The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2024

INTRODUCTION TO COMPUTER VISION

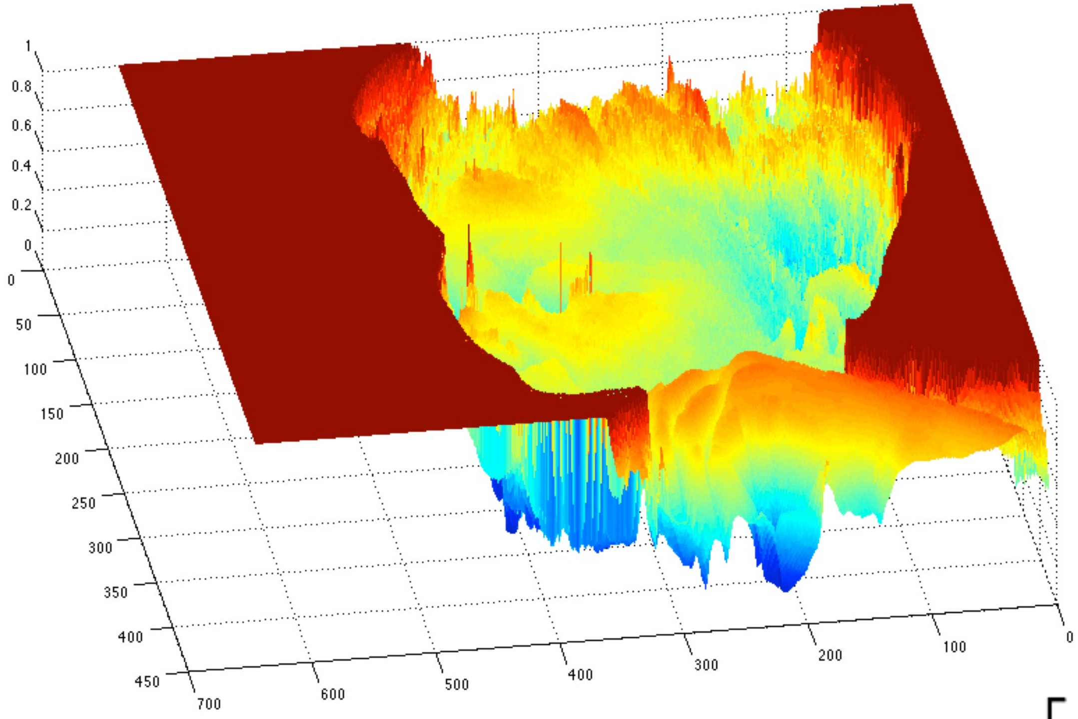
Atlas Wang
Associate Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

What is an image?

grayscale image

$f(\boldsymbol{x})$



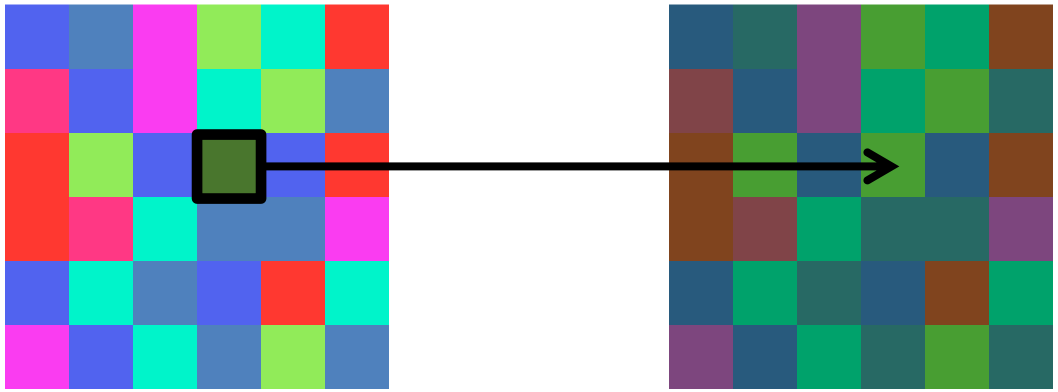
domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

A (grayscale) image is a 2D function.

What is the range of the image function f?

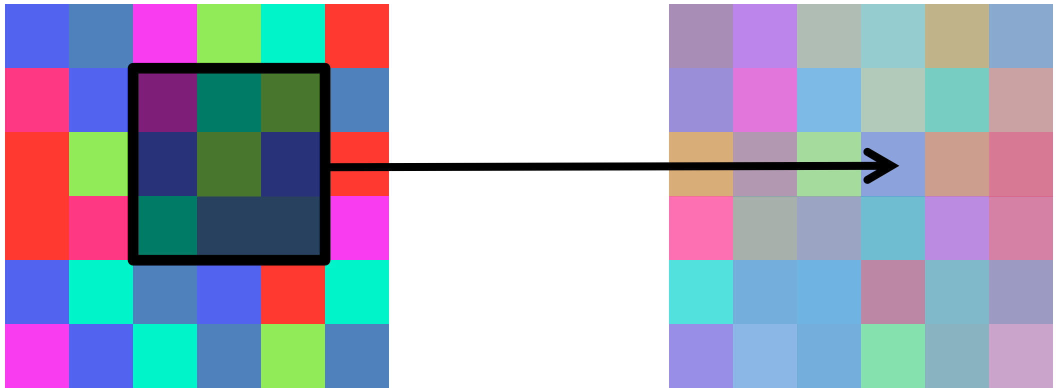
What types of image filtering can we do?

Point Operation



point processing

Neighborhood Operation



“filtering”

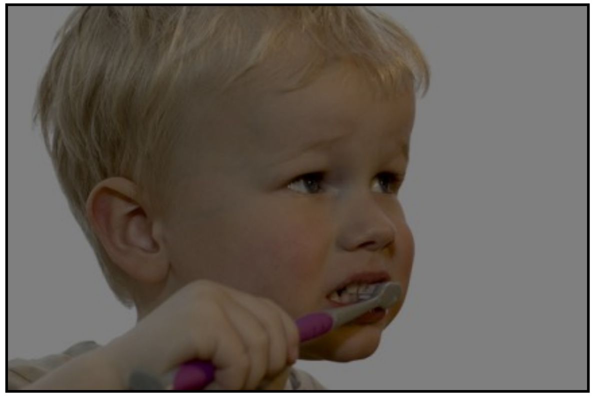
How would you implement these?

Examples of point processing

original

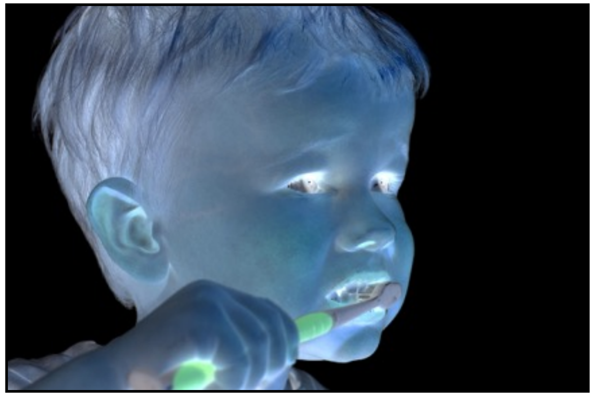
darken

lower contrast



non-linear raise contrast

invert



lighten

raise contrast

non-linear lower contrast

How would you implement these?

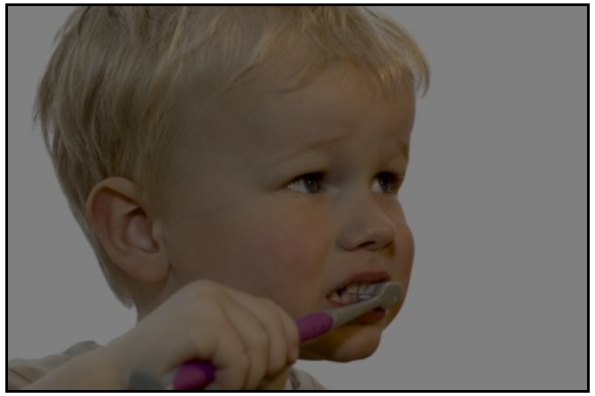
Examples of point processing

original

x

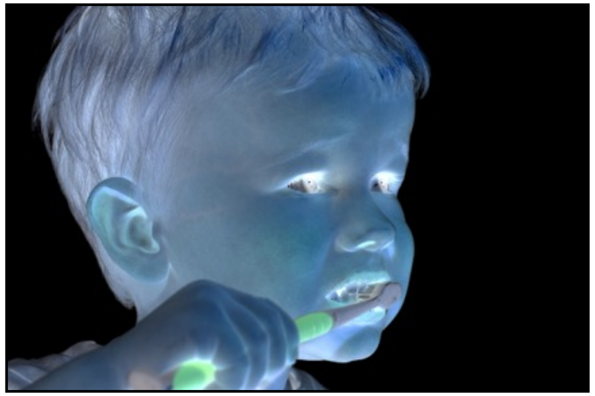
darken

lower contrast



non-linear raise contrast

invert



lighten

raise contrast

non-linear lower contrast

How would you implement these?

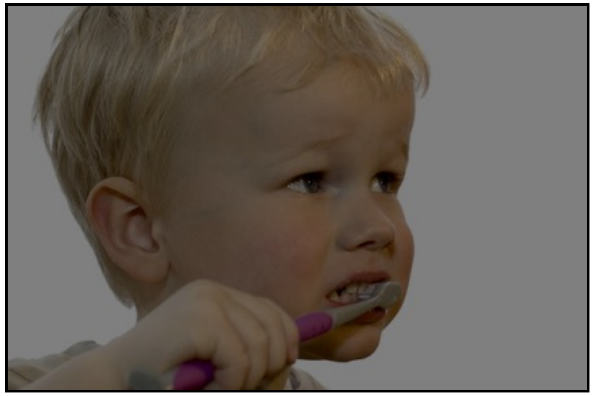
Examples of point processing

original

x

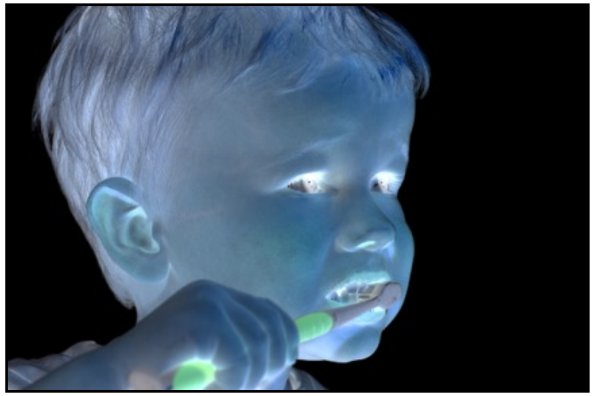
darken

lower contrast



non-linear raise contrast

invert



$255 - x$

lighten

raise contrast

non-linear lower contrast

How would you implement these?

Examples of point processing

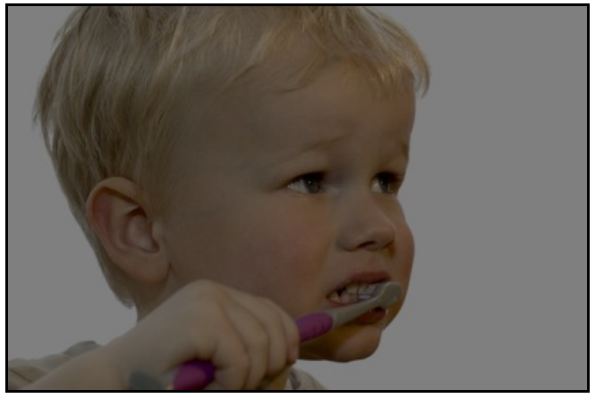
original

x

darken

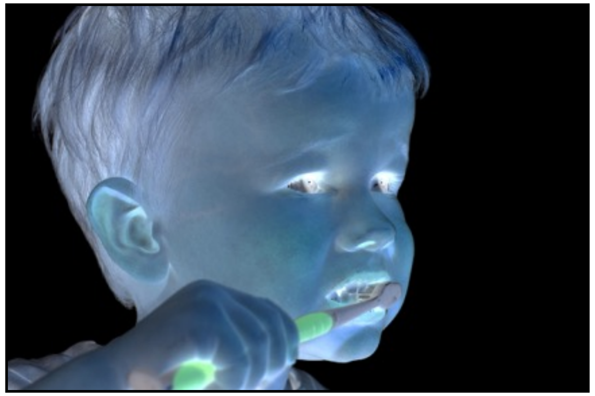
$x - 128$

lower contrast



non-linear raise contrast

invert



$255 - x$

lighten

raise contrast

non-linear lower contrast

How would you implement these?

Examples of point processing

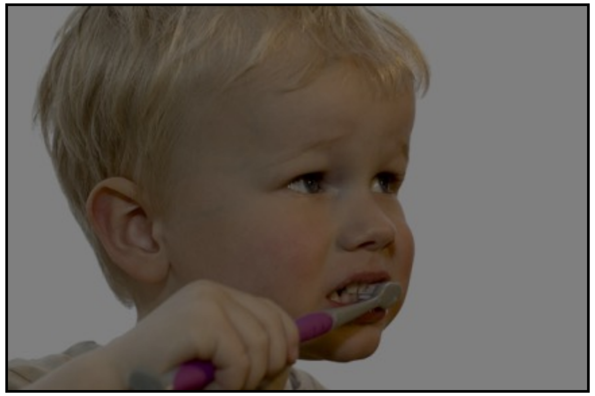
original

x

darken

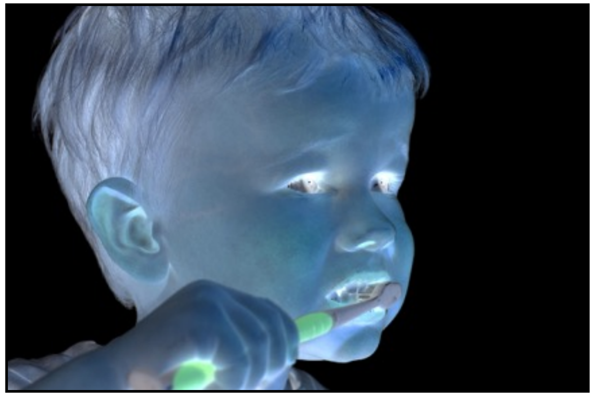
$x - 128$

lower contrast



non-linear raise contrast

invert



$255 - x$

lighten

$x + 128$

raise contrast

non-linear lower contrast

How would you implement these?

Examples of point processing

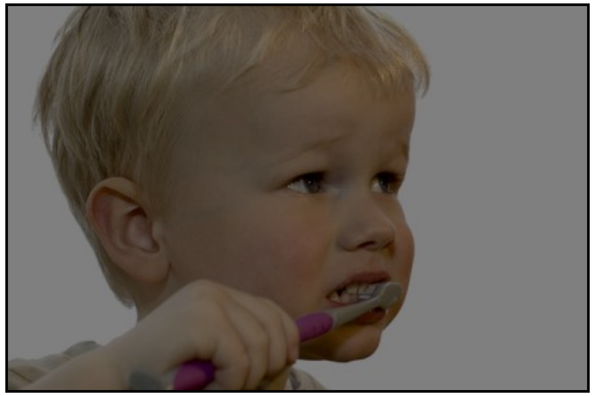
original

x

darken

$x - 128$

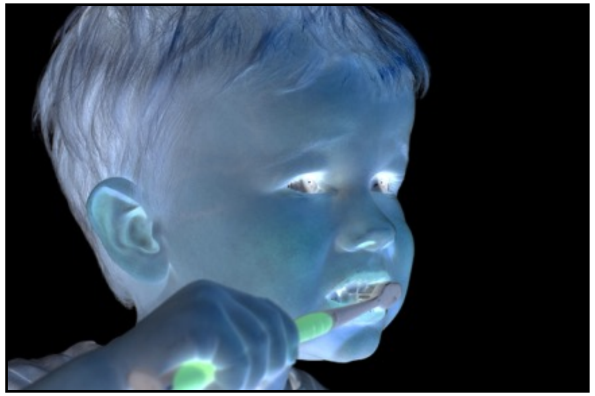
lower contrast



$\frac{x}{2}$

non-linear raise contrast

invert



$255 - x$

lighten

$x + 128$

raise contrast

non-linear lower contrast

How would you implement these?

Examples of point processing

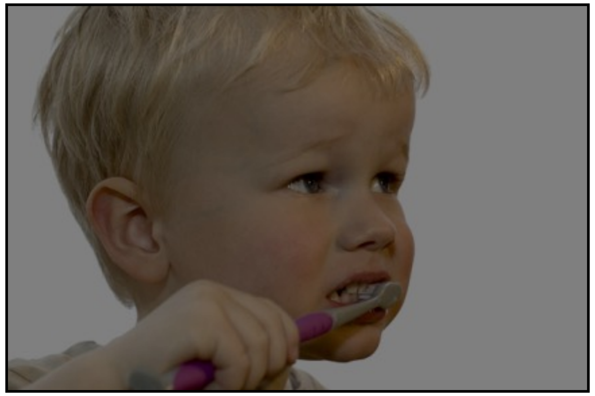
original

x

darken

$x - 128$

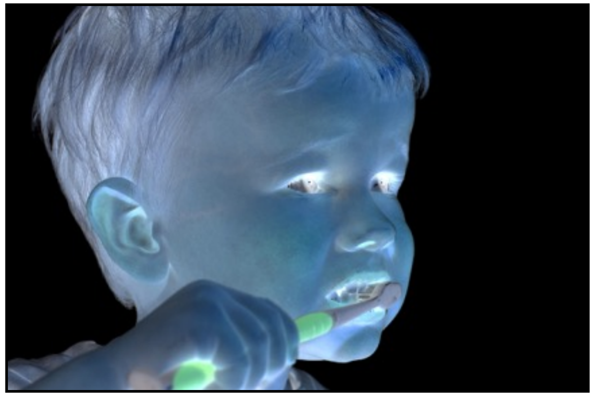
lower contrast



$\frac{x}{2}$

non-linear raise contrast

invert



$255 - x$

lighten

$x + 128$

raise contrast

$x \times 2$

non-linear lower contrast

How would you implement these?

Examples of point processing

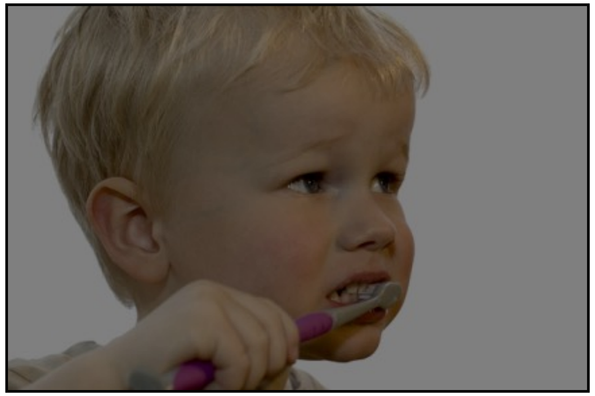
original

$$x$$

darken

$$x - 128$$

lower contrast

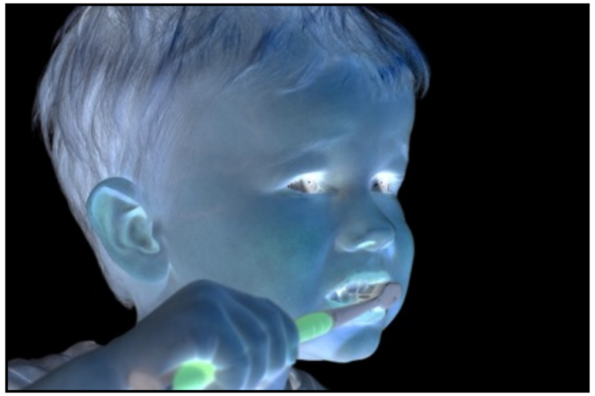


$$\frac{x}{2}$$

non-linear raise contrast

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

invert



$$255 - x$$

lighten

$$x + 128$$

raise contrast

$$x \times 2$$

non-linear lower contrast

How would you implement these?

Examples of point processing

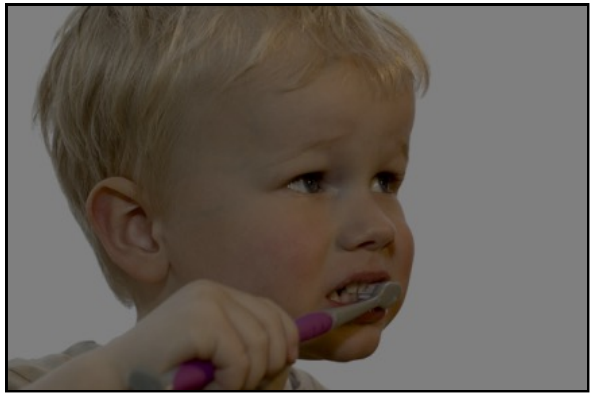
original

$$x$$

darken

$$x - 128$$

lower contrast

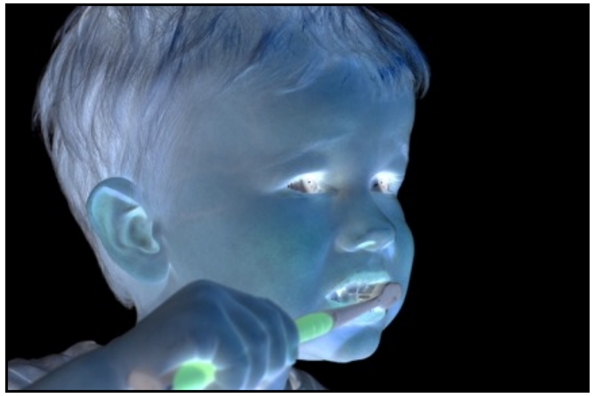


$$\frac{x}{2}$$

non-linear raise contrast

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

invert



$$255 - x$$

lighten

$$x + 128$$

raise contrast

$$x \times 2$$

non-linear lower contrast

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

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.
- Modern name? Convolution (*yes, the same guy in convolutional neural network*)

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

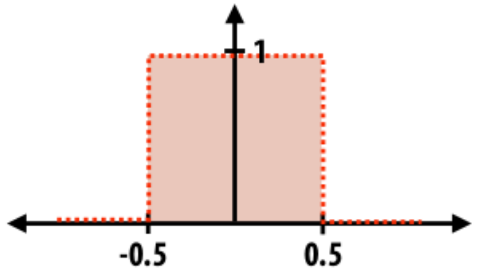
filter

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 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

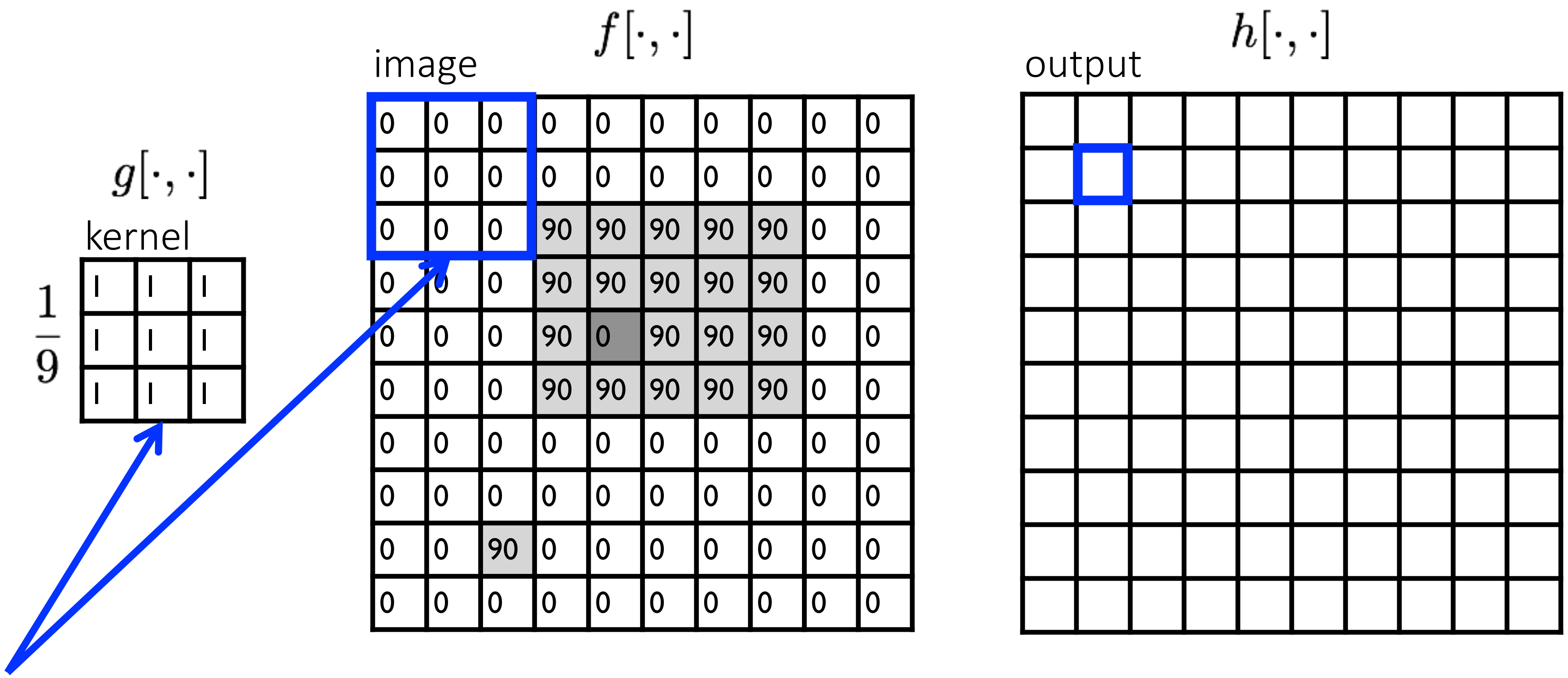
Simplest Convolution: the box filter

- also known as the 2D rectangular 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



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)

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

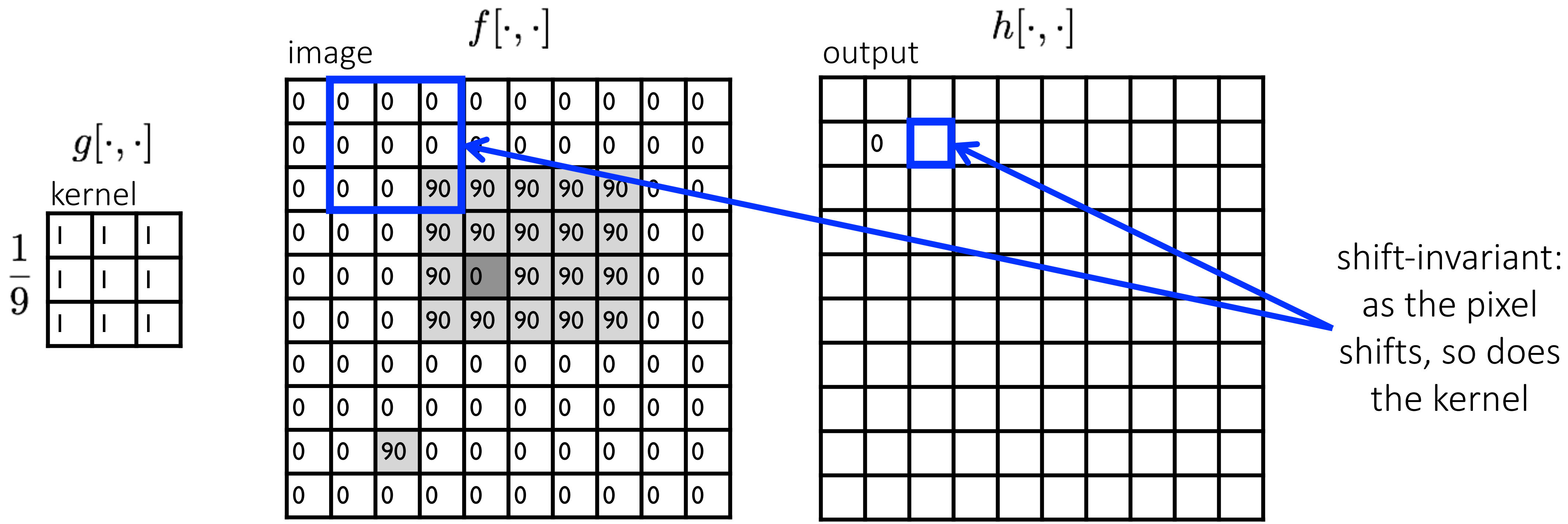
output $h[\cdot, \cdot]$

	0								

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

output — filter — image (signal)

Let’s run the box filter



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

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

output — filter — image (signal)

Let’s run the box filter

kernel $g[\cdot,\cdot]$

$\frac{1}{9}$ ×

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							

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

output — filter — image (signal)

Let’s run the box filter

kernel $g[\cdot,\cdot]$

$\frac{1}{9}$

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							

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

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						

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

output — filter — image (signal)

Let’s run the box filter

kernel $g[\cdot,\cdot]$

$\frac{1}{9}$

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					

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

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				

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

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			

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

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

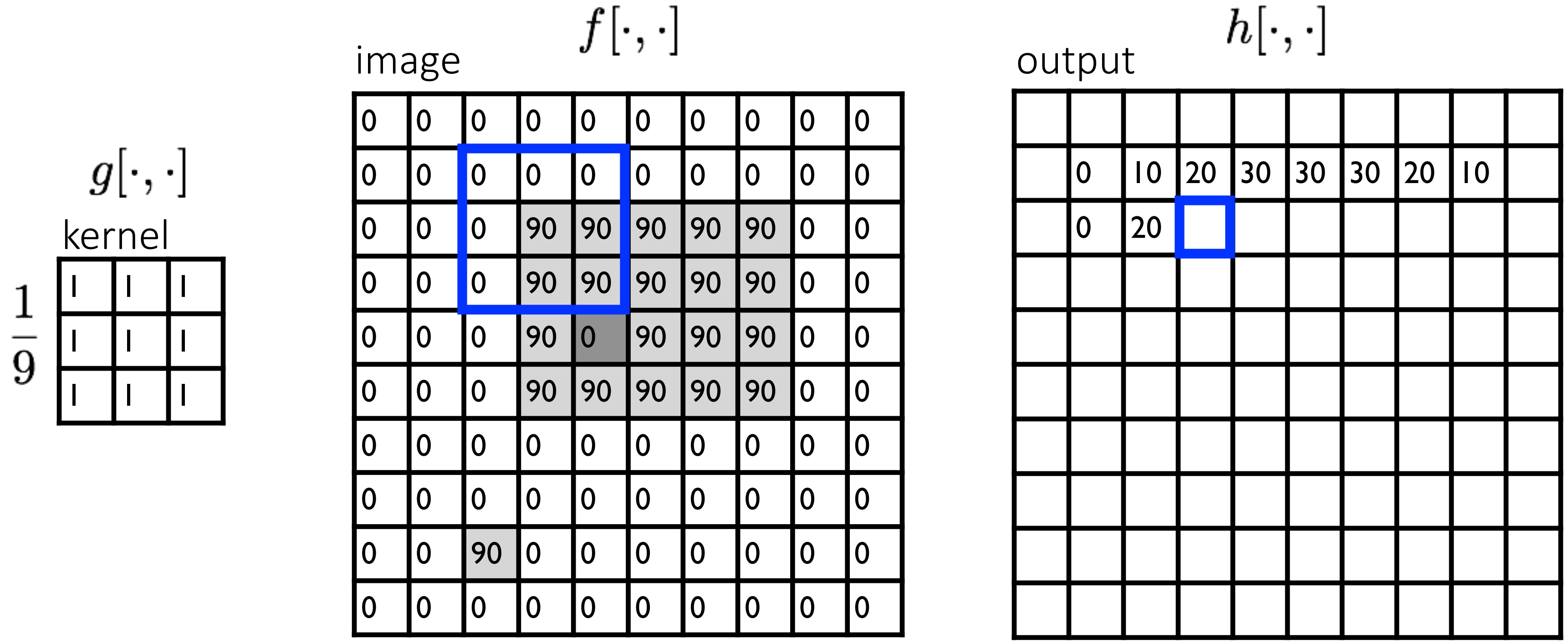
output $h[\cdot,\cdot]$

	0	10	20	30	30	30	20	10	
	0								

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

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

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						

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

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								

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

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							

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

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								

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

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

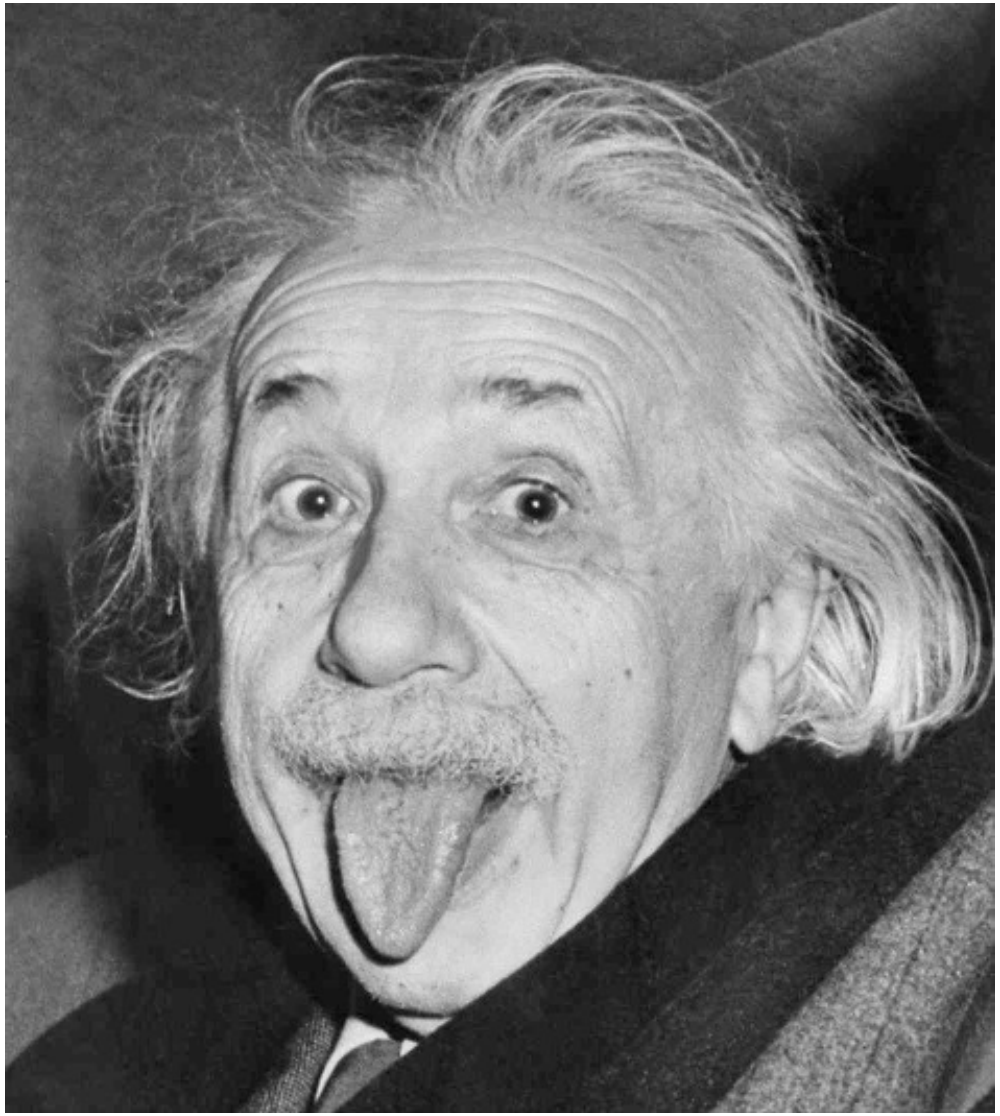
output $h[\cdot,\cdot]$

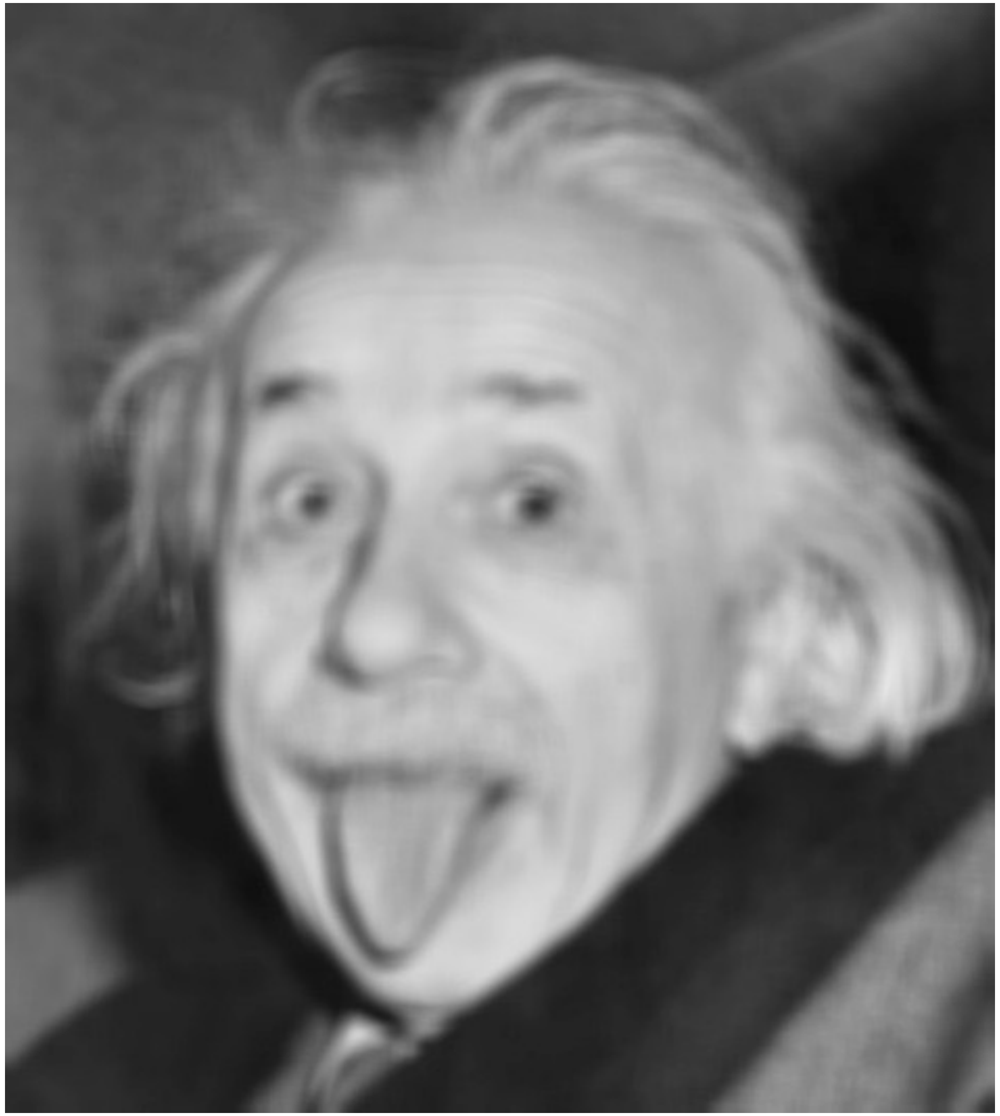
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	10	10	10	0	0	0	0	

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

output filter image (signal)

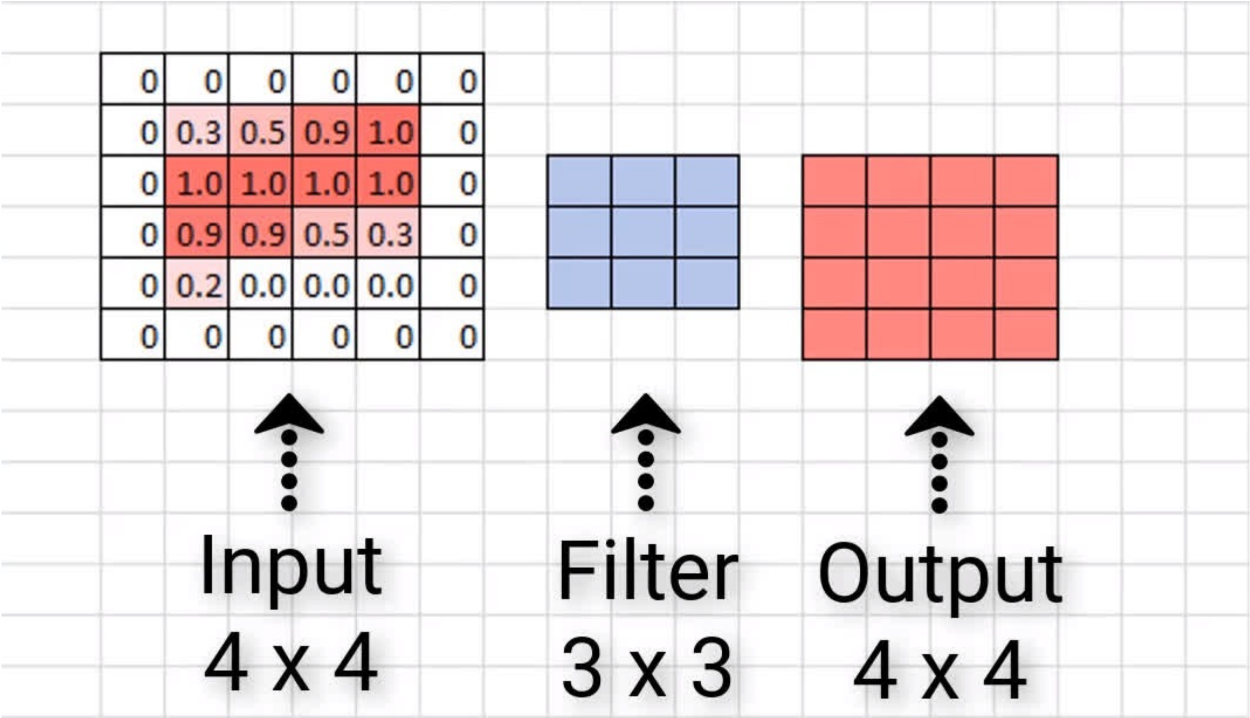
Some more realistic examples





Practical matters: what about near the edge?

- The filter window falls off the edge of the image
- Need to extrapolate!
- Common ways:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge
  - .....



Separable filters

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

example:
box filter

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

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

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

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

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

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

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?

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? ⟶ $M^2 \times N^2$
- 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

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

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

The Gaussian filter

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

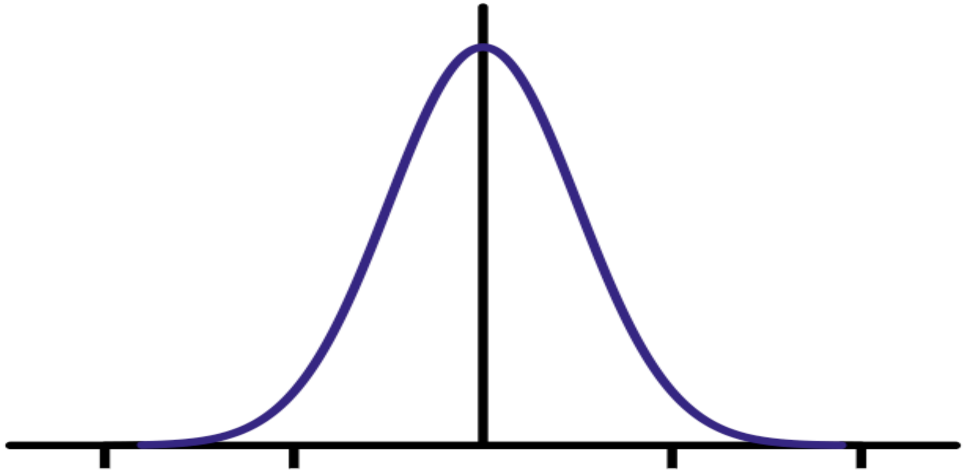
- kernel values sampled from the 2D Gaussian function:

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

- 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

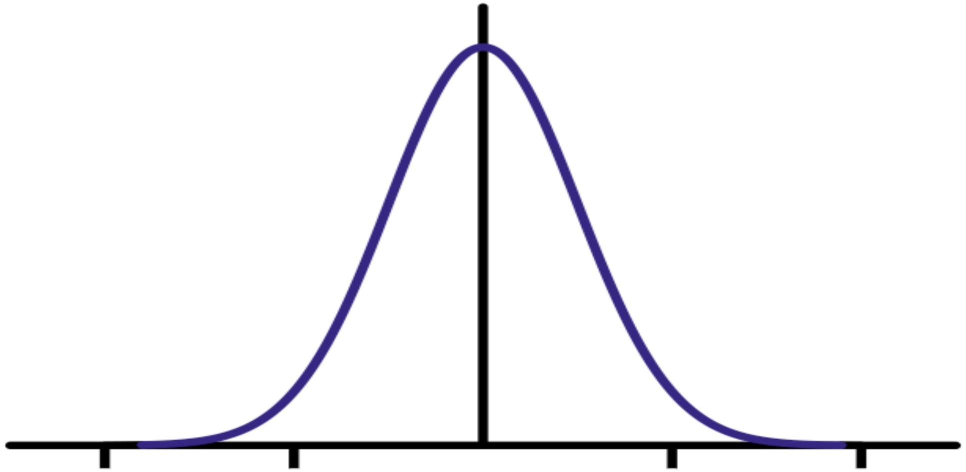
- kernel values sampled from the 2D Gaussian function:

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

- 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:

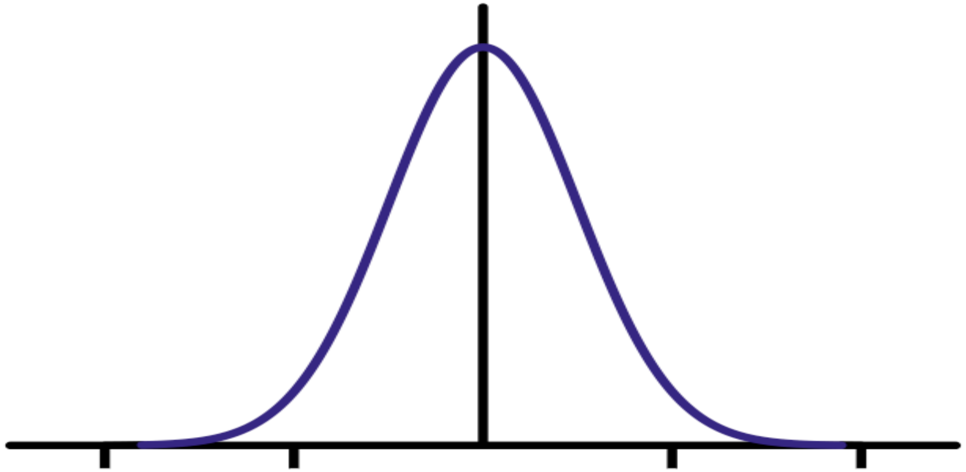
$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

- 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

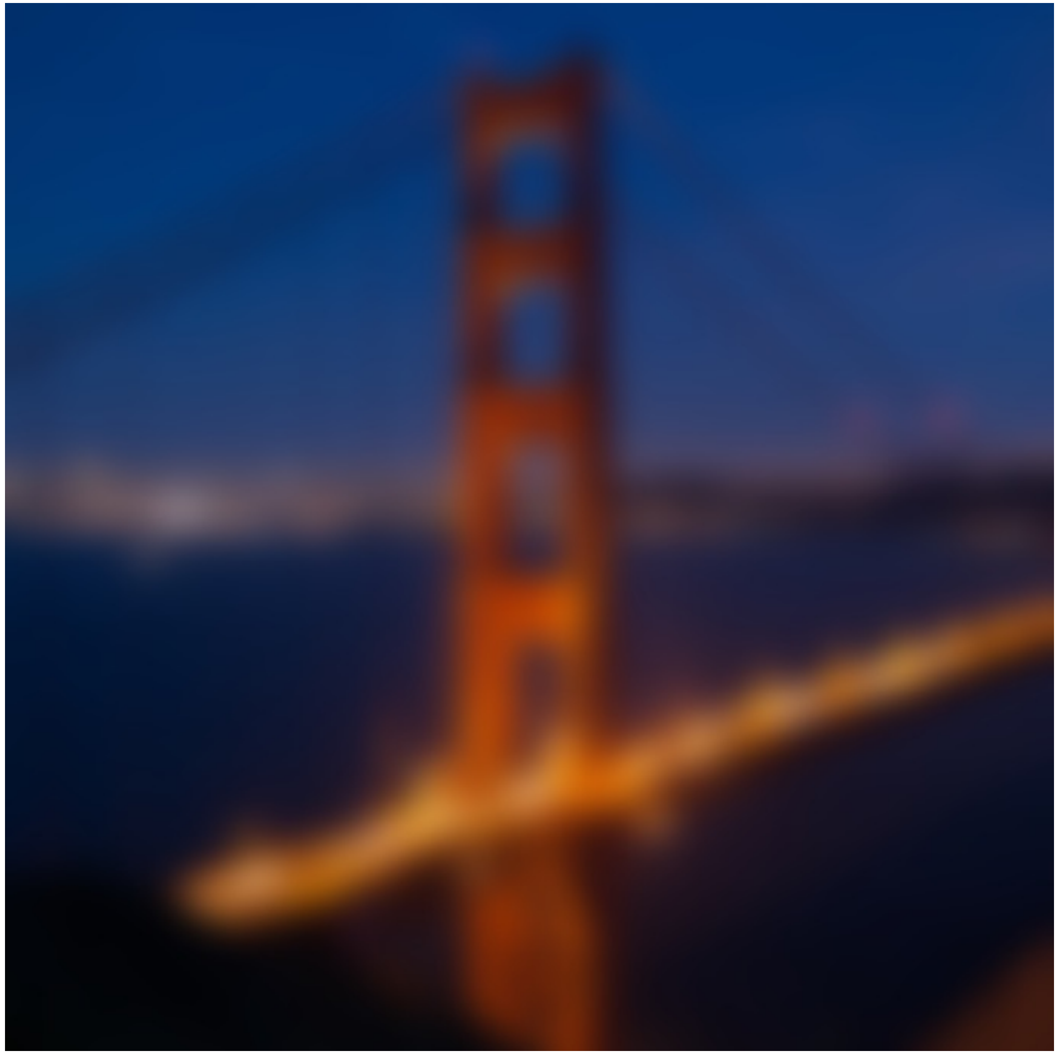


Is this a separable filter? Yes!

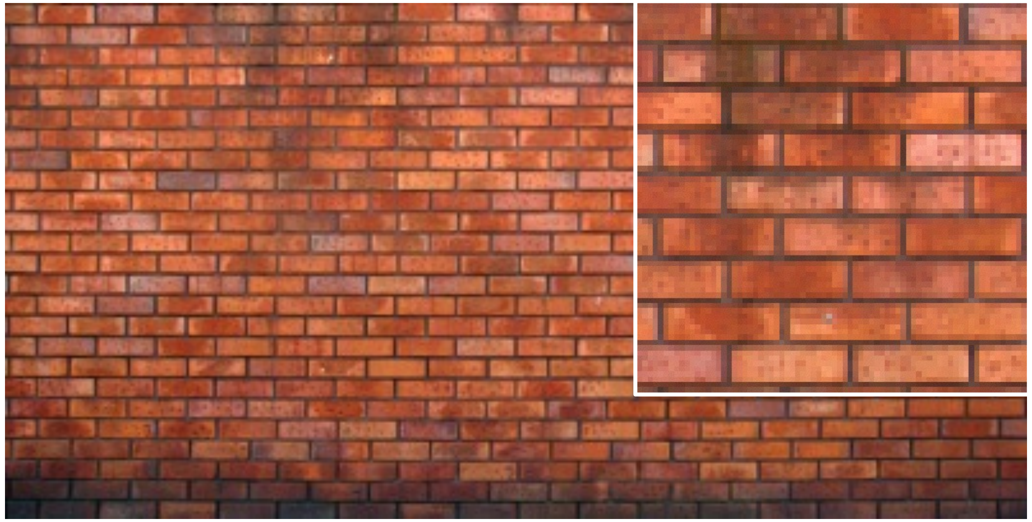
kernel $\frac{1}{16}$

1	2	1
2	4	2
1	2	1

Gaussian filtering example

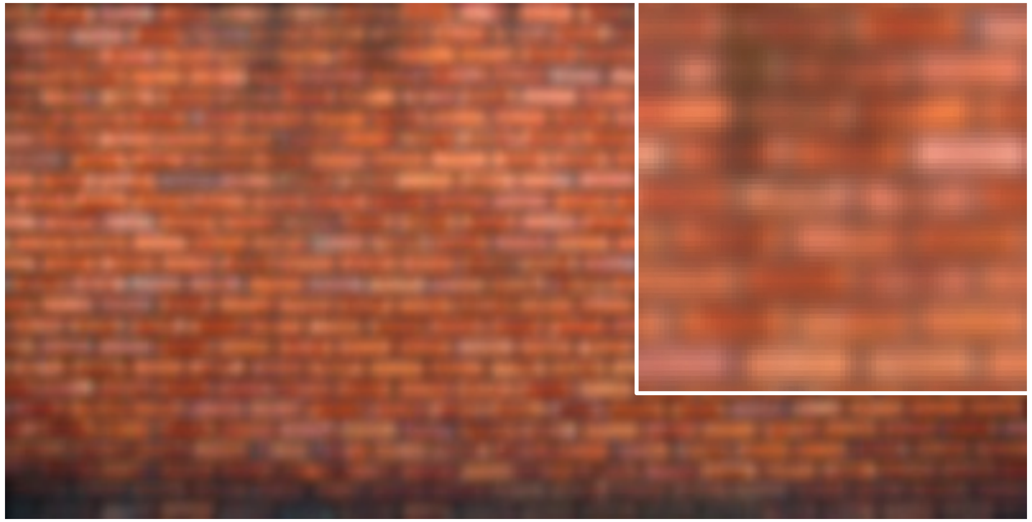


Gaussian vs box filtering

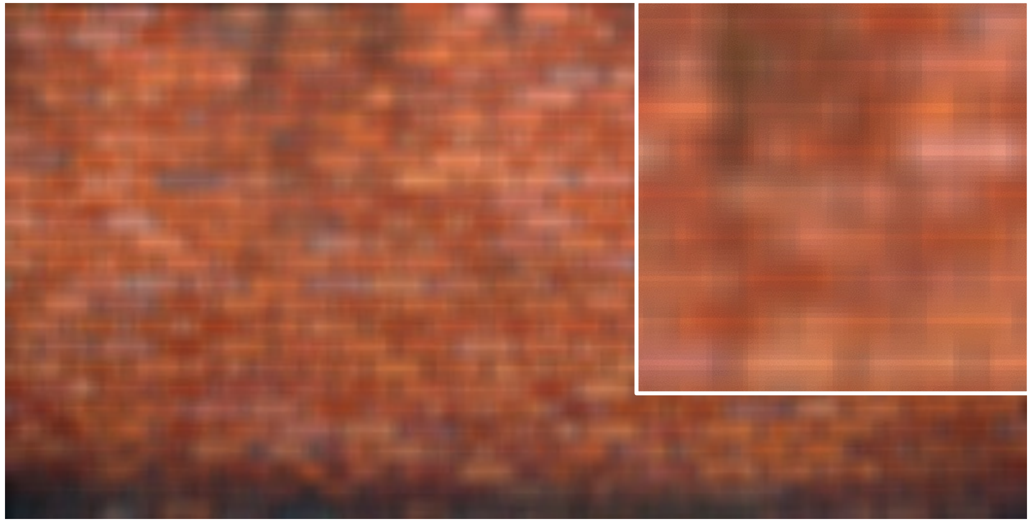


original

Which blur do you like better? Why?



7x7 Gaussian



7x7 box

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

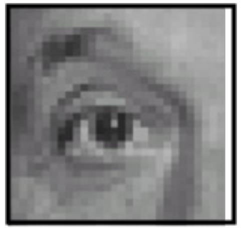
unchanged

input

filter

0	0	0
0	0	1
0	0	0

output



shift to left
by one

Other filters

input

filter

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

output

?

Other filters

input

filter

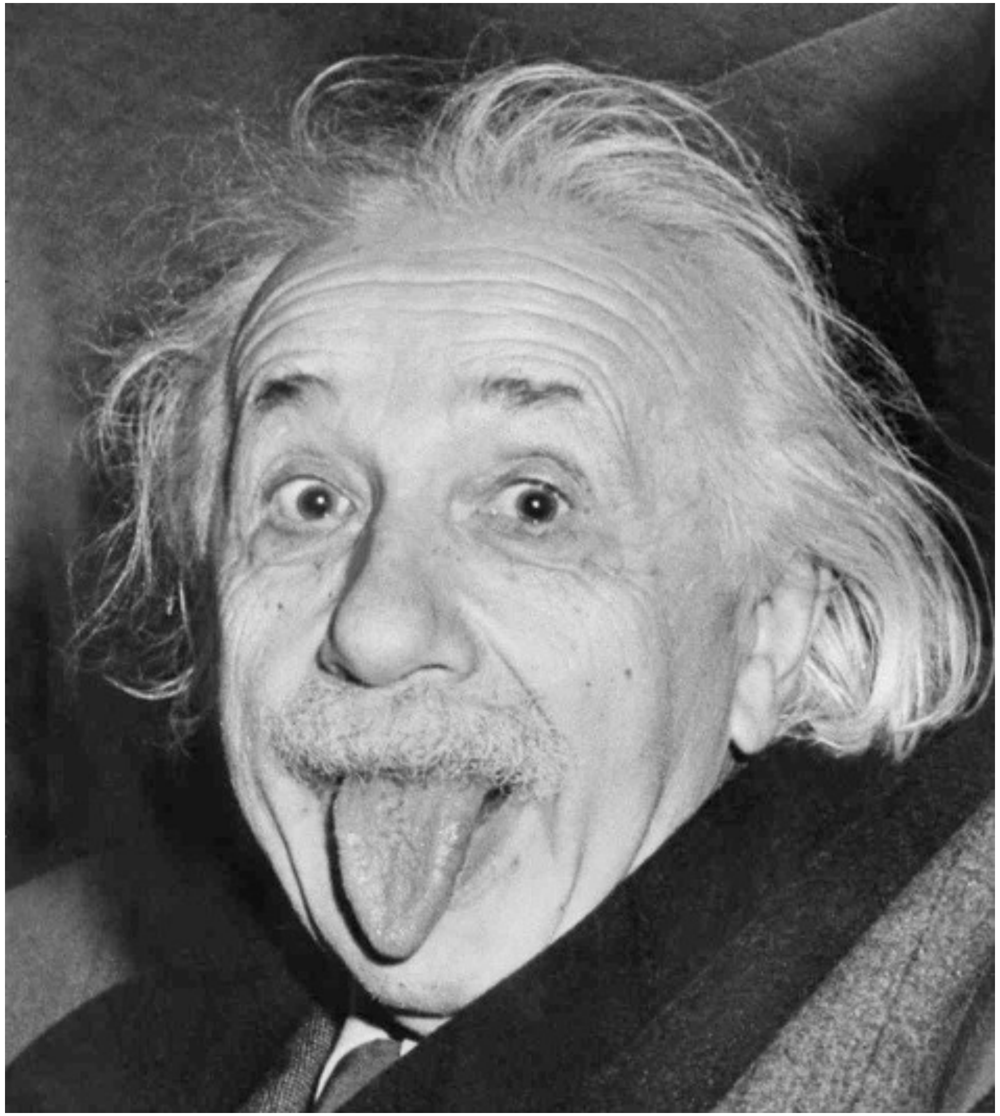
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

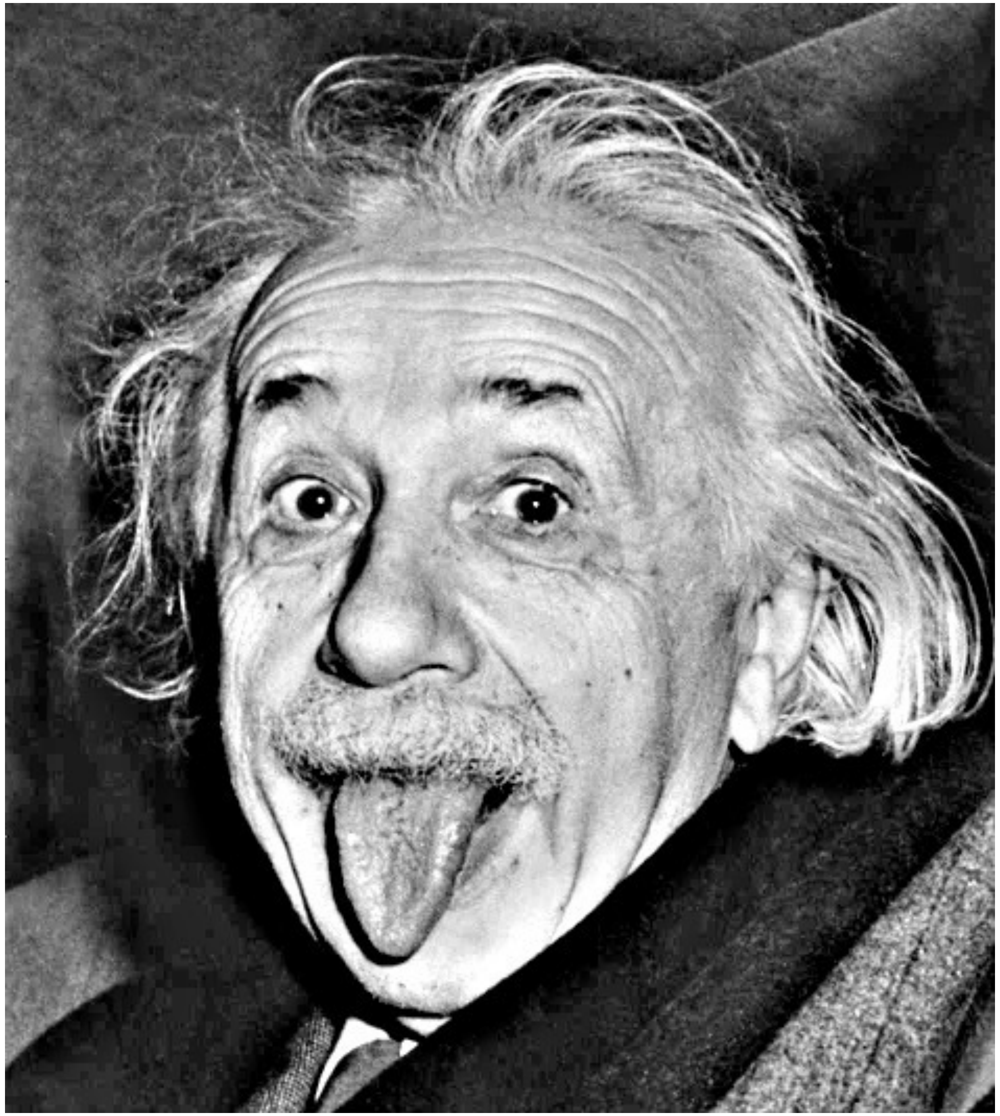
output

sharpening

- do nothing for flat areas
- stress intensity peaks

Sharpening examples





Sharpening examples

Do not overdo it with sharpening


original


sharpened

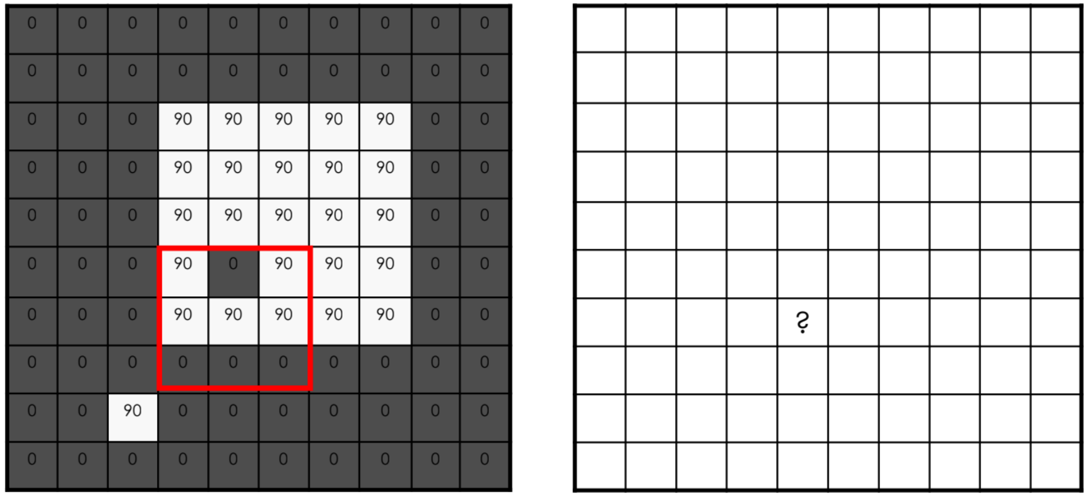

oversharpened

What is wrong in this image?

Not all simple filters are “linear transform”!

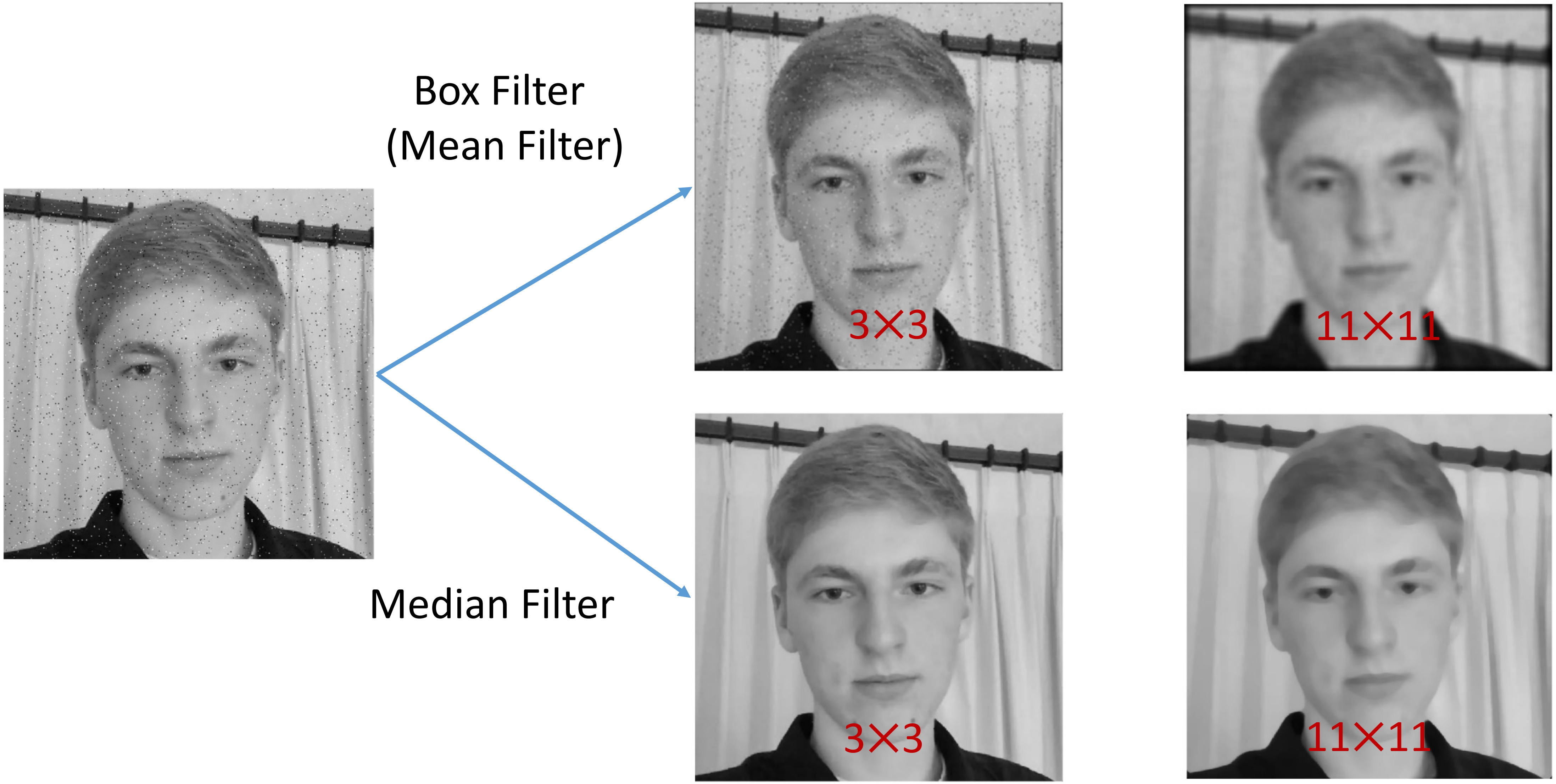
A Simple yet Important Exception: Median Filter

- Operates over a window by selecting the median intensity in the window



- Belong to the class of **“rank” filter** as based on sorting gray levels
  - More example: min, max, range…
  - “Modern name” in deep learning? “Pooling”

Median Filter: When/Why better than Box Filter?



Fourier transform

Fourier transform | inverse Fourier transform

continuous

$$F(k) = \int_{\infty}^{-\infty} f(x) e^{-j2\pi kx} dx$$

$$f(x) = \int_{\infty}^{-\infty} F(k) e^{j2\pi kx} dk$$

discrete

$$F(k) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-j2\pi kx/N}$$

$k = 0, 1, 2, \ldots, N-1$

$$f(x) = \sum_{k=0}^{N-1} F(k) e^{j2\pi kx/N}$$

$x = 0, 1, 2, \ldots, N-1$

'summation of sine waves'

Computing the discrete Fourier transform (DFT)

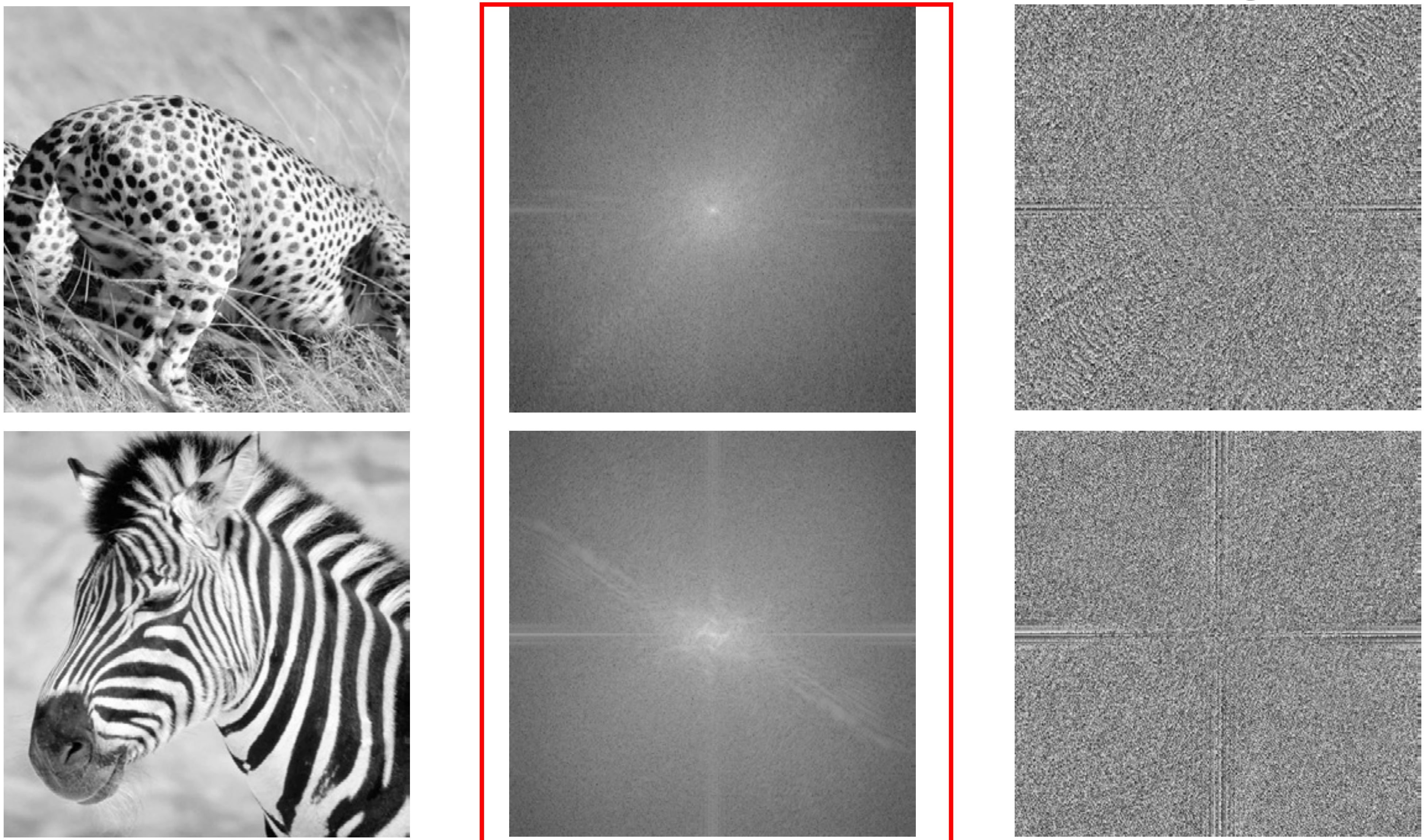
$$F(k) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-j2\pi kx/N}$$ is just a matrix multiplication:

$$\boldsymbol{F} = \boldsymbol{W} \boldsymbol{f}$$

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(3) \\ \vdots \\ F(N-1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & W^3 & \cdots & W^{N-1} \\ W^0 & W^2 & W^4 & W^6 & \cdots & W^{N-2} \\ W^0 & W^3 & W^6 & W^9 & \cdots & W^{N-3} \\ \vdots & & & & \ddots & \vdots \\ W^0 & W^{N-1} & W^{N-2} & W^{N-3} & \cdots & W^1 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \\ \vdots \\ f(N-1) \end{bmatrix} \qquad \boldsymbol{W = e^{-j2\pi/N}}$$

In practice this is implemented using the *fast Fourier transform* (FFT) algorithm.

Fourier transforms of natural images



original amplitude phase

The convolution theorem

The Fourier transform of the convolution of two functions is the product of their Fourier transforms:

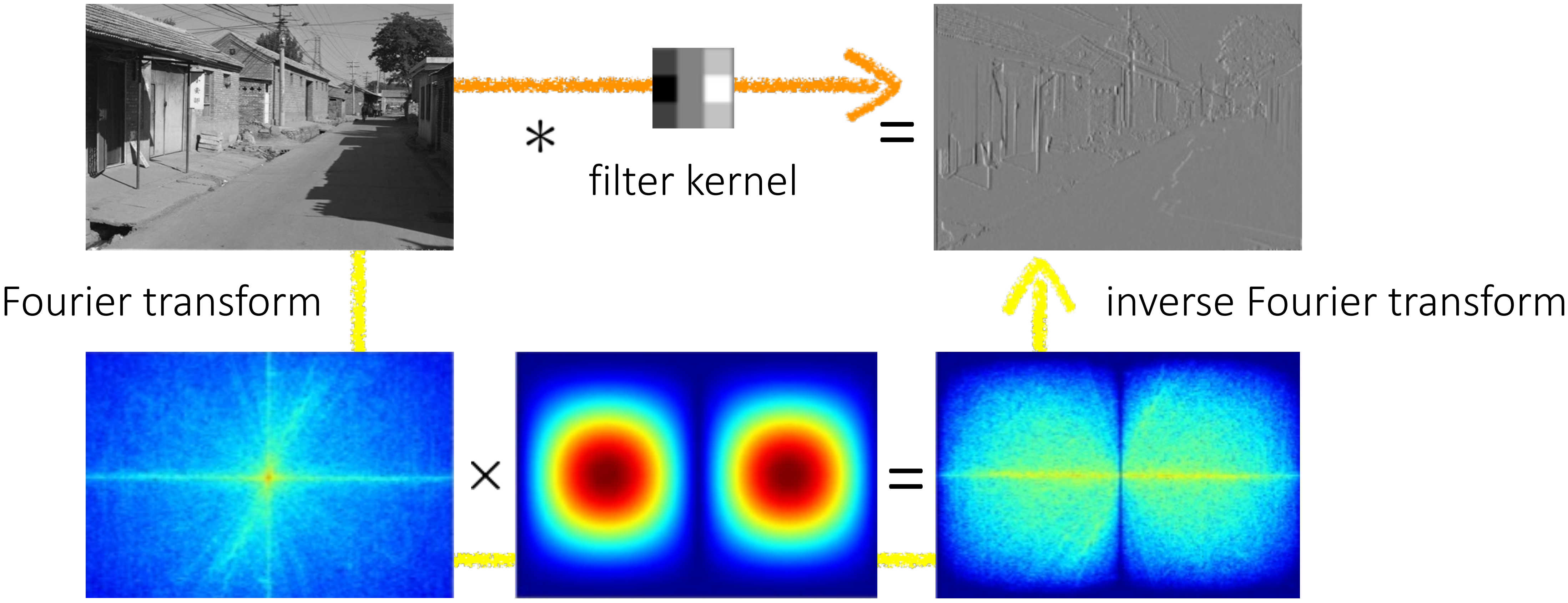
$$\mathcal{F}\{g * h\} = \mathcal{F}\{g\}\mathcal{F}\{h\}$$

The inverse Fourier transform of the product of two Fourier transforms is the convolution of the two inverse Fourier transforms:

$$\mathcal{F}^{-1}\{gh\} = \mathcal{F}^{-1}\{g\} * \mathcal{F}^{-1}\{h\}$$

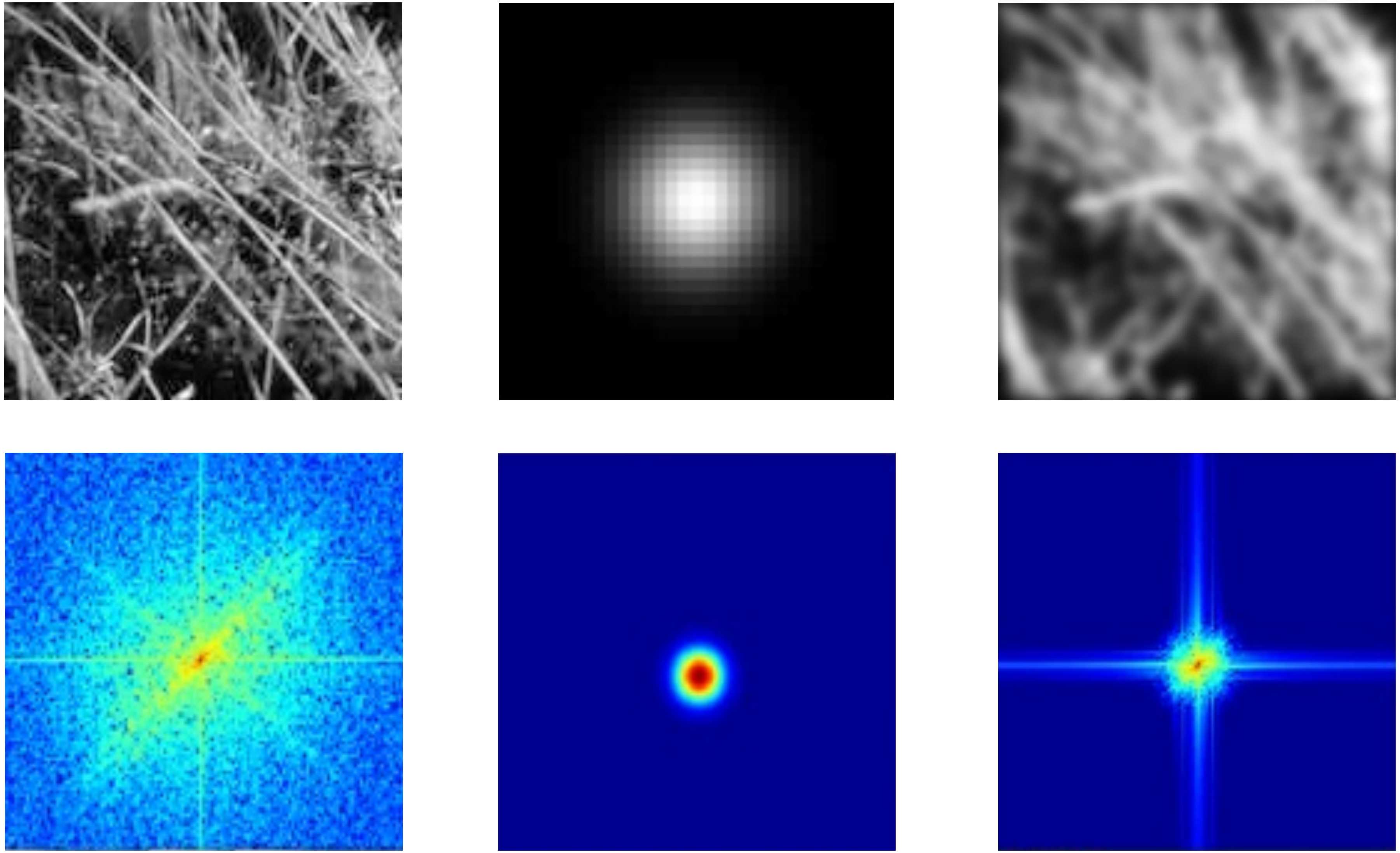
Convolution in spatial domain is equivalent to multiplication in frequency domain!

Spatial domain filtering

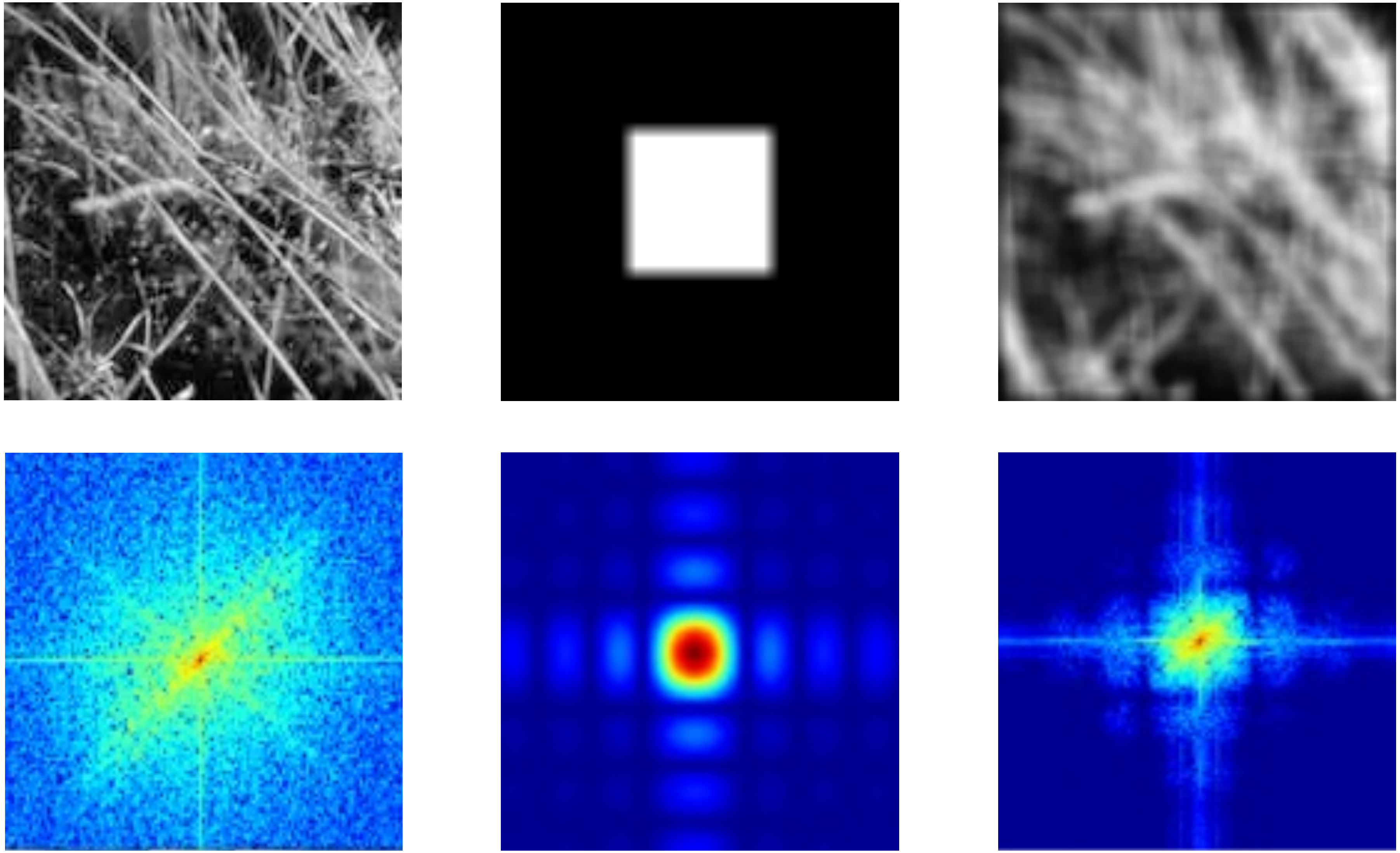


Frequency domain filtering

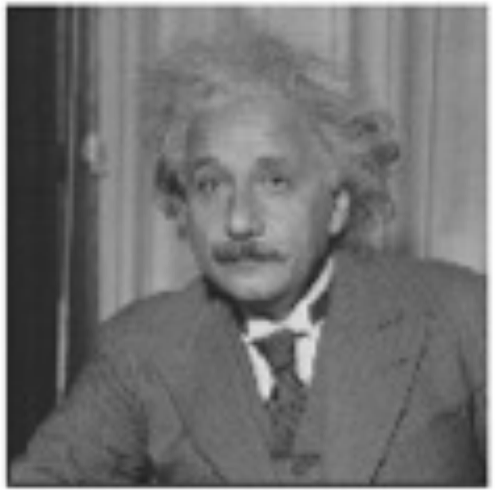
Gaussian blur



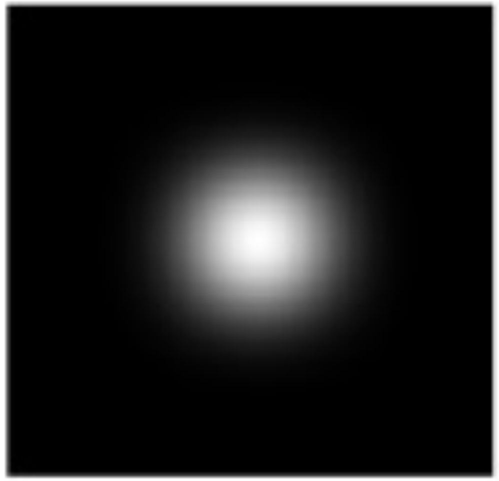
Box blur



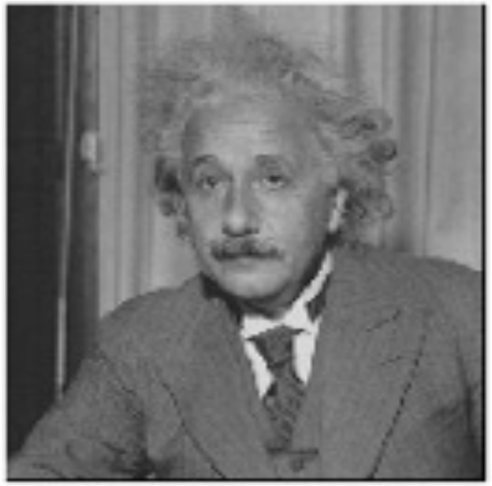
More filtering examples



?



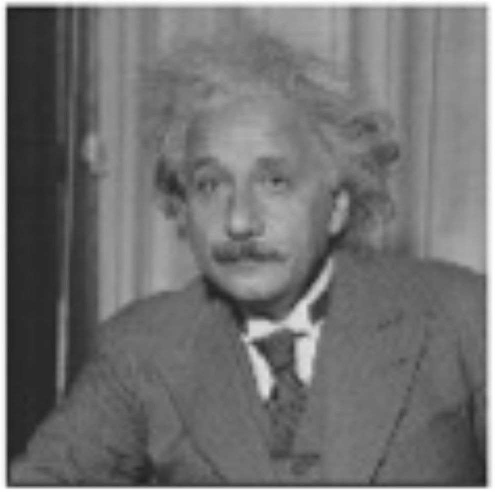
filters shown in frequency-domain

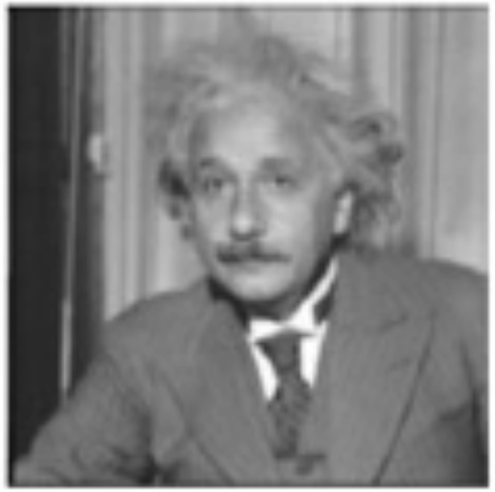


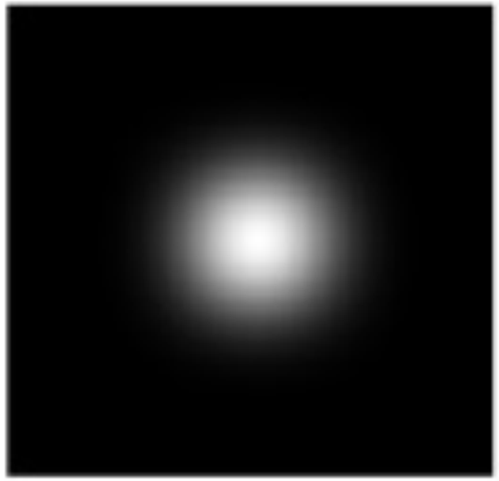
?

More filtering examples

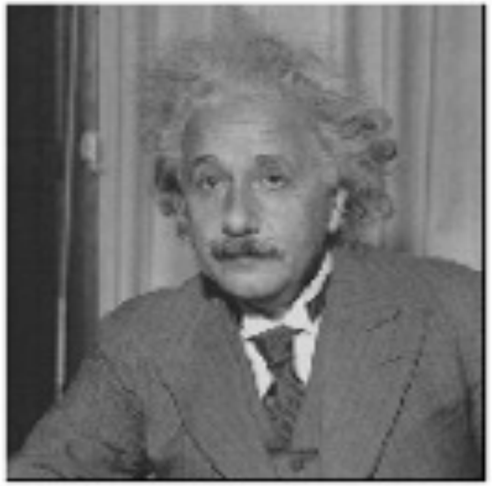
low-pass







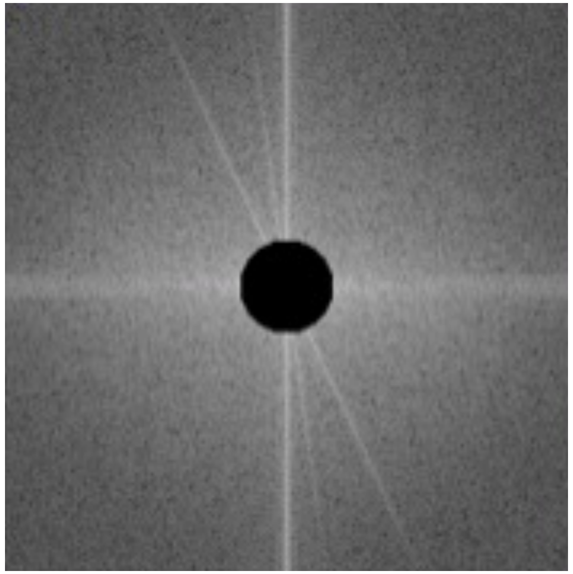
band-pass



filters shown in frequency-domain

More filtering examples

high-pass



The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering