Linear Filters

April 7th, 2020

Yong Jae Lee
UC Davis

Announcements

- PS0 due 4/10 Friday at 11:59 pm
- Carefully read course website
- Sign-up for piazza

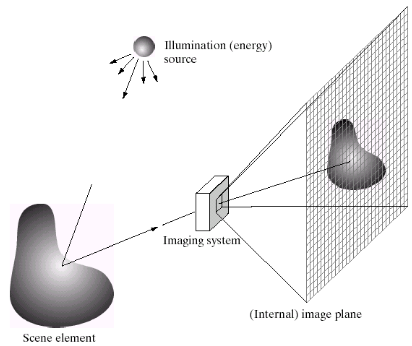
2

Plan for today

- Image formation
- Image noise
- Linear filters
  - Examples: smoothing filters
- Convolution / correlation

3

Image Formation



Slide credit: Derek Hoiem 4

Digital camera

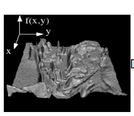
A digital camera replaces film with a sensor array

- Each cell in the array is light-sensitive diode that converts photons to electrons
- http://electronics.howstuffworks.com/digital-camera.htm

Slide credit: Steve Seitz 5

Digital images

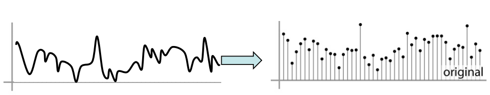
- **Sample** the 2D space on a regular grid
- **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.



j →, i ↓

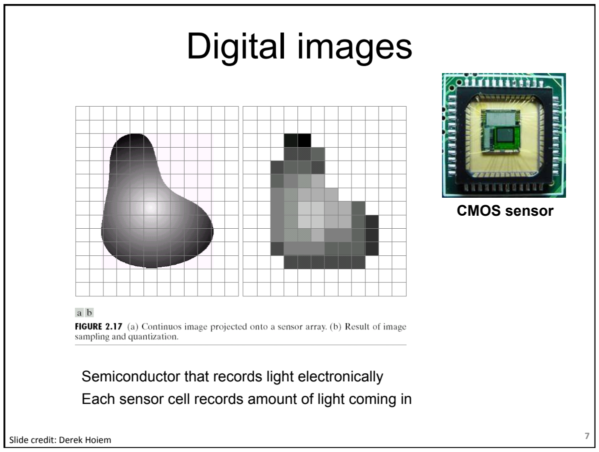
62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

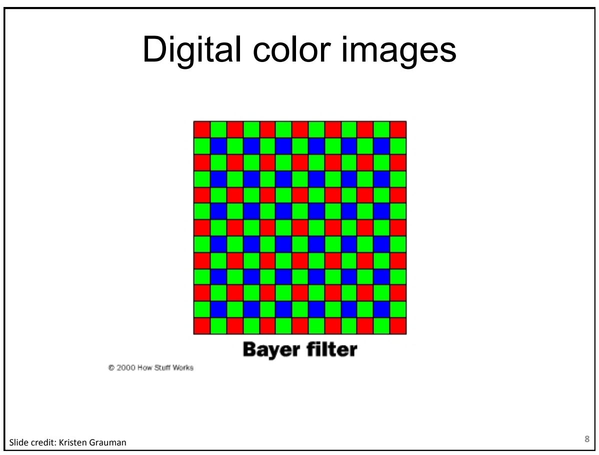
2D

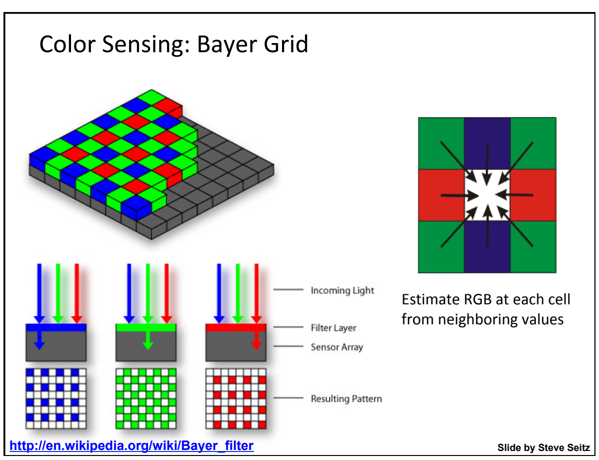


1D

Slide credit: Kristen Grauman, Adapted from Steve Seitz 6

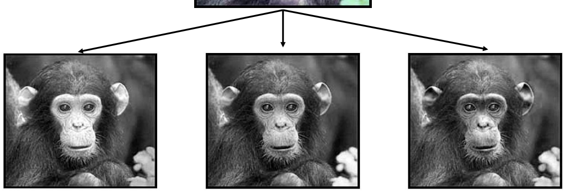

Digital images
CMOS sensor
a b
FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.
Semiconductor that records light electronically
Each sensor cell records amount of light coming in
Slide credit: Derek Hoiem
7


Digital color images
Bayer filter
© 2000 How Stuff Works
Slide credit: Kristen Grauman
8


Color Sensing: Bayer Grid
Incoming Light
Filter Layer
Sensor Array
Resulting Pattern
Estimate RGB at each cell from neighboring values
http://en.wikipedia.org/wiki/Bayer_filter
Slide by Steve Seitz

Digital color images

Color images,
RGB color
space



R G B

Slide credit: Kristen Grauman

10

Images in Matlab

- Images represented as a matrix
- Suppose we have an NxM RGB image called "im"
  - im(1,1,1) = top-left pixel value in R-channel
  - im(y, x, b) = y pixels down, x pixels to right in the b^th channel
  - im(N, M, 3) = bottom-right pixel in B-channel
- imread(filename) returns a uint8 image (values 0 to 255)
  - Convert to double format (values 0 to 1) with im2double

column →

row ↓

R

0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

G

0.92	0.99
0.95	0.91
0.91	0.92
0.97	0.95
0.79	0.85
0.45	0.33
0.49	0.74
0.82	0.93
0.90	0.99
0.93	0.97
0.99	0.93

0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

B

0.92	0.99
0.95	0.91
0.91	0.92
0.97	0.95
0.79	0.85
0.45	0.33
0.49	0.74
0.82	0.93
0.90	0.99
0.93	0.97
0.99	0.93

0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

Slide credit: Derek Hoiem

11

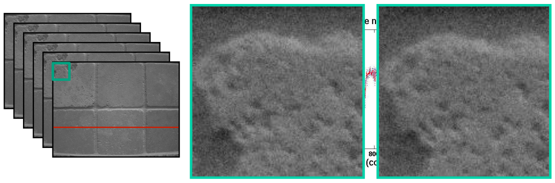
Image filtering

- Compute a function of the local neighborhood at each pixel in the image
  - Function specified by a "filter" or mask saying how to combine values from neighbors
- Uses of filtering:
  - Enhance an image (denoise, resize, increase contrast, etc)
  - Extract information (texture, edges, interest points, etc)
  - Detect patterns (template matching)

Slide credit: Kristen Grauman, Adapted from Derek Hoiem

12

Motivation: noise reduction



- Even multiple images of the **same static scene** will not be identical.

Slide adapted from Kristen Grauman

13

Common types of noise

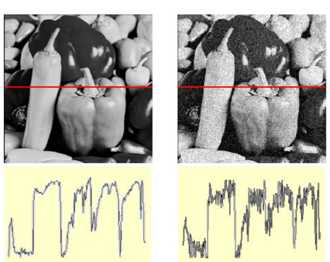
- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Original

Slide credit: Steve Seitz

14

Gaussian noise



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

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

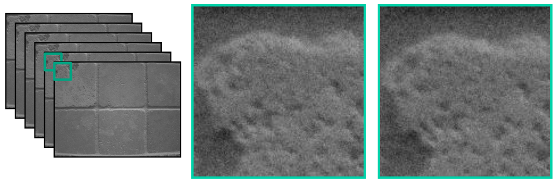
```
>> noise = randn(size(im)).*sigma;
>> output = im + noise;
```

What is impact of the sigma?

Figure from Martial Hebert

15

Motivation: noise reduction



- Even multiple images of the same static scene will not be identical.
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

Slide credit: Kristen Grauman

16

First attempt at a solution

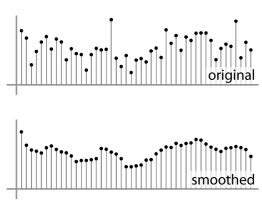
- Let's replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors
  - Expect noise processes to be independent from pixel to pixel

Slide credit: Kristen Grauman

17

First attempt at a solution

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



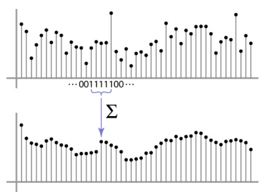
Slide credit: S. Marschner

18

Weighted Moving Average

Can add weights to our moving average

Weights [1, 1, 1, 1, 1] / 5

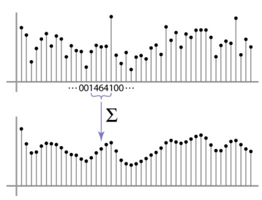


Slide credit: S. Marschner

19

Weighted Moving Average

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



Slide credit: S. Marschner

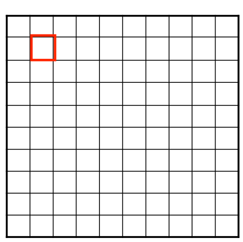
20

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

$g[x, y]$



Slide credit: Steve Seitz

21

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								

Slide credit: Steve Seitz

22

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								

Slide credit: Steve Seitz

23

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							

Slide credit: Steve Seitz

24

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							

Slide credit: Steve Seitz

25

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						

Slide credit: Steve Seitz

26

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						

Slide credit: Steve Seitz

27

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					

Slide credit: Steve Seitz

28

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					

Slide credit: Steve Seitz

29

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				

Slide credit: Steve Seitz

30

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	

Slide credit: Steve Seitz

31

Correlation filtering

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

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

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

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{h(u,v)}_{\textit{Non-uniform weights}} f(i+u, j+v)$$

Slide adapted from Kristen Grauman

32

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.

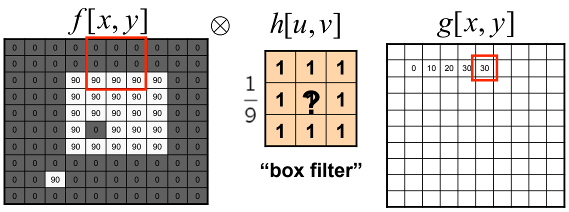
The filter "**kernel**" or "**mask**" *h*[*u,v*] is the prescription for the weights in the linear combination.

Slide adapted from Kristen Grauman

33

Averaging filter

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



$$g = h \otimes f$$

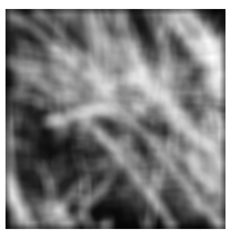
Slide adapted from Kristen Grauman

34

Smoothing by averaging


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


original


filtered

What if the filter size was 5 x 5 instead of 3 x 3?

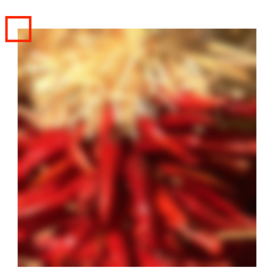
Slide credit: Kristen Grauman

35

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



Slide credit: S. Marschner

36

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')

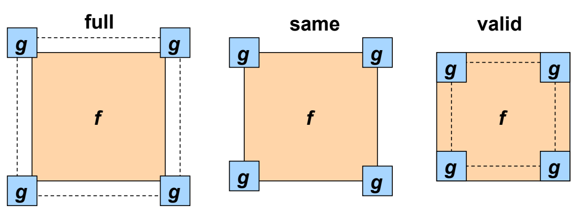
Slide credit: S. Marschner

37

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



Slide credit: Svetlana Lazebnik

38

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

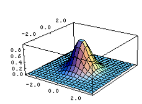
$f[x, y]$

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

$h[u, v]$

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

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

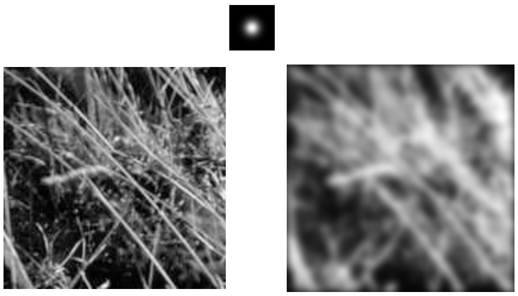


- Removes high-frequency components from the image ("low-pass filter").

Slide credit: Steve Seitz

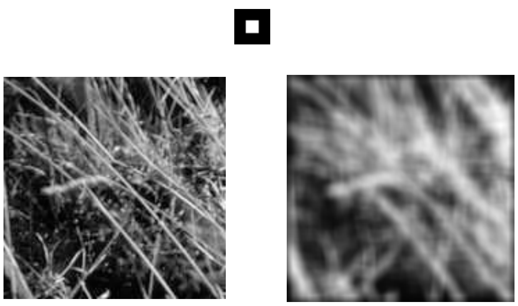
39

Smoothing with a Gaussian



40

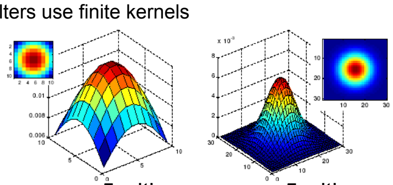
Smoothing with a box-filter



41

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

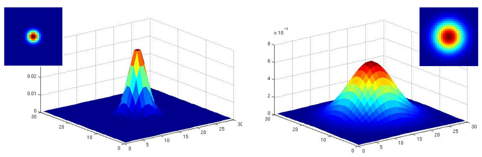
σ = 5 with 30 x 30 kernel

Slide credit: Kristen Grauman

42

Gaussian filters

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



σ = 2 with 30 x 30 kernel

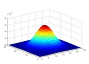
σ = 5 with 30 x 30 kernel

Slide credit: Kristen Grauman 43

Matlab

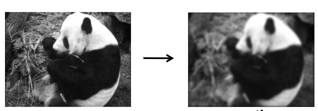
```
>> hsize = 30;
>> sigma = 5;
>> h = fspecial('gaussian', hsize, sigma);

>> mesh(h);
```



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

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

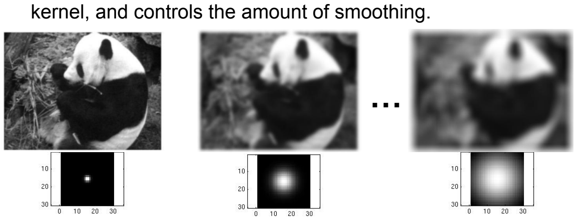


outim

Slide credit: Kristen Grauman 44

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', hsize, sigma);
  out = imfilter(im, h);
  imshow(out);
  pause;
end
```

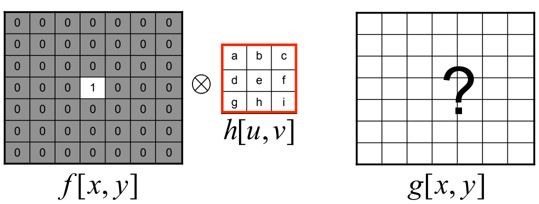
Slide credit: Kristen Grauman 45

Properties of smoothing filters

- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove "high-frequency" components; "low-pass" filter

Filtering an impulse signal

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



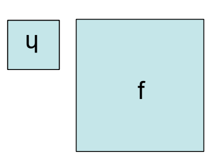
Convolution

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

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(i-u, j-v)$$

$$g = h * f$$

Notation for convolution operator



Convolution vs. correlation

Convolution

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h(u,v) f(i-u, j-v)$$

$$g = h * f$$

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?

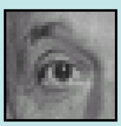
If the input is an impulse signal, how will the outputs differ?

Slide adapted from Kristen Grauman 49

Predict the outputs using *correlation* filtering

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

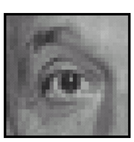
$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = ?$$



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

Slide credit: Kristen Grauman 50

Practice with linear filters



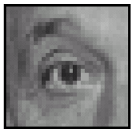
Original

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

?

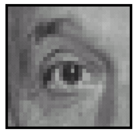
Slide credit: David Lowe 51

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0

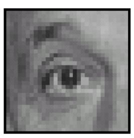


Filtered
(no change)

Slide credit: David Lowe

52

Practice with linear filters



Original

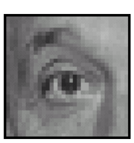
0	0	0
0	0	1
0	0	0

?

Slide credit: David Lowe

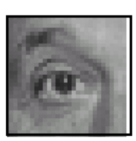
53

Practice with linear filters



Original

0	0	0
0	0	1
0	0	0

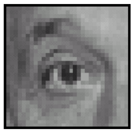


Shifted left
by 1 pixel
with
correlation

Slide credit: David Lowe

54

Practice with linear filters



Original

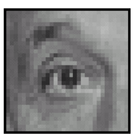
$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

?

Slide credit: David Lowe

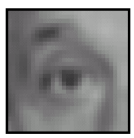
55

Practice with linear filters



Original

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

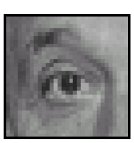


Blur (with a box filter)

Slide credit: David Lowe

56

Practice with linear filters



Original

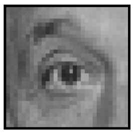
$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

?

Slide credit: David Lowe

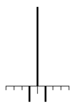
57

Practice with linear filters



Original

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

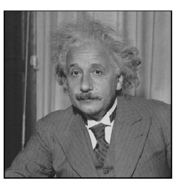


Sharpening filter:
accentuates differences with local average

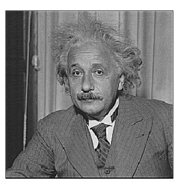
Slide credit: David Lowe

58

Filtering examples: sharpening



before



after

Slide credit: Kristen Grauman

59

Properties of convolution

- 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.
- Superposition:
  - h * (f1 + f2) = (h * f1) + (h * f2)

Slide credit: Kristen Grauman

60

Properties of convolution

- Commutative:
  f * g = g * f
- Associative
  (f * g) * h = f * (g * h)
- Distributes over addition
  f * (g + h) = (f * g) + (f * h)
- Scalars factor out
  kf * g = f * kg = k(f * g)
- Identity:
  unit impulse e = […, 0, 0, 1, 0, 0, …]. f * e = f

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.,

g $\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$ **h** $\begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$

f

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

Effect of smoothing filters

5x5

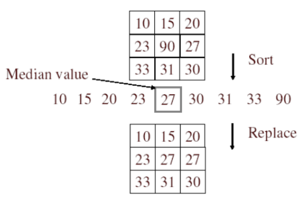
Additive Gaussian noise

Salt and pepper noise

Slide credit: Kristen Grauman

64

Median filter



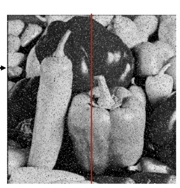
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Non-linear filter

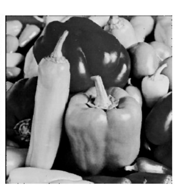
Slide credit: Kristen Grauman

65

Median filter

Salt and pepper noise





Median filtered

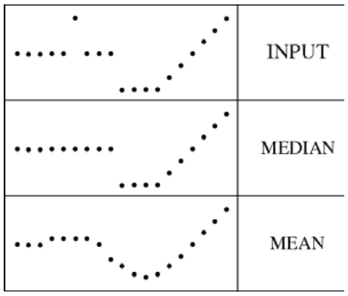
Matlab: output im = medfilt2(im, [h w]);

Slide credit: Martial Hebert

66

Median filter

- Median filter is edge preserving



Slide credit: Kristen Grauman

67

Filtering application: Hybrid Images

Slide credit: Kristen Grauman

Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006

68

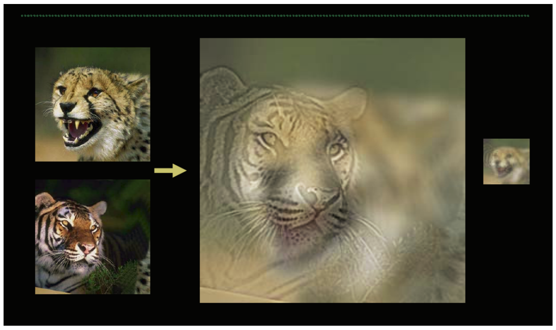
Application: Hybrid Images

A. Oliva, A. Torralba, P.G. Schyns, "Hybrid Images," SIGGRAPH 2006

unit

Slide credit: Kristen Grauman

69



Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006 70

Changing expression

Sad ⟷ Surprised

Aude Oliva & Antonio Torralba & Philippe G Schyns, SIGGRAPH 2006 71

Summary

- Image formation
- 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

72

Coming up

- Thursday:
  - Filtering part 2: filtering for features

Questions?

See you Thursday!