Image filtering

http://16385.courses.cs.cmu.edu/

16-385 Computer Vision
Fall 2020, Lecture 2

Overview of today’s lecture

- Types of image transformations.
- Point image processing.
- Linear shift-invariant image filtering.
- Convolution.
- Image gradients.

Slide credits

Most of these slides were adapted directly from:

- Kris Kitani (15-463, Fall 2016).

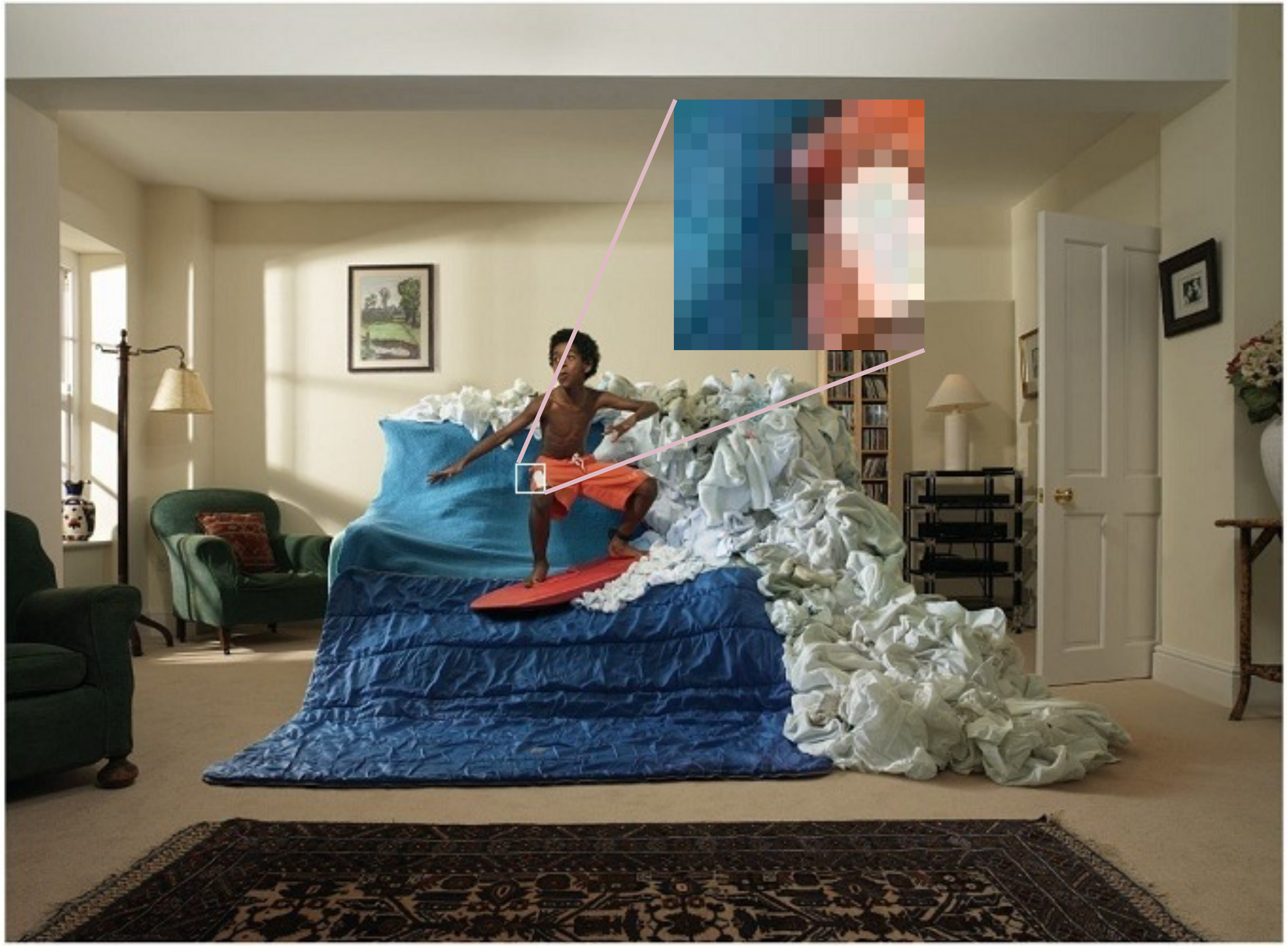
Inspiration and some examples also came from:

- Fredo Durand (Digital and Computational Photography, MIT).
- Kayvon Fatahalian (15-769, Fall 2016).

Types of image transformations

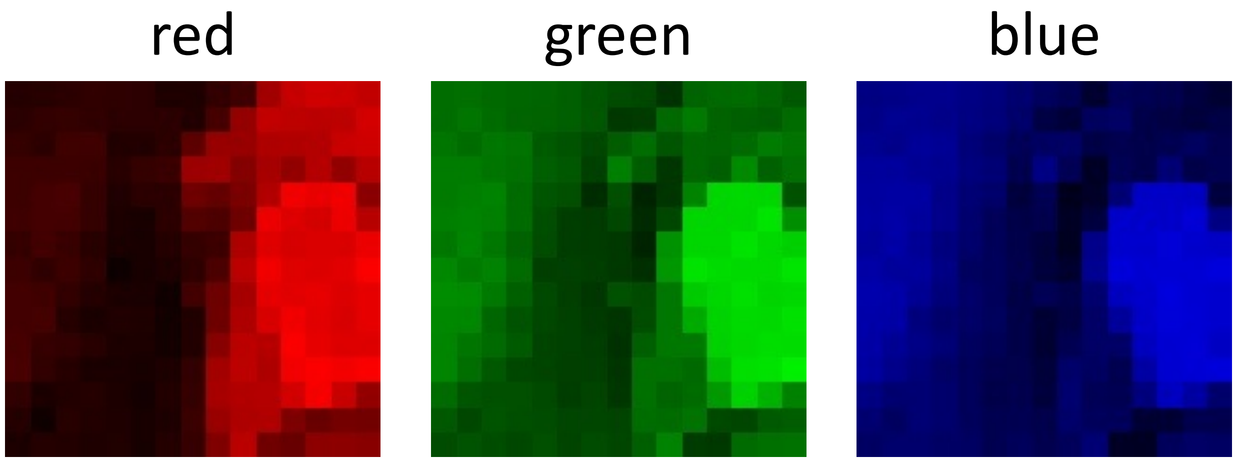
What is an image?

What is an image?

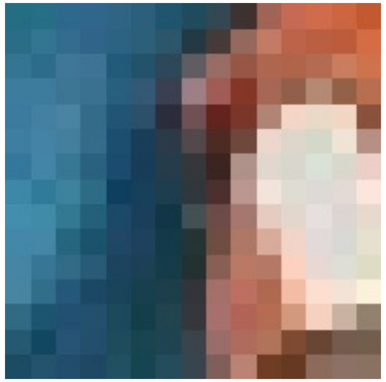


A (color) image is a 3D tensor of numbers.

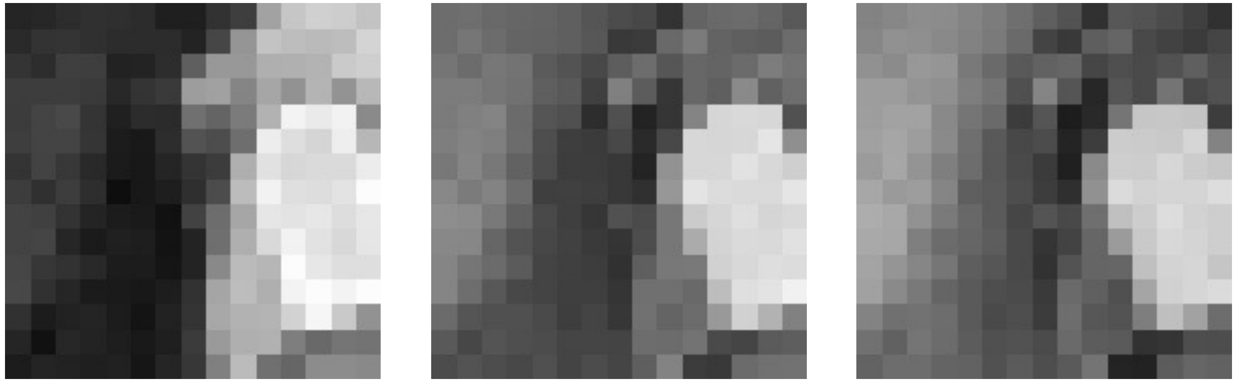
What is an image?



colorized for visualization



color image patch



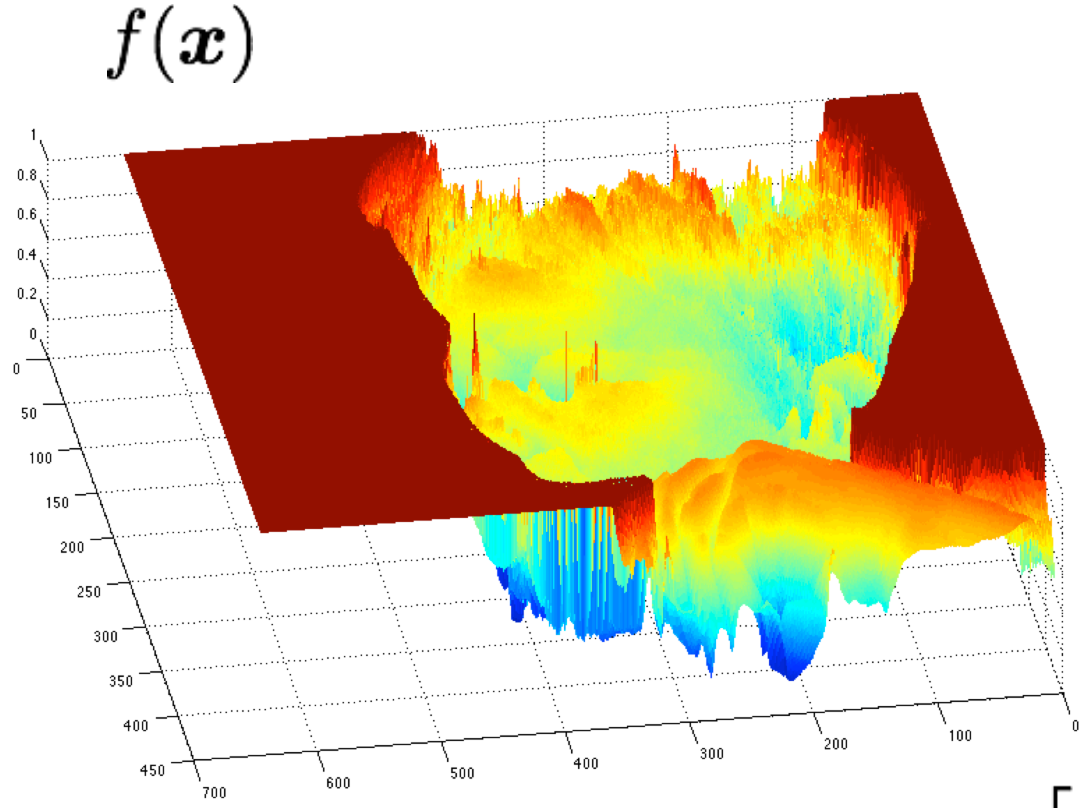
actual intensity values per channel

Each channel is a 2D array of numbers.

How many bits are the intensity values?

What is an image?

grayscale image



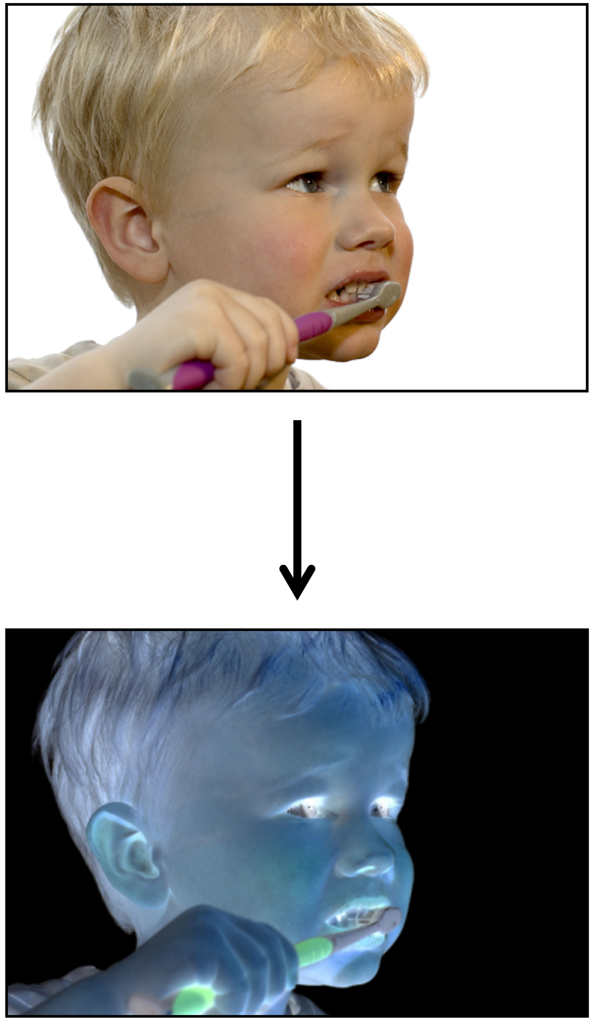
domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

A (grayscale) image is a 2D function.

What is the range of the image function f?

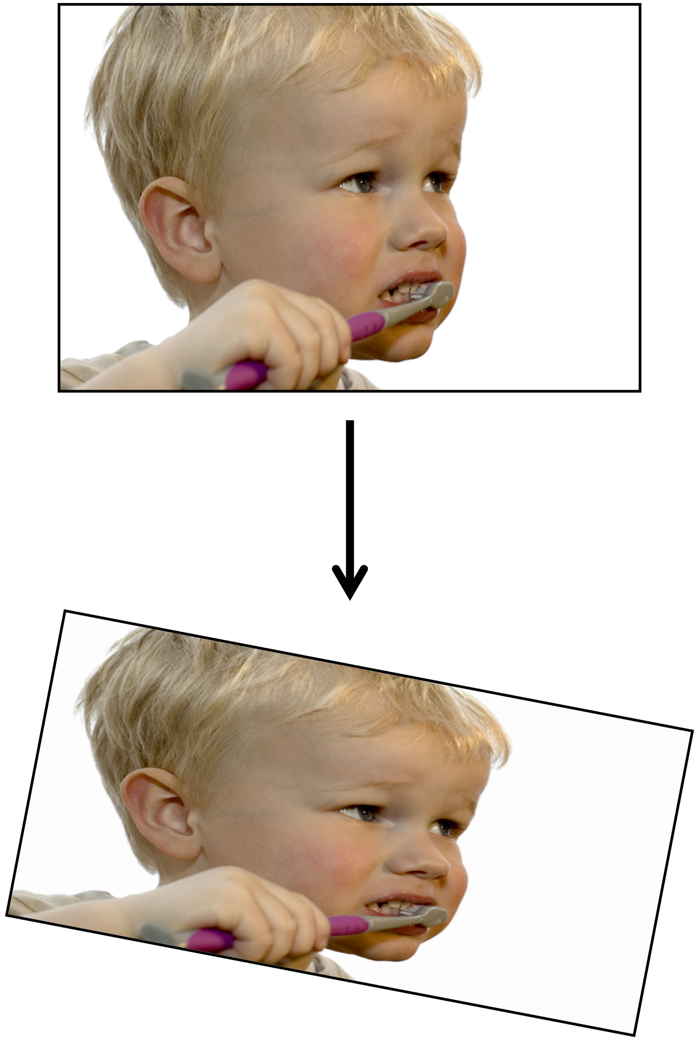
What types of image transformations can we do?

Filtering



changes pixel *values*

Warping



changes pixel *locations*

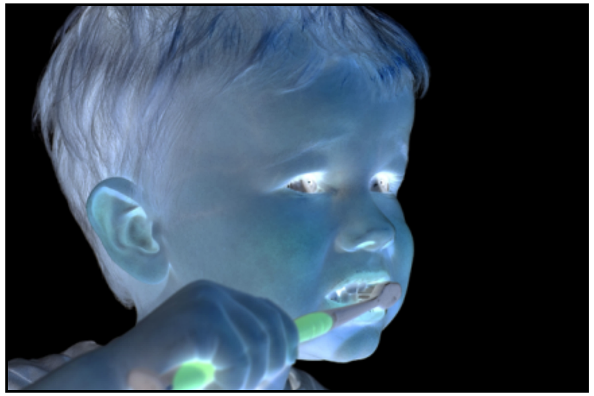
What types of image transformations can we do?

F

Filtering

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

G



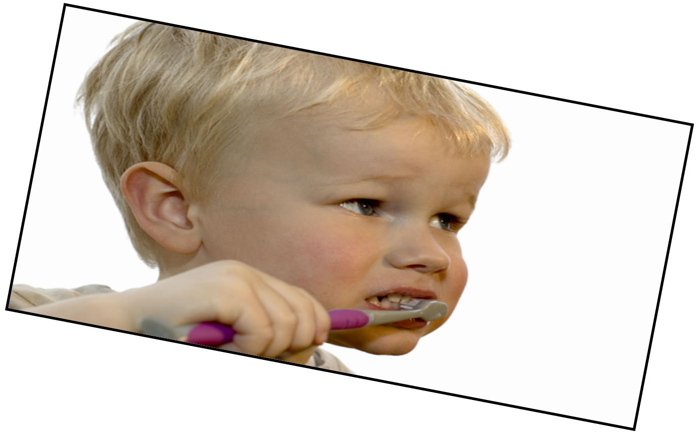
changes *range* of image function

F

Warping

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

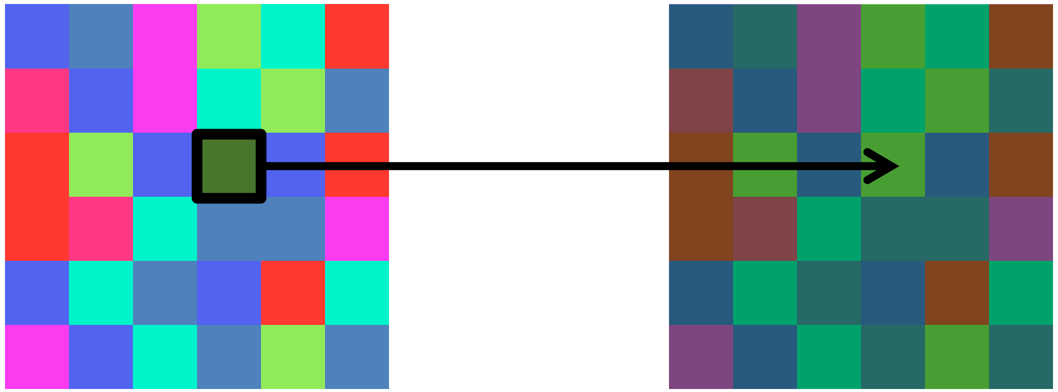
G



changes *domain* of image function

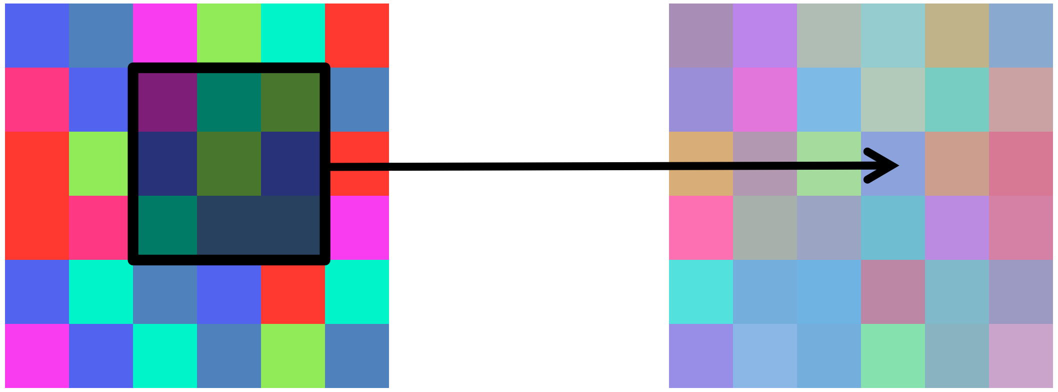
What types of image filtering can we do?

Point Operation



point processing

Neighborhood Operation

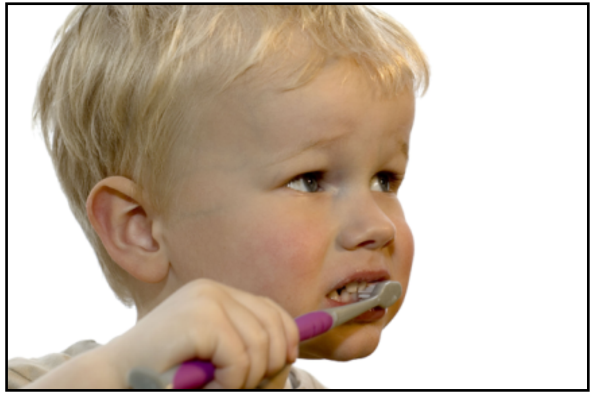


“filtering”

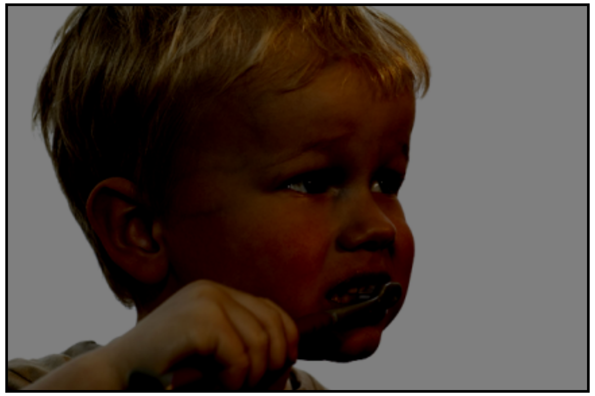
Point processing

Examples of point processing

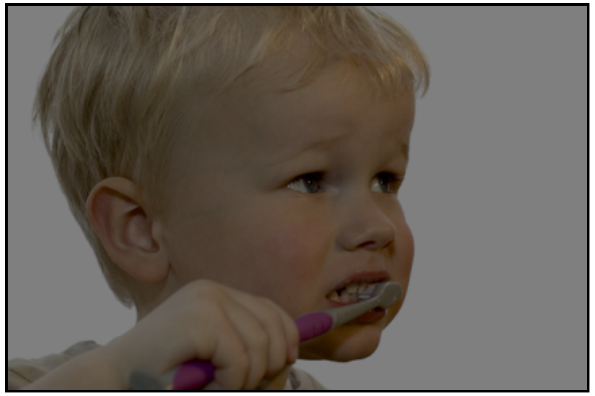
original



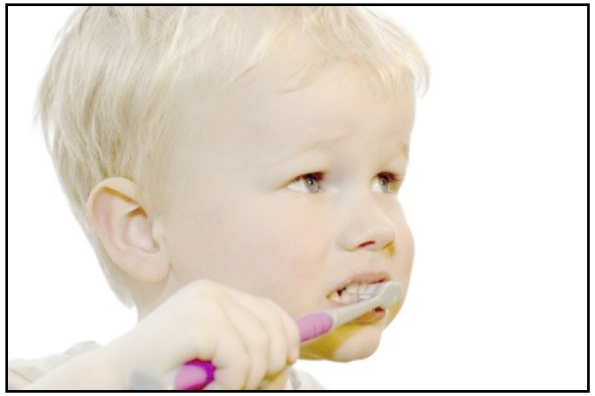
darken



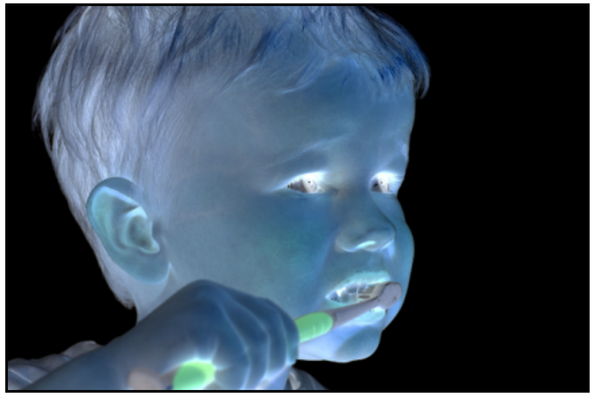
lower contrast



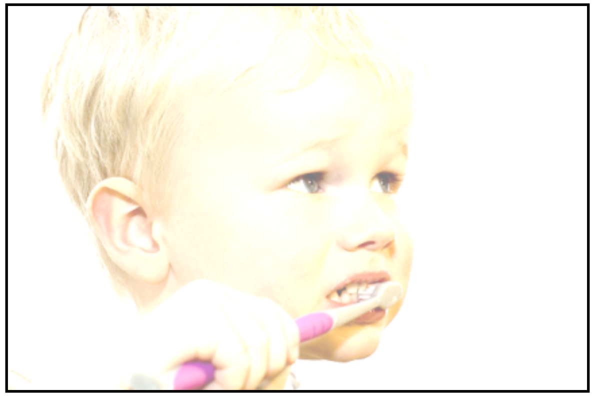
non-linear lower contrast



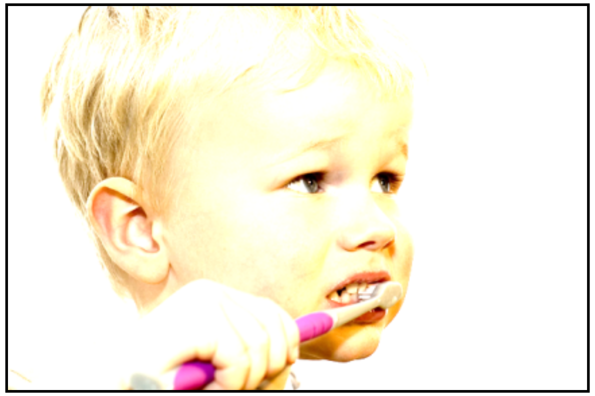
invert



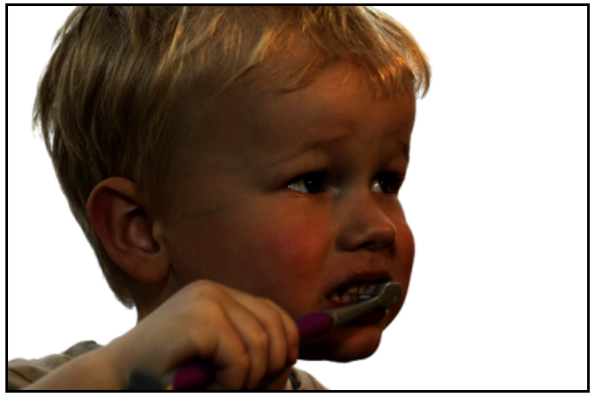
lighten



raise contrast



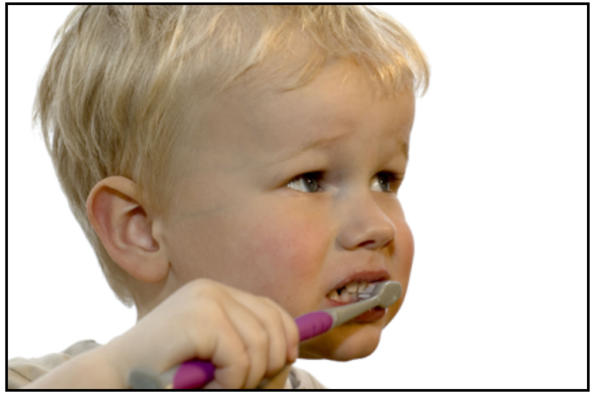
non-linear raise contrast



How would you implement these?

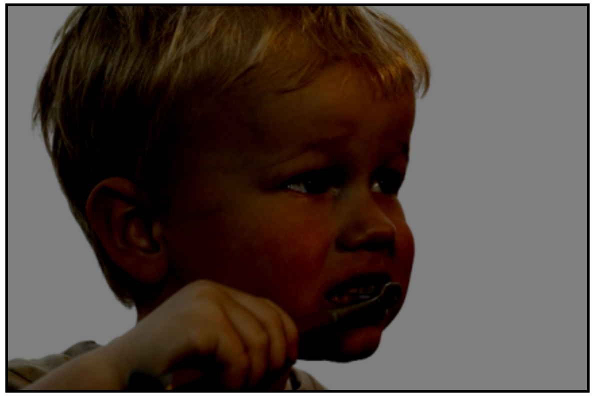
Examples of point processing

original

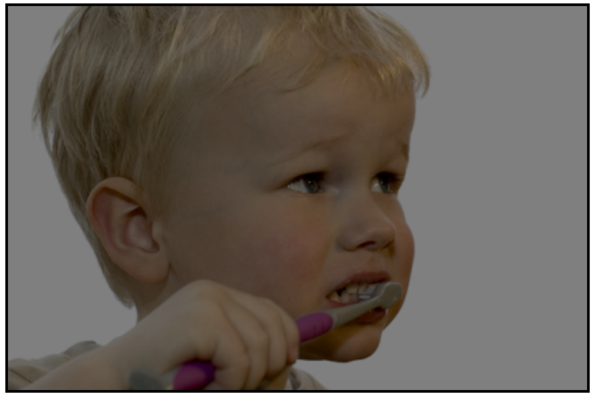


x

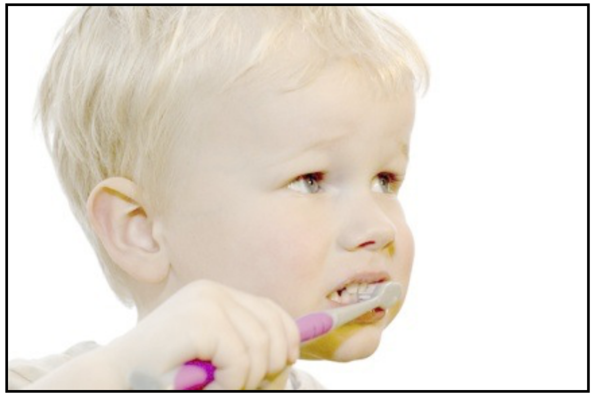
darken



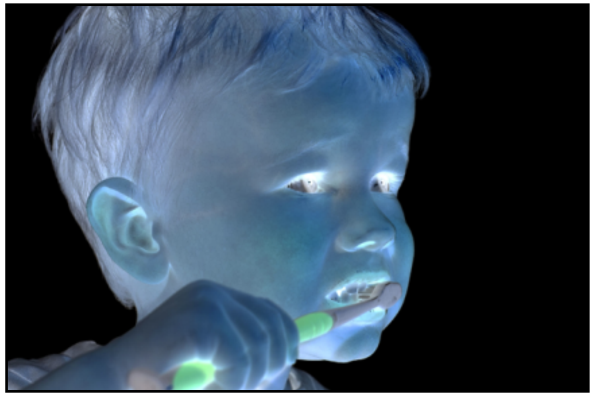
lower contrast



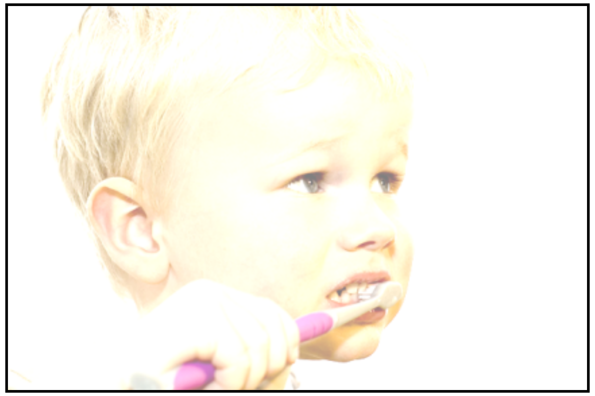
non-linear lower contrast



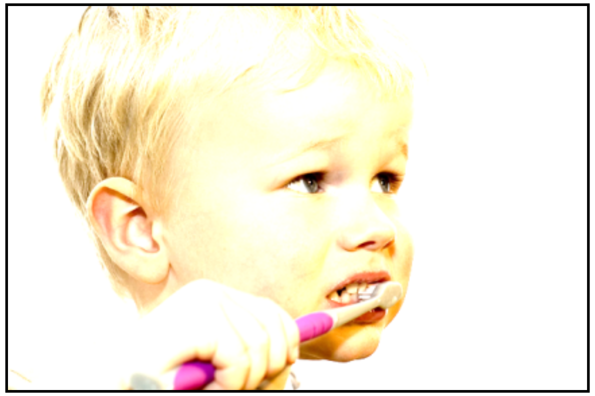
invert



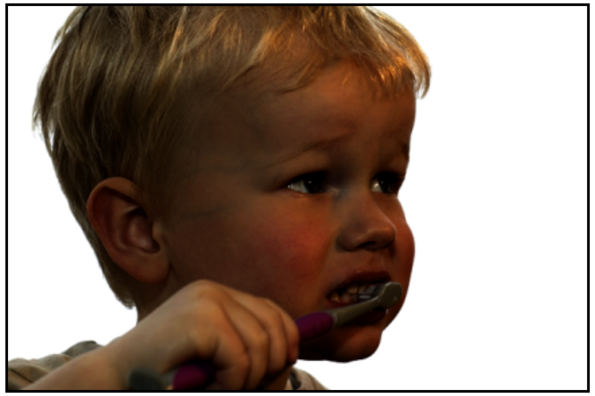
lighten



raise contrast



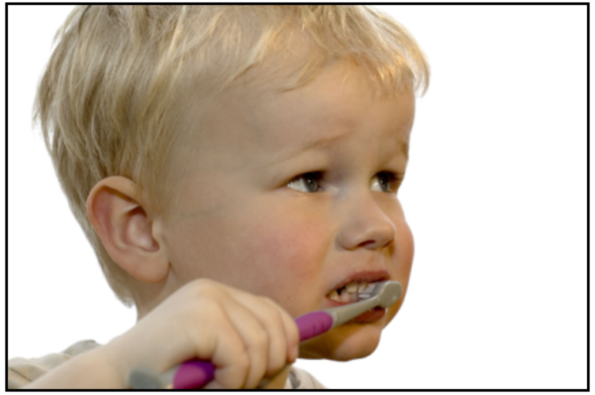
non-linear raise contrast



Examples of point processing

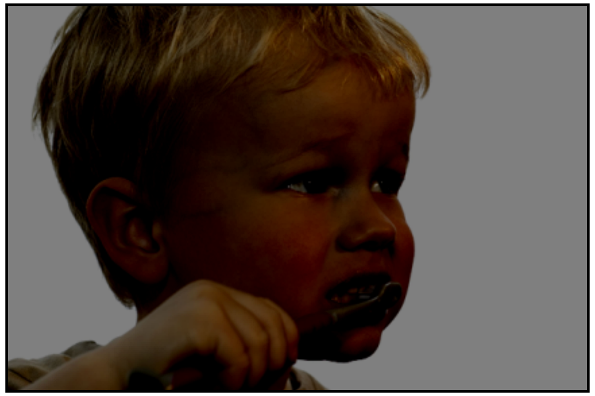
How would you implement these?

original



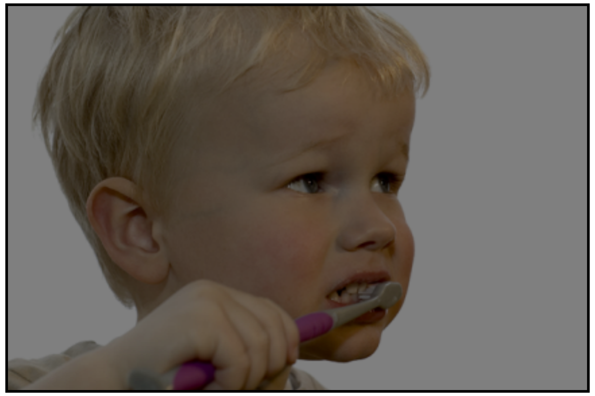
x

darken

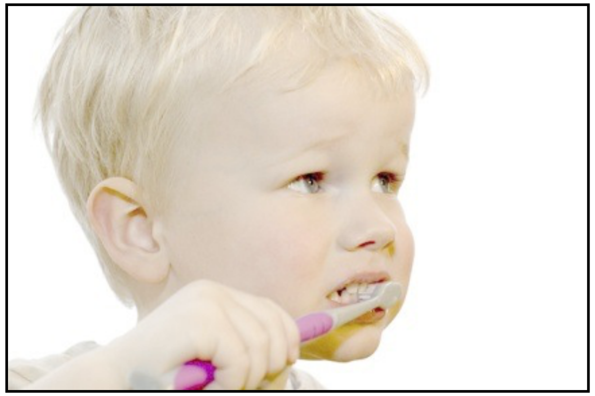


$x - 128$

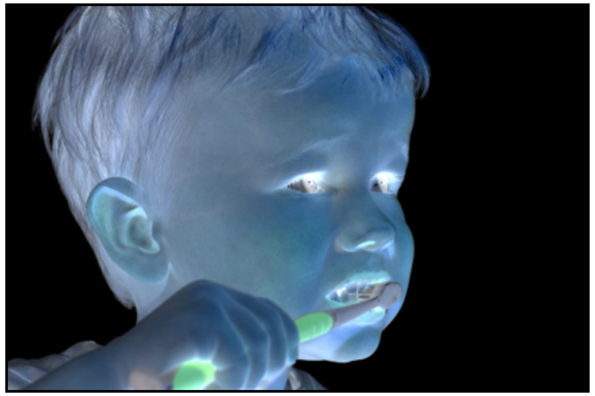
lower contrast



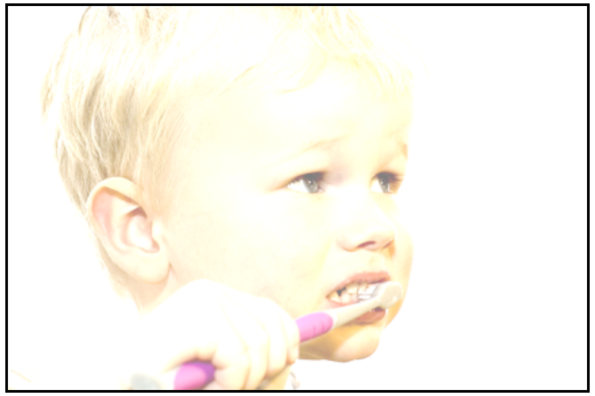
non-linear lower contrast



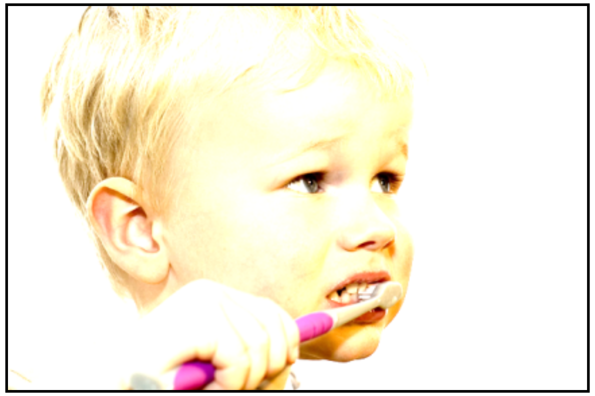
invert



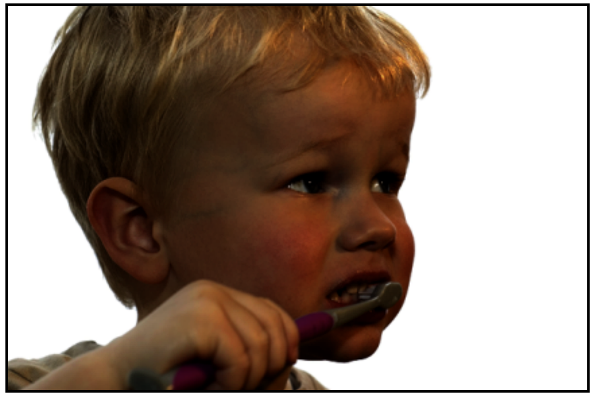
lighten



raise contrast



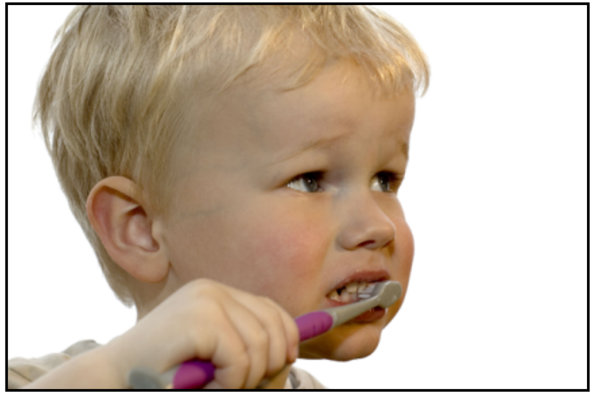
non-linear raise contrast



How would you implement these?

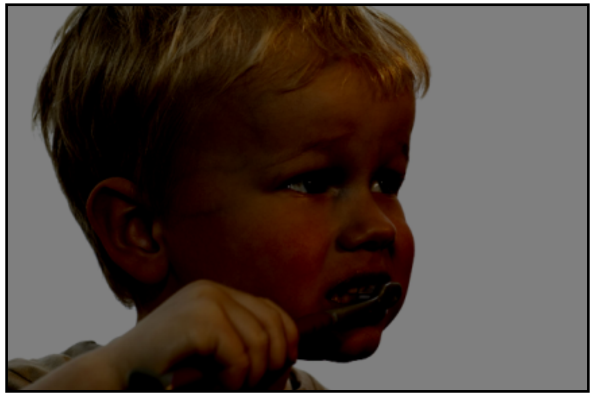
Examples of point processing

original



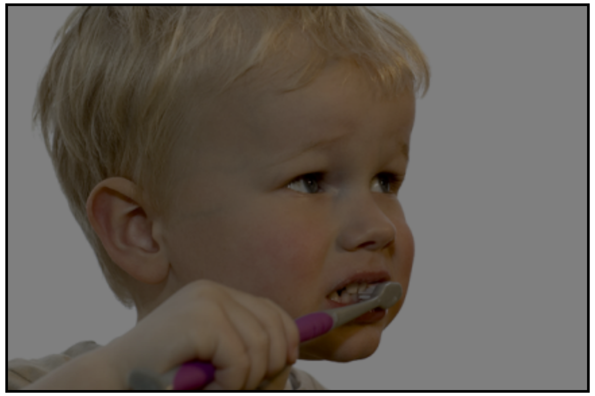
x

darken



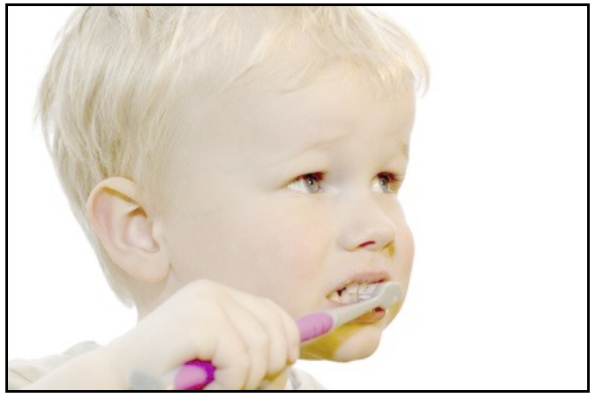
$x - 128$

lower contrast

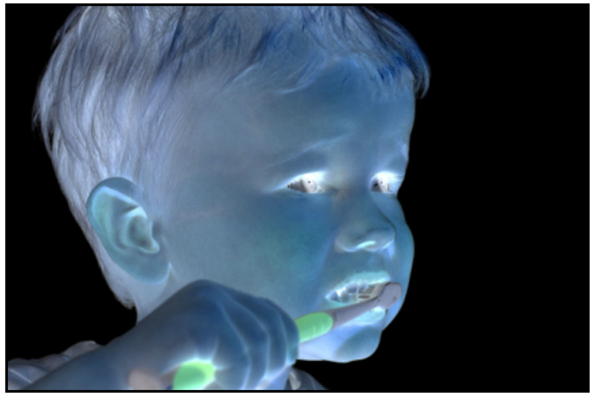


$\frac{x}{2}$

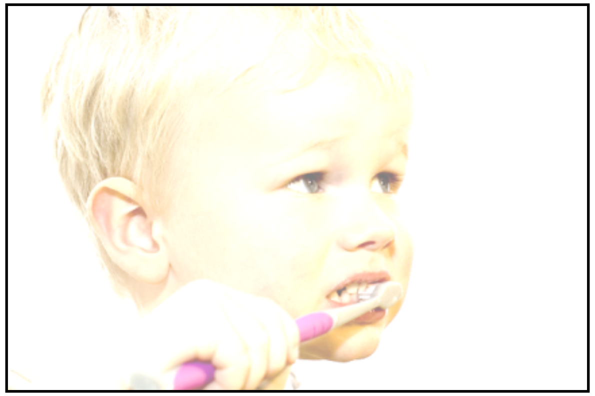
non-linear lower contrast



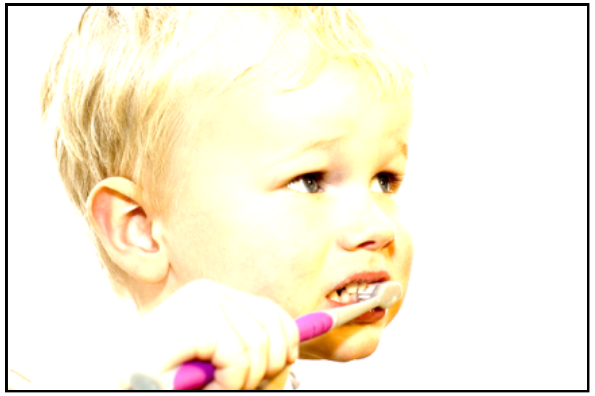
invert



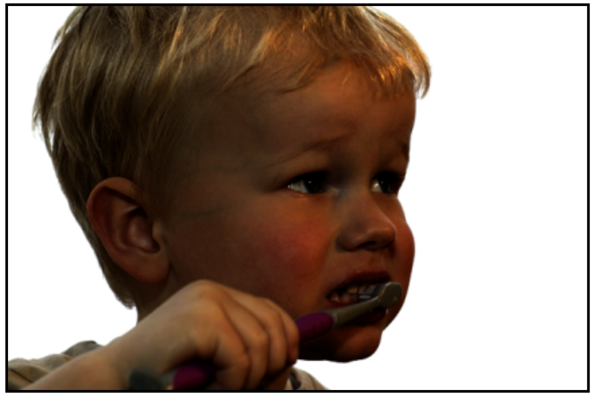
lighten



raise contrast



non-linear raise contrast



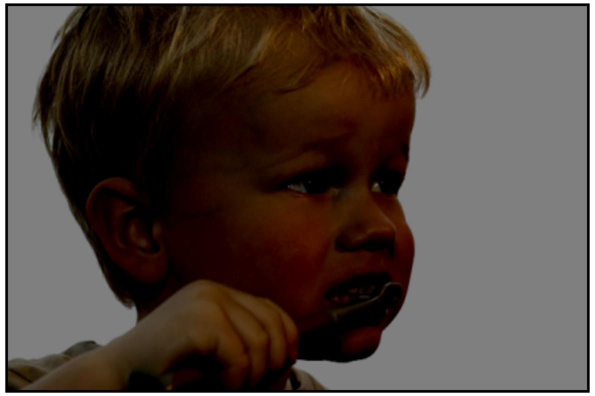
How would you implement these?

Examples of point processing

original

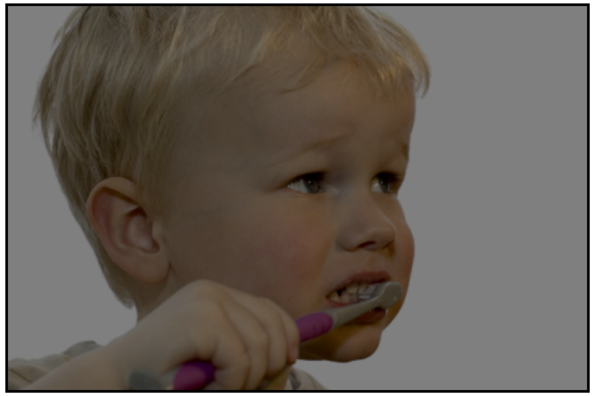
$$x$$

darken



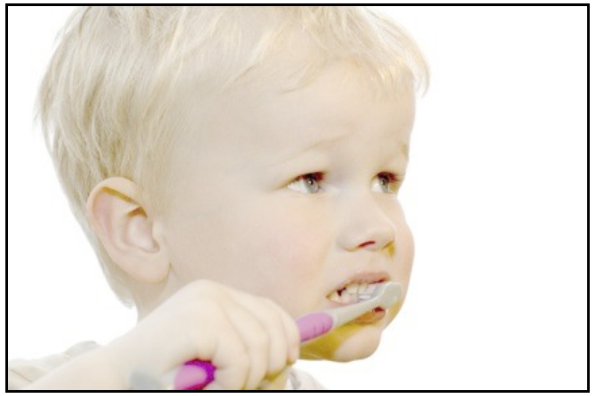
$$x - 128$$

lower contrast



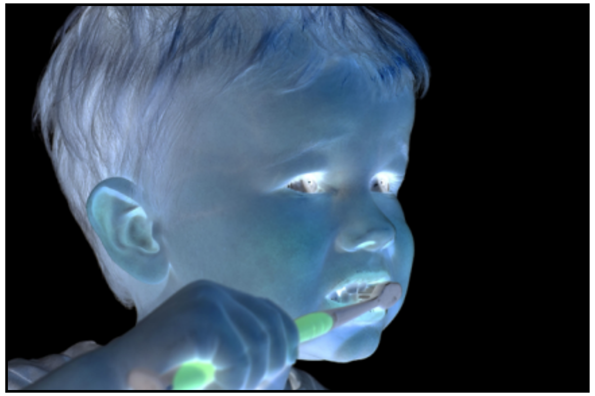
$$\frac{x}{2}$$

non-linear lower contrast

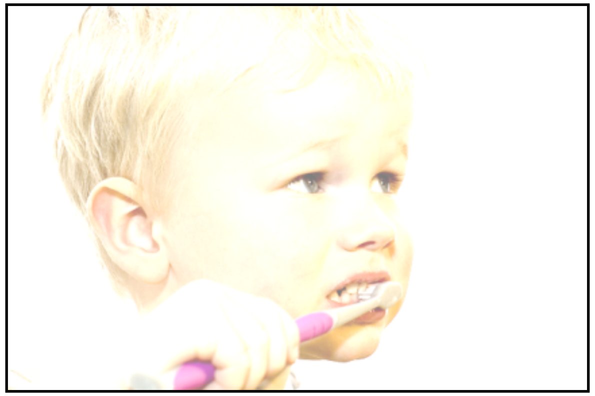


$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

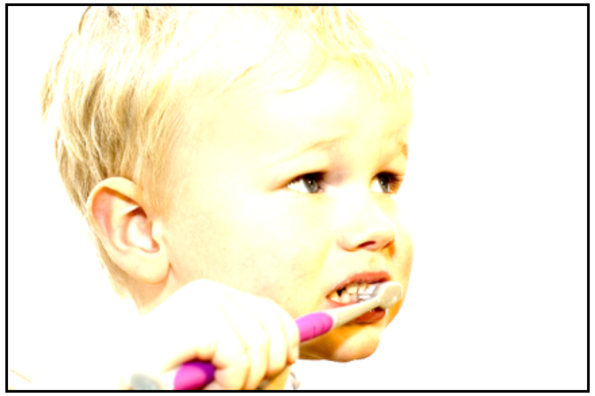
invert



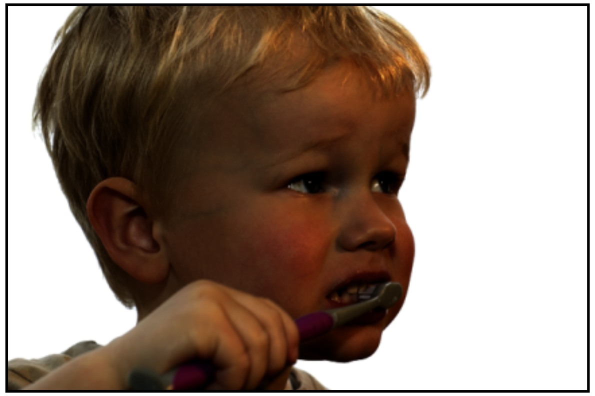
lighten



raise contrast



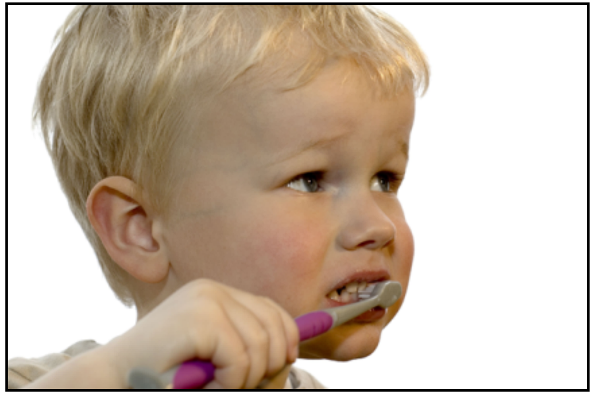
non-linear raise contrast



How would you implement these?

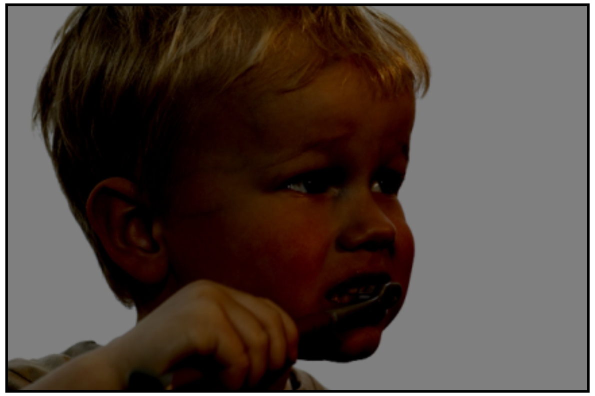
Examples of point processing

original



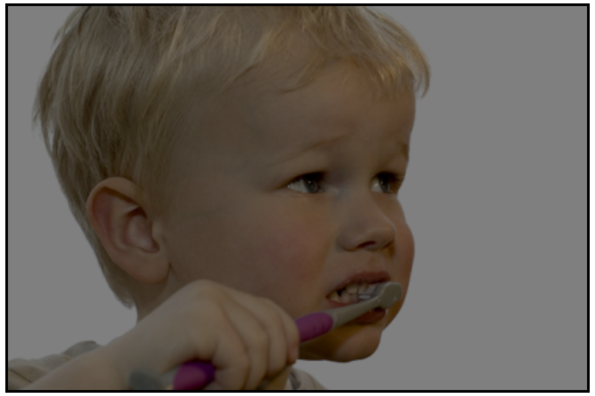
x

darken



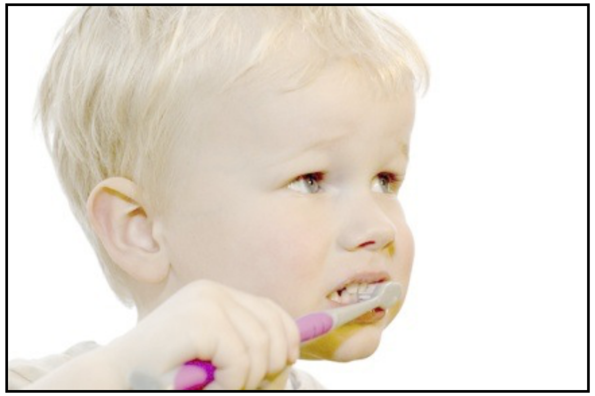
$x - 128$

lower contrast



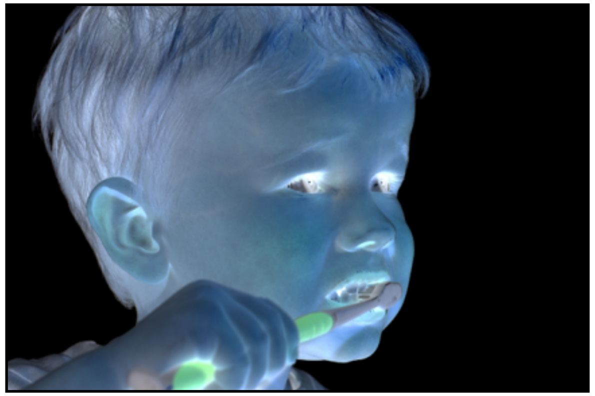
$\frac{x}{2}$

non-linear lower contrast



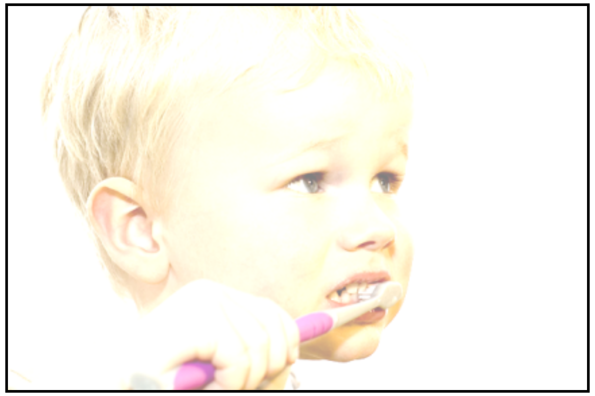
$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert

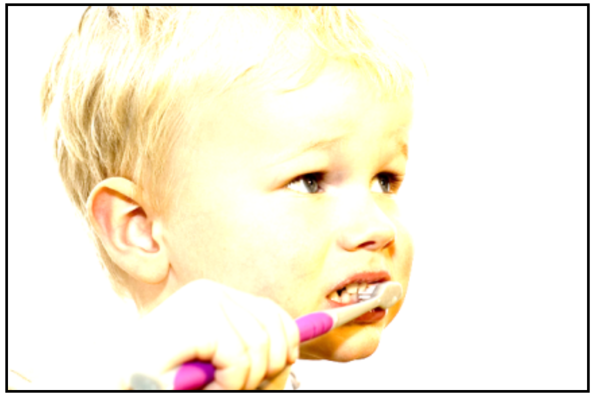


$255 - x$

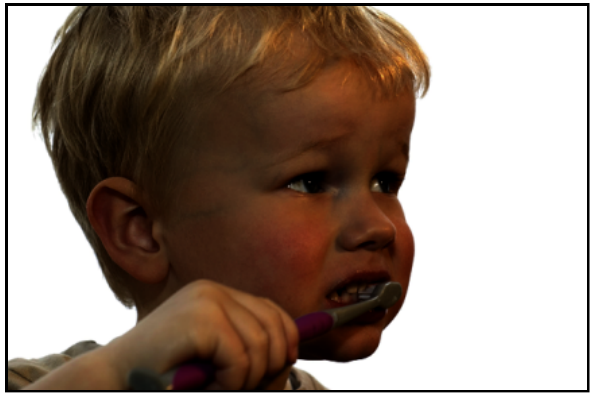
lighten



raise contrast



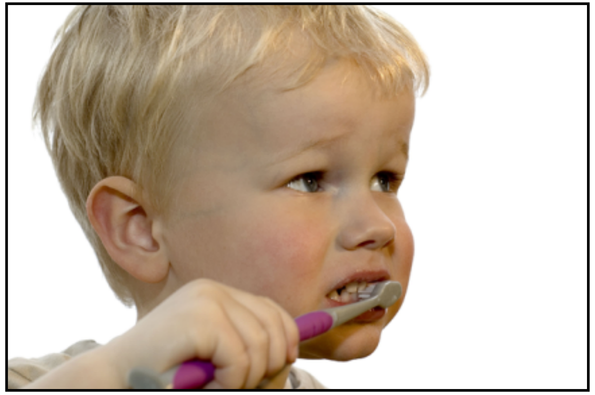
non-linear raise contrast



How would you implement these?

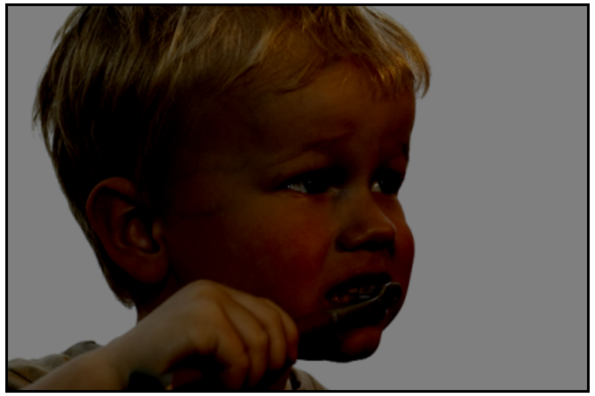
Examples of point processing

original



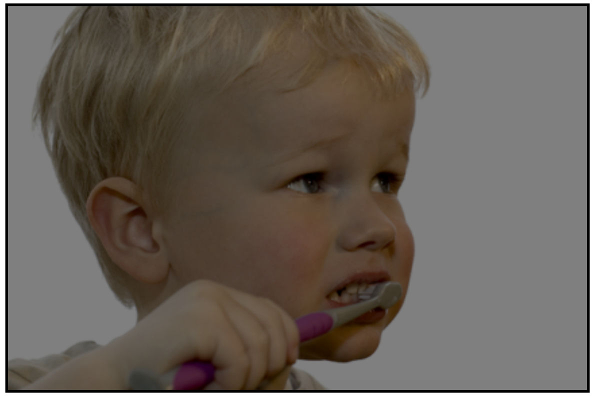
$$x$$

darken



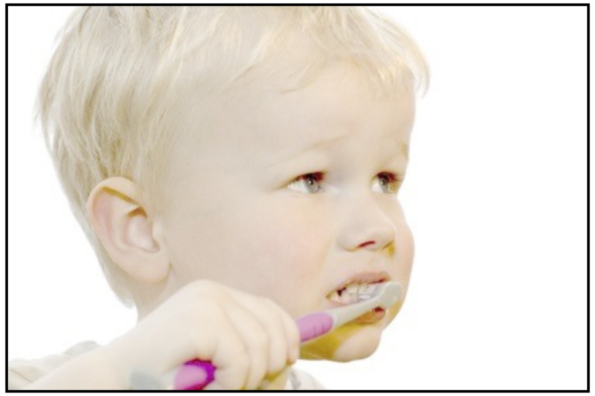
$$x - 128$$

lower contrast



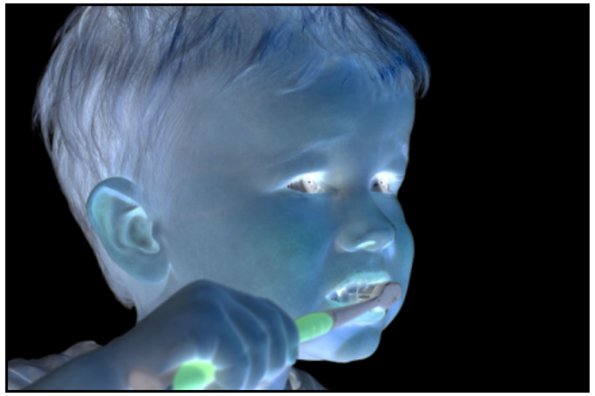
$$\frac{x}{2}$$

non-linear lower contrast



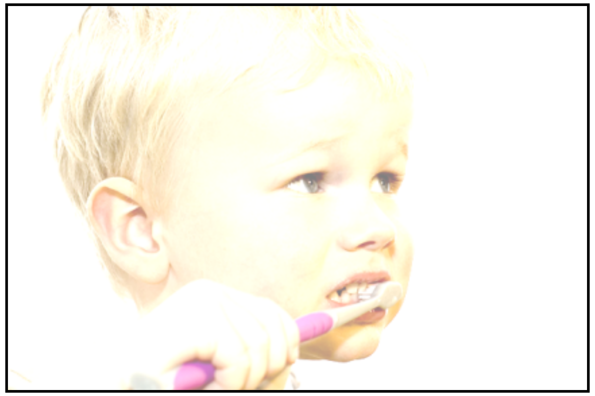
$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert



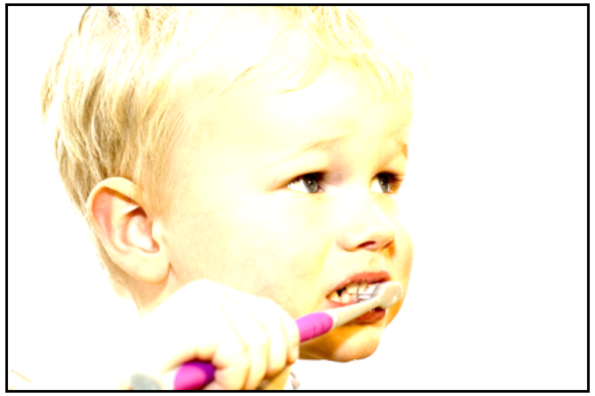
$$255 - x$$

lighten

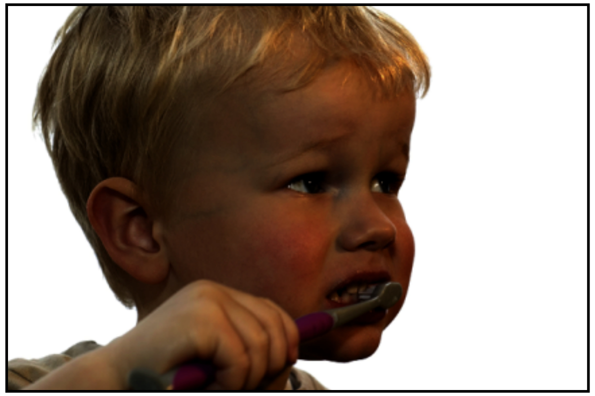


$$x + 128$$

raise contrast



non-linear raise contrast



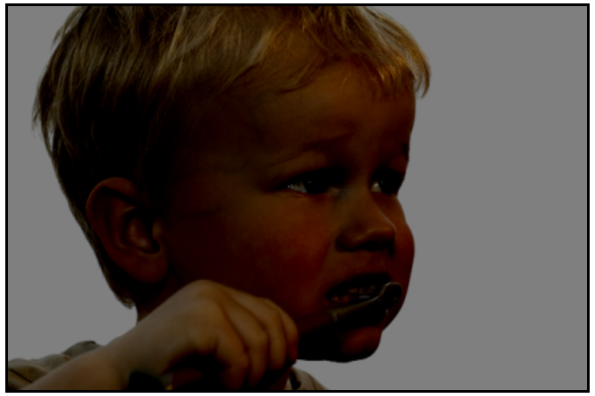
How would you implement these?

Examples of point processing

original

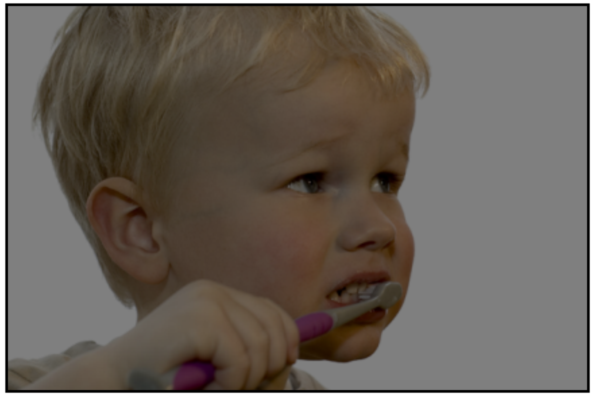
$$x$$

darken



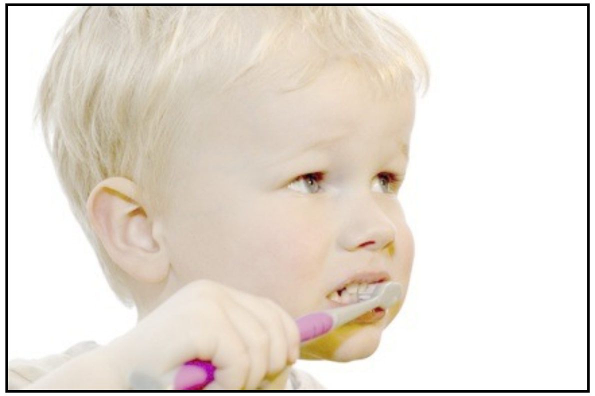
$$x - 128$$

lower contrast



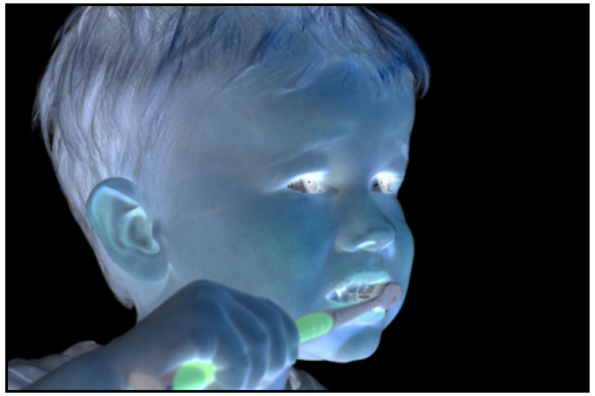
$$\frac{x}{2}$$

non-linear lower contrast



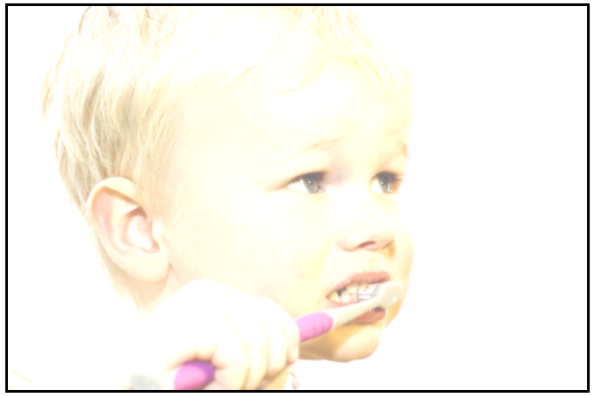
$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert



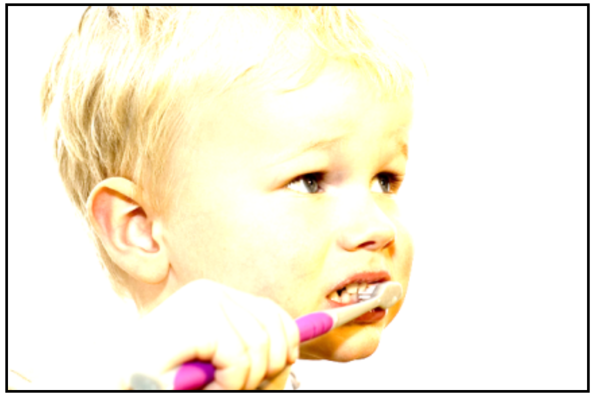
$$255 - x$$

lighten



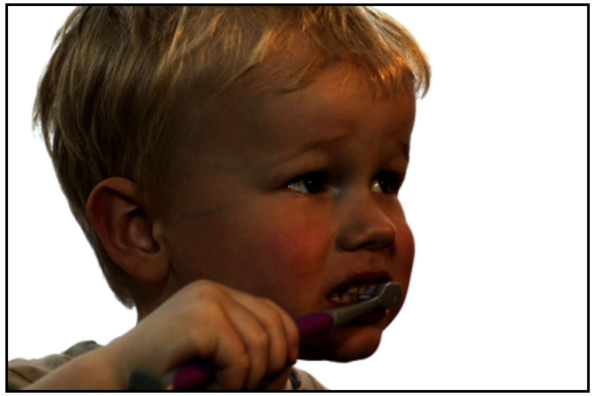
$$x + 128$$

raise contrast



$$x \times 2$$

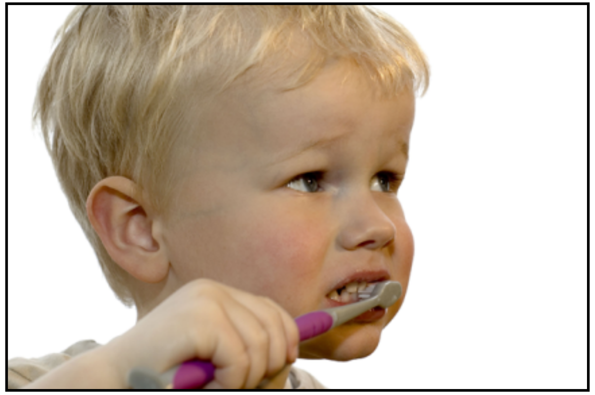
non-linear raise contrast



How would you implement these?

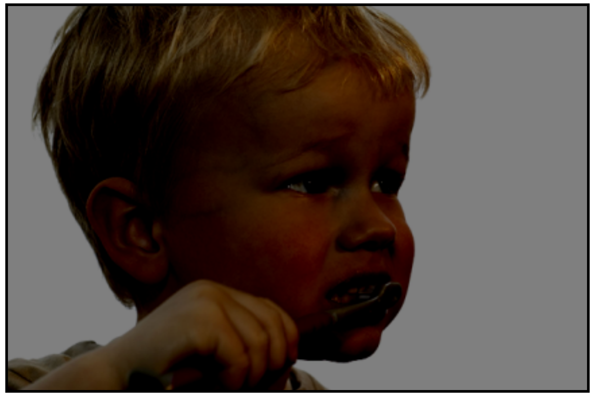
Examples of point processing

original



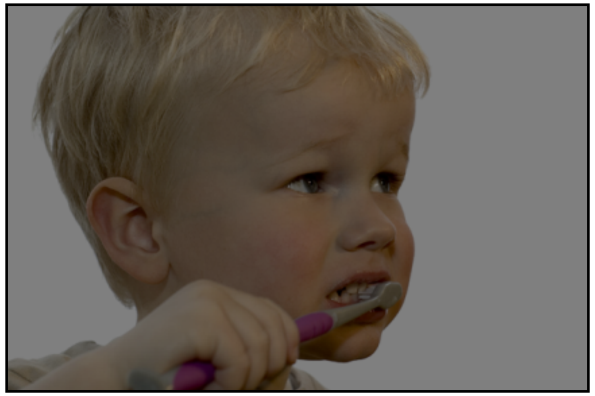
$$x$$

darken



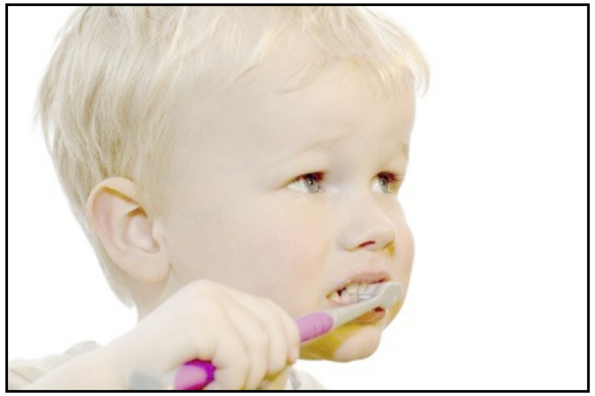
$$x - 128$$

lower contrast



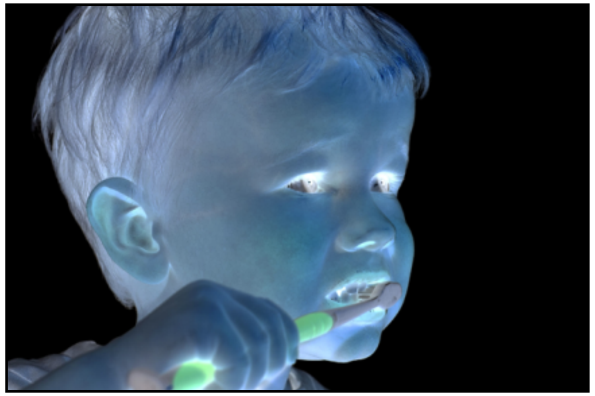
$$\frac{x}{2}$$

non-linear lower contrast



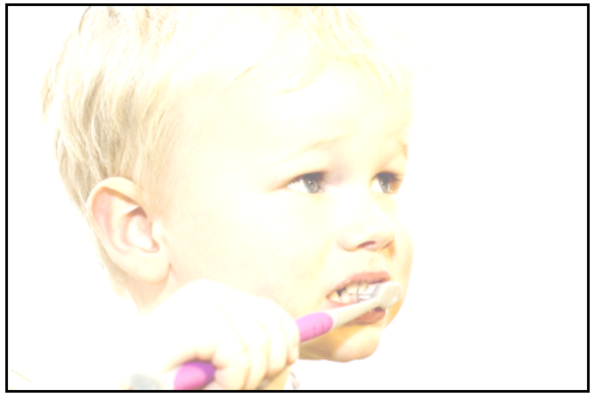
$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert



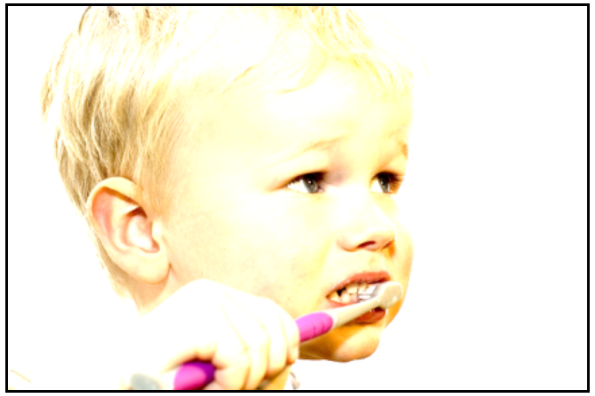
$$255 - x$$

lighten



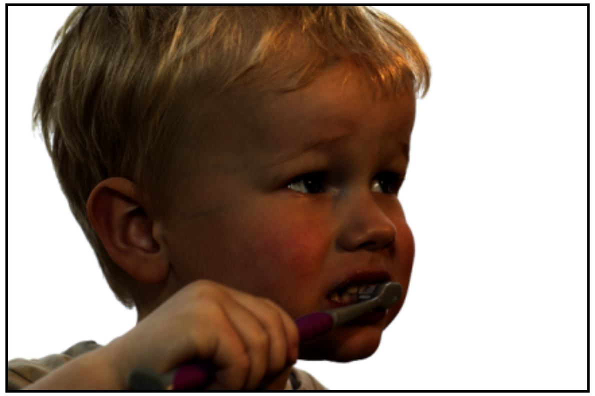
$$x + 128$$

raise contrast



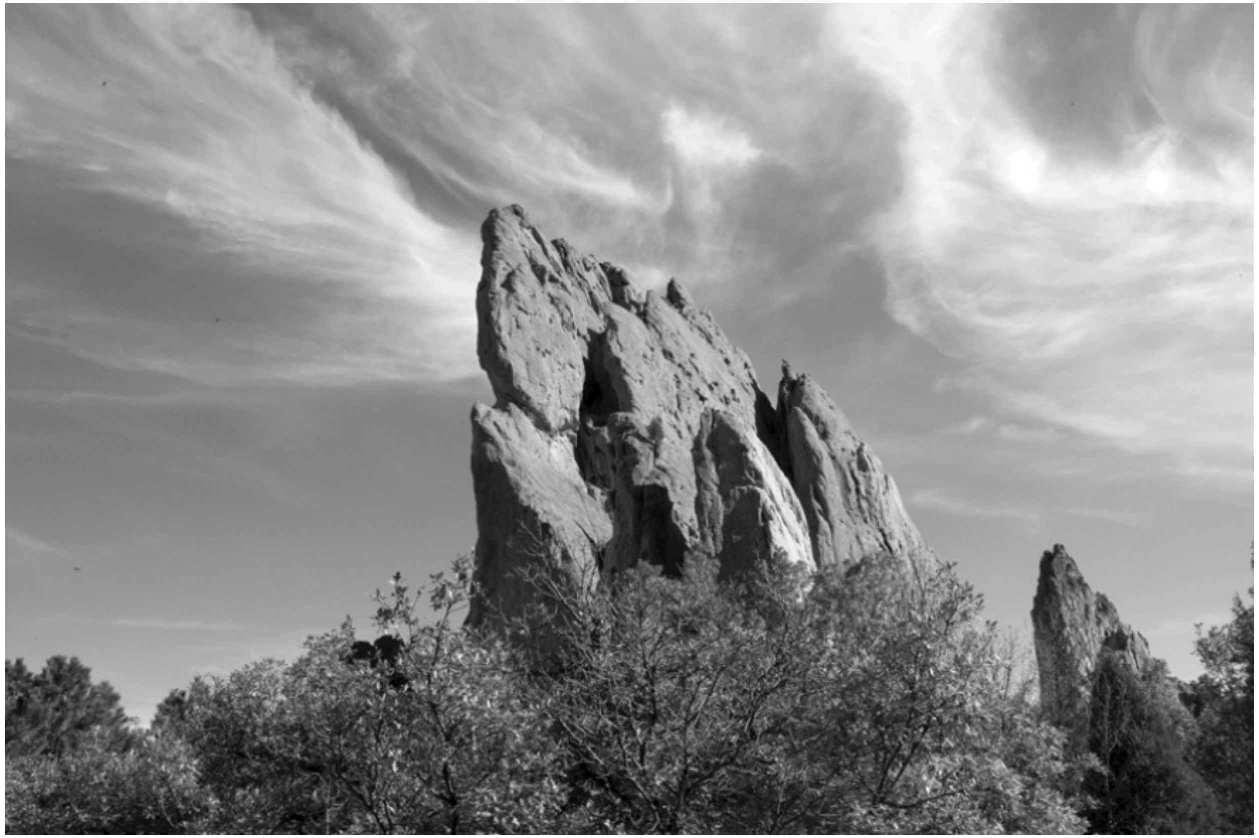
$$x \times 2$$

non-linear raise contrast



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

Many other types of point processing



camera output

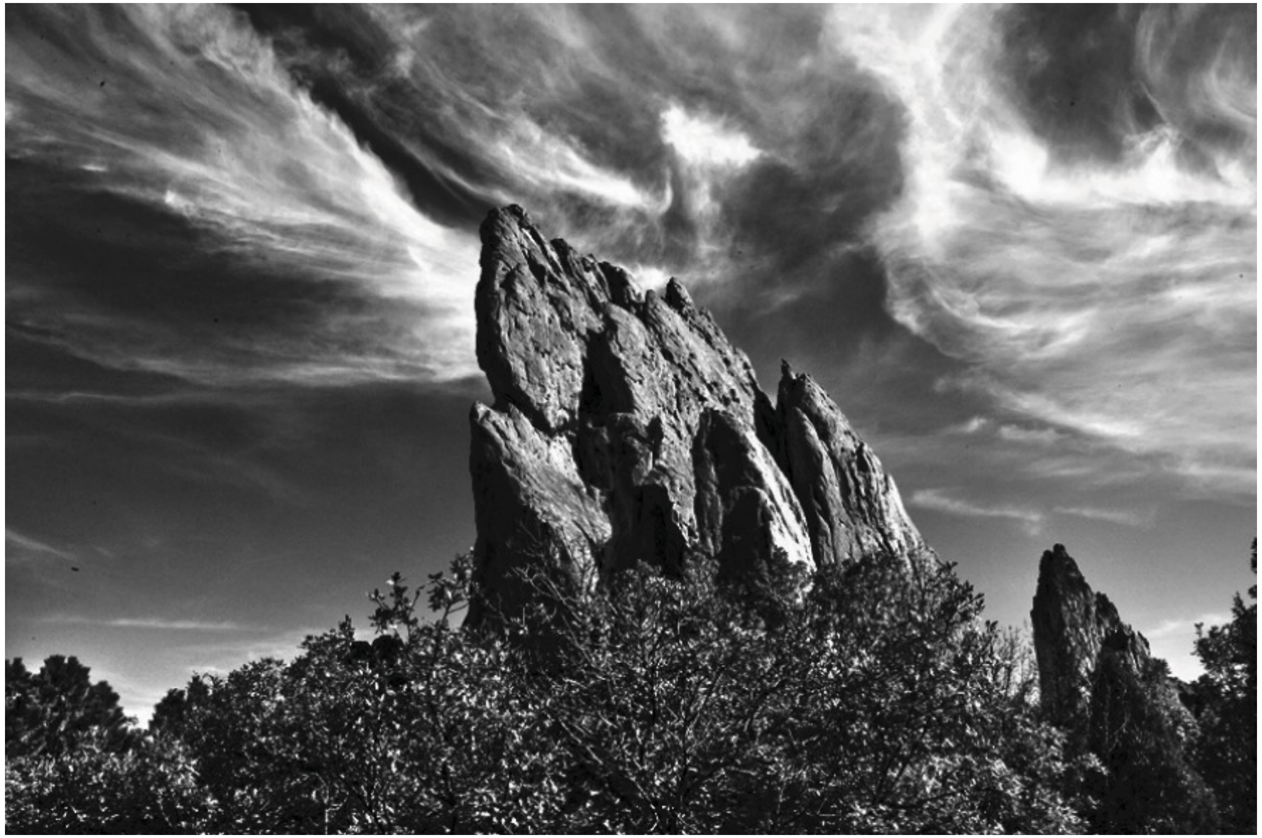
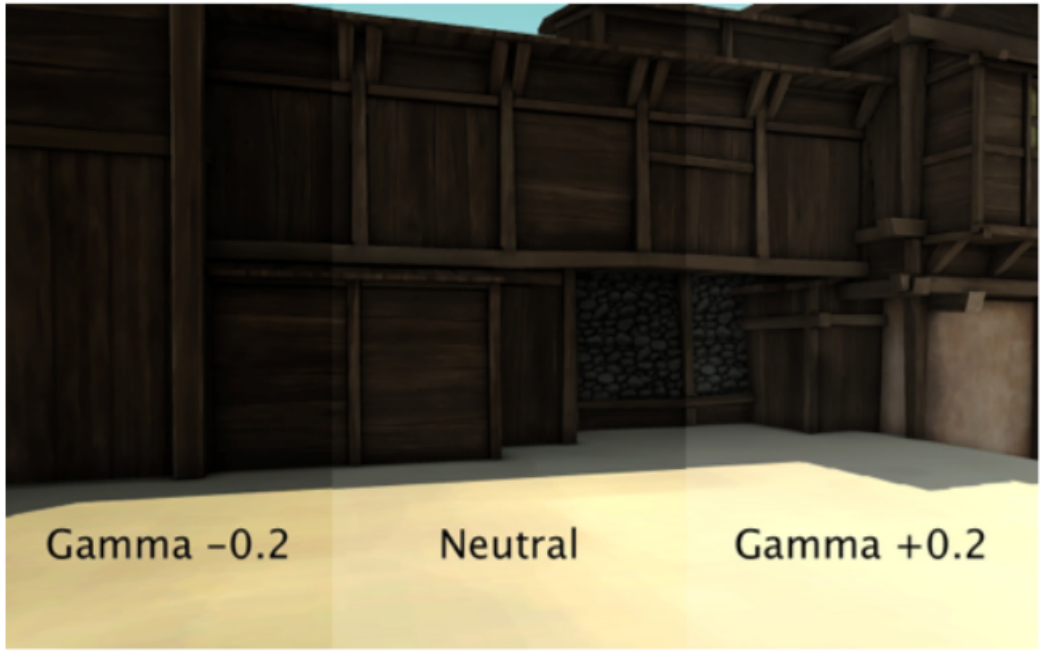


image after stylistic tonemapping

[Bae et al., SIGGRAPH 2006]

Many other types of point processing



Linear shift-invariant image filtering

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.

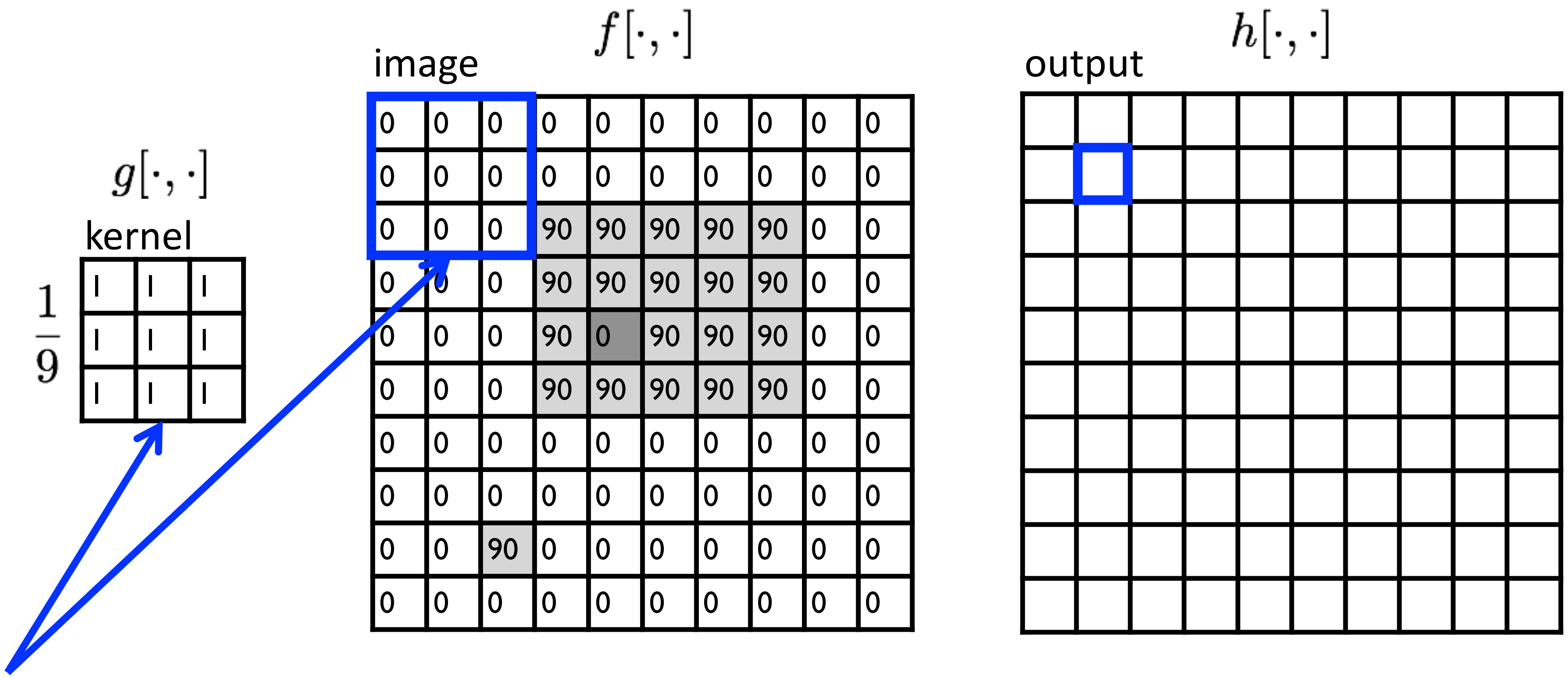
Example: the box filter

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

kernel $$g[\cdot,\cdot] = \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

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

Let’s run the box filter



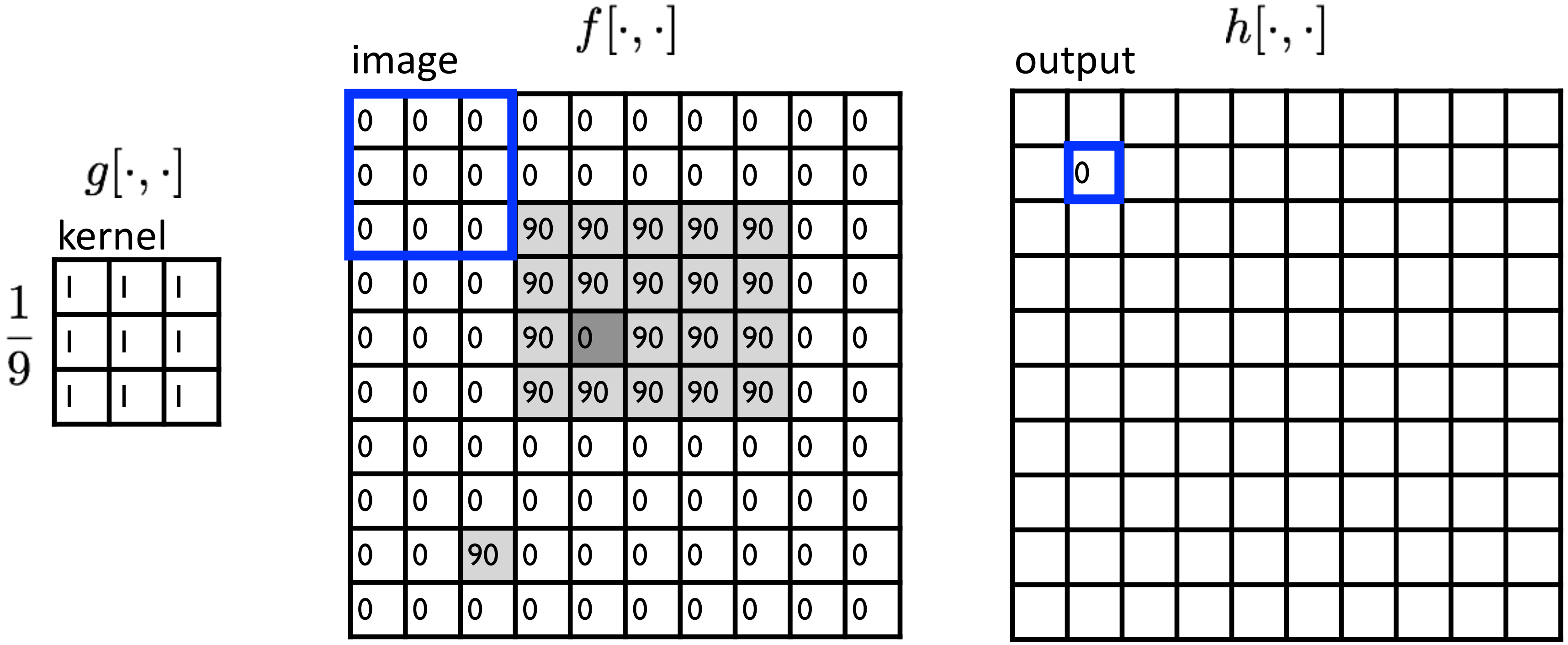
1	1	1
1	1	1
1	1	1

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

note that we assume that the kernel coordinates are centered

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

I	I	I
I	I	I
I	I	I

image $f[\cdot,\cdot]$

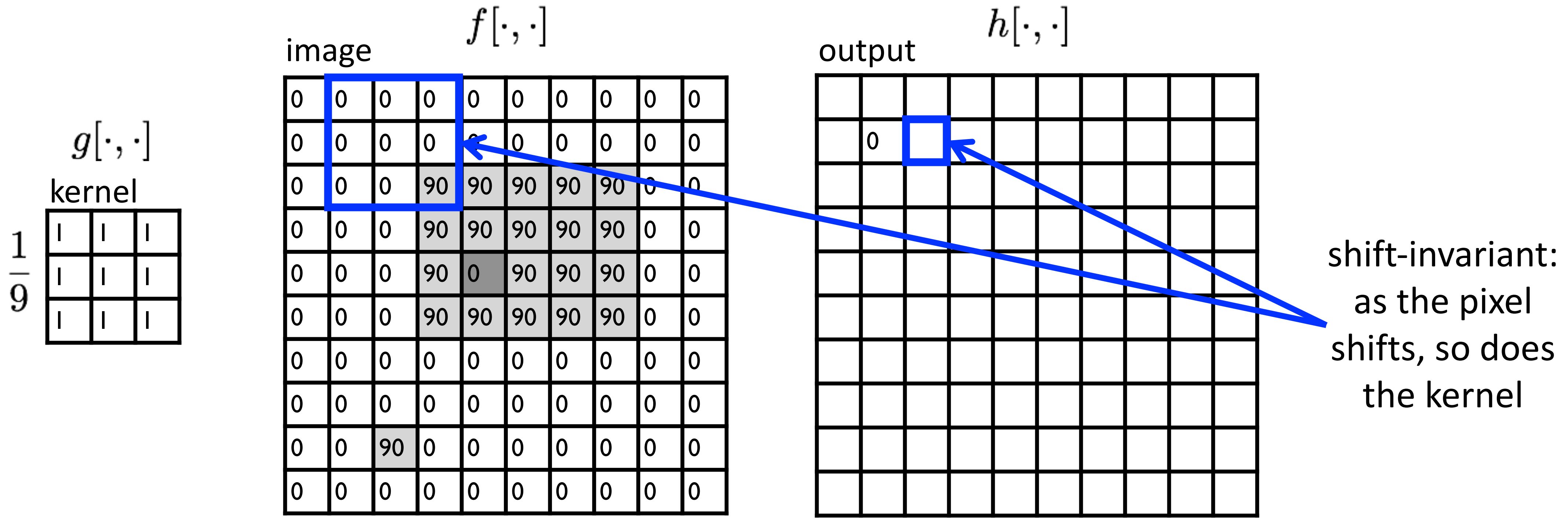
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

output $h[\cdot,\cdot]$

	0								

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}}\, \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

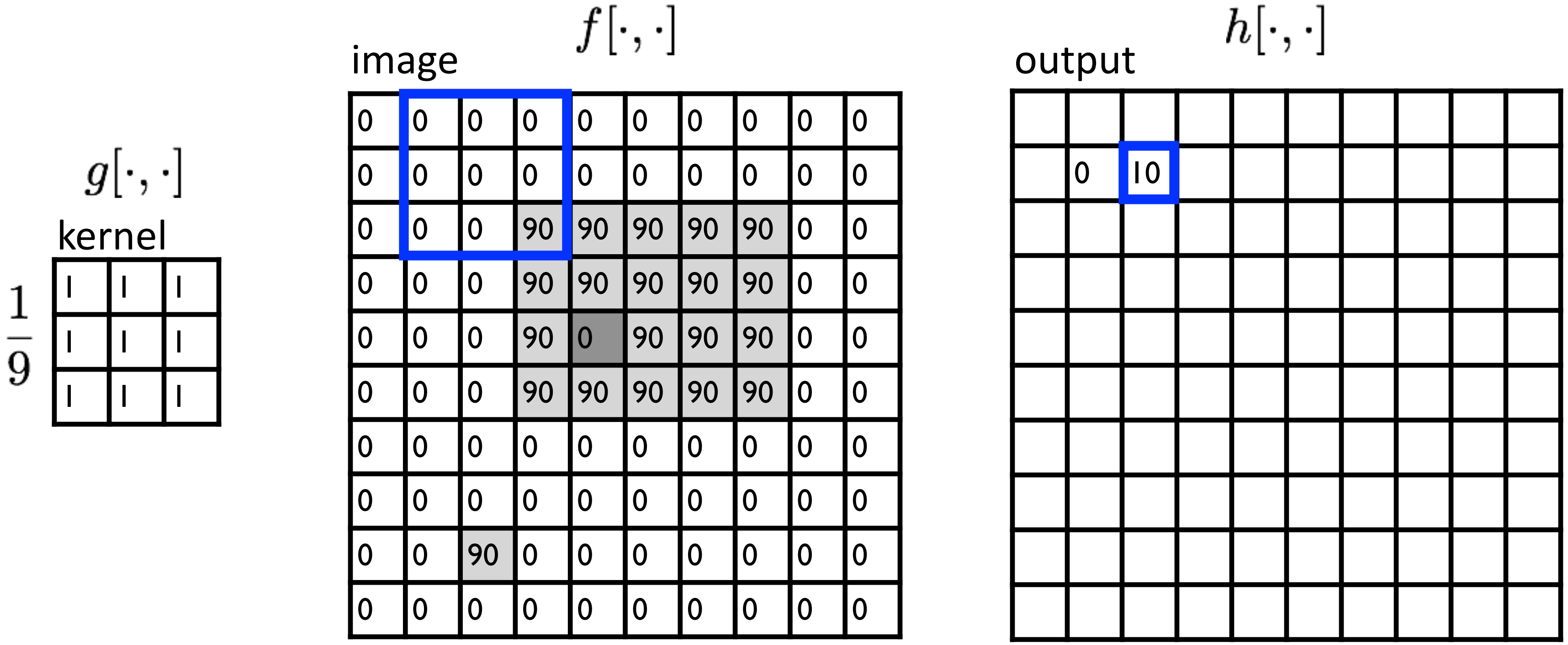
	0								

shift-invariant: as the pixel shifts, so does the kernel

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

output filter image (signal)

Let’s run the box filter



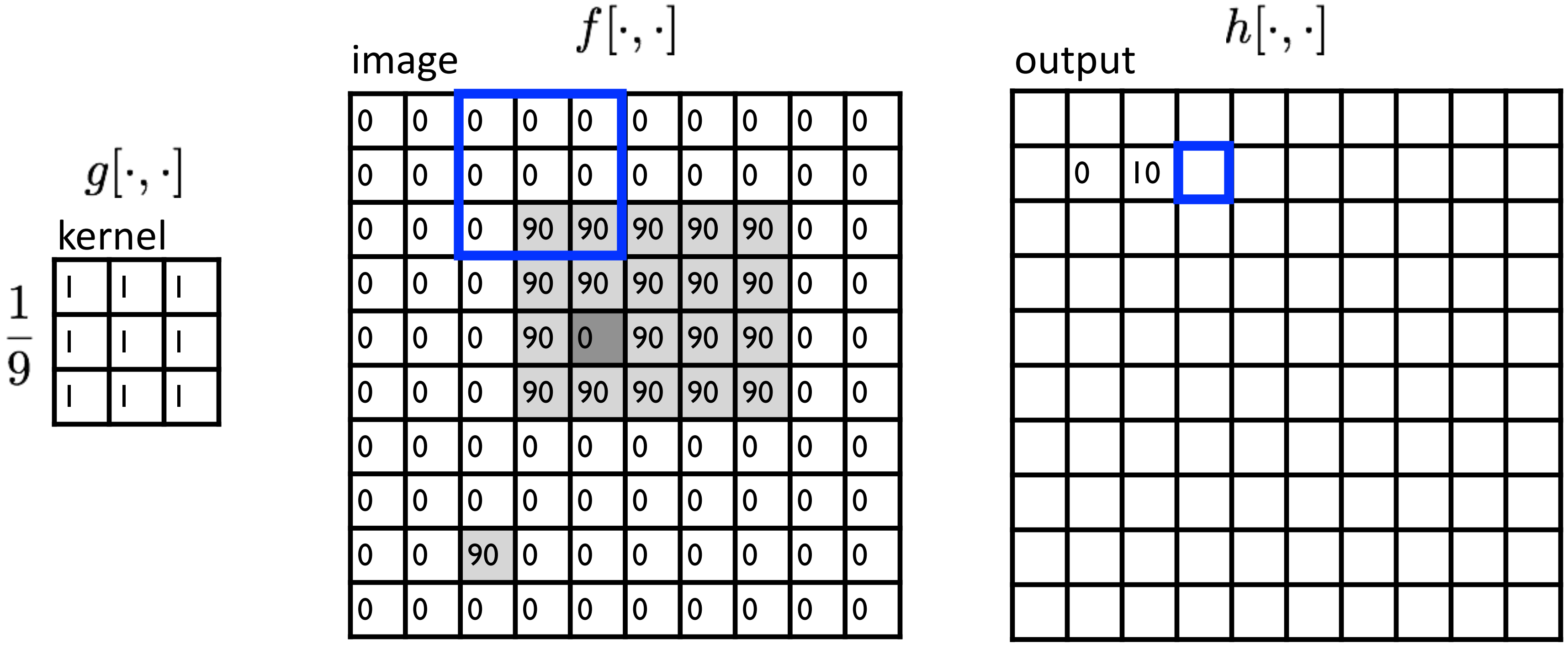
1	1	1
1	1	1
1	1	1

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

	0	10							

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

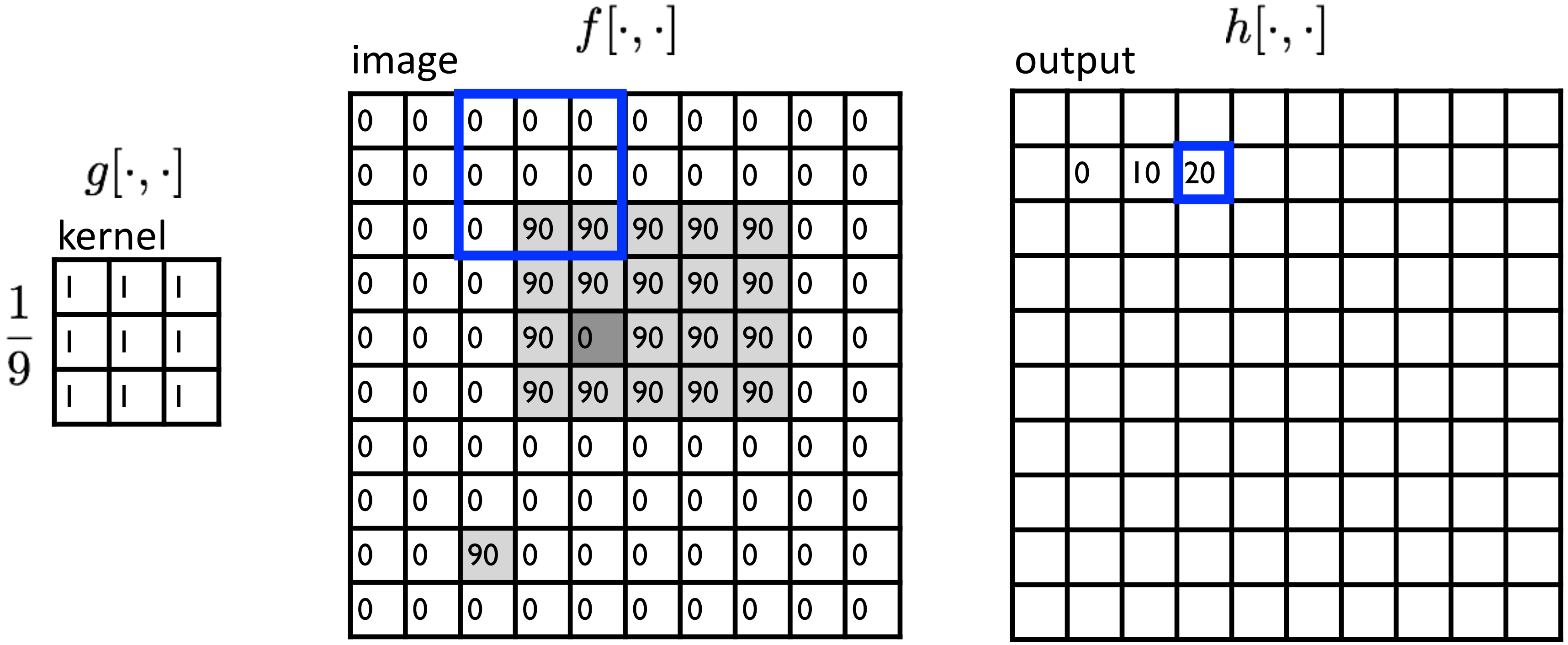
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

output $h[\cdot,\cdot]$

	0	10							

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20						

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20						

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

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

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20	30					

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

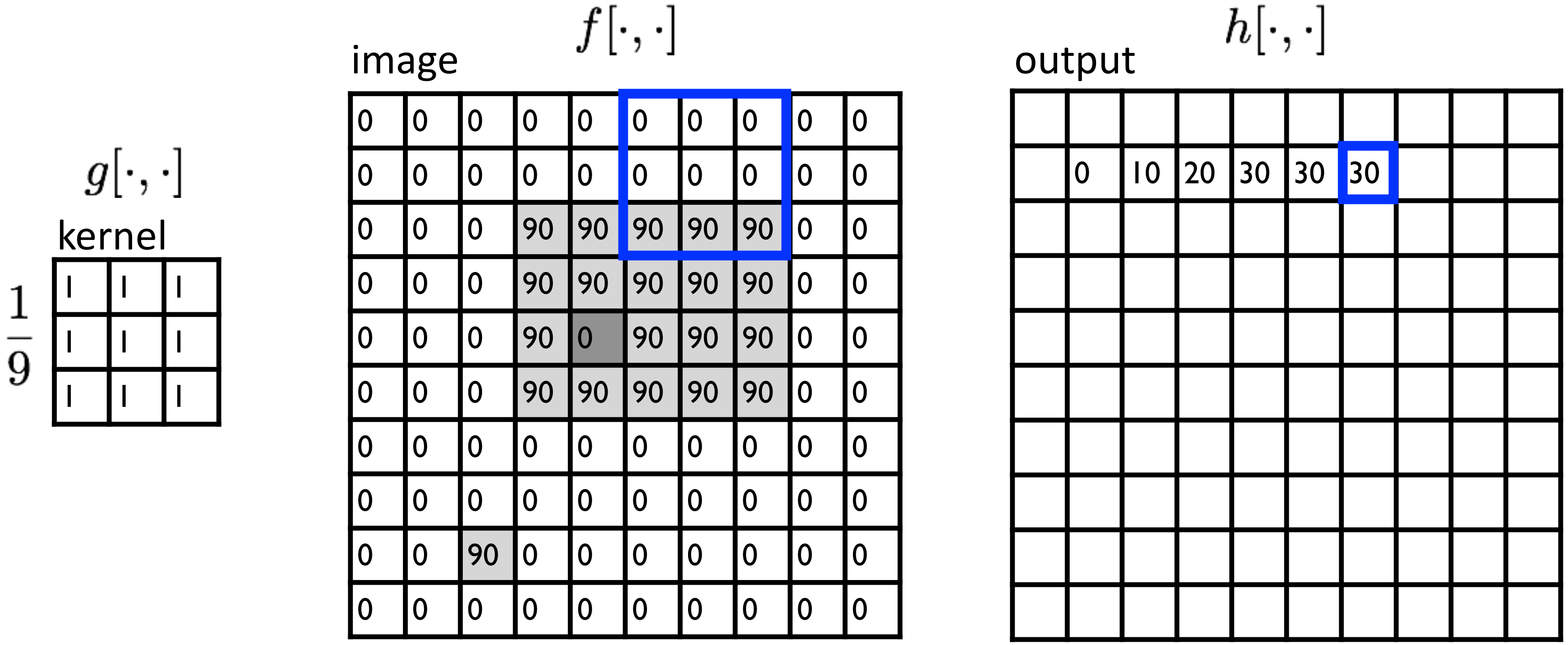
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

output $h[\cdot,\cdot]$

	0	10	20	30	30				

$$\underbrace{h[m,n]}_{\text{output}}=\sum_{k,l}\underbrace{g[k,l]}_{\text{filter}}\underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30			

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

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

image $f[\cdot,\cdot]$

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

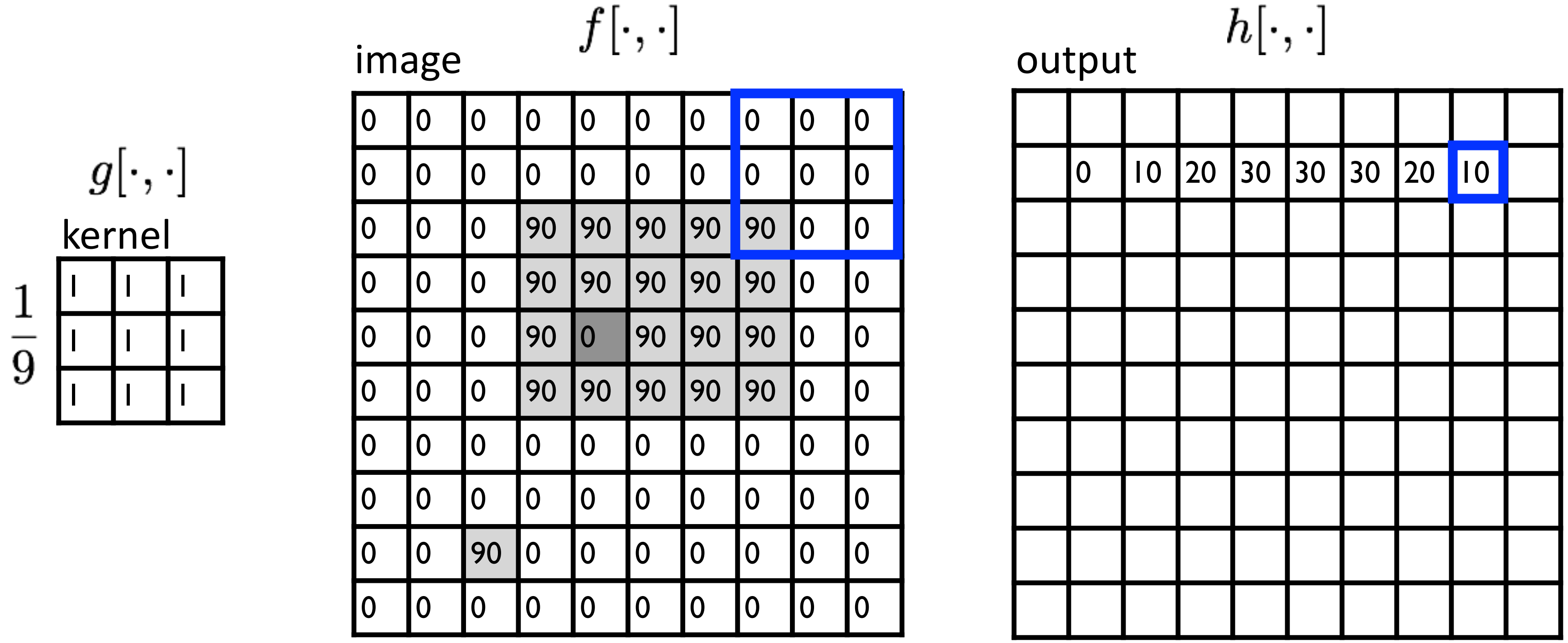
output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20		

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

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

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	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

image $f[\cdot,\cdot]$

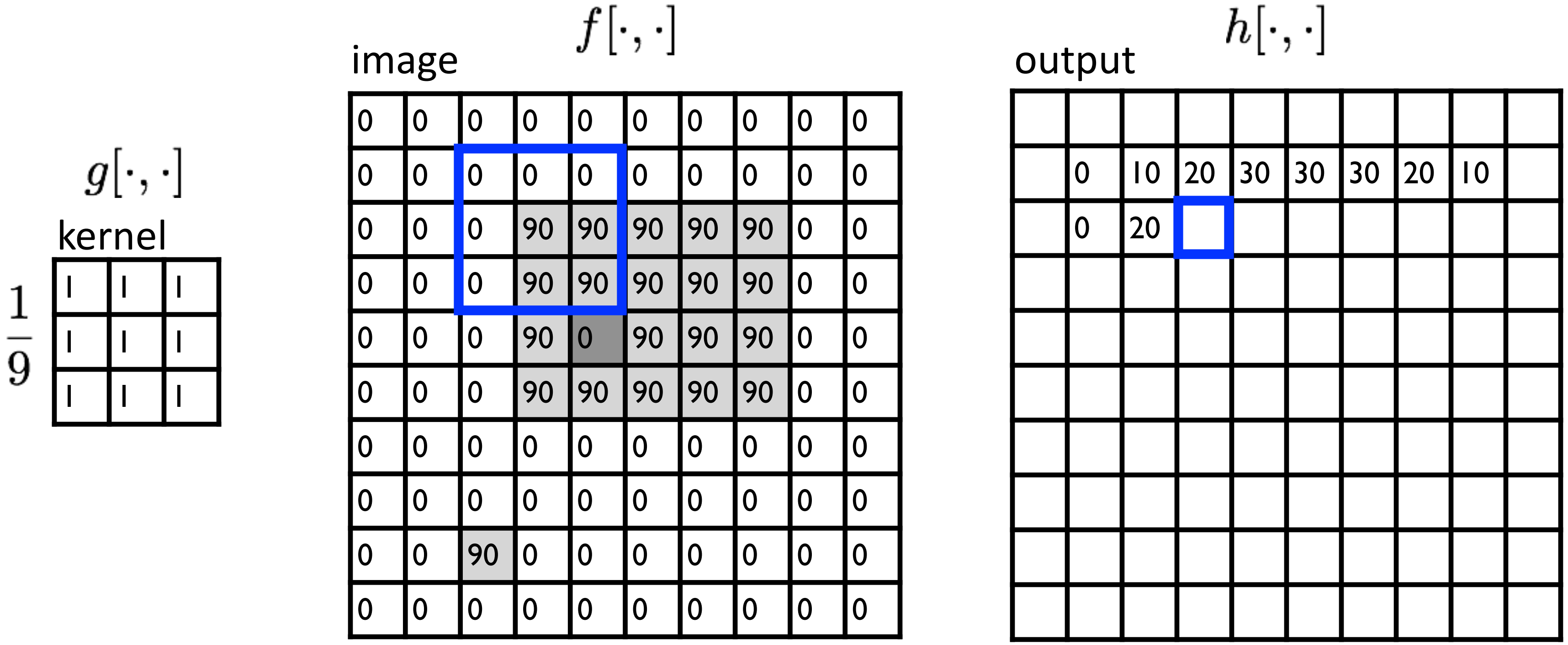
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0								

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20							

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

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

image $f[\cdot,\cdot]$

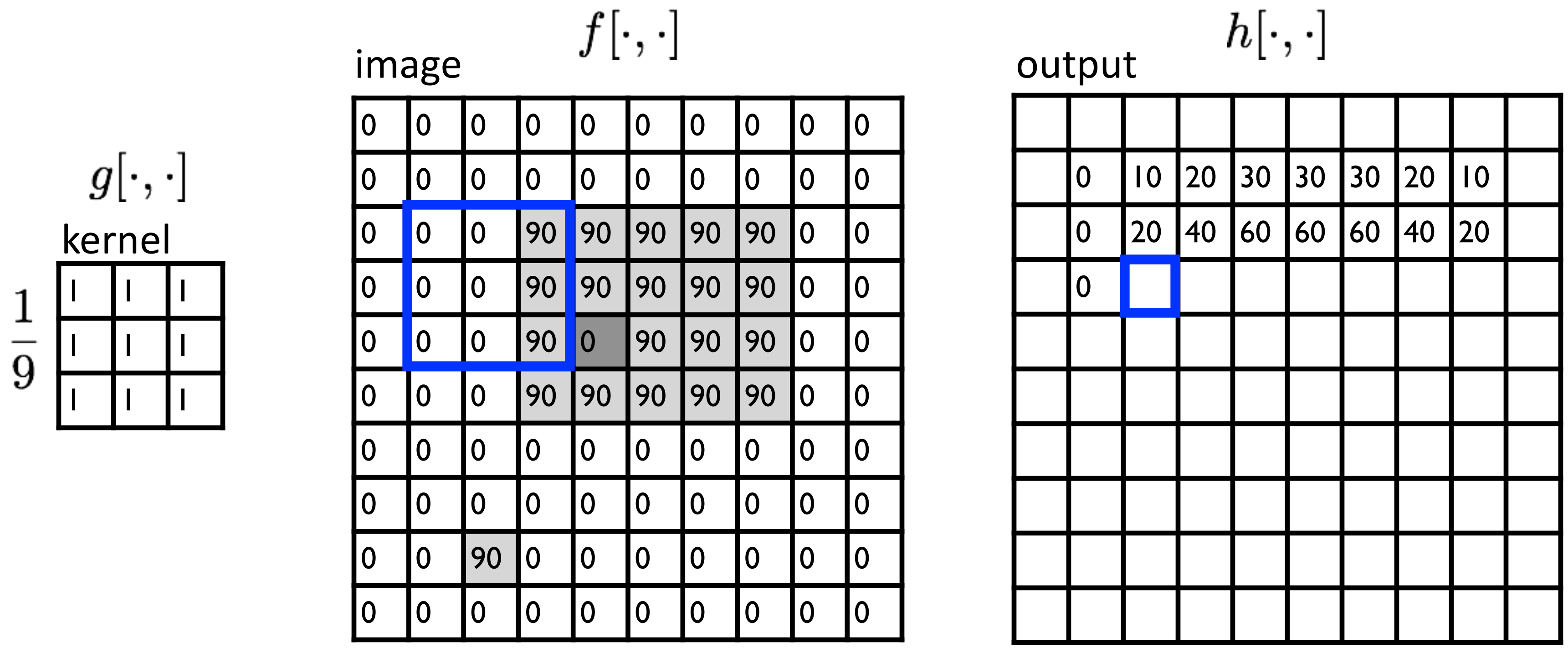
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40						

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter



$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

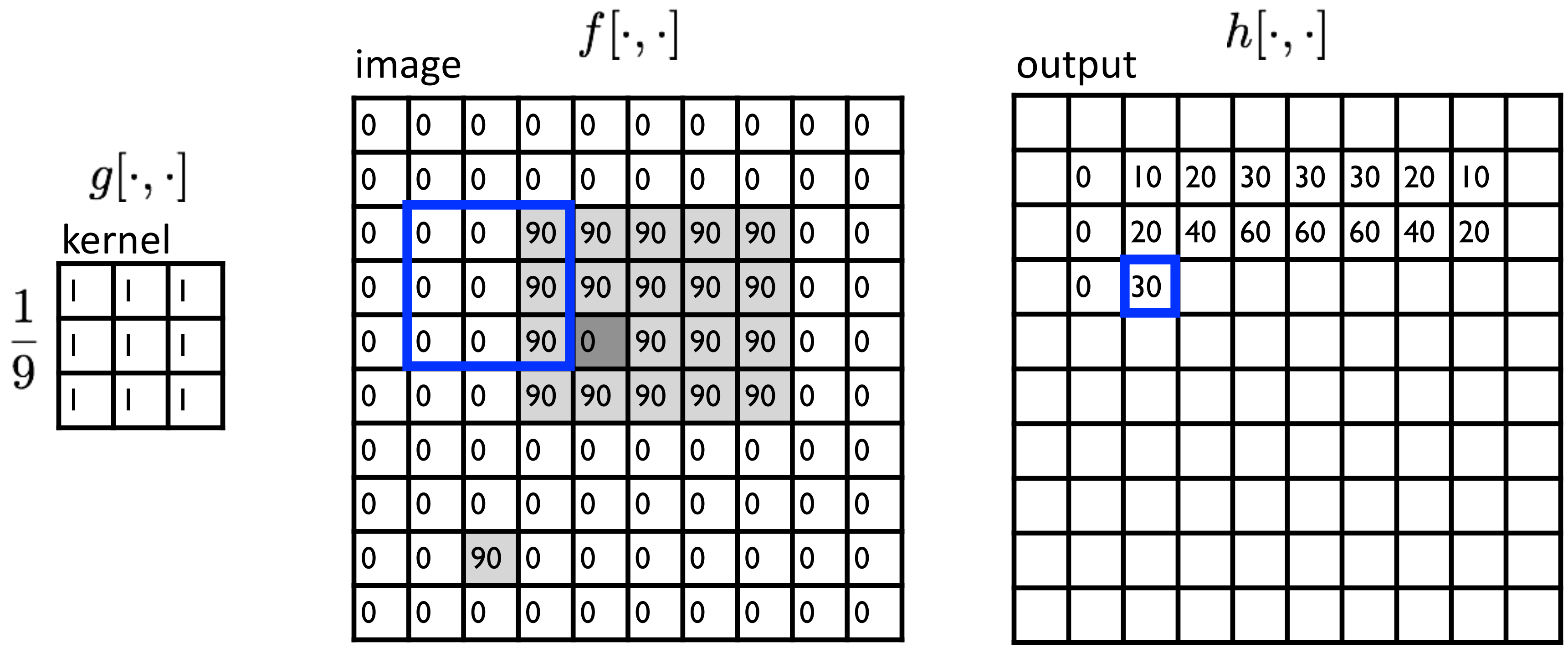
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0								

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

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

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30							

$$\underbrace{h[m,n]}_{\text{output}} = \sum_{k,l} \underbrace{g[k,l]}_{\text{filter}} \underbrace{f[m+k,n+l]}_{\text{image (signal)}}$$

Let’s run the box filter

$g[\cdot,\cdot]$

kernel

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

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	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

image $f[\cdot,\cdot]$

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

output $h[\cdot,\cdot]$

	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

image $f[\cdot,\cdot]$

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

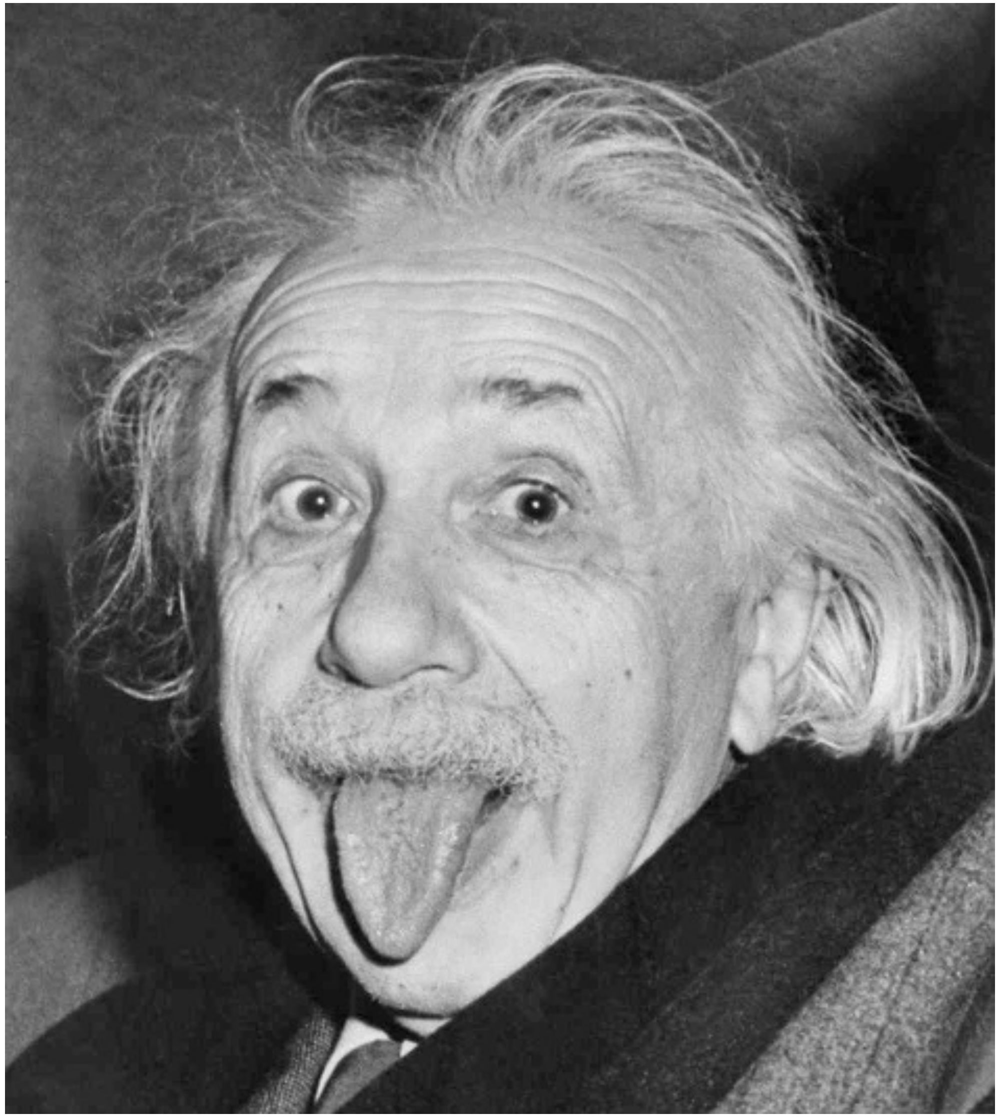
output $h[\cdot,\cdot]$

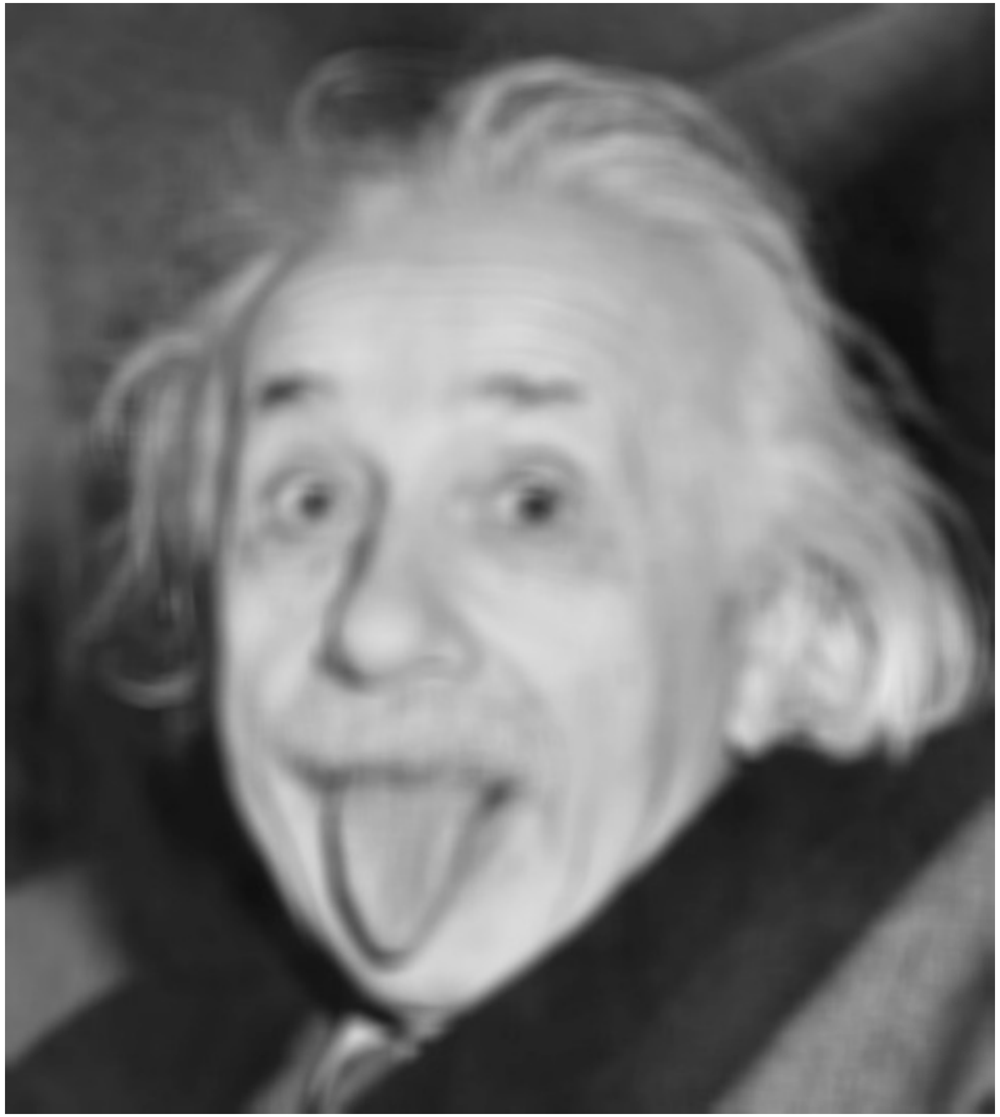
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	10	10	10	0	0	0	0	

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

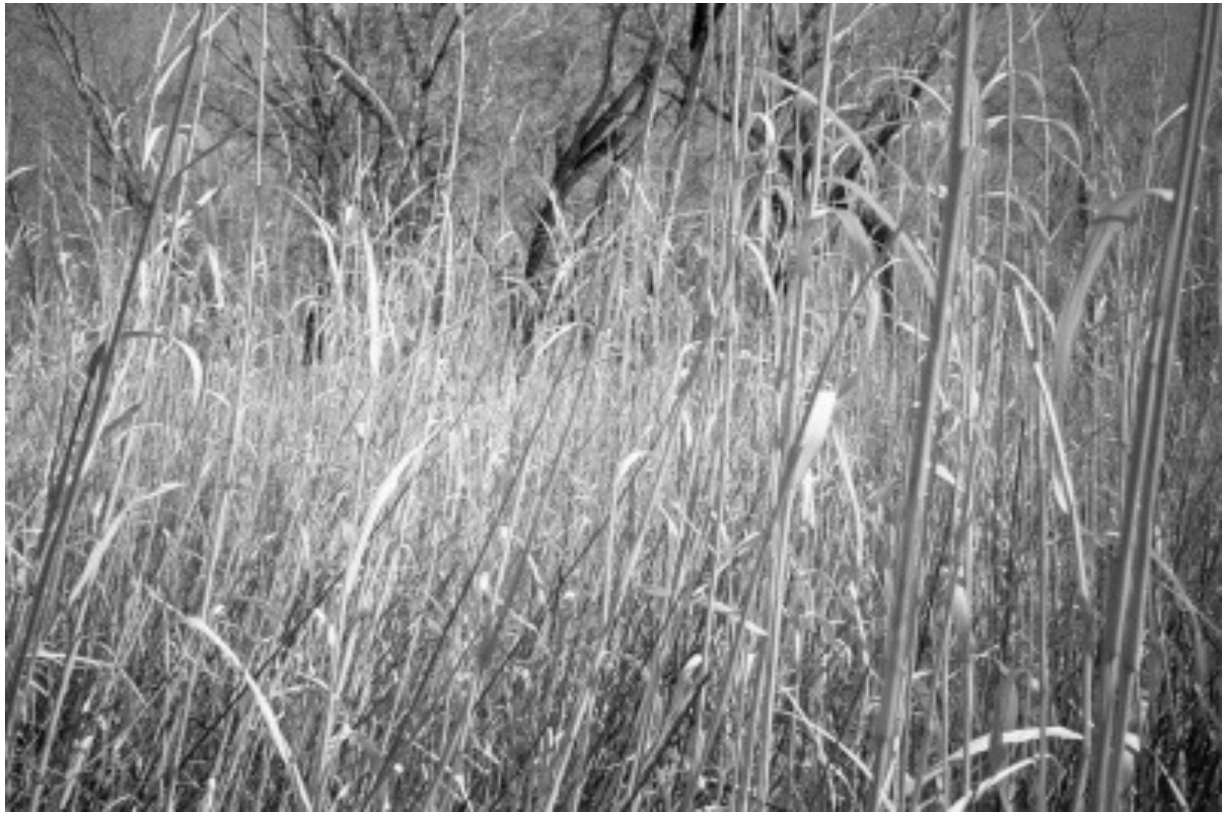
output filter image (signal)

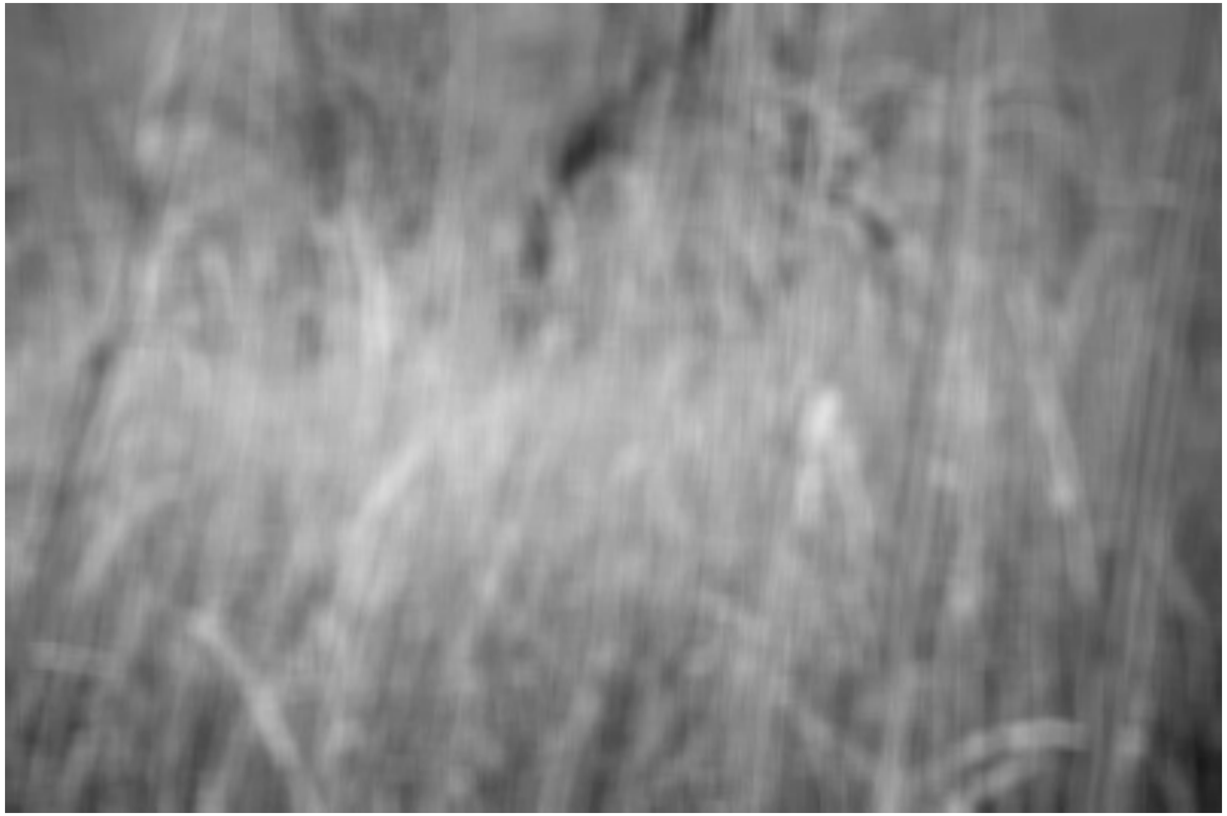
Some more realistic examples



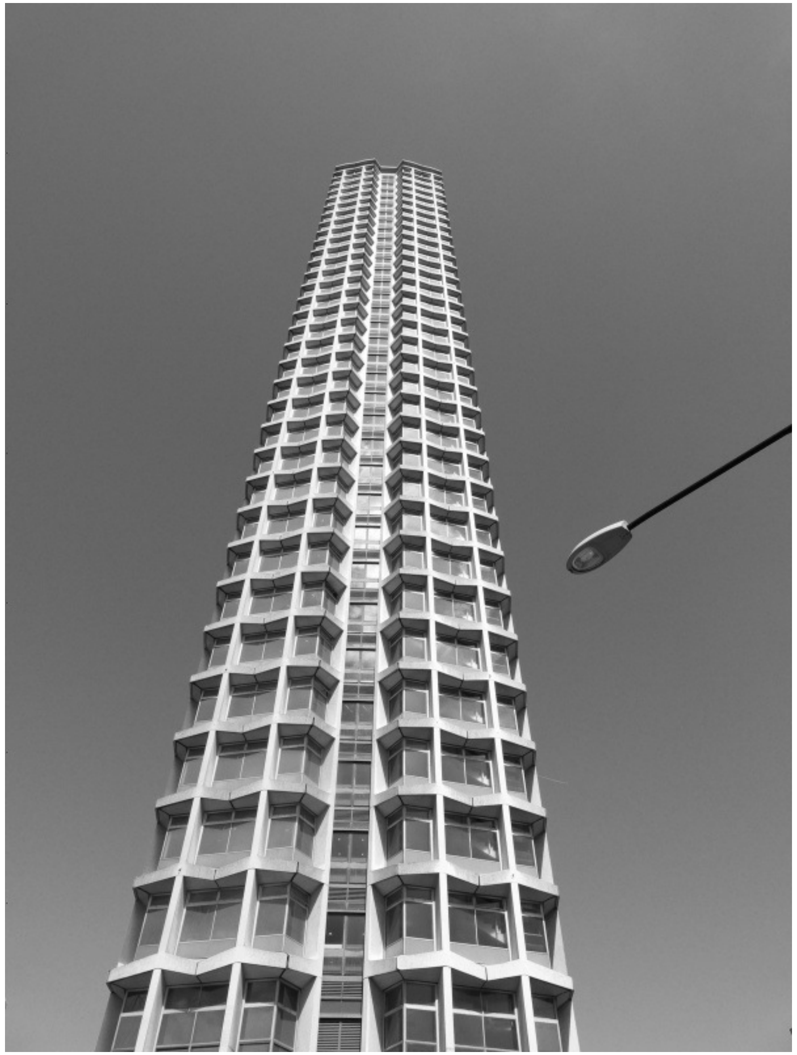


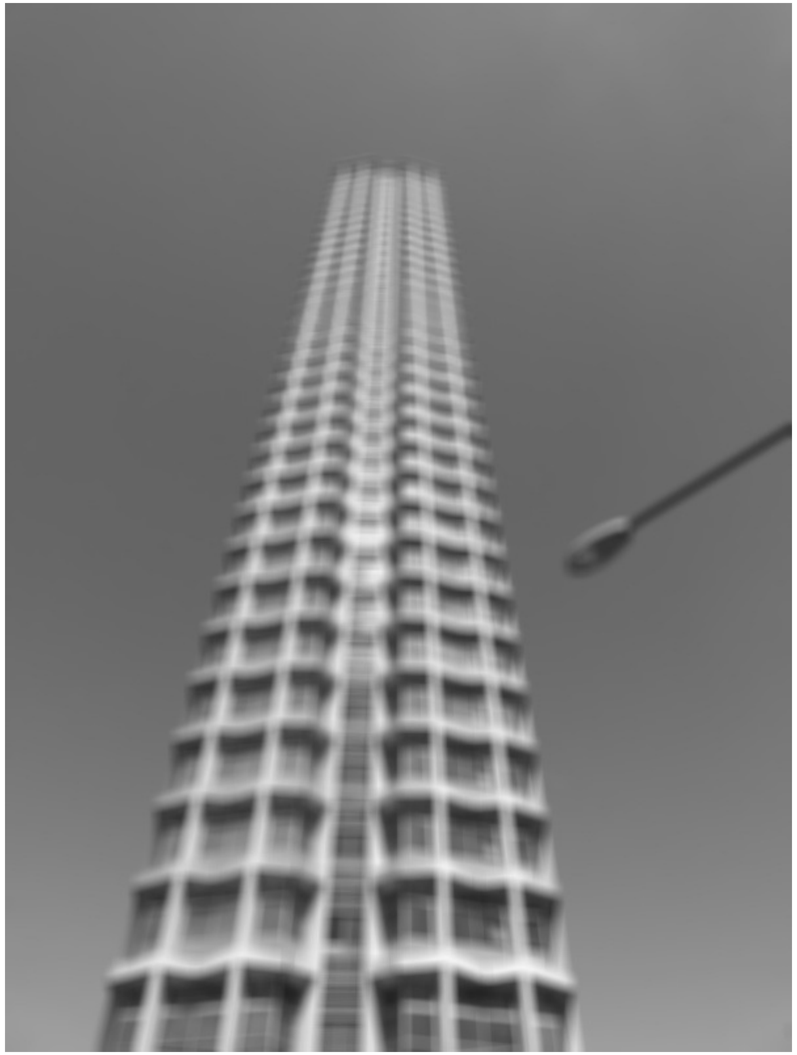
Some more realistic examples





Some more realistic examples





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

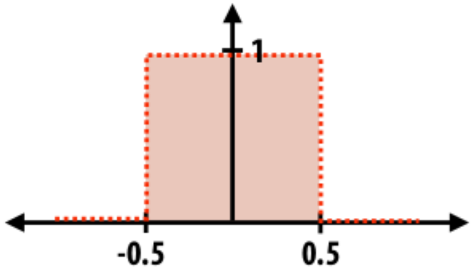
filter

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

$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \underset{\text{column}}{\begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}} * \underset{\text{row}}{\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline \end{array}}$$

What is the rank of this filter matrix?

Separable filters

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

example:
box filter

$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \underset{\text{column}}{\begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}} * \underset{\text{row}}{\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline \end{array}}$$

Why is this important?

Separable filters

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

example:
box filter

$$\begin{array}{|c|c|c|}\hline 1&1&1\\ \hline 1&1&1\\ \hline 1&1&1\\ \hline \end{array} \quad = \quad \underset{\text{column}}{\begin{array}{|c|}\hline 1\\ \hline 1\\ \hline 1\\ \hline \end{array}} \quad * \quad \underset{\text{row}}{\begin{array}{|c|c|c|}\hline 1&1&1\\ \hline \end{array}}$$

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

$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \underset{\text{column}}{\begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}} * \underset{\text{row}}{\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline \end{array}}$$

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

$$\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = \underset{\text{column}}{\begin{bmatrix}1\\1\\1\end{bmatrix}} * \underset{\text{row}}{\begin{bmatrix}1&1&1\end{bmatrix}}$$

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? $\longrightarrow$ M^2 x 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

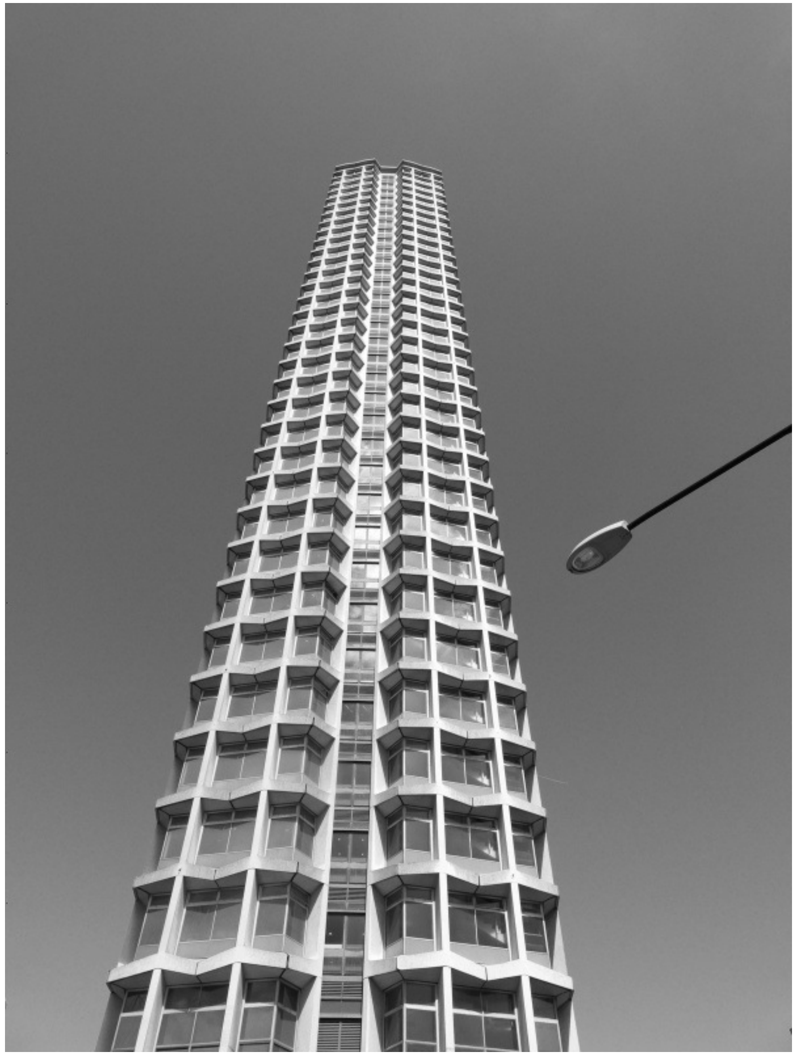
$$\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} = \underbrace{\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}}_{\text{column}} * \underbrace{\begin{bmatrix}1 & 1 & 1\end{bmatrix}}_{\text{row}}$$

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

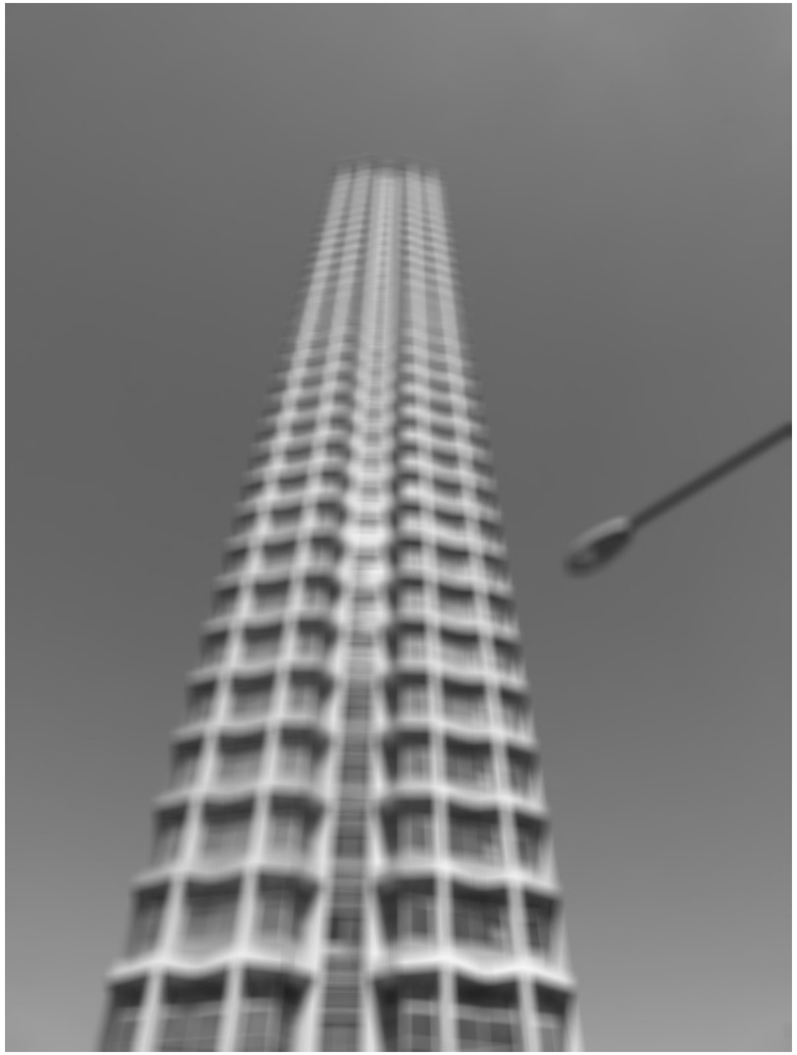
If the image has M x M pixels and the filter kernel has size N x N:

- What is the cost of convolution with a non-separable filter? ⟶ M^2 x N^2
- What is the cost of convolution with a separable filter? ⟶ 2 x N x M^2

A few more filters



original



3x3 box filter

do you see
any problems
in this image?

The Gaussian filter

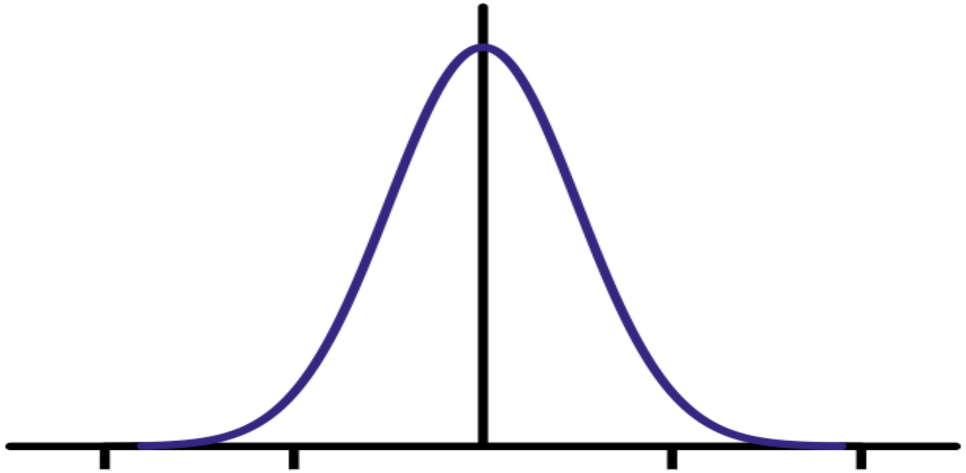
- named (like many other things) after Carl Friedrich Gauss

- 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:

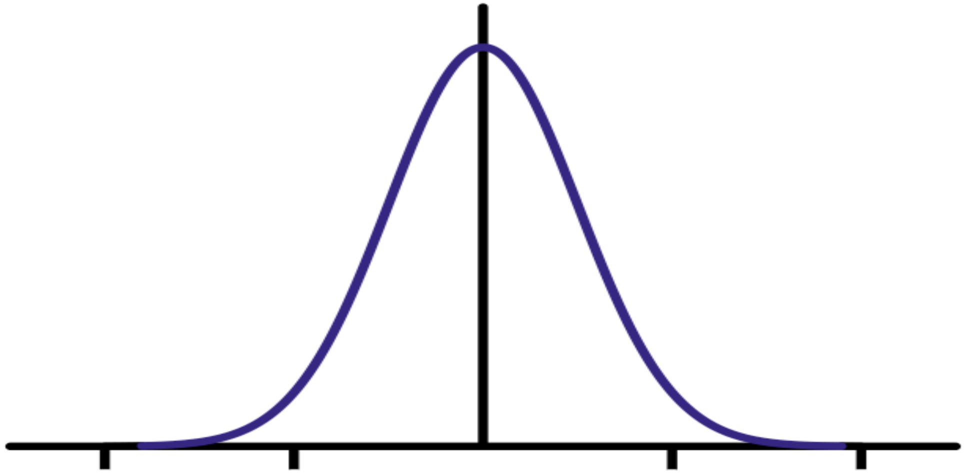
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- 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:

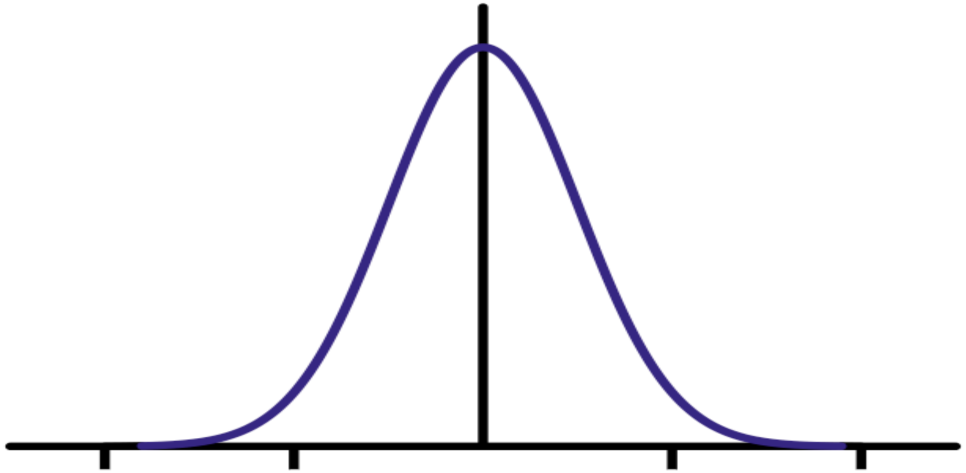
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- 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

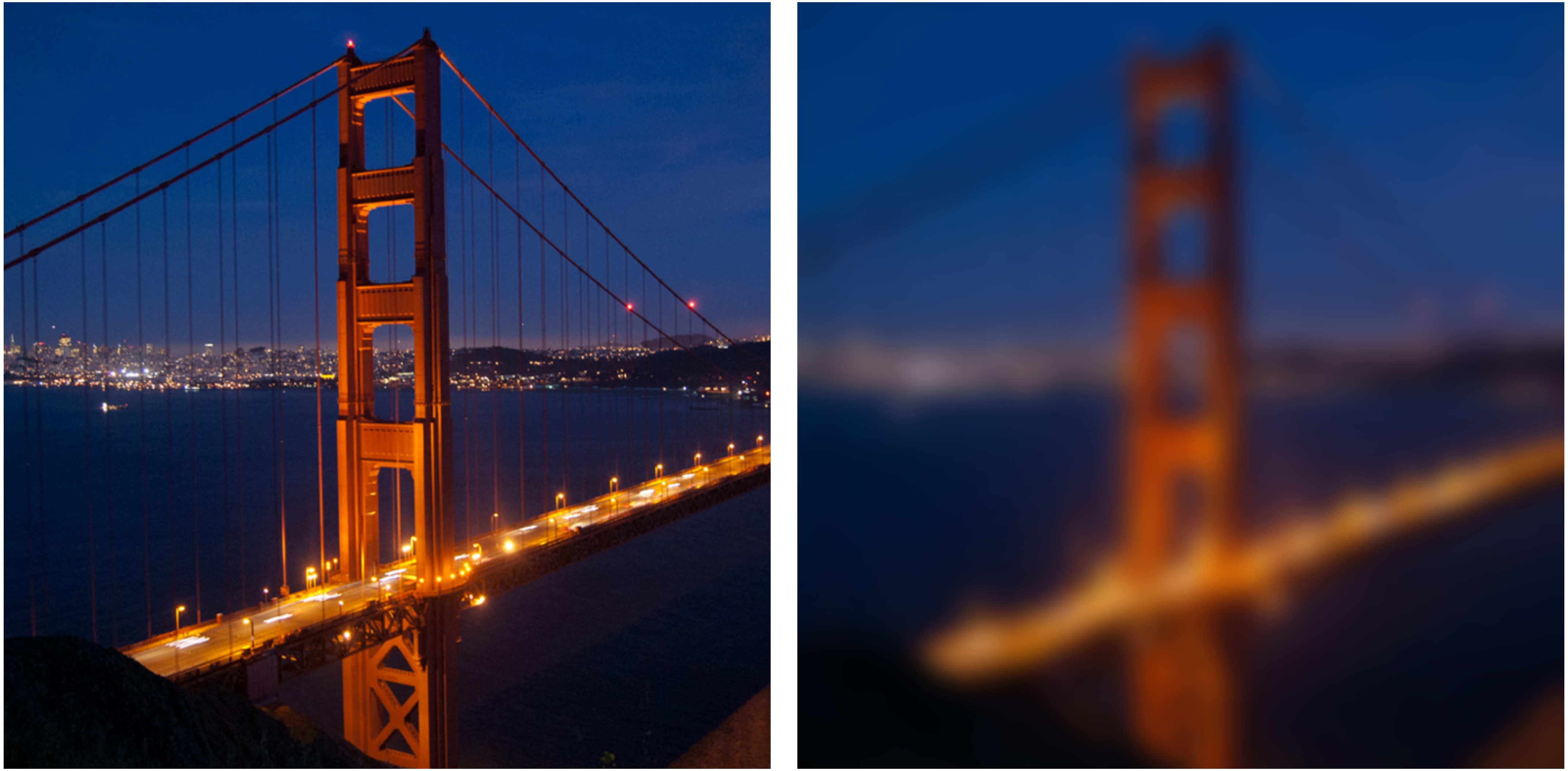


Is this a separable filter? Yes!

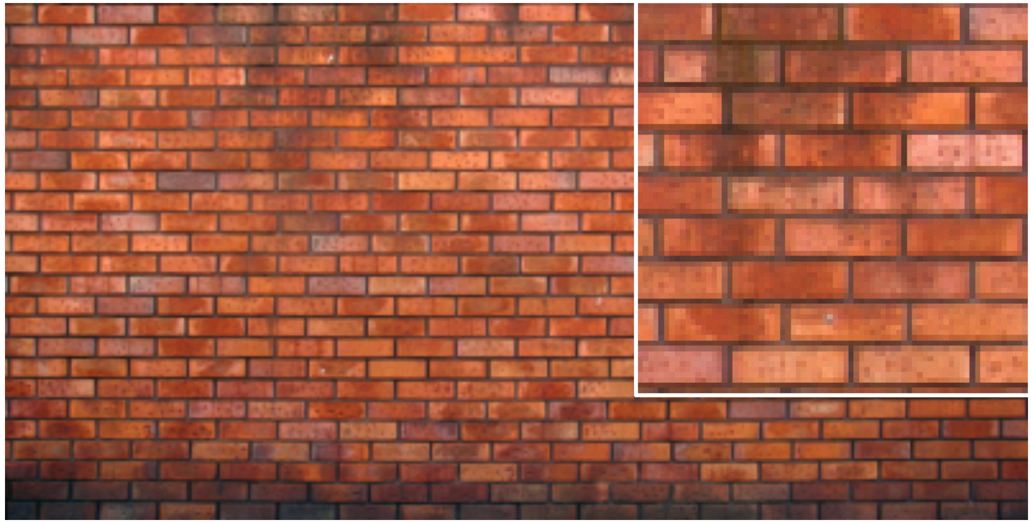
kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Gaussian filtering example

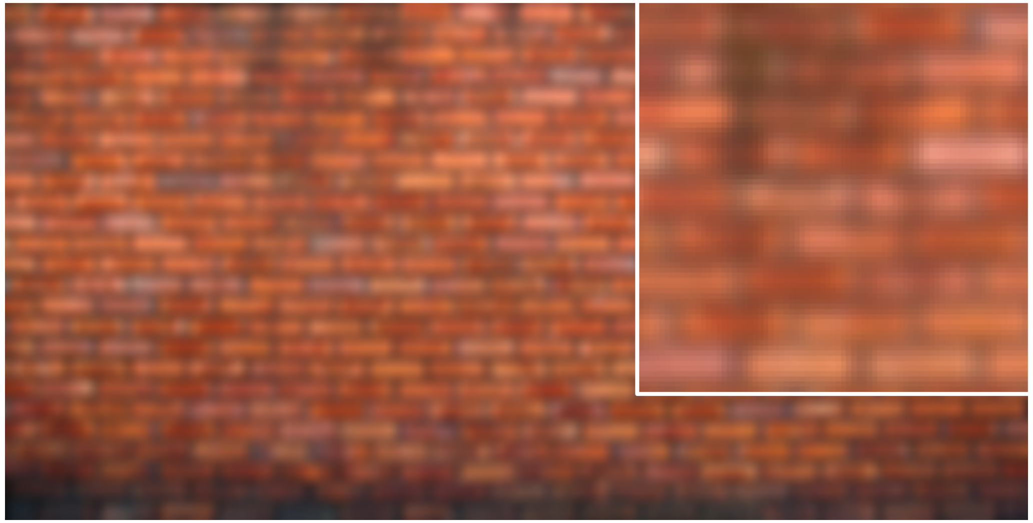


Gaussian vs box filtering

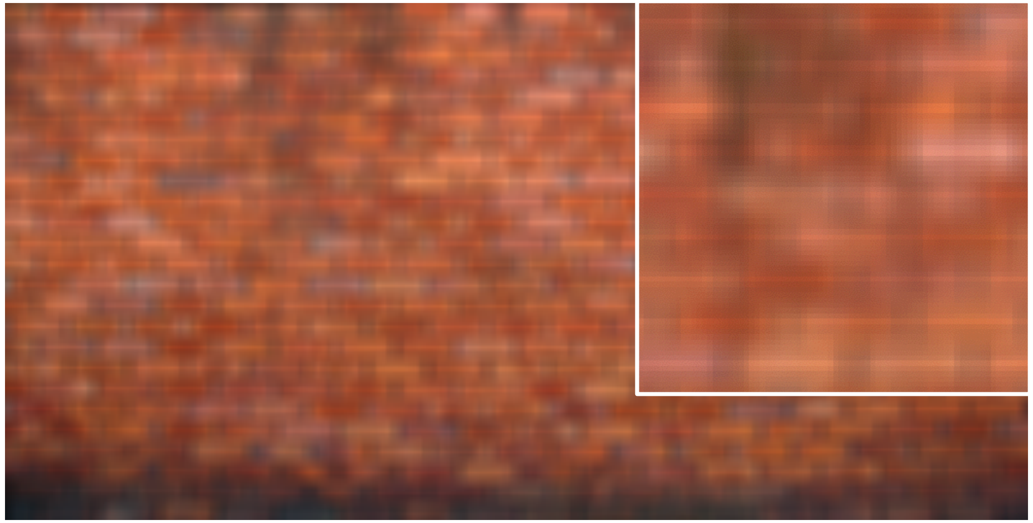


original

Which blur do you like better?

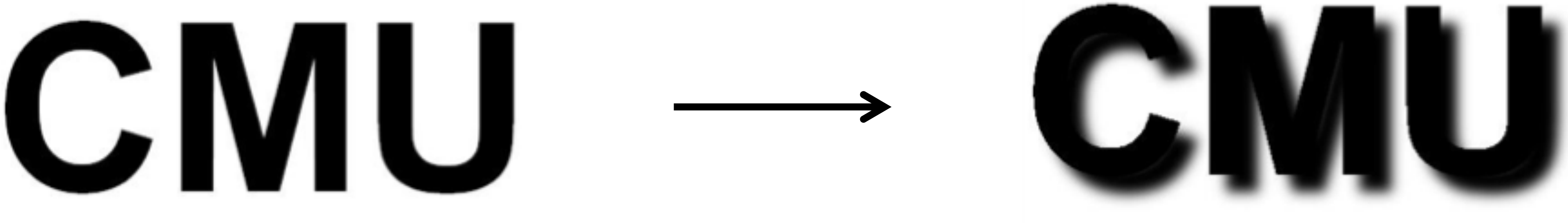


7x7 Gaussian

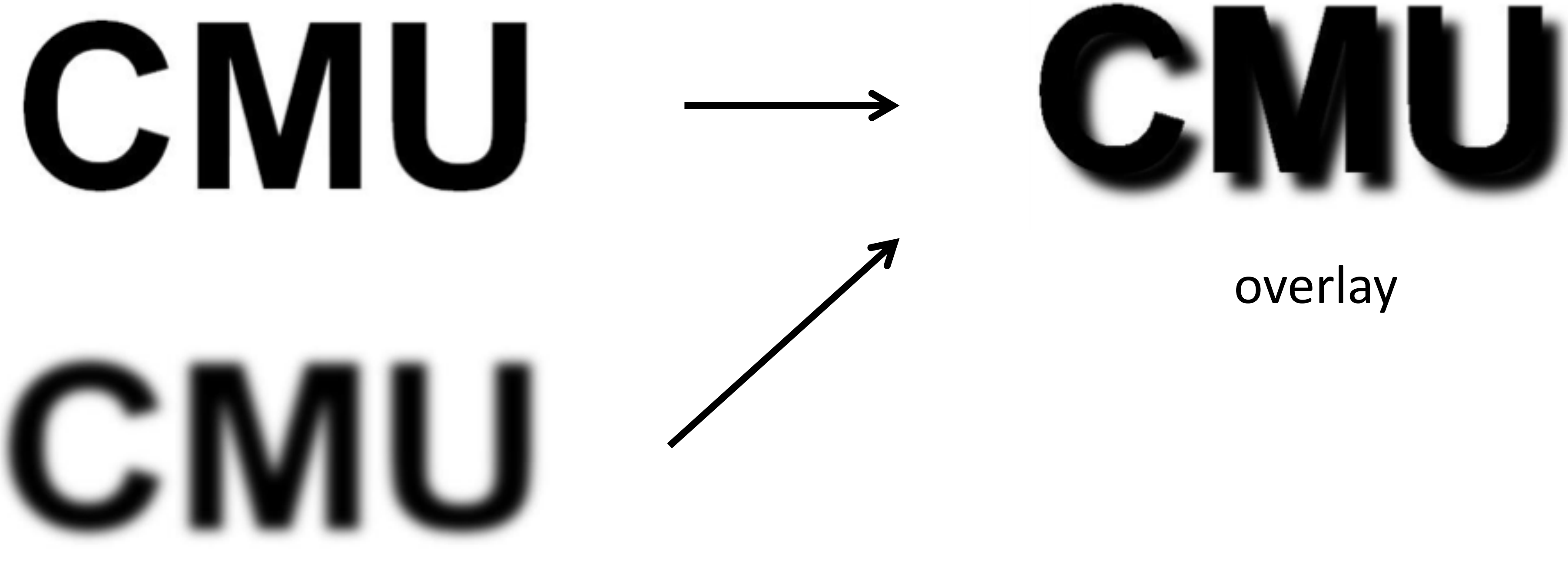


7x7 box

How would you create a soft shadow effect?



How would you create a soft shadow effect?



overlay

Gaussian blur

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

?

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

unchanged

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

unchanged

input

filter

0	0	0
0	0	1
0	0	0

output

?

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

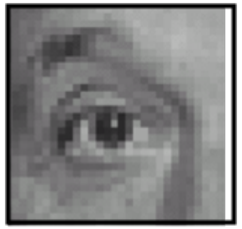
unchanged

input

filter

0	0	0
0	0	1
0	0	0

output



shift to left
by one

Other filters

input

filter

$$\begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&2&0\\\hline 0&0&0\\\hline\end{array} - \frac{1}{9}\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}$$

output

?

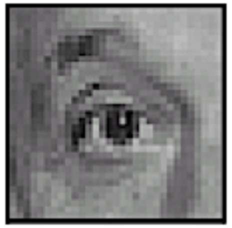
Other filters

input

filter

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9} \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

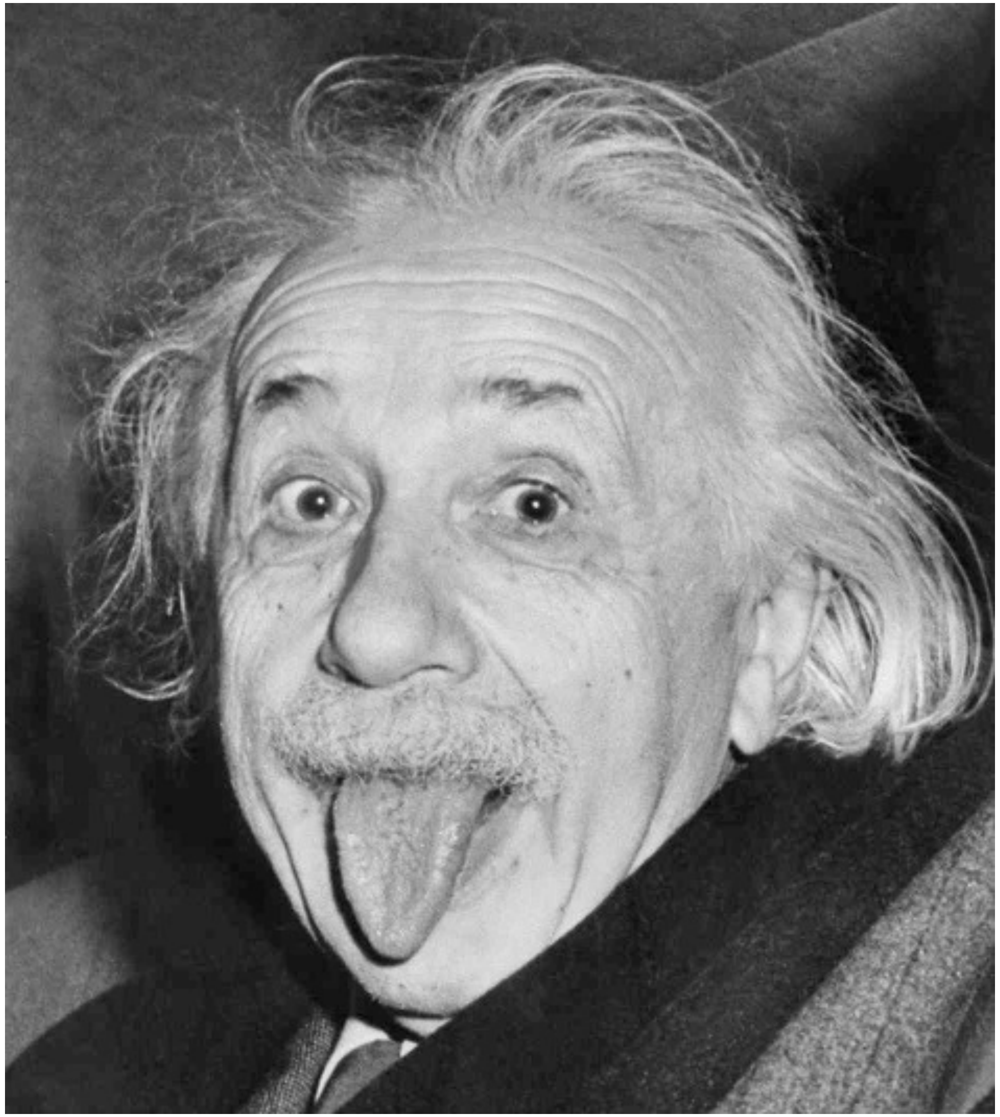
output

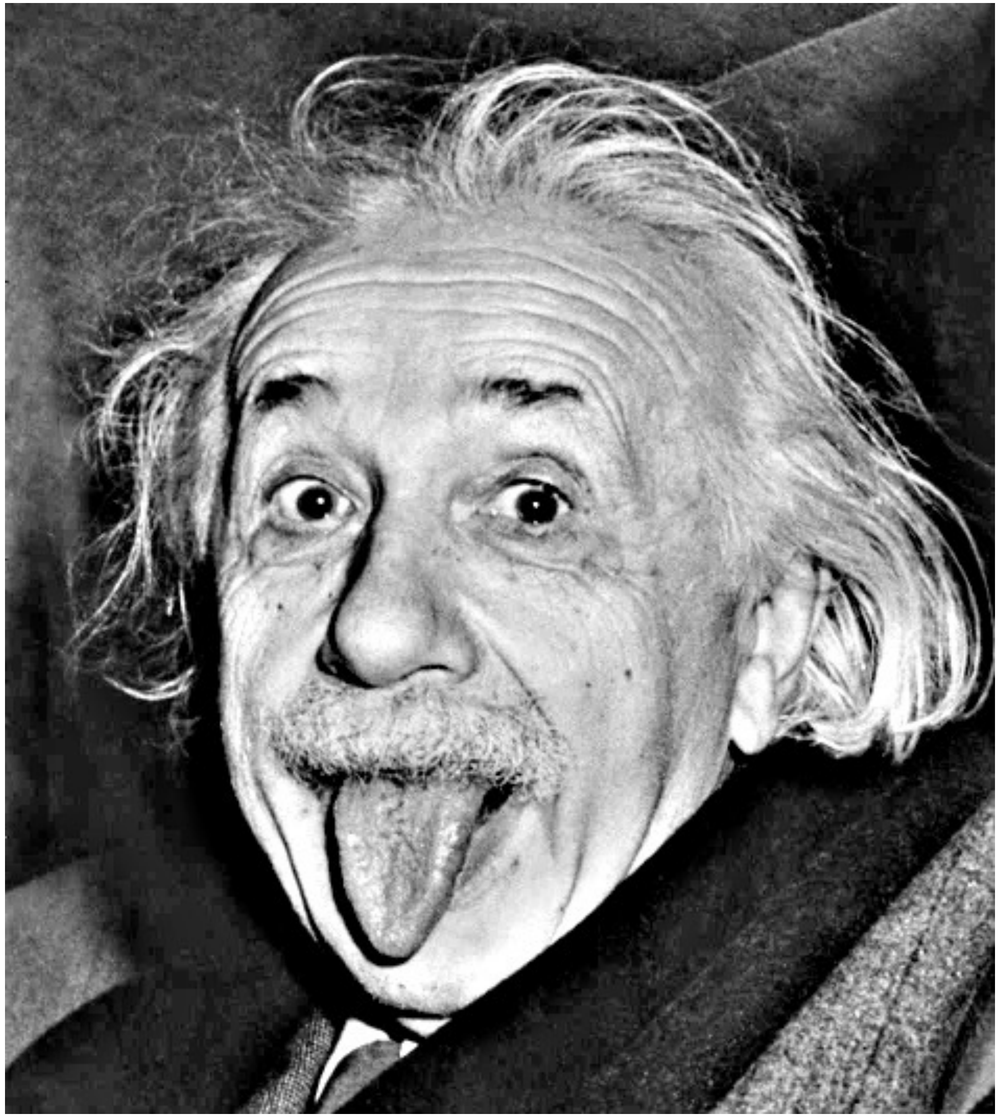


sharpening

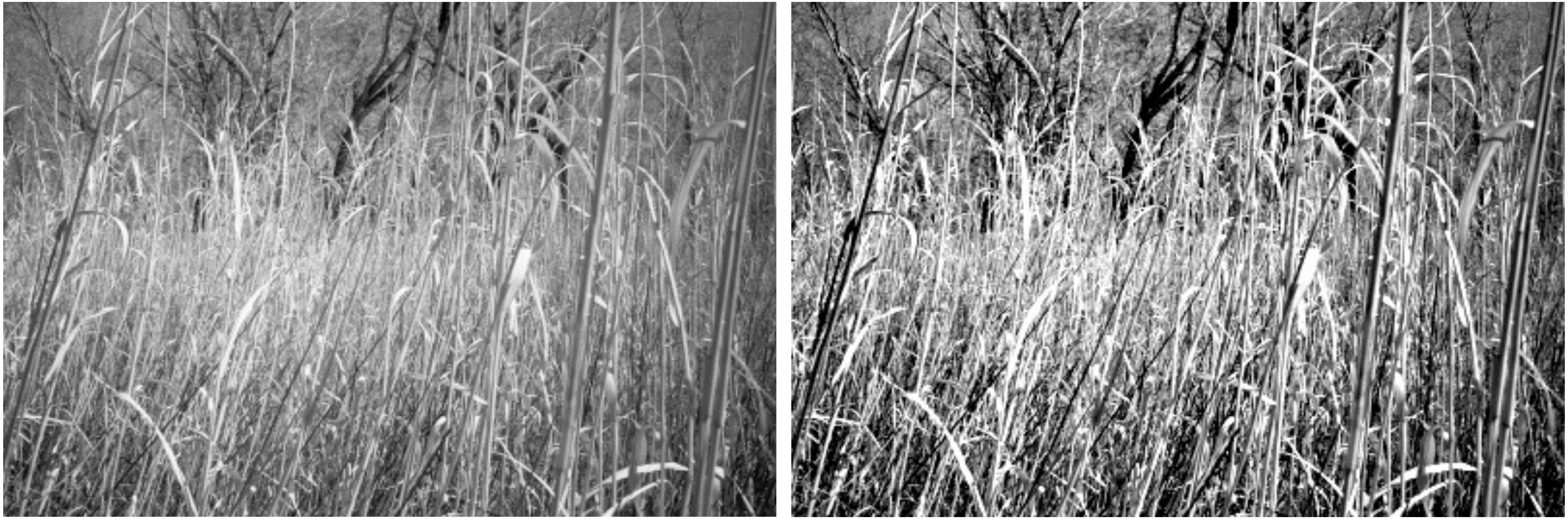
- do nothing for flat areas
- stress intensity peaks

Sharpening examples

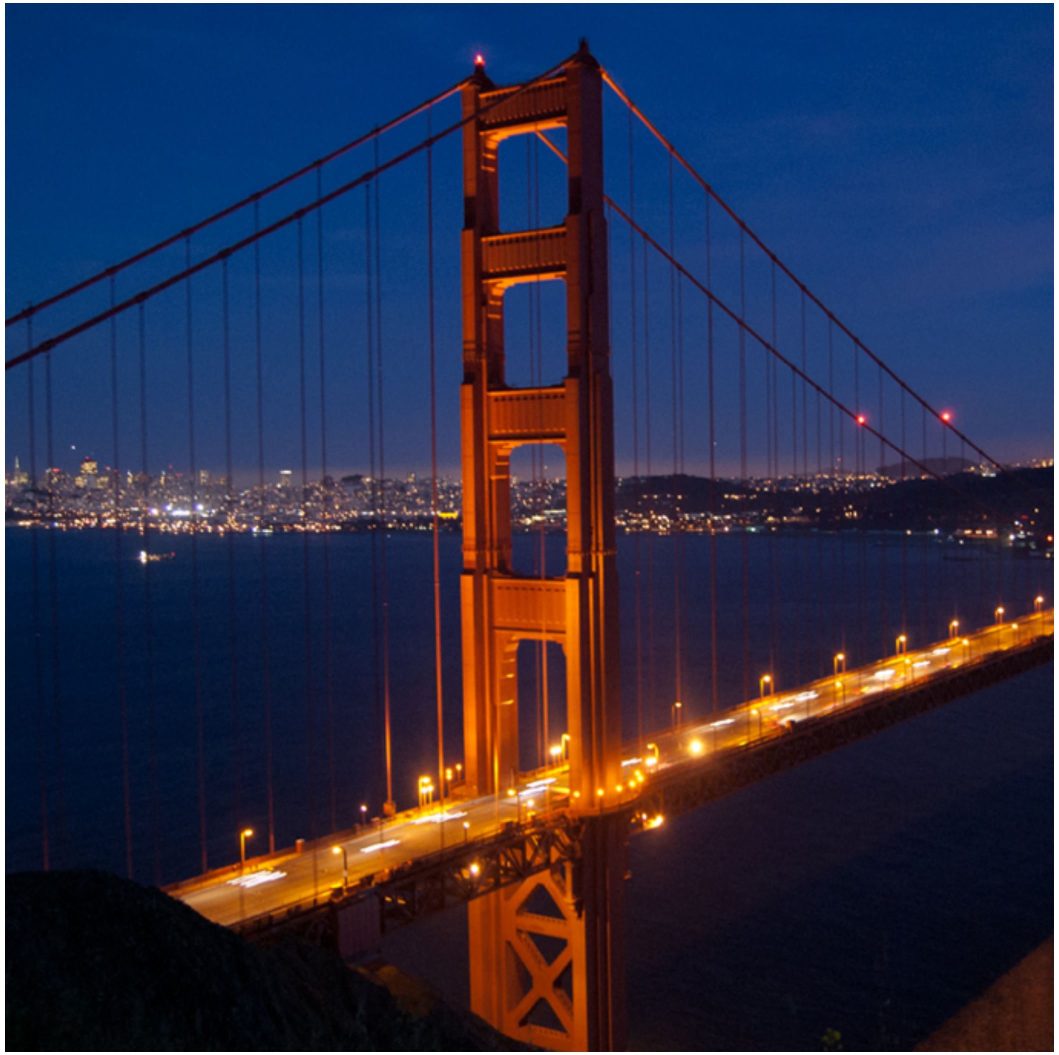


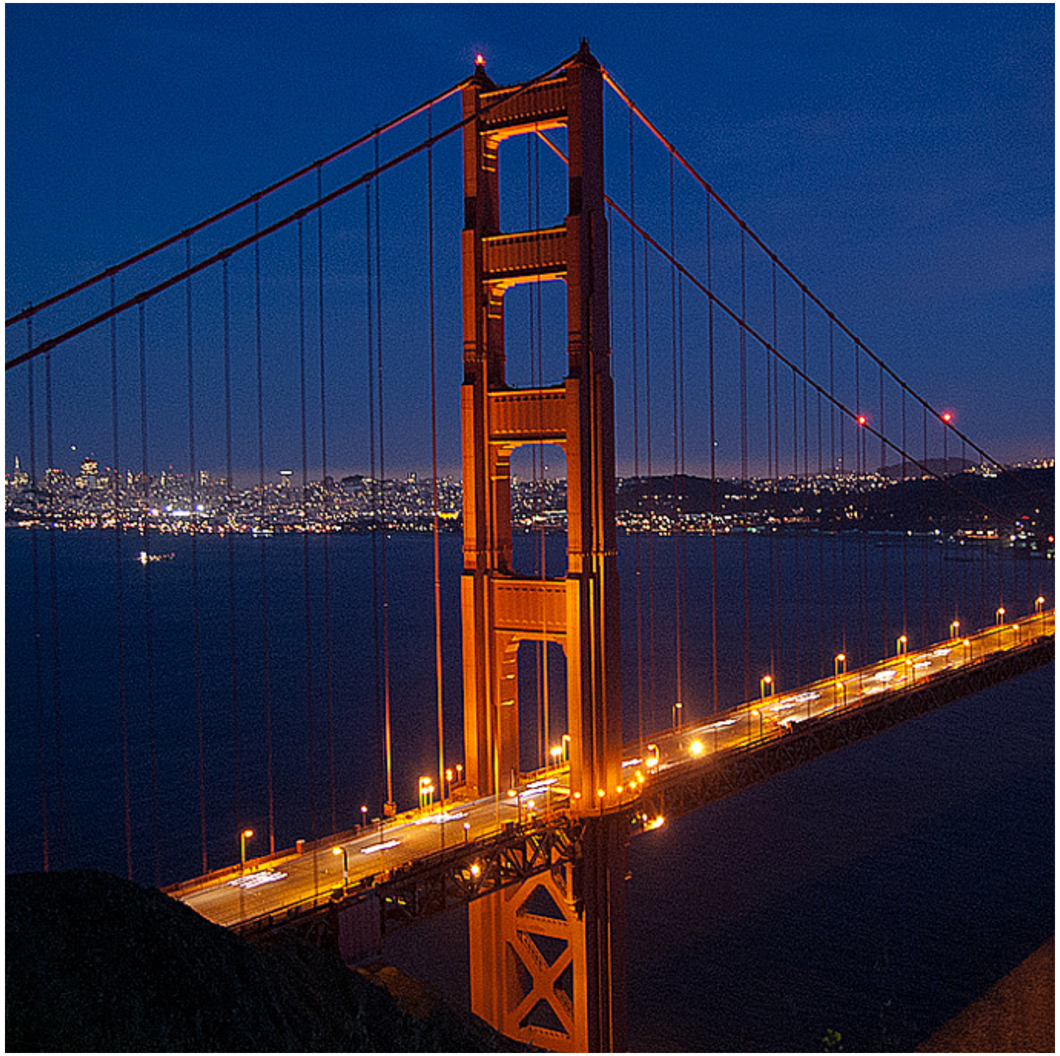


Sharpening examples

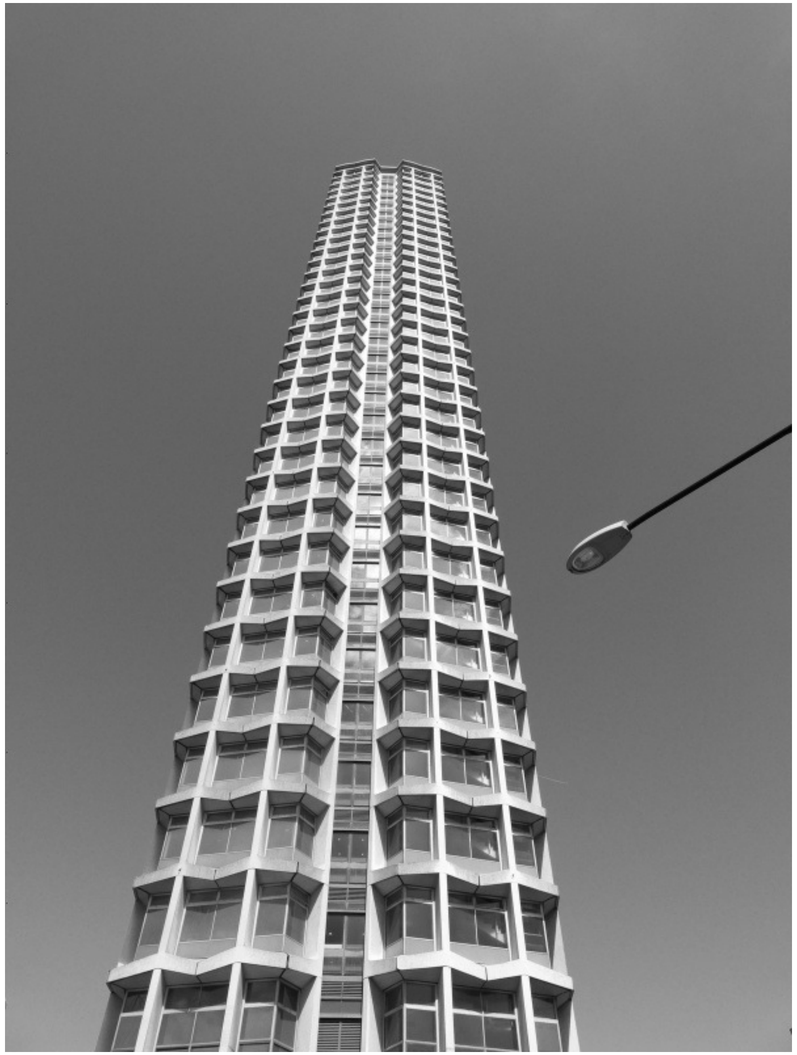


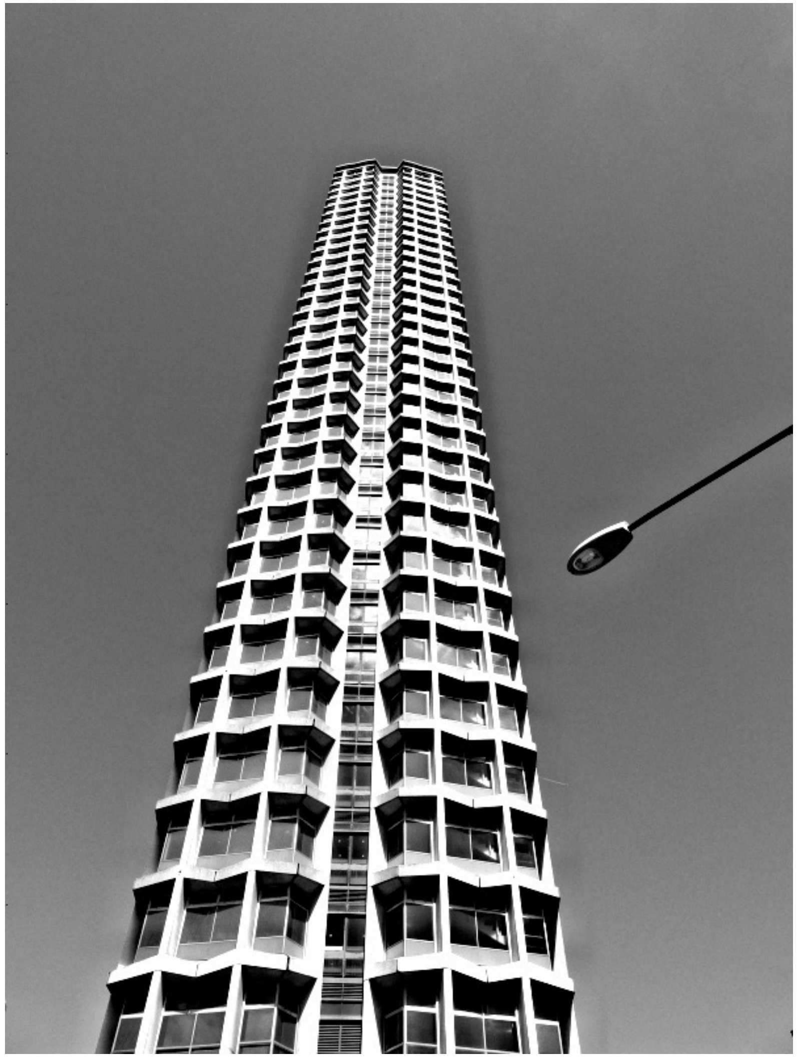
Sharpening examples





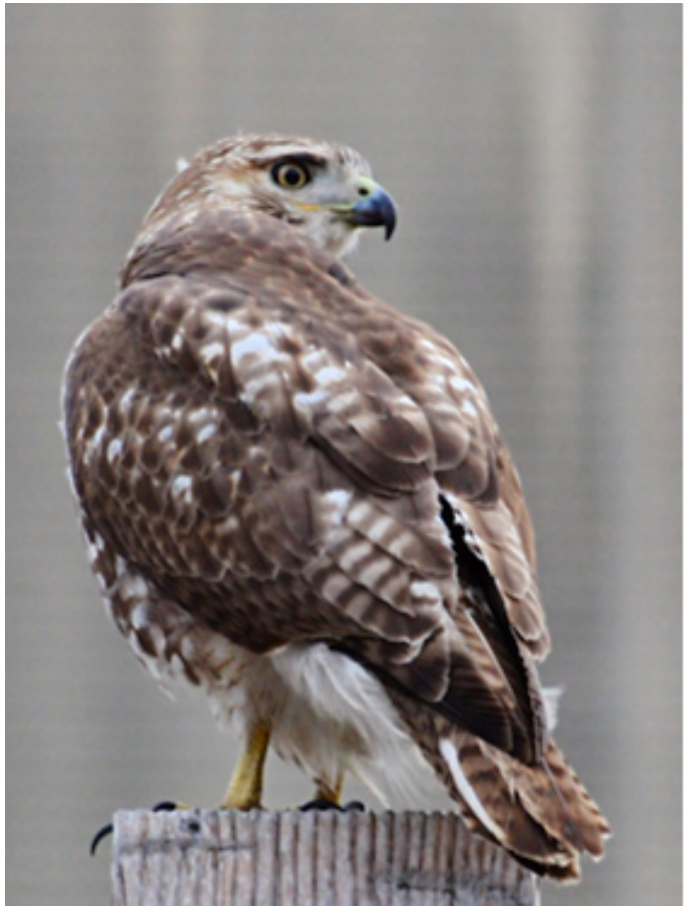
Sharpening examples



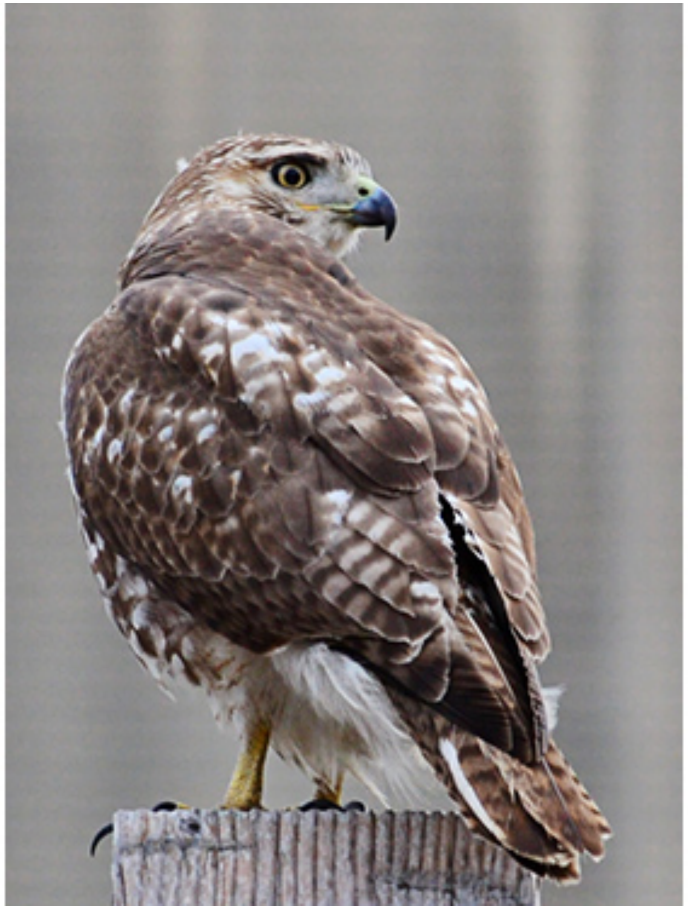


do you see
any problems
in this image?

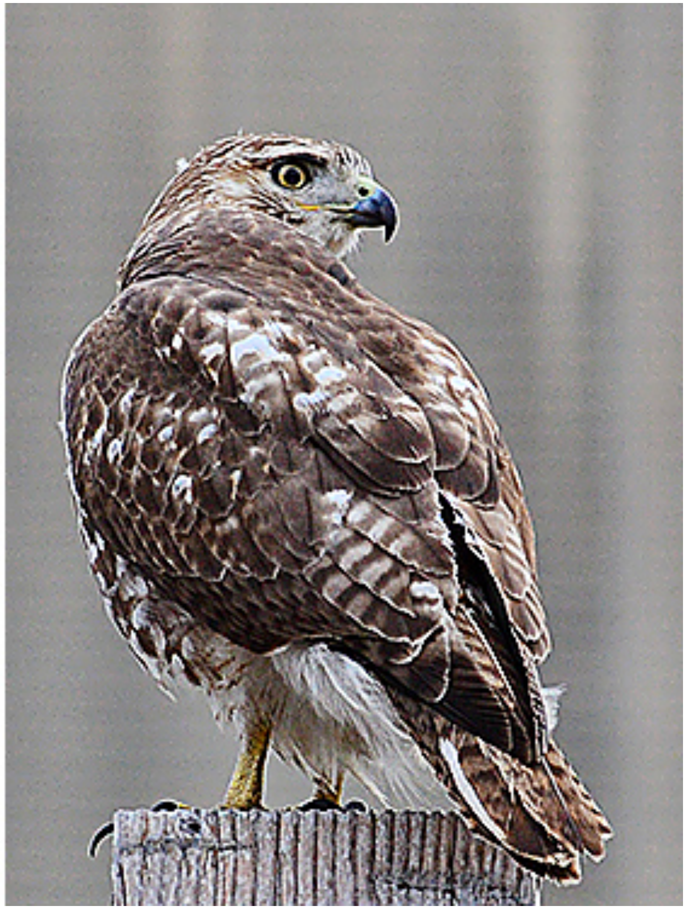
Do not overdo it with sharpening



original



sharpened

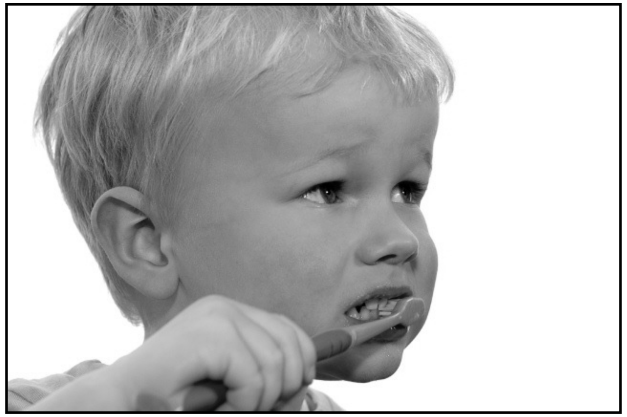


oversharpened

What is wrong in this image?

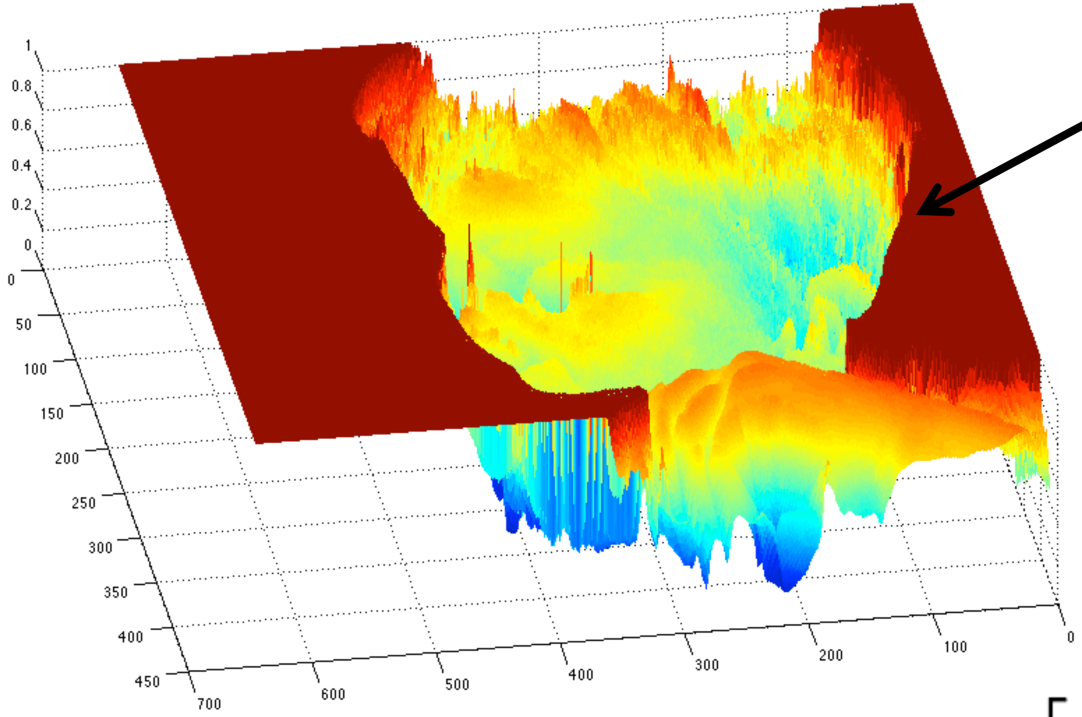
Image gradients

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

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

-1	0	1

?

1	0	-1

?

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

$$\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline \end{array}$$

Sobel filter

What filter
is this?

1D derivative
filter

The Sobel filter

$$\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline \end{array}$$

Sobel filter

Blurring

1D derivative filter

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

The Sobel filter

$$\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \quad = \quad \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline \end{array}$$

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

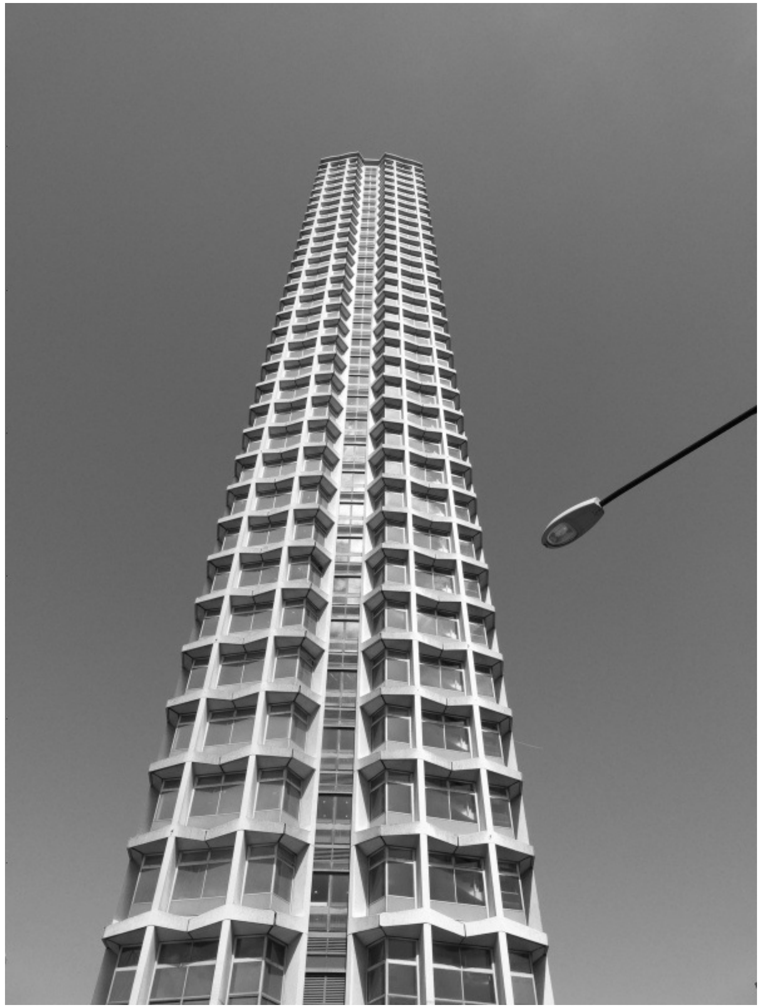
Horizontal Sober filter:

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

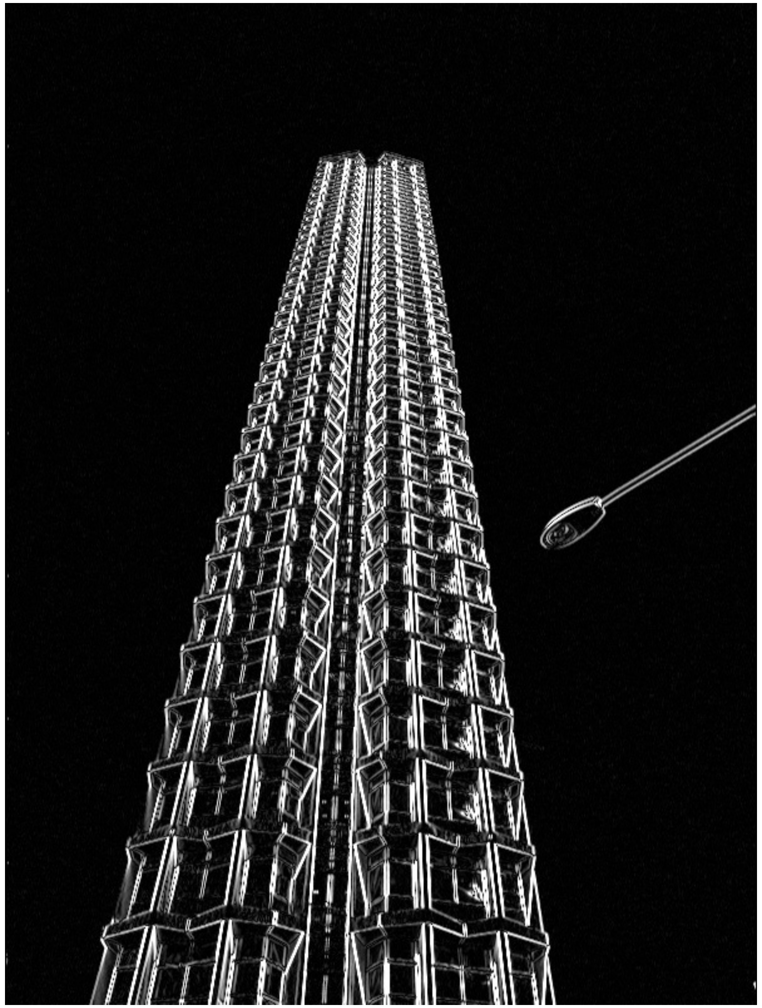
Vertical Sobel filter:

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

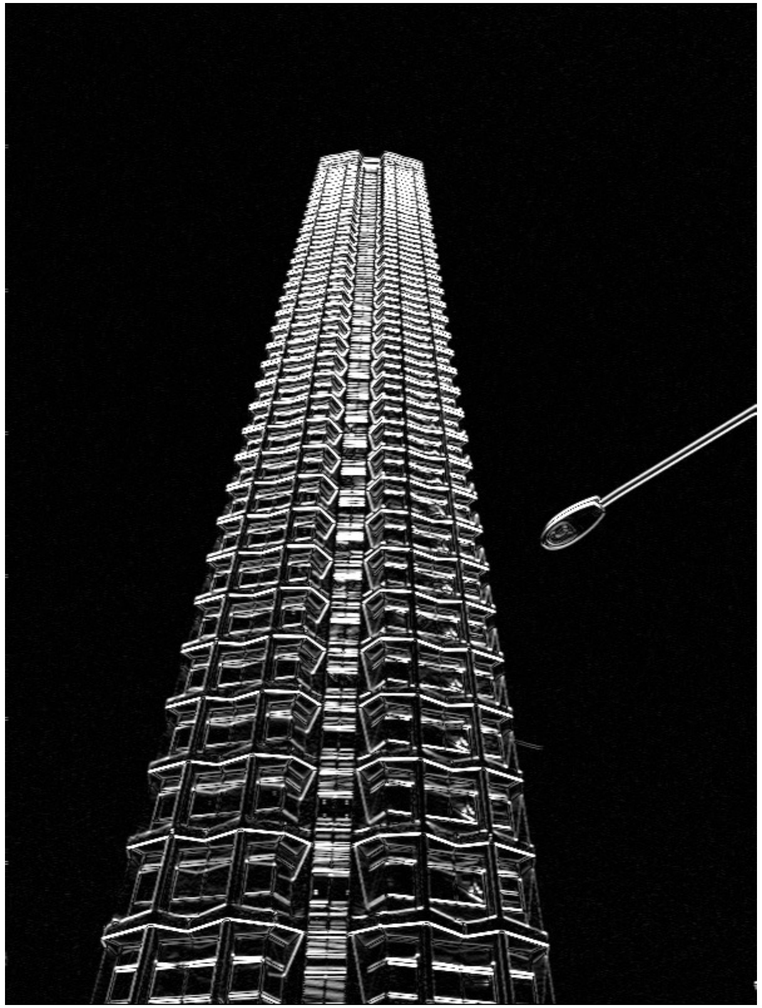
Sobel filter example



original

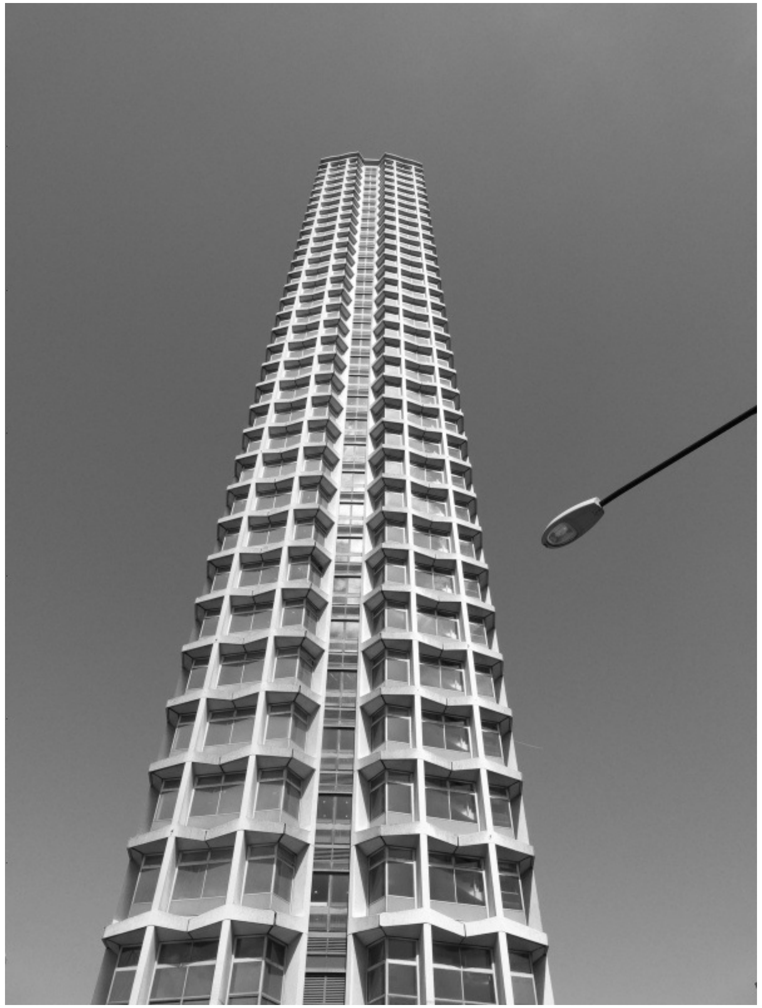


which Sobel filter?

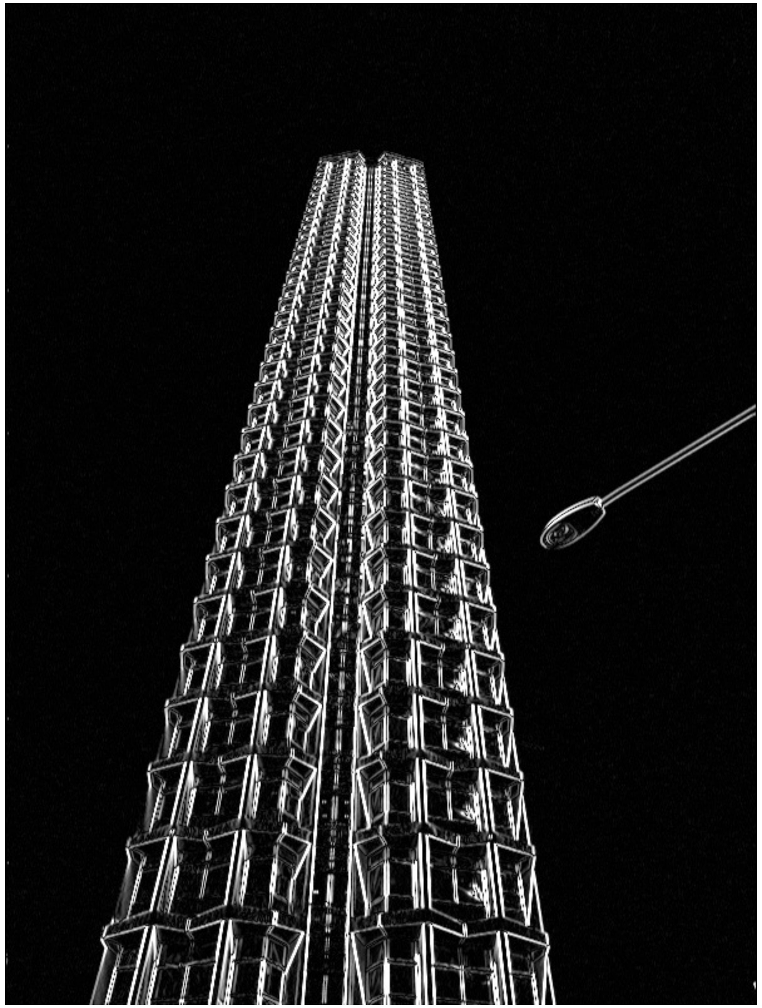


which Sobel filter?

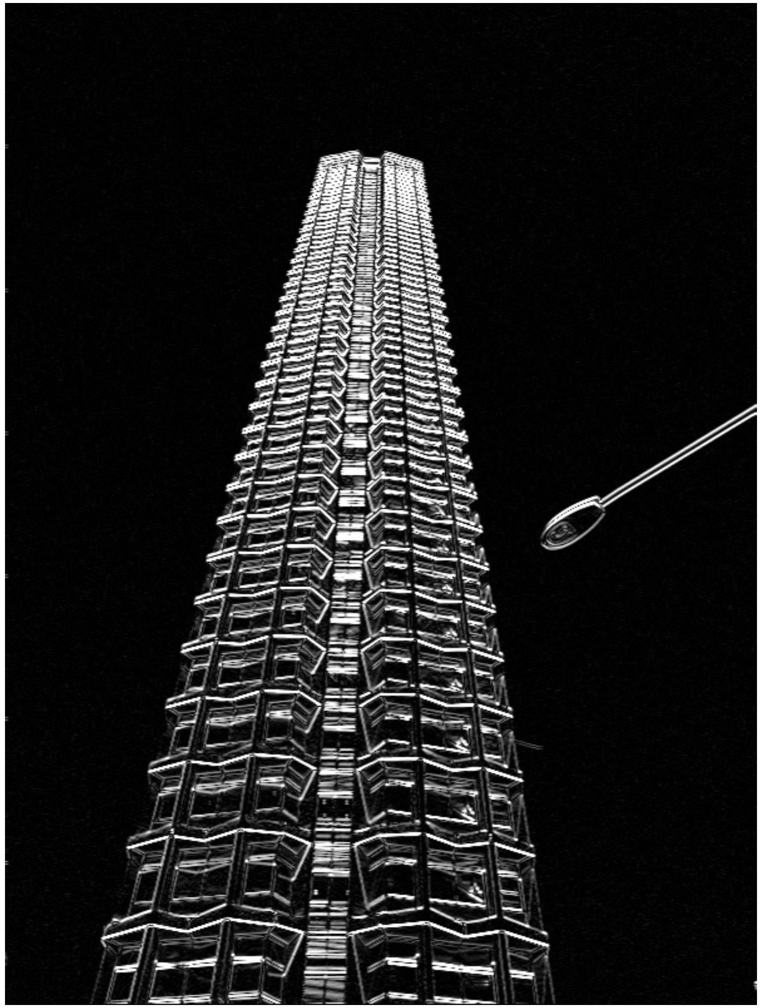
Sobel filter example



original

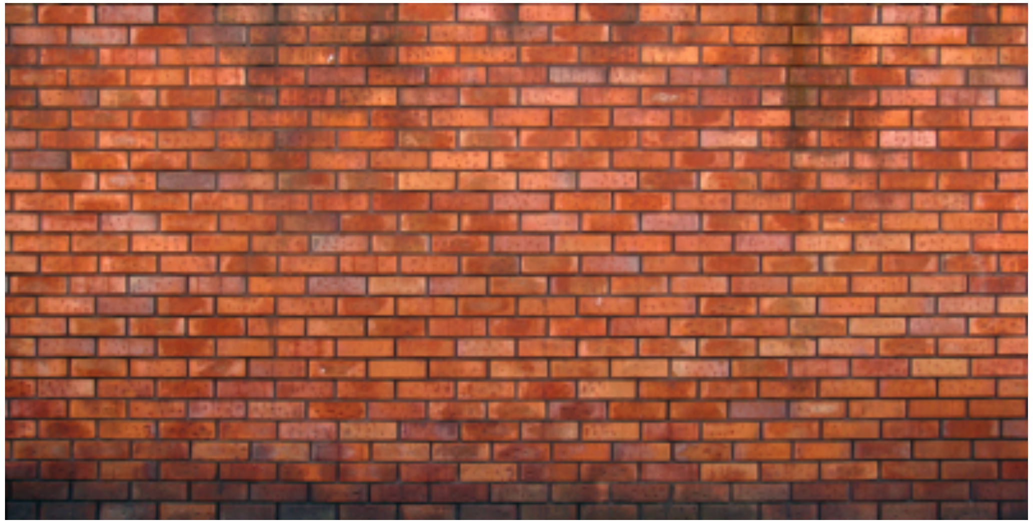


horizontal Sobel filter

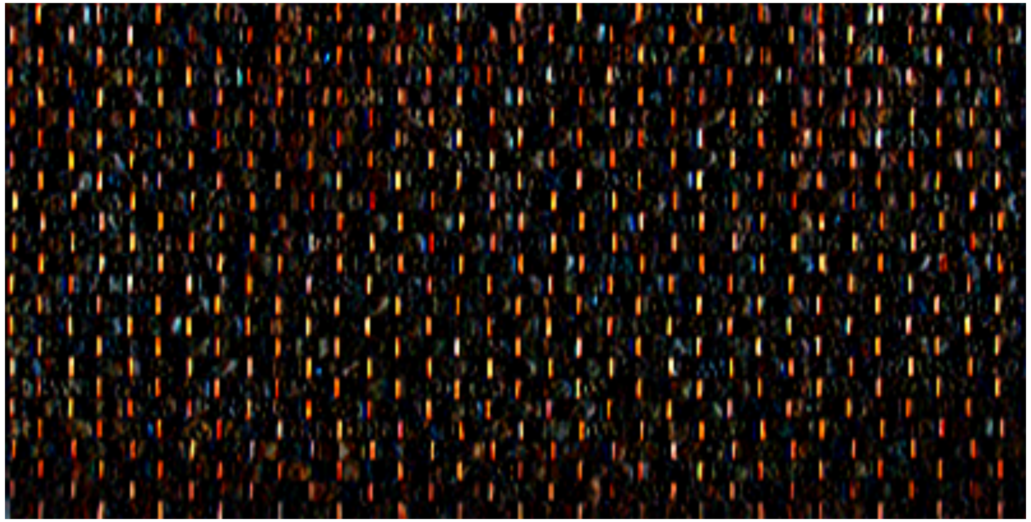


vertical Sobel filter

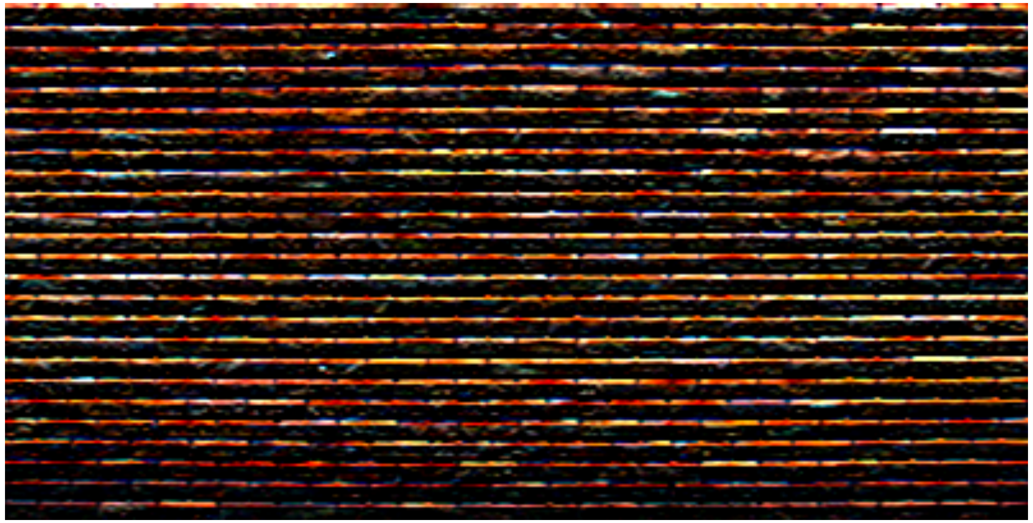
Sobel filter example



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

- 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.

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

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.

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

Computing image gradients

1. Select your favorite derivative filters.

$\boldsymbol{S}_x =$

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

$\boldsymbol{S}_y =$

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

2. Convolve with the image to compute derivatives.

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

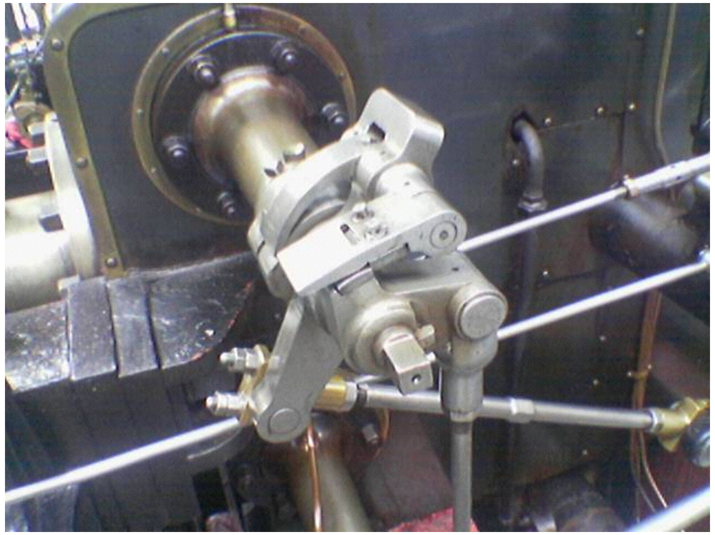
3. Form the image gradient, and compute its direction and amplitude.

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

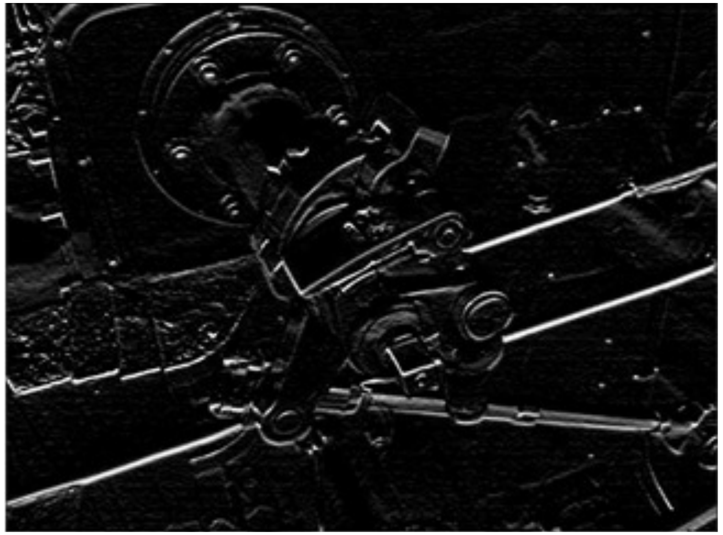
gradient direction amplitude

Image gradient example

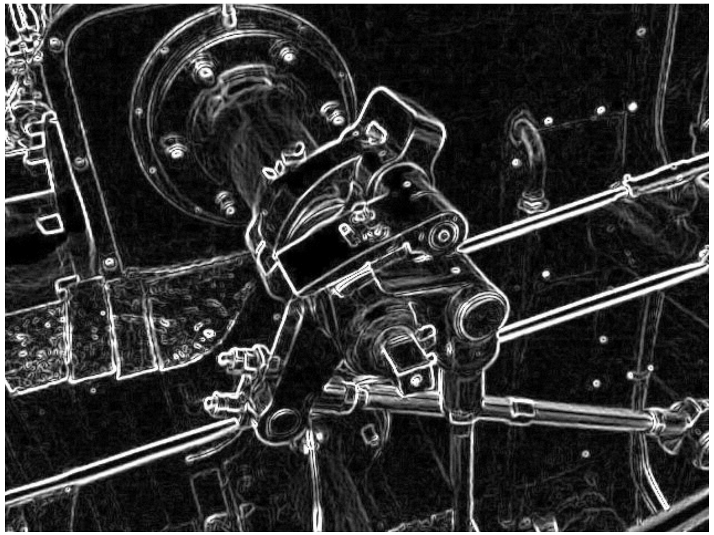
original



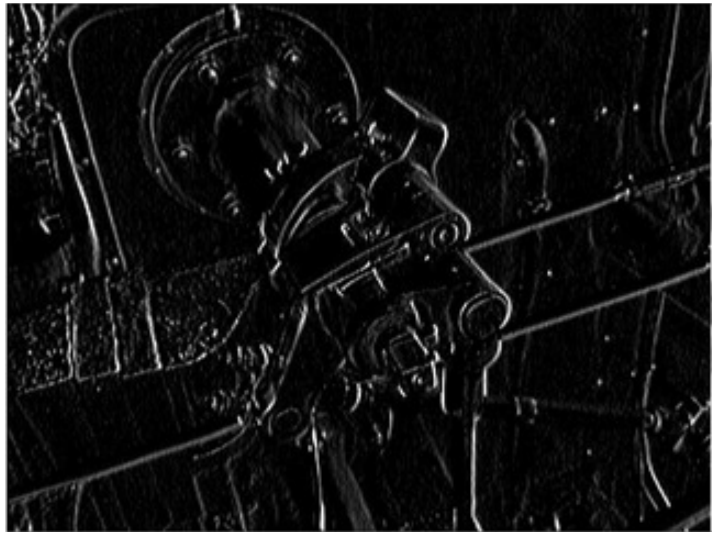
vertical
derivative



gradient
amplitude



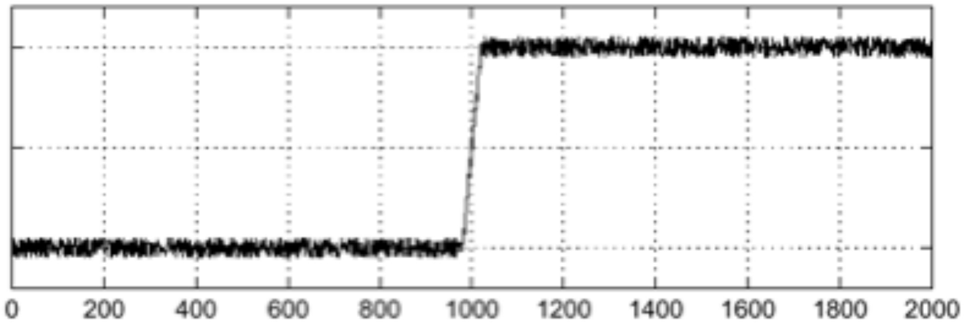
horizontal
derivative



How does the gradient direction relate to these edges?

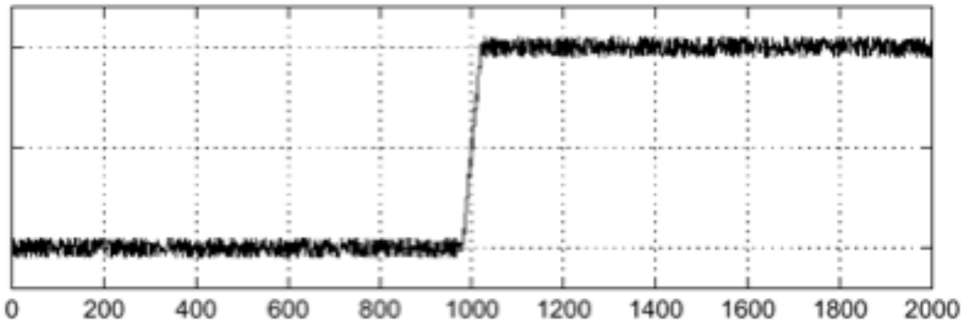
How do you find the edge of this signal?

intensity plot



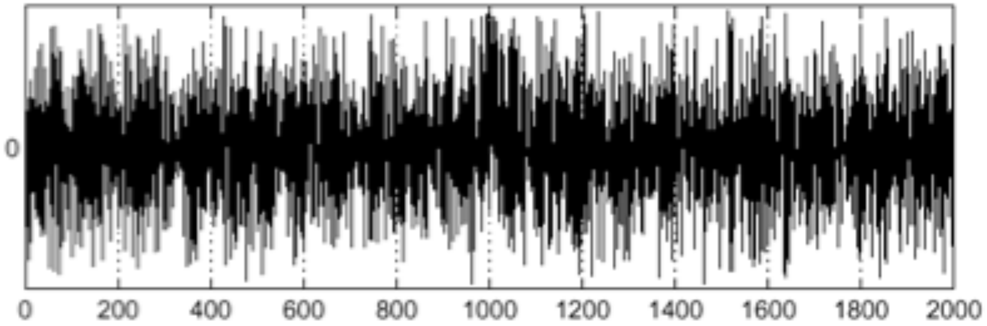
How do you find the edge of this signal?

intensity plot



Using a derivative filter:

derivative plot



What's the problem here?

Differentiation is very sensitive to noise

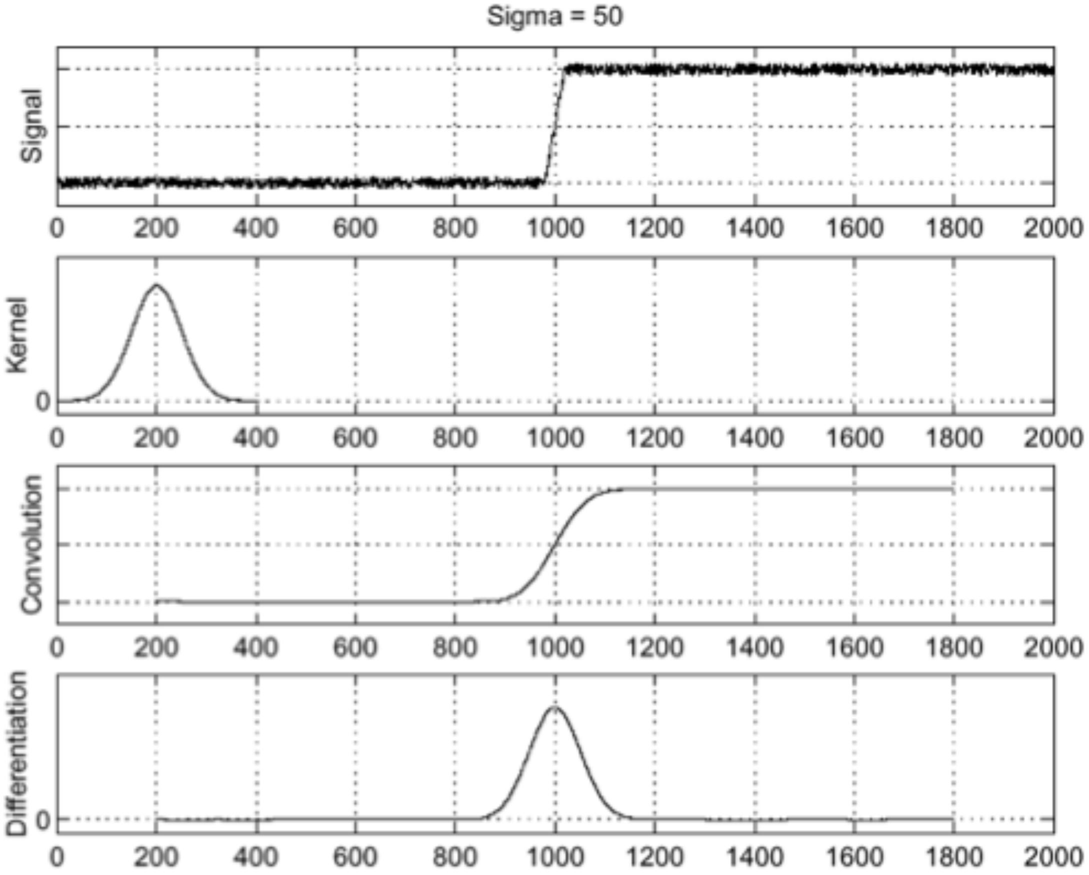
When using derivative filters, it is critical to blur first!

input

Gaussian

blurred

derivative of blurred



How much should we blur?

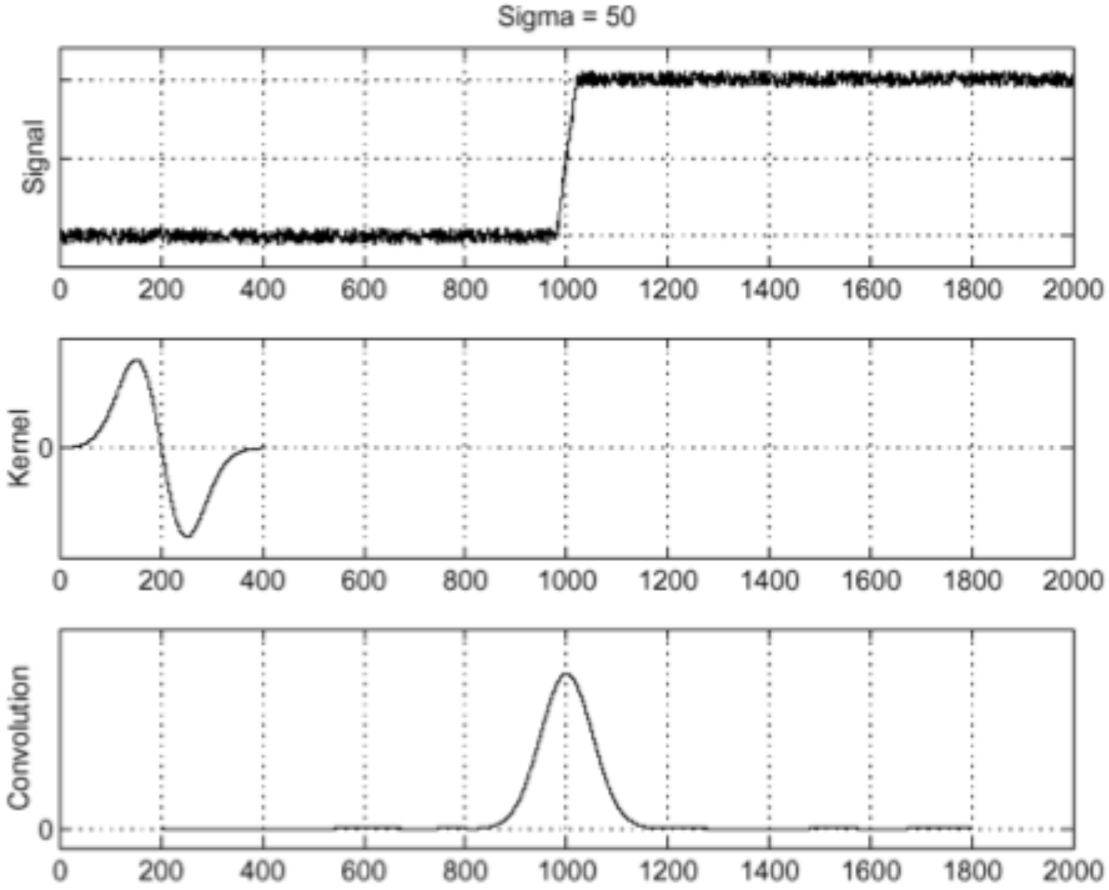
Derivative of Gaussian (DoG) filter

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

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

1D derivative filter

1	0	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} \longrightarrow$$

Laplace filter

1	-2	1

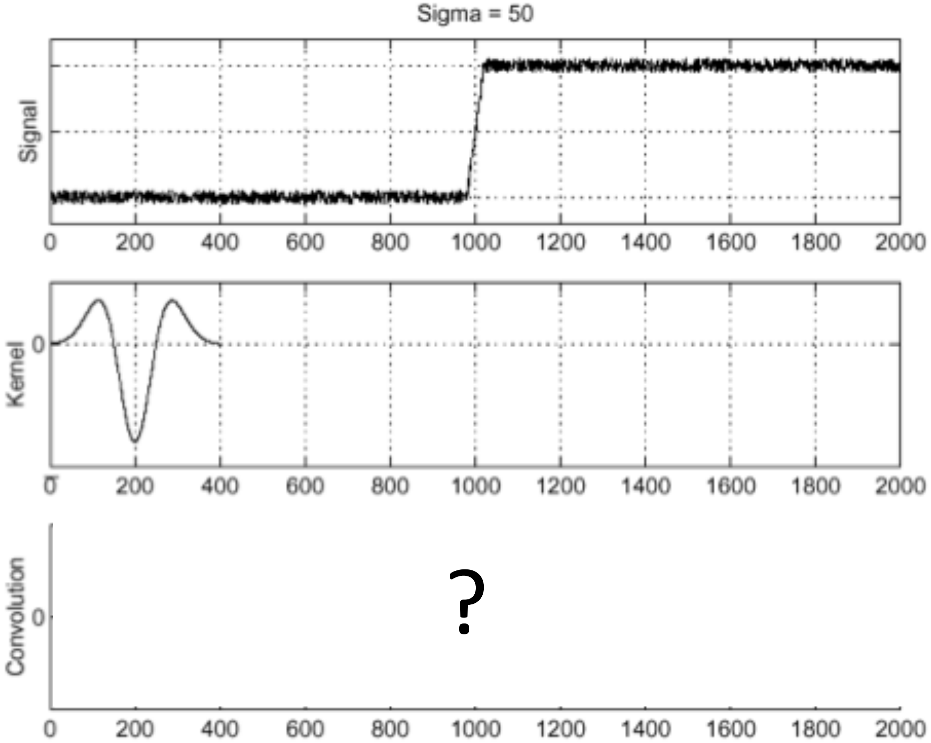
Laplacian of Gaussian (LoG) filter

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

input

Laplacian of Gaussian

output



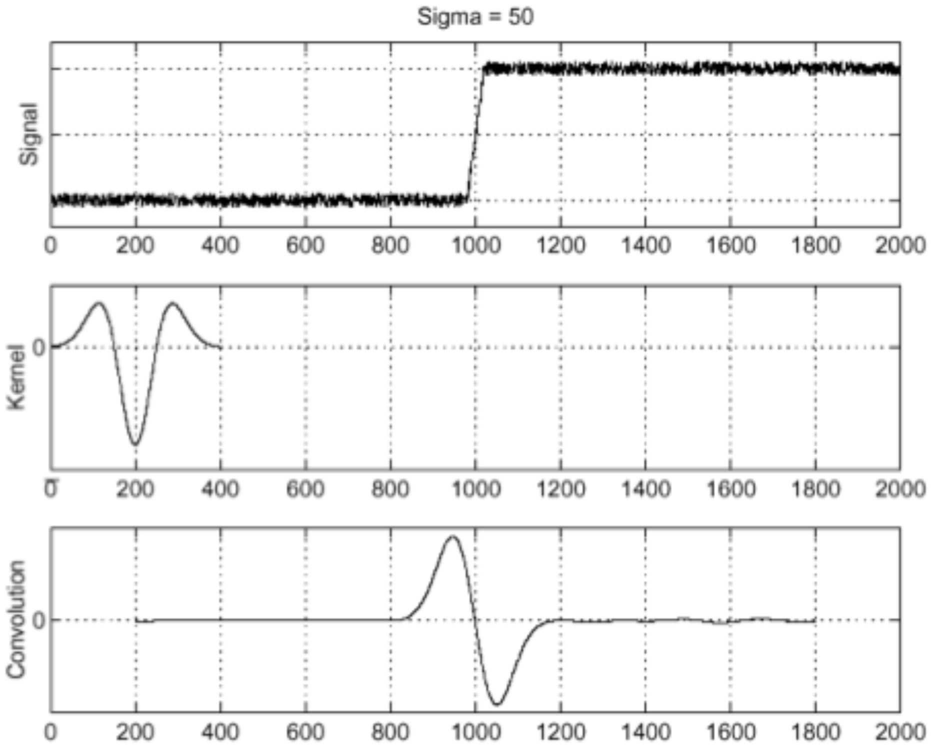
Laplacian of Gaussian (LoG) filter

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

input

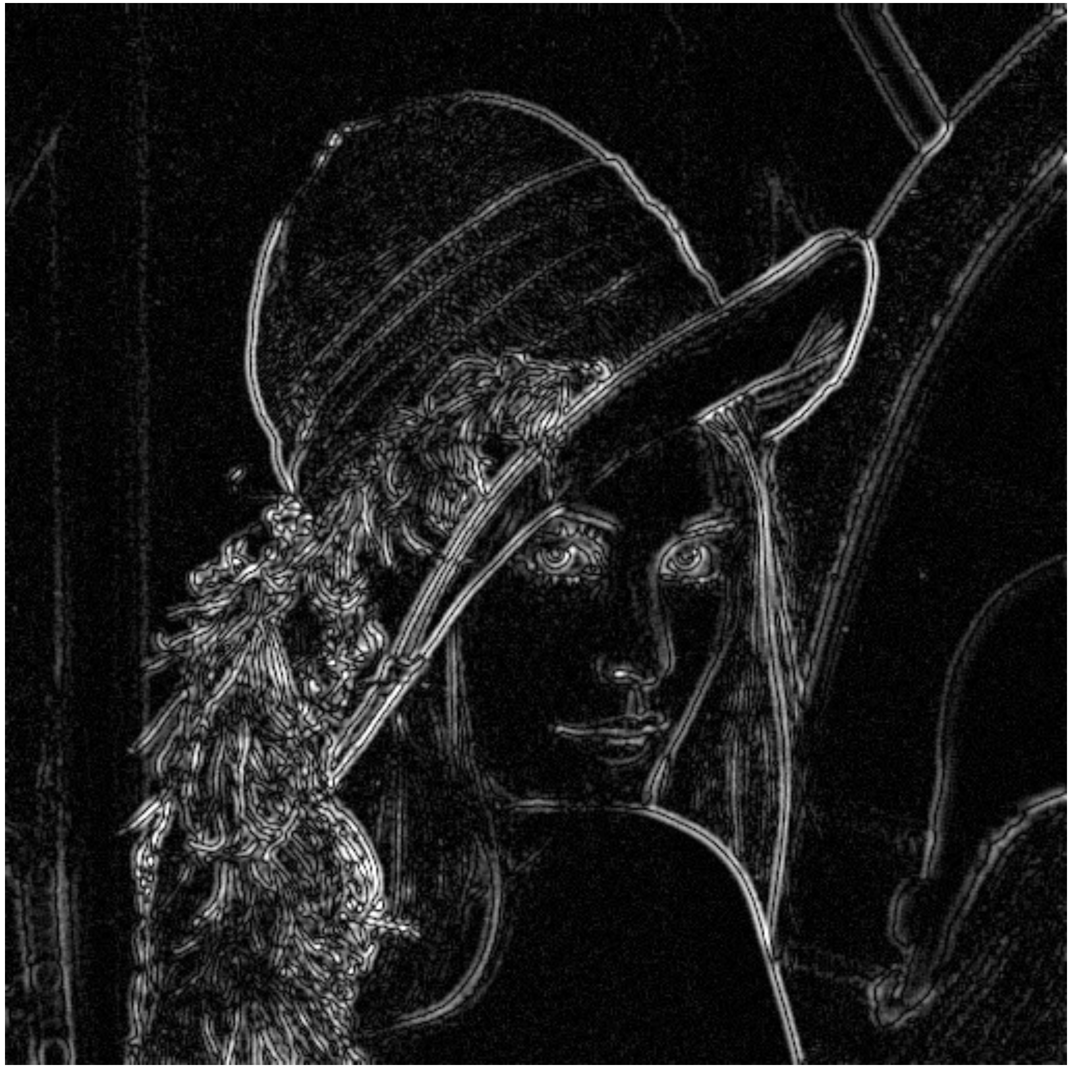
Laplacian of Gaussian

output

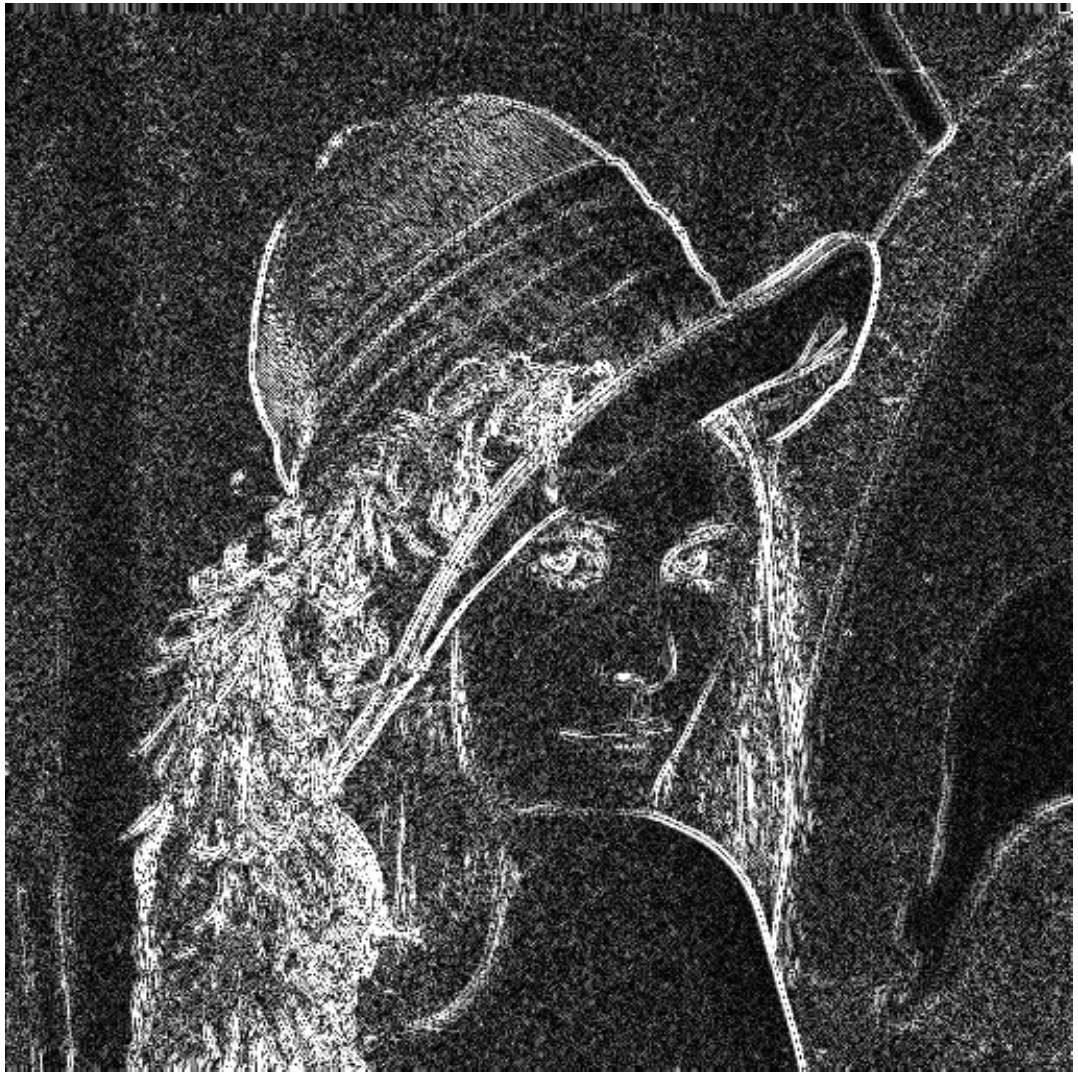


“zero crossings” at edges

Laplace and LoG filtering examples

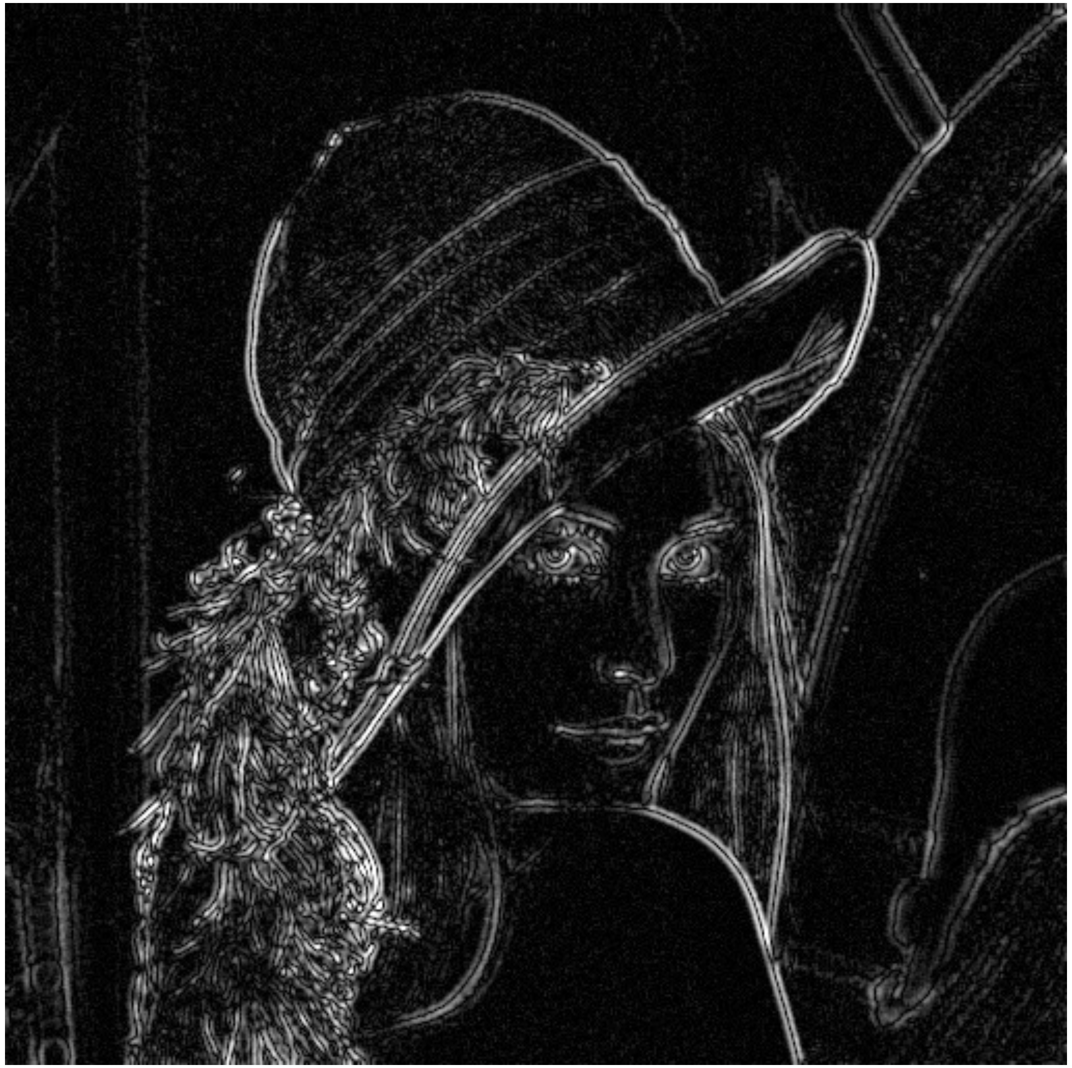


Laplacian of Gaussian filtering

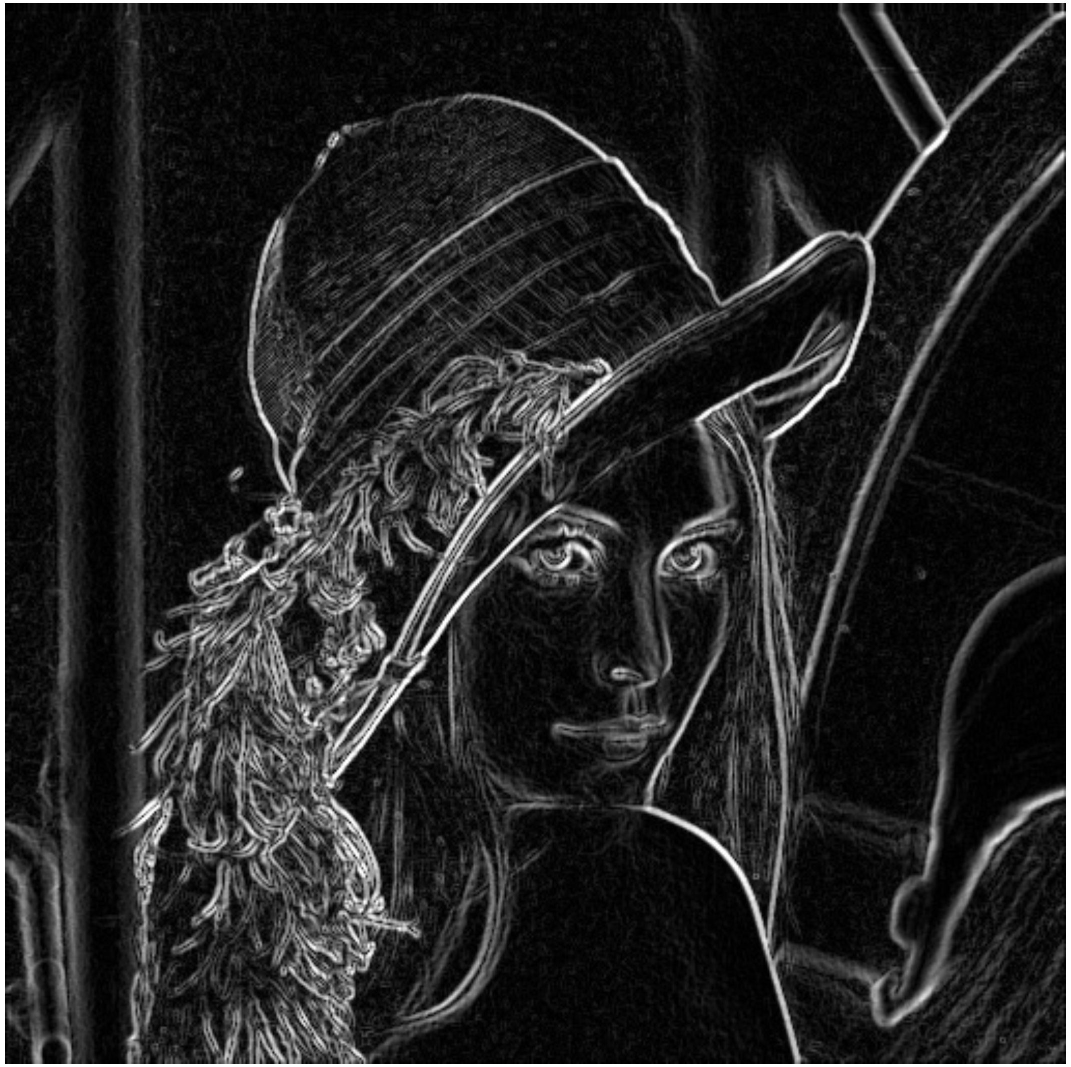


Laplace filtering

Laplacian of Gaussian vs Derivative of Gaussian

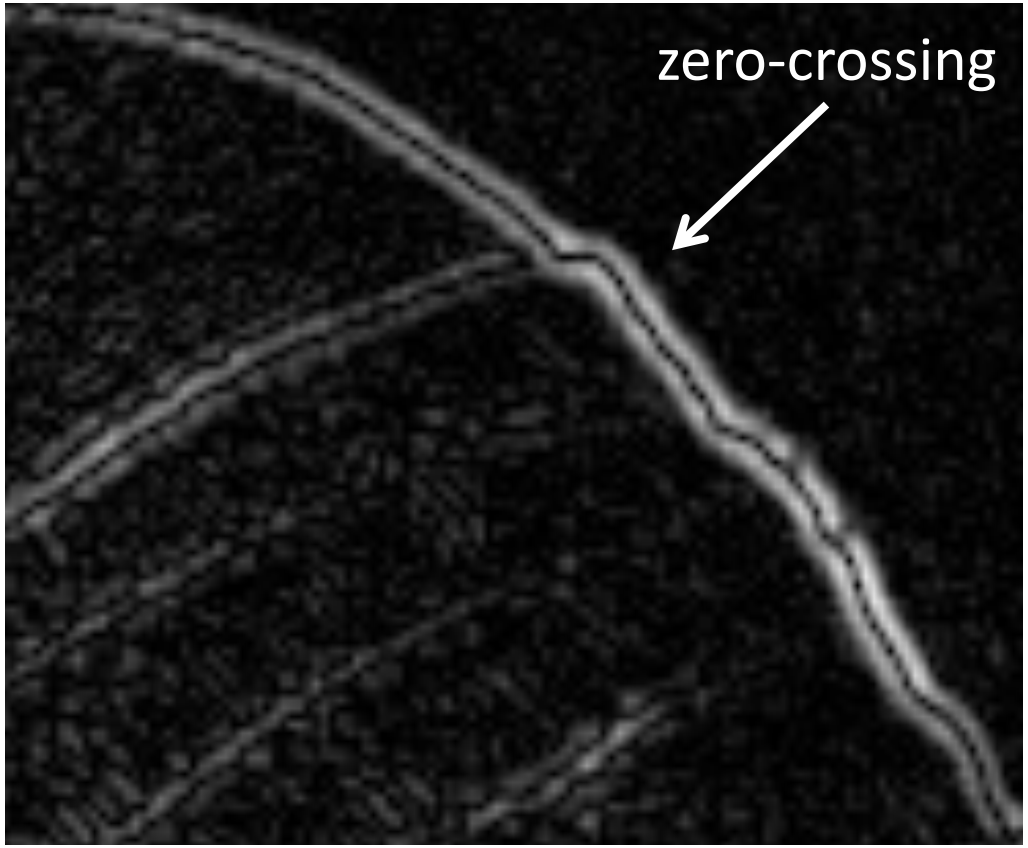


Laplacian of Gaussian filtering

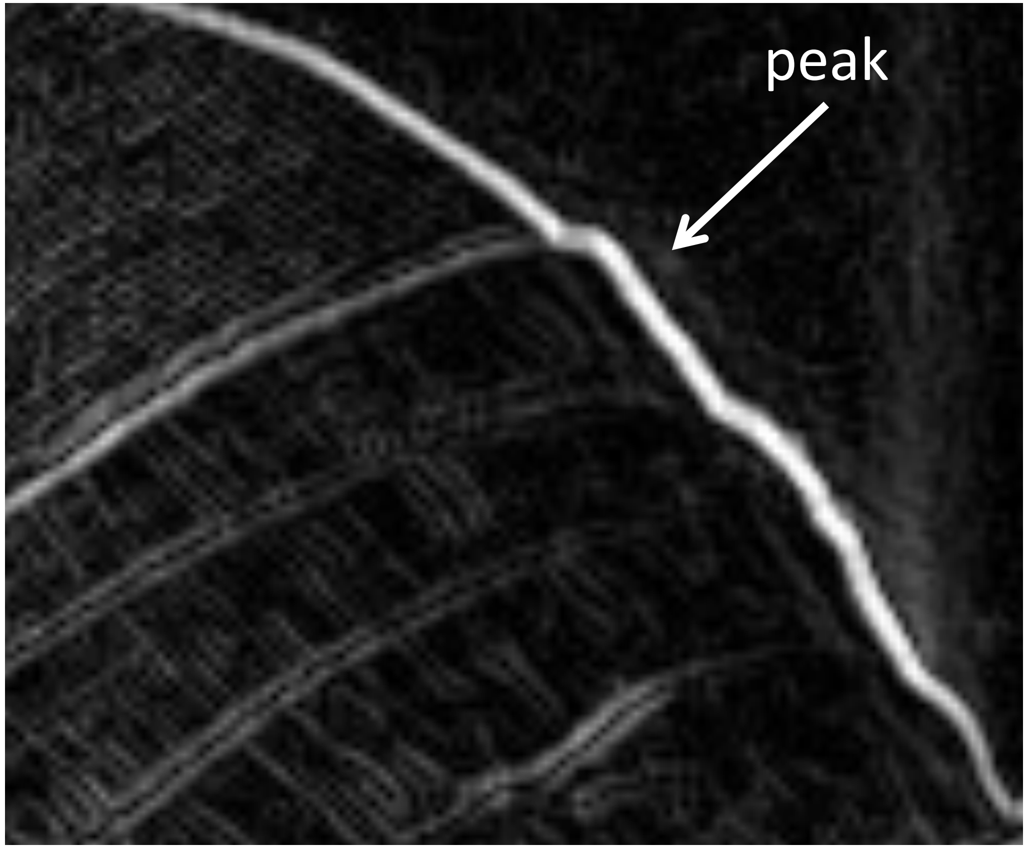


Derivative of Gaussian filtering

Laplacian of Gaussian vs Derivative of Gaussian



Laplacian of Gaussian filtering

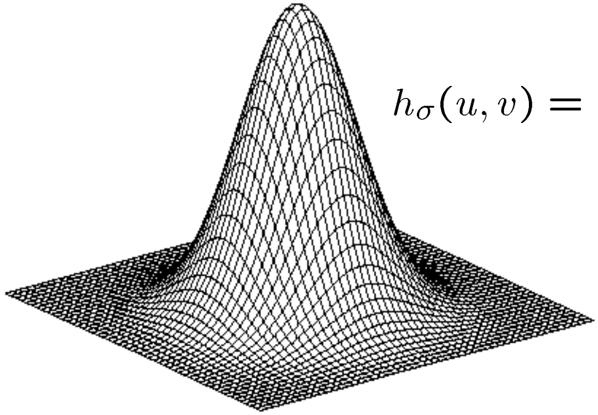


Derivative of Gaussian filtering

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

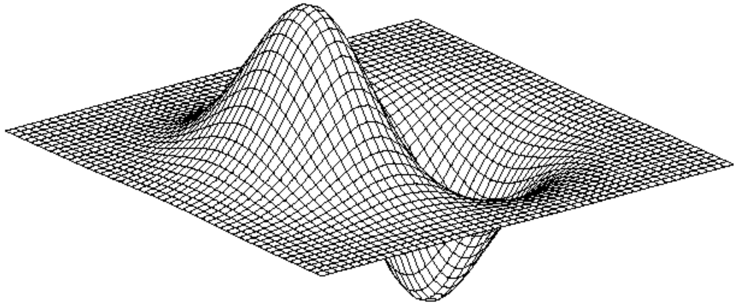
2D Gaussian filters

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



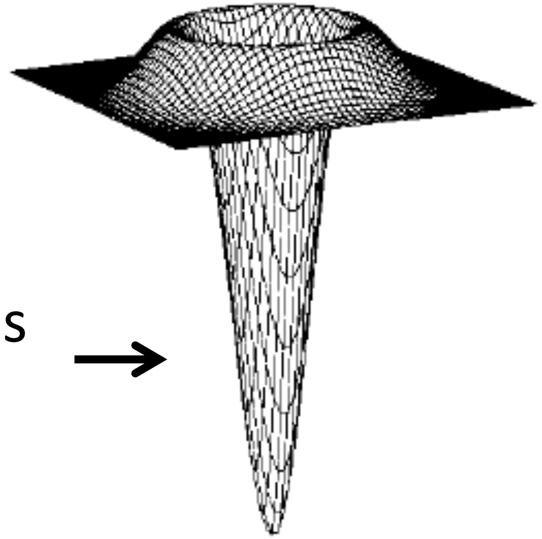
Gaussian

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



Derivative of Gaussian

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



how does this relate to this lecture’s cover picture? →

Laplacian of Gaussian