Derivative and Template Filters

CS194: Image Manipulation, Comp. Vision, and Comp. Photo

Alexei Efros, UC Berkeley, Spring 2020

Taking derivative by convolution

Partial derivatives with convolution

For 2D function f(x,y), the partial derivative is:

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

For discrete data, we can approximate using finite differences:

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

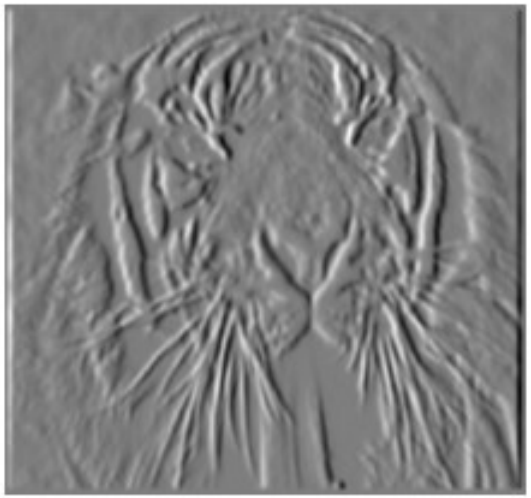
To implement above as convolution, what would be the associated filter?

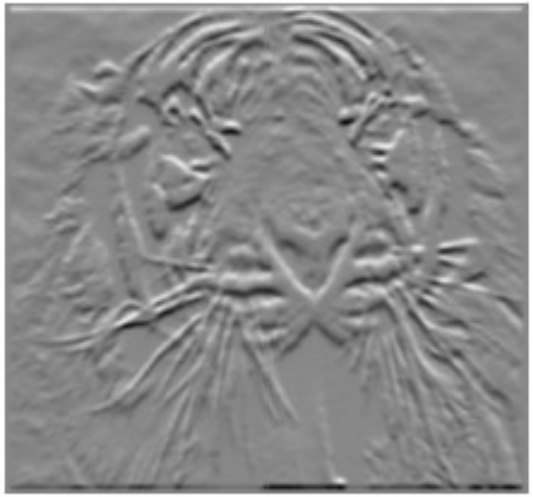
Source: K. Grauman

Partial derivatives of an image

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

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





-1	**1**

-1
1

or

1
-1

Which shows changes with respect to x?

Finite difference filters

Other approximations of derivative filters exist:

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

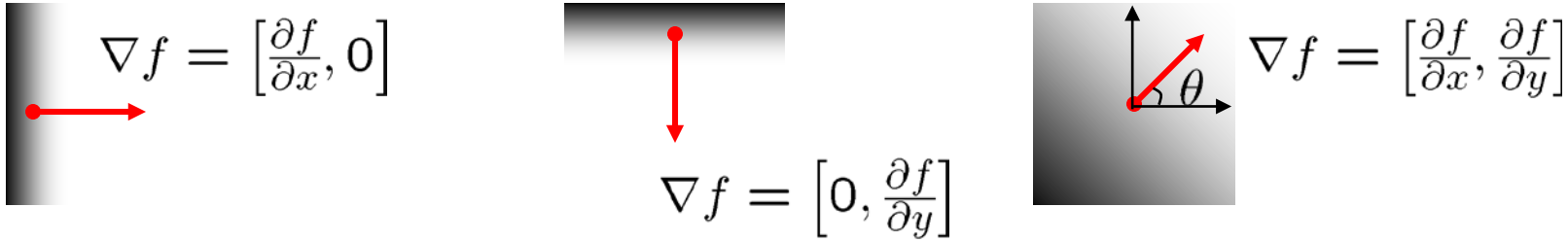
Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

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

Source: K. Grauman

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?

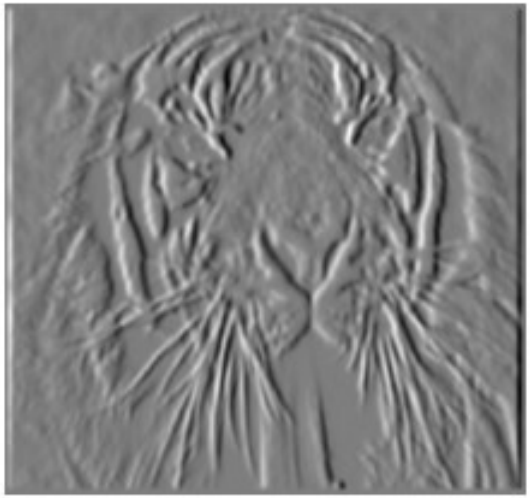
The gradient direction is given by $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

The *edge strength* is given by the gradient magnitude

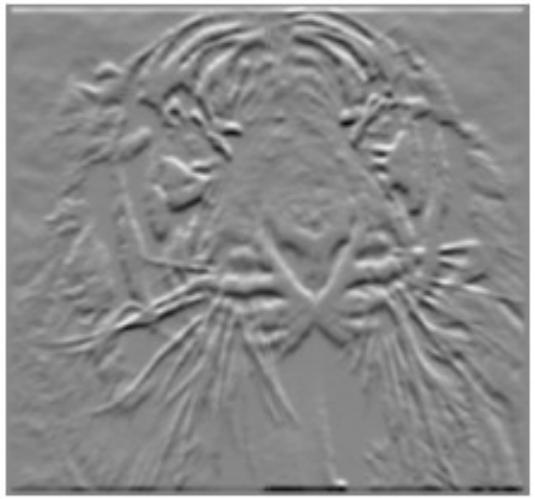
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Source: Steve Seitz

Image Gradient



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



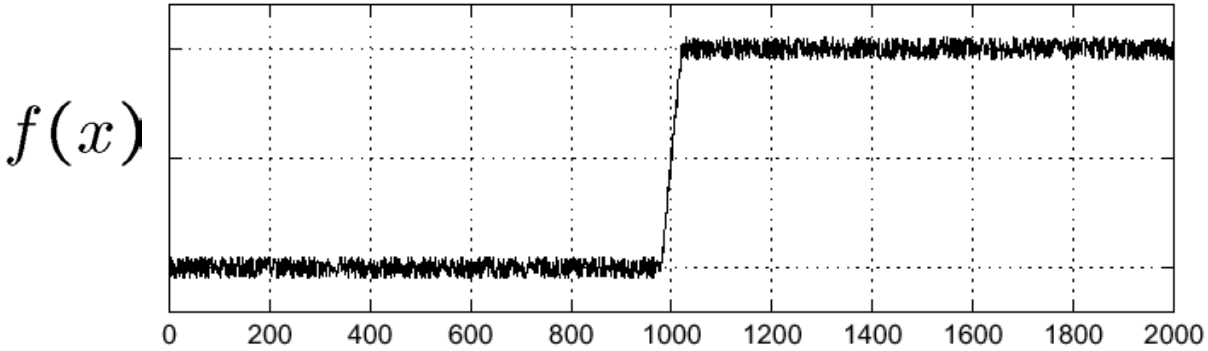
$$\frac{\partial f(x,y)}{\partial y}$$

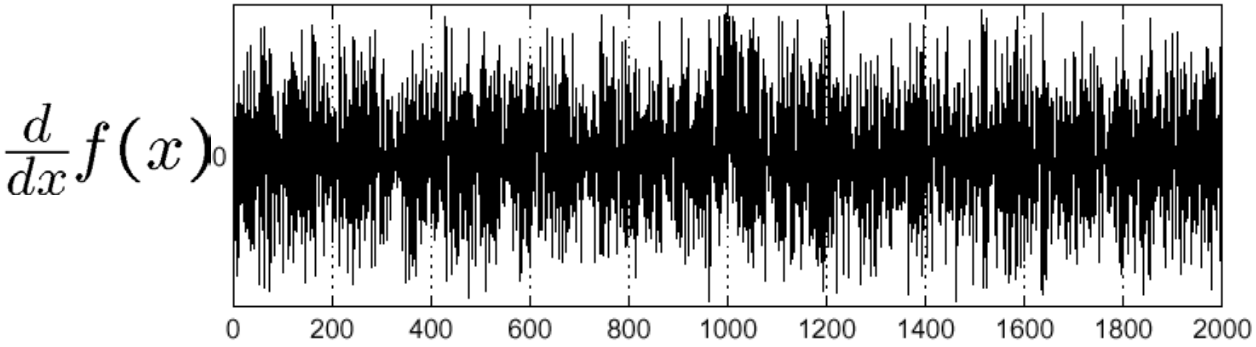
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Effects of noise

Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal

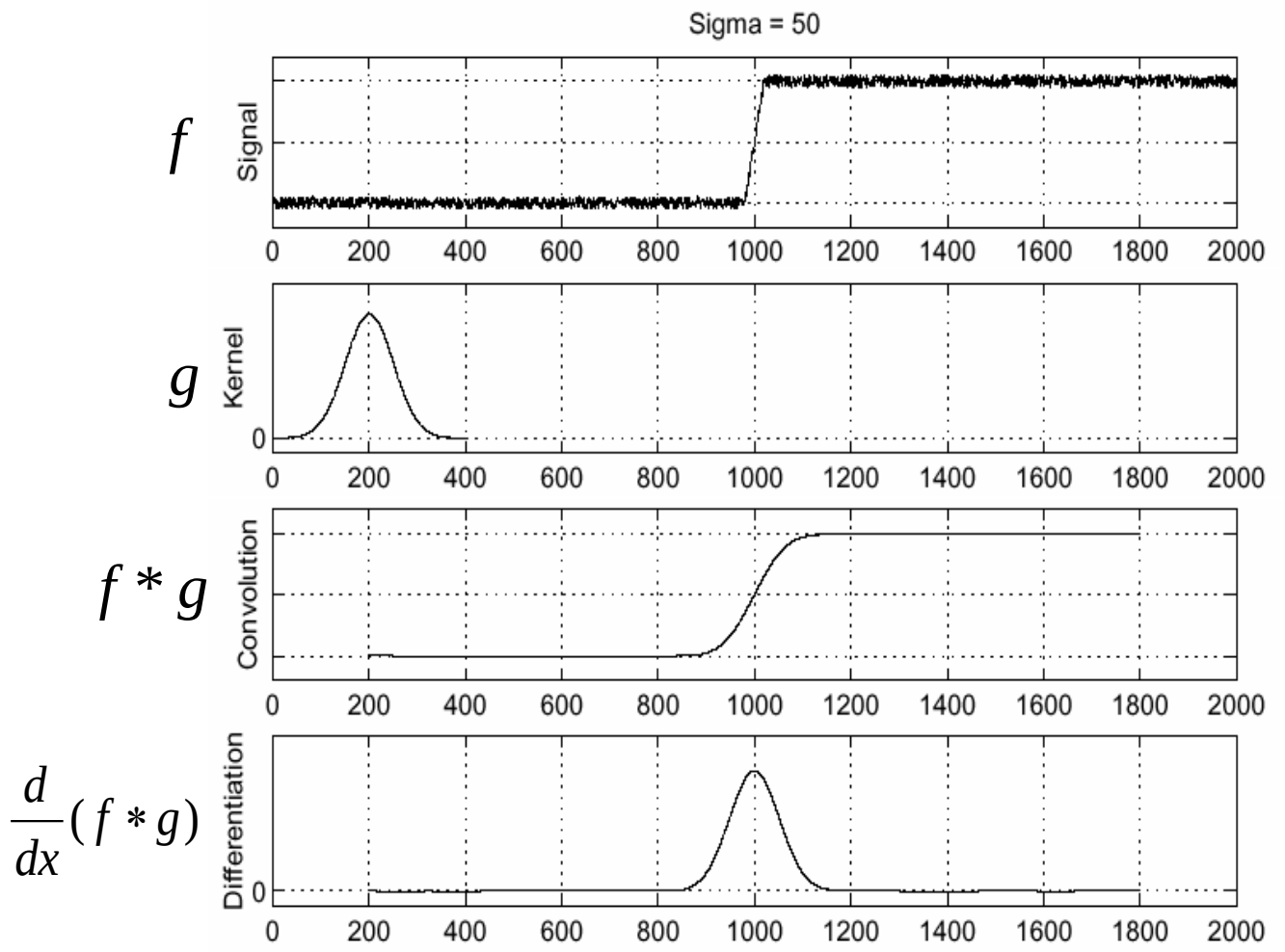




Where is the edge?

Source: S. Seitz

Solution: smooth first



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

Source: S. Seitz

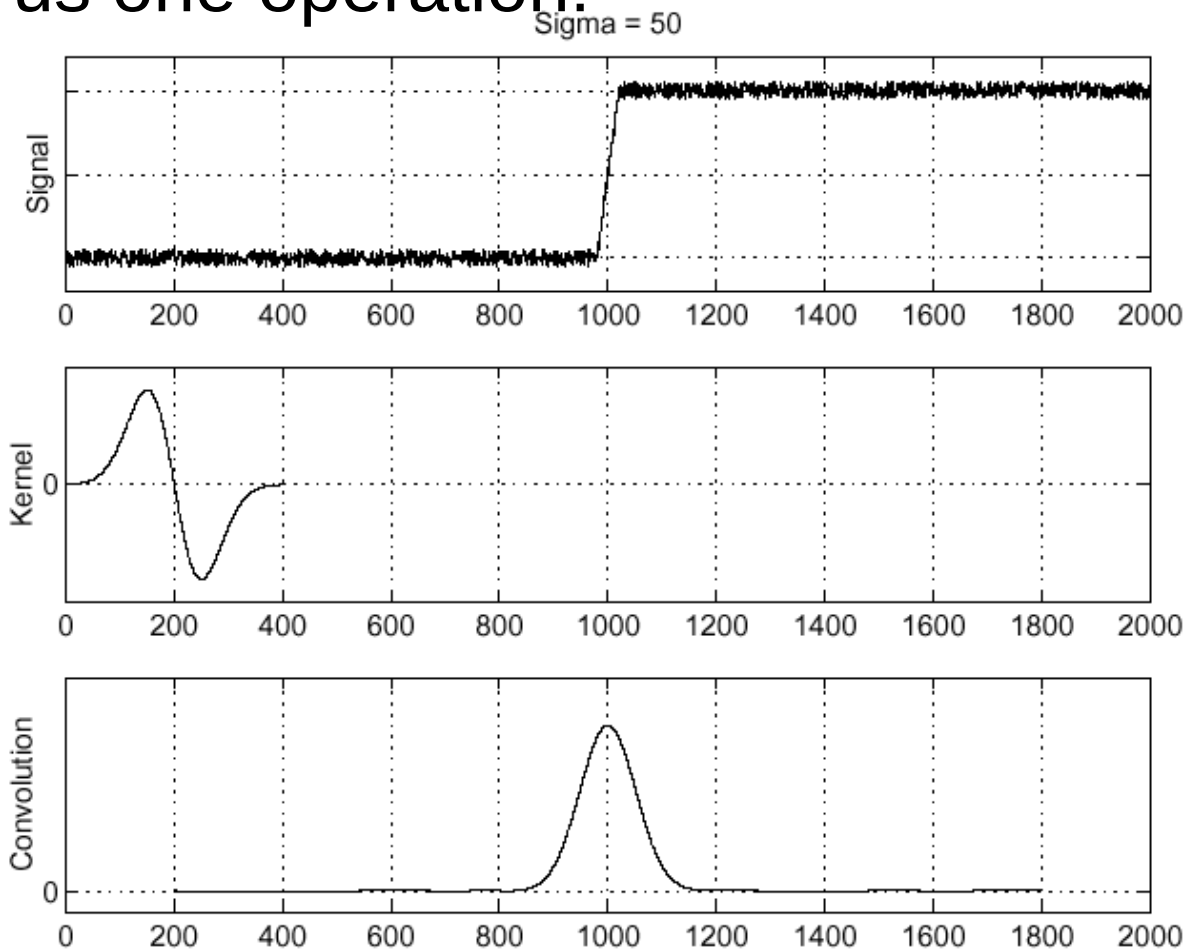
Derivative theorem of convolution

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

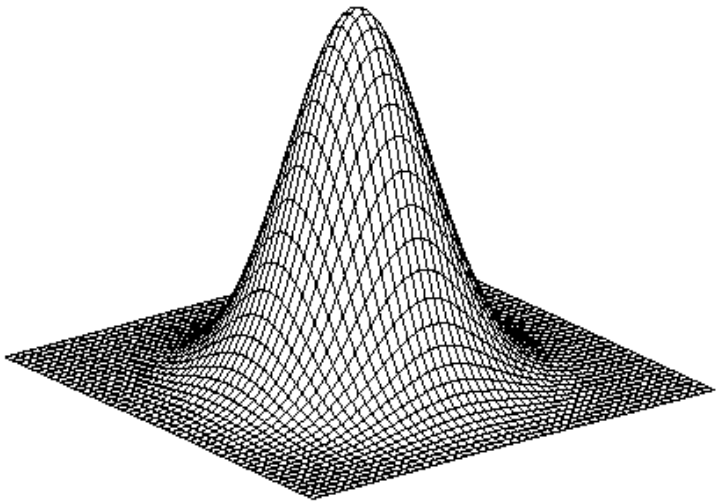
This saves us one operation:

$$\frac{\partial}{\partial x}h$$

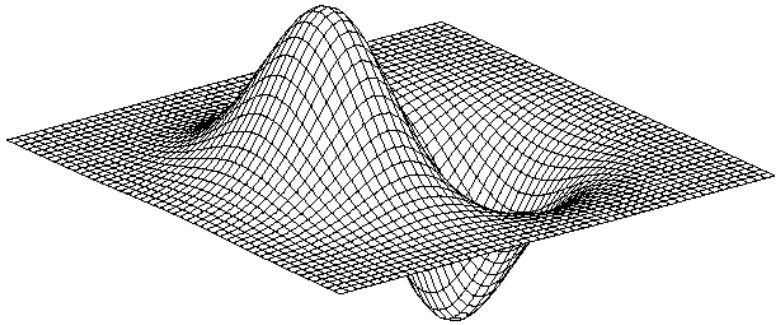
$$(\frac{\partial}{\partial x}h) \star f$$



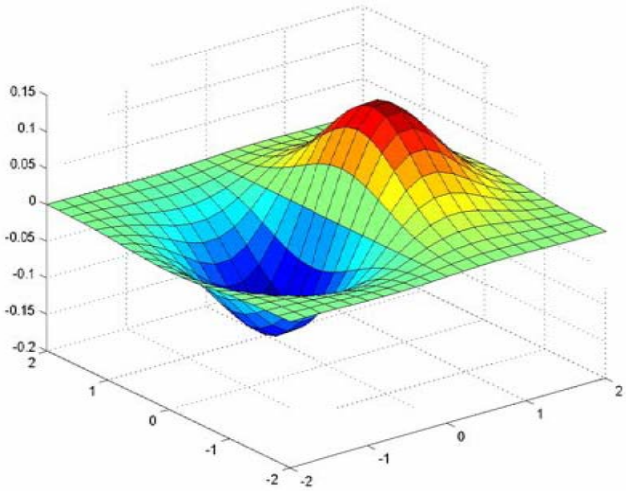
Derivative of Gaussian filter



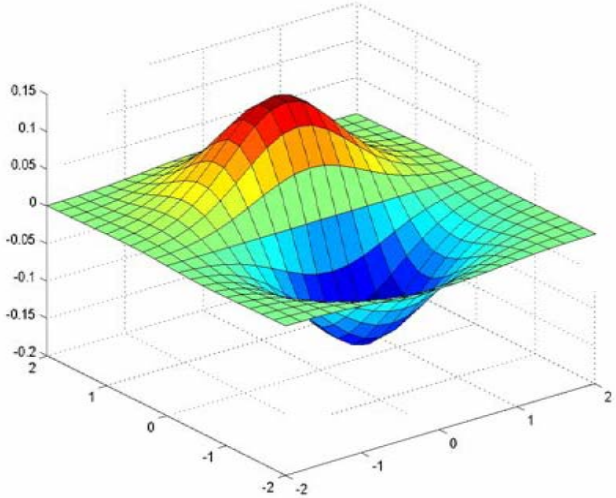
* [1 -1] =



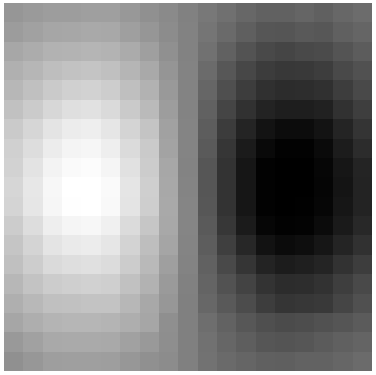
Derivative of Gaussian filter

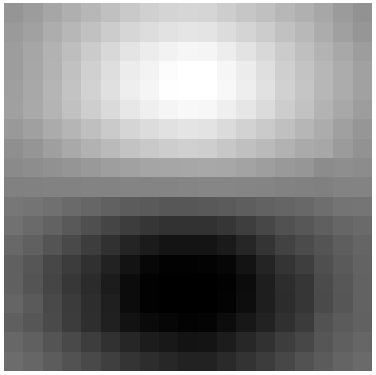


x-direction



y-direction



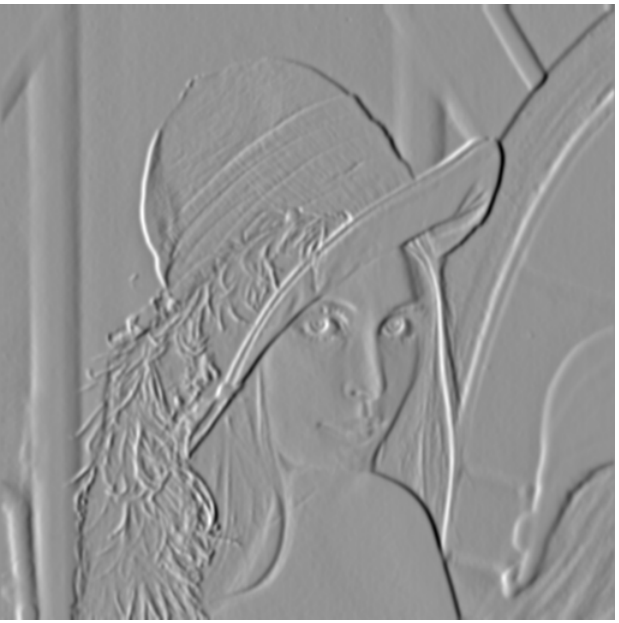


Which one finds horizontal/vertical edges?

Example

input image (“Lena”)

Compute Gradients (DoG)



X-Derivative of Gaussian

Y-Derivative of Gaussian

Gradient Magnitude

Get Orientation at Each Pixel

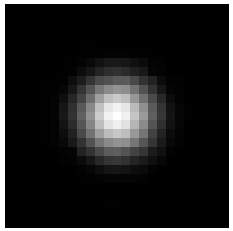
Threshold at minimum level

Get orientation

theta = atan2(-gy, gx)

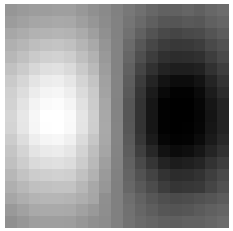
Review: Smoothing vs. derivative filters

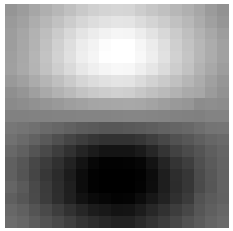
Smoothing filters



- Gaussian: remove “high-frequency” components; “low-pass” filter
- Can the values of a smoothing filter be negative?
- What should the values sum to?
  - **One:** constant regions are not affected by the filter

Derivative filters





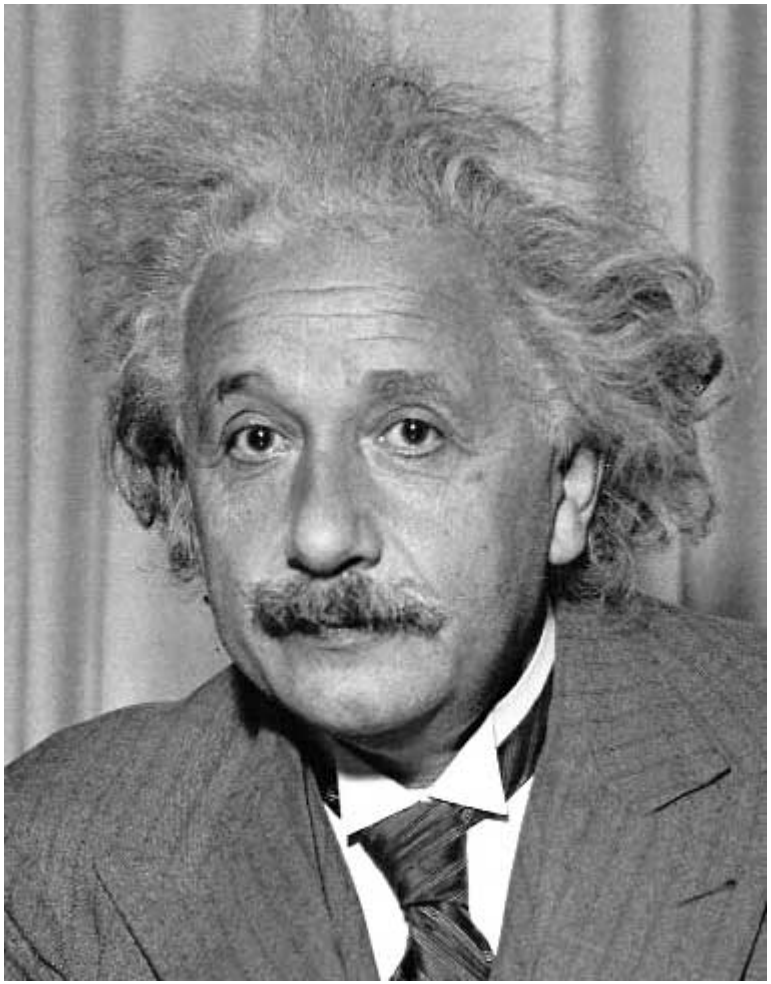
- Derivatives of Gaussian
- Can the values of a derivative filter be negative?
- What should the values sum to?
  - **Zero:** no response in constant regions
- High absolute value at points of high contrast

Template matching

Goal: find  in image

Main challenge: What is a good similarity or distance measure between two patches?

- Correlation
- Zero-mean correlation
- Sum Square Difference
- Normalized Cross Correlation



Side by Derek Hoiem

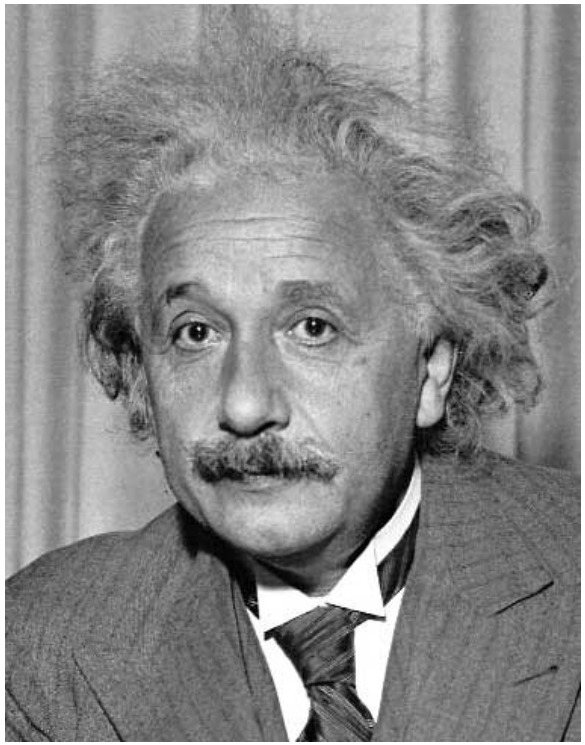
Matching with filters

Goal: find  in image

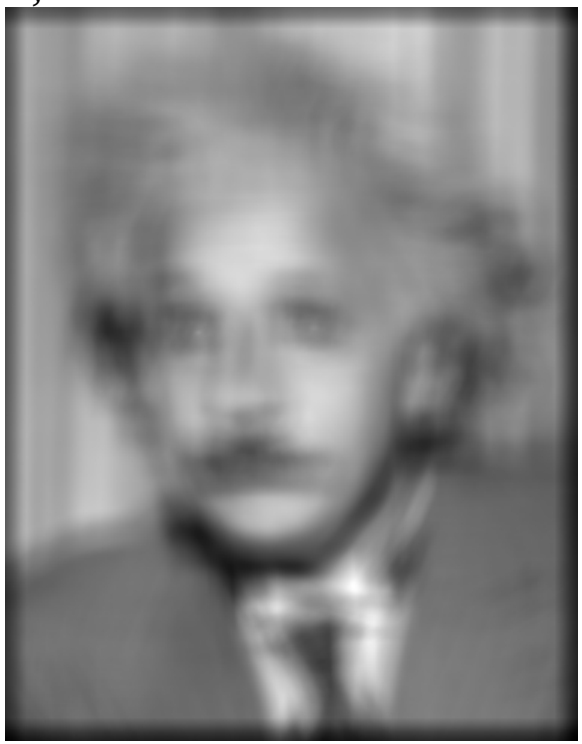
Method 0: filter the image with eye patch

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

f = image
g = filter



Input



Filtered Image

What went wrong?

Side by Derek Hoiem

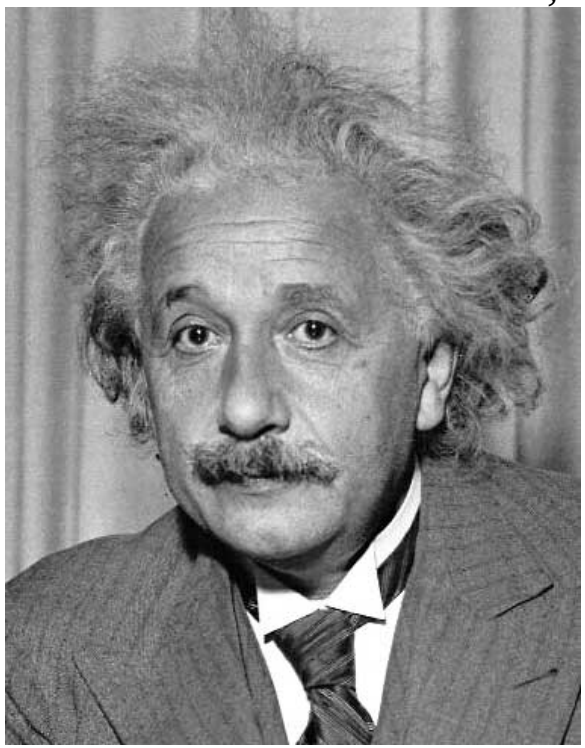
Matching with filters

Goal: find  in image

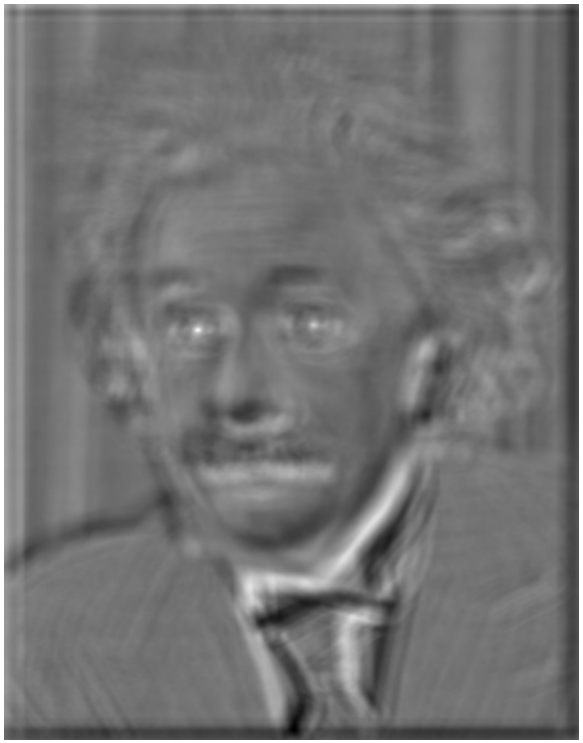
Method 1: filter the image with zero-mean eye

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

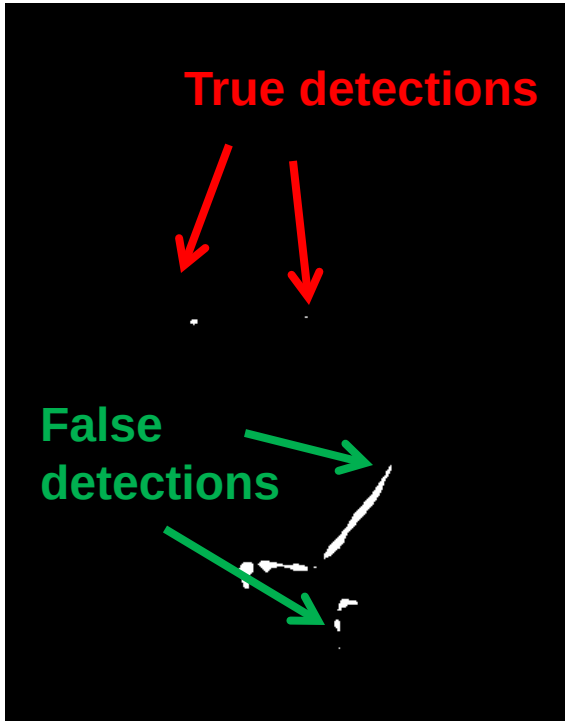
mean of f



Input



Filtered Image (scaled)



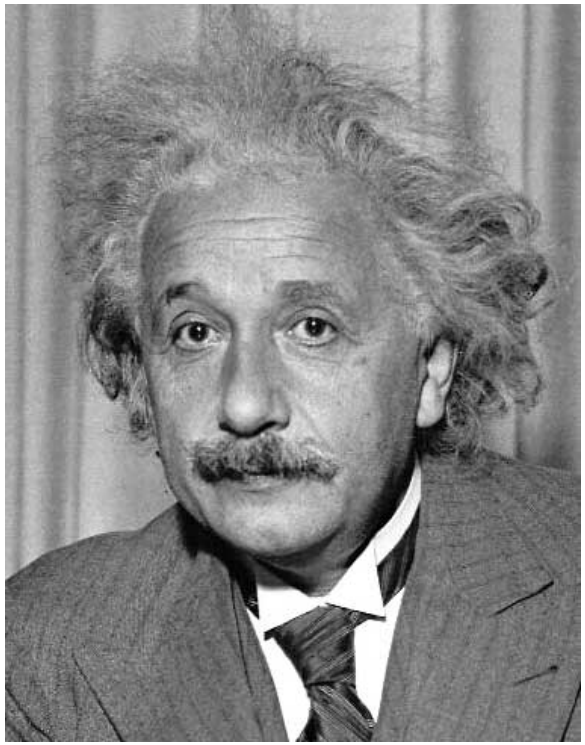
Thresholded Image

Matching with filters

Goal: find  in image

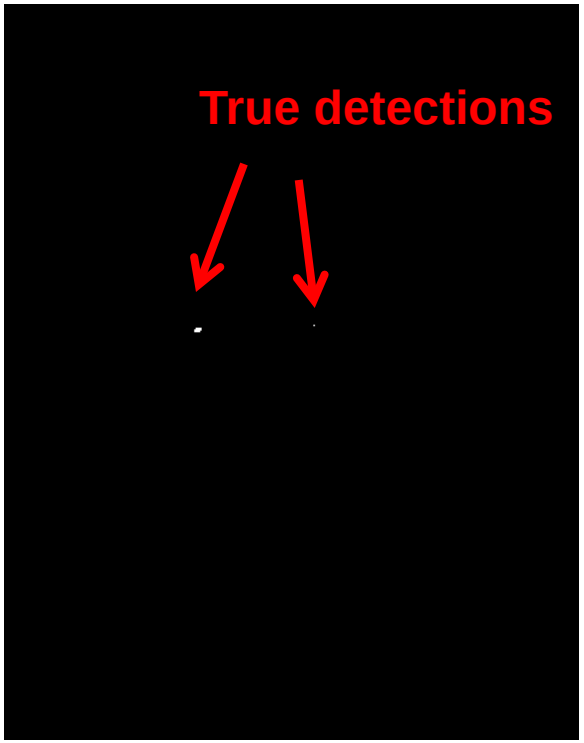
Method 2: SSD

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$



Input

1- sqrt(SSD)



Thresholded Image

Matching with filters

Can SSD be implemented with linear filters?

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$

Side by Derek Hoiem

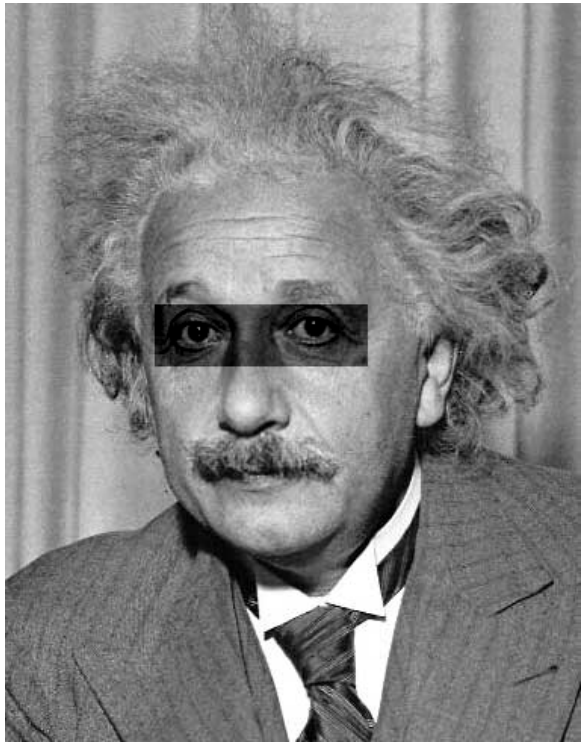
Matching with filters

Goal: find  in image

What's the potential downside of SSD?

Method 2: SSD

$$h[m,n]=\sum_{k,l}(g[k,l]-f[m+k,n+l])^2$$



Input

1- sqrt(SSD)

Side by Derek Hoiem

Matching with filters

Goal: find  in image

Method 3: Normalized cross-correlation

mean template

mean image patch

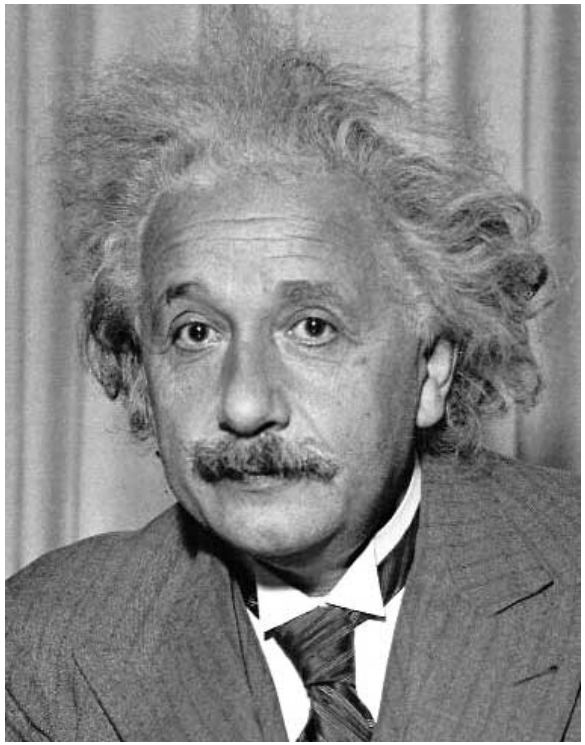
$$h[m,n]=\frac{\sum_{k,l}(g[k,l]-\bar{g})(f[m+k,n+l]-\bar{f}_{m,n})}{\left(\sum_{k,l}(g[k,l]-\bar{g})^2\sum_{k,l}(f[m+k,n+l]-\bar{f}_{m,n})^2\right)^{0.5}}$$

Side by Derek Hoiem

Matching with filters

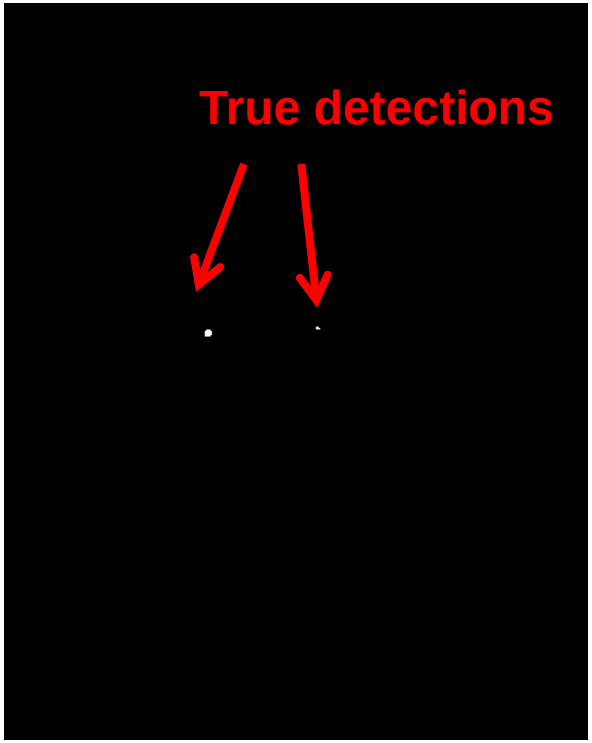
Goal: find  in image

Method 3: Normalized cross-correlation



Input

Normalized X-Correlation

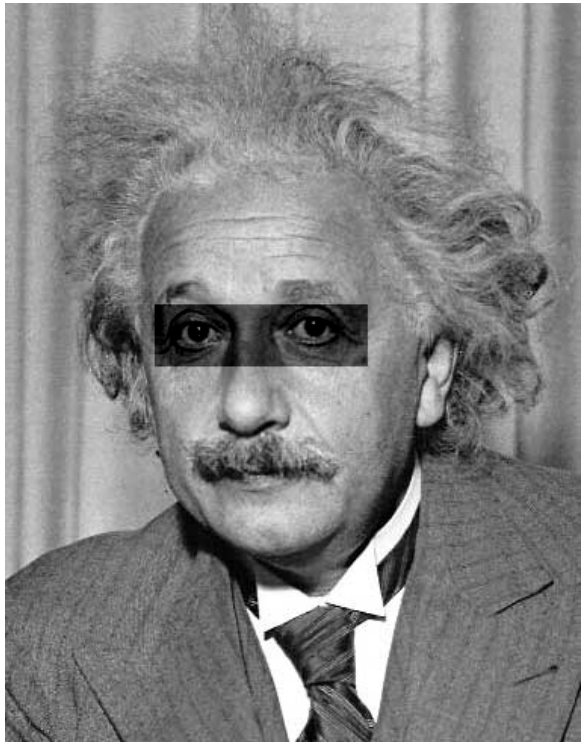


Thresholded Image

Matching with filters

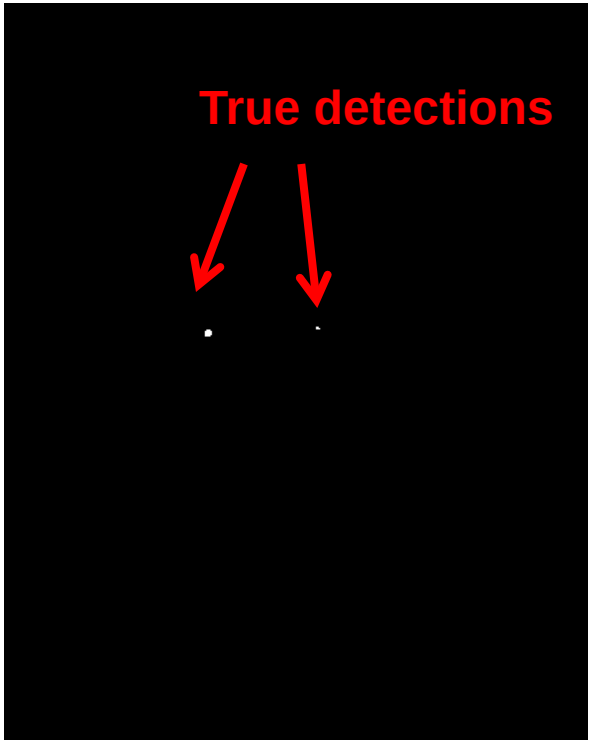
Goal: find  in image

Method 3: Normalized cross-correlation



Input

Normalized X-Correlation



Thresholded Image

Q: What is the best method to use?

A: Depends

Zero-mean filter: fastest but not a great matcher

SSD: next fastest, sensitive to overall intensity

Normalized cross-correlation: slowest, invariant to local average intensity and contrast

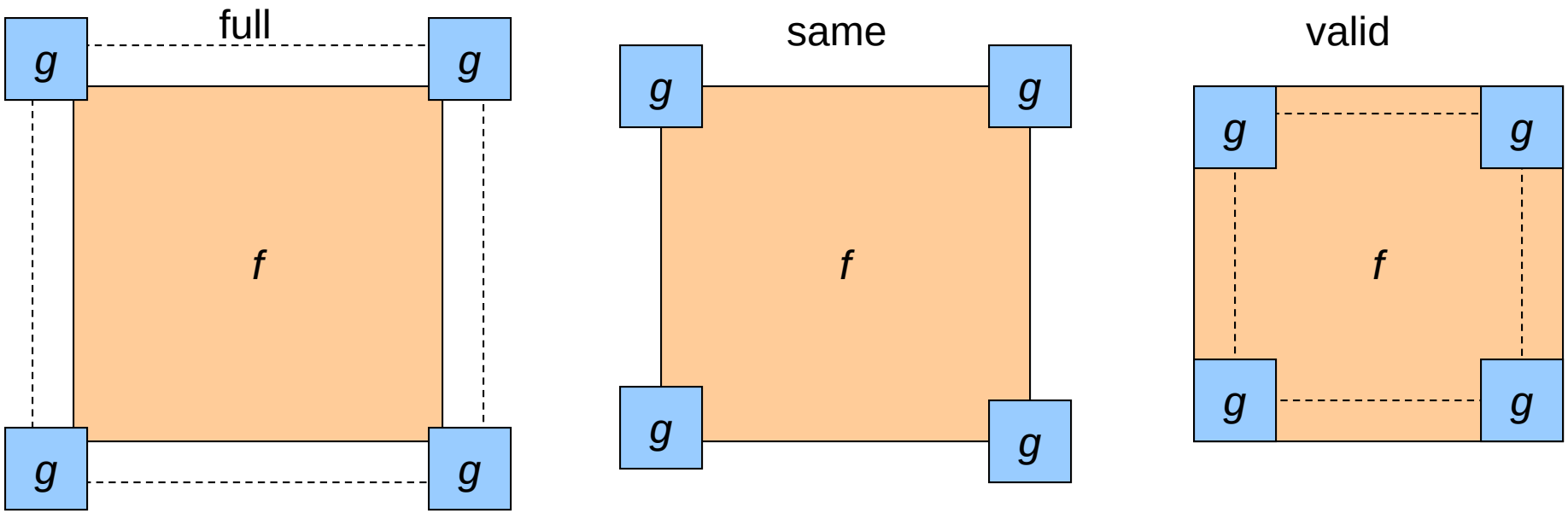
Side by Derek Hoiem

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



Source: S. Lazebnik

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
  - copy edge
  - reflect across edge

Source: S. Marschner

Q?

Practical matters

- 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