

① convolution (padding) → 피싱?

② 인리?



6

vector

neural network

⇒ MLP vector

image  
9

convolutional NN  
(CNN)

## Inclass 23: Image Convolution

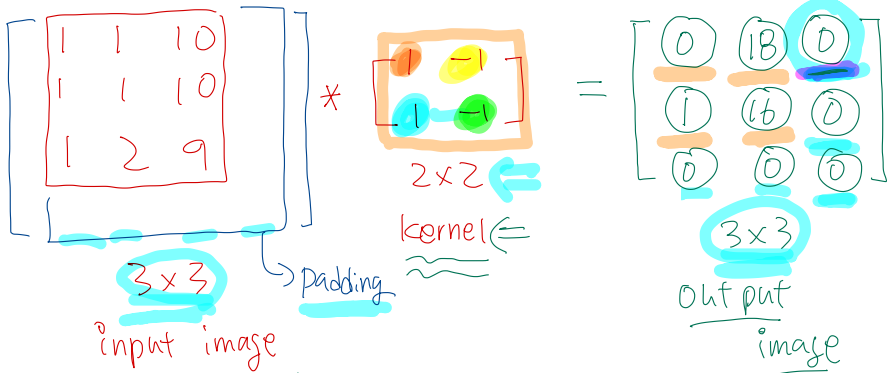
[SCS4049] Machine Learning and Data Science

---

Seongsik Park (s.park@dgu.ac.kr)

School of AI Convergence & Department of Artificial Intelligence, Dongguk University

# Image Convolution



input image

out put image

(Feature map)

$$1 \cdot (-1) + 1 \cdot 1 + 1 \cdot (-1) + 1 \cdot 1 = 0$$

$$1 \cdot (-1) + 10 \cdot 1 + 1 \cdot (-1) + 10 \cdot 1 = 18$$

$$1 \cdot (-1) + 1 \cdot 1 + 1 \cdot (-1) + 2 \cdot 1 = 1$$

$$1 \cdot (-1) + 10 \cdot 1 + 2 \cdot (-1) + 9 \cdot 1 = 16$$

|   |   |    |    |
|---|---|----|----|
| 1 | 1 | 10 | 10 |
| 1 | 1 | 10 | 10 |
| 1 | 2 | 9  | 9  |
| 1 | 2 | 9  | 9  |

3x3

input image

\*

|   |    |
|---|----|
| 1 | -1 |
| 1 | -1 |

2x2

kernel

=

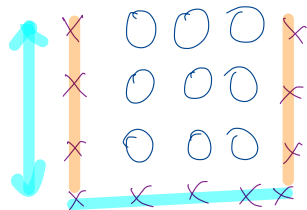
|   |    |   |
|---|----|---|
| 0 | 18 | 0 |
| 1 | 16 | 0 |
| 0 | 0  | 0 |

3x3

out put  
image

(Feature  
map)

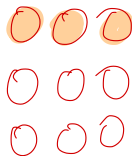




3x3  
input

$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

2x3  
kernel



3x3  
feature  
output.

padding 크기  $\Rightarrow$   $\frac{\text{필요한 크기}}{2}$ .

$\Rightarrow$  kernel size가 짝수.

# Linear filter and convolution

Many important effects can be modeled with a simple model. Construct a new array, the same size as the image. Fill each location of this new array with a weighted sum of the pixel values from the locations surrounding the corresponding location in the image *using the same set of weights each time*.

One example is computing a local average taken over a fixed region.

$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv} \quad (1)$$

We could average all pixels within a  $(2k+1) \times (2k+1)$  block of the pixel of interest.

# Linear filter and convolution

Whatever the weights chosen, the output of this procedure is

- **superposition:**  $R(f + g) = R(f) + R(g)$
- **scaling:**  $R(kf) = kR(f)$
- **shift invariance:**  $R(\text{Shift}(f, k)) = \text{Shift}(R(f), k)$

This procedure is known as **linear filtering**

# Linear filter and convolution

Given a filter kernel  $\mathcal{H}$ , the convolution of the kernel with image  $\mathcal{F}$  is an image  $\mathcal{R}$ . The  $(i,j)$ -th component of  $\mathcal{R}$  is given by

$$R_{ij} = \sum_{u,v} H_{i,j-u,v} F_{uv}.$$

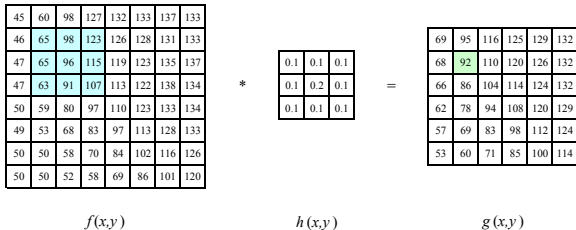
output  $i, j$



- **kernel** of the filter: the pattern of weights used for a linear filter
- **convolution**: the process of applying the filter



# Linear filter and convolution



**Figure 3.10** Neighborhood filtering (convolution): The image on the left is convolved with the filter in the middle to yield the image on the right. The light blue pixels indicate the source neighborhood for the light green destination pixel.

# Linear filter and convolution

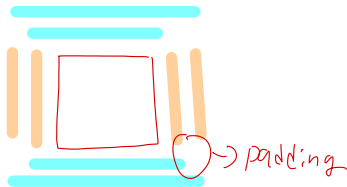
This operation is called **convolution**

$$R(f) = (h * f) \tag{3}$$

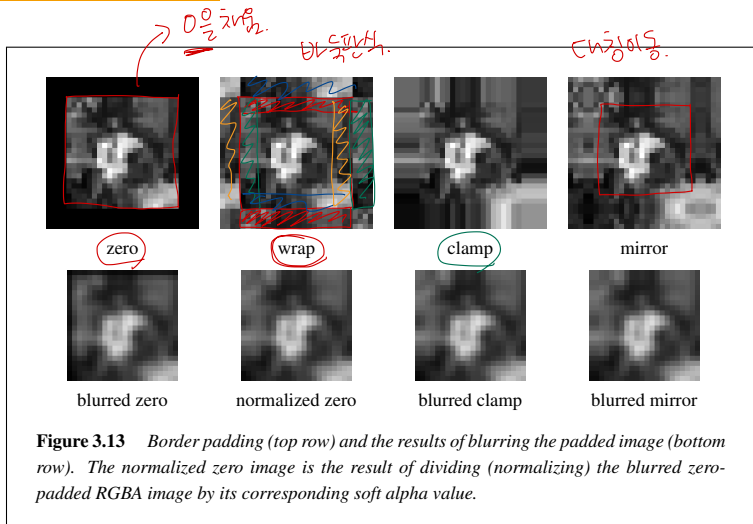
- *commutative*:  $(g * h)(x) = (h * g)(x)$
- *associative*:  $f * (g * h) = (f * g) * h$
- *distributive*:  $f * (g + h) = f * g + f * h$

# Padding (border effects)

- **zero**: set all pixels outside the source image to 0  $\leftarrow$  constant.
- **constant**: set all pixels outside the source image to a specified border value
- **clamp**: repeat edge pixels indefinitely
- **wrap**: loop “around” the image in a “toroidal” configuration
- **mirror**: reflect pixels across the image edge
- **extend**: extend the signal by subtracting the mirrored version of the signal from the edge pixel value

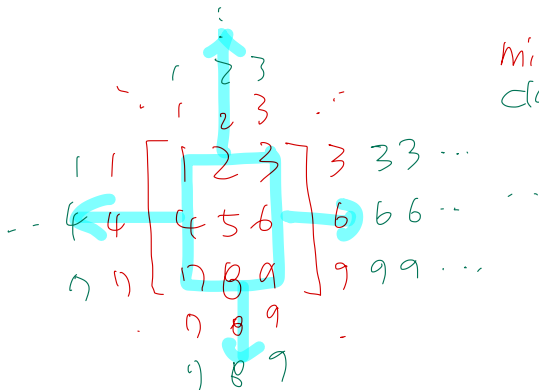


# Padding (border effects)



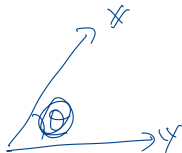
Handwritten notes on the right side:

↑  
1  
...  
0  
↓  
경계선

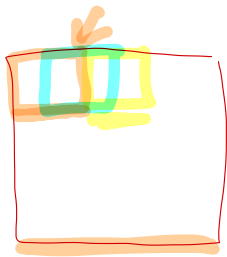


$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 1 \cdot 1 + 2 \cdot 0 + 3 \cdot 1 = 4$$

두 벡터의 내적은?  
 두 벡터의 길이와  
 코사인.



$$x \cdot y = \|x\| \|y\| \cos \theta$$

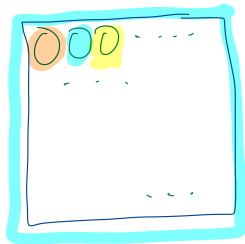


input

\*



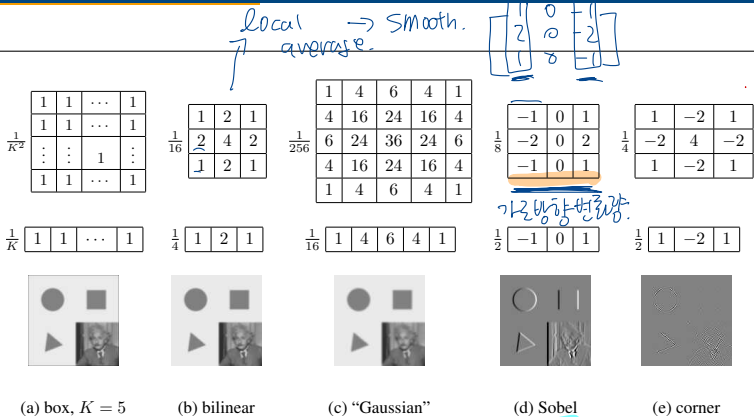
=



output.

feature map  
heat map

# Examples of linear filter



**Figure 3.14** Separable linear filters: For each image (a)–(e), we show the 2D filter kernel (top), the corresponding horizontal 1D kernel (middle), and the filtered image (bottom). The filtered Sobel and corner images are signed, scaled up by  $2\times$  and  $4\times$ , respectively, and added to a gray offset before display.



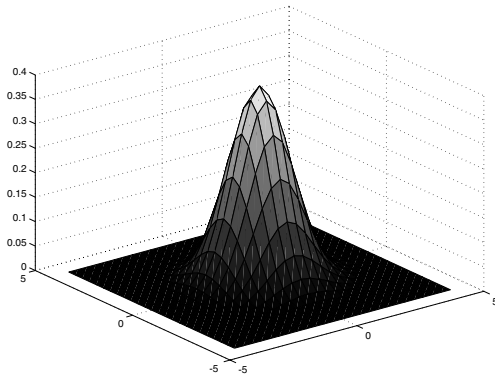
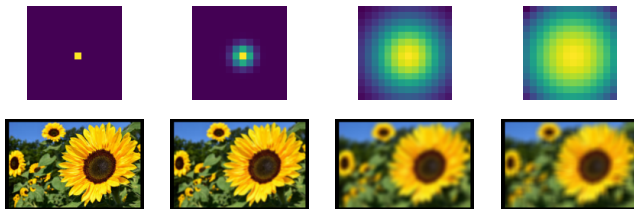


FIGURE 4.2: The symmetric Gaussian kernel in 2D. This view shows a kernel scaled so that its sum is equal to one; this scaling is quite often omitted. The kernel shown has  $\sigma = 1$ . Convolution with this kernel forms a weighted average that stresses the point at the center of the convolution window and incorporates little contribution from those at the boundary. Notice how the Gaussian is qualitatively similar to our description of the point spread function of image blur: it is circularly symmetric, has strongest response in the center, and dies away near the boundaries.

# Gaussian kernel in 2D

Symmetric Gaussian kernel in 2D (noise reduction smoothing)

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (4)$$



**Figure 1:** Gaussian blur with  $\sigma = [0.1, 1, 4, 10]$ , no padding.

# <sup>6</sup>Edge Detection<sup>9</sup>

---

(  $\sim 1/2$   $\sim 1/2$  )

# Derivative and finite differences

Image derivatives can be approximated using a convolution process.

We might estimate a partial derivative as a symmetric *finite difference*



가장방향  
Horizontal:  $\frac{\partial h}{\partial x} \approx h_{i+1,j} - h_{i-1,j}$   $\mathcal{H} = \begin{Bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{Bmatrix}$  (5)  
difference.

Vertical:  $\frac{\partial h}{\partial x} \approx h_{i,j+1} - h_{i,j-1}$   $\mathcal{H} = \begin{Bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{Bmatrix}$  (6)  
가장방향

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

# Derivative and finite differences

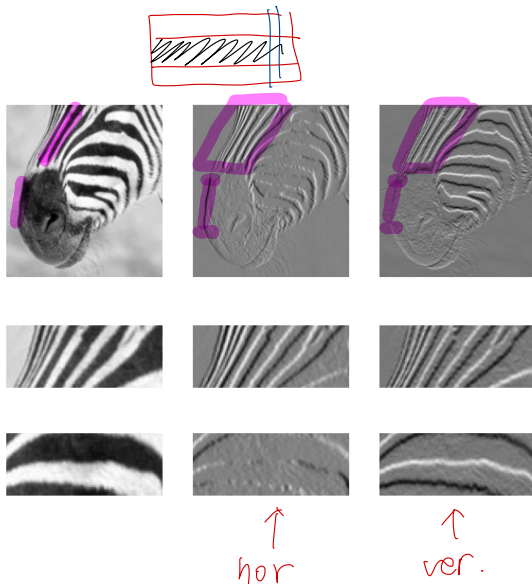


Figure 2: Horizontal and vertical difference using convolution filter

# Image gradient and Canny edge detection

The gradient of an image is a vector of its partials derivatives

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (7)$$

$$\text{magnitude} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \quad (8)$$

$$\text{direction} = \text{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right) \quad (9)$$

Canny edge detection

1. Gaussian blur
2. Image gradient magnitude
3. Threshold

# Image gradient and Canny edge detection

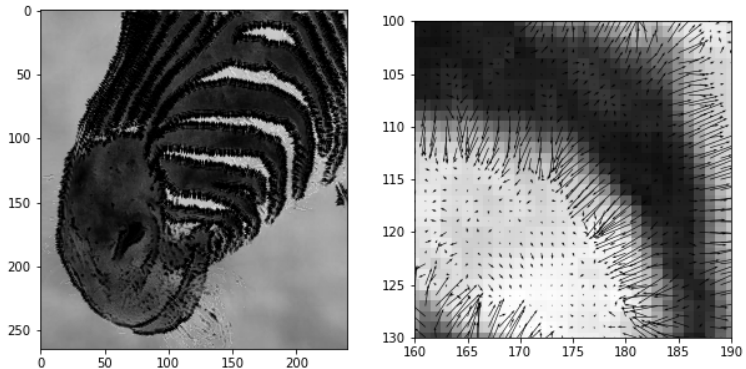
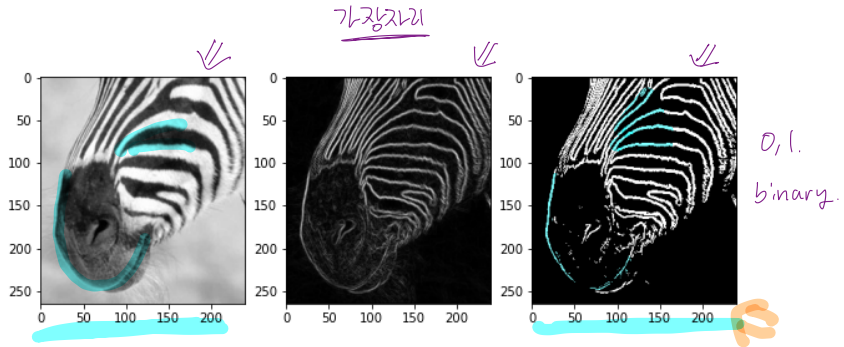


Figure 3: Quiver plot of image gradient

# Image gradient and Canny edge detection



**Figure 4:** Canny edge detection. Original image, gradient magnitude and thresholding (canny edge detection) image.



## Laplacian of Gaussian filter (LoG filter)

More sophisticated kernels can be created by first smoothing the image with a Gaussian filter,

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \quad (10)$$

and then taking the first and second derivatives. Such filters are known collectively as *band-pass filters*, since they filter out both low and high frequencies.

The undirected second derivative of a two-dimensional image,

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}, \quad (11)$$

is known as the *Laplacian* operator. Blurring an image with a Gaussian and then taking its Laplacian is equivalent to convolving directly with the *Laplacian of Gaussian* (LoG) filter,

$$\nabla^2 G(x, y; \sigma) = \left( \frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y; \sigma). \quad (12)$$

# Laplacian of Gaussian filter (LoG filter)

LoG filter

$$\nabla^2 G(x, y; \sigma) = \left( \frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2} \right) G(x, y; \sigma). \quad (13)$$

Or, simply

$$\mathcal{H} = \begin{Bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{Bmatrix} \quad (14)$$

# Laplacian of Gaussian filter (LoG filter)

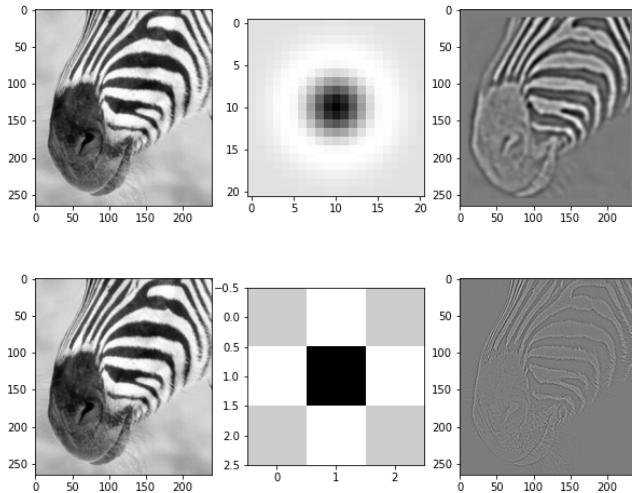


Figure 5: Laplacian of Gaussian filter

# Sobel operator

The *Sobel operator* is a simple approximation to a *directional* or *oriented* filter, which can be obtained by smoothing with a Gaussian (or some other filter) and then taking a *directional derivative*  $\nabla_{\hat{\mathbf{u}}} = \frac{\partial}{\partial \hat{\mathbf{u}}}$ , which is obtained by taking the dot product between the gradient field  $\nabla$  and a unit direction  $\hat{\mathbf{u}} = (\cos \theta, \sin \theta)$ ,

$$\hat{\mathbf{u}} \cdot \nabla(G * f) = \nabla_{\hat{\mathbf{u}}}(G * f) = (\nabla_{\hat{\mathbf{u}}}G) * f. \quad (15)$$

The smoothed directional derivative filter,

$$G_{\hat{\mathbf{u}}} = u_1 G_x + u_2 G_y = u_1 \frac{\partial G}{\partial x} + u_2 \frac{\partial G}{\partial y} \quad (16)$$

where  $\hat{\mathbf{u}} = (u_1, u_2)$ , is an example of a *steerable* filter, since the value of an image convolved with  $G_{\hat{\mathbf{u}}}$  can be computed by first convolving with the pair of filter  $(G_x, G_y)$  and then *steering* the filter by multiplying this gradient field with a unit vector  $\hat{\mathbf{u}}$ .

# Sobel operator

Sobel operator for a given  $\hat{\mathbf{u}} = (u_1, u_2)$

$$G_{\hat{\mathbf{u}}} = u_1 G_x + u_2 G_y = u_1 \frac{\partial G}{\partial x} + u_2 \frac{\partial G}{\partial y} \quad (17)$$

with

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \quad (18)$$

$$\frac{\partial G}{\partial x} = -\frac{x}{\sigma^2} G(x, y; \sigma) \quad \frac{\partial G}{\partial y} = -\frac{y}{\sigma^2} G(x, y; \sigma) \quad (19)$$

# Sobel operator

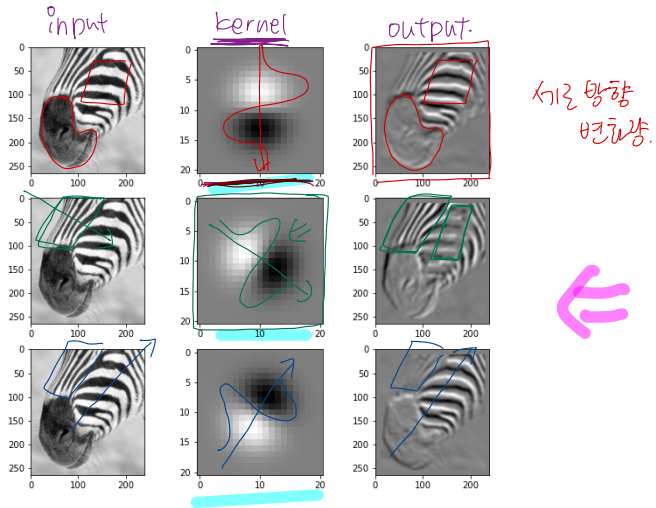


Figure 6: Sobel operator with  $u = (1, 0)$  (top),  $u = (0.447, 0.894)$  (middle),  $u = (-0.894, 0.447)$  (bottom)