순천향대학교
SOON CHUN HYANG UNIVERSITY

2024년 2학기

선형대수

6장

Chapter 06

- SECTION 06-1 다변량 데이터 공분산 행렬
- SECTION 06-2 행렬-벡터 곱셈을 통한 기하학적 변환
- SECTION 06-3 이미지 특징 탐지

SCH

SECTION 06-1 다변량 데이터 공분산 행렬(1)

공분산(covariance)

- 공분산은 상관 계수를 구하는 공식에서 분자 부분. 즉, 두 평균중심화된 변수 사이의 내적
- 공분산은 데이터의 크기가 반영되므로 크기 제한(-1~1)이 없음
- 상관관계는 변수가 함께 이동할 때 양수, 변수가 따로 이동할 때 음수, 변수 사이에 선형 관계가 없을 때 0과 같은 방식으로 해석할 수 있음
- 공분산의 정규화 인자는 n=1이며 n은 데이터 점의 개수
  - 이 정규화를 통해 더 많은 데이터값을 합할 때마다 공분산이 커지는 것을 방지

$$c_{a,b} = (n-1)^{-1} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

피어슨 상관계수 공식: $$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

SECTION 06-1 다변량 데이터 공분산 행렬(2)

- $\tilde{x}$ 를 **x** 의 평균중심화된 변수라고 하면 공분산은 $\tilde{x}^{T}\tilde{y}/(\mathbf{n-1})$
- 다중 변수에 대해 이 공식을 구현할 때, 행렬 곱셈이 왼쪽 행렬의 행과 오른쪽 행렬의 열 사이의 내적들로 이루어진 집합이라는 사실을 이용
  - 먼저 각 변수를 열에 담은 행렬을 생성(변수는 데이터 특징)
    - 이 행렬을 **X** 라고 부르고, 이때 **XX** 곱셈은 의미가 없음
      (그리고 데이터 행렬이 보통 높기 때문에($M>N$) 성립하지 않음)
  - 만약 첫 번째 행렬을 전치한다면 전치한 $\mathbf{X}^{T}$ 행렬의 행은 **X** 행렬의 열이 됨
    - 따라서 행렬 곱셈 $\mathbf{X}^{T}\mathbf{X}$ 는 모든 열과 열 사이의 공분산이 됨
    - 열이 평균중심화 되었다고 가정하고 n=1 로 나눈다면, 다시 말해 공분산 행렬의 (i, j)번째 원소는 데이터 특징 i와 j사이의 내적

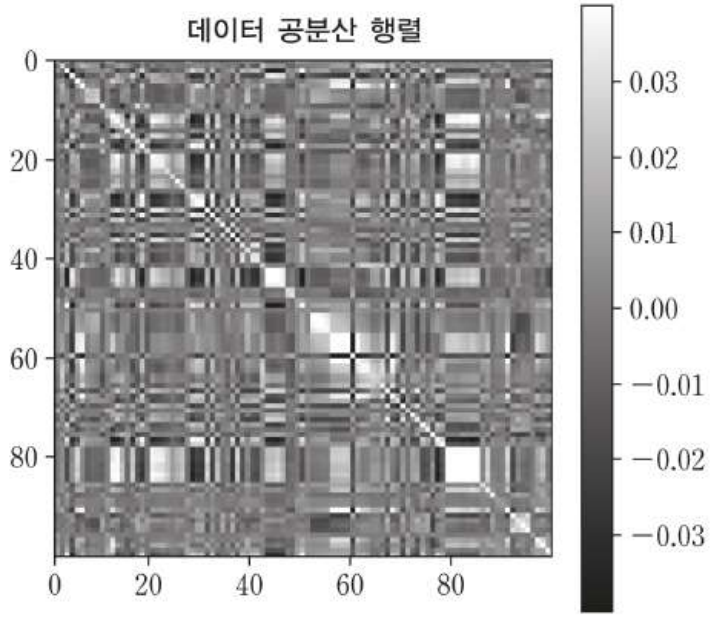
$$\mathrm{C} = \mathrm{X}^{\mathrm{T}}\mathrm{X}\frac{1}{n-1}$$

  - 공분산 행렬에 대한 행렬 방정식
    - 행렬 **C** 는 대칭 (어떤 행렬이든 자신의 전치 행렬을 곱하면 정방 대칭)
    - **C** 의 대각 원소는 각 변수의 자기 자신에 대한 공분산이며 통계에서는 이를 분산(variance)이라고 함
      - 평균 주변에 흩어진 정도를 정량화한 것 (분산은 표준 편차의 제곱)

SECTION 06-1 다변량 데이터 공분산 행렬(3)

- 공분산 행렬을 계산하는 코드

```
datamean = np.mean(dataMat,axis=0) # 특징 평균 벡터
dataMatM = dataMat - datamean # 브로드캐스트를 이용한 평균중앙화
covMat = dataMatM.T @ dataMatM # 데이터 행렬의 전치와 데이터 행렬의 곱
covMat /= (dataMatM.shape[0]-1) # N-1로 나누기
```



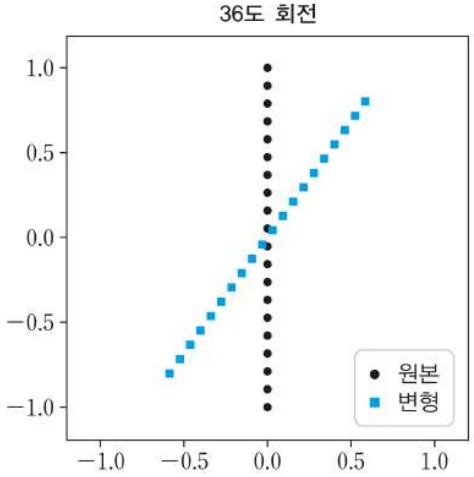
- 밝은 색은 변수 사이에 양의 상관
  (예를 들어 이혼한 남성의 비율 대 빈곤층의 수)
- 어두운 색은 변수 사이에 음의 상관
  (예를 들어 이혼한 남성의 비율 대 중위 소득)
- 회색은 변수 사이에 관련이 없는 것을 나타냄

공분산: np.cov()
상관 행렬: np.corrcoef()

SECTION 06-2 행렬-벡터 곱셈을 통한 기하학적 변환(1)

■ 순수 회전 행렬

- 길이를 유지하면서 벡터를 회전시킴
- 순수 회전 행렬은 기본적으로 직교 행렬
- 변환 행렬을 사용할 때는 θ를 특정 시계 방향 회전 각도로 설정한 다음, 행렬 **T**에 2x*N* 의 기하학적 점 행렬을 곱
  - 여기서 행렬의 각 열은 *N*개의 데이터 점에 대한 (X,Y) 좌표



- θ=π/5로 설정

순수 회전 행렬을 통해 원점을 중심으로 점을 회전

SECTION 06-2 행렬-벡터 곱셈을 통한 기하학적 변환(1)

■ 순수 회전 행렬

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

th = np.pi/5 #
#th = np.deg2rad(36)

T = np.array([[ np.cos(th),np.sin(th)], [np.sin(th),np.cos(th)]]) #cos(), sin(): radians

x = np.linspace(-1,1,20)
origPoints = np.vstack( (np.zeros(x.shape),x) )
transformedPoints = T @ origPoints

plt.figure(figsize=(6,6))
plt.plot(origPoints[0,:],origPoints[1,:],'ko',label='Original')
plt.plot(transformedPoints[0,:],transformedPoints[1,:],'s',color=[0,0,1],label='Transformed')

plt.axis('square') #축의 길이 동일
plt.xlim([-1.2,1.2])
plt.ylim([-1.2,1.2])
plt.legend()
plt.title(f'Rotation by {np.rad2deg(th):.0f} degrees.')
plt.show()
```

SECTION 06-2 행렬-벡터 곱셈을 통한 기하학적 변환(2)

- '불순한' 회전(즉 회전하면서 동시에 크기 조정)을 사용하고 변환에 동영상을 적용
  - 동영상의 각 프레임마다 변환 행렬을 부드럽게 조정
    - 동영상의 각 프레임에 들어가는 장면의 내용을 만드는 파이썬 함수를 정의한 다음 matplotlib 루틴을 호출해서 각 반복마다 해당 함수를 실행
  - 영상의 제목을 흔들리는 원(The Wobbly Circle)이라고 가칭
    - 원은 cos(θ)와 sin(θ)의 점들로 이루어져 있으며 θ각의 범위는 0~2π
  - 변환 행렬은 다음과 같이 설정

$$\mathrm{T} = \begin{bmatrix} 1 & 1-\phi \\ 0 & 1 \end{bmatrix}$$

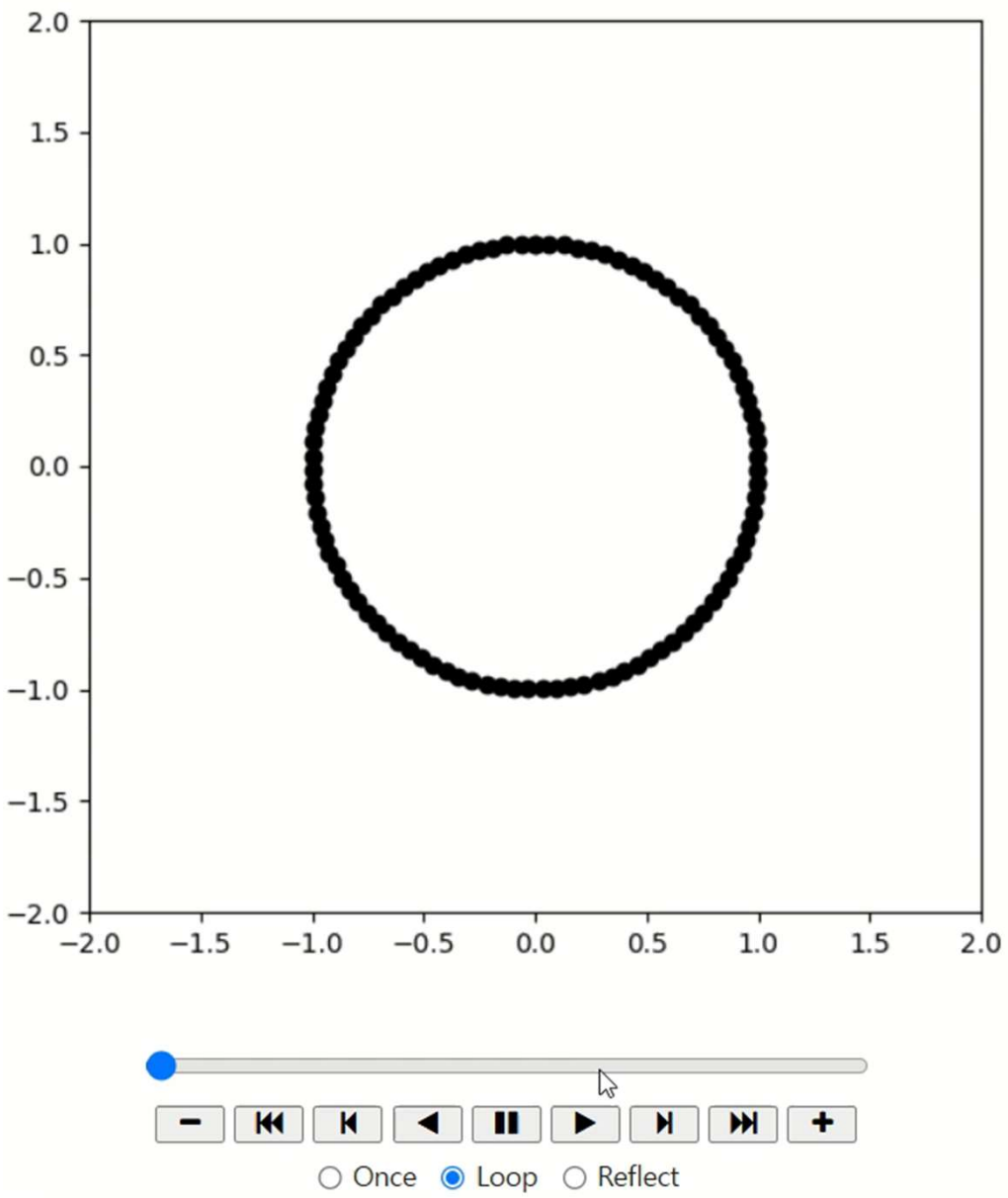
  - 영상이 진행되는 동안, Φ값은, $\Phi = x_2 - 1 \le x \le 1$ 공식을 따라 1에서 0으로 그리고 다시 1로 부드럽게 전환
    - Φ=1이면 **T**=**I** 임을 명심

SECTION 06-2 행렬-벡터 곱셈을 통한 기하학적 변환(3)

```
import matplotlib.animation as animation
from matplotlib import animation, rc
from IPython.display import HTML

# function to update the axis on each iteration
def aframe(ph):

  # create the transformation matrix
  T = np.array([[1, 1-ph], [0, 1]])

  # apply the transformation to the points using
matrix multiplication
  P = T@points

  # update the dots
  plth.set_xdata(P[0,:])
  plth.set_ydata(P[1,:])

  # export the plot handles
  return plth
```

```
# define XY points
theta  = np.linspace(0,2*np.pi,100)
points = np.vstack((np.sin(theta),np.cos(theta)))

# setup figure
fig,ax = plt.subplots(1,figsize=(12,6))
plth,  = ax.plot(np.cos(x),np.sin(x),'ko')
ax.set_aspect('equal')
ax.set_xlim([-2,2])
ax.set_ylim([-2,2])

# define values for transformation (note: clip off
the final point for a smooth animation loop)
phi = np.linspace(-1,1-1/40,40)**2
print(phi)

# run animation!
rc('animation', html='jshtml')
#animation.FuncAnimation in colab
animation.FuncAnimation(fig, aframe, phi,
interval=100, repeat=True)
```

SECTION 06-2 행렬-벡터 곱셈을 통한 기하학적 변환(4)

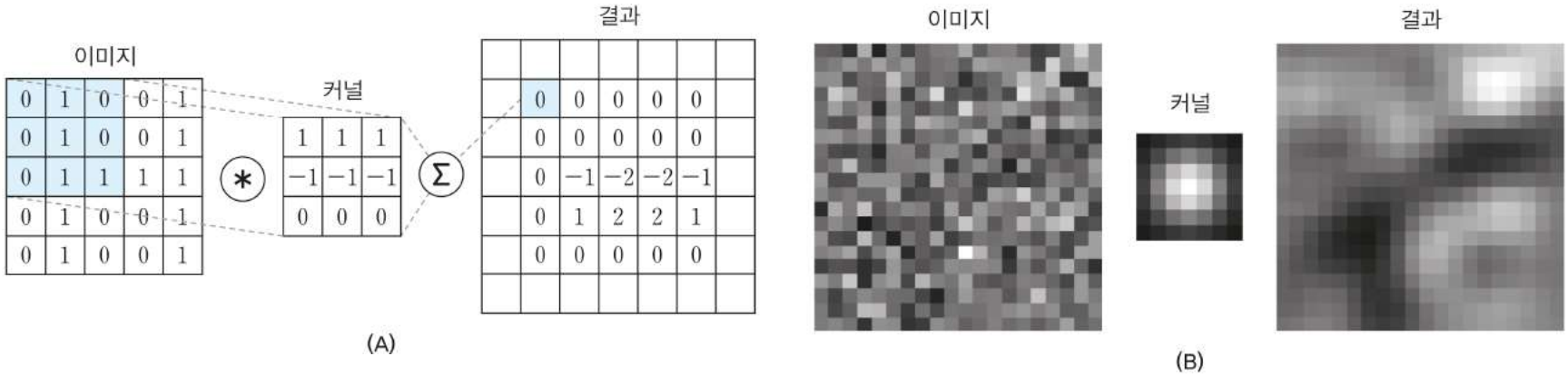


SECTION 06-3 이미지 특징 탐지(1)

■ 이미지 필터링

▪ 이미지 특징 탐지를 위한 메커니즘

- 1차원이 아닌 2차원이라는 점만 제외하고는 시계열과 동작 방식이 같음
- 2차원 커널을 설계한 다음 커널과 이미지의 겹쳐진 창 사이의 '내적'으로 구성된 새로운 이미지를 생성
  - 여기서 '내적'은 벡터-내적과 계산 자체는 원소별 곱셈과 합으로 동일하지만 연산은 두 행렬 사이에서 수행되므로 아다마르곱을 한 다음 모든 행렬 원소에 대한 합한 것과 같음



이미지 합성곱 메커니즘

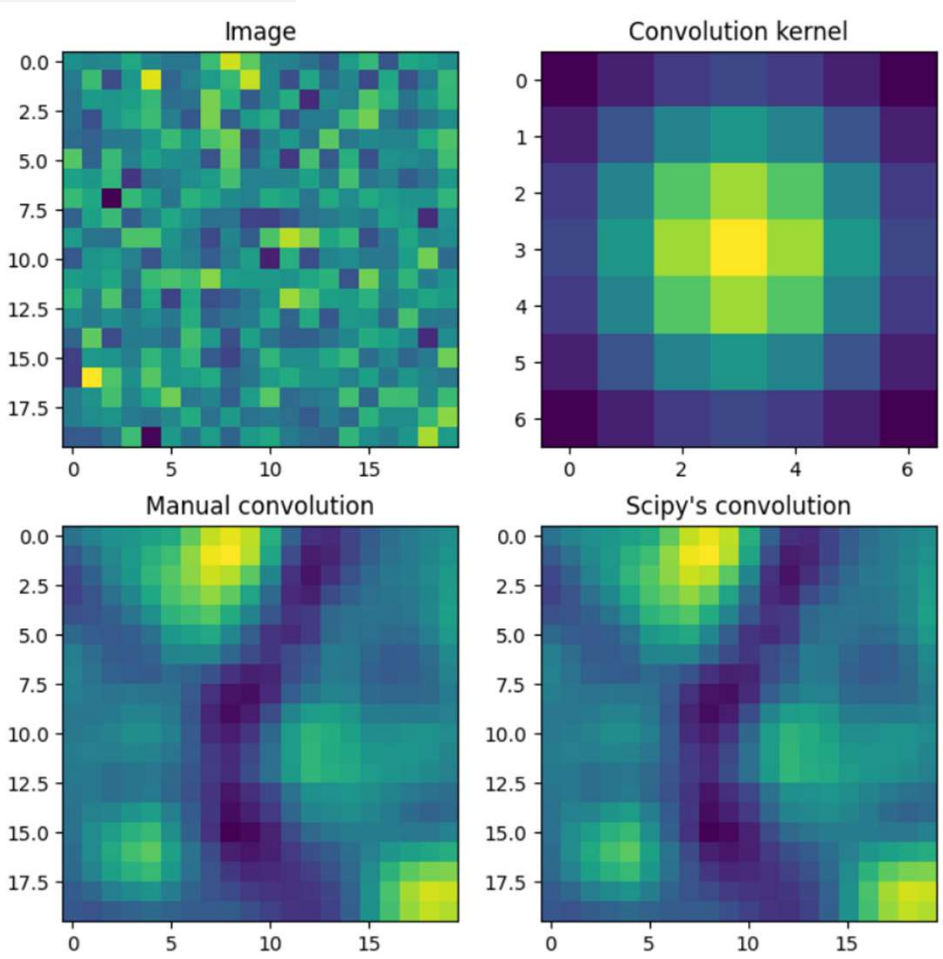
SECTION 06-3 이미지 특징 탐지(1)

```
# convolution
from scipy.signal import convolve2d

# image
imgN  = 20
image = np.random.randn(imgN,imgN)

# convolution kernel
kernelN = 7
Y,X     = np.meshgrid(np.linspace(-
3,3,kernelN),np.linspace(-3,3,kernelN))
kernel  = np.exp( -(X**2+Y**2)/kernelN )
kernel  = kernel / np.sum(kernel) # normalize

# now for the convolution
halfKr = kernelN//2
convoutput = np.zeros((imgN+kernelN-
1,imgN+kernelN-1))

imagePad = np.zeros(convoutput.shape)
imagePad[halfKr:-halfKr:1,halfKr:-halfKr:1] = image
```

```
# double for-loop over rows and columns (width
and height of picture)
for rowi in range(halfKr,imgN+halfKr):
  for coli in range(halfKr,imgN+halfKr):

    # cut out a piece of the image
    #pieceOfImg = imagePad[rowi-
halfKr:rowi+halfKr+1:1,coli-halfKr:coli+halfKr+1:1]
    pieceOfImg = imagePad[rowi-
halfKr:rowi+halfKr+1,coli-halfKr:coli+halfKr+1]

    # dot product: element-wise multiply and sum
    dotprod = np.sum( pieceOfImg*kernel )

    # store the result for this pixel
    convoutput[rowi,coli] = dotprod

# trim off edges
convoutput = convoutput[halfKr:-halfKr:1,halfKr:-
halfKr:1]
convoutput2 =
convolve2d(image,kernel,mode='same')
```

SECTION 06-3 이미지 특징 탐지(1)

```
fig,ax = plt.subplots(2,2,figsize=(8,8))

ax[0,0].imshow(image)
ax[0,0].set_title('Image')

ax[0,1].imshow(kernel)
ax[0,1].set_title('Convolution kernel')

ax[1,0].imshow(convoutput)
ax[1,0].set_title('Manual convolution')

ax[1,1].imshow(convoutput2)
ax[1,1].set_title("Scipy's convolution")

plt.savefig('Figure_06_04b.png',dpi=300)
plt.show()
```



SECTION 06-3 이미지 특징 탐지(2)

■ 2차원 가우스 커널 생성 방법

$$G = \exp\left(-\left(X^2 + Y^2\right)/\sigma\right)$$

- exp 는 자연 상수(지수 e = 2.71828...)
- exp(x)는 지수 항이 길 때 e_x대신 사용
- X와 Y는 함수를 평가할 x, y 좌표의 2차원 격자
- σ(시그마) 는 함수의 매개변수로 종종 '모양' 또는 '너비'라고 불림
  - 값이 작을수록 가우스가 좁아지는 반면 값이 클수록 가우스가 넓어짐

■ 해당 매개변수를 특정값으로 고정하고 공식을 코드로 변환

```
Y,X = np.meshgrid(np.linspace(-3,3,21),np.linspace(-3,3,21))
kernel = np.exp( -(X**2+Y**2) / 20 )
kernel = kernel / np.sum(kernel) # 정규화
```

- X와 Y격자는 21단계를 거쳐 -3에서 +3으로 이동
- 너비 매개변수는 20으로 고정
- 세 번째 줄은 전체 커널의 합이 1이 되도록 커널의 값을 정규화
  - 데이터의 원래 크기가 유지
- 적절하게 정규화되면 합성곱의 각 단계에서 필터링된 이미지의 각 픽셀은 가우스에 의해 정의된 가중치로 주변 픽셀을 가중 평균한 값이 됨

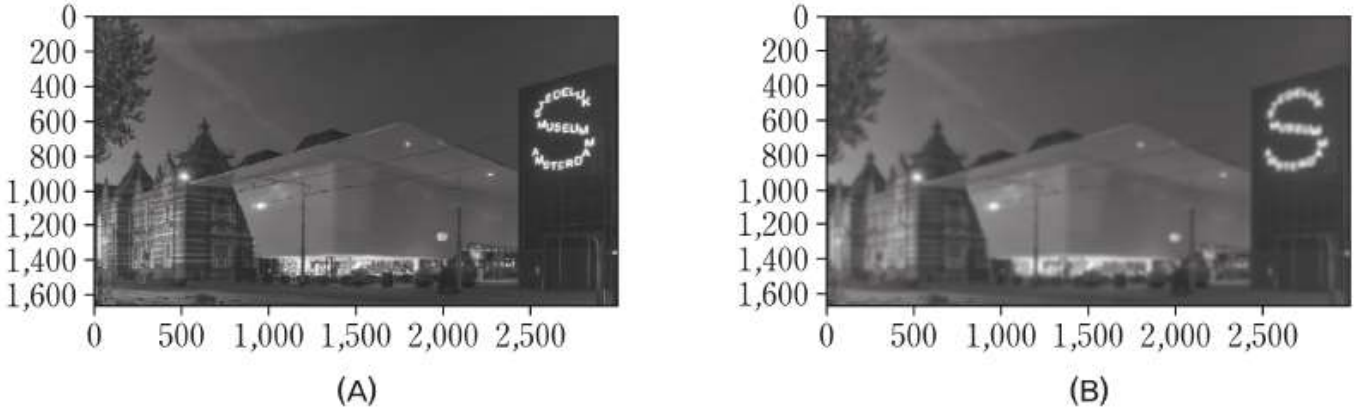
SECTION 06-3 이미지 특징 탐지(3)

- 이미지 합성곱을 파이썬으로 구현한 코드
  - 추가적인 차원 처리를 위해 for 문이 더 필요
    - for 문 대신에 SciPy의 convolve2d 함수가 유용함

```
for rowi in range(halfKr,imgN-halfKr): # 행에 대한 for 루프
  for coli in range(halfKr,imgN-halfKr): # 열에 대한 for 루프

    # 이미지 조각 자르기
    pieceOfImg = imagePad[ rowi-halfKr:rowi+halfKr+1:1,
                           coli-halfKr:coli+halfKr+1:1 ]

    # 내적: 아다마르곱과 합
    dotprod = np.sum( pieceOfImg*kernel )

    # 이 픽셀에 대한 결과를 저장
    convoutput[rowi,coli] = dotprod
```

SECTION 06-3 이미지 특징 탐지(4)

- 실제 사진을 매끄럽게 만들기
  - (예시)암스테르담에 있는 스테델리크 박물관의 사진을 사용
    - 이 사진은 행, 열, 깊이(빨강, 녹색, 파랑 색상 채널의 픽셀 강도의 값을 포함)가 있으므로 3차원 행렬
    - 이 그림은 $\mathbb{R}^{1675 \times 3000 \times 3}$의 행렬에 저장
    - 공식적으로 이것은 숫자의 표가 아니라 입방체이기 때문에 텐서(tensor)라고 불림
  - 그레이스케일로 변환해 행렬을 2차원으로 축소



평활화 전과 후의 욕조 박물관 사진

SECTION 06-3 이미지 특징 탐지(4)

```
from skimage import io,color
# read a pic from the web
bathtub =
io.imread('https://upload.wikimedia.org/wikipedia/commons/6/61/De_nieuwe_vleugel_van_het_Stedelijk_Museum_Amsterdam.jpg')

# check the size
print(bathtub.shape)

# let's see what the famous Bathtub Museum looks like
fig = plt.figure(figsize=(10,6))
plt.imshow(bathtub)
plt.savefig('Figure_06_05a.png',dpi=300)
plt.show()

# transform image to 2D for convenience (not necessary for convolution!)
bathtub2d = color.rgb2gray(bathtub)

# check the size again
print(bathtub2d.shape)
```

```
# convolution kernel
kernelN = 29

Y,X      = np.meshgrid(np.linspace(-3,3,kernelN),np.linspace(-3,3,kernelN))
kernel   = np.exp( -(X**2+Y**2)/20 )
kernel   = kernel / np.sum(kernel)

# smoothing via Gaussian convolution
smooth_bathtub =
convolve2d(bathtub2d,kernel,mode='same')

fig = plt.figure(figsize=(10,6))
plt.imshow(smooth_bathtub,cmap='gray')
plt.savefig('Figure_06_05b.png',dpi=300)
plt.show()
```