Chapter

5

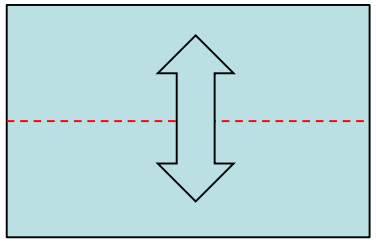
OpenCV 기본배열연산

1

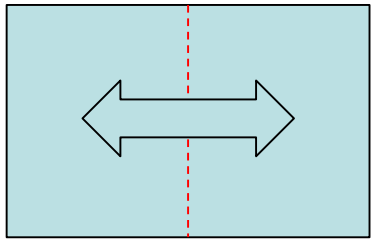
flip, repeat, transpose

flip

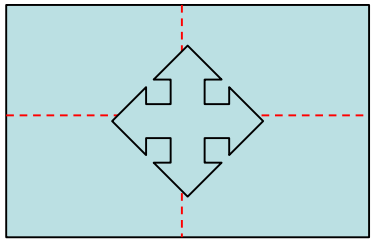
- **cv2.flip(src, flipCode, dst=None) -> dst**
  - src : 입력영상
  - filpCode : 대칭방향
    - 0 : x축을 기준으로 상하대칭
    - 1(양수) : y축을 기준으로 좌우대칭
    - -1(음수) : x, y축을 기준으로 상하, 좌우 모두 뒤집기
  - dst : 출력영상



0



1



-1

repeat / transpose

❖ **cv2.repeat(src, ny, nx[, dst]) -> dst**

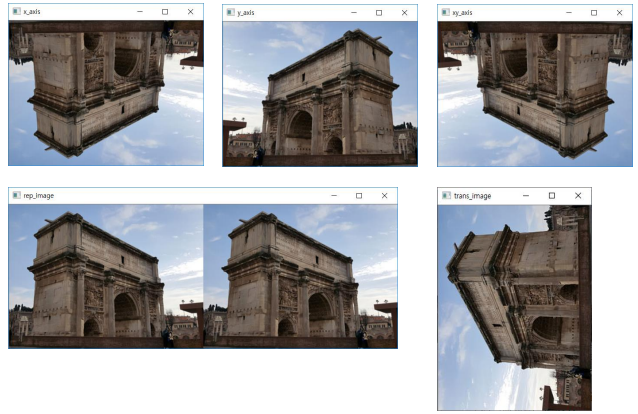
- src : 입력영상
- ny : y방향 반복회수(영상을 아래쪽 방향으로 반복적으로 복사)
- nx : x방향 반복회수(영상을 오른쪽 방향으로 반복적으로 복사)
- dst : 출력영상

❖ **cv2.transpose(src[, dst]) -> dst**

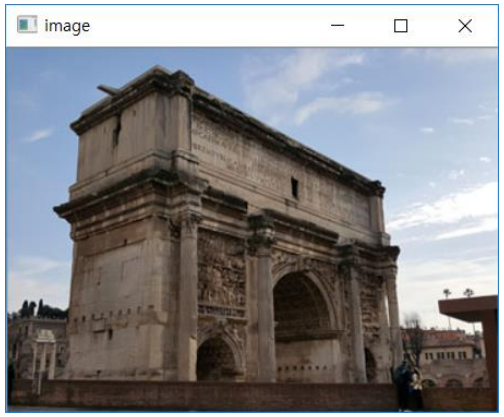
- 입력영상을 전치행렬로 변환하여 반환
- src : 입력영상
- dst : 출력영상

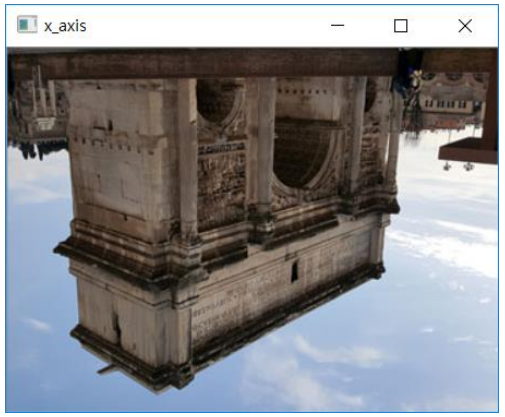
예제 5.1.1 행렬 처리 함수 – 01.mat_array.py

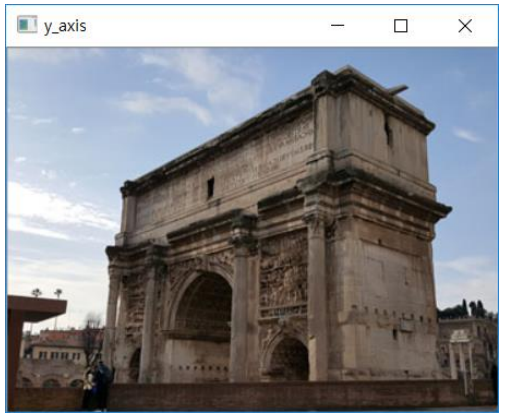
```
01  import cv2
02
03  image = cv2.imread("images/flip_test.jpg", cv2.IMREAD_COLOR)
04  if image is None: raise Exception("영상파일 읽기 오류 발생")        # 예외 처리
05
06  x_axis = cv2.flip(image, 0)                                      # x축 기준 상하 뒤집기
07  y_axis = cv2.flip(image, 1)                                      # y축 기준 좌우 뒤집기
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)
```



❖ 실행결과







2 채널(channel) 처리

split / merge

❖ **cv2.split(m[, mv]) -> mv**

- m : 분리될 다채널영상
- mv : 분리되어 반환될 단일채널영상배열들의 벡터

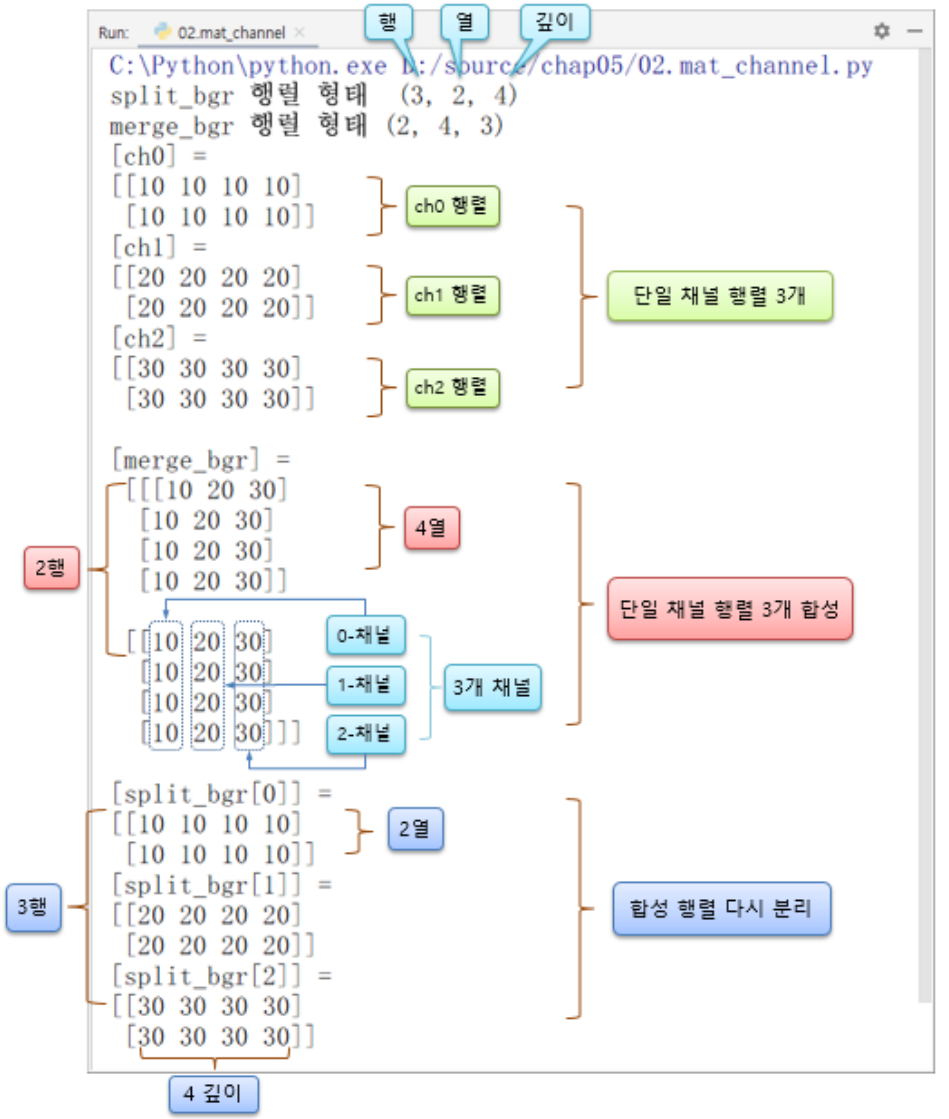
❖ **cv2.merge(mv[, dst]) -> dst**

- mv : 합성될 단일채널영상배열(size와 depth가 동일해야 함)
- dst : 출력영상

예제 5.2.1 채널 분리 및 합성 – 02.mat_channel.py

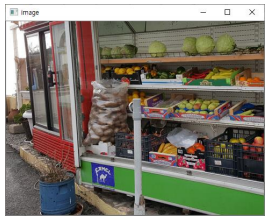
```
01  import numpy as np
02  import cv2
03
04  ## numpy.ndarray를 이용해 행렬 생성 및 초기화 방법
05  ch0 = np.zeros((2, 4), np.uint8) + 10          # 0 원소 행렬 선언 후 10 더하기
06  ch1 = np.ones((2, 4), np.uint8) * 20           # 1 원소 행렬 선언 후 20 곱하기
07  ch2 = np.full((2, 4), 30, np.uint8)            # 행렬을 생성하며 30으로 초기화
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
13
14  print("split_bgr 행렬 형태", np.array(split_bgr).shape)
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])
```

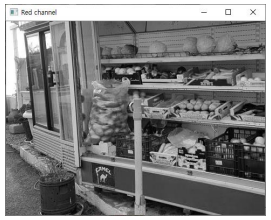
❖ 실행결과



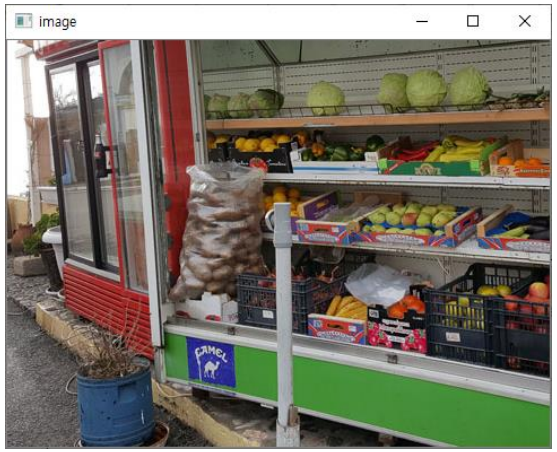
예제 5.2.2 컬러 채널 분리 – 03.image_channels.py

```
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)
```





❖ 실행결과

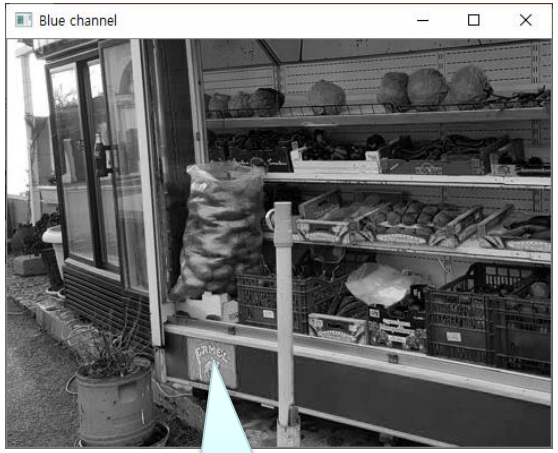



```
Run:  03.image_channels
C:\Python\python.exe D:/source/chap05/03.image_channels.py
bgr 자료형: <class 'list'> <class 'numpy.ndarray'> <class 'numpy.uint8'>
bgr 원소개수: 3
```

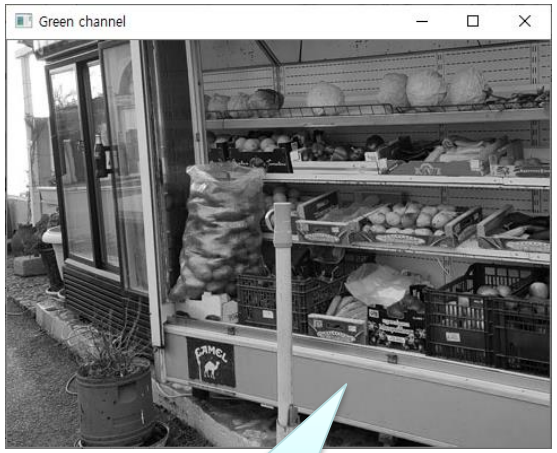
bgr 자료형

bgr 원소(단일채널) 자료형

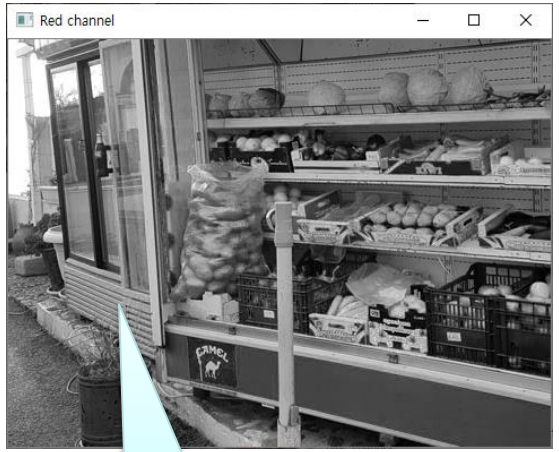
단일채널의 원소 자료형



파란색 - 밝은색



녹색 - 밝은색



붉은색 - 밝은색

참고:

❖ numpy를 이용한 채널분리

- 매우 효율적임
  - `b = img[:, :, 0]`
  - `g = img[:, :, 1]`
  - `r = img[:, :, 2]`

❖ 채널의 합성

- cv2.split()함수와 반대로 cv2.merge()함수 사용
- (예)
  - `mergedimg = cv2.merge((b, g, r))`

3

산술연산, 논리연산 등

사칙연산

- **cv2.add(src1, src2[, dst[, mask[, dtype]]]) ->dst**
  - src1, src2 : 입력영상, dst : 출력영상
  - mask : 연산 마스크(마스크 요소 중 0이 아닌것만 연산수행)
    - 마스크는 8bit 단일채널이어야 함
  - dtype : 출력영상의 depth
- **cv2.subtract(src1, src2[, dst[, mask[, dtype]]]) ->dst**
- **cv2.multiply(src1, src2[, dst[, scale[, dtype]]]) ->dst**
  - scale : 추가로 곱해주는 배율
- **cv2.divide(src1, src2[, dst[, scale[, dtype]]]) -> dst**
- **cv2.divide(scale, src2[, dst[, dtype]]) ->dst**
- **cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]]) ->dst**
  - alpha : src1의 가중치, beta : src2의 가중치, gamma : 추가로 더해주는 scalar값

❖ 산술연산시 고려사항

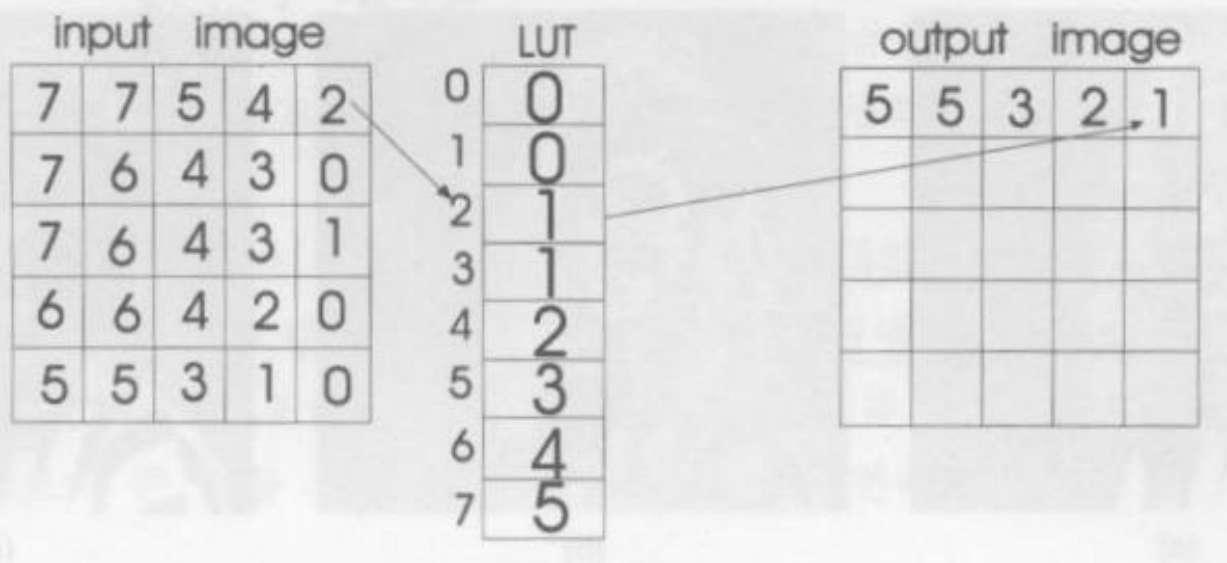
- clamping
- lookup table

❖ Clamping(satutate 연산)

- 입출력영상의 depth는 둘 다 0~255사이의 값
- 값을 더하거나 조작하여 결과값이 0~255사이의 범위를 벗어나는 경우 방지 필요
  - `OutImg[x][y] = (OutImg[x][y]>255) ? 255 : OutImg[x][y];`
  - `OutImg[x][y] = (OutImg[x][y]<  0) ?   0 : OutImg[x][y];`

❖ Lookup Table(처리속도 개선)

- 룩업 테이블(Lookup Table: LUT) 연산
  - 산술연산의 고속수행을 위해 사용
  - LUT 배열을 미리 생성함에 의해 반복 산술연산의 회피가 가능함
  - 포인트처리 기법에서 가장 효과적으로 사용됨



3비트 LUT 를 이용한 연산

예제 5.3.1 행렬 산술 연산 – 04.arithmethic_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, 1을 할당
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 형변환
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] = \n%s \n" % (title, eval(title)))
```

```
Run: 04.arithmethic_op
C:\Python\python.exe D:/source/chap05/04.arithmethic_op.py
[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]]
```

지수, 로그, 제곱근 관련연산

- **cv2.exp(src[, dst]) -> dst**
- **cv2.log(src[, dst]) -> dst**
- **cv2.sqrt(src[, dst]) -> dst**
- **cv2.pow(src, power[, dst]) -> dst**
- **좌표계변환 관련함수**
  - cv2.magnitude(x, y[, magnitude]) -> magnitude
    - 크기계산: $magnitude(i) = \sqrt{x(i)^2 + y(i)^2}$
  - cv2.phase(x, y[, angle[, angleInDegrees]]) -> angle
    - 회전각도계산: $angle(i) = atan2(y(i), x(i)) \cdot (\frac{180}{\pi})$
    - angleInDegrees(**True**: degree, **False**: radian)
  - cv2.cartToPolar(x, y[, magnitude[, angle[, angleInDegrees]]]) -> magnitude, angle
  - cv2.polarToCart(magnitude, angle[, x[, y[, angleInDegrees]]]) -> x, y

예제 5.3.2 행렬 지수 및 로그 연산 – 05.exp_log.py

```
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 가능함
09  v_exp = cv2.exp(v1)                                     # 1차원 행렬에 대한 지수
10  m_exp = cv2.exp(v2)                                     # 행벡터(1×3)에 대한 지수 계산
11  m_exp = cv2.exp(v3)                                     # 열벡터(3×1)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
```

```
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.]
```

3행,1열 → 열벡터

3열 → 1차원 행렬

1행, 3열 → 행벡터

OpenCV 함수에서 행렬로 반환하는 것은 대부분 ndarray 객체임

2차원 행렬로 반환

논리(비트) 연산

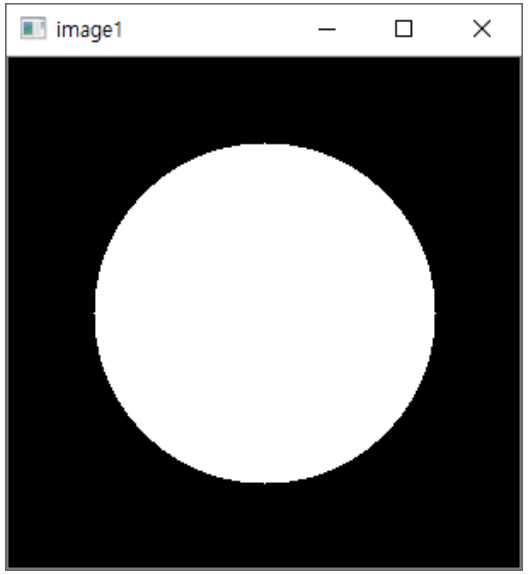
- **cv2.bitwise_and(src1, src2[, dst[, mask]]) -> dst**
  - src1, src2 : 입력영상
    - src1과 src2중 하나는 scalar값 가능
  - dst : 출력영상
  - mask : 마스크의 원소가 0이 아닌 것만 연산
- **cv2.bitwise_or(src1, src2[, dst[, mask]]) -> dst**
- **cv2.bitwise_xor(src1, src2[, dst[, mask]]) -> dst**
- **cv2.bitwise_not(src[, dst[, mask]]) -> dst**

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                                  # 중심 좌표
08  cv2.circle(image1, (cx, cy), 100, 255, -1)           # 중심에 원 그리기
09  cv2.rectangle(image2, (0, 0, cx, h), 255, -1)        # 영상의 가로 절반
10
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)
```

내부 채움

❖ 실행결과



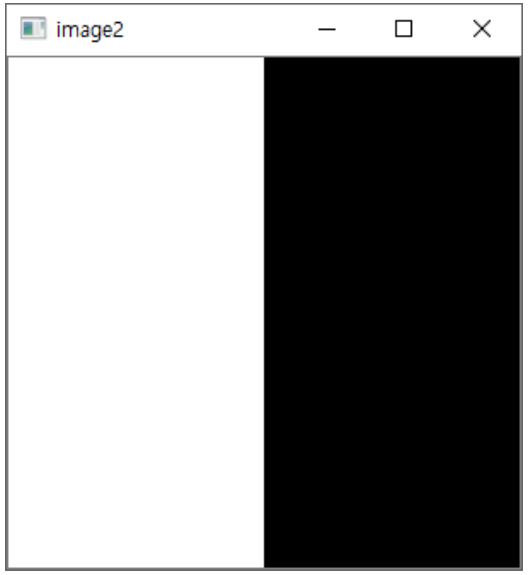


image1 **or** image2

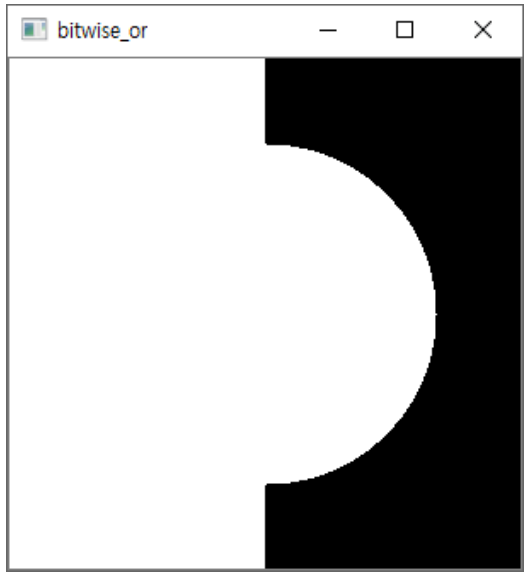
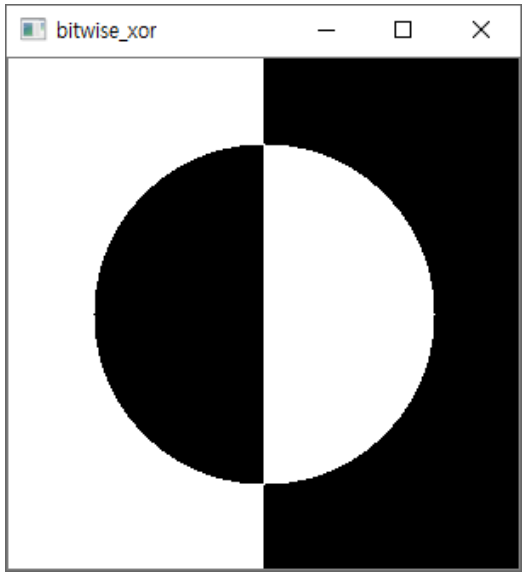
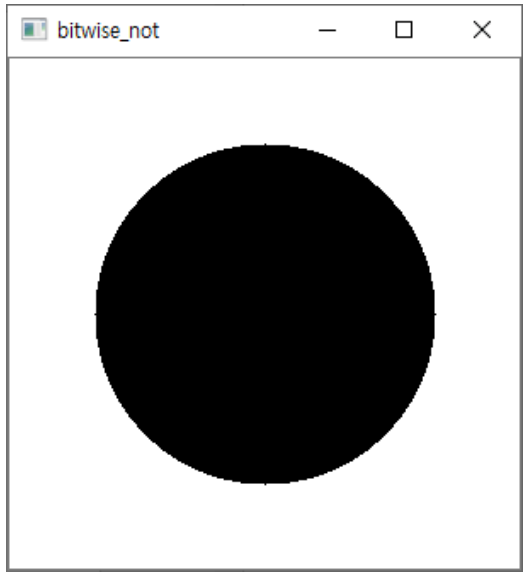


image1 **and** image2

image1 **xor** image2



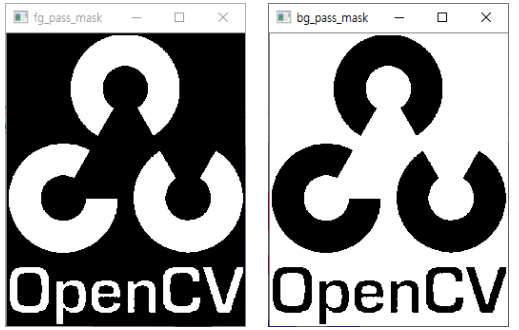
not image1

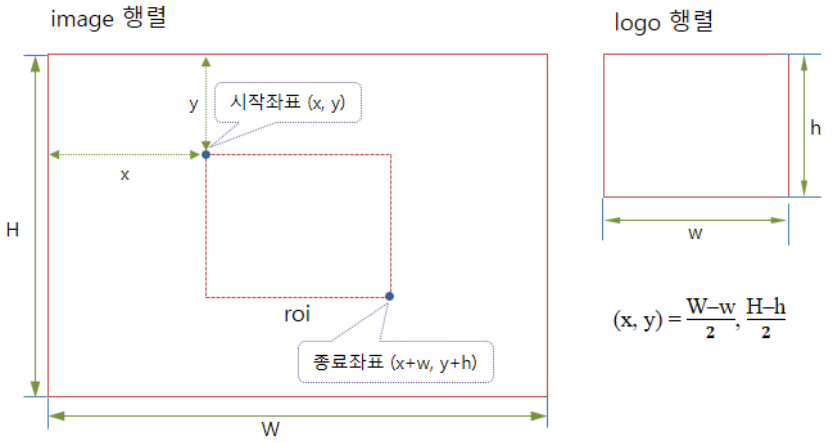


심화예제 5.3.4 행렬 비트 연산2 – 08.bitwise_overlap.py

```
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, 로고 영상 크기
15  x, y = (W-w)//2, (H-h)//2                              # 시작 좌표 계산
16  roi = image[y:y+h, x:x+w]                              # 관심 영역(roi) 지정
17
18  ## 행렬 논리곱과 마스킹을 이용한 관심 영역 복사
19  foreground = cv2.bitwise_and(logo, logo, mask=fg_pass_mask)     # 로고의 전경만 복사
20  background = cv2.bitwise_and(roi , roi , mask=bg_pass_mask)     # 원본 roi의 배경만 복사
21
22  dst = cv2.add(background, foreground)                  # 로고 전경과 원본 배경 간 합성
23  image[y:y+h, x:x+w] = dst                              # 합성 영상을 원본에 복사
24
25  cv2.imshow('background', background); cv2.imshow('forground', foreground)
26  cv2.imshow('dst', dst);               cv2.imshow('image', image)
27  cv2.waitKey()
```

전경& 배경 마스크

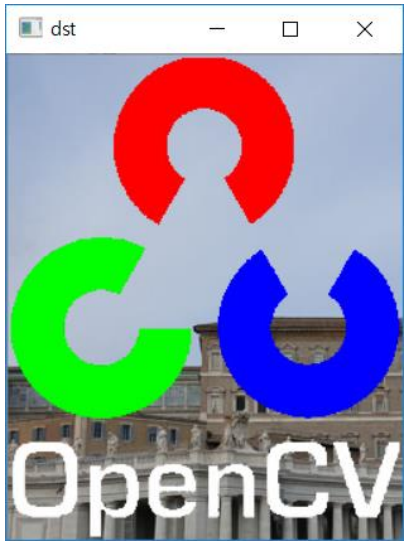




❖ 실행결과


background
OpenCV


forground
OpenCV


dst
OpenCV

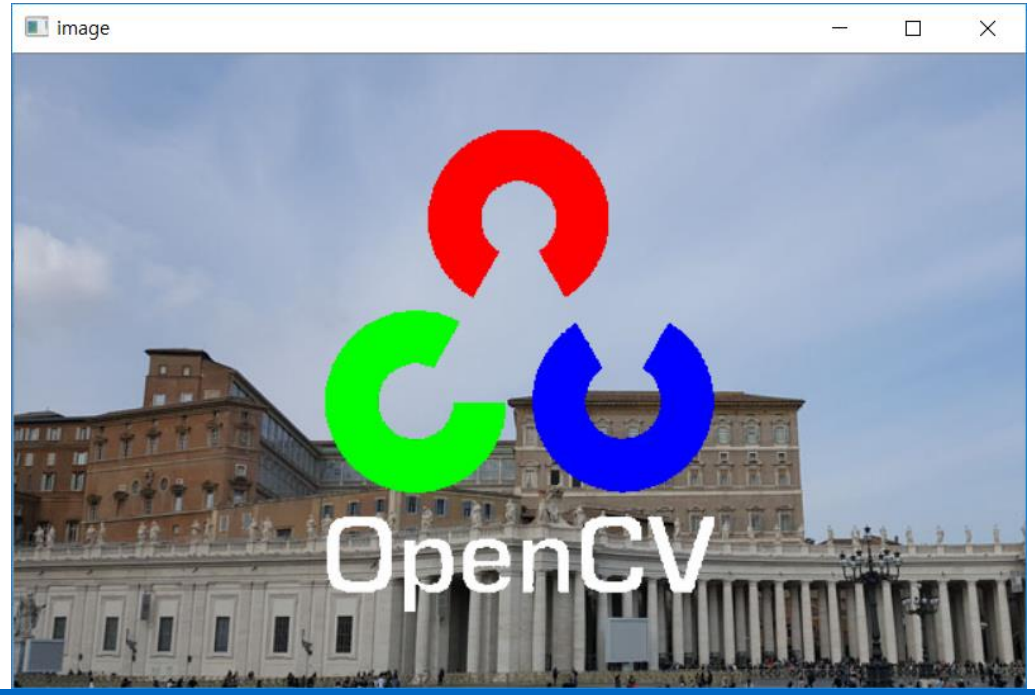


image
OpenCV

절대값, 최대 최소 연산

❖ cv2.absdiff(src1, src2[, dst]) -> dst

- 두 입력영상간의 차이에 대한 절대값을 계산
- src1, src2 : 입력영상
  - src1과 src2중 하나는 scalar값 가능
- dst : 출력영상

❖ cv2.convertScaleAbs(src[, dst[, alpha[, beta]]]) -> dst

- $dst(i) = saturate(|src(i) * \alpha + \beta|)$
- src : 입력영상
- dst : 출력영상(depth 8bit)
- alpha, beta : 곱해지는 스케일 팩터와 더해지는 델타값

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

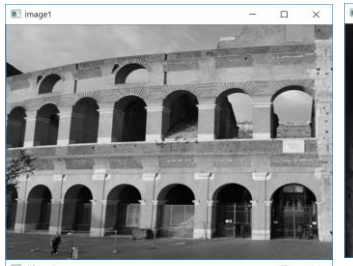
```
01  import numpy as np, cv2
02
03  image1 = cv2.imread("images/abs_test1.jpg", cv2.IMREAD_GRAYSCALE) # 명암도 영상 읽기
04  image2 = cv2.imread("images/abs_test2.jpg", cv2.IMREAD_GRAYSCALE)
05  if image1 is None or image2 is None: raise Exception("영상파일 읽기 오류")
06
07  dif_img1 = cv2.subtract(image1, image2)                    # 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)
```

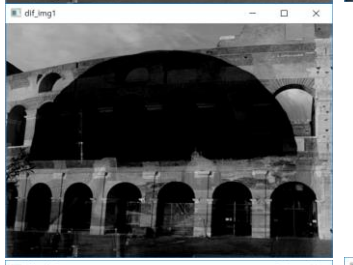
```
Run: 09.mat_abs
C:\Python\python.exe D:/source/chap05/09.mat_abs.py
[dif_img1(roi) uint8] =
[[ 0  0  0  0  9 12  7]
 [ 0  0  0  0  4  9  3]
 [ 0  0  0 15  0  4  0]]

[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]]
```

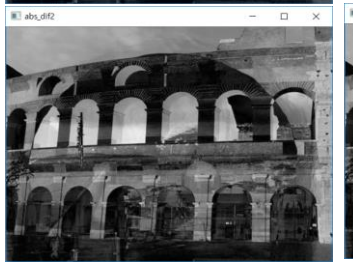
차분 영상의 관심영역 일부

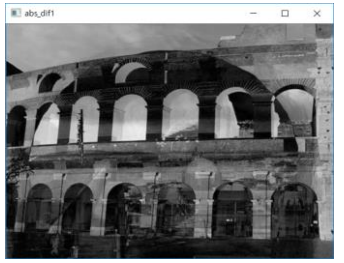
int16형으로 변환 후 계산

차분 영상의 절대값 계산 결과









최소값과 최대값

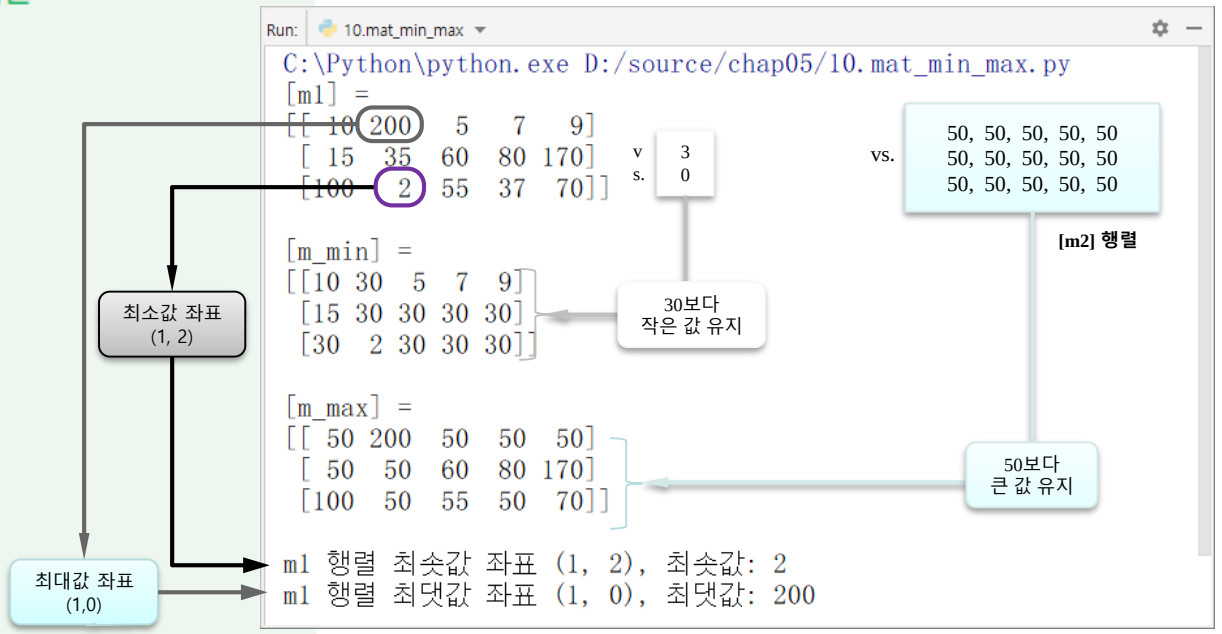
- **cv2.min(src1, src2[, dst]) -> dst**
  - src1, src2 : 입력영상
  - 출력영상: $dst(i) = \min(src1(i), src2(i))$

- **cv2.max(src1, src2[, dst]) -> dst**
  - src1, src2 : 입력영상
  - 출력영상: $dst(i) = \max(src1(i), src2(i))$

- **cv2.minMaxLoc(src[, mask]) -> minVal, maxVal, minLoc, maxLoc**
  - src : 입력영상
  - mask : 마스크(요소가 0이 아닌것만 연산)
  - minVal, maxVal : 최소값/최대값 배열
  - minLoc, maxLoc : 최소값/최대값을 갖는 원소의 위치(int형 tuple)를 갖는 배열

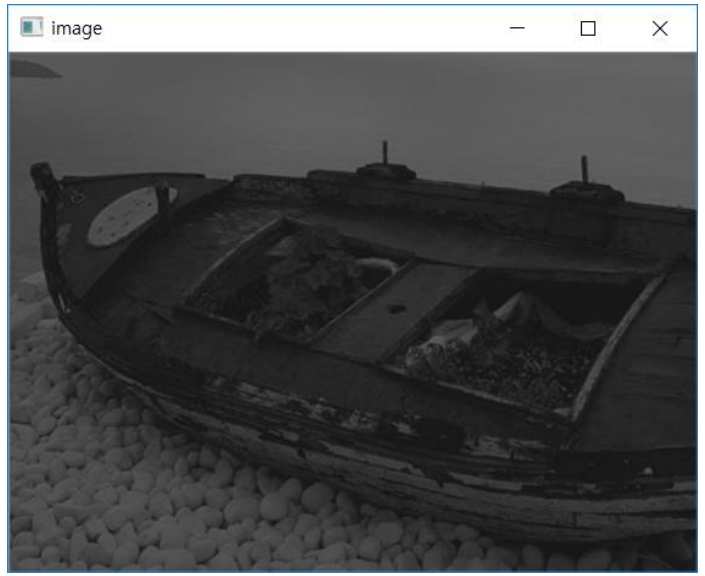
예제 5.4.2 행렬 최소값 및 최대값 연산 – 10.mat_min_max.py

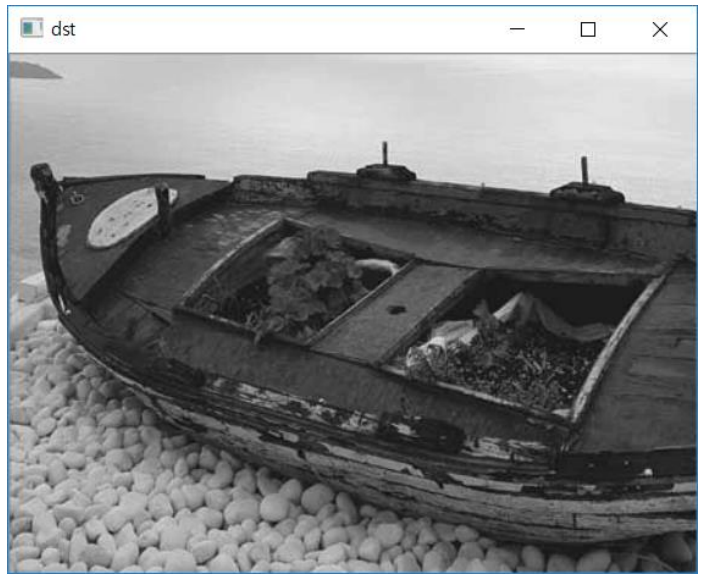
```
01  import numpy as np, cv2
02
03  data = [  10, 200, 5, 7, 9,                    # 1차원 리스트 생성
04            15, 35, 60, 80, 170,
05            100, 2, 55, 37, 70 ]
06  m1 = np.reshape(data, (3, 5))                  # 리스트 행태 변환하여 2차원 행렬 생성
07  m2 = np.full((3, 5), 50)                       # 원소값 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_loc와max_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) )
```



심화예제 5.4.3 영상 최소값 최대값 연산 – 11.image_min_max.py

```
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
```

4

통계연산, 행렬연산 등

통계관련 연산

- **cv2.sumElems(src) -> retval**
- **cv2.mean(src[, mask]) -> retval**
- **cv2.meanStdDev(src[, mean[, stddev[, mask]]]) -> mean, stddev**
- **cv2.countNonZero(src) -> retval**
- **cv2.reduce(src, dim, rtype[, dst[, dtype]]) -> dst**
  - src : 입력영상(depth가 np.float32, np.float64인 것만 가능)
  - dim : reduce의 방향(0: 열방향, 1: 행방향)
  - rtype : reduce 연산의 종류
    - cv2.REDUCE_SUM, cv2.REDUCE_AVG, cv2.REDUCE_MIN, cv2.REDUCE_MAX,

- **참고만하세요....**
  - cv2.**sort**(src, flags[, dst]) -> dst
  - cv2.**sortIdx**(src, flags[, dst]) -> dst

예제 5.5.1 행렬 합/평균 연산 – 12.sum_avg.py

```
01  import numpy as np, cv2
02
03  image = cv2.imread("images/sum_test.jpg", cv2.IMREAD_COLOR)
04  if image is None: raise Exception("영상파일 읽기 오류 발생")
05
06  mask = np.zeros(image.shape[:2], np.uint8)
07  mask[60:160, 20:120] = 255                          # 관심 영역에 값(255) 할당
08
09  sum_value    = cv2.sumElems(image)                  # 채널별 합 – 튜플
10  mean_value1 = cv2.mean(image)                       # 채널별 평균– 튜
11  mean_value2 = cv2.mean(image, mask)
12
13  print("sum_value 자료형:", type(sum_value), type(sum_value[0]))    # 결과
14  print("[sum_value] =", sum_value)
15  print("[mean_value1] =", mean_value1)
16  print("[mean_value2] =", mean_value2)
```

```
Run: 12.sum_avg
C:\Python\python.exe D:/source/chap05/12.sum_avg.p
sum_value 자료형: <class 'tuple'> <class 'float'>
[sum_value] =  (15865577.0, 15880547.0, 16470875.0, 0.0)
[mean_value1] =  (132.21314166666667, 132.33789166666668, 137.25729166666667, 0.0)
[mean_value2] =  (80.26520000000001, 81.59740000000001, 90.3211, 0.0)

mean 자료형: <class 'numpy.ndarray'> <class 'numpy.float64'>
[mean] =  [132.21314167 132.33789167 137.25729167]
[stddev] =  [73.35044328 68.76754506 63.96477788]

[mean2] =  [80.2652 81.5974 90.3211]
[stddev2] =  [58.91488326 57.57273064 54.0648388 ]
```

4 원소 튜플 반환 , 마지막 원소 0

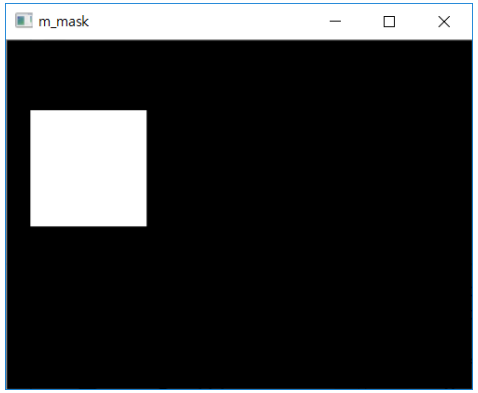
전체 영역 평균

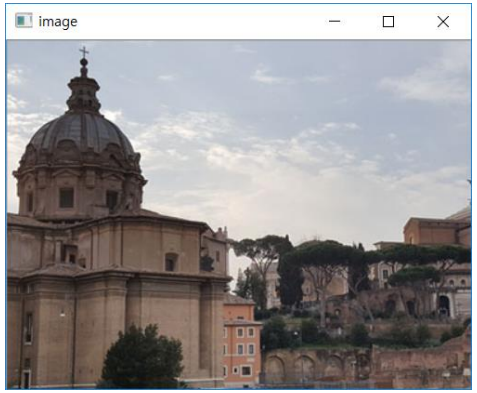
관심 영역 평균

전체영역 평균, 표준편차

관심영역 평균, 표준편차

```
17  print()
18
19  ## 평균과 표준편차 결과 저장
20  mean, stddev = cv2.meanStdDev(image)                  # 2 원소 튜플로 반환
21  mean2, stddev2 = cv2.meanStdDev(image, mask=mask)     # 마스크가 255인 영역만 계산
22  print("mean 자료형:", type(mean), type(mean[0][0]))    # 반환 행렬 자료형, 원소 자료형
23  print("[mean] =",  mean.flatten())                    # 벡터 변환 후 출력
24  print("[stddev] =", stddev.flatten())
25  print()
26
27  print("[mean2] =",  mean2.flatten())
28  print("[stddev2] =", stddev2.flatten())
29
30  cv2.imshow('image', image)
31  cv2.imshow('mask', mask)
32  cv2.waitKey(0)
```





❖ cv2.reduce(src, dim, rtype[, dst[, dtype]]) -> dst

- src : 입력영상(depth가 np.float32, np.float64인 것만 가능)
- dim : reduce의 방향(0: 열방향, 1: 행방향)
- rtype : reduce 연산의 종류

종류	값	설명
cv2.REDUCE_SUM	0	행렬의 모든 행(열)의 합을 구한다.
cv2.REDUCE_AVG	1	행렬의 모든 행(열)의 평균을 구한다.
cv2.REDUCE_MAX	3	행렬의 모든 행(열)의 최대값을 구한다.
cv2.REDUCE_MIN	4	행렬의 모든 행(열)의 최소값을 구한다.

- dst : 출력영상
- dtype : depth

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())            # 1차원 벡터로 출력
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.]
```

행렬변환 연산

❖ **cv2.gemm(src1, src2, alpha, src3, beta[, dst[, flags]]) -> dst**

- src1, src2 : 입력영상
- alpha : 두 입력영상의 행렬곱에 대한 가중치
- src3 : 주 입력영상의 행렬곱에 더해지는 델타행렬
- beta : src3에 곱해지는 가중치
- dst : 출력영상
- flags : 입력행렬들의 전치연산 옵션
  - **cv2.GEMM_1_T** : src1을 전치
  - **cv2.GEMM_2_T** : src2를 전치
  - **cv2.GEMM_3_T** : src3를 전치

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%s\n" % (title, eval(title)))
```

```
Run:  17.mat_gemm
C:\Python\python.exe D:/source/chap05/
[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.]]
```

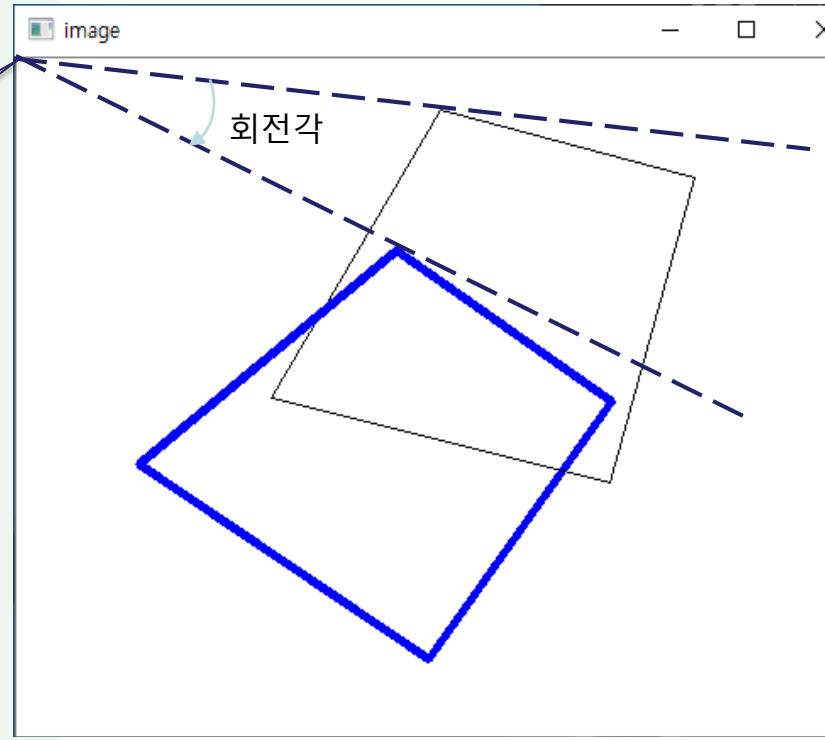
$$src1^T \cdot src2 = \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix}^T \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}1 & 1\\ 2 & 2\\ 3 & 3\end{bmatrix} \cdot \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} = \begin{bmatrix}5 & 7 & 9\\ 10 & 14 & 18\\ 15 & 21 & 27\end{bmatrix}$$

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

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

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

```
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, pst2[%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)
```



❖ cv2.perspectiveTransform(src, m[, dst]) -> dst

- 입력영상(src)에 대한 투영변환(m)을 실행
- src : 입력영상
- m : 3x3 또는 4x4 부동소수점의 투영변환 행렬

- 참고자료 : https://minimin2.tistory.com/135