

Convolution and Image Derivatives

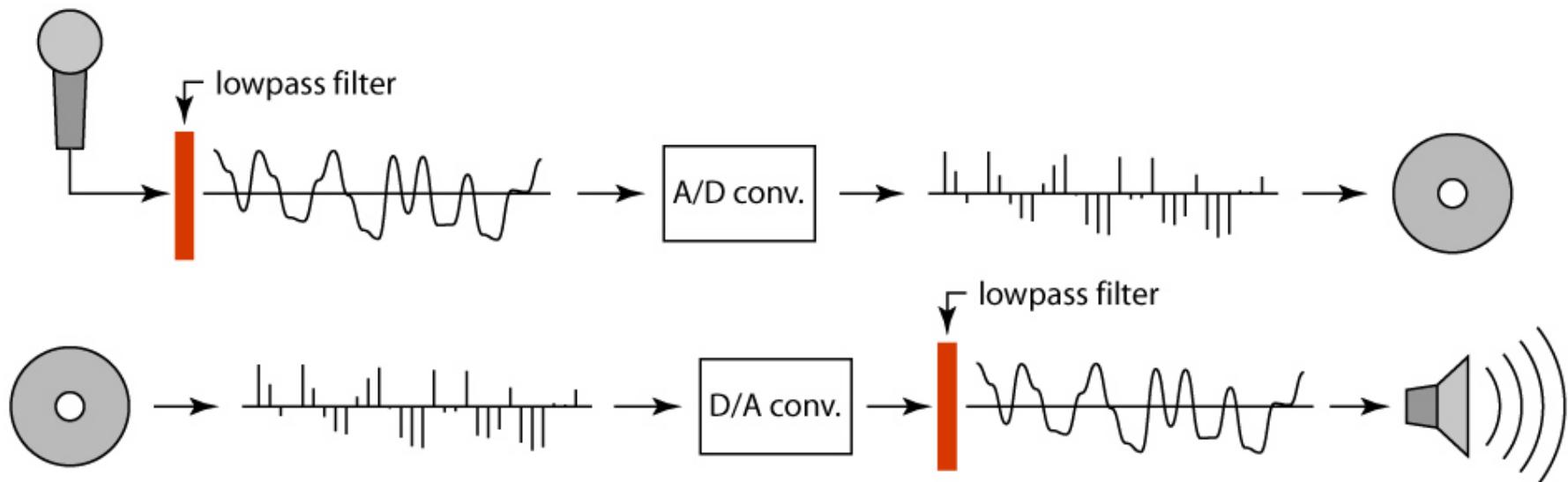


Zoltán Vódl

CS180: Intro to Comp. Vision and Comp. Photo
Alexei Efros, UC Berkeley, Fall 2024

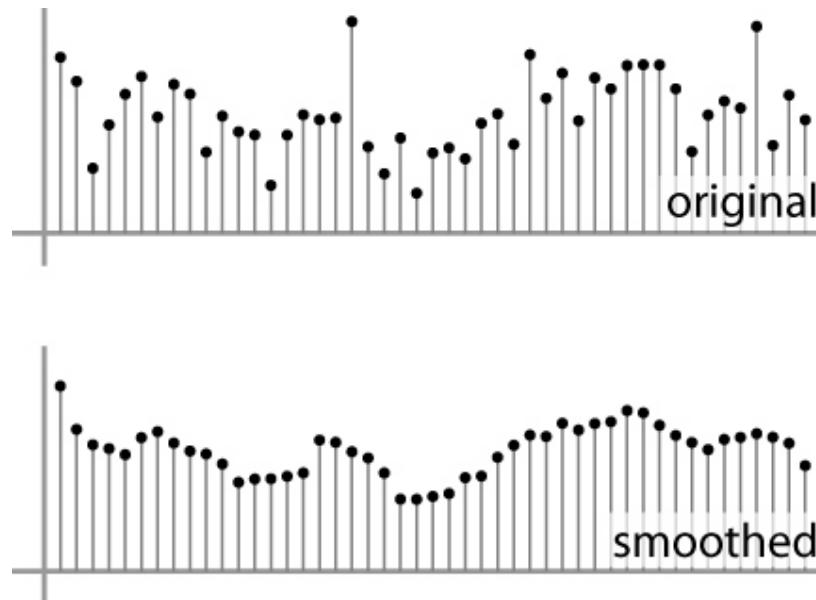
Preventing aliasing

- Introduce **lowpass filters**:
 - remove high frequencies leaving only safe, low frequencies
 - choose lowest frequency in reconstruction (disambiguate)



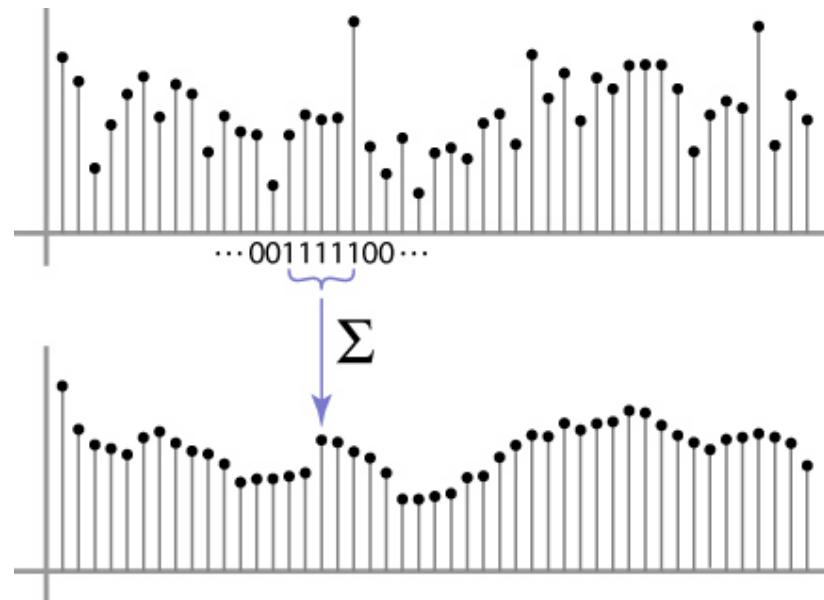
Moving Average

- basic idea: define a new function by averaging over a sliding window
- a simple example to start off: smoothing



Moving Average

- Can add weights to our moving average
- *Weights* $[..., 0, 1, 1, 1, 1, 1, 0, ...] / 5$



In 2D: box filter

$$h[\cdot, \cdot]$$

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

1	1	1
1	1	1
1	1	1

Image filtering

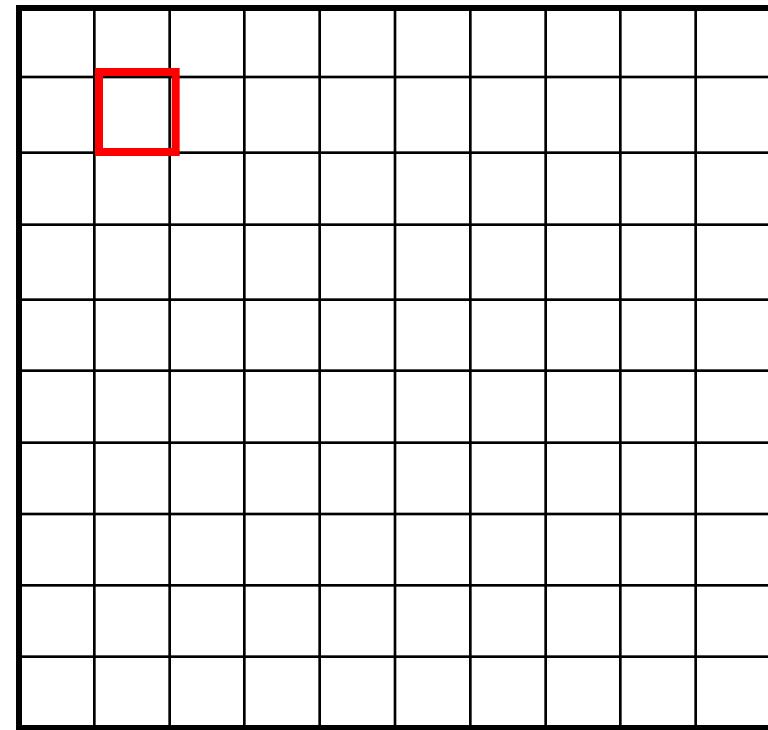
$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$g[.,.]$

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



$$g[m, n] = \sum_{k,l} h[k, l] f[m + k, n + l]$$

Credit: S. Seitz

Image filtering

$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

$g[.,.]$

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

$$g[m, n] = \sum_{k,l} h[k, l] f[m + k, n + l]$$

Credit: S. Seitz

Image filtering

$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$g[.,.]$

			0	10	20				

Image filtering

$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$g[.,.]$

			0	10	20	30			

Image filtering

$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$g[.,.]$

0 10 20 30 30

Image filtering

$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$g[.,.]$

	0	10	20	30	30				

Image filtering

$$h[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$g[.,.]$

	0	10	20	30	30					

Image filtering

$$h[\cdot, \cdot] \quad \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$f[\cdot, \cdot]$

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

$g[\cdot, \cdot]$

	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	

Cross-correlation

Let F be the image, H be the kernel (of size $2k+1 \times 2k+1$), and G be the output image

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

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

$$G = H \otimes F$$

- Can think of as a “dot product” between local neighborhood and kernel for each pixel

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$h[\cdot, \cdot]$$

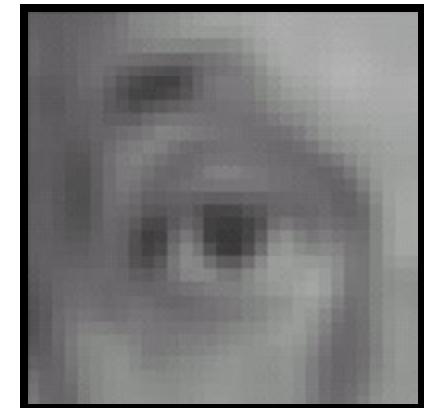
$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Linear filters: examples



Original

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



Blur (with a mean filter)

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0

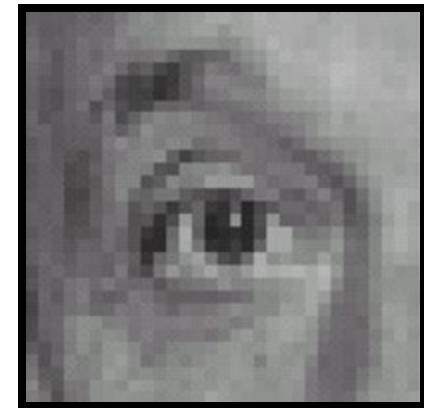
?

Practice with linear filters



Original

0	0	0
0	1	0
0	0	0



Filtered
(no change)

Practice with linear filters



0	0	0
0	0	1
0	0	0

?

Practice with linear filters



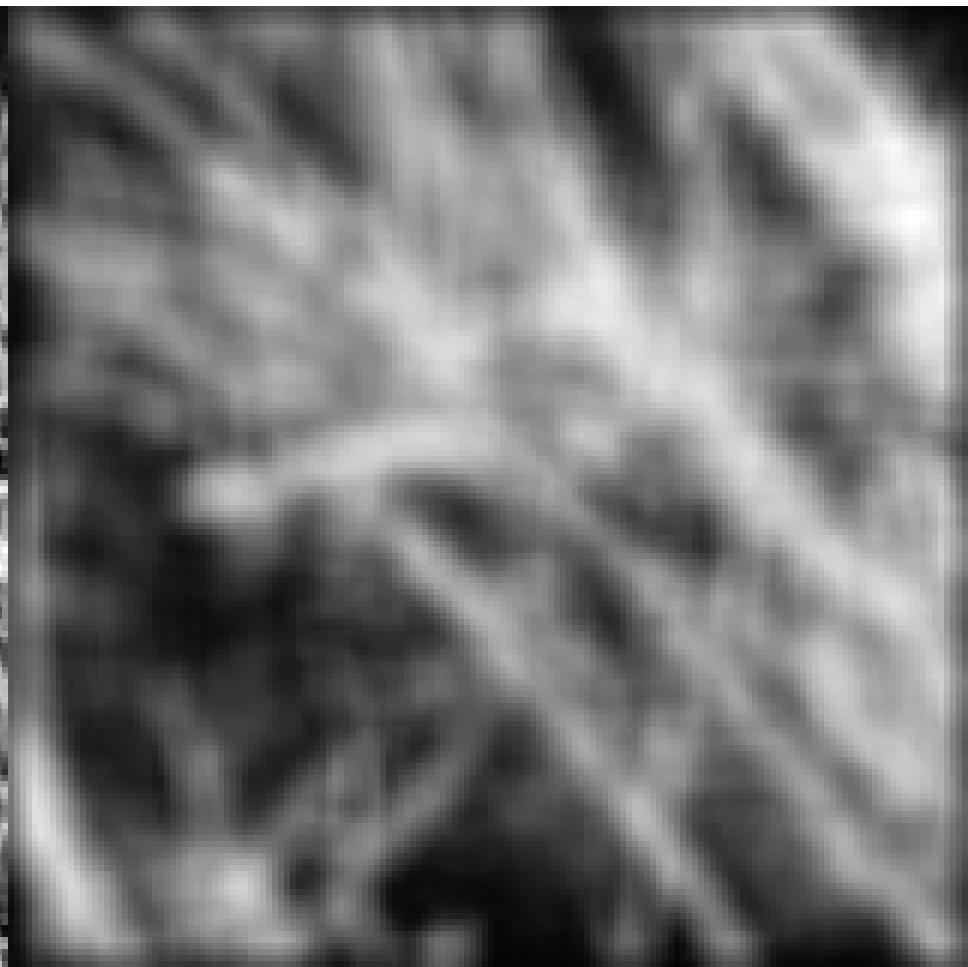
Original

0	0	0
0	0	1
0	0	0



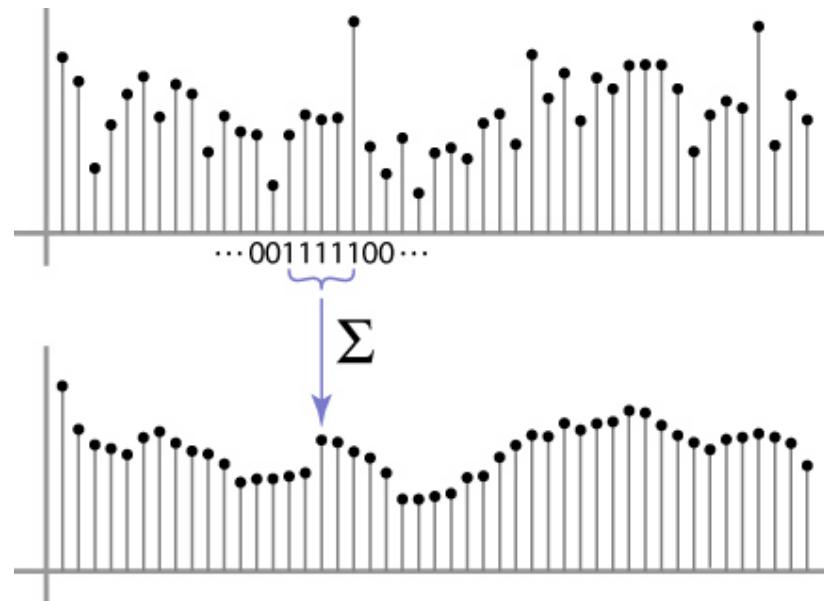
Shifted left
By 1 pixel

Back to the box filter



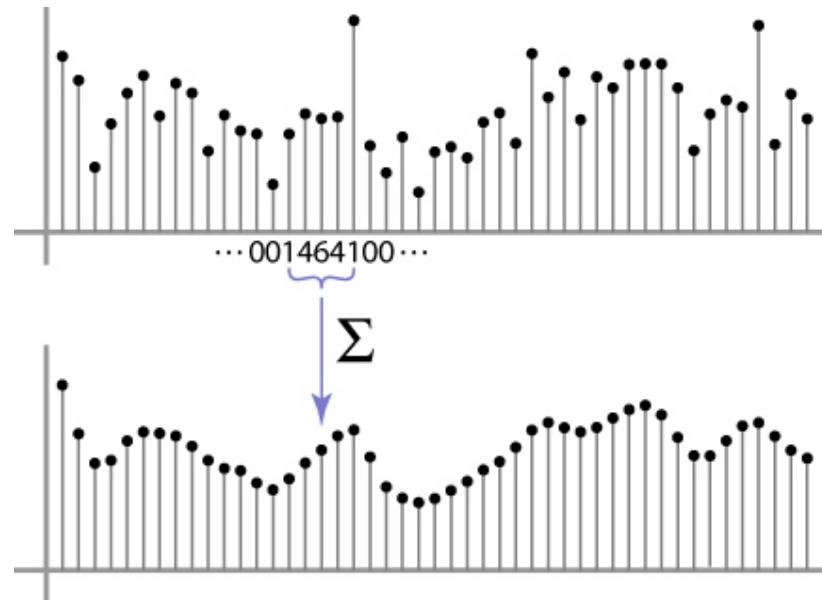
Moving Average

- Can add weights to our moving average
- *Weights* $[..., 0, 1, 1, 1, 1, 1, 0, ...] / 5$



Weighted Moving Average

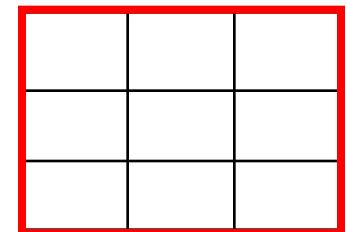
- bell curve (gaussian-like) weights $[..., 1, 4, 6, 4, 1, ...]$



Moving Average In 2D

What are the weights H?

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

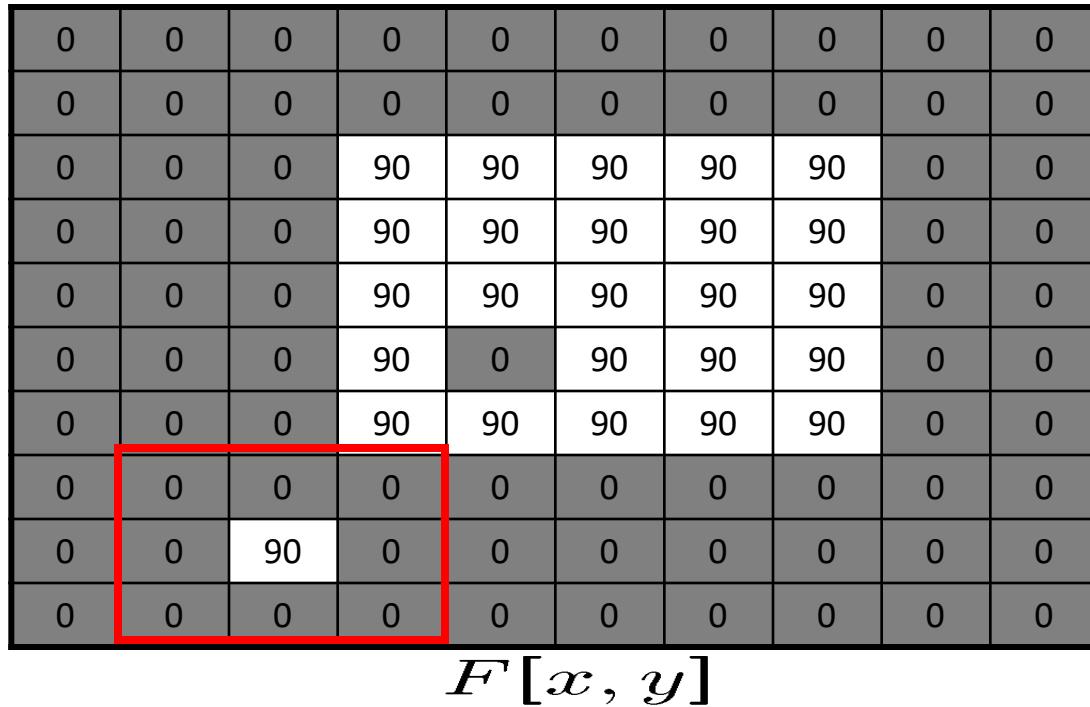


$$H[u, v]$$

$$F[x, y]$$

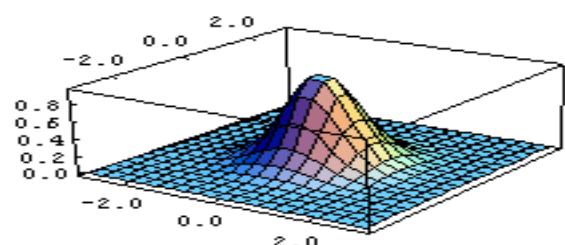
Gaussian filtering

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

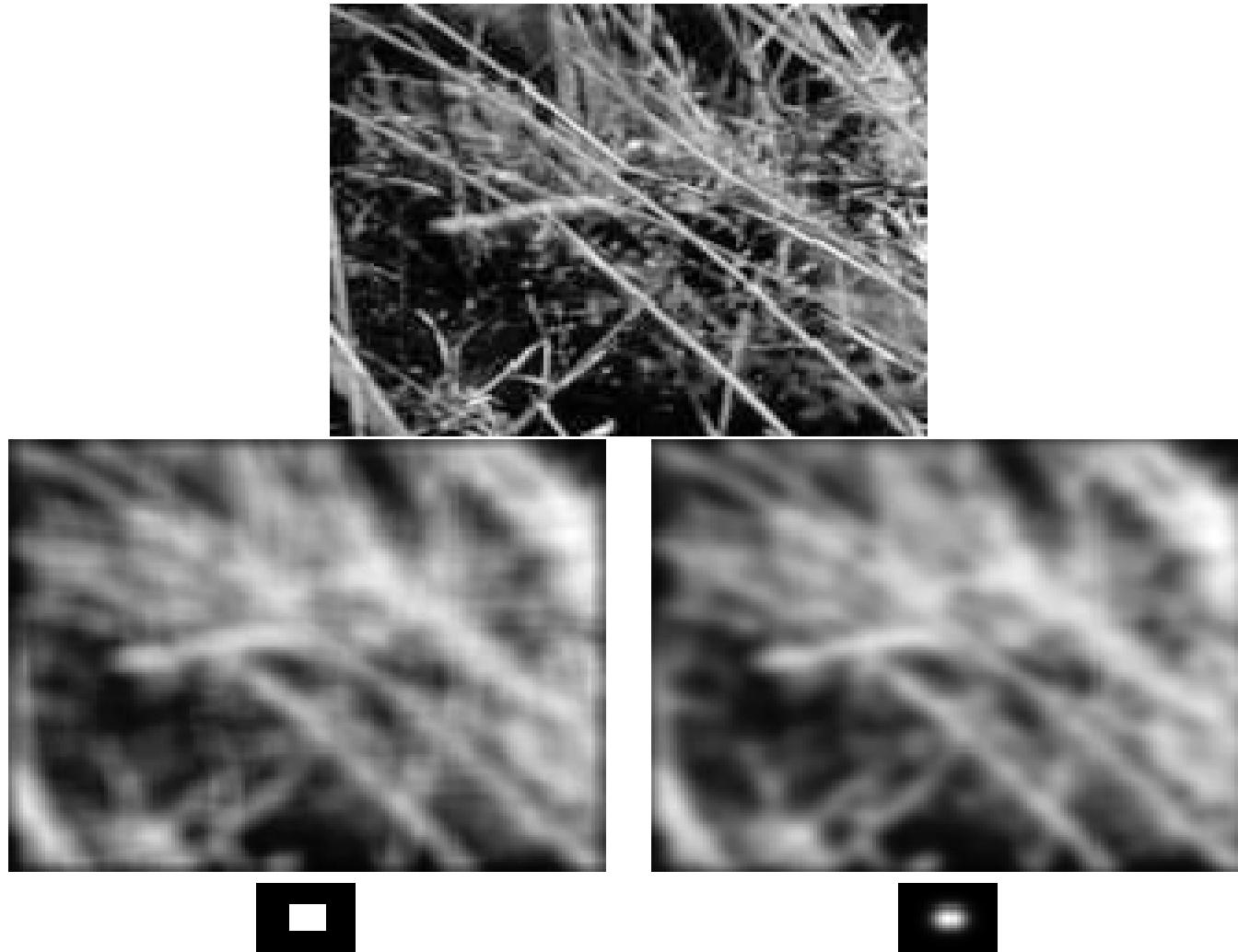


$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

This kernel is an approximation of a Gaussian function:

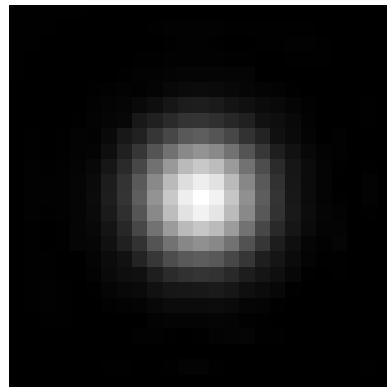
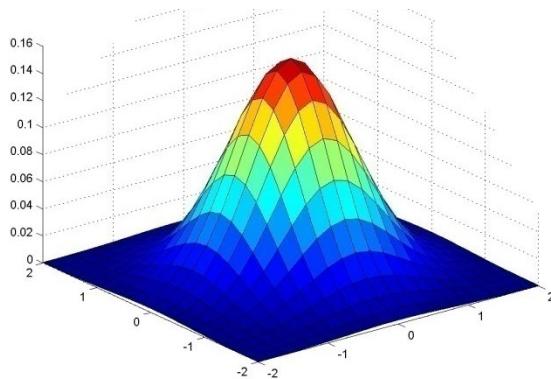


Mean vs. Gaussian filtering



Important filter: Gaussian

Weight contributions of neighboring pixels by nearness



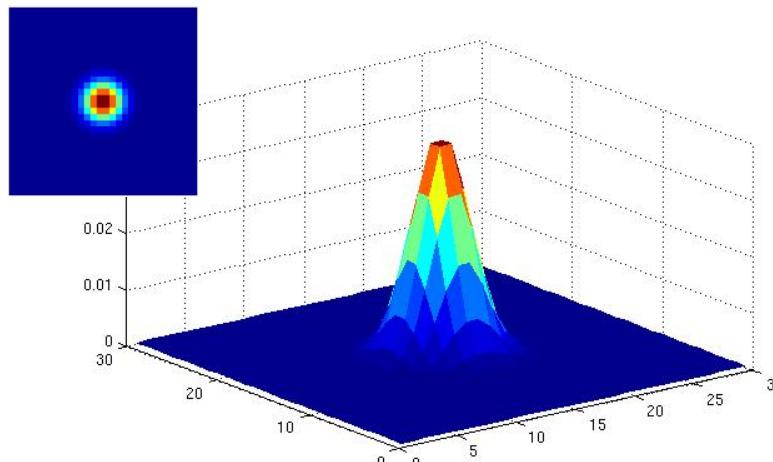
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

$5 \times 5, \sigma = 1$

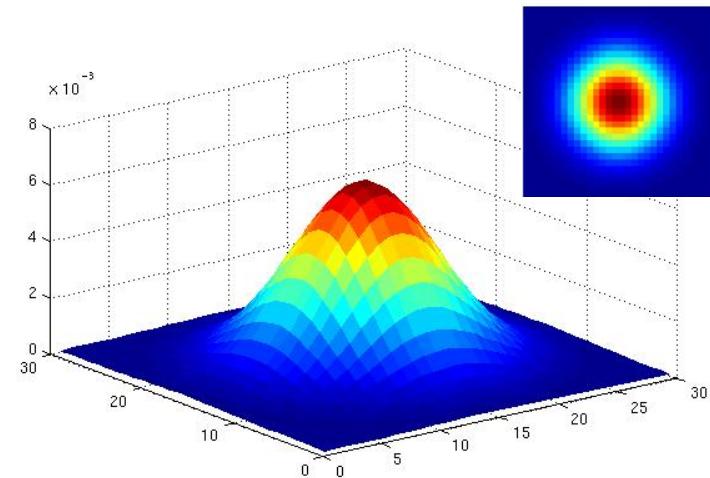
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Gaussian Kernel

$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



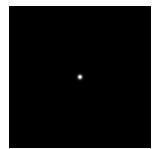
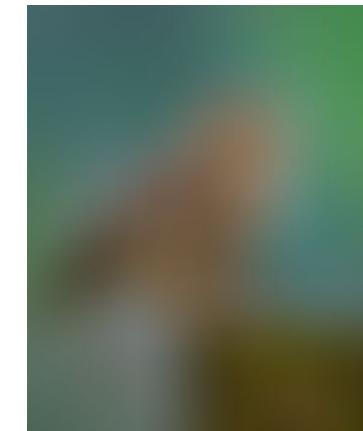
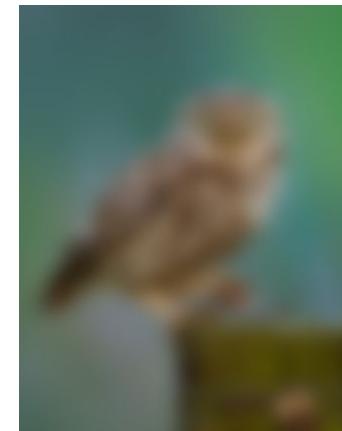
$\sigma = 2$ with 30×30 kernel



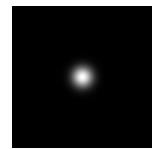
$\sigma = 5$ with 30×30 kernel

- Standard deviation σ : determines extent of smoothing

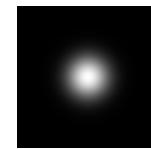
Gaussian filters



$\sigma = 1$ pixel



$\sigma = 5$ pixels



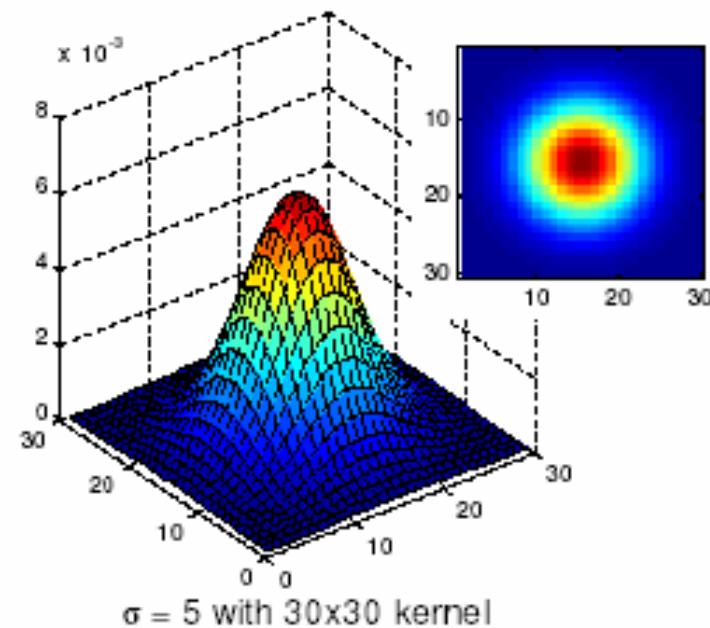
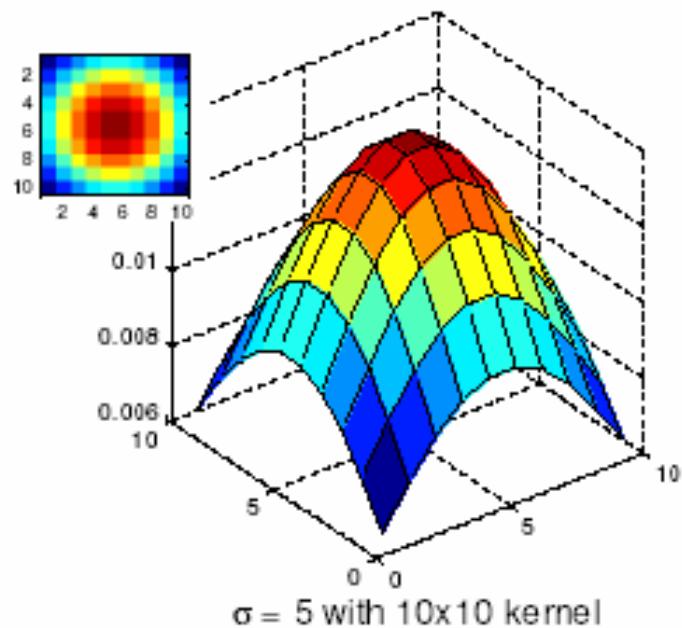
$\sigma = 10$ pixels



$\sigma = 30$ pixels

Choosing kernel width

- The Gaussian function has infinite support, but discrete filters use finite kernels

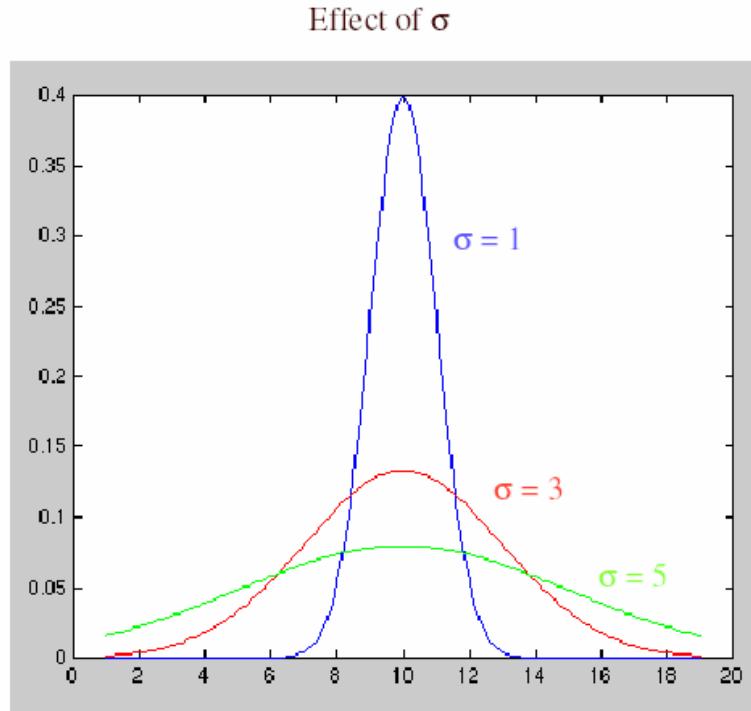


Practical matters

How big should the filter be?

Values at edges should be near zero

Rule of thumb for Gaussian: set filter half-width to about 3σ



Cross-correlation vs. Convolution

cross-correlation: $G = H \otimes F$

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

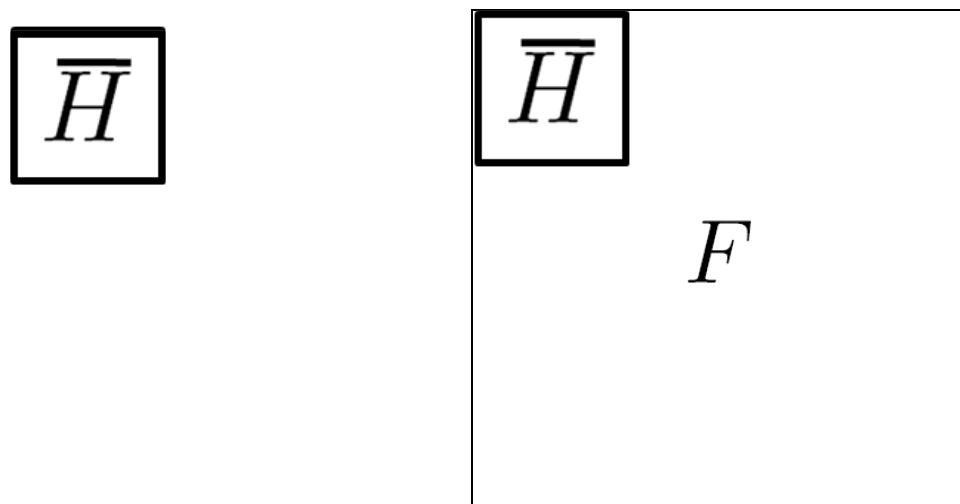
A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

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

It is written:

$$G = H \star F$$

Convolution



Cross-correlation vs. Convolution

cross-correlation: $G = H \otimes F$

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

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:

$$G = H \star F$$

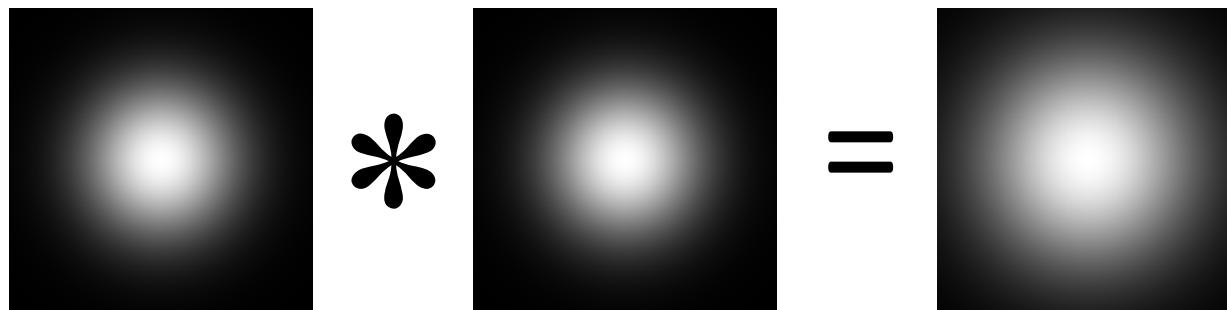
Convolution is **commutative** and **associative**

Convolution is nice!

- Notation: $b = c \star a$
- Convolution is a multiplication-like operation
 - commutative $a \star b = b \star a$
 - associative $a \star (b \star c) = (a \star b) \star c$
 - distributes over addition $a \star (b + c) = a \star b + a \star c$
 - scalars factor out $\alpha a \star b = a \star \alpha b = \alpha(a \star b)$
 - identity: unit impulse $e = [..., 0, 0, 1, 0, 0, ...]$
$$a \star e = a$$
- Conceptually no distinction between filter and signal
- Usefulness of associativity
 - often apply several filters one after another: $((a * b_1) * b_2) * b_3$
 - this is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

Gaussian and convolution

- Removes “high-frequency” components from the image (low-pass filter)
- Convolution with self is another Gaussian



- Convolving twice with Gaussian kernel of width σ
 $=$ convolving once with kernel of width $\sigma\sqrt{2}$

Image half-sizing

This image is too big to fit on the screen. How can we reduce it?

How to generate a half-sized version?

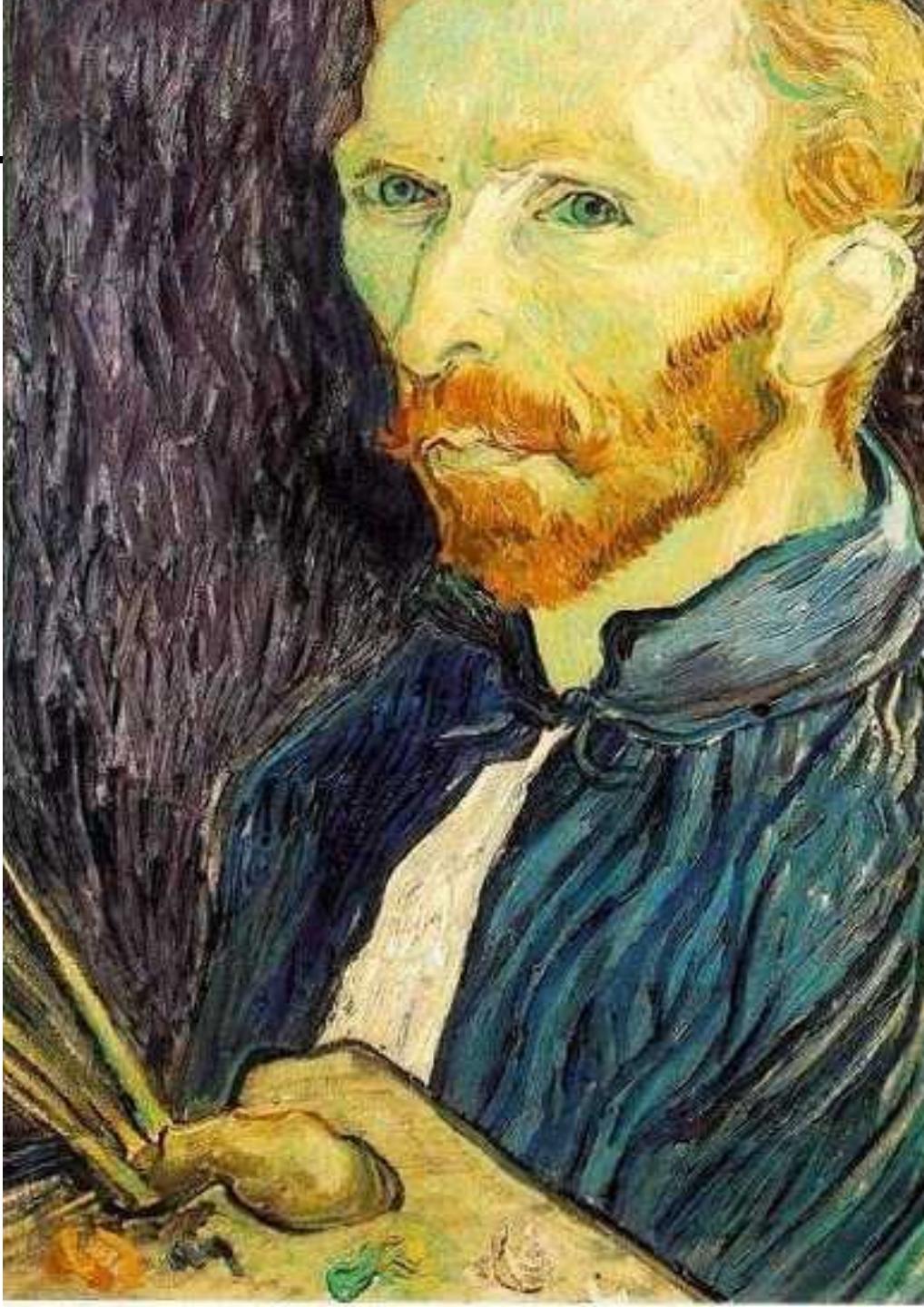
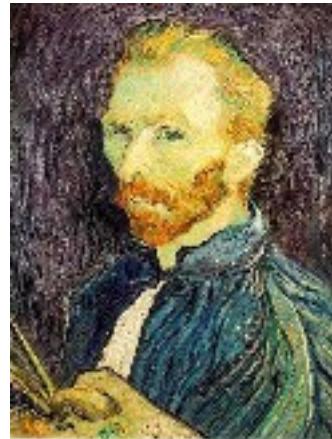
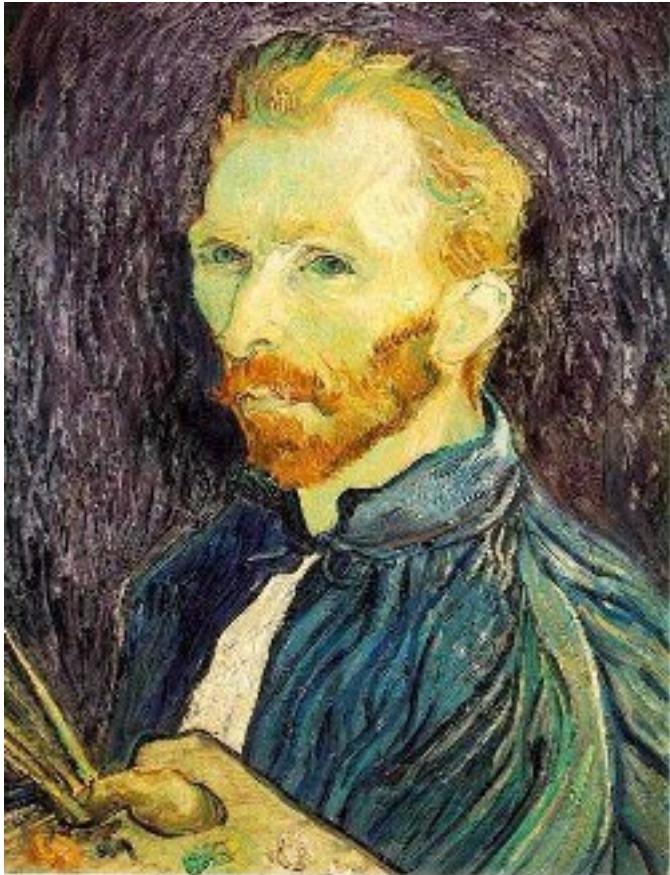


Image sub-sampling



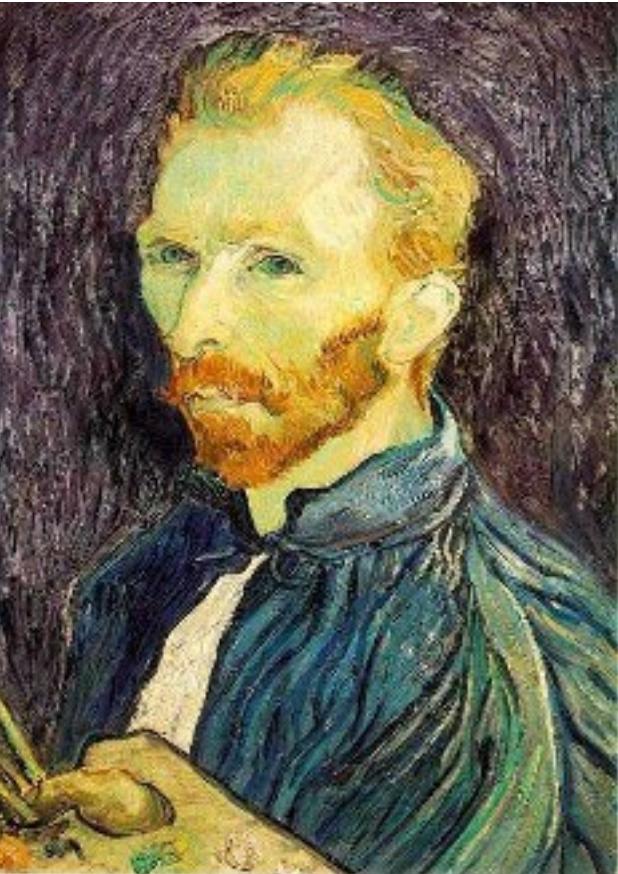
1/4



1/8

Throw away every other row and
column to create a $1/2$ size image
- called *image sub-sampling*

Image sub-sampling



1/2



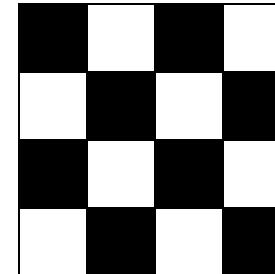
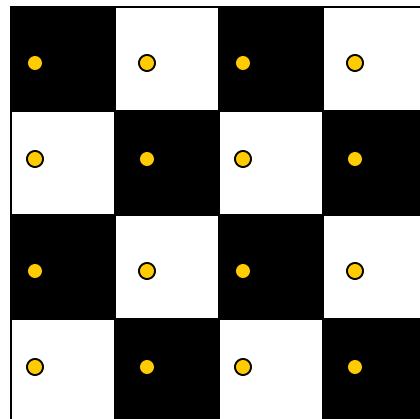
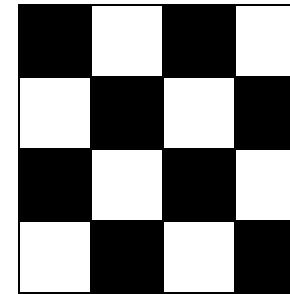
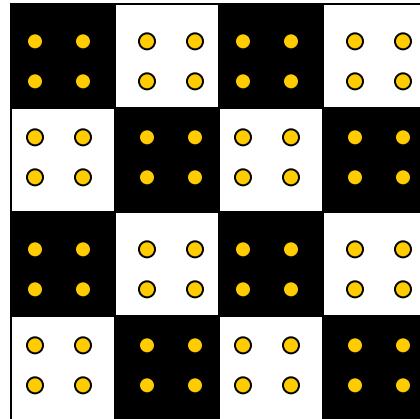
1/4 (2x zoom)



1/8 (4x zoom)

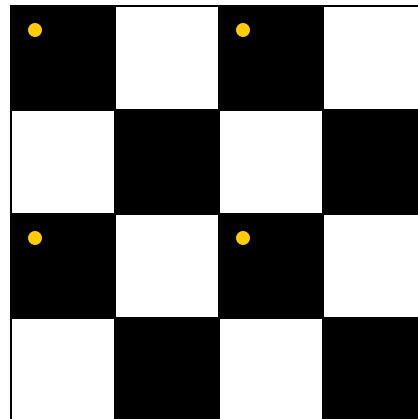
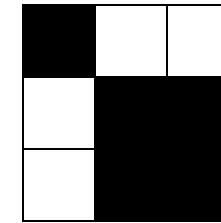
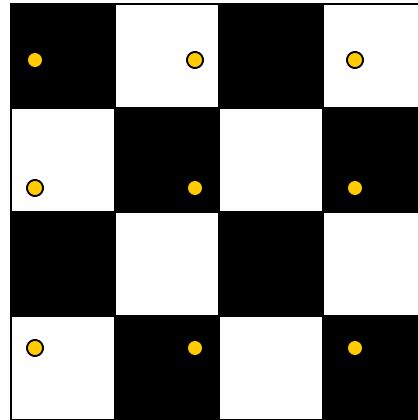
Aliasing! What do we do?

Sampling an image



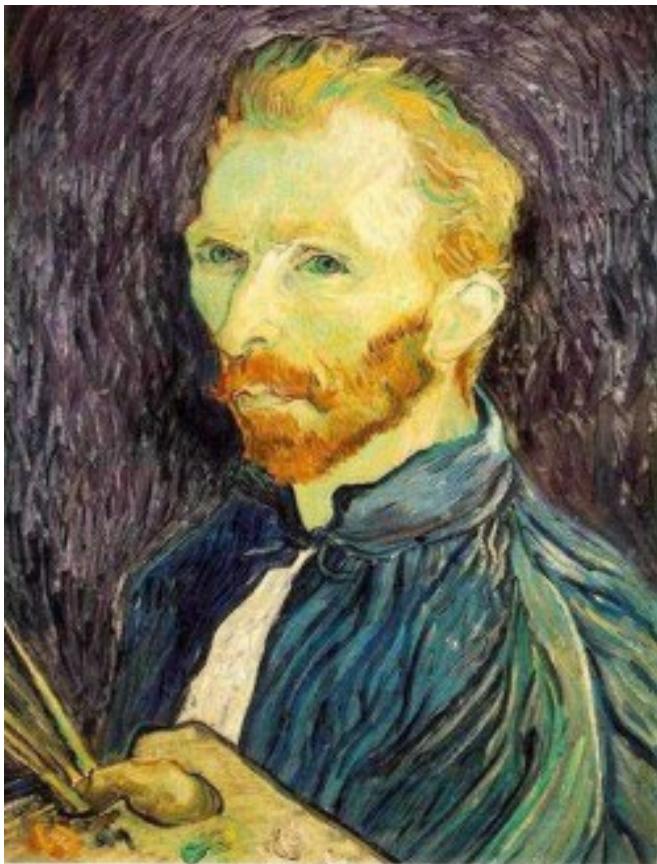
Examples of GOOD sampling

Undersampling

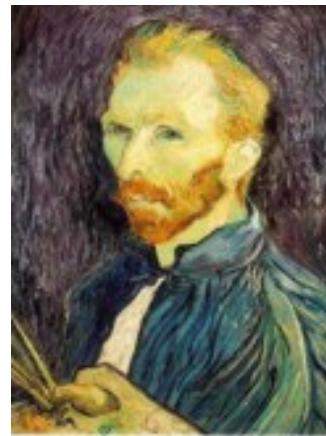


Examples of BAD sampling -> Aliasing

Gaussian (lowpass) pre-filtering



Gaussian 1/2



G 1/4

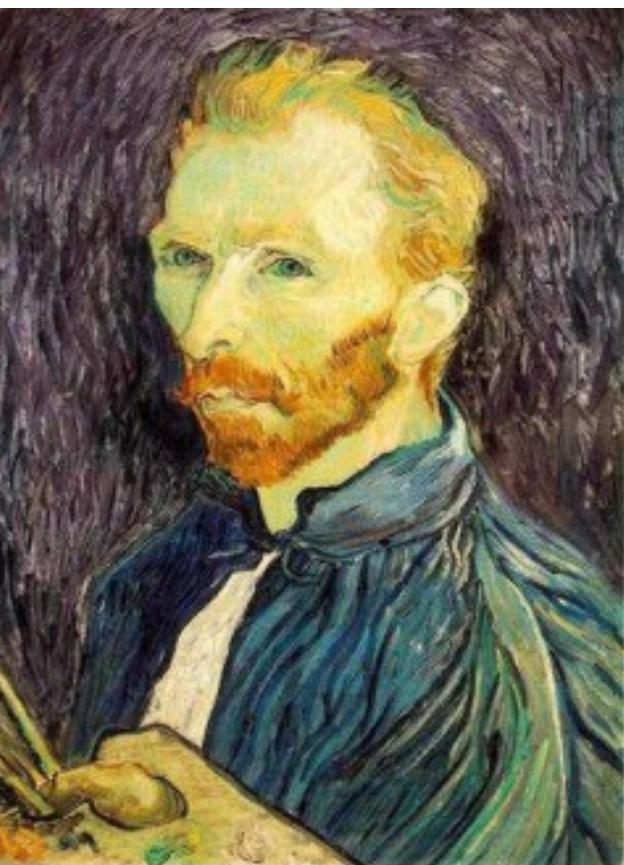


G 1/8

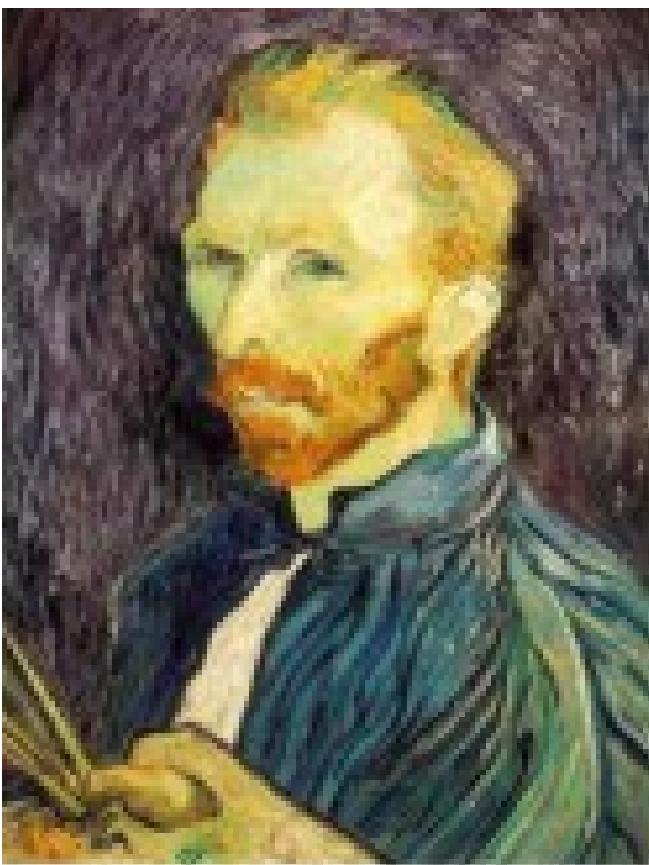
Solution: filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?

Subsampling with Gaussian pre-filtering



Gaussian 1/2

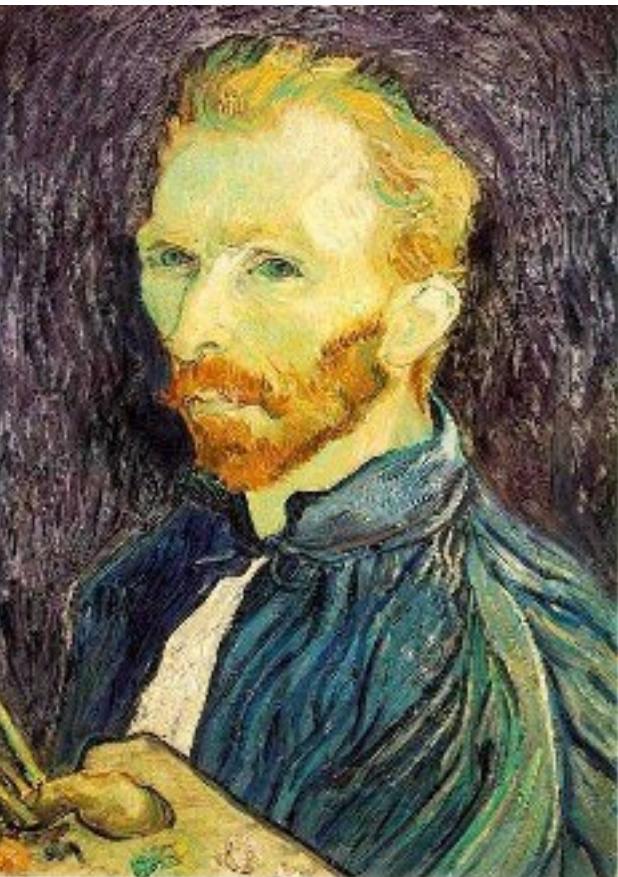


G 1/4

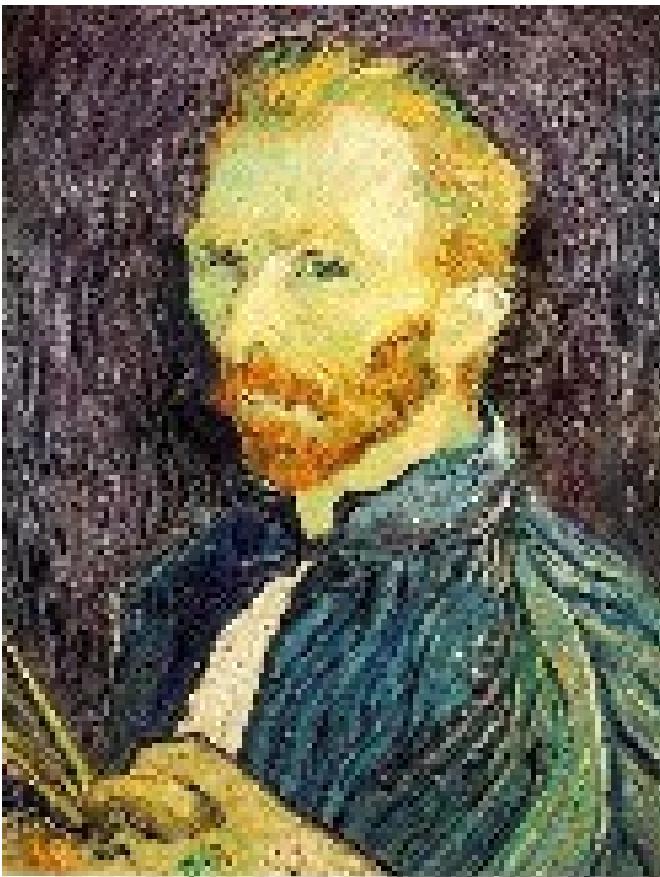


G 1/8

Compare with...



1/2

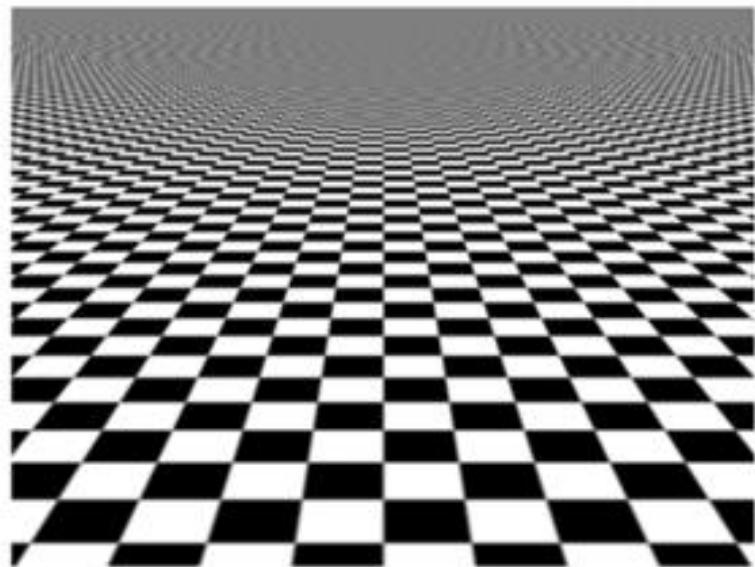
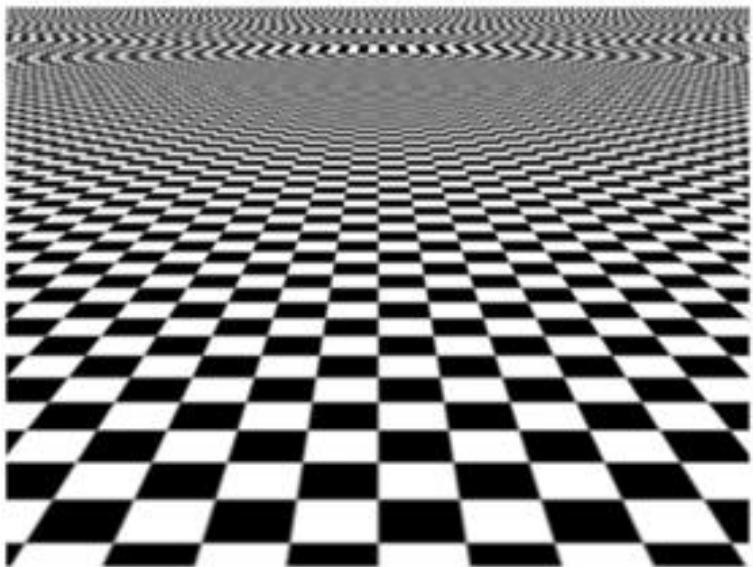


1/4 (2x zoom)



1/8 (4x zoom)

More Gaussian pre-filtering



A real problem!

128 x 128 → 64x64



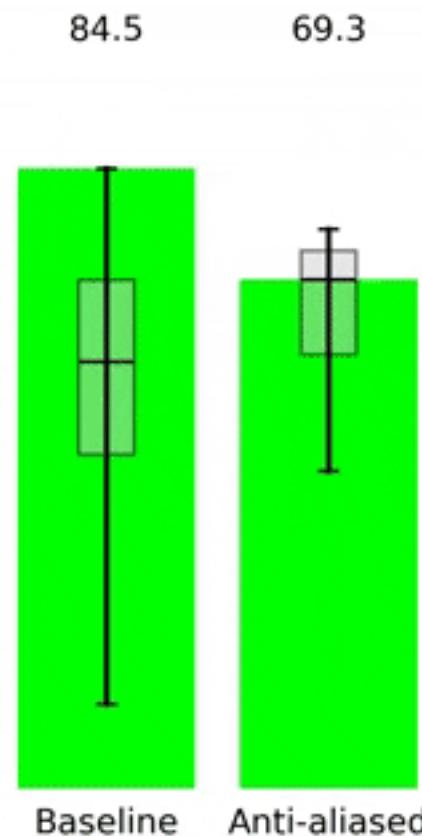
Open-CV:
default, bicubic, Lanczos4

Pytorch:
bilinear, bicubic

PIL: Lanczos

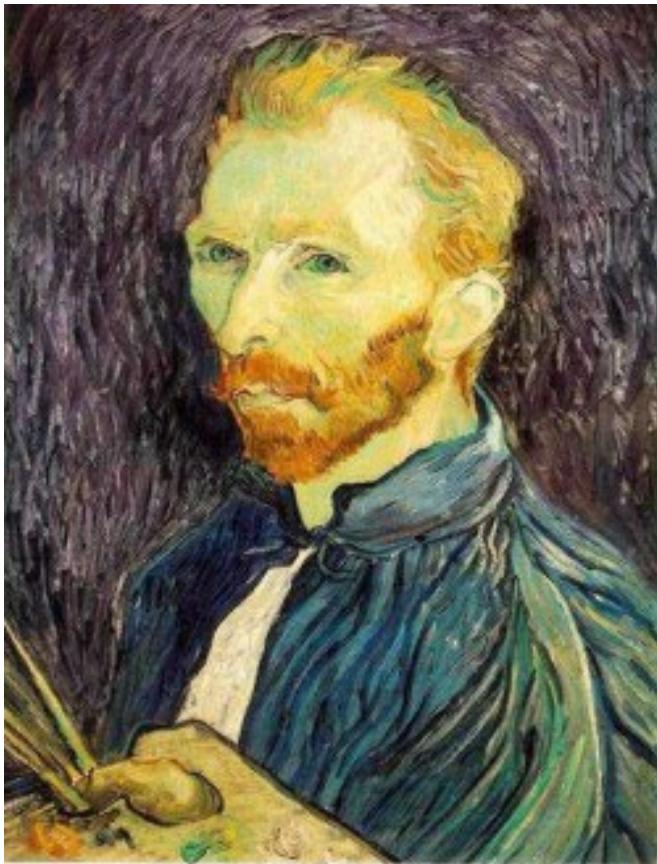
Credit: [@jaakkolehtinen](#)

problems in ConvNets too

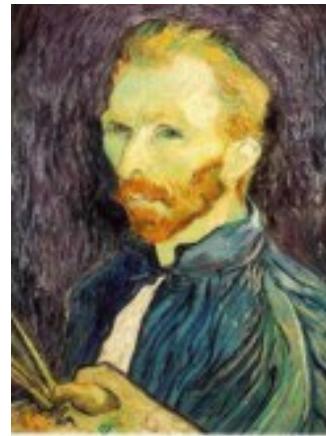


pip install antialiased-cnns

Iterative Gaussian (lowpass) pre-filtering



Gaussian 1/2



G 1/4



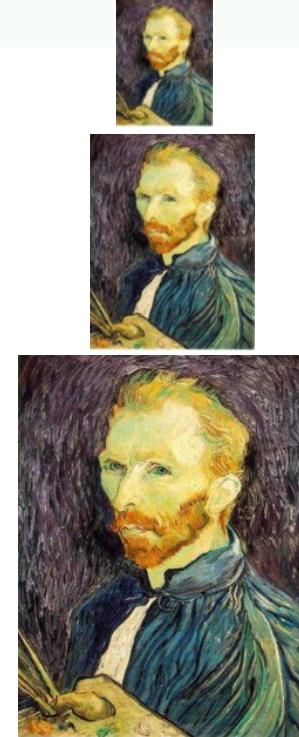
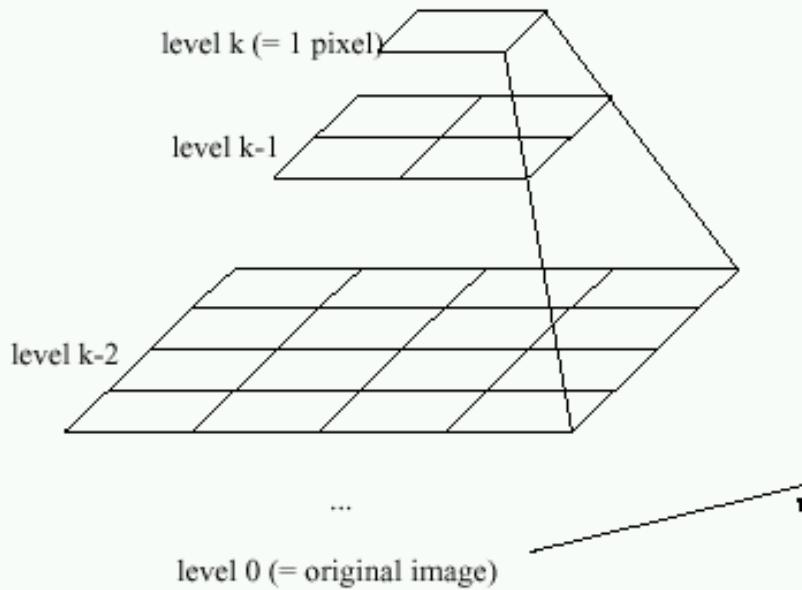
G 1/8

filter the image, *then* subsample

- Filter size should double for each $\frac{1}{2}$ size reduction. Why?
- How can we speed this up?

Image Pyramids

Idea: Represent $N \times N$ image as a “pyramid” of $1 \times 1, 2 \times 2, 4 \times 4, \dots, 2^k \times 2^k$ images (assuming $N = 2^k$)



Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*



512

256

128

64

32

16

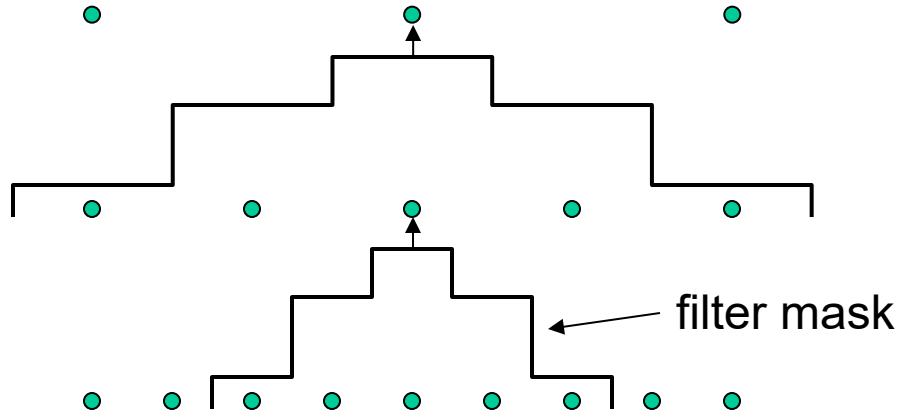
8

A bar in the big images is a hair on the zebra's nose; in smaller images, a stripe; in the smallest, the animal's nose



The whole pyramid is only $\frac{4}{3}$ the size of the original image!

Gaussian pyramid construction



Repeat

- Filter
- Subsample

Until minimum resolution reached

- can specify desired number of levels (e.g., 3-level pyramid)

What are they good for?

Improve Search

- Search over translations
 - Classic coarse-to-fine strategy
 - Project 1!
- Search over scale
 - Template matching
 - E.g. find a face at different scales

What else are convolutions good for?

Taking derivative by convolution
(on board)

Partial derivatives with convolution

Image is function $f(x, y)$

Remember:

$$\frac{\partial f(x, y)}{\partial x} = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon, y) - f(x, y)}{\epsilon}$$

Approximate:

-1	1
----	---

Another one:

-1	0	1
----	---	---

$$\frac{\partial f(x, y)}{\partial x} \approx \frac{f(x + 1, y) - f(x, y)}{1}$$

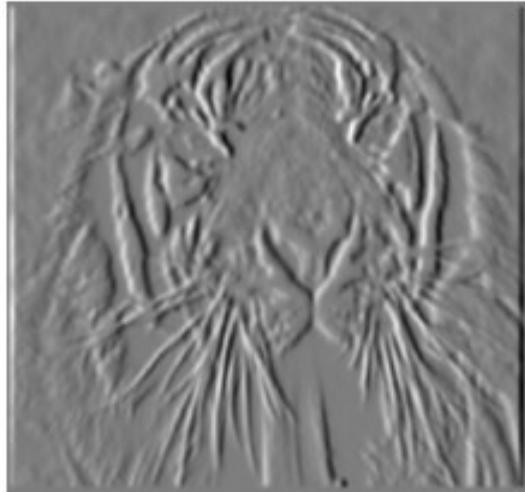
$$\frac{\partial f(x, y)}{\partial x} \approx \frac{f(x + 1, y) - f(x - 1, y)}{2}$$

Partial derivatives of an image



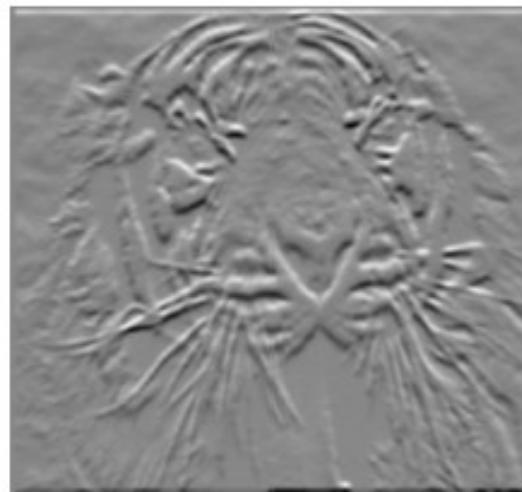
$$\frac{\partial f(x, y)}{\partial x}$$

-1	1
----	---



$$\frac{\partial f(x, y)}{\partial y}$$

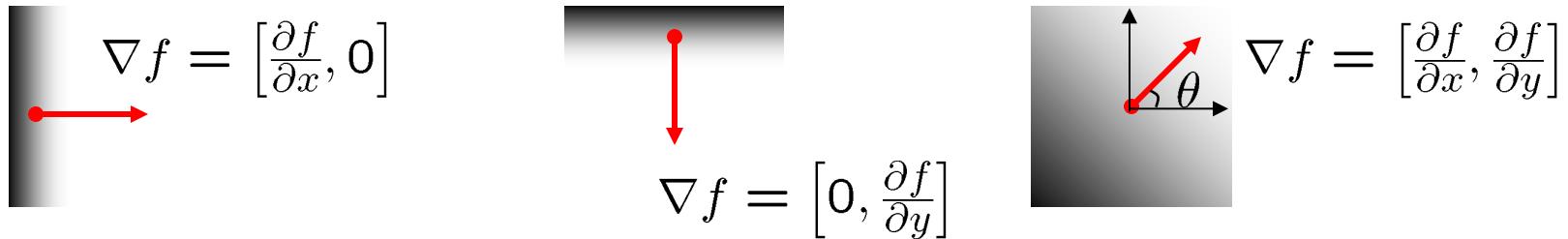
-1	or	1
1		-1



Which shows changes with respect to x?

Image gradient

The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$



The gradient points in the direction of most rapid increase in intensity

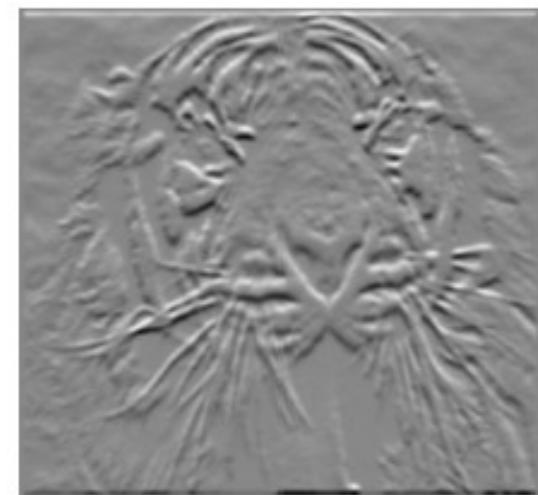
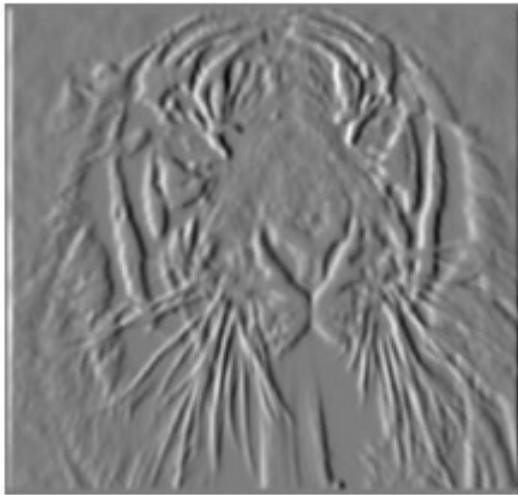
- How does this direction relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

The gradient direction is given by $\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$

Image Gradient



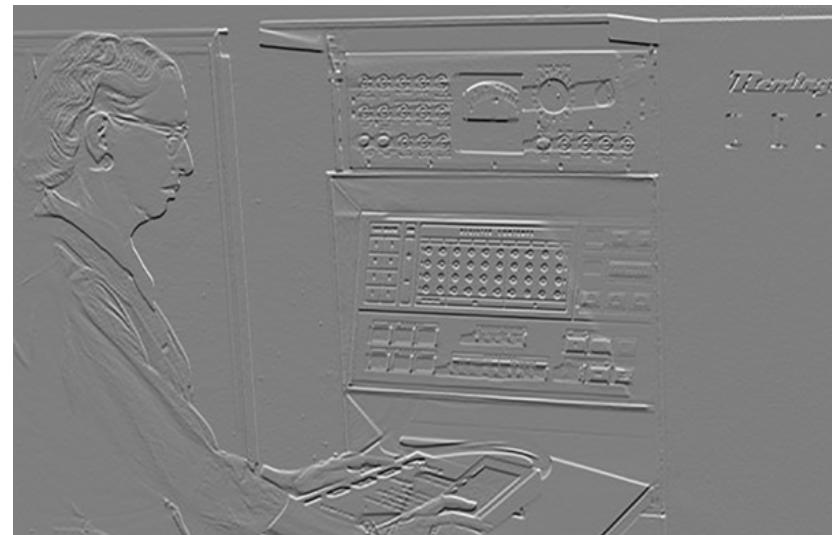
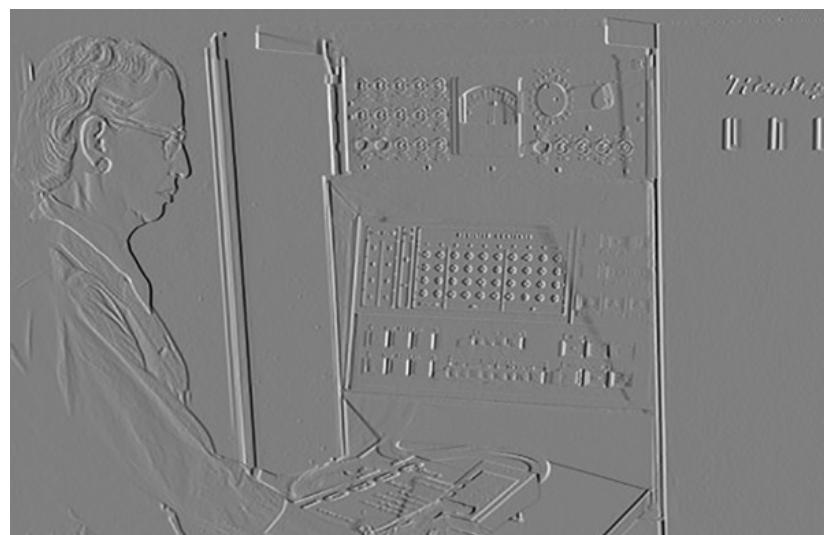
$$\frac{\partial f(x, y)}{\partial x}$$

$$\frac{\partial f(x, y)}{\partial y}$$



$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Partial Derivatives



$$\frac{\partial f(x, y)}{\partial x}$$

$$\frac{\partial f(x, y)}{\partial y}$$

Gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



Gradient Orientation

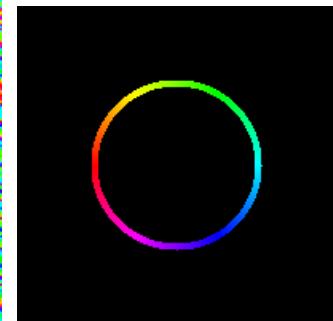
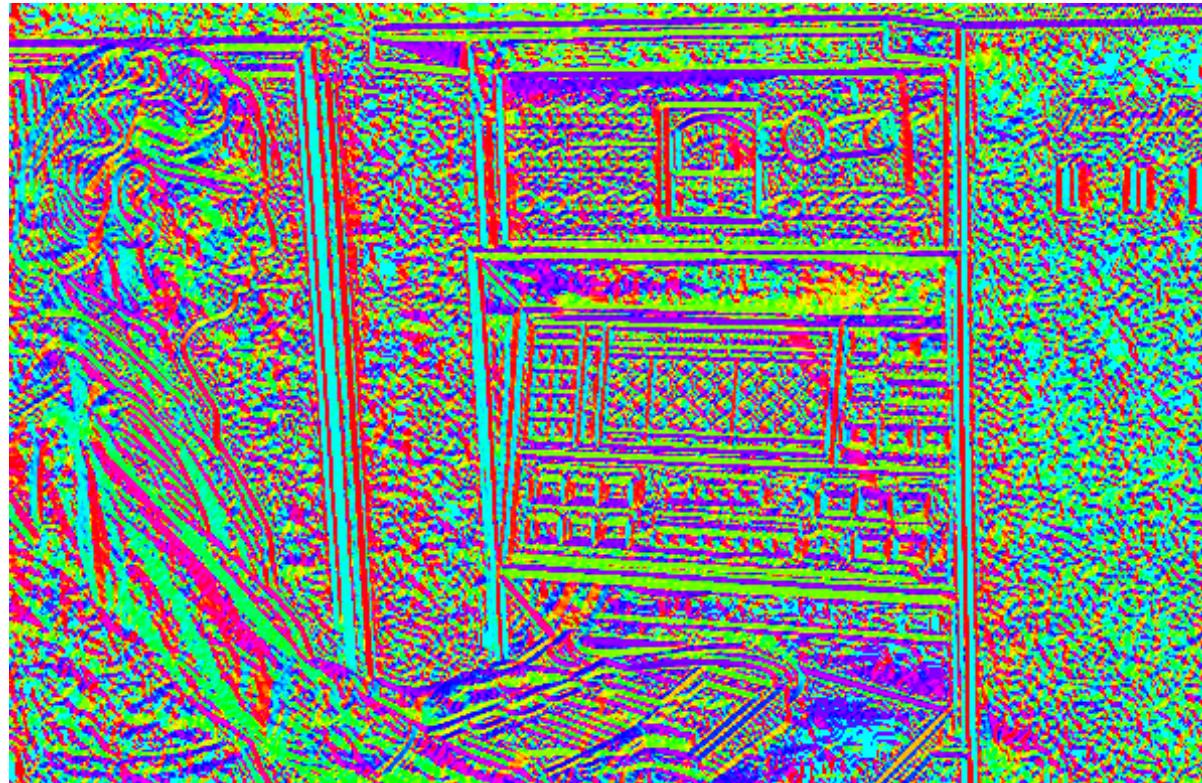
$$\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \text{ atan2}(dy, dx)$$



lightness is equal to gradient magnitude

Image Gradient

$$\theta = \tan^{-1} \left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right)$$

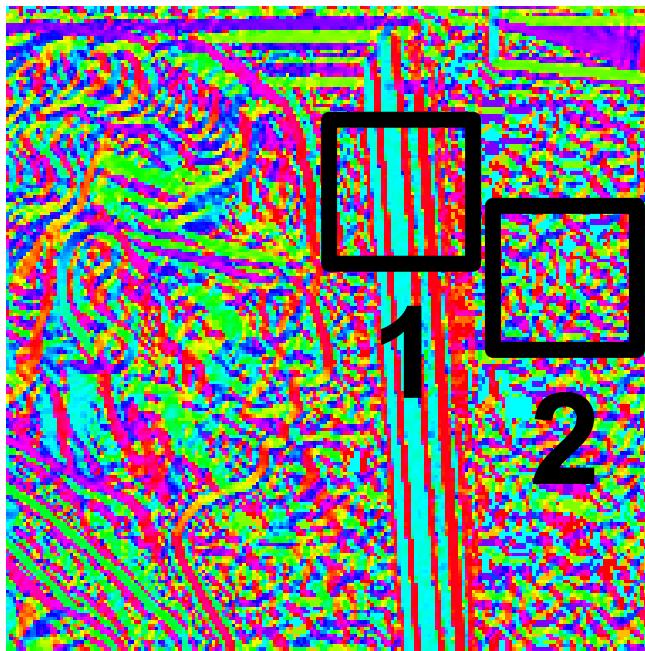


all the gradients

Source: D. Fouhey

Image Gradient

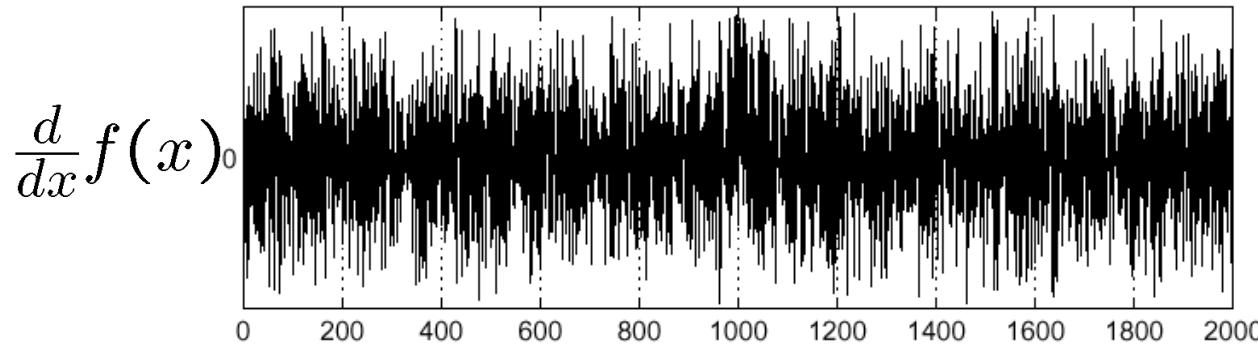
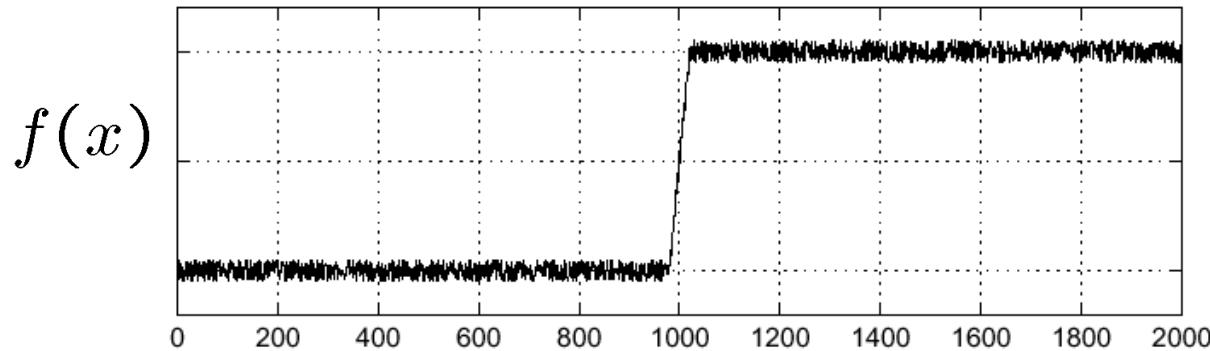
Why is there structure at 1 and not at 2?



Effects of noise

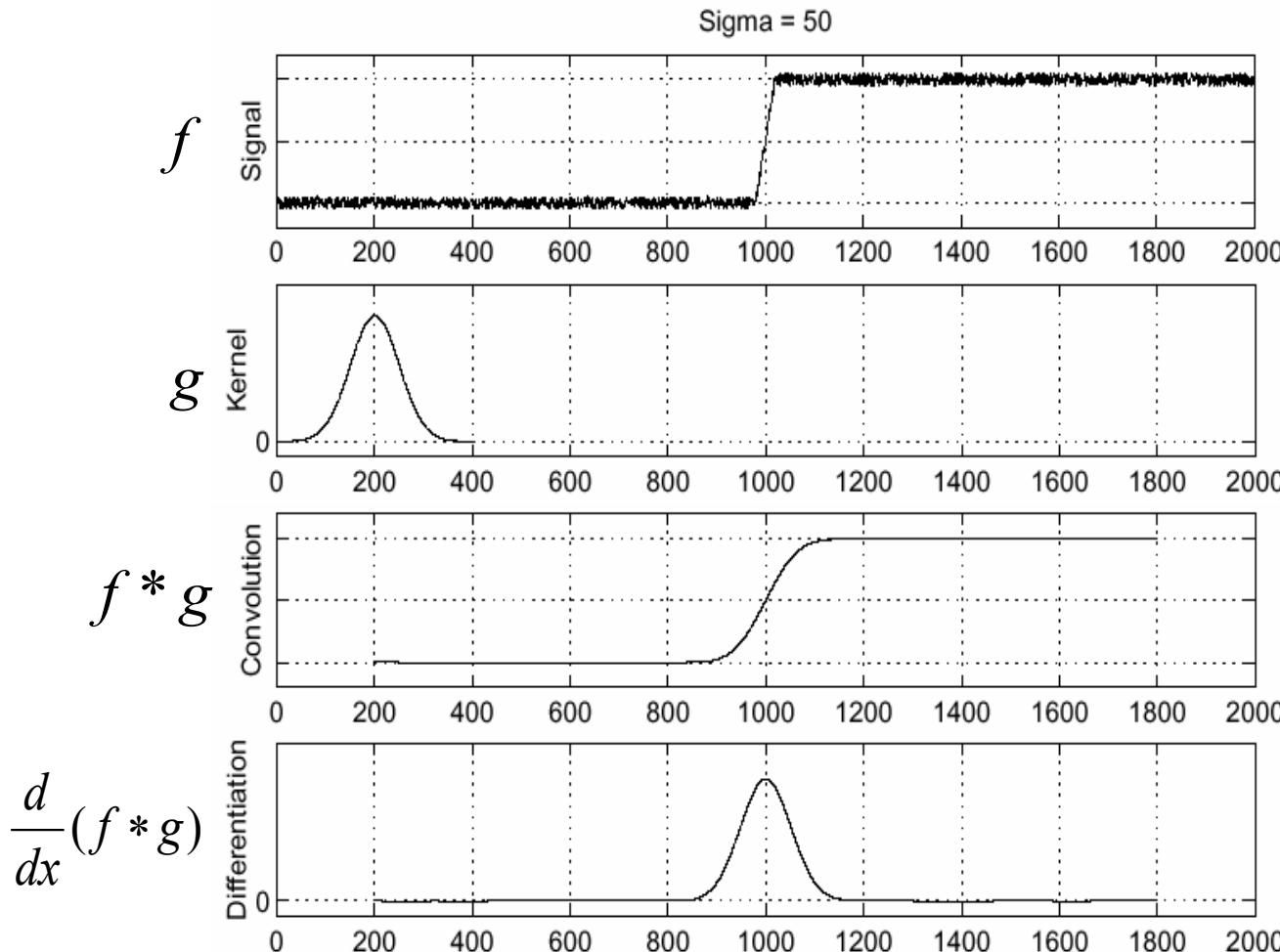
Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal



Where is the edge?

Solution: smooth first

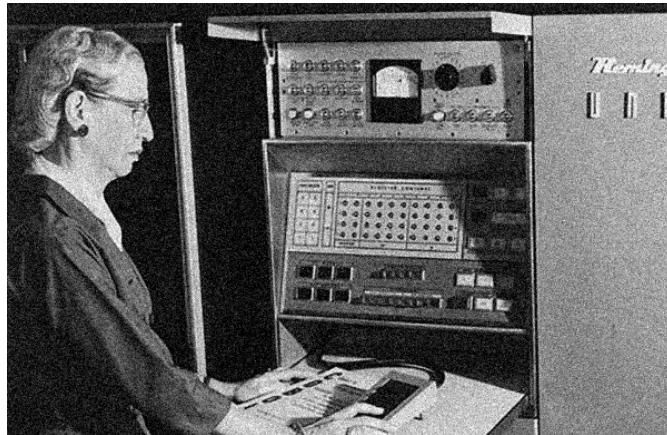


- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

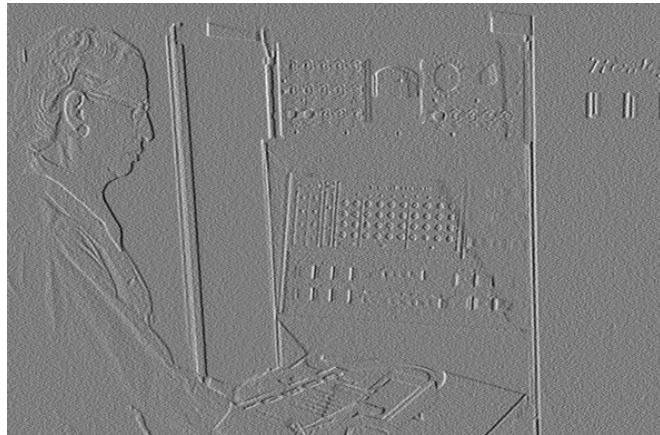
Source: S. Seitz

Noise in 2D

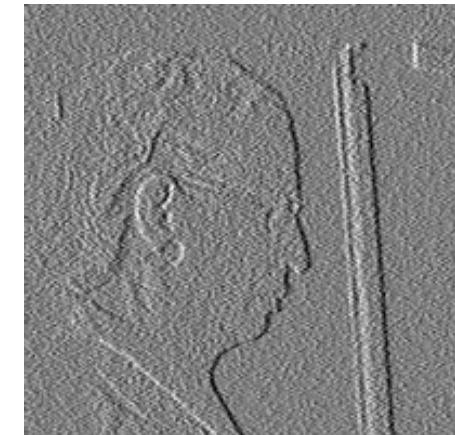
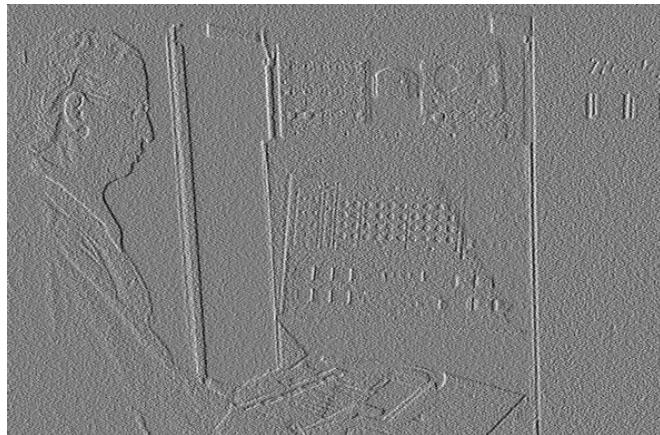
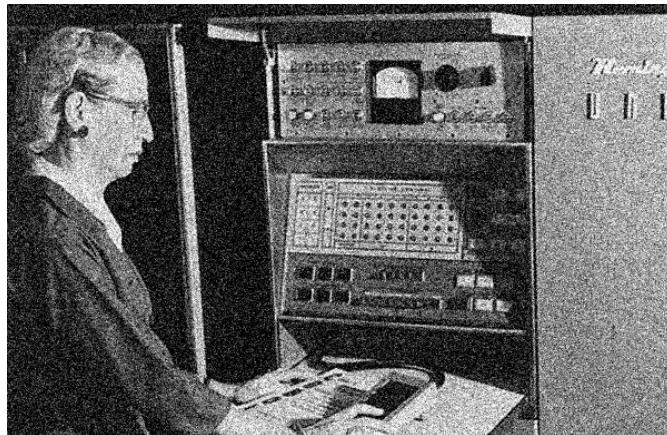
Noisy Input



$\text{dx via } [-1,01]$



Zoom



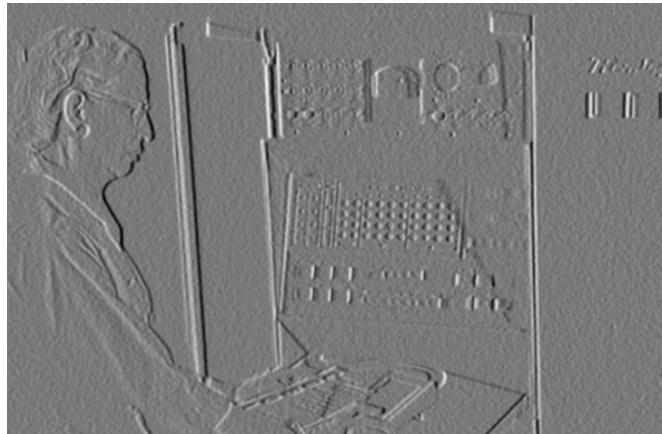
Source: D. Fouhey

Noise + Smoothing

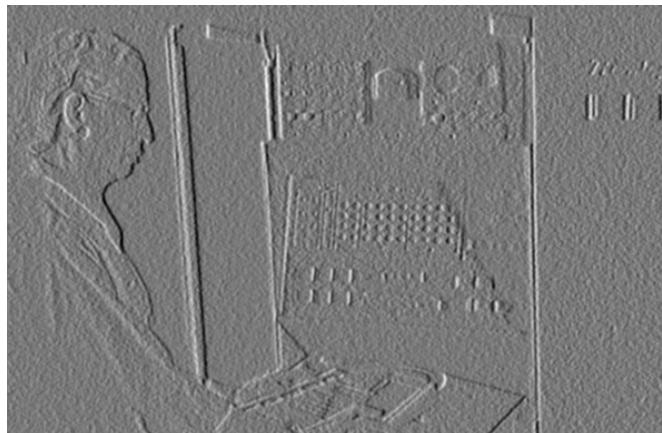
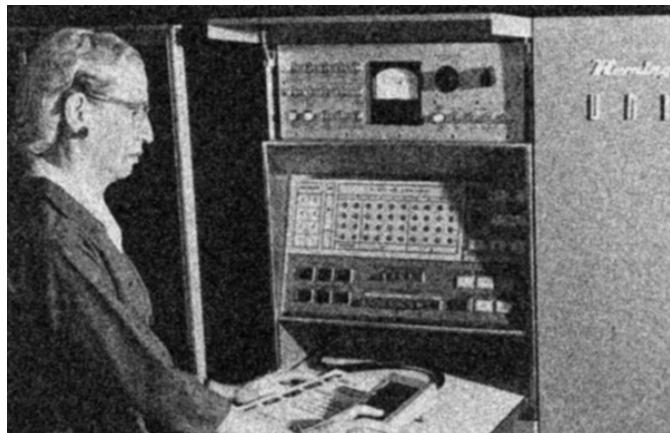
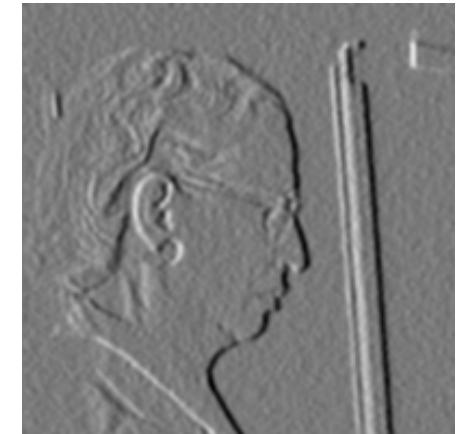
Smoothed Input



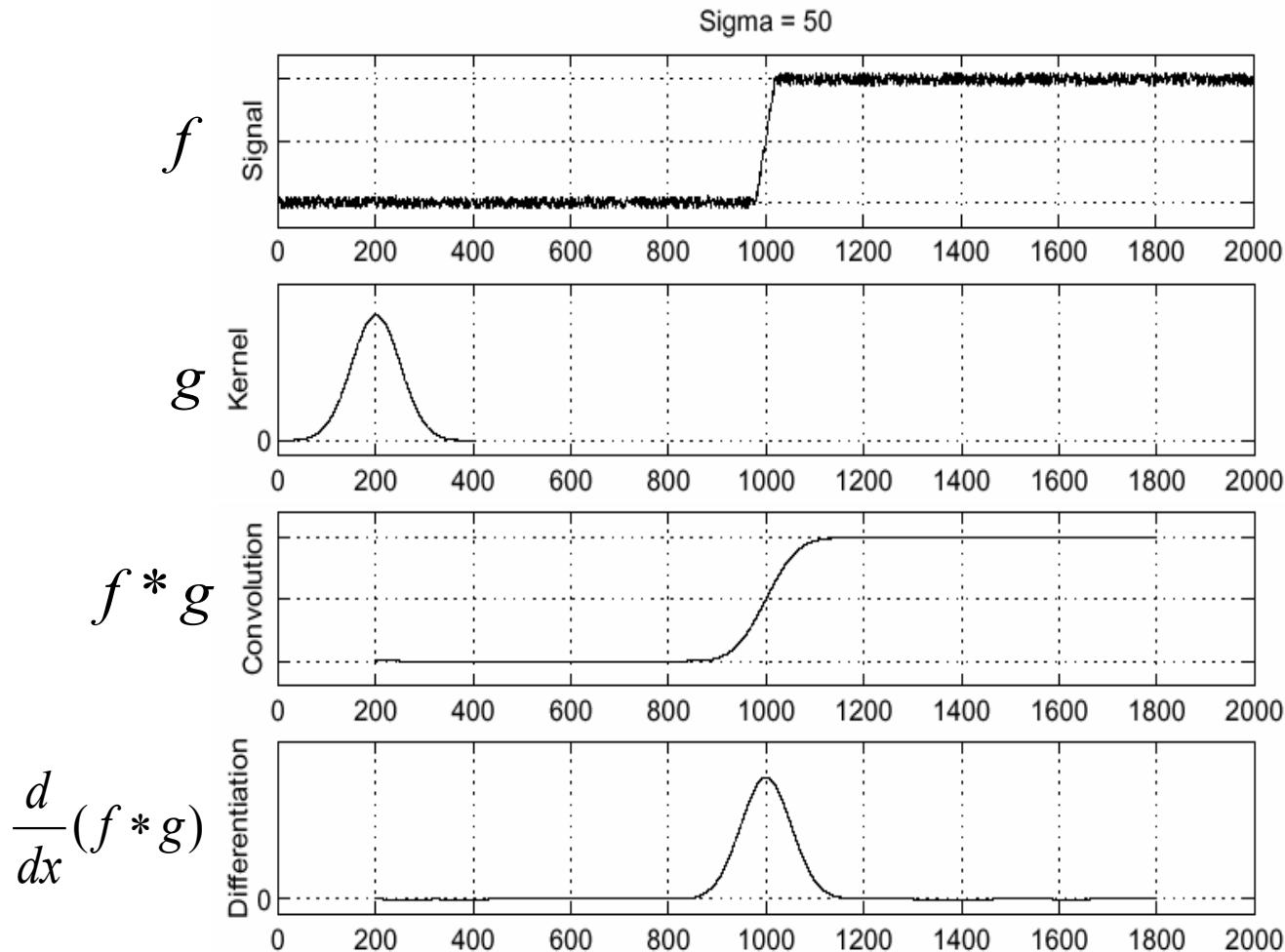
$\text{dx via } [-1,01]$



Zoom



How many convolutions here?

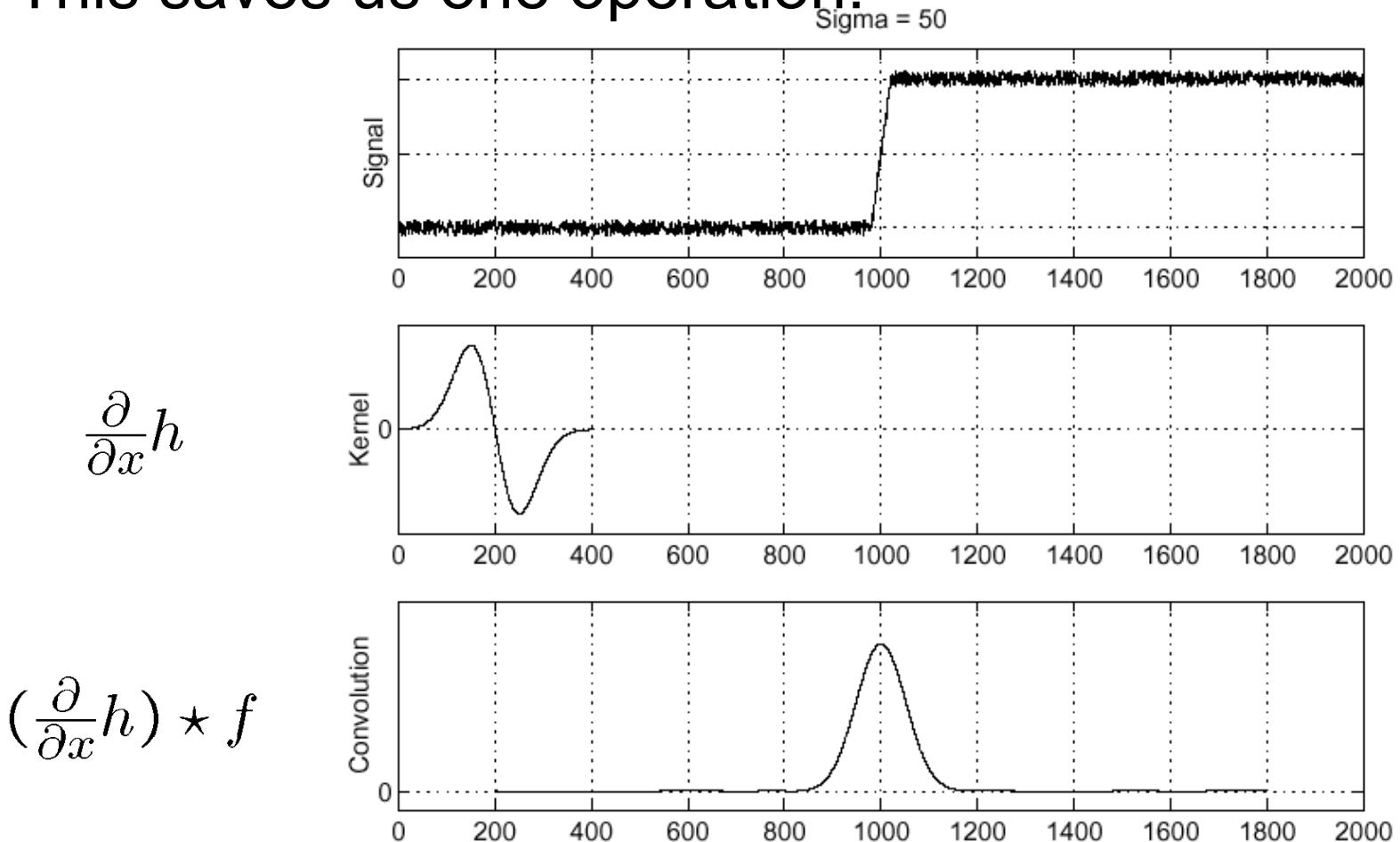


can we reduce this?

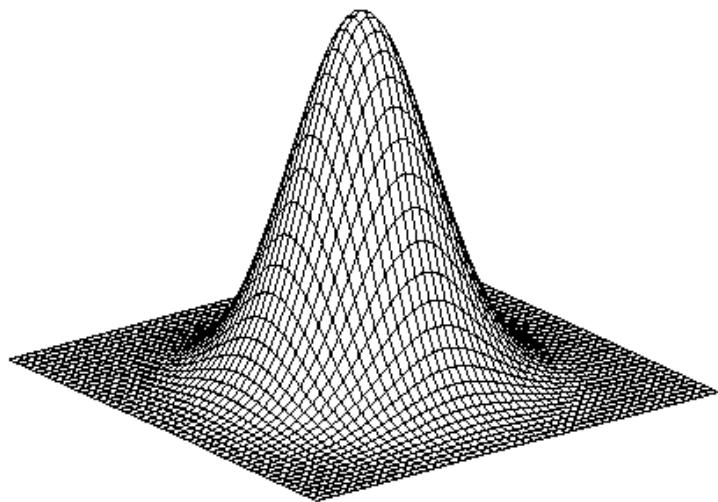
Derivative theorem of convolution

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

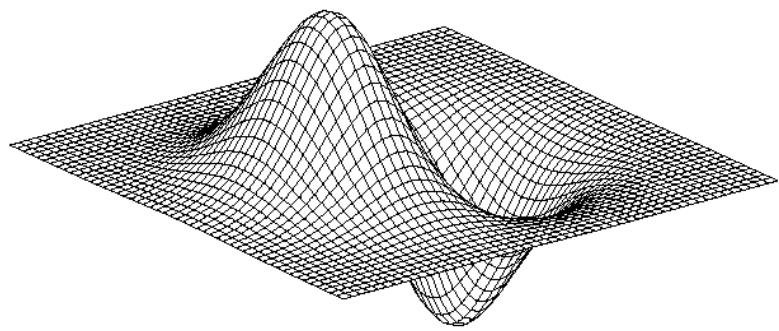
This saves us one operation:



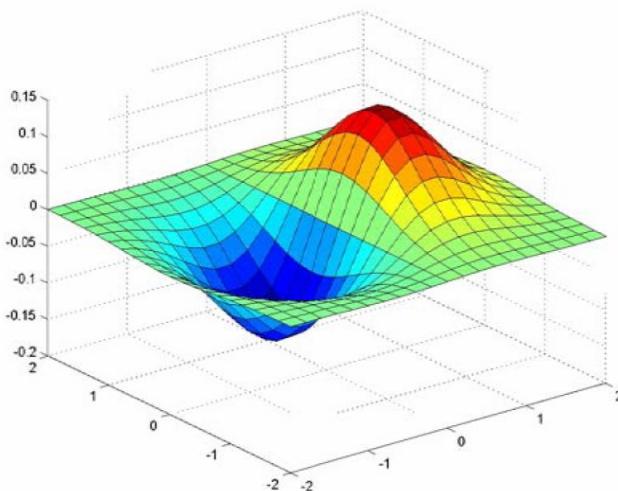
Derivative of Gaussian filter



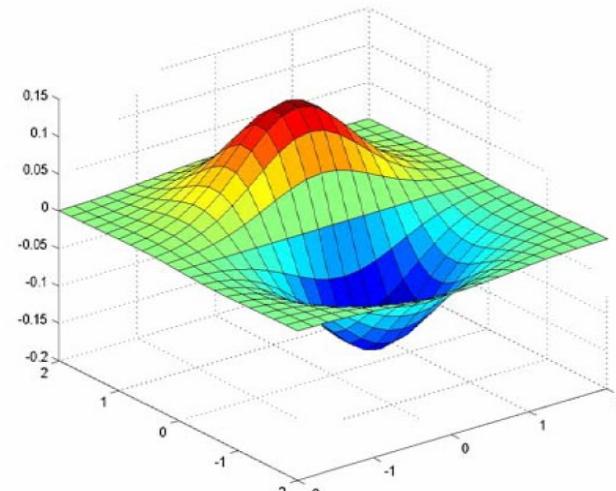
$$* [1 \ -1] =$$



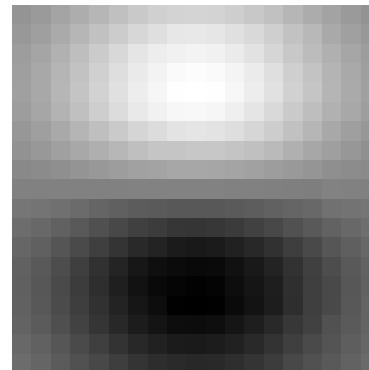
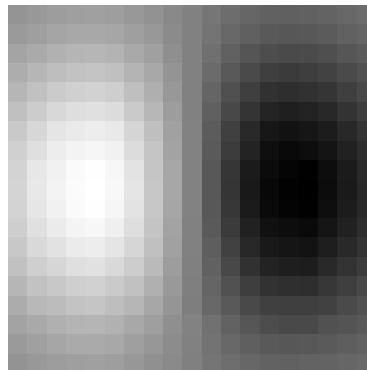
Derivative of Gaussian filter



x-direction



y-direction



Which one finds horizontal/vertical edges?

Compare to classic derivative filters

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

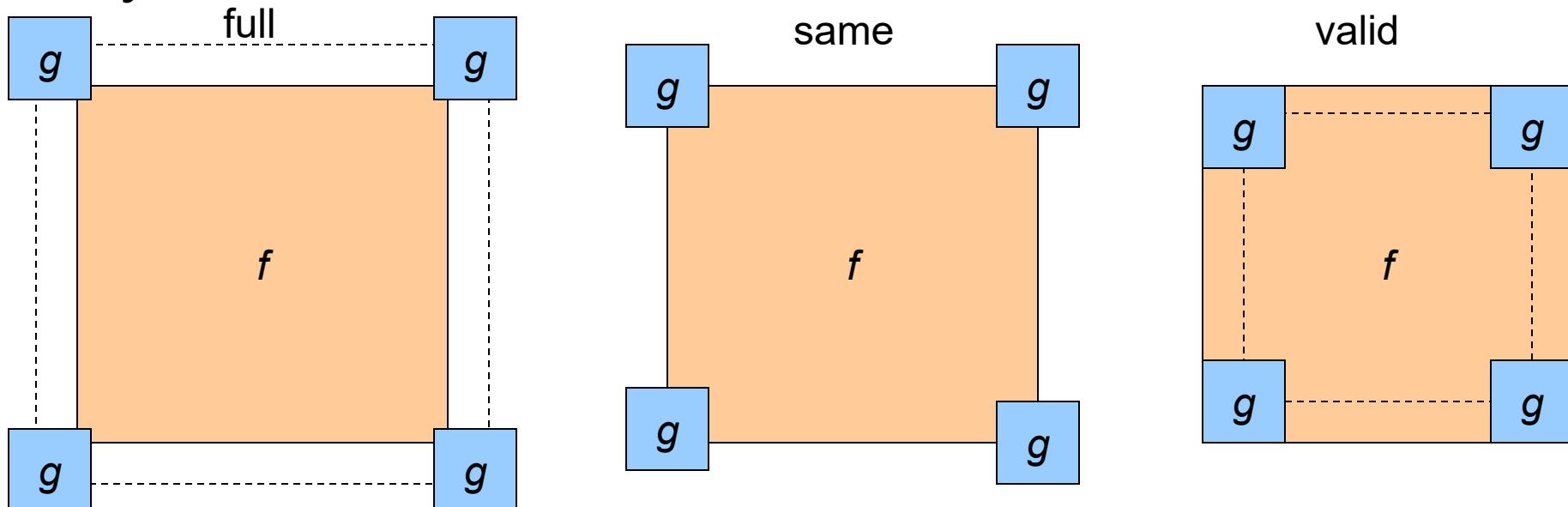
Filtering: practical matters

What is the size of the output?

(MATLAB) `filter2(g, f, shape)` or `conv2(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*

Pytorch `conv2d` ‘valid’ or ‘same’



Practical matters

What about near the edge?

- the filter window falls off the edge of the image
- need to extrapolate
- methods:
 - clip filter (black)
 - wrap around (circular)
 - copy edge
 - reflect across edge

