CSCI 631
Foundations of Computer Vision

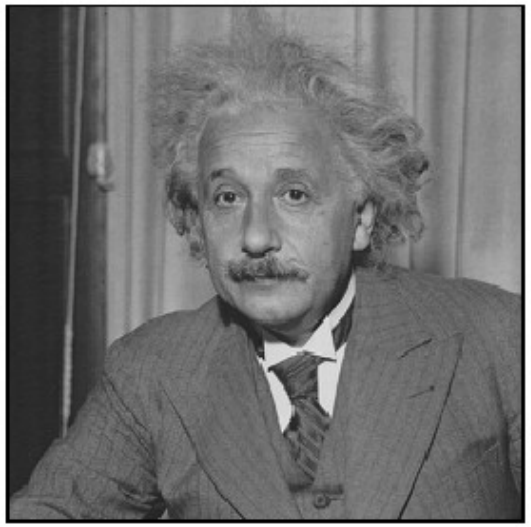
Ifeoma Nwogu

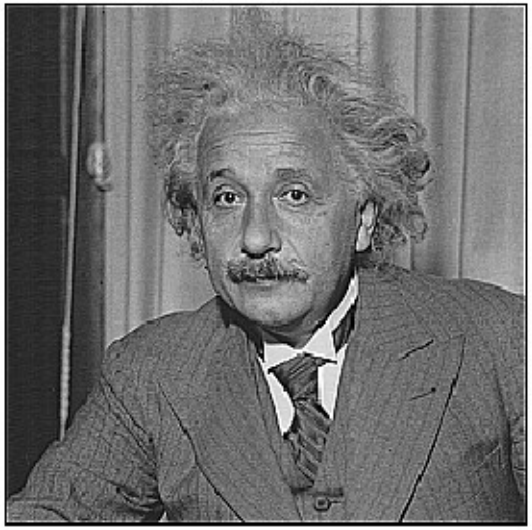
ion@cs.rit.edu

Schedule

- Last class
  - Introduction
- Today
  - Linear filters and convolution

Image Filters





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

Overview

- Images as functions
- Linear filters
- Convolution and correlation

Overview

- Images as functions
- Linear filters
- Convolution and correlation

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: $[a,b]$ x $[c,d]$ → $[0, 1.0]$

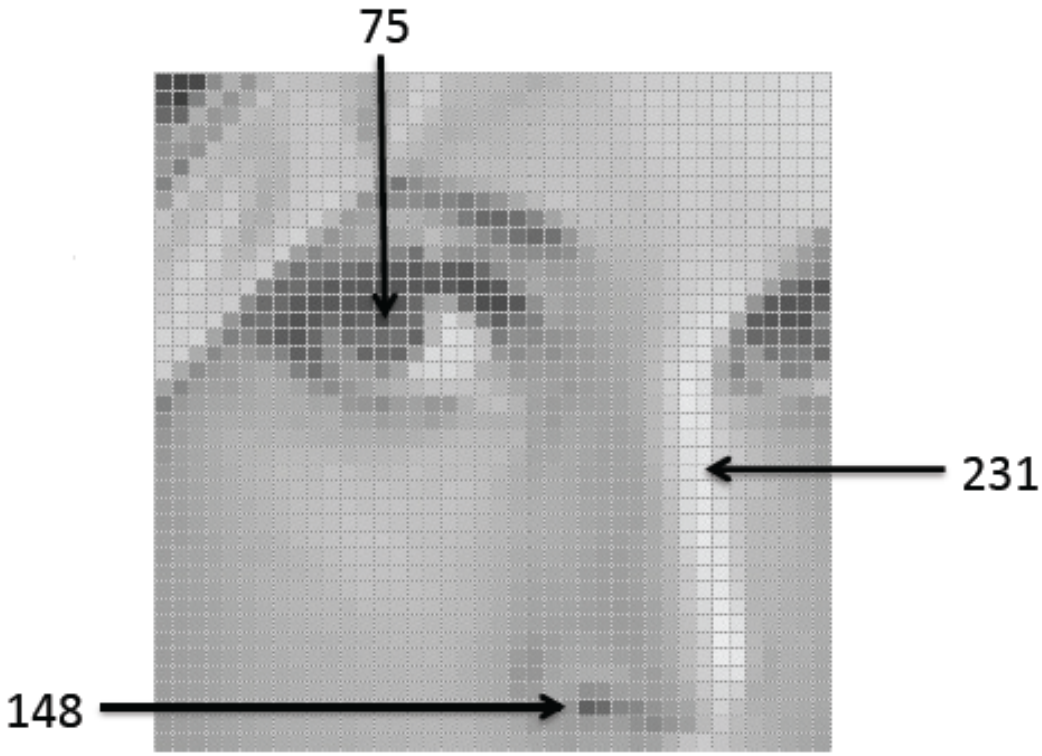
- A color image is just three functions pasted together. We can write this as a "vector-valued" function:

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

Source: S. Seitz

Images as functions

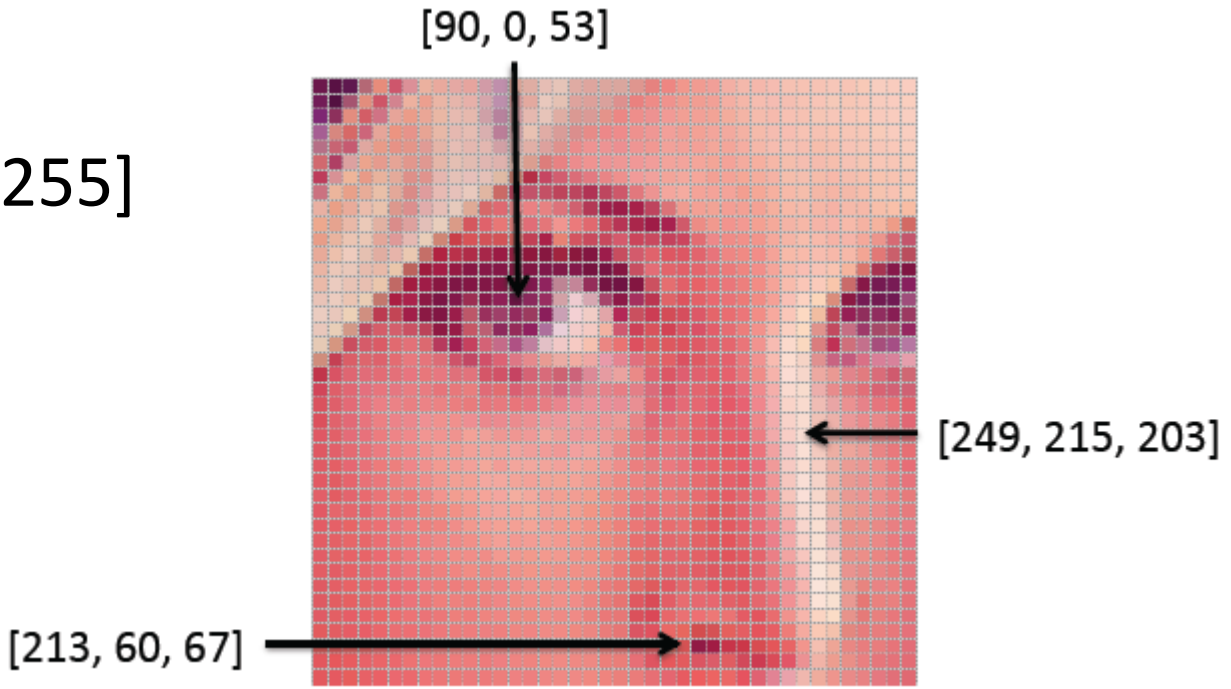
- An image contains discrete number of pixels, e.g.
  - "grayscale" (or "intensity"): [0,255]



Source: F.F Lei

Images as functions

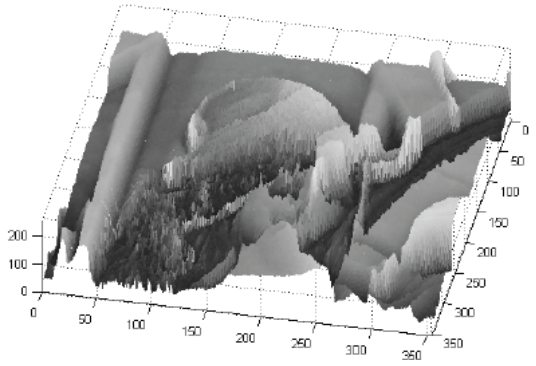
- An image contains discrete number of pixels, e.g.
  - "grayscale" (or "intensity"): [0,255]
  - Color
    - RGB: [R, G, B]
    - Lab: [L, a, b]
    - HSV: [H, S, V]



Source: F.F Lei

Images as discrete functions

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

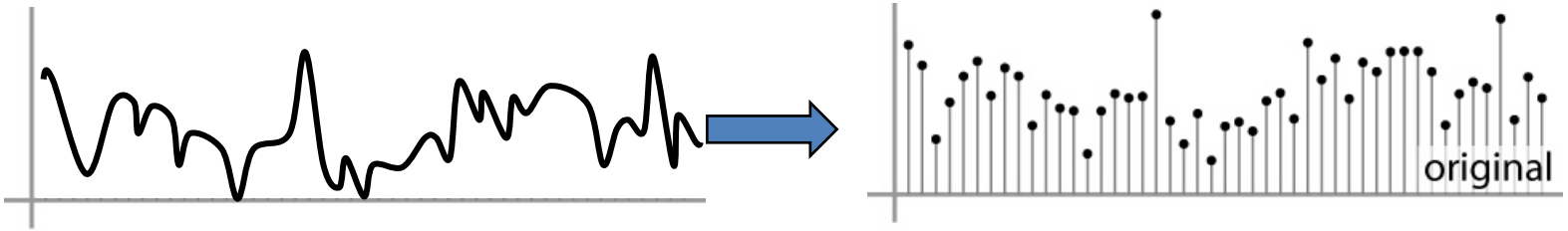


j

i

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

2D



1D

Images as discrete functions

- Cartesian coordinates

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

Notation for discrete functions

Overview

- Images as functions
- Linear filters
- Convolution and correlation

Motivation

Filtering:

- Form a new image whose pixels are a combination original pixel values

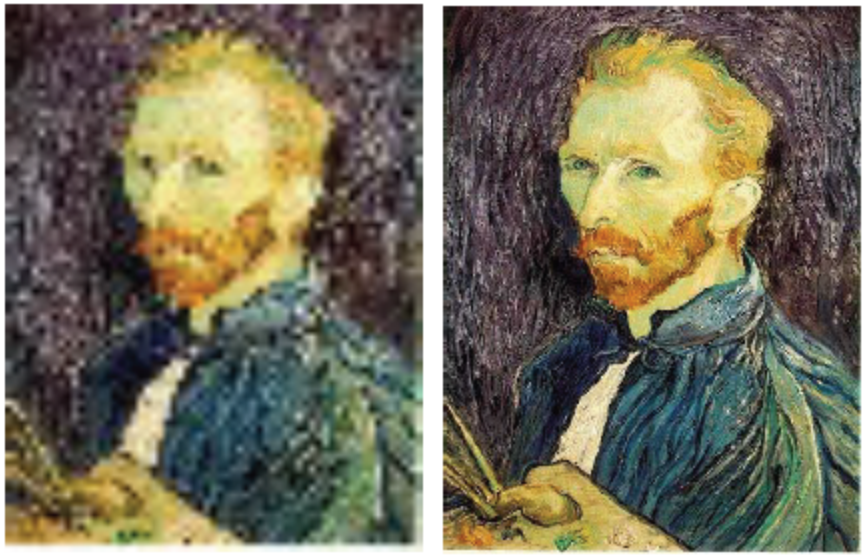
Goals:

- Extract useful information from the images
  - Features (edges, corners, blobs…)
- Modify or enhance image properties:
  - super-resolution; in-painting; de-noising

De-noising

Salt and pepper noise

Super-resolution

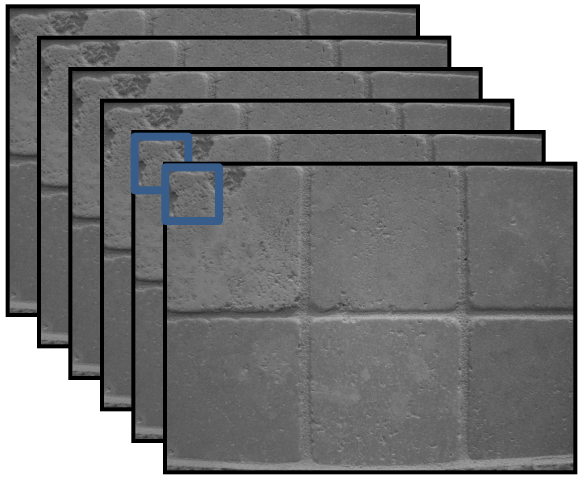


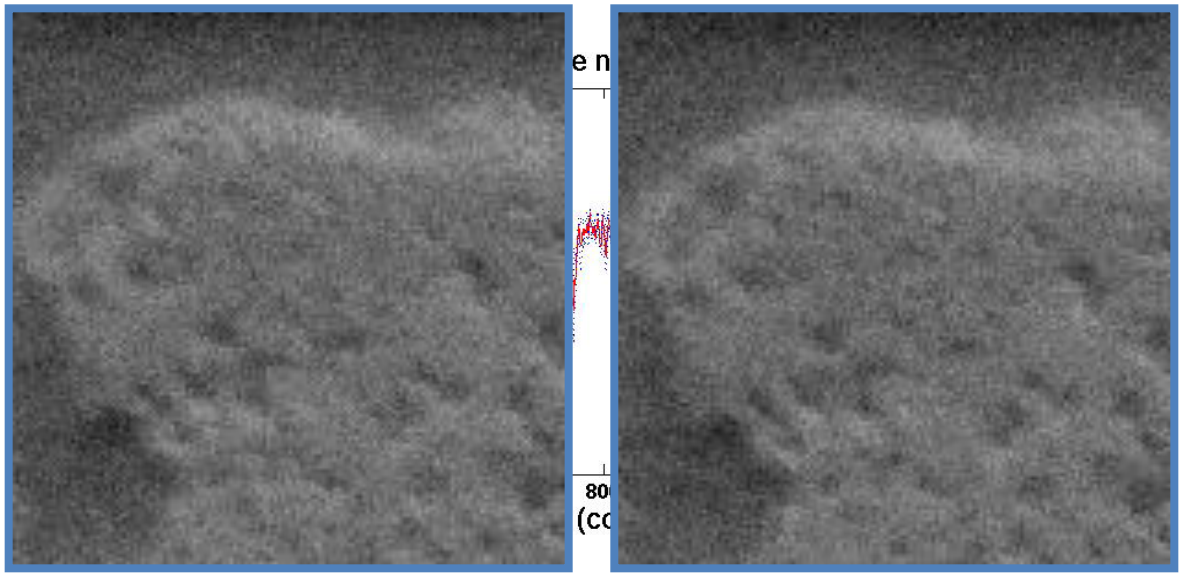
In-painting

Bertamio et al

Source: F.F. Li

Motivation: noise reduction





- We can measure **noise** in multiple images of the same static scene.
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

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

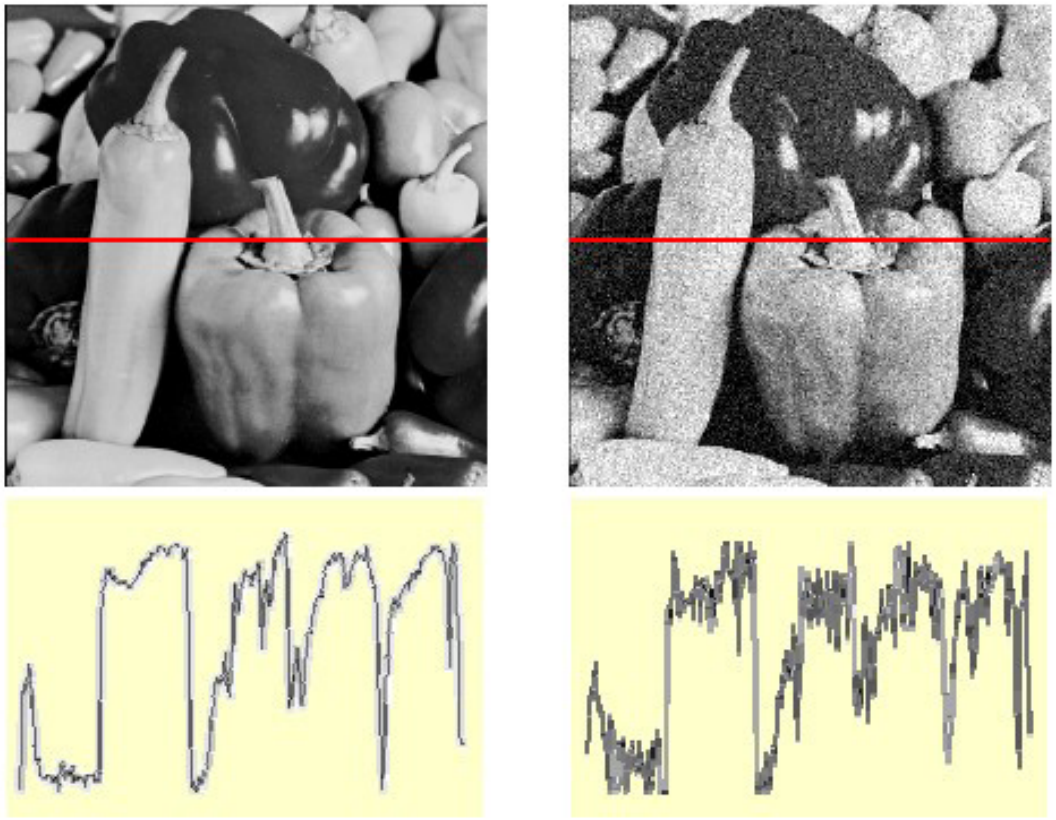
Salt and pepper noise

Impulse noise

Gaussian noise

Source: S. Seitz

Gaussian noise



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

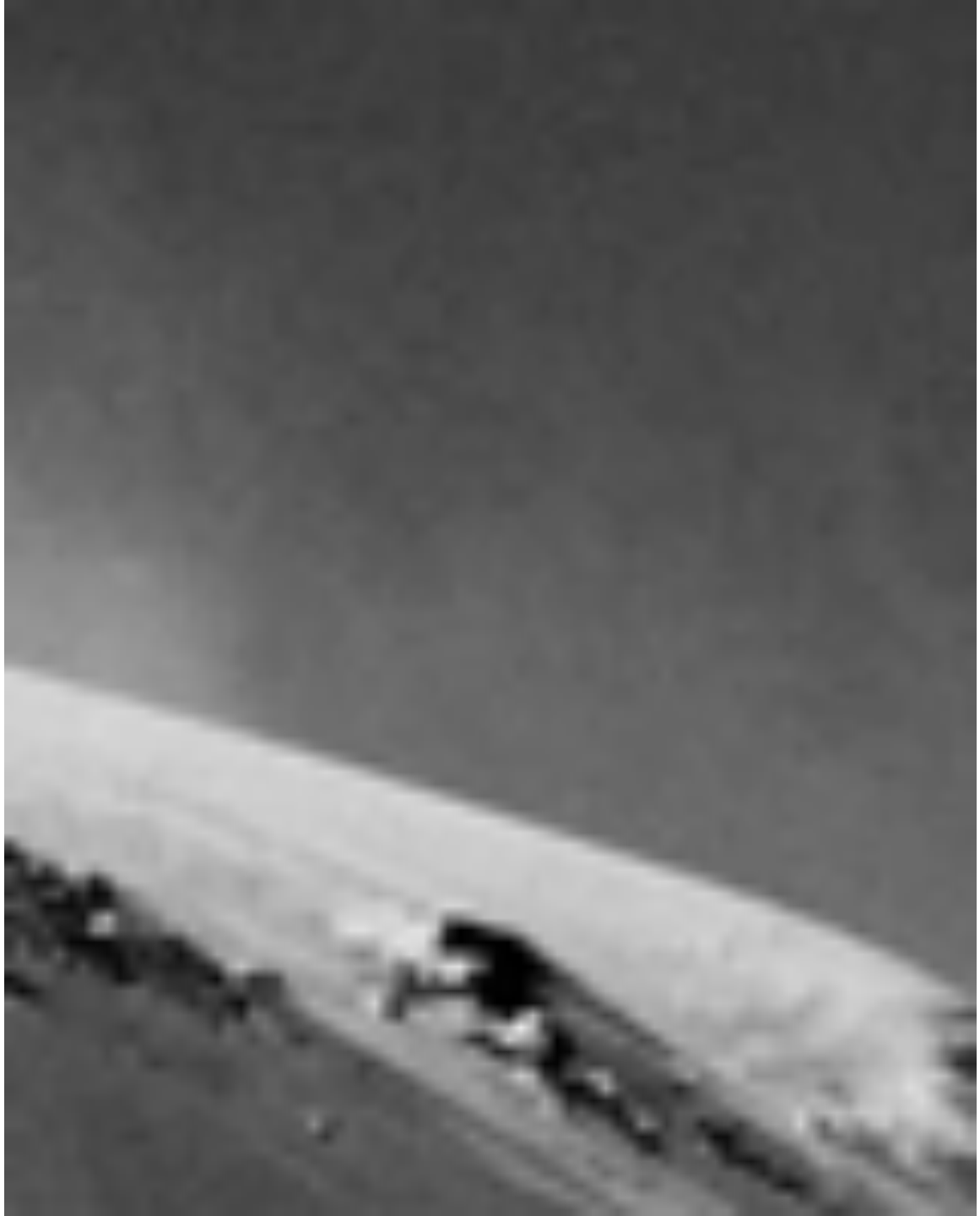
Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$

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

>> output = im + noise;
```


Fig: M. Hebert

sigma=1



Effect of sigma on Gaussian noise:

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

sigma=16

Effect of sigma on Gaussian noise

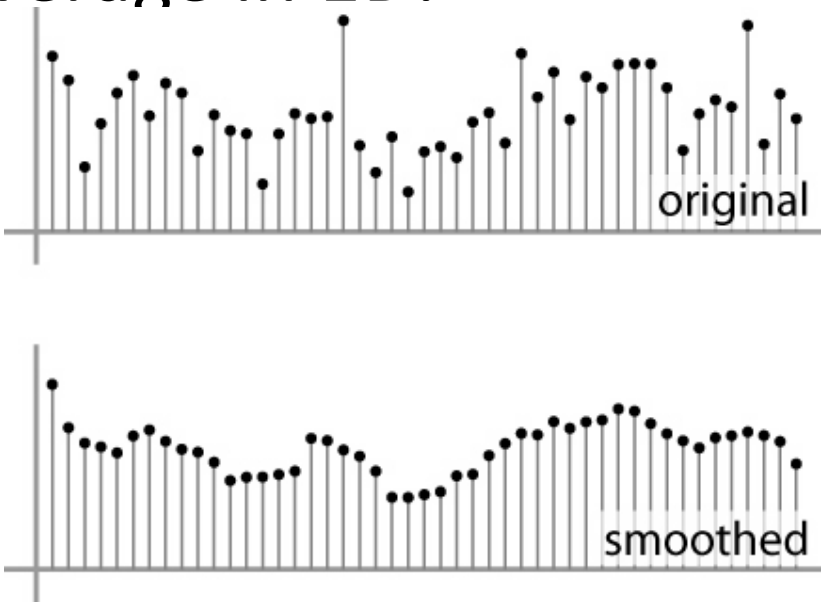
This shows the noise values added to the raw intensities of an image.

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

First attempt at a solution

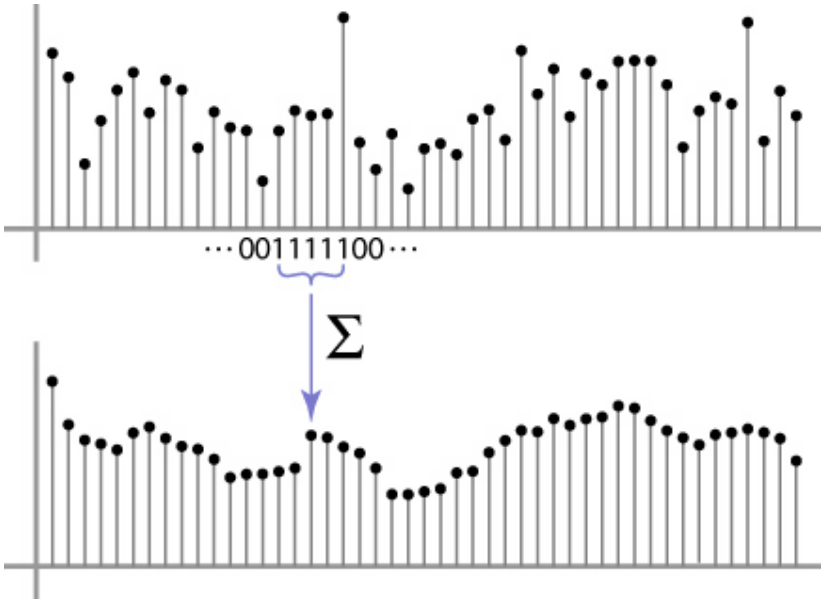
- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



Source: S. Marschner

Weighted Moving Average

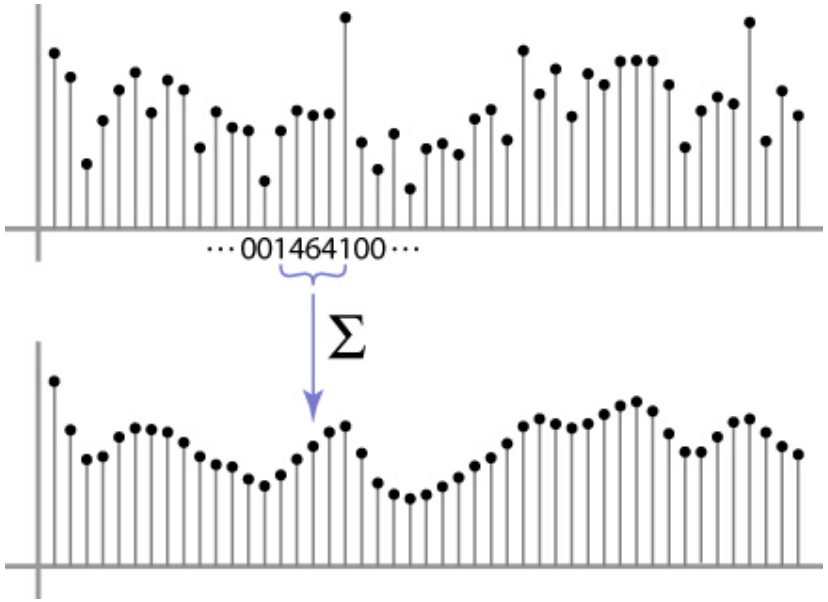
- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5



Source: S. Marschner

Weighted Moving Average

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



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: 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: 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: 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: 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: S. Seitz

Correlation filtering

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

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

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

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

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: S. Seitz

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.

Averaging filter

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

$F[x, y]$ $\otimes$ $H[u, v]$ $G[x, y]$

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

$\frac{1}{9}$

1	1	1
1	?	1
1	1	1

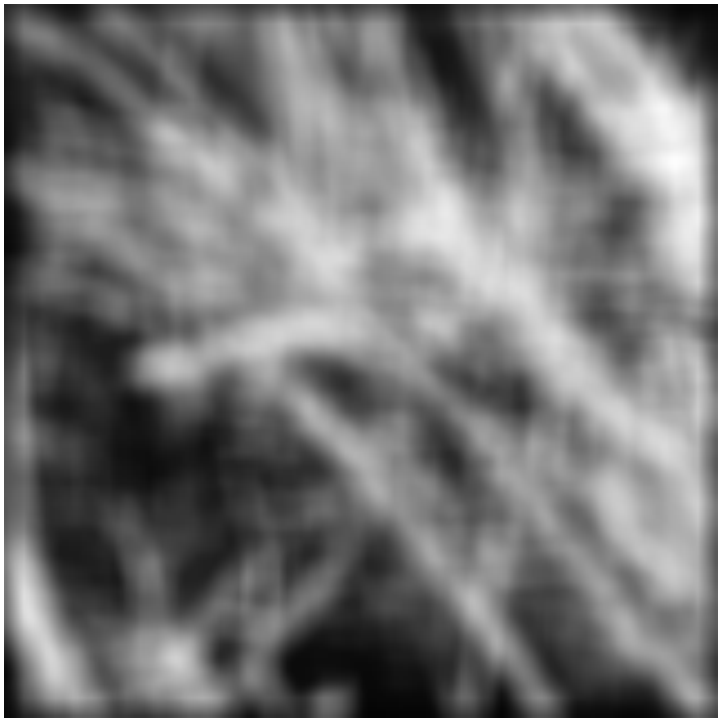
"box filter"

	0	10	20	30	30				

$$G = H \otimes F$$

Smoothing by averaging

original



filtered

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

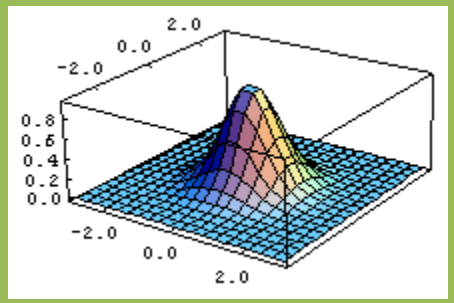
$F[x, y]$

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

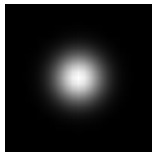
1	2	1
2	4	2
1	2	1

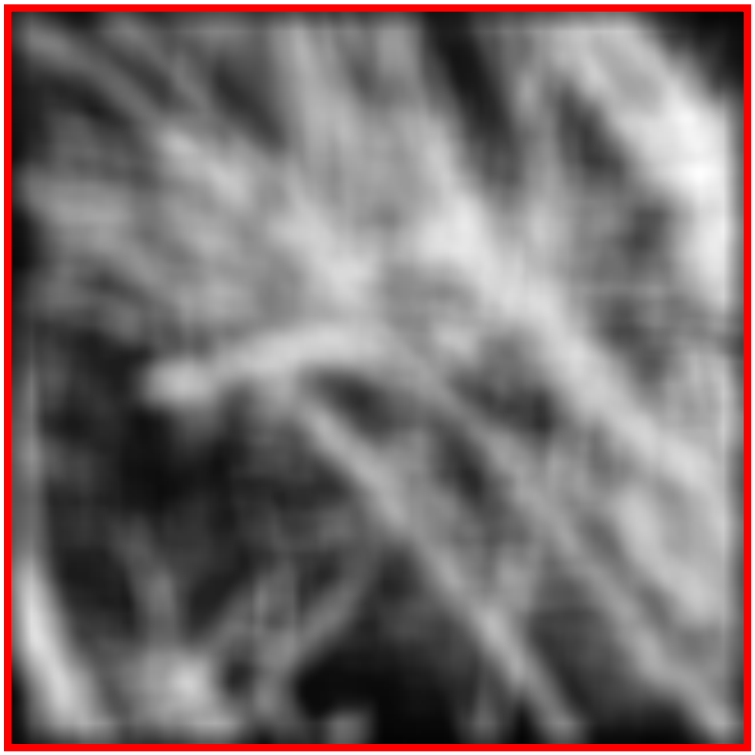
$H[u, v]$

This kernel is an approximation of a Gaussian function:



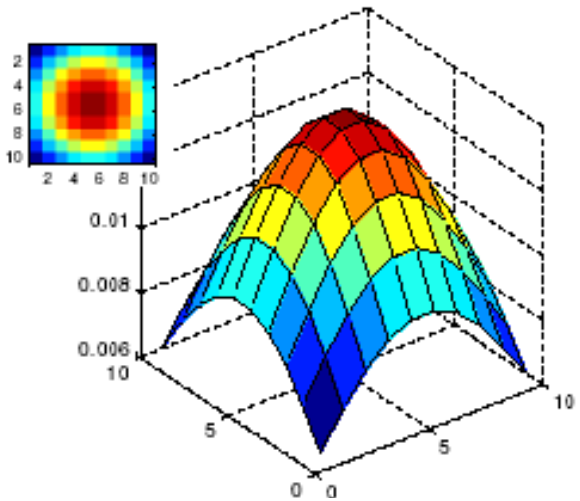
Smoothing with a Gaussian



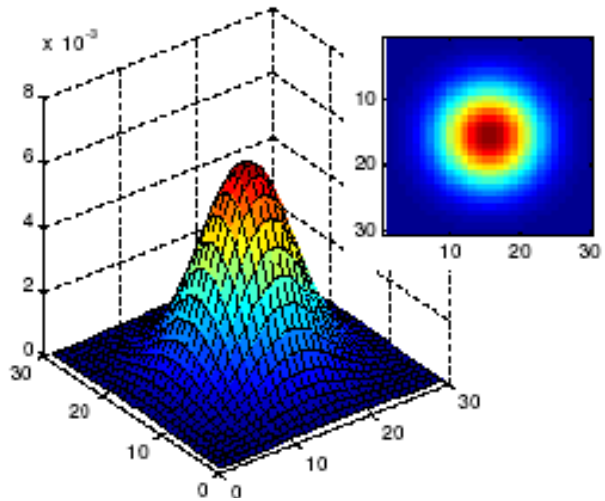


Gaussian filters

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



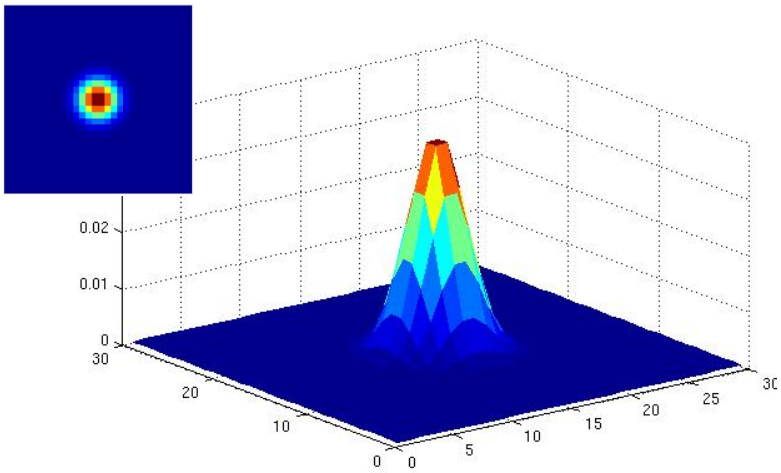
σ = 5 with 10 x 10 kernel



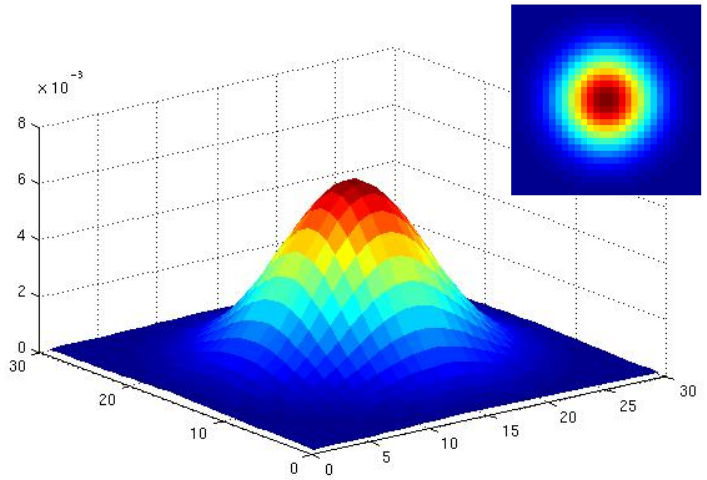
σ = 5 with 30 x 30 kernel

Gaussian filters

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



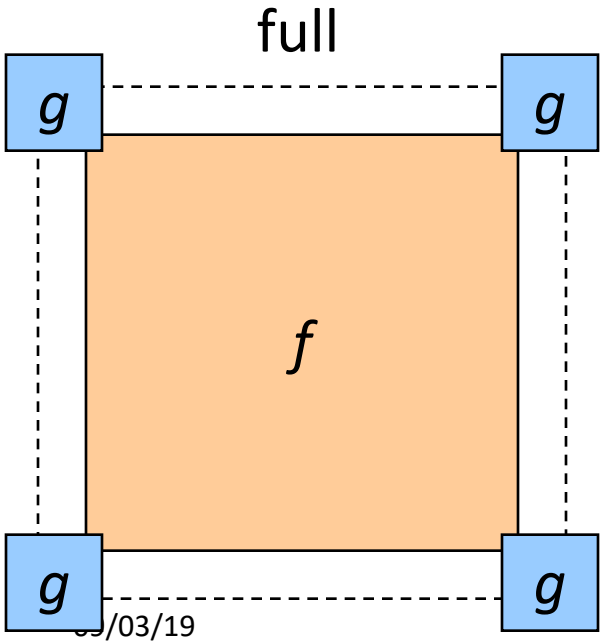
σ = 2 with 30 x 30 kernel

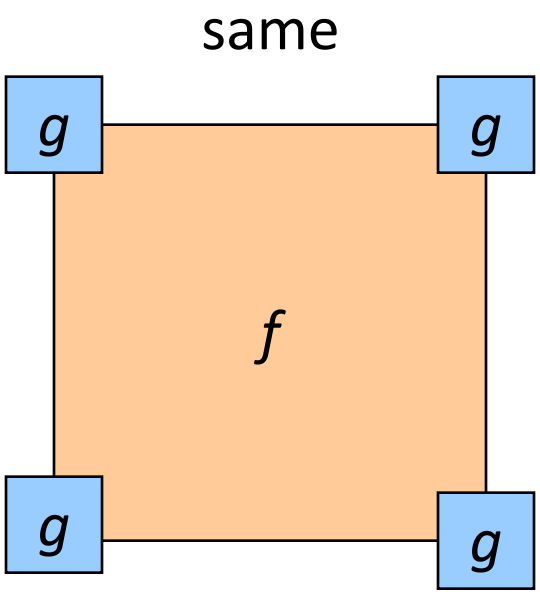


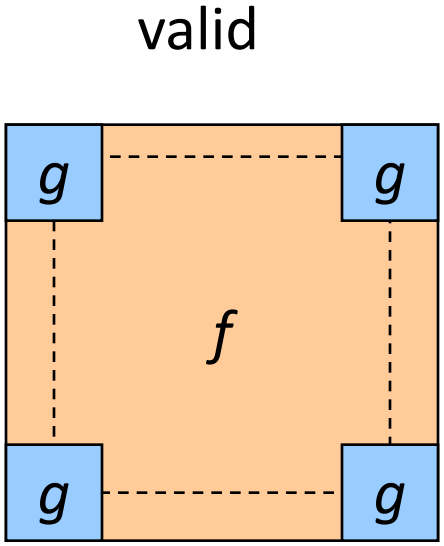
σ = 5 with 30 x 30 kernel

Boundary issues

- What is the size of the output?
  1. *shape* = 'full': output size is sum of sizes of f and g
  2. *shape* = 'same': output size is same as f
  3. *shape* = 'valid': output size is difference of sizes of f and g







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: S. Marschner

Filtering an impulse signal

What is the result of filtering the impulse signal (image) *F* with the 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

$H[u, v]$

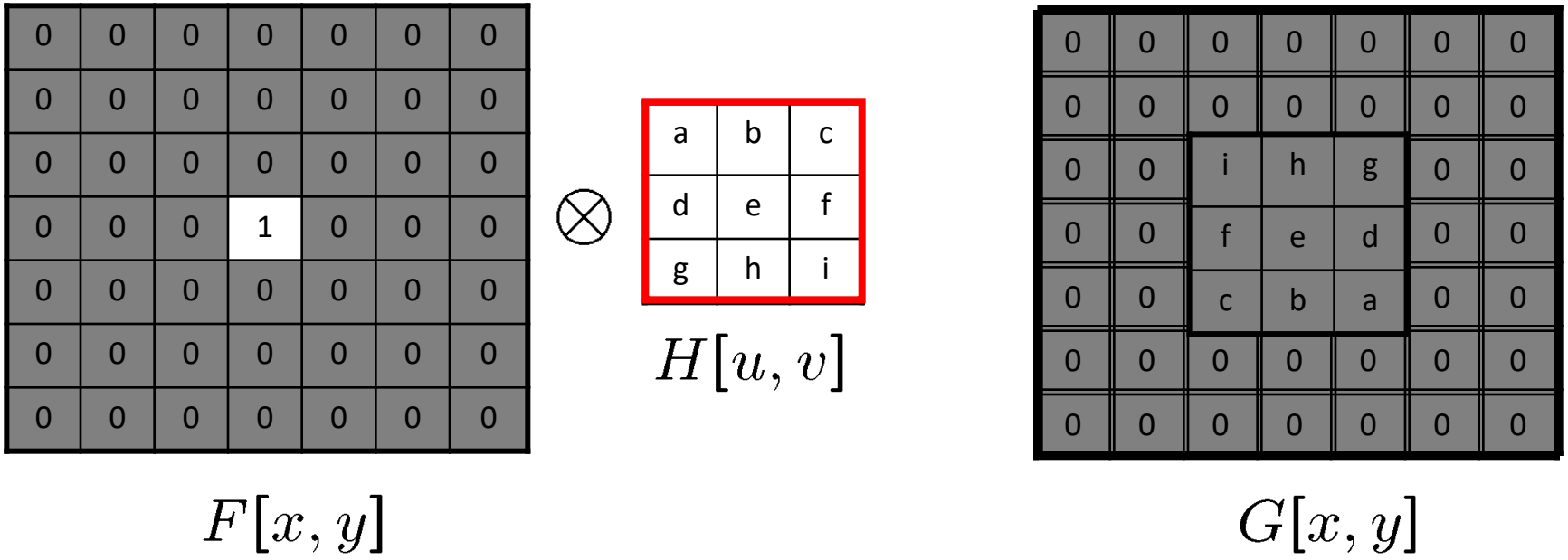
			?			

$G[x, y]$

Filtering an impulse signal (correlation)

$$G = H \otimes F$$

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



$F[x, y]$ $\quad H[u, v]$ $\quad G[x, y]$

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

Filtering an impulse signal (convolution) $G = H \star F$

What is the result of filtering the impulse signal (image) *F* with the 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

$F[x, y]$

$\star$

a	b	c
d	e	f
g	h	i

$H[u, v]$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	a	b	c	0	0
0	0	d	e	f	0	0
0	0	g	h	i	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$G[x, y]$

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

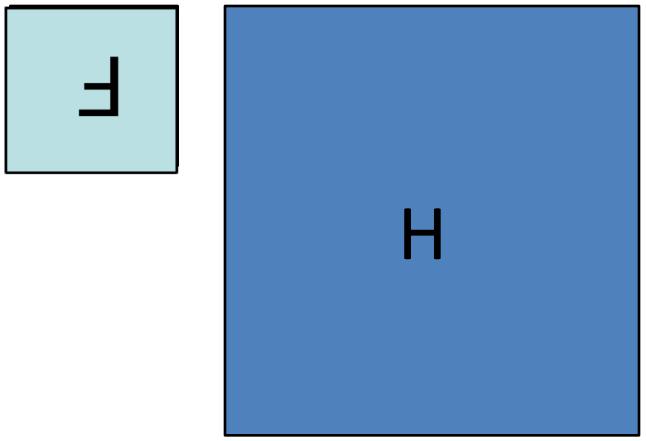
Convolution

- 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



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

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

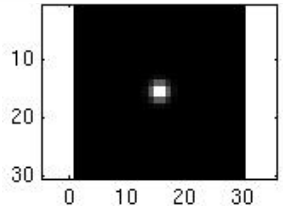
$$G = H \otimes F$$

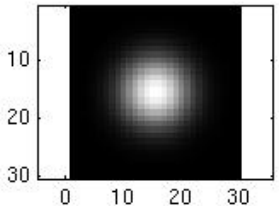
For a Gaussian or box filter, how will the outputs differ?

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

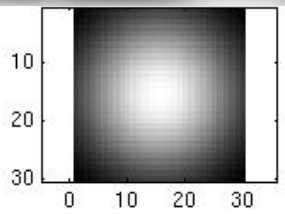
Smoothing with a Gaussian

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
```

Convolving with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Convolving with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Convolving with linear filters

Original

0	0	0
0	0	1
0	0	0

?

Source: D. Lowe

Convolving with linear filters

Original

0	0	0
0	0	1
0	0	0

Shifted left
by 1 pixel with
correlation

Source: D. Lowe

Practice with linear filters

Original

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

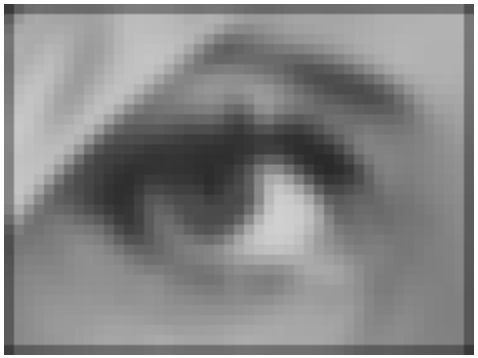
?

Source: D. Lowe

Practice with linear filters

Original

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



Blur (with a box filter)

Source: D. Lowe

Practice with linear filters

Original

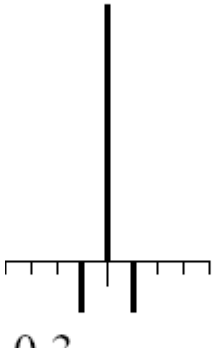
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad ?$$

Source: D. Lowe

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

Source: D. Lowe

What does blurring remove?

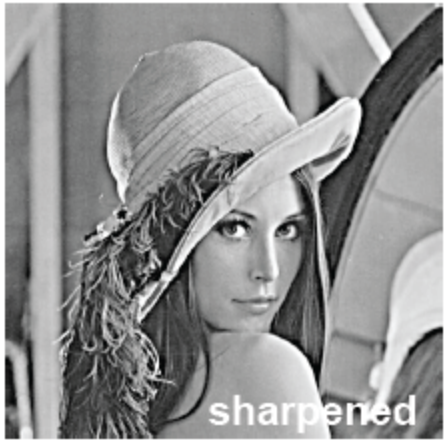
–

=

Now let's add it back……

+ a

=



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: $h * (k f) = k (h * f)$

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:

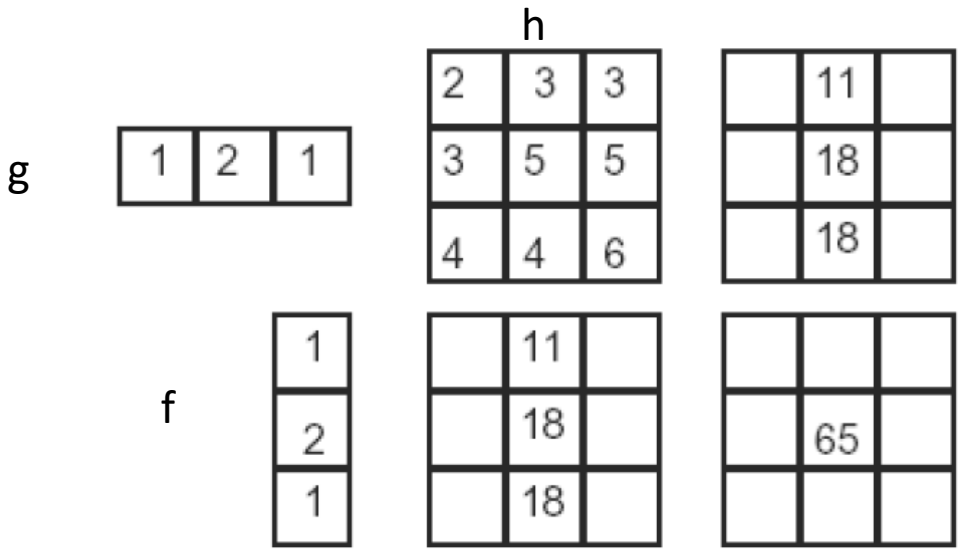
$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

Separability

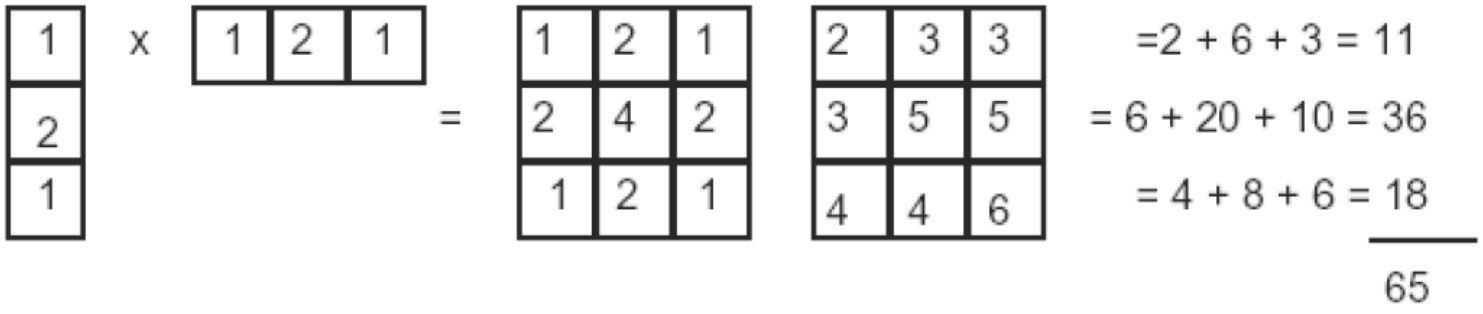
- In some cases, filter is separable, and we can factor into two steps:
  - Convolve all rows
  - Convolve all columns

Separability

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

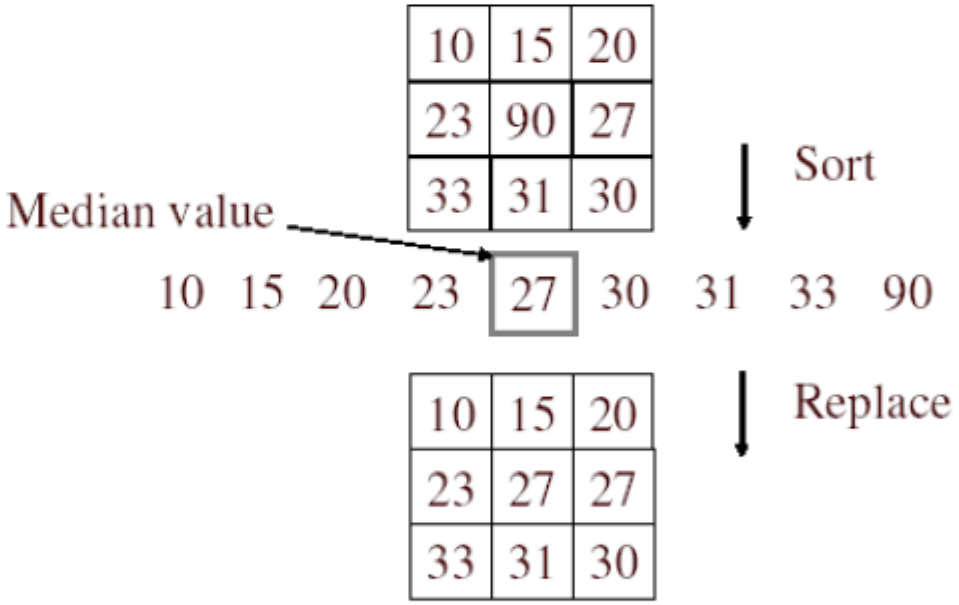


What is the computational complexity advantage for a separable filter of size k x k, in terms of number of operations per output pixel?



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

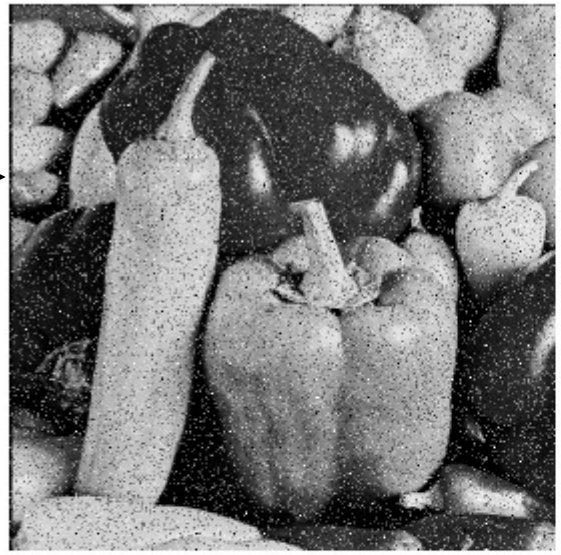
Median filter



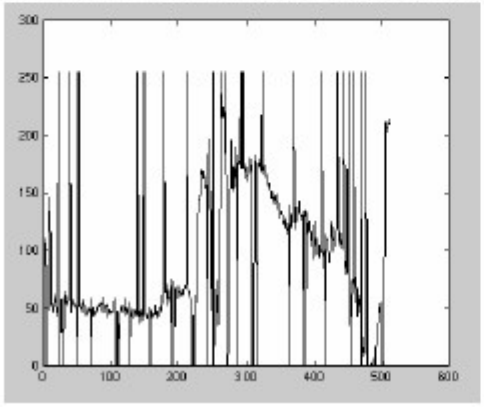
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise

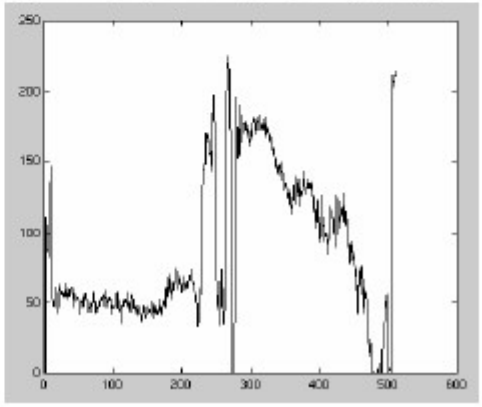
Median filter

Salt and pepper noise



Median filtered



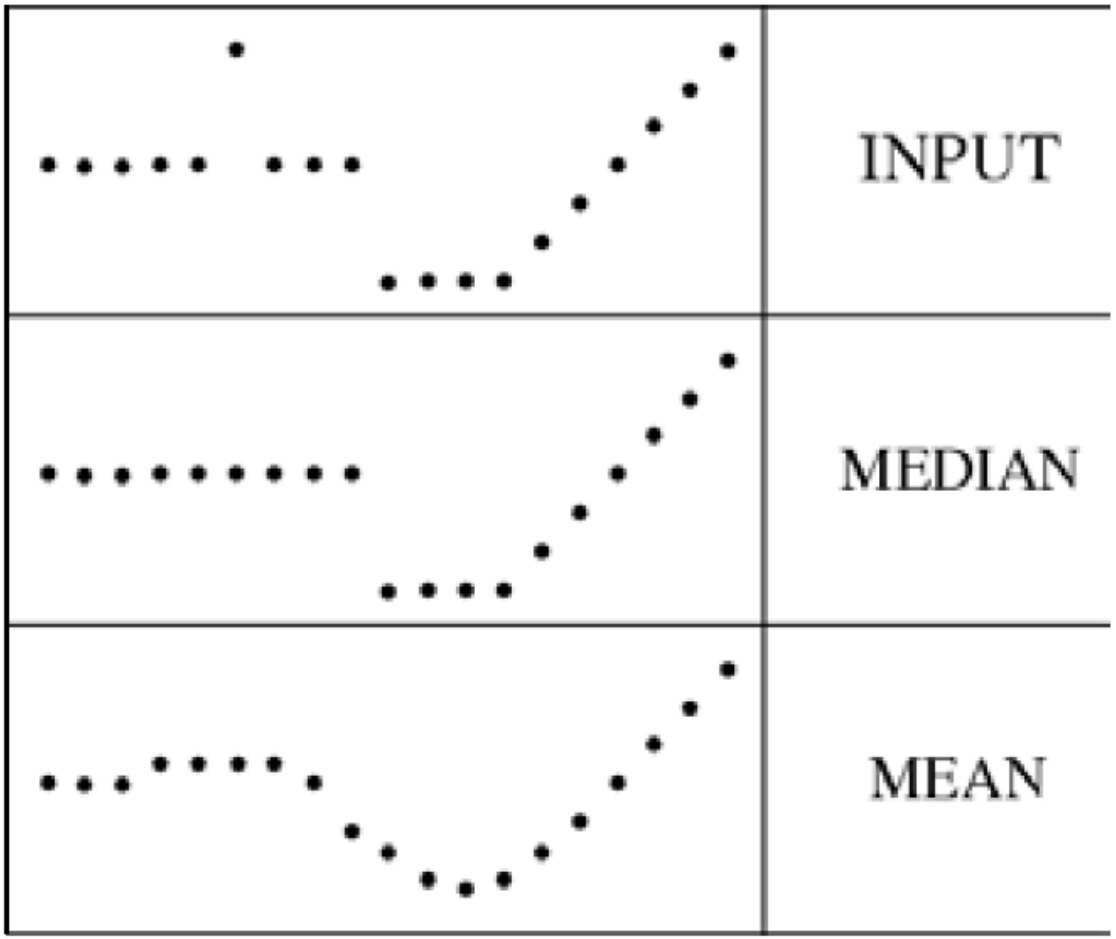


Plots of a row of the image

Source: M. Hebert

Median filter

- Median filter is edge preserving



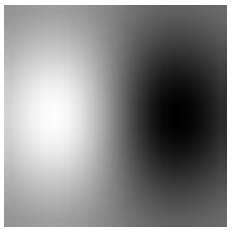
Template Matching

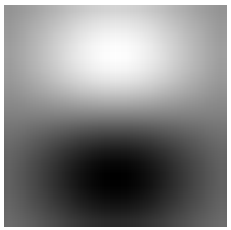
via correlations and other related techniques

Template matching

- Filters as **templates**:

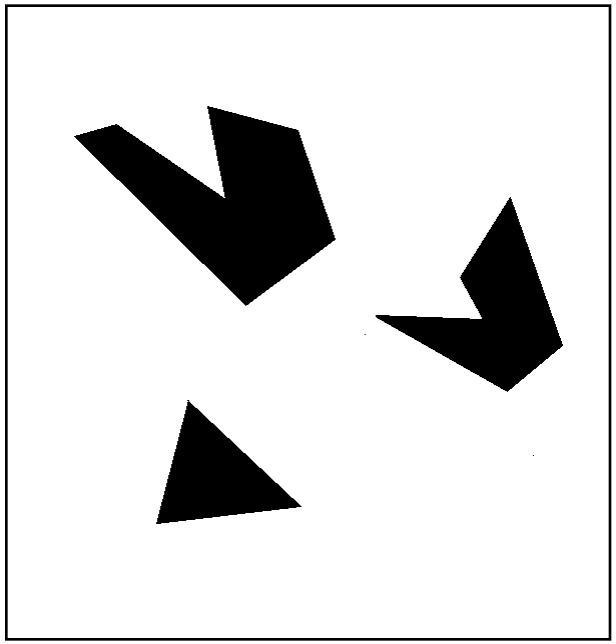
  Note that filters look like the effects they are intended to find --- “matched filters”



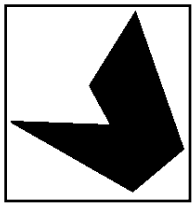


- Use normalized cross-correlation score to find a given pattern (template) in the image.
  - Szeliski Eq. 8.11
- Normalization needed to control for relative brightness.

Template matching



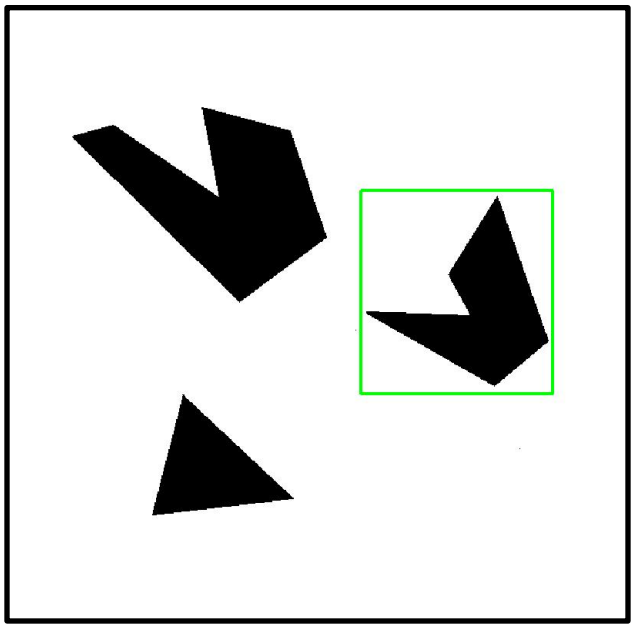
Scene



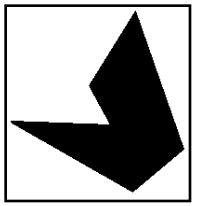
Template (mask)

A toy example

Template matching

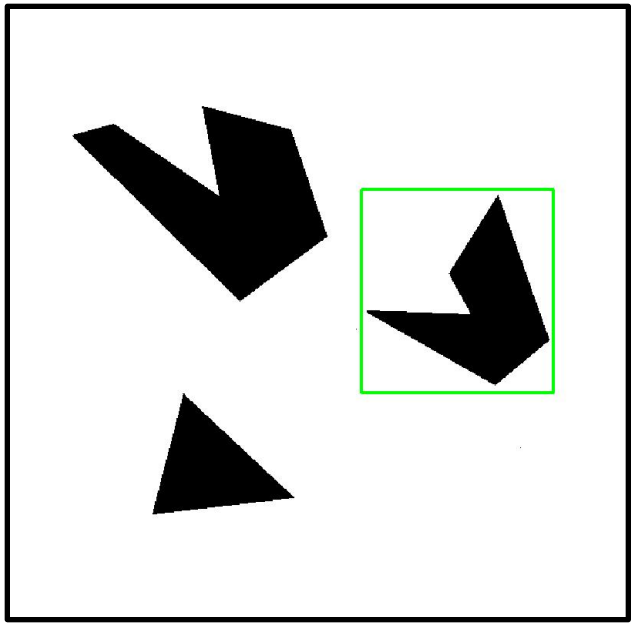


Detected template

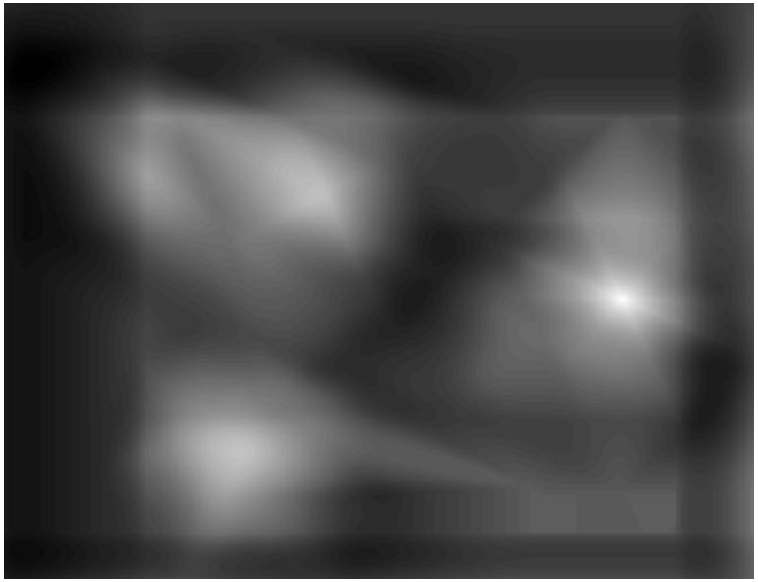


Template

Template matching

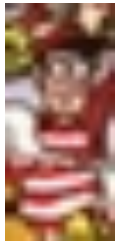


Detected template



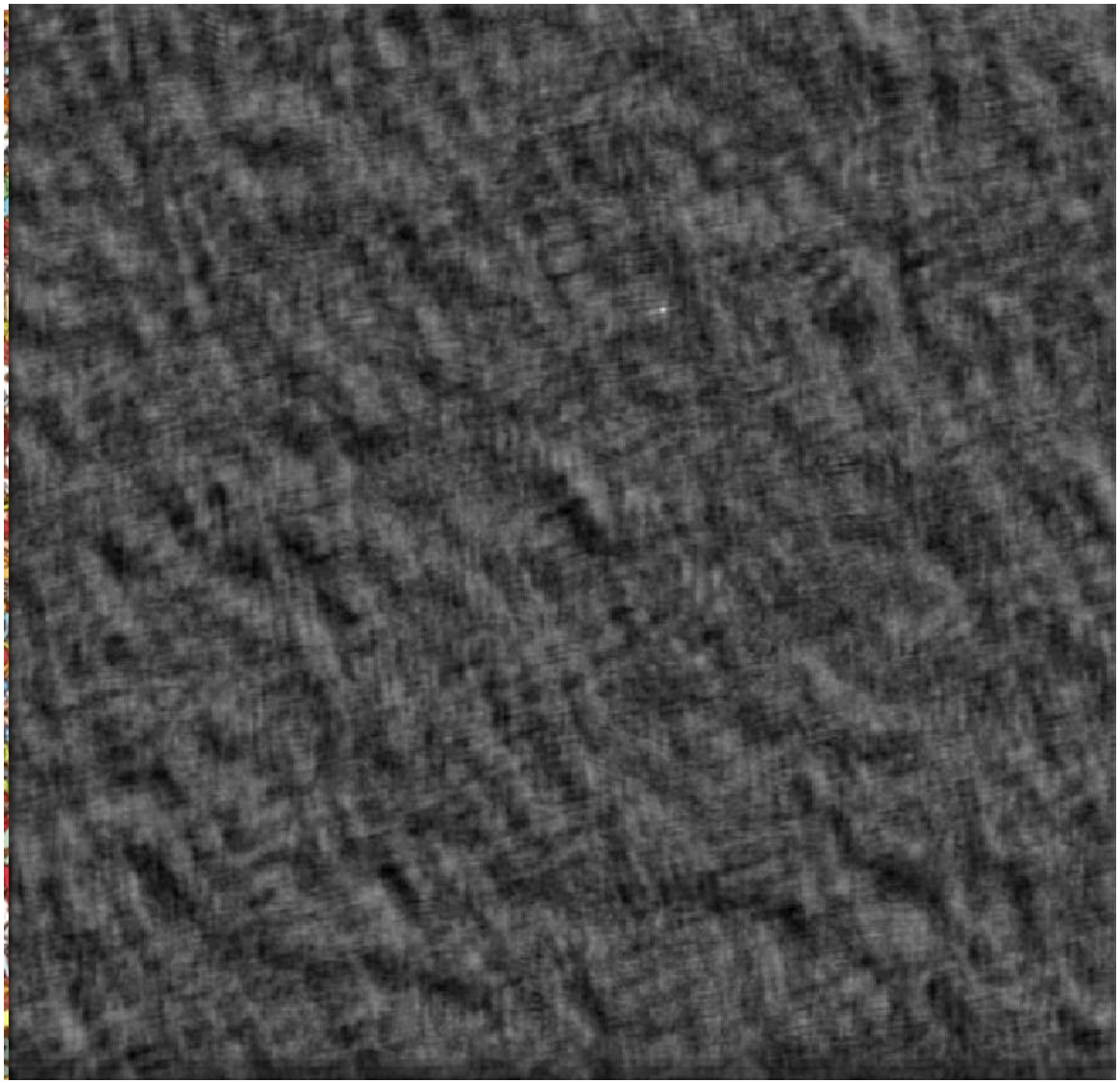
Correlation map

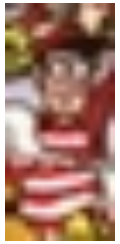
Where's Waldo?



Template

Where's Waldo?

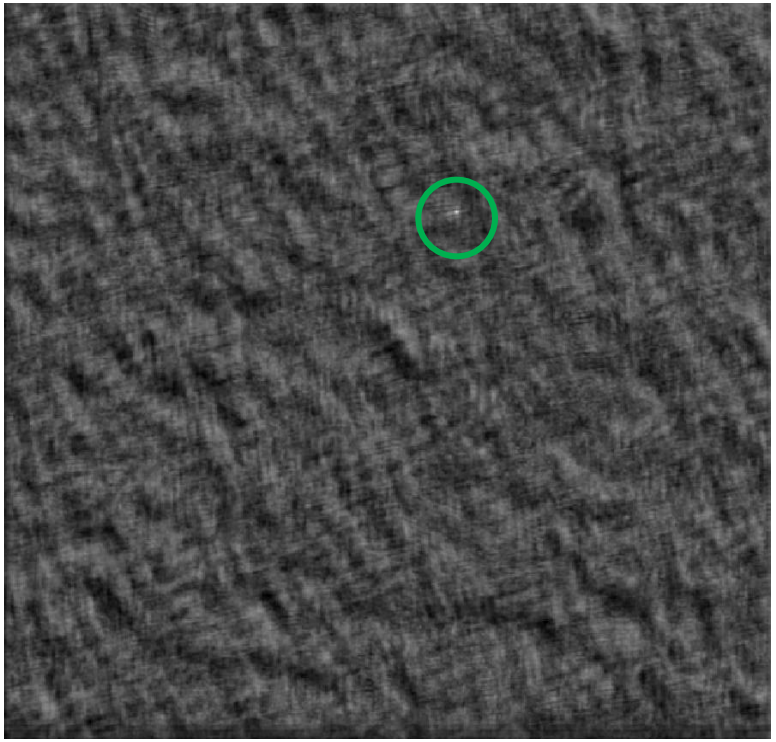




Template

Where's Waldo?

Detected template



Correlation map

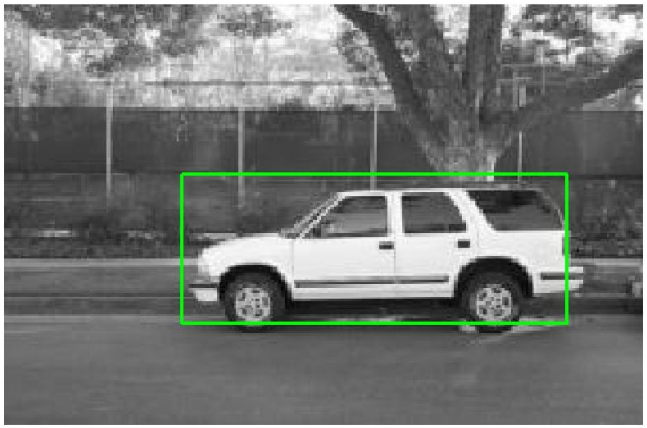
Template matching

Scene

Template

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

Template matching

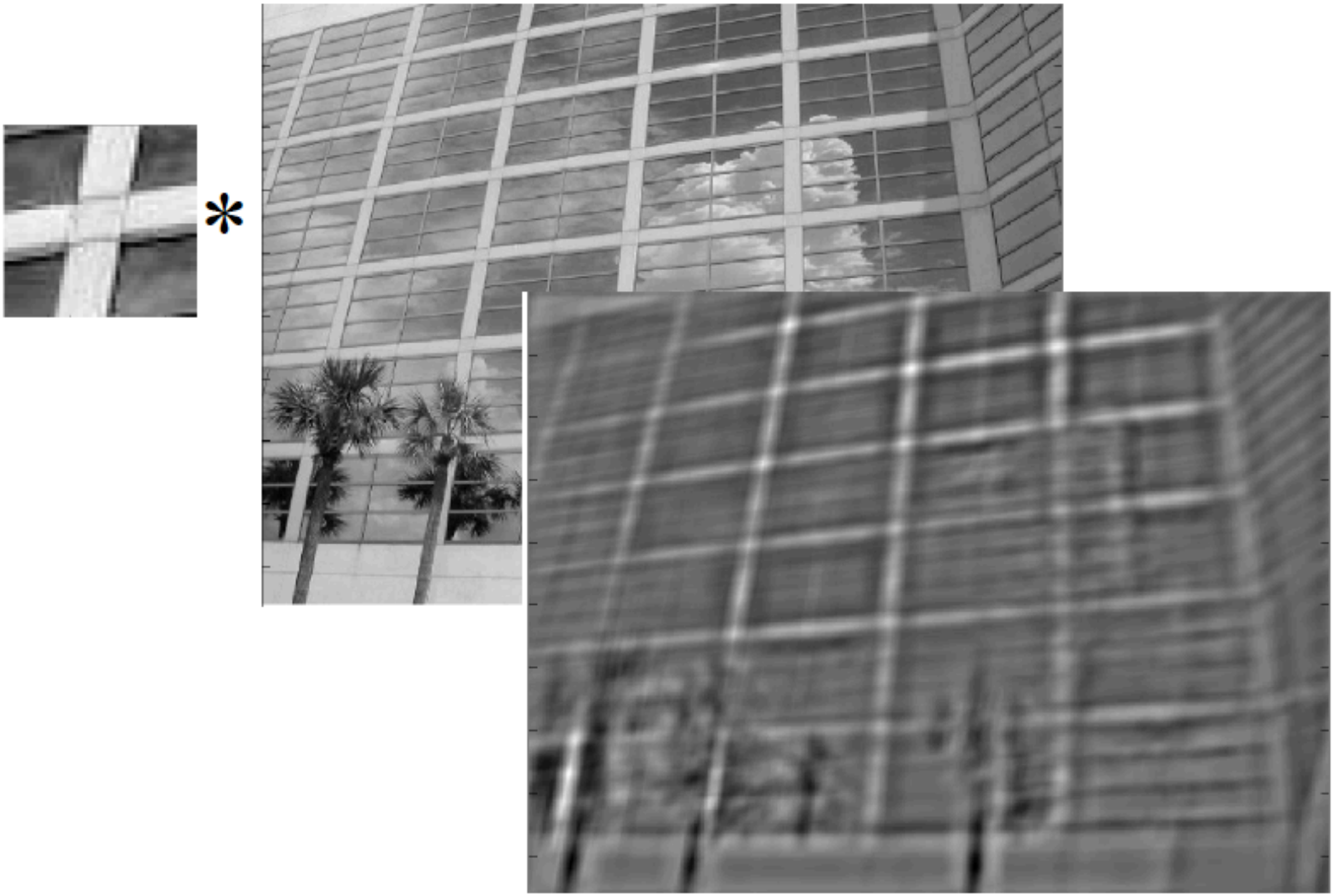


Detected template

Template

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

Template Matching



Comparing windows

- Some possible measures include:

$$\max_{[i,j]\in R} |f(i,j) - g(i,j)|$$

$$\sum_{[i,j]\in R} |f(i,j) - g(i,j)|$$

$$SSD = \sum_{[i,j]\in R} (f(i,j) - g(i,j))^2$$

$$C_{fg} = \sum_{[i,j]\in R} f(i,j)g(i,j)$$

Most popular

Correlations for comparing windows

Correlation C_{fg}

$$C_{fg} = \sum_{[i,j] \in R} f(i,j) g(i,j)$$

If we are doing an exhaustive search over all image patches in the second image, this is cross-correlation of a template with an image.

Example

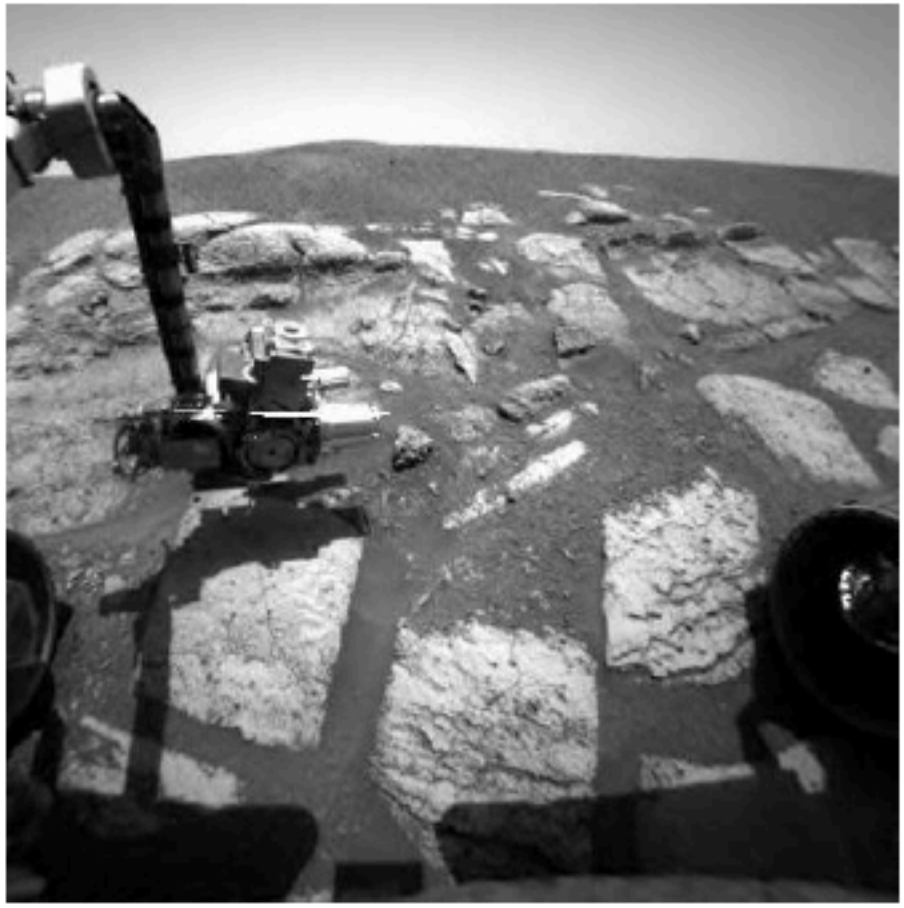


Image 1

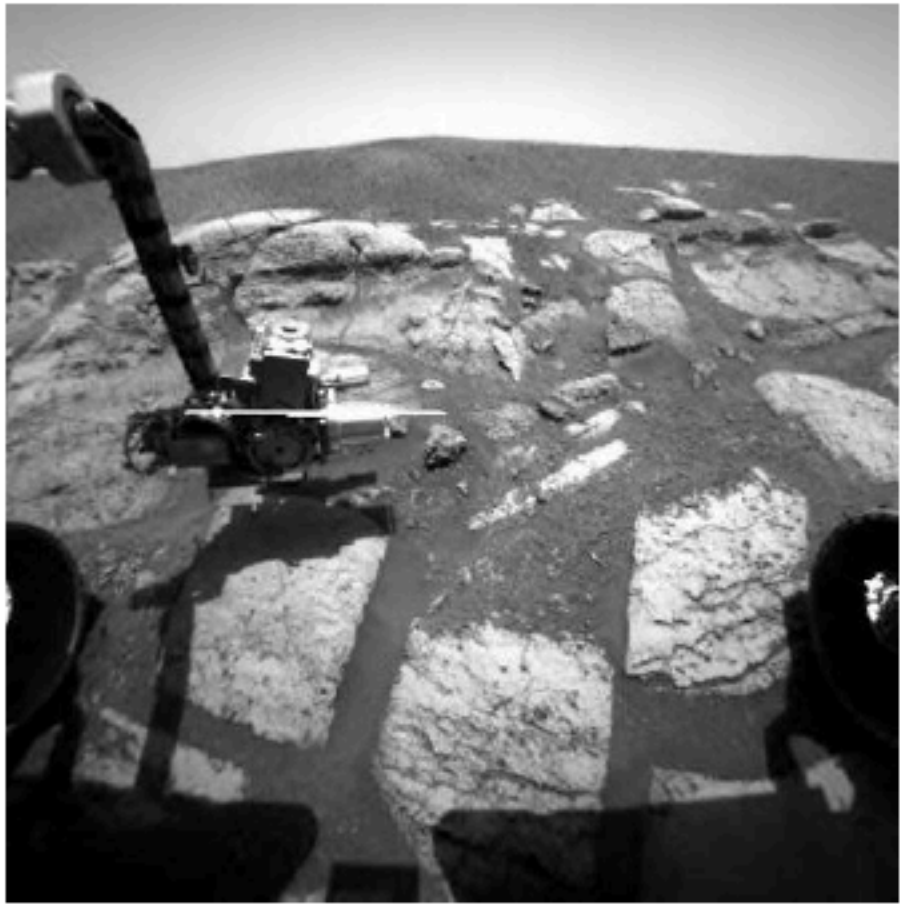
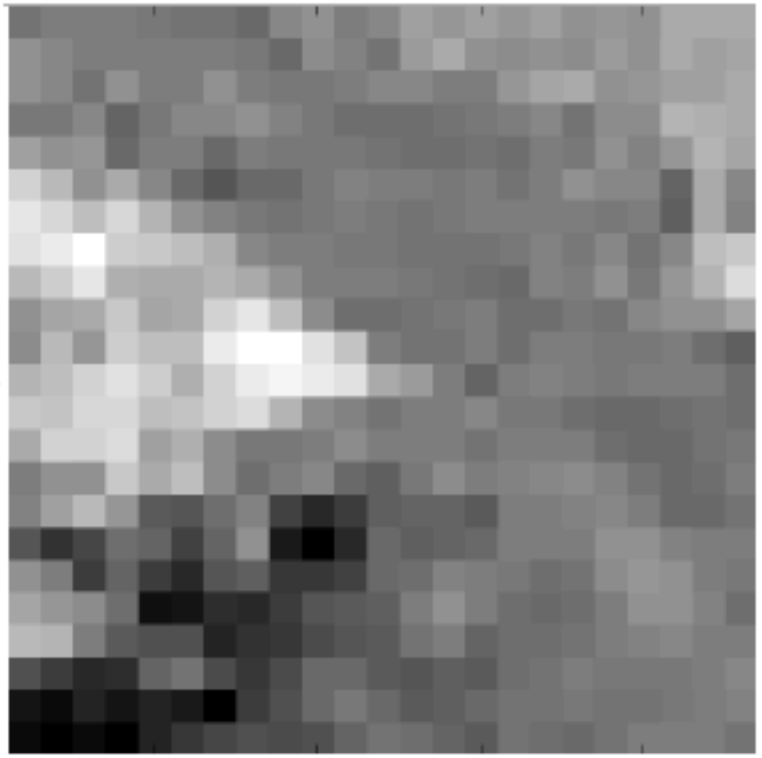


Image 2

Note: this is a stereo pair from the NASA mars rover. The rover is exploring the "El Capitan" formation.

Example

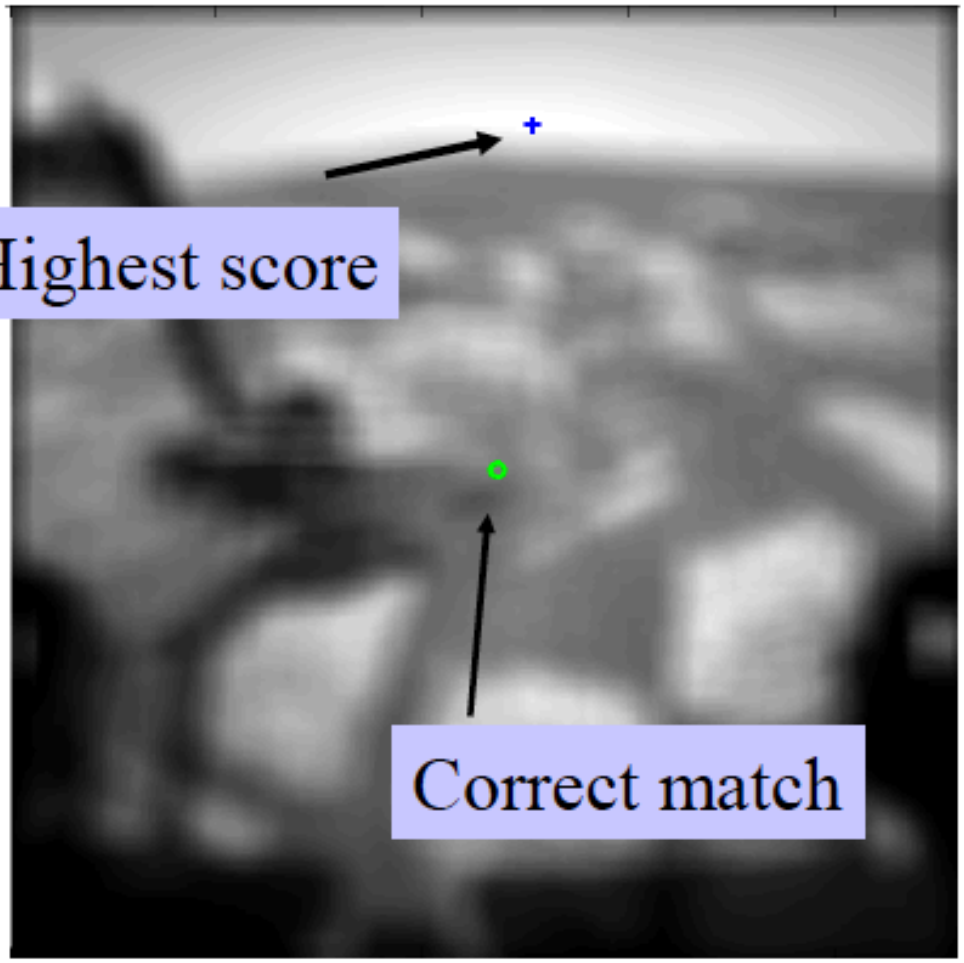
Image 1



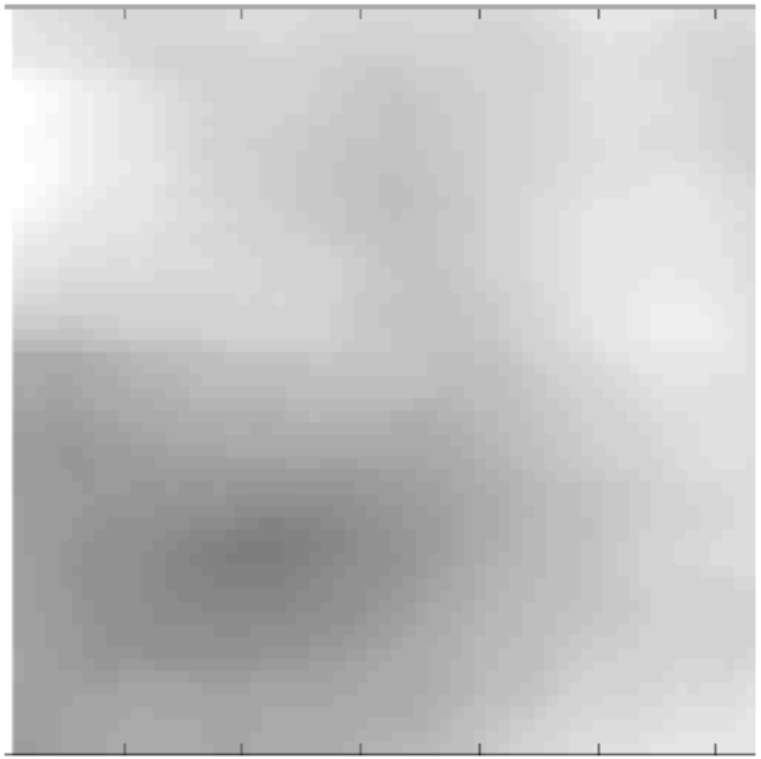
Template
(image patch)

Raw cross-correlation

score = imfilter(image2,tmpl)



Score around correct match



Example cont'd

Note that score image looks a lot like a blurry version of image 2.

This clues us in to the problem with straight correlation with an image template.

Problem with raw correlations

a	b	c
d	e	f
g	h	i

⊗

2v	2v	2v
2v	2v	2v
2v	2v	2v

Versus

a	b	c
d	e	f
g	h	i

⊗

v	v	v
v	v	v
v	v	v

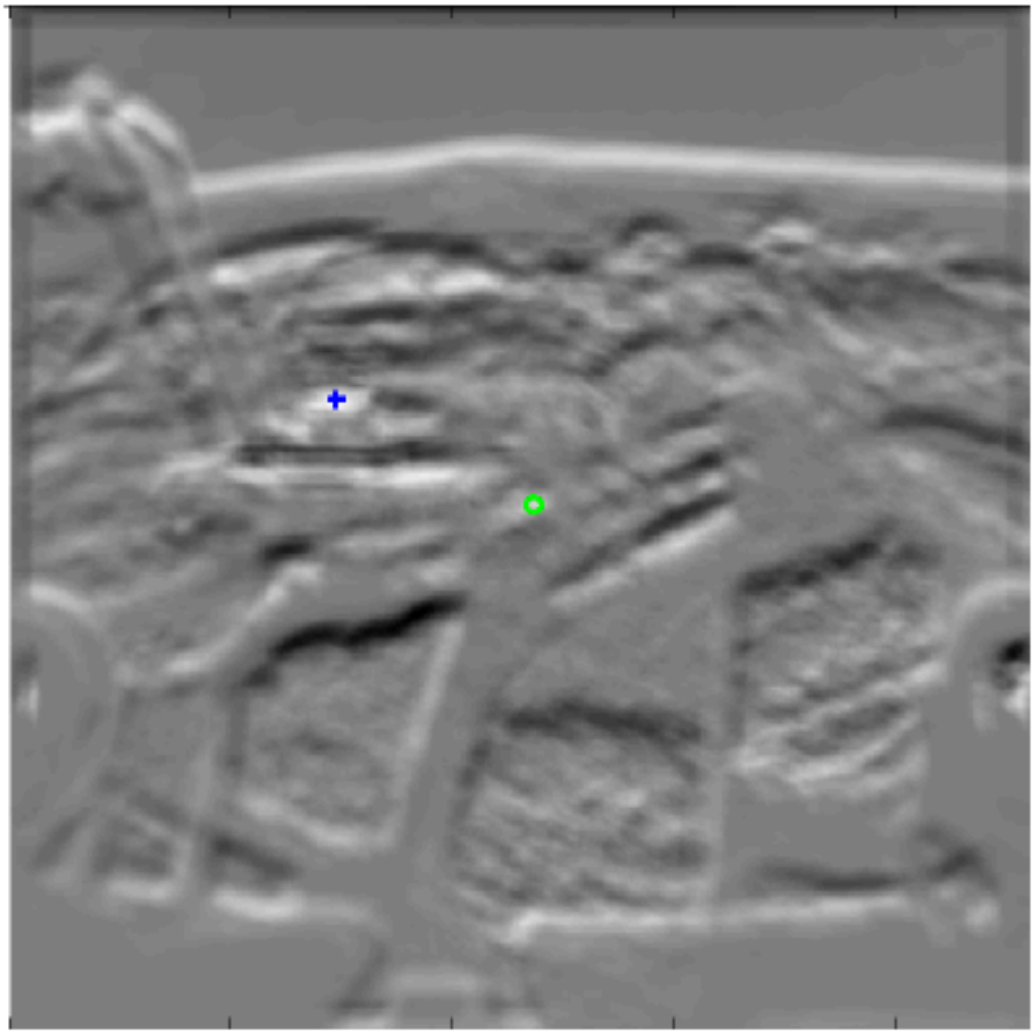
Result is based on the brightness of the underlying image!

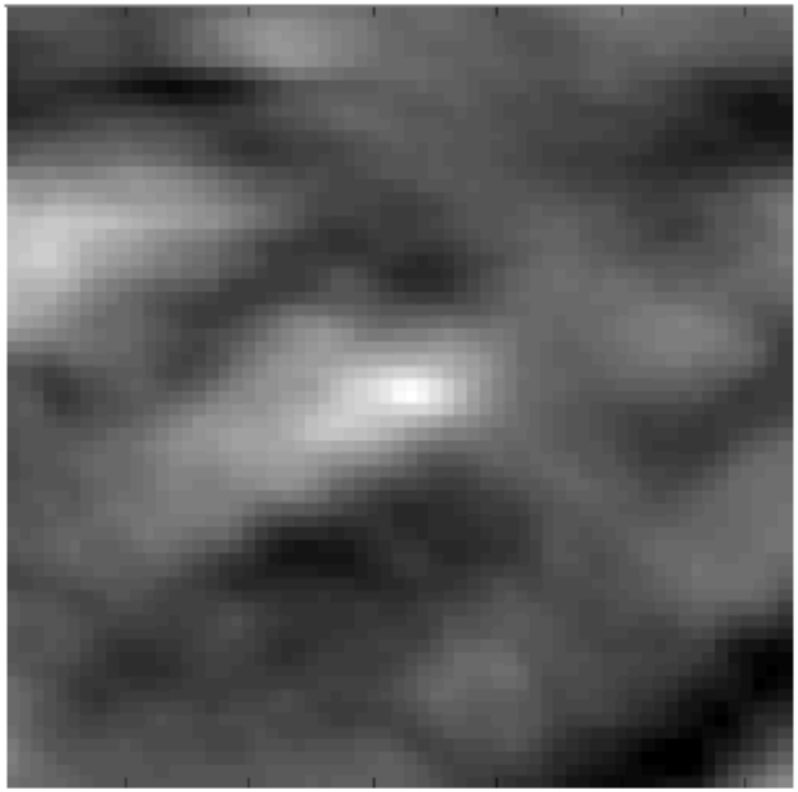
Solution

Subtract off the mean value of the template.

In this way, the correlation score is higher only when darker parts of the template overlap darker parts of the image, and brighter parts of the template overlap brighter parts of the image.

Correlations – zero-mean template





Better! But highest score is still not the correct match.

Note: highest score IS best within local NN of correct match.

"SSD" or "block matching" (Sum of Squared Differences)

- The most popular matching score.
- We will use it when deriving Harris corners
- Some researchers claim it works better than cross-correlation

$$\sum_{[i,j] \in R} (f(i,j) - g(i,j))^2$$

Relation between SSD and Correlation

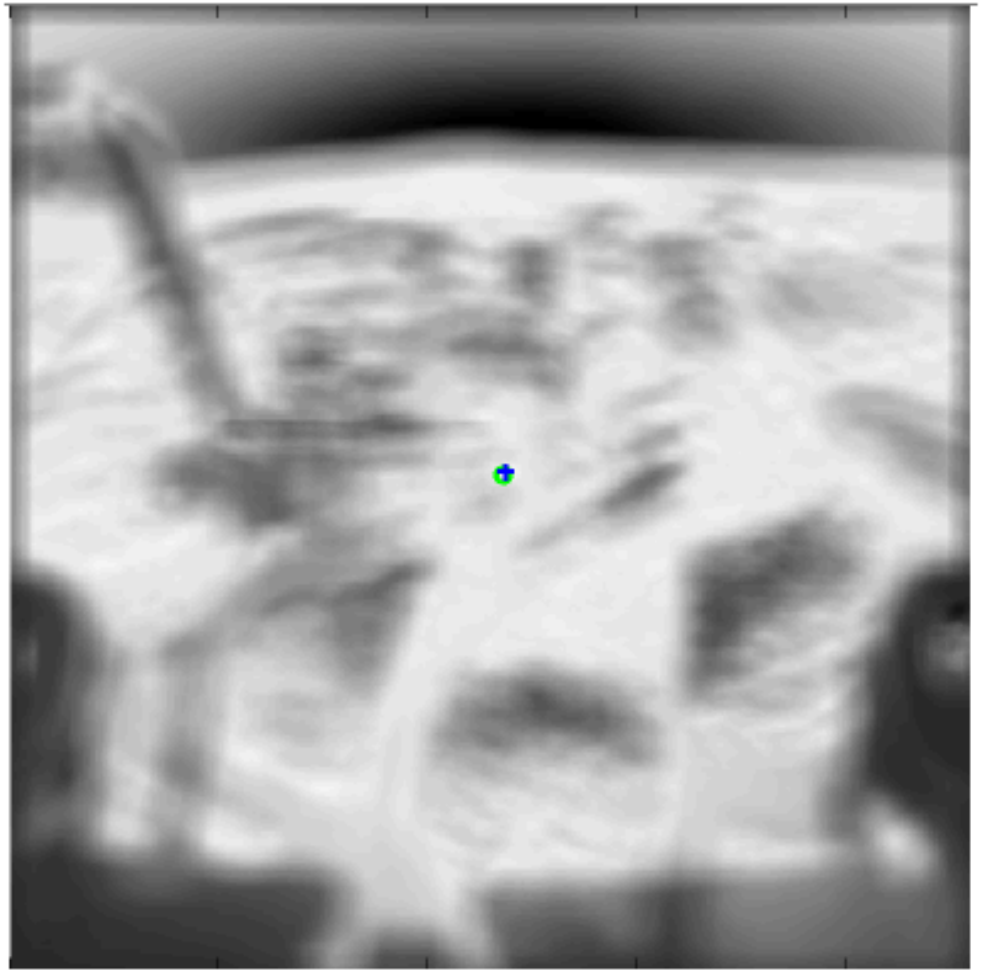
$$SSD = \sum_{[i,j]\in R} (f-g)^2$$

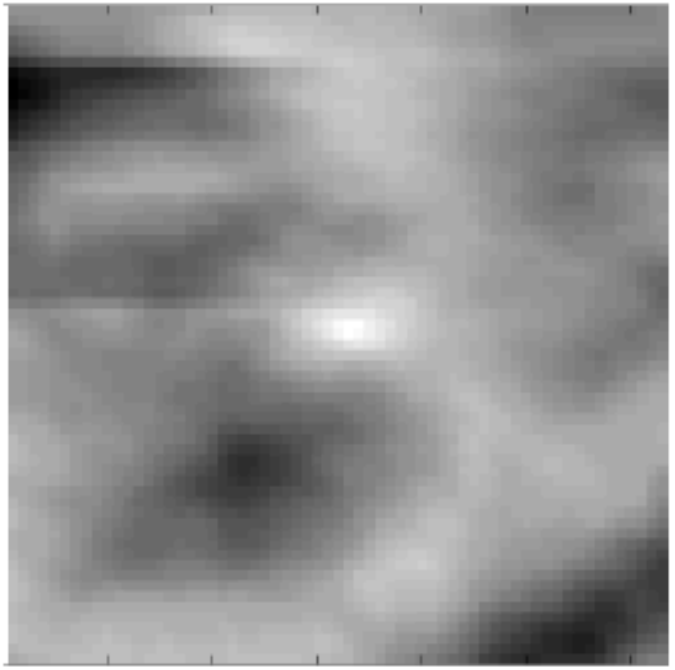
$$= \sum_{[i,j]\in R} f^2 + \sum_{[i,j]\in R} g^2 - 2\sum_{[i,j]\in R} fg$$

$$C_{fg} = \sum_{[i,j]\in R} f(i,j)g(i,j)$$

Correlation!

SSD





Best match (highest score) in image coincides with correct match in this case!

Handling intensity changes

Intensity Changes:

- the camera taking the second image might have different intensity response characteristics than the camera taking the first image
- Illumination in the scene could change
- The camera might have auto-gain control set, so that it's response changes as it moves through the scene.

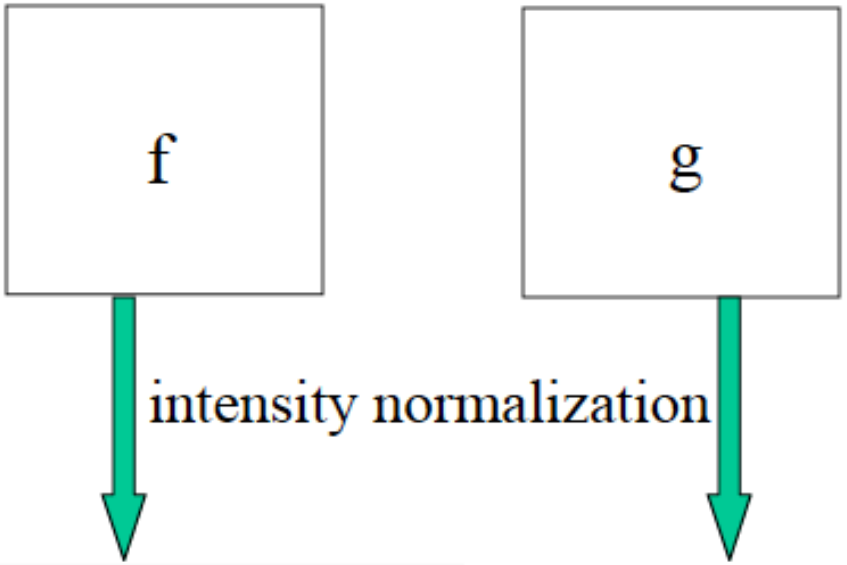
Intensity normalization

- When a scene is imaged by different sensors, or under different illumination intensities,, both the SSD and the C_{fg} can be large for windows representing the same area in the scene!
- A solution is to **NORMALIZE** the pixels in the windows before comparing them by subtracting the mean of the patch intensities and dividing by the std.dev.

Intensity normalization

$$\hat{f} = \frac{f - \bar{f}}{\sqrt{\sum (f - \bar{f})^2}} \qquad \hat{g} = \frac{g - \bar{g}}{\sqrt{\sum (g - \bar{g})^2}}$$

Normalized Cross Correlation



$$\hat{f} = \frac{f - \bar{f}}{\sqrt{\sum (f - \bar{f})^2}} \qquad \hat{g} = \frac{g - \bar{g}}{\sqrt{\sum (g - \bar{g})^2}}$$

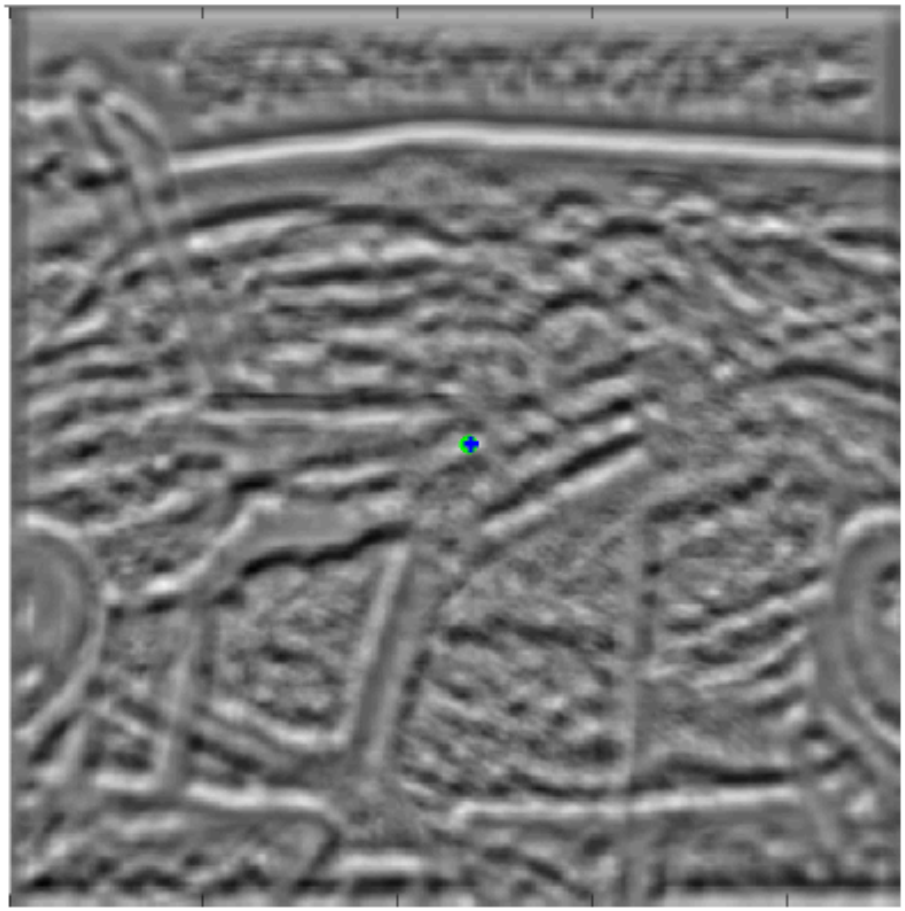
$$\mathrm{NCC(f,g)} = C_{fg}(\hat{f}, \hat{g}) = \sum_{[i,j] \in R} \hat{f}(i,j)\hat{g}(i,j)$$

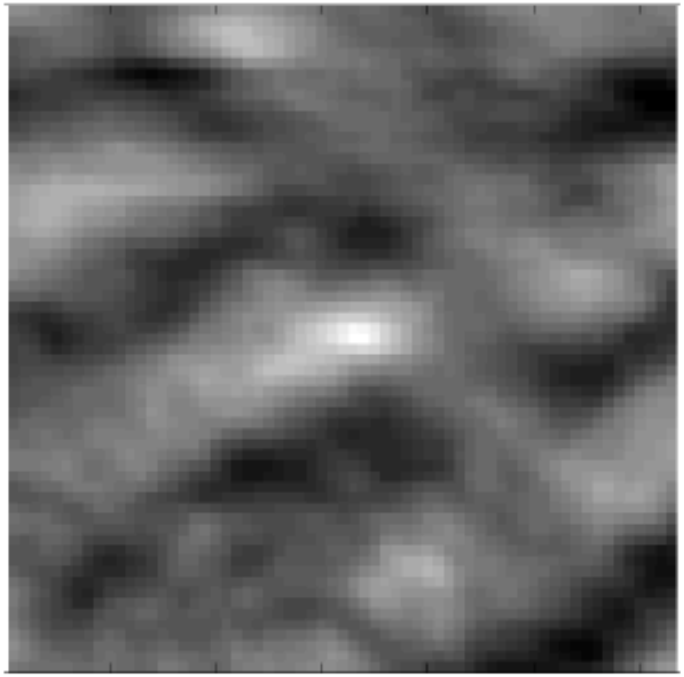
Normalized Cross Correlation

- Important point about NCC:
  - Score values range from 1 (perfect match) to -1 (completely anti-correlated)

- Intuition: treating the normalized patches as vectors, we see they are unit vectors. Therefore, correlation becomes dot product of unit vectors, and thus must range between -1 and 1.

Normalized Cross Correlation





Highest score also coincides with correct match.
Also, looks like less chances of getting a wrong match.

Filters and Convolution - Crucial Points

- A filter is a pattern of weights (<< ||image||)
- Convolution applies the filter to the image
- Output measures similarity between filter and image patch
  - only a rough estimate
- Normalized correlation gives improved pattern detection

Slide Credits

- Rob Fergus – NYU
- Darell Trevor – UC Berkeley
- Fei Fei Li - Stanford
- Svetlana Lazebnik – UIUC
- David A. Forsyth – UIUC
- Robert Collins - PennState

Questions