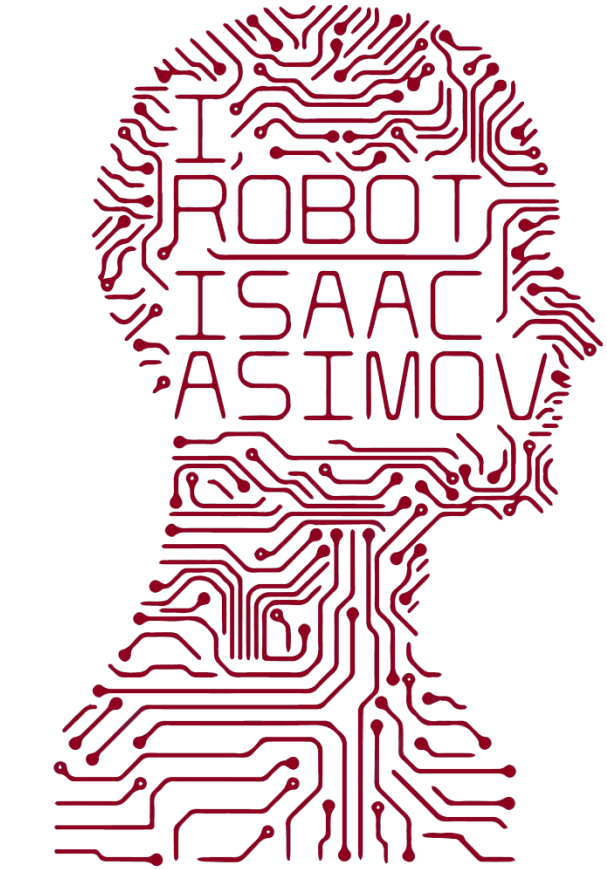
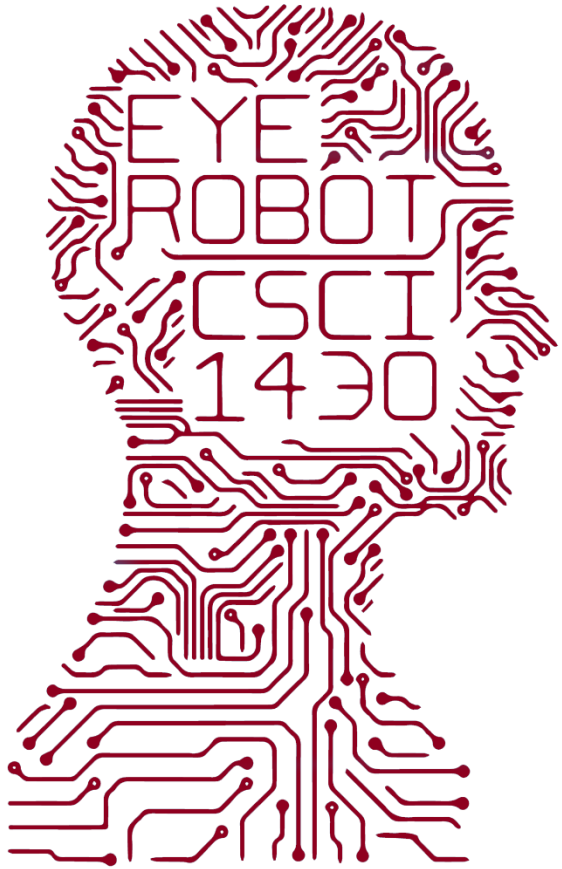


FUTURE VISION



COMPUTER VISION

Prismia.Chat https://tinyurl.com/y26ragof

Types of image transformations

What is an image?

What is an image?

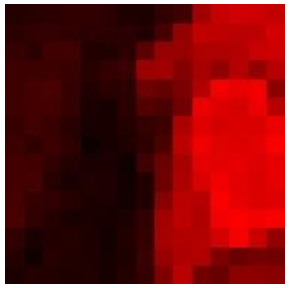
A (color) image is a 3D tensor of numbers.

What is an image?

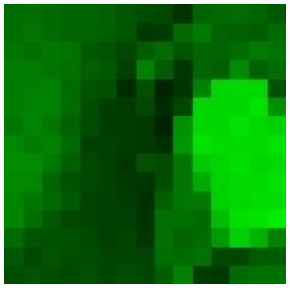
color image patch

How many bits are the intensity values?

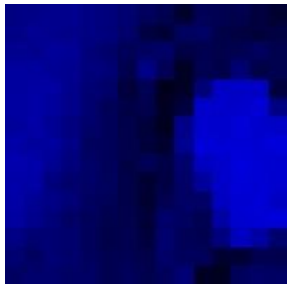
red



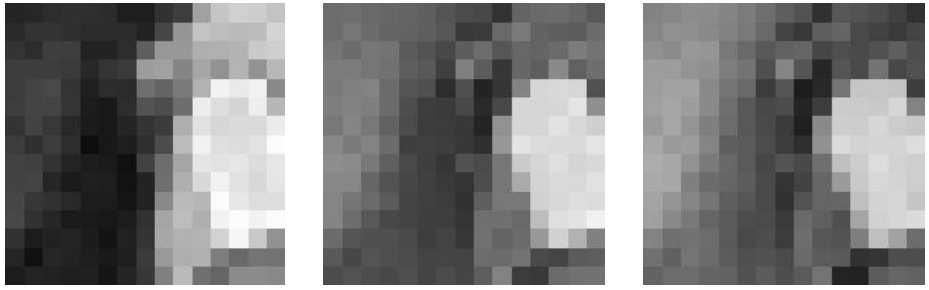
green



blue



colorized for visualization



actual intensity values per channel

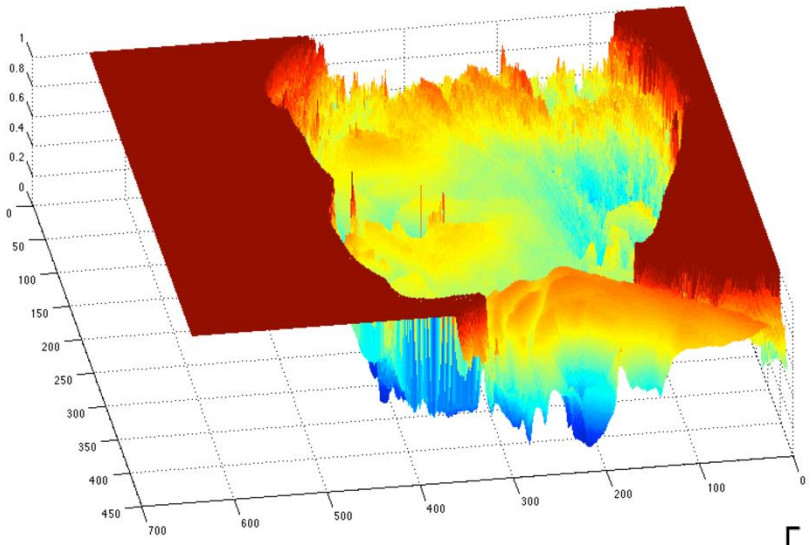
Each channel is a 2D array of numbers.

What is an image?

grayscale image

What is the range of the image function f?

$f(\boldsymbol{x})$



domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

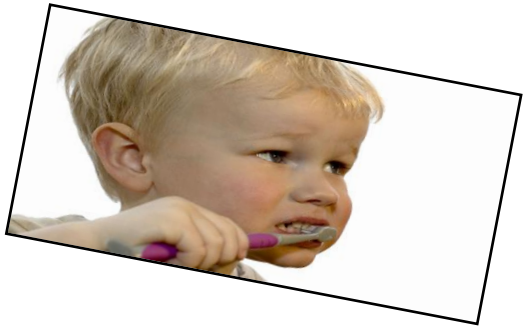
A (grayscale) image is a 2D function.

What types of image transformations can we do?

Filtering

changes pixel *values*

Warping



changes pixel *locations*

What types of image transformations can we do?

Filtering

F

$G(\boldsymbol{x}) = h\{F(\boldsymbol{x})\}$

G

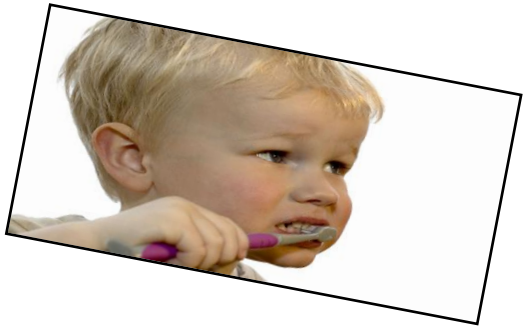
changes *range* of image function

Warping

F

$G(\boldsymbol{x}) = F(h\{\boldsymbol{x}\})$

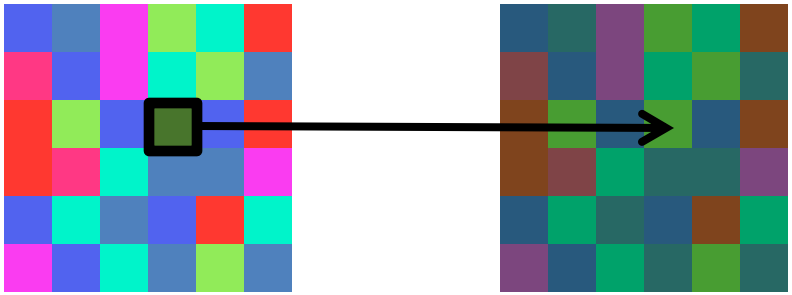
G



changes *domain* of image function

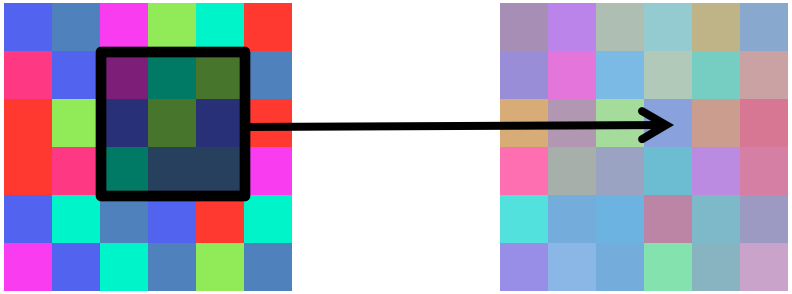
What types of image filtering can we do?

Point Operation



point processing

Neighborhood Operation



“filtering”

Point processing

Examples of point processing

original

darken

lower contrast

non-linear lower contrast

invert

lighten

raise contrast

non-linear raise contrast

Examples of point processing

How would you implement these?

original

darken

lower contrast

non-linear lower contrast

x

invert

lighten

raise contrast

non-linear raise contrast

Examples of point processing

How would you implement these?

original

x

darken

$x - 128$

lower contrast

non-linear lower contrast

invert

lighten

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

$x - 128$

lower contrast

$\frac{x}{2}$

non-linear lower contrast

invert

lighten

raise contrast

non-linear raise contrast

How would you implement these?

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

lighten

raise contrast

non-linear raise contrast

How would you implement these?

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

raise contrast

non-linear raise contrast

How would you implement these?

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

non-linear raise contrast

How would you implement these?

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

How would you implement these?

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

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

Many other types of point processing

camera output

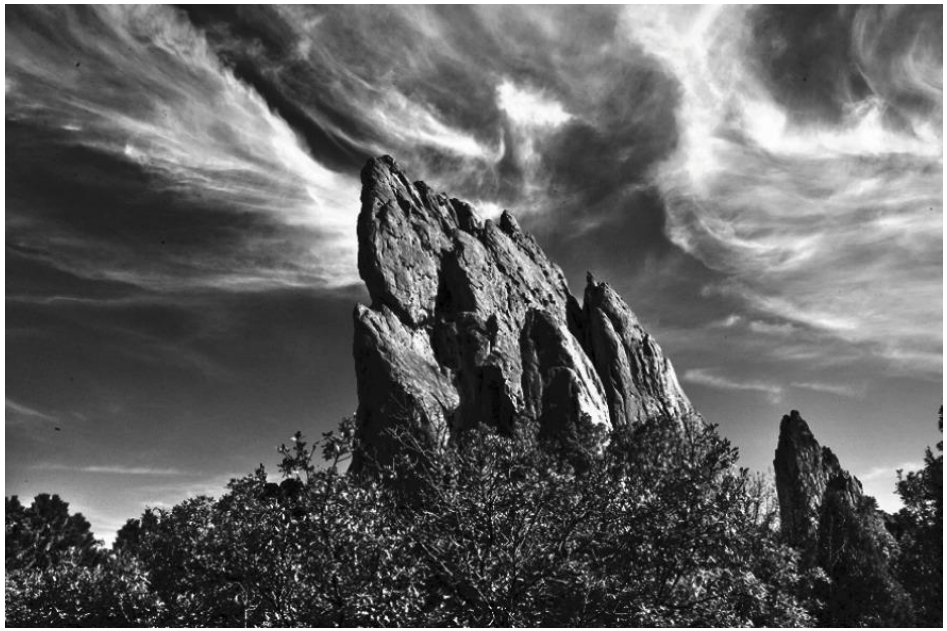
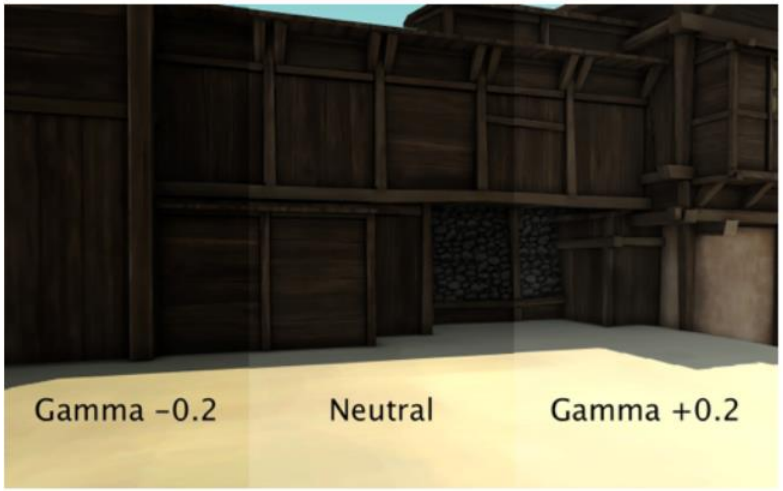


image after stylistic tonemapping

[Bae et al., SIGGRAPH 2006]

Many other types of point processing



Dimensionality of an Image

@ 8bit = 256 values ^ 65,536

- Computer says ‘Inf’ combinations.

Some depiction of all possible scenes would fit into this memory.

Dimensionality of an Image

@ 8bit = 256 values ^ 65,536

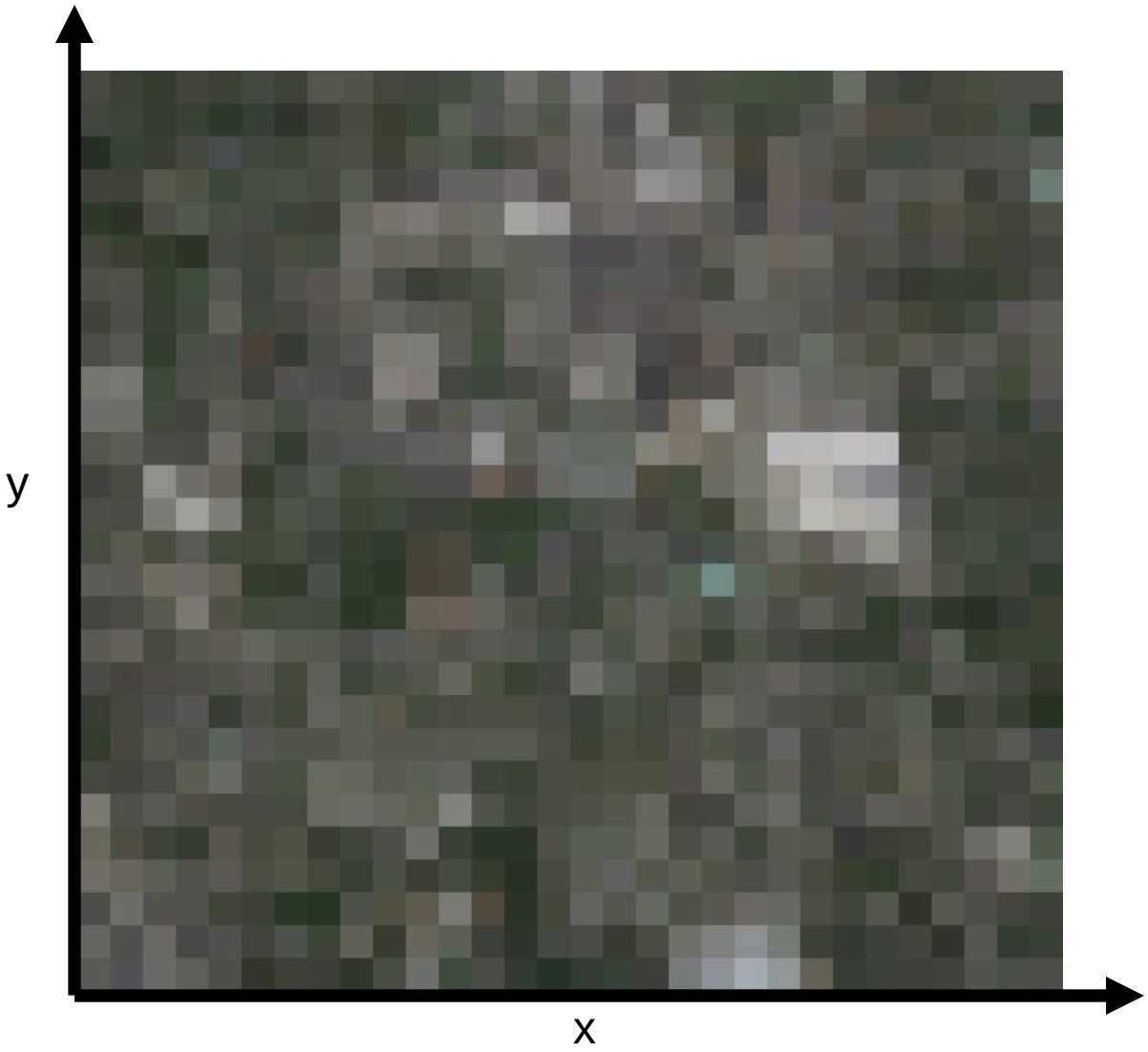
- Computer says ‘Inf’ combinations.

Some depiction of all possible scenes would fit into this memory.

Computer vision as making sense of an extremely high-dimensional space.

- Subspace of ‘natural’ images.
- Deriving low-dimensional, explainable models.

What is each part of an image?



What is each part of an image?

- Pixel -> picture element

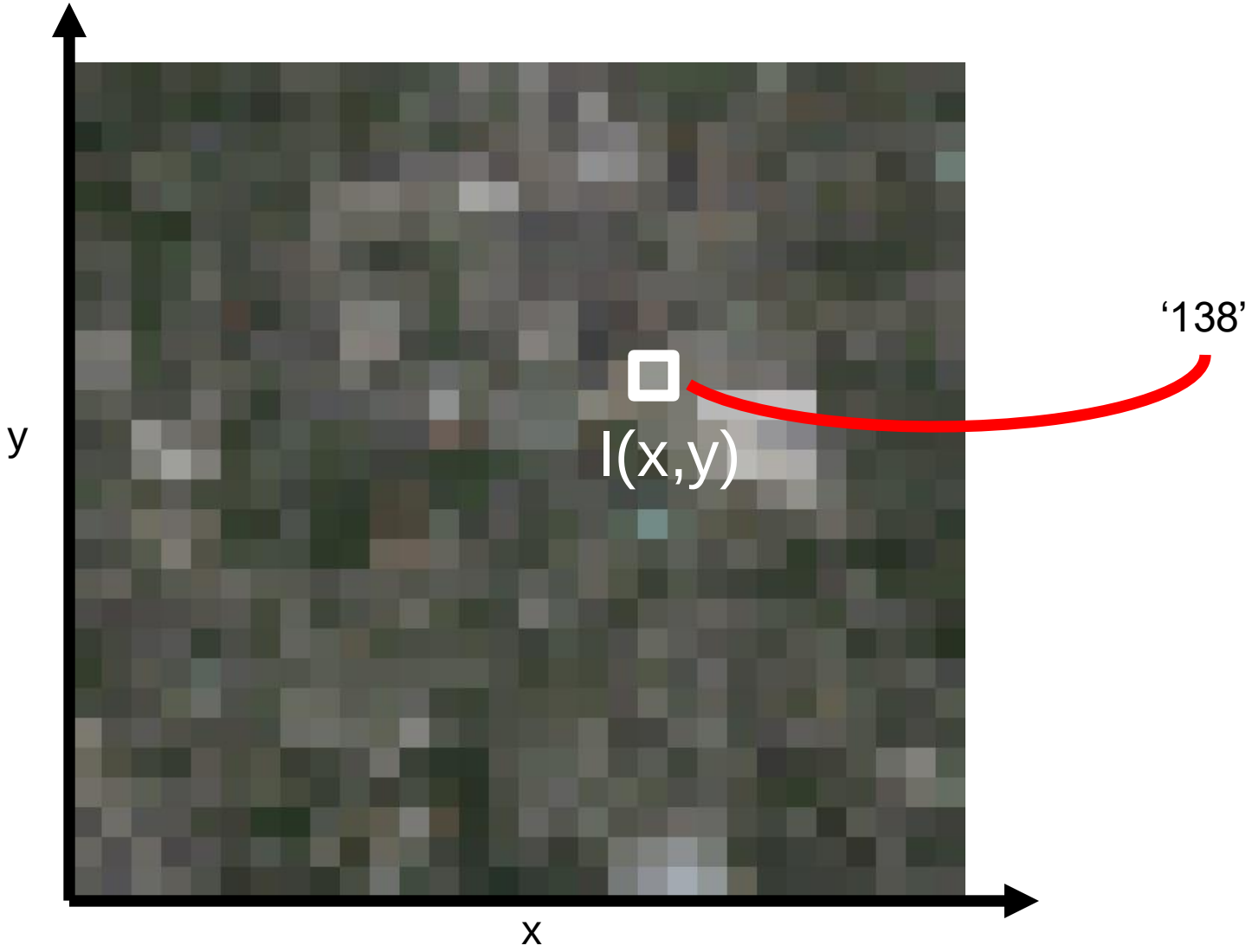


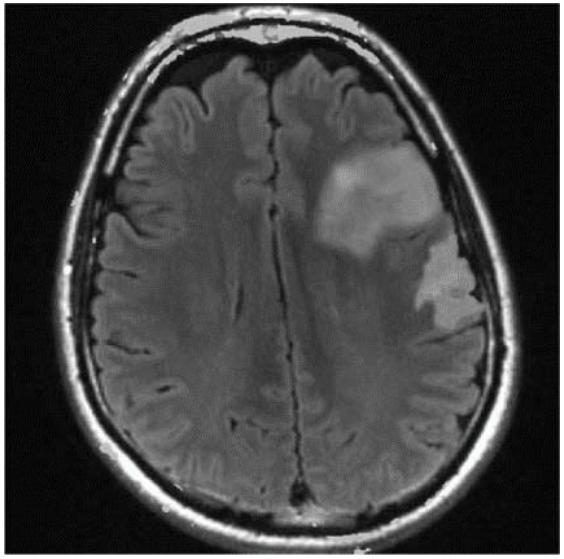
Image as a 2D sampling of signal

- Signal: function depending on some variable with physical meaning.

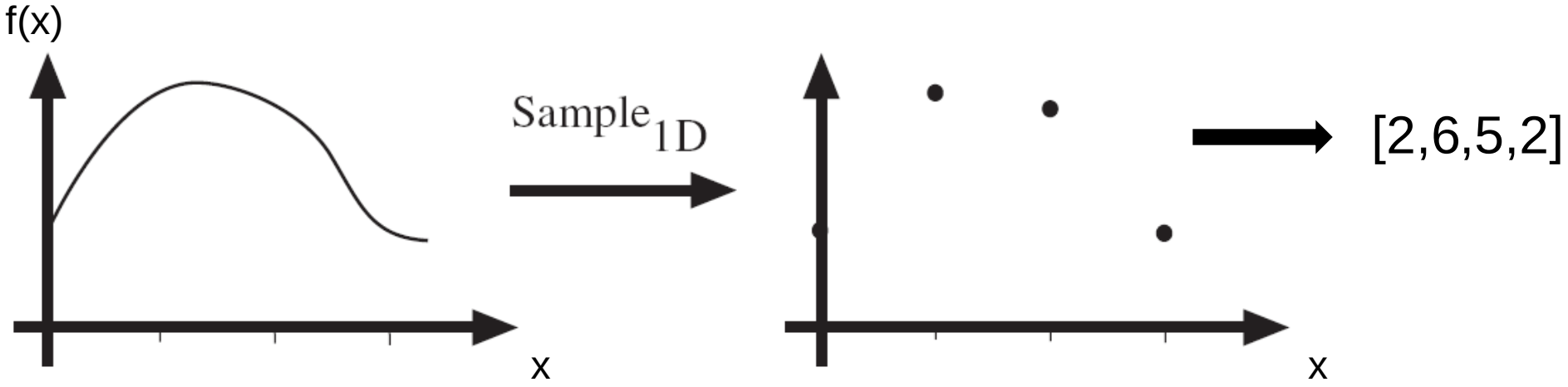
- Image: sampling of that function.
  - 2 variables: xy coordinates
  - 3 variables: xy + time (video)
  - ‘Brightness’ is the value of the function for visible light

- Can be other physical values too: temperature, pressure, depth …

Example 2D Images

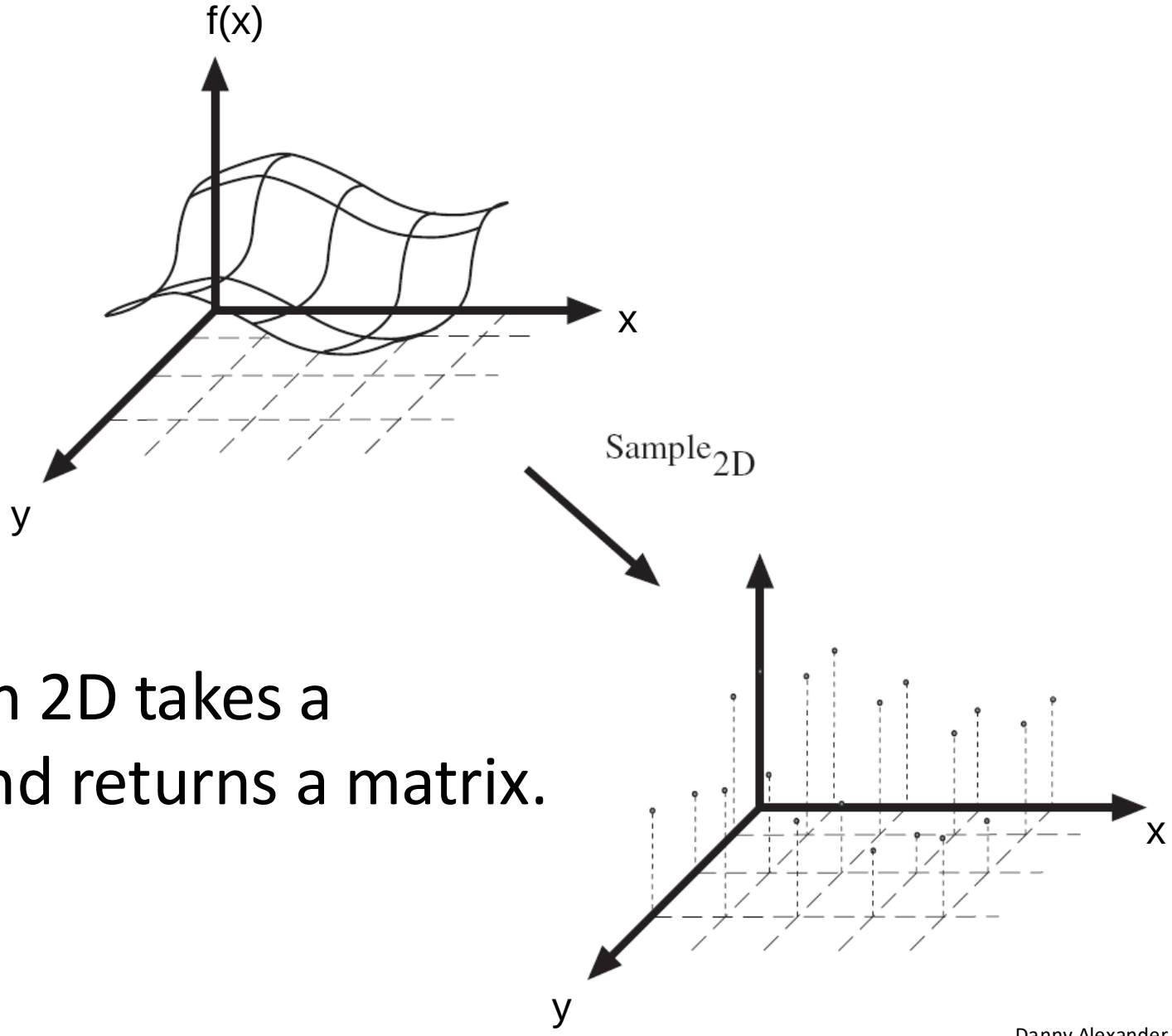


Sampling in 1D



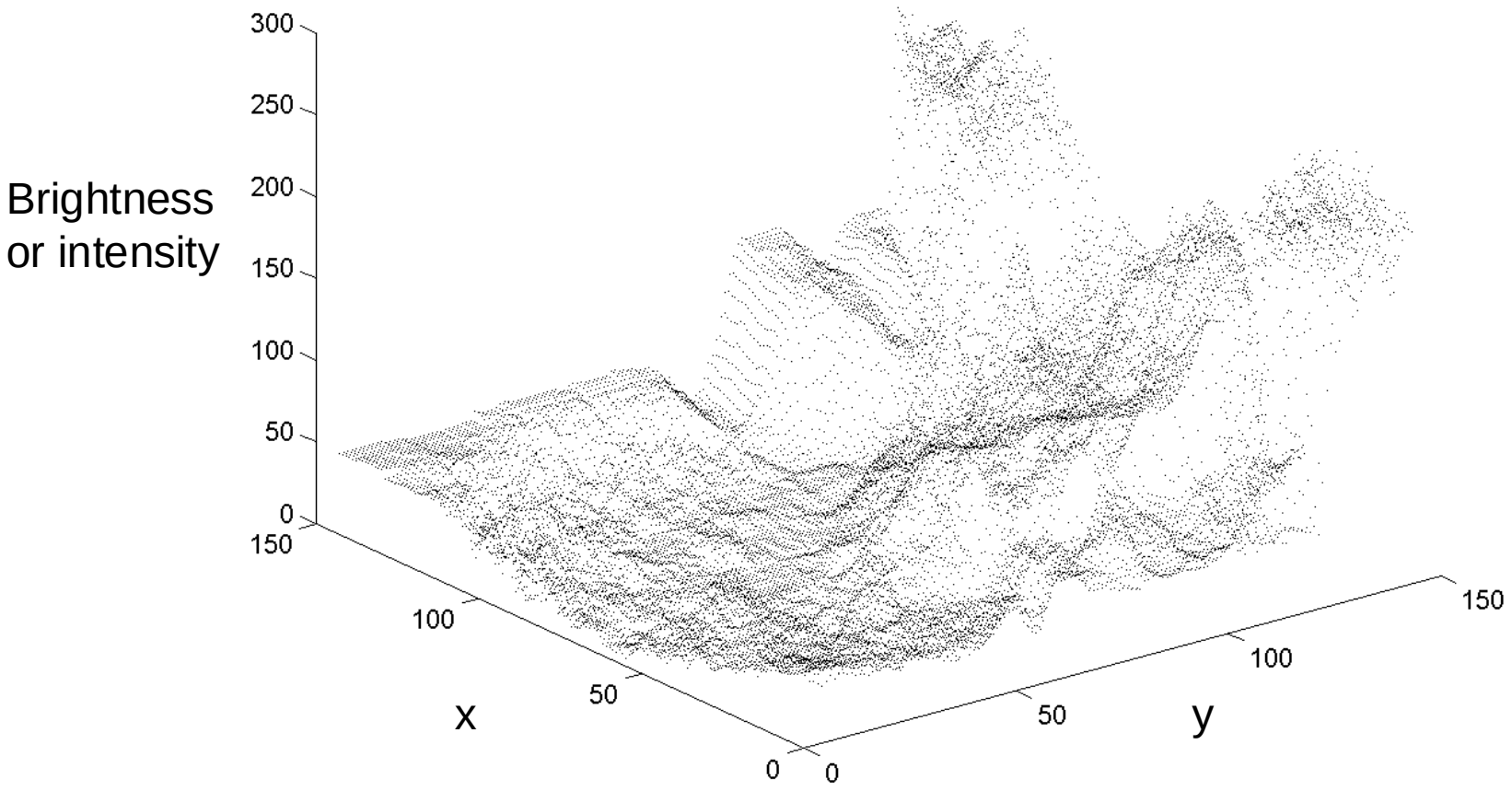
Sampling in 1D takes a function and returns a vector whose elements are values of that function at the sample points.

Sampling in 2D



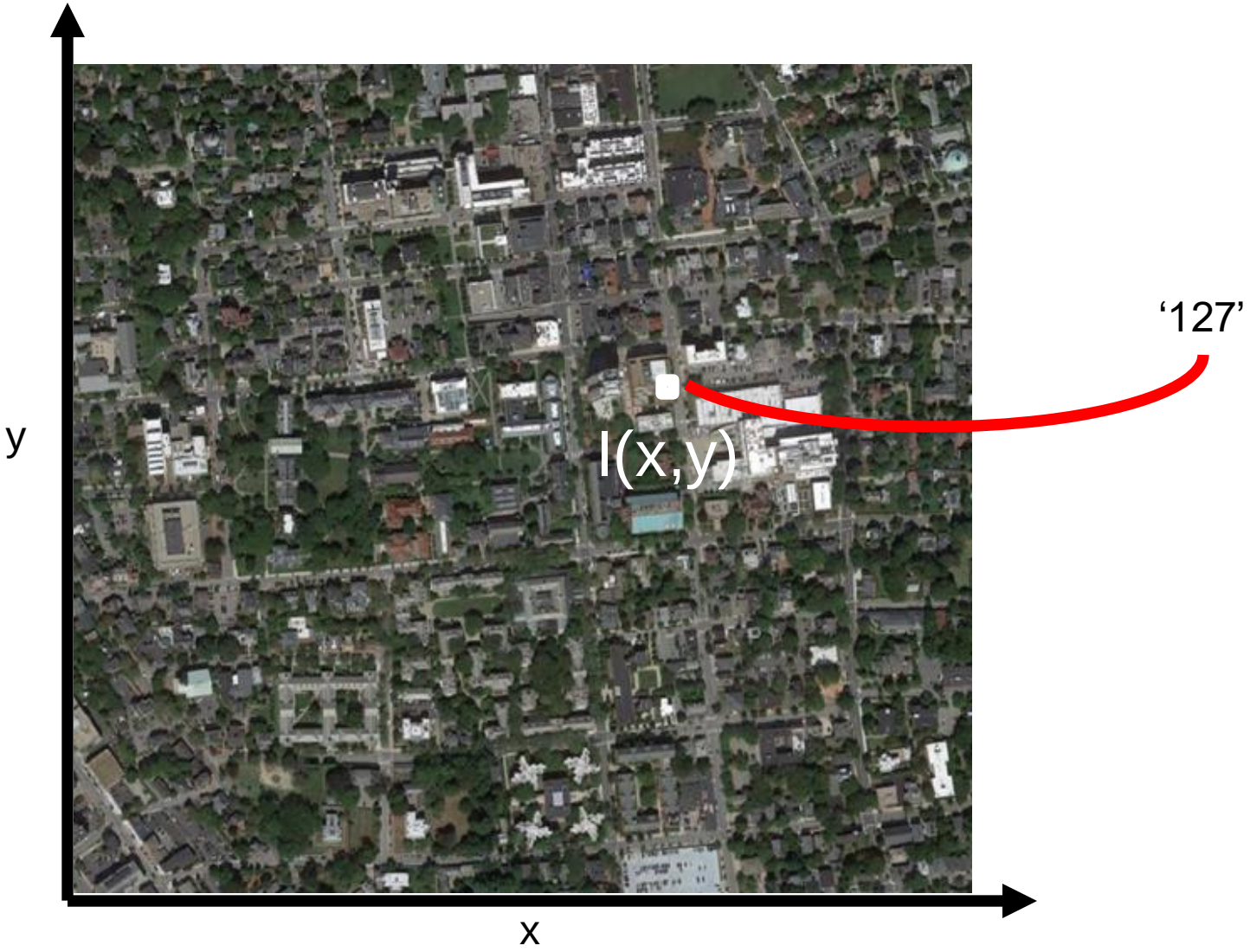
Sampling in 2D takes a function and returns a matrix.

Grayscale Digital Image as ‘Height Map’

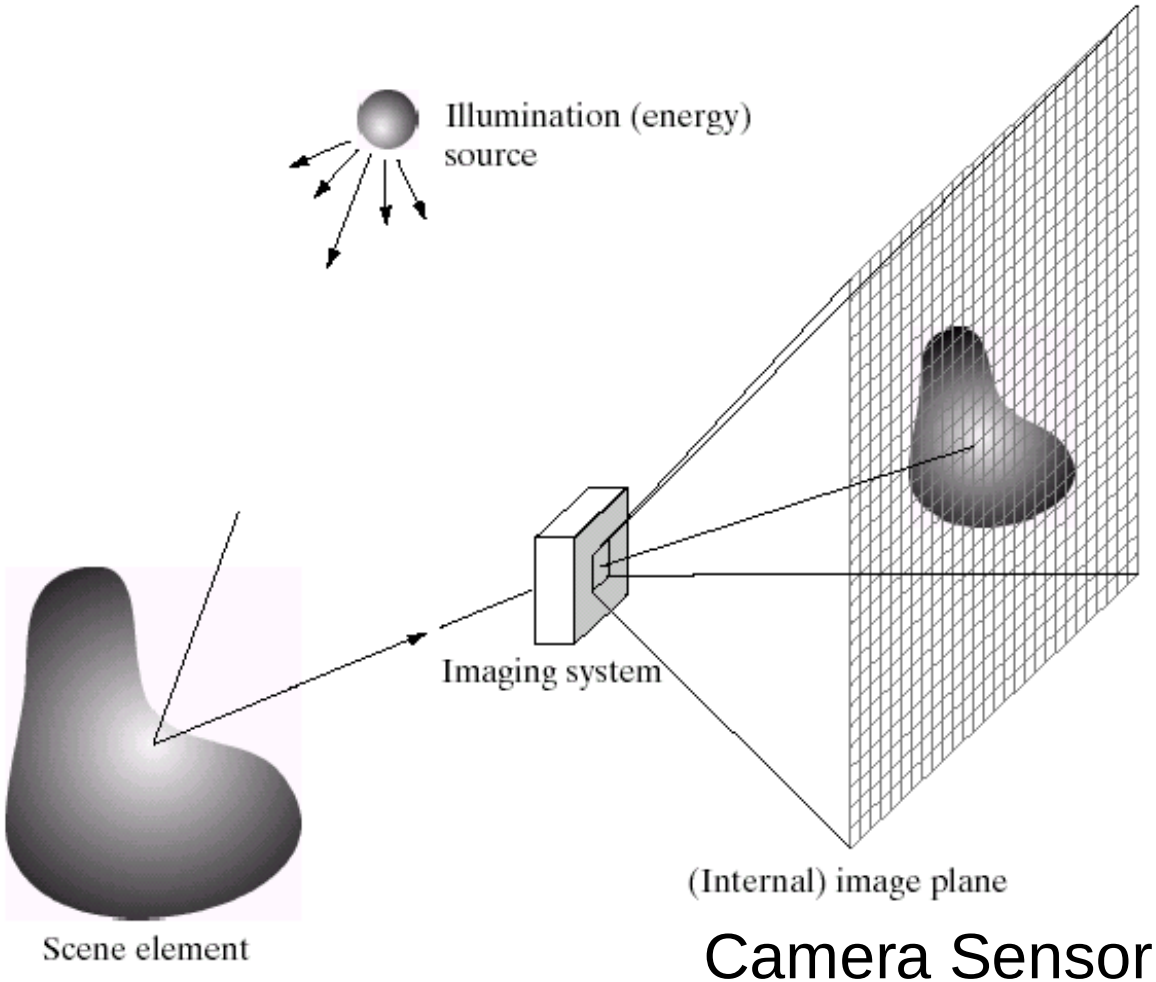


What is each part of a photograph?

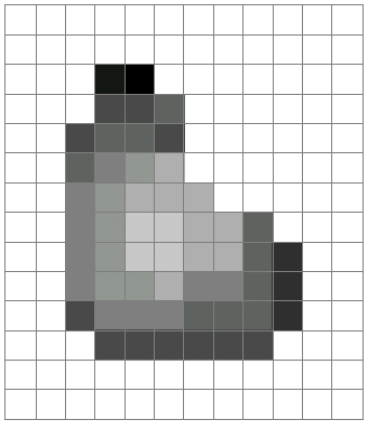
- Pixel -> picture element



Integrating light over a range of angles.

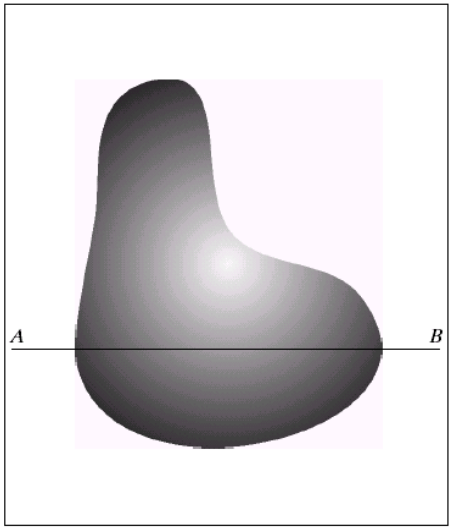


Camera Sensor

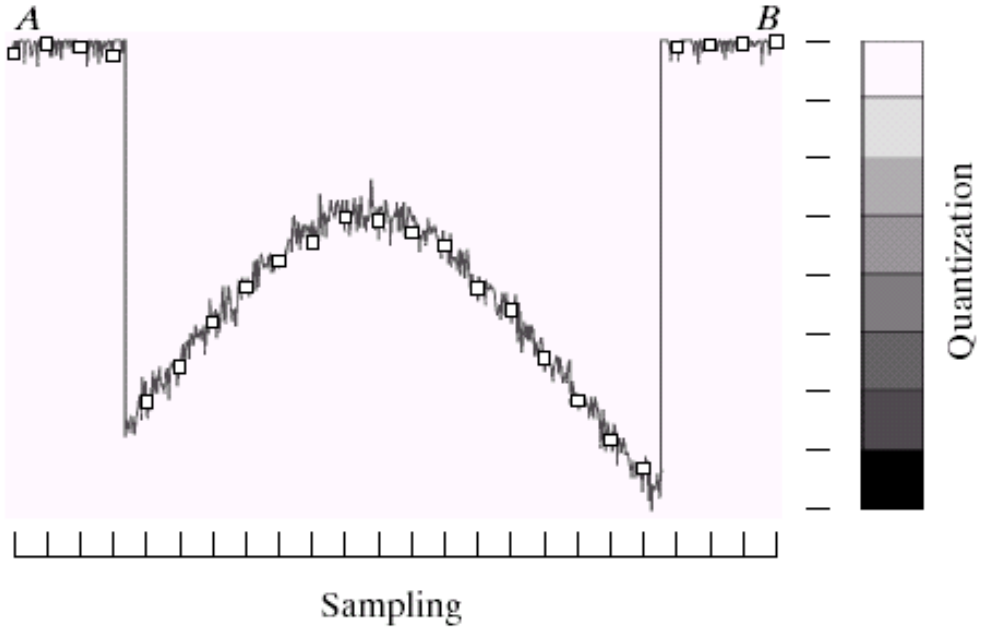


Output Image

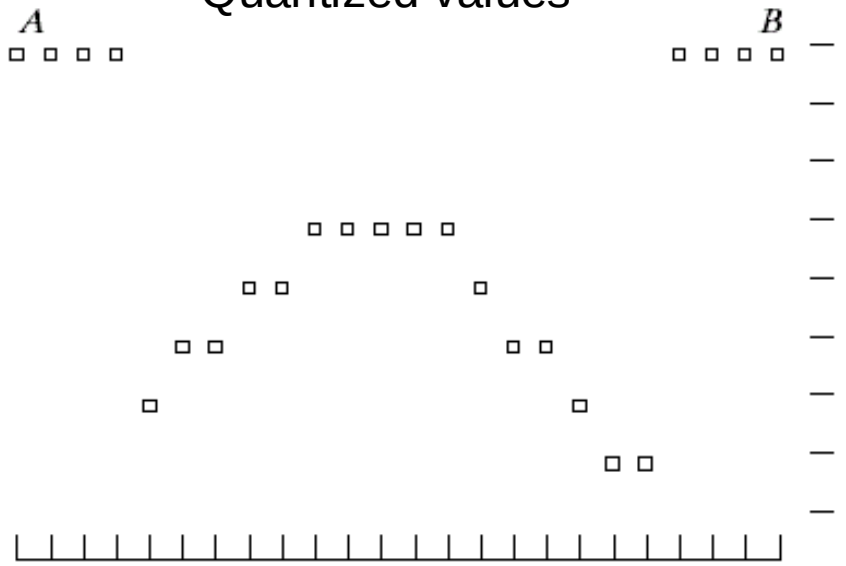
Quantization



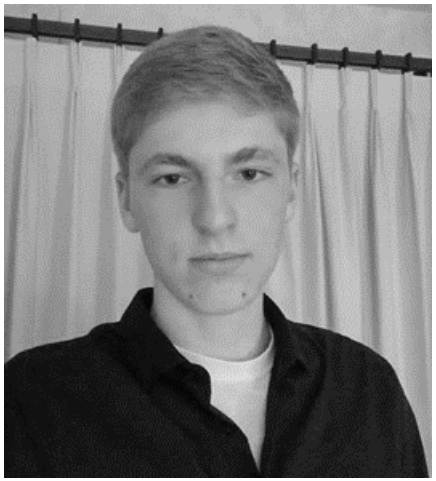
Underlying signal



Quantized values

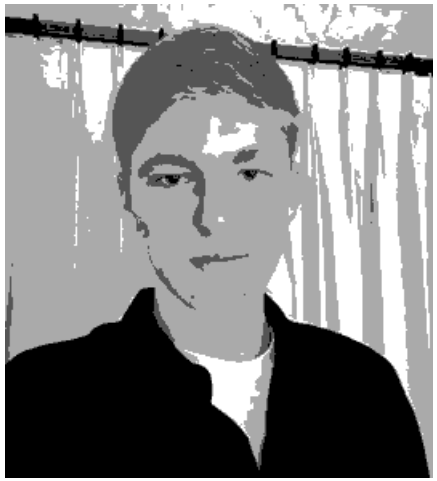


Quantization Effects – Radiometric Resolution



8 bit – 256 levels

4 bit – 16 levels



2 bit – 4 levels

1 bit – 2 levels

We often call this *bit depth.*
For photography, this is also related to *dynamic range.*

Linear shift-invariant image filtering

This week: three views of filtering

- Image filters in spatial domain
  - Filter is a mathematical operation of a grid of numbers
  - Smoothing, sharpening, measuring texture
- Image filters in the frequency domain
  - Filtering is a way to modify the frequencies of images
  - Denoising, sampling, image compression
- Image pyramids
  - Scale-space representation allows coarse-to-fine operations

Linear shift-invariant image filtering

- Replace each pixel by a *linear* combination of its neighbors (and possibly itself).
- The combination is determined by the filter's *kernel*.
- The same kernel is *shifted* to all pixel locations so that all pixels use the same linear combination of their neighbors.

Example: the box filter

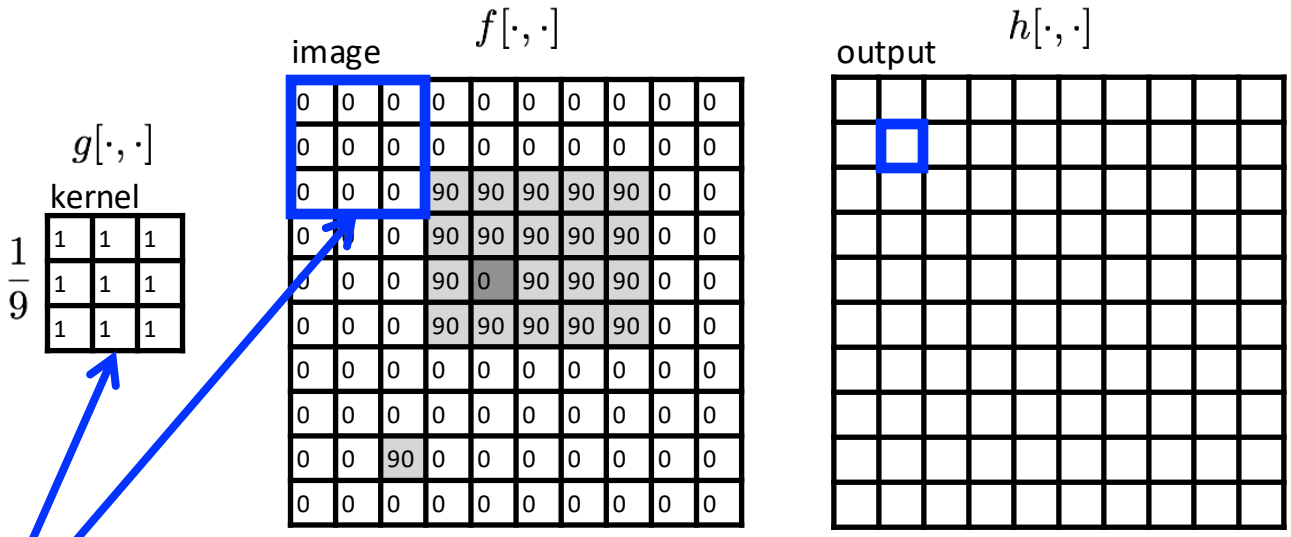
- also known as the 2D rect (not rekt) filter
- also known as the square mean filter

kernel $g[\cdot,\cdot] = \frac{1}{9}$

1	1	1
1	1	1
1	1	1

- replaces pixel with local average
- has smoothing (blurring) effect

Let's run the box filter



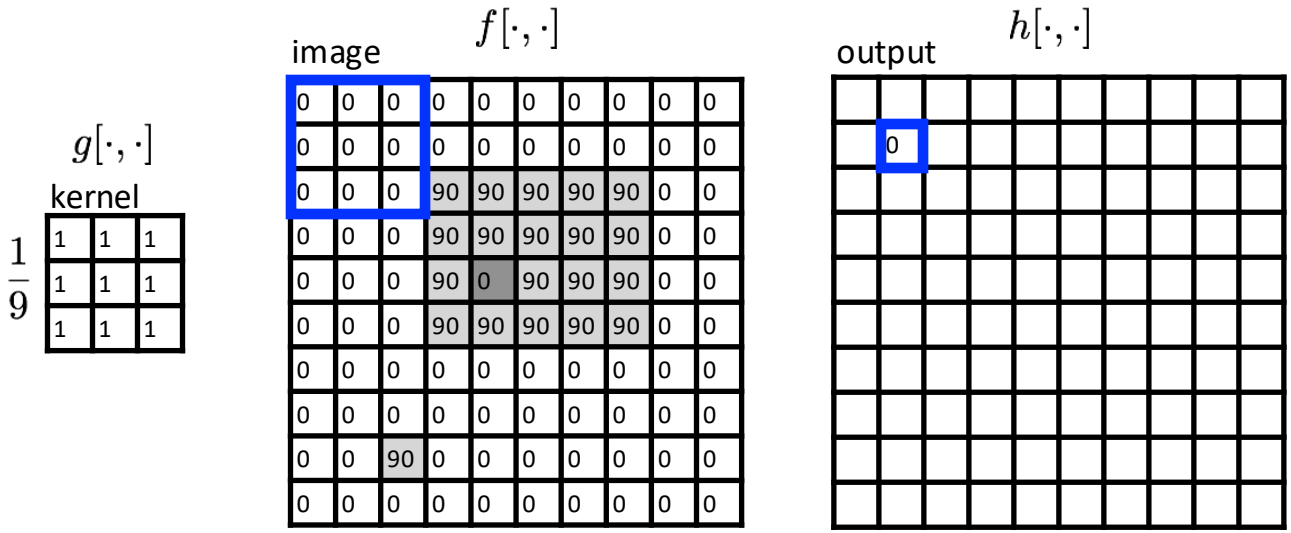
note that we assume that the kernel coordinates are centered

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

output — filter — image (signal)

$$k, l = [-1, 0, 1]$$

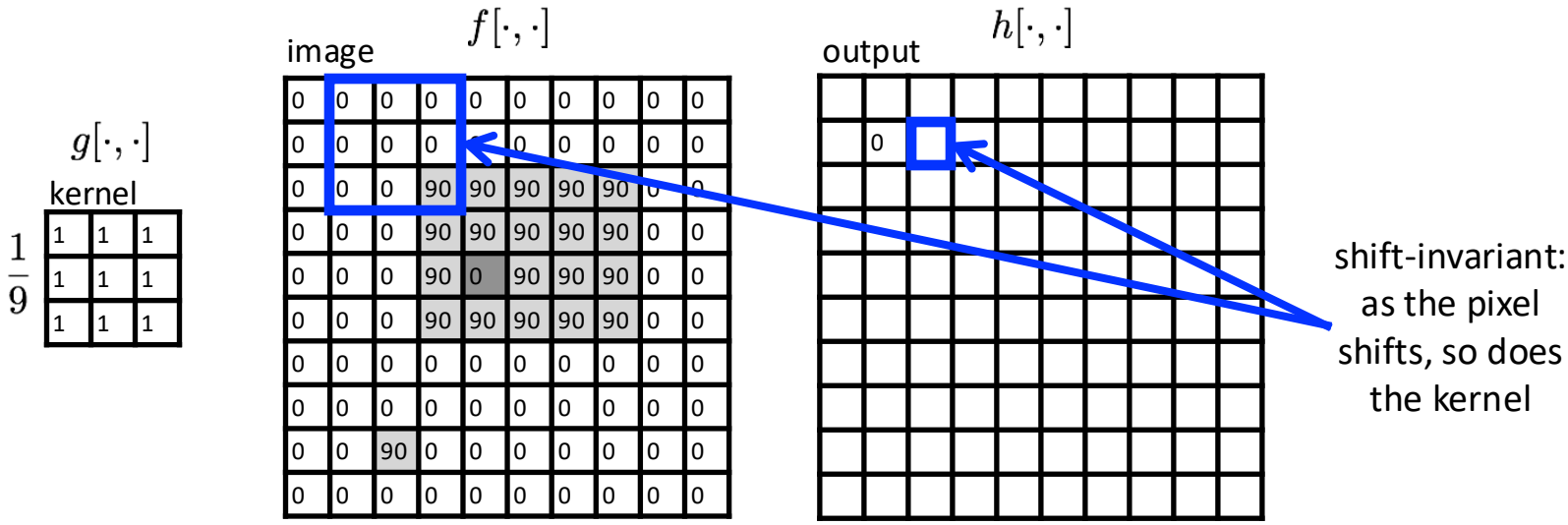
Let's run the box filter



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

output filter image (signal)

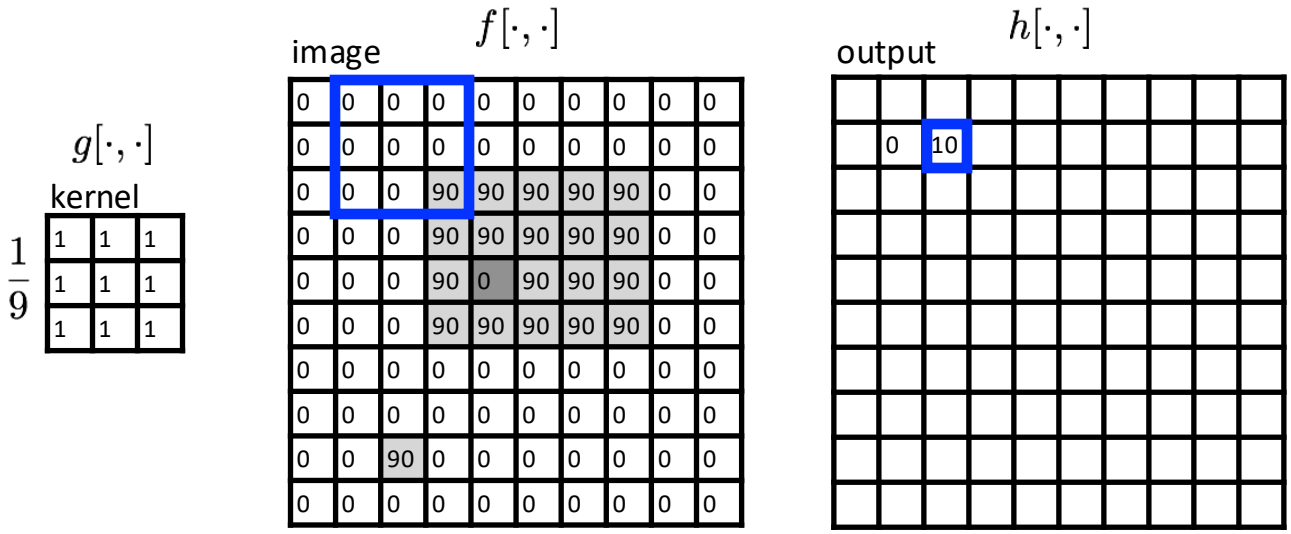
Let's run the box filter



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

output filter image (signal)

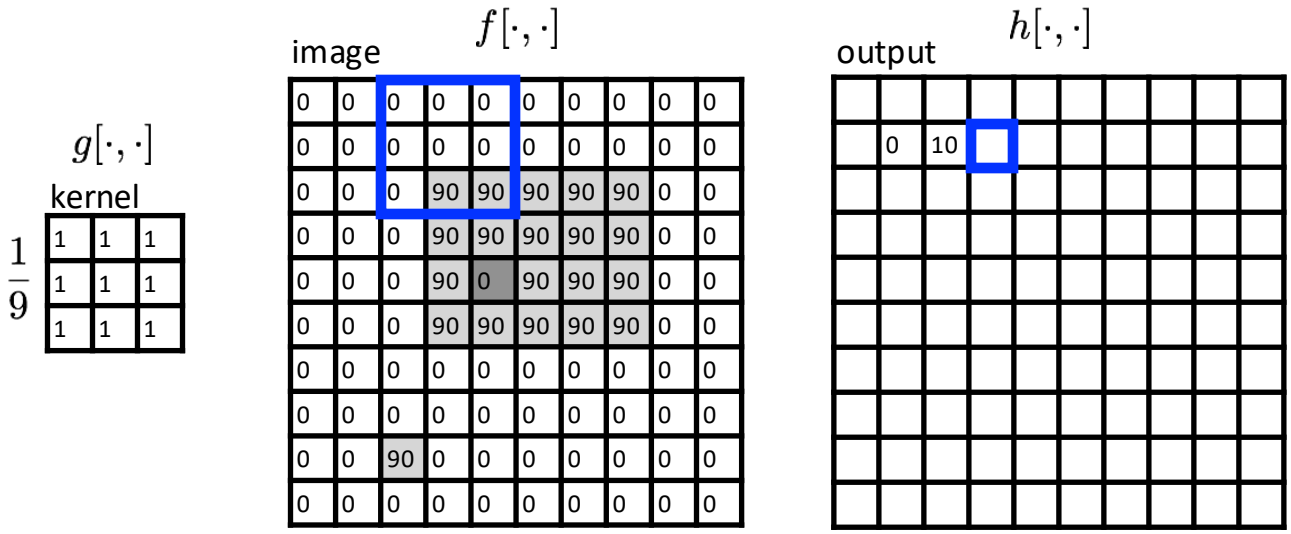
Let's run the box filter



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

output — filter — image (signal)

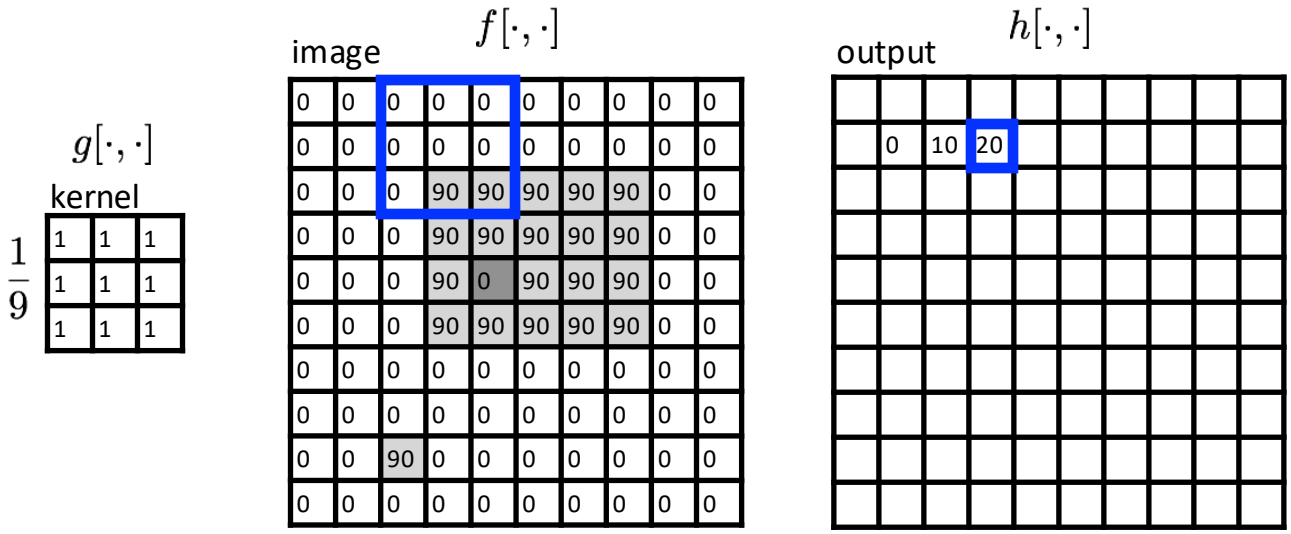
Let's run the box filter



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

output filter image (signal)

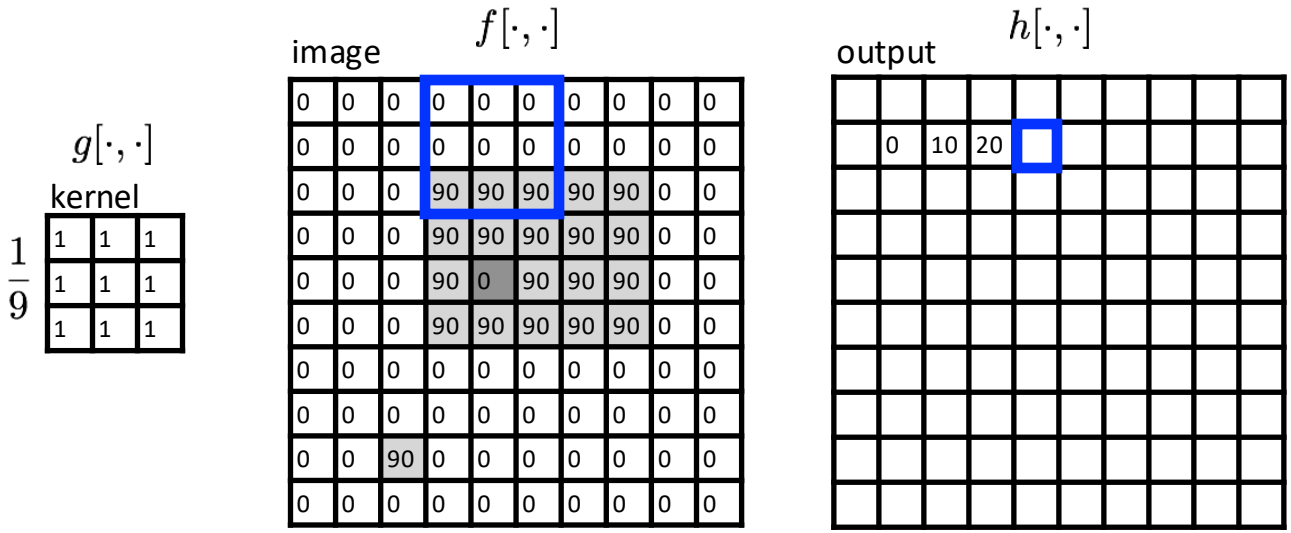
Let's run the box filter



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

output filter image (signal)

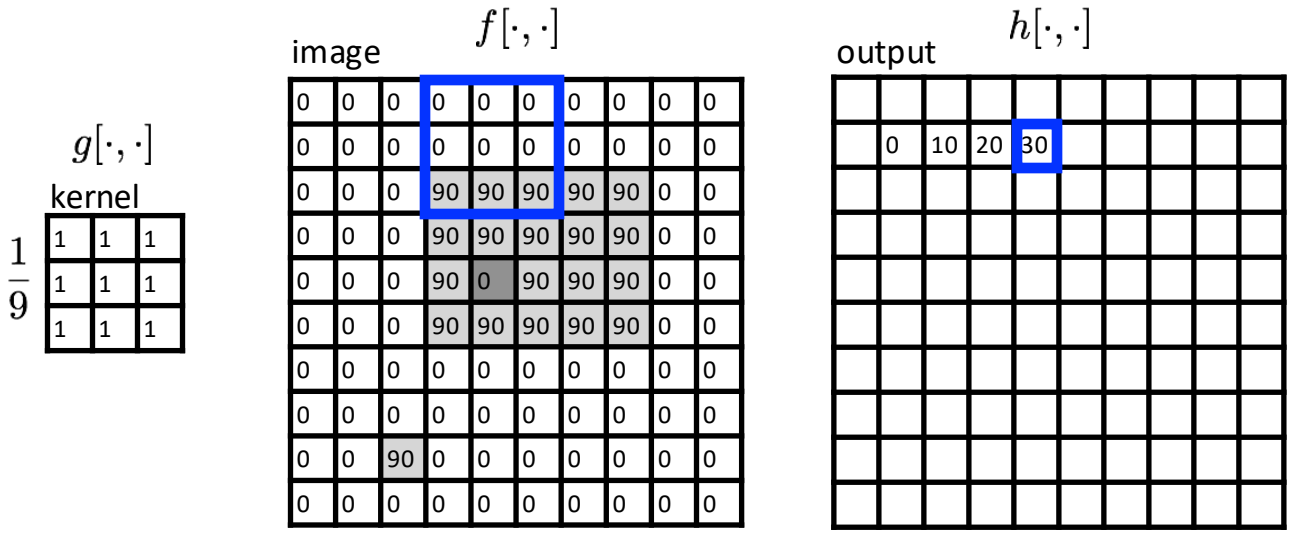
Let's run the box filter



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

output filter image (signal)

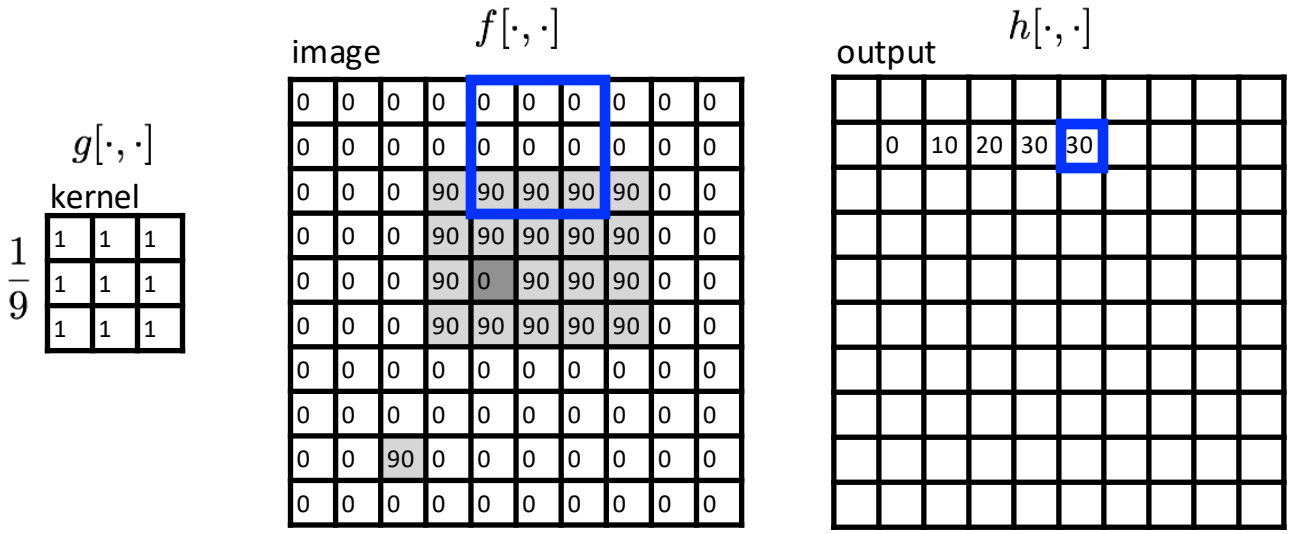
Let's run the box filter



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

output filter image (signal)

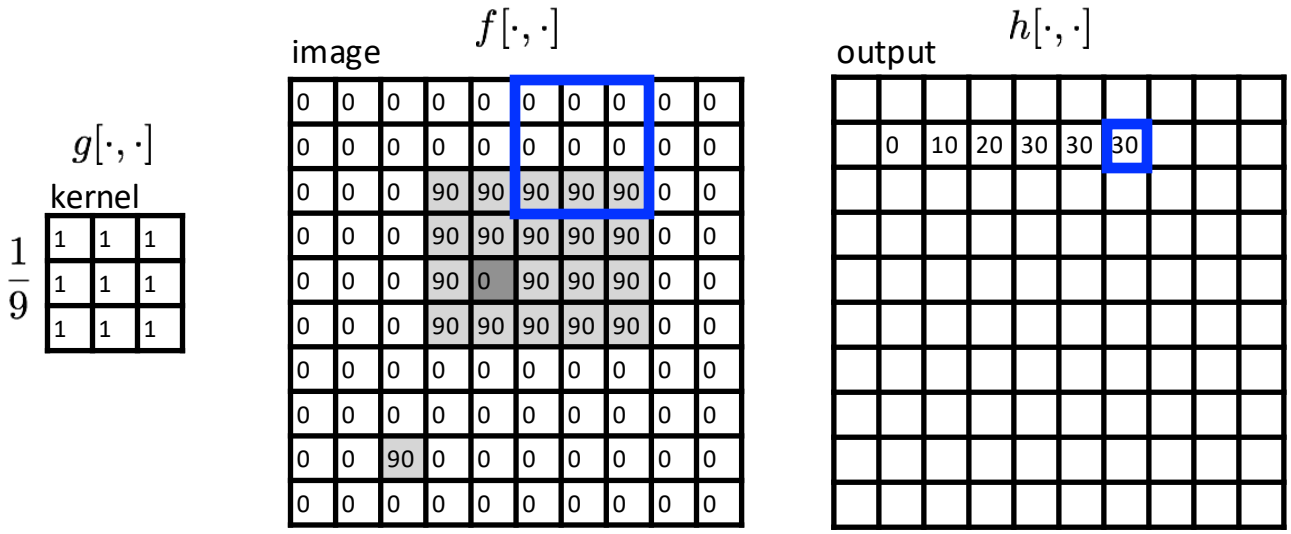
Let's run the box filter



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

output filter image (signal)

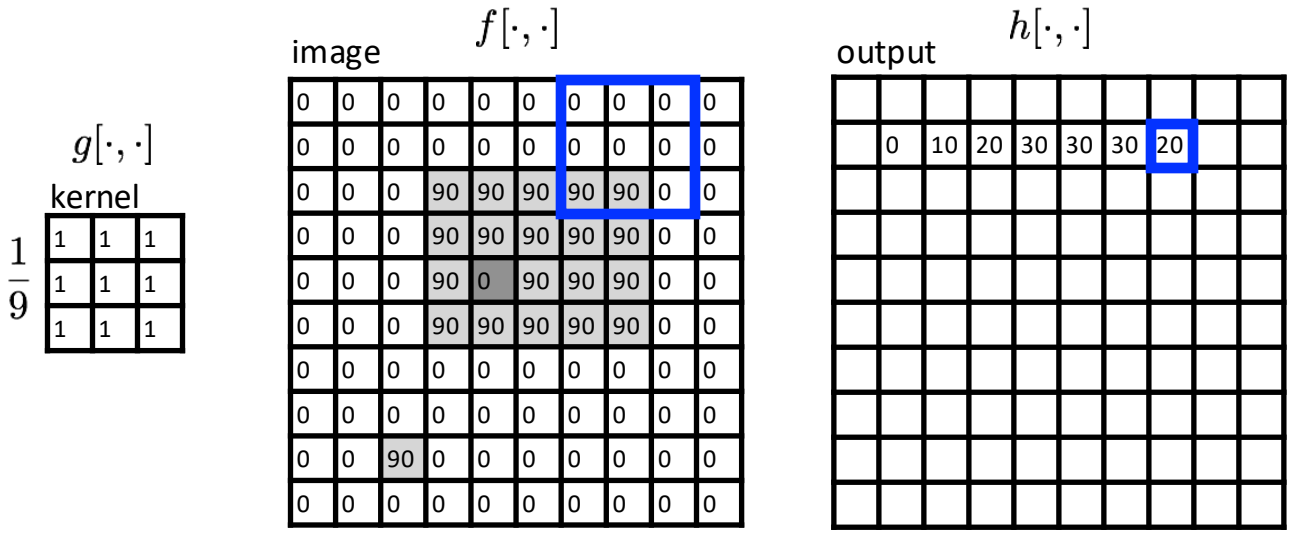
Let's run the box filter



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

output filter image (signal)

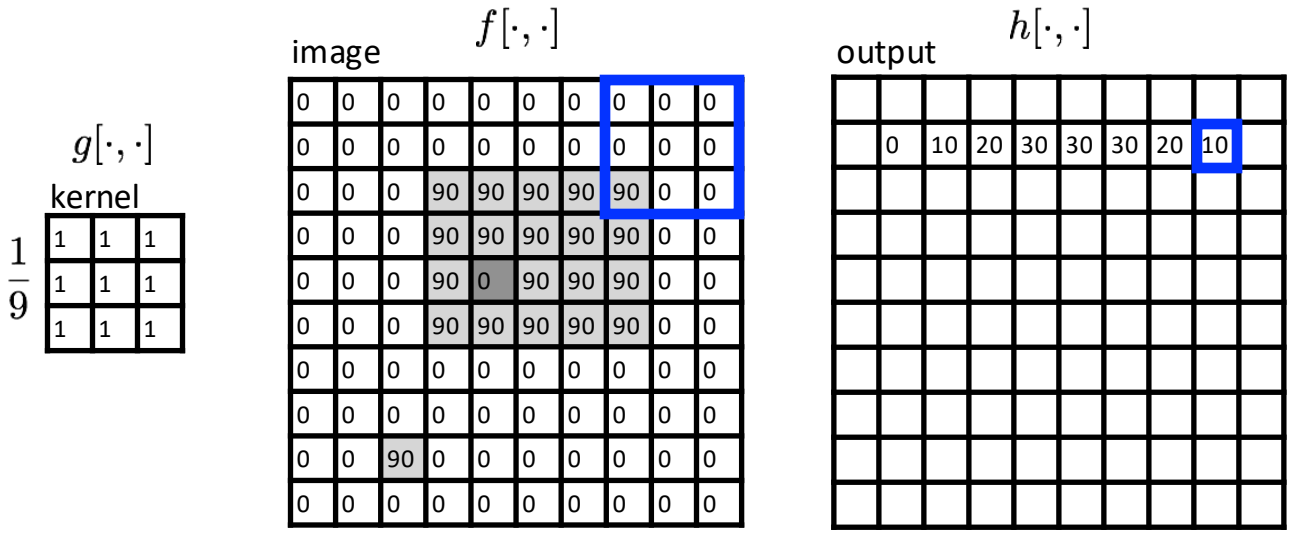
Let's run the box filter



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

output filter image (signal)

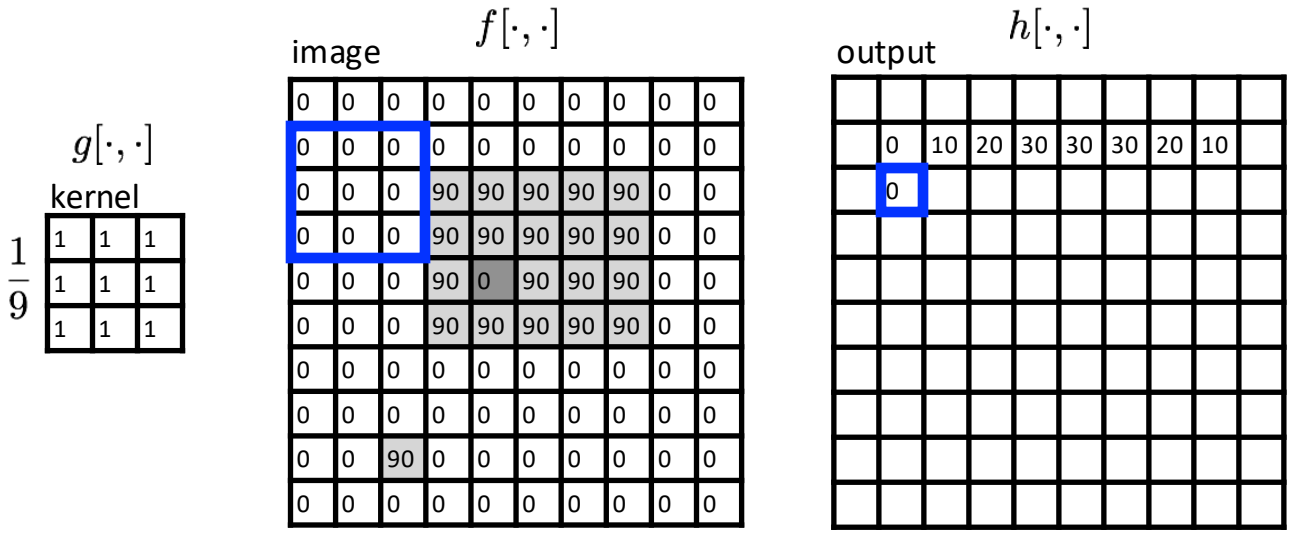
Let's run the box filter



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

output filter image (signal)

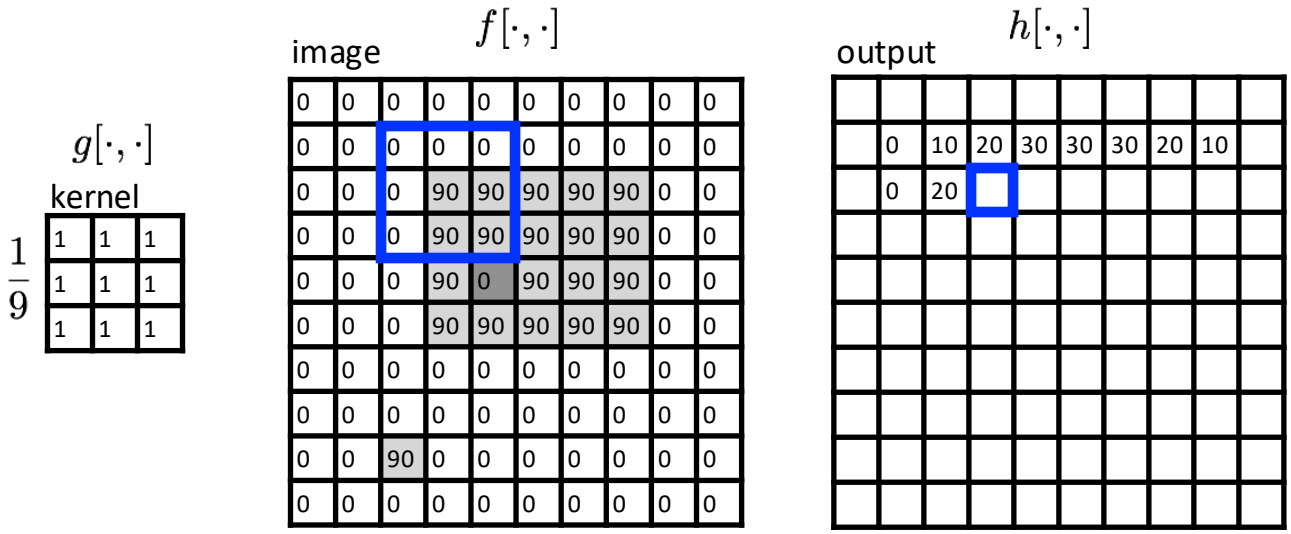
Let's run the box filter



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

output filter image (signal)

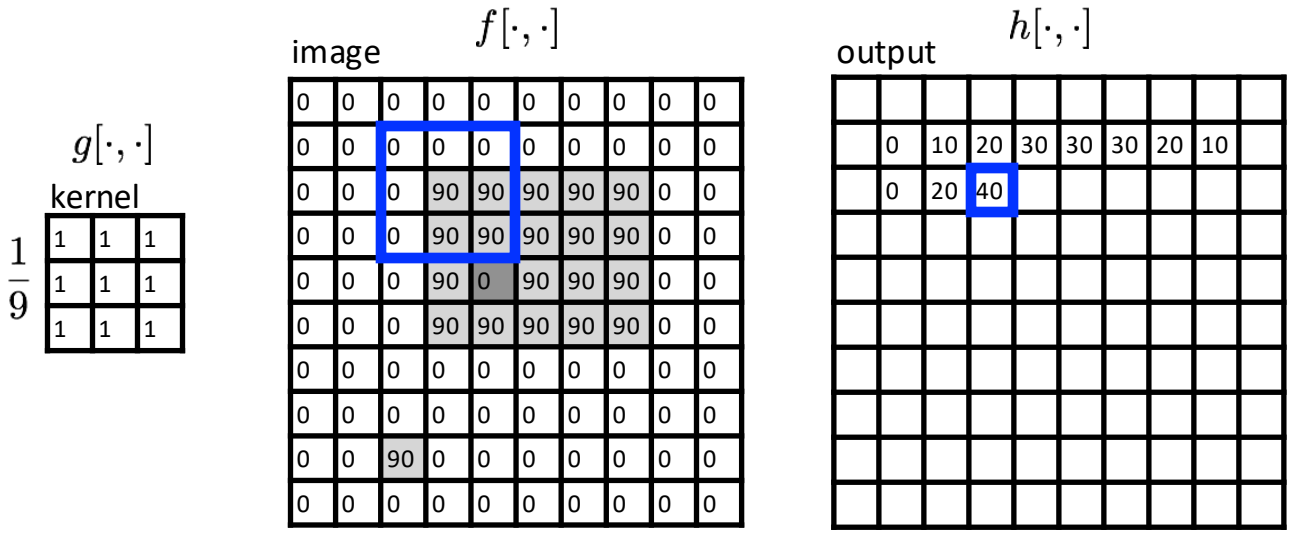
Let's run the box filter



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

output filter image (signal)

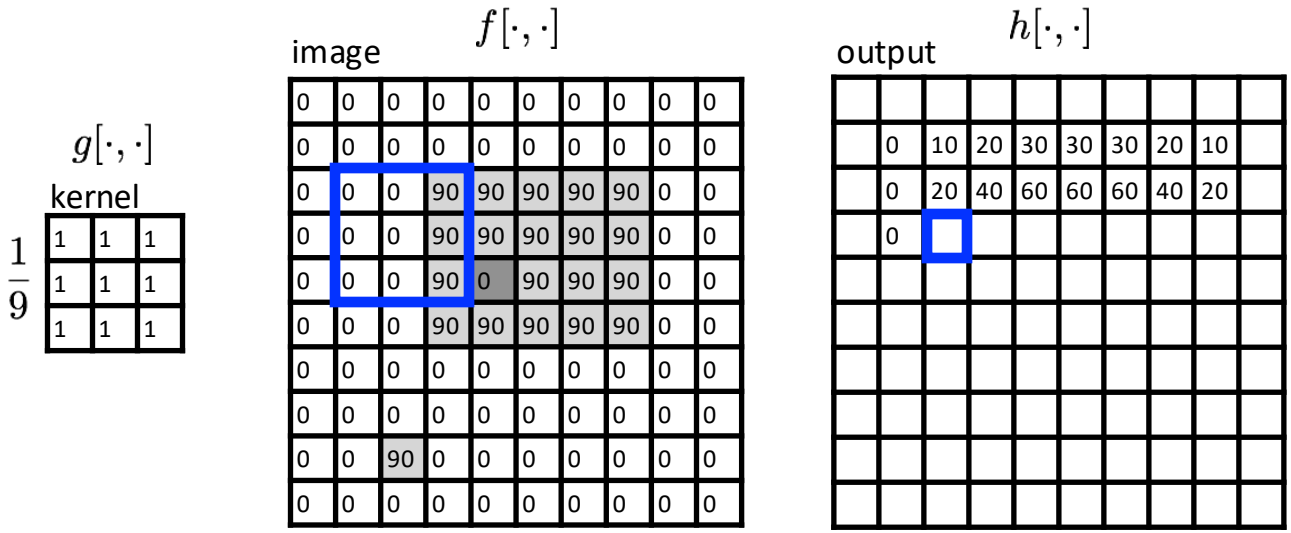
Let's run the box filter



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

output filter image (signal)

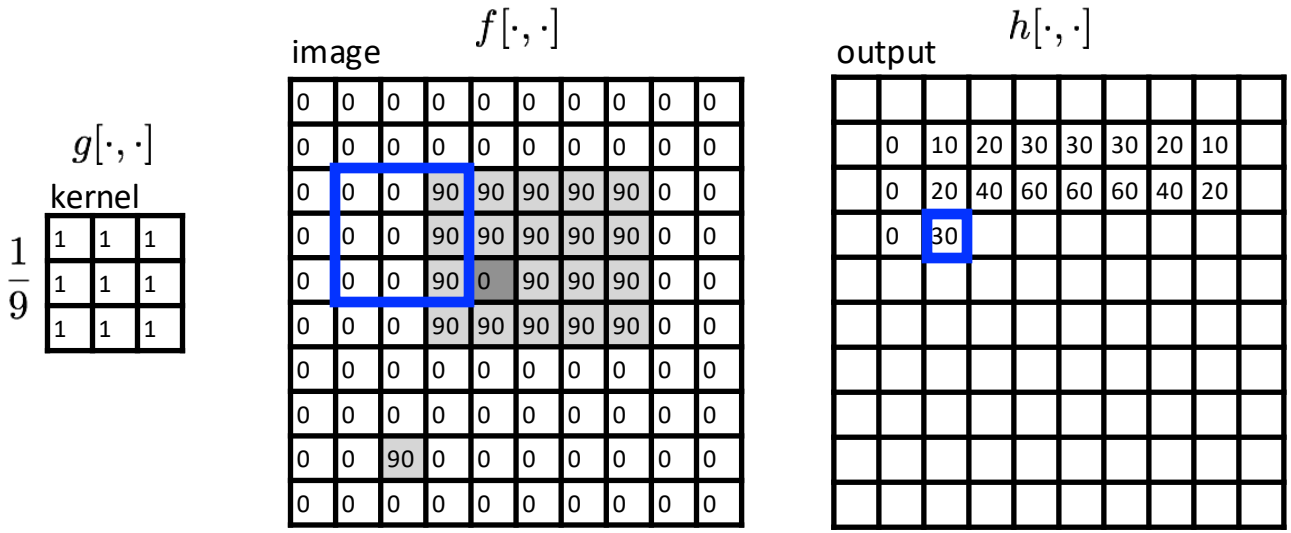
Let's run the box filter



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

output filter image (signal)

Let's run the box filter



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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

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

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

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

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

output filter image (signal)

… and the result is

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

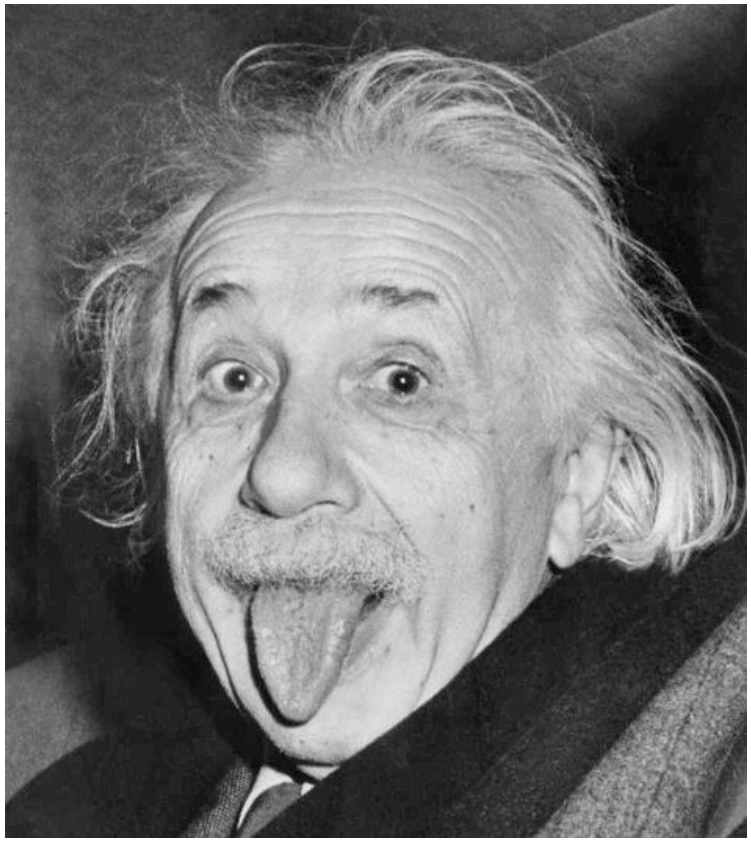
output

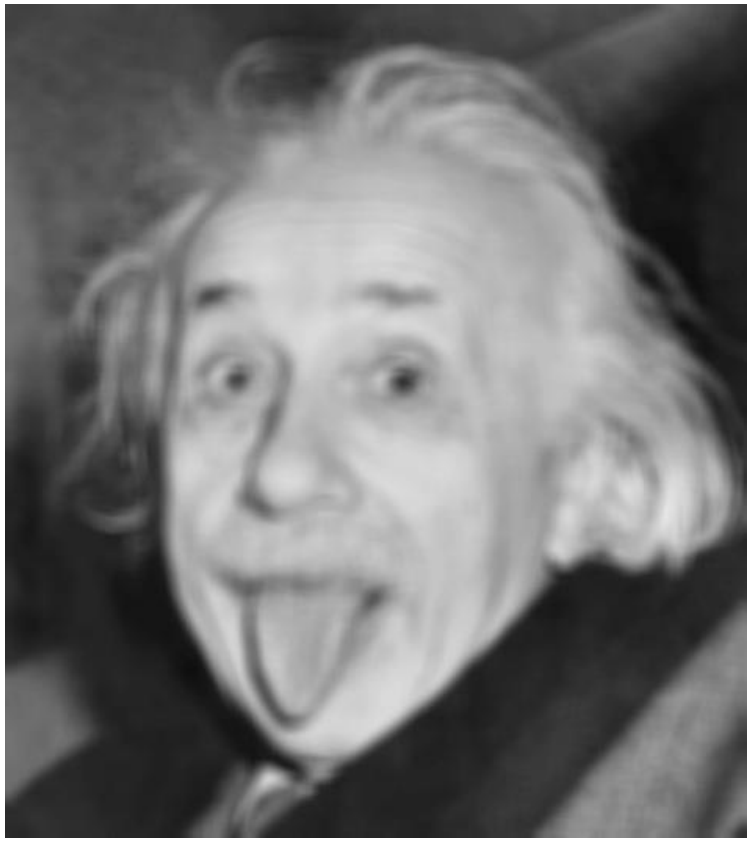
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	10	10	10	0	0	0	0	

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

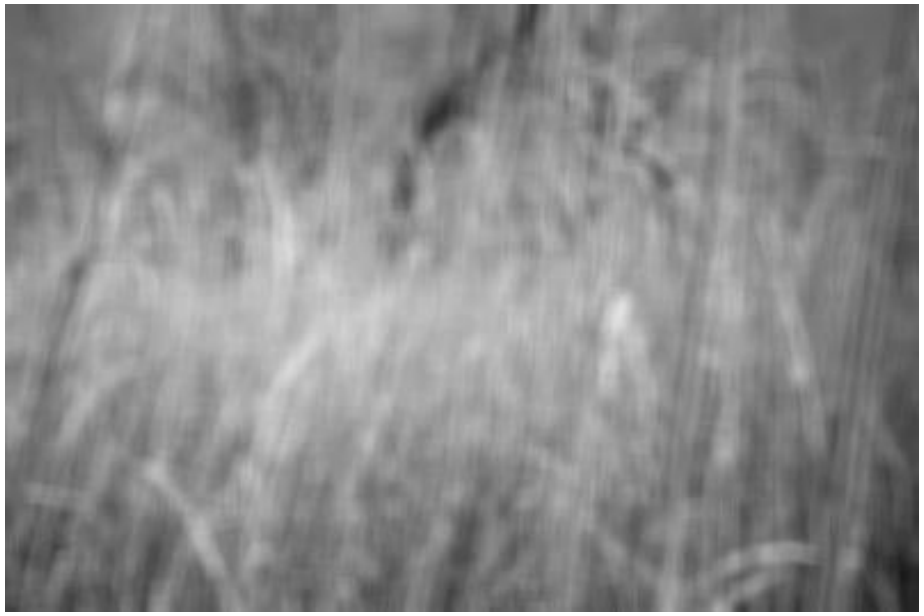
output filter image (signal)

Some more realistic examples





Some more realistic examples



Some more realistic examples

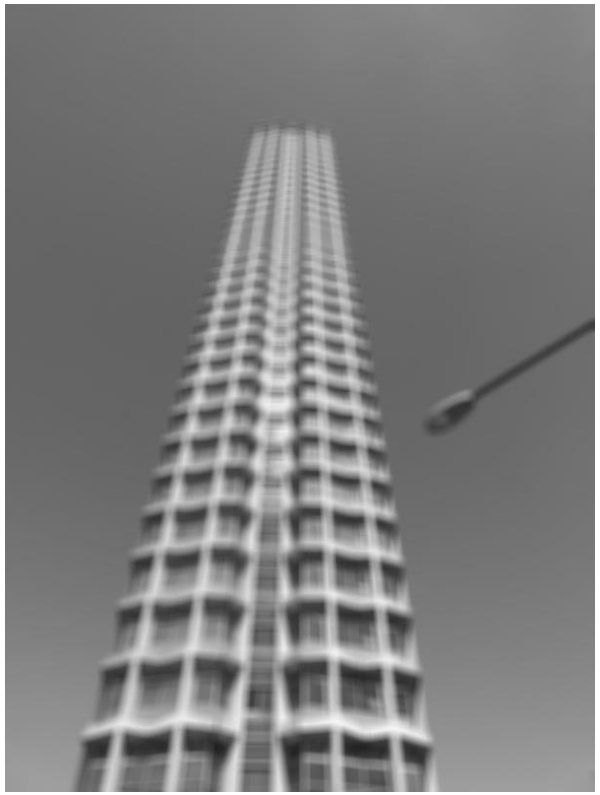


Image filtering

Compute function of local neighborhood at each position:

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

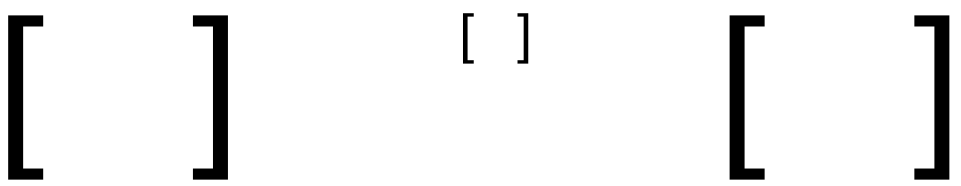
Image filtering

Compute function of local neighborhood at each position:

h=output f=filter I=image

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

2d coords=k,l 2d coords=m,n



Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

- Achieve smoothing effect (remove ‘sharp’ features)

$f[\cdot,\cdot]$

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

Slide credit: David Lowe (UBC)

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove ‘sharp’ features)
- Why does it sum to one?

$f[\cdot,\cdot]$

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

Slide credit: David Lowe (UBC)

Smoothing with box filter

David Lowe

$$f[\cdot,\cdot]$$

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

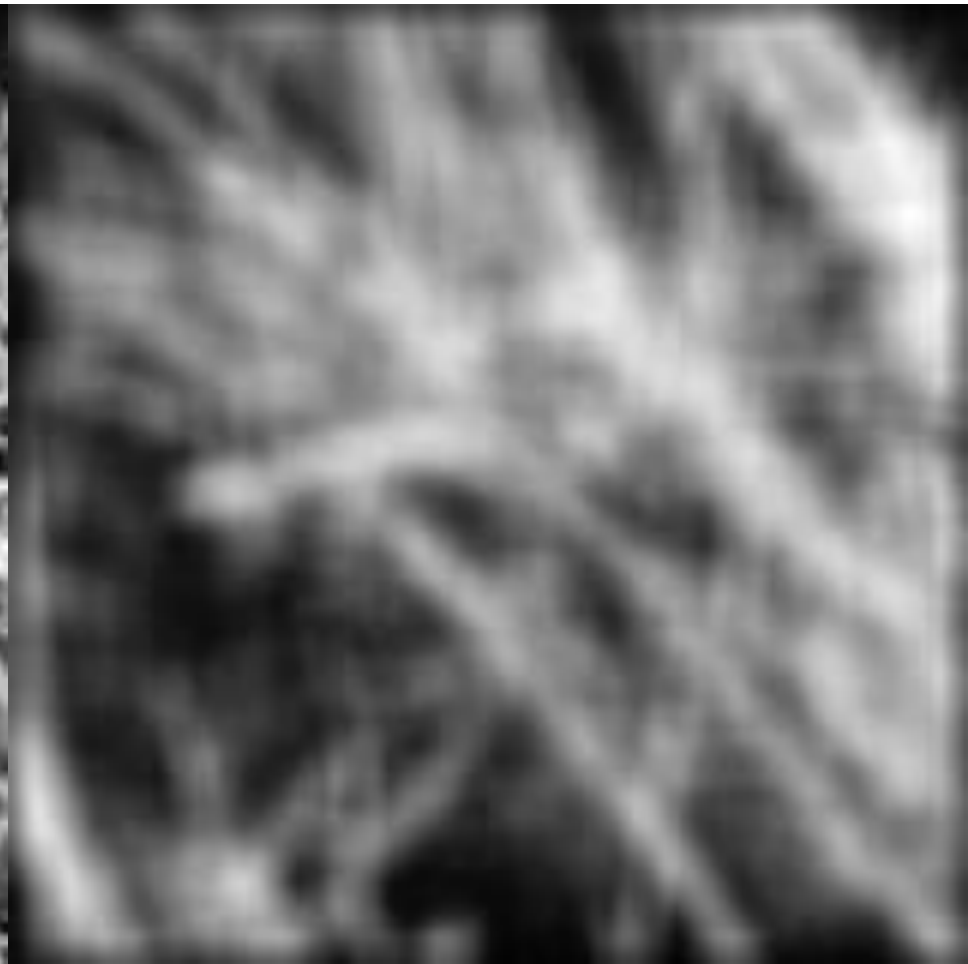


Image filtering

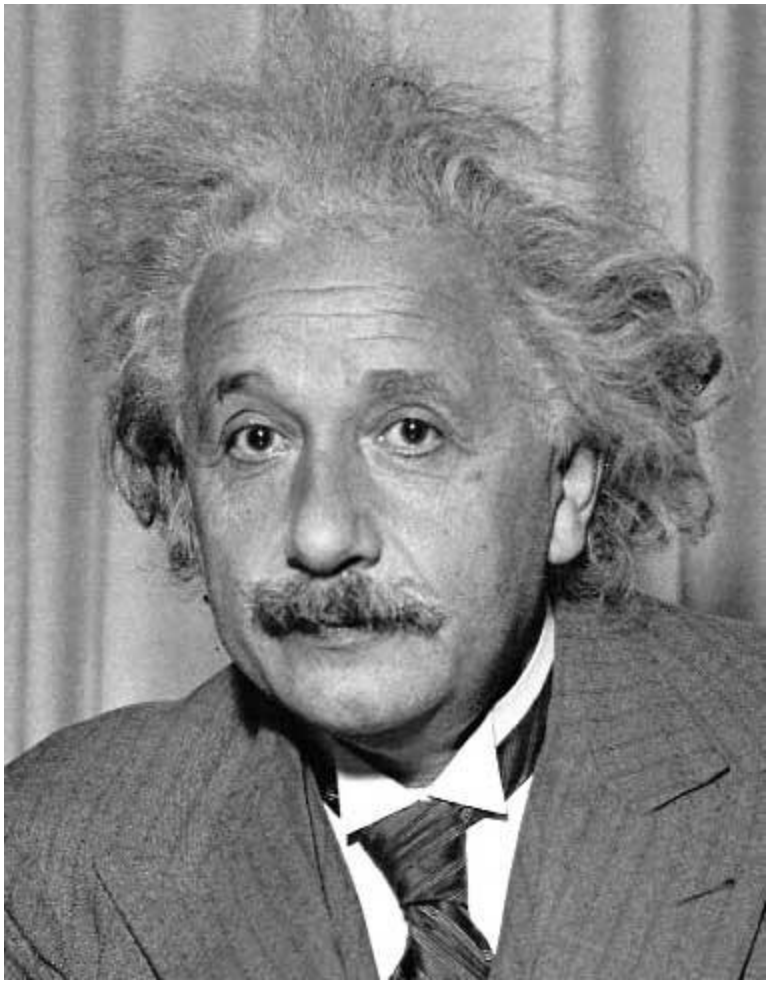
Compute function of local neighborhood at each position:

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

Really important!

- Enhance images
  - Denoise, resize, increase contrast, etc.
- Extract information from images
  - Texture, edges, distinctive points, etc.
- Detect patterns
  - Template matching

Filtering practice



1.

0	0	0
0	1	0
0	0	0

2.

0	0	0
0	0	1
0	0	0

3.

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

4.

0	0	0
0	2	0
0	0	0

$-\frac{1}{9}$

1	1	1
1	1	1
1	1	1

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

1. Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

2. Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

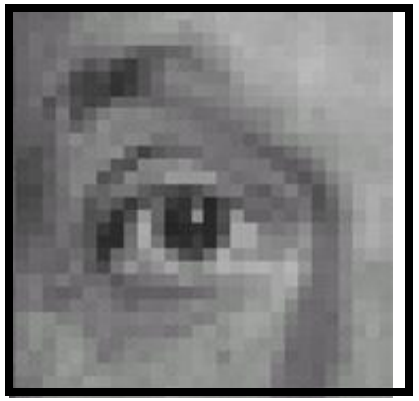
?

Source: D. Lowe

2. Practice with linear filters

Original

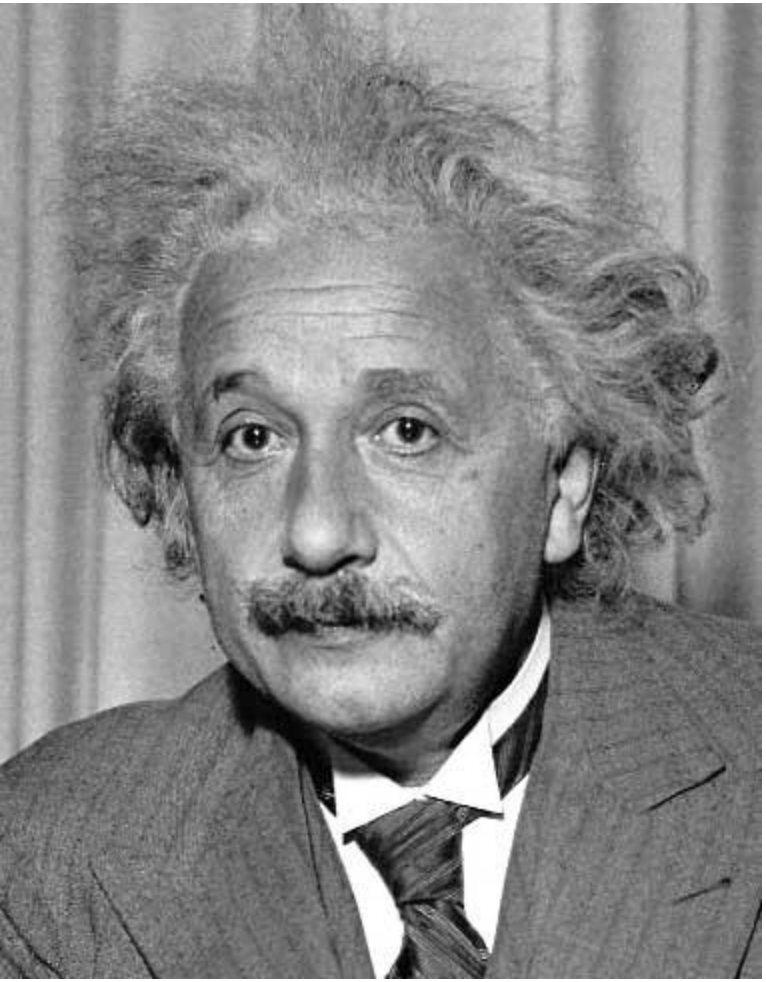
0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

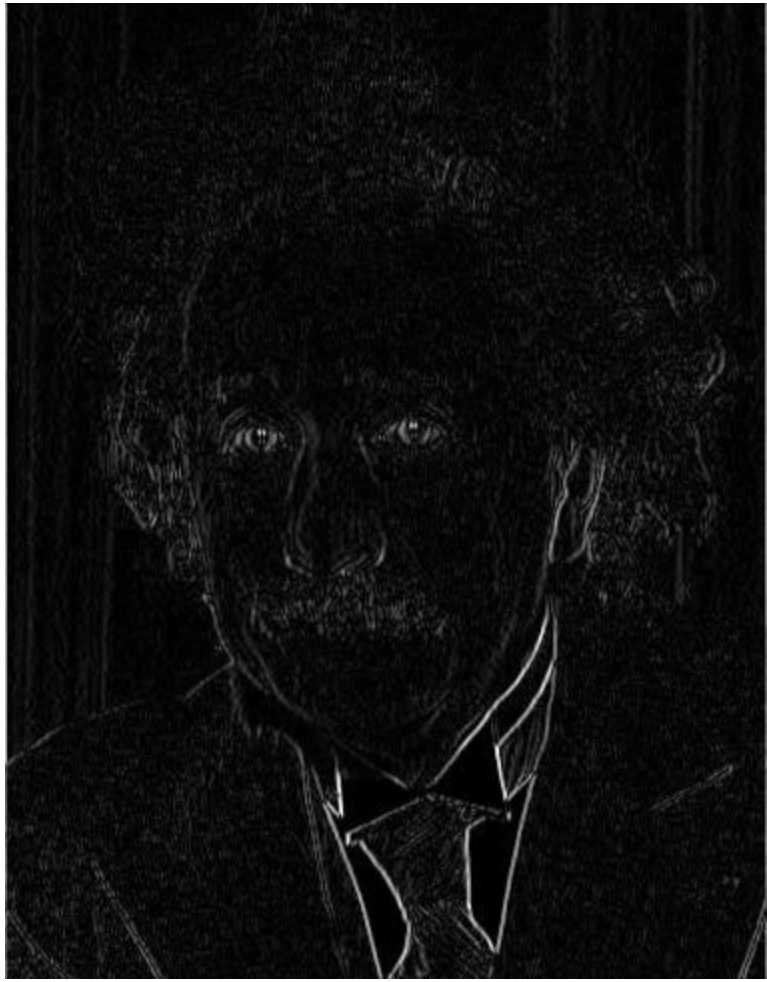
Source: D. Lowe

3. Practice with linear filters



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

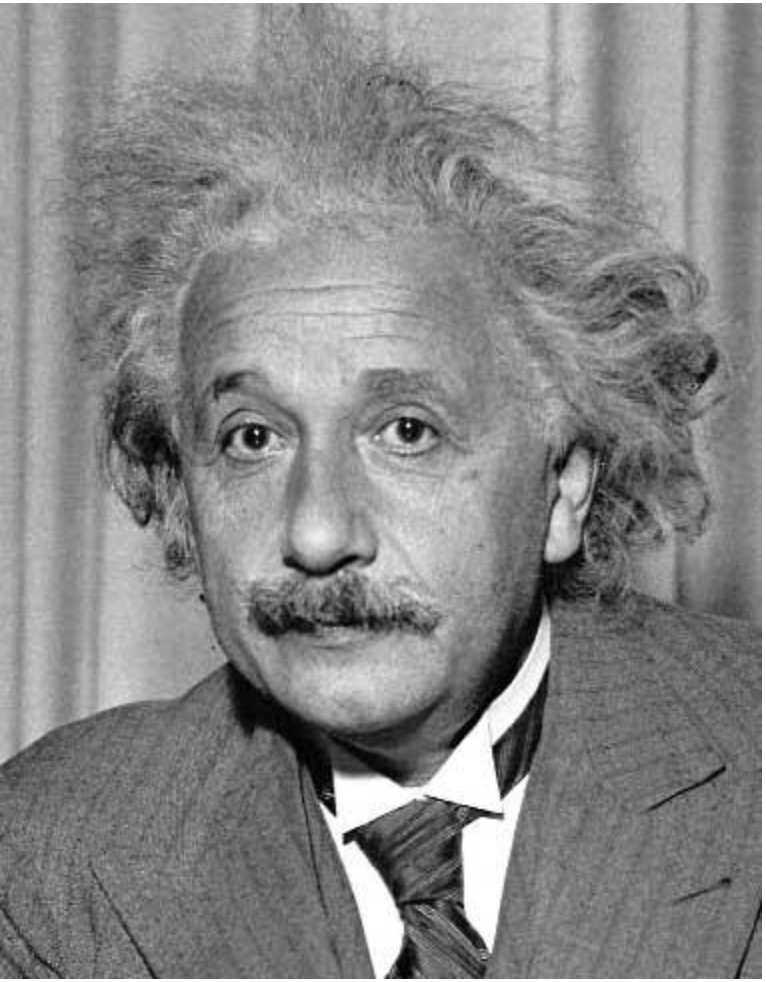
Sobel



Vertical Edge
(absolute value)

David Lowe

3. Practice with linear filters



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

Sobel

Horizontal Edge
(absolute value)

David Lowe

4. Practice with linear filters

Original

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

(Note that filter sums to 1)

Source: D. Lowe

4. Practice with linear filters

Original

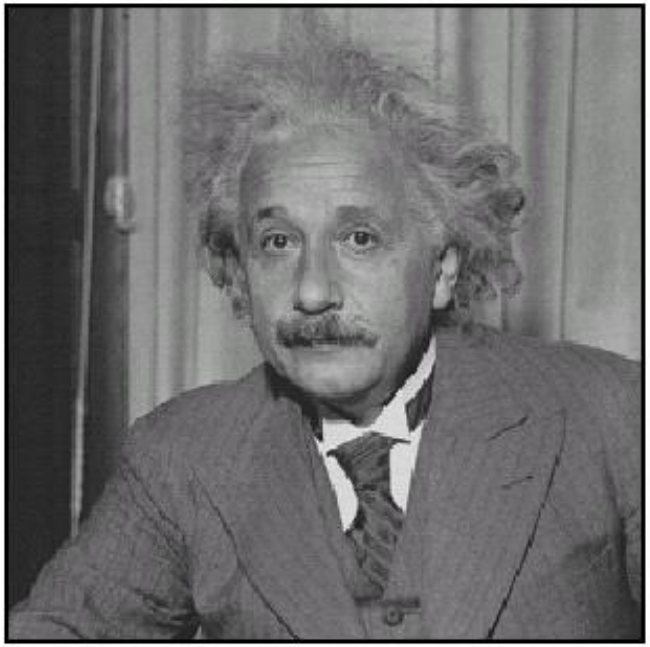
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Sharpening filter

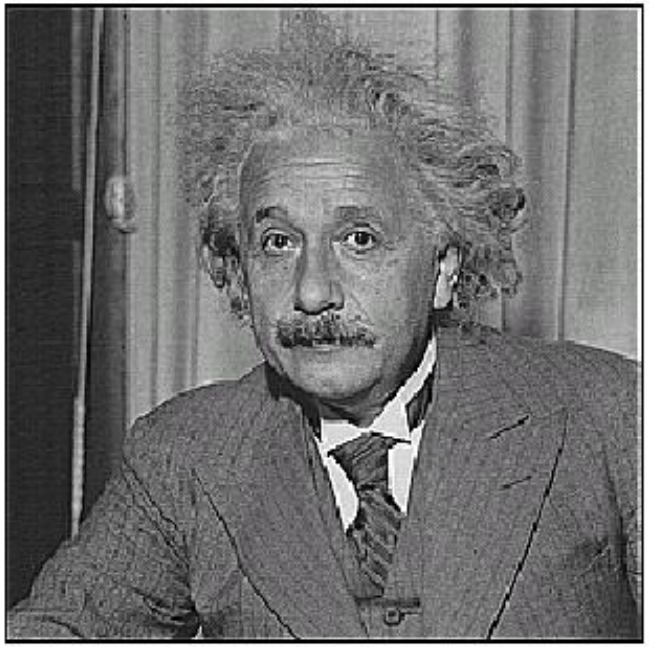
- Accentuates differences with local average

Source: D. Lowe

4. Practice with linear filters



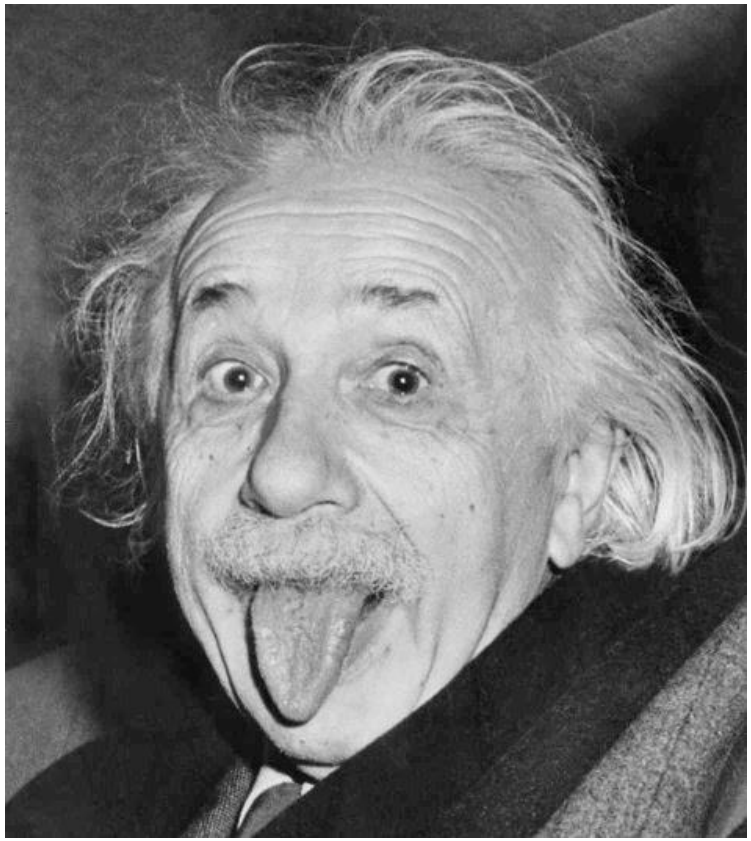
before

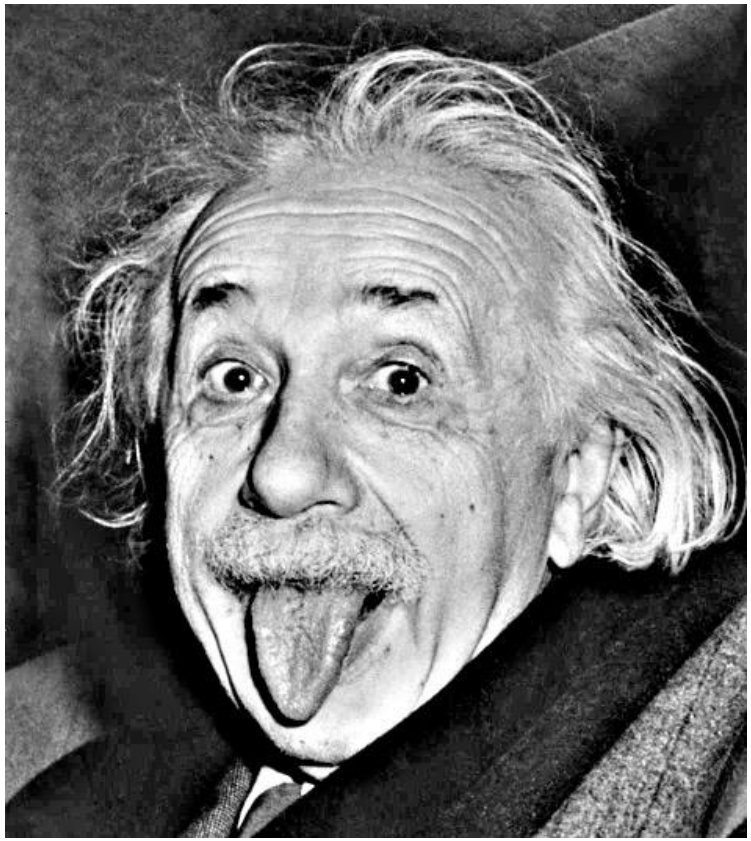


after

Source: D. Lowe

Sharpening examples





Sharpening examples

Sharpening examples

Sharpening examples

do you see
any problems
in this image?

Do not overdo it with sharpening

original

sharpened

oversharpened

What is wrong in this image?

Convolution

Convolution for 1D continuous signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

filtered signal

filter

input signal

Convolution for 1D continuous signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

filtered signal

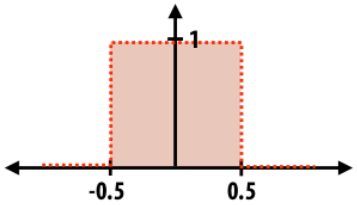
filter

input signal

Consider the box filter example:

1D continuous box filter

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}$$



filtering output is a blurred version of g

$$(f * g)(x) = \int_{-0.5}^{0.5} g(x-y)dy$$

Convolution for 2D discrete signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image

filter

input image

Convolution for 2D discrete signals

Definition of filtering as convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image — filter — input image

If the filter $f(i, j)$ is non-zero only within $-1 \leq i, j \leq 1$, then

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j) I(x - i, y - j)$$

The kernel we saw earlier is the 3x3 matrix representation of $f(i, j)$.

Convolution vs correlation

Definition of discrete 2D convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)$$

Definition of discrete 2D correlation:

notice the lack of a flip

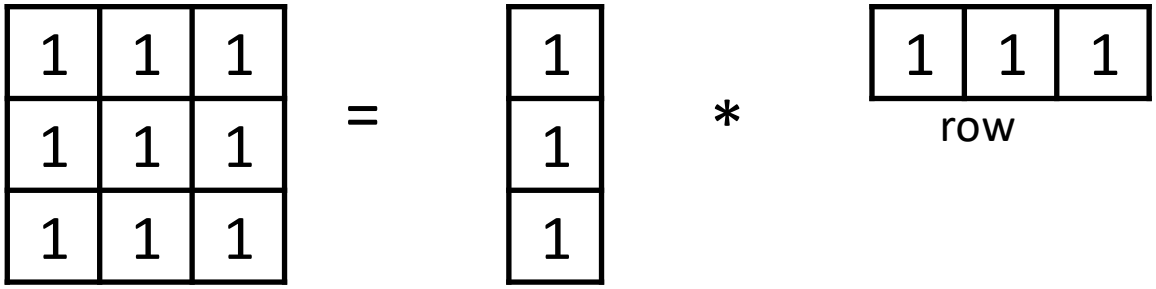
$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x + i, y + j)$$

- Most of the time won't matter, because our kernels will be symmetric.
- Will be important when we discuss frequency-domain filtering (lectures 5-6).

Separable filters

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

example:
box filter

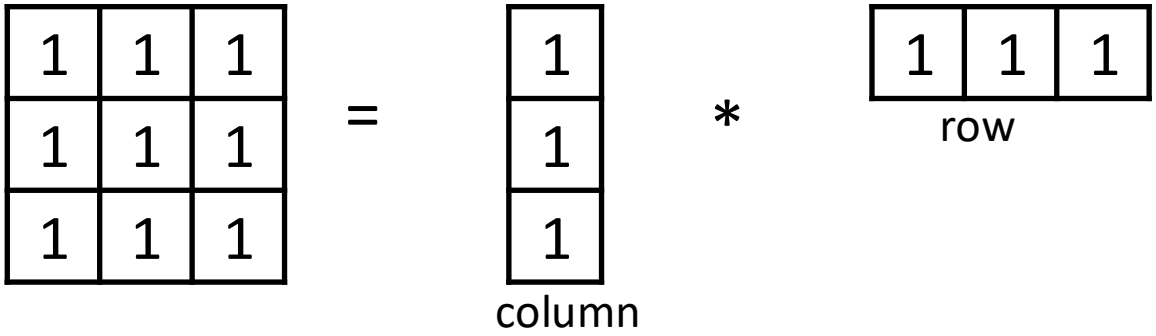


2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

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).

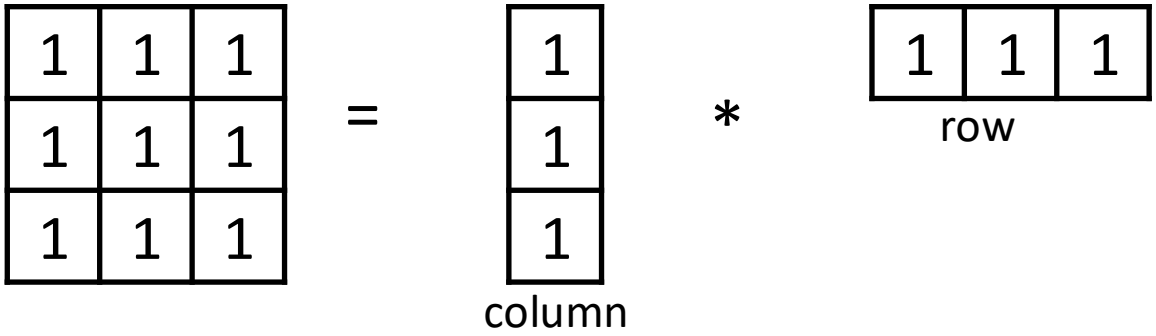
If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter?

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).

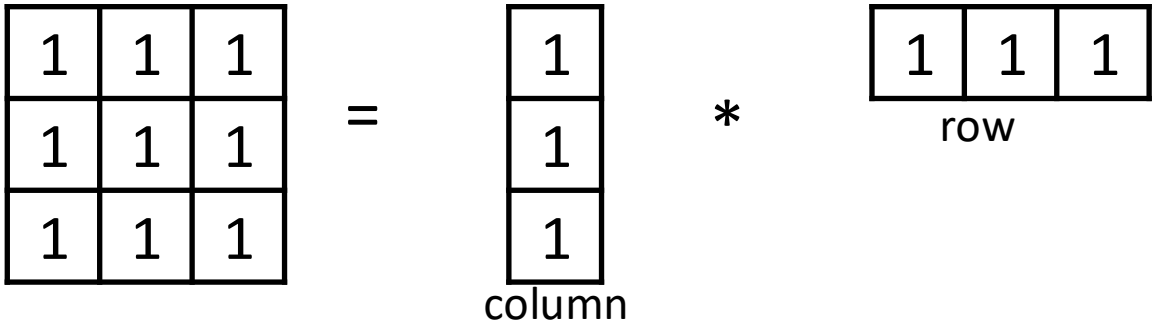
If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? → $M^2 \times N^2$
- What is the cost of convolution with a separable filter?

Separable filters

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

example:
box filter



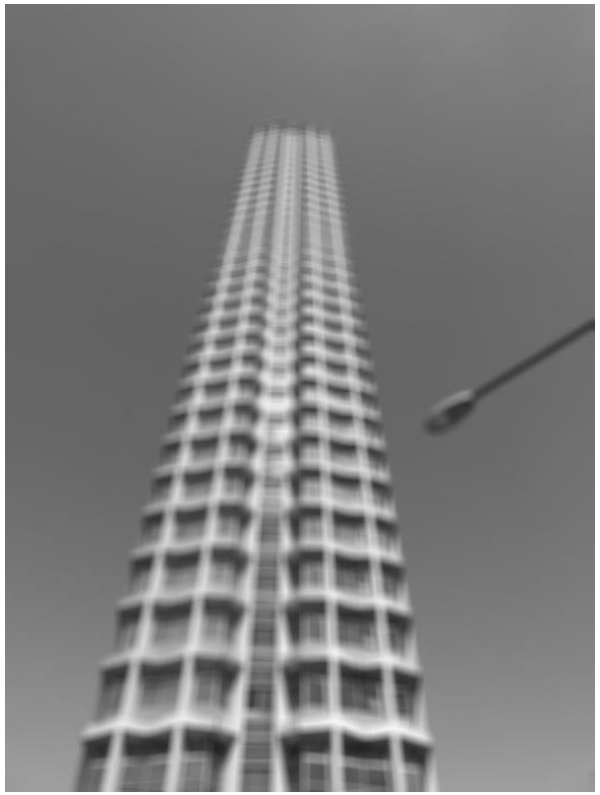
2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

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

- What is the cost of convolution with a non-separable filter? → $M^2 \times N^2$
- What is the cost of convolution with a separable filter? → $2 \times N \times M^2$

A few more filters

original



3x3 box filter

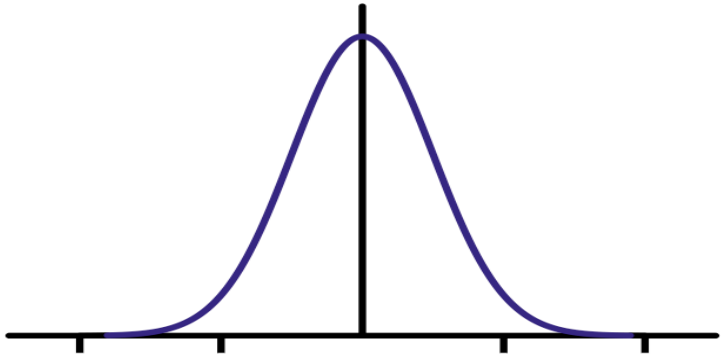
do you see
any problems
in this image?

The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance



Any heuristics for selecting where to truncate?

The Gaussian filter

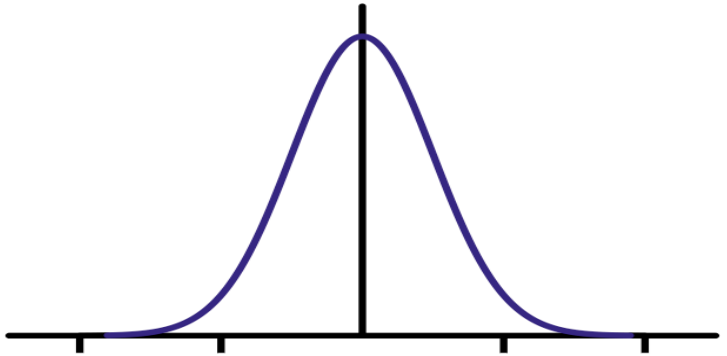
- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

- usually at 2-3σ



Is this a separable filter?

kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

The Gaussian filter

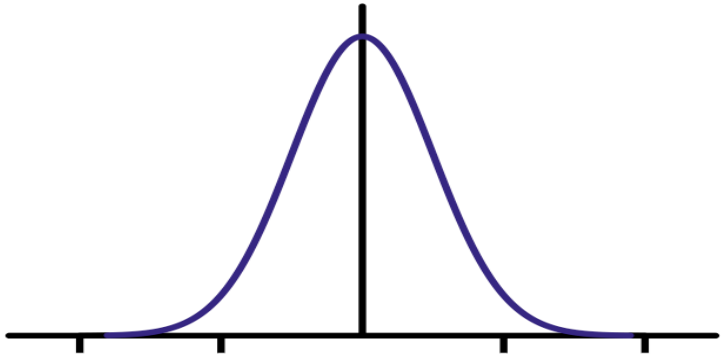
- named (like many other things) after Carl Friedrich Gauss
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

- usually at 2-3σ

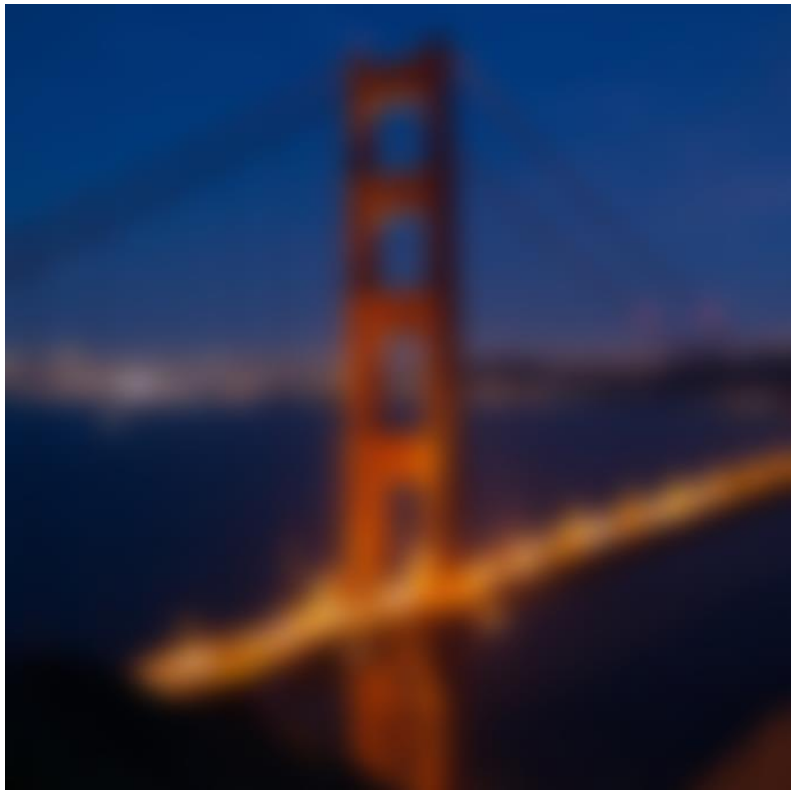


Is this a separable filter? Yes!

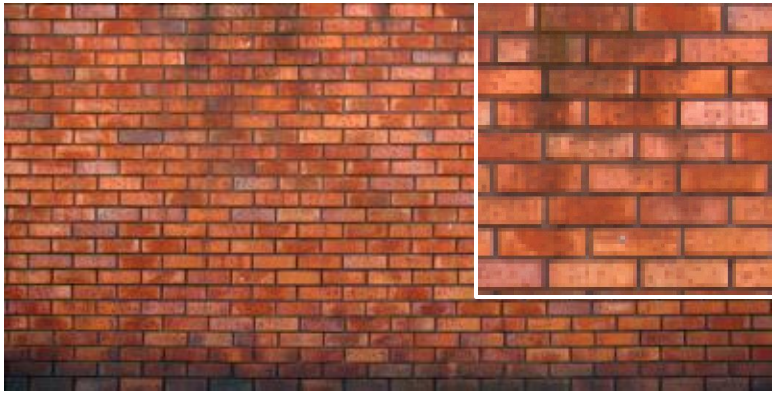
kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

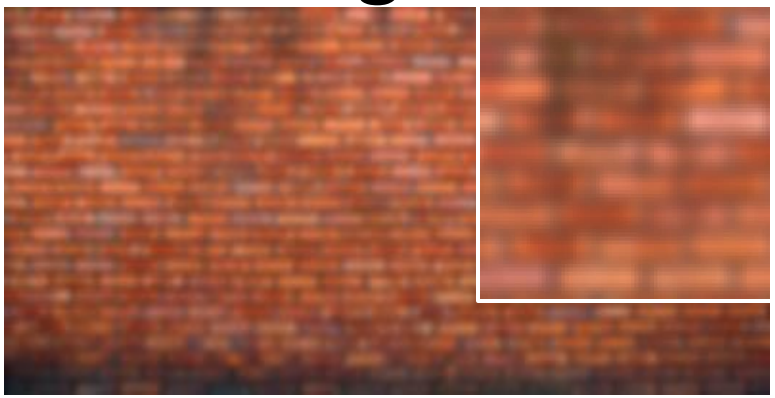
Gaussian filtering example



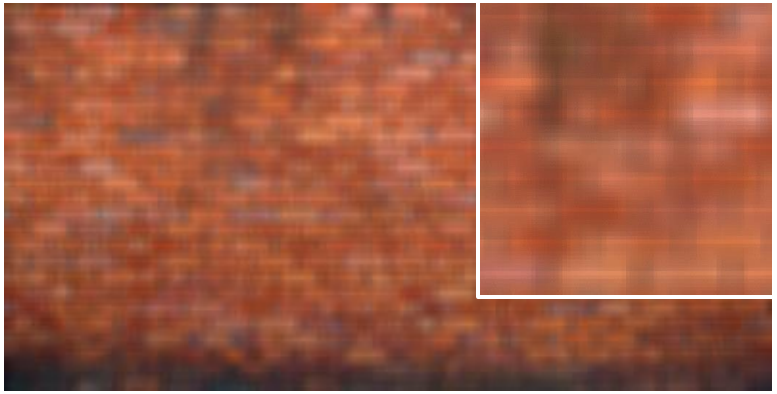
Gaussian vs box filtering



original



7x7 Gaussian



7x7 box

Which blur do you like better?

Correlation and Convolution

- 2d correlation

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

```
e.g., h = scipy.signal.correlate2d(f,I)
```

Correlation and Convolution

- 2d correlation

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

```
e.g., h = scipy.signal.correlate2d(f,I)
```

- 2d convolution

$$h[m,n] = \sum_{k,l} f[k,l]\, I[m-k, n-l]$$

```
e.g., h = scipy.signal.convolve2d(f,I)
```

Convolution is the same as correlation with a 180° rotated filter kernel.
Correlation and convolution are identical when the filter kernel is symmetric.

* Symmetric in the geometric sense, not in the matrix linear algebra sense.

Key properties of linear filters

Linearity:

$\texttt{imfilter(I, } f_1 + f_2\texttt{)} =$

$\texttt{imfilter(I,} f_1\texttt{)} + \texttt{imfilter(I,} f_2\texttt{)}$

Shift invariance:

Same behavior given intensities regardless of pixel location m,n

`imfilter(I,shift(f)) = shift(imfilter(I,f))`

Any linear, shift-invariant operator can be represented as a convolution.

S. Lazebnik

Convolution properties

Commutative: $a * b = b * a$

- Conceptually no difference between filter and signal
- But filtering implementations might break this equality, e.g., image edges

Associative: $a * (b * c) = (a * b) * c$

- Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
- This is equivalent to applying one filter: a * $(b_1 * b_2 * b_3)$
- Why important?
- Correlation is _not_ associative (rotation effect)

Distributes over addition: $a * (b + c) = (a * b) + (a * c)$

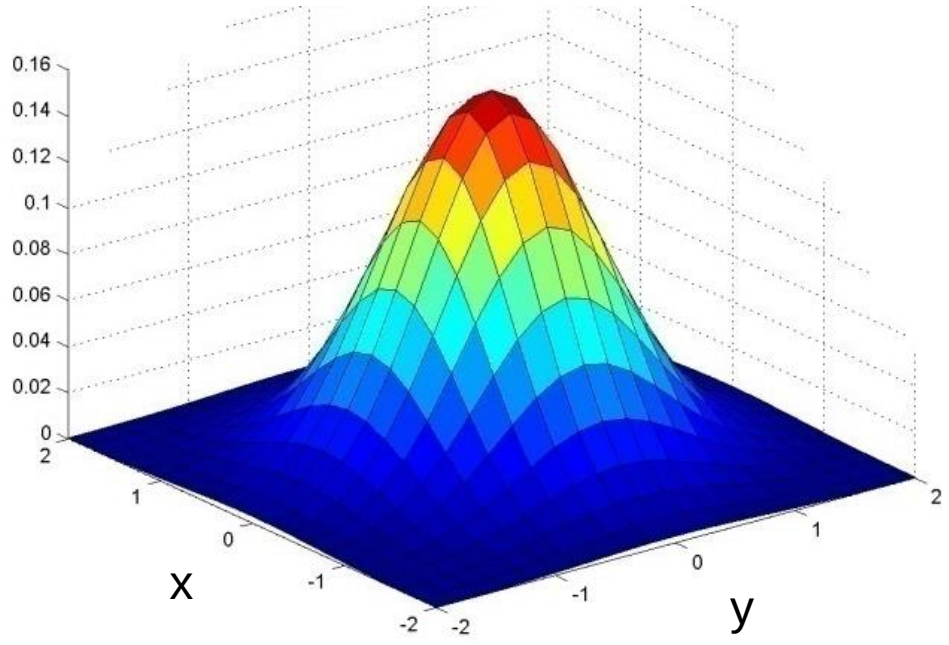
Scalars factor out: $ka * b = a * kb = k\,(a * b)$

Identity: unit impulse $e = [0, 0, 1, 0, 0]$, $a * e = a$

Source: S. Lazebnik

Important filter: Gaussian

Weight contributions of neighboring pixels by nearness



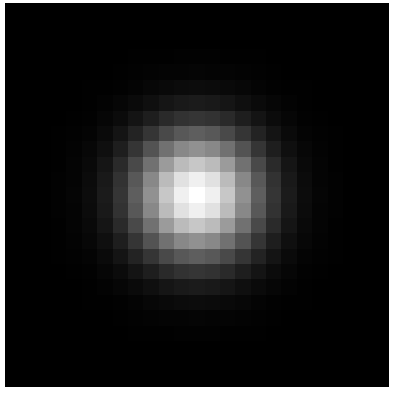
x

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

y

Kernel size 5 x 5,
Standard deviation $\sigma = 1$

Viewed from top



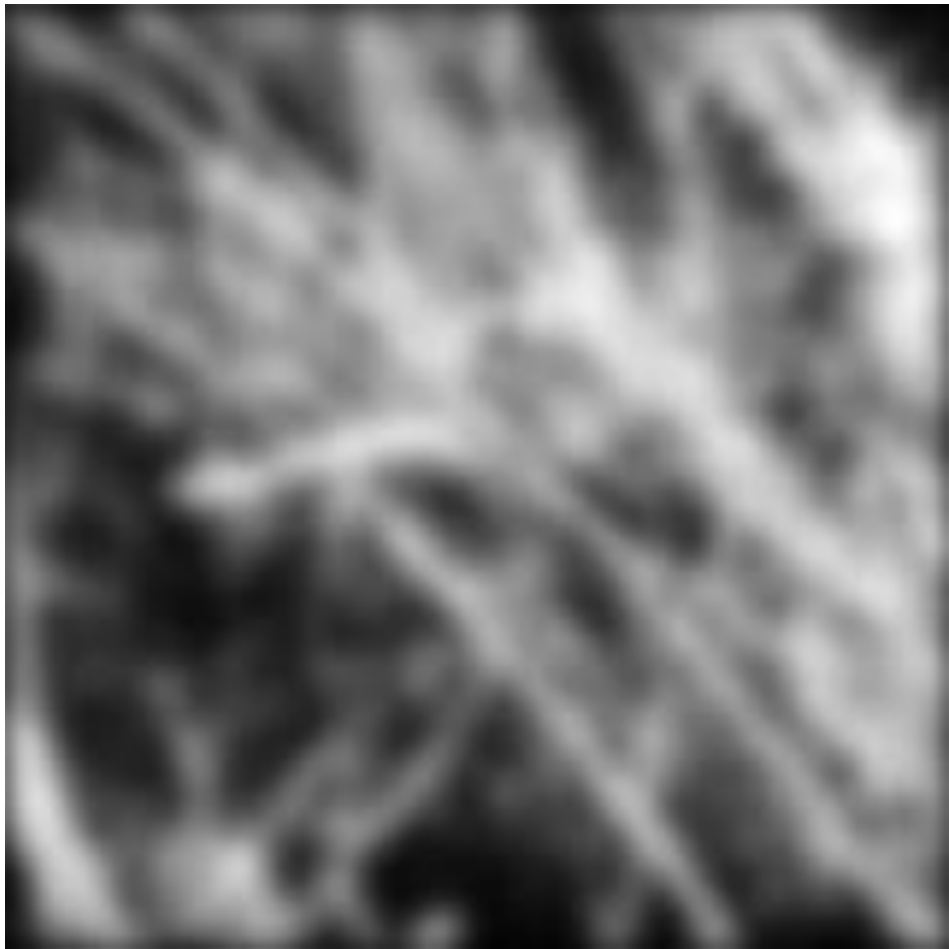
y

x

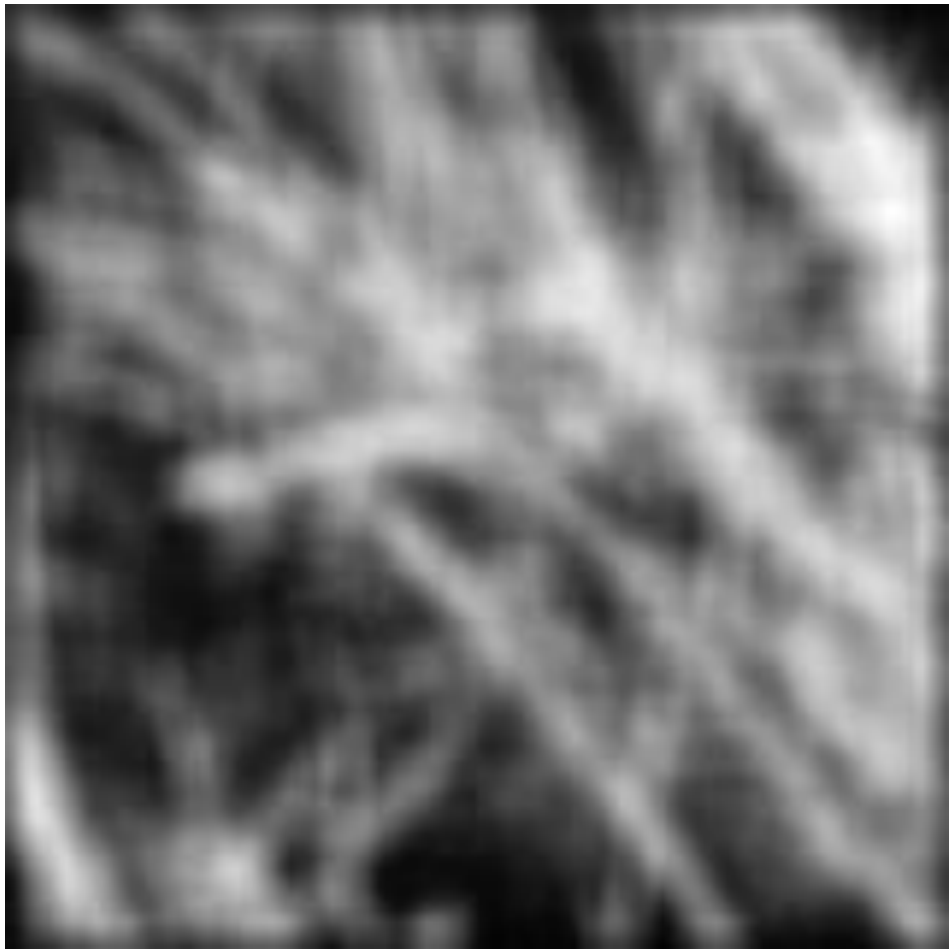
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Slide credit: Christopher Rasmussen

Smoothing with Gaussian filter



Smoothing with box filter



Gaussian filters

Gaussian convolved with Gaussian…

…is another Gaussian

- So can smooth with small-width kernel, repeat, and get same result as larger-width kernel
- Convolving twice with Gaussian kernel of width σ is same as convolving once with kernel of width $\sigma\sqrt{2}$

Separable kernel

- Factors into product of two 1D Gaussians

Source: K. Grauman

Separability of the Gaussian filter

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

$$= \left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{y^2}{2\sigma^2}}\right)$$

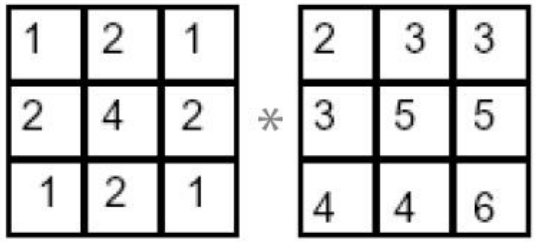
The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

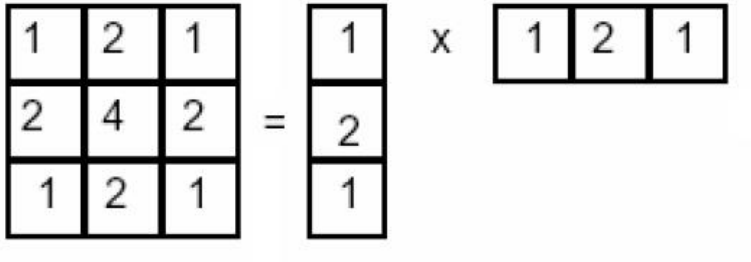
Source: D. Lowe

Separability example

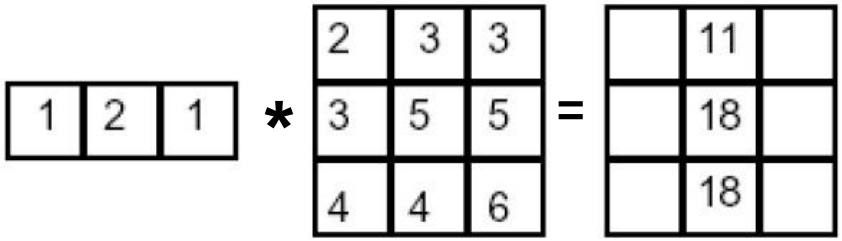
2D convolution
(center location only)



The filter factors
into a product of 1D
filters:



Perform convolution
along rows:



Followed by convolution
along the remaining column:

Source: K. Grauman

Separability

Why is separability useful in practice?

MxN image, PxQ filter

- 2D convolution: ~MNPQ multiply-adds
- Separable 2D: ~MN(P+Q) multiply-adds

Speed up = PQ/(P+Q)

9x9 filter = ~4.5x faster

Practical matters

How big should the filter be?

- Values at edges should be near zero
- Gaussians have infinite extent…
- Rule of thumb for Gaussian: set filter half-width to about 3 σ

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

Convolution in Convolutional Neural Networks

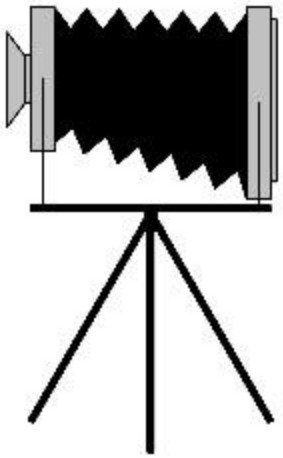
- Convolution is the basic operation in CNNs
- Learning convolution kernels allows us to learn which `features' provide useful information in images.


Ben Thomas

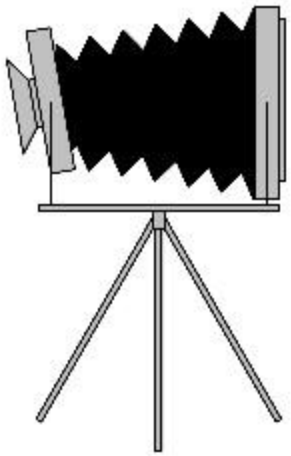
Tilt-shift photography

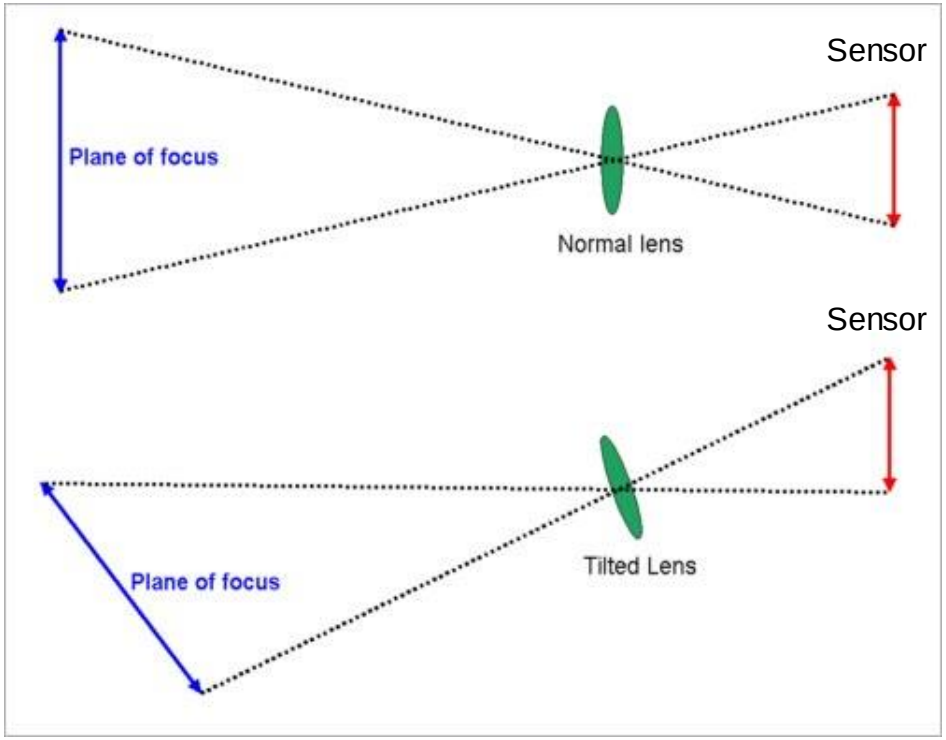
Tilt shift camera

Shift



Tilt





Can we fake tilt shift?

We need to blur the image

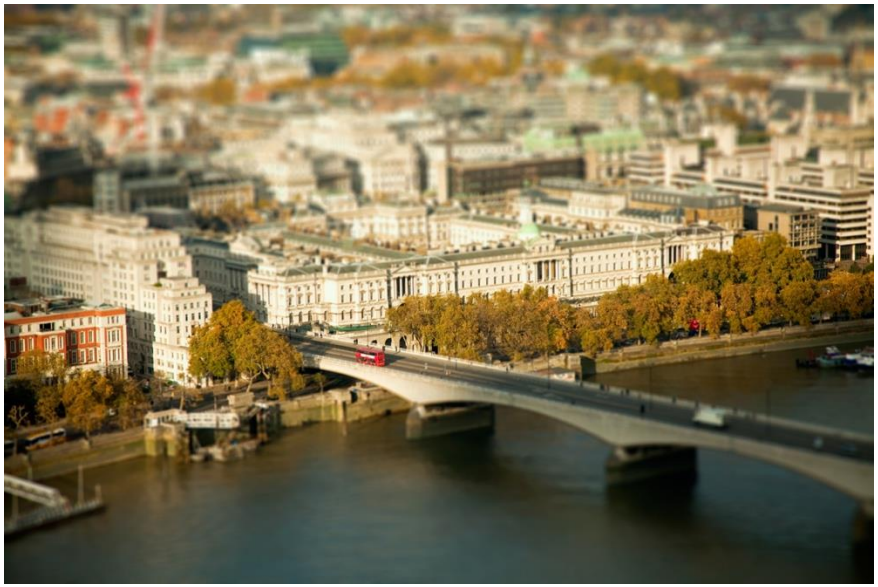
- OK, now we know how to do that.

Can we fake tilt shift?

We need to blur the image

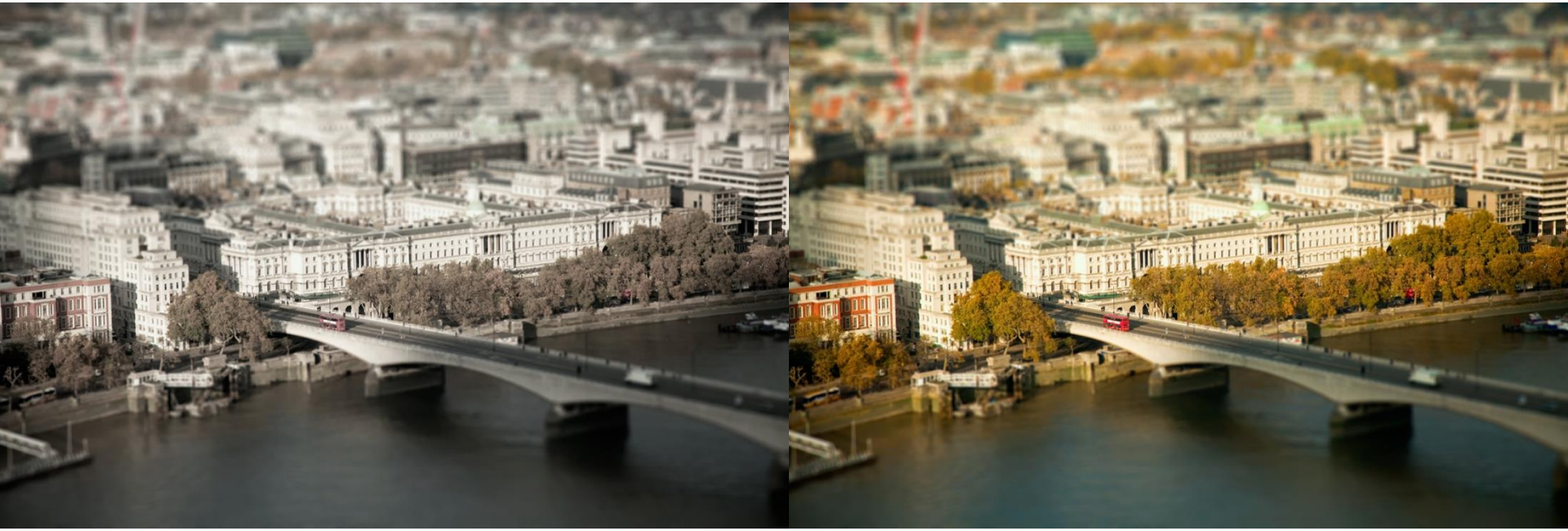
– OK, now we know how to do that.

We need to blur progressively more away from our ‘fake’ focal point



But can I make it look more like a toy?

- Boost *saturation* – toys are very colorful
- We'll learn how to do this when we discuss color



Think-Pair-Share

* = Convolution operator

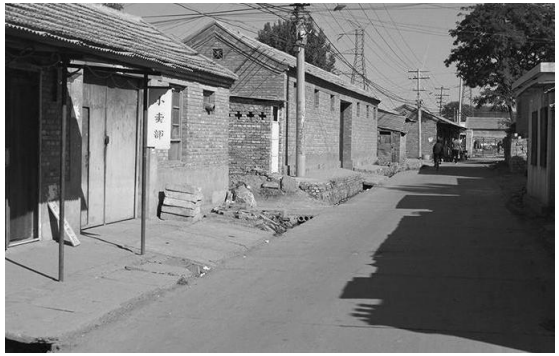
1) $\_ = D * B$

2) $A = \_ * \_$

3) $F = D * \_$

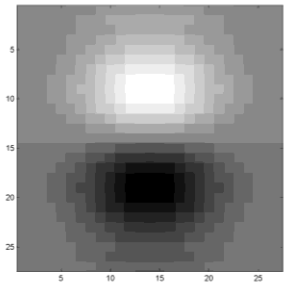
4) $\_ = D * D$

D

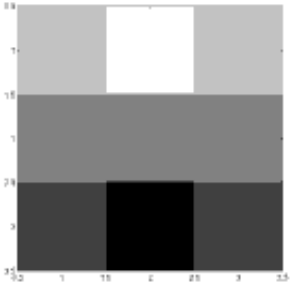


H

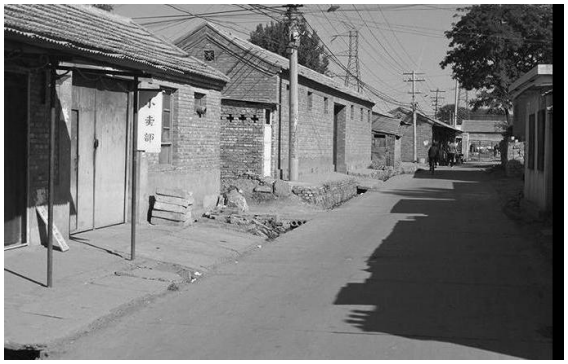
A



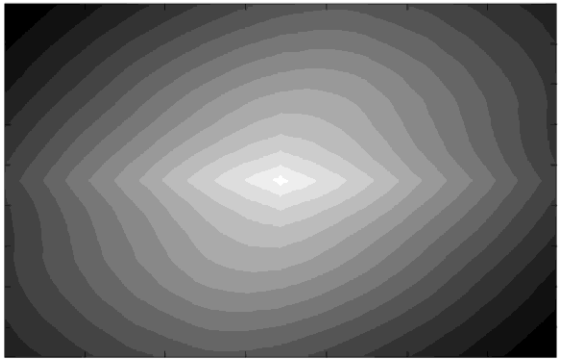
B



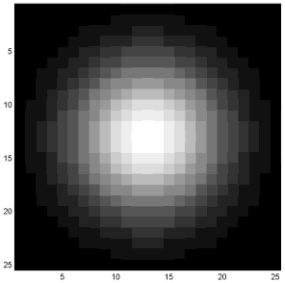
F



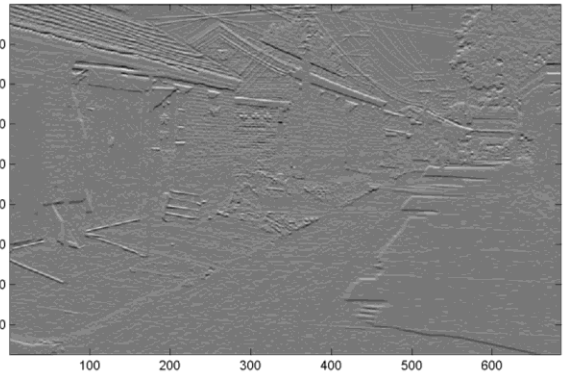
I



C



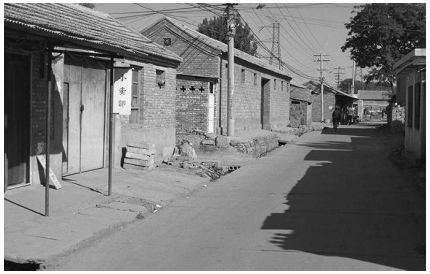
G



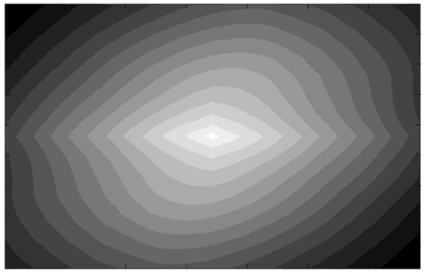
E

I = D * D

D (275 x 175 pixels)



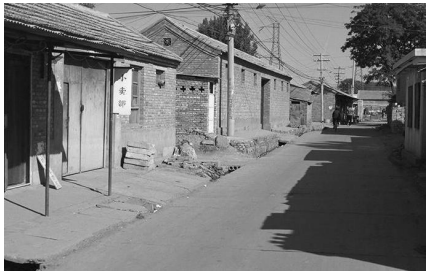
I (from slide – 275 x 175)



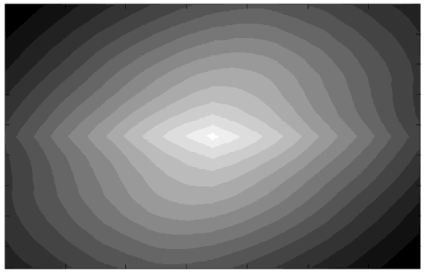
“…something to do with lack of content (black) at edges…”

I = D * D

D (275 x 175 pixels)

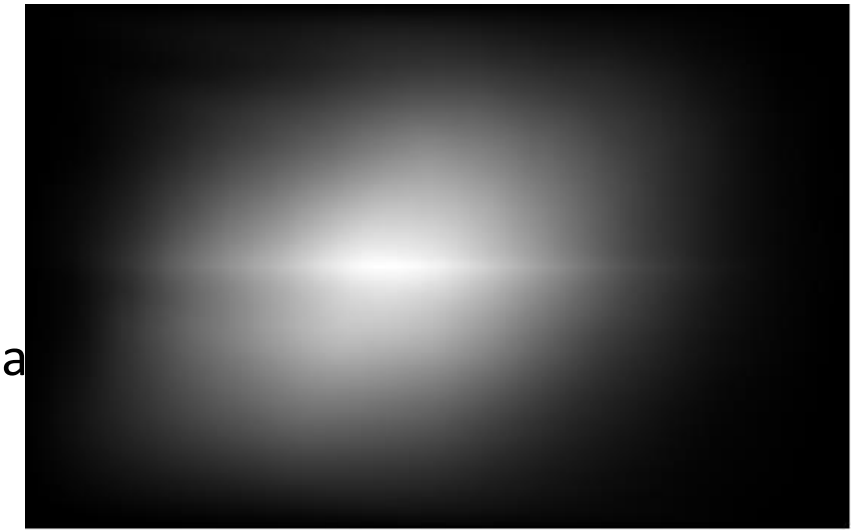


I (from slide – 275 x 175)



```
>> D = img_to_float32( io.imread( 'convexample.png' ) )
>> I = convolve2d( D, D )
>> np.max(I)
1.1021e+04
```

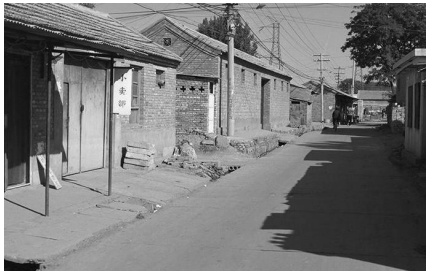
I_norm



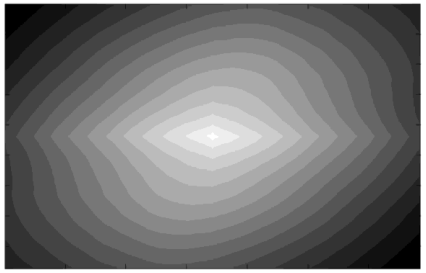
```
# Normalize for visualization
>> I_norm = (I – np.min(I)) / (np.ma
>> plt.imshow( I_norm )
```

$I = D * D$

D (275 x 175 pixels)



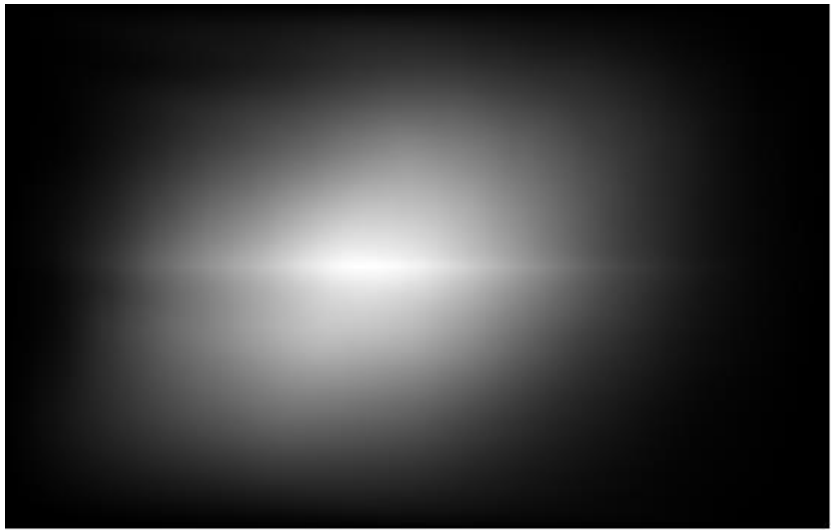
I (from slide – 275 x 175)



(275-1)/2

(275-1)/2

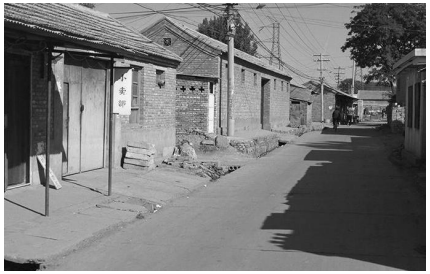
I_norm (549 x 349 pixels)



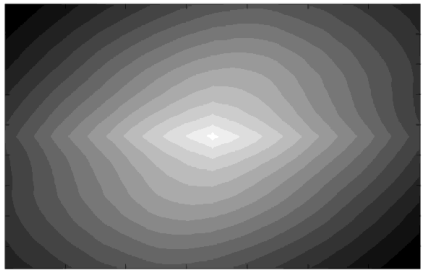
For x: 275 + (275-1)/2 + (275-1)/2
= 549

I = D * D

D (275 x 175 pixels)

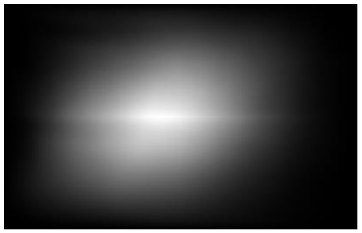


I (from slide – 275 x 175)



>> I = convolve2d(D, D, mode='full')

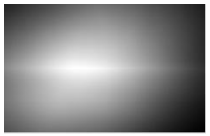
(Default; pad with zeros)



549 x 349

>> I = convolve2d(D, D, mode='same')

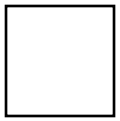
(Return same size as D)



275 x 175

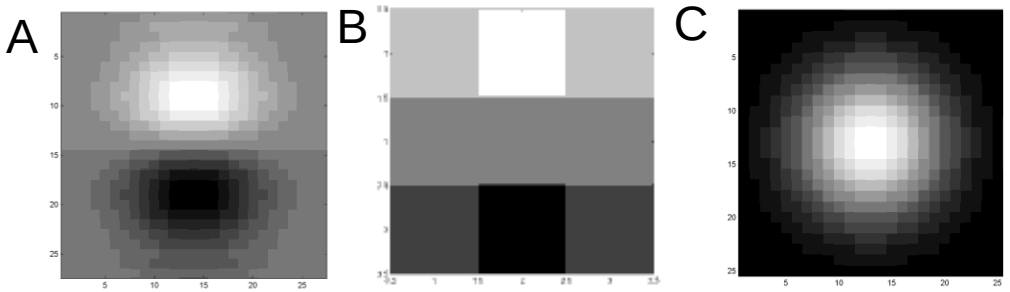
>> I = convolve2d(D, D, mode='valid')

(No padding)



Value = 10528.3
1x1

A = B * C – “because it kind of looks like it.”



C is a Gaussian filter
(or something close to it it),
and we know that it ‘blurs’.

When the filter ‘looks like’ the image = ‘template matching’

Filtering viewed as comparing an image of
what you want to find against all image regions.

For symmetric filters: use either convolution or correlation.

For nonsymmetric filters: correlation is template matching.

Filtering: Correlation and Convolution

- 2d correlation

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

```
e.g., h = scipy.signal.correlate2d(f,I)
```

- 2d convolution

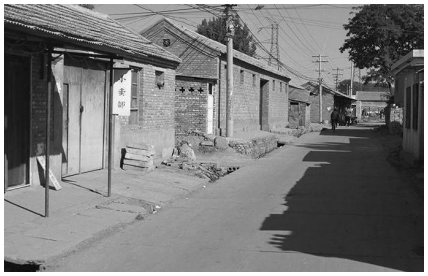
$$h[m,n] = \sum_{k,l} f[k,l]\, I[m-k, n-l]$$

```
e.g., h = scipy.signal.convolve2d(f,I)
```

Convolution is the same as correlation with a 180° rotated filter kernel.
Correlation and convolution are identical when the filter kernel is symmetric.

OK, so let's test this idea. Let's see if we can use correlation to 'find' the parts of the image that look like the filter.

D (275 x 175 pixels)



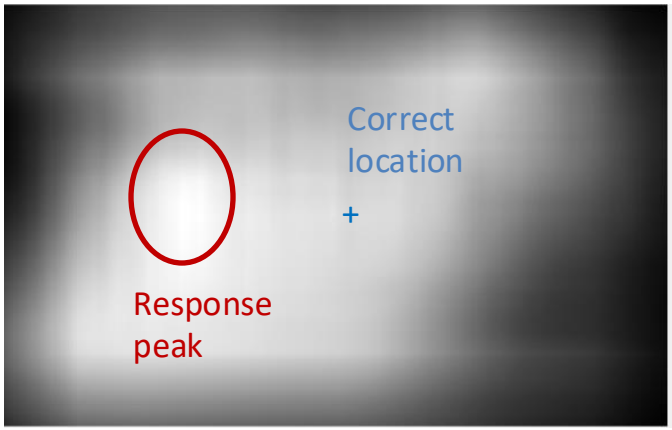
>> f = D[57:117, 107:167]

Expect response 'peak' in middle of I

f
61 x 61

>> I = correlate2d(D, f, 'same')

I



Hmm…
That didn't work – why not?

[Thanks to Robert Collins @ Penn State]

Correlation

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

```
e.g., h = scipy.signal.correlate2d(f,I)
```

As brightness in I increases, the response in h will increase, as long as f is positive.

Overall brighter regions will give higher correlation response -> not useful!

OK, so let's subtract the mean

>> f = D[57:117, 107:167]

>> f2 = f – np.mean(f)

>> D2 = D – np.mean(D)

Now zero centered.
Score is higher only when dark parts match and when light parts match.

>> I2 = correlate2d(D2, f2, 'same')

D2 (275 x 175 pixels)

f2
61 x 61

Remember – float data type– goes negative!

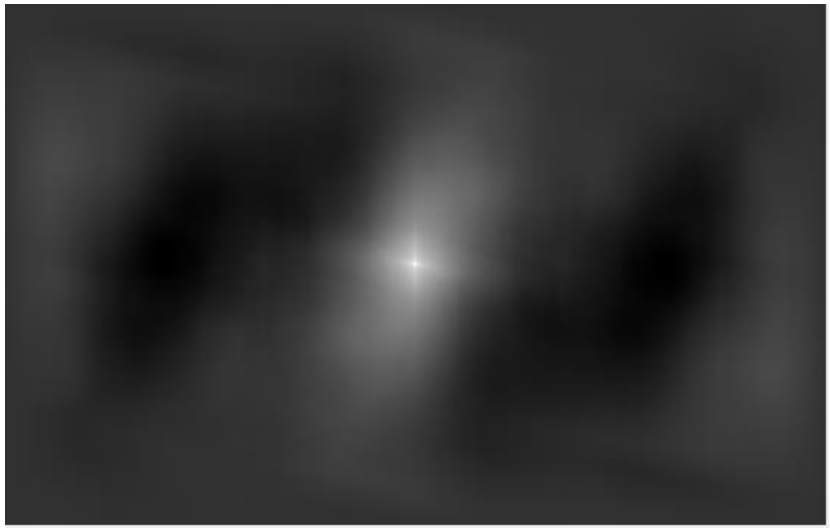
I2

D2 (275 x 175 pixels)

Or our original example: D2 correlated with itself

```
>> I3 = correlate2d( D2, D2, 'full' )
```

I3



What happens with convolution?

```
>> f = D[ 57:117, 107:167 ]
>> f2 = f – np.mean(f)
>> D2 = D – np.mean(D)

>> I2 = convolve2d( D2, f2, ‘same’ )
```

D2 (275 x 175 pixels)

f2
61 x 61

I2

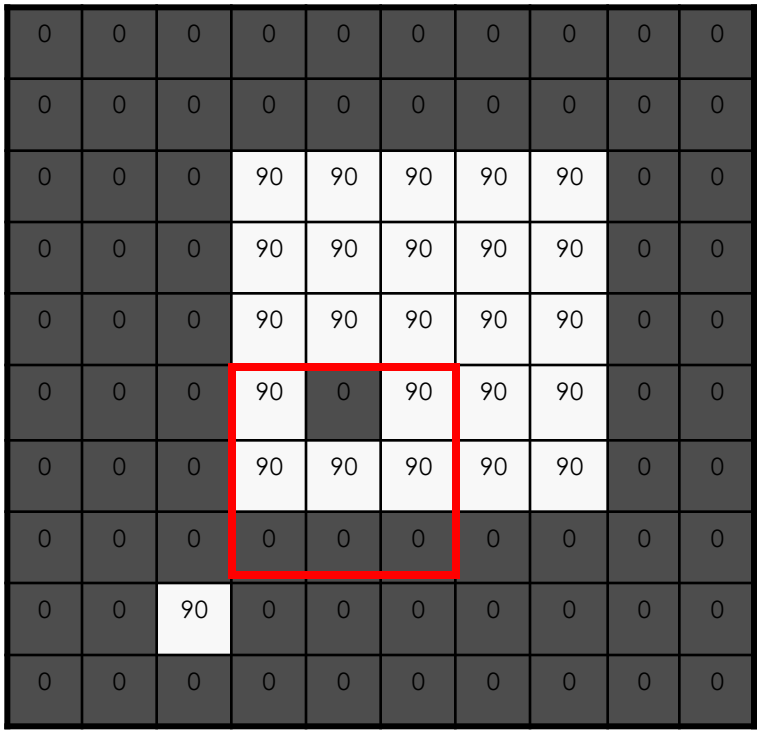
NON-LINEAR FILTERS

Median filters

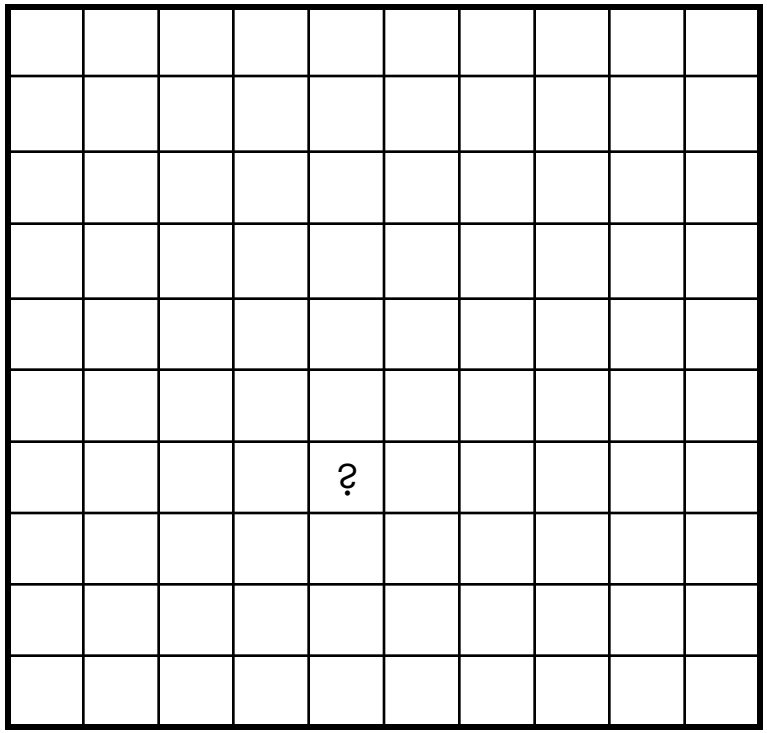
- Operates over a window by selecting the median intensity in the window.
- ‘Rank’ filter as based on ordering of gray levels
  - E.G., min, max, range filters

Median filter?

$I[.,.]$



$h[.,.]$

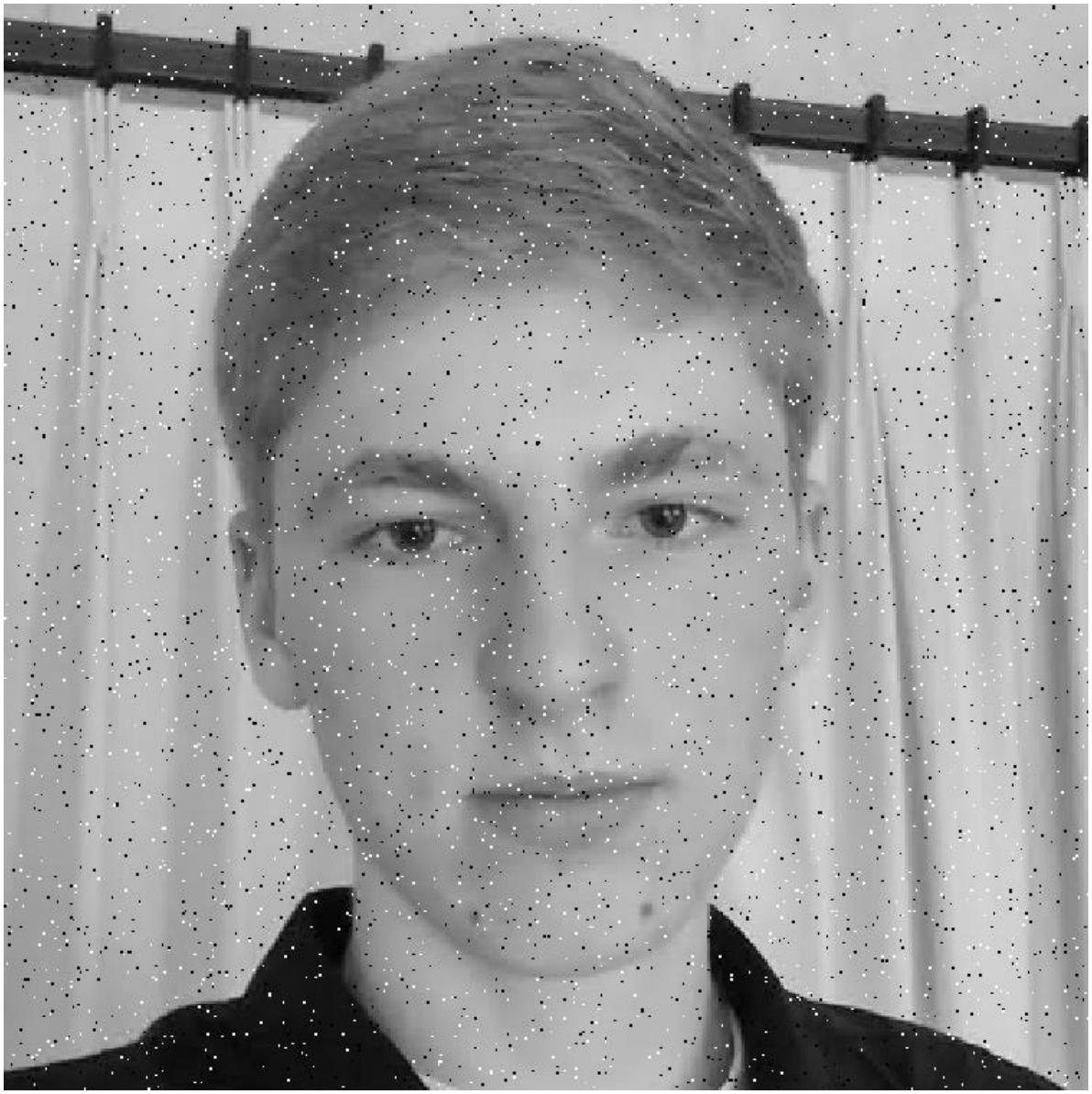


Credit: S. Seitz

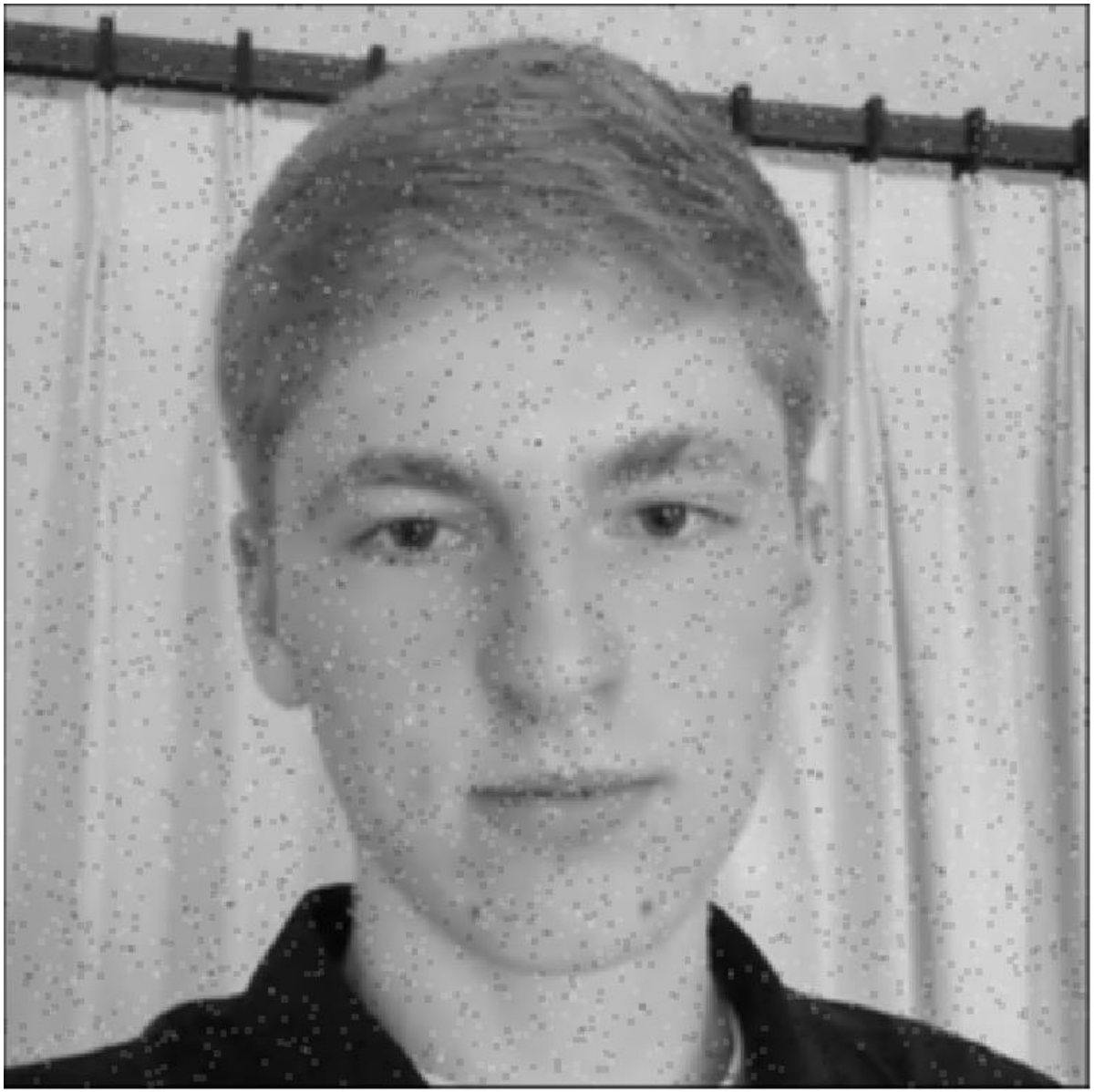
Median filters

- Operates over a window by selecting the middle intensity in the window.
- What advantage does a median filter have over a mean filter?

Noise – Salt and Pepper Jack

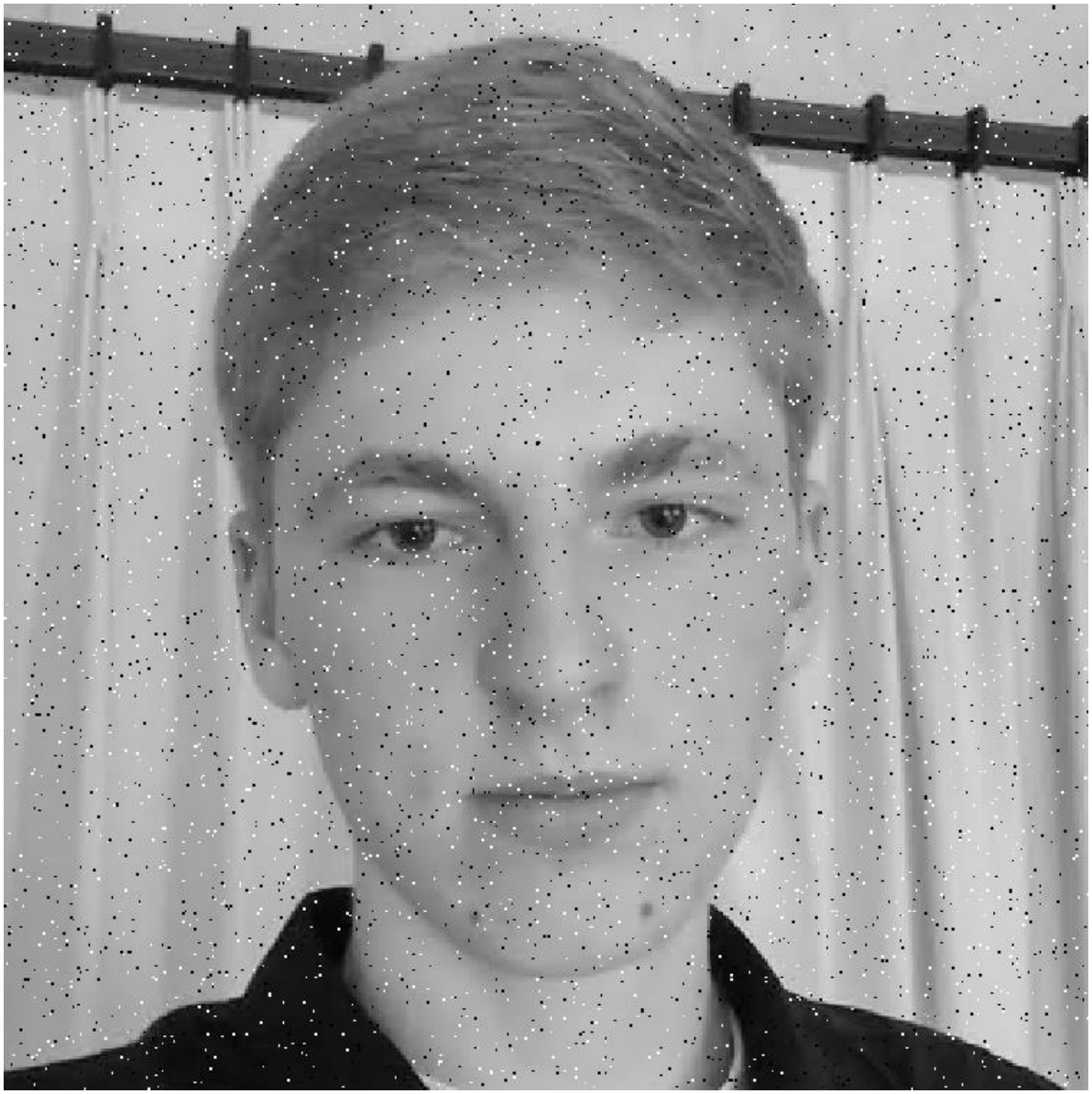


Mean Jack – 3 x 3 filter

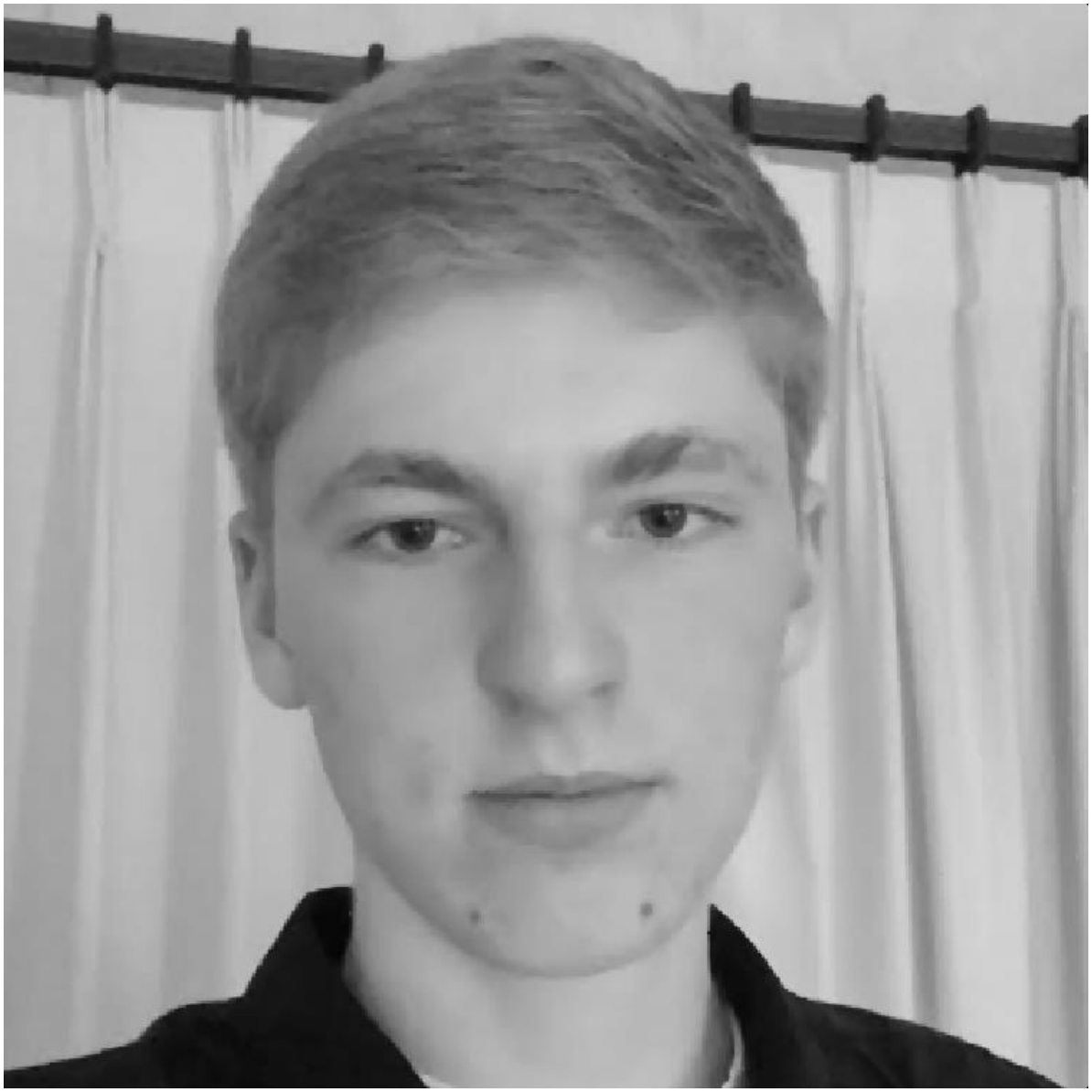


Very Mean Jack – 11 x 11 filter

Noisy – Salt and Pepper Jack



Median Jack – 3 x 3



Very Median Jack – 11 x 11

Median filters

- Operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?

Median filters

- Operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?

Interpretation: Median filtering is sorting.

Review: questions

1. Write down a 3x3 filter that both:
   - Returns a positive value if the average value of the 4-adjacent neighbors is less than the center,
   - Returns a negative value otherwise.

2. Write down a filter that will compute the gradient in the x-direction:

```
gradx(x,y) = im(x+1,y) - im(x,y) for each x, y
```

Review: questions

1. Write down a 3x3 filter that both:
   - Returns a positive value if the average value of the 4-adjacent neighbors is less than the center,
   - Returns a negative value otherwise.

$$\begin{bmatrix} 0 & -1/4 & 0 \\ -1/4 & 1 & -1/4 \\ 0 & -1/4 & 0 \end{bmatrix}$$

2. Write down a filter that will compute the gradient in the x-direction:

```
gradx(x,y) = im(x+1,y) - im(x,y) for each x, y
```

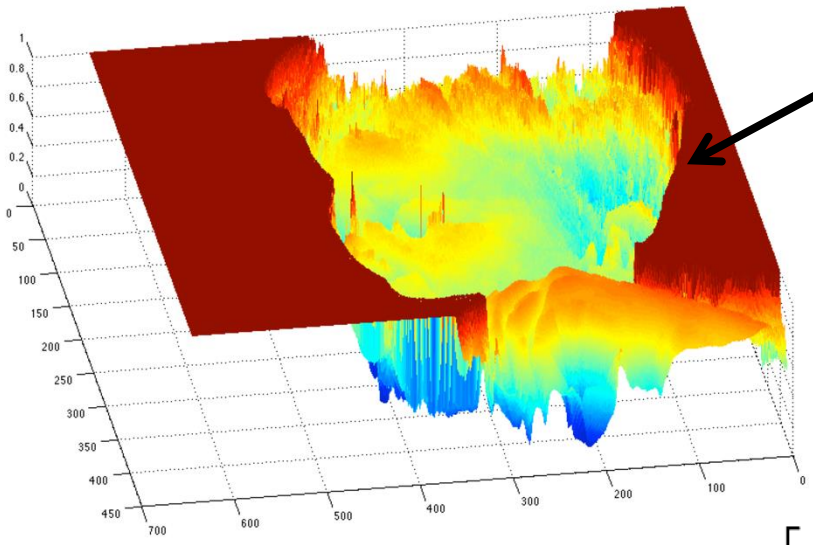
```
[0 -1 1]
```

Image gradients

What are image edges?

grayscale image

$f(\boldsymbol{x})$



Very sharp discontinuities in intensity.

domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

✓ You take derivatives: derivatives are large at discontinuities.

How do you differentiate a discrete image (or any other discrete signal)?

Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

✓ You take derivatives: derivatives are large at discontinuities.

How do you differentiate a discrete image (or any other discrete signal)?

✓ You use finite differences.

Finite differences

High-school reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Finite differences

High-school reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

What convolution kernel does this correspond to?

Finite differences

High-school reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

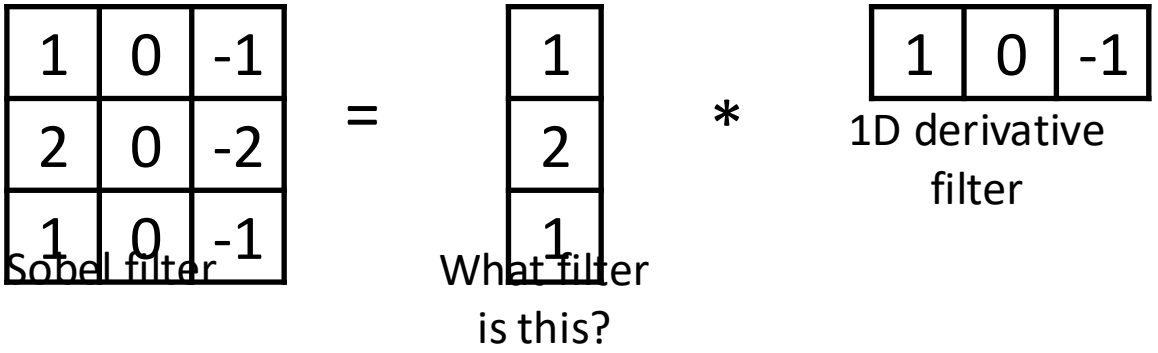
For discrete signals: Remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

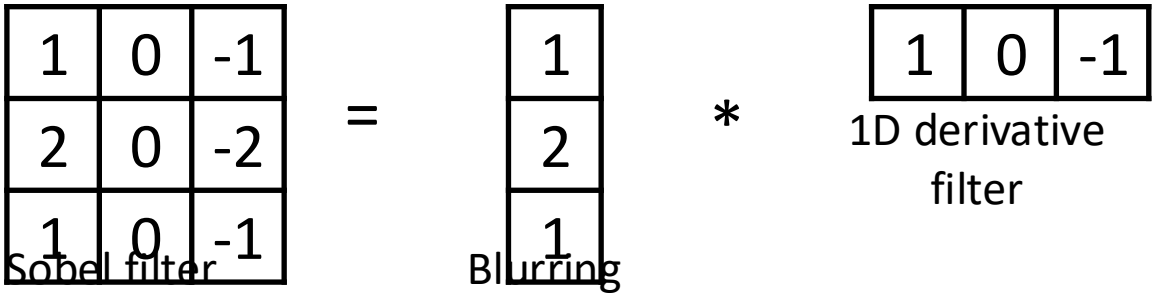
1D derivative filter

1	0	-1

The Sobel filter

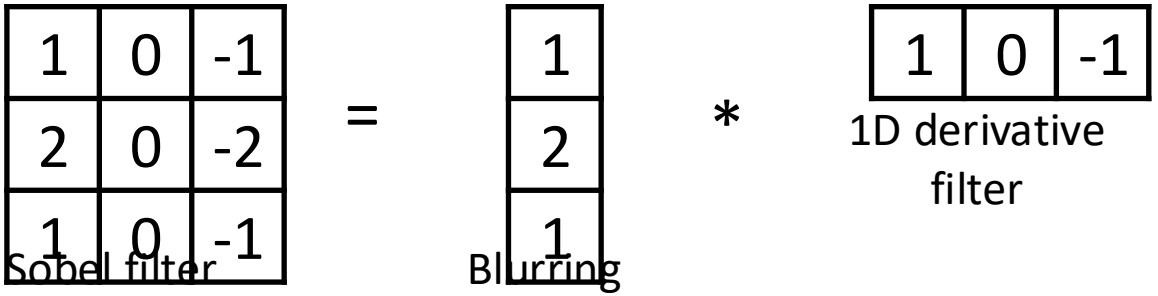


The Sobel filter



In a 2D image, does this filter responses along horizontal or vertical lines?

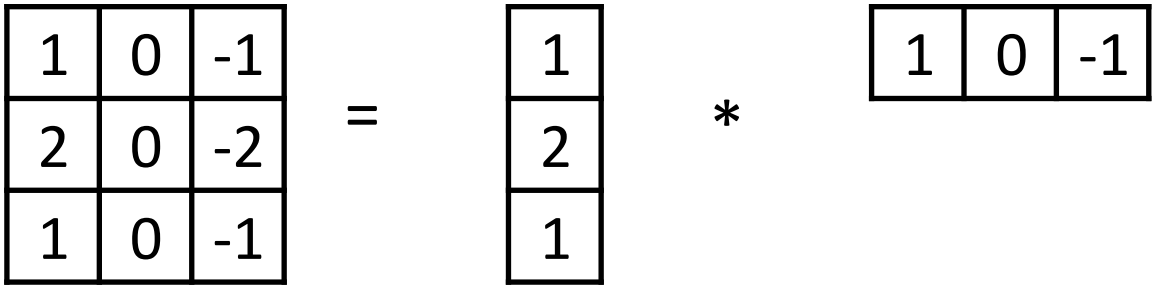
The Sobel filter



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

The Sobel filter

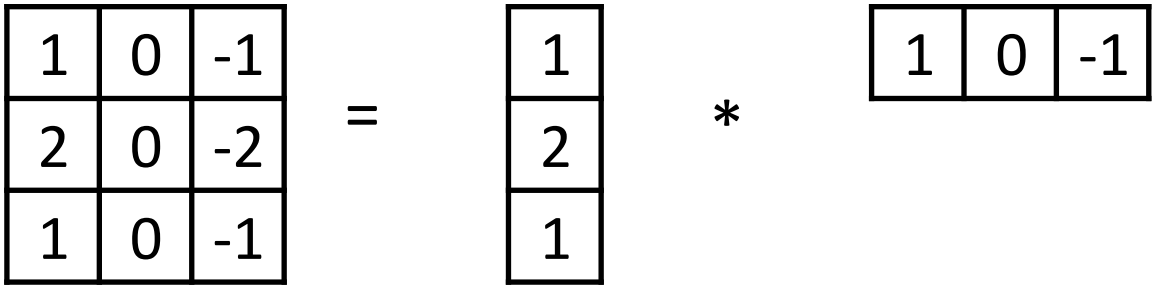
Horizontal Sober filter:



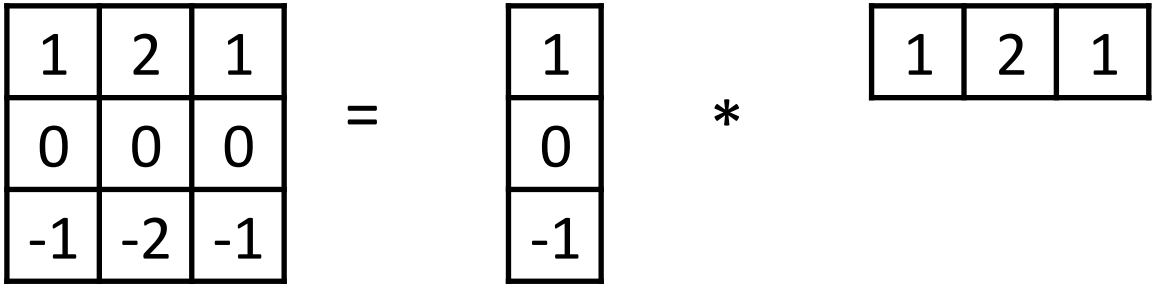
What does the vertical Sobel filter look like?

The Sobel filter

Horizontal Sober filter:

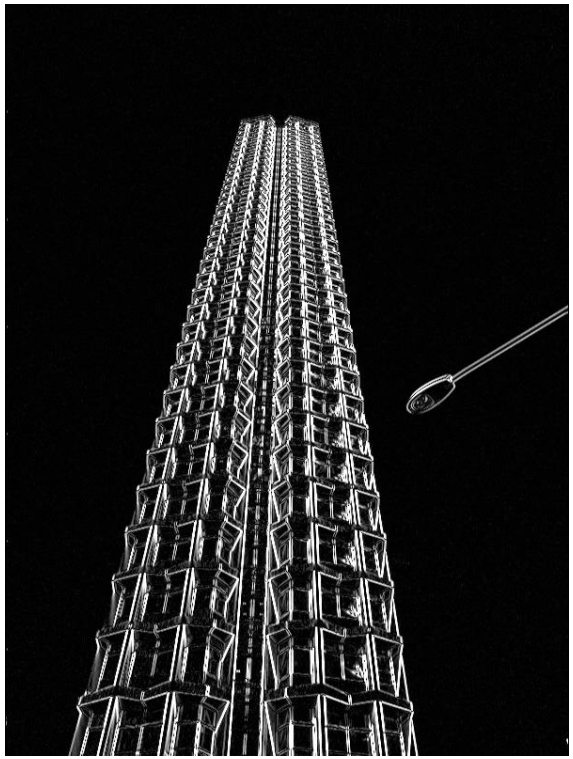


Vertical Sobel filter:

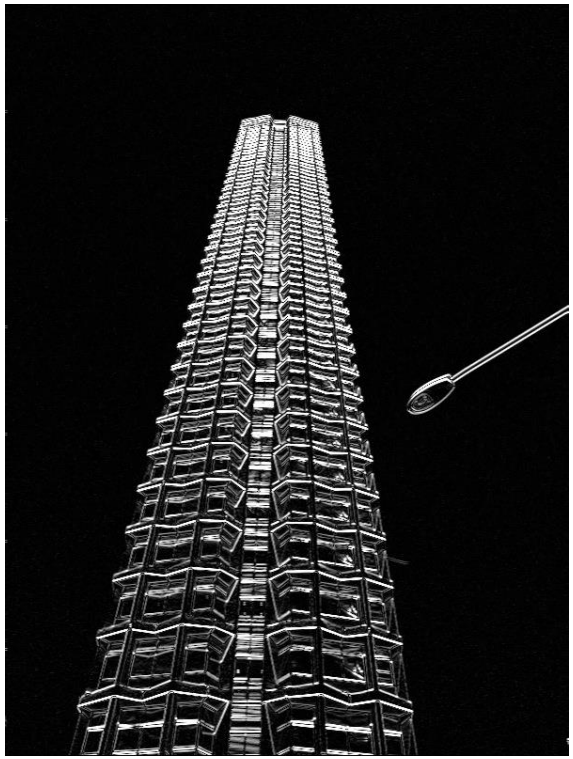


Sobel filter example

original



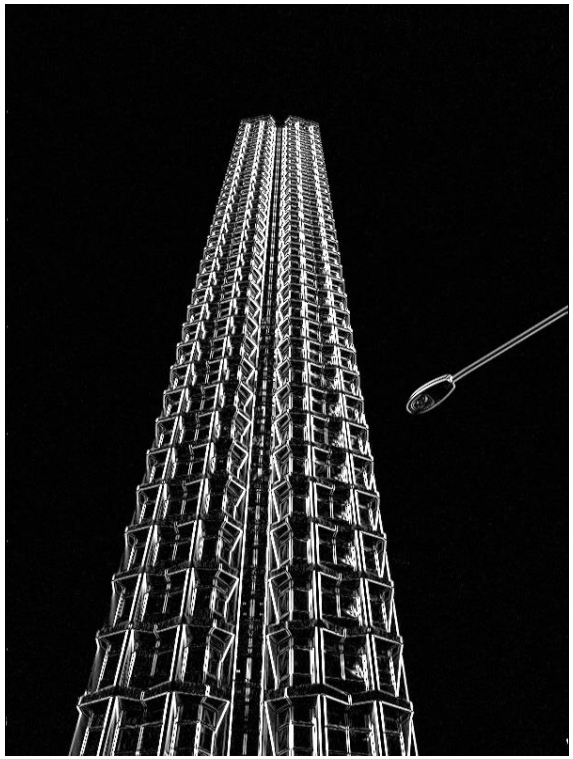
which Sobel filter?



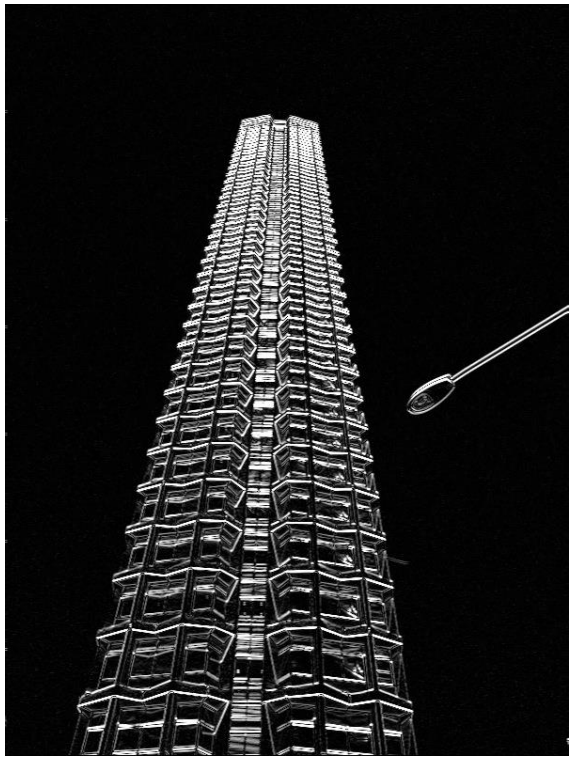
which Sobel filter?

Sobel filter example

original

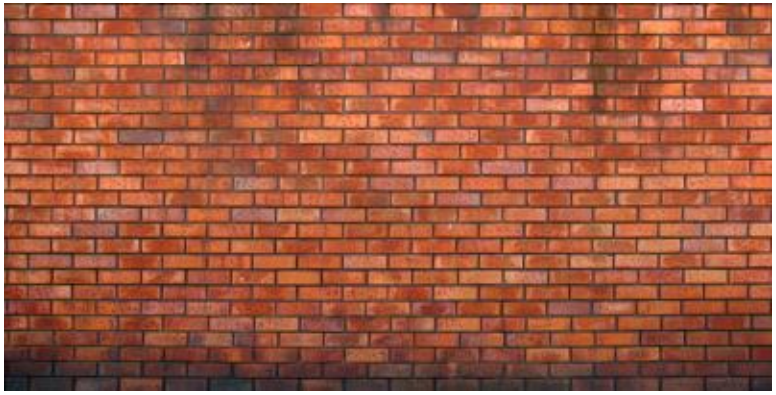


horizontal Sobel filter

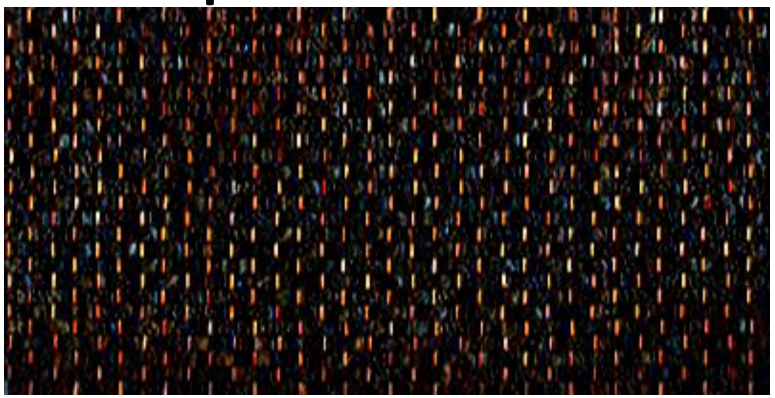


vertical Sobel filter

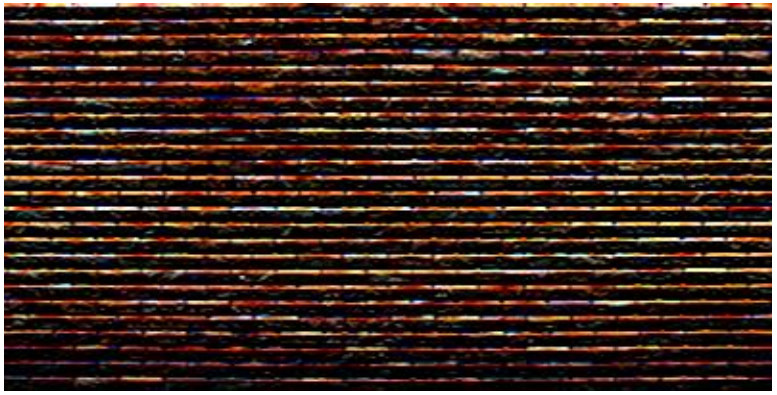
Sobel filter example



original



horizontal Sobel filter



vertical Sobel filter

Several derivative filters

Sobel

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

Scharr

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

Prewitt

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

Roberts

0	1
-1	0

1	0
0	-1

- How are the other filters derived and how do they relate to the Sobel filter?
- How would you derive a derivative filter that is larger than 3x3?

Computing image gradients

1. Select your favorite derivative filters.

$S_x =$

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

$S_y =$

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

Computing image gradients

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \boldsymbol{S}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

2. Convolve with the image ~~to compute d~~erivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

Computing image gradients

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \qquad \boldsymbol{S}_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

2. Convolve with the image ~~to compute~~ derivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

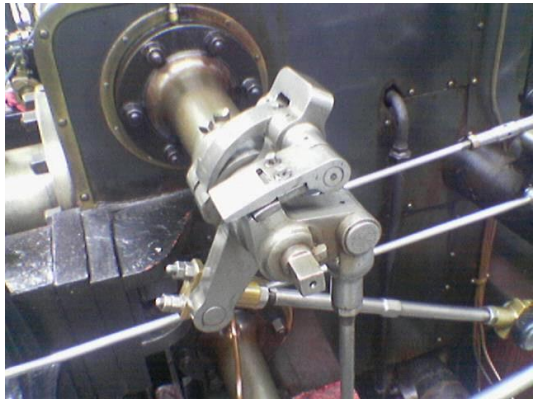
3. Form the image gradient, and compute its direction and amplitude.

$$\nabla \boldsymbol{f} = \left[\frac{\partial \boldsymbol{f}}{\partial x}, \frac{\partial \boldsymbol{f}}{\partial y}\right] \qquad \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right) \qquad ||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

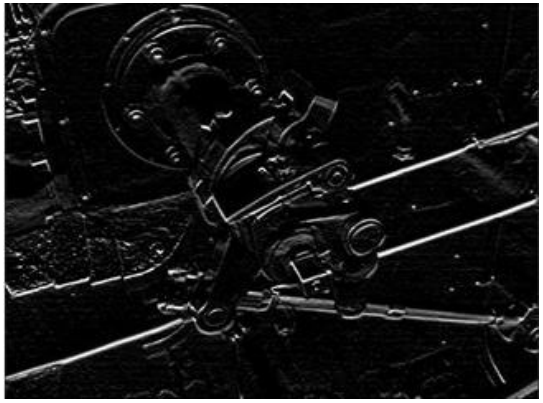
gradient direction amplitude

Image gradient example

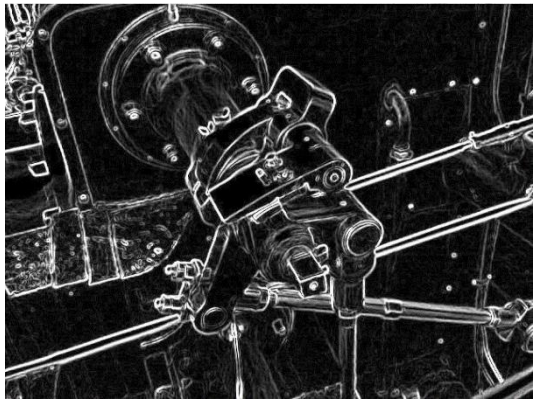
original



vertical derivative



gradient amplitude

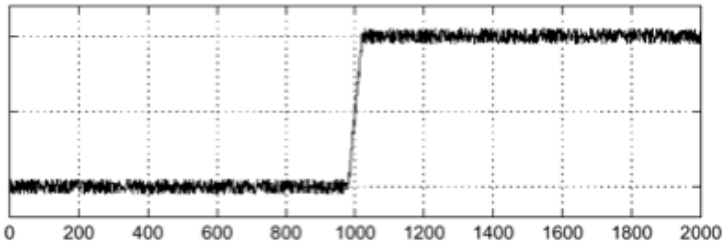


horizontal derivative

How does the gradient direction relate to these edges?

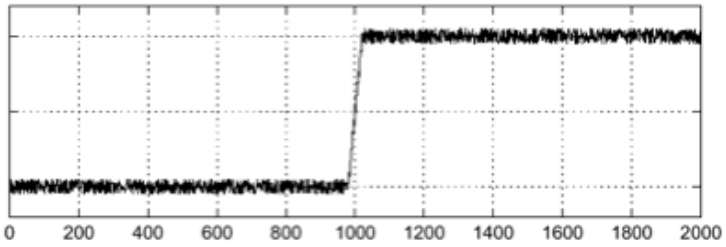
How do you find the edge of this signal?

intensity plot



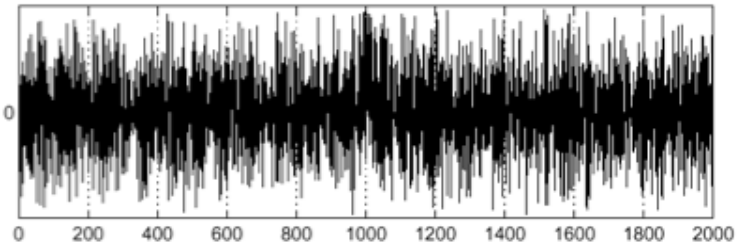
How do you find the edge of this signal?

intensity plot



Using a derivative filter:

derivative plot



What's the problem here?

Differentiation is very sensitive to noise

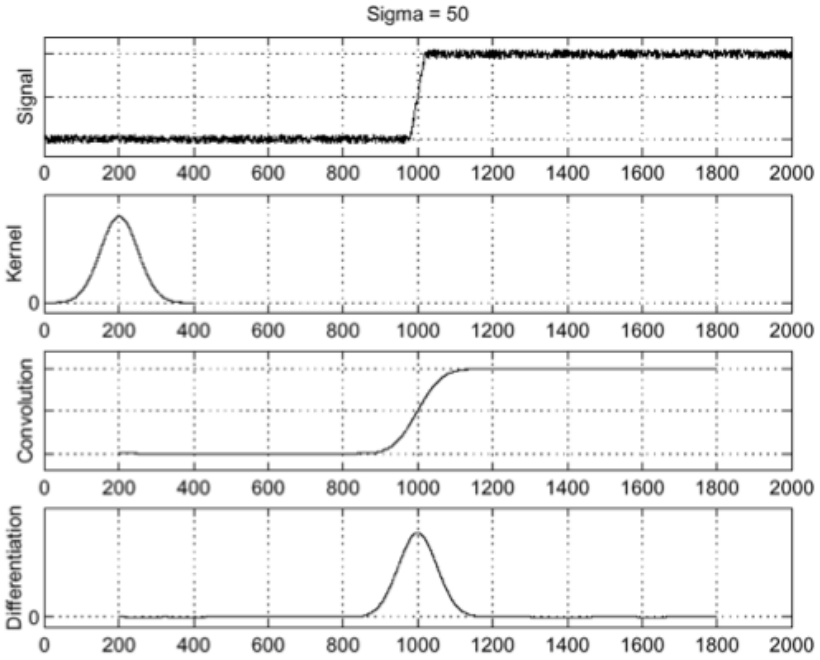
When using derivative filters, it is critical to blur first!

input

Gaussian

blurred

derivative of blurred



How much should we blur?

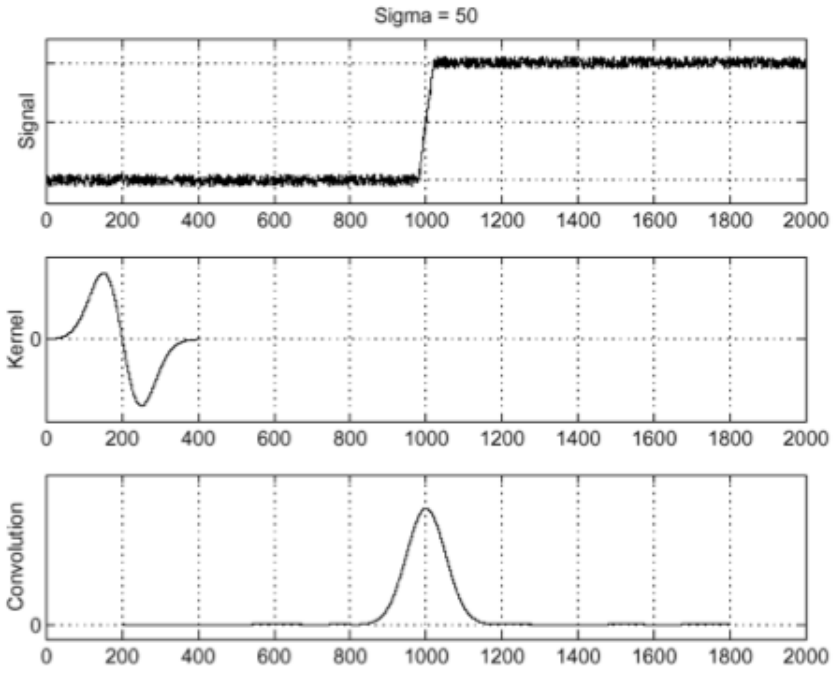
Derivative of Gaussian (DoG) filter

Derivative theorem of convolution: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

input

derivative of Gaussian

output (same as before)



- How many operations did we save?
- Any other advantages beyond efficiency?

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

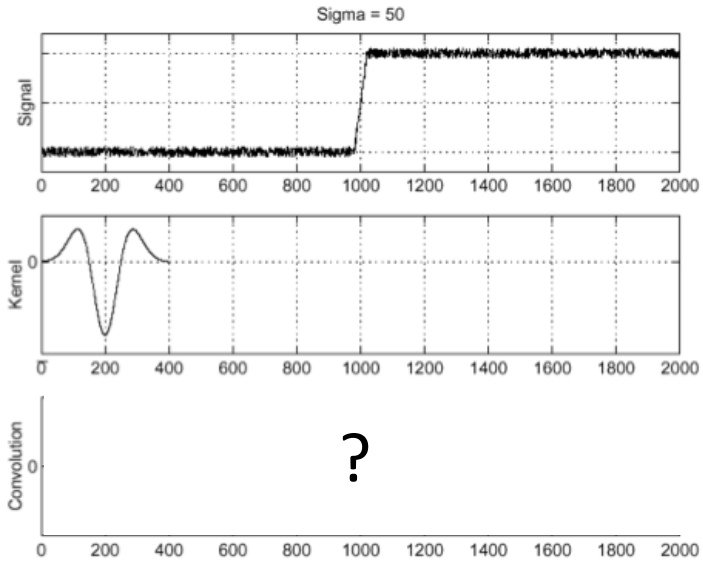
Laplacian of Gaussian (LoG) filter

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

input

Laplacian of Gaussian

output



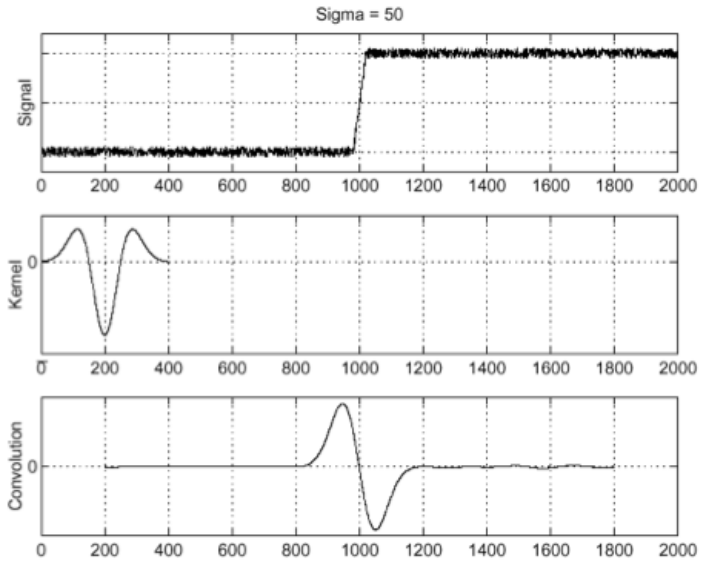
Laplacian of Gaussian (LoG) filter

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

input

Laplacian of Gaussian

output



"zero crossings" at edges

Laplace and LoG filtering examples

Laplacian of Gaussian filtering

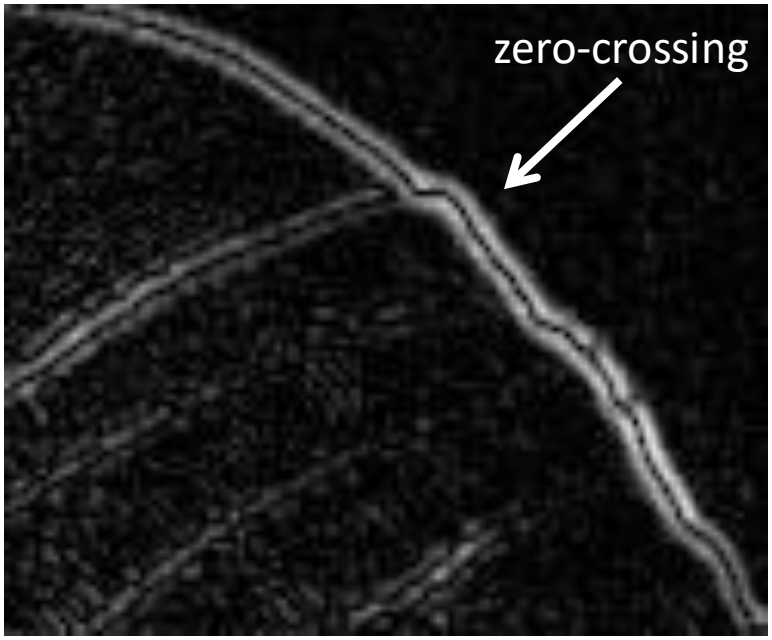
Laplace filtering

Laplacian of Gaussian vs Derivative of Gaussian

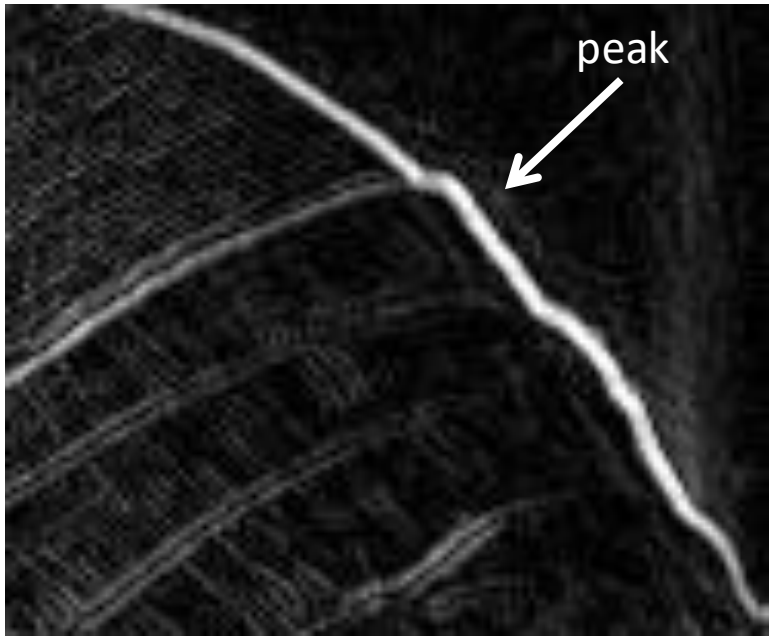
Laplacian of Gaussian filtering

Derivative of Gaussian filtering

Laplacian of Gaussian vs Derivative of Gaussian



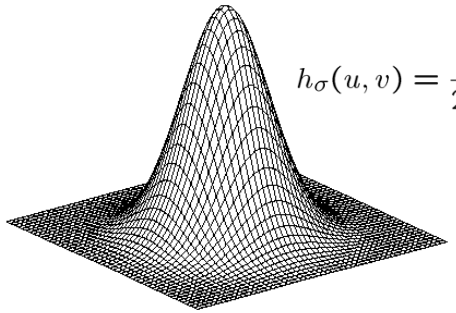
Laplacian of Gaussian filtering



Derivative of Gaussian filtering

Zero crossings are more accurate at localizing edges
(but not very convenient).

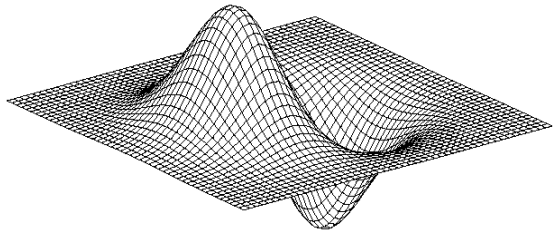
2D Gaussian filters



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

Gaussian

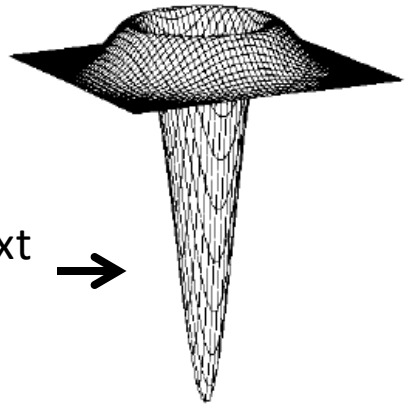
$$\frac{\partial}{\partial x} h_\sigma(u,v)$$



Derivative of Gaussian

how does this relate to next picture?

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



Laplacian of Gaussian