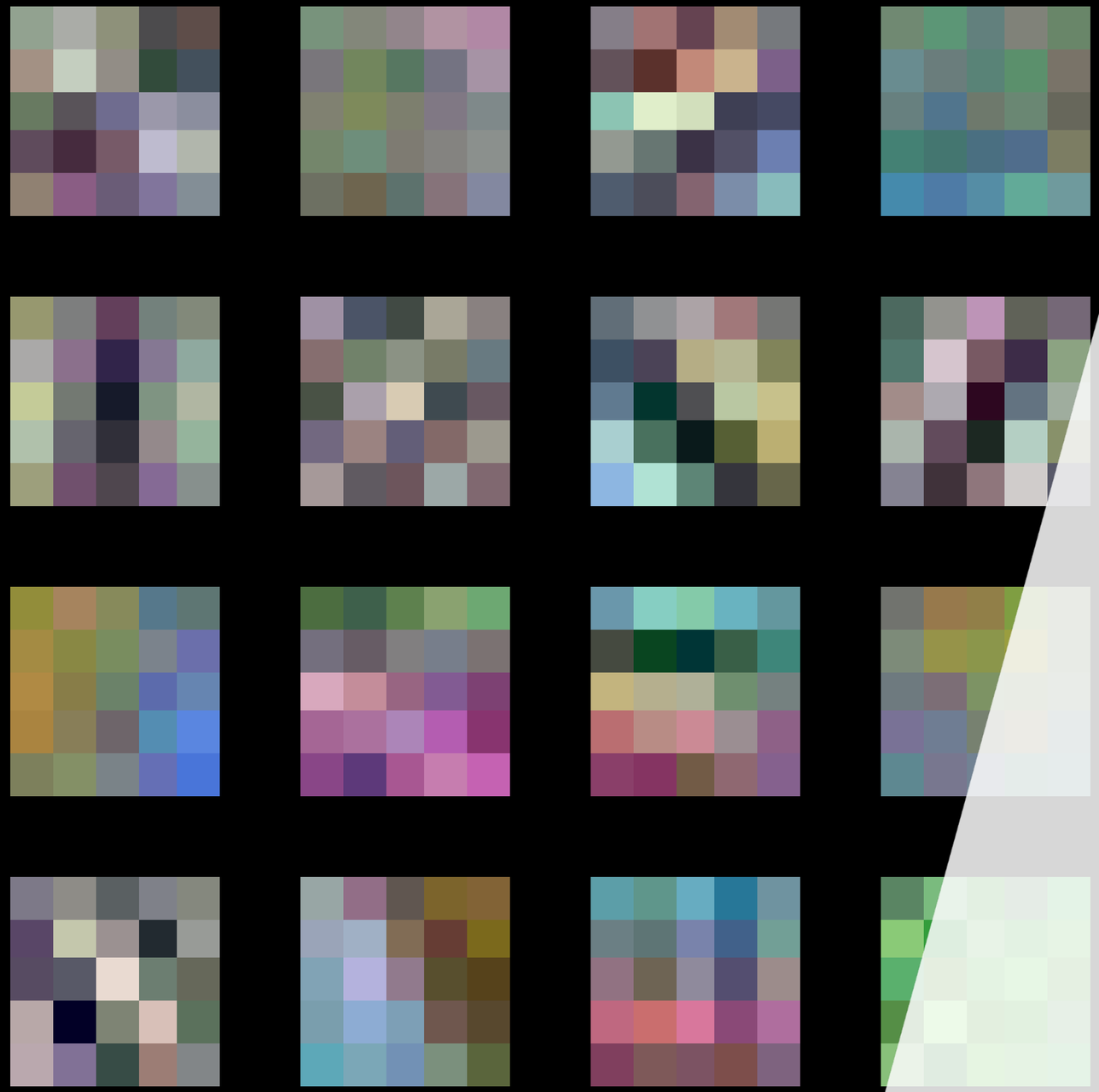


Technische Hochschule Ingolstadt
Fakultät Informatik

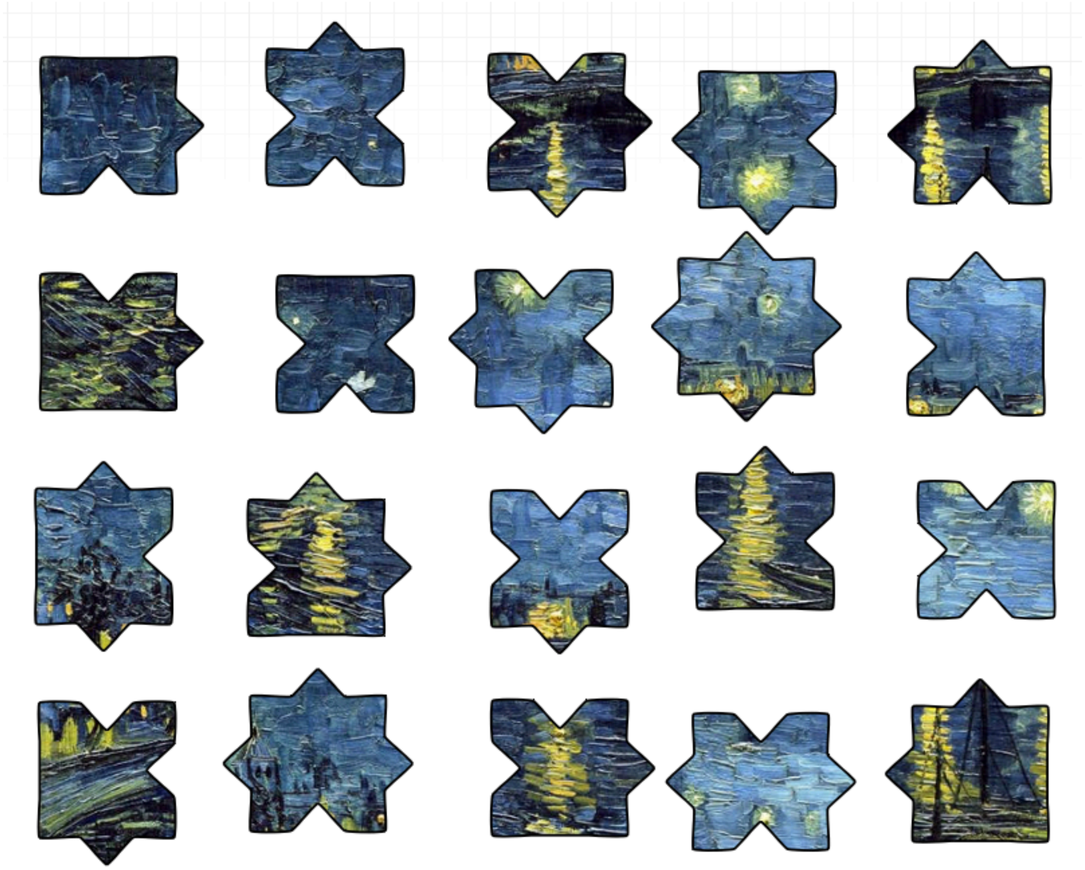
Computer Vision

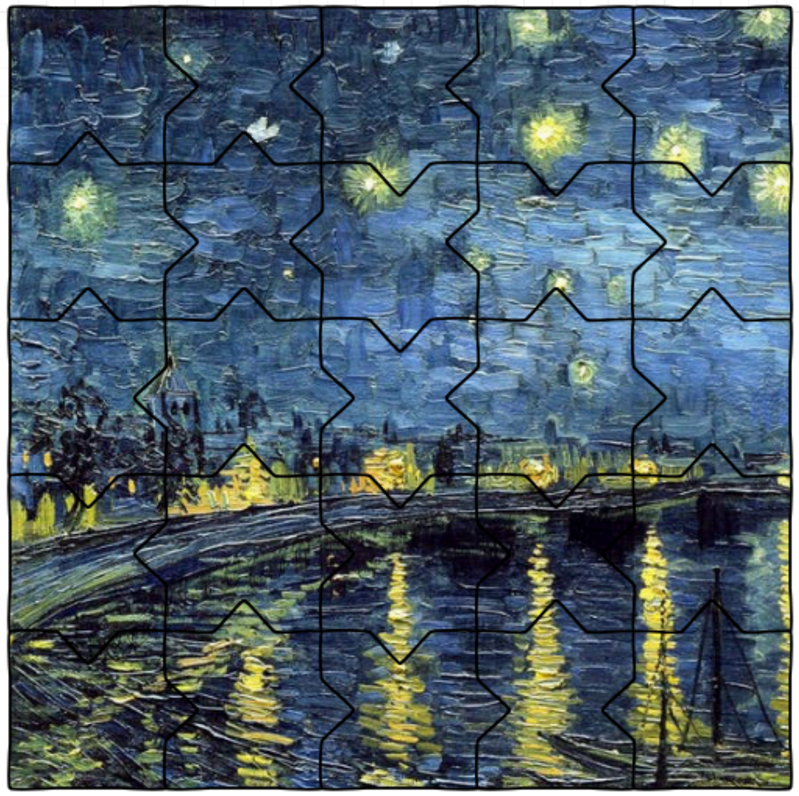
Filtering and Kernels

Prof. Dr. Marc Aubreville, SS 2022

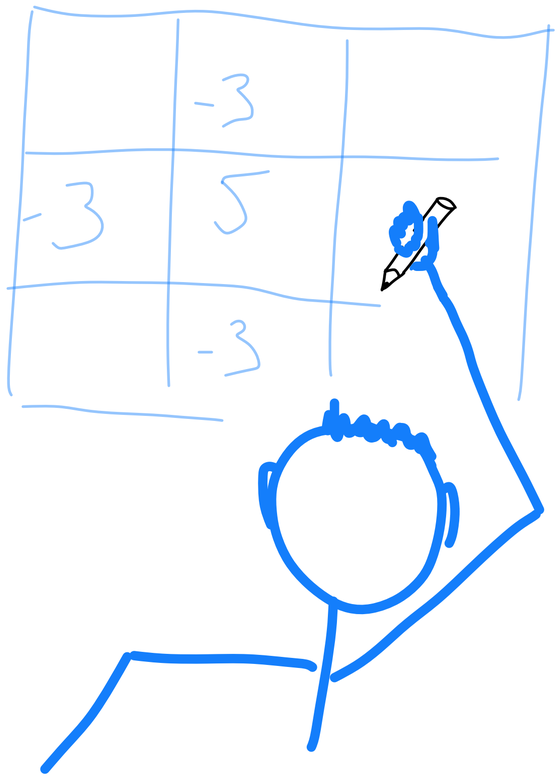
15.03.2022

Motivation: In images, position matters!





https://flbulgarelli.github.io/headbreaker/#autogenerated



Convolutional

kernels

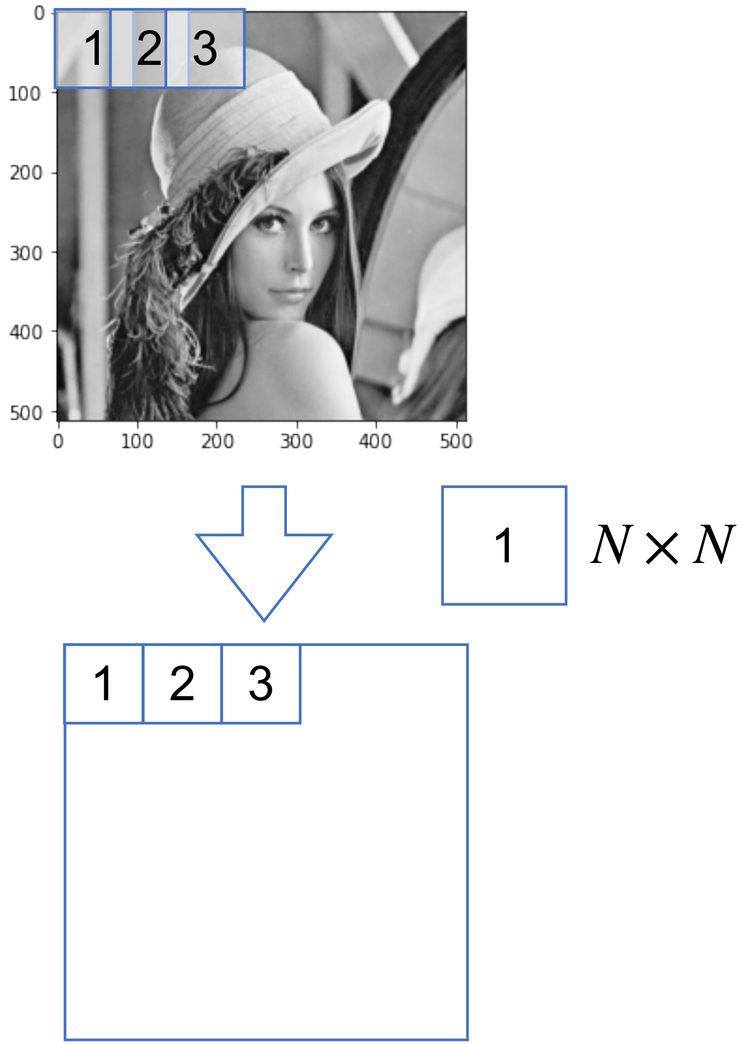
- So far, every operation that we have been doing, has been performed on **independent pixels** only
- The location of these pixels in the image could have been randomly changed without affecting the operations
- We used the values of the pixels from the image to derive characteristics and we have seen some modifications that can be performed on these pixels

In many cases, we are not only interested in single pixel values,
but also in the values of its neighbors!

Slide credit: Prof. Schön

Motivation: The convolution

- **We want to be able to apply an operation that extracts the relationships between neighboring pixels.**
- **We call the result of this operation a descriptor, because it describes the pixel relationship**
- **Naively, this operation should always consider a local environment (e.g., N pixels squared)**
- **And the operation should be applied to all such local environments.**



Convolution

- **The convolution of two functions is defined as:**

$$s(t) * h(t) = \int_{\mathbb{R}^N} s(x+\tau) - h(\tau) d\tau \quad \textbf{or, in discrete terms:} \quad s(n) * h(n) = \sum_u s(n+u) - h(n)$$

- **We can extend this to a two-dimensional convolution**

$$s(x,y) * h(u,v) = \sum_u \sum_v s(x+u, y+v) \cdot h(u,v)$$

Convolution (cont'd)

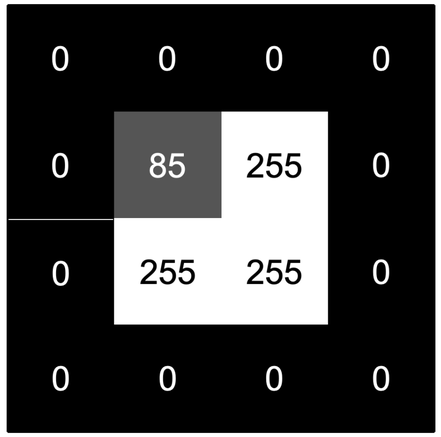


Image S

1	1	1
1	1	1
1	1	1

Filter kernel h

size: $m \times m$

- **It is sensible to normalize the convolution by the size of the filtering element (kernel):**

$$s(x,y) * h(u,v) = \frac{1}{m^2} \sum_u \sum_v s(x+u, y+v) \cdot h(u,v)$$

- **For convenience, let's shift the equation so the kernel is centered:**

$$s_{out}(x,y) = \frac{1}{m^2} \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} s_{in}(x+k-u, y+k-v) \cdot h(u,v)$$

$$k = \frac{m-1}{2}$$

Note: To have a clear center definition, odd sized kernels are sensible.

Kernels

The general idea

4x4 Image s_e

0	0	0	0
0	85	255	0
0	255	255	0
0	0	0	0

3x3 Kernel $h(u, v)$

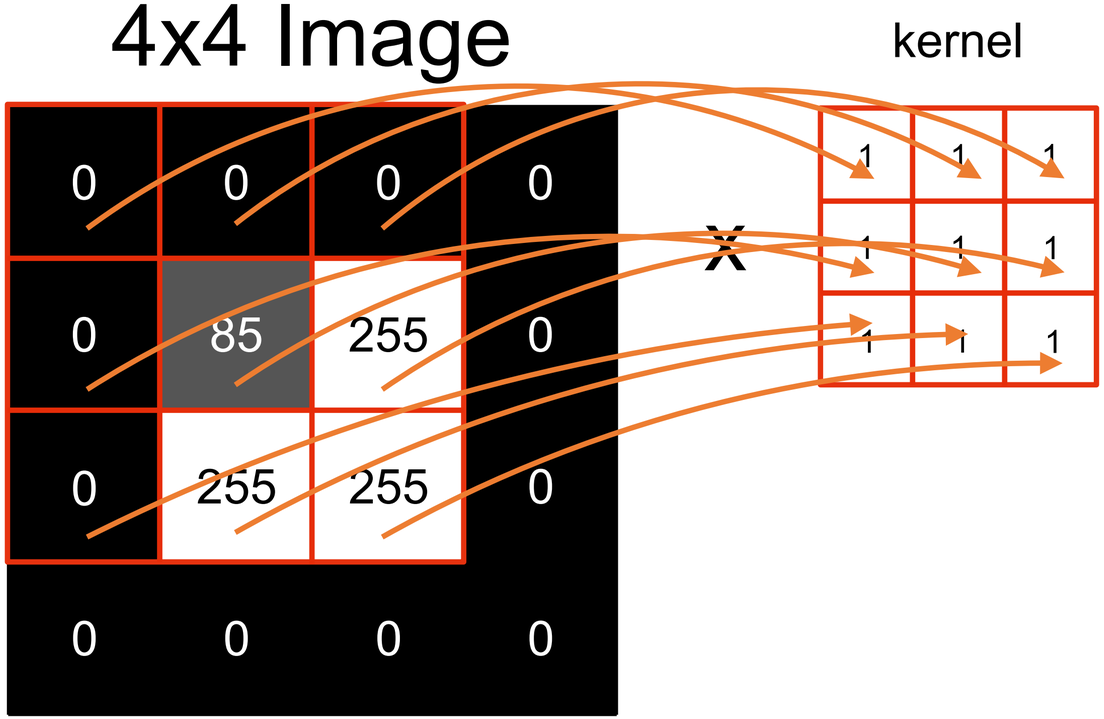
1	1	1
1	1	1
1	1	1

$$s_{out}(x, y) = \frac{1}{m^2} \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} s_{in}(x + k - u, y + k - v) \cdot h(u, v)$$

where m is the length of the mask

Slide credit: Prof. Schön

Convolutional kernels: Example



$$0 * 1 + 0 * 1 + 0 * 1$$

$$+ 0 * 1 + 85 * 1 + 255 * 1 + 0 * 1 + 255 * 1 + 255 * 1$$

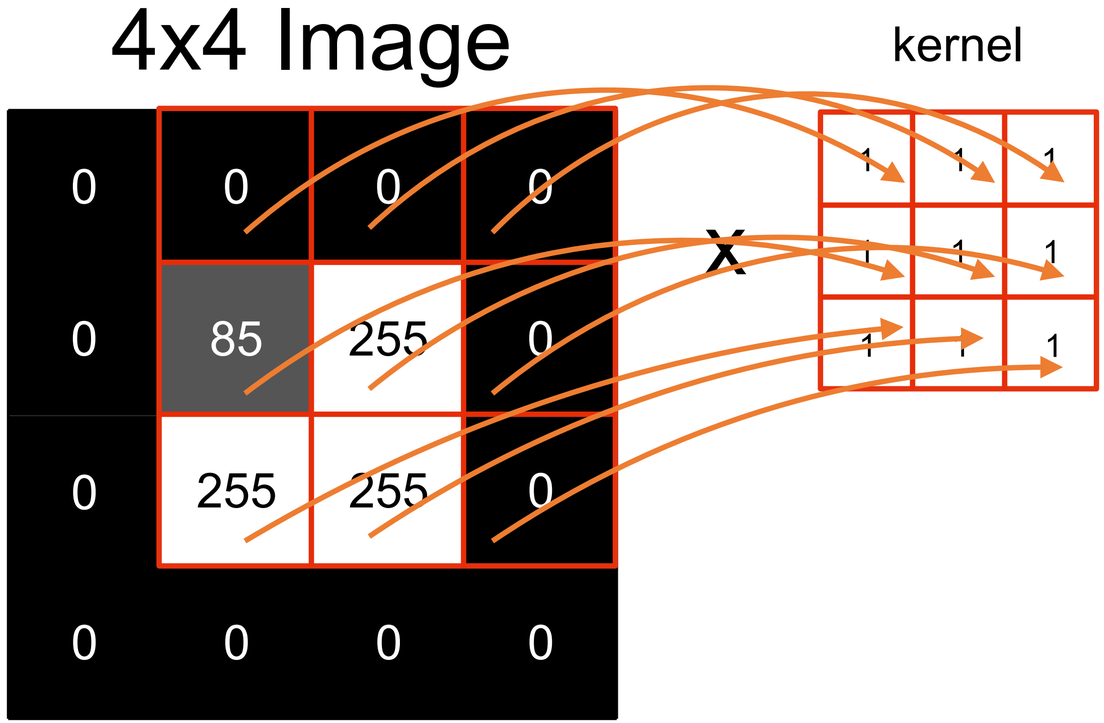
$$= 85 + 255 + 255 + 255$$

$$= 850$$

$$\frac{850}{9} = 94$$

Slide credit: Prof. Schön

Convolutional Kernels: Example (2)



$$0 * 1 + 0 * 1 + 0 * 1$$

$$+ 85 * 1 + 255 * 1 + 0 * 1 + 255 * 1 + 255 * 1 + 0 * 1$$

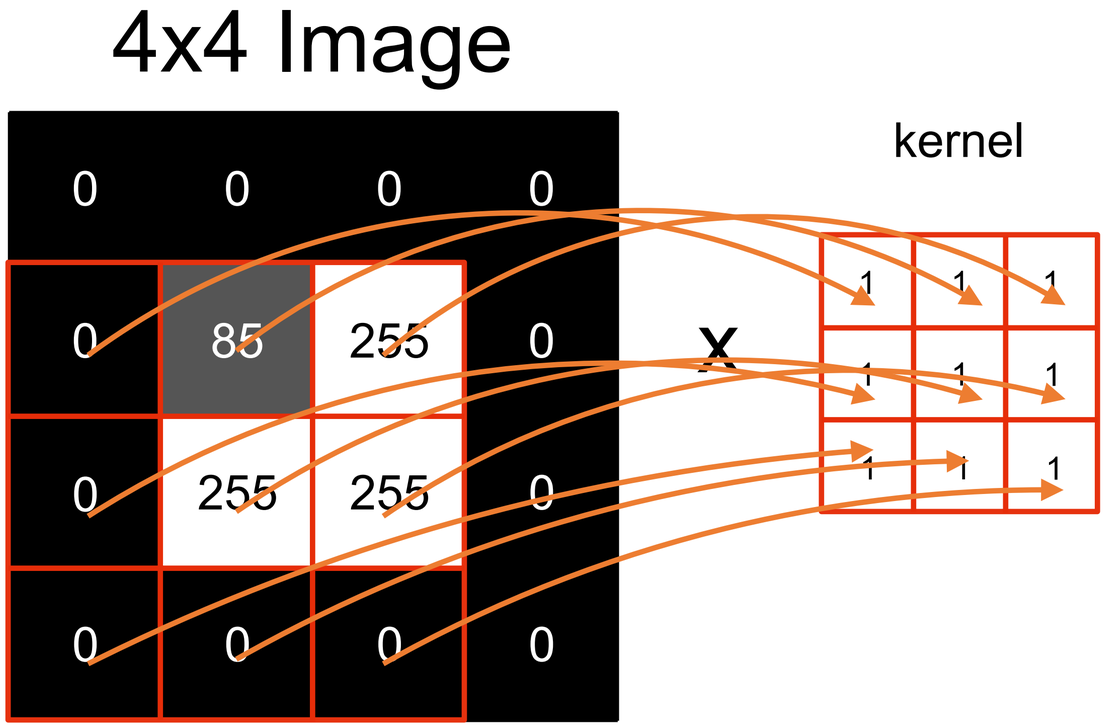
$$= 85 + 255 + 255 + 255$$

$$= 850$$

$$\frac{850}{9} = 94$$

Slide credit: Prof. Schön

Convolutional Kernels: Example (3)



$$0 * 1 + 85 * 1 + 255 * 1 + 0 * 1 + 255 * 1 + 255 * 1 + 0 * 1 + 0 * 1 + 0 * 1$$

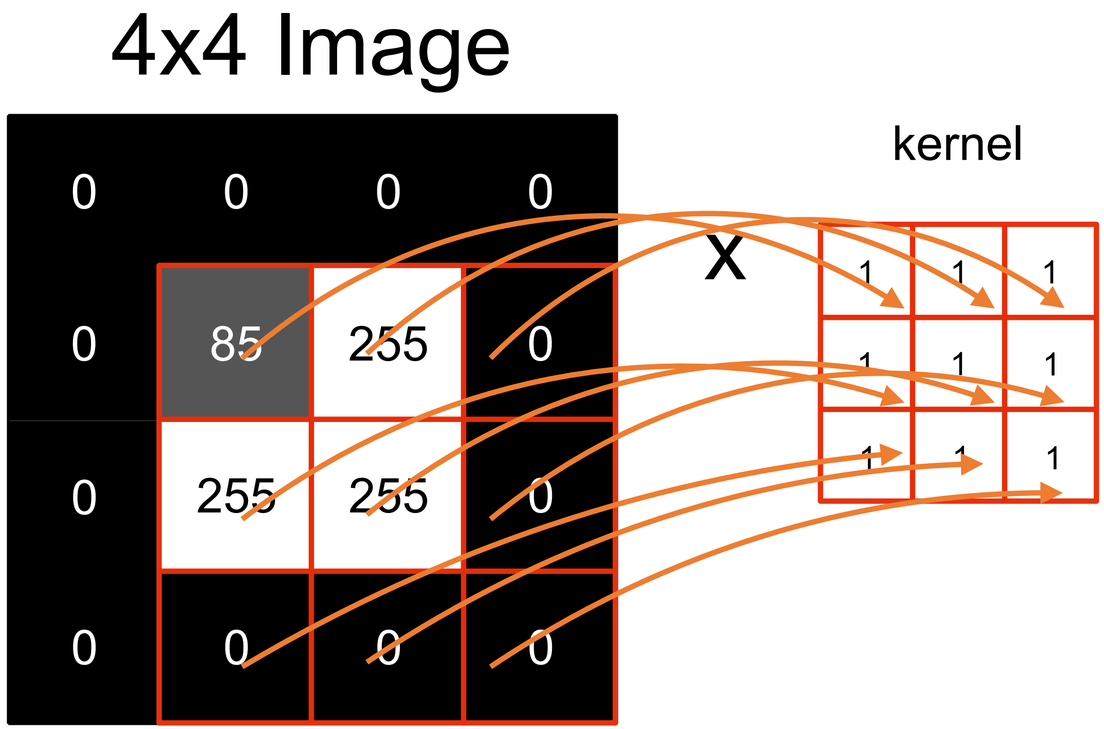
$$= 85 + 255 + 255 + 255$$

$$= 850$$

$$\frac{850}{9} = 94$$

Slide credit: Prof. Schön

Convolutional Kernels: Example (4)



$$85 * 1 + 255 * 1 + 0 * 1 + 255 * 1 + 255 * 1 + 0 * 1 + 0 * 1 + 0 * 1 + 0 * 1$$

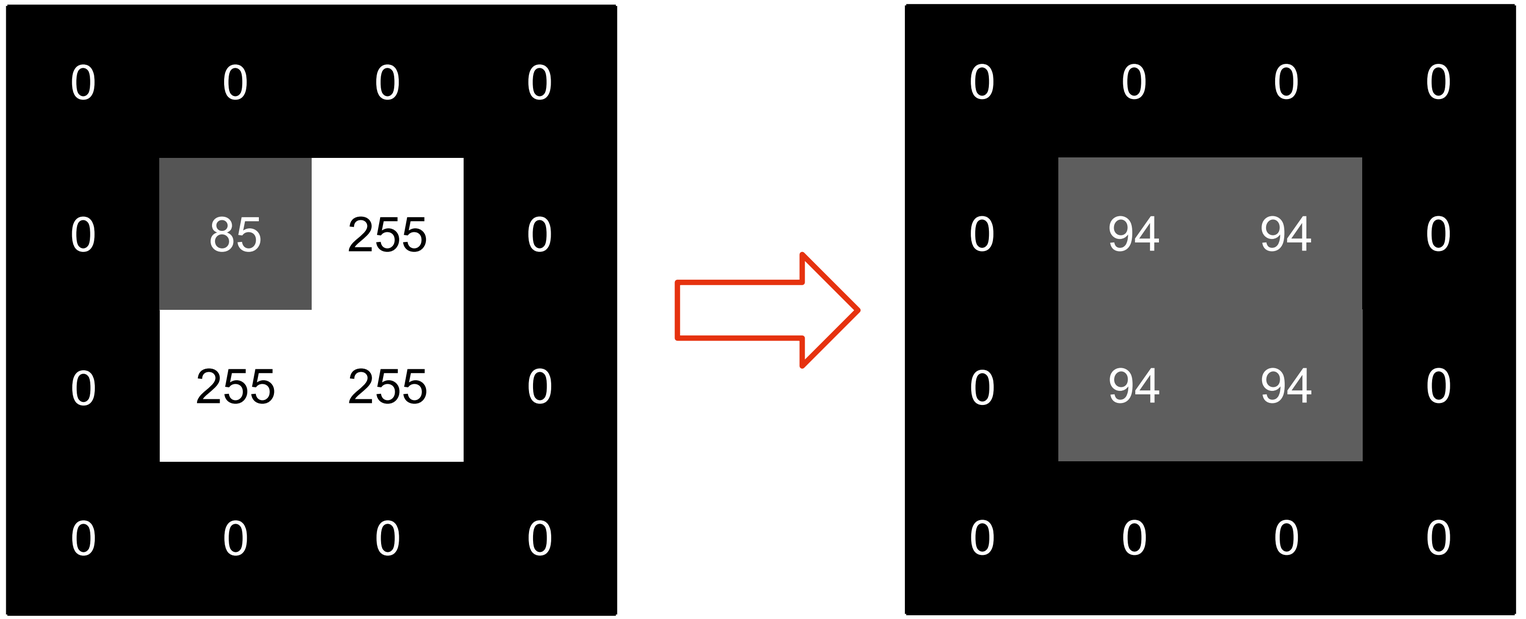
$$= 85 + 255 + 255 + 255$$

$$= 850$$

$$\frac{850}{9} = 94$$

Slide credit: Prof. Schön

The general idea

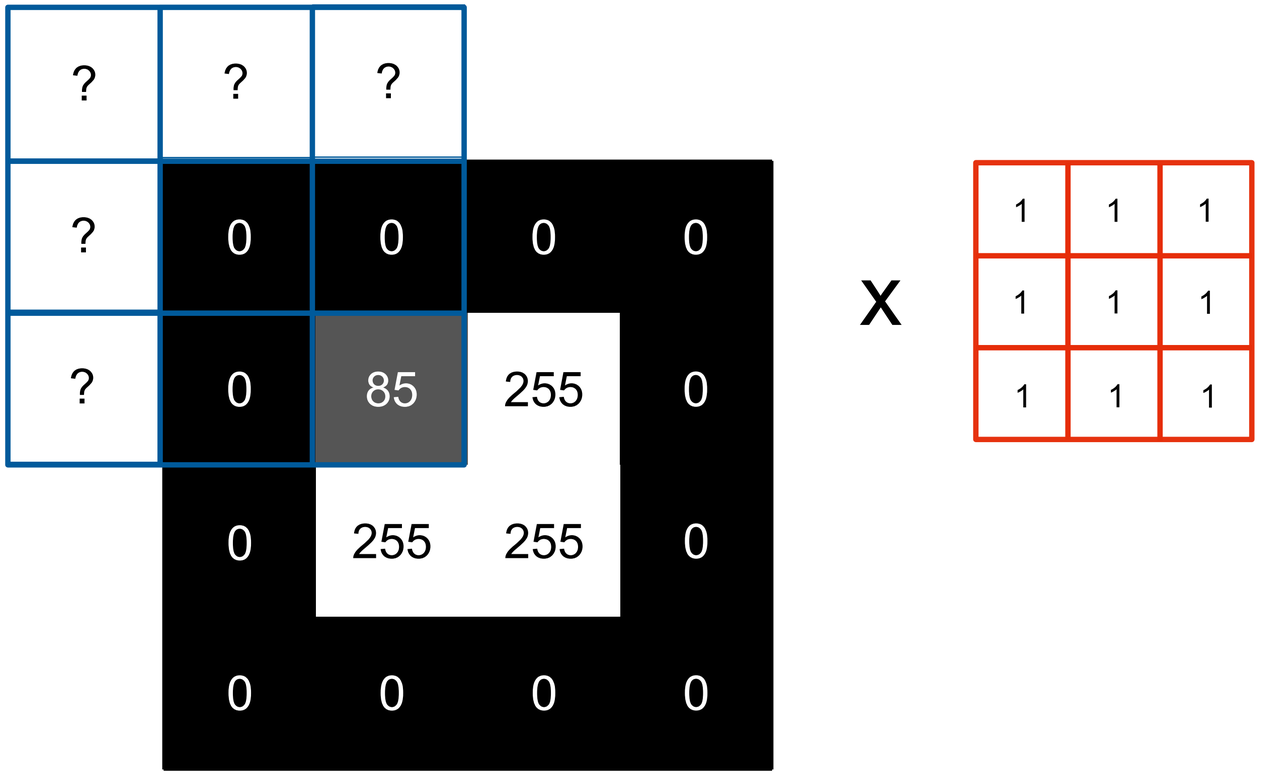


We skipped all the border pixels

Slide credit: Prof. Schön

Kernels

What can we do at the borders?



Slide credit: Prof. Schön

Kernels

What can we do at the borders?

94	94
94	94

1st option:

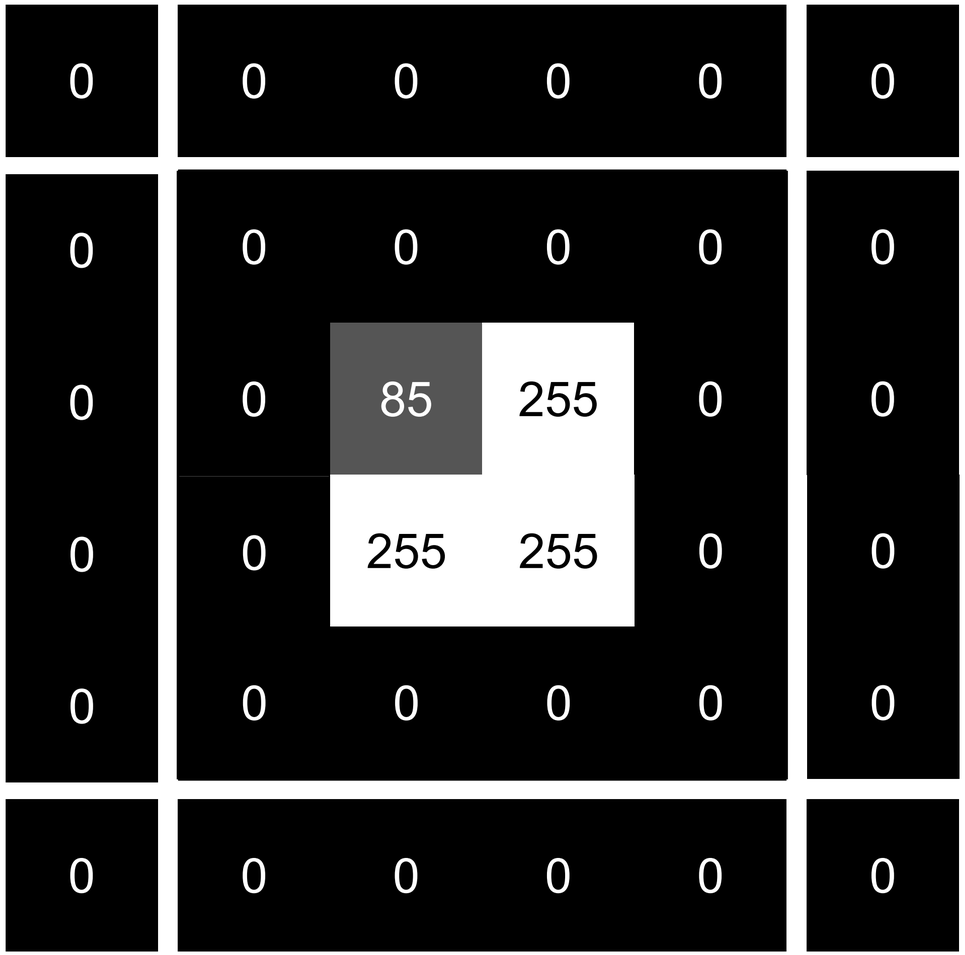
➔ Skip the outer pixels

➔Image gets smaller

Slide credit: Prof. Schön

Kernels

What can we do at the borders?



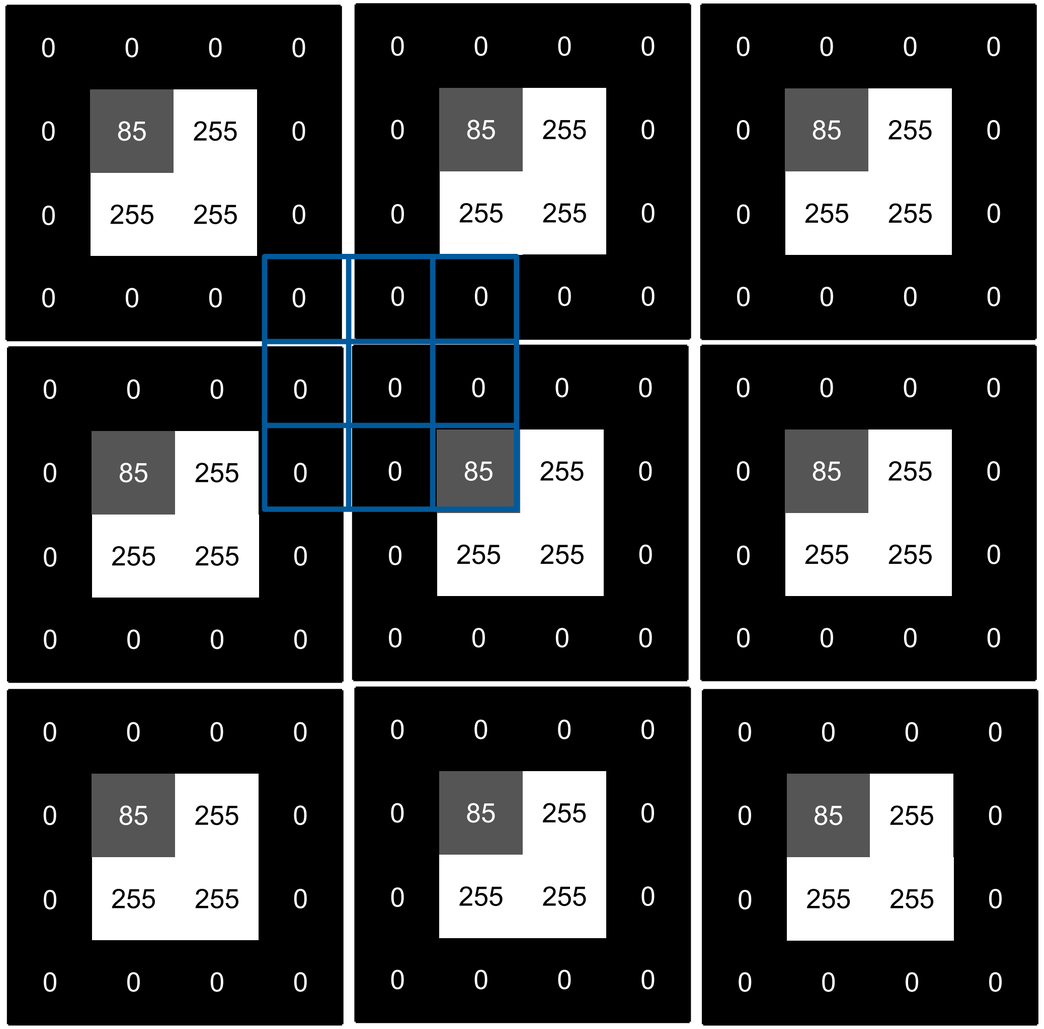
2nd option: Extend:

➔ Copy the border pixels
➔Image keeps same size

Slide credit: Prof. Schön

Kernels

What can we do at the borders?



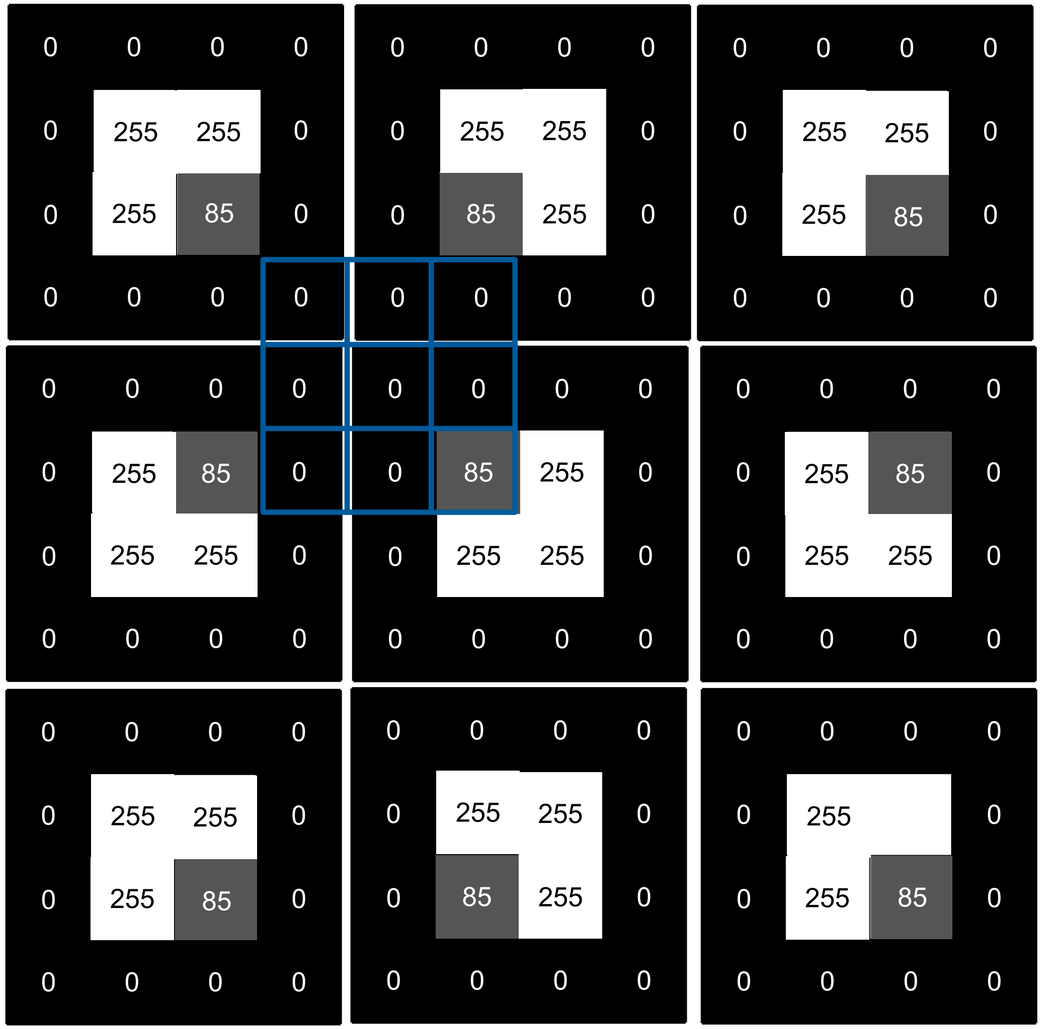
3nd option: Concatenate:

➔ Add the image next to each other

Slide credit: Prof. Schön

Kernels

What can we do at the borders?



4th Option: Mirror:

➔ Add a mirrored image to each side

.

Slide credit: Prof. Schön

Kernels

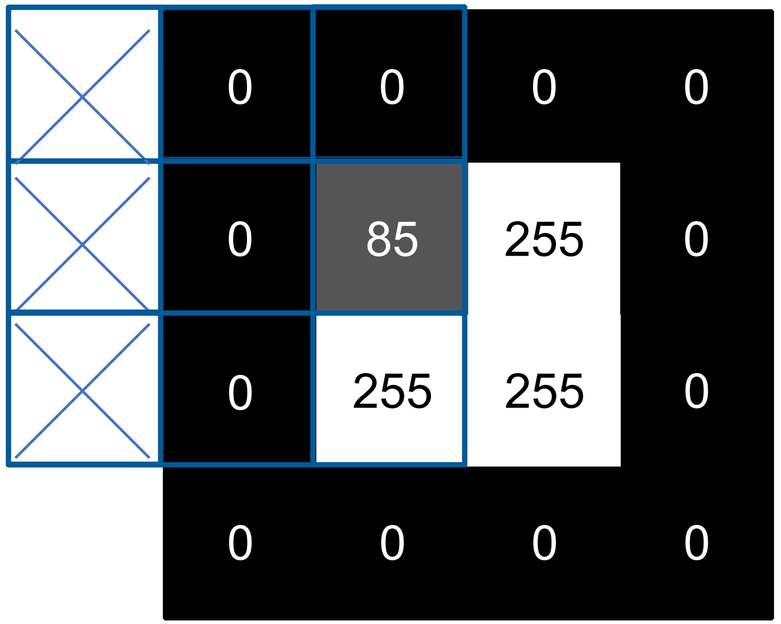
What can we do at the borders?

4th Option: Mirror:

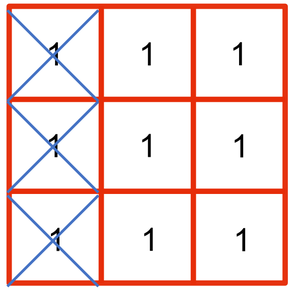
➔ Add a mirrored image to each side

Kernels

What can we do at the borders?



X



5th option: kernel crop

➔ Crop the kernel according to the available pixels in the image.

Kernel normalization

- **We defined the convolution to include a normalization of the kernel**

$$s(x,y) * h(u,v) = \frac{1}{m^2} \sum_u \sum_v s(x+u, y+v) \cdot h(u,v)$$

- **This is sensible, since the value range of pixels (0..255) shall be retained in the filtered image.**
- **Dividing by m^2 actually ensures that we don't exceed the value limit of 255.**
- **This also makes applications of (e.g.) cropped filters possible (different normalization in the cropped case)**

Kernel normalization (cont'd)

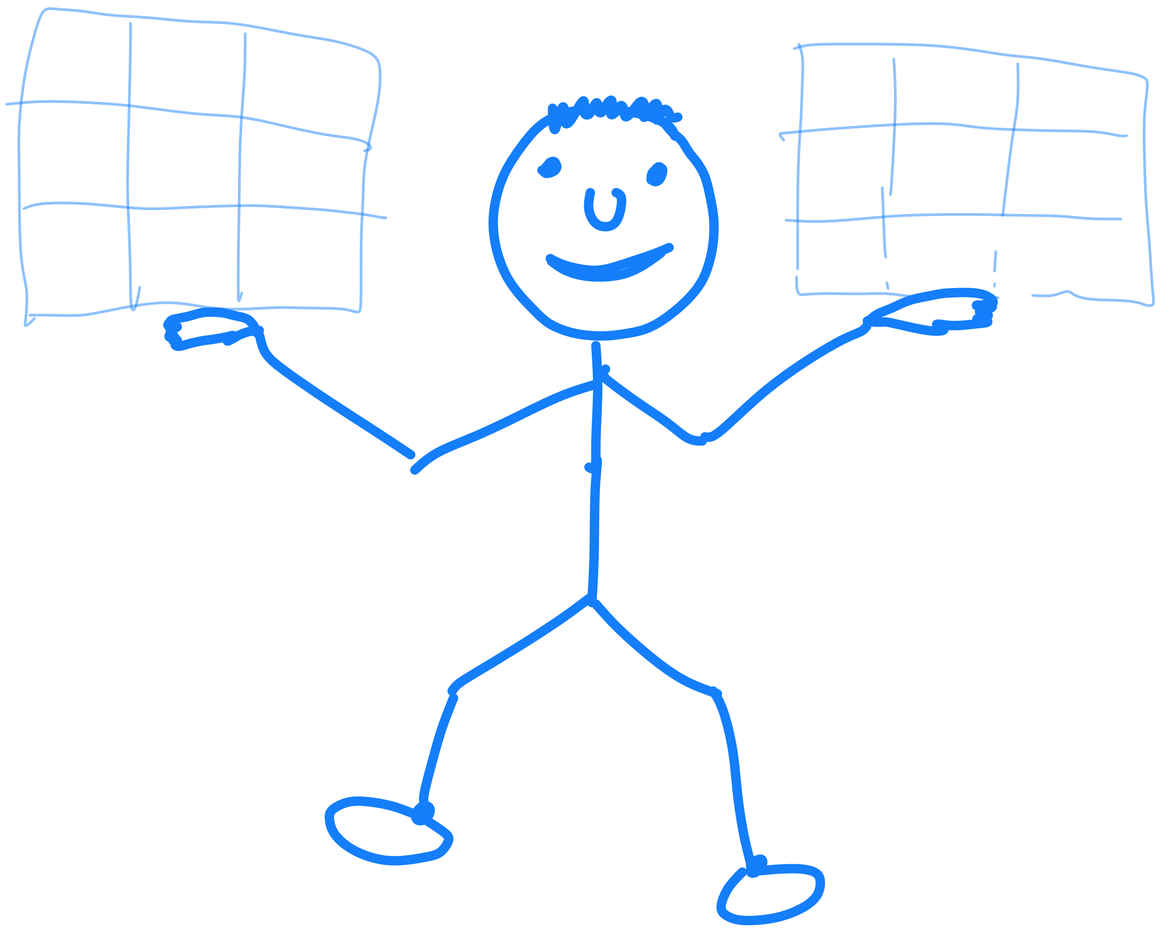
- **But what if the kernel is not only ones?**

0	1	0
1	2	1
0	1	0

- **Dividing by 9 would lead to a too low value range.**
- **Instead, we can divide by the sum of the kernel:**

$$s_{out}(x,y) = \frac{1}{\sum_u \sum_v h(u,v)} \sum_{u=0}^{m-1} \sum_{v=0}^{m-1} s_{in}(x+k-u, y+k-v) \cdot h(u,v)$$

- **For the example on the left, the normalization term would be $\frac{1}{6}$**



Kernel

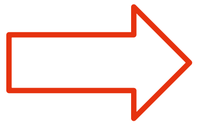
types

Kernel Types
Identity

0	0	0
0	1	0
0	0	0

Keeps the original image

Normalizing factor: $\frac{1}{1} = 1$



Slide credit: Prof. Schön

Kernel Types
Box Blur

1	1	1
1	1	1
1	1	1

Takes an **average** of all neighbors with **equal weight**

Normalizing factor: $\frac{1}{9}$

Kernel
15x15

Slide credit: Prof. Schön

Kernel Types
Gaussian Blur

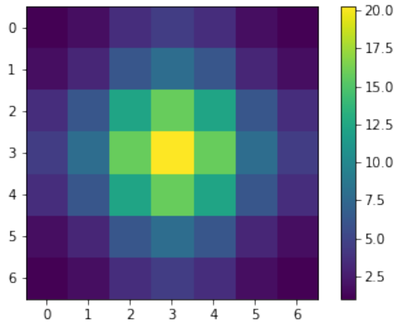
3x3 gaussian blur

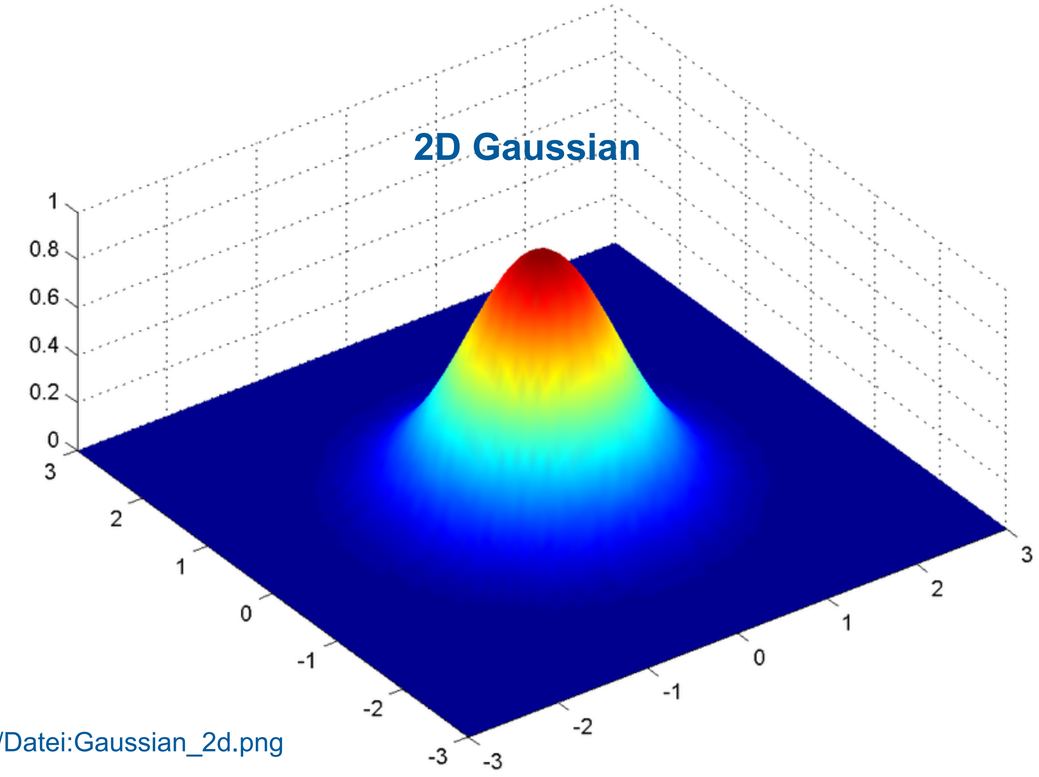
1	2	1
2	4	2
1	2	1

Takes an **average** of all neighbors with **approximated gauss distribution weight**

Normalizing factor: $\frac{1}{16}$

7x7 gaussian blur





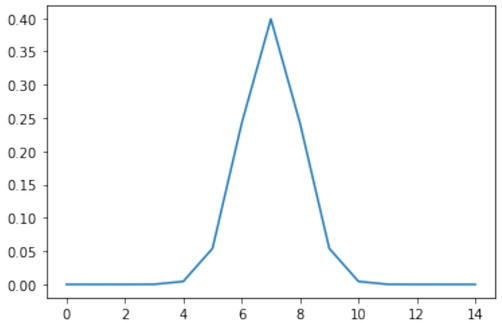
Quelle:
https://de.m.wikipedia.org/wiki/Datei:Gaussian_2d.png

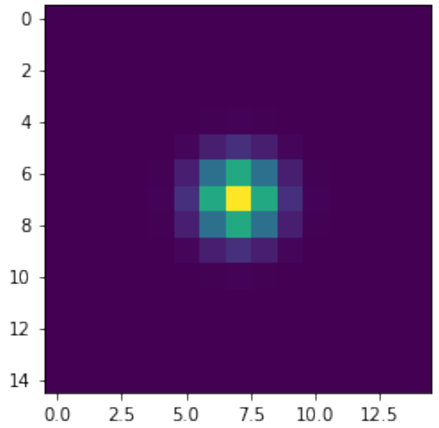
Slide credit: Prof. Schön

Gaussian blur filters

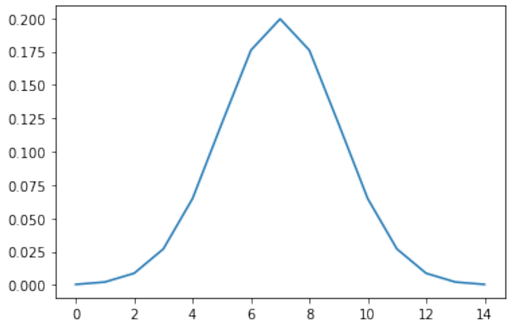
- **Gaussian bell curves have a parameter: σ**

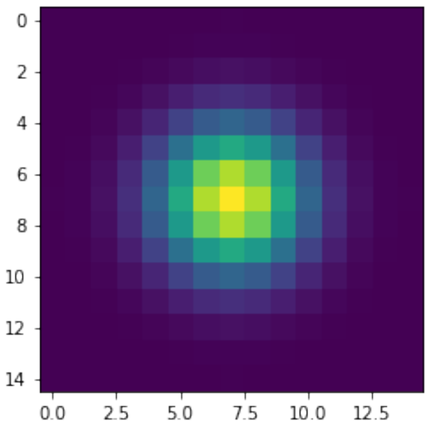
$\sigma = 1$



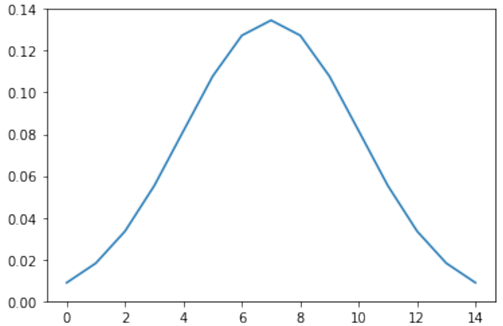


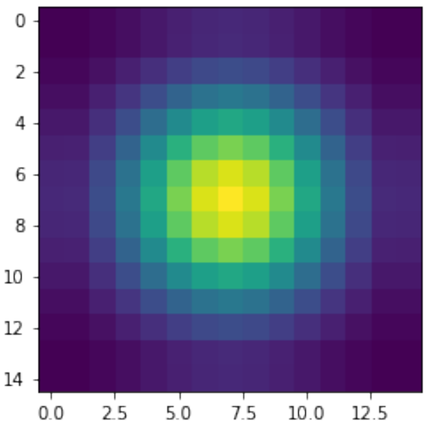
$\sigma = 2$





$\sigma = 3$



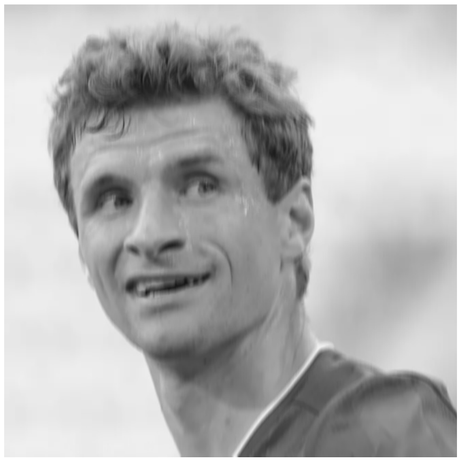


Kernel Types
Gaussian Blur 5x5

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

Takes an **average** of all neighbors with **approximated gauss bell weight**

Normalizing factor: $\frac{1}{256}$



Kernel 10x10

$\sigma = 10$

Slide credit: Prof. Schön

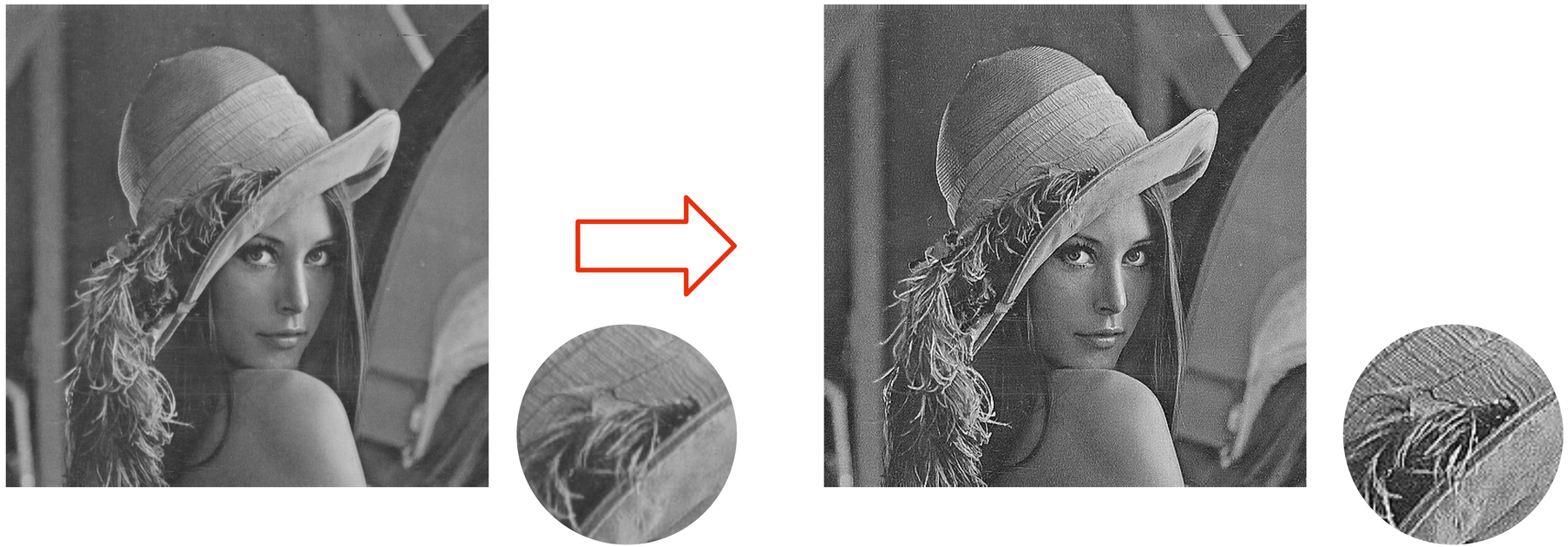
Kernel Types

Sharpen

0	-1	0
-1	5	-1
0	-1	0

Takes the difference to the neighboring pixels which leads to a sharpening effect

Normalizing factor: $\frac{1}{4 * -1 + 5} = \frac{1}{1} = 1$

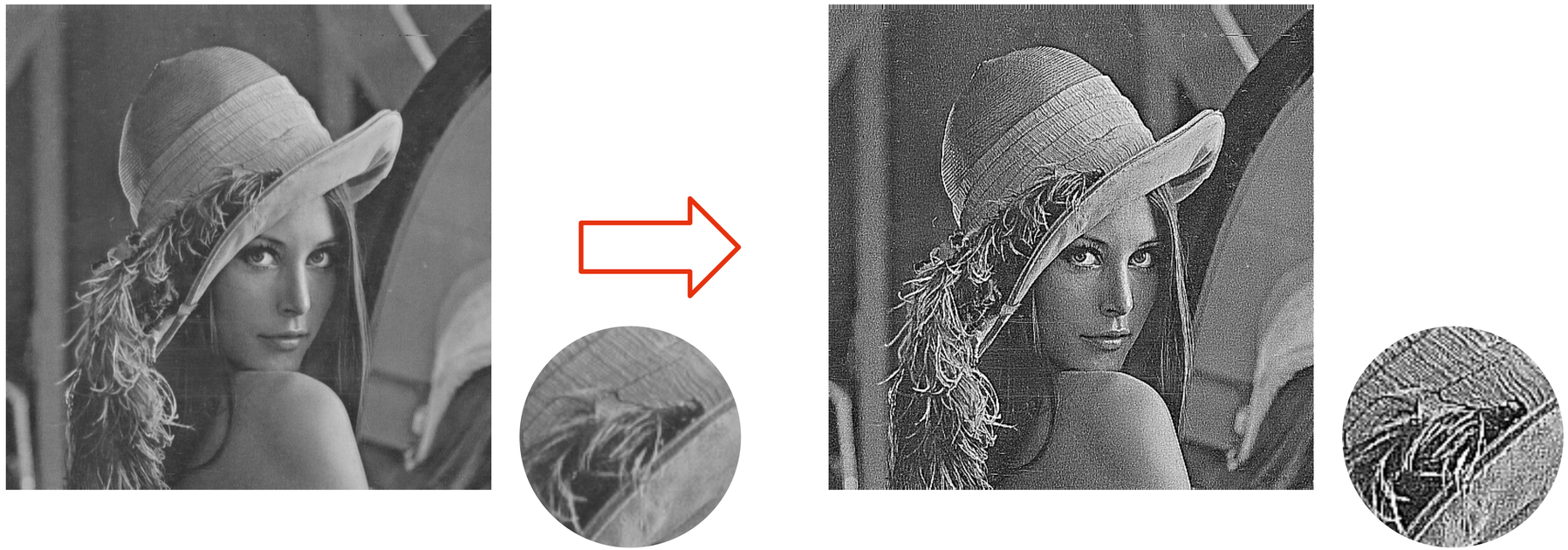


Kernel Types

Sharpen

-1	-1	-1
-1	9	-1
-1	-1	-1

If we enhance the steepness of the filter, we get a stronger effect



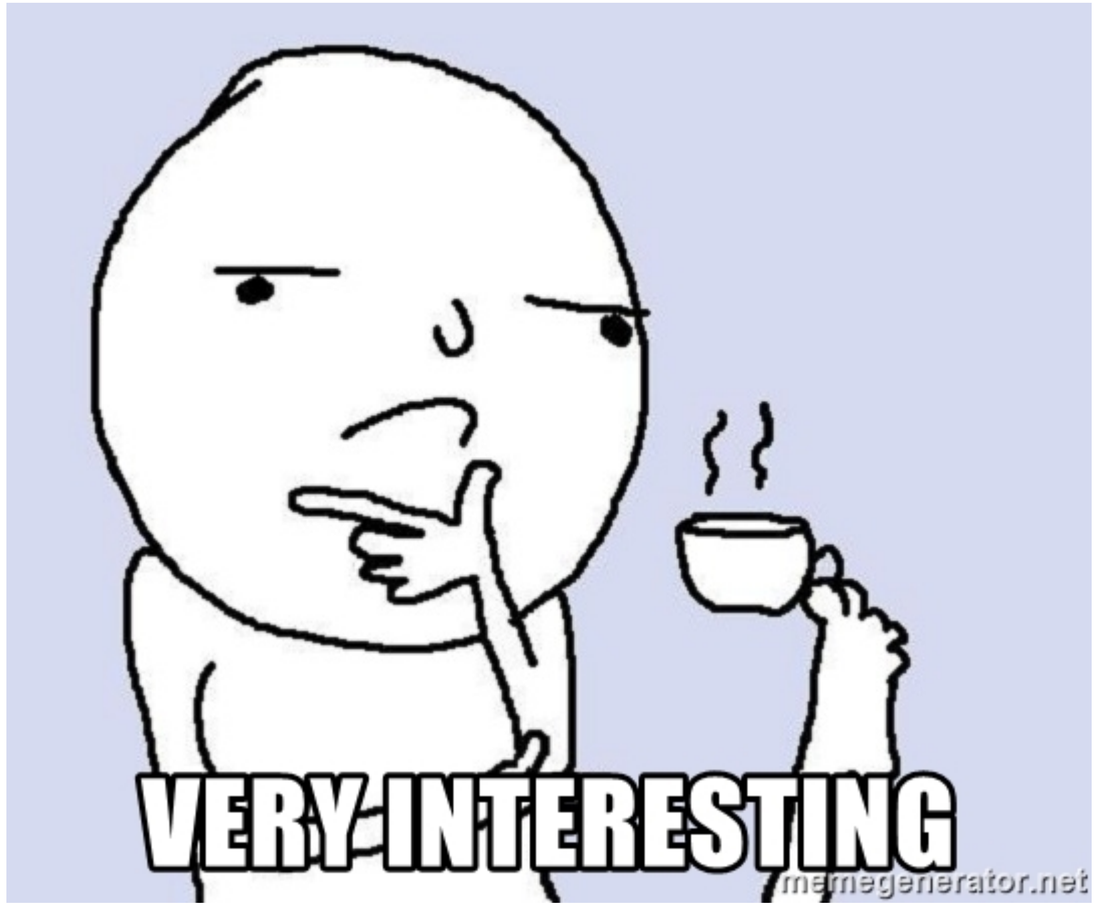
Kernel Types

Ridge detection

-1	-1	-1
-1	8	-1
-1	-1	-1

If we vary the sharpening kernel just a tiny bit, we get to a ridge detection filter.

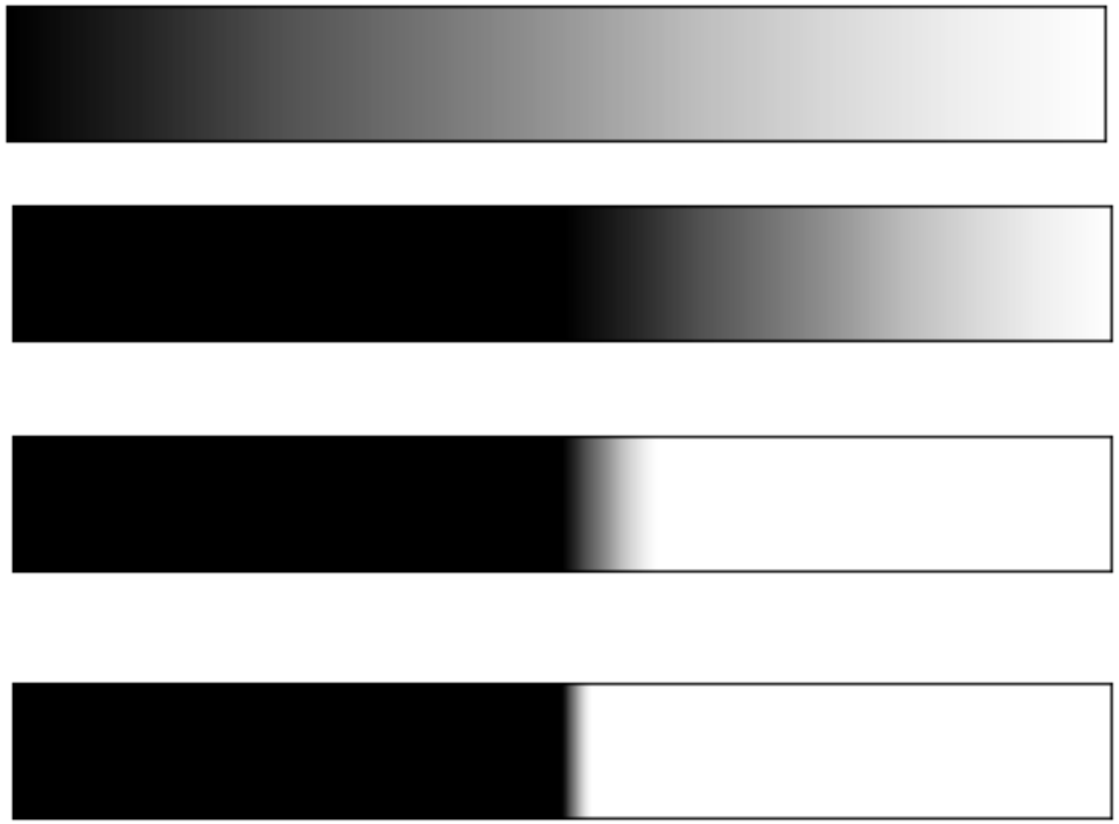
Isn’t that a bit unexpected?



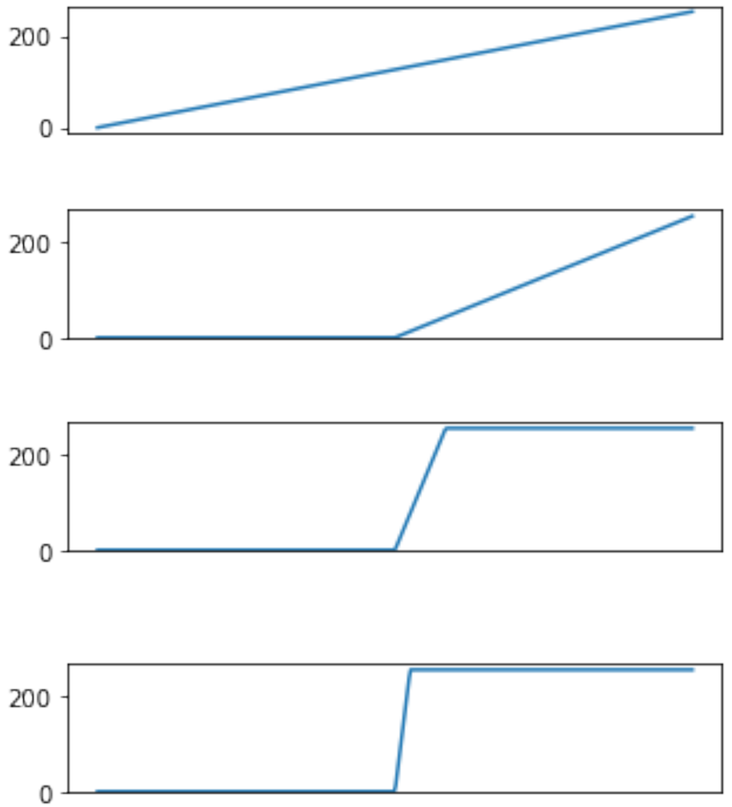
Could this be used for detecting edges somehow?

Kernel Types
Edge detection

What is an edge?



Distribution



→ The higher the slope at a given pixel, the higher the probability of an edge
→ We can get the slope by calculating the first derivative

$$\frac{\mathrm{ds}}{\mathrm{dx}} = \mathrm{s}'(x) = \lim_{\Delta x \to 0} \frac{s(x + \Delta x) - s(x)}{\Delta x}$$

→ As images have discrete integer values, we can simplify $\Delta x = 1$:

$$s(x + 1) - s(x)$$

Slide credit: Prof. Schön

Kernel Types
Edge detection

As an image is defined in two dimensions $\boldsymbol{S} = s(x, y)$, the gradient is defined by

$$\nabla s(x, y) = \begin{pmatrix} s(x+1, y) - s(x, y) \\ s(x, y+1) - s(x, y) \end{pmatrix}$$

→ This means, we can simply use difference operations for edge detections like

$\boldsymbol{H}_x$

0	0	0
0	-1	1
0	0	0

$\boldsymbol{H}_y$

0	0	0
0	-1	0
0	1	0

Slide credit: Prof. Schön

Kernel Types
Edge detection

0	0	0
0	-1	1
0	0	0

0	0	0
0	-1	0
0	1	0

→ These simple versions are very sensitive to noise, thus we often use a more robust one who takes the neighbors into account as well

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

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

Slide credit: Prof. Schön

Kernel Types

Edge detection → Sobel filter

→ A filter delivering good results, is the **Sobel Filter**

H_x

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

H_y

1	2	1
0	0	0
-1	2	-1

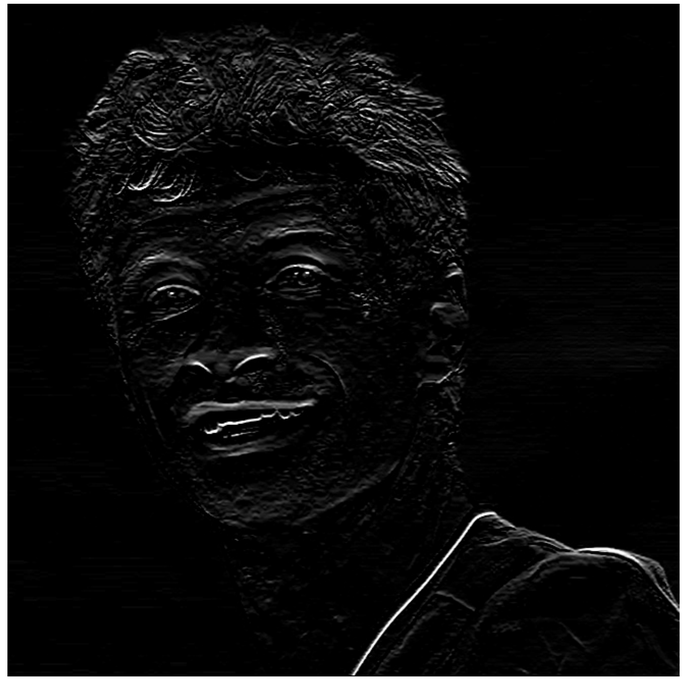
Slide credit: Prof. Schön

Kernel Types

Edge detection → Sobel filter

H_x

H_y



Slide credit: Prof. Schön

Kernel Types

Edge detection → Sobel filter

H_x

H_y

Slide credit: Prof. Schön

→ We can combine the filters in both directions to get a single edge detection result by

$$H = \sqrt{H_x^2 + H_y^2}$$

→ And the orientation of the edge by

$$\theta = \arctan(H_x, H_y)$$

Slide credit: Prof. Schön

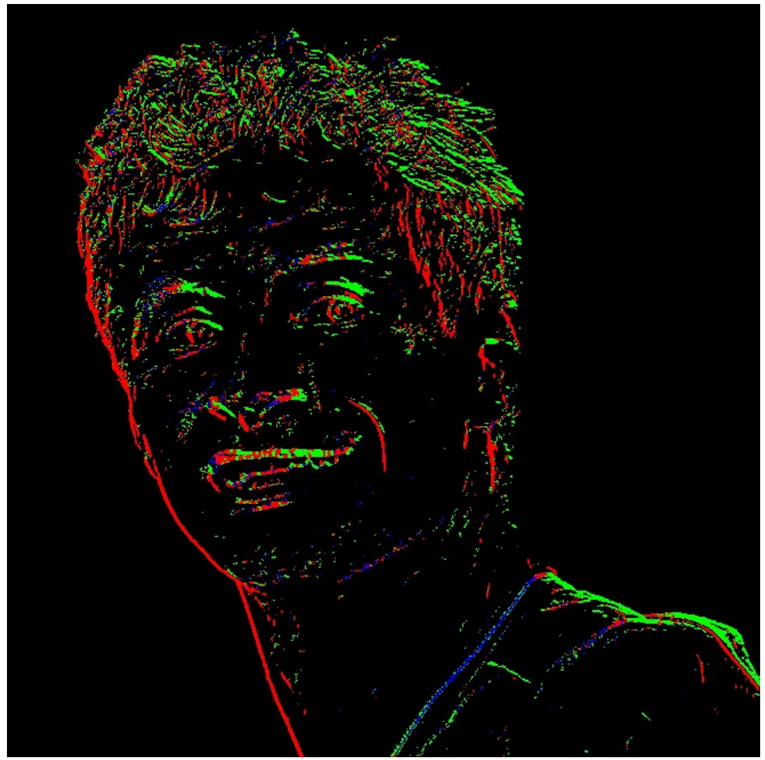
Kernel Types

Edge detection → Sobel filter

$$H = \sqrt{H_x^2 + H_y^2}$$

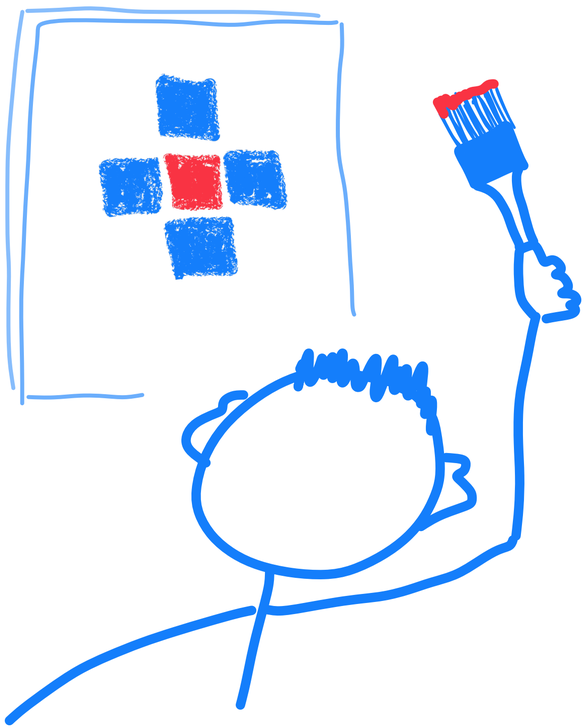
Multiplied by 5 for better visibility

$$\theta = \arctan(H_x, H_y)$$



Color coded

Slide credit: Prof. Schön

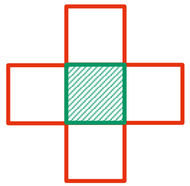


Morphology

kernels

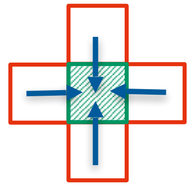
Morphology Kernels
Idea

1. Define a structuring element (SE)



2. Define a morphological operation

3. Apply it to the image as we did with filters

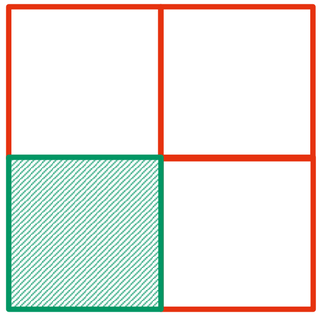


4. Set the pixel of interest to a new value based on the other pixel values within the structuring element and the morphological operation

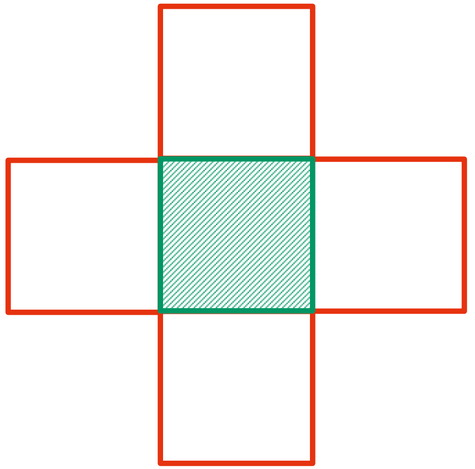
Slide credit: Prof. Schön

Morphology Kernels

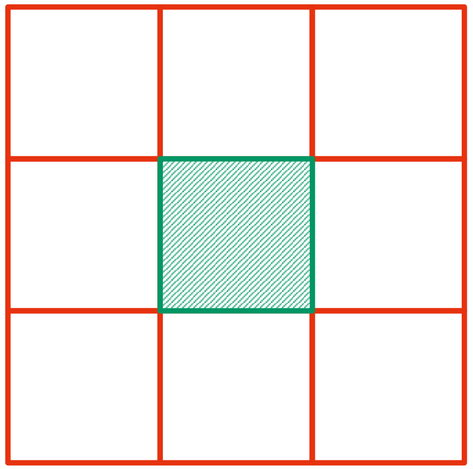
Structuring Elements: Examples



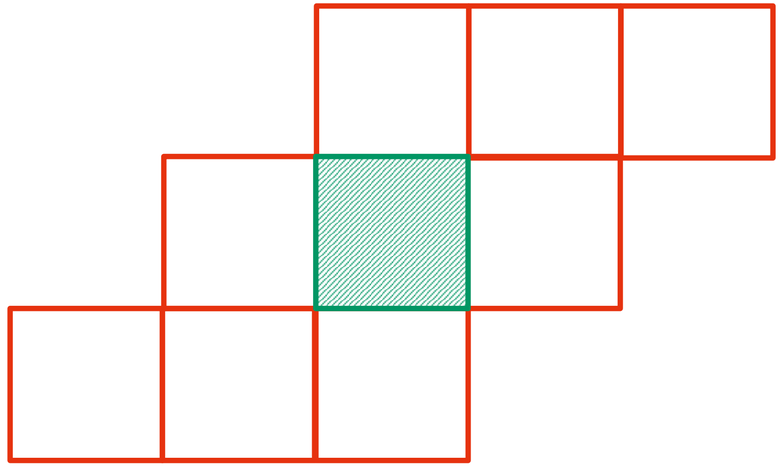
Square



Cross



8 Neighbors



diagonal

Slide credit: Prof. Schön

Morphology Kernels
Algorithm

- Let $S_{in} = (s_{in}(x, y))$ be a grayscale image
- Define structuring element, we use a Kernel K, which defines which points within the image are considered neighbors of a point (x, y)
- Let's call the point at position (x, y) *reference point*
- The kernel K is applied to the image S_e in the same way as the other filters so far
- The borders of the image are handled as we learned before
- For each position, all intensity values of all neighbors defined by the structuring element are ordered by their value and build a rank r
- The reference point is next replaced with a value $v \in r$
- Depending on how to chose v from r, we can define different morphological operations

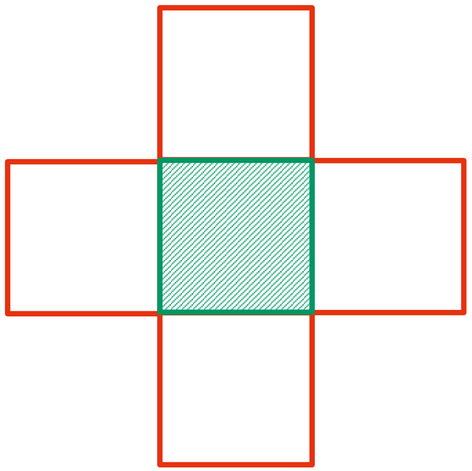
Slide credit: Prof. Schön

Morphology Kernels
Algorithm, Example:

Example Image

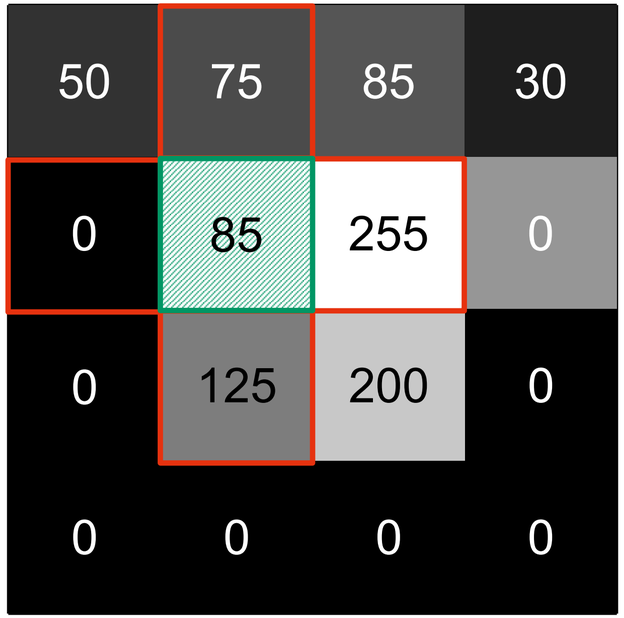
50	75	85	30
0	85	255	0
0	125	200	0
0	0	0	0

Structuring element



Slide credit: Prof. Schön

Morphology Kernels
Algorithm, Example:

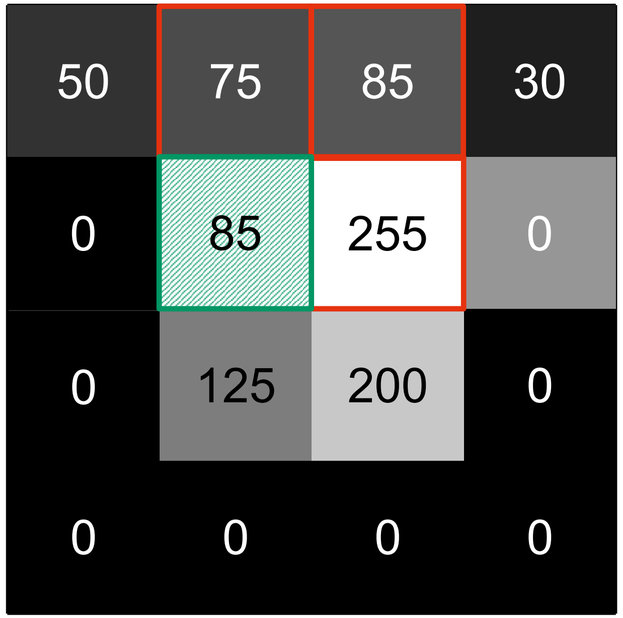


Example at position $(1,1)$:

$$r = \{0, 75, 85, 125, 255\}$$

Slide credit: Prof. Schön

Morphology Kernels
Algorithm, Example:



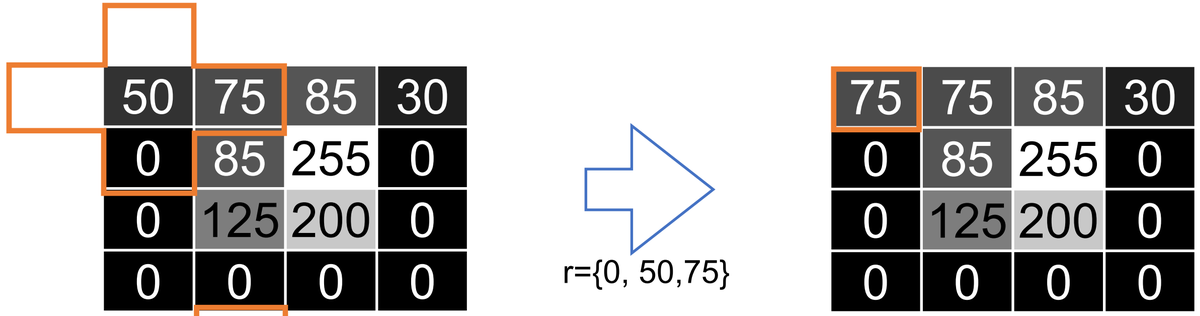
Example at position $(1,1)$:

$$r = \{75, 85, 85, 255\}$$

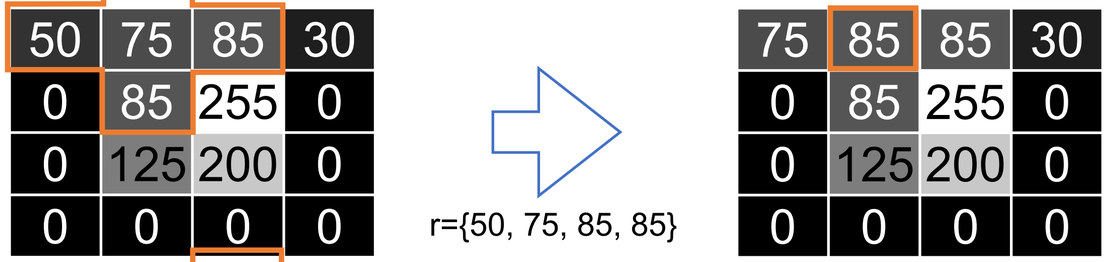
Slide credit: Prof. Schön

Example: max()

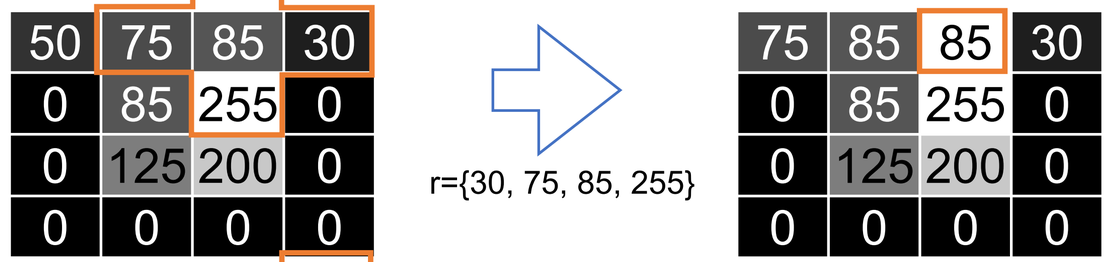
1.



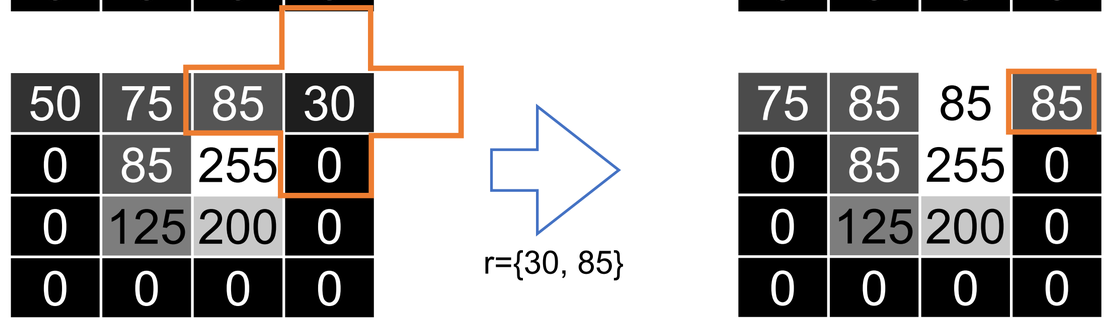
2.



3.

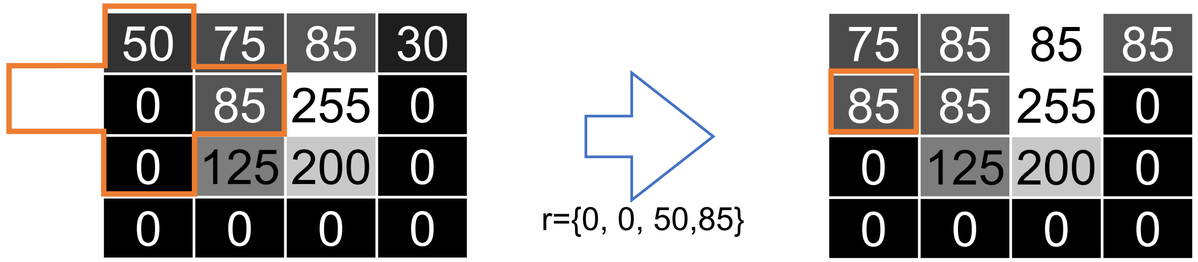


4.

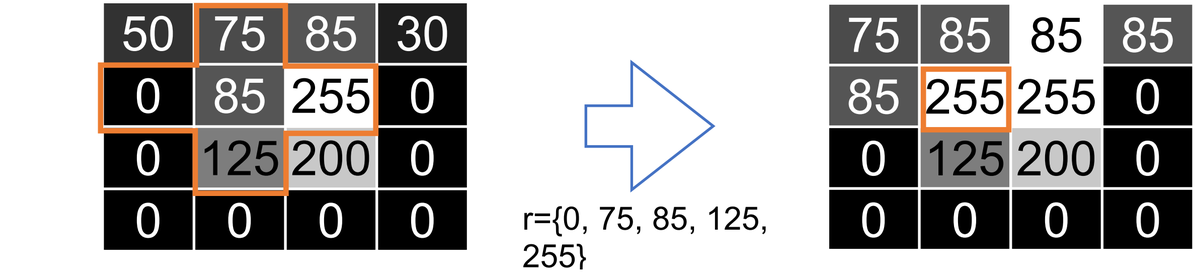


Example: max()

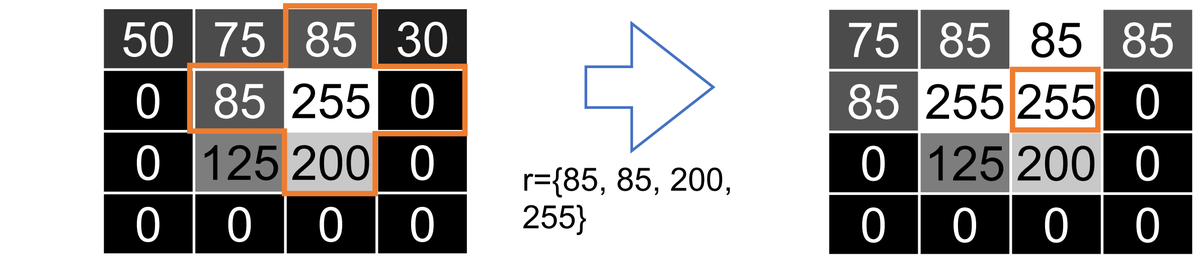
5.



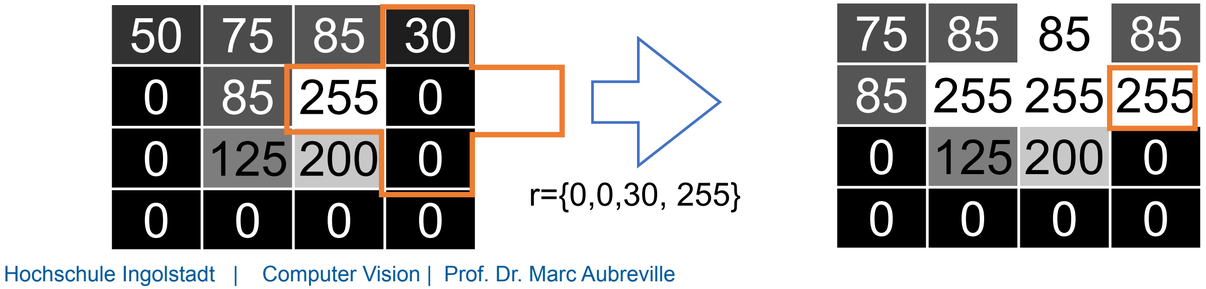
6.



7.



8.



Result after 16 steps:

75	85	85	85
85	255	255	255
125	200	255	200
0	125	200	0

Input image:

50	75	85	30
0	85	255	0
0	125	200	0
0	0	0	0

Comparing various morphological operations

Input image:

50	75	85	30
0	85	255	0
0	125	200	0
0	0	0	0

max() operation

75	85	85	85
85	255	255	255
125	200	255	200
0	125	200	0

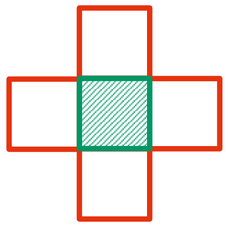
min() operation

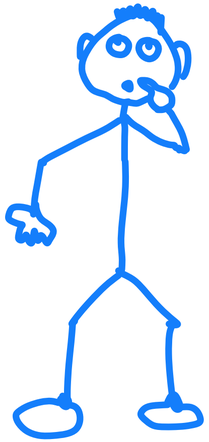
0	50	75	30
0	0	0	0
0	0	0	0
0	0	0	0

median() operation

50	80	80	85
0	85	85	15
0	85	125	0
0	0	0	0

Structural element:



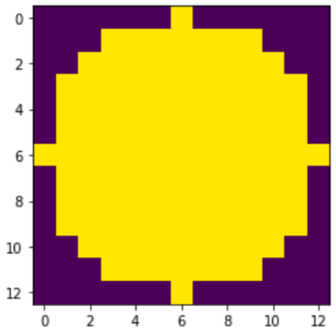


Which graphical effect can we observe from those operations?

Comparing various morphological operations

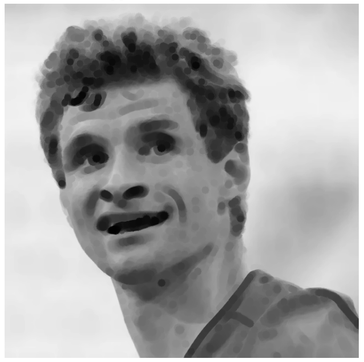
Input image:

SE:

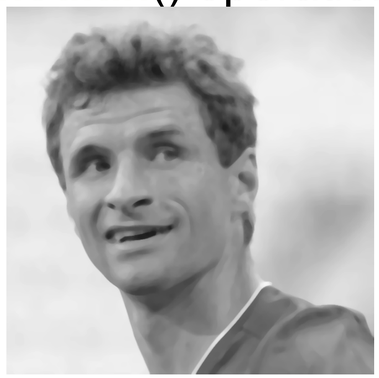


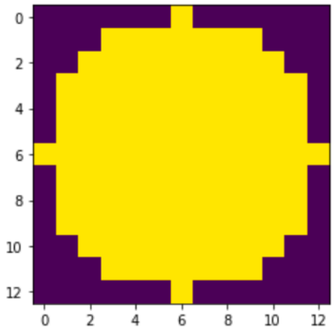
max() operation

min() operation



median() operation





Comparing various morphological operations

Bright areas are extended

Dark areas are thinned

max() operation

dilation filter

Dark areas are extended

Bright areas are thinned

min() operation

erosion filter

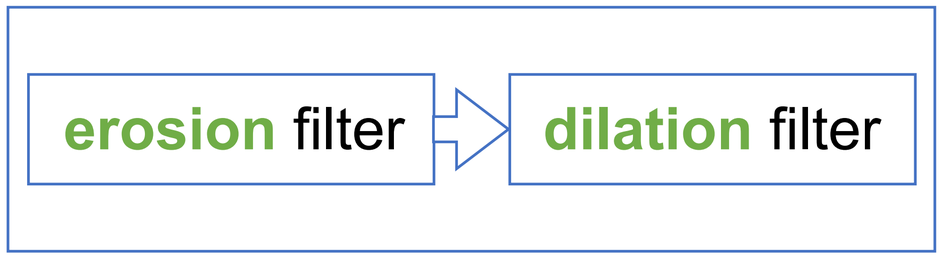
Morphology Kernels
Opening & Closing

What if we combine dilation and erosion?

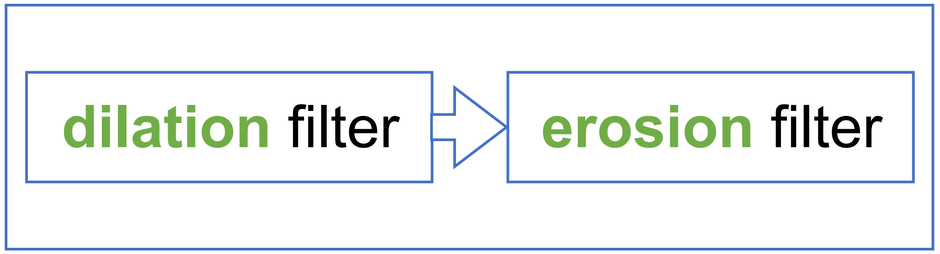
Slide credit: Prof. Schön

Combining erosion and dilation

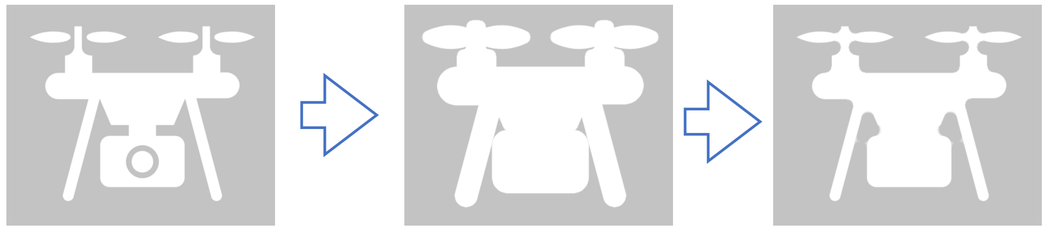
- **Erosion and dilation filters can be applied after each other.**



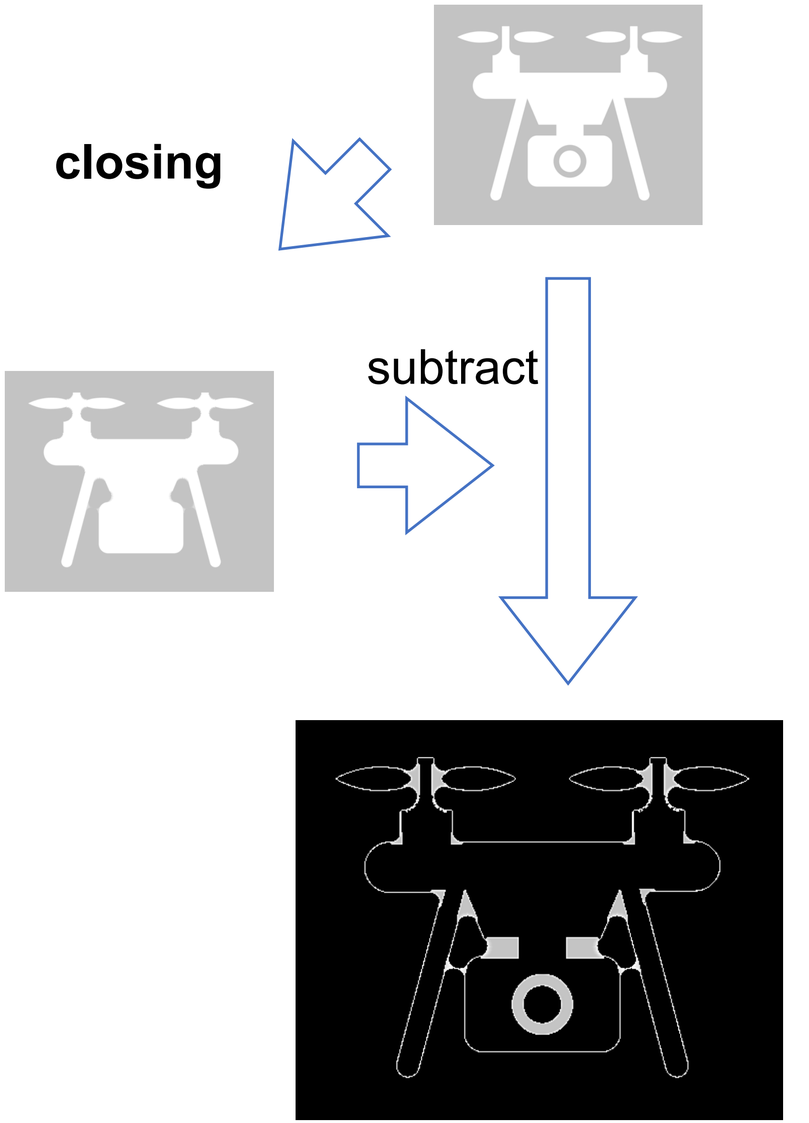
opening filter



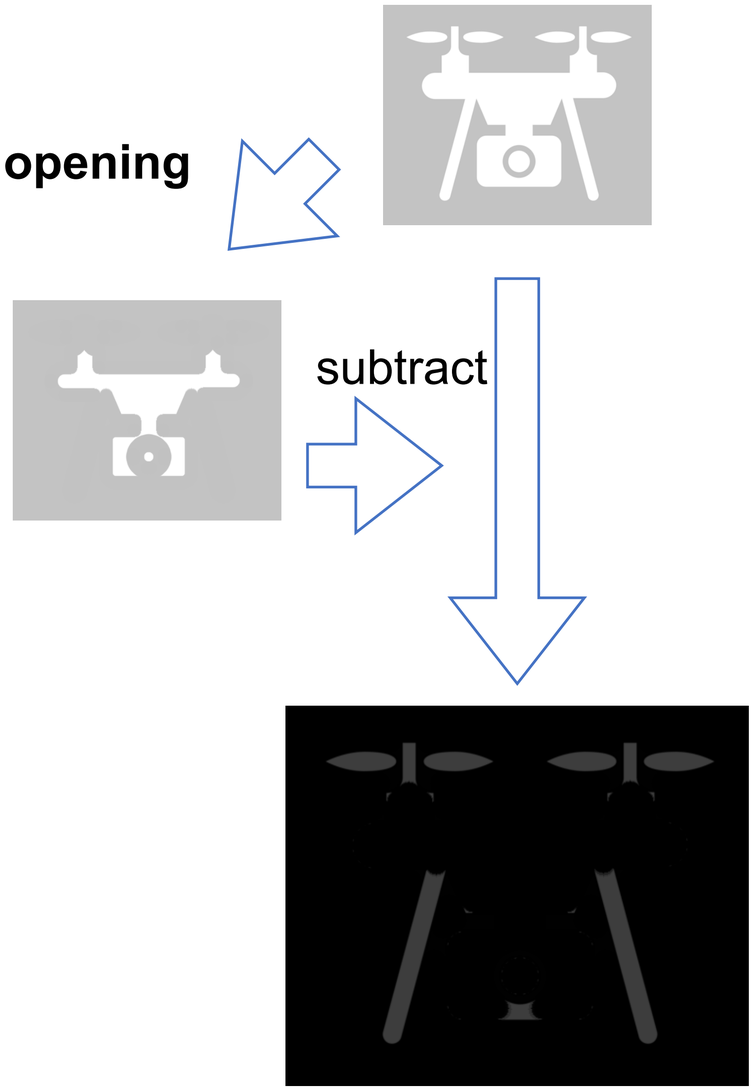
closing filter



Let's have a bit more fun with opening and closing



hello
world

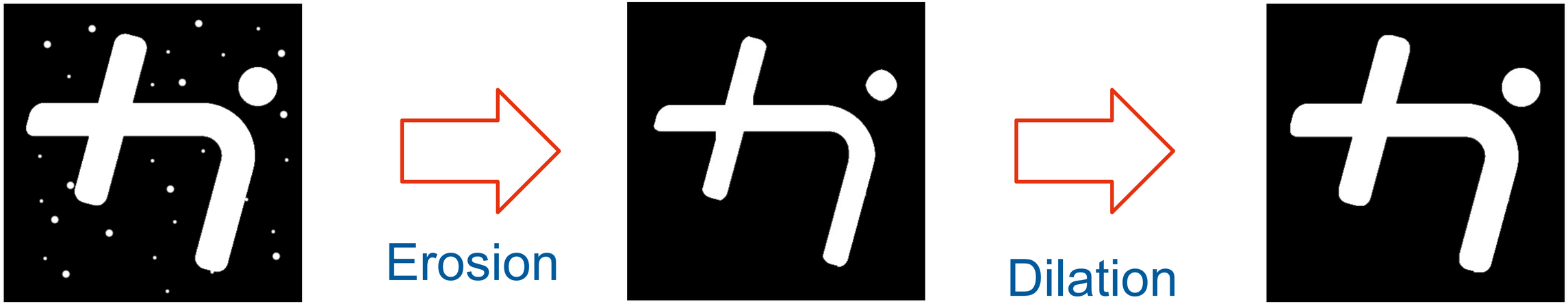


black tophat filter

white tophat filter

Yay, so we have a new tool! What could we use this for?

Morphology Kernels
Opening



➔ Opening: removing noise

Slide credit: Prof. Schön

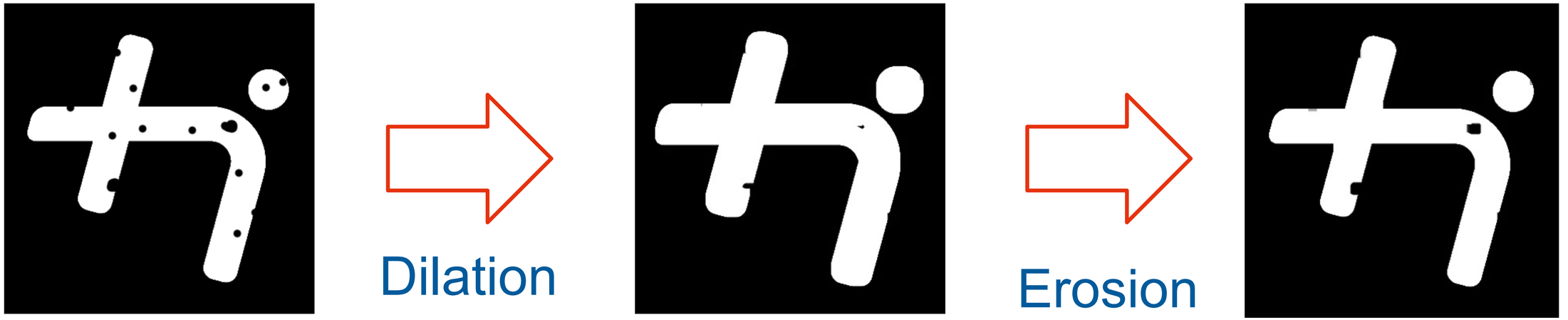
Morphology Kernels
Closing

➔ Closing: closes small holes in foreground objects

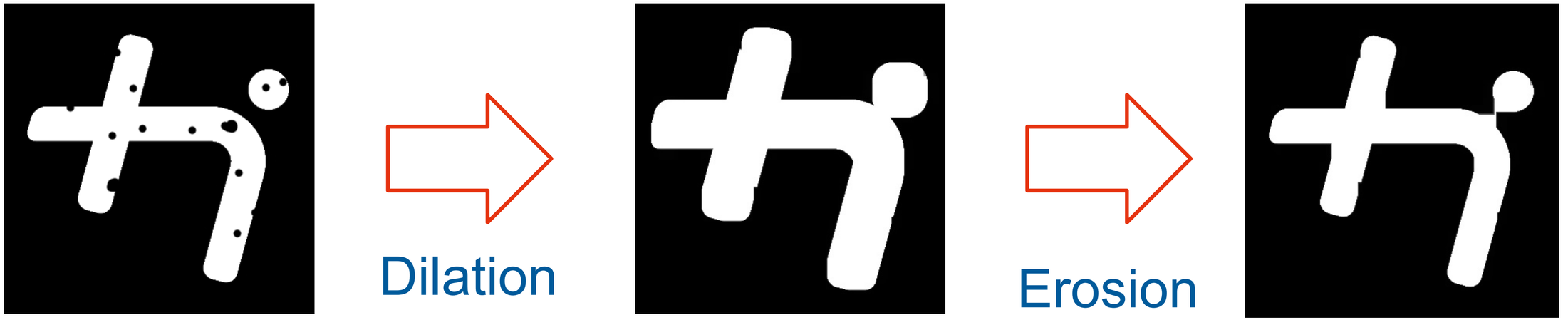
Slide credit: Prof. Schön

Morphology Kernels
Closing

K: 10x10



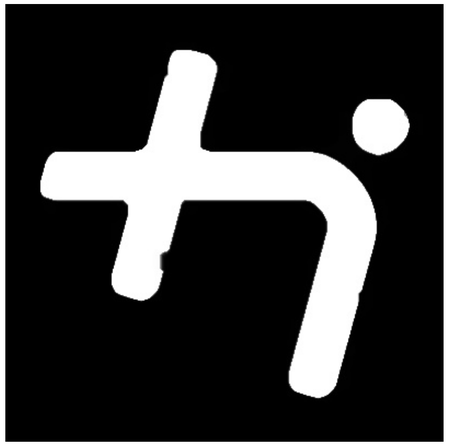
K: 20x20



Slide credit: Prof. Schön

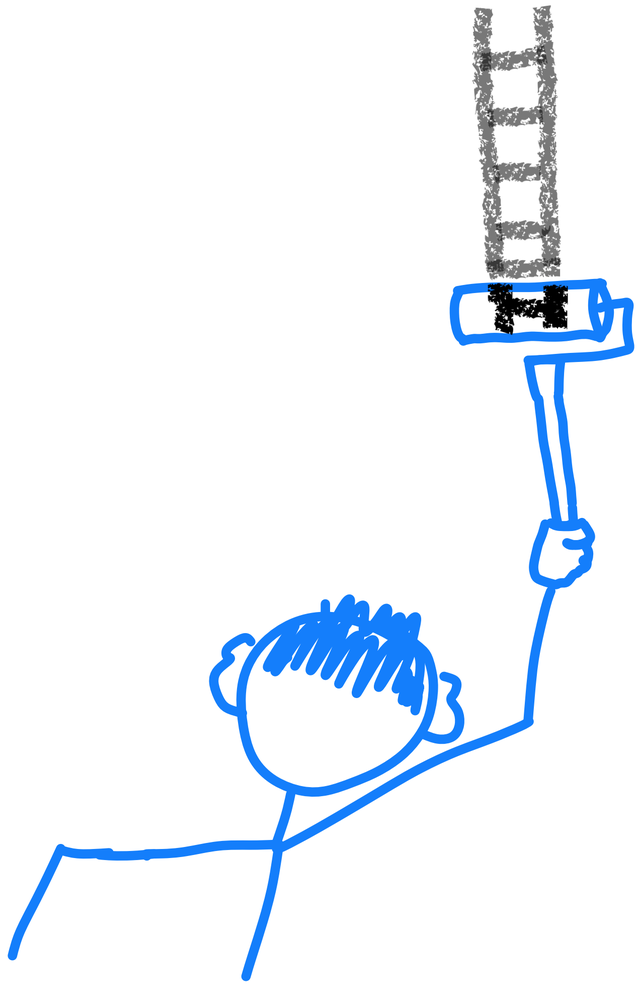
Morphology Kernels
Closing

- Options to get better result:
  - Try different structured elements
  - Try different kernel sizes
  - Try different number of iterations for dilate and erode



Example Solution:
Ellipse kernel 10x10
2 dilate followed by 2 erode

Slide credit: Prof. Schön

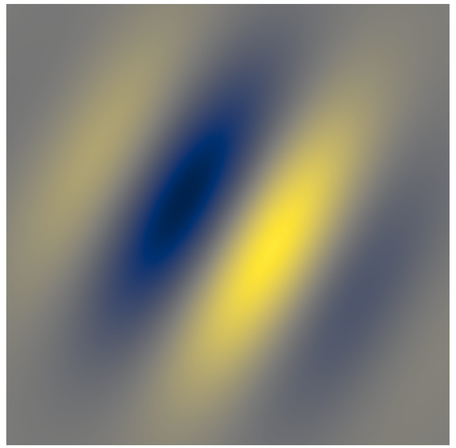


Texture

kernels

Texture Kernels
Gabor Filter

- Named after *Dennis Gabor*
- Used for texture analysis
- Localizes a signal in both, space and frequency domain
- Analyzes for specific frequency content in specific regions around the pixel of interest
- Particularly appropriate for texture representation and distinction

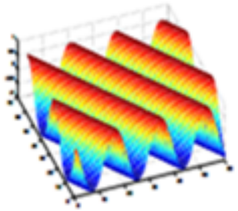


Quelle:
https://en.wikipedia.org/wiki/Gabor_filter

Slide credit: Prof. Schön

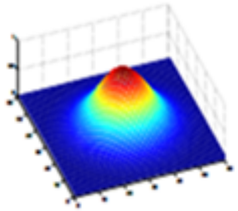
Texture Kernels
Gabor Filter

- In the spatial domain, a 2D Gabor filter is a **sinusoidal plane wave** modulated with a **Gaussian kernel function**.
- It is thus some special kind of band pass filter



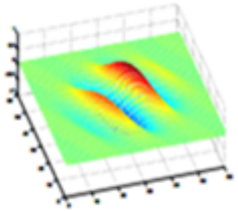
A Sinusoid oriented 30^0 with X-axis

(a)



A 2-D Gaussian

(b)



The corresponding 2-D Gabor filter

(c)

Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

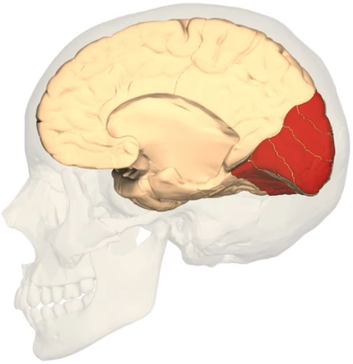
Slide credit: Prof. Schön

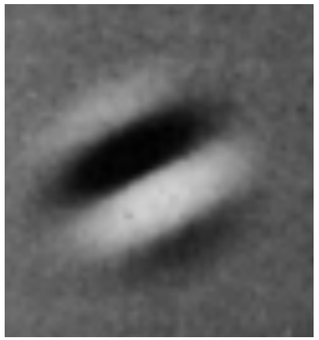
Texture Kernels
Gabor Filter

- Some papers have shown that some simple cells in the visual cortex of mammalian brains can be modelled by Gabor filters [1][2][3]

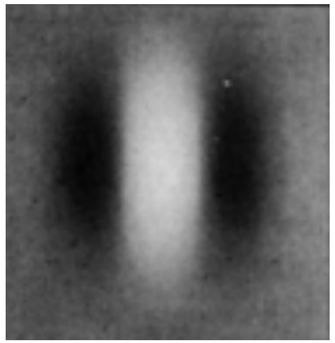
- Gabor filters are thus often related to human perception

- Gabor filters are very powerful and can detect textures as well as edges in many different orientations and scales





Neuron #1 of visual cortex (model)



Neuron #2 of visual cortex (model)

http://viscomp.csail.mit.edu/resource/slides/lecture5.pptx

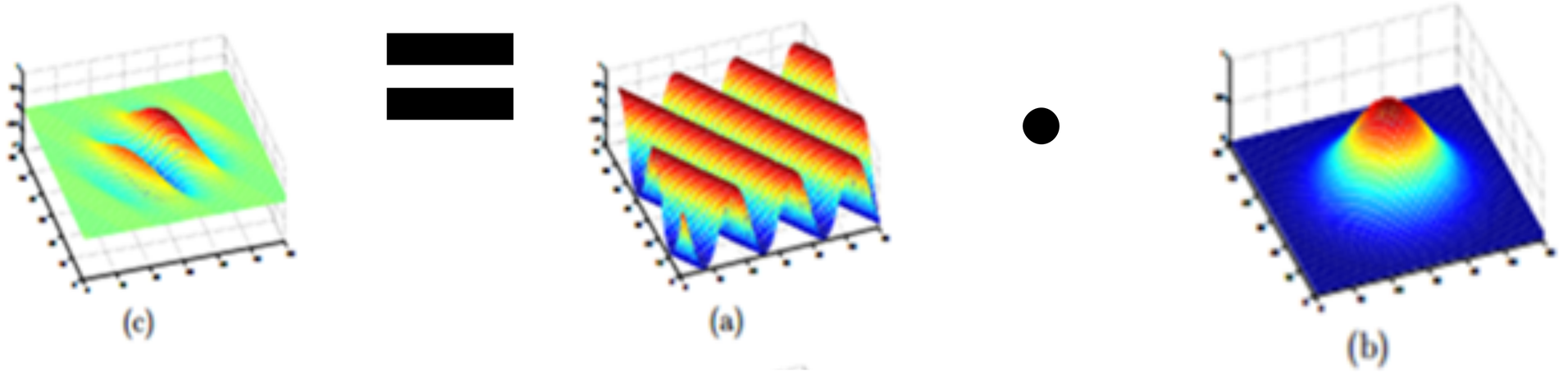
[1] *Olshausen, B. A. & Field, D. J. (1996). "Emergence of simple-cell receptive-field properties by learning a sparse code for natural images". Nature.* ***381*** *(6583): 607–609*
[2] *Marčelja, S. (1980). "Mathematical description of the responses of simple cortical cells". Journal of the Optical Society of America.* ***70*** *(11): 1297–1300*
[3] Daugman, John G. (1985-07-01). "Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters". Journal of the Optical Society of America A. 2 (7): 1160–9

Slide credit: Prof. Schön

Gabor Filter

Sinusoidal plane wave

Gaussian function



Slide credit: Prof. Schön

- The Gabor filter consists of a **real** and a **complex** component and is defined by:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

where

$$x' = x\cos\theta + y\sin\theta \quad y' = -x\sin\theta + y\cos\theta$$

Slide credit: Prof. Schön

- The **real** and a **complex** component can also be used individually and are then given by

Real:
$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

Imaginary:
$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\sin\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

Slide credit: Prof. Schön

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

What do the parameters mean?

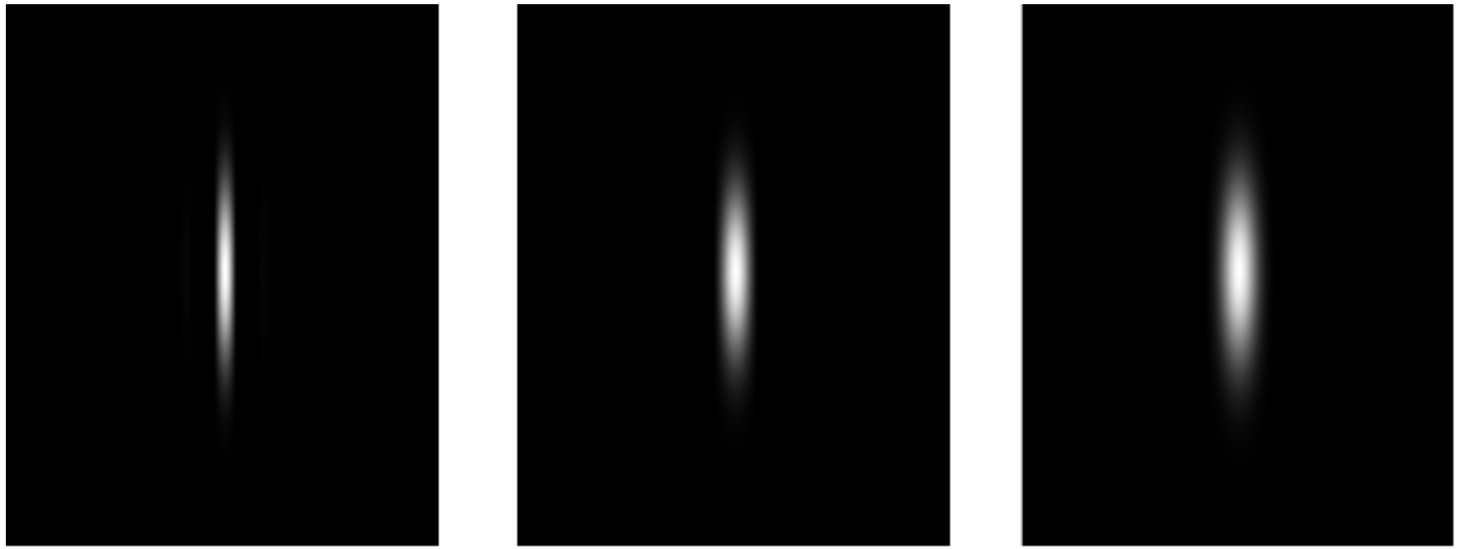
- λ: Wavelength of the sinusoidal component
- θ: Orientation of the normal to the parallel stripes
- ψ: Phase offset of the sinusoidal function
- σ: Standard deviation of the Gaussian
- γ: Spatial aspect ratio → ellipticity

Let's dig into those parameters now …

Slide credit: Prof. Schön

λ : Wavelength of the sinusoidal component

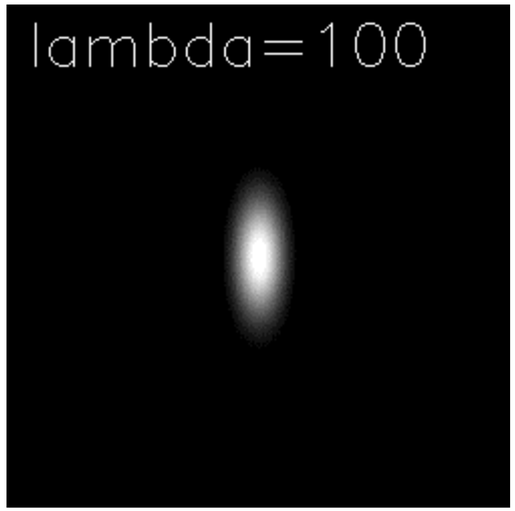
The wavelength influences the width of the peaks in the sinus signal, thus increasing λ lead to thicker stripes, decreasing λ leads to thinner stripes



Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

Slide credit: Prof. Schön

λ: Wavelength of the sinusoidal component

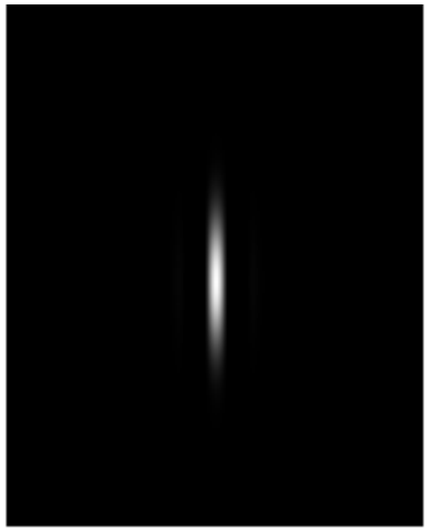


θ Theta = 0
ψ Psi = 0
σ Sigma = 10
γ Gamma = 0.5

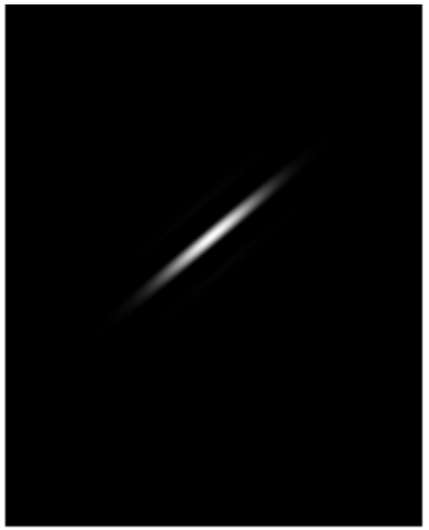
Slide credit: Prof. Schön

θ: Orientation of the normal to the parallel stripes

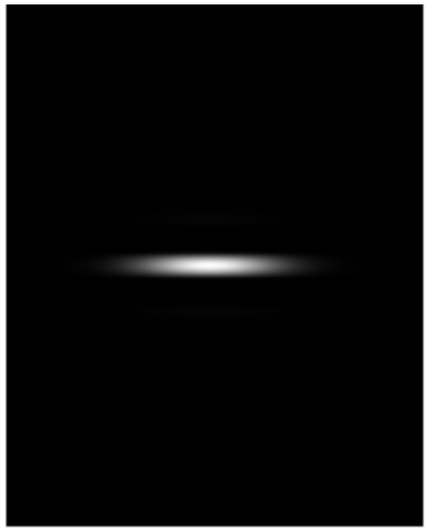
Controls the orientation of the stripes, zero means vertical position



Theta (Θ) = 0



Theta (Θ) = 45



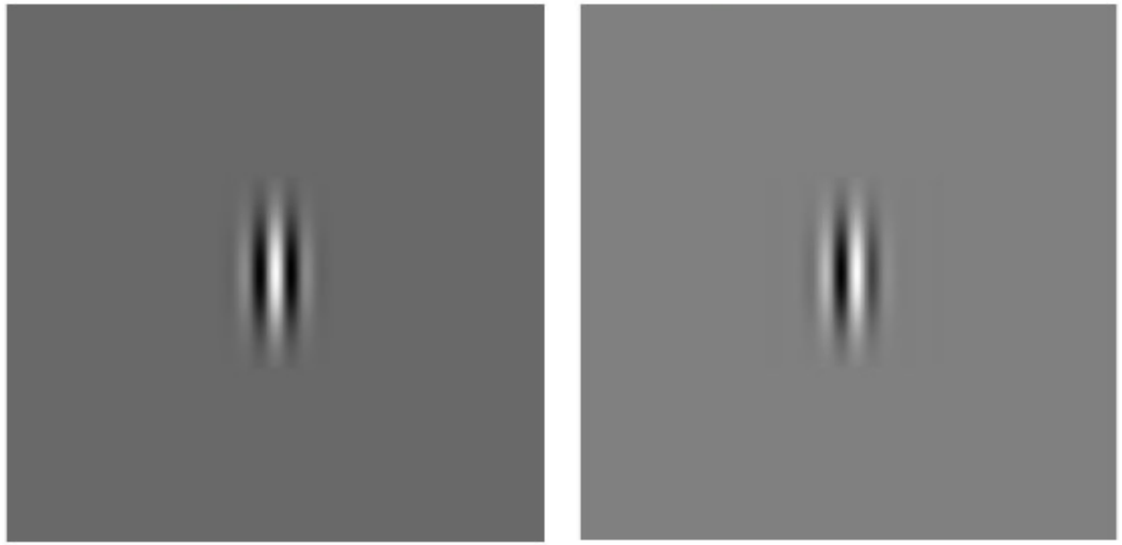
Theta (Θ) = 90

Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

Slide credit: Prof. Schön

ψ: Phase offset of the sinusoidal function

Controls offset of the phase and thus the symmetry of the filter



Phase offset = 0
(symmetric function)

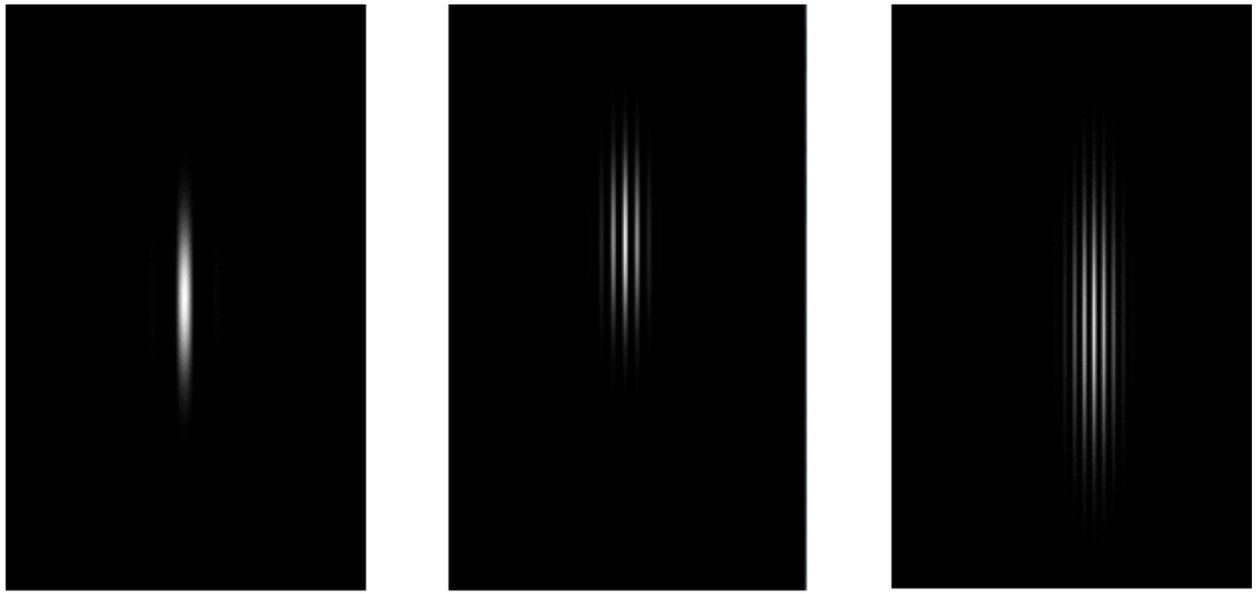
Phase offset = -90
(anti-symmetric function)

Quelle: https://slidetodoc.com/2-d-gabor-functions-for-image-processing-and/

Slide credit: Prof. Schön

σ: Standard deviation of the Gaussian

As the standard deviation increases, it gets more broad and thus keeps more of the sinusoidal peaks activated, which leads to more stripes

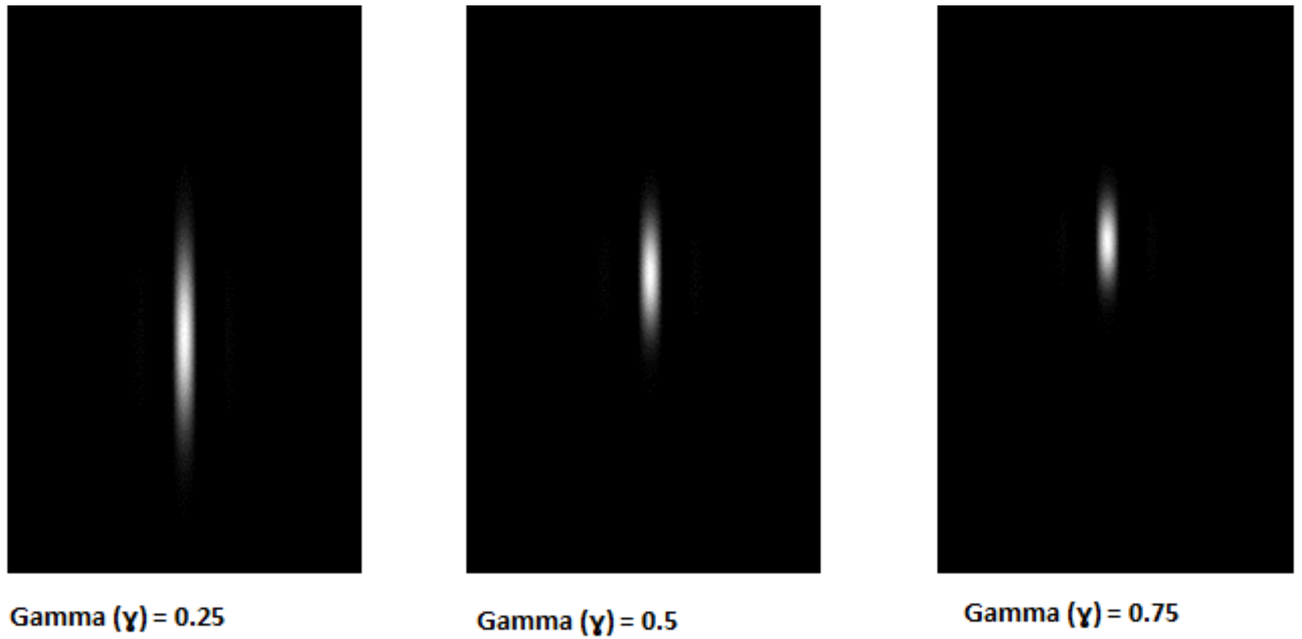


Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

Slide credit: Prof. Schön

γ: Spatial aspect ratio (of the x- and y-axis of the Gaussian ellipse)

Controls the height of the Gabor filter where increasing γ leads to a smaller height and vise versa.

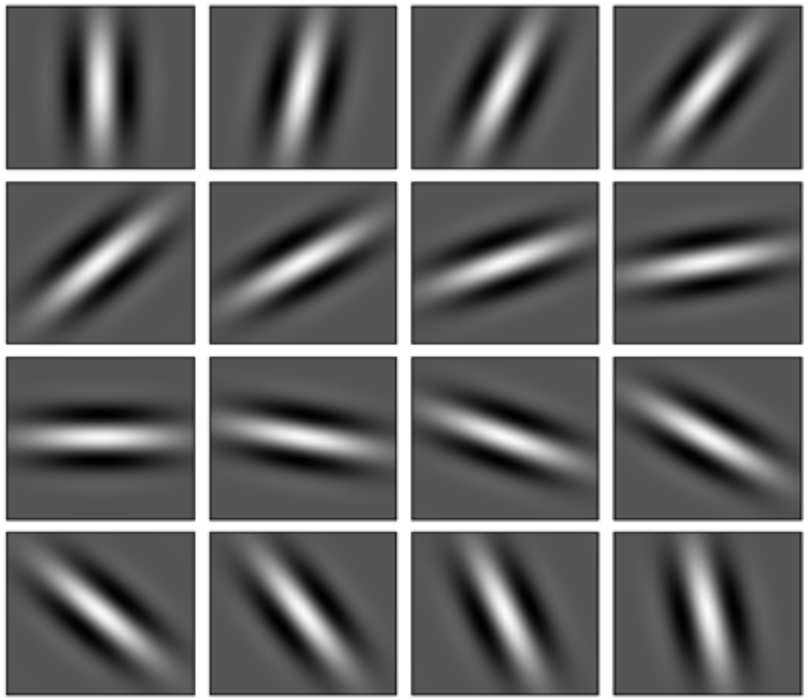


Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

Slide credit: Prof. Schön

Texture Kernels
Gabor Filter

- As differently parameterized Gabor filters detect different patters in different orientation, usually a whole bank of filters is applied to the image



A bank of 16 Gabor filters with varying orientation

Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

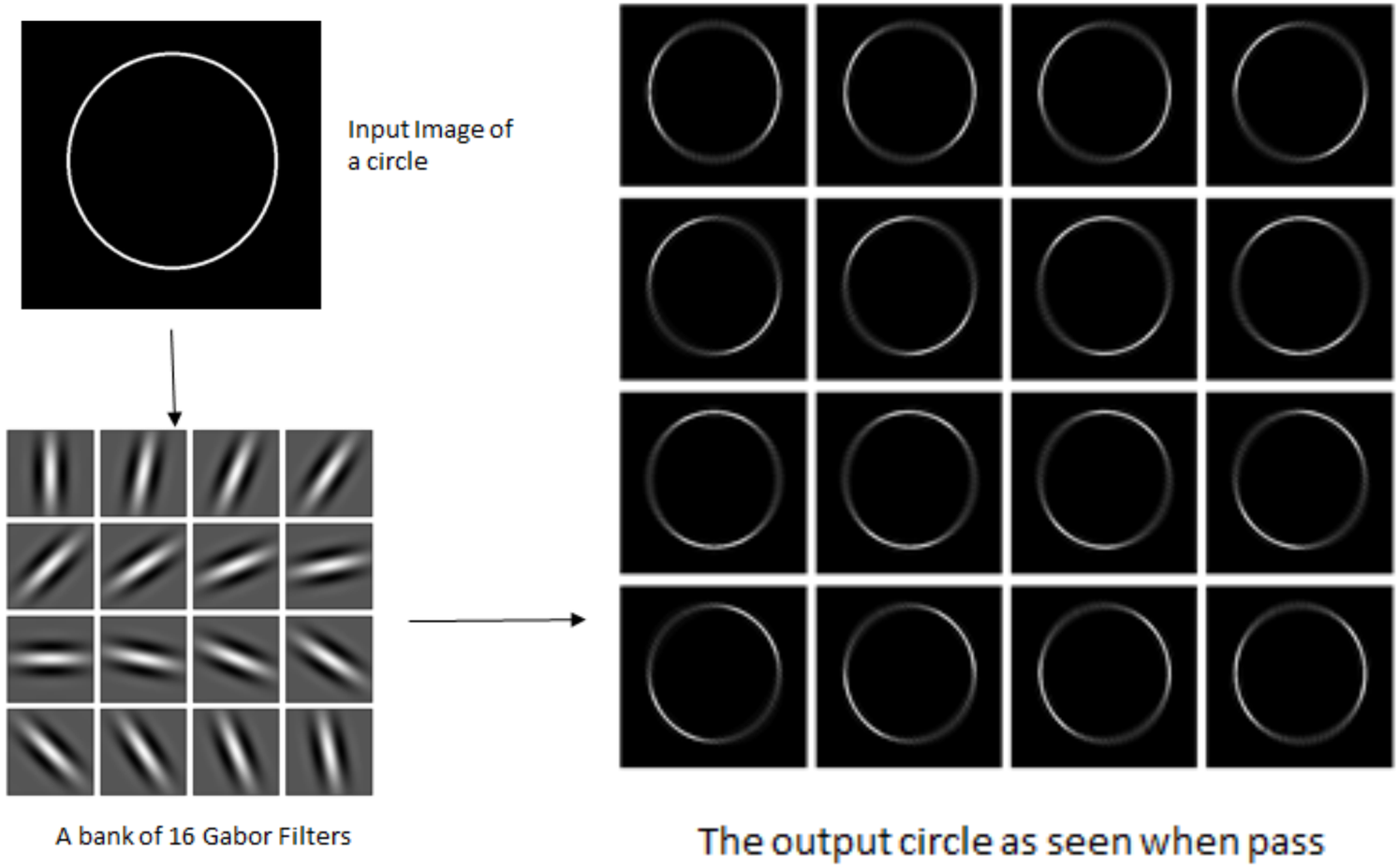
Slide credit: Prof. Schön

Let’s apply that now!

Texture Kernels

Gabor Filter

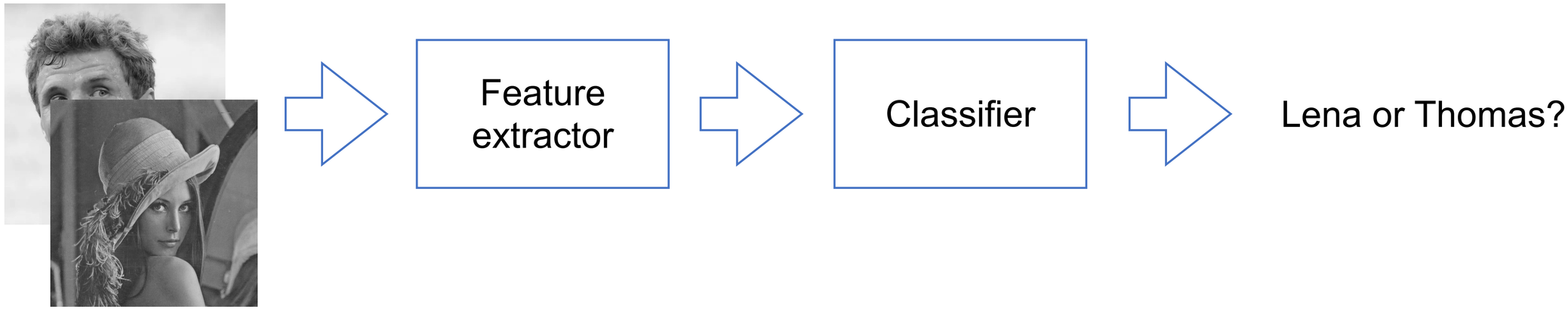
Application to a circle example



Quelle: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

Slide credit: Prof. Schön

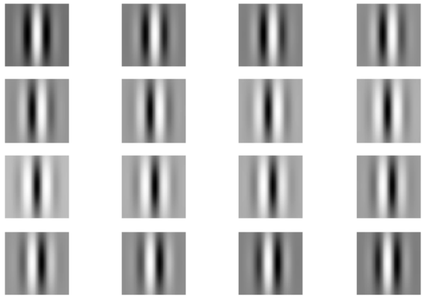
We ultimately want to recognize something from this image.



- **We thus need meaningful features that explain the variance of the classes „Thomas“ and „Lena“ and allow a decision using some classifier.**
- **It’s often being perceived that the choice of classifier is the most important in this problem.**
- **But in reality, the selection of proper features takes up 80% of the time.**

Let's extract some features from Thomas

Gabor filters (kernel size: 31)

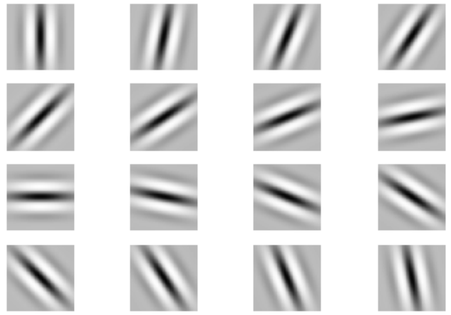


$$\psi \in \left[0, \frac{\pi}{8}, \ldots \frac{15\pi}{8}\right]$$

$$\theta = 0, \gamma = 0.5, \lambda = 10, \sigma = 5$$

Let's extract some features from Thomas

Gabor filters (kernel size: 31)



$$\theta \in \left[0, \frac{\pi}{8}, \ldots \frac{15\pi}{8}\right]$$

$$\gamma = 0.5, \lambda = 10, \psi = \pi, \sigma = 5$$

Eyes / lips highlighted: Might be a good feature

Texture Kernels
Gabor Filter

- Finding the right bank of Gabor filters is a hard and time consuming task
- Has to be fine tuned for different kind of images and problems
- You will get a chance to make this experience in the practical course

Wouldn’t it be nice to have some tool which is creating them for us automatically?

Slide credit: Prof. Schön