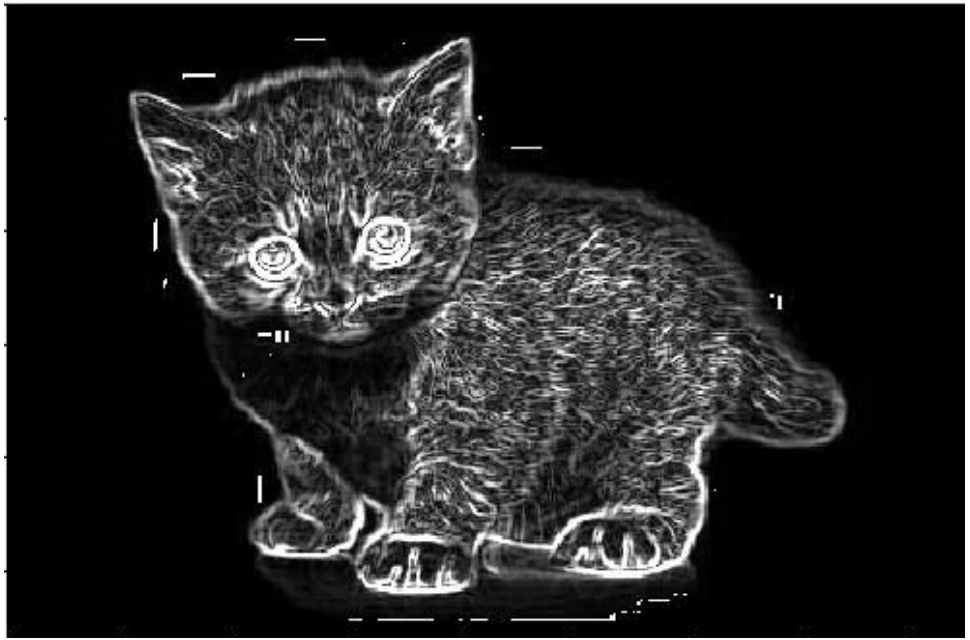
CS-GY 6643
Computer Vision



Lecture 3: Image filtering + convolutions + edge detection

Prof. Erdem Varol

Today's menu

Announcements

Image filtering continued (Szeliski 3.3)

Mid-level imaging features (Szeliski 7.2)

Coding lab

Announcements

Homework 1 was due at 10:59am

Thank you for completing!

1 point will be deducted for every hour the assignment is late (rounded down).

Make sure your assignment is submitted on brightspace asap.

We plan to release HW grades and solutions by 9/26.

CS GY 6643 - Computer Vision, Fall 2024 **Erdem Varol**

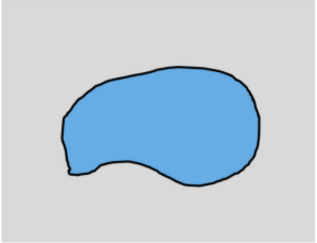
Homework 1

Out: 09/12/2024

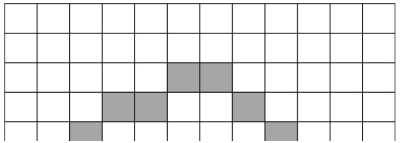
Due: 09/18/2024 11:59 PM

1. Connectivity: Connected component labeling

The famous theorem of Jordan states that every simple closed curve in the plane separates the complement into two connected nonempty sets: an interior region and an exterior. Although this looks simple and intuitive, the proof can get very complex (for those who want to look it up: Link). Elements are a closed curve and the complement, which is the plane, which is divided into 2 regions. Now let us explore how the continuous case transfers to our discrete domain.



Given the following pixel image with white areas and a gray closed pixel curve. How many connected components can you detect?



Project 1 out

Due October 4, 10:59am (~2 weeks)

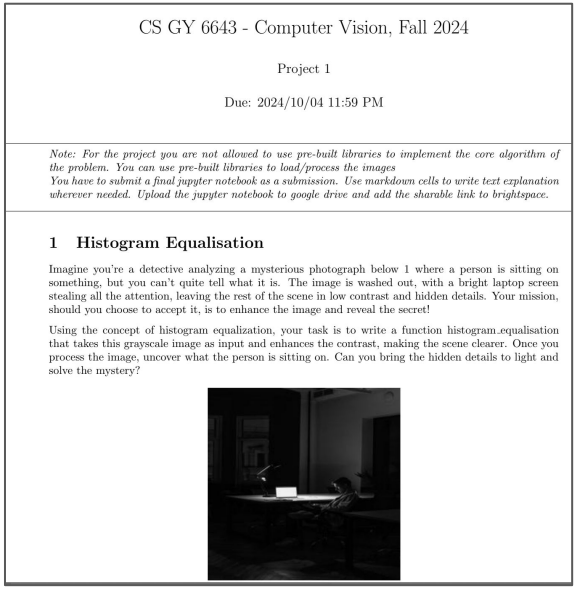
15 % of total course grade.

A python notebook with executed outputs + a pdf write up of each step explained to be submitted on brightspace.

You are encouraged to discuss with other students and look online for examples.

Project must be submitted individually (**We will check for blatant code copying**).

1 point will be deducted for every hour the project is late (rounded down).



CS GY 6643 - Computer Vision, Fall 2024

Project 1

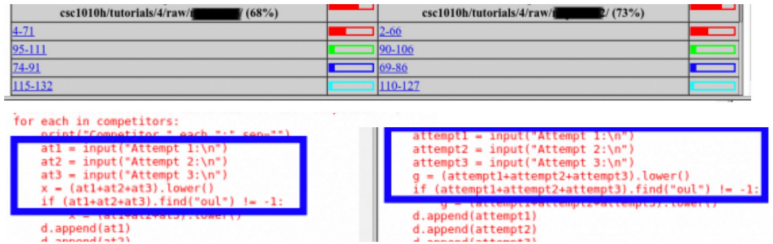
Due: 2024/10/04 11:59 PM

Note: For the project you are not allowed to use pre-built libraries to implement the core algorithm of the problem. You can use pre-built libraries to load/process the images
You have to submit a final jupyter notebook as a submission. Use markdown cells to write text explanation wherever needed. Upload the jupyter notebook to google drive and add the sharable link to brightspace.

1 Histogram Equalisation

Imagine you're a detective analyzing a mysterious photograph below 1 where a person is sitting on something, but you can't quite tell what it is. The image is washed out, with a bright laptop screen stealing all the attention, leaving the rest of the scene in low contrast and hidden details. Your mission, should you choose to accept it, is to enhance the image and reveal the secret!

Using the concept of histogram equalization, your task is to write a function histogram_equalisation that takes this grayscale image as input and enhances the contrast, making the scene clearer. Once you process the image, uncover what the person is sitting on. Can you bring the hidden details to light and solve the mystery?



csc1010h/tutorials/4/raw/[redacted] (68%)		csc1010h/tutorials/4/raw/[redacted]/ (73%)	
4-71		2-66	
95-111		90-106	
74-91		69-86	
115-132		110-127	

```
for each in competitors:
    at1 = input("Attempt 1:\n")
    at2 = input("Attempt 2:\n")
    at3 = input("Attempt 3:\n")
    x = (at1+at2+at3).lower()
    if (at1+at2+at3).find("oul") != -1:
    d.append(at1)
```

```
    attempt1 = input("Attempt 1:\n")
    attempt2 = input("Attempt 2:\n")
    attempt3 = input("Attempt 3:\n")
    g = (attempt1+attempt2+attempt3).lower()
    if (attempt1+attempt2+attempt3).find("oul") != -1:
    d.append(attempt1)
    d.append(attempt2)
```

Please complete final project team selection!

Only 37 out of 65 students have chosen teams or asked to be assigned on google spreadsheet.

Only 5 submitted on brightspace…

Due September 22, 10:59am (3 days!)

Worth 2% of total course grade

Each student must submit the names and NYU ID's of teammates and project area on brightspace.

Use the match-making spreadsheet to declare project teams + ask to join other teams.

https://docs.google.com/spreadsheets/d/1RkjD30T0BsH8VO79zaxYJUE6GJFWTgVNu7nE4SJ1UYw/edit?usp=sharing

A	B	C	D	E	F	G	H	I	J
Project area	**Project tentative title**	**Team member 1 name**	**Team member 1 email**	**Team member 2 name**	**Team member 2 email**	**Team member 3 name**	**Team member 3 email**	**Team member 4 name**	**Team member 4 email**
Object tracking	Cell tracking in C. elegans microsc	Maren Eberle	mie8014@nyu.edu	Nalini Ramanathan	nar8991@nyu.edu	Ryan Fleishman	rmf9265@nyu.edu	Neel Gandhi	njg9191@nyu.edu
Biometrics	TBA	Allison Lee	al6897@nyu.edu	ASSIGN ME TEAMMATES					
Object tracking	Ball Tracking in Sports	Pratham Shah	prathamshah@nyu.edu	Ohm Patel		Varad Naik	vvn7114@nyu.edu		
Image Reconstruction	TBA	Jaydeep Mulani	jgm9413@nyu.edu	Aditya Azad	aa10878@nyu.edu	Sidhved Warik	sw6071@nyu.edu	Vedangi Kirtane	vrk3359@nyu.edu
Image Alignment	TBA	Tianxiao He	th3129@nyu.edu	Yurong Liu	yl6624@nyu.edu	Jialin Li	jl10897@nyu.edu	Remo Shen	ys6345@nyu.edu
Superresolution	TBA	Shambhavi Seth	ss17936@nyu.edu	Aryan Prasad	ap7949@nyu.edu	Roman Vakhrushev	rv1057@nyu.edu		
Image Alignment	TBA	Jizheng Dong	jd4587@nyu.edu	Yifei Zhuang	yz9865@nyu.edu	Bowen Gong	bg1941@nyu.edu	Lucy Shi	ls4900@nyu.edu
Image Colorization	TBA	Nika Emami	ne2213@nyu.edu	ASSIGN ME TEAMMATES					
		Wei-Lun Hung	wh2528@nyu.edu	Yu-Yuan Chang	yc6549@nyu.edu	Guan-Cherng Lin	gl2547@nyu.edu	Da Hye Kim	dk3343@nyu.edu
		Gordon Lu	gl1589@nyu.edu	Suemy Inagaki	si2324@nyu.edu	Felipe Oliveira	fd2264@nyu.edu	Chhatrapathi Sivaji Lakk	cl7203@nyu.edu
Image Generation	TBA	Daoyu Li	dl5312@nyu.edu	Ziming Song	zs2815@nyu.edu	ASSIGN ME TEAMMATES			
		Aravindsrinivas Krishnar	ak11115@nyu.edu	Priyangshu Pal	pp2833@nyu.edu	Charan K Raja	ck3470@nyu.edu	Bharat S	

Project ideas

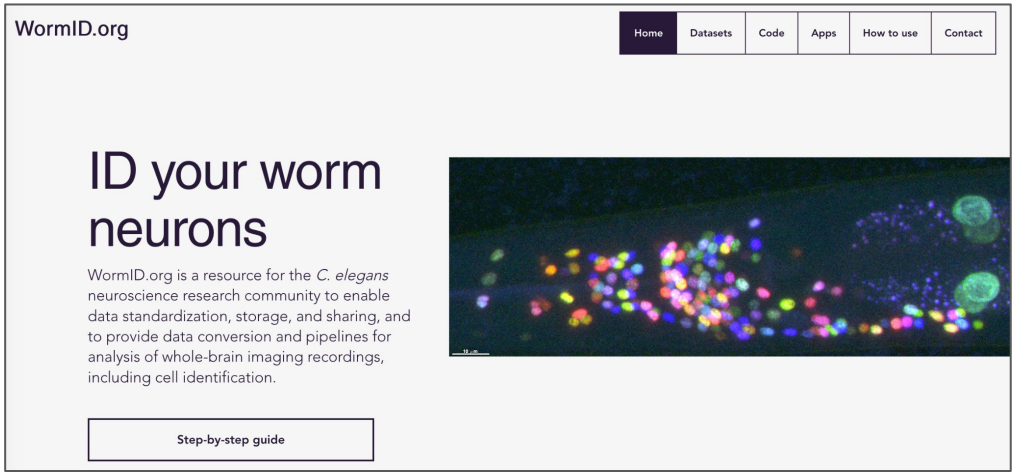
Check slack channel #project-ideas for more suggestions.

What is COCO?

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features:

- Object segmentation
- Recognition in context
- Superpixel stuff segmentation
- 330K images (>200K labeled)
- 1.5 million object instances
- 80 object categories
- 91 stuff categories
- 5 captions per image
- 250,000 people with keypoints

https://cocodataset.org



https://www.wormid.org/

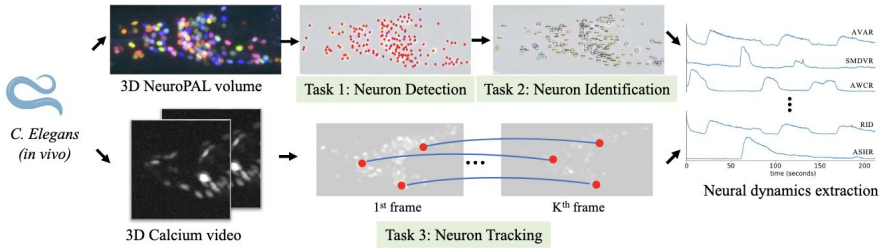


Figure 1: Task overview. To extract neural dynamics from microscopy images of *in vivo C. elegans*, **WormND** benchmarks each step of the pipeline: neuron *detection*, *identification*, and *tracking*.

Take advantage of office hours to discuss the next project + final project ideas

Instructor: Prof. Erdem Varol

Email: ev2240@nyu.edu

Office Hours: On Zoom,15 min appointments.

TA: Rishabh Raj

Email: rr4574@nyu.edu

Office Hours: Tuesdays 1-2pm on Zoom.

TA: Malhar Patel

Email: mkp6112@nyu.edu

Office Hours: Mondays 1-2pm on Zoom.

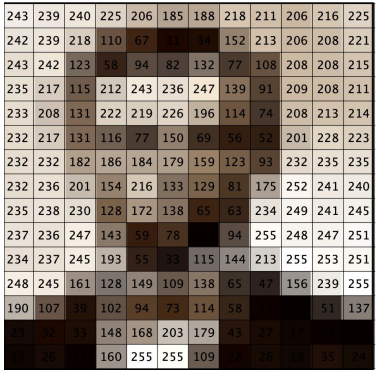
Also in person in 370 Jay Street, room 1166

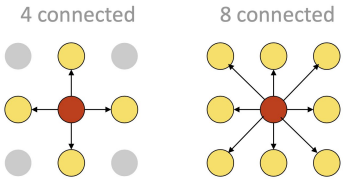
Recap of previous lecture

Recap of previous lecture

We learned

- How images are represented digitally and topologically
- How we can manipulate images through pixel and patchwise operations, and filters.



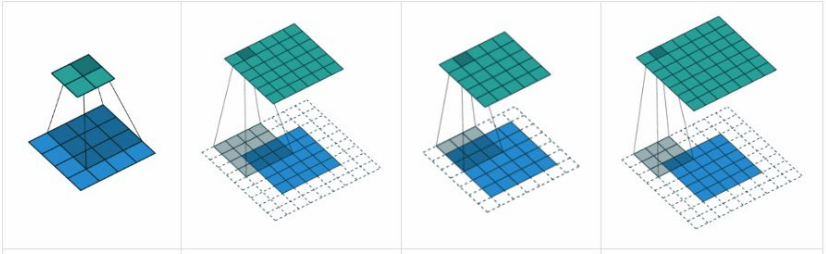


Today we will learn

More filtering, convolutions

Non-linear filtering

Going from pixel-wise to feature-wise understanding of images

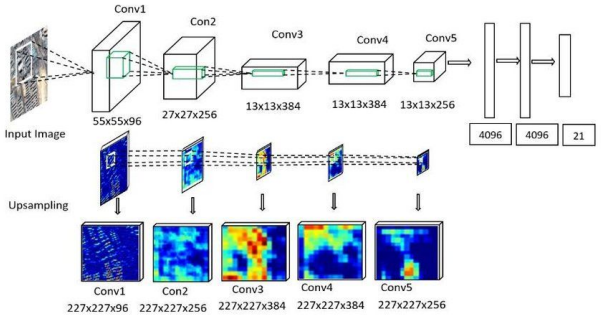


pixels

patches

243	239	240
242	239	218
243	242	123

features



Low level → High level

Recall the box filter

$I[x, y]$: Image intensity at pixel (x,y)

243	222	171
203	145	137
209	132	193

121	111	85
101	183	68
104	66	96

$$f : pa \rightarrow p'$$

Input **patch** Output pixel

$I[0,0]$	$I[0,1]$	$I[0,2]$
$I[1,0]$	$I[1,1]$	$I[1,2]$
$I[2,0]$	$I[2,1]$	$I[2,2]$

Input

Filter g

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

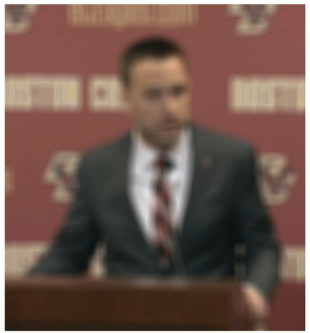
(Dot product in 1D)

	$J[1,1]$	

Output

$$J[1,1] = \frac{1}{N} \sum_{x,y} I[x, y]$$

Input Image



Output: Filtered image
(mimic focus blur)

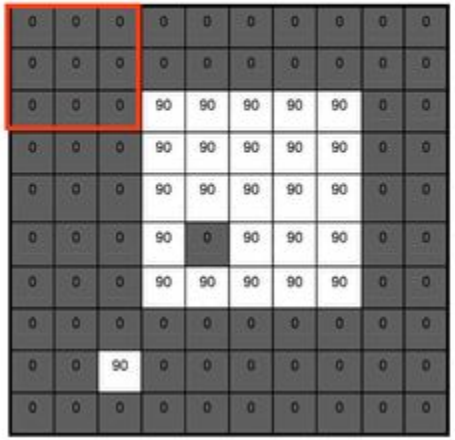
This filter computes dot products then slides

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

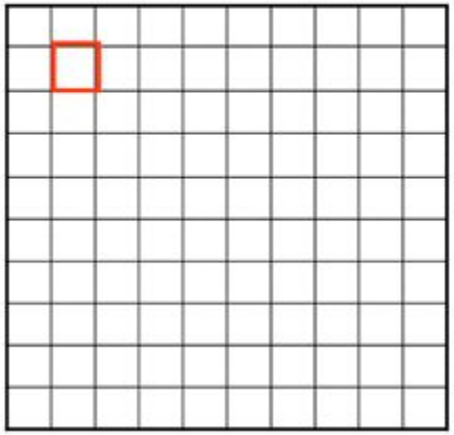
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$F[x,y]$



$G[x,y]$



Sliding + computing dot products is known as cross-correlation

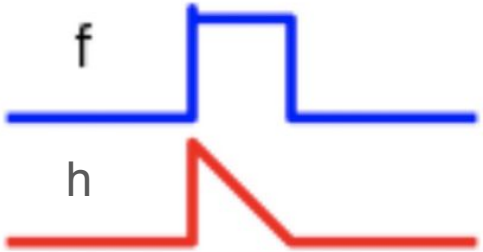
- 1D $$g(i) = \sum_k f(i+k)h(k)$$

- 2D $$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

$$\boxed{g = f \otimes h}$$

$\otimes$: correlation operator

Why is cross-correlation not generally used as a filter? Non-commutative



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

Why do we care? We want to be able to potentially apply many filters without caring about their order. Flexibility.

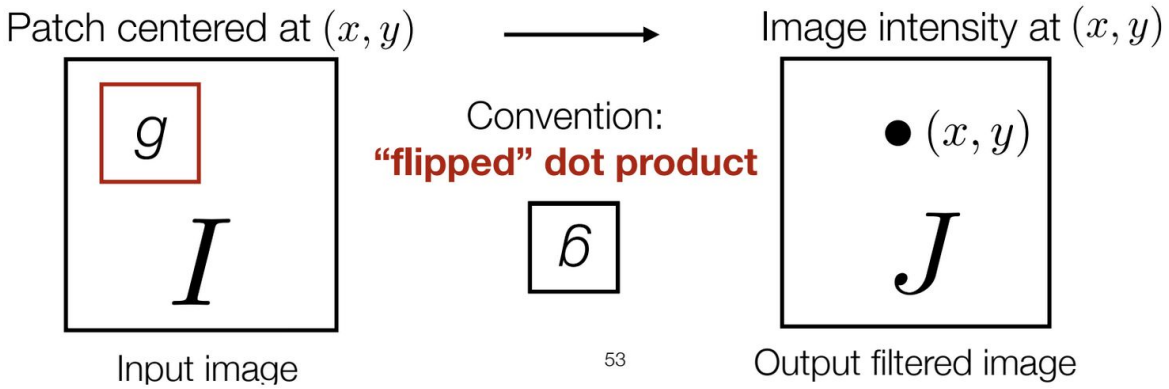
Solution: Convolutions!

Convolution

Let I be the input image and g be the filter (conv. **kernel**).

Indexes go backward!

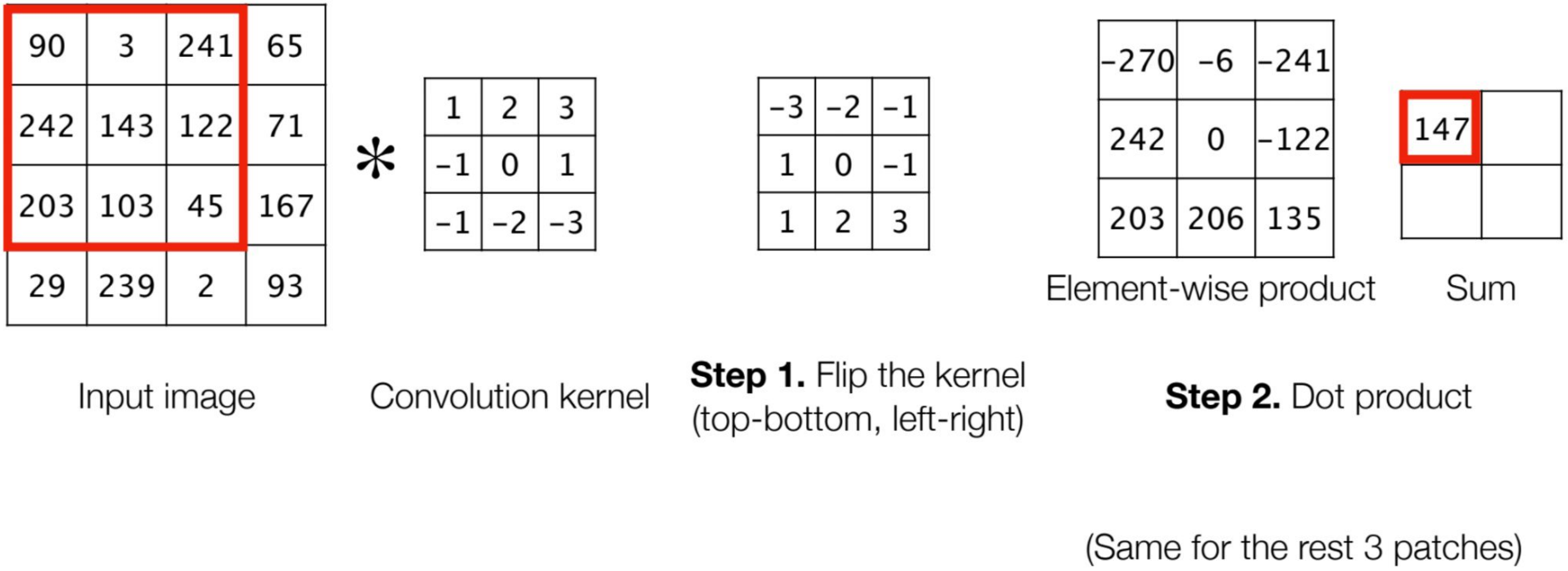
$$J[x,y] = I * g = \sum_{k,l} I[x-k, y-l]g[k,l]$$



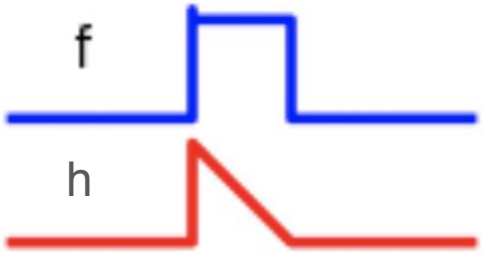
53

Source: F. Durand

Convolution example



Convolution is commutative



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

Other convolution properties

Commutative:

$$h[n] \circ g[n] = g[n] \circ h[n]$$

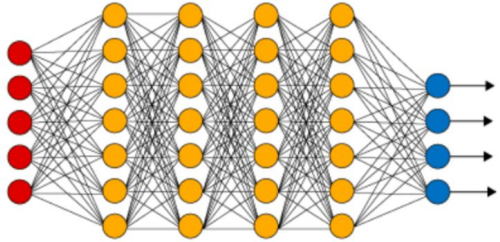
Associative:

$$h[n] \circ g[n] \circ f[n] = h[n] \circ (g[n] \circ f[n]) = (h[n] \circ g[n]) \circ f[n]$$

Distributive with respect to the sum:

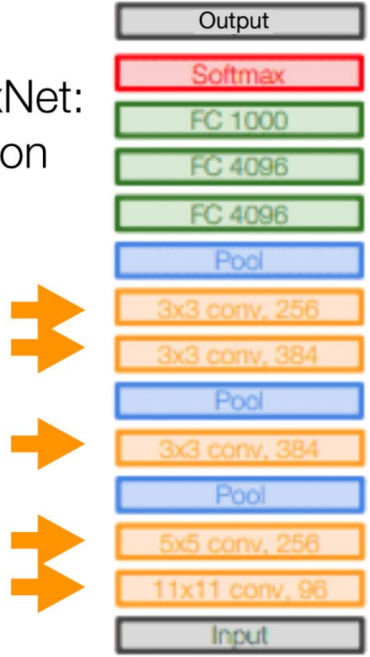
$$h[n] \circ (f[n] + g[n]) = h[n] \circ f[n] + h[n] \circ g[n]$$

Flexibility of convolutions makes them ideal filtering operators



What you see in the media

The revolutionary AlexNet:
5 layers of convolution



18

Q: Is box filter a convolution?

$I[x, y]$: Image intensity at pixel (x,y)

$I[0,0]$	$I[0,1]$	$I[0,2]$
$I[1,0]$	$I[1,1]$	$I[1,2]$
$I[2,0]$	$I[2,1]$	$I[2,2]$

Input

Filter g

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

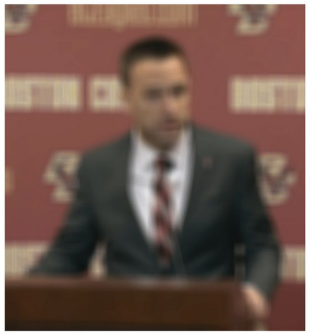
(Dot product in 1D)

	$J[1,1]$	

Output

$$J[1,1] = \frac{1}{N} \sum_{x,y} I[x,y]$$

Input Image



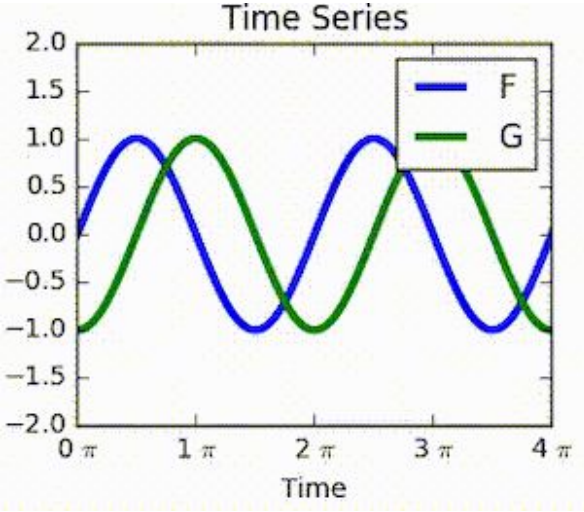
Output: Filtered image
(mimic focus blur)

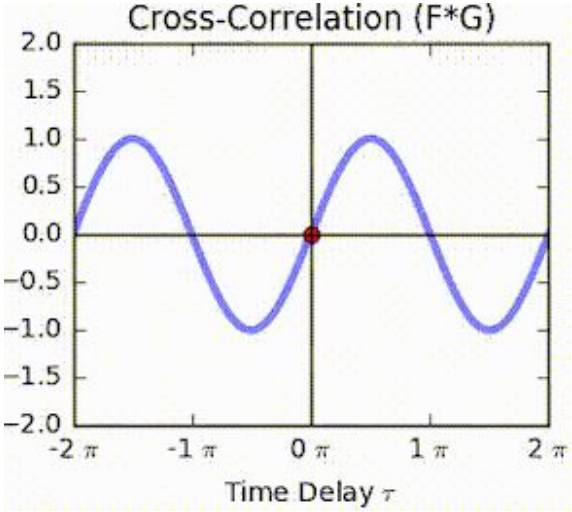
Yes - it's both a convolution and a cross-correlation.

- Convolution same as cross correlation with kernel transposed around each axis.
- The two operations (correlation and convolution) are **the same** if the kernel is symmetric about the axes.

$$g = f \otimes h = f * h^{\wedge} \qquad h^{\wedge}: \ reflection \ of \ h$$

- General consensus:
  - Cross-correlation: measure the similarity between two signals
  - Convolution: filtering operation on a signal

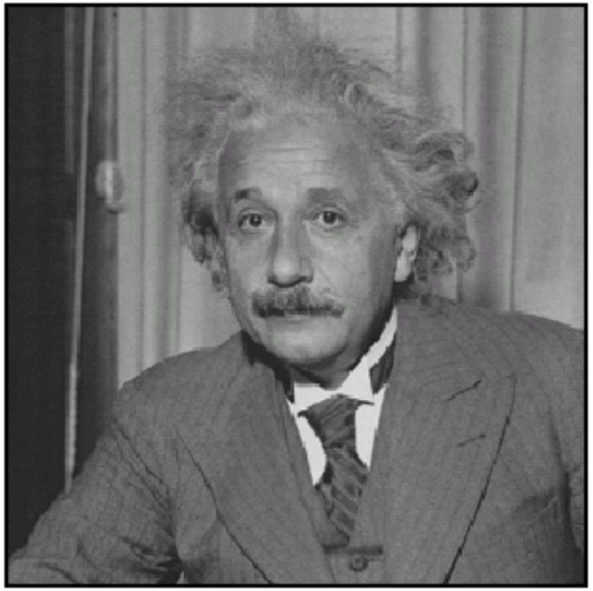




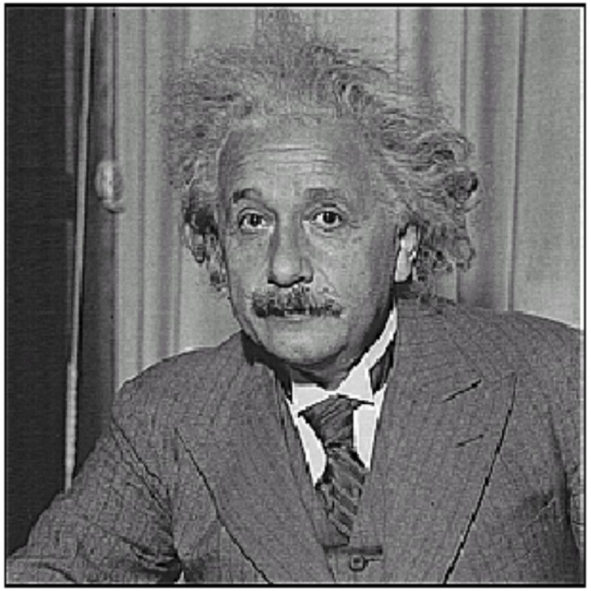
Filters are stackable

How can we design a sharpening filter?

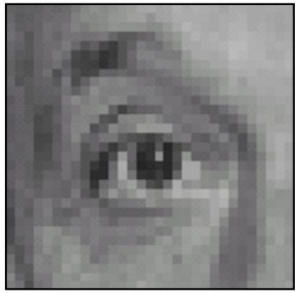
Input



Sharpened result



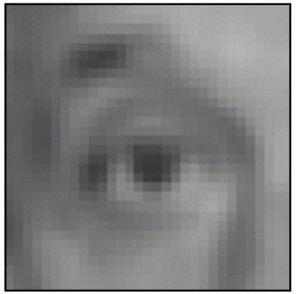
Box filter: blurring effect



$*$ $\frac{1}{9}$

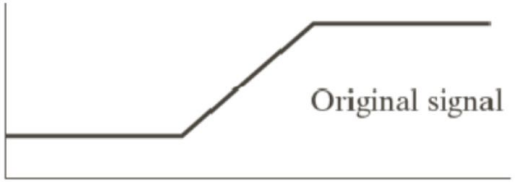
1	1	1
1	1	1
1	1	1

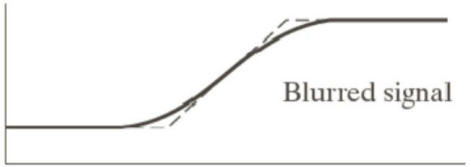
$=$



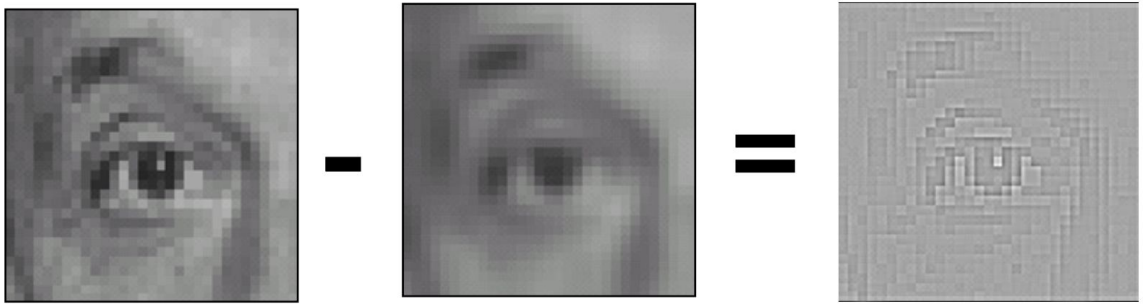
Original

Blur (with a box filter)



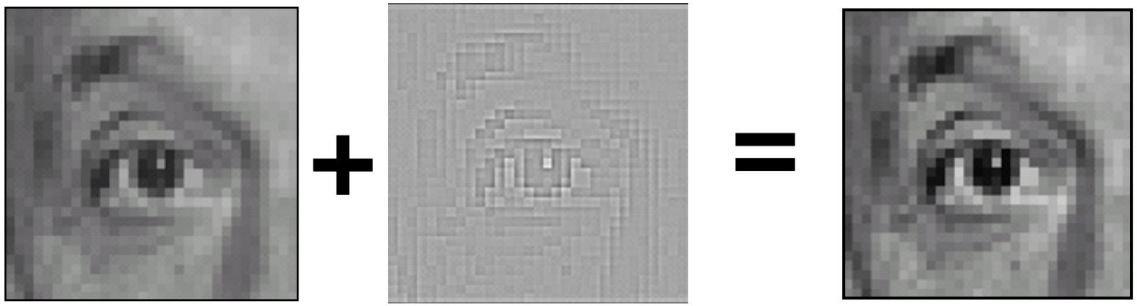


Sharpen filter: get details

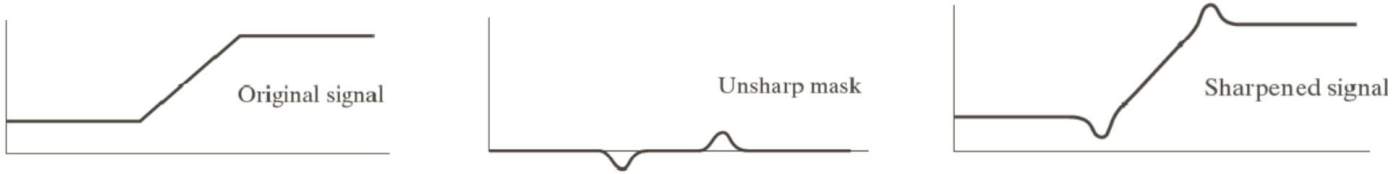


Original | Blur (with a box filter) | Details

Sharpen filter: add details

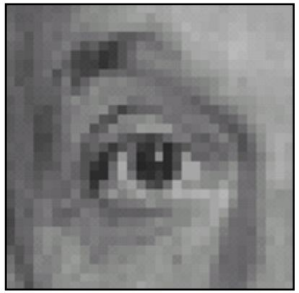


Original + Details = **Sharpened**



Impulse filter: Identity

Box size = [1,1]

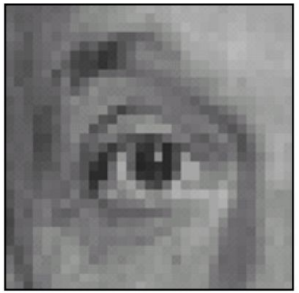


Original

$*$

0	0	0
0	1	0
0	0	0

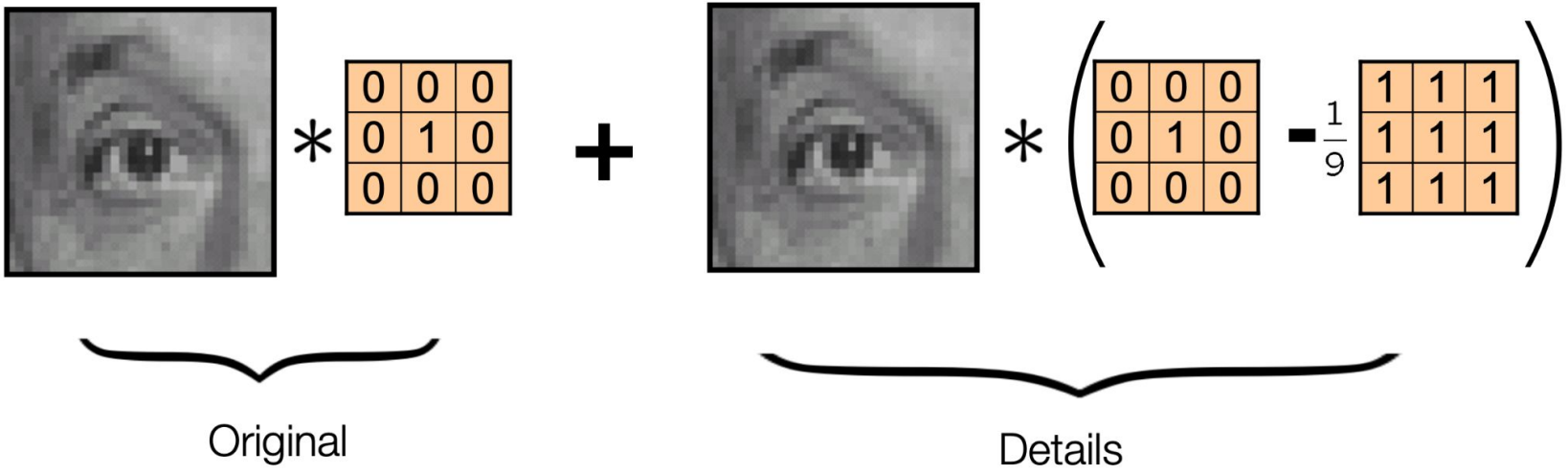
$=$



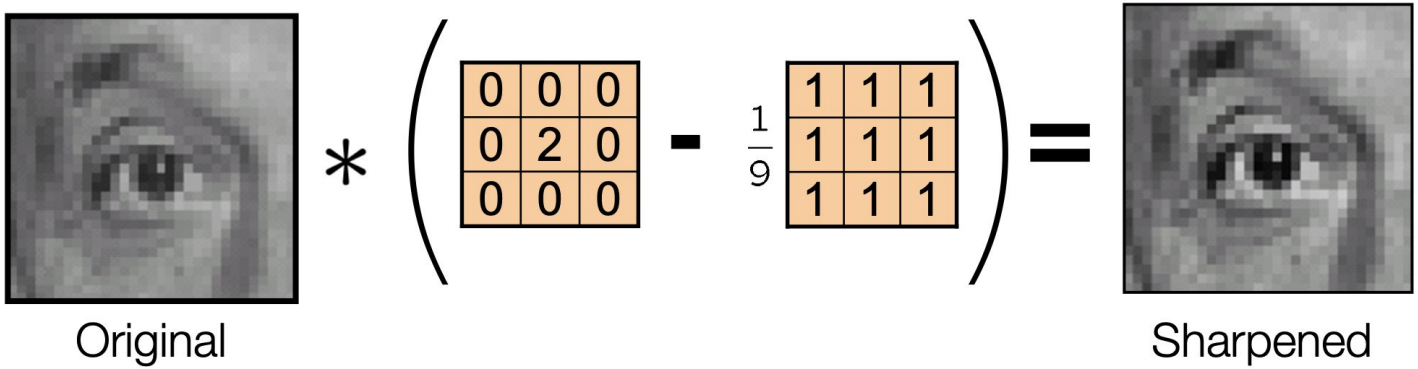
Filtered
(no change)

Sharpen filter: put them together

Remember: convolution filters are linear!



Sharpen filter: one filter

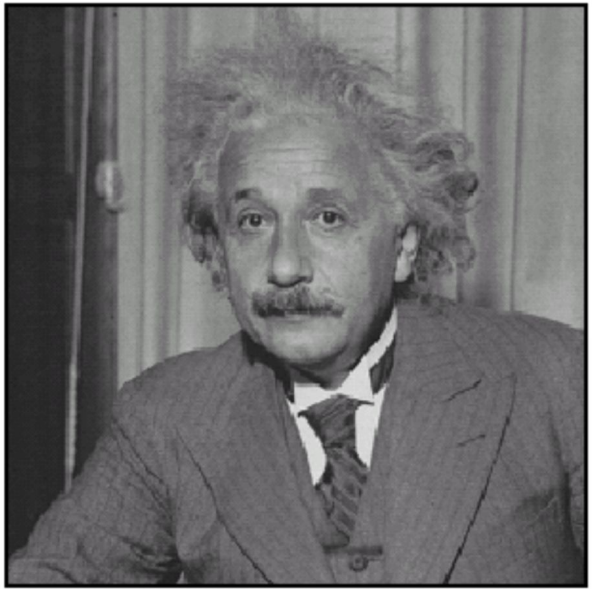


$$\text{Original} * \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) = \text{Sharpened}$$

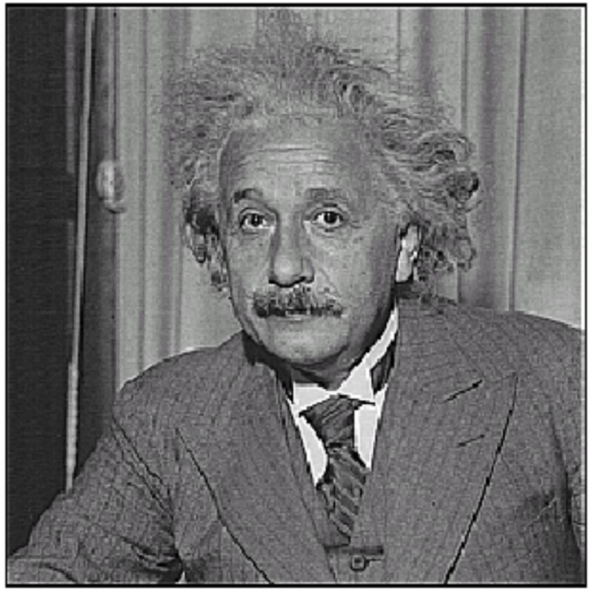
$$\frac{1}{9} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 17 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Result

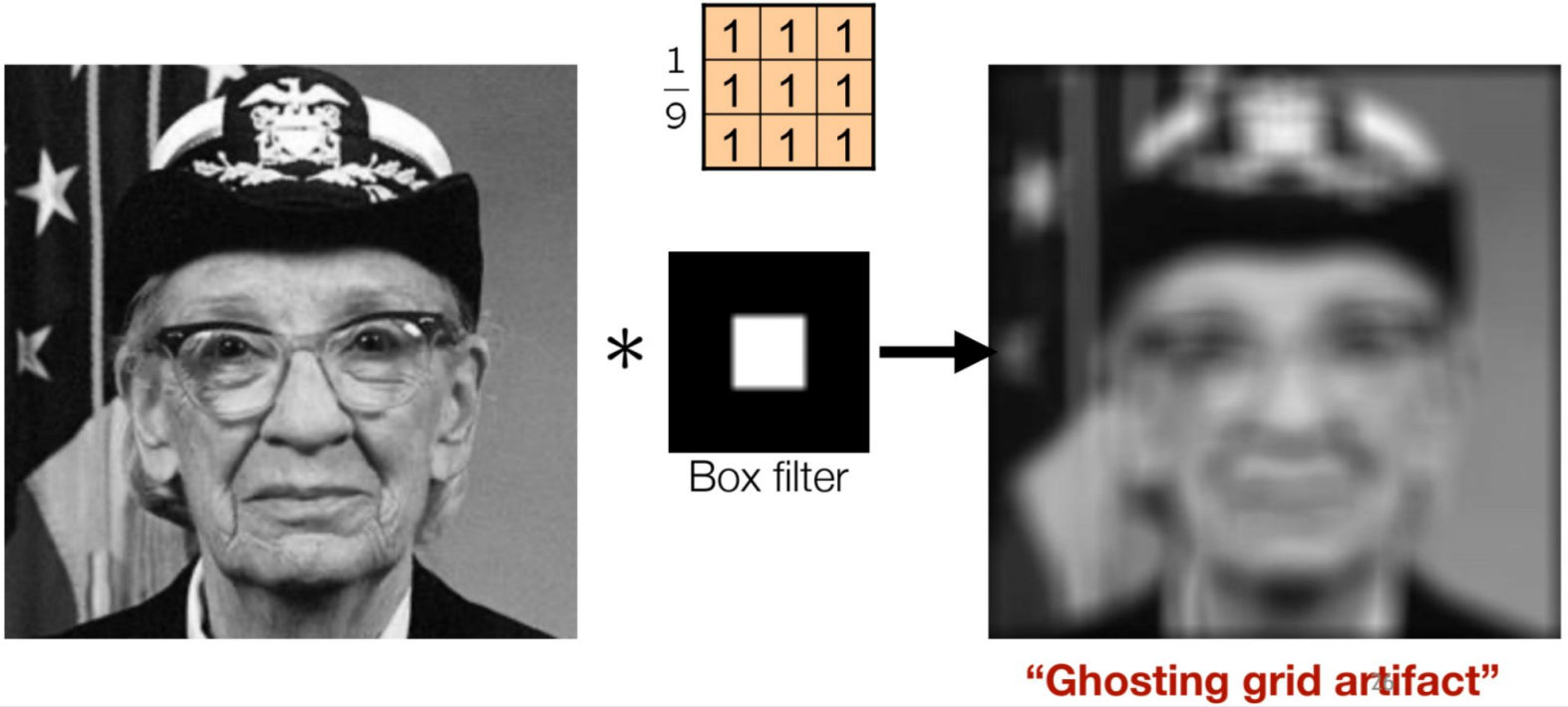
Input



Sharpened result

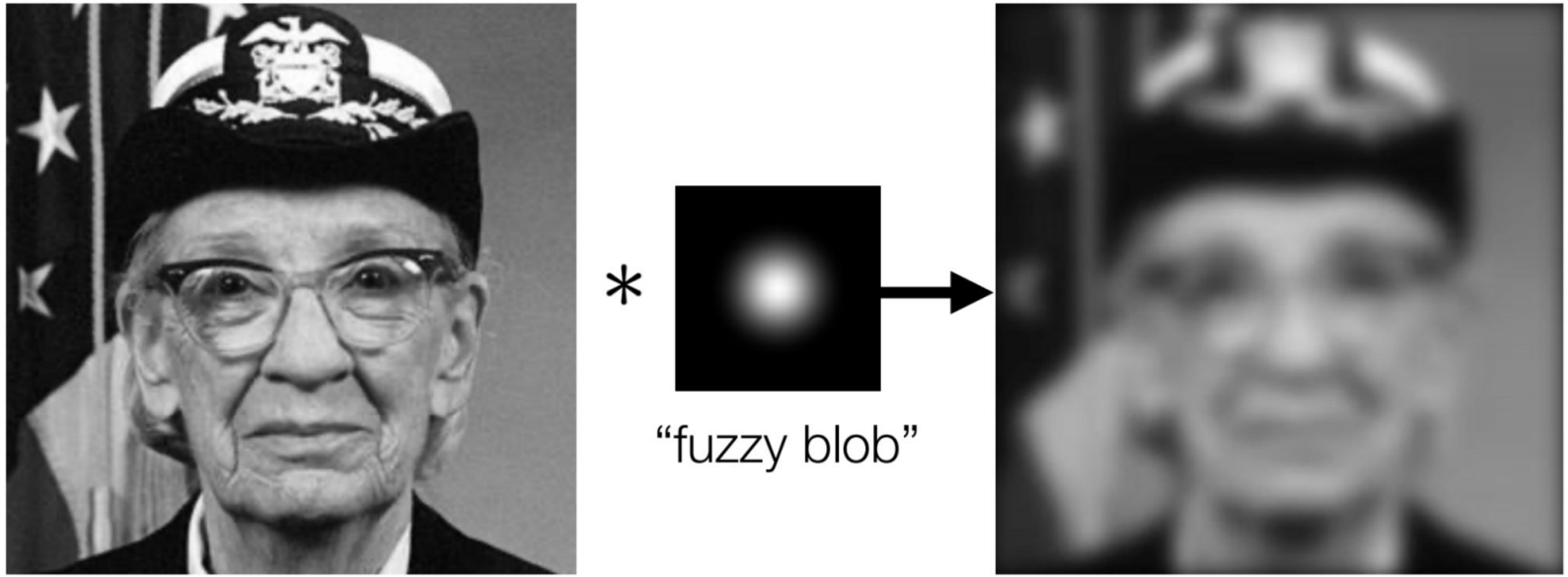


Box filters are … blocky.



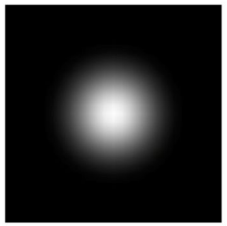
Can we make this smoother?

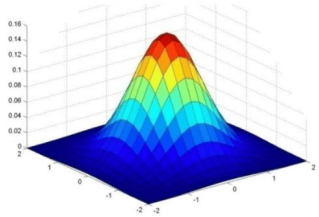
Idea: weight of neighborhood pixels according to their closeness to the center



Gaussian filter

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



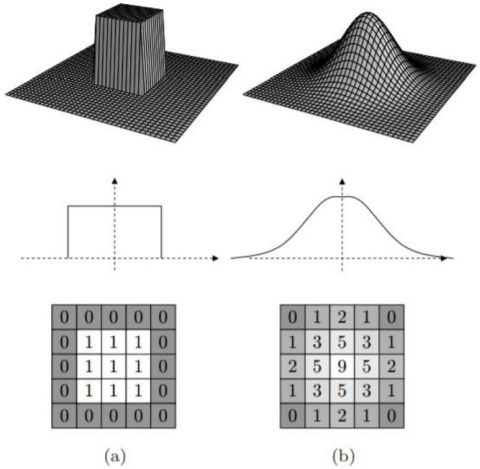


Convolution kernel ***g***

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, σ = 1)

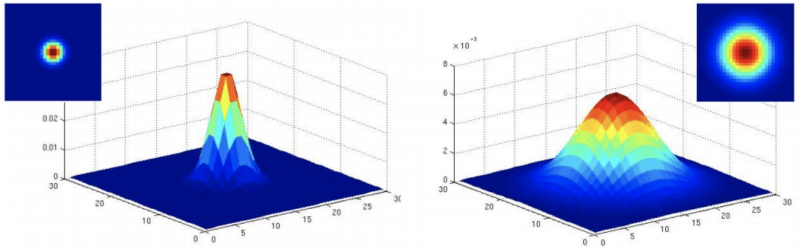
Box vs. gaussian



- Divide a constant factor to make the kernel sum to 1

Property 1: Bigger sigma means more blurring (and also higher computational cost. Why?)

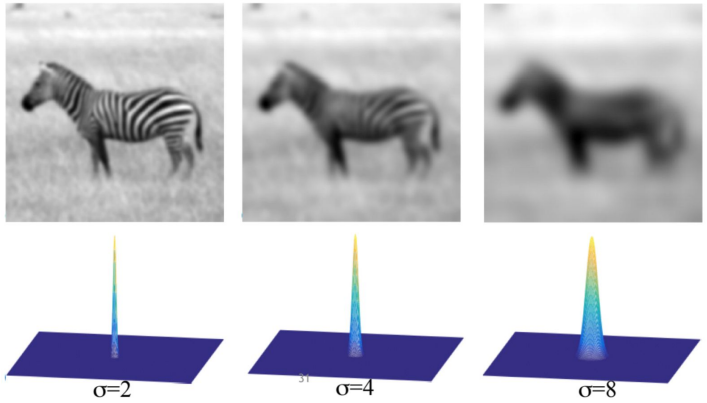
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



kernel: σ = 2 with 30 x 30

kernel: σ = 5 with 30 x 30

- Standard deviation σ: determines extent of smoothing



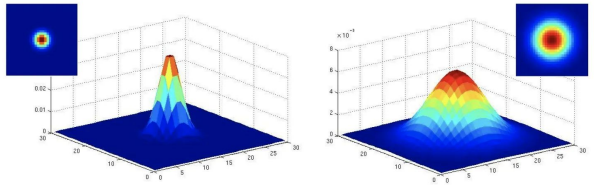
Bigger σ
- More blurry
- Bigger effective kernel size

Complexity detour: Each filter can be written as a linear matrix operation

$$\begin{array}{|c|c|c|c|c|} \hline 72 & 88 & 62 & 52 & 37 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1/4 & 1/2 & 1/4 \\ \hline \end{array} \Leftrightarrow \frac{1}{4}\begin{bmatrix} 2 & 1 & . & . & . \\ 1 & 2 & 1 & . & . \\ . & 1 & 2 & 1 & . \\ . & . & 1 & 2 & 1 \\ . & . & . & 1 & 2 \end{bmatrix} \begin{bmatrix} 72 \\ 88 \\ 62 \\ 52 \\ 37 \end{bmatrix}$$

Figure 3.12 *One-dimensional signal convolution as a sparse matrix-vector multiplication,* $\mathbf{g} = \mathbf{H}\mathbf{f}$.

- Convert 1D or 2D images into raster-ordered vectors g and f
- H is sparse matrix of convolution (respectively correlation)



$\sigma = 2$ with 30×30 kernel

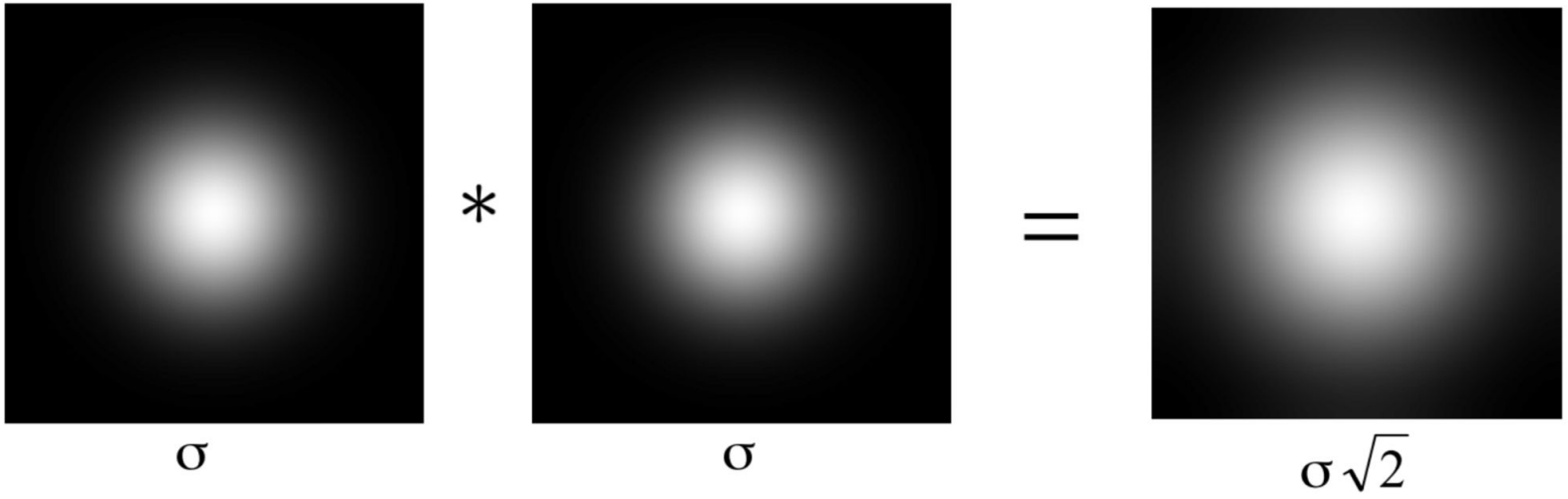
$\sigma = 5$ with 30×30 kernel

More zeros in filter, less multiplication operations.

~O(n^2) operations instead of O(n^3) for small kernels.

Property 2: Recursive

Convolve a Gaussian filter with itself? Get another Gaussian.



Property 3: separability

Thanks to the commutativity of convolution!

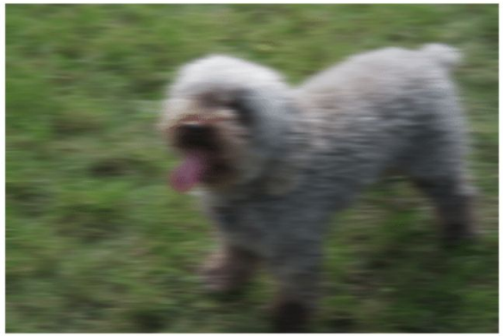
It's a **separable kernel**.

- Blur with 1D Gaussian in one direction, then the other.
- Fast! For an $n \times n$ kernel, $\mathcal{O}(n)$ instead of $\mathcal{O}(n^2)$

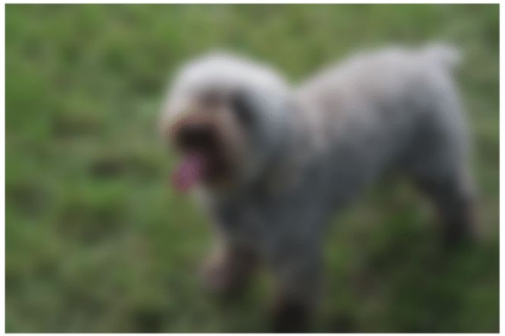
$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Separable kernel

I



$\text{blur}_x(I)$



$\text{blur}_y(\text{blur}_x(I))$

Complexity detour: separability improves efficiency

- Compute time
  - MxM mask
  - NxN image

$O(M^2N^2)$ "for" loops are nested 4 deep

- Special case: *separable*

Two 1D kernels

$$w = w_x * w_y$$

Due to associativity

$$w * f = \underbrace{(w_x * w_y)}_{O(M^2N^2)} * f = \underbrace{w_x * (w_y * f)}_{O(MN^2)}$$

Examples of separable filters

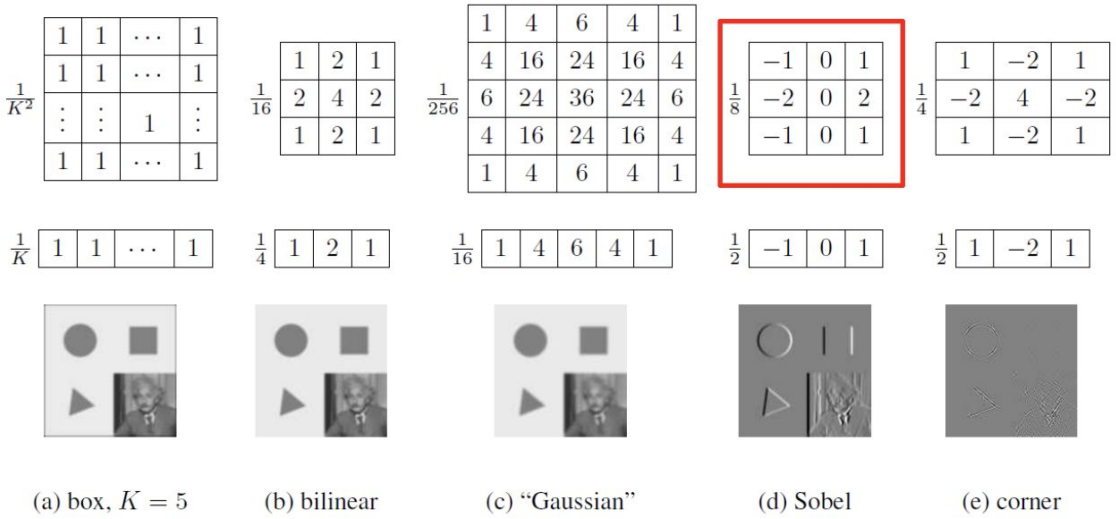
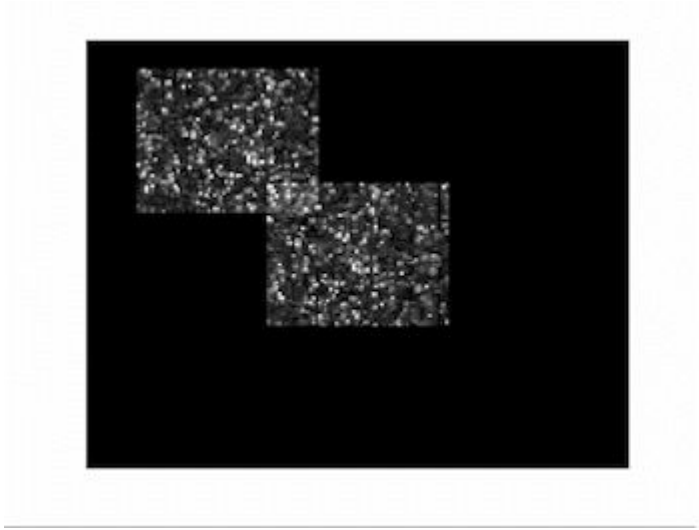
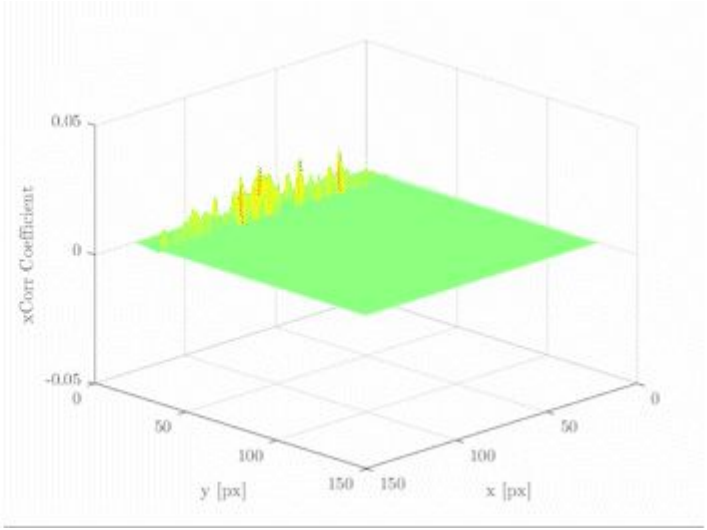


Figure 3.14 *Separable linear filters: For each image (a)–(e), we show the 2D filter kernel (top), the corresponding horizontal 1D kernel (middle), and the filtered image (bottom). The filtered Sobel and corner images are signed, scaled up by 2× and 4×, respectively, and added to a gray offset before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

So far, we have treated filters as things separate from images that are to be filtered.

Images can be their own self filters:





One key application of filtering - pattern matching

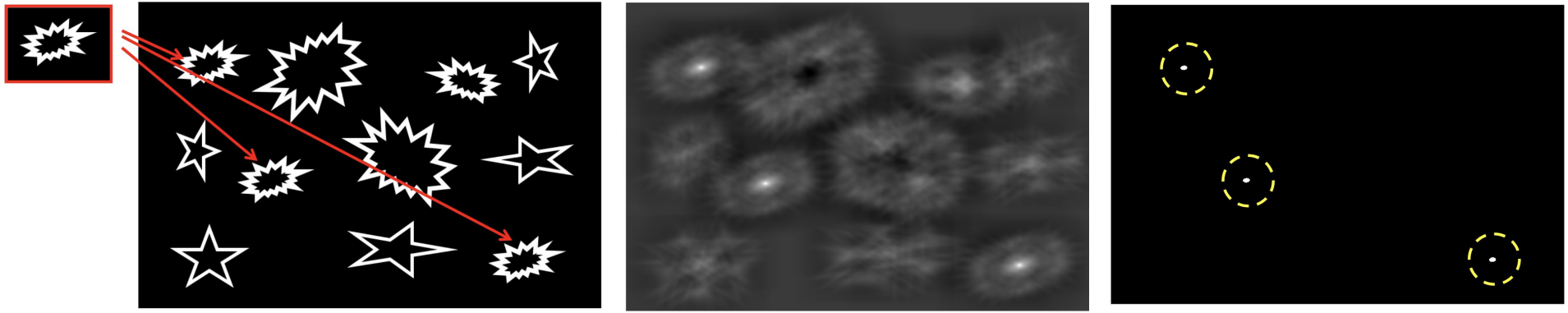
- The optimal (highest) response from a filter is the autocorrelation evaluated at position zero

$$\max_{\bar{x}} C_{ff}(\bar{x}) = C_{ff}(0) = \int f(\bar{s}) f(\bar{s}) d\bar{s}$$

- A filter responds best when it matches a pattern that looks itself
- Strategy
  - Detect objects in images by correlation with "matched" filter

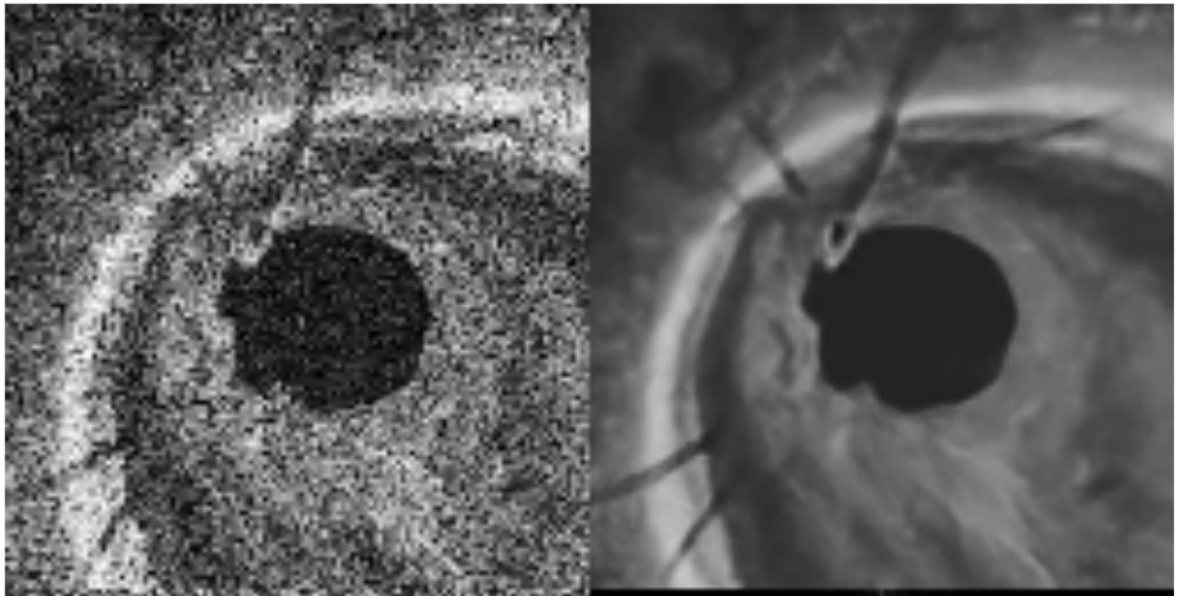
Convolving template against larger image - pattern matching

f h f * h (f * h) > threshold



*project 1 problem 3

Convolving linear filters ultimately have its limits



Example OCT
(optical coherence
Tomography for studying
Eye diseases)

Result from Shijie Li, new
deep learning-based
methodology

Noise patterns might be non-linear, anisotropic

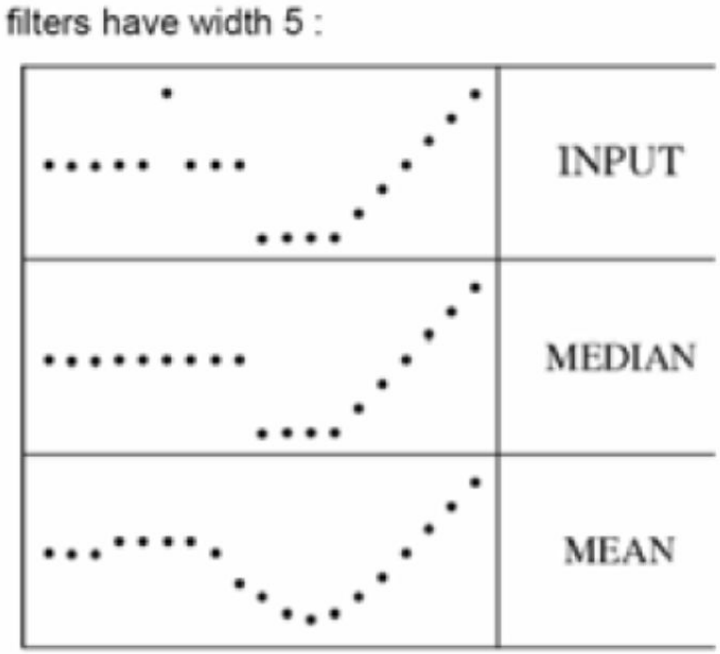
High amount of outliers = median filter

- Also: “shot noise”, “salt&pepper”
- Replace certain % of pixels with samples from pdf
- Best filtering strategy: filter to avoid outliers

gaussian

median

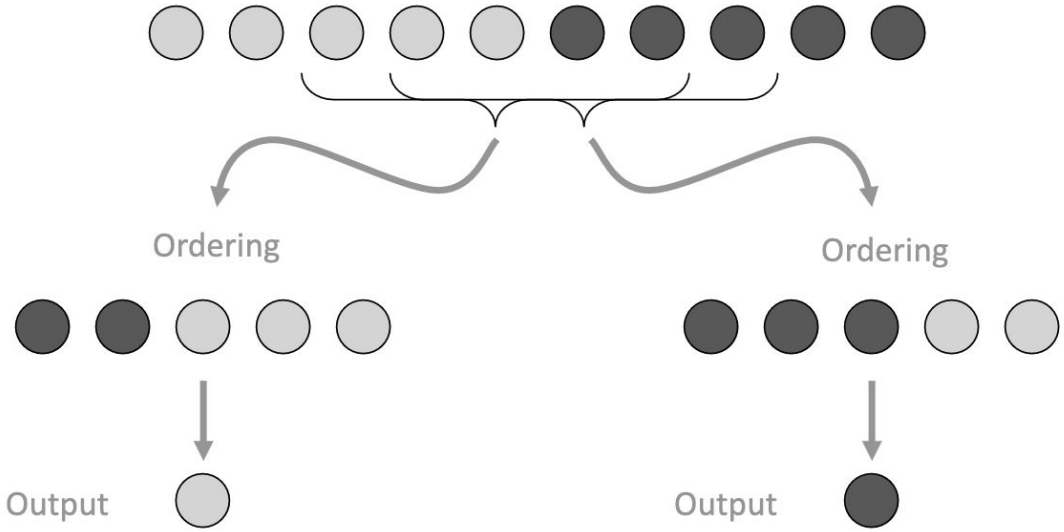
- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers



Insight: Can we use our assumptions about images to help filter them?

Source: K. Grauman

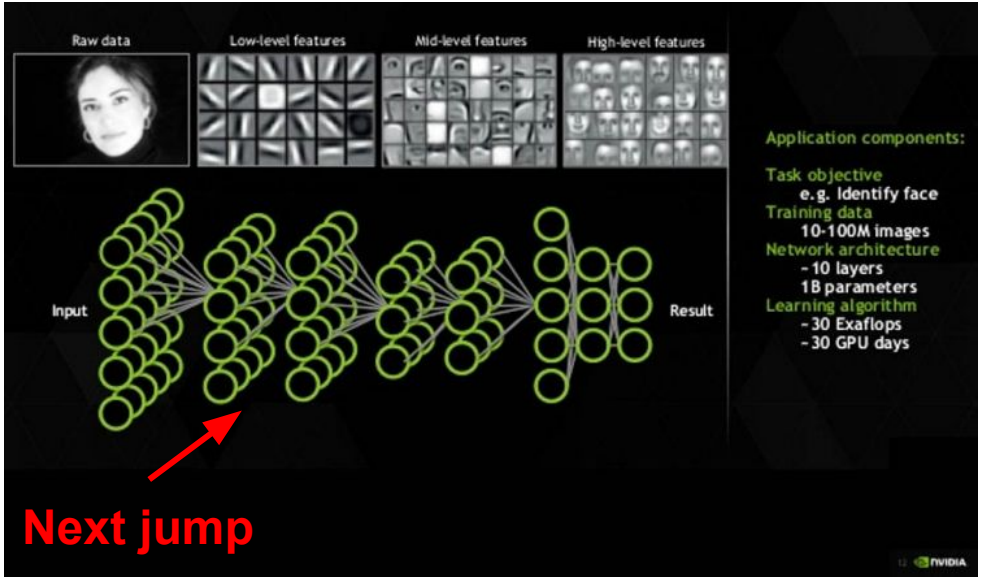
Inductive bias in median filtering: images are piecewise “flat”



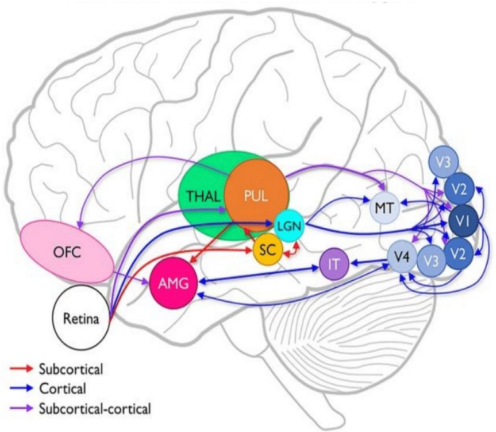
Piecewise Flat Image Models

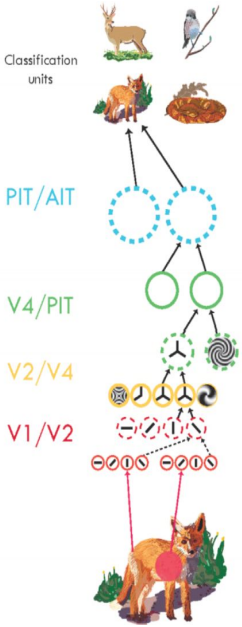
- Image piecewise flat -> average only within similar regions
- Problem: don't know region boundaries

Linear filters are here



We need to make a new jump in our evolution to the next layer of the visual cortex to understand region boundaries





High-level vision (object, scene) — After midterm

Mid-level vision (segmentation) — Up to midterm

Low-level vision (Edge, corner)

features

Lecture 3

243	239	240
242	239	218
243	242	123

patches

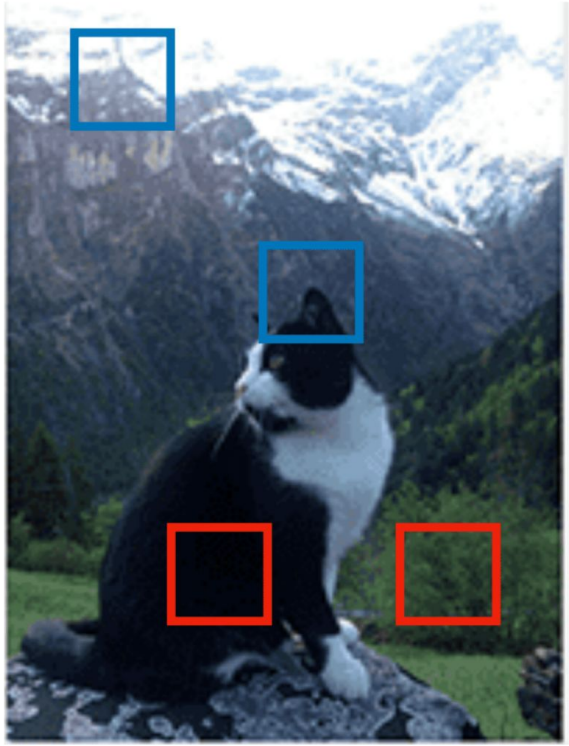
Lecture 2

243

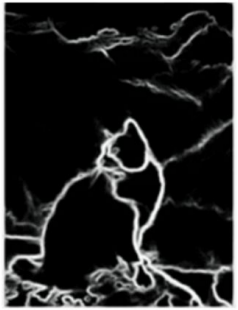
pixels

start

Intermediate computation in the brain



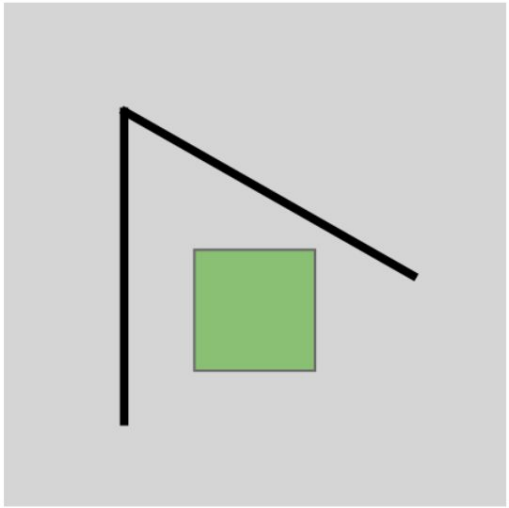
Edge



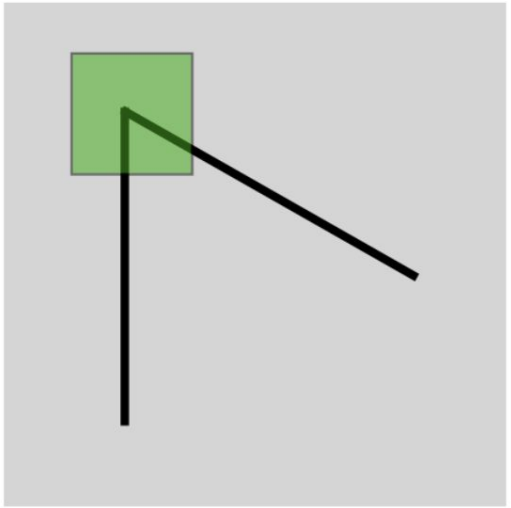
Corner

Texture
(fur, leaves)

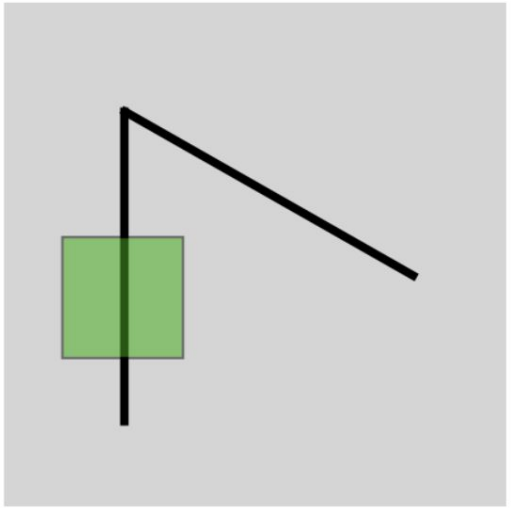
Three types of regions



Textured region
(appearance)



Corner
(anchor for matching)



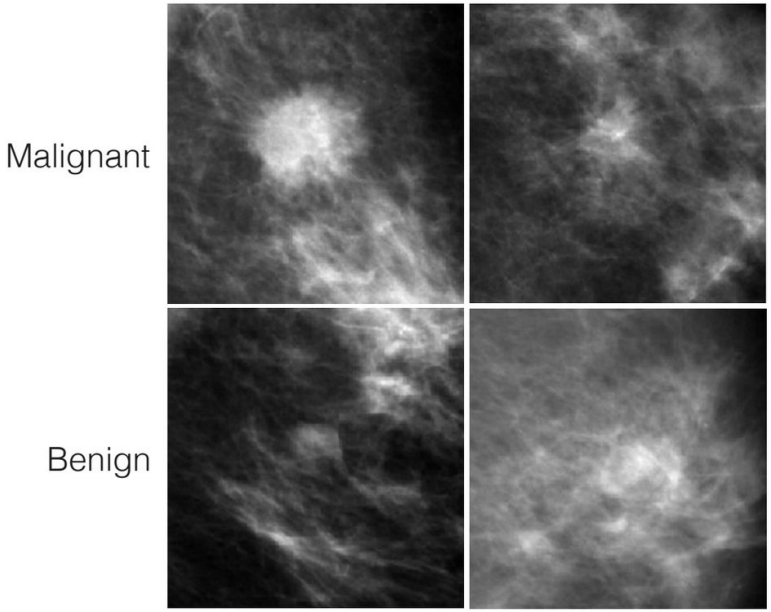
Edge
(object contour)

Quick detour on textures

- Structural approach: a texture is composed of fundamental texture elements, called “textons”.

Bela Julesz, "Textons, the Elements of Texture Perception, and their Interactions". Nature. 1981.

Non-deterministic textures



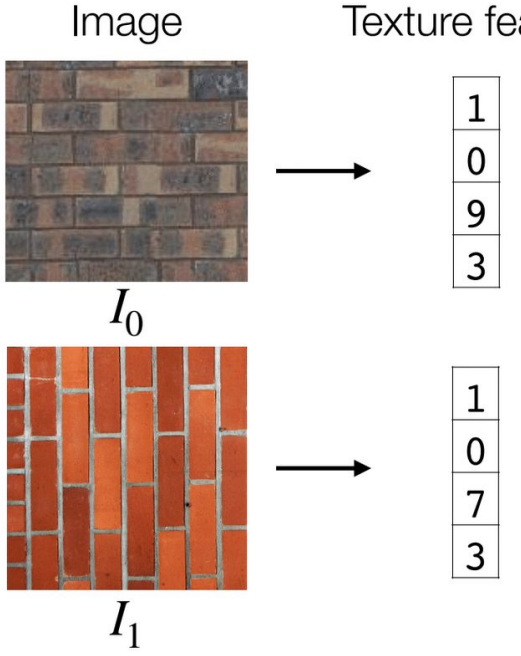
42

medical images (mammography)

- Stochastic approach: a texture is ``a stochastic, possibly periodic, image field''

G. R. Cross and A. K. Jain. Markov Random Field Texture Models. 1983

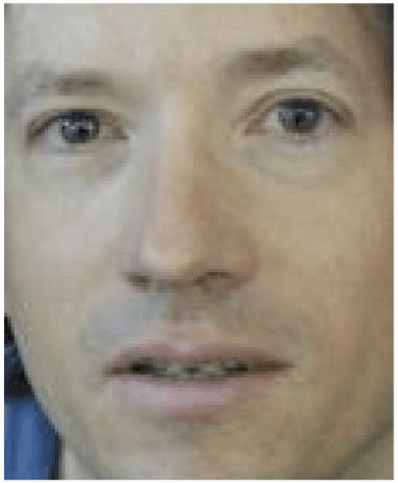
We want series of filters that can yield textural invariances



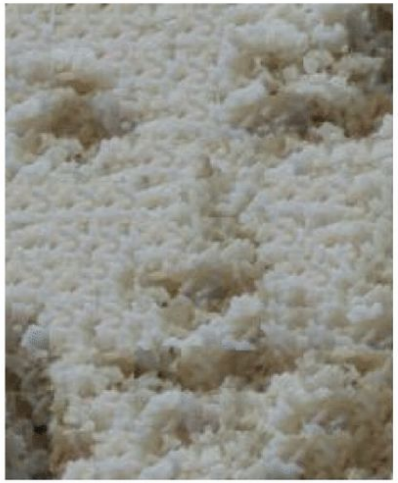
Desired property: **Invariant (intra-class)**

- The features have similar values for the same texture despite the pixel-level difference
- e.g. robust to color, orientation changes

We can also use non-linear filters to retexture input images (generative models)



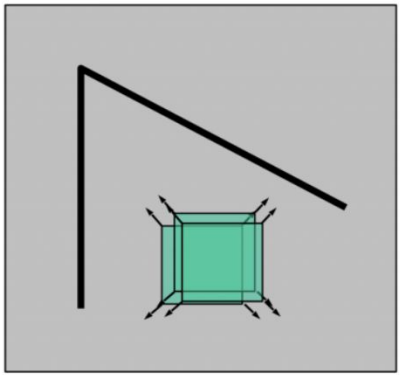
input images



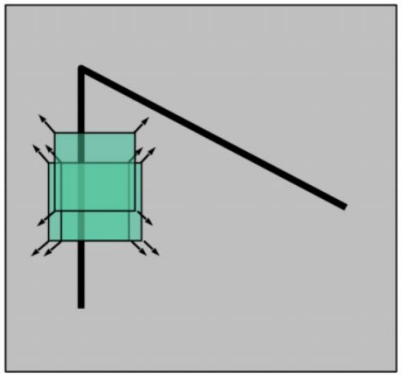
quilting results

Quick detour on corner detection

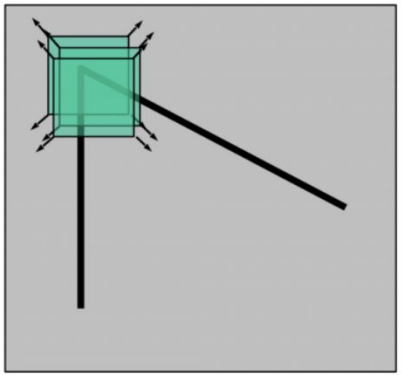
Intuition: Find the patches where a window shift in *any direction* to give *a large change* in intensity.



"flat" region:
no change in any direction

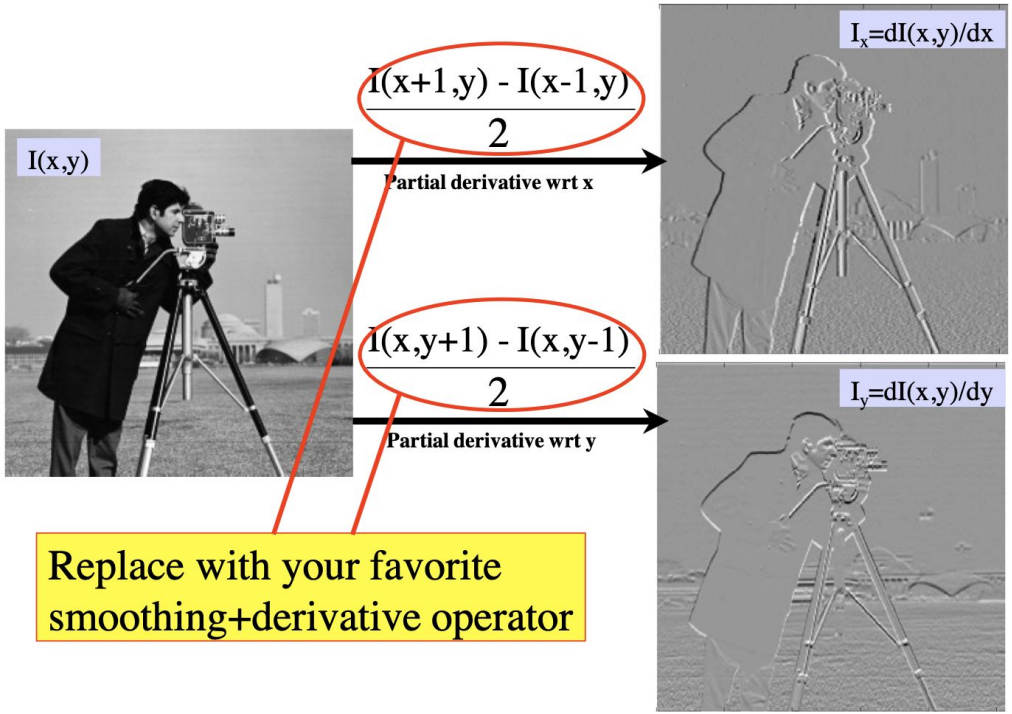


"edge":
no change along the edge direction



"corner":
significant change in all directions [46]

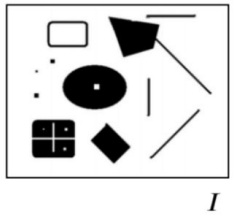
Another quick detour for computing “change”



Harris Corner Detector [Harris88]

James Hays

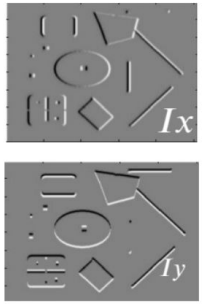
Input image



I

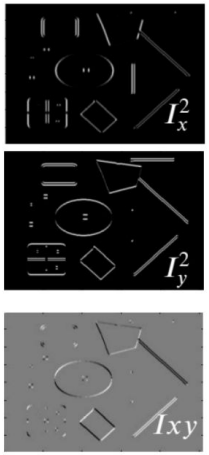
Step 1. Edges

-1(black)~1(white)



Step 2. Compute **M**

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix}$$



Step 3. Compute cornerness

$$C = \det(M) - \alpha \, \mathrm{trace}(M)^2$$

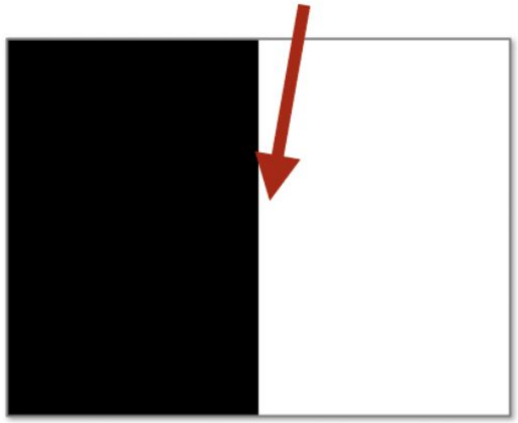
Step 4. Threshold on C + Non-maximal suppression

Corner detection result

47

Edge detection

Edge: discontinuity between image appearance

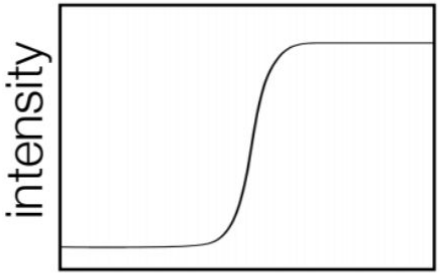


Application: cell segmentation (grouping)

Input image → Edge detection → Individual cells

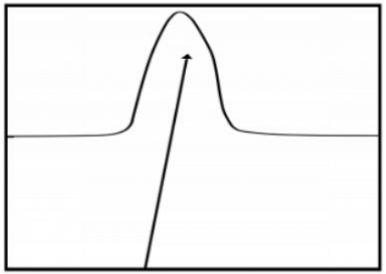
Edge = an area of maximum change

Image



1D profile

$f(x)$



Edge filter

$$f(x+1) - f(x) = f(x) * [1, -1]$$

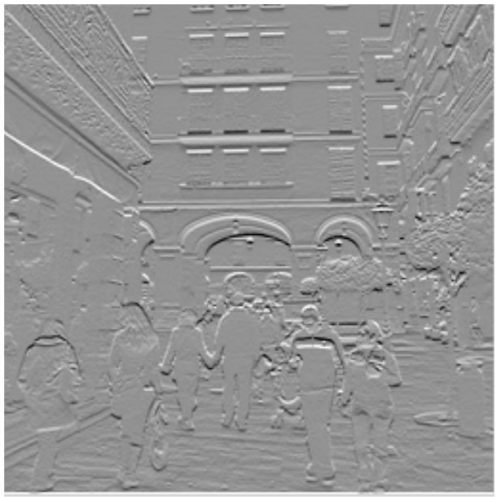
f[0]	f[1]	f[2]	$* [1, -1] =$	f[1]-f[0]	f[2]-f[1]

(flip kernel+dot product)

Horizontal edge: $[-1\ 1]^T$

Input image

$* \begin{bmatrix} -1, \\ 1 \end{bmatrix} =$



Horizontal edge

Vertical edge: [-1 1]

Input image

$* \ [-1, 1] \ =$

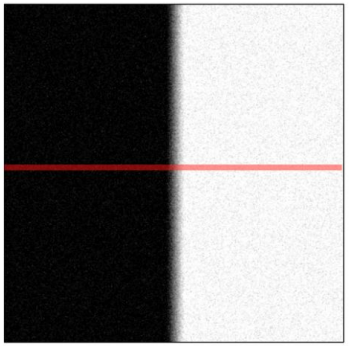
Vertical edge

Canny edge detector

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

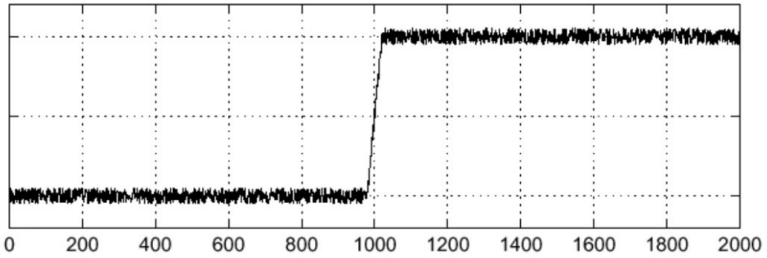
- The choice of σ depends on desired behavior
  - large σ detects "large-scale" edges
  - small σ detects fine edges

Idea 1: Robust to noise

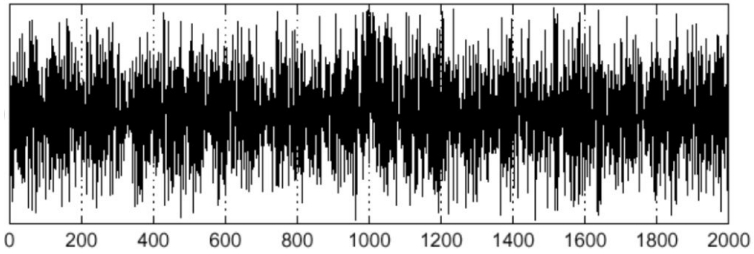


Noisy input image

$f(x)$



$* [1, -1]$

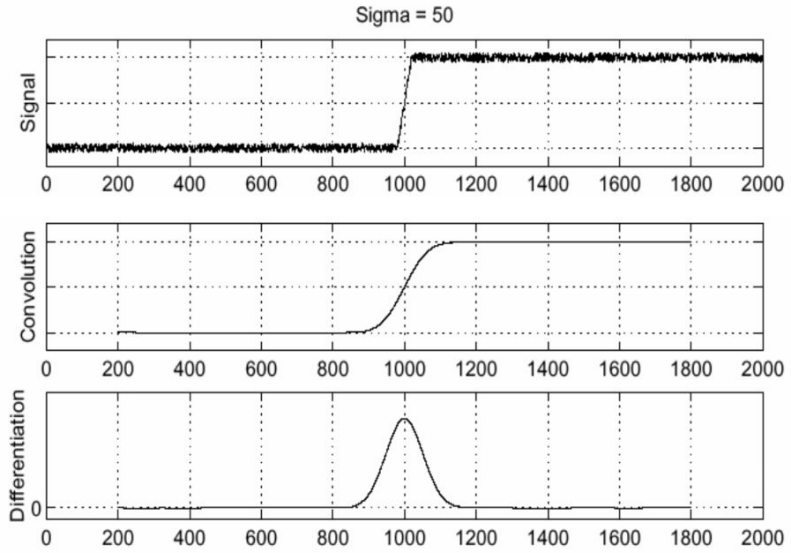


Solution: smooth image first

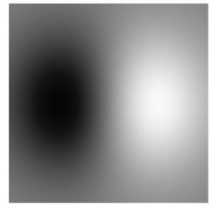
I

$I * g$

$(I * g) * [1, -1]$

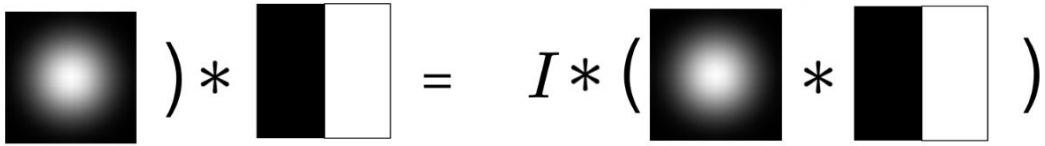


Combined into 1 filter



Associativity of convolution

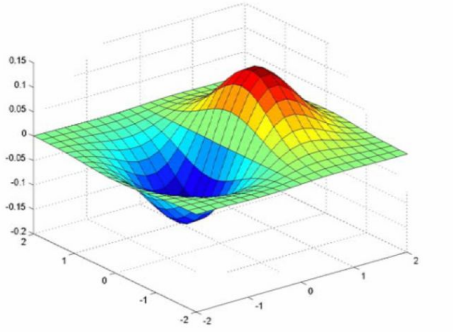
$$\left(I * \right) * = I * \left(* \right)$$



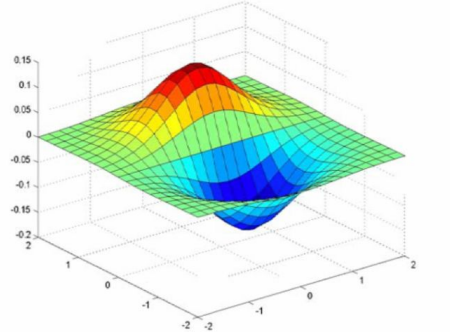
(a+b)+c = a+(b+c)

55

Derivative of Gaussian filter



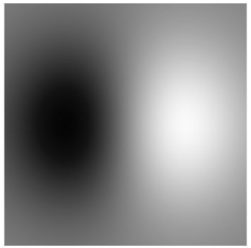
x-direction



y-direction

Naive edge filter

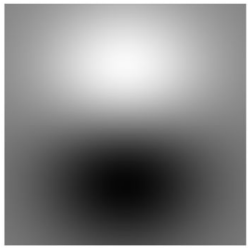
[-1, 1]



DoG-x

Robust edge filter

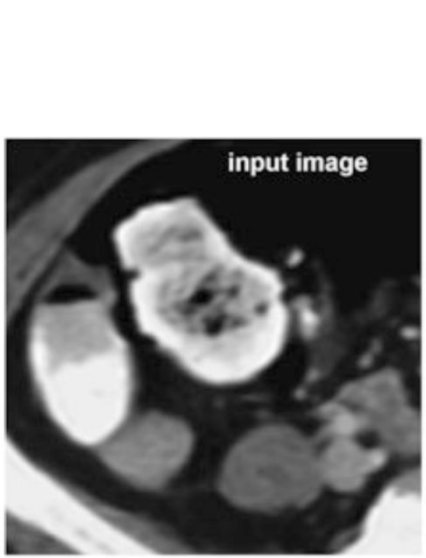
$[-1, 1]^T$



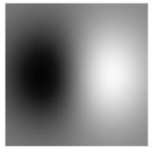
DoG-y

56

Edge: Norm of the gradient vector

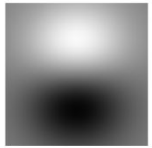


$*$

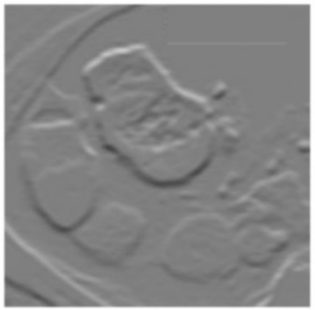


DoG-x

$*$



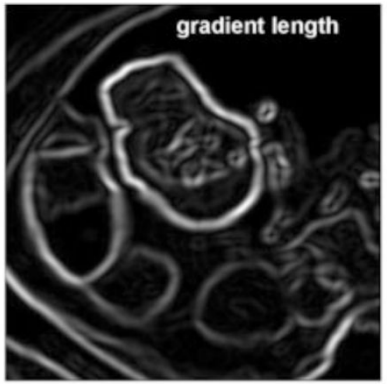
DoG-y



At each pixel, we have the gradient vector: $(I_x(I), I_y(I))$

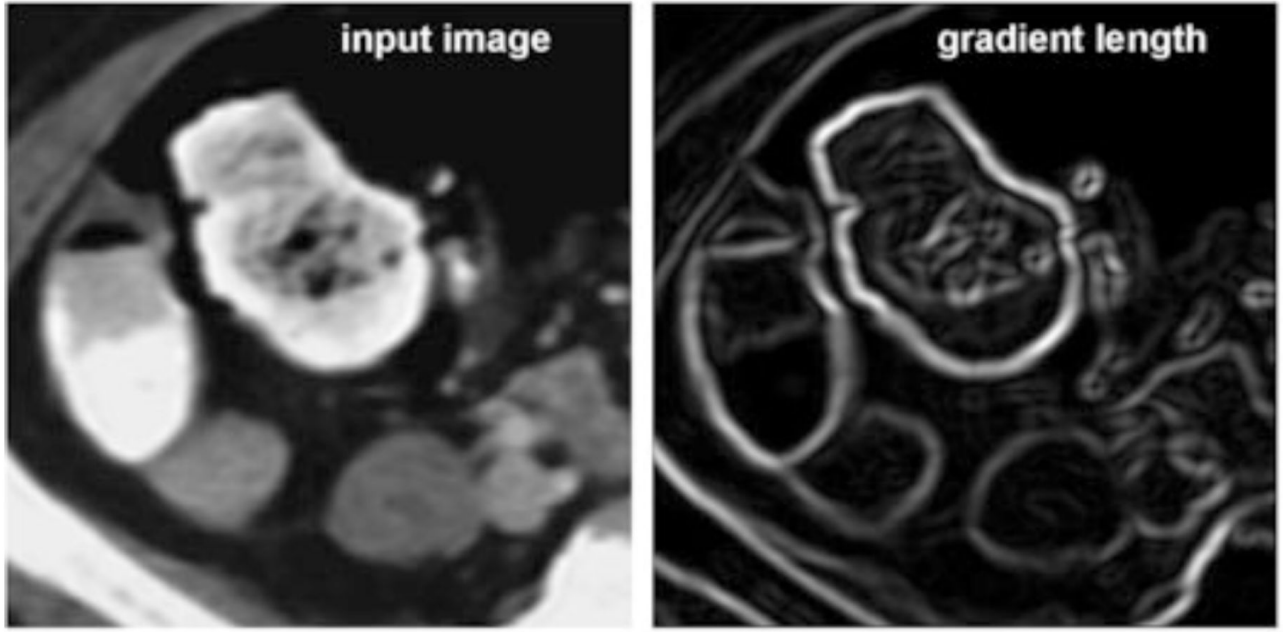
$$\sqrt{[I_x(I)^2 + I_y(I)^2]}$$

Edge magnitude



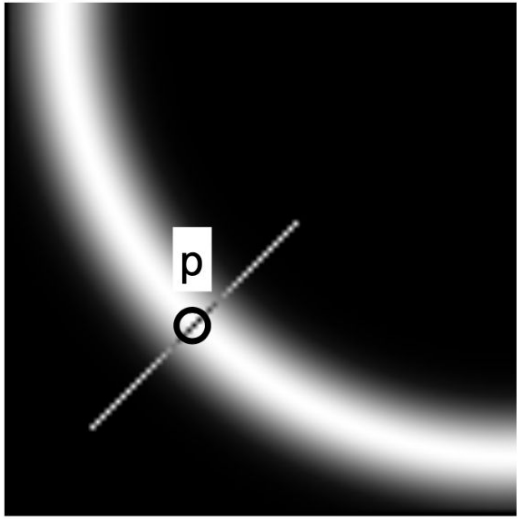
*We lose linearity here

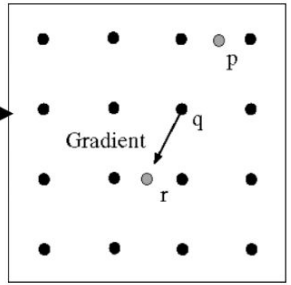
Idea 2: Need thin/sharp edge

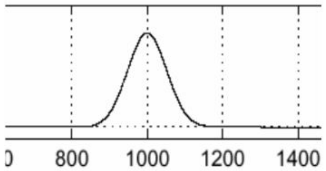


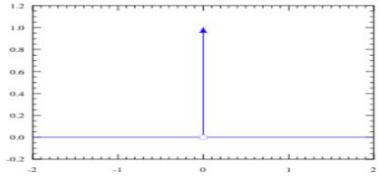
Edge is too thick!

Non-maximum suppression (NMS)







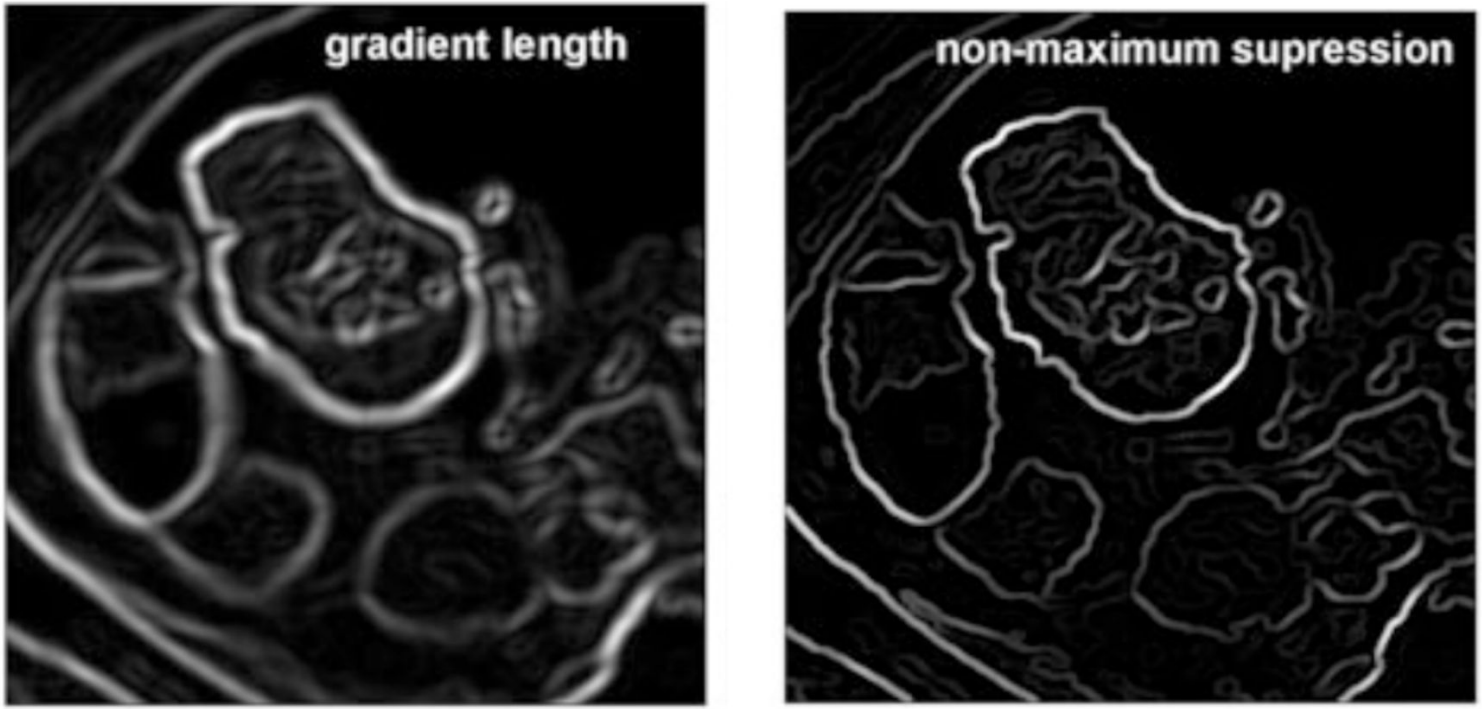


Thick edge

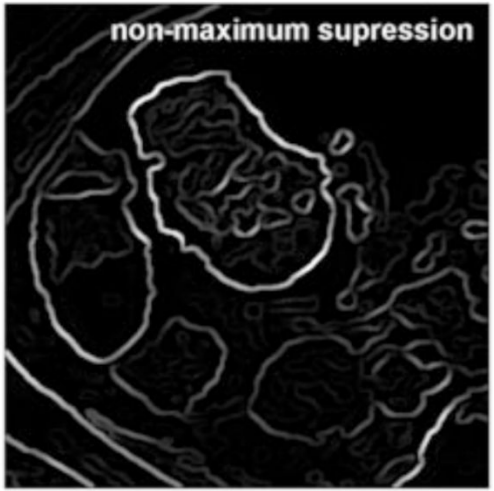
Profile along the gradient direction

suppress non-max to 0

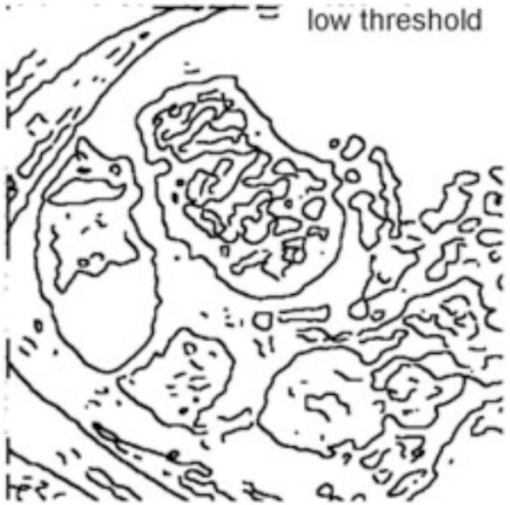
After Non-max Suppression (NMS)



Idea 3: Two types of edges



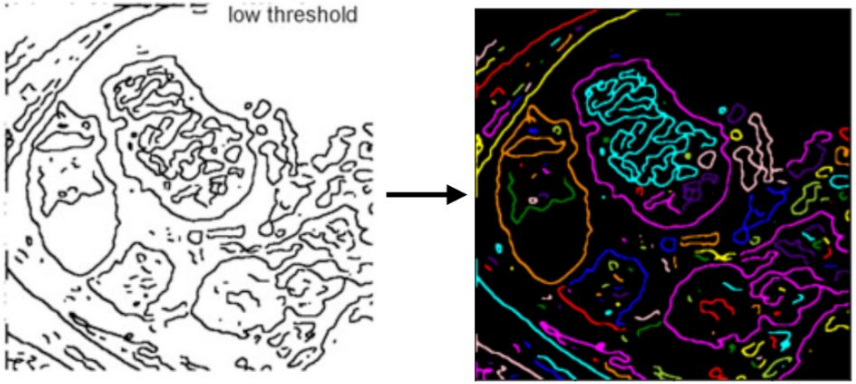
- Good: strong edges
- Bad: broken/short edges



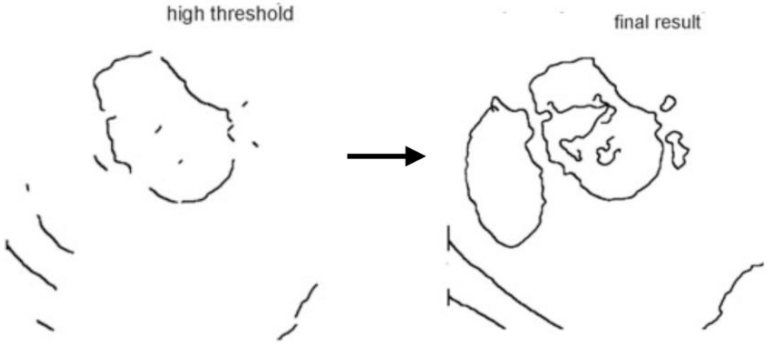
- Good: long edges
- Bad: noisy weak edges

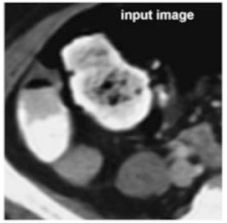
Connecting edges (hysteresis thresholding)

1. Find connected component

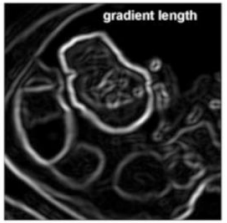


2. Use strong edge as **seeds** and select weak components

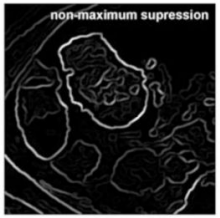




Put it together: Canny edge detector



1. Filter image with derivative of Gaussian and compute magnitude (P1. Robust to noise)



2. Non-maximum suppression (P2. Thin edges)

3. Linking and thresholding (hysteresis): (P3. Long/Strong edges)

- Define two thresholds: low and high
- Use the high threshold to start edge curves and the low threshold to continue them

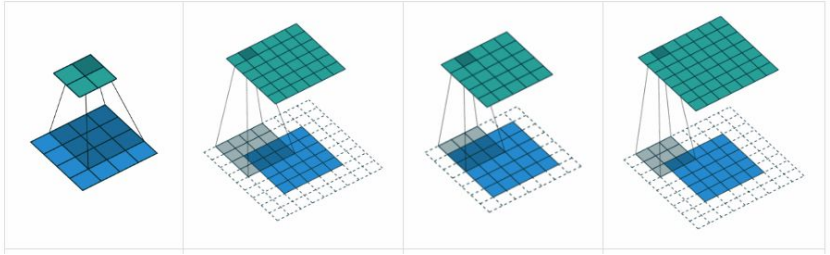
63

Recap of today

Showed convolutions as the basic building block of most filtering operations

Used combinations of simple filtering to yield more complex filters from images

Each filter has its own implicit target information within an image (which makes it ripe for learning!)

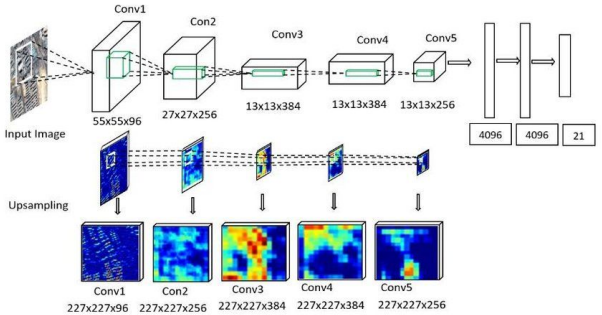


pixels

patches

243	239	240
242	239	218
243	242	123

features



Low level → High level

Next: code lab

Next week: Feature detection and matching