Image Processing & Vision

Lecture 02: Image Filtering

Hak Gu Kim

hakgukim@cau.ac.kr

Immersive Reality & Intelligent Systems Lab (IRIS LAB)

Graduate School of Advanced Imaging Science, Multimedia & Film (GSAIM)

Chung-Ang University (CAU)

13 Mar. 2023

IRIS LAB
Immersive Reality & Intelligent Systems Lab.

Topics

- Image Transformation
- Image Filtering: Linear Filters
- Image Filtering: Non-linear Filters

***Note:** Many of these slides in this course were adapted from Computer Vision at CMU (16-385) and UBC (CPSC425)

Topics

- Image Transformation
- Image Filtering: Linear Filters
- Image Filtering: Non-linear Filters

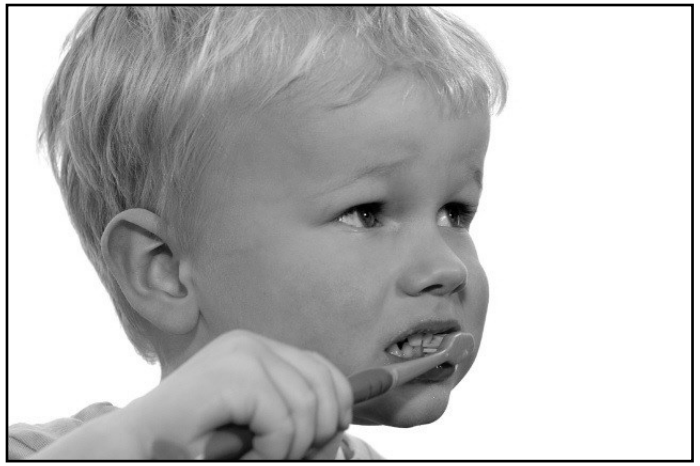
Image as a 2D Function

- An image is a 2D function

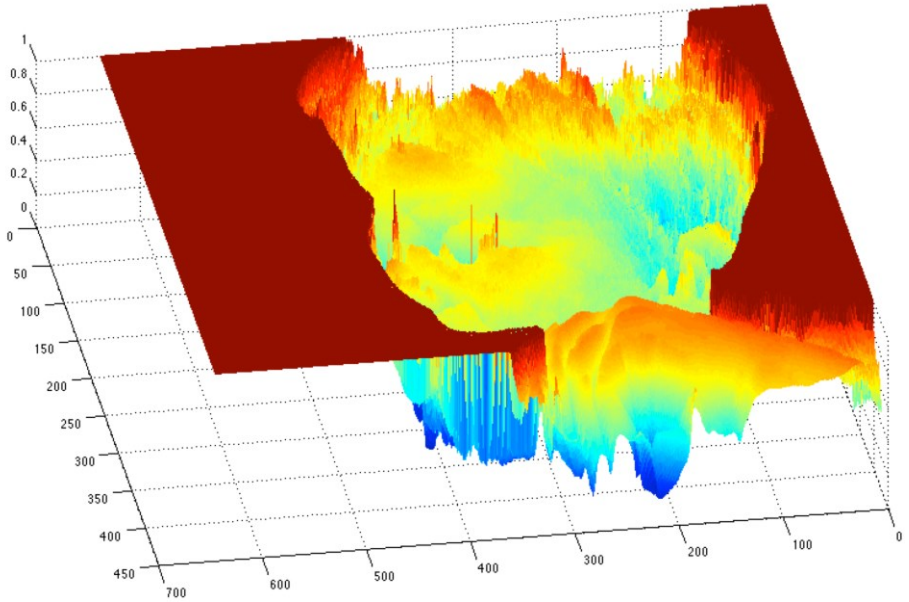
— Image function: $I(x, y)$

— The range of the image function: $I(x, y) \in [0, 255]$

— The domain of the image function: $(x, y) \in ([1, Width],\ [1, Hight])$



A gray scale image



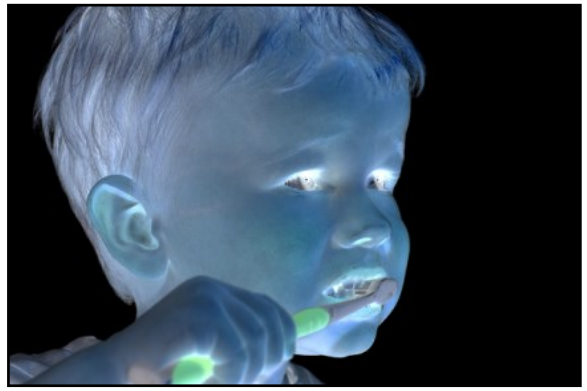
A function in 2D domain

What Types of Image Transformations?

$I(x, y)$

Filtering

$I'(x, y)$

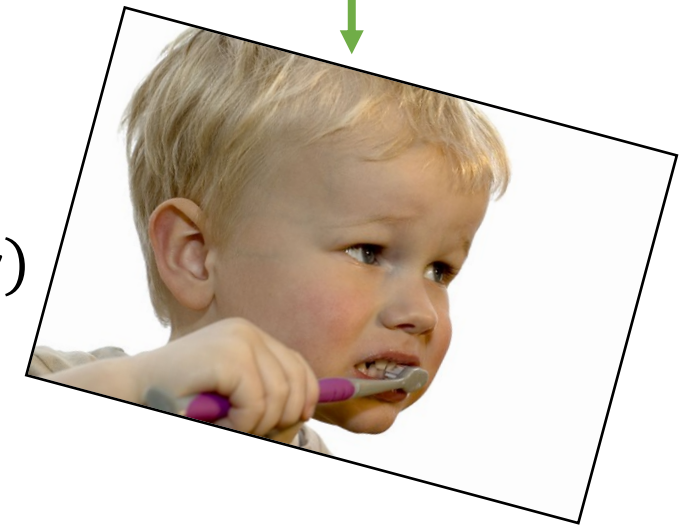


Changes pixel ***values***

$I(x, y)$

Warping

$I'(x, y)$



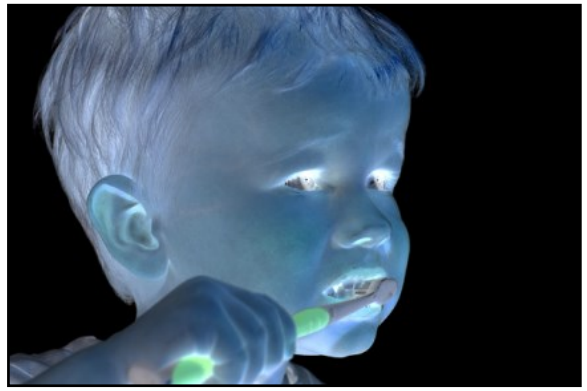
Changes pixel ***locations***

What Types of Image Transformations?

$I(x, y)$

Filtering

$I'(x, y)$

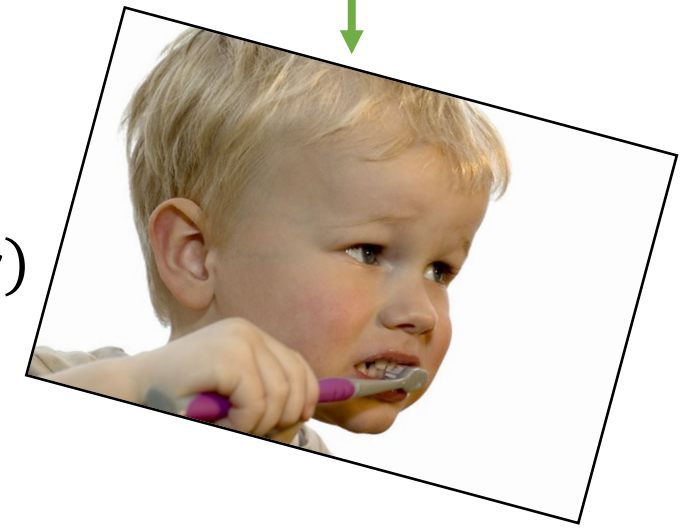


Changes **range** of image function

$I(x, y)$

Warping

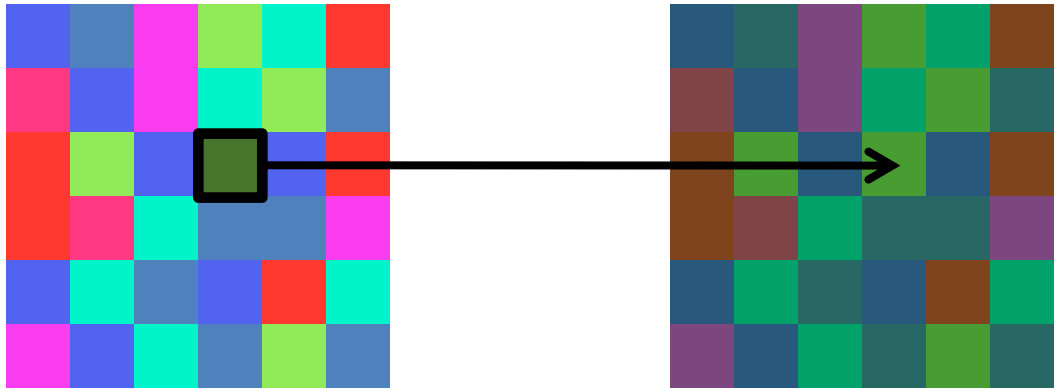
$I'(x, y)$



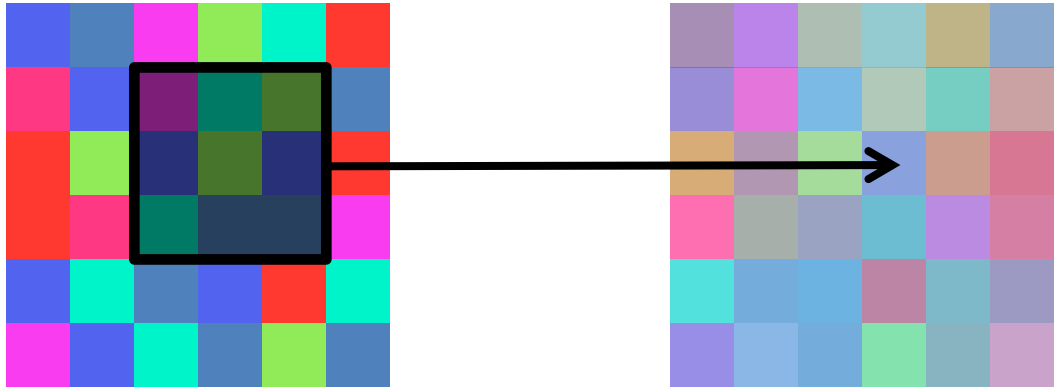
Changes **domain** of image function

What Types of Filtering?

- **Point** operation: Point processing

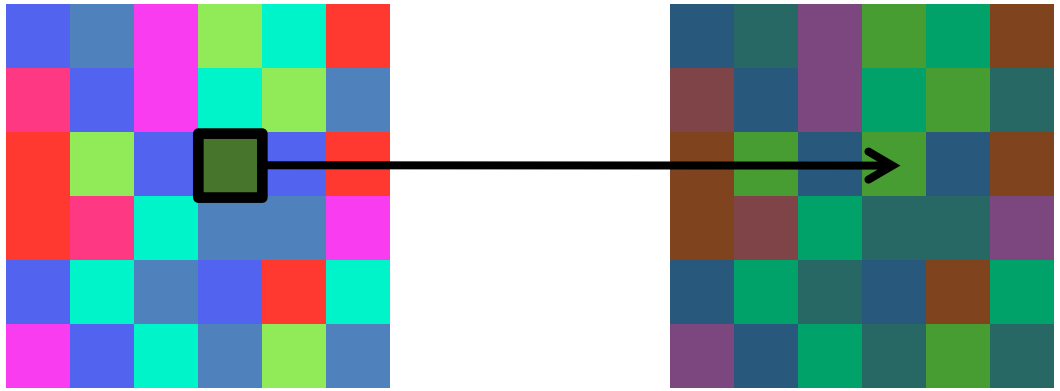


- **Neighborhood** operation: Filtering



What Types of Filtering?

- **Point** operation: Point processing



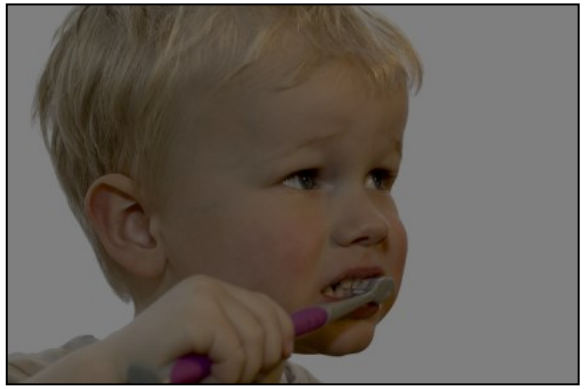
- **Neighborhood** operation: Filtering

Examples of **Point Processing**

Original

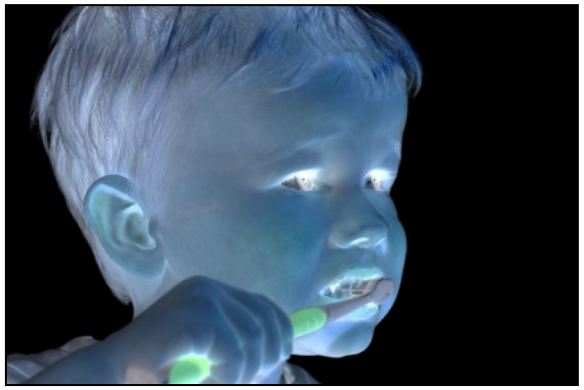
Darken

Lower contrast



Non-linear lower contrast

Invert



Lighten

Higher contrast

Non-linear higher contrast

Examples of **Point Processing**

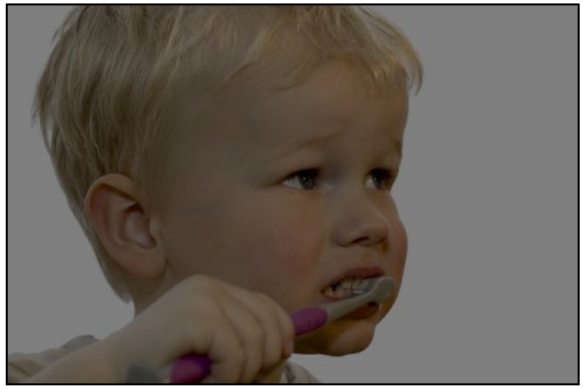
Original

$I(x,y)$

Darken

$I(x,y) - 128$

Lower contrast

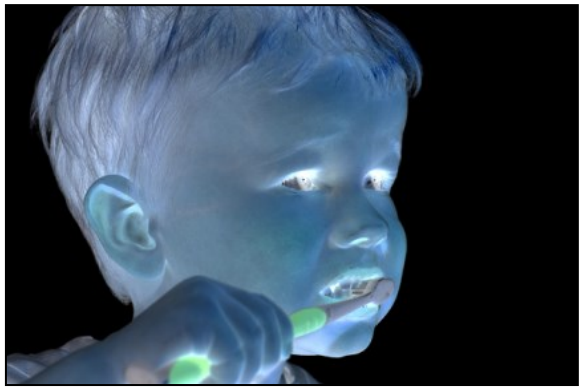


$\frac{I(x,y)}{2}$

Non-linear lower contrast

$\left(\frac{I(x,y)}{255}\right)^{1/3} \times 255$

Invert



$255 - I(x,y)$

Lighten

$I(x,y) + 128$

Higher contrast

$I(x,y) \times 2$

Non-linear higher contrast

$\left(\frac{I(x,y)}{255}\right)^{2} \times 255$

Examples of **Point Processing**

Image from camera

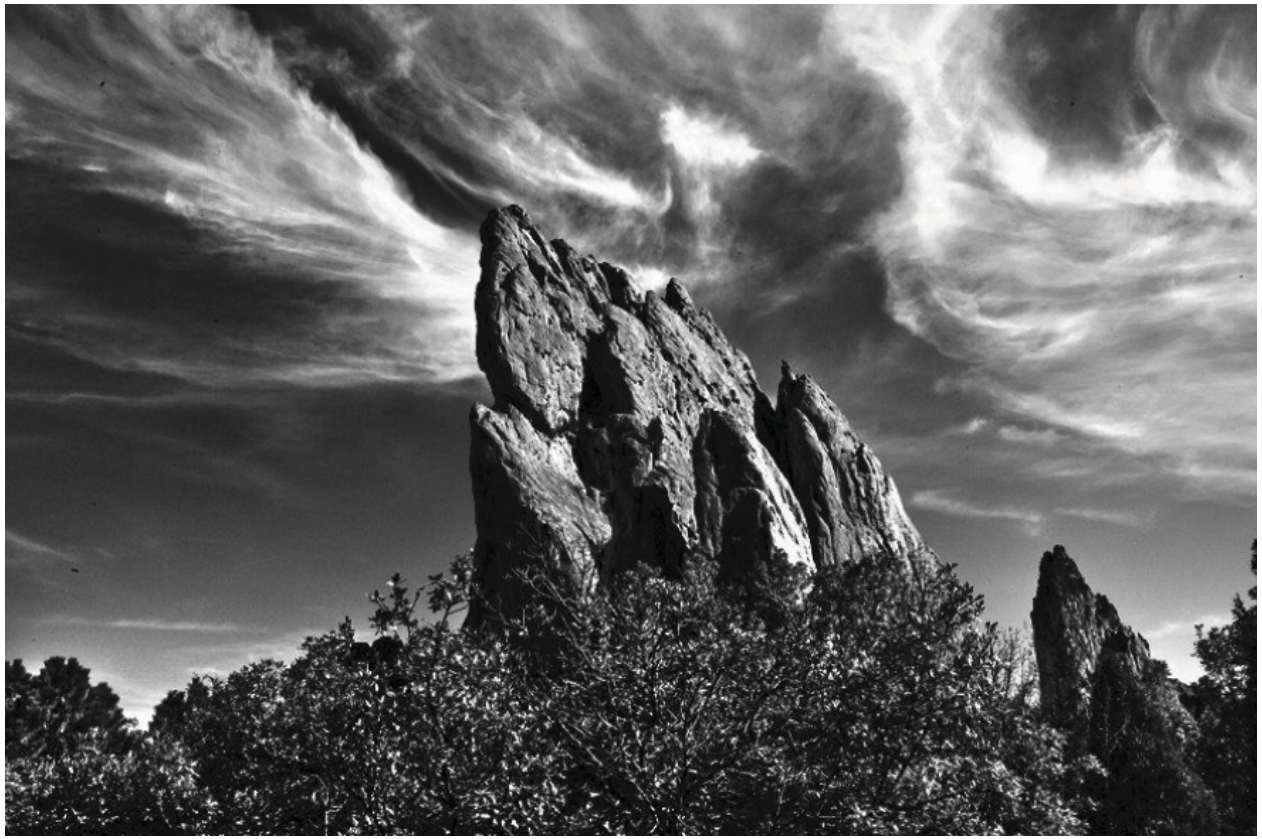


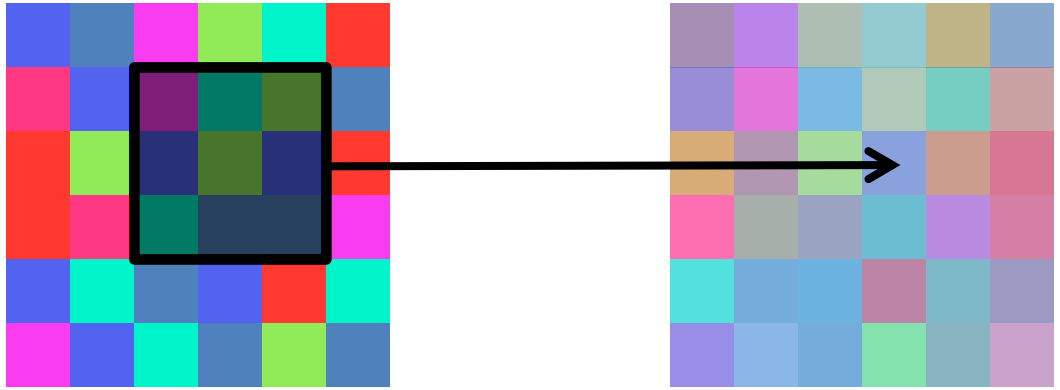
Image after tone mapping

S. Bae et al., Two-scale tone management for photographic look, **ACM SIGGRAPH 2006**

What Types of Filtering?

- **Point** operation: Point processing

- **Neighborhood** operation: Filtering



Topics

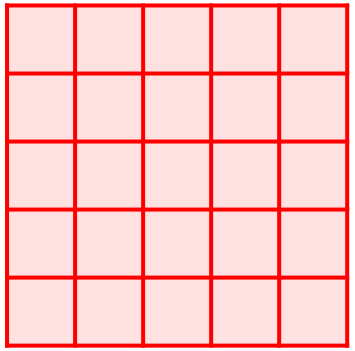
- Image Transformation

- Image Filtering: Linear Filters

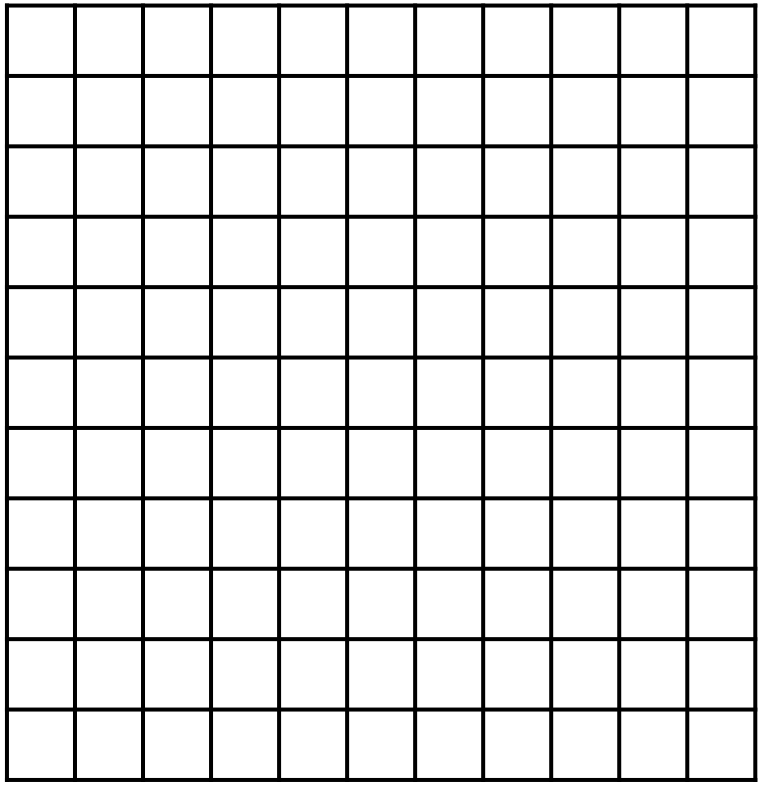
- Image Filtering: Non-linear Filters

Linear Filters

- Let $I(x, y)$ be an $n \times n$ digital image (for convenience, we let *width=height*)
- Let $F(x, y)$ be an $m \times m$ digital image, **filter** or **kernel**



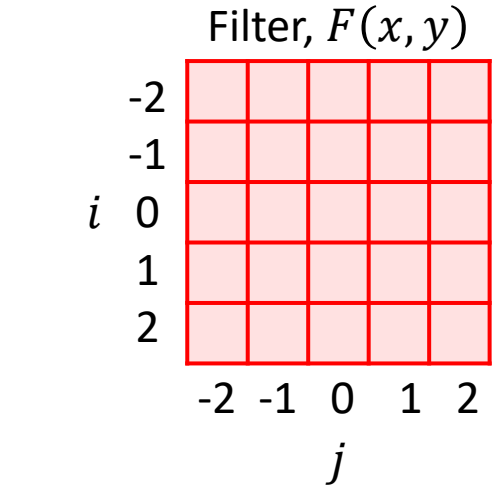
Filter, $F(x, y)$



Image, $I(x, y)$

Linear Filters

- **Intuition:** Each pixel in the output image is a linear combination of the same index pixel and its neighboring pixels in the original image



- Compute **the filtered image, $I'(x, y)$:**

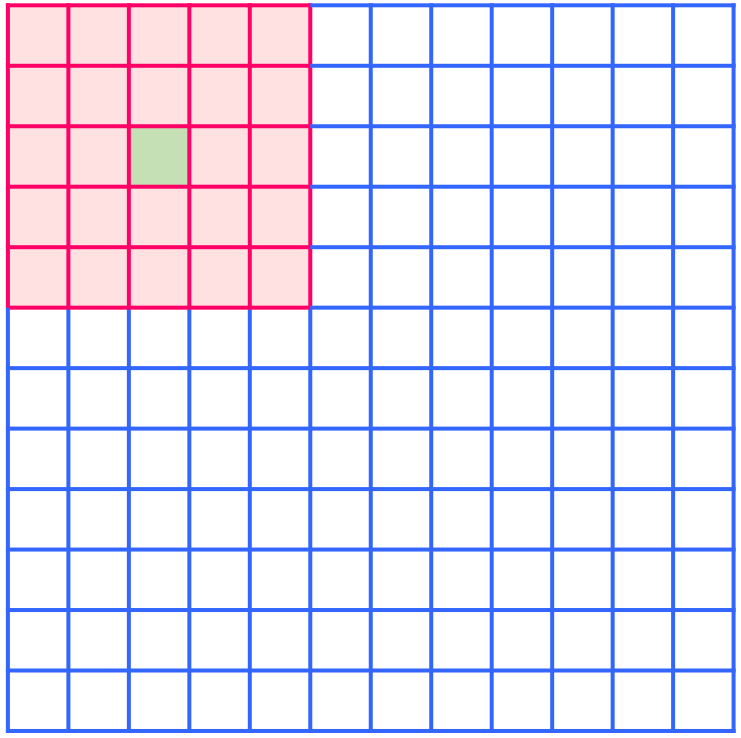
— Let $k = \left\lfloor \frac{m}{2} \right\rfloor$

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

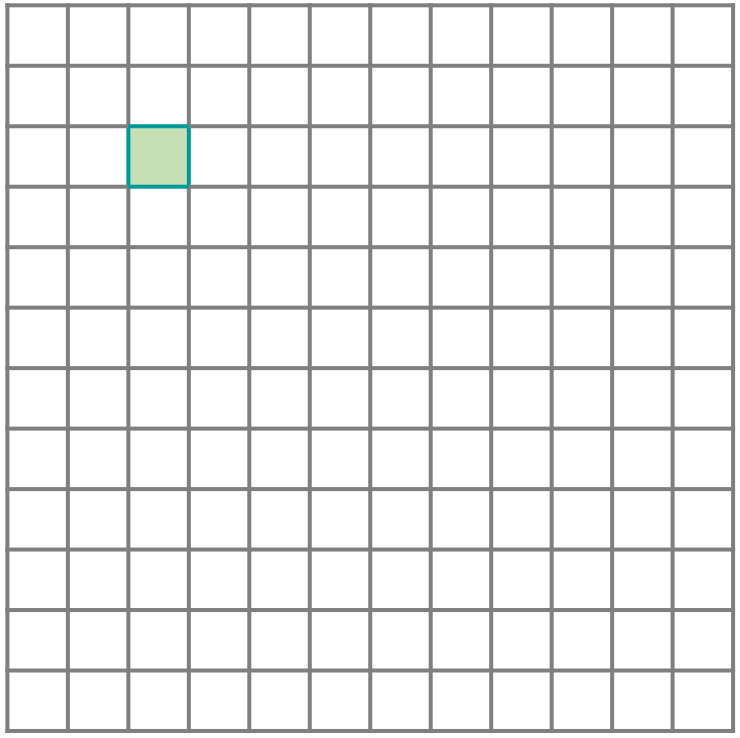
Output (Filtered image) | Filter | Image

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



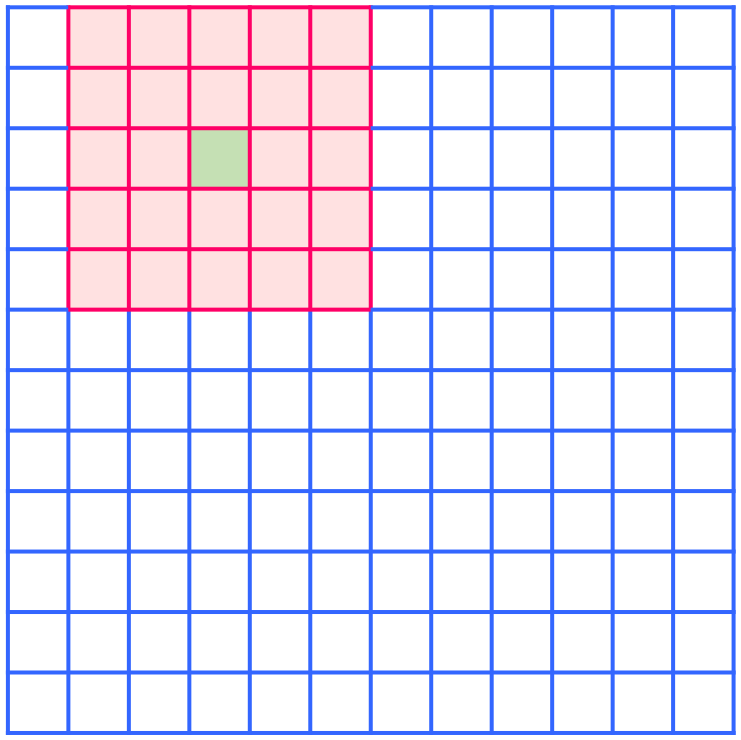
Input, $I(x, y)$



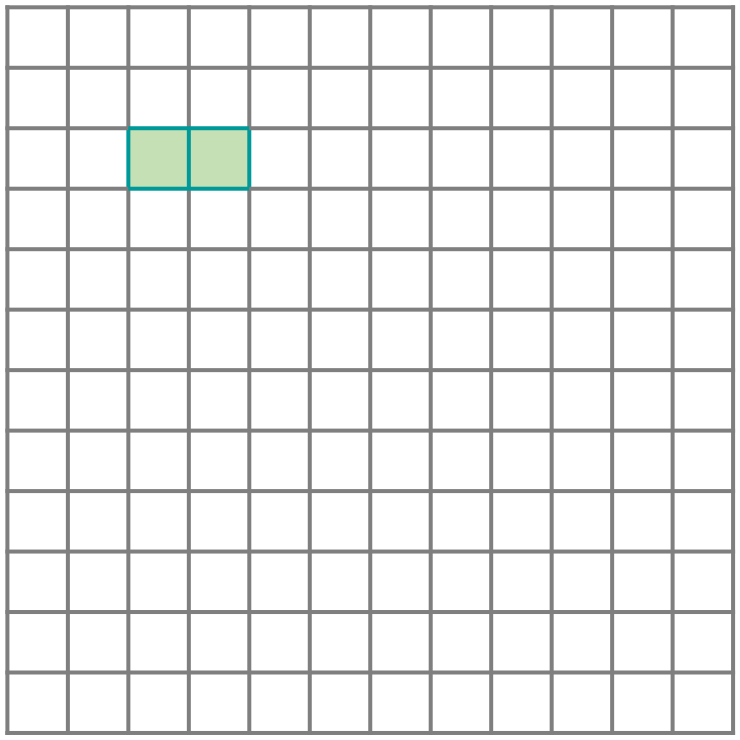
Output, $I'(x, y)$

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



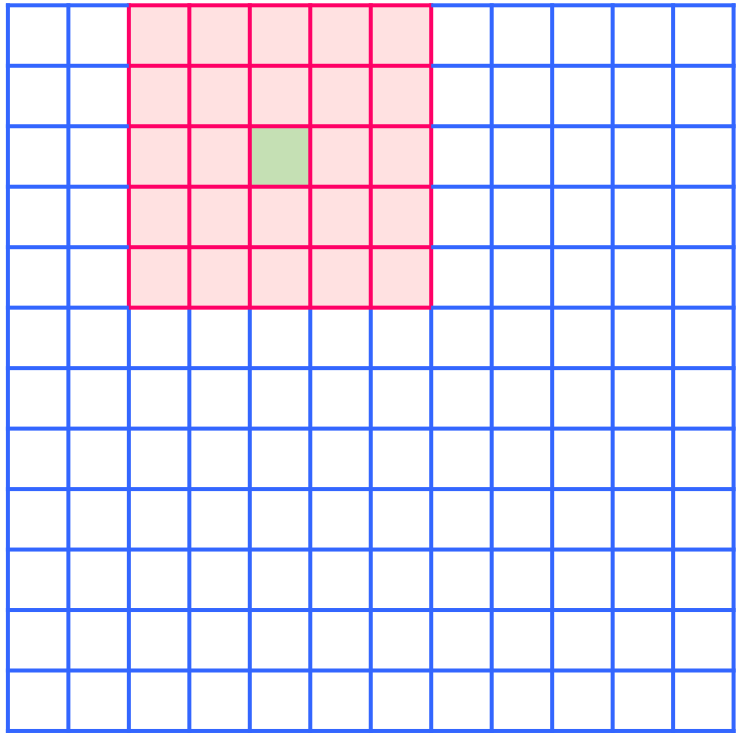
Input, $I(x, y)$



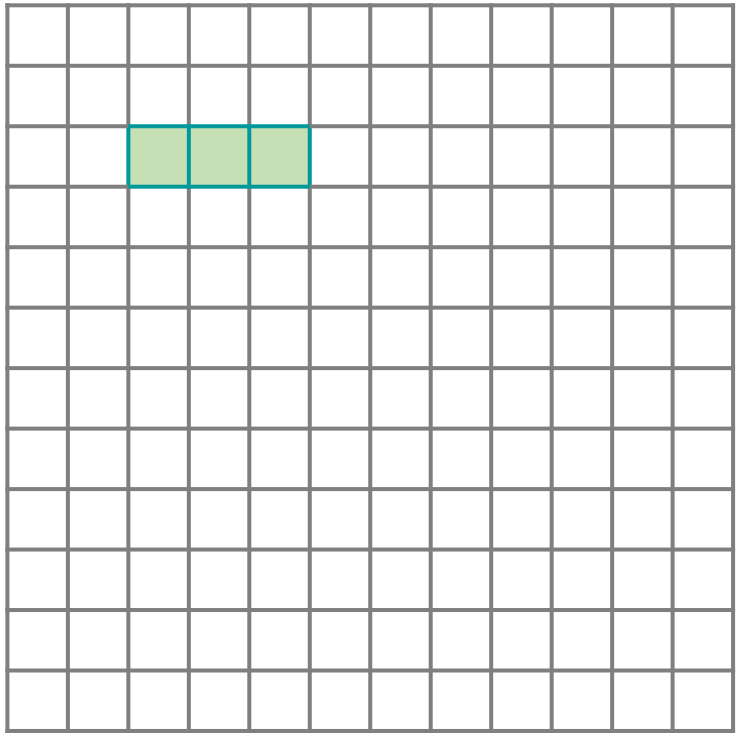
Output, $I'(x, y)$

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



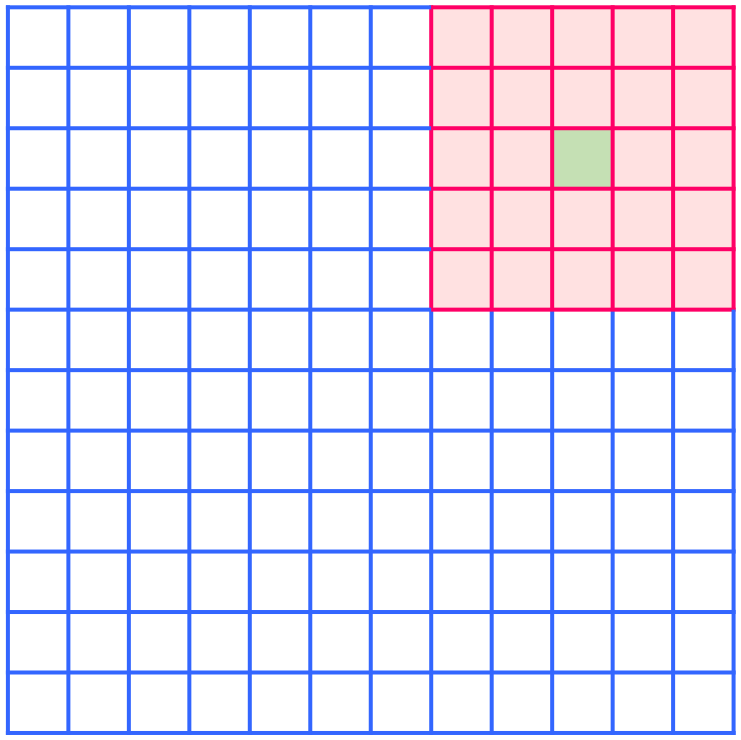
Input, $I(x, y)$



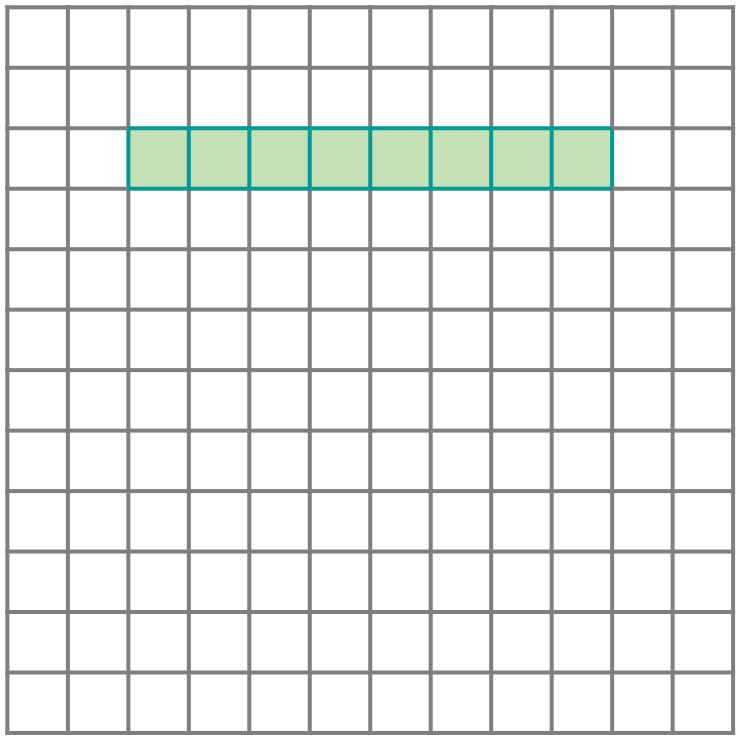
Output, $I'(x, y)$

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



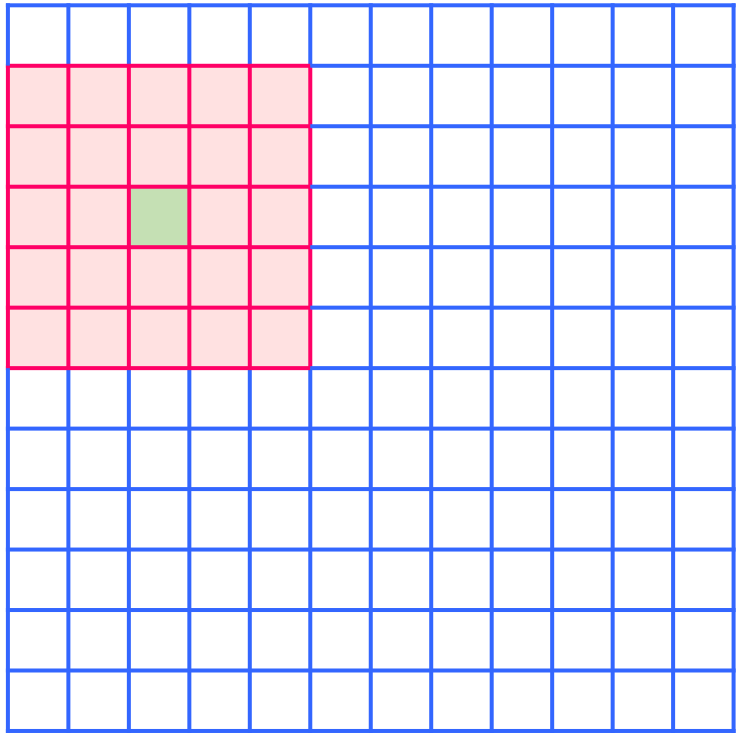
Input, $I(x, y)$



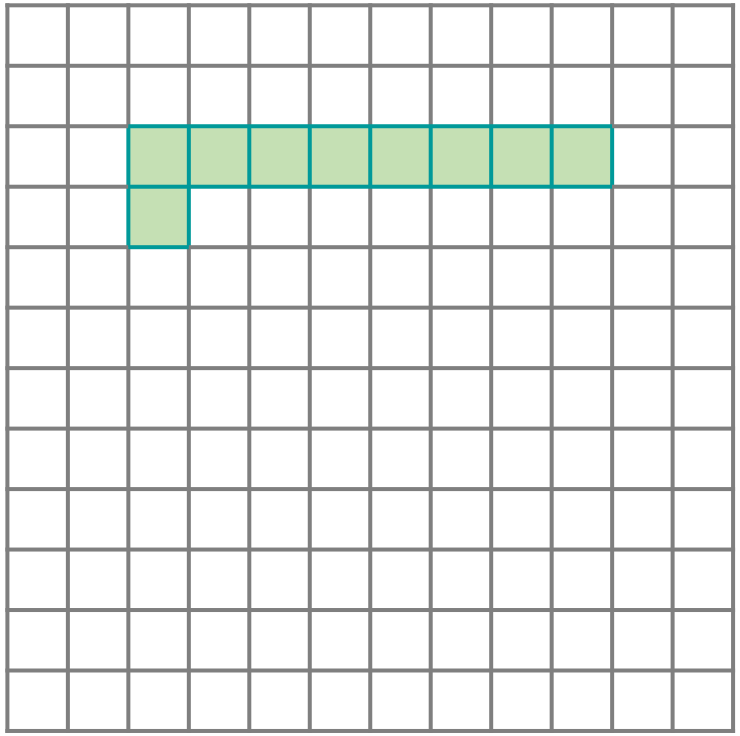
Output, $I'(x, y)$

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



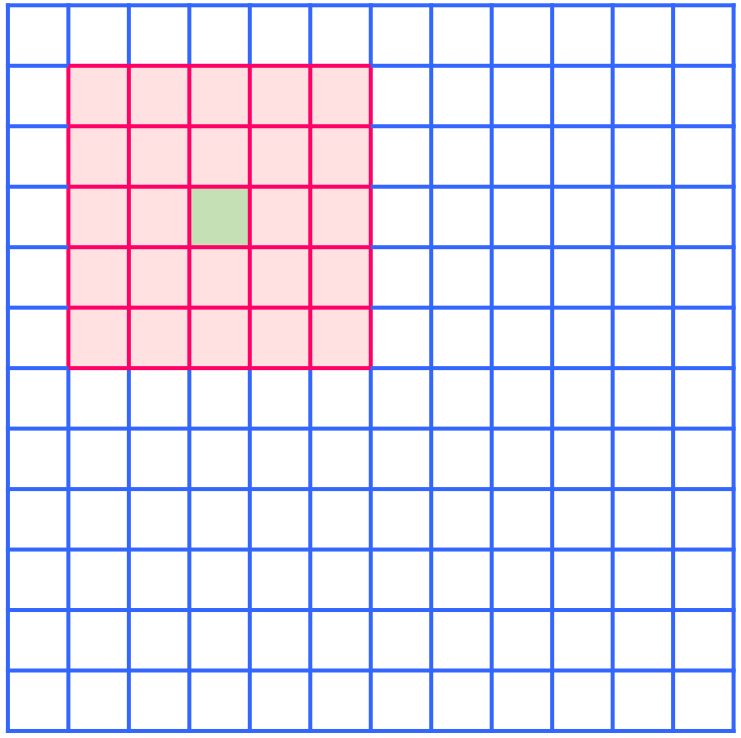
Input, $I(x, y)$



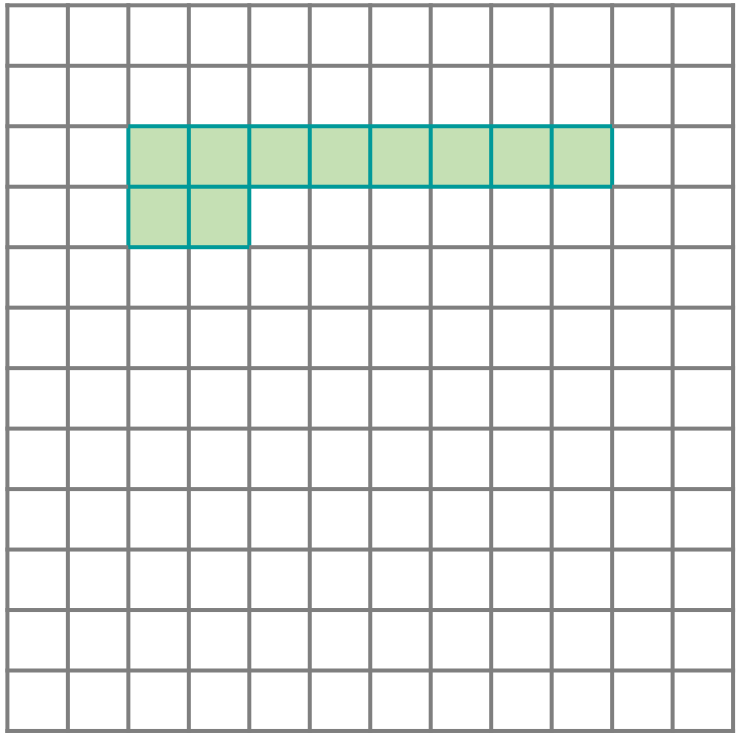
Output, $I'(x, y)$

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



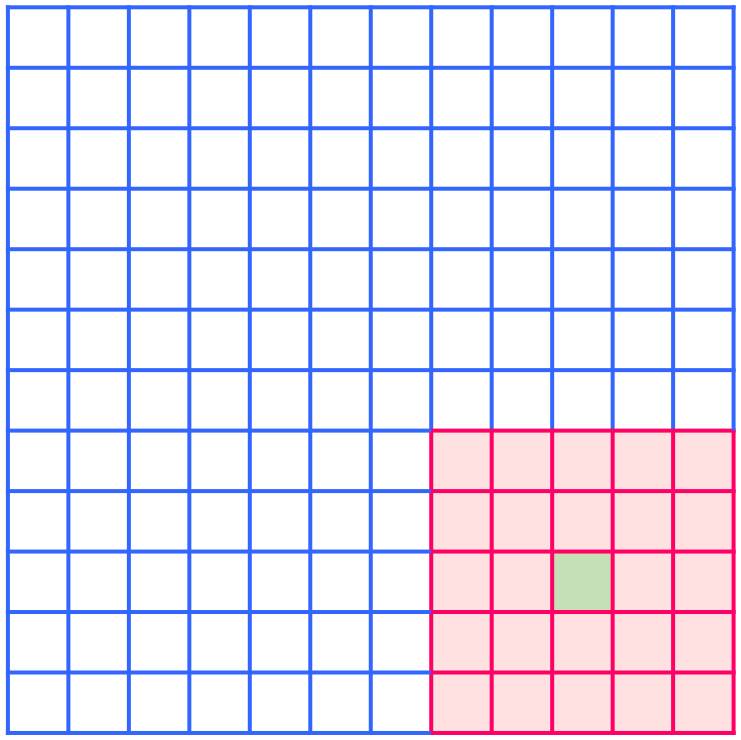
Input, $I(x, y)$



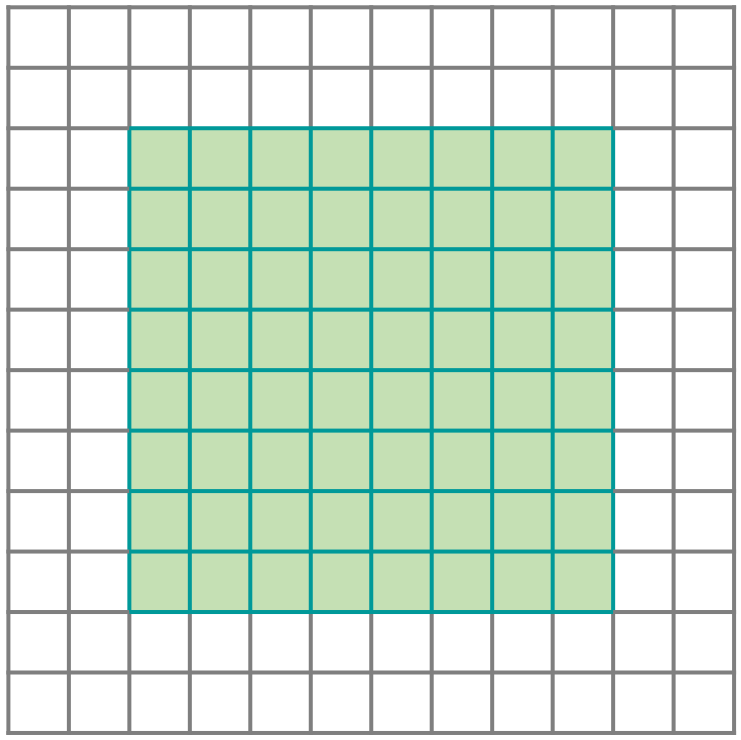
Output, $I'(x, y)$

Linear Filters

- For a given x and y, superimpose the filter on the image centered at (x, y)
- Compute the new pixel value, $I'(x, y)$, as the sum of $m \times m$ values, where each value is the product of the original pixel value in $I(x, y)$ and the corresponding values in the filter



Input, $I(x, y)$



Output, $I'(x, y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$

Filter, $F(x,y)$

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0								

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20							

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40						

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60					

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output — Filter — Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60				

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output — Filter — Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60	60			

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output — Filter — Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60	60	40		

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60	60	40	20	

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

Filter, $F(x,y)$

	0	20	40	60	60	60	40	20	
	0								

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output — Filter — Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60	60	40	20	
	0	40							

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output — Filter — Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60	60	40	20	
	0	40	80						

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$*$ $\frac{1}{9}$

1	1	1
1	1	1
1	1	1

Filter, $F(x,y)$

$=$

	0	20	40	60	60	60	40	20	
	0	40	80	120	120	120	80	40	
	0	60	120	180	180	180	120	60	
	0	60	100	160	160	180	120	60	
	0	60	100	160	160	180	120	60	
	0	40	60	100	100	120	80	40	
	20	40	60	60	60	60	40	20	
	20	20	20	0	0	0	0	0	

Output, $I'(x,y)$

Example of **Linear Filter**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

Output | Filter | Image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	180	0	180	180	180	0	0
0	0	0	180	180	180	180	180	0	0
0	0	0	0	0	0	0	0	0	0
0	0	180	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Image, $I(x,y)$

$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$

Filter, $F(x,y)$

0	20	40	60	60	60	40	20	
0	40	80	120	120	120	80	40	
0	60	120	180	180	180	120	60	
0	60	100	160	160	180	120	60	
0	60	100	160	160	180	120	60	
0	40	60	100	100	120	80	40	
20	40	60	60	60	60	40	20	
20	20	20	0	0	0	0	0	

Output, $I'(x,y)$

Computation of **Linear Filters**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

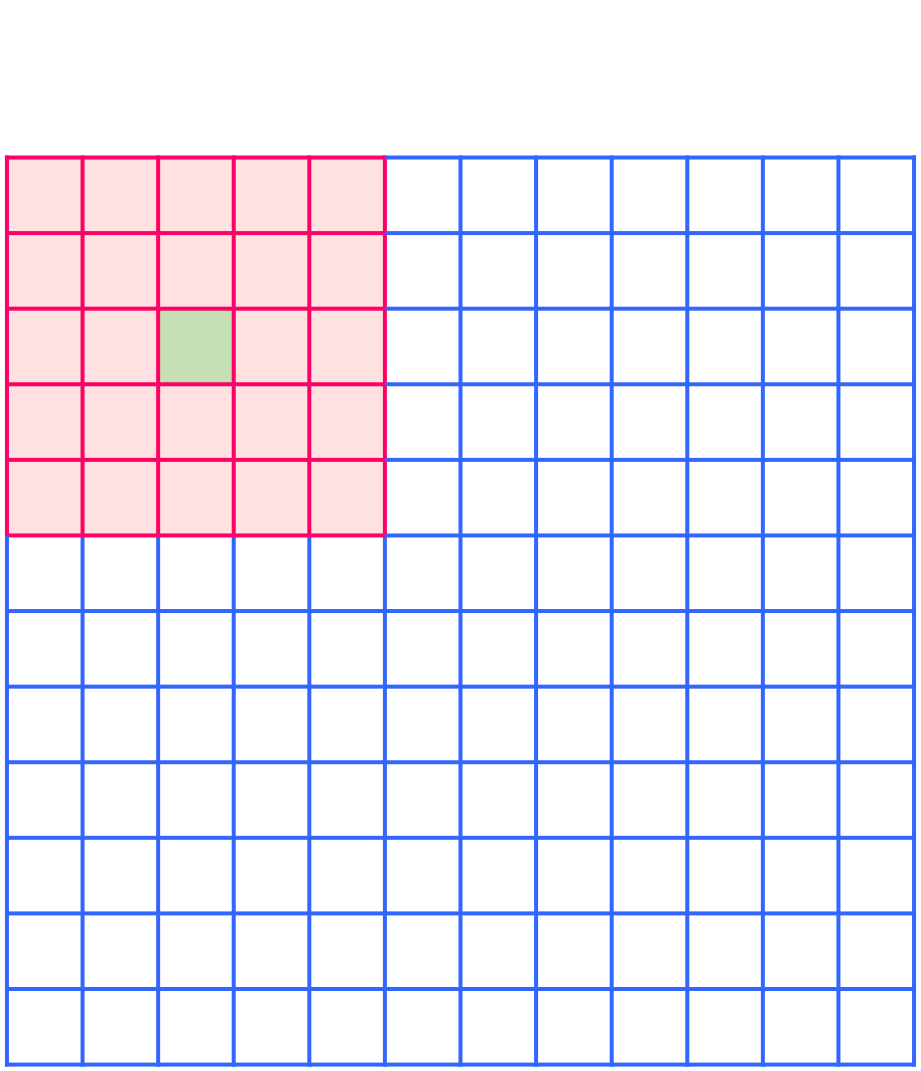
Output — Filter — Image

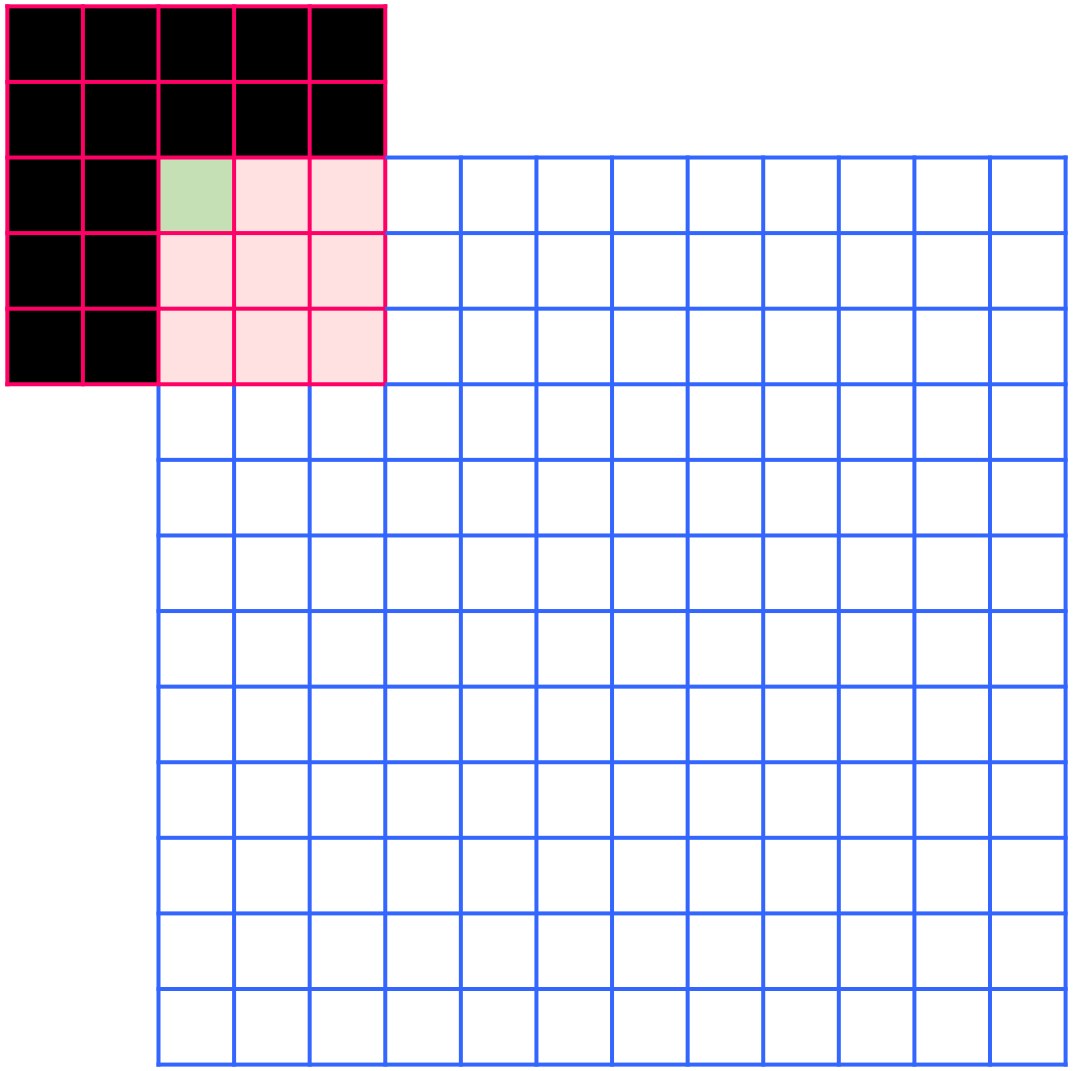
At each pixel, (x, y), there are $m \times m$ multiplications

There are $n \times n$ pixels in (x, y)

Total: $m^2 \times n^2$ multiplications

Boundary Effects





Boundary Effects

- Four standard ways to deal with filtering at boundaries:

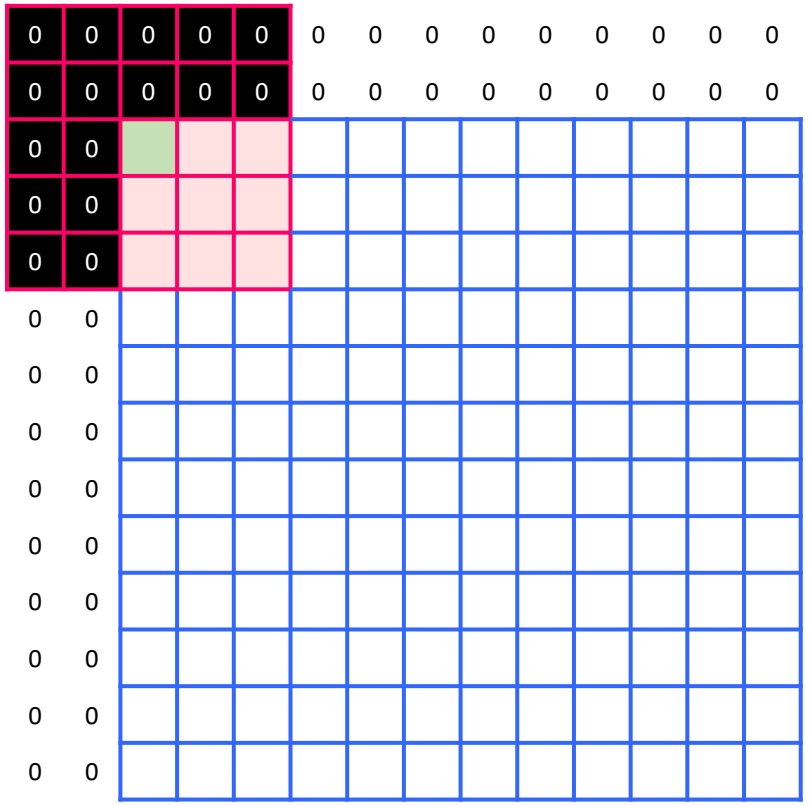
— **Ignore these locations:** Make the computation undefined for the top and bottom k rows and the leftmost and rightmost k columns

— **Pad the image with zeros (Zero padding):** Return zero whenever a value of image is required at some position outside the defined limits of x and y

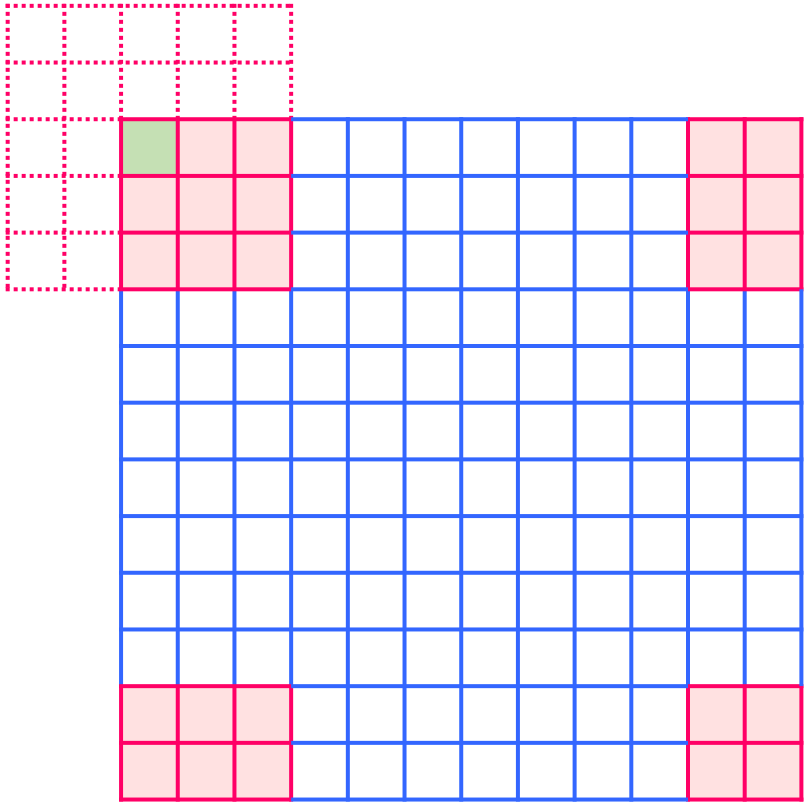
— **Assume periodicity (Periodic padding):** The top row wraps around to the bottom row; the leftmost column wraps around to the rightmost column

— **Reflect boarder (Mirror padding):** Copy rows/columns locally by reflecting over the edge

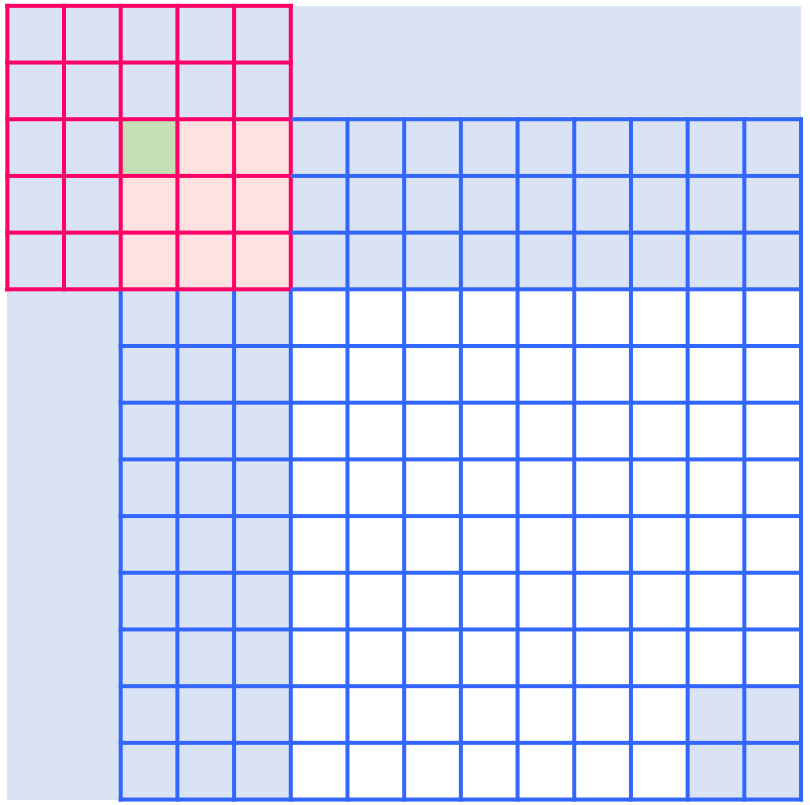
Boundary Effects



Zero padding

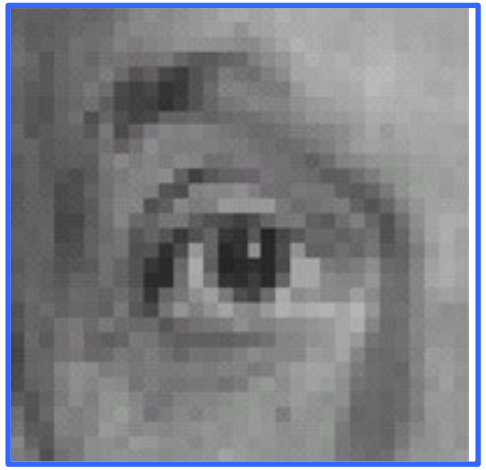


Periodic padding



Mirror padding

Examples of **Linear Filters**



Original image

$*$

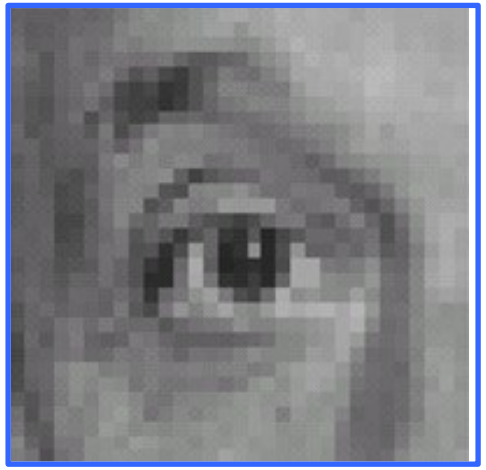
0	0	0
0	1	0
0	0	0

Filter

$=$

?

Examples of **Linear Filters**



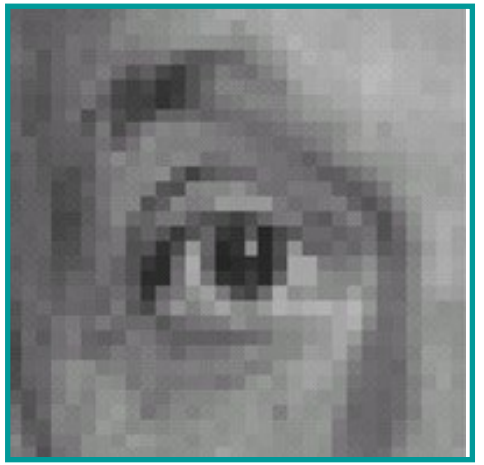
Original image

$*$

0	0	0
0	1	0
0	0	0

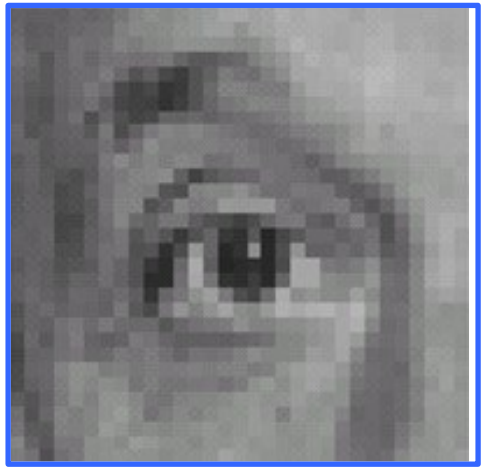
Filter

$=$



Result
(No change)

Examples of **Linear Filters**



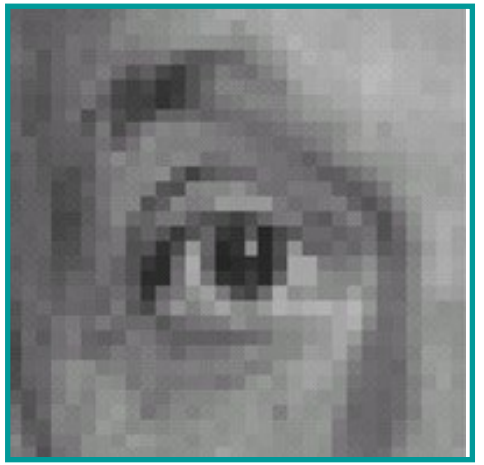
Original image

*

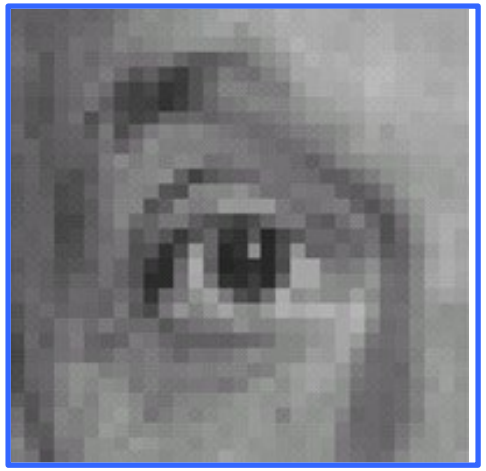
0	0	0
0	1	0
0	0	0

Filter

=



Result
(No change)



Original image

*

0	0	0
0	0	1
0	0	0

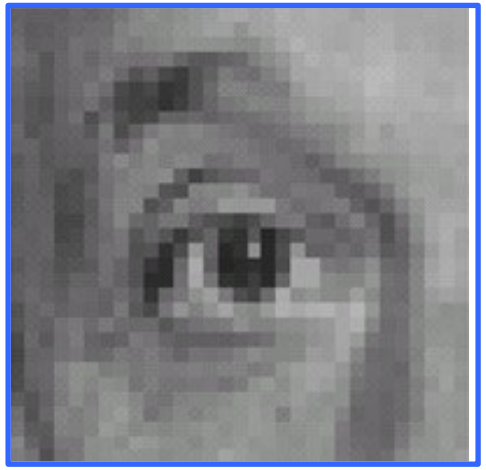
Filter

=

?

Result
(Shift left by 1 pixel)

Examples of **Linear Filters**



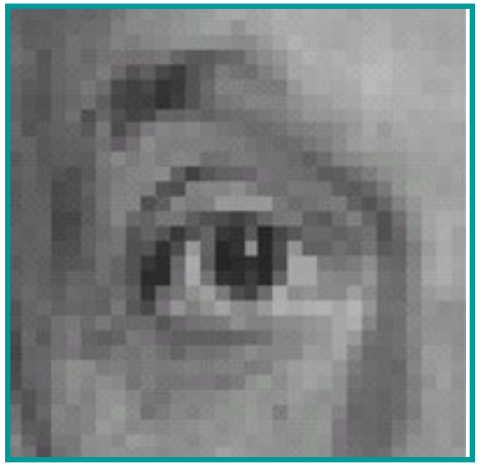
Original image

*

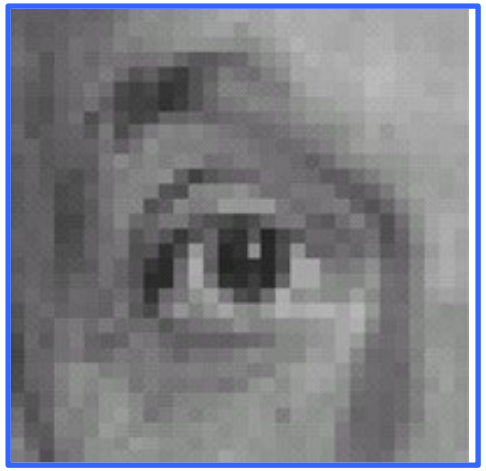
0	0	0
0	1	0
0	0	0

Filter

=



Result
(No change)



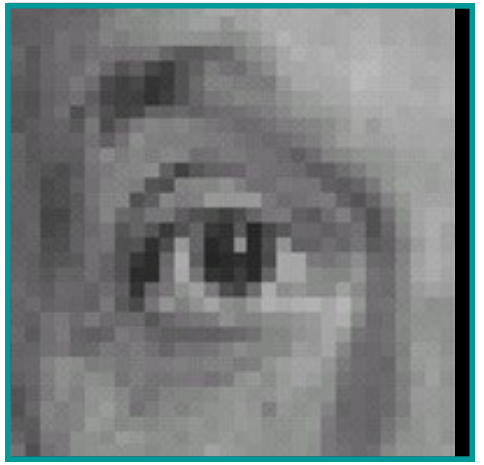
Original image

*

0	0	0
0	0	1
0	0	0

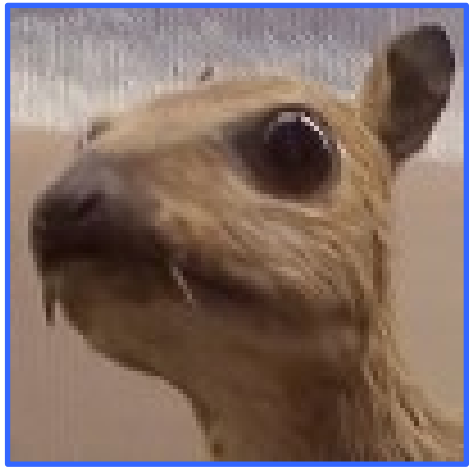
Filter

=



Result
(Shift left by 1 pixel)

Examples of **Linear Filters: Smoothing**

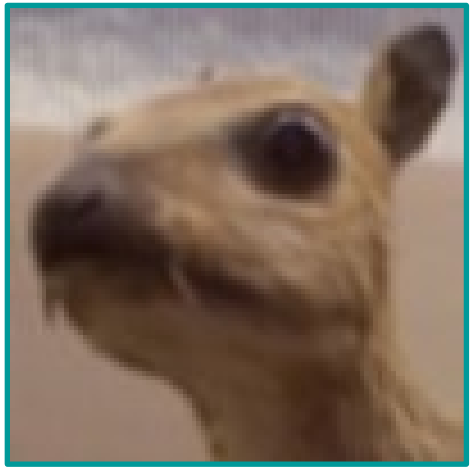


Original image

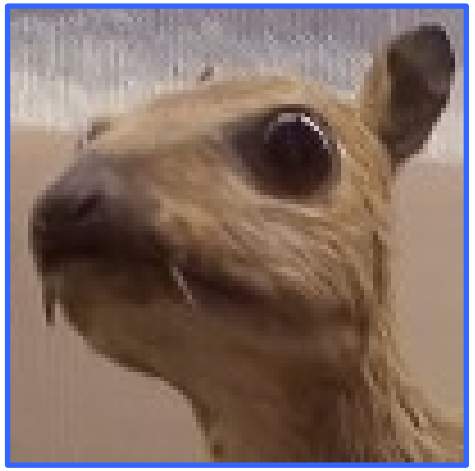
$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filter
(filter sums to 1)

$=$



Result
(Blur with a box filter)

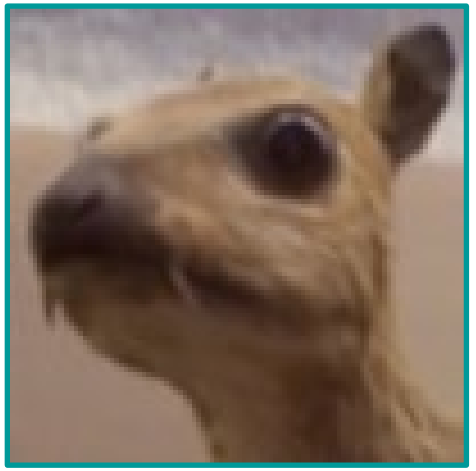


Original image

$$* \quad \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

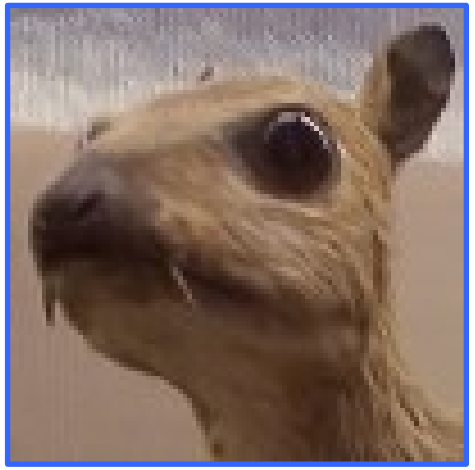
Filter
(filter sums to 1)

$=$



Result
(Blur with a box filter)

Examples of **Linear Filters: Sharpening**

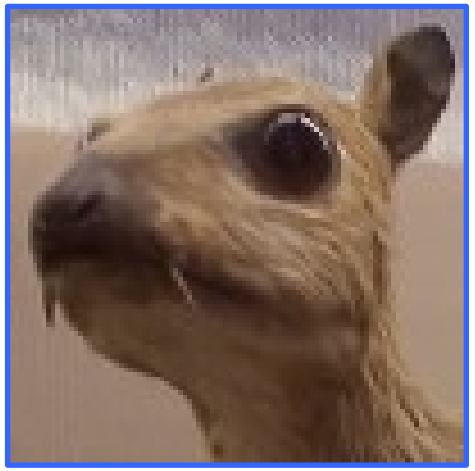


Original image

$$* \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right] = \boxed{?}$$

Filter
(filter sums to 1)

Examples of **Linear Filters: Sharpening**



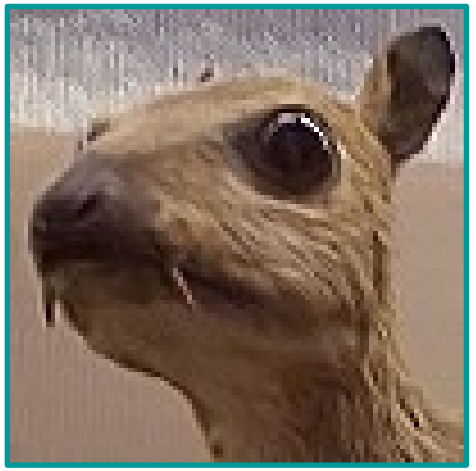
Original image

$$*\left[\begin{matrix}0&0&0\\0&2&0\\0&0&0\end{matrix}-\frac{1}{9}\begin{matrix}1&1&1\\1&1&1\\1&1&1\end{matrix}\right]=$$

Image itself (scaled)

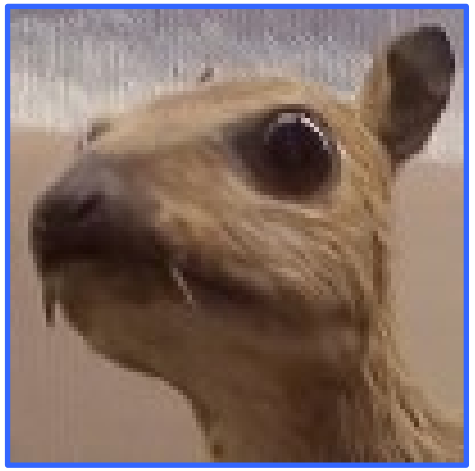
Blur kernel

Filter (filter sums to 1)



Result (Sharpening)

Examples of **Linear Filters: Sharpening**



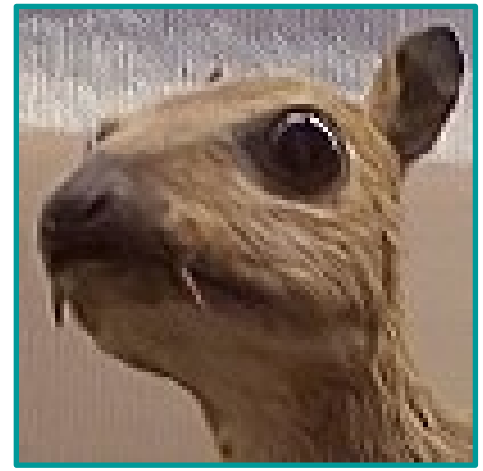
Original image

$$* \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right] =$$

Image itself (scaled)

Blur kernel

Filter (filter sums to 1)



Result (Sharpening)

Original image

$$* \left[\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right] =$$

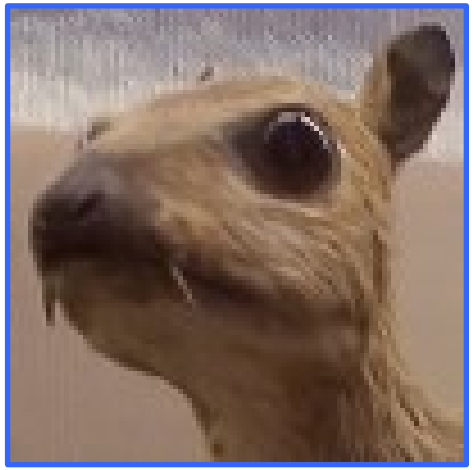
Image itself (scaled)

Blur kernel

Filter (filter sums to 1)

Result (Sharpening)

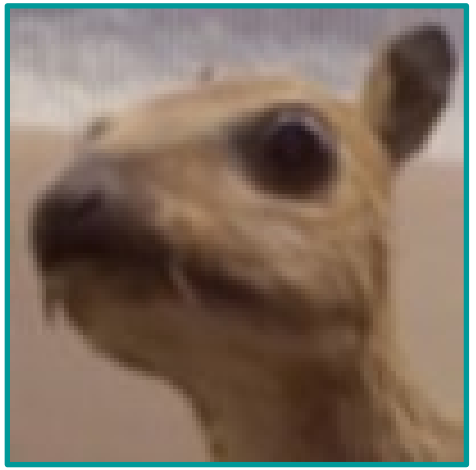
Smoothing **with a Box Filter**



Original image

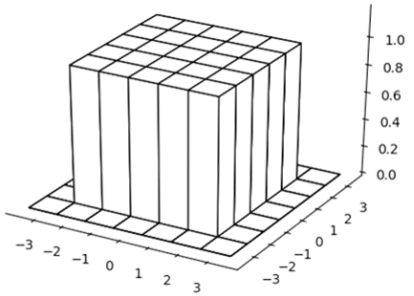
$$* \quad \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$

Filter
(filter sums to 1)



Result
(Blur with a box filter)

- Box filter has **equal positive values** that some up to 1



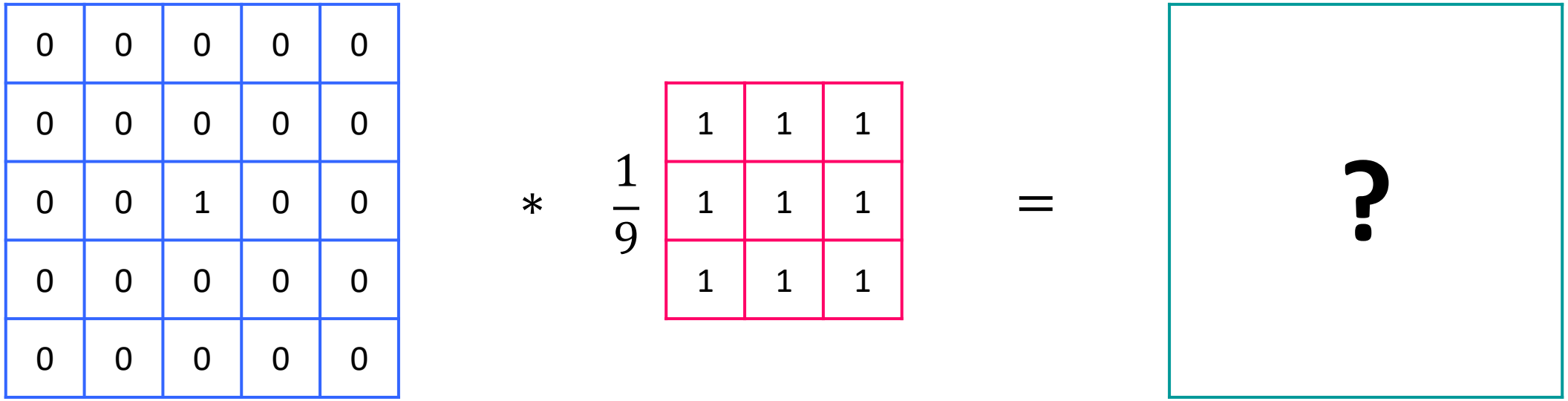
- Replaces each pixel with **the average of itself and its local neighborhood**
  - Box filter is also referred to as **average filter** or **mean filter**

Smoothing **with a Box Filter**

- According to the Size of the Box Filter

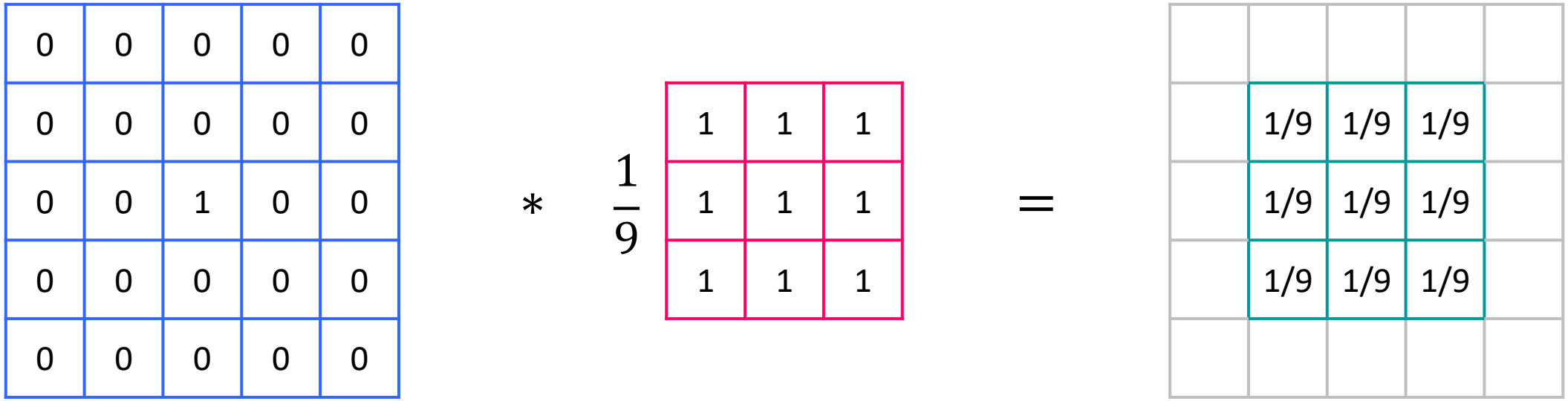
Limitations of **a Box Filter**

- Smoothing with **a box filter doesn't model lens defocus** well
  - Smoothing with a box filter depends on direction
  - Image in which the center point is 1 and every other point is 0



Limitations of **a Box Filter**

- Smoothing with **a box filter doesn't model lens defocus** well
  - Smoothing with a box filter depends on direction
  - Image in which the center point is 1 and every other point is 0

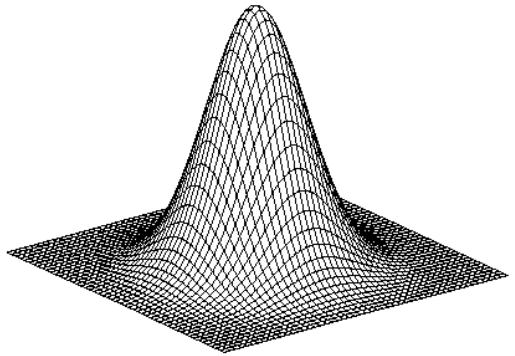


Solution for **Better Smoothing**

- Smoothing with **a box filter doesn't model lens defocus** well
  - Smoothing with a box filter depends on direction
  - Image in which the center point is 1 and every other point is 0

- Smoothing with **a circular pillbox is a better** model for defocus

- The **Gaussian** is a good general smoothing model
  - For phenomena that are the sum of other small effects
  - Whenever the central limit theorem (CLT) applies

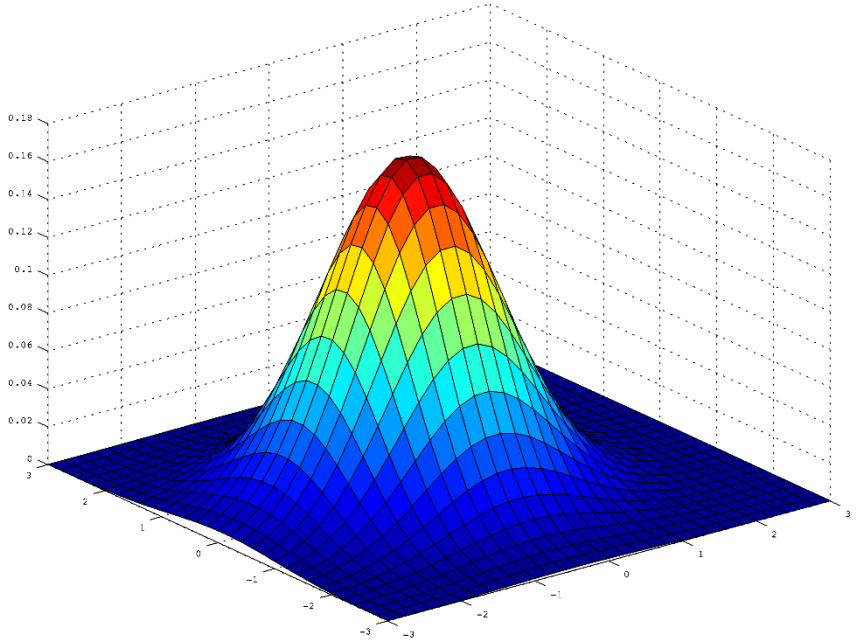


Smoothing **with a Gaussian**

- **Idea:** Weight contributions of pixels by spatial proximity (nearness)

- **2D Gaussian model (continuous):**

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



- **2D Gaussian model (discrete):**
  1. Define a continuous 2D Gaussian function
  2. Discretize it by evaluating this function on the discrete pixel positions to obtain a filter

Smoothing **with a Gaussian**

- Quantized and truncated **3×3 Gaussian filter**

$G_\sigma(-1, \quad 1)$	$G_\sigma(0, \quad 1)$	$G_\sigma(1, \quad 1)$
$G_\sigma(-1, \quad 0)$	$G_\sigma(0, \quad 0)$	$G_\sigma(1, \quad 0)$
$G_\sigma(-1, -1)$	$G_\sigma(0, -1)$	$G_\sigma(1, -1)$

Smoothing **with a Gaussian**

- Quantized and truncated **3×3 Gaussian filter**

$G_\sigma(-1, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$	$G_\sigma(0, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(1, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$
$G_\sigma(-1, \quad 0) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(0, \quad 0) = \frac{1}{2\pi\sigma^2}$	$G_\sigma(1, \quad 0) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$
$G_\sigma(-1, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$	$G_\sigma(0, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(1, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$

Smoothing **with a Gaussian**

- Quantized and truncated **3×3 Gaussian filter**

$G_\sigma(-1, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$	$G_\sigma(0, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(1, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$
$G_\sigma(-1, \quad 0) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(0, \quad 0) = \frac{1}{2\pi\sigma^2}$	$G_\sigma(1, \quad 0) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$
$G_\sigma(-1, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$	$G_\sigma(0, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(1, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$

— With $\sigma = 1$:

.059	.097	.059
.097	.159	.097
.059	.097	.059

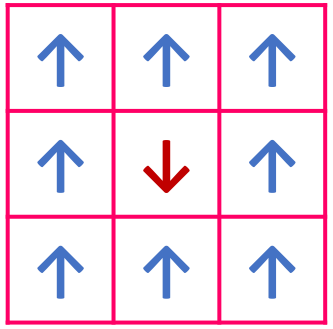
Smoothing **with a Gaussian**

- Quantized and truncated **3×3 Gaussian filter**

$G_\sigma(-1, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$	$G_\sigma(0, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(1, \quad 1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$
$G_\sigma(-1, \quad 0) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(0, \quad 0) = \frac{1}{2\pi\sigma^2}$	$G_\sigma(1, \quad 0) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$
$G_\sigma(-1, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$	$G_\sigma(0, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2\sigma^2}}$	$G_\sigma(1, -1) = \frac{1}{2\pi\sigma^2} e^{-\frac{2}{2\sigma^2}}$

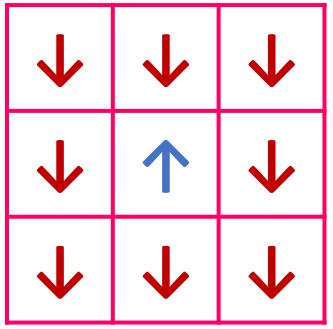
— If σ is larger:

- **More** blur



— If σ is smaller:

- **Less** blur



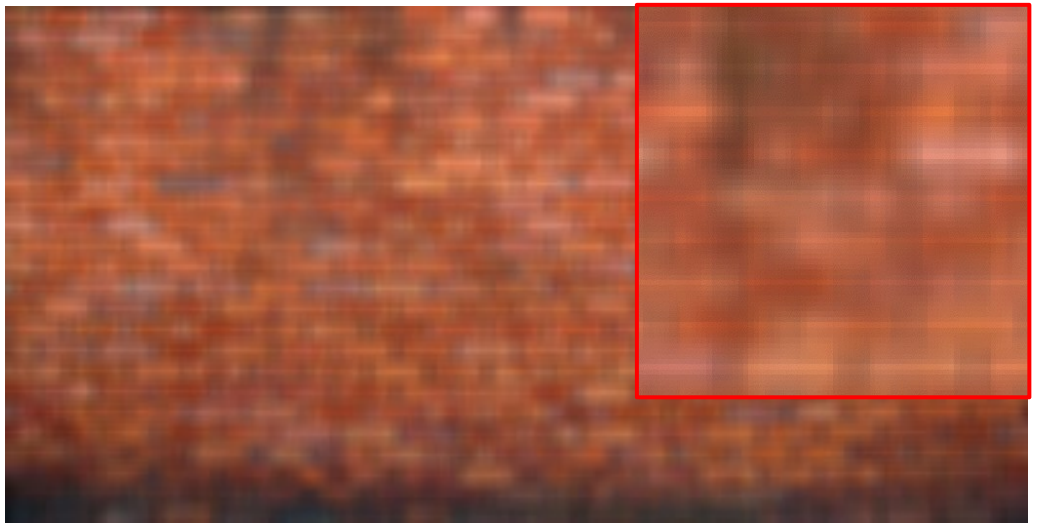
Smoothing **with a Gaussian**

- According to the Size of the Gaussian Filter

Smoothing **with a Gaussian**

- Box Filter vs. Gaussian Filter

7x7 Gaussian

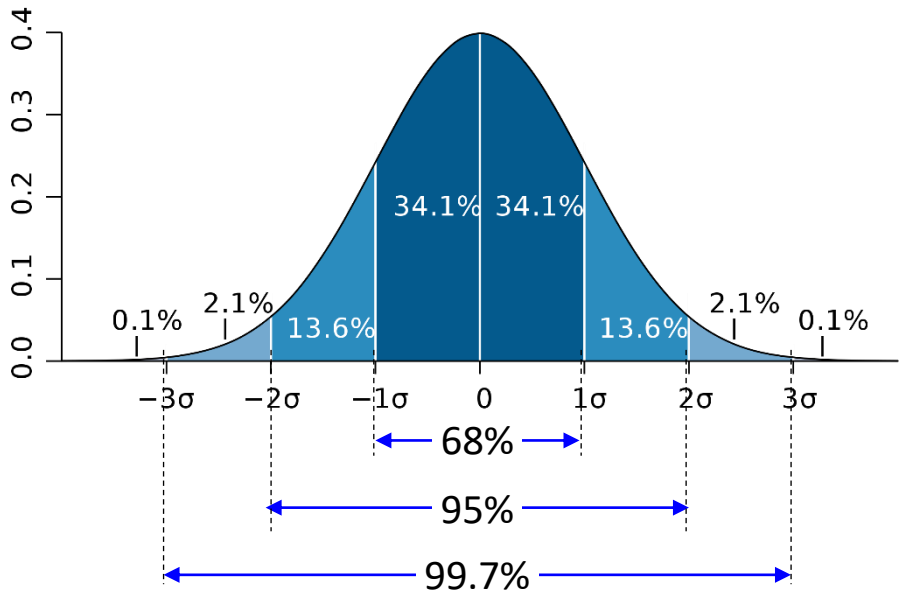


7x7 box

Smoothing **with a Gaussian**

- Problem of 3×3 Gaussian with $\sigma = 1$:
  - Does not sum to 1
  - Truncated too much

.059	.097	.059
.097	.159	.097
.059	.097	.059



- Better version of Gaussian filter:
  - Sums to 1 (normalized)
  - Captures $\pm 2\sigma$

$$\frac{1}{273}$$

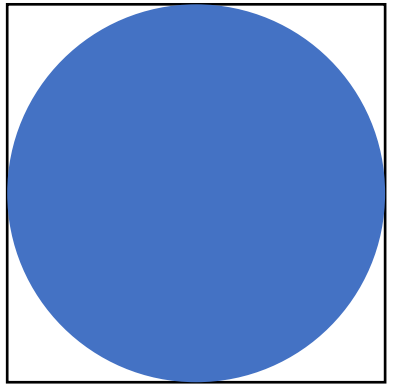
1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Smoothing **with a Pillbox**

- Let the radius (i.e., half diameter) of the filter be r

- In a contentious domain, a 2D (circular) pillbox filter, $f(x, y)$, is defined as:

$$f(x,y) = \frac{1}{\pi r^2}\begin{cases}1, & if\ x^2 + y^2 \leq r^2 \\ 0, & otherwise\end{cases}$$



- The scaling constant, $\frac{1}{\pi r^2}$, ensures that the area of the filter is one

Smoothing **with a Pillbox**

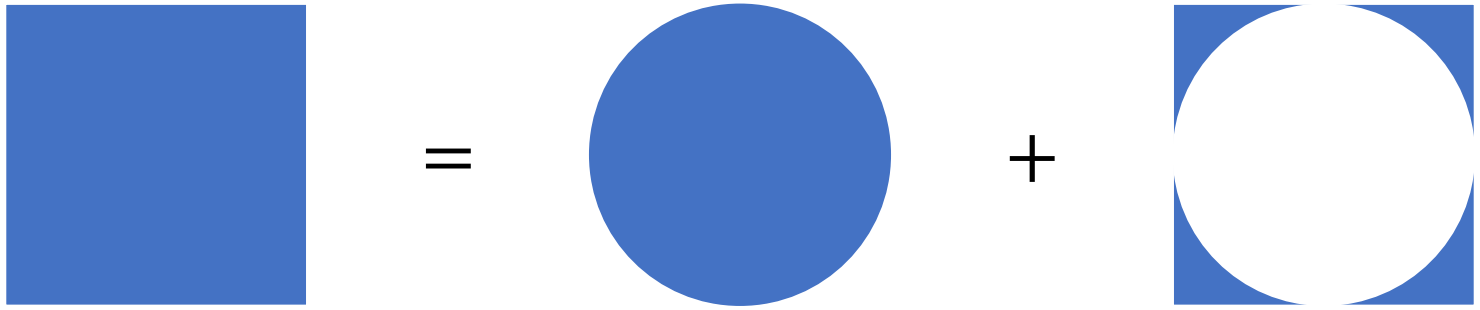
- The **2D Gaussian** is the only (non-trivial) 2D function that is both **separable** and **rotationally invariant**

- A **2D pillbox** is **rotationally invariant** but **not separable**

- There are occasions when we want to convolve an image with a 2D pillbox. Thus, it worth exploring possibilities for **efficient implementation**

Smoothing **with a Pillbox**

- A 2D box filter can be expressed as the sum of a 2D pillbox and some “extra corner bits”



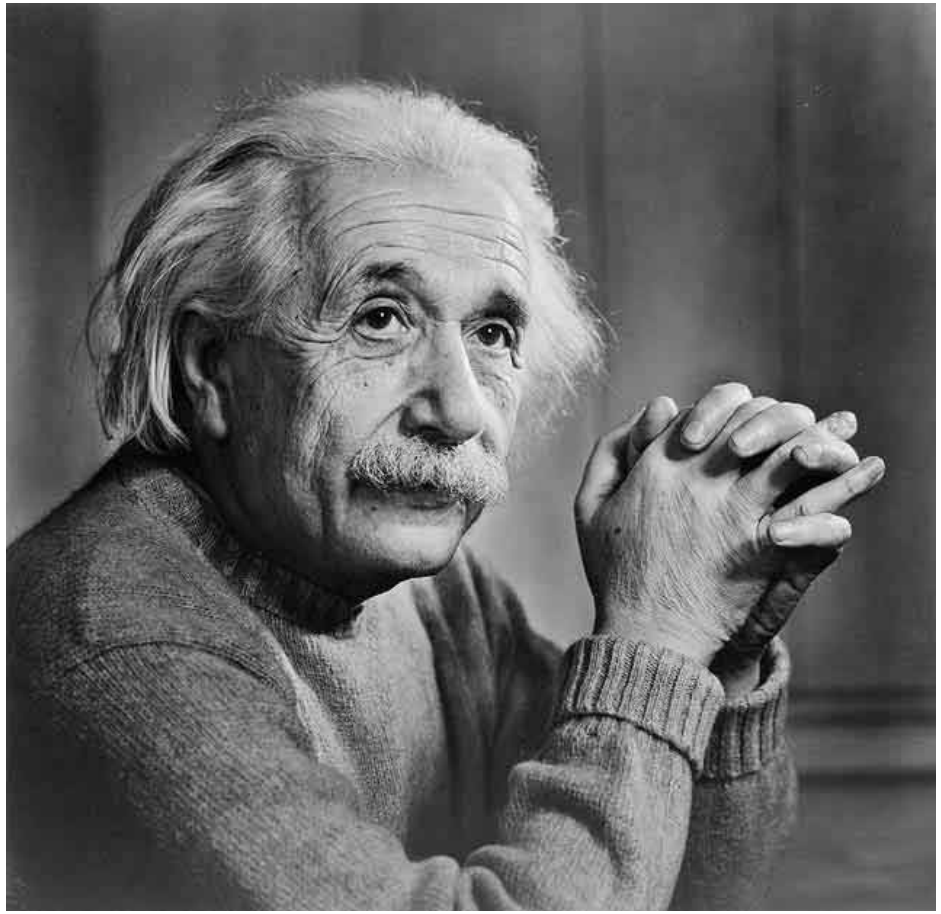
- A 2D pillbox filter can be expressed as the difference of a 2D box filter and those same “extra corner bits”

Smoothing **with a Pillbox**

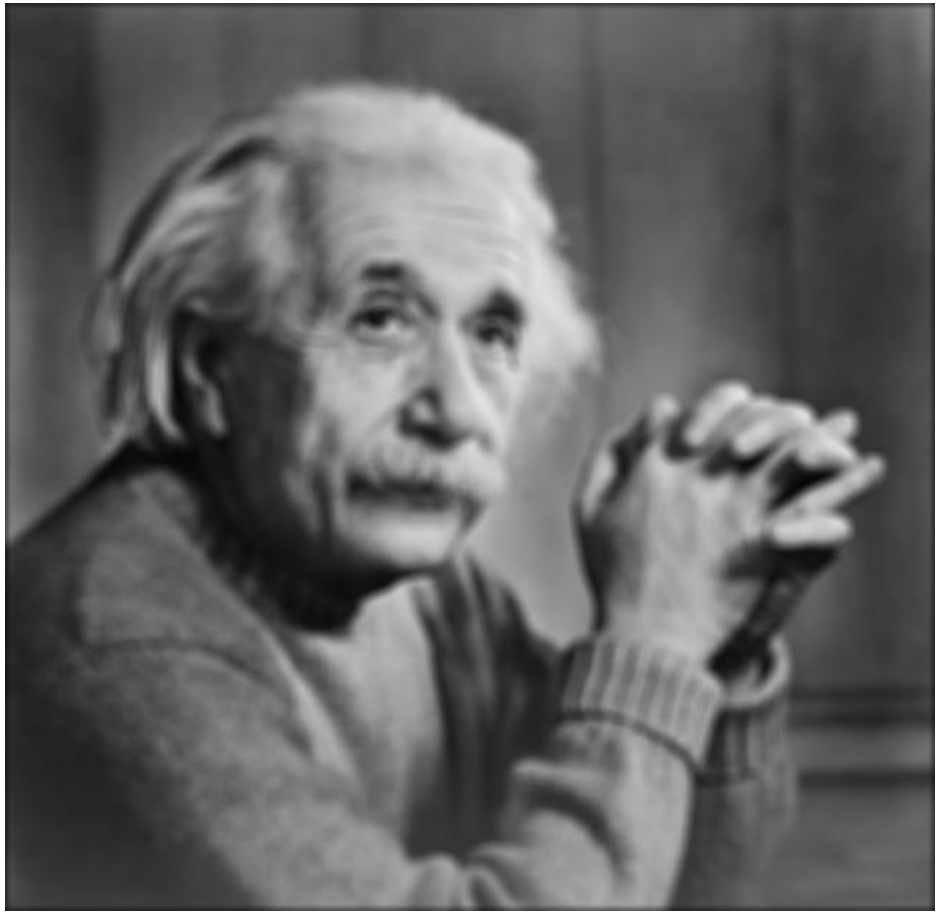
- Implementing convolution with a 2D pillbox filter as the difference between convolution with a box filter and convolution with the “extra corner bits” filter allows us to take advantage of the separability of a box filter

- Further, we can postpone scaling the output to a single, final step so that convolution involves filters containing all 0’s and 1’s
  - This means the required convolutions can be implemented without any multiplication at all

Smoothing **with a Pillbox**



Original image

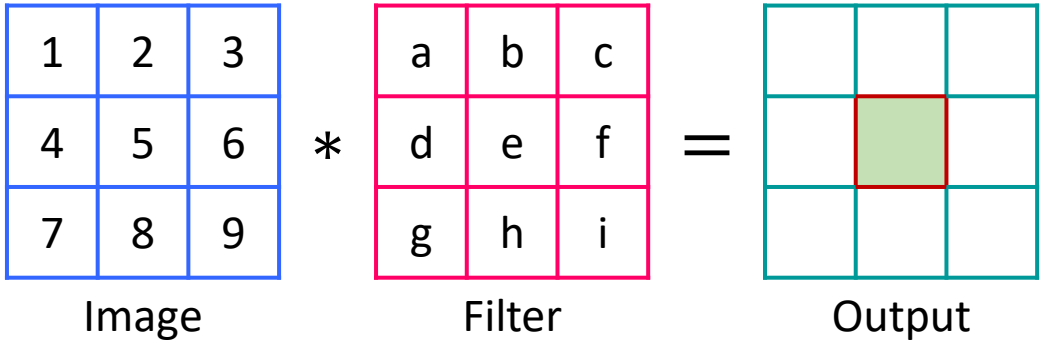


Result
by 11×11 pillbox filter

Linear Filters: **Correlation vs. Convolution**

- Definition: **Correlation**

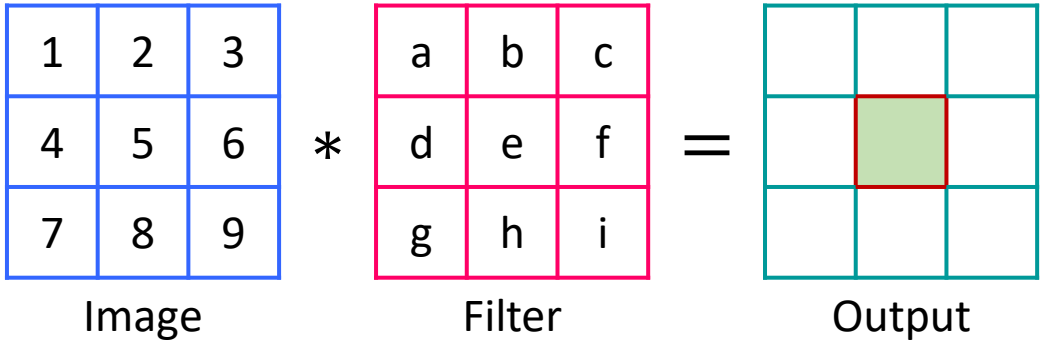
$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$



Linear Filters: **Correlation vs. Convolution**

- Definition: **Correlation**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$



= 1a+2b+3c
+4d+5e+6f
+7g+8h+9i

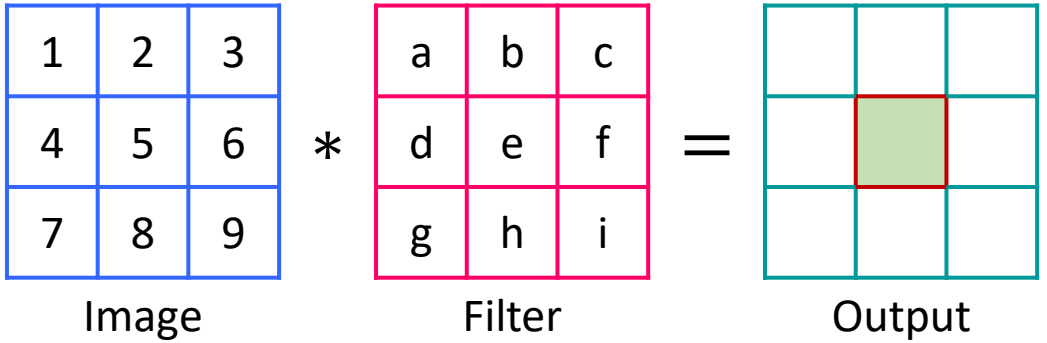
Linear Filters: **Correlation vs. Convolution**

- Definition: **Correlation**

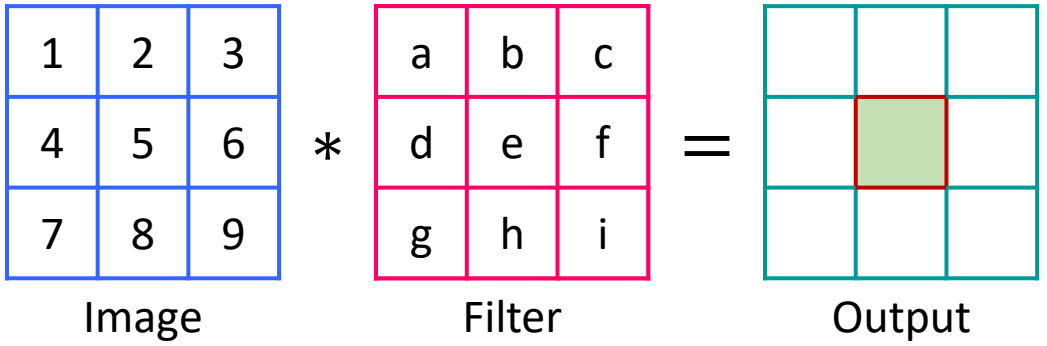
$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

- Definition: **Convolution**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x-i, y-j)$$



= 1a+2b+3c
+4d+5e+6f
+7g+8h+9i



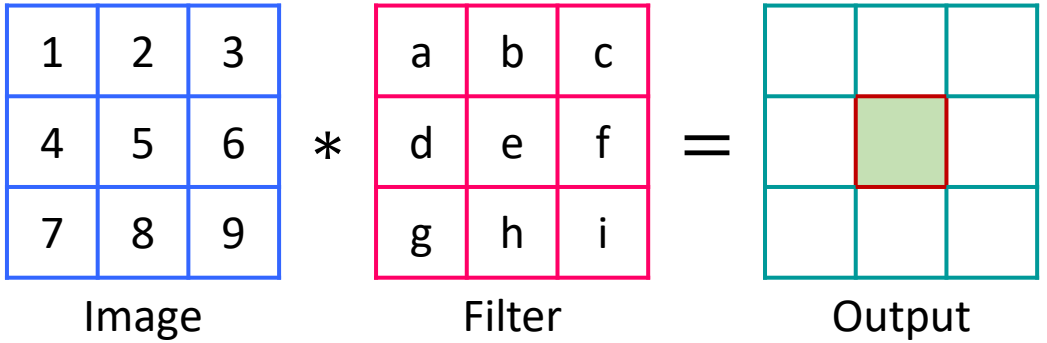
Linear Filters: **Correlation vs. Convolution**

- Definition: **Correlation**

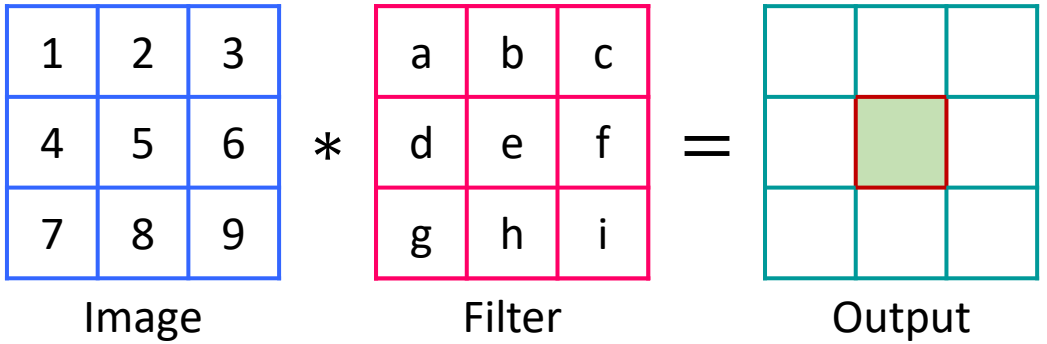
$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

- Definition: **Convolution**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x-i, y-j)$$



= 1a+2b+3c
+4d+5e+6f
+7g+8h+9i

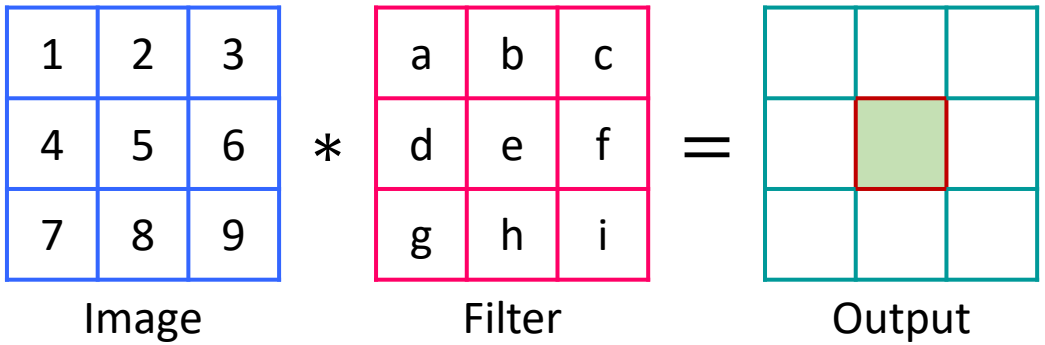


= 1i+2h+3g
+4f+5e+6d
+7c+8b+9a

Linear Filters: **Correlation vs. Convolution**

- Definition: **Correlation**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

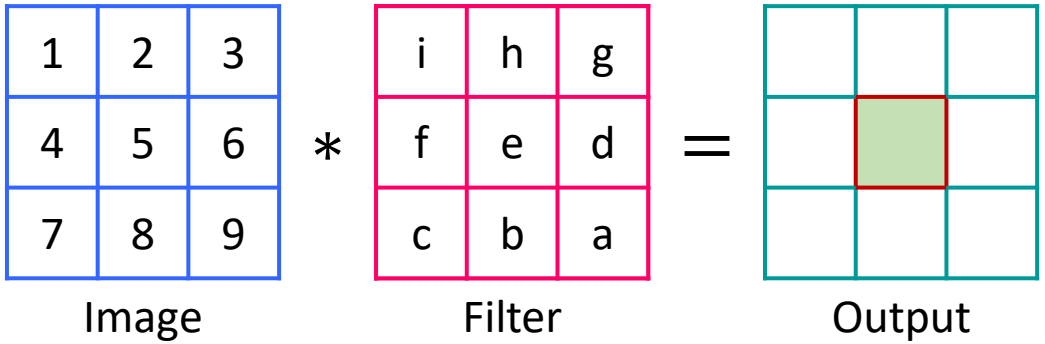


= 1a+2b+3c
+4d+5e+6f
+7g+8h+9i

- Definition: **Convolution**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x-i, y-j)$$

$$= \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(-i,-j)\, I(x+i, y+j)$$

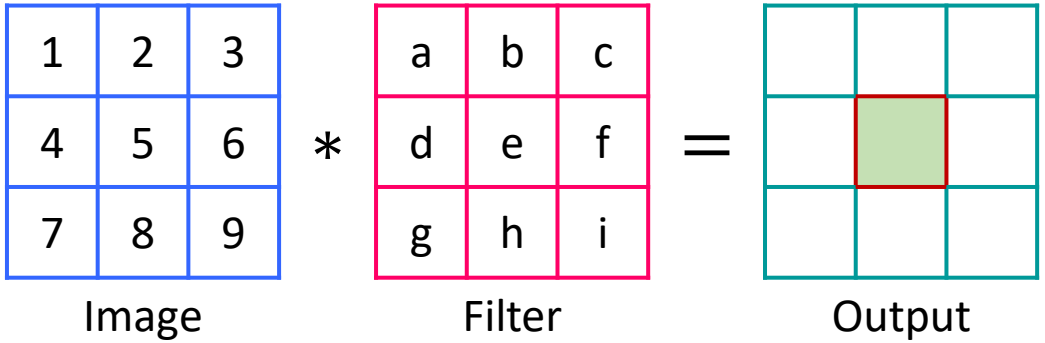


= 1i+2h+3g
+4f+5e+6d
+7c+8b+9a

Linear Filters: **Correlation vs. Convolution**

- Definition: **Correlation**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x+i, y+j)$$

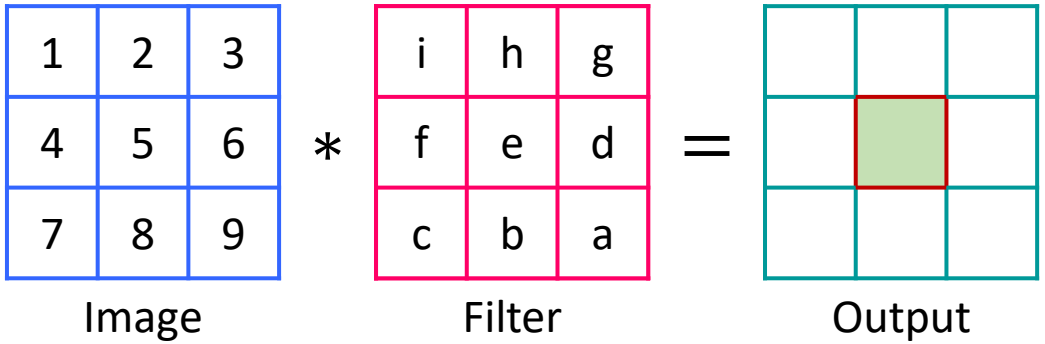


= 1a+2b+3c
+4d+5e+6f
+7g+8h+9i

- Definition: **Convolution**

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i,j)\, I(x-i, y-j)$$

$$= \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(-i,-j)\, I(x+i, y+j)$$

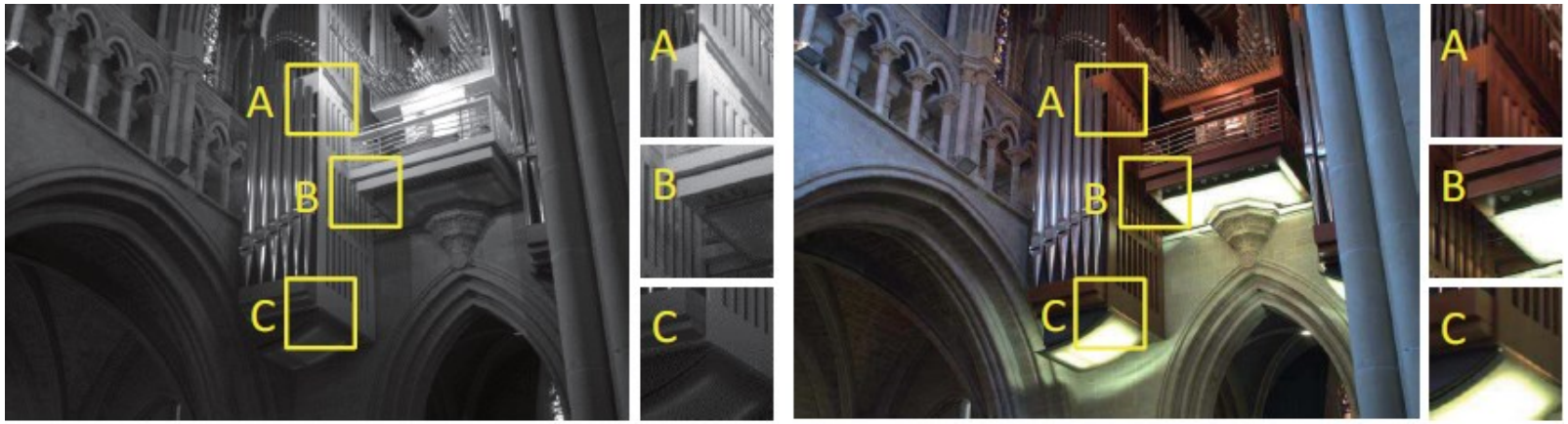


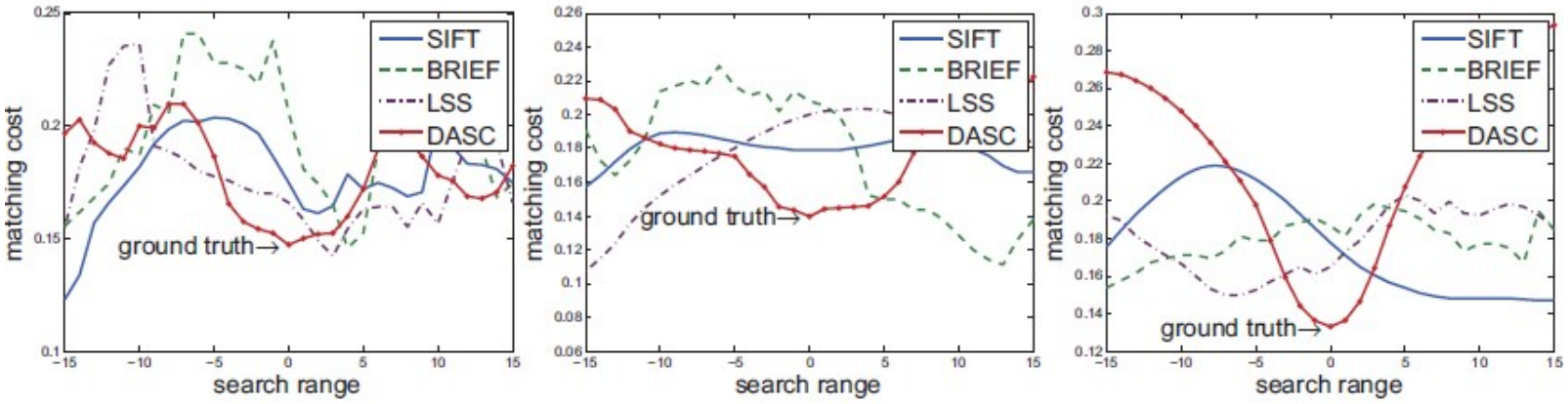
= 1i+2h+3g
+4f+5e+6d
+7c+8b+9a

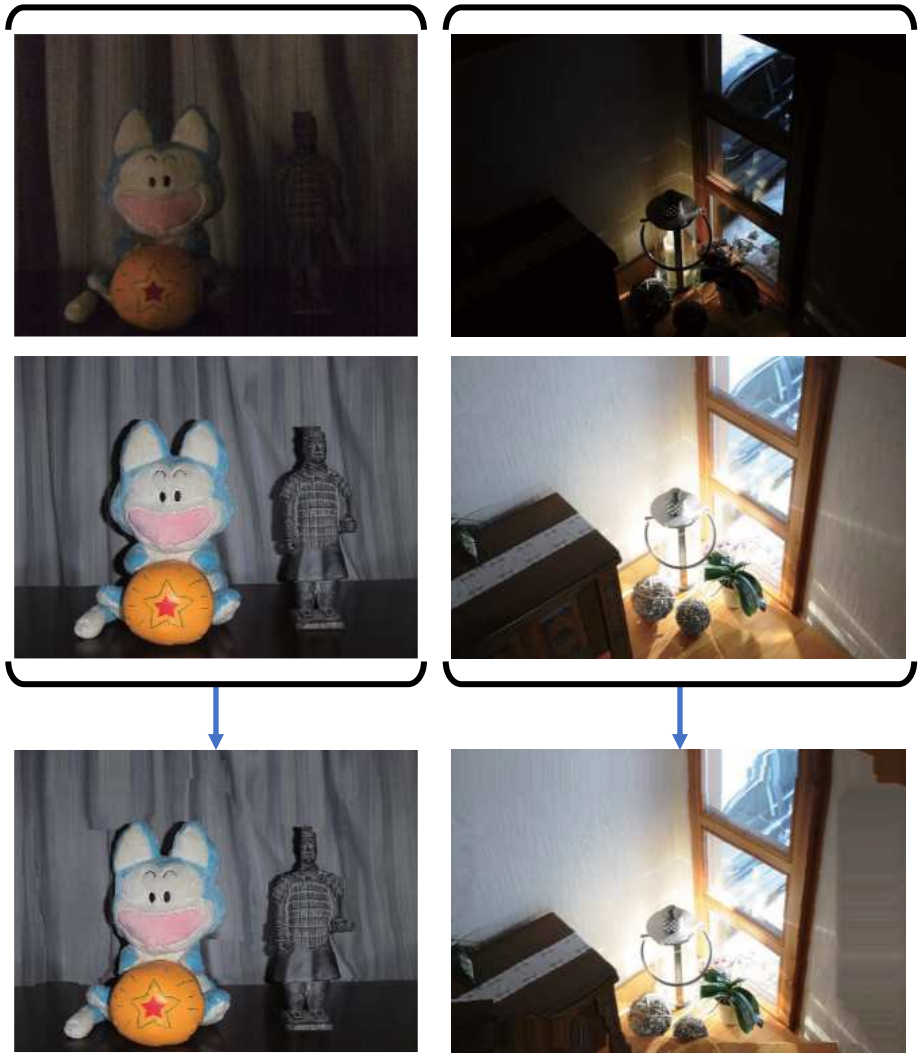
— If $F(x,y) = F(-x,-y)$, then "correlation = convolution"

Application of **Correlation**

- For a Measurement of the Similarity of Two Signals



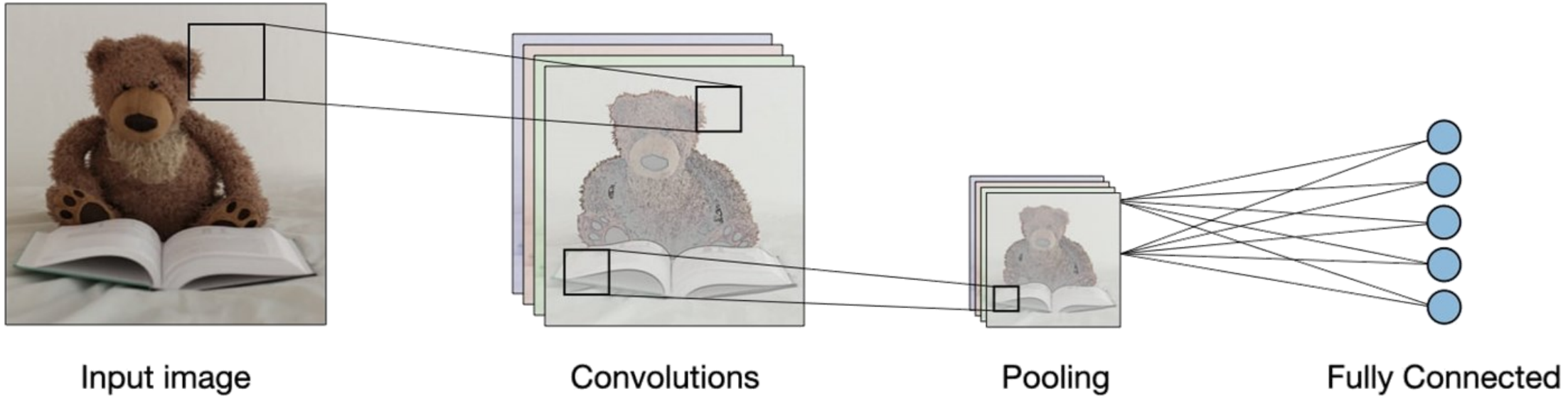


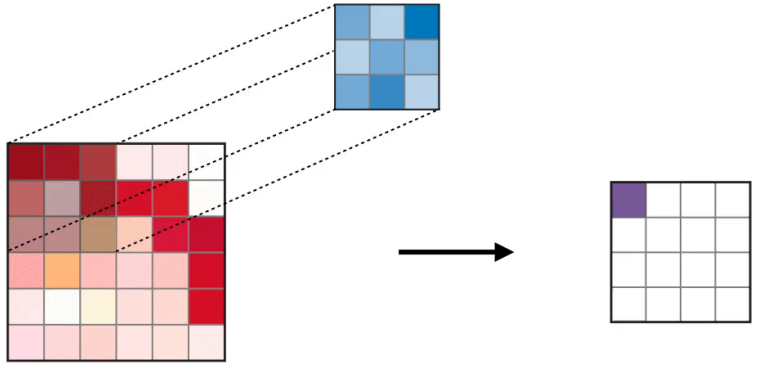


DASC: Dense Adaptive Self-Correlation Descriptor for Multi-modal and Multi-Spectral Correspondence, **CVPR 2015**

Application of **Convolution**

- In a Convolutional Neural Networks (CNNs)





Convolution

Properties of Linear Filters

- Let $\otimes$ denote convolution operator. Let $I(x, y)$ be a digital image

— **Superposition:** Let F_1 and F_2 be digital filters

$$(F_1 + F_2) \otimes I(x, y) = F_1 \otimes I(x, y) + F_2 \otimes I(x, y)$$

— **Scaling:** Let F be digital filter and let k be a scalar

$$(kF) \otimes I(x, y) = F \otimes \big(kI(x, y)\big) = k\big(F \otimes I(x, y)\big)$$

— **Shift Invariance:** Output is local (i.e., no dependence on absolute position)

- If an operation satisfies both **superposition** and **scaling**, it is **linear**

Additional Properties of Linear Filters

- Let $\otimes$ denote convolution operator. Let $I(x, y)$ be a digital image
- Let F and G be digital filters

— **Associative:**

$$G \otimes \left(F \otimes I(x, y)\right) = (G \otimes F) \otimes I(x, y)$$

— **Symmetric:**

$$(G \otimes F) \otimes I(x, y) = (F \otimes G) \otimes I(x, y)$$

- In general, **correlation** is **not associative**

Summary: **Correlation & Convolution**

- **Correlation** of $F(x, y)$ and $I(x, y)$:

$$I'(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(i, j)\, I(x + i, y + j)$$

- **Convolution** ($=$ **correlation** without flipped):

$$I'(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} F(-i, -j)\, I(x + i, y + j)$$

- **Characterization Theorem:** Any linear, spatially invariant operation can be expressed as a convolution

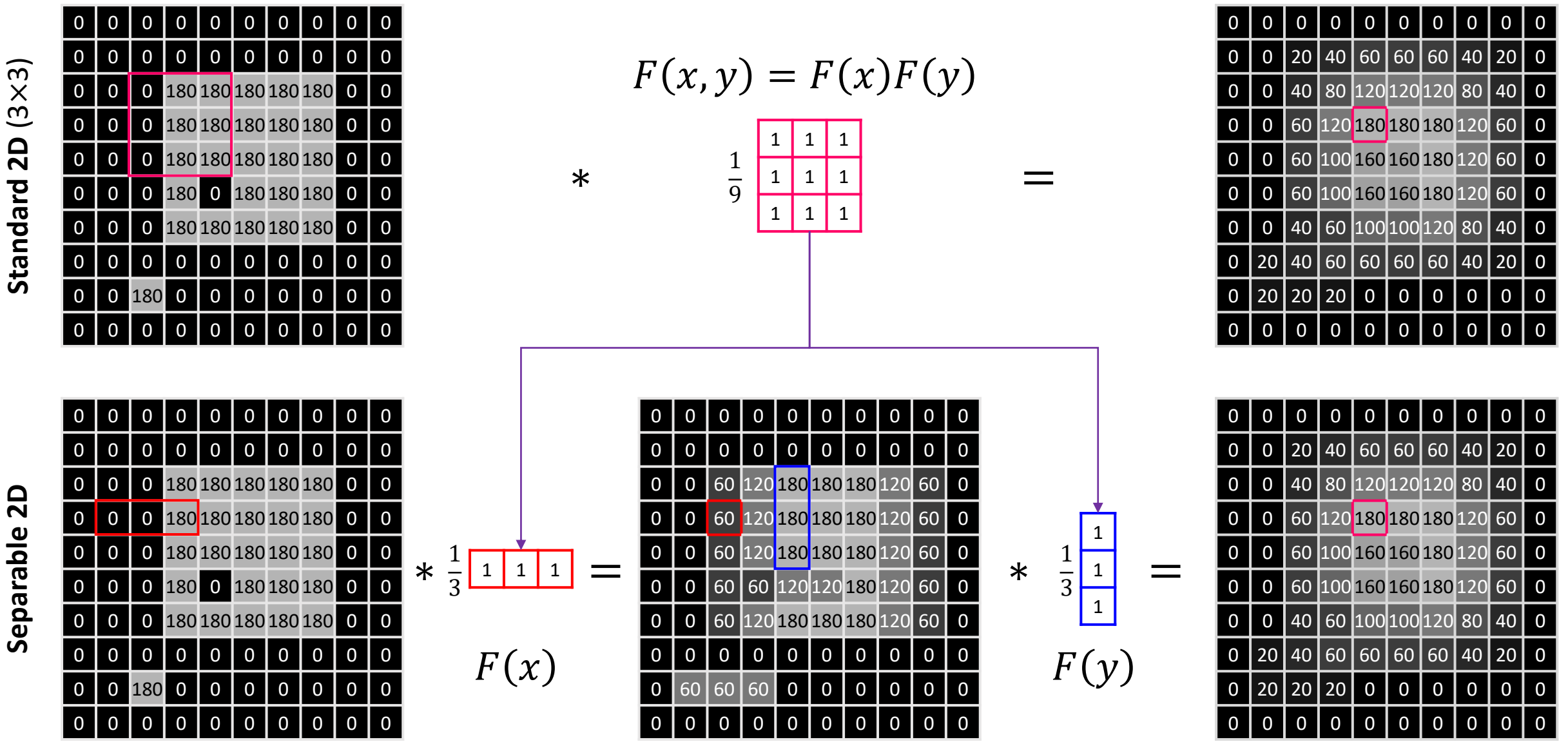
Efficiency Implementation: **Separability**

- A 2D function of x and y is **separable** if it can be written as the product of two functions, one a function only of x and the other a function only of y

- Surely, both the **2D box filter** and the **2D Gaussian filter** are **separable**
  - ① Convolve each row with a 1D filter
  - ② Convolve each column with a 1D filter
  - — Aside: or vice versa

- The **2D Gaussian** is the only (non trivial) 2D function that is both **separable** and **rotationally invariant**

Efficiency Implementation: **Separability**



Efficiency Implementation: **Separability**

- Recall – **2D Gaussian**:

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

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

Function of x **Function of y**

— The 2D Gaussian can be expressed as a product of two functions, one a function of x and another a function of y

— In this case, the two function are identical 1D Gaussians

Efficiency Implementation: **Separability**

- **Standard** 2D Gaussian:

At each pixel, (x, y), there are $m \times m$ multiplications

There are $n \times n$ pixels in (x, y)

Total: $m^2 \times n^2$ multiplications

Efficiency Implementation: **Separability**

- **Standard** 2D Gaussian:

At each pixel, (x, y), there are	$m \times m$ multiplications
There are	$n \times n$ pixels in (x, y)
Total:	$m^2 \times n^2$ multiplications

- **Separable** 2D Gaussian

At each pixel, (x, y), there are	$2 \times m$ multiplications
There are	$n \times n$ pixels in (x, y)
Total:	$2m \times n^2$ multiplications

Speeding Up **Rotation**

- **2D rotation of a point by an angle** α about the origin

- The standard approach, in Euclidean coordinates, involves a matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Suppose we transform to polar coordinates

$$(x, y) \longrightarrow (\rho, \theta) \longrightarrow (\rho, \theta + \alpha) \longrightarrow (x', y')$$

- Rotation becomes addition, at expense of one polar coordinate transform and one inverse polar coordinate transform

Speeding Up **Convolution**: The Convolution Theorem

- Let z be the product of two numbers x and y, that is,

$$z = xy$$

- Taking logarithms of both sides, one obtains

$$\ln z = \ln x + \ln y$$

- Therefore,

$$z = e^{\ln z} = e^{(\ln x + \ln y)}$$

- **Interpretation:** At the expense of two $\ln(\quad)$ and one $exp(\quad)$ computations, multiplication is reduced to admission

Speeding Up **Convolution**: The Convolution Theorem

- **Convolution Theorem:**

— Let $i'(x,y) = f(x,y) \otimes i(x,y)$

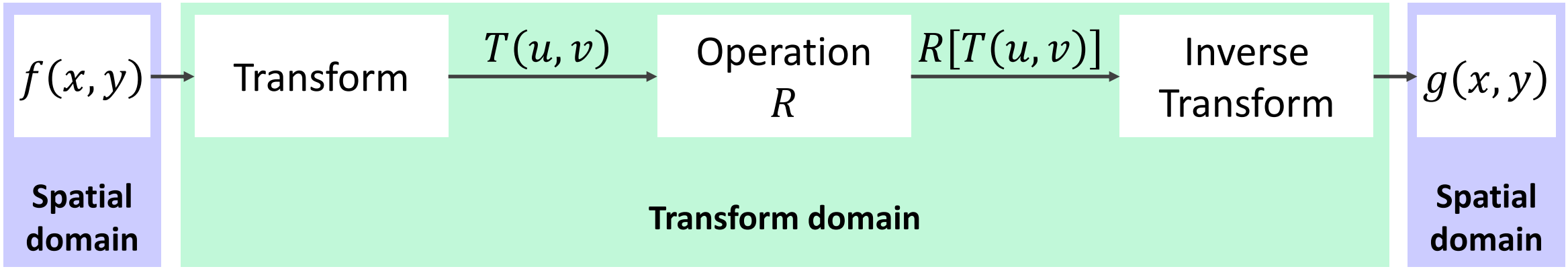
— Then $\mathcal{I}'(\omega_x, \omega_y) = \mathcal{F}(\omega_x, \omega_y)\mathcal{I}(\omega_x, \omega_y)$

  - where $\mathcal{I}'(\omega_x, \omega_y), \mathcal{F}(\omega_x, \omega_y)$ and $\mathcal{I}(\omega_x, \omega_y)$ are Fourier transforms of $i'(x,y), f(x,y)$ and $i(x,y)$

- At the expense of **two Fourier** transforms and **one inverse Fourier** transform, convolution can be reduced to complex multiplication

Speeding Up **Convolution**: The Convolution Theorem

- Some image processing operations become cheaper in transform domain



Speeding Up **Convolution**: The Convolution Theorem

- Convolution **in Spatial Domain:**

  At each pixel, (x, y), there are $m \times m$ multiplications

  There are $n \times n$ pixels in (x, y)

  Total: $m^2 \times n^2$ multiplications

- Convolution **in Frequency Domain:**

  Cost of FFT & IFFT for image: $n^2 \log n$ multiplications

  Cost of FFT & IFFT for filter: $m^2 \log m$ multiplications

  Cost of convolution: n^2 multiplications

  Total: $n^2 \log n + m^2 \log m + n^2$ multiplications

Topics

- Image Transformation
- Image Filtering: Linear Filters
- Image Filtering: Non-linear Filters

Non-linear Filters

- **Linear filters** can perform a variety of image transformations
  - Shifting
  - Smoothing
  - Sharpening

- **Non-linear filters** can provide better performances in some applications
  - Median filter
  - Bilateral filter
  - ReLU

Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter

5	13	5	221	7
4	16	7	34	98
24	54	34	23	21
23	75	90	123	89
54	25	67	12	24

Image

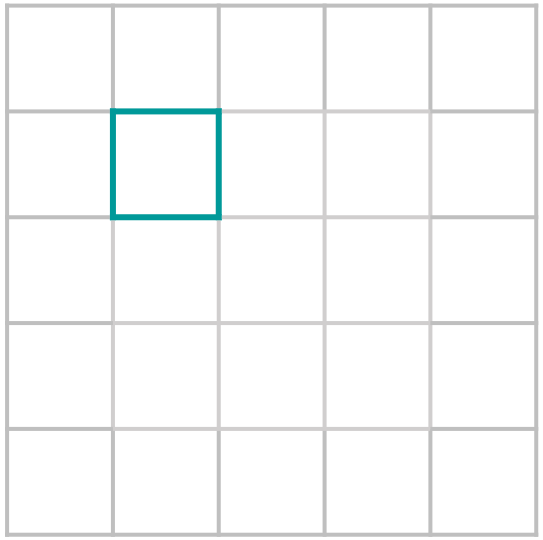
Output

Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter

5	13	5	221	7
4	16	7	34	98
24	54	34	23	21
23	75	90	123	89
54	25	67	12	24

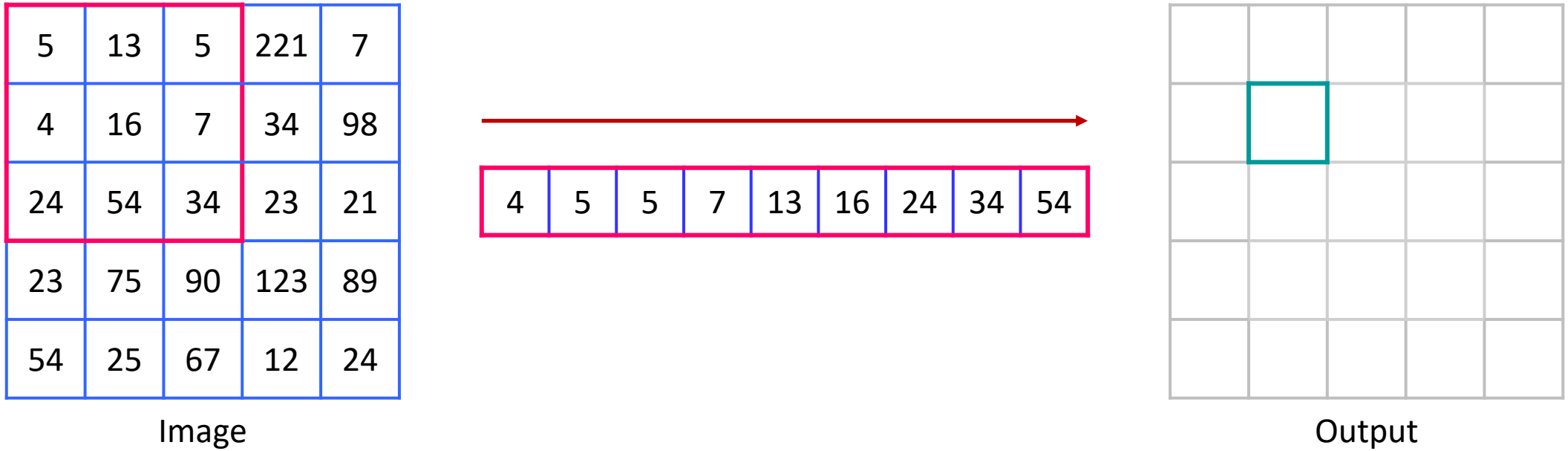
Image



Output

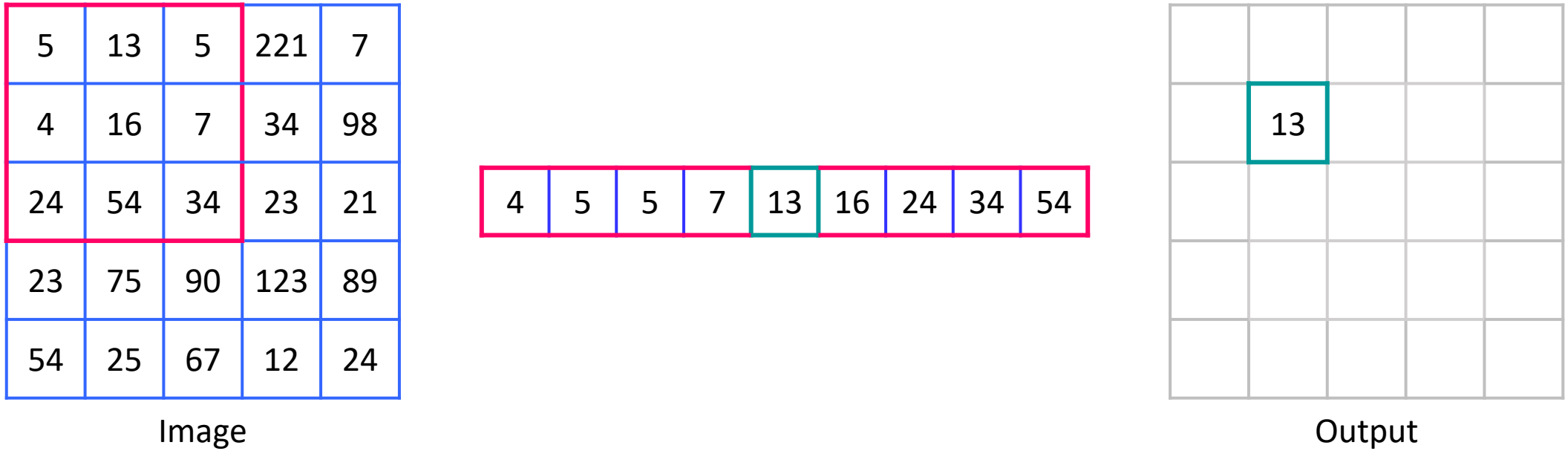
Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter



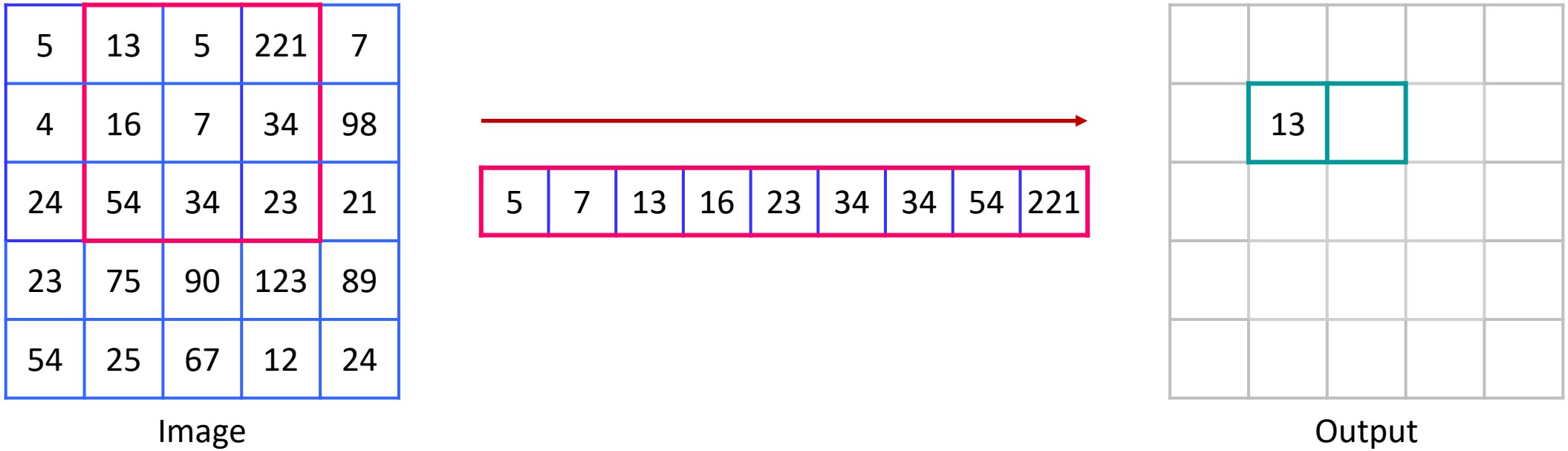
Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter
  ② Pick the median value as an output pixel



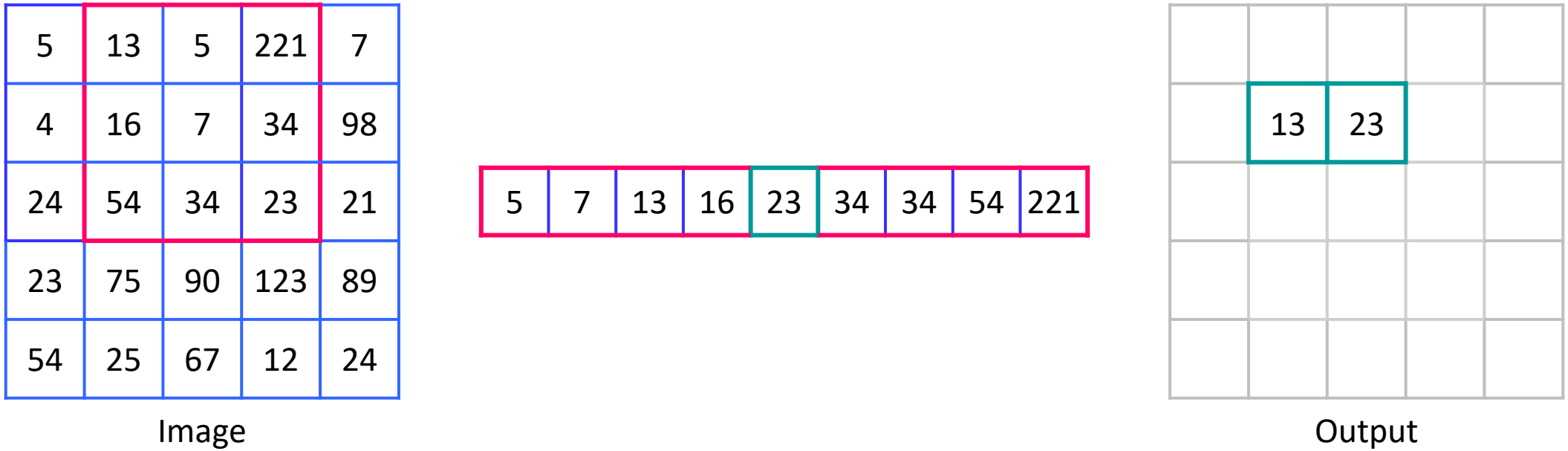
Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter



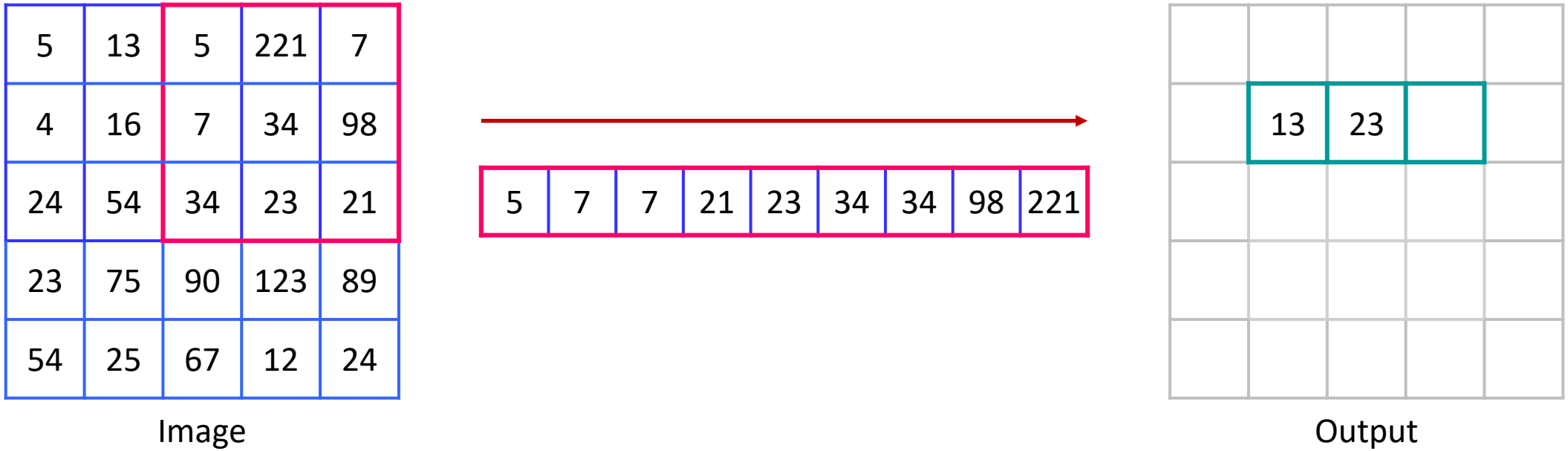
Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter
  ② Pick the median value as an output pixel



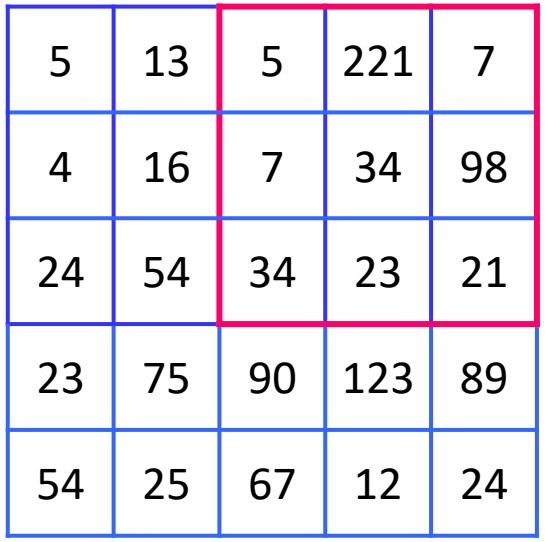
Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter

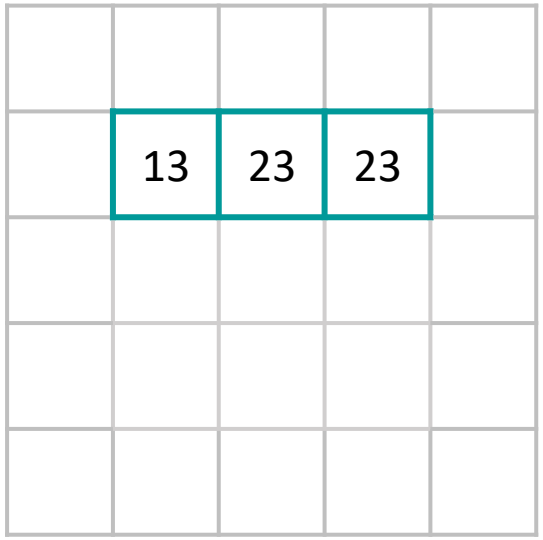


Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter
  ② Pick the median value as an output pixel



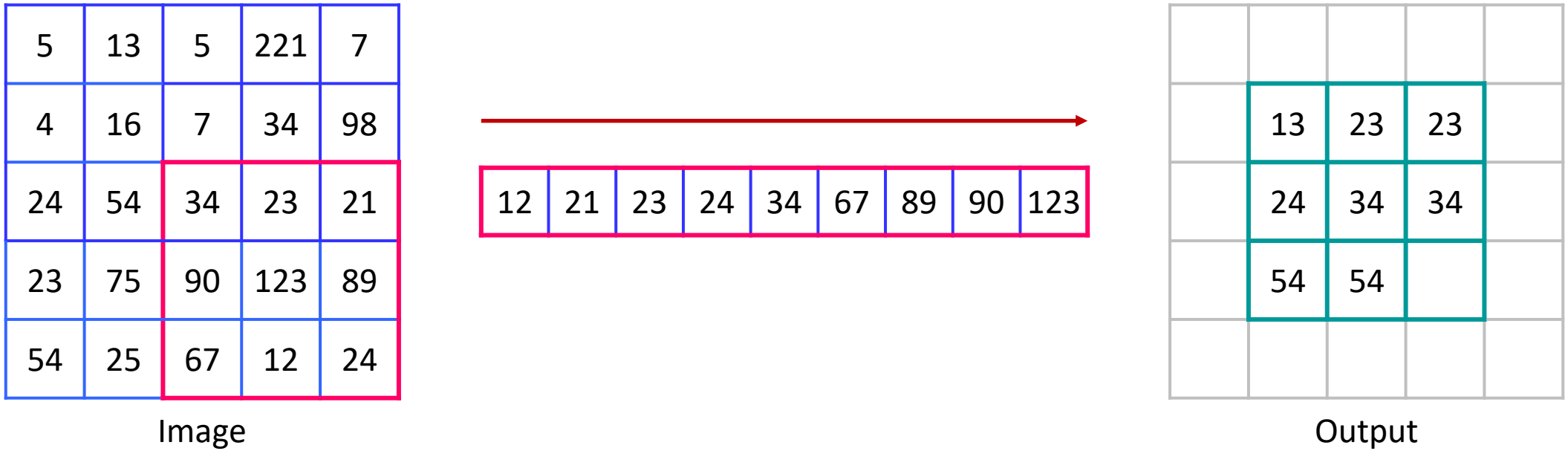
Image



Output

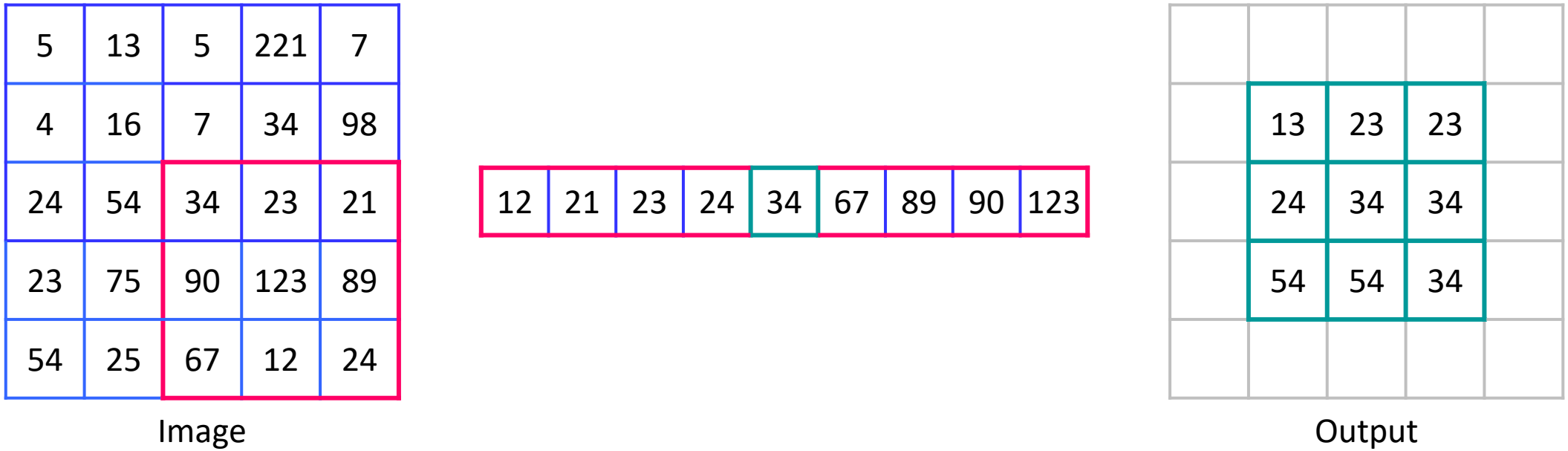
Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  - ① Sorting the pixel values in the filter
  - ② Pick the median value as an output pixel



Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter
  ② Pick the median value as an output pixel



Non-linear Filters: **Median Filter**

- **Median** filter takes the **median value** of the pixels under the filter
  ① Sorting the pixel values in the filter
  ② Pick the median value as an output pixel

5	13	5	221	7
4	16	7	34	98
24	54	34	23	21
23	75	90	123	89
54	25	67	12	24

Image

	13	23	23	
	24	34	34	
	54	54	34	

Output

Non-linear Filters: **Median Filter**

- Median filter is effective at **reducing certain kinds of noise**, such as **impulse noise** (a.k.a ‘salt and pepper’ noise or ’shot’ noise)
- The median filter forces points with distinct values to be more like their neighbors

https://en.wikipedia.org/wiki/Median_filter

Non-linear Filters: **Median Filter**

- Bilateral filter is an **edge-preserving** non-linear filter

- **Like** a Gaussian filter:

— The filter weights depend on spatial distance from the center pixel

— Pixels nearby (in space) should have greater influence than pixels far away

- **Unlike** a Gaussian filter:

— The filter weights also depend on range distance from the center pixel

— Pixels with similar brightness value should have greater influence than pixels with dissimilar brightness value

Non-linear Filters: **Bilateral Filter**

- **Gaussian** filter

— Weights of neighbor at a spatial offset (k,l) away from the center pixel (i,j) given by:

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

- **Bilateral** filter:

— Weights of neighbor at a spatial offset (k,l) away from the center pixel (i,j) given by a product:

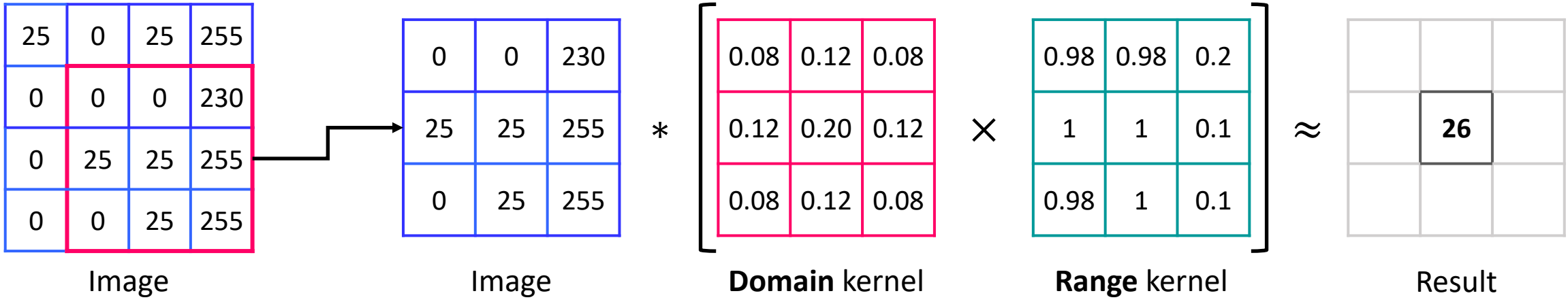
$$B_{\sigma_d,\sigma_r}(i,j,k,l) = e^{-\frac{(k^2+l^2)}{2\sigma_d^2}} e^{-\frac{\left(I(i+k,j+l)-I(i,j)\right)^2}{2\sigma_r^2}}$$

Domain kernel — **Range** kernel

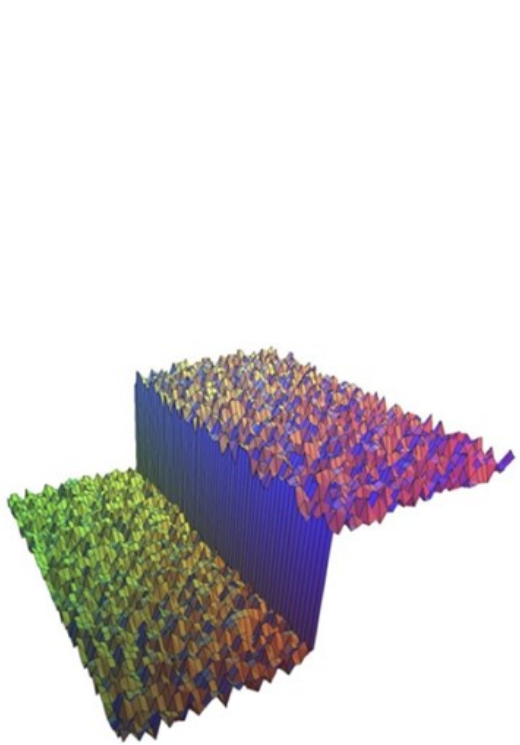
Non-linear Filters: **Bilateral Filter**

- **Domain** Kernel

- **Range×Domain** Kernel

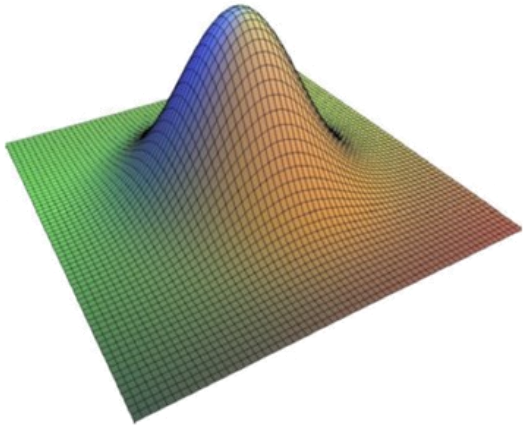


Non-linear Filters: **Bilateral Filter**

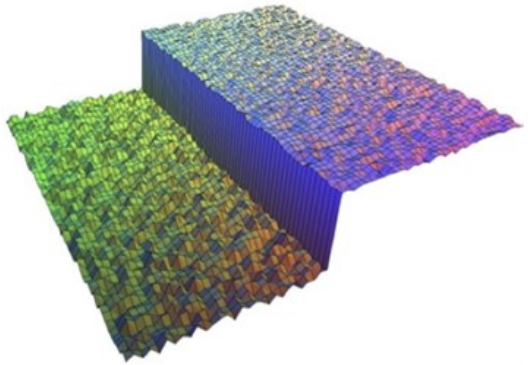


Input

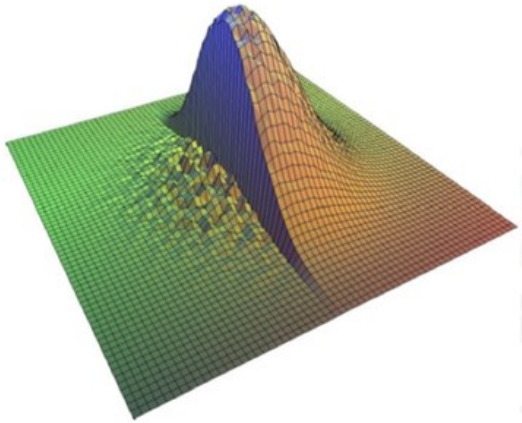
*



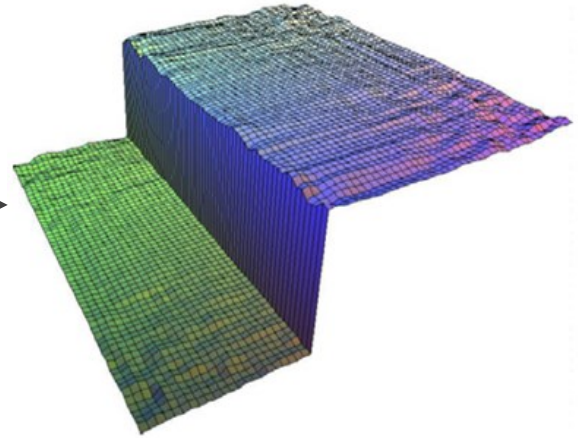
Domain (Spatial) Kernel



Range Kernel Influence



Bilateral Filter
(=Domain×Range)



Output

Durand and Dorsey, Image from Fast Bilateral Filtering for the Display of High-Dynamic Range Images, **ACM Trans. Graphics (ToG), 2002**

Bilateral Filter: Applications

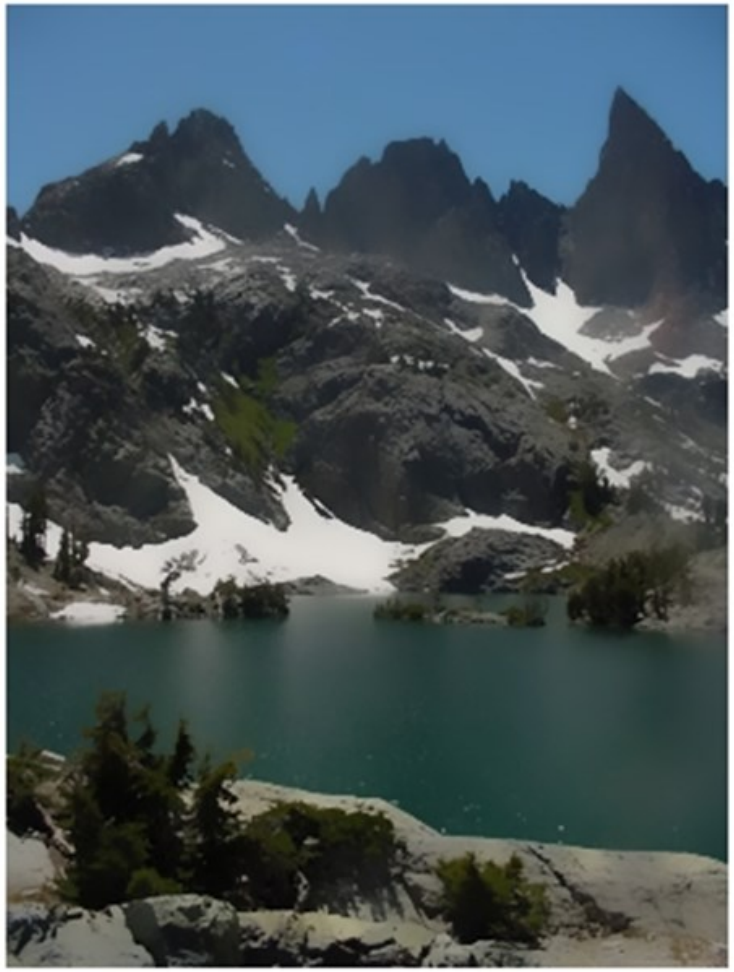
Noisy image

Result of **Gaussian** filter

Result of **Bilateral** filter

Bilateral Filter: Applications

Original image



Result of **Bilateral** filter

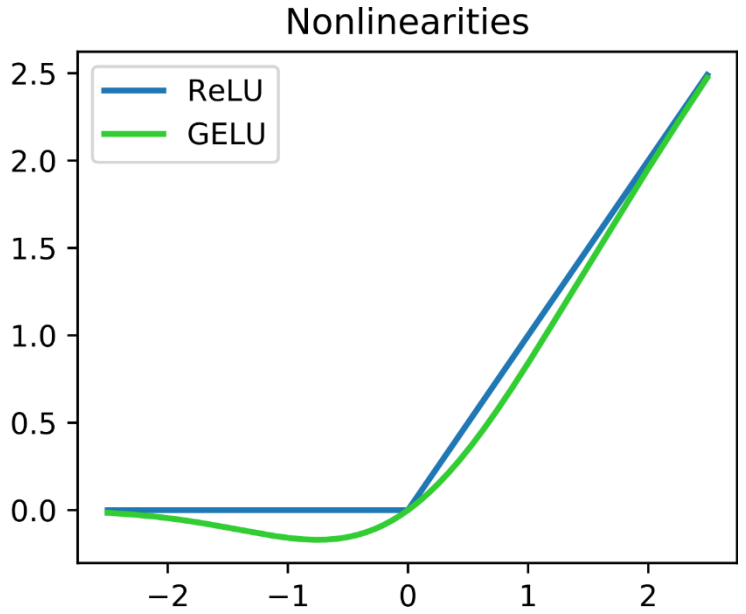
https://en.wikipedia.org/wiki/Bilateral_filter

Non-linear Filters: **ReLU**

- In the context of neural networks, the rectifier or ReLU (Rectified Linear Unit) is **an activation function** defined as the positive part of its argument:

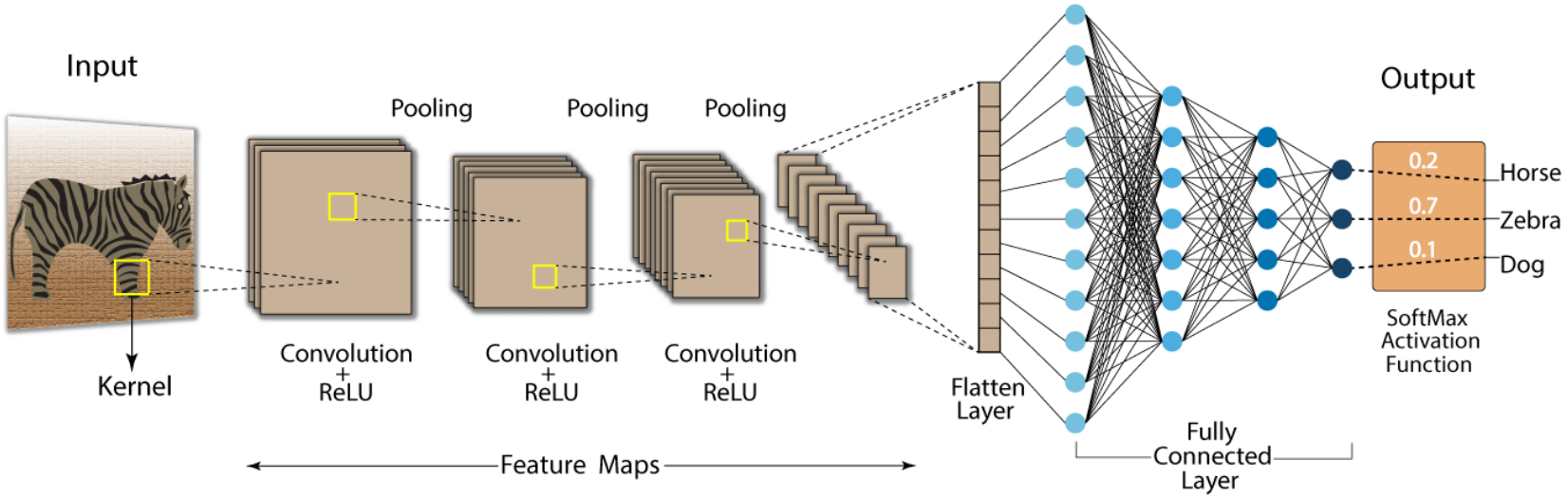
$$f(x) = \max(0, x)$$

- This is also known as **a ramp function** and is analogous to half-wave rectification in electrical engineering

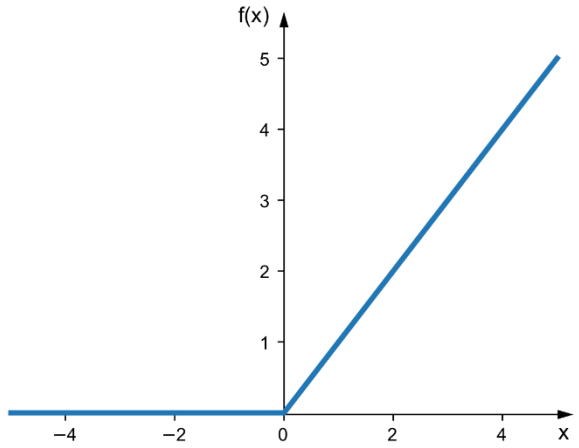


https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

Non-linear Filters: Linear Filter with **ReLU**



-5	13	5	-10	7
4	16	7	34	-8
24	-54	34	23	21
23	75	90	-21	89
-24	25	-27	12	24



0	13	5	0	7
4	16	7	34	0
24	0	34	23	21
23	75	90	0	89
0	25	0	12	24

Result of linear filtering (**Convolution**)

Result of non-linear filtering (**ReLU**)

Summary: **Non-Linear Filter**

- Three different non-linear filters: Median, Bilateral, and ReLU

— The **median filter** is a non-linear filter that selects the median value in the neighborhood

— The **bilateral filter** is a non-linear filter that considers both spatial distance and range (intensity) distance, and has edge-preserving properties.

— The **ReLU (Rectified Linear Unit)** is a non-linear activation function defined as the positive part of its argument