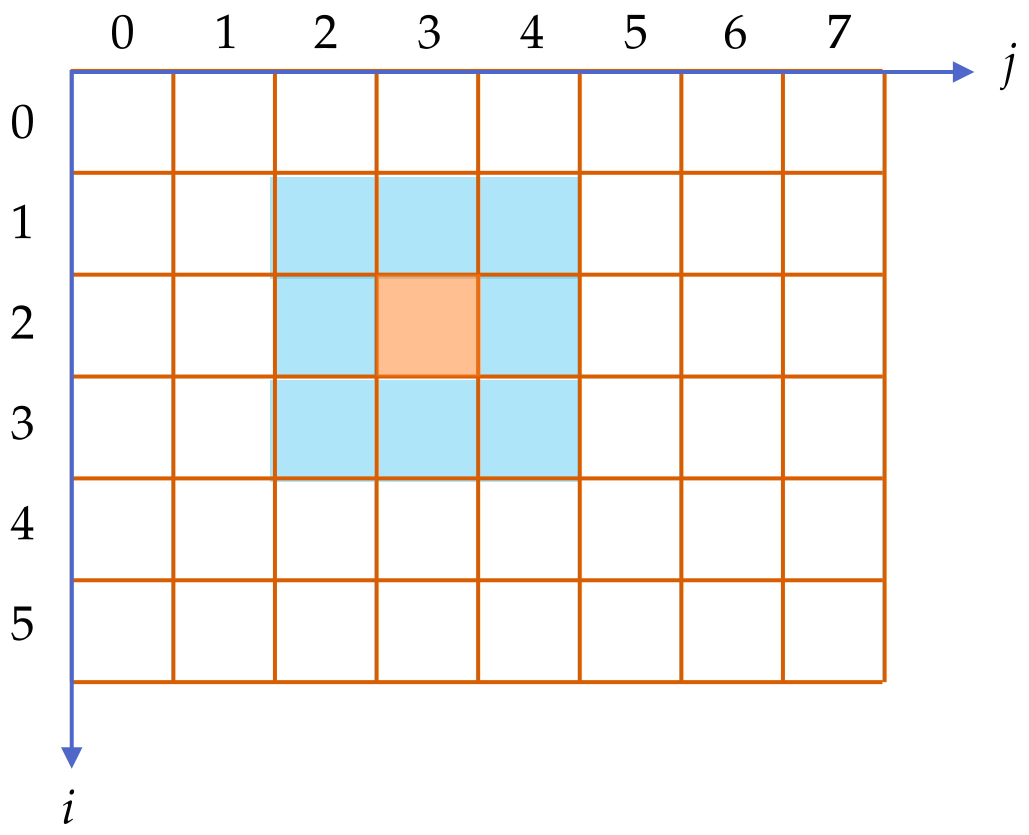
L03 Spatial Filtering

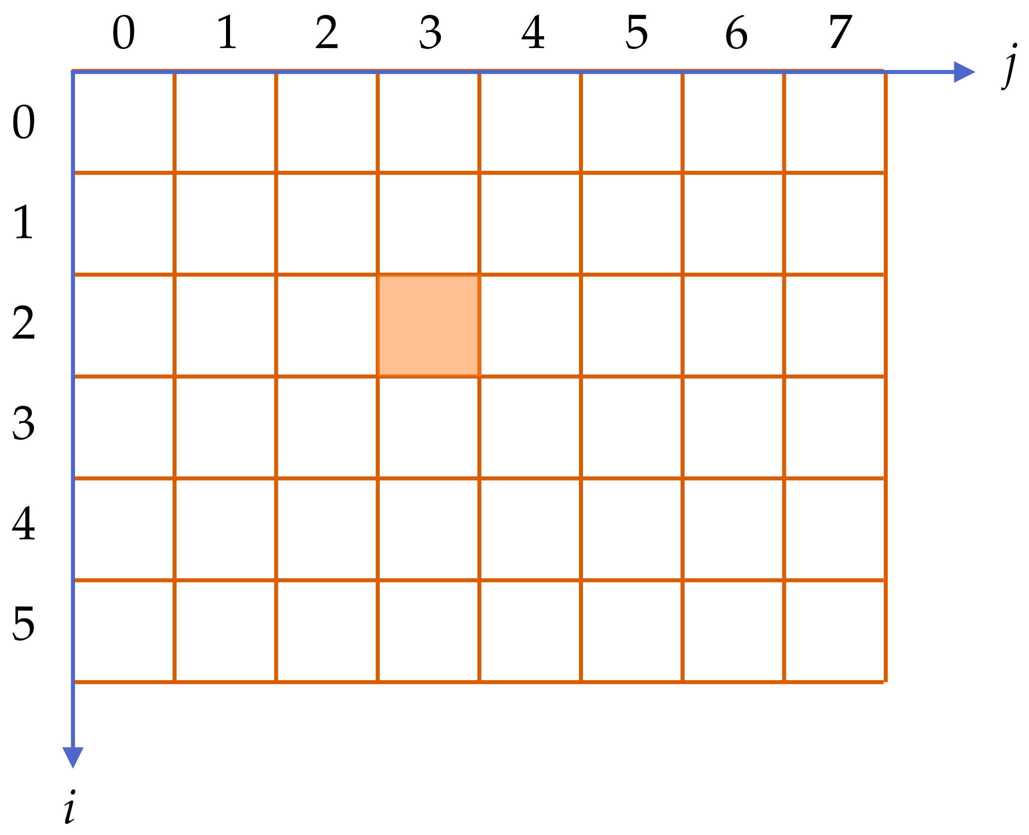
Introduction

- We use spatial filtering to enhance images, e.g., noise filtering.
- We can use the same technique to locate objects in images, called template matching. After completing this lesson, we would be able to process images using filters, as done in popular image processing software such as Photoshop.
- Intensity transformations affect the brightness (intensity level)of an image. The resulting value of a pixel after an intensity operation is just dependent on the original value of the same pixel.
- In contrast, the resulting value of a pixel after a spatial filtering operation depends on the neighborhood of the pixel in question. So, the space around the pixel in question matters, hence, the name spatial filtering.
- In linear spatial filtering we use the 2-D convolution operation.

Spatial Filtering



A 3×3 window (kernel or filter)



Spatial Filtering

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	18	90	90	18	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

i \ j	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Spatial Filtering

$i \backslash j$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	180	90	90	180	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

$i \backslash j$	0	1	2	3	4	5	6	7
0								
1		20	30					
2								
3								
4								
5								

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Spatial Filtering

$i \backslash j$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	180	90	90	180	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

$i \backslash j$	0	1	2	3	4	5	6	7
0								
1		20	30	40				
2								
3								
4								
5								

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Spatial Filtering

$i \backslash j$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	180	90	90	180	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

$i \backslash j$	0	1	2	3	4	5	6	7
0								
1		20	30	40	40			
2								
3								
4								
5								

Spatial Filtering

$i \backslash j$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	180	90	90	180	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

$i \backslash j$	0	1	2	3	4	5	6	7
0								
1		20	30	40	40	40		
2								
3								
4								
5								

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Spatial Filtering

$i \backslash j$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	180	90	90	180	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

$i \backslash j$	0	1	2	3	4	5	6	7
0								
1		0	30	40	40	40	30	
2								
3								
4								
5								

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Spatial Filtering

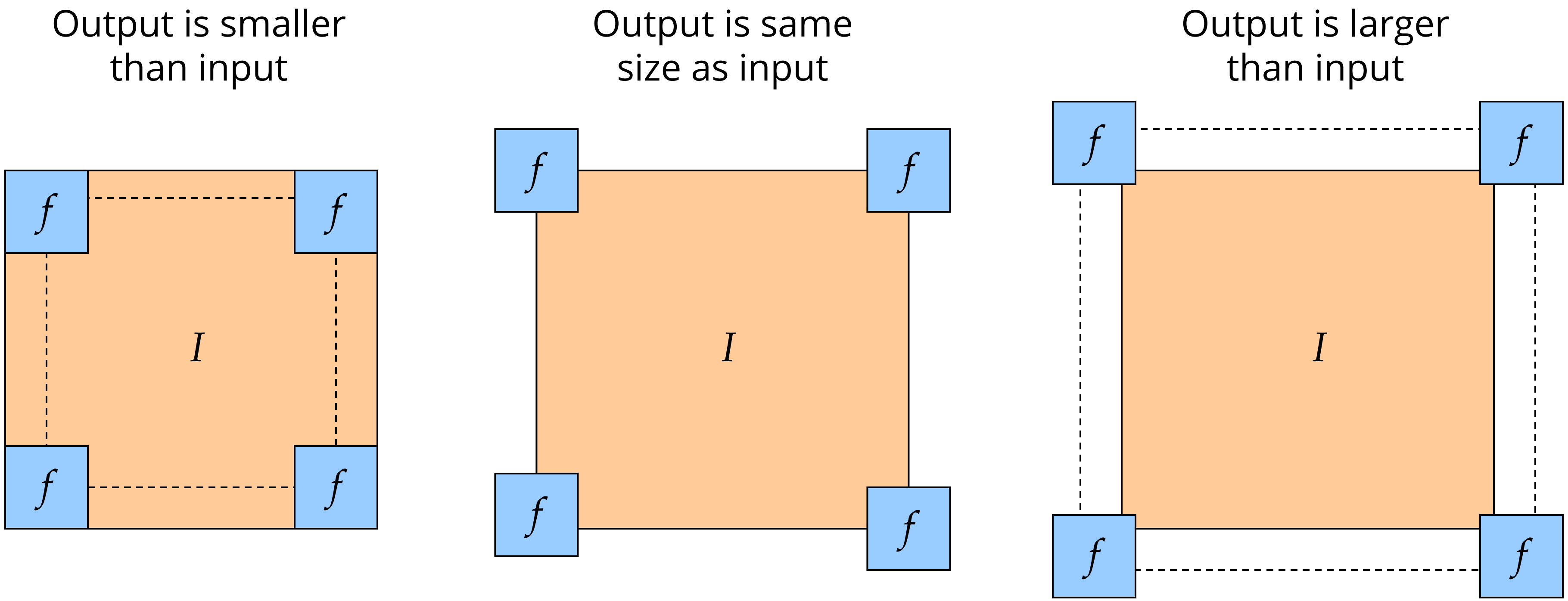
i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	180	90	90	180	90	0
3	0	0	180	90	180	180	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0

i \ j	0	1	2	3	4	5	6	7
0								
1		0	30	40	40	40	30	
2		40	60	90	90	80	50	
3		50	90	90	90	80	50	
4		20	30	50	50	40	20	
5								

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Practical Details: Dealing with Image Boundaries

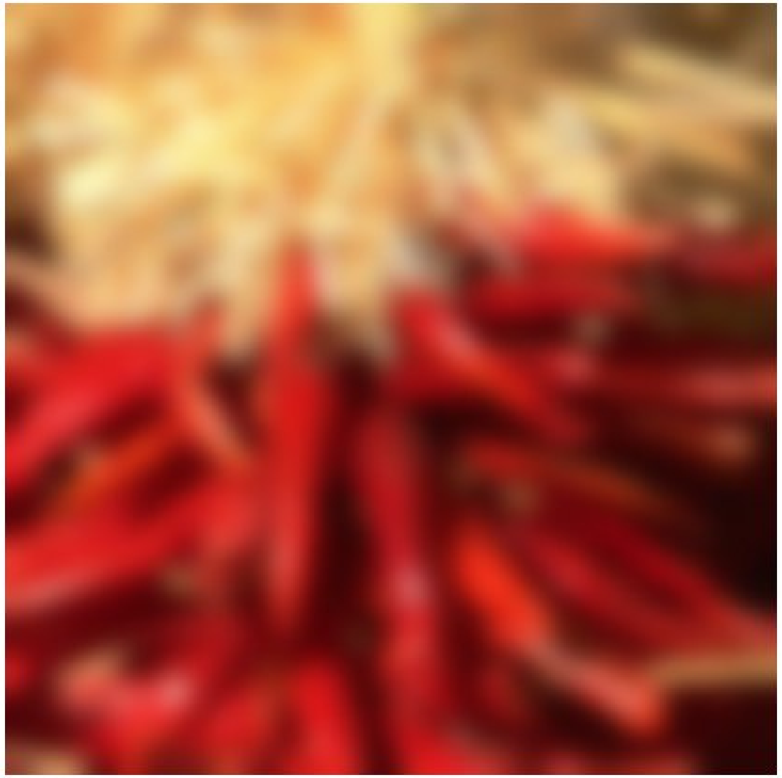
To control the size of the output, we need to use *padding*



Source: Lazebnik with a slight modification.

Practical details: Dealing with Image Boundaries

- To control the size of the output, we need to use *padding*
- What values should we pad the image with?
  - Zero pad (or clip filter)
  - Wrap around
  - Copy edge
  - Reflect across edge



Source: S. Marschner

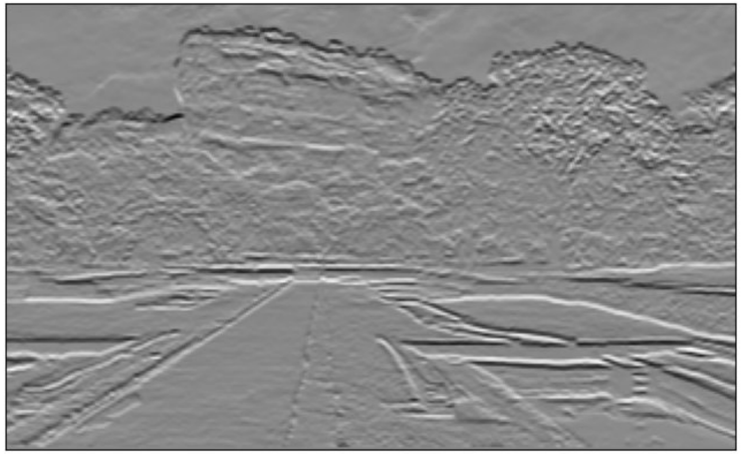
Examples of Effect of Kernel Choices

Original

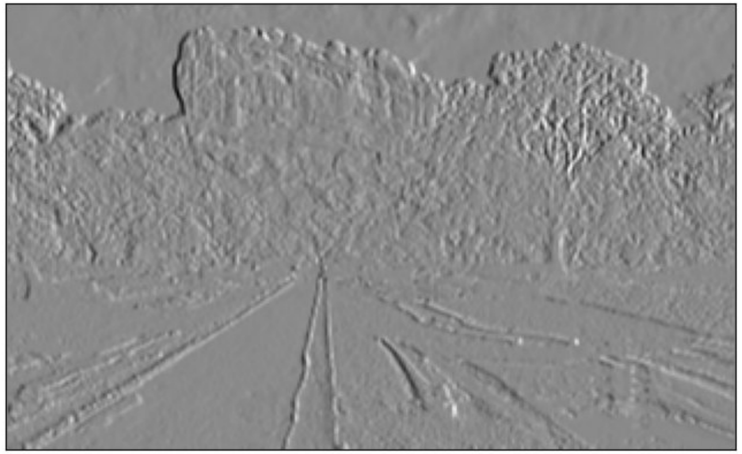
Averaging

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Averaging



Sobel vertical



Sobel horizontal

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

Sobel vertical

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

Sobel horizontal

Averaging Using cv.filter2D

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

im = cv.imread('images/sigiriya.jpg', cv.IMREAD_REDUCED_GRAYSCALE_2)
assert im is not None

kernel = np.ones((3,3),np.float32)/9
imavg = cv.filter2D(im, cv.CV_32F, kernel)

fig, axes  = plt.subplots(1,2, sharex='all', sharey='all', figsize=(18,9))
axes[0].imshow(im, cmap='gray')
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])
axes[1].imshow(imavg, cmap='gray')
axes[1].set_title('Averaging')
axes[1].set_xticks([]), axes[1].set_yticks([])
plt.show()
```

Sobel Filtering Using cv.filter2D

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

im = cv.imread('images/sigiriya.jpg', cv.IMREAD_REDUCED_GRAYSCALE_2)
assert im is not None

sobel_x = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
sobel_y = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

im_x = cv.filter2D(im, cv.CV_64F, sobel_x)
im_y = cv.filter2D(im, cv.CV_64F, sobel_y)

fig, ax  = plt.subplots(1,2, sharex='all', sharey='all', figsize=(18,9))
ax[0].imshow(im_x, cmap='gray')
ax[0].set_title('Sobel X')
ax[0].set_xticks([]), ax[0].set_yticks([])
ax[1].imshow(im_y, cmap='gray')
ax[1].set_title('Sobel Y')
ax[1].set_xticks([]), ax[1].set_yticks([])
plt.show()
```

Convolution and Correlation

- Spatial filtering is, in fact, convolution.
- As the filters are typically symmetric—i.e., a 180° rotation results in the same kernel—correlation is equivalent to convolution.
- Correlation is also the scalar product between the kernel and the underlying image patch. Therefore, it is a measure of similarity between the kernel and the underlying image patch.
- As a result, when the kernel and the patch are "similar", the output is high. In view of this, spatial filtering seeks for patches in the image that are similar to the kernel.
- Implementing filtering using loops (four nested for loops) in a non-C fashion is inefficient. Instead, use filter2D.

Convolution and Correlation

The convolution sum expression that we leaned in signal and systems is

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k].$$

In 2-D, as applicable in image processing, collation sum with a kernel $w[m, n]$ with non-zero values in $(m, n) \in ([-a, a], [-b, b])$ is

$$(w * f)[m, n] = w[m, n] * f[m, n] = \sum_{s=-a}^{a} \sum_{k=-b}^{b} w[s, t]f[m-s, n-t].$$

Correlation:

$$(w \circledast f)[m, n] = w[m, n] \circledast f[m, n] = \sum_{s=-a}^{a} \sum_{k=-b}^{b} w[s, t]f[m+s, n+t].$$

Example

Consider the image

$$f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the filtering kernel

$$w = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

. Appropriately pad the image. Carry out a. correlation b. convolution.

Example

Consider the image

$$f_{\text{padded}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad w = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Correlation result and convolution result, respectively

$$(w \circledast f) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 8 & 7 & 0 \\ 0 & 6 & 5 & 4 & 0 \\ 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (w * f) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Convolution: Key Properties

1. Linearity: $\text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2)$
2. Shift invariance: same behavior regardless of pixel location:
   $\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$
3. Theoretical result: any linear shift-invariant operator can be represented as a convolution.

Other Properties

1. Commutative: $a * b = b * a$
2. Conceptually no difference between filter and signal
3. Associative: $a * (b * c) = (a * b) * c$
4. Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
5. This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
6. Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
7. Scalars factor out: $ka * b = a * kb = k(a * b)$
8. Identity: unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$, $a * e = a$

At the Edge

The filter window falls off the edge of the image.

Need to extrapolate the image.

Methods: various border types, image boundaries are denoted with ’|’

- BORDER_REPLICATE: aaaaaa|abcdefgh|hhhhhhh
- BORDER_REFLECT: fedcba|abcdefgh|hgfedcb
- BORDER_REFLECT_101: gfedcb|abcdefgh|gfedcba
- BORDER_WRAP: cdefgh|abcdefgh|abcdefg
- BORDER_CONSTANT: iiiiii|abcdefgh|iiiiiii with some specified ’i’

Sharpening

(a) Original

$-$

(b) Smoothed

$=$

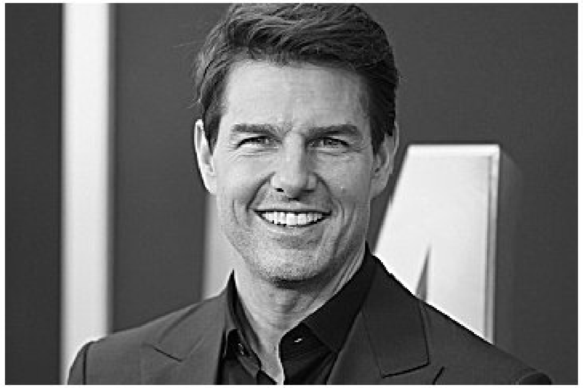
(c) Original – Smoothed

(d) Original

$+$

(e) Original – Smoothed

$=$

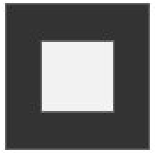


(f) Sharpened

Box Filter vs. Gaussian Filter

- What's wrong with this filtering operation?
- What's the solution?

(a) Original



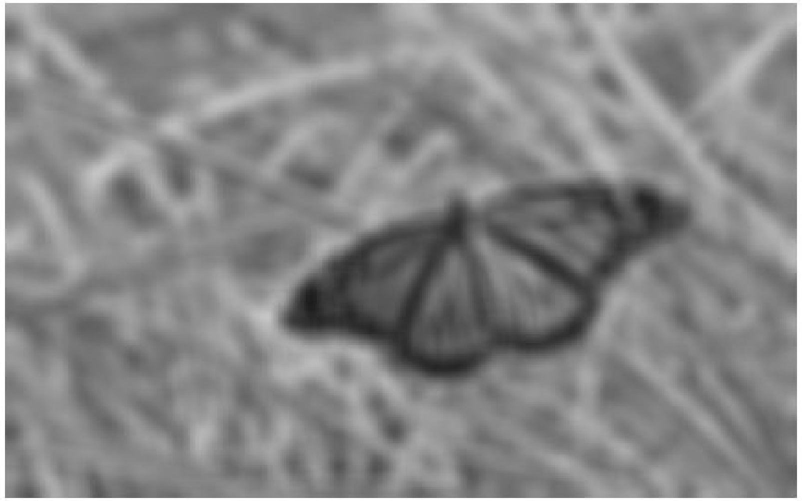
(b) Kernel (much zoomed)

(c) Box Filtered

Box Filter vs. Gaussian Filter

(a) Original

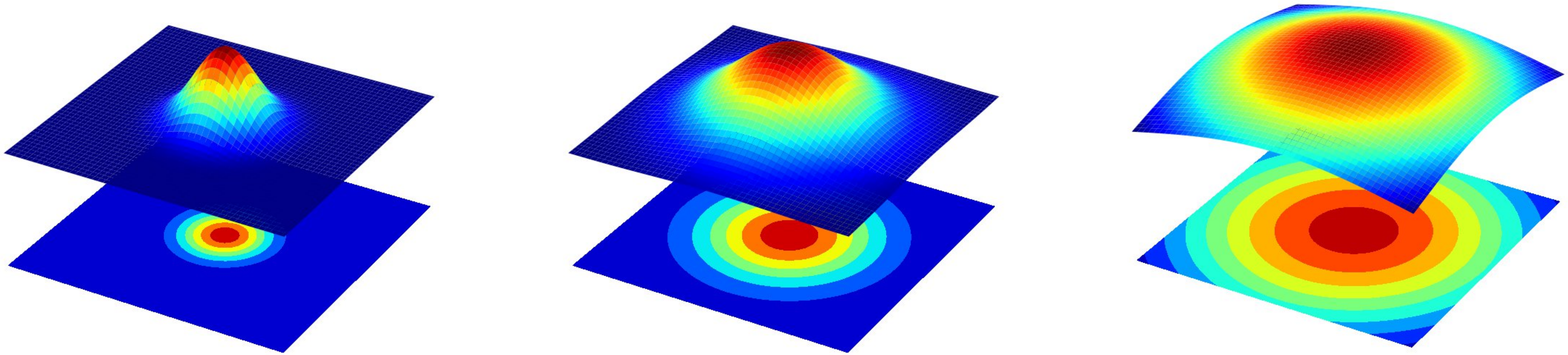
(b) Box Filtered



(c) Gaussian Filtered

Gaussian Kernel

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

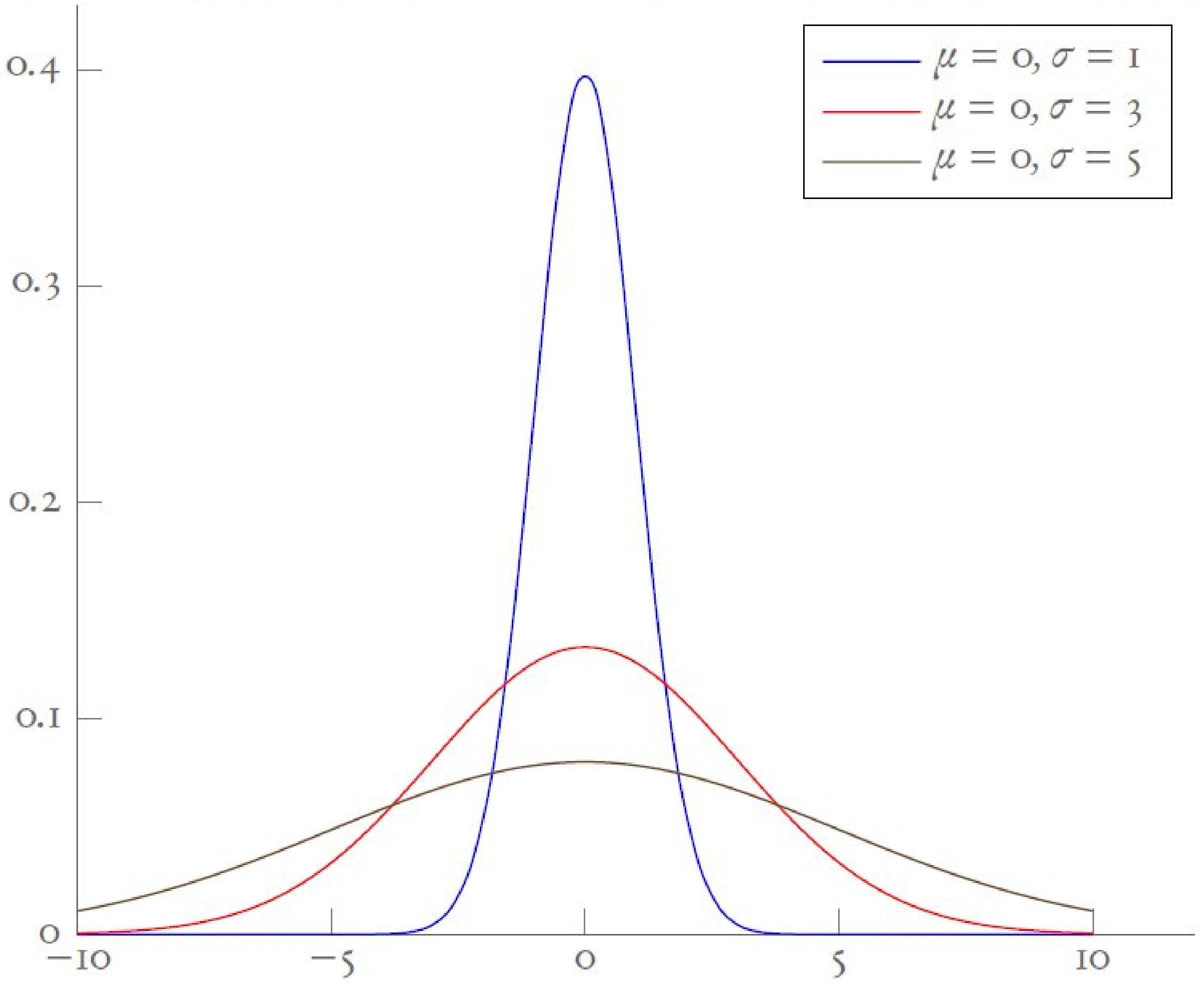


- Constant factor at front makes volume sum to 1 (can be ignored when computing the filter values, as we should renormalize weights to sum to 1 in any case).
- The Gaussian function has infinite support, but discrete filters use finite kernels.

Choosing Gaussian Kernel Width

$$g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

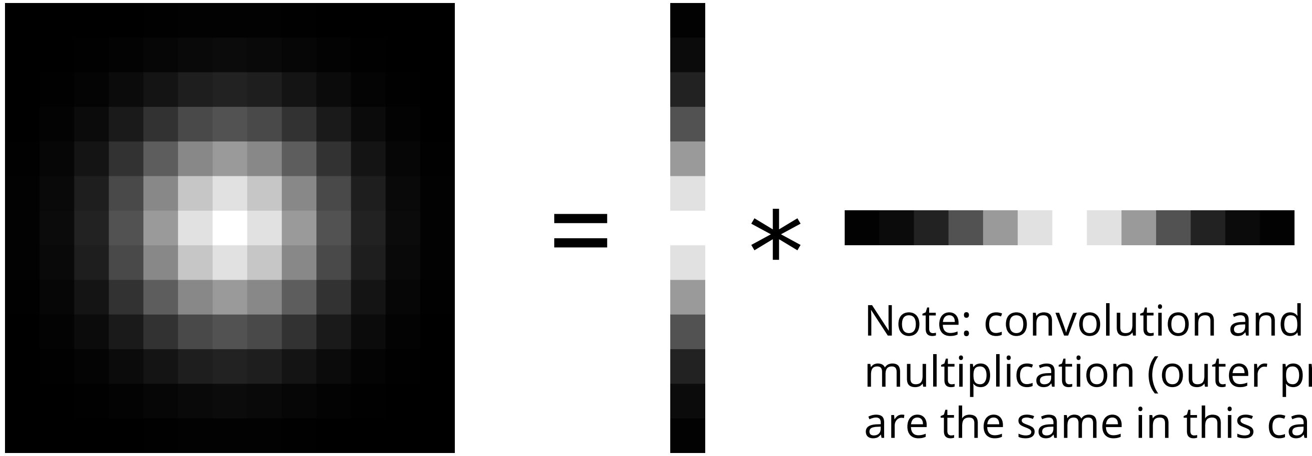
Rule of thumb: set the filter half-width to about 3σ.



Separability of the Gaussian Filter

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

$$\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right) \quad \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{y^2}{2\sigma^2}\right)$$



Note: convolution and matrix multiplication (outer product) are the same in this case

- The 2-D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y.
- In this case, the two Gaussians are the (identical) 1-D Gaussians.
- Separability means that a 2-D convolution can be reduced to two 1-D convolutions (one among rows and one among columns).
- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?

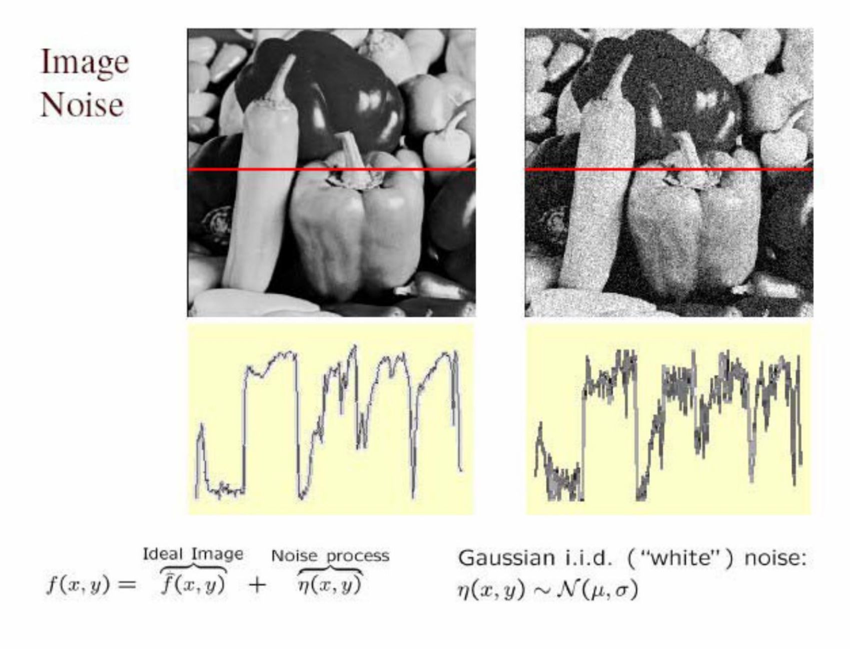
$$O(n^2m^2).$$

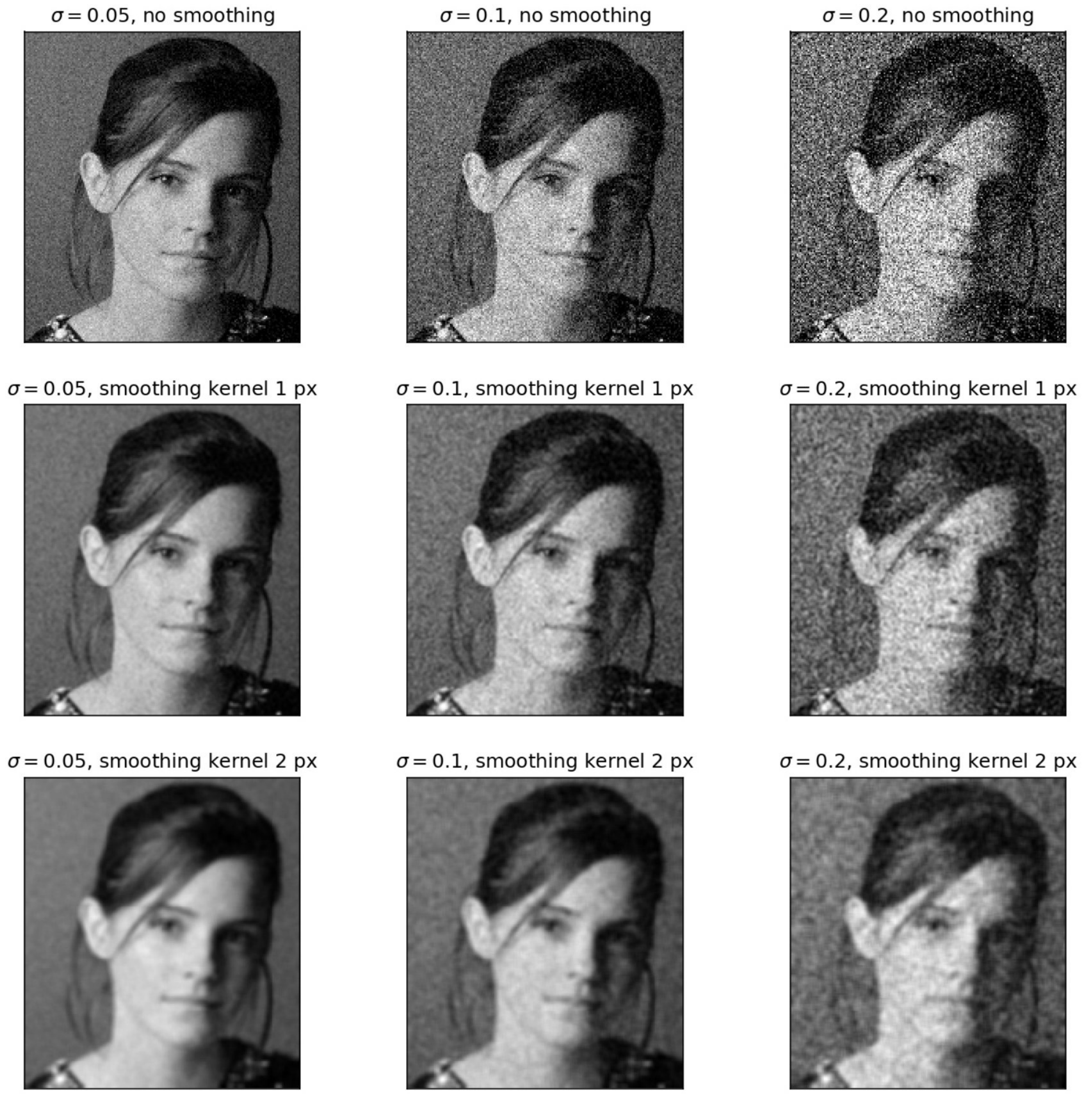
- What is the complexity if the kernel is separable?

$$O(n^2m)$$

Gaussian Noise

- Mathematical model: sum of many independent factors
- Good for small standard deviations
- Assumption: independent, zero-mean noise

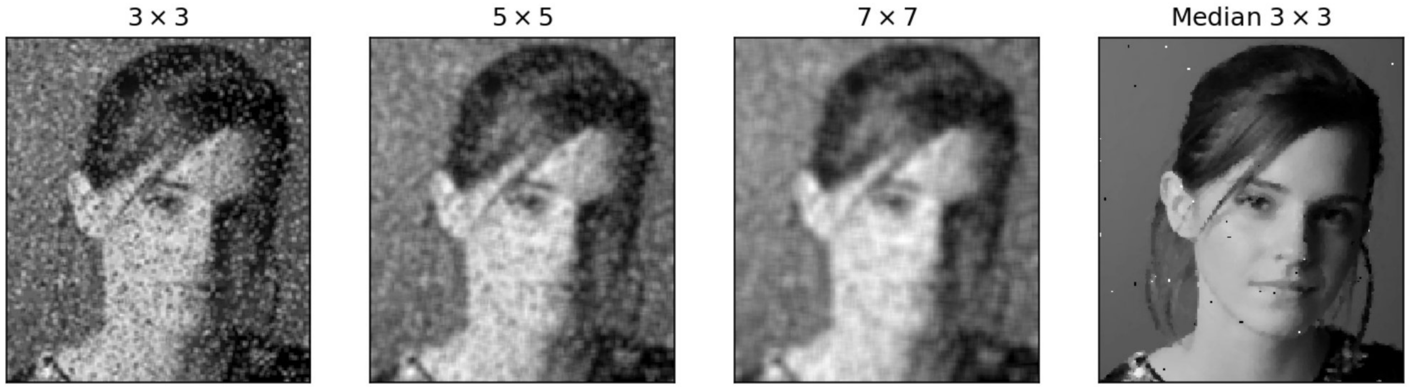




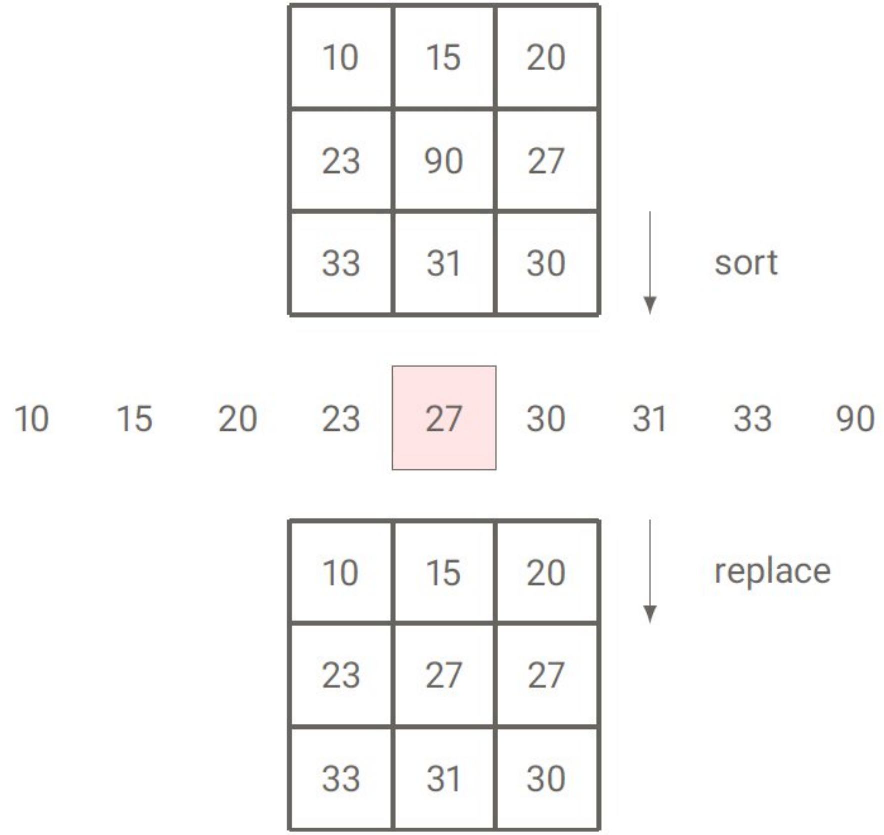
Reducing Gaussian Noise
Smoothing with larger standard deviations suppresses noise but also blurs the image.

Source: M. Hebert

Reducing Salt-and-Pepper Noise: Median Filtering



Inability to reduce salt and pepper noise with Gaussian filtering



A median filter operates over a window by selecting the median intensity in the window.
Is median filtering linear?

Source: K. Grauman

Bilateral Filter

What is the Bilateral Filter?

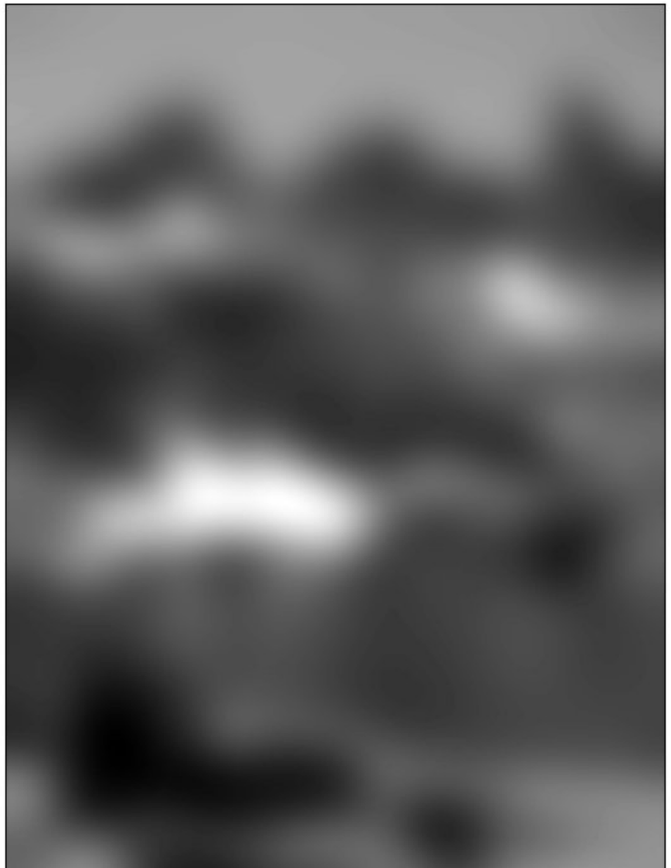
- A non-linear, edge-preserving, and noise-reducing smoothing filter.
- Combines domain (spatial) and range (intensity) filtering.
- Useful in denoising while preserving edges, unlike Gaussian smoothing.

Mathematical Formulation

$$I_{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} G_{\sigma_s}\left(\|x_i - x\| \right) \cdot G_{\sigma_r}\left(\|I(x_i) - I(x)\| \right)$$

Original

Gaussian Filtered



Bilateral Filtered