CSC 422/522
Computer Vision and Pattern Recognition

Image Processing (Filtering)

2026 Spring

South Dakota State University

Today

- Examples of Filtering
- Linear Filtering
  - Concept
  - Some Classical Kernels
- From Correlation to Convolution
  - Mathematical Formulation
  - Border Effect
  - Separable filtering
- Brief Introduction on Binary Image Processing

Reminder:
please avoid using the *Lenna* image in examples or demonstrations ☺

Important Announcement: Adjustment of Course Schedule

I plan to move the 3D computer vision part after spring break

1	13-Jan	Introduction and Logistics
2	15-Jan	Image Formation (Camera Intrinsic and Extrinsic Matrices)
3	20-Jan	Camera Calibration with OpenCV
4	22-Jan	Image Processing (Basics with OpenCV)
5	27-Jan	Image Processing: Filtering
6	29-Jan	Feature Detection: Edges, Lines
7	3-Feb	Feature Detection: Corners, scale-invariant, SIFT
8	5-Feb	Image Alignment and Stitching, RANSAC
9	10-Feb	Neural Networks: Basic Architecture
10	12-Feb	Neural Networks: Training and Inference
11	17-Feb	CNN: Basic Architecture
12	19-Feb	CNN: Backpropagation
13	24-Feb	CNN Based Classification
14	26-Feb	CNN Based Object Detection
15	3-Mar	CNN Based Object Tracking
16	5-Mar	CNN Based Segmentation
17	10-Mar	Review Topics for Exam 1
18	12-Mar	Exam 1
19	17-Mar	*Spring Break*
20	19-Mar	*Spring Break*

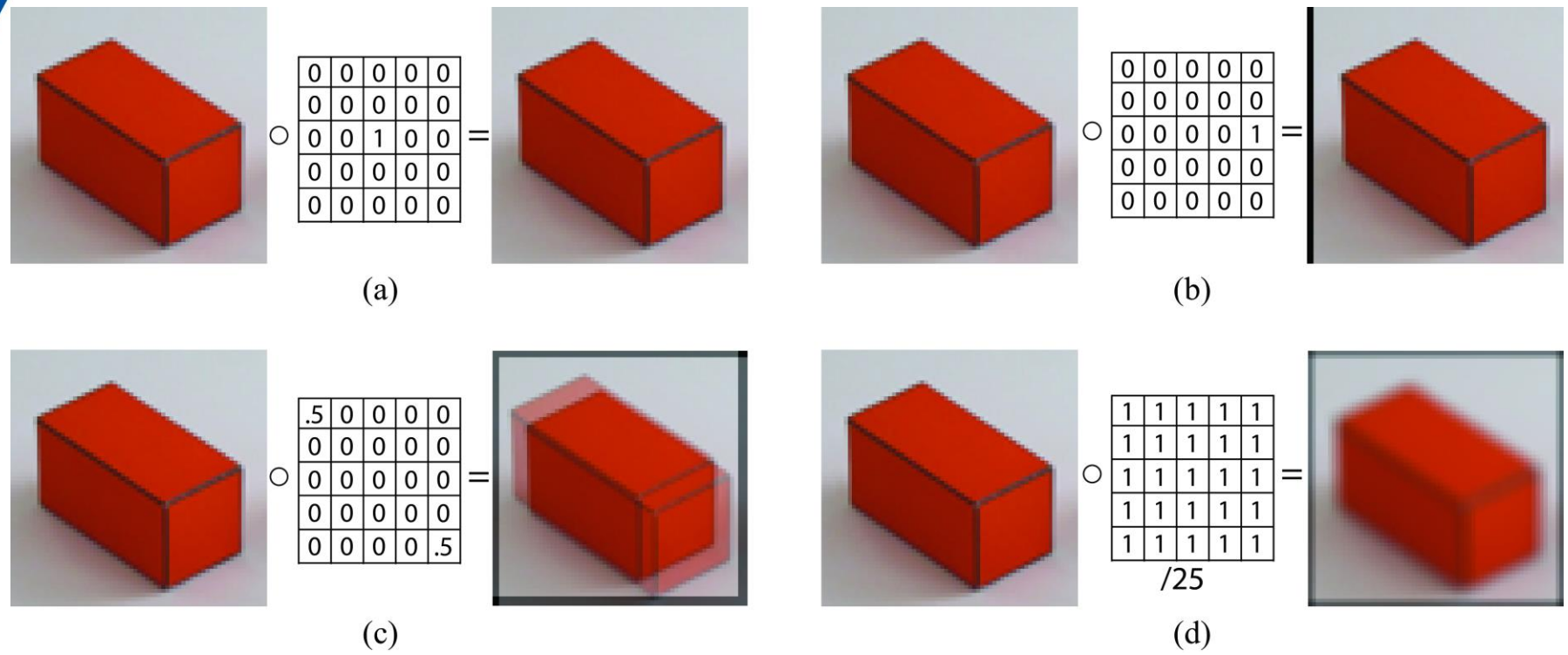


Image source: https://visionbook.mit.edu/linear_image_filtering.html

Filtering (i.e., Neighborhood Operators)

Filtering

- Definition: using a collection of pixel values in the vicinity of a given pixel to determine its final output value.
- It is also called neighborhood operator or local operator.

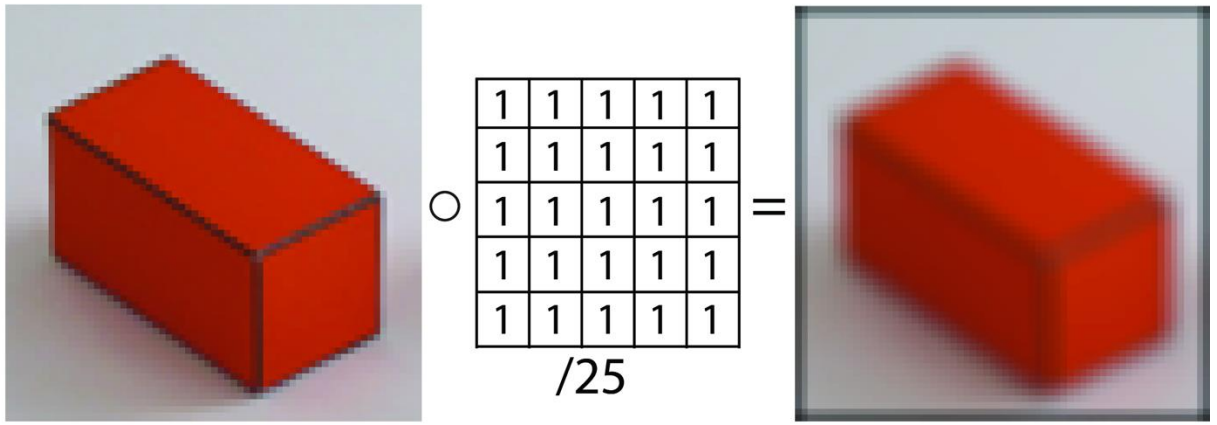


Image source: https://visionbook.mit.edu/linear_image_filtering.html

Recap: In last lecture, we have seen(global) histogram equalization, which is NOT Filtering

before after

Histogram equalization determines a new pixel value based *only* on that pixel's intensity value relative to the *global* cumulative distribution function (CDF) of the entire image.

Result from https://colab.research.google.com/drive/1wZt00QnxYgMVjlih16RI73lS_99UlHQt?usp=sharing

However, locally adaptive histogram equalization is filtering

(a) (b) (c)

Figure 3.8 *Locally adaptive histogram equalization: (a) original image; (b) block histogram equalization; (c) full locally adaptive equalization.*

Reference: Computer Vision: Algorithms and Applications, 2nd ed.

Filtering Example: Blurring

before

after

Computer Vision: Algorithms and Applications, 2nd ed.

Filtering Example: Sharpening

before

after

Computer Vision: Algorithms and Applications, 2nd ed.

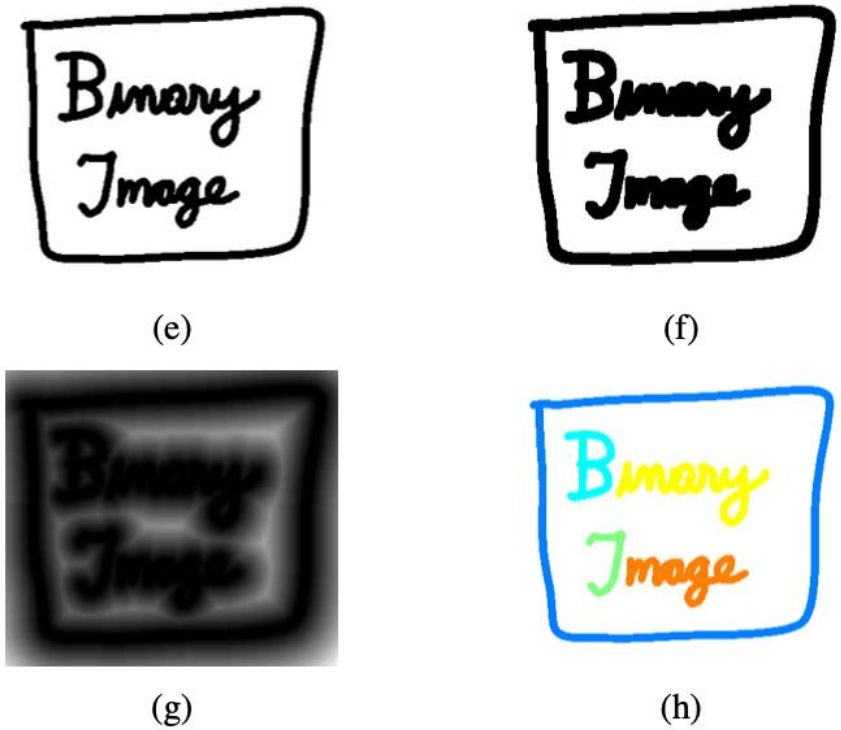
Filtering Example: Smoothing with edge-preserving filter

before

after

Computer Vision: Algorithms and Applications, 2nd ed.

Filtering Example: Some Binary Image Processing Operators



- (e) original binary image;
- (f) dilated;
- (g) distance transform;
- (h) connected components

Computer Vision: Algorithms and Applications, 2nd ed.

South Dakota State University

Original

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Blur (with a box filter)

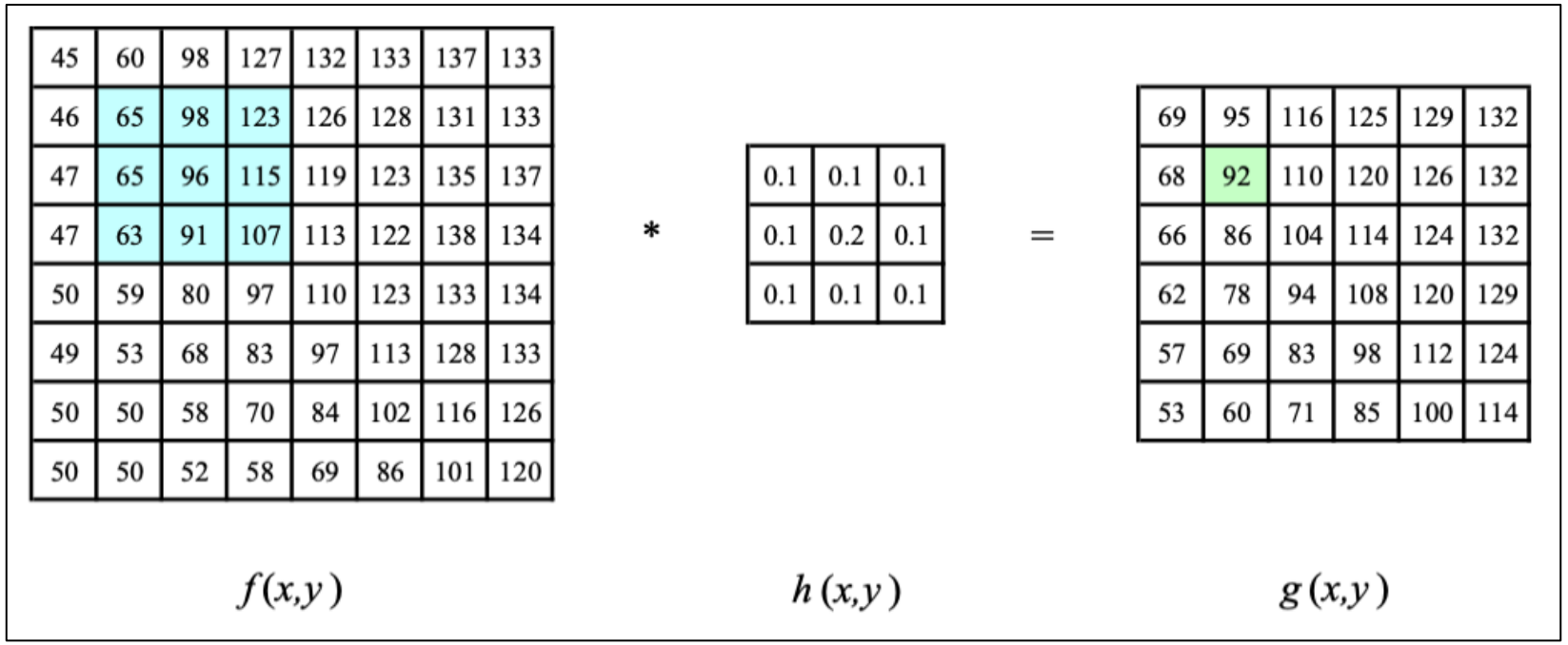
Credits: some slides of this section are adapted from CSC492/592, Spring 2024

Linear Filtering

Filtering: Linear vs. Non-linear

- Linear filtering
  - Linear combination (e.g., weighted sum) of pixels in neighborhoods
- Non-linear filtering
  - Non-linear combination (e.g., median) of pixels in neighborhoods

Linear filtering



$$g(i,j) = \sum_{i,j} f(i+k, j+l)h(k,l)$$

Linear filtering (Guess the Kernel!)

Original

0	0	0
0	1	0
0	0	0

?

Linear filtering (Guess the Kernel!)

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Linear filtering (Guess the Kernel!)

Original

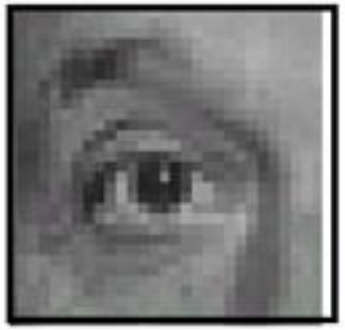
0	0	0
0	0	1
0	0	0

?

Linear filtering (Guess the Kernel!)

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Linear filtering (Guess the Kernel!)

Original

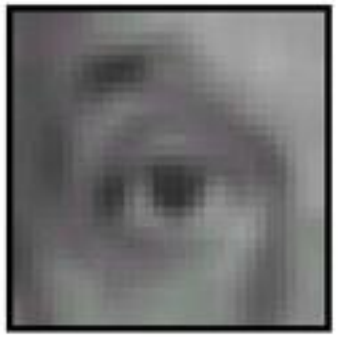
$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

?

Linear filtering (Guess the Kernel!)

Original

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Blur (with a box filter)

Linear filtering (Guess the Kernel!)

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad ?$$

(Note that filter sums to 1)

Linear filtering (Guess the Kernel!)

Original

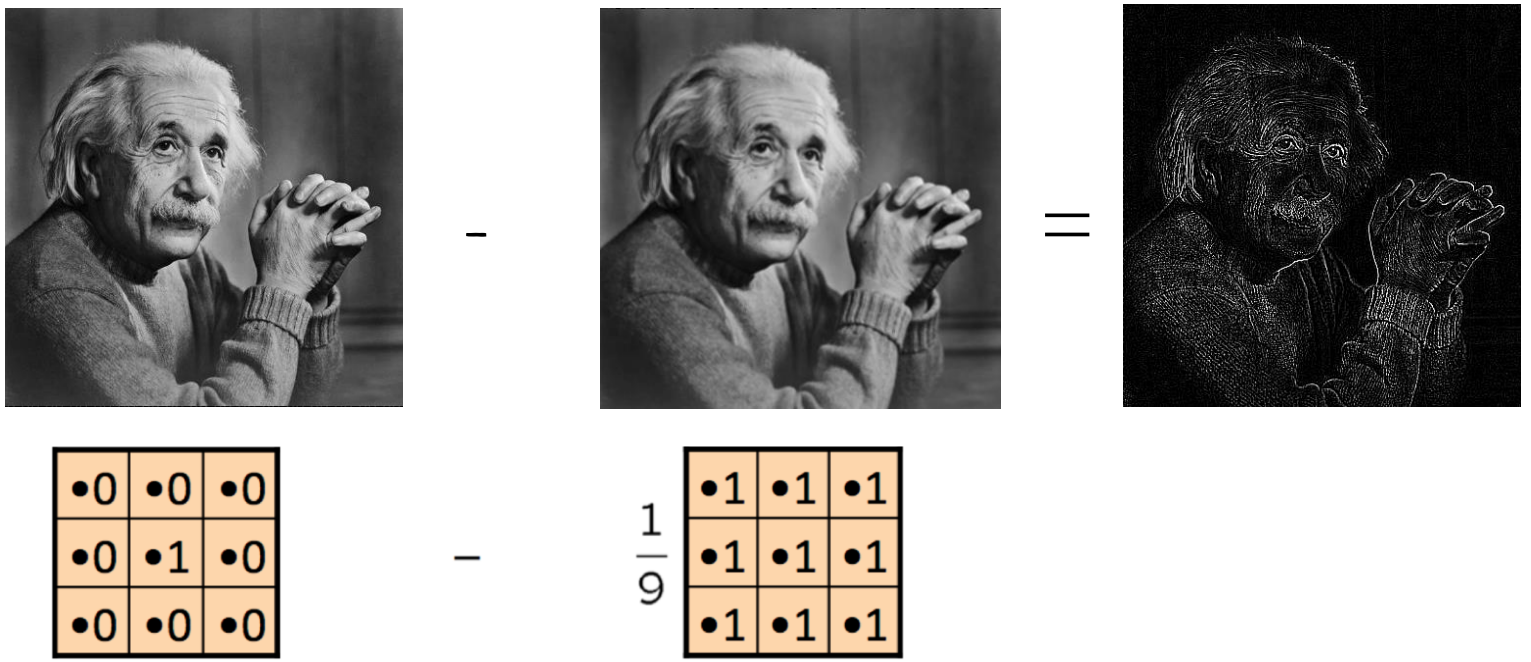
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

- Accentuates differences with local average
- Also known as Laplacian

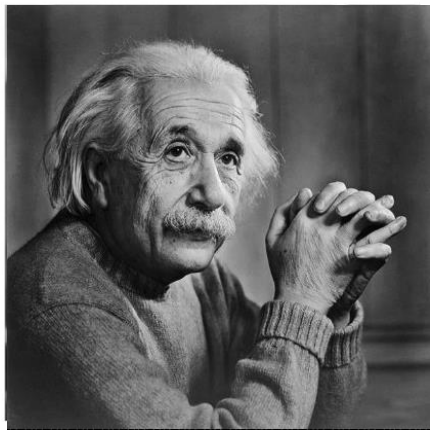
Linear filtering (Sharpening)

- Original - ”Smoothed” = Details

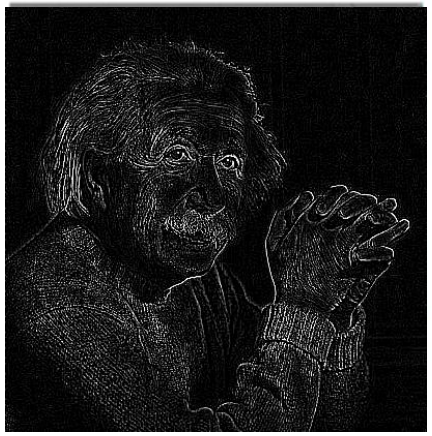


Linear filtering (Sharpening)

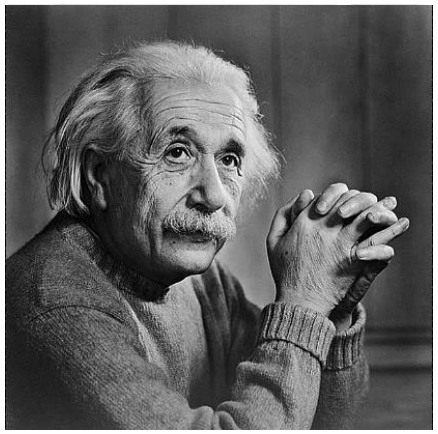
- Original + ”Details” = Sharpened



+ a



=



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Linear filtering (Blurring)

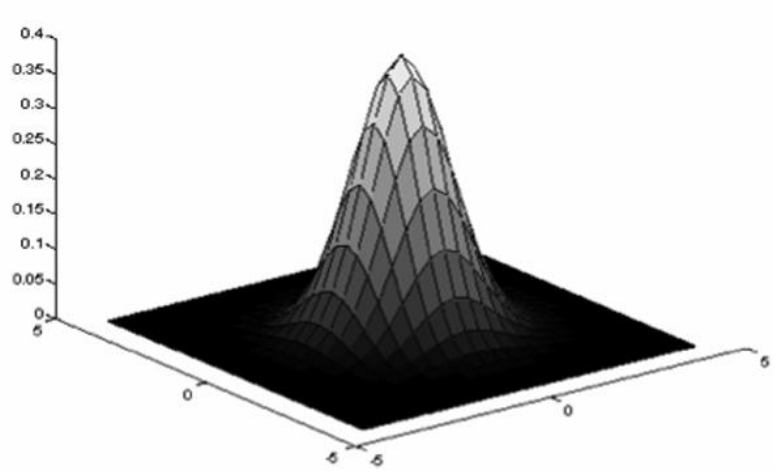
- Box filter:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Can we do better?

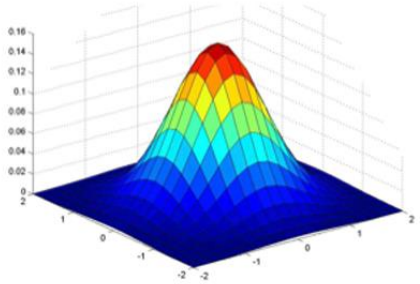
Linear filtering (Blurring)

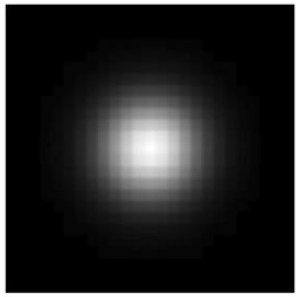
- Gaussian Filter:
  - A Gaussian gives a good model of a fuzzy blob
  - It closely models many physical processes
  - Effect of low-pass filter



Linear filtering (Blurring)

- Gaussian Kernel:



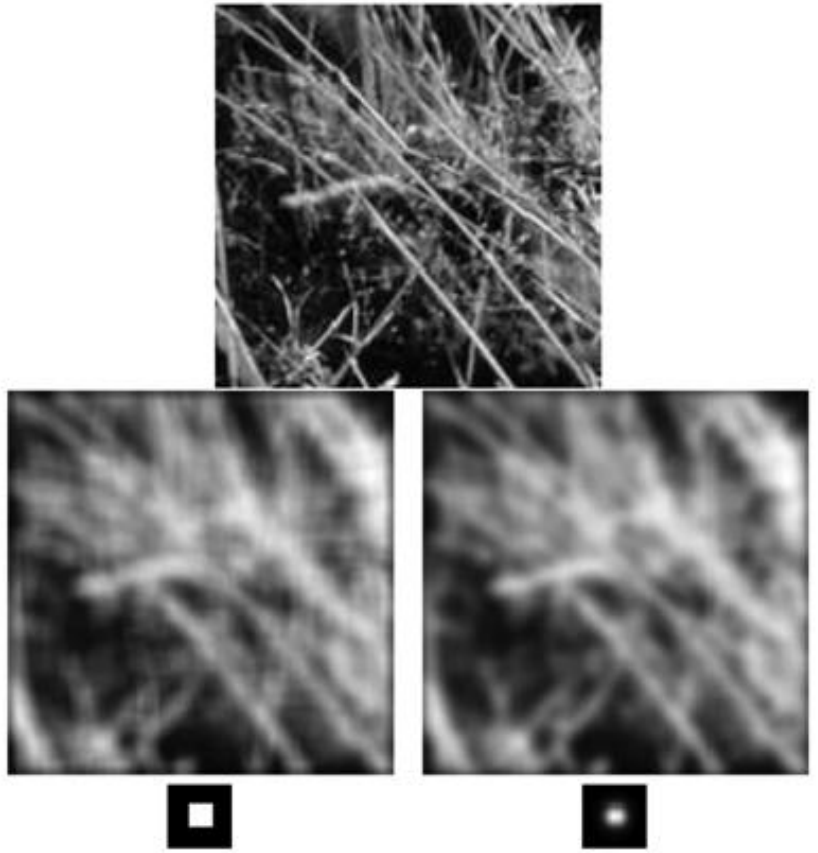


0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

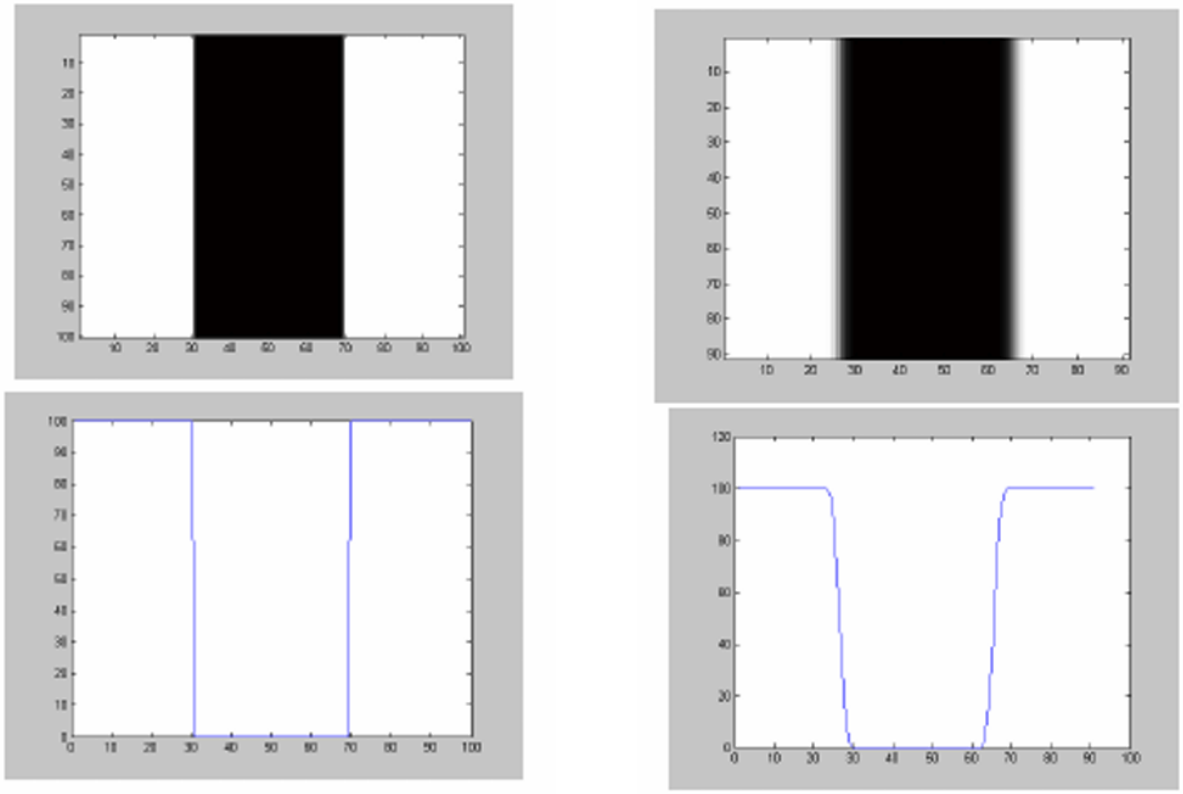
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Linear filtering (Blurring): Box vs. Gaussian



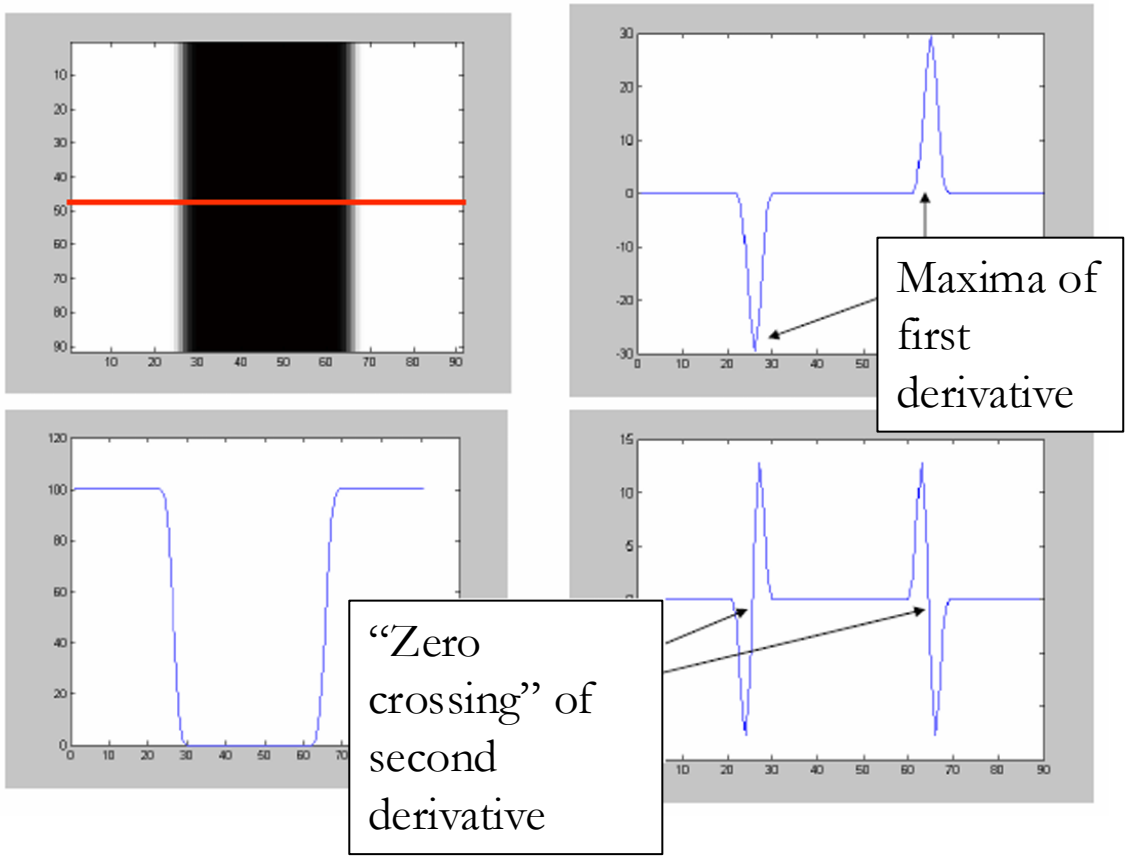
Edge Detection

- Edges correspond to fast changes



Edge Detection

- How can we find the edges?



Edge Detection – Finite Difference Filter

- Finite difference filter:
  - ***Concept.*** Calculates the derivatives of an image by approximating the derivative of a function
  - ***Idea.*** Approximate the derivative of a function by taking differences between successive pixel values.

Edge Detection – Finite Difference Filter

- M=[-1 0 1]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	-12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	3	6	6	6	3	0	0

(c) S_3 is an upward ramp

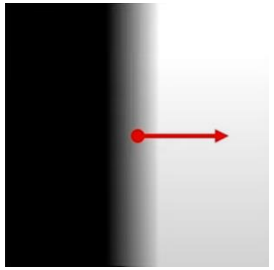
S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	12	0	-12	0	0	0	0

(d) S_4 is a bright impulse or "line"

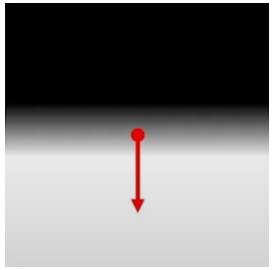
Edge Detection – Finite Difference Filter

- **Gradient** (partial derivatives) represents the direction of most rapid change in intensity

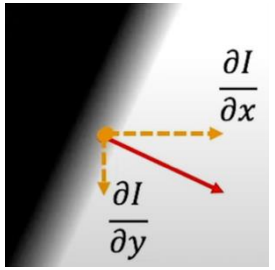
$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$



$$\nabla I = \left[\frac{\partial I}{\partial x}, 0\right]$$



$$\nabla I = \left[0, \frac{\partial I}{\partial y}\right]$$



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

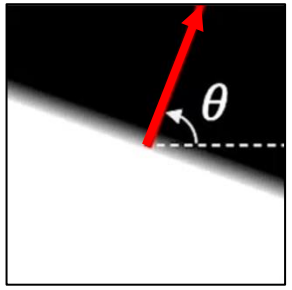
Edge Detection – Finite Difference Filter

- ***Gradient Magnitude***

$$S = \|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

- ***Gradient Orientation***

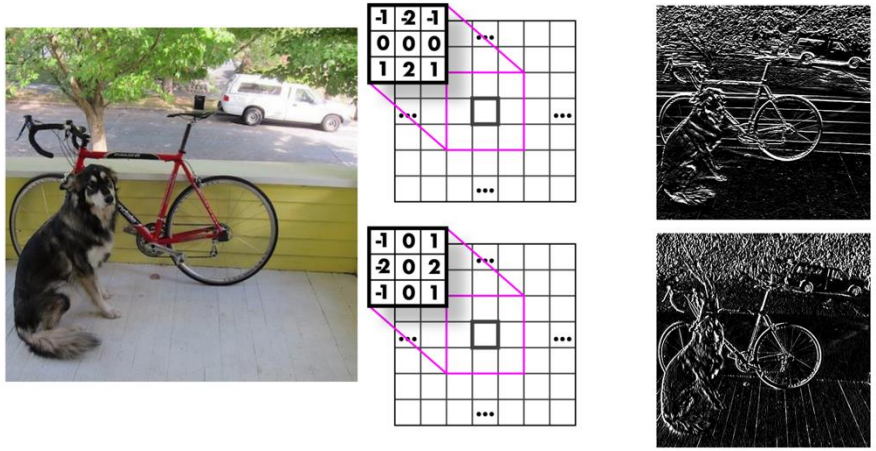
$$\theta = \tan^{-1}\left(\frac{\partial I}{\partial x} / \frac{\partial I}{\partial y}\right)$$



Edge Detection – Finite Difference Filter

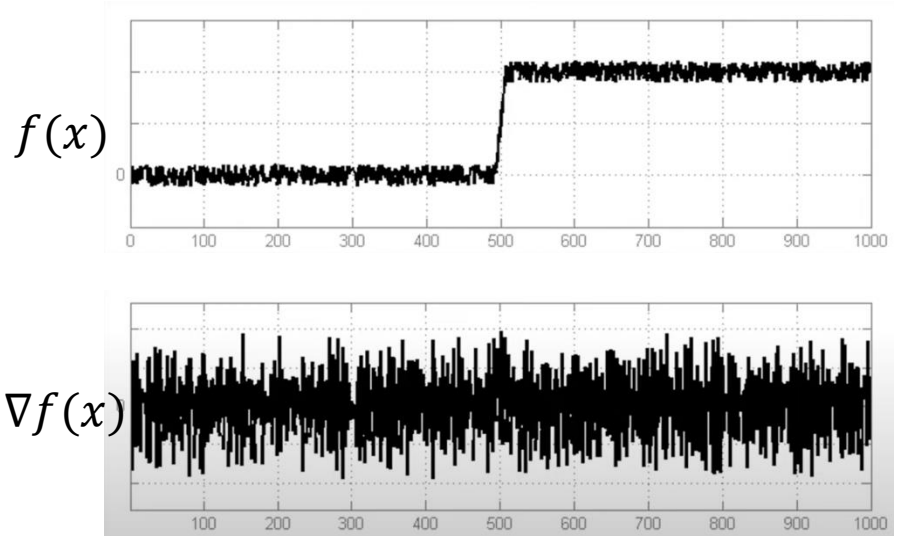
- ***Convolution Mask***:

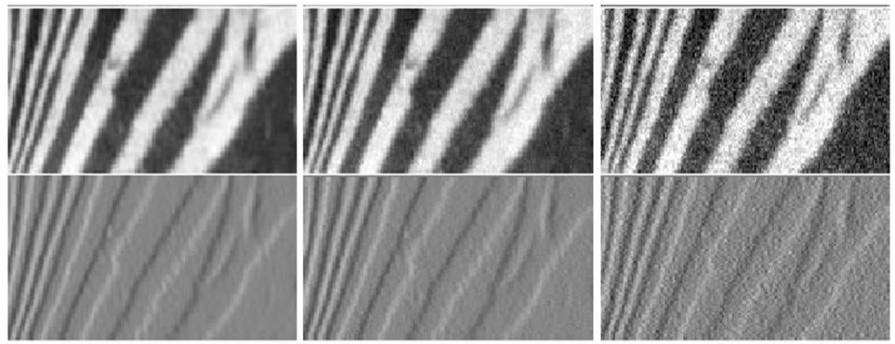
	Prewitt	Sobel
$\frac{\partial I}{\partial x}$	$\begin{bmatrix} -1 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
$\frac{\partial I}{\partial y}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$



Edge Detection – Finite Difference Filter

- Strong response to fast change → sensitive to noise

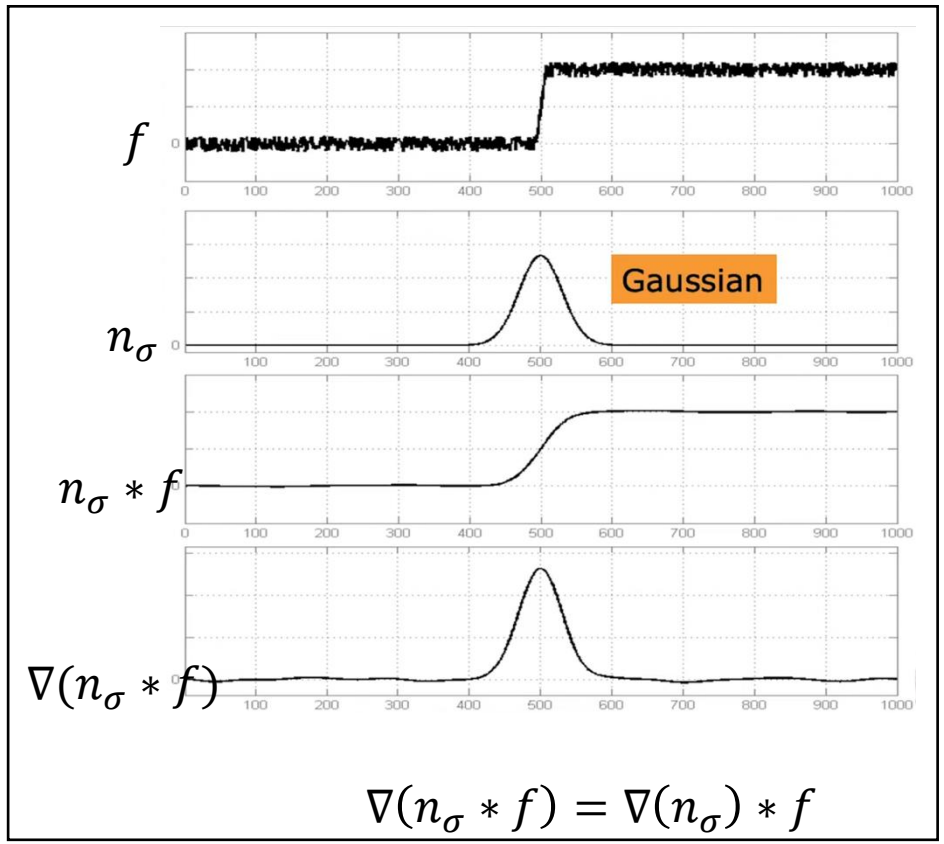


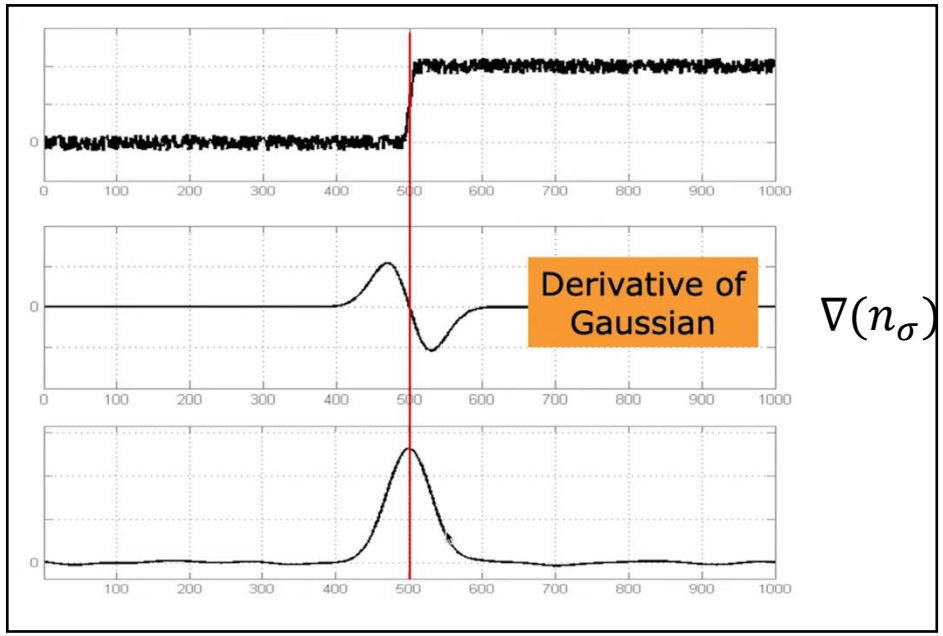


- Can we do better?

Edge Detection – Finite Difference Filter

- ***Idea.*** **Smooth** before differentiation!





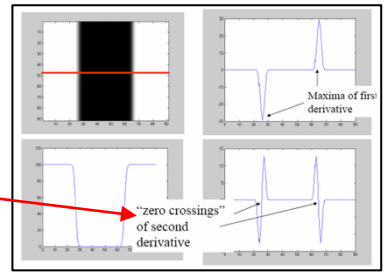
Edge Detection – Finite Difference Filter

- Laplacian (2nd derivative)
  - Recall: ”Zero crossings”
  - Finite difference approximations:

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{1}{\varepsilon^2}(I_{i,j+1} - 2I_{i,j} + I_{i,j-1})$$

$$\frac{\partial^2 I}{\partial y^2} \approx \frac{1}{\varepsilon^2}(I_{i+1,j} - 2I_{i,j} + I_{i-1,j})$$

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$



$I_{i-1,j-1}$	$I_{i-1,j}$	$I_{i-1,j+1}$
$I_{i,j-1}$	$I_{i,j}$	$I_{i,j+1}$
$I_{i+1,j-1}$	$I_{i+1,j}$	$I_{i+1,j+1}$

ε

- ***Convolution Mask***:

$$\nabla^2 \approx \frac{1}{\varepsilon^2}$$

0	1	0
1	−4	1
0	1	0

or

$$\nabla^2 \approx \frac{1}{6\varepsilon^2}$$

1	4	1
4	−20	4
1	1	1

More accurate!

Edge Detection – Finite Difference Filter

- Gradient vs. Laplacian

Gradient	Laplacian
Provides **location**, **magnitude**, and **direction** of the edge	Provides only **location** of the edge
Detection using Maxima **Thresholding**	Detection based on Zero-crossing
- **Non-linear** operation - Requires two convolutions	- **Linear** operation - Requires only one convolution

Let's play with different kernels – You can also define your customized ones!

- Try this online interactive visualization: https://visualize-it.github.io/image_filters/simulation.html
- Still, all kernel parameters must be **manually specified in advance**.
- In contrast, deep neural networks learn parameters from data.

SOUTH DAKOTA STATE UNIVERSITY

Correlation

$$g(x, y) = f \star K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x+u, y+v)\mathbf{K}(u, v)$$

Convolution

h x h kernel K

$$g(x, y) = f * K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x-u, y-v)\mathbf{K}(u, v)$$

Image source: https://www.oreilly.com/library/view/hands-on-image-processing/9781789343731/2e5f00c5-9e07-44fc-9632-fb347b95a487.xhtml

Correlation vs. Convolution

How do we compute the filtering results in the previous slides?

- Linear filter: an output pixel's value is a weighted sum of pixel values within a small neighborhood

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l). \qquad (3.12)$$

- The entries in the weight *kernel* or *mask h(k, l)* are often called the *filter coefficients*.
- The Eq. (3.12) is called **correlation** operator. It can be also denoted as

$$g = f \otimes h. \qquad (3.13)$$

Reference: Computer Vision: Algorithms and Applications, 2nd ed.

Convolution

- A common variant of

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l). \tag{3.12}$$

  is

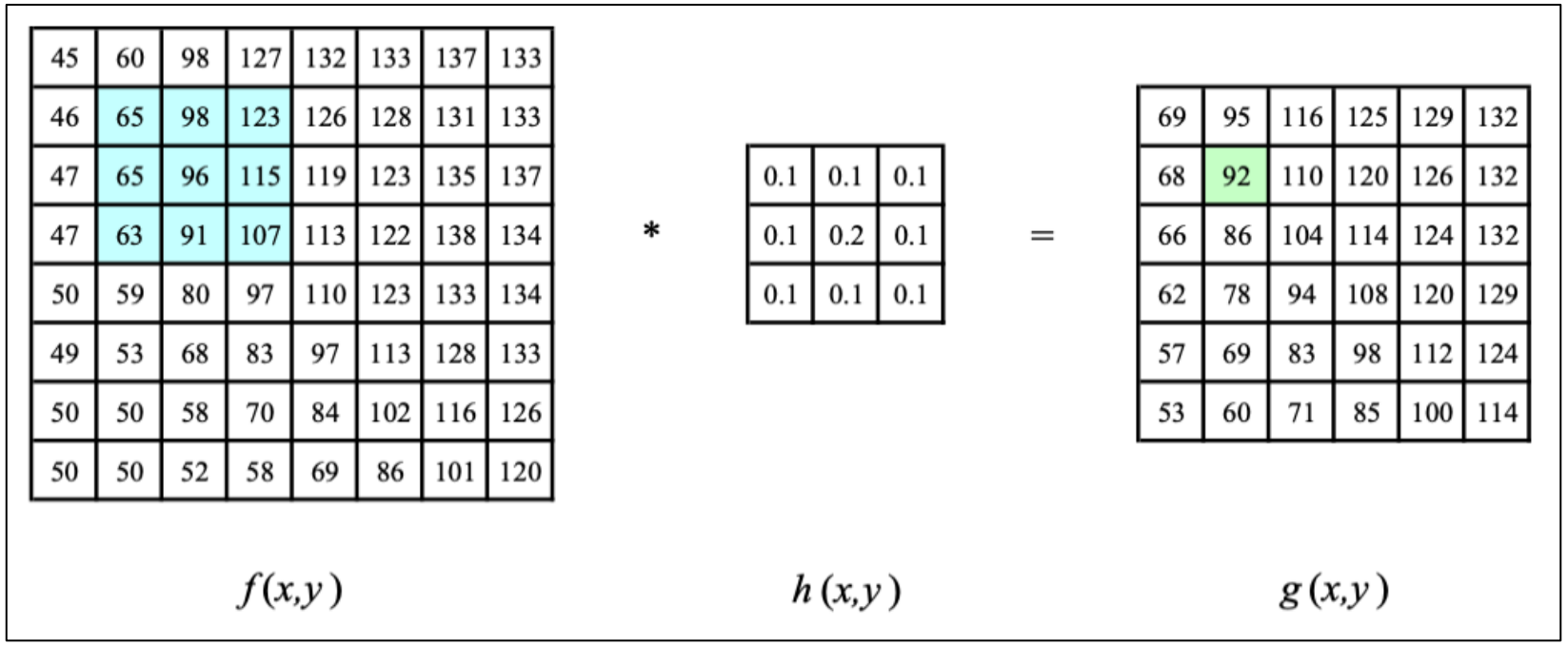
$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l), \tag{3.14}$$

  where the sign of the offsets in *f* has been reversed.

- This is called the **convolution** operator, also denoted as

$$g = f * h, \tag{3.15}$$

Recap this example...



45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

$$g(i,j) = \sum_{i,j} f(i+k, j+l)h(k,l) \quad \text{or} \quad g(i,j) = \sum_{i,j} f(i-k, j-l)h(k,l) \quad ?$$

Correlation vs. Convolution

$$g(i,j) = \sum_{i,j} f(i+k, j+l)h(k,l)$$

$$g(i,j) = \sum_{i,j} f(i-k, j-l)h(k,l)$$

- Both are linear shift-invariant (LSI) operators , satisfying

$$h \circ (f_0 + f_1) = h \circ f_0 + h \circ f_1, \tag{3.16}$$

$$g(i,j) = f(i+k, j+l) \quad \Leftrightarrow \quad (h \circ g)(i,j) = (h \circ f)(i+k, j+l), \tag{3.17}$$

which means that shifting a signal commutes with applying the operator ($\circ$ stands for the LSI operator). Another way to think of shift invariance is that the operator "behaves the same everywhere".

Correlation vs. Convolution

- Mathematically, they are the different concepts. 
- In practice, correlation and convolution are often used interchangeably in computer vision.
- This is because in many applications, the difference is negligible.
  - Many hand-crafted kernels are symmetric
  - In deep learning, the kernel weights are learned, so flipping or not flipping makes no difference after training.
- In this course (including exams), we will not distinguish these two concepts. ☺

Boundary Effects

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

- The correlation/convolution produces a result smaller than the input. Why?
- This may not be desirable in many applications ☹

Padding

- To deal with the boundary effects, many padding models have been developed.

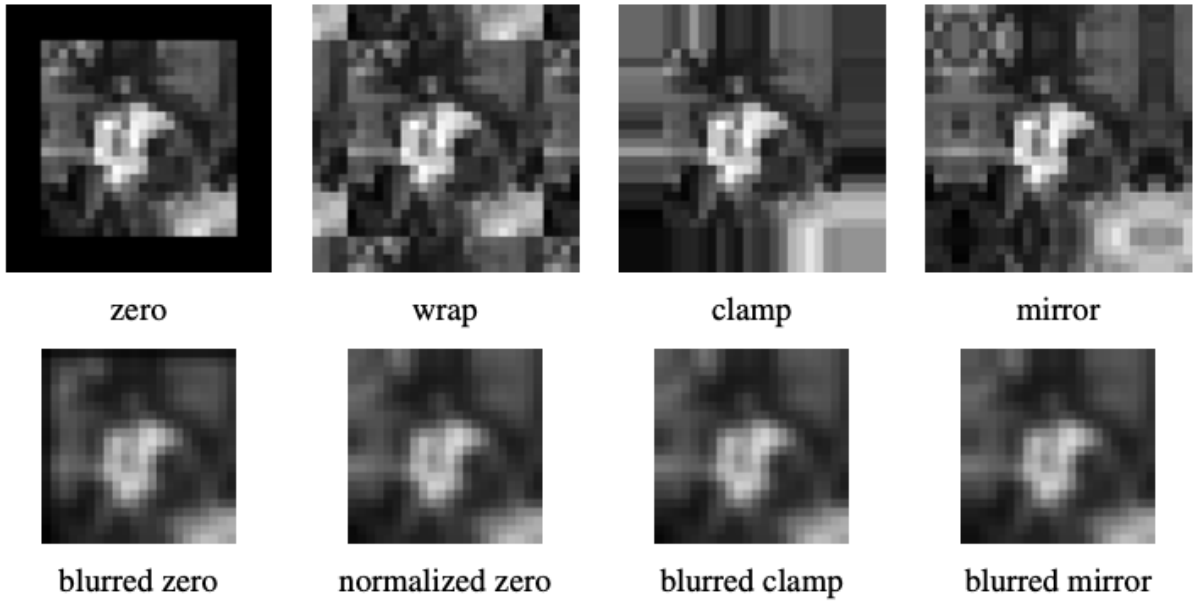


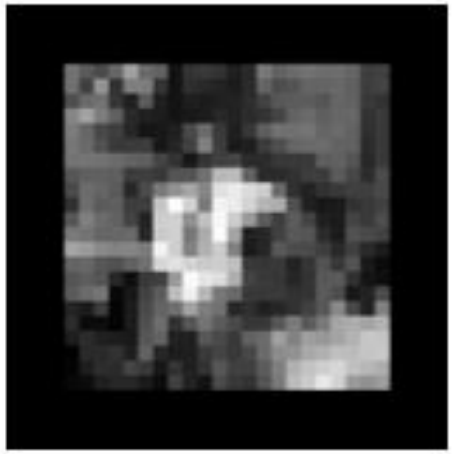
Figure 3.13 *Border padding (top row) and the results of blurring the padded image (bottom row). The normalized zero image is the result of dividing (normalizing) the blurred zero-padded RGBA image by its corresponding soft alpha value.*

Reference: Computer Vision: Algorithms and Applications, 2nd ed.

Zero Padding and Constant Padding

- Zero padding: set all pixels outside the source image to 0
- Usage: simple, common in CNN to keep output size, but can introduce constant-value artifacts.
- Constant padding: set all pixels outside the source image to a specified border value. (Zero padding is a special case)

0	0	0	0	0
0	0	0	0	0
0	0	30	60	90
0	0	120	150	180



zero

blurred zero

Reference:
- Computer Vision: Algorithms and Applications, 2nd ed.
- https://www.mathworks.com/help/images/imfilter-boundary-padding-options.html

Clamp (replicate or clamp to edge)

- Definition: repeat edge pixels indefinitely.
- Usage: help reduce boundary artifacts such as edges and contrast compared to padding with zeros or a constant.

30	30	30	60	90
30	30	30	60	90
30	30	30	60	90
120	120	120	150	180

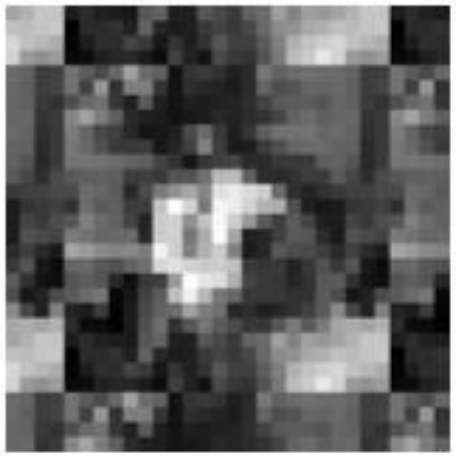
clamp

blurred clamp

Reference:
- Computer Vision: Algorithms and Applications, 2nd ed.
- https://www.mathworks.com/help/images/imfilter-boundary-padding-options.html

(Cyclic) Wrap (Repeat or Tile)

- Definition: loop “around” the image in a “toroidal” configuration.
- The padded pixels are copied from the opposite side of the image. The effect gives an appearance of a tiled or looped image.
- Specify this value when you want to treat an image as periodically repeating, such as when you are performing filtering in the frequency domain.



wrap

60	90	30	60	90
150	180	120	150	180
60	90	30	60	90
150	180	120	150	180

Reference:
- Computer Vision: Algorithms and Applications, 2nd ed.
- https://www.mathworks.com/help/images/imfilter-boundary-padding-options.html

Mirror (or Symmetric)

- Definition: reflect pixels across the image edge.
- Usage: help to remove edge contrast and maintain edge texture.

150	120	120	150	180
60	30	30	60	90
60	30	30	60	90
150	120	120	150	180

mirror

blurred mirror

Reference:
- Computer Vision: Algorithms and Applications, 2nd ed.
- https://www.mathworks.com/help/images/imfilter-boundary-padding-options.html

Compare blurred results after different paddings

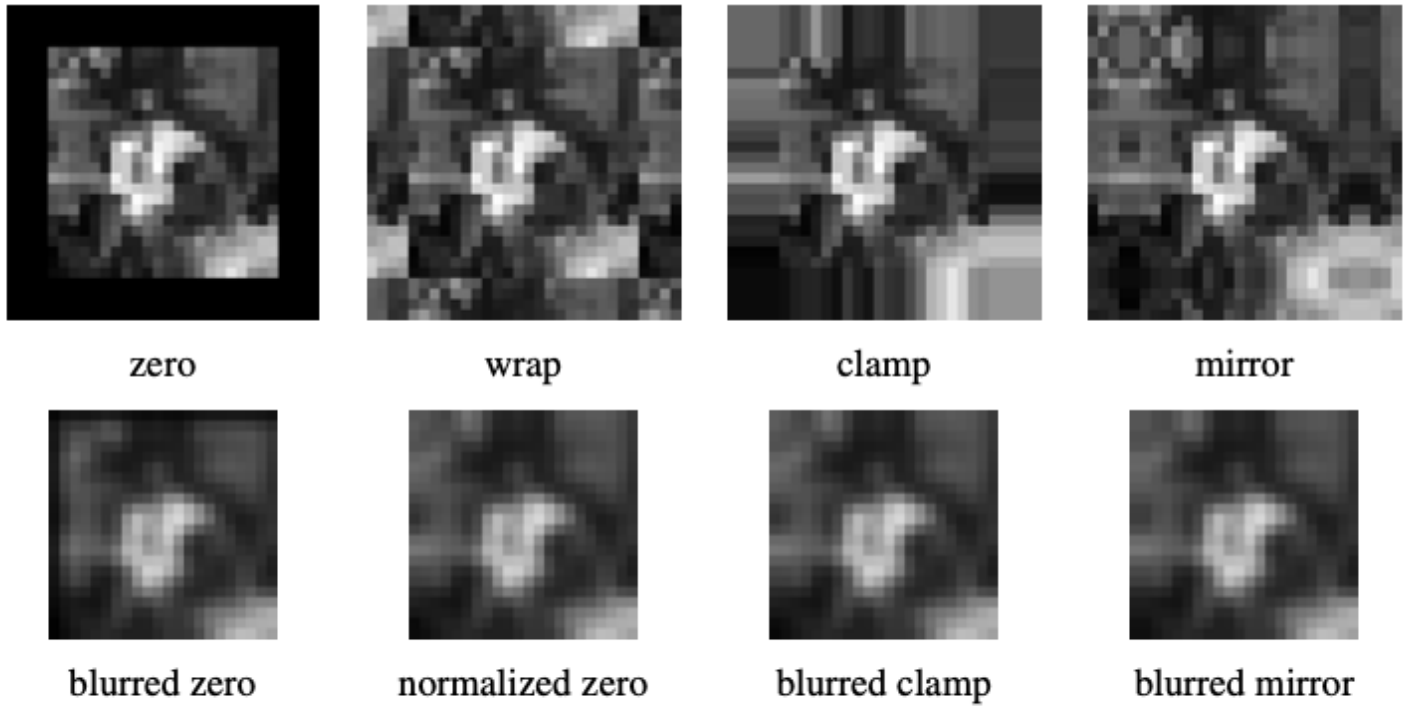


Figure 3.13 *Border padding (top row) and the results of blurring the padded image (bottom row). The normalized zero image is the result of dividing (normalizing) the blurred zero-padded RGBA image by its corresponding soft alpha value.*

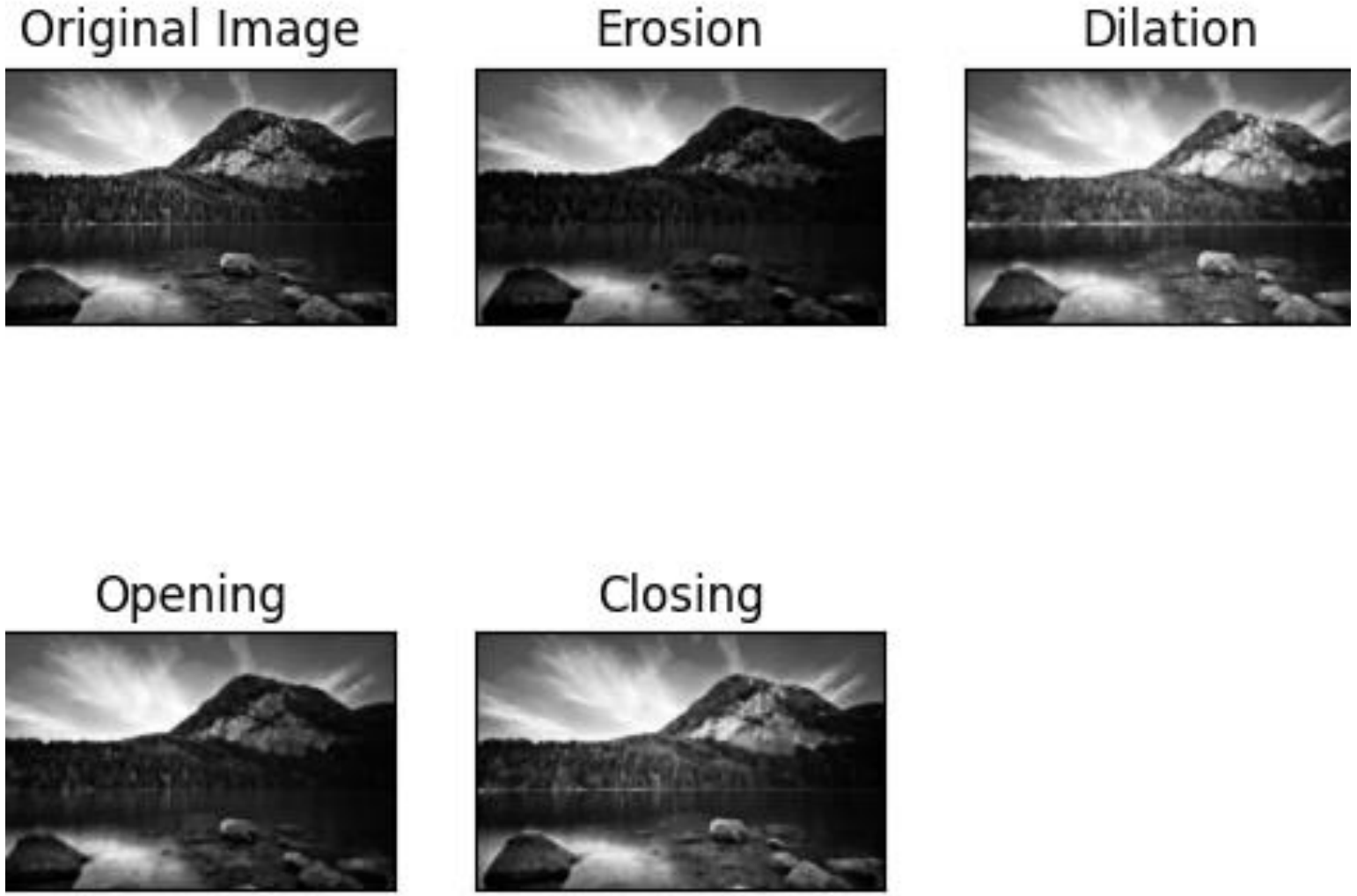
Observation:
- zero padding darkens the edges,
- clamp (replication) padding propagates border values inward,
- mirror (reflection) padding preserves colors near the borders

Reference: Computer Vision: Algorithms and Applications, 2nd ed.

Non-Linear Filtering

- The filters we have looked at so far have all been filter.
- In many cases, however, better performance can be obtained using a non-linear combination of neighboring pixels.
- Examples: median filtering, bilateral filtering, etc.
- Read *Chapter 3 of Computer Vision: Algorithms and Applications, 2nd ed* if you are interested!

South Dakota State University



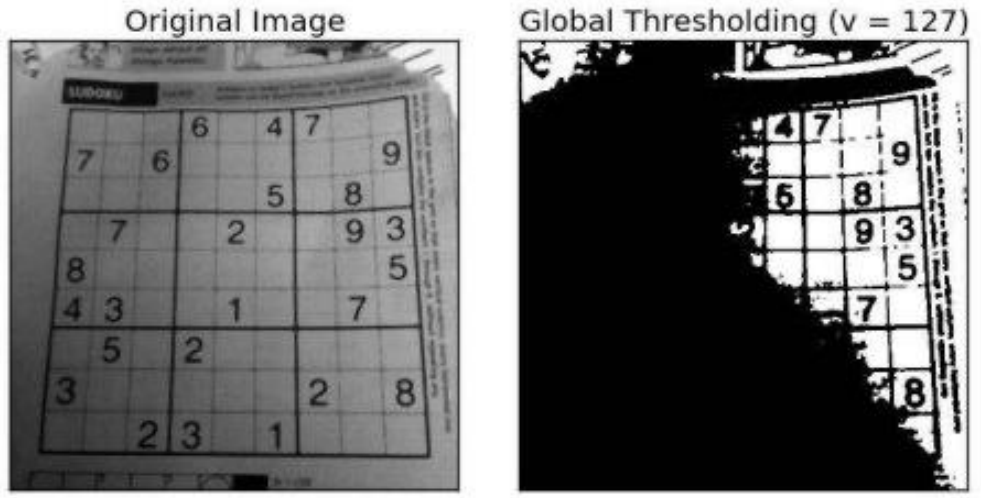
Binary Image Processing

Image source: https://www.geeksforgeeks.org/computer-vision/different-morphological-operations-in-image-processing/

Thresholding Operation

- While non-linear filters are often used to enhance grayscale and color images, they are also used extensively to process binary images.
- Such images often occur after a thresholding operation,

$$\theta(f,t) = \begin{cases} 1 & \text{if } f \geq t, \\ 0 & \text{else,} \end{cases} \qquad (3.44)$$




```
# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
```

Image source: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Reference: Computer Vision: Algorithms and Applications, 2nd ed.

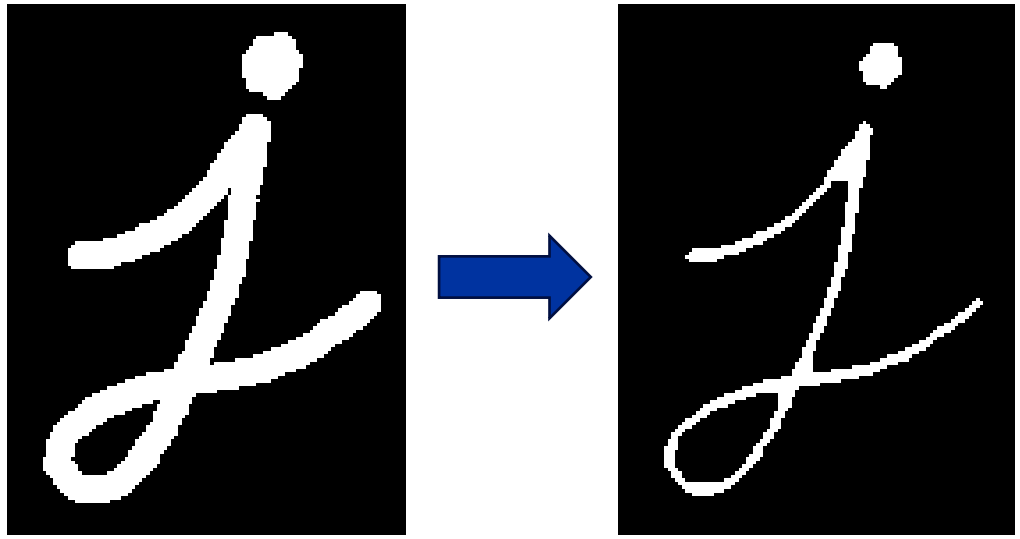
Morphological Operations

- Morphological transformations are some simple operations based on the image **shape**.
- It is normally performed on binary images.
- It needs two inputs, one is our original image, second one is called **structuring element** or **kernel** which decides the nature of operation.

Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

Morphological Operations: Erosion

- The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white).
- The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).



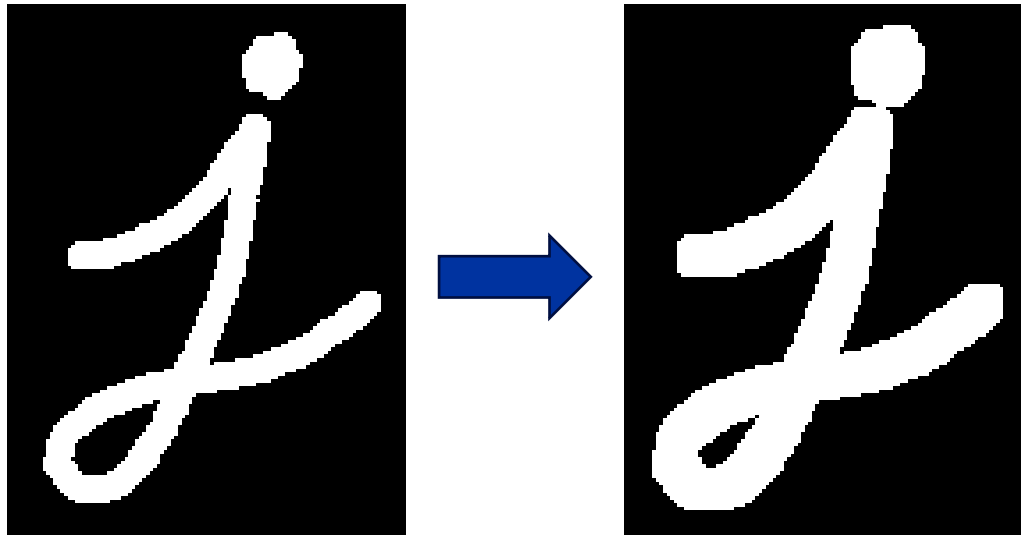
```
import cv2 as cv
import numpy as np

img = cv.imread('j.png', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
```

Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

Morphological Operations: Dilation

- It is just opposite of erosion.
- Here, a pixel element is '1' if at least one pixel under the kernel is '1'.



```
dilation = cv.dilate(img,kernel,iterations = 1)
```

Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

Morphological Operations: Opening

- Opening is just another name of **erosion followed by dilation**.
- It is useful in removing noise.

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

Morphological Operations: Closing

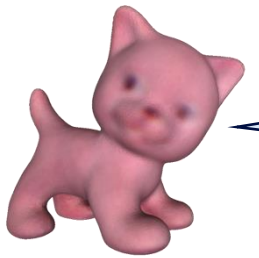
- Closing is reverse of Opening, **Dilation followed by Erosion**.
- It is useful in closing small holes inside the foreground objects, or small black points on the object.

```
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

Morphology for Data Preprocessing

Manually labelled segmentation masks may contain noises. We can use morphological operations to preprocessing the annotated masks before training segmentation models.



Examples from Linemod Occlusion Dataset.
https://bop.felk.cvut.cz/datasets/#LM-O

Takeaway

- Linear Filtering vs. Non-Linear Filtering?
- Correlation vs. Convolution?
- Morphological Operations?