CSC 422/522
Computer Vision and Pattern Recognition

Filtering (Convolution) and Feature Detection

2026 Spring

South Dakota State University

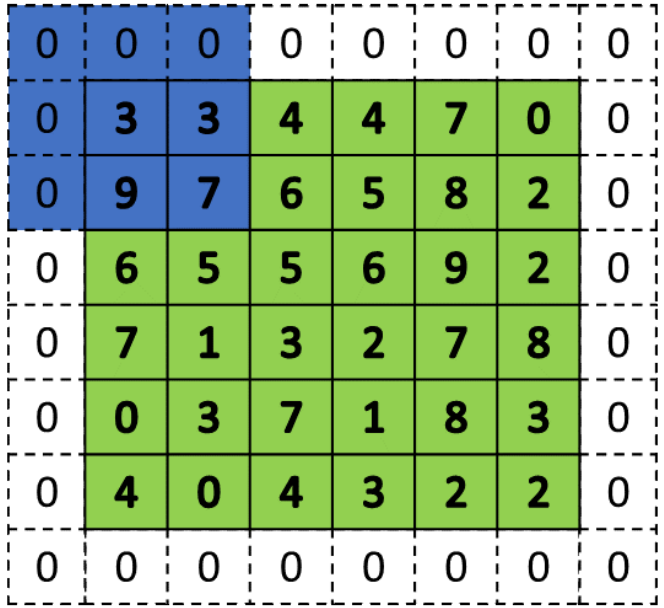
Today

- Boundary Effects and Padding
- Binary Image Processing
- Features vs. Pixels
- How to detect edges?

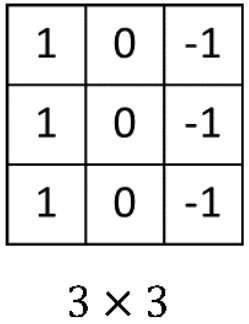
South Dakota State University

Boundary Effects and Padding

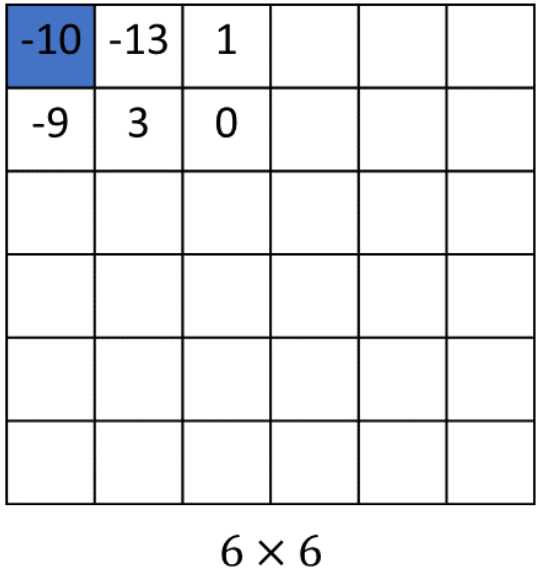
Image source: https://datahacker.rs/what-is-padding-cnn/



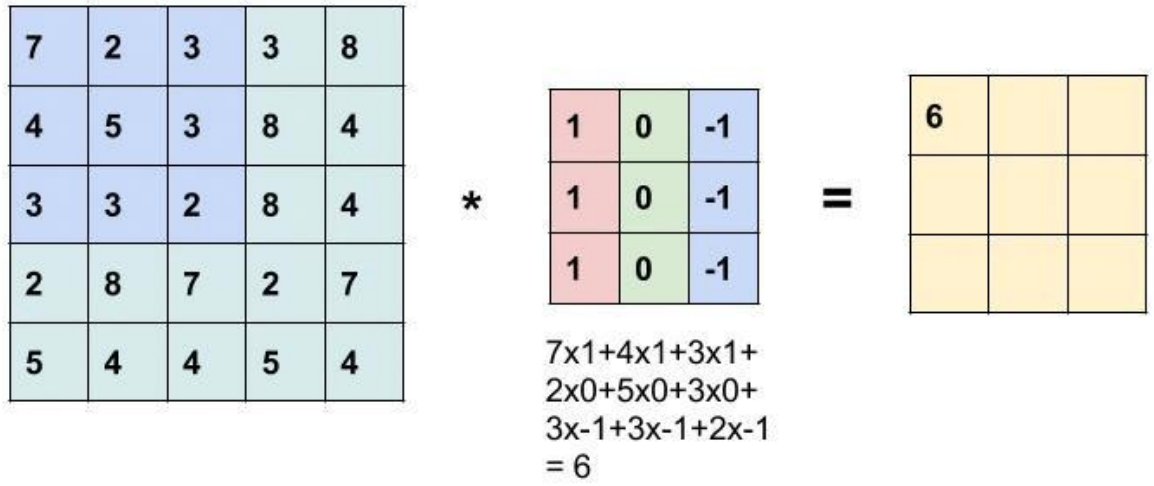
*



=



Recap: Convolution



Animation source: https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

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.

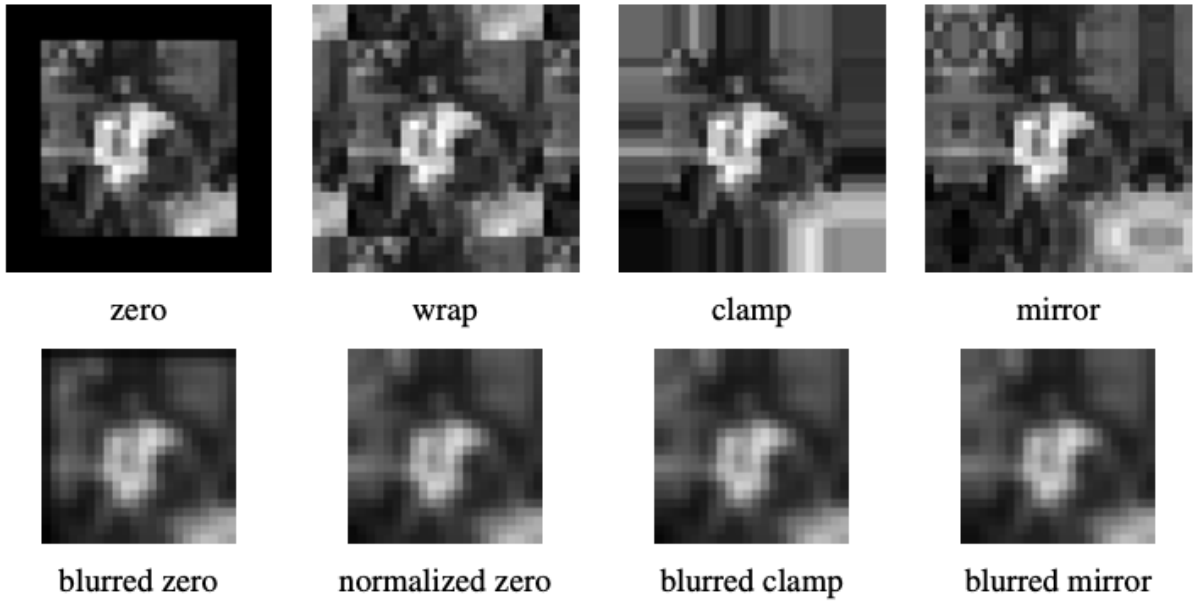


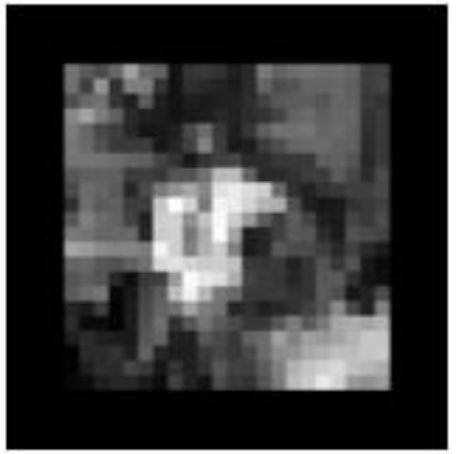
Figure 3.13 *Border padding (top row) and the results of blurring the padded image (bottom row). The normalized zero image is the result of dividing (normalizing) the blurred zero-padded RGBA image by its corresponding soft alpha value.*

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

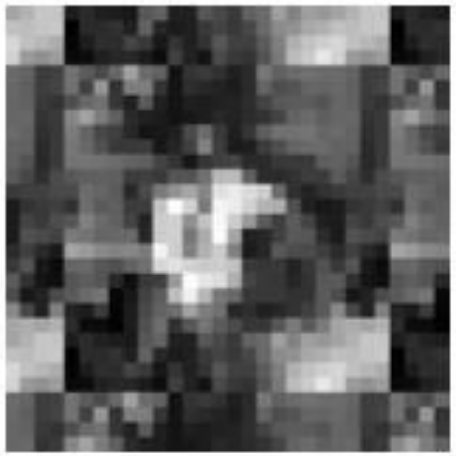
clamp

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

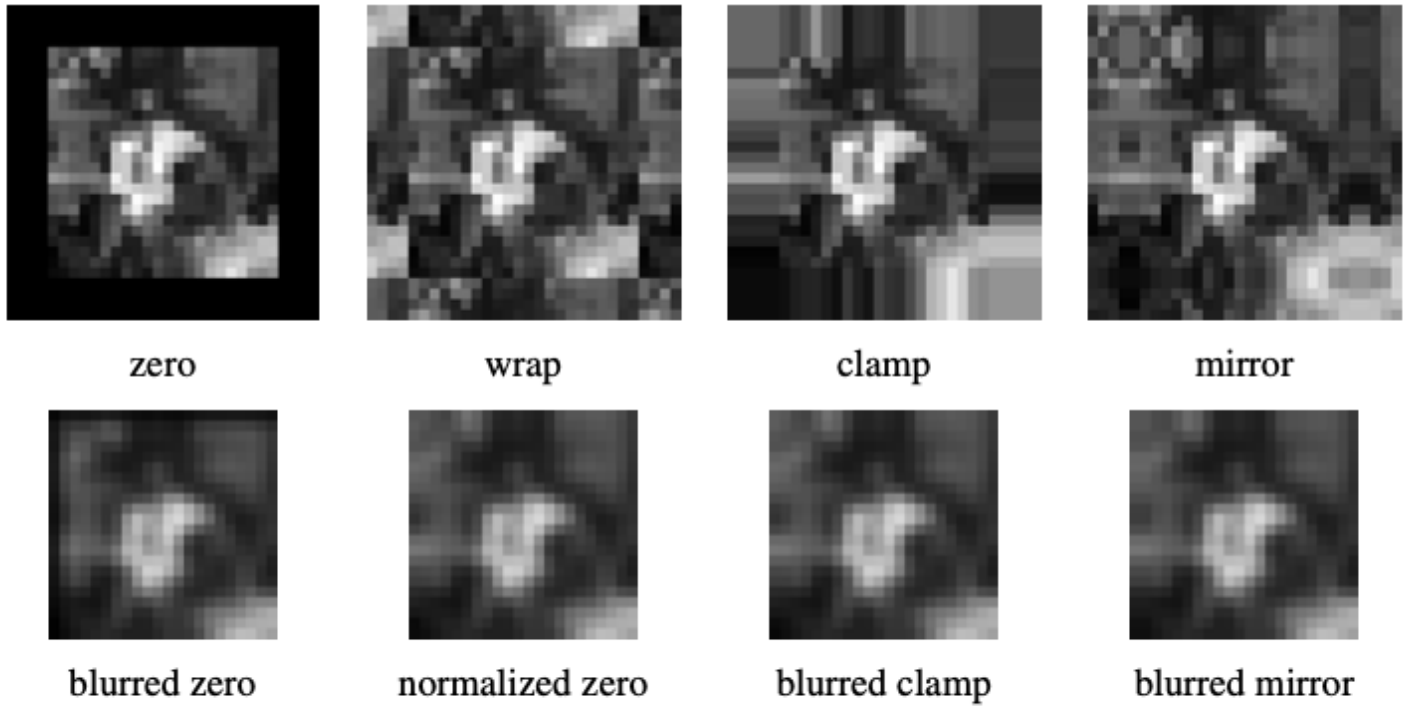
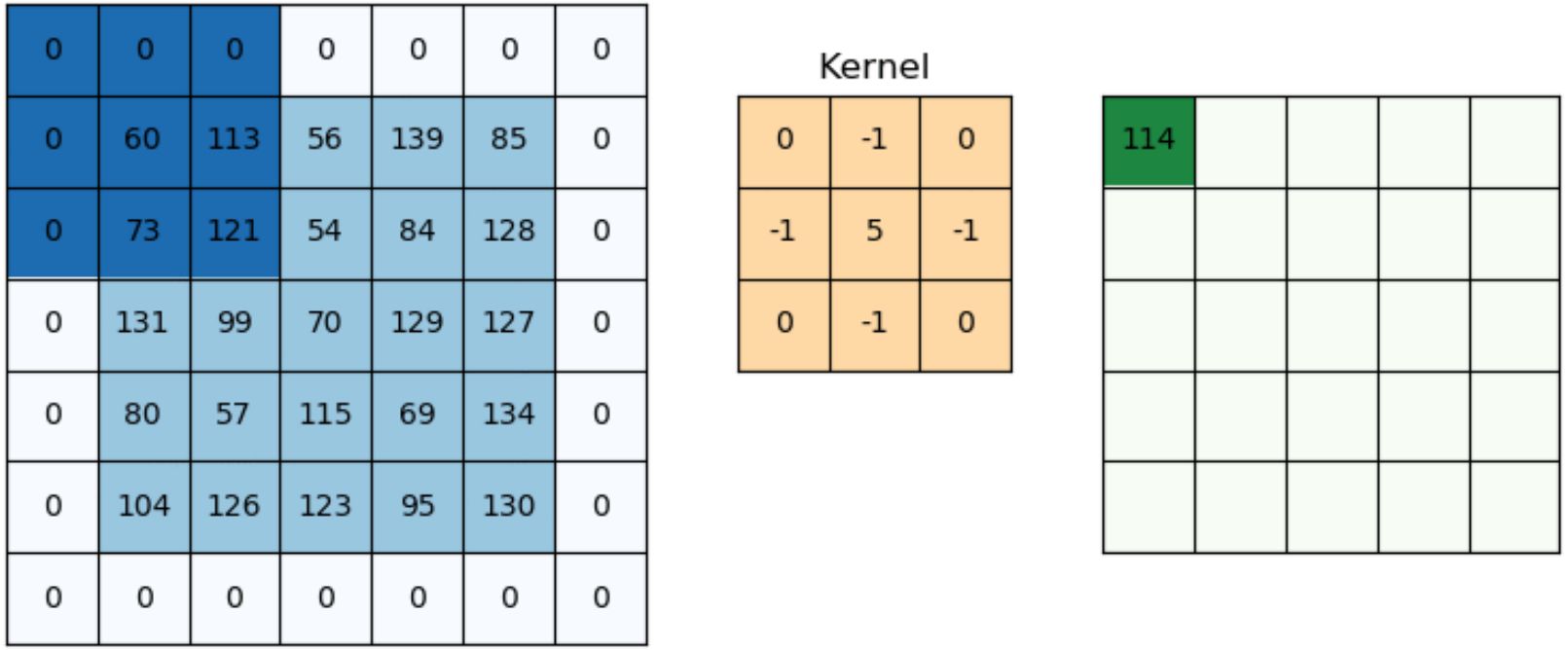


Figure 3.13 *Border padding (top row) and the results of blurring the padded image (bottom row). The normalized zero image is the result of dividing (normalizing) the blurred zero-padded RGBA image by its corresponding soft alpha value.*

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

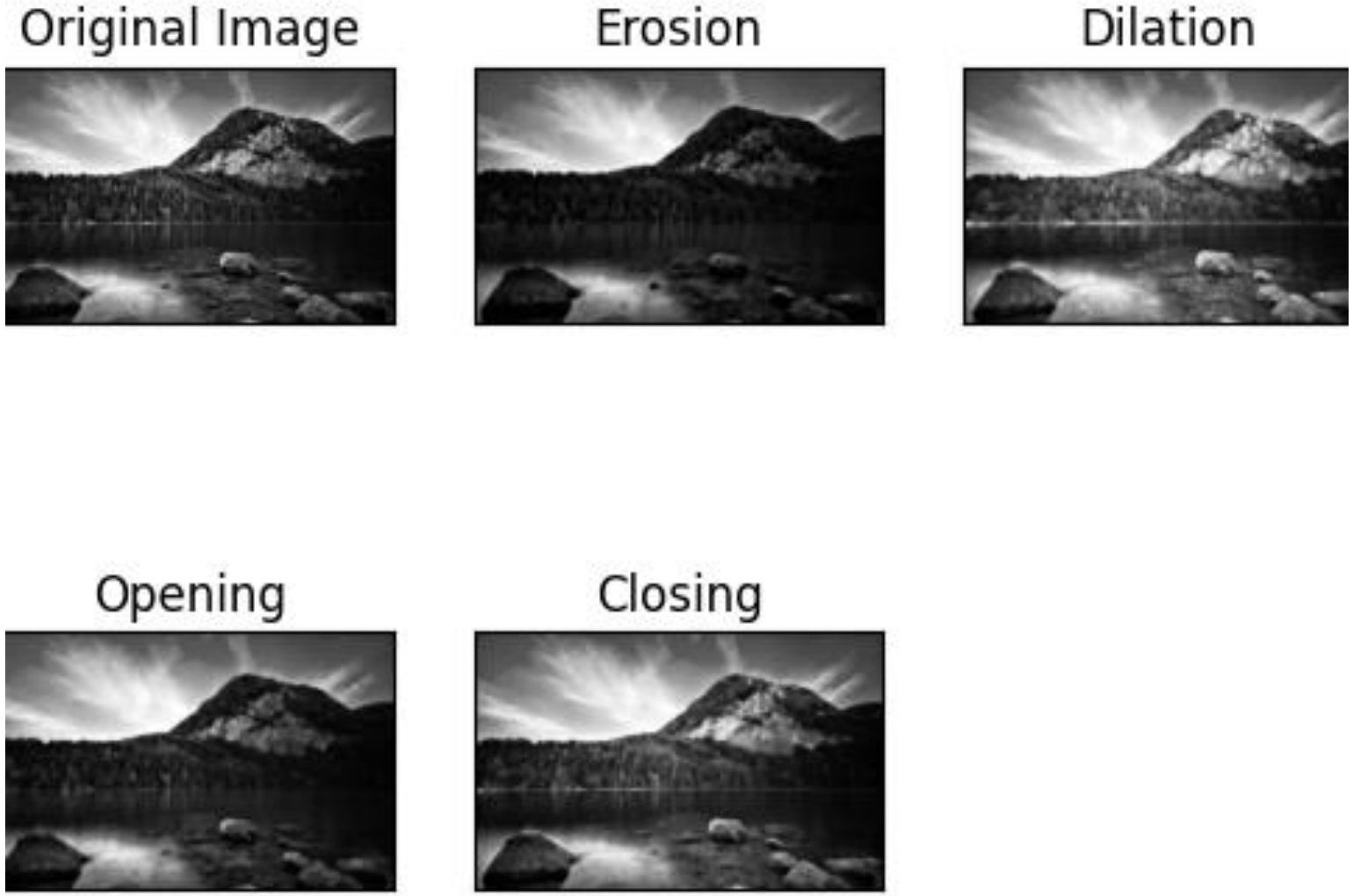
Reference: Computer Vision: Algorithms and Applications, 2nd ed.

Animation: Convolution with Zero Padding



Animation source: https://medium.com/@draj0718/zero-padding-in-convolutional-neural-networks-bf1410438e99

South Dakota State University



Binary Image Processing

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

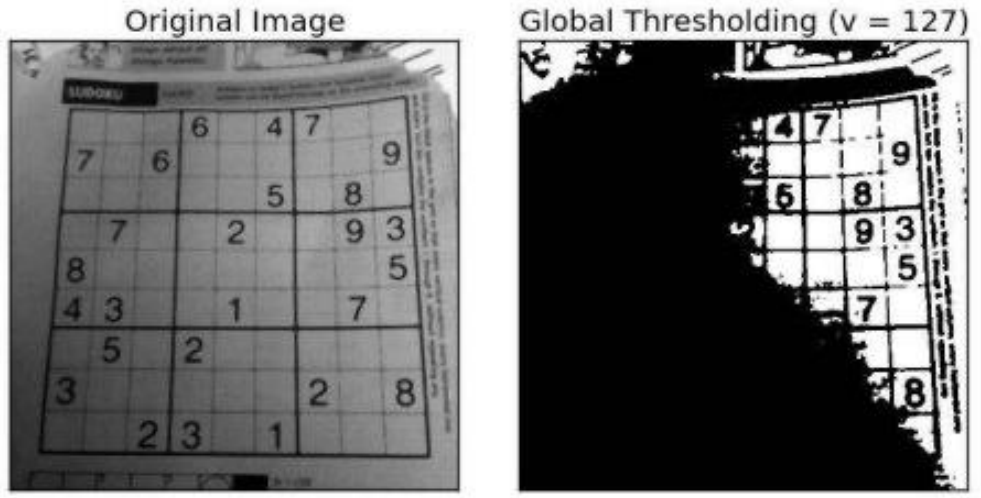
Non-Linear Filtering

- The filters we have looked at so far have all been linear, i.e., each output pixel is a weighted summation of neighborhood pixels.
- 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!

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.

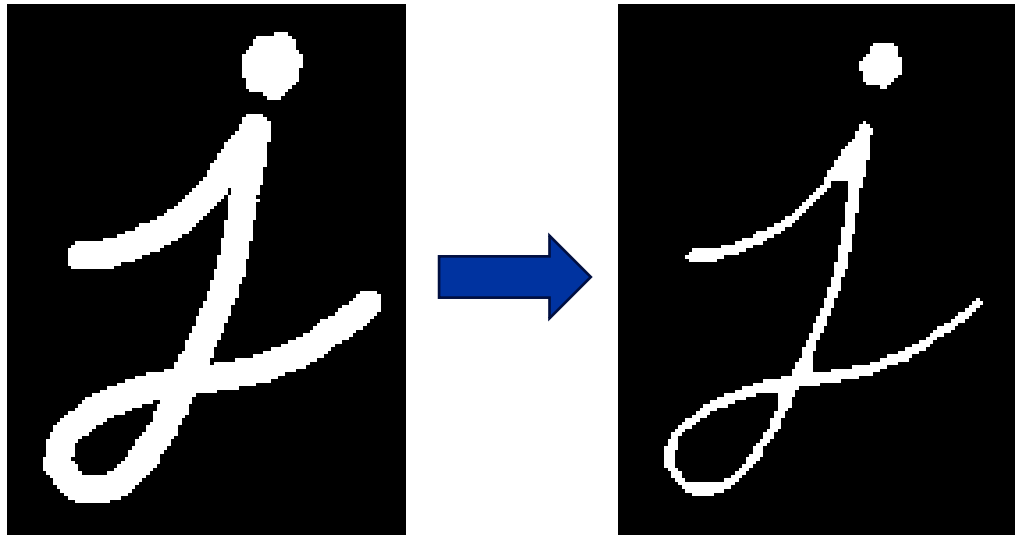
Morphological Operations

- 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).

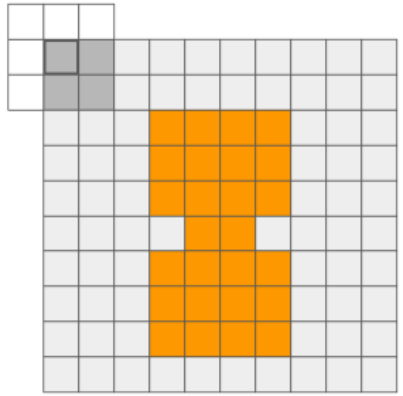


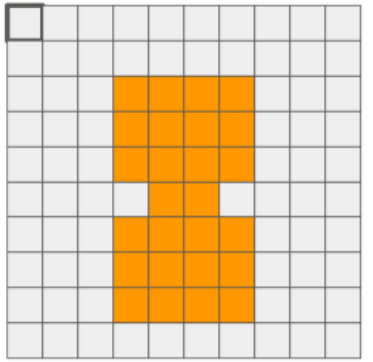
```
import cv2 as cv
import numpy as np

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: Erosion

- 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).

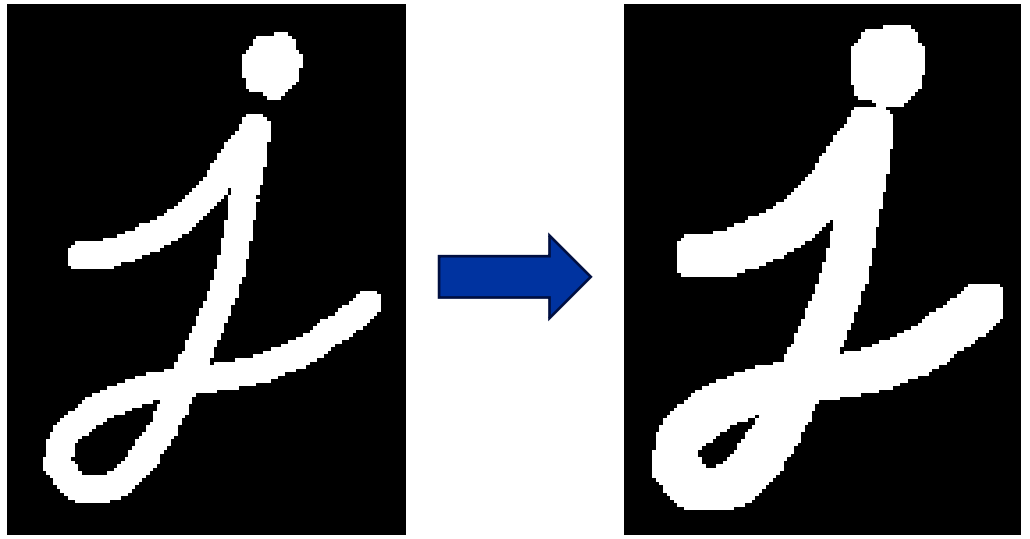




Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
Visualization found: https://penny-xu.github.io/blog/mathematical-morphology

Morphological Operations: Dilation

- It is just opposite of erosion.

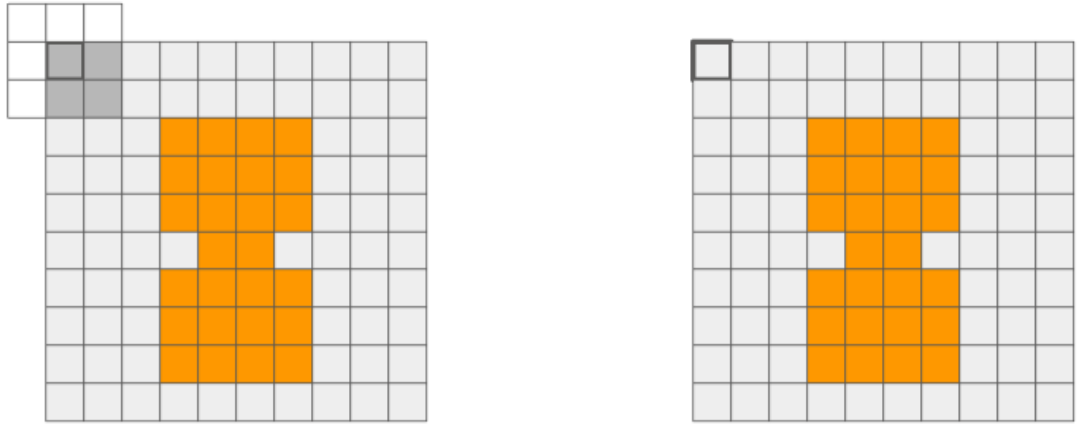


```
dilation = cv.dilate(img,kernel,iterations = 1)
```

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

Morphological Operations: Dilation

- Here, a pixel element is '1' if at least one pixel under the kernel is '1'.



Reference: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
Visualization found: https://penny-xu.github.io/blog/mathematical-morphology

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

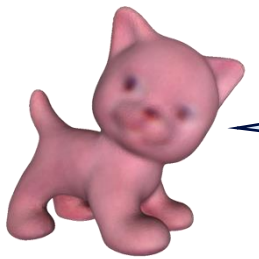
- Closing is reverse of Opening, **Dilation followed by Erosion**.
- It is useful in closing small holes inside the foreground objects, or small black points on the object.

```
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 consider using morphological operations to preprocessing the annotated masks before training segmentation models.



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

South Dakota State University

Why Features Matter in Computer Vision?

Image source: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html

Let's Play a Game: Jigsaw Puzzle!

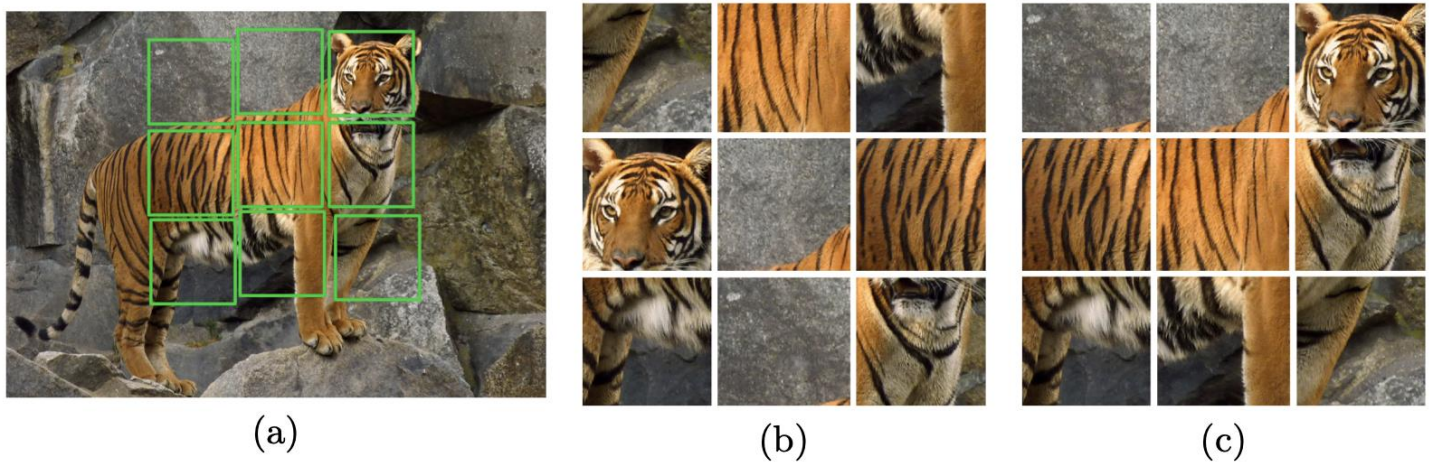
- Link: https://onlinejigsawpuzzles.net/puzzle.php?image=images/puzzle/Spring-in-farmland-Sweden-old-bar--in-rural-surroundings.jpg#
- Note: you can change the number of pieces
- Think: How do you solve the puzzle? What visual cues help you place pieces?

Jigsaw Puzzles in Computer Vision Research

- In a jigsaw puzzle, we reconstruct a global image from local pieces.
- During the process, we may consider edge continuity, texture similarity, color consistency, etc.
- Computer vision aims to automate the same reasoning process.
- Example research papers: Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (ECCV 2016, paper link: https://arxiv.org/abs/1603.09246)

Screenshot for https://onlinejigsawpuzzles.net/puzzle.php?image=images/puzzle/Spring-in-farmland-Sweden-old-bar--in-rural-surroundings.jpg#

Jigsaw Puzzles in Computer Vision Research



(a) (b) (c)

Fig. 1: Learning image representations by solving Jigsaw puzzles. (a) The image from which the tiles (marked with green lines) are extracted. (b) A puzzle obtained by shuffling the tiles. Some tiles might be directly identifiable as object parts, but others are ambiguous (*e.g.*, have similar patterns) and their identification is much more reliable when all tiles are jointly evaluated. In contrast, with reference to (c), determining the relative position between the central tile and the top two tiles from the left can be very challenging [10].

Example research papers: Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (ECCV 2016, paper link: https://arxiv.org/abs/1603.09246)

What is a Feature in Computer Vision?

- A feature in computer vision is an identifiable part of an image or video that conveys meaningful information for tasks like object detection, tracking, or classification.
- Features are often distinct patterns, such as edges, corners, textures, or specific shapes, that algorithms use to understand and analyze visual data.

Texts adapted from https://milvus.io/ai-quick-reference/what-is-a-feature-in-computer-vision

Screenshot for https://onlinejigsawpuzzles.net/puzzle.php?image=images/puzzle/Spring-in-farmland-Sweden-old-bar--in-rural-surroundings.jpg#

Why Not Just Use Pixels?

- Pixels are sensitive to noise, lighting variations, viewpoint changes, etc.
- Therefore, pixels are not stable representations for robust computer vision tasks.

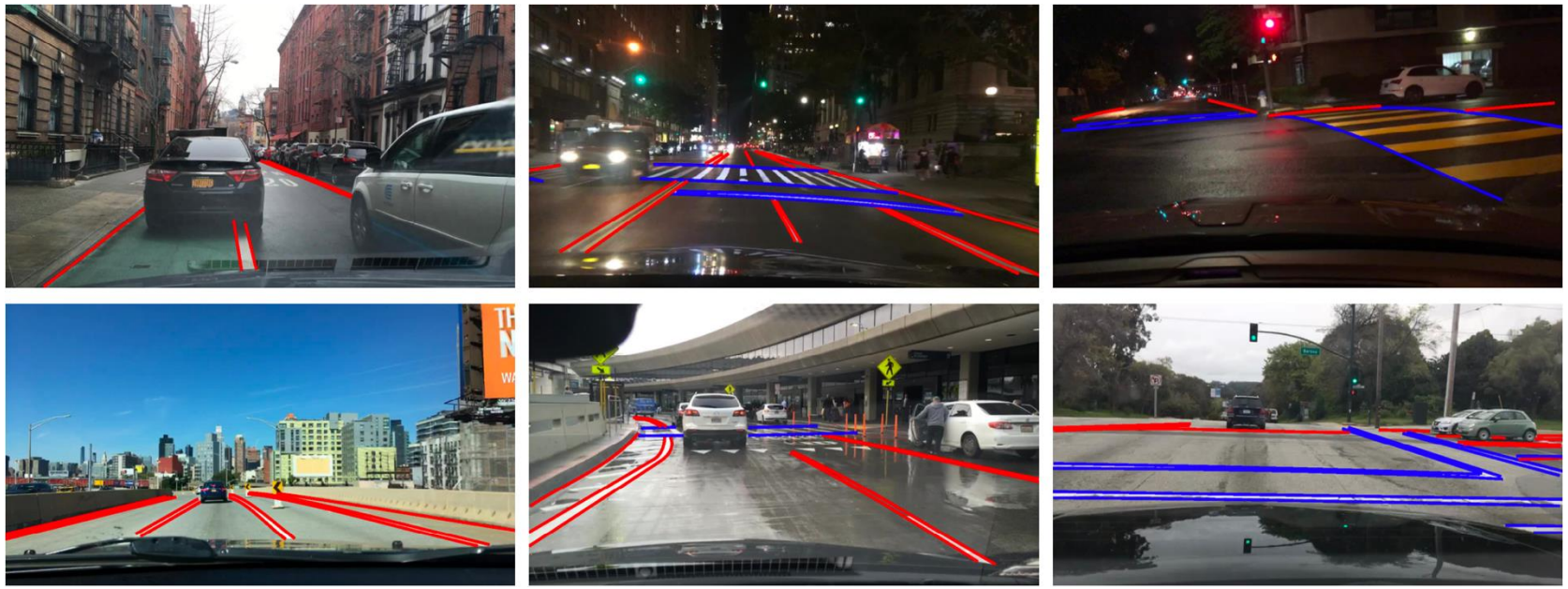
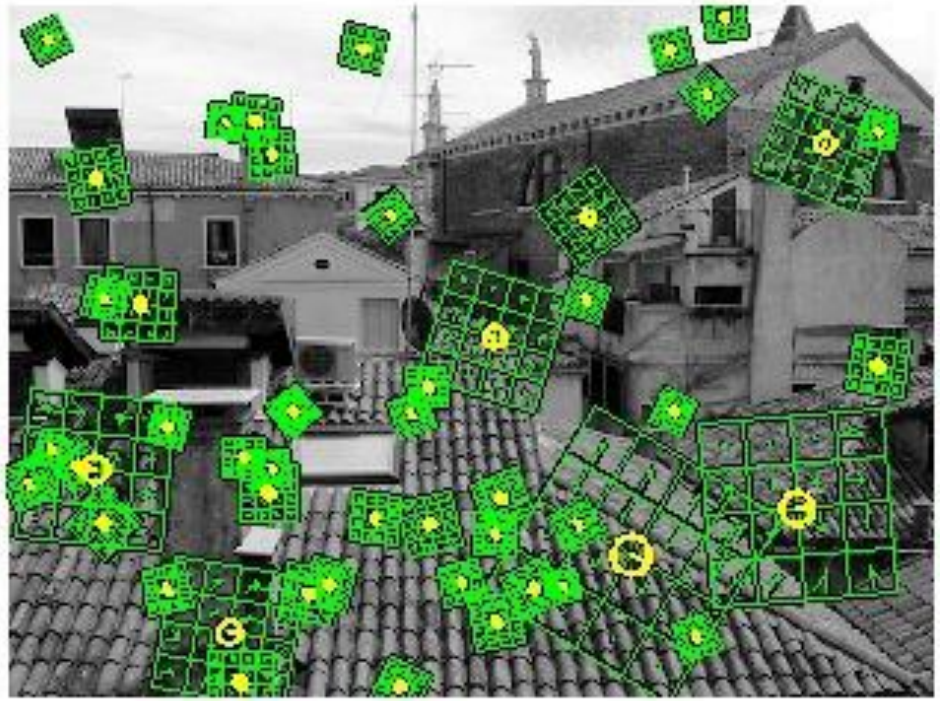


Image source: BDD100K: A Large-scale Diverse Driving Video Database
https://bair.berkeley.edu/blog/2018/05/30/bdd/

Classical (Hand-Crafted) Features

- Traditional feature extraction methods rely on mathematical techniques to identify and describe these key points.
- Algorithms like SIFT (Scale-Invariant Feature Transform) detect stable features by analyzing gradients, corners, or blobs in an image.



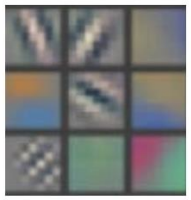
Texts adapted from https://milvus.io/ai-quick-reference/what-is-a-feature-in-computer-vision

Image found at https://ics.uci.edu/~majumder/VC/211HW3/vlfeat/doc/overview/sift.html

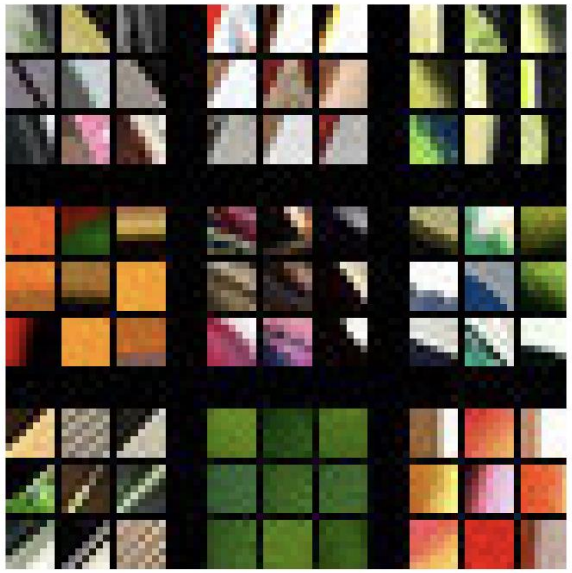
Learned Features

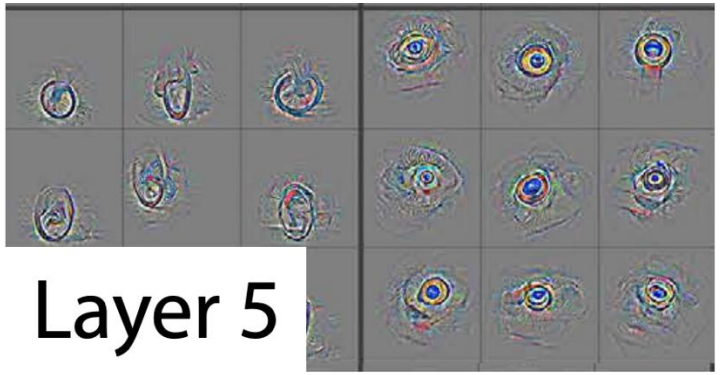
- Modern approaches, particularly deep learning, automate feature extraction using convolutional neural networks (CNNs).
- In CNNs, layers learn hierarchical features directly from data.
- Early layers detect simple patterns like edges or color gradients, while deeper layers combine these to recognize complex shapes or objects.

“Visualizing and Understanding Convolutional Networks” (ECCV 2014)



Layer 1





Layer 5

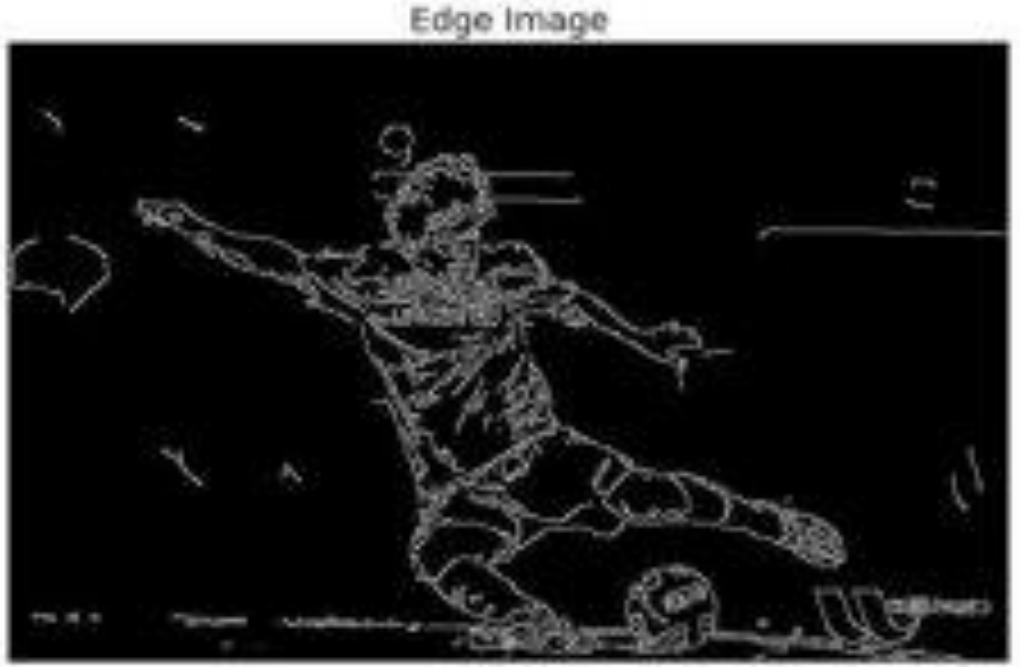
Check full Figure 2 in the original paper.
You can read the paper in this link:
https://arxiv.org/pdf/1311.2901

South Dakota State University

Feature Detection I: Edge Detection

Credits: some mathematical formulations taken from http://vision.stanford.edu/teaching/cs131_fall1718/files/05_edges.pdf with some modifications! Because this document explicitly differentiate convolution and correlation.

Image source: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html



Edges are about changes…

- Derivatives in 1D

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

- Discrete derivative in 1D

$$\frac{df}{dx} = \frac{f(x) - f(x-1)}{1} = f'(x)$$

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$$

Types of Discrete derivative in 1D (Approximation!)

Backward $$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$$

Forward $$\frac{df}{dx} = f(x) - f(x+1) = f'(x)$$

Central $$\frac{df}{dx} = f(x+1) - f(x-1) = f'(x)$$

1D discrete derivate filters

Input: [f(x-1), f(x), f(x+1)]

- Backward filter:

$$f(x) - f(x-1) = f'(x)$$

Kernel: [-1 1 0]

- Forward:

$$f(x) - f(x+1) = f'(x)$$

Kernel: [0 1 -1]

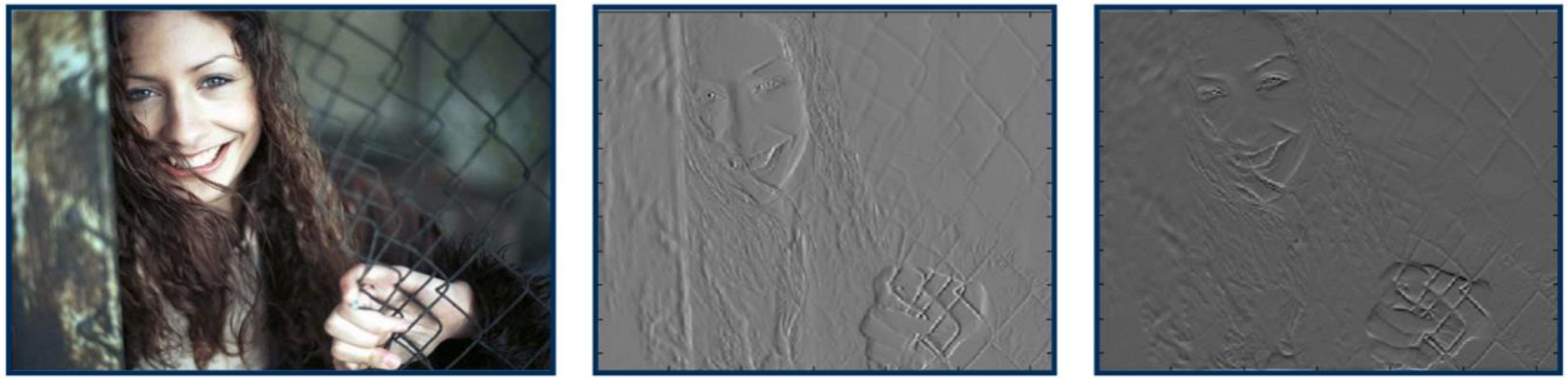
- Central:

$$f(x+1) - f(x-1) = f'(x)$$

Kernel: [-1 0 1]

3x3 image gradient filters

$$\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

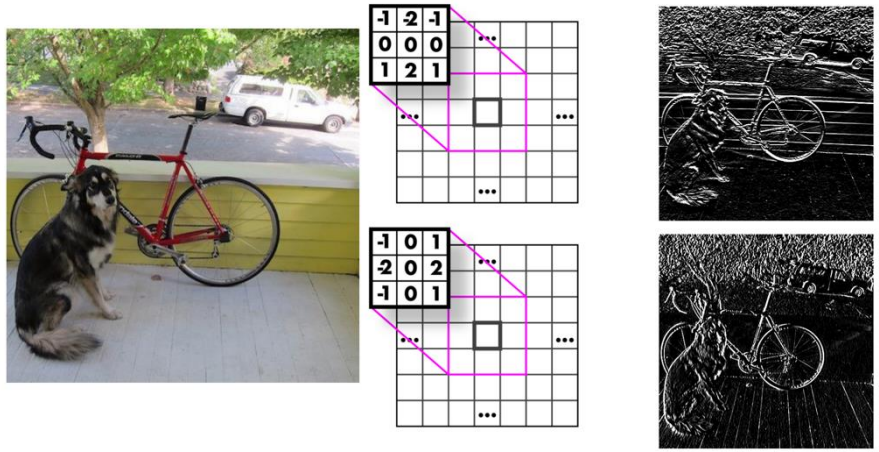


Visualization found in
http://vision.stanford.edu/teaching/cs131_fall1718/files/05_edges.pdf

Edge Detection – Finite Difference Filter

- ***Convolution Mask***:

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



Note: when a convolution kernel sums to 1 (e.g., box filter), it guarantees the overall brightness of the images stays the same. This is desired for many image processing operations.
For Prewitt and Sobel kernels, the sum is 0 → detect changes

Discrete derivate in 2D

Given function $f(x, y)$

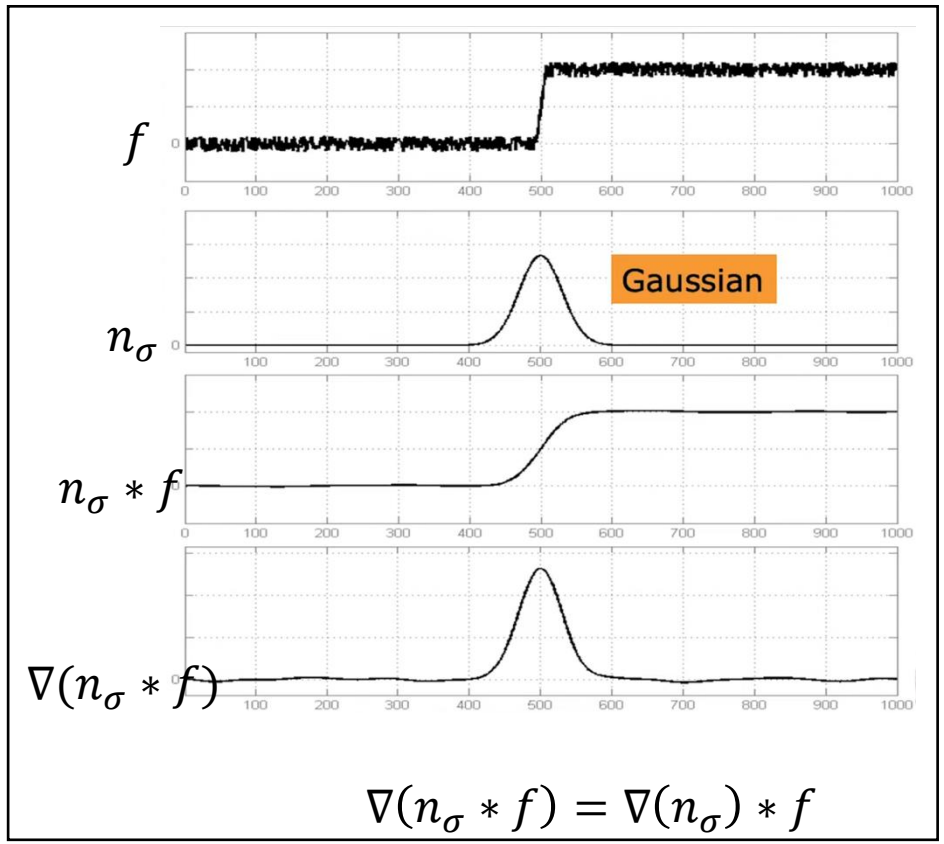
Gradient vector $$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

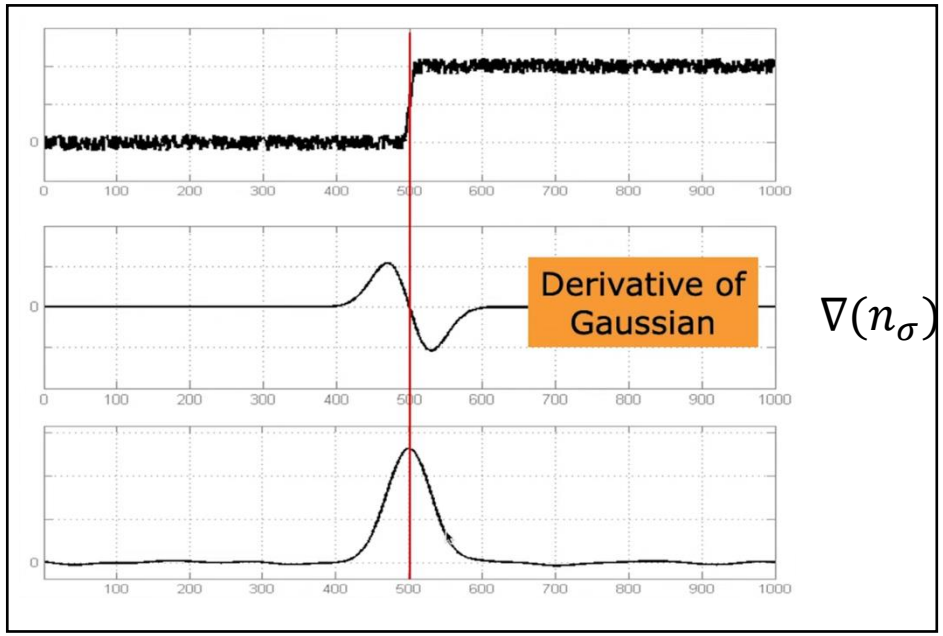
Gradient magnitude $$\left|\nabla f(x, y)\right| = \sqrt{f_x^2 + f_y^2}$$

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

Derivative of Gaussian Filter (DoG)

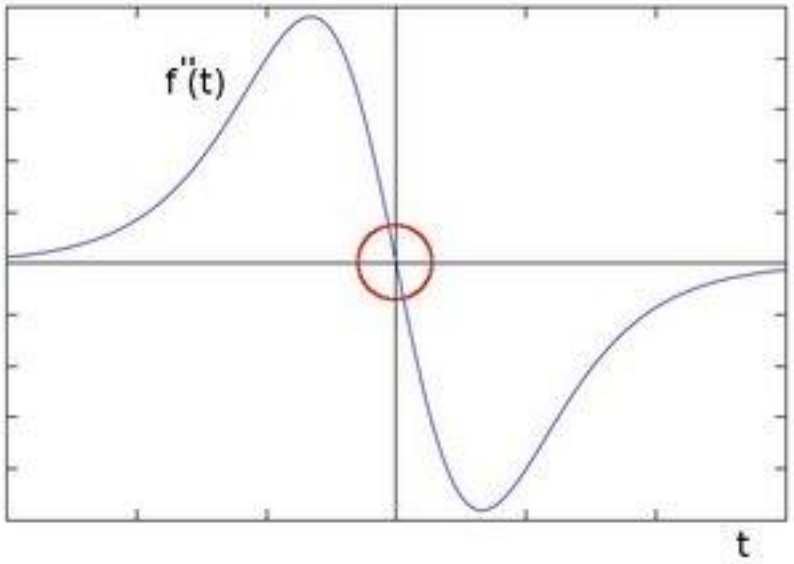
- ***Idea.*** **Smooth** before differentiation!





Laplacian Filter (Based on 2nd derivative)

1D intuition: zero crossing



2D Laplacian Operator

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

0	1	0
1	−4	1
0	1	0

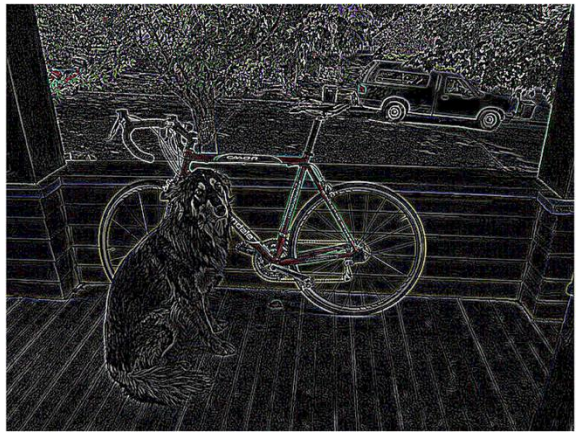
Reference: https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html

Summary: Edge Detection

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

Laplacian

$$\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$



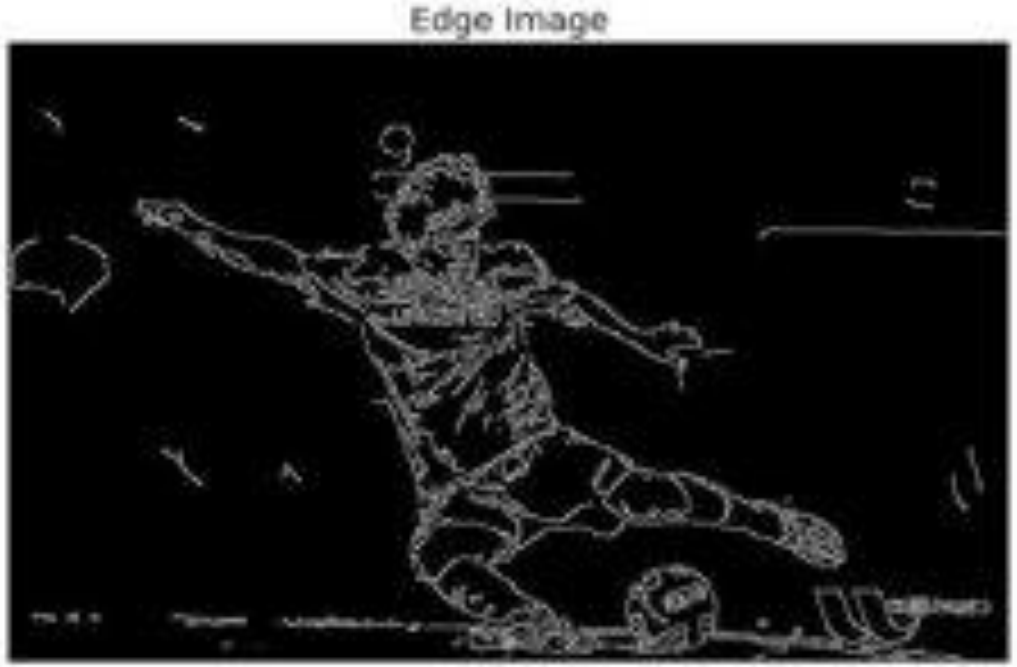
Gradient	Laplacian
Provides **location**, **magnitude**, and **direction** of the edge	Provides only **location** of the edge
Detection using Maxima **Thresholding**	Detection based on Zero-crossing
- **Non-linear** operation - Requires two convolutions	- **Linear** operation - Requires only one convolution

South Dakota State University

Canny Edge Detection

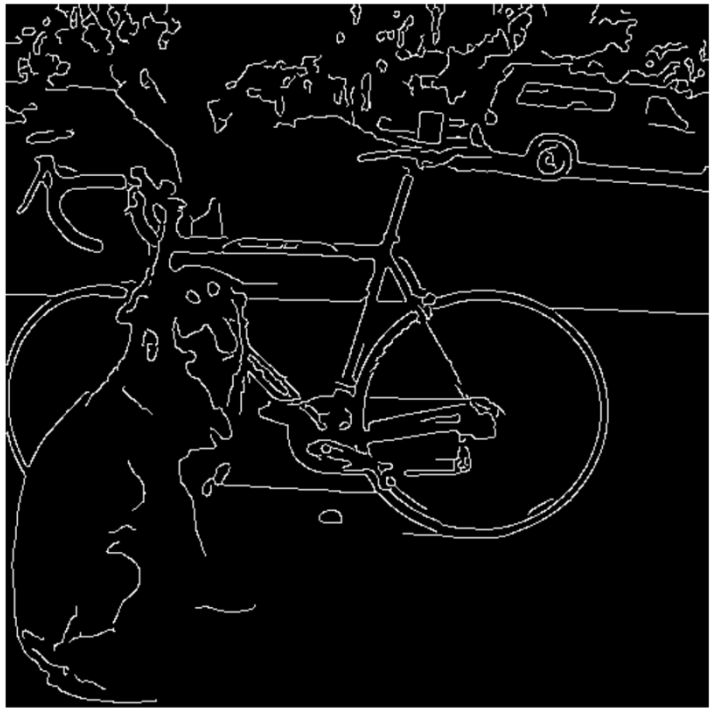
Credit: slides of this section are adapted from CSC 492/592, Spring 2024

Image source: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html



Canny Edge Detection

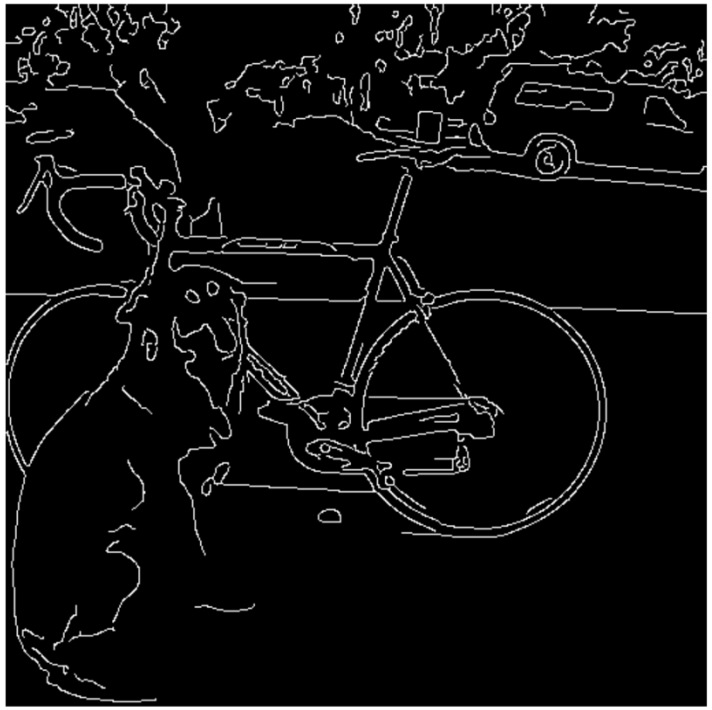
- The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.
- It was developed by John F. Canny in 1986.
- Among the edge detection methods developed so far, Canny's algorithm is one of the most strictly defined methods that provides good and reliable detection.



Reference: https://en.wikipedia.org/wiki/Canny_edge_detector

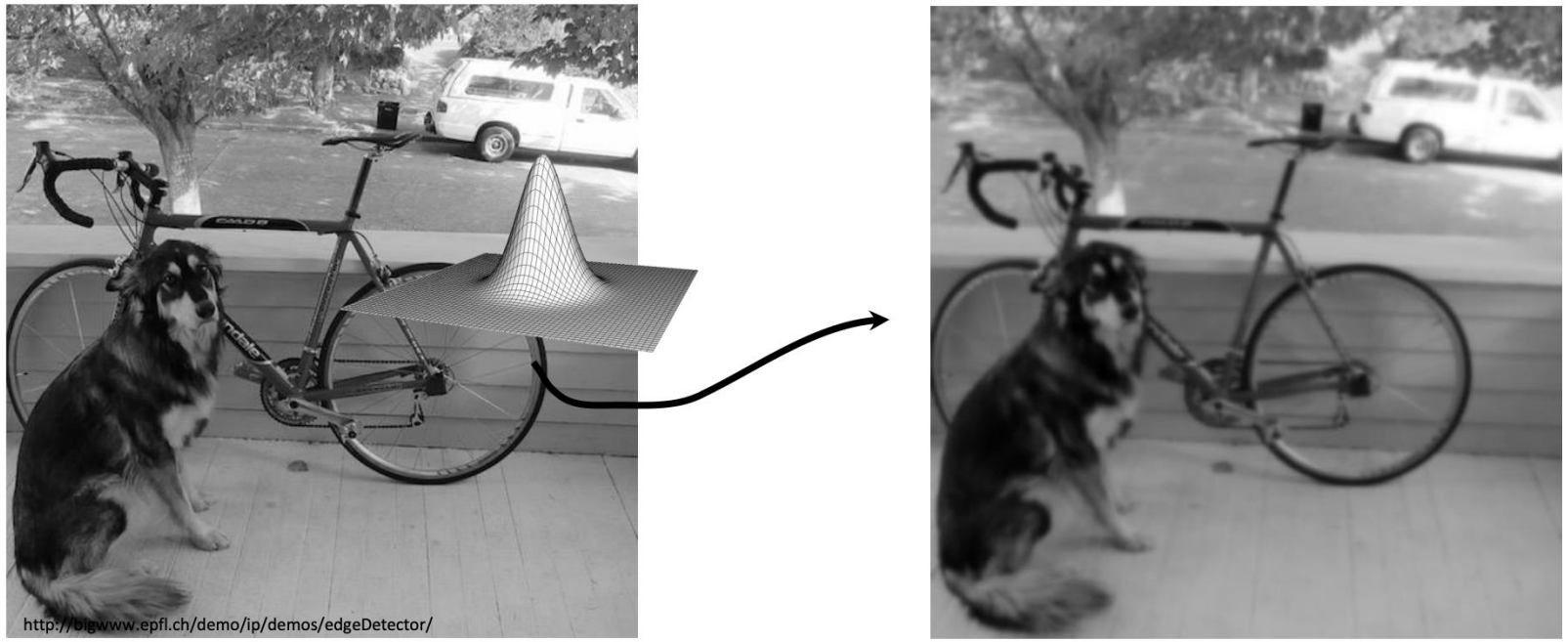
Canny Edge Detection: Algorithm

1. Smooth image
2. Calculate gradient direction and magnitude
3. Perform non-maximum suppression
4. Threshold the edges
5. Link the edges



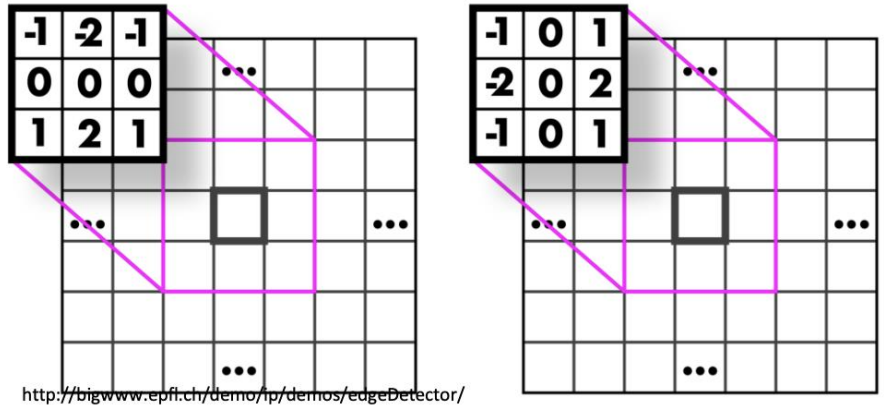
Step 1: Smooth image

Apply Gaussian filter to smooth the image to remove the noise.



Step 2: Calculate gradient direction and magnitude

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

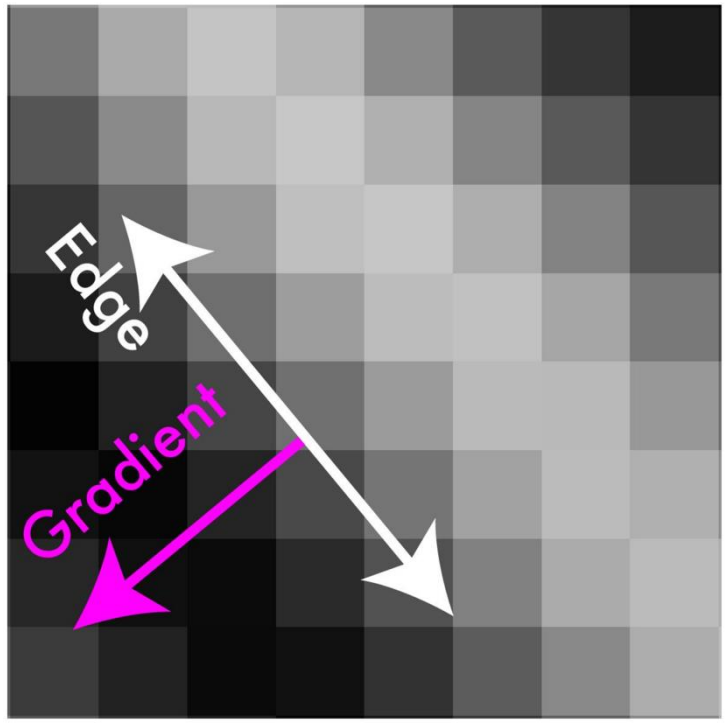


http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Step 3: Perform non-maximum suppression

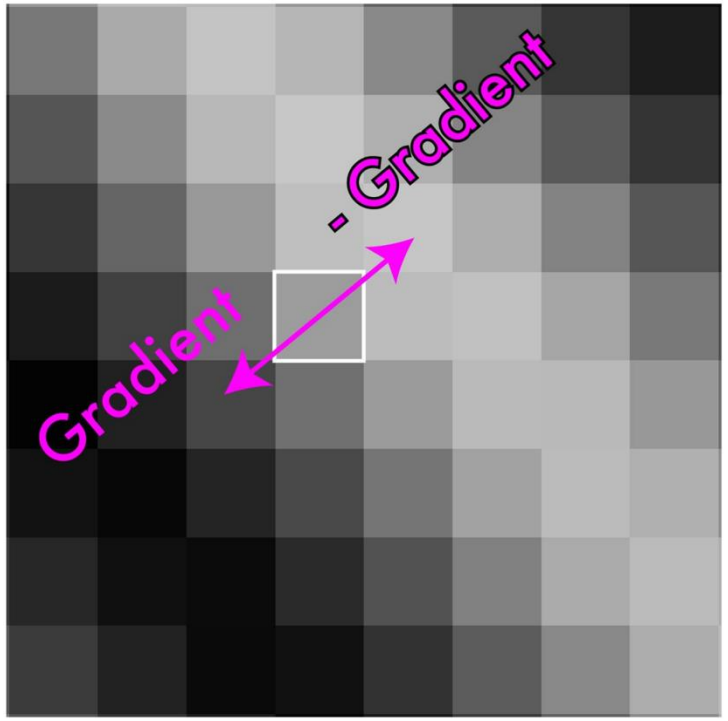
- ***Why?*** Want single pixel edges, not thick blurry lines
- ***Idea.*** Check nearby pixels and keep the high response
- ***How?*** Go through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions

Step 3: Perform non-maximum suppression



Go through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge directions.

Step 3: Perform non-maximum suppression

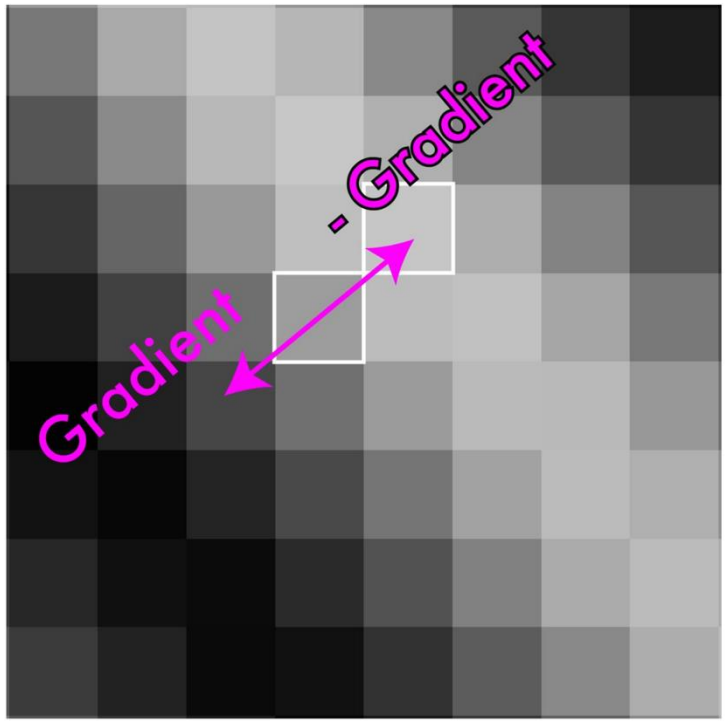


Example pixel 1:

(highlighted in white box)

compare its gradient magnitude with the magnitudes of its neighboring pixels along the gradient direction

Step 3: Perform non-maximum suppression



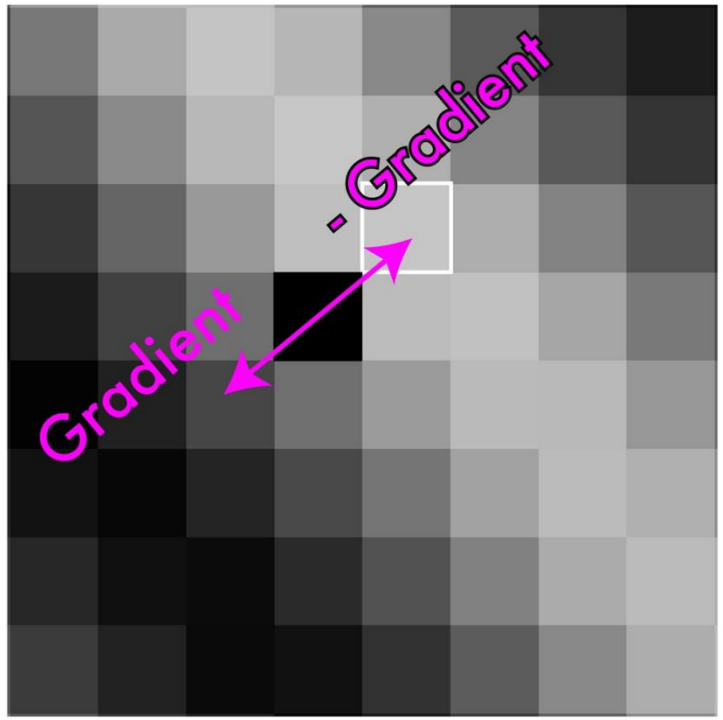
Example pixel 1:

(highlighted in left-bottom white box)

compare its gradient magnitude with the magnitudes of its neighboring pixels along the gradient direction

→ We find a neighboring pixel with larger gradient magnitude

Step 3: Perform non-maximum suppression



Example pixel 1:

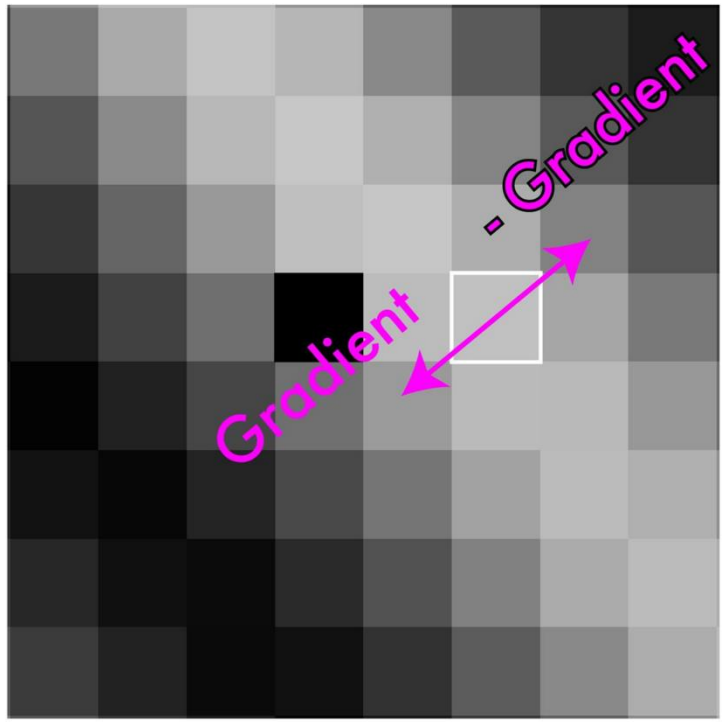
(highlighted in left-bottom white box)

compare its gradient magnitude with the magnitudes of its neighboring pixels along the gradient direction

→ We find a neighboring pixel with larger gradient magnitude

→ It is suppressed ☹

Step 3: Perform non-maximum suppression

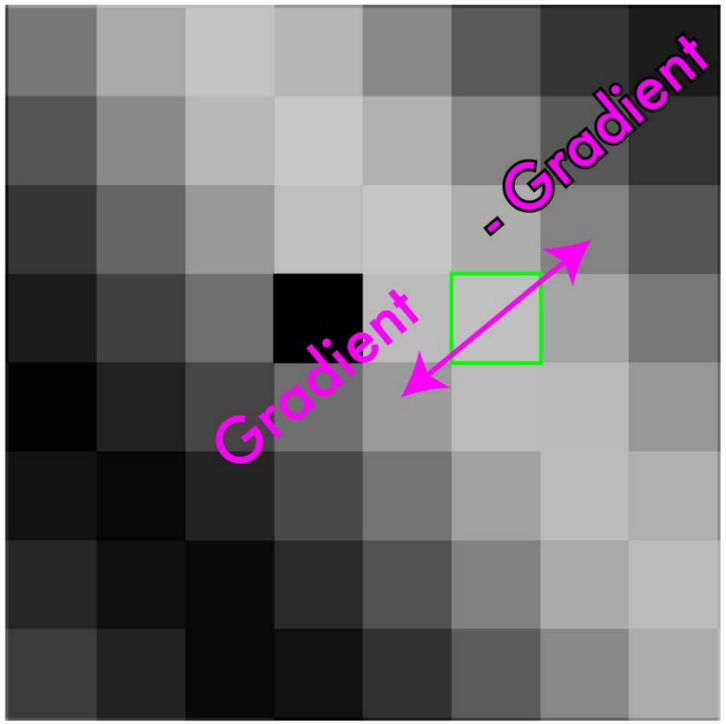


Example pixel 2:

(highlighted in white box)

compare its gradient magnitude with the magnitudes of its neighboring pixels along the gradient direction

Step 3: Perform non-maximum suppression



Example pixel 2:

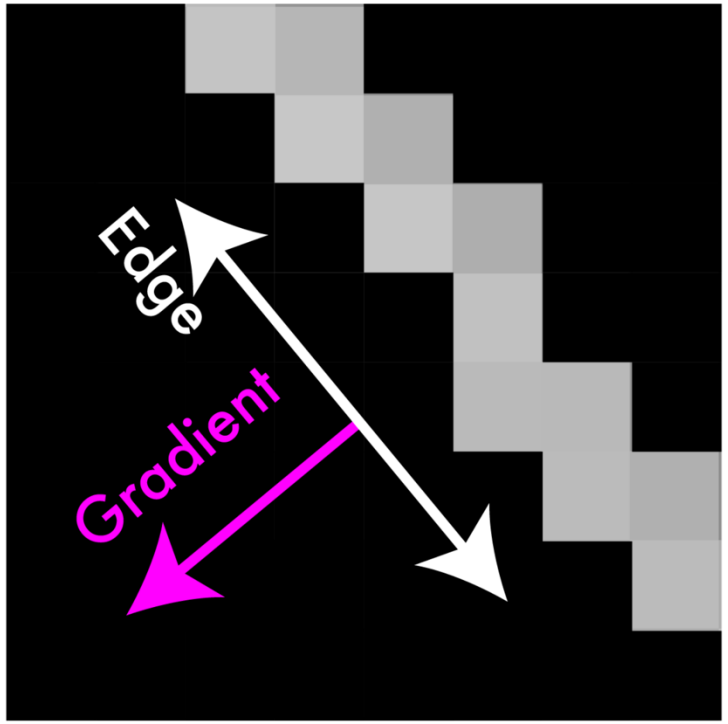
(highlighted in green box now)

compare its gradient magnitude with the magnitudes of its neighboring pixels along the gradient direction

→ Greater than that of both of its neighboring pixels

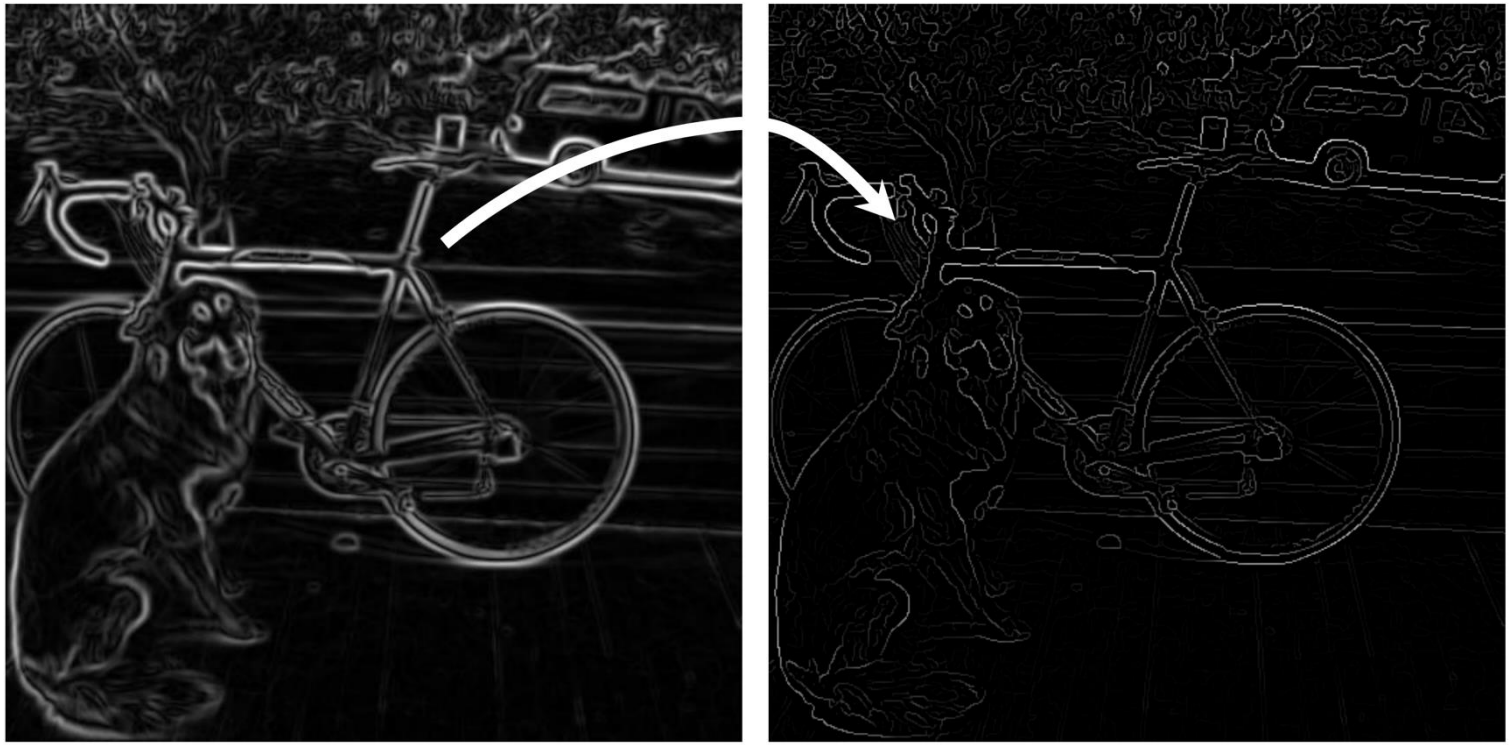
→ It is kept ☺

Step 3: Perform non-maximum suppression



This process is performed for all pixels in the image, resulting in a thinned set of edge pixels where only local maxima **along the edges** remain.

Step 3: Perform non-maximum suppression

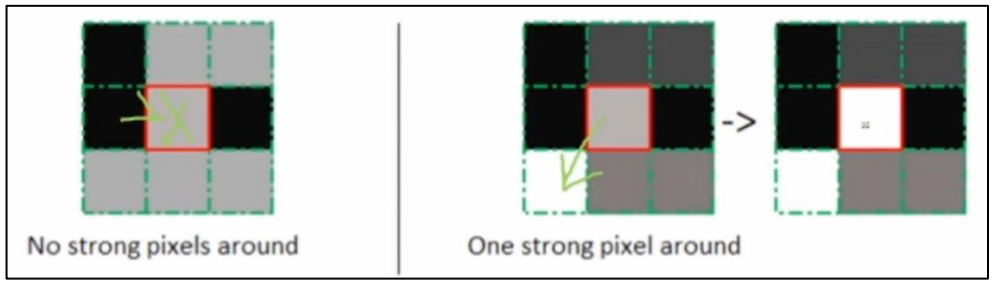


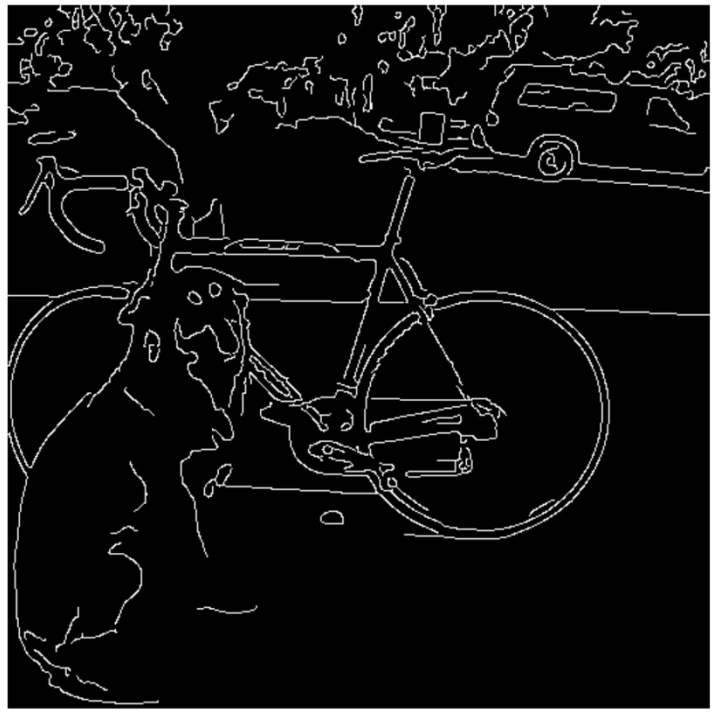
Step 4: Threshold the edges

- ***Why?*** Still some noise
- ***Idea.*** Thresholding edges
- ***How?***
  - Categorize edges into strong, weak, non-relevant
  - 2 thresholds T and t, 3 cases:
    - T < : strong edge
    - t < and < T : weak edge
    - < t : non-relevant

Step 5: Link the edges

- ***Idea.***
  - Strong edges are edges!
  - Weak edges are edges if and only if they are connected to strong edges
- ***How?***
  - Look in some neighborhood (usually 8 closet)



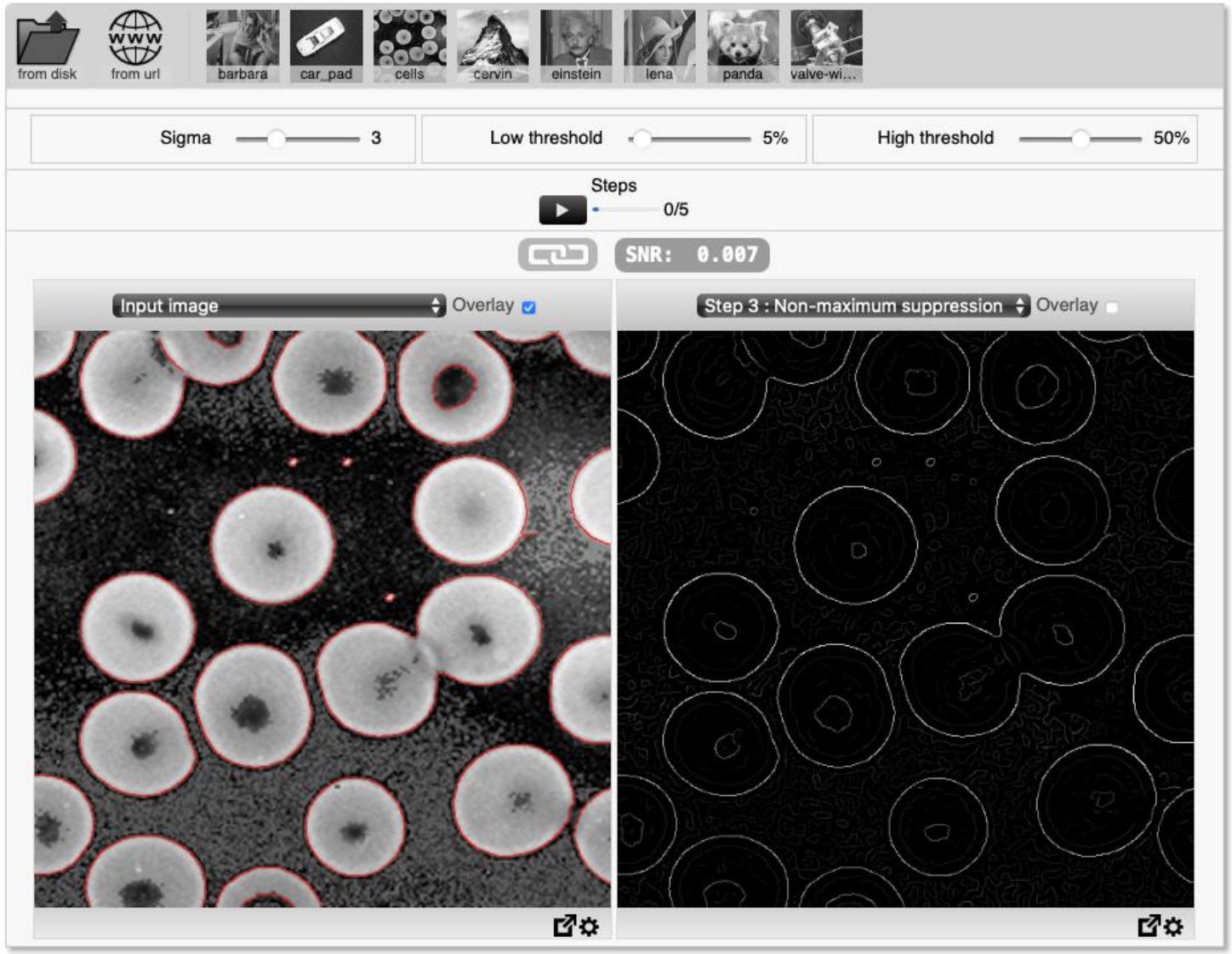


Demo

Let's try: https://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Think: how the hyperparameters affect the results?

In OpenCV: simply call "cv2.Canny" function (reference: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html)



Takeaway

- Padding?
- Binary Image: Erosion, Dilation?
- Why Features matter in computer vision?
- Different ways to detect edges
  - Prewitt & Sobel
  - DoG
  - Laplacian
  - Canny Edge Detector