Linear Filters

CS 554 – Computer Vision

Pinar Duygulu

Bilkent University

Zebras vs. Dalmatians

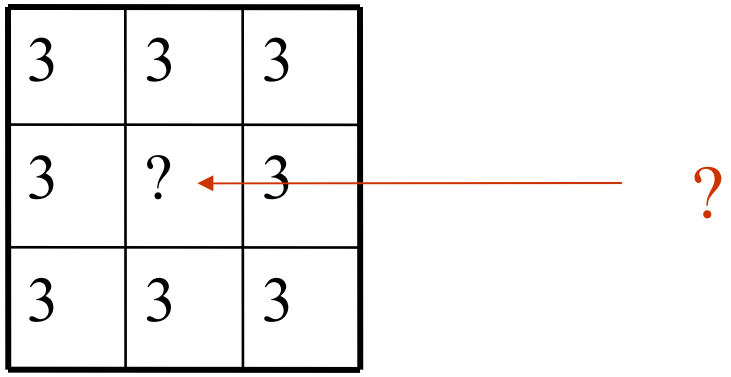
Both zebras and dalmatians have black and white pixels in about the same number

The differences between the two is with the characteristic appearance of small group of pixels rather than individual pixel values

Today's lecture

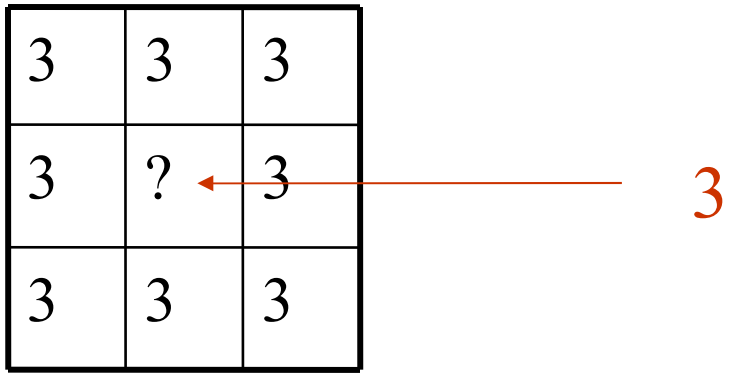
- Discuss methods for obtaining descriptions of the appearance of a small group of pixels
- Strategy : use the weighted sums of pixel values using different weights to find different image patterns
- Helpful for
  - Smooth noise in the image
  - Find edges and other patterns in the image

Linear Filters



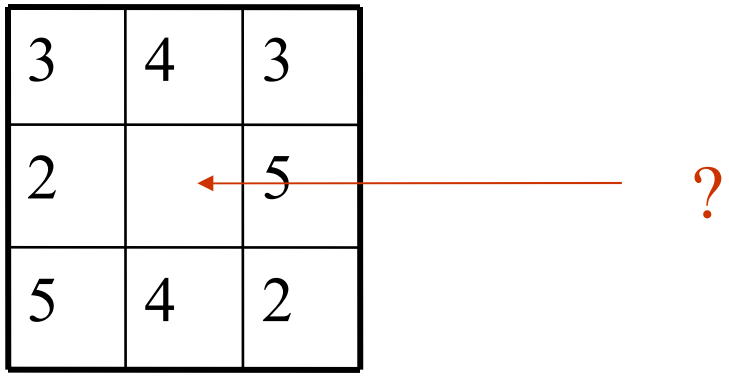
adapted from Michael Black, Brown University

Linear Filters



adapted from Michael Black, Brown University

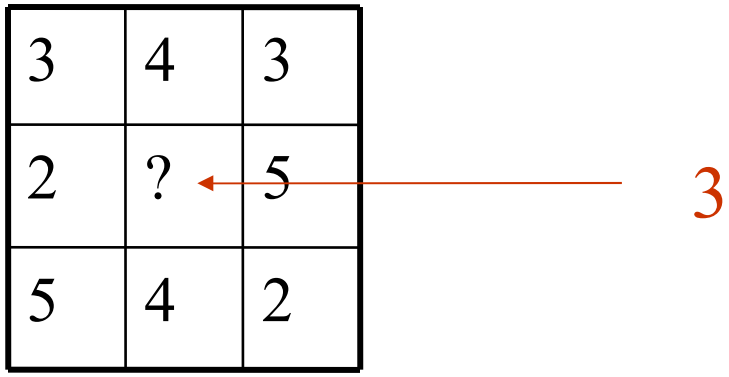
Linear Filters



What assumptions are you making to infer the center value?

adapted from Michael Black, Brown University

Linear Filters

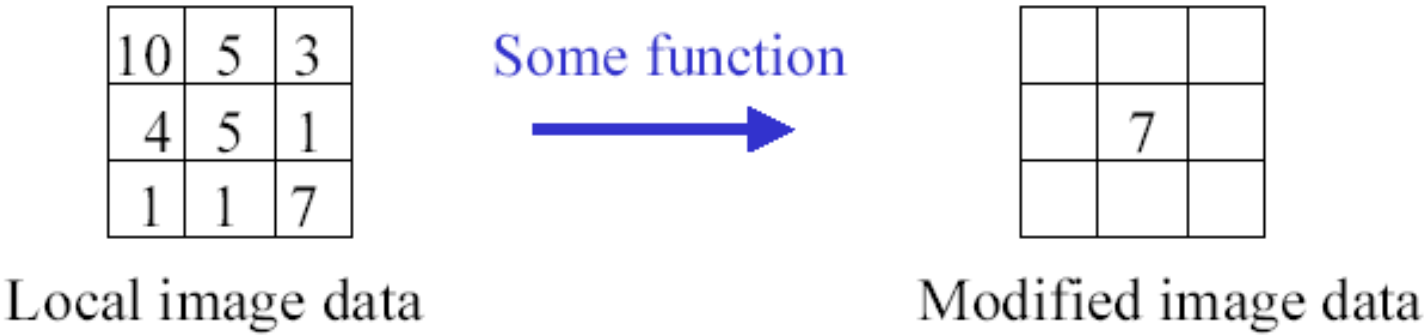


Replace each pixel by a linear combination of its neighbours

adapted from Michael Black, Brown University

Linear Filters

Modify the pixels in an image based on some function of the pixels.



adapted from Darrell and Freeman, MIT

Linear Filters

Simplest: linear filtering.
– Replace each pixel by a linear combination of its neighbours.

The prescription for the linear combination is called the "convolution kernel".

10	5	3
4	5	1
1	1	7

Local image data

0	0	0
0	0.5	0
0	1	0.5

kernel

	7	

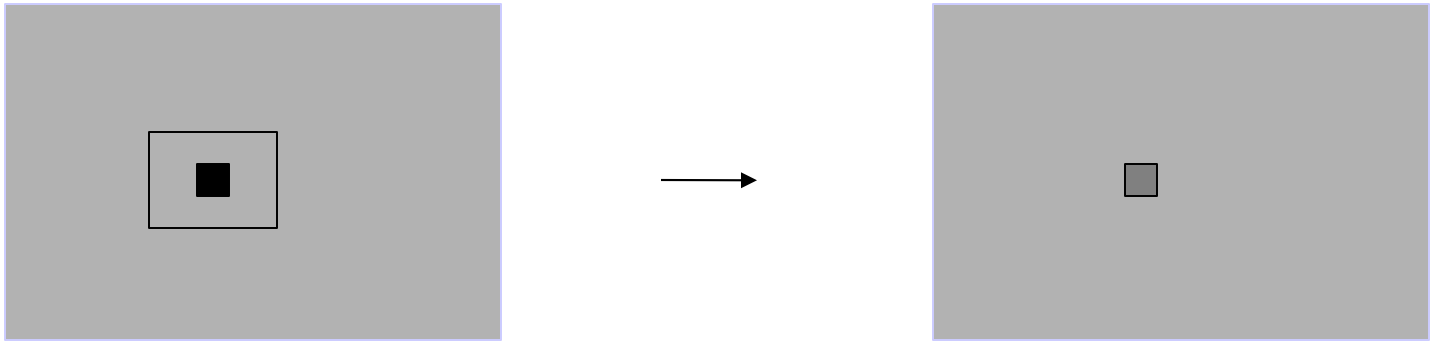
Modified image data

adapted from Darrell and Freeman, MIT

Linear Filters

Idea :

- Construct a new array – same size as the image
- Fill each location of this new array with a weighted sum of the pixel values from the locations surrounding the corresponding location in the image using the same set of weights each time



Image

Result

Linear Filters

Example:

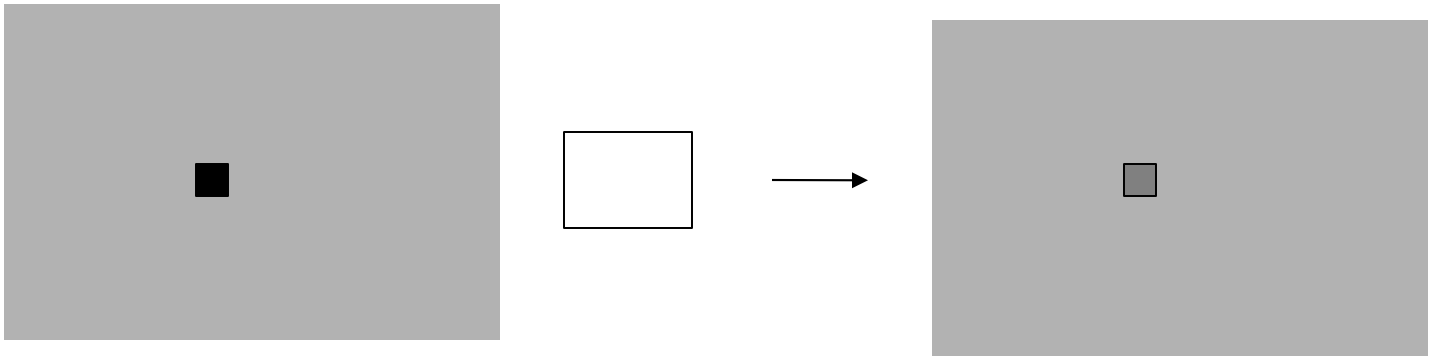
Computing a local average taken over a fixed region

Average all pixels within a 2k+1 x 2k+1 block of the pixel of interest

For an input image F, this gives an output

$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv}$$

Convolution

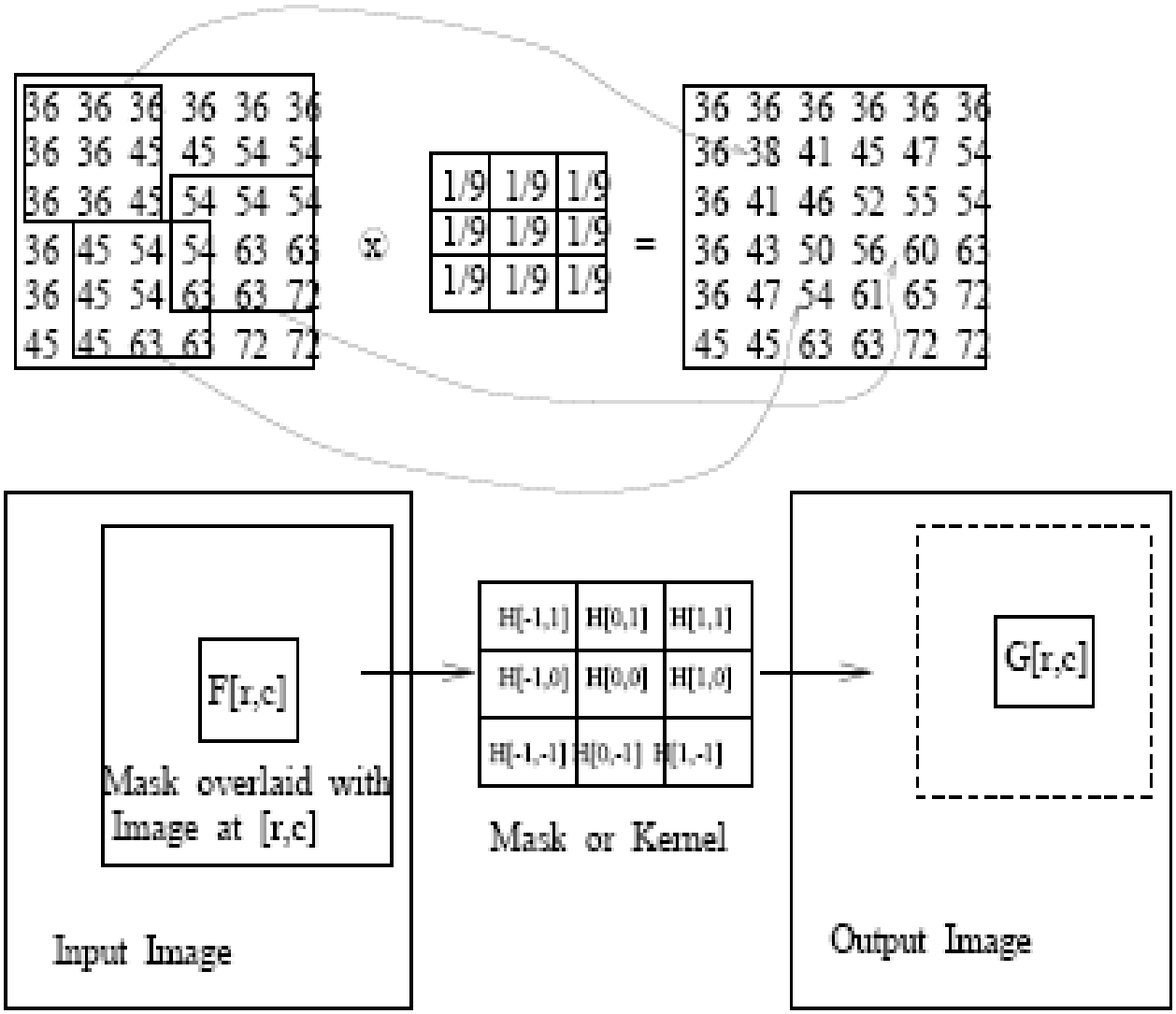


Image

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

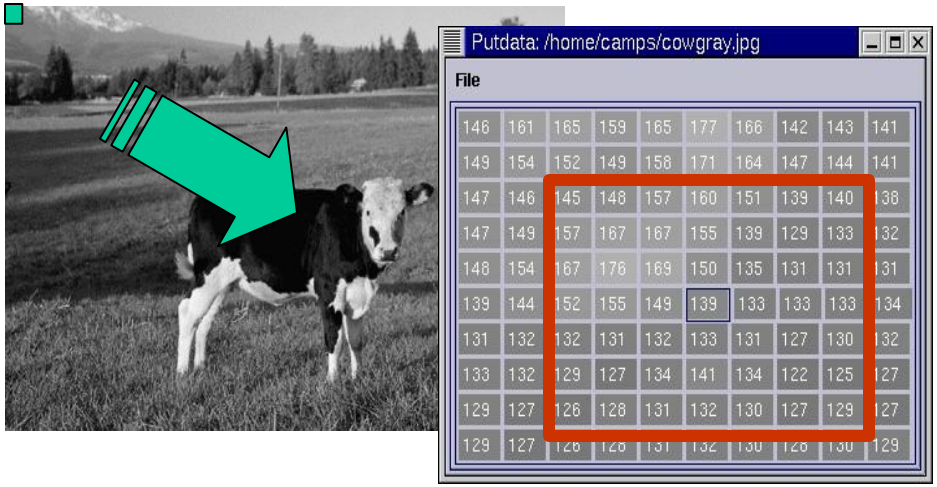
Result

Convolution



adapted from Shapiro&Stockman

Example



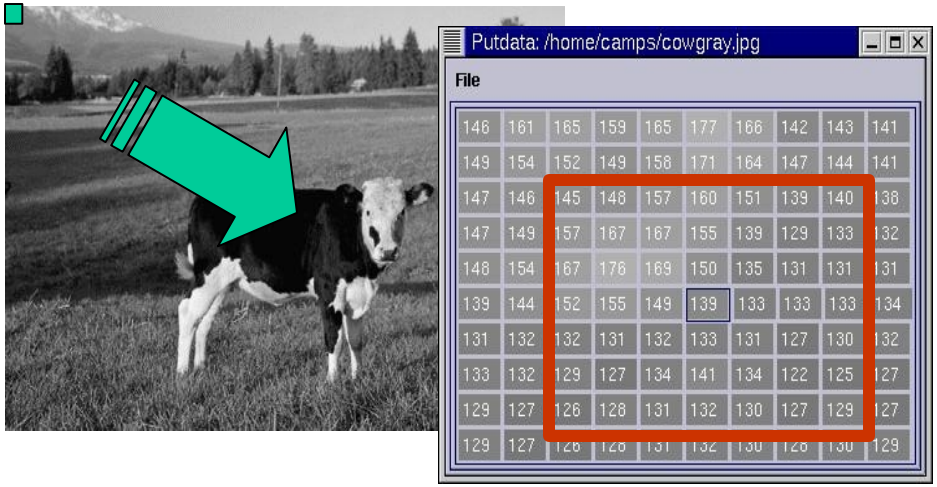
145	148	157	160	151	139	140
157	167	167	155	139	129	133
167	176	169	150	135	131	131
152	155	149	139	133	133	133
132	131	132	133	131	127	130
129	127	134	141	134	122	125
126	128	131	132	130	127	129

→

			141			

$$(169 + 150 + 135 + 149 + 139 + 133 + 132 + 133 + 131) / 9 = 141.2 = 141$$

Example



145	148	157	160	151	139	140
157	167	167	155	139	129	133
167	176	169	150	135	131	131
152	155	149	139	133	133	133
132	131	132	133	131	127	130
129	127	134	141	134	122	125
126	128	131	132	130	127	129

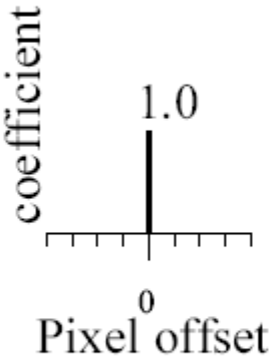
→

			141	137		

(150+135+131+139+133+133+133+131+127) / 9 = 134 + 6 = 137

Linear Filtering

original

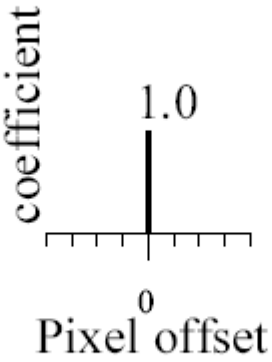


?

adapted from Darrell and Freeman, MIT

Linear Filtering

original

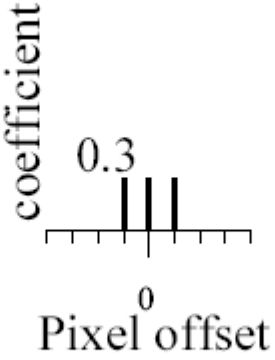


Filtered
(no change)

adapted from Darrell and Freeman, MIT

Linear Filtering

original

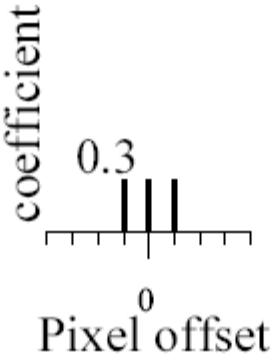


?

adapted from Darrell and Freeman, MIT

Linear Filtering - Blurring

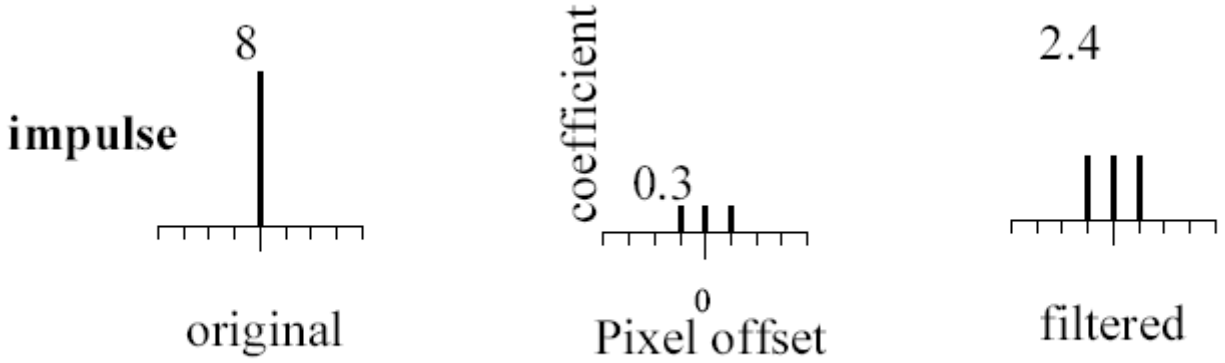
original



Blurred (filter applied in both dimensions).

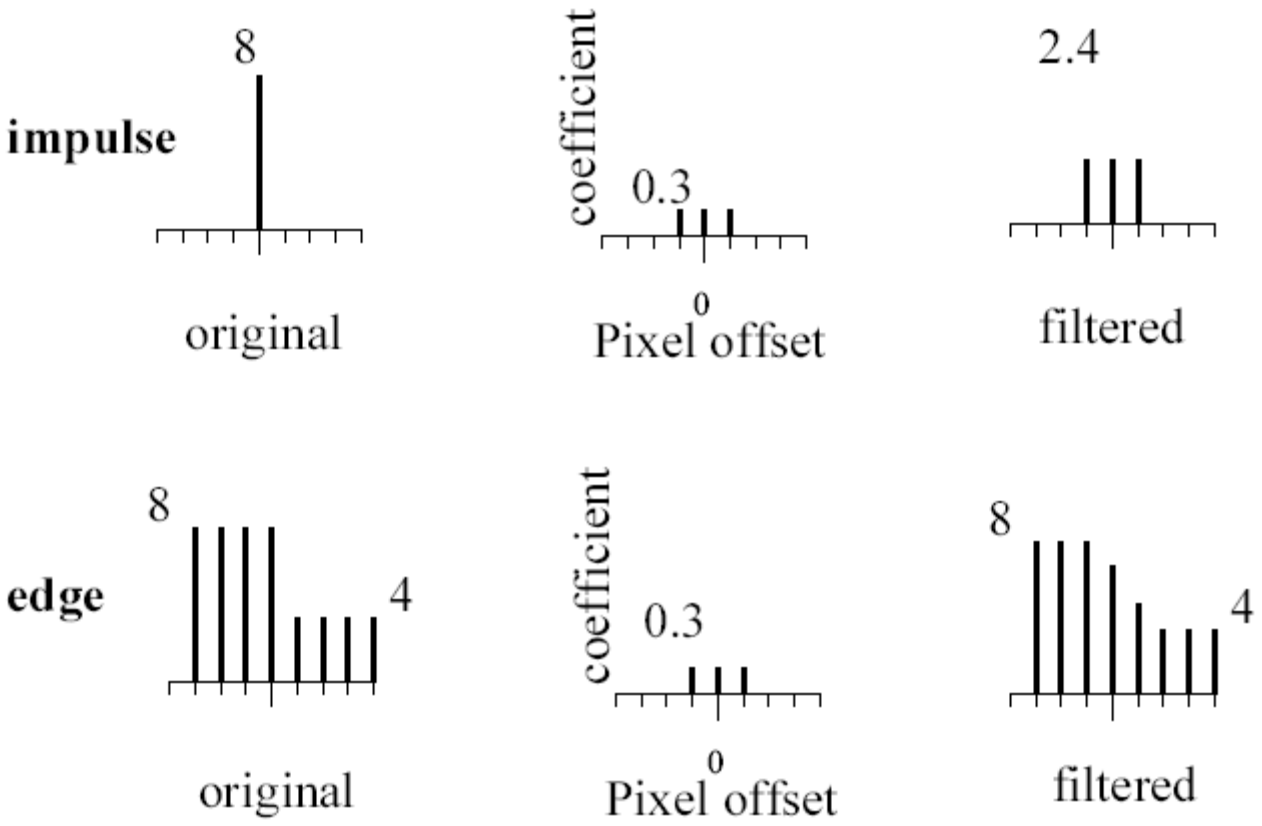
adapted from Darrell and Freeman, MIT

Linear Filtering - Blurring



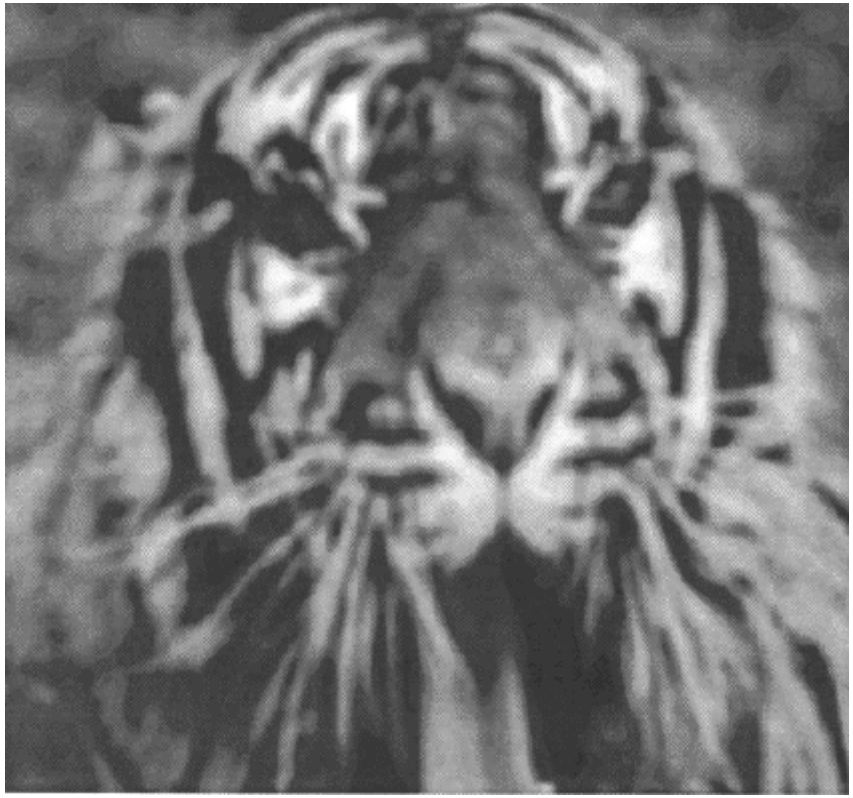
adapted from Darrell and Freeman, MIT

Linear Filtering - Blurring



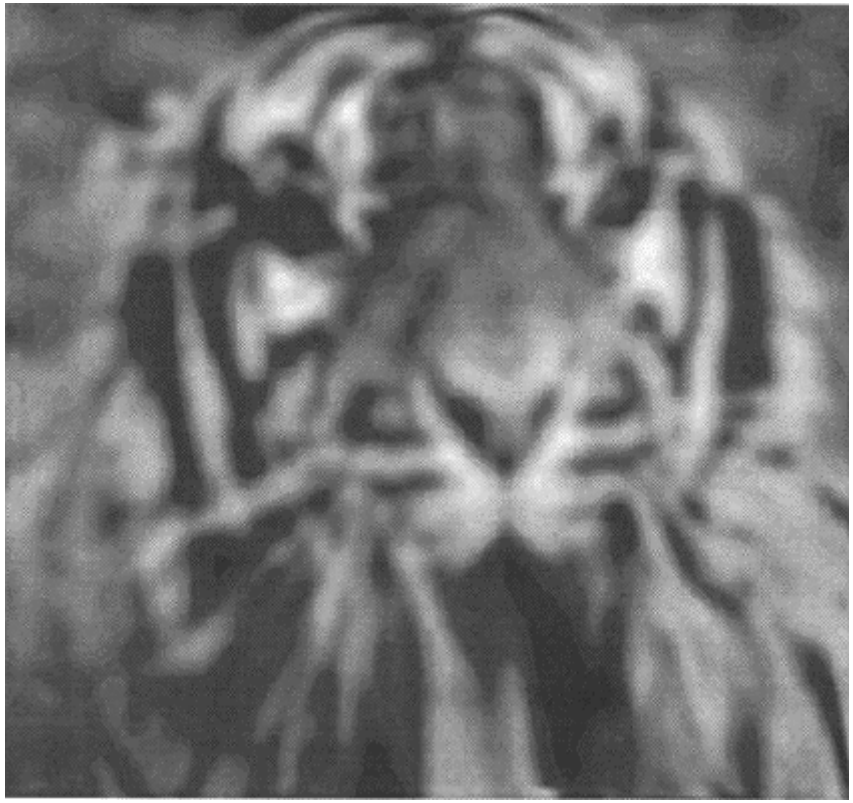
adapted from Darrell and Freeman, MIT

Example



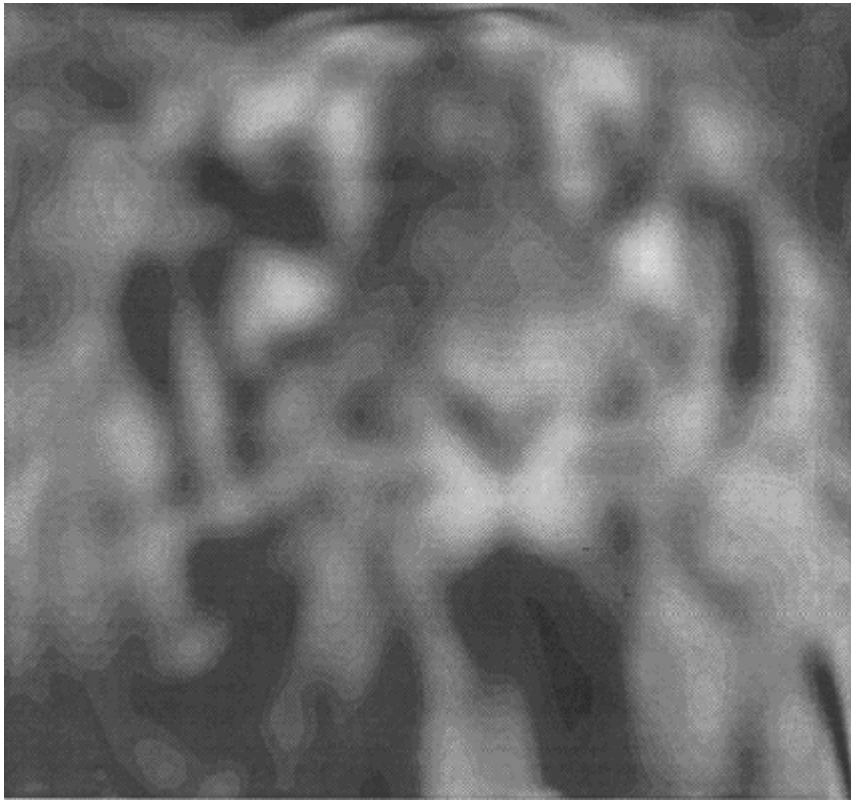
adapted from Martial Hebert, CMU

Example



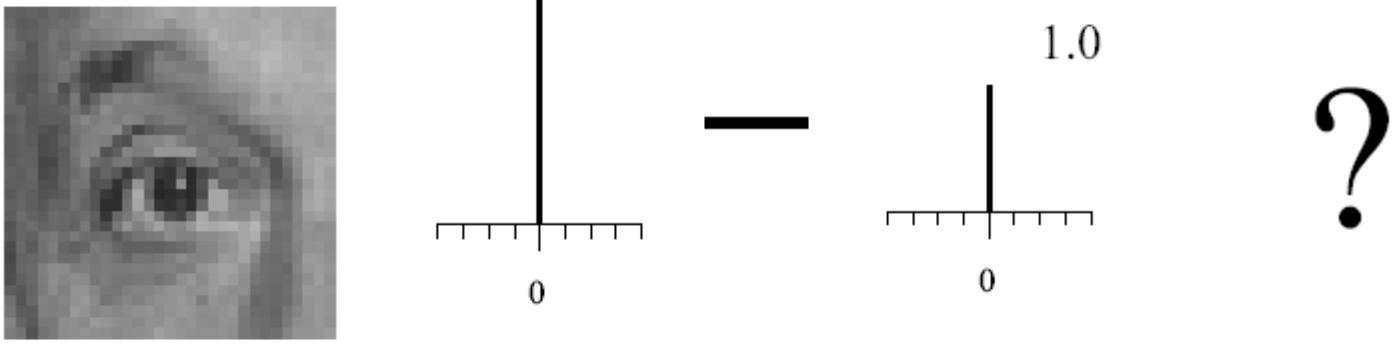
adapted from Martial Hebert, CMU

Example



adapted from Martial Hebert, CMU

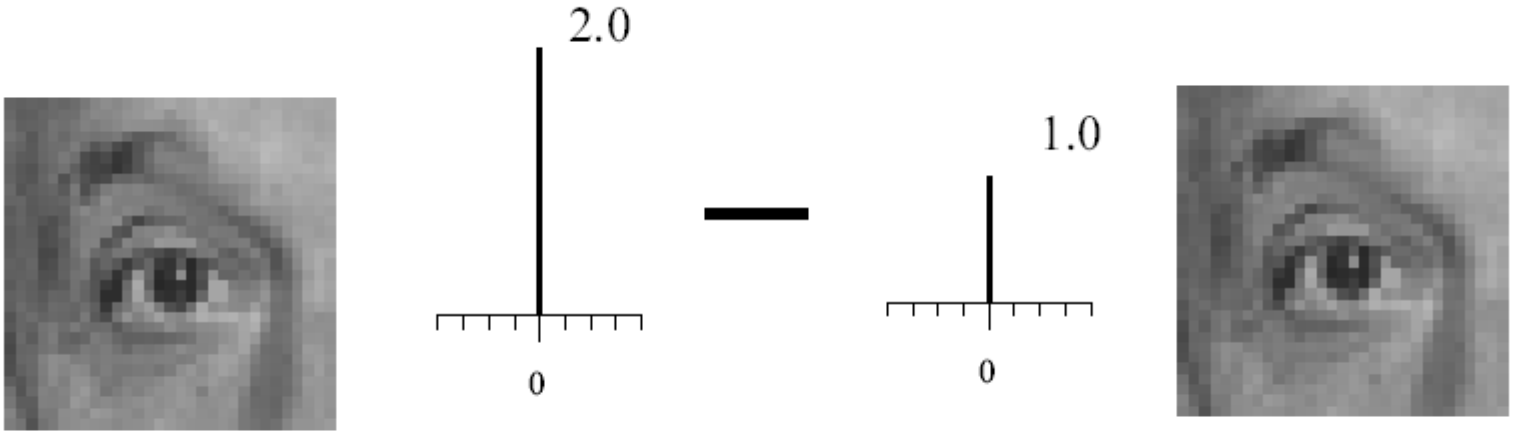
Linear Filtering



original

adapted from Darrell and Freeman, MIT

Linear Filtering

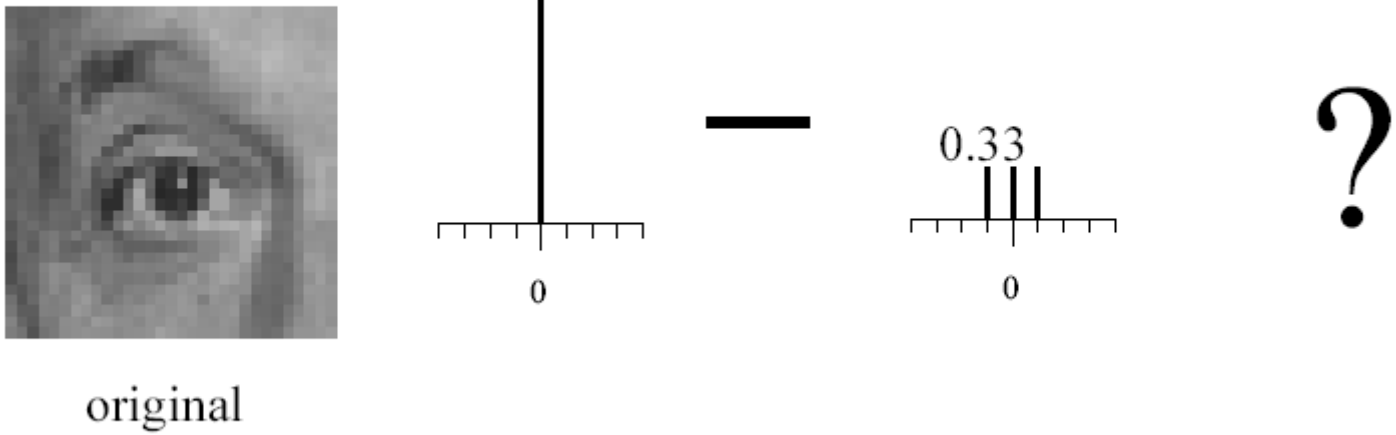


original

Filtered
(no change)

adapted from Darrell and Freeman, MIT

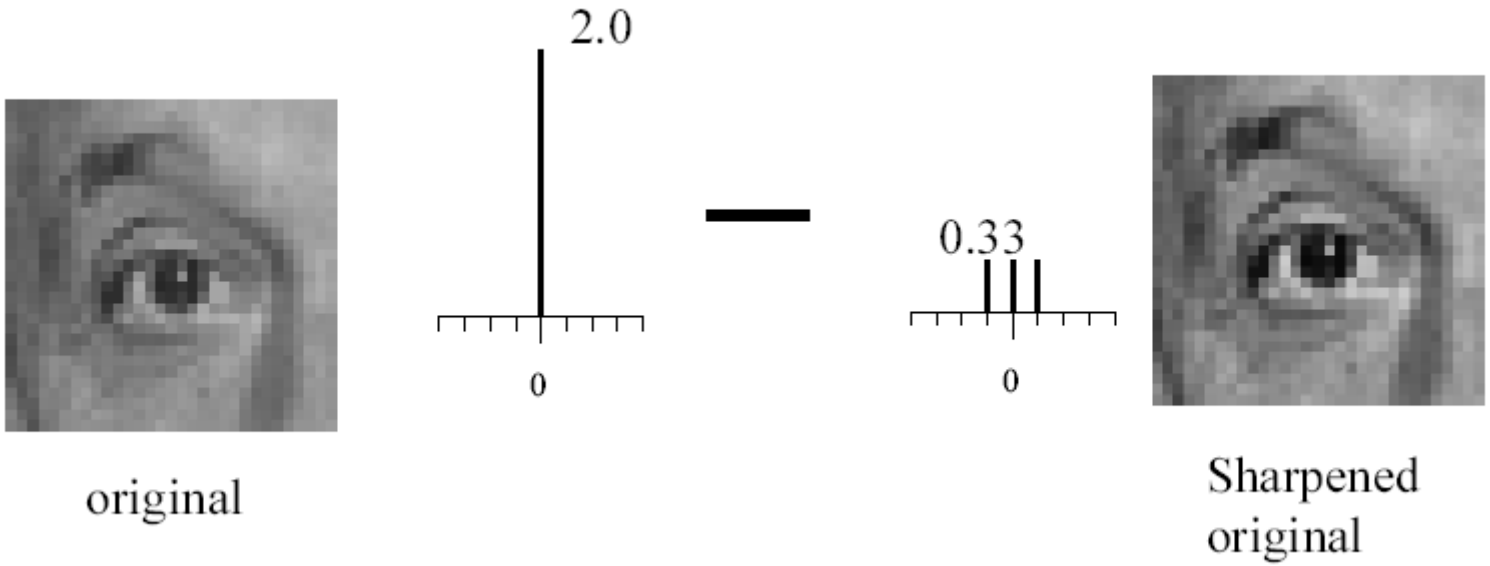
Linear Filtering



original

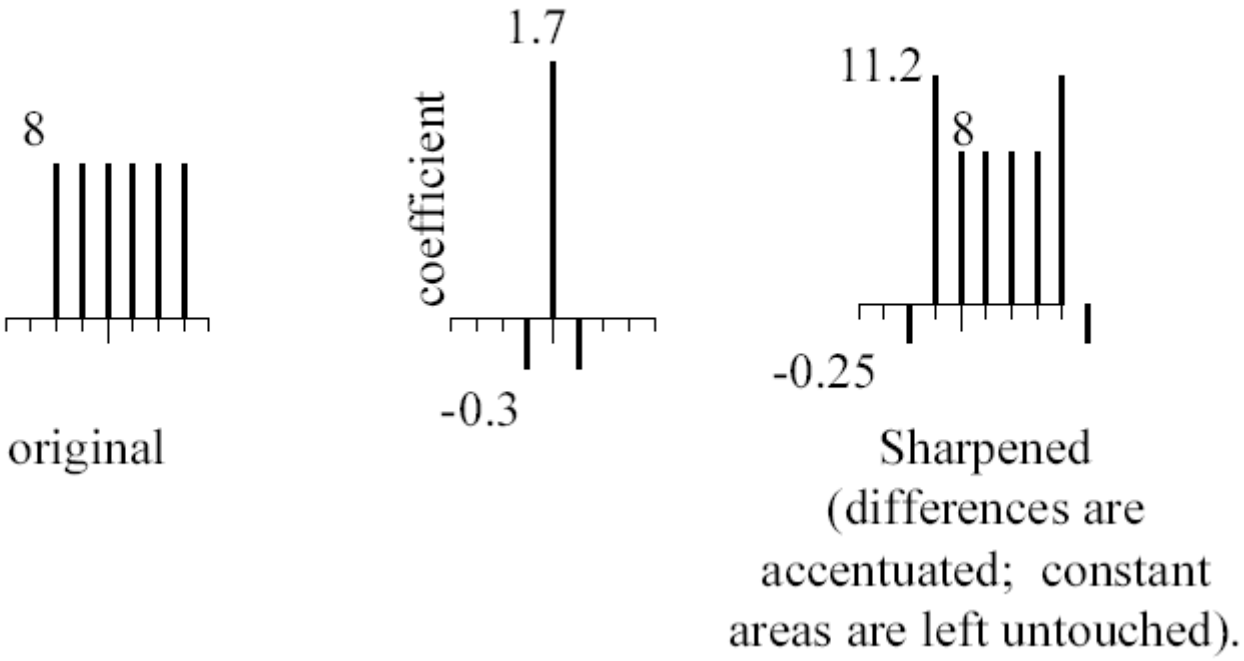
adapted from Darrell and Freeman, MIT

Linear Filtering - Sharpening



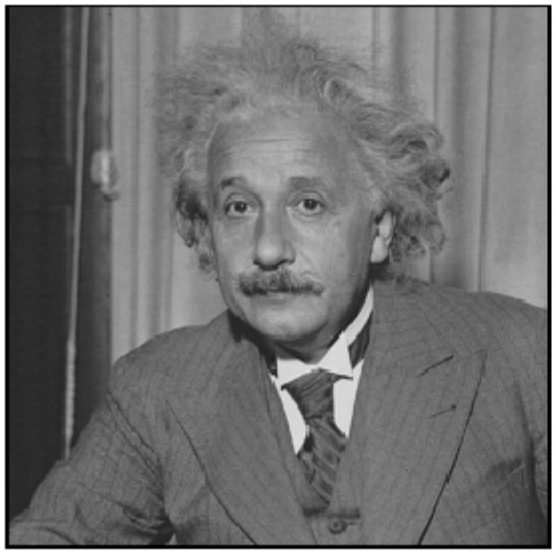
adapted from Darrell and Freeman, MIT

Linear Filtering - Sharpening

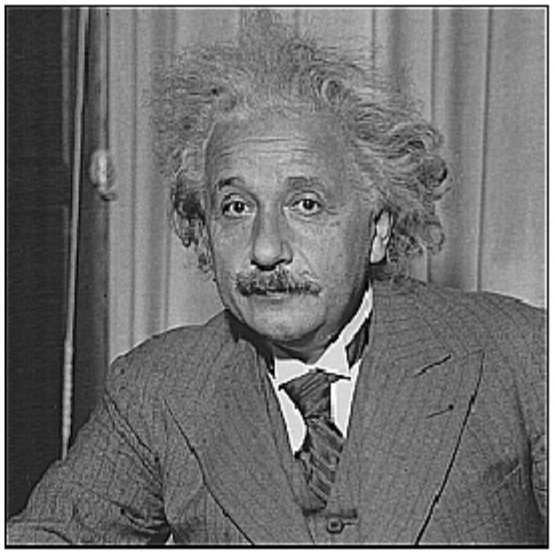


adapted from Darrell and Freeman, MIT

Linear Filtering - Sharpening



before



after

adapted from Darrell and Freeman, MIT

Linear Filters

Whatever the weights, the output is

- *Shift invariant* :
  - The value of the output depends on the pattern in an image neighborhood
  - Not to the position of the neighborhood
- *Linear* :
  - the output for the sum of two images is the same as the the sum of the outputs obtained from the images separately.

This procedure is called *linear filtering*

Convolution

The pattern of weights used for linear filtering

- *Kernel* of the filter

The process of applying the filter

- *convolution*

$$R_{ij} = \sum_{u,v} H_{i-u,j-v} F_{uv}$$

F : image
H : filter kernel
R: the convolution of the kernel H with the image F

- Notice weird order of indices
  - –all examples can be put in this form
  - –it's a result of the derivation expressing any shift-invariant linear operator as a convolution.

Smoothing by averaging

- The value of a pixel is usually similar to that of its neighbor
- Assume that the image is affected by noise
  - There may be occasional dead pixels
  - Or small random numbers with zero mean
- Reduce the effects of this noise by replacing each pixel with a weighted average of its neighbors
  - Smoothing or blurring

C x

```
……..
….1 1 1….
….1 1 1….
….1 1 1….
…….
```

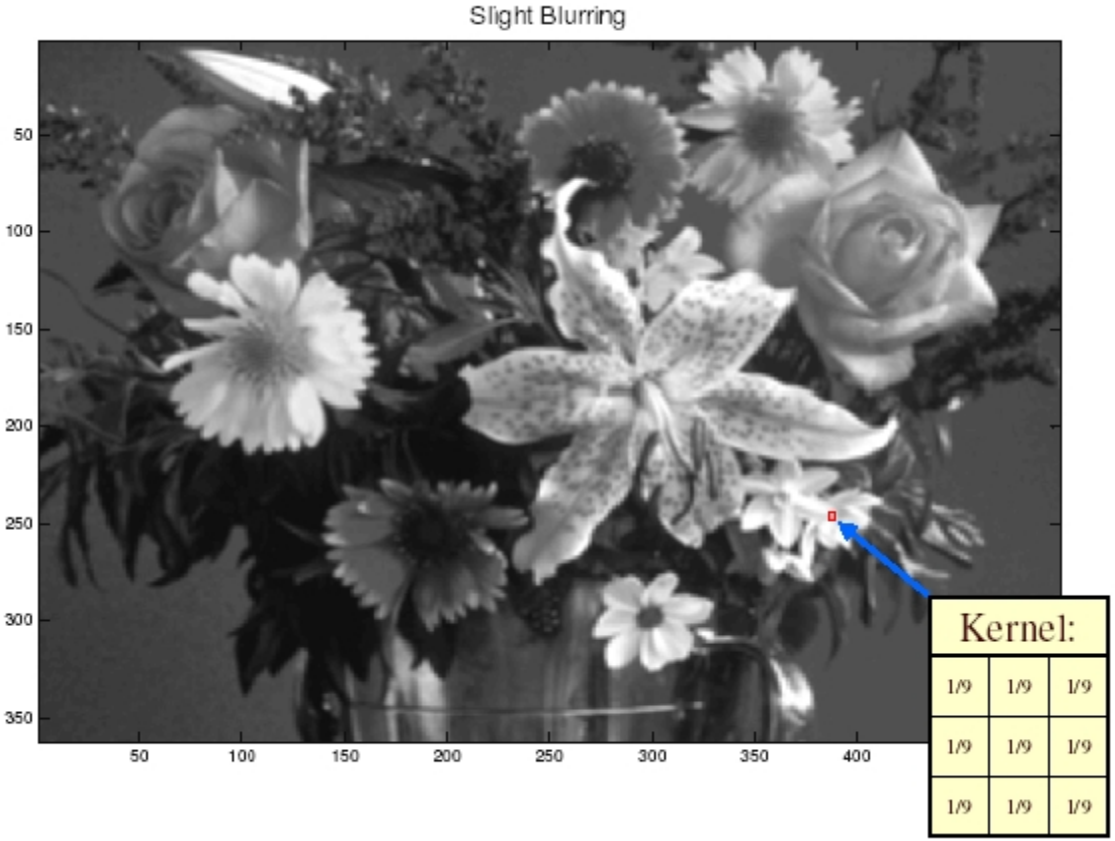
A block of ones multiplied by a constant

Replacing each pixel with an un-weighted average computed over some fixed region centered at the pixel

Smoothing by averaging

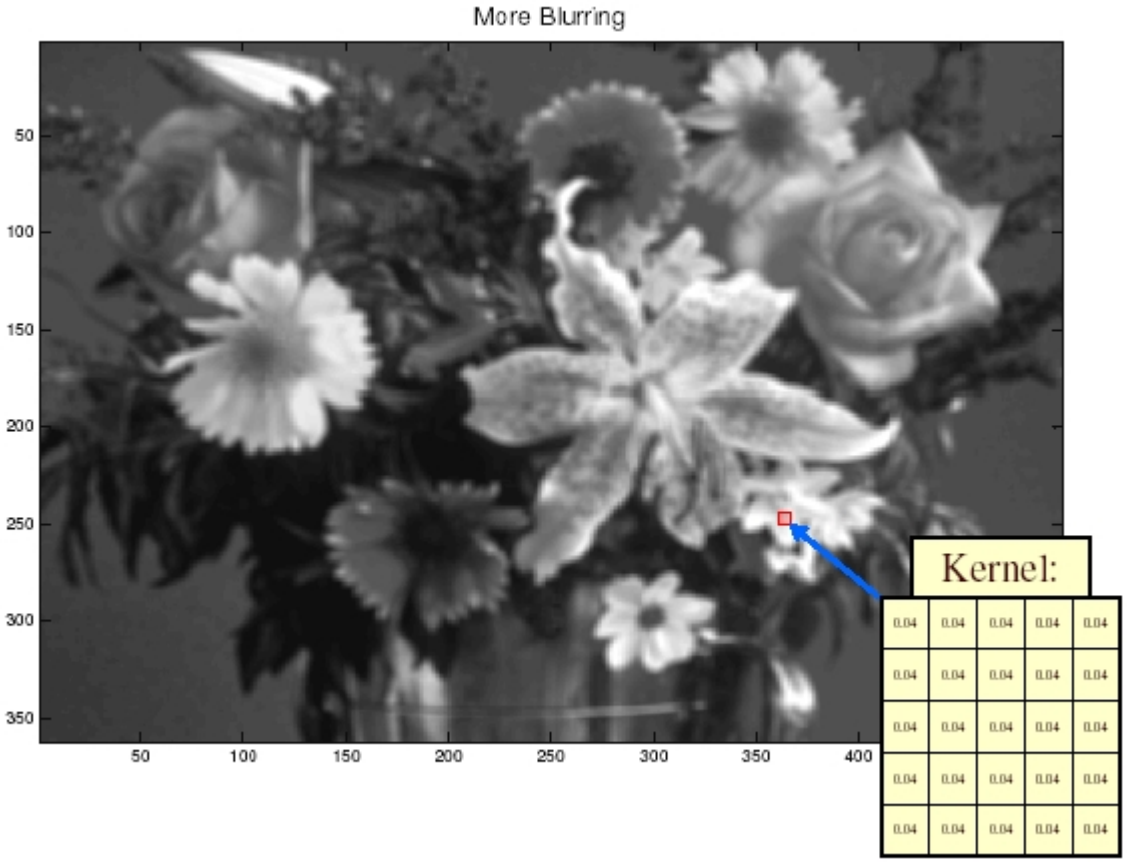
adapted from Martial Hebert, CMU

Smoothing by averaging



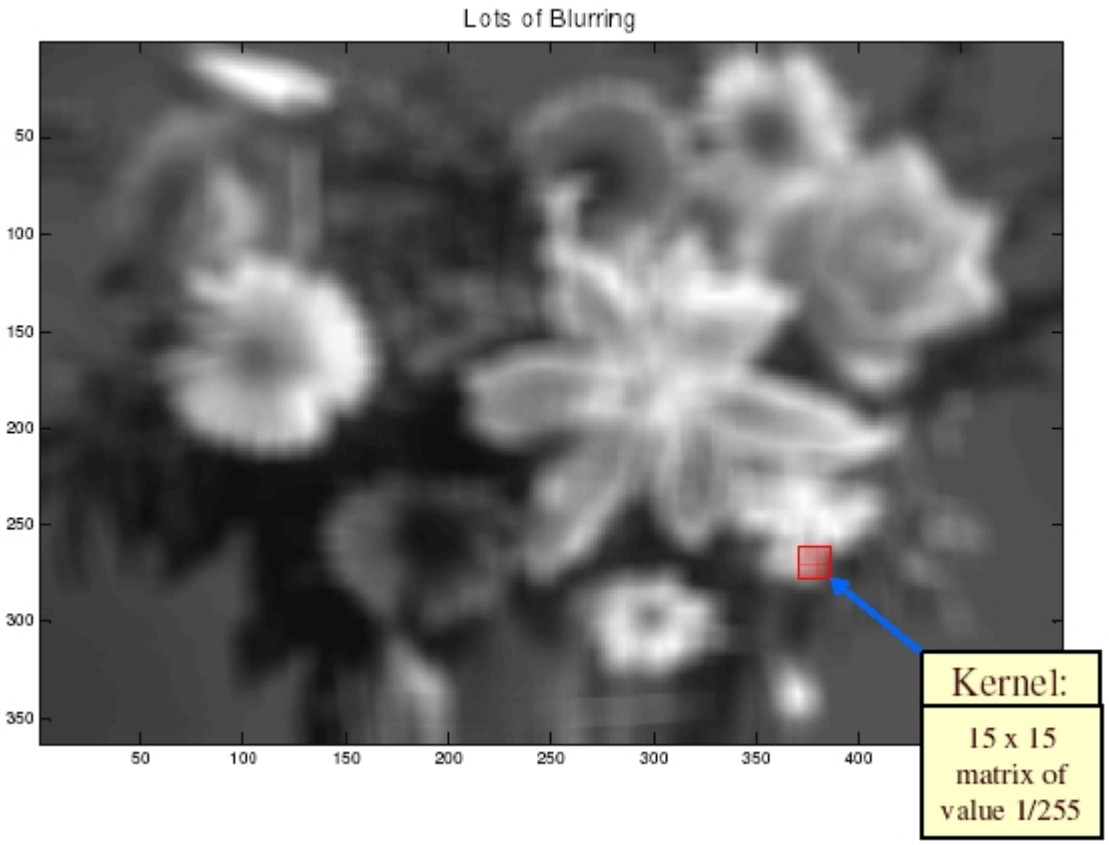
adapted from Martial Hebert, CMU

Smoothing by averaging



adapted from Martial Hebert, CMU

Smoothing by averaging



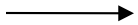
adapted from Martial Hebert, CMU

Smoothing by averaging

- It is a poor model for blurring
  - Not like a defocused camera

Noise : a bright dot on a dark background

0	0	0
0	1	0
0	0	0

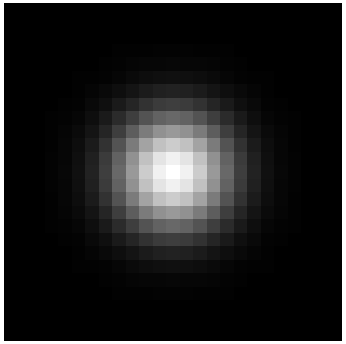


A small bright box

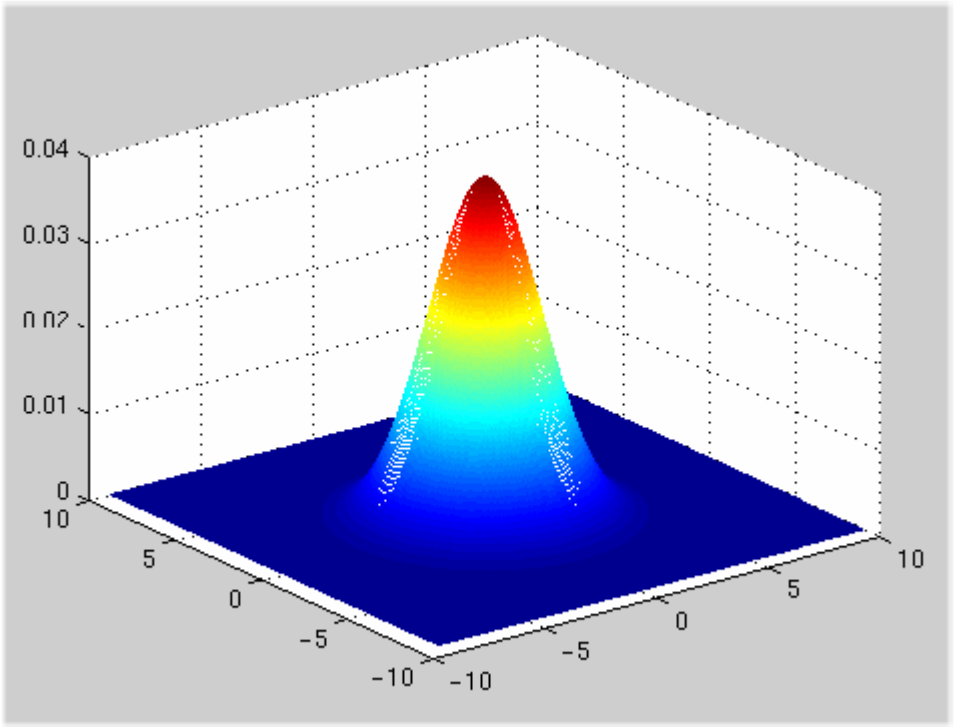
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

It should be

a small bright dot at the center, fading slowly to darkness



Smoothing with a Gaussian



2D Gaussian filter

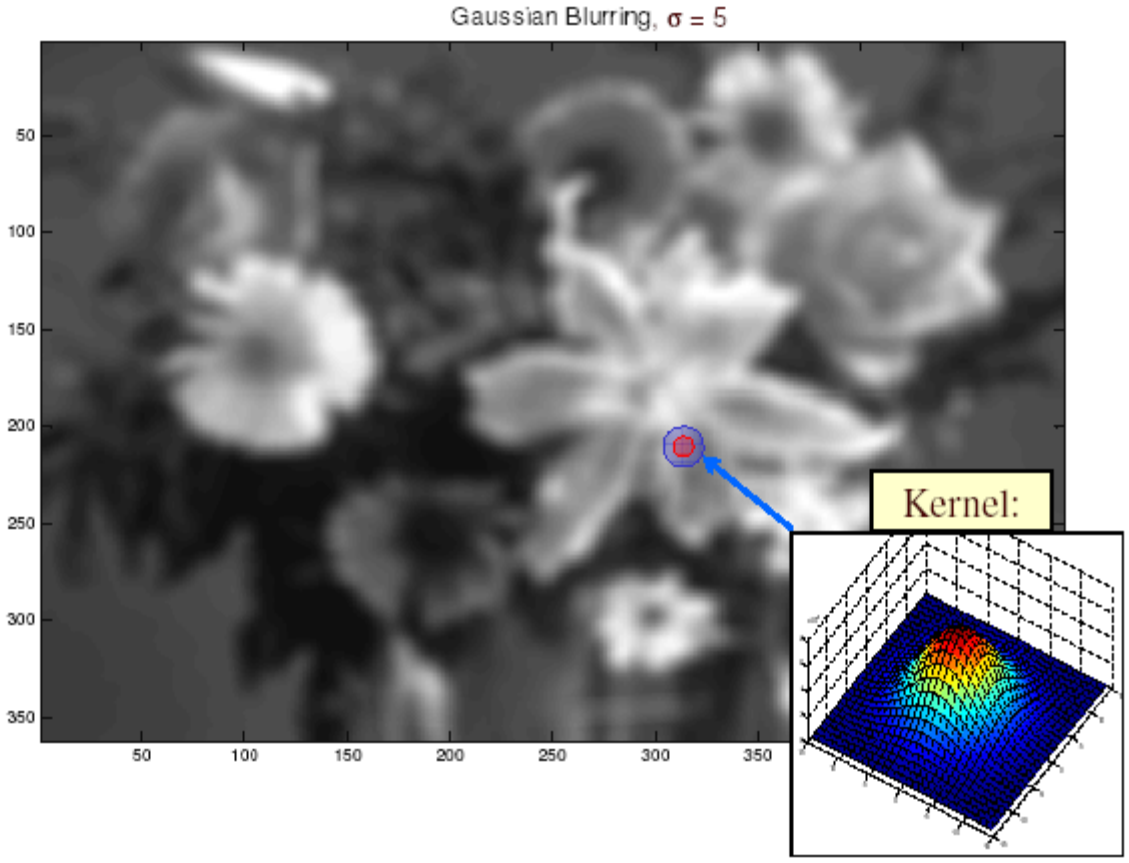
$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$

σ : standard deviation

A weighted average that weights pixels at its center much more strongly than its boundaries

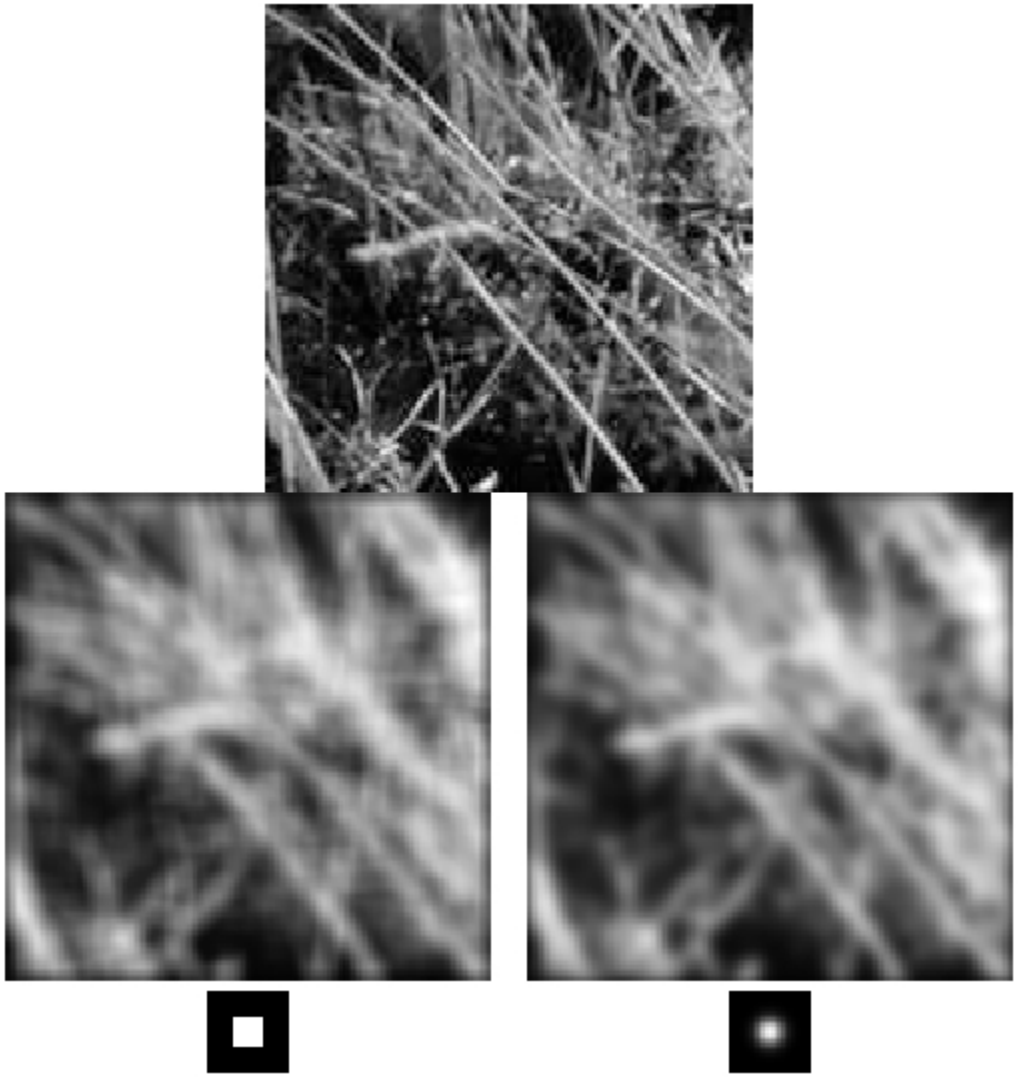
adapted from Martial Hebert, CMU

Smoothing with a Gaussian



adapted from Martial Hebert, CMU

Smoothing with a Gaussian



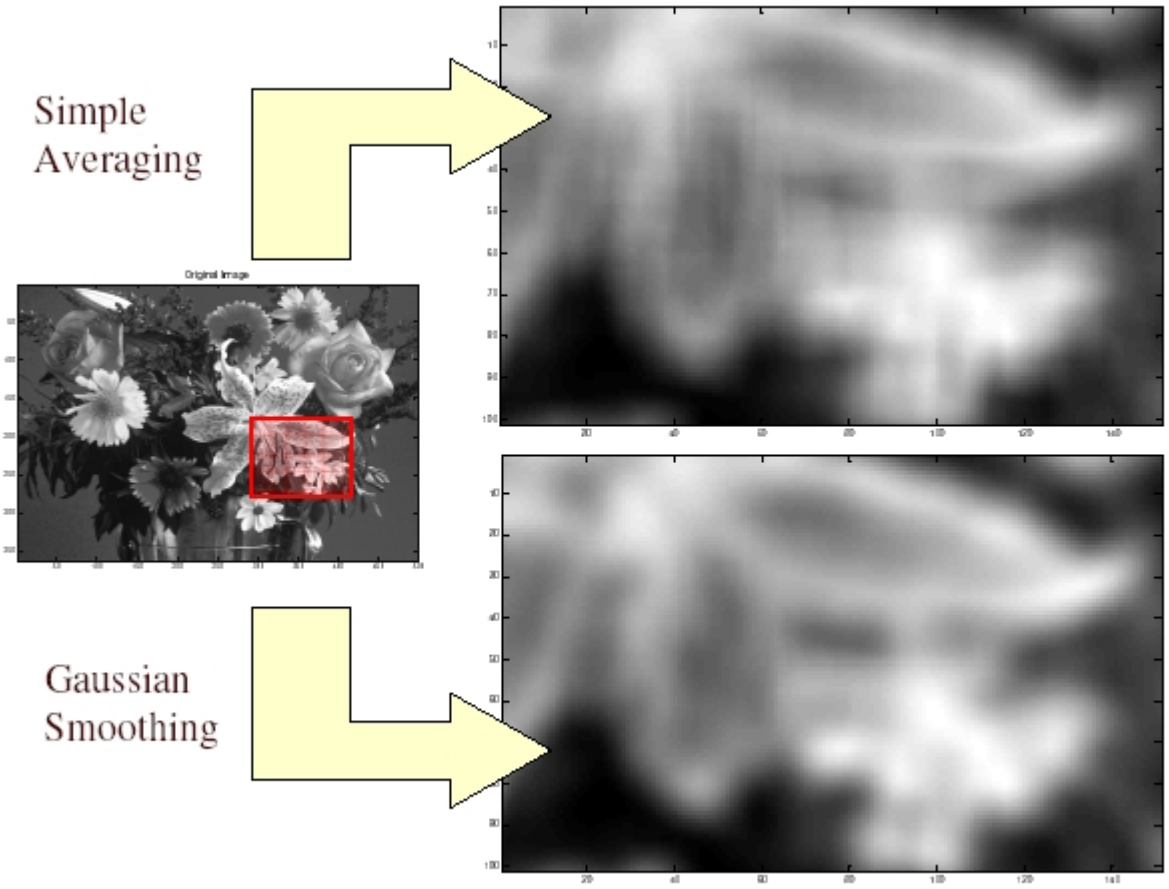
Result of blurring using a uniform local model

Produces a set of narrow vertical horizontal and vertical bars – ringing effect

Result of blurring using a set of Gaussian weights

adapted from David Forsyth, UC Berkeley

Smoothing with a Gaussian



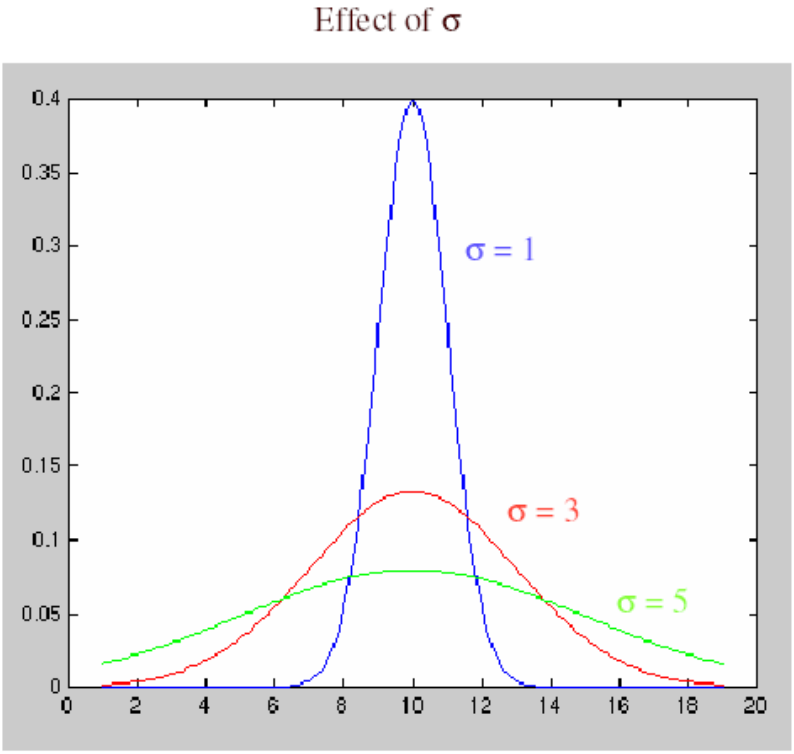
adapted from Martial Hebert, CMU

Smoothing with a Gaussian

If σ is small : the smoothing will have little effect

If σ is larger : neighboring pixels will have larger weights resulting in consensus of the neighbors

If σ is very large : details will disappear along with the noise

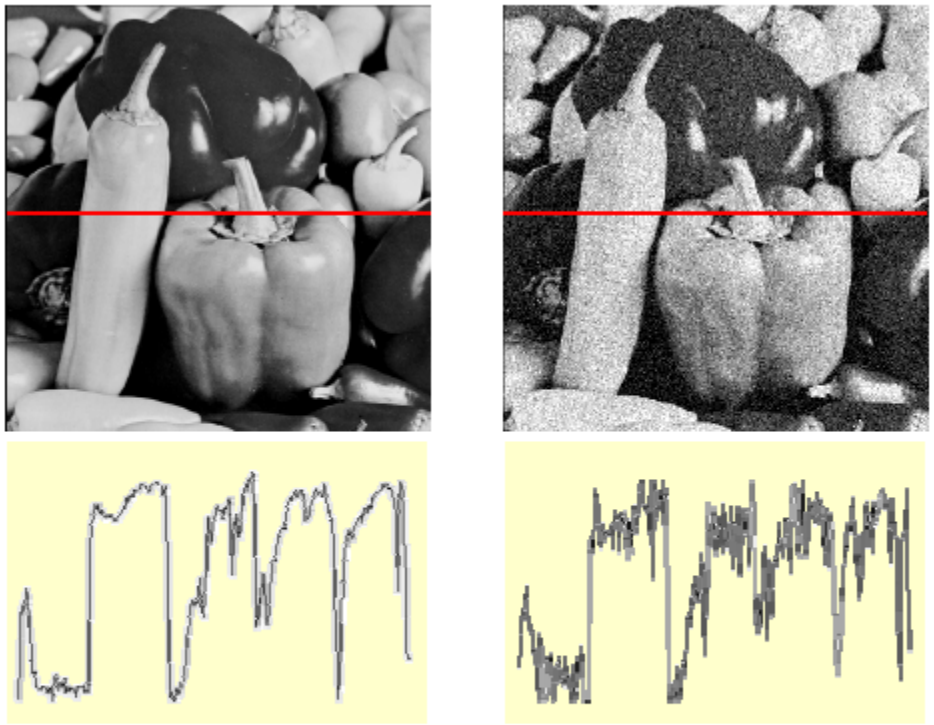


adapted from Martial Hebert, CMU

Gaussian smoothing to remove noise

Image

Noise

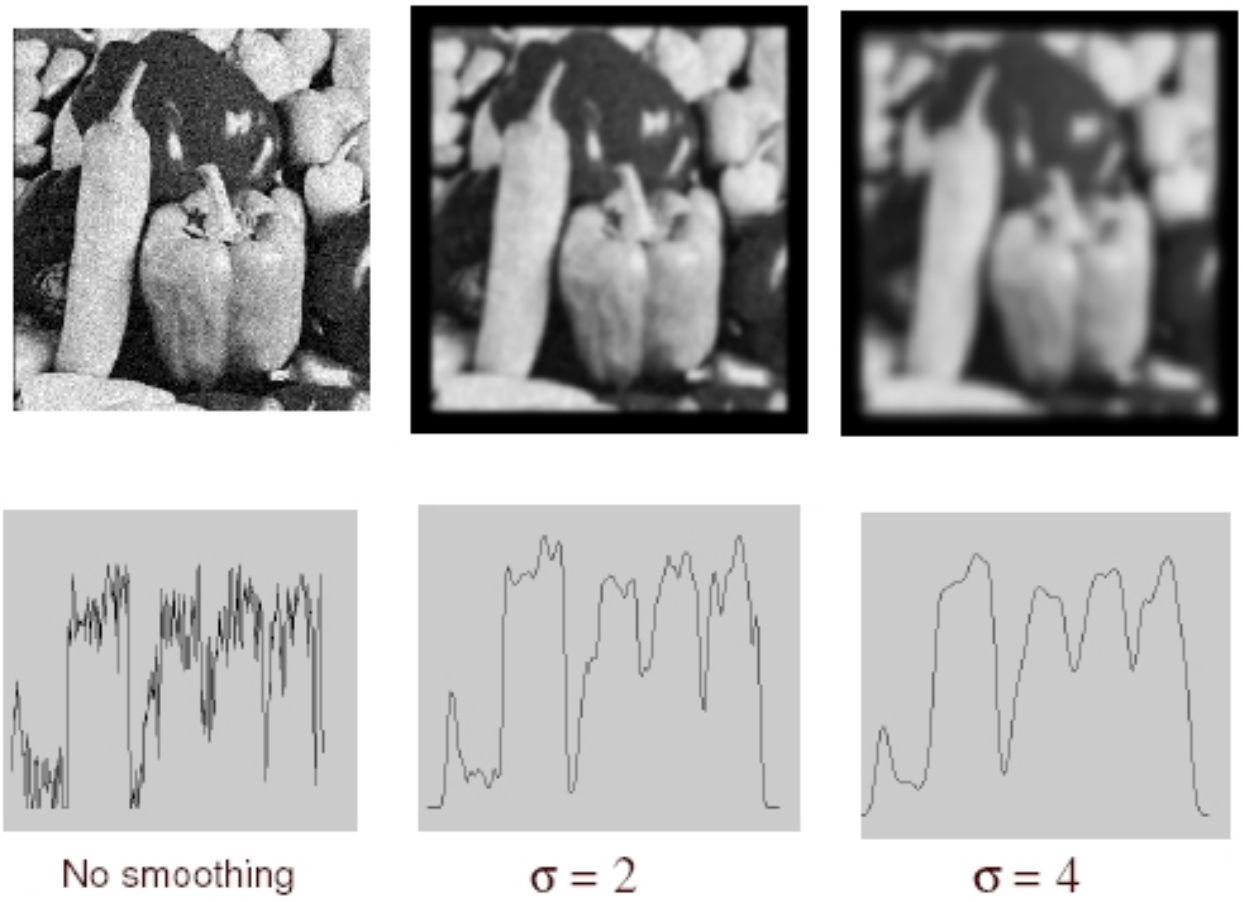


$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

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

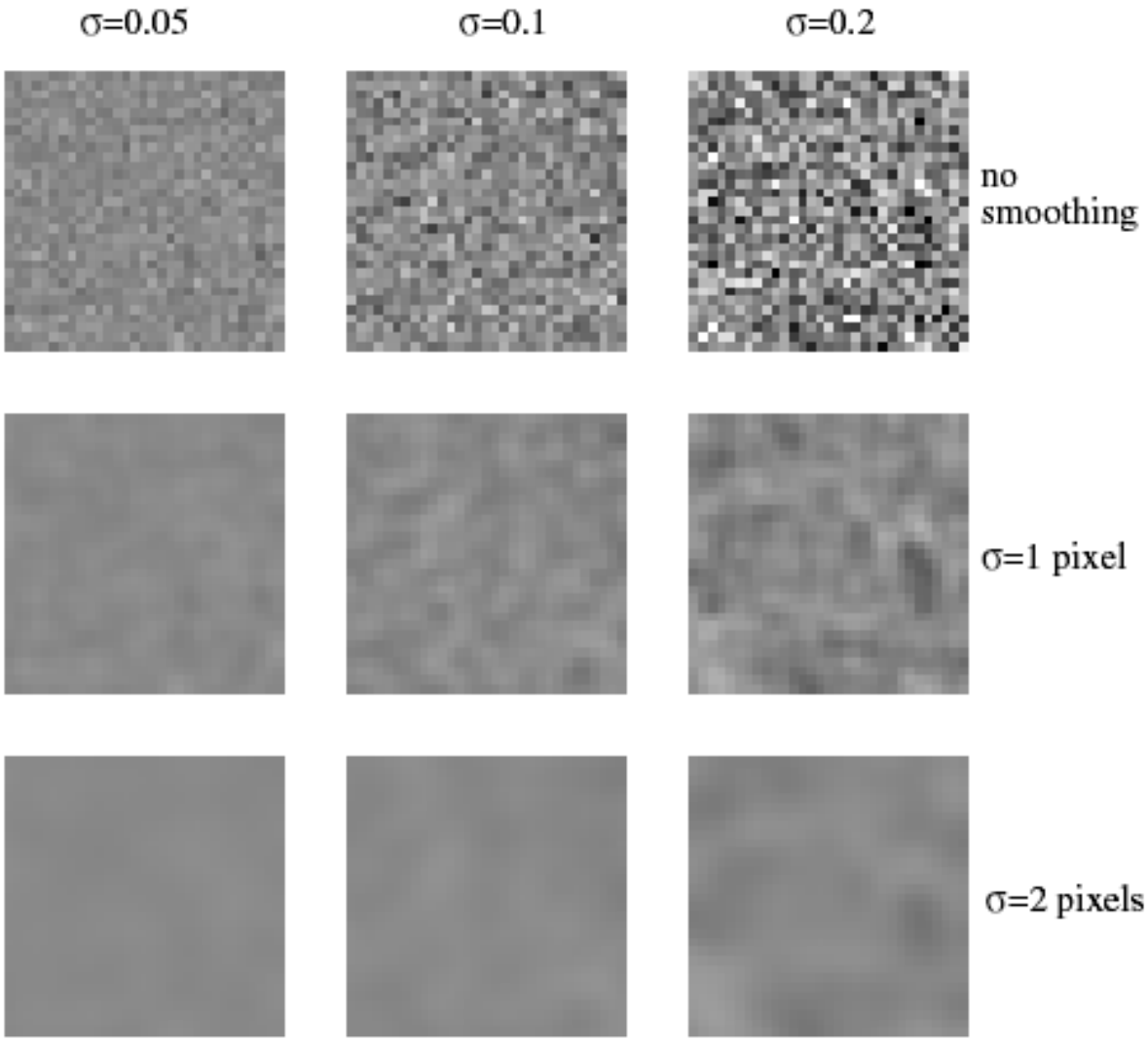
adapted from Martial Hebert, CMU

Gaussian smoothing to remove noise



adapted from Martial Hebert, CMU

Smoothing with a Gaussian

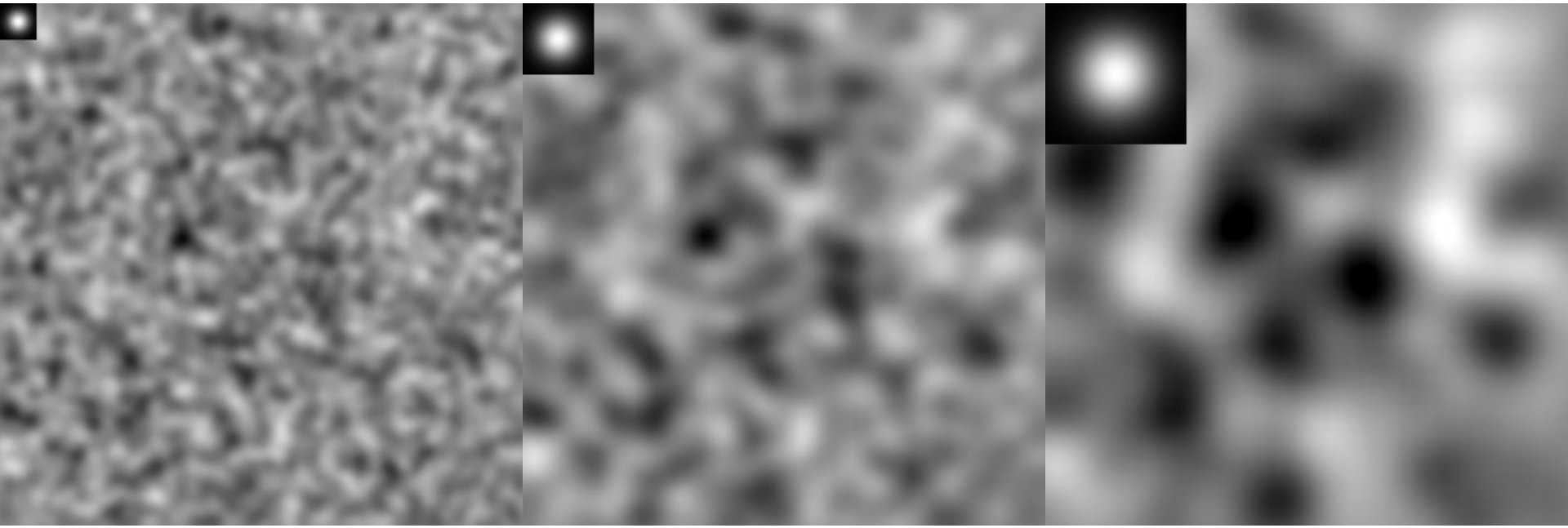


adapted from David Forsyth, UC Berkeley

The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

Smoothing with a Gaussian

- Filtered noise is sometimes useful
  - looks like some natural textures, can be used to simulate fire, etc.



adapted from David Forsyth, UC Berkeley

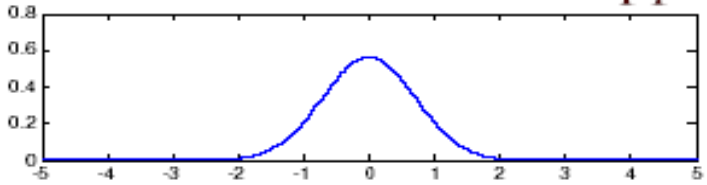
Smoothing with a Gaussian

Gaussian kernel

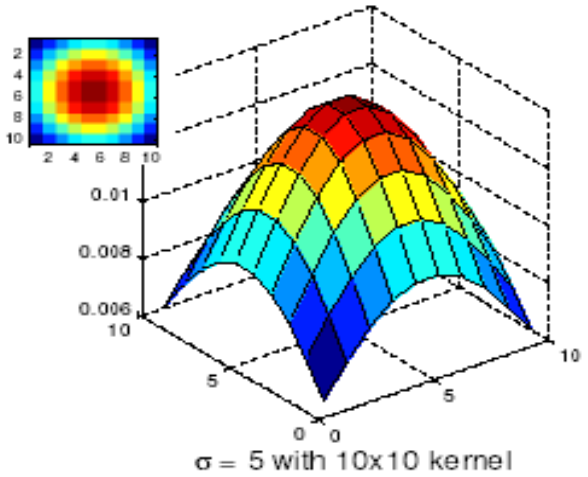
$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{((i-k-1)^2 + (j-k-1)^2)}{2\sigma^2}\right)$$

Note about Finite Kernel Support

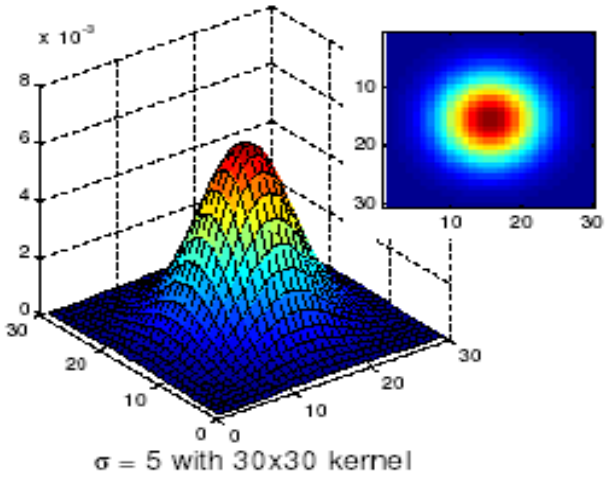
- Gaussian function has infinite support



- In discrete filtering, we have finite kernel size



σ = 5 with 10x10 kernel



σ = 5 with 30x30 kernel

2k+1 x 2k+1 window

If σ is large k should be large

adapted from Martial Hebert, CMU

Gaussian kernel

$$g(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$

0.0751	0.1238	0.0751
0.1238	0.242	0.1238
0.0751	0.1238	0.0751

Gaussian is an approximation to the binomial distribution.

Can approximate Gaussian using binomial coefficients.

$$a_{nr} \equiv \frac{n!}{r!(n-r)!} \equiv \binom{n}{r}$$

1X3 filter: n=(3-1)=2, r=0,1,2

1	2	1

n = number of elements in the 1D filter minus 1

r = position of element in the filter kernel (0, 1, 2…)

g = 1/4

1	2	1

g' g =

0.0625	0.1250	0.0625
0.1250	0.2500	0.1250
0.0625	0.1250	0.0625

adapted from Michael Black

Smoothing Edges

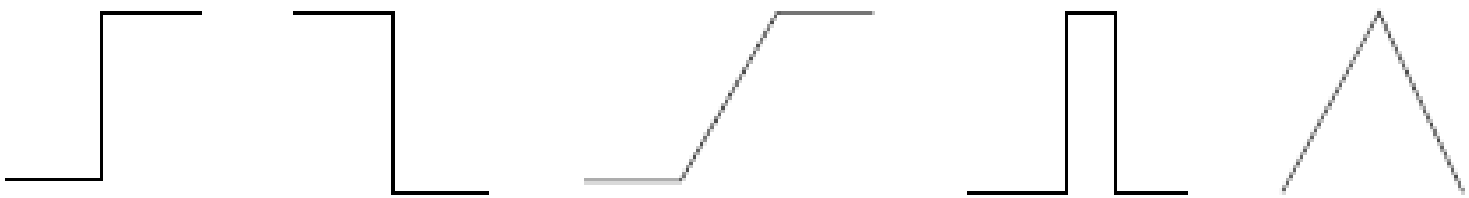
Box

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

Gaussian

$$1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Edge Types



upward step edge | downward step edge | upward ramp | impulse or line | tent

adapted from Shapiro & Stockman

Smoothing Edges

box smoothing mask M $= [1/3, 1/3, 1/3]$

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	12	12	12	12	16	20	24	24	24	24

(a) S_1 is an upward step edge

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	12	12	12	16	16	16	12	12	12	12

(d) S_4 is a bright impulse or "line"

adapted from Shapiro & Stockman

Smoothing Edges

Gaussian smoothing mask M = [1/4, 1/2, 1/4]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	12	12	12	12	15	21	24	24	24	24

(a) S_1 is an upward step edge

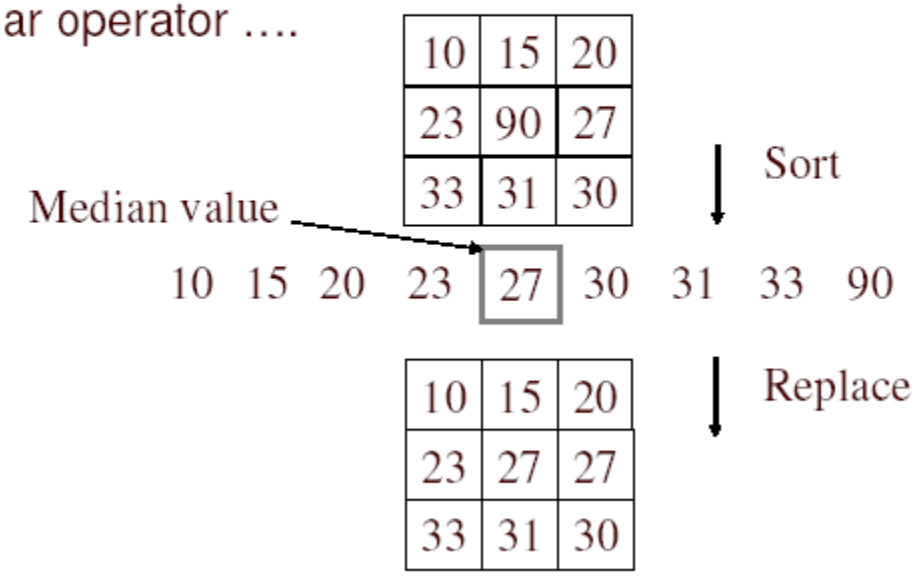
S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	12	12	12	15	18	15	12	12	12	12

(d) S_4 is a bright impulse or "line"

adapted from Shapiro & Stockman

Median Filtering

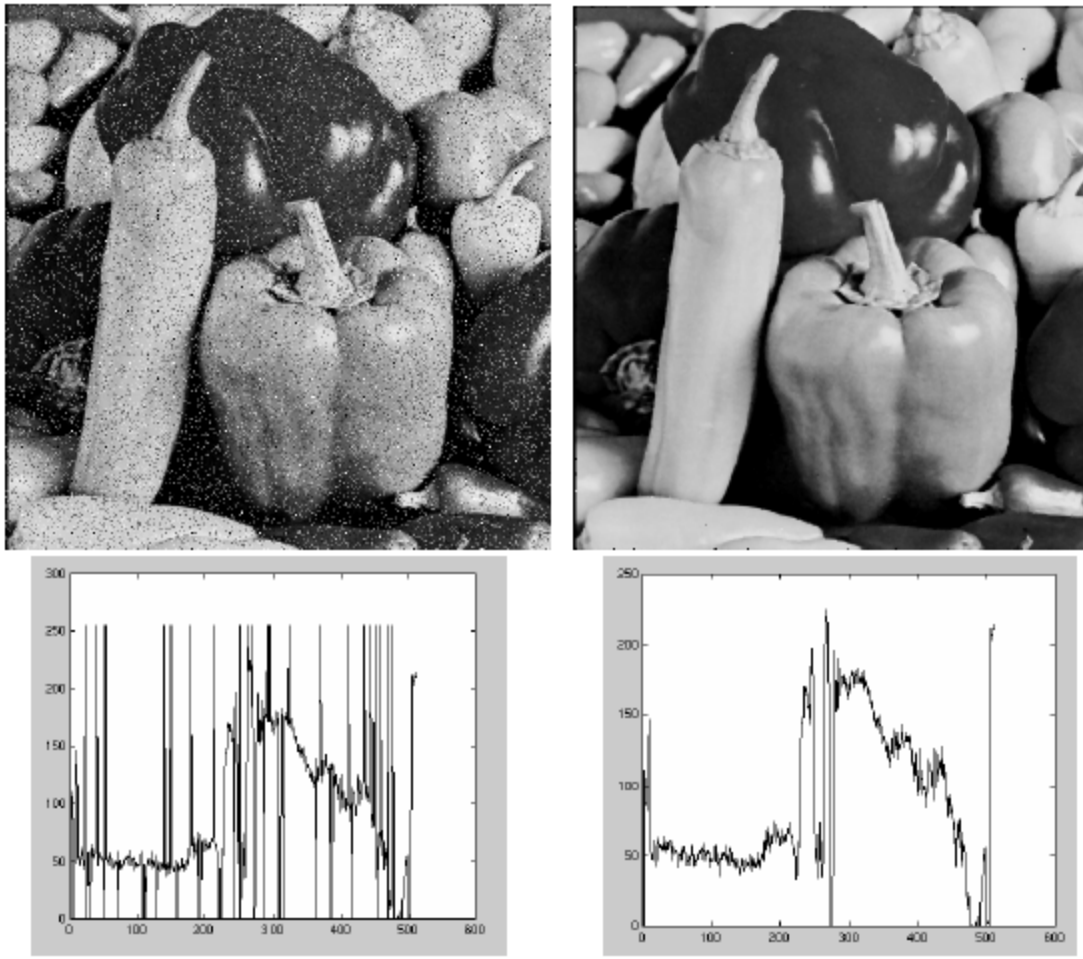
- Good: Does not smooth edges. True edges remain sharp.
- does remove isolated outliers
- Bad: *Not* a linear operator ….



adapted from Martial Hebert, CMU

Median Filtering

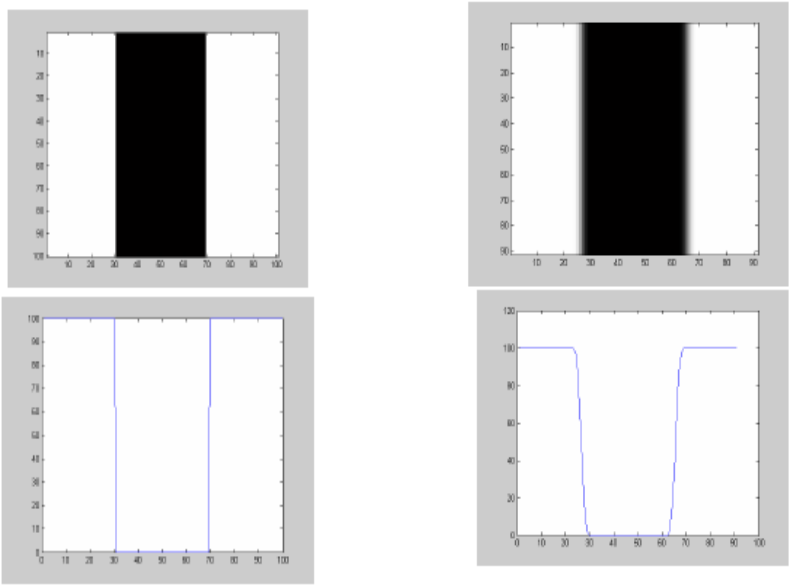
Effect of median filter on salt and pepper noise



adapted from Martial Hebert, CMU

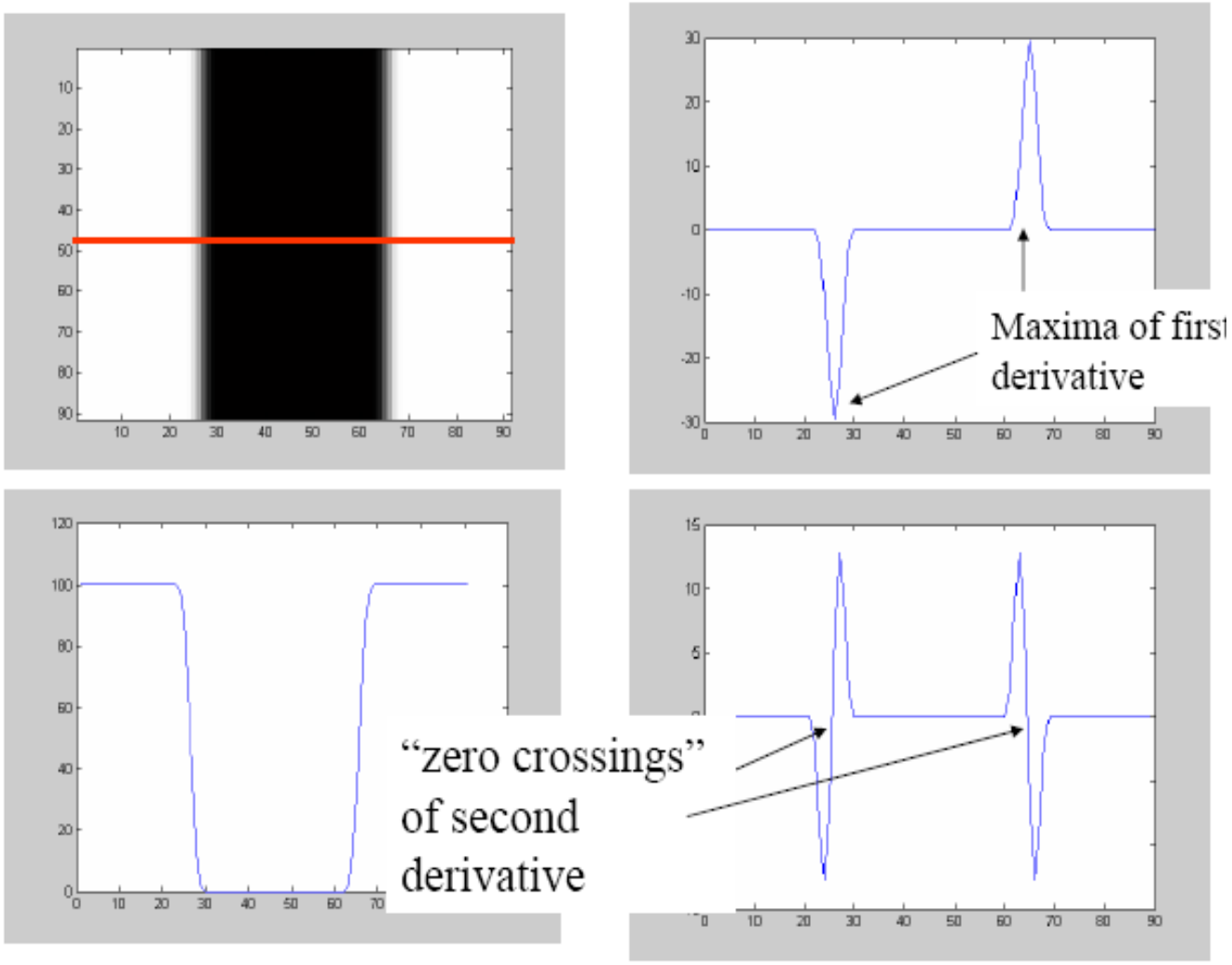
Edges

- Correspond to fast changes
  - Where the magnitude of the derivative is large



adapted from Michael Black

Derivatives



adapted from Michael Black

Differentiation and Convolution

Derivatives can be approximated by a convolution

- Recall

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

- Now this is linear and shift invariant, so must be the result of a convolution.

- We could approximate this as

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

- (which is obviously a convolution; it's not a very good way to do things, as we shall see)

adapted from David Forsyth, UC Berkeley

Finite Differences

- Same as a convolution with a kernel

$H =$

0	0	0
1	0	-1
0	0	0

This kernel gives large positive response to an image configuration which is positive on one side and negative on the other side and a large negative response to the mirror image

Finite Differences

M = [-1 0 1]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	-12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	3	6	6	6	3	0	0

(c) S_3 is an upward ramp

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	12	0	-12	0	0	0	0

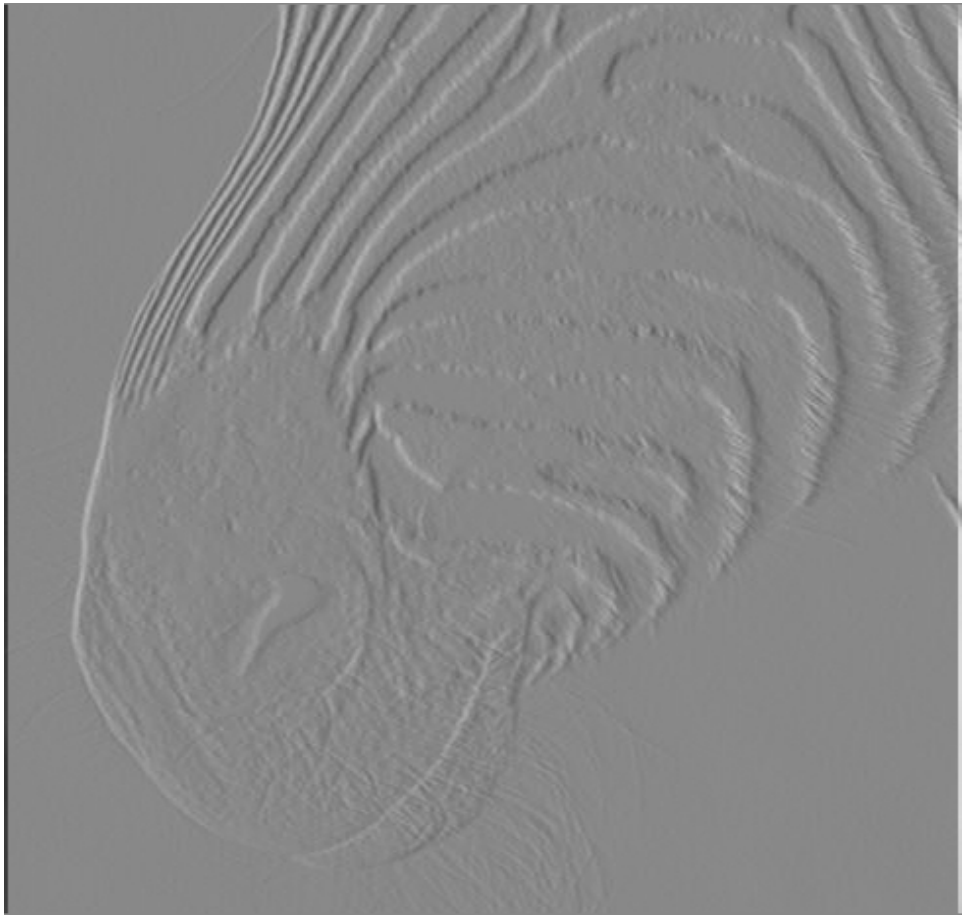
(d) S_4 is a bright impulse or "line"

adapted from Shapiro & Stockman

Finite Differences

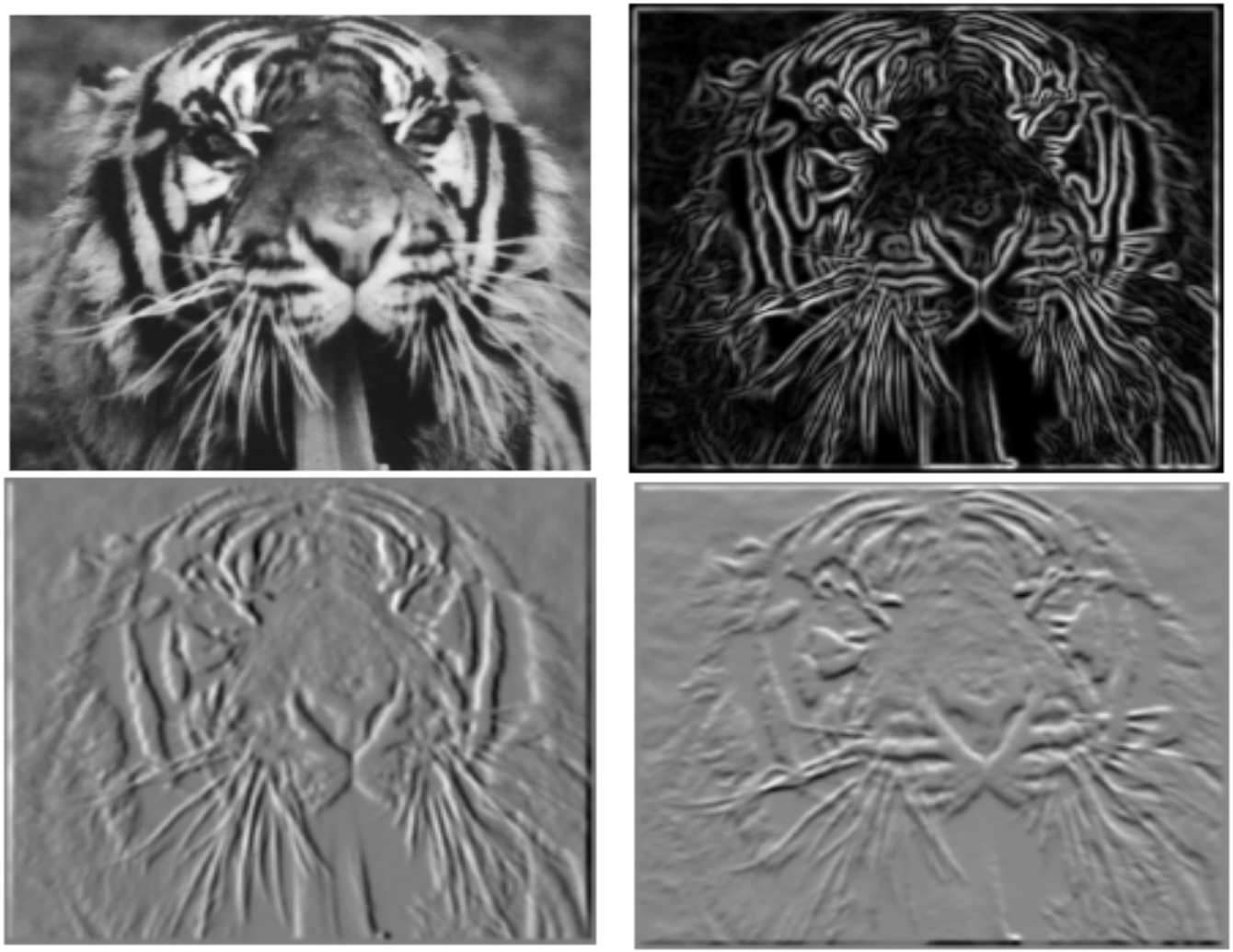
darker is negative, lighter is positive, and mid grey is zero.

Which derivative is it, x or y ?



adapted from David Forsyth, UC Berkeley

Finite Differences



Derivative in the x direction

Derivative in the y direction

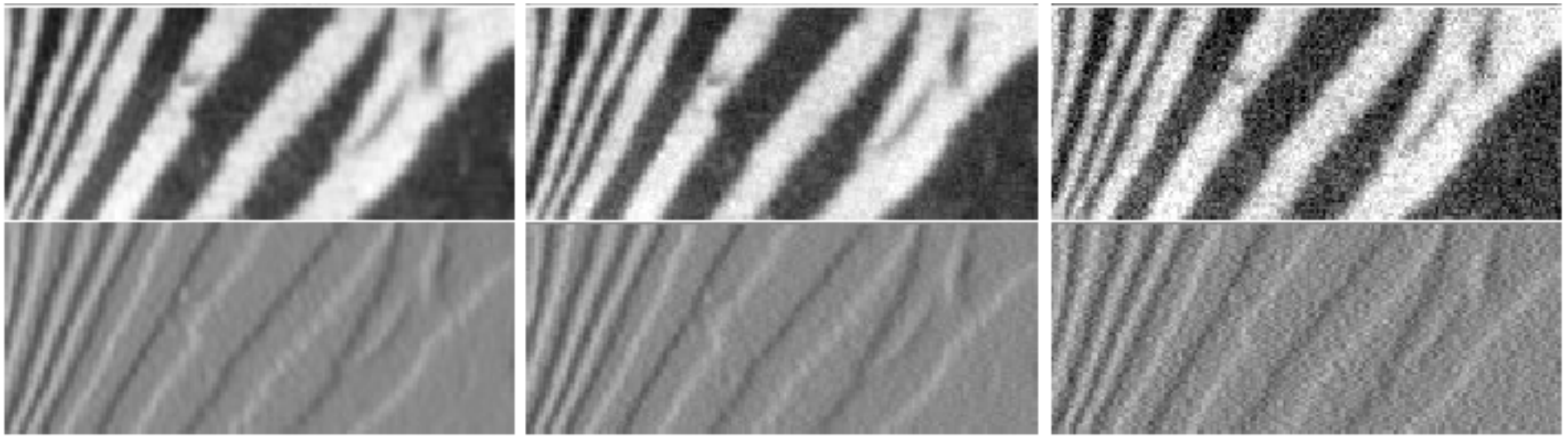
adapted from Martial Hebert, CMU

Finite differences

- The most unsatisfactory estimate of a derivative

- Because they respond strongly to the fast changes, which is a characteristic of noise (they look very different from their neighbors)

Finite differences responding to noise



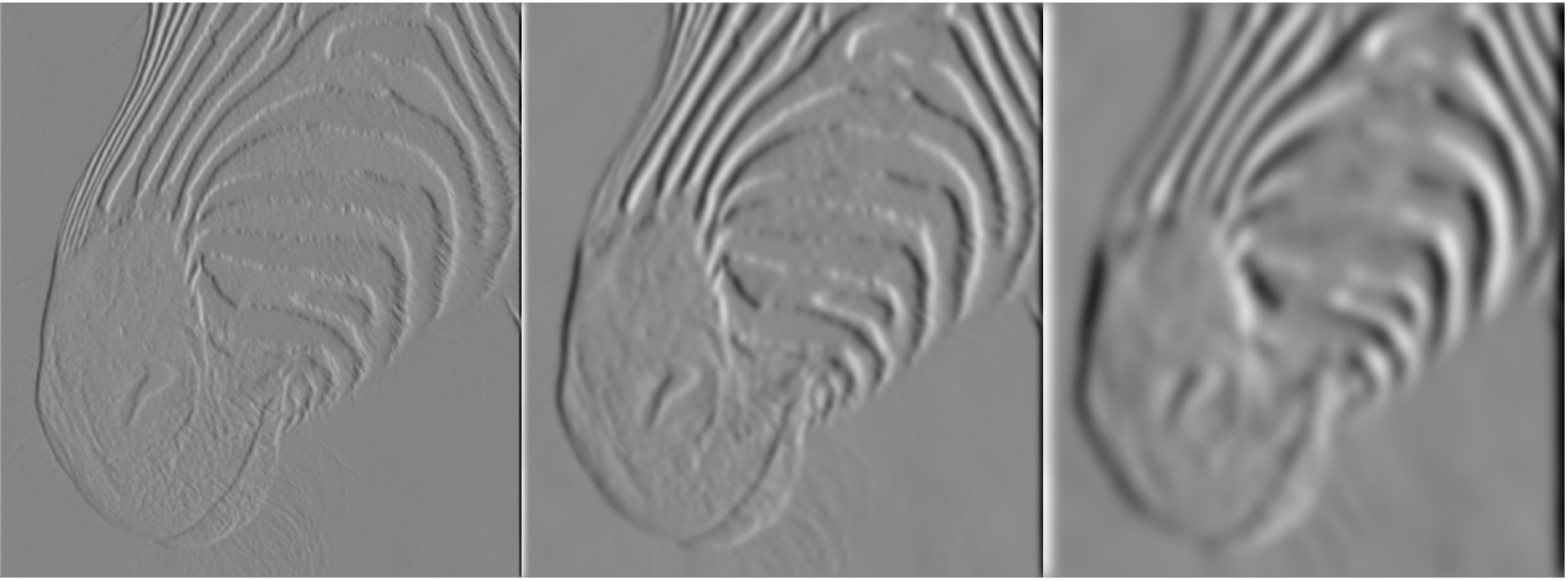
adapted from David Forsyth, UC Berkeley

Smoothed derivatives

- Solution : Smooth before differentiating

- intuitively, most pixels in images look quite a lot like their neighbors
- this is true even at an edge; along the edge they're similar, across the edge they're not
- suggests that smoothing the image should help, by forcing pixels that are different from their neighbors (=noise pixels?) to look more like neighbors

Smoothed derivatives



1 pixel 3 pixels 7 pixels

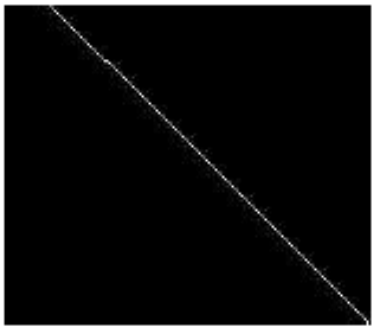
The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.

adapted from David Forsyth, UC Berkeley

Smoothed derivatives

There is ALWAYS a tradeoff between smoothing and good edge localization!

Image with Edge



Edge Location

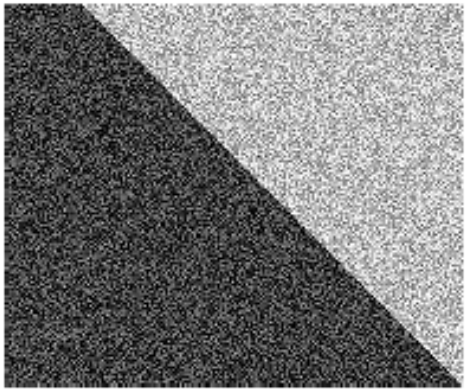
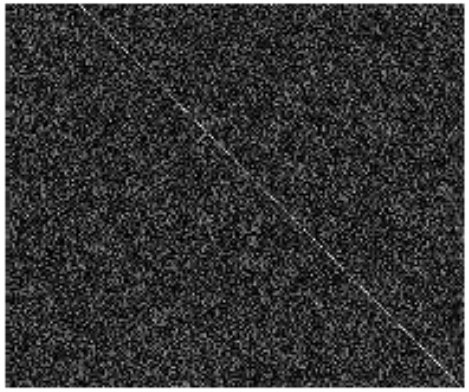
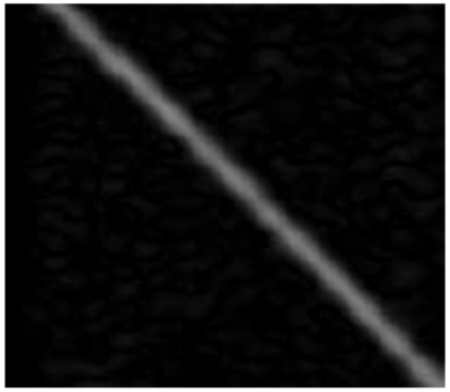


Image + Noise



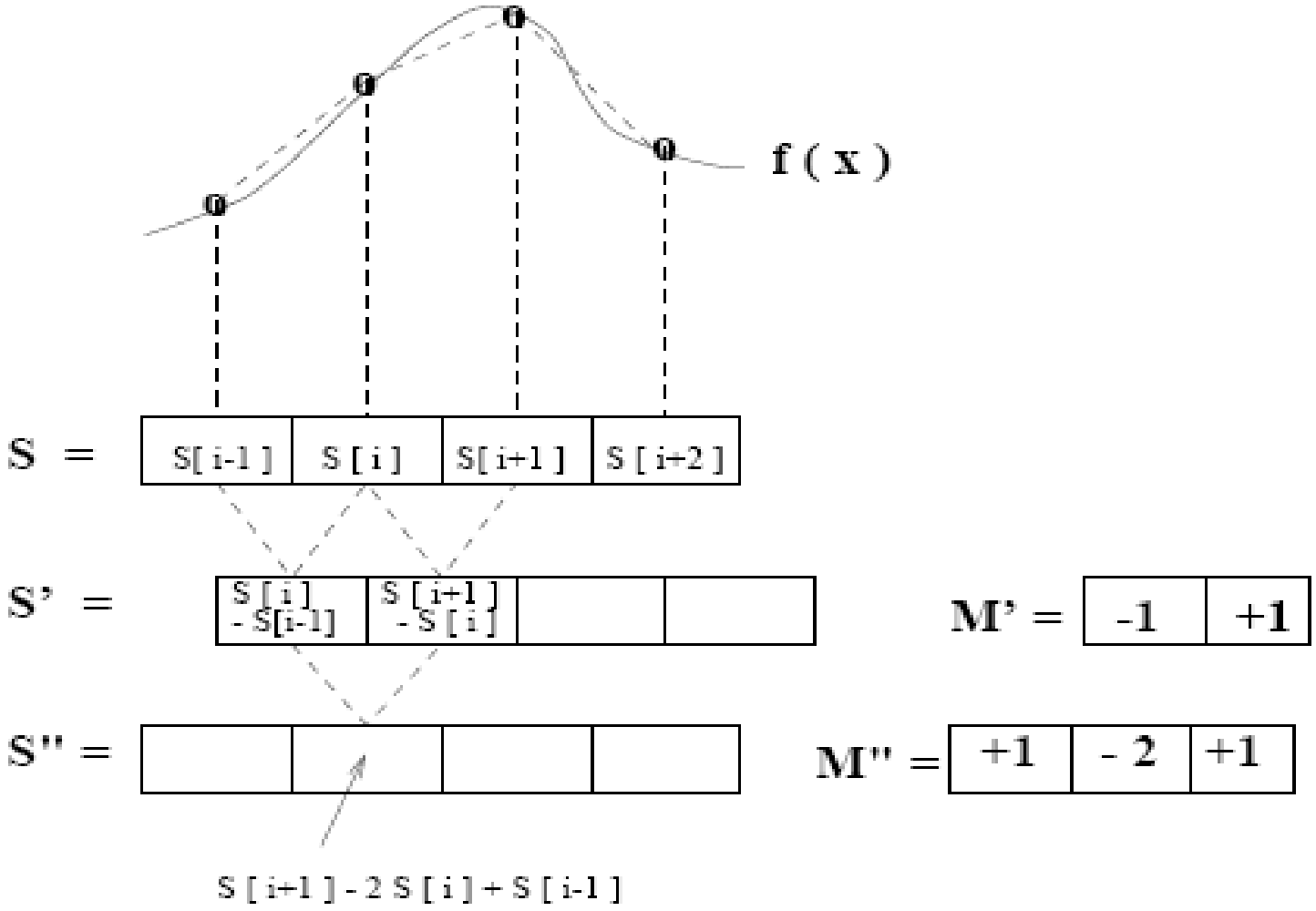
Derivatives detect edge *and* noise



Smoothed derivative removes noise, but blurs edge

adapted from Martial Hebert, CMU

Second Derivatives



adapted from Shapiro & Stockman

Second Derivatives

M = [-1 2 -1]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	-12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	-3	0	0	0	3	0	0

(c) S_3 is an upward ramp

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	-12	24	-12	0	0	0	0

(d) S_4 is a bright impulse or "line"

adapted from Shapiro & Stockman

Common masks for computing gradient

Prewitt: $M_x =$

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

; $M_y =$

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

Sobel: $M_x =$

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

; $M_y =$

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

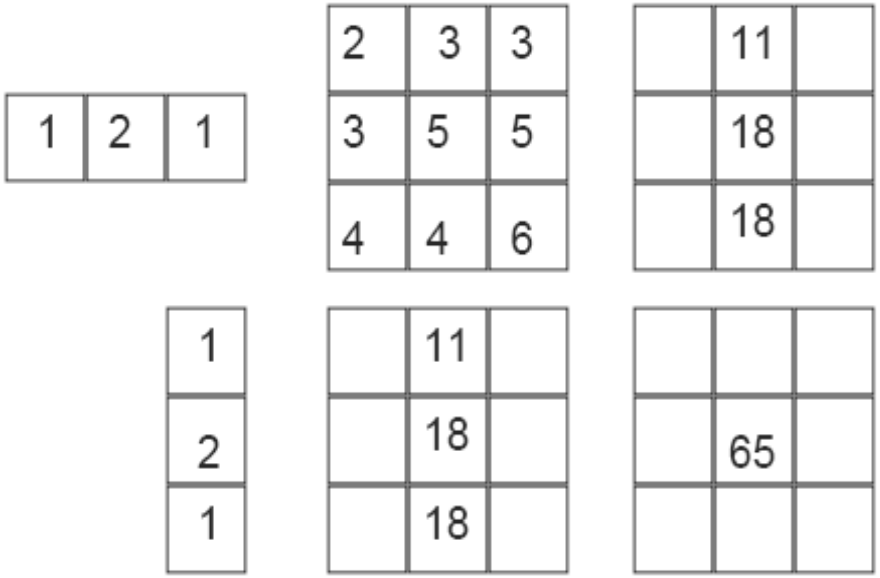
Roberts: $M_x =$

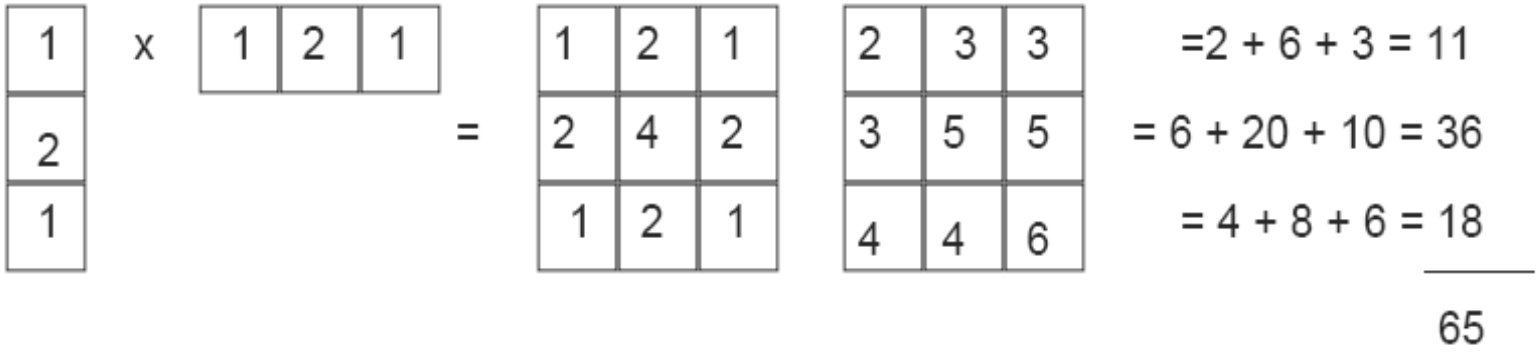
0	1
-1	0

; $M_y =$

1	0
0	-1

Separability





Advantages of separability

First convolve the image with a one dimensional horizontal filter

Then convolve the result of the first convolution with a one dimensional vertical filter

For a kxk Gaussian filter, 2D convolution requires k^2 operations per pixel

But using the separable filters, we reduce this to 2k operations per pixel.

adapted from Larry Davis, University of Maryland

Seperable Gaussian

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-x^2/(2\sigma^2))$$

$$g(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-y^2/(2\sigma^2))$$

Product?

$$g(x,y) = \frac{1}{2\pi\sigma^2} \exp(-(x^2+y^2)/(2\sigma^2))$$

Advantages of Gaussians

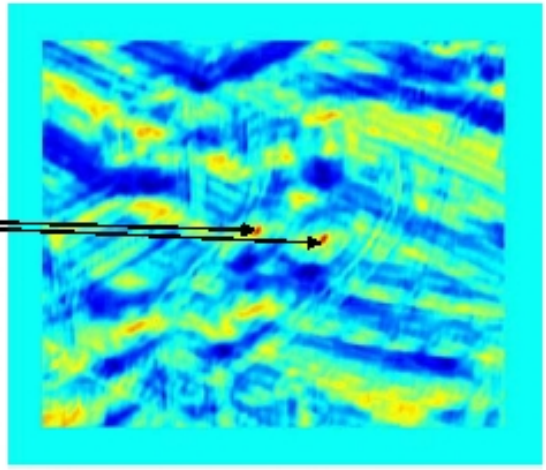
- Convolution of a Gaussian with itself is another Gaussian
  - so we can first smooth an image with a small Gaussian
  - then, we convolve that smoothed image with another small Gaussian and the result is equivalent to smoother the original image with a larger Gaussian.
  - If we smooth an image with a Gaussian having sd σ twice, then we get the same result as smoothing the image with a Gaussian having standard deviation $(2\sigma)^{1/2}$

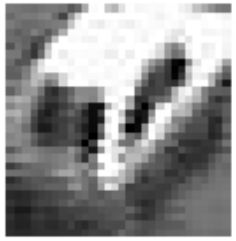
adapted from Larry Davis, University of Maryland

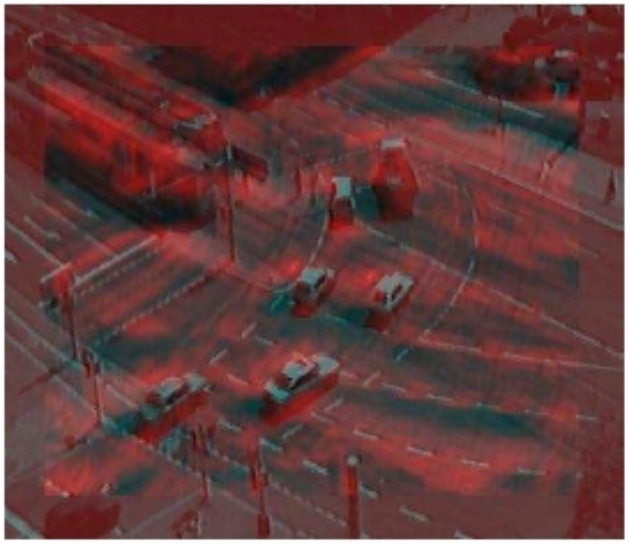
Application

Figure from "Computer Vision for Interactive Computer Graphics," W.Freeman et al, IEEE Computer Graphics and Applications, 1998 copyright 1998, IEEE

Normalized Correlation







Normalized Correlation

Scaled Representations

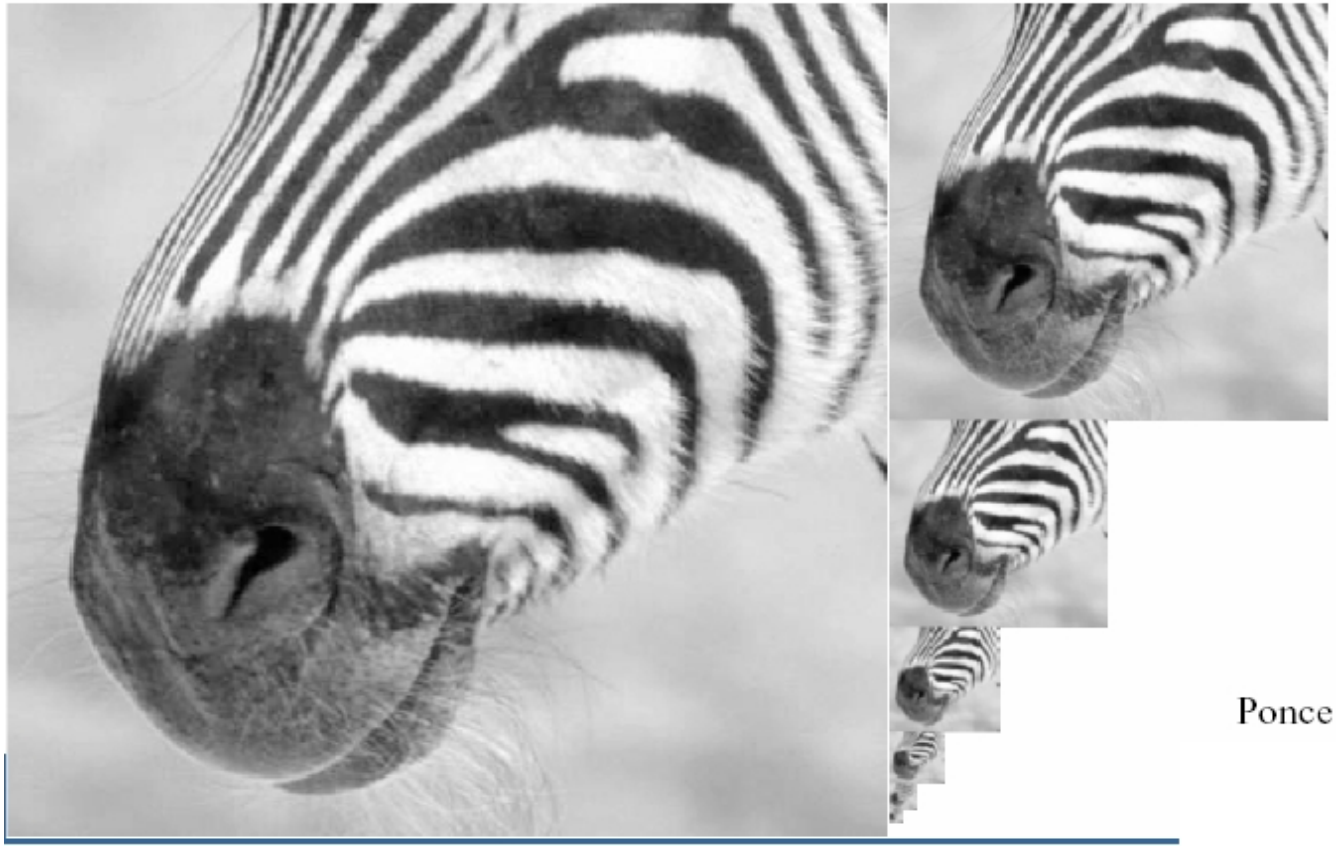
- Big bars (resp. spots, hands, etc.) and little bars are both interesting
  - Stripes and hairs, say
- Inefficient to detect big bars with big filters
  - And there is superfluous detail in the filter kernel

- Alternative:
  - Apply filters of fixed size to images of different sizes
  - Typically, a collection of images whose edge length changes by a factor of 2 (or root 2)
  - This is a pyramid (or Gaussian pyramid) by visual analogy

adapted from David Forsyth, UC Berkeley

Scaled Representation – Gaussian Pyramid

512 256 128 64 32 16 8



A bar in the big images is a hair on the zebra's nose; in smaller images, a stripe; in the smallest, the animal's nose

Ponce & Forsyth

Scaled representation

Motivation for studying scale.

ELDER AND ZUCKER: LOCAL SCALE CONTROL FOR EDGE DETECTION AND BLUR ESTIMATION

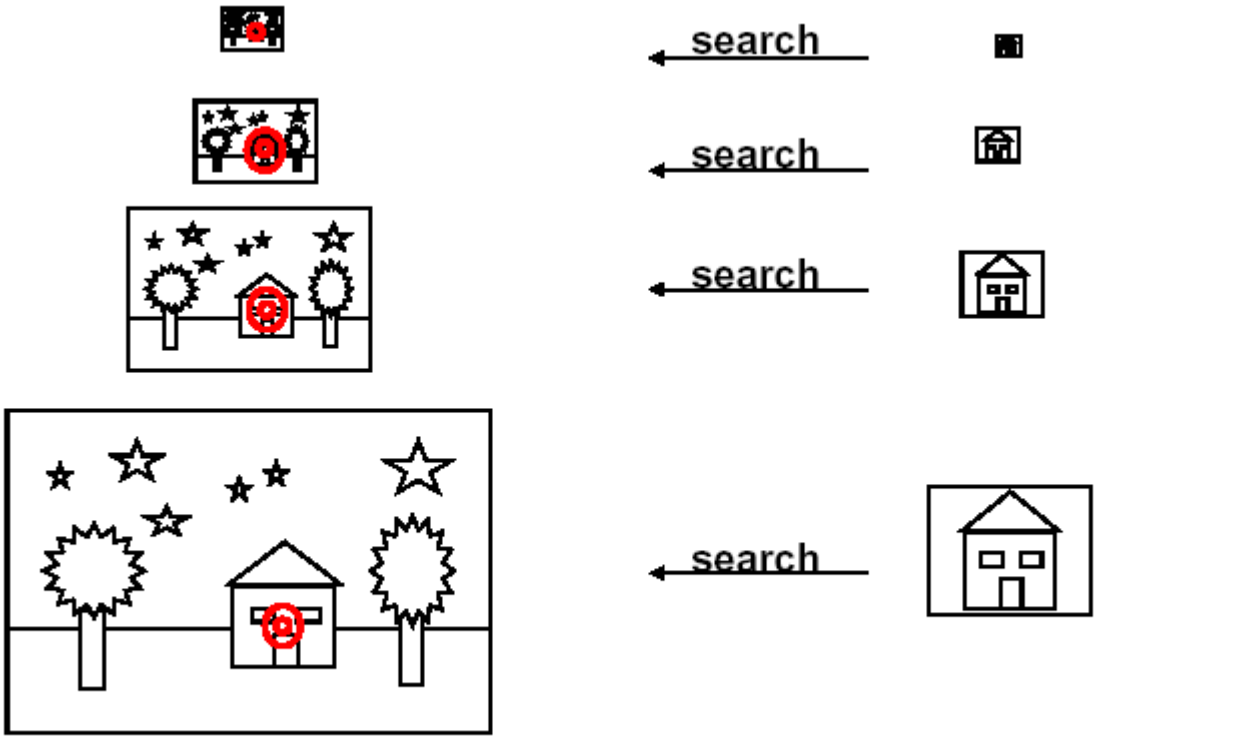
IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 20, NO. 7, JULY 1998

Applications of scaled representations

- Search for correspondence
  - look at coarse scales, then refine with finer scales
- Edge tracking
  - a “good” edge at a fine scale has parents at a coarser scale
- Control of detail and computational cost in matching
  - e.g. finding stripes
  - terribly important in texture representation

adapted from David Forsyth, UC Berkeley

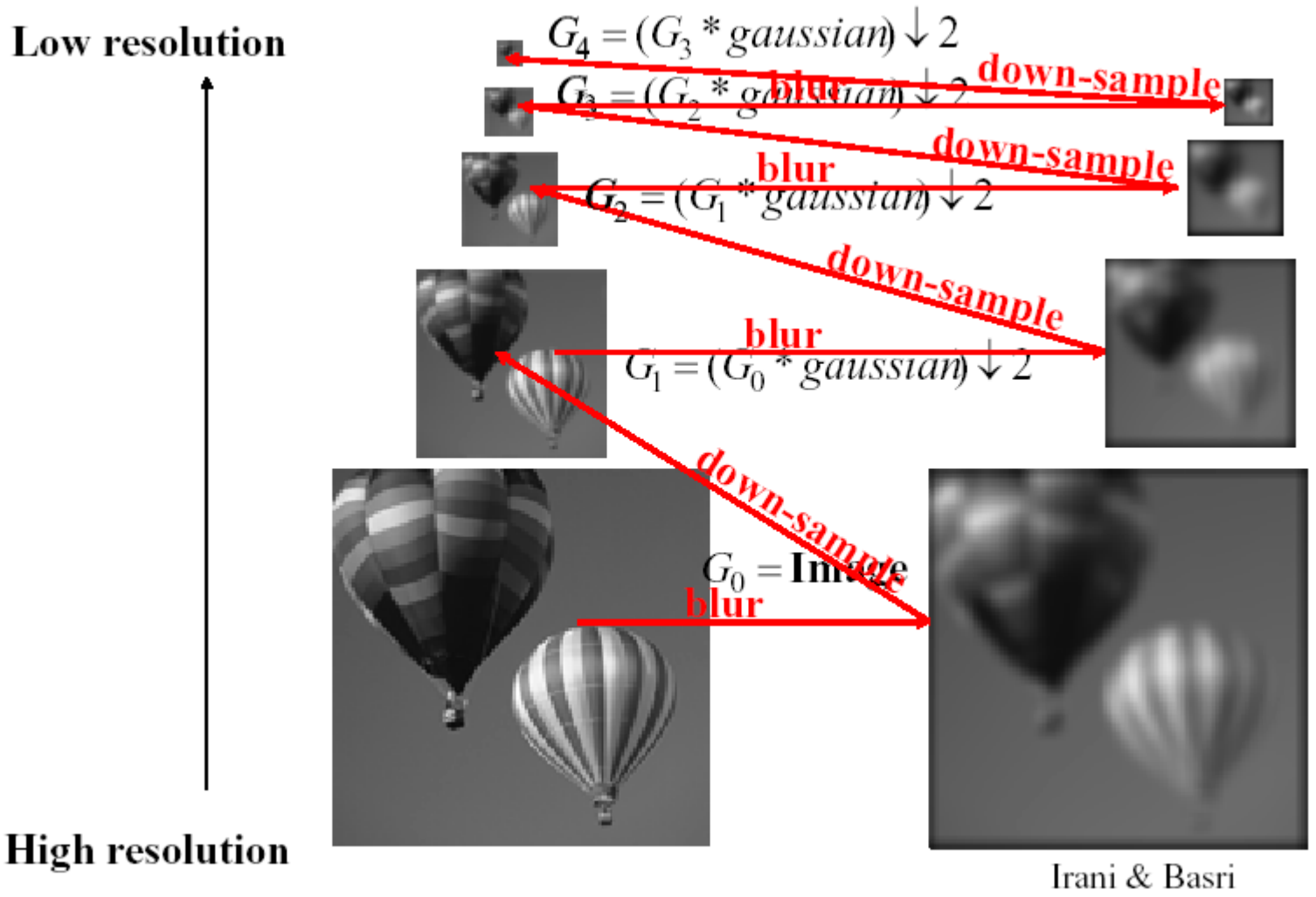
Motivation : Search



Irani & Basri

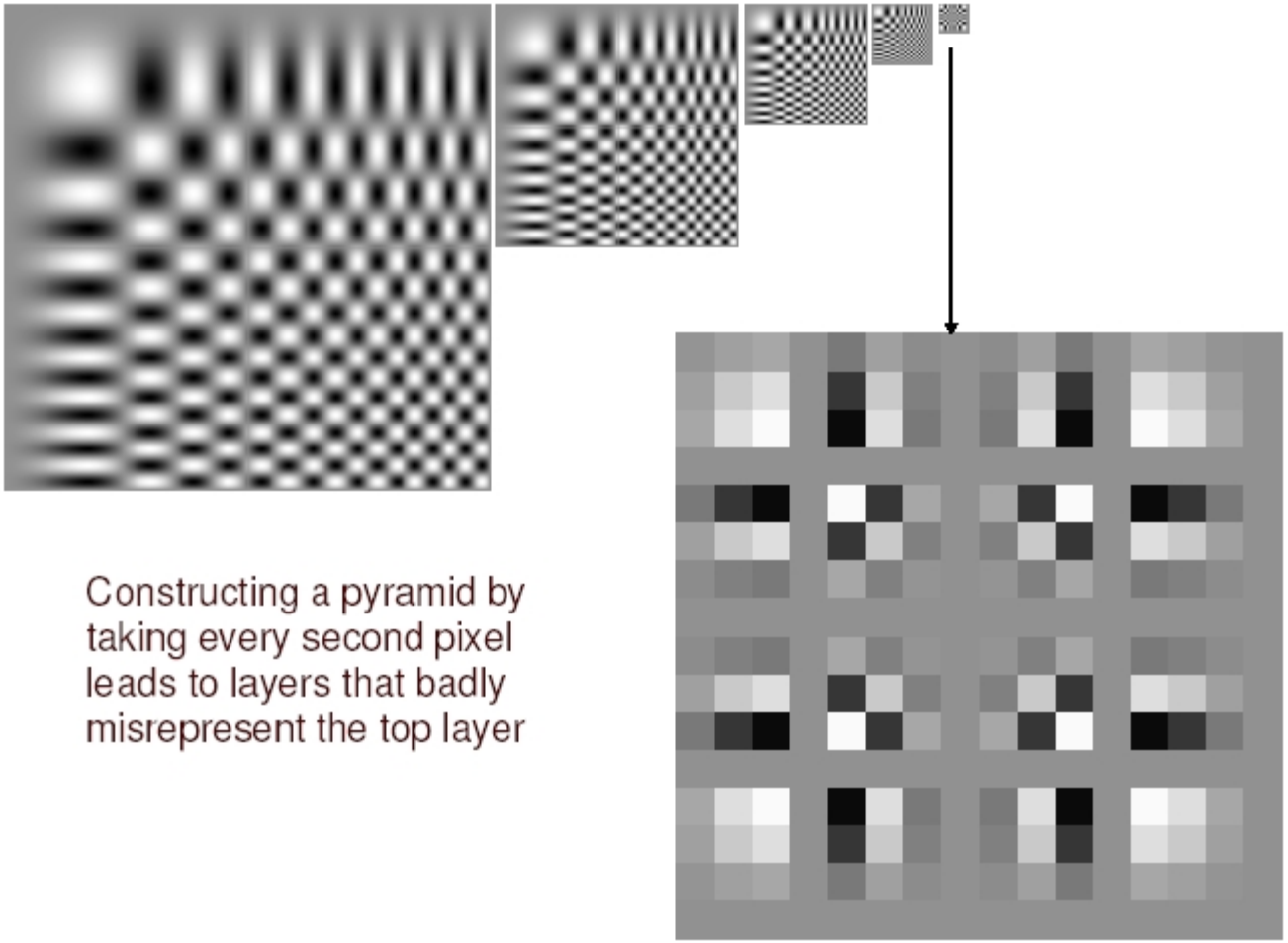
adapted from Michael Black, Brown University

Gaussian Pyramid



adapted from Michael Black, Brown University

Problems



Constructing a pyramid by taking every second pixel leads to layers that badly misrepresent the top layer

adapted from David Forsyth, UC Berkeley

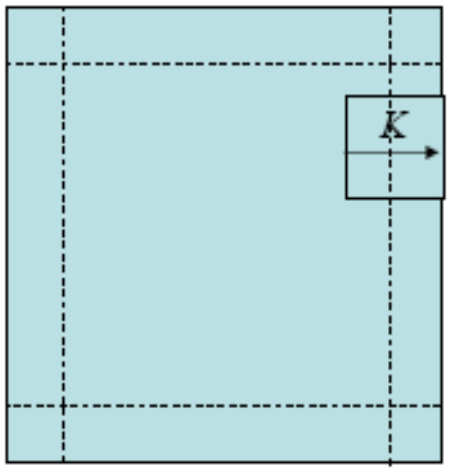
Programming issues

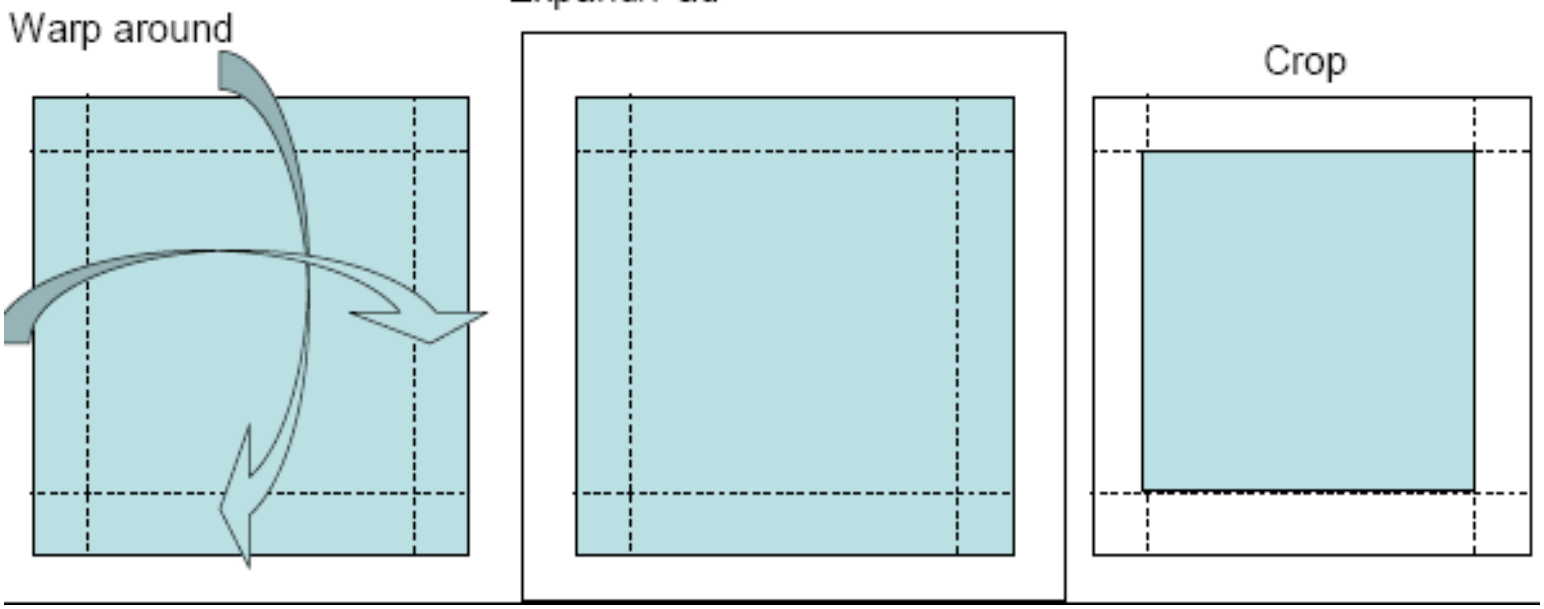
Edge effects in discrete convolutions

- Ignore these locations
  - Output is smaller than input
- pad the image with constant values
  - Gradient effects near the boundary
- Pad the image in some other way
  - For example make m+1 = (m-1)th column
    - Creates appearance of substantial second derivatives at the boundary
  - Make the image a cylinder

Programming Issues

- MATLAB: `conv` (1D) or `conv2` (2D)
- Border issues:
  - When applying convolution with a KxK kernel, the result is undefined for pixels closer than K pixels from the border of the image
- Options:





Linear Filters

Whatever the weights, the output is

- *Shift invariant* :
  - The value of the output depends on the pattern in an image neighborhood
  - Not to the position of the neighborhood
- *Linear* :
  - the output for the sum of two images is the same as the the sum of the outputs obtained from the images separately.

This procedure is called *linear filtering*

Shift Invariant Linear Systems

- Superposition:
  - R(f+g) = R(f) + R(g)
  - The response to the sum of stimuli is the sum of individual responses
- Scaling:
  - R(kf) = kR(f)
  - The response to a zero input is zero
  - The response to a scaled stimulus is a scaled version of the response to the original stimulus
- Superposition + scaling --> linear

Shift Invariant Linear Systems

- Shift invariance:
  - The response to a translated stimulus is the translation of the response to the stimulus

- Linear + shift invariant --> shift invariant linear system

Discrete convolution in 1D

In infinite input vector **f**
Represent as a weighted sum of basis elements
$\mathbf{e_0}$ = ...0 0 0 1 0 0 0...

Shift(**f**,i) : jth component is the j-ith component of **f**

$\mathbf{f} = \sum_i f_i \text{ Shift}(e_o, i)$

Discrete convolution in 1D

R(**f**) : response of the system to vector f
since shift invariant
R(Shift(**f**,k)) = Shift(R(**f**),k)

since linear
R(kf) =kR(f)

$$R(f) = R(\sum_i f_i \, Shift(e_o, i))$$
$$= \sum_i R(f_i \, Shift(e_o, i))$$
$$= \sum_i f_i \, R(Shift(e_o, i))$$
$$= \sum_i f_i \, Shift(R(e_o), i)$$

To obtain the system's response to any vector we only need to know it's impulse response

Discrete convolution

If impulse response is written as **g**
$R(\mathbf{f}) = \sum_i f_i \, \text{Shift}(\mathbf{g}, i)$

$Rj = \sum_i g_{j-i} f_i$
which we write as g*f

Discrete convolution in 2D

$E_{oo} =$
.....
... 0 0 0 ...
... 0 1 0 ...
... 0 0 0 ...
.....

$Rij = \sum_{u,v} G_{i-u,\, j-v} \, F_{uv}$

which we write as G ** F

Continuous convolution in 1D

Take a discrete function and change each value with a box
Make the boxes narrower

$R(f) \sim R(f)(u)$
$Shift(f,c) = f(u-c)$
$R(Shift(f,c)) = Shift(R(f), c)$

A box function
$box(x) = 0 \quad abs(x) > \varepsilon/2$
$\qquad\qquad 1 \quad abs(x) < \varepsilon/2$

$f = \sum_i f_i \, Shift(box, xi)$

Continuous convolution in 1D

Following similar equations as in 1D

$R(\sum_i f_i \text{ Shift}(box, xi)) = \sum_i f_i \text{ Shift}(R(box/\varepsilon), xi)\varepsilon$

approximate integral if $\varepsilon \to 0$

Introduce δ-function -> $\lim(\varepsilon \to 0)\ box(x)/\varepsilon$

Response of a system to δ-function is zero except on a finite number of intervals of finite length

e.g. In 2D extremely small extremely bright light

$R(f) = \int\{R(\delta)(u-x')\}f(x')dx'$

$R(f) = \int g(u-x')f(x')dx'$

Where g is the response of the system to δ-function -> impulse response