영상처리프로그래밍
영상 필터링: 에지 검출

한림대학교 소프트웨어융합대학
박섭형

2022년 1학기

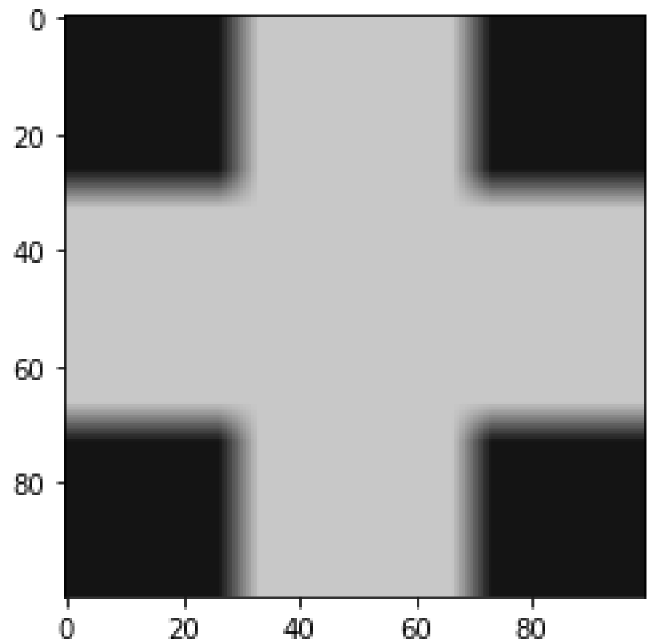
0.1 영상의 에지 검출

- 에지 (Edge)
  - 어떤 물체의 윤곽선 또는 경계선
  - 영상에서 급속한 밝기 변화가 있는 곳

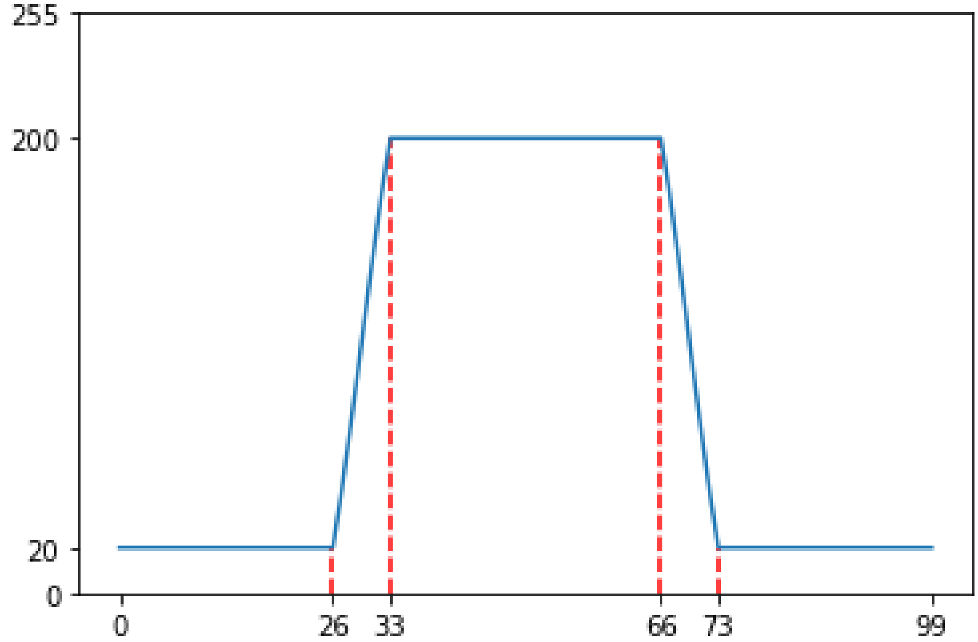
[1]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np
print("OpenCV version", cv2.__version__)
print("NumPy version", np.__version__)
```

```
OpenCV version 4.5.2
NumPy version 1.20.3
```

[4]:
```
img = np.ones((100,100), np.uint8) * 20
img[:,30:70] = 200
img[30:70,:] = 200
img = cv2.GaussianBlur(img, (7,7), 5., 5.)
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.show()
```



[5]:
```
idx1 = np.where(img[0]!=20)[0][0] - 1
idx2 = np.where(img[0]==200)[0][0]
plt.plot(range(img.shape[1]), img[0])
plt.vlines([idx1], 0, 20, colors='r', ls='--')
plt.vlines([idx2], 0, 200, colors='r', ls='--')
plt.vlines([99-idx1], 0, 20, colors='r', ls='--')
plt.vlines([99-idx2], 0, 200, colors='r', ls='--')
plt.xticks([0, idx1, idx2, 99-idx2, 99-idx1, 99])
plt.yticks([0, 20, 200, 255])
plt.ylim(0, 255)
plt.show()
```



1차 미분 마스크

- 에지
  - 화소의 밝기가 급속하게 변하는 부분
  - 함수의 기울기가 큰 부분
- 미분
  - 함수의 순간 변화율

Gradient

- 1차원 신호의 미분

$$f'(x) = \frac{df(x)}{dx} = \lim_{dx \to 0} \frac{f(x+dx) - f(x)}{dx}$$

- 2차원 디지털 영상 신호의 밝기 변화
  - 1차 편미분: $dx = dy = 1$ 인 경우

$$\frac{\partial f(x,y)}{\partial x} = \lim_{dx \to 0} \frac{f(x+dx,y) - f(x,y)}{dx} \approx f(x+1,y) - f(x,y)$$

$$\frac{\partial f(x,y)}{\partial y} = \lim_{dy \to 0} \frac{f(x,y+dy) - f(x,y)}{dy} \approx f(x,y+1) - f(x,y)$$

- 점 $p = (x,y)$에서 f의 gradient ∇f

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- Gradient의 크기: $||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$
- Gradient의 방향: $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$

[6]:

```
ipp.partial_gradient_demo()
```

$f(x-2,y-2)$	$f(x-2,y-1)$	$f(x-2,y)$	$f(x-2,y+1)$	$f(x-2,y+2)$
$f(x-1,y-2)$	$f(x-1,y-1)$	$f(x-1,y)$	$f(x-1,y+1)$	$f(x-1,y+2)$
$f(x,y-2)$	$f(x,y-1)$	$f(x,y)$	$f(x,y+1)$	$f(x,y+2)$
$f(x+1,y-2)$	$f(x+1,y-1)$	$f(x+1,y)$	$f(x+1,y+1)$	$f(x+1,y+2)$
$f(x+2,y-2)$	$f(x+2,y-1)$	$f(x+2,y)$	$f(x+2,y+1)$	$f(x+2,y+2)$

0	0	0
0	−1	0
0	1	0

x 방향 커널

0	0	0
0	−1	1
0	0	0

y 방향 커널

[7]:

```
k_diff_x = np.array([[0, 0, 0],
                     [0, -1, 0],
                     [0, 1, 0]])
k_diff_y = np.array([[0, 0, 0],
```

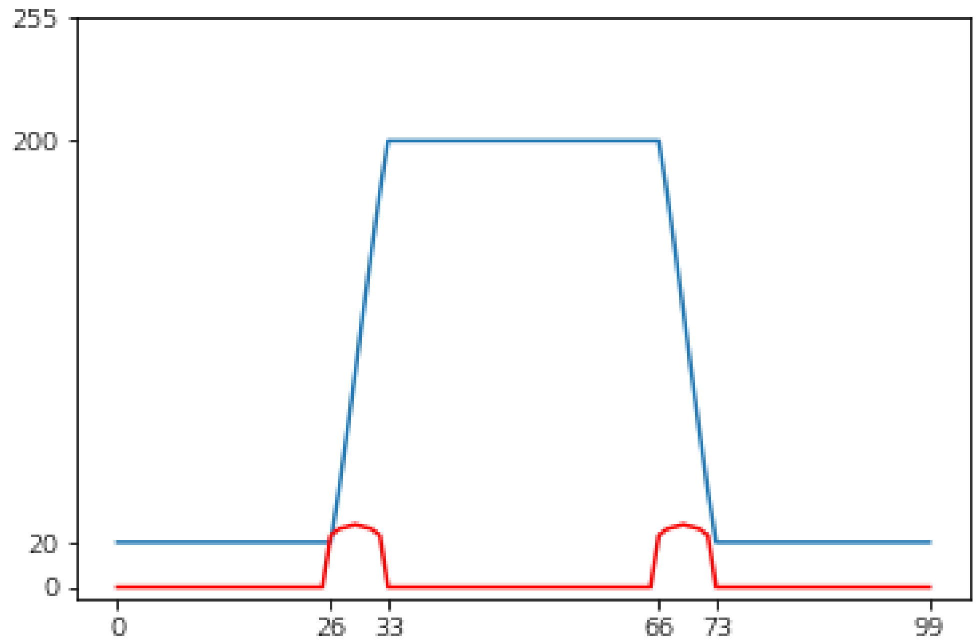
```
                [0, -1, 1],
                [0, 0, 0]])
```

[8]:

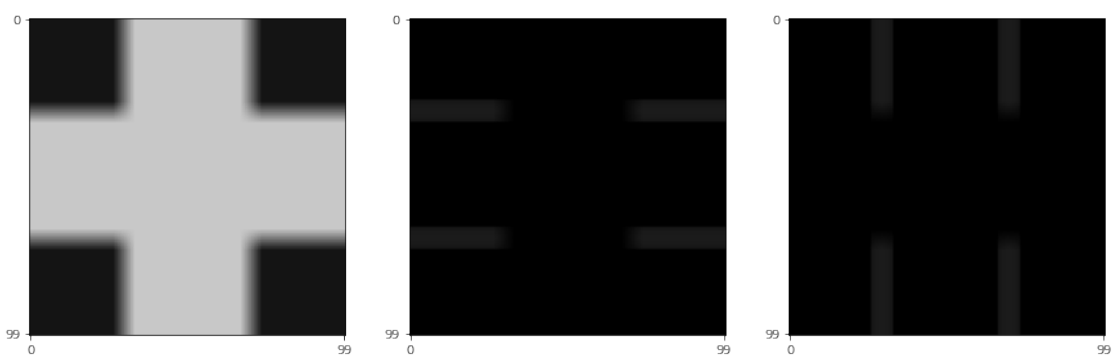
```
Gx_img = cv2.filter2D(img, cv2.CV_32F, k_diff_x)
Gy_img = cv2.filter2D(img, cv2.CV_32F, k_diff_y)

Gxabs_img = np.abs(Gx_img).clip(0,255).astype(np.uint8)
Gyabs_img = np.abs(Gy_img).clip(0,255).astype(np.uint8)
```

[9]:

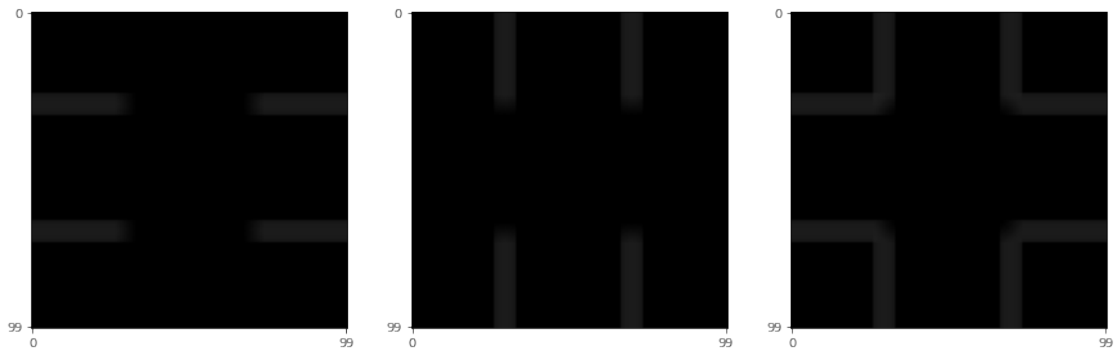
```
idx1 = np.where(img[0]!=20)[0][0] - 1
idx2 = np.where(img[0]==200)[0][0]
plt.plot(range(img.shape[1]), img[0])
#plt.vlines([idx1], 0, 20, colors='r', ls='--')
#plt.vlines([idx2], 0, 200, colors='r', ls='--')
#plt.vlines([99-idx1], 0, 20, colors='r', ls='--')
#plt.vlines([99-idx2], 0, 200, colors='r', ls='--')
plt.xticks([0, idx1, idx2, 99-idx2, 99-idx1, 99])
plt.yticks([0, 20, 200, 255])
plt.ylim(-5, 255)

plt.plot(range(img.shape[1]), Gyabs_img[0], 'r')

plt.show()
```



[10]:
```
ipp.show_images(img, Gxabs_img, Gyabs_img)
```

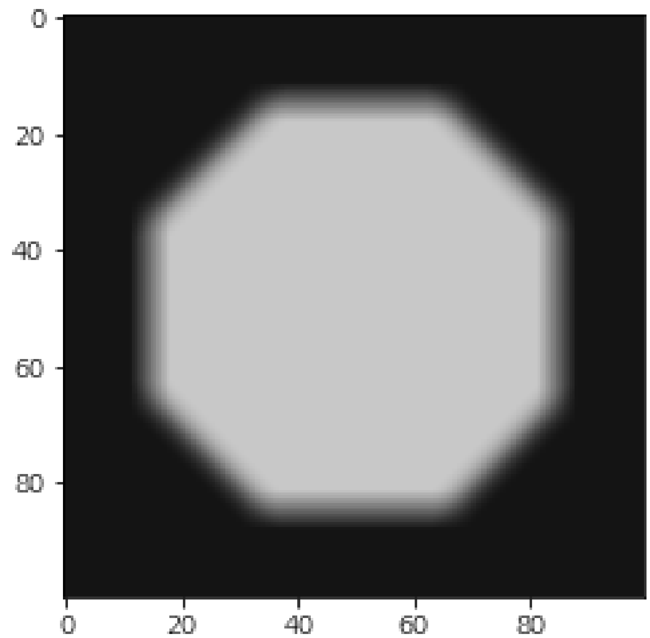


[11]:
```
G_img = np.sqrt(Gx_img**2 + Gy_img**2).astype(np.uint8)
ipp.show_images(Gxabs_img, Gyabs_img, G_img)
```



대각선 에지가 있는 경우

[12]:
```
img2 = np.ones((100,100), np.uint8) * 200
ix = np.arange(img2.shape[0]).reshape(img2.shape[0], 1)
iy = np.arange(img2.shape[1]).reshape(1, img2.shape[1])
img2[iy>ix+img2.shape[1]//2] = 20
img2[iy<ix-img2.shape[1]//2] = 20
img2[iy<-ix+img2.shape[1]//2] = 20
img2[iy>-ix+3*img2.shape[1]//2] = 20
img2[:,85:] = 20
img2[:,:15] = 20
img2[85:,:] = 20
img2[:15,:] = 20
img2 = cv2.GaussianBlur(img2, (7,7), 5., 5.)
plt.imshow(img2, cmap='gray', vmin=0, vmax=255)
plt.show()
```

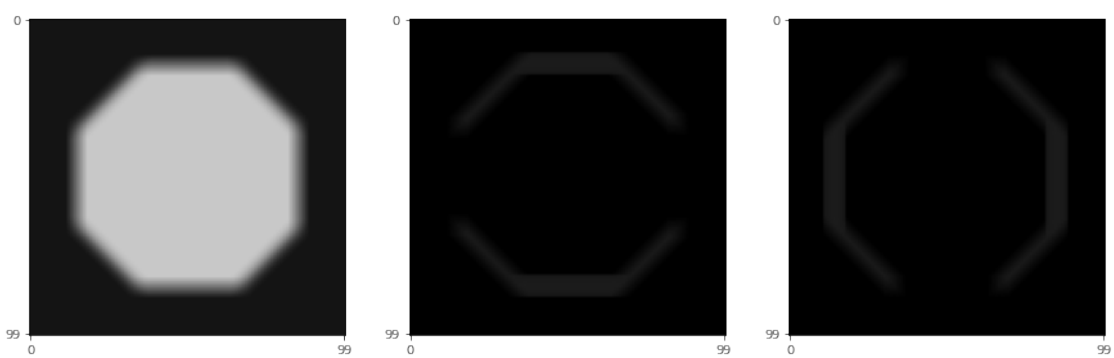


[13]:

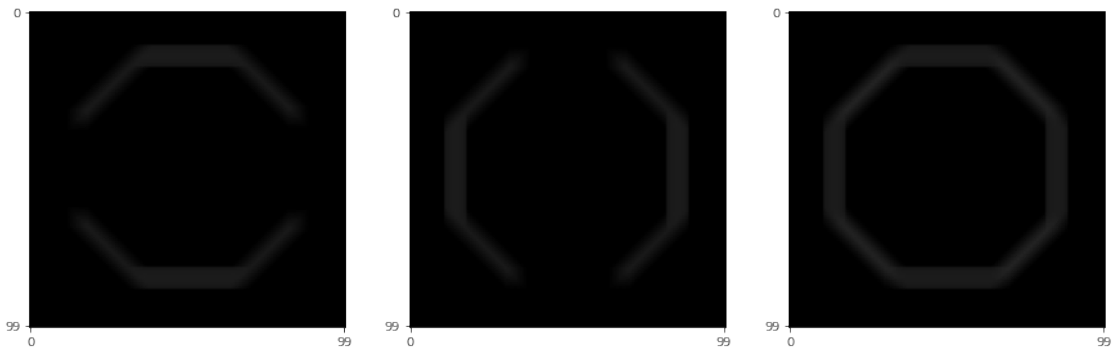
```
Gx_img2 = cv2.filter2D(img2, cv2.CV_32F, k_diff_x)
Gy_img2 = cv2.filter2D(img2, cv2.CV_32F, k_diff_y)

Gxabs_img2 = np.abs(Gx_img2).clip(0,255).astype(np.uint8)
Gyabs_img2 = np.abs(Gy_img2).clip(0,255).astype(np.uint8)
```

[14]:

```
ipp.show_images(img2, Gxabs_img2, Gyabs_img2)
```



[15]:
```
def magnitude(img1, img2):
    tmp = np.sqrt(img1.astype(float)**2 + img2.astype(float)**2)
    return tmp.clip(0, 255).astype(np.uint8)

G_img2 = magnitude(Gx_img2, Gy_img2)
ipp.show_images(Gxabs_img2, Gyabs_img2, G_img2)
```

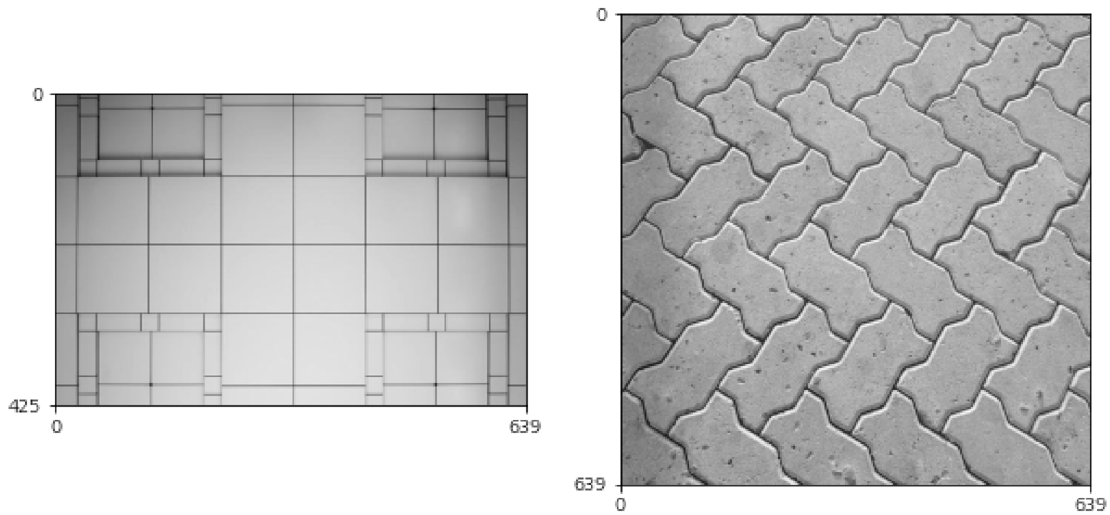


[16]:
```
pavement = cv2.imread('pavement_640.jpg')
abstract = cv2.imread('abstract_640.jpg')
```

[17]:
```
pavement_L = cv2.cvtColor(pavement, cv2.COLOR_BGR2YCrCb)[...,0]
abstract_L = cv2.cvtColor(abstract, cv2.COLOR_BGR2YCrCb)[...,0]
pavement_L.min(), pavement_L.max(), abstract_L.min(), abstract_L.max()
```

[17]:
```
(0, 255, 0, 255)
```

[18]:
```
ipp.show_images(abstract_L, pavement_L)
```

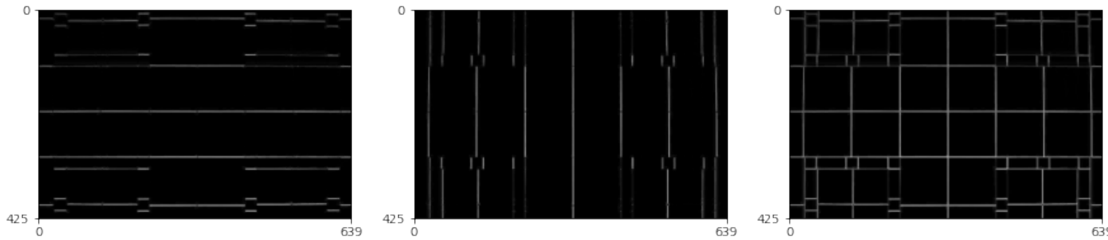


[19]:

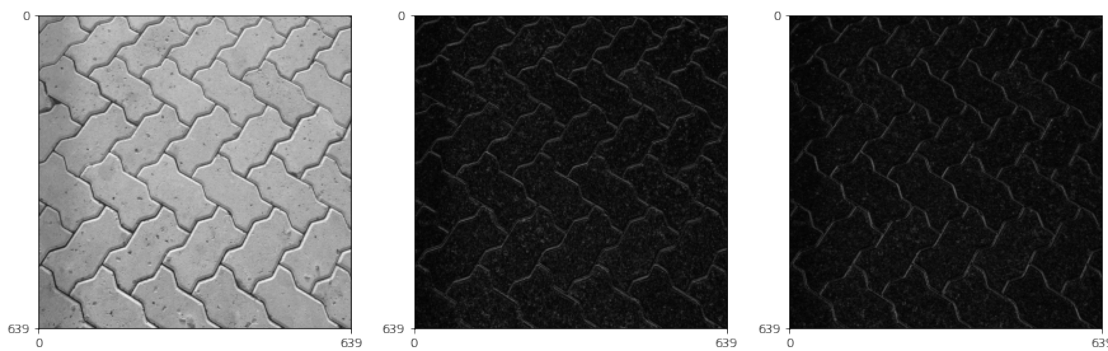
```
Gx_abstract = cv2.filter2D(abstract_L, cv2.CV_32F, k_diff_x)
Gy_abstract = cv2.filter2D(abstract_L, cv2.CV_32F, k_diff_y)

Gxabs_abstract = np.abs(Gx_abstract).clip(0,255).astype(np.uint8)
Gyabs_abstract = np.abs(Gy_abstract).clip(0,255).astype(np.uint8)
ipp.show_images(abstract_L, Gxabs_abstract, Gyabs_abstract)
```

[20]:

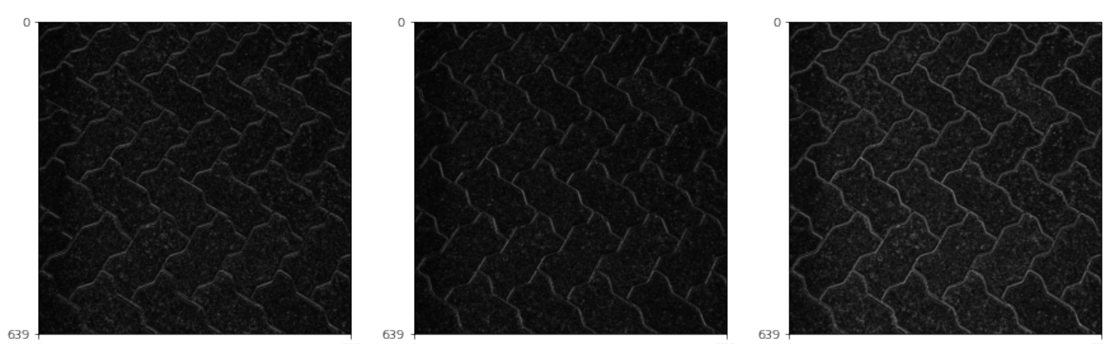
```
G_abstract = magnitude(Gx_abstract, Gy_abstract)
ipp.show_images(Gxabs_abstract, Gyabs_abstract, G_abstract)
```



[21]:
```
Gx_pavement = cv2.filter2D(pavement_L, cv2.CV_32F, k_diff_x)
Gy_pavement = cv2.filter2D(pavement_L, cv2.CV_32F, k_diff_y)

Gxabs_pavement = np.abs(Gx_pavement).clip(0,255).astype(np.uint8)
Gyabs_pavement = np.abs(Gy_pavement).clip(0,255).astype(np.uint8)
ipp.show_images(pavement_L, Gxabs_pavement, Gyabs_pavement)
```



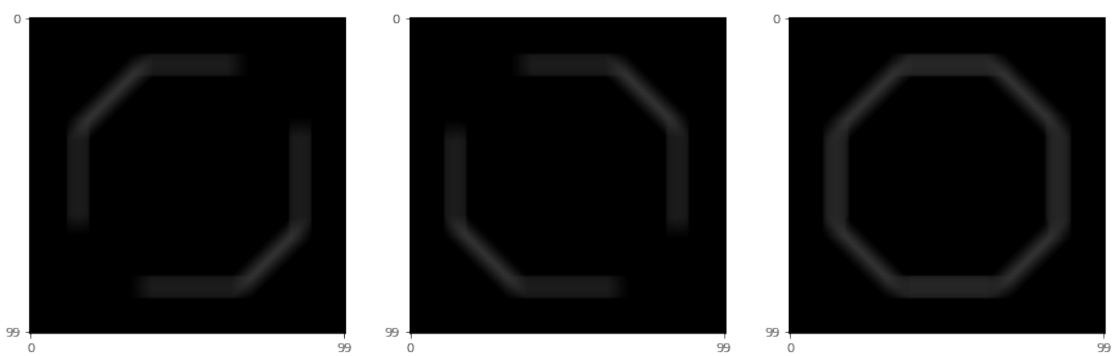
[22]:
```
G_pavement = magnitude(Gx_pavement, Gy_pavement)
ipp.show_images(Gxabs_pavement, Gyabs_pavement, G_pavement)
```



Roberts 커널

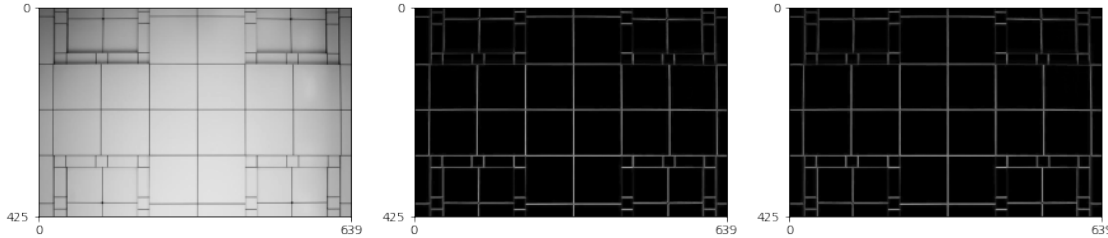
[23]:
```
k_roberts_1 = np.array([[-1, 0, 0],
                        [ 0, 1, 0],
                        [ 0, 0, 0]])
k_roberts_2 = np.array([[ 0, 0,-1],
                        [ 0, 1, 0],
                        [ 0, 0, 0]])
```

[24]:
```
G1_img2_roberts = cv2.filter2D(img2, cv2.CV_32F, k_roberts_1)
G2_img2_roberts = cv2.filter2D(img2, cv2.CV_32F, k_roberts_2)
G1_img2_roberts_abs = np.abs(G1_img2_roberts).clip(0,255).astype(np.uint8)
G2_img2_roberts_abs = np.abs(G2_img2_roberts).clip(0,255).astype(np.uint8)
G_img2_roberts = magnitude(G1_img2_roberts, G2_img2_roberts)

ipp.show_images(G1_img2_roberts_abs, G2_img2_roberts_abs, G_img2_roberts)
```

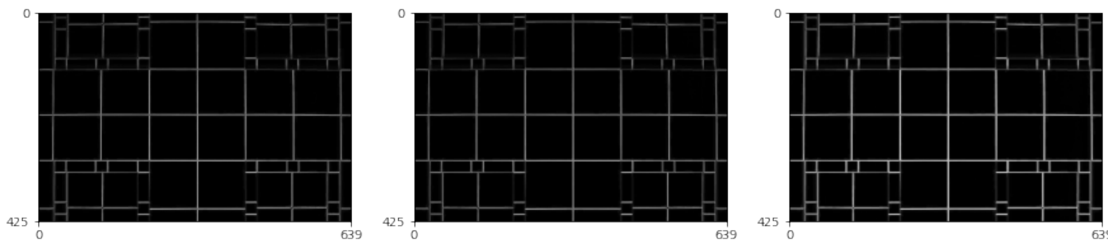


[25]:
```
G1_abstract_roberts = cv2.filter2D(abstract_L, cv2.CV_32F, k_roberts_1)
G2_abstract_roberts = cv2.filter2D(abstract_L, cv2.CV_32F, k_roberts_2)

G1_abstract_roberts_abs = np.abs(G1_abstract_roberts).clip(0,255).astype(np.
 ↪uint8)
G2_abstract_roberts_abs = np.abs(G2_abstract_roberts).clip(0,255).astype(np.
 ↪uint8)
```

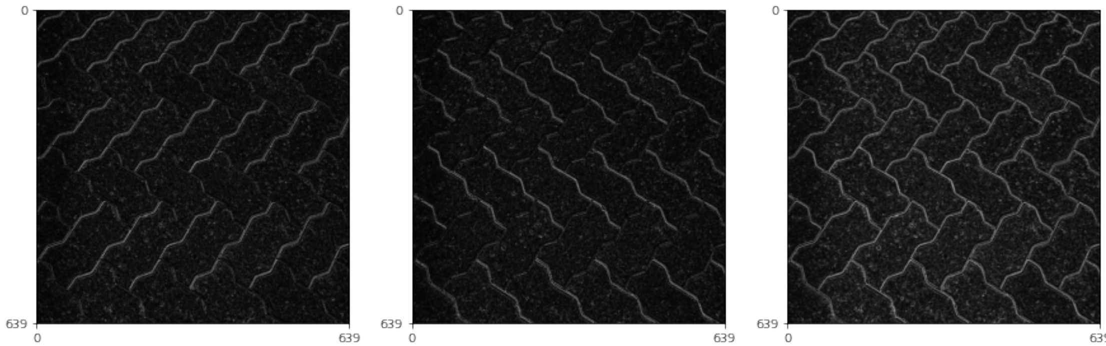
```
ipp.show_images(abstract_L, G1_abstract_roberts_abs, G2_abstract_roberts_abs)
```



[26]:
```
G_abstract_roberts = magnitude(G1_abstract_roberts, G2_abstract_roberts)
ipp.show_images(G1_abstract_roberts_abs, G2_abstract_roberts_abs, G_abstract_roberts)
```



[27]:
```
G1_pavement_roberts = cv2.filter2D(pavement_L, cv2.CV_32F, k_roberts_1)
G2_pavement_roberts = cv2.filter2D(pavement_L, cv2.CV_32F, k_roberts_2)

G1_pavement_roberts_abs = np.abs(G1_pavement_roberts).clip(0,255).astype(np.uint8)
G2_pavement_roberts_abs = np.abs(G2_pavement_roberts).clip(0,255).astype(np.uint8)
G_pavement_roberts = magnitude(G1_pavement_roberts, G2_pavement_roberts)

ipp.show_images(G1_pavement_roberts_abs, G2_pavement_roberts_abs, G_pavement_roberts)
```



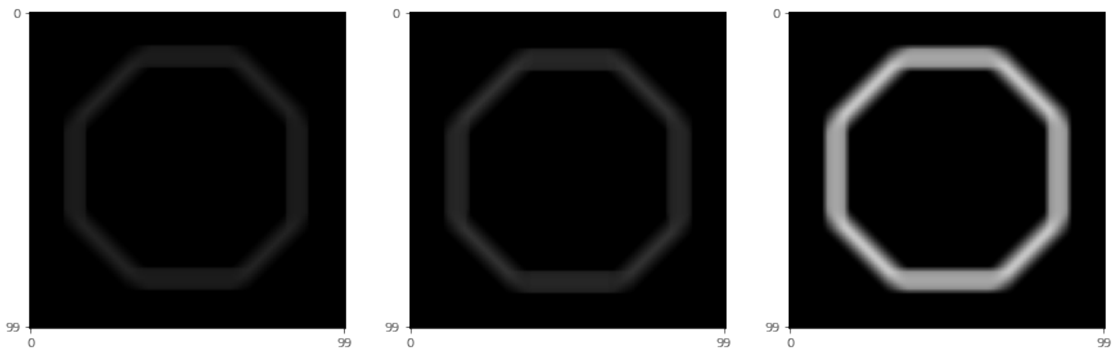
Prewitt 커널

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$k_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

[28]:
```
k_prewitt_x = np.array([[-1, 0, 1],
                        [-1, 0, 1],
                        [-1, 0, 1]])
k_prewitt_y = np.array([[-1,-1,-1],
                        [ 0, 0, 0],
                        [ 1, 1, 1]])
```

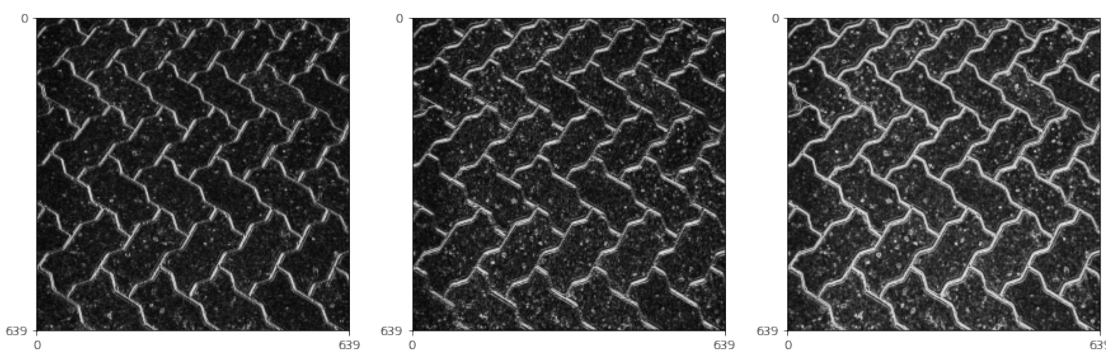
[29]:
```
Gx_img2_prewitt = cv2.filter2D(img2, cv2.CV_32F, k_prewitt_x)
Gy_img2_prewitt = cv2.filter2D(img2, cv2.CV_32F, k_prewitt_y)

Gx_img2_prewitt_abs = np.abs(Gx_img2_prewitt).clip(0,255).astype(np.uint8)
Gy_img2_prewitt_abs = np.abs(Gy_img2_prewitt).clip(0,255).astype(np.uint8)
G_img2_prewitt = magnitude(Gx_img2_prewitt, Gy_img2_prewitt)

ipp.show_images(G_img2, G_img2_roberts, G_img2_prewitt)
```



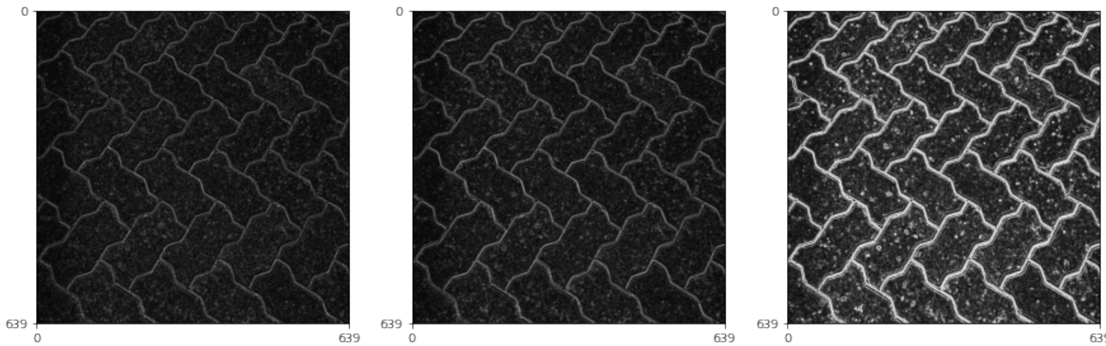
[30]:
```
Gx_img2.min(), Gx_img2.max(), G1_img2_roberts.min(), G1_img2_roberts.max(),␣
 ↪Gx_img2_prewitt.min(), Gx_img2_prewitt.max()
```

[30]: (-28.0, 28.0, -49.0, 49.0, -166.0, 166.0)

[31]:
```
Gx_pavement_prewitt = cv2.filter2D(pavement_L, cv2.CV_32F, k_prewitt_x)
Gy_pavement_prewitt = cv2.filter2D(pavement_L, cv2.CV_32F, k_prewitt_y)

Gx_pavement_prewitt_abs = np.abs(Gx_pavement_prewitt).clip(0,255).astype(np.
 ↪uint8)
Gy_pavement_prewitt_abs = np.abs(Gy_pavement_prewitt).clip(0,255).astype(np.
 ↪uint8)
G_pavement_prewitt = magnitude(Gx_pavement_prewitt, Gy_pavement_prewitt)

ipp.show_images(Gx_pavement_prewitt_abs, Gy_pavement_prewitt_abs,␣
 ↪G_pavement_prewitt)
```



[32]:
```
ipp.show_images(G_pavement, G_pavement_roberts, G_pavement_prewitt)
```



Sobel 커널

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$k_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

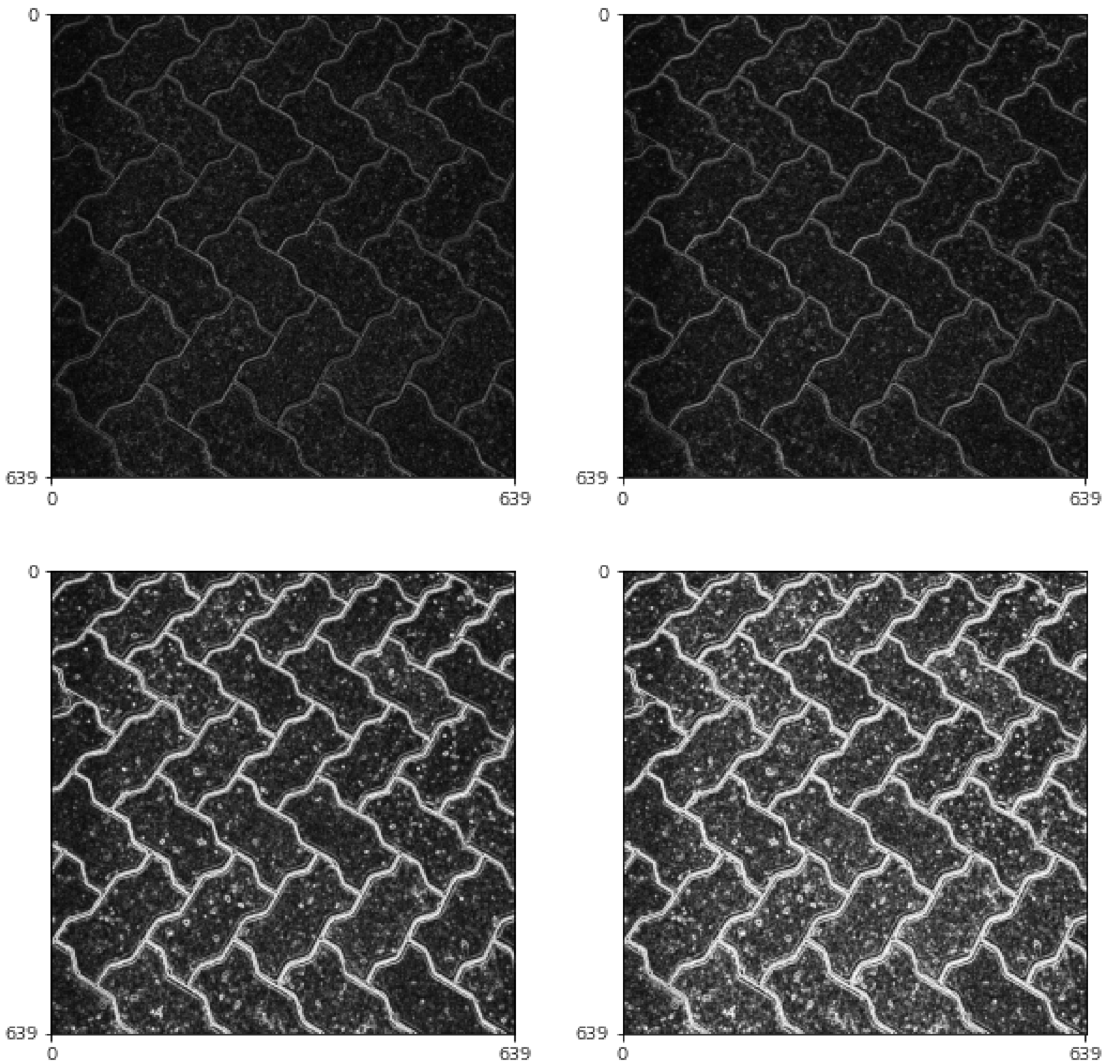
[33]:
```
k_sobel_x = np.array([
  [-1, 0, 1],
  [-2, 0, 2],
  [-1, 0, 1]
])

k_sobel_y = np.array([
  [-1, -2, -1],
  [ 0,  0,  0],
  [ 1,  2,  1]
])
```

[34]:
```
Gx_pavement_sobel = cv2.filter2D(pavement_L, cv2.CV_32F, k_sobel_x)
Gy_pavement_sobel = cv2.filter2D(pavement_L, cv2.CV_32F, k_sobel_y)
```

```python
Gx_pavement_sobel_abs = np.abs(Gx_pavement_sobel).clip(0,255).astype(np.uint8)
Gy_pavement_sobel_abs = np.abs(Gy_pavement_sobel).clip(0,255).astype(np.uint8)

G_pavement_sobel = magnitude(Gx_pavement_sobel, Gy_pavement_sobel)
```

[35]:

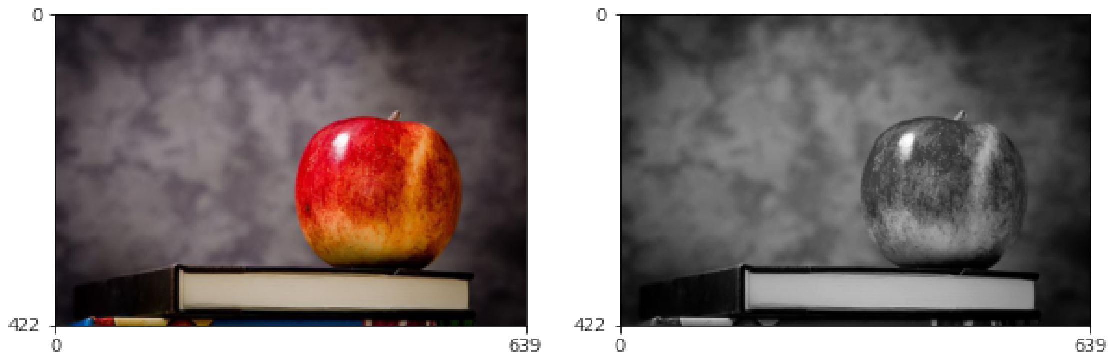
```python
ipp.show_images(G_pavement, G_pavement_roberts, G_pavement_prewitt,
 ↪G_pavement_sobel)
```



[36]:

```python
apple = cv2.imread('apple_640.jpg')
apple_L = cv2.cvtColor(apple, cv2.COLOR_BGR2GRAY)
```

```
ipp.show_images(apple[...,::-1], apple_L)
```



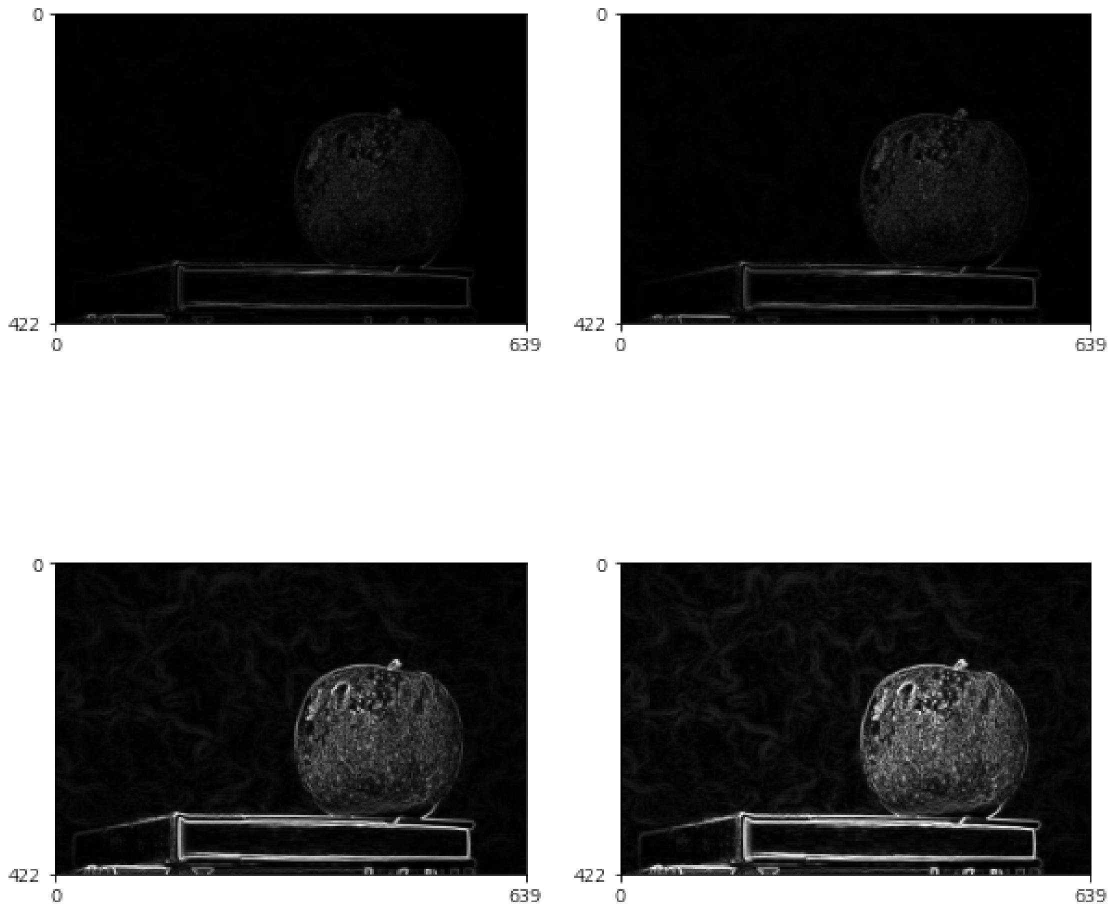
[37]:
```
Gx_apple = cv2.filter2D(apple_L, cv2.CV_32F, k_diff_x)
Gy_apple = cv2.filter2D(apple_L, cv2.CV_32F, k_diff_y)
G_apple = magnitude(Gx_apple, Gy_apple)
```

[38]:
```
G1_apple_roberts = cv2.filter2D(apple_L, cv2.CV_32F, k_roberts_1)
G2_apple_roberts = cv2.filter2D(apple_L, cv2.CV_32F, k_roberts_2)
G_apple_roberts = magnitude(G1_apple_roberts, G2_apple_roberts)
```

[39]:
```
Gx_apple_prewitt = cv2.filter2D(apple_L, cv2.CV_32F, k_prewitt_x)
Gy_apple_prewitt = cv2.filter2D(apple_L, cv2.CV_32F, k_prewitt_y)
G_apple_prewitt = magnitude(Gx_apple_prewitt, Gy_apple_prewitt)
```

[40]:
```
Gx_apple_sobel = cv2.filter2D(apple_L, cv2.CV_32F, k_sobel_x)
Gy_apple_sobel = cv2.filter2D(apple_L, cv2.CV_32F, k_sobel_y)
G_apple_sobel = magnitude(Gx_apple_sobel, Gy_apple_sobel)
```

[41]:
```
ipp.show_images(G_apple, G_apple_roberts, G_apple_prewitt, G_apple_sobel)
```



2차 미분 커널

라플라시안 (Laplacian) **연산자** 유클리드 공간 $\mathbb{R}^n$ 위의 무한 미분이 가능한 실수 값 함수 $f\colon \mathbb{R}^n \to \mathbb{R}$ 의 라플라스 연산자는 다음과 같다.

$$\Delta f(x_1, \cdots, x_n) = \nabla^2 f(x_1, \cdots, x_n) = \sum_{i=1}^{n} \frac{\partial^2 f(x_1, \cdots, x_n)}{\partial x_i^2}$$

2차원 유클리드 공간에서 라플라스 연산자는 다음과 같다.

$$\Delta f(x, y) = \nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

이산 라플라시안 (discrete Laplacian) **연산자**

- 점 $p = (x, y)$ 에서 f 의 이산 Laplacian

– 유한 차분법 (finite differences)를 사용하여 연속 라플라시안의 근사값을 다음과 같이 구할 수 있다.

$$\nabla^2 f(x,y) \approx \frac{f(x-h,y)+f(x+h,y)+f(x,y-h)+f(x,y+h)-4f(x,y)}{h^2}$$

- 여기에서 $h=1$로 두면

$$\nabla^2 f(x,y) \approx f(x-1,y)+f(x+1,y)+f(x,y-1)+f(x,y+1)-4f(x,y),$$

- 4 방향 커널:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{또는} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- 8 방향 커널:

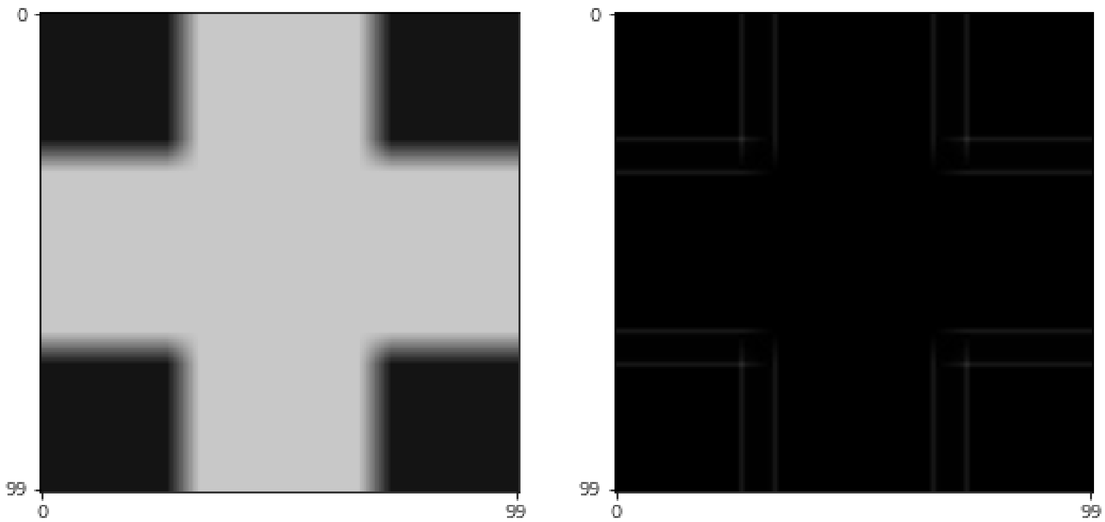
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{또는} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

[42]:
```
k_4_Laplacian_1 = np.array([[0, -1, 0],
                            [-1, 4, -1],
                            [0, -1, 0]])
```

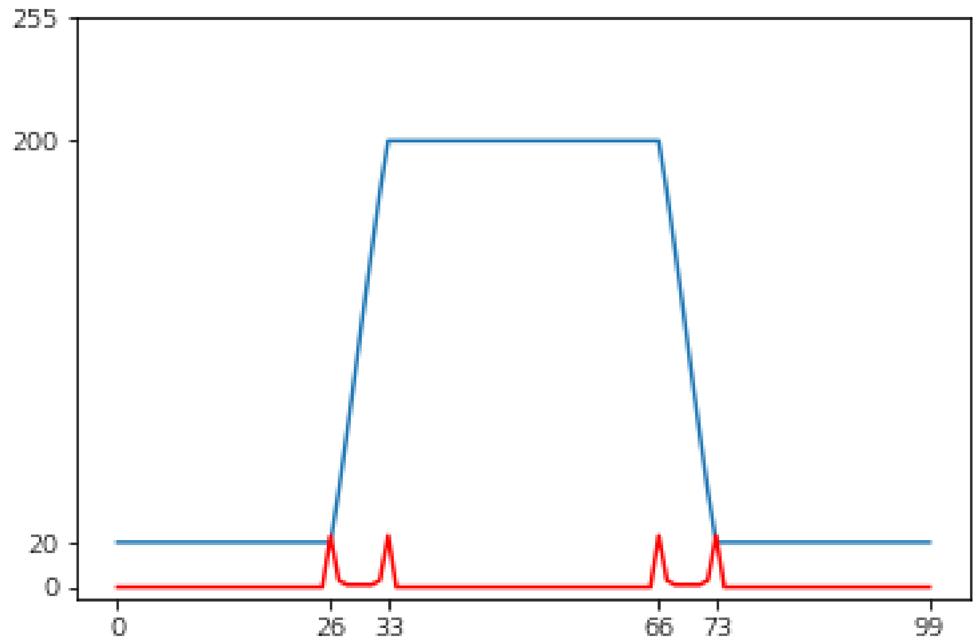
[43]:
```
img_Laplcian_4_1 = cv2.filter2D(img, cv2.CV_32F, k_4_Laplacian_1)
img_Laplcian_4_1.min(), img_Laplcian_4_1.max()
```

[43]:
```
(-46.0, 23.0)
```

[44]:
```
img_Laplcian_4_1 = np.abs(img_Laplcian_4_1).clip(0, 255).astype(np.uint8)
ipp.show_images(img, img_Laplcian_4_1)
```



[45]:
```
idx1 = np.where(img[0]!=20)[0][0] - 1
idx2 = np.where(img[0]==200)[0][0]
plt.plot(range(img.shape[1]), img[0])
plt.xticks([0, idx1, idx2, 99-idx2, 99-idx1, 99])
plt.yticks([0, 20, 200, 255])
plt.ylim(-5, 255)

plt.plot(range(img.shape[1]), img_Laplcian_4_1[0], 'r')

plt.show()
```



[46]:

```
k_8_Laplacian_1 = np.array([[-1, -1, -1],
                            [-1,  8, -1],
                            [-1, -1, -1]])
```

[47]:

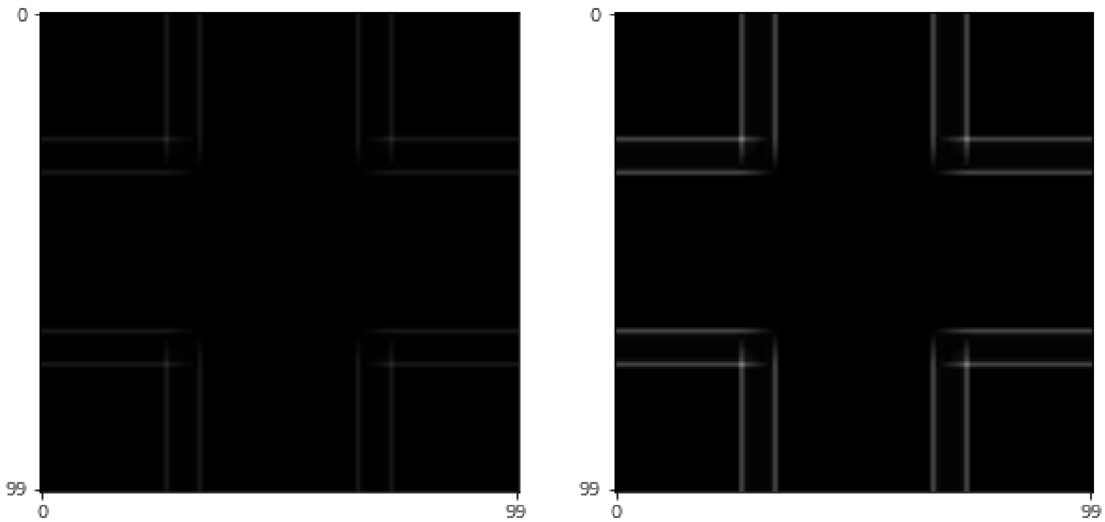
```
img_Laplcian_8_1 = cv2.filter2D(img, cv2.CV_32F, k_8_Laplacian_1)
img_Laplcian_8_1.min(), img_Laplcian_8_1.max()
```

[47]:

```
(-135.0, 69.0)
```

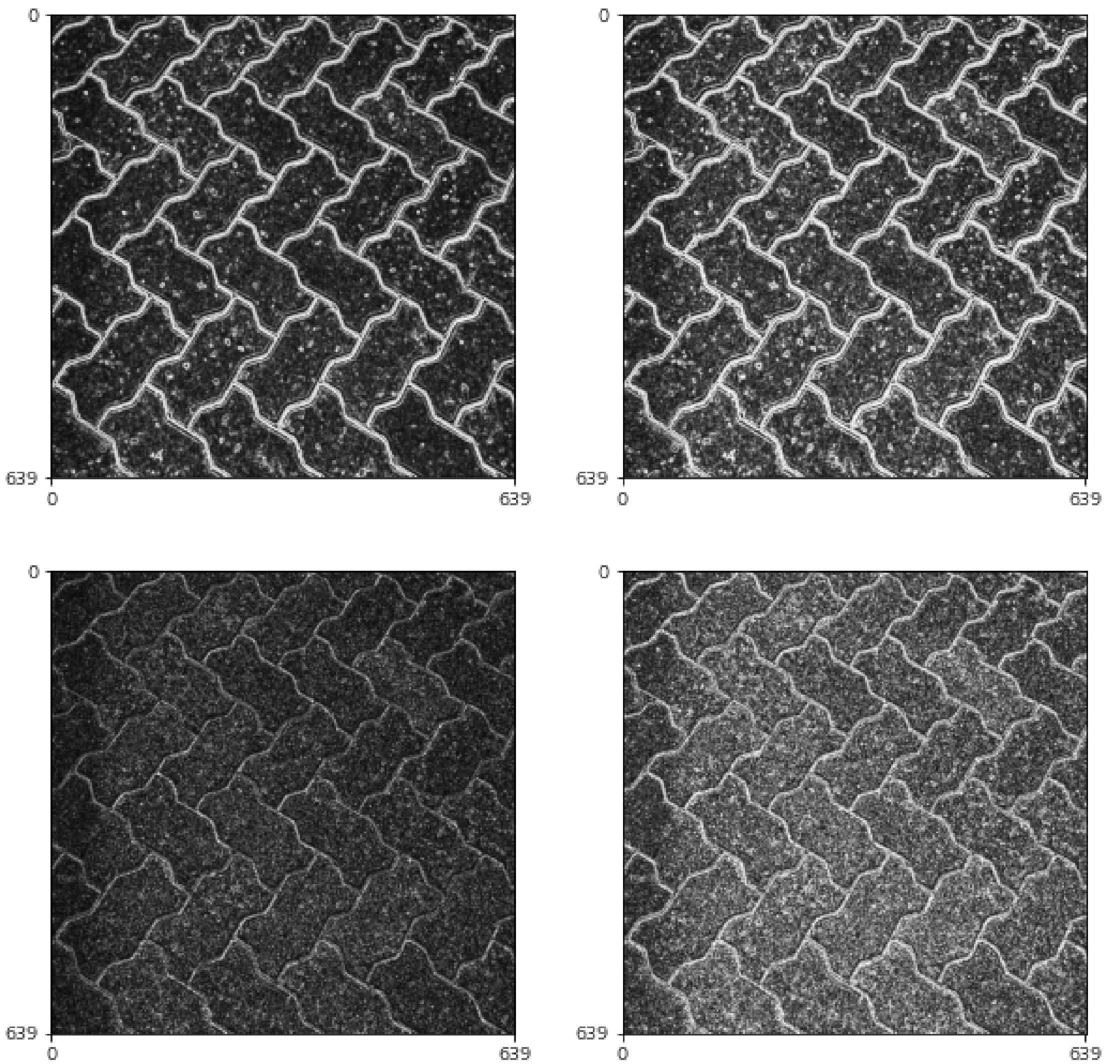
[48]:

```
img_Laplcian_8_1 = np.abs(img_Laplcian_8_1).clip(0, 255).astype(np.uint8)
ipp.show_images(img_Laplcian_4_1, img_Laplcian_8_1)
```



[49]:
```
pavement_4_Laplacian = cv2.filter2D(pavement_L, cv2.CV_32F, k_4_Laplacian_1)
pavement_8_Laplacian = cv2.filter2D(pavement_L, cv2.CV_32F, k_8_Laplacian_1)

pavement_4_Laplacian_abs = np.abs(pavement_4_Laplacian).clip(0,255).astype(np.
 ↪uint8)
pavement_8_Laplacian_abs = np.abs(pavement_8_Laplacian).clip(0,255).astype(np.
 ↪uint8)
```

[50]:
```
ipp.show_images(G_pavement_prewitt, G_pavement_sobel, pavement_4_Laplacian_abs,␣
 ↪pavement_8_Laplacian_abs)
```



영상에 가산 잡음 추가하기

$$g(x,y) = f(x,y) + n(x,y)$$

- $f(x,y)$: 원 영상 - $n(x,y)$: 잡음 신호 - $g(x,y)$: 잡음이 포함된 영상

- np.random.normal: 가우시안 분포를 갖는 샘플들을 반환

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

[51]:

```
help(np.random.normal)
```

```
Help on built-in function normal:

normal(...) method of numpy.random.mtrand.RandomState instance
    normal(loc=0.0, scale=1.0, size=None)

    Draw random samples from a normal (Gaussian) distribution.

    The probability density function of the normal distribution, first
    derived by De Moivre and 200 years later by both Gauss and Laplace
    independently [2]_, is often called the bell curve because of
    its characteristic shape (see the example below).

    The normal distributions occurs often in nature.  For example, it
    describes the commonly occurring distribution of samples influenced
    by a large number of tiny, random disturbances, each with its own
    unique distribution [2]_.

    .. note::
        New code should use the ``normal`` method of a ``default_rng()``
        instance instead; please see the :ref:`random-quick-start`.

    Parameters
    ----------
    loc : float or array_like of floats
        Mean ("centre") of the distribution.
    scale : float or array_like of floats
        Standard deviation (spread or "width") of the distribution. Must be
        non-negative.
    size : int or tuple of ints, optional
        Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
        ``m * n * k`` samples are drawn.  If size is ``None`` (default),
        a single value is returned if ``loc`` and ``scale`` are both scalars.
        Otherwise, ``np.broadcast(loc, scale).size`` samples are drawn.
```

```
Returns
-------
out : ndarray or scalar
    Drawn samples from the parameterized normal distribution.

See Also
--------
scipy.stats.norm : probability density function, distribution or
    cumulative density function, etc.
Generator.normal: which should be used for new code.

Notes
-----
The probability density for the Gaussian distribution is

.. math:: p(x) = \frac{1}{\sqrt{ 2 \pi \sigma^2 }}
                 e^{ - \frac{ (x - \mu)^2 } {2 \sigma^2} },

where :math:`\mu` is the mean and :math:`\sigma` the standard
deviation. The square of the standard deviation, :math:`\sigma^2`,
is called the variance.

The function has its peak at the mean, and its "spread" increases with
the standard deviation (the function reaches 0.607 times its maximum at
:math:`x + \sigma` and :math:`x - \sigma` [2]_).  This implies that
normal is more likely to return samples lying close to the mean, rather
than those far away.

References
----------
.. [1] Wikipedia, "Normal distribution",
       https://en.wikipedia.org/wiki/Normal_distribution
.. [2] P. R. Peebles Jr., "Central Limit Theorem" in "Probability,
       Random Variables and Random Signal Principles", 4th ed., 2001,
       pp. 51, 51, 125.
```

```
Examples
--------
Draw samples from the distribution:

>>> mu, sigma = 0, 0.1 # mean and standard deviation
>>> s = np.random.normal(mu, sigma, 1000)

Verify the mean and the variance:

>>> abs(mu - np.mean(s))
0.0  # may vary

>>> abs(sigma - np.std(s, ddof=1))
0.1  # may vary

Display the histogram of the samples, along with
the probability density function:

>>> import matplotlib.pyplot as plt
>>> count, bins, ignored = plt.hist(s, 30, density=True)
>>> plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
...                np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
...          linewidth=2, color='r')
>>> plt.show()

Two-by-four array of samples from N(3, 6.25):

>>> np.random.normal(3, 2.5, size=(2, 4))
array([[-4.49401501,  4.00950034, -1.81814867,  7.29718677],   # random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]])  # random
```

[52]:

```
np.random.normal(0, 0.1, 10)
```

[52]:

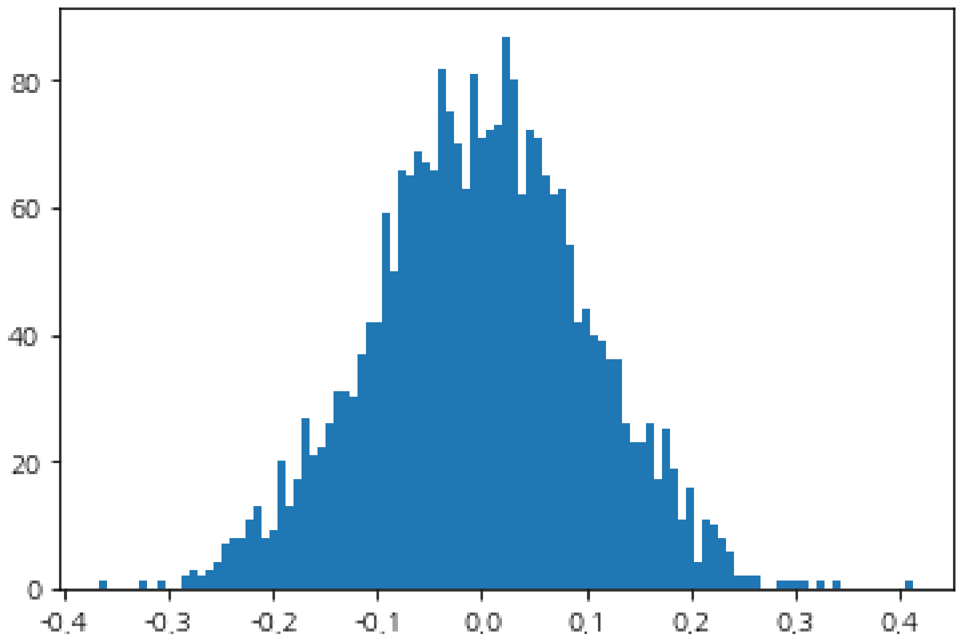
```
array([-0.04012875, -0.09745195,  0.1302848 , -0.03103422,  0.00024225,
        0.2325231 , -0.09976317,  0.09790241,  0.00715233,  0.16830369])
```

[53]:
```
noise = np.random.normal(0, 0.1, 2560)
noise.min(), noise.max()
```

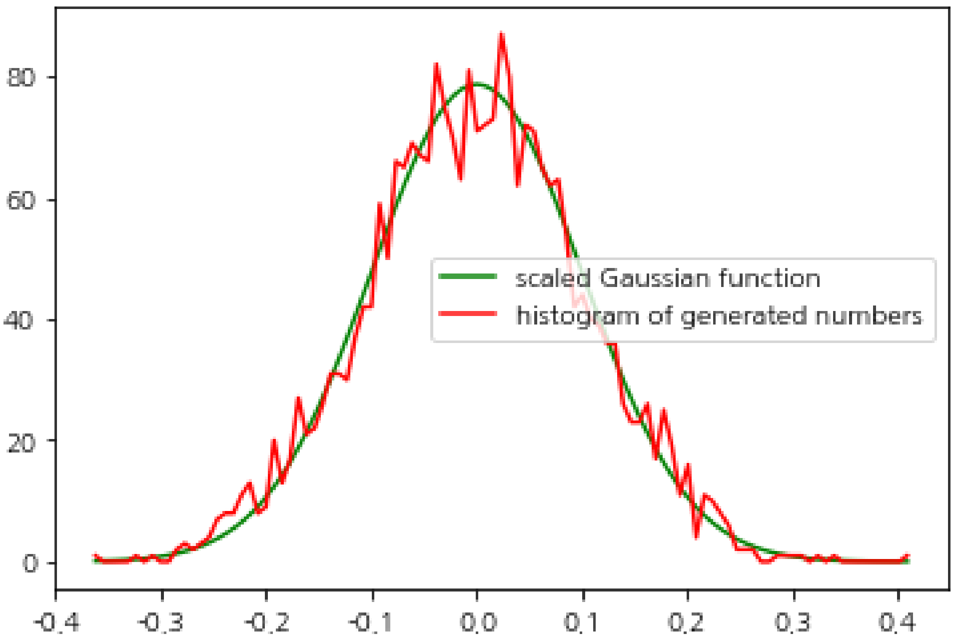
[53]:
```
(-0.36578833393264665, 0.41249722133025213)
```

[54]:
```
def get_Gaussian(x, mean, sigma):
    return np.exp(-((x-mean)/sigma)**2*0.5) / np.sqrt(2*np.pi) / sigma
```

[55]:
```
n, bins, _ = plt.hist(noise, bins=101)
```



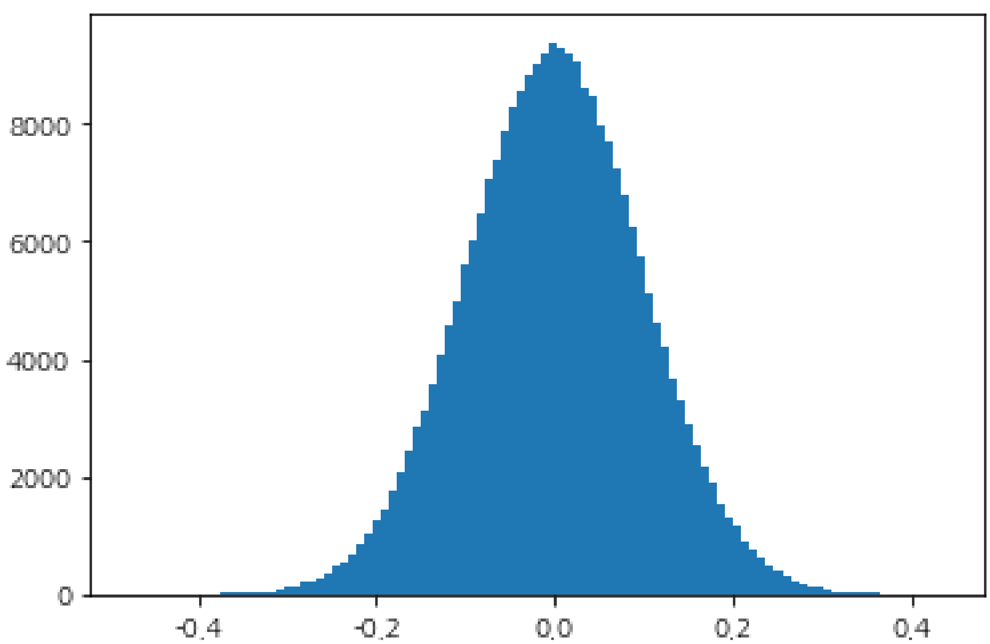
[57]:
```
x = (bins[:-1] + bins[1:]) /2
s = get_Gaussian(x, 0, 0.1)
plt.plot(x, s/s.sum()*n.sum(), 'g', label='scaled Gaussian function')
plt.plot(x, n, 'r', label='histogram of generated numbers')
plt.legend()
plt.show()
```



[58]:
```
noise2 = np.random.normal(0, 0.1, 256000)
noise2.min(), noise2.max()
```

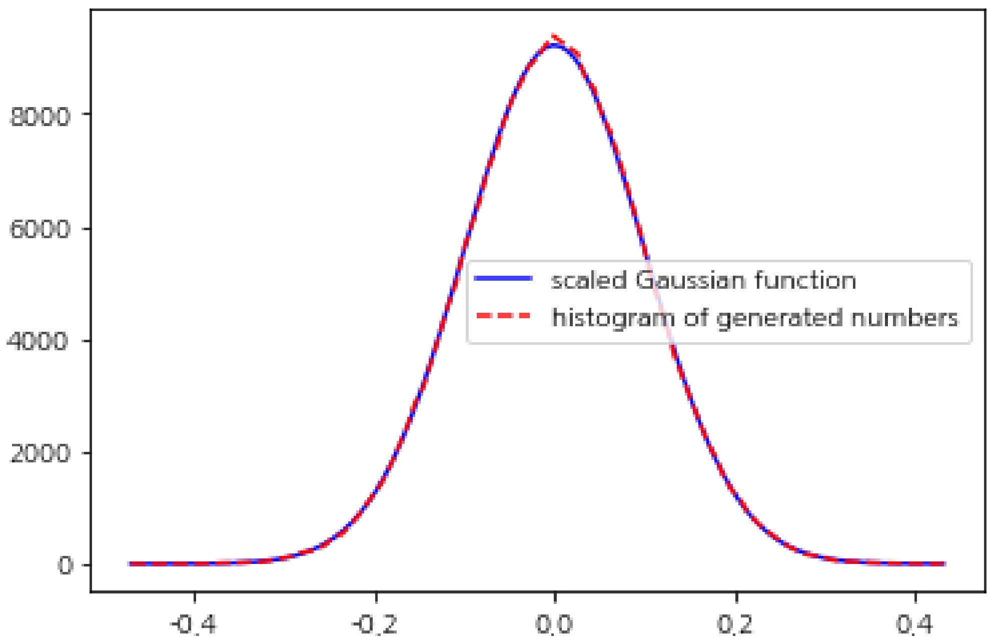
[58]: (-0.4751446280170797, 0.43630929173950533)

[59]:
```
n2, bins2, _ = plt.hist(noise2, bins=101)
```



[60]:

```
x2 = (bins2[:-1] + bins2[1:]) /2
s2 = get_Gaussian(x2, 0, 0.1)
plt.plot(x2, s2/s2.sum()*n2.sum(), 'b',label='scaled Gaussian function')
plt.plot(x2, n2, 'r--', label='histogram of generated numbers')
plt.legend()
plt.show()
```



- 정규 분포 난수의 다차원 배열 생성

[61]:

```
np.random.normal(0, 1, 15).reshape(3,5)
```

[61]:

```
array([[-1.12920012, -0.72169497,  2.31876889, -1.24219699,  0.12789992],
       [ 0.34135252, -1.0230918 , -0.79281516, -0.07394984, -0.51719812],
       [-1.59649297,  0.92740074,  0.21301876,  0.09276439,  1.77739313]])
```

[62]:

```
np.random.normal(0, 1, (3,5))
```

[62]:

```
array([[ 0.42589979, -0.84412805, -0.89767376, -0.04038192,  0.93902506],
       [ 1.57331762,  0.69910186,  2.41858022,  0.39815877, -1.07042896],
```

```
       [-0.96667342, -0.06040311,  0.72756674, -1.87383326,  1.45280572]])
```

[63]:
```python
np.random.normal(0, 1, (2,3,3))
```

[63]:
```
array([[[-1.53450729, -0.51891926,  0.52680773],
        [ 0.28983175, -0.79712354,  1.04143376],
        [-0.58772928, -0.97001365, -0.29778158]],

       [[ 0.2363708 ,  0.54655158, -0.29654715],
        [-1.08541638,  2.54840463,  2.57716362],
        [-0.96309863,  0.88213902,  0.22475701]]])
```

- random.uniform(low=0.0, high=1.0, size=None): [low, high) 구간에서 균일 분포를 갖는 난수를 반환

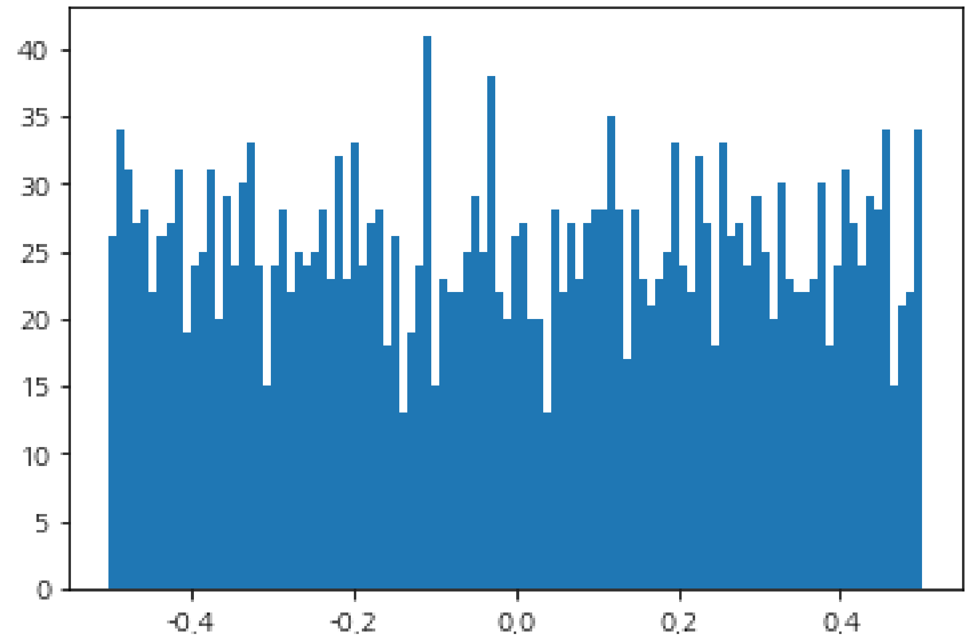
[64]:
```python
np.random.uniform(size=10)
```

[64]:
```
array([0.88090161, 0.3465047 , 0.18945243, 0.96840992, 0.69512719,
       0.77006049, 0.58577256, 0.63372168, 0.98955597, 0.80165536])
```

[65]:
```python
noise3 = np.random.uniform(-0.5, 0.5, 2560)
noise3.min(), noise3.max()
```

[65]:
```
(-0.49999855698725093, 0.4997883846961573)
```

[66]:
```python
n3, bins3, _ = plt.hist(noise3, bins=101)
```



[67]:

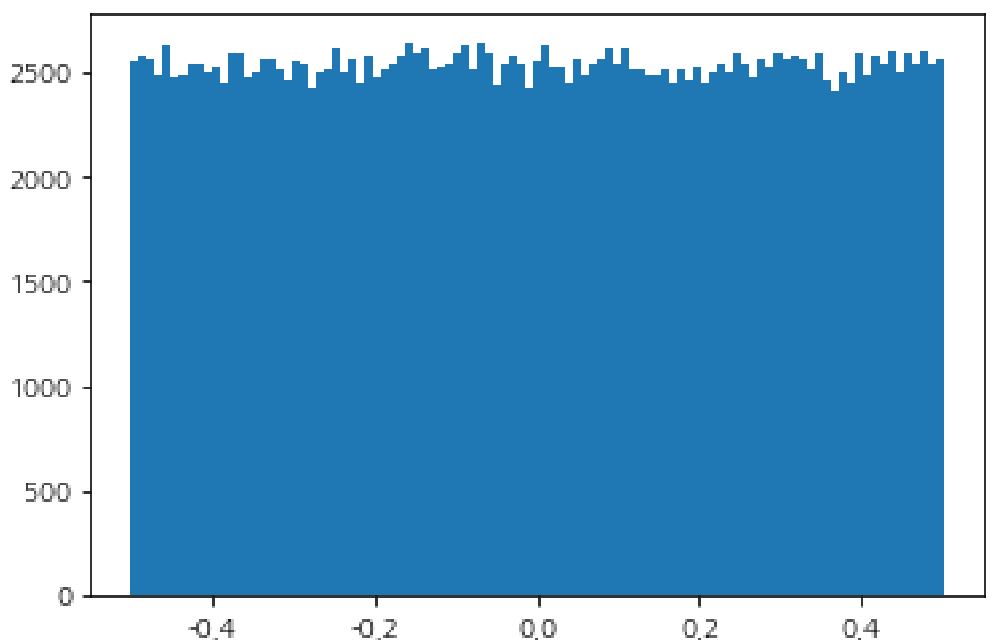
```
noise3 = np.random.uniform(-0.5, 0.5, 256000)
noise3.min(), noise3.max()
```

[67]:

```
(-0.499999535916415, 0.49999592278290894)
```

[68]:

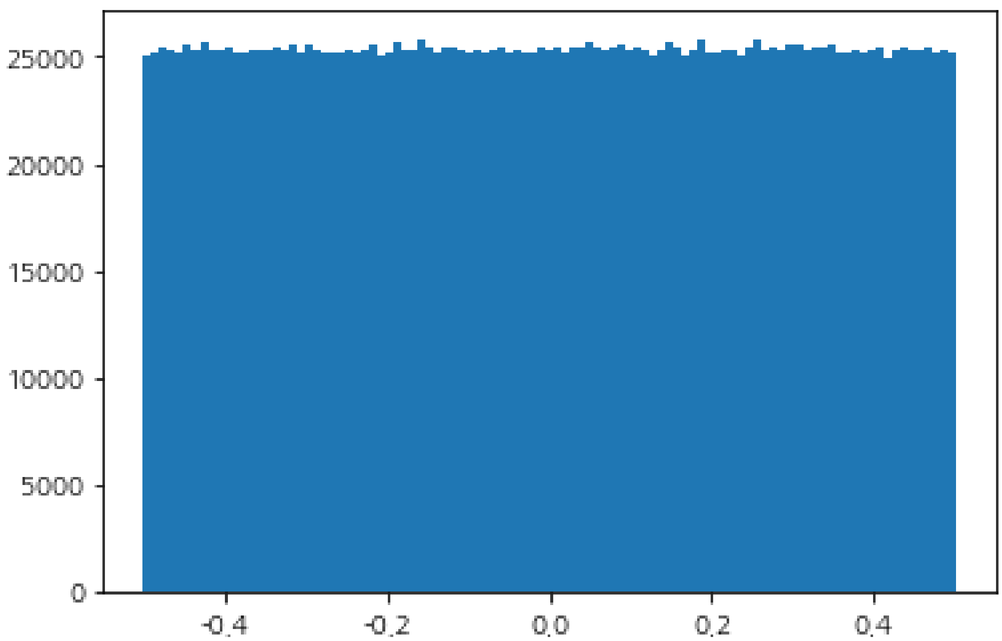
```
n3, bins3, _ = plt.hist(noise3, bins=101)
```



[69]:
```
noise3 = np.random.uniform(-0.5, 0.5, 2560000)
noise3.min(), noise3.max()
```

[69]:
```
(-0.49999968204295153, 0.4999997827111964)
```

[70]:
```
n3, bins3, _ = plt.hist(noise3, bins=101)
```

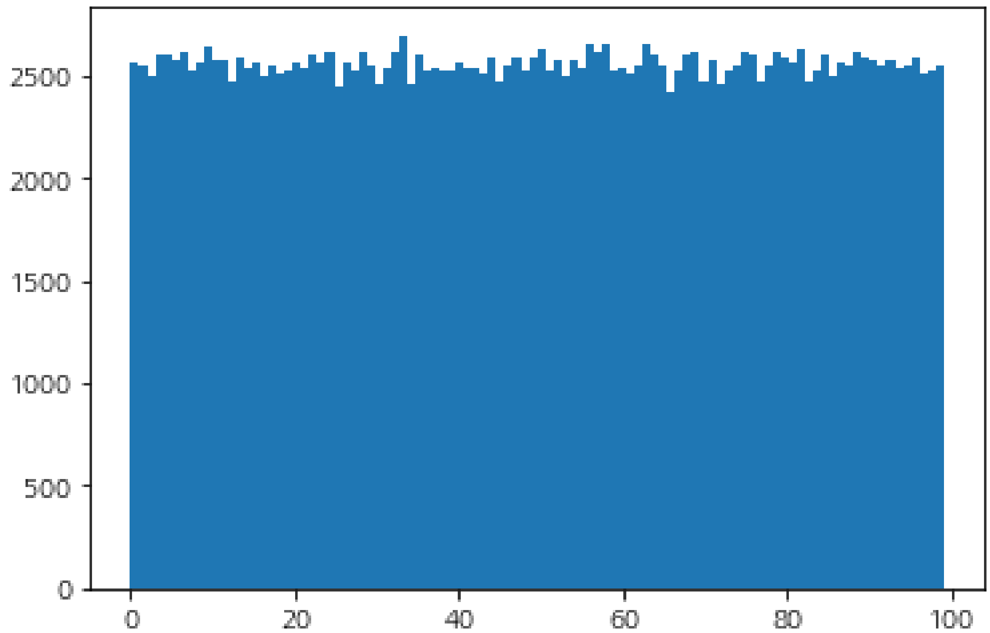


- np.random.randint(low, high, size): [low, high) 구간의 난수 정수 배열을 반환

[71]:
```
np.random.randint(0, 10, 10)
```

[71]:
```
array([6, 3, 5, 2, 1, 9, 0, 3, 3, 3])
```

[72]:
```
noise4 = np.random.randint(0, 100, 256000)
n4, bins4, _ = plt.hist(noise4, bins=100)
```



[73]:
```
pavement = cv2.imread('pavement_640.jpg')
pavement_L = cv2.cvtColor(pavement, cv2.COLOR_BGR2GRAY)
pavement_L.dtype, pavement_L.min(), pavement_L.max()
```

[73]:
```
(dtype('uint8'), 0, 255)
```

[74]:
```
np.random.seed(1)
n = np.random.normal(0, 0.5, pavement_L.shape)
```

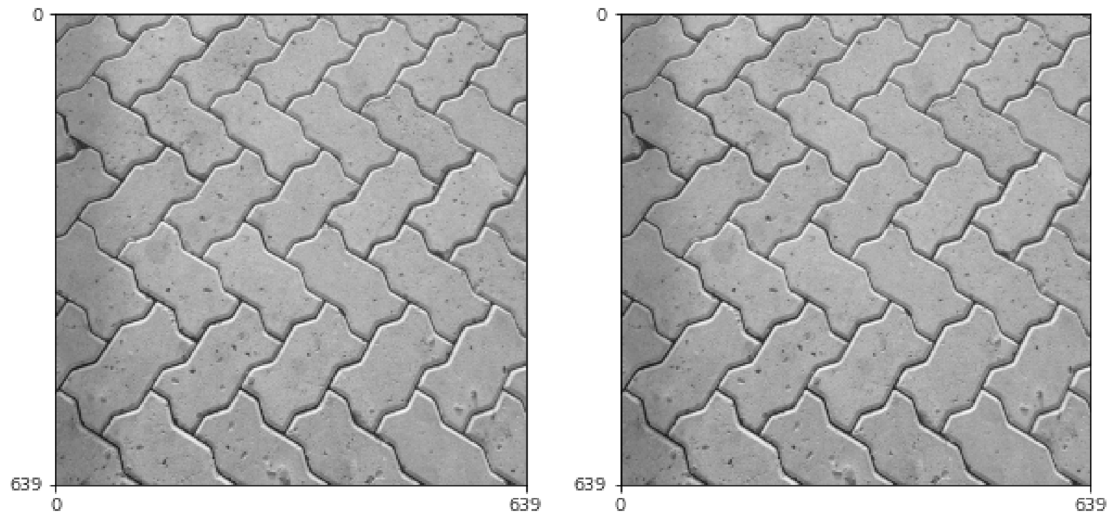
[75]:
```
n.dtype, n.shape, n.min(), n.max()
```

[75]:
```
(dtype('float64'), (640, 640), -2.3155544659709357, 2.265451332667966)
```

[76]:
```
n.mean(), n.std()
```

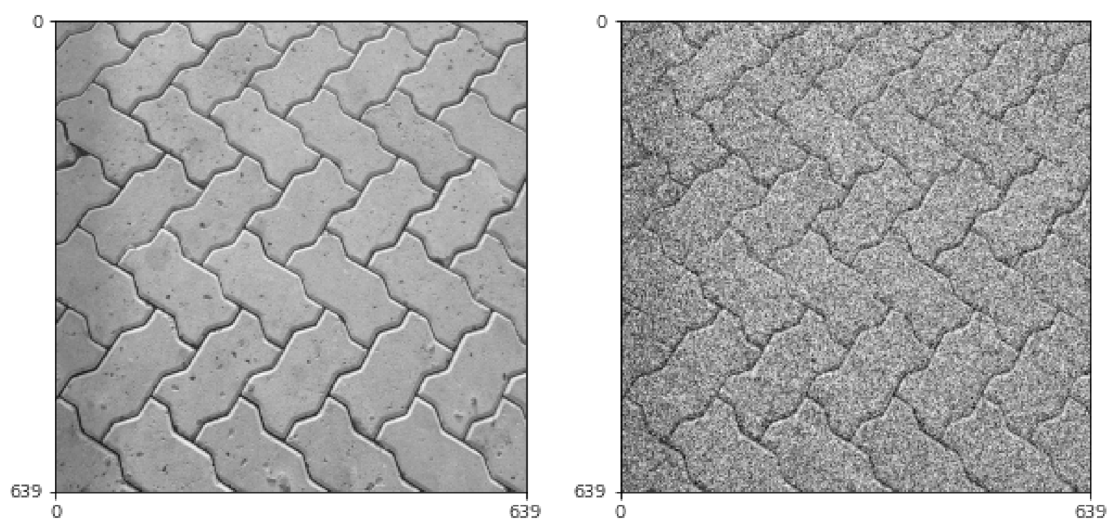
[76]:
```
(0.0005892246710953407, 0.49992191287326654)
```

[77]:
```
pavement_L_n= (pavement_L + n).round(0).clip(0, 255).astype(np.uint8)
ipp.show_images(pavement_L, pavement_L_n)
```



[78]:
```
n = np.random.normal(0, 0.5, pavement_L.shape)
noisy_pavement_L_f = pavement_L /  255 + n
noisy_pavement_L = (255*noisy_pavement_L_f).round(0).clip(0, 255).astype(np.
 ↪uint8)
```

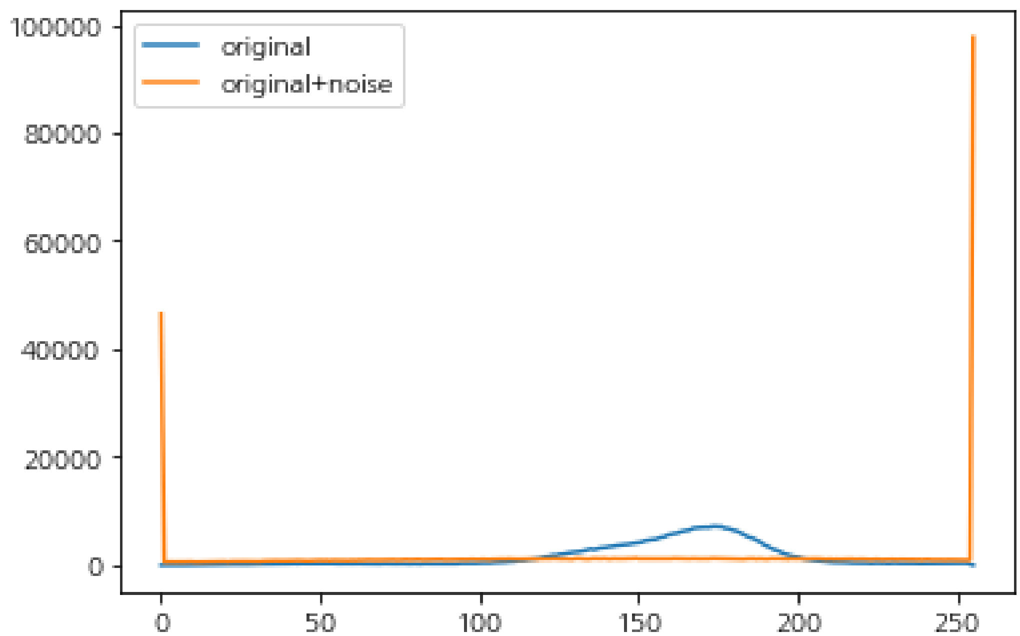
[79]:
```
ipp.show_images(pavement_L, noisy_pavement_L)
```



[80]:
```
hist, bins = np.histogram(pavement_L.ravel(),256,[0,256])
hist_n, bins_n = np.histogram(noisy_pavement_L.ravel(),256,[0,256])
```

[81]:

```
plt.plot(bins[:-1], hist, label='original')
plt.plot(bins_n[:-1], hist_n, label='original+noise')
plt.legend()
plt.show()
```



- $\sigma = 0.1$

[82]:

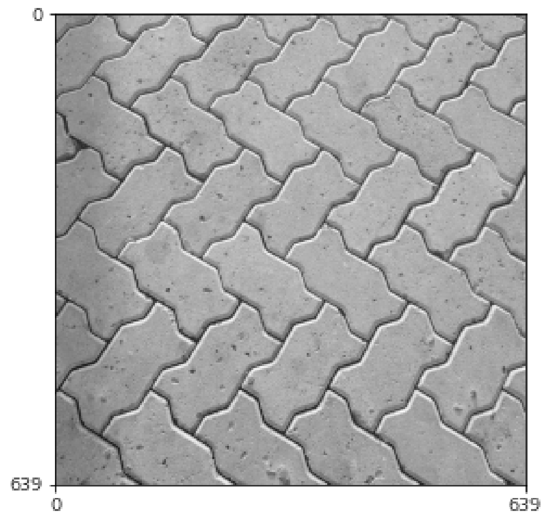
```
n = np.random.normal(0, 0.1, pavement_L.shape)
n.dtype, n.shape, n.min(), n.max()
```

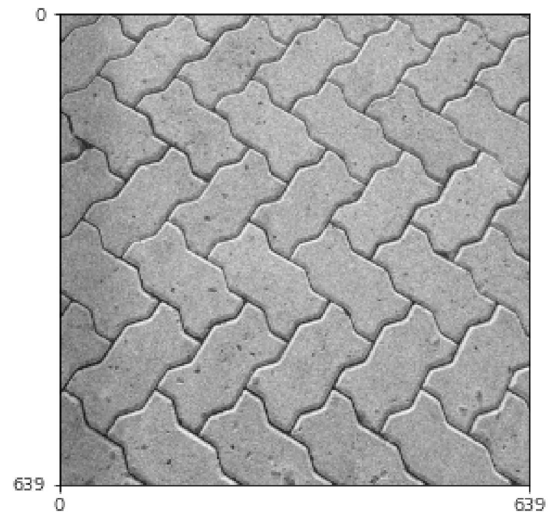
[82]:

```
(dtype('float64'), (640, 640), -0.4360805215816946, 0.4464345365538816)
```

[83]:

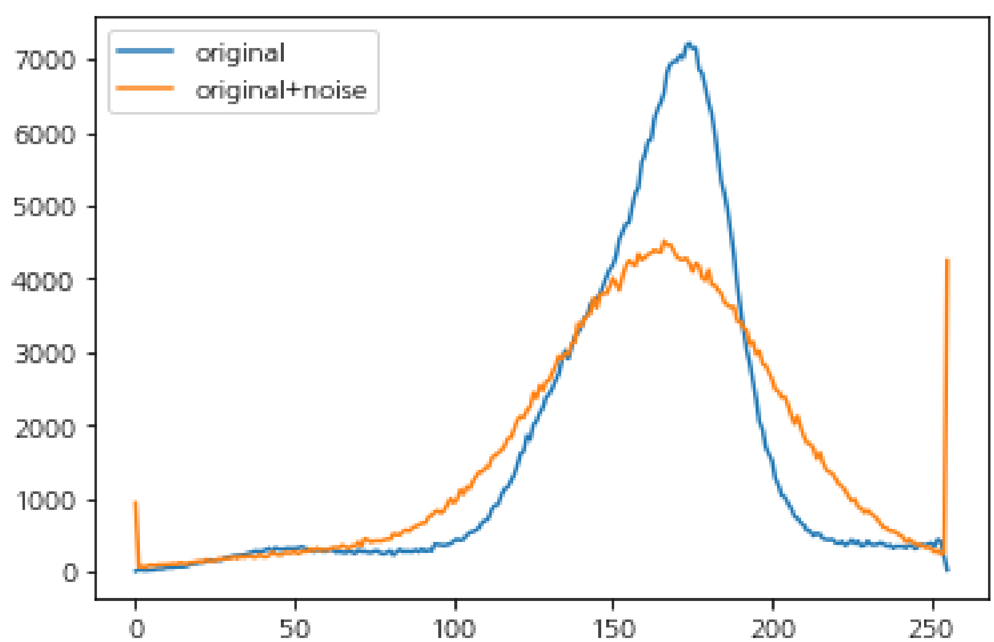
```
noisy_pavement_L_f = pavement_L/255 + n
noisy_pavement_L = (255*noisy_pavement_L_f).round(0).clip(0, 255).astype(np.
 ↪uint8)
ipp.show_images(pavement_L, noisy_pavement_L)
```





[84]:

```
hist, bins = np.histogram(pavement_L.ravel(),256,[0,256])
hist_n, bins_n = np.histogram(noisy_pavement_L.ravel(),256,[0,256])
plt.plot(bins[:-1], hist, label='original')
plt.plot(bins_n[:-1], hist_n, label='original+noise')
plt.legend()
plt.show()
```



Salt and Pepper 잡음

[86]:
```
a = np.full((10,8), 1)
a
```

[86]:
```
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

[87]:
```
np.random.seed(123)
x = np.random.randint(0, 10, 4)
y = np.random.randint(0, 8, 4)
x, y
```

[87]:
```
(array([2, 2, 6, 1]), array([3, 2, 3, 1]))
```

[88]:
```
a[x,y] = 255
a
```

[88]:
```
array([[  1,   1,   1,   1,   1,   1,   1,   1],
       [  1, 255,   1,   1,   1,   1,   1,   1],
       [  1,   1, 255, 255,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1, 255,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1]])
```

[89]:
```
x = np.random.randint(0, 10, 4)
y = np.random.randint(0, 8, 4)
x, y
```

[89]:
```
(array([6, 1, 0, 1]), array([6, 7, 1, 0]))
```

[90]:
```
a[x,y] = 0
a
```

[90]:
```
array([[  1,   0,   1,   1,   1,   1,   1,   1],
       [  0, 255,   1,   1,   1,   1,   1,   0],
       [  1,   1, 255, 255,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1, 255,   1,   1,   0,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1]])
```

[91]:
```
a = np.full((10,8), 1)
a
```

[91]:
```
array([[1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1, 1, 1]])
```

[92]:
```
salt_and_pepper = [0 if np.random.rand() < 0.5 else 255 for _ in range(8)]
salt_and_pepper
```

[92]:
```
[0, 255, 0, 0, 0, 255, 0, 0]
```

[93]:
```python
x = np.random.randint(0, 10, 8)
y = np.random.randint(0, 8, 8)
x, y
```

[93]:
```
(array([7, 2, 4, 8, 0, 7, 9, 3]), array([4, 4, 4, 6, 1, 5, 6, 6]))
```

[94]:
```python
a[x,y] = salt_and_pepper
a
```

[94]:
```
array([[  1,   0,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1, 255,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   0,   1],
       [  1,   1,   1,   1,   0,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   1,   1,   1,   1],
       [  1,   1,   1,   1,   0, 255,   1,   1],
       [  1,   1,   1,   1,   1,   1,   0,   1],
       [  1,   1,   1,   1,   1,   1,   0,   1]])
```

[96]:
```python
def add_salt_and_pepper_noise(img, ratio=0.1):
    if img.ndim == 2:
        SALT, PEPPER = 255, 0
    elif img.ndim == 3:
        SALT, PEPPER = (255, 255, 255), (0, 0, 0)
    else:
        return
    n = round(img.size * ratio)
    x = np.random.randint(0, img.shape[0], n)
    y = np.random.randint(0, img.shape[1], n)
    salt_and_pepper = [PEPPER if np.random.rand() < 0.5 else SALT for _ in␣
 ↪range(n)]
    img[x,y] = salt_and_pepper
```

잡음이 섞인 영상의 에지 검출