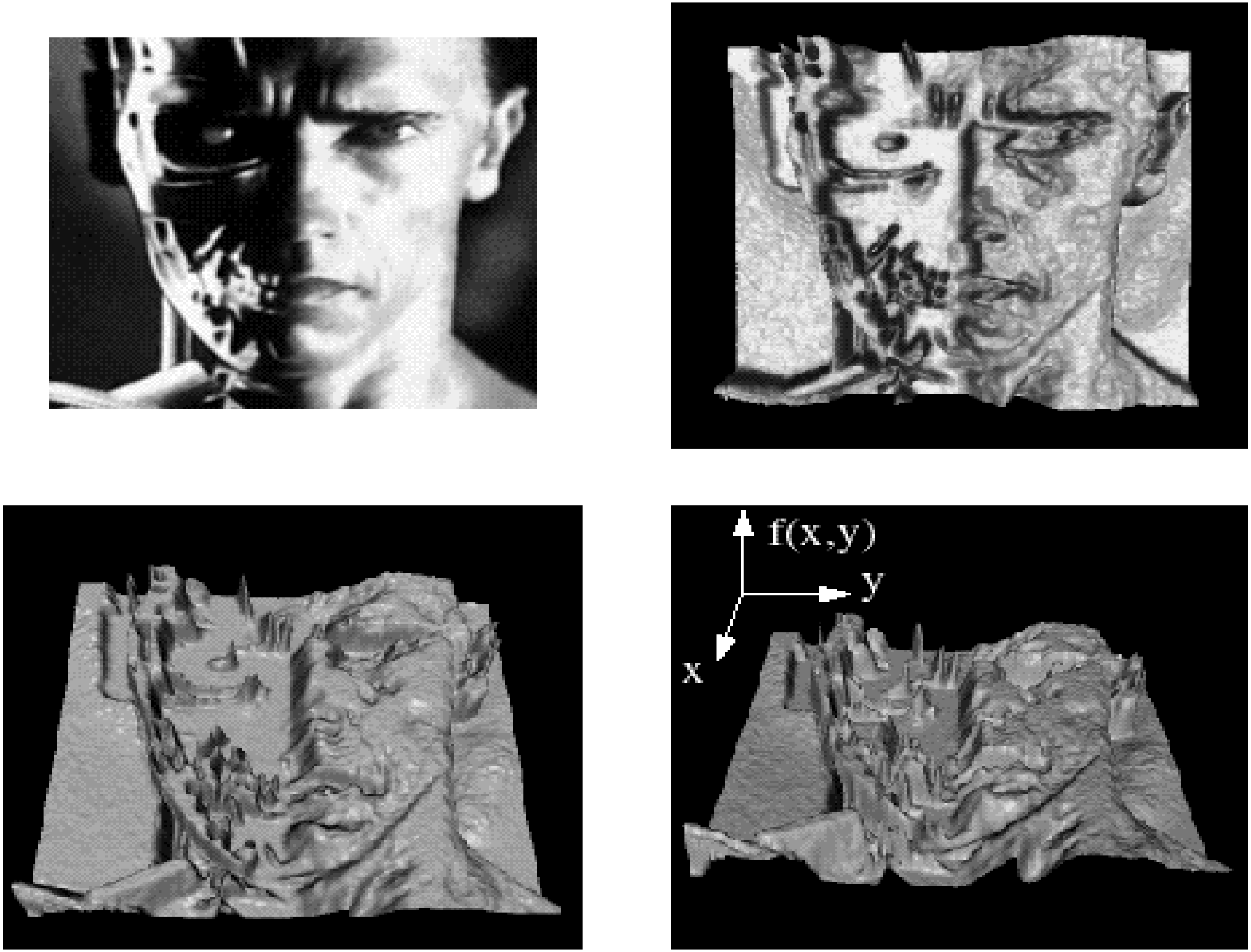
Filters

CS 554 – Computer Vision
Pinar Duygulu
Bilkent University

Today’s topics

- Image Formation
- Image filters in spatial domain
  - Filter is a mathematical operation of a grid of numbers
  - Smoothing, sharpening, measuring texture
- Image filters in the frequency domain
  - Filtering is a way to modify the frequencies of images
  - Denoising, sampling, image compression
- Templates and Image Pyramids
  - Filtering is a way to match a template to the image
  - Detection, coarse-to-fine registration

Images as functions



Source: S. Seitz

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:
    - $f: [a,b] \times [c,d] \rightarrow [0, 255]$

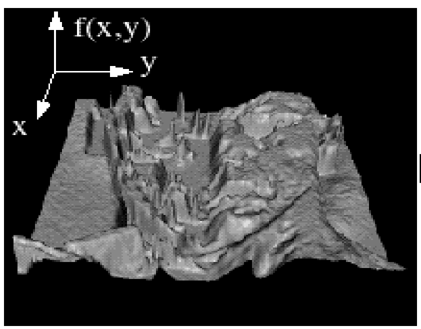
- A color image is just three functions pasted together. We can write this as a "vector-valued" function:

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

Source: S. Seitz

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.

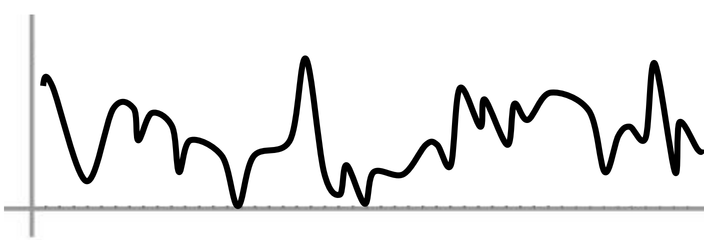


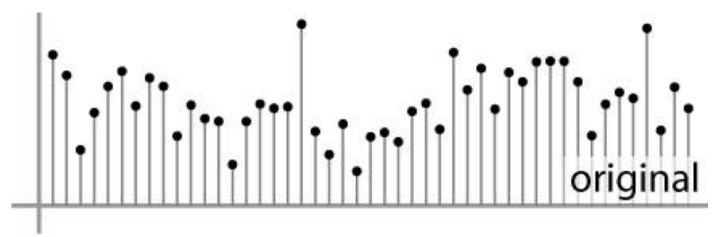
j

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

Adapted from S. Seitz

Images as discrete functions

- Cartesian Coordinates

$$f[n,m] = \begin{bmatrix} \ddots & & \vdots & & \\ & f[-1,1] & f[0,1] & f[1,1] & \\ \dots & f[-1,0] & \underline{f[0,0]} & f[1,0] & \dots \\ & f[-1,-1] & f[0,-1] & f[1,-1] & \\ & & \vdots & & \ddots \end{bmatrix}$$

Source: Fei Feli Li, Stanford University

Today's topics

- Image Formation
- Image filters in spatial domain
  - Filter is a mathematical operation of a grid of numbers
  - Smoothing, sharpening, measuring texture
- Image filters in the frequency domain
  - Filtering is a way to modify the frequencies of images
  - Denoising, sampling, image compression
- Templates and Image Pyramids
  - Filtering is a way to match a template to the image
  - Detection, coarse-to-fine registration

Zebras vs. Dalmatians

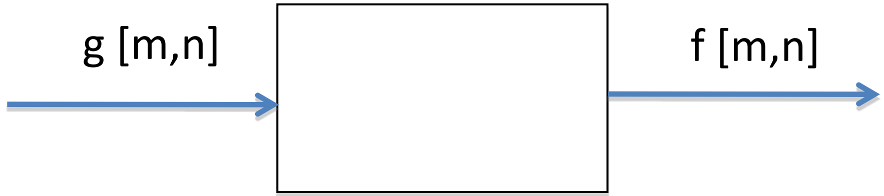
Both zebras and dalmatians have black and white pixels in about the same number

– if we shuffle the images point-wise processing is not affected

Need to measure properties relative to small *neighborhoods* of pixels

- find different image patterns

Filtering



We want to remove unwanted sources of variation, and keep the information relevant for whatever task we need to solve

Source: Torralba, MIT

Filters

- **Filtering:**

  – Form a new image whose pixels are a combination of original pixel values

  - compute function of local neighborhood at each position

- **Goals:**
- Extract useful information from the images

  Features (textures, edges, corners, distinctive points, blobs…)

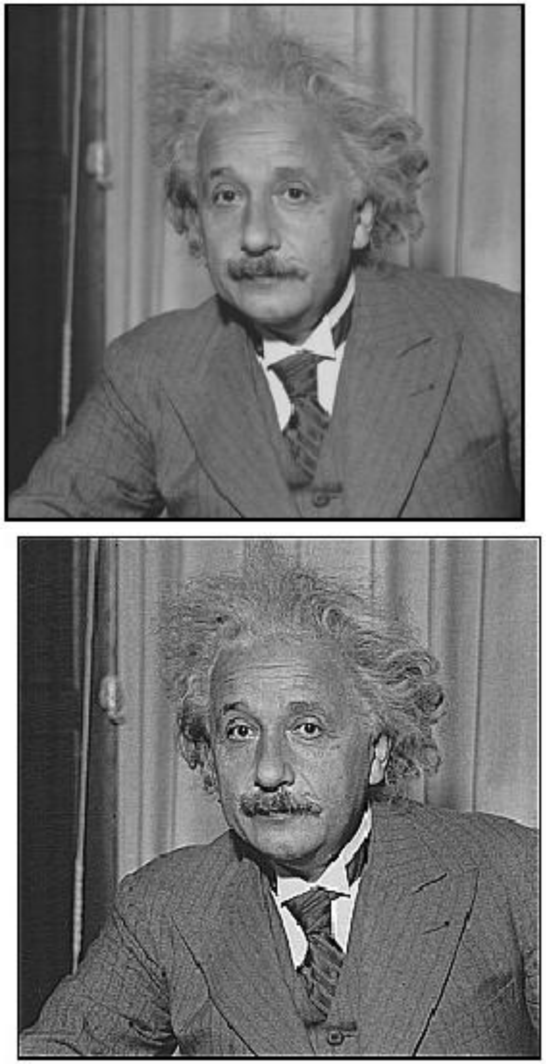
- Modify or enhance image properties:

  super-resolution; in-painting; de-noising, resizing

- Detect patterns

  Template matching

Source: Fei Feli Li, Stanford University; James Hays, Brown



Smooth/Sharpen Images...

Find edges...

Find waldo...

Source: Darrell, Berkeley

De-noising

Salt and pepper noise

Super-resolution

In-painting

Bertamio et al

Source: Fei Feli Li, Stanford University

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

Original

Salt and pepper noise

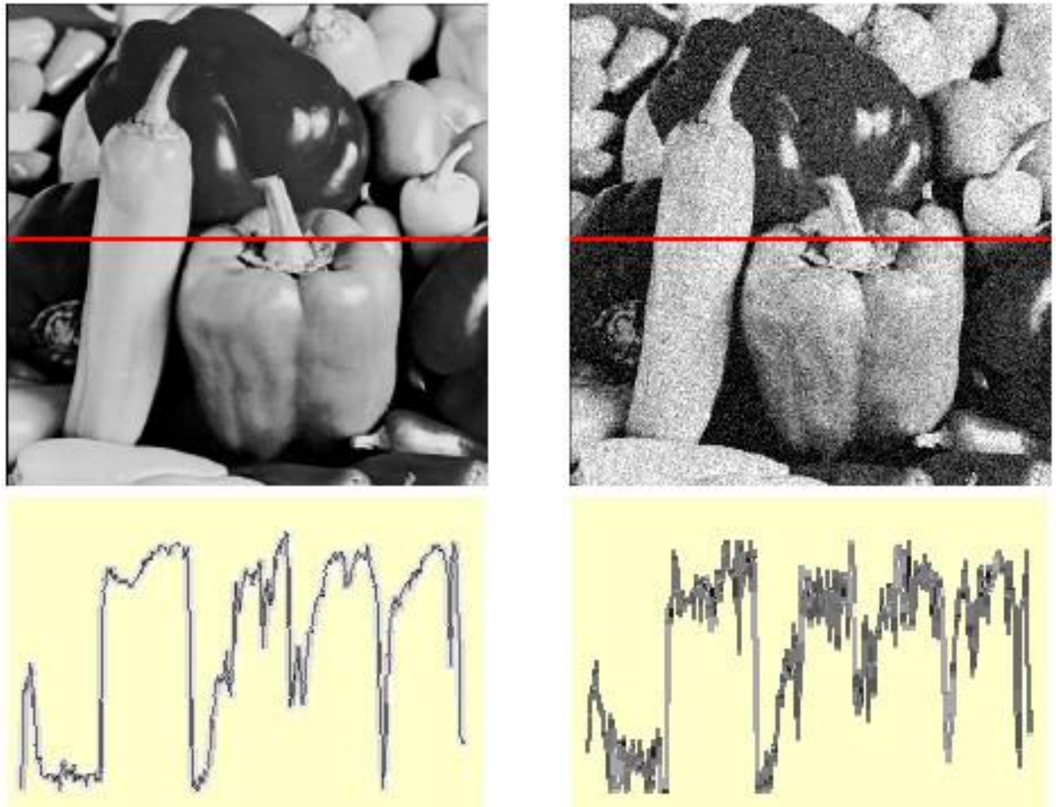
Impulse noise

Gaussian noise

Source: Darrell, Berkeley

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;
```

Source: Darrell, Berkeley

Fig: M. Hebert

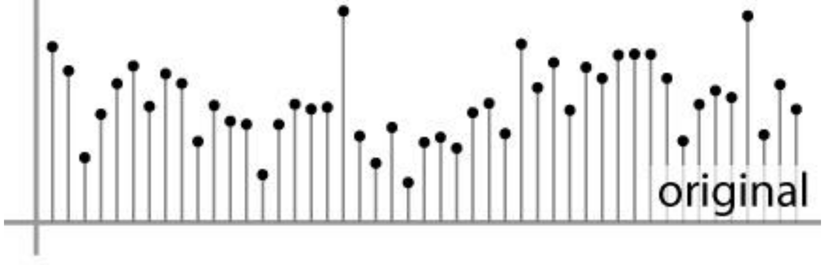
First attempt at a solution

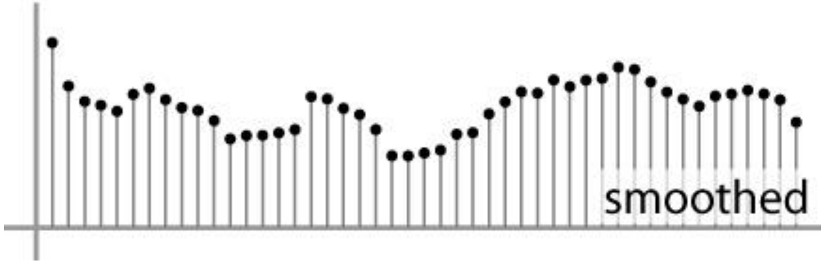
- Let’s replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors
  - Expect noise processes to be independent from pixel to pixel

Source: Darrell, Berkeley

First attempt at a solution

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



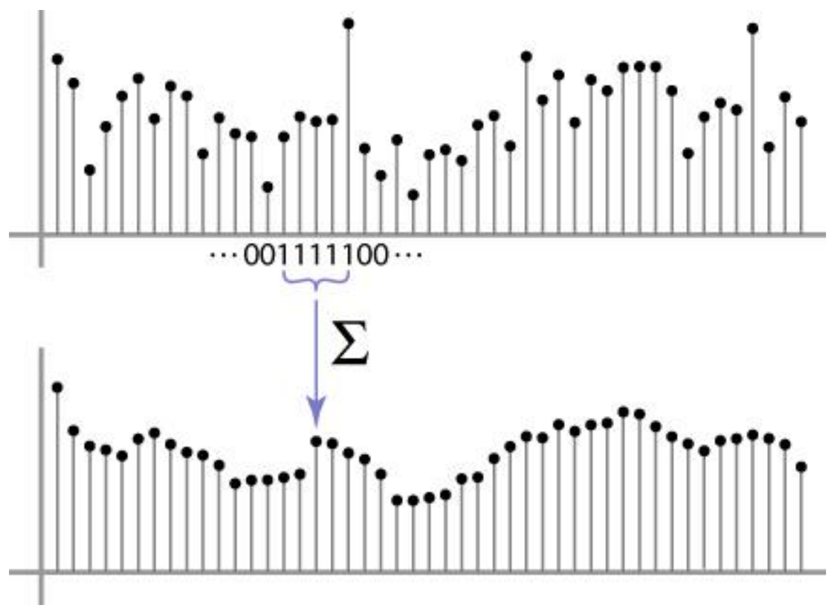


Source: Darrell, Berkeley

Source: S. Marschner

Weighted Moving Average

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

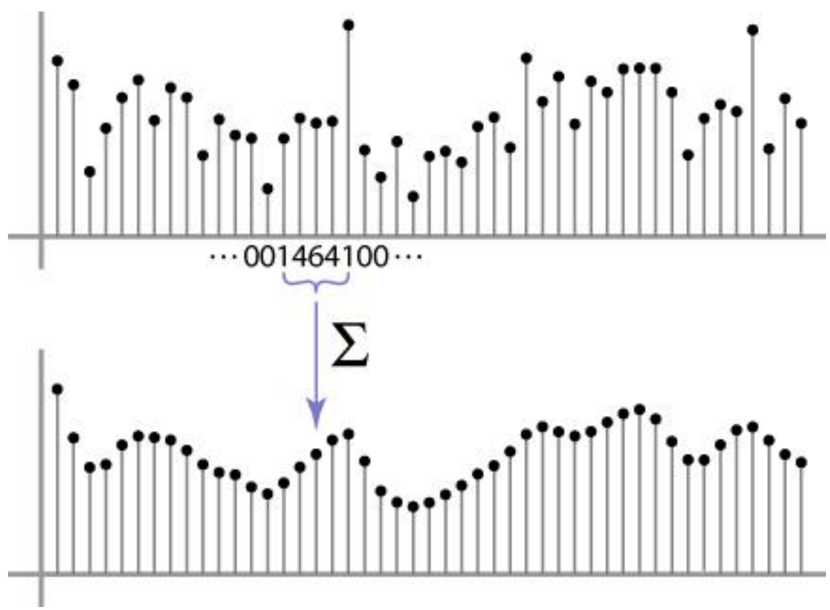


Source: Darrell, Berkeley

Source: S. Marschner

Weighted Moving Average

- Non-uniform weights [1, 4, 6, 4, 1] / 16



Source: Darrell, Berkeley

Source: S. Marschner

Moving Average In 2D

$F[x, y]$

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								

Source: Darrell, Berkeley

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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							

Source: Darrell, Berkeley

Source: S. Seitz

Moving Average In 2D

$$F[x, y]$$

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						

Source: Darrell, Berkeley

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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: Darrell, Berkeley

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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: Darrell, Berkeley

Source: S. Seitz

Moving Average In 2D

$F[x,y]$

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: Darrell, Berkeley

Source: S. Seitz

Correlation filtering

Say the averaging window size is 2k+1 x 2k+1:

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

Now generalize to allow different weights depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\textit{Non-uniform weights}} F[i+u, j+v]$$

Source: Darrell, Berkeley

Correlation filtering

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

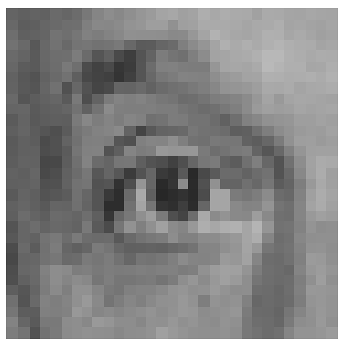
This is called cross-correlation, denoted $G = H \otimes F$

Filtering an image: replace each pixel with a linear combination of its neighbors.

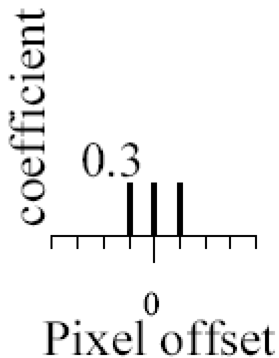
The filter "kernel" or "mask" $H[u,v]$ is the prescription for the weights in the linear combination.

Source: Darrell, Berkeley

Averaging Filter



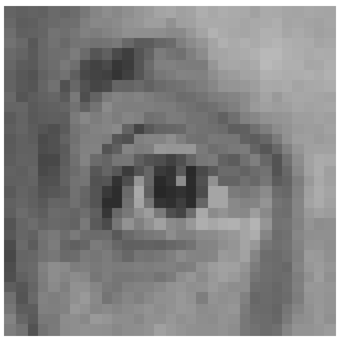
original



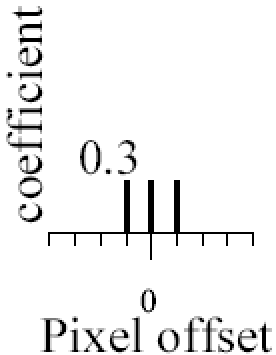
?

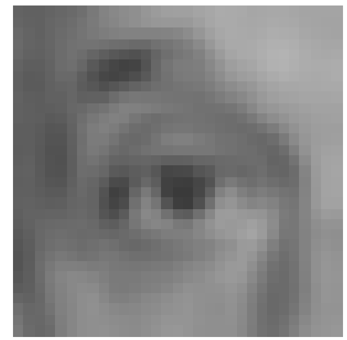
adapted from Darrell and Freeman, MIT

Averaging Filter



original

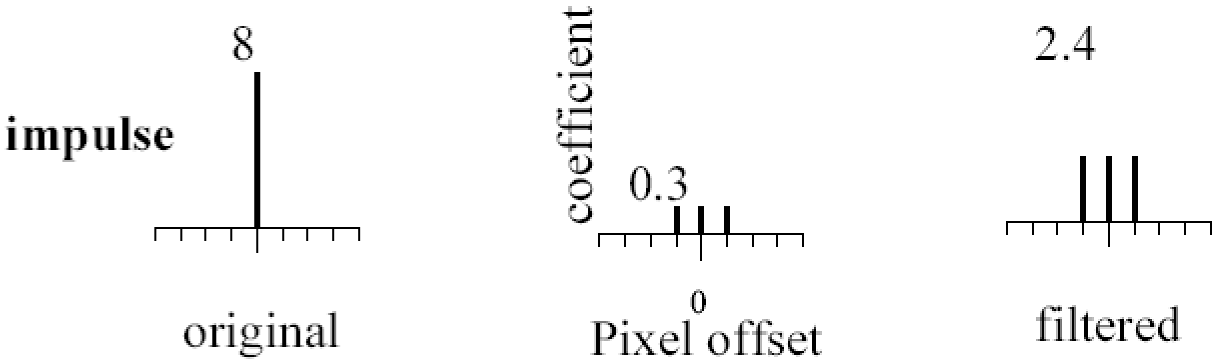




Blurred (filter applied in both dimensions).

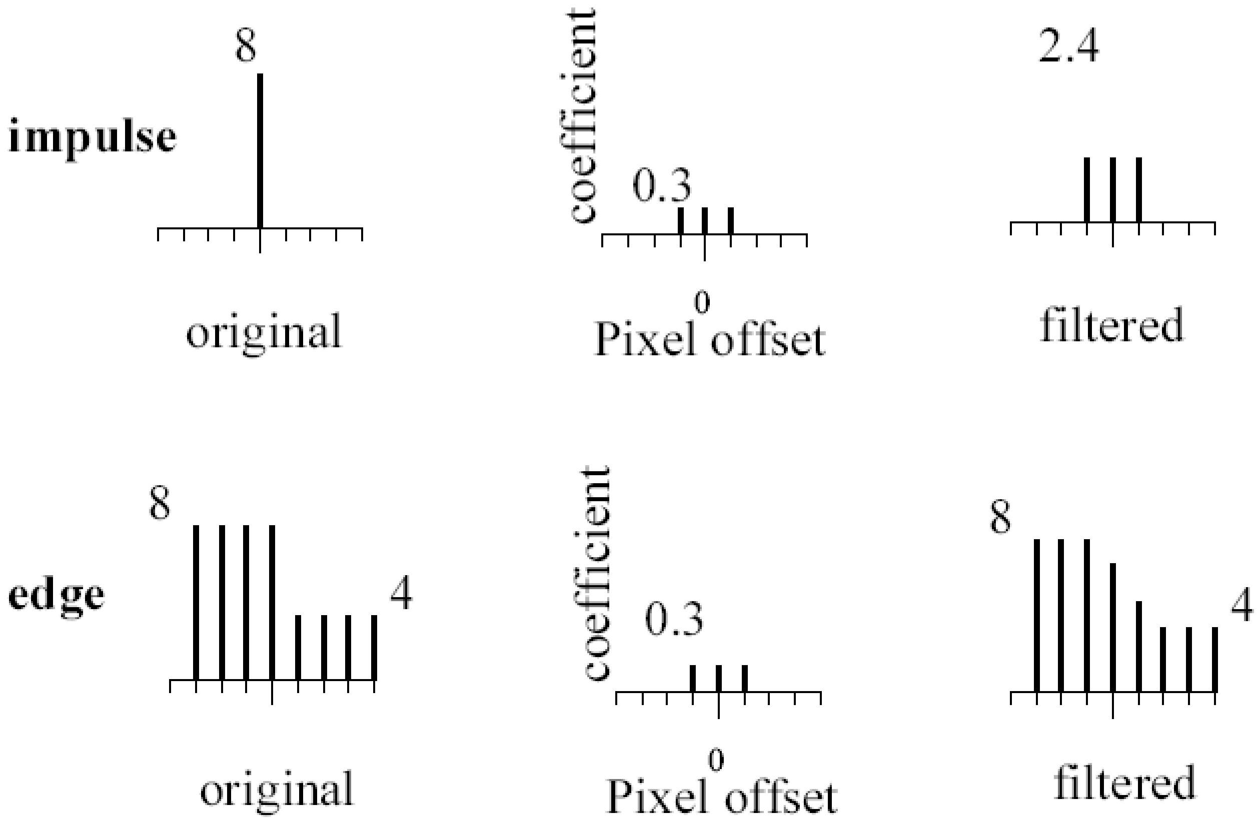
adapted from Darrell and Freeman, MIT

Averaging Filter



adapted from Darrell and Freeman, MIT

Averaging Filter



adapted from Darrell and Freeman, MIT

Averaging filter

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

$F[x, y]$

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

$\otimes$

$H[u, v]$

$\frac{1}{9}$

1	1	1
1	?	1
1	1	1

"box filter"

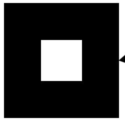
$G[x, y]$

	0	10	20	30	30				

$$G = H \otimes F$$

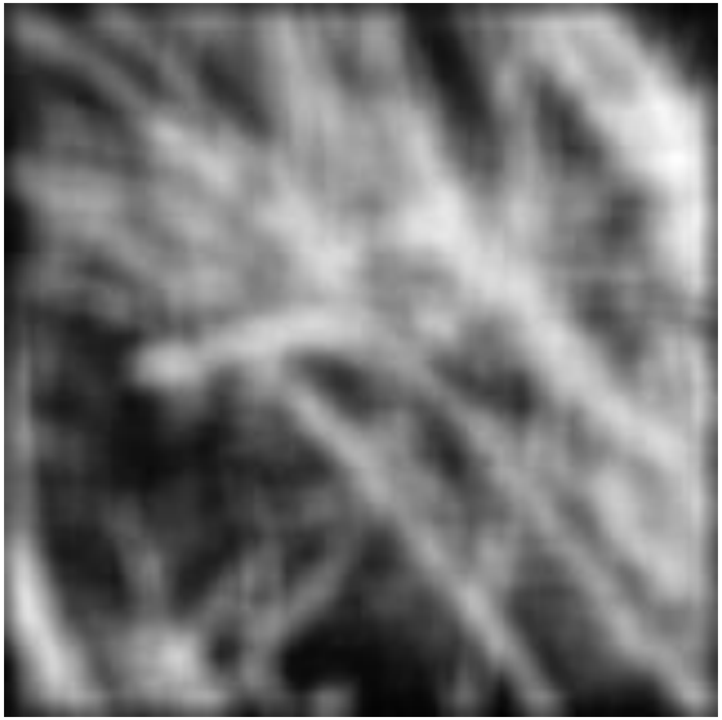
Source: Darrell, Berkeley

Smoothing by averaging



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

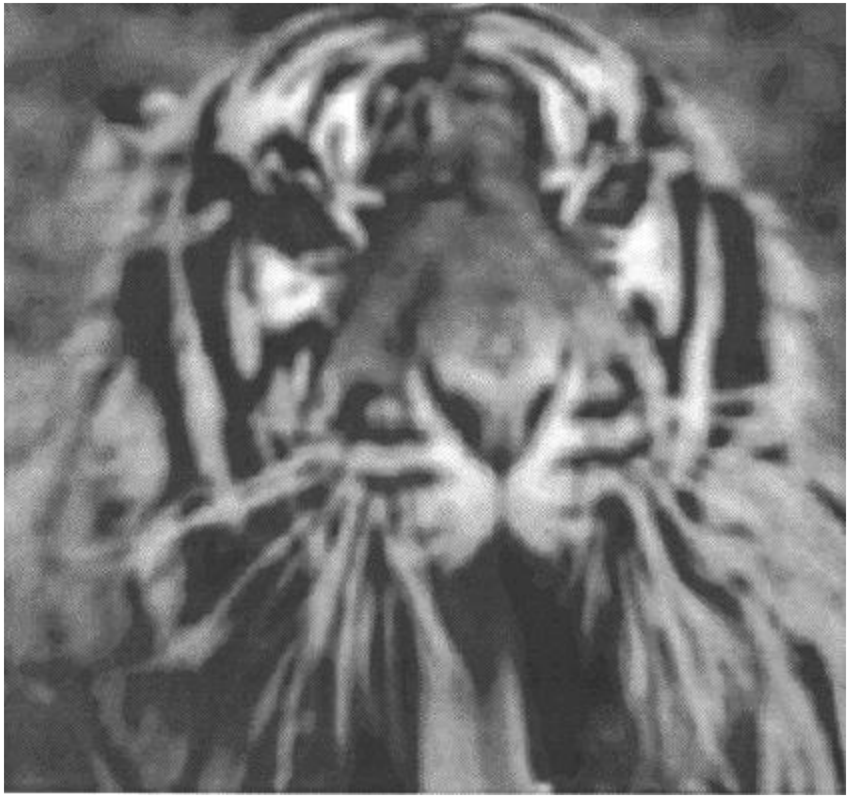
original



filtered

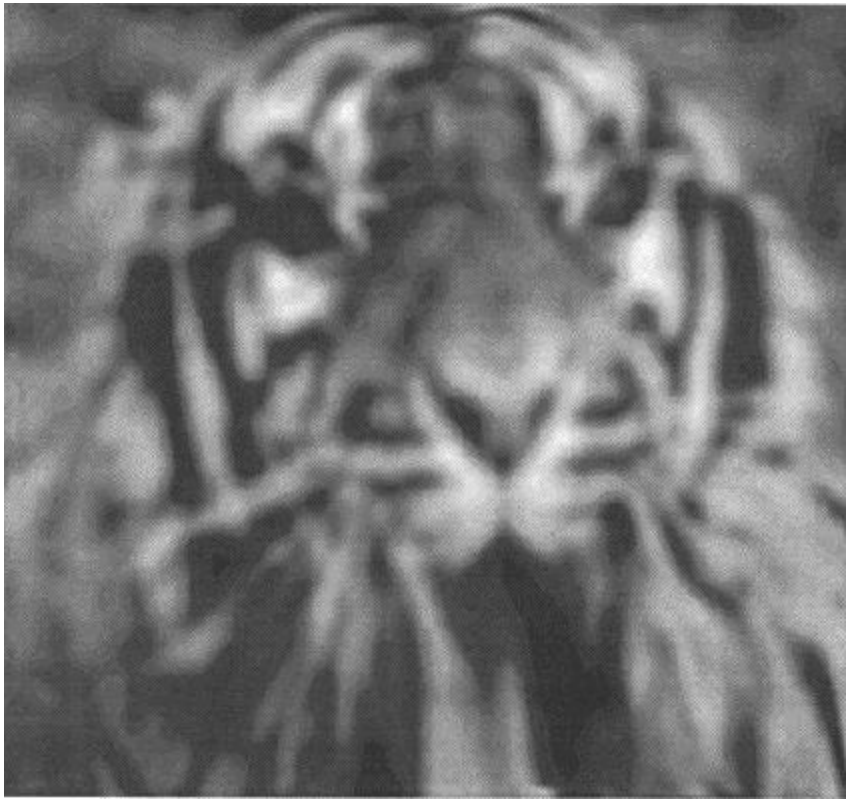
Source: Darrell, Berkeley

Example



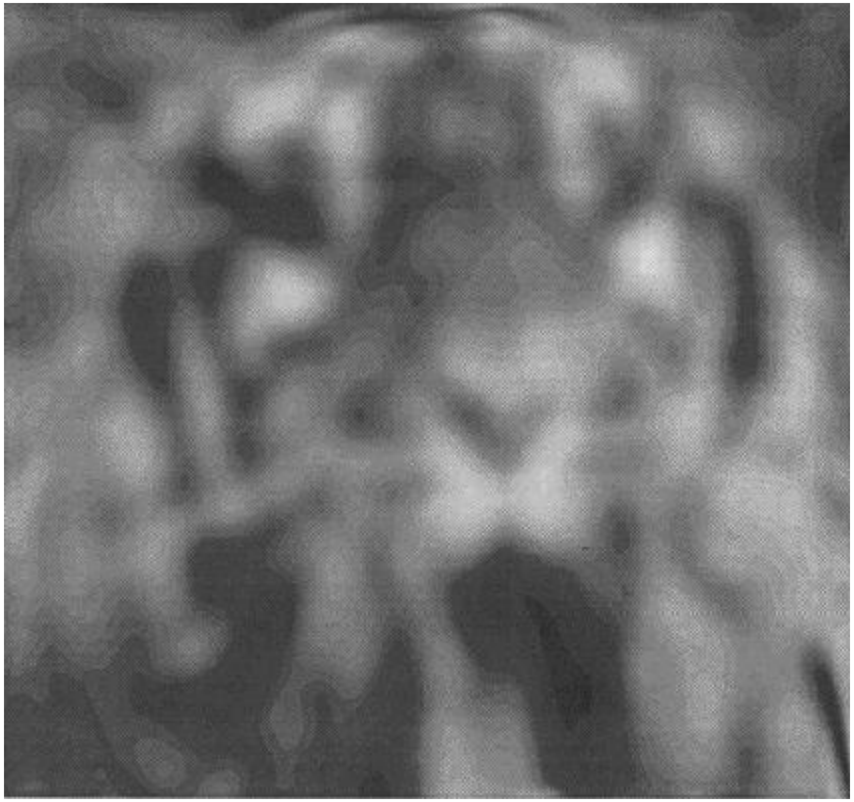
Source: Martial Hebert, CMU

Example



Source: Martial Hebert, CMU

Example

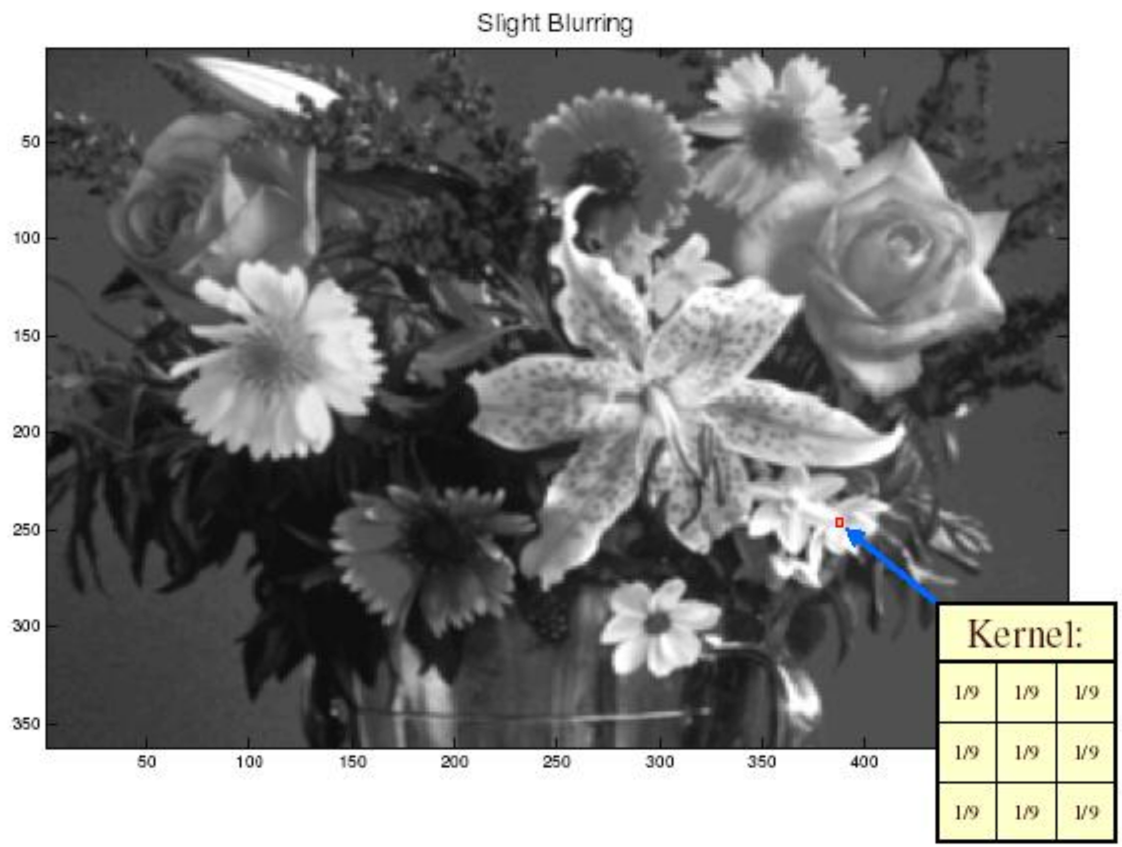


Source: Martial Hebert, CMU

Smoothing by averaging

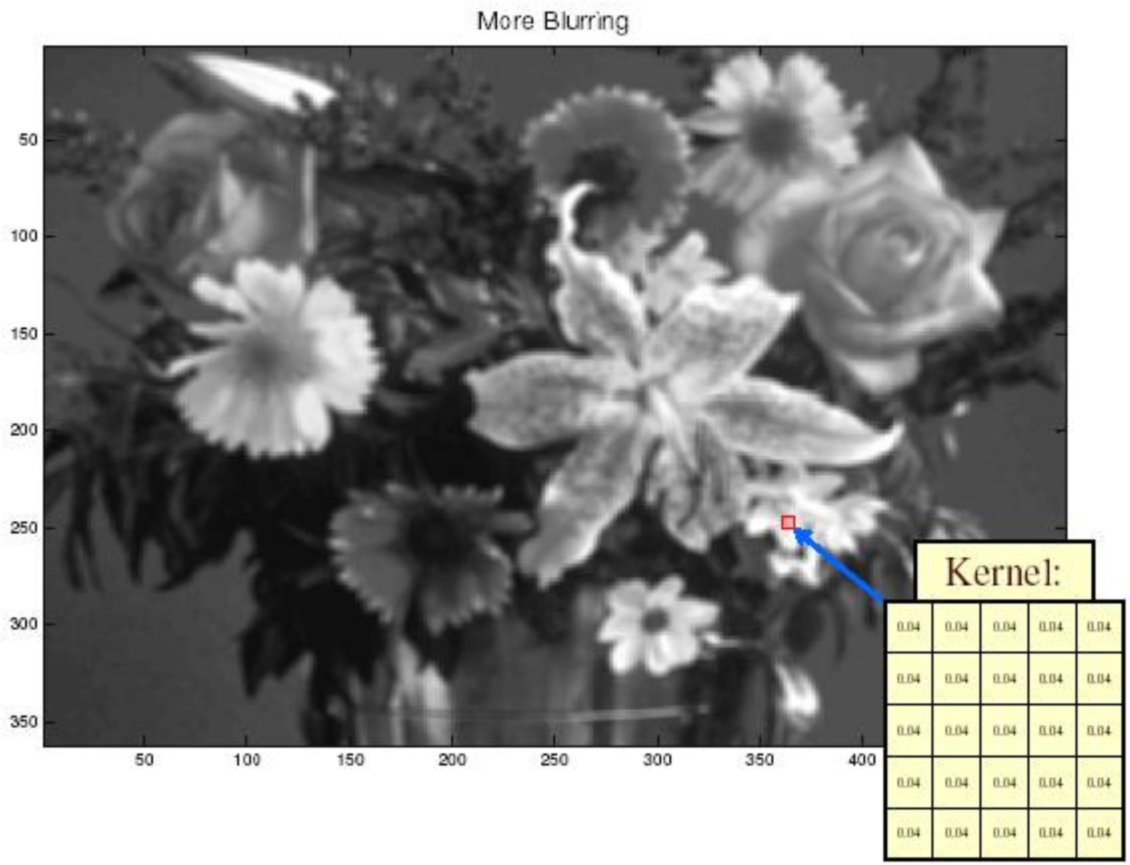
Source: Martial Hebert, CMU

Smoothing by averaging



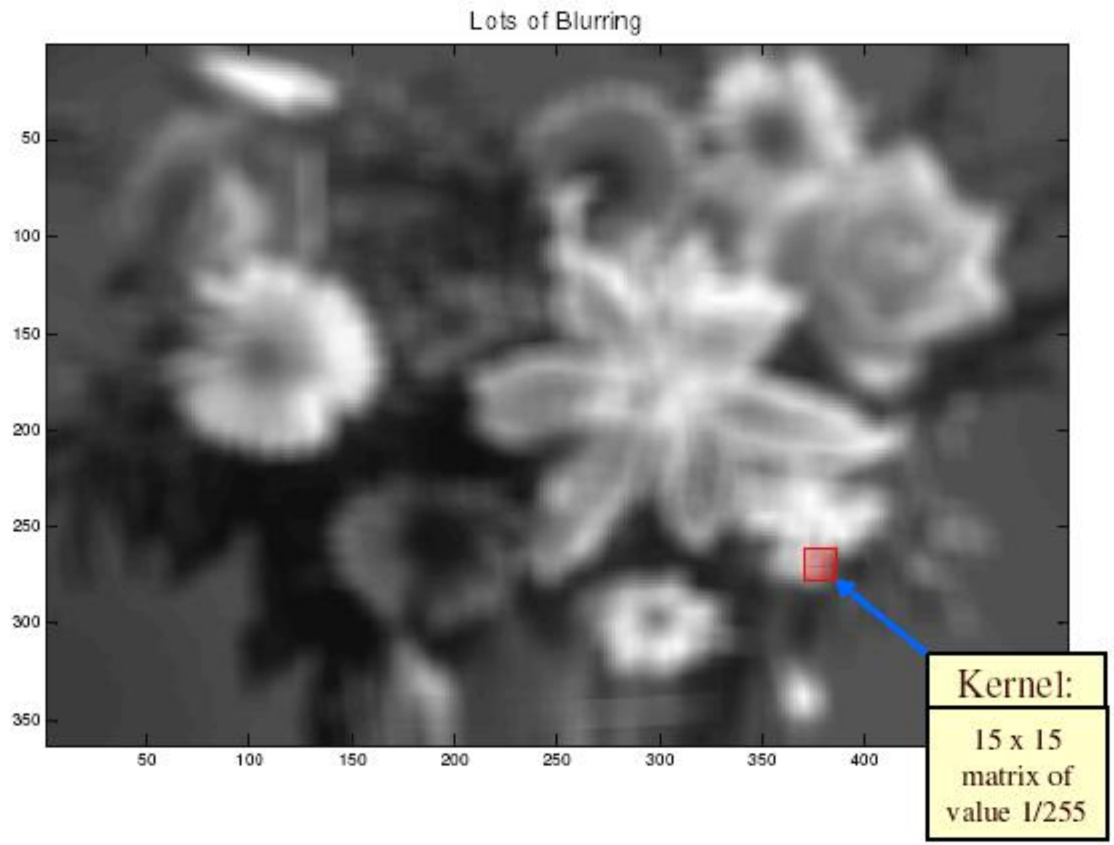
Source: Martial Hebert, CMU

Smoothing by averaging



Source: Martial Hebert, CMU

Smoothing by averaging



Source: Martial Hebert, CMU

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

$F[x, y]$

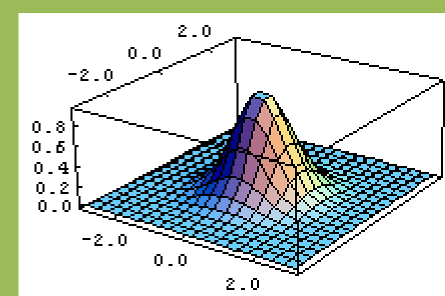
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

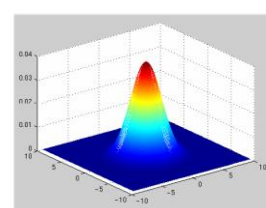
$H[u, v]$

This kernel is an approximation of a Gaussian function:

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



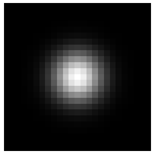
A weighted average that weights pixels at its center much more strongly than its boundaries

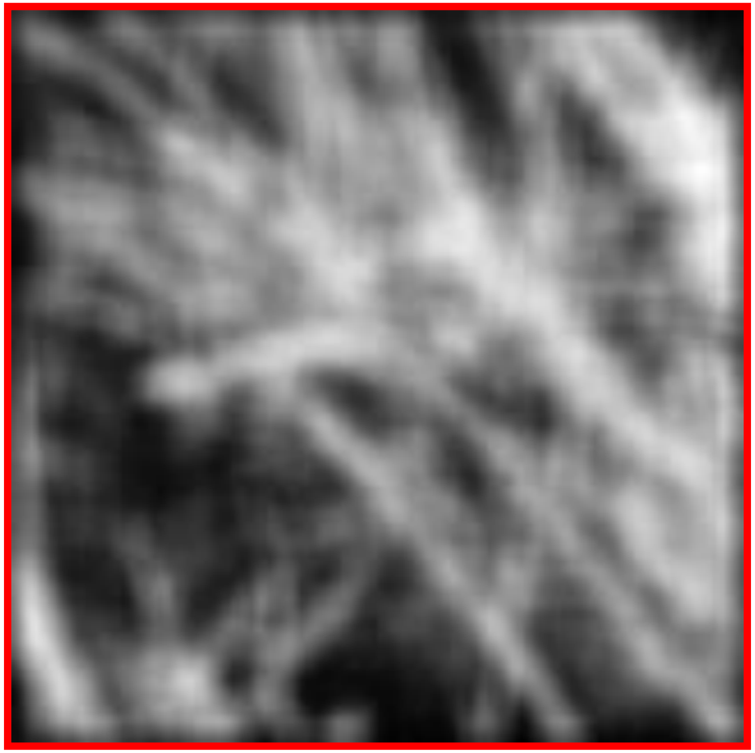


Source: Darrell, Berkeley

Source: S. Seitz

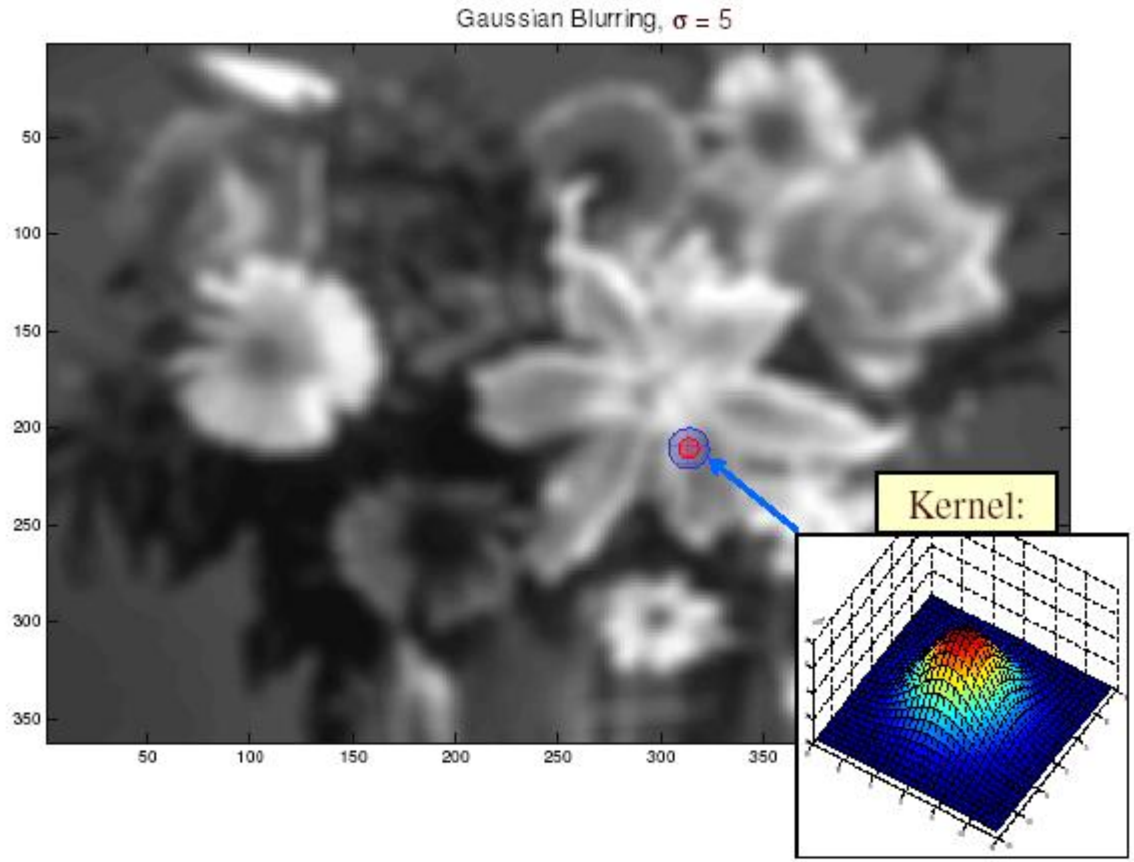
Smoothing with a Gaussian





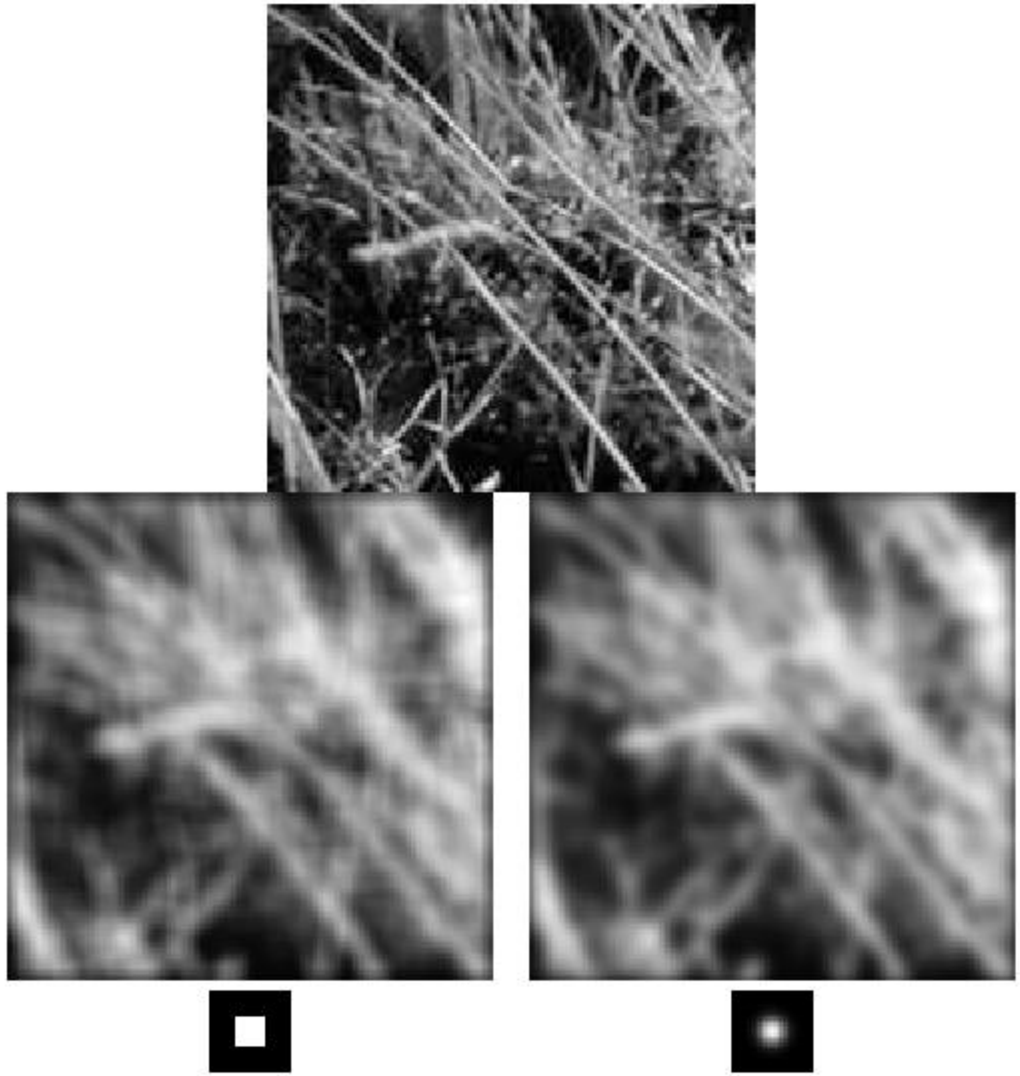
Source: Darrell, Berkeley

Smoothing with a Gaussian



Source: Martial Hebert, CMU

Smoothing with a Gaussian



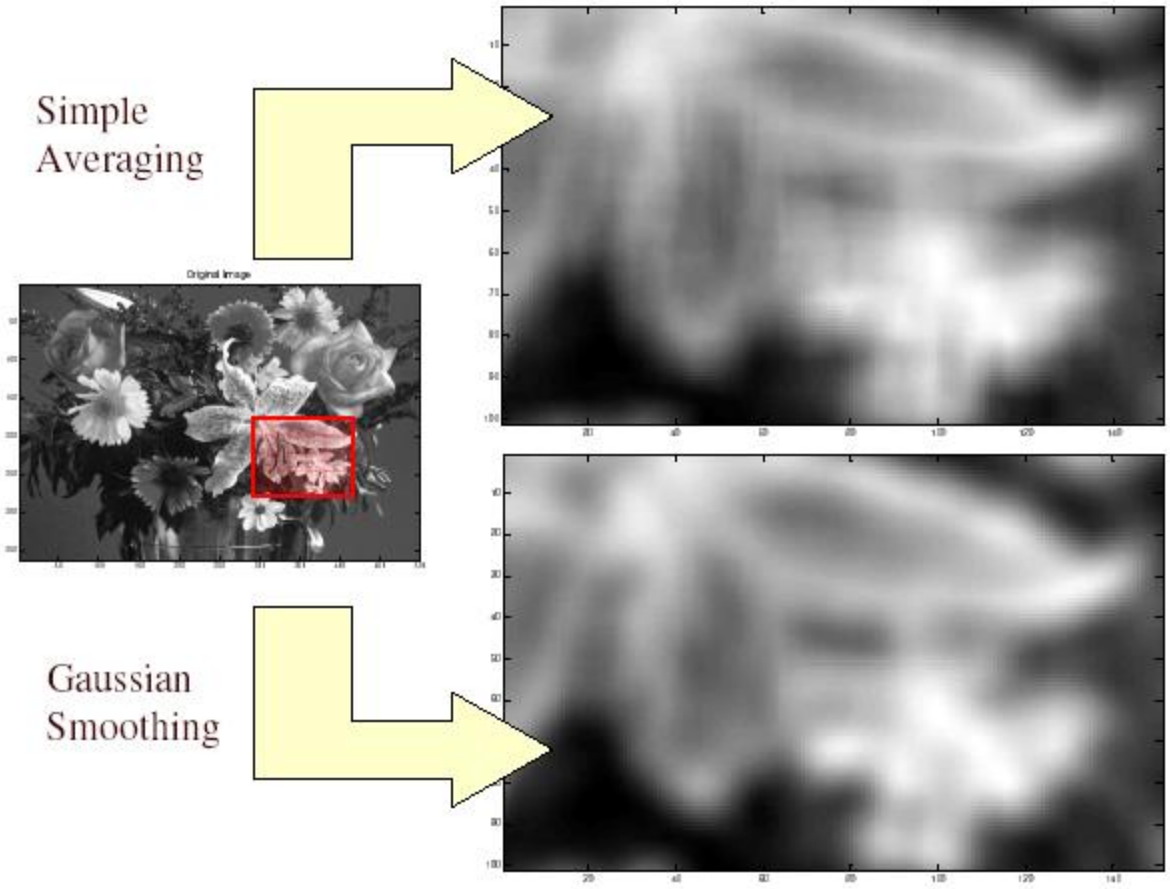
Result of blurring using a uniform local model

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

Result of blurring using a set of Gaussian weights

Source: David Forsyth, UIUC

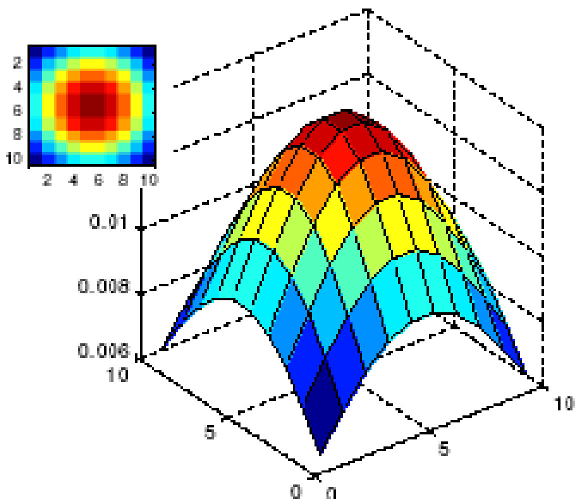
Smoothing with a Gaussian



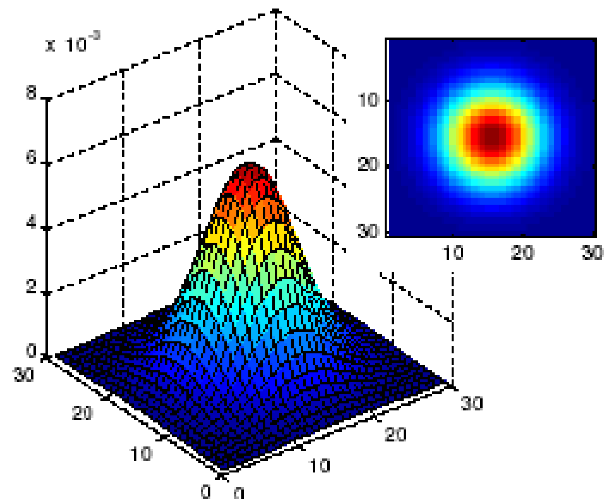
Source: Martial Hebert, CMU

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 with 10 x 10 kernel

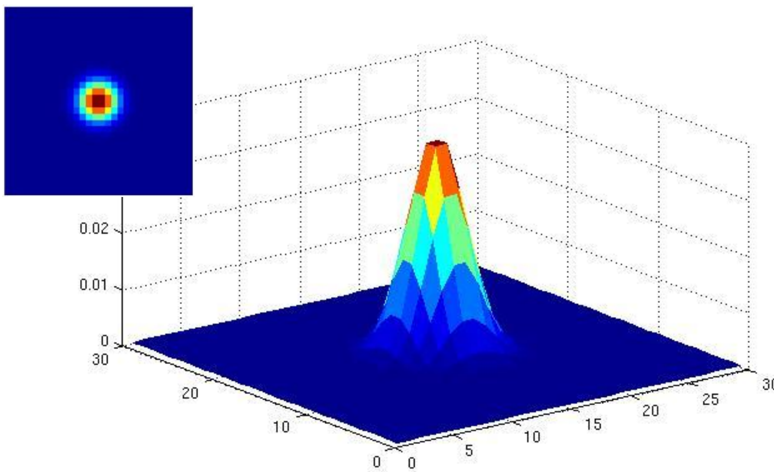


σ = 5 with 30 x 30 kernel

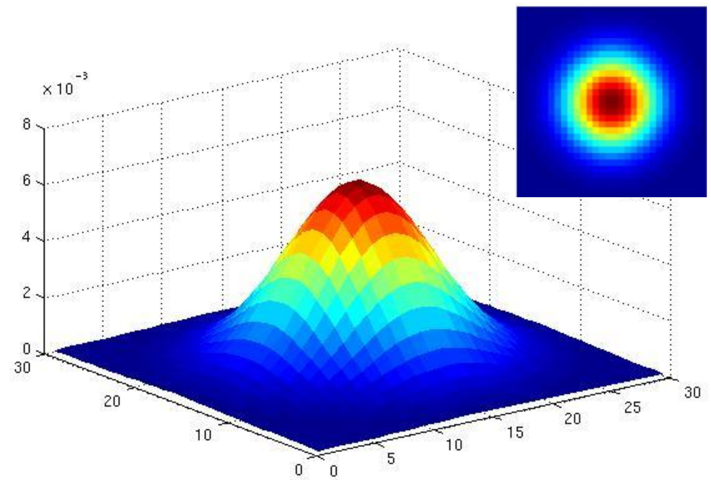
Source: Darrell, Berkeley

Gaussian filters

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



σ = 2 with 30 x 30 kernel



σ = 5 with 30 x 30 kernel

Source: Darrell, Berkeley

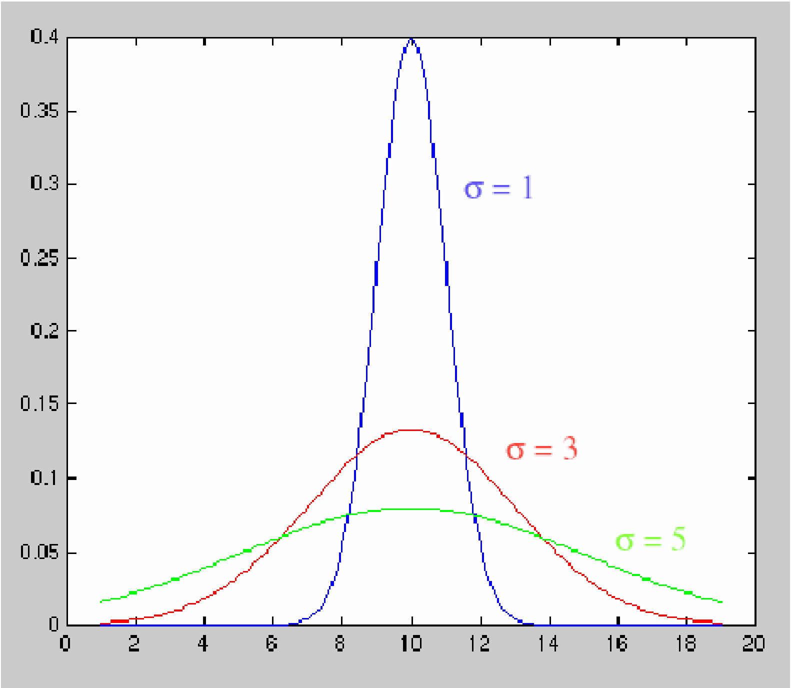
Smoothing with a Gaussian

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

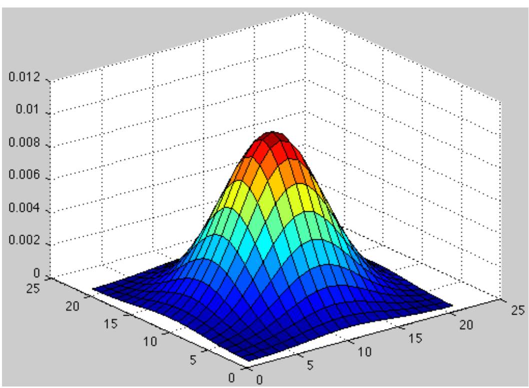
Effect of σ

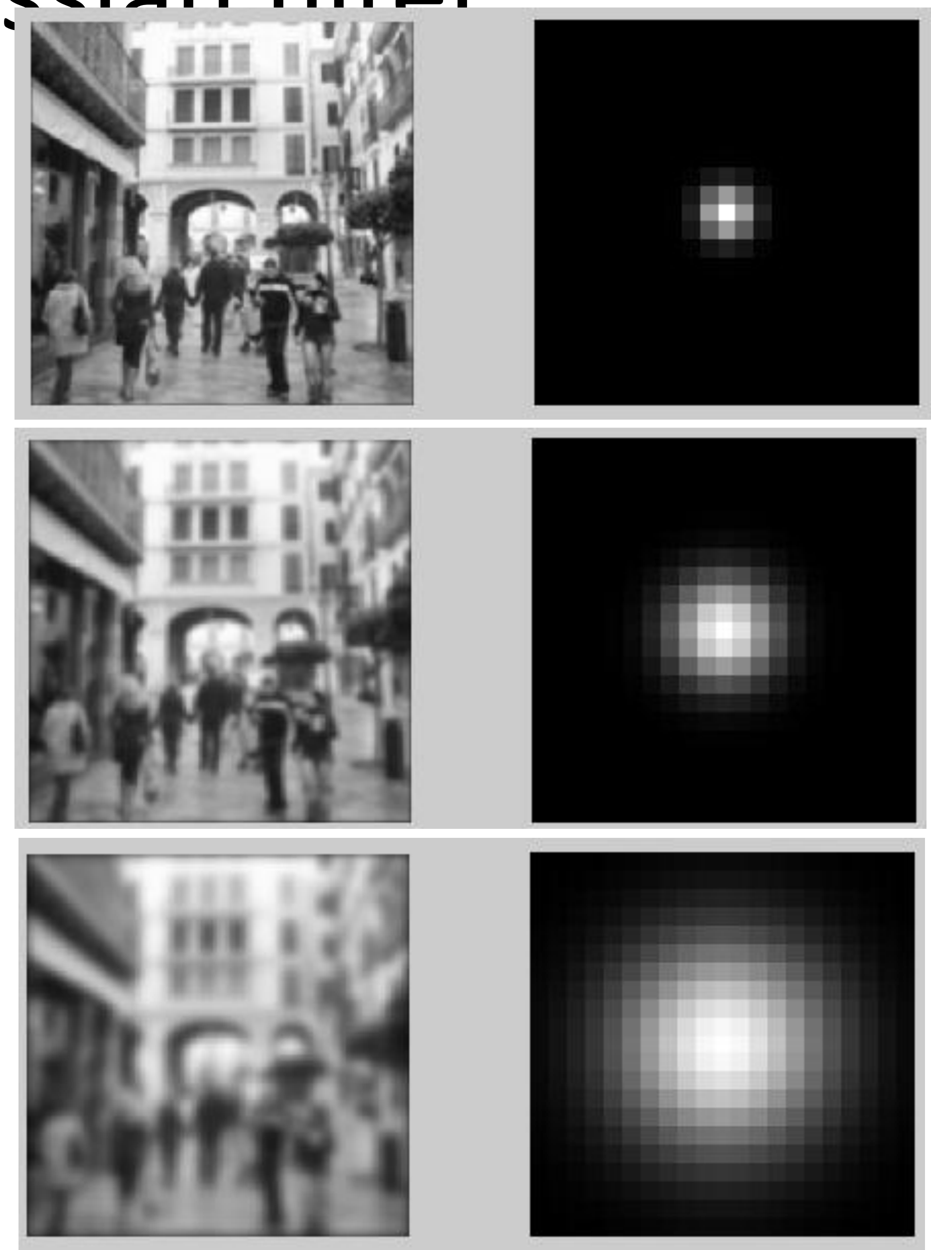


Source: Martial Hebert, CMU

Gaussian filter

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





σ=1

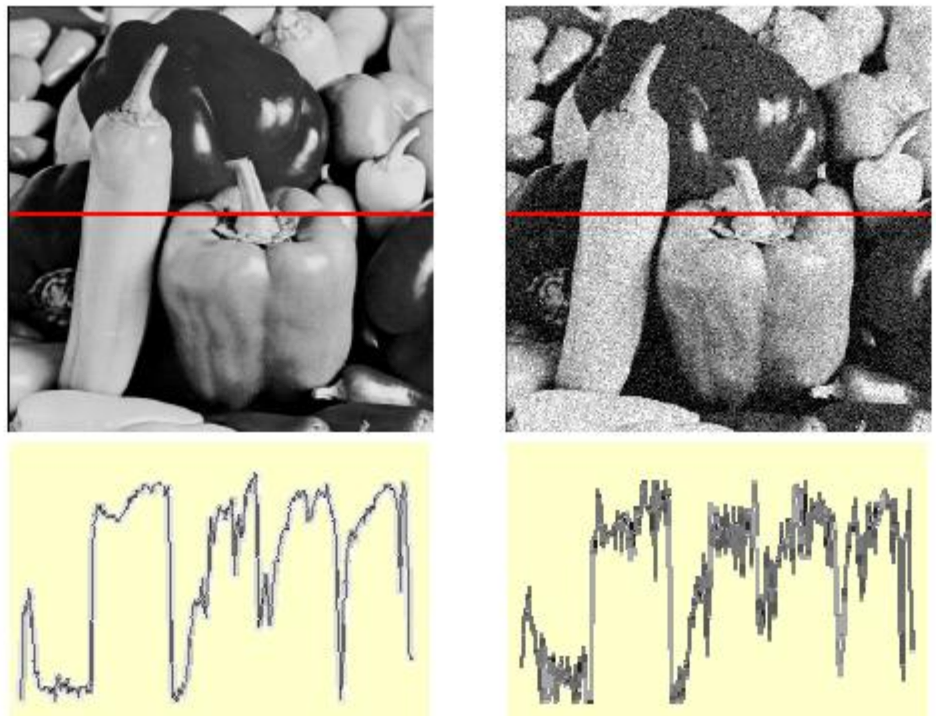
σ=2

σ=4

Source: Torralba, MIT

Gaussian smoothing to remove noise

Image Noise

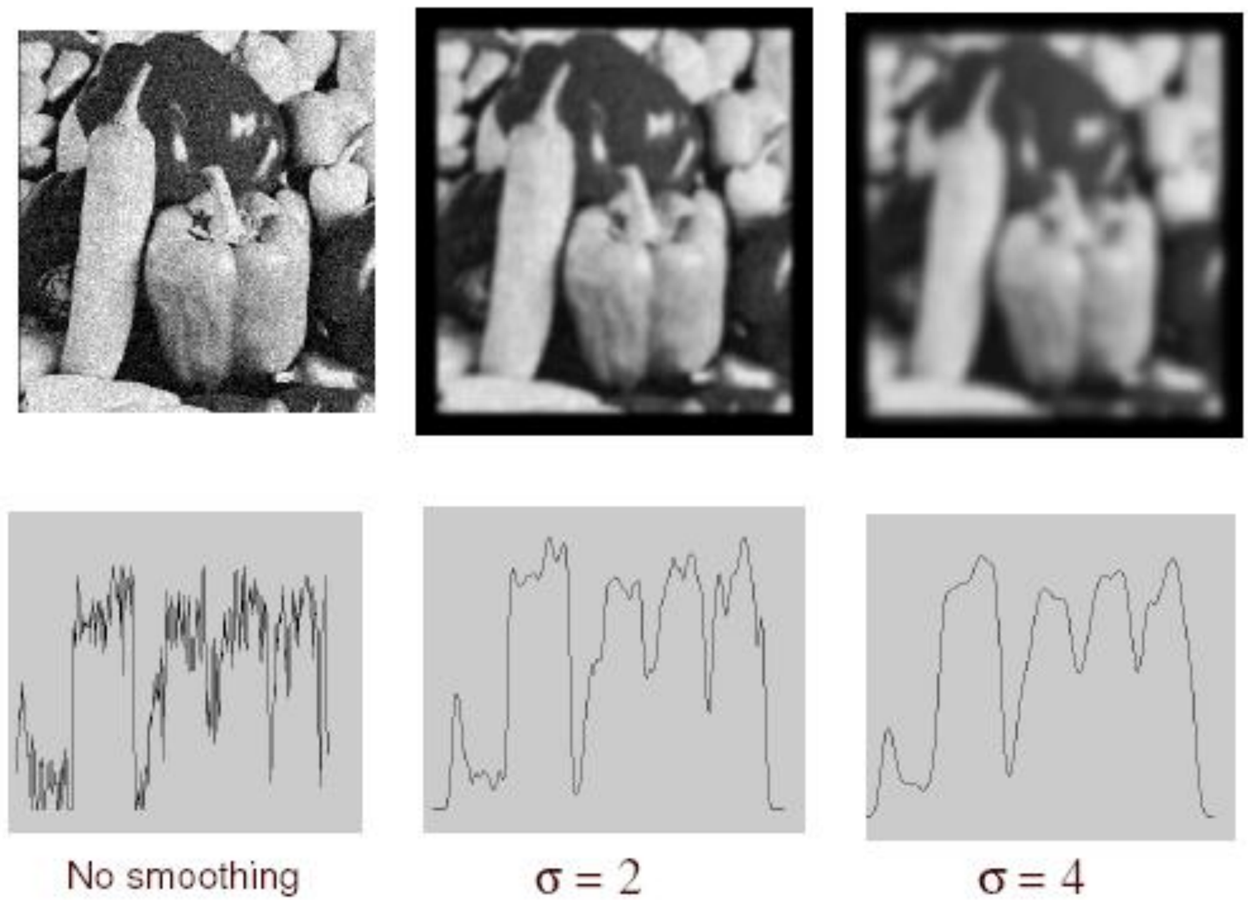


$$f(x,y) = \overbrace{\tilde{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

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

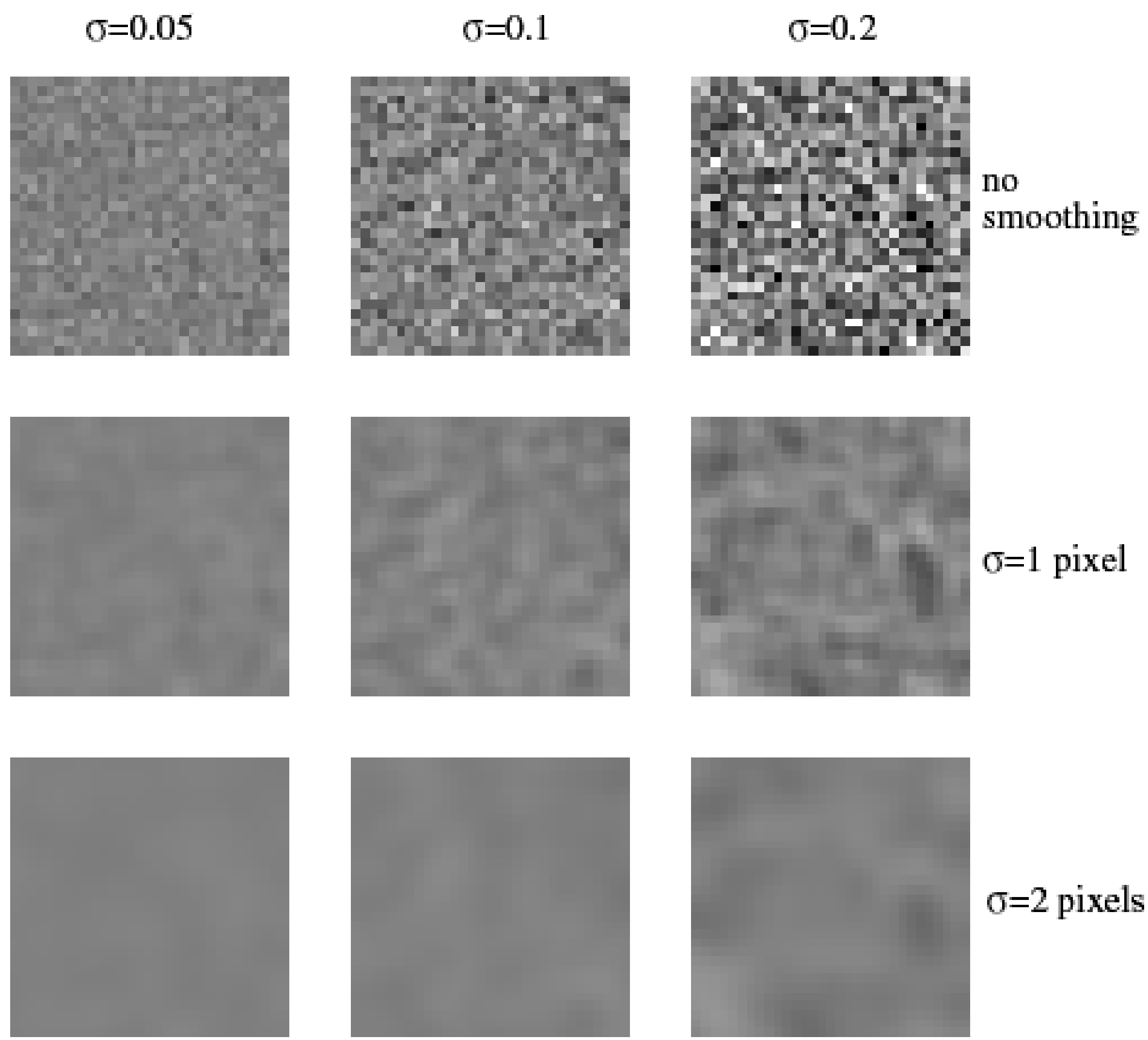
Source: Martial Hebert, CMU

Gaussian smoothing to remove noise



Source: Martial Hebert, CMU

Smoothing with a Gaussian

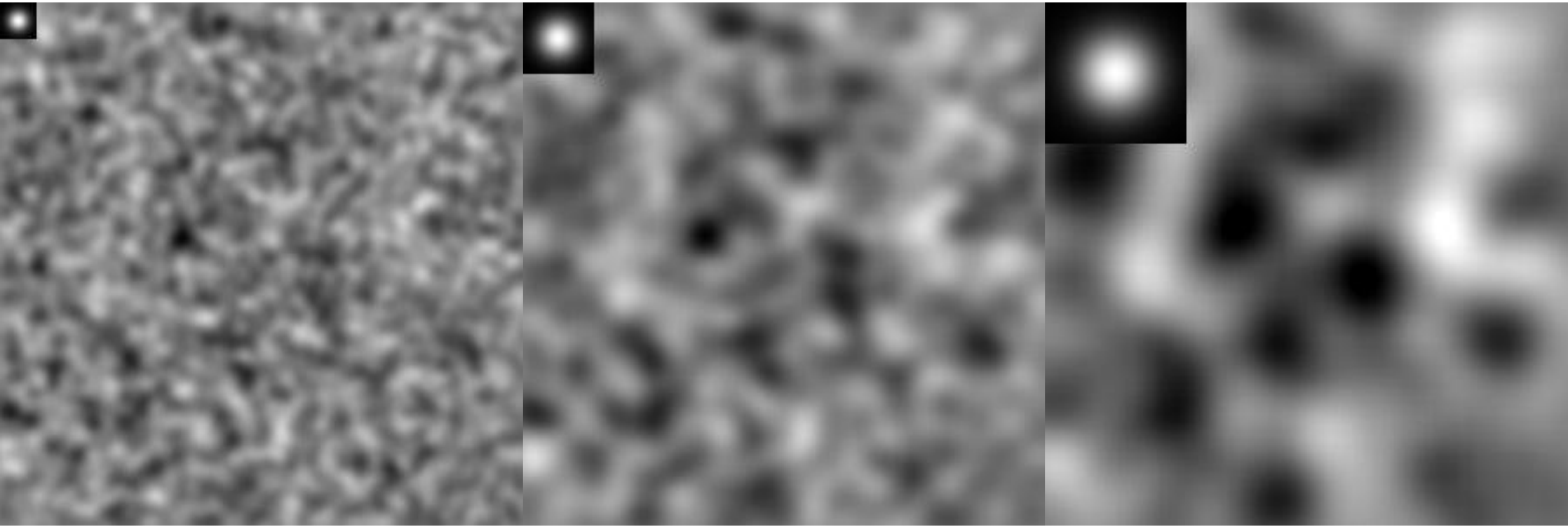


The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

Source: David Forsyth, UIUC

Smoothing with a Gaussian

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



Source: David Forsyth, UIUC

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 =

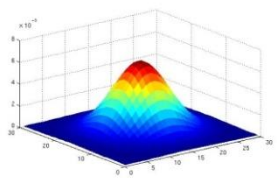
0.0625	0.1250	0.0625
0.1250	0.2500	0.1250
0.0625	0.1250	0.0625

Source: from Michael Black

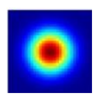
Matlab

```
>> hsize = 10;
>> sigma = 5;
>> h = fspecial(‘gaussian’ hsize, sigma);
```

```
>> mesh(h);
```



```
>> imagesc(h);
```



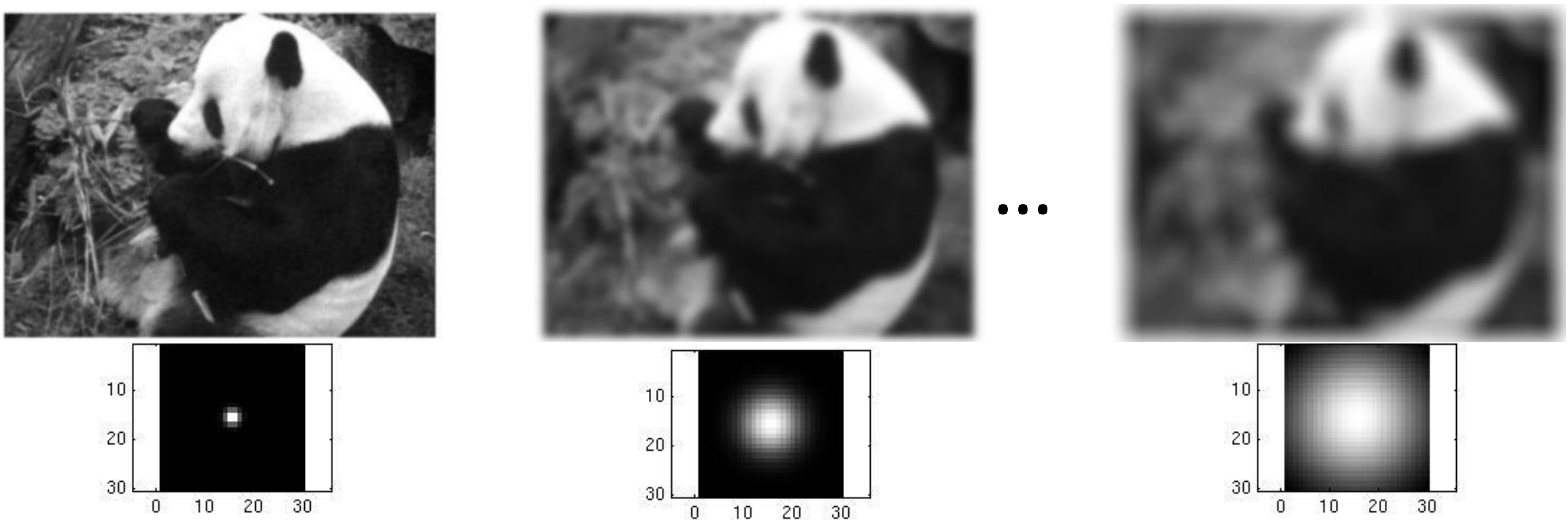
```
>> outim = imfilter(im, h);
>> imshow(outim);
```

outim

Source: Darrell, Berkeley

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
```

Source: Darrell, Berkeley

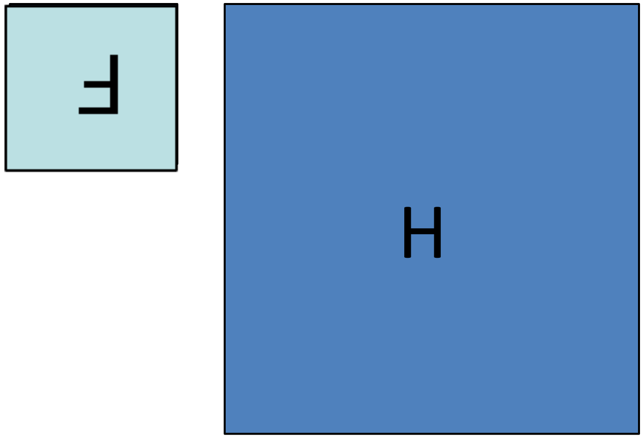
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

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

$$G = H \star F$$

Notation for convolution operator



Source: Darrell, Berkeley

Convolution vs. correlation

Convolution

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

$$G = H \star F$$

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?

Source: Darrell, Berkeley

Predict the filtered outputs

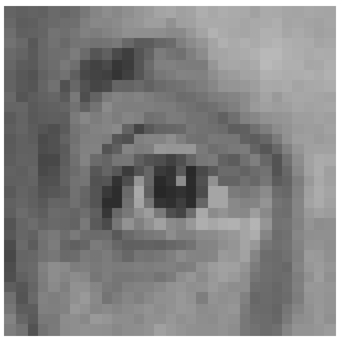
$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = ?$$

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

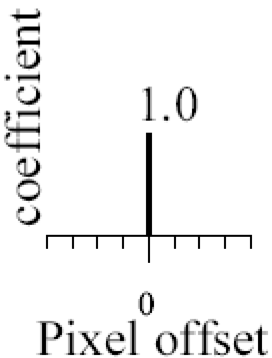
$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

Source: Darrell, Berkeley

Practice with linear filters



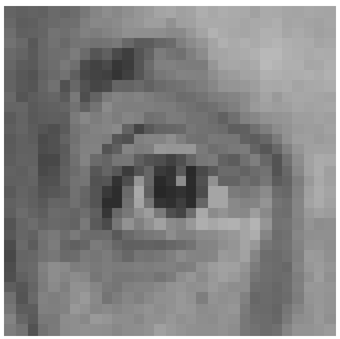
original



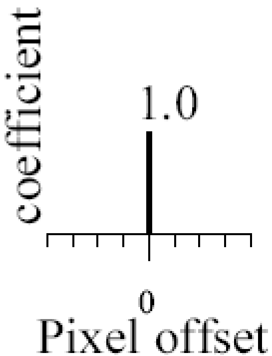
?

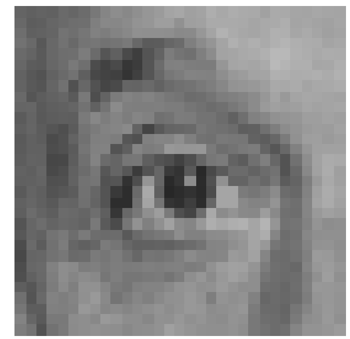
adapted from Darrell and Freeman, MIT

Practice with linear filters



original





Filtered
(no change)

adapted from Darrell and Freeman, MIT

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

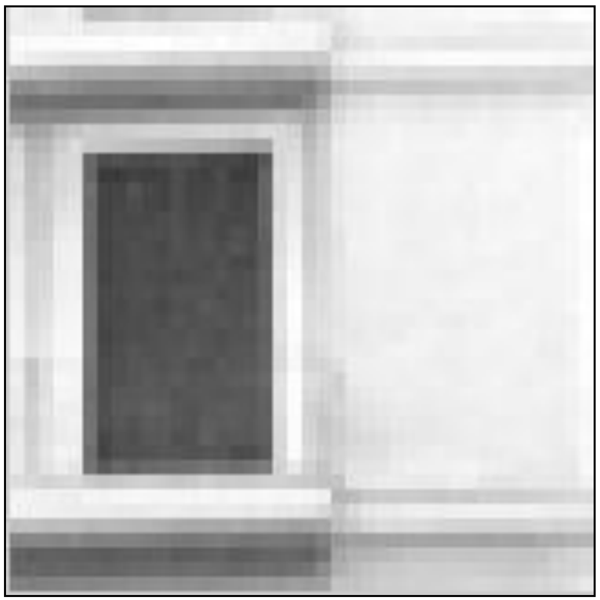
Filtered
(no change)

Source: Darrell, Berkeley

Source: D. Lowe

Impulse

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



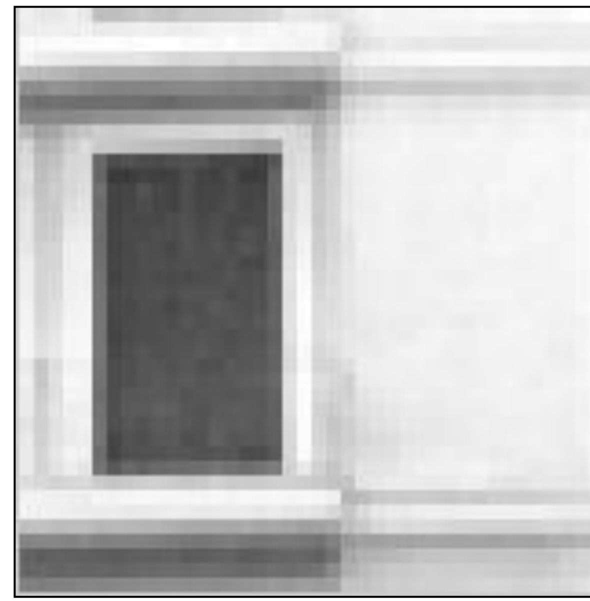
g[m,n]

$\otimes$

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

h[m,n]

=



f[m,n]

Source: Torralba, MIT

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

?

Source: Darrell, Berkeley

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

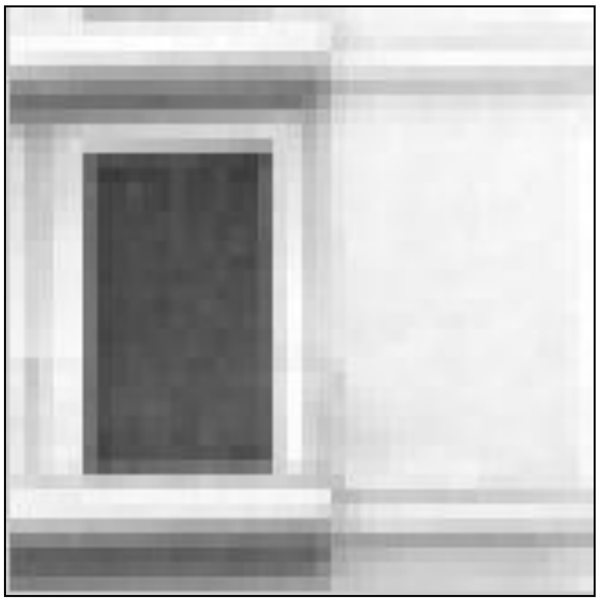
Shifted left by 1 pixel with correlation

Source: Darrell, Berkeley

Source: D. Lowe

Shifts

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



g[m,n]

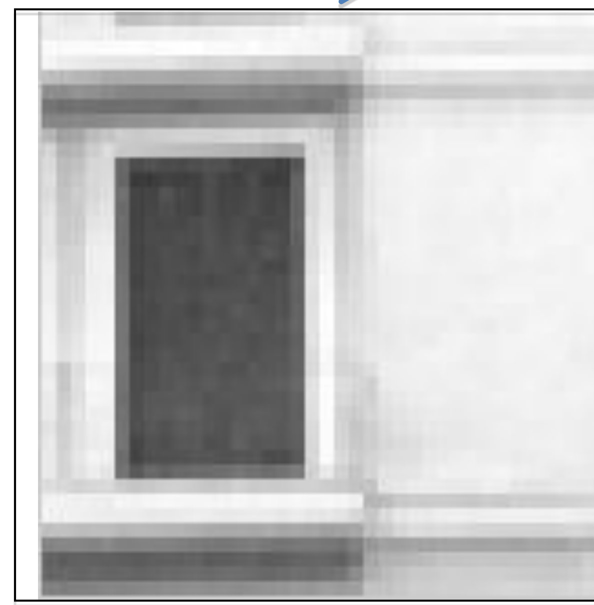
$\otimes$

0	0	0	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0

h[m,n]

=

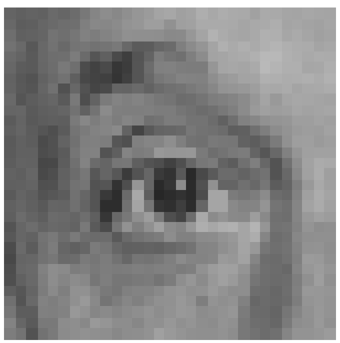
2pixels



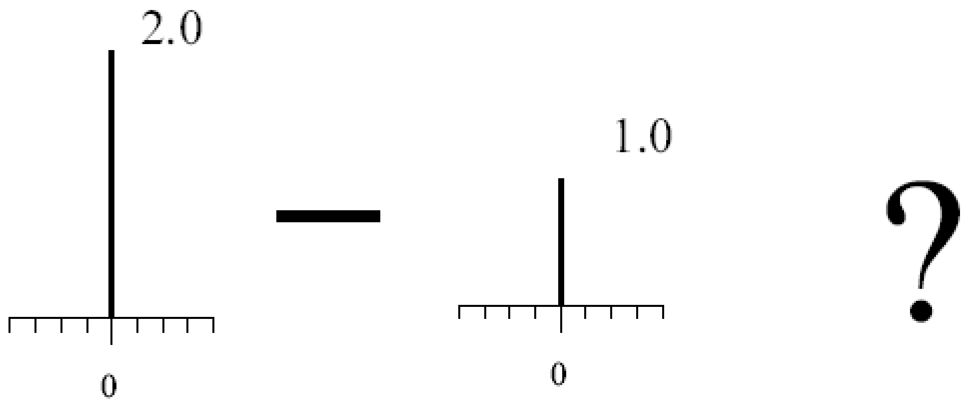
f[m,n]

Source: Torralba, MIT

Practice with linear filters

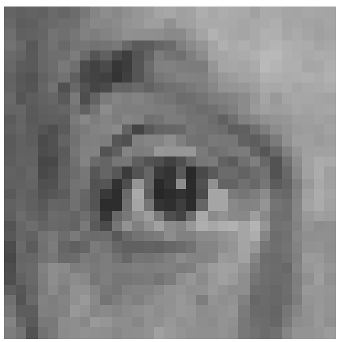


original

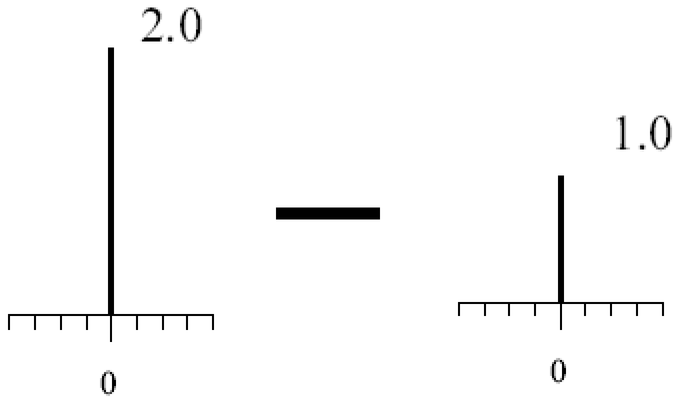


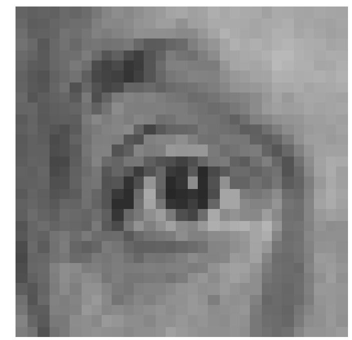
adapted from Darrell and Freeman, MIT

Practice with linear filters



original

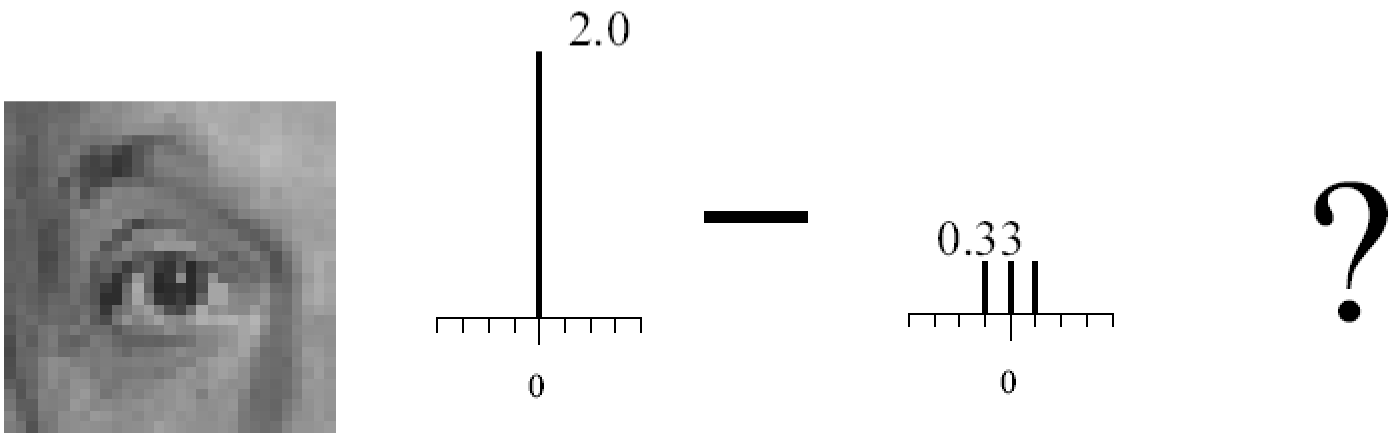




Filtered
(no change)

adapted from Darrell and Freeman, MIT

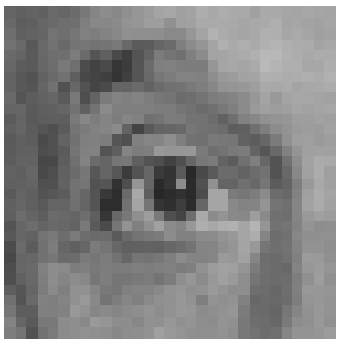
Practice with linear filters



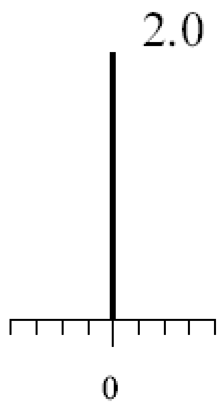
original

adapted from Darrell and Freeman, MIT

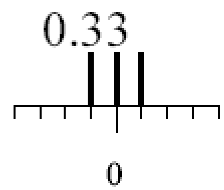
Sharpening

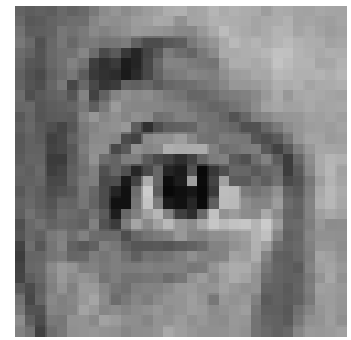


original



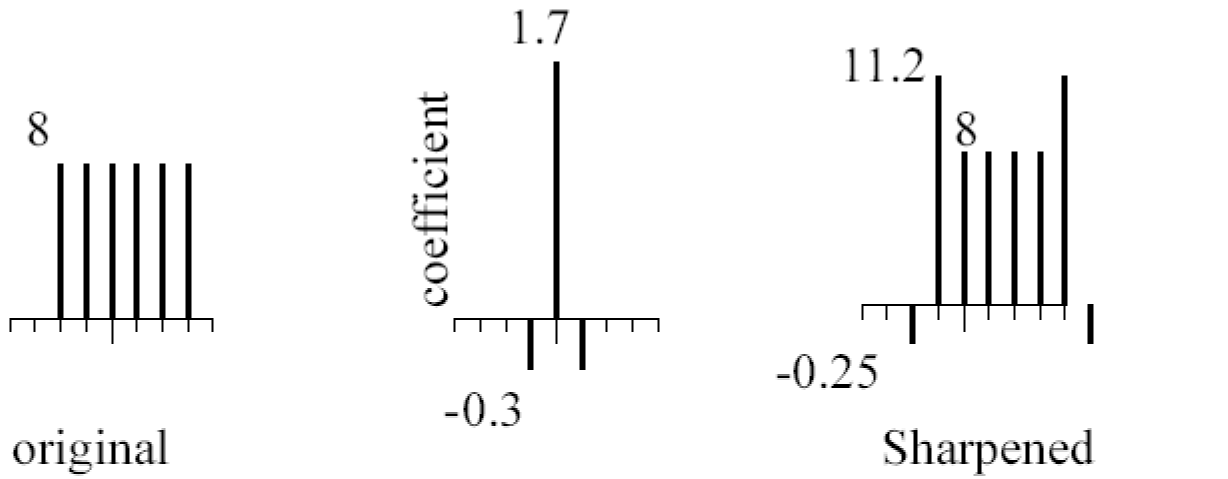
−





Sharpened
original

Sharpening



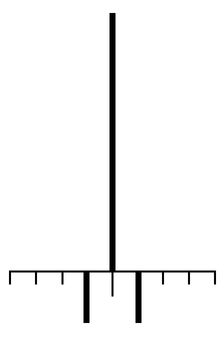
Sharpened
(differences are accentuated; constant areas are left untouched).

adapted from Darrell and Freeman, MIT

Practice with linear filters

Original

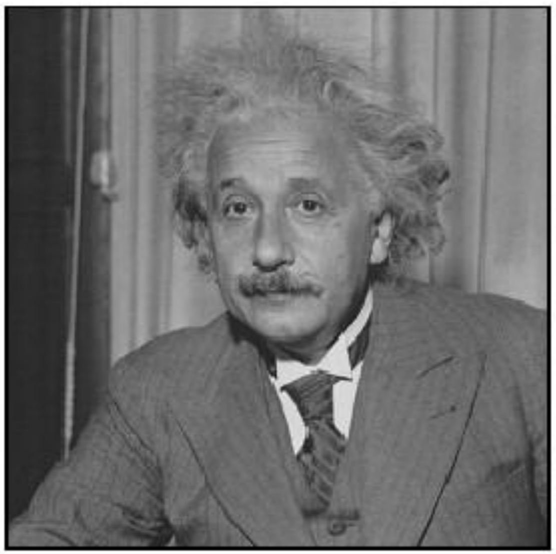
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



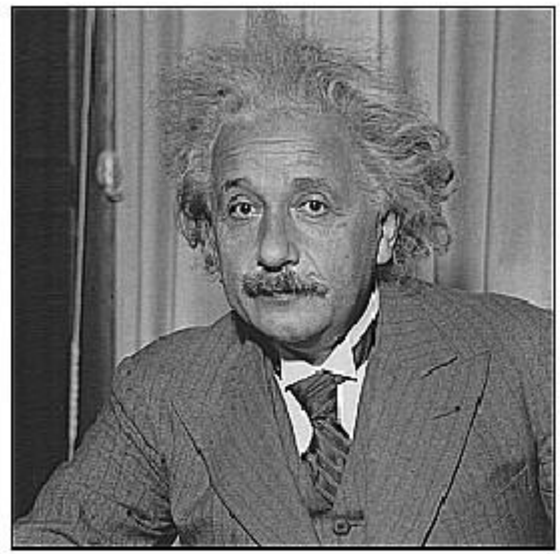
Sharpening filter

- Accentuates differences with local average

Filtering examples: sharpening



before



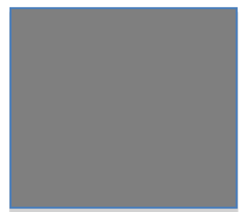
after

Source: Darrell, Berkeley

Rectangular filter

g[m,n]

$\otimes$



h[m,n]

=

f[m,n]

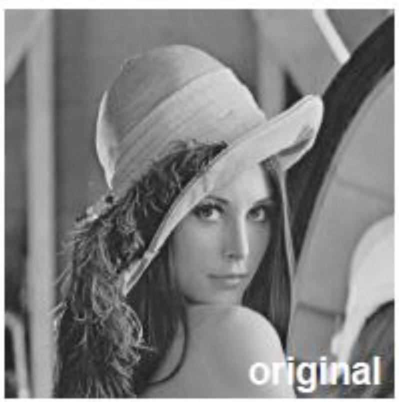
Source: Torralba, MIT

What does blurring take away?

−

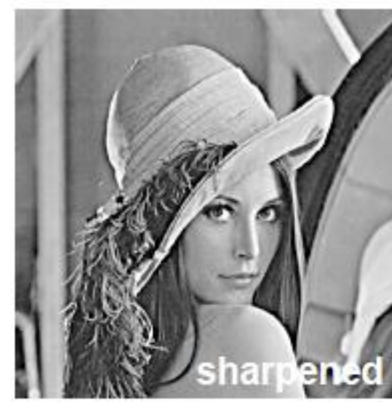
=

- Let's add it back:



+ a

=



Source: Fei Fei Li, Stanford

Rectangular filter

g[m,n]

$\otimes$

h[m,n]

=

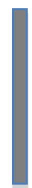
f[m,n]

Source: Torralba, MIT

Rectangular filter

g[m,n]

$\otimes$



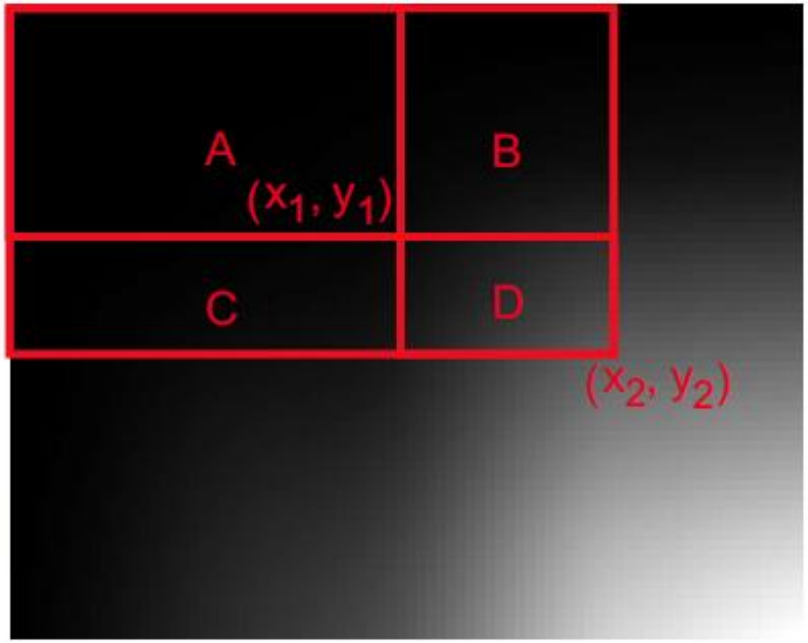
h[m,n]

=

f[m,n]

Source: Torralba, MIT

Integral image



Source: Torralba, MIT

Shift invariant linear system

- **Shift invariant:**
  - Operator behaves the same everywhere, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.

- **Linear:**
  - Superposition: $h * (f1 + f2) = (h * f1) + (h * f2)$
  - Scaling: $h * (k f) = k (h * f)$

Source: Darrell, Berkeley

Properties of convolution

- Linear & shift invariant
- Commutative:

  $f * g = g * f$

- Associative

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

- Identity:

  unit impulse e = [..., 0, 0, 1, 0, 0, ...]. $f * e = f$

- Differentiation:

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

Source: Darrell, Berkeley

Separability

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

Source: Darrell, Berkeley

Separability

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

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

$$f = \begin{bmatrix}1\\ 2\\ 1\end{bmatrix} \qquad \begin{bmatrix} & 11 & \\ & 18 & \\ & 18 & \end{bmatrix} \qquad \begin{bmatrix} & & \\ & 65 & \\ & & \end{bmatrix}$$

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

$$\begin{bmatrix}1\\ 2\\ 1\end{bmatrix} \times \begin{bmatrix}1 & 2 & 1\end{bmatrix} = \begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \quad \begin{bmatrix}2 & 3 & 3\\ 3 & 5 & 5\\ 4 & 4 & 6\end{bmatrix} \quad \begin{aligned} &= 2 + 6 + 3 = 11\\ &= 6 + 20 + 10 = 36\\ &= 4 + 8 + 6 = 18\\ \hline &\quad 65 \end{aligned}$$

Source: Darrell, Berkeley $f * (g * h) = (f * g) * h$

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

Effect of smoothing filters

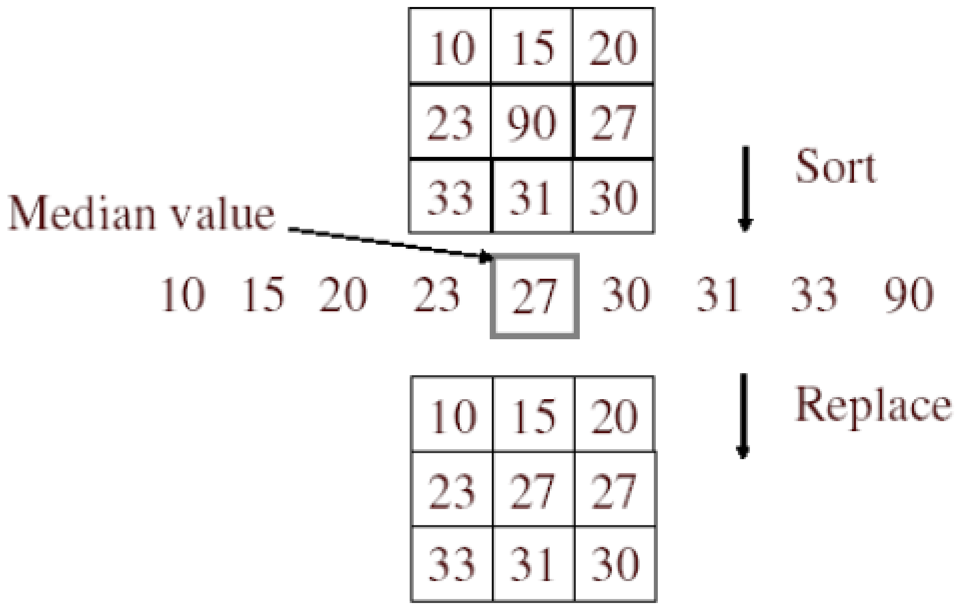
5x5

Additive Gaussian noise

Salt and pepper noise

Source: Darrell, Berkeley

Median filter

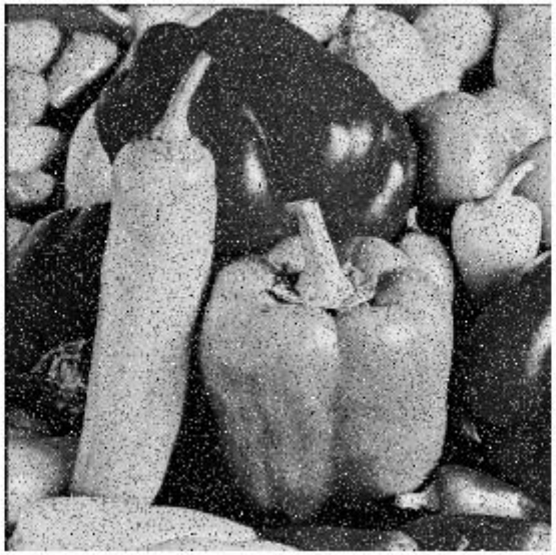


- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise

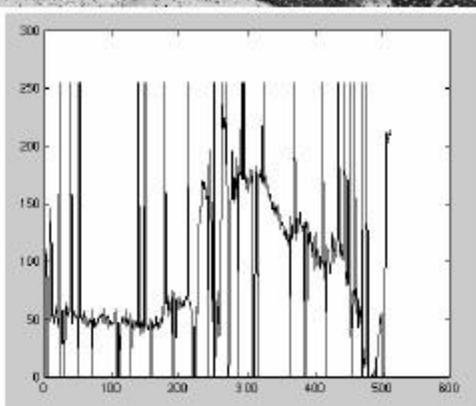
Source: Darrell, Berkeley

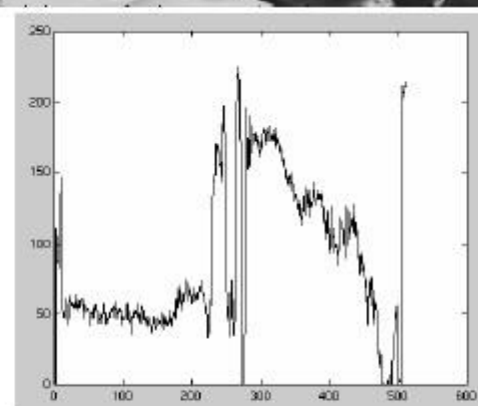
Median filter

Salt and pepper noise



Median filtered





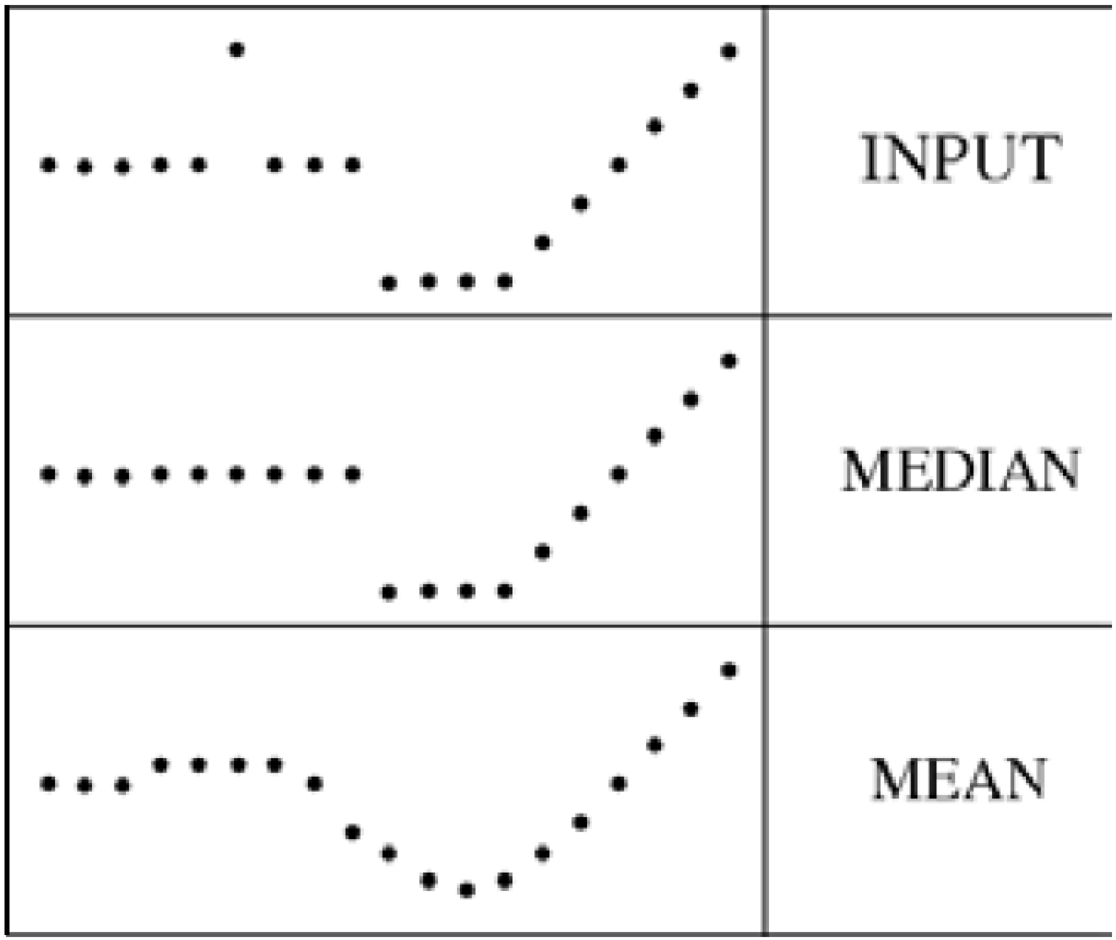
Plots of a row of the image

Source: Darrell, Berkeley

Source: M. Hebert

Median filter

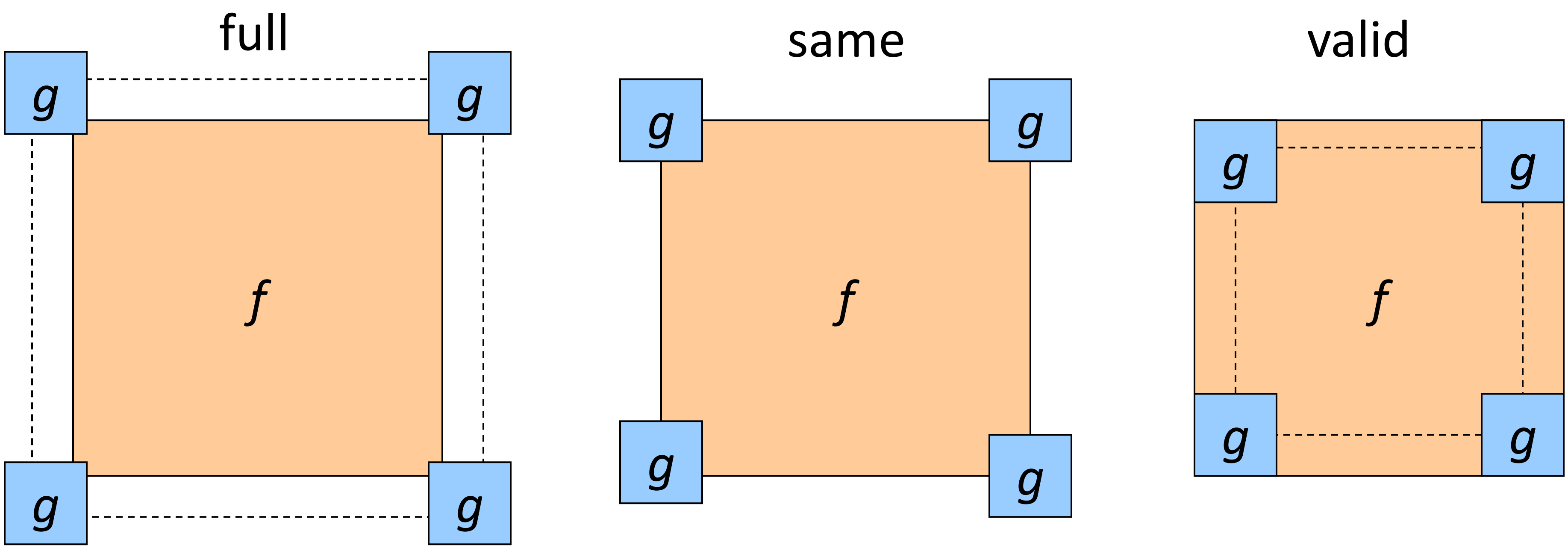
- Median filter is edge preserving



Source: Darrell, Berkeley

Boundary issues

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



Source: Darrell, Berkeley

Source: S. Lazebnik

Boundary issues

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

Source: Darrell, Berkeley

Source: S. Marschner

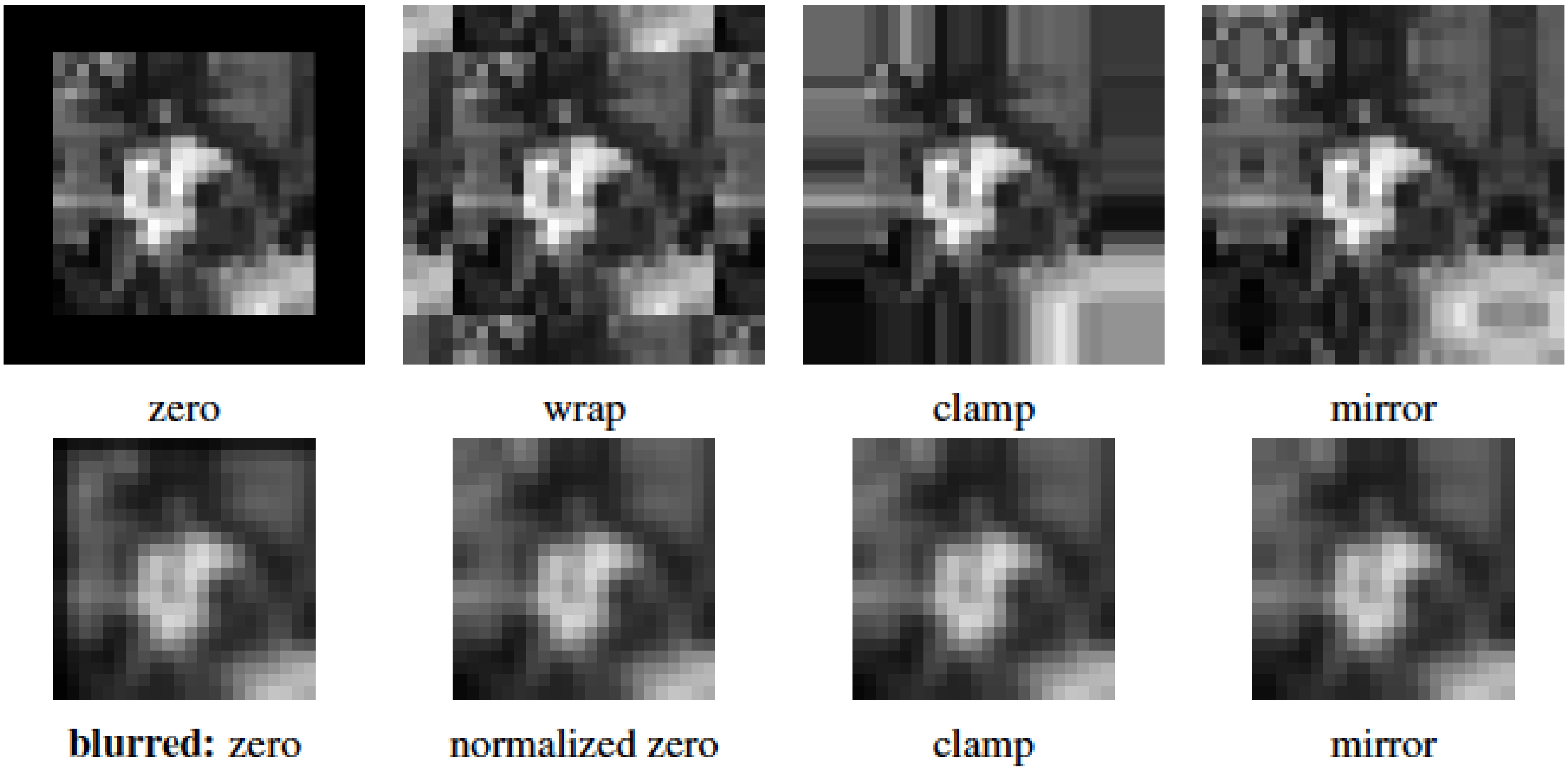
Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods (MATLAB):
    - clip filter (black): imfilter(f, g, 0)
    - wrap around: imfilter(f, g, ‘circular’)
    - copy edge: imfilter(f, g, ‘replicate’)
    - reflect across edge: imfilter(f, g, ‘symmetric’)

Source: Darrell, Berkeley

Source: S. Marschner

Borders



Source: Torralba, MIT

From Rick's book