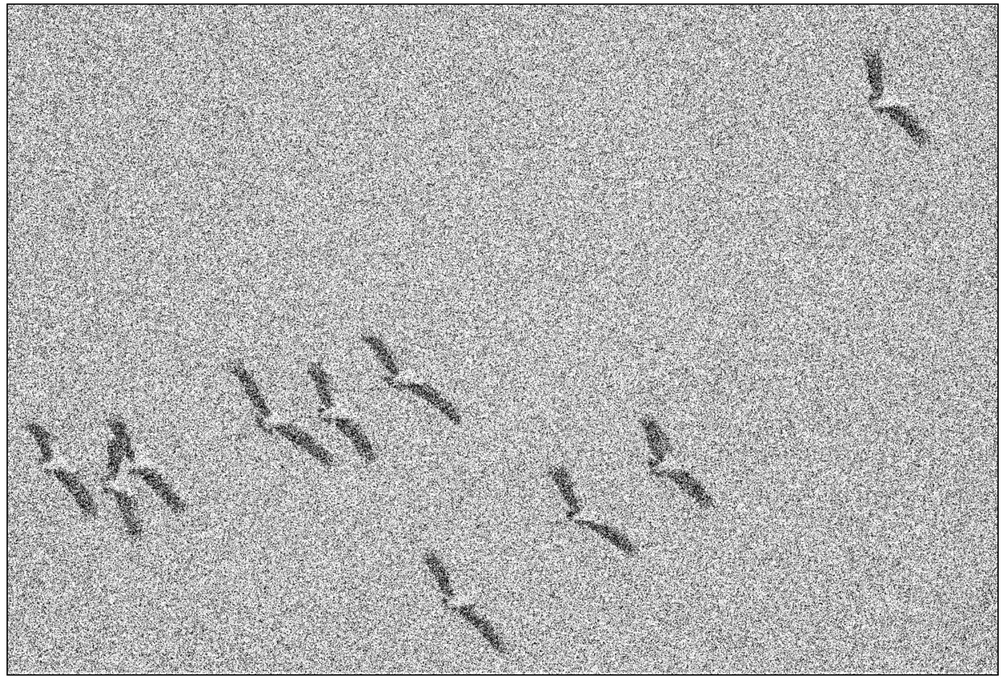
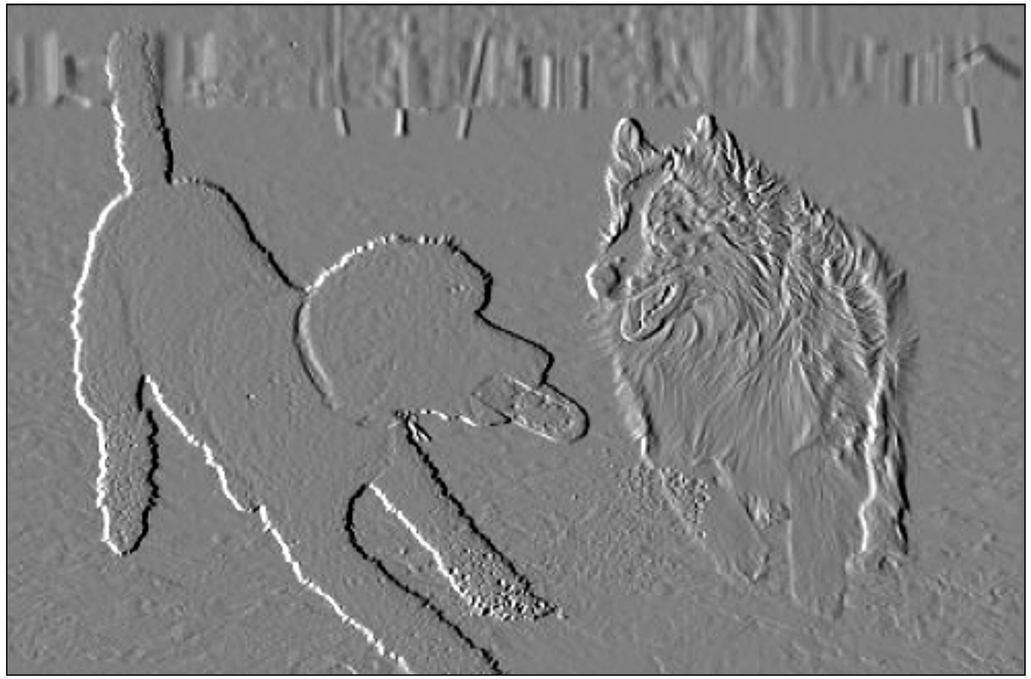
Lecture 2: Filtering

Recall: blurring and edge detection

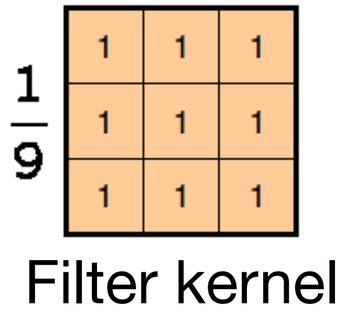


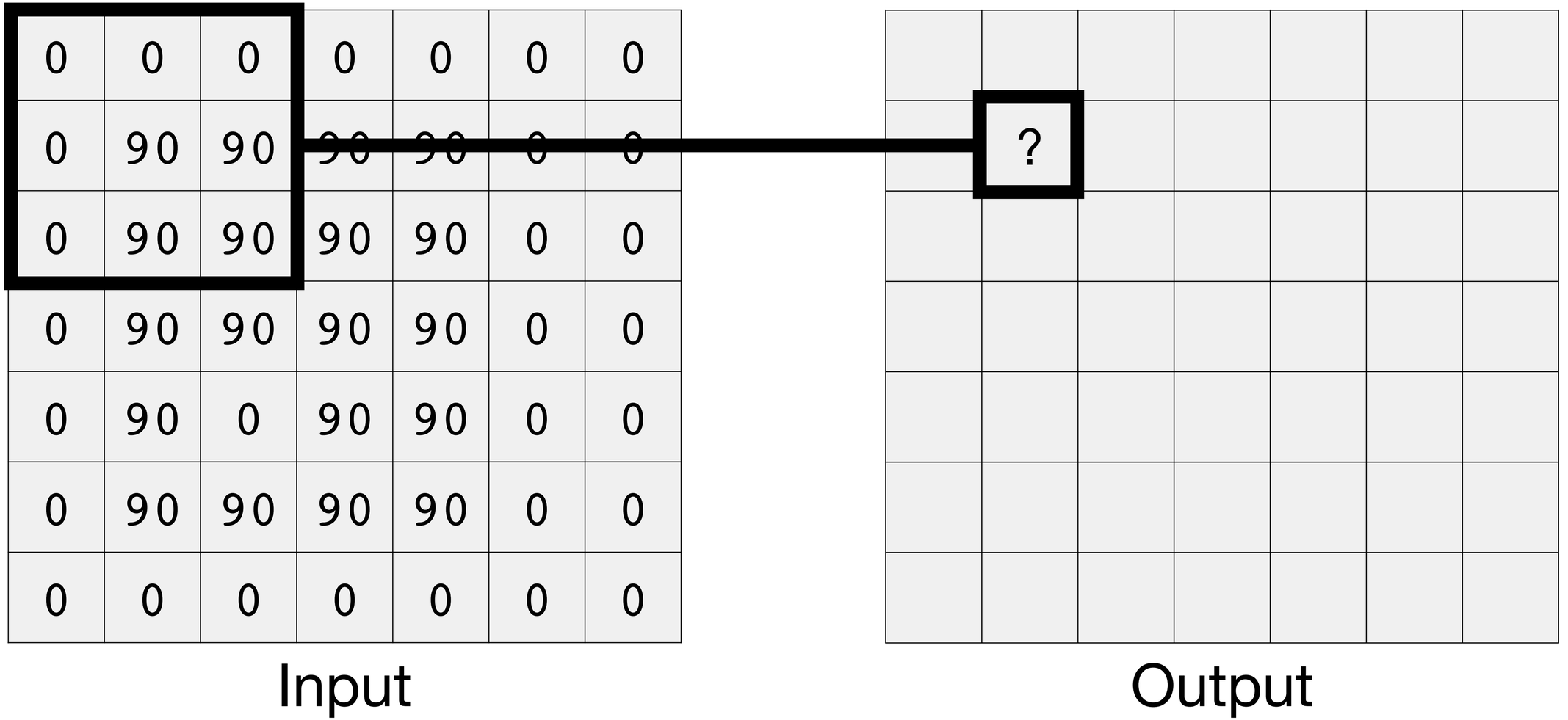
Denoising



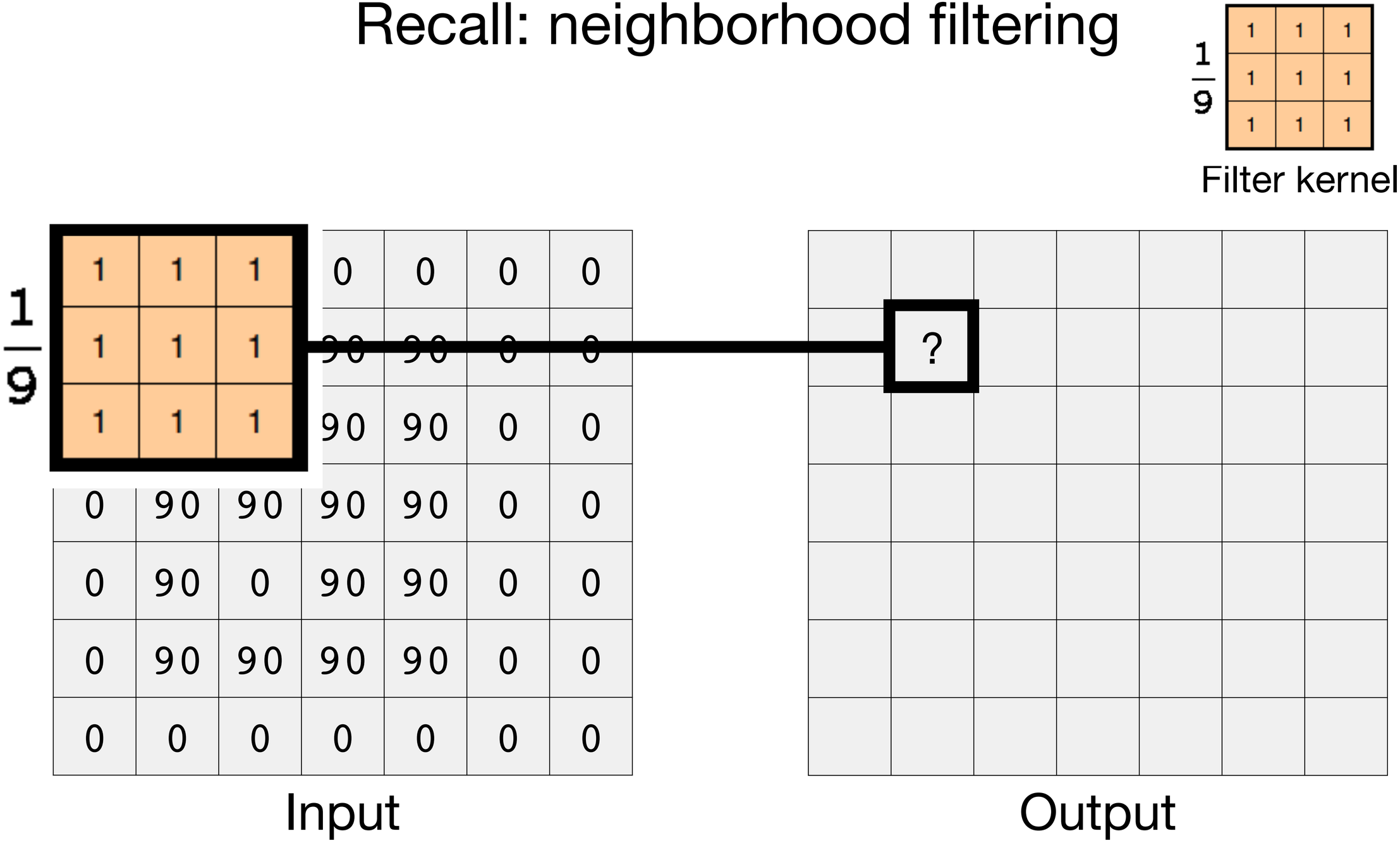
Edge detection

Recall: neighborhood filtering

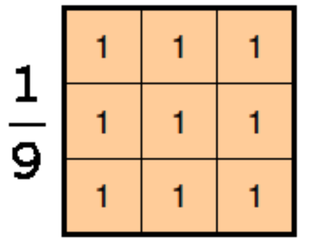




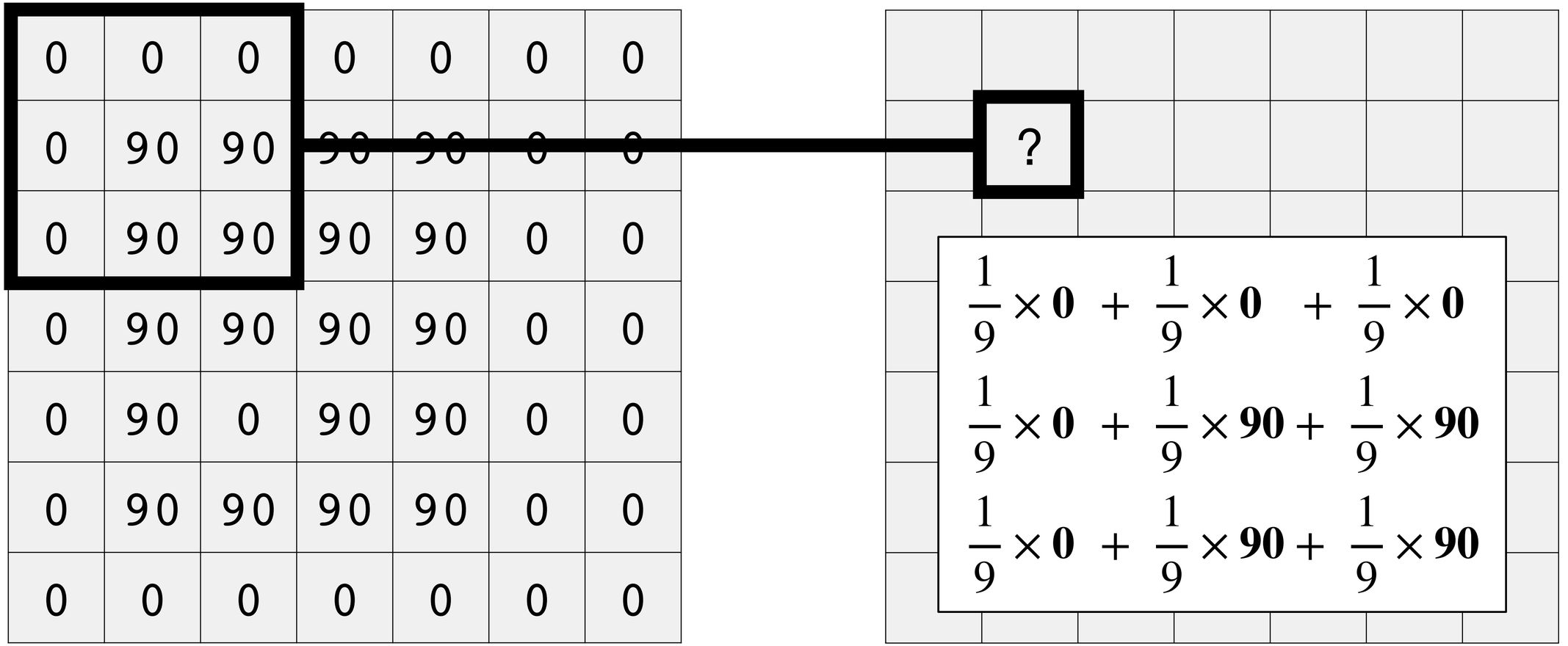
Recall: neighborhood filtering



Recall: neighborhood filtering



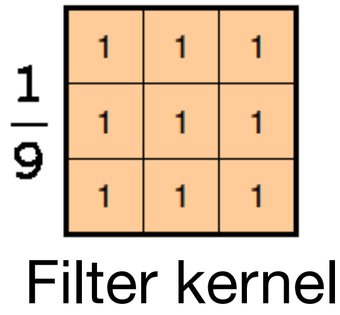
Filter kernel



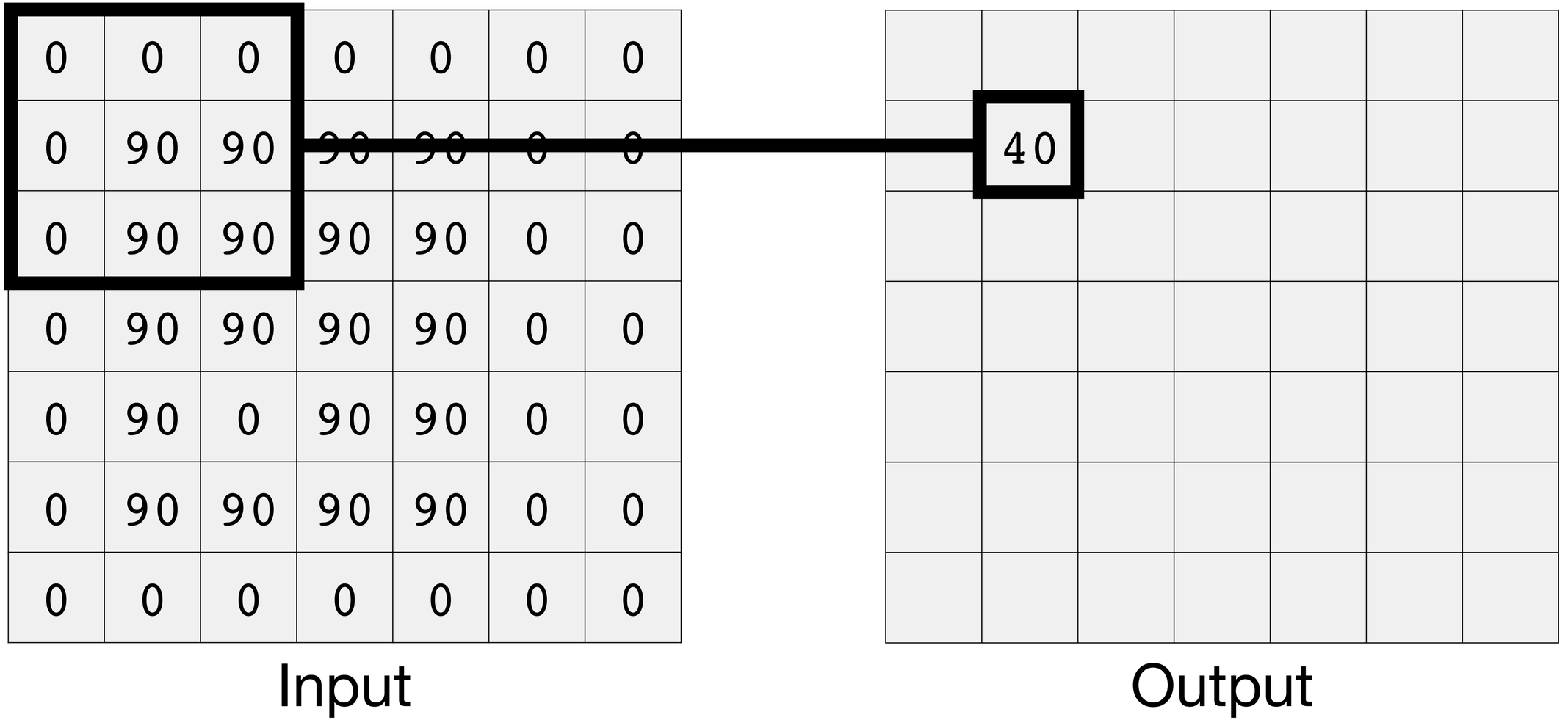
Input

Output

Recall: neighborhood filtering



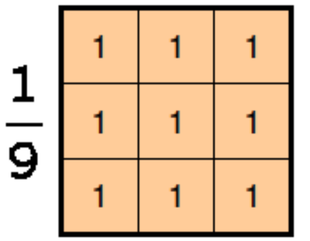
Filter kernel



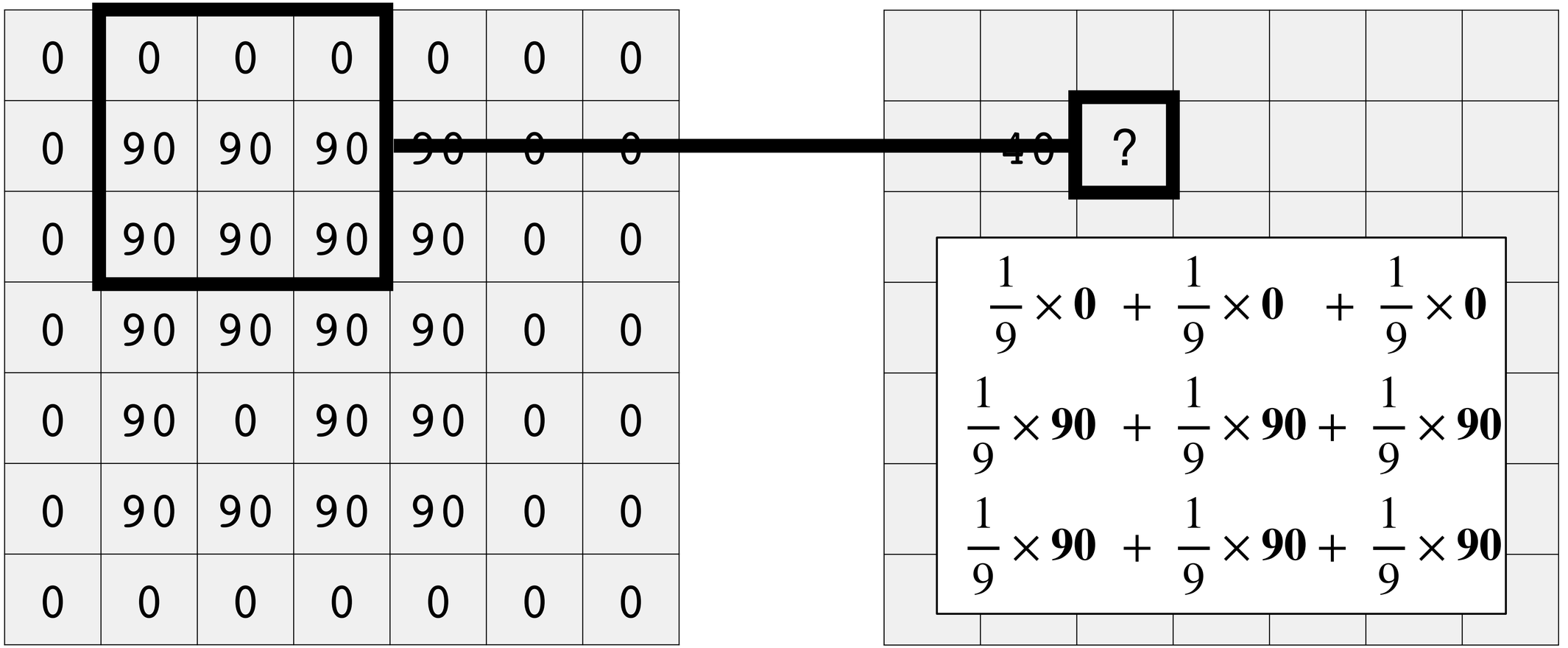
Input

Output

Recall: neighborhood filtering



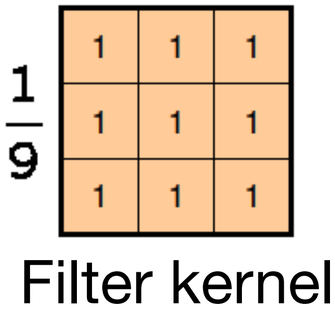
Filter kernel

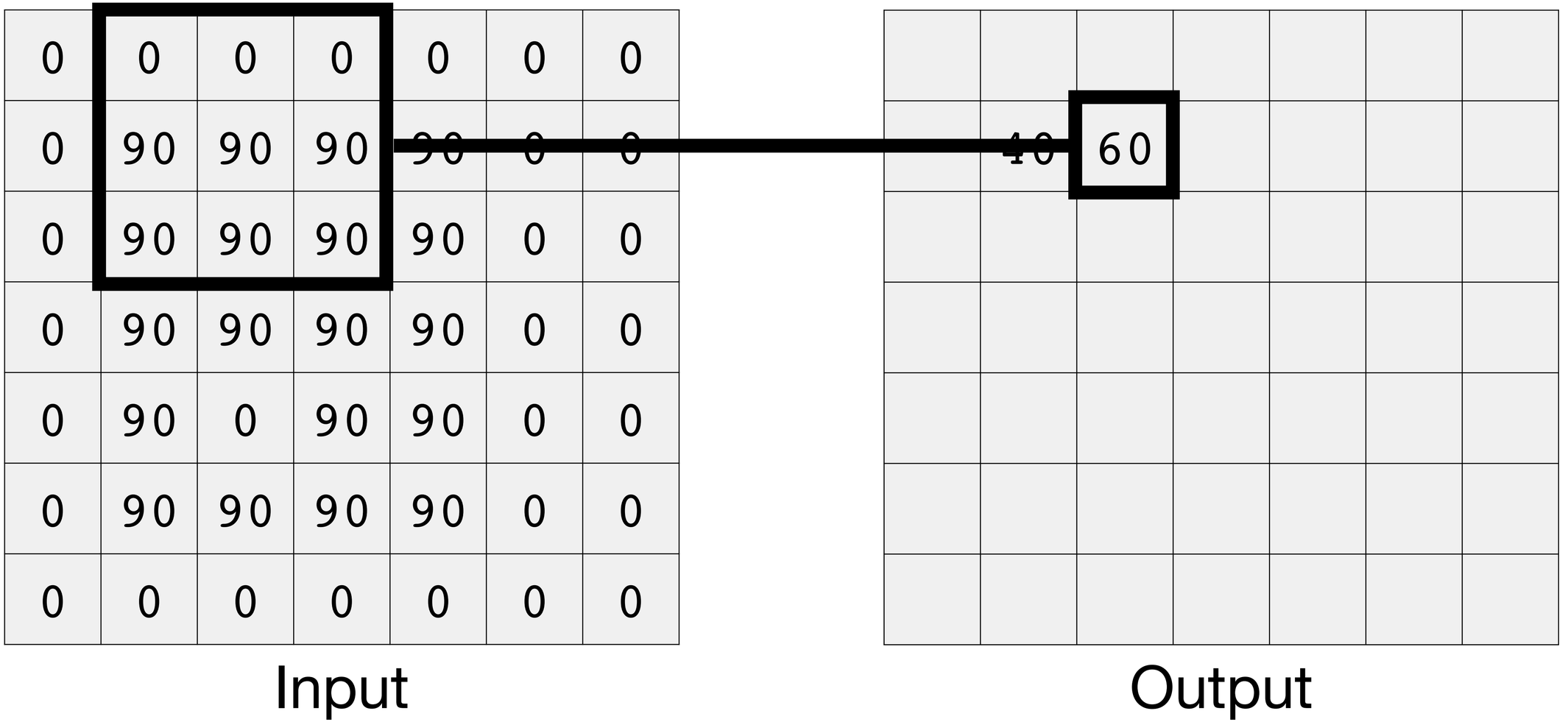


Input

Output

Recall: neighborhood filtering



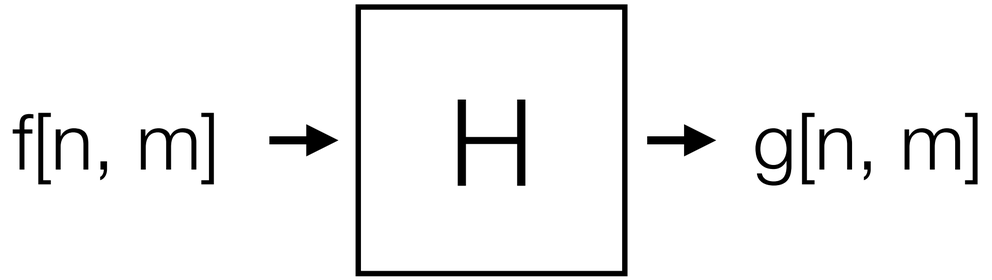


Today

- Linear filtering
- More neighborhood filters
- Nonlinear filters

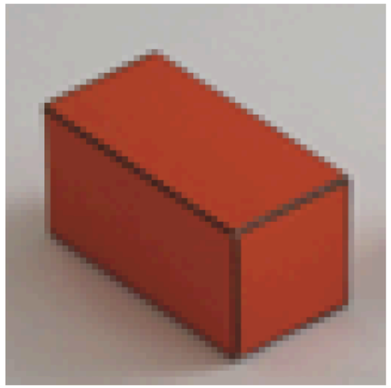
Filtering

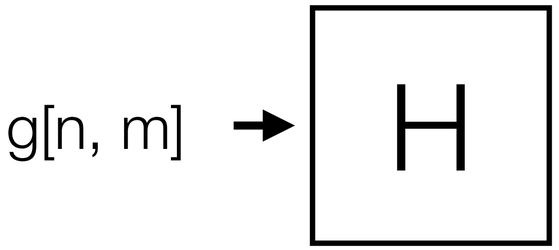
Remove unwanted sources of variation, keeping only information that's relevant for a task.

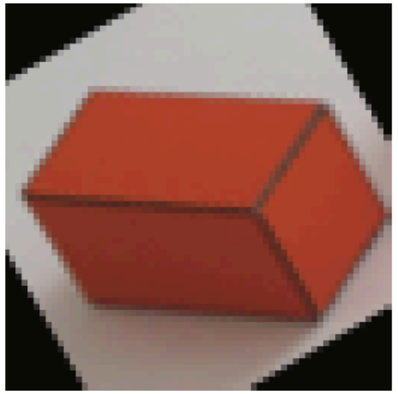


Adapted from Torralba, Freeman, Isola

Linear filtering







- Linear transformation: $f[n,m] = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h[n,m,k,l] g[k,l]$
- Equivalent to multiplication with a matrix H: $f = Hg$
- A very general transformation. Allows for may types of image changes.

Source: Torralba, Freeman, Isola

Why handle each spatial position differently?

$$f[n,m] = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} h[n,m,k,l]g[k,l]$$

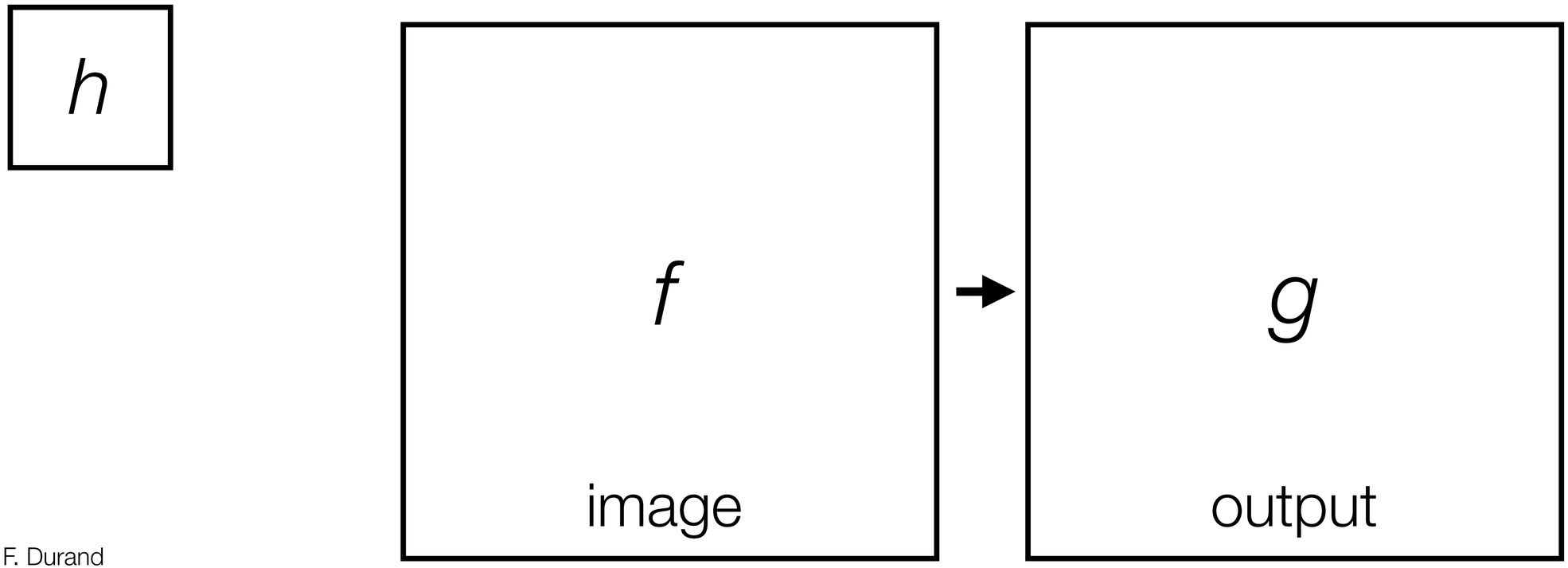
Often want **translation invariance** for our linear filters.

Photo by Fredo Durand, slide adapted from Torralba, Freeman, Isola

Cross-correlation

Let f be the image and h be the kernel. The output of cross-correlating f with h is:

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m+k, n+l] h[k,l]$$

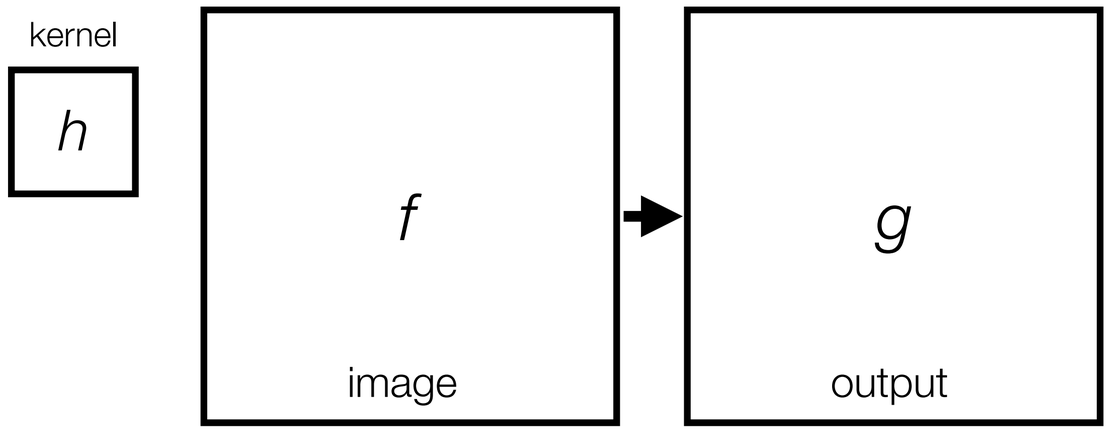


Adapted from: F. Durand

Cross-correlation

Let f be the image and h be the kernel. The output of cross-correlating f with h is:

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m+k, n+l] h[k,l]$$

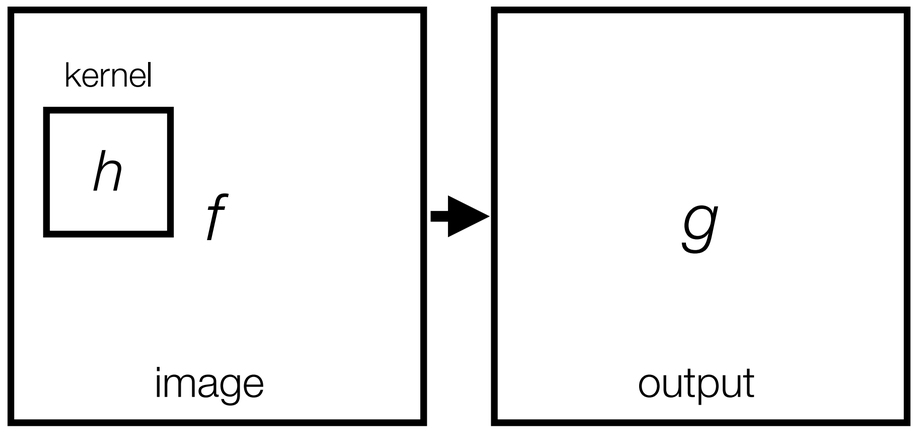


Cross-correlation

Let f be the image and h be the kernel. The output of cross-correlating f with h is:

shift by (m, n)

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m+k, n+l] h[k,l]$$



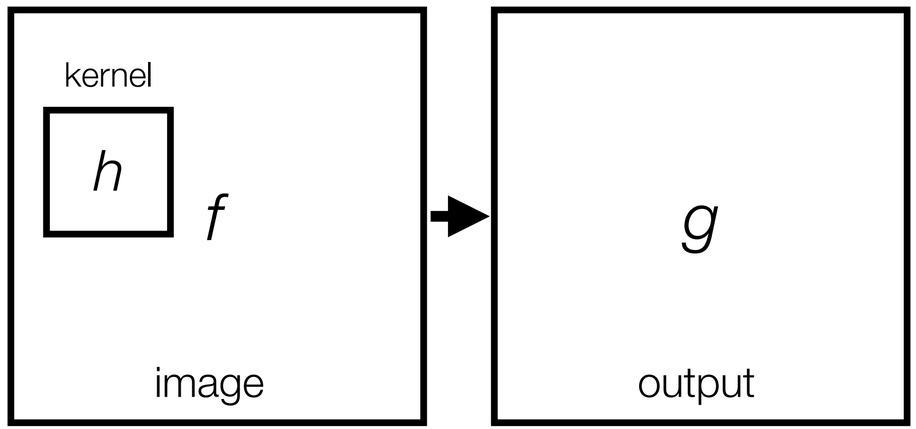
Source: F. Durand

Cross-correlation

Let f be the image and h be the kernel. The output of cross-correlating f with h is:

dot product between h and shifted f

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m+k, n+l] h[k,l]$$



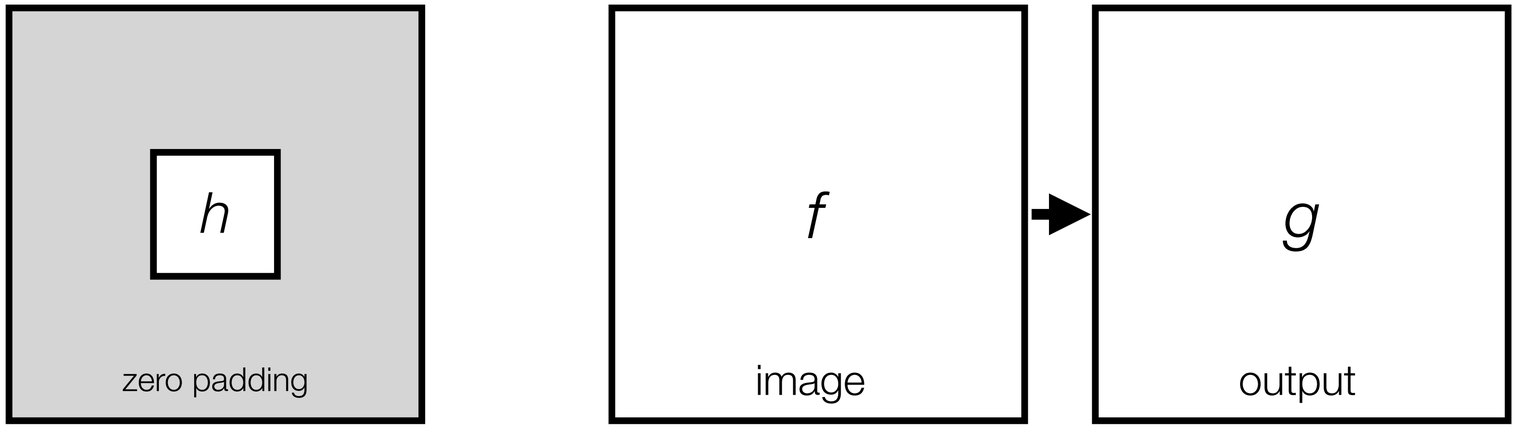
Source: F. Durand

Cross-correlation

Is this equivalent to what we saw last class? Yes.

- Given a tiny filter, pad it to make it $N \times N$.
- Convention: out of bounds indexes wrap around.

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m+k, n+l] h[k,l]$$

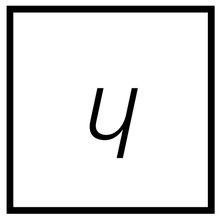


Convolution

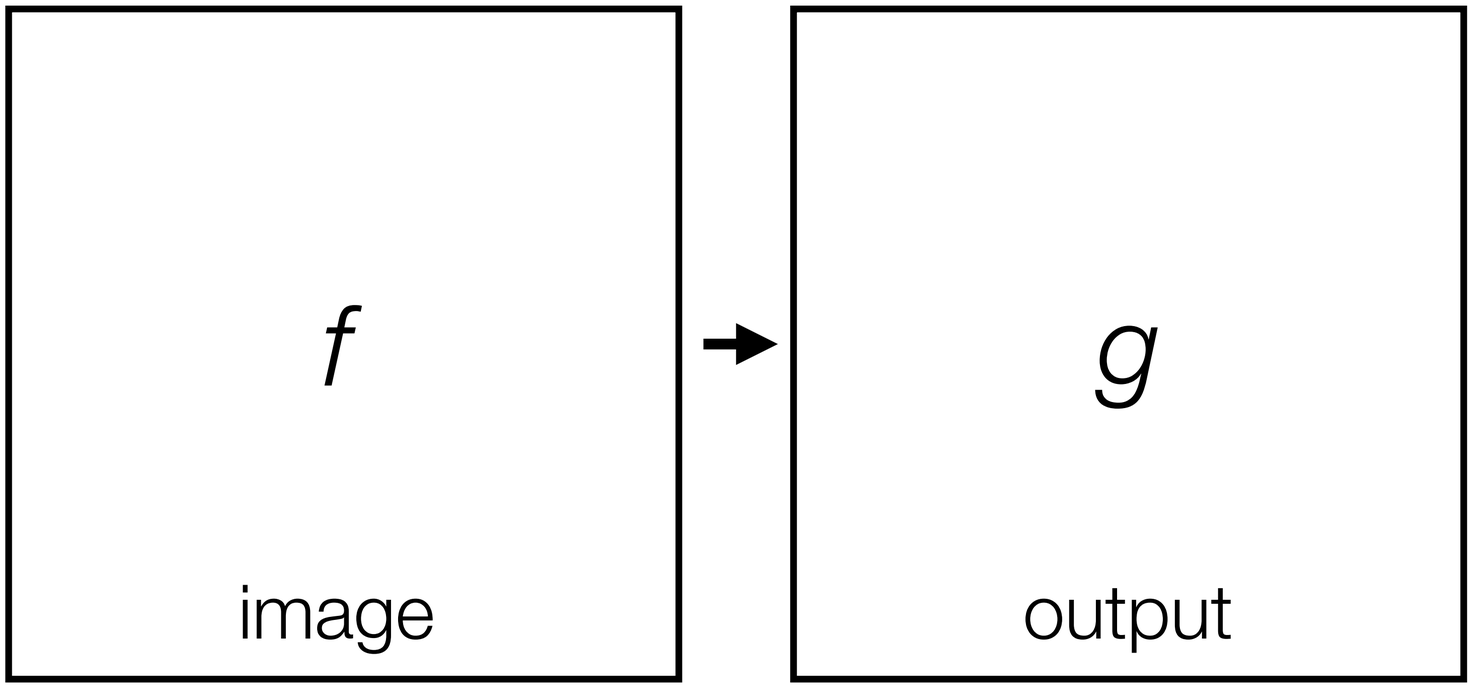
Let ***g*** be the image and ***h*** be the kernel. The output of convolving ***f*** with ***h*** is:

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m-k, n-l] h[k,l]$$

kernel



Convention:
kernel is “flipped”



Source: F. Durand

Why flip the kernel?

$$g[m,n] = f \circ h = \sum_{k=1}^{N} \sum_{l=1}^{N} f[m-k, n-l] h[k,l]$$

Indexes go backward!

- Gives it nice mathematical properties.
- When filter is symmetric, equivalent to cross-correlation.

Properties of the convolution

Commutative (no distinction between filter and image):

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

Associative (doesn't matter what order you do 2 convolutions):

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

Distributive with respect to the sum:

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

Today

- Linear filtering
- **More neighborhood filters**
- Nonlinear filters

Rectangular filters

f[m,n]

h[m,n]

=

g[m,n]

Source: Torralba, Freeman, Isola

Rectangular filters

f[m,n]

h[m,n]

=

g[m,n]

Source: Torralba, Freeman, Isola

“Naturally” occurring filters

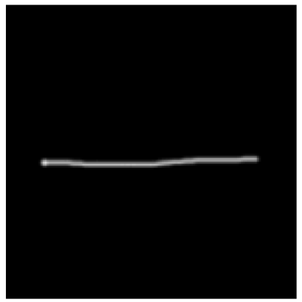
Input image

Motion blur

Source: Torralba, Freeman, Isola

“Naturally” occurring filters

Input image

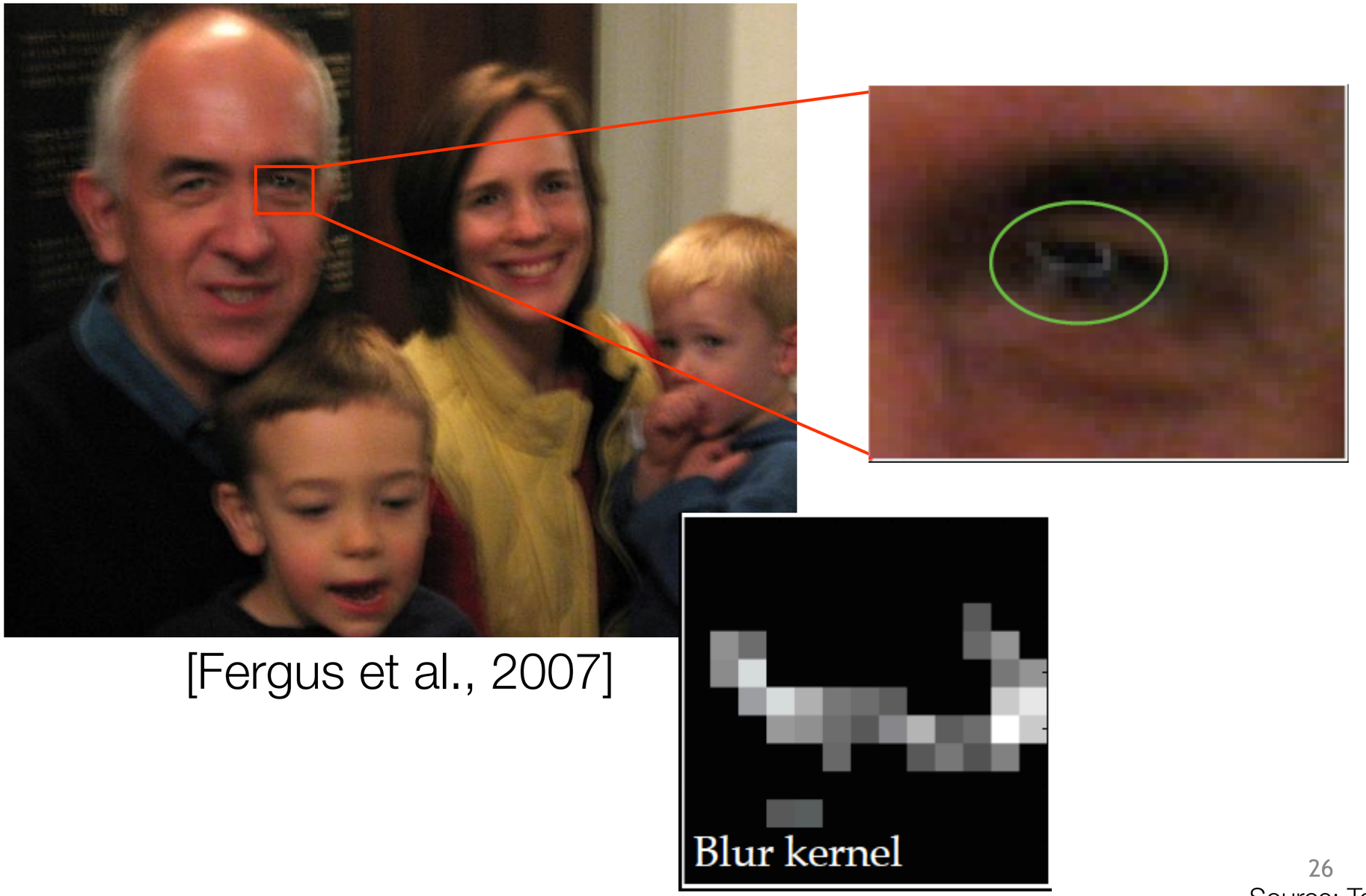


Convolution weights

Convolution output

Source: Torralba, Freeman, Isola

Camera shake



[Fergus et al., 2007]

Source: Torralba, Freeman, Isola

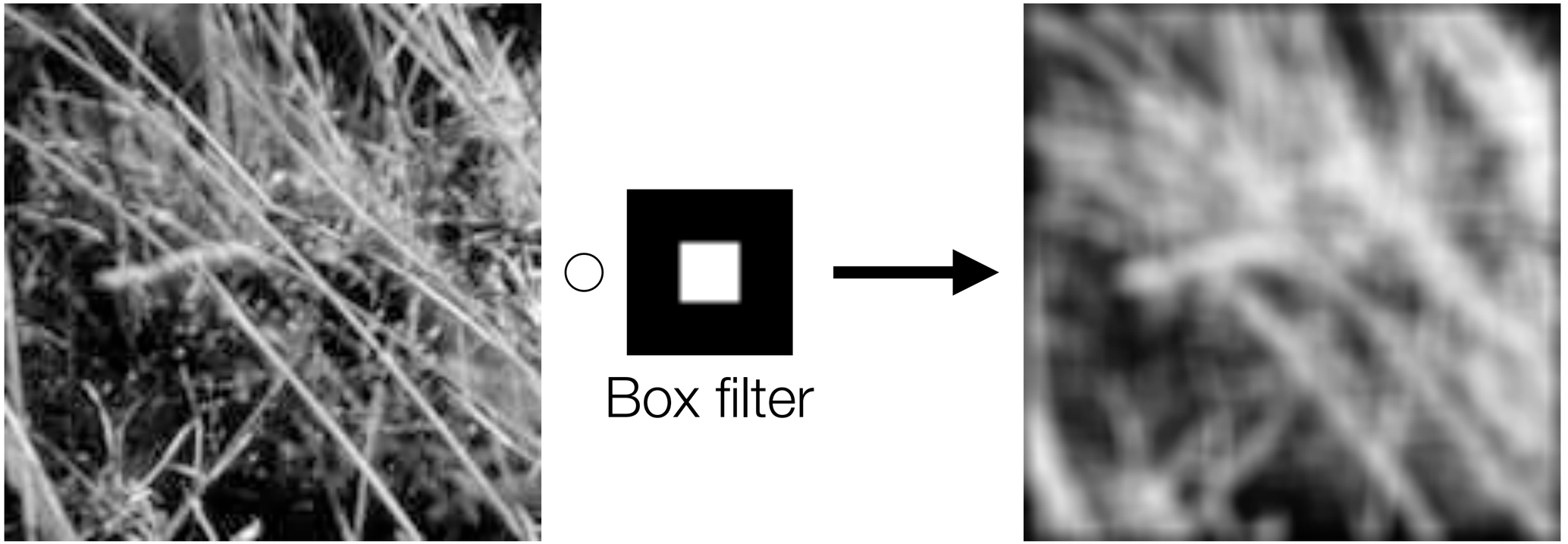
“Naturally” occurring filters

Blur

Source: Torralba, Freeman, Isola

Blurring revisited

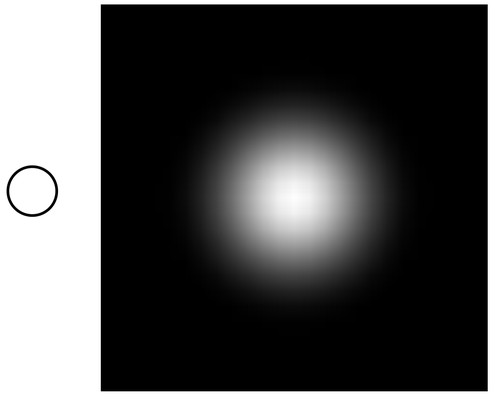
What's wrong with this result?



Source: D. Forsyth

Blurring revisited

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

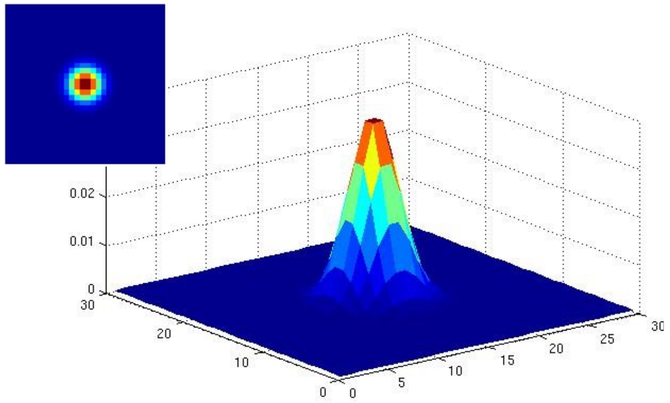


“fuzzy blob”

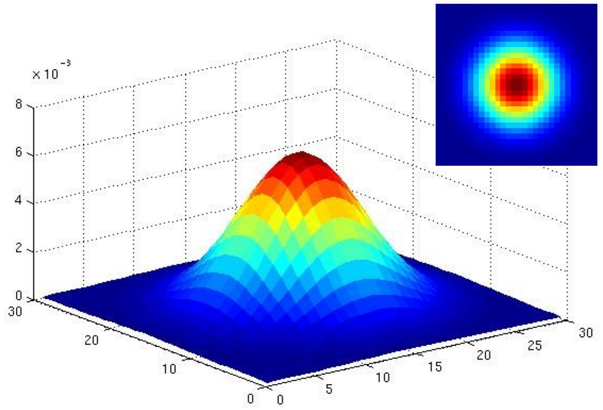
Source: S. Lazebnik

Gaussian kernel

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



σ = 2 with 30 x 30 kernel

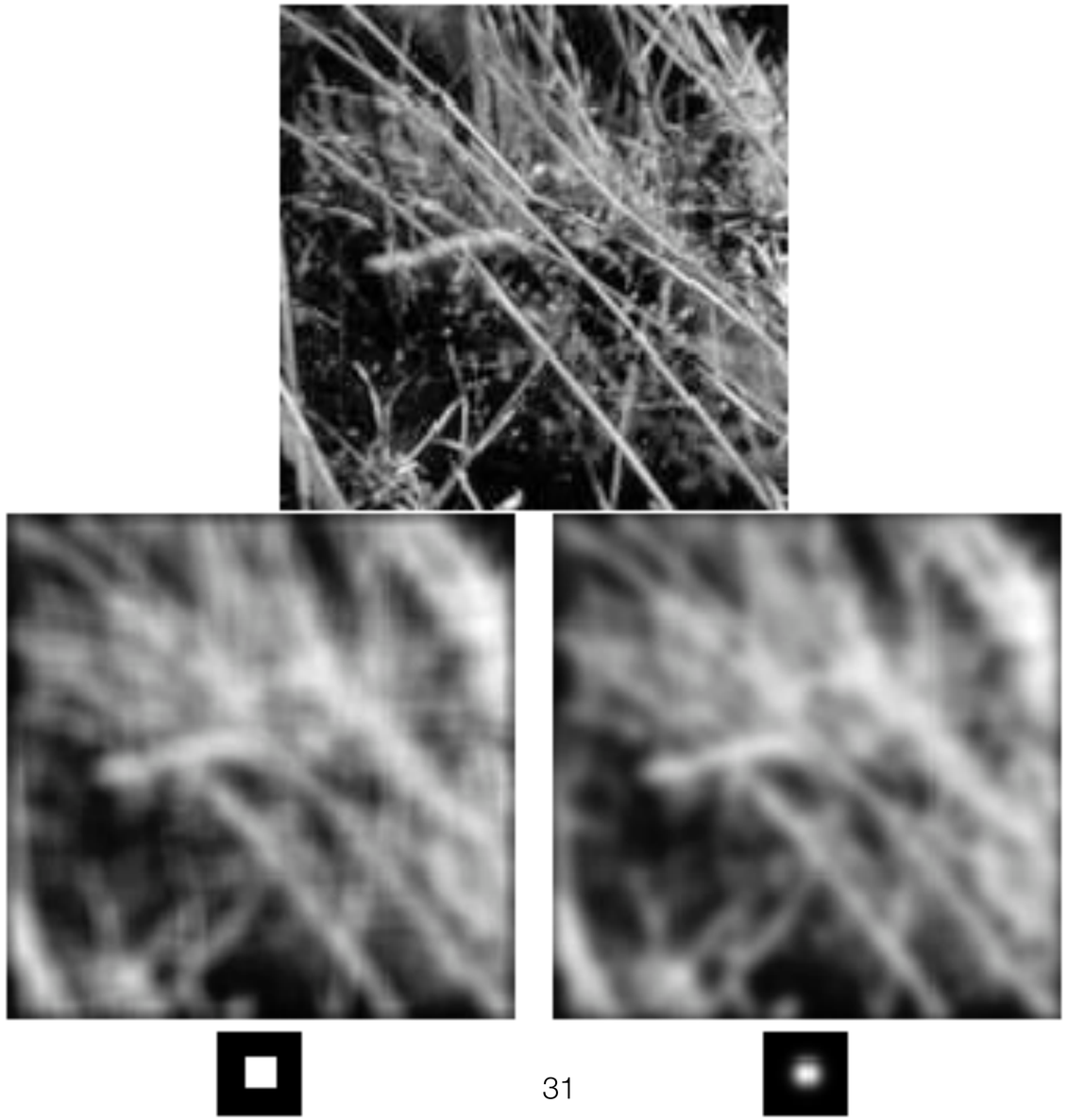


σ = 5 with 30 x 30 kernel

- Constant factor makes kernel sum to 1
  - In practice: create filter without the factor, then normalize it.
- Make sure to center it: (x, y) is the center of the kernel, not a corner

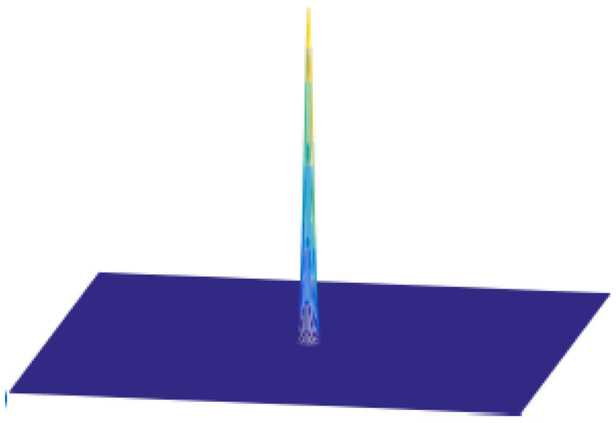
Source: K. Grauman

Gaussian vs. box filtering

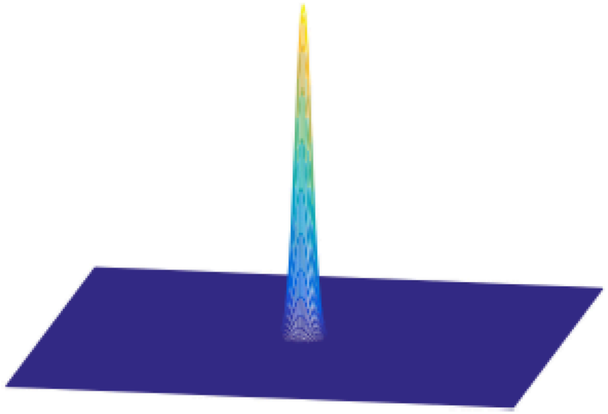


Source: S. Lazebnik

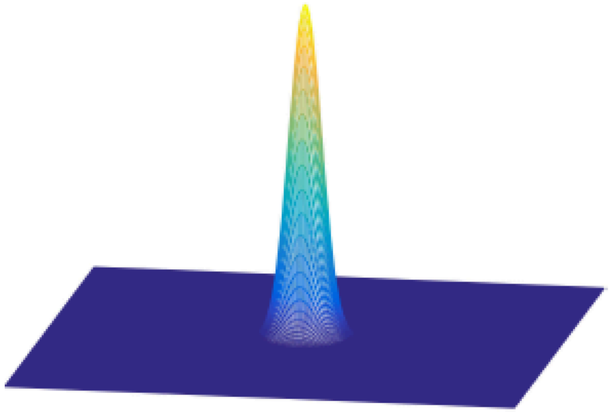
Gaussian standard deviation



σ=2



σ=4

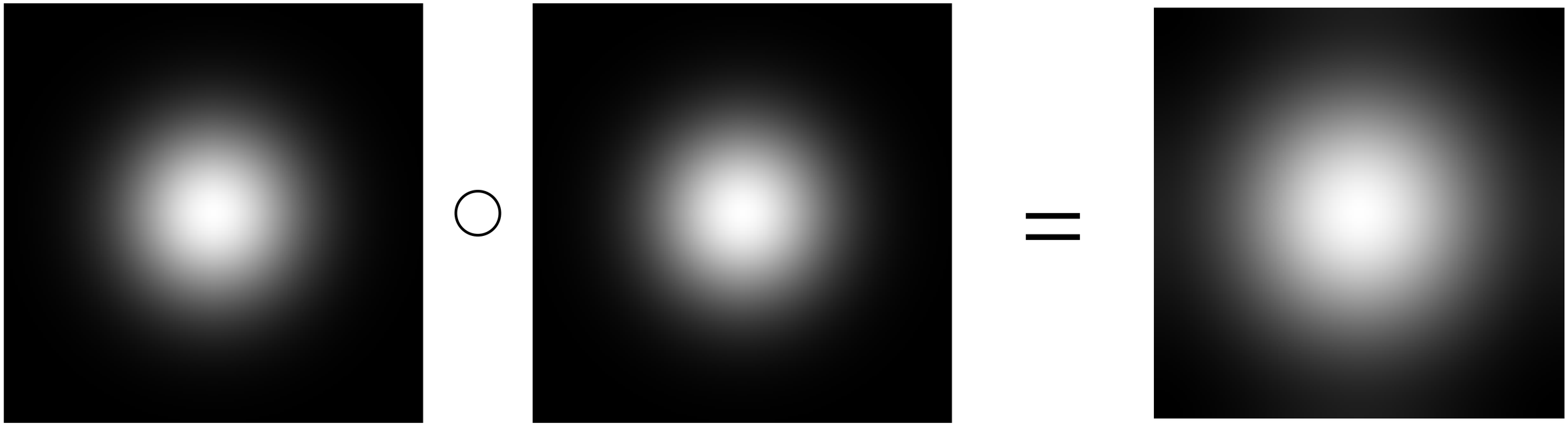


σ=8

Source: Torralba, Freeman, Isola

Useful properties of Gaussian filters

Convolve a Gaussian filter with itself? Get another Gaussian.



Useful properties of Gaussian filters

Smooth with small σ repeatedly. Equivalent to smoothing with large σ!

I

blur(blur(I))

blur(blur(blur(I)))

blur(blur(blur(blur(I))))

Exploits associativity:

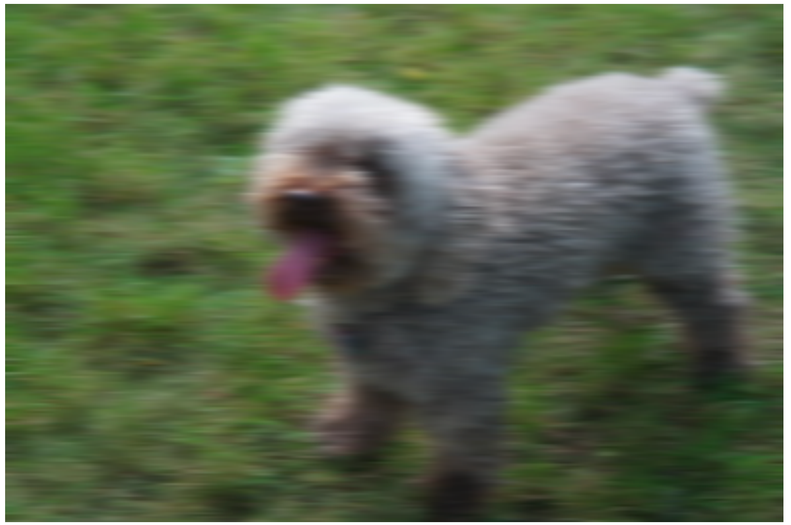
$$I \circ (h \circ h \circ \ldots \circ h) = ((I \circ h) \circ h) \circ \ldots \circ h$$

Useful properties of Gaussian filters

Gaussian kernels are *separable*.

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

I



$\mathrm{blur}_x(\mathrm{I})$

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

Recall: derivatives

$$d_0 = [1, -1] \qquad f \circ d_0 = f[n] - f[n-1]$$

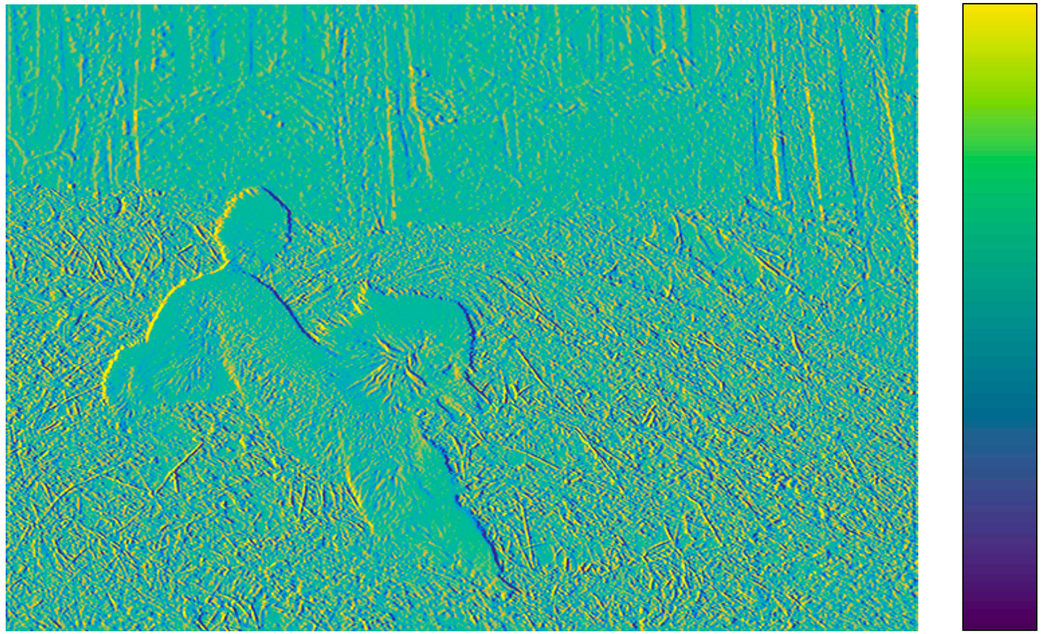
Another option:

$$d_1 = [1, 0, -1]/2 \qquad f \circ d_1 = \frac{f[n+1] - f[n-1]}{2}$$

Adapted from: Torralba, Freeman, Isola

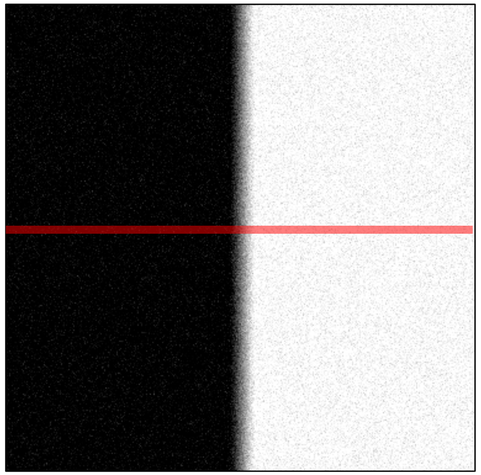
Problems with simple derivative filters

○ [1 -1]



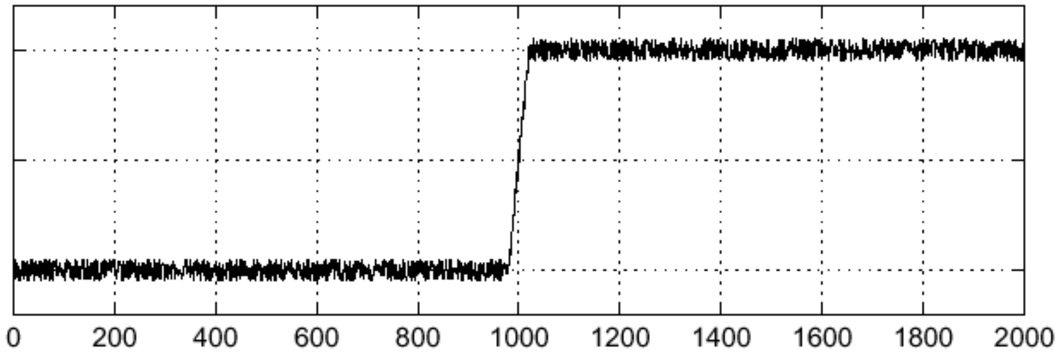
Sensitive to “tiny” edges

Where's the edge?

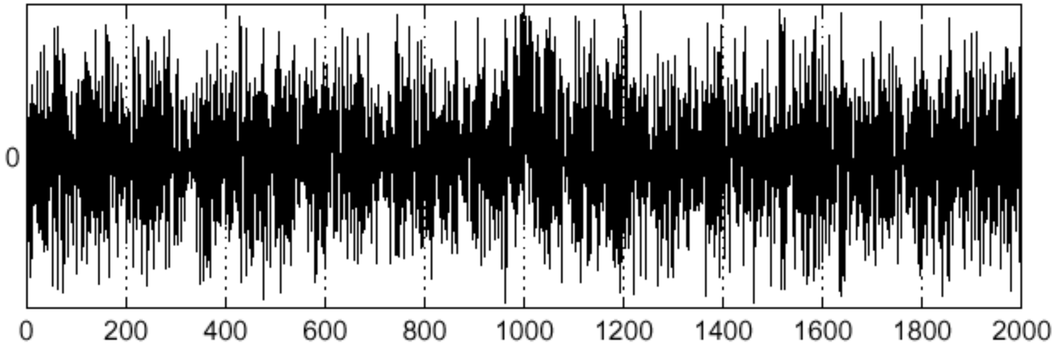


Noisy input image

$f(x)$

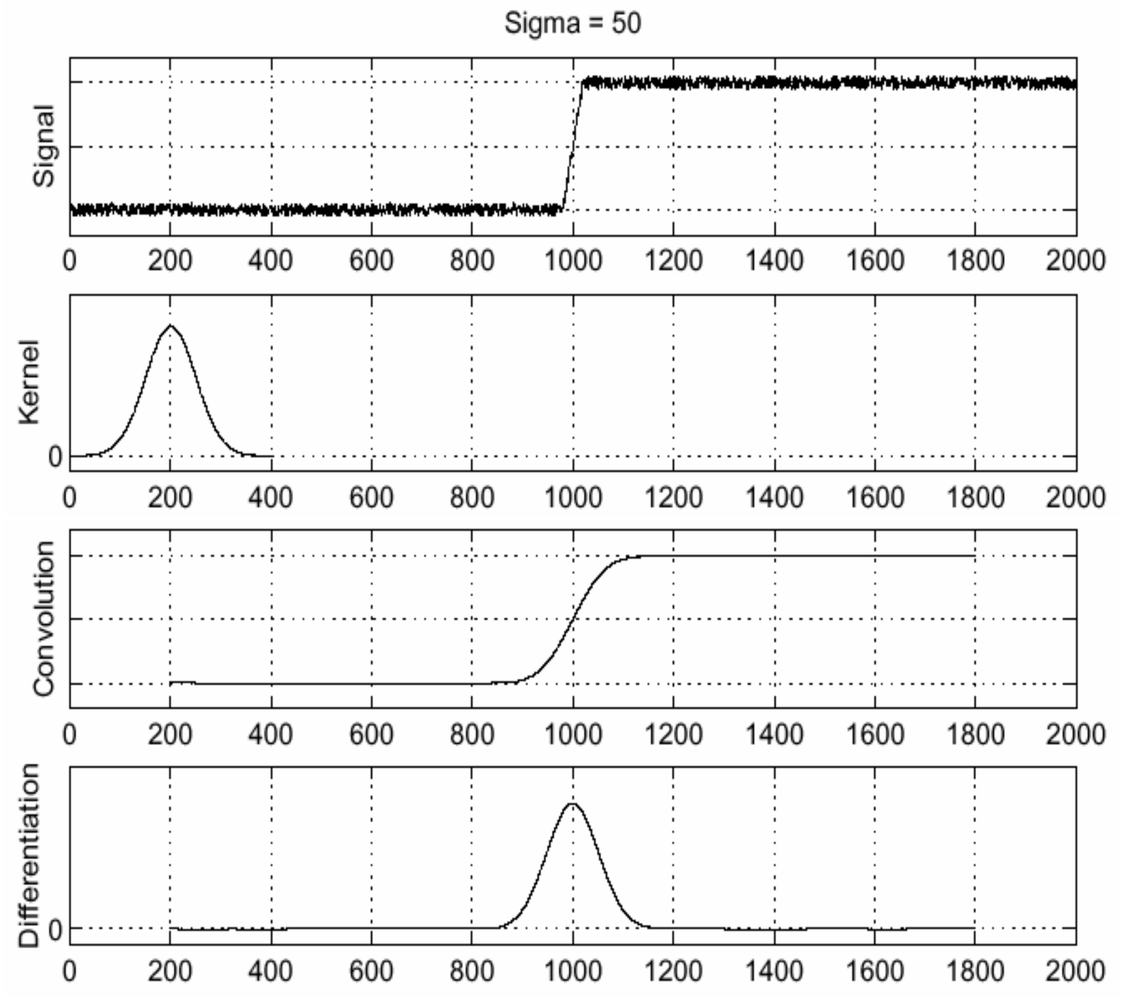


$\frac{d}{dx}f(x)$



Source: S. Seitz

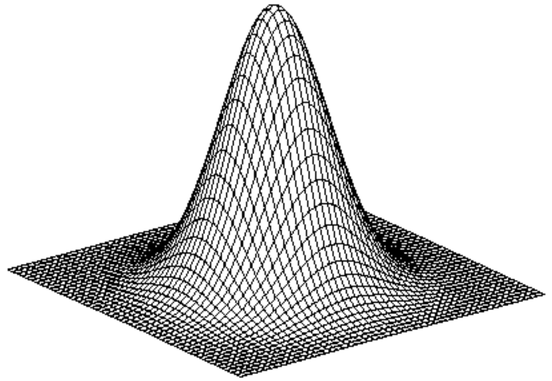
Solution: smooth first



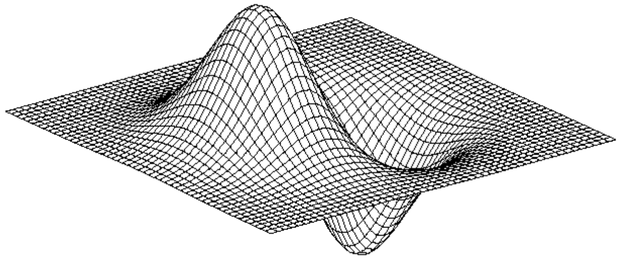
To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

Derivative of Gaussian filter



Gaussian



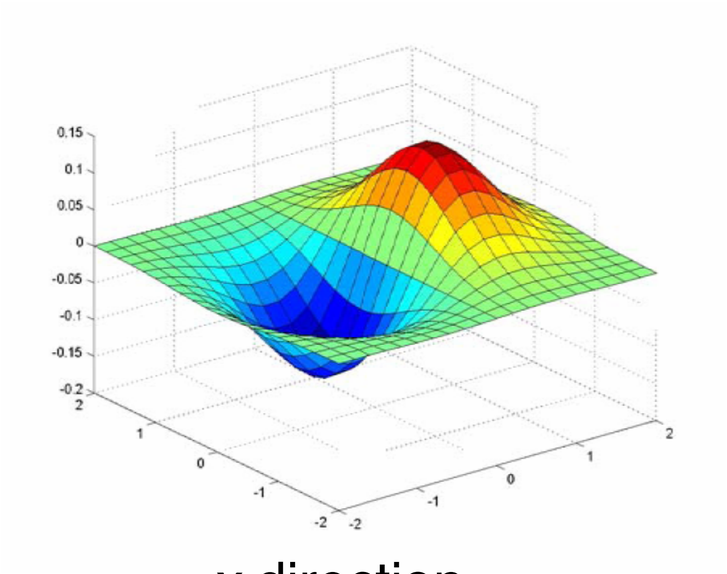
Derivative of Gaussian (x)

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

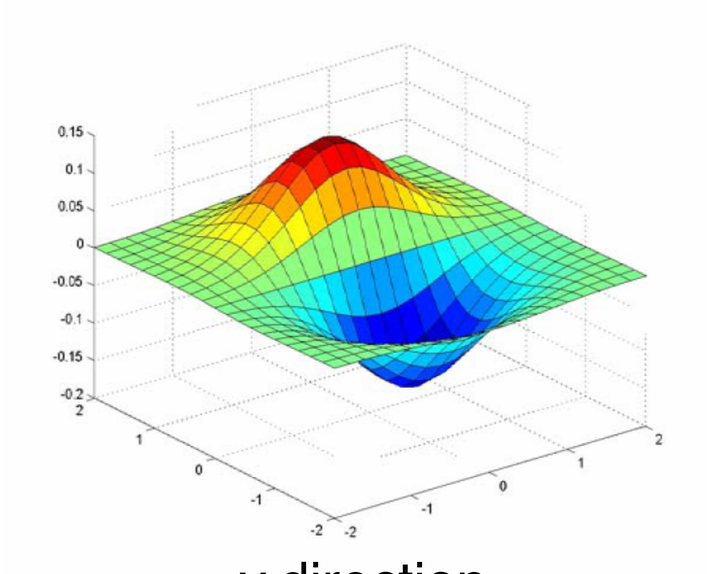
$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Source: N. Snavely

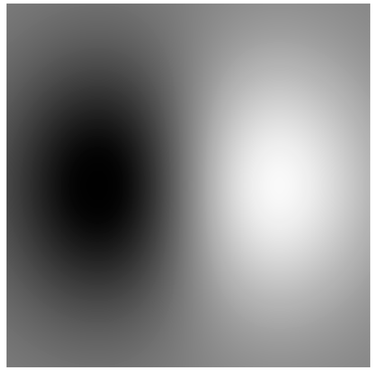
Derivative of Gaussian filter

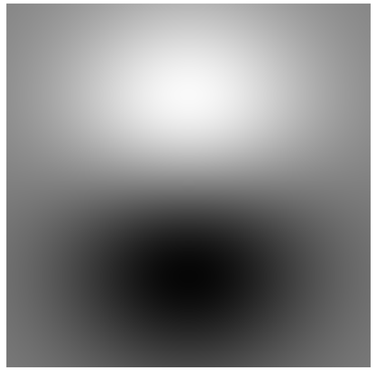


x-direction



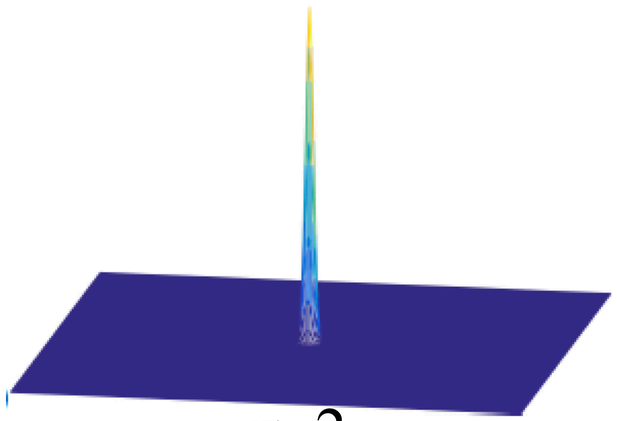
y-direction



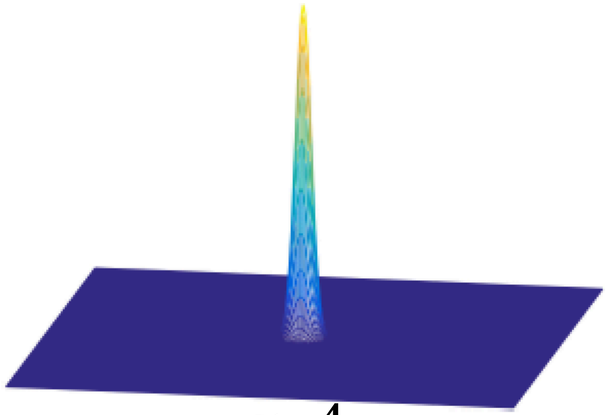


Source: N. Snavely

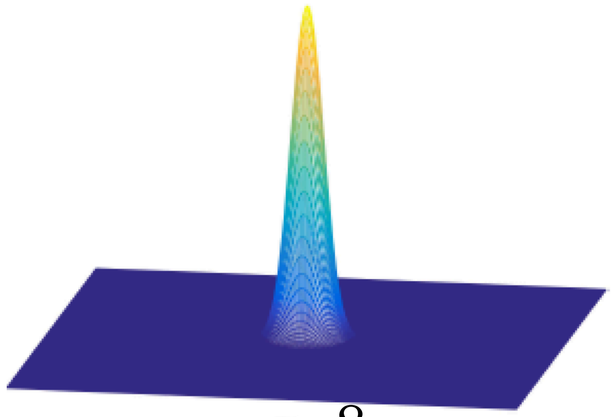
Gaussian Scale



σ=2



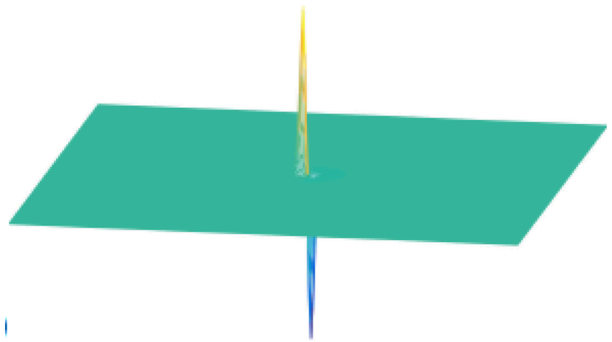
σ=4



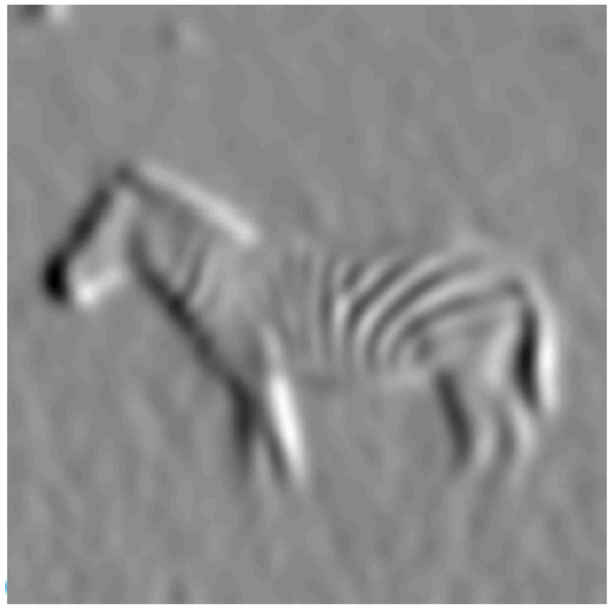
σ=8

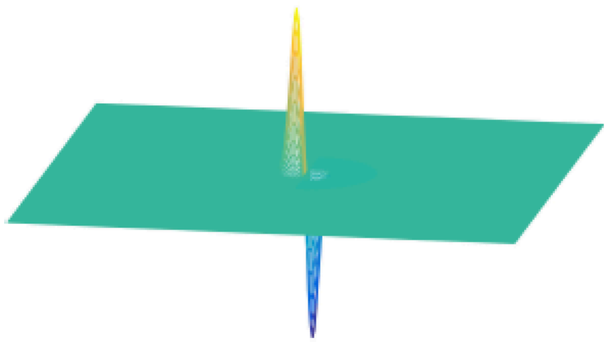
Source: Torralba, Freeman, Isola

Derivative of Gaussian scales

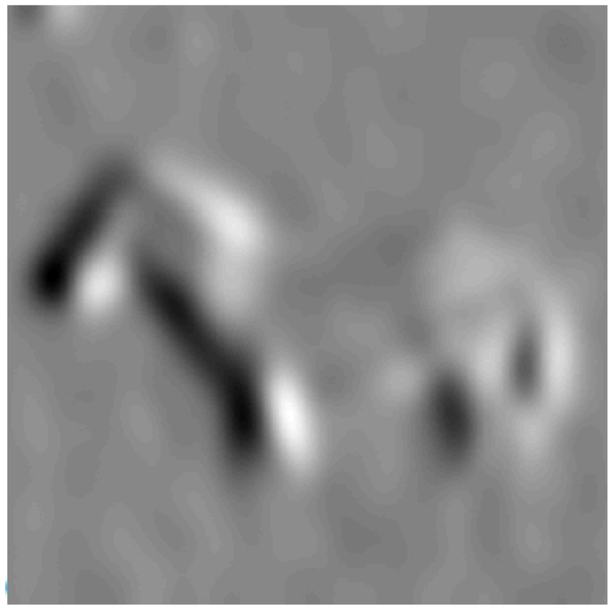


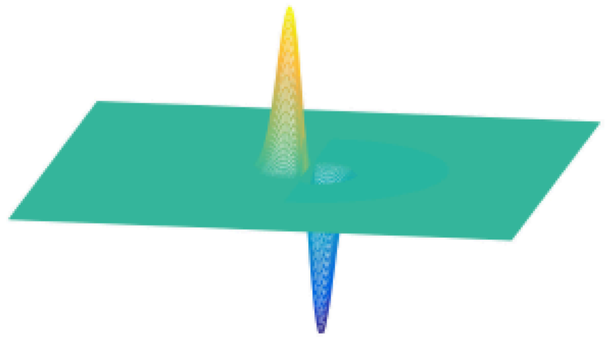
σ=2





σ=4





σ=8

Source: Torralba, Freeman, Isola

Derivatives in other directions

Define a filter?

1	0
0	-1

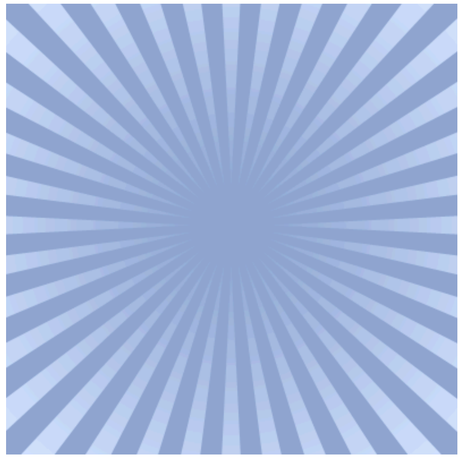
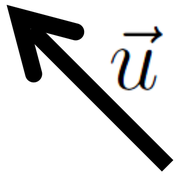
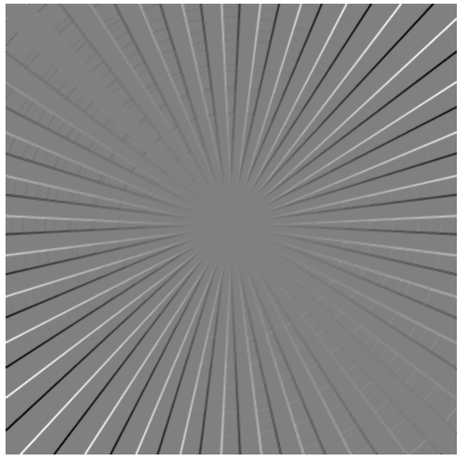


Image f

desired direction



$\nabla_{\vec{u}} f$ = ?



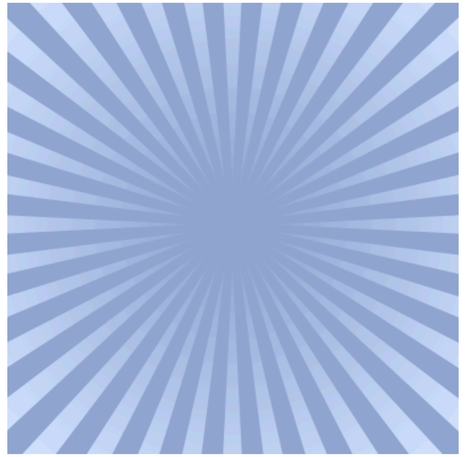
$\nabla_{\vec{u}} f$

What if we need *lots* of angles? This could get expensive.

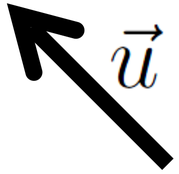
Adapted from N. Snavely

Steerable filters

Given $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$, steer to direction $\vec{u} = \begin{bmatrix} u_x & u_y \end{bmatrix}$.



f



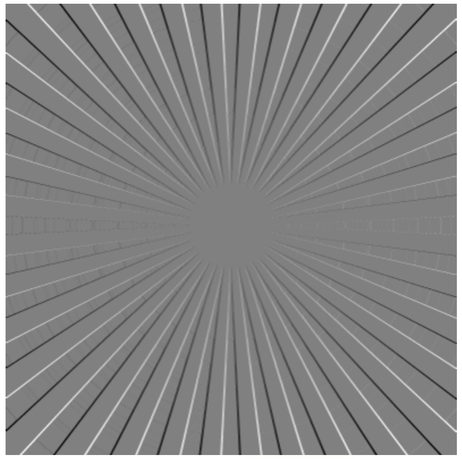
$\vec{u}$

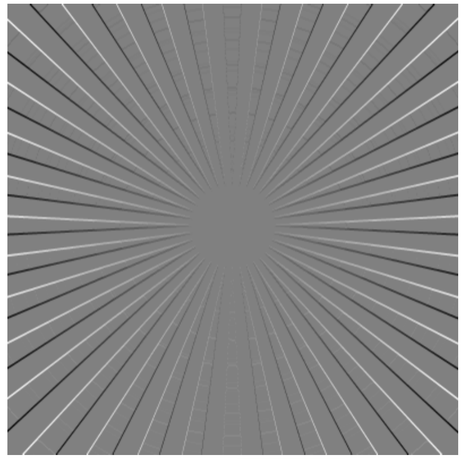
$\nabla_{\vec{u}} f = ?$

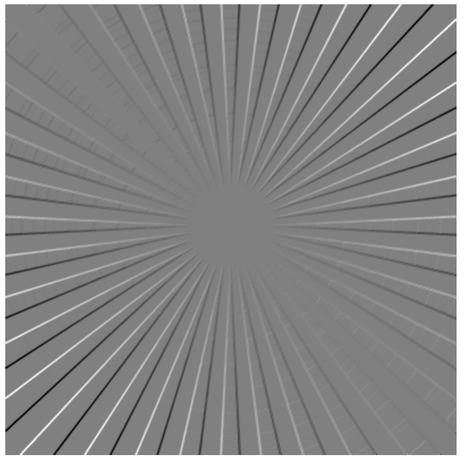
Use a multivariable calculus fact:

$$\nabla_{\vec{u}} f(\vec{x}) = \nabla f(\vec{x}) \cdot \vec{u}$$

Directional derivative = linear combination of partial derivatives.



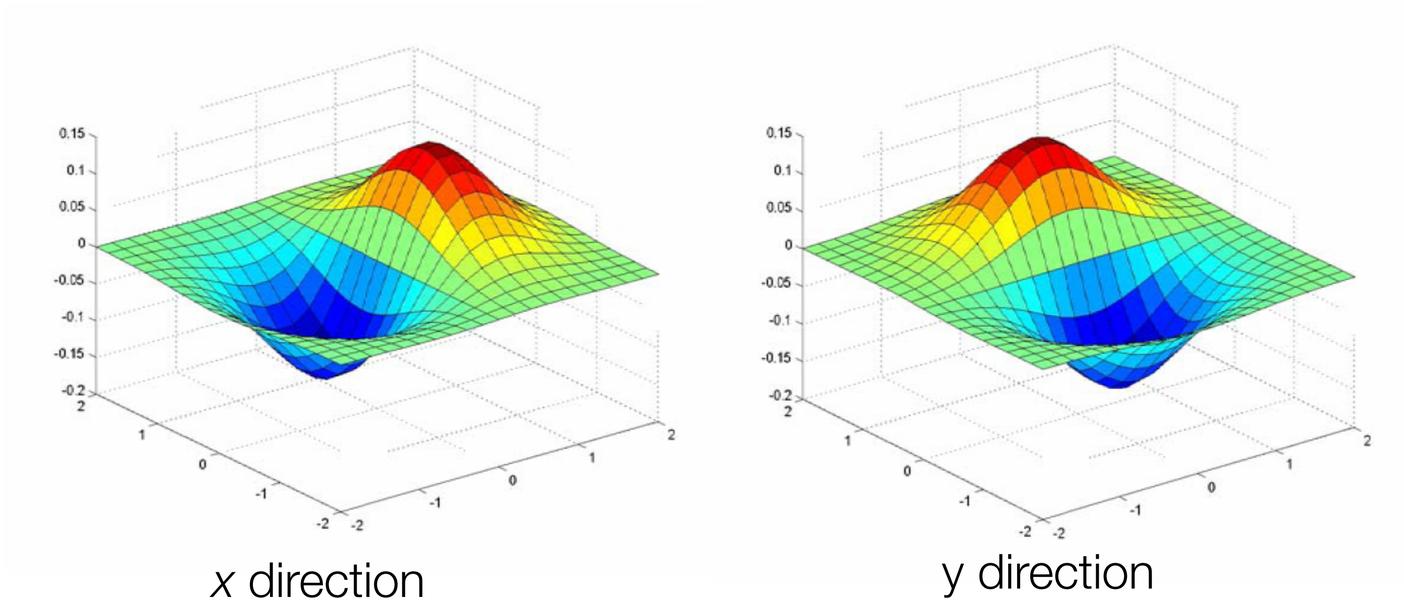
$\frac{\partial f}{\partial x} \cdot u_x$ + 

$\frac{\partial f}{\partial y} \cdot u_y$ = 

$\nabla_{\vec{u}} f$

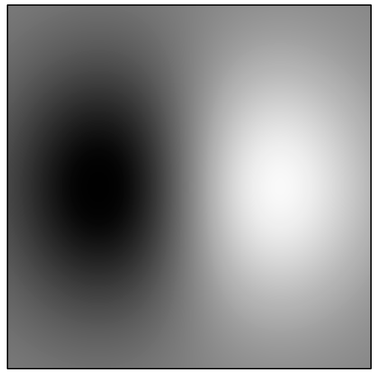
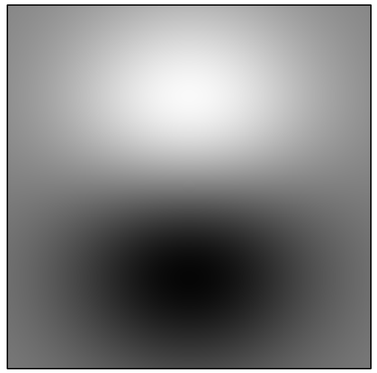
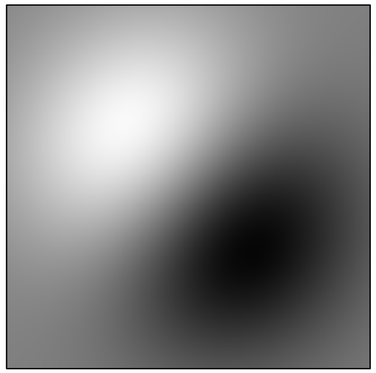
Source: N. Snavely

Also works for derivative of Gaussian filter



x direction

y direction

$$\cos(\theta)$$  $$+ \quad \sin(\theta)$$  $$=$$ 

See [Freeman and Adelson, “Design and Use of Steerable Filters”, 1991] for a deeper look.

Source: N. Snavely

Filter approximations

Binomial filter ≈ Gaussian

$\frac{1}{4}$ | 1 | 2 | 1 |

$\frac{1}{8}$ | 1 | 3 | 3 | 1 |

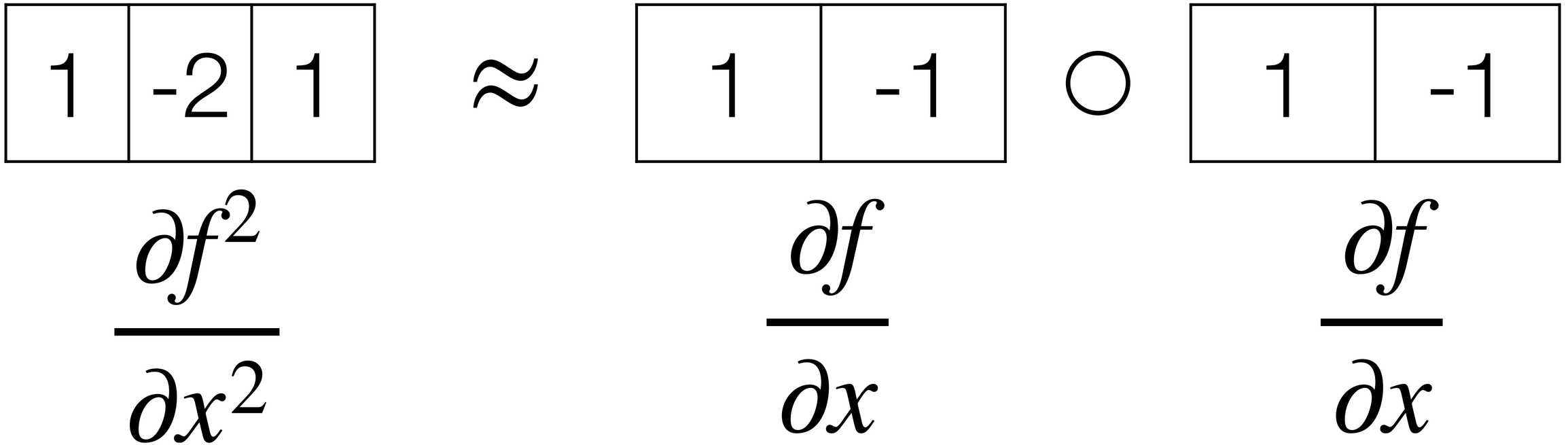
$\frac{1}{16}$ | 1 | 4 | 6 | 4 | 1 |

Sobel filter ≈
Derivative of Gaussian

$\frac{1}{8}$

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

2nd derivatives



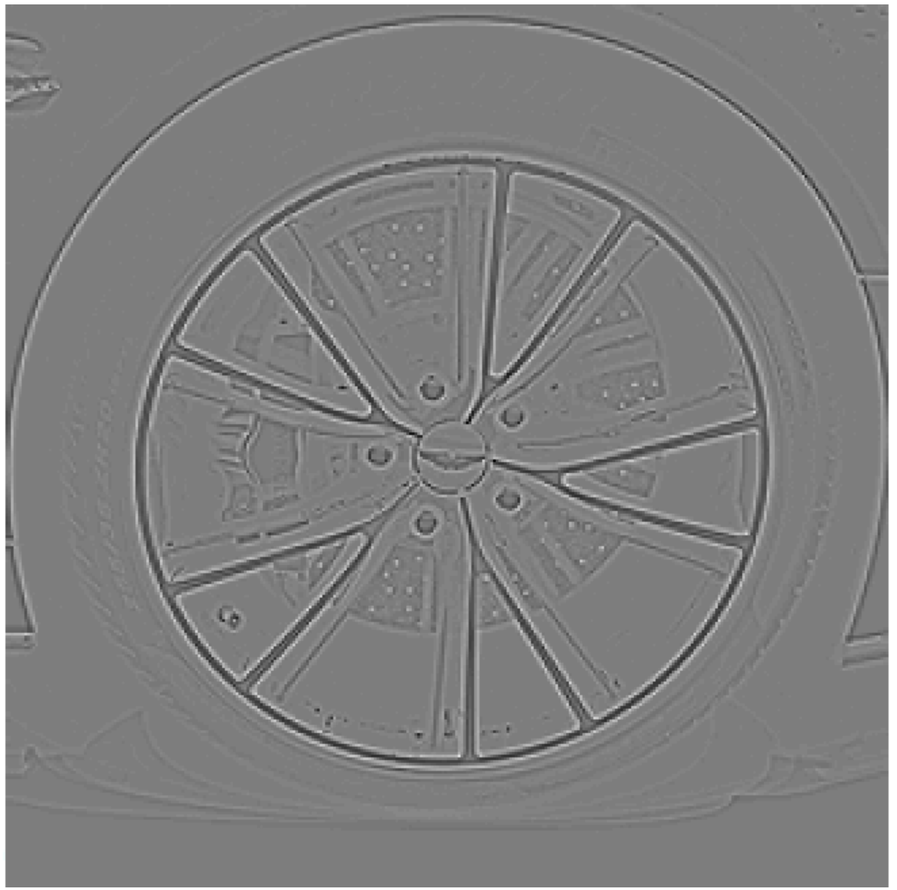
Laplacian filter

$$\frac{\partial I^2}{\partial x^2} + \frac{\partial I^2}{\partial y^2} \approx I \circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$= I \circ [1 \quad -2 \quad 1] + I \circ [1 \quad -2 \quad 1]^{\top}$$

Laplacian filter

$$\circ \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} =$$

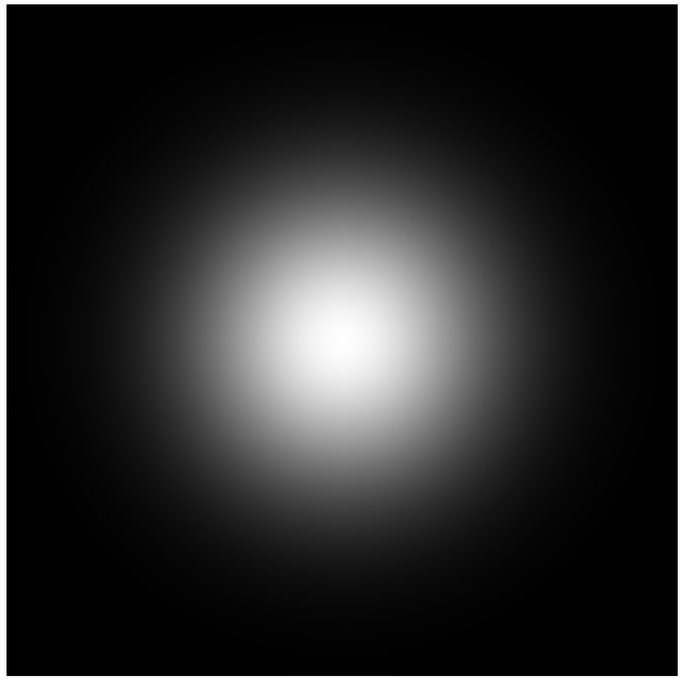


Detects edges and blobs, and not sensitive to orientation. But sensitive to noise.

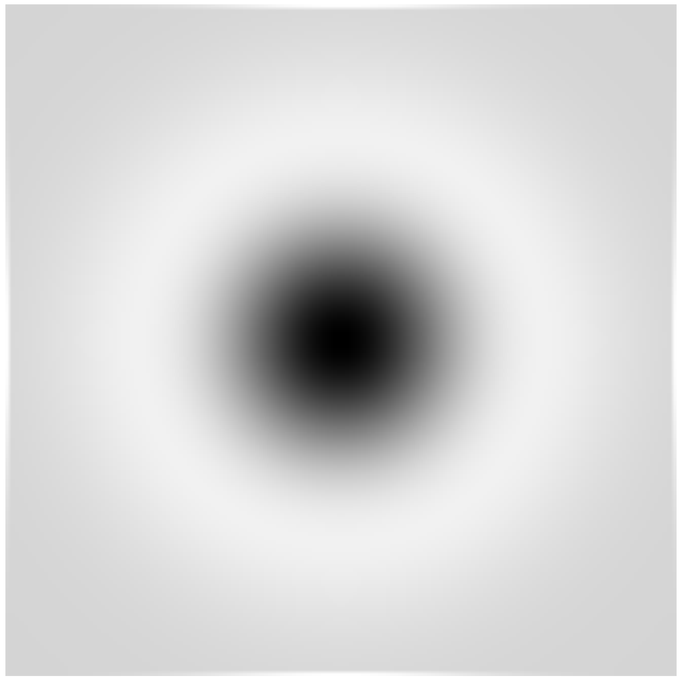
Source: Torralba, Freeman, and Isola

Laplacian of Gaussian

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \circ$$



$=$



Combine with Gaussian. Picks up “blob-like” structures.

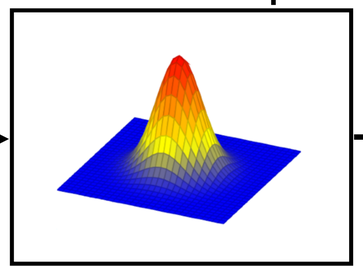
“Blob” detection with Laplacian

sigma = 11.9912

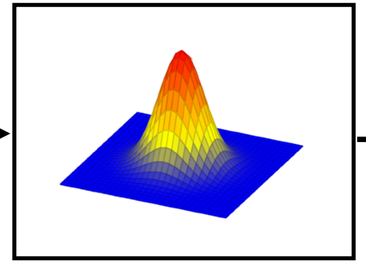
Source: S. Lazebnik

Decomposing an image

Gaussian filter
(a.k.a. “low pass”)

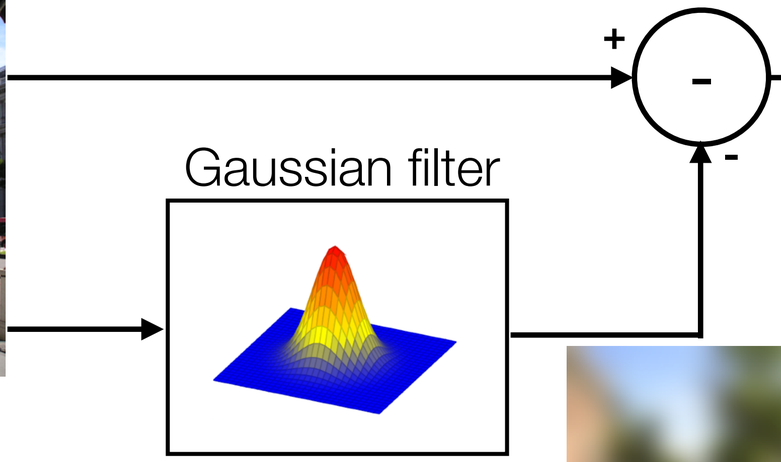


Approx. Laplacian
(a.k.a. “high pass”)



Source: Torralba, Freeman, Isola

Approx. Laplacian

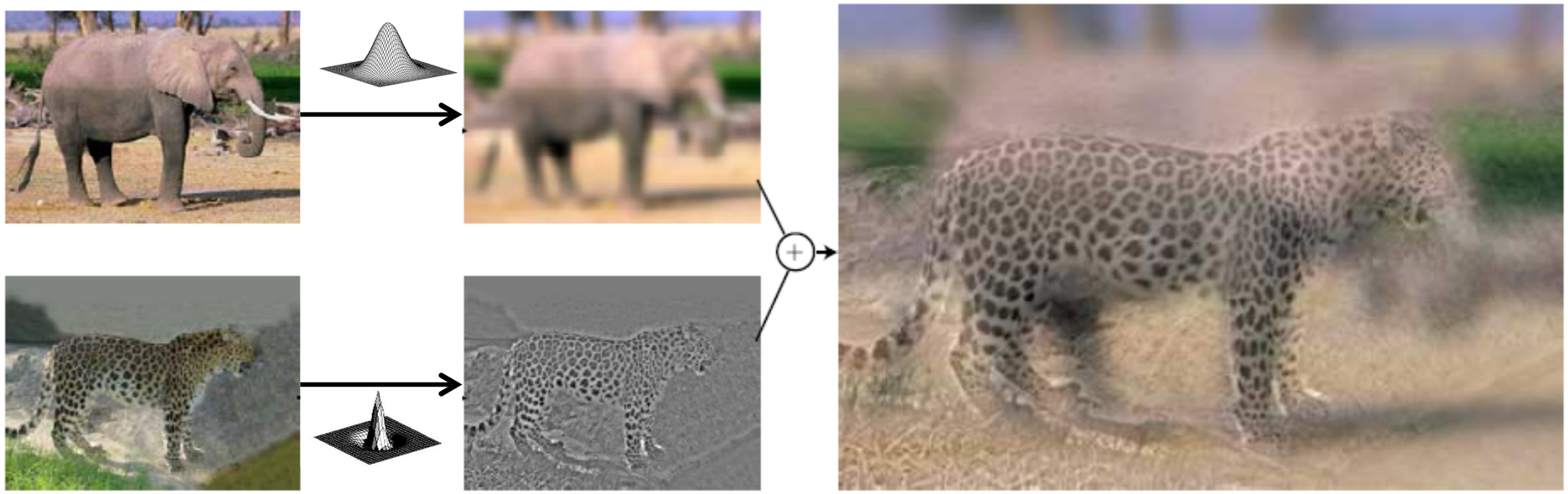


+

=

Source: Torralba, Freeman, Isola

Application: Hybrid Images



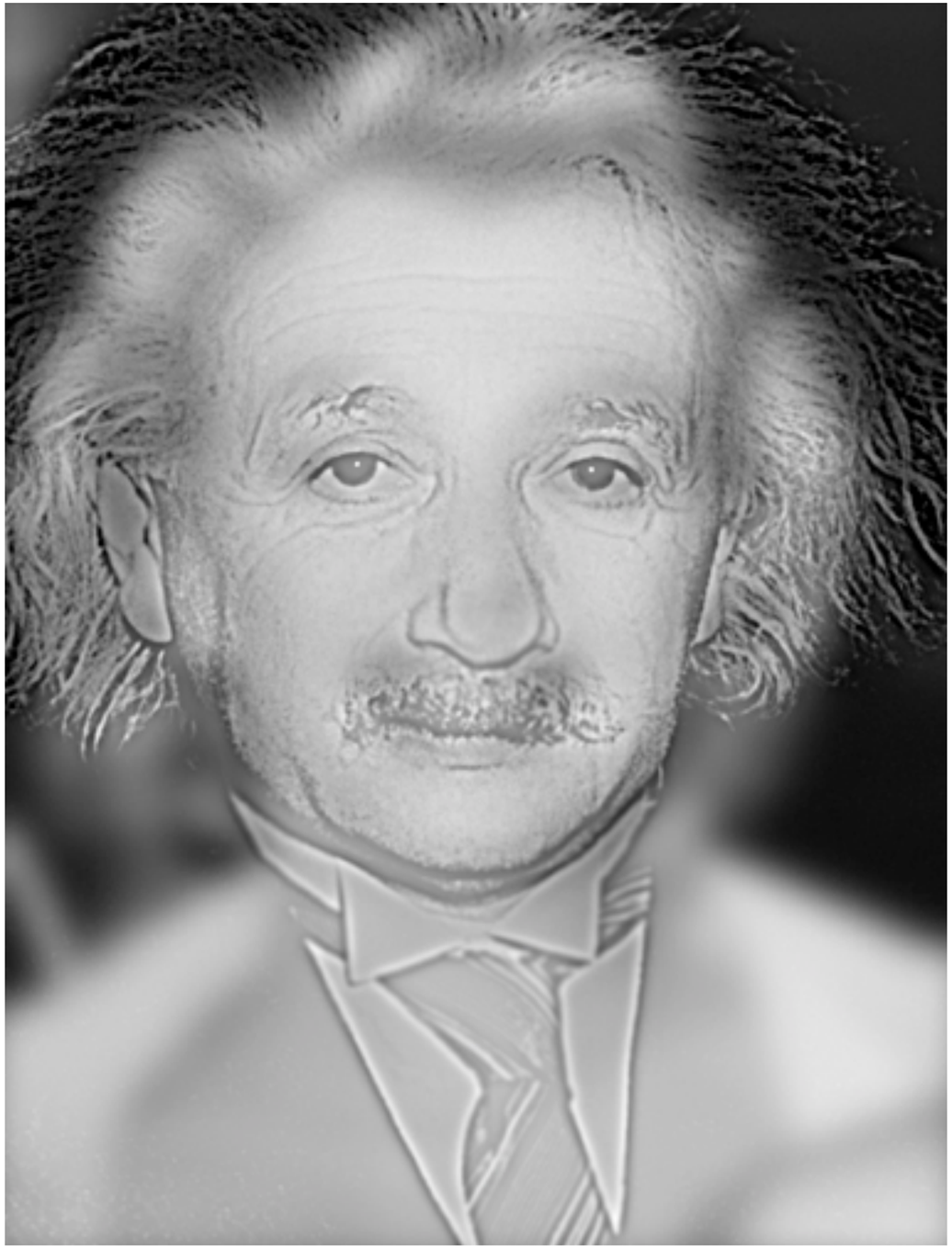
[A. Oliva, A. Torralba, P.G. Schyns, Hybrid Images, SIGGRAPH 2006]

Source: S. Lazebnik

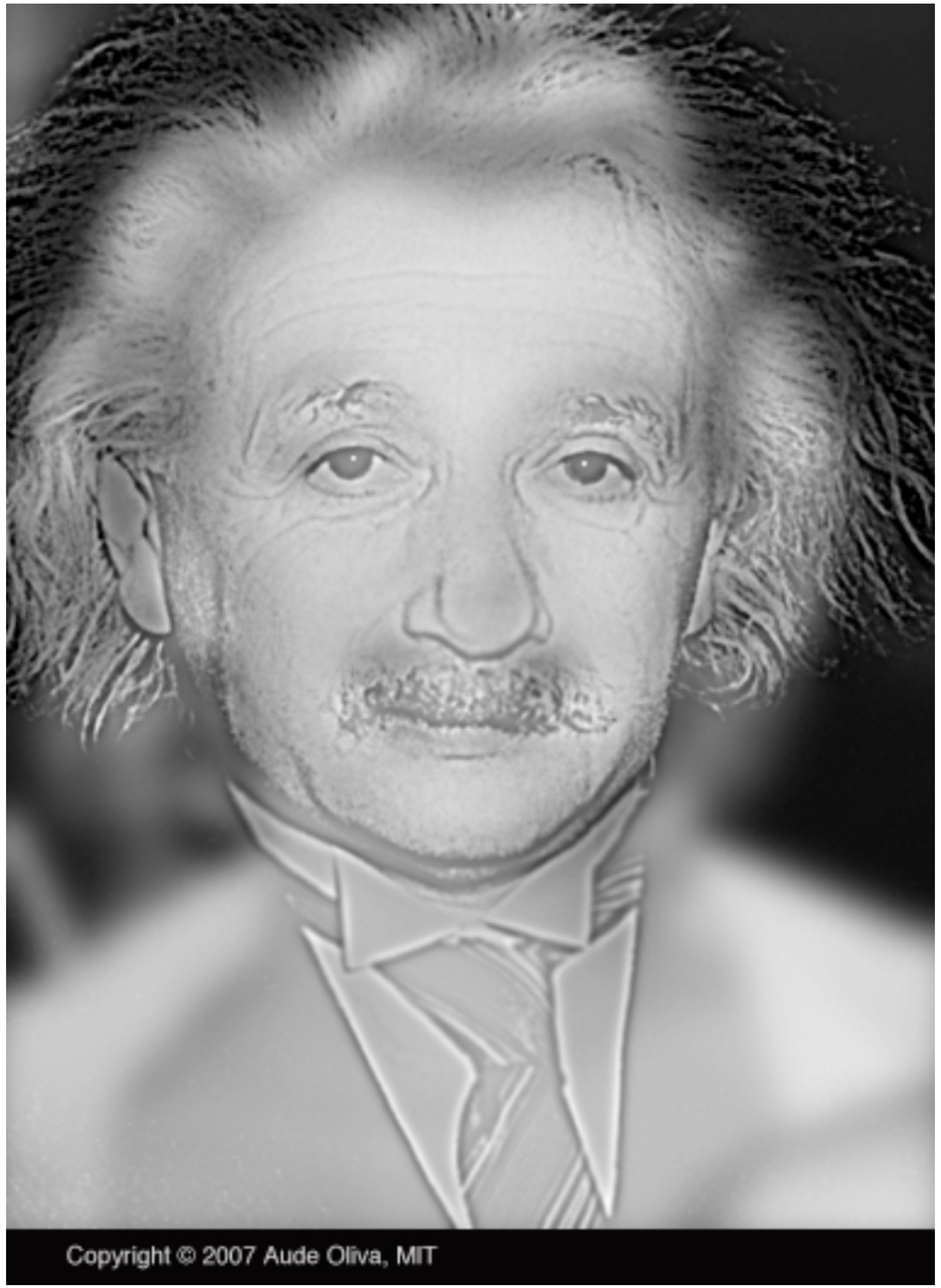
Hybrid Images

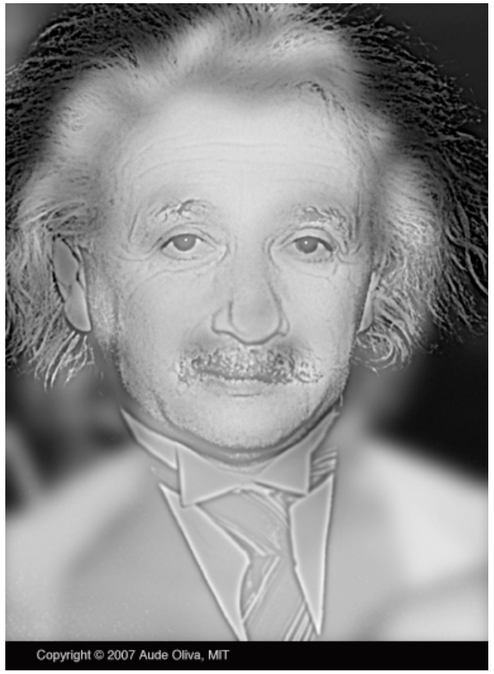
Oliva & Schyns

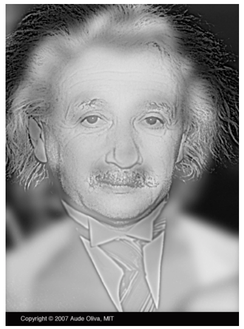
Hybrid Images



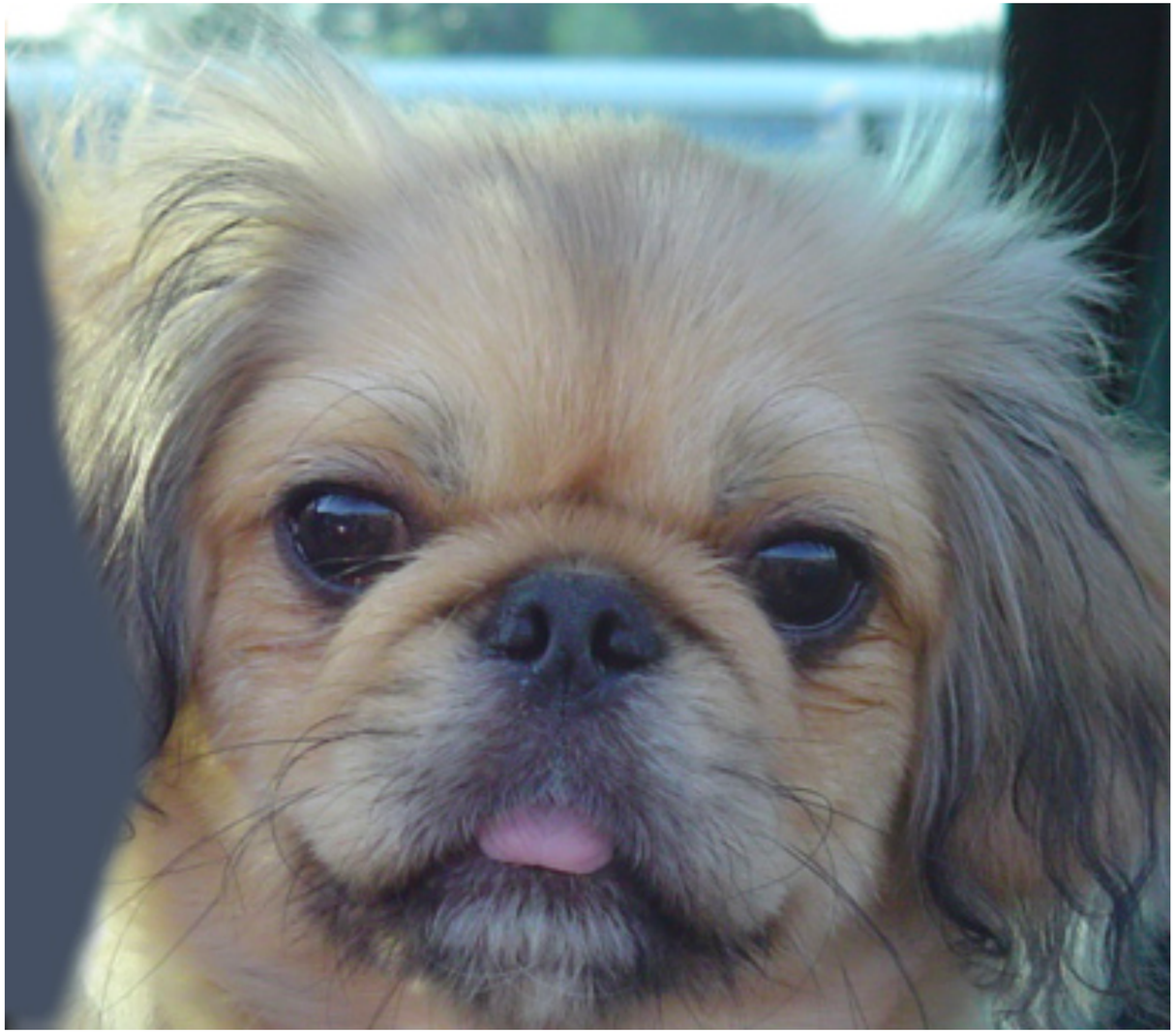
http://cvcl.mit.edu/hybrid_gallery/gallery.html







http://cvcl.mit.edu/hybrid_gallery/gallery.html



Today

- Linear filtering
- More neighborhood filters
- **Nonlinear filters**

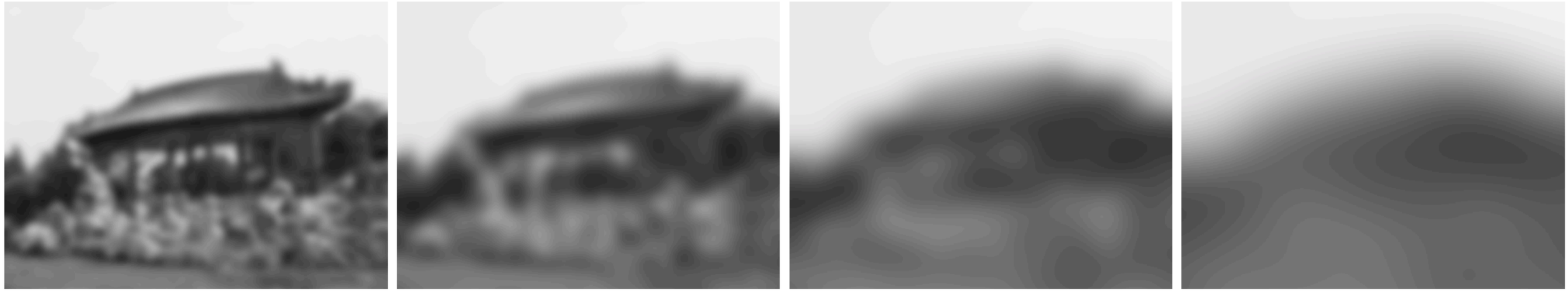
Denoising revisited: salt and pepper noise

Gaussian filter doesn’t work as well!

Source: S. Lazebnik

Blurring

Gaussian filter



$\sigma = 4$ $\sigma = 8$ $\sigma = 16$ $\sigma = 32$

Image filters aren't "aware" of edges

Images from [Paris et al. "A Gentle Introduction to Bilateral Filtering and its Applications", 2008]

Median filtering

$$MB[\mathbf{p}] = \underset{\mathbf{q}\in\mathcal{N}}{\text{median}}\ I[\mathbf{q}]$$

Median filtered result

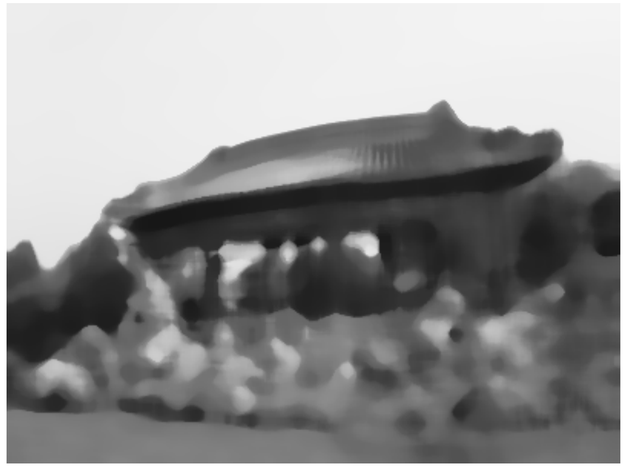
Other pixels in a window

Intensity of pixel

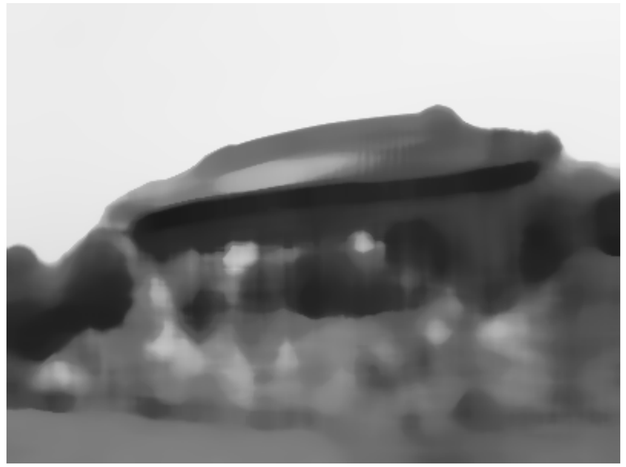
Median filtering window sizes

Original

w = 5

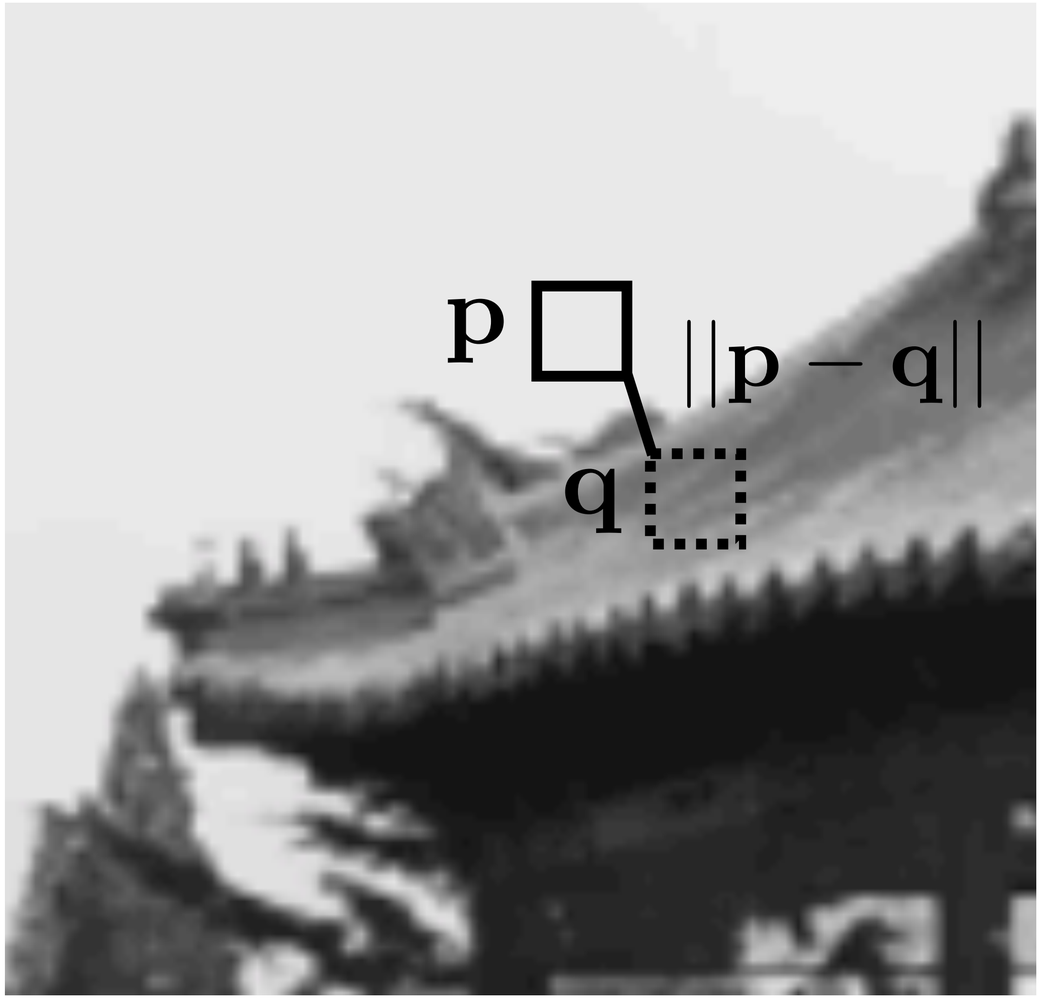


w = 13



w = 21

Another approach



Let's start by rewriting the Gaussian filter:

$$GB[\mathbf{p}] = \sum_{\mathbf{q} \in \mathcal{N}} G_\sigma(||\mathbf{p} - \mathbf{q}||) I[\mathbf{q}]$$

Blurred intensity

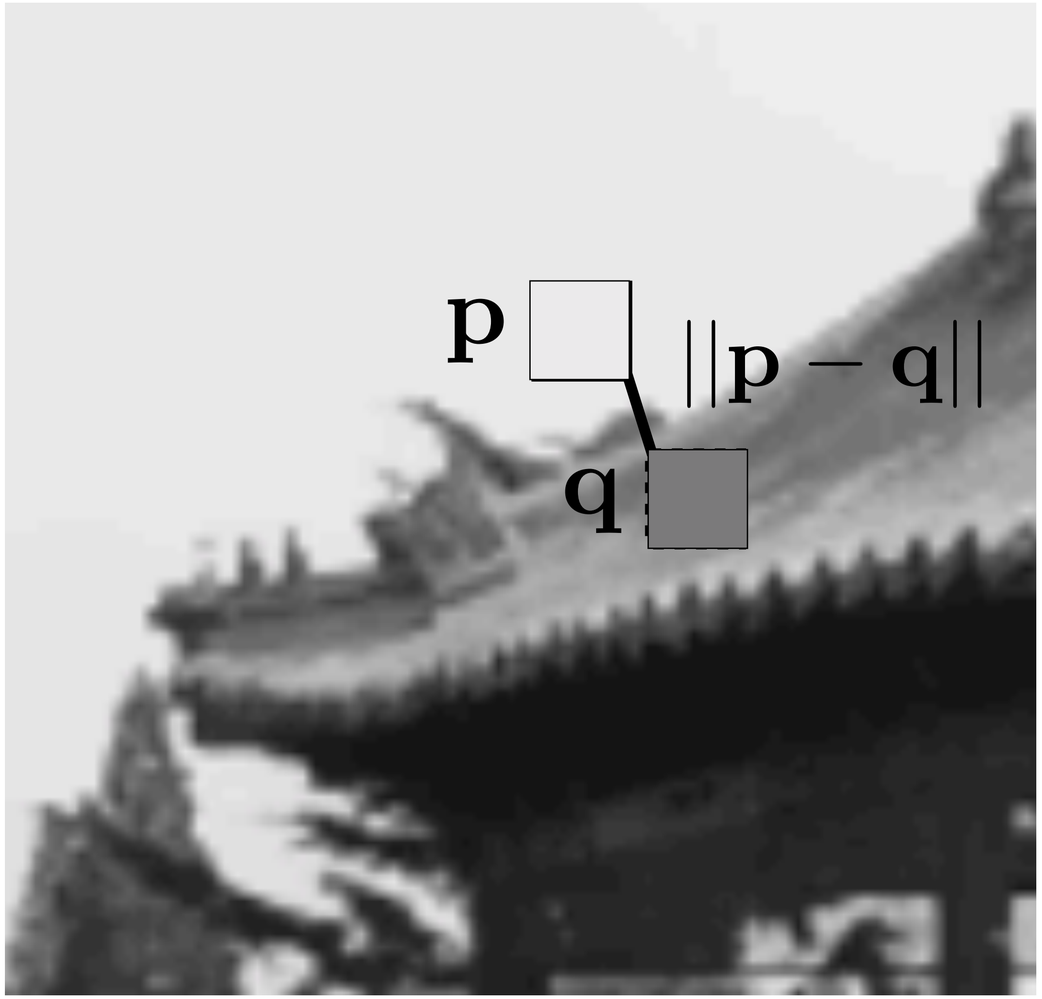
Other pixels

Intensity

Gaussian density:

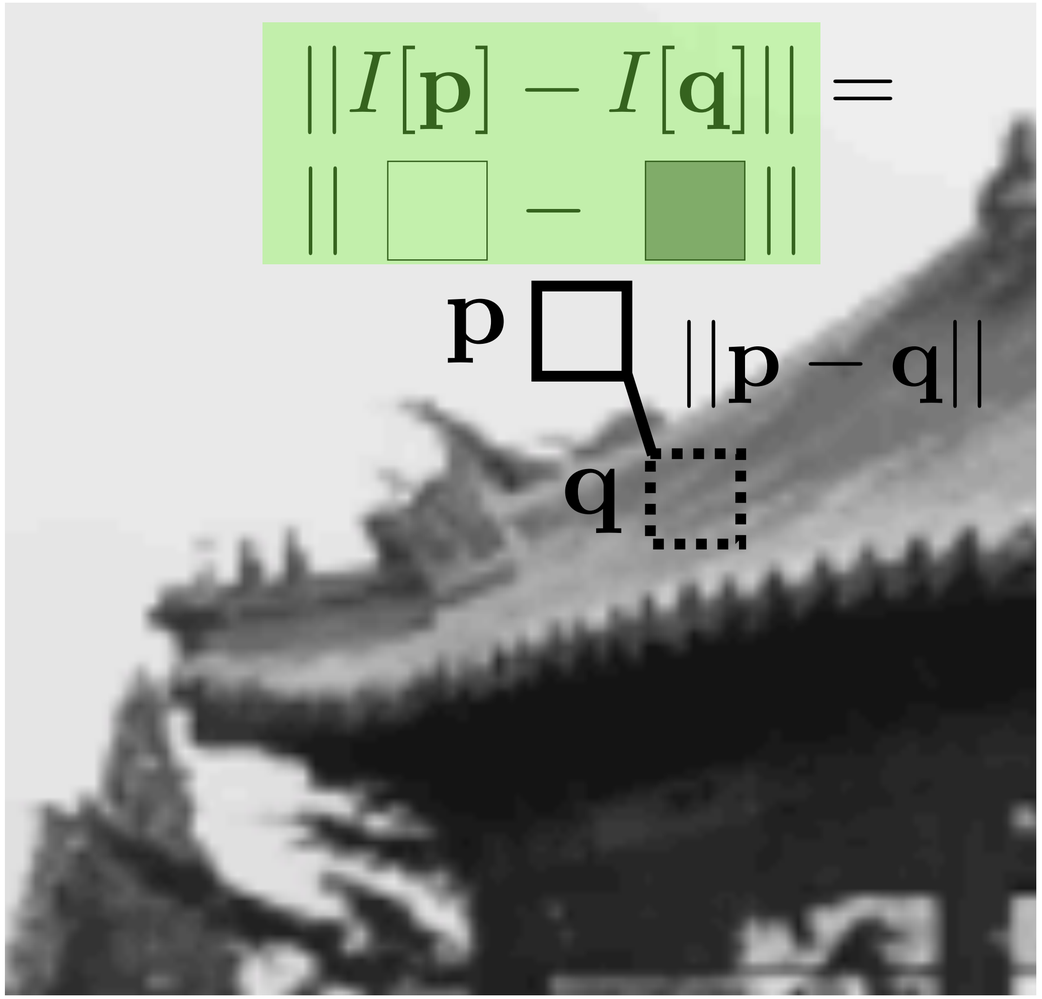
$$G_\sigma(d) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{d^2}{2\sigma}\right)$$

Bilateral filtering



What if we weight by **appearance**?

Bilateral filtering



Gaussian filter:

$$GB[\mathbf{p}] = \sum_{\mathbf{q} \in \mathcal{N}} G_\sigma(||\mathbf{p} - \mathbf{q}||) I[\mathbf{q}]$$

Bilateral filter:

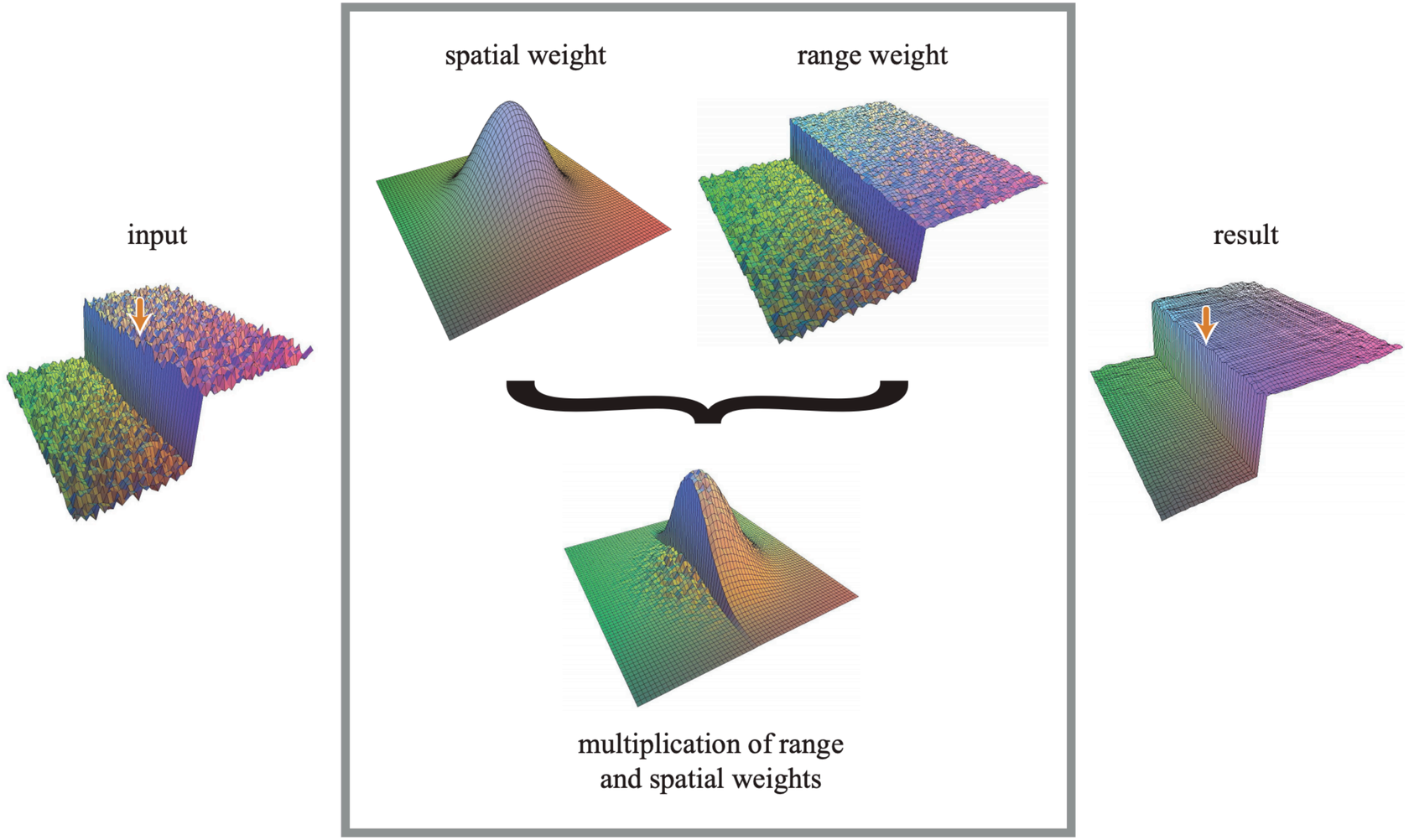
$$BB[\mathbf{p}] = \frac{1}{W_p} \sum_{\mathbf{q} \in \mathcal{N}} G_{\sigma_s}(||\mathbf{p} - \mathbf{q}||) G_{\sigma_r}(||I[\mathbf{p}] - I[\mathbf{q}]||) I[\mathbf{q}]$$

Spatial Gaussian

Range Gaussian

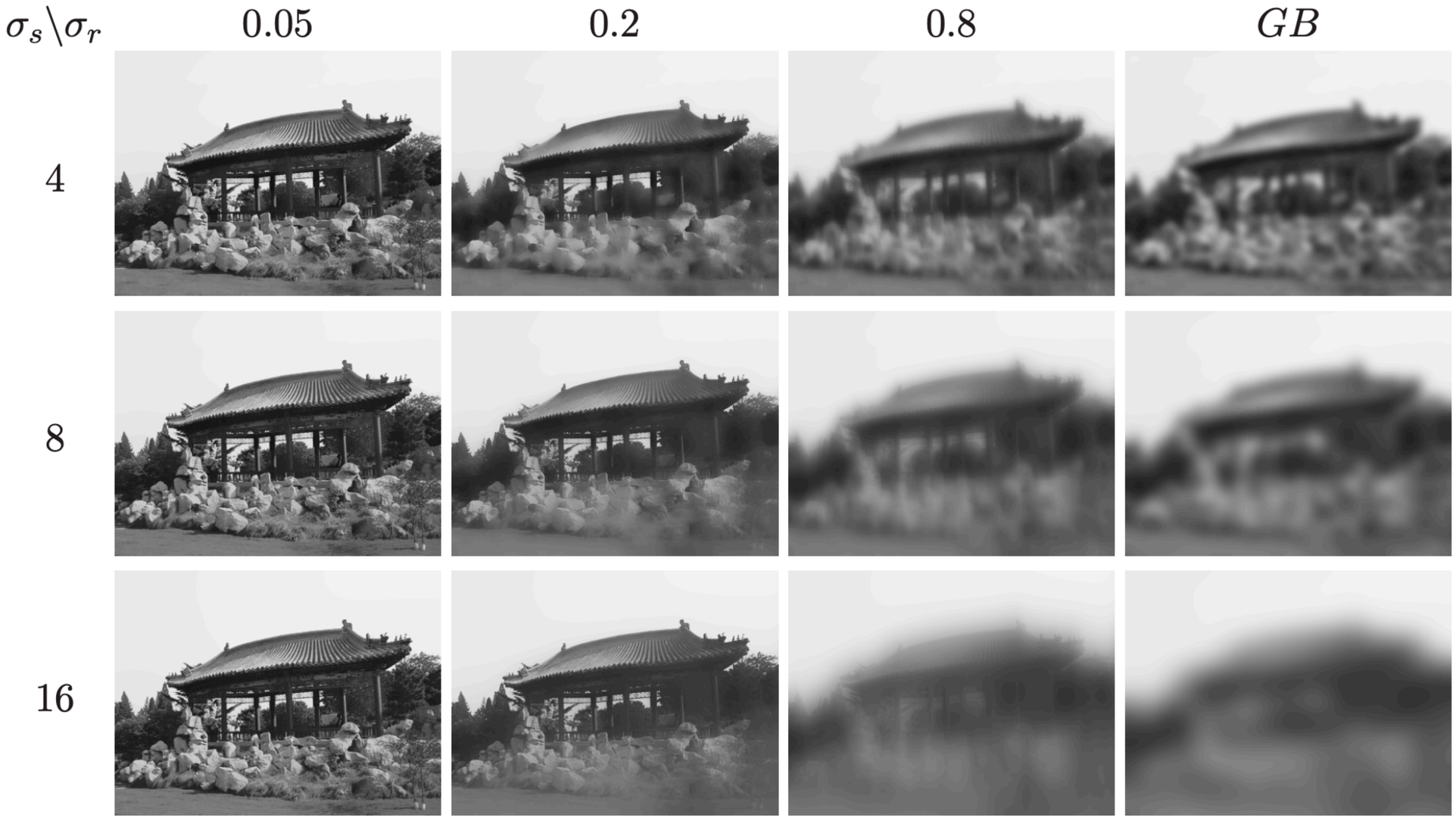
Normalization constant
(to make weights sum to 1)

Bilateral filter



[Paris et al. “A Gentle Introduction to Bilateral Filtering and its Applications”, 2008]

Bilateral filter



[Paris et al. “A Gentle Introduction to Bilateral Filtering and its Applications”, 2008]

Denoising with Gaussian

Noisy image

Gaussian blur $\sigma = 3$

Gaussian blur $\sigma = 5$

Source: [Paris et al. 2008]

Denoising with Bilateral Filter

Original

Noisy image

Bilateral filtering

Source: [Paris et al. 2008]

Iteratively applying the filter

1 iteration

2 iteration

4 iteration

Source: [Paris et al. 2008]

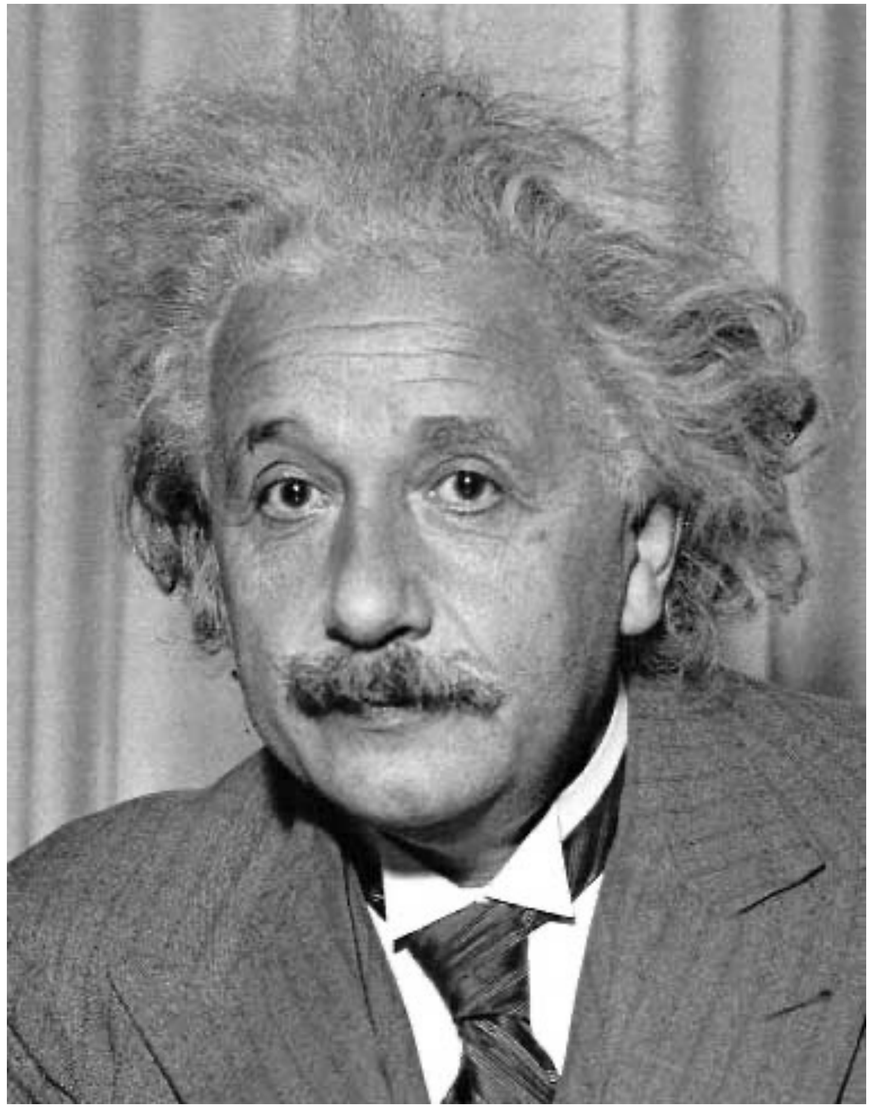
Iteratively applying the filter

“Cartoonifying” a video

[Winnemoller et al., 2006]

Image patches as filters

- Goal: find  in an image
- What's a good similarity/distance measure between two patches?



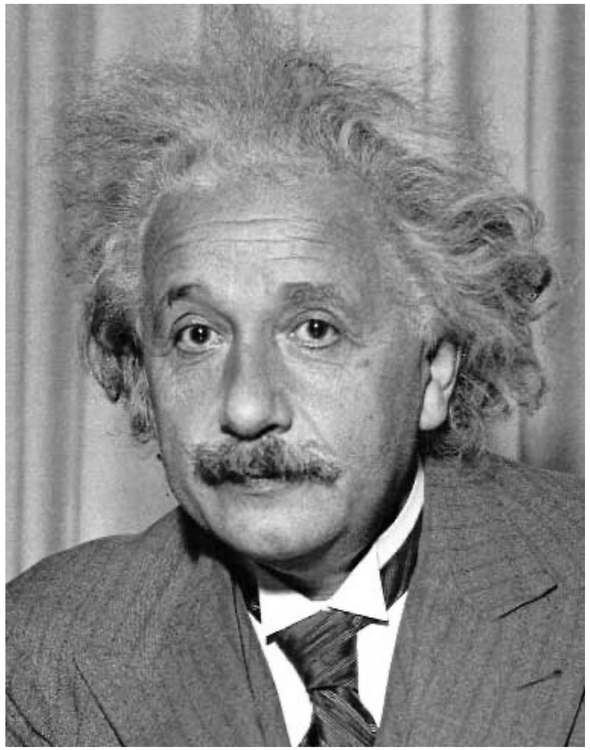
Source: D. Hoiem

Matching with filters

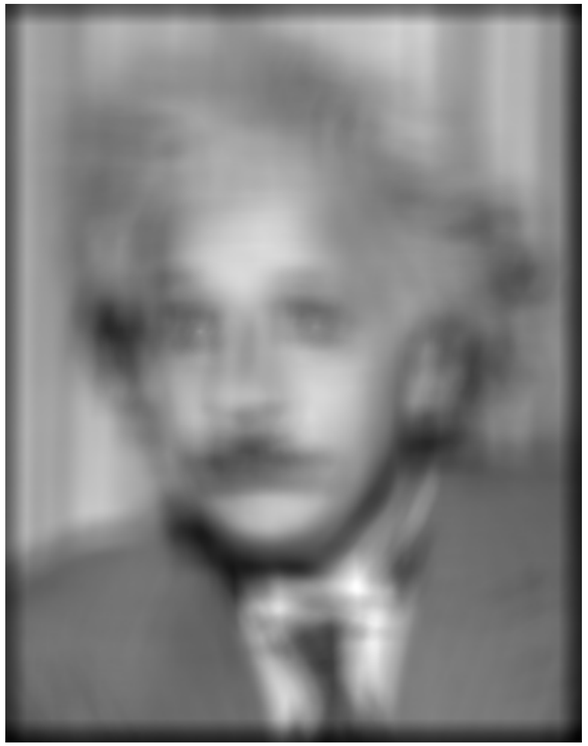
Method 1: Filter the image with 

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

f = image
g = filter



Input



Filtered Image

What went wrong?

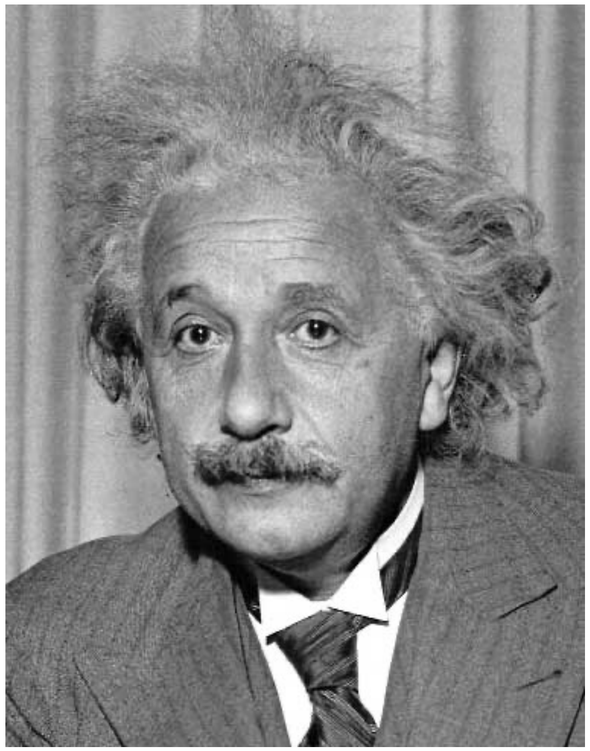
Source: D. Hoiem

Matching with filters

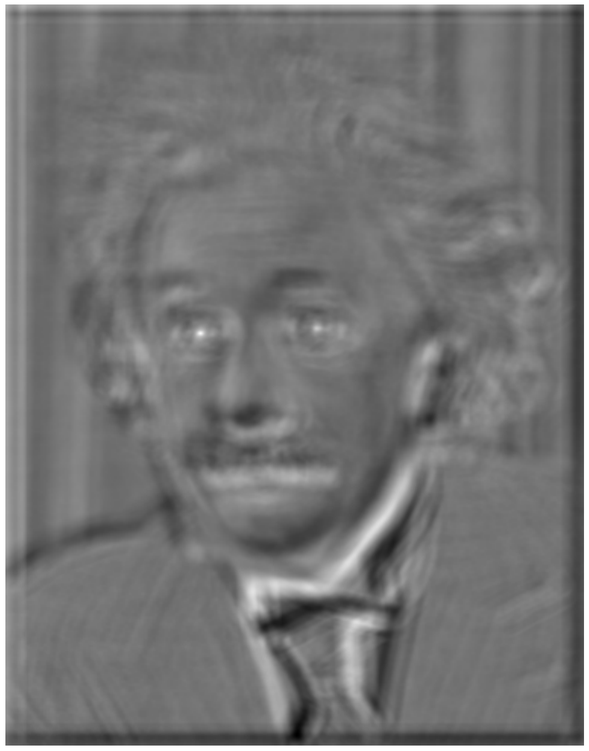
Method 2: Filter the image with zero-mean eye.

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

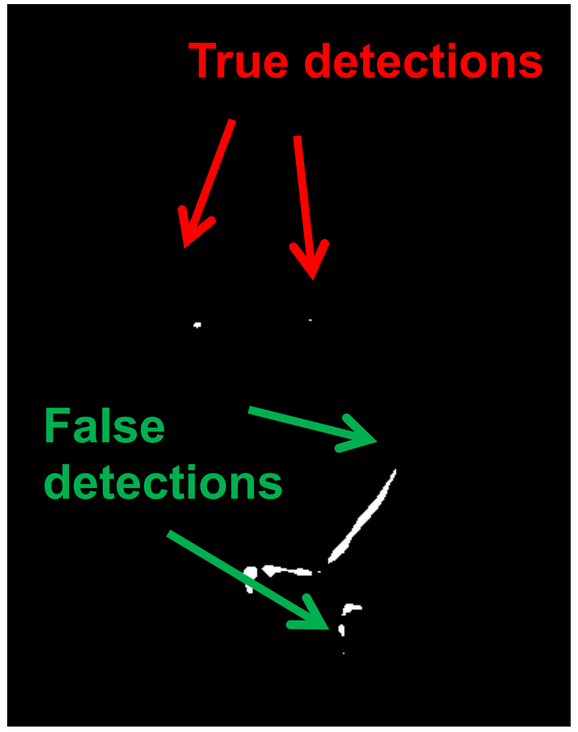
mean of f



Input



Filtered Image (scaled)



Thresholded Image

Source: D. Hoiem

Matching with filters

Method 3: Normalized cross-correlation.
Divide by standard deviation of both patches, so they are unit vectors.

mean template

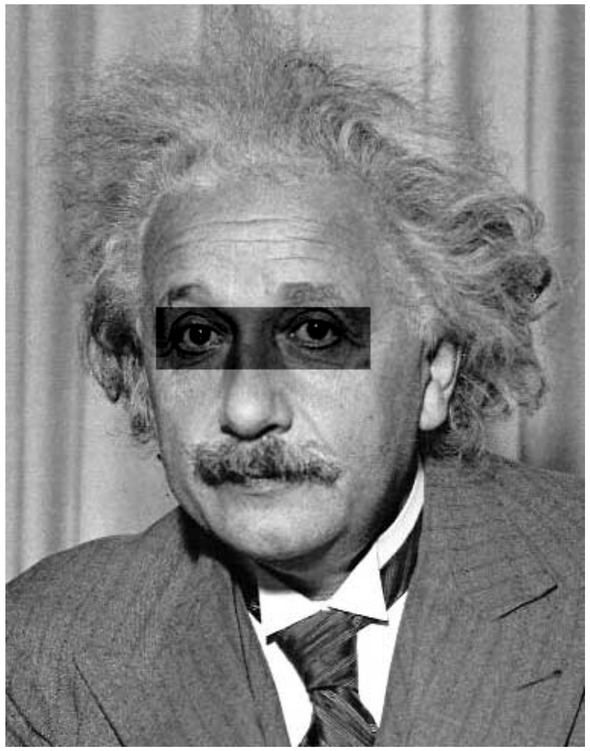
mean image patch

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

Source: D. Hoiem

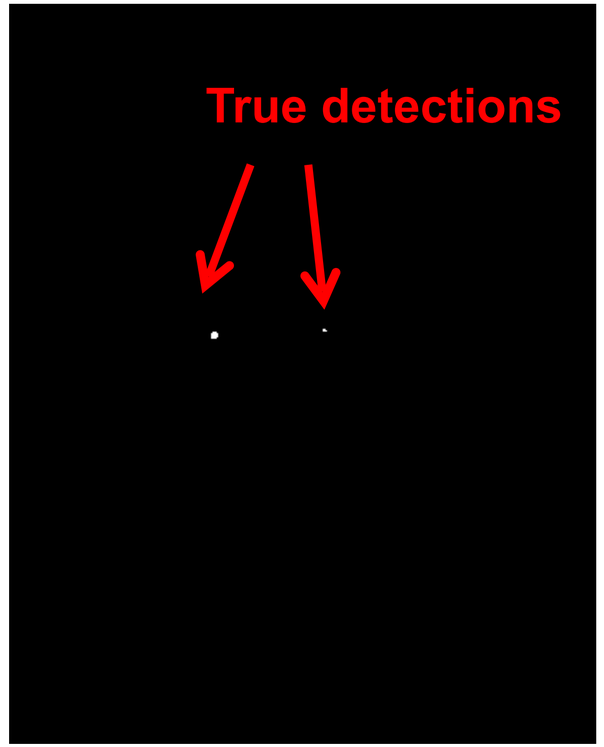
Matching with filters

Method 3: Normalized cross-correlation.



Input

Normalized X-Correlation

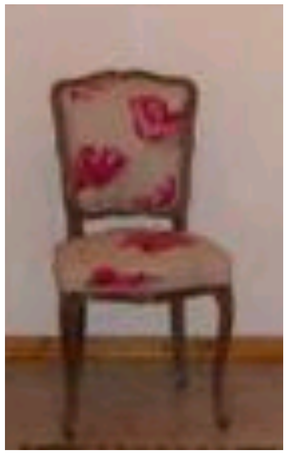


Thresholded Image

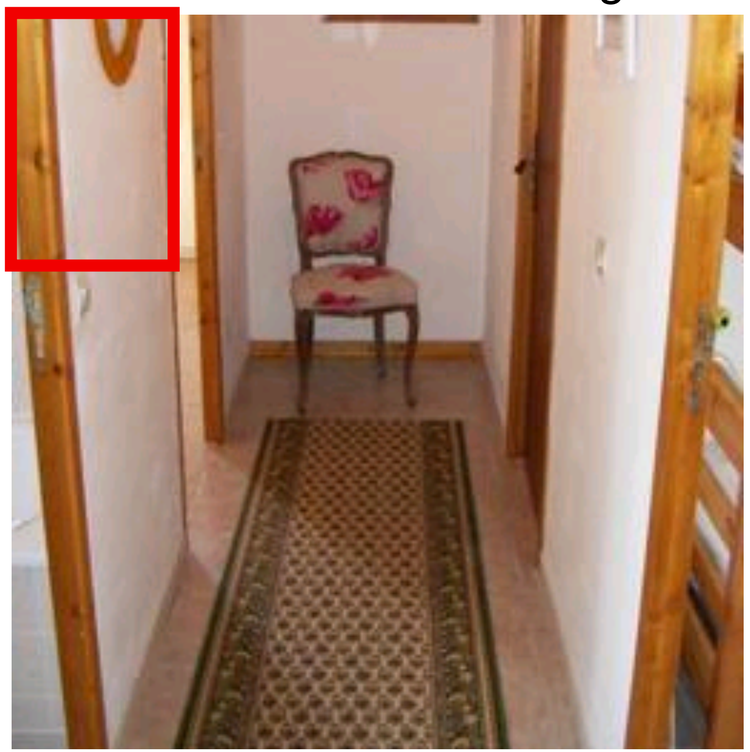
Source: D. Hoiem

Recognizing objects: is it really so hard?

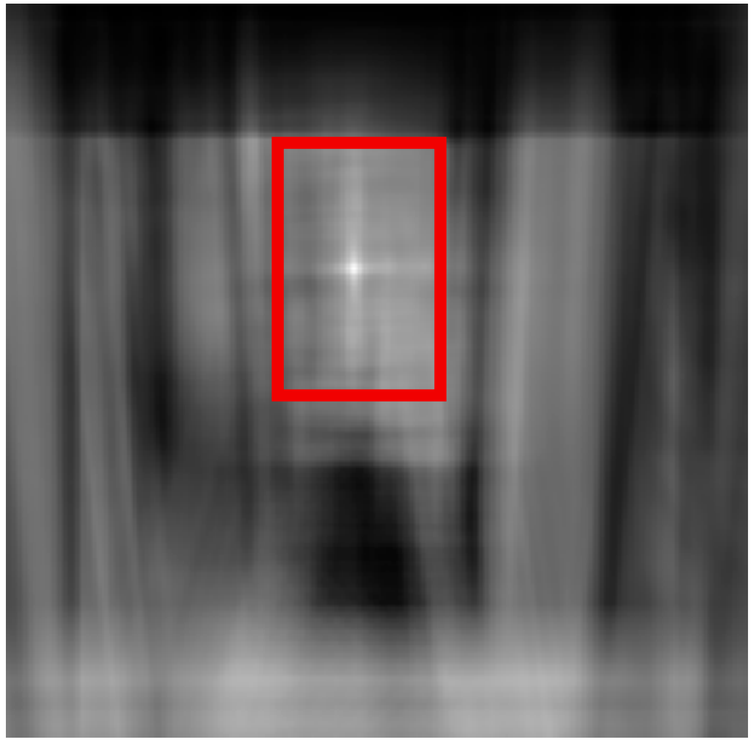
This is a chair



Find the chair in this image

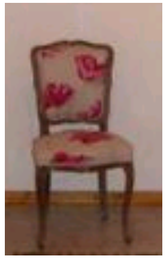


Output of normalized correlation

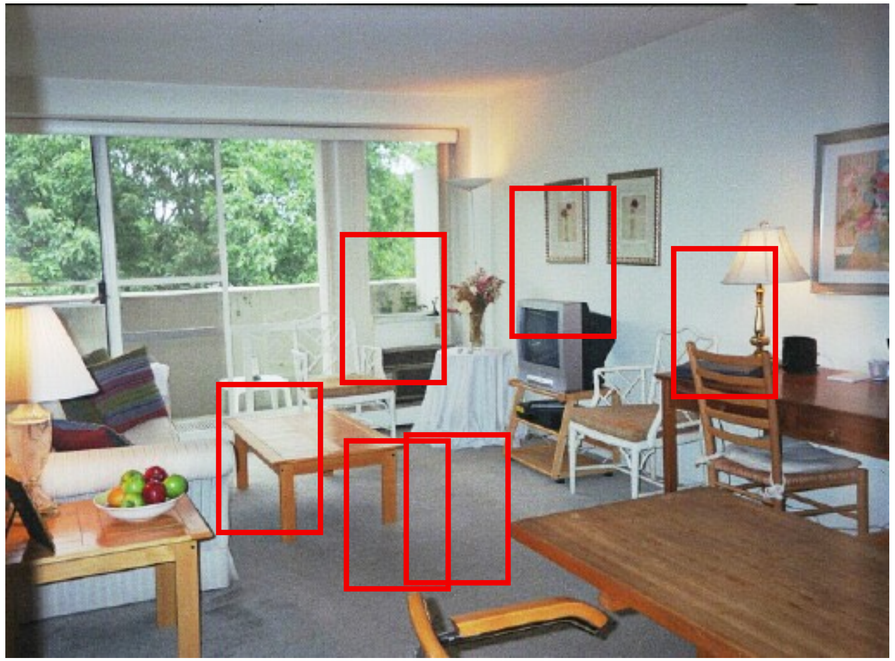


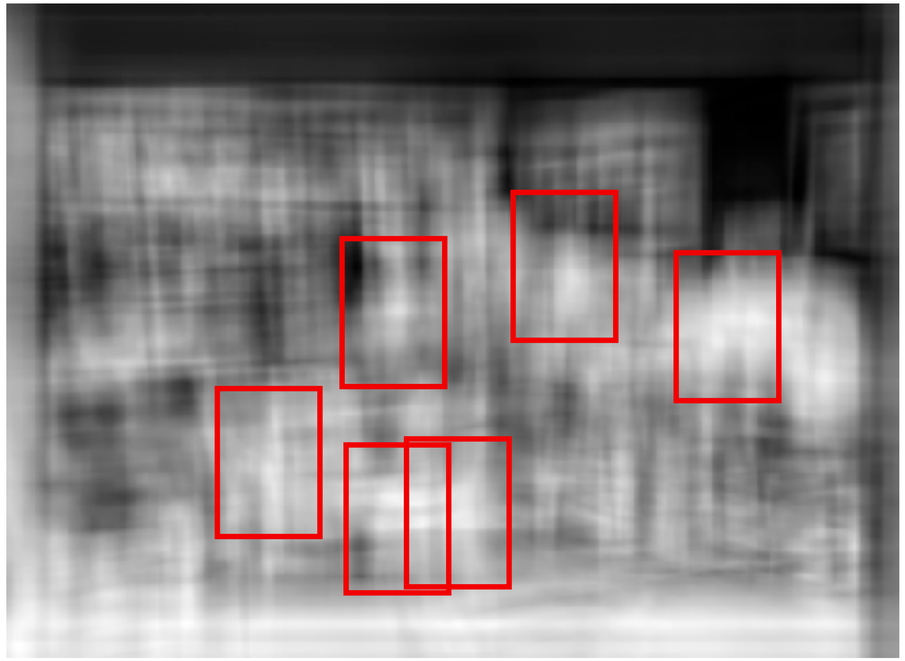
Source: A. Torralba

Recognizing objects: is it really so hard?



Find the chair in this image



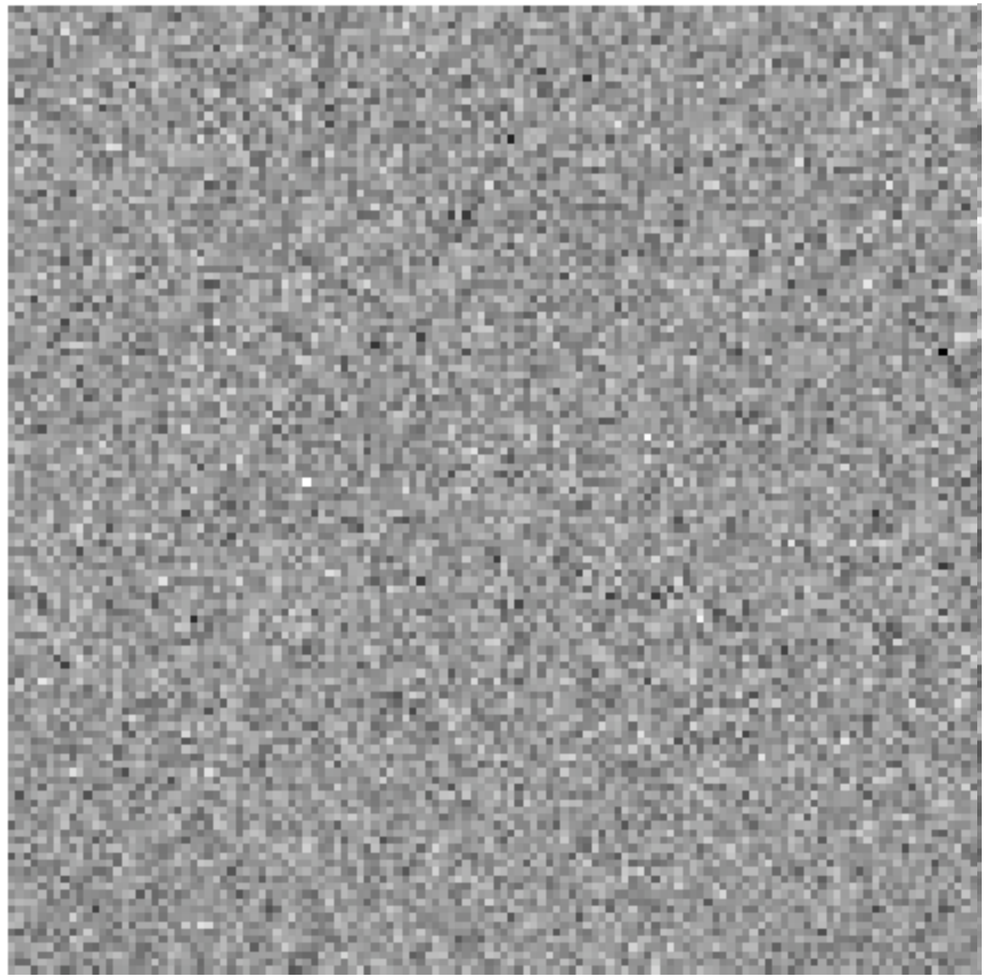


Not so great!

Source: A. Torralba

What makes an image “natural”?

Natural image

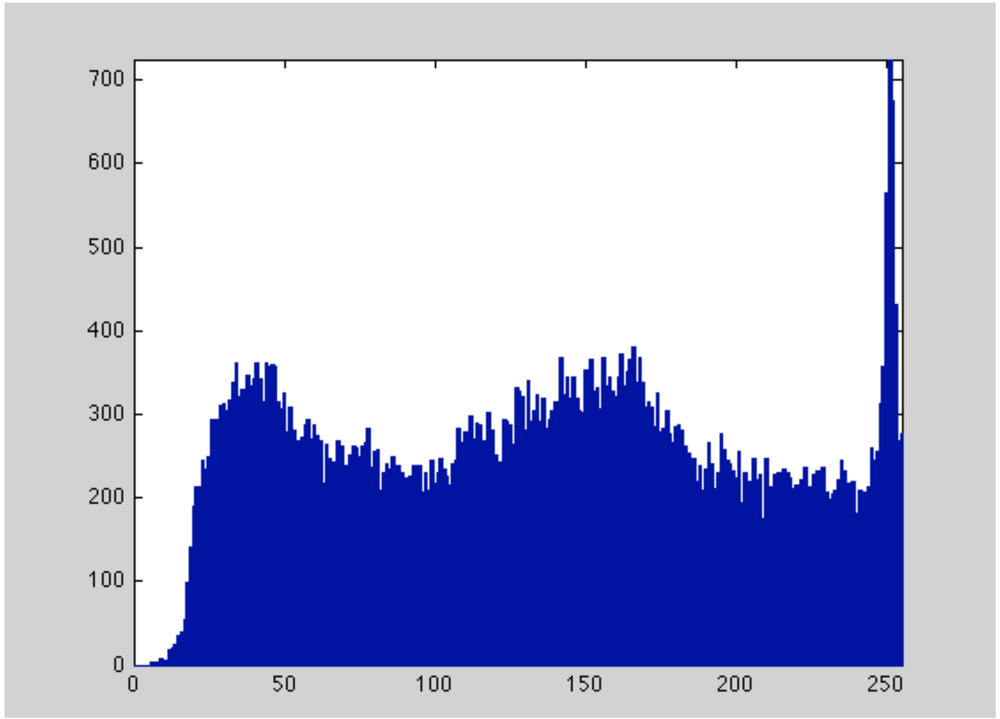


“Fake” image

Source: Torralba, Freeman, Isola

Is it the distribution of pixel intensities?

No real structure here…



Intensity histogram

What about gradients?

g[m,n]

$\otimes$

[-1, 1]

h[m,n]

=

f[m,n]

Source: Torralba, Freeman, Isola

What about gradients?

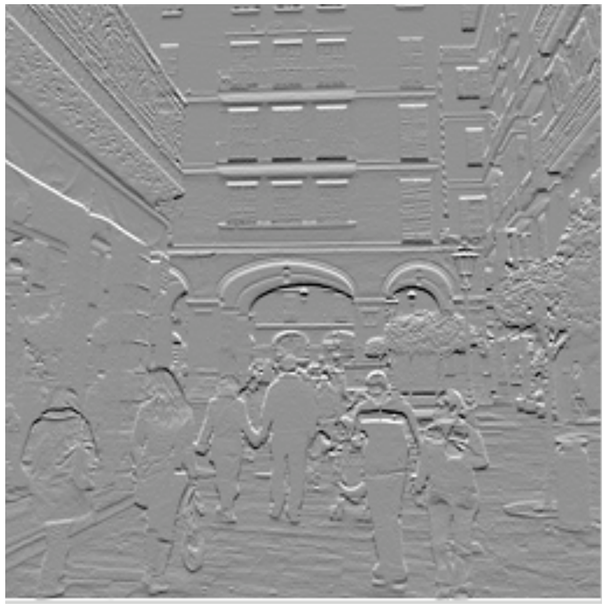
g[m,n]

$\otimes$

$[-1, 1]^T$

h[m,n]

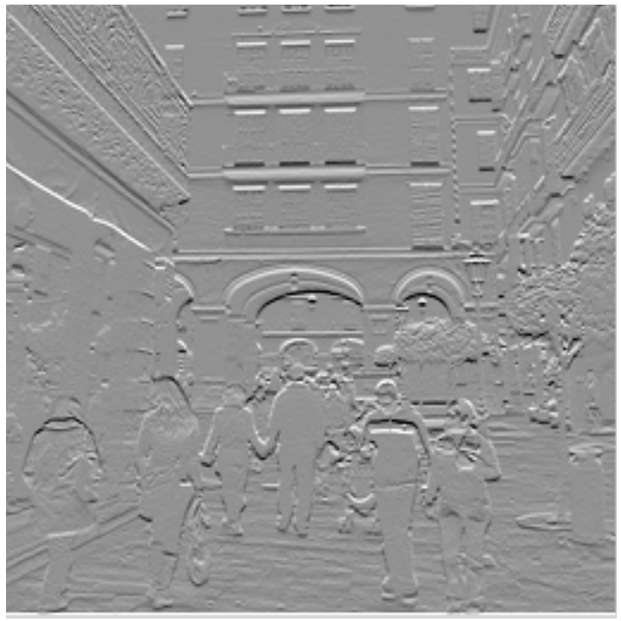
=

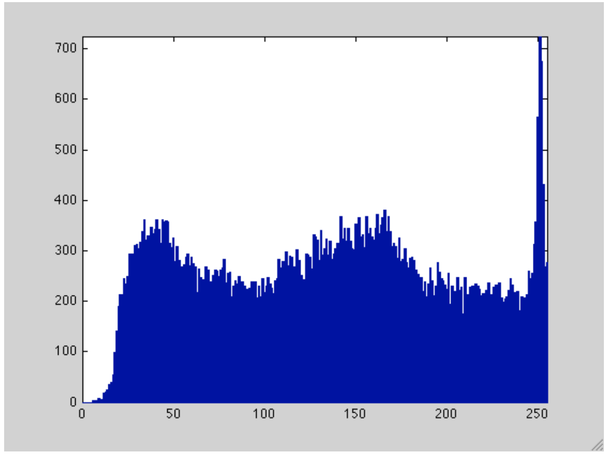


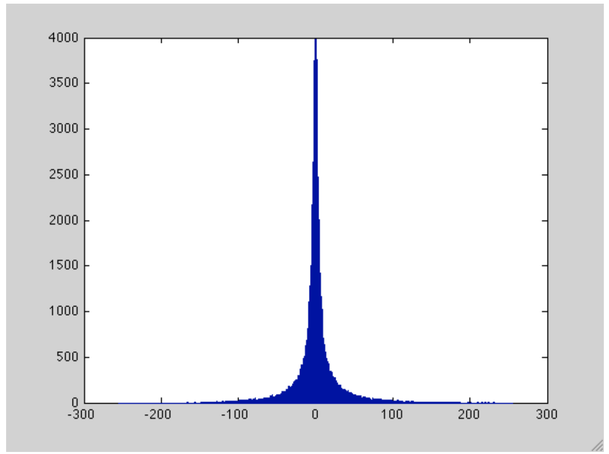
f[m,n]

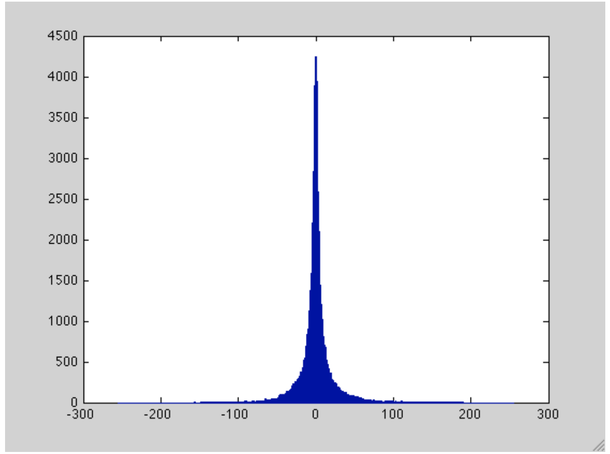
Source: Torralba, Freeman, Isola

Filter response distribution is pretty consistent!



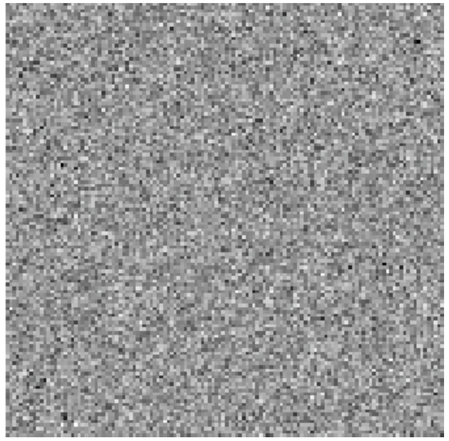


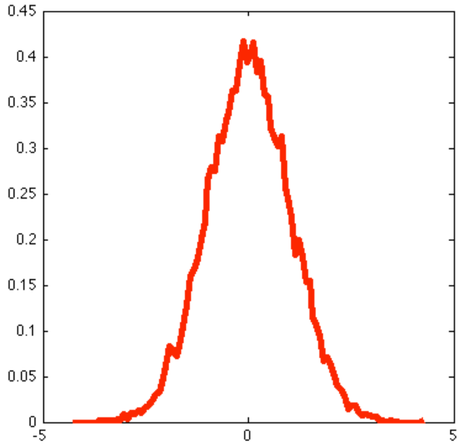


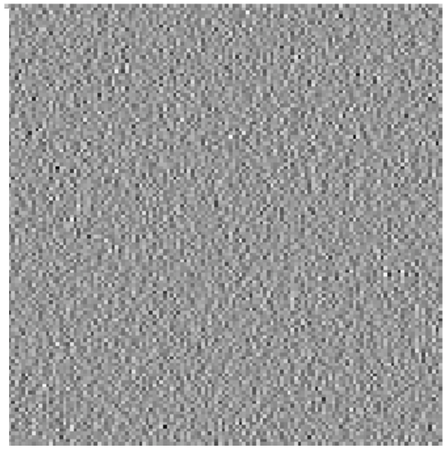


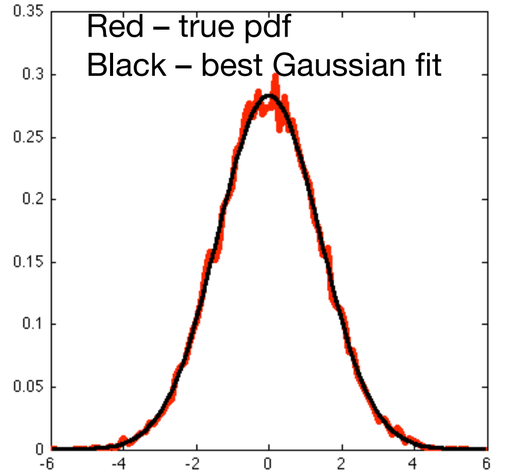
Source: Torralba, Freeman, Isola

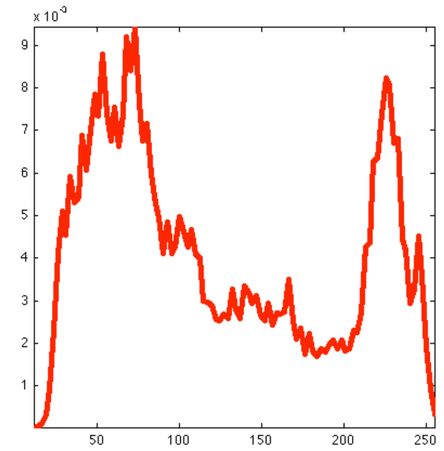
Image | Intensity histogram | [1 -1] filter output | [1 -1] output histogram

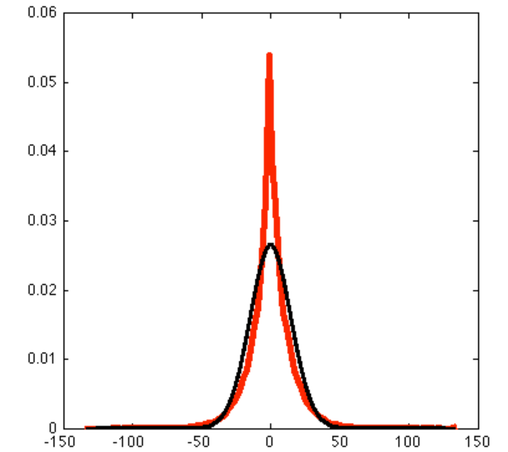


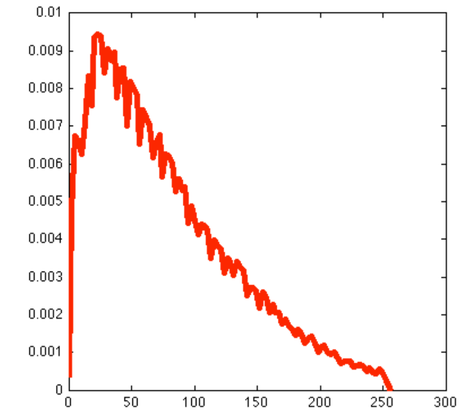


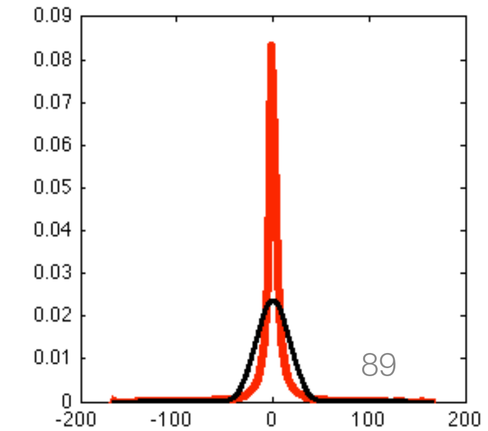










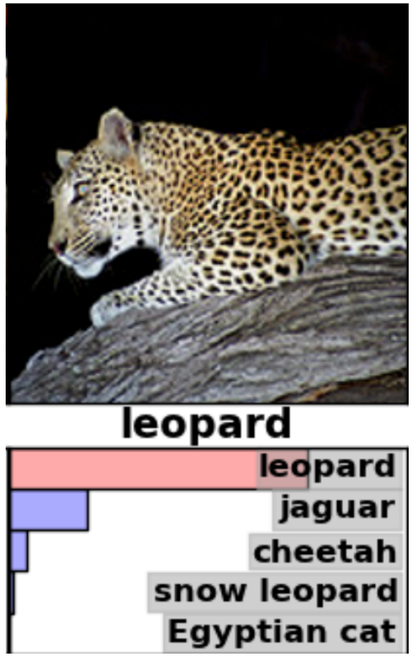


Source: Torralba, Freeman, Isola

Applications of image statistics

Compression

Image restoration



Learning
(later in course)

Taking a picture…

What the camera give us…

How do we correct this?

Slides R. Fergus

Deblurring

Slides R. Fergus

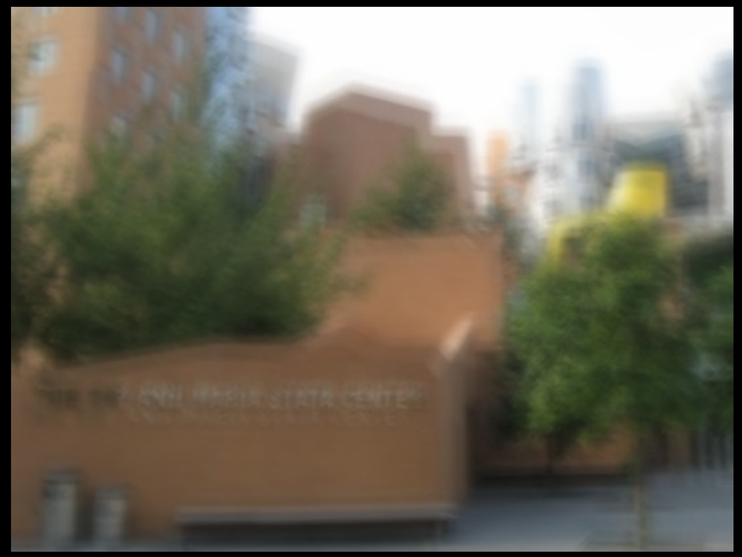
Deblurring

Slides R. Fergus

Deblurring

Slides R. Fergus

Image formation process



Blurry image

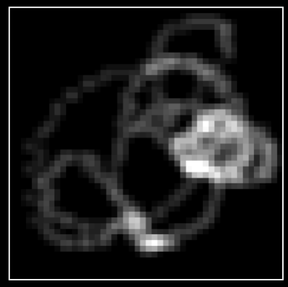
Input to algorithm

=

Sharp image

Desired output

⊗



Blur kernel

Convolution operator

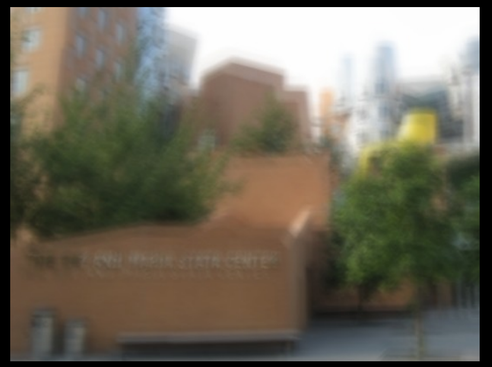
Source: R. Fergus

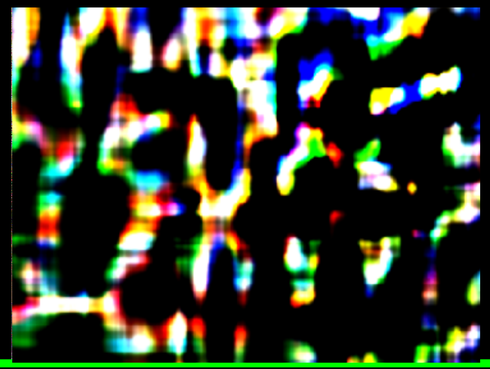
Multiple possible solutions

Blurry image

Sharp image

Blur kernel



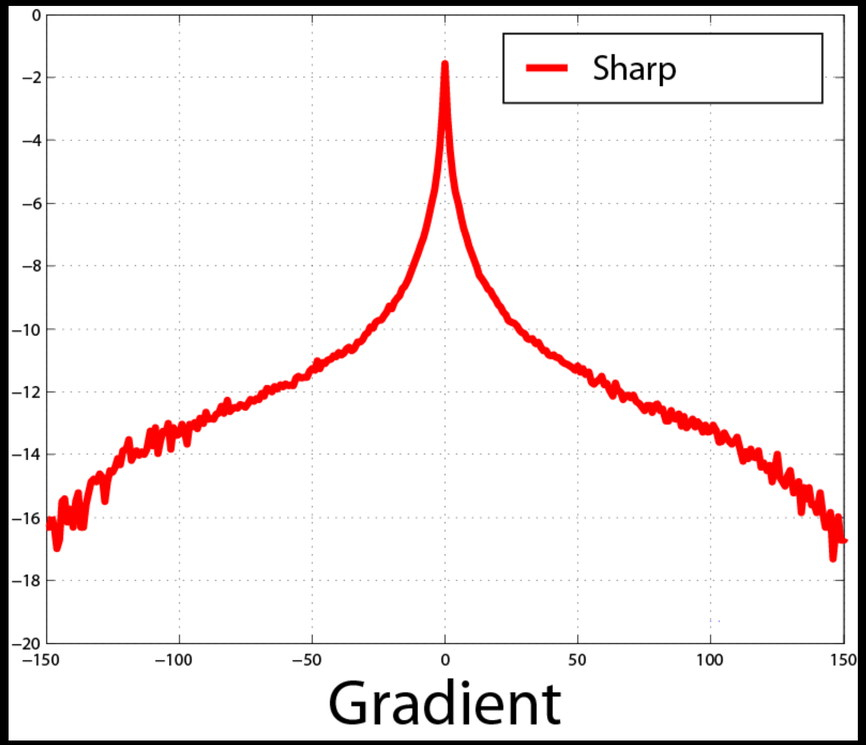


Source: R. Fergus

Natural image statistics

Characteristic distribution with heavy tails

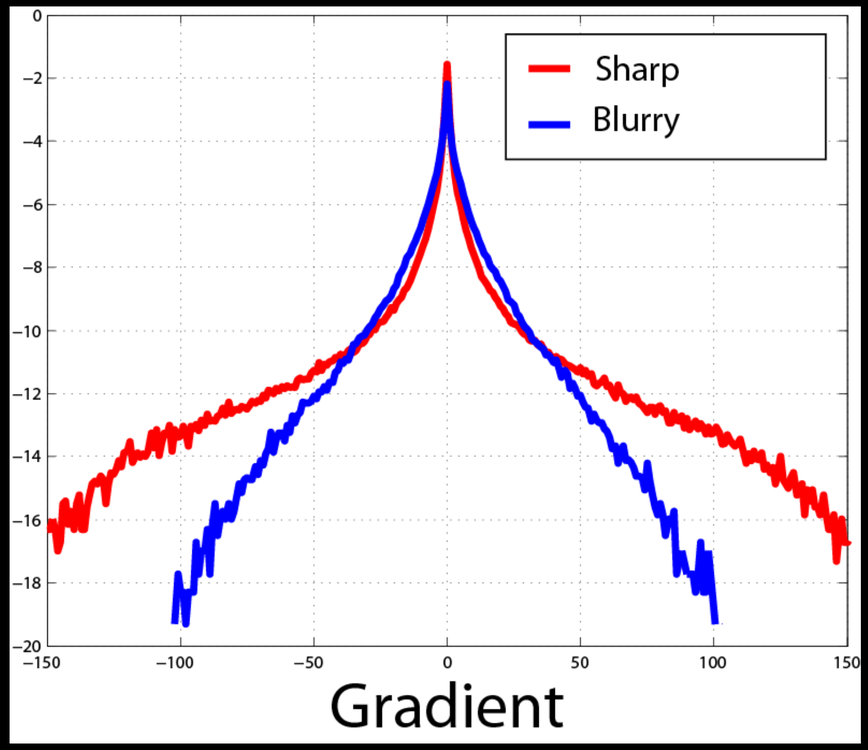
Histogram of image gradients



Source: R. Fergus

Blurry images have different statistics

Histogram of image gradients



Source: R. Fergus

Removing motion blur

Original photograph

After matching filter distribution

Solve for an image with a distribution of edge gradients that “matches” a normal image.

Source: R. Fergus

Next class: image pyramids