

# Image Processing: Filters

Marcelo Leomil Zoccoler

With material from

Robert Haase, Pol; Mauricio Rocha Martins, Norden lab, MPI CBG; Dominic Waithe, Oxford University; Benoit Lombardot, Scientific Computing Facility, MPI CBG; Alex Bird, Dan White, MPI CBG

---

October 2022

An image processing filter is an operation on an image.  
It takes an image and produces a new image out of it.  
Filters change pixel values.

There is no “best” filter. Which filter fits your needs, depends on the context.  
Filters do not do magic. They can not make things visible which are not in the image.

## Application examples

- Noise-reduction
- Artefact-removal
- Contrast enhancement
- Correct uneven illumination

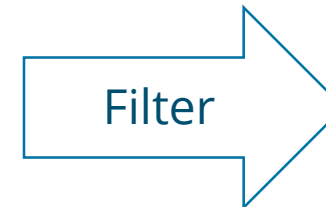
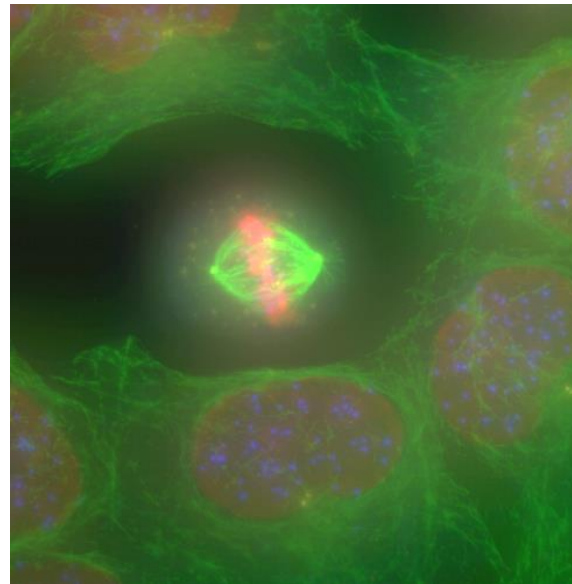
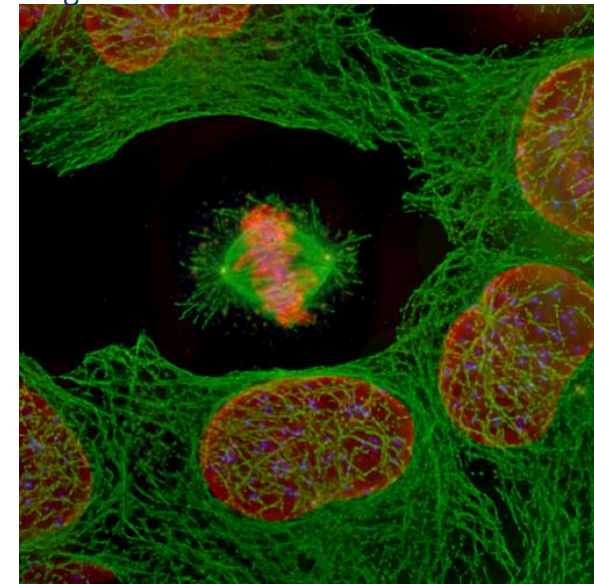


Image source: Alex Bird / Dan White MPI CBG



# Effects harming image quality

Noise is a general term for unwanted modifications that a signal may suffer during capture, storage, transmission, processing, or conversion.

In microscopy, image quality suffers from

- shot noise: Statistical variation of the photons arriving at the camera
- dark noise: Statistical variation of how many electrons are generated if a photon arrives in a camera pixel (temperature dependent).
- read out noise: introduced by the electronics, especially the analog-digital-converter
- Physical/optical effects: aberrations, defocus
- Biological/physiological/structural effects: motion, diffusion

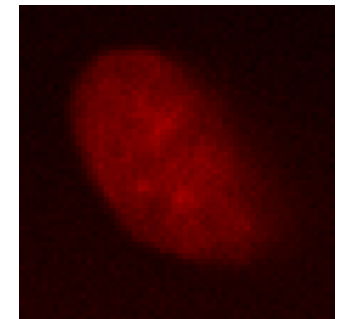
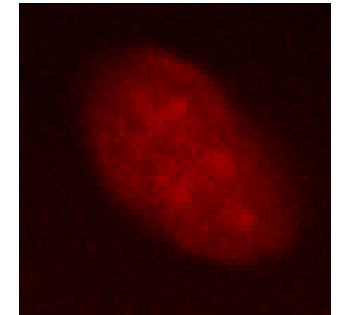
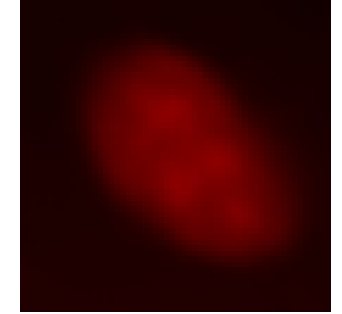


Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

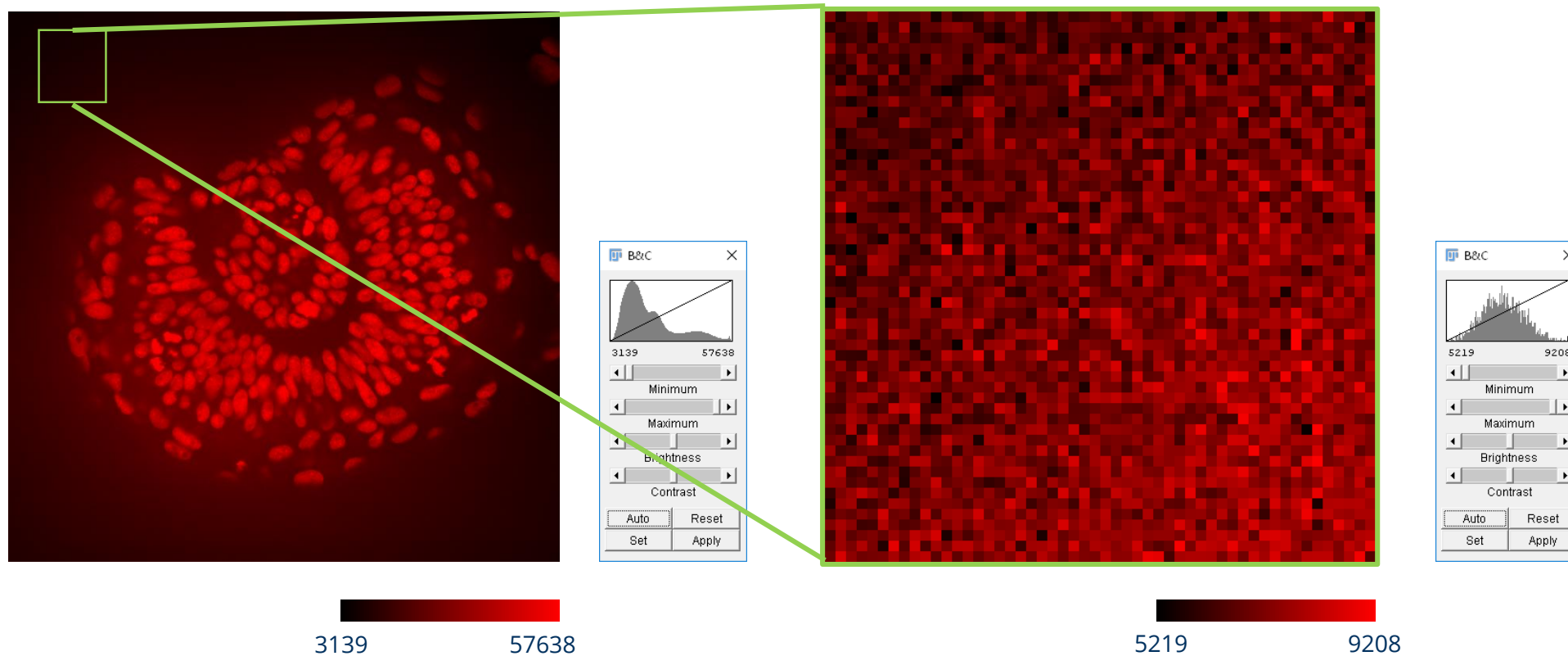
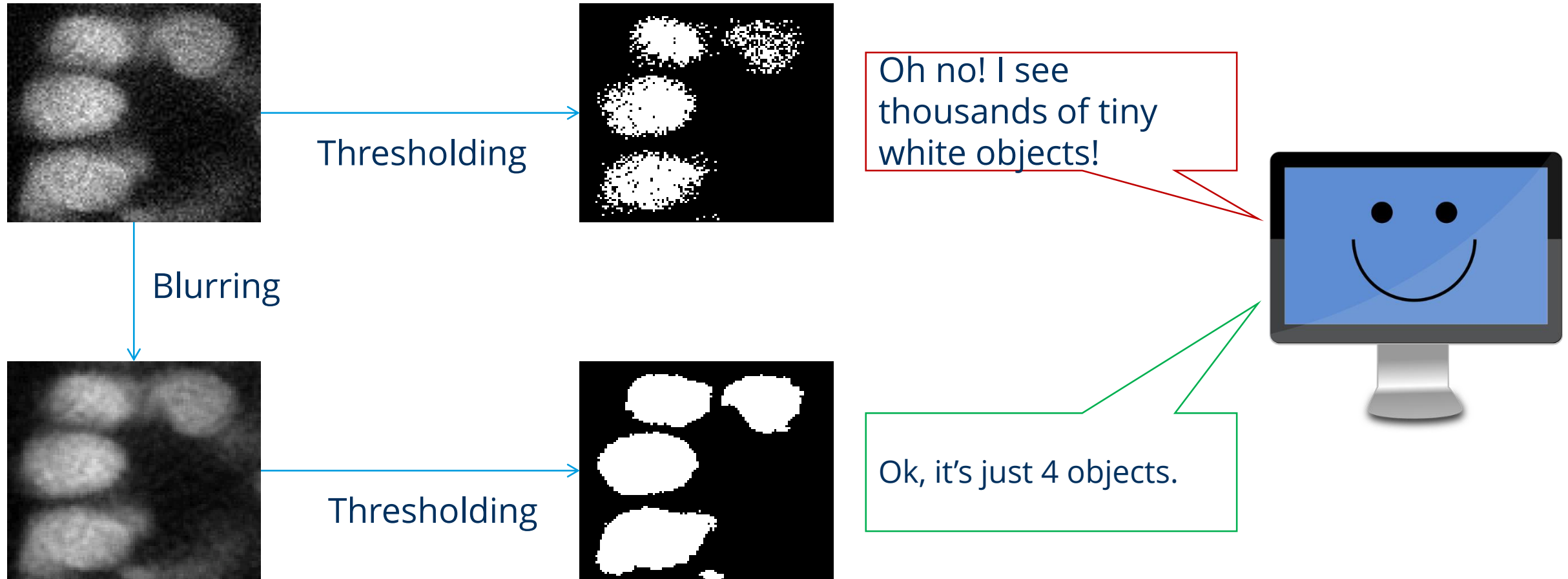


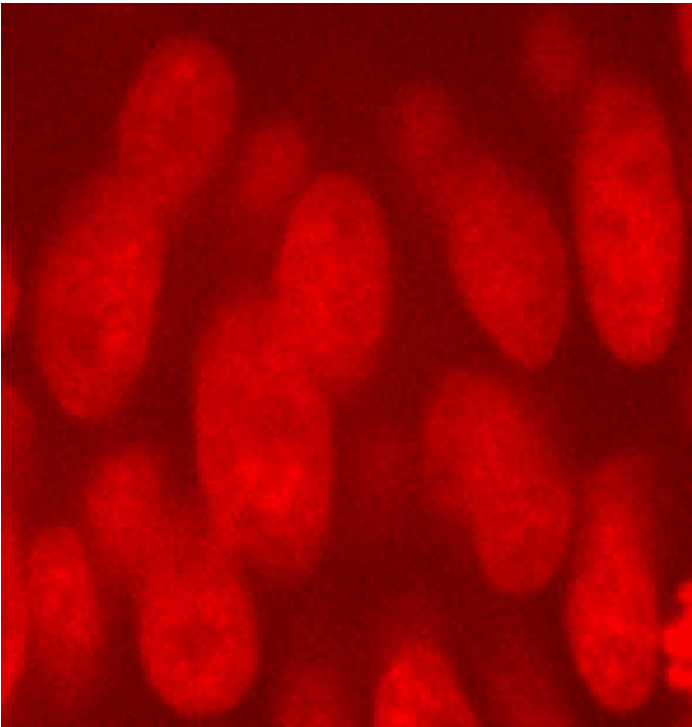
Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

# Image correction / noise removal

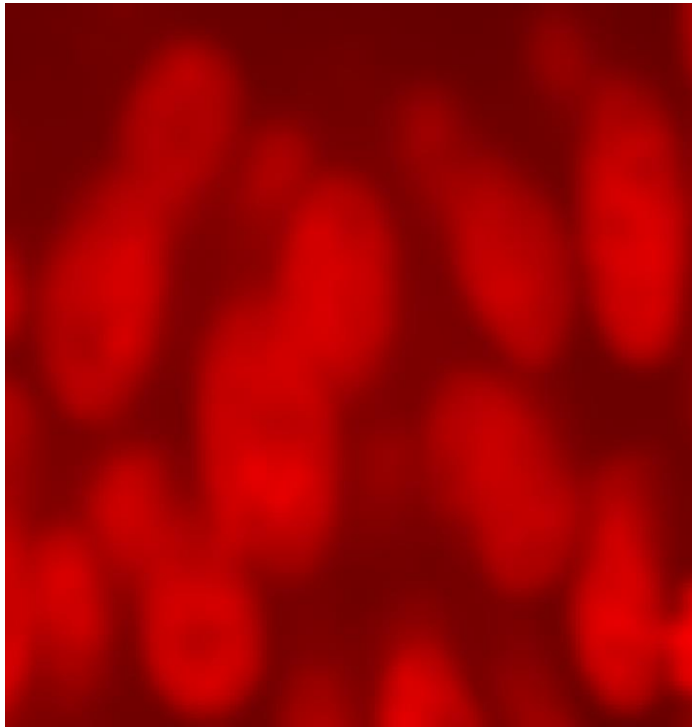
We need to remove the noise to help the computer *interpreting* the image



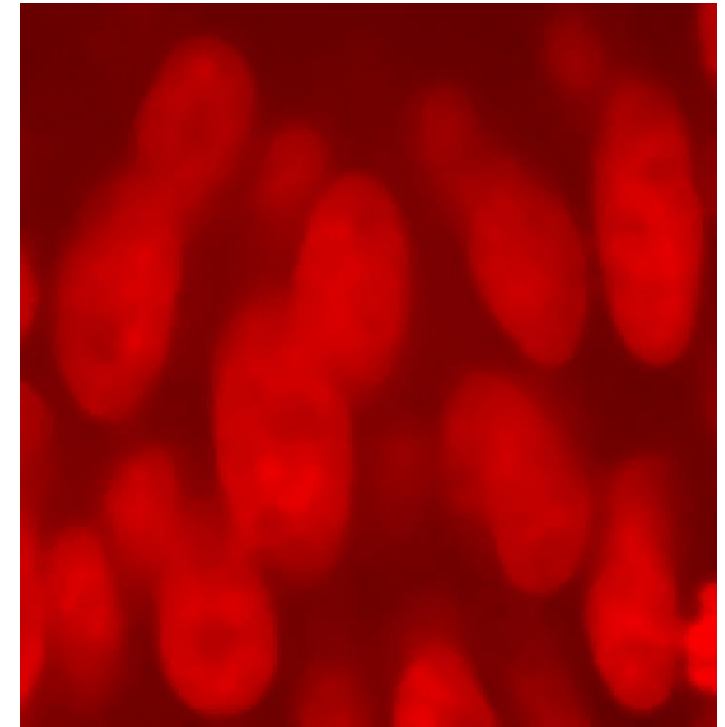
Filters can remove noise



Original image



Gaussian blur filtered



Median filtered

Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

- Linear filters replace each pixel value with a weighted sum of pixels

- Kernels are matrices describing a linear filter

Mean filter 3x3 kernel =

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

Kernel

32	48	64	96	120	128	144	152	152
40	72	96	128	160	176	184	184	184
72	104	136	160	176	184	192	192	184
104	136	168	184	192	200	200	192	184
136	160	184	184	192	192	192	184	184
168	184	192	192	184	184	176	176	176
192	192	192	192	184	184	176	176	176
208	200	184	184	184	184	176	176	168

$$\text{New\_value} = (1/9) \cdot 32 + (1/9) \cdot 48 + (1/9) \cdot 64 + (1/9) \cdot 40 + (1/9) \cdot 72 + (1/9) \cdot 96 + (1/9) \cdot 72 + (1/9) \cdot 104 + (1/9) \cdot 136$$

$$\text{New\_value} = 73.8$$

What operation is this?

- Linear filters replace each pixel value with pixels

- Kernels are matrices describing a linear filter

Mean filter 3x3 kernel =

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

Kernel

32	48	64	96	120	128	144	152	152
40	72	96	128	160	176	184	184	184
72	104	136	160	176	184	192	192	184
104	136	168	184	192	200	200	192	184
136	160	184	184	192	192	192	184	184
168	184	192	192	184	184	176	176	176
192	192	192	192	184	184	176	176	176
208	200	184	184	184	184	176	176	168

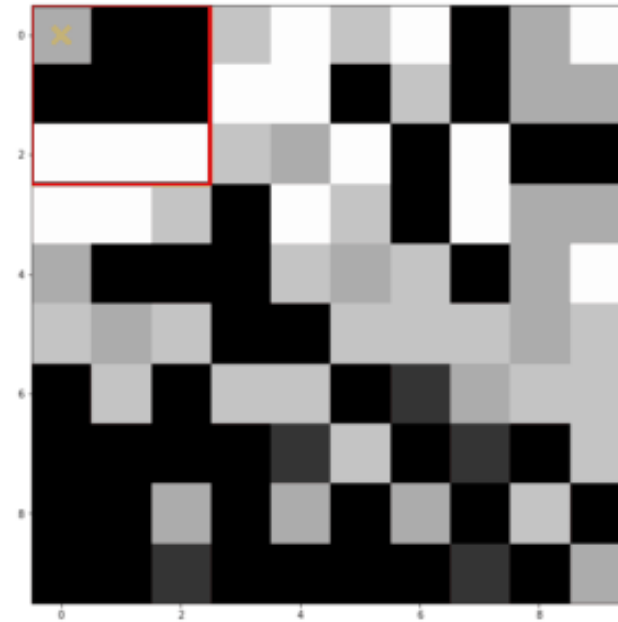
$$\text{New\_value} = (1/9)*48 + (1/9)*64 + (1/9)*96 + (1/9)*72 + (1/9)*96 + (1/9)*128 + (1/9)*104 + (1/9)*136 + (1/9)*160$$

$$\text{New\_value} = 100.4$$

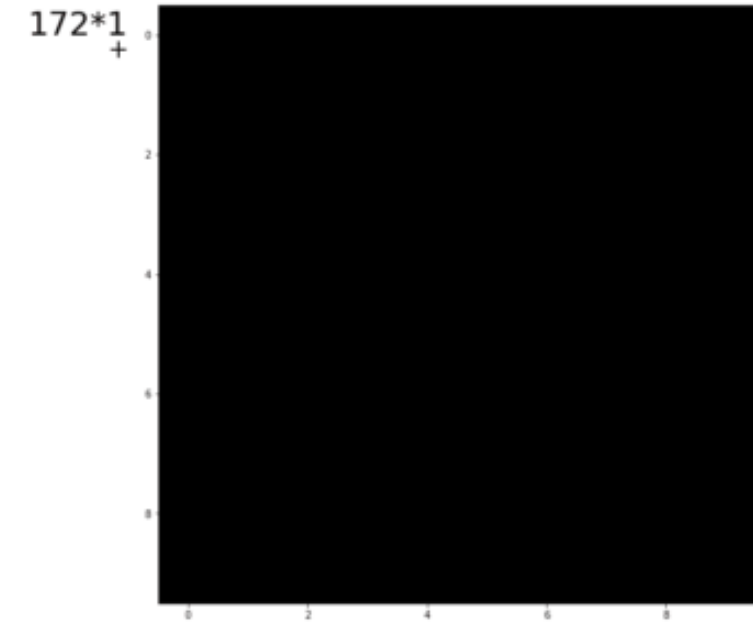
What operation is this?



- Basically, linear filtering is a Convolution  
Convolution in 2D
- It needs a kernel (weight template)
- Result: new image where each pixel is replaced by the weighted sum of pixel values in the neighborhood



Image



Output\_image

Terminology:

- “We convolve an image with a kernel.”
- Convolution operator: \*

$$\text{Output\_image} = \text{Image} * \text{Kernel}$$

Animation source: Dominic Waithe, Oxford University  
[https://github.com/dwaithe/generalMacros/tree/master/convolution\\_animate](https://github.com/dwaithe/generalMacros/tree/master/convolution_animate)

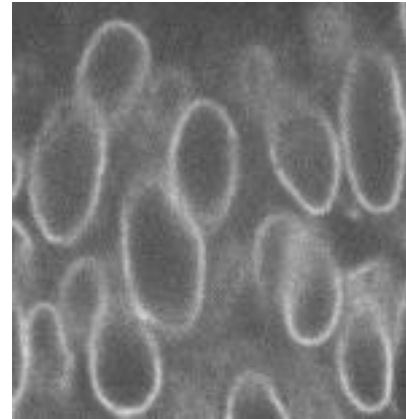
## Different linear filters use different kernels

NOTE:

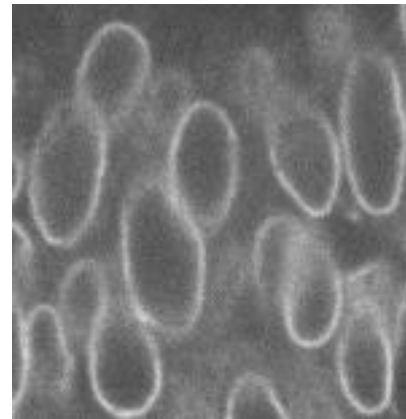
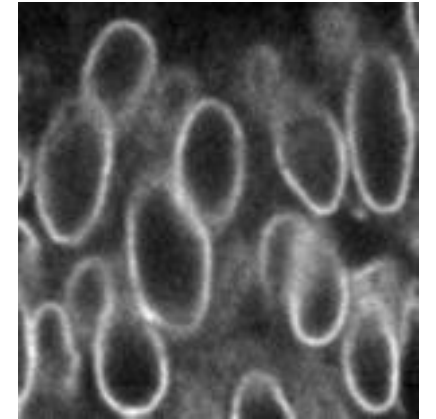
The choice of kernel  
can be pre-selected based on the  
type of filter to be applied

- Other Examples

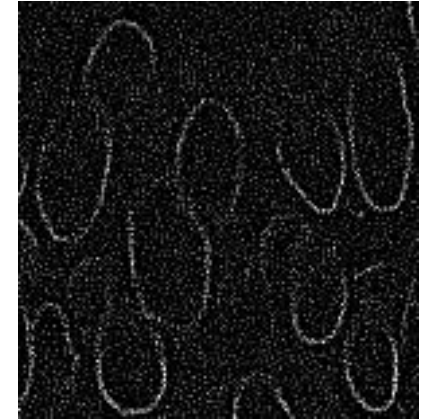
- Mean
- Gaussian blur
- Sobel-operator
- Laplace-filter



$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



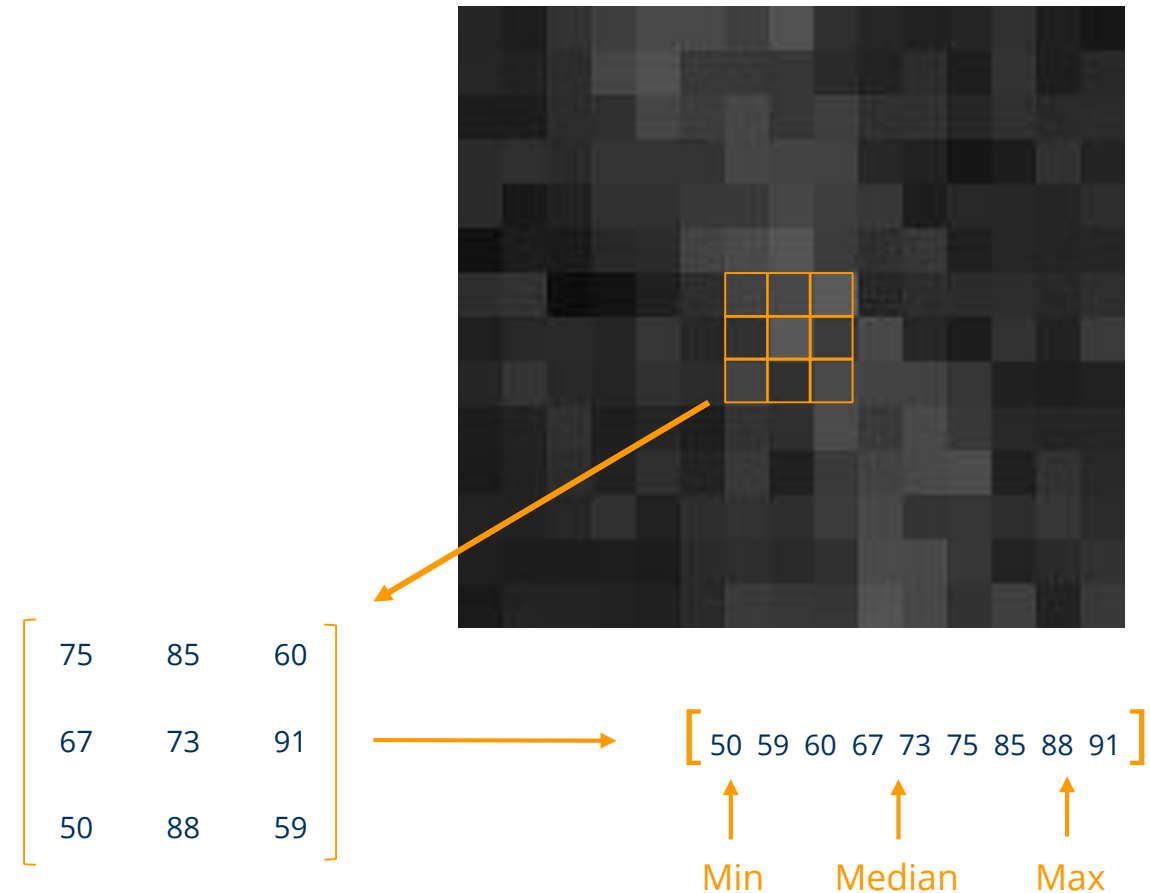
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



- Nonlinear filters also replace pixel value inside as rolling window but in a non linear function.

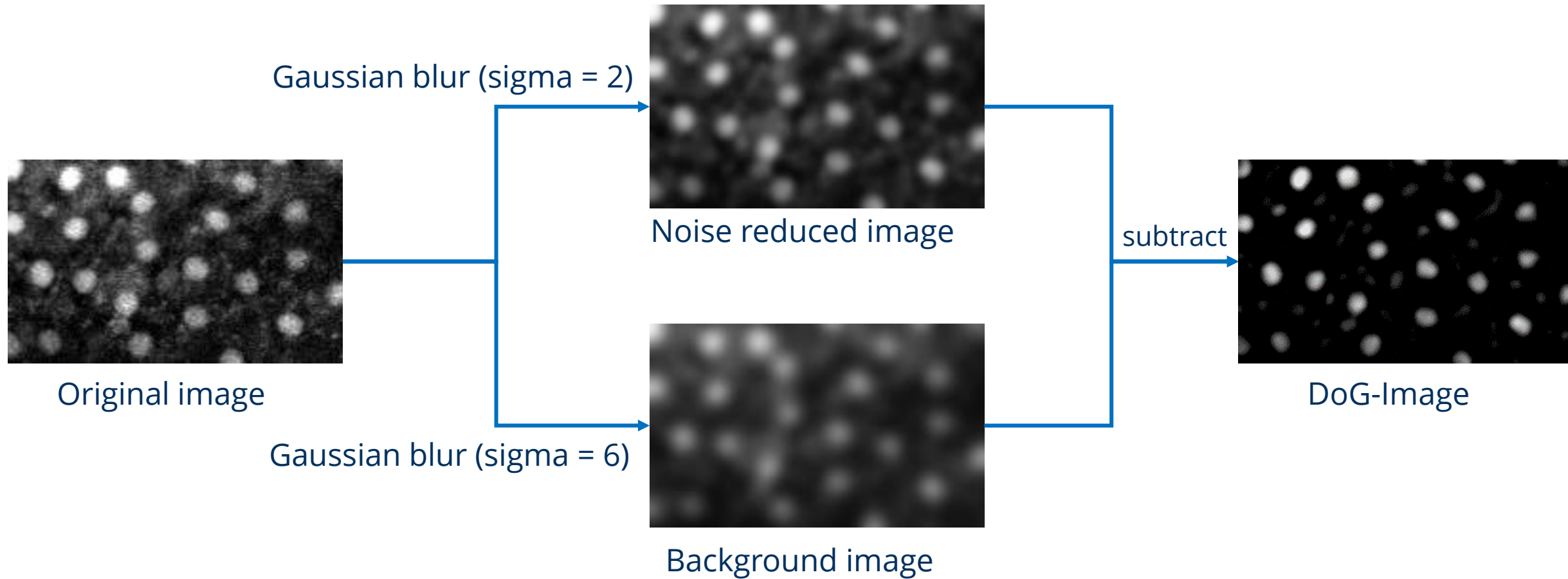
Examples: order statistics filters

- Min
- Median
- Max
- Variance
- Standard deviation



# Difference-of-Gaussian (DoG)

Improve image in order to detect bright objects.



# Laplace-filter

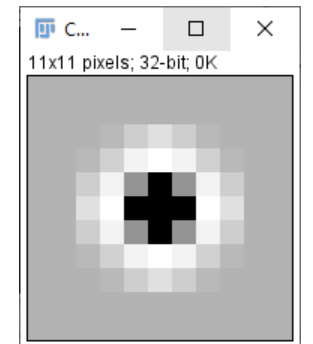
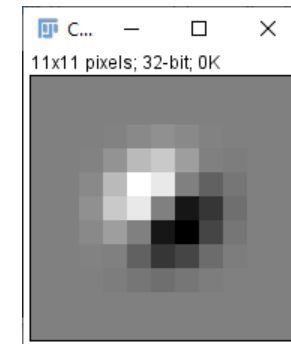
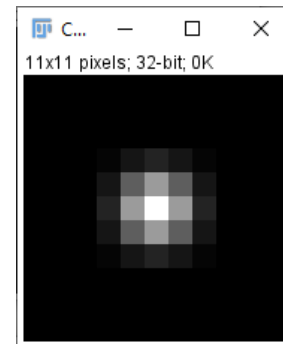
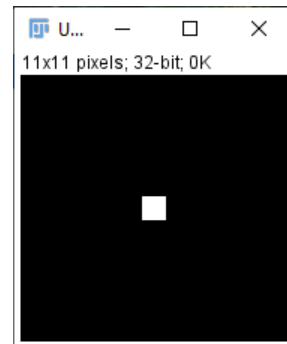
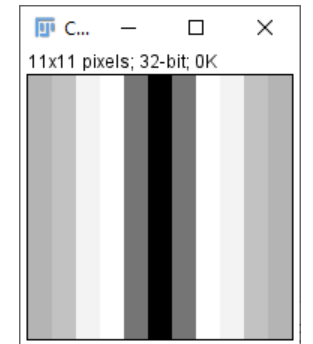
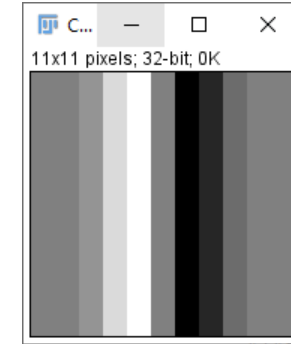
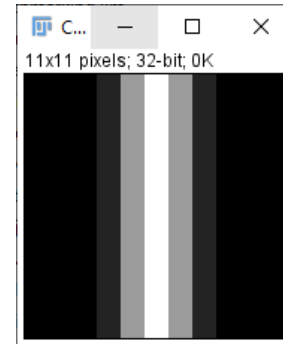
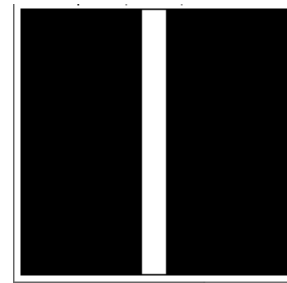
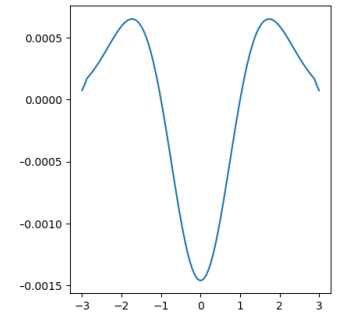
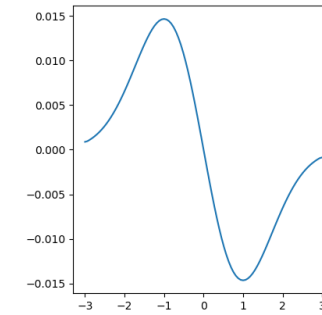
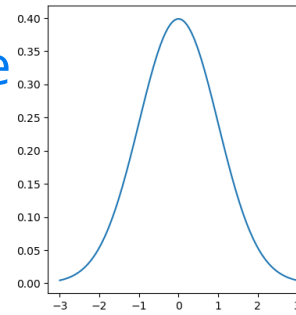
*Second derivative of a Gaussian blur filter*

Used for edge-detection and edge enhancement

Also known as the Mexican-hat-filter

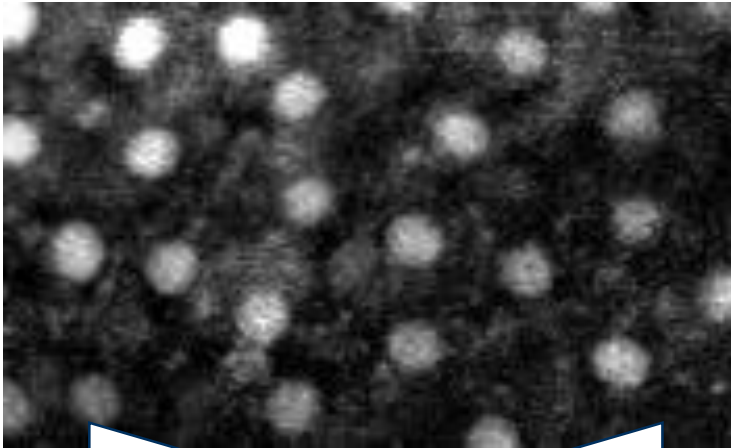
1<sup>st</sup> derivative

2<sup>nd</sup> derivative



# Laplacian-of-Gaussian (LoG)

Laplace filter



Gaussian

blur

$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

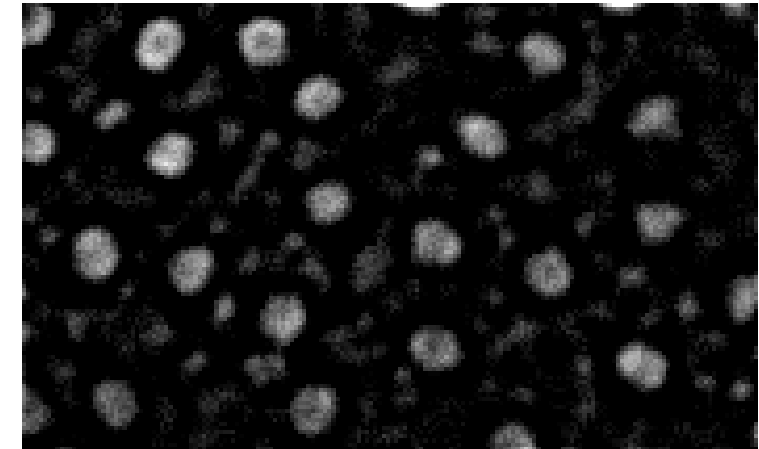


Laplace filtered image

Laplacian of Gaussian filter



$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$



LoG image

In python, for linear filters, we could apply a kernel to an image like this:

```
kernel = np.array(  
    [[1/9, 1/9, 1/9],  
     [1/9, 1/9, 1/9],  
     [1/9, 1/9, 1/9]])
```

NOTE:  
Number of kernels can be  
defined as a new variable

```
from scipy.ndimage import convolve
```

```
output = convolve(image, kernel)
```

But scikit-image already has several of these filters implemented and optimized.

```
from skimage.filters import gaussian
```

```
output = gaussian(image, sigma = 1)
```

Then there is no need to provide a kernel. And you can also directly apply non-linear filters.

```
from skimage.filters import median
```

```
output = median(image)
```

Available filters can be searched like this:

```
from skimage import filters
```

filters.

- LPIFilter2D
- median
- meijering
- prewitt
- prewitt\_h
- prewitt\_v
- rank
- rank\_order
- ridges
- roberts

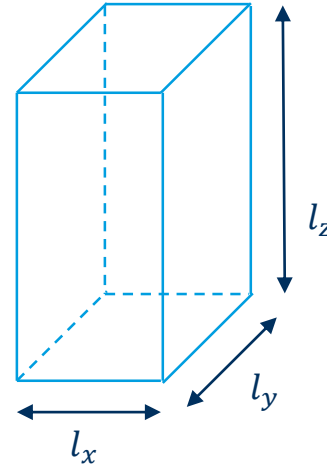
Voxel: “Volume element”, usually anisotropic

An-iso-tropy, from Greek:

- ἀν- (not)
- ἴσος *isos* (equal)
- τρόπος *tropos* (rotation, direction)

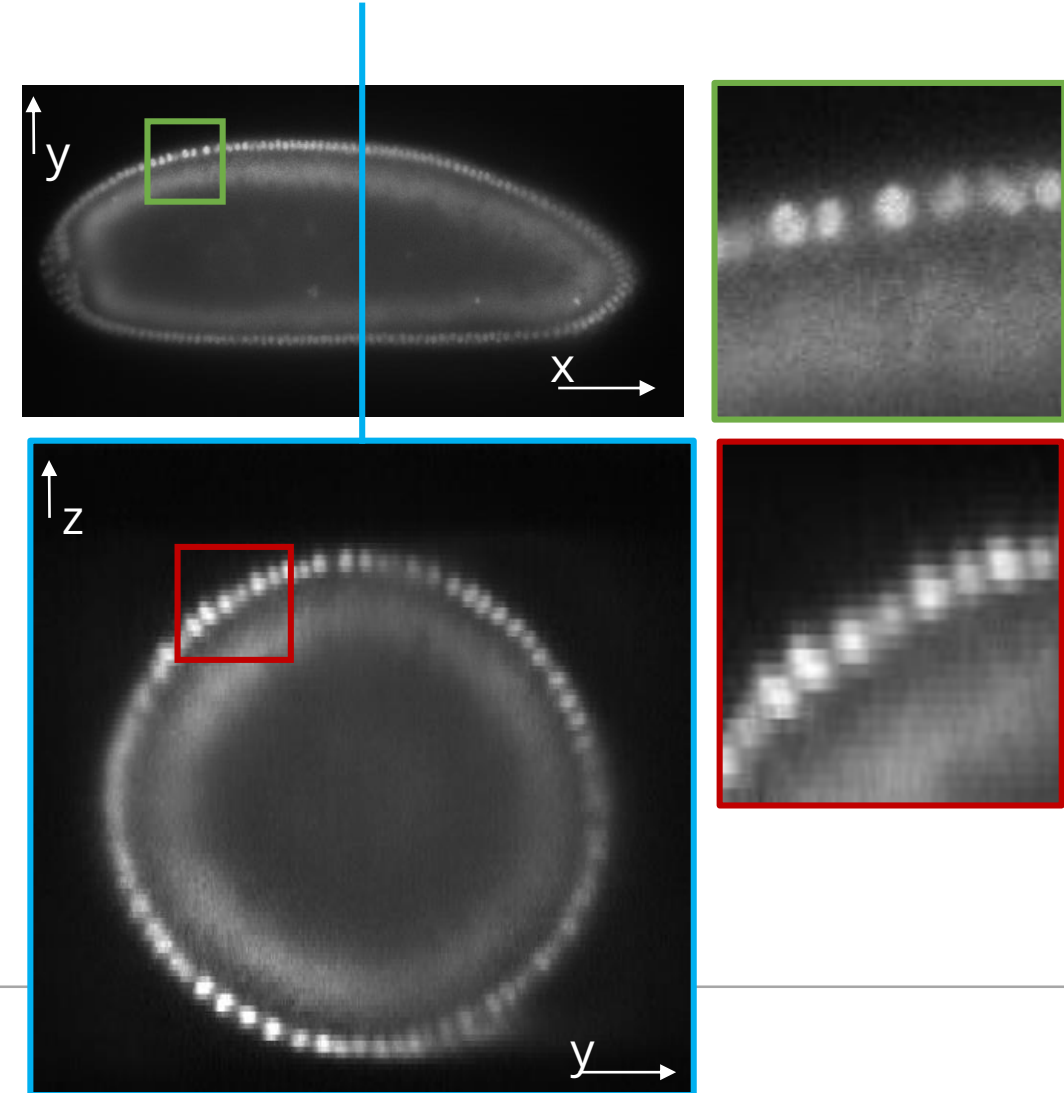
*Not the same in all directions*

Usually in 3D image processing:



$$l_x = l_y \neq l_z$$

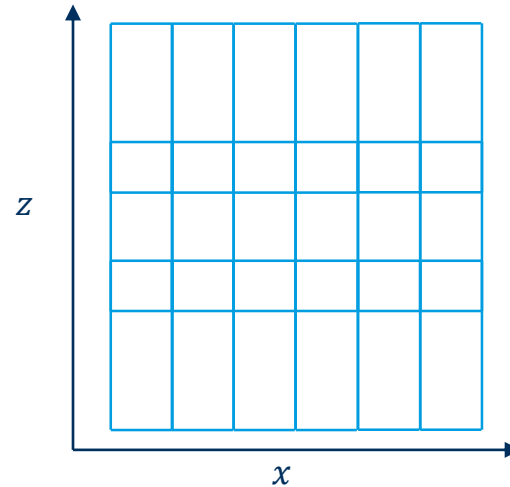
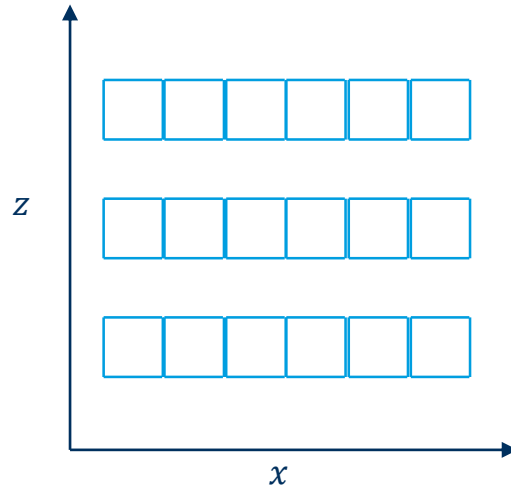
Image analysts *love* to have isotropic voxels, but it's often not possible.



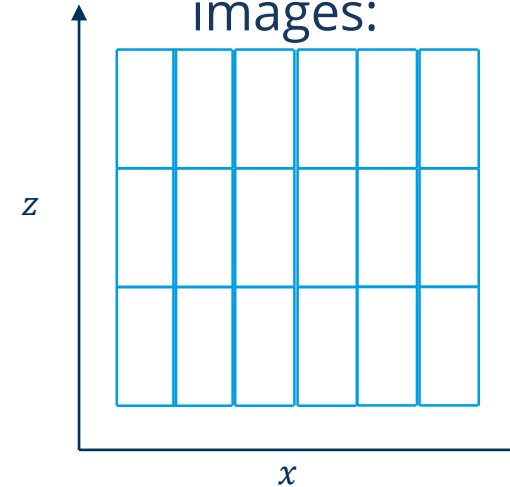


# Slice distance versus slice thickness

What you may have measured using imaging:



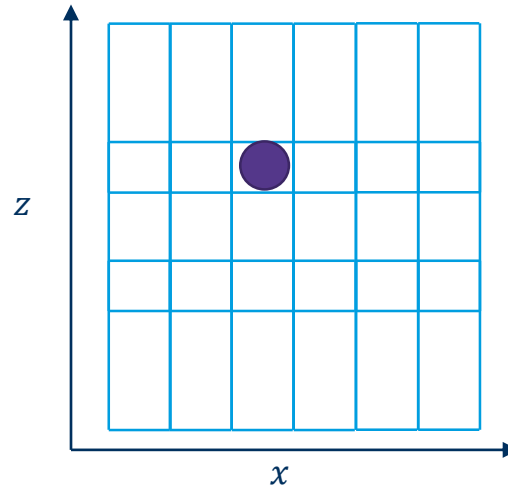
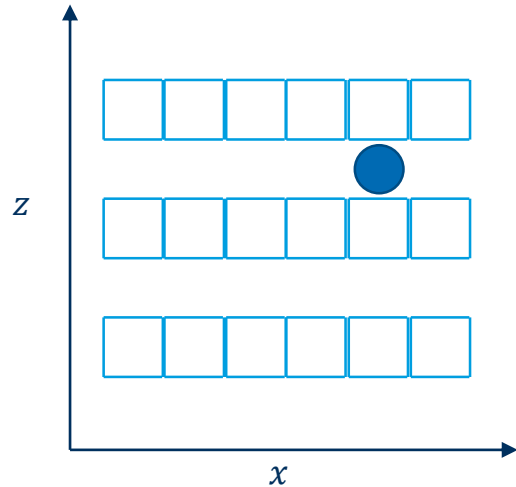
What you see when processing 3D images:



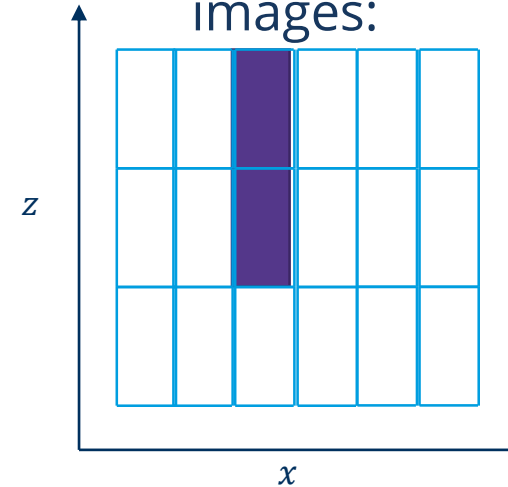
- Slice distance and slice thickness may be different, but
- many image processing tools ignore that.

# Slice distance versus slice thickness

What you may have measured using imaging:



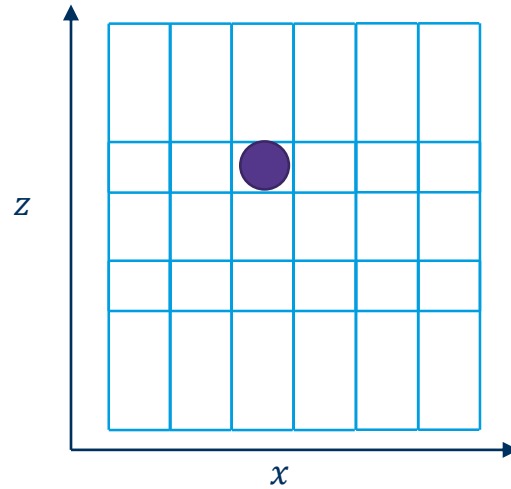
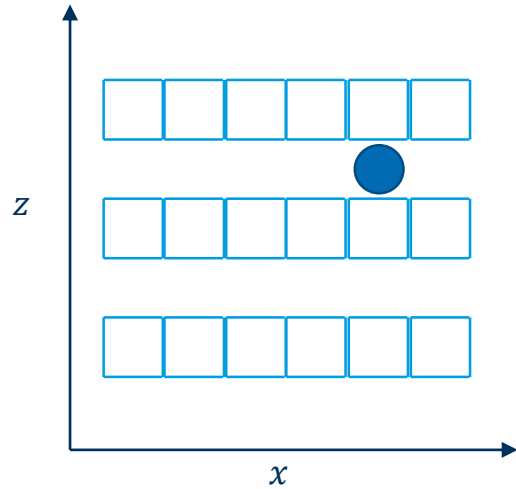
What you see when processing 3D images:



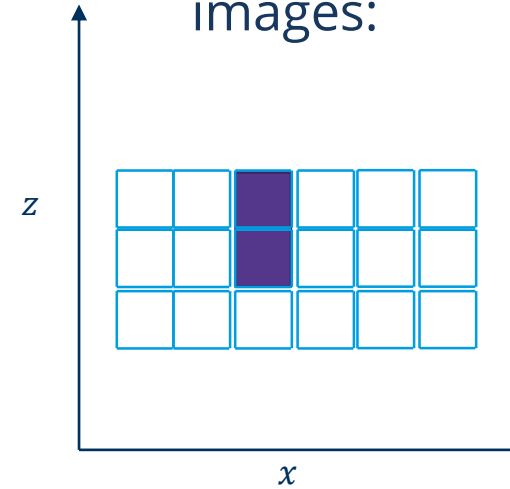
- Slice distance and slice thickness may be different, but
- many image processing tools ignore that!

# Slice distance versus slice thickness

What you may have measured using imaging:



What you see when processing 3D images:



- Slice distance and slice thickness may be different, but
- many image processing tools ignore that!

The visualization can be fixed in napari by providing a 'scale' argument.

```
viewer.add_image(image, name = 'image')
```

```
viewer.layers['image'].scale = [factor_z, factor_y, factor_x] # Z, Y, X order
```

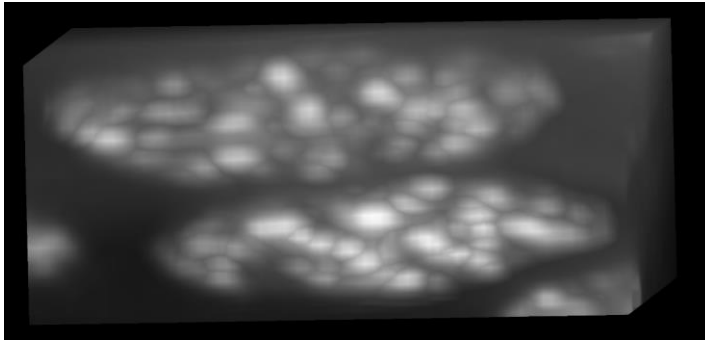


image.shape

(32, 61, 74)

```
reference_size = voxel_size_z  
  
factor_z = voxel_size_z / reference_size  
factor_y = voxel_size_y / reference_size  
factor_x = voxel_size_x / reference_size
```

Voxel\_size\_z = 1  $\mu\text{m}$   
Voxel\_size\_y = 0.2  $\mu\text{m}$   
Voxel\_size\_x = 0.2  $\mu\text{m}$

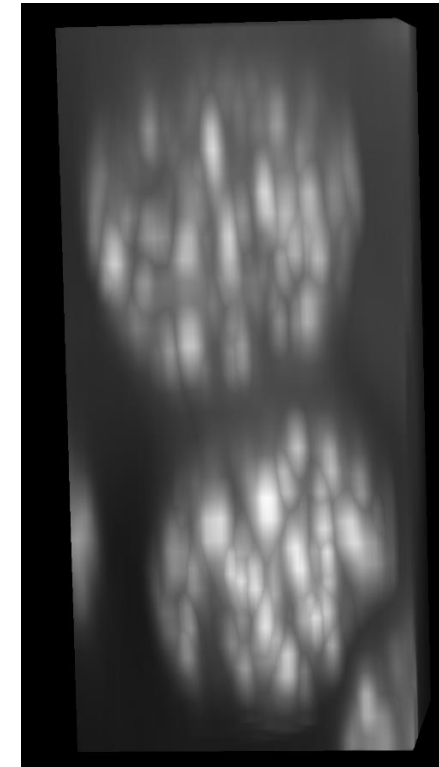


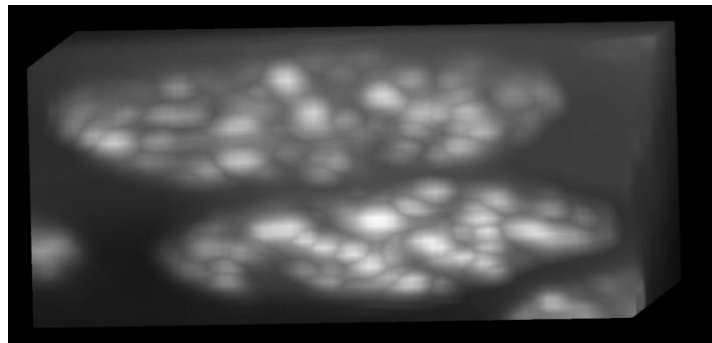
image.shape

(32, 61, 74)

But that does not change voxel size, just adjusts visualization.

The functions are typically the same, but many of them may not account by default different voxel sizes.

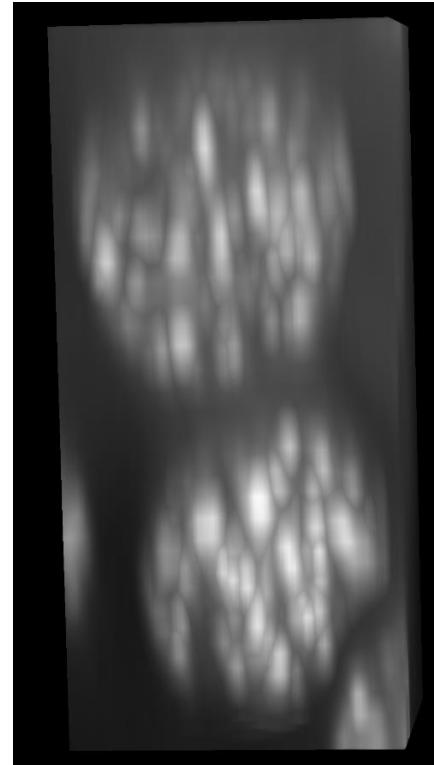
If the filter does not take voxel size into account, we would need to rescale the image before applying a filter.



```
image.shape
```

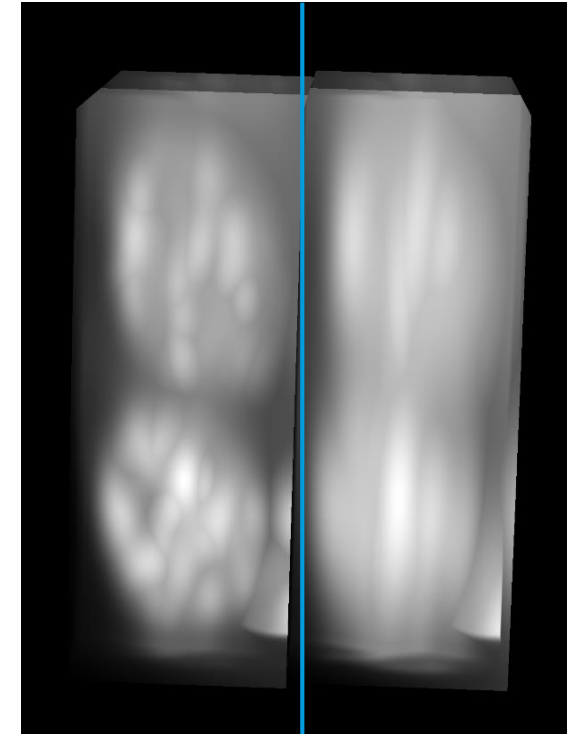
(32, 61, 74)

rescale



```
image.shape
```

(158, 61, 74)



Gaussian blur on  
rescaled image

Gaussian blur on  
image