Open CV 배열 연산

Open CV 배열 처리

❖ 영상 속성 과 픽셀 접근

- ✓ OpenCV 파이썬은 영상을 numpy.ndarray을 이용하여 표현
- ✓ 영상의 중요 속성(shape, dtype)을 확인하고 numpy의 astype(), reshape()로 속성을 변경하고 영상 픽셀를 y(행), x(열) 순으로 인덱스를 지정한 후 접근
- ✓ numpy는 다중 채널 영상을 모양(shape)에 의해 표현하고 OpenCV는 픽셀 자료형으로 표현

구분	numpy 자료형	OpenCV 자료형, 1-채널
8비트 unsigned 정수	np.uint8	cv2.CV_8U
8비트 signed 정수	np.int8	cv2.CV_8S
16비트 unsigned 정수	np.uint16	cv2.CV_16U
16비트 signed 정수	np.int16	cv2.CV_16S
32비트 signed 정수	np.int32	cv2.CV_32S
32비트 실수	np.float32	cv2.CV_32F
64비트 실수	np.float64	cv2.CV_64F

Open CV 배열 처리

- ❖ 영상 속성 과 픽셀 접근
  - ✓ 이미지 크기 변경
    - resize 함수를 이용해서 이미지 크기 변경

      resize(이미지 데이터, (가로 크기, 세로 크기))
    - 이미지들은 제각기 다양한 크기를 가지는데 이를 특성으로 사용하려면 동일한 차원으로 만들어야 하기 때문에 이미지 크기를 변경하는데 이미지는 행렬에 정보를 담고 있기 때문에 이미지 크기를 줄이면 행렬 크기와 거기에 담긴 정보도 줄어 듬
    - 머신 러닝은 수 천 에서 수 십만 개의 이미지가 필요한데 이미지가 클수록 메모리를 많이 차지하기 때문에 이미지 크기를 줄여서 메모리 사용량을 줄일 수 있음
      - ◆ 머신 러닝에서 많이 사용하는 이미지의 크기는 32X32, 64X64, 96X96, 256X256 등

Open CV 배열 처리

❖ 영상 속성 과 픽셀 접근

✓ 이미지 크기 변경

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
print('img.shape=', img.shape)

##img = img.reshape(img.shape[0]*img.shape[1])
img = img.flatten()
print('img.shape=', img.shape)

img = img.reshape(-1, 512, 512)
print('img.shape=', img.shape)

# 이미지를 출력
plt.imshow(img[0], cmap="gray")
plt.axis("off")
plt.show()
```

```
img.shape= (512, 512)
img.shape= (262144,)
img.shape= (1, 512, 512)
```

Open CV 배열 처리

- ❖ 영상 속성 과 픽셀 접근
  - ✓ 이미지 크기 변경
    - img.flatten()은 다차원 배열을 1차원 배열로 변경하여 img.shape = (262144,)
    - 3 img.reshape(-1, 512, 512)는 3차원 배열로 확장하는데 -1로 표시된 부분은 크기를 자동으로 계산하는데 img의 픽셀 크기가 512 X 512이므로 img.shape = (1, 512, 512)로 변경되고 img[0].shape은 (512, 512)
    - cv2.imshow('img', img[0])은 원본 영상을 표시
    - img.reshape()은 실제 데이터를 변경하지는 않고 모양을 변경
    - 영상의 확대 축소 크기는 cv2.resize()로 변환

Open CV 배열 처리

- 영상 속성 과 픽셀 접근
  - ✓ 이미지 크기 변경

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

# 흑백 이미지로 로드
image = cv2.imread("./data/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)

# 이미지 크기를 64x64 픽셀로 변경
image_64x64 = cv2.resize(image, (64, 64))

# 이미지를 출력
plt.imshow(image_64x64, cmap="gray"), plt.axis("off")
plt.show()
```

Open CV 배열 처리

- 영상 속성 과 픽셀 접근
  - ✓ 픽셀 접근

```
import cv2
##import numpy as np

img = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
img[100, 200] = 0  # 화소값(밝기,그레이스케일) 변경
print(img[100:110, 200:210]) # ROI 접근

##for y in range(100, 400):
##    for x in range(200, 300):
##        img[y, x] = 0

img[100:400, 200:300] = 0    # ROI 접근

plt.imshow(img, cmap="gray")
plt.axis("off")
plt.show()
```

Open CV 배열 처리

- 영상 속성 과 픽셀 접근
  - ✓ 픽셀 접근

```
[[  0 143 145 132 147 144 142 139 132 138]
 [138 138 143 151 137 144 139 139 138 138]
 [132 139 153 140 133 136 143 138 137 128]
 [137 146 138 125 132 145 139 142 130 128]
 [149 139 130 137 140 145 136 133 132 141]
 [141 139 134 149 149 137 132 127 140 140]
 [142 148 139 142 144 138 146 135 131 130]
 [151 146 136 131 142 144 149 135 126 132]
 [147 131 135 138 147 139 128 125 134 138]
 [135 132 149 142 134 128 122 135 138 129]]
```

Open CV 배열 처리

❖ 영상 속성 과 픽셀 접근

✓ 픽셀 접근

```
import cv2

##import numpy as np

img = cv2.imread('./data/lena.jpg')  # cv2.IMREAD_COLOR
img[100, 200] = [255, 0, 0]  # 컬러(BGR) 변경
print(img[100, 200:210])  # ROI 접근

##for y in range(100, 400):
##    for x in range(200, 300):
##        img[y, x] = [255, 0, 0]    # 빨강색(blue)으로 변경

img[100:400, 200:300] = [255, 0, 0]  # ROI 접근

plt.imshow(img)
plt.axis("off")
plt.show()
```

Open CV 배열 처리

- 영상 속성 과 픽셀 접근
  - ✓ 픽셀 접근

```
[[255   0   0]
 [116 115 207]
 [120 116 211]
 [107 103 198]
 [119 121 209]
 [116 118 206]
 [115 114 206]
 [112 111 203]
 [104 105 195]
 [110 112 200]]
```

Open CV 배열 처리

- 영상 속성 과 픽셀 접근
  - ✓ 채널 접근

```
import cv2
##import numpy as np

img = cv2.imread('./data/lena.jpg') # cv2.IMREAD_COLOR

##for y in range(100, 400):
##    for x in range(200, 300):
##        img[y, x, 0] = 255      # B-채널을 255로 변경

img[100:400, 200:300, 0] = 255  # B-채널을 255로 변경
img[100:400, 300:400, 1] = 255  # G-채널을 255로 변경
img[100:400, 400:500, 2] = 255  # R-채널을 255로 변경

plt.imshow(img)
plt.axis("off")
plt.show()
```

Open CV 배열 처리

- 이미지 자르기
  - ✓ 이미지는 numpy의 2차원 ndarray이기 때문에 배열 슬라이싱을 이용해서 이미지를 자를 수 있음
  - ✓ 고정 카메라가 촬영한 이미지에서는 특정 영역만 관심 대상이 될 수 있으므로 이러한 이미지 자르기를 이용하면 크기를 줄여서 사용하는 것이 가능

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

image = cv2.imread("./data/plane_256x256.jpg", cv2.IMREAD_GRAYSCALE)
# 행과 열의 절반을 선택
image_cropped = image[:128,:128]

plt.imshow(image_cropped, cmap="gray"), plt.axis("off")
plt.show()
```

Open CV 배열 처리

- 관심 영역 과 ROI
  - ✓ 영상의 사각 관심 영역(region of interest, ROI)을 numpy의 슬라이싱으로 지정하여 접근
  - ✓ ROI를 사용하여 블록 평균 영상을 생성하고 마우스를 이용한 ROI 영역 지정 함수로 selectROI()와 selectROIs()를 제공
  - ✓ selectROI(windowName, img[, showCrosshair[, fromCenter]])-> retval
    - windowName 윈도우(디폴트 ROIselector)에 img 영상을 표시하고 사용자가 마우스 클릭 과 드래그로 ROI를 선택할 수 있게 함
    - showCrosshair = True이면 선택 영역에 격자가 표시되고 fromCenter = True이면 마우스 클릭 위치 중심을 기준으로 박스가 선택됨
    - 선택을 종료하려면 Space 또는 Enter를 사용하고 반환값 retval에 선택 영역의 튜플(x, y, width, height)을 반환하는데 (x, y)는 박스의 왼쪽 상단 좌표이고 (width, height)는 박스의 가로, 세로 크기이며 C키를 사용하면 선택을 취소함
  - ✓ selectROIs(windowName, img[, showCrosshair[,fromCenter]])->boundingBoxes
    - windowName 윈도우에 img 영상을 표시하고 사용자가 마우스 클릭과 드래그로 다중 ROI를 선택할 수 있게 함
    - 마우스 클릭과 드래그로 각 ROI를 선택가능한 함수로 현재 선택을 종료 할 때는 Space 또는 Enter를 사용하고 선택 영역을 취소하려면 C 키를 선택하고 모든 ROI 선택을 종료하려면 R 키를 사용하여 선택한 영역의 리스트를 반환

Open CV 배열 처리

❖ 관심 영역 과 ROI

✓ 마우스로 ROI 영역 지정

```
import cv2

src = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
roi = cv2.selectROI(src)
print('roi =', roi)

img = src[roi[1]:roi[1]+roi[3],
          roi[0]:roi[0]+roi[2]]

cv2.imshow('Img', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

roi = (177, 181, 233, 195)

Open CV 배열 처리

❖ 관심 영역 과 ROI

✓ 마우스로 ROI 영역 지정

```
import cv2

src = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
rects = cv2.selectROIs('src', src, False, True)
print('rects =', rects)

for r in rects:
    cv2.rectangle(src, (r[0], r[1]), (r[0]+r[2], r[1]+r[3]), 255)
##    img = src[r[1]:r[1]+r[3], r[0]:r[0]+r[2]]
##    cv2.imshow('Img', img)
##    cv2.waitKey()

cv2.imshow('src', src)
cv2.waitKey()
cv2.destroyAllWindows()
```

```
rects = [[213 210  80  78]
 [275 164  76 122]]
```

Open CV 배열 처리

- 영상 복사
  - ✓ 원본 영상의 데이터를 그대로 유지하고 원본 영상의 복사본(출력 영상)에 라인, 사각 형, 원 등을 표시하는 경우가 많은데 원본 영상의 복사는 numpy.copy()로 복사하거나 np.zeros() 같은 함수로 영상을 생성한 후에 복사할 수 있음
  - ✓ dst = src와 같은 지정문은 복사가 아니라 참조(reference)를 생성하기 때문에 한 영상을 변경하면 다른 영상도 변경됨
  - ✓ 영상 복사

```
import cv2
from matplotlib import pyplot as plt
src = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)

##dst = src          #참조
dst = src.copy()     #복사
dst[100:400, 200:300] = 0

#plt.imshow(src, cmap="gray")
plt.imshow(dst, cmap="gray")
plt.axis("off")
plt.show()
```

Open CV 배열 처리

- 영상 복사
  - ✓ 영상 복사

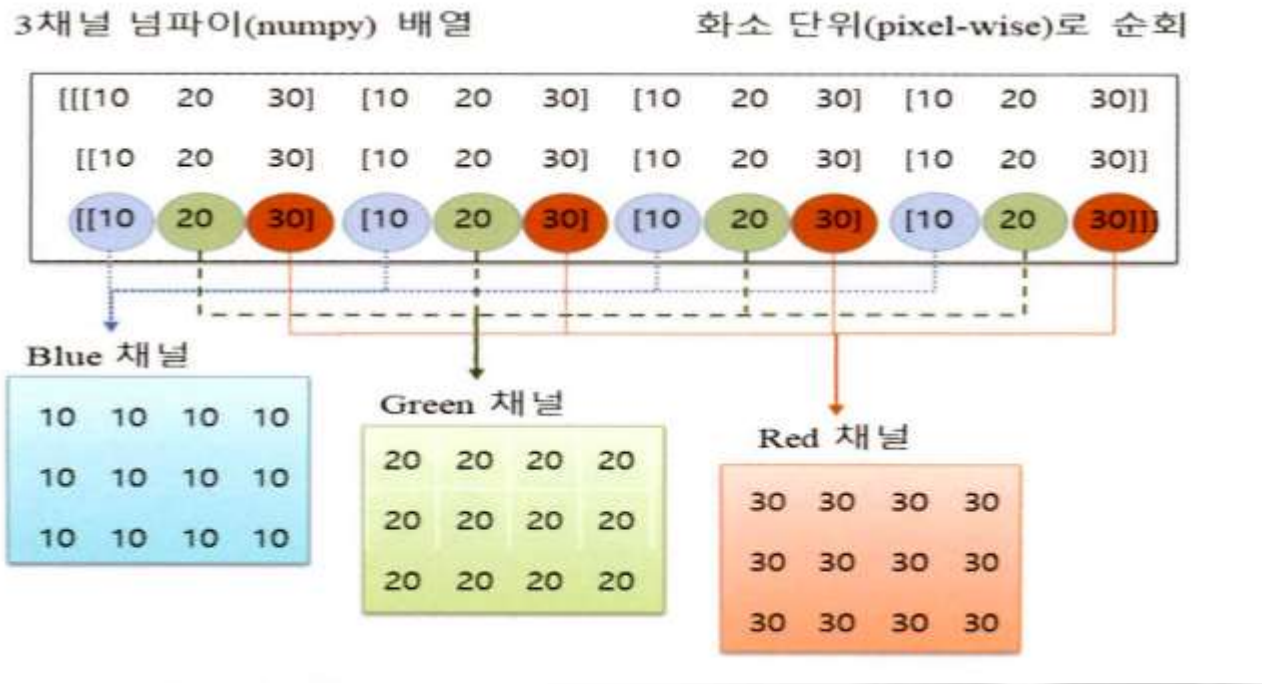
```
import cv2
import numpy as np

src = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
shape = src.shape[0], src.shape[1], 3
dst = np.zeros(shape, dtype=np.uint8)

dst[:,:,0] = src       # B-채널
##dst[:,:,1] = src     # G-채널
##dst[:,:,2] = src     # R-채널

dst[100:400, 200:300, :] = [255, 255, 255]

#plt.imshow(src)
plt.imshow(dst)
plt.axis("off")
plt.show()
```

Open CV 배열 처리

- ❖ 영상 복사
  - ✓ 영상 복사
    - dst = np.zeros(shape, dtype = np.uint8)는 src와 같은 가로, 세로 크기의 3-채널 컬러 영상 dst를 생성
    - dst[:, :, 0] = src 는 dst의 0-채널(blue)에 src를 복사
    - dst의 부분 영역 100:400, 200:300을 흰색 [255, 255, 255]로 변경하면 dst만 변경되고, src는 변경되지 않음

Open CV 배열 처리

- ❖ 영상 채널 분리 와 병합
  - ✓ 3채널 numpy 배열 및 컬러 영상의 각 채널



  - ✓ cv2.split()는 다중 채널 영상을 리스트에 단일 채널 영상으로 분리하고 cv2.merge()는 단일 채널 영상을 병합하여 다중 채널 영상을 생성

    cv2.split(m[, mv]) -> mv

    cv2.merge(mv[, dst]) -> dst

Open CV 배열 처리

- 영상 채널 분리 와 병합
  - ✓ 채널 분리 및 합성

```
import numpy as np
import cv2

# numpy array이용해 단일 채널 3개 생성
ch0 = np.zeros((2, 4), np.uint8) + 10          # 0원소 행렬 선언 후 10 더하기
ch1 = np.ones((2, 4), np.uint8) * 20           # 1원소 행렬 선언 후 20 곱하기
ch2 = np.zeros((2, 4), np.uint8); ch2[:] = 30  # 0원소 행렬 선언 후 행렬원소값 30 지정

list_bgr = [ch0, ch1, ch2]                     # 단일 채널들을 모아 리스트 구성
merge_bgr = cv2.merge(list_bgr)                # 채널 합성
split_bgr = cv2.split(merge_bgr)               # 채널 분리: 컬러영상--> 3채널 분리
```

Open CV 배열 처리

- 영상 채널 분리 와 병합
  - ✓ 채널 분리 및 합성

```
print('split_bgr 행렬 형태 ', np.array(split_bgr).shape)
print('merge_bgr 행렬 형태', merge_bgr.shape)

print("[ch0] = ₩n%s" % ch0)                    # 단일 채널 원소 출력
print("[ch1] = ₩n%s" % ch1)
print("[ch2] = ₩n%s" % ch2)
print("[merge_bgr] = ₩n %s₩n" % merge_bgr)      # 다중 채널 원소 출력

print("[split_bgr[0]] =₩n%s " % split_bgr[0])
print("[split_bgr[1]] =₩n%s " % split_bgr[1])
print("[split_bgr[2]] =₩n%s " % split_bgr[2])
```

Open CV 배열 처리

- 영상 채널 분리 와 병합
  - ✓ 채널 분리 및 합성

```
split_bgr 행렬 형태  (3, 2, 4)
merge_bgr 행렬 형태 (2, 4, 3)
[ch0] =
[[10 10 10 10]
 [10 10 10 10]]
[ch1] =
[[20 20 20 20]
 [20 20 20 20]]
[ch2] =
[[30 30 30 30]
 [30 30 30 30]]
```

Open CV 배열 처리

- 영상 채널 분리 와 병합
  - ✓ 채널 분리 및 합성

```
[merge_bgr] =
 [[[10 20 30]
  [10 20 30]
  [10 20 30]
  [10 20 30]]

 [[10 20 30]
  [10 20 30]
  [10 20 30]
  [10 20 30]]]

[split_bgr[0]] =
[[10 10 10 10]
 [10 10 10 10]]
[split_bgr[1]] =
[[20 20 20 20]
 [20 20 20 20]]
```

Open CV 배열 처리

- 영상 채널 분리 와 병합
  - ✓ 채널 분리

```
import cv2
from matplotlib import pyplot as plt
src = cv2.imread('./data/lena.jpg')

dst = cv2.split(src)
print(type(dst))
print(type(dst[0])) # type(dst[1]), type(dst[2])

plt.imshow(dst[0])
plt.axis("off")
plt.show()
```

Open CV 배열 처리

- ❖ 영상 채널 분리 와 병합
  - ✓ 채널 분리
    - dst = cv2.split(src)는 3-채널 BGR 컬러 영상 src를 채널 분리하여 리스트 dst에 저장
    - type(dst)는 'list'이고 type(dst[0]), type(dst[1]), type(dst[2])는 'numpy.ndarray'
    - 채널 순서는 0-채널은 Blue, 1-채널은 Green, 2-채널은 Red

Open CV 배열 처리

- 영상 채널 분리 와 병합
  - ✓ 채널 병합

```
import cv2
src = cv2.imread('./data/lena.jpg')

b, g, r = cv2.split(src)
dst = cv2.merge([b, g, r]) # cv2.merge([r, g, b])

print(type(dst))
print(dst.shape)

plt.imshow(dst)
plt.axis("off")
plt.show()
```

Open CV 배열 처리

❖ 컬러 공간 변환

✓ cvtColor() 함수는 GRAY, HSV, YCrCb 등의 다양한 컬러 공간 포맷으로 변환

cv2.cvtColor(src, code[, dst[, dstCn]]) -> dst

- 입력 영상 src를 code에 따라 출력 영상 dst에 변환
- dstCn은 출력 영상의 채널 수로 컬러 변환에서 R, G. B 채널의 값은 영상의 자료형에 따라 np.uint8은 0..255, np.uint16은 0..65535, n이laat32는 0..1의 범위로 간주하며 선형 변환의 경우는 픽셀 자료형을 따르는 값의 범위가 문제 되지 않지만 비선형 변환의 경우 문제가 될 수 있으므로 컬러 변환 전에 astype() 함수를 사용하여 np.uint16, np.uint32 또는 np.float32로 변환할 필요가 있을 수 있음
- BGR 채널 컬러 영상을 1-채널 GRAY 영상으로 변환하거나 컬러 영상 처리를 위하여 HSV, YCrCb 포맷으로 변환하여 처리하는 경우가 많음

입력 영상(src)	변환 코드(code)	출력 영상(dst)
BGR	cv2.COLOR_BGR2GRAY	GRAY
GRAY	cv2.COLOR_GRAY2BGR	BGR
BGR	cv2.COLOR_BGR2HSV	HSV
HSV	cv2.COLOR_HSV2BGR	BGR
BGR	cv2.COLOR_BGR2YCrCb	YCrCb
YCrCb	cv2.COLOR_YCrCb2BGR	BGR

Open CV 배열 처리

- ❖ 컬러 공간 변환
  - ✓ 컬러 변환

```
import cv2
src = cv2.imread('./data/lena.jpg')

gray  = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
yCrCv = cv2.cvtColor(src, cv2.COLOR_BGR2YCrCb)
hsv   = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)

#plt.imshow(gray, cmap='gray')
#plt.imshow(yCrCv)
plt.imshow(hsv)

plt.axis("off")
plt.show()
```

Open CV 배열 처리

- ❖ 컬러 공간 변환
  - ✓ 컬러 변환
    - gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)은 BGR 채널 컬러 영상 src를 GRAY 영상 gray로 변환
    - yCrCv = cv2.cvtColor(src, cv2.COLOR_BGR2YCrCb)는 BGR 채널 컬러 영상 src를 YCrCb 컬러 영상 yCrCv로 변환
    - hsv = cv2.cvtColor(src, cv2.COLOR_BGR2HSV)는 BGR 채널 컬러 영상 src를 HSV 컬러영상 yCrCv로 변환

Open CV 배열 처리

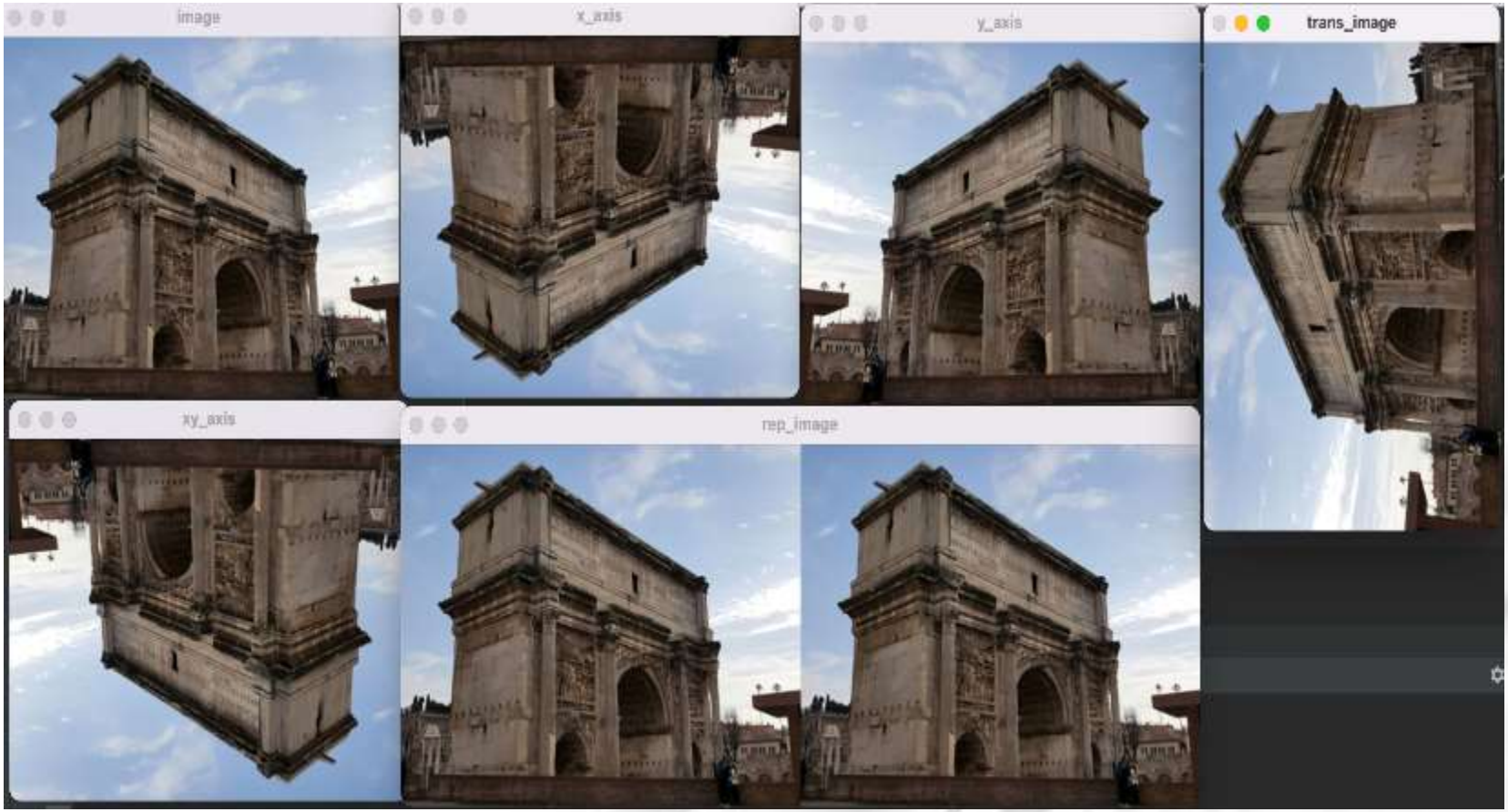
- 회전
  - ✓ 입력된 2차원 배열을 수직, 수평 양축으로 뒤집기

    cv2.flip(src, flipCode[, dst]) -> dst
    - src , dst: 입력 배열, 출력 배열
    - flipCode: 배열을 뒤집는 축
      - 0 : X축을 기준으로 위아래로 뒤집기
      - 1: y축을 기준으로 좌우로 뒤집기
      - -1: 양축(x축, y축 모두)을 기준으로 뒤집기
  - ✓ 입력된 배열을 복사

    cv2.repeat(src, ny, nx[, dst) -> dst
    - src , dst: 입력 배열, 출력 배열
    - ny, nx: 수직 방향, 수평 방향 반복 횟수
  - ✓ 입력 행렬의 전치 행렬을 출력으로 리턴
    - cv2.transpose(src[, dst]) -> dst
    - src, dst: 입력 배열, 출력 배열

Open CV 배열 처리

- 회전
  - ✓ 상하좌우로 뒤집기



Open CV 배열 처리

- 회전
  - ✓ 상하좌우로 뒤집기

```
import cv2

image = cv2.imread("./data/flip_test.jpg", cv2.IMREAD_COLOR)
if image is None: raise Exception("영상 파일 읽기 오류 발생") # 예외 처리

x_axis = cv2.flip(image, 0)                # x축 기준 상하 뒤집기
y_axis = cv2.flip(image, 1)                # y축 기준 좌우 뒤집기
xy_axis = cv2.flip(image, -1)
rep_image   = cv2.repeat(image, 1, 2)      # 반복 복사
trans_image = cv2.transpose(image)         # 행렬 전치

## 각 행렬을 영상으로 표시
titles = ['image', 'x_axis', 'y_axis','xy_axis','rep_image','trans_image']
for title in titles:
    cv2.imshow(title, eval(title))
cv2.waitKey(0)
```

Open CV 배열 처리

❖ 회전

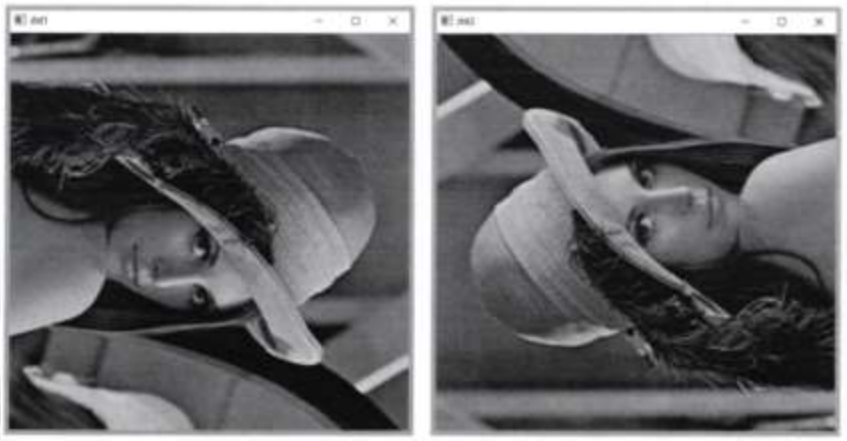
cv2.rotate(src, rotateCode[, dst]) -> dst

✓ 입력 영상 src를 크기 rotateCode에 따라 90의 배수로 회전시켜 dst에 반환

✓ rotateCode는 cv2.ROTATE_90_CLOCKWISE, cv2.ROTATE_180, cv2.ROTATE_90 COUNTERCLOCKWISE 등이 있음

✓ 회전

```
import cv2
src = cv2.imread('./data/lena.jpg')

dst1 = cv2.rotate(src, cv2.ROTATE_90_CLOCKWISE)
dst2 = cv2.rotate(src, cv2.ROTATE_90_COUNTERCLOCKWISE)

cv2.imshow('dst1',  dst1)
cv2.imshow('dst2',  dst2)
cv2.waitKey()
cv2.destroyAllWindows()
```



Open CV 배열 처리

❖ 회전

cv2.getRotationMatrix2D(center, angle, scale) -> M

✓ center 좌표를 중심으로 scale 만큼 확대/축소하고 angle 각도만큼 회전한 어파인 변환 행렬 M을 반환

✓ angle > 0 이면 반 시계 방향 회전

✓ M[:, 2] += (tx, ty)를 추가하면 이동을 추가할 수 있음

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha) \times center.x - \beta \times center.y \\ -\beta & \alpha & \beta \times center.x + (1-\alpha) \times center.y \end{bmatrix}$$

$$\alpha = scale \times cos(RADIAN(angle))$$

$$\beta = scale \times sin(RADIAN(angle))$$

Open CV 배열 처리

❖ 회전

cv2.warpAffine(src, M, dsize[, dst[, flags[,borderMode, borderValue]]]) -> dst

- ✓ cv2.warpAffine() 함수는 src 영상에 2 X 3 어파인 변환 행렬 M을 적용하여 dst에 반환
- ✓ dsize는 출력 영상 dst의 크기이며 flags는 보간법(cv2.INTER_NEAREST, cv2.INTER_LINEAR 등) 과 cv2_WARP_INVERSE_MAP의 조합
- ✓ Cv2.WARP_INVERSE_MAP은 이이 dst -> src의 역변환 을 의미
- ✓ borderMode는 경계값 처리 방식으로 borderMode = cv2_BORDER_CONSTANT 에서 borderValue는 경계값 상수

Open CV 배열 처리

- 회전
  - ✓ 영상 어파인 변환

```
import cv2
src = cv2.imread('./data/lena.jpg')

rows, cols, channels = src.shape
M1 = cv2.getRotationMatrix2D( (rows/2, cols/2),  45, 0.5 )
M2 = cv2.getRotationMatrix2D( (rows/2, cols/2), -45, 1.0 )

dst1 = cv2.warpAffine( src, M1, (rows, cols))
dst2 = cv2.warpAffine( src, M2, (rows, cols))

cv2.imshow('dst1',  dst1)
cv2.imshow('dst2',  dst2)
cv2.waitKey()
cv2.destroyAllWindows()
```

Open CV 배열 연산

- 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수
  - ✓ OpenCV_Python은 영상을 numpy.ndarray로 표현하기 때문에 numpy 연산을 사용할 수 있음
  - ✓ numpy 연산을 사용할 때 연산 결과가 자료형의 범위를 벗어나는 경우 주의해서 사용해야 하는데 uint8 자료형의 영상 src1과 src2에서 dst = src1 + src2의 연산을 하는 경우 255를 넘는 픽셀 값은 256으로 나눈 나머지를 갖지만 dst = cv2.add(src1, src2)는 255를 넘는 픽셀 값은 최대값 255를 저장 – saturate()
  - ✓ 함수들의 공통 설정
    - src, src1, src2는 입력 영상이고 dst는 연산의 결과 영상
    - mask는 8-비트 1-채널 마스크 영상으로 mask(y, x) 0인 아닌 픽셀에서만 연산을 수행
    - dtype에 출력 영상의 픽셀 자료형을 명시할 경우 cv2.CV_8U, cv2.CV_8UCl, cv2. CV_8UC3, cv2.CV_8UCl, cv2.CV_16S, cv2.CV_16SCl, cv2.CV_32F, cv2. CV_32FC3, cv2.CV_64F, cv2.CV_64FC3 등과 같이 비트, 타입, 채널 수를 명시한 데이터를 사용
    - OpenCV 함수의 매개변수 dst를 생략하거나 dst = None을 사용하면 결과 영상을 생성하여 반환

Open CV 배열 연산

❖ 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수

사칙 연산	cv2.add(src1, src2[, dst[, mask[, dtype]]]) -> dst
	cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]]) -> dst
	cv2.subtract(src1, src2[, dst[, mask[, dtype]]]) -> dst
	cv2.scaleAdd(src1, alpha, src2[, dst]) -> dst
	cv2.multiply(src1, src2[, dst[, scale[, dtype]]]) -> dst
	cv2.divide(src1, src2[, dst[, scale[, dtype]]]) -> dst
	cv2.divide(scale, src2[, dst[, dtype]]) -> dst
비트 연산	cv2.bitwise_not(src[, dst[, mask]]) -> dst
	cv2.bitwise_and(src1, src2[, dst[, mask]]) -> dst
	cv2.bitwise_or(src1, src2[, dst[, mask]]) -> dst
	cv2.bitwise_xor(src1, src2[, dst[, mask]]) -> dst
비교 범위 연산	cv2.compare(src1, src2, cmpop[, dst]) -> dst
	cv2.checkRange(a[, quiet[, minVal[, maxVal]]]) -> retval, pos
	cv2.inRange(src, lowerb, upperb[, dst]) -> dst
수치 연산	cv2.absdiff(src1, src2[, dst]) -> dst
	cv2.convertScaleAbs(src[, dst[, alpha[, beta]]]) -> dst
	cv2.exp(src[, dst]) -> dst
	cv2.log(src[, dst]) -> dst
	cv2.pow(src, power[, dst]) -> dst
	cv2.sqrt(src[, dst]) -> dst
	cv2.magnitude(x, y[, magnitude]) -> magnitude
	cv2.phase(x, y[, angle[, angleInDegrees]]) -> angle
	cv2.cartToPolar(x, y[, magnitude[, angle[, angleInDegrees]]]) -> magnitude, angle
	cv2.polarToCart(magnitude, angle[, x[, y[, angleInDegrees]]]) -> x, y

Open CV 배열 연산

❖ 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수

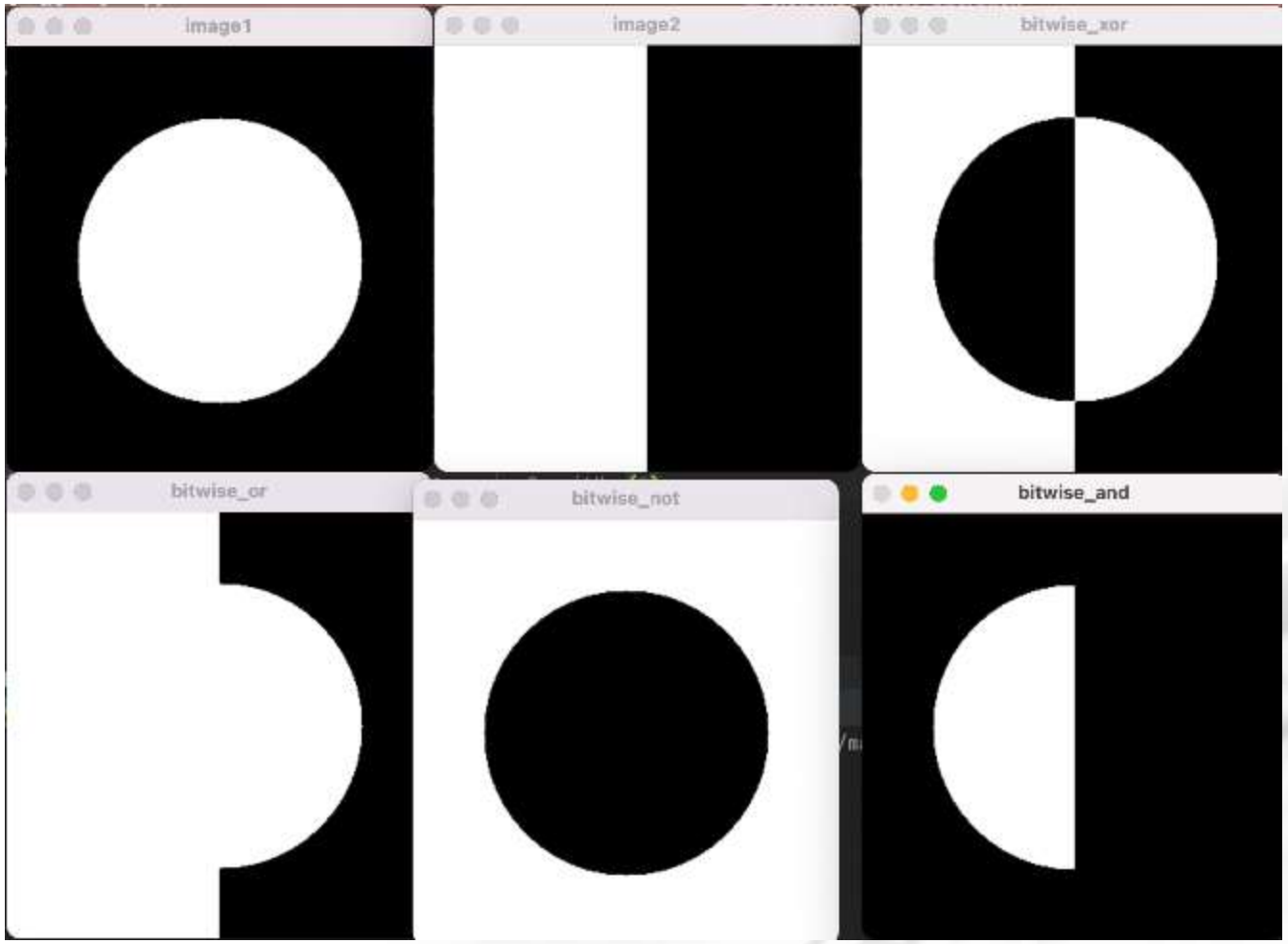
✓ 영상 연산

```
import cv2
import numpy as np

src1 = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
src2 = np.zeros(shape=(512,512), dtype=np.uint8) + 100

dst1 = src1 + src2
dst2 = cv2.add(src1, src2)
#dst2 = cv2.add(src1, src2, dtype = cv2.CV_8U)

cv2.imshow('dst1',  dst1)
cv2.imshow('dst2',  dst2)
cv2.waitKey()
cv2.destroyAllWindows()
```

(a) dst1 = src1 + src2　　(b) dst2 = cv2.add(src1, src2)

Open CV 배열 연산

- 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수
  - ✓ 비트 연산



Open CV 배열 연산

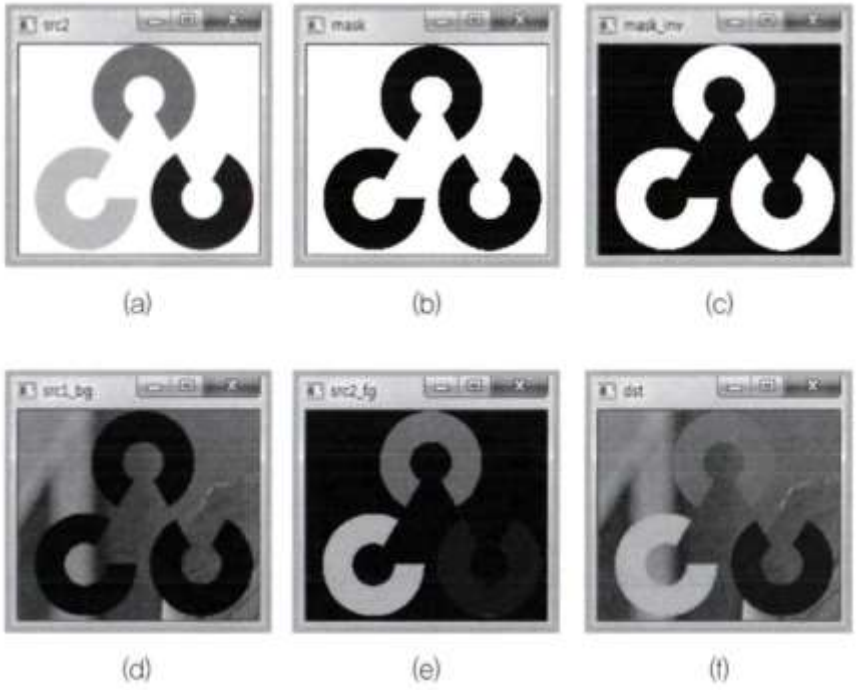
❖ 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수

✓ 비트 연산

```
import numpy as np, cv2

image1 = np.zeros((300, 300), np.uint8)       # 300행, 300열 검은색 영상 생성
image2 = image1.copy()

h, w = image1.shape[:2]
cx,cy  = w//2, h//2
cv2.circle(image1, (cx,cy), 100, 255, -1)                  # 중심에 원 그리기
cv2.rectangle(image2, (0,0, cx, h), 255, -1)

image3 = cv2.bitwise_or(image1, image2)                    # 원소 간 논리합
image4 = cv2.bitwise_and(image1, image2)                   # 원소 간 논리곱
image5 = cv2.bitwise_xor(image1, image2)                   # 원소 간 배타적 논리합
image6 = cv2.bitwise_not(image1)              # 행렬 반전

cv2.imshow("image1", image1);
cv2.imshow("image2", image2)
cv2.imshow("bitwise_or", image3);
cv2.imshow("bitwise_and", image4)
cv2.imshow("bitwise_xor", image5);
cv2.imshow("bitwise_not", image6)
cv2.waitKey(0)
```

Open CV 배열 연산

- 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수
  - ✓ 비트 연산



Open CV 배열 연산

- 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수
  - ✓ 비트 연산

```
#비트 연산
#OpenCV-Python Tutorials 참조
import cv2
import numpy as np

src1 = cv2.imread('./data/lena.jpg')
src2 = cv2.imread('./data/opencv_logo.png')
cv2.imshow('src2',  src2)

#1
rows,cols,channels = src2.shape
roi = src1[0:rows, 0:cols]

#2
gray = cv2.cvtColor(src2,cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(gray, 160, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)
cv2.imshow('mask',  mask)
cv2.imshow('mask_inv',  mask_inv)
```

Open CV 배열 연산

- 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수
  - ✓ 비트 연산

```
#3
src1_bg = cv2.bitwise_and(roi, roi, mask = mask)
cv2.imshow('src1_bg',  src1_bg)

#4
src2_fg = cv2.bitwise_and(src2, src2, mask = mask_inv)
cv2.imshow('src2_fg',  src2_fg)

#5
##dst = cv2.add(src1_bg, src2_fg)
dst = cv2.bitwise_or(src1_bg, src2_fg)
cv2.imshow('dst',  dst)

#6
src1[0:rows, 0:cols] = dst

cv2.imshow('result',src1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Open CV 배열 연산

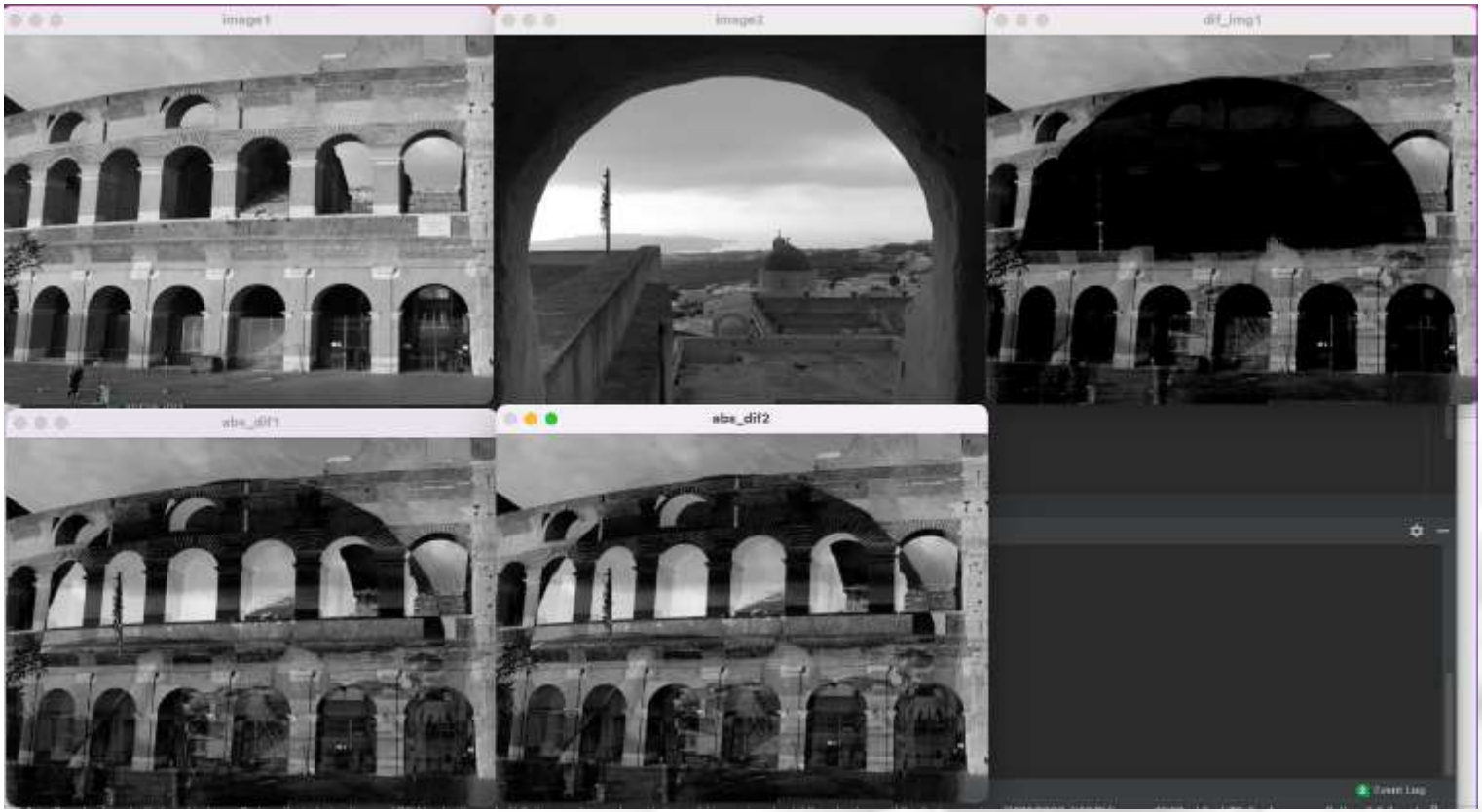
❖ 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수

✓ 반전 영상

```
#반전 영상
import cv2
import numpy as np

src1 = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)
src2 = np.zeros(shape=(512,512), dtype=np.uint8)+255

dst1 = 255 - src1
dst2 = cv2.subtract(src2, src1)
dst3 = cv2.compare(dst1, dst2, cv2.CMP_NE) # cv2.CMP_EQ
n    = cv2.countNonZero(dst3)
print('n = ', n)

cv2.imshow('dst1',  dst1)
cv2.imshow('dst2',  dst2)
cv2.waitKey()
cv2.destroyAllWindows()
```

Open CV 배열 연산

- 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수
  - ✓ 절대값 연산



Open CV 배열 연산

❖ 산술 연산, 비트 연산, 비교 범위, 수치 연산 함수

✓ 절대값 연산

```
import numpy as np, cv2

image1 = cv2.imread("./data/abs_test1.jpg", cv2.IMREAD_GRAYSCALE) # 명암도 영상 읽기
image2 = cv2.imread("./data/abs_test2.jpg", cv2.IMREAD_GRAYSCALE)
if image1 is None or image2 is None: raise Exception("영상 파일 읽기 오류 발생")

dif_img1 = cv2.subtract(image1, image2)                    # 차분 연산
dif_img2 = cv2.subtract(np.int16(image1), np.int16(image2)) # 음수 보전 위해
abs_dif1 = np.absolute(dif_img2).astype('uint8')
abs_dif2 = cv2.absdiff(image1, image2)              # 차분 절댓값 계산

x, y, w, h = 100, 100, 7, 3
print("[dif_img1(roi) uint8] = ₩n%s₩n" % dif_img1[y:y+h, x:x+w])
print("[dif_img2(roi) int16]  = ₩n%s₩n"  % dif_img2[y:y+h, x:x+w])
print("[abs_dif1(roi)] = ₩n%s₩n" % abs_dif1[y:y+h, x:x+w])
print("[abs_dif2(roi)] = ₩n%s₩n" % abs_dif2[y:y+h, x:x+w])

titles = ['image1', 'image2', 'dif_img1', 'abs_dif1','abs_dif2']
for title in titles:
    cv2.imshow(title, eval(title))
cv2.waitKey(0)
```

Open CV 배열 연산

❖ 수학 및 통계 함수

정규화	cv2.norm(src1, src2[, normType[, mask]]) -> retval
	cv2.normalize(src, dst[, alpha[,beta[,norm_type[, dtype[,mask]]]]]) -> dst
최대 최소	cv2.min(src1, src2[, dst]) -> dst
	cv2.max(src1, src2[, dst]) -> dst
	cv2.minMaxLoc(src[, mask]) -> minVal, maxVal, minLoc, maxLoc
통계	cv2.countNonZero(src) -> retval
	cv2.reduce(src, dim, rtype[, dst[, dtype]]) -> dst
	cv2.mean(src[, mask]) -> retval
	cv2.meanStdDev(src[, mean[, stddev[, mask]]]) -> mean, stddev
	cv2.calcCovarMatrix(samples, flags[, covar[, mean[, ctype]]]) -> covar, mean
	cv2.Mahalanobis(v1, v2, icovar) -> retval

Open CV 배열 연산

❖ 수학 및 통계 함수

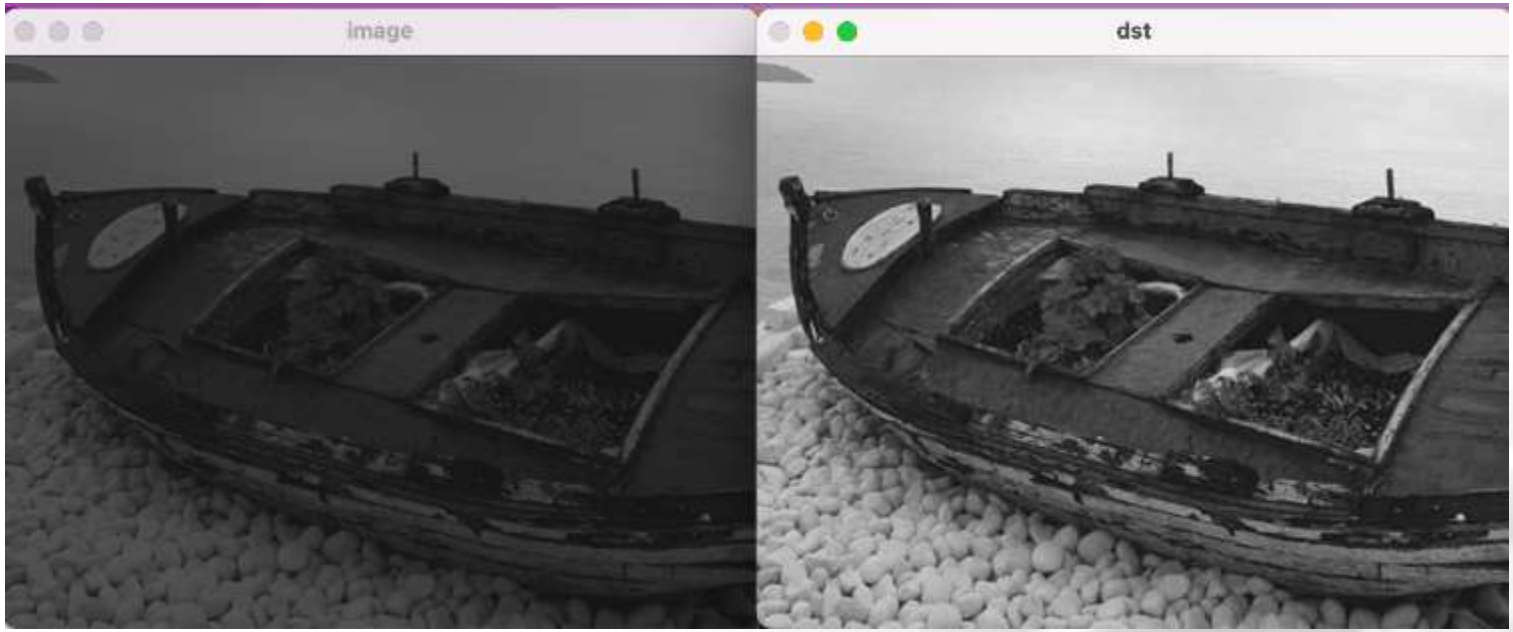
난수	cv2.randu(dst, low, high) -> dst
	cv2.randn(dst, mean, stddev) -> dst
	cv2.randShuffle(dst[, iterFactor]) -> dst
선형대수	cv2.eigen(src[, eigenvalues[, eigenvectors]]) -> retval, eigenvalues, eigenvectors
	cv2.PCACompute(data, mean[, eigenvectors[, maxComponents]]) -> mean, eigenvectors
	cv2.PCAProject(data, mean, eigenvectors[, result]) -> result
	cv2.PCABackProject(data, mean, eigenvectors[, result]) -> result
정렬	cv2.sort(src, flags[, dst]) -> dst
	cv2.sortIdx(src, flags[, dst]) -> dst

Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ 최소값 과 최대값
    - 두 행렬의 각 원소를 비교하여 큰 값이나 작은 값을 찾아야 할 때가 종종 있음
    - 하나의 행렬에서 가장 큰 값이나 가장 작은 값의 위치를 알면 히스토그램을 그리거나 다른 행렬의 연산 등에서 상당히 유용
    - 사용할 수 있는 함수는 cv2.max(), cv2.min(), cv2.minMaxLoc() 등

Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ 최소값 과 최대값을 이용한 화질 개선 – 차이가 적은 경우



Open CV 배열 연산

- 수학 및 통계 함수
  - ✓ 최소값 과 최대값을 이용한 화질 개선 – 차이가 적은 경우

```
import numpy as np, cv2

image = cv2.imread("./data/minMax.jpg", cv2.IMREAD_GRAYSCALE)
if image is None: raise Exception("영상 파일 읽기 오류 발생")

(min_val, max_val, _, _) = cv2.minMaxLoc(image)  # 최소값과 최대값 가져오기

ratio = 255/(max_val - min_val)
dst = np.round((image - min_val) * ratio).astype('uint8')
(min_dst, max_dst, _, _) = cv2.minMaxLoc(dst)

print("원본 영상 최소값= %d , 최대값= %d" % (min_val, max_val))
print("수정 영상 최소값= %d , 최대값= %d" % (min_dst, max_dst))
cv2.imshow("image", image)
cv2.imshow("dst"  , dst)
cv2.waitKey(0)
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 영상 정규화

```
import cv2
import numpy as np

src = cv2.imread('./data/lena.jpg', cv2.IMREAD_GRAYSCALE)

minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(src)
print('src:', minVal, maxVal, minLoc, maxLoc)

dst = cv2.normalize(src, None, 100, 200, cv2.NORM_MINMAX)
minVal, maxVal, minLoc, maxLoc = cv2.minMaxLoc(dst)
print('dst:', minVal, maxVal, minLoc, maxLoc)

cv2.imshow('dst',  dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

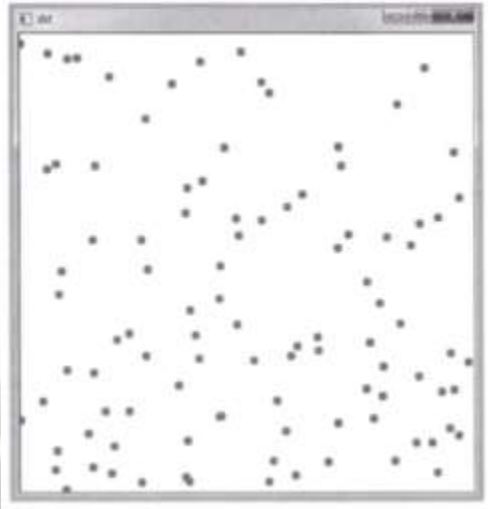
```
src: 18.0 248.0 (265, 198) (116, 273)
dst: 100.0 200.0 (265, 198) (116, 273)
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 2차원 균등 분포 난수 좌표

```
import cv2
import numpy as np
import time

dst = np.full((512,512,3), (255, 255, 255), dtype= np.uint8)
nPoints = 100
pts = np.zeros((1, nPoints, 2), dtype=np.uint16)

cv2.setRNGSeed(int(time.time()))
cv2.randu(pts, (0, 0), (512, 512))

# draw points
for k in range(nPoints):
    x, y = pts[0, k][:] # pts[0, k, :]
    cv2.circle(dst,(x,y),radius=5,color=(0,0,255),thickness=-1)

cv2.imshow('dst',  dst)
cv2.waitKey()
cv2.destroyAllWindows()
```



Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 2차원 정규 분포 난수 좌표

```
import cv2
import numpy as np
import time

dst = np.full((512,512,3), (255, 255, 255), dtype= np.uint8)
nPoints = 100
pts = np.zeros((1, nPoints, 2), dtype=np.uint16)

cv2.setRNGSeed(int(time.time()))
cv2.randn(pts, mean=(256, 256), stddev=(50, 50))

# draw points
for k in range(nPoints):
    x, y = pts[0][k, :] # pts[0, k, :]
    cv2.circle(dst,(x,y),radius=5,color=(0,0,255),thickness=-1)

cv2.imshow('dst', dst)
cv2.waitKey()
cv2.destroyAllWindows()
```

Open CV 배열 연산

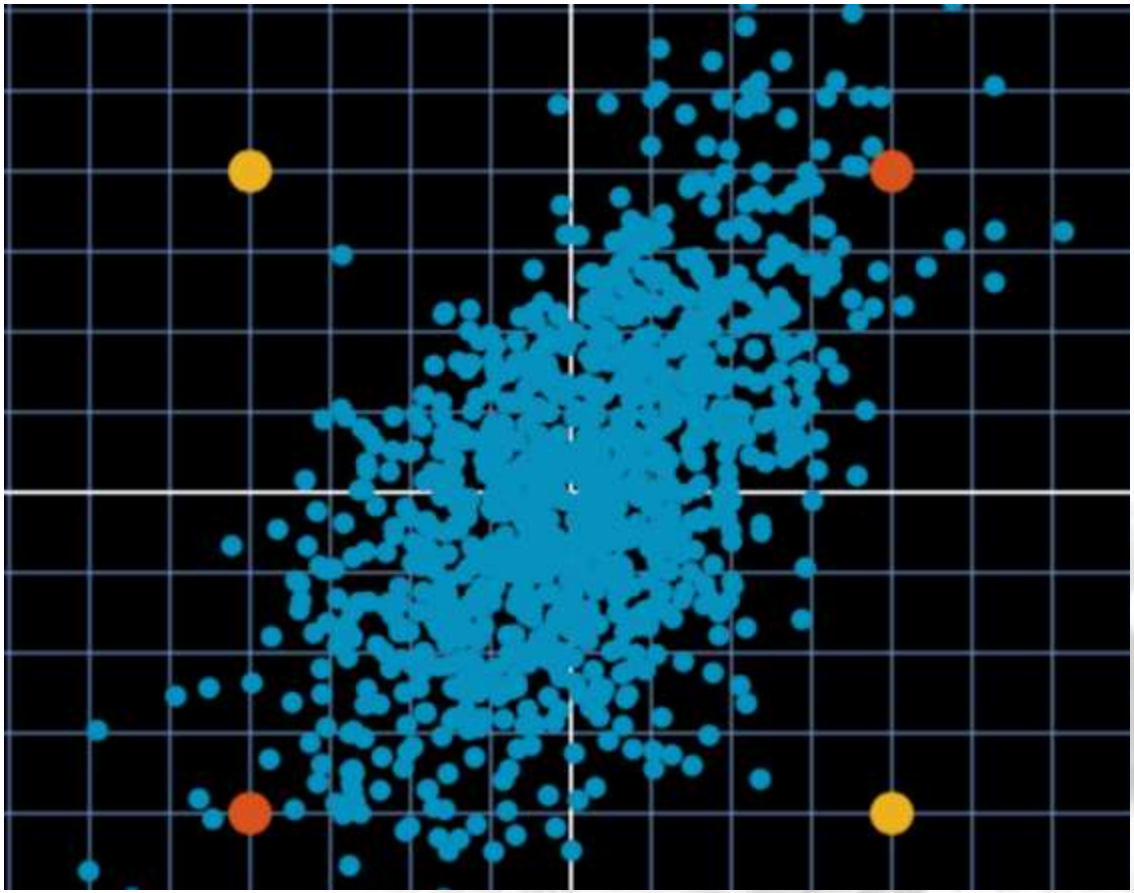
- ❖ 수학 및 통계 함수
  - ✓ 마하라노비스 거리
    - 평균 과 거리가 표준 편차의 몇 배인지 나타내는 값
    - 직관적인 이해
      - ◆ 교통량을 체크한다고 가정
      - ◆ 하루에 차가 평균 20대 표준편차는 3이라고 할 때 하루에 평균적으로 20대가 지나가고 들쑥날쑥한 정도가 평균적으로 3대라는 정도로 이해해 볼 수 있으므로 하루에 17~23대가 지나가는 경우가 일반적
      - ◆ 어느 날 26대가 지나간 경우 평균보다 6대나 더 많이 지나갔고 일반적으로 17~23대가 지나가기 때문에 이 날은 특히 교통량이 많았다고 볼 수 있음
      - ◆ 마할라노비스 거리는 그 값이 얼마나 일어나기 어려운(힘든) 값인지를 수치화한 방법
      - ◆ 마할라노비스 거리 계산

        Mahalnobis distance = (26 - 20) / 3 = 2

        - 26대가 지나간 것은 표준적인 편차의 2배 정도의 오차가 있는 값이라는 것
      - ◆ 서로 다른 두 점 사이의 거리가 아니라 한 점(관측치)과 분포 사이의 거리를 계산

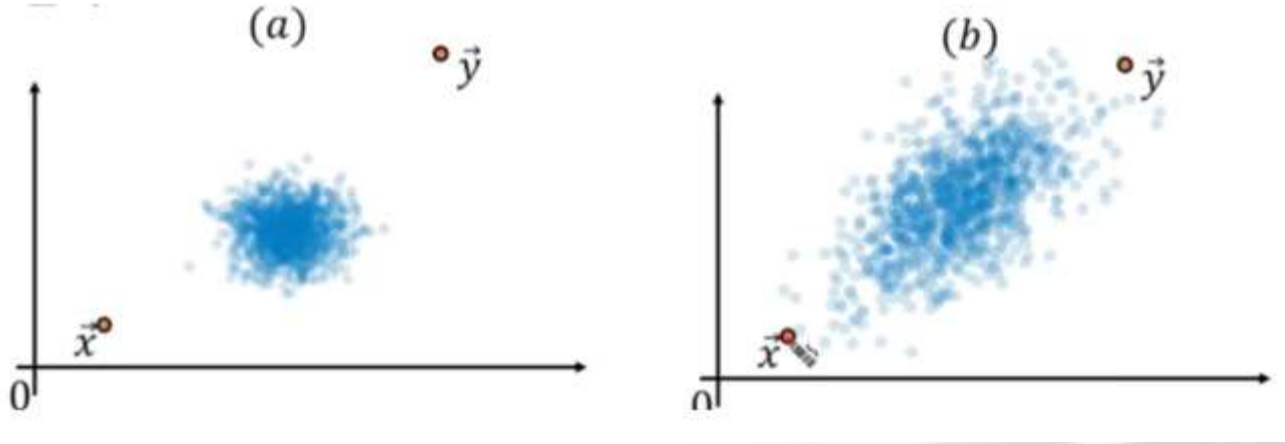
Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ 마하라노비스 거리



Open CV 배열 연산

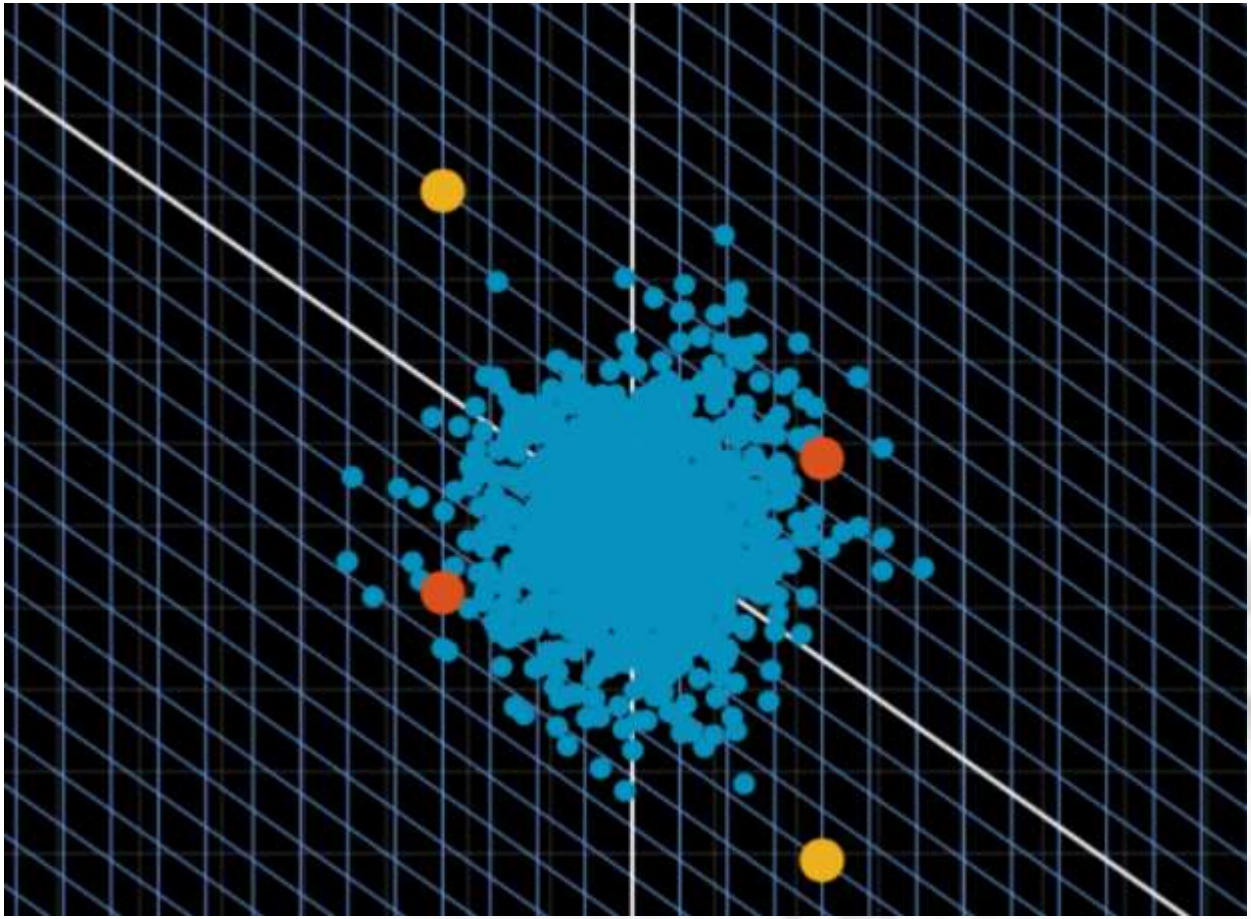
- ❖ 수학 및 통계 함수
  - ✓ 마하라노비스 거리
    - 파란 점들로 이루어진 분포가 있다고 할 때 노란색 점 끼리의 거리와 빨간색 점 끼리의 거리는 유클리디안 거리로는 같지만 분포상에서 상대적인 거리는 노란색 점 끼리의 거리가 빨간색 점 끼리의 거리보다 더 멀다고 할 수 있음



- 주변에 있는 다른 데이터 분포를 고려한다면,(a)에서의 두 점의 거리가 (b)에서의 두 점의 거리보다 더 멀다고 표현하는 게 더 맞지 않을까 하는 것을 수식적으로 표현하고자 하는 것이 마할라노비스 거리

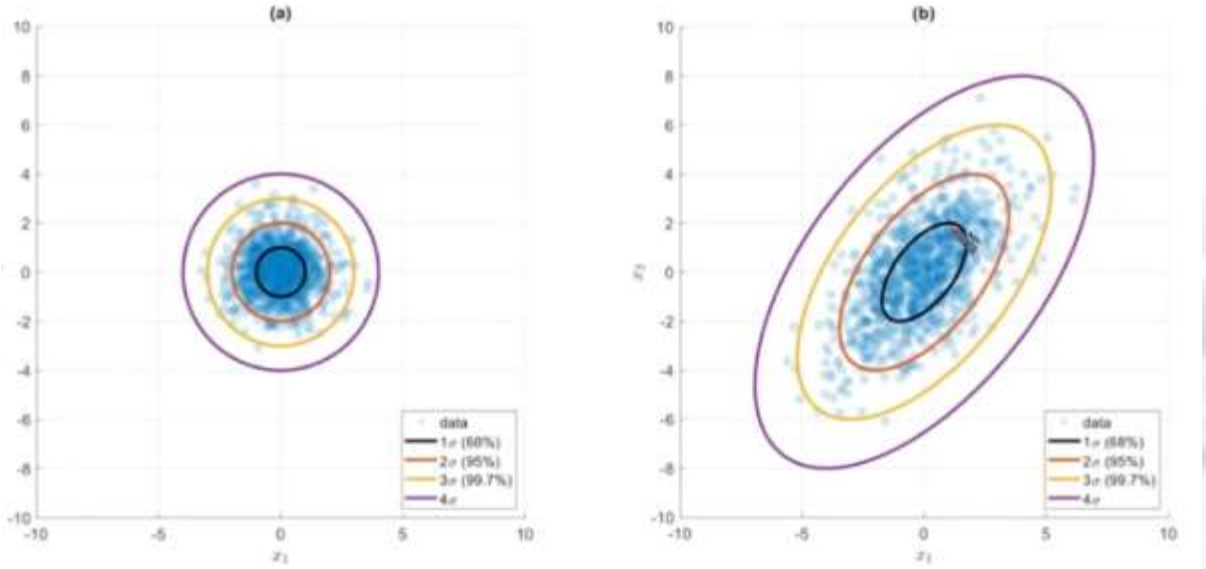
Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ 마하라노비스 거리
    - ● 계산 방법



Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ 마하라노비스 거리
    - ● 계산 방법
      - ◆ 파란색 점(데이터)들을 정규화 시키는 선형 변환을 통해 축을 변형시키고 그 위에서 노란점, 빨간점을 표현하면 노란색끼리의 거리가 더 멀다는 것을 시각적으로 확인할 수 있음



Open CV 배열 연산

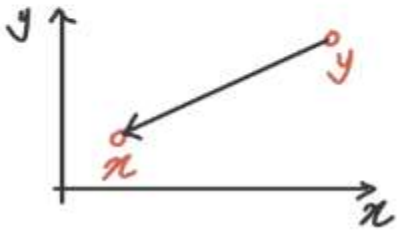
- ❖ 수학 및 통계 함수
  - ✓ 마하라노비스 거리
    - ● 계산 방법

$$d(u, v) = \sqrt{(u - v)\Sigma^{-1}(u - v)^T}$$

공분산 행렬

유클리드 거리

$$d_E = \sqrt{(\vec{x}-\vec{y})(\vec{x}-\vec{y})^T}$$



마할라노비스 거리

$$d_M = \sqrt{(\vec{x}-\vec{y})\Sigma^{-1}(\vec{x}-\vec{y})^T}$$

공분산 행렬의 역행렬

→ 정규화과정

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 마하라노비스 거리 계산

```
import cv2
import numpy as np
X = np.array([[0, 0,  0,100,100,150, -100,-150],
              [0,50,-50,  0, 30,100,  -20,-100]], dtype=np.float64)
X = X.transpose() # X = X.T

cov, mean = cv2.calcCovarMatrix(X, mean=None,
                                flags = cv2.COVAR_NORMAL + cv2.COVAR_ROWS)
print('mean=', mean)
print('cov=', cov)

ret, icov = cv2.invert(cov)
print('icov=',icov)

v1 = np.array([[0],[0]] , dtype=np.float64)
v2 = np.array([[0],[50]], dtype=np.float64)

dist = cv2.Mahalanobis(v1, v2, icov)
print('dist = ', dist)
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 마하라노비스 거리 계산

```
mean= [[12.5    1.25]]
cov= [[73750.   34875. ]
 [34875.   26287.5]]
icov= [[ 3.63872307e-05 -4.82740722e-05]
 [-4.82740722e-05   1.02084955e-04]]
dist =   0.5051854992128457
```

- X = X.transpose() #X = X.T는 X의 전치 행렬로 변경하여 각 행에 2차원 좌표를 위치 시킴
- cov, mean = cv2. calcCovarMatrix(X, mX, cv2. COVAR NORMAL + cv2.COVAR_ROWS)는 X의 각 행(cv2.COVAR_ROWS)에서 (x, y) 좌표의 평균 mean, 공분산 행렬 cov를 계산하고 평균 mean은 1 X 2 열벡터이고 공분산 행렬 cov는 2 X 2 행
- 데이터가 행렬의 열에 있으면 flags에 cv2.COVAR_COLS를 사용
- ret, icov = cv2.invert(cov)는 공분산 행렬 cov의 역행렬 icov를 계산
- dist = cv2.Mahalanobis(v1, v2, icov)는 두 벡터 v1, v2 사이의 마하라노비스 통계적 거리는 공분산 행렬의 역행렬을 이용하여 계산
- v1 = $[0, 0]^T$ 와 v2 = $[0, 100]^T$ 사이의 거리는 dist = 0.5051854992128457

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 행렬 축소 - reduce

```
import numpy as np, cv2

m = np.random.rand(3,5) * 1000//10

reduce_sum = cv2.reduce(m, dim=0, rtype=cv2.REDUCE_SUM) # 0 - 열방향 축소
reduce_avg = cv2.reduce(m, dim=1, rtype=cv2.REDUCE_AVG) # 1 - 행방향 축소
reduce_max = cv2.reduce(m, dim=0, rtype=cv2.REDUCE_MAX)
reduce_min = cv2.reduce(m, dim=1, rtype=cv2.REDUCE_MIN)

print("[m1] = ₩n%s₩n" %m)
print("[m_reduce_sum] =", reduce_sum.flatten())
print("[m_reduce_avg] =", reduce_avg.flatten())
print("[m_reduce_max] =", reduce_max.flatten())
print("[m_reduce_min] =", reduce_min.flatten())
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 행렬 축소 - reduce

```
[m1] =
[[46. 77. 37.  6.  0.]
 [38. 71. 41.  5. 85.]
 [85. 74. 80. 75. 21.]]

[m_reduce_sum] = [169. 222. 158.  86. 106.]
[m_reduce_avg] = [33.2 48.  67. ]
[m_reduce_max] = [85. 77. 80. 75. 85.]
[m_reduce_min] = [ 0.  5. 21.]
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ PCA 투영 및 역 투영

```
import cv2
import numpy as np

X = np.array([[0, 0,  0,100,100,150, -100,-150],
              [0,50,-50,  0, 30,100,  -20,-100]], dtype=np.float64)
X = X.transpose() # X = X.T

##mean = cv2.reduce(X, 0, cv2.REDUCE_AVG)
##print('mean = ', mean)

mean, eVects = cv2.PCACompute(X, mean=None)
print('mean = ', mean)
print('eVects = ', eVects)

Y =cv2.PCAProject(X, mean, eVects)
print('Y = ', Y)

X2 =cv2.PCABackProject(Y, mean, eVects)
print('X2 = ', X2)
print(np.allclose(X, X2))
```

Open CV 배열 연산

- 수학 및 통계 함수
  - ✓ PCA 투영 및 역 투영

```
mean =  [[12.5   1.25]]
eVects =  [[ 0.88390424  0.46766793]
 [-0.46766793  0.88390424]]
Y =  [[ -11.63338792    4.74096885]
 [  11.75000868   48.93618085]
 [ -35.01678451  -39.45424315]
 [  76.75703609  -42.02582434]
 [  90.78707404  -15.50869713]
 [ 167.71904127   22.98120308]
 [-109.37717055   33.82967723]
 [-190.9858171   -13.49926538]]
X2 =  [[ 1.77635684e-15  0.00000000e+00]
 [ 3.55271368e-15  5.00000000e+01]
 [ 0.00000000e+00 -5.00000000e+01]
 [ 1.00000000e+02 -7.10542736e-15]
 [ 1.00000000e+02  3.00000000e+01]
 [ 1.50000000e+02  1.00000000e+02]
 [-1.00000000e+02 -2.00000000e+01]
 [-1.50000000e+02 -1.00000000e+02]]
True
```

Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ PCA 투영 및 역 투영
    - mean, eVects = cv2.PCACompute(X, mean = None)는 X의 평균 벡터 mean, 공분산 행렬의 고유벡터 eVects를 계산
    - Y =cv2.PCAProject(X, mean, eVects)는 cv2. PCAProject() 함수는 고유 벡터 eVects에 의해 PCA 투영
    - 데이터를 고유 벡터를 축으로 한 좌표로 변환
    - 아래 수식에 A는 직교 행렬로 eVects이고 m은 평균 벡터이며 x, y, m은 2 X 1 열벡터
    - x는 X의 각 행에 저장된 좌표를 열벡터 변환
    - Y는 수식의 열벡터 y를 각 행에 저장한 PCA 투영 결과

$$y = A(x - m) \quad \#PCA\ projection$$

$$A^{-1}y = A^{-1}A(x - m)$$

$$A^{T}y = (x - m) \quad \#\text{직교행렬}, A^{-1} = A^{T}$$

$$x = A^{T}y + m \quad \#PCA\ backprojection$$

Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ PCA 투영 및 역 투영
    - X2 = cv2.PCABackProject(Y, mean, eVects)
    - Y를 PCA 역투영하면 원본 X를 복구할 수 있으며 X와 X2는 오차 범위 내에서 같은 값을 갖는 np.allclose(X, X2)는True

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 사각형 크기 정렬

```
import numpy as np, cv2

def print_rects(rects):
    print("-" * 46)                                    # 라인 출력
    print("사각형 원소₩t₩t랜덤 사각형 정보₩t₩t  크기")
    print("-" * 46)
    for i, (x,y, w,h, a) in enumerate(rects):                    # 저장 데이터 출력
        print("rects[%i] = [(%3d,%3d) from (%3d,%3d)] %5d" %(i, x, y, w, h, a))
    print()

rands = np.zeros((5, 5), np.uint16)                        # 5행 4열 행렬 생성
starts = cv2.randn(rands[:, :2 ], 100, 50)                 # 0~4행까지 시작좌표 랜
덤 생성
ends = cv2.randn(rands[:, 2:-1], 300, 50)                  # 5~9행까지 종료좌표 랜
덤 생성
```

Open CV 배열 연산

- ❖ 수학 및 통계 함수
  - ✓ 사각형 크기 정렬

```
sizes = cv2.absdiff(starts, ends)    # 시작 좌표 와 종료 좌표 간 차분 절대값
areas = sizes[:, 0] * sizes[:, 1]
rects = rands.copy()
rects[:, 2:-1] = sizes
rects[:,-1] = areas

idx = cv2.sortIdx(areas, cv2.SORT_EVERY_COLUMN).flatten()
# idx = np.argsort(areas, axis=0)

print_rects(rects)
print_rects(rects[idx.astype('int')])
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 사각형 크기 정렬

```
## 리스트 생성 방식
# rects = ["[(%3d,%3d) from (%3d,%3d)]" %(p[0], p[1], s[0], s[1])
#           for p, s in zip(starts, sizes)]              # 시작좌표와 크기로 리스트 생성
# areas = [s[0]*s[1] for s in sizes]                     # 넓이 계산 및 리스트에 저장
#
# # 정렬 후, 정렬 원소의 원본 좌표 반환
# sort_idx = cv2.sortIdx(np.array(areas), cv2.SORT_EVERY_COLUMN)
# sort_idx = map(int , sort_idx)
#
# print("-" * 46)                                        # 라인 출력
# print("사각형 원소₩t₩t랜덤 사각형 정보₩t  크기")
# print("-" * 46)
# for i, rect, area in zip(range(5), rects, areas):              # 저장 데이터 출력
#     print("rects["+ str(i) + "] =", rect, area)
#
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 사각형 크기 정렬

```
# print()
# print("-" * 46)
# print("사각형 원소₩t₩t정렬 사각형 정보₩t  크기")
# print("-" * 46)
# for idx in sort_idx:
                                        # 정렬 데이터 출력
#     print("rects[" + str(idx) + "] =", rects[idx], areas[idx])
```

Open CV 배열 연산

❖ 수학 및 통계 함수

✓ 사각형 크기 정렬

```
-------------------------------------------
사각형 원소                    랜덤 사각형 정보                    크기
-------------------------------------------
rects[0] = [(100,108) from (238,190)] 45220
rects[1] = [( 65, 79) from (300,123)] 36900
rects[2] = [(162, 86) from ( 43,187)]  8041
rects[3] = [( 68,115) from (195,280)] 54600
rects[4] = [(126, 43) from (188,188)] 35344

-------------------------------------------
사각형 원소                    랜덤 사각형 정보                    크기
-------------------------------------------
rects[0] = [(162, 86) from ( 43,187)]  8041
rects[1] = [(126, 43) from (188,188)] 35344
rects[2] = [( 65, 79) from (300,123)] 36900
rects[3] = [(100,108) from (238,190)] 45220
rects[4] = [( 68,115) from (195,280)] 54600
```

Open CV 배열 연산

- ❖ 행렬 연산
  - ✓ 행렬의 곱

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

    - 수식: dst = alpha • $srci1^T$ • src2 + beta • $src3^T$
    - 파라미터
      - ◆ src1, src2: 행렬 곱을 위한 두 입력 행렬(np.float32/np.float64형 2채널까지 가능)
      - ◆ alpha: 행렬 곱에 대한 가중치
      - ◆ src3: 행렬 곱에 더해지는 델타 행렬
      - ◆ beta: src3 행렬에 곱해지는 가중치
      - ◆ dst: 출력 행렬
      - ◆ flags: 옵션을 조합하여 입력 행렬들을 전치

        cv2.GEMM_1_T -> 1, src1을 전치

        cv2.GEMM_2_T -> 2, src2을 전치

        cv2.GEMM_3_T -> 4, src3을 전치

Open CV 배열 연산

- 행렬 연산
  - ✓ 행렬 변환: 행렬에 투영 변환을 수행

    cv2.perspectiveTransform(src, m[, dst]) -> dst
    - 파라미터
      - src: 입력 행렬(2채널 이나 3채널 부동소수점 배열)
      - dst: 출력 행렬
      - m: 투영할 3 X 3, 4 X 4 배열

Open CV 배열 연산

- ❖ 행렬 연산
  - ✓ 역 행렬

    cv2.invert(src[, dst[, flags]) -> retval, dst

    - 파라미터
      - ◆ src: 입력 행렬
      - ◆ dst: 출력 행렬
      - ◆ flags: 역행렬 계산 방법

        cv2.DECOMP_LU(0) -> 가우시안 소거법 이용

        cv2.DECOMP_SVD(1) -> 특이값 분해 이용

        cv2.DECOMP_CHOLESKY(2) -> CHOLESKY 분해 이용

Open CV 배열 연산

- ❖ 행렬 연산
  - ✓ 연립 방정식 풀이

    cv2.solve(src1, src2[, dst[, flags]) -> retval, dst

    - ● 파라미터
      - ◆ src1: 연립 방정식의 계수 행렬
      - ◆ src2: 연립 방정식의 상수 행렬
      - ◆ dst: 출력 행렬
      - ◆ flags: 역행렬 계산 방법

        cv2.DECOMP_LU(0) -> 가우시안 소거법 이용

        cv2.DECOMP_SVD(1) -> 특이값 분해 이용

        cv2.DECOMP_CHOLESKY(2) -> CHOLESKY 분해 이용

Open CV 배열 연산

❖ 행렬 연산

✓ 연립 방정식 풀이

```
import numpy as np, cv2

data = [ 3, 0, 6, -3, 4, 2, -5,-1, 9]                                  # 1차원 리
스트 생성
m1 = np.array(data, np.float32).reshape(3,3)
m2 = np.array([36, 10, 28], np.float32)

ret, inv = cv2.invert(m1, cv2.DECOMP_LU)                  # 역행렬 계산
if ret:
    dst1 = inv.dot(m2)                                  # numpy 제공 행렬곱 함수
    dst2 = cv2.gemm(inv, m2, 1, None, 1)                  # OpenC 제공 행렬곱 함수
    ret, dst3 = cv2.solve(m1, m2, cv2.DECOMP_LU)          # 연립방정식 풀이

    print("[inv] = ₩n%s₩n" % inv)
    print("[dst1] =", dst1.flatten())                   # 다행 1열 행렬을 한행에 표시
    print("[dst2] =", dst2.flatten())                   # 행렬을 벡터로 변환
    print("[dst3] =", dst3.flatten())                   # 행렬을 벡터로 변환
else:
    print("역행렬이 존재하지 않습니다.")
```

Open CV 배열 연산

- 행렬 연산
  - ✓ 연립 방정식 풀이

```
[inv] =
[[ 0.15079366 -0.02380952 -0.0952381 ]
 [ 0.06746032  0.22619048 -0.0952381 ]
 [ 0.09126984  0.01190476  0.04761905]]

[dst1] = [2.5238097 2.0238097 4.7380953]
[dst2] = [2.5238097 2.0238097 4.7380953]
[dst3] = [2.5238094 2.0238094 4.7380953]
```