CS4442/9542b
Artificial Intelligence II
prof. Olga Veksler

Lecture 8

Computer Vision

Introduction, Filtering

Some slides from: D. Jacobs, D. Lowe, S. Seitz , A.Efros , X. Li, R. Fergus, J. Hayes, S. Lazebnik, D. Hoiem, S. Marschner

Outline

- Very Brief Intro to Computer Vision
- Digital Images
- Image Filtering
  - noise reduction

Every Picture Tells a Story

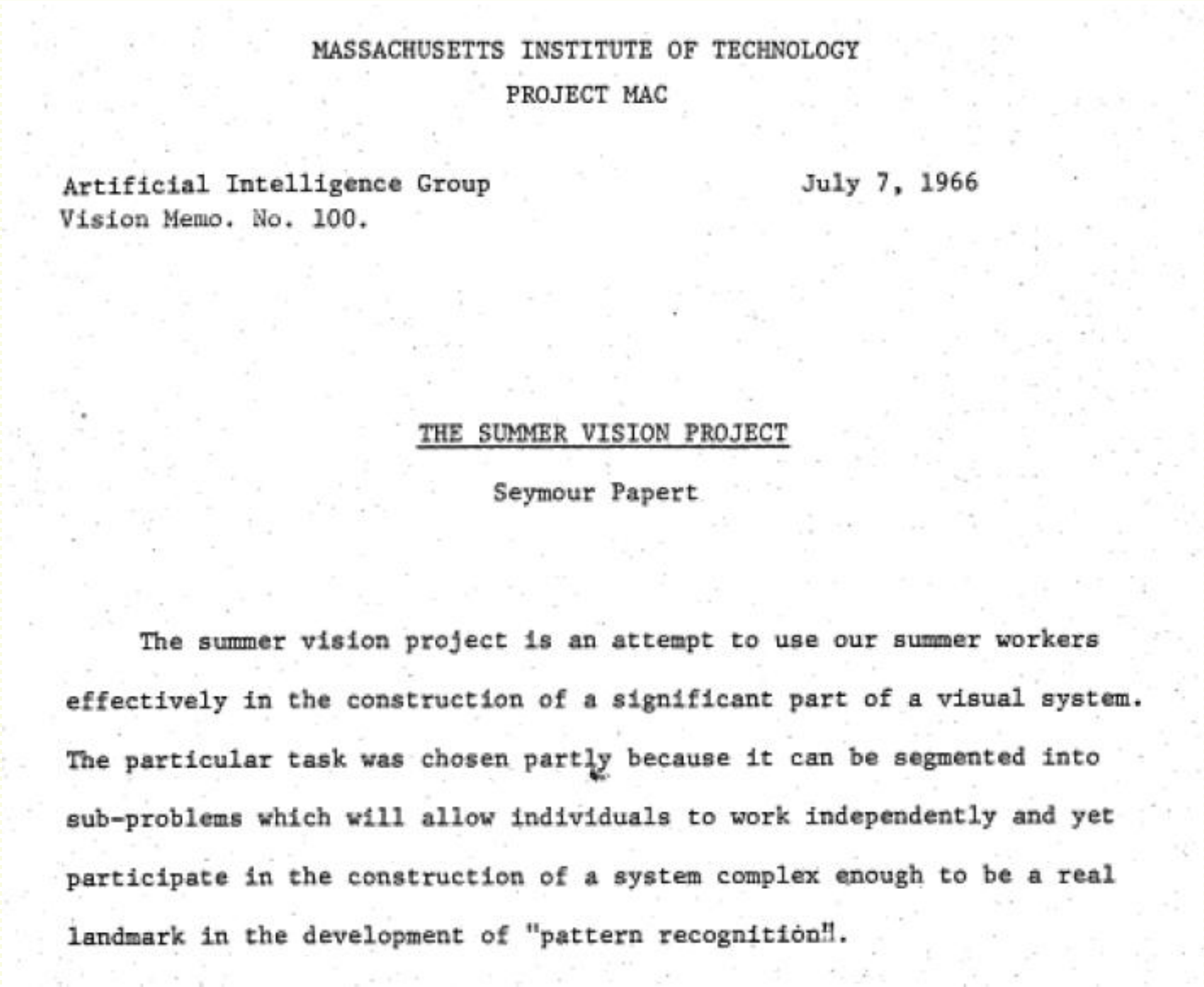
- Goal of computer vision is to write computer programs that can interpret images
  - bridge the gap between the pixels and the story

what we see

1	2	0	2	2	1
9	2	2	7	1	2
2	8	2	3	2	2
4	2	2	7	2	8
2	2	2	6	0	2
8	3	2	5	2	2
7	2	4	2	1	9

what computers see

Origin of Computer Vision: MIT Summer Project


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

The problem

- Want to make a computer understand images
- We know it is possible, we do it effortlessly!

sensing device

interpreting device

interpretations

real world scene

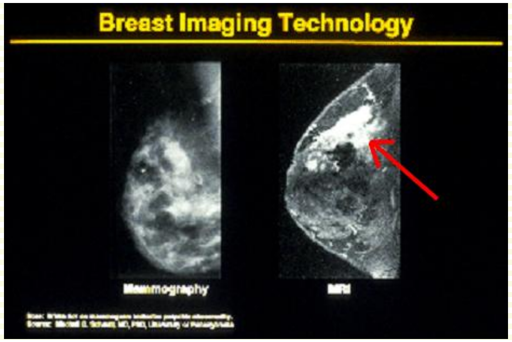
a person, a person with folded arms, Pietro Perona

Just Copy Human Visual System?

- People try to but we don't yet have a sufficient understanding of how our visual system works
- $O(10^{11})$ neurons used in vision
  - about 1/3 of human brain
- Latest CPUs have only $O(10^8)$ transistors
  - most are cache memory
- Very different architectures:
  - Brain is slow but parallel
  - Computer is fast but mainly serial
- Bird vs Airplane
  - Same underlying principles
  - Very different hardware

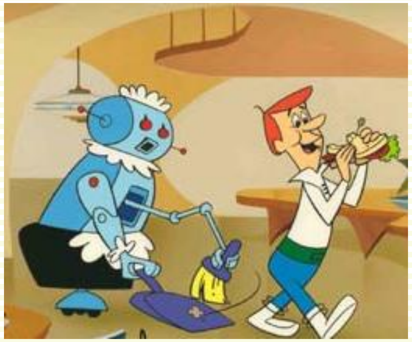
Why Computer Vision Matters

Safety



Health

Security



Comfort

Fun

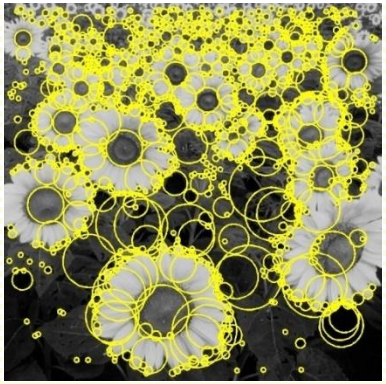
Personal Photos

"Early Vision" Problems

- Edge extraction

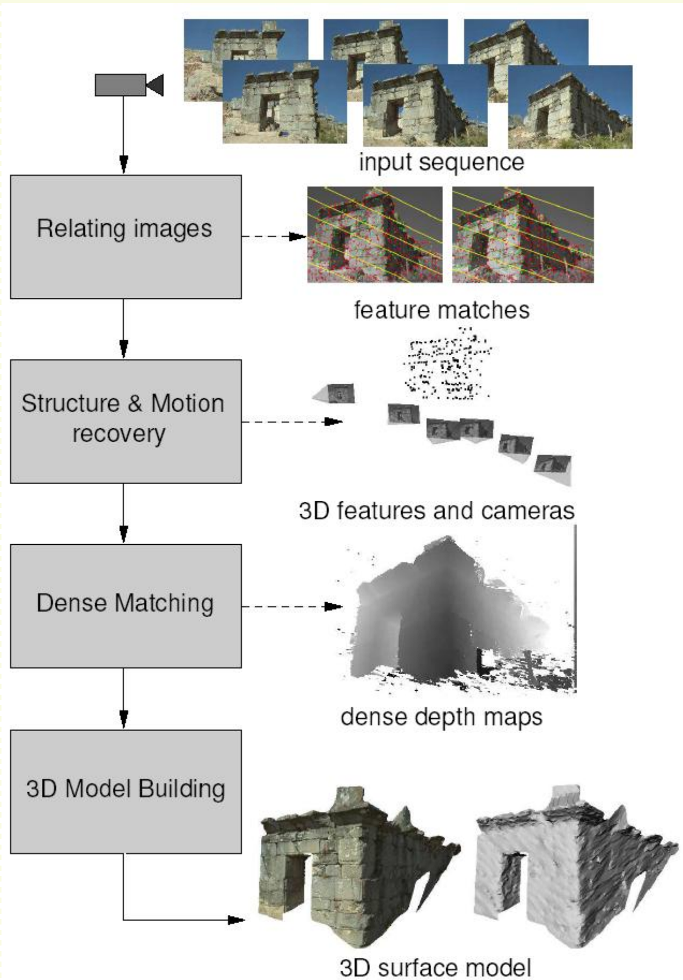
- Corner extraction

- Blob extraction

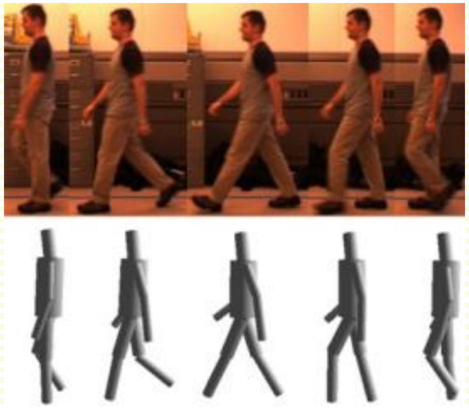


“Mid-level Vision” Problems

- 3D Structure extraction



- Motion and tracking

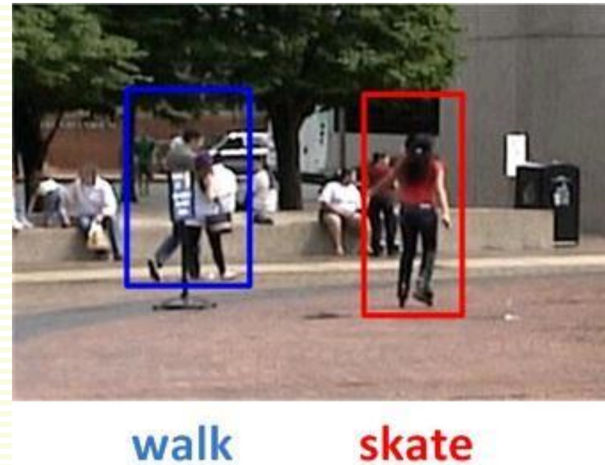


- Segmentation

“High-level Vision” Problems

- Face Detection

- Action Recognition

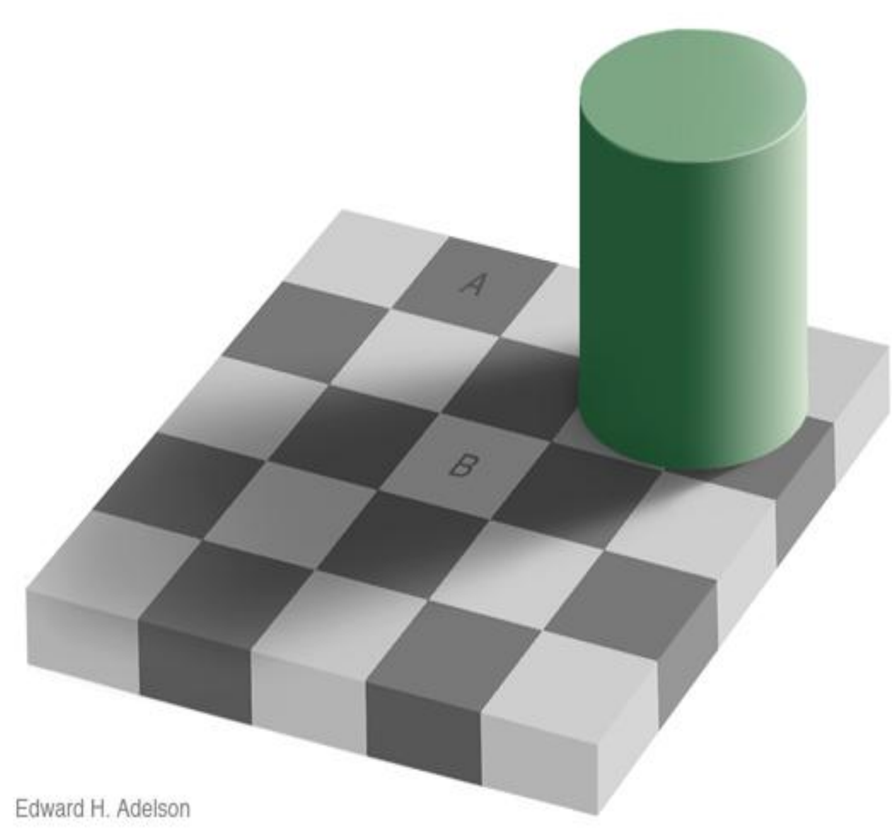


- Object Recognition

- Scene Recognition

Vision is inferential: Illumination

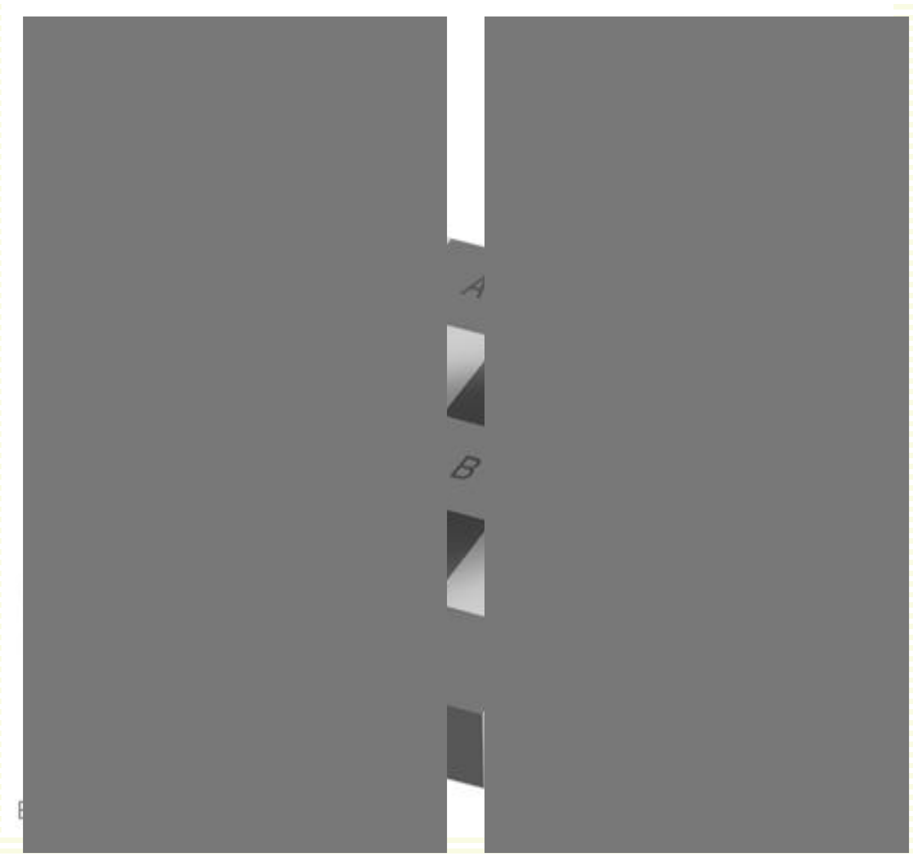
- Vision is hard: even the simple problem of color perception is inferential



http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html

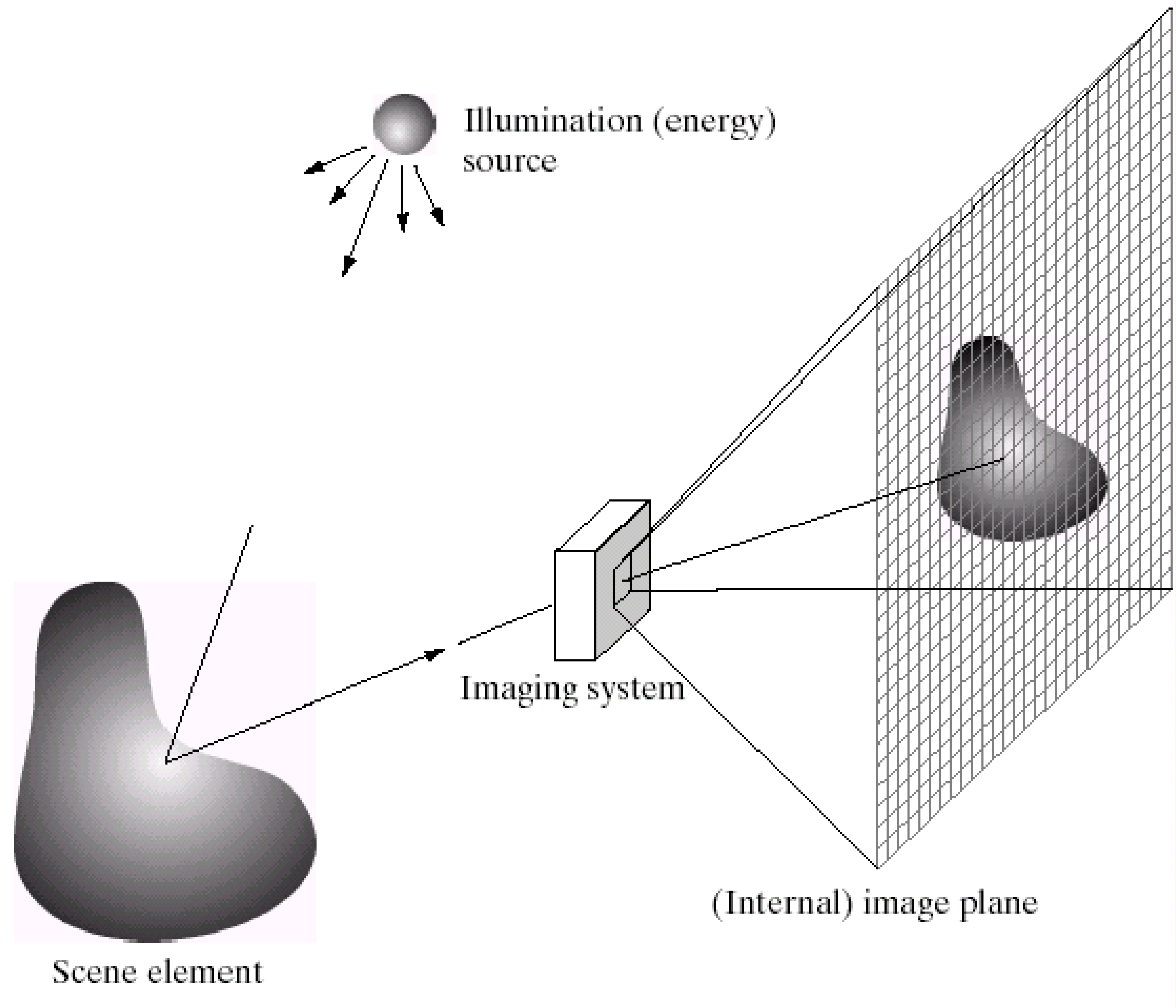
Vision is inferential: Illumination

- Vision is hard: even the simple problem of color perception is inferential



http://web.mit.edu/persci/people/adelson/checkershadow_illusion.html

Image Formation



Sampling and Quantization

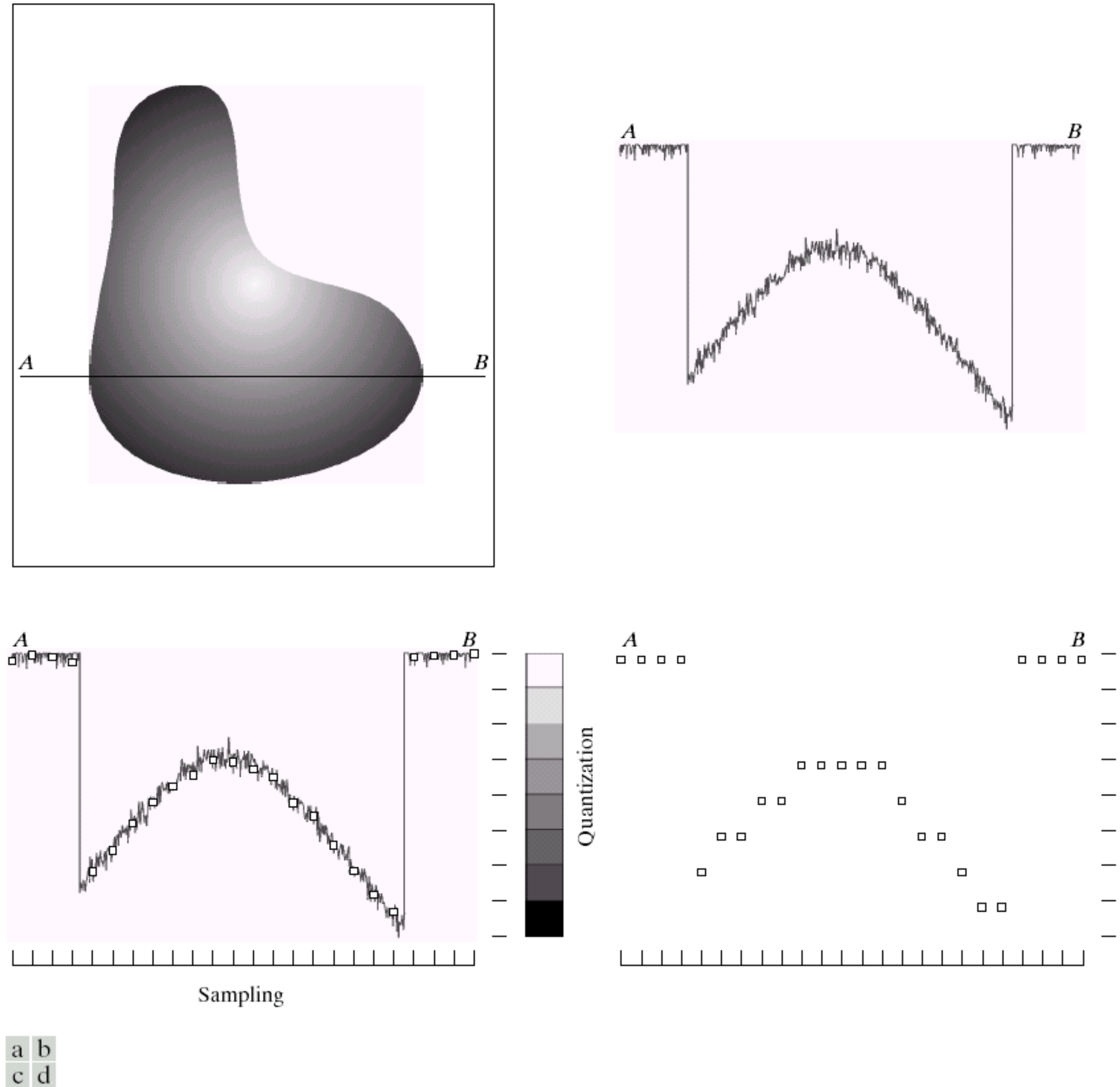
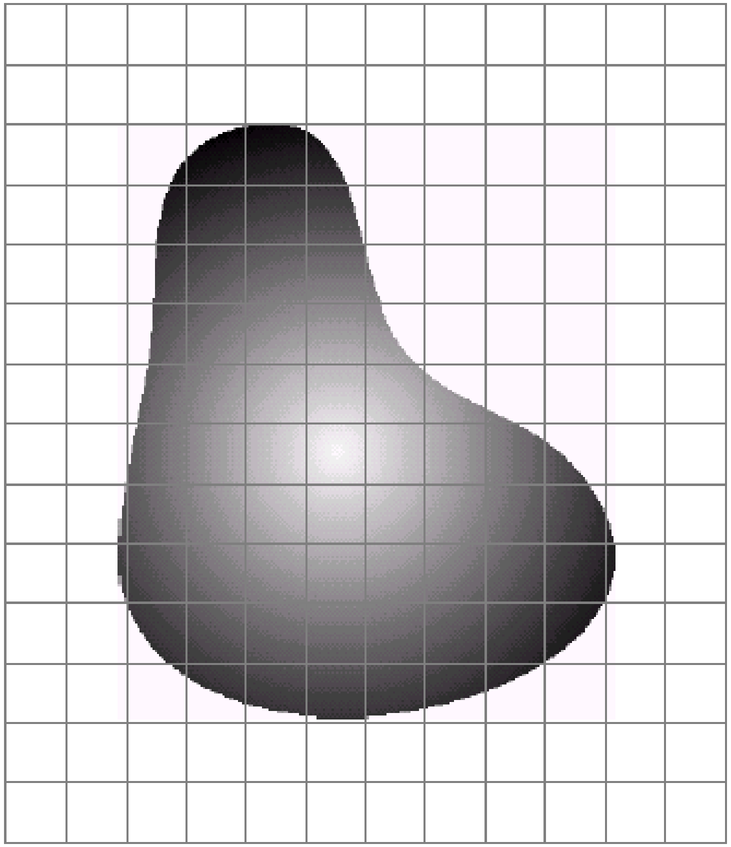
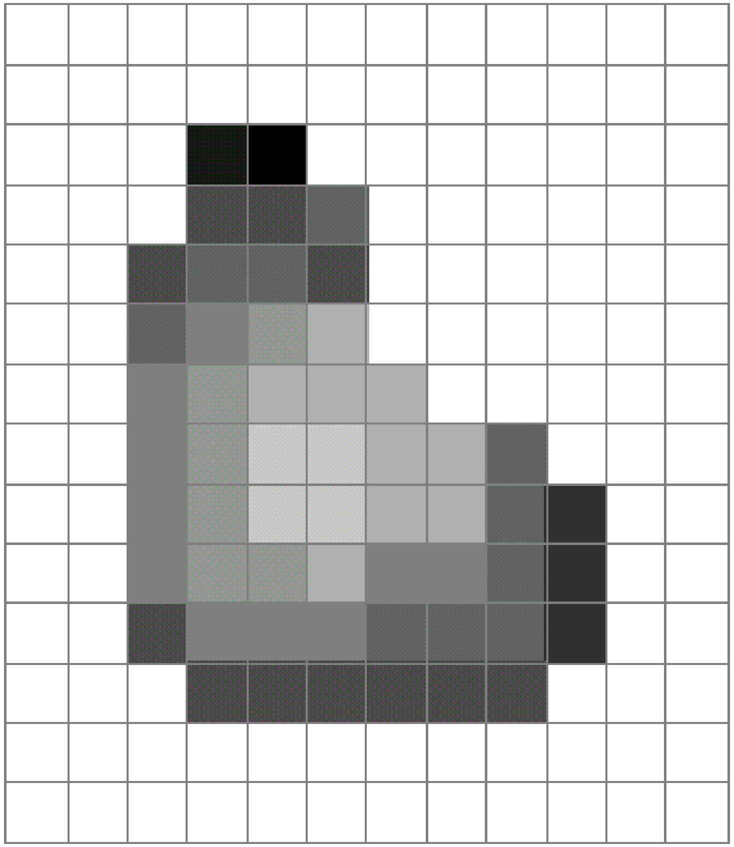


FIGURE 2.16 Generating a digital image. (a) Continuous image. (b) A scan line from A to B in the continuous image, used to illustrate the concepts of sampling and quantization. (c) Sampling and quantization. (d) Digital scan line.

Sensor Array



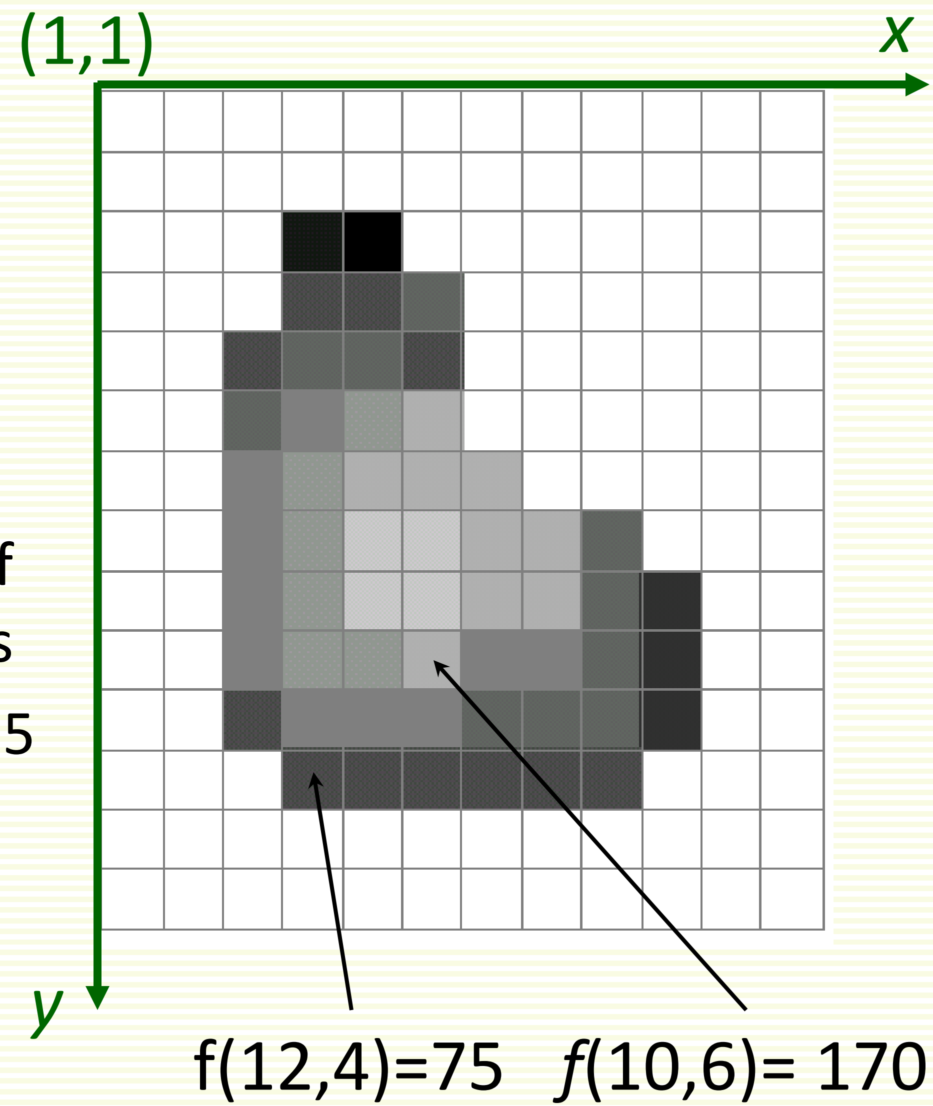
real world object



after quantization and sampling

Digital Grayscale Image

- Image is array $f(x,y)$
  - approximates continuous function $f(x,y)$ from R^2 to R:
- $f(x,y)$ is the **intensity** or **grayscale** at position (x,y)
  - proportional to brightness of the real world point it images
  - standard range: 0, 1, 2,…., 255

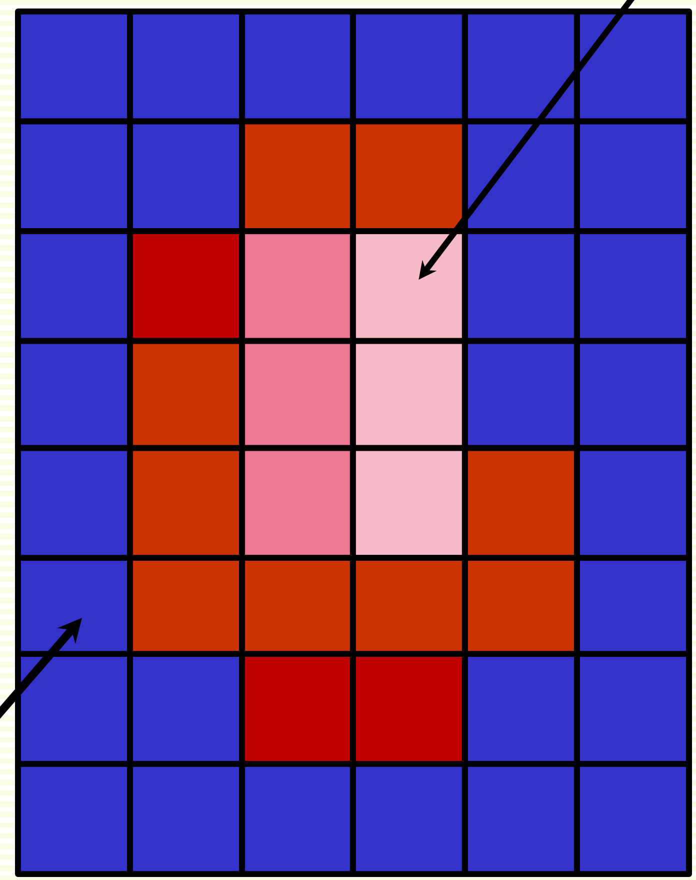


Digital Color Image

- Color image is three functions pasted together
- Write this as a vector-valued function:

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

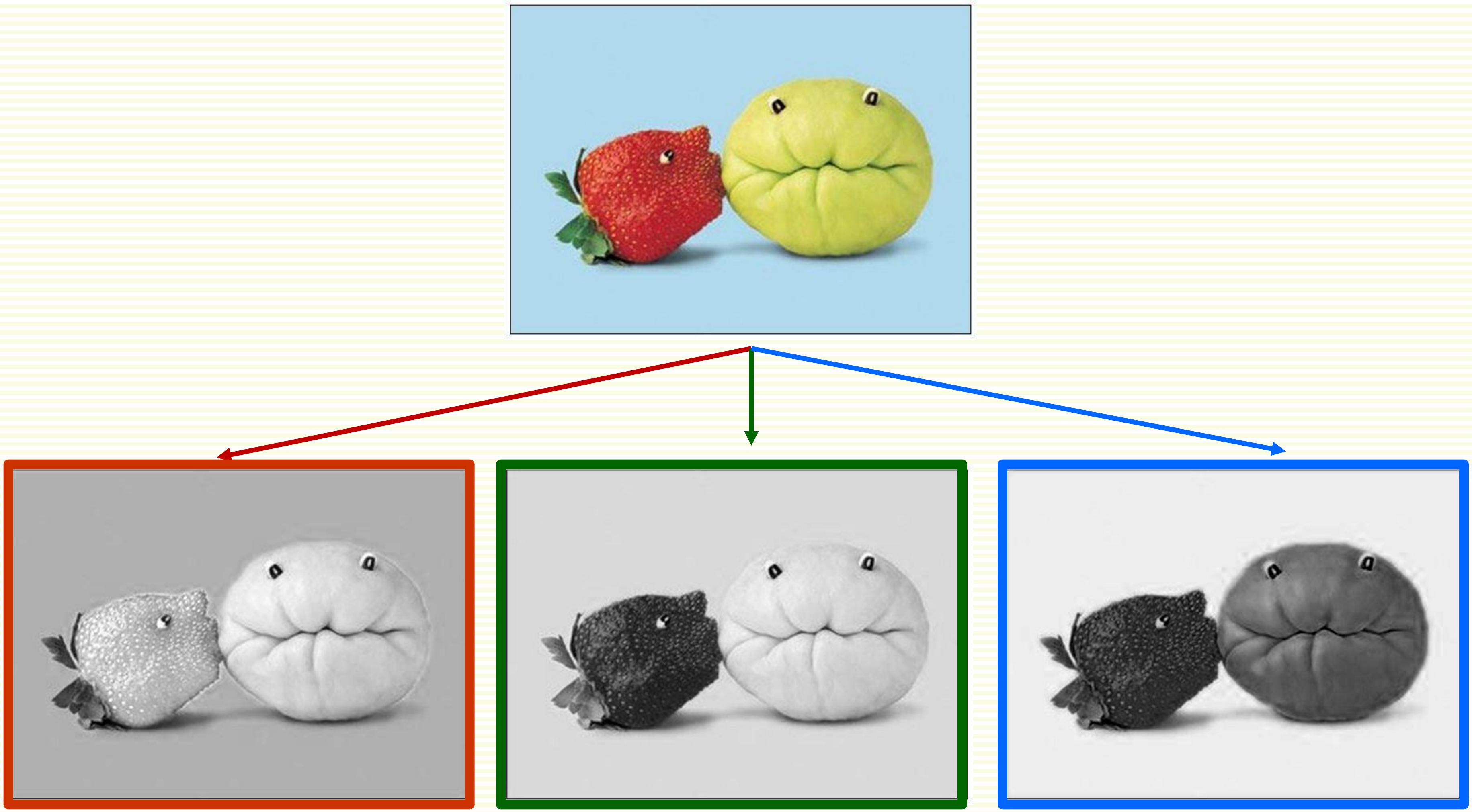
$$\begin{bmatrix} 200 \\ 50 \\ 50 \end{bmatrix}$$



$$\begin{bmatrix} 0 \\ 10 \\ 120 \end{bmatrix}$$

Digital Color Image

- Can consider color image as 3 separate images: R, G, B



R G B

Image Filtering

- Given $f(x,y)$ filtering computes new image $h(x,y)$
  - $h(x,y)$ is a function of $f(x,y)$ in a local neighborhood around (x,y)
  - example: $h(x,y) = f(x,y) + f(x-1,y) \times f(x,y-1)$
- Linear filtering: function is a weighted sum (or difference) of pixel values

  $h(x,y) = f(x,y) + 2 \times f(x-1,y-1) - 3 \times f(x+1,y+1)$
- Many applications
  - Enhance images
    - denoise, resize, increase contrast, …
  - Extract information from images
    - texture, edges, distinctive points …
  - Detect patterns
    - template matching

1	2	4	2	8
9	2	2	7	5
2	8	1	3	9
4	3	2	7	2
2	2	2	6	1
8	3	2	5	4

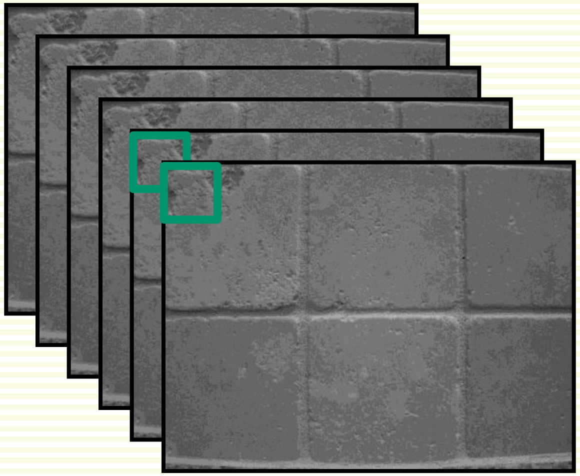
$h(4,2) = 3 + 4 \times 8 = 35$

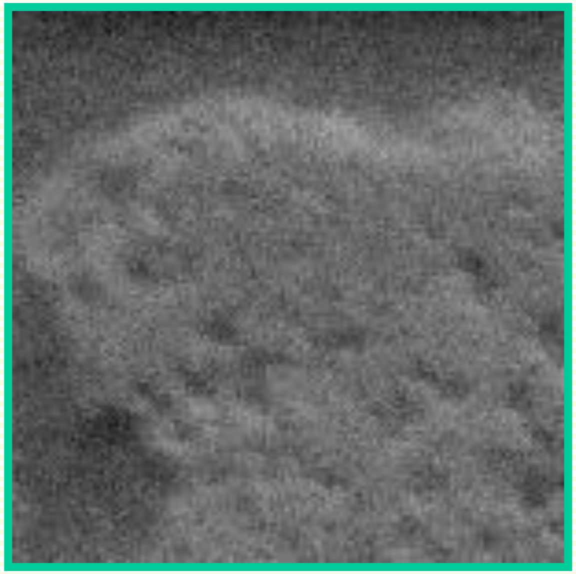
$h(6,5) = 4 + 5 \times 1 = 9$

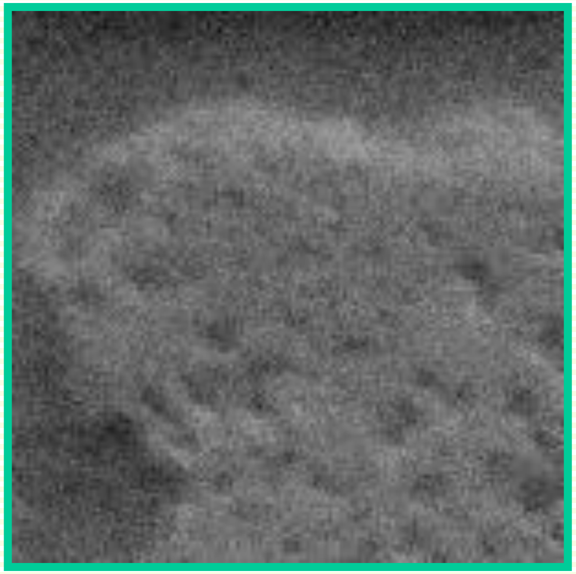
$h(2,4) = 7 + 2 \times 4 - 3 \times 9 = -12$

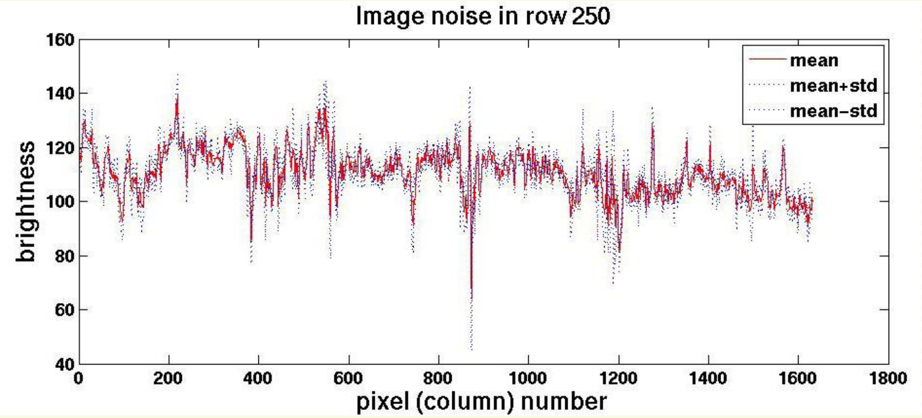
Filtering for Noise Reduction: Motivation

- Multiple images of even the **same static scene** are not identical





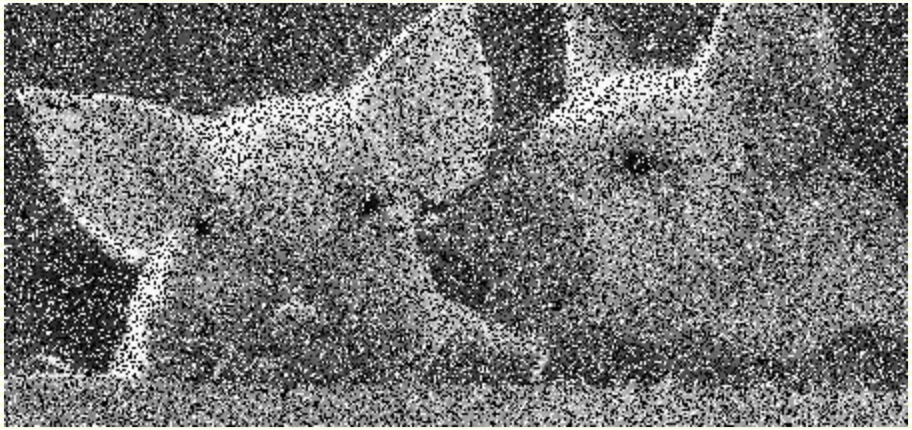




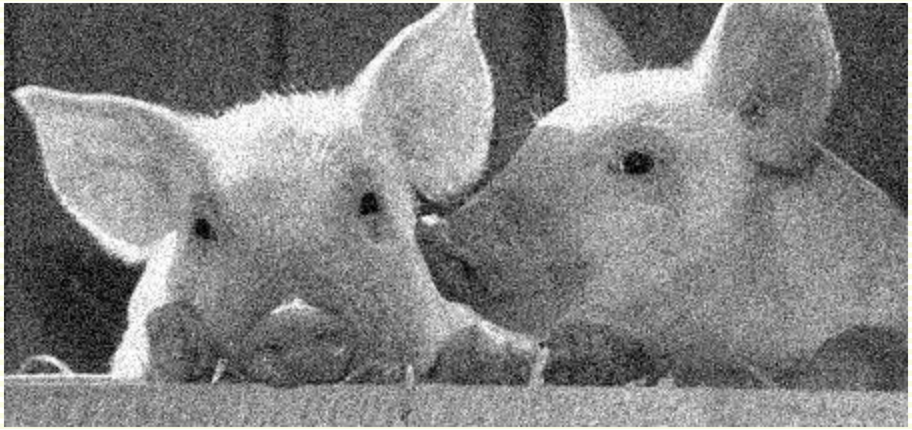
Common Types of Noise

original image

Impulse noise: random occurrences of white pixels

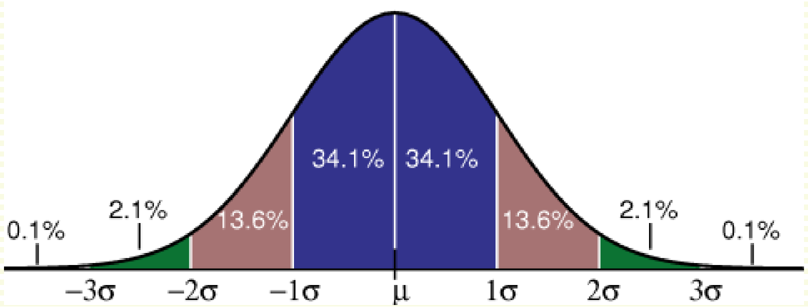


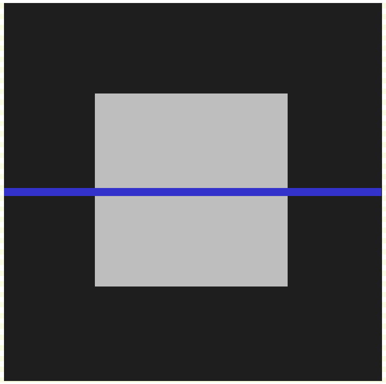
Salt and pepper noise: random occurrences of black and white pixels



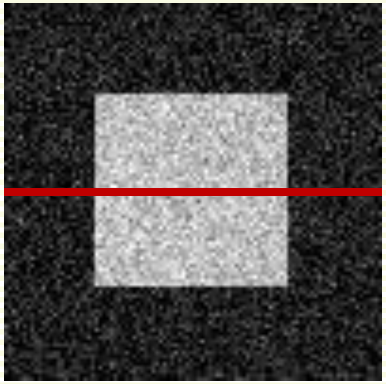
Gaussian noise: variations in intensity drawn from a Gaussian distribution

Gaussian Noise Most Commonly Assumed

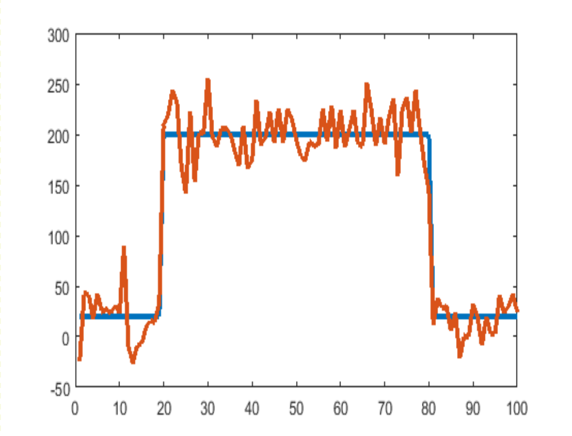




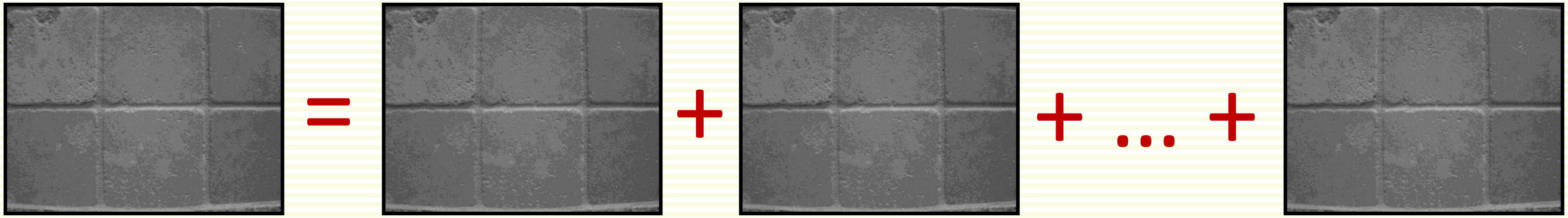
original image



***G*(0,25) noise**



Noise Reduction



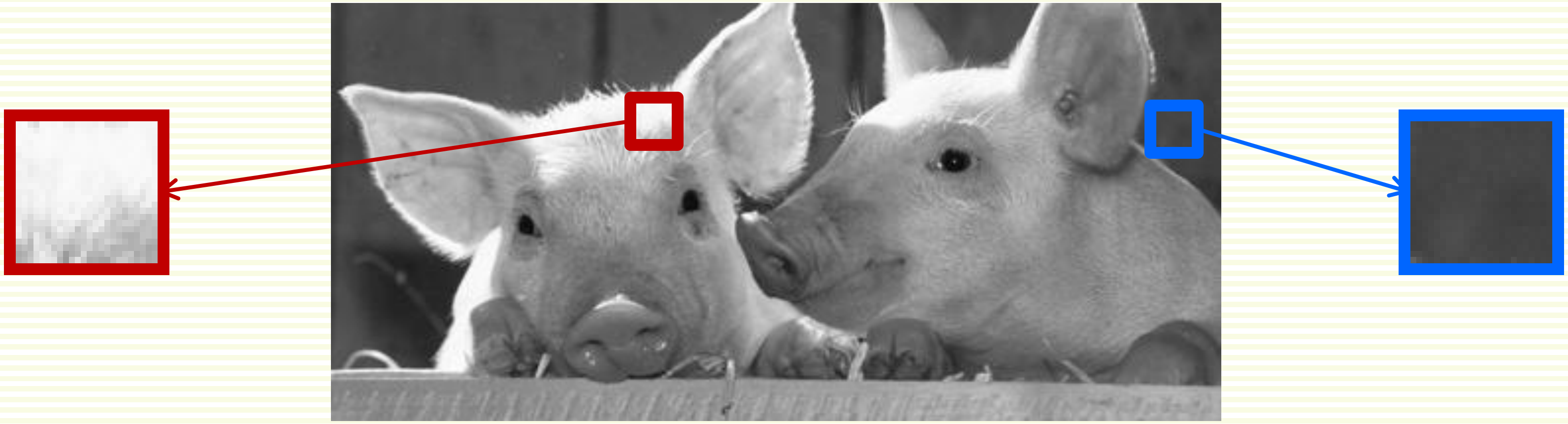
- Noise can be reduced by averaging
- If we had multiple images, simply average them

  $f_{\text{final}}(x,y) = (f_1(x,y) + f_2(x,y) + \ldots + f_n(x,y)))/n$

- **But usually there is only one image!**

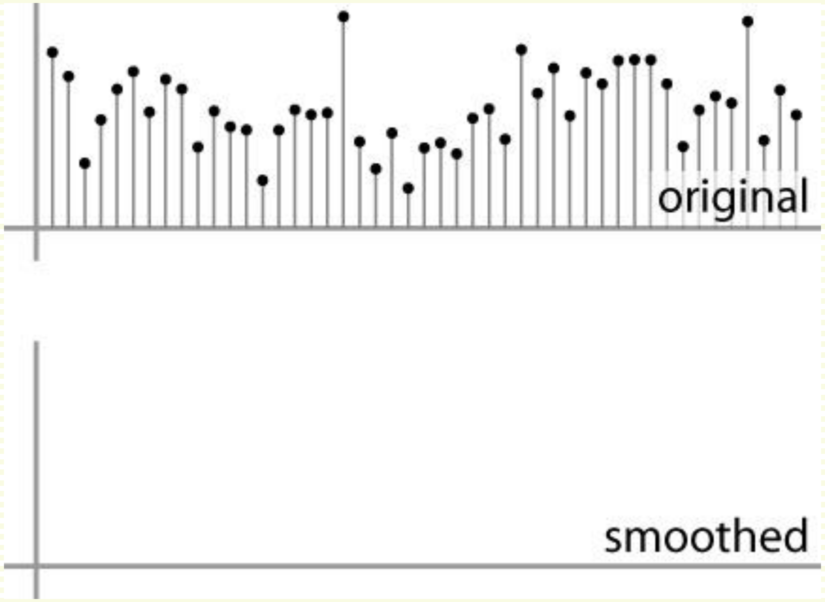
First Attempt at a Solution

- Replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - expect a pixel to have intensities similar to its neighbors
  - noise is independent at each pixel



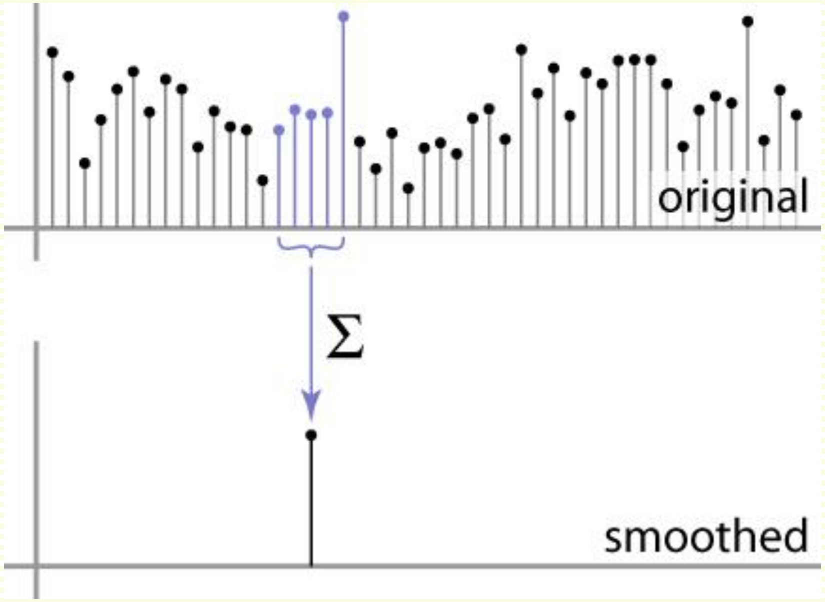
Average Filter in 1D

- Replace each pixel with an average of all the values in its neighborhood (= 5 pixels, say)
- Moving average:



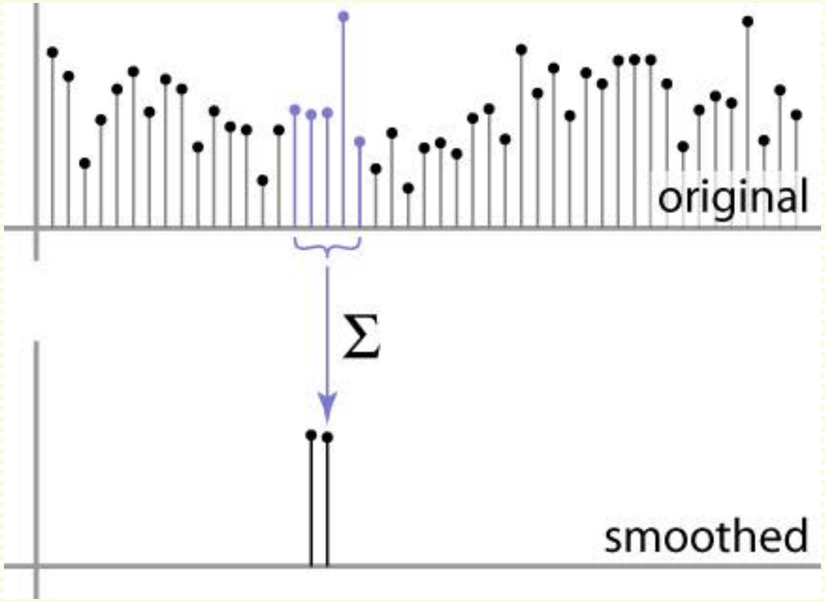
Average Filter in 1D

- Replace each pixel with an average of all the values in its neighborhood (= 5 pixels, say)
- Moving average in 1D



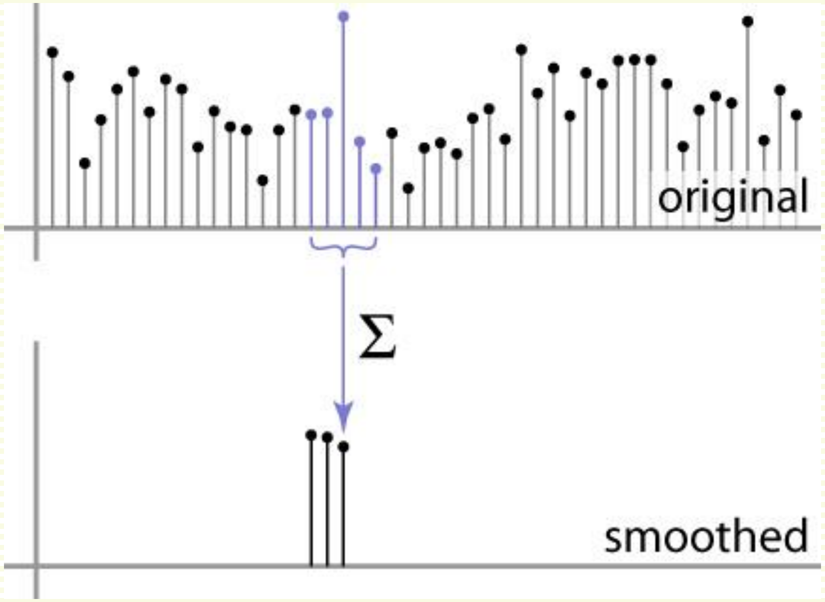
Average Filter in 1D

- Replace each pixel with an average of all the values in its neighborhood (= 5 pixels, say)
- Moving average in 1D



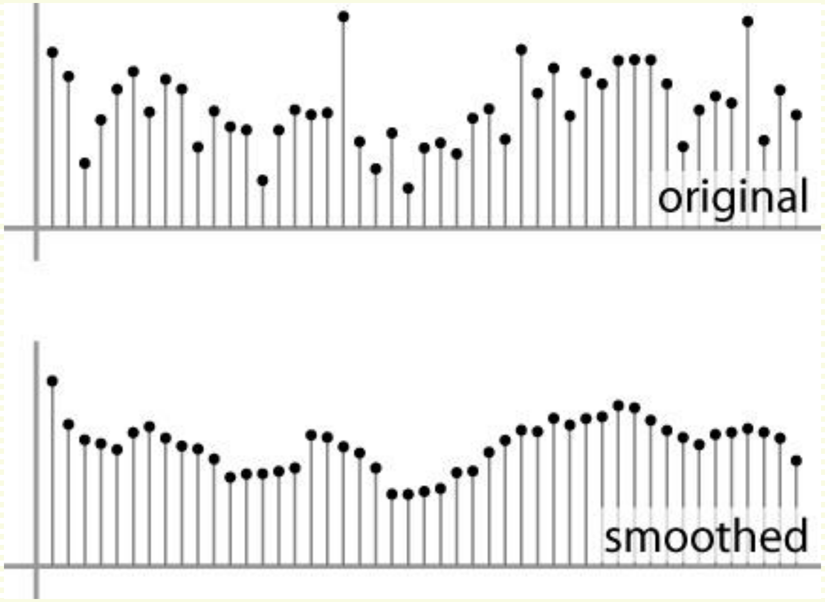
Average Filter in 1D

- Replace each pixel with an average of all the values in its neighborhood (= 5 pixels, say)
- Moving average in 1D



Average Filter in 1D

- Replace each pixel with an average of all the values in its neighborhood (= 5 pixels, say)
- Moving average in 1D



Average Filter 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								

Average Filter 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							

Average Filter 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						

Average Filter 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					

Average Filter 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				

Average Filter 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	

Average Filter in 2D

$f(x,y)$

sharp border

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

sticking out

$g(x,y)$

border washed out

	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	

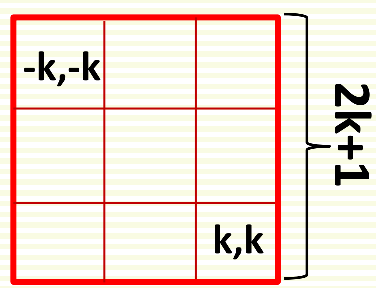
not sticking out

Average Filter in 2D

- Write as equation, averaging in window of size (2k+1)x(2k+1)

$$g(x,y)=\underbrace{\frac{1}{(2k+1)^2}}_{\text{normalizing factor}}\underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k}f(x+u,y+v)}_{\textit{loop over all pixels in neighborhood around pixel } f\text{(i,j)}}$$

- Window indexing



Average Filter in 2D

$$g(x,y)=\frac{1}{(2k+1)^2}\sum_{u=-k}^{k}\sum_{v=-k}^{k}f(x+u,y+v)$$

- Bring normalizing factor inside the sum

$$g(x,y)=\sum_{u=-k}^{k}\sum_{v=-k}^{k}\frac{1}{(2k+1)^2}f(x+u,y+v)=\sum_{u=-k}^{k}\sum_{v=-k}^{k}H[u,v]f(x+u,y+v)$$

- Visualize with **mask H**
  - also called **filter, kernel**

$$\underbrace{\begin{array}{|c|c|c|}\hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array}}_{H[u,v]} = 1/9 \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Average Filter in 2D

- Apply mask *H* to every image pixel

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

H[*u*,*v*]

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

box filter

g(*x*,*y*)

	0	10	20	30	30				

Correlation Filtering

$$g(x,y)=\sum_{u=-k}^{k}\sum_{v=-k}^{k}H[u,v]f(x+u,y+v)$$

- Box filter

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \quad H[u,v]$$

- Generalize by allowing different weights for different pixels in the neighborhood

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \quad H[u,v]$$

Filtering in 2D

- Apply the more general mask as before

$f(x,y)$

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

$H[u,v]$

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

$g(x,y)$

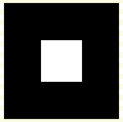
	0	6	20	23	23				

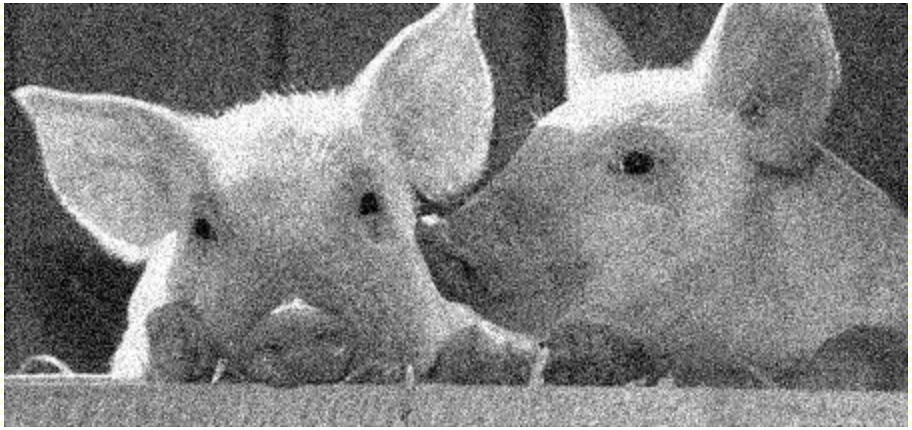
Correlation filtering

$$g(x,y) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] f(x+u, y+v)$$

- This is called correlation, denoted $g = H \otimes f$
  - The result of applying mask H to the whole image
- Filtering an image: replace each pixel with a linear combination of its neighbors
- The filter kernel or mask *H* is gives the weights in linear combination

Smoothing by Averaging

- Pictorial representation of box filter: 
  - white means large value, black means low value



original

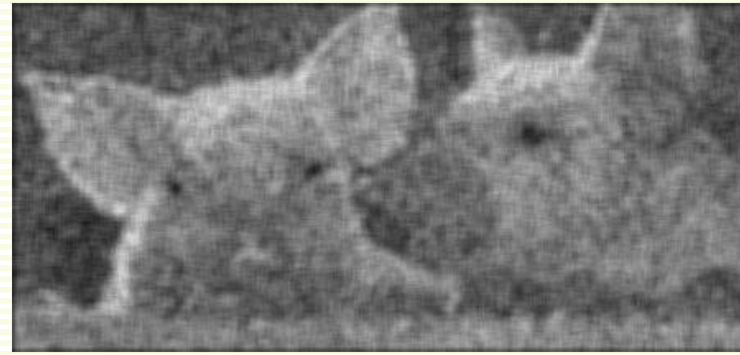
filtered

- What if the mask is larger than 3x3 ?

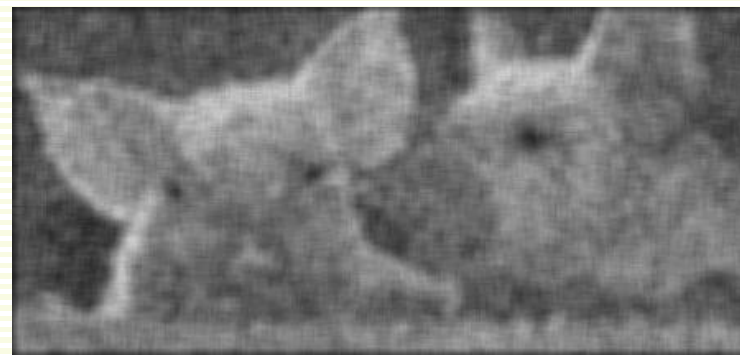
Effect of Average Filter

Gaussian noise | **Salt and Pepper noise**

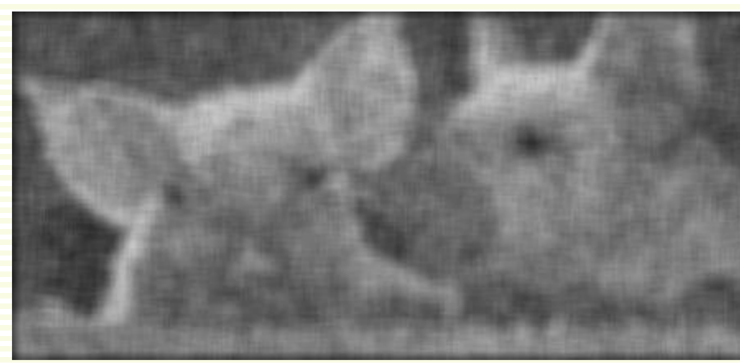
7×7

9×9

11×11

Gaussian Filter

- Nearest neighboring pixels to have the most influence
  - helps to lessen the effect of boundary smoothing

$f(x,y)$

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

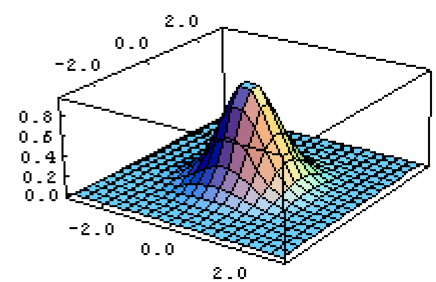
$H[u,v]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

This kernel *H* is an approximation of a 2d Gaussian function:

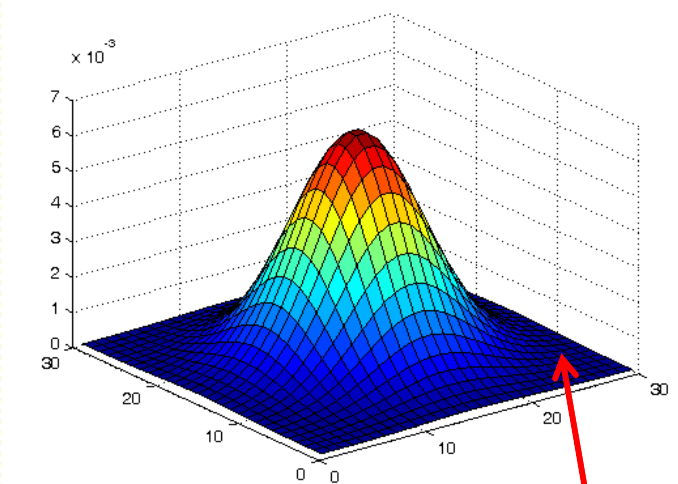
$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

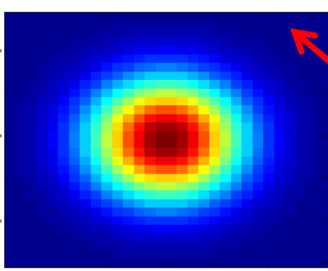


Gaussian Filters: Mask Size

- Gaussian has infinite domain, discrete filters use finite mask
  - set mask size to exclude non-useful (effectively zero) weights

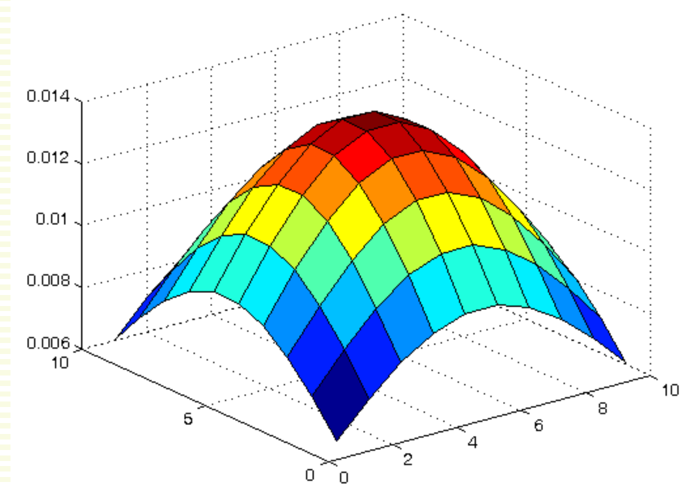
σ = 5 with 30 x 30 mask

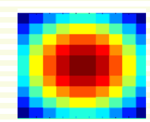




blue weights
are so small
they are
effectively **0**

σ = 5 with 10 x 10 mask

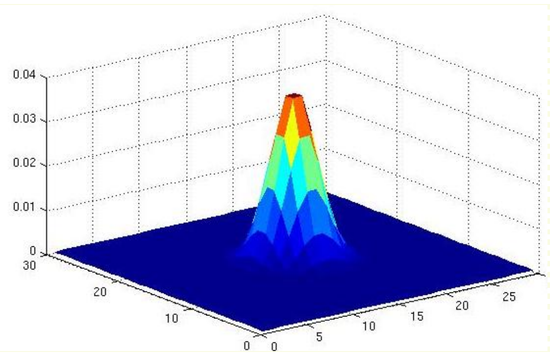


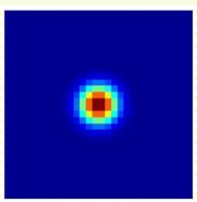


Gaussian filters: Variance

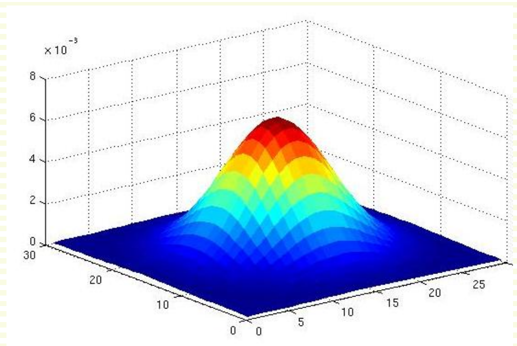
- Variance (σ) contributes to the extent of smoothing
  - larger σ gives less rapidly decreasing weights
  - can construct a larger mask with non-negligible weights

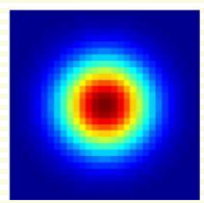
σ = 2 with 30 x 30 kernel



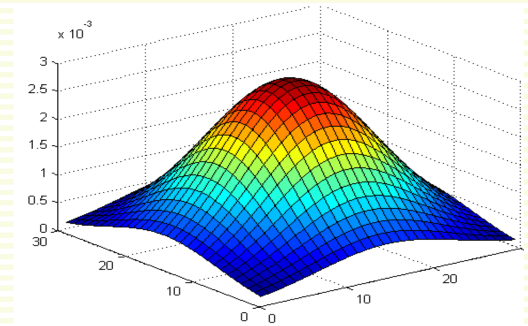


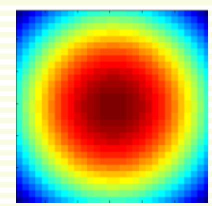
σ = 5 with 30 x 30 kernel





σ = 8 with 30 x 30 kernel





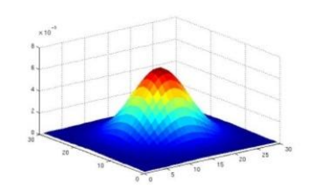
Matlab

im

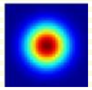
outim

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

>> mesh(h);
```

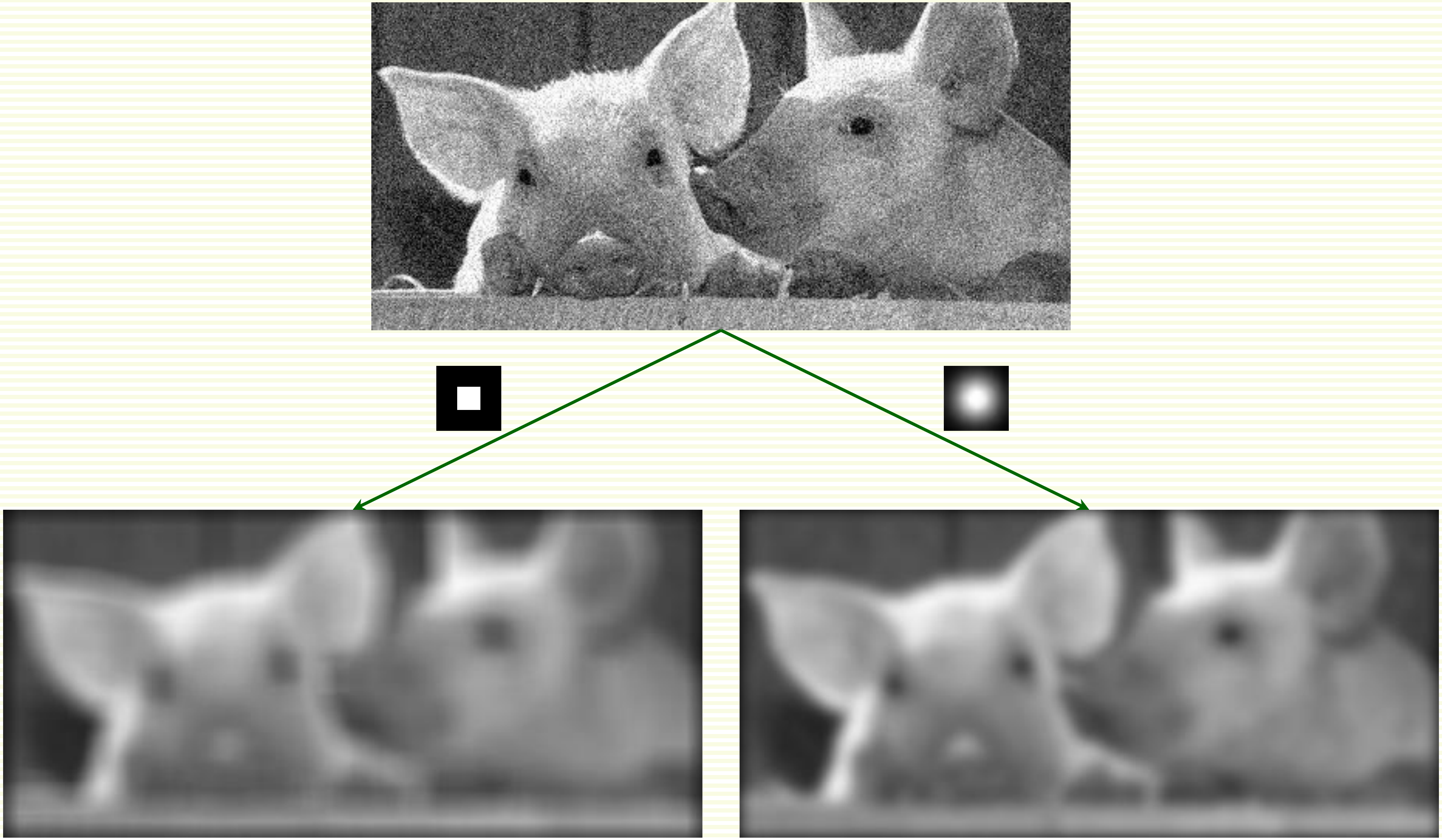


```
>> imagesc(h);
```



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

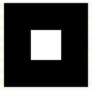
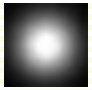
Average vs. Gaussian Filter



mean filter

Gaussian filter

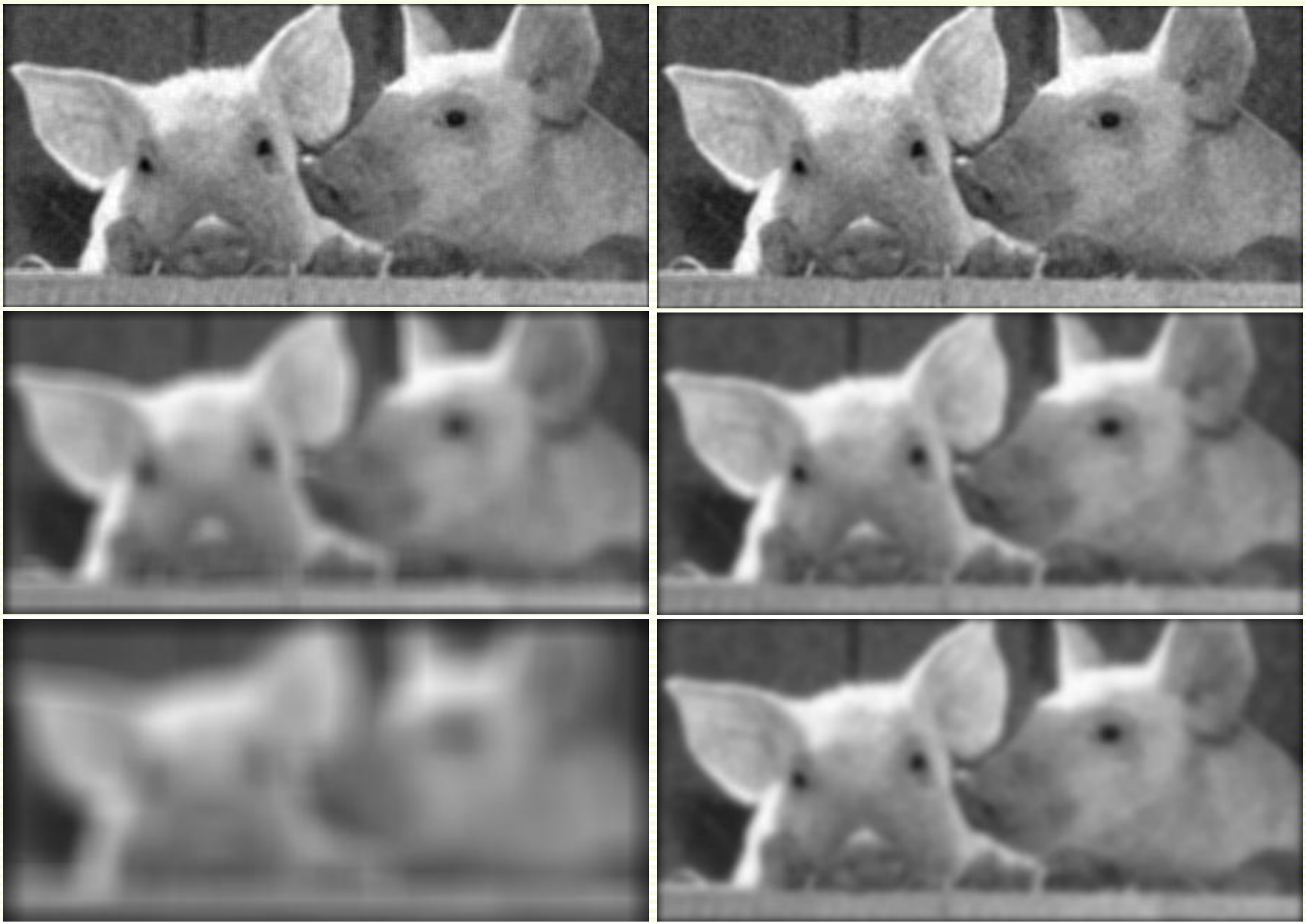
More Average vs. Gaussian Filter

mean filter  **Gaussian filter** 

5 × 5

15 × 15

31 × 31



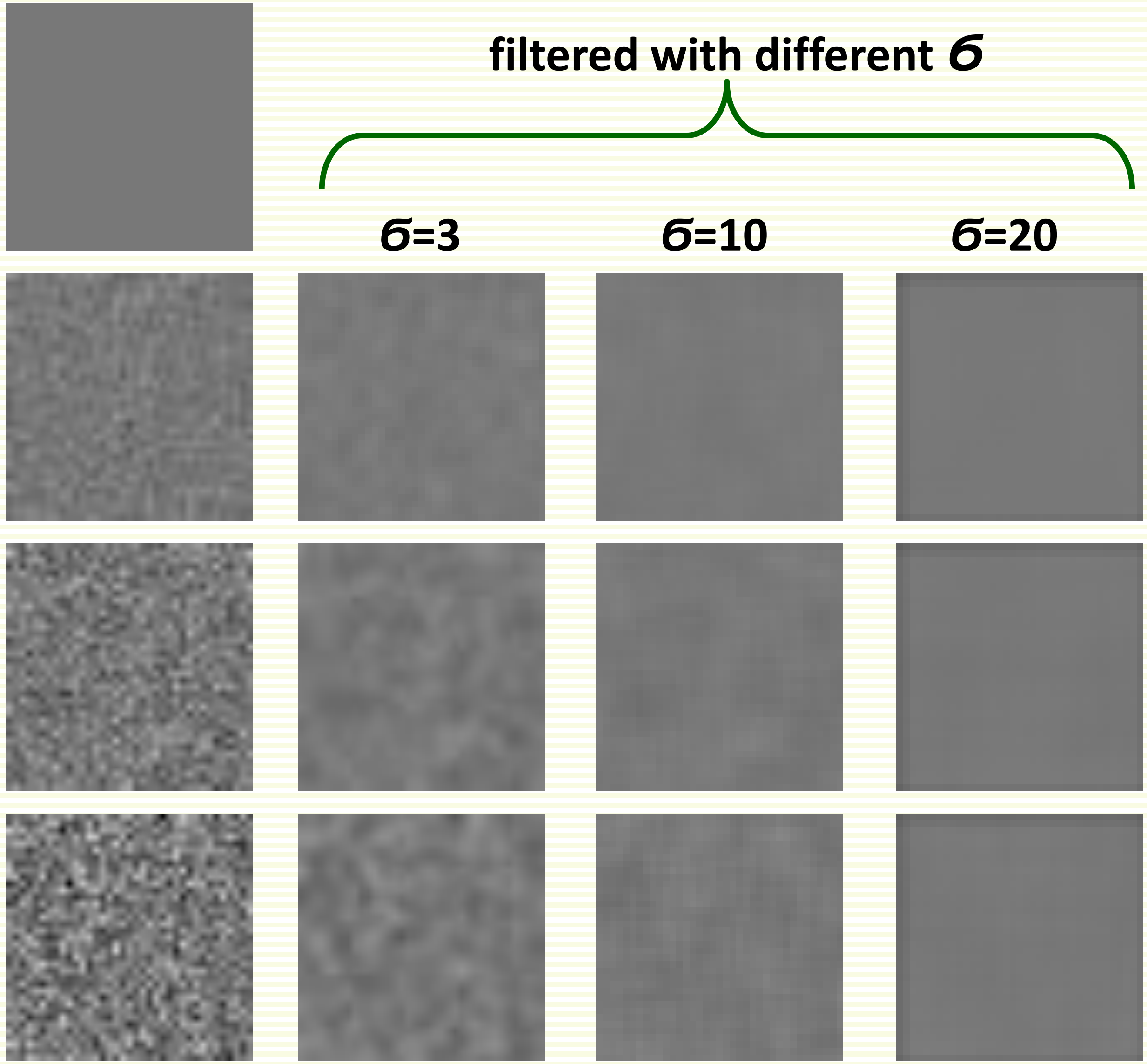
Gaussian Filter with different σ

original image

corrupted by
noise σ = 10

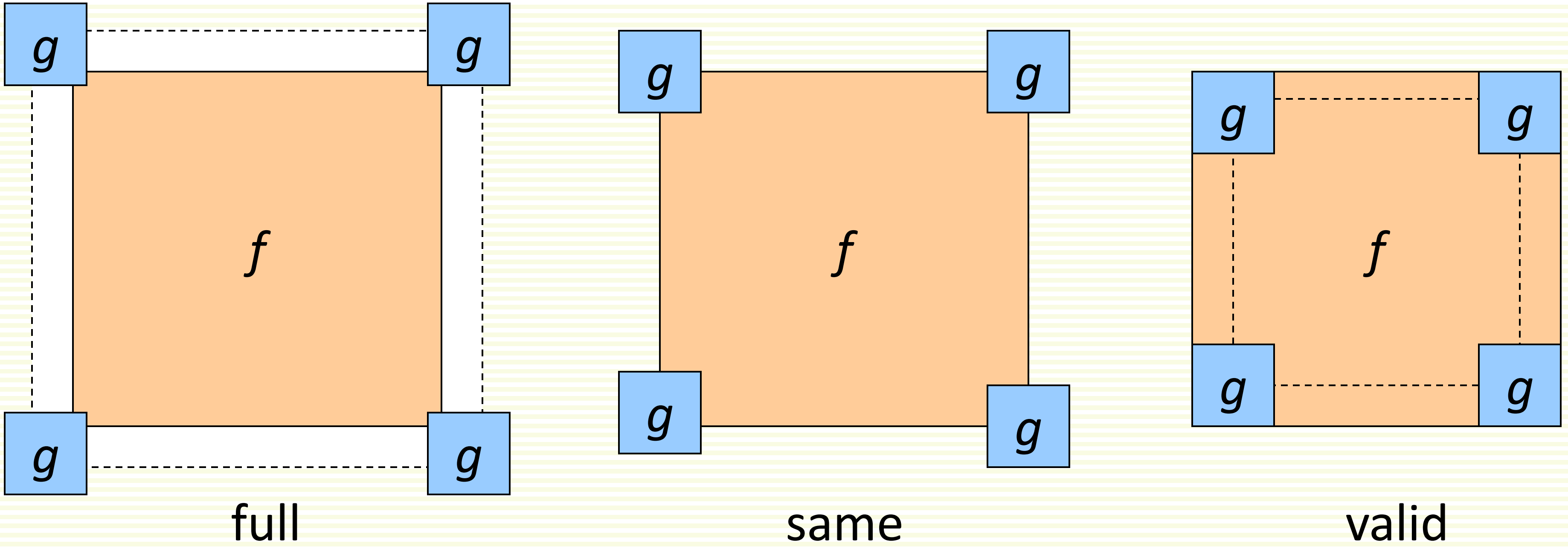
corrupted by
noise σ = 20

corrupted by
noise σ = 30



Boundary Issues

- What is the size of the output?
- MATLAB: output size / "shape" options
  - *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



Boundary issues

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate image

clip filter (black)

copy edge

wrap around

reflect across edge

Properties of Smoothing Filters

- Values positive
- Sum to 1
  - constant regions same as input
  - overall image brightness stays unchanged
- Amount of smoothing proportional to mask size
  - larger mask means more extensive smoothing

Filtering an Impulse Signal

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

0	0	0	0	0	0	0
0	0	0	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
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$f(x,y)$

$\otimes$

a	b	c
d	e	f
g	h	i

$H[u,v]$

=

$g(x,y)=?$

Filtering an Impulse Signal

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

0	0	0	0	0	0	0
0	0	0	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
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$f(x,y)$

$\otimes$

a	b	c
d	e	f
g	h	i

$H[u,v]$

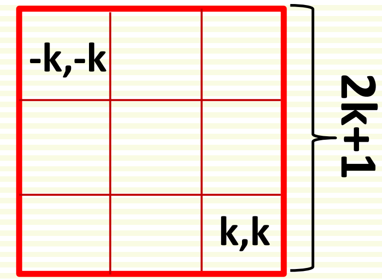
=

		i	h	g		
		f	e	d		
		c	b	a		

$g(x,y)$=?

Convolution

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

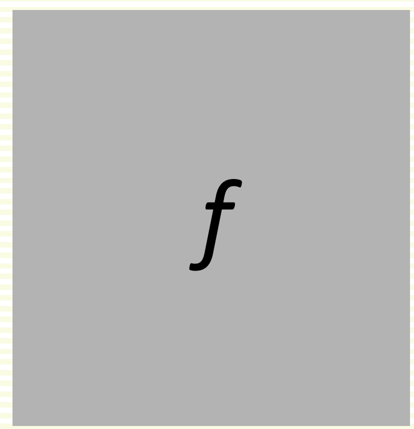


$$g(x,y)=\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]f(x-u,y-v)$$

H

H

flipped



- Notation for convolution: $g = H*f$

Convolution vs. Correlation

- Convolution: g = H*f

$$g(x,y)=\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]f(x-u,y-v)$$

- Correlation: g = $H \otimes f$

$$g(x,y)=\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]f(x+u,y+v)$$

- For Gaussian or box filter, how the outputs differ?
- If the input is an impulse signal, how the outputs differ?

Practice with Correlation Filtering

original

$\otimes$

0	0	0
0	1	0
0	0	0

= ?

Practice with Correlation Filtering

original

$$\otimes \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} =$$

filtered (no change)

Practice with Correlation Filtering

original

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

Practice with Correlation Filtering

original

$$\otimes \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} =$$

shifted left
by 1 pixel with
correlation

Practice with Correlation Filtering

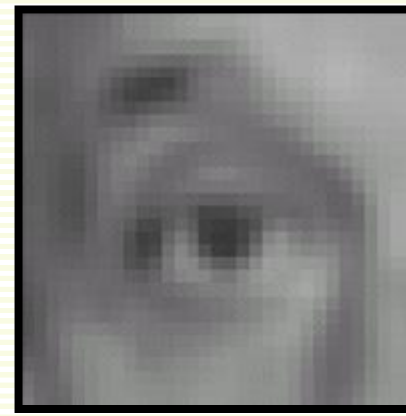
$$\otimes \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

Original

Practice with Correlation Filtering

original

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



blur (with a box filter)

Practice with Correlation Filtering

apply one mask
after the other,
or subtract masks
and apply one
resulting mask

$$\begin{bmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 17/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{bmatrix}$$

original

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

Practice with Correlation Filtering

original

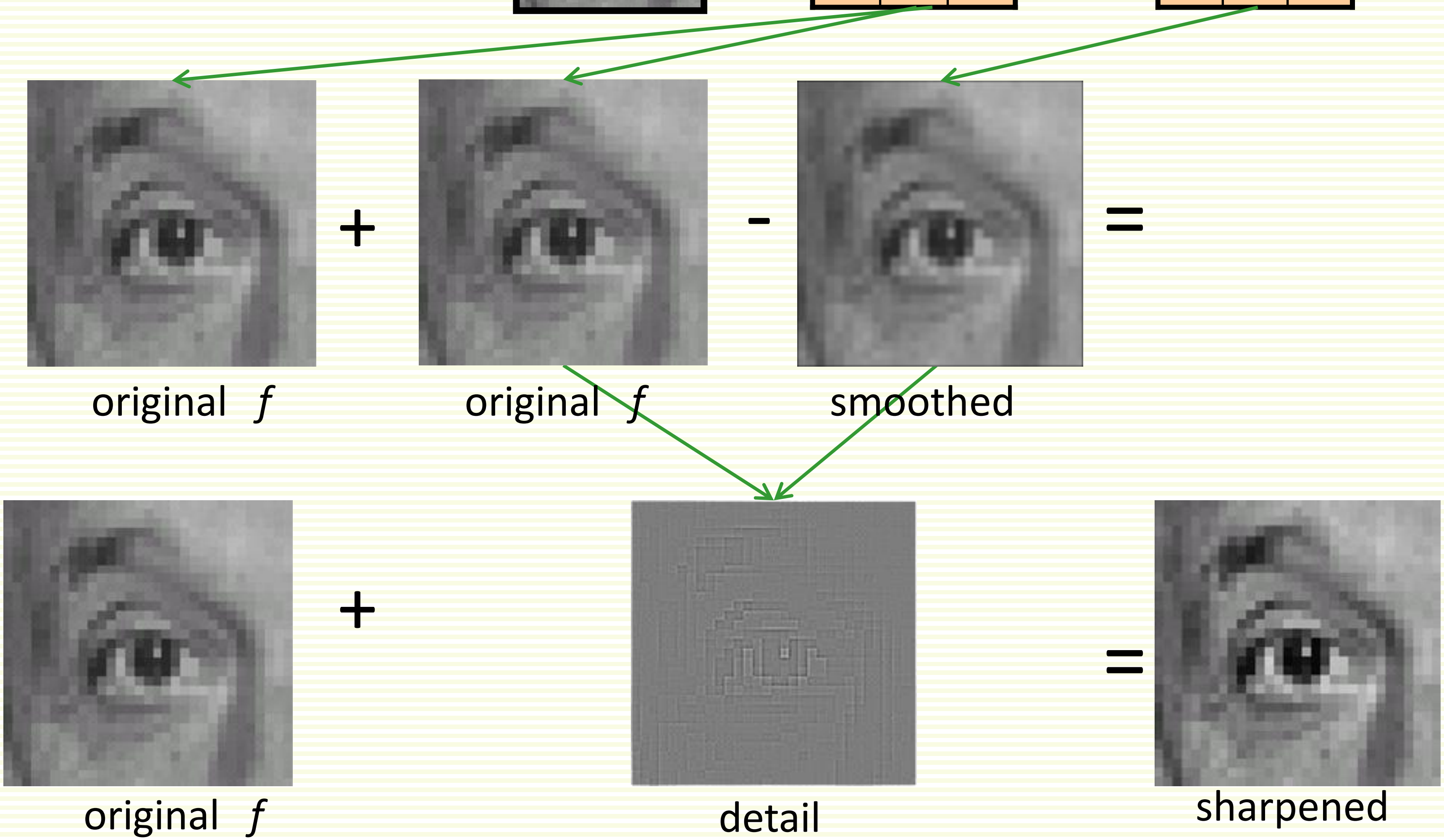
$$\otimes \begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} =$$

sharpened

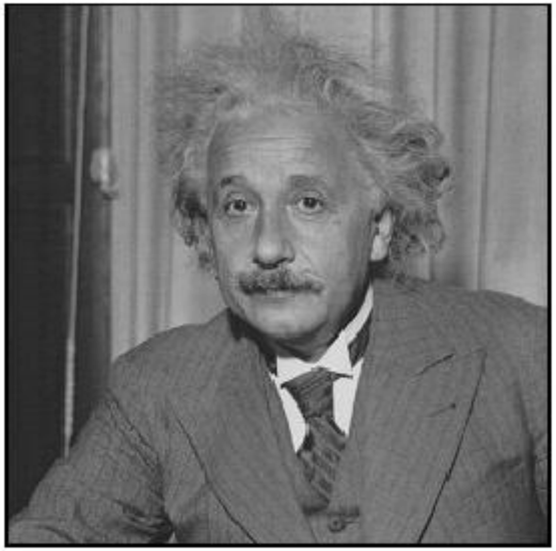
Practice with Correlation Filtering

- Why sharpens?

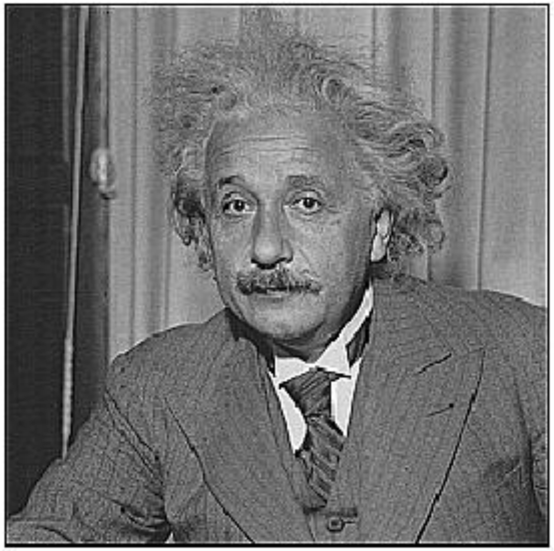
$$\otimes \begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$



Sharpening Example



before



after

Separability

- Sometimes filter is separable, can split into two steps:
  - Convolve all rows with 1D filter
  - Convolve all columns with 1D filter
- Both box and Gaussian filters are separable
- Great for efficiency!

Box Filter

$$\underbrace{\begin{array}{|c|c|c|}\hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array}}_{H} = \underbrace{\begin{array}{|c|}\hline 1/3 \\ \hline 1/3 \\ \hline 1/3 \\ \hline \end{array}}_{H_c} * \underbrace{\begin{array}{|c|c|c|}\hline 1/3 & 1/3 & 1/3 \\ \hline \end{array}}_{H_r}$$

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

$*H =$

0	0	0	0	0	0
0	40	60	60	40	0
0	60	90	90	60	0
0	60	90	90	60	0
0	40	60	60	40	0
0	0	0	0	0	0

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

$*H_c*H_r =$

0	0	0	0	0	0
0	60	60	60	60	0
0	90	90	90	90	0
0	90	90	90	90	0
0	60	60	60	60	0
0	0	0	0	0	0

$*H_r =$

0	0	0	0	0	0
0	40	60	60	40	0
0	60	90	90	60	0
0	60	90	90	60	0
0	40	60	60	40	0
0	0	0	0	0	0

Gaussian Filter: Example

- To convolve image with this:

$$\frac{1}{115}\begin{array}{|c|c|c|c|c|}\hline 2 & 4 & 5 & 4 & 2 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 5 & 12 & 15 & 12 & 5 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 2 & 4 & 5 & 4 & 2 \\ \hline \end{array}$$

H

- First convolve each row with:

$$\frac{1}{10.7}\begin{array}{|c|c|c|c|c|}\hline 1.3 & 3.2 & 3.8 & 3.2 & 1.3 \\ \hline \end{array}$$

H_r

- Then each column with:

$$\frac{1}{10.7}\begin{array}{|c|c|c|c|c|}\hline 1.3 & 3.2 & 3.8 & 3.2 & 1.3 \\ \hline \end{array}$$

H_c

Gaussian Filter: Example

- Straightforward convolution with 5×5 kernel
  - 25 multiplications, 24 additions per pixel
- Smart convolution
  - 10 multiplications, 9 additions per pixel
- Savings are even larger for larger kernels
  - for $n \times n$ kernel, straightforward convolution is $O(n^2)$
  - Smart convolution is $O(n)$ per pixel

Median Filters

1	2	25
3	24	22
20	21	23

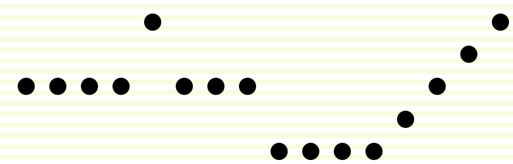
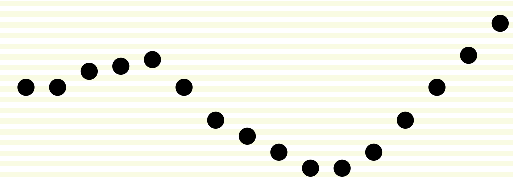
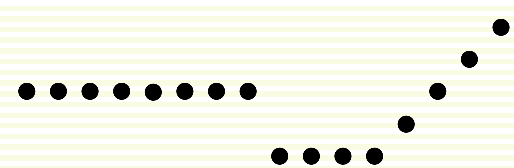
X	X	X
X	21	X
X	X	X

Median of {1,2,25,3,24,22,20,21,23} = {1,2,3,20,21,22,23,24,25} is 21

- A **Median Filter** selects median intensity in the window
- No new intensities are introduced
- Median filter preserves sharp details better than mean filter, it is not so prone to oversmoothing
- Better for salt and pepper, impulse (spiky) noise
- Is a median filter a kind of convolution?

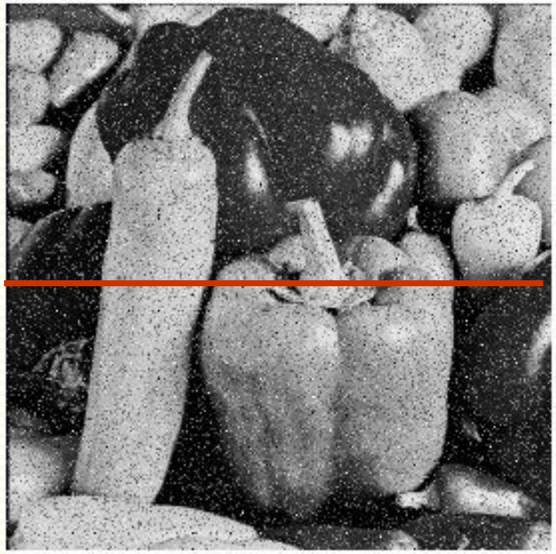
Median Filter

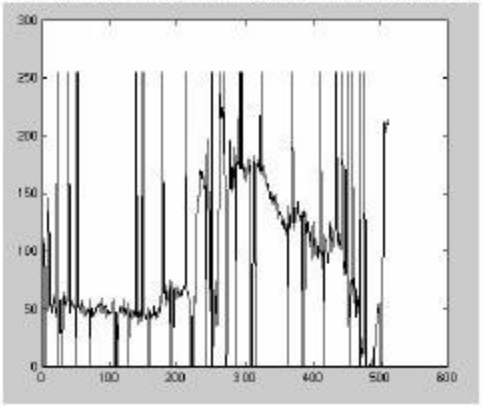
- Median filter is edge preserving

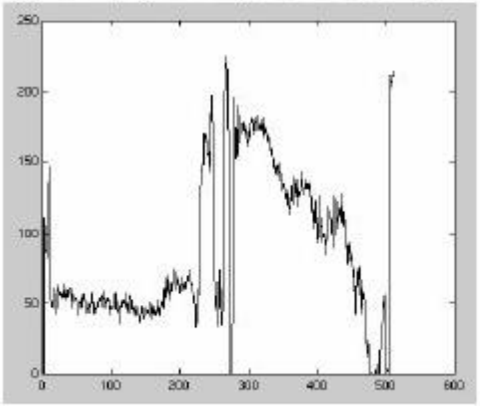
input:	
average:	
median:	

Median filter

Salt and pepper noise **median filtered**

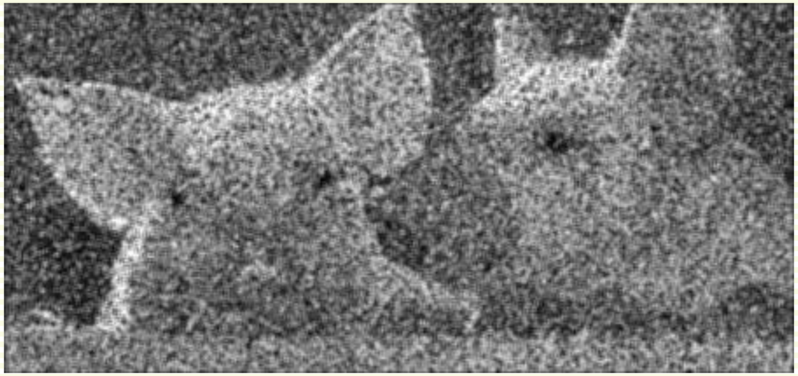
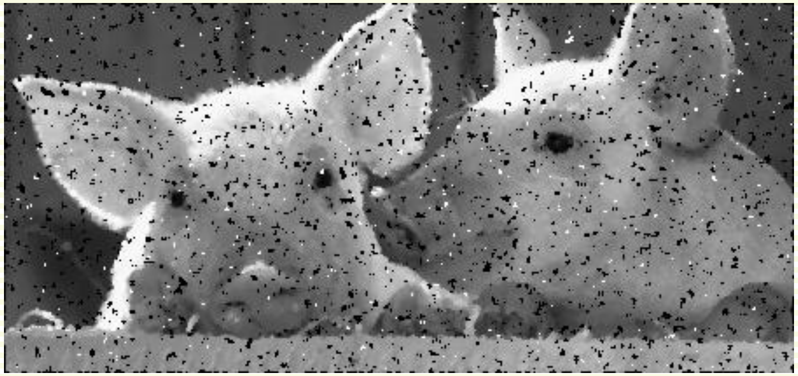
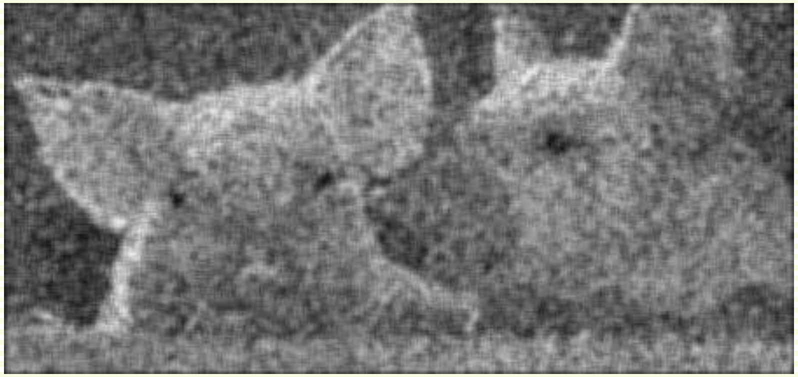
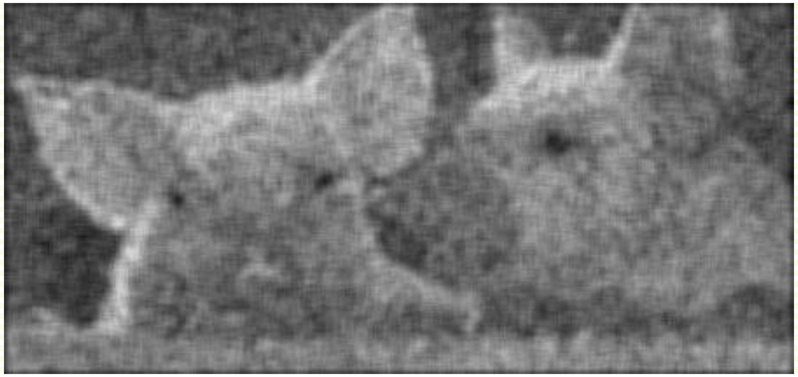






row of noisy image **row of filtered image**

Comparison: Salt and Pepper Noise Image

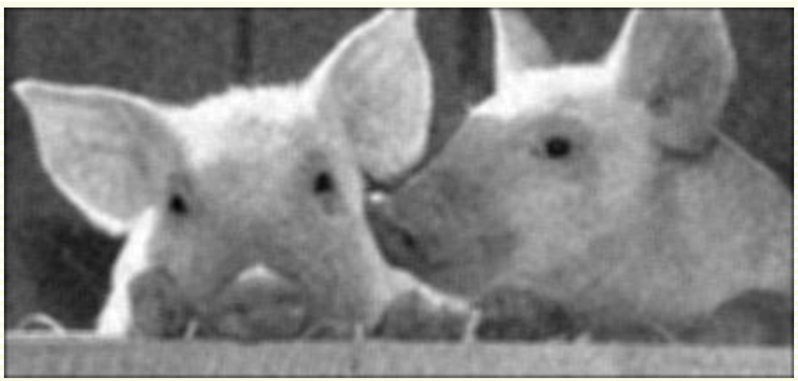
	Gaussian filter	median filter
3 × 3		
5 × 5		
7 × 7		

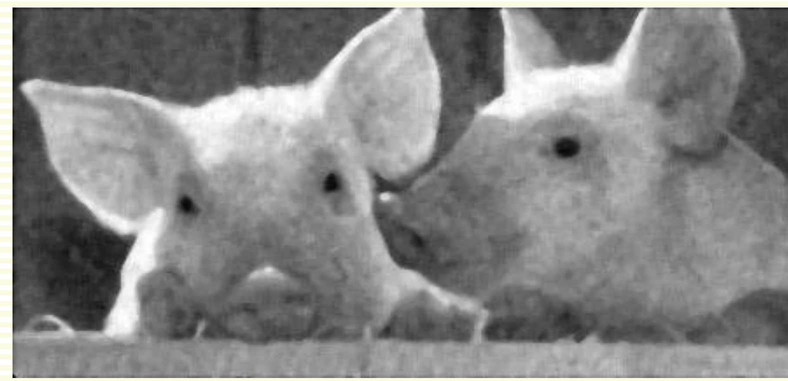
Comparison: Gaussian Noise Image

Gaussian filter | median filter

3 × 3

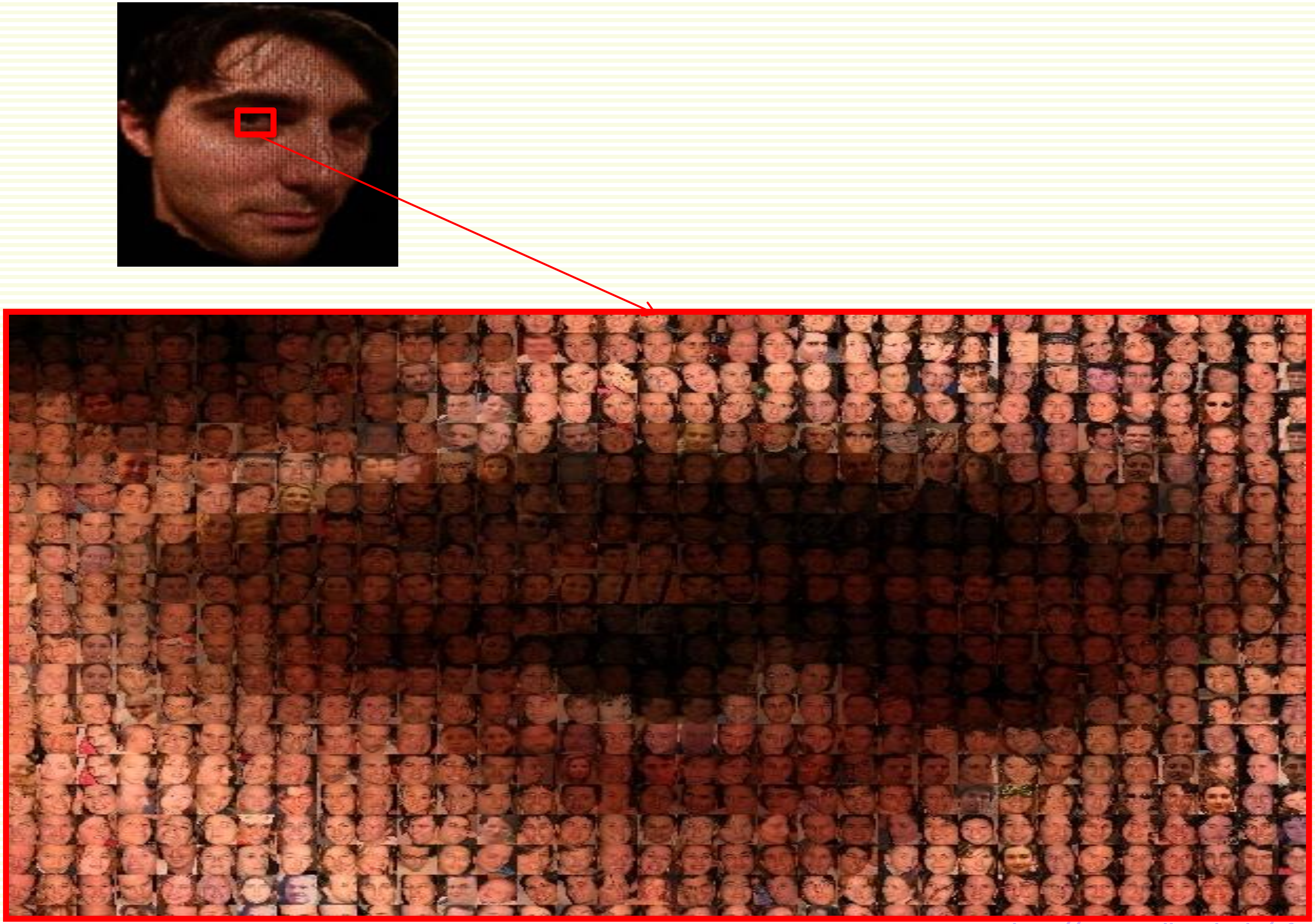
5 × 5





7 × 7

Filtering Fun: Face of Faces



http://www.salle.url.edu/~ftorre/

Salvador Dali, *“Gala Contemplating the Mediterranean Sea, which at 30 meters becomes the portrait of Abraham Lincoln*”, 1976

Summary

- Image “noise”
- Linear filters and convolution useful for
  - Enhancing images (smoothing, removing noise)
    - Box filter
    - Gaussian filter
    - Impact of scale / width of smoothing filter
  - Detecting features (next time)
- Separable filters more efficient
- Median filter: a non-linear filter, edge-preserving