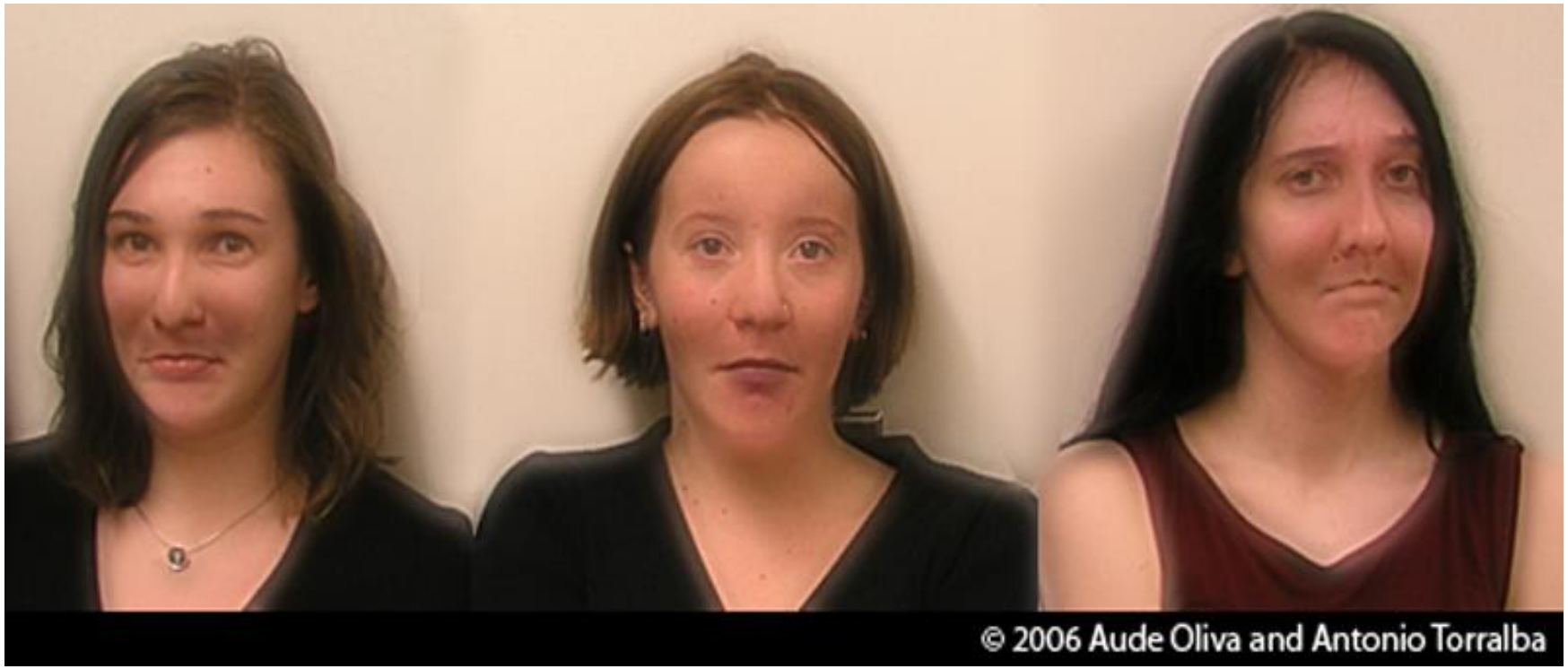
Computer Vision (10224)

Lecture 2+3: Filtering
Eyal Katz – Spring 2021

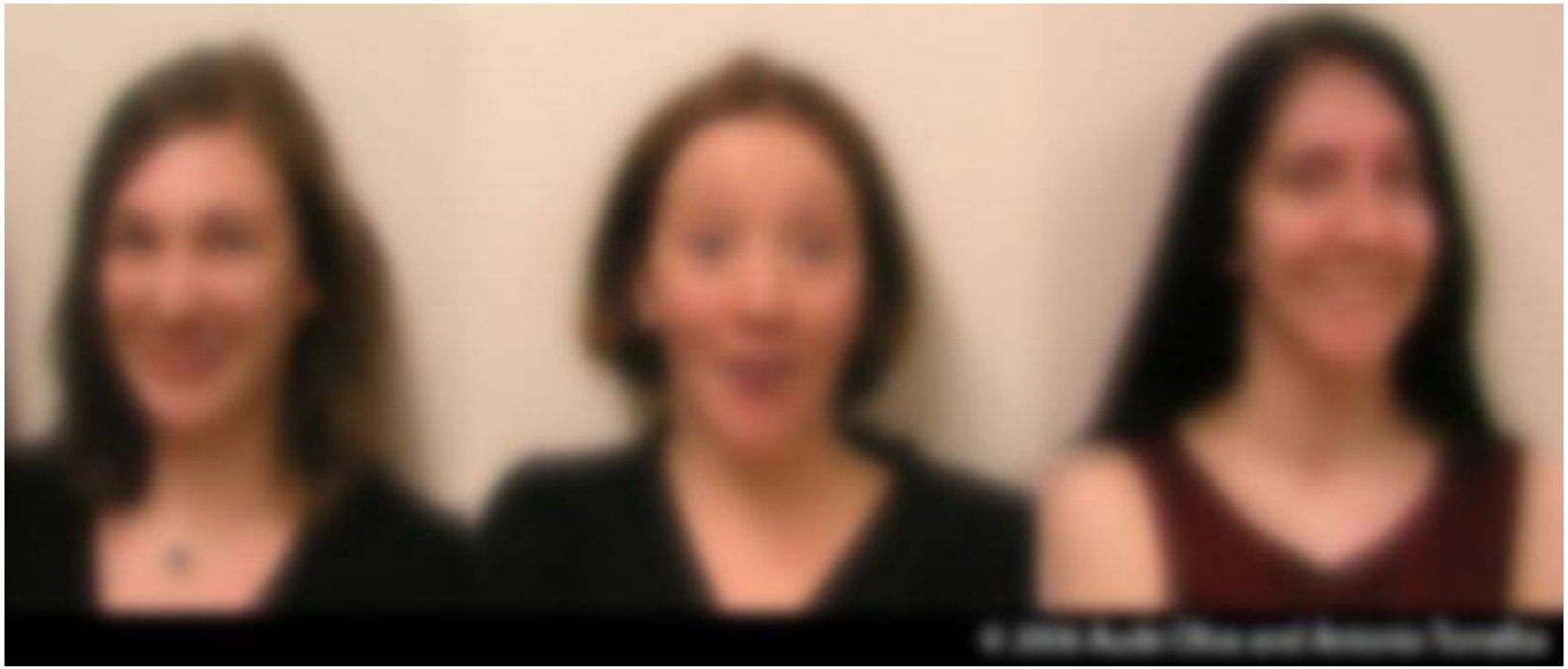
This presentation is (mainly) based on Berkeley EECS'

Prof. Trevor Darrell, Lecture 4: Filtering



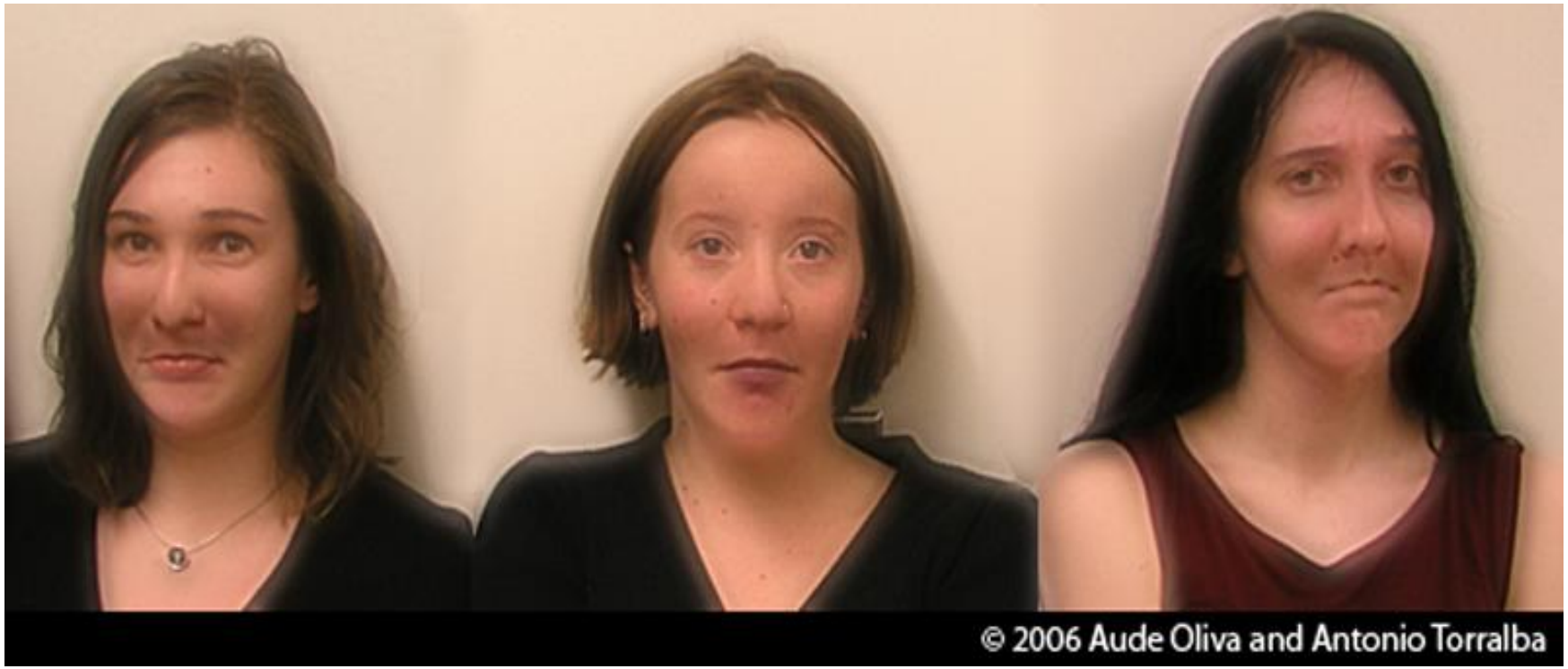
Hybrid Images, Oliva et al., http://cvcl.mit.edu/hybridimage.htm

Image filtering



Hybrid Images, Oliva et al., http://cvcl.mit.edu/hybridimage.htm

Image filtering


© 2006 Aude Oliva and Antonio Torralba

Last Time

- Course admin
- Topic Overview

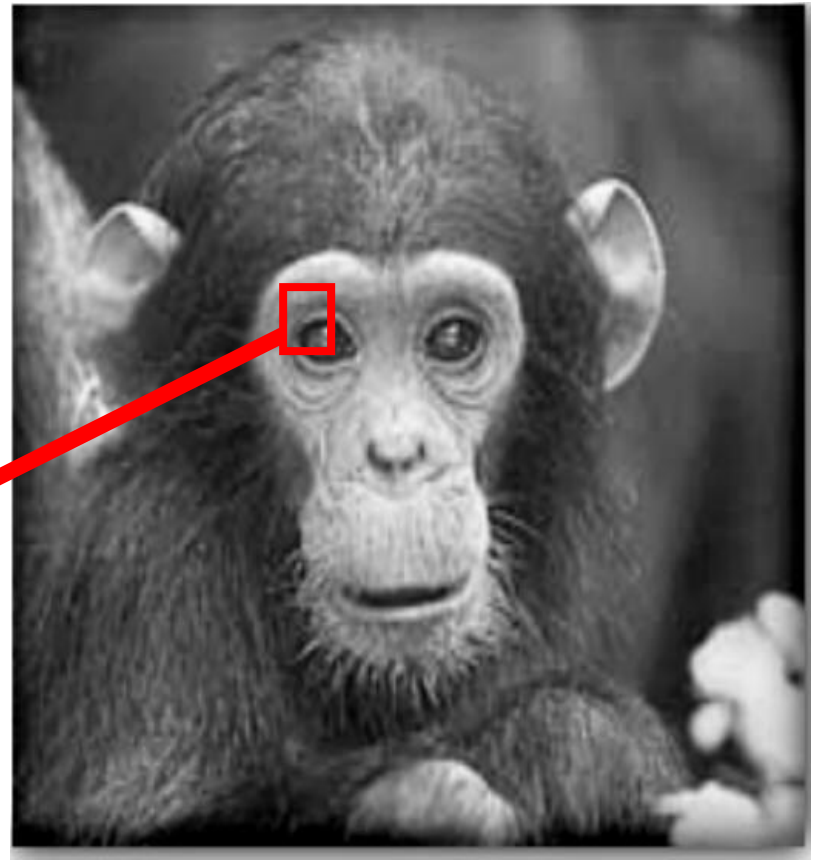
- Image Formation
- Digital image representation (Gray level, color)
- Introduction to basic image processing

Recall: Digital images as a matrix

jj=0/1 (python / matlab)

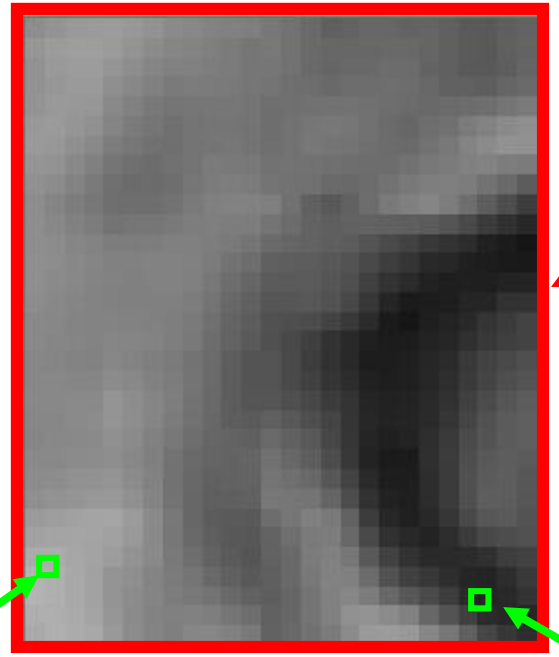
width
520

Intensity : [0,255]



ii=0
or 1

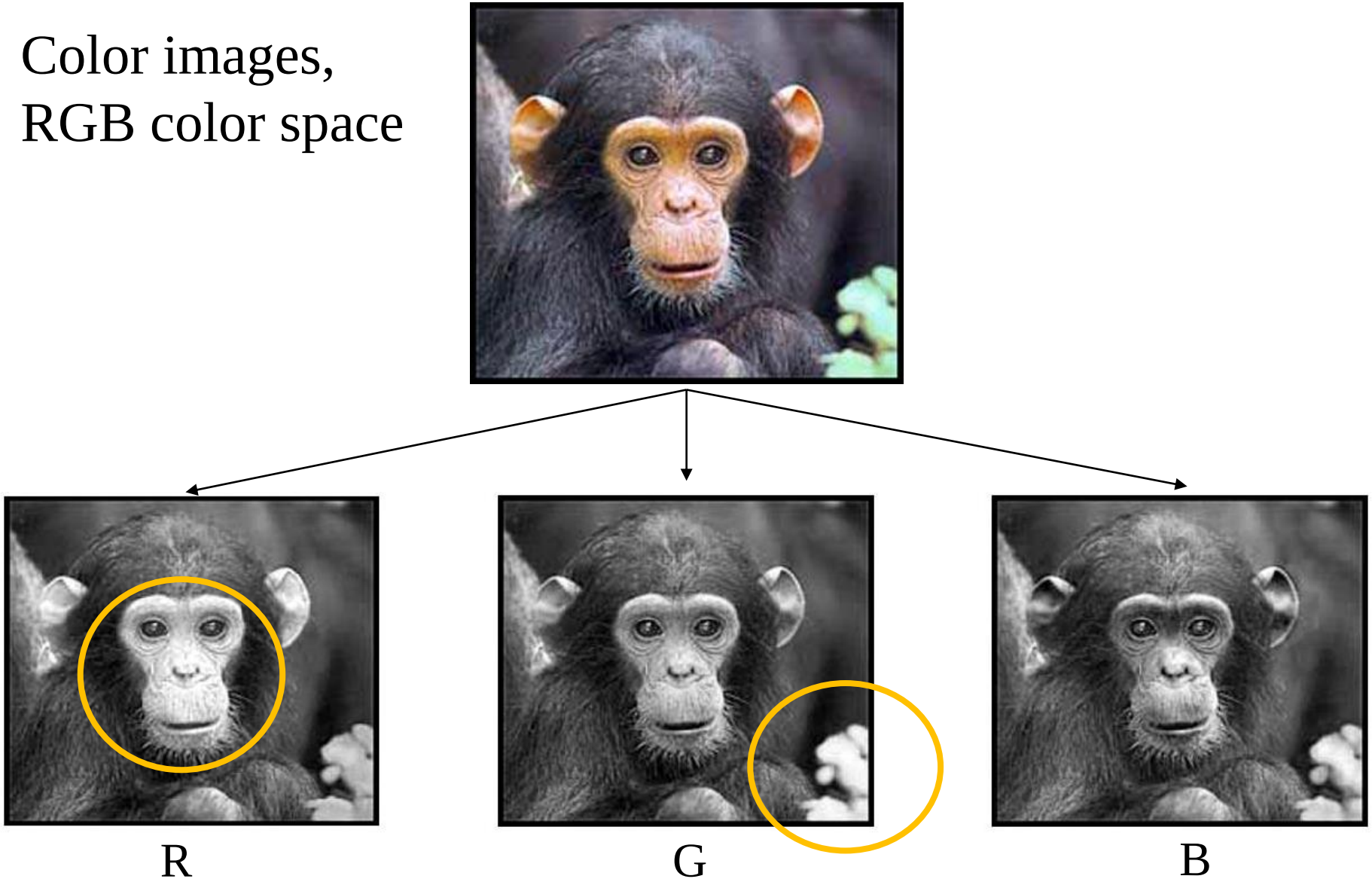
500
height



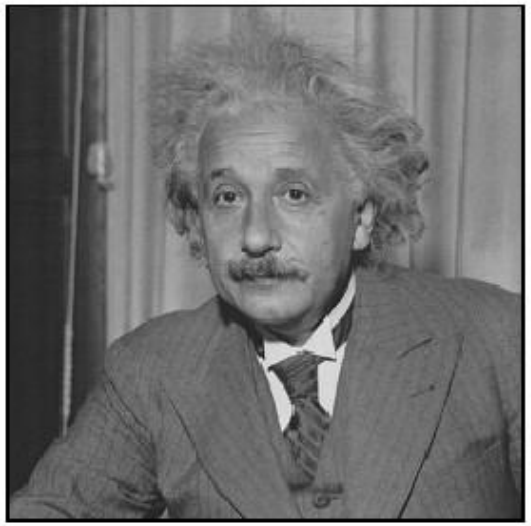
im[176][201] has value 164

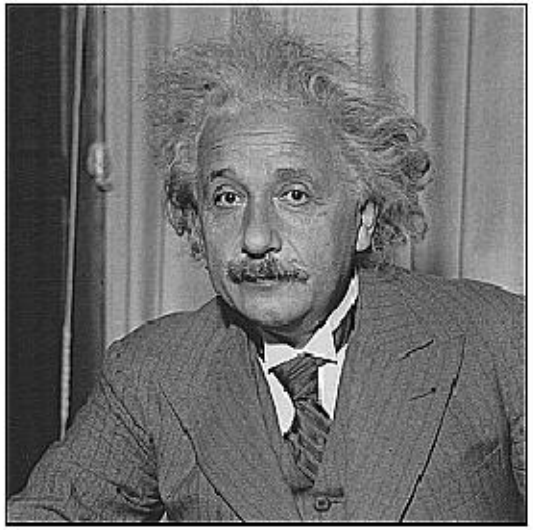
im[194][203] has value 37

Color images, RGB color space



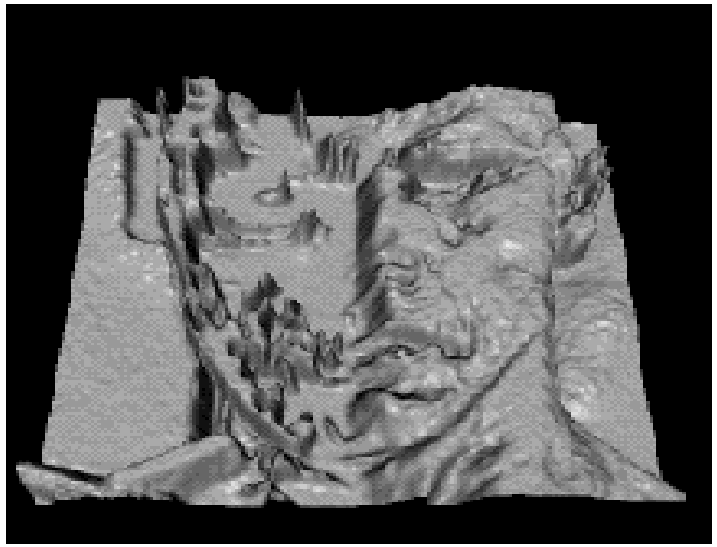
Today: Image Filters

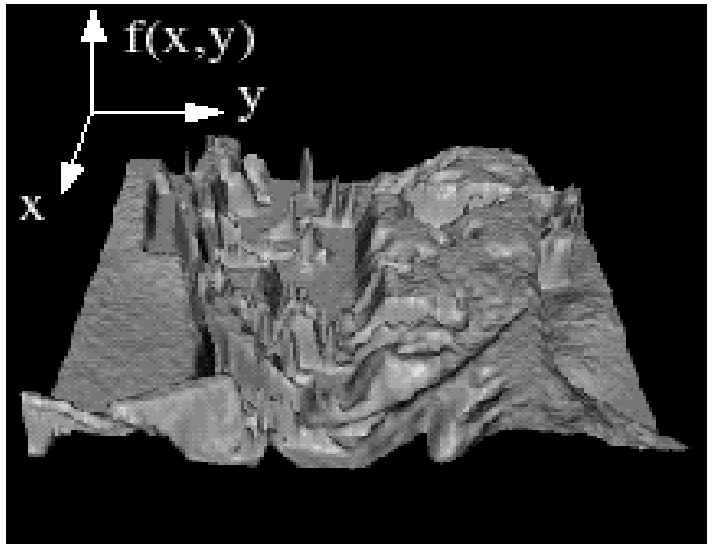




Smooth/Sharpen Images... ***Find edges...*** ***Find waldo…***

Images as functions

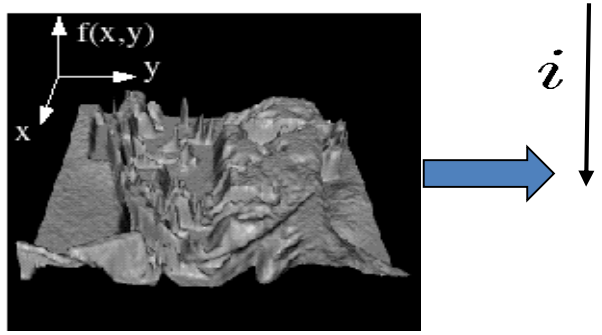




Source: S. Seitz

Digital images

- In computer vision we operate on **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.

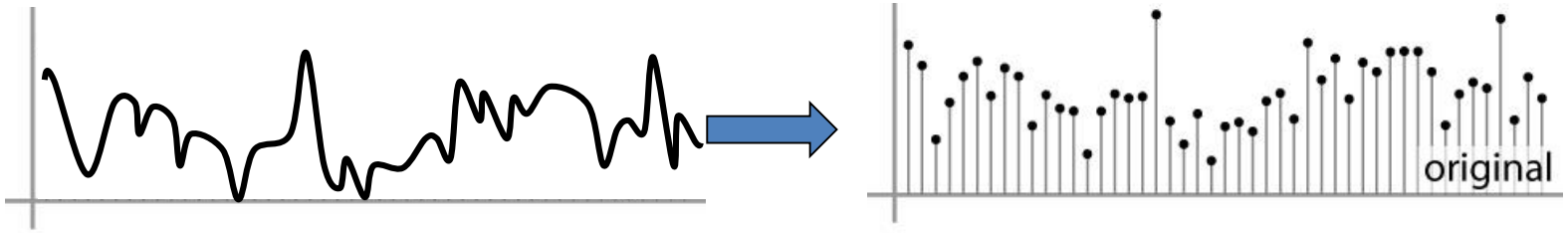


j

i

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D



1D

Images as functions

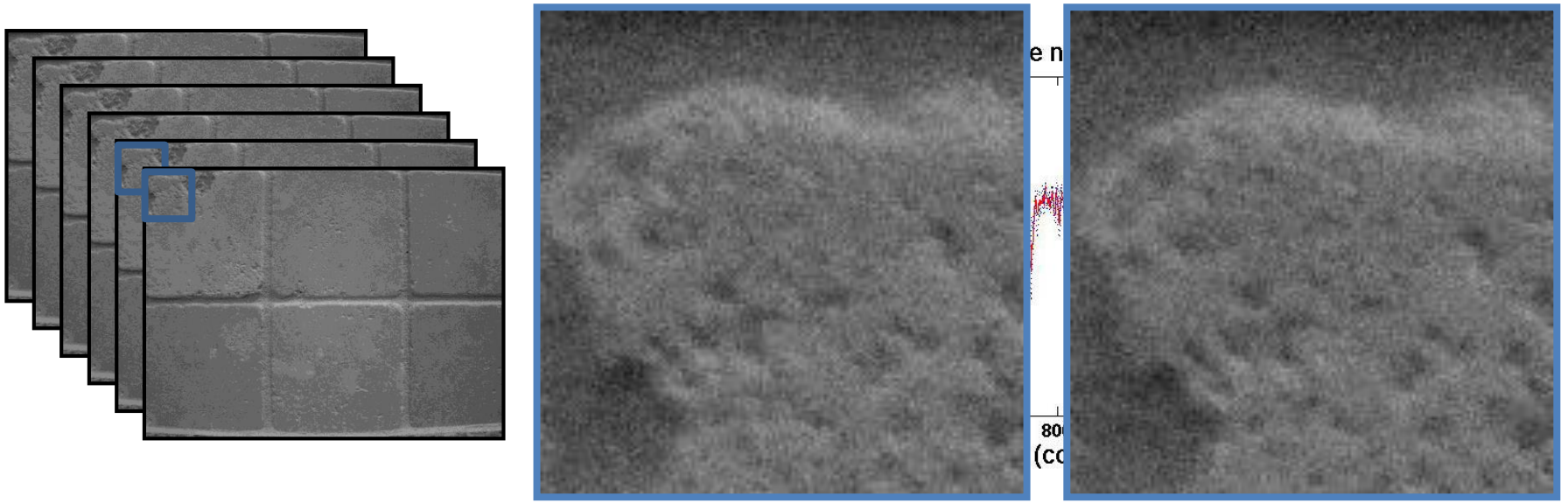
- We can think of an image as a function, *f*, from R^2 to R:
  - $f(x, y)$ gives the intensity at position (x, y)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - $f: [a,b] \times [c,d] \rightarrow [0, 1.0]$

- A color image is just three functions pasted together. We can write this as a "vector-valued" function:

$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

Source: S. Seitz

Motivation: noise reduction



- We can measure **noise** in multiple images of the same static scene. (Assume original niseless image is I, and noisy images are $I_1, \ldots, I_N$)
- **How could we reduce the noise, i.e., give an estimate of the true intensities?** (in other words, we wish to reconstruct I from the noisy images $I_1, \ldots, I_N$ into a new image, I_R)
  Write a mathematical expression for each pixel of $I_R(x,y)$:

Common types of noise

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution
  (this is **Additive Gaussian Noise**, since the noise is added to the "clean" image – see next slide)

Original

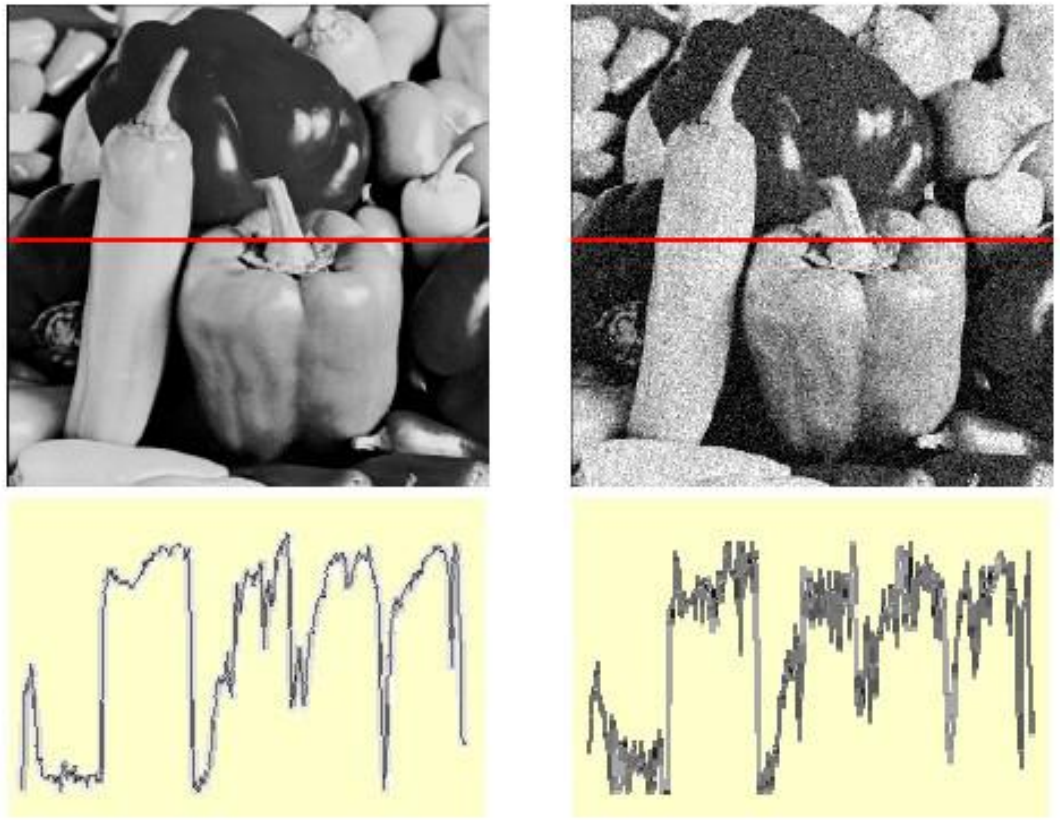
Salt and pepper noise

Impulse noise

Gaussian noise

Source: S. Seitz

Gaussian noise



$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

```
>> noise = randn(size(im)).*sigma;

>> output = im + noise;
```

sigma=1

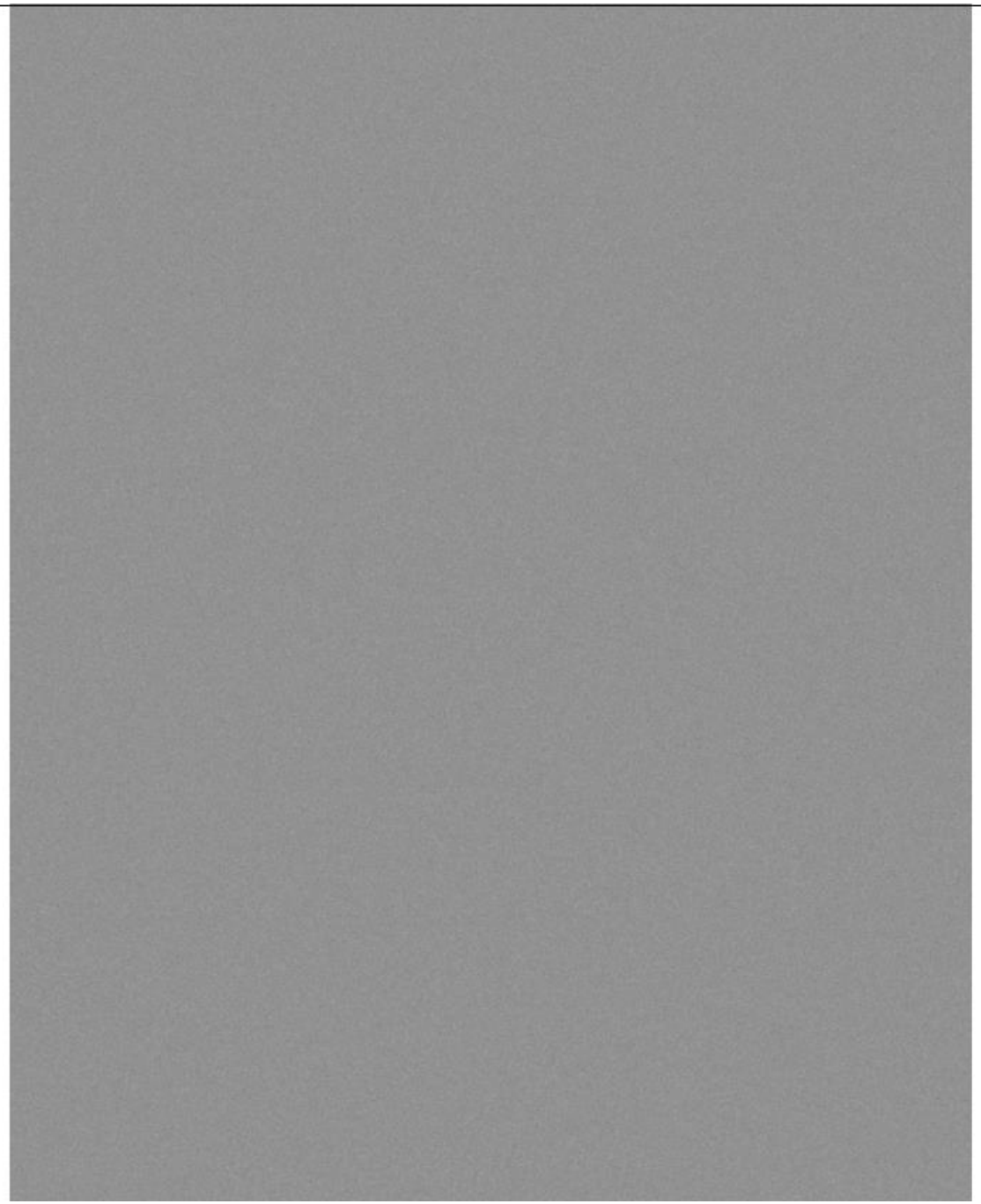
Effect of sigma (σ) on Gaussian noise η(μ,**σ**):

Im1=Im+ η,

where Im is the noiseless constant intensity image.

Im1 - the image to the left, has noise at every pixel, with standard deviation **σ=1.**

I.e. the pixels variations show the noise values assuming η(0,**σ**).

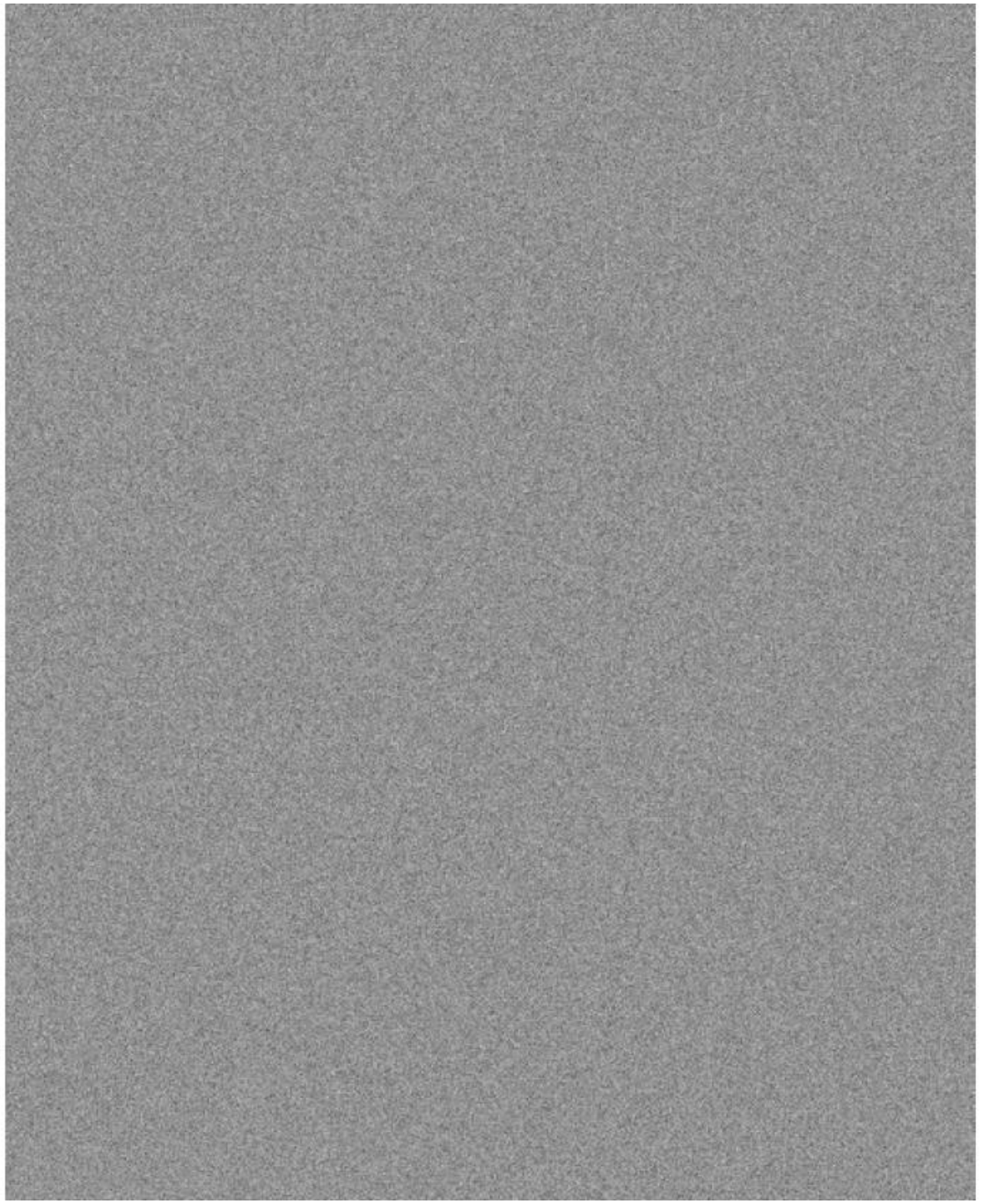


sigma=4

Effect of sigma (**σ**) on Gaussian noise:

Image shows the noise values for **σ=4.**

sigma=
16



Effect of sigma (**σ**) on Gaussian noise:

Image shows the noise values for **σ=16.**

sigma=1

Effect of sigma on Gaussian noise:

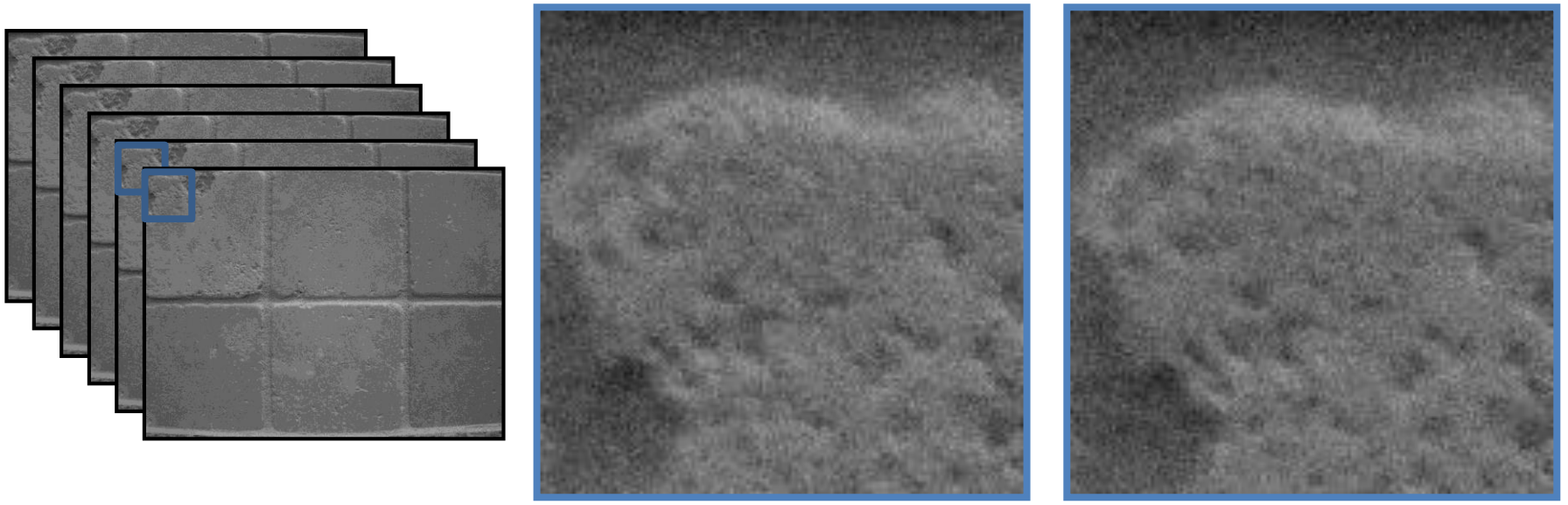
This shows the noise values added to the raw intensities of an image.
σ=1.

sigma=16

Effect of sigma on Gaussian noise

This shows the noise values added to the raw intensities of an image. **σ=16.**

Motivation: noise reduction



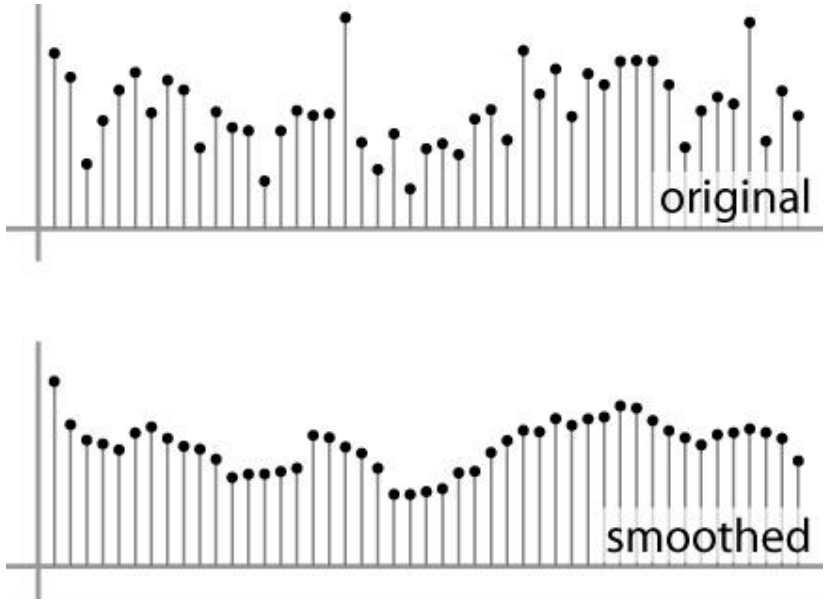
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

First attempt at a solution: Remove noise = Noise filtering

- Let's replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors
  - Expect noise processes to be independent from pixel to pixel

First attempt at a solution: Average of the pixel's values

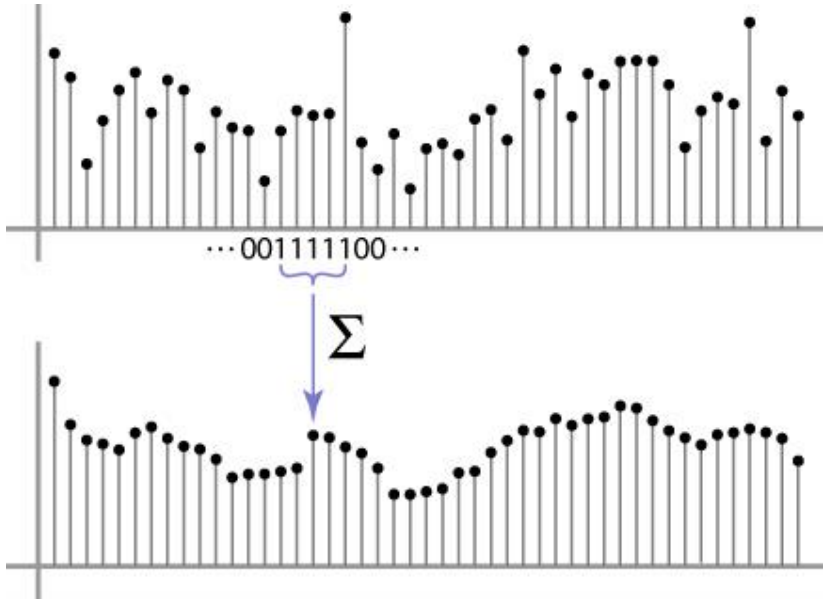
- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



Source: S. Marschner

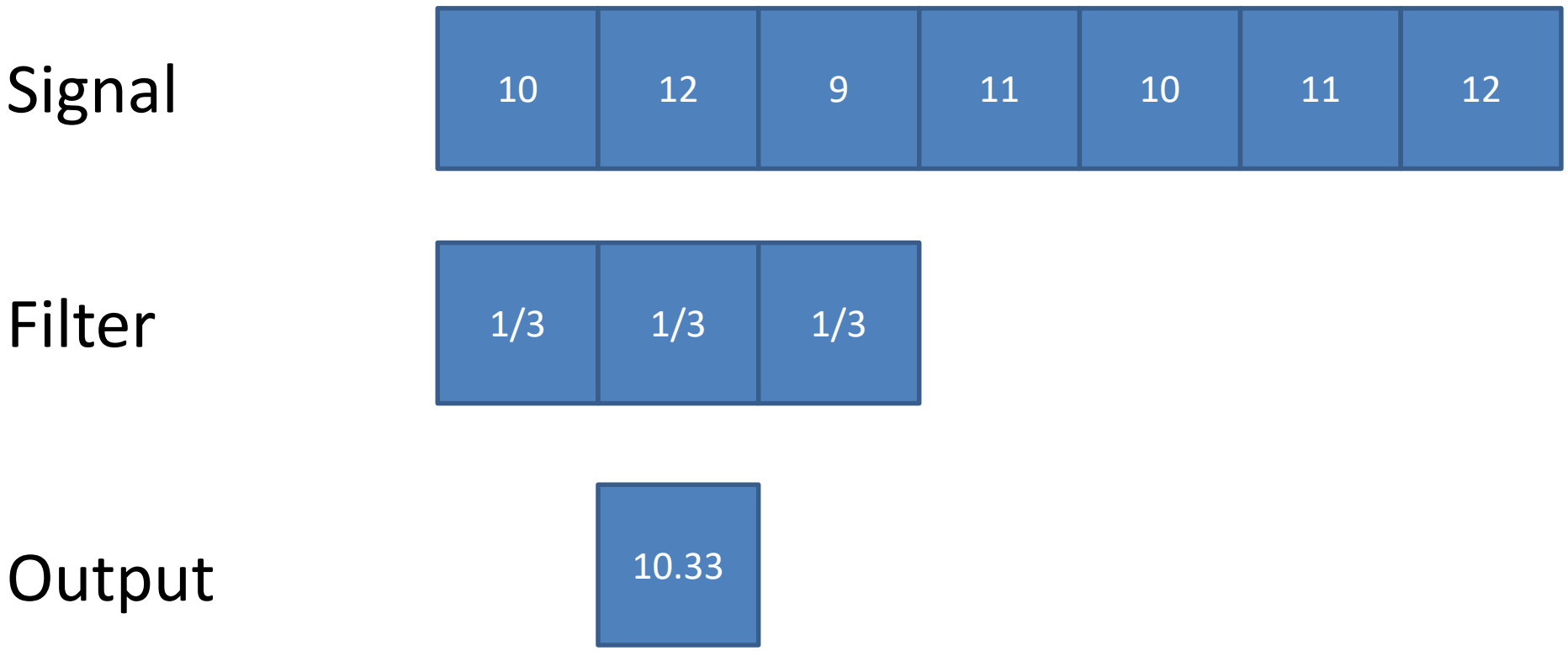
First attempt: Moving Average

- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5 (= [0.2, 0.2, 0.2, 0.2, 0.2])



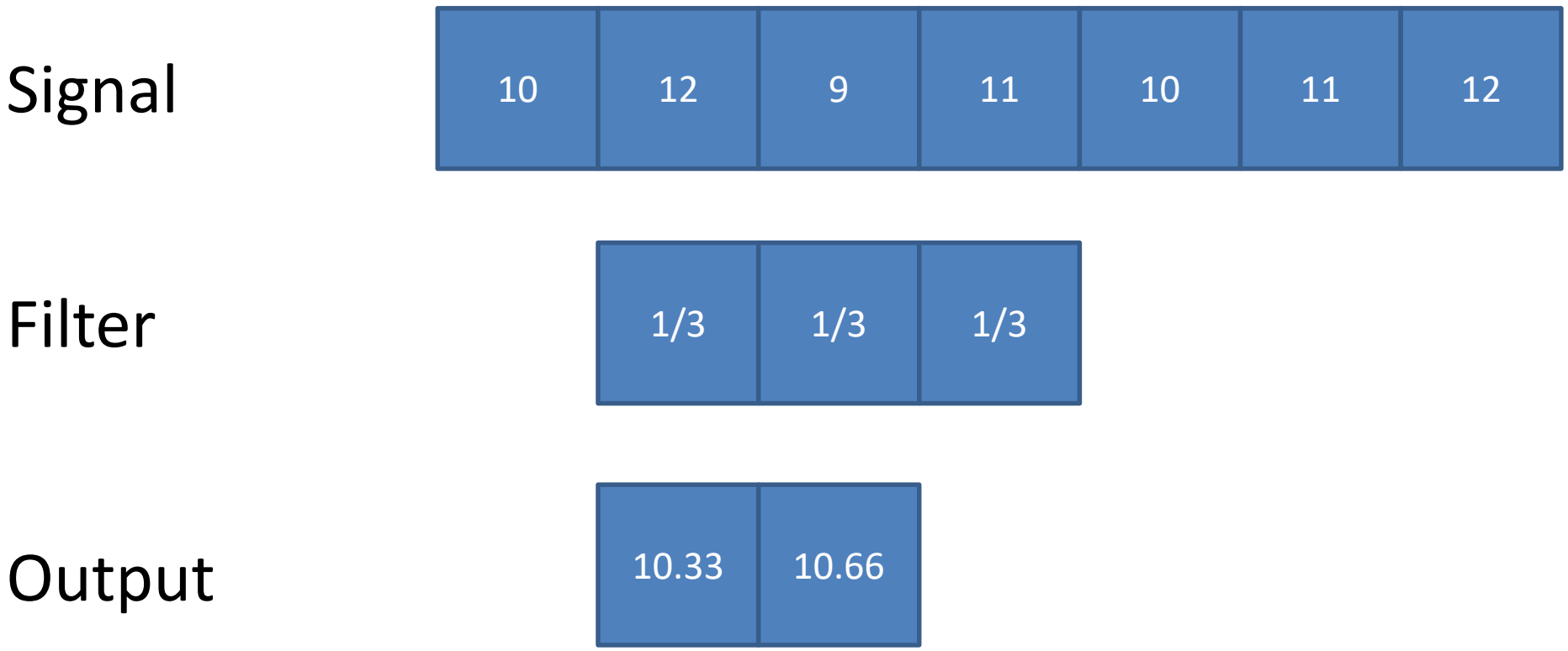
Source: S. Marschner

1D Case



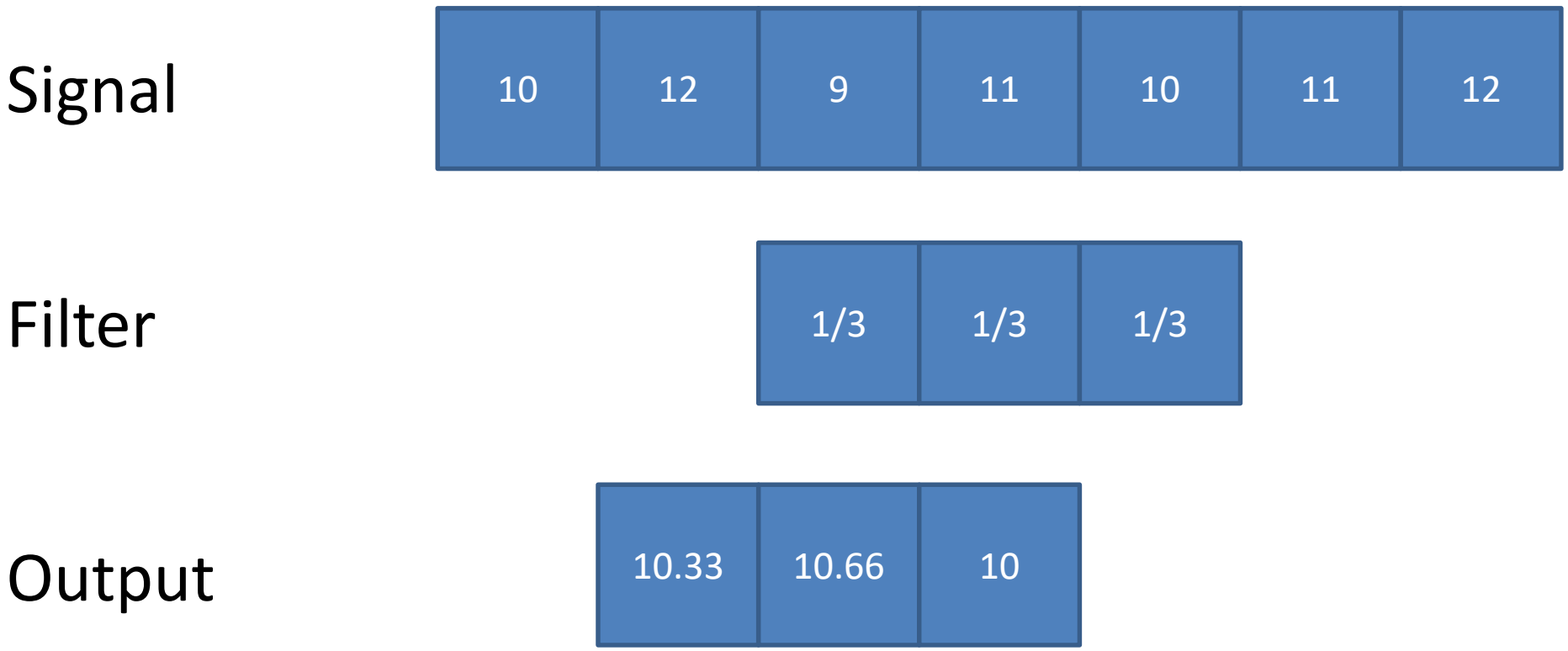
Source: J. Johnson

1D Case



Source: J. Johnson

1D Case



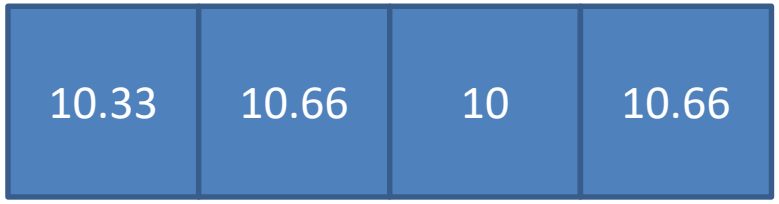
Source: J. Johnson

1D Case

Signal

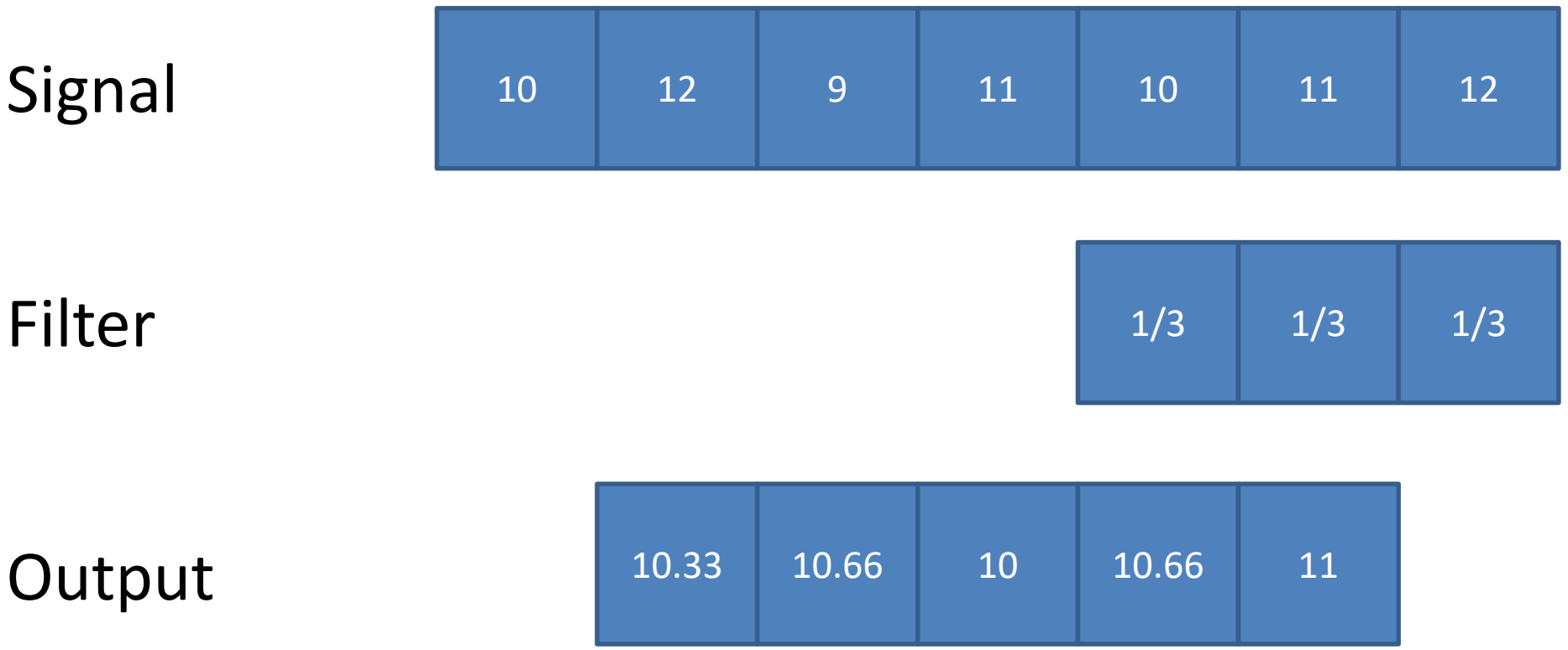
Filter

Output



Source: J. Johnson

1D Case

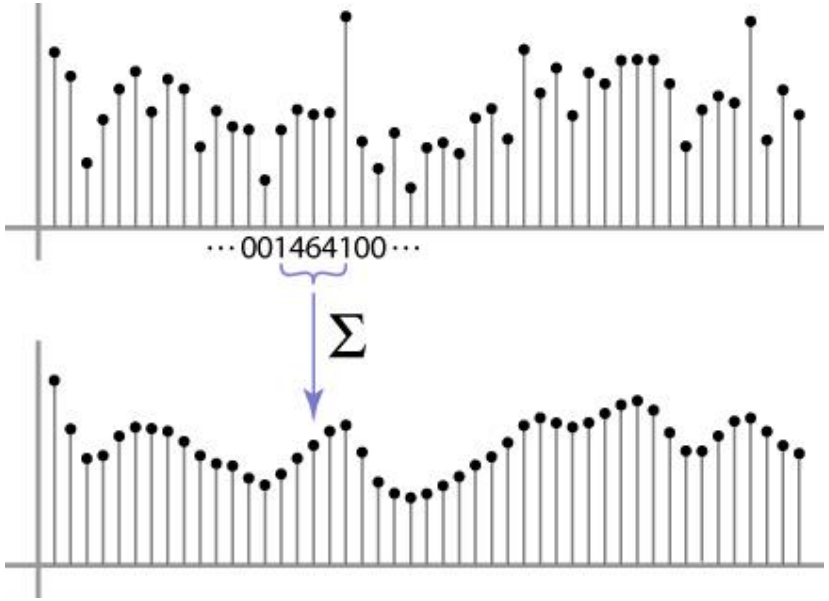


Source: J. Johnson

Second attempt: Weighted Moving Average

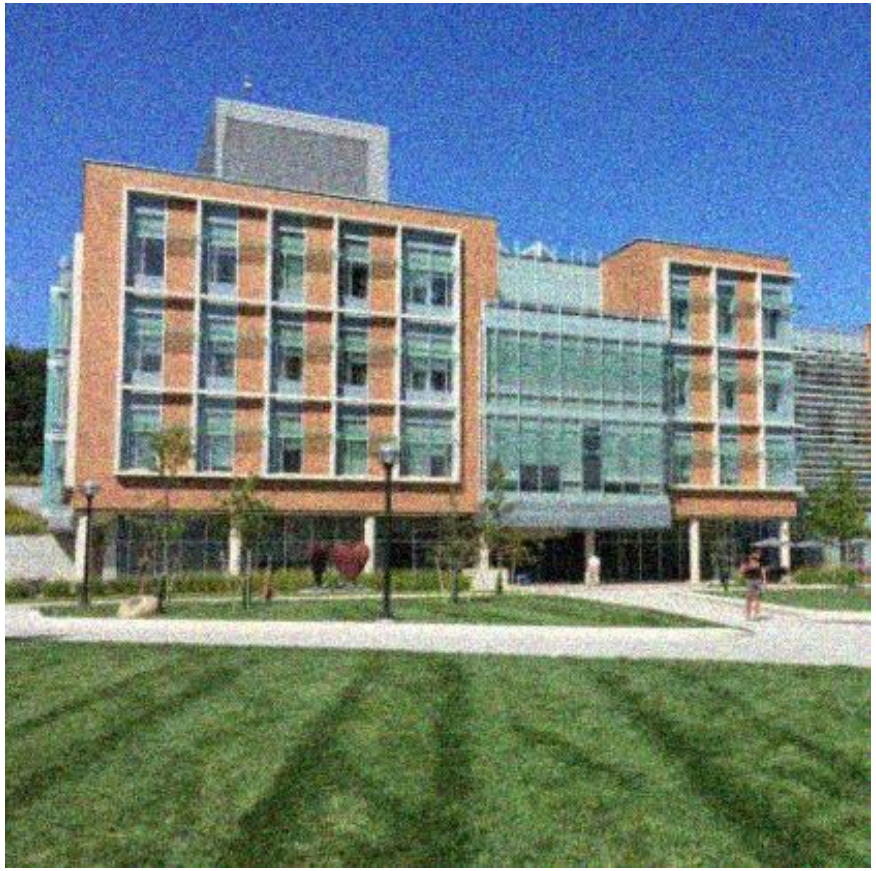
- We can use Non-uniform weights, to give the closer neighboring pixels, a larger weight
  [1, 4, 6, 4, 1] / 16

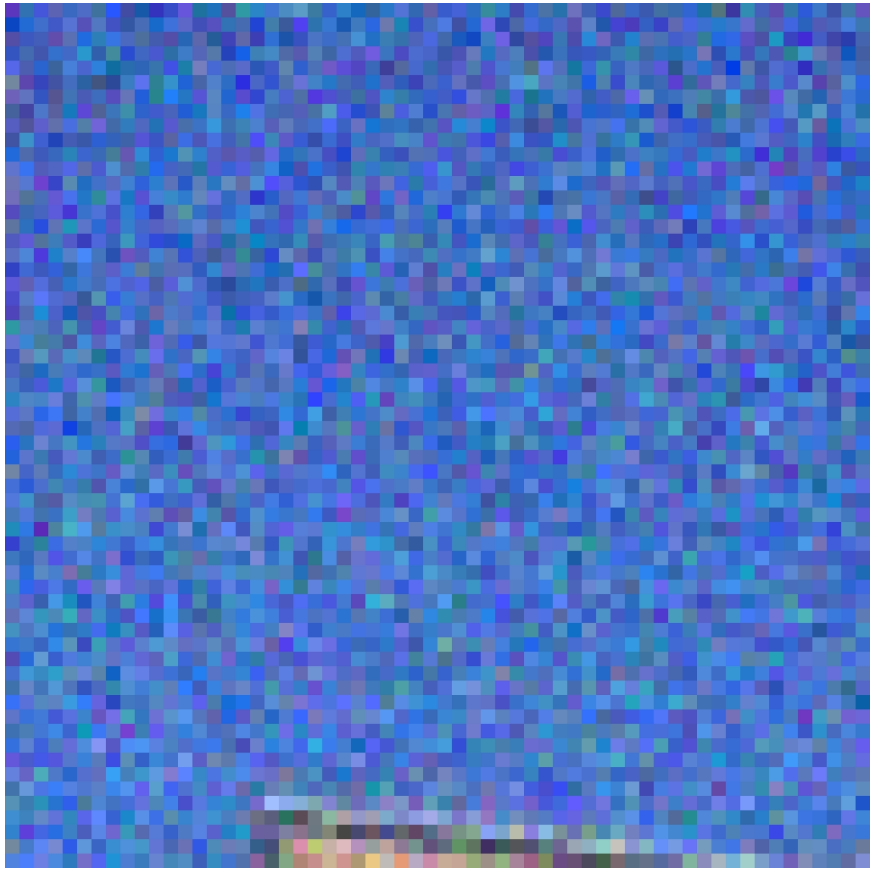
  (Note:
  in averaging,
  all weights >= 0
  the weights sum = 1)



Source: S. Marschner

A Noisy Image

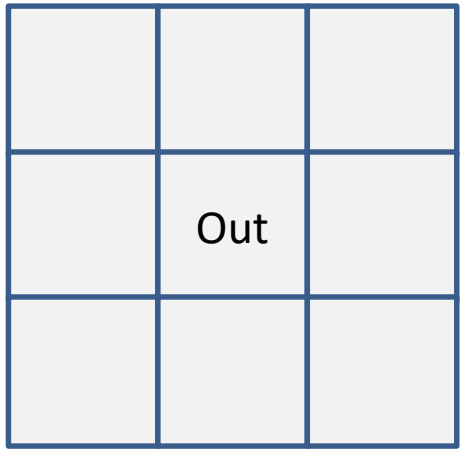




Source: J. Johnson

Cleaning it up

- We have noise in our image
- Let's replace each pixel with a *weighted* average of its neighborhood
- Weights are *filter kernel*



1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Slide Credit: D. Lowe

Source: J. Johnson

Applying a 2D Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

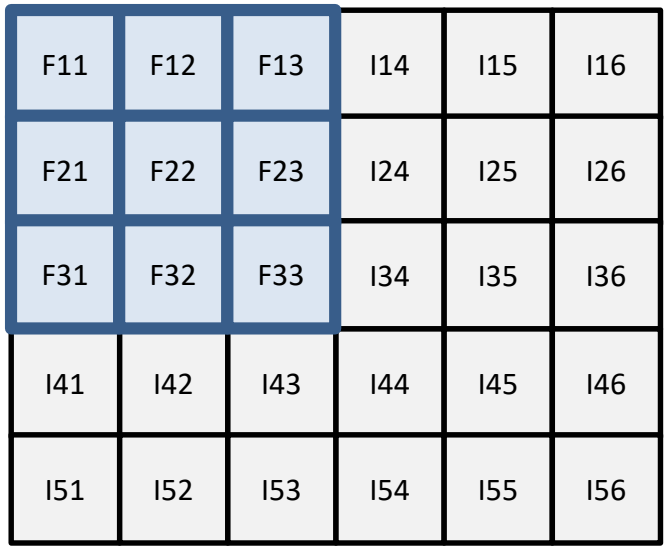
Output

O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

Source: J. Johnson

Applying a 2D Filter

Input & Filter



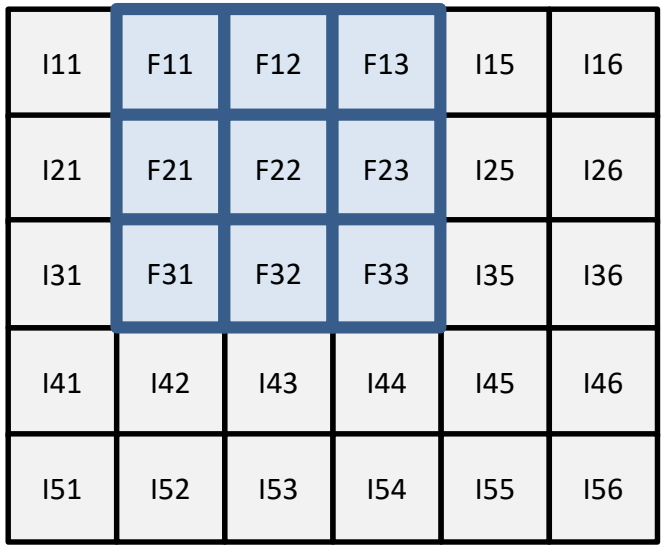
Output

$$O11 = I11*F11 + I12*F12 + \ldots + I33*F33$$

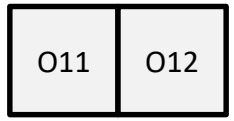
Source: J. Johnson

Applying a 2D Filter

Input & Filter



Output



$$O12 = I12*F11 + I13*F12 + \ldots + I34*F33$$

Source: J. Johnson

Applying a 2D Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

How many times can we apply a 3x3 filter to a 5x6 image?

Source: J. Johnson

Applying a 2D Filter

Input

I11	I12	I13	I14	I15	I16
I21	I22	I23	I24	I25	I26
I31	I32	I33	I34	I35	I36
I41	I42	I43	I44	I45	I46
I51	I52	I53	I54	I55	I56

Filter

F11	F12	F13
F21	F22	F23
F31	F32	F33

Output

O11	O12	O13	O14
O21	O22	O23	O24
O31	O32	O33	O34

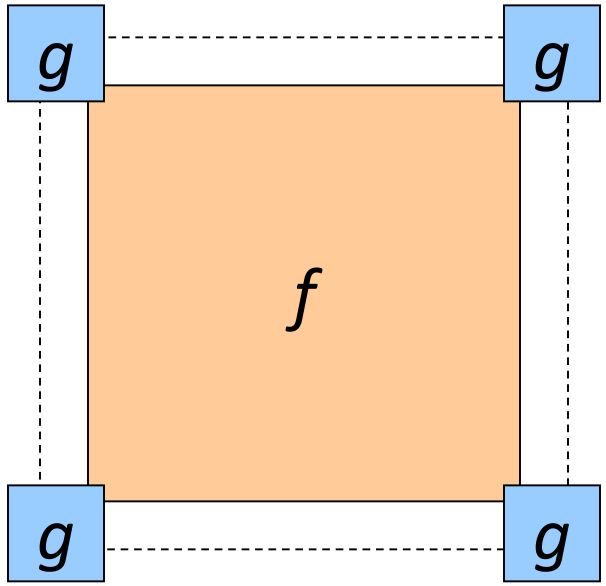
$$Oij = Iij*F11 + Ii(j+1)*F12 + \ldots + I(i+2)(j+2)*F33$$

Source: J. Johnson

Edge Cases

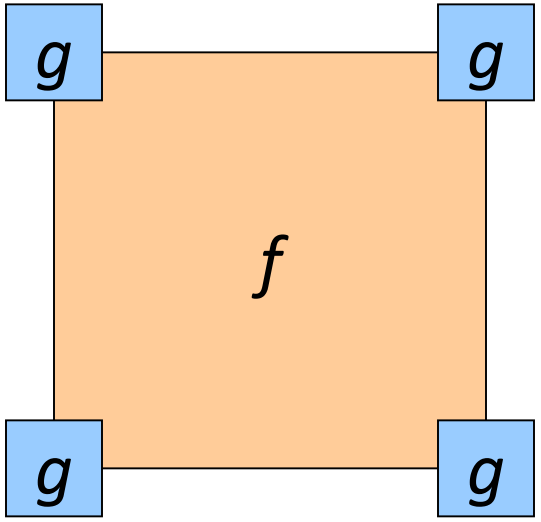
Convolution doesn't keep the whole image.
Suppose **f** is the image and **g** the filter.

Full: Any part of g touches f.

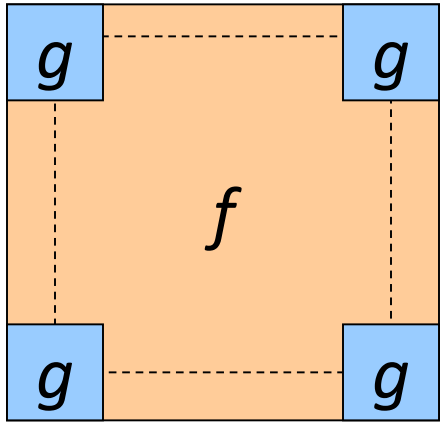


f/g Diagram Credit: D. Lowe

Same: Output is same size as f



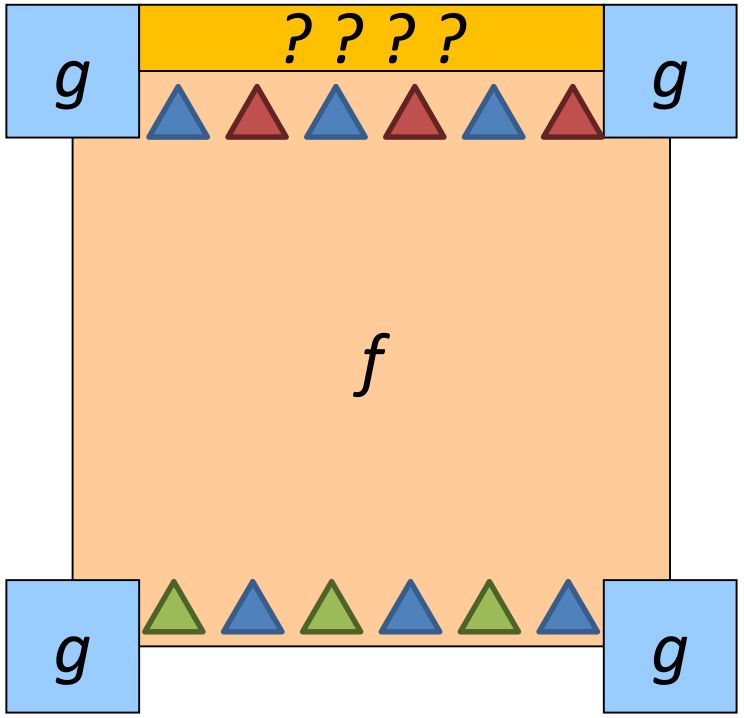
Valid: Filter doesn't fall off edge



Source: J. Johnson

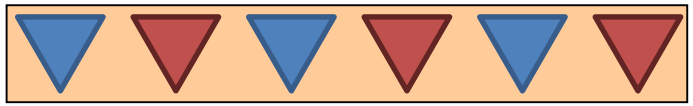
Edge Cases

What to about the “?” region?



f/g Diagram Credit: D. Lowe

Symm: fold sides over



Circular/Wrap: wrap around

pad/fill: add value, often 0

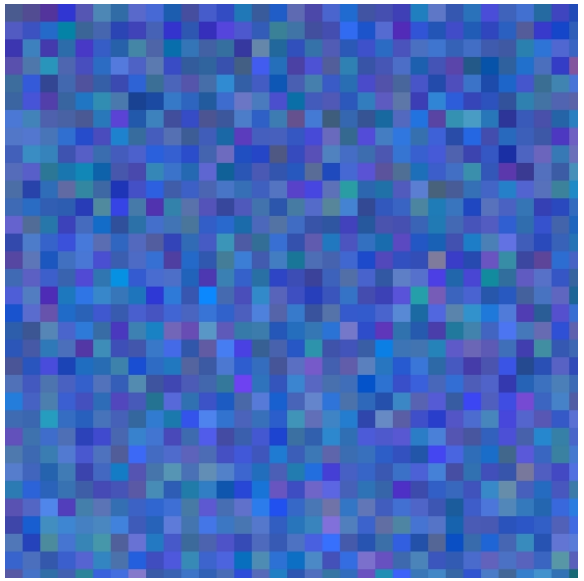
Source: J. Johnson

Edge Cases: Does It Matter?

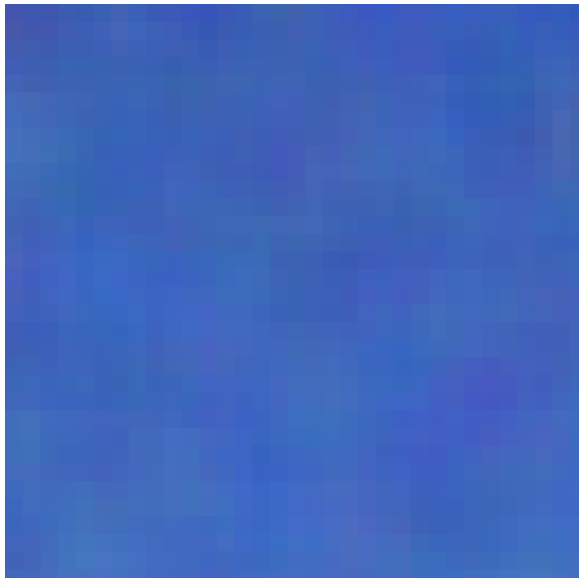
(I've applied the filter per-color channel)

Which padding did I use and *why*?

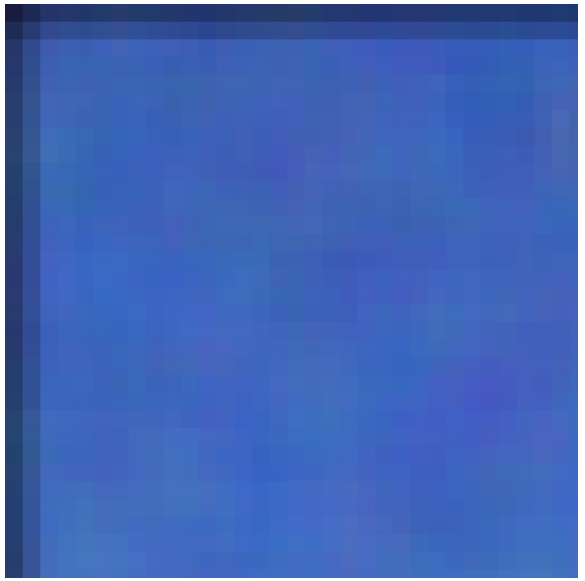
Input Image



Box Filtered ???



Box Filtered ???



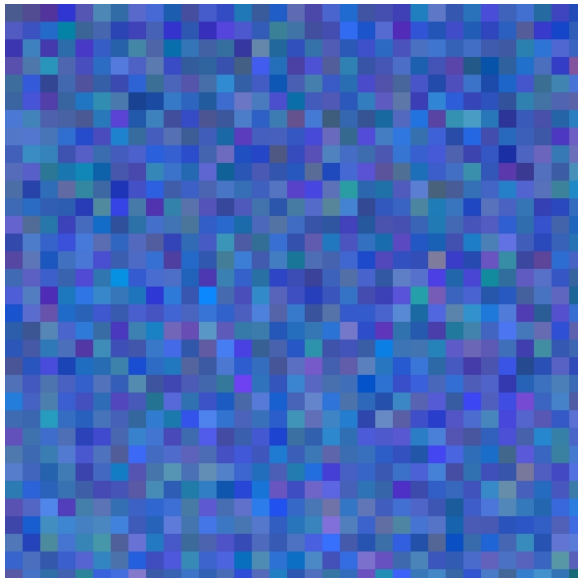
Note – this is a zoom of the filtered, not a filter of the zoomed

Source: J. Johnson

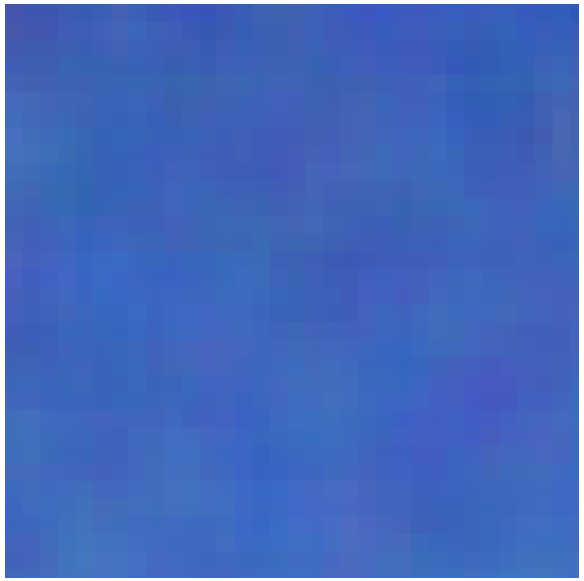
Edge Cases: Does It Matter?

(I've applied the filter per-color channel)

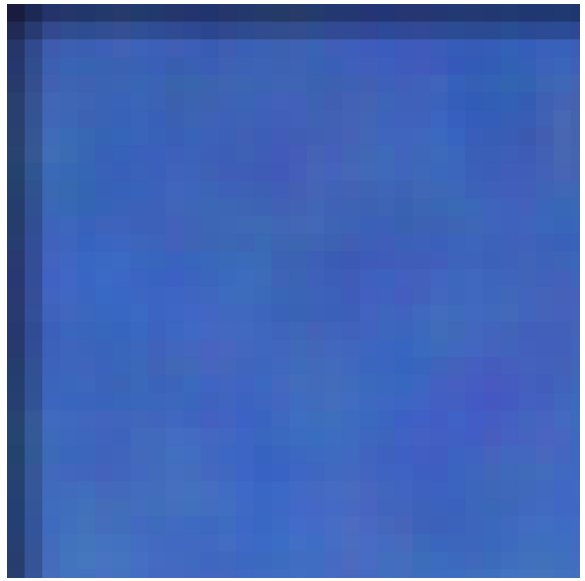
Input Image



Box Filtered Symm Pad



Box Filtered Zero Pad



Note – this is a zoom of the filtered, not a filter of the zoomed

Source: J. Johnson

Next: Practicing 2D Moving Average

- The averaging operation can be performed using either of the following mathematical operations:
  - Correlation
  - Convolution
- We start with an example (in the next slides), where:
  - The original image is marked as $F[x, y]$
  - The averaged result image is marked as $G[x, y]$
  - Each of the 3x3 averaging weights equals 1/9

Source: S. Seitz

Moving Average In 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$F[x, y]$

1/9	1/9	1/9	0	0	0	0	0	0	0
1/9	1/9	1/9	0	0	0	0	0	0	0
1/9	1/9	1/9	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

$G[x, y]$

	0								

Source: S. Seitz

Moving Average In 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$F[x, y]$

0	1/9	1/9	1/9	0	0	0	0	0	0
0	1/9	1/9	1/9	0	0	0	0	0	0
0	1/9	1/9	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

$G[x, y]$

	0	10							

Source: S. Seitz

Moving Average In 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$F[x, y]$

0	0	1/9	1/9	1/9	0	0	0	0	0
0	0	1/9	1/9	1/9	0	0	0	0	0
0	0	1/9	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

$G[x, y]$

	0	10	20						

Source: S. Seitz

Moving Average In 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$F[x, y]$

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

$G[x, y]$

	0	10	20	30					

Source: S. Seitz

Moving Average In 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$F[x, y]$

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

$G[x, y]$

	0	10	20	30	30				

Source: S. Seitz

Moving Average In 2D

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$F[x, y]$

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

$G[x, y]$

	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	

Source: S. Seitz

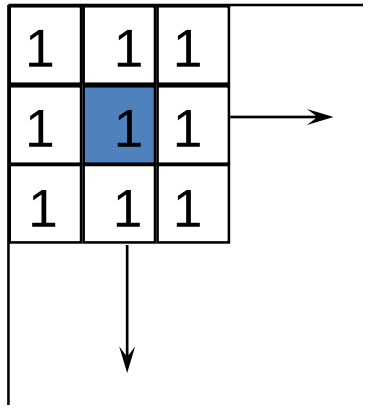
Spatial filtering

In spatial filtering, the pixel in the output image is given a value calculated from a local neighborhood in the input image.

The local neighborhood is described by a mask, or spatial filter (typical are odd squares of size 3x3, 5x5, 7x7… pixels)

For Convolution, the mask is first flipped (rotated in 180˚) - (shown later)

Filtering is performed by letting the (rotated) mask move over the image. The center pixel in the output image is given a value that depends on the input image and the weights of the mask.



Several types of spatial filters:

- Linear filters (shown): Correlation / Convolution
- Non linear, fixed filters (one example will be shown)
- Adaptive filters (advanced ☺)

Correlation (“filtering”)

For averaging filter (all weights are 1, divided by the number of elements):
Assume the averaging window size is $(2k+1)$ x $(2k+1)$:

K=1

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

$$\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

k, k, k, k

k=2

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel F[i,j]}}$$

Side Note: Convolution vs. Correlation

2D Convolution **(filtering)**

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$

2D Cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

$$G = H \otimes F$$

Convolution "filtering"

For averaging filter (all weights are 1, divided by the number of elements):

Assume the averaging window size is $2k+1$ x 2k+1:

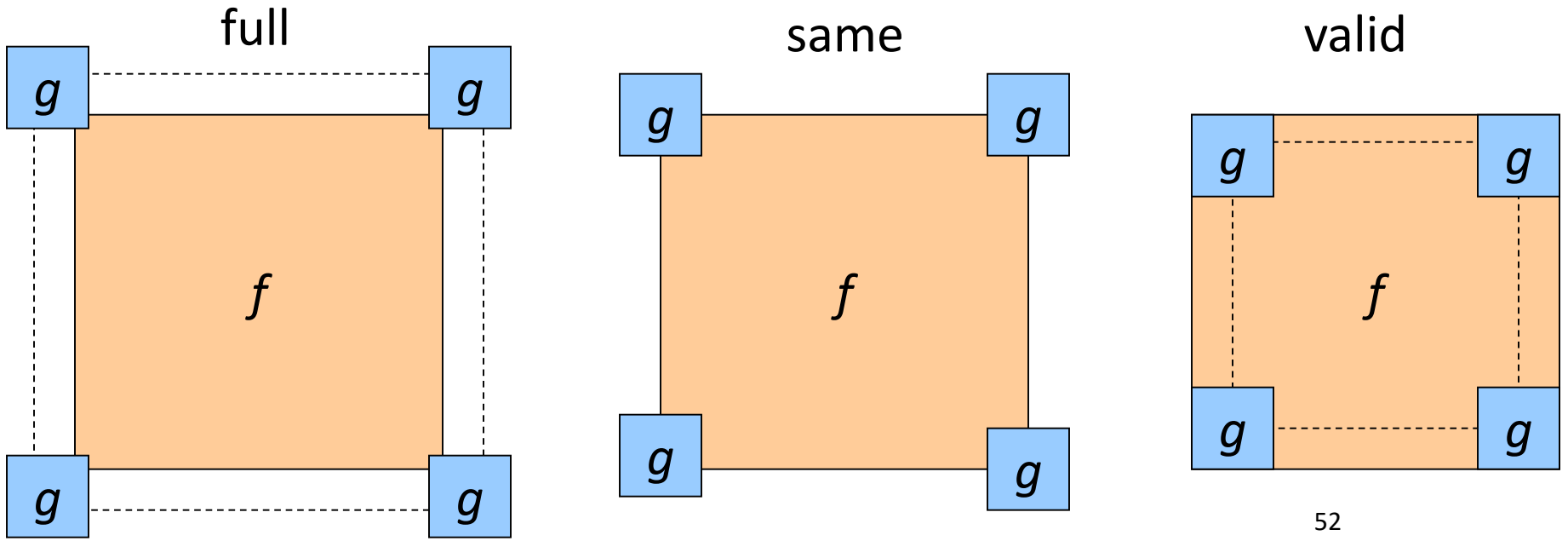
$$G[i,\ j] = \underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k}}_{\text{Loop over all pixels in neighborhoods of 2k+1 pixels in each dimension}} \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute (uniform) weight to each pixel}} \cdot \underbrace{F[i-u,\ j-v]}_{\text{Image value in neighborhood around current image pixel F[i,j]}}$$

Now generalize to allow different weights, $H[u,v]$, depending on neighboring pixel's **relative** position:

$$G[i,\ j] = \sum_{u=-k}^{k}\sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Uniform or non-uniform weights}} \cdot F[i-u,\ j-v]$$

Boundary issues

- What is the size of the output?
- Boundary options: filter2(f, h, *shape*)
  - *shape* = 'full': output size is sum of sizes of f and h -1
  - *shape* = 'same': output size is same as f
  - *shape* = 'valid': output size is difference of sizes of f and h+1



Source: S. Lazebnik

Convolution Size

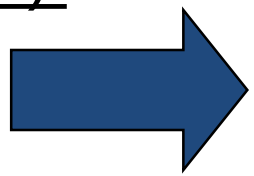
Image size = $M_1 \times N_1$

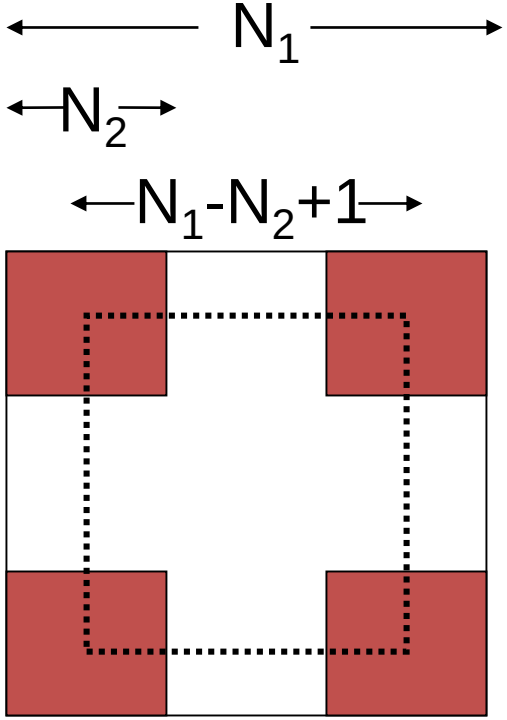
Mask size = $M_2 \times N_2$

<u>IF TRUNCATED ('valid'):</u>

Convolution size =

$(M_1 - M_2 + 1) \times (N_1 - N_2 + 1)$





<u>IF PADDED ('full'):</u>

Convolution size =

$(M_1 + M_2 - 1) \times (N_1 + N_2 - 1)$

Typical Mask sizes
= 3×3, 5 ×5, 7×7,
9 ×9, 11×11

What is the convolved image size for a 128 × 128 image and 7 × 7 mask?

Convolution (Review)

- Extremely important concept in computer vision, image processing, signal processing, etc.
- General idea: reduce a filtering operation to the repeated application of a mask (or filter kernel) to the image
  - Kernel can be thought of as an NxN image
  - N is usually odd so kernel has a central pixel
- In practice
  - (flip kernel – convolution / no flip - correlation)
  - Align kernel center pixel with an image pixel
  - Pointwise multiply each kernel pixel value with corresponding image pixel value and add results
  - Resulting sum is normalized by kernel weight
  - Result is the value of the pixel centered on the kernel

Correlation "operating a kernel"

$$G[i,\ j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] \cdot F[i+u,\ \ j+v]$$

This is called **cross-correlation**, denoted $G = H \otimes F$

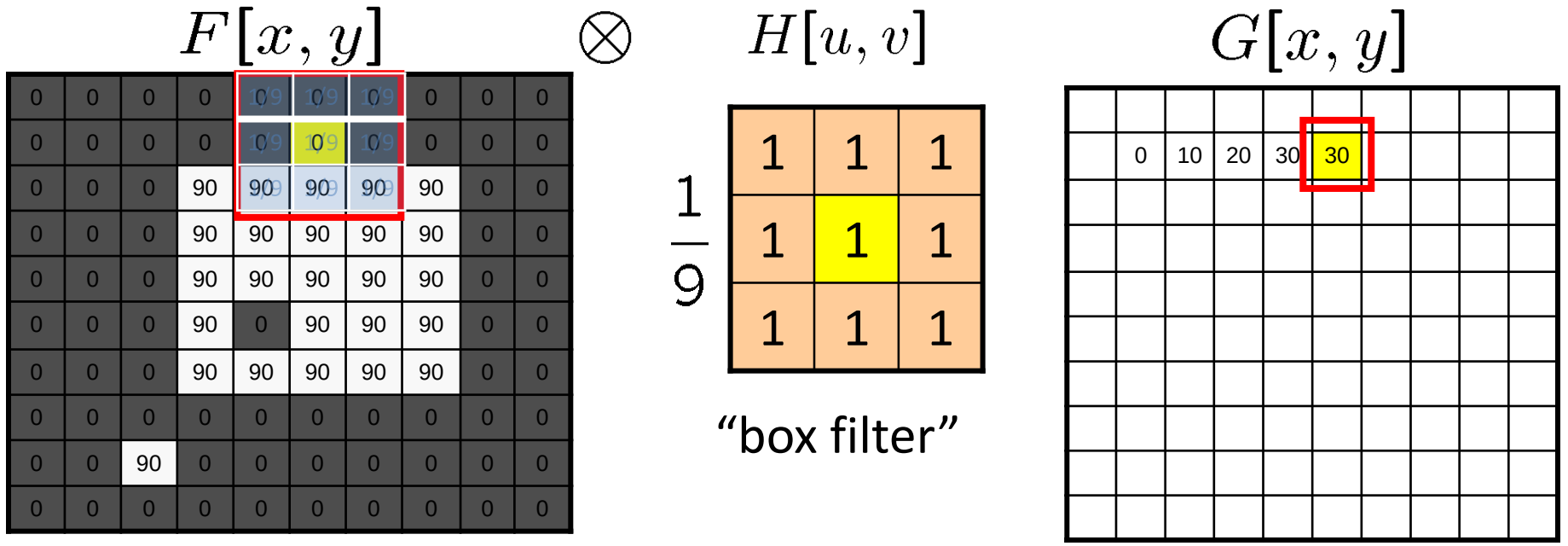
Correlation / Filtering an image:

for a source image *F*, with pixels in position [*i,j*] marked as *F*[*i,j*], and a result image *G*, with pixels in position [*i,j*] marked as *G*[*i,j*]:

- Calculate the value of **each pixel in the result image**, ***G*[*i,j*]**, using a linear combination of the corresponding pixel and its neighbors around *F*[*i,j*].
- The filter "kernel" or "mask" *H*[*u,v*] is the prescription for the weights in the linear combination.

Averaging filter

- What values belong in the kernel *H* for the moving average example?



$$G = H \otimes F$$

Correlation Example (I)

Image Data:

10	12	40	16	19	10
14	22	52	10	55	41
10	14	51	21	14	10
32	22	9	**9**	19	14
41	18	9	22	27	11
10	7	8	8	4	5

Mask:

1	1	1	1
0	1	1	1
-1	1	1	1
	-1	0	1

Mask:

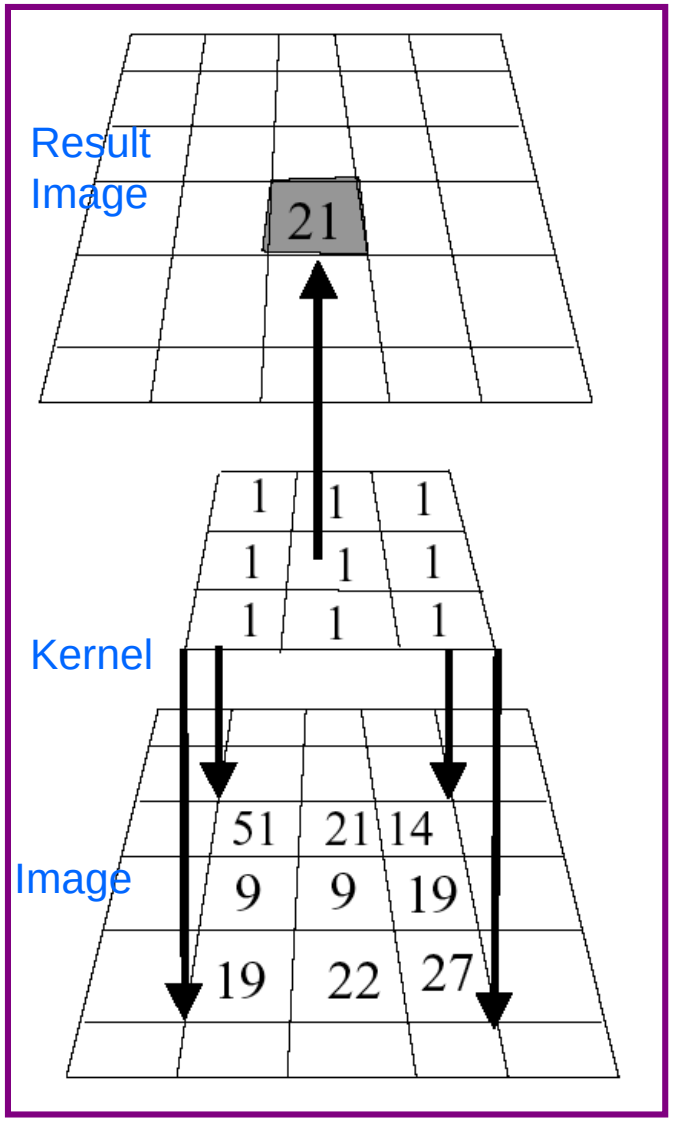
1	1	1	1
0	1	1	1
-1	1	1	1
	-1	0	1

$$S = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \text{Mask}[u,v]$$

$$\Rightarrow H[u,v] = \frac{1}{S} \text{Mask}[u,v]$$

$$G[i,j] = \sum_{u=-1}^{1} \sum_{v=-1}^{1} H[u,v] \cdot F[i+u,\ j+v] = F \otimes H = H \otimes F$$

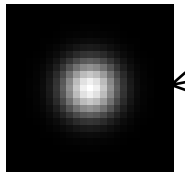
Correlation / Convolution Example (II)



- Kernel is aligned with pixel in image, multiplicative sum is computed, normalized, and stored in result image.
- (flip kernel for convolution)
- Process is repeated across image.
- What happens when kernel is near edge of input image?

$$\begin{bmatrix} 1*51 + 1*21 + 1*14 + \\ 1*9 + 1*9 + 1*19 + \\ 1*19 + 1*22 + 1*27 \end{bmatrix} 1/9 = 21$$

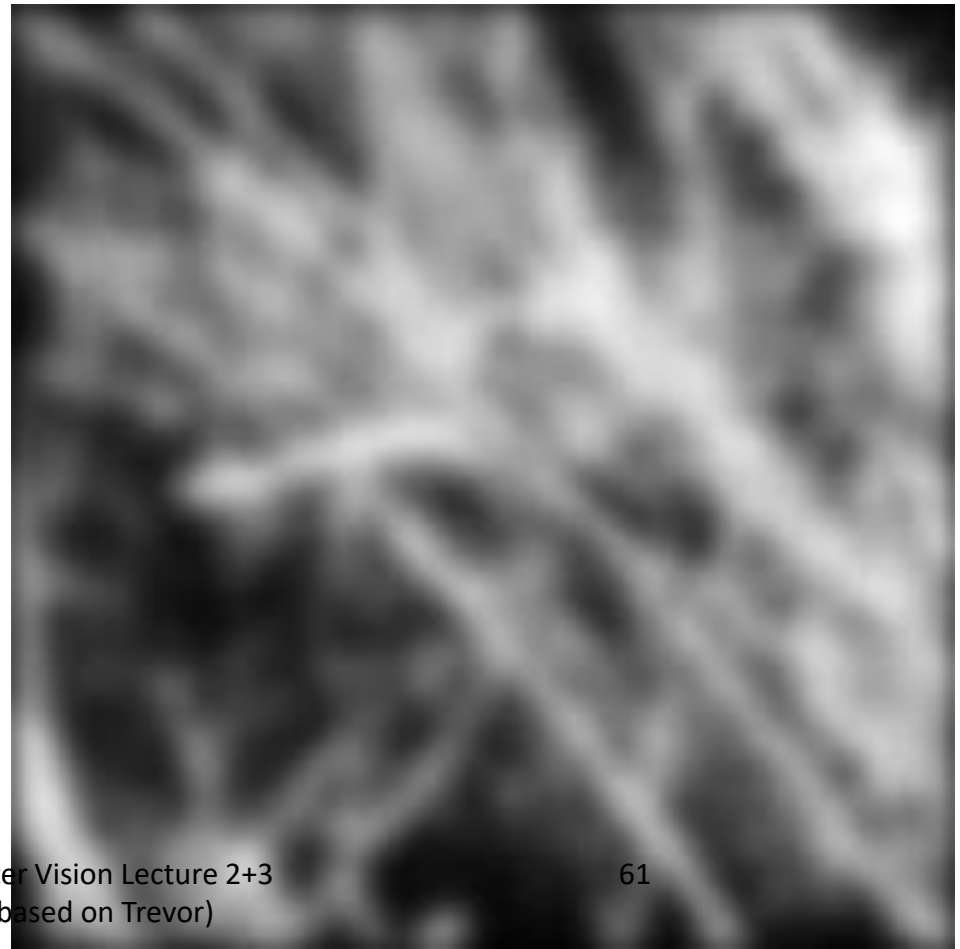
Smoothing with Gaussian filter



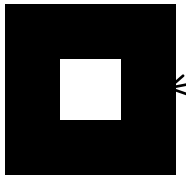
depicts Gaussian filter:
white = high value, black = low value

original

filtered



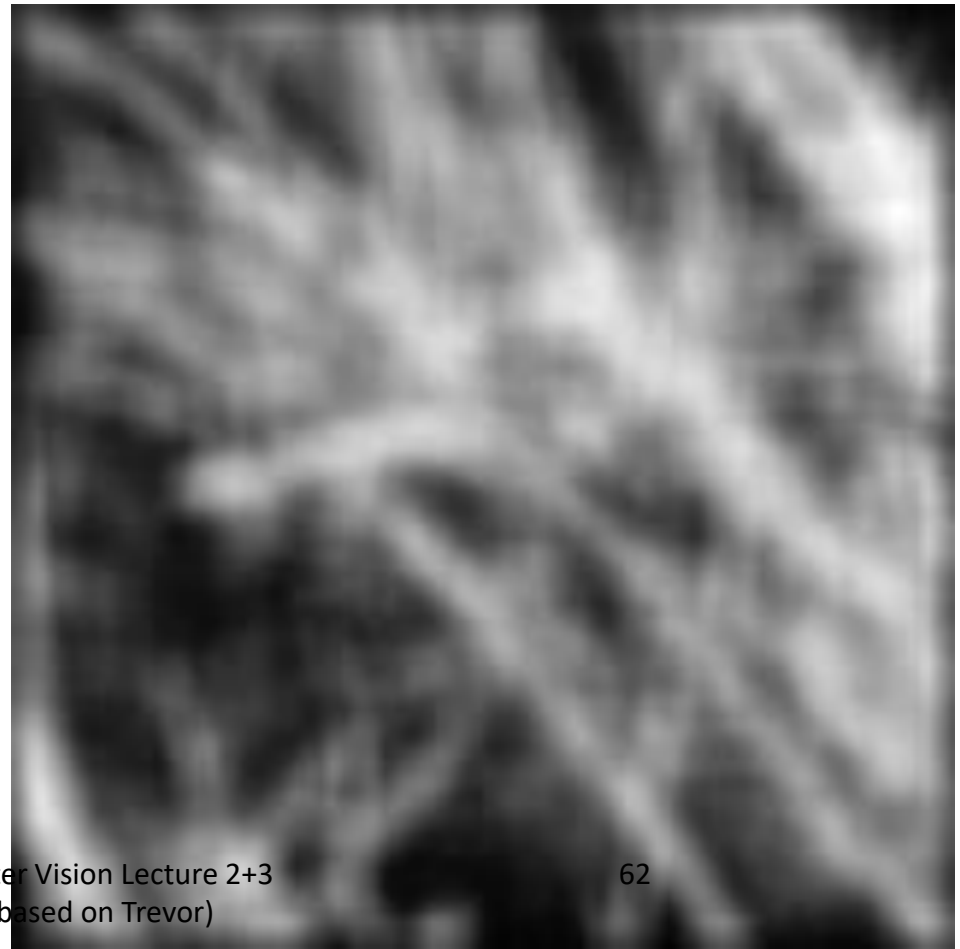
Smoothing with averaging (box) filter



depicts box filter:
white = high value, black = low value

original

filtered



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
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians

Source: K. Grauman

Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

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

$F[x, y]$

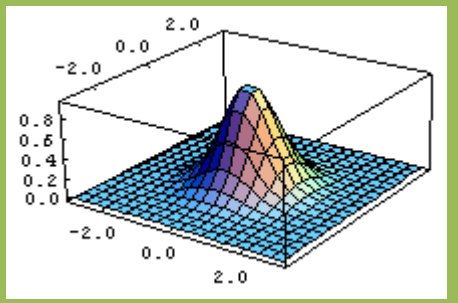
$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$H[u, v]$

This kernel is an approximation of a Gaussian function:

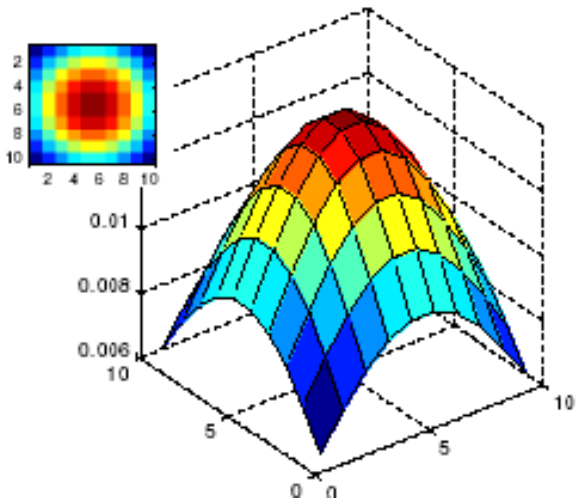
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Source: S. Seitz

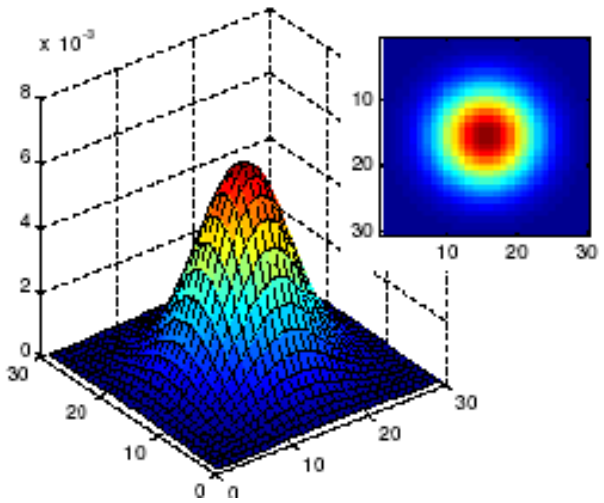
Gaussian filters

- What parameters matter here?
- **Size** of kernel or mask
  - Note, Gaussian function has infinite support, but discrete filters use finite kernels



σ = 5

10 x 10 kernel

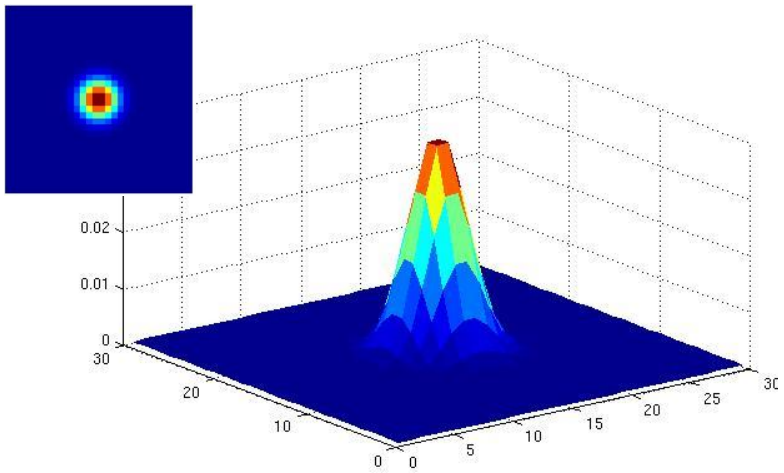


σ = 5

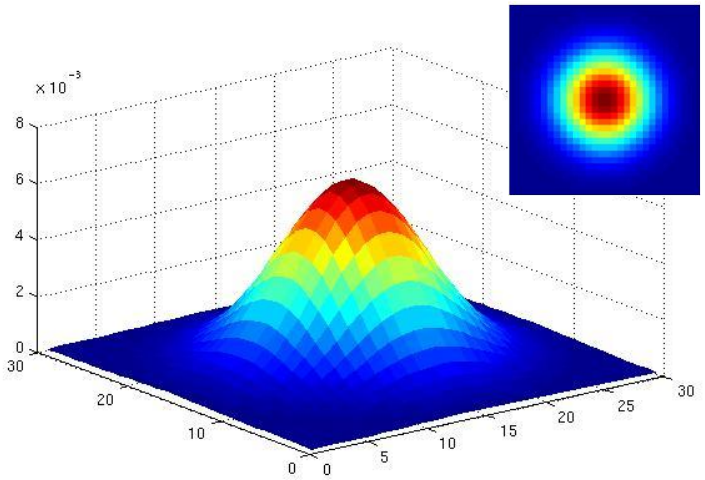
30 x 30 kernel

Gaussian filters

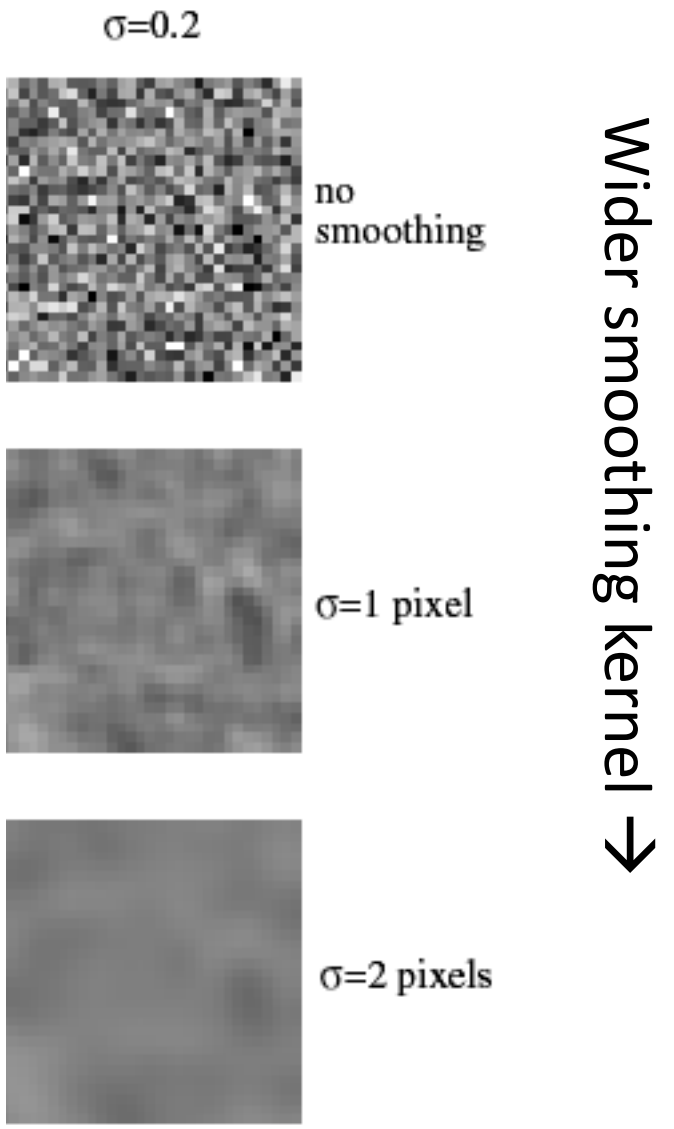
- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing



σ = 2
30 x 30 kernel



σ = 5
30 x 30 kernel


σ=0.2
no smoothing
σ=1 pixel
σ=2 pixels
Wider smoothing kernel →

Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black = 0)
    - wrap around
    - copy edge
    - reflect across edge

Source: S. Marschner

Shift invariant linear system (applied to convolution)

- **Shift invariant:**
  - Operator behaves the same everywhere, i.e. the value of the output **depends on the pattern in the image neighborhood**, **not on the position of the neighborhood**.
- **Linear:**
  - Superposition: $h*\left(f_1+f_2\right)=\left(h*f_1\right)+\left(h*f_2\right)$
  - Scaling: $h*\left(k\cdot f\right)=k\cdot\left(h*f\right)$
  - **Linear, Shift Invariant** system are marked as **LSI Systems**.

Reminder:
Properties of convolution (for any dimension)

- Linear & shift invariant
- Commutative: $f * g = g * f$
- Associative: $(f * g) * h = f * (g * h)$
- Identity: $f[\mathrm{m},\mathrm{n}] * \delta[m,n] = f[m,n] \Leftrightarrow f * \delta = f$

  where the unit impulse:

  $\delta[x] = [\ldots, 0, 0, 1, 0, 0, \ldots]$ $(1D)$

  ; where $\delta[x,y] = \begin{bmatrix} \ddots & \vdots & \iddots \\ & 0\ 0\ 0 & \\ \cdots & 0\ 1\ 0 & \cdots \\ & 0\ 0\ 0 & \\ \iddots & \vdots & \ddots \end{bmatrix}$ $(2D)$
- Differentiation:

$$\frac{\partial}{\partial x}(f * g) = \left(\frac{\partial}{\partial x} f\right) * g = f * \left(\frac{\partial}{\partial x} g\right)$$

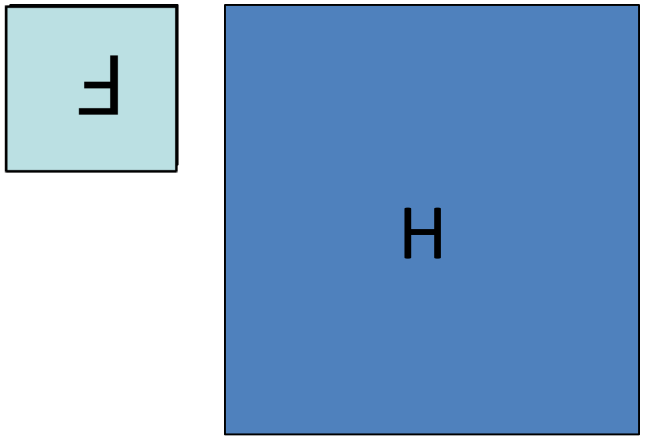
Summary of Convolution (Filtering)

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left = rotate 180°)
  - Then, for each pixel in the output image, apply sum of multiplications

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H \star F$$

Notation for convolution operator



Convolution vs. correlation

2D Convolution (filtering)

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$

2D Cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

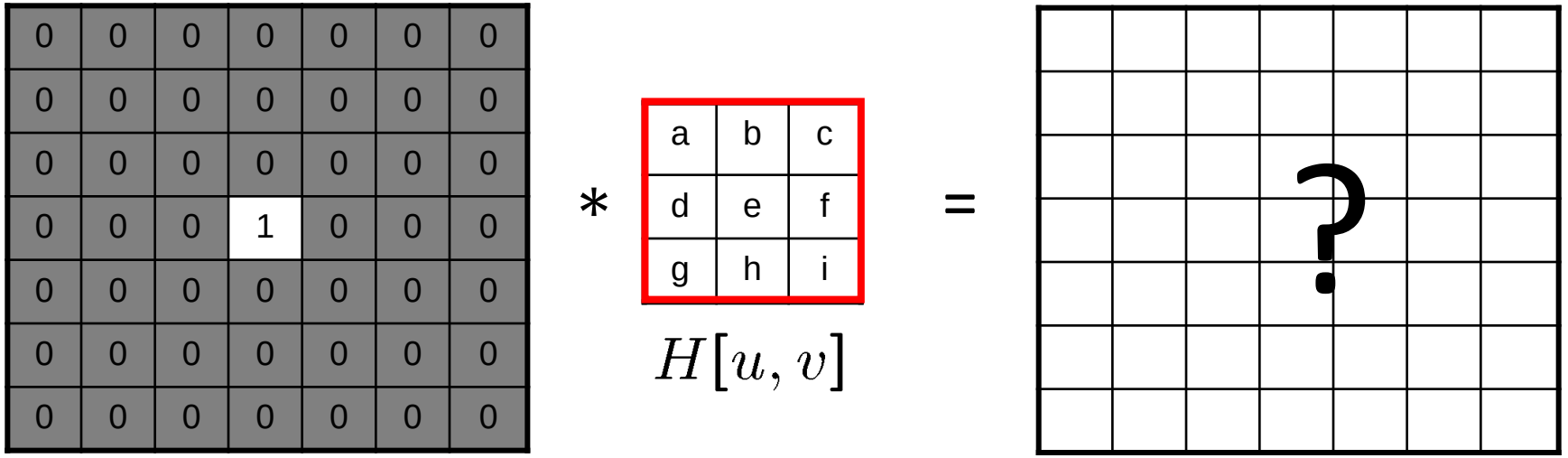
$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

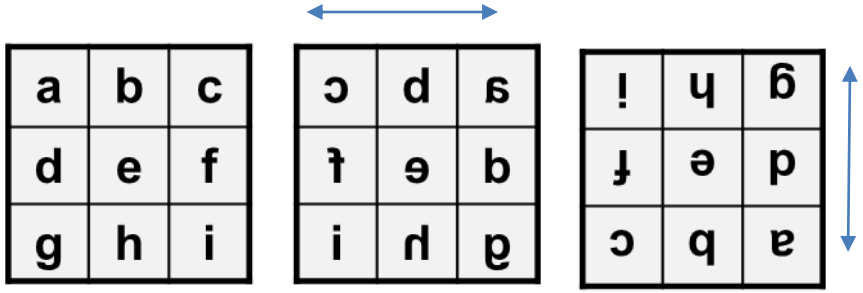
If the input is an impulse signal, how will the outputs differ?

Filtering an impulse signal

What is the result of **filtering** the impulse signal (image) *F* with the arbitrary kernel *H*?



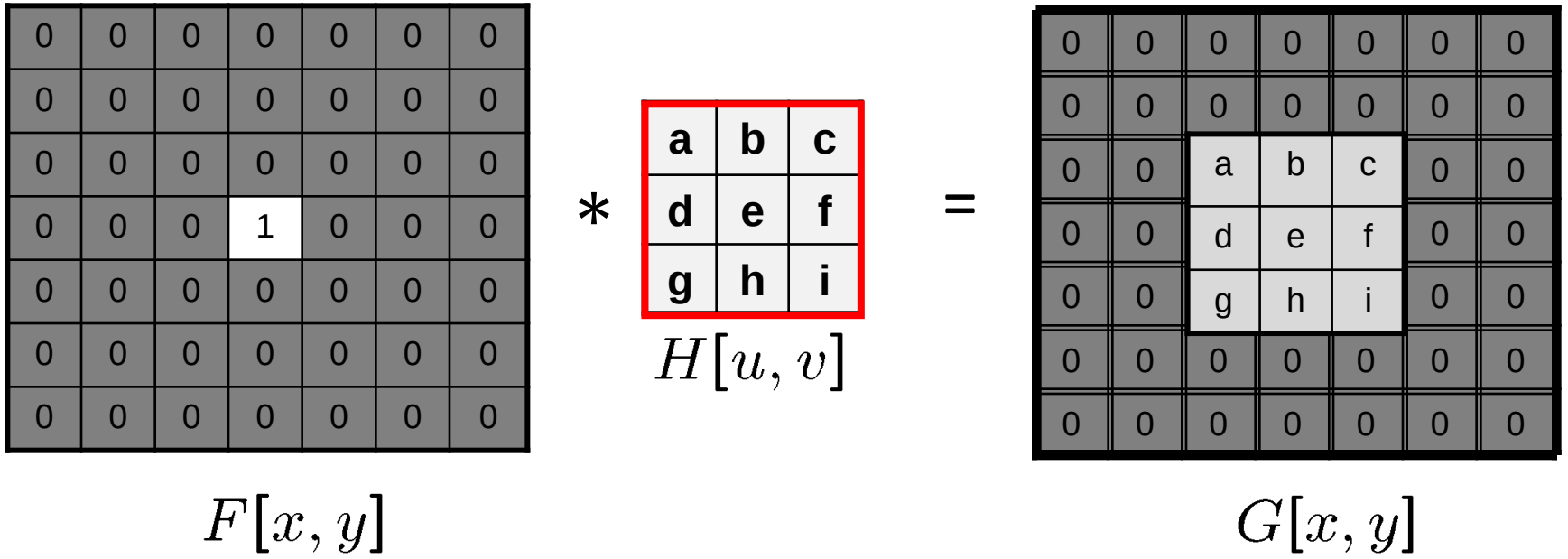
Note: Rotate 180°:



Convolution with an impulse signal

(Convolution = **Filtering**)

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?



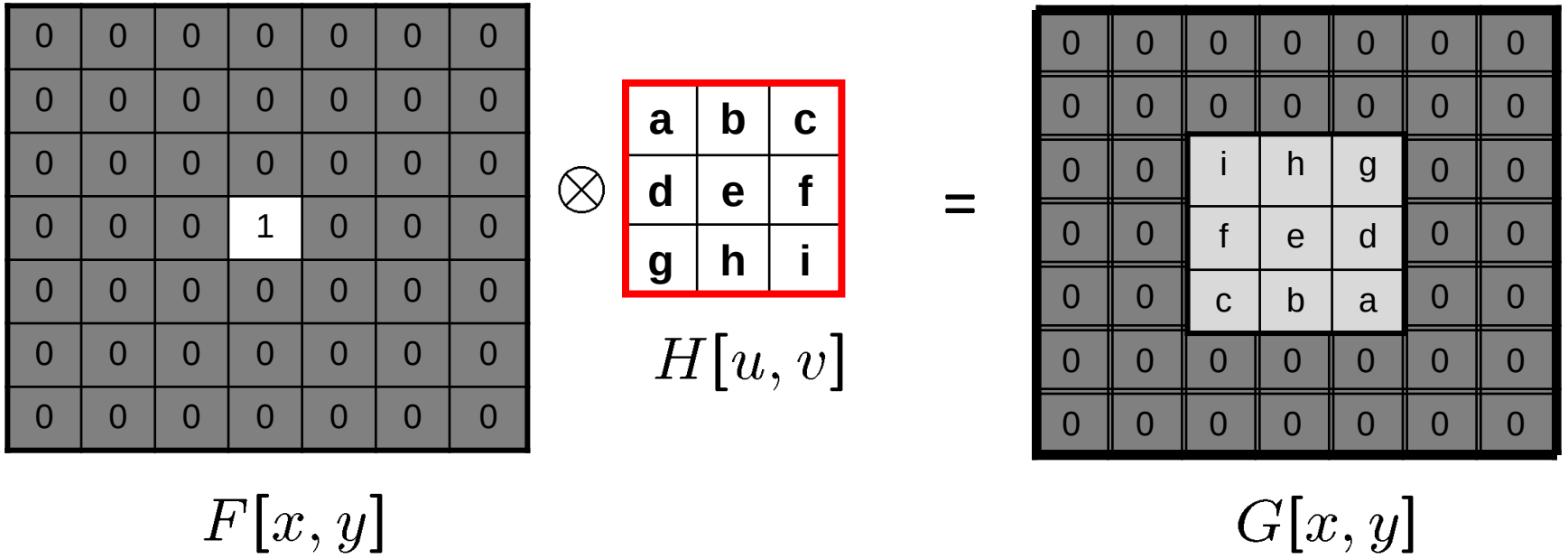
$F[x, y]$

$H[u, v]$

$G[x, y]$

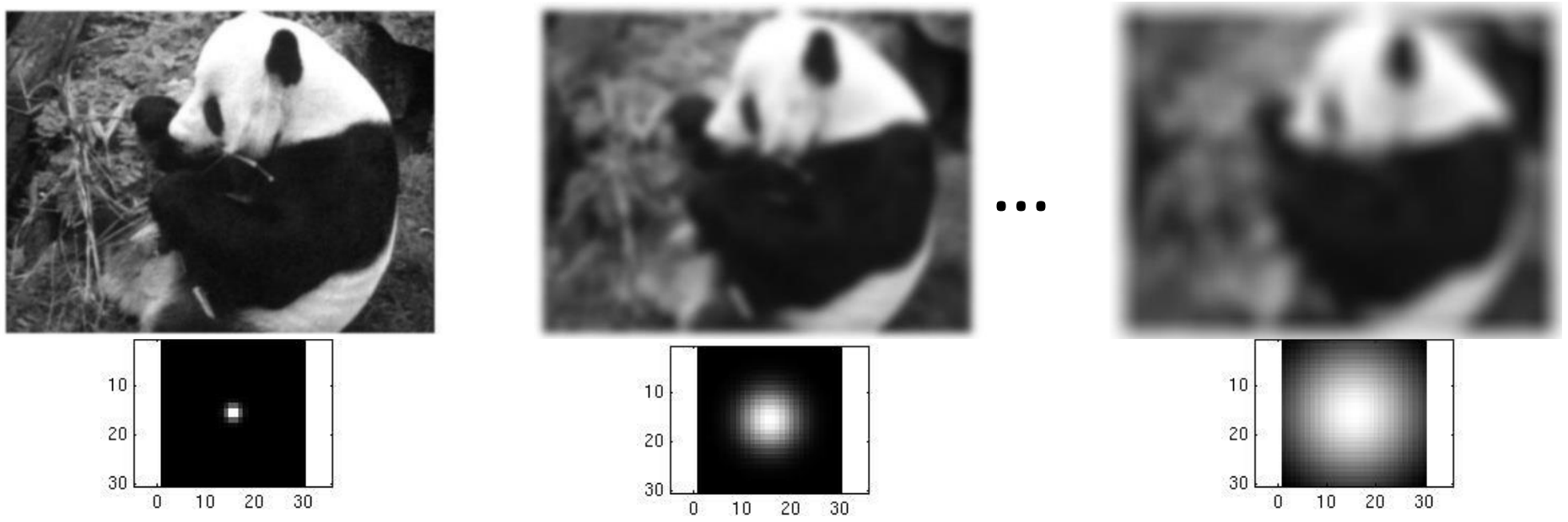
Correlation with an impulse signal

What is the result of correlating the impulse signal (image) *F* with the arbitrary kernel *H*?



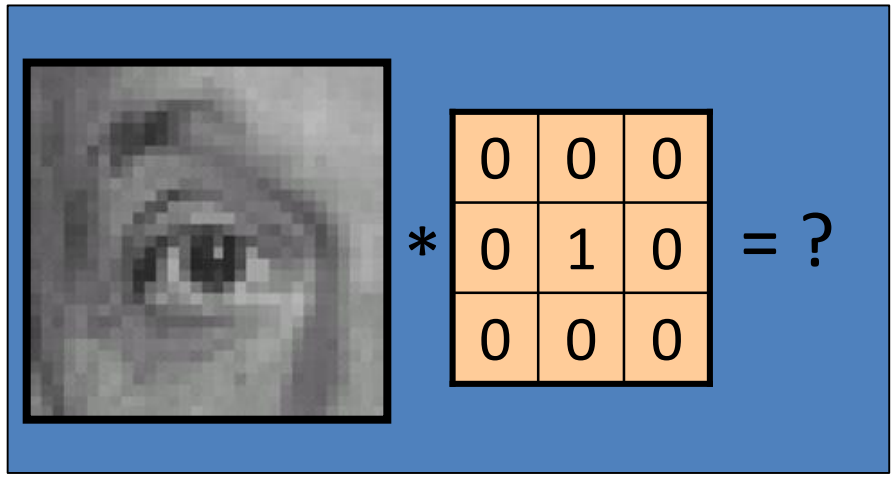
Smoothing with a Gaussian

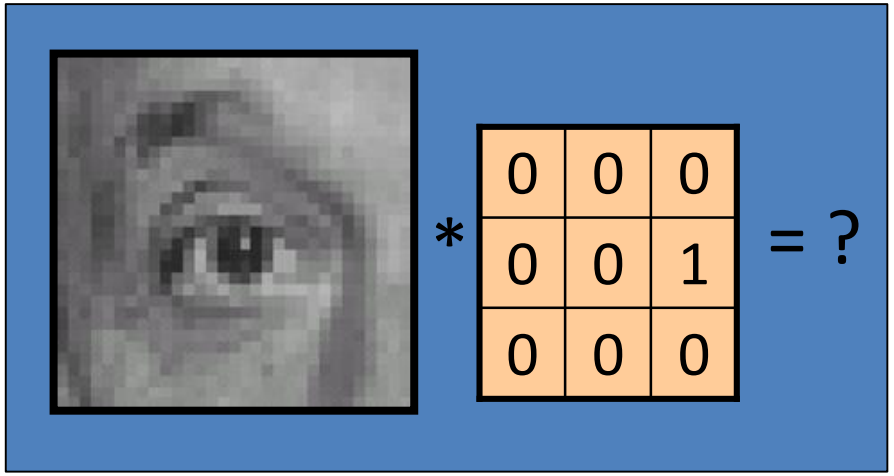
Parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.

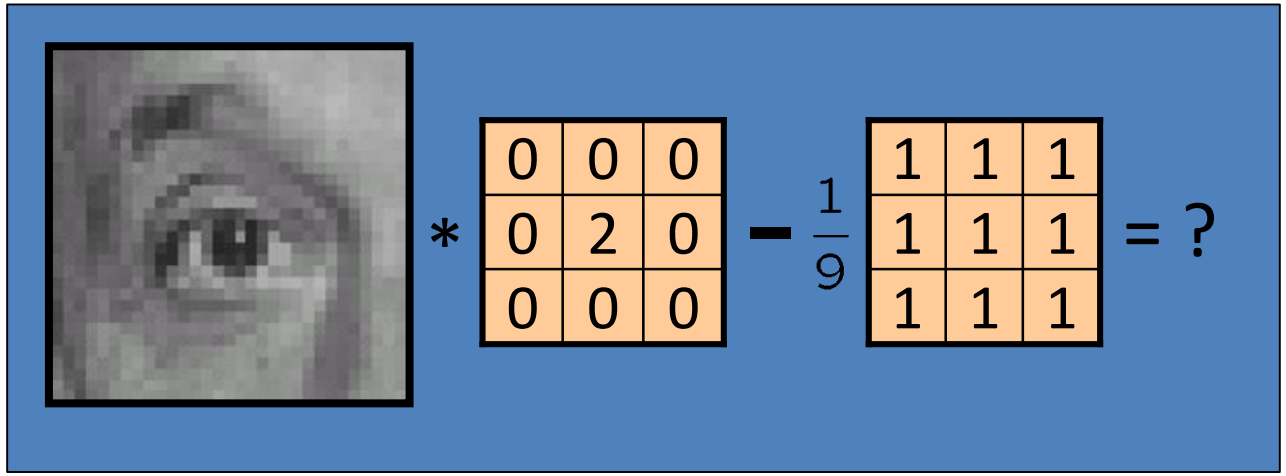



```
for sigma=1:3:10
   h = fspecial('gaussian', fsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

Predict the filtered outputs







Practice with linear filters

Original

$$* \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad = \quad ?$$

Source: D. Lowe

Practice with linear filters

Original

*

0	0	0
0	1	0
0	0	0

=

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

$$*\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{bmatrix} = ?$$

Source: D. Lowe

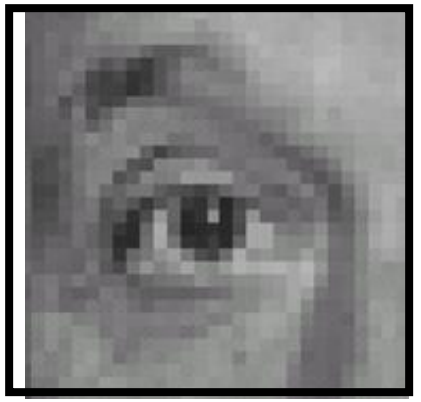
Practice with linear filters

Original

*

0	0	0
0	0	1
0	0	0

=



Shifted right
by 1 pixel with **convolution**

Shifted left
by 1 pixel with **correlation**

Source: D. Lowe

Practice with linear filters

Original

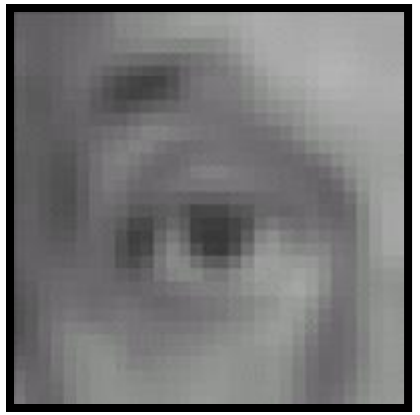
$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad = \quad ?$$

Source: D. Lowe

Practice with linear filters

Original

$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$



Blur (with a box filter)

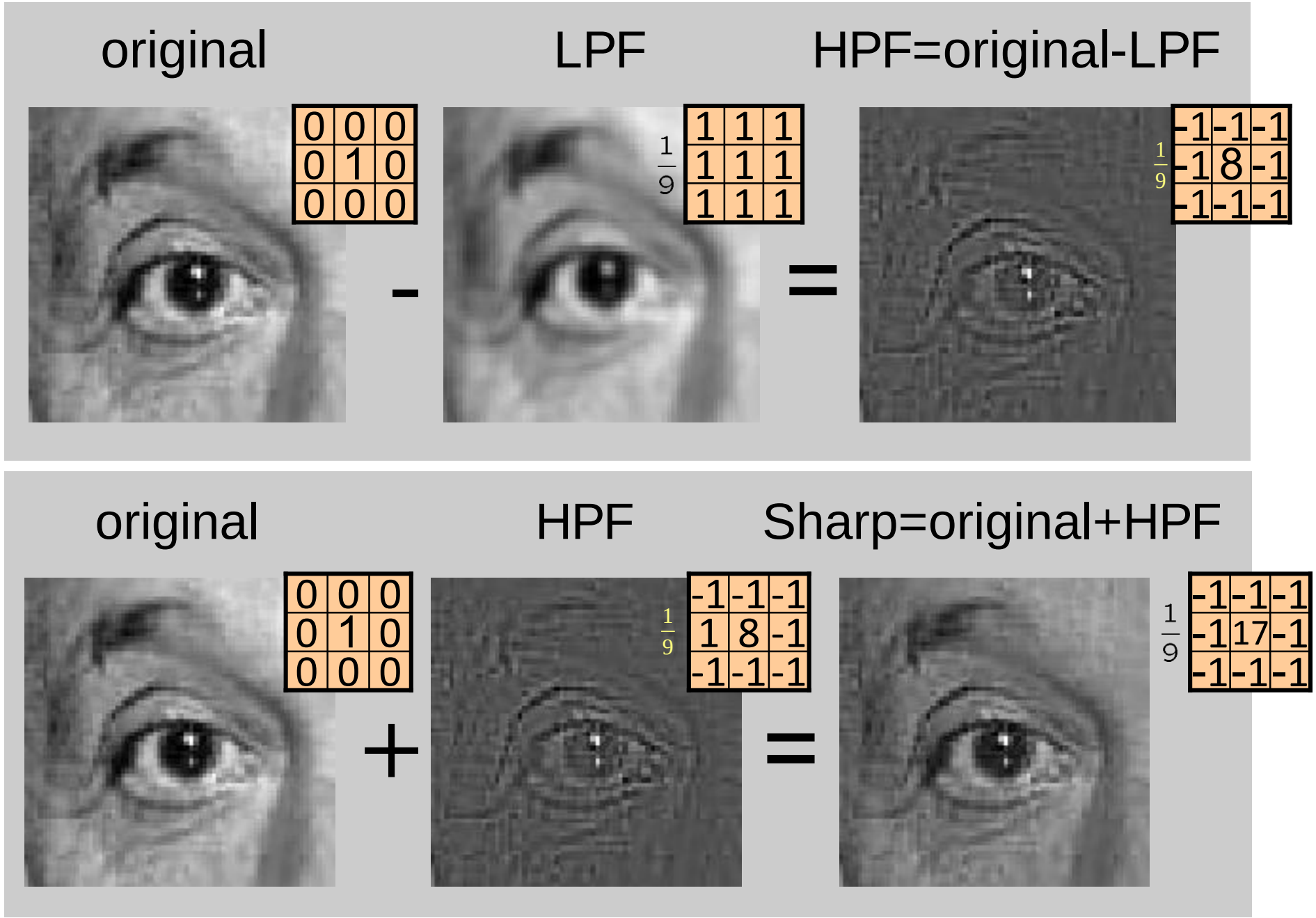
Source: D. Lowe

Practice with linear filters

Original

$$* \left(\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right) = \; ?$$

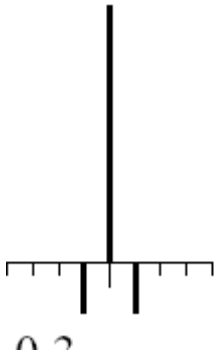
Source: D. Lowe


original
LPF
HPF=original-LPF
0 0 0
0 1 0
0 0 0
-
1/9
1 1 1
1 1 1
1 1 1
=
1/9
-1 -1 -1
-1 8 -1
-1 -1 -1
original
HPF
Sharp=original+HPF
0 0 0
0 1 0
0 0 0
+
1/9
-1 -1 -1
1 8 -1
-1 -1 -1
=
1/9
-1 -1 -1
-1 17 -1
-1 -1 -1

Practice with linear filters

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

Original

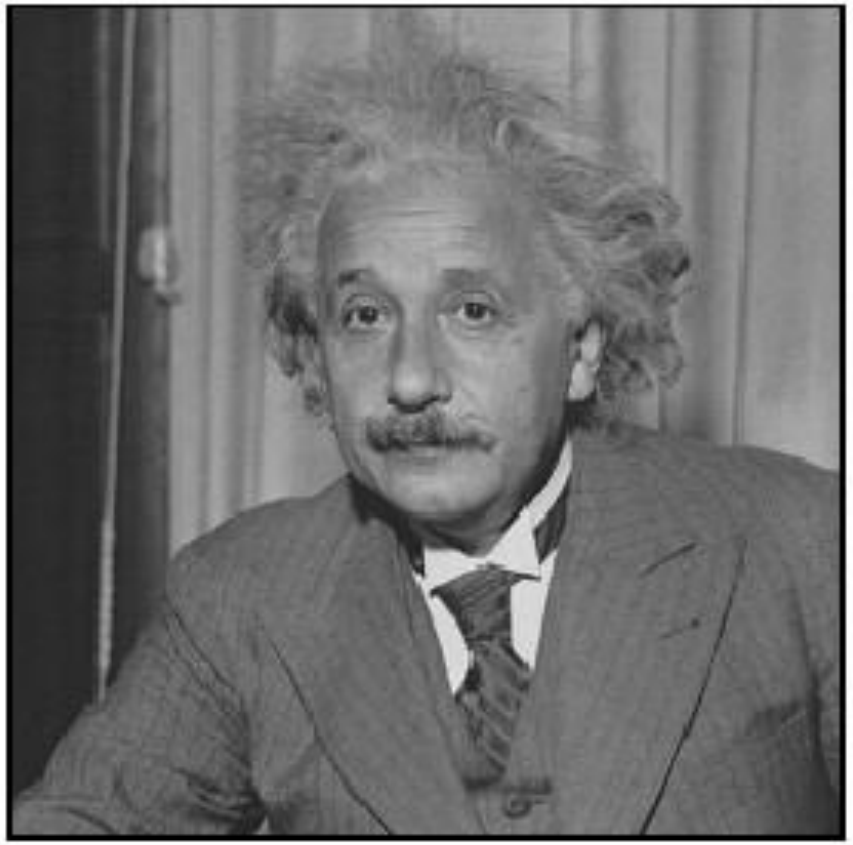


Sharpening filter

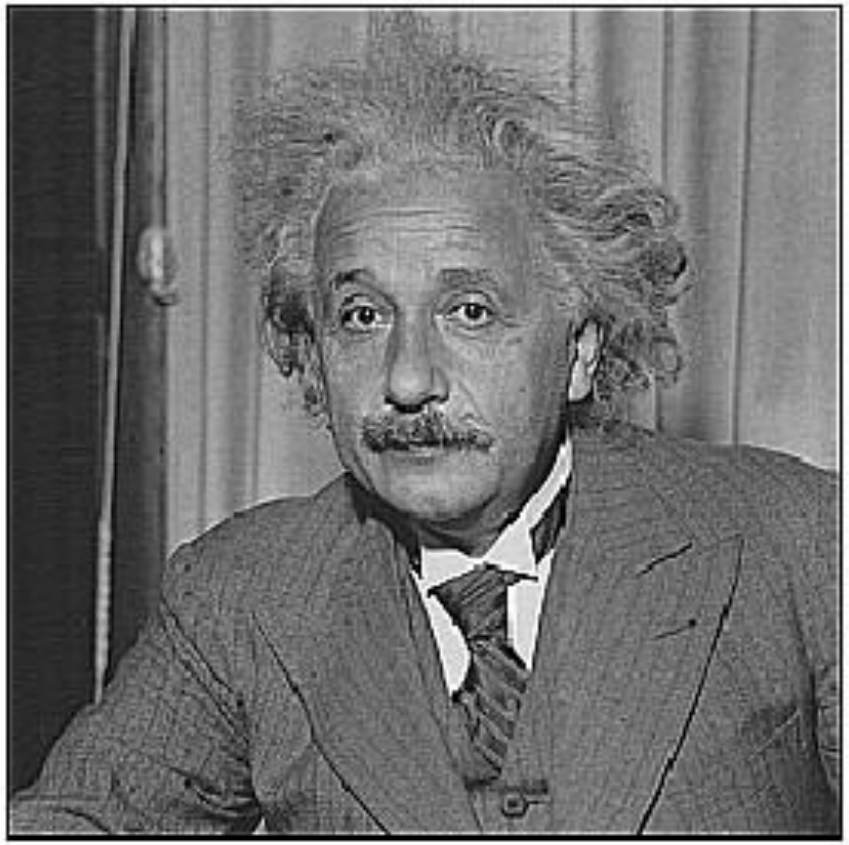
- Accentuates differences with local average

Source: D. Lowe

Filtering examples: sharpening

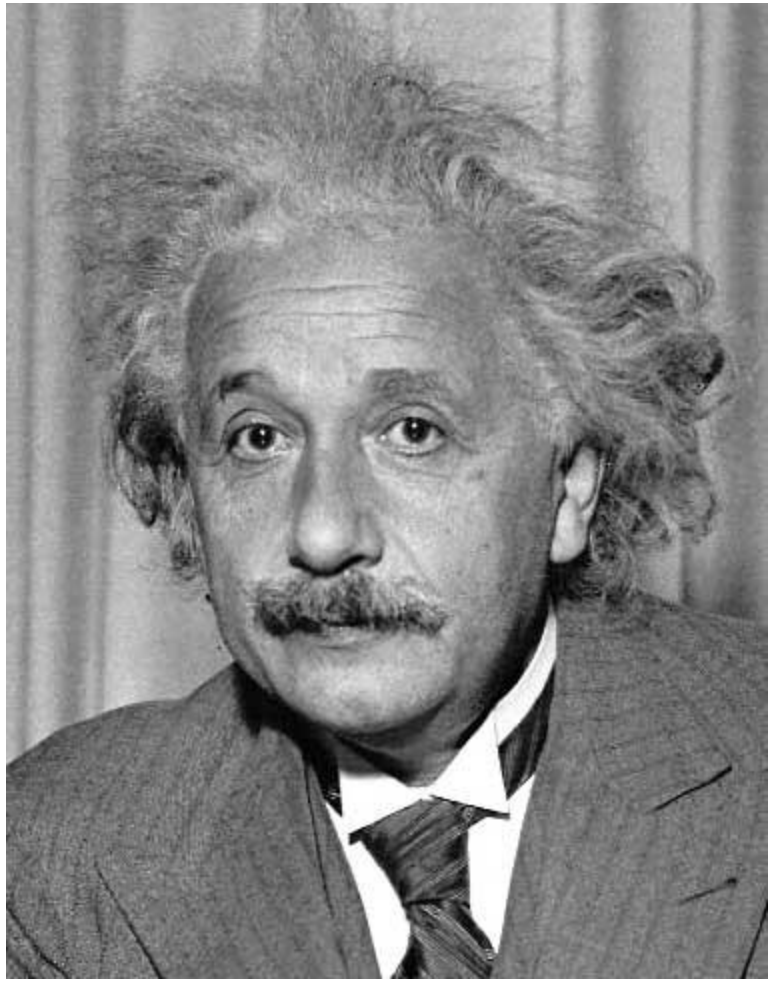


before



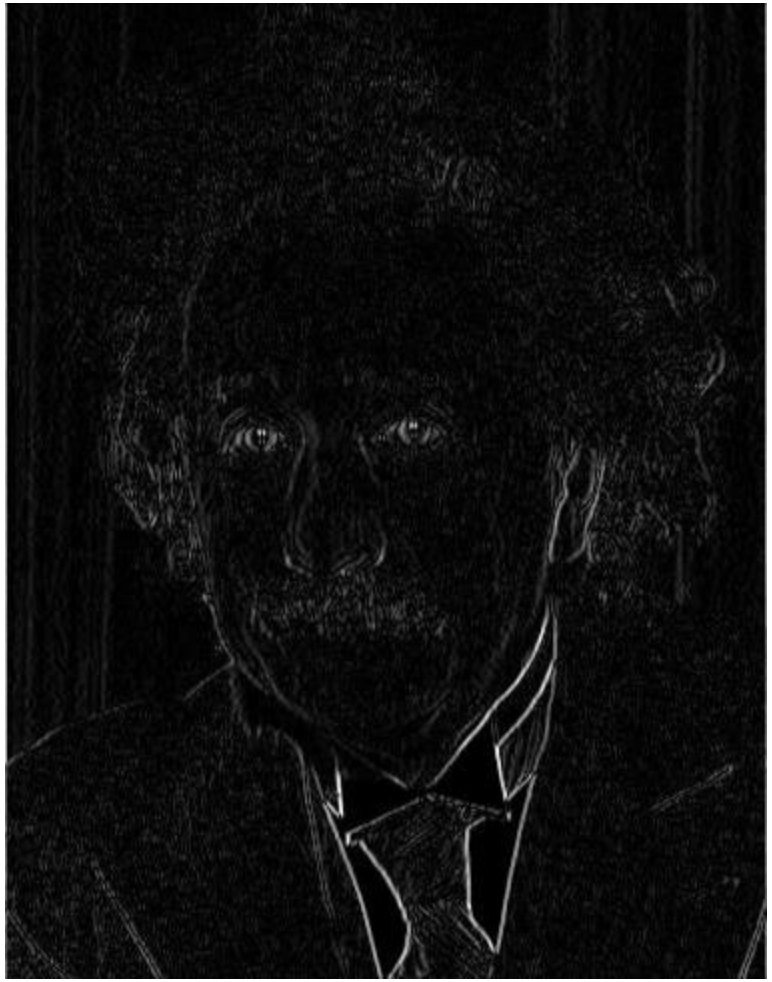
after

Other filters



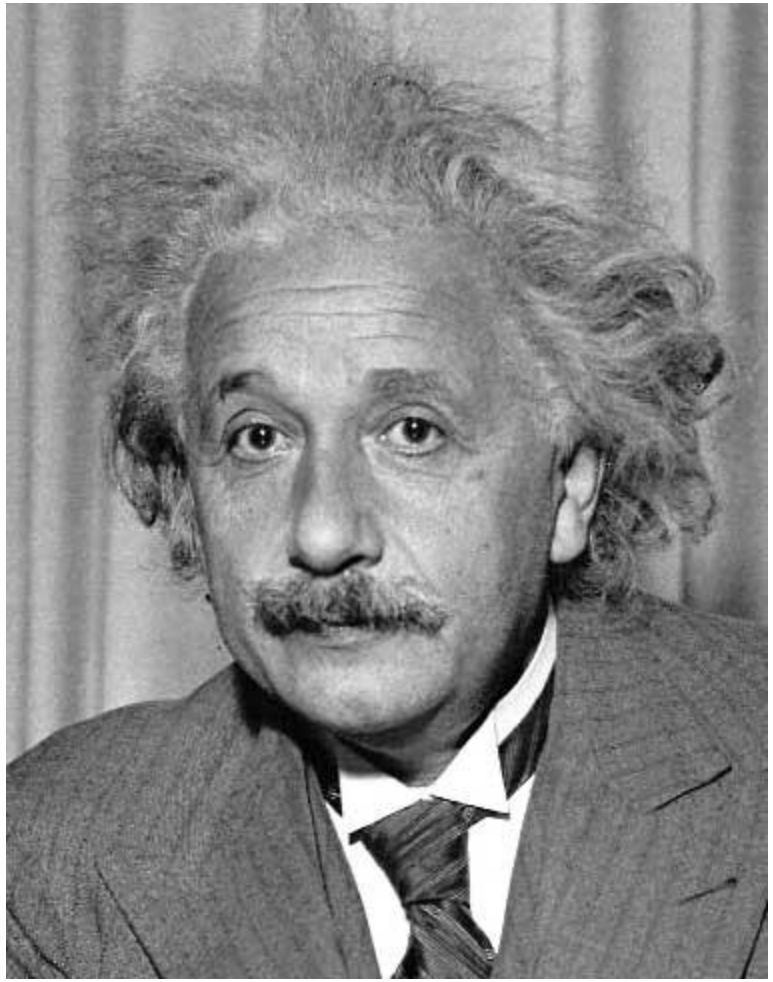
1	0	-1
2	0	-2
1	0	-1

Sobel



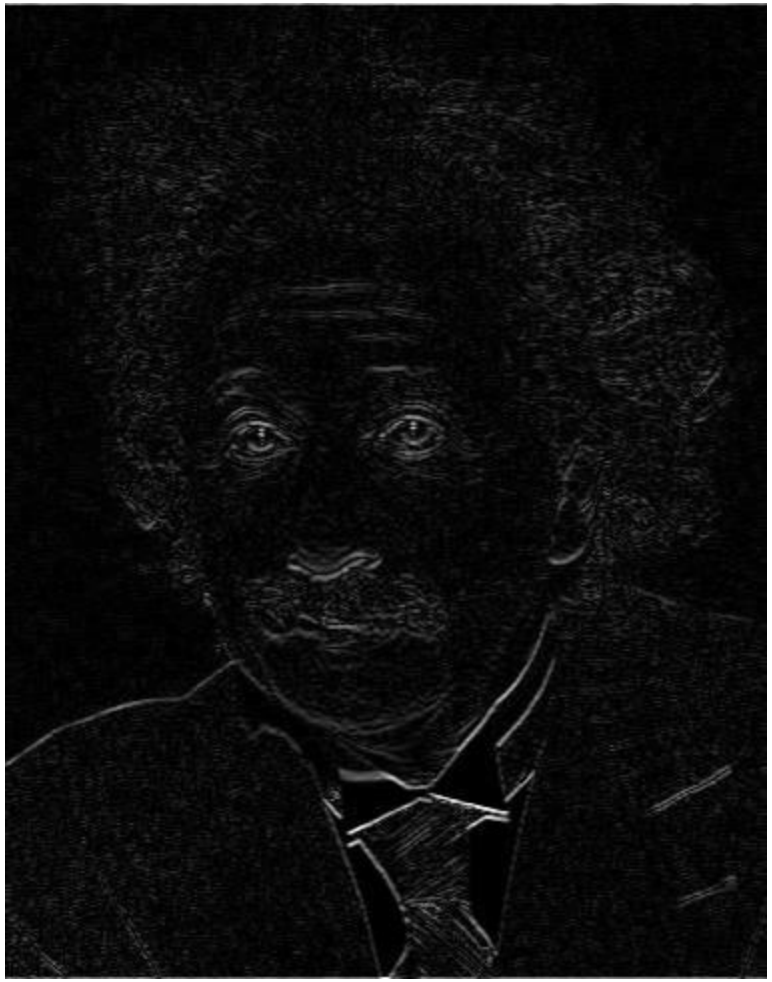
Vertical Edge
(absolute value)

Other filters



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

Sobel



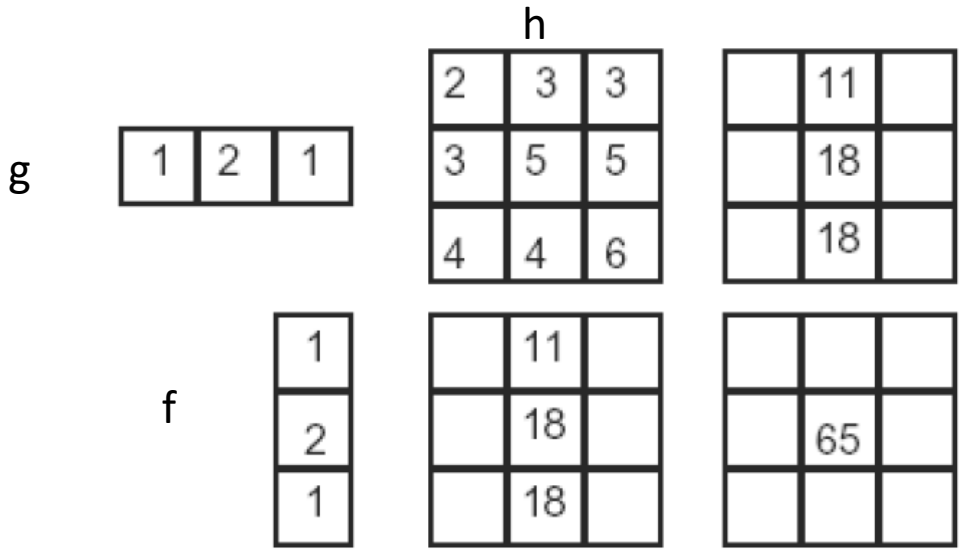
Horizontal Edge (absolute value)

Separability

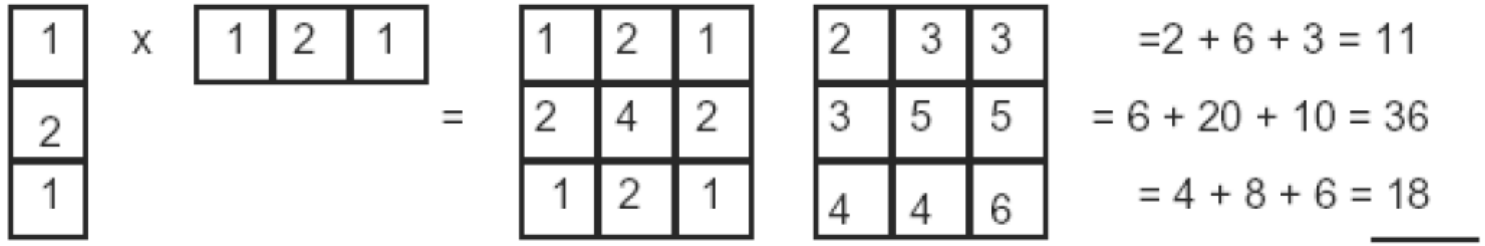
- In some cases, filter is separable, and we can factor into two steps:
  - Convolve all rows
  - Convolve all columns

Separability

- In some cases, filter is separable, and we can factor into two steps: e.g.,



What is the computational complexity advantage for a separable filter of size k x k, in terms of number of operations per output pixel?



f * (g * h) = (f * g) * h

Edge detection

- **Goal**: map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**

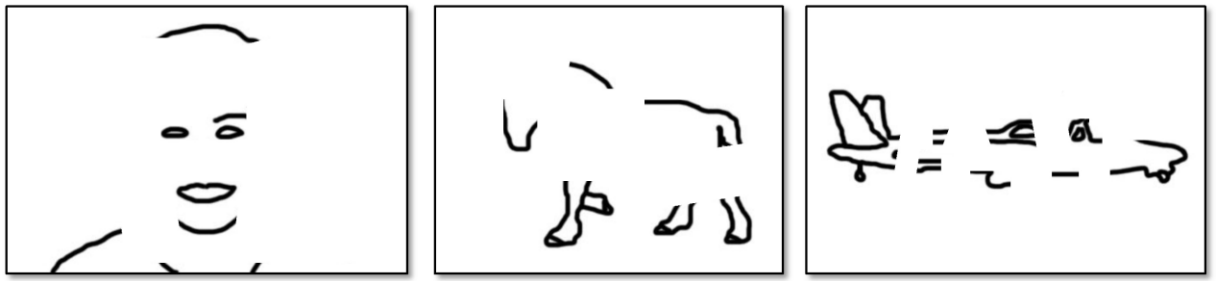
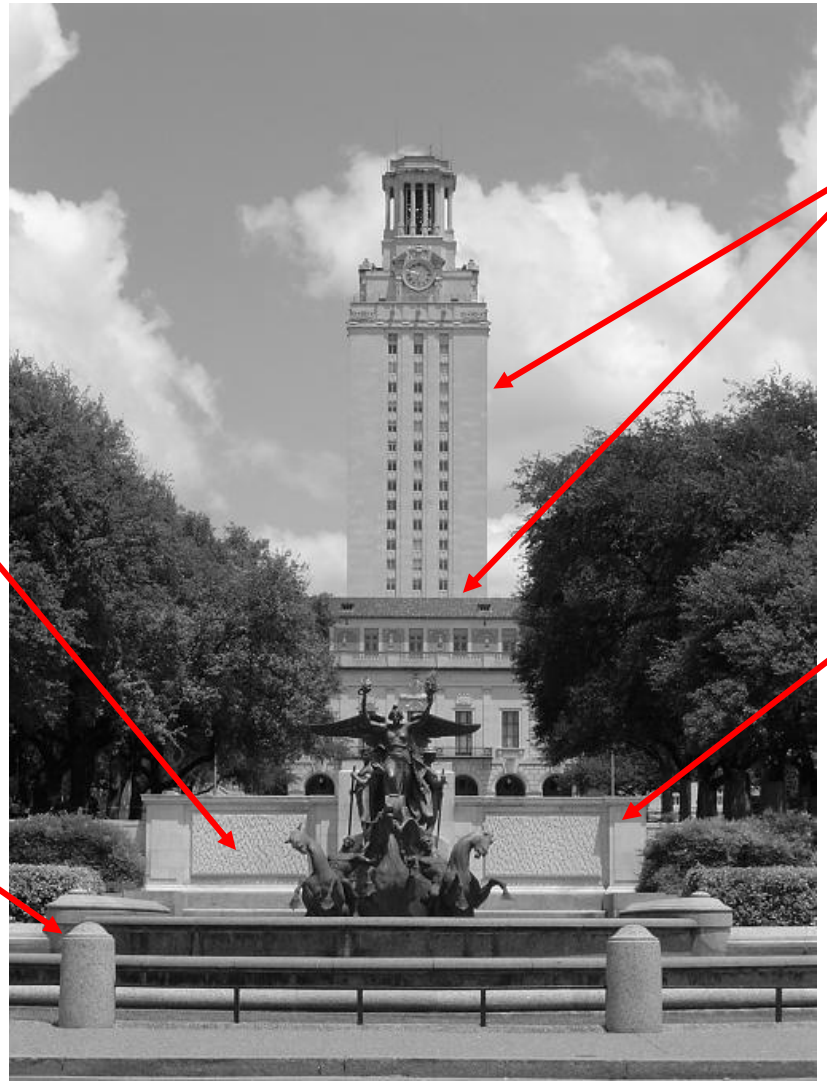


Figure from J. Shotton et al., PAMI 2007

- **Main idea**: look for strong gradients, post-process

What can cause an edge?



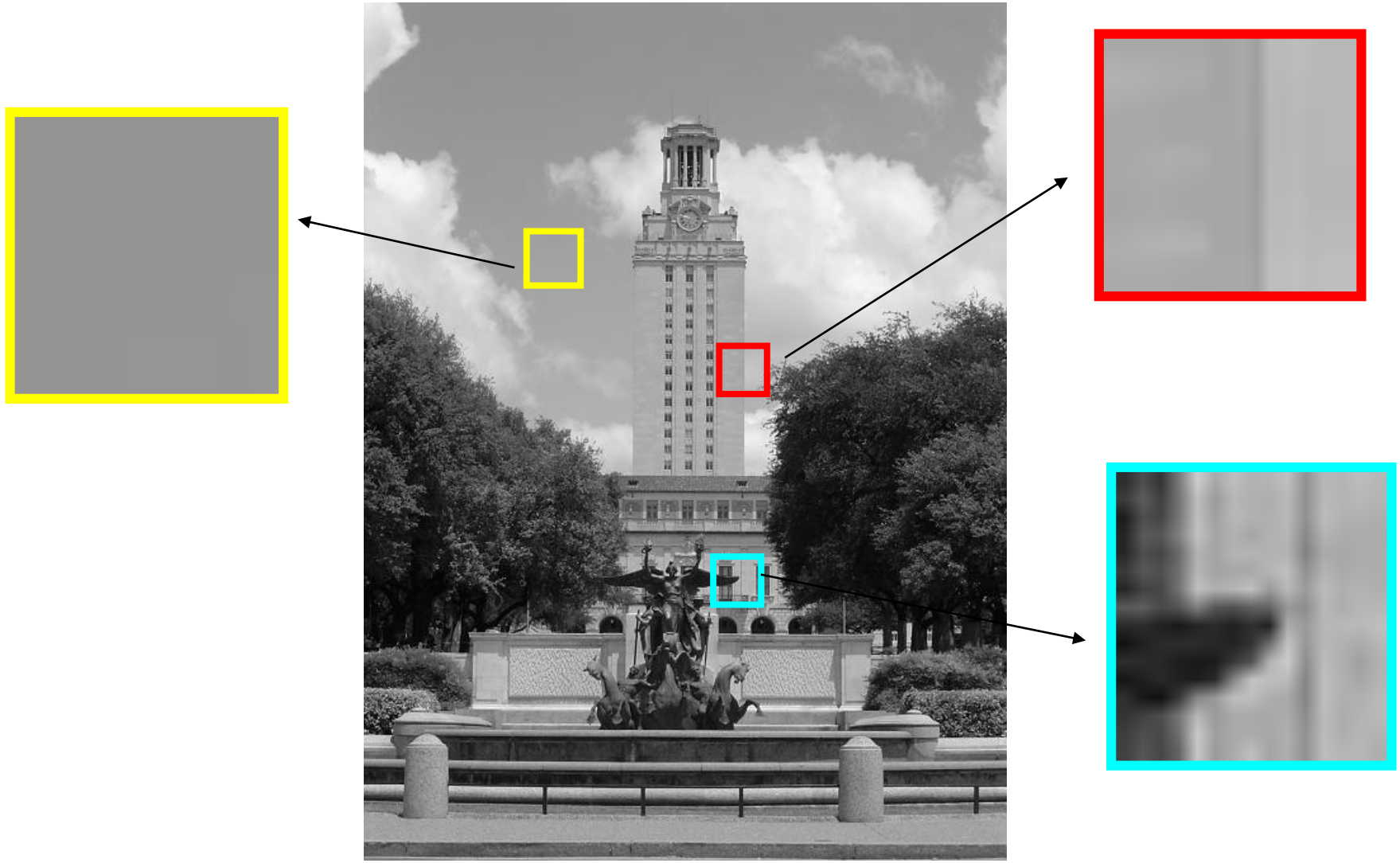
Reflectance change: appearance information, texture

Depth discontinuity: object boundary

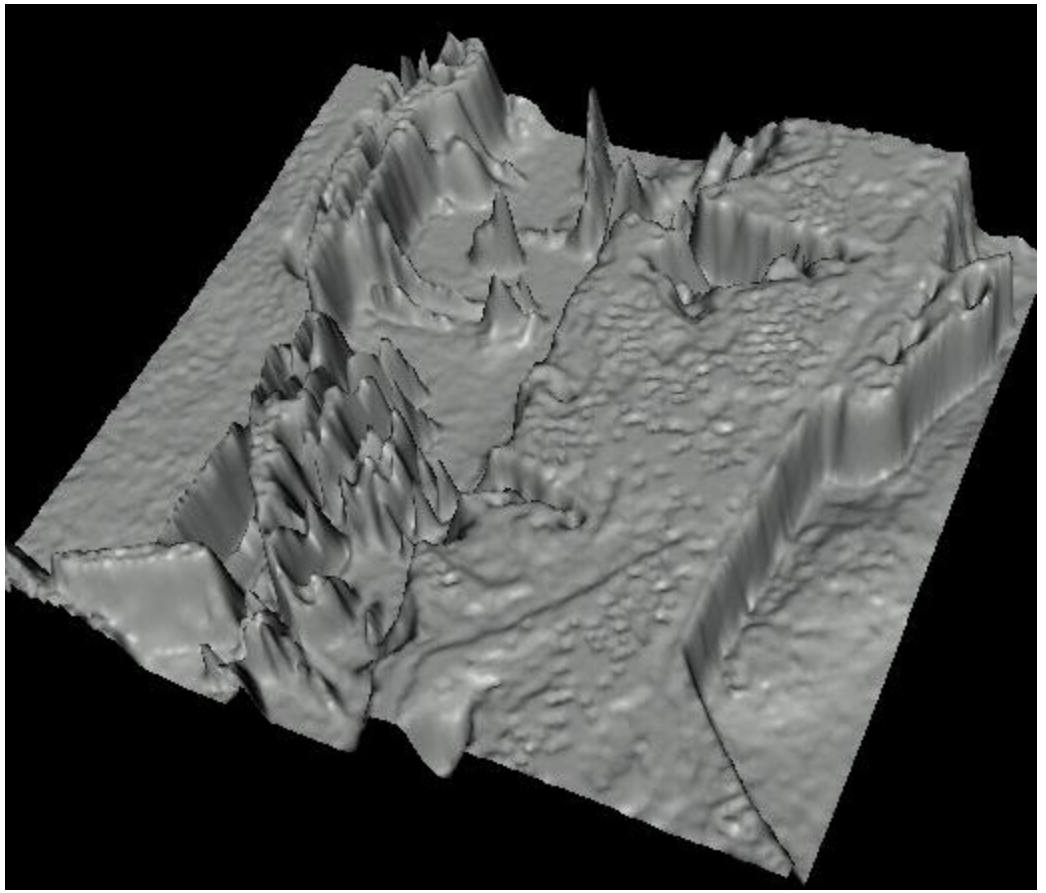
Change in surface orientation: shape

Cast shadows

Contrast and invariance



Recall : Images as functions

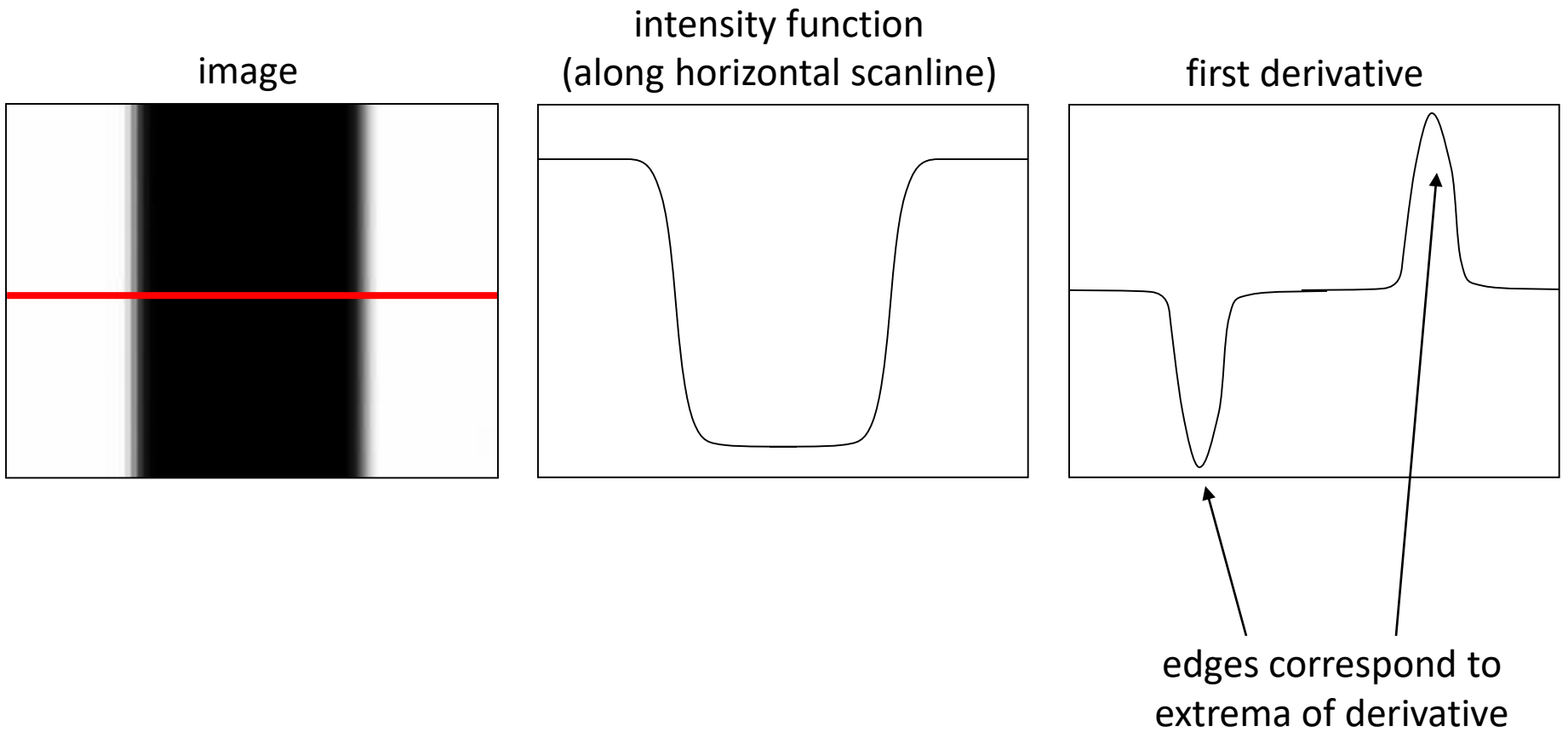


- Edges look like steep cliffs

Source: S. Seitz

Derivatives and edges

An edge is a place of rapid change in the image intensity function.



Source: L. Lazebnik

Differentiation and convolution

For 2D function, f(x,y), the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

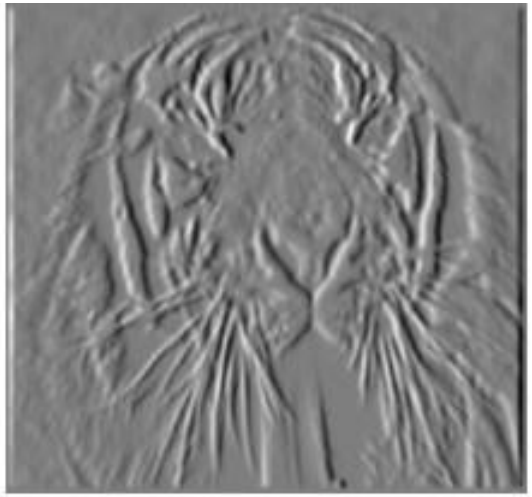
For discrete data, we can approximate using finite differences:

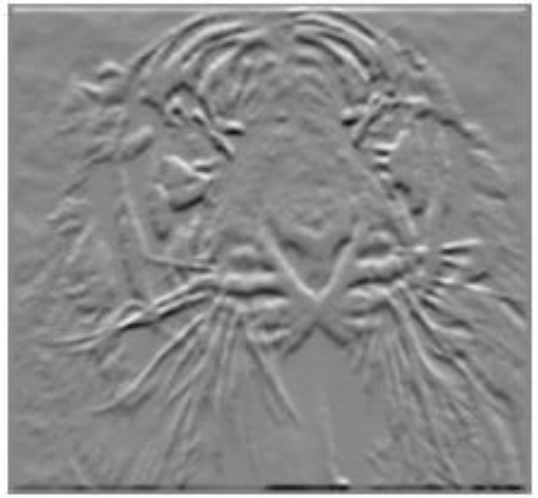
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

To implement above as convolution, what would be the associated filter?

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$





$$\frac{\partial f(x,y)}{\partial y}$$

-1	1

-1
1

?

or

1
-1

Which shows changes with respect to x?

(showing flipped filters)

Assorted finite difference filters

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```

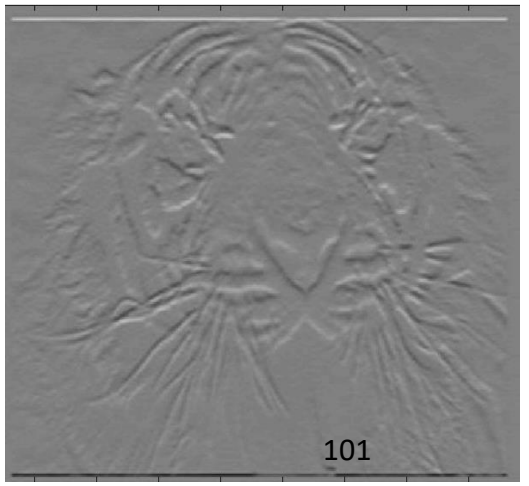
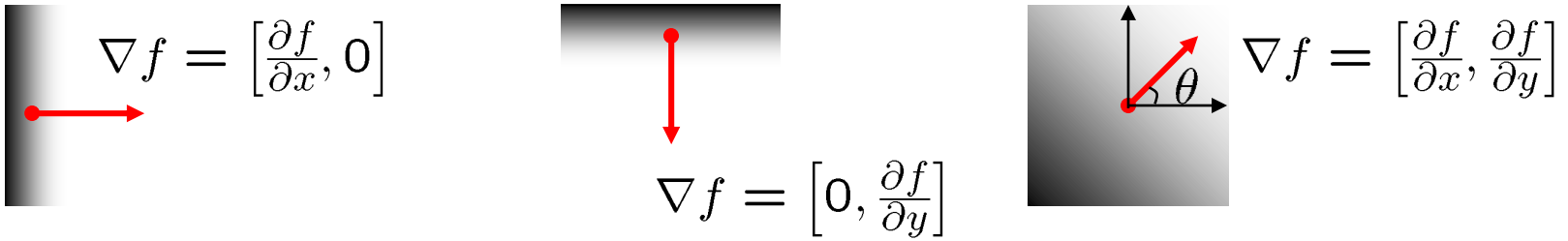


Image gradient

The gradient of an image:

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

The gradient points in the direction of most rapid change in intensity



The gradient direction (orientation of edge normal) is given by:

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

The *edge strength* is given by the gradient magnitude

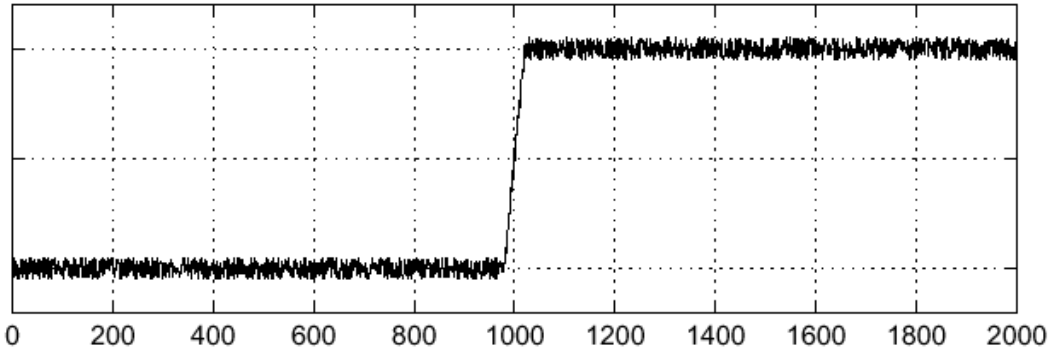
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Effects of noise

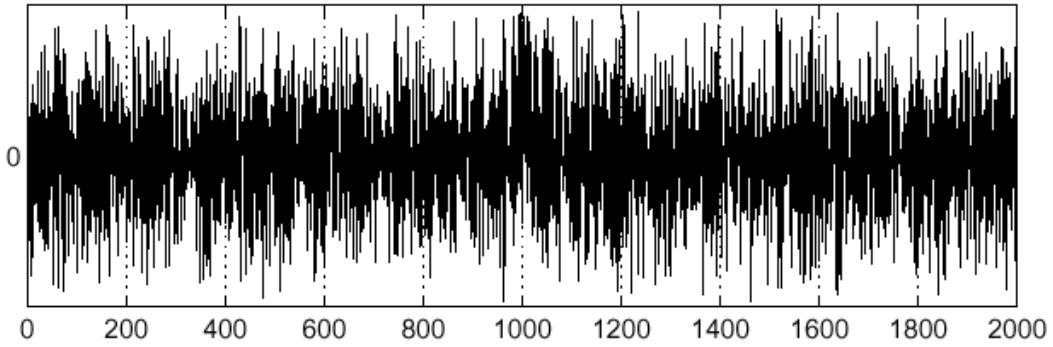
Consider a single row or column of the image

– Plotting intensity as a function of position gives a signal

$f(x)$



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



Where is the edge?

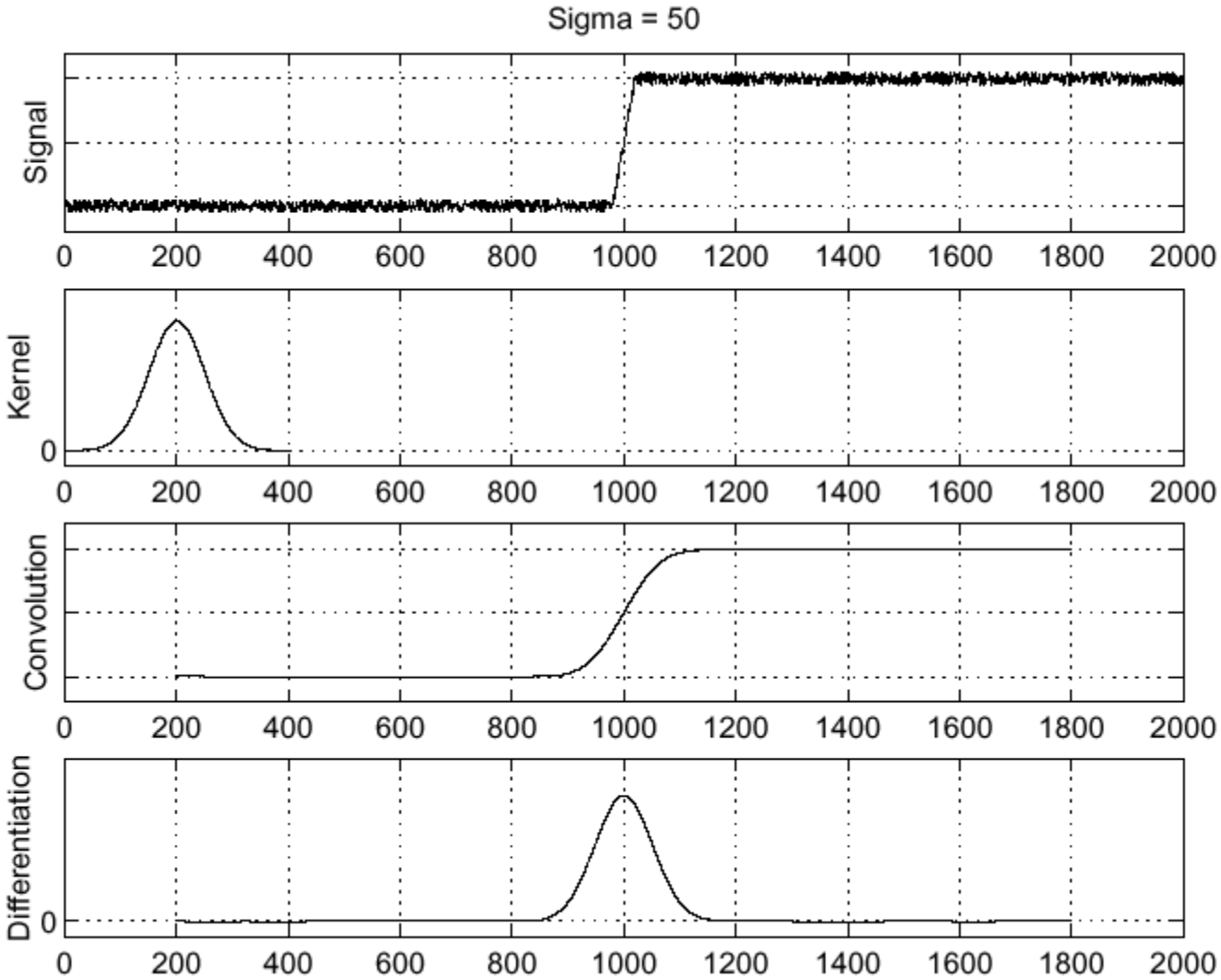
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

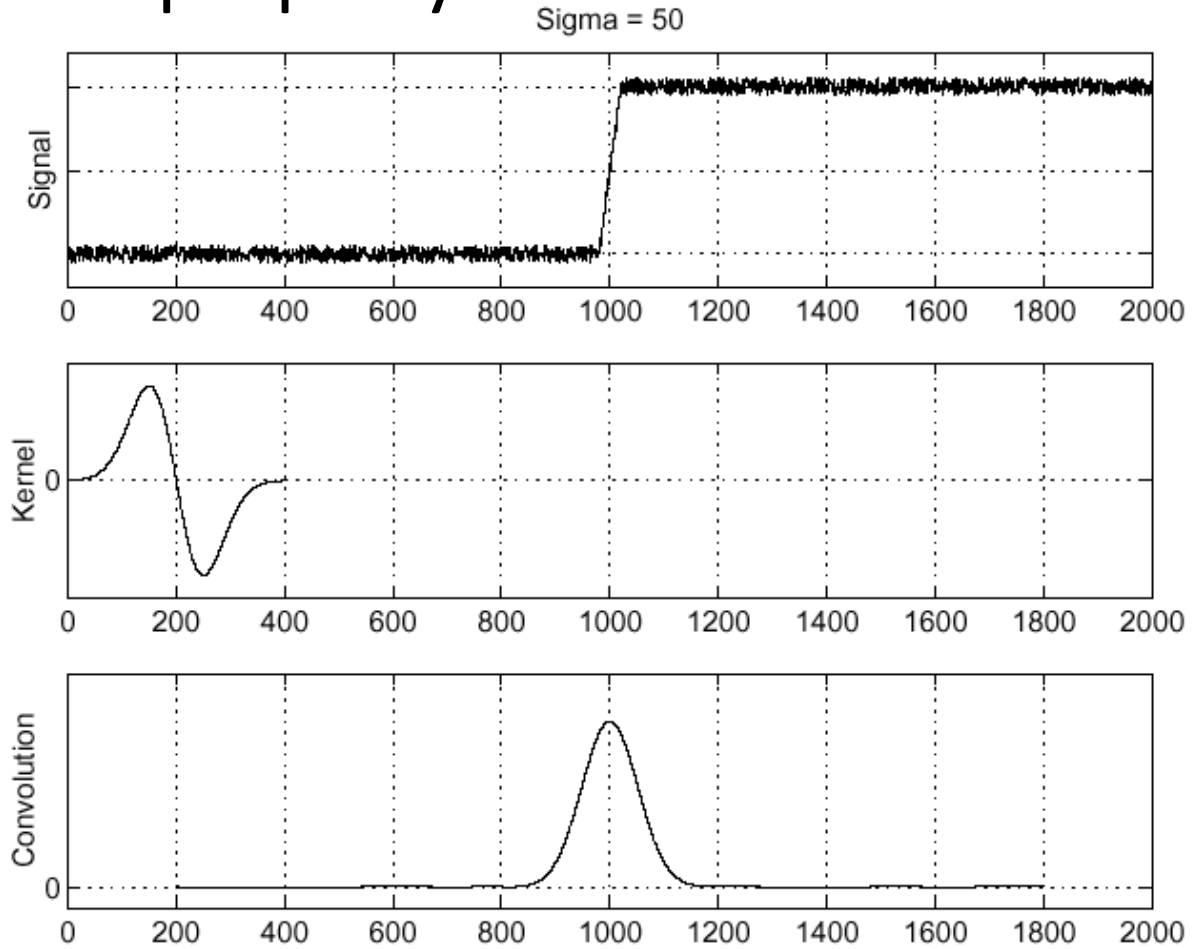
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

Differentiation property of convolution.

f

$\frac{\partial}{\partial x}h$

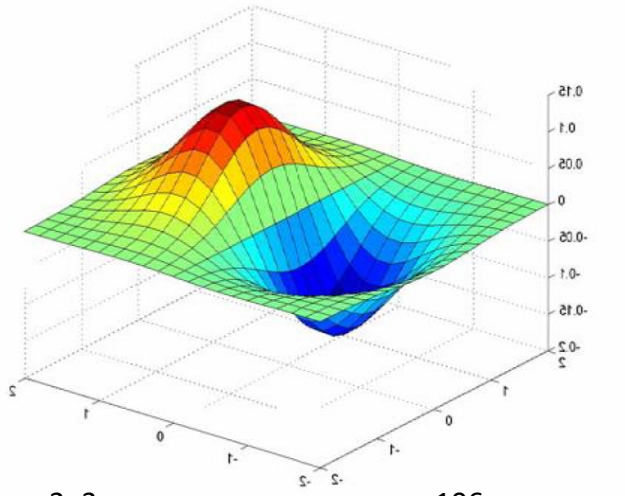
$(\frac{\partial}{\partial x}h) \star f$



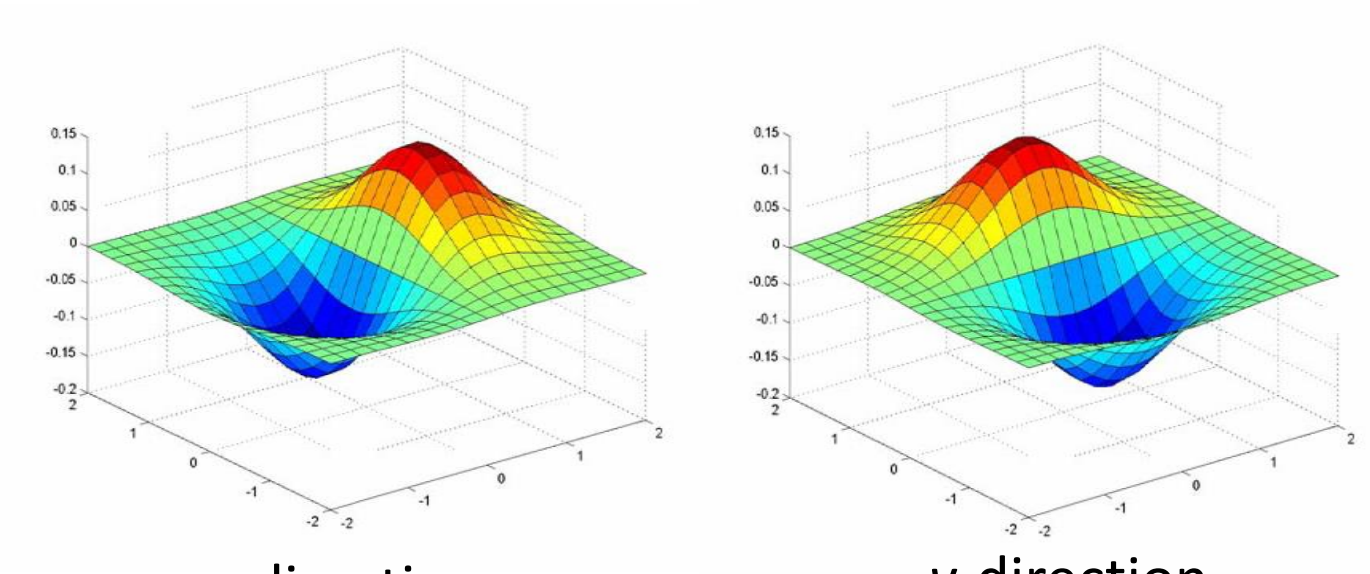
Derivative of Gaussian filter

$$(I * g) * h = I * (g * h)$$

$$\begin{bmatrix} 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0219 & 0.0983 & 0.1621 & 0.0983 & 0.0219 \\ 0.0133 & 0.0596 & 0.0983 & 0.0596 & 0.0133 \\ 0.0030 & 0.0133 & 0.0219 & 0.0133 & 0.0030 \end{bmatrix} * \begin{bmatrix} 1 & -1 \end{bmatrix}$$

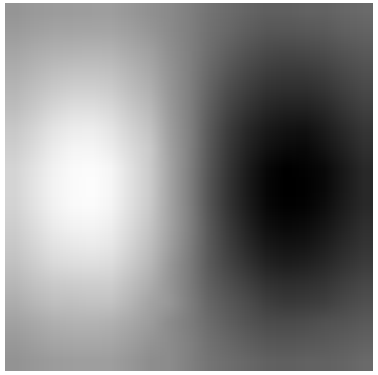


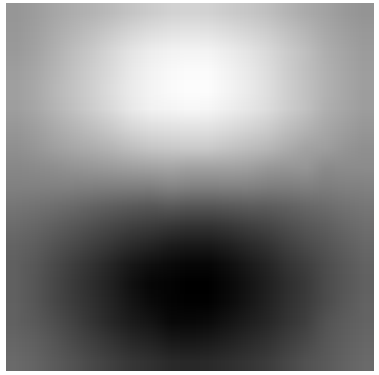
Derivative of Gaussian filters



x-direction

y-direction



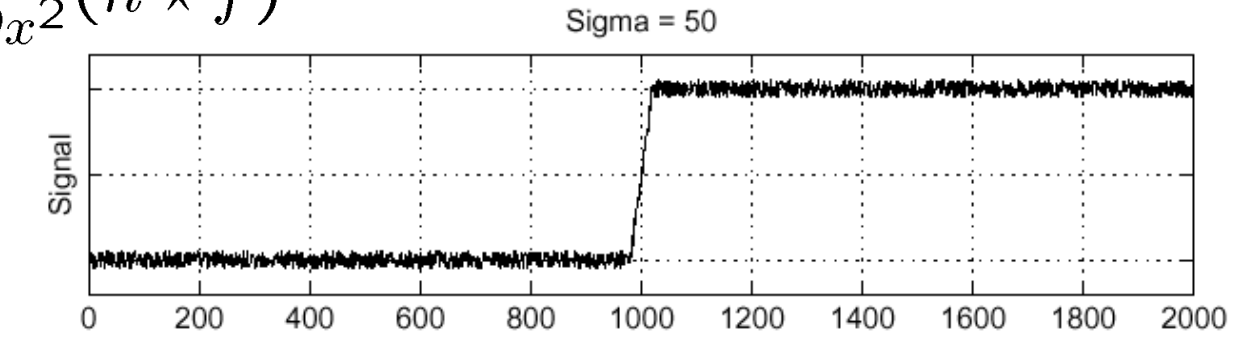


Source: L. Lazebnik

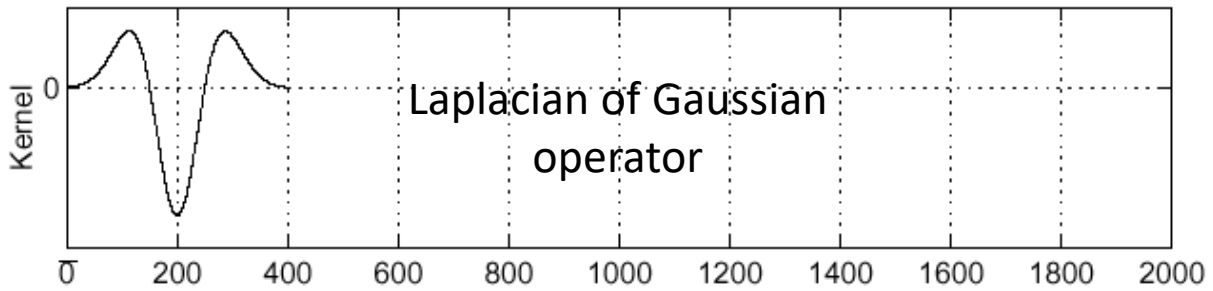
Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

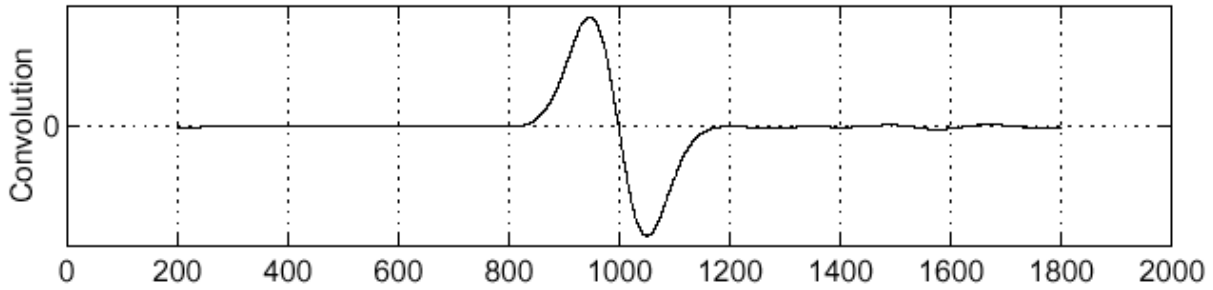
f



$\frac{\partial^2}{\partial x^2}h$



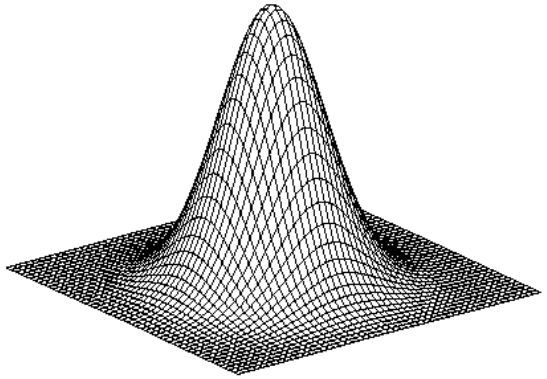
$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge?

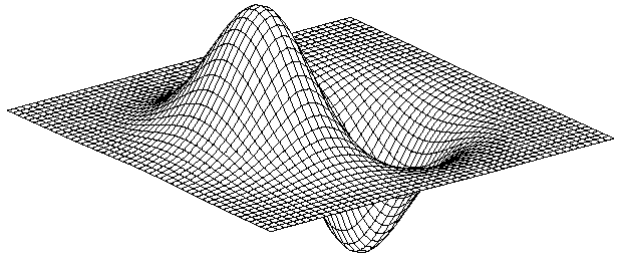
Zero-crossings of bottom graph

2D edge detection filters



Gaussian

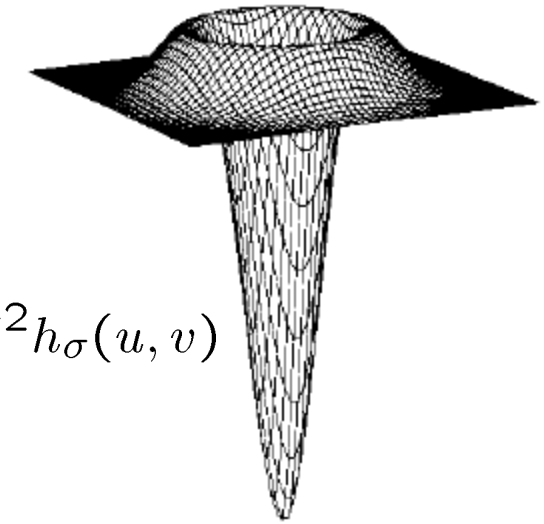
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

- $\nabla^2$is the Laplacian operator:

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

Mask properties

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter
- Derivatives
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast
- Filters act as templates
  - Highest response for regions that “look the most like the filter”
  - Dot product as correlation

Gradients -> edges

Primary edge detection steps:

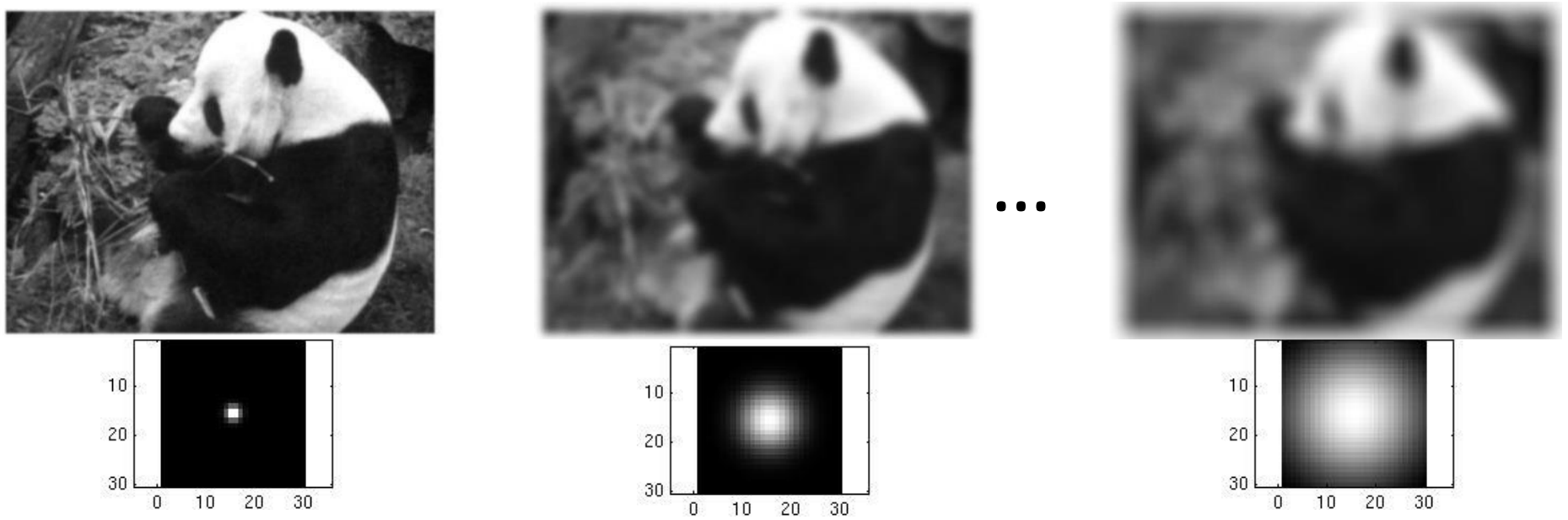
1. Smoothing: suppress noise
2. Edge enhancement: filter for contrast
3. Edge localization

   Determine which local maxima from filter output are actually edges vs. noise

   - Threshold, Thin

Smoothing with a Gaussian

Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



Effect of σ on derivatives

σ = 1 pixel

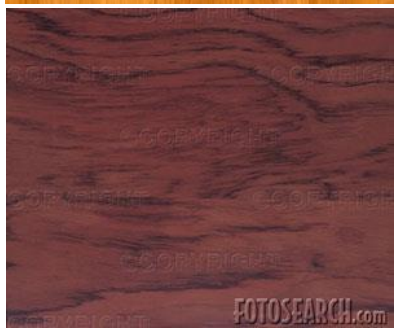
σ = 3 pixels

The apparent structures differ depending on Gaussian's scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

So, what scale to choose?

It depends what we're looking for.



Too fine of a scale…can't see the forest for the trees.

Too coarse of a scale…can't tell the maple grain from the cherry.

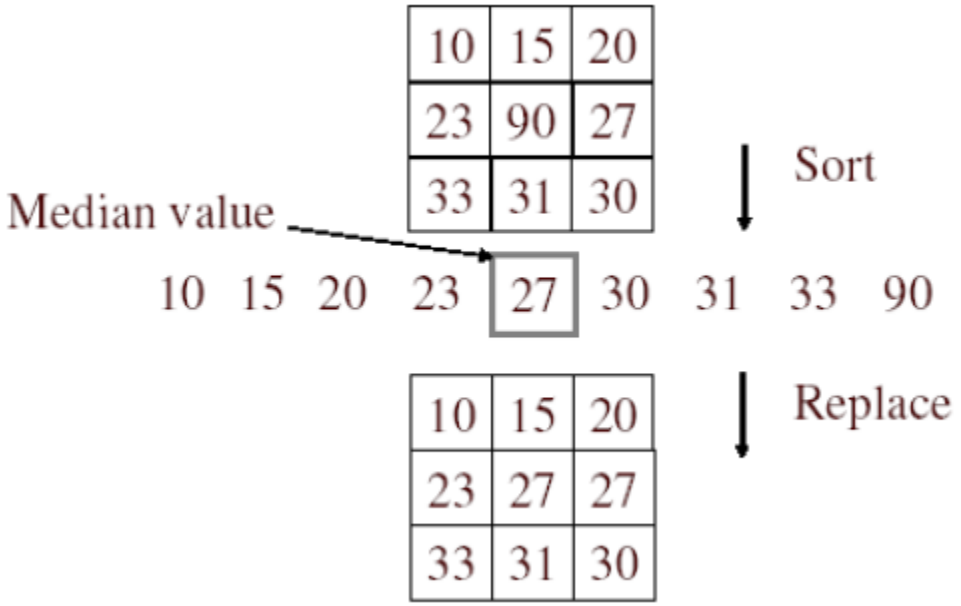
Effect of smoothing filters

5x5

Additive Gaussian noise

Salt and pepper noise

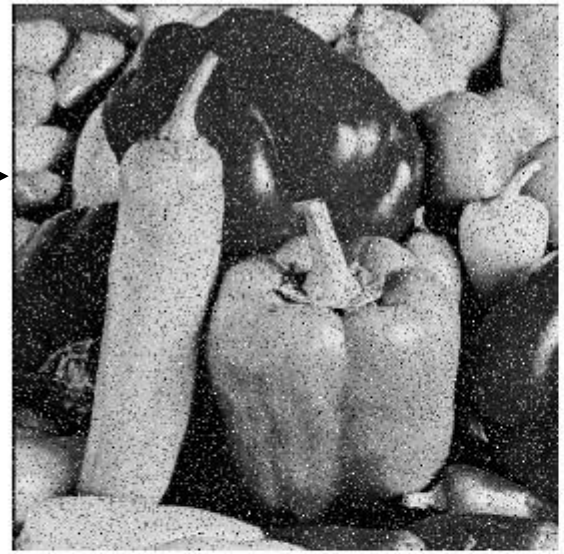
Median filter (Not a linear operation)



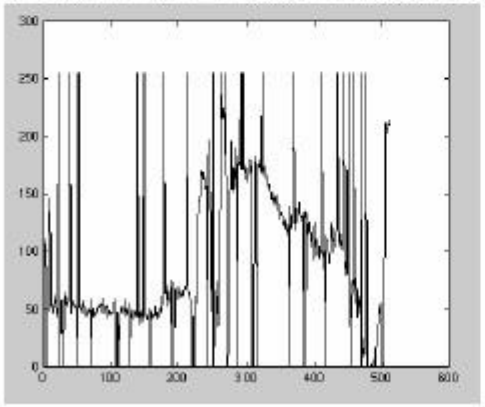
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Not a linear operation
- Does not use convolution

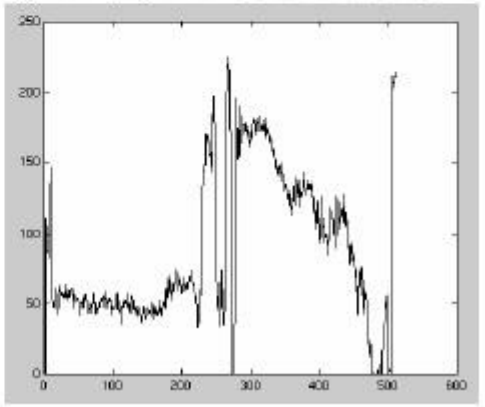
Median filter

Salt and pepper noise



Median filtered



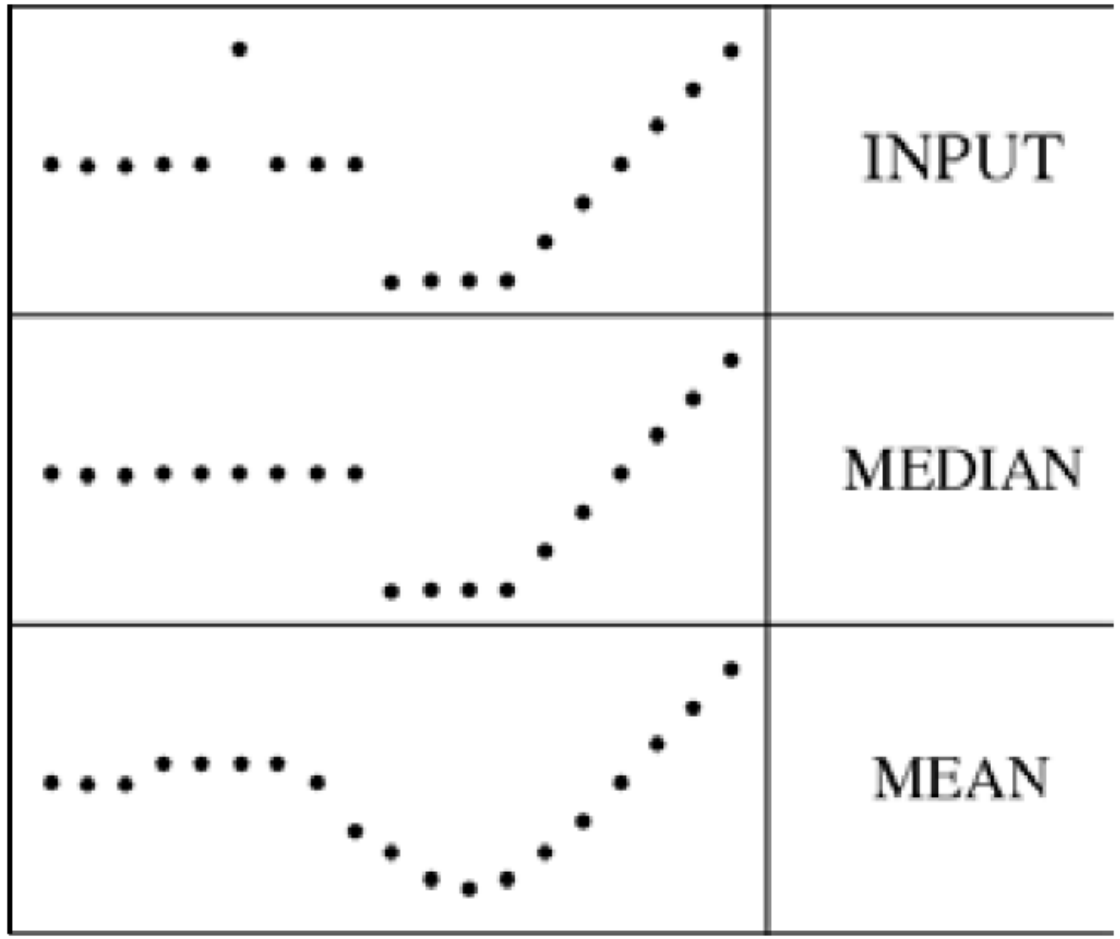


Plots of a row of the image

Source: M. Hebert

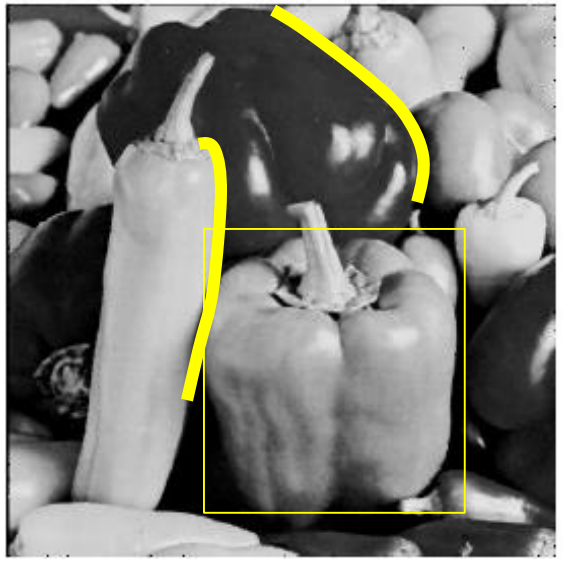
Median filter

- Median filter is edge preserving



Filters for features

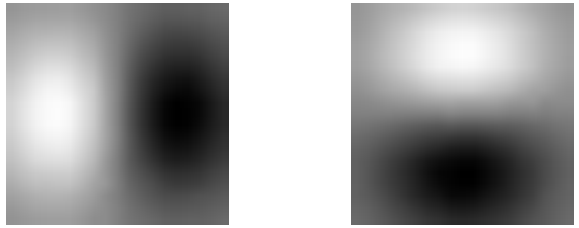
- Previously, thinking of filtering as a way to remove or reduce **noise**
- Now, consider how filters will allow us to abstract higher-level "**features**".
  - Map raw pixels to an intermediate representation that will be used for subsequent processing
  - Goal: reduce amount of data, discard redundancy, preserve what's useful



Template matching

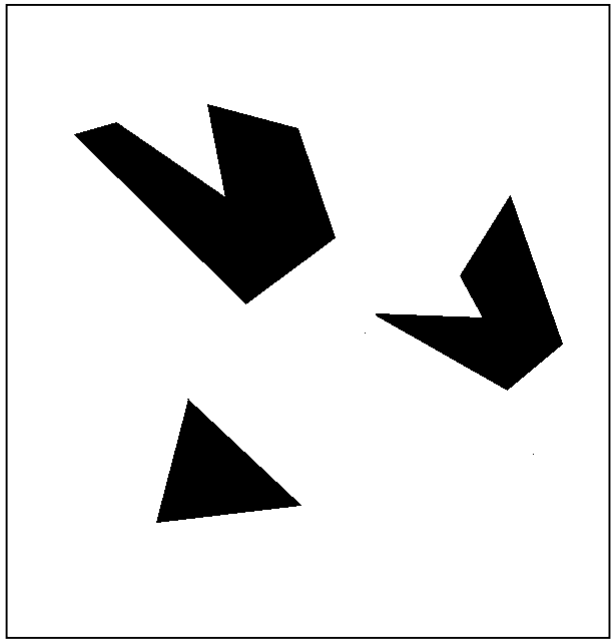
- Filters as **templates**:

  Note that filters look like the effects they are intended to find --- “matched filters”

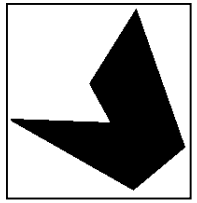


- Use normalized cross-correlation score to find a given pattern (template) in the image.
  - Szeliski Eq. 8.11
- Normalization needed to control for relative brightnesses.

Template matching



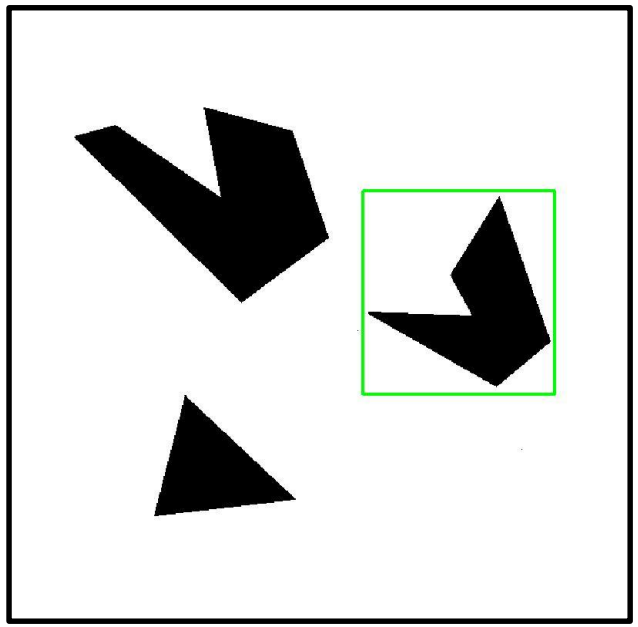
Scene



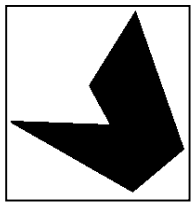
Template (mask)

A toy example

Template matching

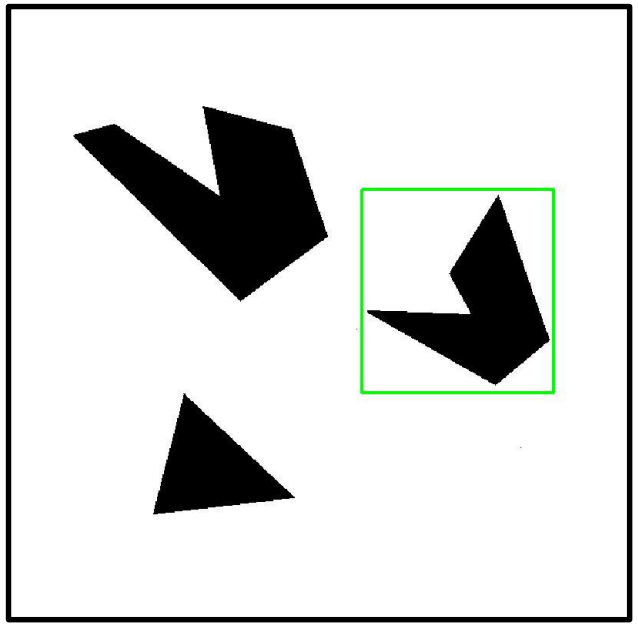


Detected template

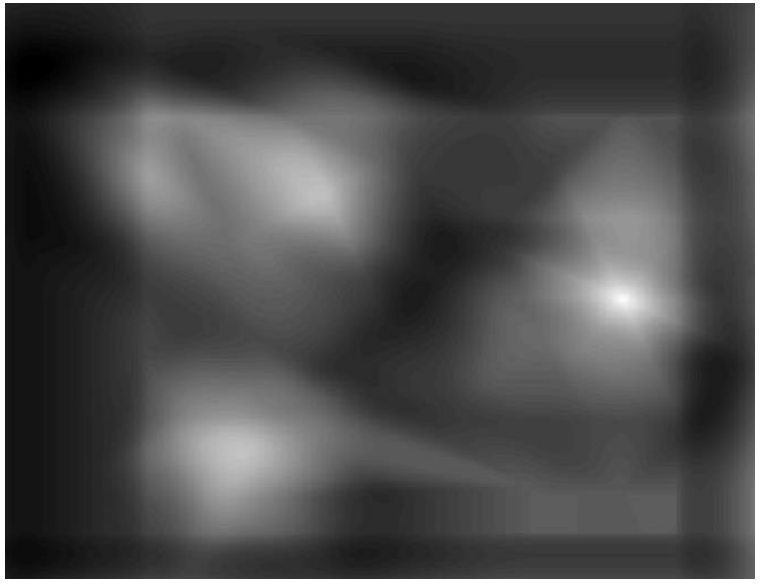


Template

Template matching



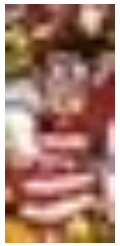
Detected template



Correlation map

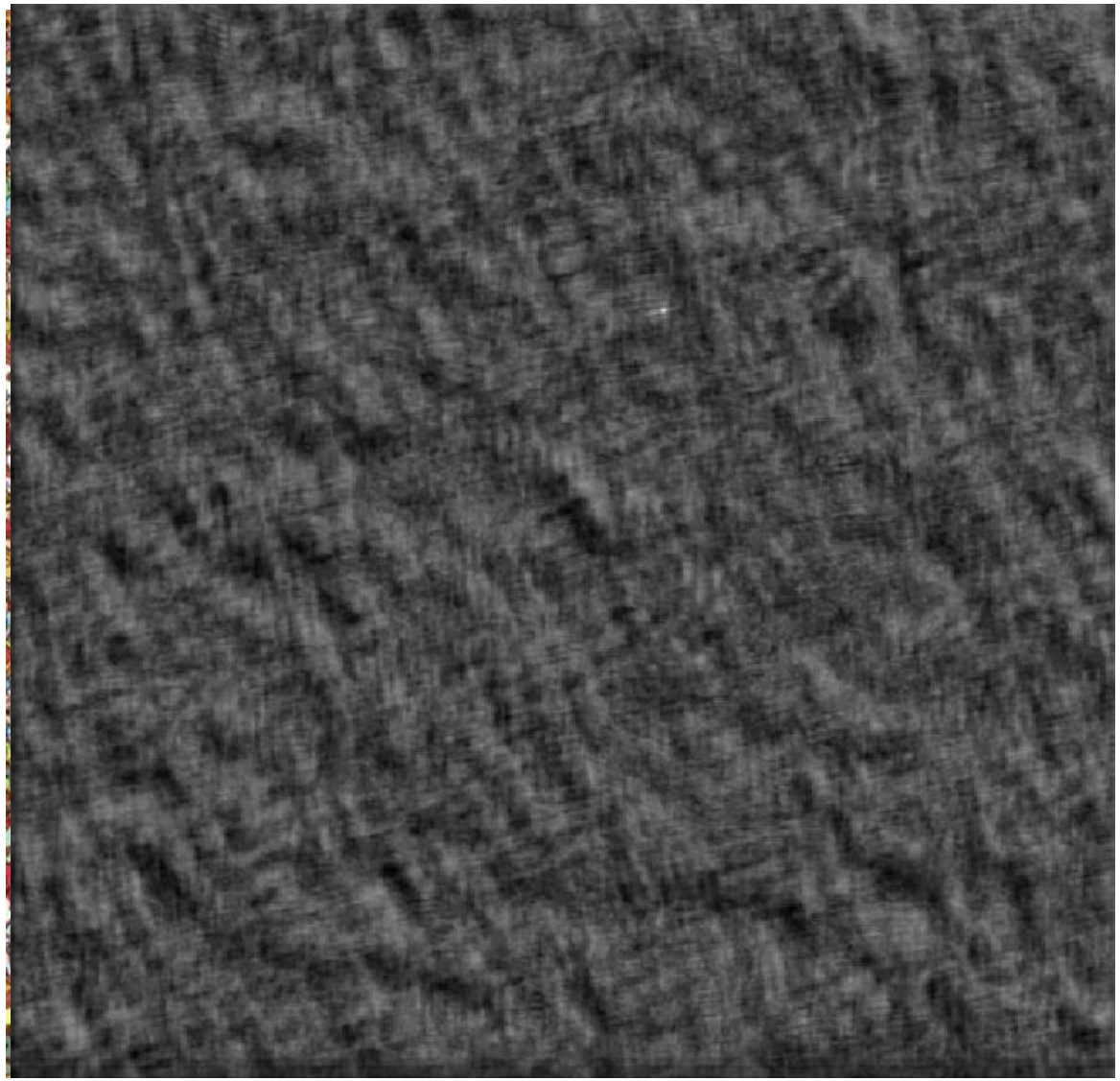
Where's Waldo?

Scene

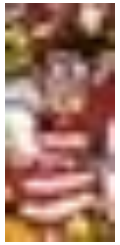


Template

Where’s Waldo?



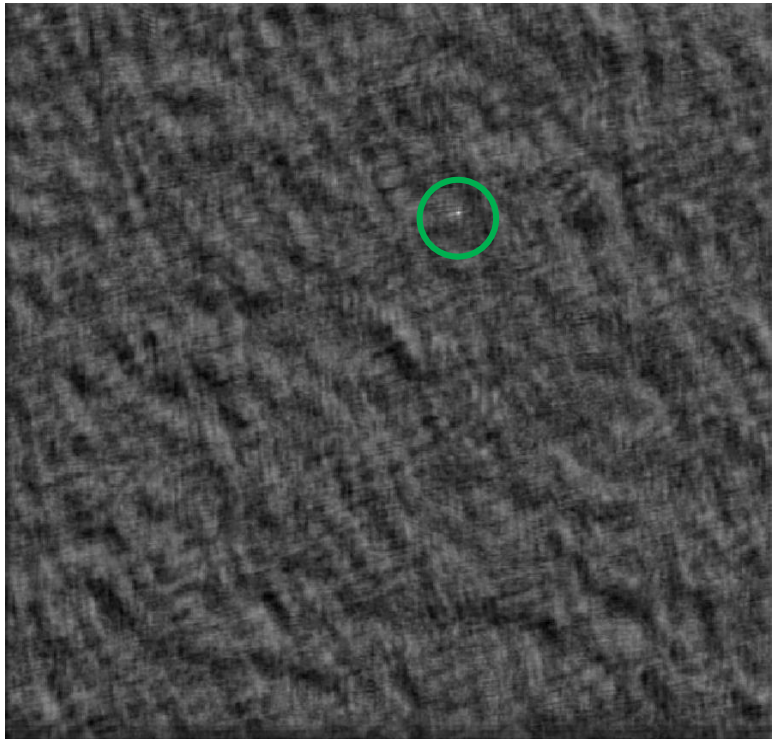
Scene



Template

Where’s Waldo?

Detected template



Correlation map

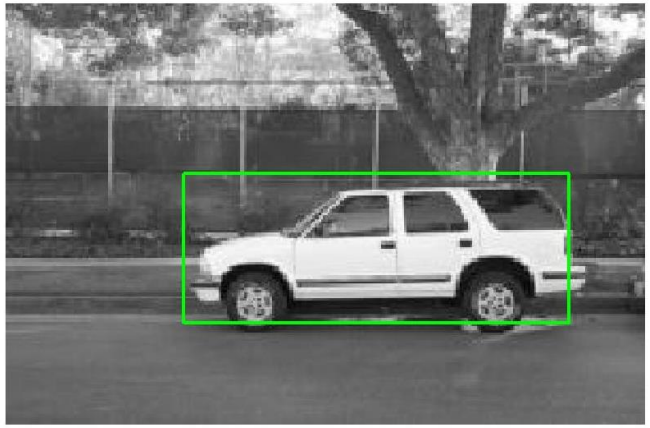
Template matching

Scene

Template

What if the template is not identical to some subimage in the scene?

Template matching



Detected template

Template

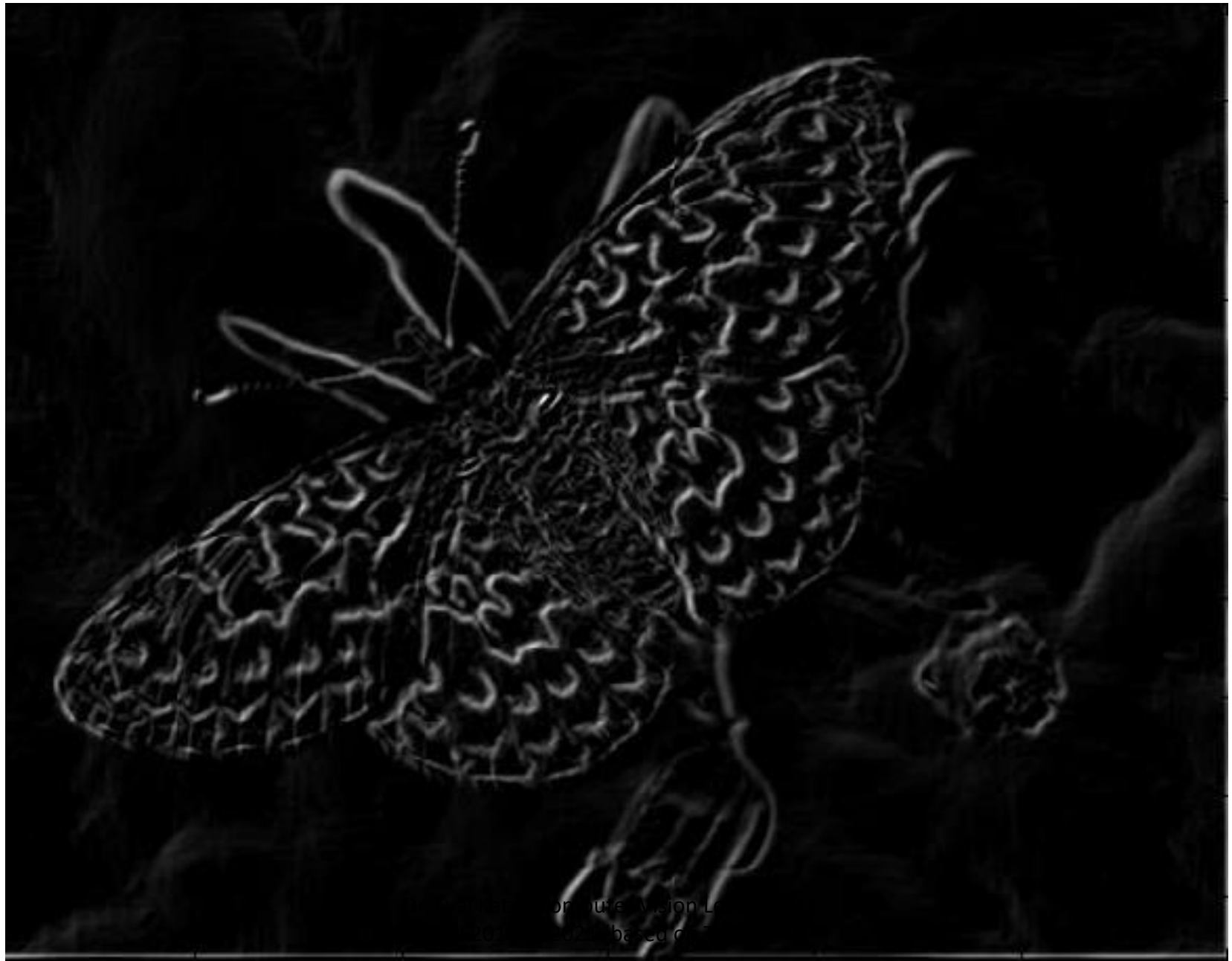
Match can be meaningful, if scale, orientation, and general appearance is right.

Thresholding

- Choose a threshold value t
- Set any pixels less than t to zero (off)
- Set any pixels greater than or equal to t to one (on)

Original image

Gradient magnitude image



Thresholding gradient with a lower threshold

Thresholding gradient with a higher threshold

Canny edge detector

- Filter image with derivative of Gaussian
- Find magnitude and orientation of gradient
- **Non-maximum suppression**:
  - Thin multi-pixel wide “ridges” down to single pixel width
- Linking and thresholding (**hysteresis**):
  - Define two thresholds: low and high
  - Use the high threshold to start edge curves and the low threshold to continue them

- MATLAB: `edge(image, 'canny');`
- `>>help edge`

Source: D. Lowe, L. Fei-Fei

The Canny edge detector

original image (Lena)

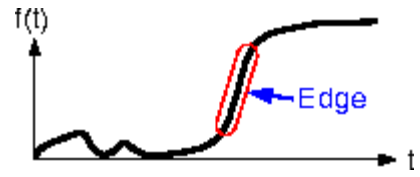
The Canny edge detector

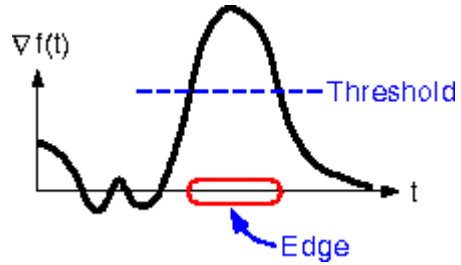
norm of the gradient

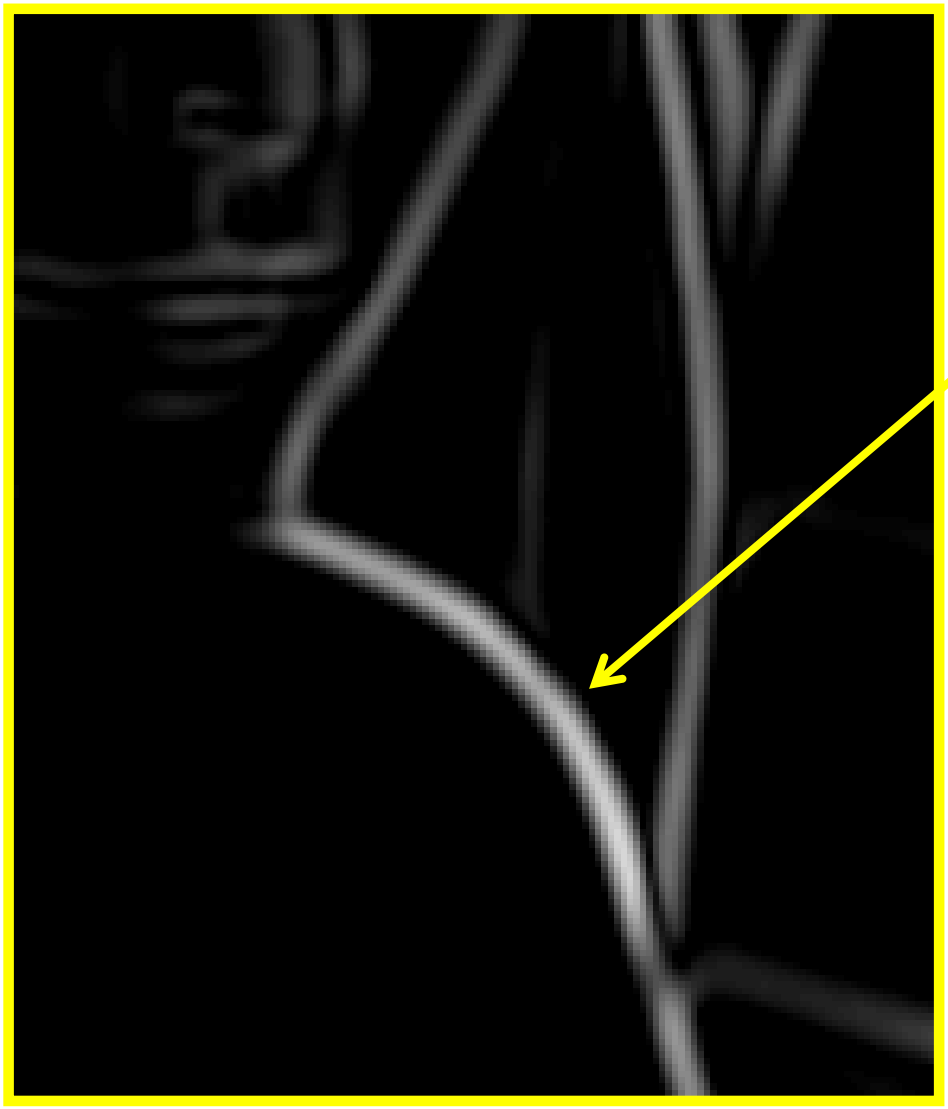
The Canny edge detector

thresholding

The Canny edge detector

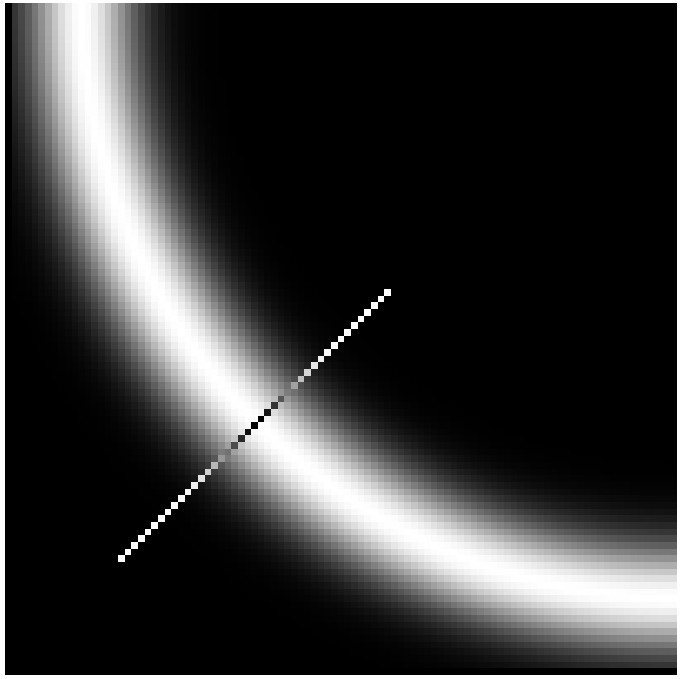


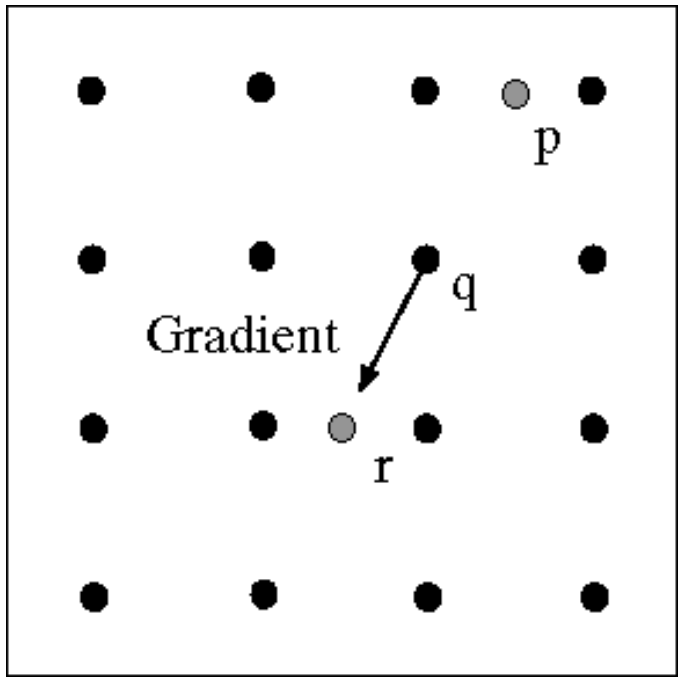




How to turn these thick regions of the gradient into curves?

Non-maximum suppression





Check if pixel is local maximum along gradient direction, select single max across width of the edge

- requires checking interpolated pixels p and r

The Canny edge detector

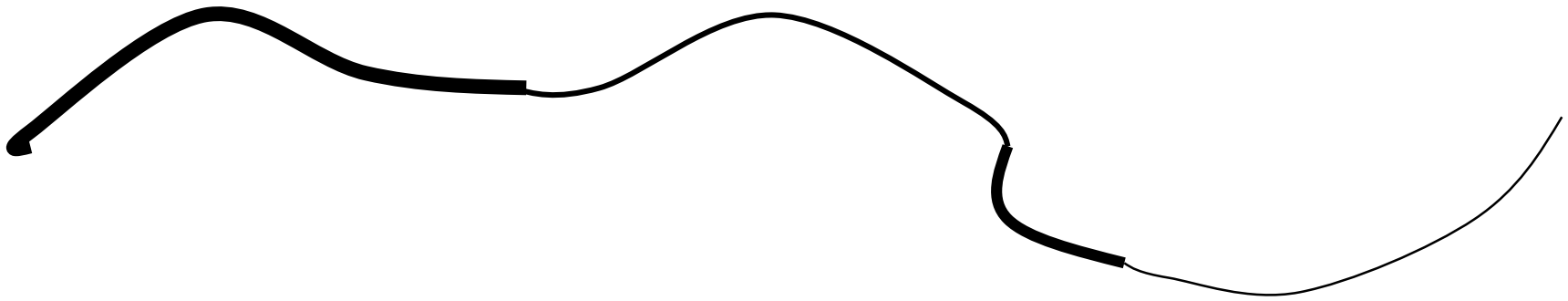
Problem: pixels along this edge didn't survive the thresholding

thinning

(non-maximum suppression)

Hysteresis thresholding

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Source: S. Seitz

Hysteresis thresholding

original image

high threshold
(strong edges)

low threshold
(weak edges)

hysteresis threshold

Source: L. Fei-Fei

Hysteresis thresholding

hysteresis threshold

Source: L. Fei-Fei

Dr. Eyal Katz - Computer Vision Lecture 2+3 - F2013-SP2021 (based on Trevor)

Object boundaries vs. edges

Background

Texture

Shadows

Edge detection is just the beginning…

image | human segmentation | gradient magnitude

Berkeley segmentation database:

http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

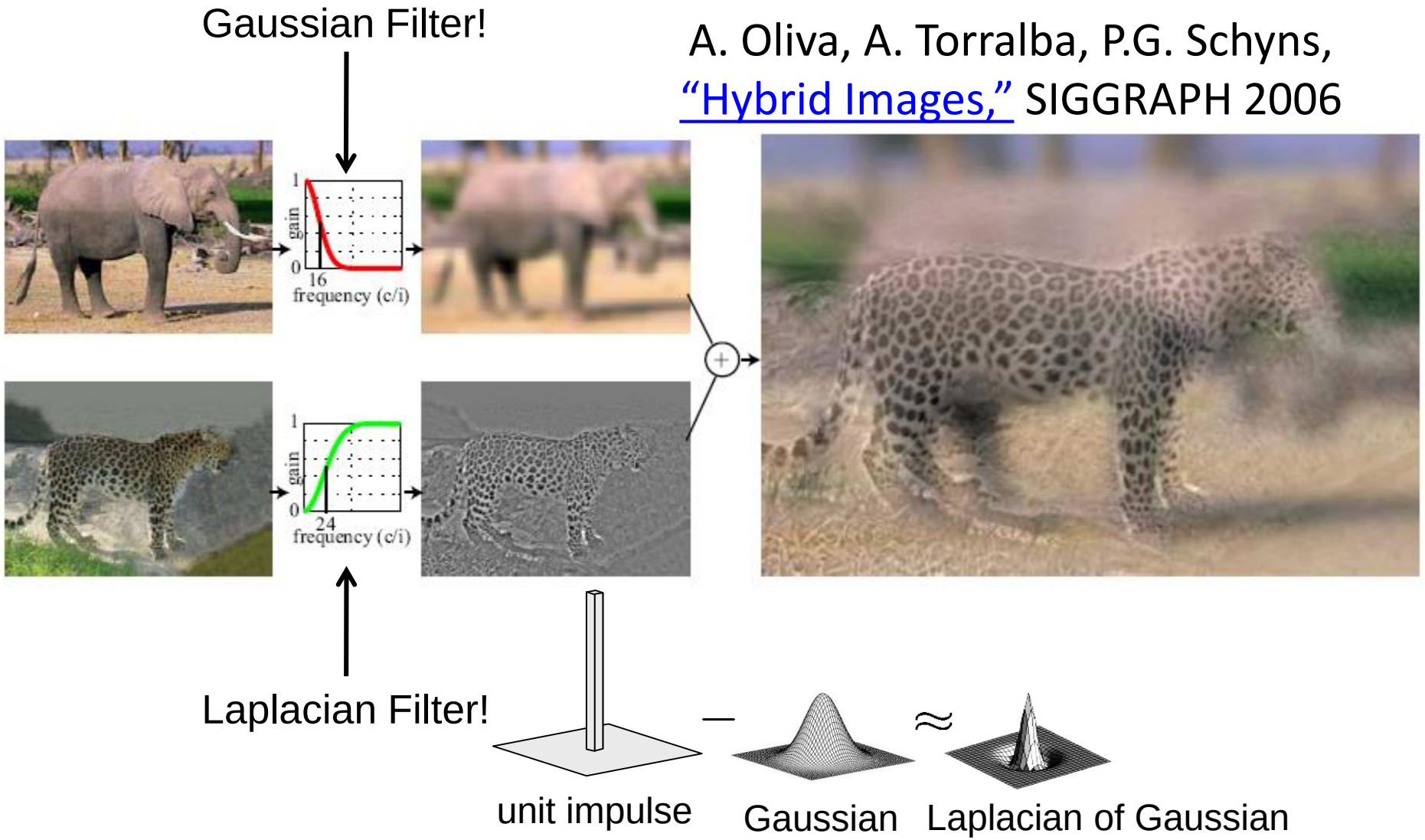
Much more on segmentation later in term…

Source: L. Lazebnik

Summary

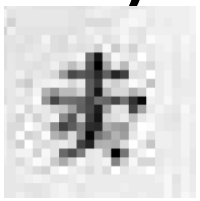
- Filters allow local image neighborhood to influence our description and features
  - Smoothing to reduce noise
  - Derivatives to locate contrast, gradient
- Filters have highest response on neighborhoods that “look like” it; can be thought of as template matching.
- Convolution properties will influence the efficiency with which we can process images.
  - Associative
  - Filter separability
- Edge detection processes the image gradient to find curves, or chains of edgels.

Extra: Hybrid Images (ref to the title slide…)



Summary (in pictures ☺)

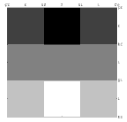
- Image is a matrix of numbers



=

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

- Filter is also a matrix of numbers



=

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

- Linear filtering calculates the convolution (rotate filter 180°; sum of dot product at each position)
  - e.g. Can smooth, sharpen, translate
- Some filters are non linear
  - e.g. median, morphological
- Be aware of details for filter size, border issues, extrapolation, cropping

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Practice questions

1. Write down a 3x3 filter that returns a positive value if the average value of the 4-adjacent neighbors is less than the center and a negative value otherwise

2. Write down a filter that will compute the gradient in the x-direction:

```
gradx(y,x) = im(y,x+1)-im(y,x) for each x, y
```

Practice questions (cont.)

3. Fill in the blanks:

Filtering Operator

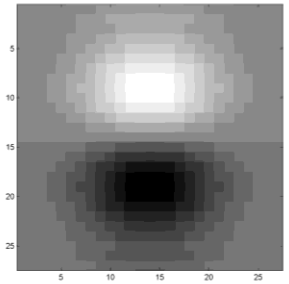
a) _ = D * B

b) A = _ * _

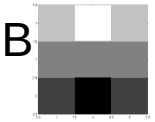
c) F = D * _

d) _ = D * D

A

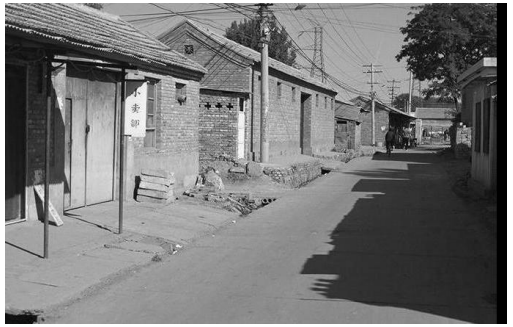


B

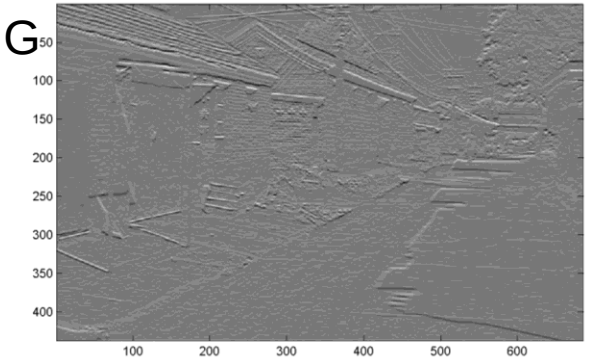


E

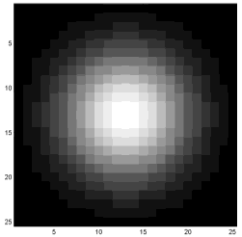
F



G



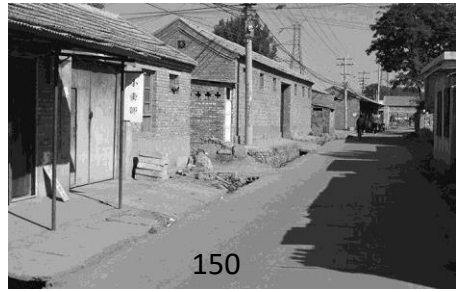
C



H

I

D



Slide Credits

- Prof. Trevor Darrell
- Kristen Grauman
- James Hays
- and Seitz, Marschner, Lazebnik and others, as noted…

Next time

- Sampling
- 2D Fourier Transform
- Pyramids…