Image Filtering

Readings: Ch 5: 5.4, 5.5, 5.6,5.7.3, 5.8
(This lecture does not follow the book.)

Images by Pawan Sinha

- formal terminology
- filtering with masks
  - mean filter
  - Gaussian filter
  - general cross-correlation
  - convolution
- median filter

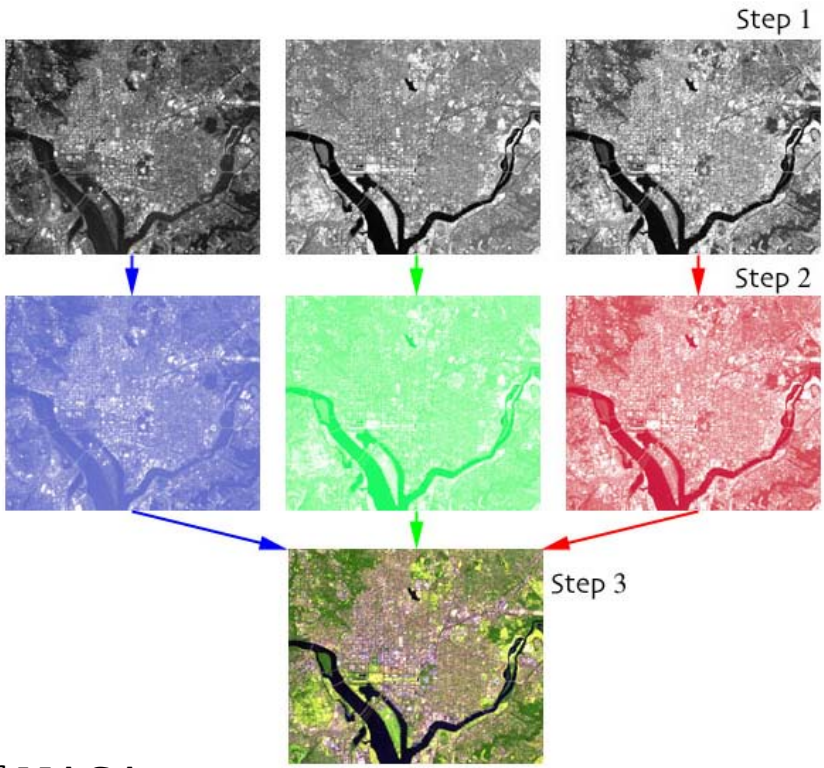
What is an image?

We can think of an **image** as a function, *f*, from $\mathbb{R}^2$ to $\mathbb{R}$:

- $f(x, y)$ gives the **intensity** at position (x, y).
- It is a continuous function, usually over a rectangle.

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

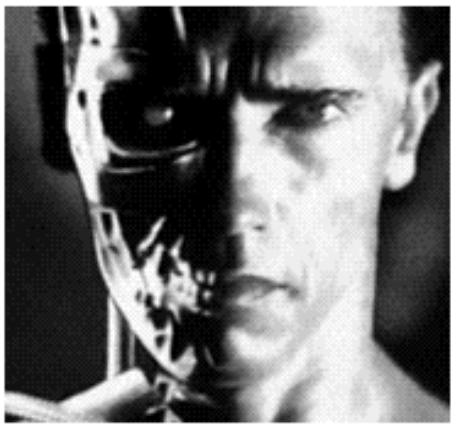
$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$

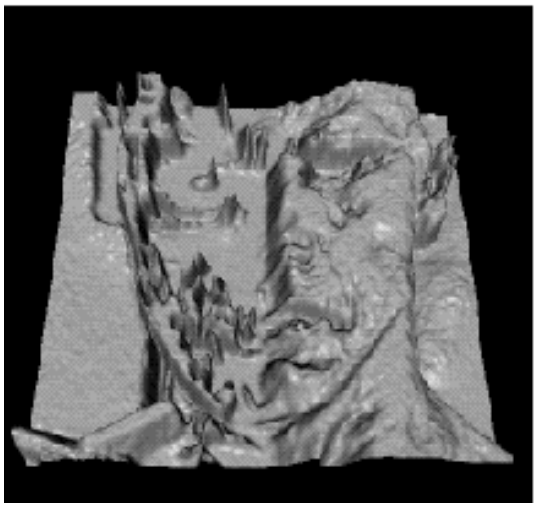


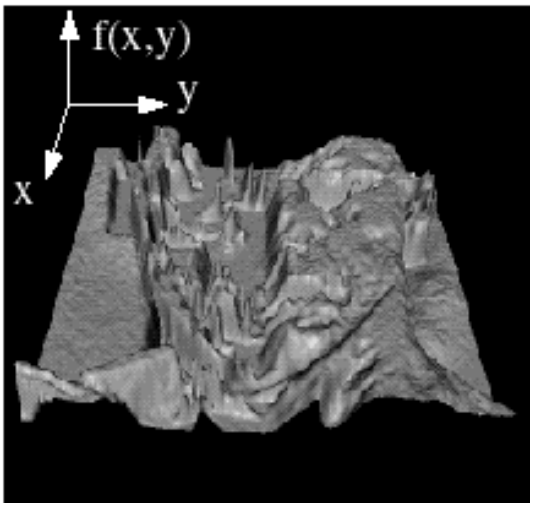
Courtesy of NASA

Images as functions

These are 4 different ways that people use to show gray tones, just for illustration.





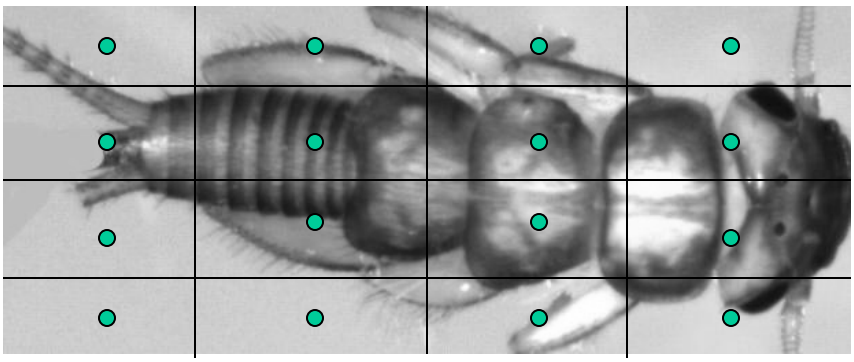


The 3D shows the “height” of each pixel.

Digital images

In computer vision we usually operate on **digital** (**discrete**) images:

- **Sample** the continuous 2D space on a regular grid.
- **Quantize** each sample by rounding to nearest integer.



The image can now be represented as a matrix of integer values.

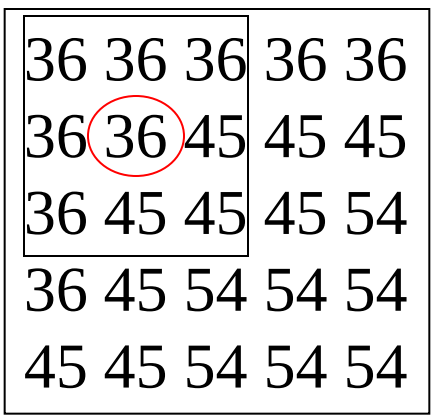
j

i

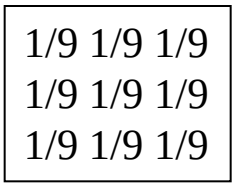
62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Filtering Operations Use Masks

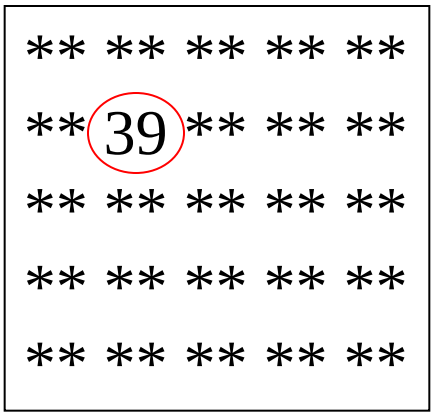
- **Masks operate on a neighborhood of pixels.**
- **A mask of coefficients is centered on a pixel.**
- **The mask coefficients are multiplied by the pixel values in its neighborhood and the products are summed.**
- **The result goes into the corresponding pixel position in the output image.**



Input Image



3x3 Mask



Output Image

Application: Noise Filtering

Image processing is useful for noise reduction...

Original

Salt and pepper noise

Impulse noise

Gaussian noise

Common types of noise:

- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Practical noise reduction

How can we “smooth” away noise in a single image?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	100	130	110	120	110	0	0
0	0	0	110	90	100	90	100	0	0
0	0	0	130	100	90	130	110	0	0
0	0	0	120	100	130	110	120	0	0
0	0	0	90	110	80	120	100	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Mean filtering

$F[x, y]$

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

$G[x, y]$

Mean filtering

$F[x, y]$

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

Input Image

Sum the values in a 3x3 nbd. Divide by 9. Replace center.

$G[x, y]$

	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	

Output Image

Effect of mean filters

Generalization: Cross-Correlation Filtering

Let's write this down as an equation. Assume the averaging window is (2k+1)x(2k+1):

$$G[i,j] = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]$$

We can generalize this idea by allowing different weights for different neighboring pixels:

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

This is called a **cross-correlation** operation and written:

$$G = H \otimes F$$

H is called the "filter," "kernel," or "mask."

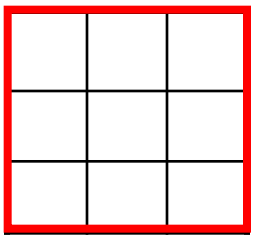
Mean Kernel

What's the kernel for a 3x3 mean filter?

In other words, what do you multiply each pixel by when you find the mean?

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

$F[x, y]$



$H[u, v]$

Mean Kernel

What's the kernel for a 3x3 mean filter?

In other words, what do you multiply each pixel by when you find the mean?

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

$F[x, y]$

mean filter

$1/9$

1	1	1
1	1	1
1	1	1

$H[u, v]$

Gaussian Filtering

A Gaussian kernel gives less weight to pixels further from the center of the window

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

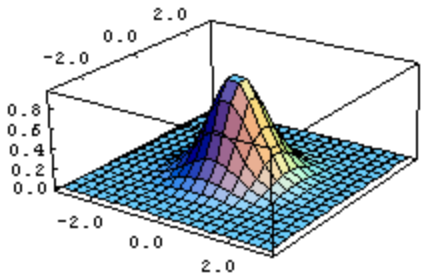
$F[x, y]$

$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

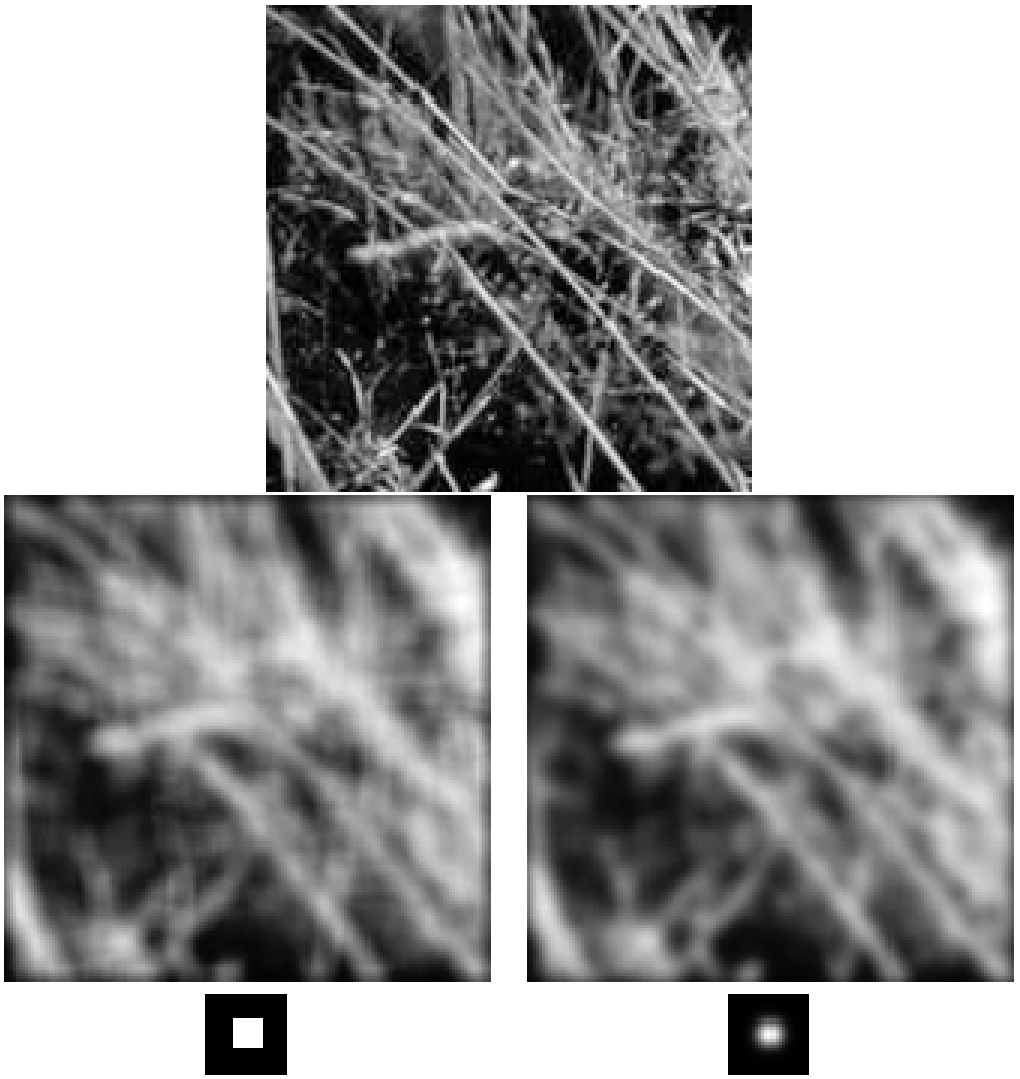
$H[u, v]$

This kernel is an approximation of a Gaussian function:

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



Mean vs. Gaussian filtering



Convolution

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

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

It is written: $G = H \star F$

Suppose H is a Gaussian or mean kernel. How does convolution differ from cross-correlation?

In computer vision, we tend to use symmetric kernels most of the time, and we tend to call them convolution kernels.

In EE, convolution is useful for solving linear systems problems.

Convolution vs. Correlation
When do they differ?
1D Example from Signal Processing

Correlation

Mask: w = 1 2 3 2 0
Signal: 0 0 0 1 0 0 0 0
Zero padding
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Apply mask to first position
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 0
Produces a 0 for first out

Apply mask to 4th position
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 0
Produces a 0

Final result:
0 0 0 0 2 3 2 1 0 0 0 0
Remove padding:
0 0 2 3 2 1 0 0

Convolution

Mask: w' = 0 2 3 2 1
Signal: 0 0 0 1 0 0 0 0
Zero padding
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Apply mask to first position
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 2 3 2 1
Produces a 0 for first out

Apply mask to 4th position
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 2 3 2 1
Produces a 1

Final results:
0 0 0 1 2 3 2 0 0 0 0 0
Remove padding:
0 1 2 3 2 0 0 0

Convolution Examples

Simple box blur

Here's a first and simplest. This convolution kernel has an averaging effect. So you end up with a slight blur. The image convolution kernel is:

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

Note that the sum of all elements of this matrix is 1.0. This is important. If the sum is not exactly one, the resultant image will be brighter or darker.

Here's a blur that I got on an image:

A simple blur done with convolutions

http://www.aishack.in/2010/08/image-convolution-examples/

Convolution Examples

Gaussian blur

Gaussian blur has certain mathematical properties that makes it important for computer vision. And you can approximate it with an image convolution. The image convolution kernel for a Gaussian blur is:

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

Here's a result that I got:

Convolution Examples

Line detection with image convolutions

With image convolutions, you can easily detect lines. Here are four convolutions to detect horizontal, vertical and lines at 45 degrees:

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

Vertical lines

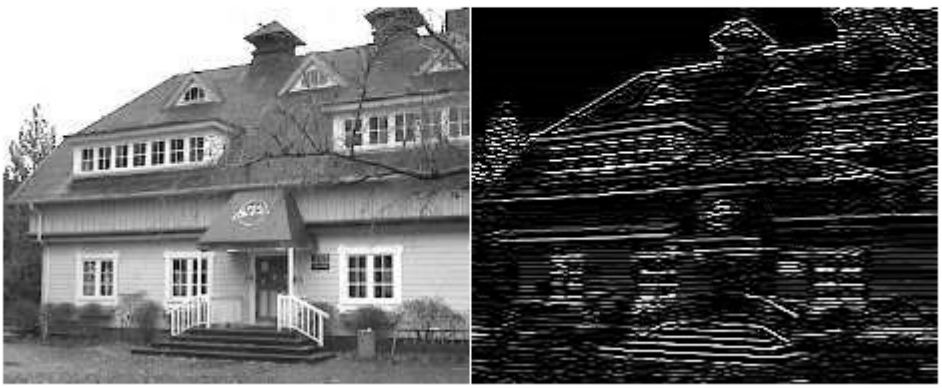
-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

I looked for horizontal lines on the house image. The result I got for this image convolution was:



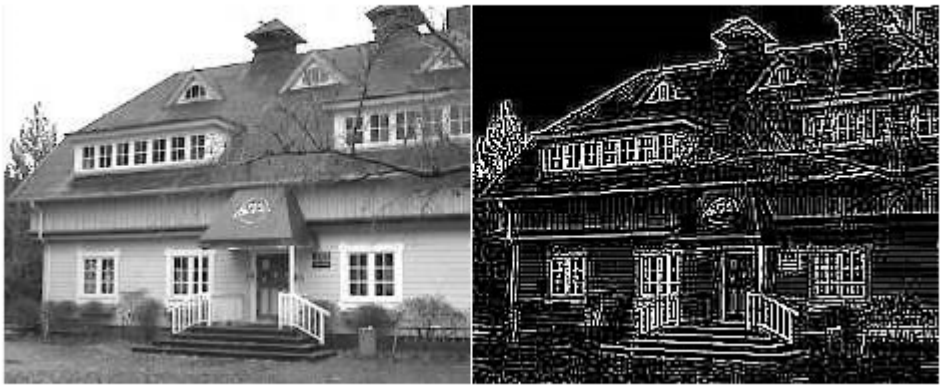
Convolution Examples

Edge detection

The above kernels are in a way edge detectors. Only thing is that they have separate components for horizontal and vertical lines. A way to "combine" the results is to merge the convolution kernels. The new image convolution kernel looks like this:

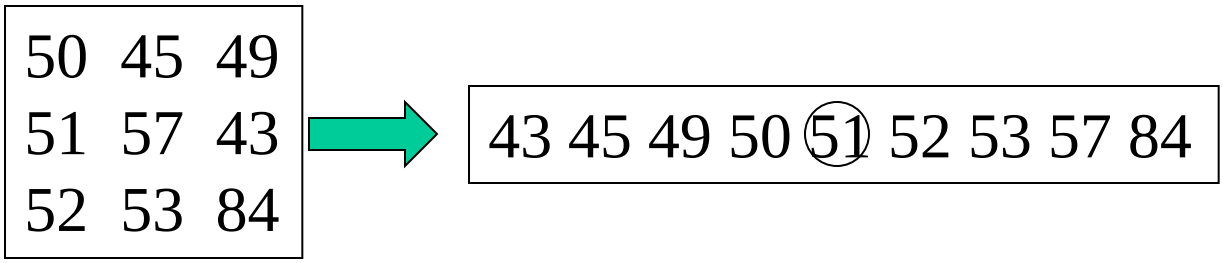
-1	-1	-1
-1	8	-1
-1	-1	-1

Below result I got with edge detection:



Median filters

A **Median Filter** operates over a window by selecting the median intensity in the window.

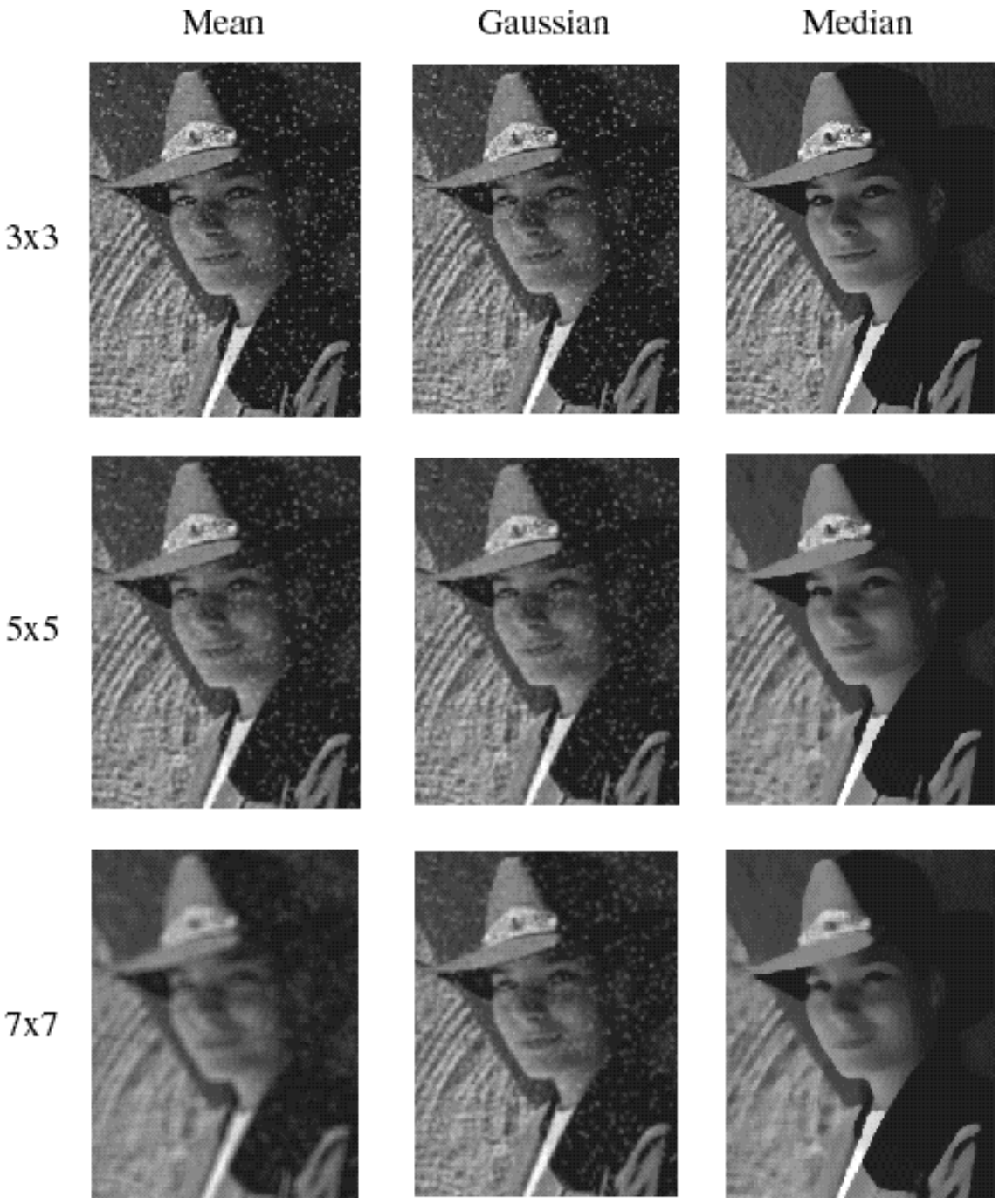


What advantage does a median filter have over a mean filter?

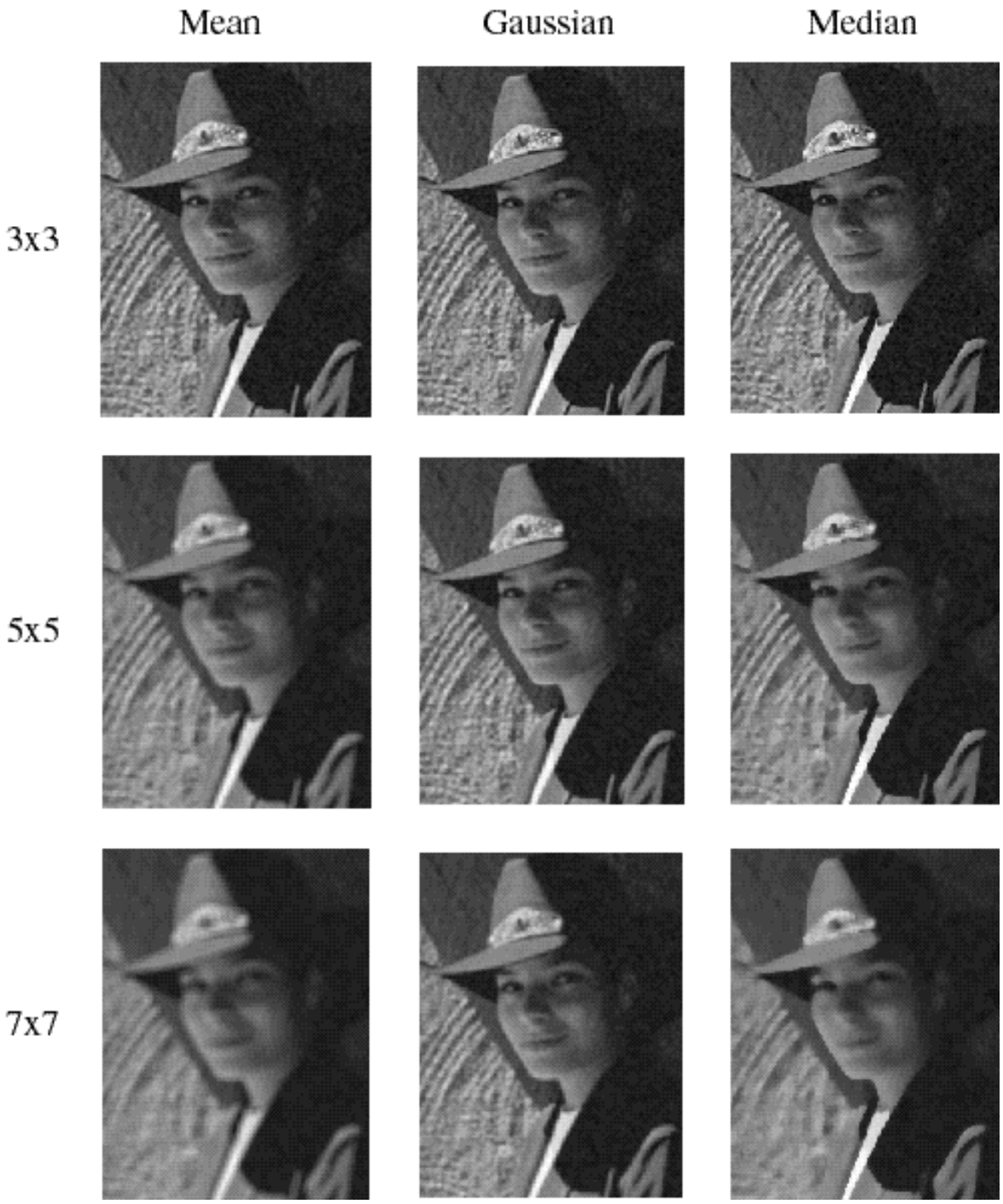
Is a median filter a kind of convolution?

No, it's called a nonlinear filter.

Comparison: salt and pepper noise



Comparison: Gaussian noise



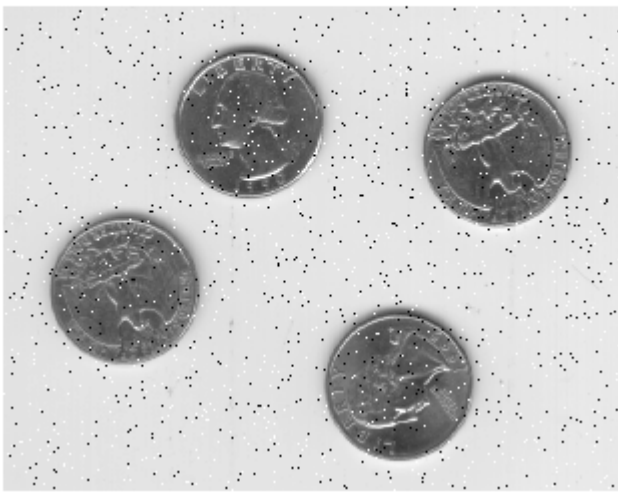
More Comparisons:
Mean vs. Median using Matlab

1. Read in the image and display it.

```
I = imread('eight.tif');
imshow(I)
```

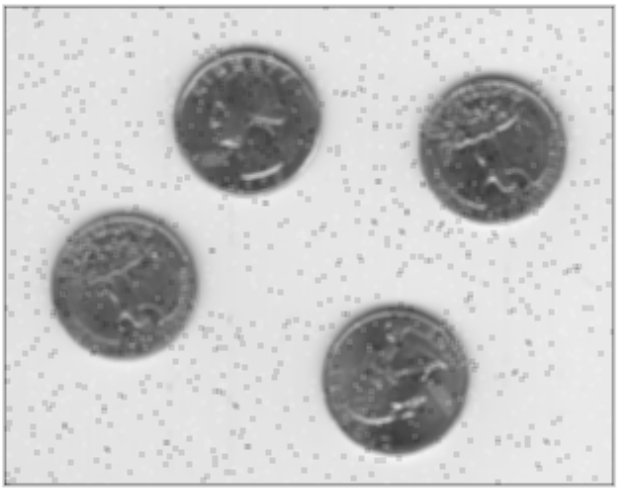
2. Add noise to it.

```
J = imnoise(I,'salt & pepper',0.02);
figure, imshow(J)
```



3. Filter the noisy image with an averaging filter and display th

```
K = filter2(fspecial('average',3),J)/255;
figure, imshow(K)
```



4. Now use a median filter to filter the noisy image and display of edges.

```
L = medfilt2(J,[3 3]);
figure, imshow(L)
```

http://www.mathworks.com/help/toolbox/images/f11-12251.html

Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

Problems:

- neighborhood size
- how to detect change

81	82	26	24
82	33	25	25
81	82	26	24

Differential Operators

Differential operators

- attempt to approximate the gradient at a pixel via masks
- threshold the gradient to select the edge pixels

What's a gradient?

Def: the gradient of a scalar function $f(x_1, x_2, \ldots x_n)$ is denoted by $\nabla \mathbf{f}$ (del f) and is defined by:

$$\nabla \mathbf{f} = (\partial \mathbf{f}/\partial \mathbf{x_1}, \partial \mathbf{f}/\partial \mathbf{x_2}, \ldots \partial \mathbf{f}/\partial \mathbf{x_n})$$

What's a derivative?

Example: Sobel Operator

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

On a pixel of the image I

- let g_x be the response to mask S_x
- let g_y be the response to mask S_y

Then the gradient is

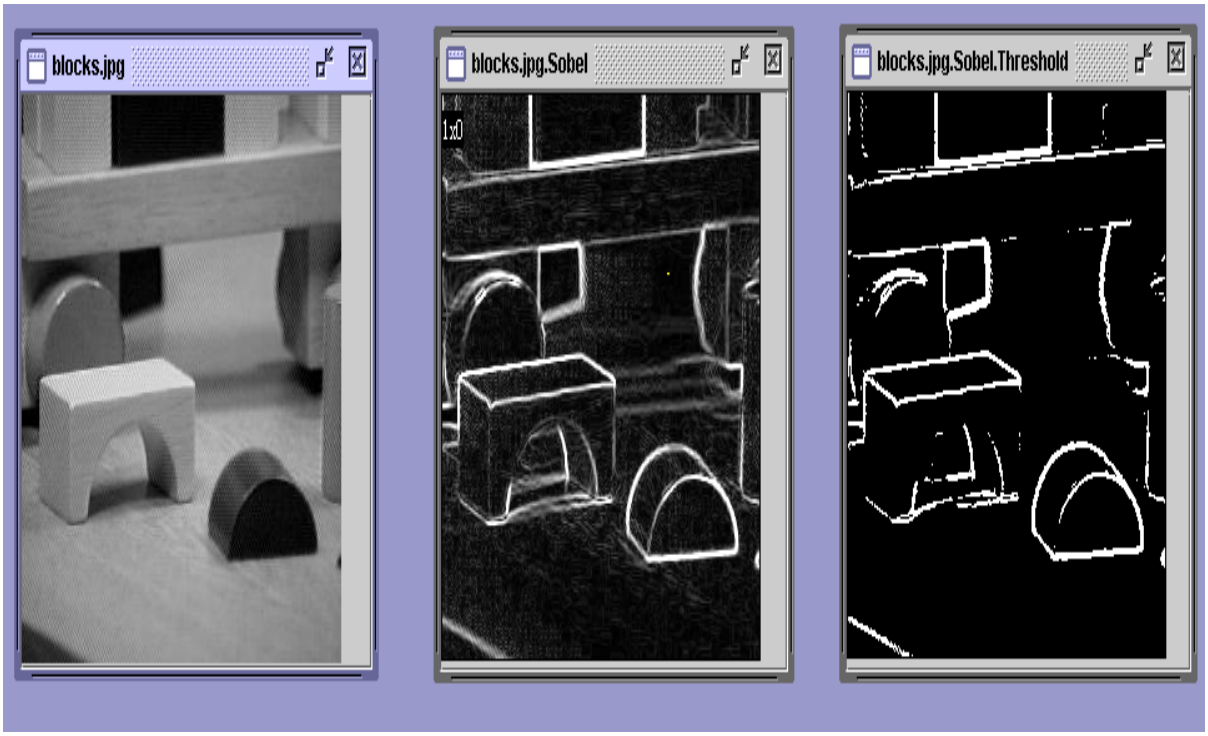
$\nabla I = [g_x \;\; gy]^T$

$g = (g_x^{\,2} + g_y^{\,2})^{1/2}$ is the gradient magnitude.

$\theta = \text{atan2}(g_y, g_x)$ is the gradient direction.

28

Sobel Operator on Blocks Image



original image

gradient magnitude

thresholded gradient magnitude

Some Well-Known Masks for Computing Gradients

	Sx	Sy
Sobel:	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
Prewitt:	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Roberts	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$