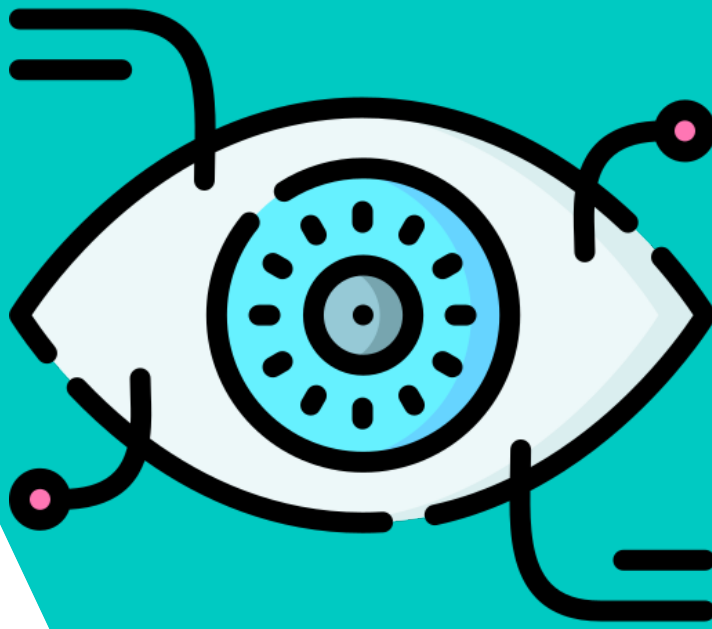
Computer Vision

A.A. 2024-20245

Lecture 2: Image processing



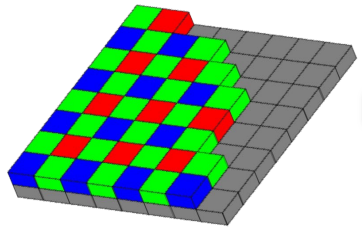
SAPIENZA
UNIVERSITÀ DI ROMA

ALCOR Lab

Reading

- Szeliski textbook, Chapter 3.1,3.2, 3.3, 3.5

A typical color imaging pipeline



Sensor with color filter array (CFA) (CCD/CMOS)

ISO gain and raw-image processing

RGB Demosaicing

Bayer Pattern: RGGB

raw-RGB: linear w.r.t luminance

Noise Reductio n

White-Balance & Color Space Transform (CIE XYZ)

White-balance on raw-RGB

raw-RGB -> CIE XYZ

Color Manipulation (Photo-finishing)

Mapping to sRGB output

s-RGB

JPEG Compression

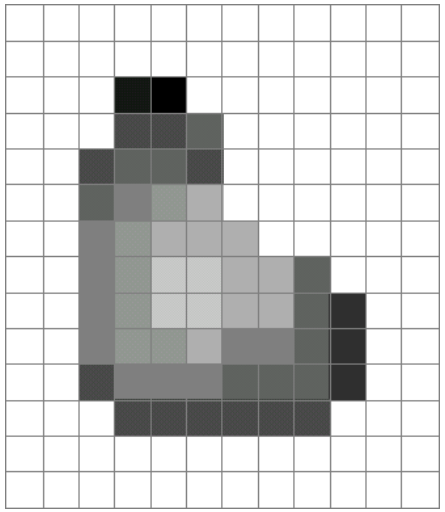
Save to file

Research topics:

- Noise reduction, especially low-light imaging
- Auto White Balancing (AWB)

What is an image?

- A grid (matrix) of intensity values



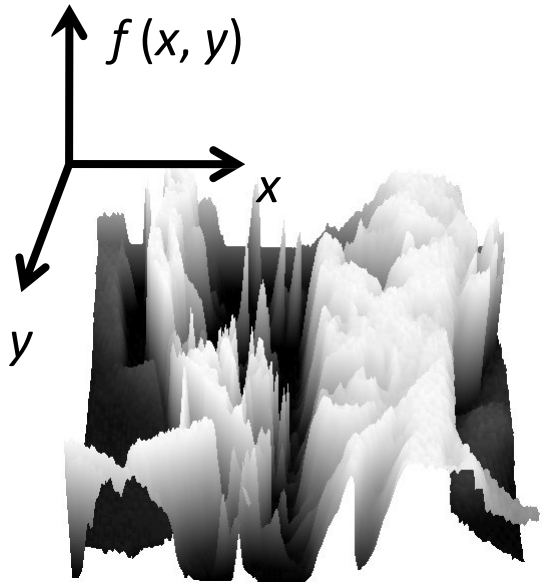
=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

(common to use one byte per value: 0 = black, 255 = white)

What is an image?

- Can think of a (grayscale) image as a **function** f from R^2 to R:
  - $f(x,y)$ gives the **intensity** at position (x,y)



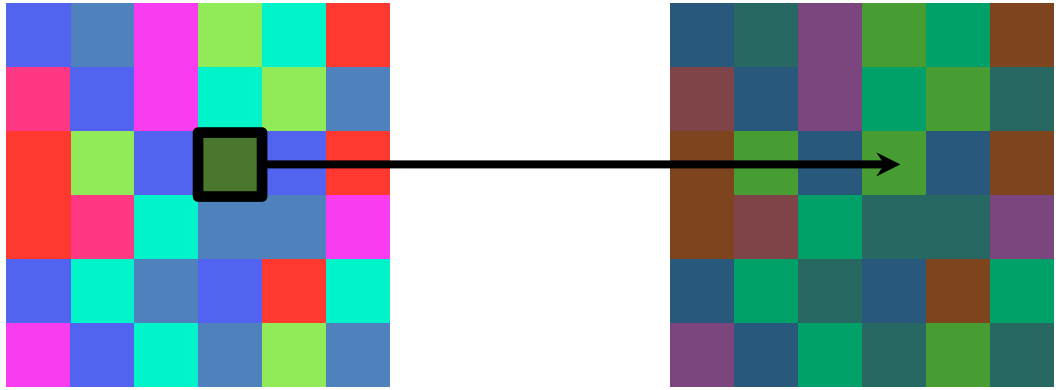
  - A **digital** image is a discrete (**sampled**, **quantized**) version of this function

Image Transformations & Filtering

- Point Processing
- Linear Filtering
- Sampling & Aliasing
- Image Derivatives
- Edge Detection

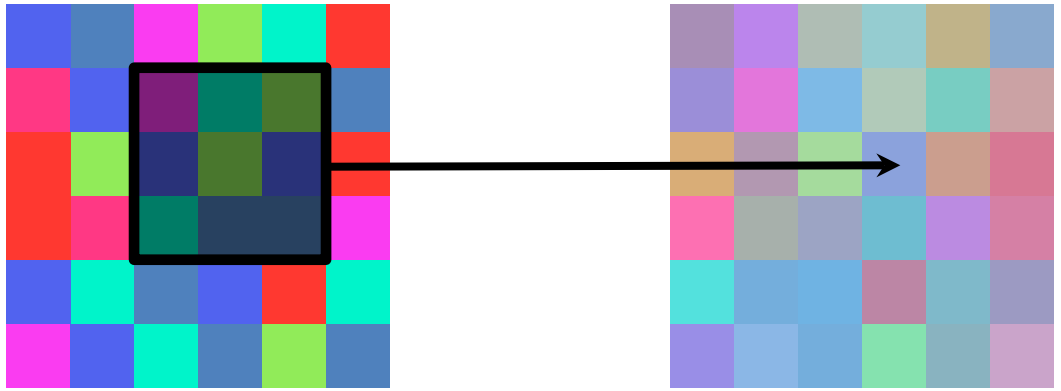
What types of image filtering can we do?

Point Operation



point processing

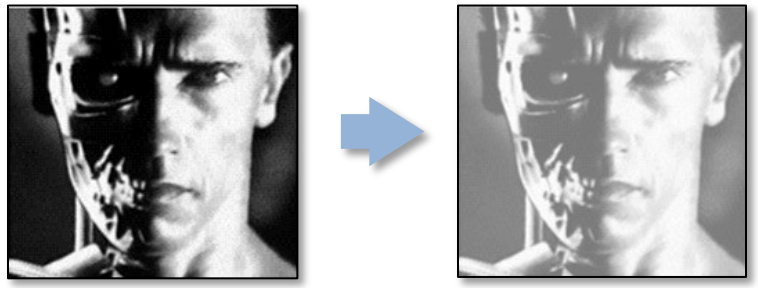
Neighborhood Operation



“filtering”

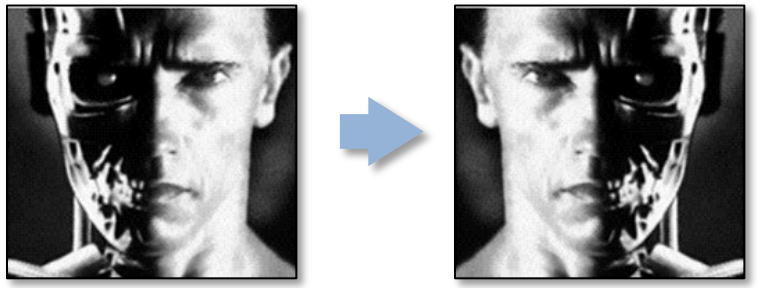
Examples of Point Processing

- As with any function, we can apply operators to an image



$g(x,y) = f(x,y) + 20$

brightness



$g(x,y) = f(-x,y)$

Horizontal flip

Examples of point processing

original

x

darken

$x - 128$

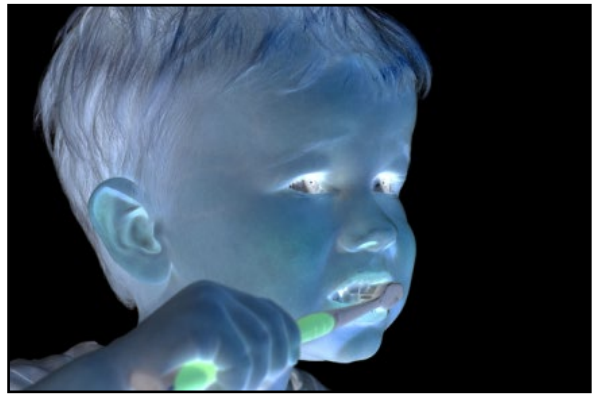
lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert



$255 - x$

lighten

$x + 128$

raise contrast

$x \times 2$

non-linear raise contrast

$\left(\frac{x}{255}\right)^{2} \times 255$

Gamma Correction

- Adjust brightness and contrast in images for accurate display on different devices (monitors, TVs, etc.).
- Compensates for the non-linear relationship between input pixel values and output display brightness.

Why is it needed?

- Devices like monitors have a non-linear response to brightness.
- The human eye perceives light in a non-linear way, and gamma correction ensures images appear natural. Human vision is more sensitive to changes in darker tones than in lighter tones.

$$V_{\text{out}} = AV_{\text{in}}^{\gamma},$$

γ > 1: Reduces the brightness (darkens the image).

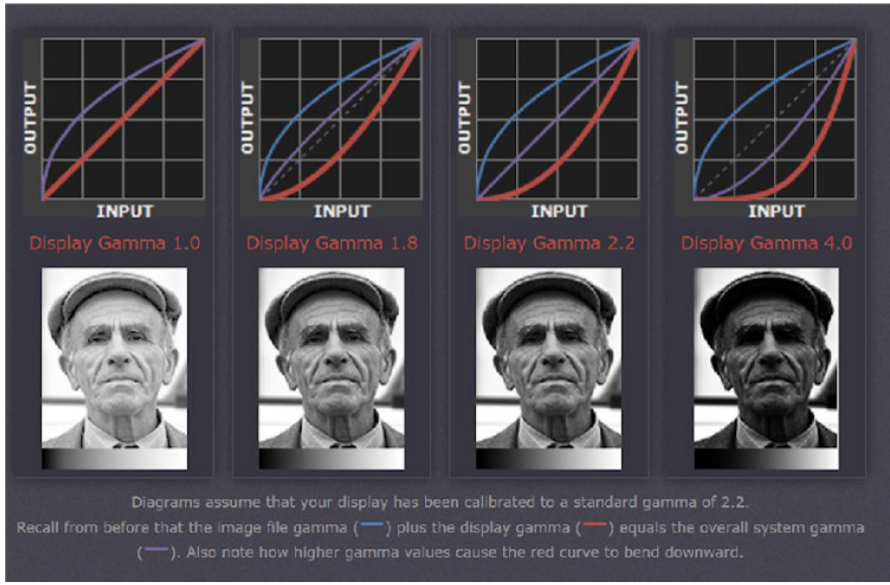
γ < 1: Increases brightness (lightens the image).

γ = 1: No change (linear).

A: A normalization constant, which might ensure that values are within a standard range (like 0-255 for an 8-bit image or 0-1 for normalized images).

Gamma Correction

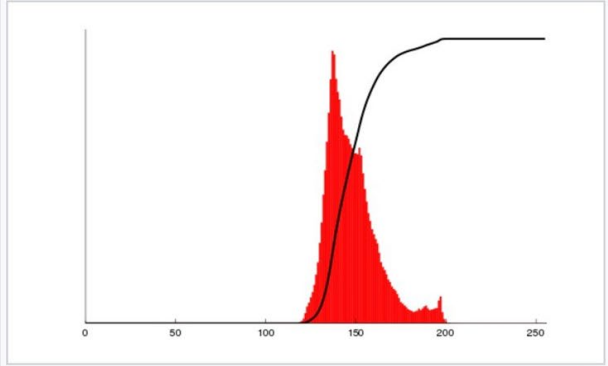
- Most images we work with are in sRGB (Standard Red Green Blue) space, i.e., already tone-mapped.
- Many Vision algorithms expects the image to be in linear space.
- sRGB -> Linear space conversion requires explicit knowledge of camera processing pipeline. Requires knowledge of tone-production curve.
- In absence of it it is very common to use $\gamma = 2.2$.
- Note: This is not accurate, just a cheap approximation that works for most Vision tasks.



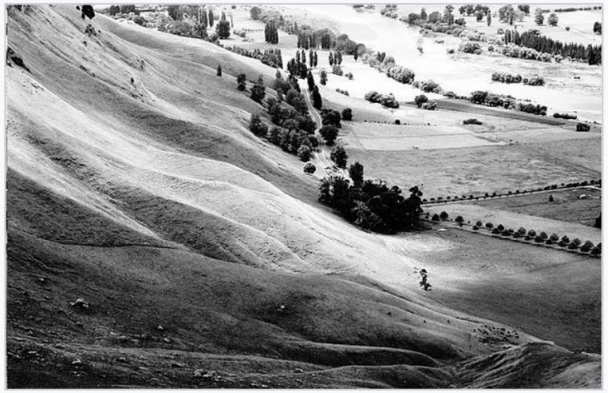
Histogram Equalization

Image processing technique that adjusts the contrast of an image by using its histogram.

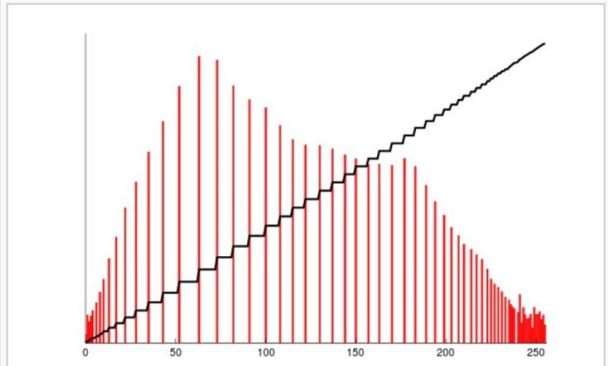
Before Histogram Equalization



Corresponding histogram (red) and cumulative histogram (black)



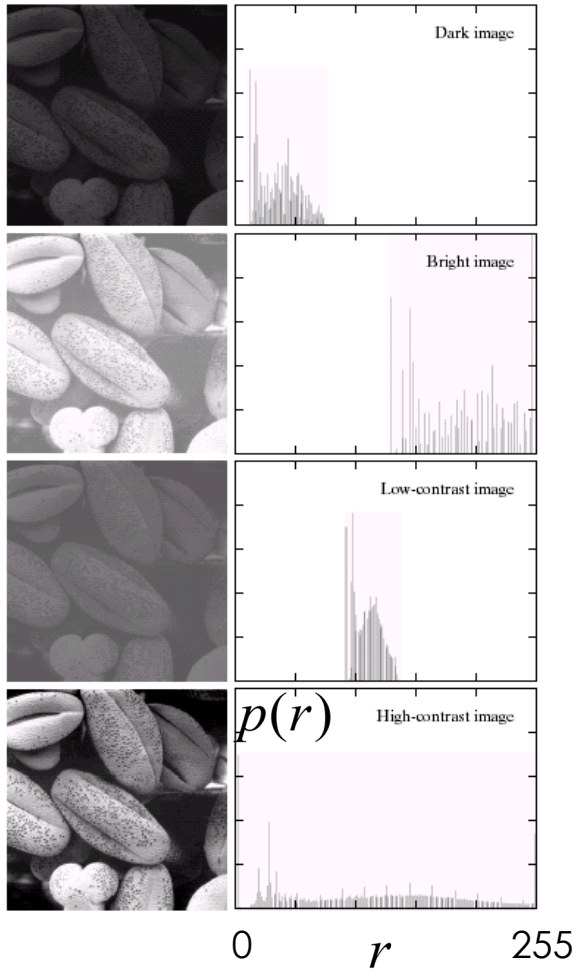
After Histogram Equalization



Corresponding histogram (red) and cumulative histogram (black)

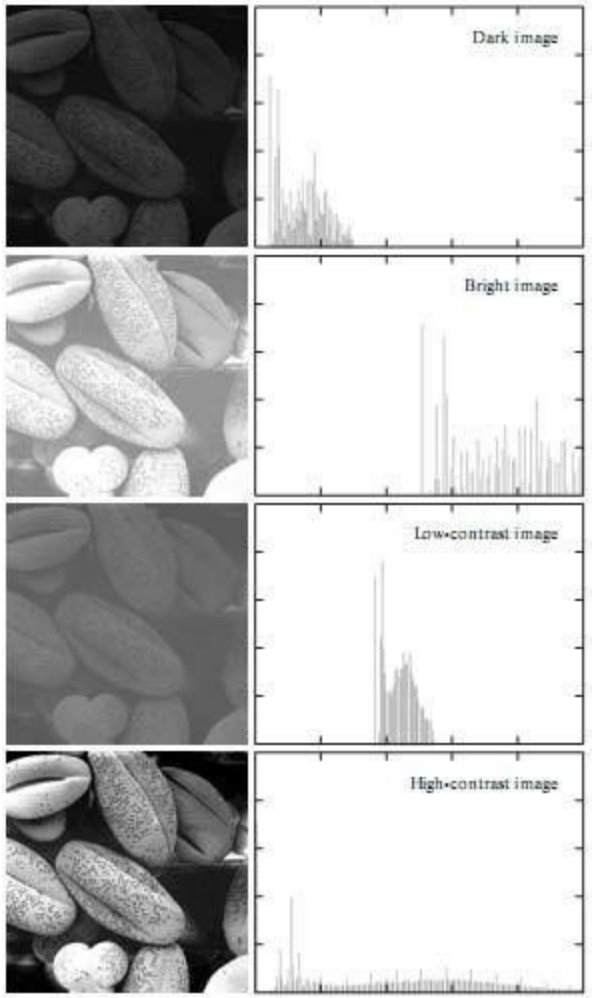
Intensity Histogram

$$p(r) = \left(\frac{\text{Number of pixels with intensity } r}{\text{Total number of pixels}} \right)$$



High contrast image

- The histogram components are distributed over all the intensity range.
- The distribution is almost uniform, with few peaks.
- If the distribution is uniform, the image tends to have a high dynamic range and the details are more easily perceived.
- This is the effect pursued by the histogram-based transformations.



How it works

- **Calculate the Histogram**: Count the number of pixels at each intensity level.

$$p(r_k) = \frac{\text{Number of pixels with intensity } r_k}{\text{Total number of pixels}}$$

- **Normalize the Histogram**: Find the cumulative distribution function (CDF) of the histogram.

$$CDF(r_k) = \sum_{i=0}^{\kappa} p(r_i)$$

- **Map Intensities**: Adjust pixel values based on the CDF to achieve a more uniform distribution. The CFD is normalized in such a way the minimum and maximum values of the CDF correspond to the minimum and maximum possible intensity values (0 and 255 for an 8-bit image).

$$r'_k = \text{round}\left(\frac{(CDF(r_k) - CDF_{\min}) \cdot (L-1)}{N - CDF_{\min}}\right)$$

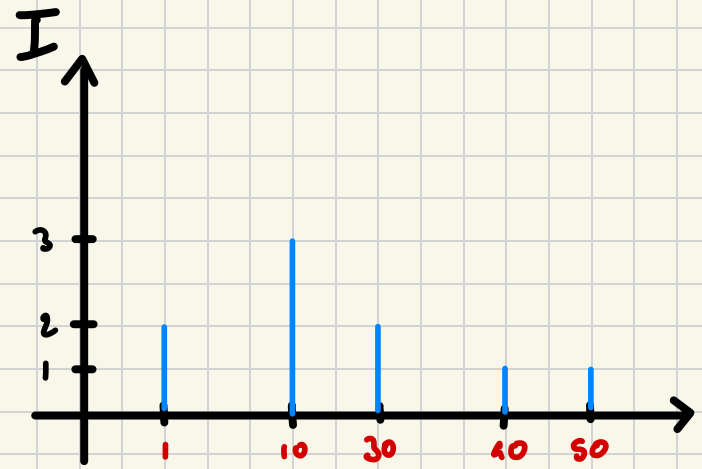
Where:

- r'_k is the new intensity.
- L is the maximum pixel intensity (e.g., 255 for 8-bit images).
- N is the total number of pixels in the image.
- $CDF_{\min}$ is the minimum non-zero value of the CDF.

$$I = \begin{bmatrix} 10 & 10 & 30 \\ 40 & 10 & 30 \\ 50 & 1 & 1 \end{bmatrix}$$

$$CDF : \left(\frac{(5-2)\cdot(255-1)}{9-2} \right) = 109$$

$$I = \begin{bmatrix} 109 & 109 & \cancel{30} \\ \cancel{40} & 109 & \cancel{30} \\ \cancel{50} & \cancel{1} & \cancel{1} \end{bmatrix}$$



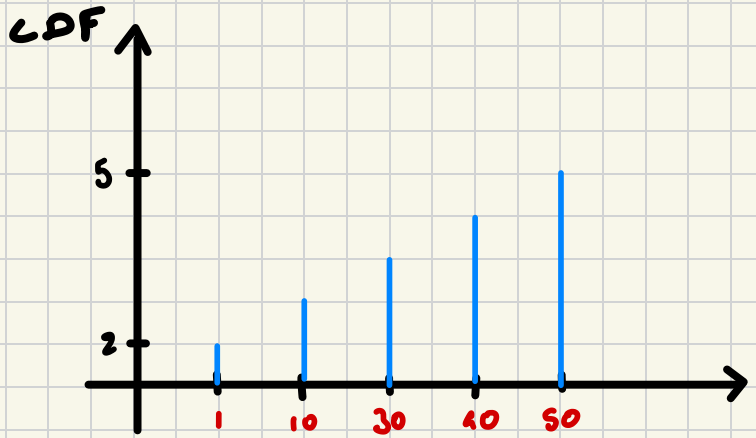
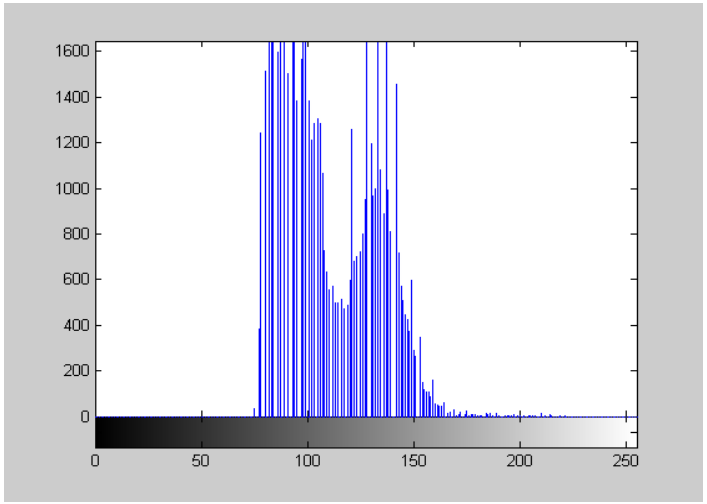
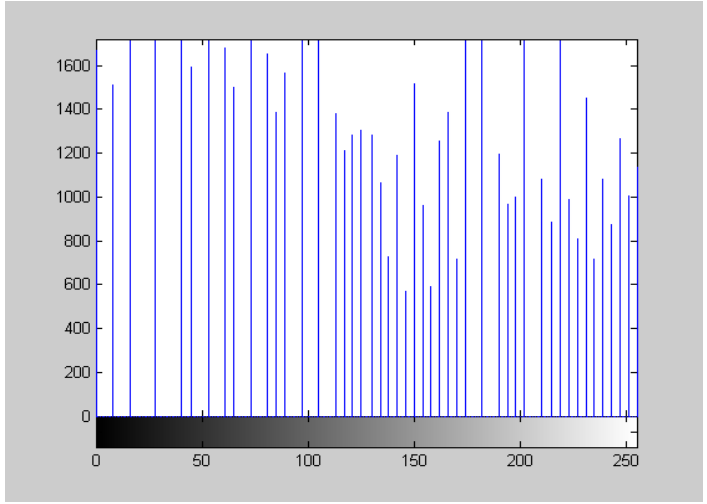


Image Enhancement by Point Processing

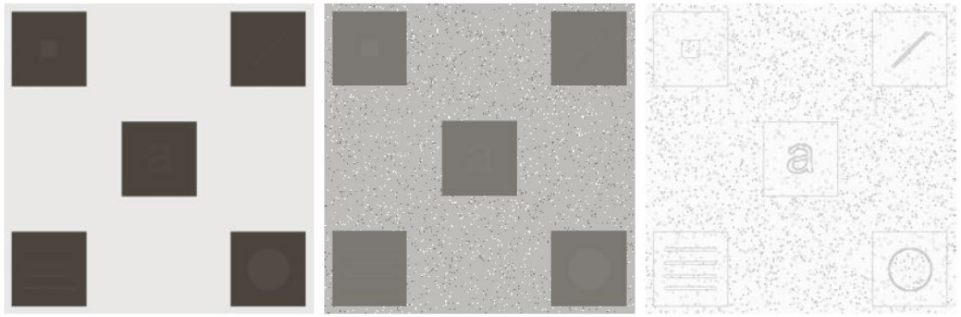
- Histogram Equalization





Local Histogram processing

- Histogram equalization is a global approach.
- Local histogram equalization is realized selecting, for each pixel, a suitable neighborhood on which the histogram equalization (or matching) is computed.
- More computational intensive
- Non overlapping regions may produce "blocky" effect.
- Adaptive histogram equalization (AHE), Contrast limited adaptive histogram equalization (CLAHE)



a | b | c

(a) original image
(b) equalized image
(c) locally equalized image (3×3 neighborhood)

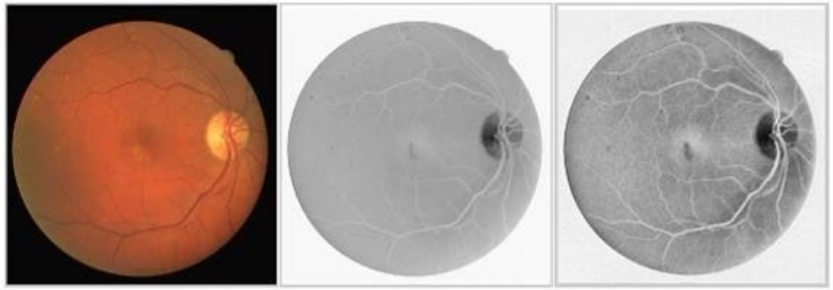


Image Transformations & Filtering

- Point Processing
- Linear Filtering
- Sampling & Aliasing
- Image Derivatives
- Edge Detection

Filters

- Filtering
  - Form a new image whose pixel values are a combination of the original pixel values.

- Why?
  - To get useful information from images
    - E.g., extract edges or contours (to understand shape)
  - To enhance the image
    - E.g., to remove noise
    - E.g., to sharpen and “enhance image”
  - A key operator in Convolutional Neural Networks

Canonical Image Processing Problems

- Image Restoration
  - denoising
  - deblurring
- Image Compression
  - JPEG, HEIF, MPEG, …
- Computing Field Properties
  - optical flow
  - disparity
- Locating Structural Features
  - corners
  - edges

Noise reduction

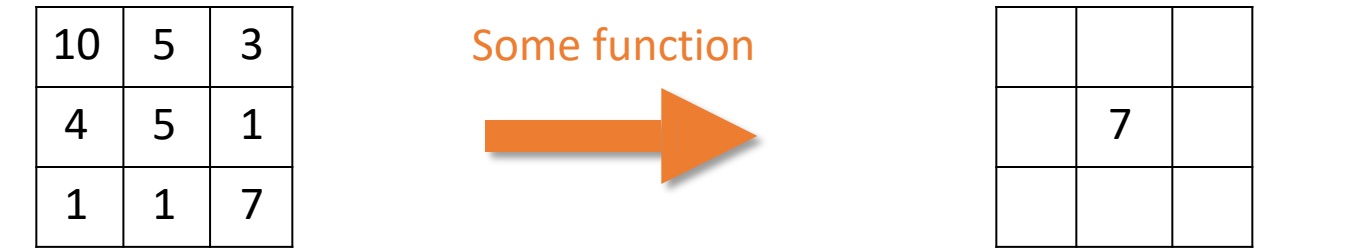
- Given a camera and a still scene, how can you reduce noise?

Take lots of images and average them!
What's the next best thing?

Source: S. Seitz

Image filtering

- Modify the pixels in an image based on some function of a local neighborhood of each pixel



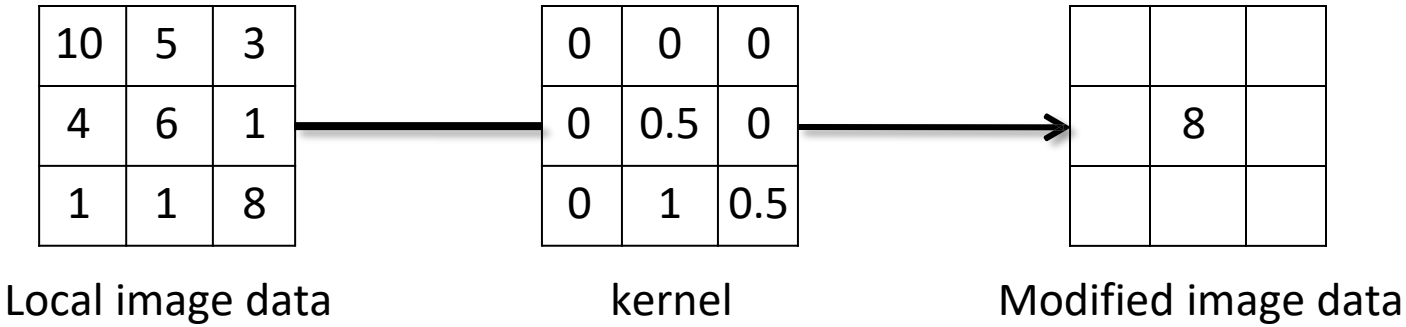
Local image data

Modified image data

Source: L. Zhang

Linear filtering

- One simple version of filtering: linear filtering (cross-correlation, convolution)
  - Replace each pixel by a linear combination (a weighted sum) of its neighbors
- The prescription for the linear combination is called the “kernel” (or “mask”, “filter”)



Source: L. Zhang

Cross-correlation

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

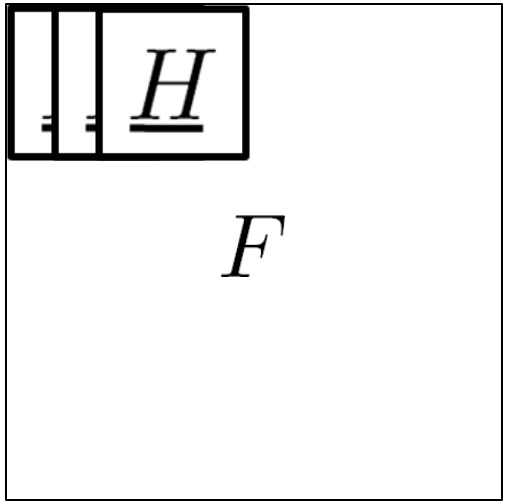
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

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

Cross-correlation



- Sometimes called just **correlation**
- Neither associative nor commutative
- In the literature, people often just call "convolution"
- Filters often symmetric, so won't matter

Adapted from F.

Convolution

- Same as cross-correlation, except that the kernel is "flipped" (horizontally and vertically)

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

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

Cross-correlation

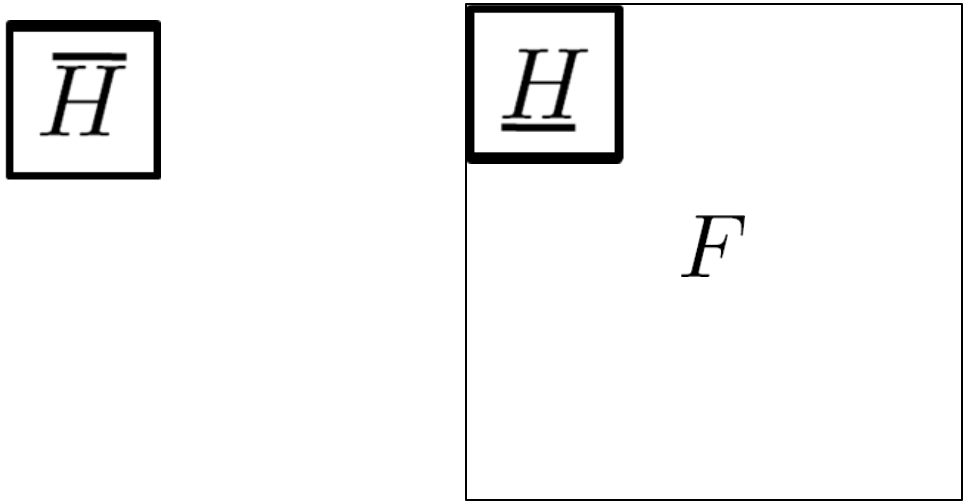
- Convolution is **commutative** and **associative**

Convolution is nice!

- Notation: $b = c \star a$
- Convolution is a multiplication-like operation
  - commutative
  - associative $a \star b = b \star a$
  - distributes o $a \star (b \star c) = (a \star b) \star c$
  - scalars factor out $a \star (b + c) = a \star b + a \star c$
  - identity: unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$ $\alpha a \star b = a \star \alpha b = \alpha(a \star b)$ $a \star e = a$
- Usefulness of associativity
  - often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - this is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$

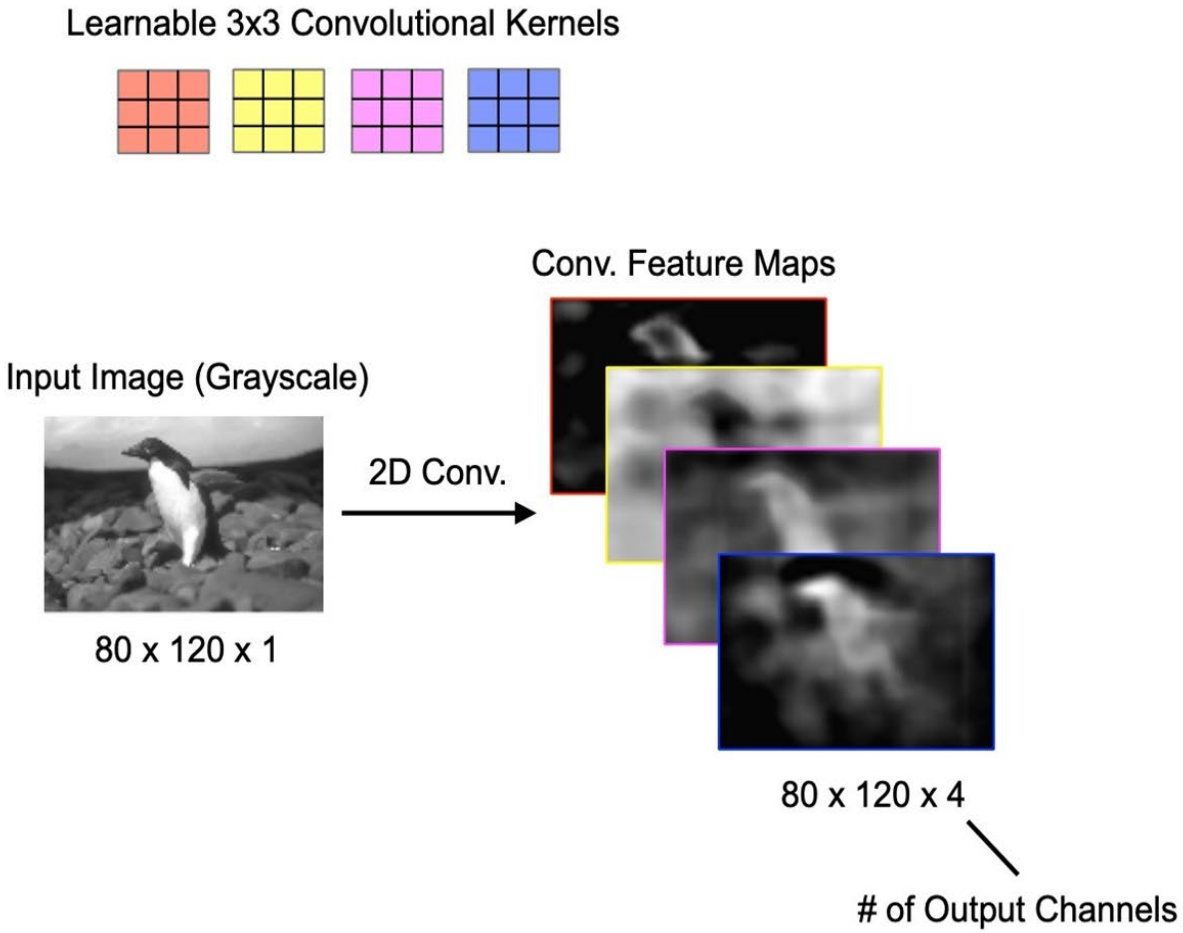
Convolution

The kernel is “flipped” (horizontally and vertically)



Adapted from F.

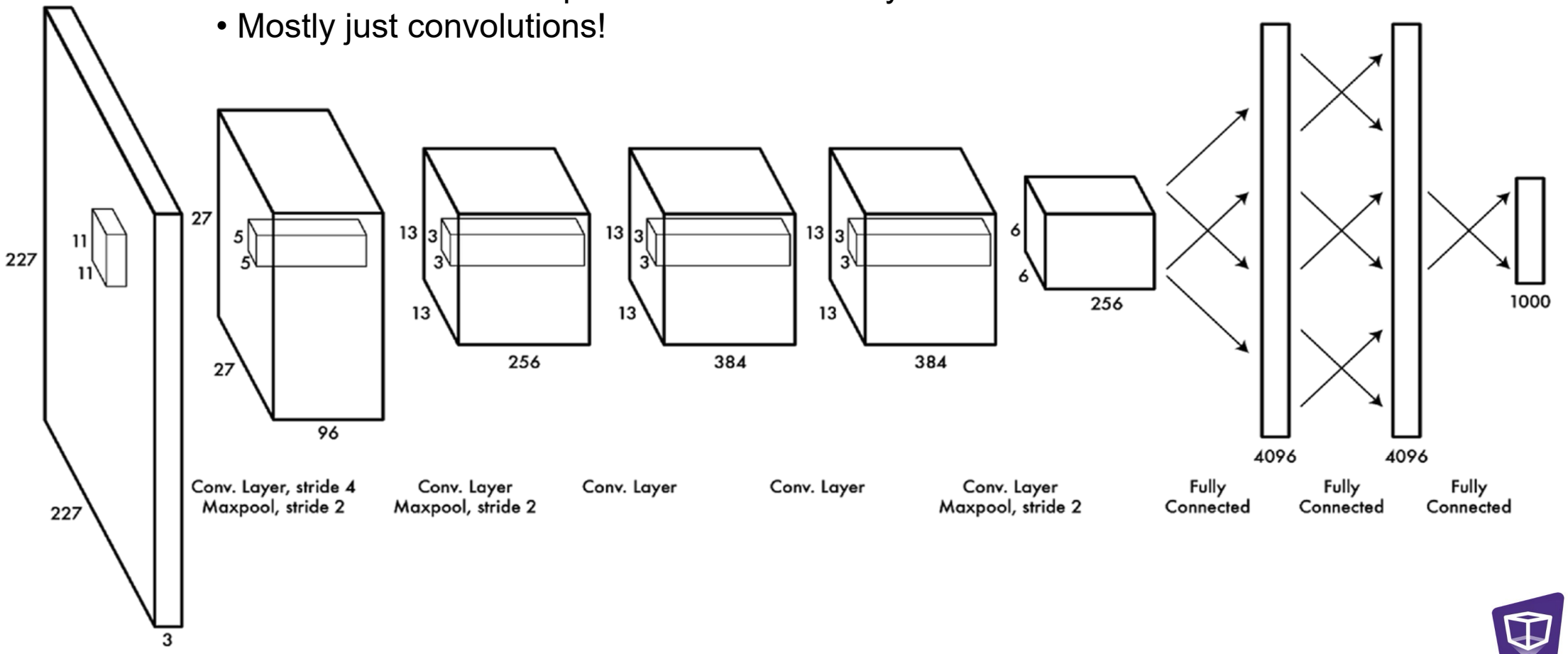
Convolutional Networks



Slide Credits: Gedas Bertasius

AlexNet: An Early Example

- Neural network with specialized connectivity structure
- Mostly just convolutions!



https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

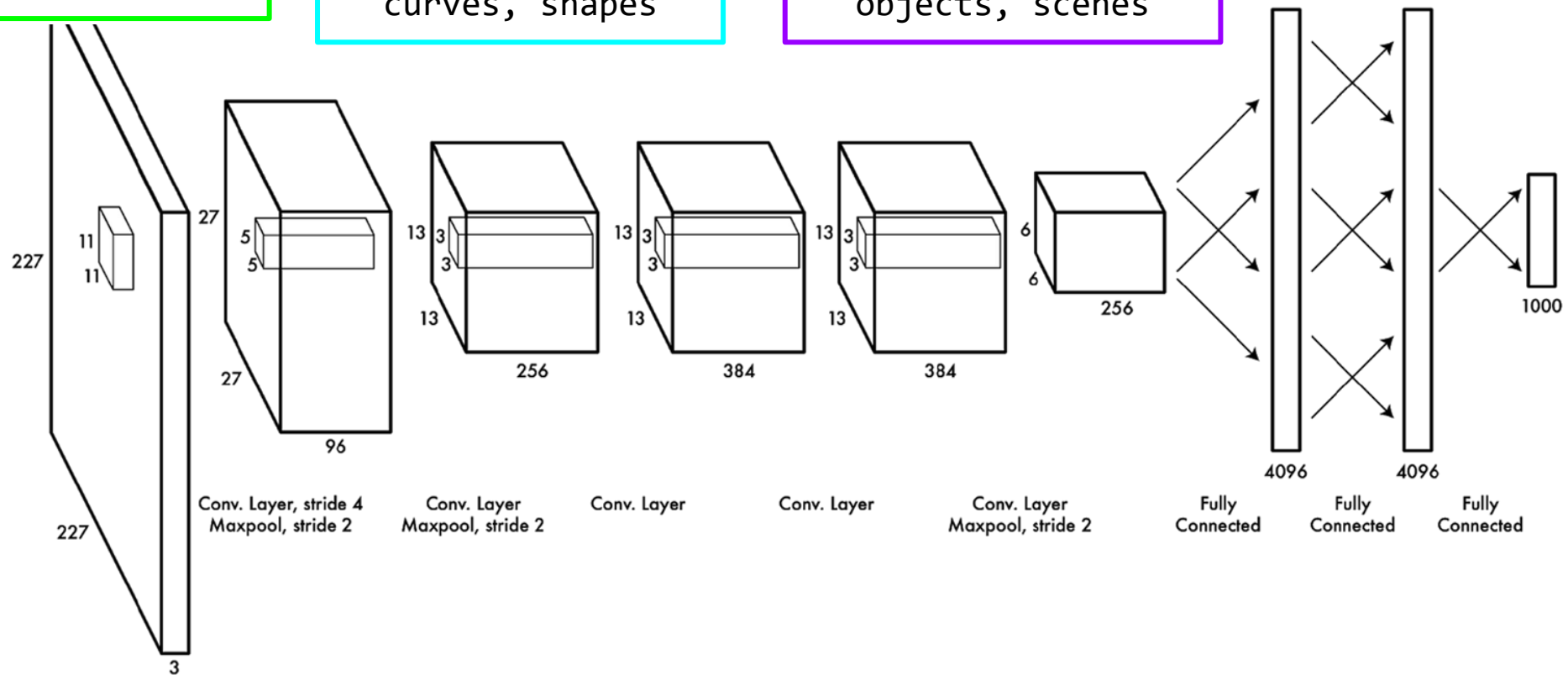
Where Models Learn Features of an Image

Low-level features
Lines, oriented edges

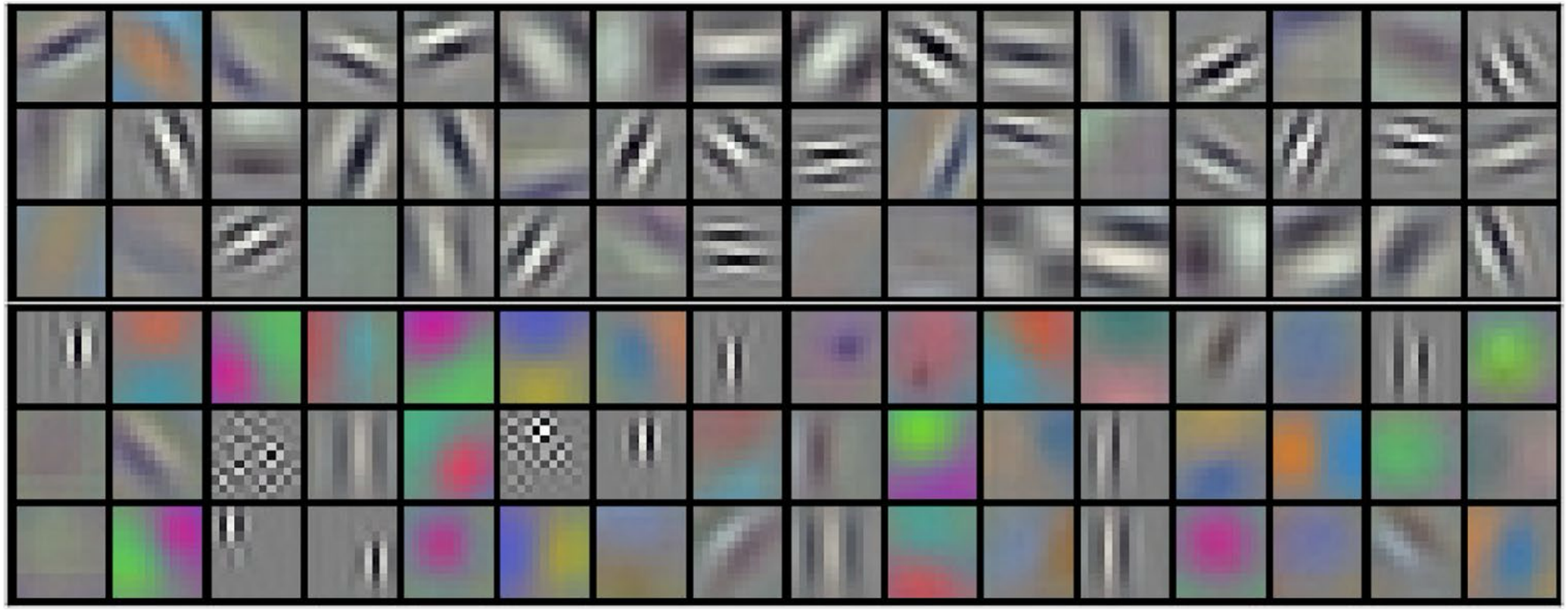
Mid-level features
Combine edges: curves, shapes

High-level features
Combine shapes: objects, scenes

Predictor
Process features and predict output

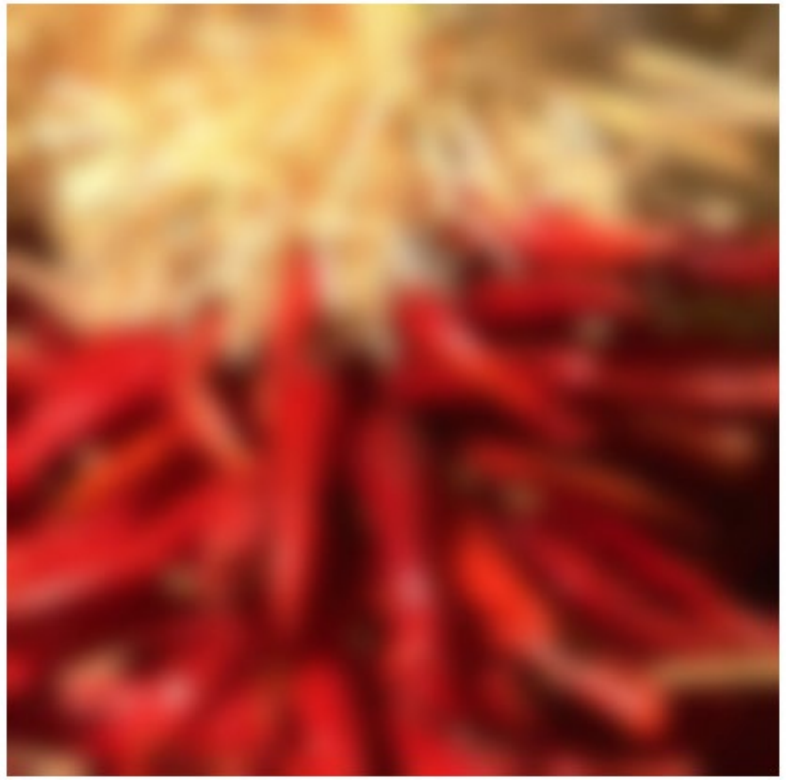


What are Models Actually Looking For?



Practical matters

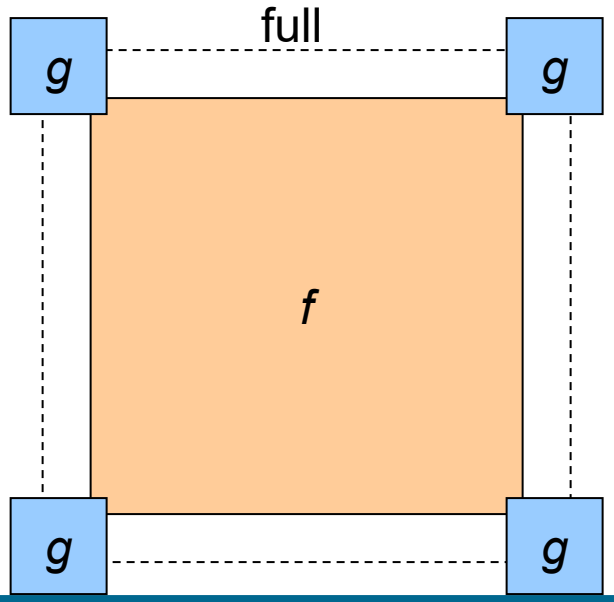
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - Methods (padding):
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

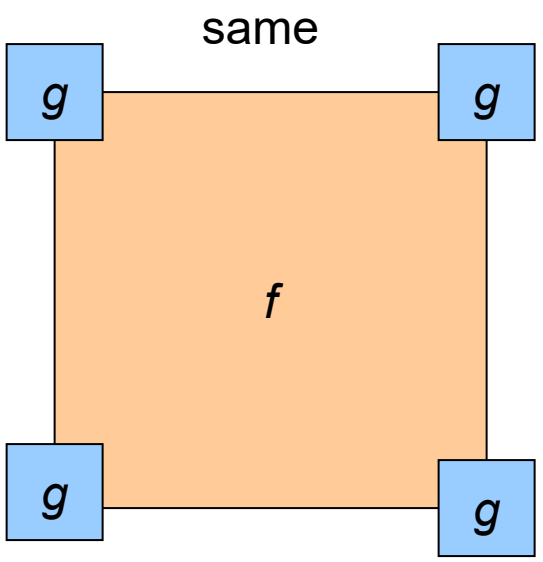


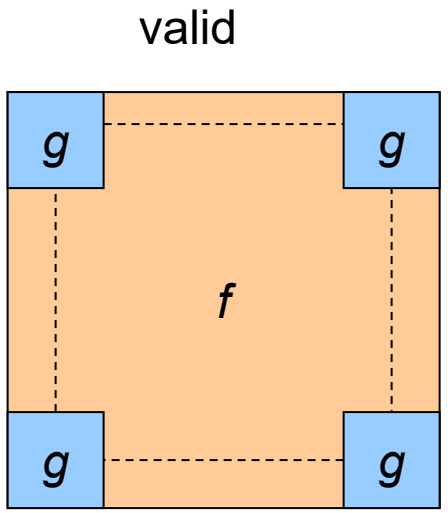
Source: S. Marschner

Practical matters

- What is the size of the output?
  - *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

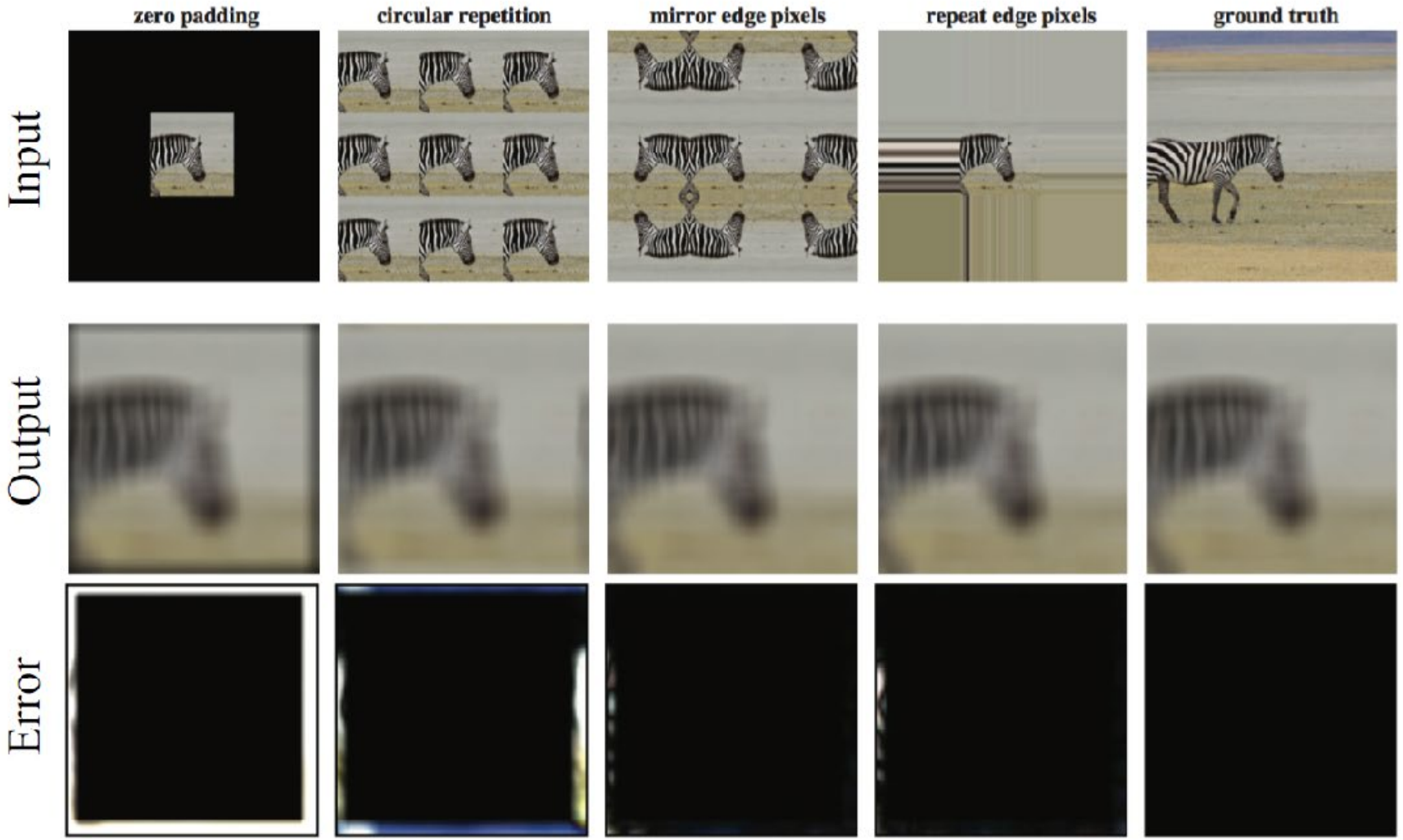






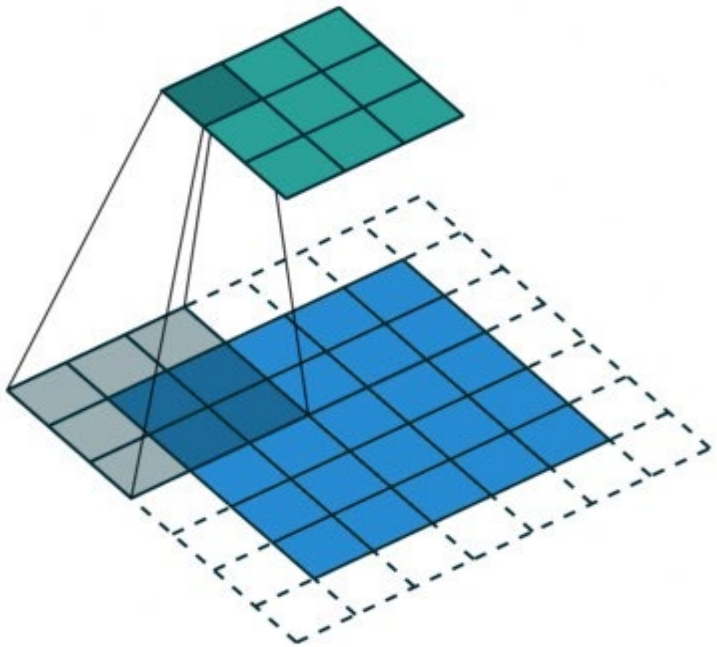
Source: S. Lazebnik

Handling boundaries



Source: Torralba, Freeman, Isola

Padding & Stride in CNN



padding=1, stride=2

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

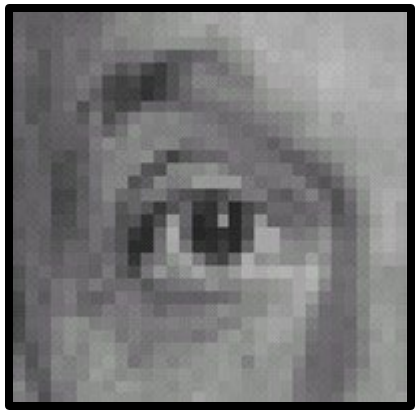
$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

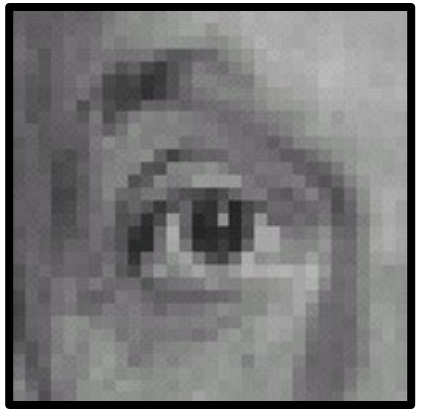
Linear filters: examples



Original

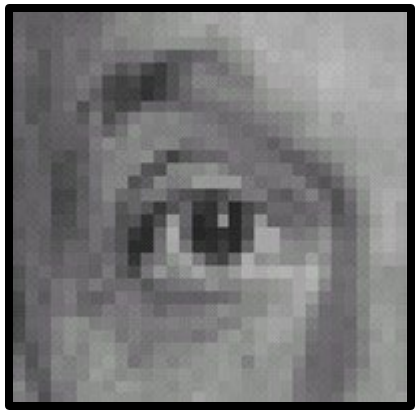
$$*$$

0	0	0
0	1	0
0	0	0



Source: D. Lowe

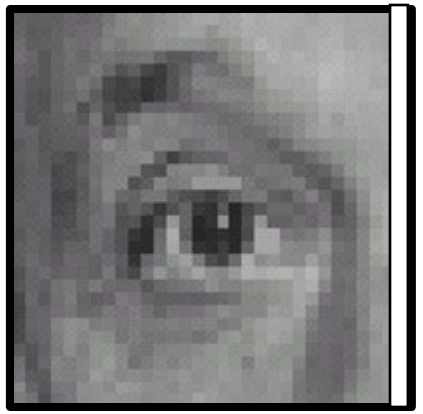
Linear filters: examples



Original

$*$

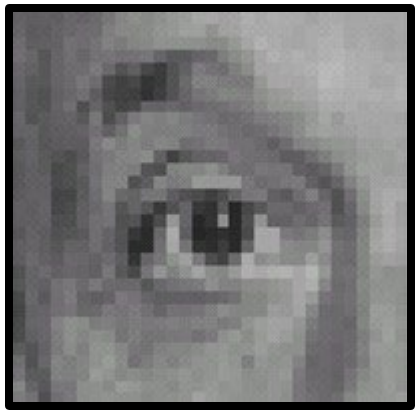
0	0	0
1	0	0
0	0	0



Shifted left by 1 pixel

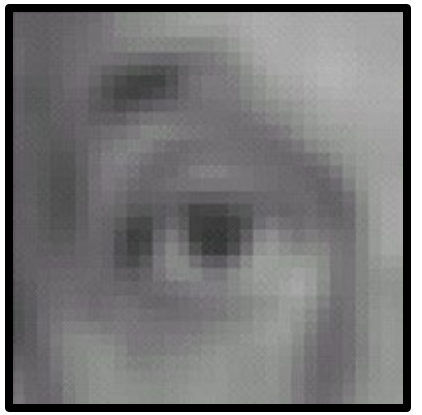
Source: D. Lowe

Linear filters: examples



Original

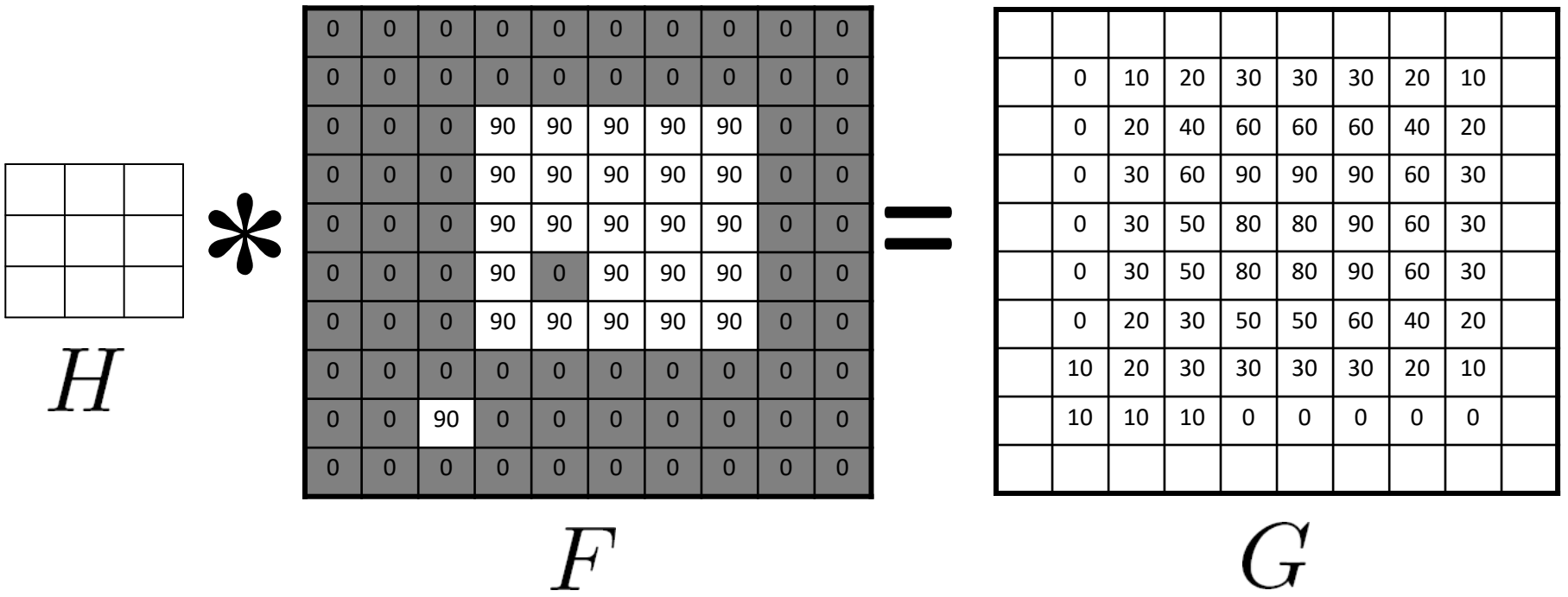
$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Blur (with a mean filter)

Source: D. Lowe

Mean filtering

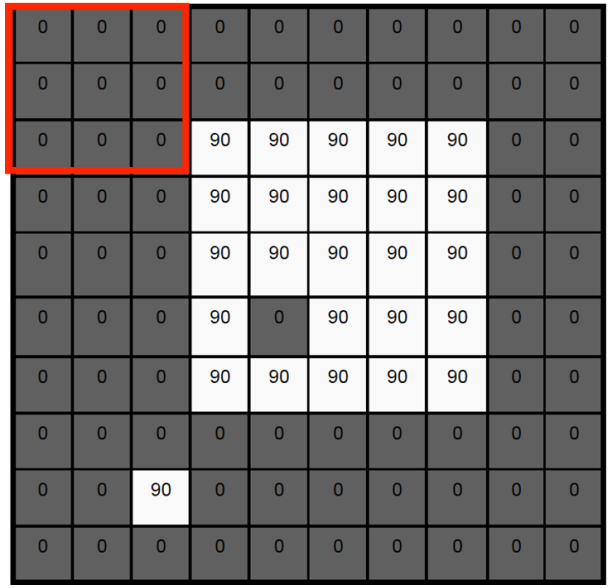


0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

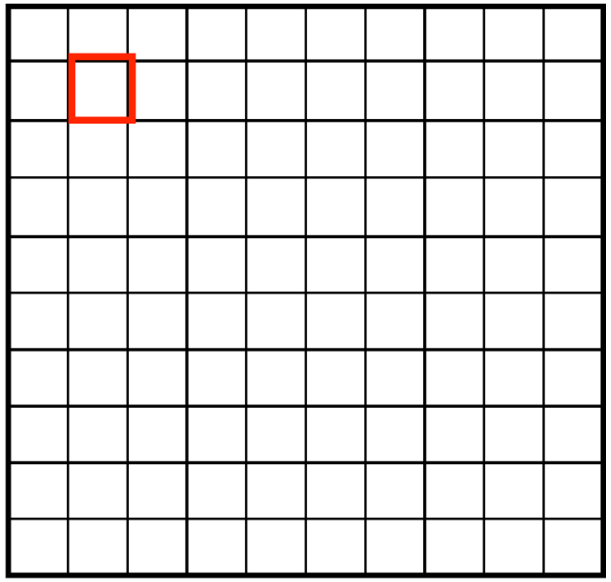
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Mean filtering/Moving average

$F[x, y]$

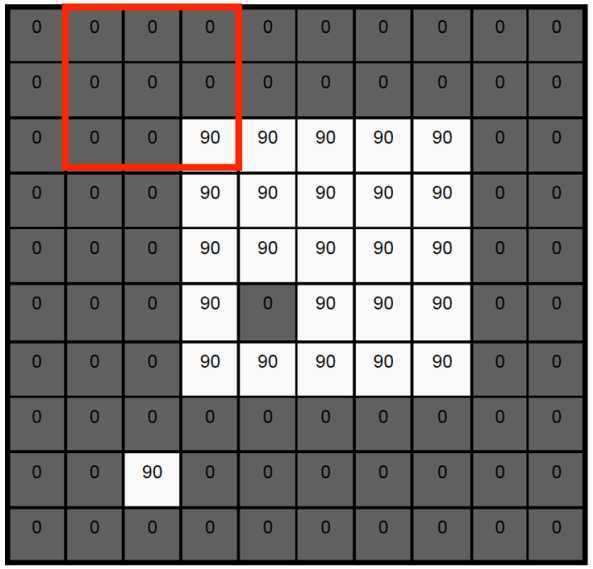


$G[x, y]$

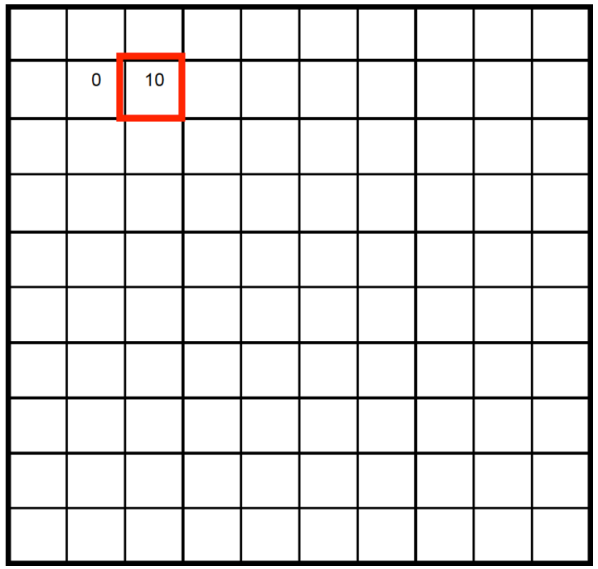


Mean filtering/Moving average

$F[x, y]$

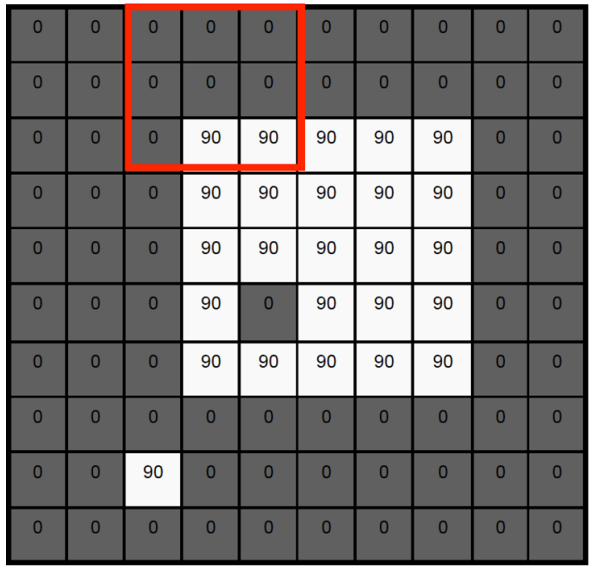


$G[x, y]$

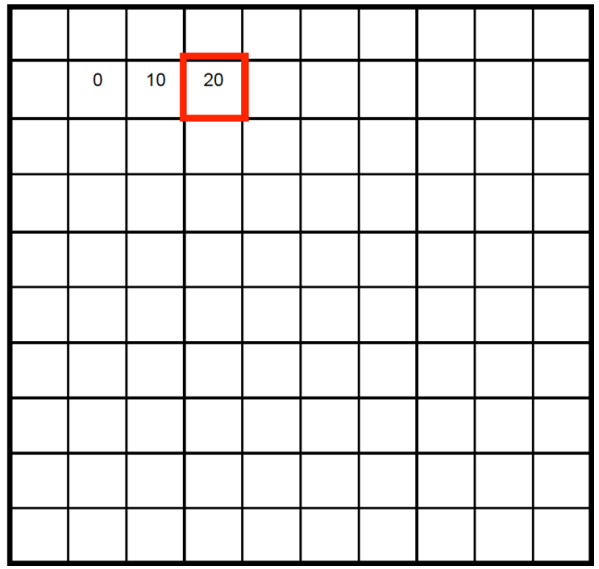


Mean filtering/Moving average

$F[x, y]$

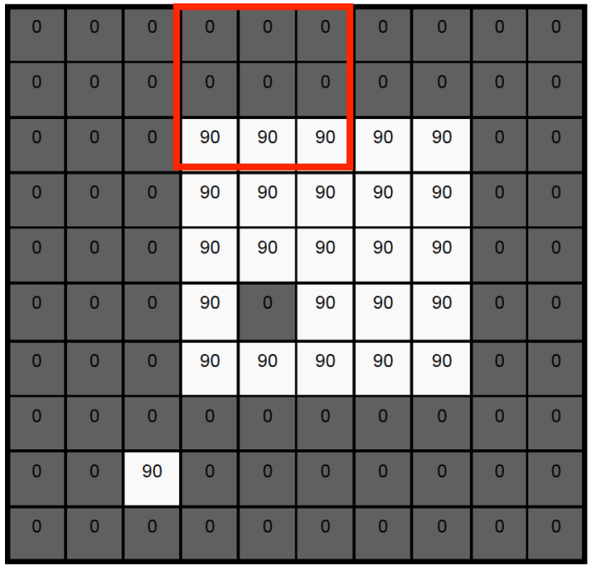


$G[x, y]$

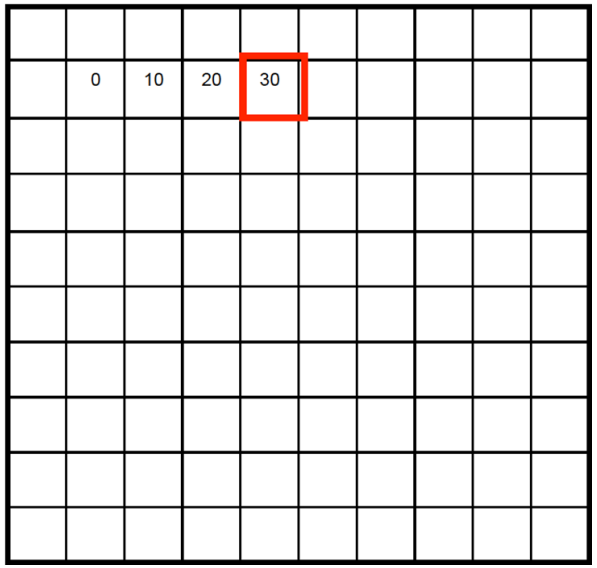


Mean filtering/Moving average

$F[x, y]$

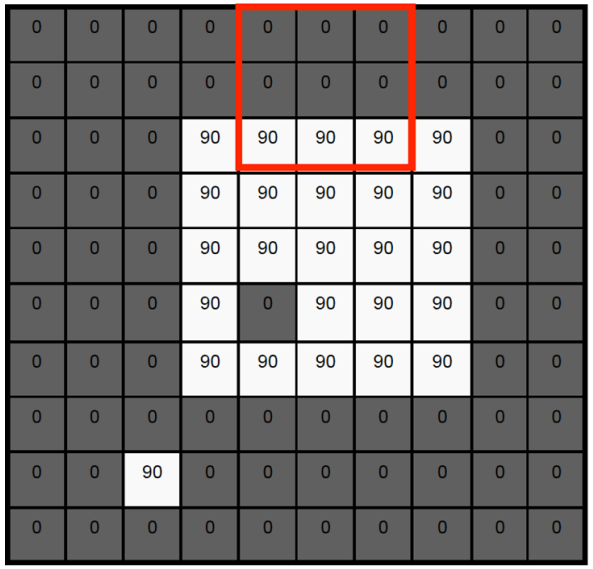


$G[x, y]$

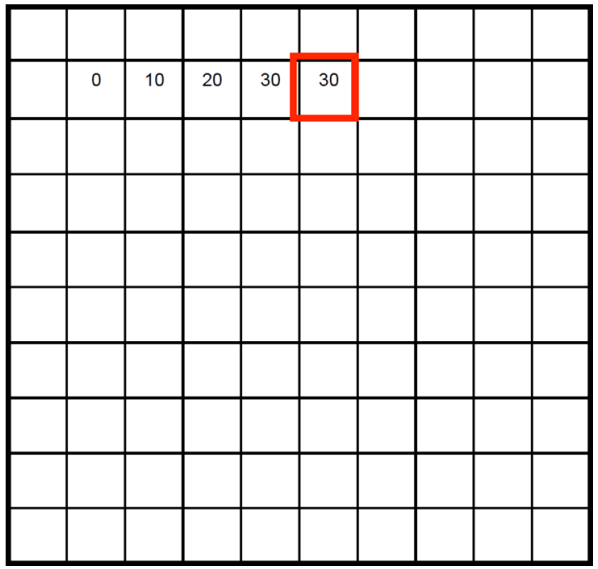


Mean filtering/Moving average

$F[x, y]$



$G[x, y]$



Mean filtering/Moving average

$F[x, y]$

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

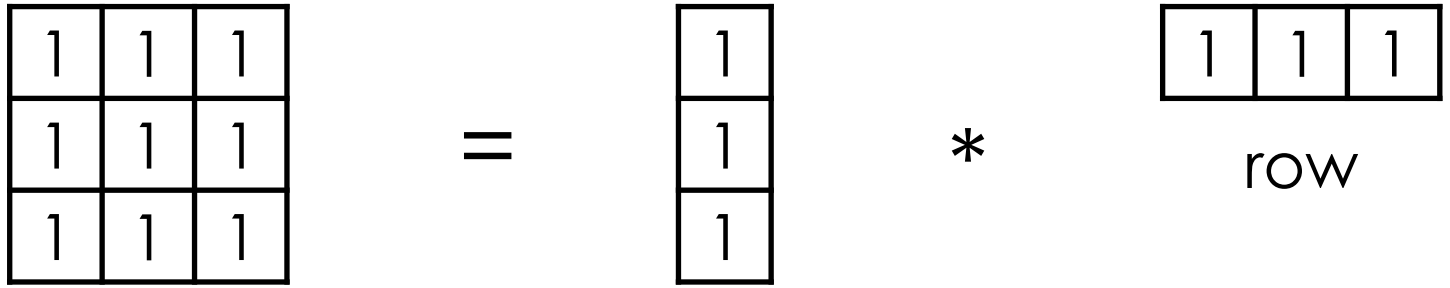
$G[x, y]$

	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	

Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter

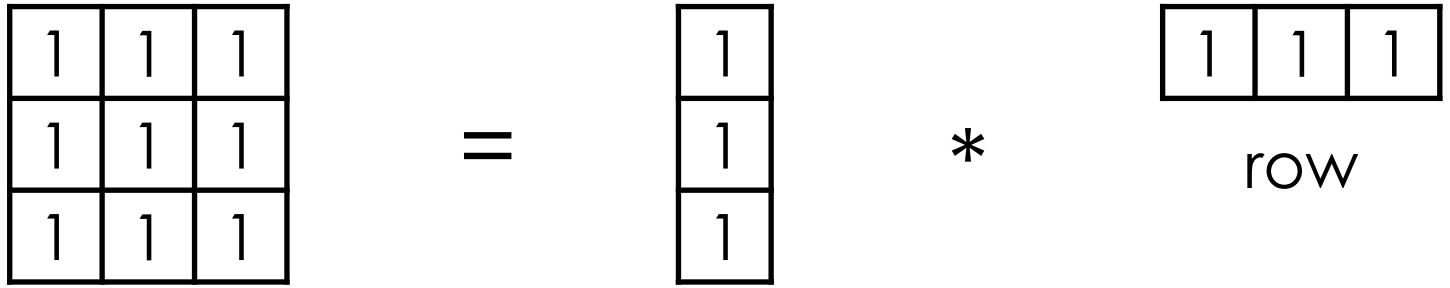


2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

Separable filters

A 2D filter is separable if it can be written as the product of a "column" and a "row".

example:
box filter



2D convolution with a separable filter is equivalent to two 1D convolutions (with the "column" and "row" filters).

If the image has M x N pixels and the filter kernel has size L x L:

- What is the cost of convolution with a non-separable filter? ⟶ L^2 x MxN
- What is the cost of convolution with a separable filter? ⟶ 2 x (L x Mx N)

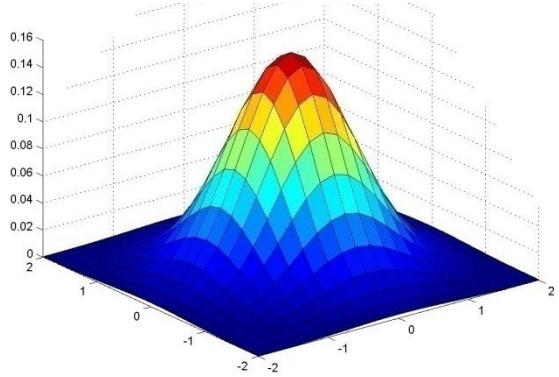
How to verify it?

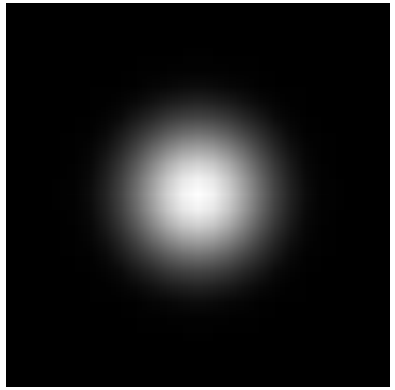
- If a matrix has a rank of 1 (all of its row or columns are scalar multiples of each others). In this case the matrix can be factored as the outer product of two vectors.

Outer product: takes two vector and produces a matrix whose elements are the pairwise products of the elements of the two vectors.

- The outer product of two vectors always results in a matrix of rank 1 (unless one of the vector is the zero vector).

Gaussian kernel

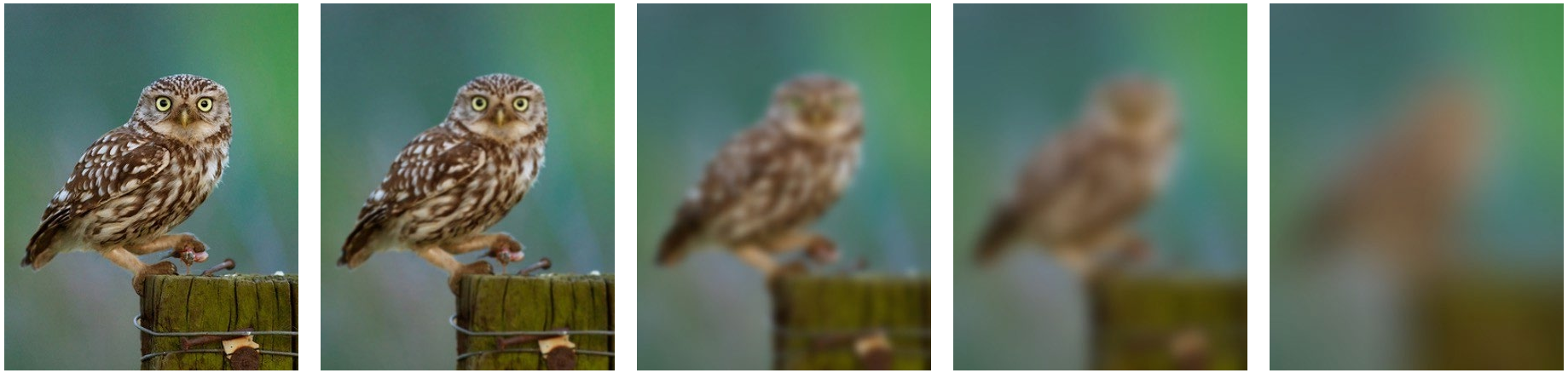




$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Source: C. Rasmussen

Gaussian kernel



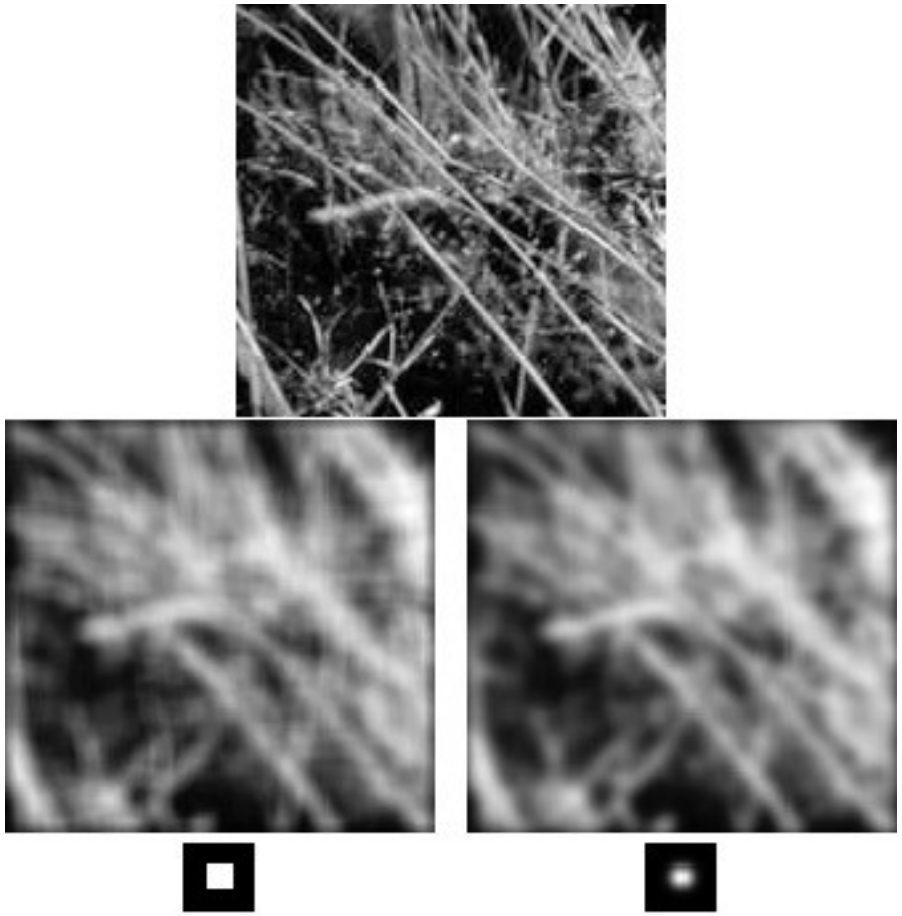
σ = 1 pixel

σ = 5 pixels

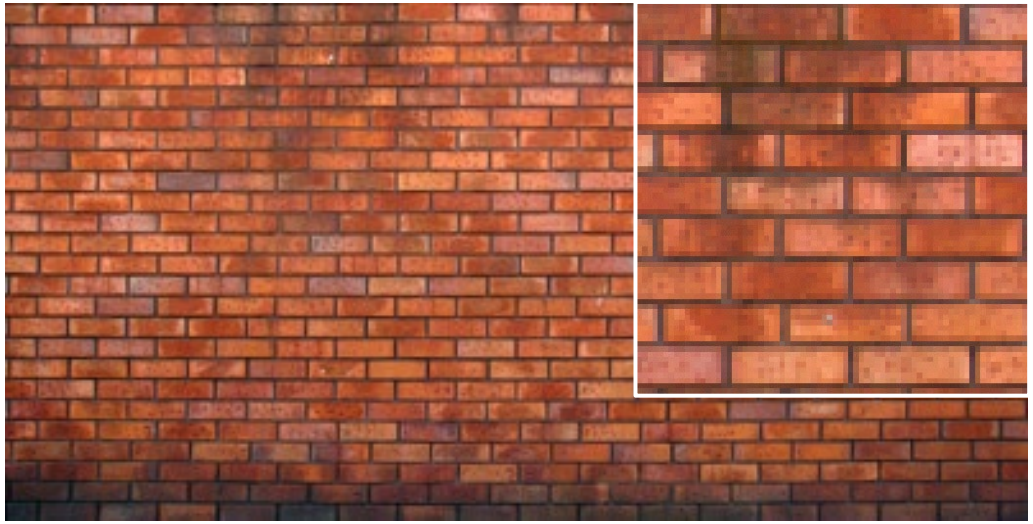
σ = 10 pixels

σ = 30 pixels

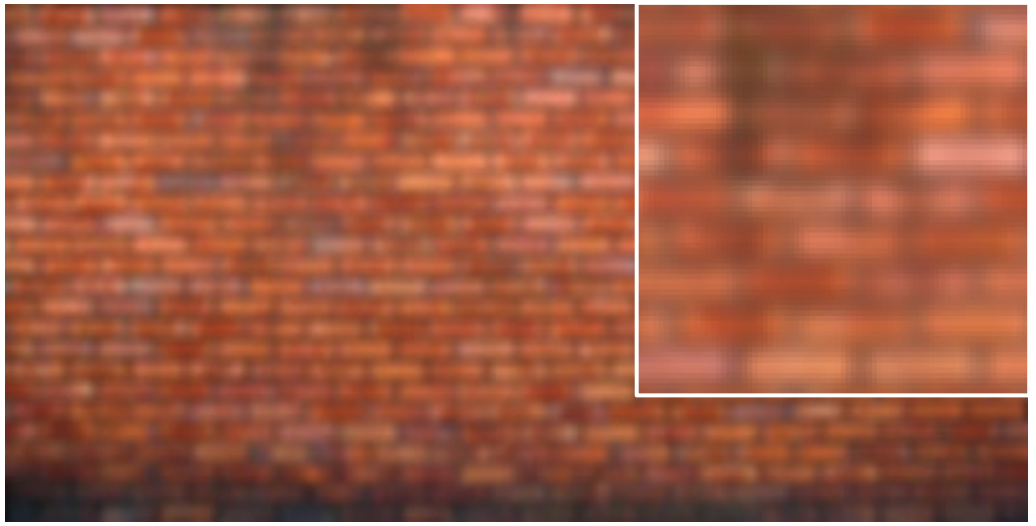
Mean (or box filtering) vs Gaussian filtering



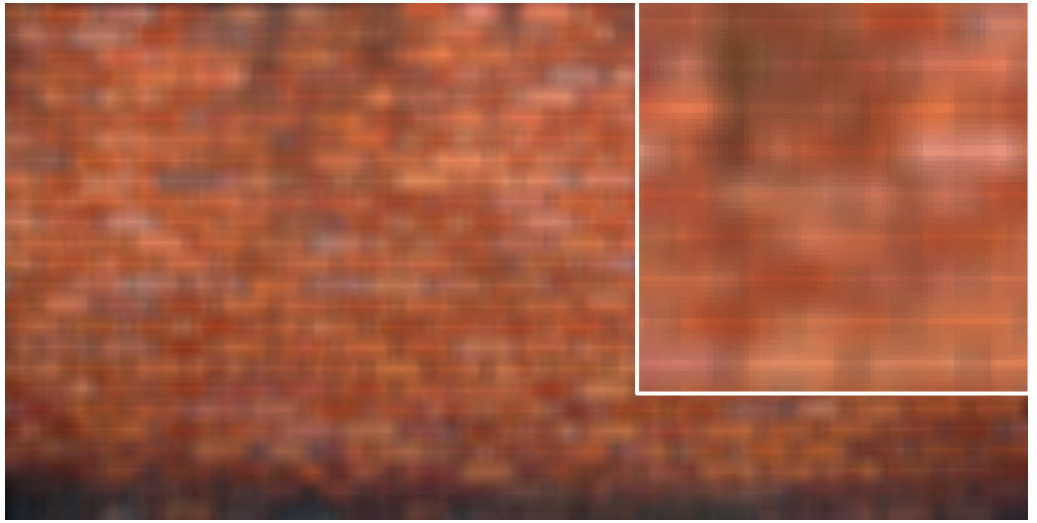
Another example



original



7x7 Gaussian

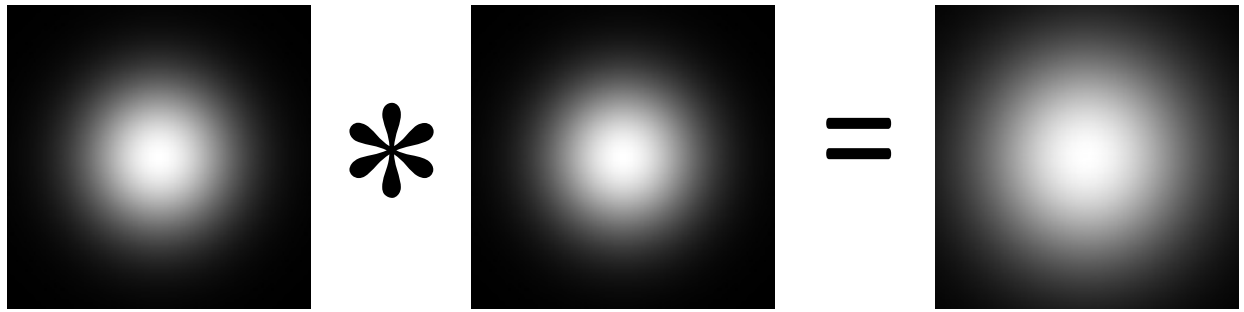


7x7 box

Which blur do you like better?

Gaussian filter

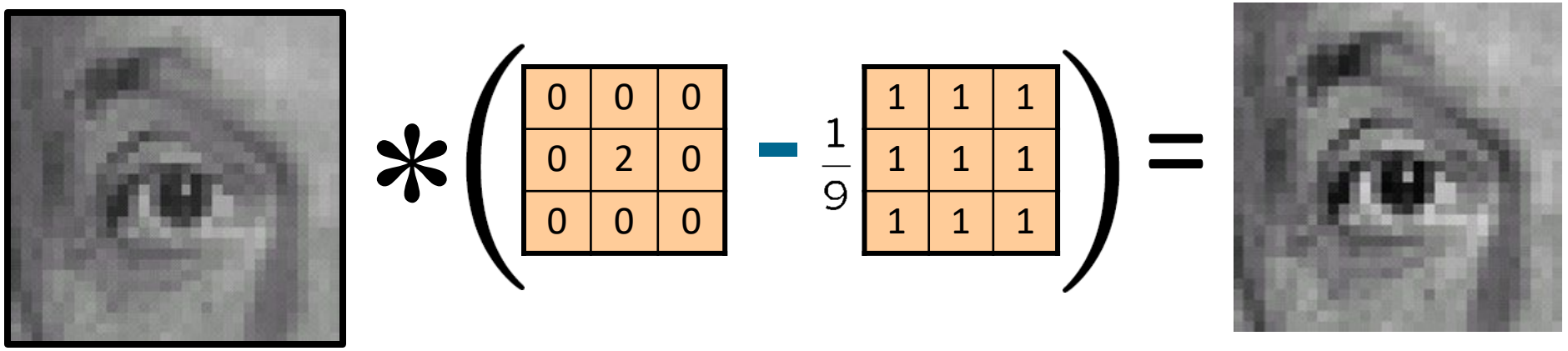
- Removes “high-frequency” components from the image (low-pass filter)
- Convolution with self is another Gaussian



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

Source: K. Grauman

Linear filters: examples



Original

Sharpening filter

(accentuates edges)

Source: D. Lowe

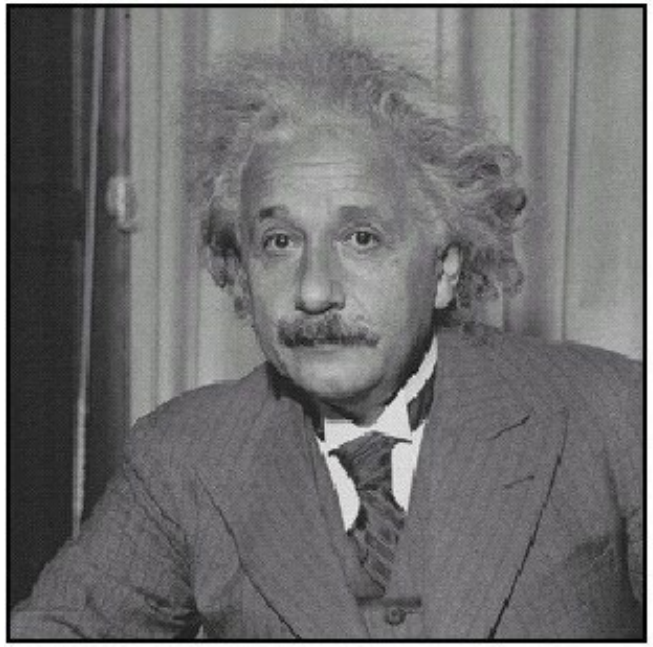
Sharpening (derivation)

+

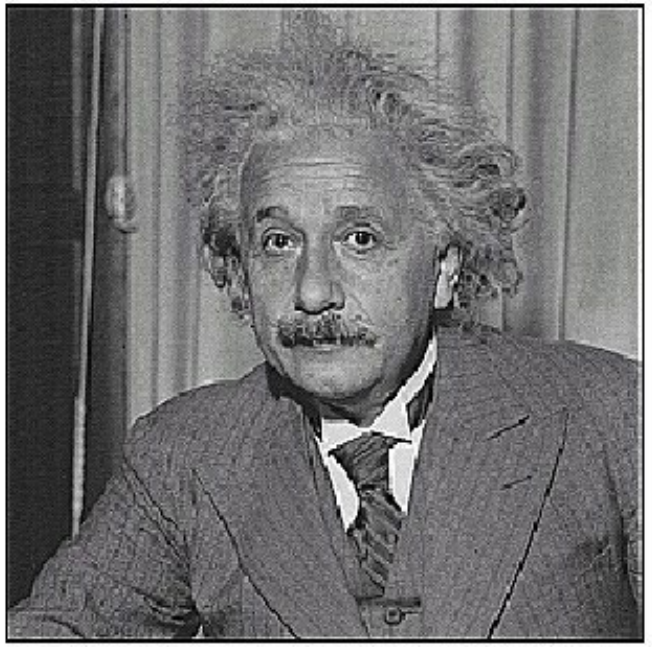
=

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

Sharpening



before

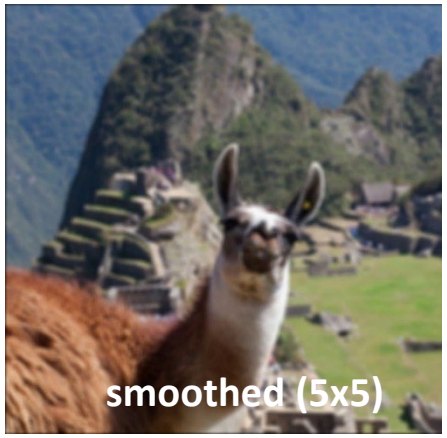


after

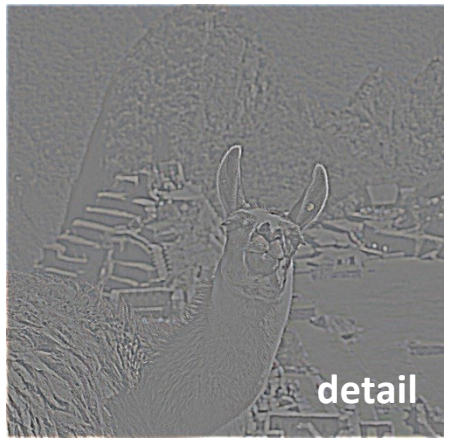
Source: D. Lowe

Sharpening

−



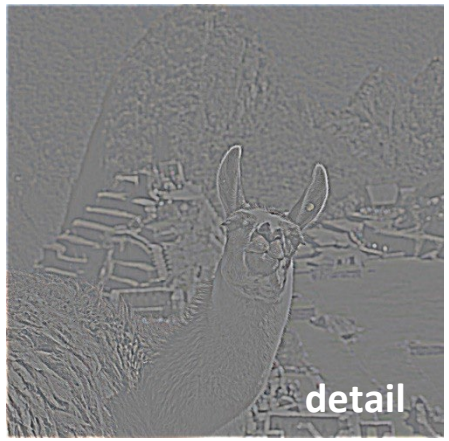
=



(This “detail extraction” operation is also called a ***high-pass filter***)

Let’s add it back:

+ α



=

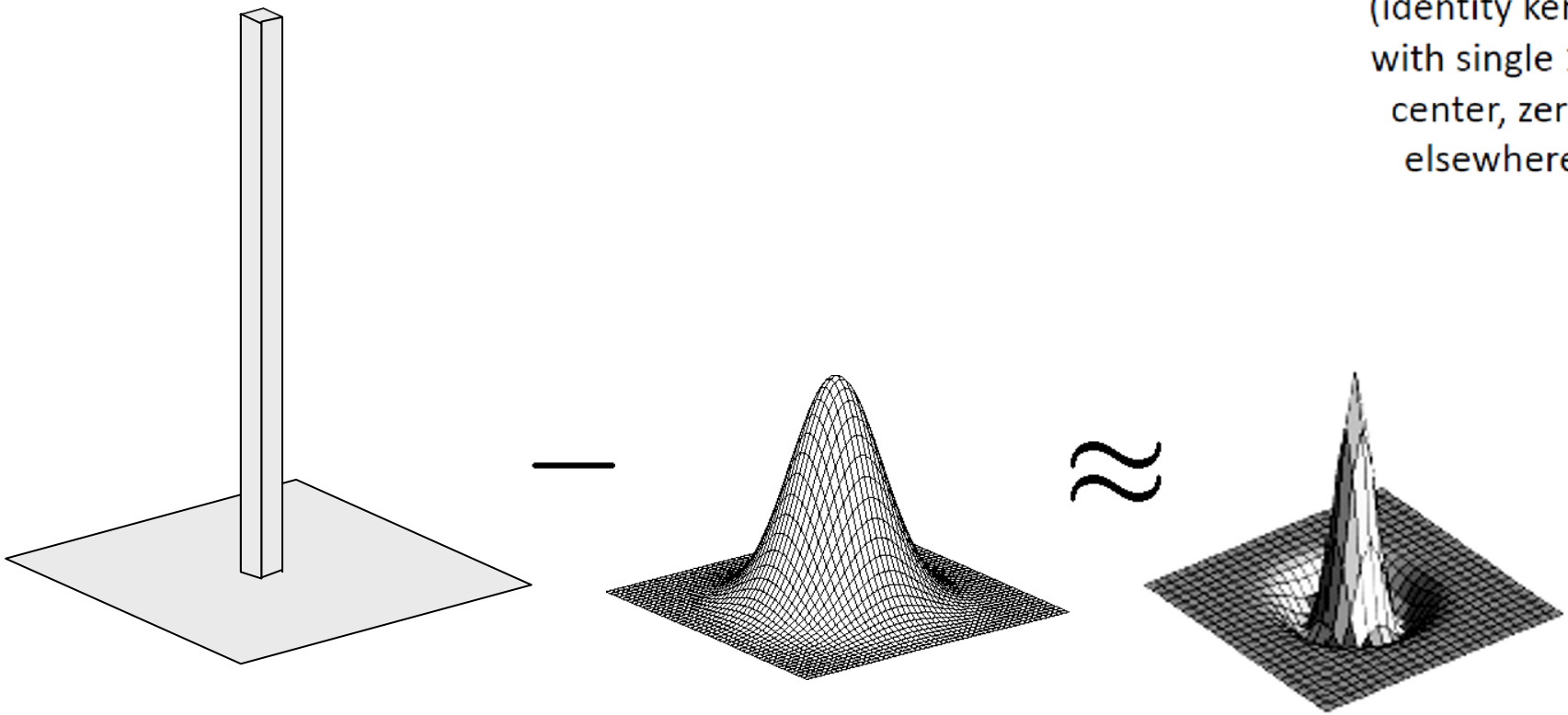
Photo credit: https://www.flickr.com/photos/geezaweezer/16089096376/

Sharpen filter

$$F + \alpha\,(F - \underbrace{F * H}_{\text{blurred image}}) = (1+\alpha)\,F - \alpha\,(F * H) = F * ([1+\alpha]\,e - \alpha H)$$

image

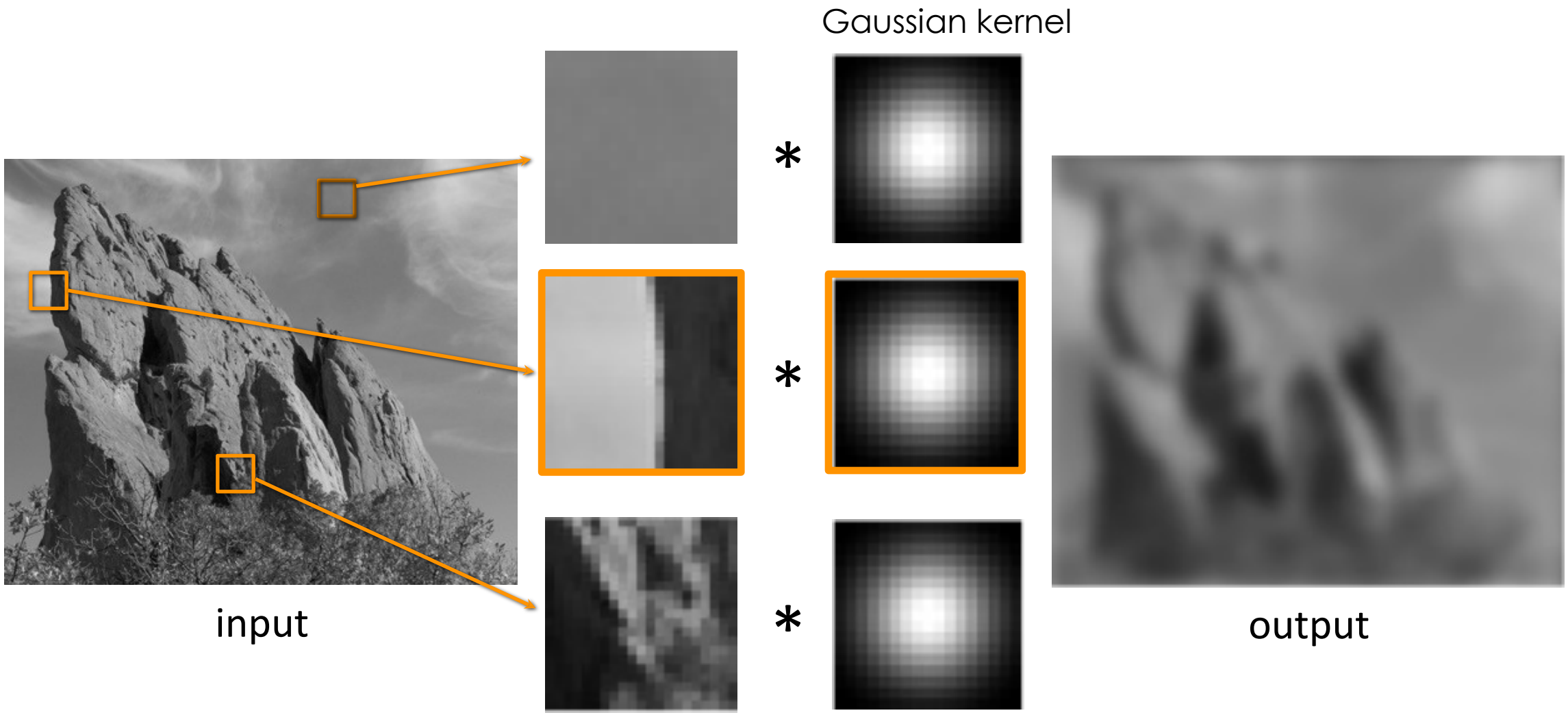
unit impulse (identity kernel with single 1 in center, zeros elsewhere)



scaled impulse — Gaussian ≈ Sharpen filter

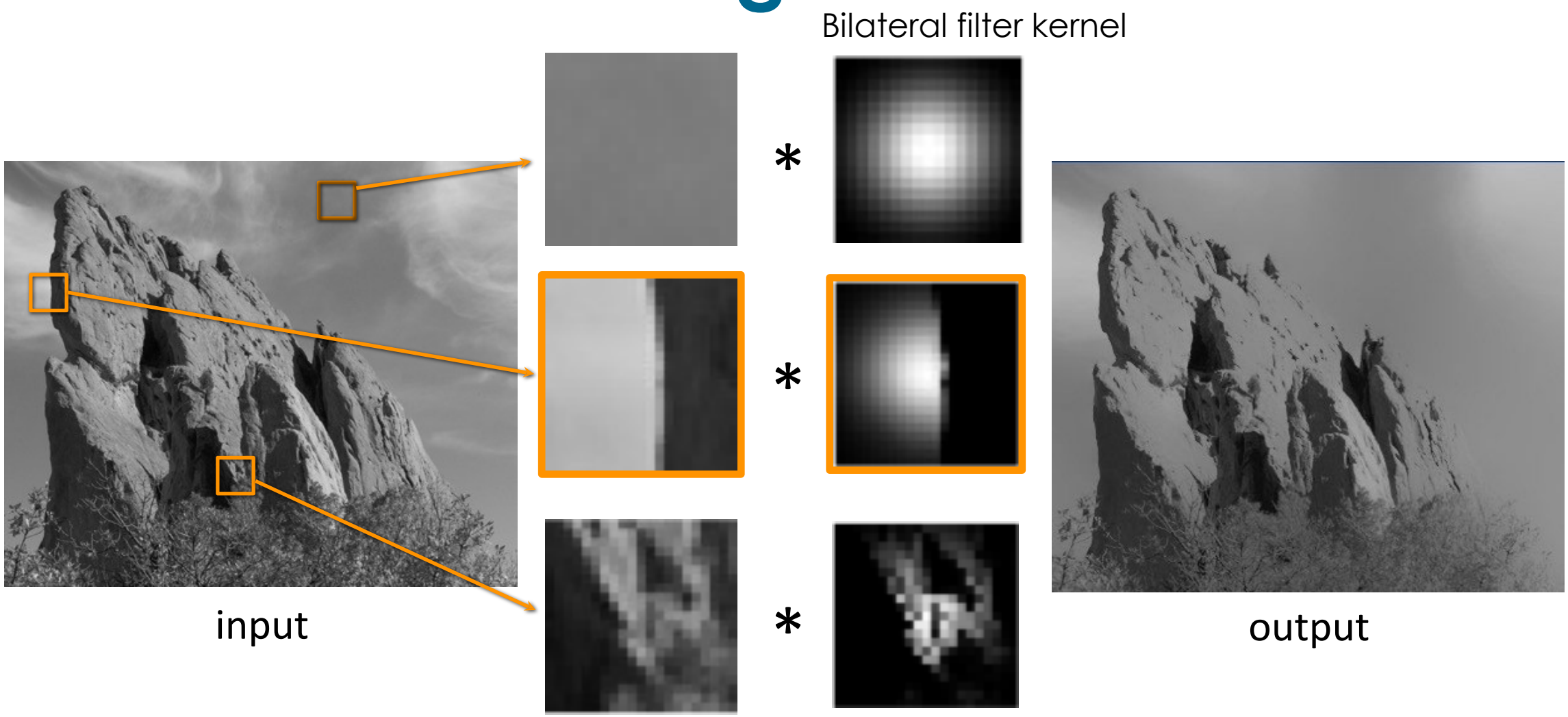
The amount we want to sharpen the image is controlled by a variable alpha, which is the weight of the mask

The problem with Gaussian filtering



Blur kernel averages across edges

The bilateral filtering solution



Do not blur if there is an edge! How does it do that?

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

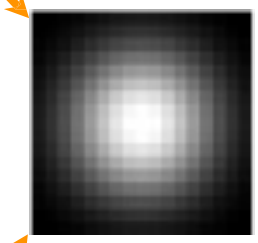
Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

Bilateral filtering

σ_s



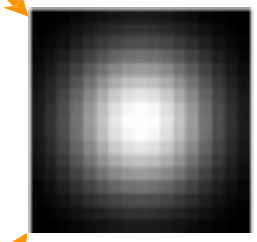
Spatial weighting: favor *nearby* pixels

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

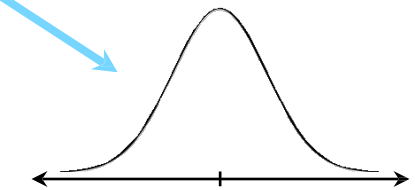


σ_s

Spatial weighting: favor *nearby* pixels

Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$



σ_r

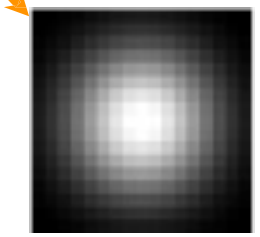
$x = f[m,n] - f[m+k, n+l]$

Intensity range weighting: favor *similar* pixels

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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



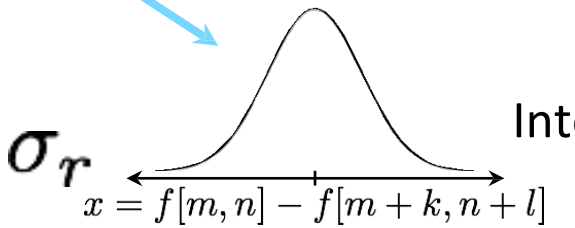
σ_s

Spatial weighting: favor *nearby* pixels

Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Normalization factor



Intensity range weighting: favor *similar* pixels

Bilateral filtering vs Gaussian filtering

Gaussian filtering

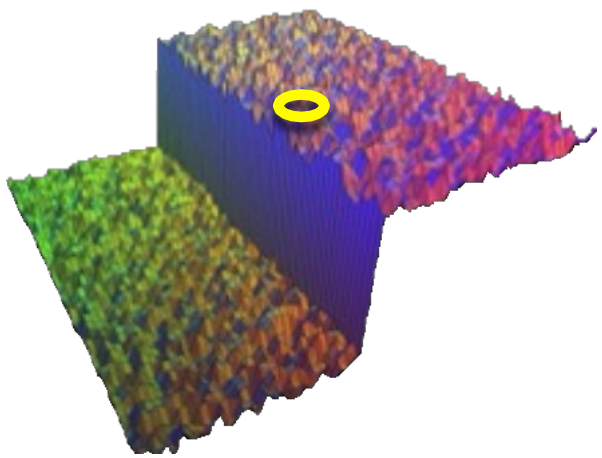
Smooths everything nearby (even edges) Only depends on *spatial* distance

Bilateral filtering

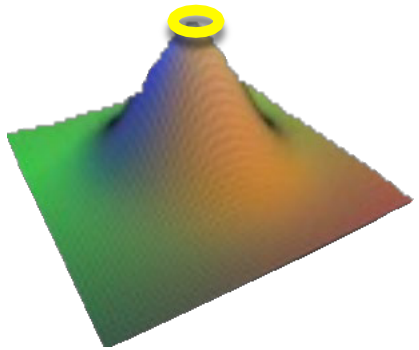
Smooths ‘close’ pixels in space and intensity Depends on *spatial* and *intensity* distance

Gaussian filtering visualization

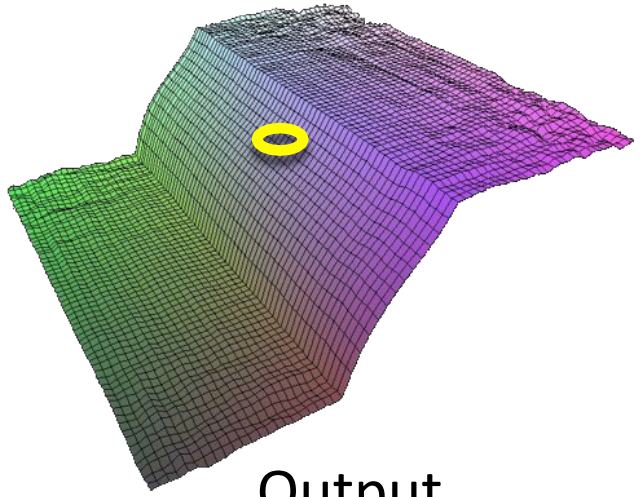
$$h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]$$



Input



Gaussian Filter



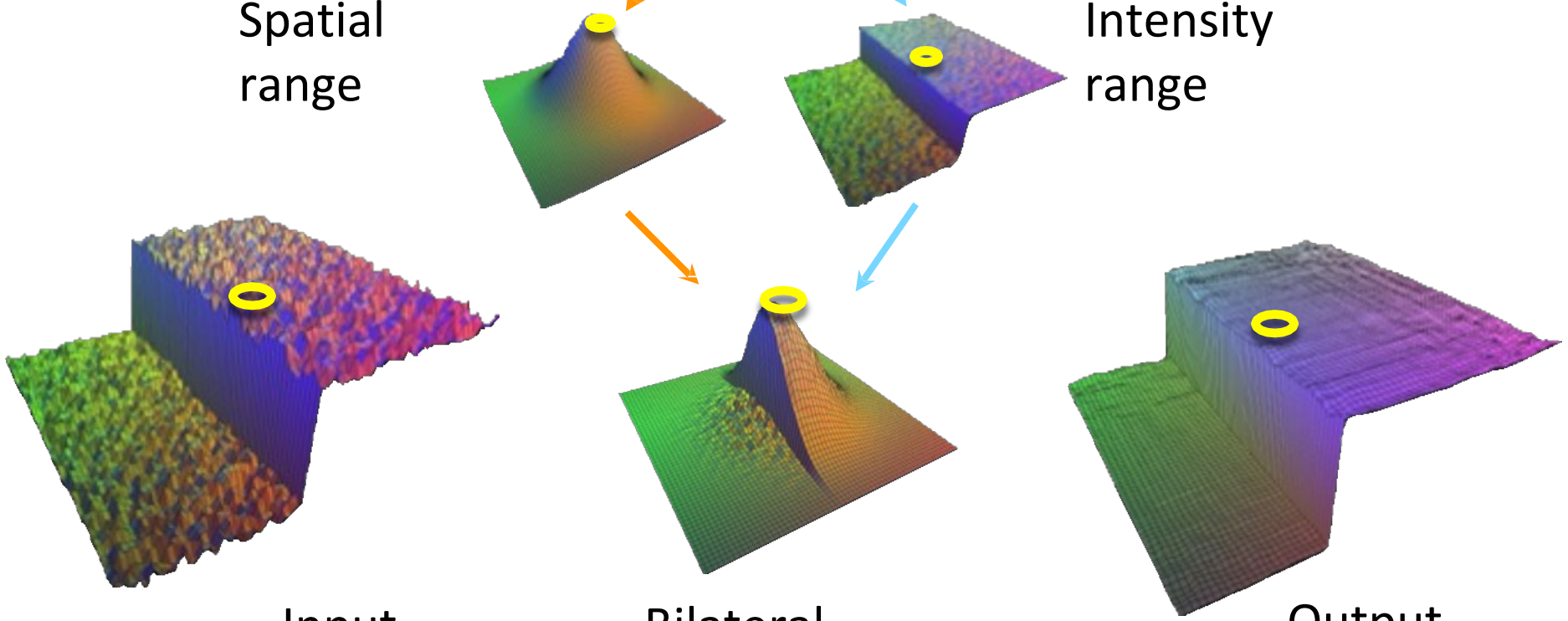
Output

Bilateral filtering visualization

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Spatial range

Intensity range

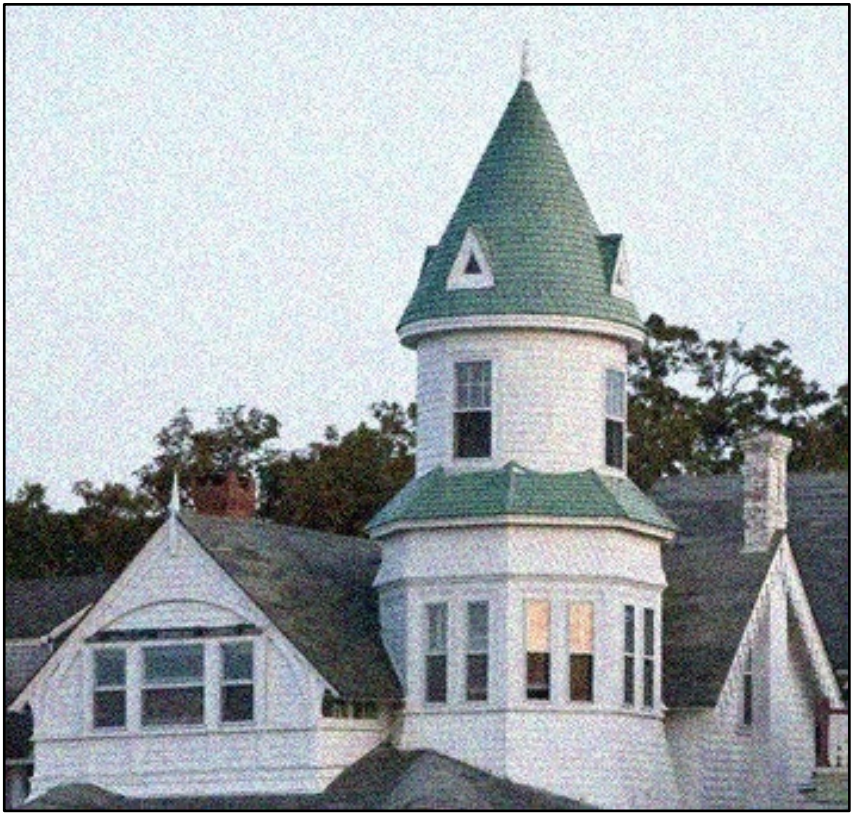


Input

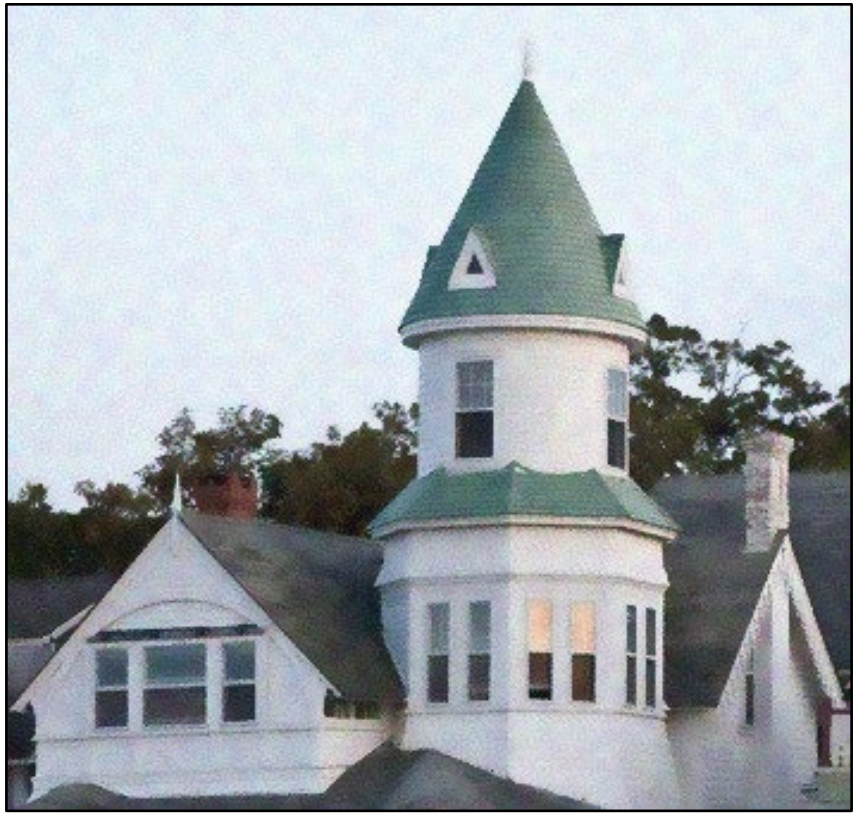
Bilateral Filter

Output

Denoising



noisy
input



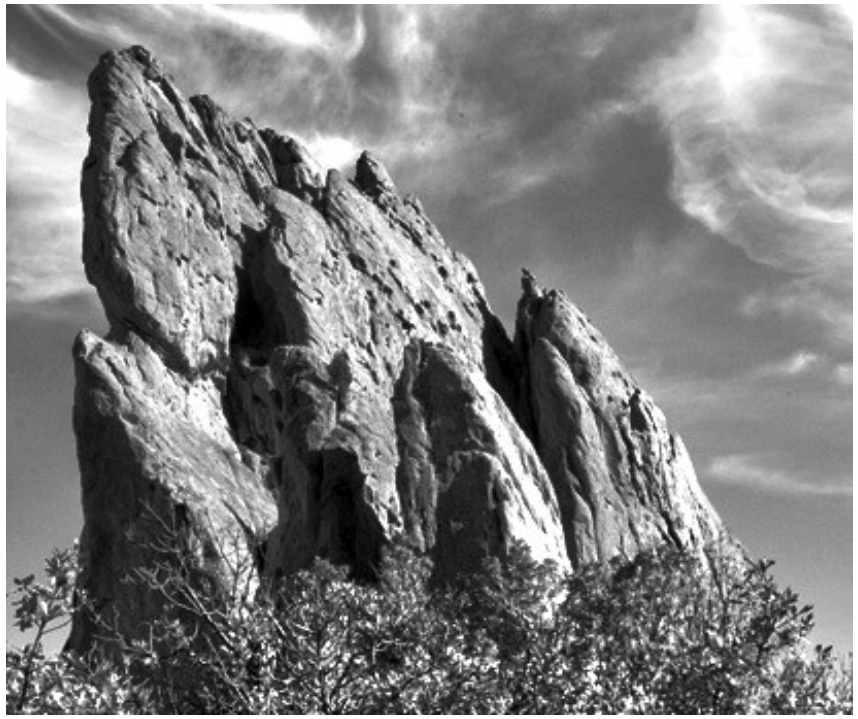
bilateral
filtering

median
filtering

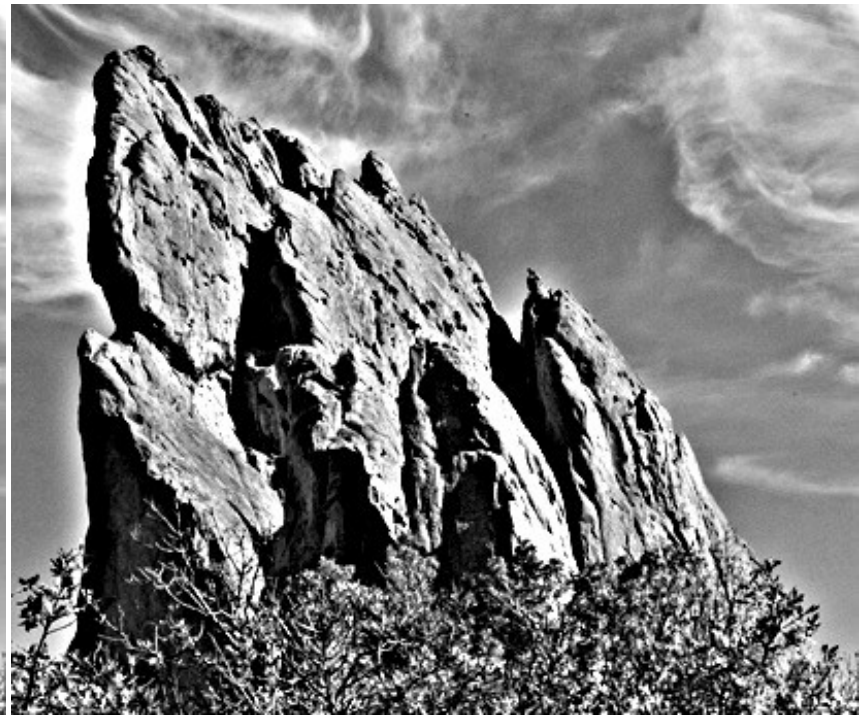
Contrast enhancement

How would you use Gaussian or bilateral filtering for sharpening?

input



sharpening based on bilateral filtering



sharpening based on Gaussian filtering

Photo retouching

Photo retouching

origina
l

digital pore removal (aka bilateral filtering)

Close-up comparison

origina
l

digital pore removal (aka bilateral filtering)

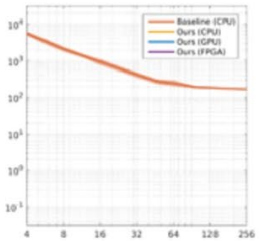
Cartoonization

input

cartoon rendition

The bilateral filter will reduce the color palette, or the numbers of colors that are used in the image.
Then we can apply edge detection to the resulting image to generate bold silhouettes.

Actively used in various research problems

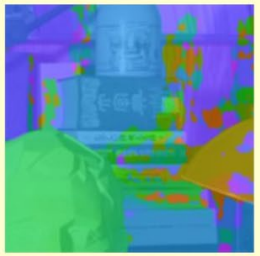


A Hardware-Friendly Bilateral Solver for Real-Time Virtual Reality Video
Amrita Mazumdar, Armin Alaghi, **Jonathan T. Barron**, David Gallup, Luis Ceze, Mark Oskin, Steven M. Seitz
High-Performance Graphics (HPG), 2017
project page

A reformulation of the bilateral solver can be implemented efficiently on GPUs and FPGAs.

Deep Bilateral Learning for Real-Time Image Enhancement
Michaël Gharbi, Jiawen Chen, **Jonathan T. Barron**, Samuel W. Hasinoff, Frédo Durand
SIGGRAPH, 2017
project page / video / bibtex / press

By training a deep network in bilateral space we can learn a model for high-resolution and real-time image enhancement.



The Fast Bilateral Solver
Jonathan T. Barron, Ben Poole
ECCV, 2016 **(Oral Presentation, Best Paper Honorable Mention)**
arXiv / bibtex / video (they messed up my slides, use →) / keynote (or PDF) / code / depth super-res results / reviews

Our solver smooths things better than other filters and faster than other optimization algorithms, and you can backprop through it.

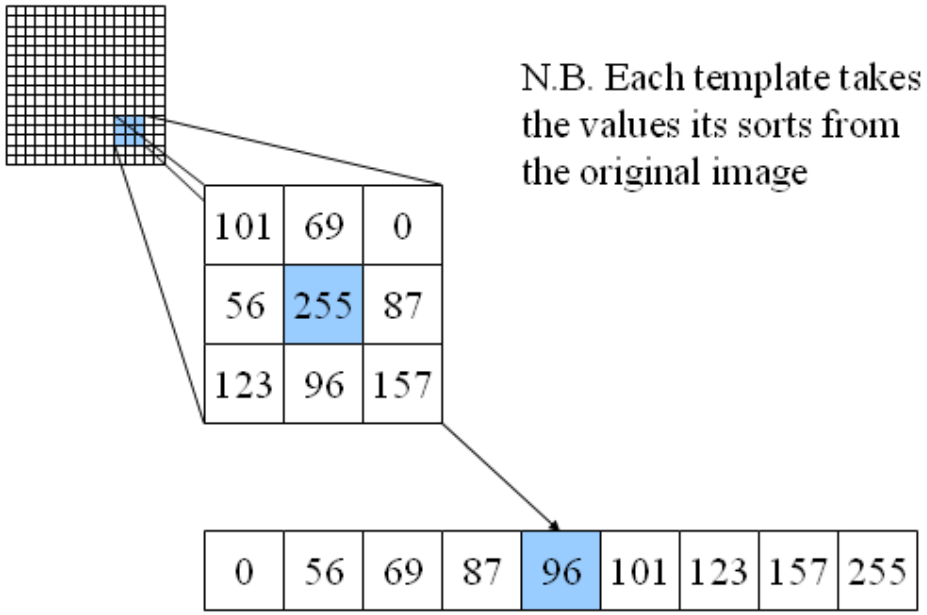
Filters: Thresholding

$$g(m,n) = \begin{cases} 255, & f(m,n) > A \\ 0 & otherwise \end{cases}$$

Non-Linear filters

- Is thresholding a linear filter?
- No is a non-linear filter together with image equalization or median filters

Median filter



Salt and pepper noise

Image Transformations & Filtering

- Point Processing
- Linear Filtering
- Sampling & Aliasing
- Image Derivatives
- Edge Detection

Image resizing

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

How to generate a half-sized version?

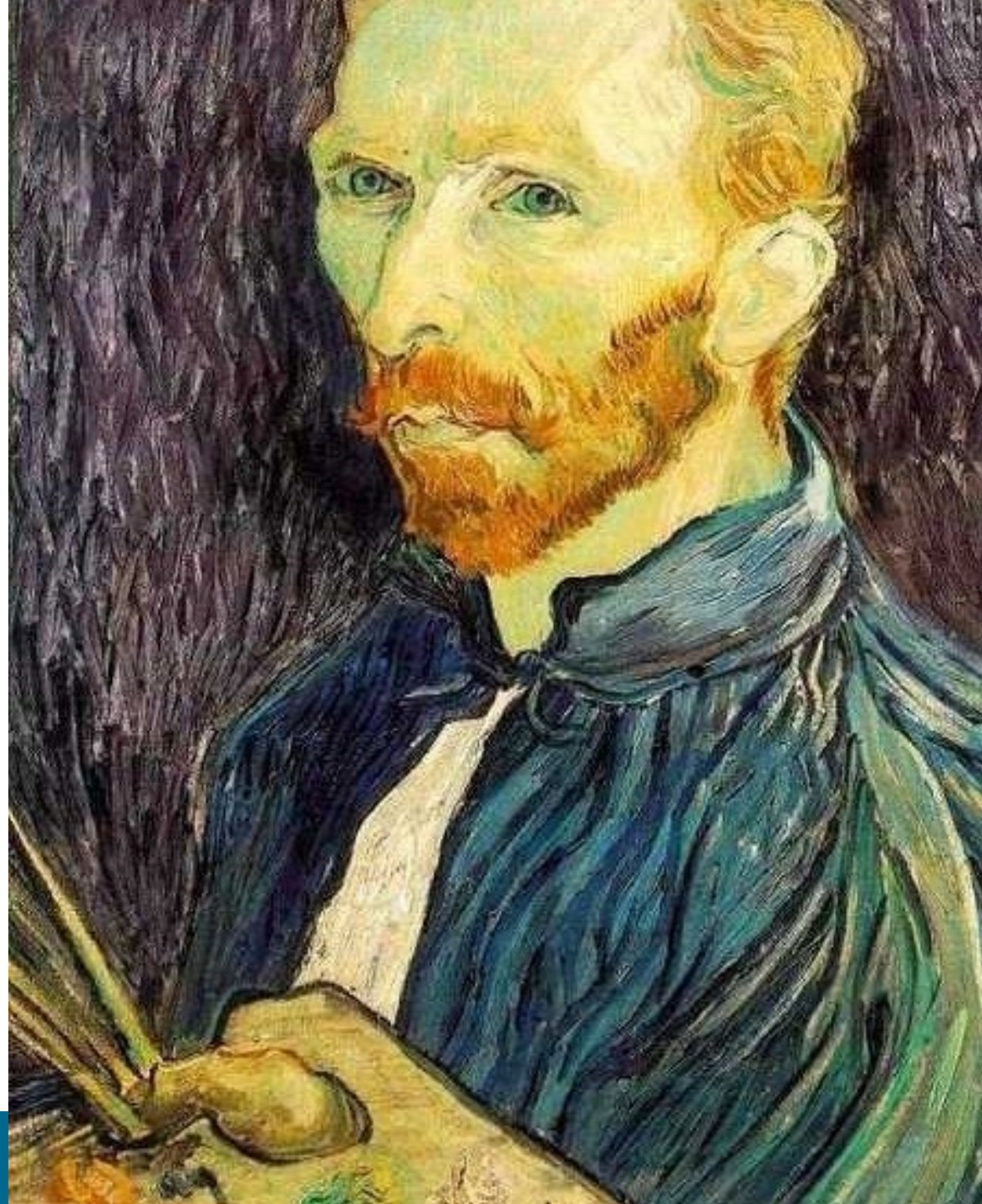
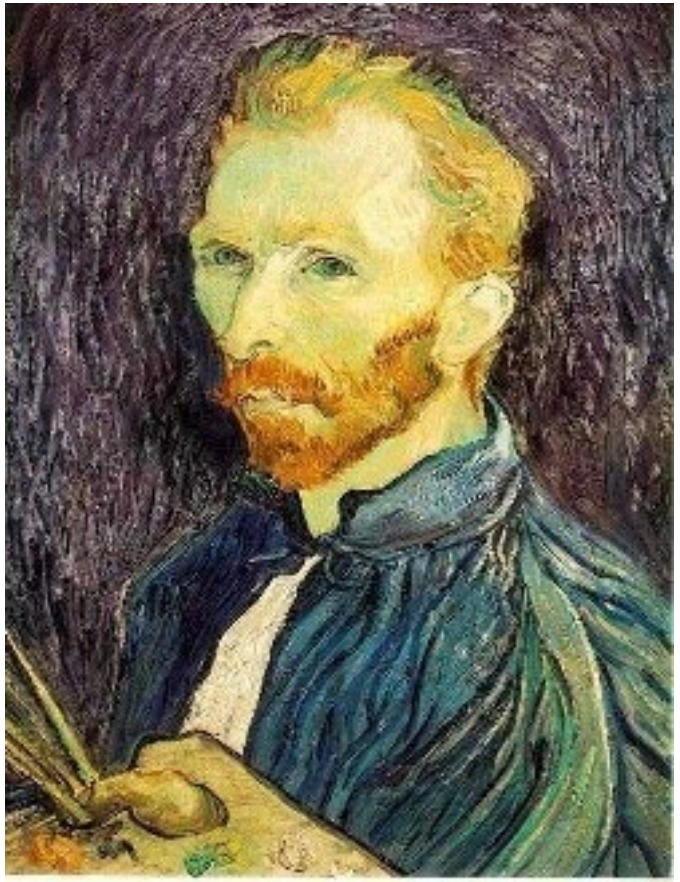


Image sub-sampling

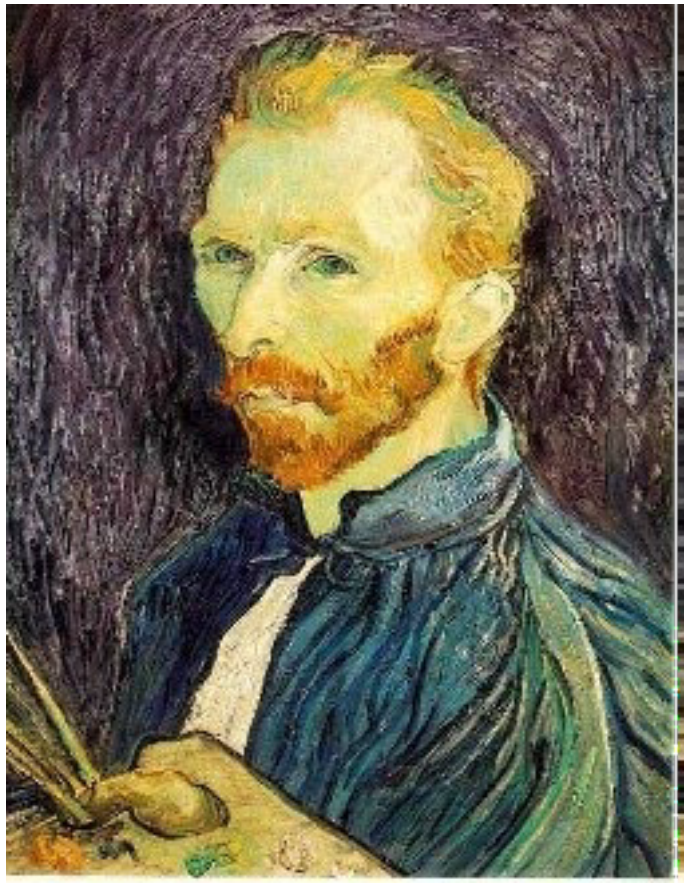


1/4

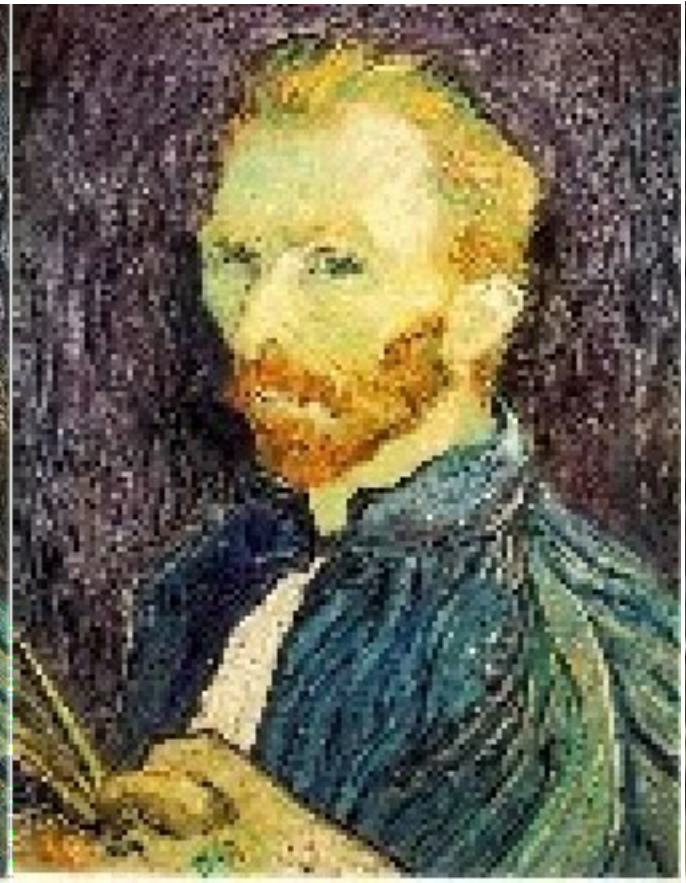
1/8

Throw away every other row and column to create a 1/2 size image

- called *image sub-sampling*



1/2



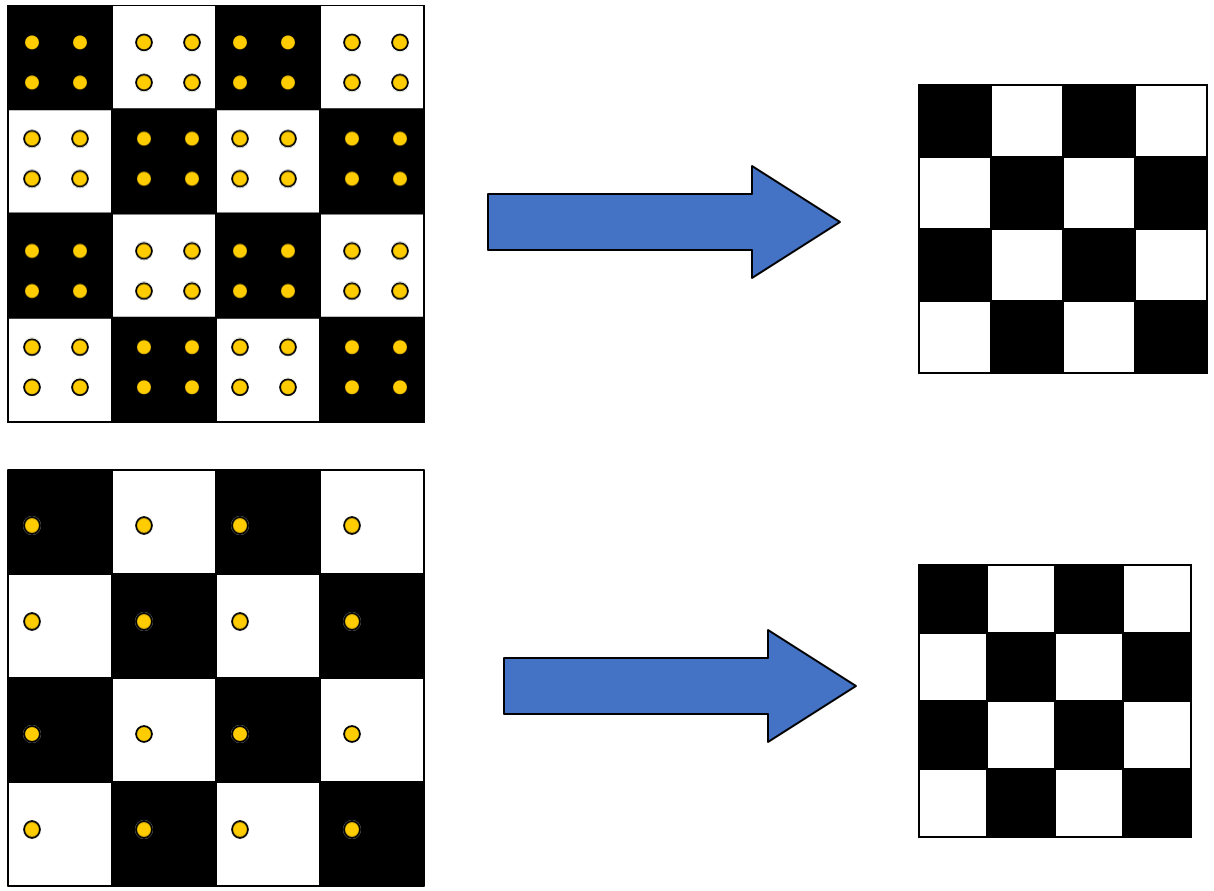
1/4 (2x zoom)

1/8 (4x zoom)

Aliasing!

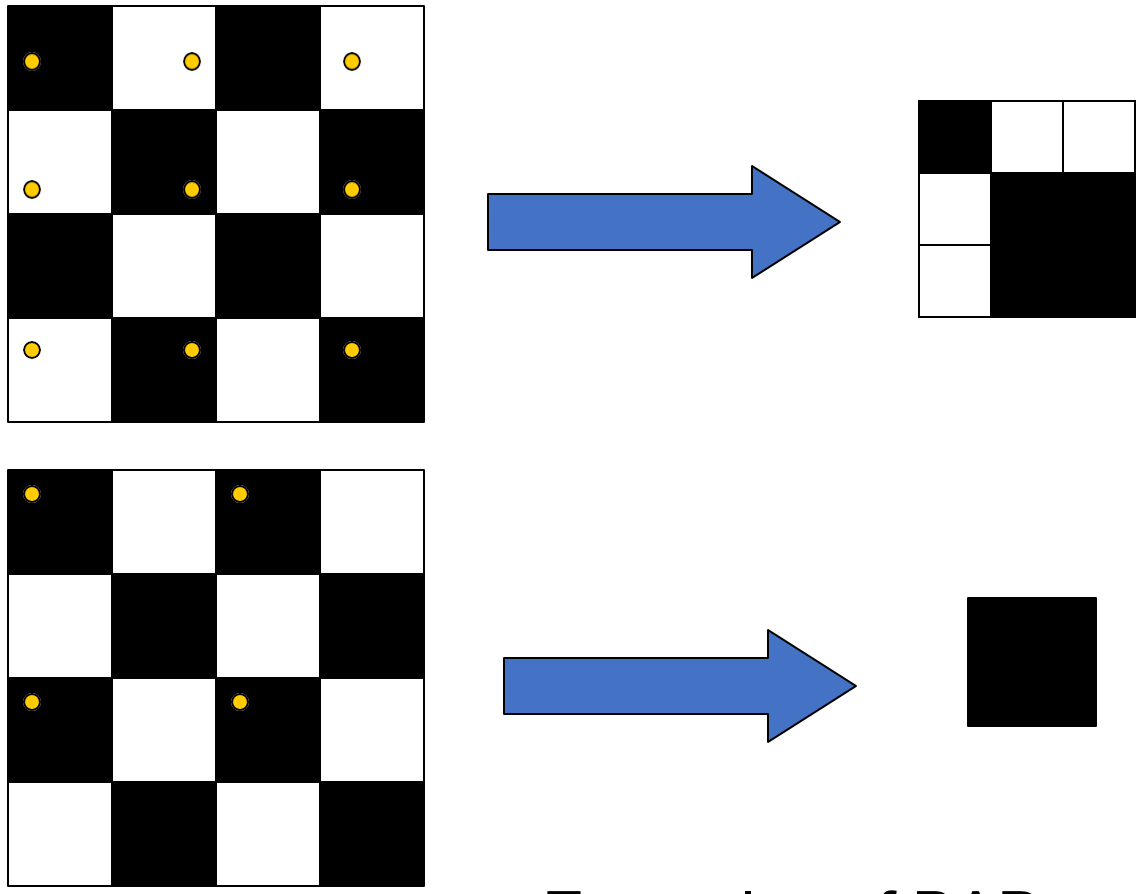
What do we do?

Sampling an image



Examples of GOOD sampling

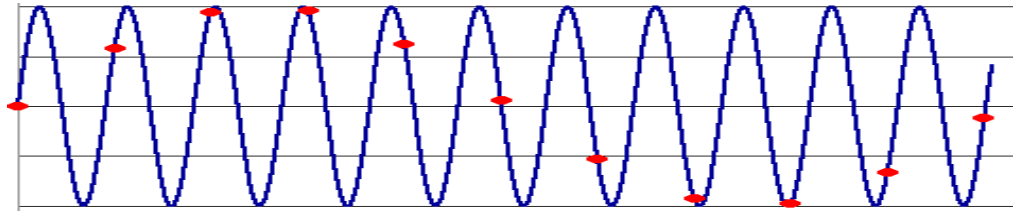
Undersampling



Examples of BAD sampling -> Aliasing

Aliasing

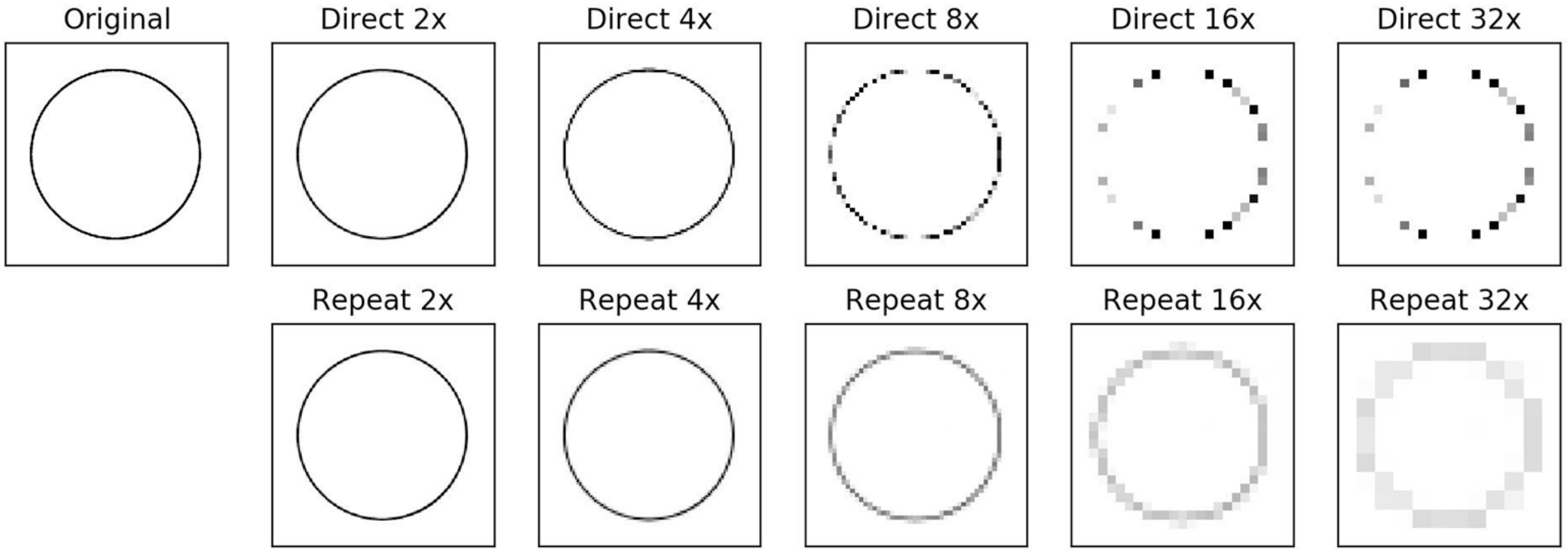
- Occurs when your sampling rate is not high enough to capture the amount of detail in your image
- Can give you the wrong signal/image—an *alias*
- To do sampling right, need to understand the structure of your signal/image
- Fourier Transform (we will talk about it in the next class)
- To avoid aliasing:
  - sampling rate ≥ 2 * max frequency in the image
    - said another way: ≥ two samples per cycle
  - this minimum sampling rate is called the **Nyquist rate**



Source: L. Zhang

A real problem!

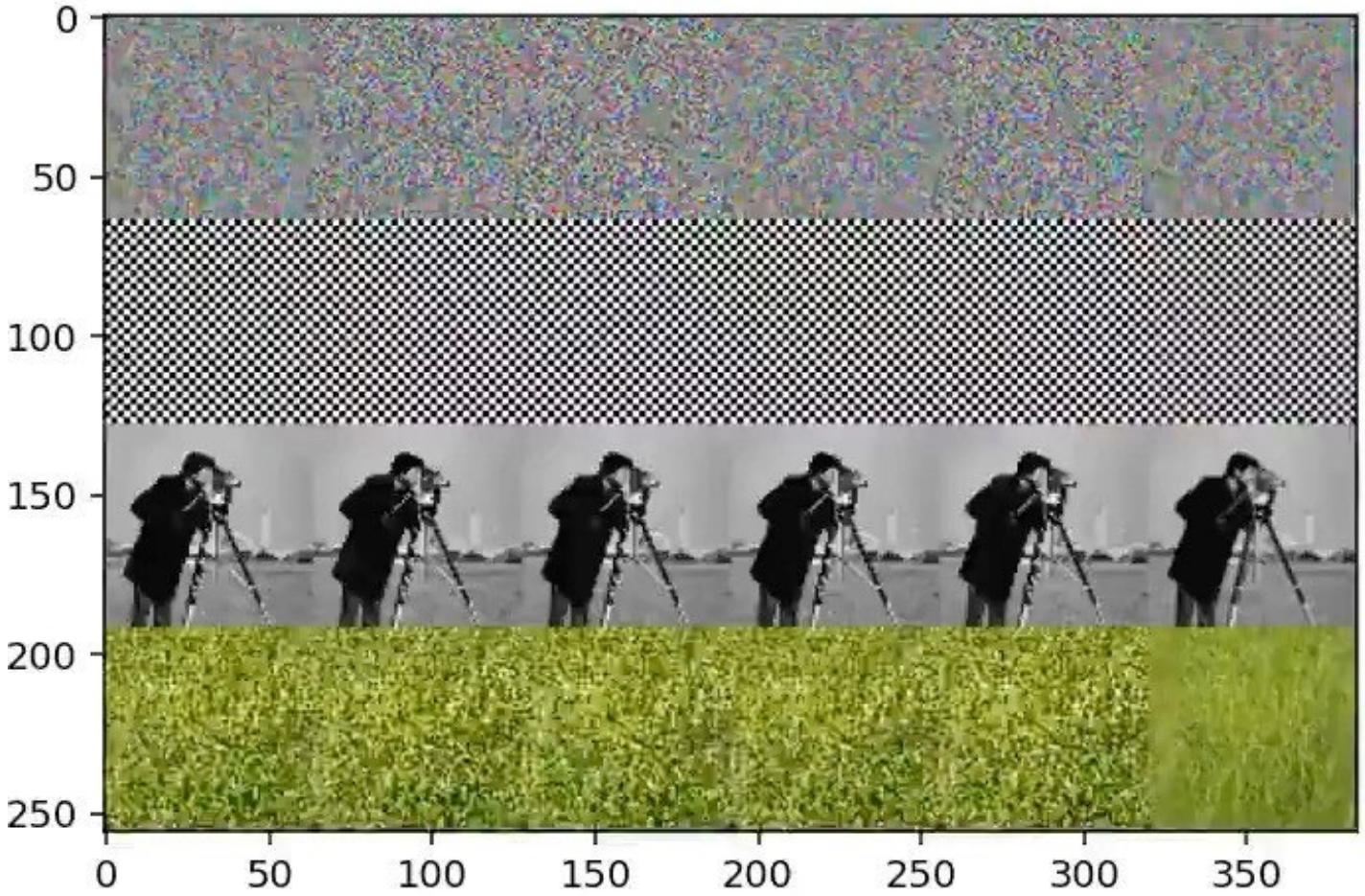
Bilinear downsampling with F.interpolate() in PyTorch



Credits: Richard Zhang

A real problem!

128 x 128 □64x64



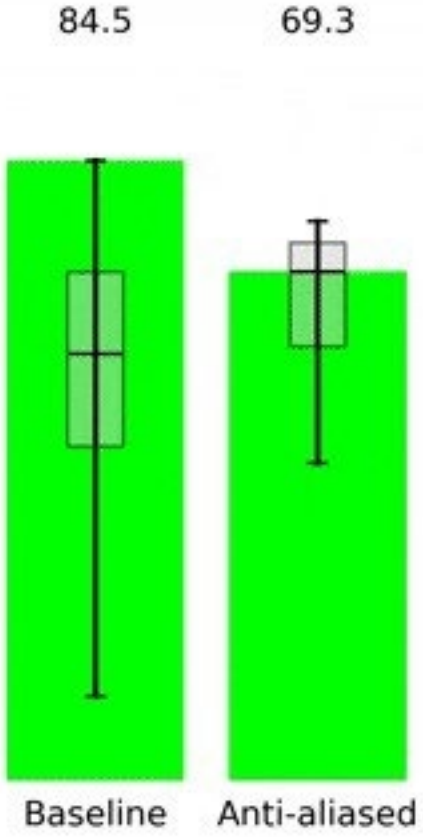
Open-CV:
default, bicubic, Lanczos4

Pytorch:
bilinear,
bicubic

PIL: Lanczos

credit: @jaakkolehtinen

Problems in NN too



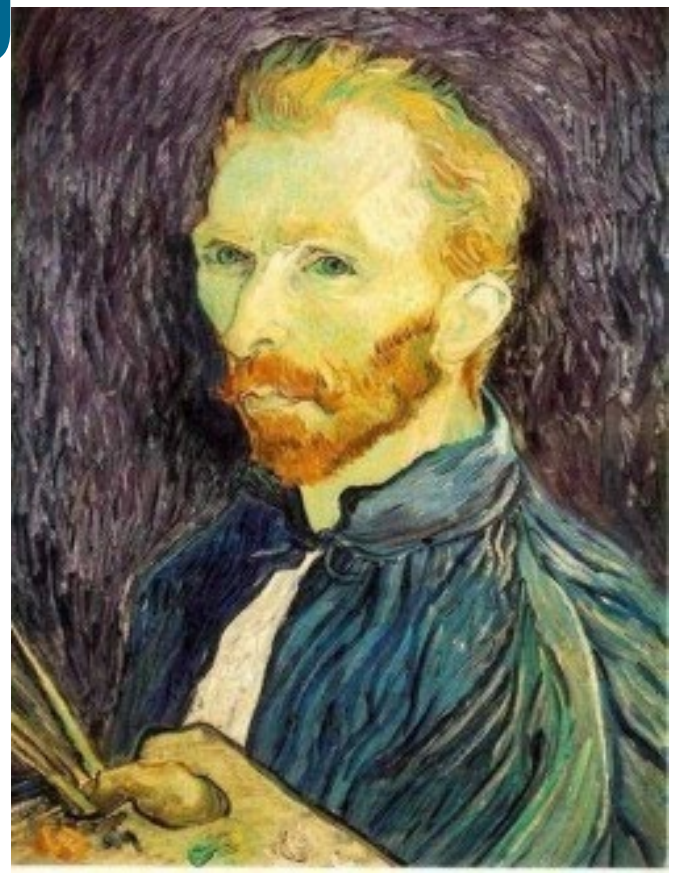
Classification stability: predicted probability of the correct class changes when shifting the image. Anti-aliased max-pooling. Pooling does not preserve shift-equivariance

Anti-aliasing in CNNs

```
pip install antialiased-cnns
```

Making Convolutional Networks Shift-Invariant Again, Richard Zhang ICML 2019

Gaussian (lowpass) pre-filtering



Gaussian 1/2

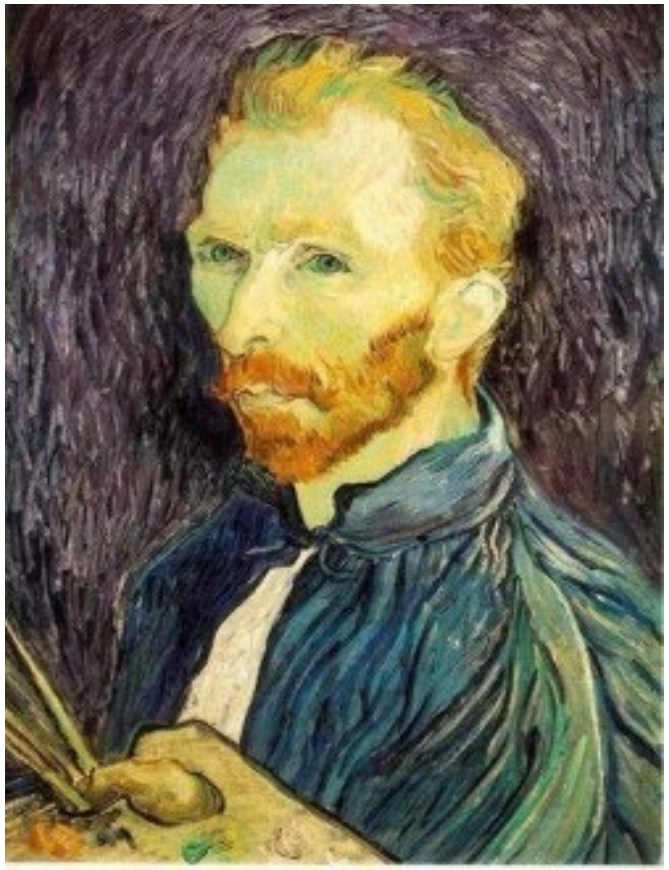
G 1/4

G 1/8

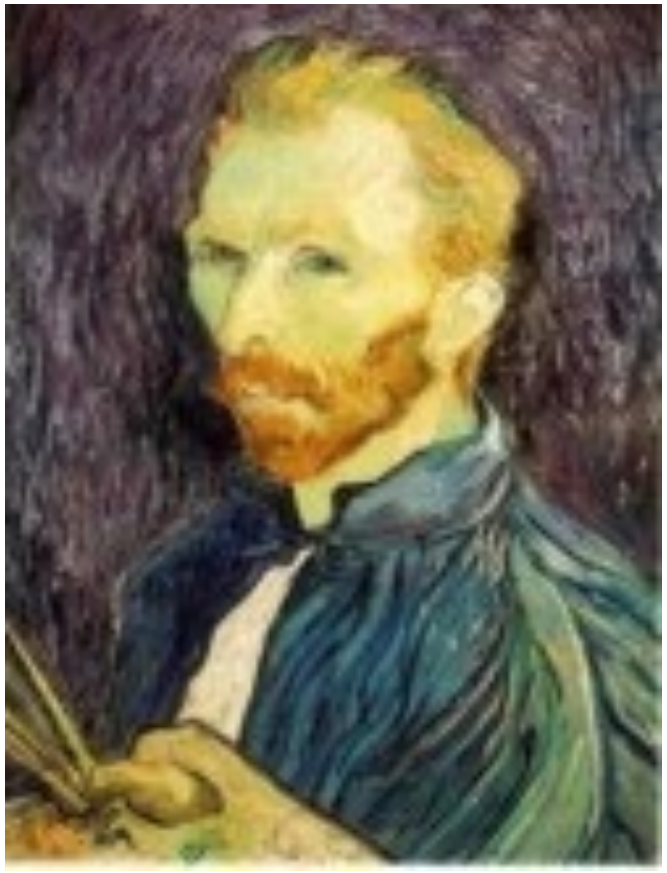
Solution: filter the image, *then* subsample

- Filter size should double for each ½ size reduction.

Subsampling with Gaussian pre-filtering



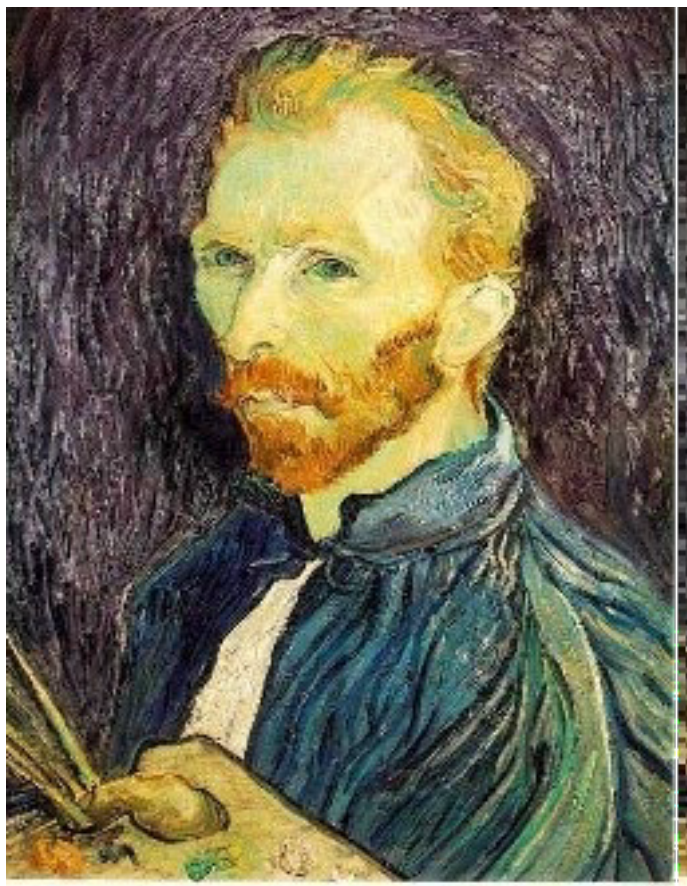
Gaussian 1/2



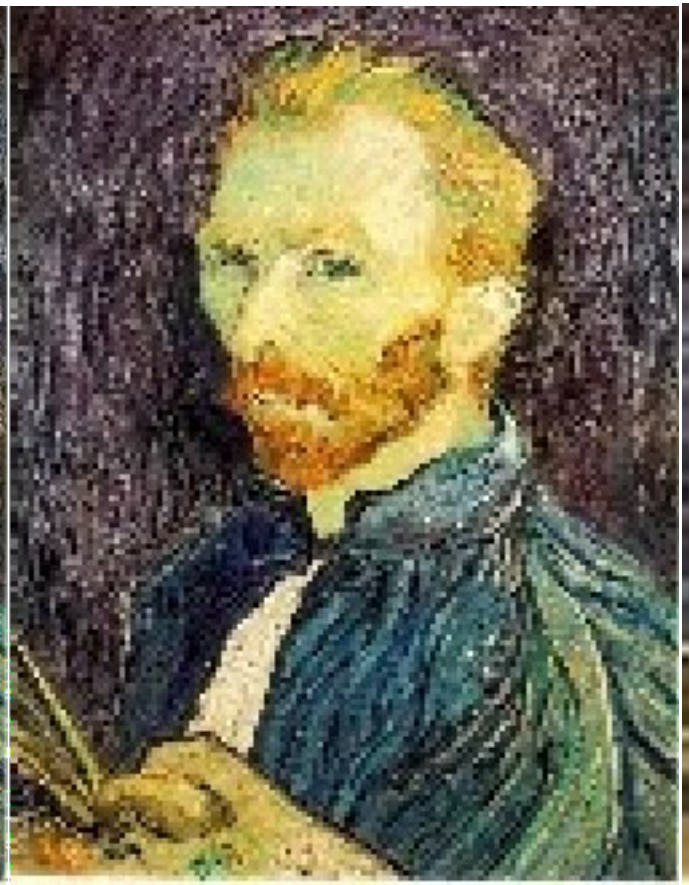
G ¼ (2x zoom)

G 1/8 (4x zoom)

Compare with...



1/2



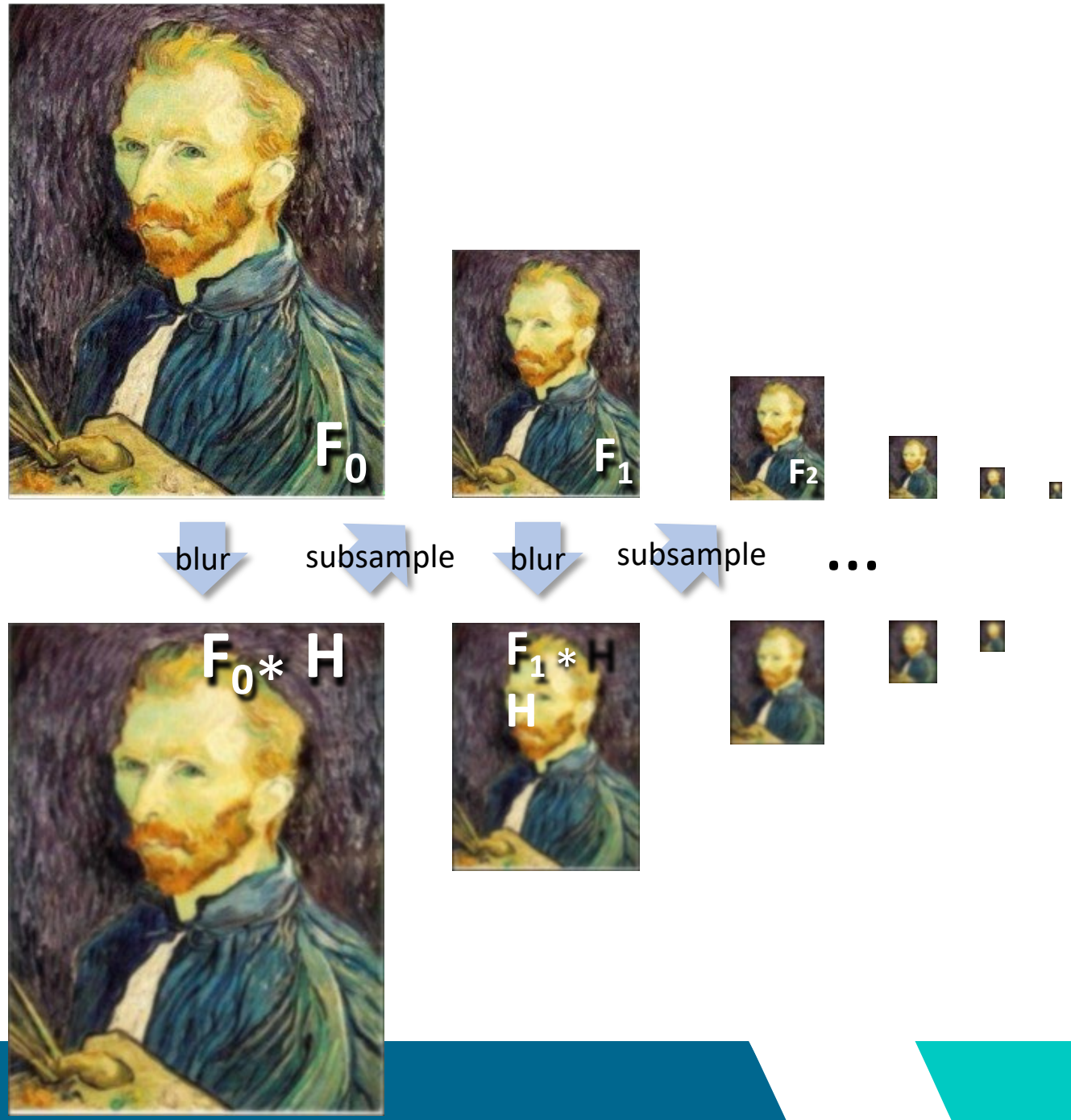
1/4 (2x zoom)

1/8 (4x zoom)

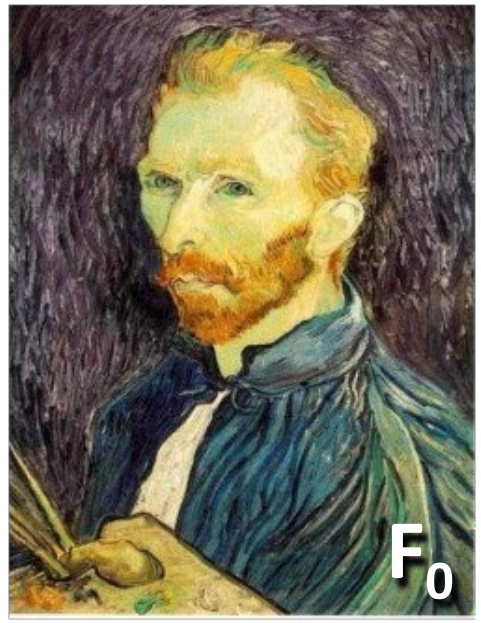
Slide by Steve Seitz

Gaussian pre-filtering

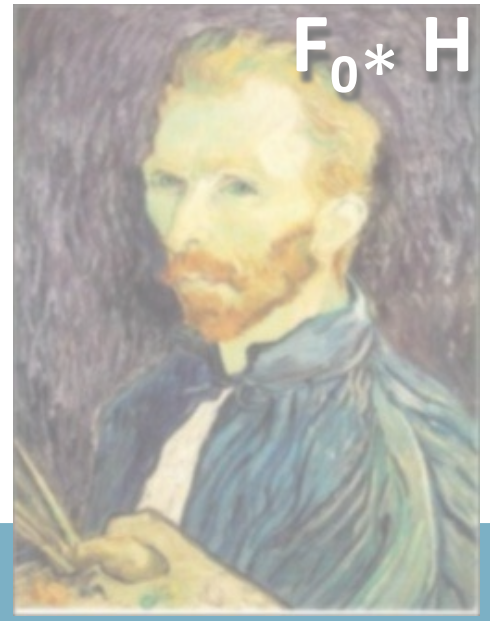
- Solution: filter the image, *then* subsample

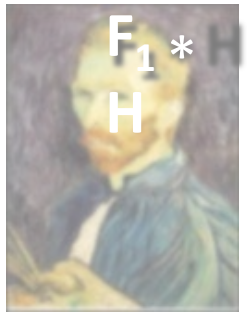


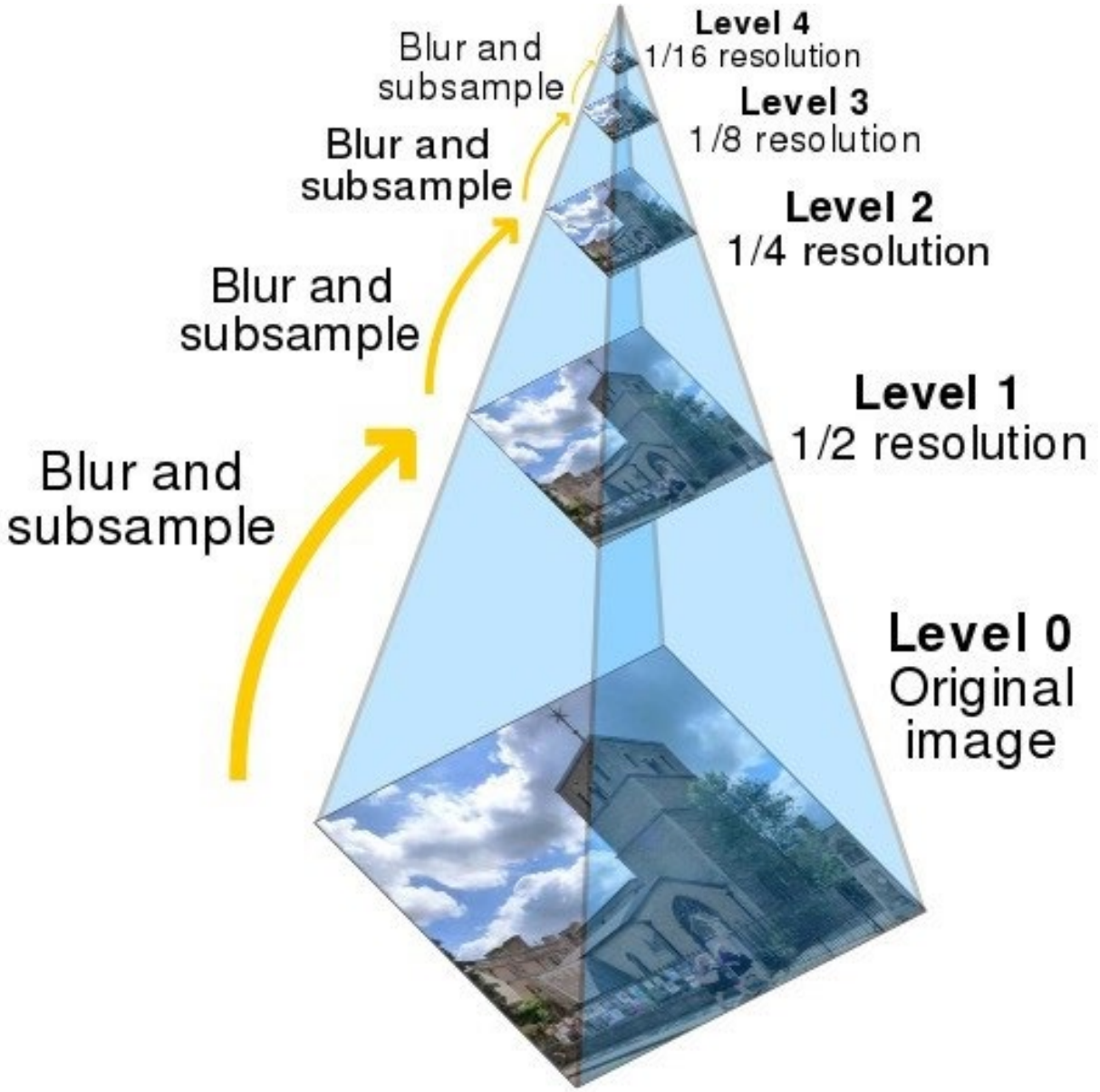
Gaussian pyramid



blur subsample blur subsample • • •





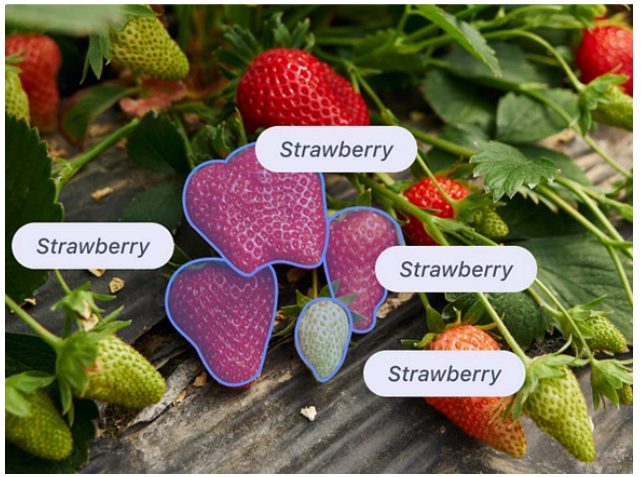


Similar to Gaussian Pyramids, there are Laplacian Pyramids.

- We will see them in the following class.

What are they good for?

- Improve Search: search over scale
  - Template matching
  - Classic coarse-to-fine strategy
    - ✓ E.g. find an object at different scales



Upsampling

- This image is too small for this screen: 
- How can we make it 10 times as big?
- Simplest approach: repeat each row and column 10 times (“nearest neighbor interpolation”)

Image interpolation

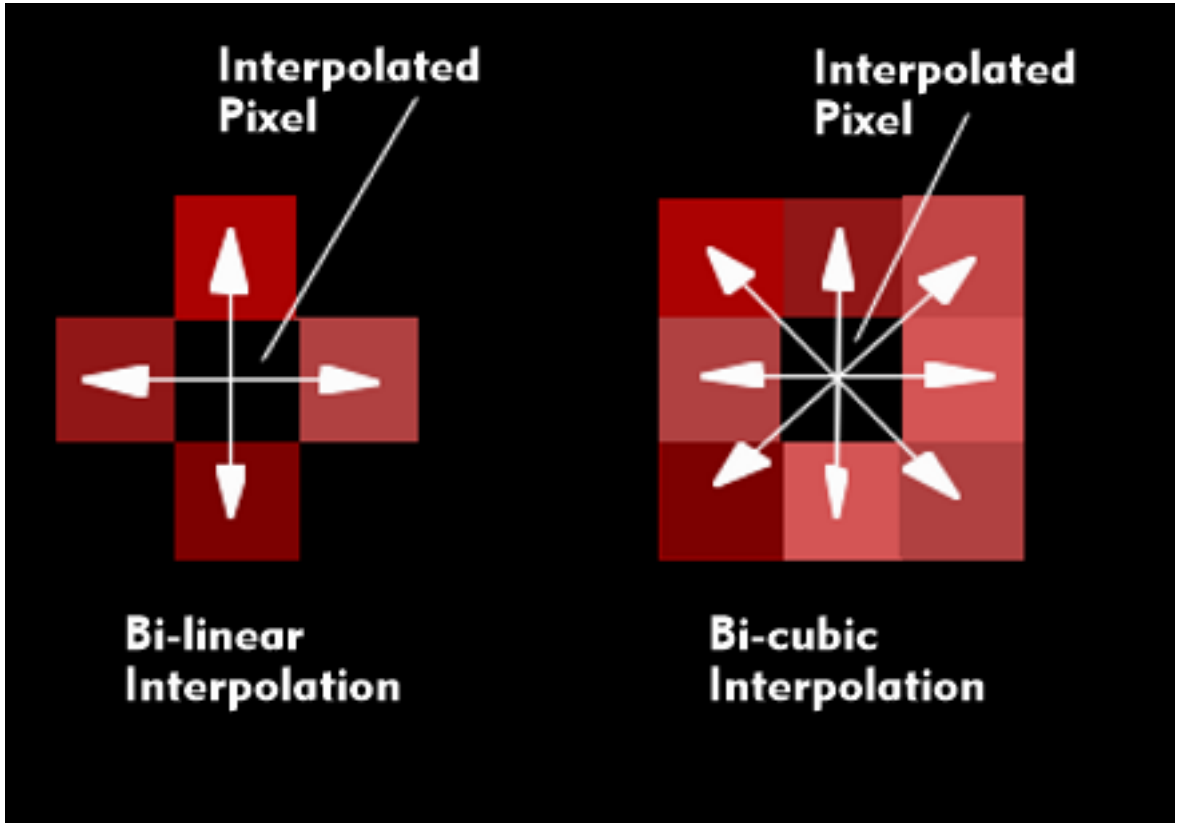
Original image:  x 10

Nearest-neighbor interpolation

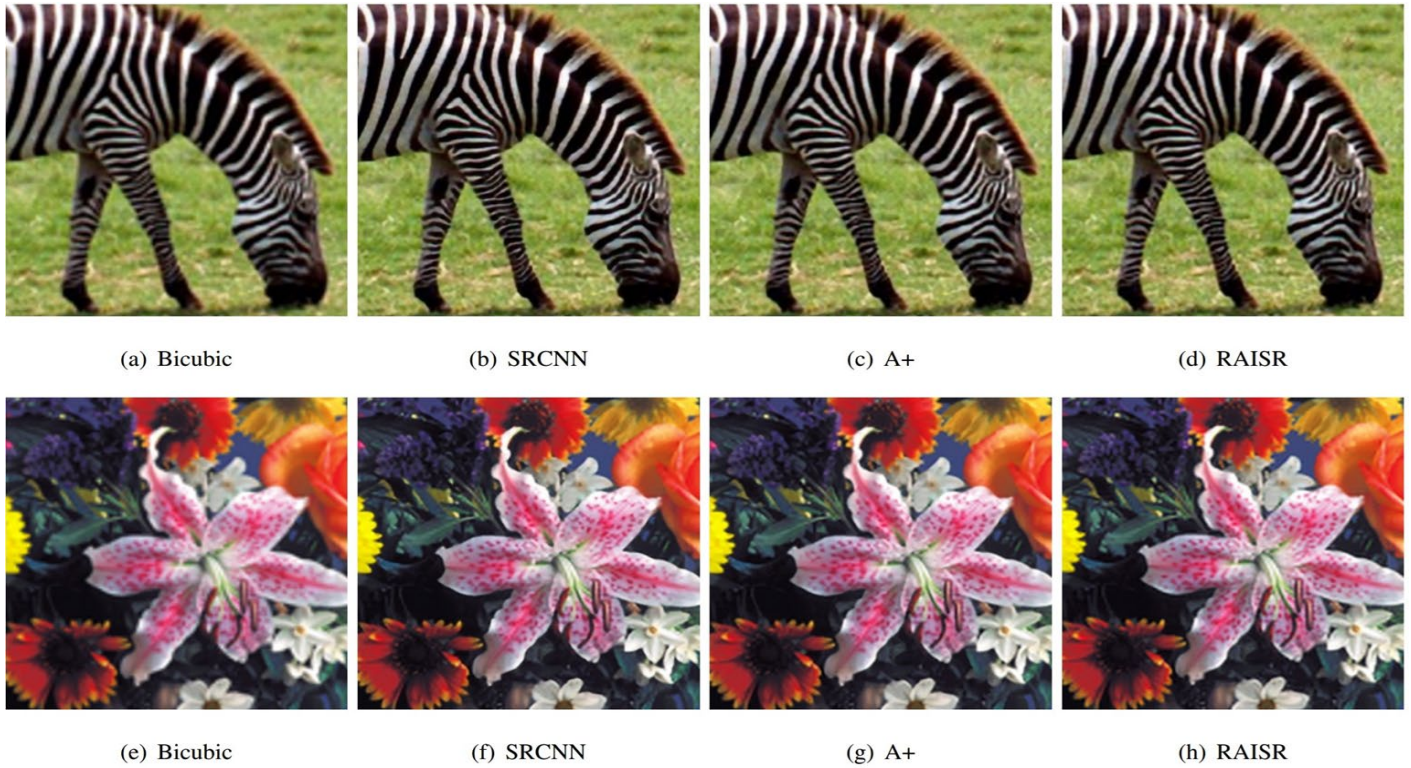
Bilinear interpolation

Bicubic interpolation

Bilinear and Bicubic interpolation



Modern methods



(a) Bicubic (b) SRCNN (c) A+ (d) RAISR

(e) Bicubic (f) SRCNN (g) A+ (h) RAISR

Romano, et al: RAISR: Rapid and Accurate Image Super Resolution 2016, IEEE Transaction on Computational Image

R. Timofte, V. De Smet, and L. Van Gool. A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution. In Asian Conference on Computer Vision (ACCV 2014), 1-5 November 2014

Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang. Learning a Deep Convolutional Network for Image Super-Resolution, in Proceedings of European Conference on Computer Vision

Super-resolution with multiple images

- Can do better upsampling if you have multiple images of the scene taken with small (subpixel) shifts
- Some cellphone cameras (like the Google Pixel line) capture a **burst** of photos
- Can we use that burst for upsampling?

Google Pixel 3 Super Res Zoom

Super-resolution Using Hand Motion

Effect of hand tremor as seen in a cropped burst of photos, after global alignment

Example photo with and without super res zoom (smart burst align and merge)

https://ai.googleblog.com/2018/10/see-better-and-further-with-super-res.html

Image Transformations & Filtering

- Point Processing
- Linear Filtering
- Sampling & Aliasing
- Image Derivatives
- Edge Detection

Partial derivatives with convolution

Image is a function f(x,y)

Remember:

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

Approximate:

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

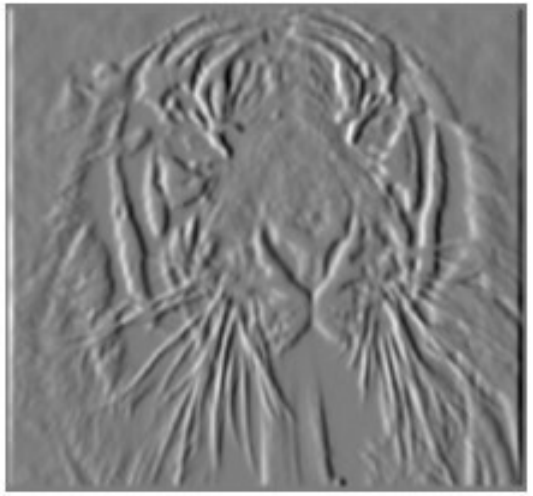
-1	1

Another one:

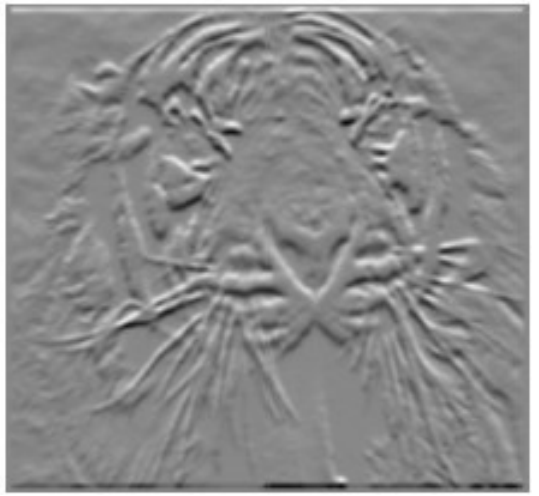
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{1}{2}\left(f(x+1, y) - f(x-1, y)\right)$$

-1	0	1

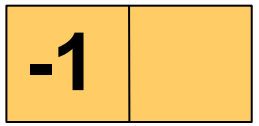
Image Gradient

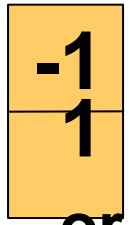


$$\frac{\partial f(x,y)}{\partial x}$$

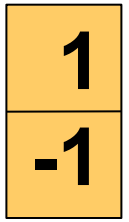


$$\frac{\partial f(x,y)}{\partial y}$$





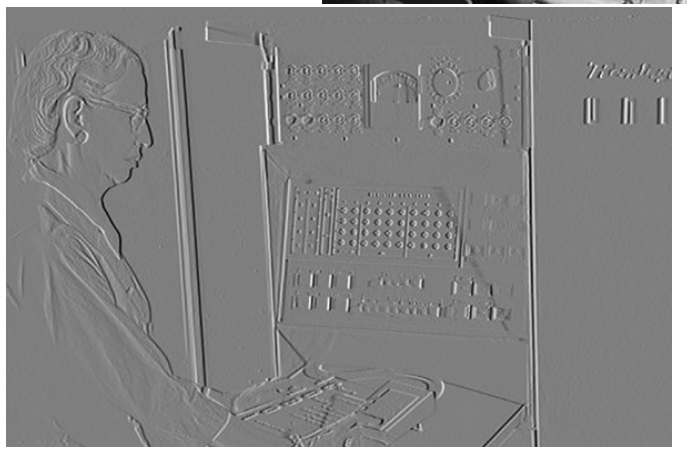
or

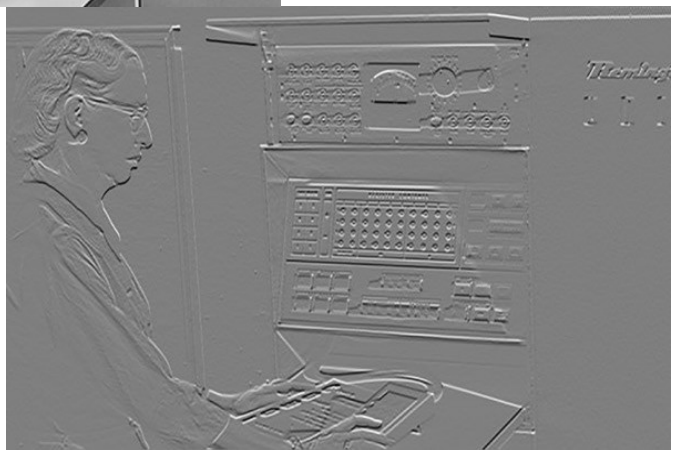


$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Partial Derivatives

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





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

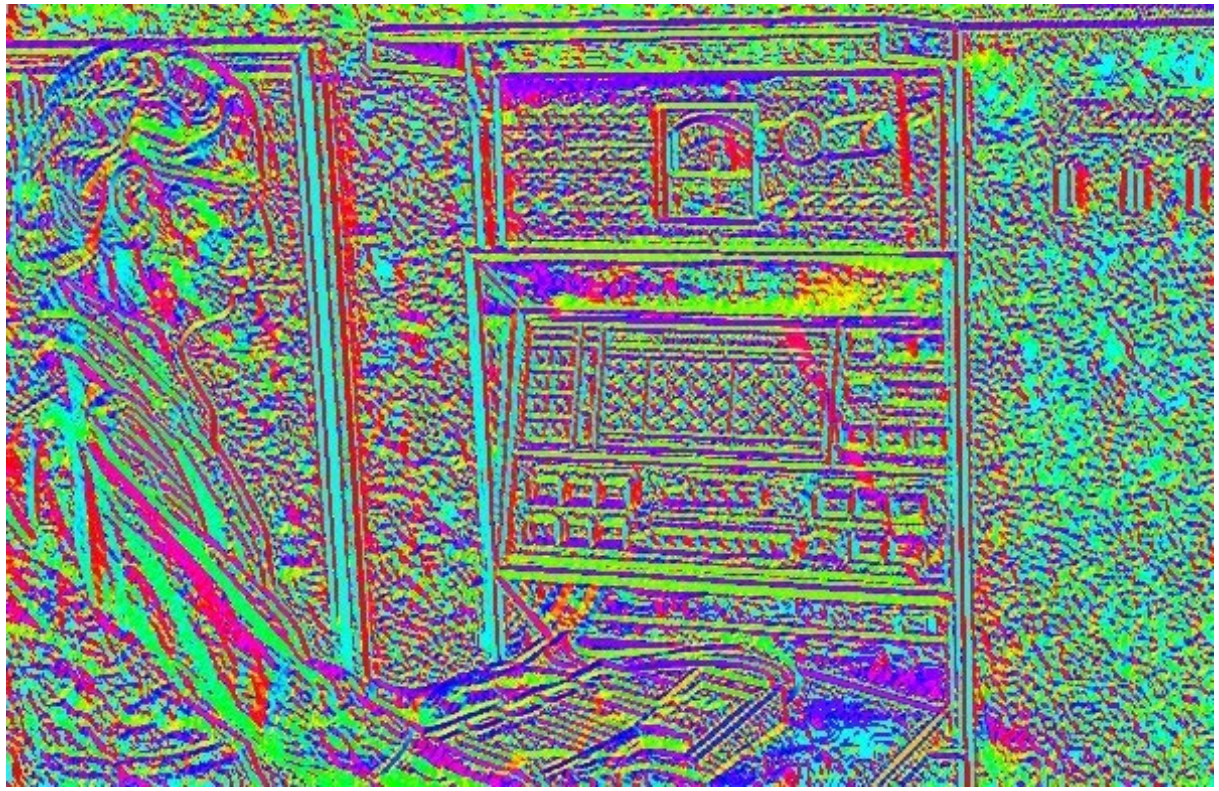
Gradient magnitude

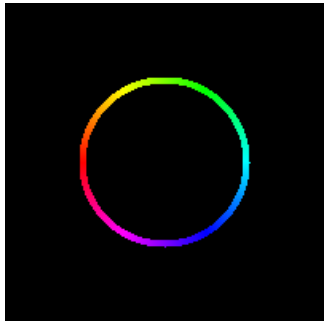
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Gradient Orientation

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

all the gradients



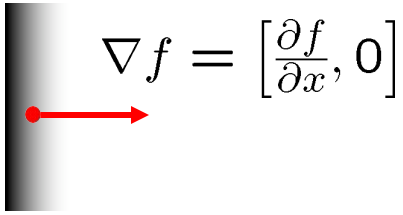


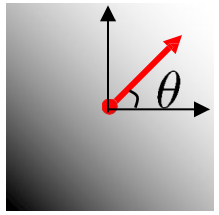
Source: D. Fouhey

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?

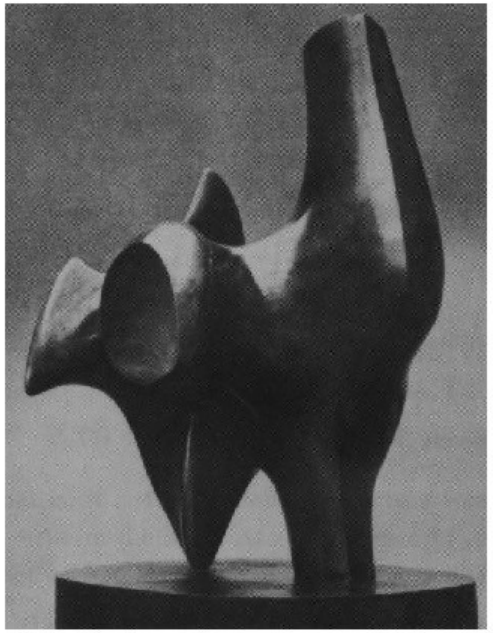
The *edge strength* is given by the gradient magnitude

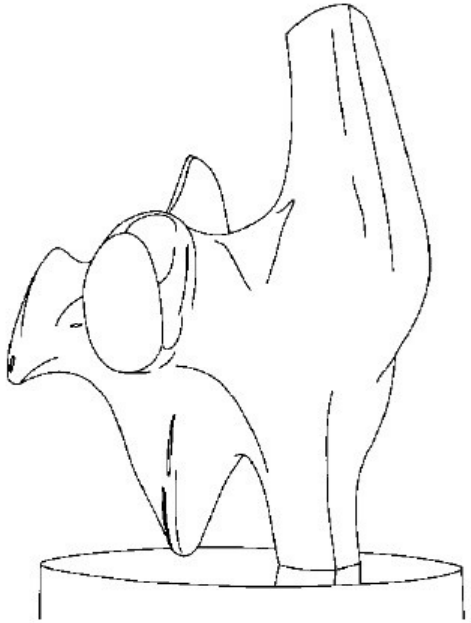
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

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

Source: Steve Seitz

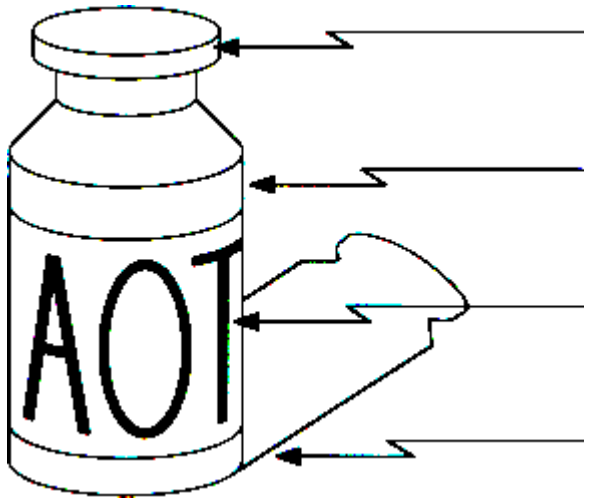
Edge detection





- Convert a 2D image into a set of curves
  - Extracts salient features of the scene
  - More compact than pixels

Origin of edges



surface normal discontinuity

depth discontinuity

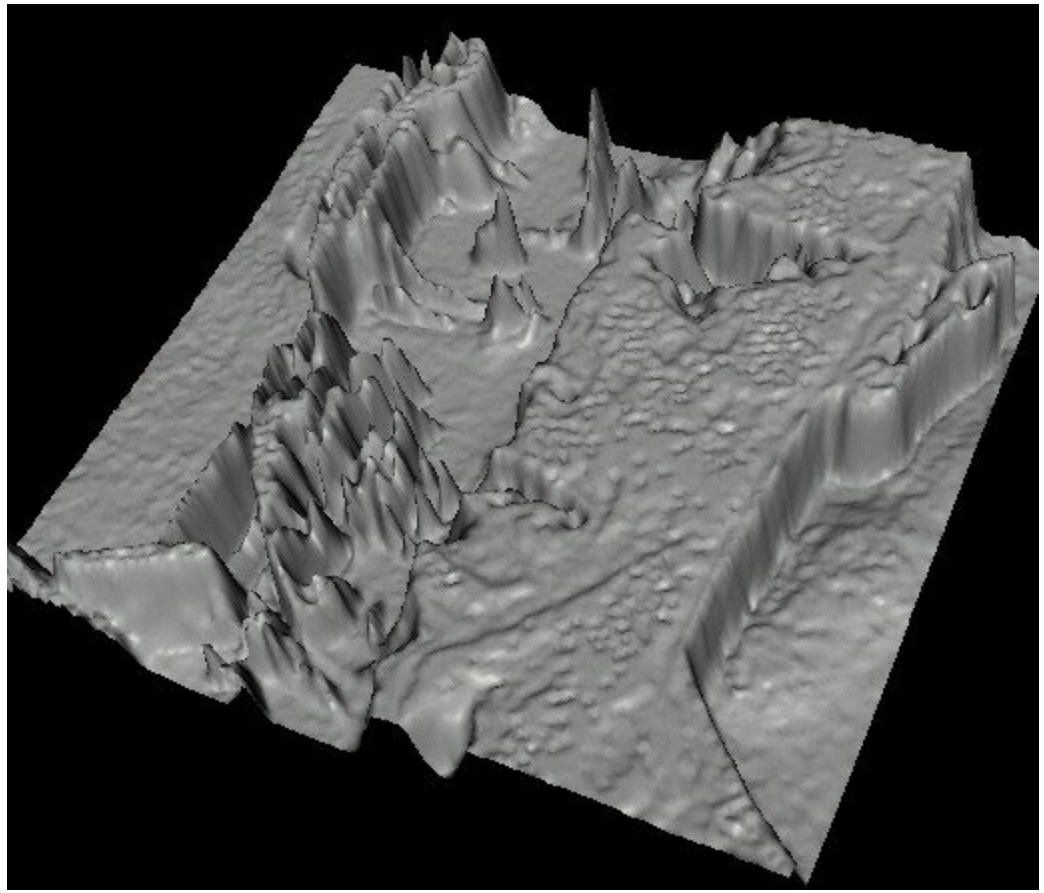
surface color

discontinuity illumination

discontinuity

- Edges are caused by a variety of factors

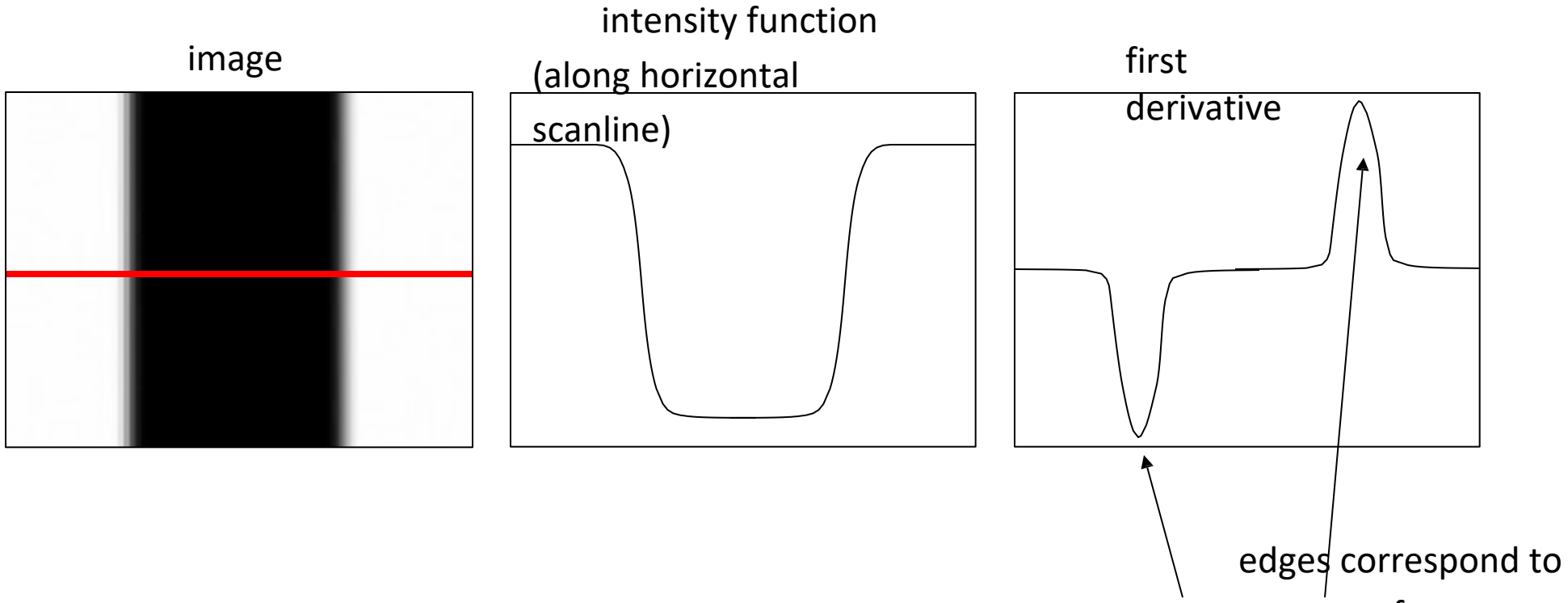
Images as functions…



- Edges look like steep cliffs

Characterizing edges

- An edge is a place of *rapid change* in the image intensity function



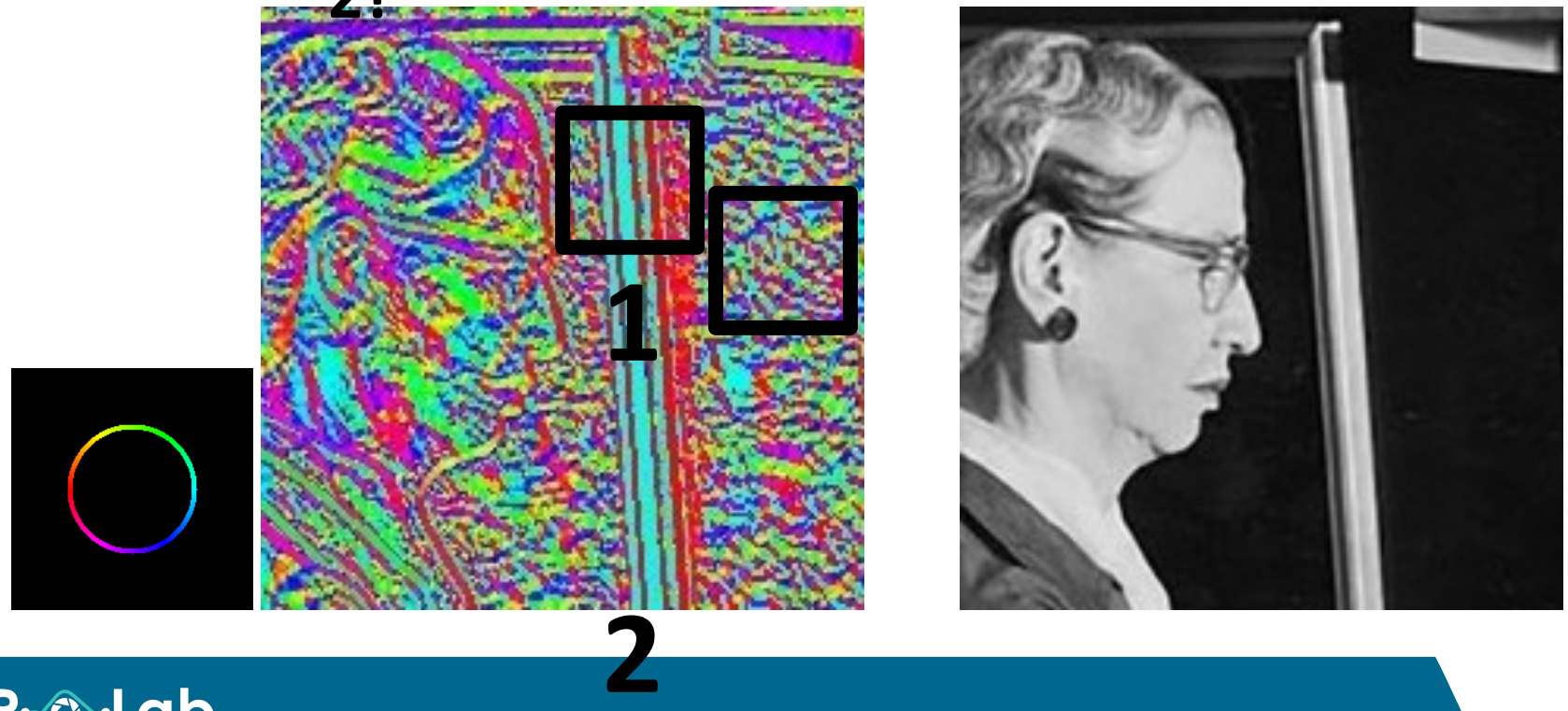
Source: L. Lazebnik

Image Gradient & Edges

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

Direction of image gradients

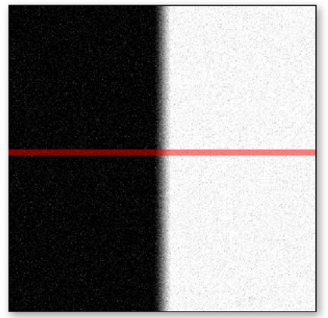
Why is there structure at 1 and not at 2?



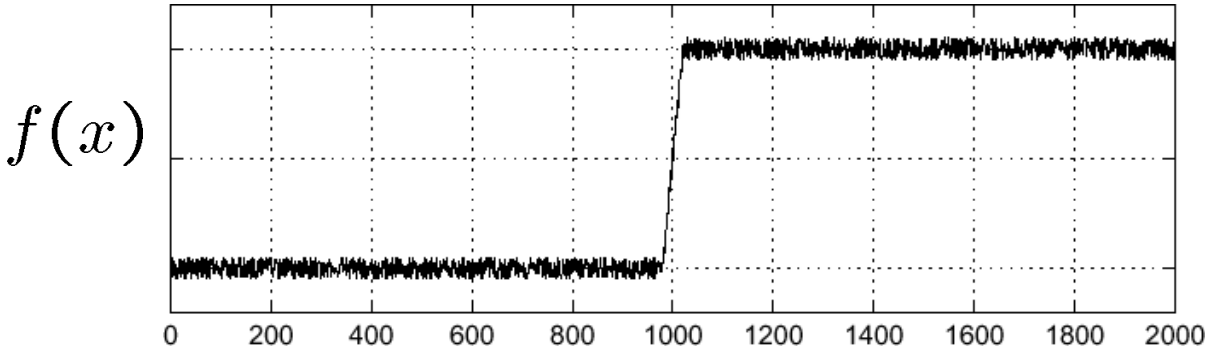
Source: D. Fouhey

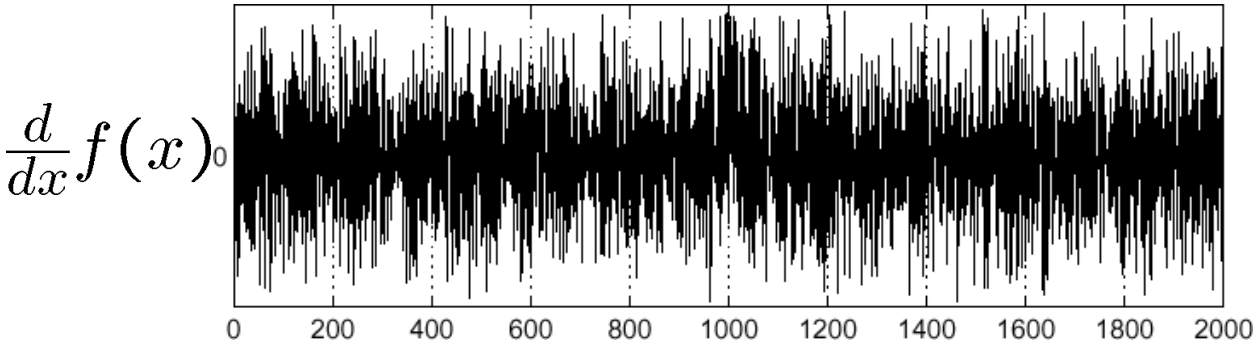
Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



Noisy input image





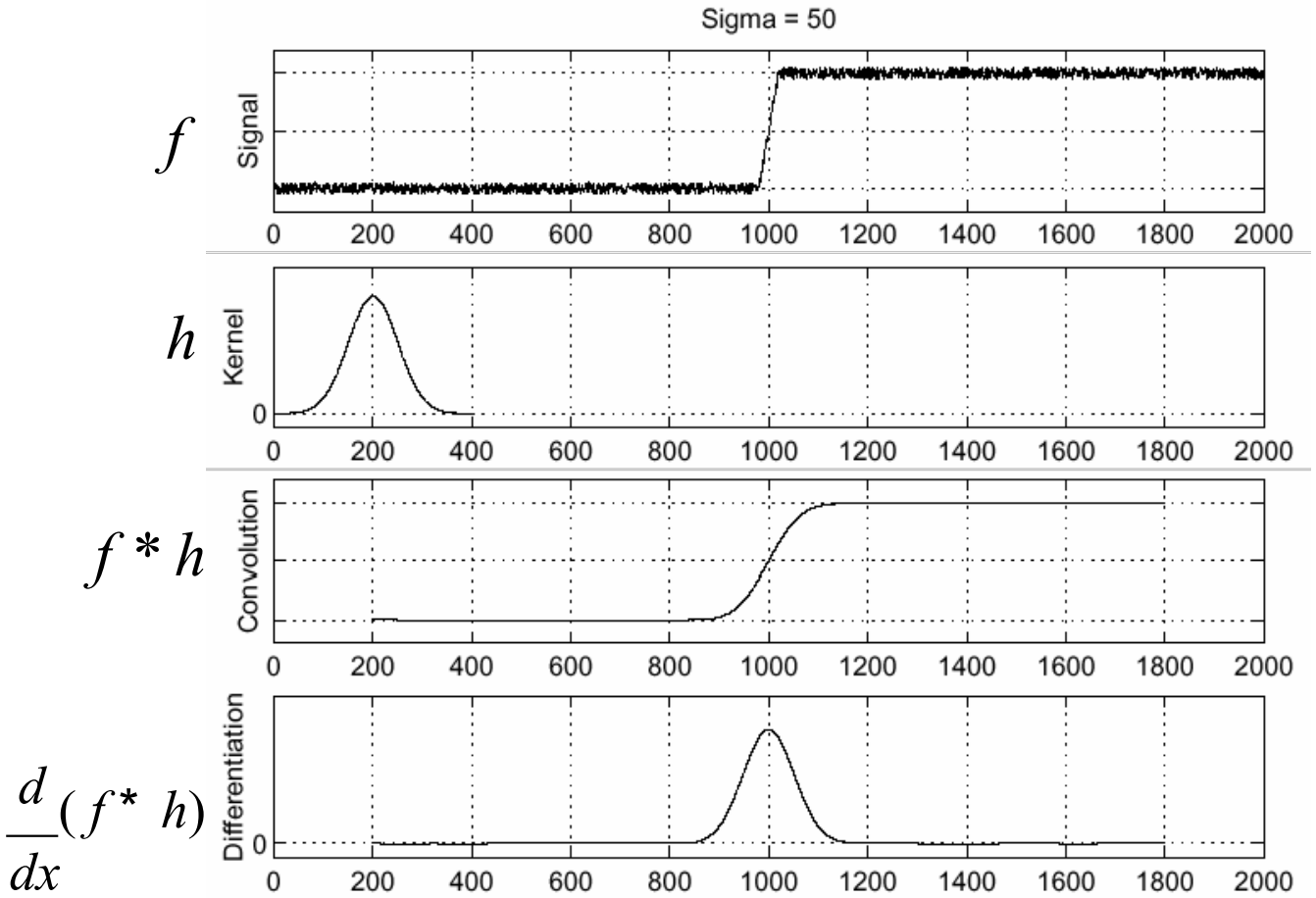
Where is the edge?

Issues with derivate filters:

- Sensitive to edges at small spatial scales
- Also sensitive to noise

Source: S. Seitz

Solution: smooth first

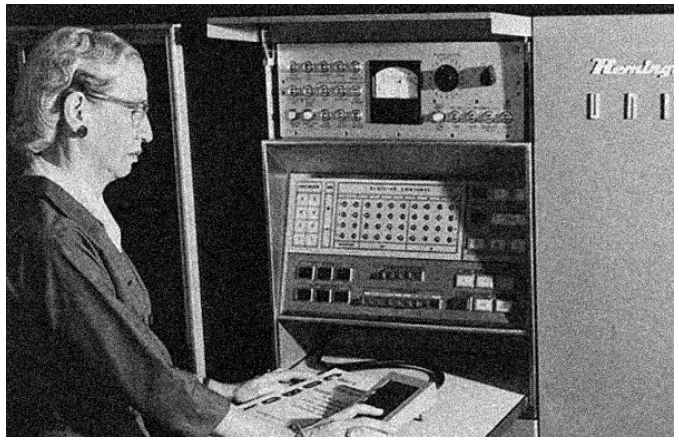


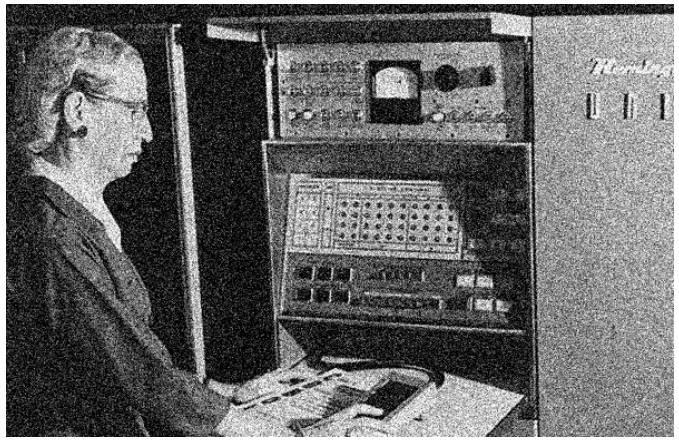
- To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

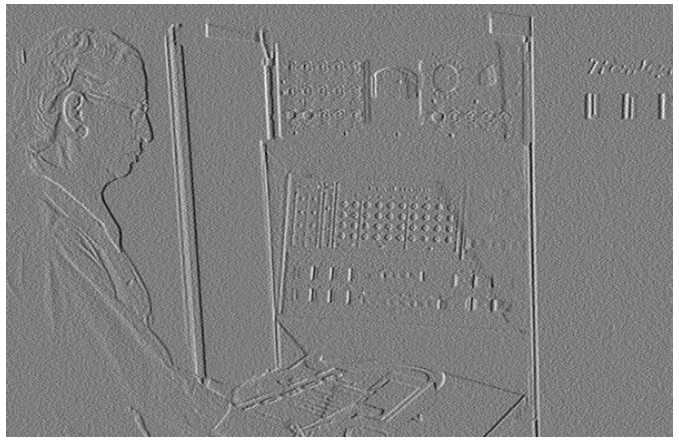
Noise in 2D

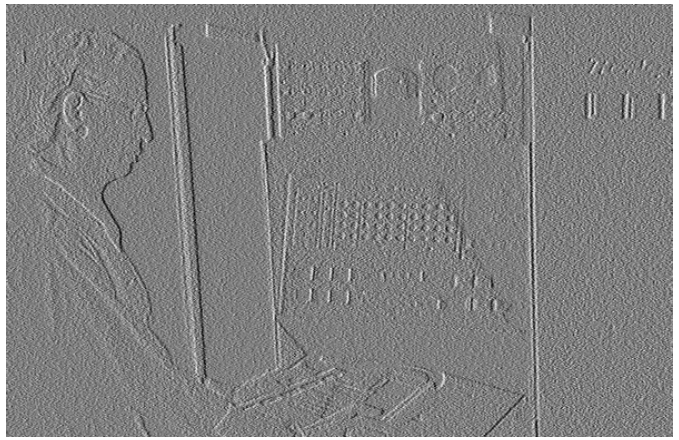
Noisy Input



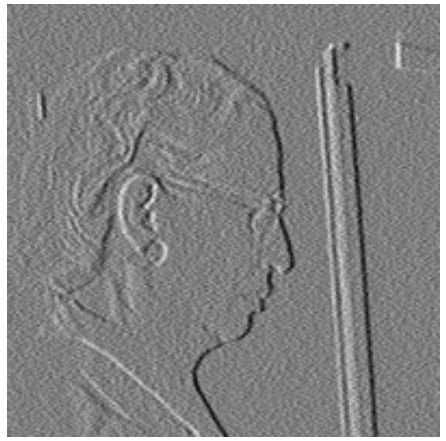


Ix via [-1,0,1]





Zoom



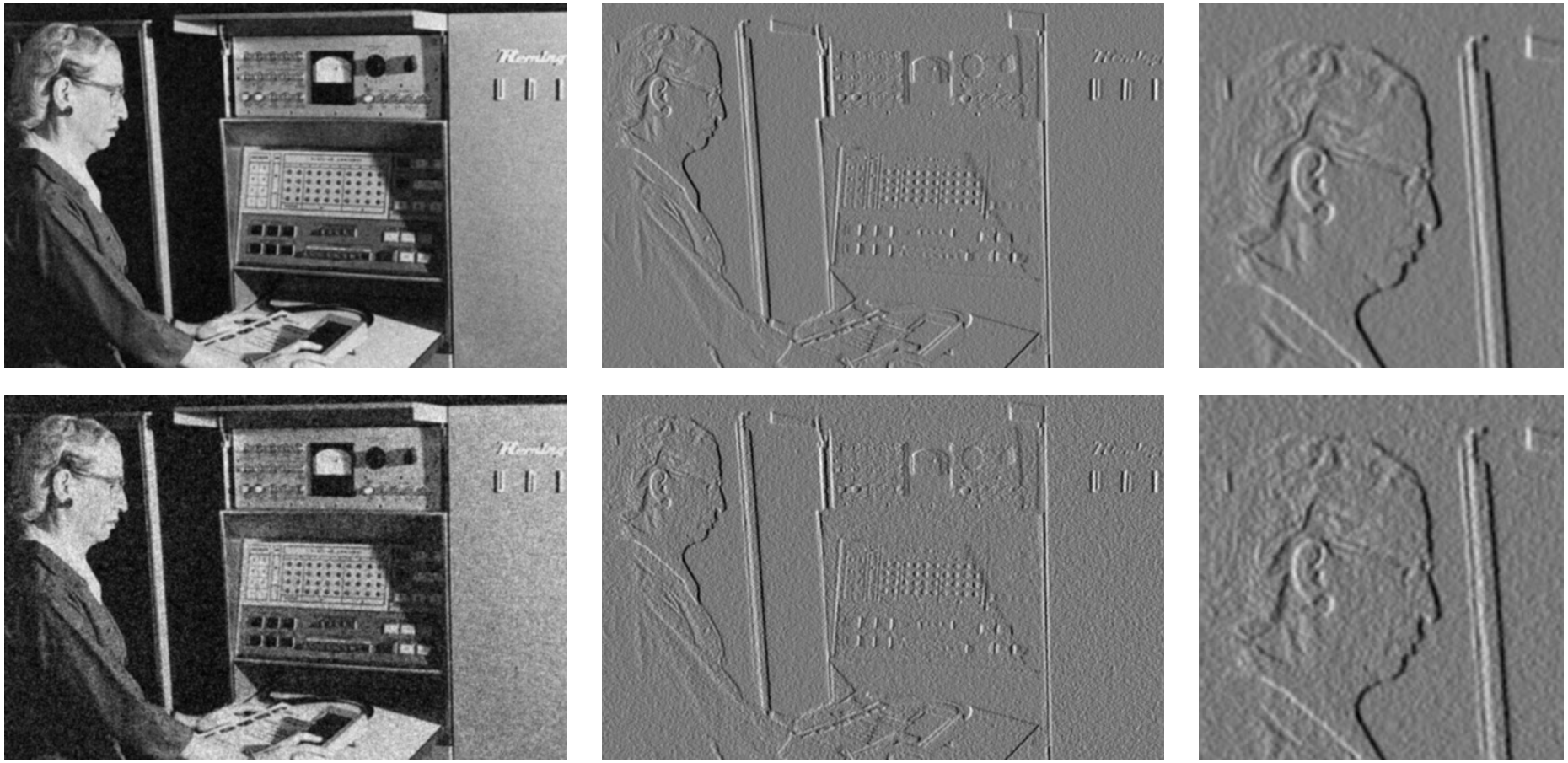
Source: D. Fouhey

Noise + Smoothing

Smoothed Input

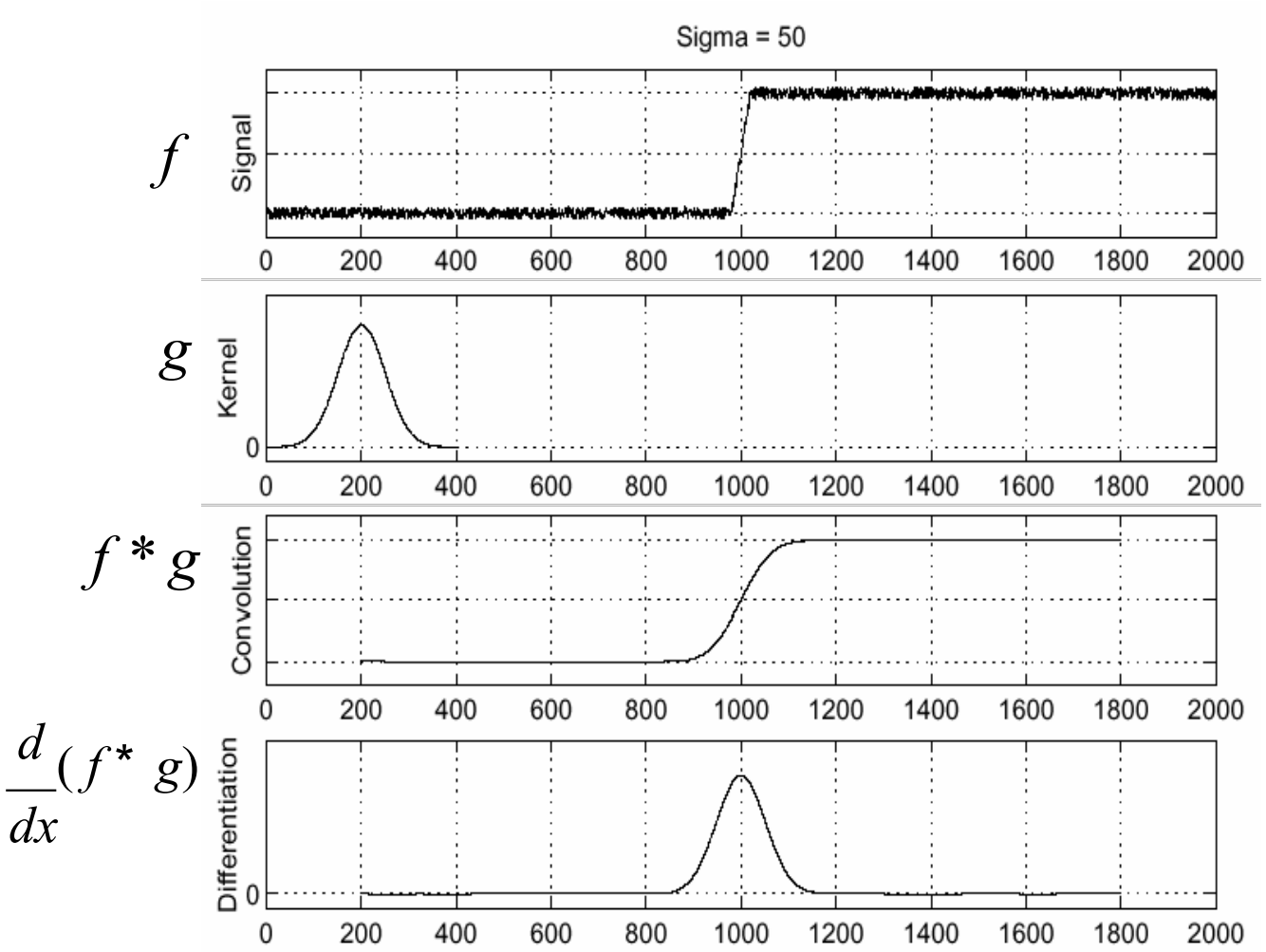
Ix via [-1,01]

Zoom



Source: D. Fouhey

How many steps here?



can we reduce this?

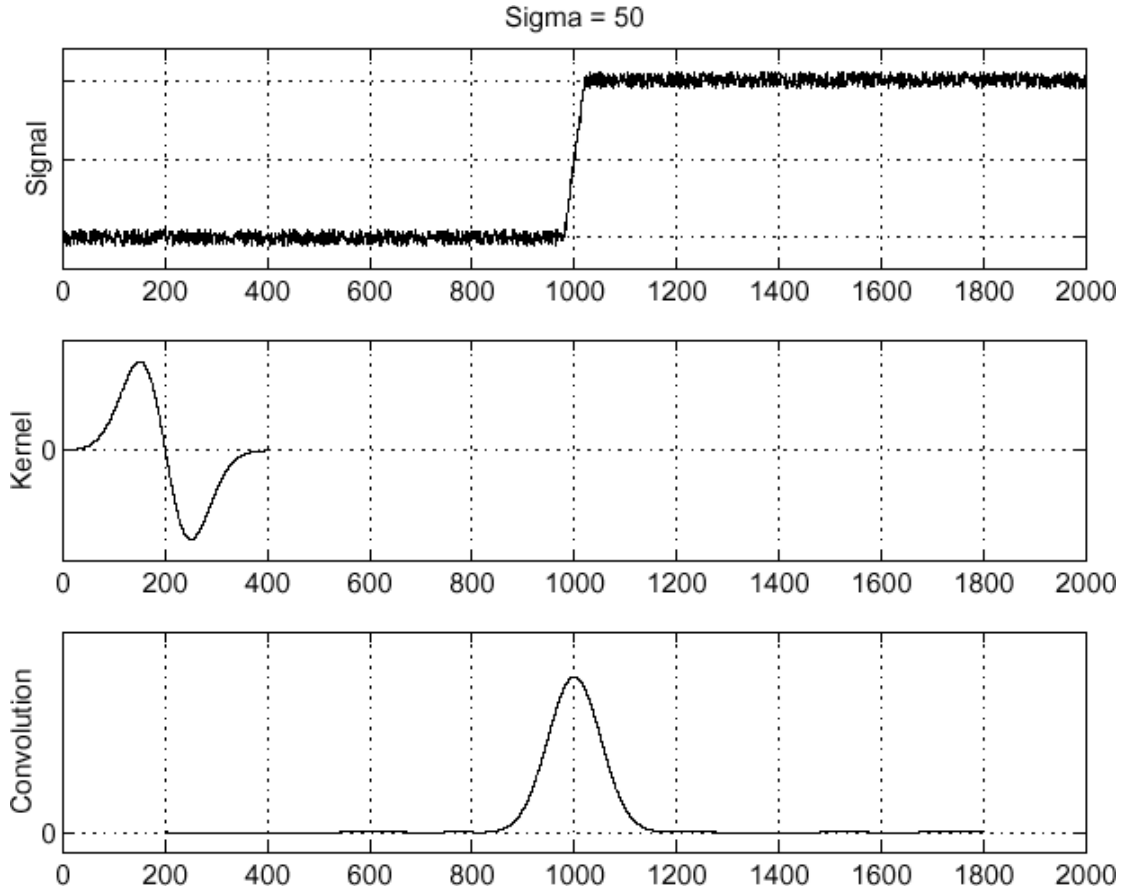
Derivative theorem of convolution

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

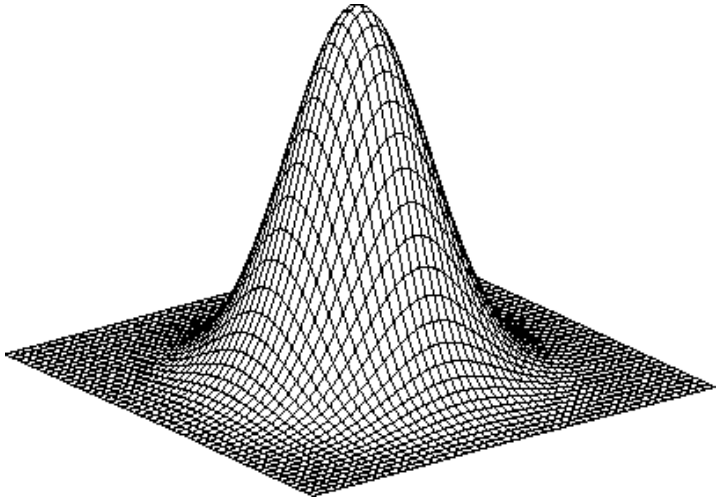
- This saves us one operation:

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

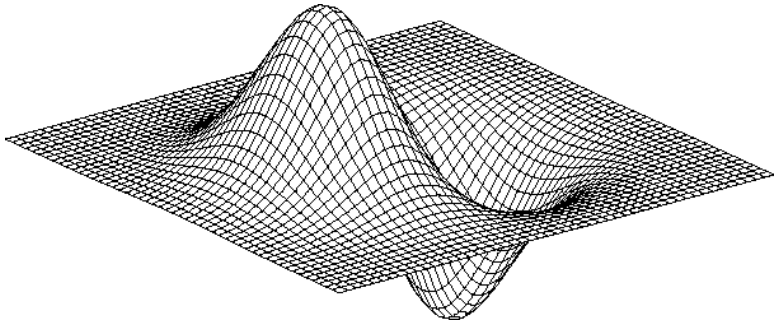
$(\frac{\partial}{\partial x}h) \star f$



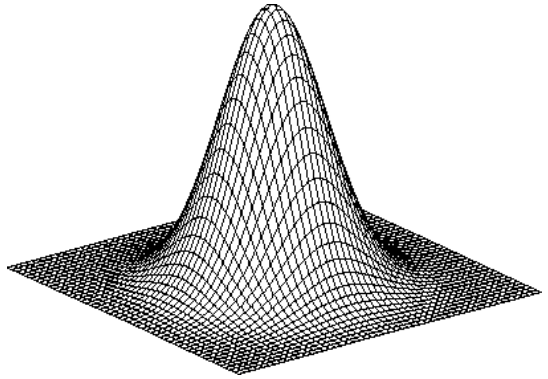
Derivative of Gaussian filter



* [1 -1]=

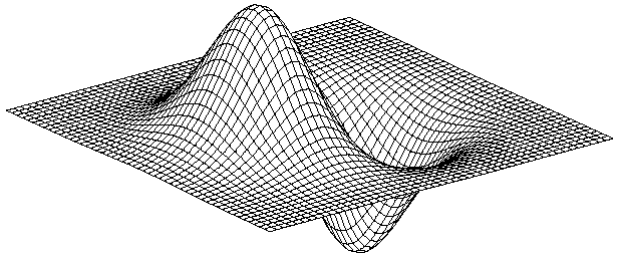


2D edge detection filters



Gaussian

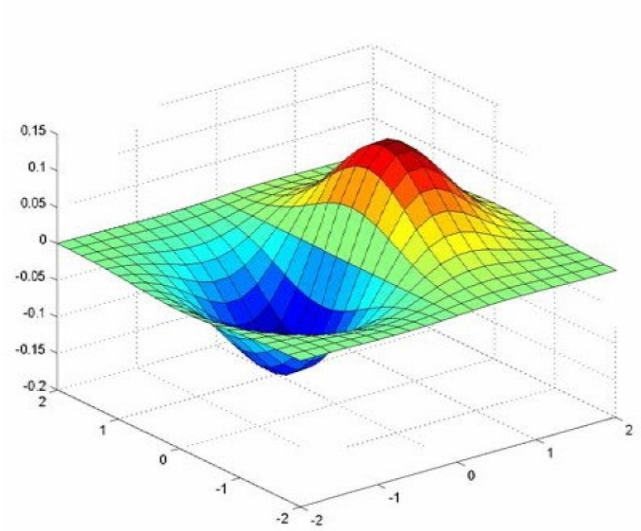
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



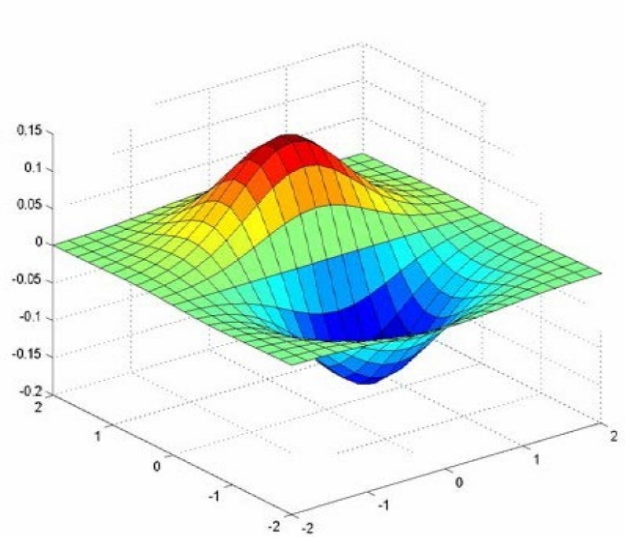
Derivative of Gaussian (x)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

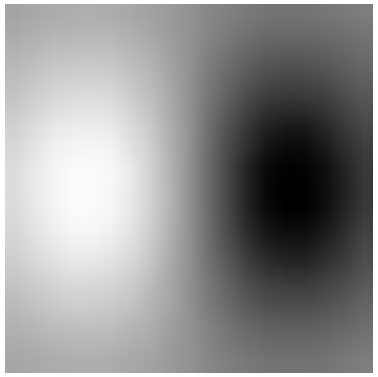
Derivative of Gaussian filter

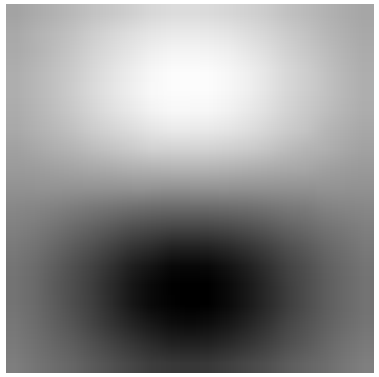


x-direction



y-direction





Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference $f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$ ⟶ 1D derivative filter

1	0	-1

second-order finite difference $f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$ ⟶ Laplace filter ?

Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference $f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$ → 1D derivative filter

1	0	-1

second-order finite difference $f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}$ → Laplace filter

1	-2	1

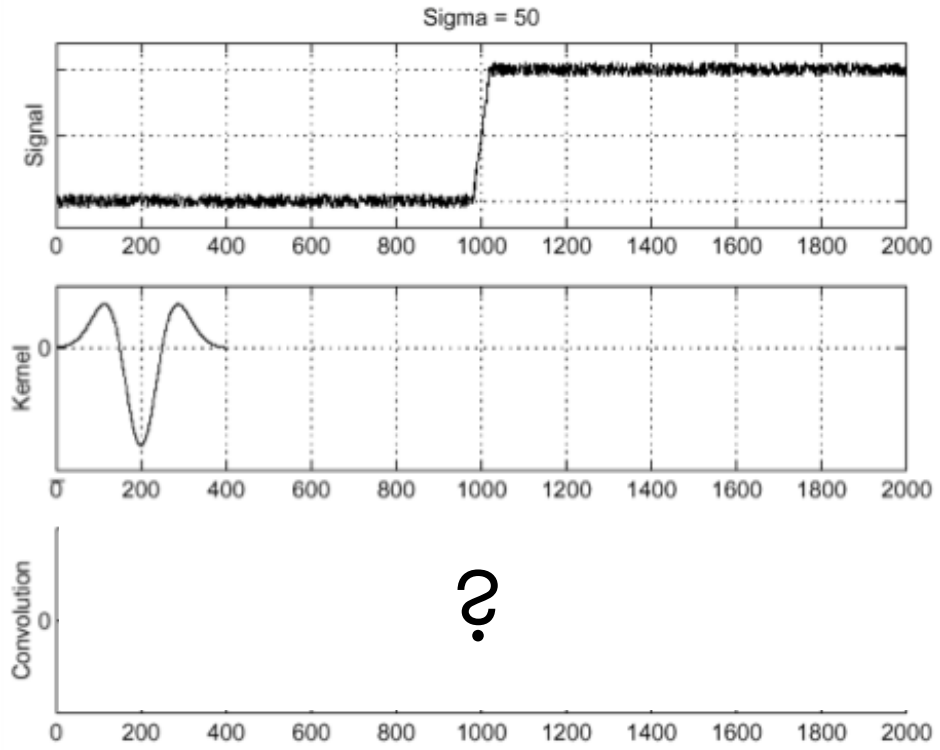
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering

input

Laplacian of Gaussian

output



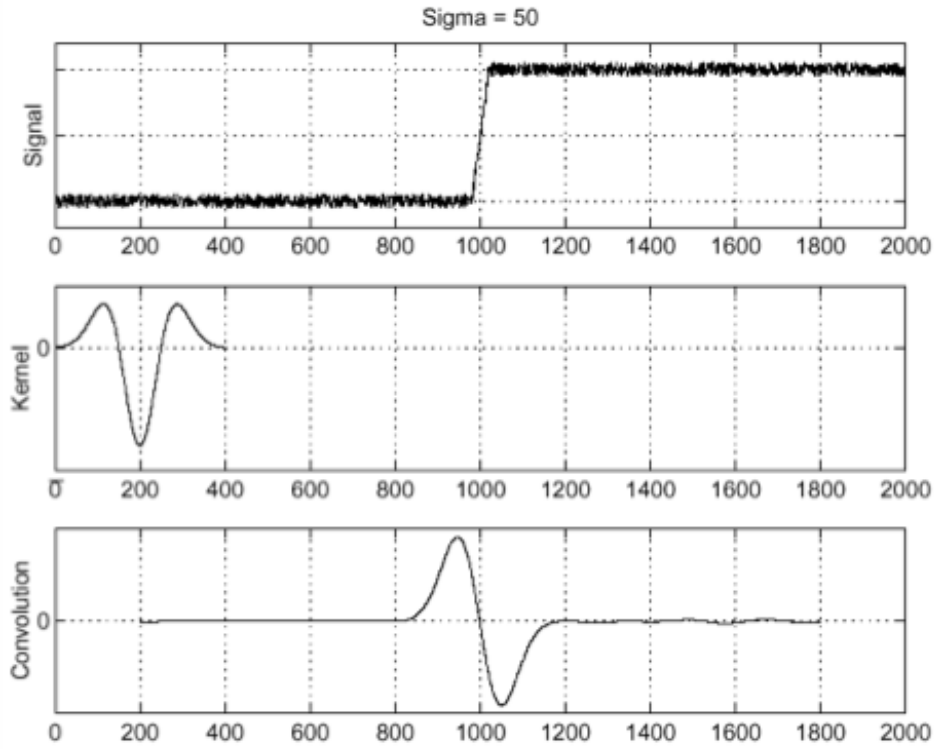
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering

input

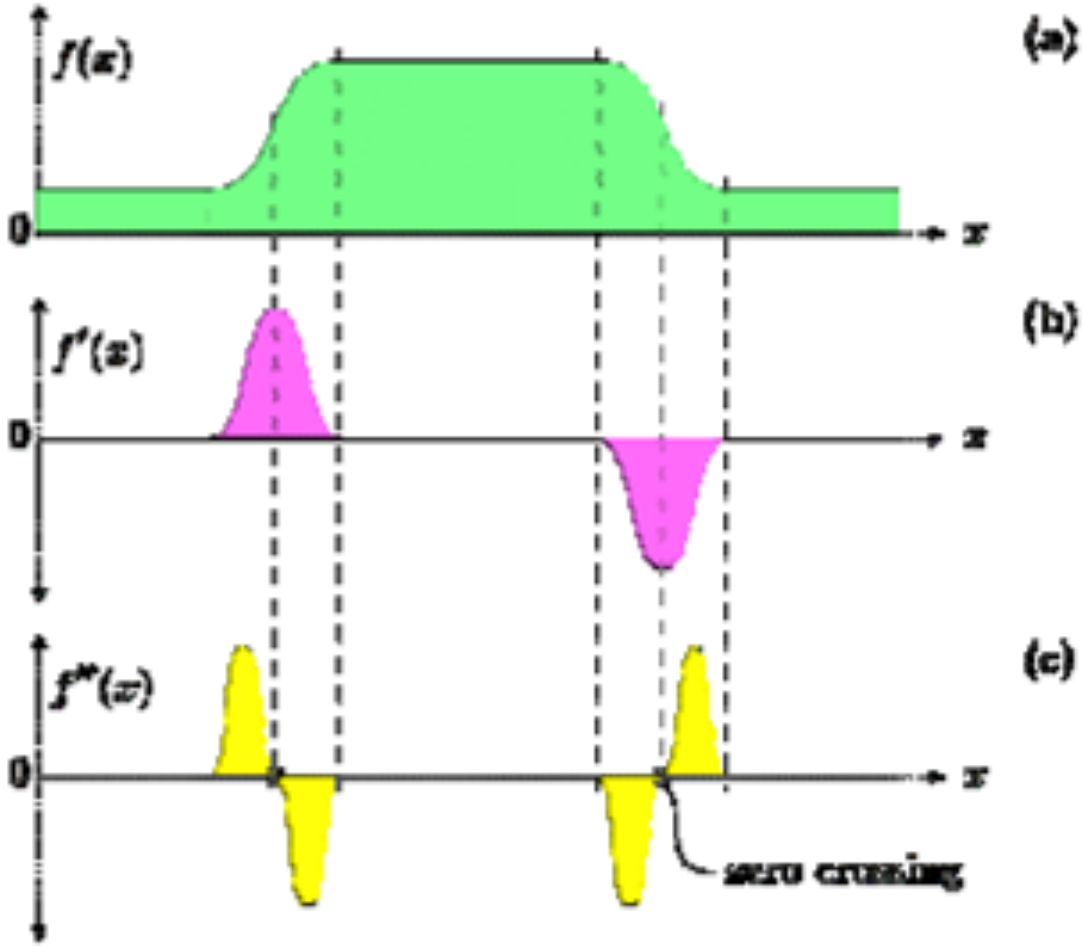
Laplacian of Gaussian

output



"zero crossings" at edges

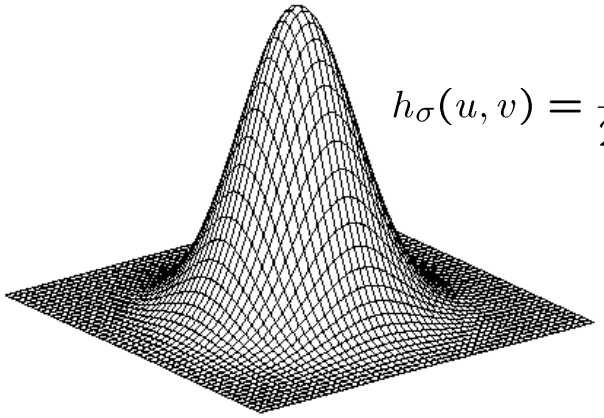
Zero-crossing



First derivative at edge is a maximum
Second derivative at edge is zero

It is more easy to understand when a function is zero than find a maximum

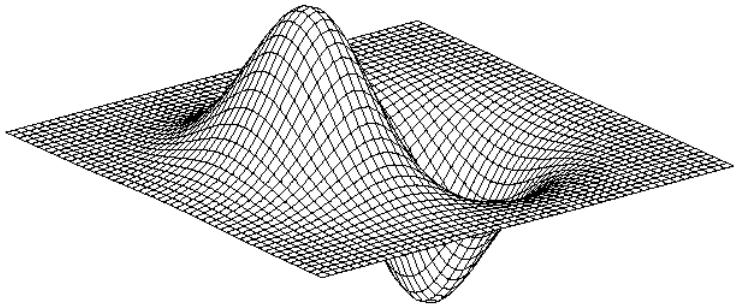
2D Gaussian filters



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

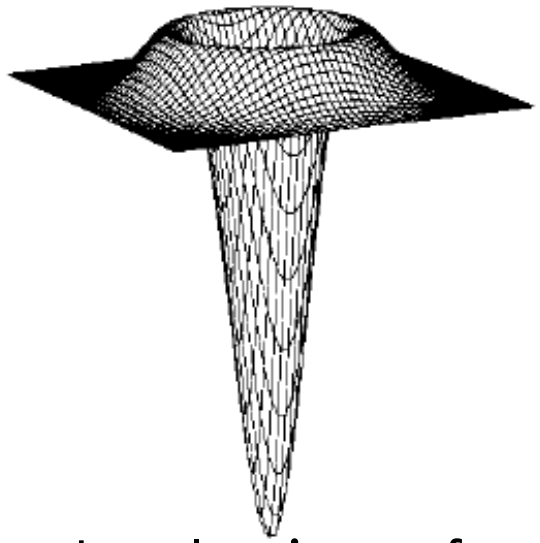
Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$



Derivative of Gaussian

$$\nabla^2 h_\sigma(u,v)$$



Laplacian of Gaussian

Laplace and LoG filtering examples

Laplacian of Gaussian filtering

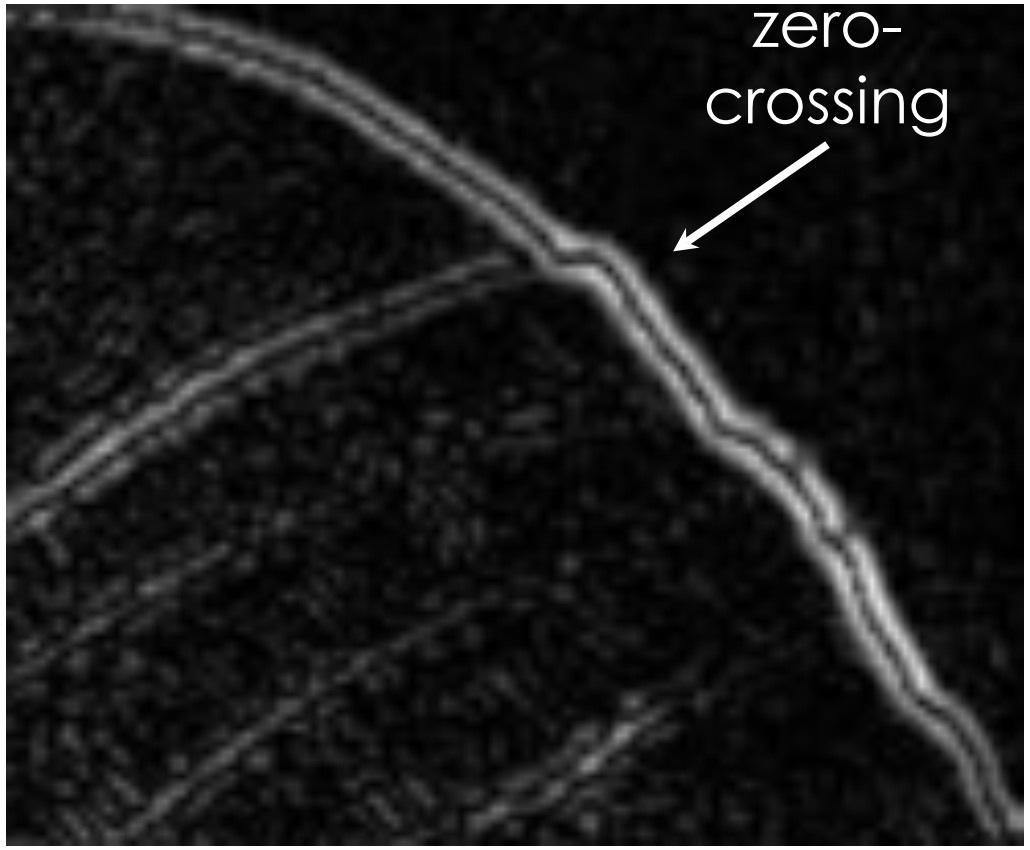
Laplace filtering

Laplacian of Gaussian vs Derivative of Gaussian

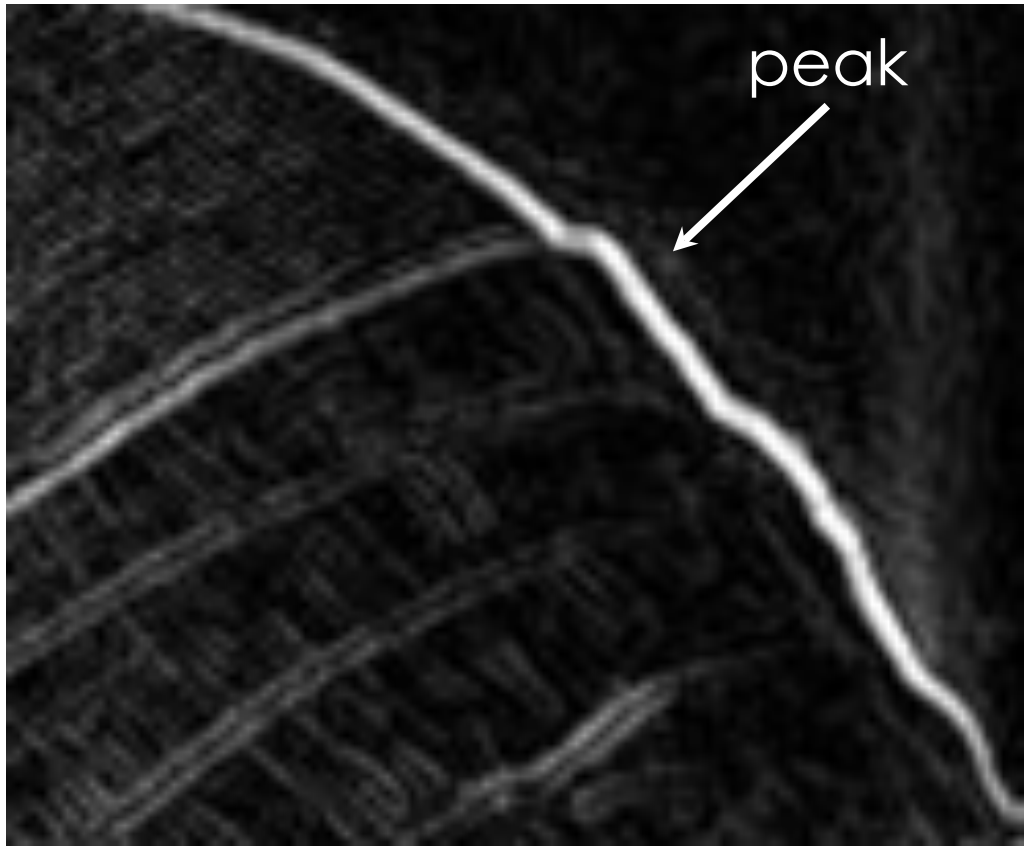
Laplacian of Gaussian filtering

Derivative of Gaussian filtering

Laplacian of Gaussian vs Derivative of Gaussian



Laplacian of Gaussian filtering



Derivative of Gaussian filtering

Zero crossings are more accurate at localizing edges but DoG is computationally more efficient

The Sobel operator

- Common approximation of Derivative of Gaussian

$\frac{1}{8}$

-1	0	1
-2	0	2
-1	0	1

s_x

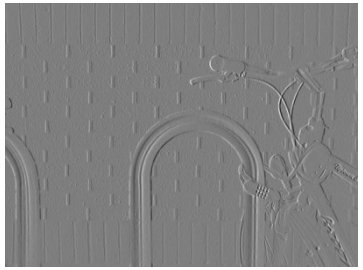
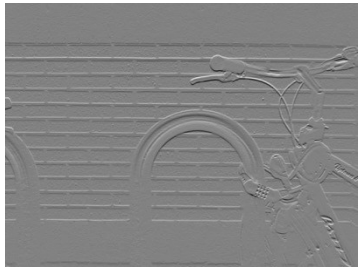
$\frac{1}{8}$

1	2	1
0	0	0
-1	-2	-1

s_y

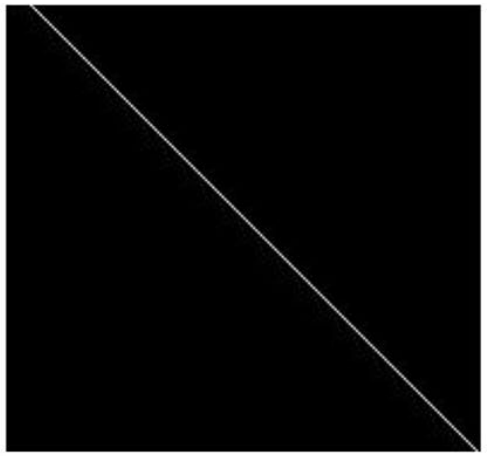
- The standard definition of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient magnitude

Sobel operator: example

Source: Wikipedia

Image with Edge



Edge Location

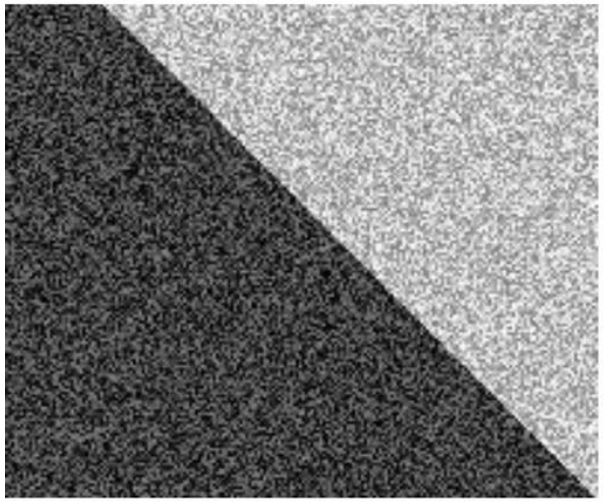
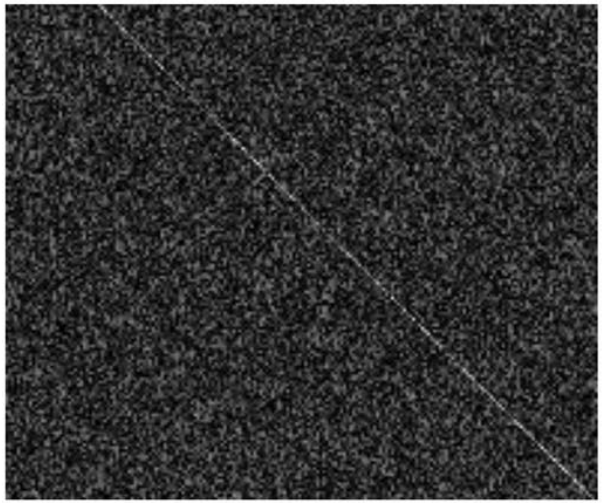
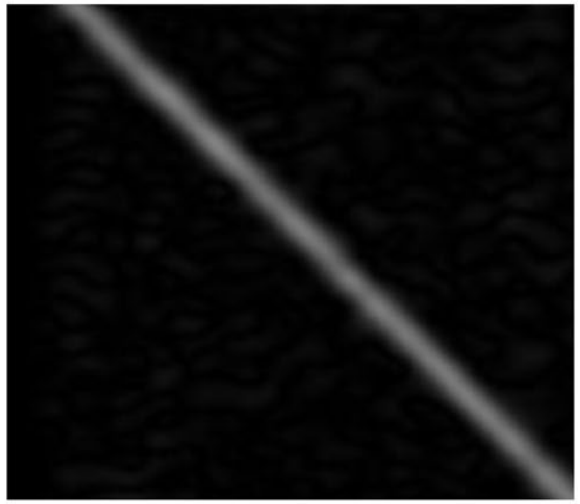


Image + Noise

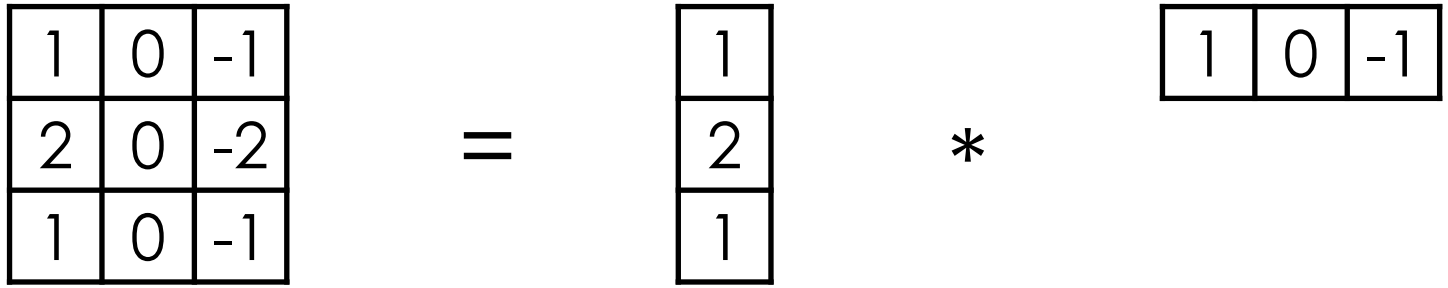


Derivatives detect edge *and* noise



Smoothed derivative removes noise, but blurs edge

The Sobel filter

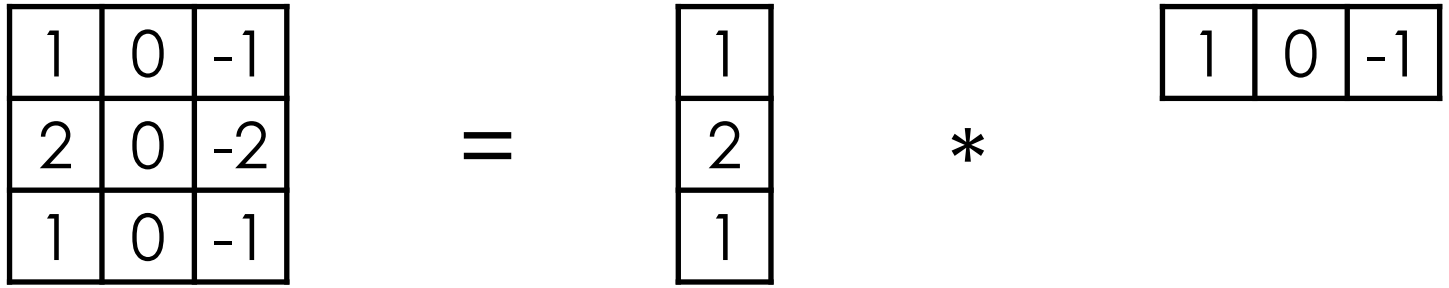


Sobel filter

What filter is this?

1D derivative filter

The Sobel filter



Sobel filter = Blurring * 1D derivative filter

Does this filter return large responses on vertical or horizontal lines?

The Sobel filter

Horizontal Sober filter:

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

What does the vertical Sobel filter look like?

The Sobel filter

Horizontal Sober filter:

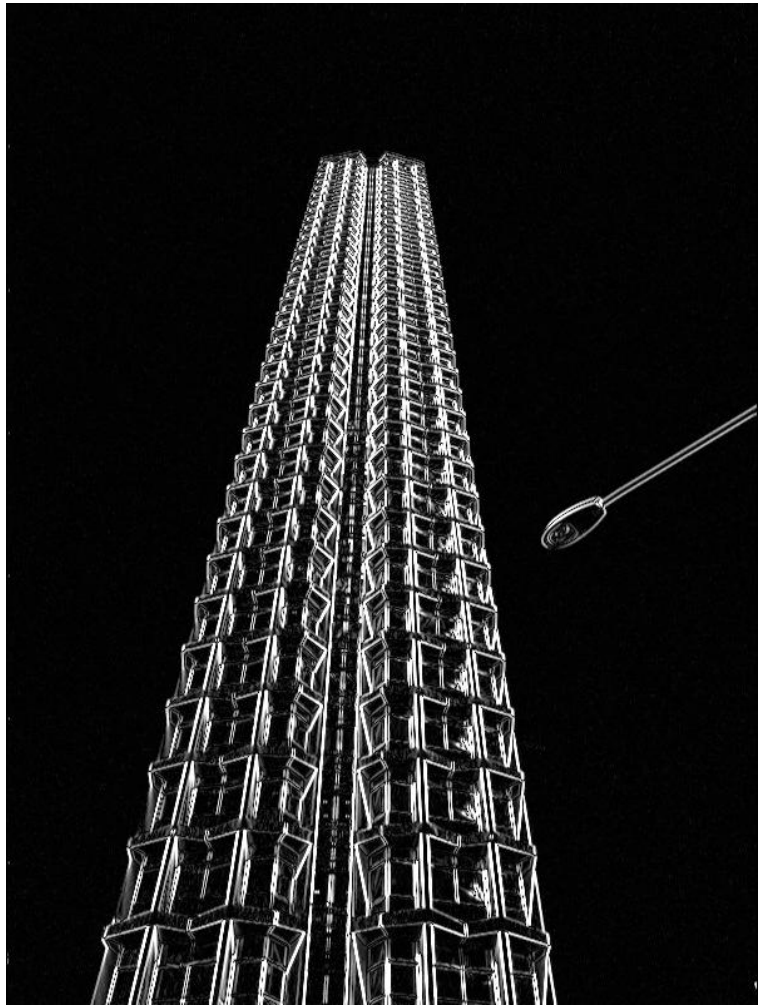
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Vertical Sobel filter:

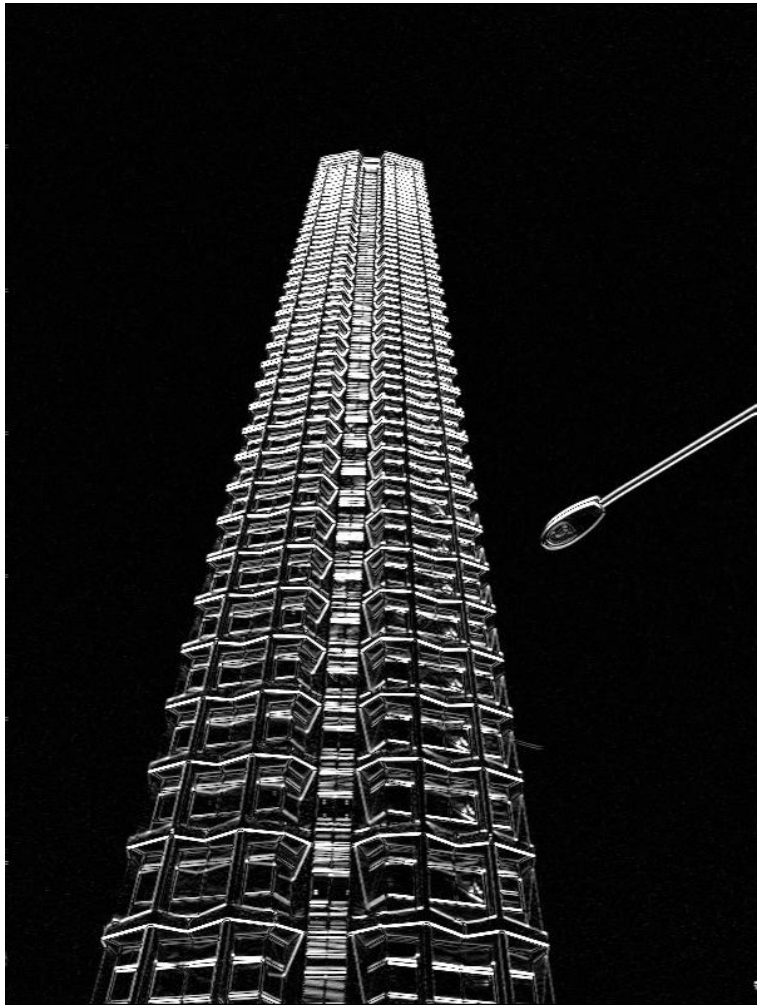
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Sobel filter example

original



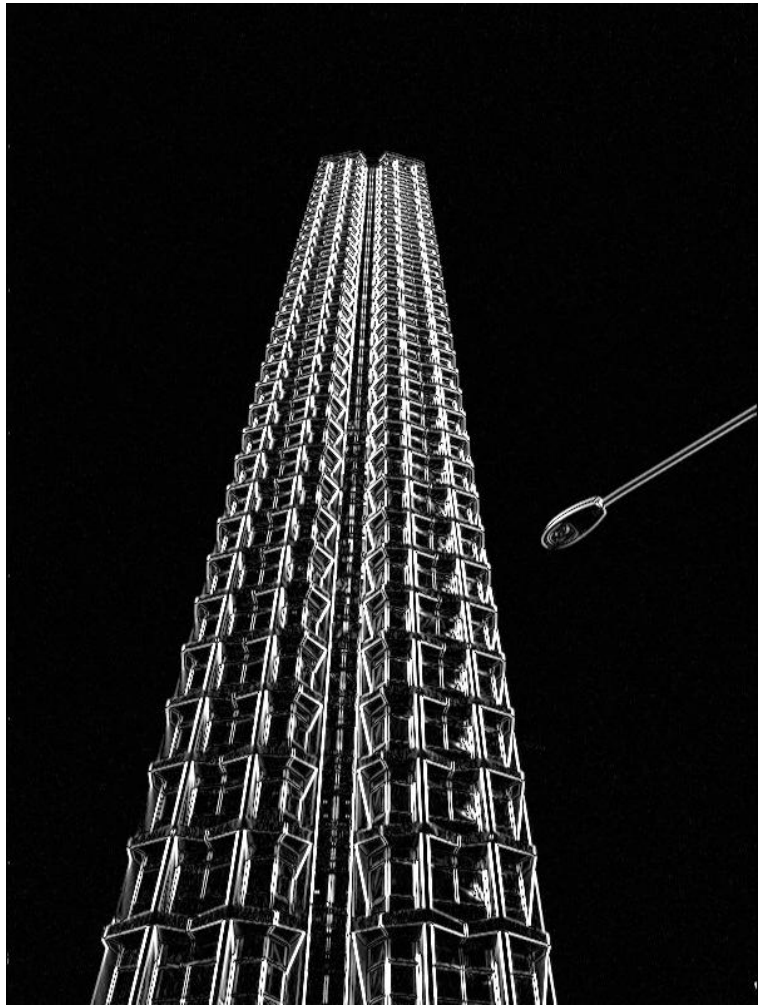
which Sobel filter?



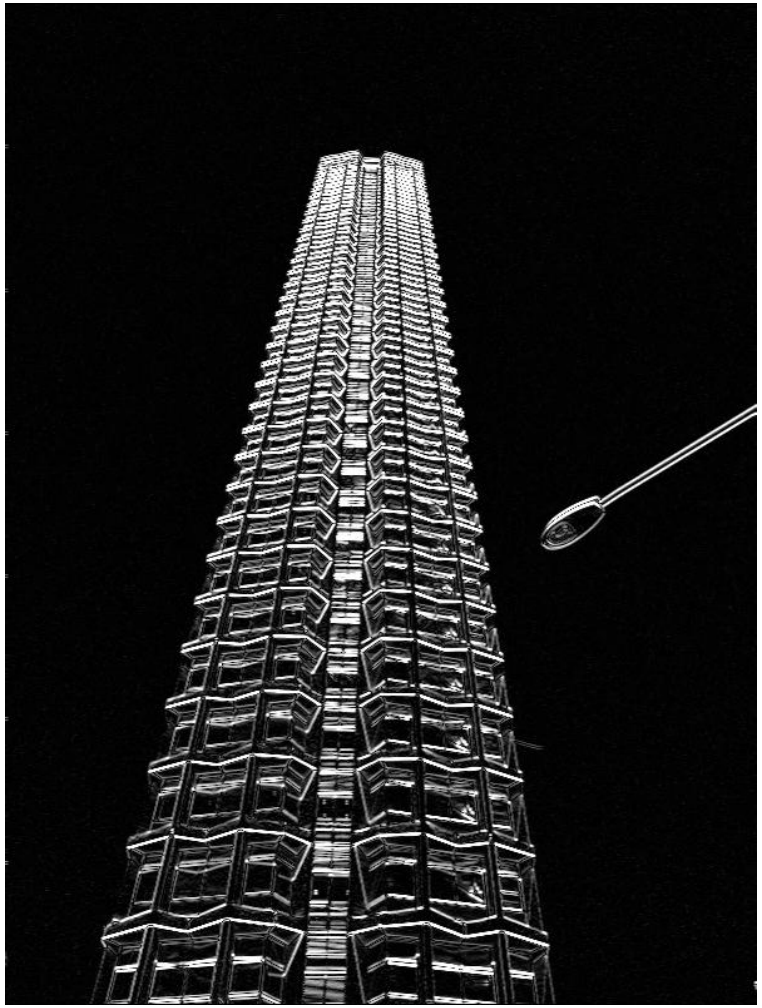
which Sobel filter?

Sobel filter example

original

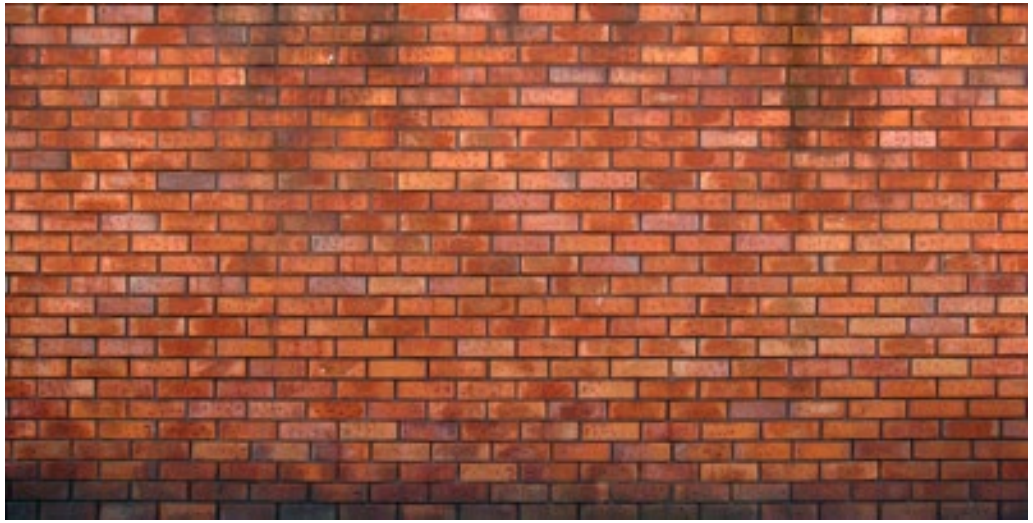


horizontal Sobel filter



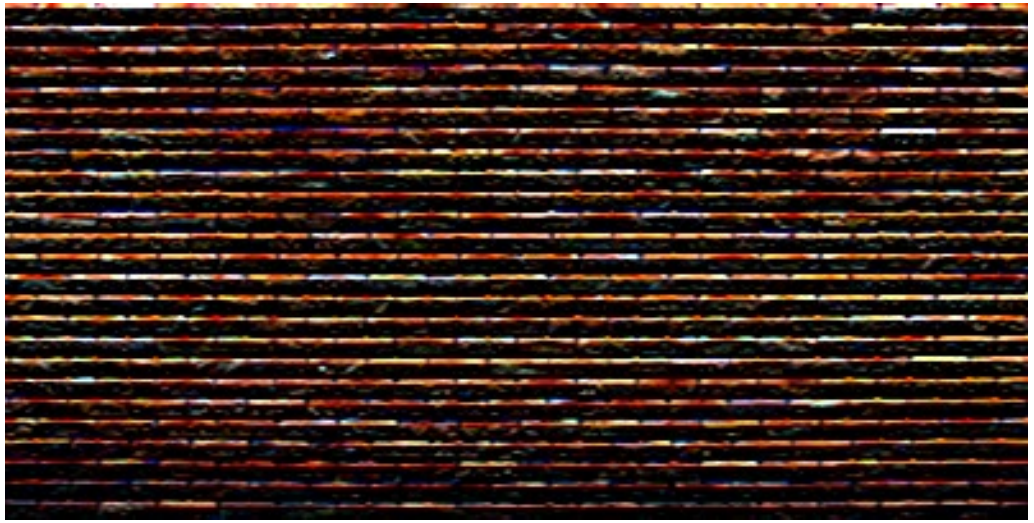
vertical Sobel filter

Sobel filter example



original

horizontal Sobel filter



vertical Sobel filter

Several derivative filters

Sobel

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

Scharr

3	0	-3
10	0	-10
3	0	-3

3	10	3
0	0	0
-3	-10	-3

Prewitt

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

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

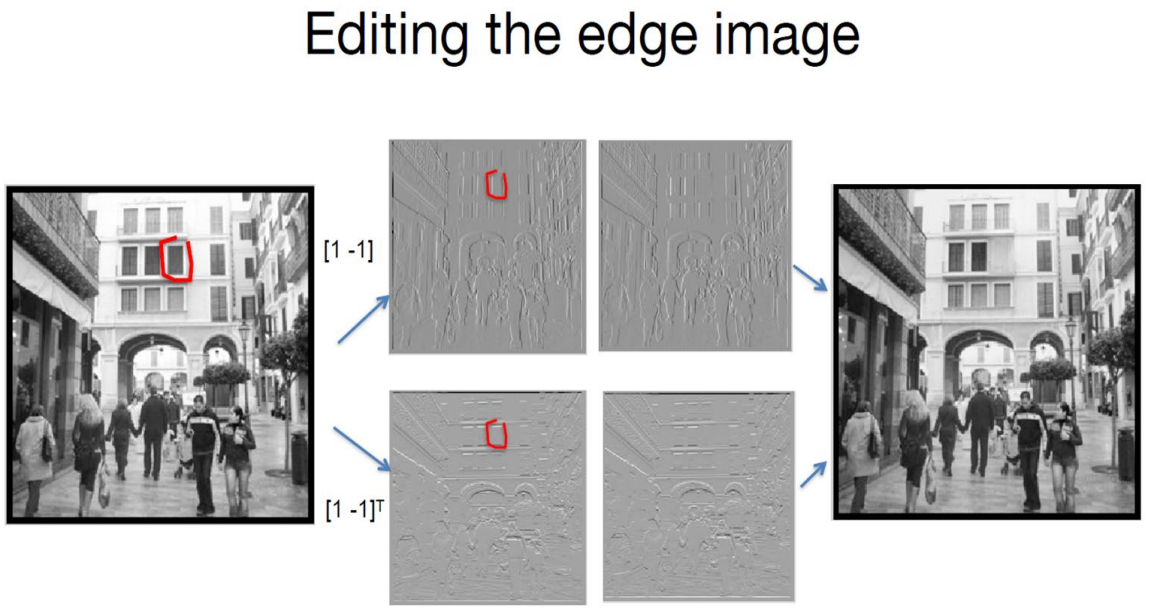
Roberts

0	1
-1	0

1	0
0	-1

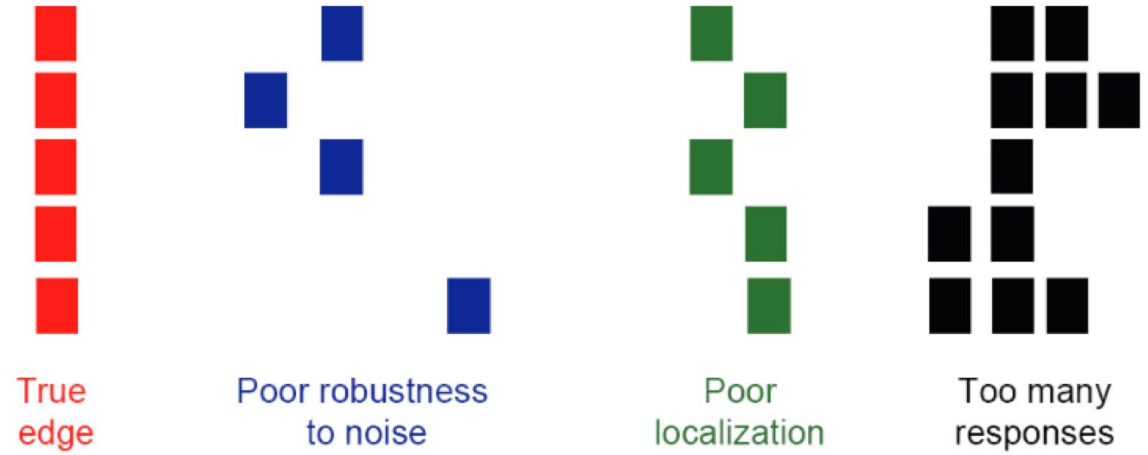
Reconstruction from 2D derivatives

- In 2D, we have multiple derivatives (along n and m) and we compute the pseudo-inverse of the full matrix



Edge detector

- The optimal edge detector must be accurate, minimizing the number of false positives and false negatives; have precise localization, pinpointing edges at the positions where they actually occur; and have single response, ensuring that only one edge is found where there only is one edge



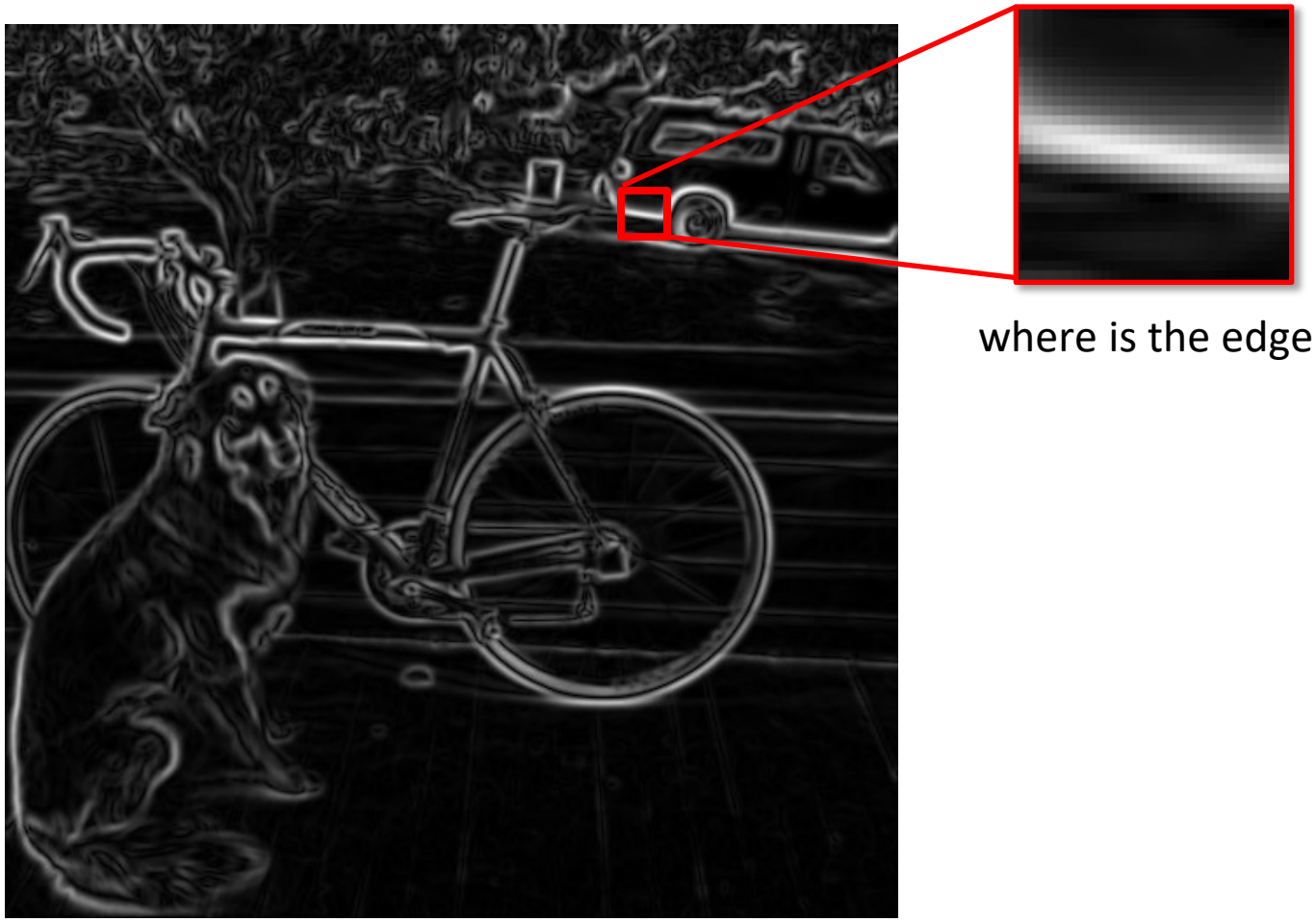
Example

original image

Demo:
http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Image credit: Joseph Redmon

Finding edges

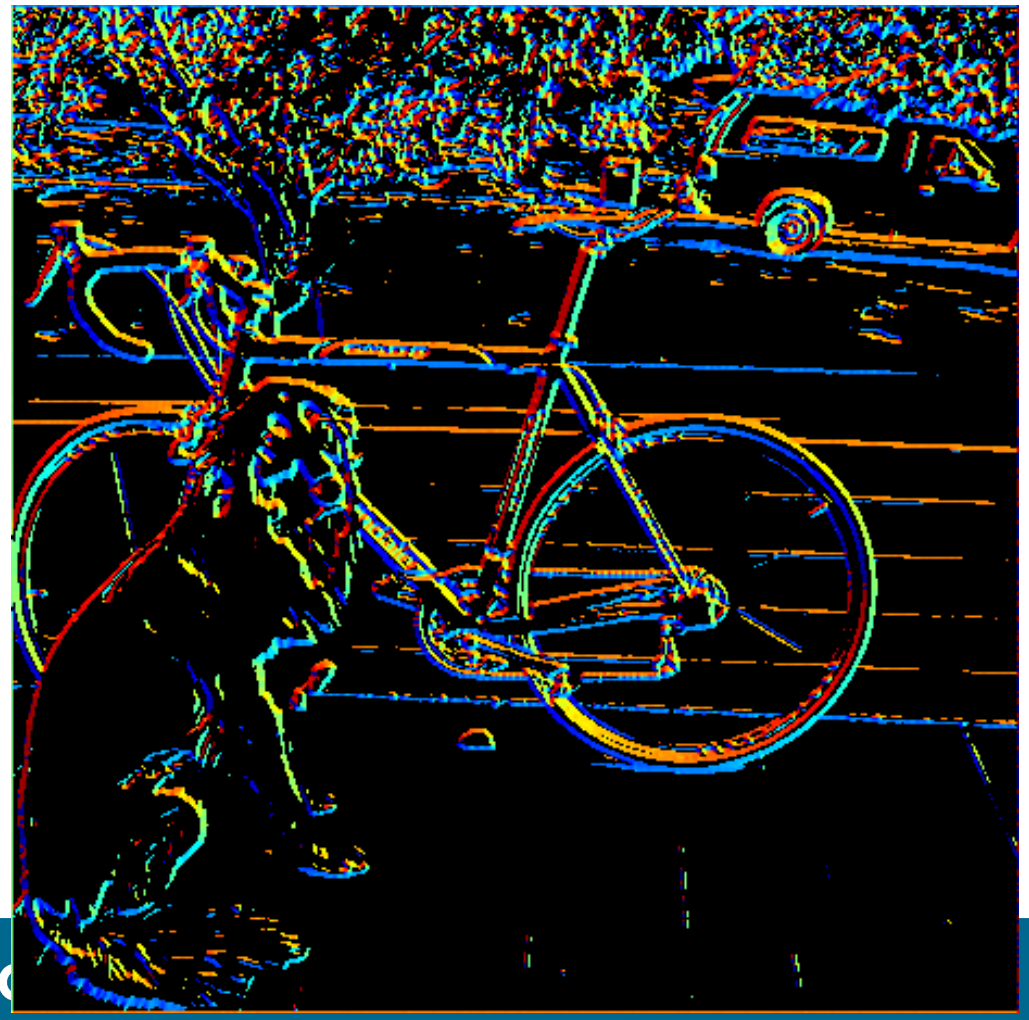


where is the edge?

smoothed gradient magnitude

Get Orientation at Each Pixel

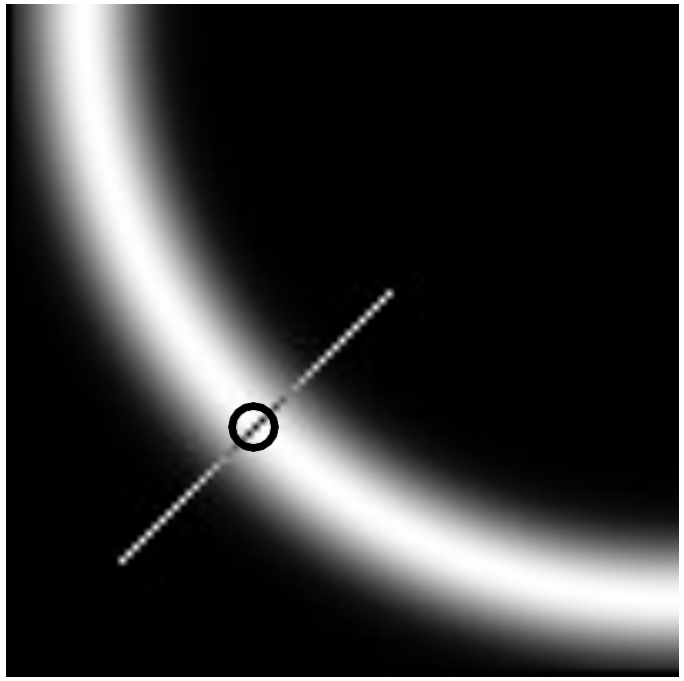
- Get orientation (below, threshold at minimum gradient magnitude)

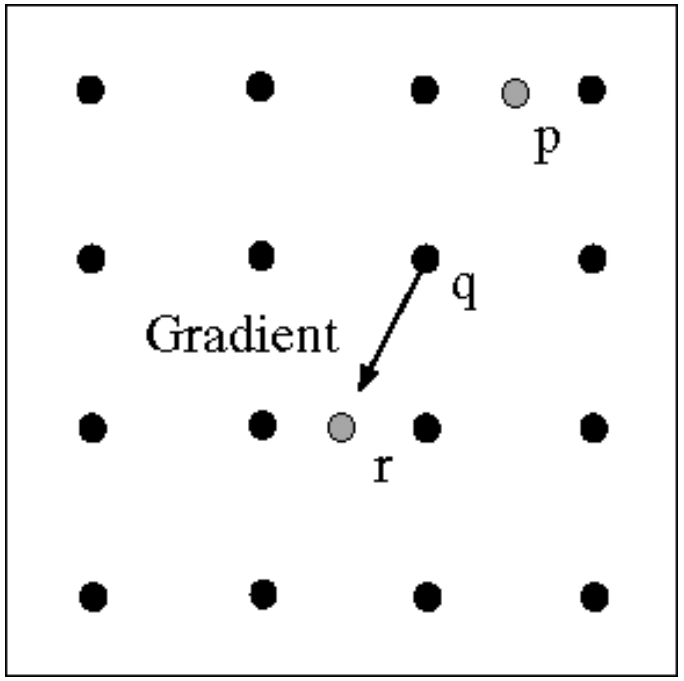


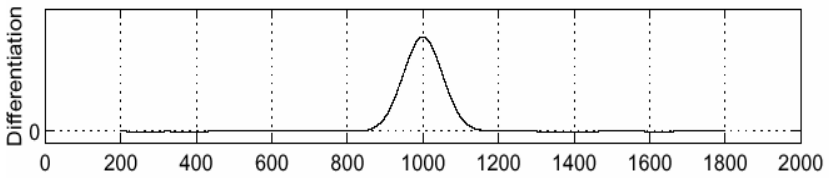
theta = atan2(gy, gx)

Gradient orientation angle

Non-maximum supression





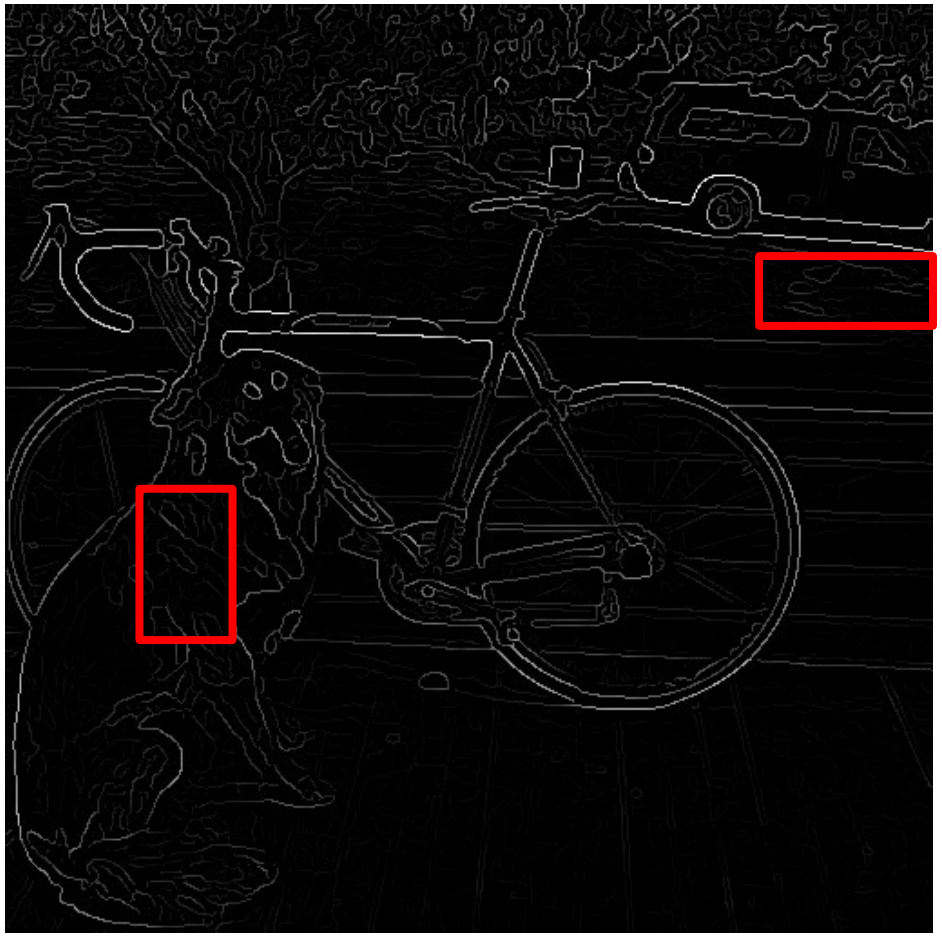


- Check if pixel is local maximum along gradient direction
  - requires *interpolating* pixels p and r

Before Non-max Suppression

After Non-max Suppression

Still noise exists!

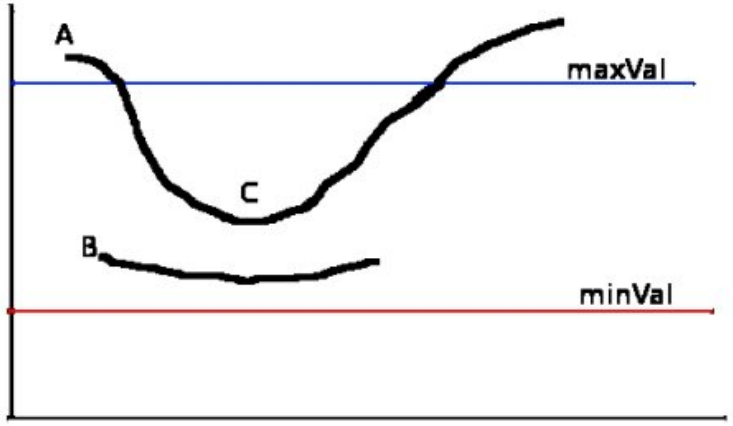


Thresholding edges

Still some noise but we want only strong edges

- 2 thresholds, 3 cases (R is the intensity gradient)
  - R > T: strong edge
  - R < T but R > t: weak edge
  - R < t: no edge
- Strong edges are edges!
- Weak edges are edges iff they connect to strong
  - Look in some neighborhood (usually 8 closest)

Hysteresis thresholding

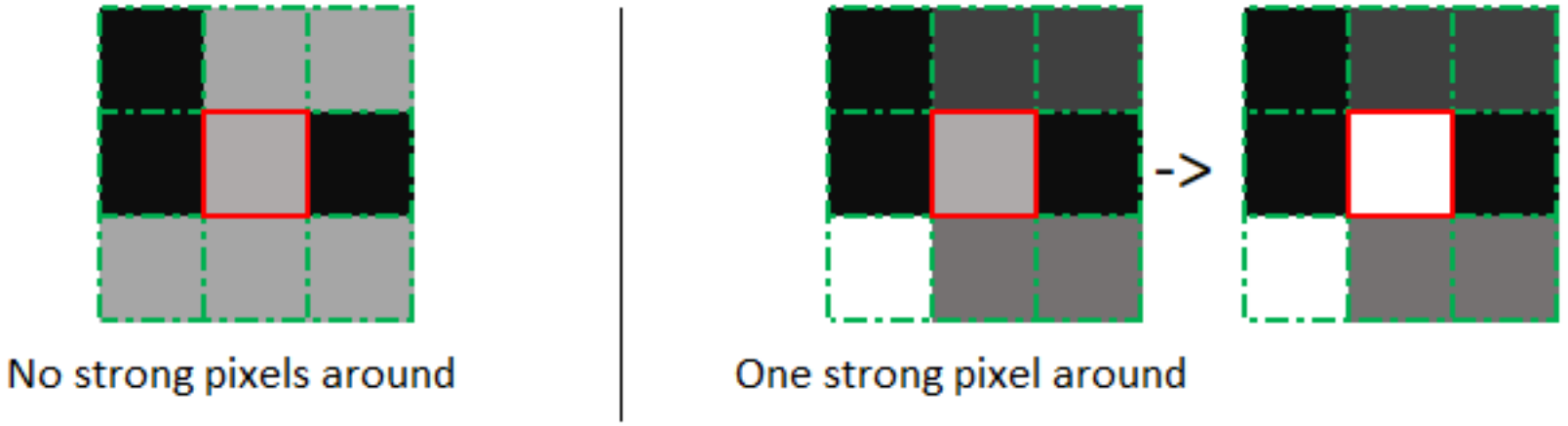


- The edge A is above the maxVal, so considered as "sure-edge".
- Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve.
- Edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded.
- So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

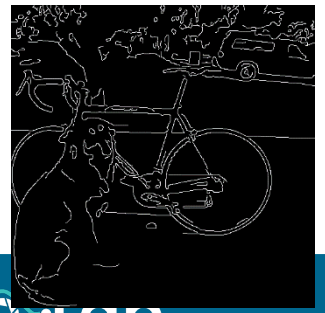
This stage also removes small pixels noises on the assumption that edges are long lines.

Edge Tracking by Hysteresis

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one, as described below:



Canny edge detector



1. Filter image with Derivative of Gaussian

1. Find magnitude and orientation of gradient

1. Non-maximum suppression

1. Linking and thresholding (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

Source: D. Lowe, L. Fei-Fei, J. Redmon

Canny edge detector

- Our first computer vision pipeline!
- Still a widely used edge detector in computer vision
  - J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.
- Depends on several parameters:
  - high threshold low threshold
  - width of the Gaussian blur σ

Canny edge detector

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

- The choice of σ depends on desired behavior
  - large σ detects "large-scale"
  - small σ edges detects fine edges

Source: S. Seitz

Slide Credits

- CS5670, Introduction to Computer Vision, Cornell Tech, by Noah Snavely.
- CS 194-26/294-26: Intro to Computer Vision and Computational Photography, UC Berkeley, by Alyosha Efros.
- CS 15-463, 663, 862, CMU, by Computational Photography, Ioannis Gkioulekas.

Acknowledgements: some slides and material from Bernt Schiele, Mario Fritz, Michael Black, Bill Freeman, Fei-Fei Li, Justin Johnson, Serena Yeung, R. Szelisky, Ioannis Gkioulekas, Roni Sengupta, Andreas Geiger