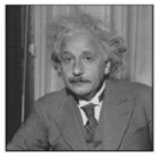
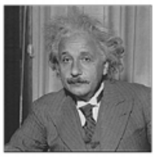
Linear Filters



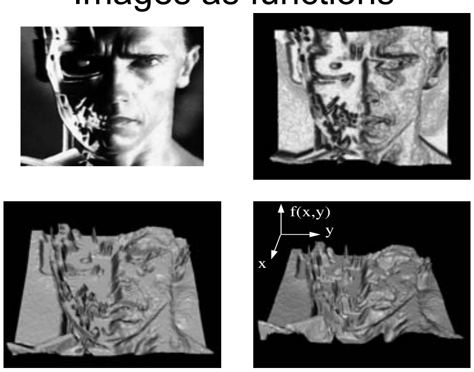
Tuesday, Sept 1



Plan for today

- Image noise
- Linear filters
  - Smoothing filters
- Convolution / correlation

Images as functions



Source: S. Seitz

Images as functions

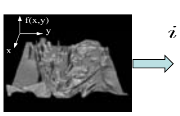
- We can think of an image as a function, *f*, from $\mathbb{R}^2$ to $\mathbb{R}$:
  - $f(x, y)$ gives the **intensity** at position (x, y)
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - $f: [a,b] \times [c,d] \rightarrow [0, 1.0]$
- A color image 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

Digital images

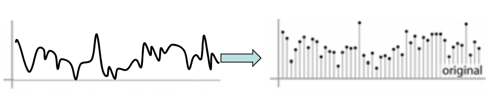
- In computer vision we operate on **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- Image thus represented as a matrix of integer values.



j →, i ↓

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

2D

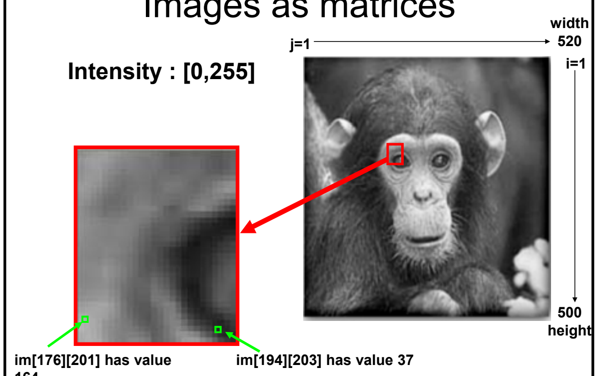


1D

Adapted from S. Seitz

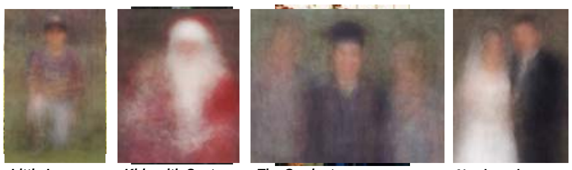
Images as matrices

Intensity : [0,255]



Images as matrices

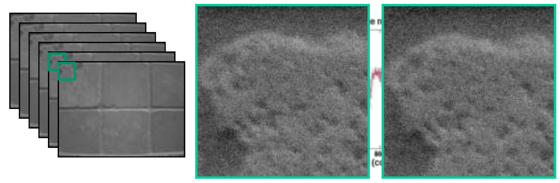
Result of averaging 100 similar snapshots



Little Leaguer *Kids with Santa* *The Graduate* *Newlyweds*

From: ***100 Special Moments*****, by Jason Salavon (2004)**
http://salavon.com/SpecialMoments/SpecialMoments.shtml

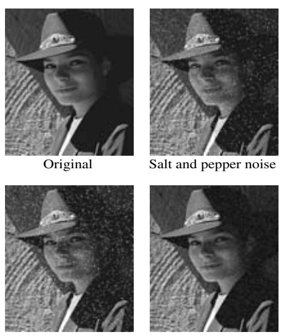
Motivation: noise reduction



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

Common types of noise

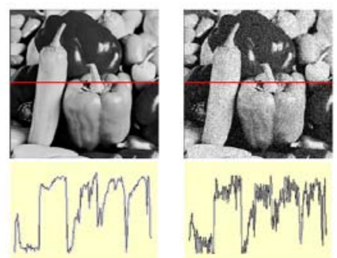
- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution



Original Salt and pepper noise
Impulse noise Gaussian noise

Source: S. Seitz

Gaussian noise



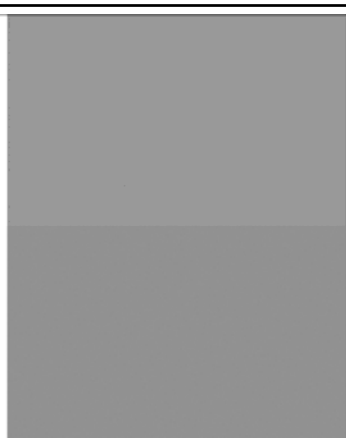
$$f(x,y) = \overbrace{\tilde{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

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

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

Fig: M. Hebert

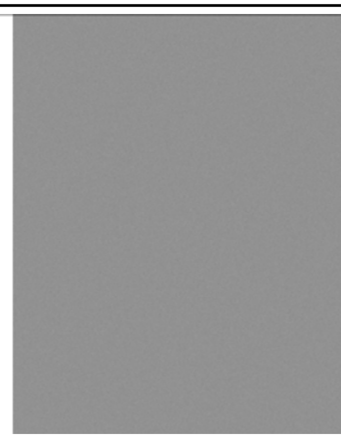
sigma=1



Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

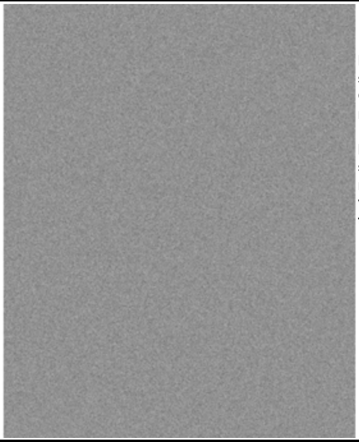
sigma=4



Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

sigma= 16



Effect of sigma on Gaussian noise:

Image shows the noise values themselves.

sigma=1

Effect of sigma on Gaussian noise:

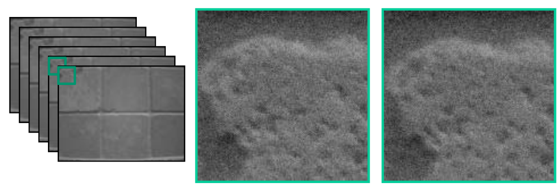
This shows the noise values added to the raw intensities of an image.

sigma=16

Effect of sigma on Gaussian noise

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

Motivation: noise reduction



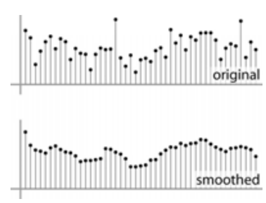
- Even multiple images of the same static scene will not be identical.
- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

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:

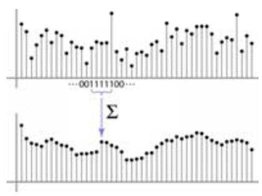


Source: S. Marschner

Weighted Moving Average

Can add weights to our moving average

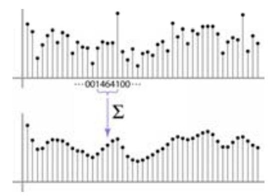
Weights [1, 1, 1, 1, 1] / 5



Source: S. Marschner

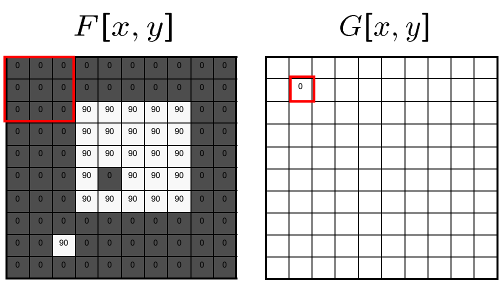
Weighted Moving Average

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



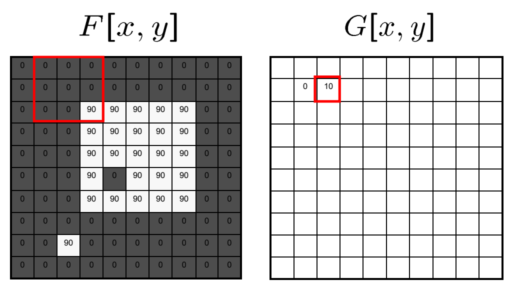
Source: S. Marschner

Moving Average In 2D



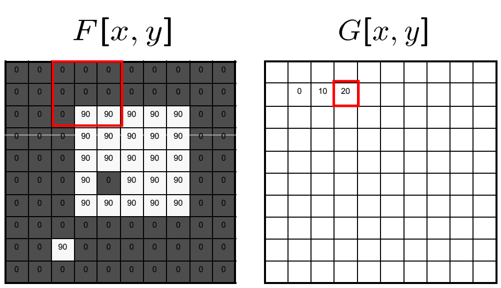
Source: S. Seitz

Moving Average In 2D



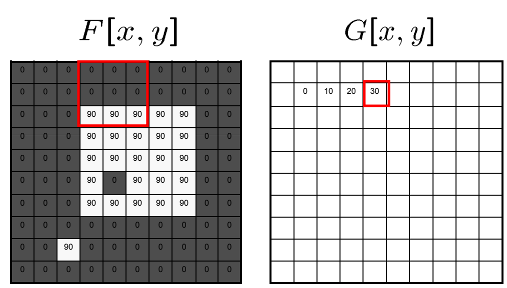
Source: S. Seitz

Moving Average In 2D



Source: S. Seitz

Moving Average In 2D



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

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

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

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

Attribute uniform weight to each pixel

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:

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

Non-uniform weights

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

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

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

$F[x, y]$

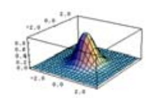
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$H[u, v]$

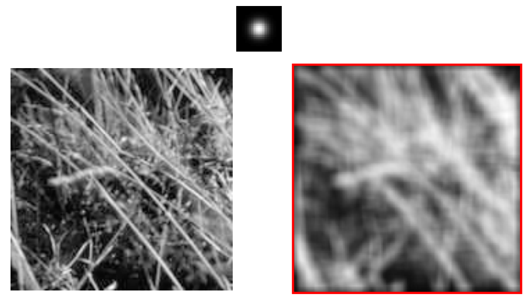
This kernel is an approximation of a Gaussian function:

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



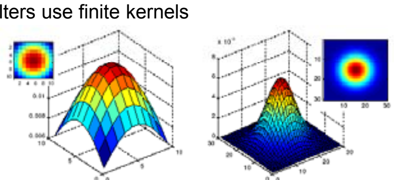
Source: S. Seitz

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

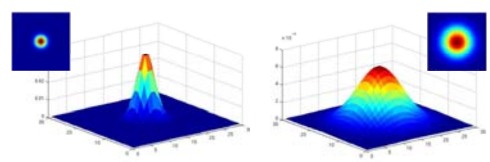


σ = 5 with 10 x 10 kernel

σ = 5 with 30 x 30 kernel

Gaussian filters

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



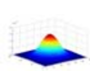
σ = 2 with 30 x 30 kernel

σ = 5 with 30 x 30 kernel

Matlab

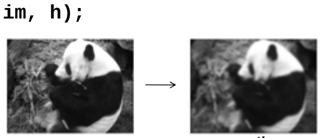
```
>> hsize = 10;
>> sigma = 5;
>> h = fspecial('gaussian' hsize, sigma);

>> mesh(h);
```



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

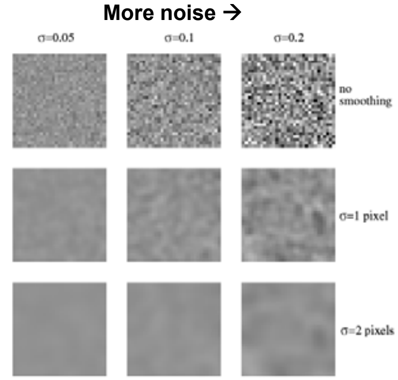
```
>> outim = imfilter(im, h);
>> imshow(outim);
```



outim

Keeping the two Gaussians in play straight…

More noise →

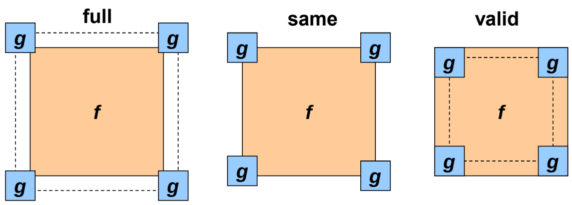


Wider smoothing kernel →

Boundary issues

What is the size of the output?

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



Source: 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

Boundary issues

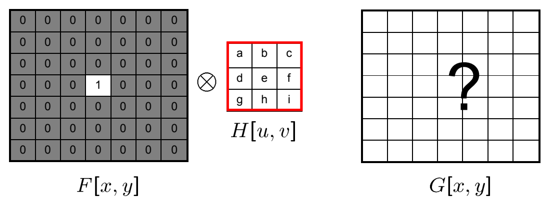
What about near the edge?

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

Source: S. Marschner

Filtering an impulse signal

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



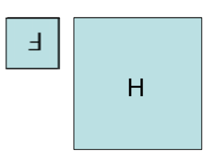
Convolution

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

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

$$G = H \star F$$

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

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

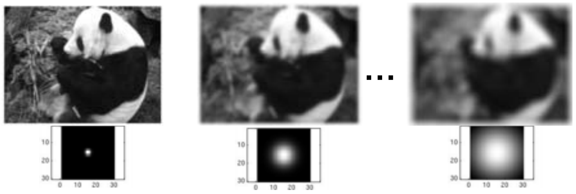
$$G = H \otimes F$$

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
```

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

Predict the filtered outputs

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

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

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

Practice with linear filters

Original

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

?

Source: D. Lowe

Practice with linear filters

Original

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

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

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

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

Shifted left by 1 pixel with correlation

Source: D. Lowe

Practice with linear filters

Original

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

Source: D. Lowe

Practice with linear filters

Original

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Blur (with a box filter)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	2	0
0	0	0

\- $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	2	0
0	0	0

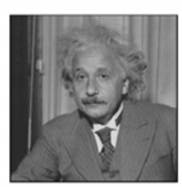
\- $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

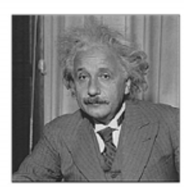
Sharpening filter
- Accentuates differences with local average

Source: D. Lowe

Filtering examples: sharpening



before



after

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:

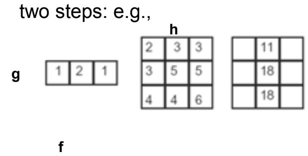
$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

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

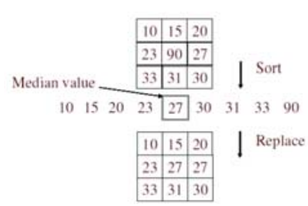
Effect of smoothing filters

5x5

Additive Gaussian noise

Salt and pepper noise

Median filter

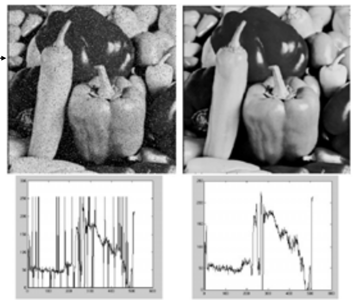


- **No new pixel values introduced**
- **Removes spikes: good for impulse, salt & pepper noise**
- **Linear?**

Median filter

Salt and pepper noise

Median filtered

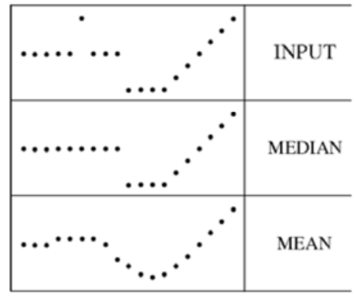


Plots of a row of the image

Source: M. Hebert

Median filter

- Median filter is edge preserving



Summary

- Various models for image “noise”
- Linear filters and convolution useful for
  - Image 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

Coming up

- **Tomorrow (Wed)**: my office hours cancelled
- TA’s available as usual
- **Thursday**:
  - Matlab tutorial, with guest lecture by Yong Jae
  - Bring questions about Pset 0
- **Monday**:
  - Pset 0 is due, 11:59 PM
- **Tuesday**:
  - Lecture: Linear filters, part 2
  - See course page for reading
  - Pset 1 out