

COMP9517

Computer Vision

2024 Term 3 Week 2

Professor Erik Meijering



UNSW
SYDNEY

0	2	2	2	5	8	11	8	2	2
0	0	2	11	76	136	164	85	11	5
0	2	25	172	181	133	133	164	90	14
2	5	175	130	104	127	141	164	206	65
2	28	212	124	110	204	104	232	133	155
2	73	178	133	121	195	34	31	198	175
2	45	226	141	113	184	53	59	70	192
0	2	70	184	102	116	155	161	175	155
0	0	5	141	121	133	209	215	133	206
0	0	2	73	164	124	121	198	252	147

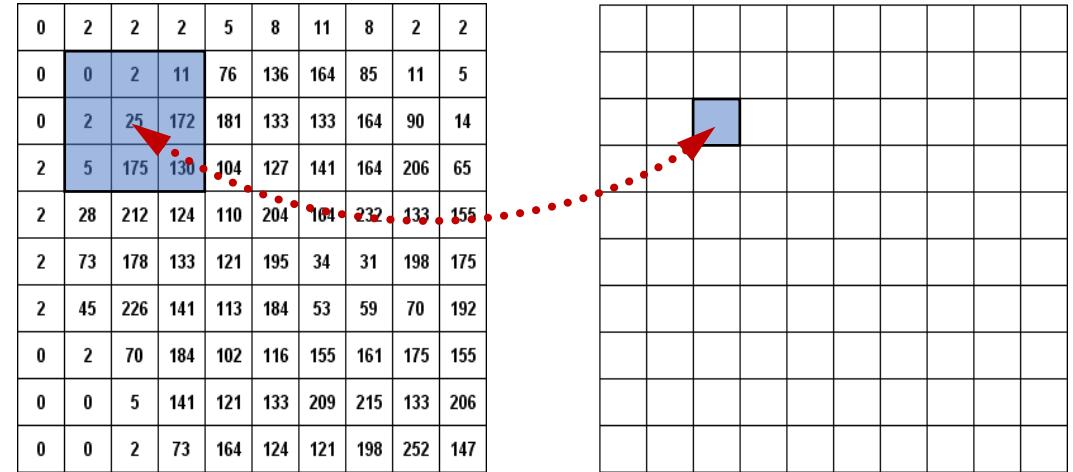


Image Processing

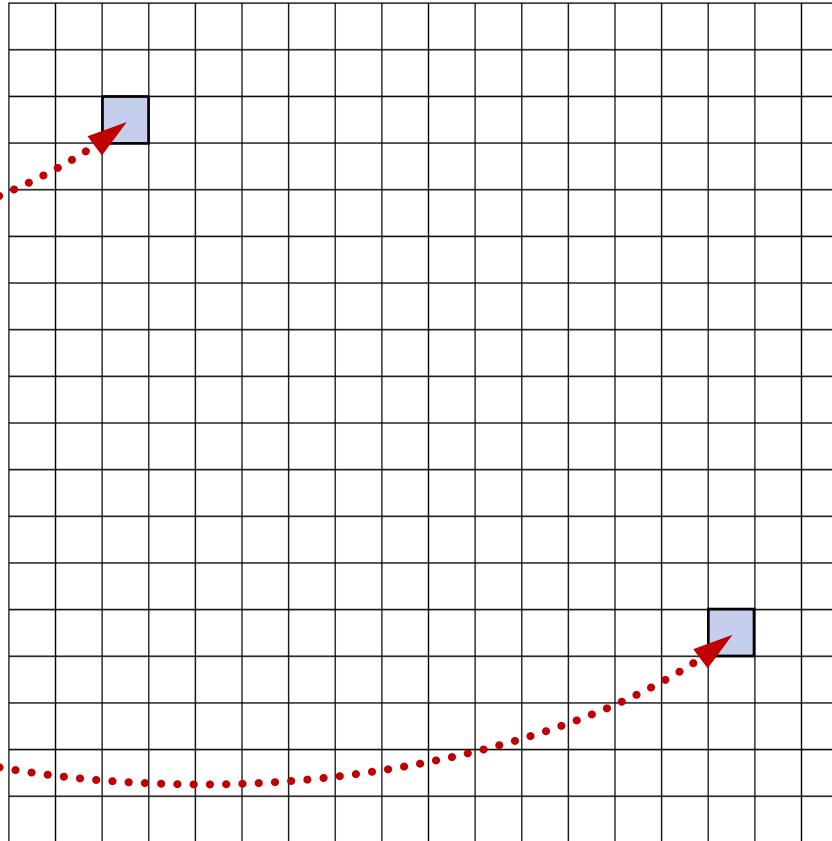
Part 1

Types of image processing (recap)

- Two main types of image processing operations:
 - **Spatial domain operations** (in image space) Next time
 - **Transform domain operations** (mainly in Fourier space)
- Two main types of spatial domain operations:
 - **Point operations** (intensity transformations on individual pixels) Today
 - **Neighbourhood operations** (spatial filtering on groups of pixels)

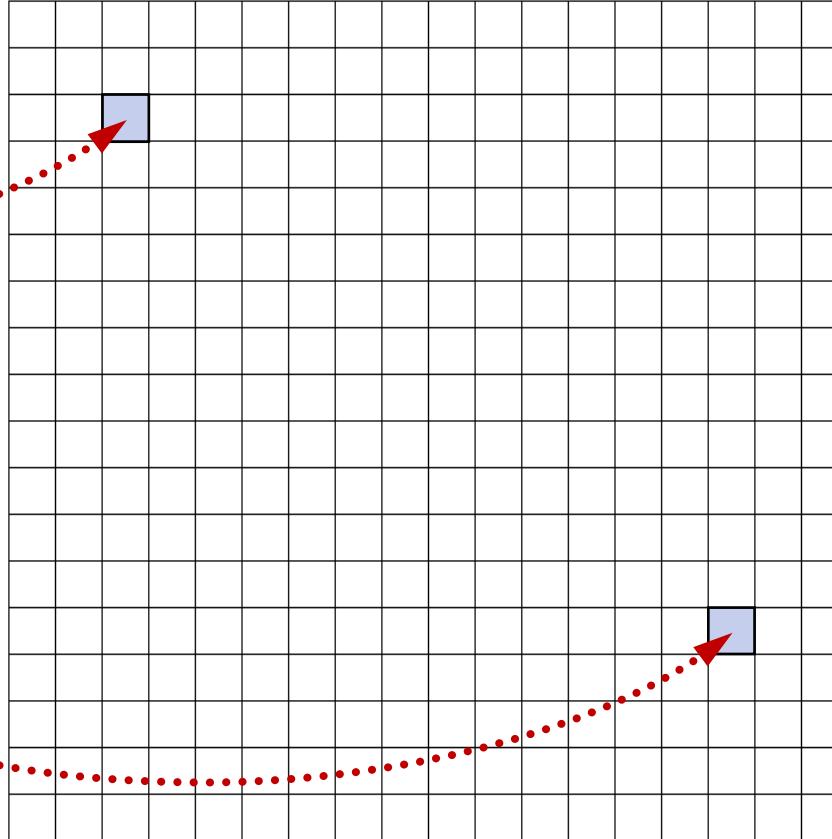
Point operations (recap)

0	2	2	2	5	8	11	8	2	2	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	11	76	136	164	85	11	5	2	2	0	0	0	0	0	0	0	0	0	0
0	2	25	172	181	133	133	164	90	14	5	2	2	0	0	0	0	0	0	0	0	0
2	5	175	130	104	127	141	164	206	65	31	11	2	2	0	0	0	0	0	0	0	0
2	28	212	124	110	204	164	232	133	155	218	87	14	2	2	0	0	0	0	0	0	0
2	73	178	133	121	195	34	31	198	175	204	167	104	14	5	0	0	0	0	0	0	0
2	45	226	141	113	184	53	59	70	192	133	138	167	99	11	2	0	0	0	0	0	0
0	2	70	184	102	116	155	161	175	155	141	184	255	138	34	5	2	0	0	0	0	0
0	0	5	141	121	133	209	215	133	206	124	121	130	153	104	8	2	0	0	0	0	0
0	0	2	73	164	124	121	198	252	147	121	127	119	119	150	19	2	0	0	0	0	0
0	0	0	5	93	150	102	119	130	127	104	121	124	133	153	25	2	0	0	0	0	0
0	0	0	0	5	62	153	155	119	136	198	155	127	124	187	19	2	2	0	0	0	0
0	0	0	0	0	5	138	161	178	155	127	124	141	158	232	5	2	2	0	0	0	0
0	0	0	0	0	0	11	113	164	172	184	102	121	164	79	2	2	0	0	0	0	0
0	0	0	0	0	0	2	5	36	206	187	147	164	153	5	2	2	0	0	0	0	0
0	0	0	0	0	0	0	0	2	5	25	76	31	2	2	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Neighbourhood operations

0	2	2	2	5	8	11	8	2	2	0	0	0	0	0	0	0	0	0
0	0	2	11	76	136	164	85	11	5	2	2	0	0	0	0	0	0	0
0	2	25	172	181	133	133	164	90	14	5	2	2	0	0	0	0	0	0
2	5	175	130	104	127	141	164	206	65	31	11	2	2	0	0	0	0	0
2	28	212	124	110	204	164	232	133	155	218	87	14	2	2	0	0	0	0
2	73	178	133	121	195	34	31	198	175	204	167	104	14	5	0	0	0	0
2	45	226	141	113	184	53	59	70	192	133	138	167	99	11	2	0	0	0
0	2	70	184	102	116	155	161	175	155	141	184	255	138	34	5	2	0	0
0	0	5	141	121	133	209	215	133	206	124	121	130	153	104	8	2	0	0
0	0	2	73	164	124	121	198	252	147	121	127	119	119	150	19	2	0	0
0	0	0	5	93	150	102	119	130	127	104	121	124	133	153	25	2	0	0
0	0	0	0	5	62	153	155	119	136	198	155	127	124	187	19	2	2	0
0	0	0	0	0	5	138	161	178	155	127	124	141	158	232	5	2	2	0
0	0	0	0	0	0	11	113	164	172	184	102	121	164	79	2	2	2	0
0	0	0	0	0	0	2	5	36	206	187	147	164	153	5	2	2	0	0
0	0	0	0	0	0	0	0	2	5	25	76	31	2	2	2	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Topics and learning goals

- Describe the workings of **neighborhood operations**
Convolution, spatial filtering, linear shift-invariant operations, border problem
- Understand the effects of various **filtering methods**
Uniform filter, Gaussian filter, median filter, smoothing, differentiation, separability, pooling
- Combine filtering operations to perform **image enhancement**
Sharpening, unsharp masking, gradient vector & magnitude, edge detection

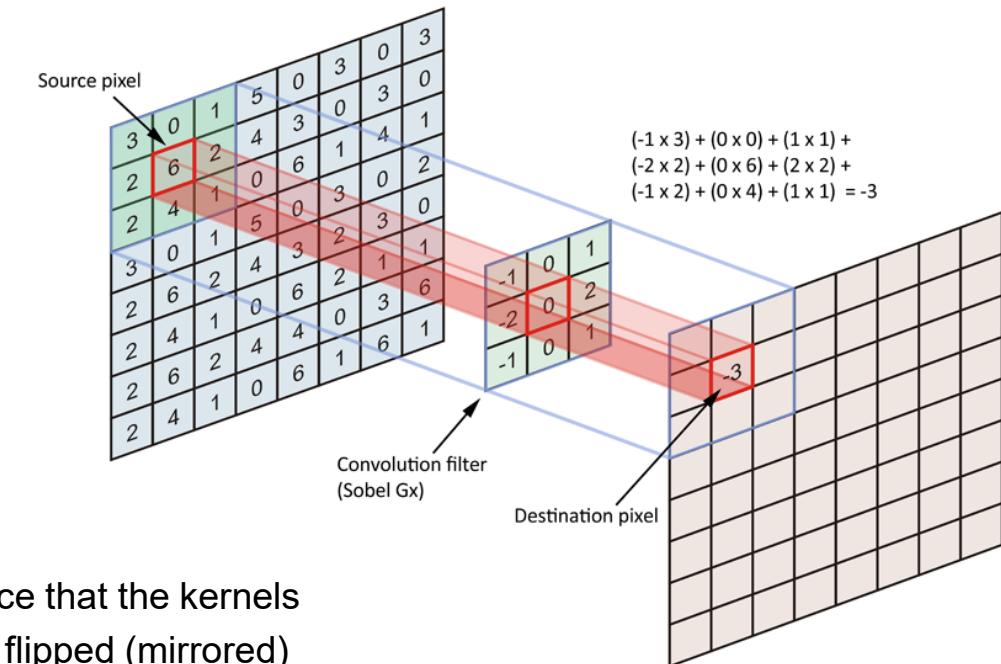
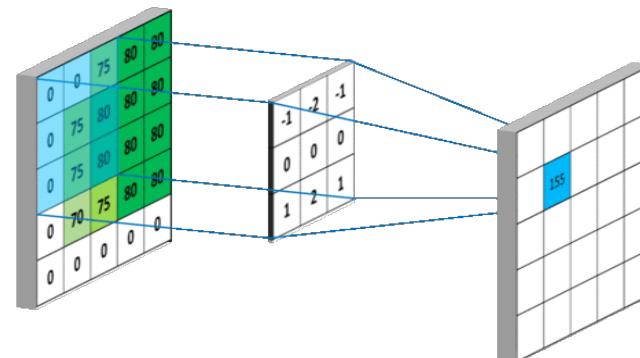
Spatial filtering on groups of pixels

- Use the gray values in a small **neighbourhood** of a pixel in the input image to produce a new gray value for that pixel in the output image
- Also called **filtering** techniques because, depending on the weights applied to the pixel values, they can suppress (filter out) or enhance information
- Neighbourhood of (x, y) is usually a square or rectangular subimage centred at (x, y) and the weights matrix is called the **filter or kernel**
- Typical **kernel sizes** are 3×3 pixels, 5×5 pixels, 7×7 pixels, but can be larger and have different shape (e.g. circular rather than rectangular)

Spatial filtering by convolution

- The output image $o(x, y)$ is computed by **discrete convolution** of the given input image $f(x, y)$ and kernel $h(x, y)$:

$$o(x, y) = \sum_{i=-n}^n \sum_{j=-m}^m f(x - i, y - j)h(i, j)$$



Notice that the kernels
are flipped (mirrored)

Equivalent notations of convolution

$$o(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f(x - i, y - j)h(i, j)$$

Image coordinates are flipped
Out-of-bound values of f and h are 0

$$o(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(x + k, y + l)h(-k, -l)$$

Kernel coordinates are flipped
Substitution of variables:
 $k = -i$ and $l = -j$

$$o(x, y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k, l)h(x - k, y - l)$$

Kernel is shifted and flipped
Substitution of variables:
 $k = x - i$ and $l = y - j$

Why must the kernel be flipped?

Because by definition the kernel is the impulse response of the convolution

Define input f to be an impulse image: $f(0,0) = 1$ and $f(x,y) = 0$ otherwise

Convolution:
$$o(x,y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(x+k, y+l)h(-k, -l) = h(x, y)$$

Output is the kernel

Correlation:
$$o(x,y) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(x+k, y+l)h(k, l) = h(-x, -y)$$

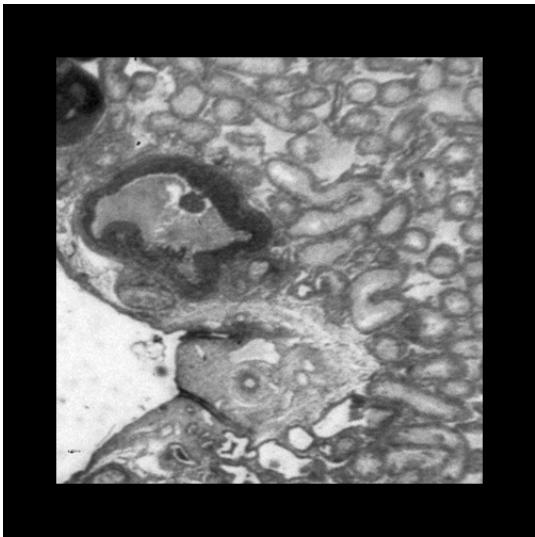
Output is the flipped kernel

Fixing the border problem

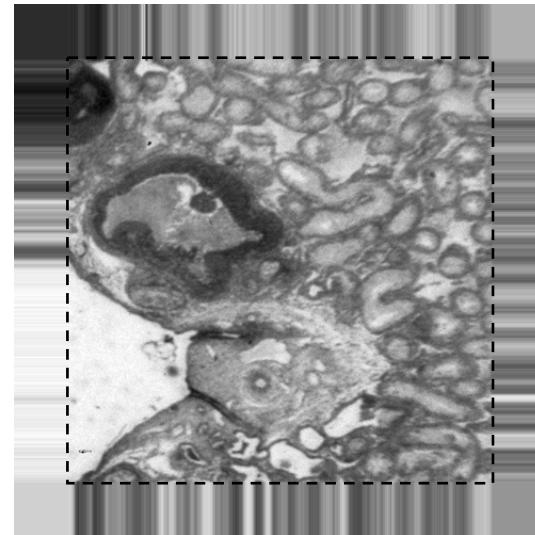
- Expand the image outside the original border using:
 - **Padding:** Set all additional pixels to a constant (zero) value
Hard transitions yield border artifacts (requires windowing)
 - **Clamping:** Repeat all border pixel values indefinitely
Better border behaviour but arbitrary (no theoretical foundation)
 - **Wrapping:** Copy pixel values from opposite sides
Implicitly used in the (fast) Fourier transform
 - **Mirroring:** Reflect pixel values across borders
Smooth, symmetric, periodic, no boundary artifacts

Fixing the border problem

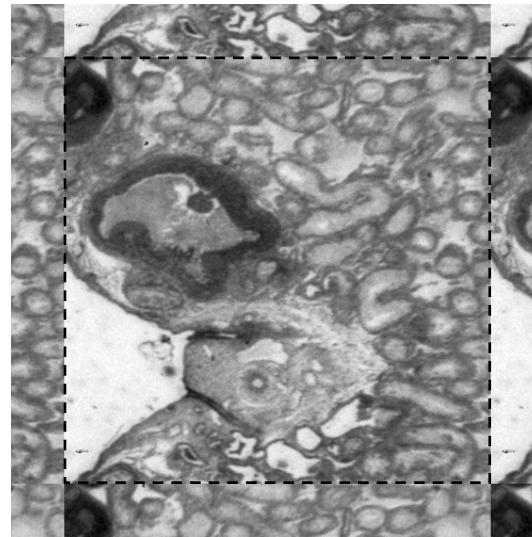
Padding



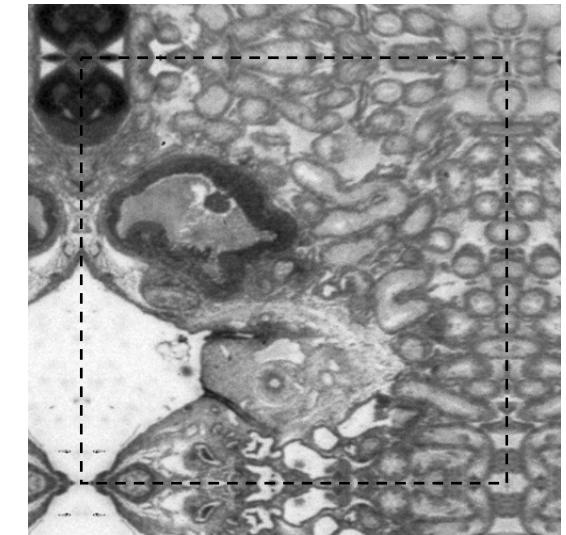
Clamping



Wrapping



Mirroring



Spatial filtering by convolution

- Convolution is a **linear, shift-invariant operation**
- **Linearity:** If input $f_1(x, y)$ yields output $g_1(x, y)$ and $f_2(x, y)$ yields $g_2(x, y)$, then a linear combination of inputs $a_1f_1(x, y) + a_2f_2(x, y)$ yields the same combination of outputs $a_1g_1(x, y) + a_2g_2(x, y)$, for any constants a_1, a_2
- **Shift invariance:** If input $f(x, y)$ yields output $g(x, y)$, then the shifted input $f(x - \Delta x, y - \Delta y)$ yields the shifted output $g(x - \Delta x, y - \Delta y)$, in other words, the operation does not discriminate between spatial positions

Properties of convolution

For any set of images (functions) f_i the convolution operation $*$ satisfies:

- **Commutativity:** $f_1 * f_2 = f_2 * f_1$
- **Associativity:** $f_1 * (f_2 * f_3) = (f_1 * f_2) * f_3$
- **Distributivity:** $f_1 * (f_2 + f_3) = f_1 * f_2 + f_1 * f_3$
- **Multiplicativity:** $a \cdot (f_1 * f_2) = (a \cdot f_1) * f_2 = f_1 * (a \cdot f_2)$
- **Derivation:** $(f_1 * f_2)' = f_1' * f_2 = f_1 * f_2'$
- **Theorem:** $f_1 * f_2 \leftrightarrow \hat{f}_1 \cdot \hat{f}_2$ convolution in spatial domain amounts to multiplication in spectral domain... (next lecture)

Simplest smoothing filter

- Calculates the **mean pixel value** in a neighbourhood N with $|N|$ pixels

$$g(x, y) = \frac{1}{|N|} \sum \sum_{(i,j) \in N} f(x + i, y + j)$$

- Often used for **image blurring** and **noise reduction**
- Reduces fluctuations due to disturbances in image acquisition
- Neighbourhood averaging also **blurs the object edges** in the image
- Can use **weighted averaging** to give more importance to some pixels

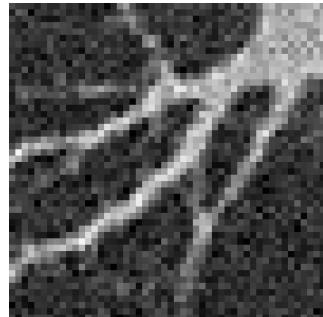
Simplest smoothing filter

- Also called **uniform filter** as it implicitly uses a uniform kernel

3×3

$$u_3 = \frac{1}{9} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & \textcolor{blue}{1} & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

f

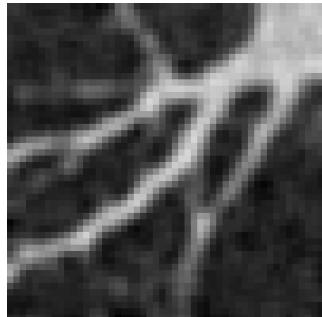


5×5

$$u_5 = \frac{1}{25} \cdot$$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

$f * u_3$

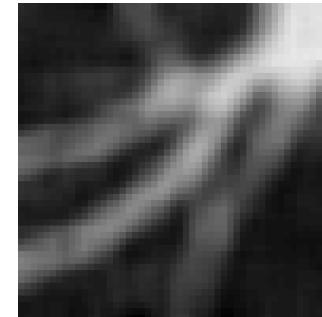


7×7

$$u_7 = \frac{1}{49} \cdot$$

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

$f * u_7$



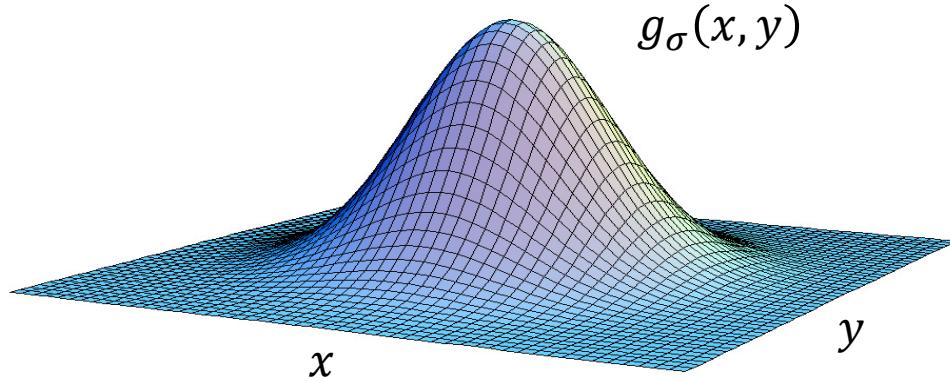
... and so forth

... and so forth

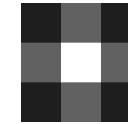
Gaussian filter

- The Gaussian filter is one of the most important basic image filters

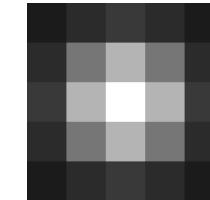
$$g_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



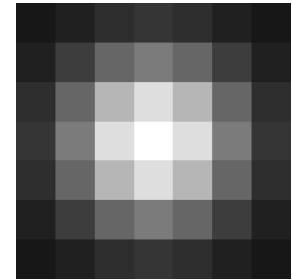
3×3
 $\sigma = 0.5$



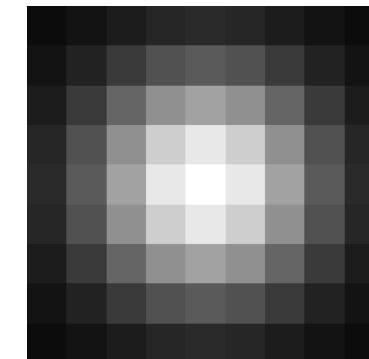
5×5
 $\sigma = 1.0$



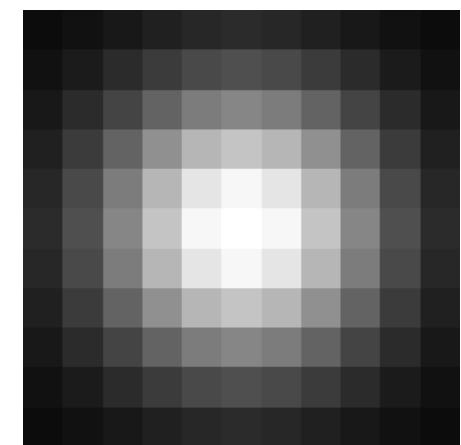
7×7
 $\sigma = 1.5$



9×9
 $\sigma = 2.0$



11×11
 $\sigma = 2.5$



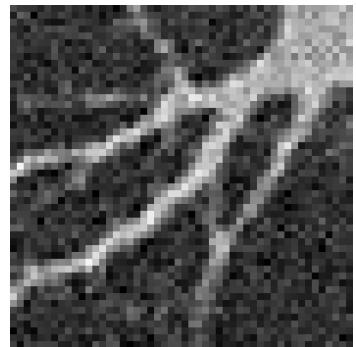
Gaussian filter

Many nice properties motivate the use of the Gaussian filter:

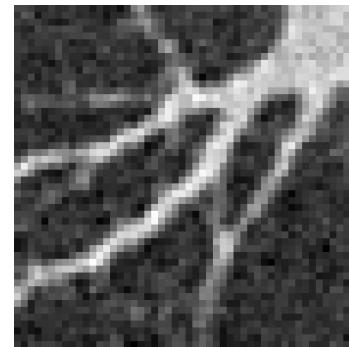
- It is the only filter that is both separable and circularly symmetric
- It has optimal joint localization in spatial and frequency domain
- The Fourier transform of a Gaussian is also a Gaussian function
- The n-fold convolution of any low-pass filter converges to a Gaussian
- It is infinitely smooth so it can be differentiated to any desired degree
- It scales naturally (σ) and allows for consistent scale-space theory

Gaussian filtering examples

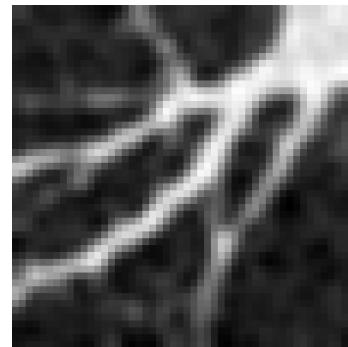
Input



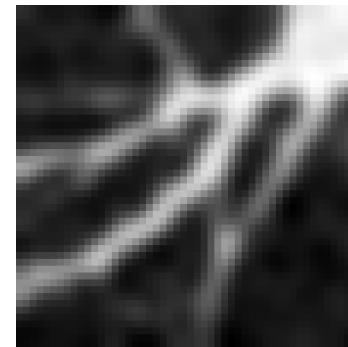
Gaussian smoothed...



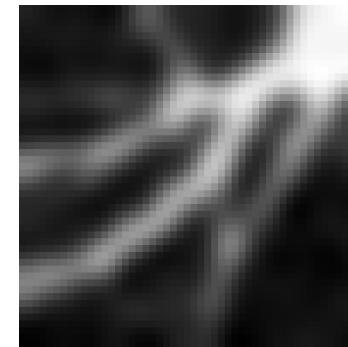
$\sigma = 0.5$



$\sigma = 1.0$



$\sigma = 1.5$



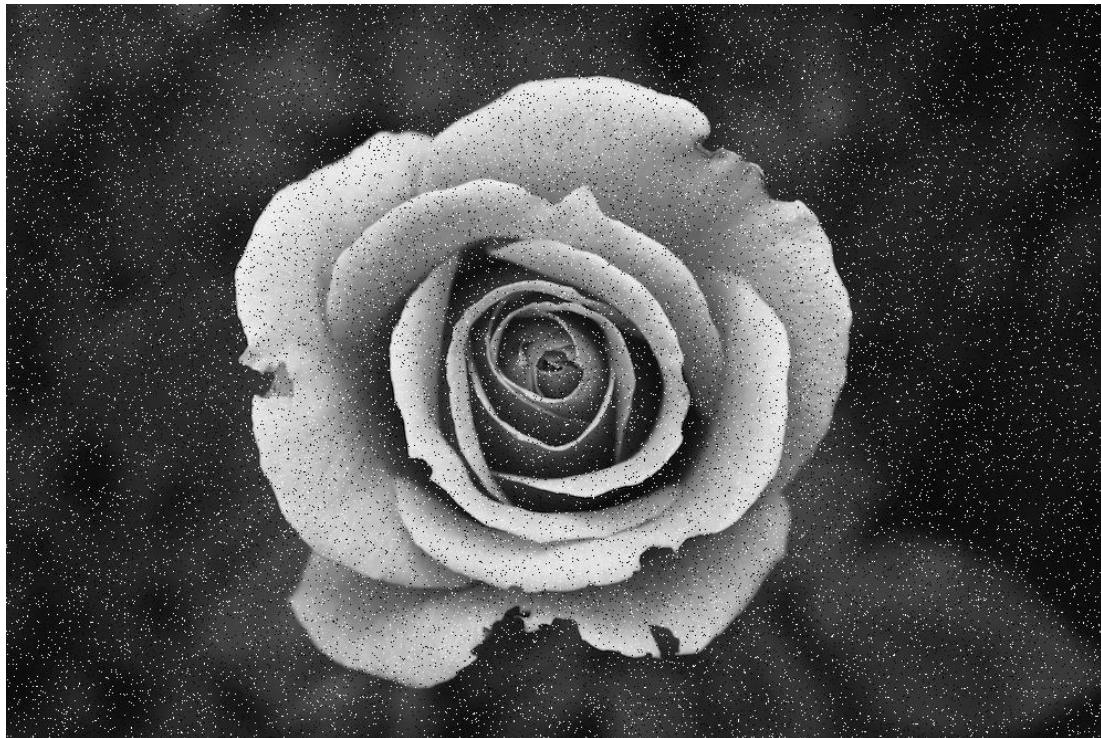
$\sigma = 2.0$



$\sigma = 2.5$

Gaussian filtering examples

Input



Gaussian smoothed



Median filter

- Is an **order-statistics filter** (based on ordering and ranking pixel values)
- Calculates the **median pixel value** in a neighbourhood N with $|N|$ pixels
- The median value m of a set of ordered values is the **middle value**
- At most half the values in the set are $< m$ and the other half $> m$

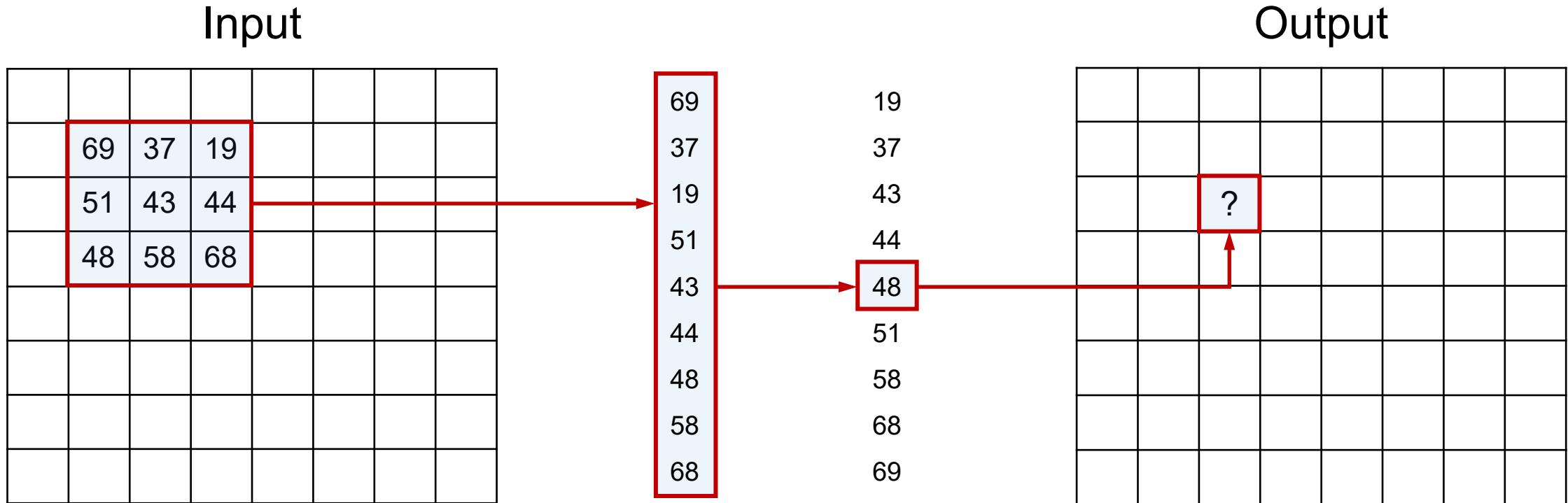
Set: {115, 10, 25, 12, 221, 46, 91, 178, 193}

Ordered: {10, 12, 25, 46, 91, 115, 178, 193, 221}

Median

In the case of an even number of values,
often the median is taken to be the arithmetic
mean of the two middle values

Median filter



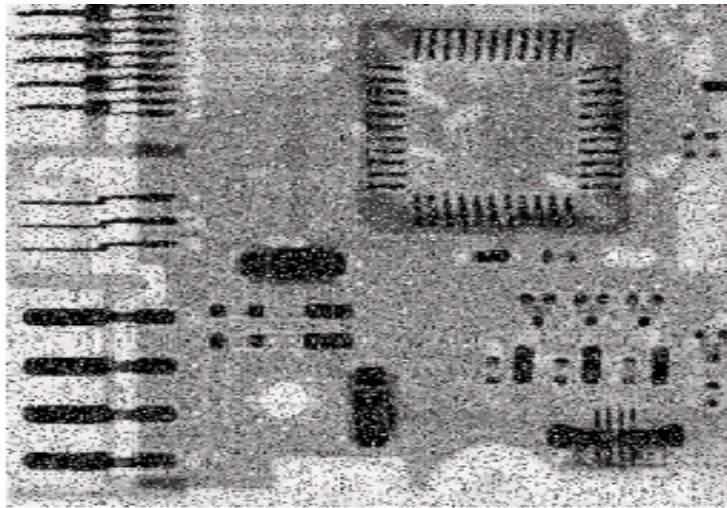
Taking the minimum or maximum instead of the median is called **min-filtering** and **max-filtering** respectively

Median filter

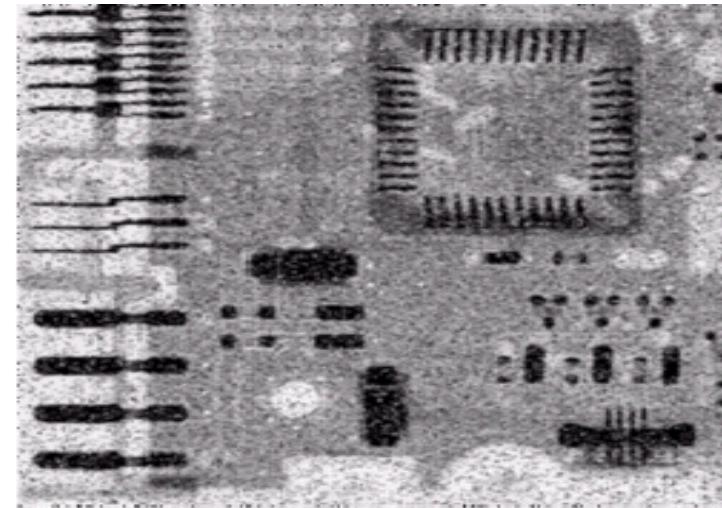
- Forces pixels with distinct intensities to be **more like their neighbours**
- It **eliminates isolated intensity spikes** (salt and pepper image noise)
- Neighbourhood is **typically of size $n \times n$ pixels** with $n = 3, 5, 7, \dots$
- This also **eliminates pixel clusters** (light or dark) with area $< n^2/2$
- Is not a convolution filter but an example of a **nonlinear filter**

Median filtering example

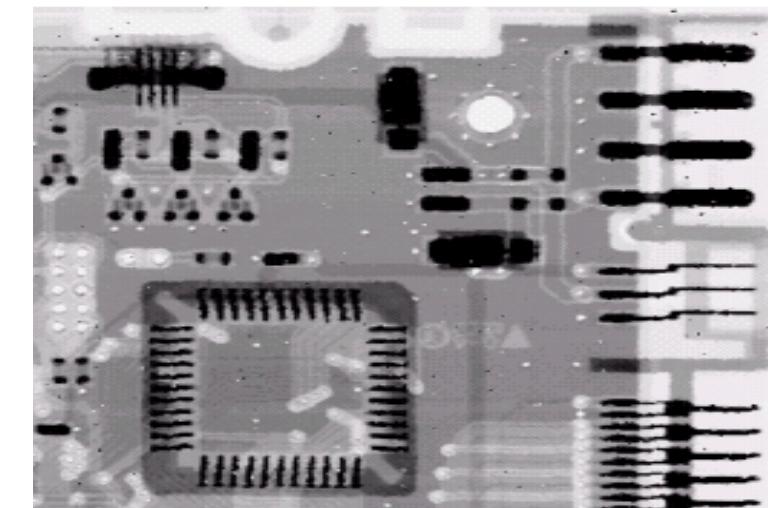
Input



3 x 3 mean filtered

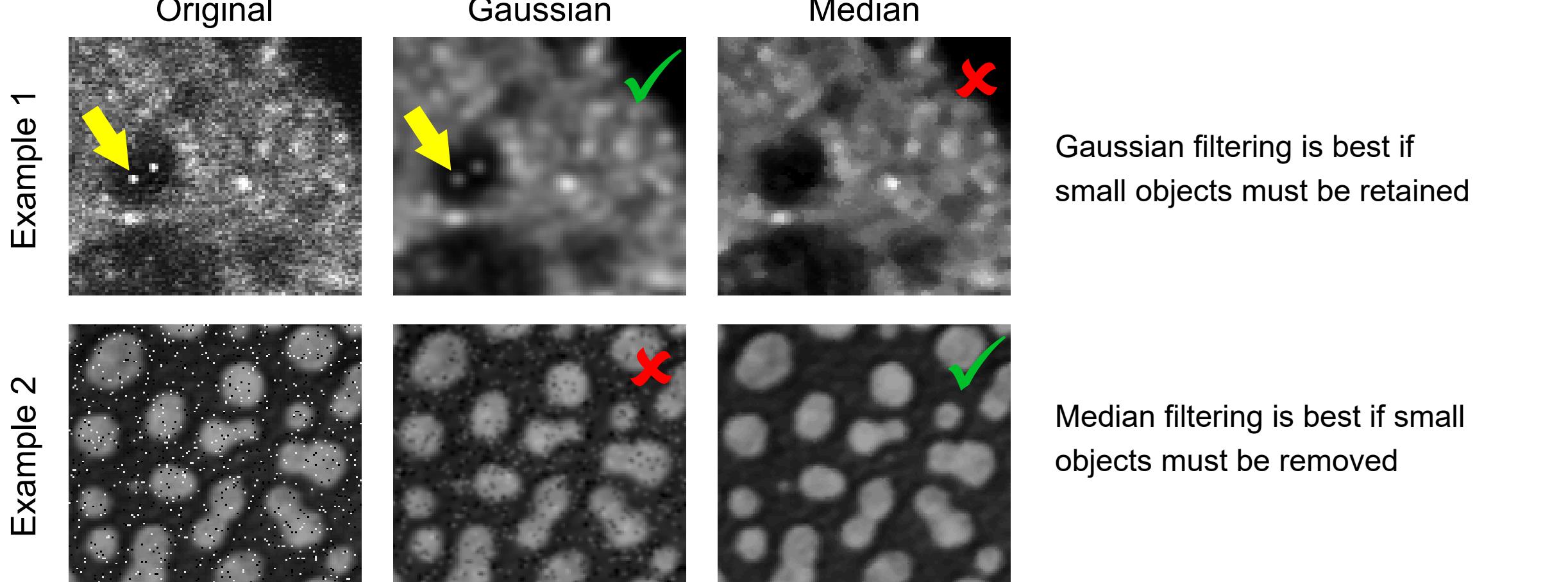


3 x 3 median filtered

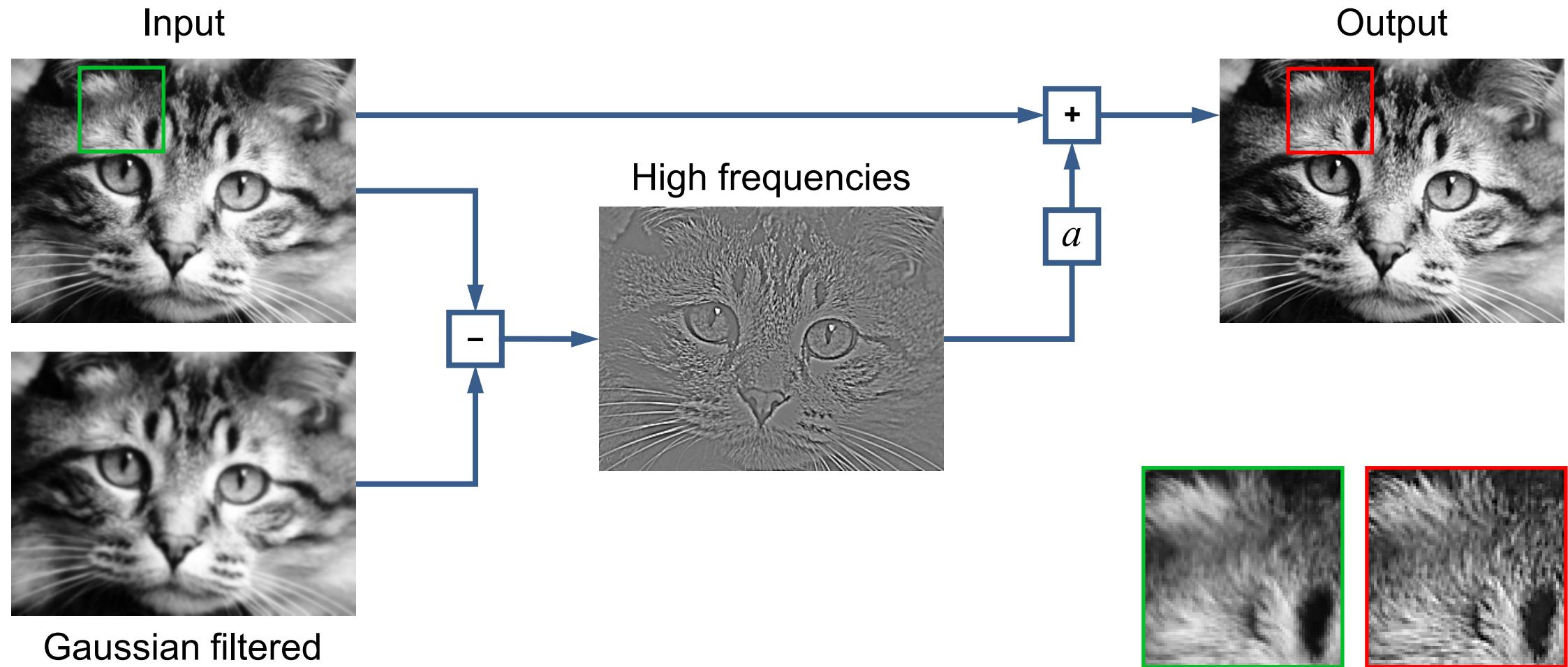


Noise pixels are completely removed rather than averaged out

Gaussian versus median filtering

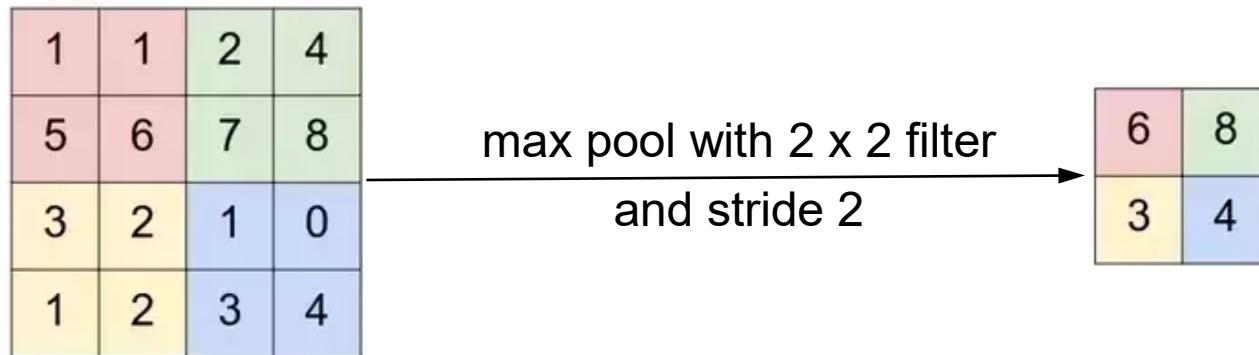


Sharpening by unsharp masking



Pooling

- Combines filtering and downsampling in one operation
- Examples include max / min / median / average pooling
- Makes the image smaller and reduces computations
- Popular in deep convolutional neural networks



Derivative filters

- Spatial derivatives respond to intensity changes (such as object edges)
- In digital images they are approximated using finite differences
- Different possible ways to take finite differences

Forward difference

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

Kernel:

1	-1
---	----

Backward difference

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

1	-1
---	----

Central difference

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

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

Reminder: kernels are flipped in the convolution process

Derivative filters

- Second-order spatial derivative using finite differences

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{\partial f}{\partial x}(x) - \frac{\partial f}{\partial x}(x-1) = [f(x+1) - f(x)] - [f(x) - f(x-1)] = f(x+1) - 2f(x) + f(x-1)$$

Backward difference

Forward differences

1	-2	1
---	----	---

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{\partial f}{\partial x}\left(x + \frac{1}{2}\right) - \frac{\partial f}{\partial x}\left(x - \frac{1}{2}\right) = [f(x+1) - f(x)] - [f(x) - f(x-1)] = f(x+1) - 2f(x) + f(x-1)$$

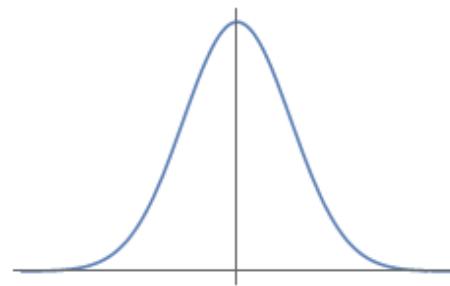
Central difference 1/2

Central differences 1/2

1	-2	1
---	----	---

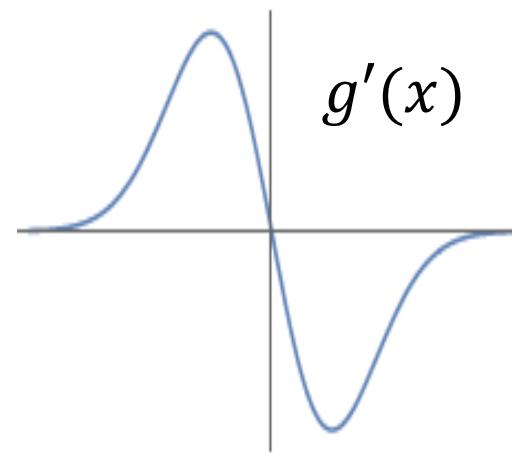
Derivative filters

- Sampled approximations of the continuous Gaussian derivatives



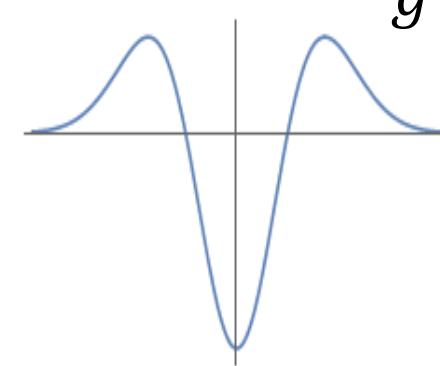
$g(x)$

1	2	1
---	---	---



$g'(x)$

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



$g''(x)$

1	-2	1
---	----	---

Similarly in y

1	
2	
1	

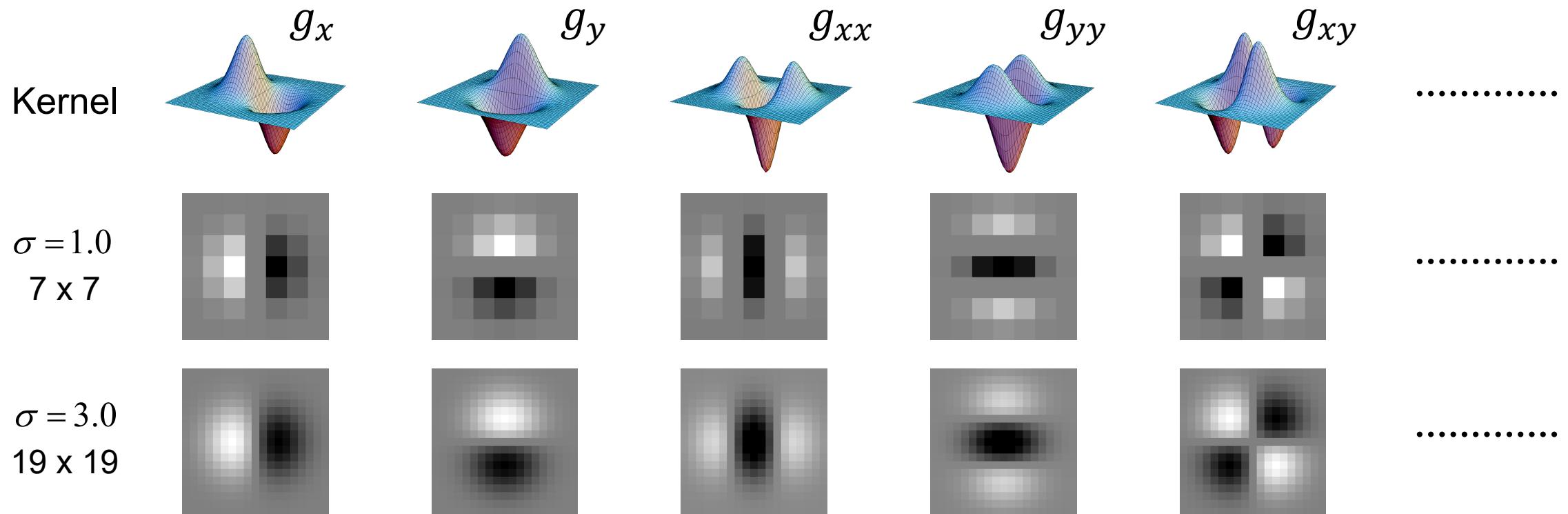
1	
0	
-1	

1	
-2	
1	

Gaussian derivative filters

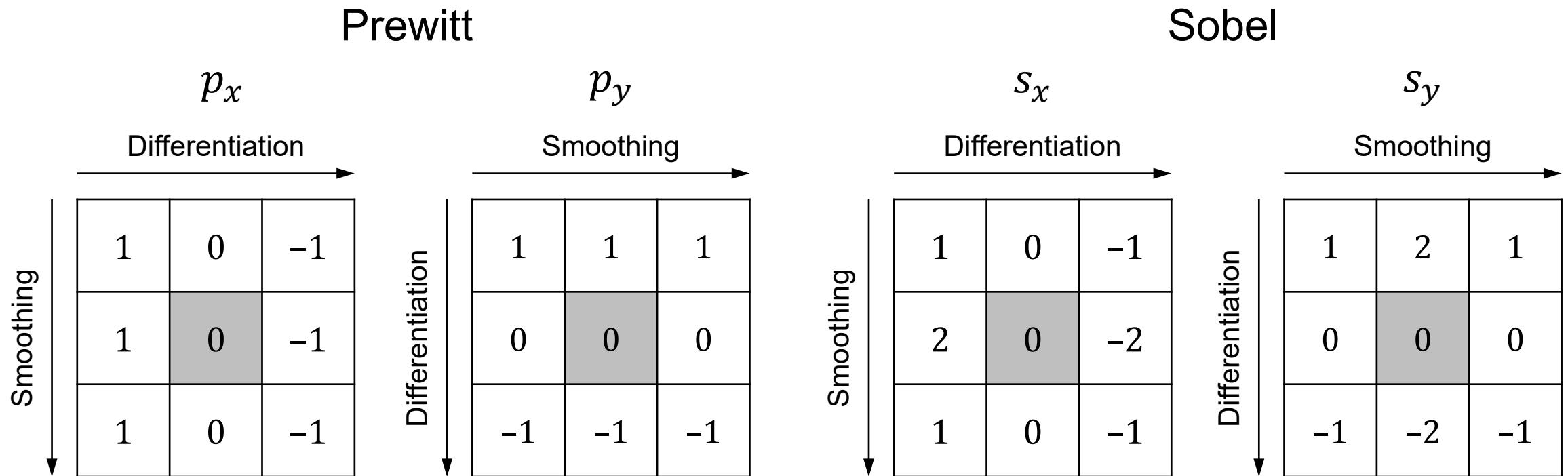
$$\frac{\partial g}{\partial x} \equiv g_x$$

- Extension of Gaussian filter kernels to 2D and different spatial scales



Prewitt and Sobel kernels

- Differentiation in one dimension and smoothing in the other dimension



Separable filter kernels

Uniform: $u = \frac{1}{9} \cdot$

1	1	1
1	1	1
1	1	1

$$= \frac{1}{3} \cdot \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \frac{1}{3} \cdot$$

1
1
1

Smoothing in x

Smoothing in y

Prewitt: $p_x =$

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

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

1
1
1

First derivative in x

Smoothing in y

Sobel: $s_y =$

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

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

1
0
-1

Smoothing in x

First derivative in y

Gauss: $g = \frac{1}{16} \cdot$

1	2	1
2	4	2
1	2	1

$$= \frac{1}{4} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \frac{1}{4} \cdot$$

1
2
1

Smoothing in x

Smoothing in y

Separable filter kernels

- Allow for a much more computationally efficient implementation

$$g(x, y) = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

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

Can be rewritten as:

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

$$g(x) = \frac{1}{4} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

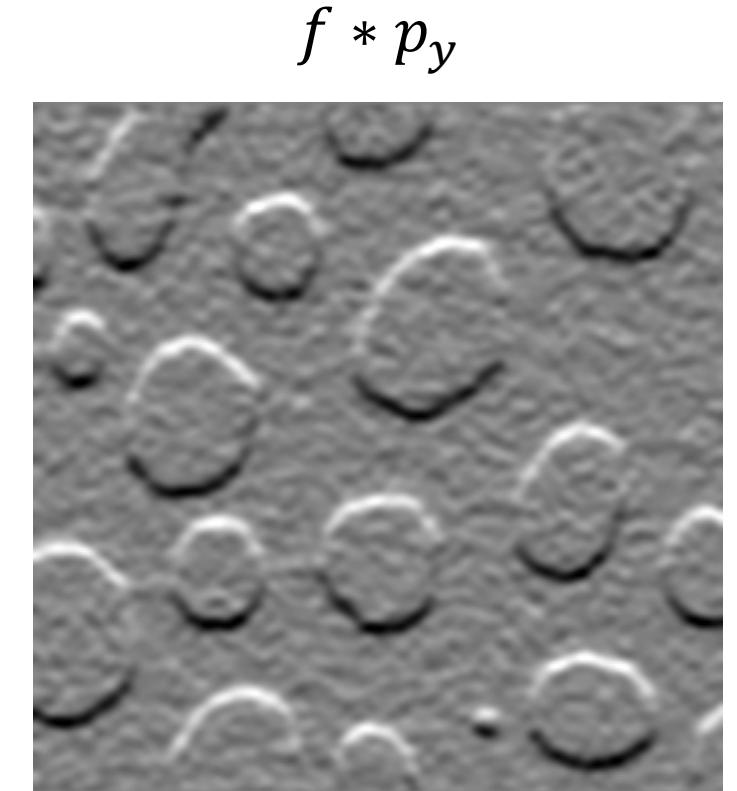
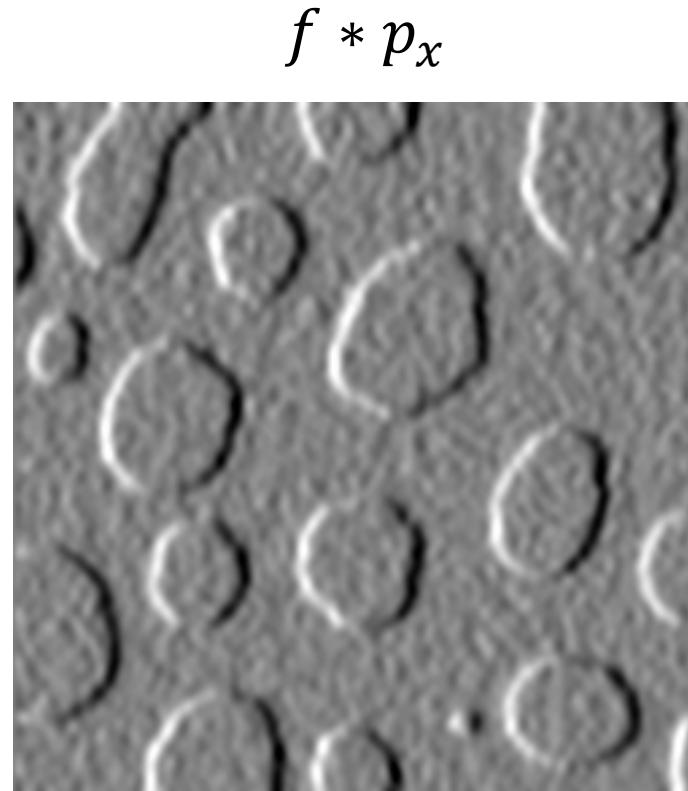
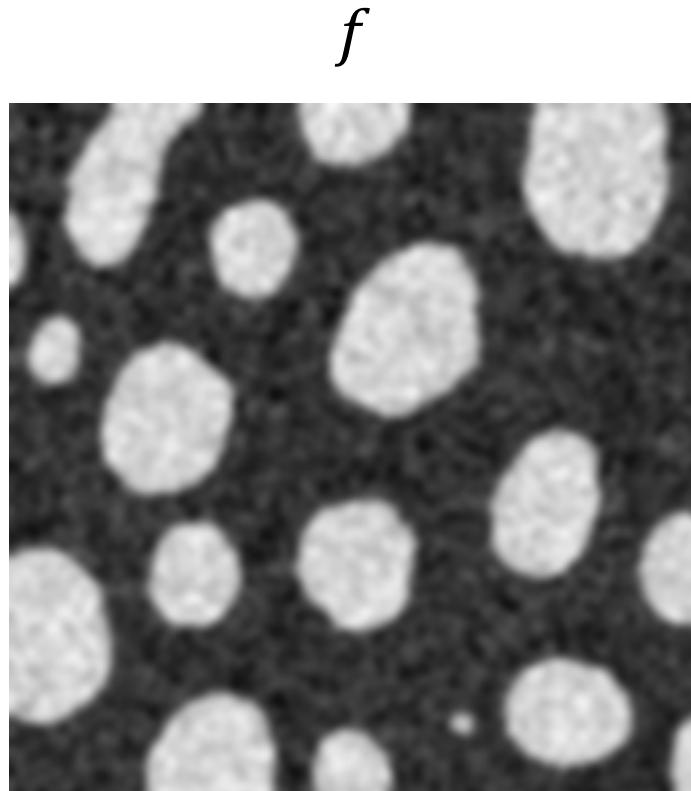
$$g(y) = \frac{1}{4} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}^T$$

$$o(x, y) = \sum_{j=-1}^1 \left[\sum_{i=-1}^1 f(x - i, y - j)g(i) \right] g(j)$$

$$2 \times (3 \text{ multiplies} + 2 \text{ adds}) = 10 \text{ ops/pixel}$$

Even higher gains
for larger kernels
and 3D images

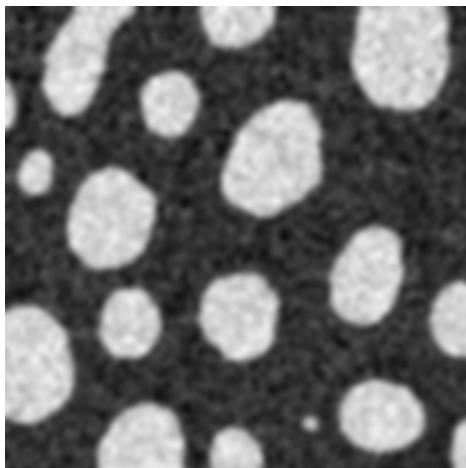
Example of Prewitt filtering



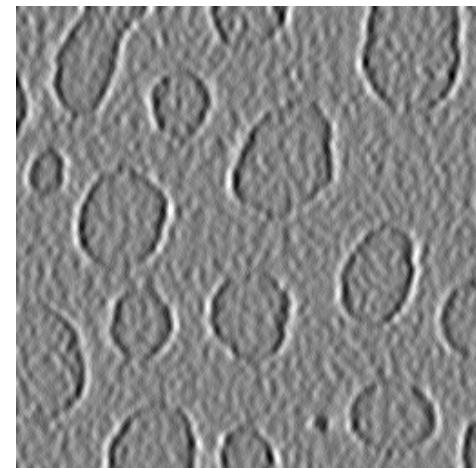
Laplacean filtering

- Approximating the sum of second-order derivatives

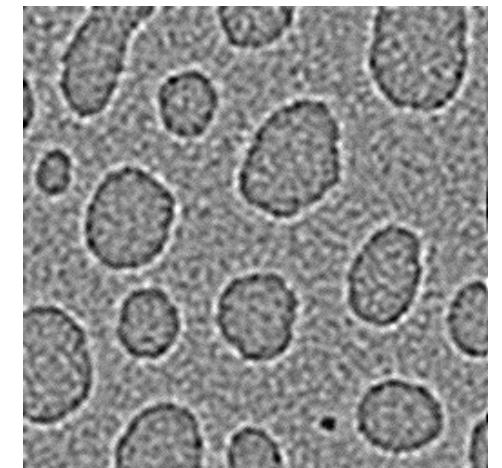
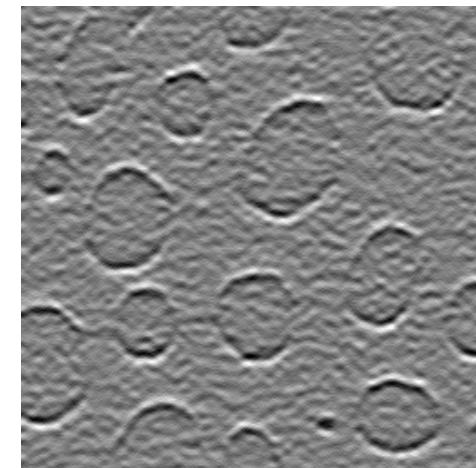
$$f \rightarrow f_{xx} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + f_{yy} \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} = \nabla^2 f \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



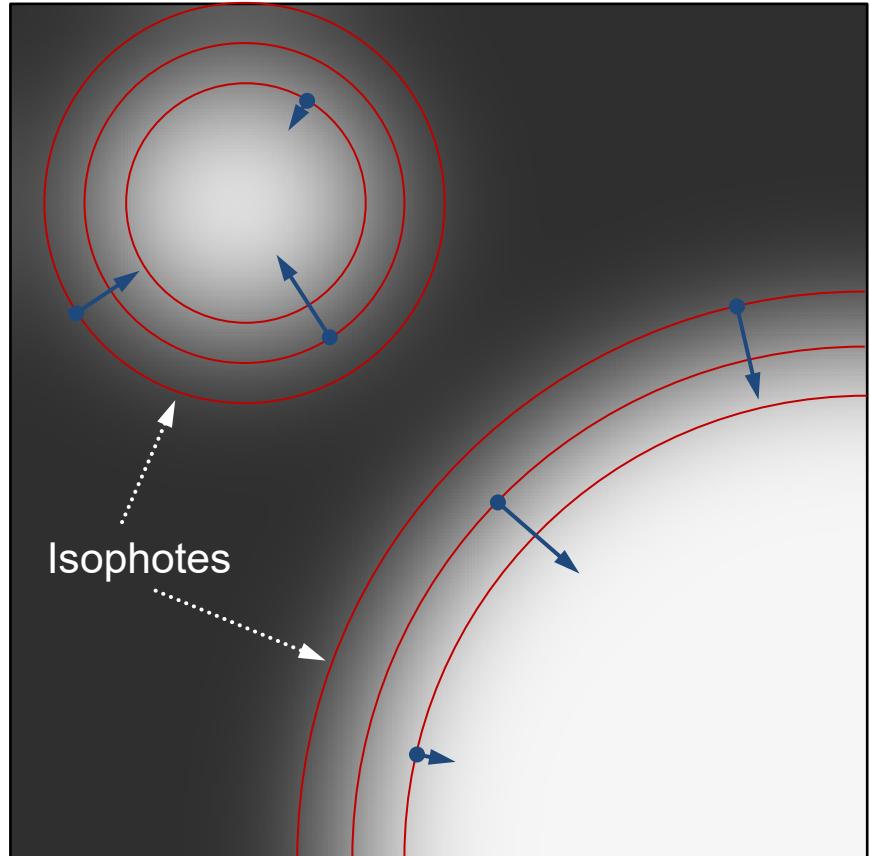
0 255



-255 0 255



Intensity gradient vector



Gradient vector (2D)

$$\nabla f(x, y) = [f_x(x, y), f_y(x, y)]^T$$

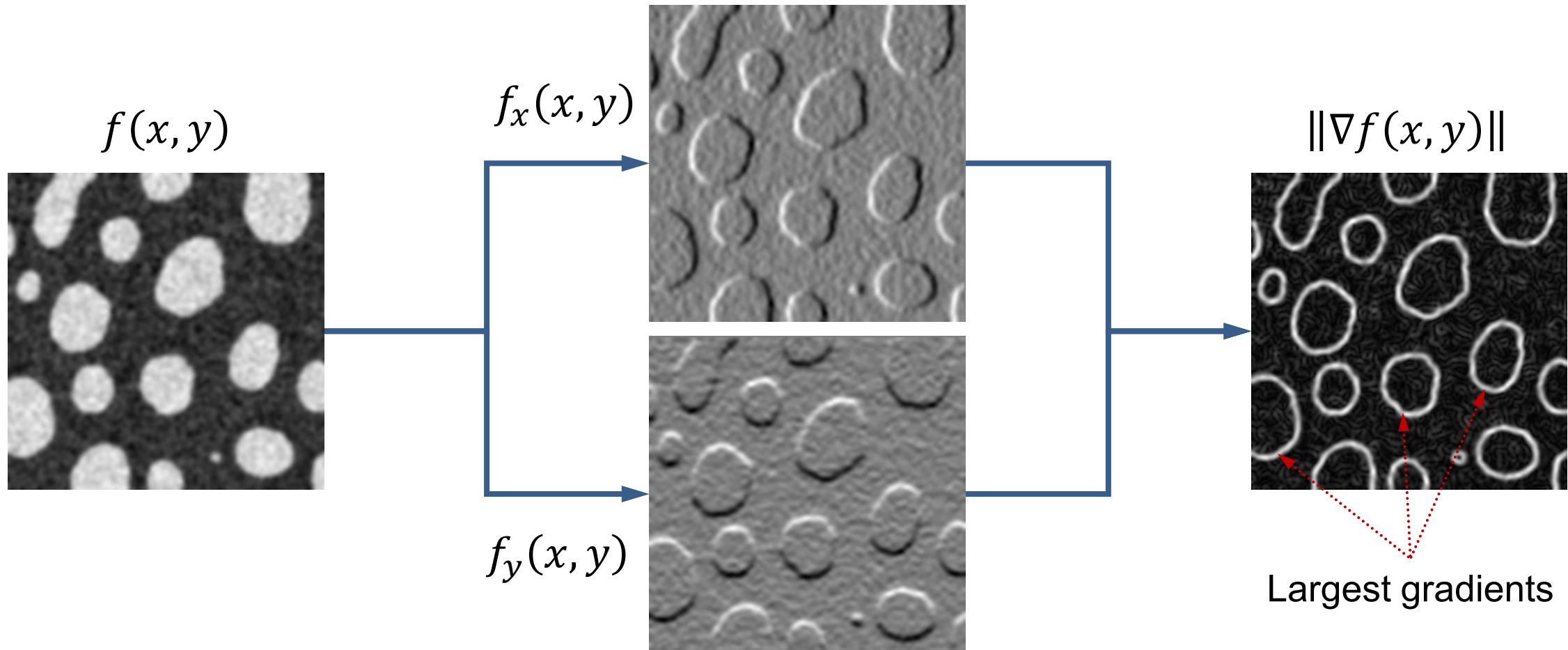
- Points in the direction of steepest intensity increase
- Is orthogonal to isophotes (lines of equal intensity)

Gradient magnitude (2D)

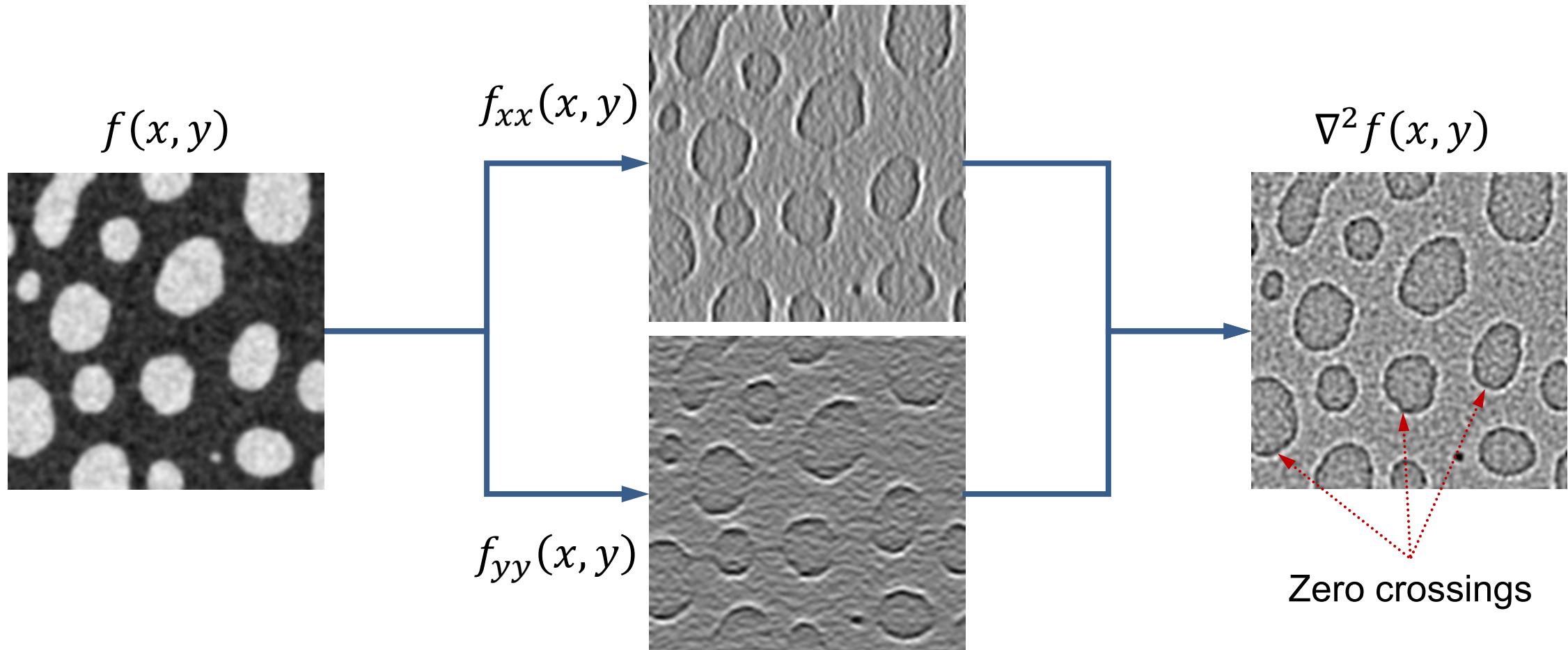
$$\|\nabla f(x, y)\| = \sqrt{f_x^2(x, y) + f_y^2(x, y)}$$

- Represents the length of the gradient vector
- Is the magnitude of the local intensity change

Edge detection with the gradient magnitude



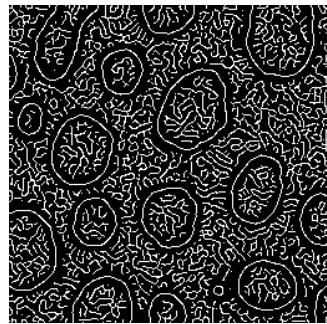
Edge detection with the Laplacean



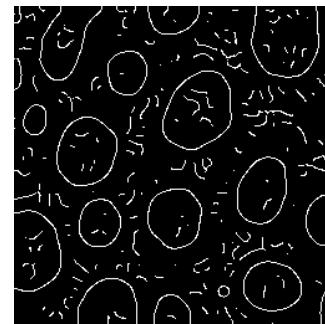
Selecting the right spatial scale

- Computing image derivatives using Gaussian derivative kernels

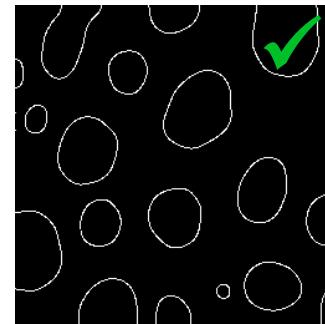
$\sigma = 1$



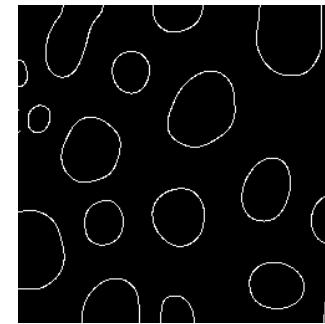
$\sigma = 3$



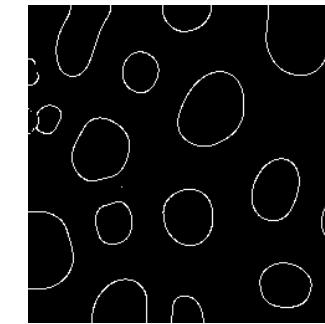
$\sigma = 5$



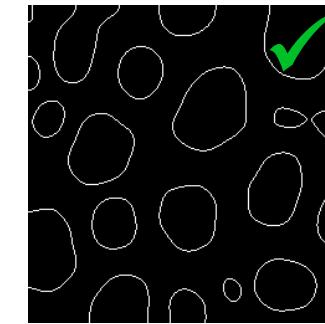
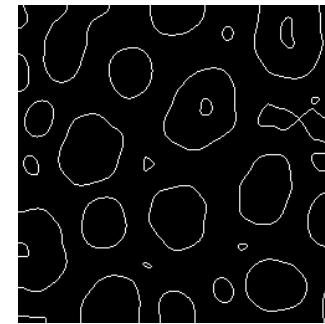
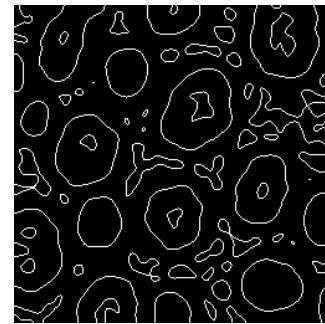
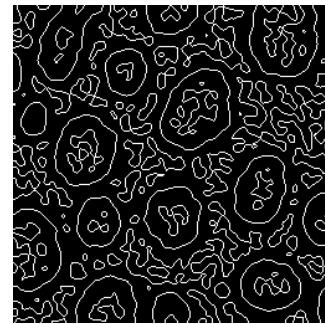
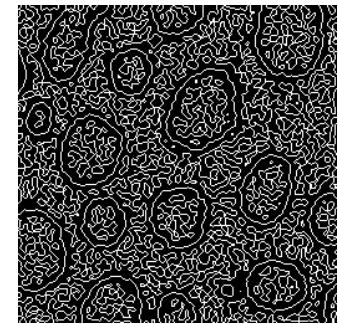
$\sigma = 7$



$\sigma = 9$



Edges from
thresholding local
maxima of $\|\nabla f(x, y)\|$



Edges from finding
the zero-crossings
of $\nabla^2 f(x, y)$

Differentiation in the Fourier domain

Spatial domain

$$f(x)$$



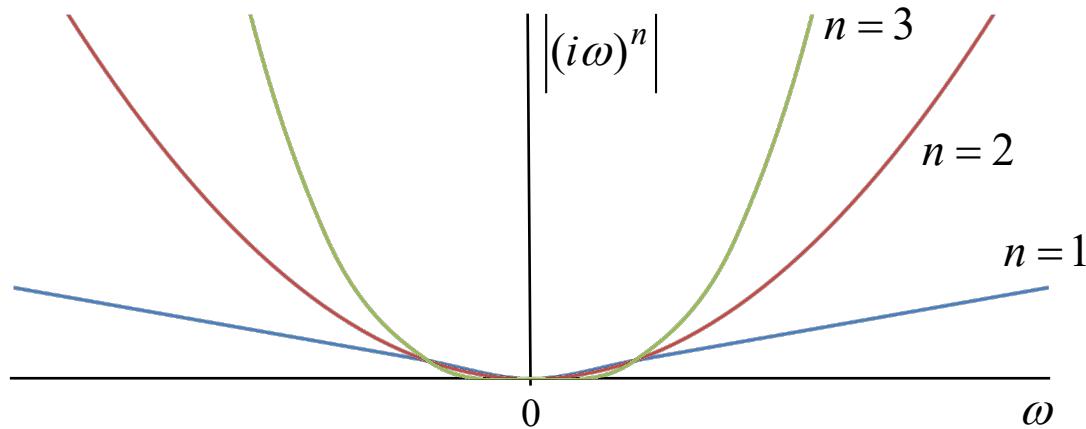
Fourier domain

$$\hat{f}(\omega)$$



$$\frac{\partial^n f}{\partial x^n}(x)$$

$$(i\omega)^n \hat{f}(\omega)$$



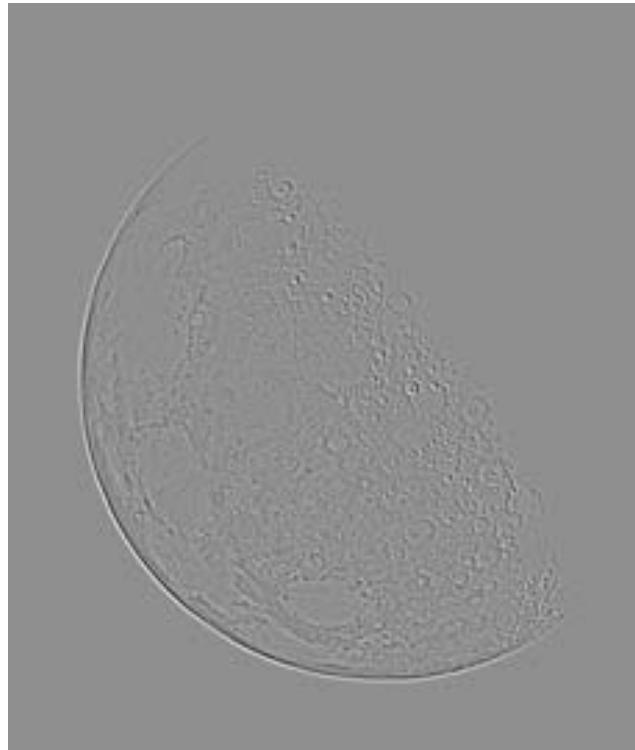
Differentiation
suppresses low
frequencies but blows
up high frequencies
(including noise)

Sharpening using the Laplacean

$f(x, y)$



$\nabla^2 f(x, y)$



$f(x, y) - \nabla^2 f(x, y)$



Further reading on discussed topics

- Chapter 3 of Gonzalez and Woods 2002
- Sections 3.1-3.3 of Szeliski

Acknowledgement

- Some images drawn from the above resources

Example exam question

What is the effect of the 2D convolution kernel shown on the right when applied to an image?

0	1	0
1	-4	1
0	1	0

- A. It approximates the sum of first-order derivatives in x and y .
- B. It approximates the sum of second-order derivatives in x and y .
- C. It approximates the product of first-order derivatives in x and y .
- D. It approximates the product of second-order derivatives in x and y .