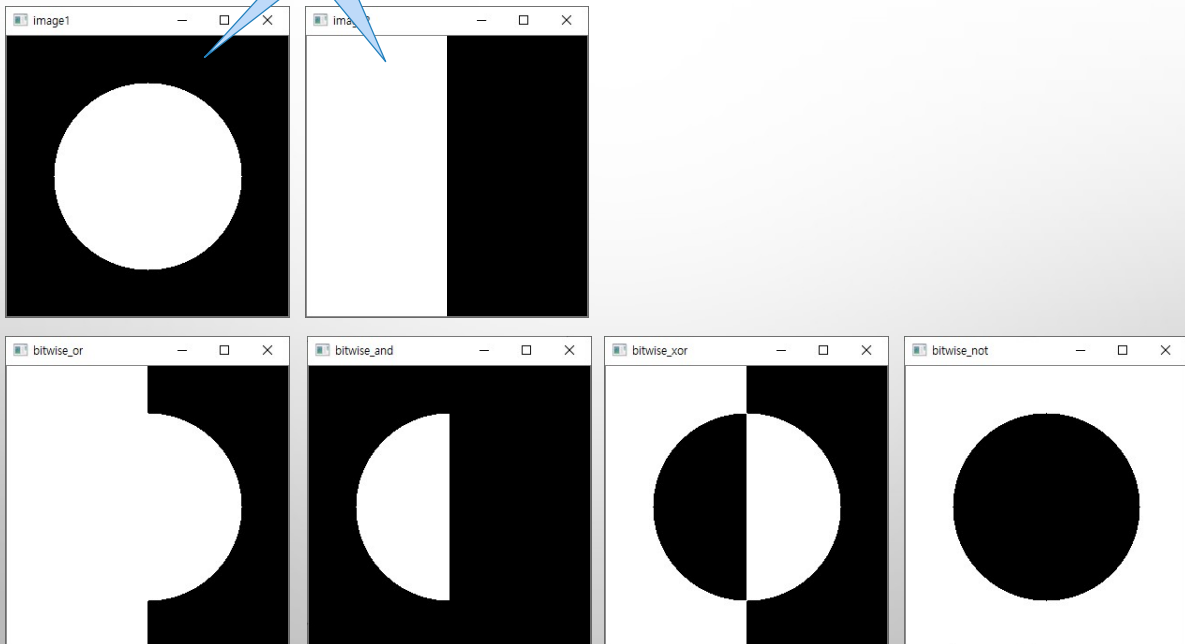
```

20

5.3.3 논리(비트) 연산 함수

◆ 실행 결과

입력 영상



21

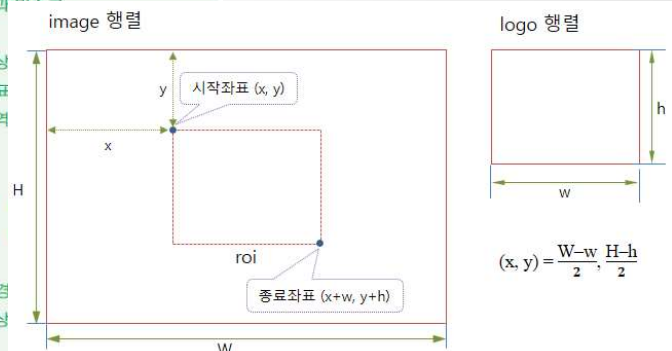
심화예제 5.3.4 행렬 비트 연산2 - 08.bitwise_overlap.py

5.3.3 논리(비트) 연산 함수

```
01 import numpy as np, cv2
02
03 image = cv2.imread("images/bit_test.jpg", cv2.IMREAD_COLOR) # 원본 영상 읽기
04 logo = cv2.imread("images/logo.jpg", cv2.IMREAD_COLOR) # 로고 영상 읽기
05 if image is None or logo is None: raise Exception("영상파일 읽기 오류")
06
07 masks = cv2.threshold(logo, 220, 255, cv2.THRESH_BINARY)[1] # 로고 영상 이진화
08 masks = cv2.split(masks)
09
10 fg_pass_mask = cv2.bitwise_or(masks[0], masks[1])
11 fg_pass_mask = cv2.bitwise_or(masks[2], fg_pass_mask) # 전경 통과 마스크
12 bg_pass_mask = cv2.bitwise_not(fg_pass_mask) # 배경 통과 마스크
13
14 (H, W), (h, w) = image.shape[:2], logo.shape[:2] # 전체 영상
15 x, y = (W-w)//2, (H-h)//2 # 시작 좌표
16 roi = image[y:y+h, x:x+w] # 관심 영역
17
18 ## 행렬 논리곱과 마스크를 이용한 관심 영역 복사
19 foreground = cv2.bitwise_and(logo, logo, mask=fg_pass_mask)
20 background = cv2.bitwise_and(roi, roi, mask=bg_pass_mask)
21
22 dst = cv2.add(background, foreground) # 로고 전경
23 image[y:y+h, x:x+w] = dst # 합성 영상
24
25 cv2.imshow('background', background); cv2.imshow('foreground', foreground)
26 cv2.imshow('dst', dst); cv2.imshow('image', image)
27 cv2.waitKey()
```

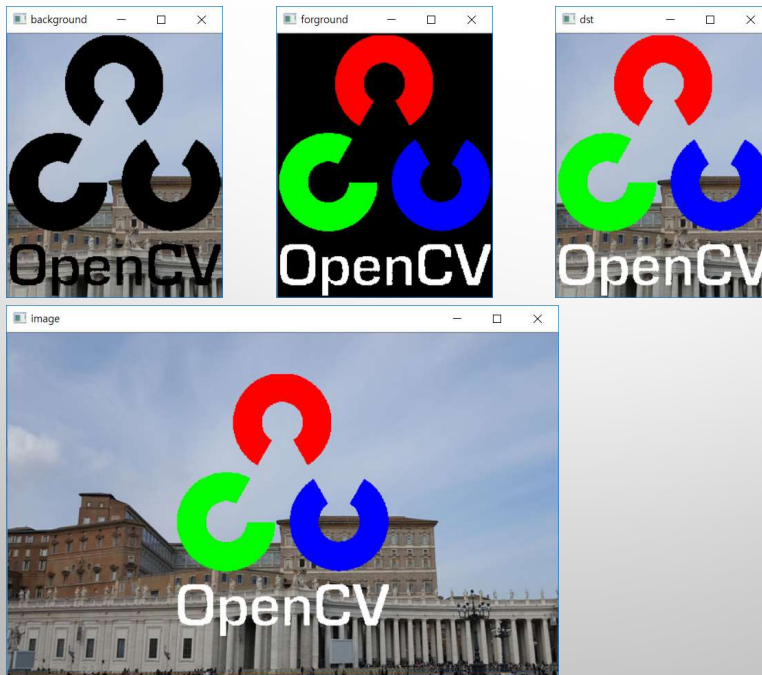
220보다 크면 255(흰색),
아니면 0(검정색)
6장에서 자세히

전경 & 배경 마스크



22

◆ 실행결과



23

5.4 원소의 절댓값 연산

함수 설명		
cv2.absdiff(src1, src2[, dst]) → dst		
■ 설명: 두 배열간 각 원소간(per-element) 차분 절댓값을 계산한다. src1, src2 중 하나는 스칼라값이 될 수 있다.		
■ 수식: $dst(i) = saturate(src1(i) - src2(i))$		
$dst(i) = saturate(src1(i) - src2(i))$		
$dst(i) = saturate(src1(i) - src2(i))$		
인수	■ src1, src2	첫 번째 입력 배열, 두 번째 입력 배열
설명	■ dst	계산된 결과 출력 배열
cv2.convertScaleAbs(src[, dst[, alpha[, beta]]]) → dst		
■ 설명: 입력 배열의 각 원소에 alpha만큼 배율을 곱하고 beta 만큼 더한 후에 절댓값을 계산한 결과를 8비트 자료형으로 변환한다.		
■ 수식: $dst(i) = saturate(cast<uchar>(src(i)*\alpha + \beta))$		
인수	■ alpha	입력 배열의 각 원소에 곱해지는 스케일 팩터(scale factor)
설명	■ beta	스케일된 값에 더해지는 델타 옵션

24

5.4 원소의 절댓값 연산

예제 5.4.1 행렬 절댓값 및 차분 연산 - 10.mat_abs.py

```
01 import numpy as np, cv2
02
03 image1 = cv2.imread("images/abs_test1.jpg", cv2.IMREAD_GRAYSCALE) # 명암도 영상 읽기
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) # 차분 연산
08 dif_img2 = cv2.subtract(np.int16(image1), np.int16(image2)) # 음수 결과 보존
09 abs_dif1 = np.absolute(dif_img2).astype('uint8')
10 abs_dif2 = cv2.absdiff(image1, image2) # 차분 절댓값 계산
11
12 x, y, w, h = 100, 150, 7, 3 # 관심 영역
13 print("[dif_img1(roi) uint8] = \n%s\n" % dif_img1[y:y+h, x:x+w]) # 관심 영역 원소값 출력
14 print("[dif_img2(roi) int16] = \n%s\n" % dif_img2[y:y+h, x:x+w])
15 print("[abs_dif1(roi)] = \n%s\n" % abs_dif1[y:y+h, x:x+w])
16 print("[abs_dif2(roi)] = \n%s\n" % abs_dif2[y:y+h, x:x+w])
17
18 titles = ['image1', 'image2', 'dif_img1', 'abs_dif1', 'abs_dif2'] # 윈도우 제목 리스트
19 for title in titles:
20     cv2.imshow(title, eval(title)) #
21 cv2.waitKey(0)
```

25

5.4 원소의 절댓값 연산

◆실행 결과

Run: 09.mat_abs.py

C:\Python\python.exe D:/source/chap05/09.mat_abs.py

[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]]
```

← 차분 영상의 관심영역 일부

[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]]
```

← int16형으로 변환 후 계산

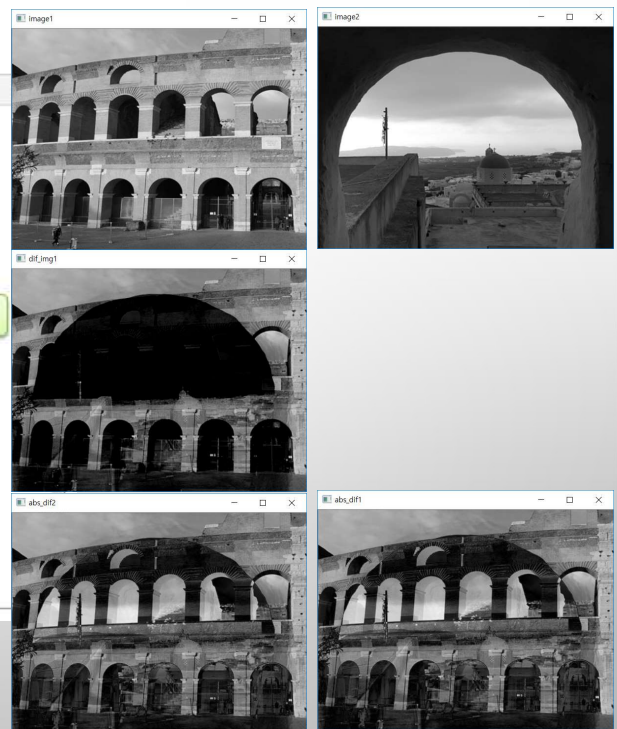
[abs_dif1(roi)] =

```
[[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]]
```

← 차분 영상의 절댓값 계산 결과



26

5.4.1 원소의 최솟값과 최댓값

함수 설명		
cv2.min(src1, src2[, dst]) → dst		
<ul style="list-style-type: none"> ■ 설명: 두 입력 배열의 원소 간 비교하여 작은 값을 출력 배열로 반환한다. ■ 수식: $dst(i) = \min(src1(i), src2(i))$ 		
인수	■ src1, src2	두 개의 입력 배열
설명	■ dst	계산 결과 출력 배열
cv2.max(src1, src2[, dst]) → dst		
<ul style="list-style-type: none"> ■ 설명: 두 입력 배열의 원소 간 비교하여 큰 값을 배열로 반환한다. ■ 수식: $dst(i) = \max(src1(i), src2(i))$ 		
cv2.minMaxLoc(src[, mask]) → minVal, maxVal, minLoc, maxLoc		
<ul style="list-style-type: none"> ■ 설명: 입력 배열에서 최솟값과 최댓값, 최솟값과 최댓값을 갖는 원소 위치를 반환한다. 		
인수 설명	■ src	입력 배열
	■ minVal, maxVal	최솟값, 최댓값
	■ minLoc, maxLoc	최솟값, 최댓값을 갖는 원소 위치(정수형 튜플)

27

5.4.1 원소의 최솟값과 최댓값

예제 5.4.2 행렬 최솟값 및 최대값 연산 - 10.mat_min_max.py

```

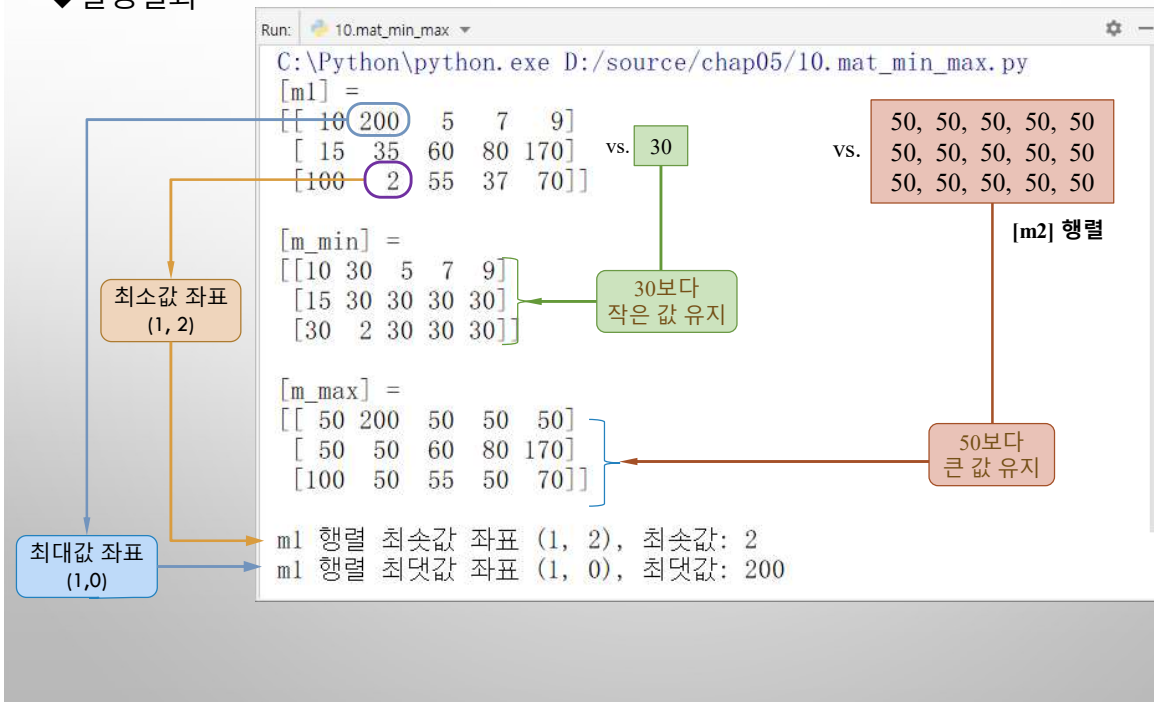
01 import numpy as np, cv2
02
03 data = [ 10, 200, 5, 7, 9,           # 1차원 리스트 생성
04          15, 35, 60, 80, 170,
05          100, 2, 55, 37, 70 ]
06 m1 = np.reshape(data, (3, 5))       # 리스트 행태 변환하여 2차원 행렬 생성
07 m2 = np.full((3, 5), 50)            # 원소값 50인 2차원 행렬 생성
08
09 m_min = cv2.min(m1, 30)              # 행렬 원소와 스칼라 간 최솟값을 행렬로 저장
10 m_max = cv2.max(m1, m2)              # 두 행렬 원소간 최댓값 계산
11
12 ## 행렬의 최솟값/최댓값과 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 최솟값 좌표%s, 최솟값: %d" % (min_loc, min_val) )
21 print("m1 행렬 최댓값 좌표%s, 최댓값: %d" % (max_loc, max_val) )

```

28

5.4.1 원소의 최솟값과 최댓값

◆ 실행결과



29

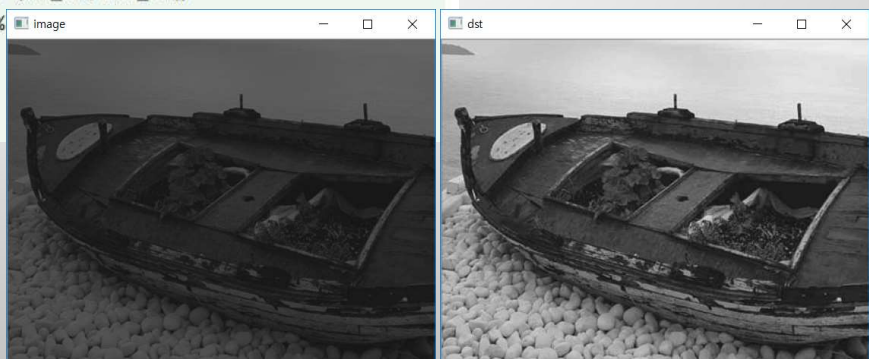
심화예제 5.4.3 영상 최솟값 최대값 연산 - 11.image_min_max.py

5.4.1 원소의 최솟값과 최댓값

```

01 import numpy as np, cv2
02
03 image = cv2.imread("images/minMax.jpg", cv2.IMREAD_GRAYSCALE)
04 if image is None: raise Exception("영상파일 읽기 오류 발생")
05
06 min_val, max_val, _, _ = cv2.minMaxLoc(image) # 최솟값과 최댓값 가져오기
07
08 ratio = 255 / (max_val - min_val)
09 dst = np.round((image - min_val) * ratio).astype('uint8')
10 min_dst, max_dst, _, _ = cv2.minMaxLoc(dst)
11
12 print("원본 영상 최솟값= %d, 최댓값= %d" % (min_val, max_val))
13 print("수정 영상 최솟값= %d, 최댓값= %d" % (min_dst, max_dst))
14 cv2.imshow('image', image)
15 cv2.imshow('dst', dst)
16 cv2.waitKey(0)
    
```

Run: 11.image_min_max
C:\Python\python.exe D:/source/chap05/11.image_min_max.py
원본 영상 최솟값= 13, 최댓값= 107
수정 영상 최솟값= 0, 최댓값= 255



30

cv2.sumElems(src) → retval

■ 설명: 배열의 각 채널별로 원소들의 합 N 을 계산하여 스칼라값으로 반환한다.

■ 수식: $S = \sum_i src(i)$

인수
설명 ■ src 1개에서 4개 채널을 갖는 입력 배열

cv2.mean(src[, mask]) → retval

■ 설명: 배열의 각 채널별로 원소들의 평균을 계산하여 스칼라값으로 반환한다.

■ 수식: $N = \sum_{i:mask(i) \neq 0} 1$
 $M_c = \left(\sum_{i:mask(i) \neq 0} src(i) \right) / N$

인수
설명 ■ src 1개에서 4개 채널을 갖는 입력 배열
 ■ mask 연산 마스크 - 마스크가 0이 아닌 좌표만 연산 수행

cv2.meanStdDev(src[, mean[, stddev[, mask]]]) → mean, stddev

■ 설명: 배열 원소들의 평균과 표준편차를 계산한다.

인수
설명 ■ src 1개에서 4개 채널을 갖는 입력 배열
 ■ mean 계산된 평균이 반환되는 출력 인수, np.float64형으로 반환
 ■ stddev 계산된 표준편차가 반환되는 출력 인수, np.float64형으로 반환
 ■ mask 연산 마스크 - 마스크가 0이 아닌 좌표만 연산 수행

cv2.countNonZero(src) → retval

■ 설명: 0이 아닌 배열 원소를 개수 N 을 반환한다.

■ 수식: $N = \sum_{i:src(i) \neq 0} 1$

31

cv2.reduce(src, dim, rtype[, dst[, dtype]]) → dst

■ 설명: 행렬을 열방향/행방향으로 옵션 상수(rtype)에 따라 축소한다.

인수
설명 ■ src 2차원 입력 배열 (np.float32, np.float64형만 수행 가능)
 ■ dst 출력 벡터, 감소방향과 타입은 dim, dtype 인수에 따라 정해짐
 ■ dim 행렬이 축소될 때 차원 감소 첨자
 - 0 : 열 방향으로 연산하여 1행으로 축소
 - 1 : 행 방향으로 연산하여 1열로 축소
 ■ rtype 축소 연산 종류

옵션 상수	값	설명
cv2.REDUCE_SUM	0	행렬의 모든 행(열)들을 합한다.
cv2.REDUCE_AVG	1	행렬의 모든 행(열)들을 평균한다.
cv2.REDUCE_MAX	3	행렬의 모든 행(열)들의 최댓값을 구한다.
cv2.REDUCE_MIN	4	행렬의 모든 행(열)들의 최솟값을 구한다.

■ dtype 감소된 벡터의 자료형

cv2.sort(src, flags[, dst]) → dst

■ 설명: 행렬의 각 행 혹은 각 열의 방향으로 정렬한다.

인수
설명 ■ src 단일채널 입력 배열
 ■ dst 정렬된 출력 배열
 ■ flags 연산 플래그 - 다음의 상수를 조합해서 정렬 방식 구성

옵션 상수	값	설명
cv2.SORT_EVERY_ROW	0	각 행을 독립적으로 정렬
cv2.SORT_EVERY_COLUMN	1	각 열을 독립적으로 정렬
cv2.SORT_ASCENDING	0	오름차순으로 정렬
cv2.SORT_DESCENDING	16	내림차순으로 정렬

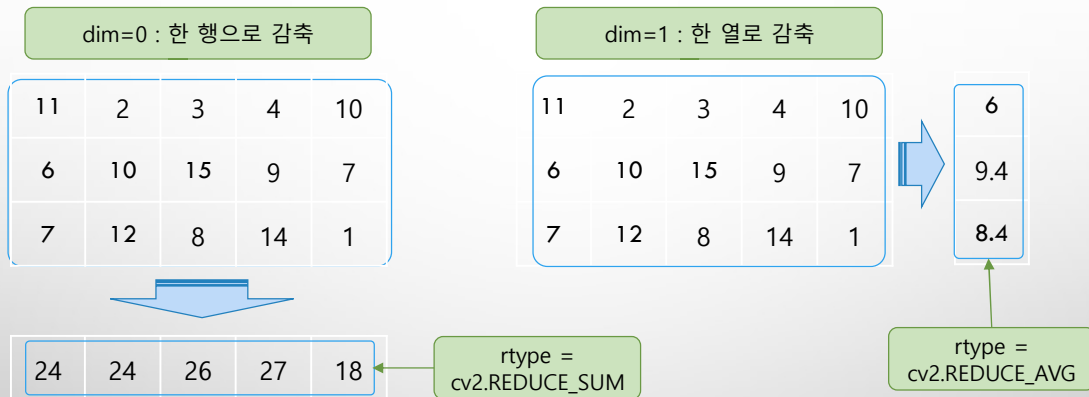
cv2.sortIdx(src, flags[, dst]) → dst

■ 설명: 행렬의 각 행 혹은 각 열로 정렬한다. 출력 배열(dst)에 정렬된 원소의 첨자들을 저장한다. 인수는 cv2.sort()와 동일하다.

32

5.5 통계 관련 함수

◆reduce() 함수 감축 방향



33

5.5 통계 관련 함수

◆reduce() 함수 예제

예제 5.5.5 행렬 축소 연산 - 17.mat_reduce.py

```

01 import numpy as np, cv2
02
03 m = np.random.rand(3, 5) * 1000//10          # 0~99 사이 실수(float)생성
04
05 reduce_sum = cv2.reduce(m, dim=0, rtype=cv2.REDUCE_SUM)   # 0 - 열방향 축소
06 reduce_avg = cv2.reduce(m, dim=1, rtype=cv2.REDUCE_AVG)  # 1 - 행방향 축소
07 reduce_max = cv2.reduce(m, dim=0, rtype=cv2.REDUCE_MAX)
08 reduce_min = cv2.reduce(m, dim=1, rtype=cv2.REDUCE_MIN)
09
10 print("[m1] = \n%s\n" % m1)
11 print("[m_reduce_sum] =", reduce_sum.flatten())
12 print("[m_reduce_avg] =", reduce_avg.flatten())
13 print("[m_reduce_max] =", reduce_max.flatten())
14 print("[m_reduce_min] =", reduce_min.flatten())

```

```

Run: 16.mat_reduce
C:\Python\python.exe D:/source/chap05/16.mat_reduce.py
[m1] =
[[41. 16. 59. 33. 38.]
 [34. 59. 23. 32. 64.]
 [14. 79. 86. 76. 84.]]

[m_reduce_sum] = [ 89. 154. 168. 141. 186.]
[m_reduce_avg] = [37.4 42.4 67.8]
[m_reduce_max] = [41. 79. 86. 76. 84.]
[m_reduce_min] = [16. 23. 14.]

```

34

5.5 통계 관련 함수

예제 5.5.2 행렬 원소 정렬 - 13.sort.py

```

01 import numpy as np, cv2
02
03 m = np.random.randint(0, 100, 15).reshape(3,5)      # 임의 난수 생성
04 ## 행렬 원소 정렬
05 sort1 = cv2.sort(m, cv2.SORT_EVERY_ROW)            # 행단위(가로 방향) 오름차순
06 sort2 = cv2.sort(m, cv2.SORT_EVERY_COLUMN)         # 열단위(세로 방향) 오름차순
07 sort3 = cv2.sort(m, cv2.SORT_EVERY_ROW+cv2.SORT_DESCENDING) # 행단위 내림차순
08 sort4 = np.sort(m, axis=1)                          # x축 (가로 방향) 정렬
09 sort5 = np.sort(m, axis=0)                          # y축 (세로 방향) 정렬
10 sort6 = np.sort(m, axis=1)[::-1]                   # 열 방향 내림차순 정렬
11
12 titles= ['m', 'sort1', 'sort2', 'sort3', 'sort4', 'sort5', 'sort6']
13 for title in titles:
14     print("[{}] = \n{}\n" % (title, eval(title)))

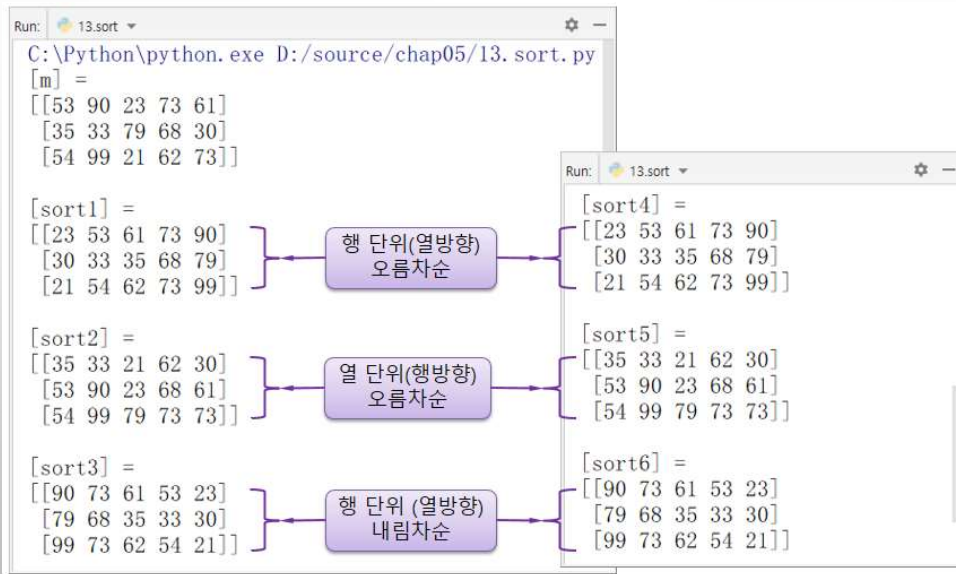
```



35

5.5 통계 관련 함수

◆ 실행결과



36

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

■ 설명: 일반화된 행렬 곱셈을 수행한다.

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

인수 설명	■ src1, src2	행렬 곱을 위한 두 입력 행렬(np.float32/np.float64형 2채널까지 가능)
	■ alpha	행렬 곱($src1^T \cdot src2$)에 대한 가중치
	■ src3	행렬 곱($src1^T \cdot src2$)에 더해지는 델타 행렬
	■ beta	src3 행렬에 곱해지는 가중치
	■ dst	출력 행렬
	■ flags	연산 플래그 - 옵션을 조합하여 입력 행렬들을 전치

옵션	값	설명
cv2.GEMM_1_T	1	src1을 전치
cv2.GEMM_2_T	2	src2을 전치
cv2.GEMM_3_T	4	src3을 전치

cv2.perspectiveTransform(src, m[, dst]) → dst

■ 설명: 입력 벡터들에 대해서 투영(perspective) 변환 m을 수행한다.

■ 수식: 3차원 좌표인 경우

$$(x, y, z) \rightarrow (x'/w, y'/w, z'/w)$$

여기서

$$(x', y', z', w') = m \cdot [x, y, z, 1] \quad w = \begin{cases} w' & \text{if } w' \neq 0 \\ \infty & \text{otherwise} \end{cases}$$

인수 설명	■ src	좌표로 변환될 2채널 혹은 3채널의 부동소수점 배열
	■ dst	src와 같은 크기와 타입의 출력 배열
	■ m	3×3 혹은 4×4 부동소수점의 투영 변환 행렬

37

5.6 행렬 연산 함수

예제 5.6.1 행렬곱(내적) 연산 - 18.gemm.py

```
01 import numpy as np, cv2
02
03 src1 = np.array([1, 2, 3, 1, 2, 3], np.float32).reshape(2, 3)    # 2×3 행렬 선언
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)    # 3×2 행렬 선언
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, src3, alpha, None, beta)
11
12 titles = ['src1', 'src1', 'src1', 'dst1', 'dst2', 'dst3']
13 for title in titles:
14     print("[{}s] = \n{}\n" % (title, eval(title)))
```

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

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

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

38

5.6 행렬 연산 함수

◆ 실행결과

```
Run: 17.mat_gemm
C:\Python\python.exe D:/source/chap05/17.mat_gemm.py
[src1] =
[[1. 2. 3.]
 [1. 2. 3.]]

[src1] =
[[1. 2. 3.]
 [1. 2. 3.]]

[src1] =
[[1. 2. 3.]
 [1. 2. 3.]]

[dst1] =
[[ 5.  7.  9.]
 [10. 14. 18.]
 [15. 21. 27.]]

[dst2] =
[[14. 32.]
 [14. 32.]]

[dst3] =
[[ 6. 12.]
 [ 6. 12.]]
```

39

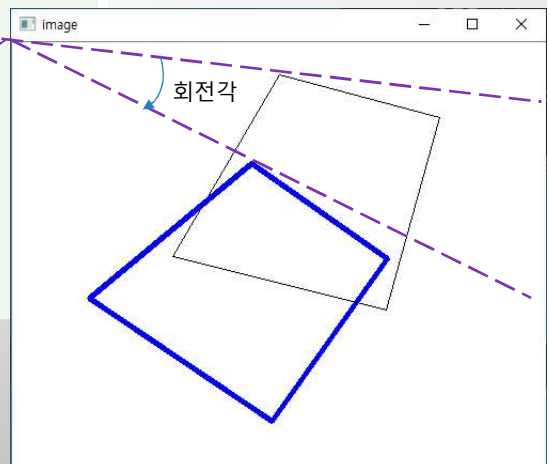
심화예제 5.6.2 cv2.gemm()을 이용한 회전 변환 - 19.point_transform.py

5.6 행렬 연산 함수

```
01 import numpy as np, cv2
02
03 theta = 20 * np.pi / 180 # 회전할 라디안 각도 계산
04 rot_mat = np.array([ [np.cos(theta), -np.sin(theta)],
05                      [np.sin(theta), np.cos(theta)]] , np.float32) # 회전 변환 행렬 생성
06
07 pts1 = np.array([ (250, 30), (400, 70),
08                  (350, 250), (150, 200)], np.float32) # 입력 좌표 지정
09 pts2 = cv2.gemm(pts1, rot_mat, 1, None, 1, flags=cv2.GEMM_2_T) # 행렬곱으로 회전변환
10
11 for i, (pt1, pt2) in enumerate(zip(pts1, pts2)):
12     print("pts1[%d] = %s, p2[%d]= %s" % (i, pt1, i, pt2))
13
14 ## 영상 생성 및 4개 좌표 그리기
15 image = np.full((400, 500, 3), 255, np.uint8)
16 cv2.polylines(image, [np.int32(pts1)], True, (0, 255, 0), 2)
17 cv2.polylines(image, [np.int32(pts2)], True, (255, 0, 0), 3)
18 cv2.imshow('image', image)
19 cv2.waitKey(0)
```

원점

회전각



40