

Computer Vision

CS-E4850, 5 study credits

Juho Kannala
Aalto University



Lecture 2: Image Processing

- Lecture concentrates on image filtering
- Relevant reading: Chapter 3 of Szeliski's book

Acknowledgement: many slides from James Hays, Derek Hoiem, Svetlana Lazebnik, Esa Rahtu, Steve Seitz, David Lowe, Kristen Grauman, Alexei Efros, and others.

Three views of filtering

- Image filters in spatial domain
 - Filter is a mathematical operation of a grid of numbers
 - Smoothing, sharpening, edge detection
- Image filters in the frequency domain
 - Filtering is a way to modify the frequencies of images
 - Hybrid images, sampling, image resizing
- Templates and image pyramids
 - Filtering is a way to match a template to the image
 - Detection, coarse-to-fine registration

Image filtering

- Image filtering: compute function of local neighborhood at each position
- Really important in practice!
- Enhance images (Denoise, resize, increase contrast, etc.)
- Extract information from images (Texture, edges, distinctive points, etc.)
- Detect patterns (Template matching)
- Deep Convolutional Networks (Sequence of filters and non-linear functions)

Motivation: Image denoising

- How can we reduce noise in a photograph?



Moving average

- Let's replace each pixel with a *weighted* average of its neighborhood
- The weights are called the *filter kernel*
- The weights for the average of a 3x3 neighborhood:

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

1	1	1
1	1	1
1	1	1

“box filter”

Image filtering

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

$g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10							

$g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

 $f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

 $h[\cdot, \cdot]$

	0	10	20						

 $g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30					

$g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30	30				

$g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30	30				

$g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

 $f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

 $h[\cdot, \cdot]$

	0	10	20	30	30				
							?		
				50					

 $g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

 $f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

 $h[\cdot, \cdot]$

		0	10	20	30	30	30	20	10
		0	20	40	60	60	60	40	20
		0	30	60	90	90	90	60	30
		0	30	50	80	80	90	60	30
		0	30	50	80	80	90	60	30
		0	20	30	50	50	60	40	20
		10	20	30	30	30	30	20	10
		10	10	10	0	0	0	0	0

 $g[\cdot, \cdot] \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Box filter - what does it do?

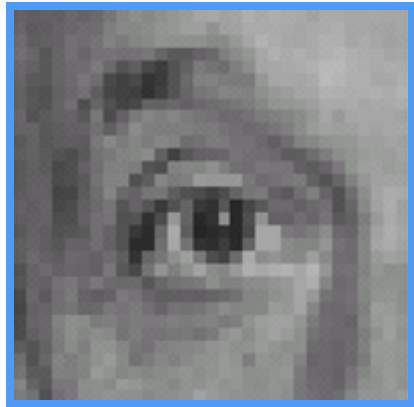
- Replaces each pixel with an average of its neighborhood
- Achieves smoothing effect (removes sharp features)

$$\frac{1}{9} \begin{matrix} & & g[\cdot, \cdot] \\ \begin{matrix} 1 \\ | \\ 9 \end{matrix} & \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \end{matrix}$$

Smoothing with box filter



Practice with linear filters

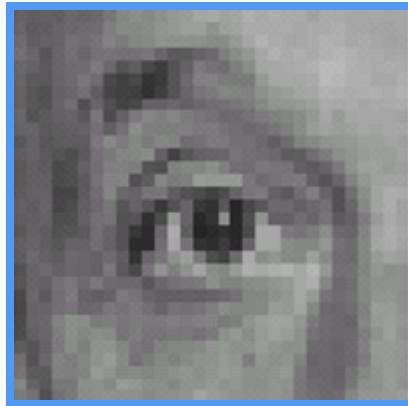


Original

0	0	0
0	1	0
0	0	0

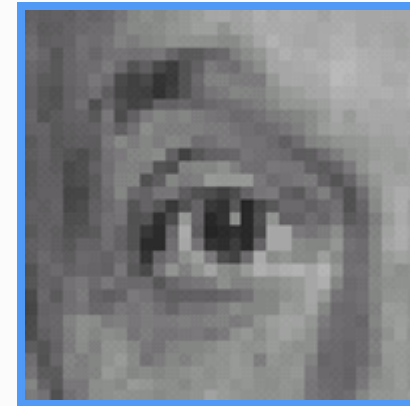
?

Practice with linear filters



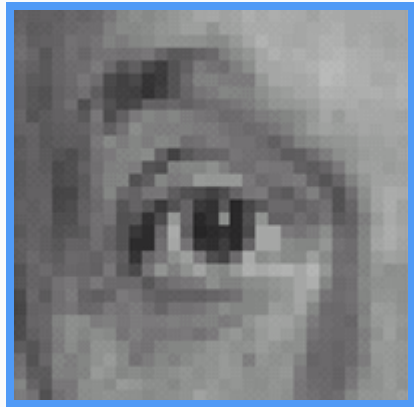
Original

0	0	0
0	1	0
0	0	0



Filtered
(no change)

Practice with linear filters

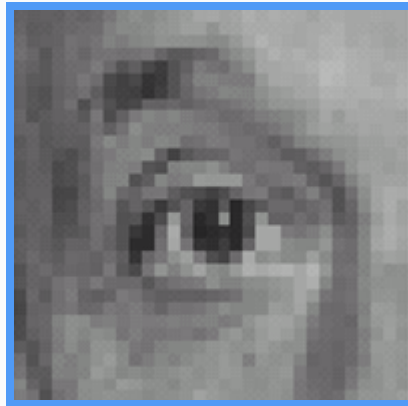


Original

0	0	0
0	0	1
0	0	0

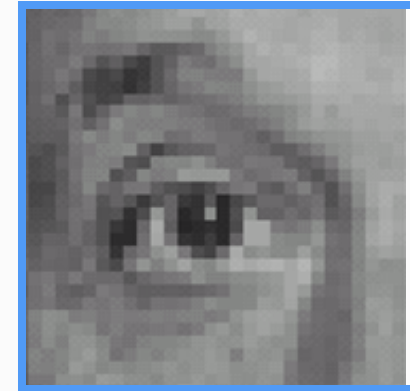
?

Practice with linear filters



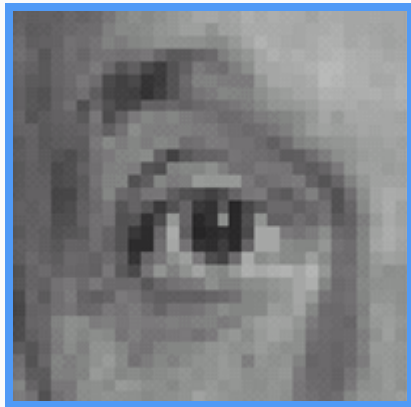
Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

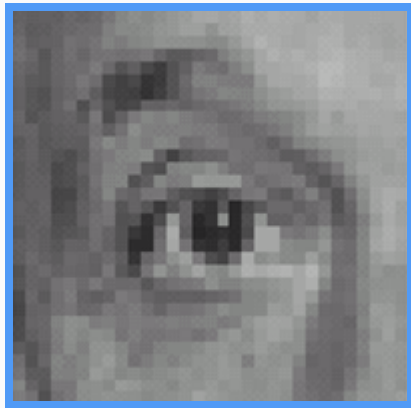
$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

(Note that filter sums to 1)

Practice with linear filters



Original

0	0	0
0	2	0
0	0	0

-

$\frac{1}{9}$

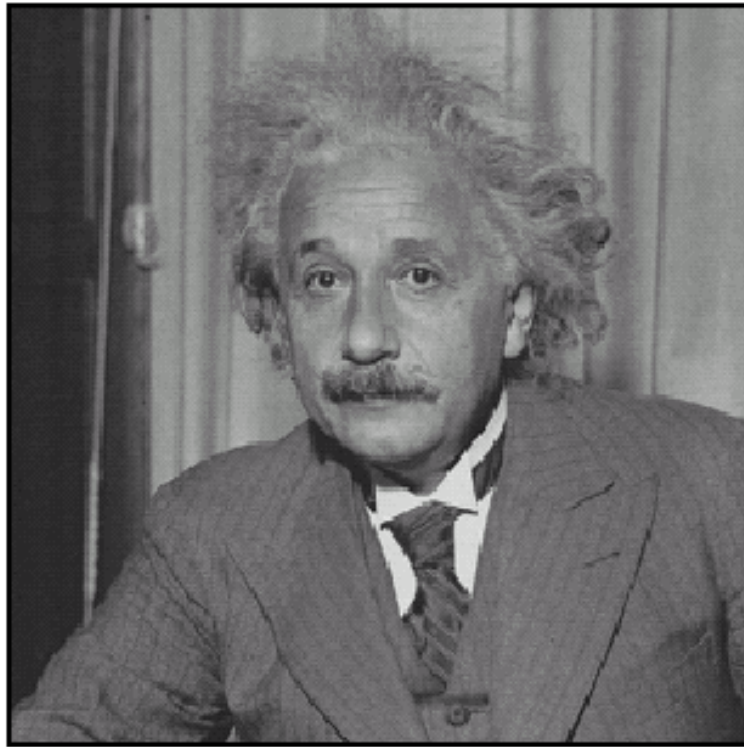
1	1	1
1	1	1
1	1	1



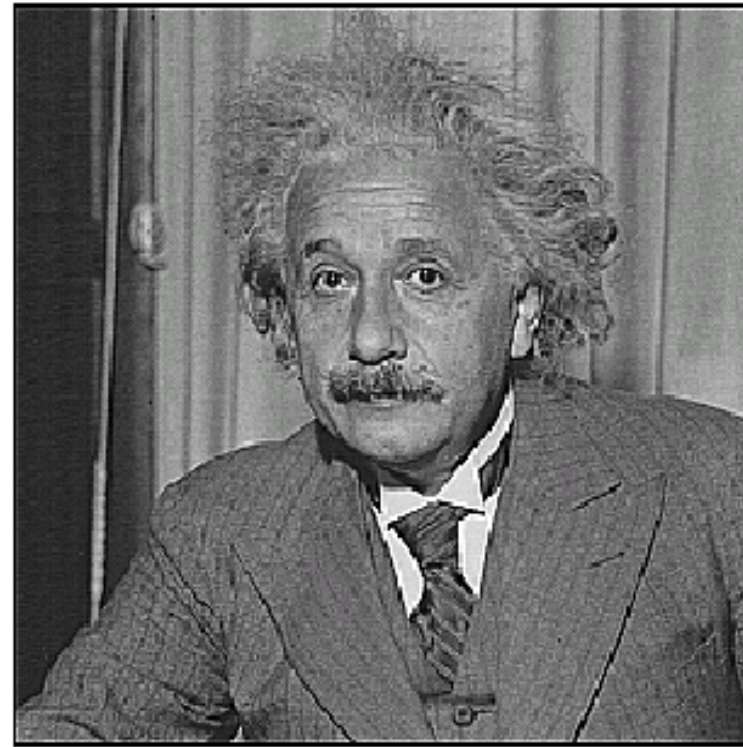
Sharpening filter

- Accentuates differences with local average

Sharpening

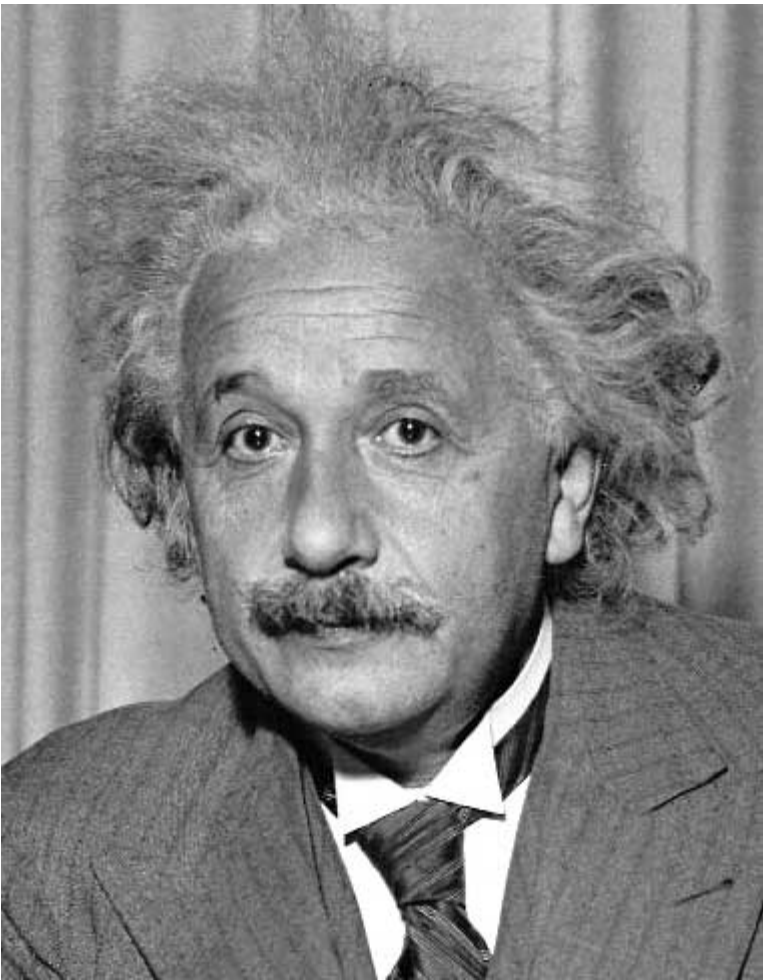


before



after

Other filters



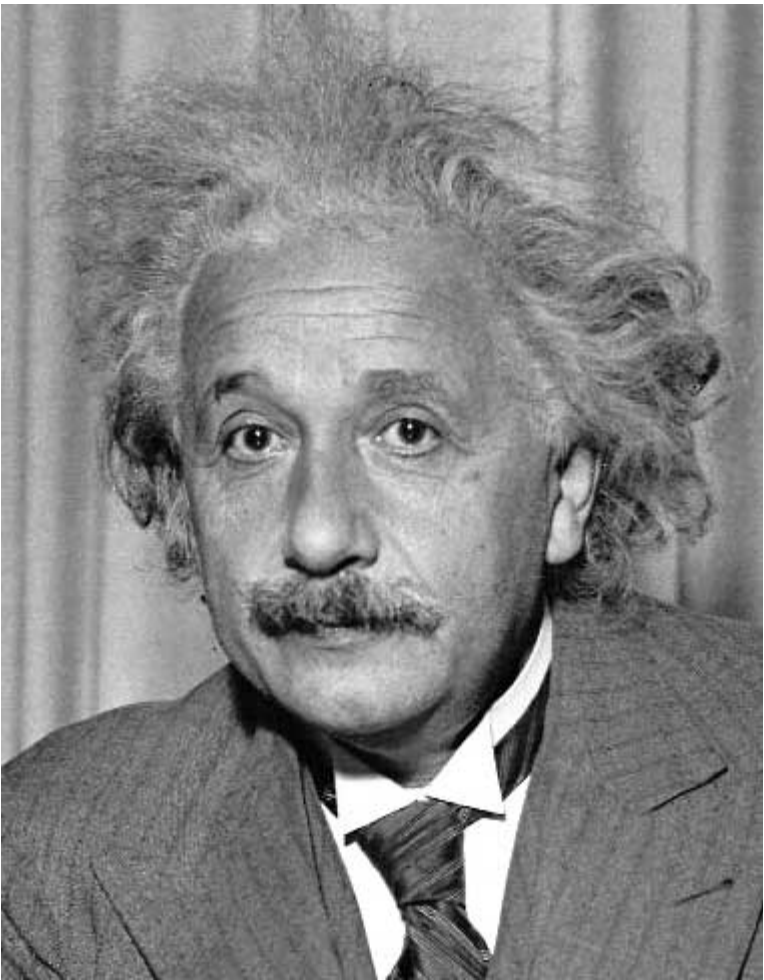
1	0	-1
2	0	-2
1	0	-1

Sobel



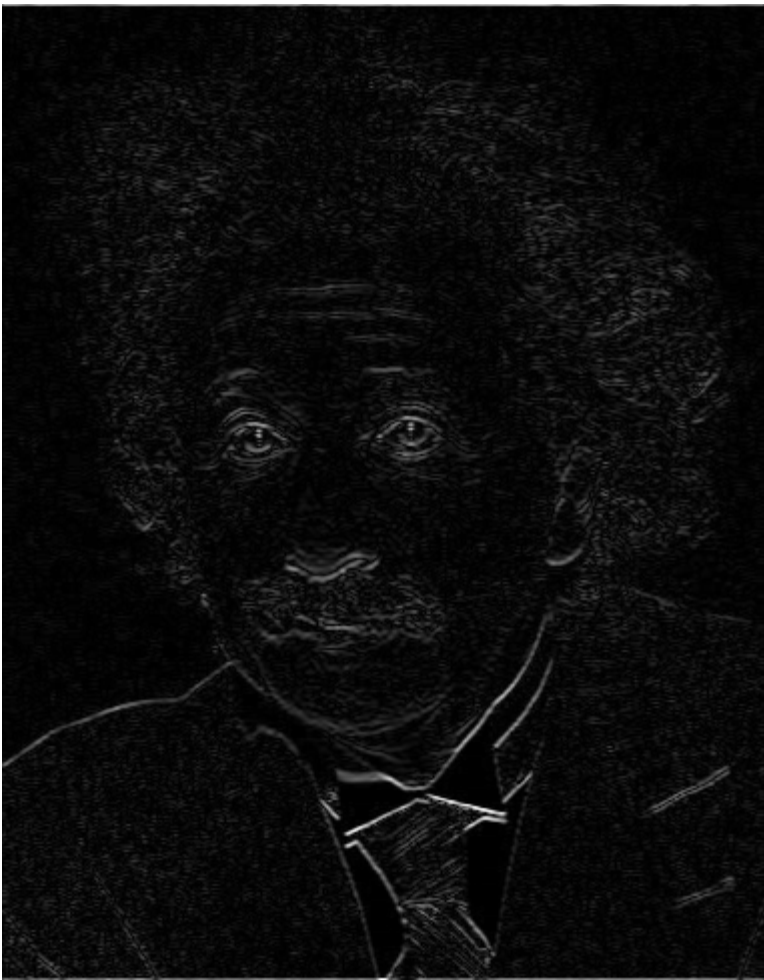
Vertical Edge
(absolute value)

Other filters



1	2	1
0	0	0
-1	-2	-1

Sobel



Horizontal Edge
(absolute value)

Key properties

- Linearity:

$$\text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2)$$

- Shift invariance:

$$\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$$

-> same behavior regardless of pixel location

- Theoretical result: any linear shift-invariant operator can be represented as a convolution

Properties in more detail

- Commutative: $a * b = b * a$
 - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
 - Often apply several filters one after another: $((a * b_1) * b_2) * b_3$
 - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k(a * b)$
- Identity: unit impulse $e = [\dots, 0, 0, 1, 0, 0, \dots]$,
 $a * e = a$

Filtering vs. Convolution

- 2D filtering:

I=image f=filter



`h=filter2(f,I);` or `h=imfilter(I,f);`

$$h[m,n] = \sum_{k,l} f[k,l] I[m+k,n+l]$$

- 2D convolution:

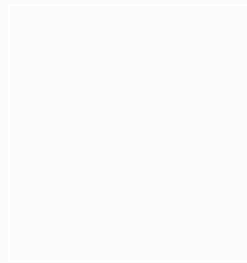
`h=conv2(f,I);`

$$h[m,n] = \sum_{k,l} f[k,l] I[m-k,n-l]$$

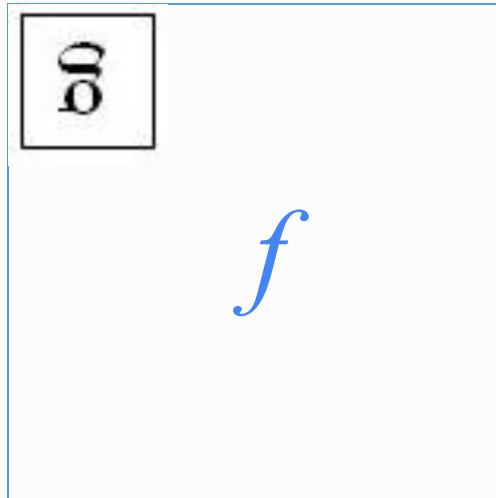
Definition of convolution

- Let f be the image and g be the kernel. The output of convolving f with g is denoted $f * g$

$$(f * g)[m, n] = \sum_{k, l} f[m - k, n - l] g[k, l]$$



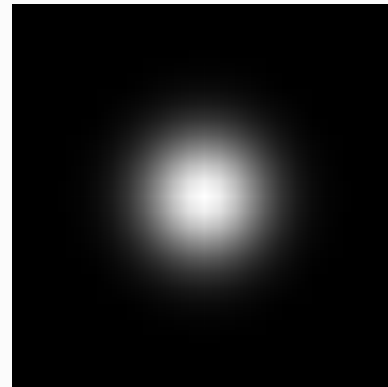
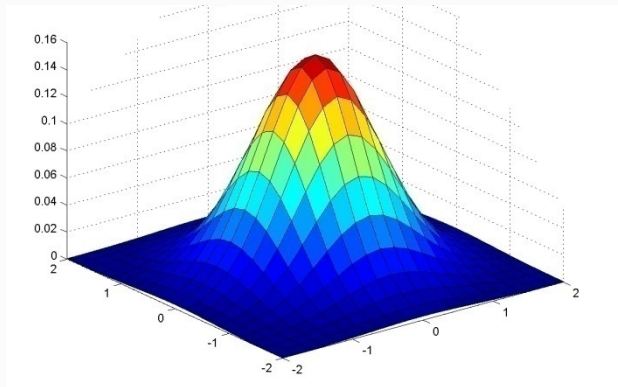
Convention:
kernel is “flipped”



- See MATLAB functions: [conv2](#), [filter2](#), [imfilter](#) (the latter two don't flip the kernel)

Important filter - Gaussian

- Spatially-weighted average

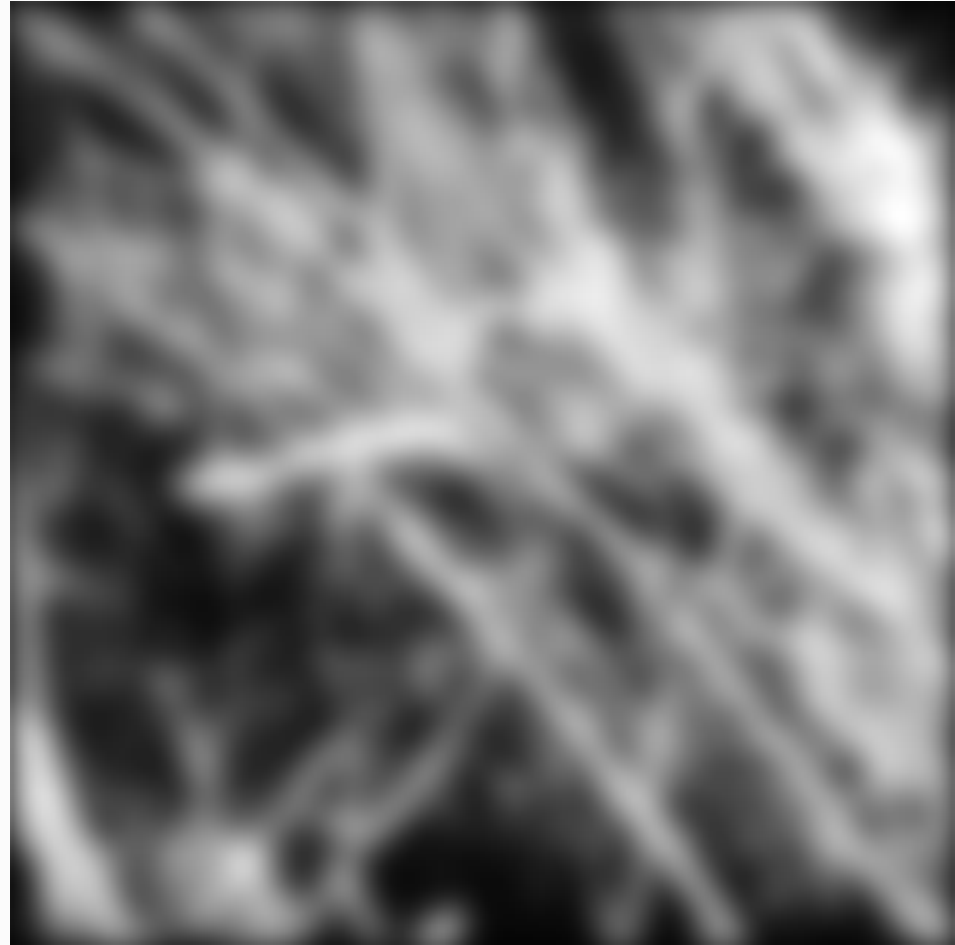


0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

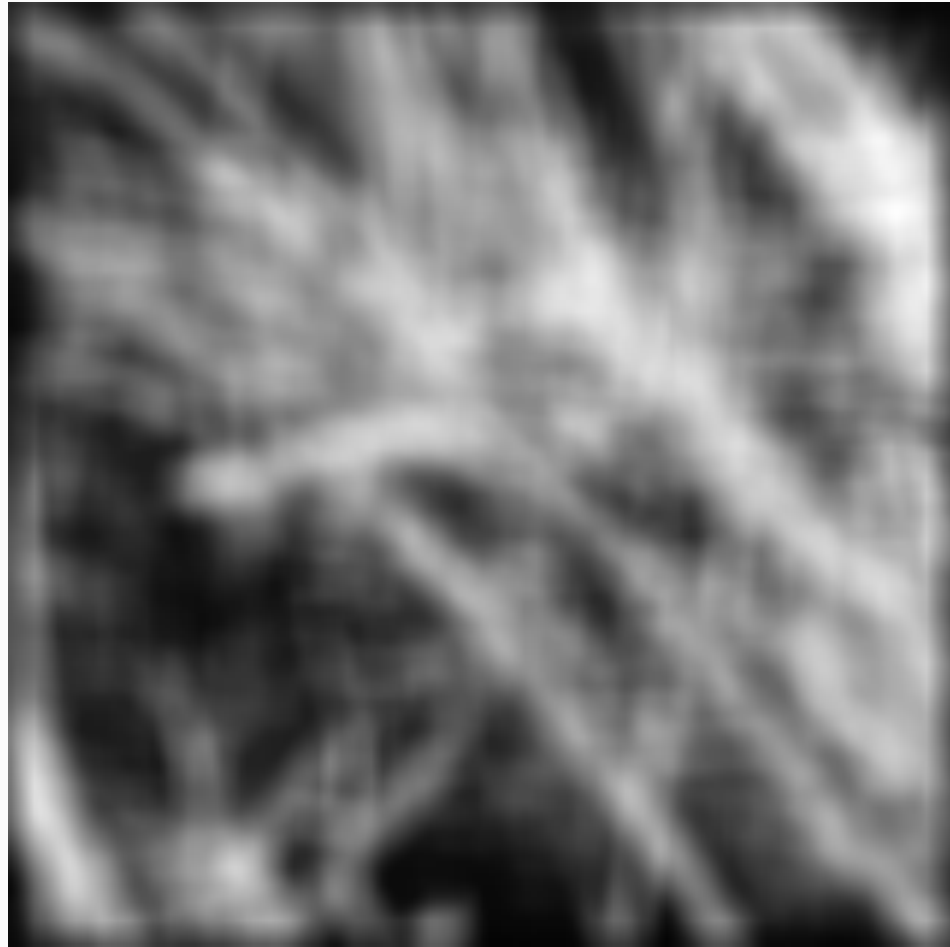
5 x 5, $\sigma = 1$

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

Smoothing with Gaussian filter



Smoothing with box filter



Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
 - Images become more smooth
- Convolution with self is another Gaussian
 - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
 - Convoluting two times with Gaussian kernel of width σ is same as convoluting once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
 - Factors into product of two 1D Gaussians

Separability of the Gaussian filter

$$\begin{aligned} G_{\sigma}(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)\right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2\sigma^2}\right)\right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Separability example

2D filtering
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

The filter factors
into a product of 1D
filters:

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

Perform filtering
along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix}$$

Followed by filtering
along the remaining column:

Separability

- Why is separability useful in practice?

Separability

- Why is separability useful in practice?
- Filter of size $k \times k$ requires k^2 operations per pixel
- Only $2k$ operations for separable kernels:

$$\mathbf{K} = \mathbf{v}\mathbf{h}^T$$

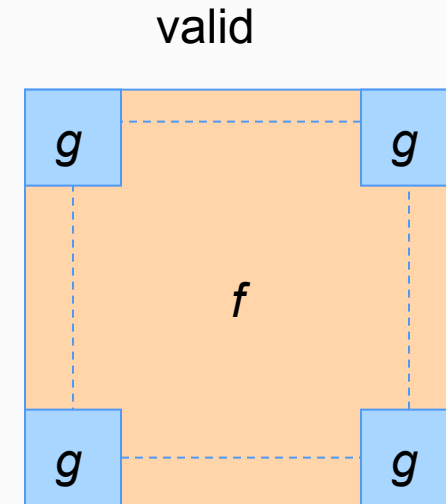
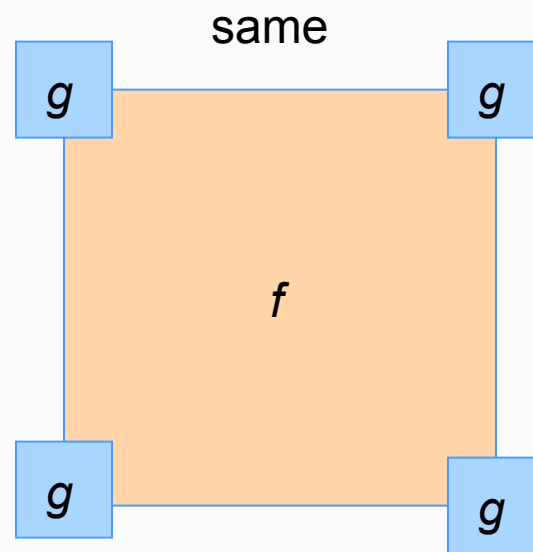
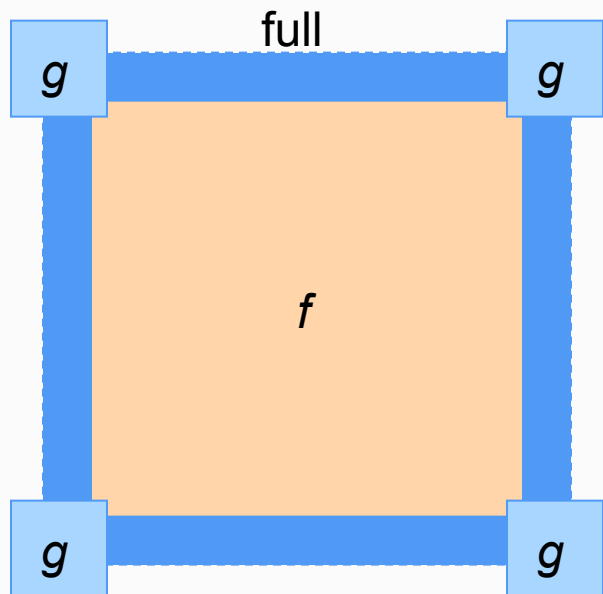
Practical matters – what happens near the edge?

- The filter window falls off the edge of the image
- Need to extrapolate:
 - clip filter (black)
Matlab: `imfilter(f, g, 0)`
 - wrap around
Matlab: `imfilter(f, g, 'circular')`
 - copy edge
Matlab: `imfilter(f, g, 'replicate')`
 - reflect across edge
Matlab: `imfilter(f, g, 'symmetric')`



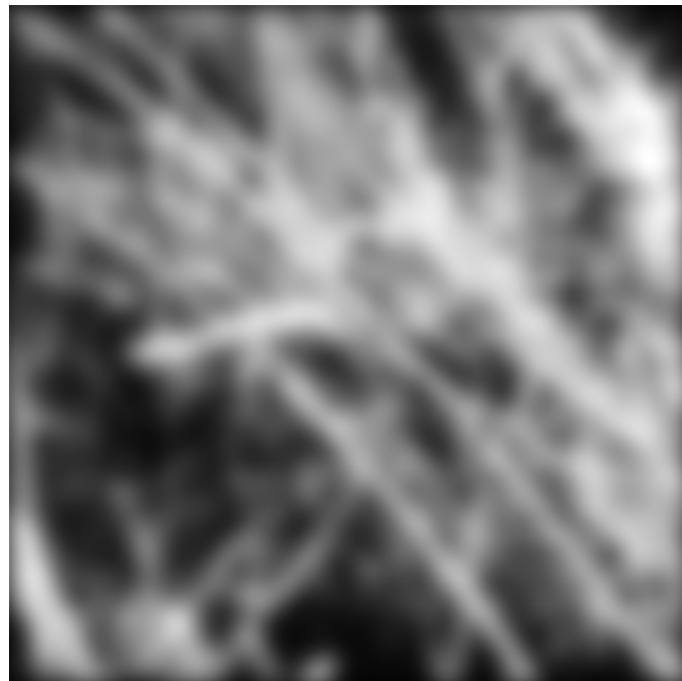
Practical matters

- What is the size of the output?
- Matlab: `filter2(g, f, shape)`
 - *shape* = 'full': output size is sum of sizes of *f* and *g*
 - *shape* = 'same': output size is same as *f*
 - *shape* = 'valid': output size is difference of sizes of *f* and *g*

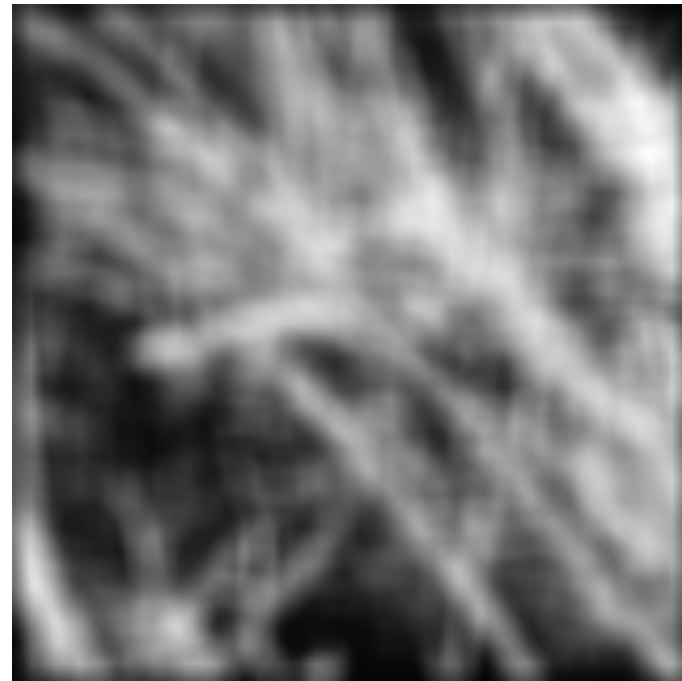


Why Gaussian gives smooth output compared to box filter?

Gaussian



Box filter



Why lower resolution image still make sense? What is lost?



Thinking in terms of frequency

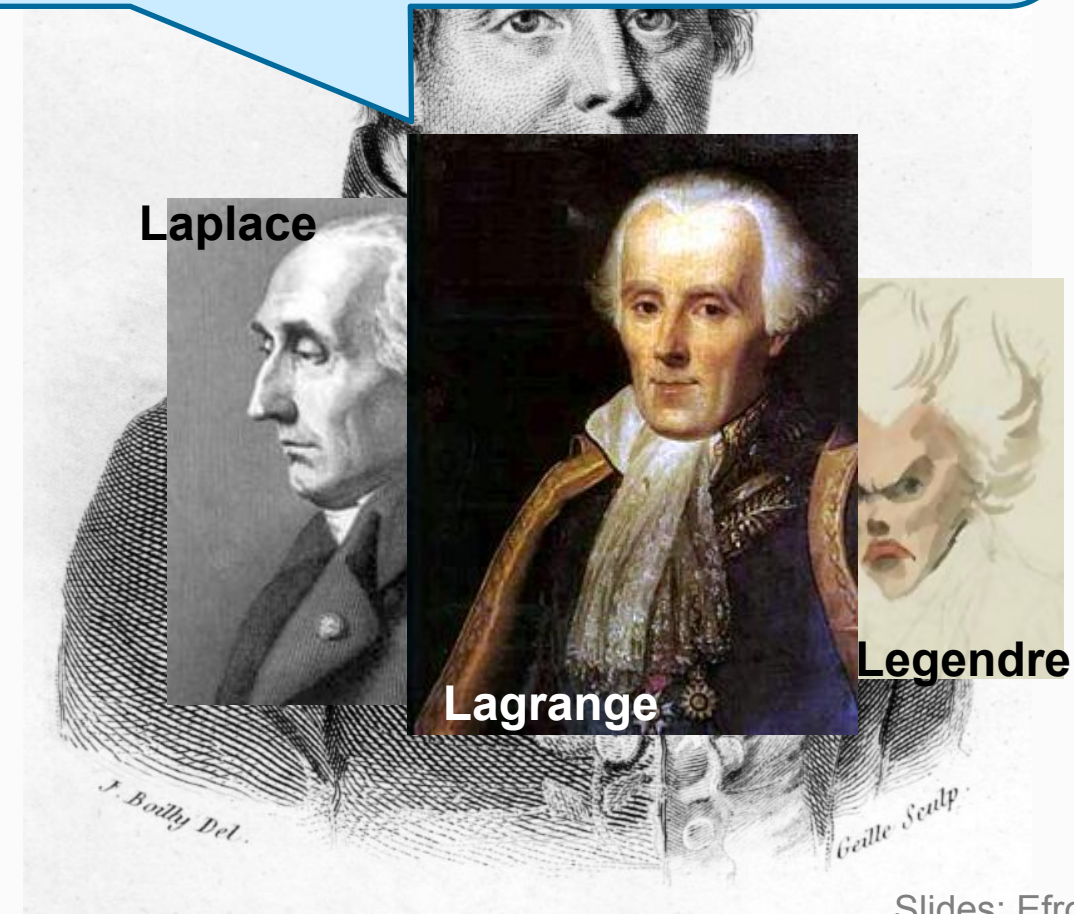
Jean Baptiste Joseph Fourier (1768-1830)

- He had a crazy idea in 1807:

Any univariate function can be rewritten as a weighted sum of sines and cosines of different frequencies.

- Don't believe it?
 - Neither did Lagrange, Laplace, Poisson and other big wigs
 - Not translated into English until 1878!
- But it's (mostly) true!
 - Called Fourier Series
 - There are some subtle restrictions

...the manner in which the author arrives at these equations is not exempt of difficulties and...his analysis to integrate them still leaves something to be desired on the score of generality and even rigour.

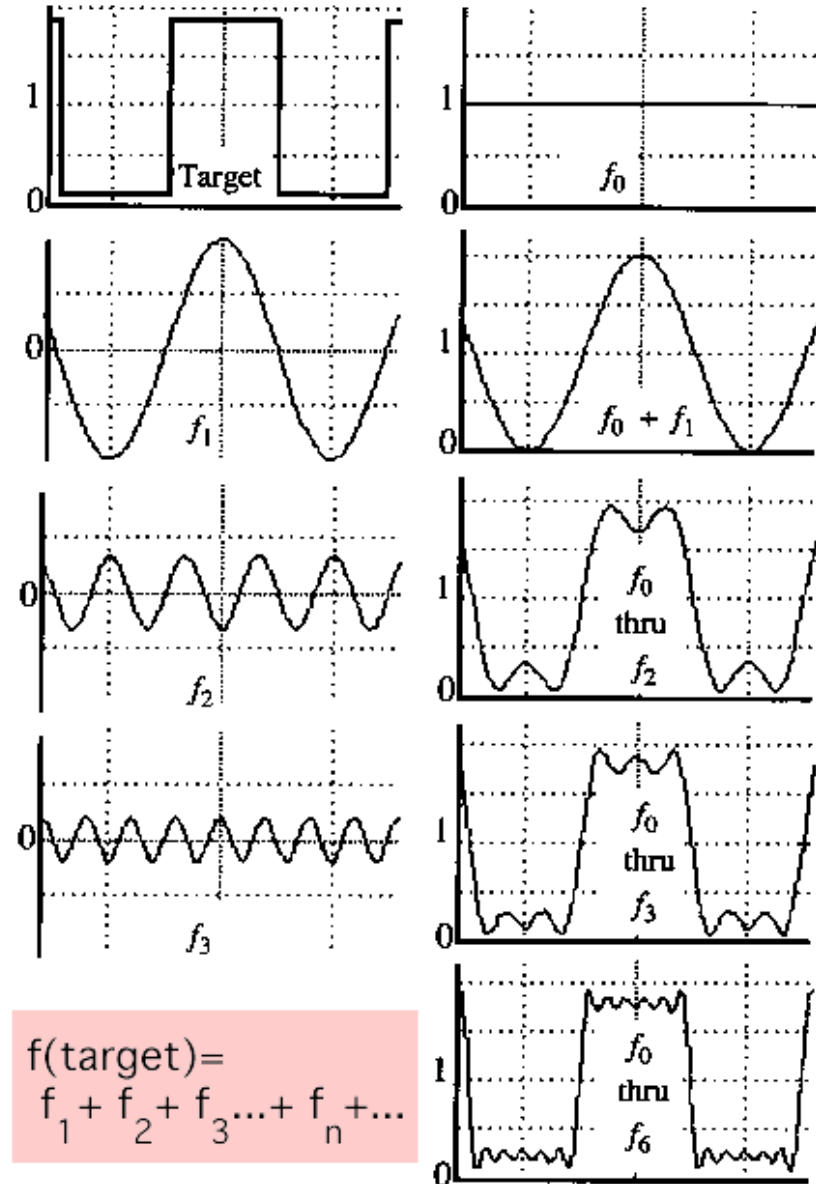


A sum of sines

- Our building block:

$$A \sin(\omega x + \phi)$$

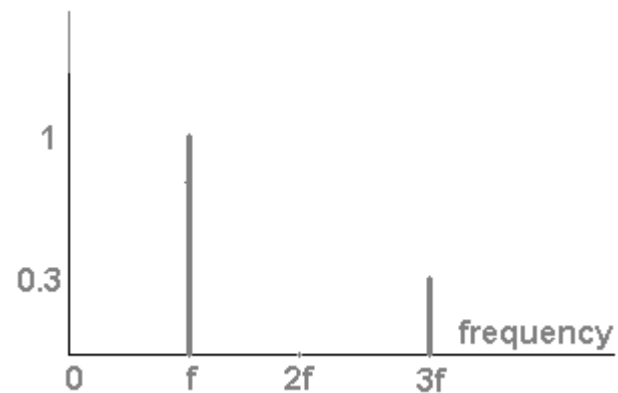
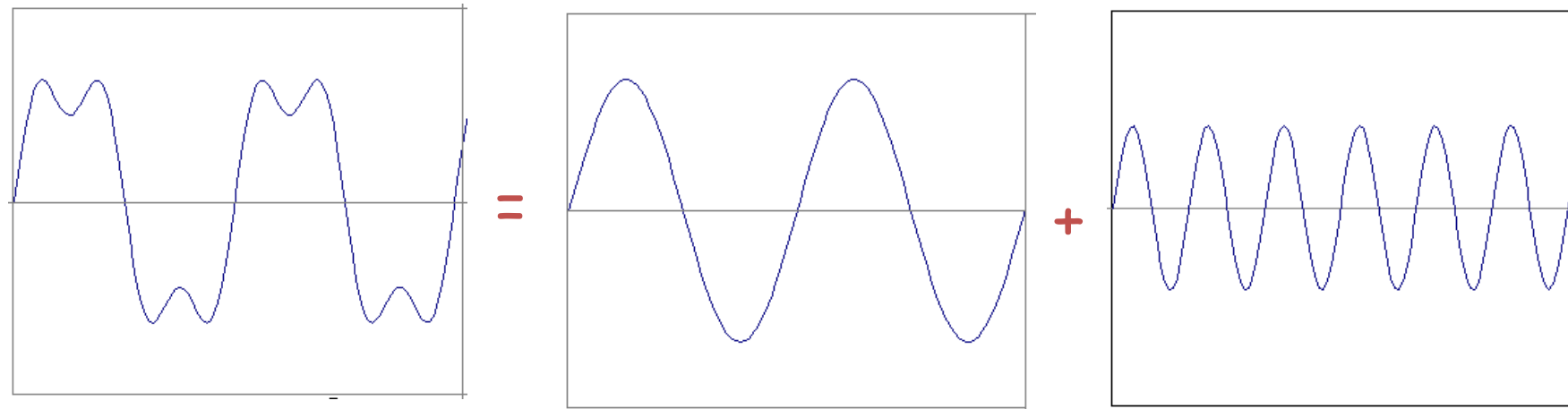
- Add enough of them to get any signal $f(x)$ you want!



A sum of sines

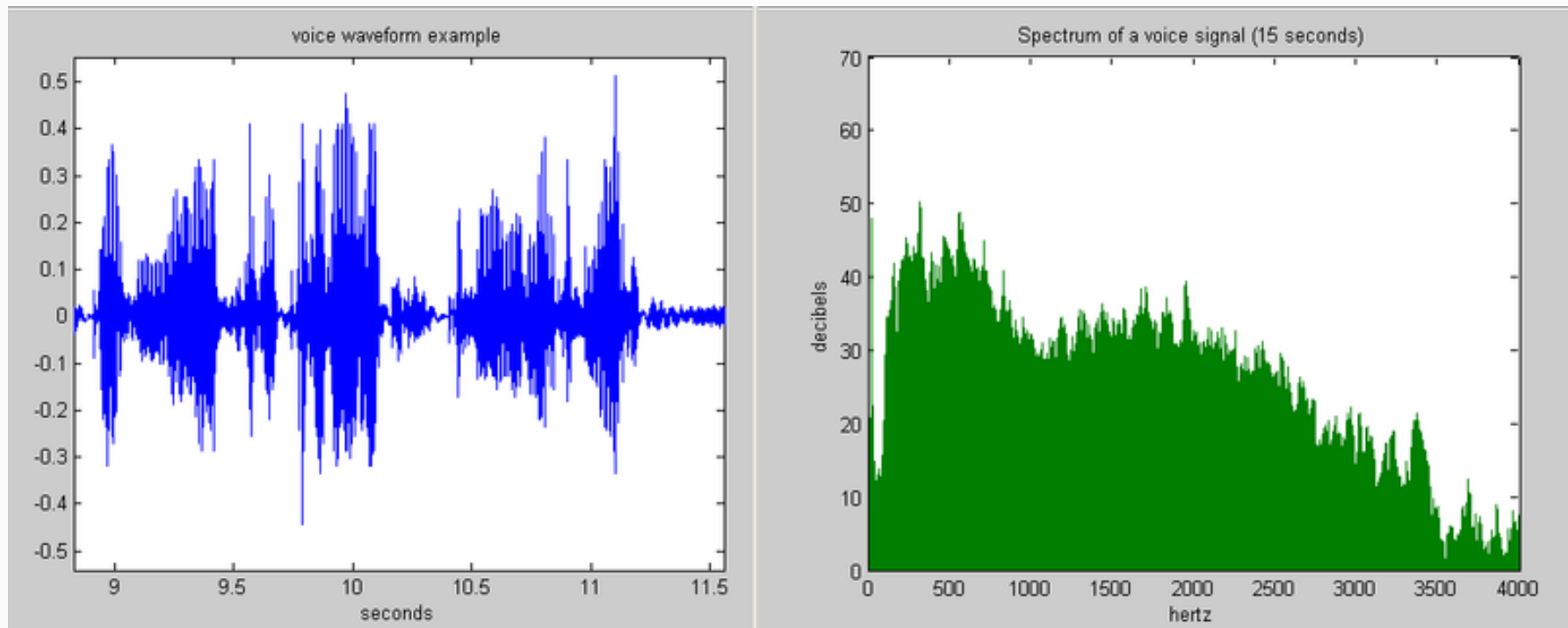
- Example:

$$g(t) = \sin(2\pi f t) + (1/3)\sin(2\pi(3f) t)$$



Example: Music

- We think of music in terms of frequencies at different magnitudes



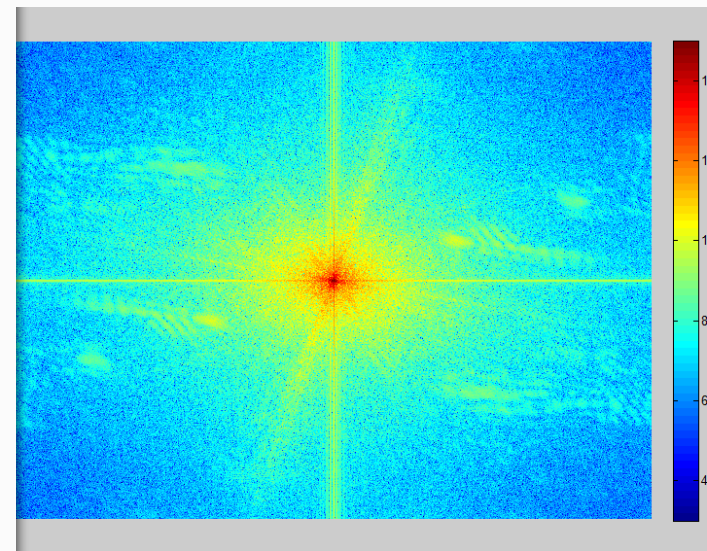
2D signals

- We can also think of all kinds of other signals the same way



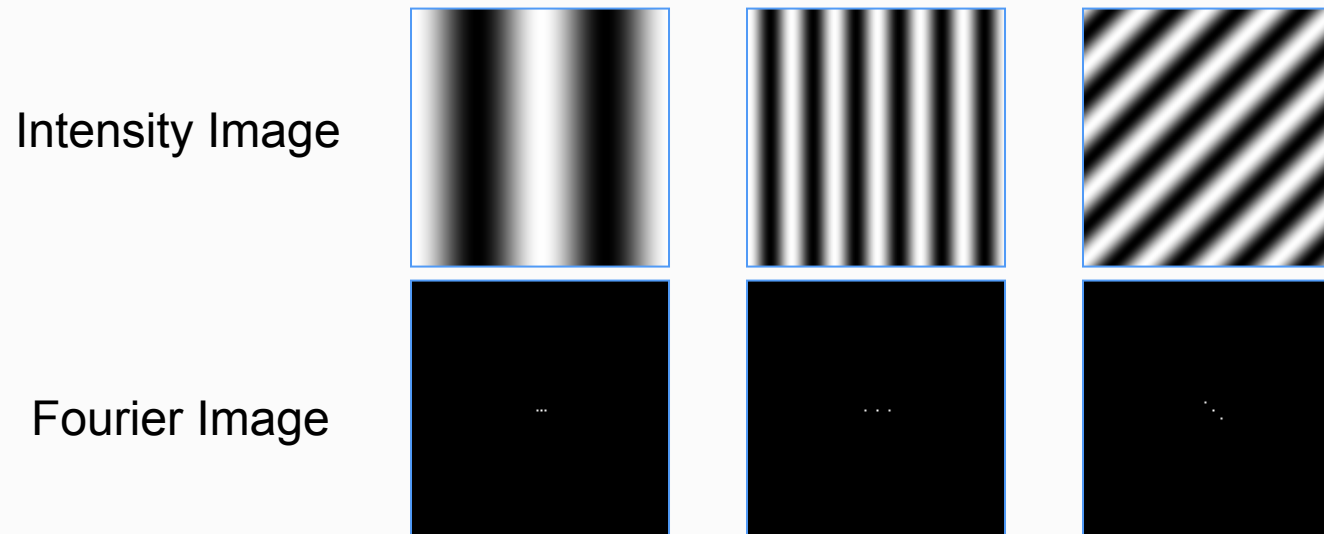
Other signals

- We can also think of all kinds of other signals the same way

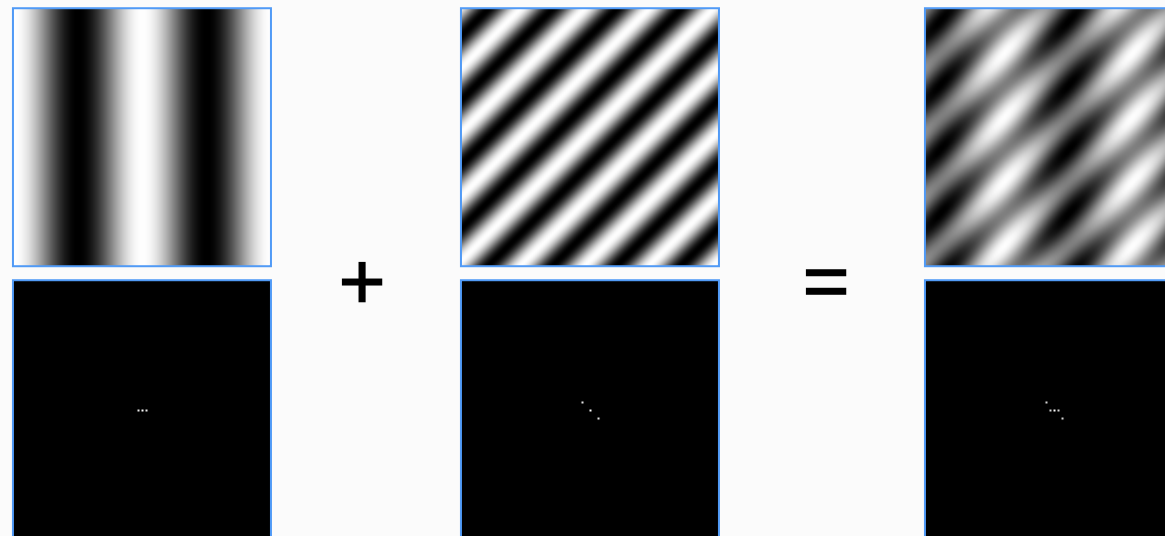


Fourier analysis in images

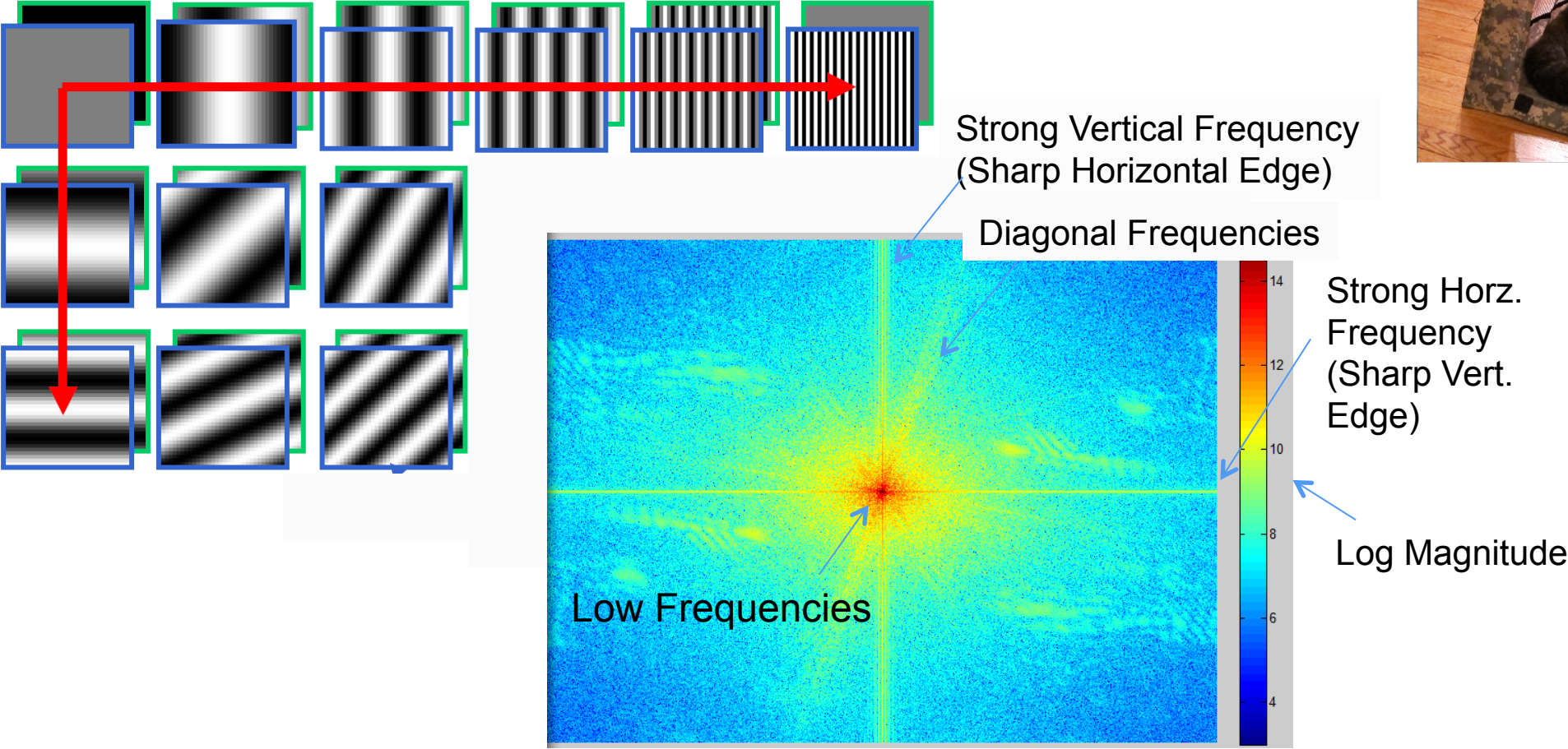
- In 2D case we have two-dimensional frequency (which encodes also the 2D orientation of the sine wave)



Signals can be composed



Fourier Bases



This change of basis is the Fourier Transform

Fourier Transform

- Fourier transform stores the magnitude and phase at each frequency
 - Magnitude encodes how much signal there is at a particular frequency
 - Phase encodes spatial information (indirectly)
 - For mathematical convenience, this is often notated in terms of real and complex numbers

$$\text{Amplitude: } A = \pm \sqrt{R(\omega)^2 + I(\omega)^2} \qquad \text{Phase: } \phi = \tan^{-1} \frac{I(\omega)}{R(\omega)}$$

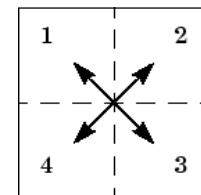
$$\text{Euler's formula: } e^{inx} = \cos(nx) + i \sin(nx)$$

$$\text{DFT} \quad y(u, v) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m, n) e^{-\frac{j2\pi um}{M}} e^{-\frac{j2\pi vn}{N}}$$

$$\text{IDFT} \quad x(m, n) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} y(u, v) e^{\frac{j2\pi um}{M}} e^{\frac{j2\pi vn}{N}}$$

- Discrete, 2-D Fourier & inverse Fourier transforms are implemented in `fft2` and `ifft2`, respectively
- `fftshift`: Move origin (DC component) to image center for display
- Example:

```
>> I = imread('test.png'); % Load grayscale image
>> F = fftshift(fft2(I)); % Shifted transform
>> imshow(log(abs(F)), []); % Show log magnitude
>> imshow(angle(F), []); % Show phase angle
```



The Convolution Theorem

- The Fourier transform of the convolution of two functions is the product of their Fourier transforms

$$F[g * h] = F[g]F[h]$$

- The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms

$$F^{-1}[gh] = F^{-1}[g] * F^{-1}[h]$$

- Convolution in spatial domain is equivalent to multiplication in frequency domain!

Properties of Fourier Transforms

- Linearity $\mathcal{F}[ax(t) + by(t)] = a\mathcal{F}[x(t)] + b\mathcal{F}[y(t)]$
- Fourier transform of a real signal is symmetric about the origin
- The energy of the signal is the same as the energy of its Fourier transform

Questions

- Which has more information, the phase or the magnitude?
- What happens if you take the phase from one image and combine it with the magnitude from another image?

Example: amplitude vs. phase

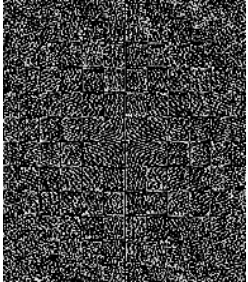
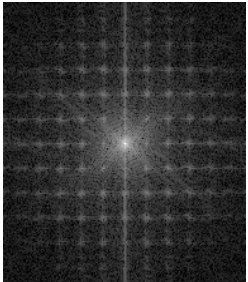
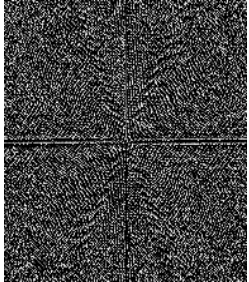
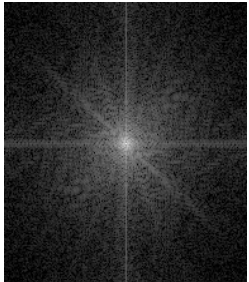
A = "Aron"

FA = fft2(A)

log(abs(FA))

angle(FA)

ifft2(abs(FA), angle(FP))



P = "Phyllis"

FP = fft2(P)

log(abs(FP))

angle(FP)

ifft2(abs(FP), angle(FA))

What this all has to do with filtering?

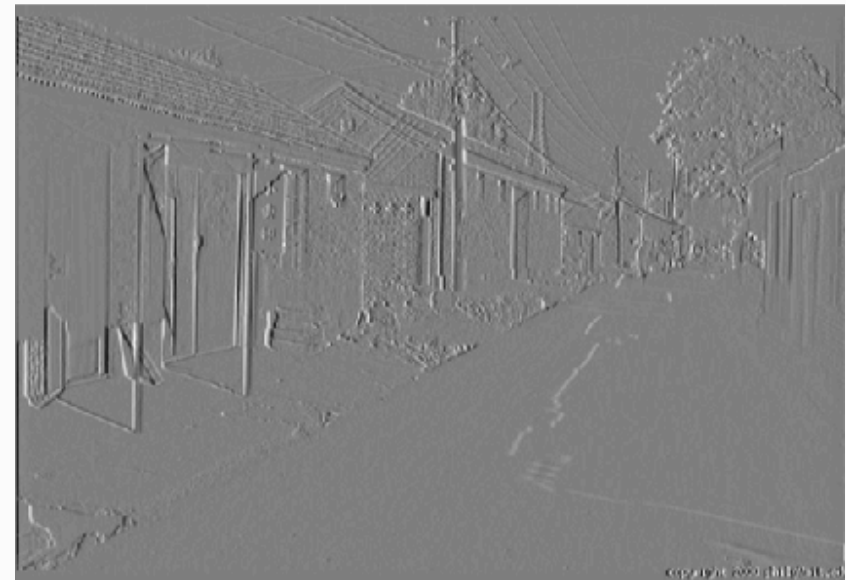
Filtering in spatial domain



*

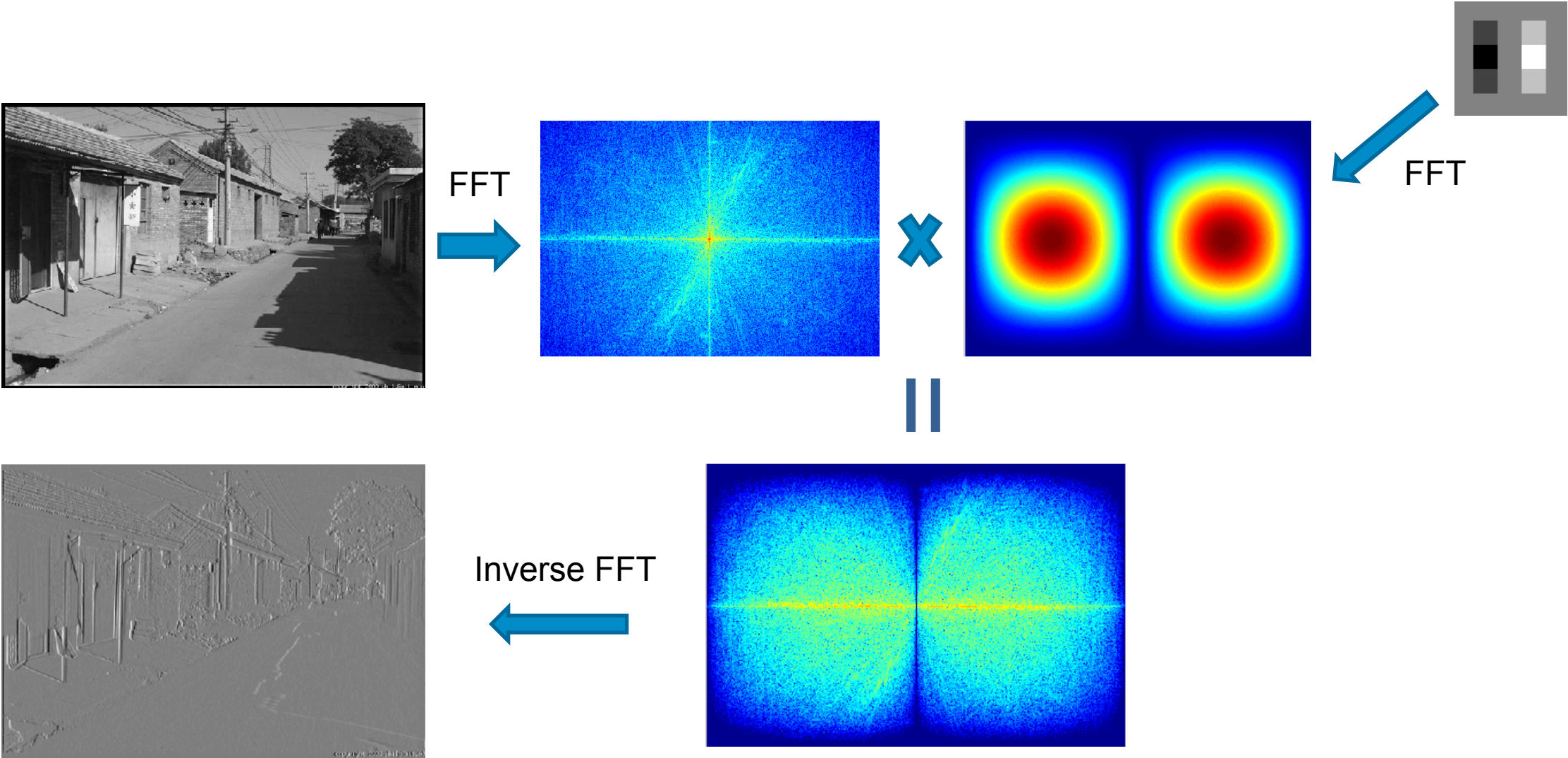


=



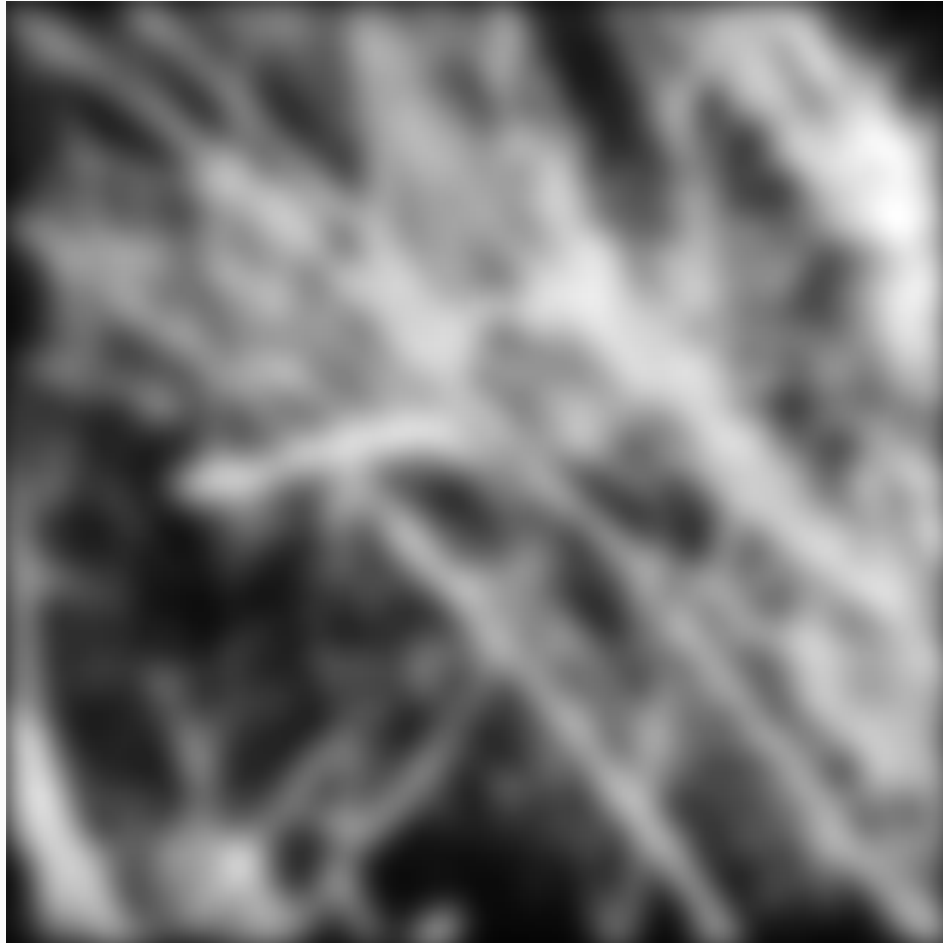
1	0	-1
2	0	-2
1	0	-1

Filtering in frequency domain

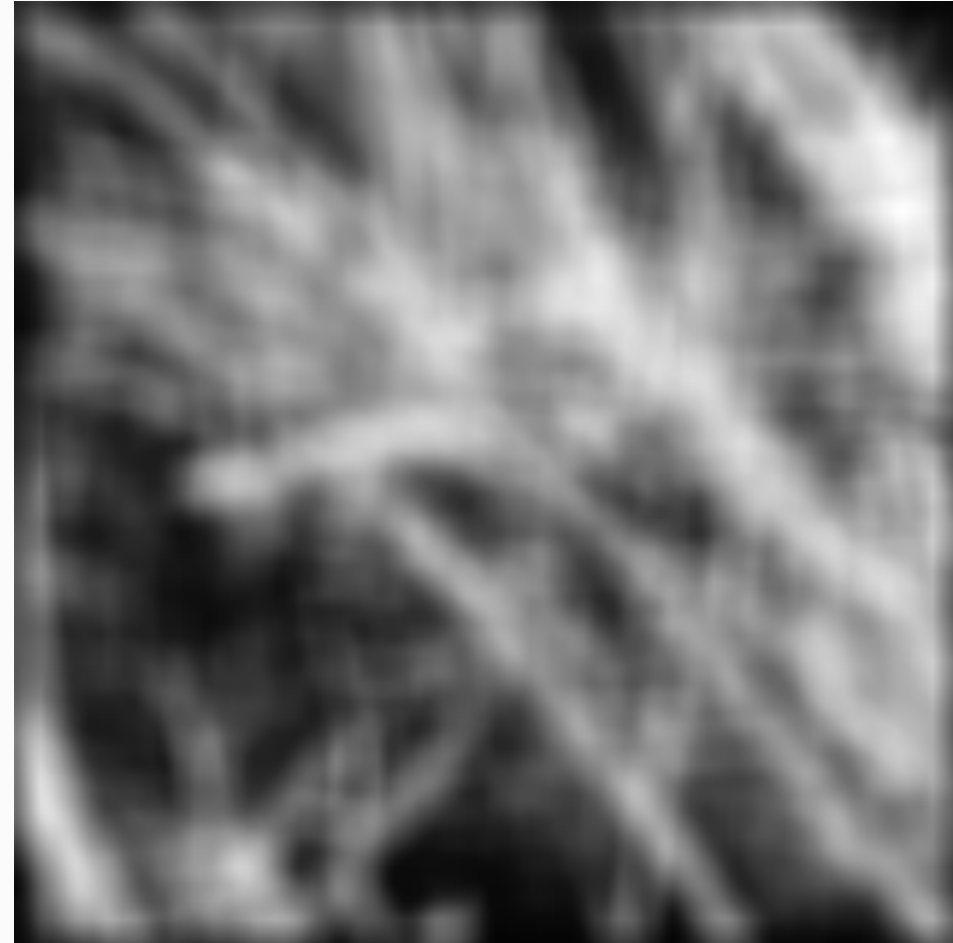


Why Gaussian gives smooth output compared to box filter?

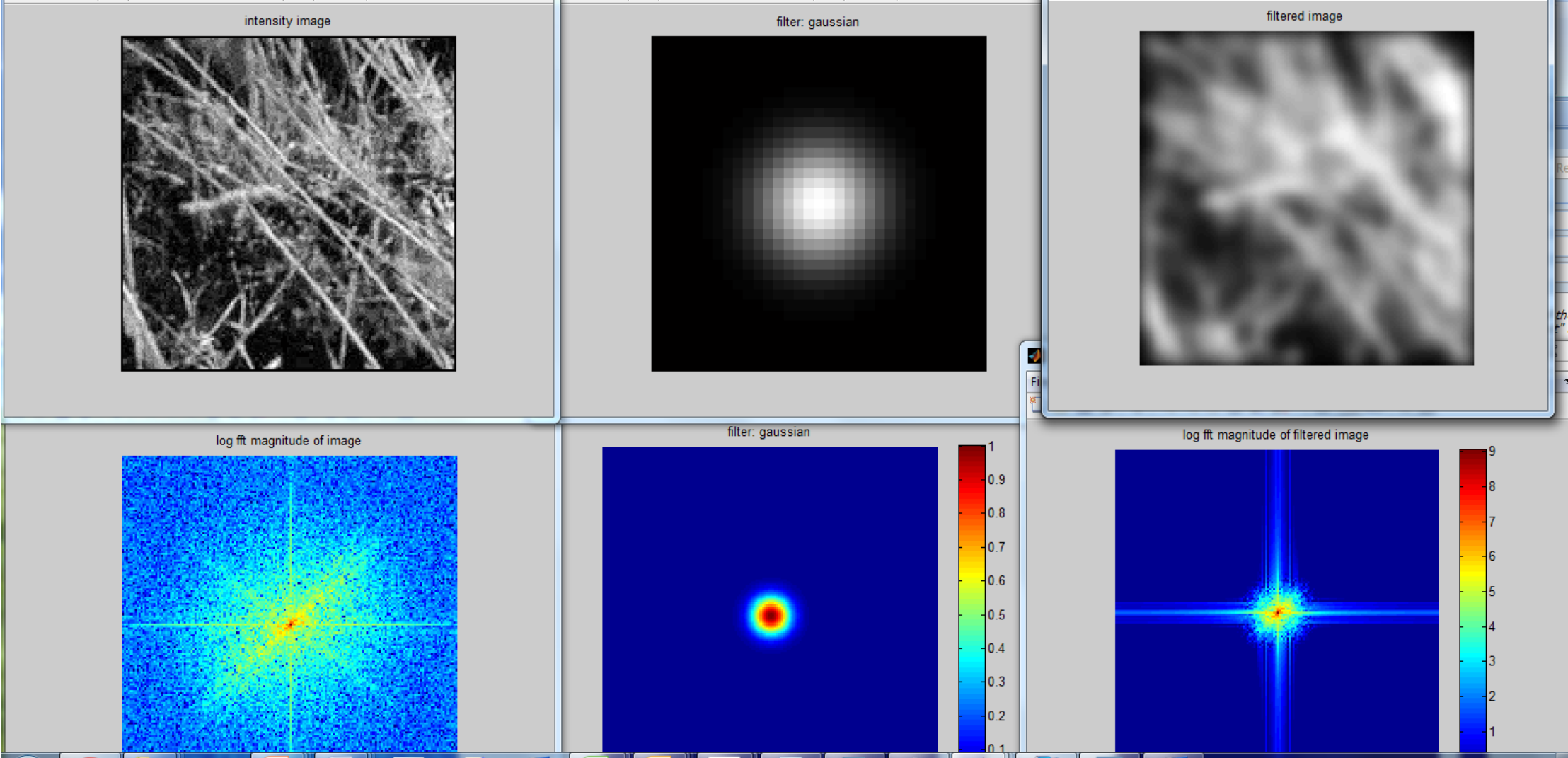
Gaussian



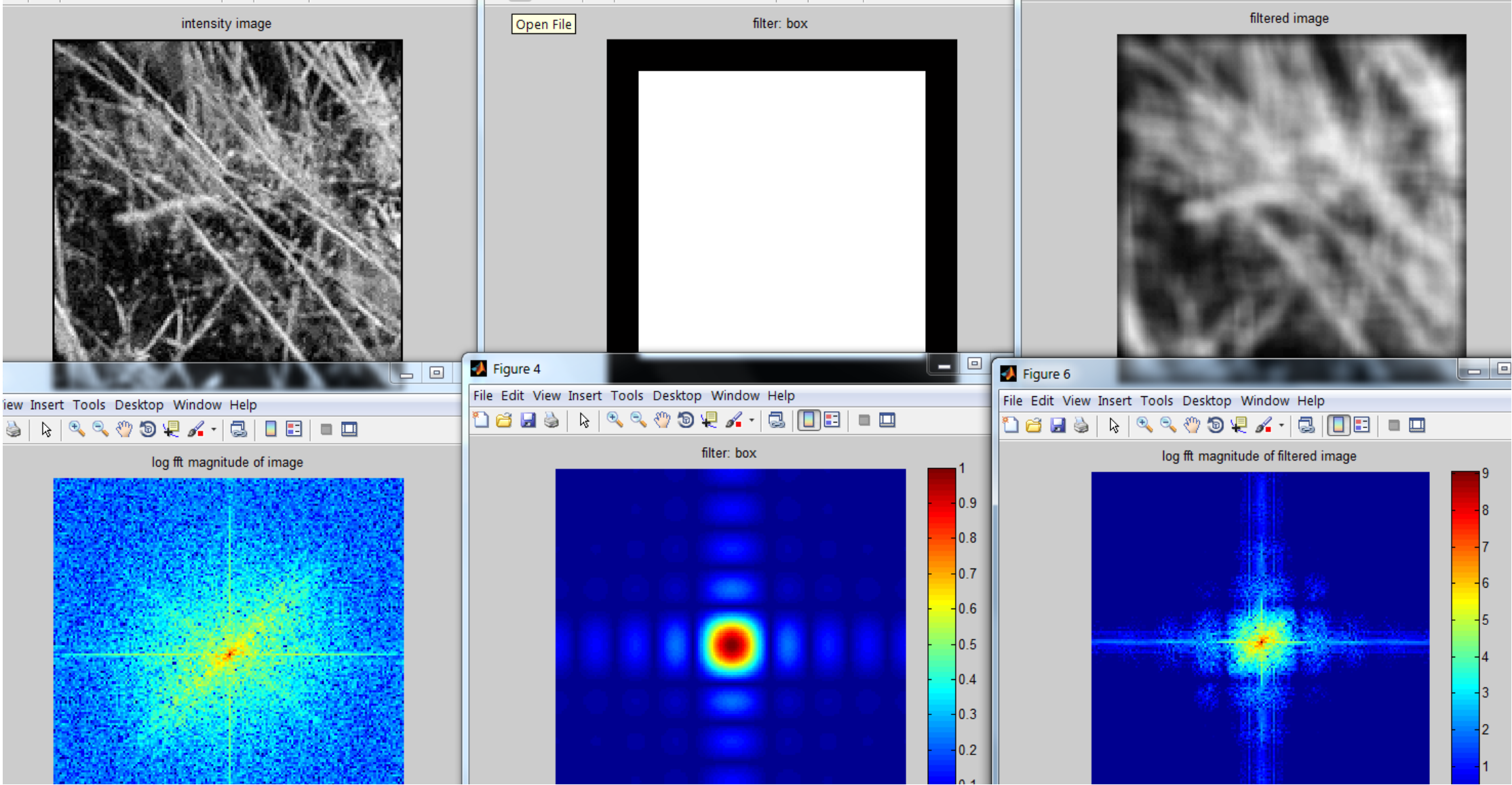
Box filter



Gaussian filter



Box filter



Why lower resolution image still make sense? What is lost?



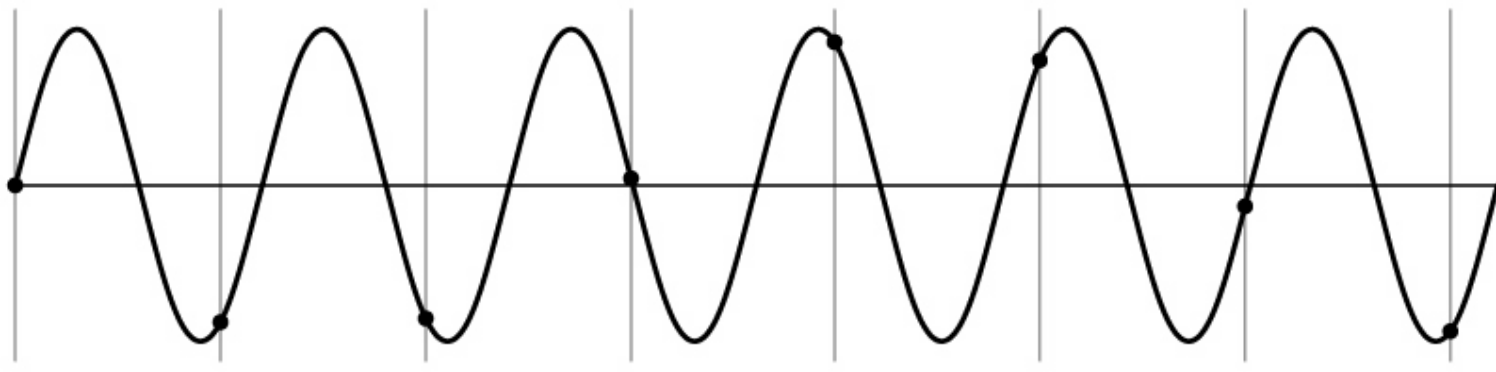
Subsampling by a factor of two



Throw away every other row and column to create a $\frac{1}{2}$ size image

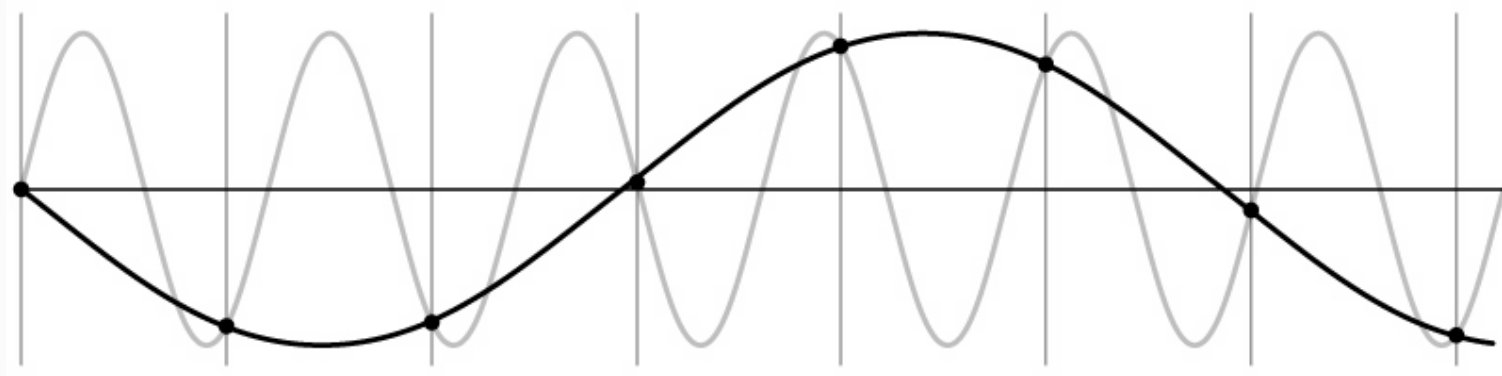
Problem: Aliasing

- One-dimensional example (sinewave):



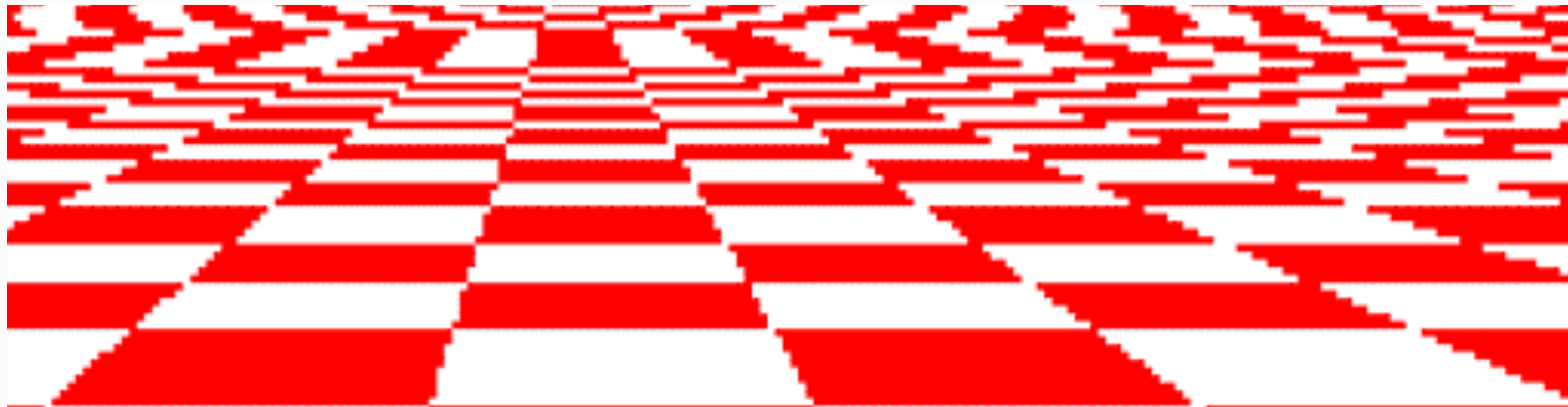
Problem: Aliasing

- One-dimensional example (sinewave):



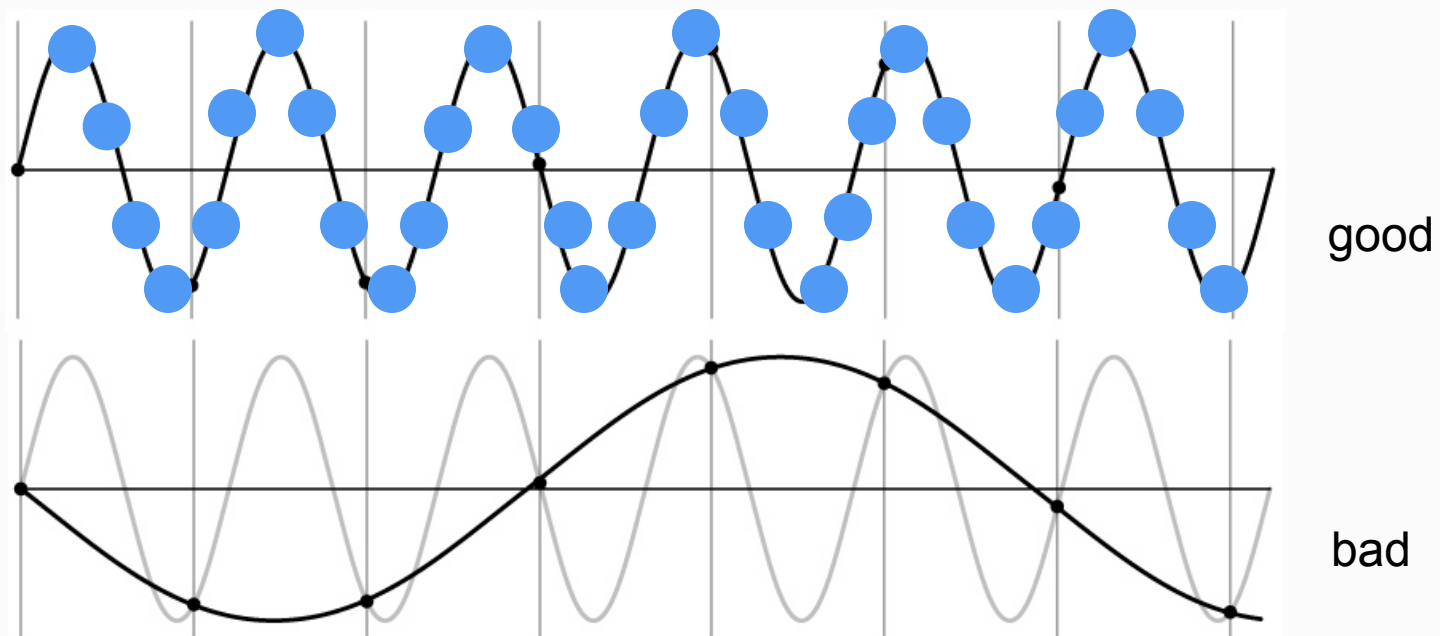
Aliasing in graphics

- Characteristic errors may appear "checker board disintegrate", "striped shirts look funny",....



Nyquist-Shannon sampling theorem

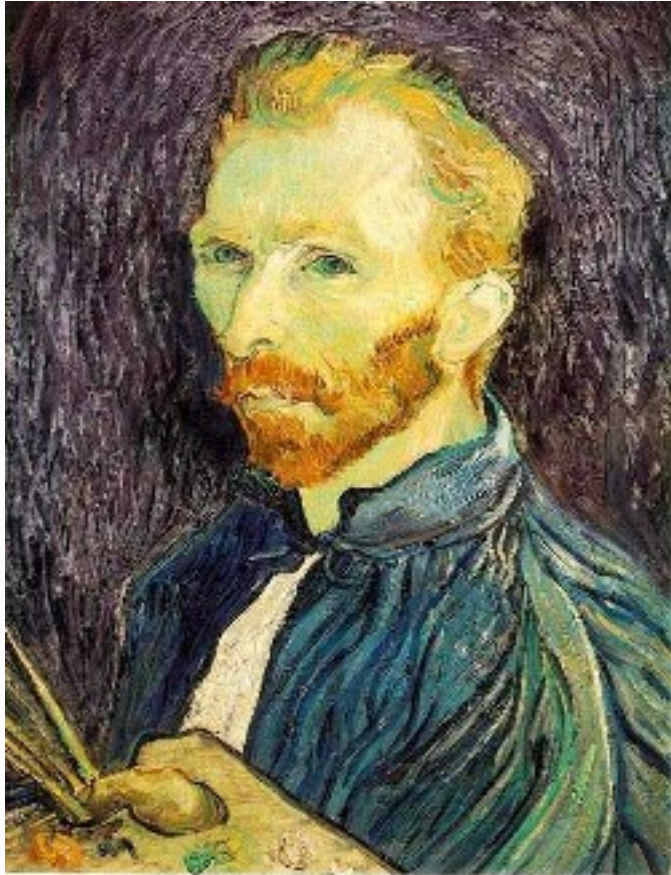
- When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{\max}$
- This allows to reconstruct the original perfectly from the sampled version



Solution: Anti-aliasing

- Option 1: Sample more often
- Option 2: Get rid of frequencies greater than half the new sampling frequency (i.e. filter)
-> Loss of information, but still better than aliasing
- Example algorithm for downsampling by factor 2 (Matlab):
 1. Apply low-pass filter
`im_blur = imfilter(image,fspecial('gaussian',7,1));`
 2. Sample every other pixel
`im_small = im_blur(1:2:end , 1:2:end);`

Subsampling without pre-filtering



1/2

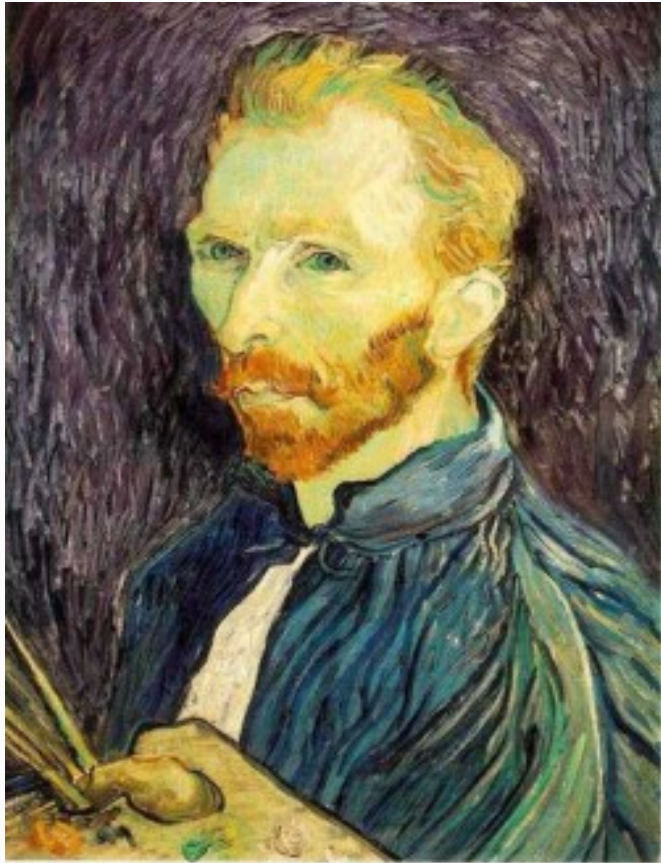


1/4 (2x zoom)



1/8 (4x zoom)

Subsampling with pre-filtering



Gaussian $1/2$



G $1/4$

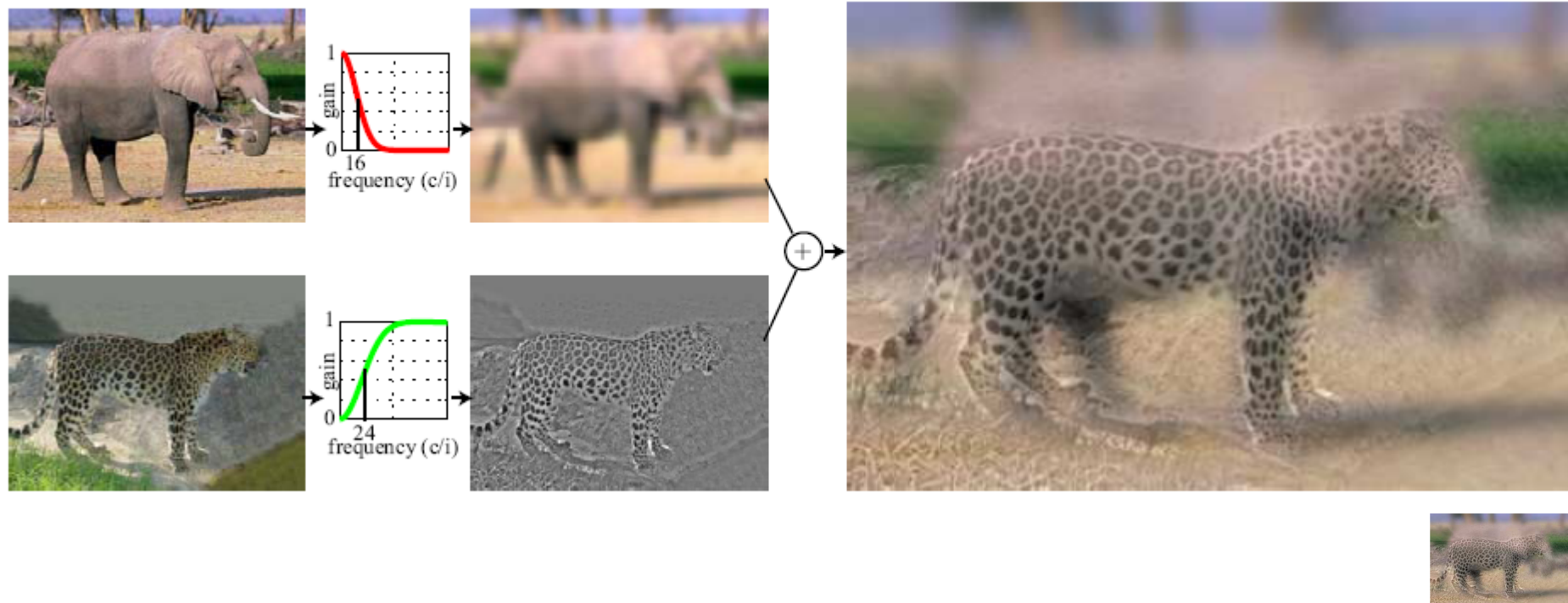


G $1/8$

Why lower resolution image still make sense? What is lost?



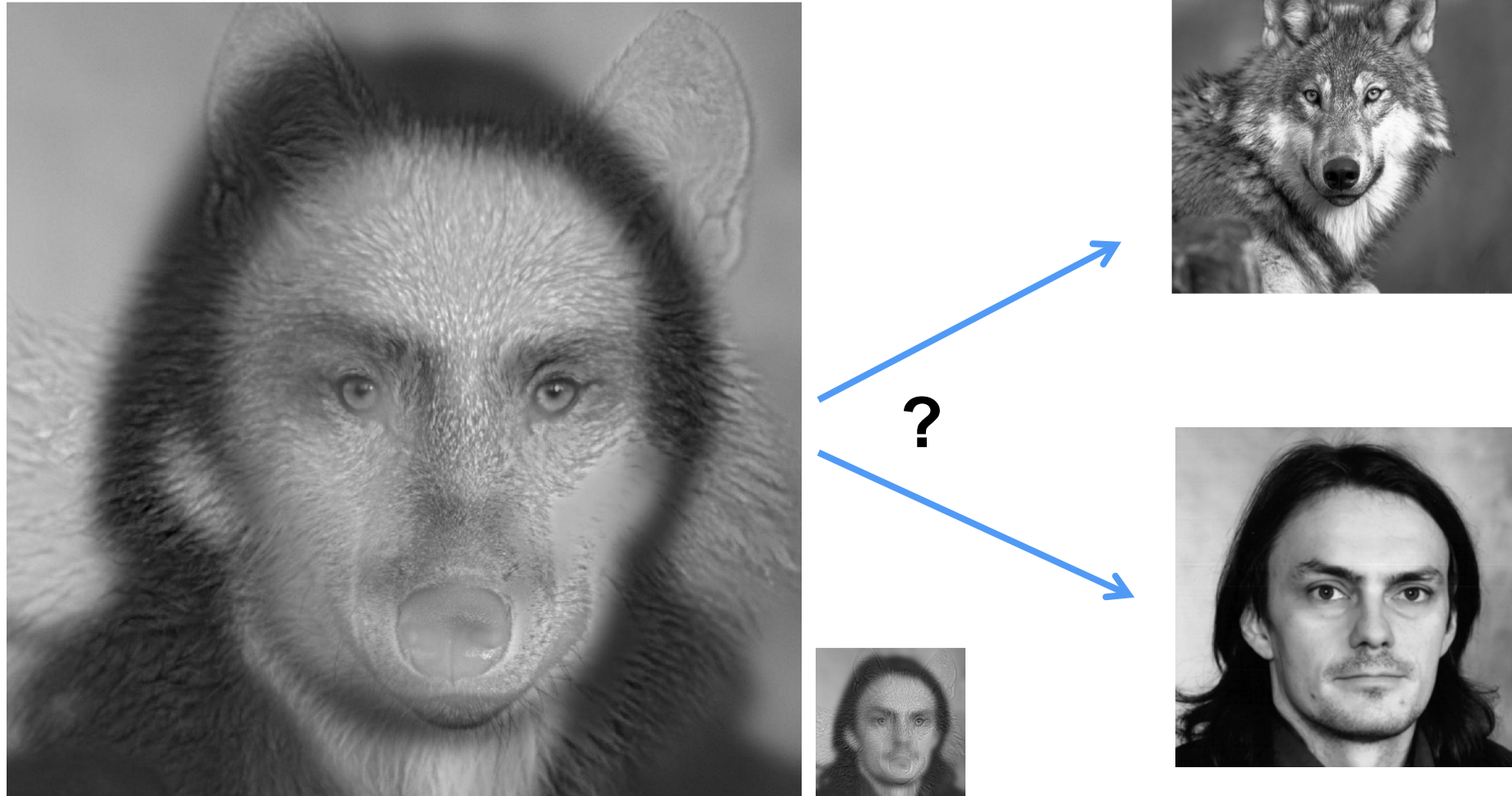
Hybrid Images



A. Oliva, A. Torralba, P.G. Schyns, ["Hybrid Images,"](#) SIGGRAPH 2006

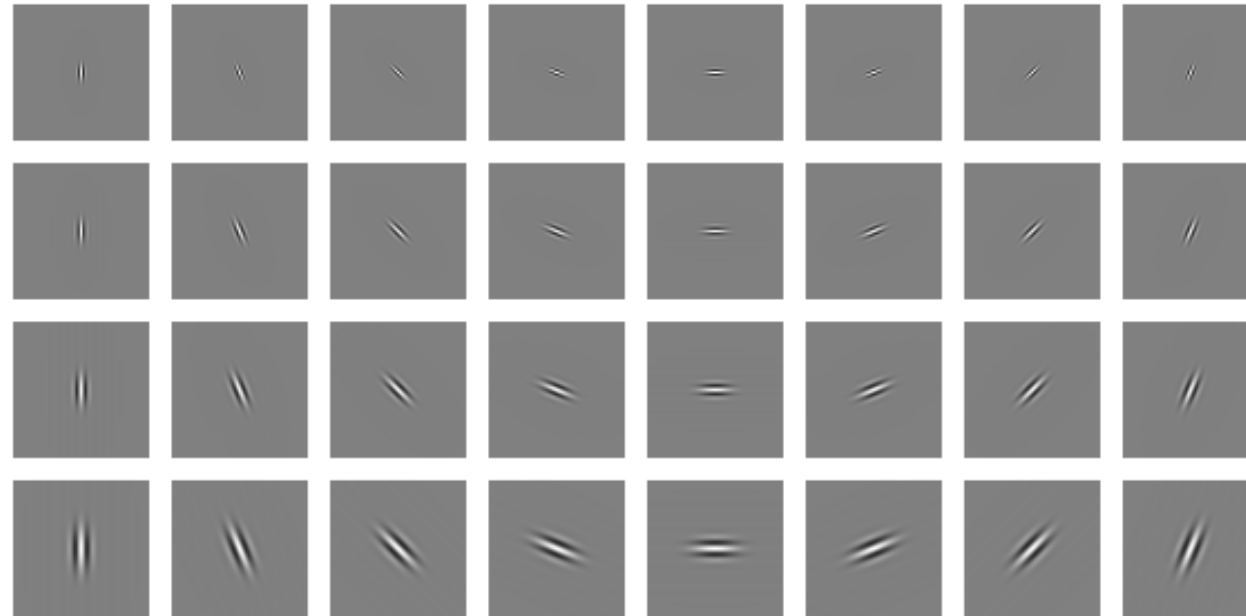
Source: D. Hoiem

Why do we get distance-dependent interpretation of a hybrid image?



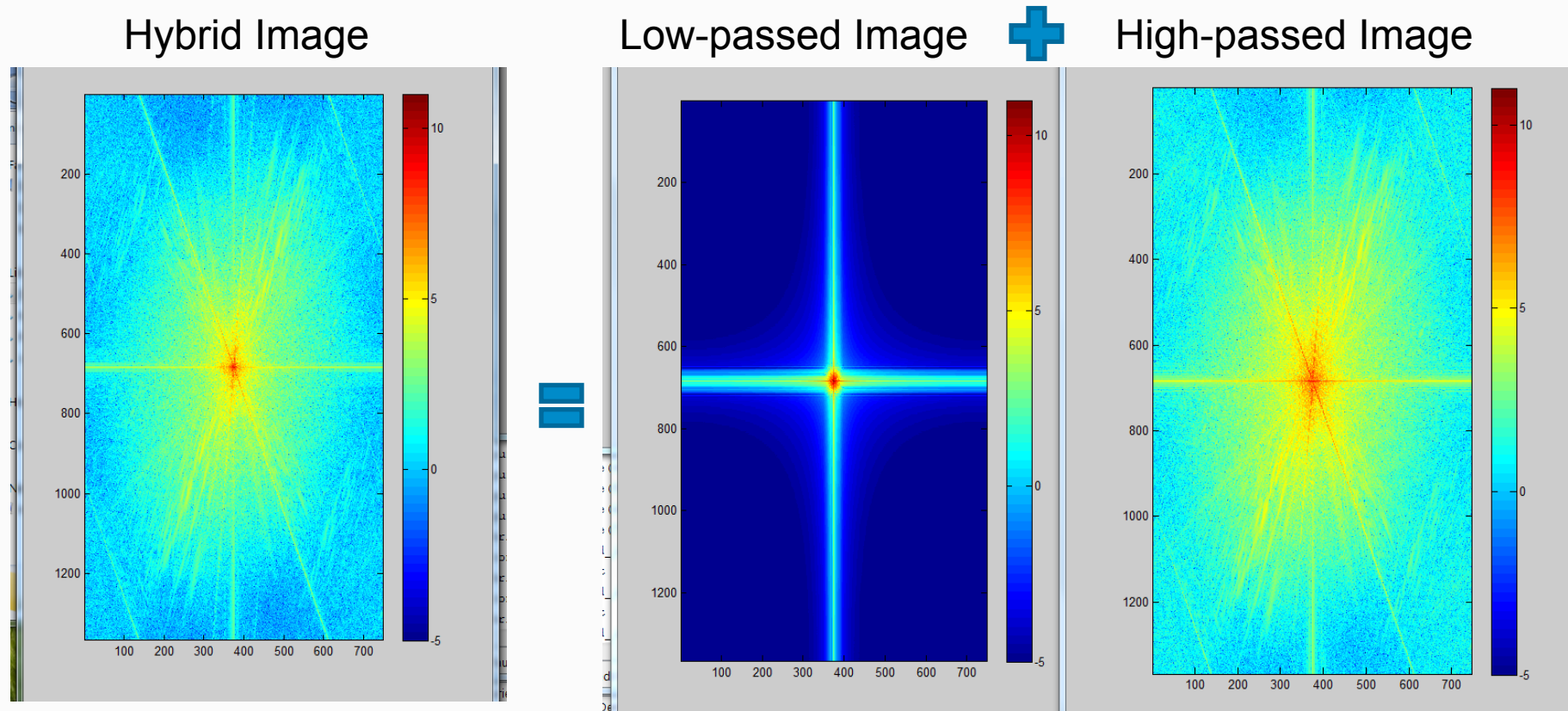
Clues from Human Perception

- Early processing in humans filters for various orientations and scales of frequency
- Perceptual cues in the mid-high frequencies dominate perception
- When we see an image from far away, we are effectively subsampling it (and low pass filtering)

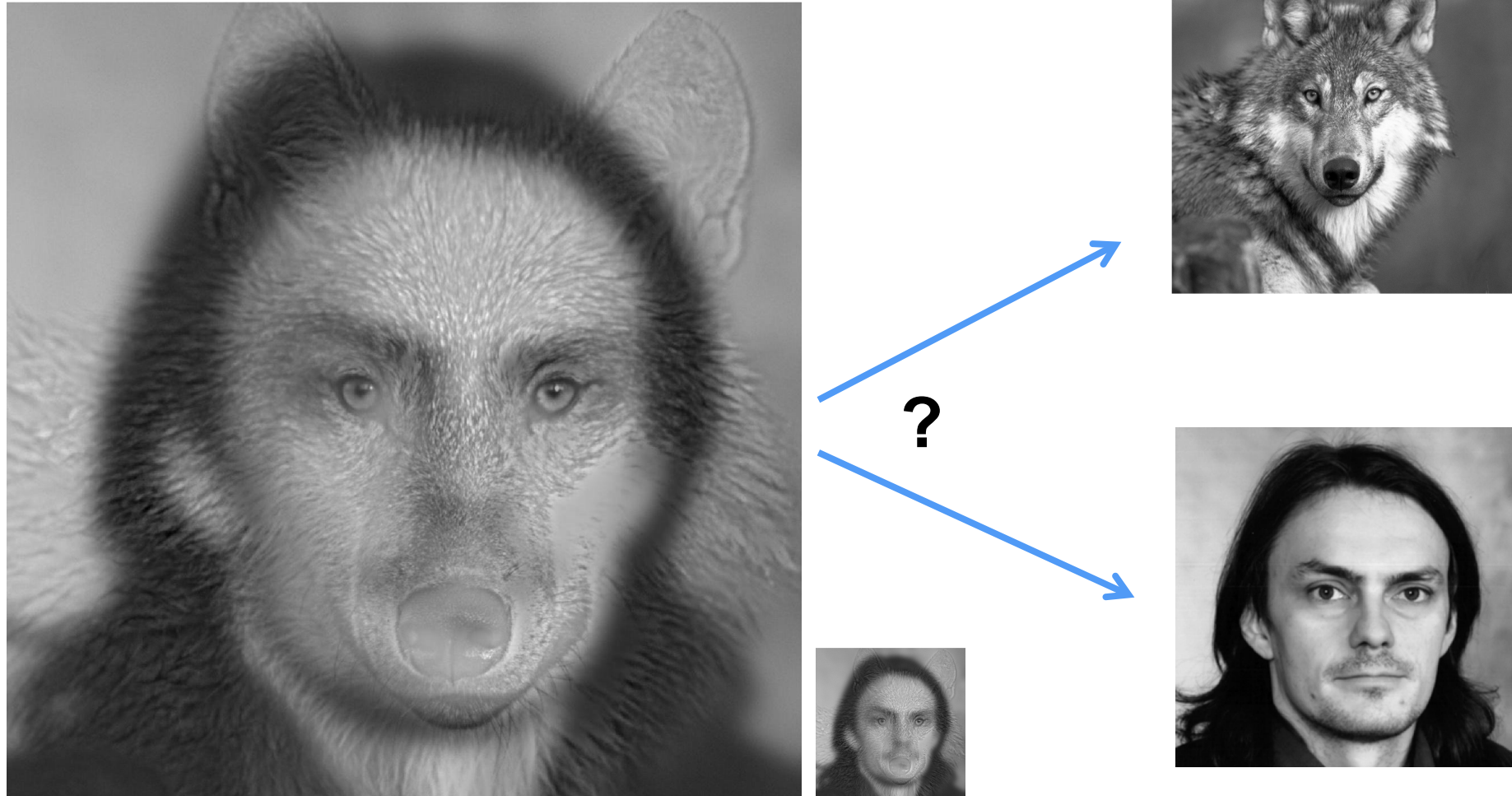


Early Visual Processing: Multi-scale edge and blob filters

Hybrid Image in FFT




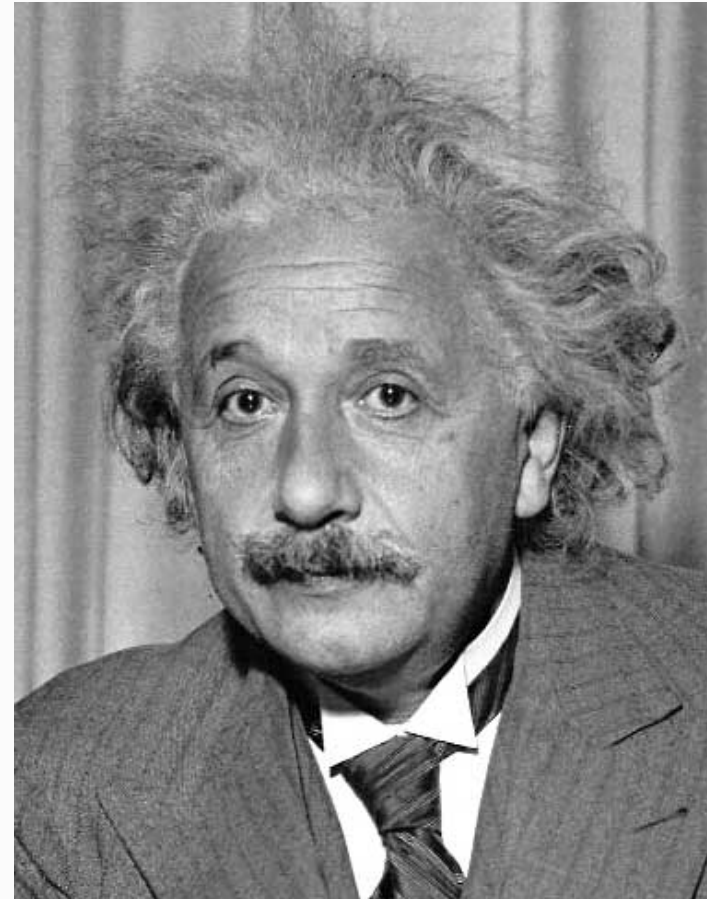
Thus, we get distance-dependent interpretation of a hybrid image




Template matching using filtering

Template matching

- Goal: find  in image
- Approach: Filter image using the template
- What is a good filter function (i.e. similarity measure) between two patches?

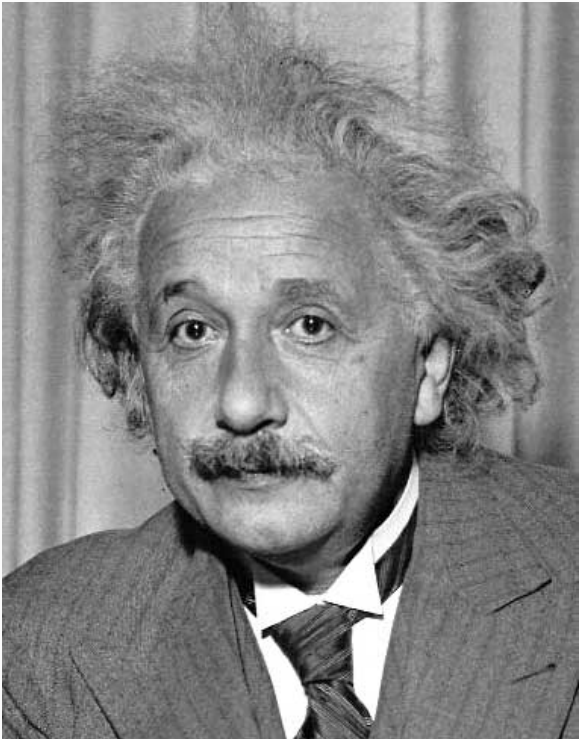


Matching with filters

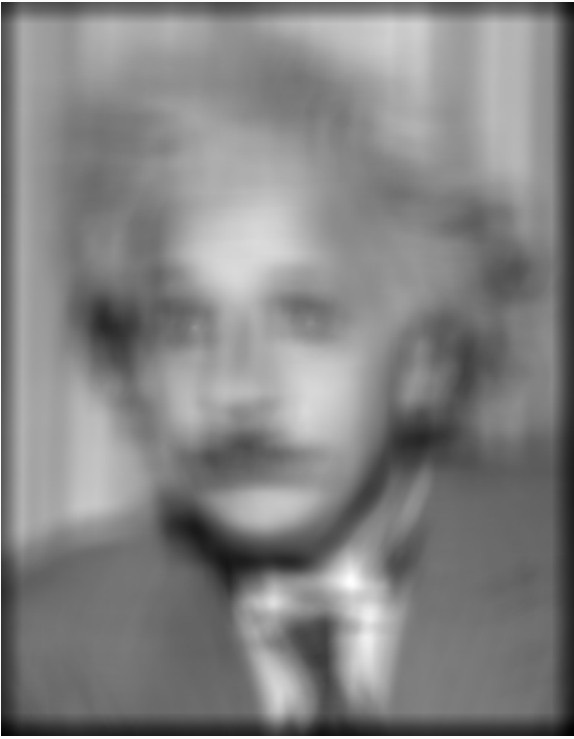
- Goal: find  in image
- Method 1: filter the image with eye patch

$$h[m,n] = \sum_{k,l} g[k,l] f[m+k,n+l]$$

f = image
g = filter




Input



Filtered Image

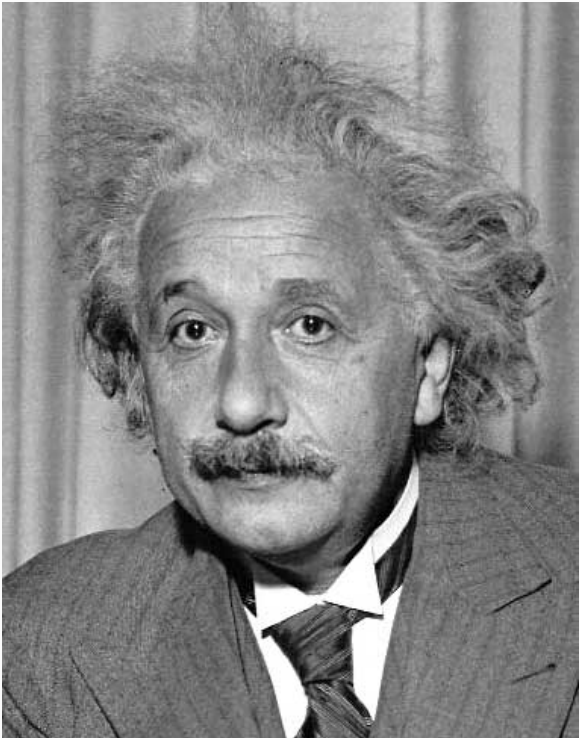
What went wrong?

Matching with filters

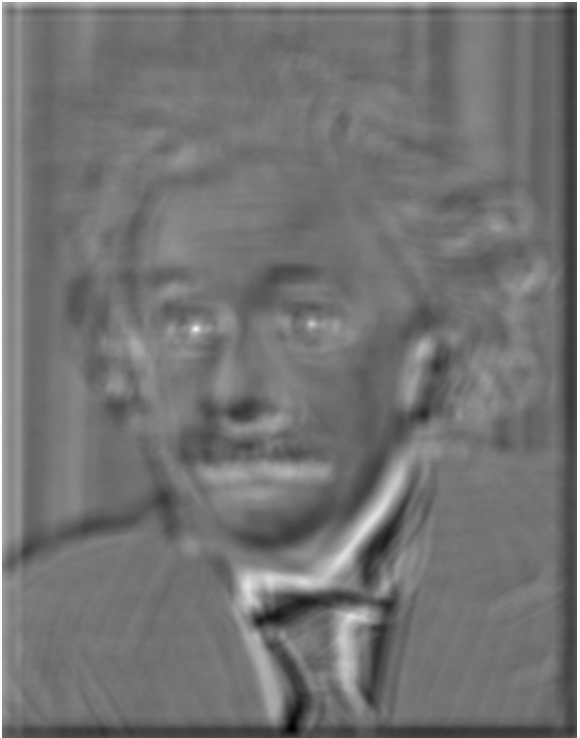
- Goal: find  in image
- Method 2: filter with zero-mean eye

$$h[m,n] = \sum_{k,l} (g[k,l] - \bar{g}) (f[m+k,n+l])$$

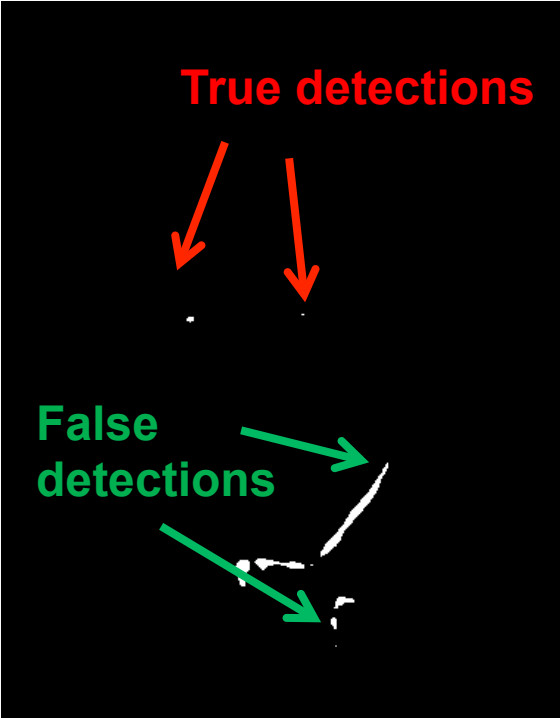
mean of template g



Input




Filtered Image (scaled)



Thresholded Image

Matching with filters

- Goal: find  in image
- Method 3: Normalized cross-correlation


$$h[m,n] = \frac{\sum_{k,l} (g[k,l] - \bar{g})(f[m+k,n+l] - \bar{f}_{m,n})}{\left(\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m+k,n+l] - \bar{f}_{m,n})^2 \right)^{0.5}}$$

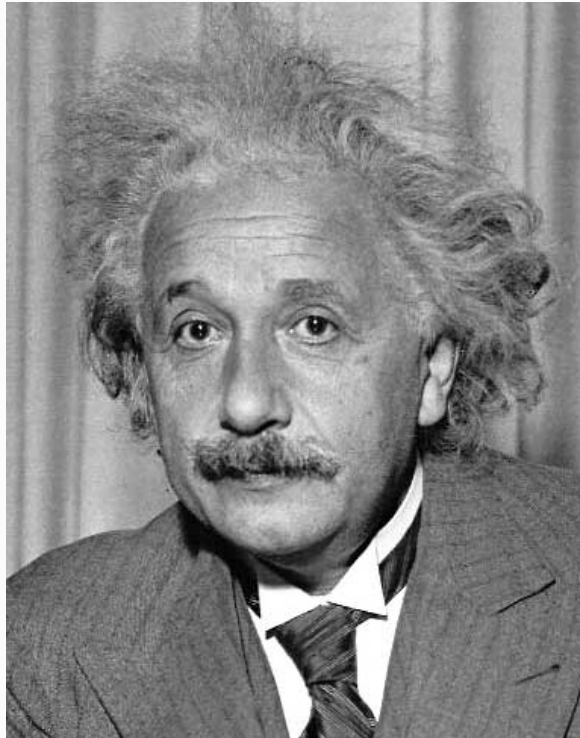
mean template mean image patch

↓ ↓

Matlab: `normxcorr2(template, im)`

Matching with filters

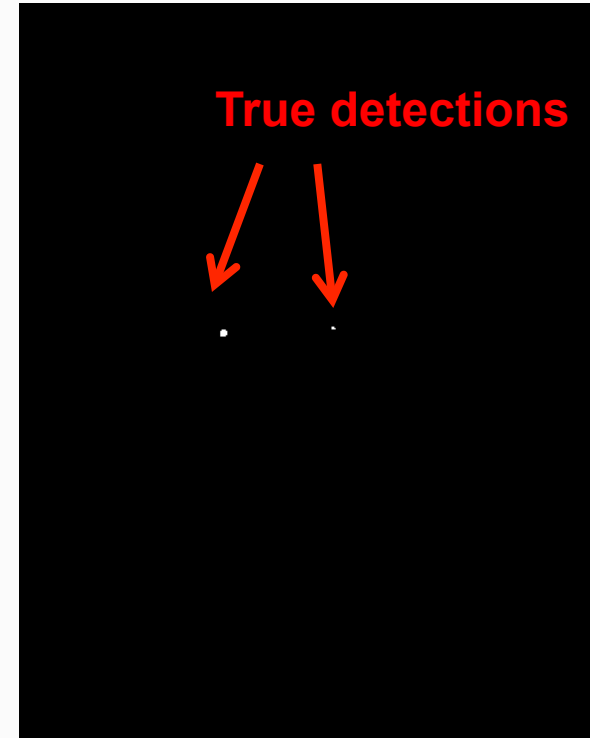
- Goal: find  in image
- Method 3: Normalized cross-correlation



Input




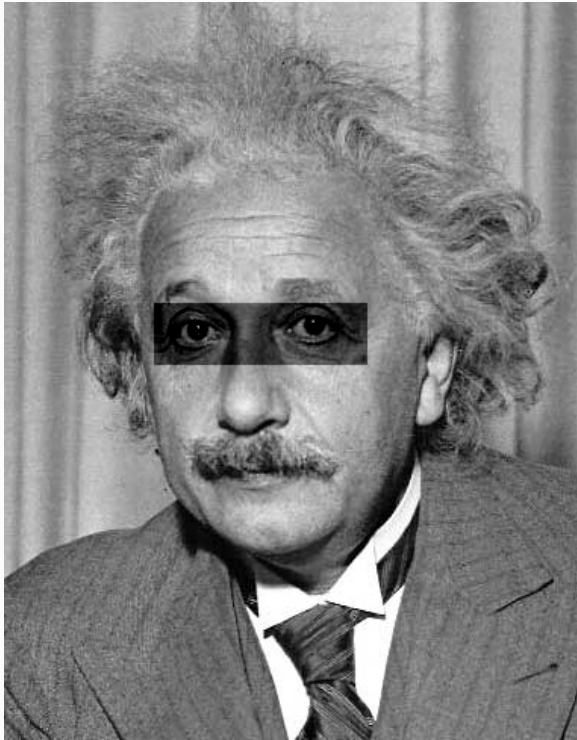
Normalized X-Correlation



Thresholded Image

Matching with filters

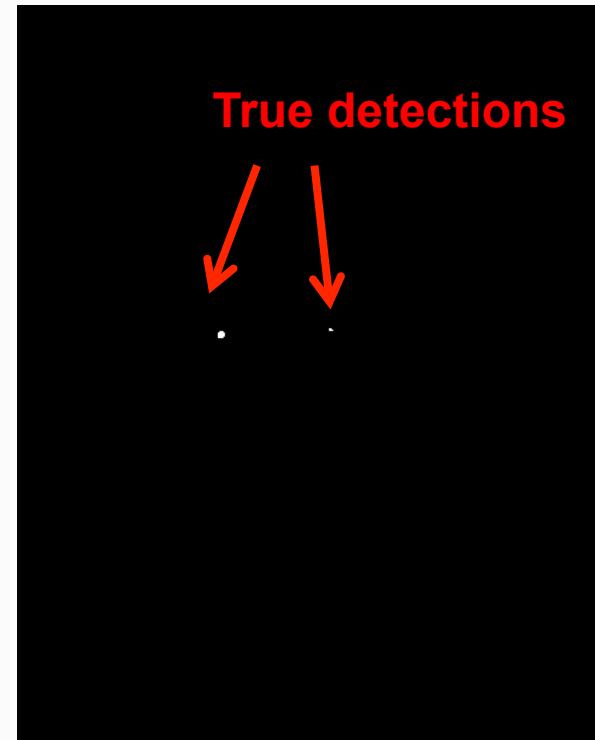
- Goal: find  in image
- Method 3: Normalized cross-correlation



Input



Normalized X-Correlation



Thresholded Image

Q: What is the best method to use?

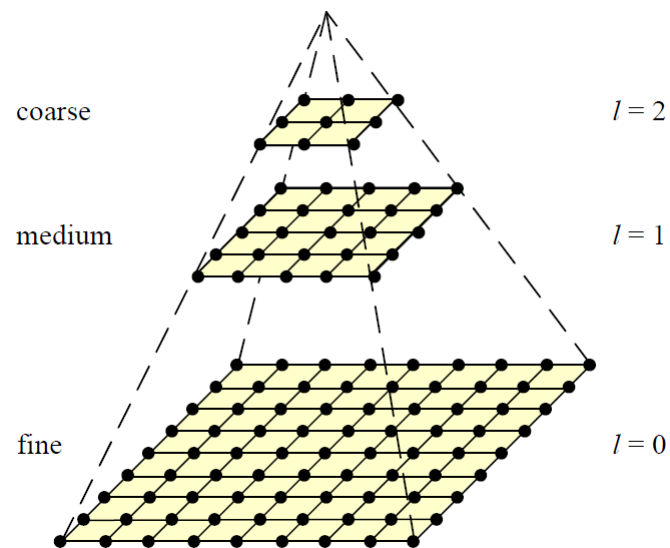
A: Depends

- Zero-mean filter: fast but not a great matcher
- Normalized cross-correlation: slow but invariant to local average intensity and contrast

Q: What if we want to find larger or smaller eyes?

A: Image pyramids: multiresolution image representations

- Repeated decimation with a Gaussian low-pass filter gives Gaussian pyramid



GAUSSIAN PYRAMID



0

1

2

3

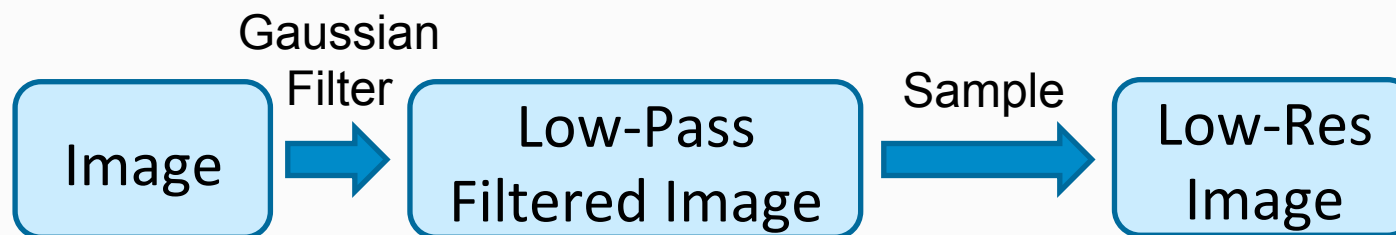
4

5

Template Matching with Image Pyramids

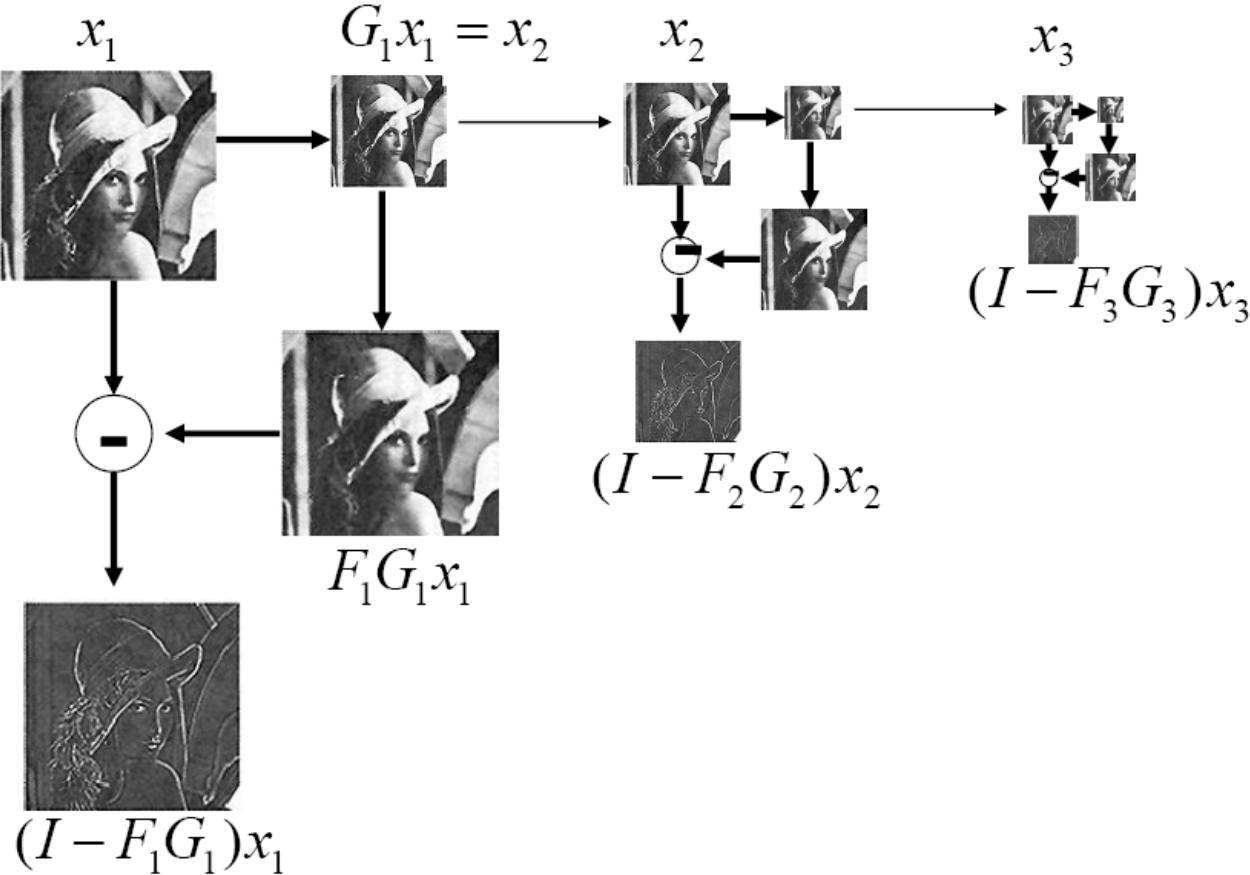
Input: Image, Template

1. Match template at current scale
2. Downsample image
 - In practice, scale step of 1.1 to 1.2
3. Repeat 1-2 until image is very small
4. Take responses above some threshold, perhaps with non-maxima suppression

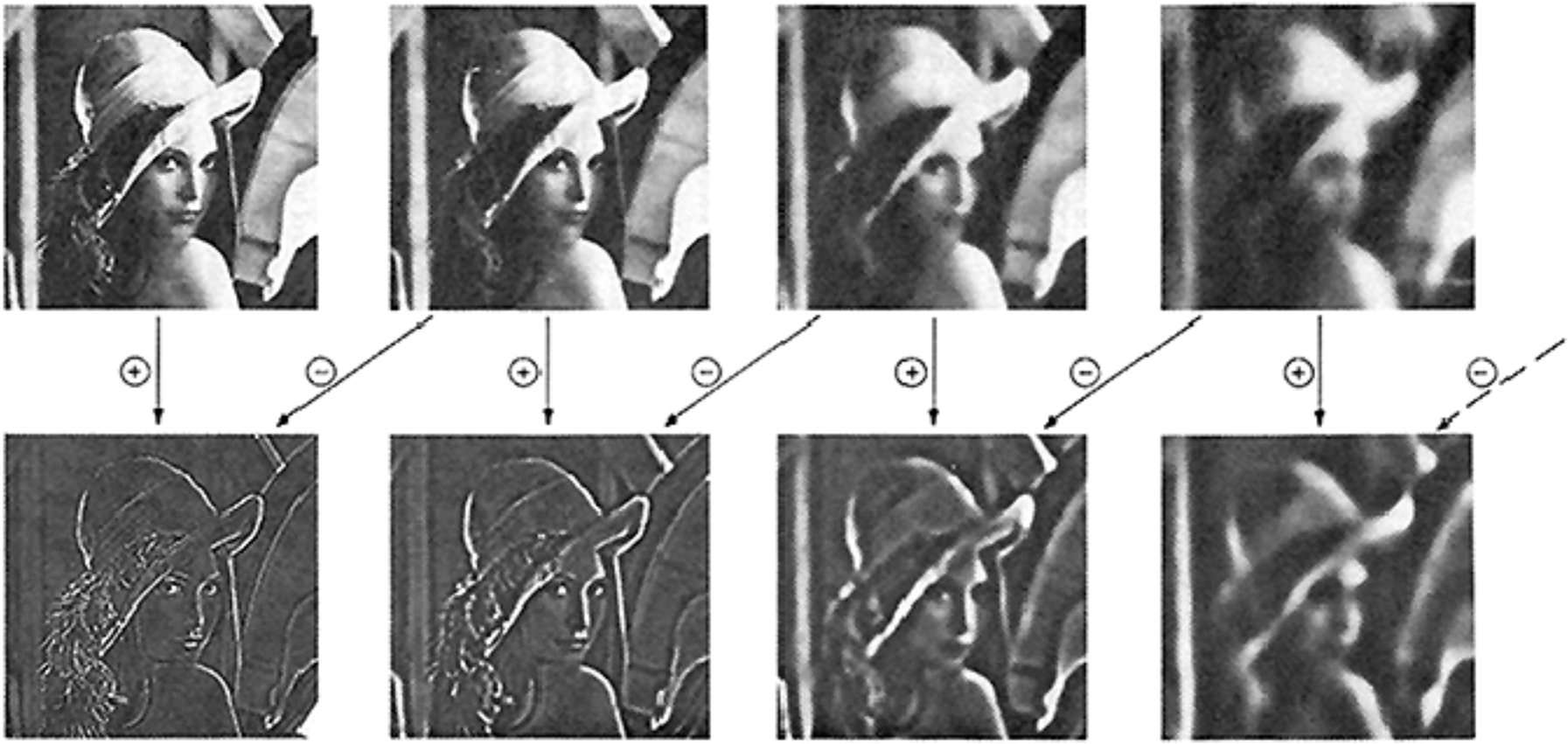


Laplacian pyramid

- Contains the difference images between two successive Gaussian pyramid levels:



Showing, at full resolution, the information captured at each level of a Gaussian (top) and Laplacian (bottom) pyramid.



Major uses of image pyramids

- Compression
- Object detection
 - Scale search
 - Features
- Detecting stable interest points
- Registration
 - Coarse-to-fine

Things to Remember

- Image filtering: compute function of local neighborhood at each position
- Sometimes it makes sense to think of images and filtering in the frequency domain
- Can be faster to filter using FFT for large images ($N \log N$ vs. N^2 for auto-correlation)
- Template matching: localize given template in image
- Image pyramid: multiresolution representation of image (Remember to low pass filter before sub-sampling)

