5. Image Processing - Filtering in the Spatial Domain

Institute for Biomedical Imaging, Hamburg University of Technology

- Lecture: Prof. Dr.-Ing. Tobias Knopp
- Exercise: Konrad Scheffler, M.Sc.

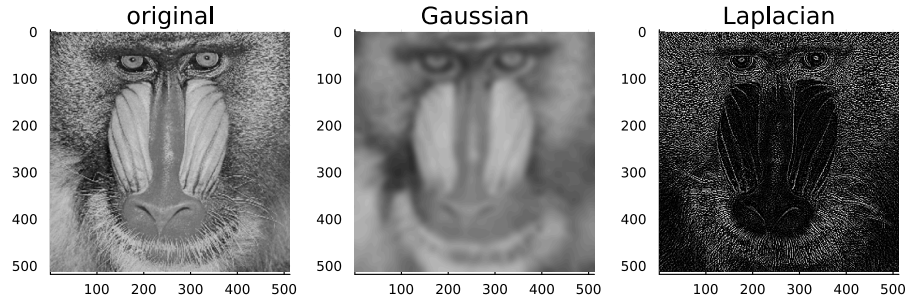
Table of Contents

5.1 Basics of Spatial Filtering

We next investigate transformations that are

- linear
- shift-invariant
- taking a neighborhood into account

So in comparison with the intensity based filters we restrict us in one dimension (linearity) but extend us in another dimension (spatial relationships taken into account). Before we dig deeper into this topic we want to have a look what you can do with filtering:



You can see that the filtering with a Gaussian kernel leads to a smoothing, whereas the filtering with a Laplacian kernel leads to an image where the edges are highlighted. All this is done by just changing the kernel, which is introduced next.

Convolution vs Correlation

There are actually two different operations that one associates with filtering. For two images $f, h : \mathbb{Z}^2 \to \Gamma$, the first is called *correlation* and defined as:

$$(f \star h)(u,v) = \sum_{y=-L_y}^{L_y} \sum_{x=-L_x}^{L_x} f(u+x, v+y) h(x,y)$$

with $L_x, L_y \in \mathbb{N}$. The other one is called *convolution*:

$$(f * h)(u,v) = \sum_{y=-L_y}^{L_y} \sum_{x=-L_x}^{L_x} f(u-x, v-y) h(x,y)$$

It's important to not get them mixed up.

The reason that they are often mixed up is that they are equivalent under certain circumstances: If the kernel h is symmetric (i.e. $h(x,y) = h(-x,-y)$) it holds that

$$\begin{aligned}(f * h)(u,v) &= \sum_{y=-L_y}^{L_y} \sum_{x=-L_x}^{L_x} f(u-x, v-y) h(x,y) \\ &= \sum_{y=-L_y}^{L_y} \sum_{x=-L_x}^{L_x} f(u+x, v+y) h(-x,-y) \\ &\overset{h \text{ symm.}}{=} \sum_{y=-L_y}^{L_y} \sum_{x=-L_x}^{L_x} f(u+x, v+y) h(x,y) \\ &= (f \star h)(u,v)\end{aligned}$$

We will mostly consider the convolution since it needs to be considered when looking at equivalent relations in Fourier space but we note that many/most image processing methods actually consider the correlation as being the filtering operation. For instance you can have a look at the Julia package ImageFiltering.jl where this is documented.

Note

The term convolutional neural networks (CNN) in machine learning frameworks is actually a misnomer since the operation is usually defined as a correlation. In the context of CNNs the kernels are not symmetric since they are optimized value-by-values when training a neural network.

Note

Filtering is actually a frequency domain concept. Since one can carry out the operation both in the spatial as well as in the frequency domain one calls it filtering in both domains.

5.1.1 Dimensions

The function h that we use in the convolution is usually called the kernel. It is of size $M_x \times M_y = (2L_x + 1) \times (2L_y + 1)$. The input image f was of size $N_x \times N_y$ where usually $M_x \ll N_x$ and $M_y \ll N_y$.

This brings us to the issue of choosing the size of the output image, i.e. how to choose u and v. There are basically three choices.

1. $$u = 1 + L_x, \ldots, N_x - L_x, v = 1 + L_y, \ldots, N_y - L_y$$

   This is the safest choice since it ensures that we do not evaluate f outside of its definition range. The downside is that we shrank the image, which is often not desired.

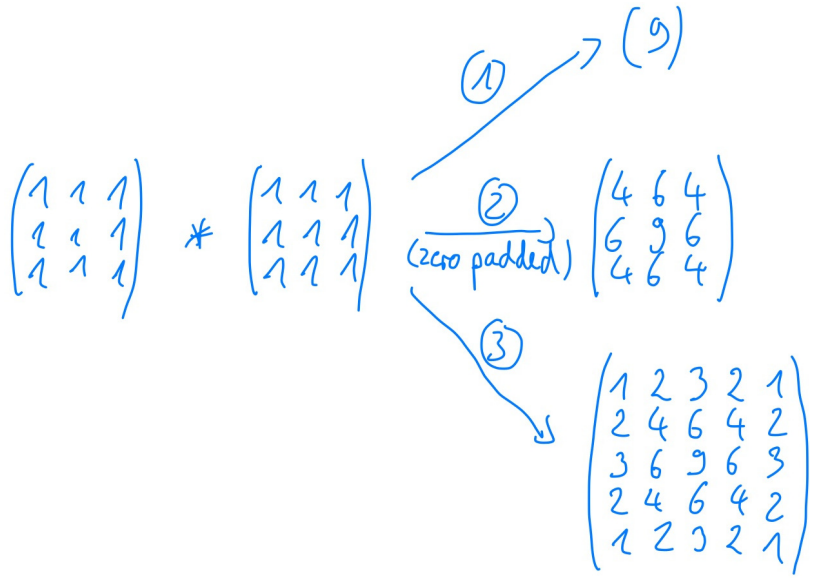
2. $$u = 1, \ldots, N_x, v = 1, \ldots, N_y$$

   This is the standard choice since the output image has the same size as the input image. However, it needs padding (see next section).

3. $$u = 1 - L_x, \ldots, N_x + L_x, v = 1 - L_y, \ldots, N_y + L_y$$

   This option is considered to be the full convolution. It basically puts the kernel outside the image with just one overlapping pixel. Option 3 is what one would expect when comparing discrete and continuous convolution. This option also requires padding.

The following example shows the three choices for example kernels with $N_x = N_y = 3$ and $L_x = L_y = 1$ and zero padding.



5.1.2 Boundary Handling

Padding means that we extend the image f in a meaningful way. There are different options that we illustrate with a row $\boxed{a, b, c, d, e, f}$ that has the two border pixels a and f and we need to extend it to the left and to the right.

- **replicate:** The border pixels extend beyond the image boundaries.

  $$\boxed{a, a, a, a \mid a, b, c, d, e, f \mid f, f, f, f}$$

- **circular:** The border pixels wrap around. For instance, indexing beyond the left border returns values starting from the right border.

  $$\boxed{c, d, e, f \mid a, b, c, d, e, f \mid a, b, c, d}$$

- **reflect:** The border pixels reflect relative to a position between pixels. That is, the border pixel is omitted when mirroring.

  $$\boxed{e, d, c, b \mid a, b, c, d, e, f \mid e, d, c, b}$$

- **symmetric:** The border pixels reflect relative to the edge itself.

  $$\boxed{d, c, b, a \mid a, b, c, d, e, f \mid f, e, d, c}$$

- **zero fill:** Fill the values with zero.

  $$\boxed{0, 0, 0, 0 \mid a, b, c, d, e, f \mid 0, 0, 0, 0}$$

Which one is the best one cannot be answered in general because it depends on the image characteristic itself. If the underlying signal is periodic, the best is to use *circular* boundary handling.

Otherwise *reflect/replicate/symmetric* are usually the methods of choice.

5.1.3 Calculation Rules

If we ignore for the moment the special boundary handling and consider the convolution to take place on the entire plane $\mathbb{Z}^2$ (and functions f, g, h having compact support) the following rules hold:

- commutativity:

$$f * g = g * f$$

- associativity:

$$f * (g * h) = (f * g) * h$$

- distributivity:

$$f * (g + h) = (f * g) + (f * h)$$

- associativity with scalar argument:

$$a(f * g) = (af) * g = f * (ag)$$

- integration

$$\int_{\mathbb{R}^d} (f * g)(x)\, dx = \left(\int_{\mathbb{R}^d} f(x)\, dx\right)\left(\int_{\mathbb{R}^d} g(x)\, dx\right)$$

- differentiation

$$\frac{\partial}{\partial x_i}(f * g) = \frac{\partial f}{\partial x_i} * g = f * \frac{\partial g}{\partial x_i}$$

5.1.4 Composing Kernels

Quite often it is handy to build complex filters based on more simple ones. If we take the associativity rule it becomes clear that $(f * h_1) * h_2 = f * (h_1 * h_2) = f * h_{\text{combined}}$ where now h_{combined} is the combined filter.

Examples:

$$(1 \quad 1) * (1 \quad 1) = (1 \quad 2 \quad 1)$$

In particular this is handy when building 2D filters based on 1D ones. We can simply combine a 3×1 and a 1×3 filter like this

$$(1 \quad 2 \quad 1) * \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

This automatically leads to a separable kernel. This is often named a tensor product approach. We note that tensors can also be made without convolution by plain linear algebra:

$$\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} (1 \quad 2 \quad 1) = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

The difference becomes apparent when considering the 1D example above where the dimension along which it is convolved is larger than 1.

5.1.5 Normalizing Kernels

If a kernel does not sum up to 1 one can simply normalize it using

$$h_{\text{normalized}}(x, y) = \frac{h(x, y)}{\sum_{u=-L_x}^{L_x} \sum_{v=-L_y}^{L_y} h(u, v)}.$$

5.2 Smoothing Spatial Filters

A smoothing filter does smooth all sharp elements in an image. This is what happens in photography when using a lens and focussing on a different plane than where the objects are. This effect is exploited to get depth of field.

In image processing the smoothing filter is used to reduce the noise in an image. Let's assume that two neighboring pixels have the same value z and and additional noise component ε with a standard deviation of η and a Gaussian noise distribution. Then the mean of both is

$$\frac{1}{2}(z_1 + \varepsilon_1 + z_2 + \varepsilon_2) = z + \frac{1}{2}(\varepsilon_1 + \varepsilon_2) = z + \varepsilon_{\text{new}}.$$

One can show that the variance of $\varepsilon_1 + \varepsilon_2$ is $2\eta^2$ and in turn the standard deviation is $\sqrt{2}\eta$. If we then take the factor $\frac{1}{2}$ into account we see that the standard deviation of ε_{new} is $\frac{\sqrt{2}}{2} = \frac{1}{\sqrt{2}}$. It easy to see that this generalizes to an $\frac{1}{\sqrt{N}}$ law when considering N pixels to be averaged.

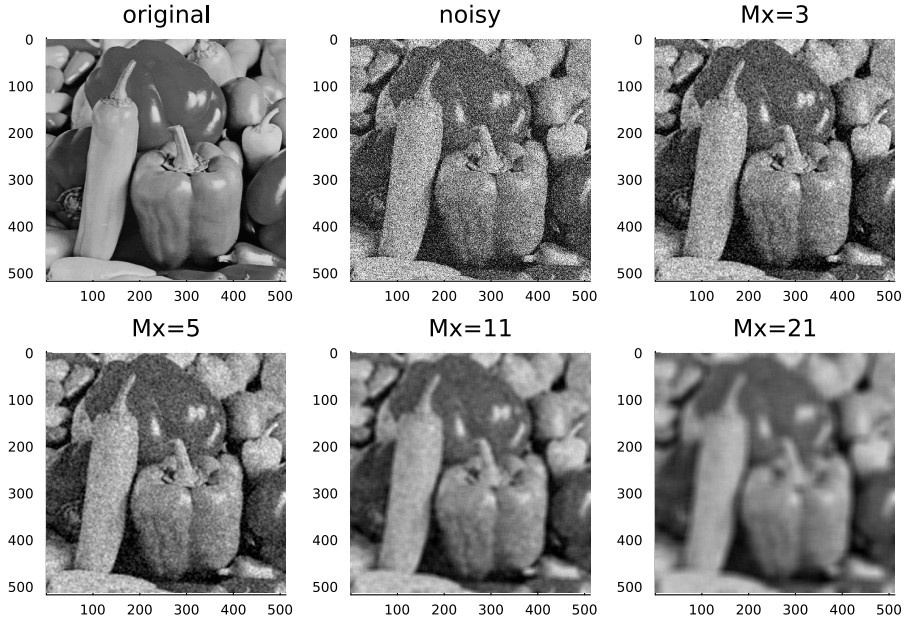
The downside of averaging is of course that z_1 and z_2 are in practice not the same since an image has variations and in turn we distort the image by this operation. What happens is that values are leaking into the neighboring pixels and in turn the spatial resolution is decreased.

5.2.1 Box Filter

Let us now switch to concrete smoothing filters. The *box filter*, or *moving average filter* looks like this

$$\frac{1}{M}\begin{pmatrix} 1 & \dots & 1 \\ \vdots & \vdots & \vdots \\ 1 & \dots & 1 \end{pmatrix} \in \mathbb{R}^{M_x \times M_y}$$

where $M = M_x M_y$. The factor in front of the filter is important to not change the overall intensity of the image.



5.2.2 Binomial Filter

The downside of the box filter is that it has a bad frequency characteristic. We will later see that it corresponds in Fourier space to a multiplication with a sinc function, which has a very inhomogeneous frequency profile leading to certain frequencies being completely blocked.

The downside of the box filter can also be discussed in image space: The box filter assigns to the central pixel all pixels in a certain neighborhood. The distance to the central pixel is not taken into account. This is not desired since in that way a corner pixel on an edge of an image have the same influence as the central pixel.

Thus, what we want is to account for the distance to the central pixel, which can be done by introducing weights. To this end we can exploit the binomial coefficients $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ where k is chosen to be the filter width and n is the running index. With that we obtain filter kernels

$$h^2_{\text{binom}} = \frac{1}{4}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, h^3_{\text{binom}} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix},$$

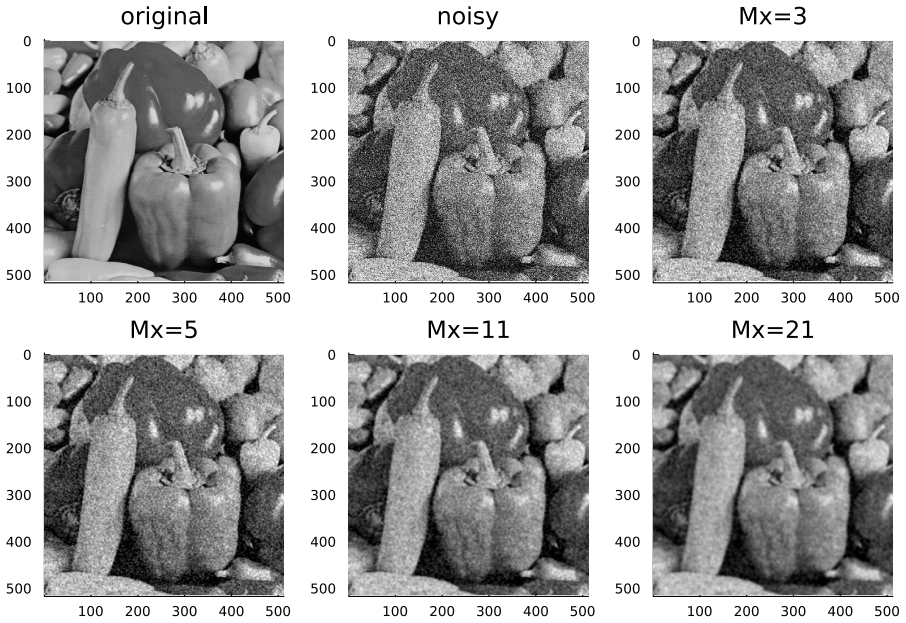
$$h^4_{\text{binom}} = \frac{1}{64}\begin{pmatrix} 1 & 3 & 3 & 1 \\ 3 & 9 & 9 & 3 \\ 3 & 9 & 9 & 3 \\ 1 & 3 & 3 & 1 \end{pmatrix}, h^5_{\text{binom}} = \frac{1}{256}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

where we already have applied the tensor product approach to generate 2D filters based on the 1D filters.

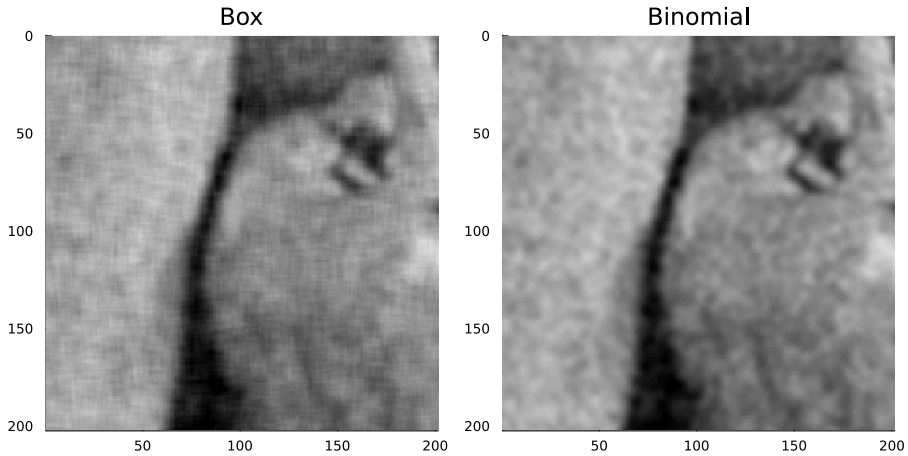
It's interesting to note that these filter can in general be derived by combining the binomial filter with itself:

$$h^d_{\text{binom}} = h^{d-1}_{\text{binom}} * h^2_{\text{binom}}$$

Let us apply the binomial filter to the noisy data:



One can see that we can choose the kernel much larger now without reducing the spatial resolution too much. But this is more a difference in parameter choice. The differences between both filters are not huge but the binomial filter generates no block artifacts that one can identify for the box filter:

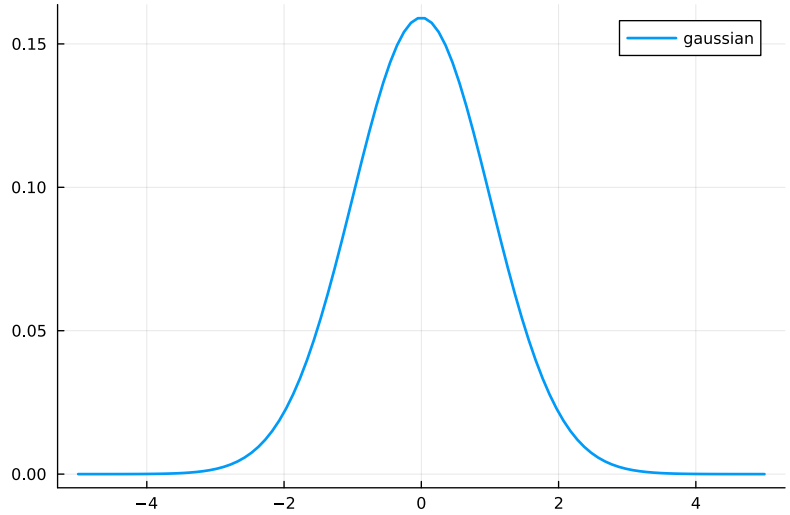


5.2.3 Gaussian Filter

The binomial kernel was a clear improvement compared to the box filter and it has the nice property that in principle it can be performed in integer arithmetic if the factor is applied after the convolution. The drawback is that one has no continuous parameter to adjust the smoothing effect. This brings us to the Gaussian filter that doesn't have this drawback. It is defined as

$$h^{\sigma}_{\text{Gaussian}}(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

Let's have a look how this function looks like for $\sigma = 1.0$ and a profile along the x direction:



What is interesting to note here is that the kernel is infinitely sized, i.e. one needs to sample it and cut of at a certain value. Often a value of $L_x = L_y = 2\sigma$ is chosen in which case the curve has dropped to $\exp(-2) \approx 13.5\%$ of its maximum value.

Since the kernel is cut of, the scaling factor is not valid anymore to let the kernel sum up to 1. Thus one needs to apply the kernel normalization technique discussed earlier. With that we can define the gaussian kernel as

```
gaussian_kernel (generic function with 2 methods)
```

```
function gaussian_kernel(σ,m=2σ)
    h = [ exp(-(x^2+y^2)/(2*σ^2)) for x=-m:m, y=-m:m]
    return h./sum(vec(h))
end
```

```
5×5 Matrix{Float64}:
 0.00296902  0.0133062  0.0219382  0.0133062  0.00296902
 0.0133062   0.0596343  0.0983203  0.0596343  0.0133062
 0.0219382   0.0983203  0.162103   0.0983203  0.0219382
 0.0133062   0.0596343  0.0983203  0.0596343  0.0133062
 0.00296902  0.0133062  0.0219382  0.0133062  0.00296902
```

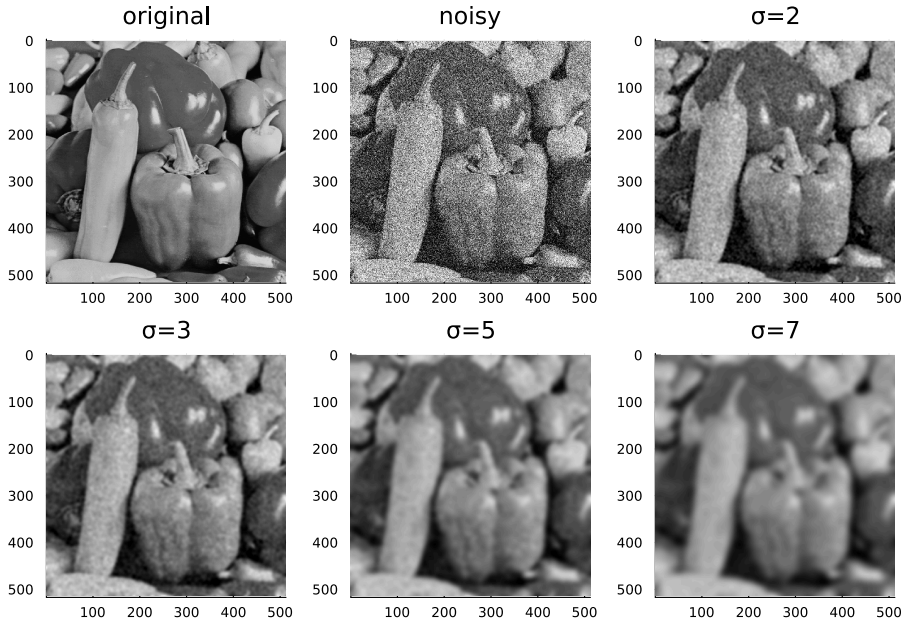
```
gaussian_kernel(1.0)
```

Note that the **ImageFiltering.jl** package has this and other kernels readily available:

```
5×5 OffsetArray(::Matrix{Float64}, -2:2, -2:2) with eltype Float64 with indices -2:2×-2:2:
 0.00296902  0.0133062  0.0219382  0.0133062  0.00296902
 0.0133062   0.0596343  0.0983203  0.0596343  0.0133062
 0.0219382   0.0983203  0.162103   0.0983203  0.0219382
 0.0133062   0.0596343  0.0983203  0.0596343  0.0133062
 0.00296902  0.0133062  0.0219382  0.0133062  0.00296902
```

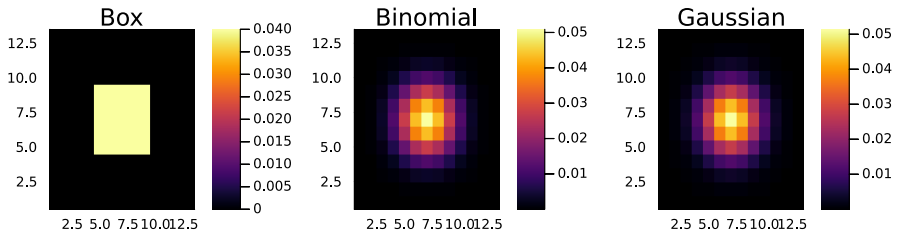
```
Kernel.gaussian(1.0)
```

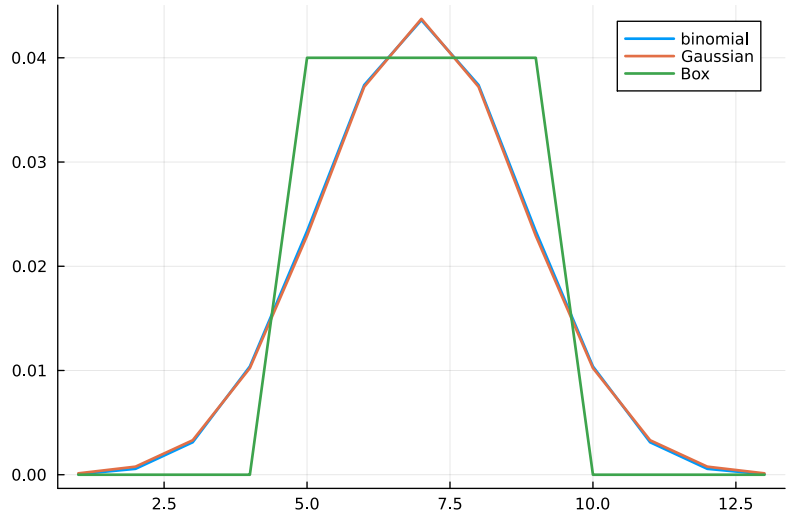
Let's apply the Gaussian as noise reducing filter:



5.2.4 Comparison of Filter kernels

The following plot shows the box, binomial and Gaussian kernel in comparison for $M_x = M_y = 13$ and $\sigma = 1.76$. Below the 2D image a line profile is shown. One can clearly see that the binomial filter matches the Gaussian filter for the chosen σ.





5.3 Sliding Window Mapping

The basic idea of convolution-based filtering is to take a small region around a pixel and derive an updated value based on this region. What is essential here, is that the filter kernel does not depend on the actual values within the image region.

When loosening this restriction we can build more powerful methods that take for an image pixel index $\mathbf{r}$ the region $R(\mathbf{r})$ and apply some function h yielding the output image, i.e.

$$g(\mathbf{r}) = h(f(R(\mathbf{r})))$$

where $f(\mathbf{r})$ is the input image. The important difference is that the regular filter does not depend on f but here h depends on f.

5.3.1 Median Filter

The most prominent example of a sliding window mapping is the application of a median. The median can be defined as $\mathbf{median} : \mathbb{R}^N \to \mathbb{R}$

$$\mathbf{median}(\mathbf{u}) = \begin{cases} \mathbf{sort}(\mathbf{u})_{\frac{N-1}{2}+1} & \text{if } N \text{ is odd} \\ \frac{1}{2}\left(\mathbf{sort}(\mathbf{u})_{\frac{N}{2}} + \mathbf{sort}(\mathbf{u})_{\frac{N}{2}+1}\right) & \text{if } N \text{ is even} \end{cases}$$

For instance we have

$$\mathbf{median}(\,(1,2,7,9)^T\,) = 4.5$$

$$\mathbf{median}(\,(1,2,7,9,5000)^T\,) = 7$$

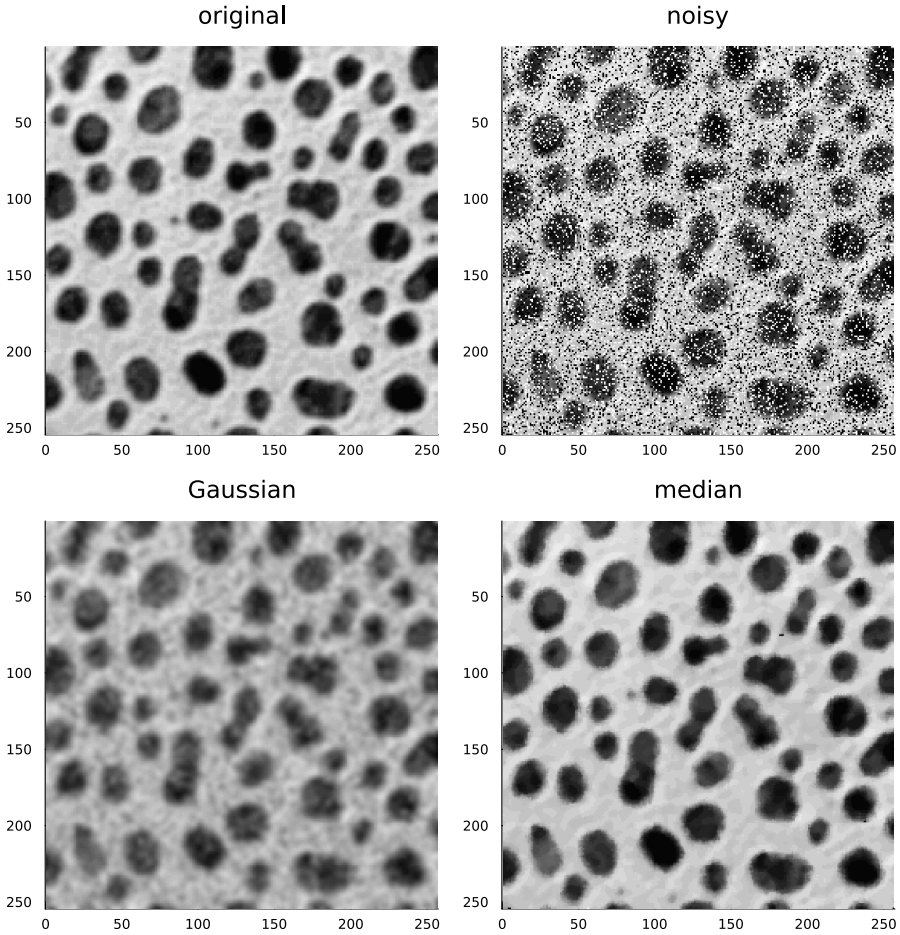
Applications

The median is an alternative to the convolution-based filters discussed before. For uncorrelated Gaussian noise the median filter will not improve the result since the Gaussian filter is tailored towards the noise model.

However, in the case of so-called salt-and-pepper noise the Gaussian filter works much worse.

Salt and pepper noise means that the image contains noise in the form of impulses, i.e. strong deviations in individual pixels. This is typical for the case where one has defective pixels. What happens when applying a Gaussian filter is that the defective pixels is smeared into several neighboring pixels and it can only be removed when using a large kernel width strongly degrading the spatial resolution.

The median filter, however, can handle salt-and-pepper noise case much better. It orders the pixel and in turn the defective value is very unlikely to be the middle of the sorted array. Instead it will be at the beginning or the end and is filtered out. Here is an example showcasing this:



5.4 Sharpening Spatial Filters

The opposite of a smoothing filter is a sharpening filter. In fact one can think of the two as the inverses of each other. But convolution operations cannot be simply inverted in spatial domain and we will therefore take a more heuristic approach in the spatial domain (inverting will be discussed next lecture).

What is required to make an image more sharp? In fact, a sharp image is characterized by its sharp edges. Thus, image sharpening methods have the goal to highlight edges more. This means, if you have a blurred edge you want to add an image where only the edge is sharply drawn. Thus the idea is consider an approach like this

$$f_{\text{sharpened}}(\mathbf{r}) = f_{\text{smooth}}(\mathbf{r}) + f_{\text{edge}}(\mathbf{r})$$

where $f_{\text{smooth}}(\mathbf{r})$ is the original image.

5.4.1 Edge Enhancement

We need a filter that enhances the edges of an image. This can be done by applying the spatial derivative. For discrete functions, the spatial derivative is carried out using finite differences, i.e. instead of

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

which would be used for a continuous function we use

$$\delta(f)(x) = f(x+1) - f(x).$$

Similarly, instead of the second derivative, which can be shown to be

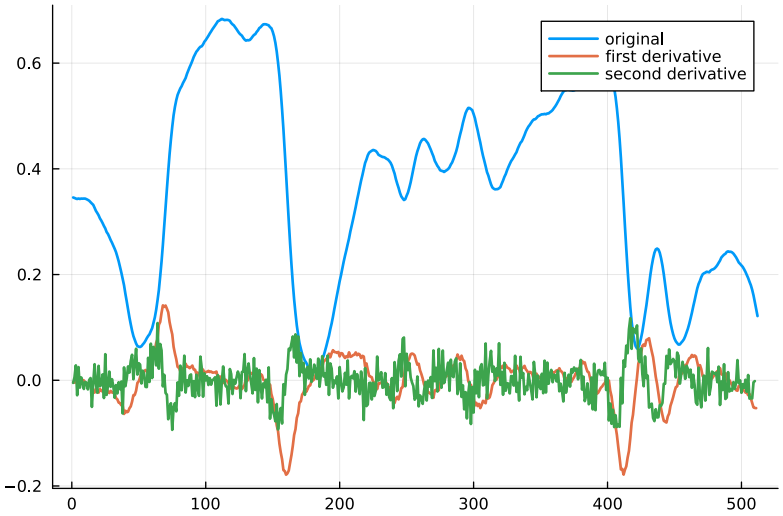
$$\frac{\partial^2 f}{\partial^2 x} = \lim_{h \to 0} \frac{2f(x) - f(x+h) - f(x-h)}{h^2},$$

we can use

$$\delta^2(f)(x) = 2f(x) - f(x+1) - f(x-1).$$

In both cases we exploited that the pixel spacing is $h = 1$.

Now the question is: What derivative is more appropriate for edge enhancement. Let have a look at the following case study:

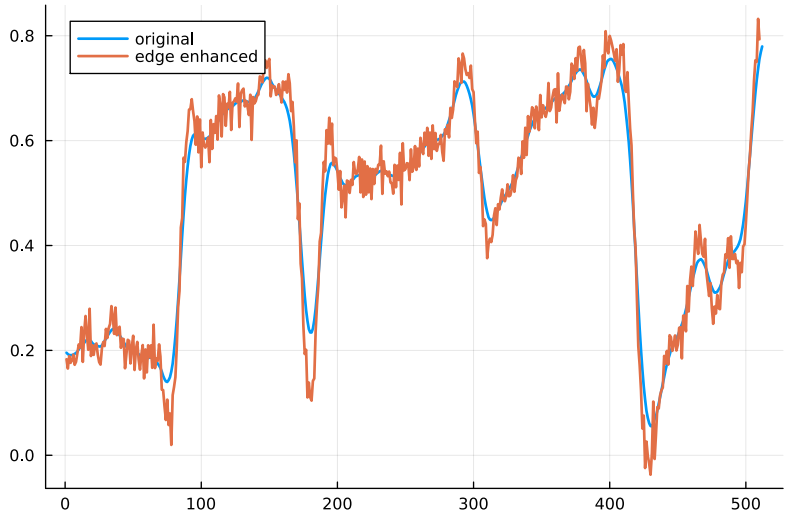


From this example we can already learn several things:

- The first derivative has a single peak for each edge. This peak appears where the tangent has the maximum rise.
- The second derivative has two peaks: One where the blurred function has the maximum curvature at the beginning and one at the end of the transition.
- Derivatives enhance noise, especially when they are applied multiple times.

From the perspective of edge enhancement we actually do not want to change the image where the strongest rise is but we want to make the transition narrower. Thus, the second derivative is more attractive since it can cancel out exactly those signals where the transition already began because of the artificial blurring in the image.

Let's add the edge enhanced signal to the original signal and see the edge enhancement in action:



Kernel Perspective

Looking at the first and second order difference operator we can see that they can actually can be written as a filters

$$h_{\text{diff}} = (-1 \quad 1) \quad \text{and} \quad h^2_{\text{diff}} = (-1 \quad 2 \quad -1)$$

In fact we have

$$h^2_{\text{diff}} = h_{\text{diff}} * h_{\text{diff}}$$

what you can also verify in Julia:

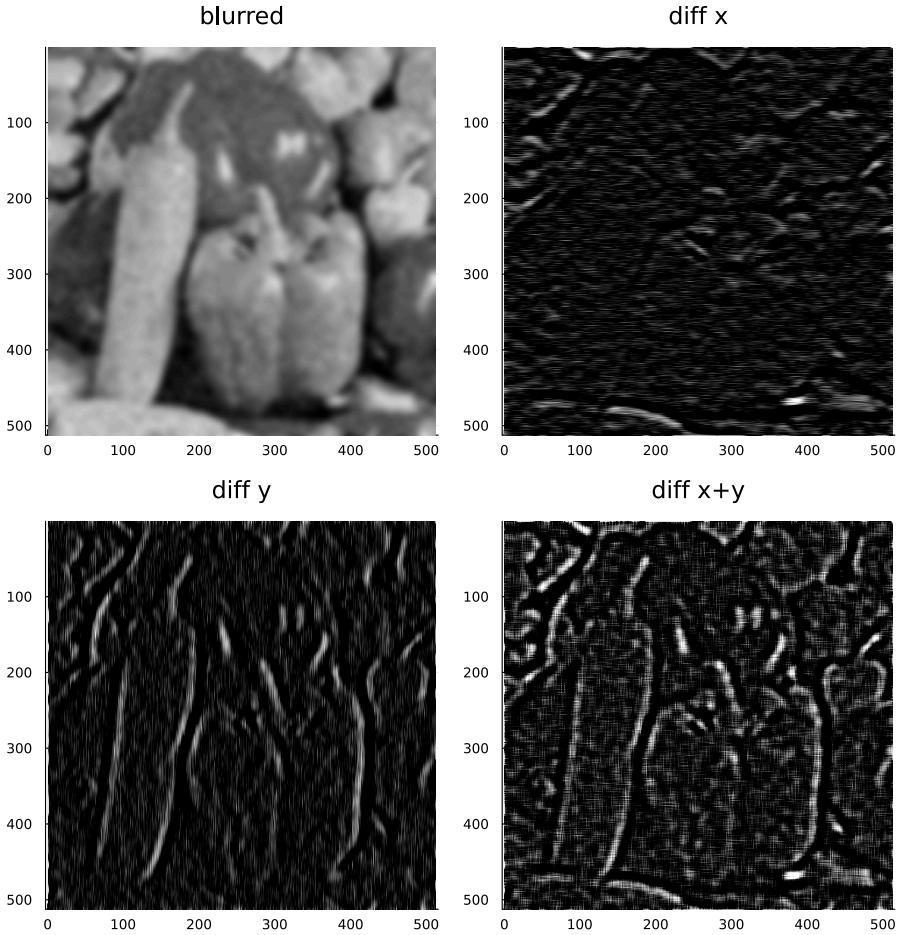
```
▸ [1, -2, 1]
1 conv([-1,1],[-1,1])
```

5.4.2 Laplace Filter

In 2D images, the direction of edges needs to be taken into account. To this end one can combine the partial derivatives

$$\frac{\partial^2 f}{\partial^2 x} \quad \text{and} \quad \frac{\partial^2 f}{\partial^2 y}.$$

Let's have a look at the two for an example image:



We can see that $\frac{\partial^2 f}{\partial^2 x}$ enhanced the edges in the x direction while $\frac{\partial^2 f}{\partial^2 y}$ enhances the edges in the other direction. To combine both we can add them, which is known to be the Laplace operator:

$$\begin{aligned}\Delta f &= \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \\ &= \nabla \cdot \nabla f \\ &= \nabla^2 f\end{aligned}$$

where $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)^T$ is the gradient of the image.

Translating this to the discrete setting we thus obtain the Laplacian kernel

$$h_{\text{Laplacian}} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 2 & 0 \\ 0 & -1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

In practice, this kernel is only isotropic when considering 90° angles. To also take 45° angles into account one can define a rotated Laplacian

$$h^{45°}_{\text{Laplacian}} = \begin{pmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{pmatrix}$$

which can be added to the regular one:

$$h^{\text{isotropic}}_{\text{Laplacian}} = h_{\text{Laplacian}} + h^{45°}_{\text{Laplacian}} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Combining with Smooth Image

Recall that we aimed for combining the smoothed image with the edge image in an additive manner. With our knowledge about the Laplace operator we can express this in continuous notation as

$$f_{\text{sharpened}}(\mathbf{r}) = f_{\text{smooth}}(\mathbf{r}) + \alpha \Delta f_{\text{smooth}}(\mathbf{r})$$

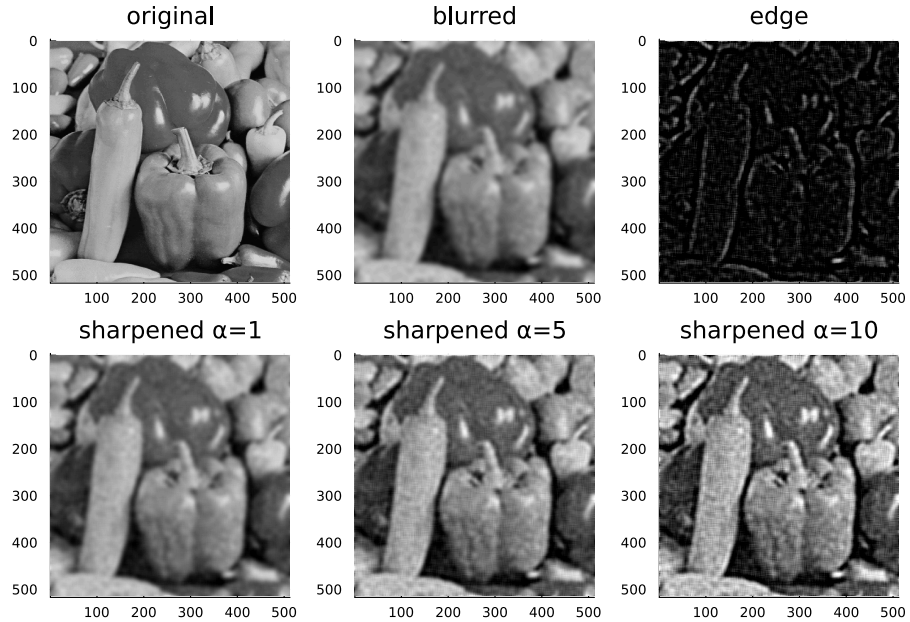
Here, the parameter α is used to weight the edge image. This allows us to control, how sharp the edges should get. If α is chosen too low the image is still blurred. If it is chosen too large the edges are over-expressed and the entire image contains more noise artifacts.

Note

When using the parameter α the operation is also known as unsharp masking and used in the printing and publishing industry since the 1930s to sharpen images.

The following shows unsharp masking in action. You can also change the value of α using this slider

10



How does the kernel $h^{\alpha}_{\text{sharpened}}$ look like?

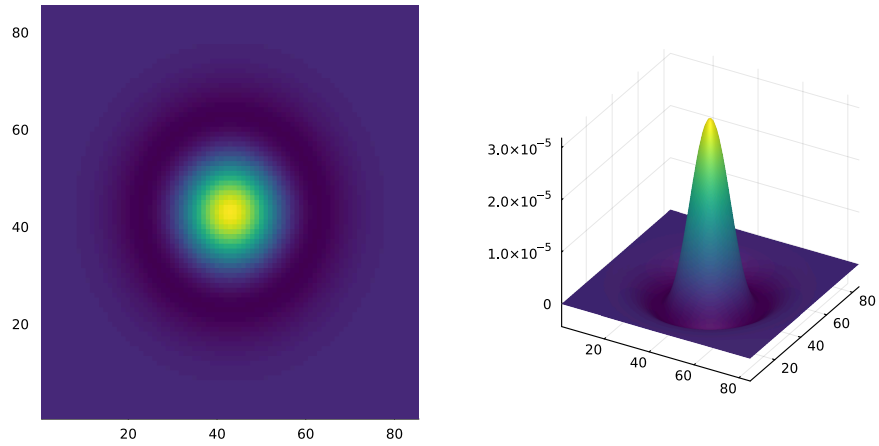
$$
\begin{aligned}
h^{\alpha}_{\text{sharpened}} &= h_{\text{identity}} + \alpha h^{\text{isotropic}}_{\text{Laplacian}} \\
&= \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \alpha \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \\
&= \begin{pmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & 1+\alpha 8 & -\alpha \\ -\alpha & -\alpha & -\alpha \end{pmatrix}
\end{aligned}
$$

5.4.3 Laplacian of Gaussian

One issue of the edge image is that it increases the noise. In order to suppress the noise one can instead:

- First, smooth the image using a Gaussian filter, which reduces the noise.
- Then, apply the edge enhancement filter.

This is known as the Laplacian of Gaussian short LoG. Let's first have a look at this filter:



Because this looks like a sombrero, the filter is also known as the maxican hat filter.

Mathematically we can write the LoG in a continuous form as

$$\Delta(f(x,y) * h^{\sigma}_{\text{Gaussian}})$$

But instead due to the differentiation rule of the convolution we can write this as

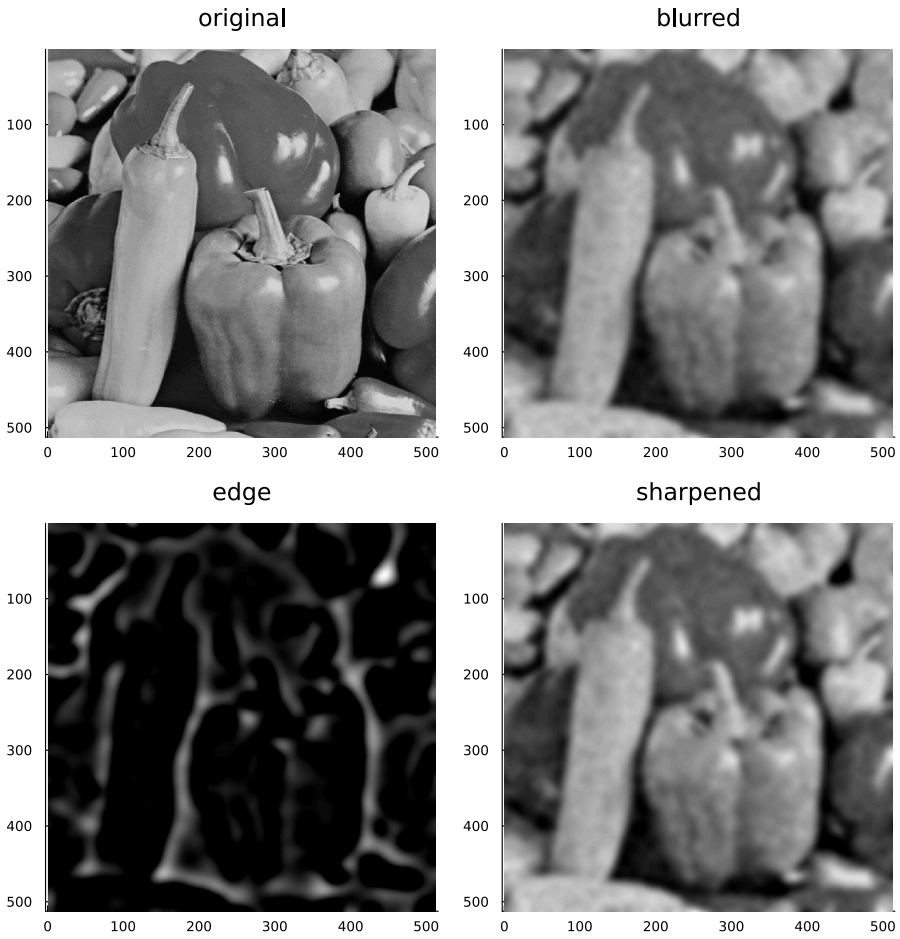
$$f(x,y) * (\Delta h^{\sigma}_{\text{Gaussian}})$$

This has the advantage that we can derive the kernel h_{LoG} analytically:

$$
\begin{aligned}
h_{\text{LoG}} &= \frac{\partial^2 h^{\sigma}_{\text{Gaussian}}(x,y)}{\partial^2 x} + \frac{\partial^2 h^{\sigma}_{\text{Gaussian}}(x,y)}{\partial^2 y} \\
&= \frac{\partial^2}{\partial^2 x} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) + \frac{\partial^2}{\partial^2 y} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= \frac{\partial}{\partial x} \frac{-x}{2\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) + \frac{\partial}{\partial y} \frac{-y}{2\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= \frac{x^2}{2\pi\sigma^6} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) - \frac{1}{2\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) + \\
&\quad \frac{y^2}{2\pi\sigma^6} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) - \frac{1}{2\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
&= -\frac{1}{\pi\sigma^4} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \left(1 - \frac{x^2+y^2}{2\sigma^2}\right)
\end{aligned}
$$

We can use the LoG for image sharpening. Then we have two parameters

σ 10 and α 0.1



5.5 Wrapup

In this lecture we have introduced spatial filtering as a tool for image enhancement. Both image smoothing and image sharpening filters have been discussed. In practice one often applies a combination of several filter, which can be done be applying the filter sequentially.