Convolutional Neural Networks

Elena Congeduti, 20-11-2024

TU Delft

Lecture’s Agenda

1. Structured data
2. Convolutions
3. Convolutional Layers
4. Relation to Fully Connected Layers

What makes images different?

What makes images different?

1. High dimensionality

What makes images different?

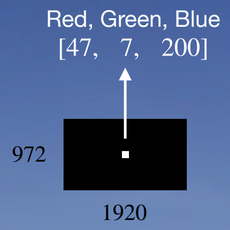
1. High dimensionality

Size (972, 1920) RGB pixel

What makes images different?

1. High dimensionality

Size (972, 1920) RGB pixel

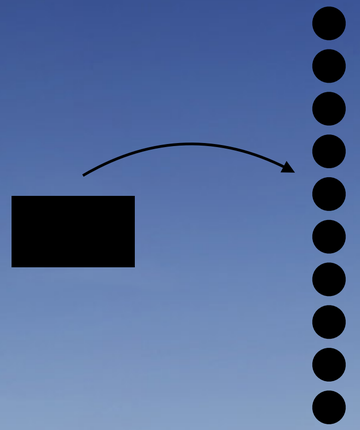


What makes images different?

1. High dimensionality

Size (972, 1920) RGB pixel

Fully connected layers



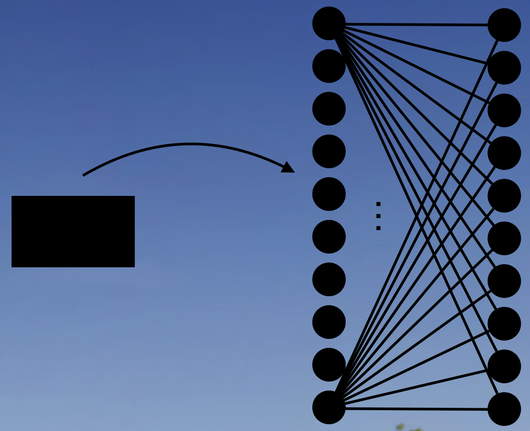
#input variables ≈ 5M

What makes images different?

1. High dimensionality

Size (972, 1920) RGB pixel

Fully connected layers



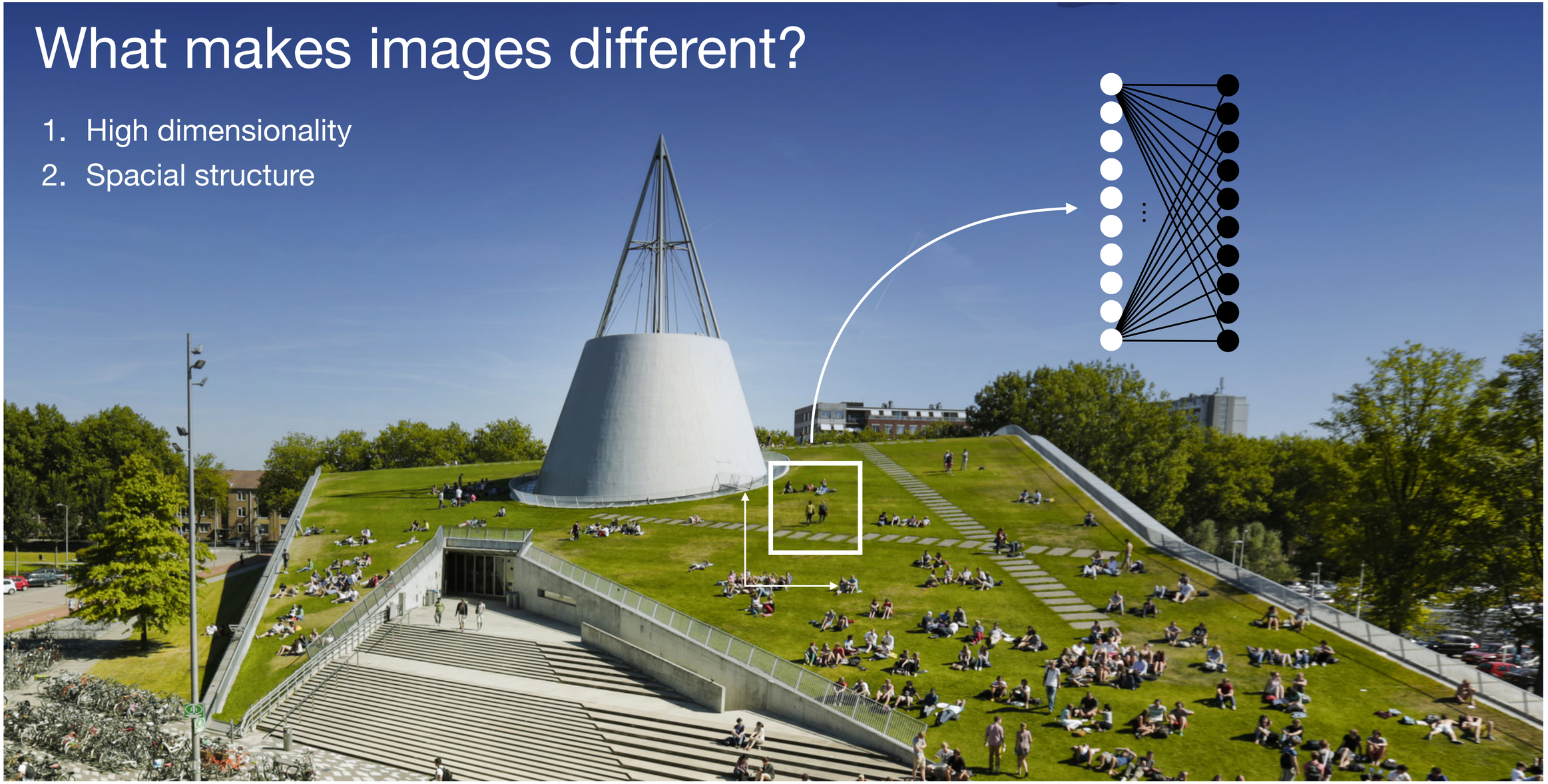
#weights ≈ 25 trillions

What makes images different?

1. High dimensionality
2. Spacial structure

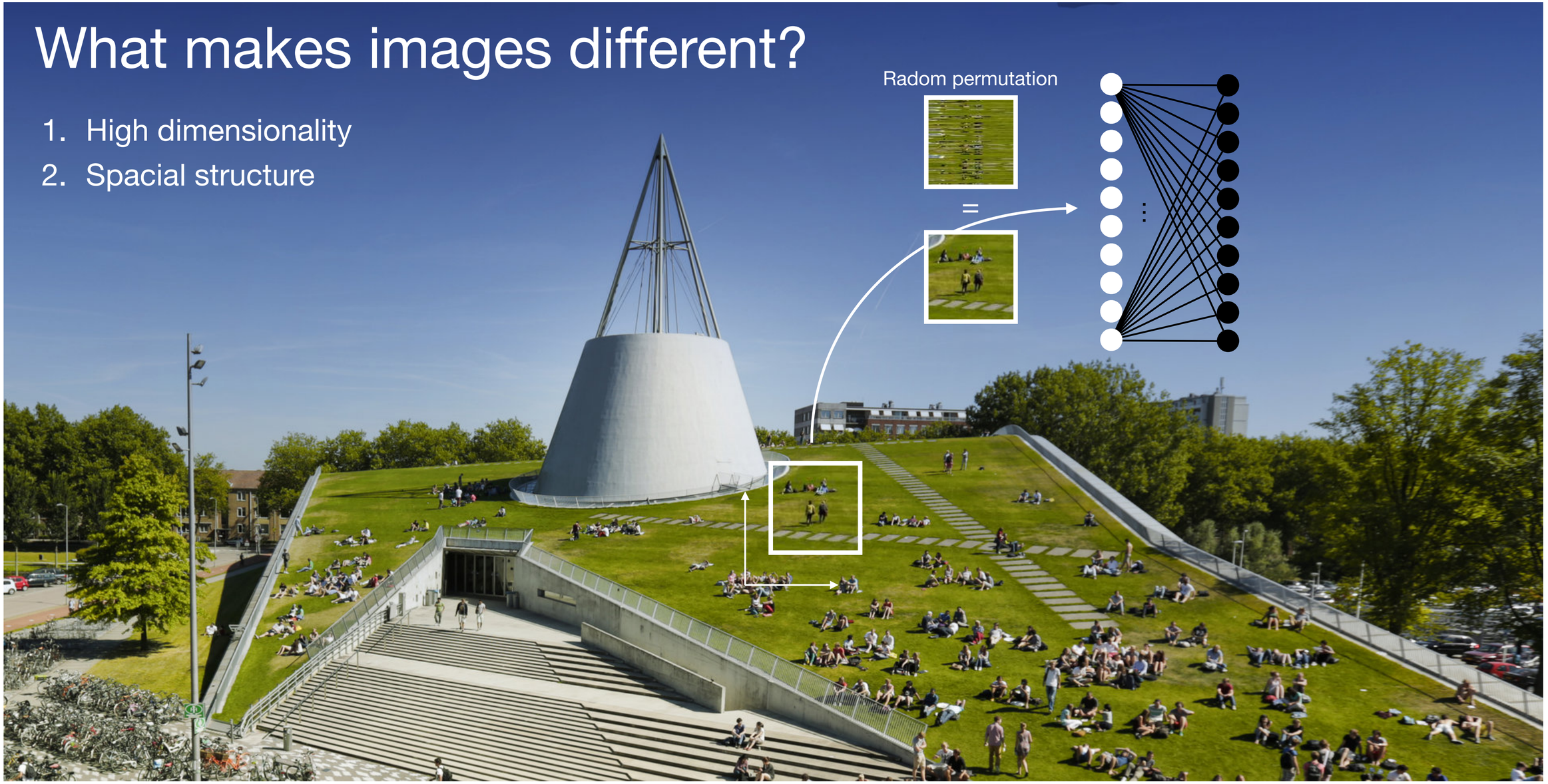
What makes images different?

1. High dimensionality
2. Spacial structure



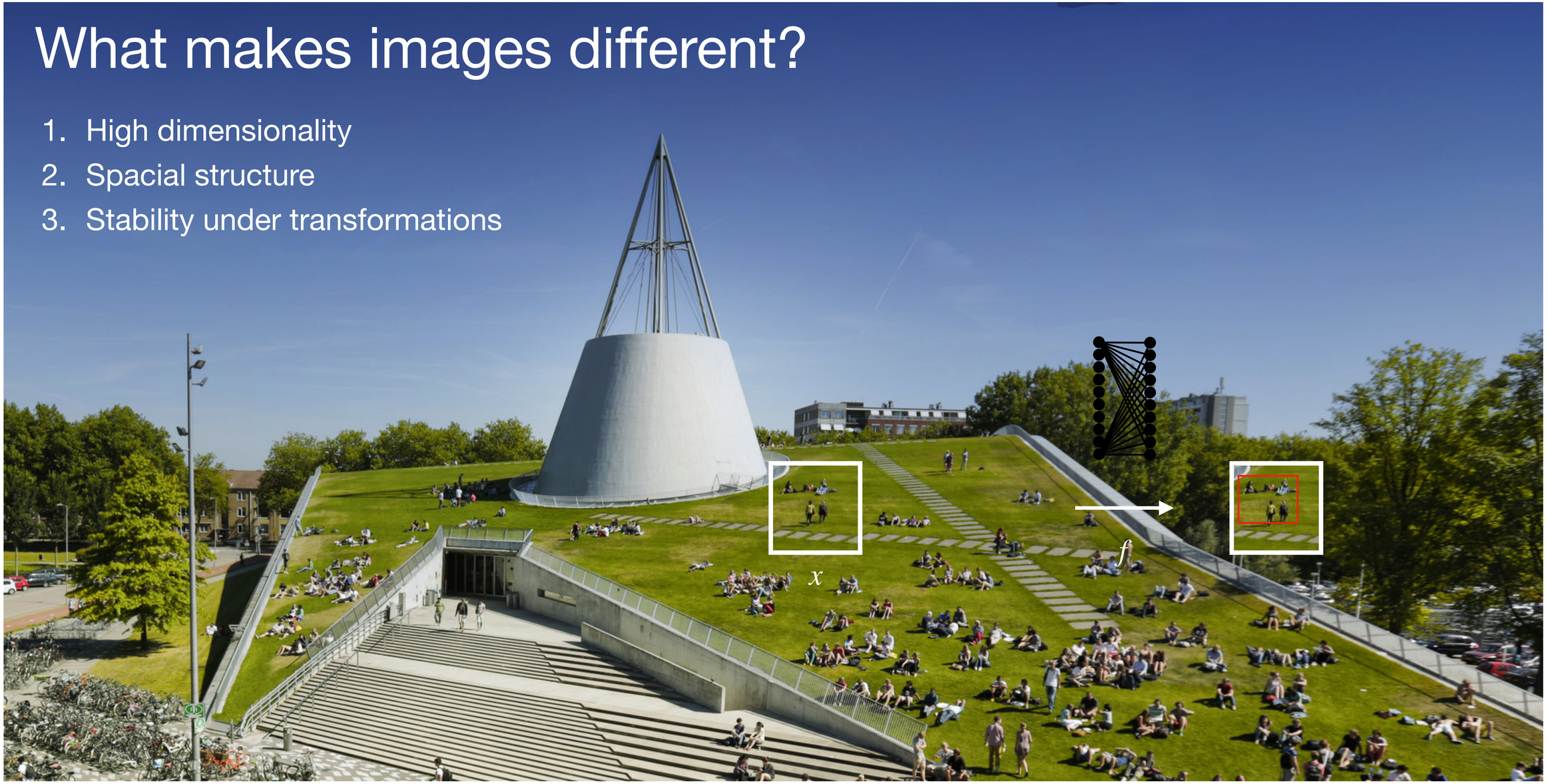
What makes images different?

1. High dimensionality
2. Spacial structure



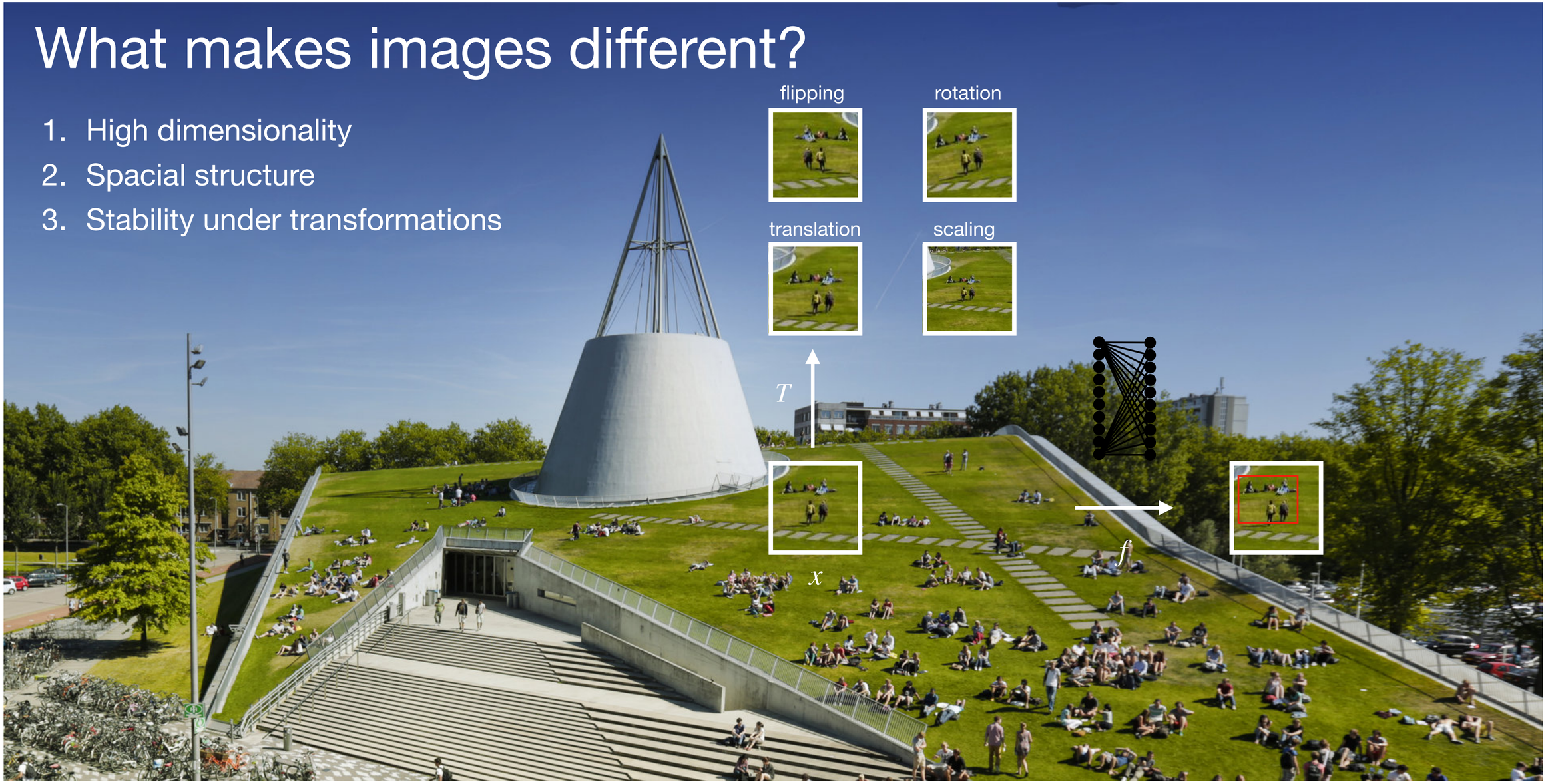
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



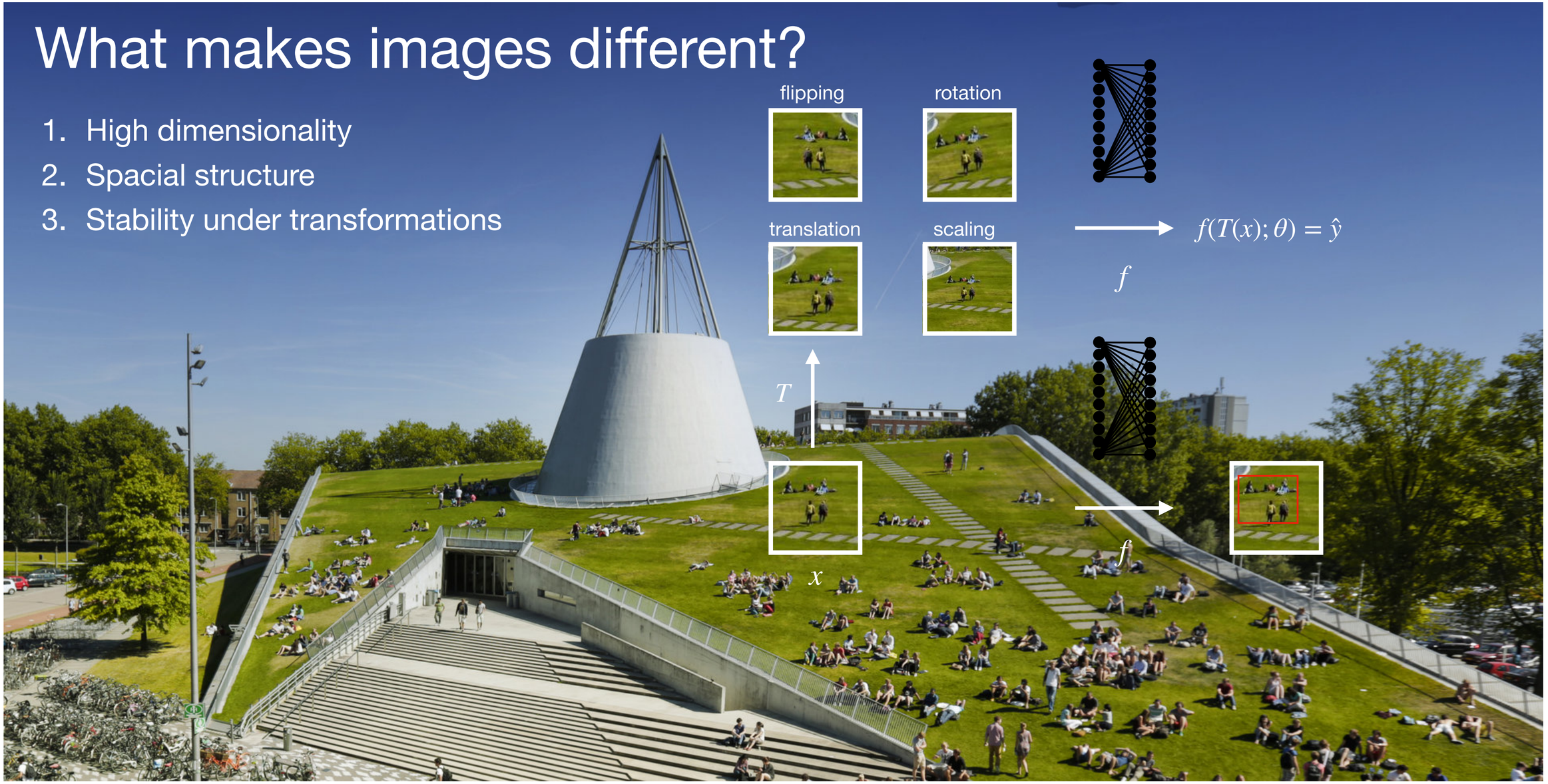
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



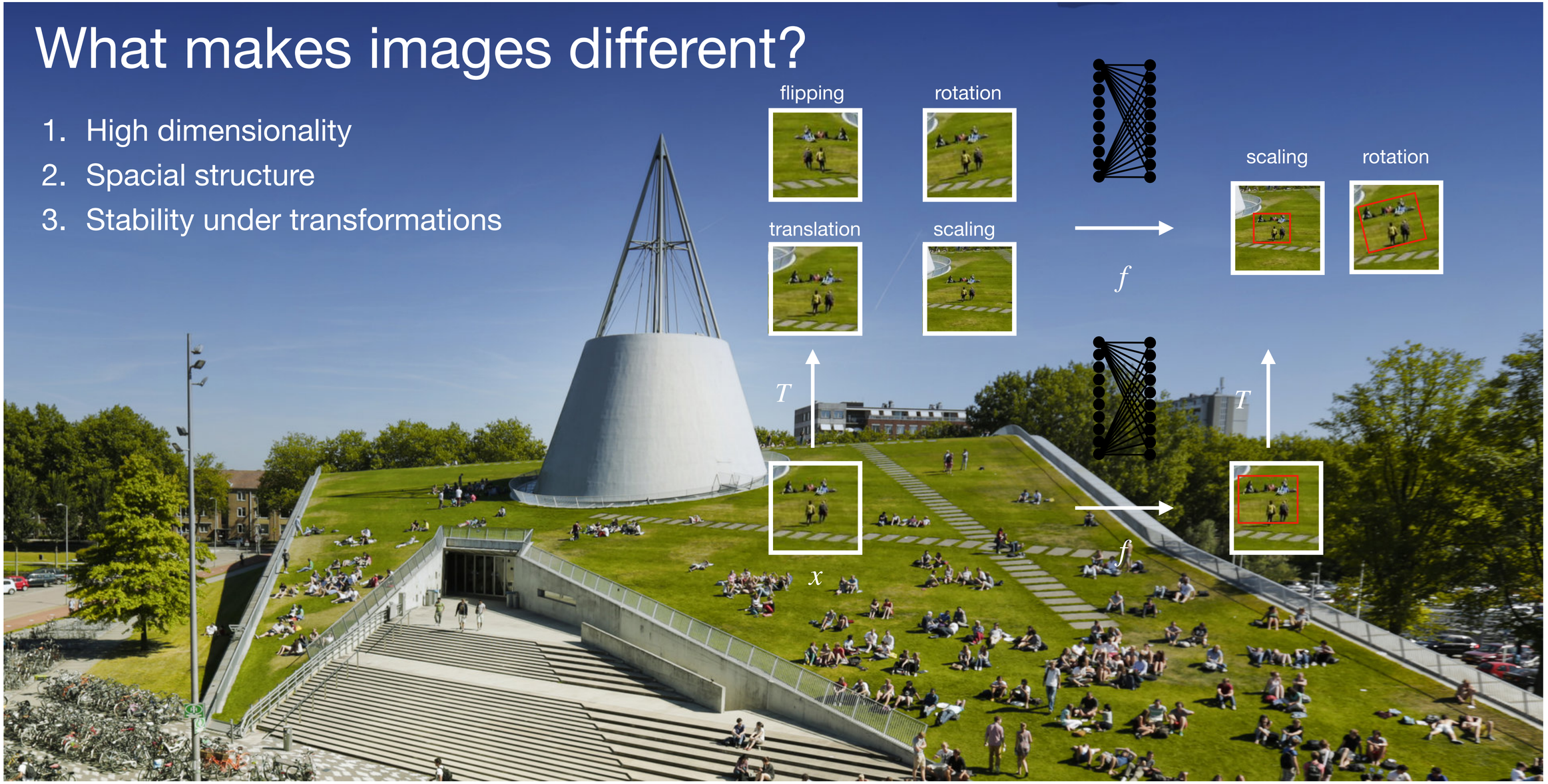
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



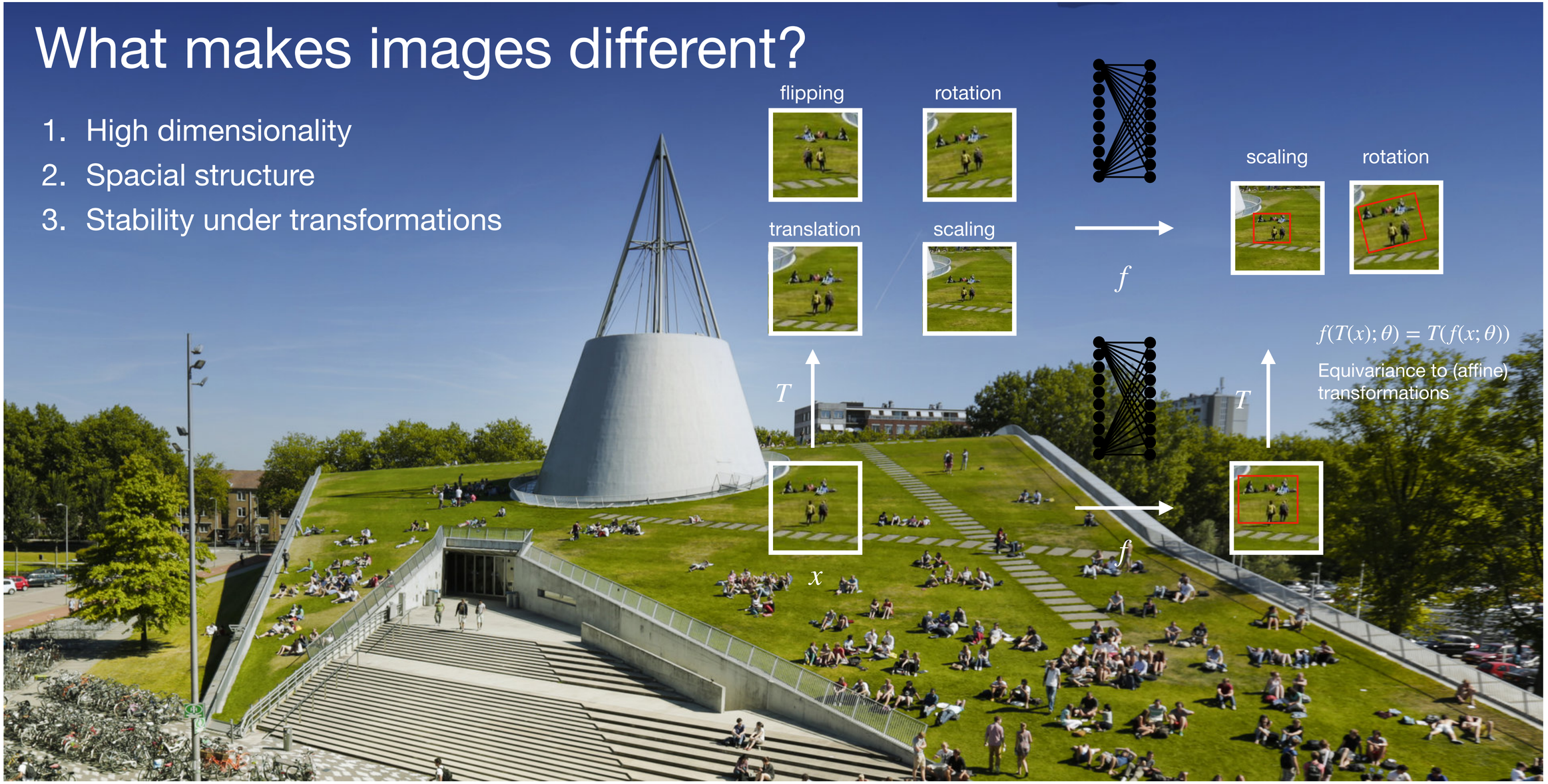
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



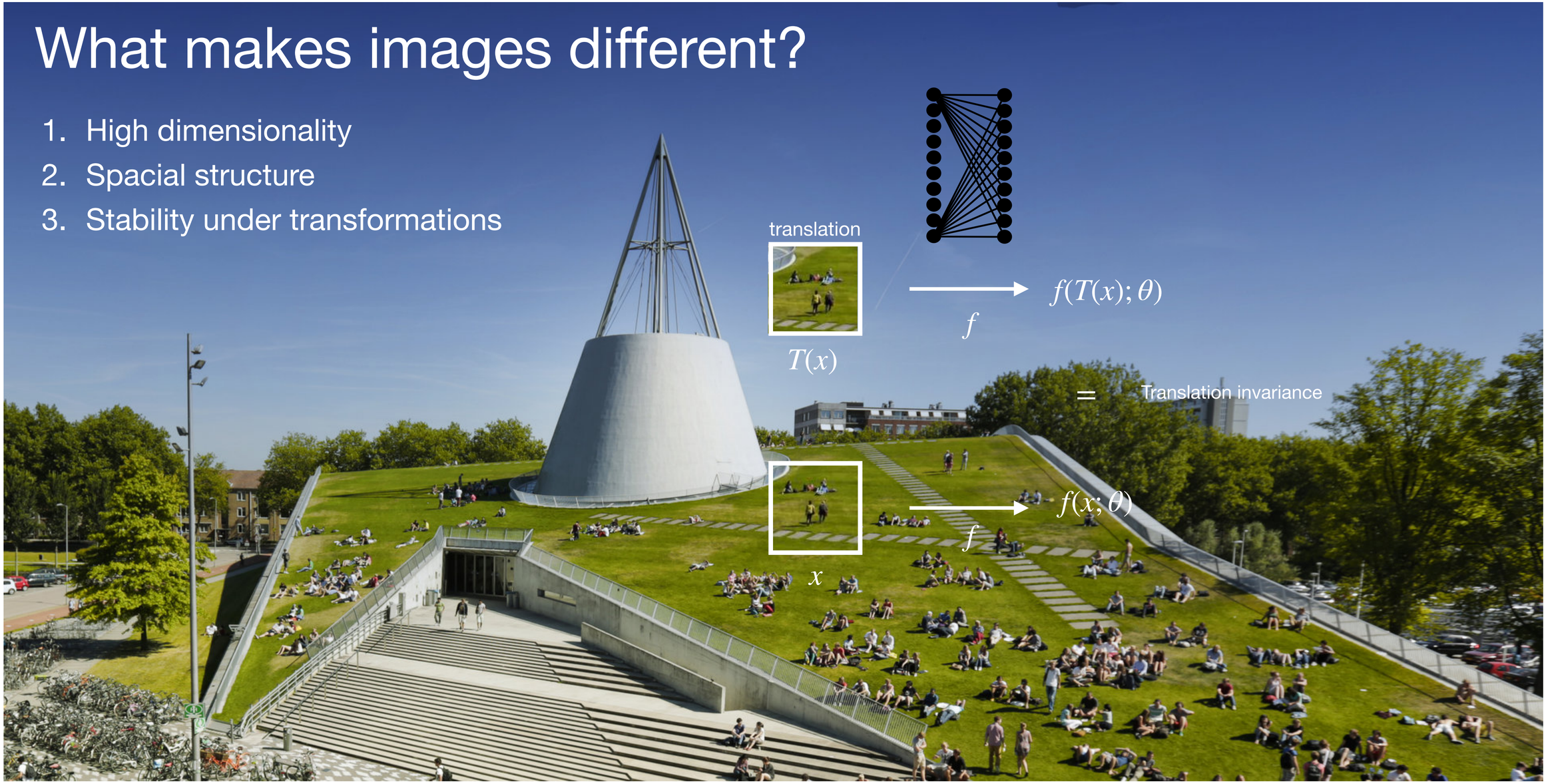
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



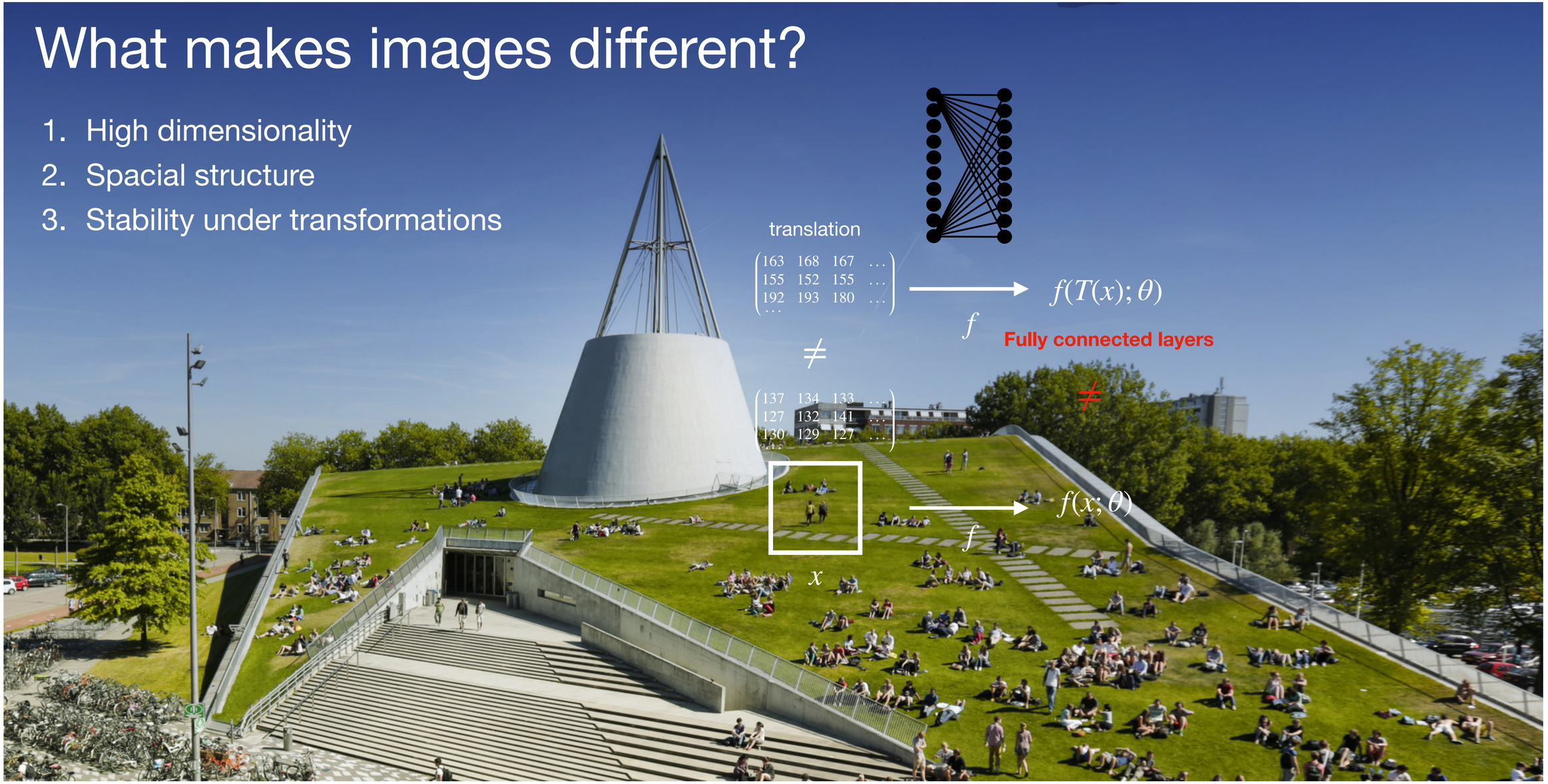
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



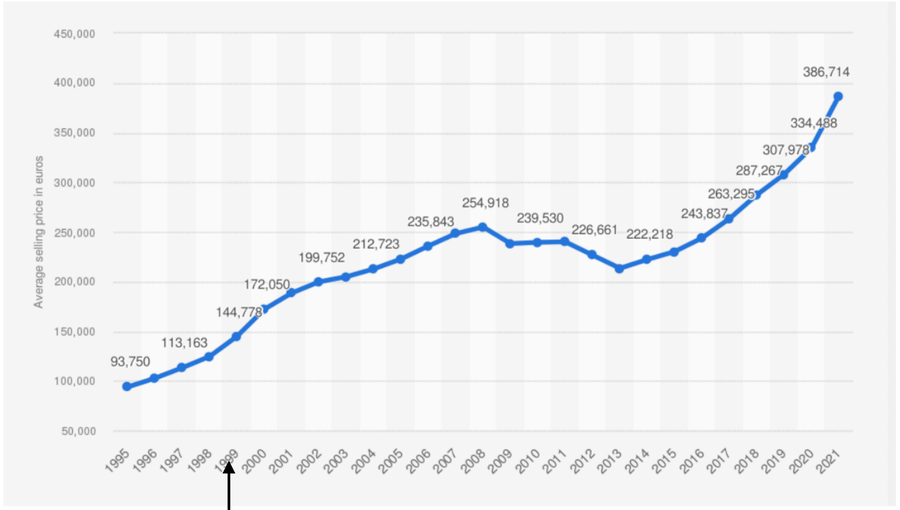
What makes images different?

1. High dimensionality
2. Spacial structure
3. Stability under transformations



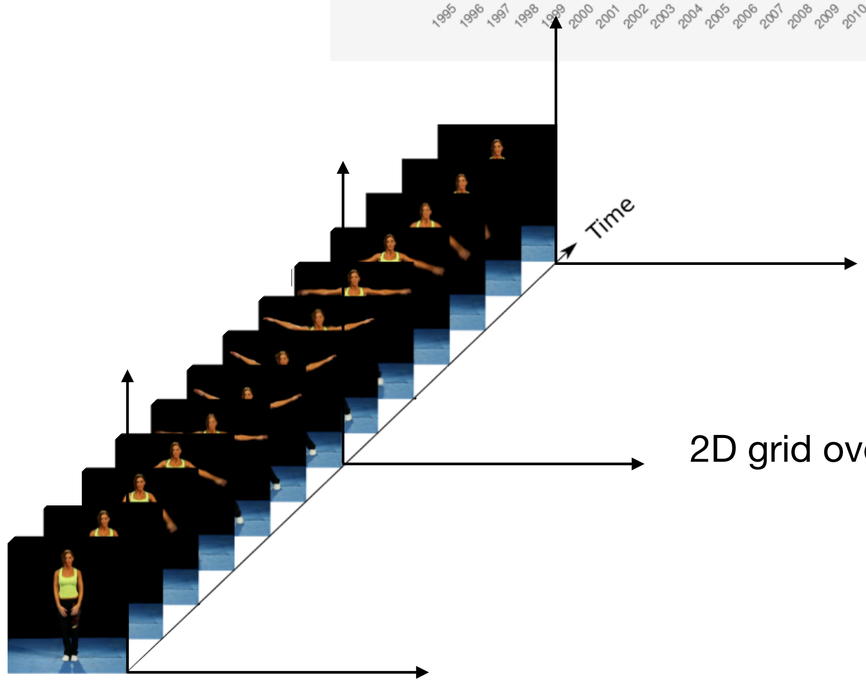
Grid topology

1. Time series
2. Video
3. Audio
4. Text



$$(93750, 113163, 144778, 172050, \ldots) \xrightarrow{t}$$

1D grid - time dimension



2D grid over time dimension

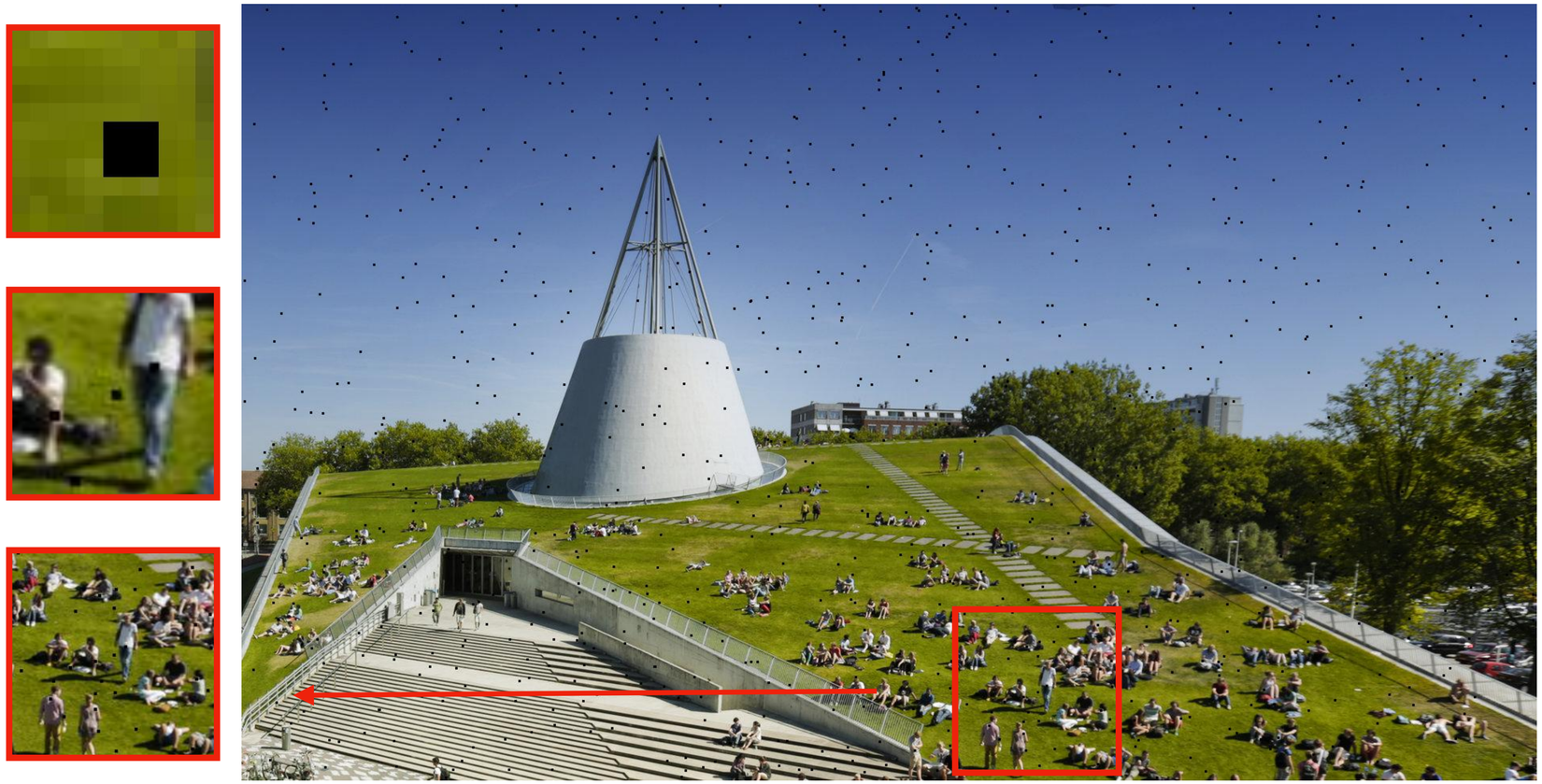
Convolutions

Image processing example

Simple solution for noise removal ⟶ neighbourhood average to replace the lost pixels

Convolutions

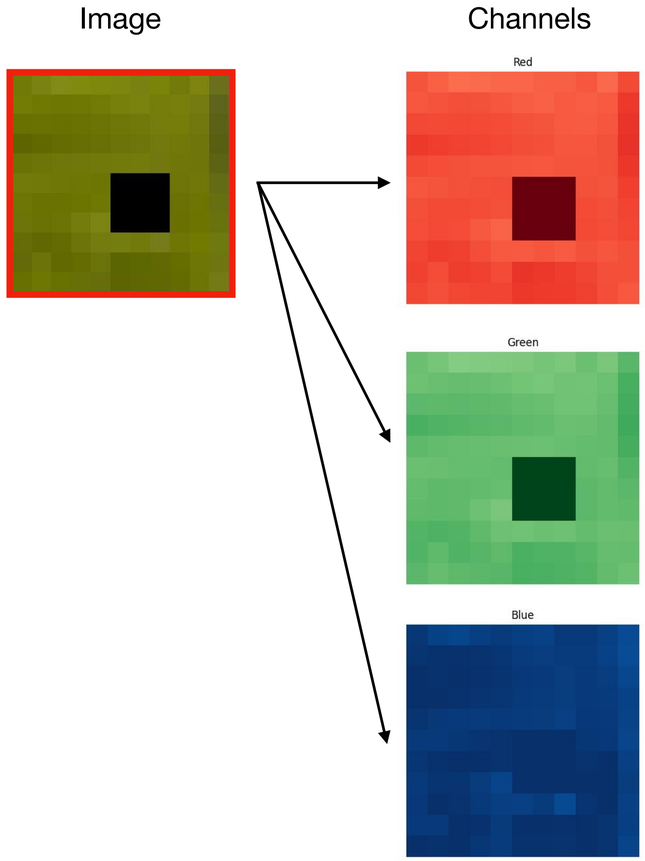
Image processing example



Simple solution for noise removal ⟶ neighbourhood average to replace the lost pixels

Convolutions

Image processing example

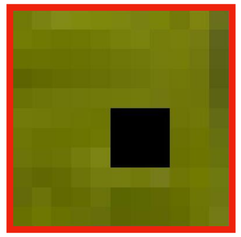


Simple solution for noise removal ⟶ neighbourhood average to replace the lost pixels

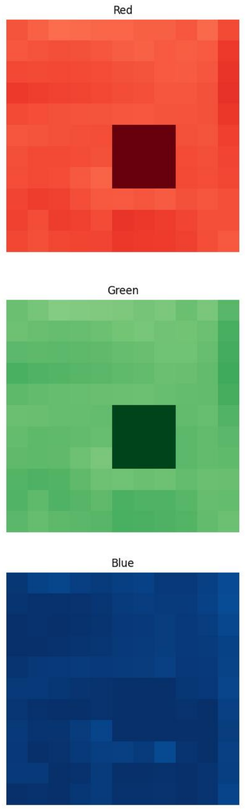
Convolutions

Image processing example

Image



Channels



Pixel matrix

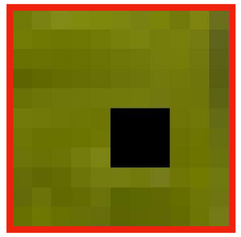
113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Simple solution for noise removal → neighbourhood average to replace the lost pixels

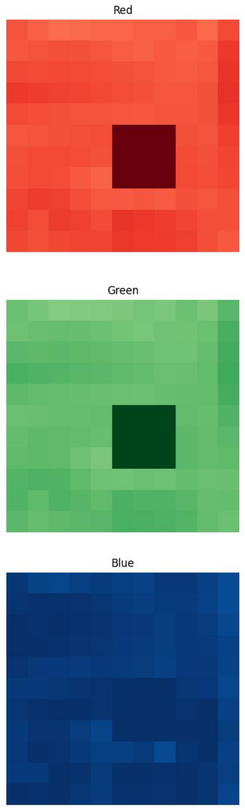
Convolutions

Image processing example

Image



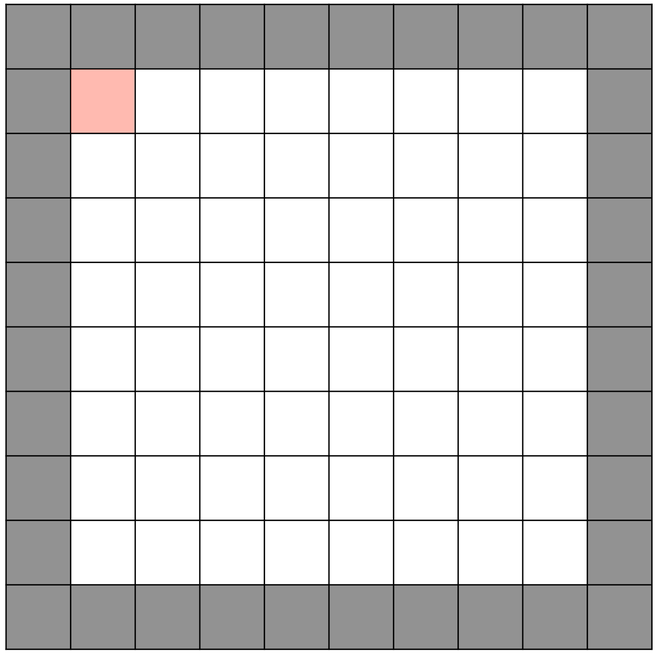
Channels



Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

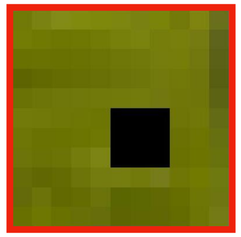


Simple solution for noise removal → neighbourhood average to replace the lost pixels

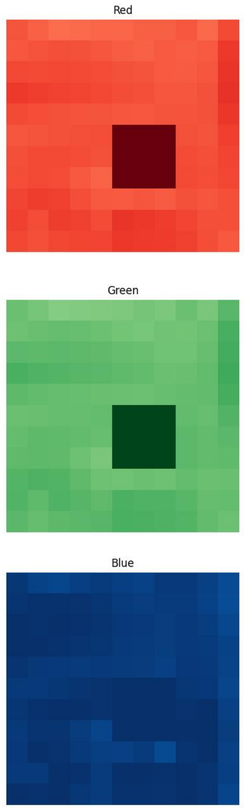
Convolutions

Image processing example

Image



Channels

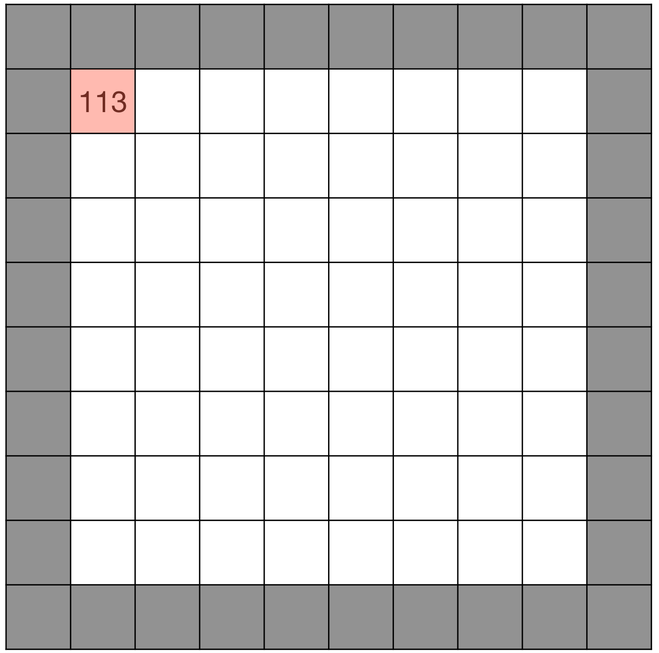


Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)

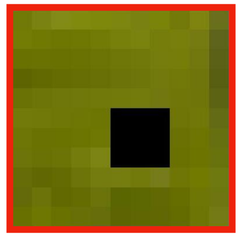


Simple solution for noise removal → neighbourhood average to replace the lost pixels

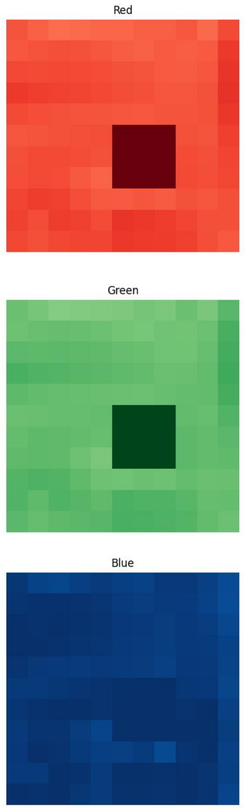
Convolutions

Image processing example

Image



Channels



Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)

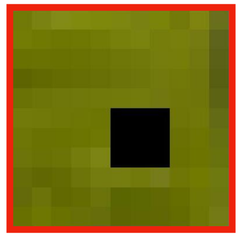
	113	114							

Simple solution for noise removal → neighbourhood average to replace the lost pixels

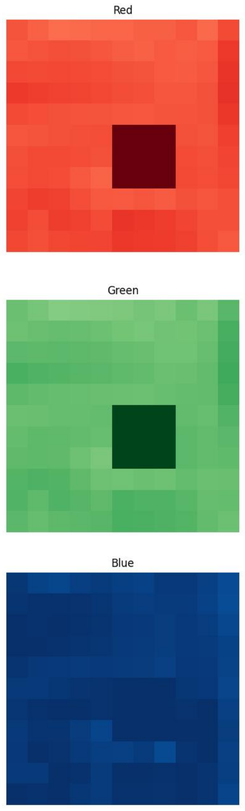
Convolutions

Image processing example

Image



Channels



Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)

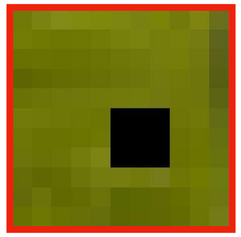
	113	114	115						

Simple solution for noise removal → neighbourhood average to replace the lost pixels

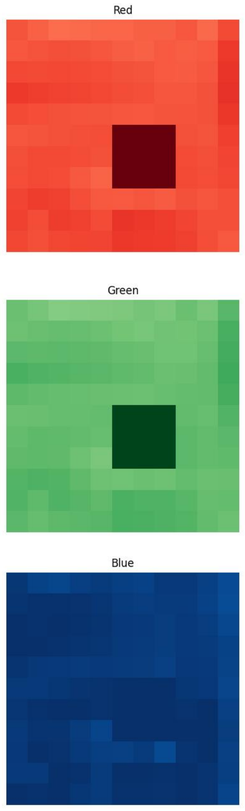
Convolutions

Image processing example

Image



Channels



Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)

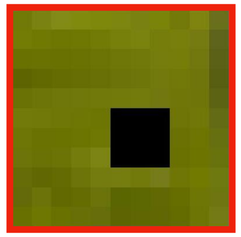
	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97					

Simple solution for noise removal → neighbourhood average to replace the lost pixels

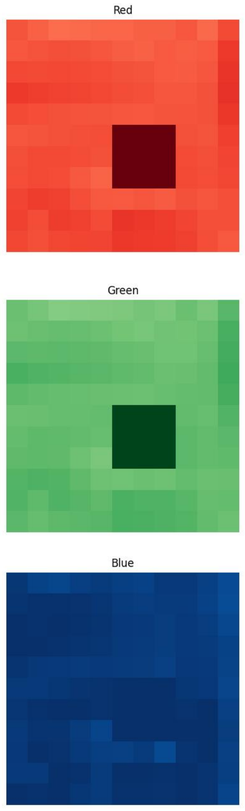
Convolutions

Image processing example

Image



Channels



Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)

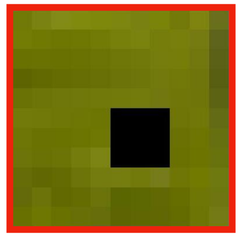
	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86				

Simple solution for noise removal → neighbourhood average to replace the lost pixels

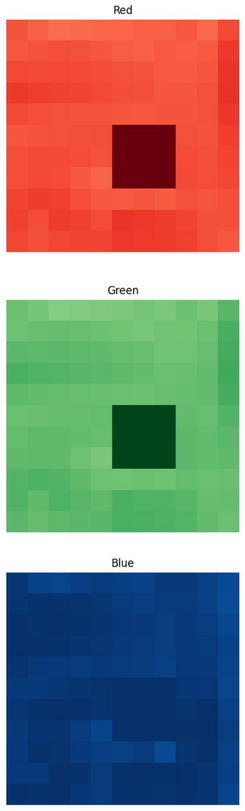
Convolutions

Image processing example

Image



Channels



Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)

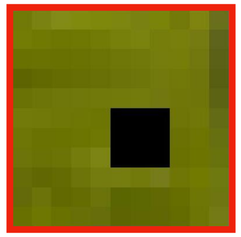
	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

Simple solution for noise removal ⟶ neighbourhood average to replace the lost pixels

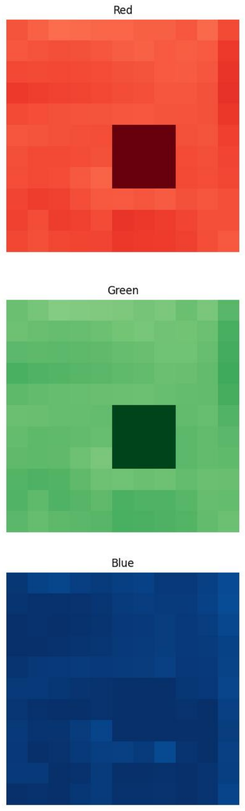
Convolutions

Image processing example

Image



Channels

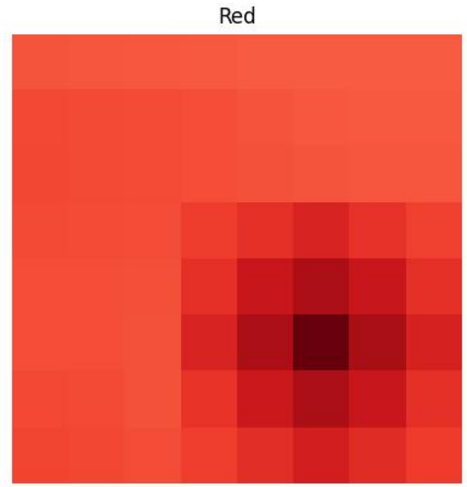


Pixel matrix

113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

Moving average filter

(rounded to integers)



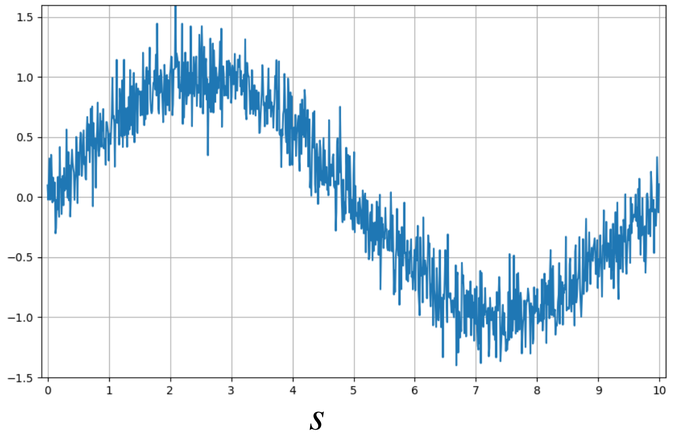
Simple solution for noise removal → neighbourhood average to replace the lost pixels

Convolutions

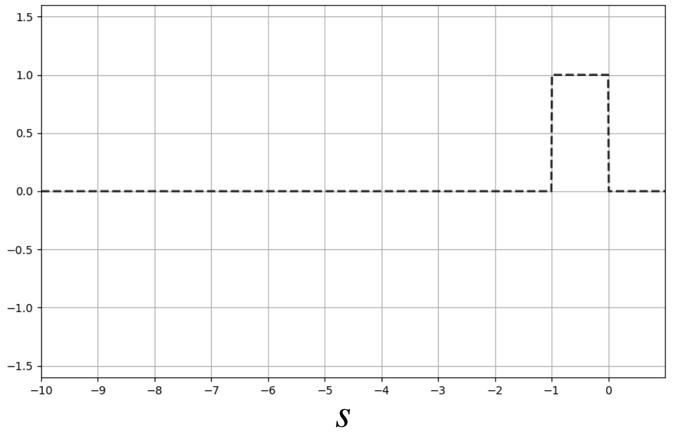
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

$x(s)$



$k(s)$

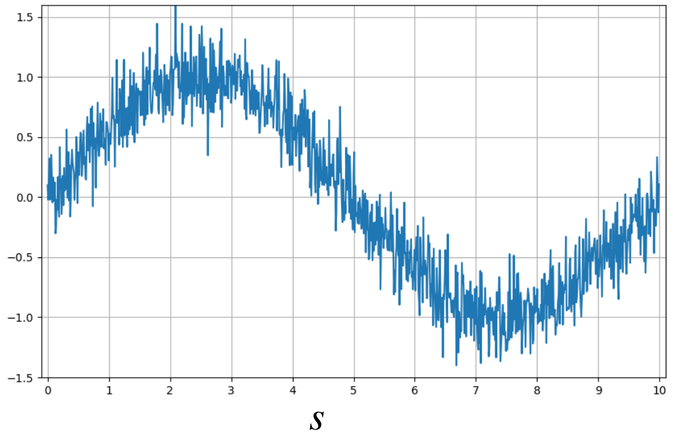


Convolutions

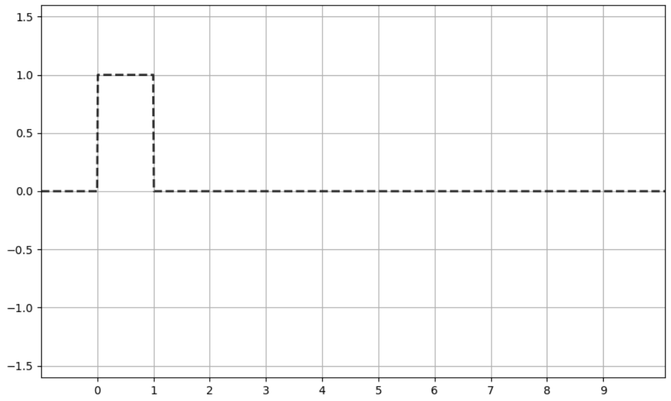
General definition of convolution between an input signal x and the kernel k

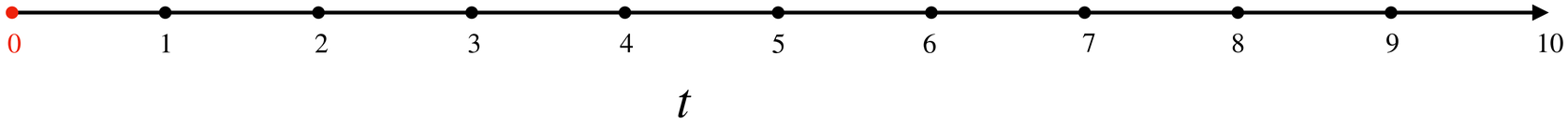
$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

$x(s)$



$k(-s)$

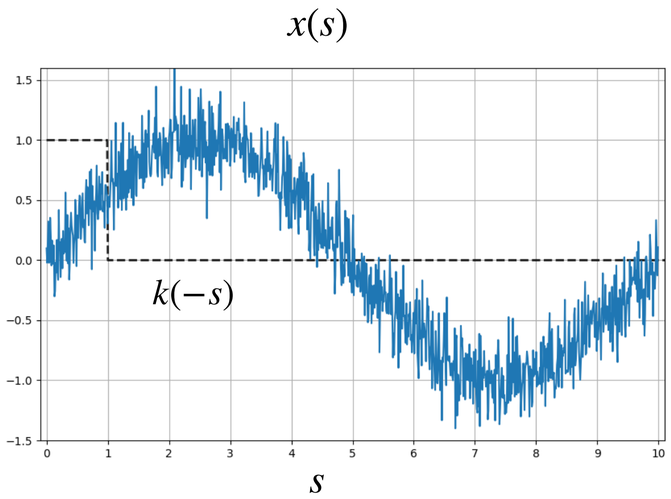


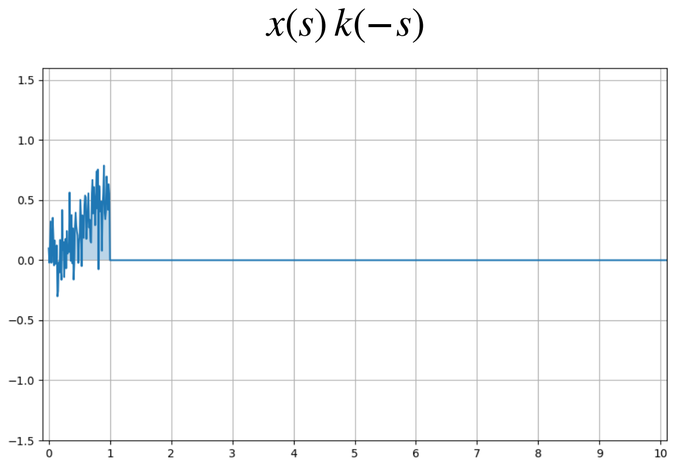


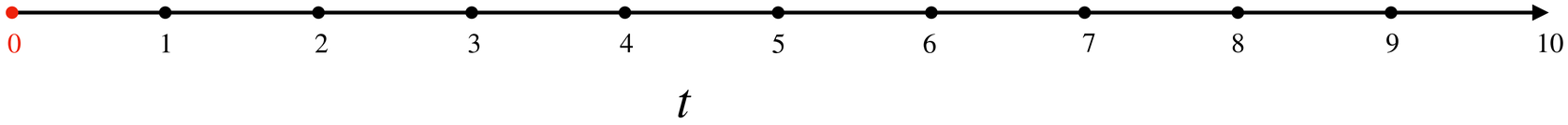
Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$



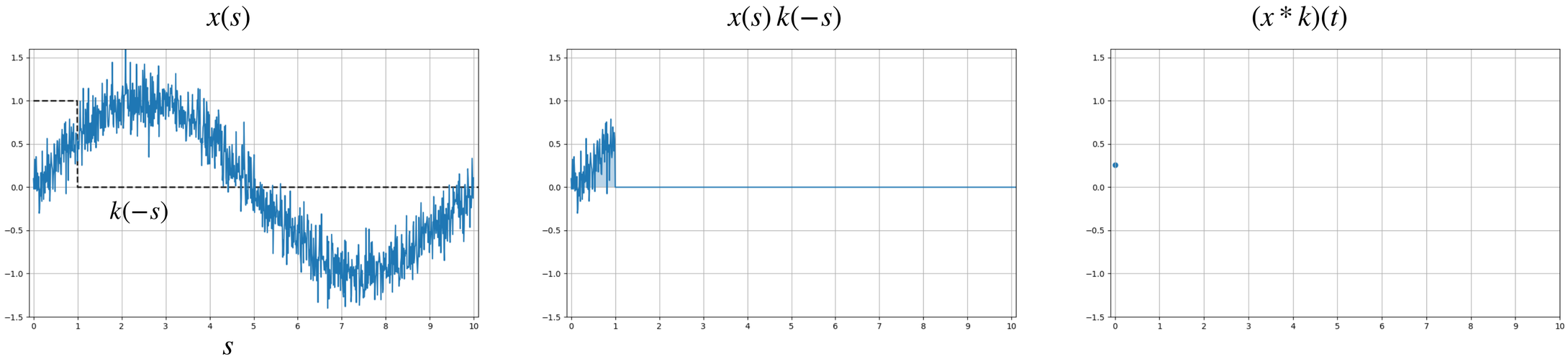




Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

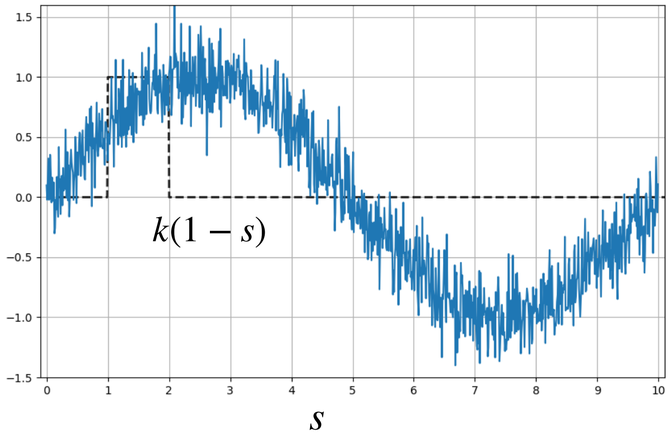


Convolutions

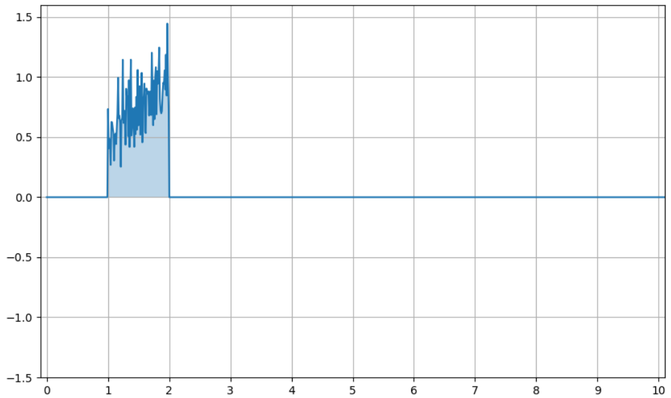
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

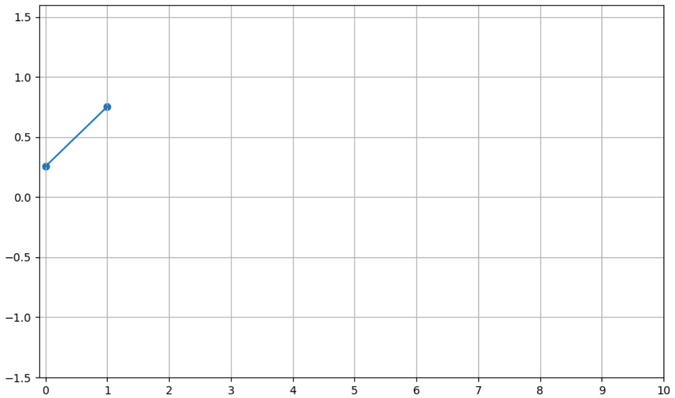
$x(s)$

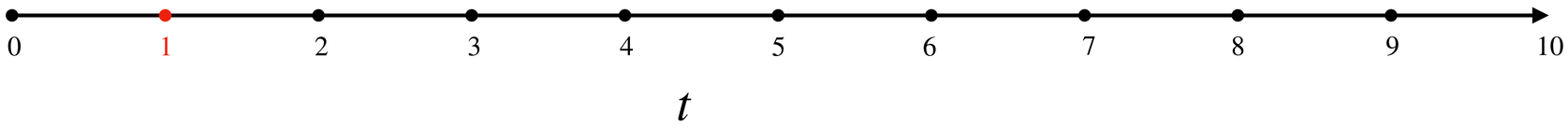


$x(s)\, k(1 - s)$



$(x * k)(t)$



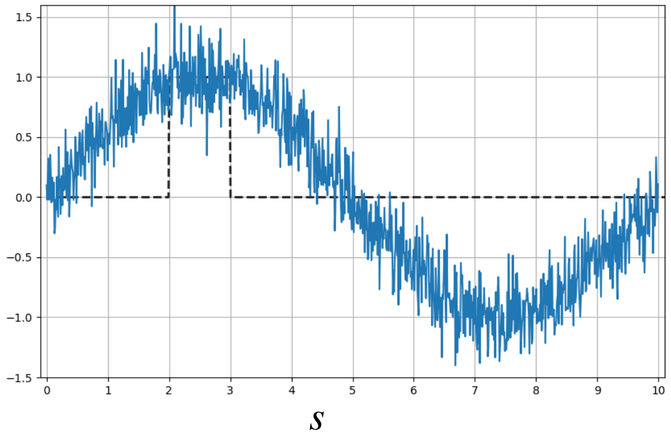


Convolutions

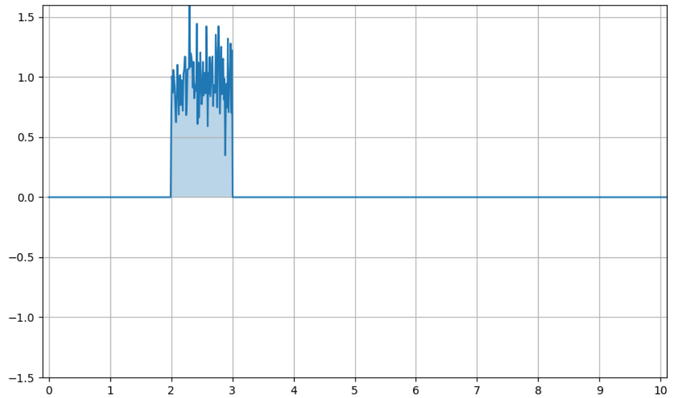
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

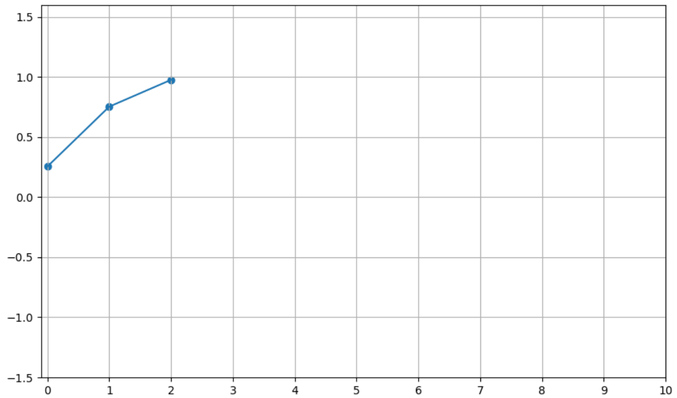
$x(s)$

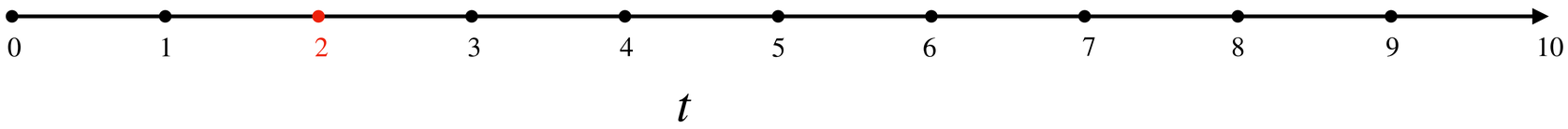


$x(s)\, k(2 - s)$



$(x * k)(t)$

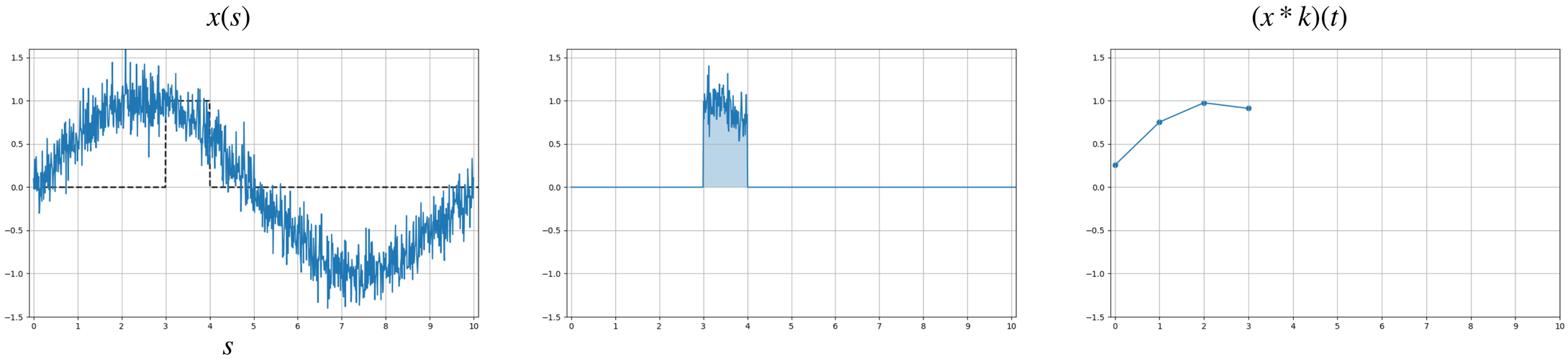


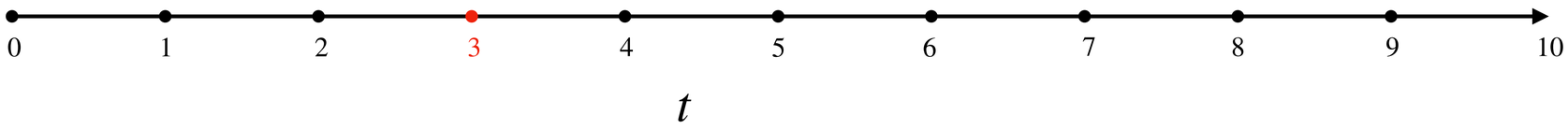


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

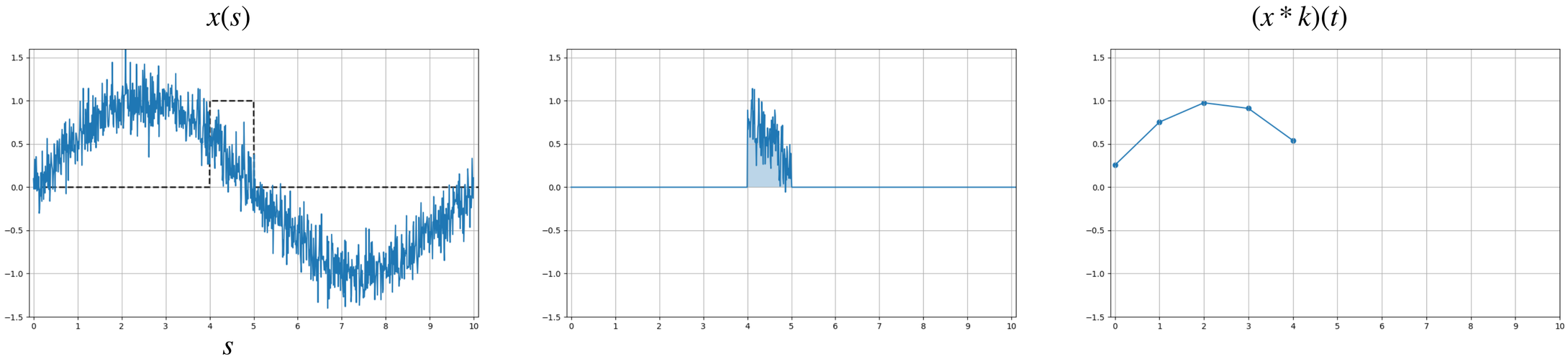


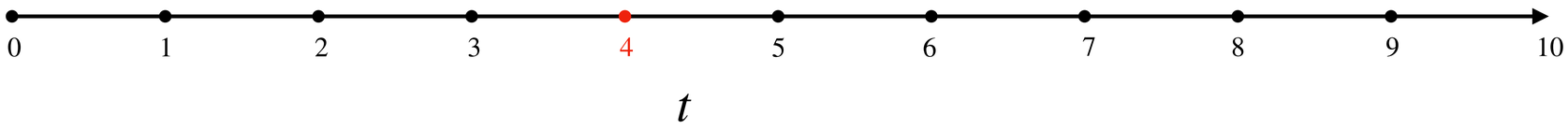


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

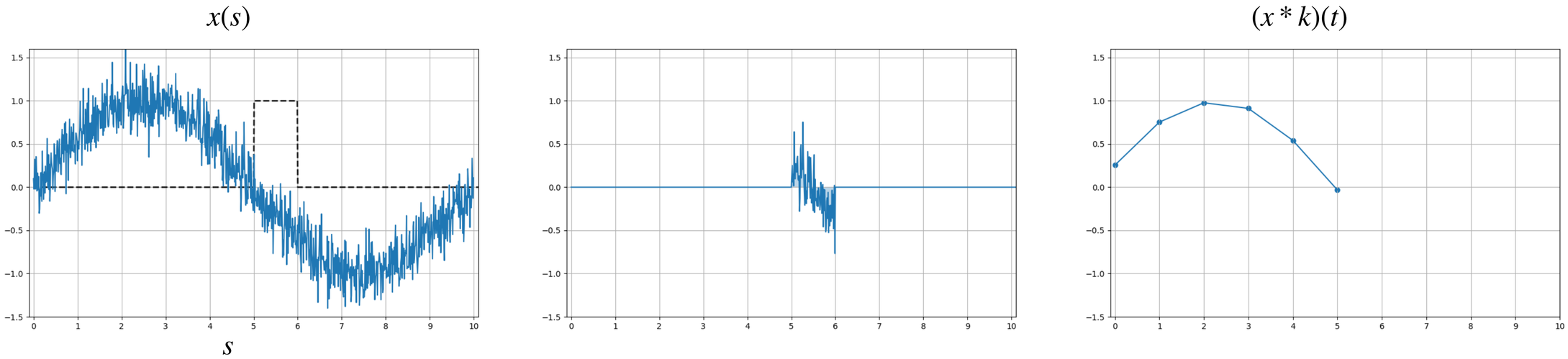


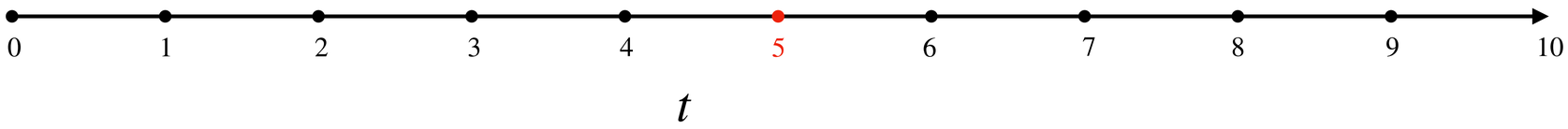


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

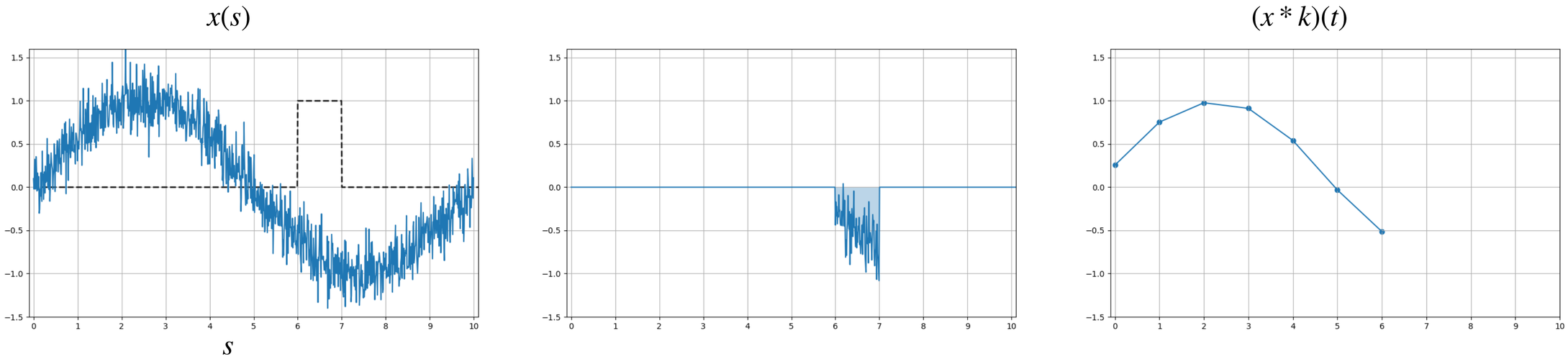


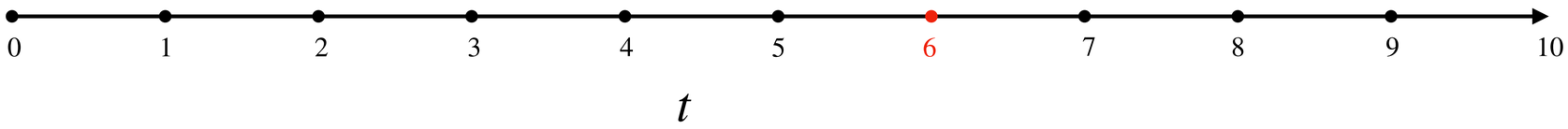


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

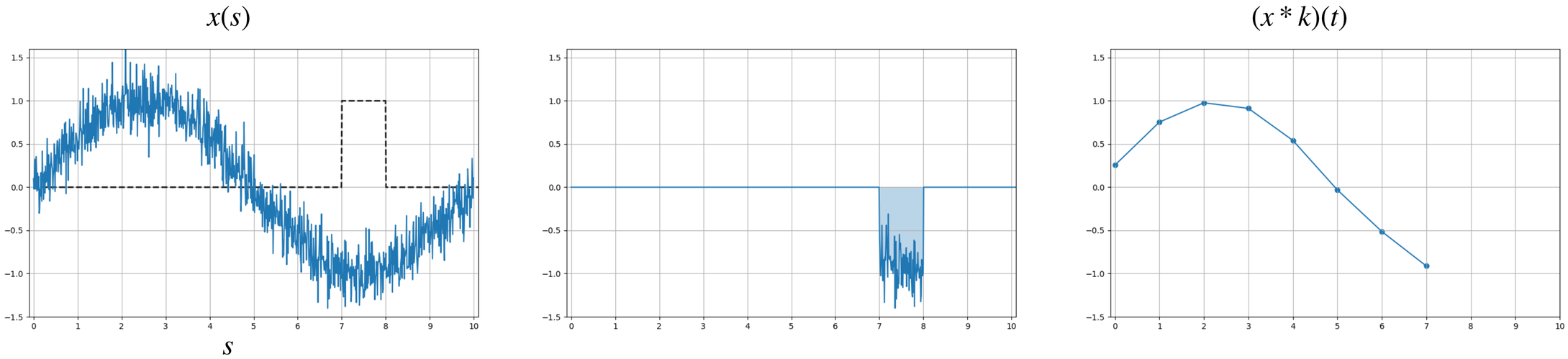


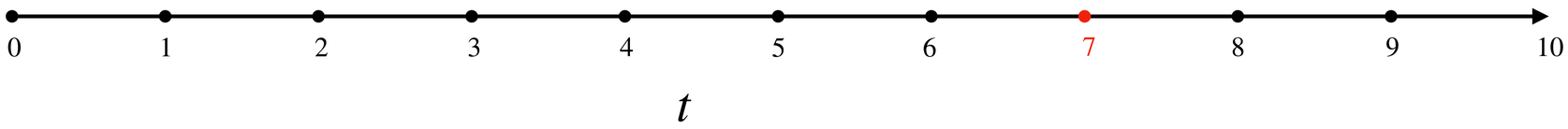


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

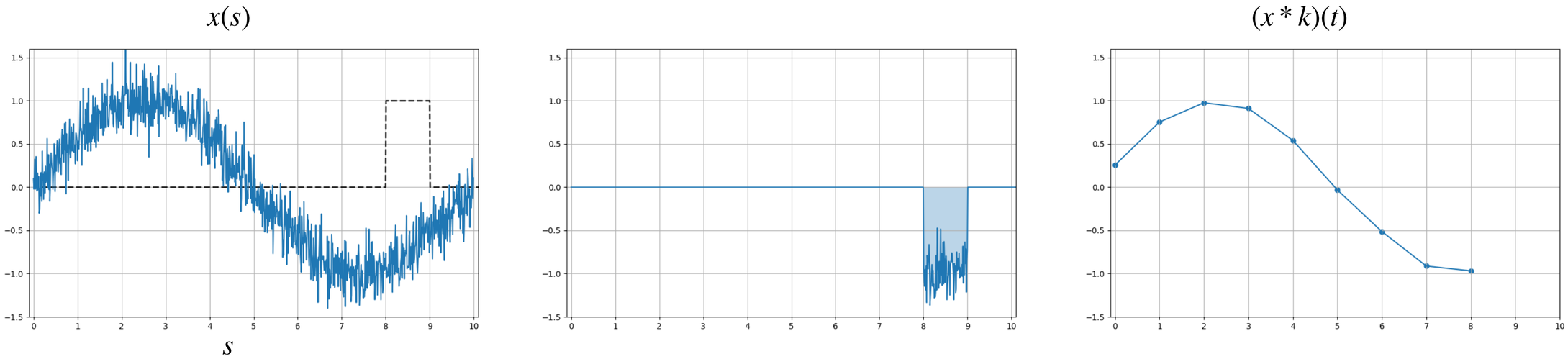


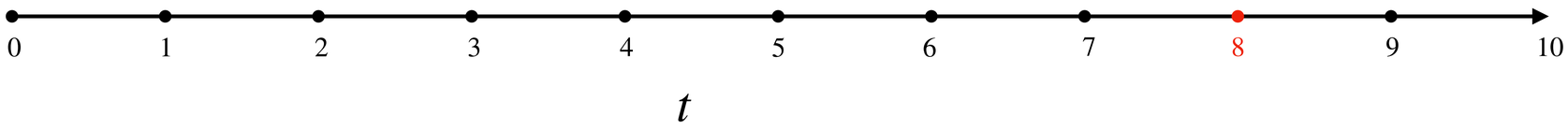


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

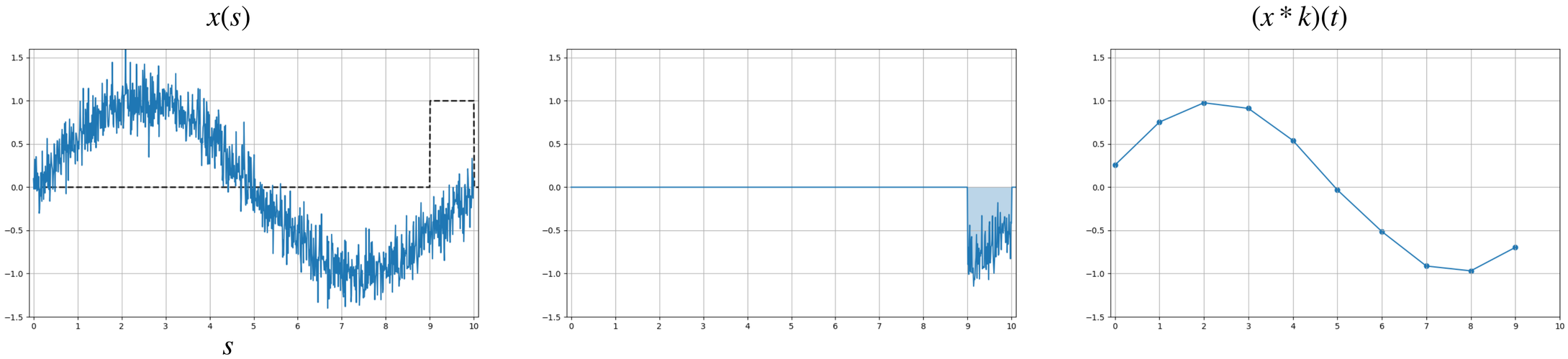


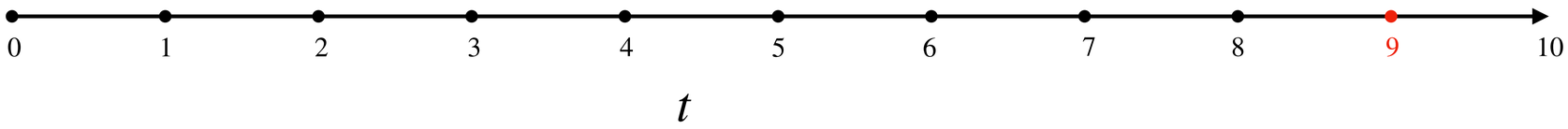


Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$





Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Discrete input x

Discrete 1D convolution

$$(x * k)_i = \sum_n x_n\, k_{i-n}$$

$x = (x_1, \ldots, x_N)$ is a 1D vector

Discrete 2D convolution

$$(x * k)_{i,j} = \sum_n \sum_m x_{n,m}\, k_{i-n,j-m}$$

$$x = \begin{pmatrix} x_{1,1} & \cdots & x_{1,W} \\ & \cdots & \\ x_{H,1} & \cdots & x_{H,W} \end{pmatrix}$$ is a 2D vector/matrix

Convolutions

General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t - s)\, ds$$

Most ML libraries use *cross-correlation*

$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

Convolutions

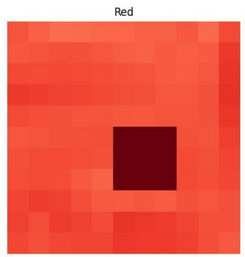
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

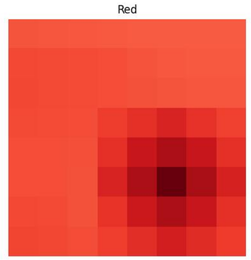
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

$x * k$



	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

Convolutions

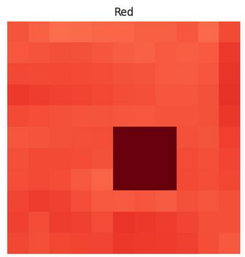
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

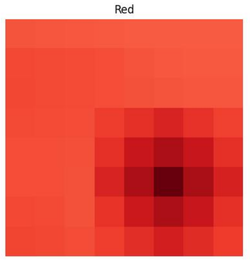
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



$k = ?$

$x * k$



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

Convolutions

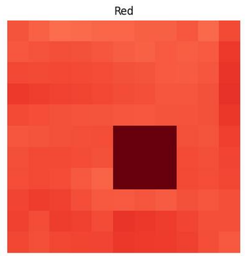
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

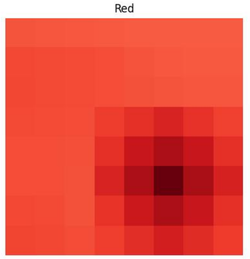
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



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

$x * k$



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	111
106	109	111	112	113	114	115	113	112	111
114	113	111	110	109	0	0	0	111	113
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

*More precisely $k_{i,j} = \begin{cases} \frac{1}{9} & \text{if } i \le 2 \text{ and } j \le 2 \\ 0 & \text{otherwise} \end{cases}$

Convolutions

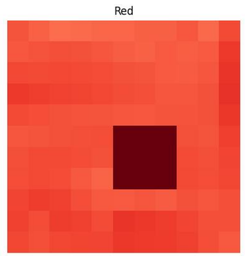
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

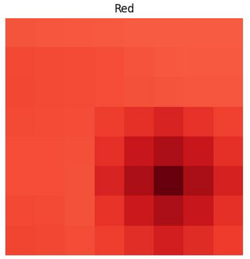
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



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

$x * k$



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114		
106	109	111	112	113	114	113			
114	113	111	110	109	0	0	0		
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

$\times$ $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$
$\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$
$\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$

	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

*More precisely $k_{i,j} = \begin{cases} \frac{1}{9} & \text{if } i \le 2 \text{ and } j \le 2 \\ 0 & \text{otherwise} \end{cases}$

Convolutions

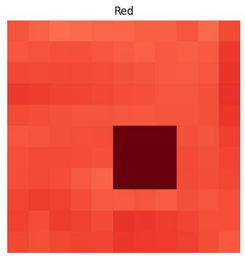
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

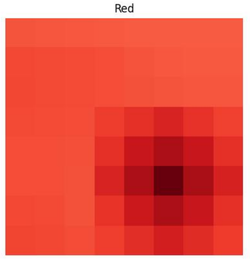
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



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

$x * k$



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114		
106	109	111	112	113	114	113			
114	113	111	110	109	0	0	0		
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

$\times$ $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$ / $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$ / $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$ $=$

$\frac{103}{9}$	$\frac{105}{9}$	$\frac{108}{9}$
$\frac{112}{9}$	$\frac{113}{9}$	$\frac{114}{9}$
$\frac{110}{9}$	$\frac{109}{9}$	$\frac{0}{9}$

	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

*More precisely $k_{i,j} = \begin{cases} \frac{1}{9} & \text{if } i \le 2 \text{ and } j \le 2 \\ 0 & \text{otherwise} \end{cases}$

Convolutions

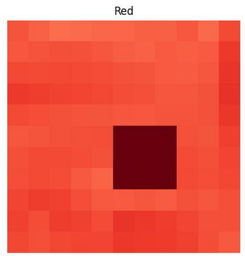
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

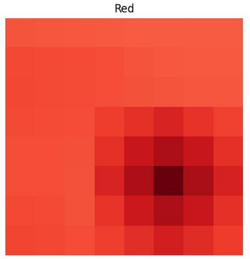
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



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

$x * k$



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114		
106	109	111	112	113	114	111	113		
114	113	111	110	109	0	0	0		
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

$\times \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} = \begin{bmatrix} \frac{103}{9} & \frac{105}{9} & \frac{108}{9} \\ \frac{112}{9} & \frac{113}{9} & \frac{114}{9} \\ \frac{110}{9} & \frac{109}{9} & \frac{0}{9} \end{bmatrix} +$

	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

**More precisely* $k_{i,j} = \begin{cases} \frac{1}{9} & \text{if } i \le 2 \text{ and } j \le 2 \\ 0 & \text{otherwise} \end{cases}$

Convolutions

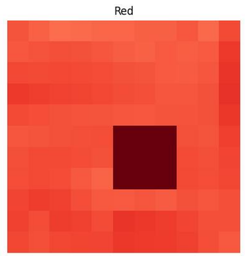
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

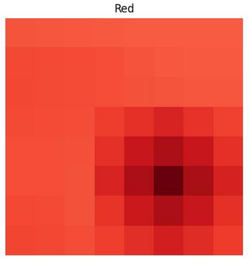
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

x



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

$x * k$



113	122	130	128	126	125	121	121	115	127
115	112	110	111	114	119	122	117	120	117
105	106	105	106	108	111	113	117	118	114
94	98	101	103	105	108	110	114	114	112
106	109	111	112	113	114	115	111	112	110
114	113	111	110	109	0	0	0	111	110
110	106	106	108	111	0	0	0	107	110
109	106	108	116	124	0	0	0	106	109
102	97	100	109	116	117	114	118	111	114
100	108	97	100	107	90	93	97	102	110

$\times$ $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$ / $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$ / $\frac{1}{9}$ $\frac{1}{9}$ $\frac{1}{9}$ $=$

$+$

	113	114	115	116	118	118	118	118	
	105	106	107	109	112	115	116	116	
	104	106	107	109	111	113	114	114	
	106	108	108	97	86	75	88	100	
	110	110	110	86	62	38	62	86	
	109	109	111	75	38	0	36	73	
	105	106	111	89	65	39	62	86	
	103	105	109	98	85	70	82	96	

*More precisely $k_{i,j} = \begin{cases} \frac{1}{9} & \text{if } i \le 2 \text{ and } j \le 2 \\ 0 & \text{otherwise} \end{cases}$

Convolutions

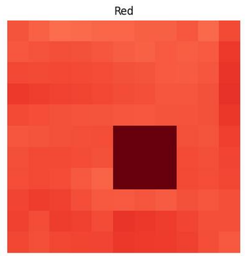
General definition of convolution between an input signal x and the kernel k

$$(x * k)(t) = \int x(s)\, k(t-s)\, ds$$

Most ML libraries use *cross-correlation*

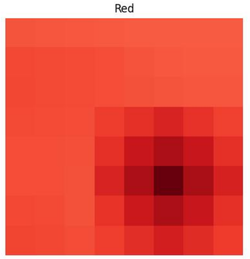
$$(x * k)_{i,j} = \sum_n \sum_m x_{i+n,j+m}\, k_{n,m}$$

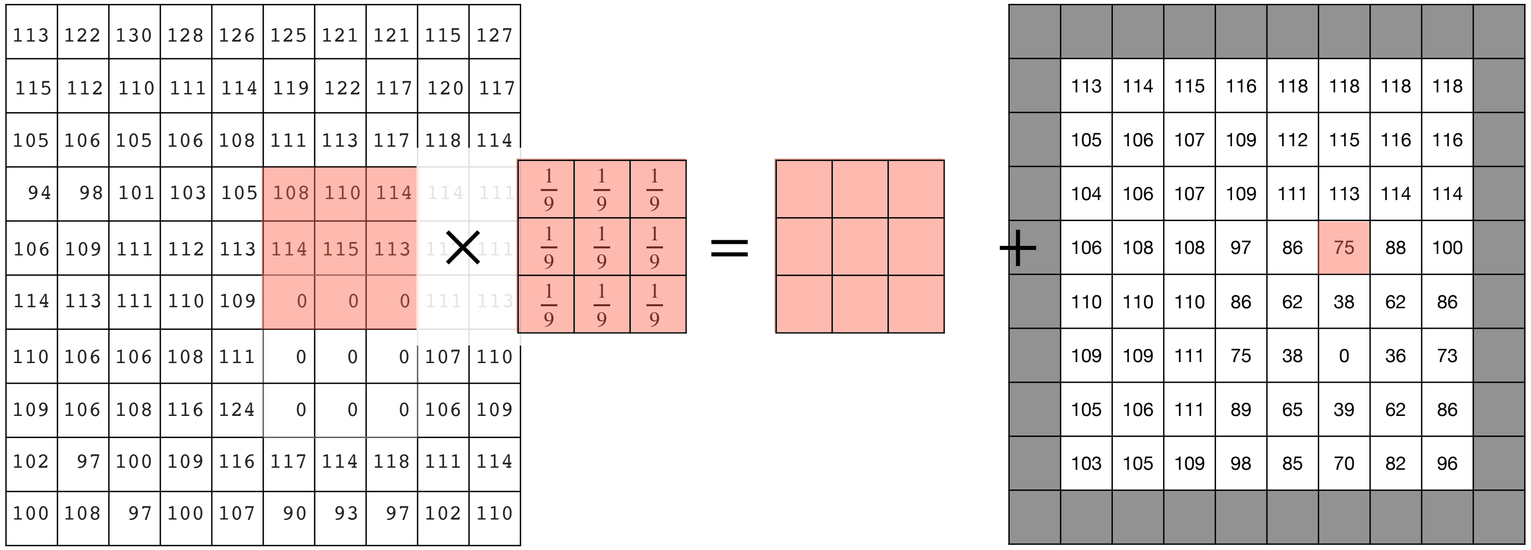
x



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

$x * k$





Computing cross-correlation/convolution:

1. Element-wise multiplication of kernel weights and pixels
2. Sum all the elements
3. Slide the kernel

**More precisely* $k_{i,j} = \begin{cases} \frac{1}{9} & \text{if } i \le 2 \text{ and } j \le 2 \\ 0 & \text{otherwise} \end{cases}$

Convolutional layer

Explore in practice the effects of different components of convolutional layers

1. Go to Brightspace → 5. Convolutional Networks → Convolutional Layer Practice
2. Follow instructions in the notebook
3. 10 mins
4. Discuss together

Note: it's not a programming exercise.

Convolutional layer: input


```
Shape of cat (1068, 1600, 3)
Maximum value: 255
Minimum value: 0
```

```
Shape of logo (166, 166)
Maximum value: 255
Minimum value: 0
```

Convolutional layer: input


```
Shape of cat (1068, 1600, 3)
Maximum value: 255
Minimum value: 0
```

Size of the image ⟶ (height, width, rgb channels)

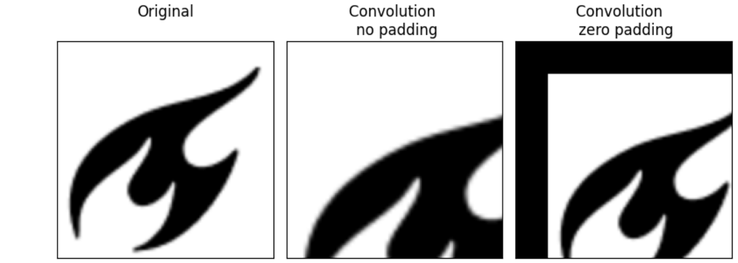
```
Shape of logo (166, 166)
Maximum value: 255
Minimum value: 0
```

Channel range [0, 255] integers

Grey scale black to white 0 to 255

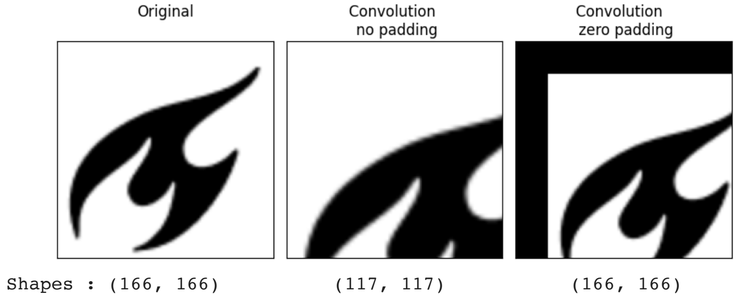
Convolutional layer: padding

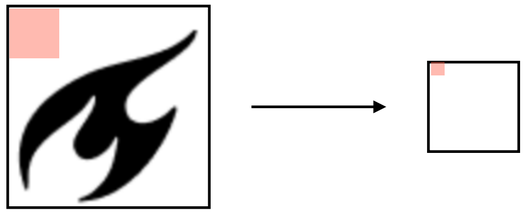
- **Padding**



Convolutional layer: padding

- **Padding**



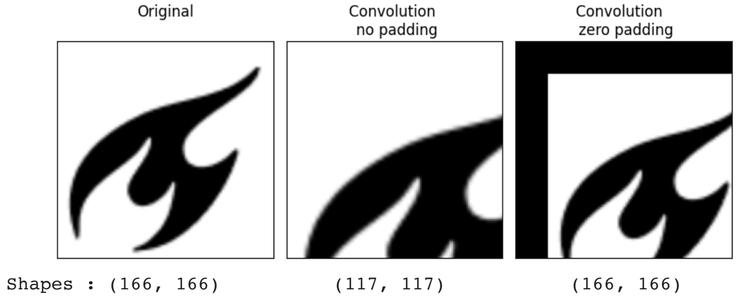


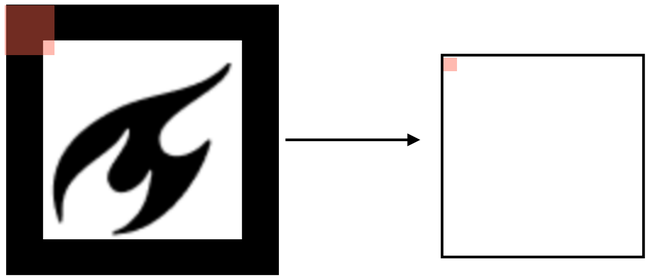
Convolution has a shrink effect*

How much exactly? What is the shape of the output?

Convolutional layer: padding

- **Padding**

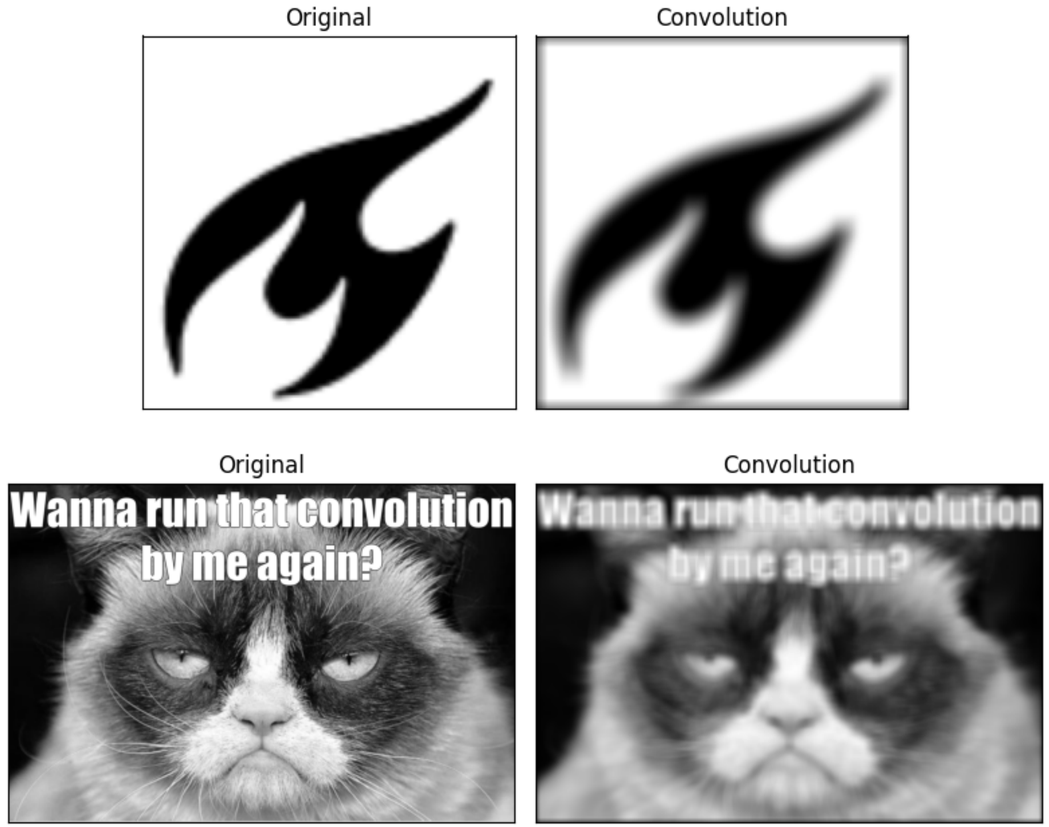




Zero-padding → input and output have same shape

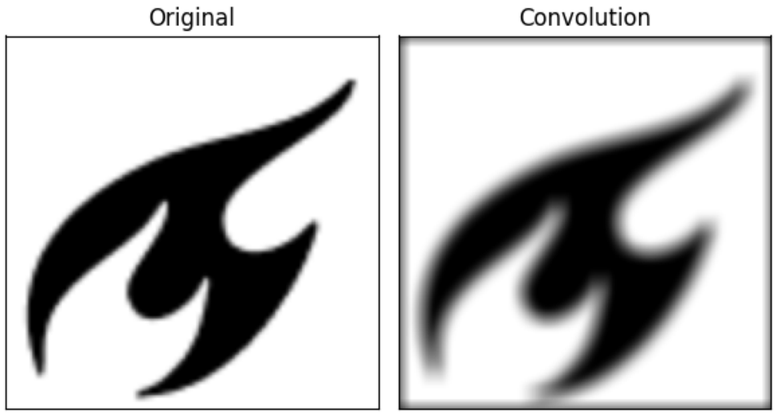
Convolutional layer: kernel effects

- Padding
- **Moving Average Filter**

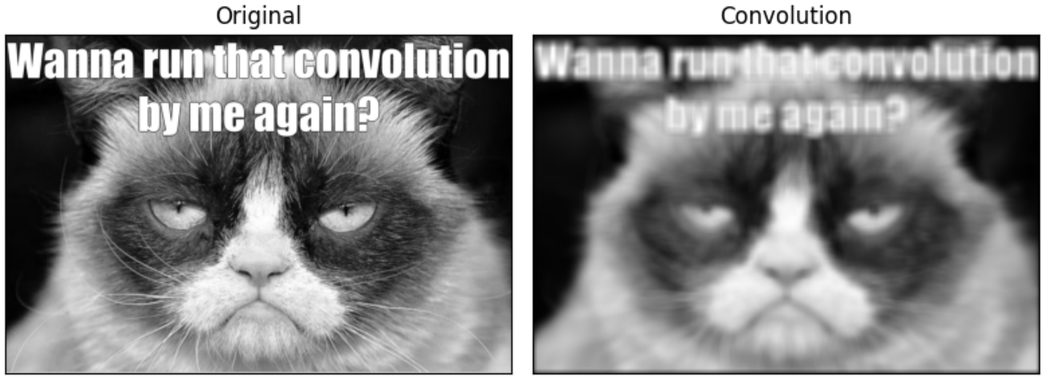


Convolutional layer: kernel effects

- Padding
- **Moving Average Filter**



size = 10

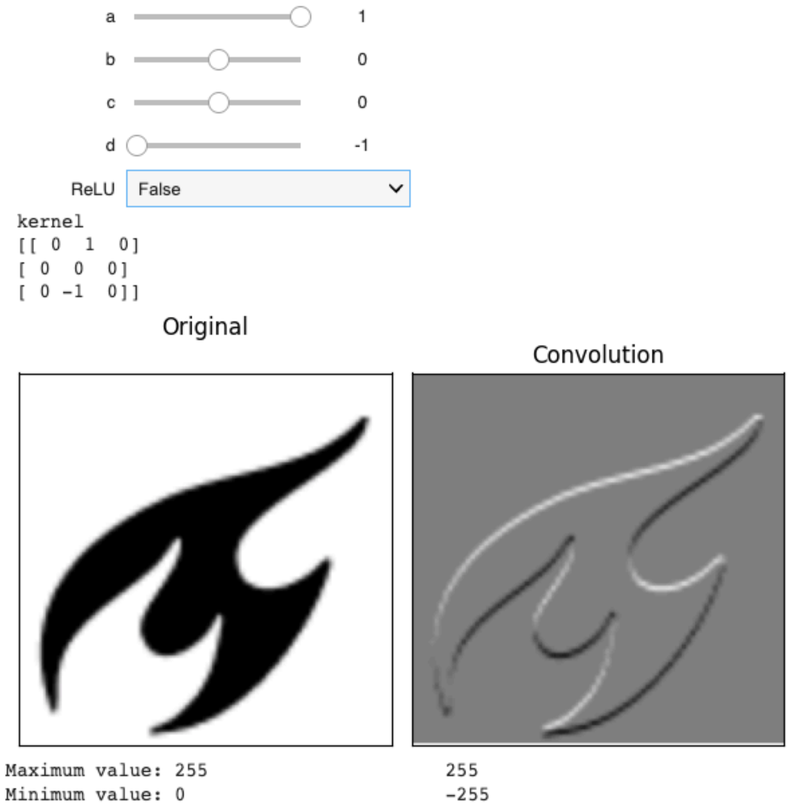


size = 30

Blur Filter

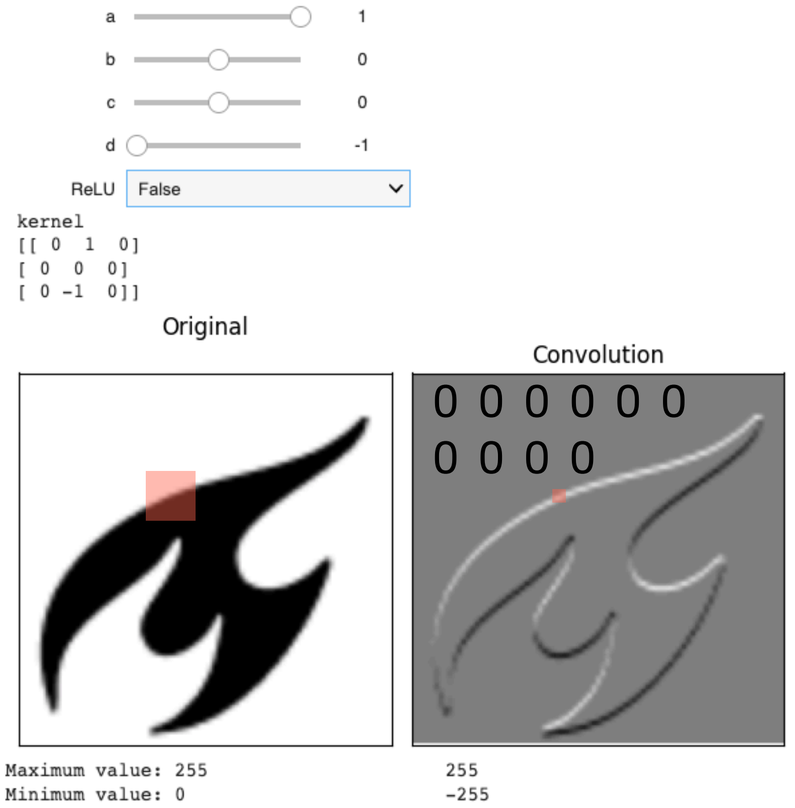
Convolutional layer: kernel effects

- Padding
- Moving Average Filter
- **Detector kernel**



Convolutional layer: kernel effects

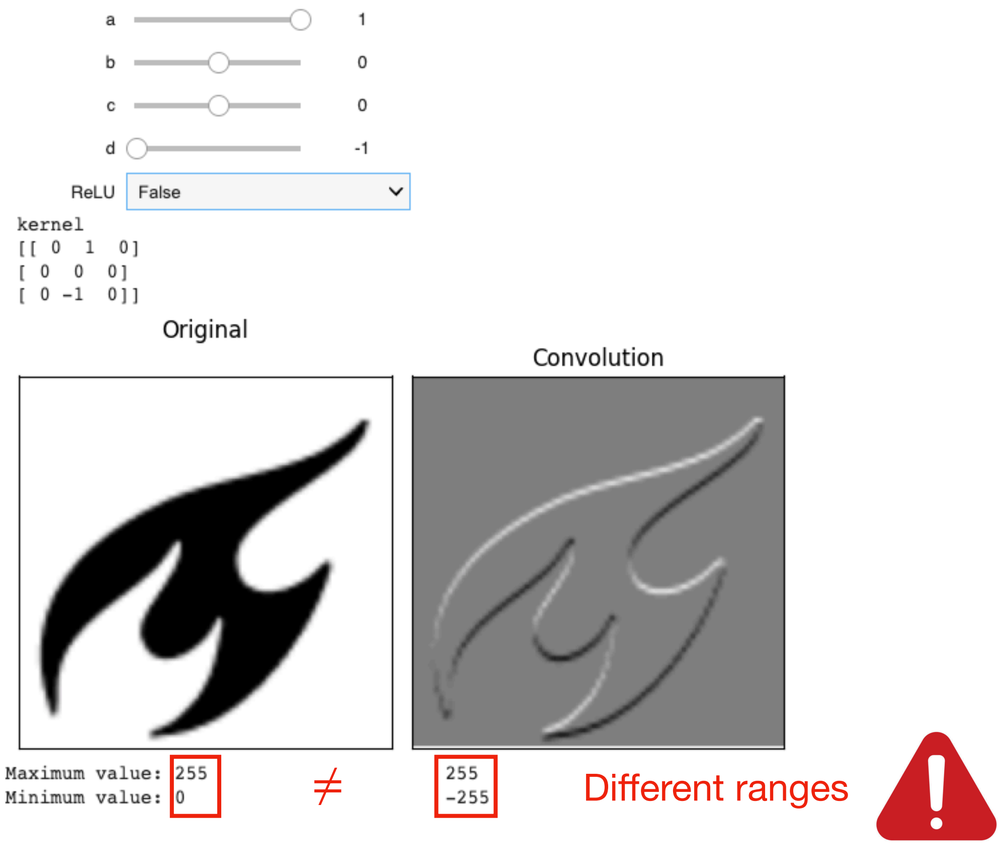
- Padding
- Moving Average Filter
- **Detector kernel**



horizontal edges detector

Convolutional layer: kernel effects

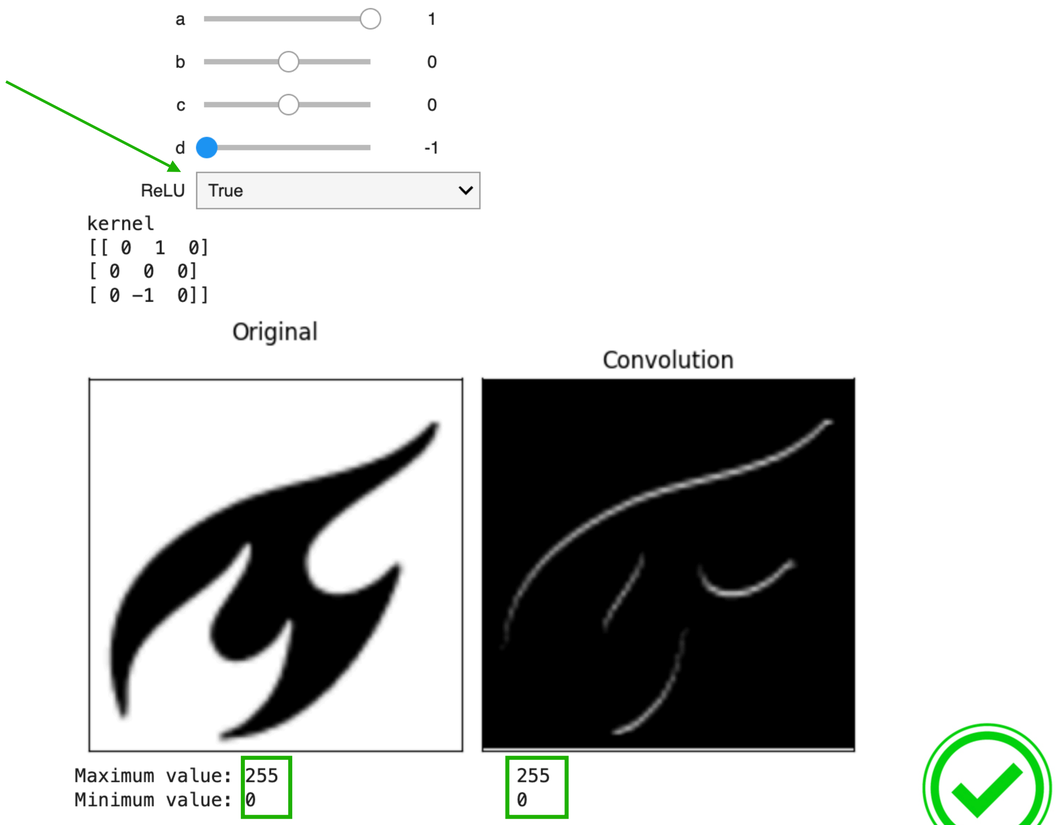
- Padding
- Moving Average Filter
- **Detector kernel**



horizontal edges detector

Convolutional layer: activation

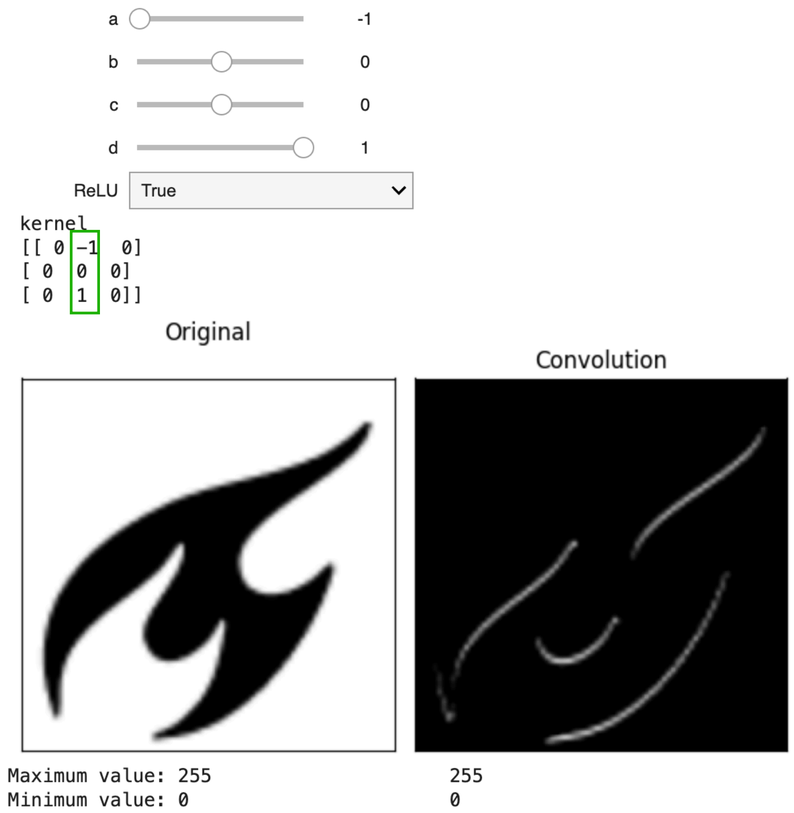
- Padding
- Moving Average Filter
- Detector kernel
- **ReLU activation**



horizontal edges detector

Convolutional layer: activation

- Padding
- Moving Average Filter
- Detector kernel
- **ReLU activation**



horizontal edges detector

Convolutional layer: channels

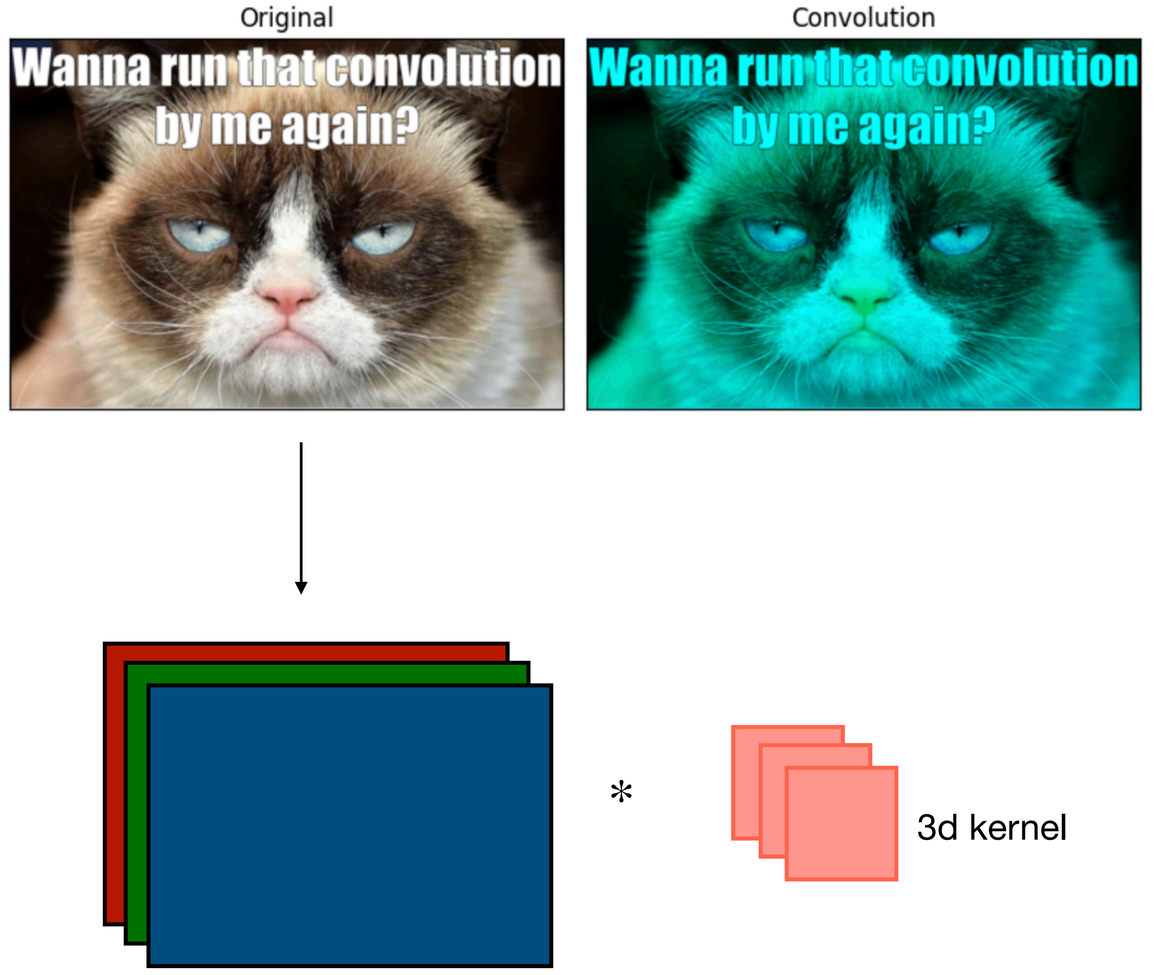
- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- **Input channels**

Original

Convolution

Convolutional layer: channels

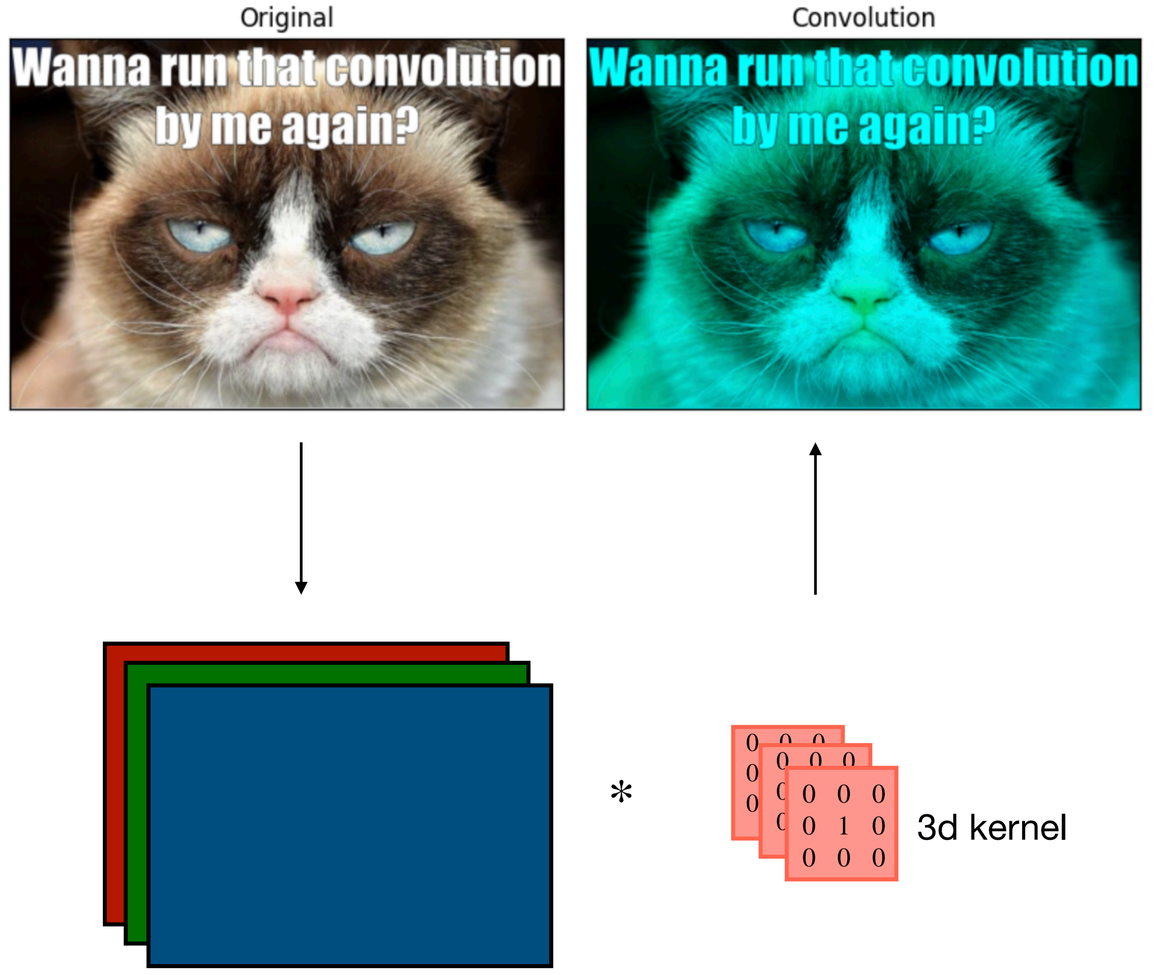
- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- **Input channels**



3d convolution

Convolutional layer: channels

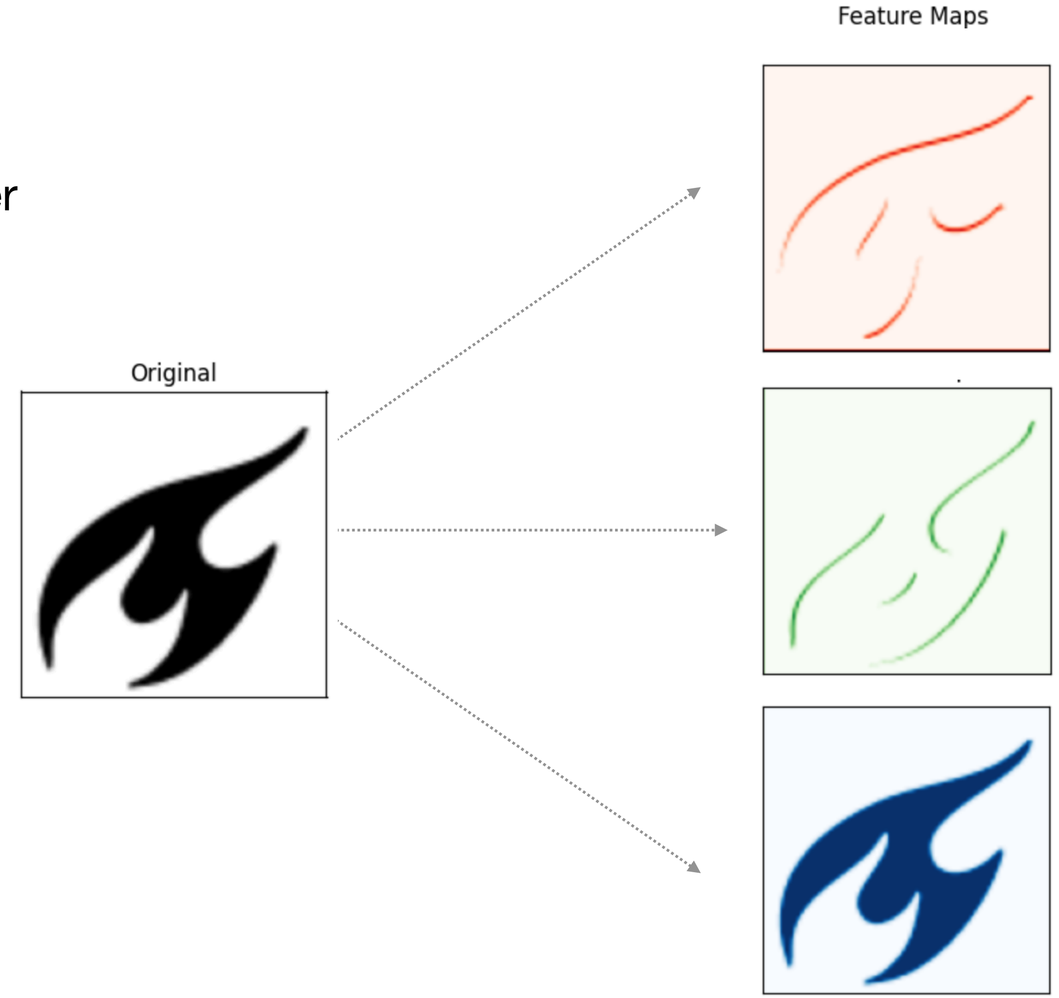
- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- **Input channels**



3d convolution

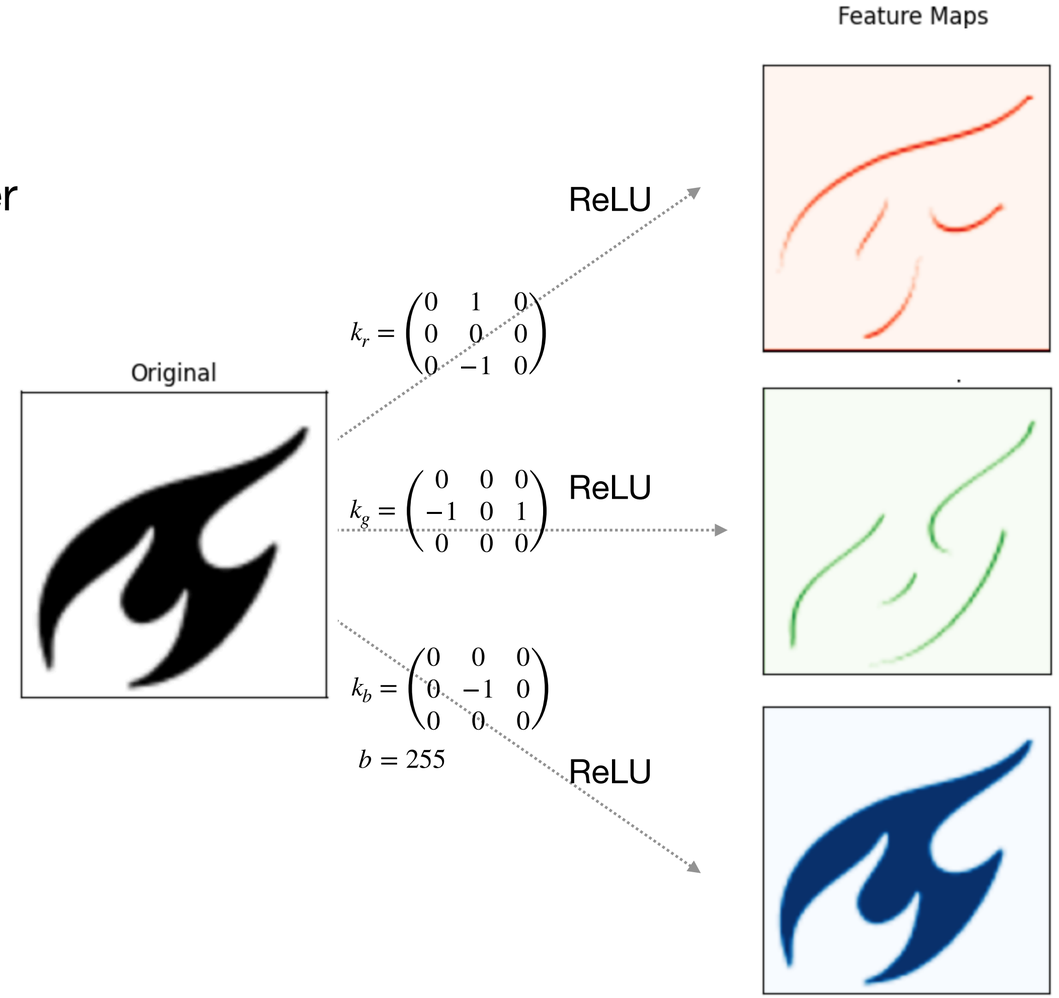
Convolutional layer: feature maps

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- **Feature Maps**



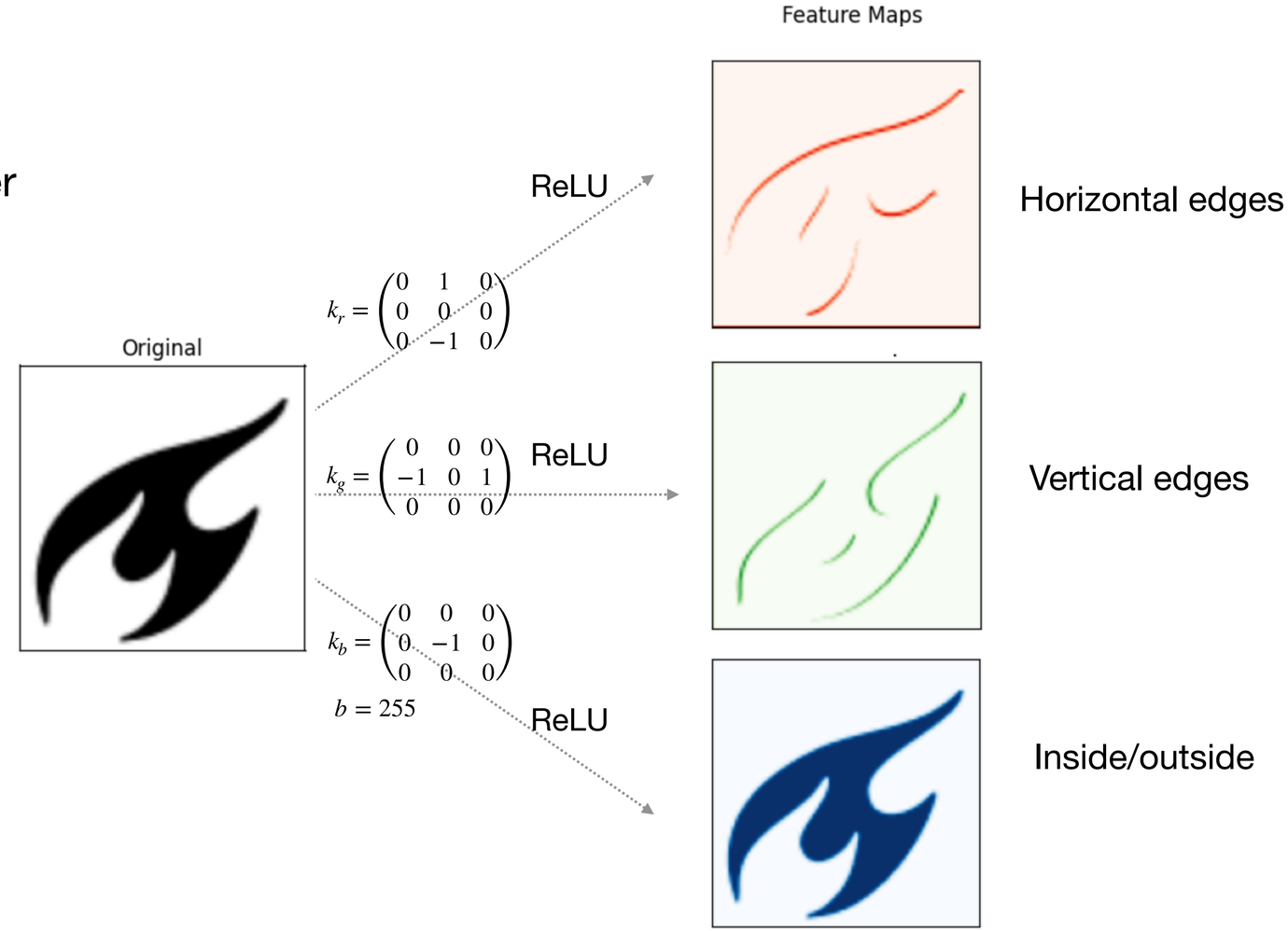
Convolutional layer: feature maps

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- **Feature Maps**



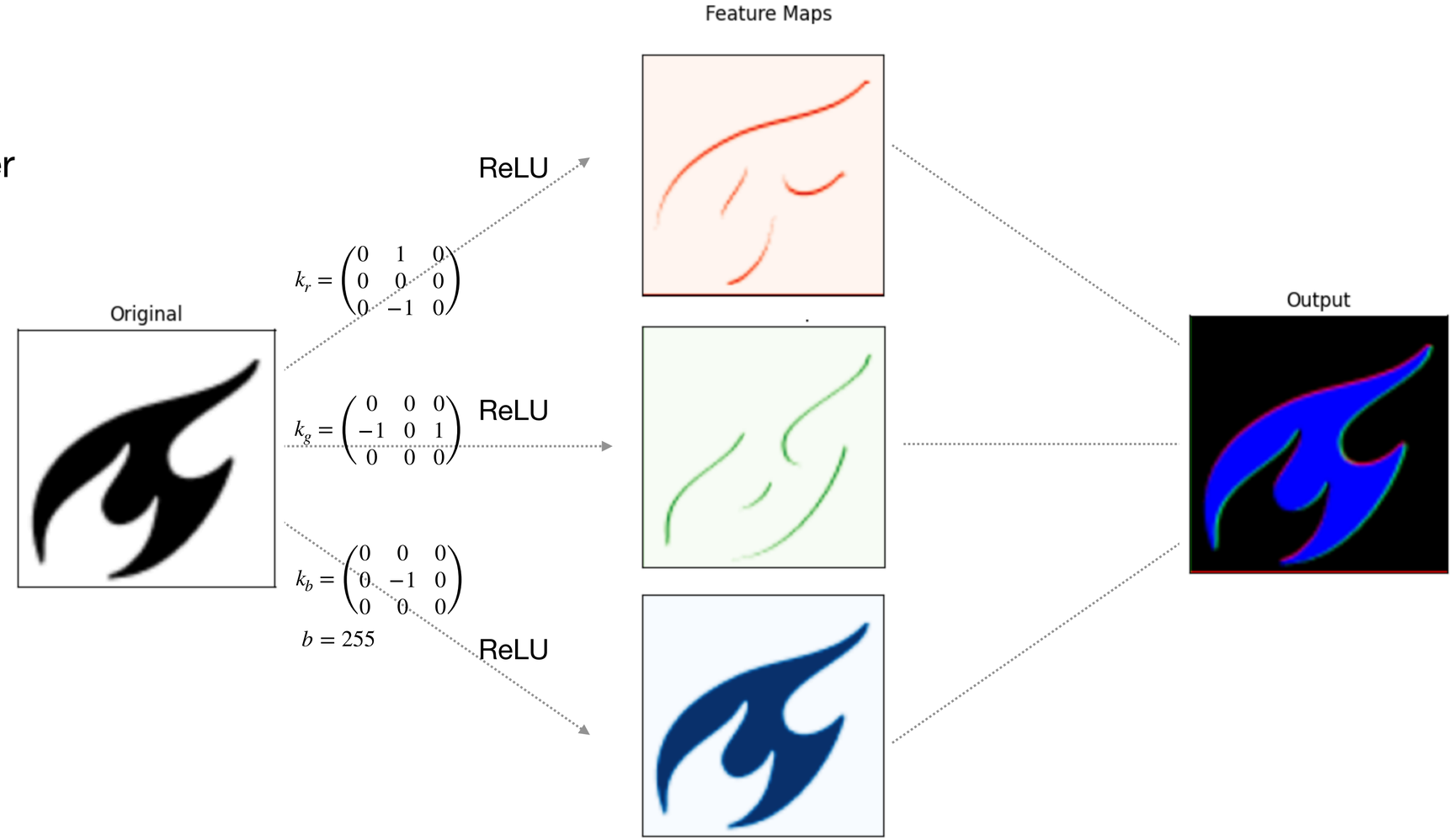
Convolutional layer: feature maps

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- **Feature Maps**



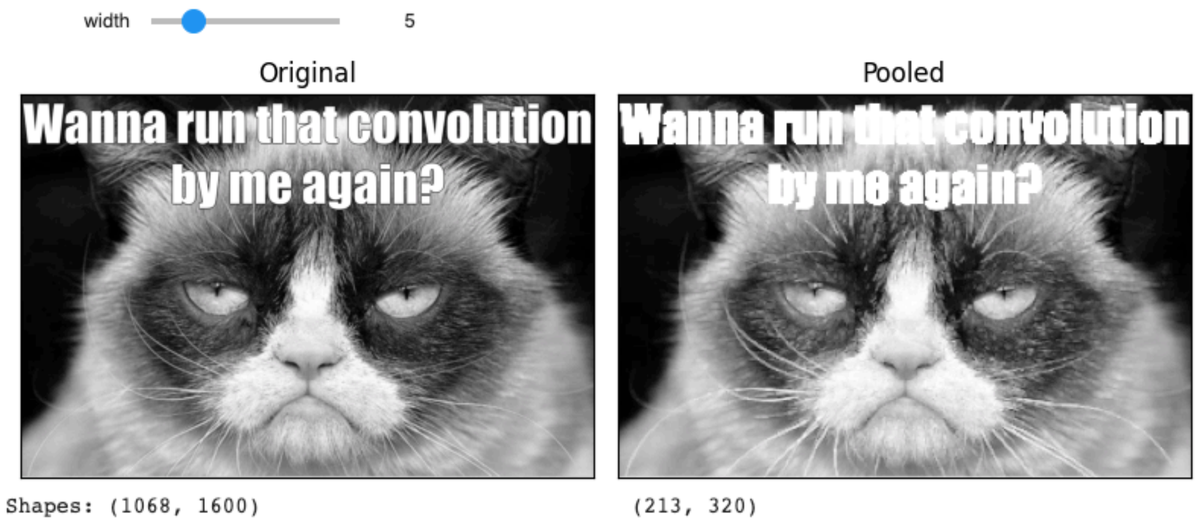
Convolutional layer: feature maps

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- **Feature Maps**



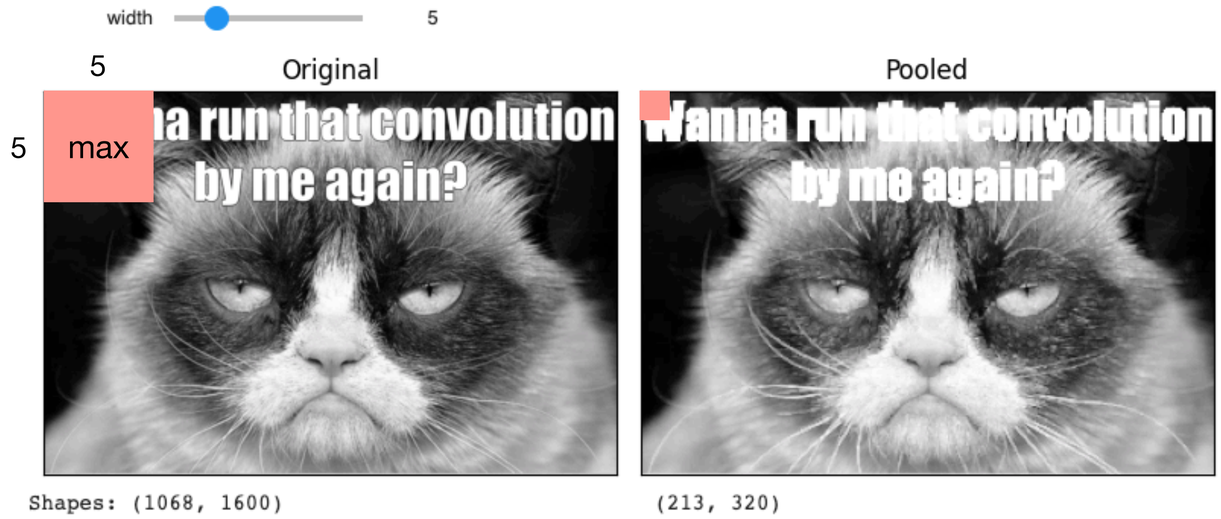
Convolutional layer: pooling

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- Feature Maps
- **Pooling**



Convolutional layer: pooling

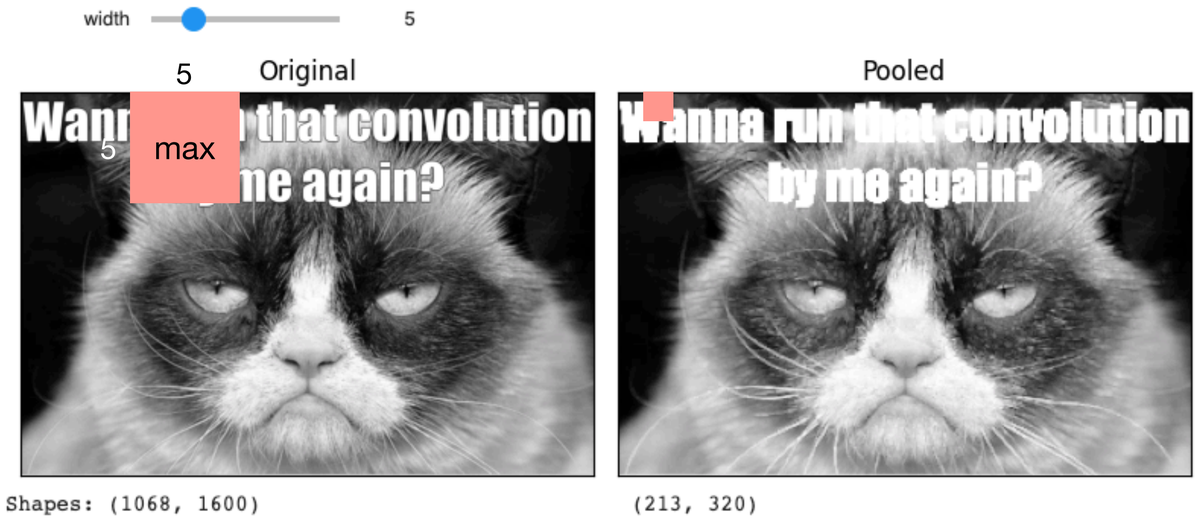
- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- Feature Maps
- **Pooling**



Max Pooling preserves the features of the cat

Convolutional layer: pooling

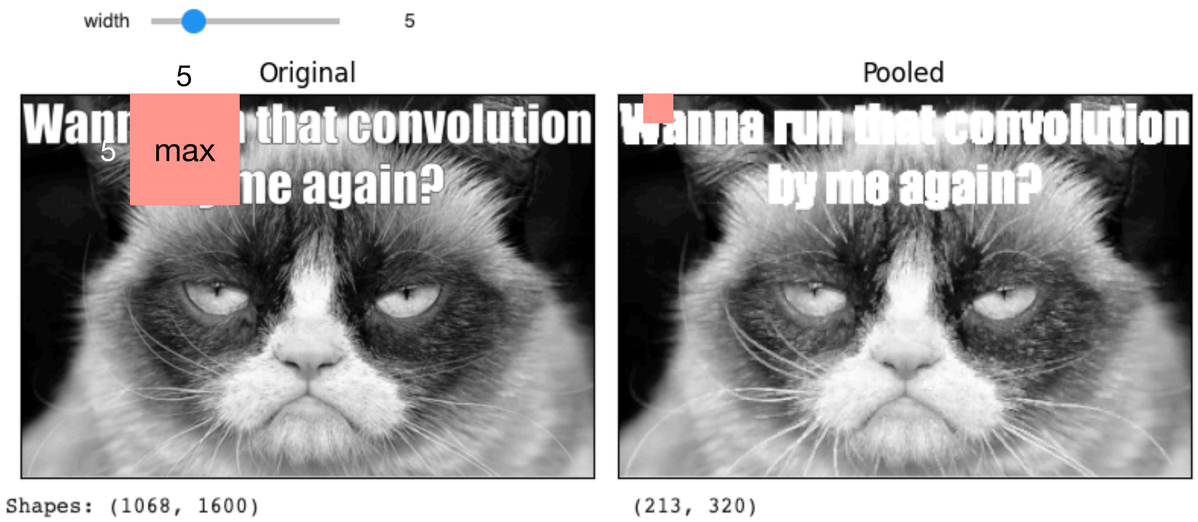
- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- Feature Maps
- **Pooling**



Max Pooling preserves the features of the cat

Convolutional layer: pooling

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- Feature Maps
- **Pooling**

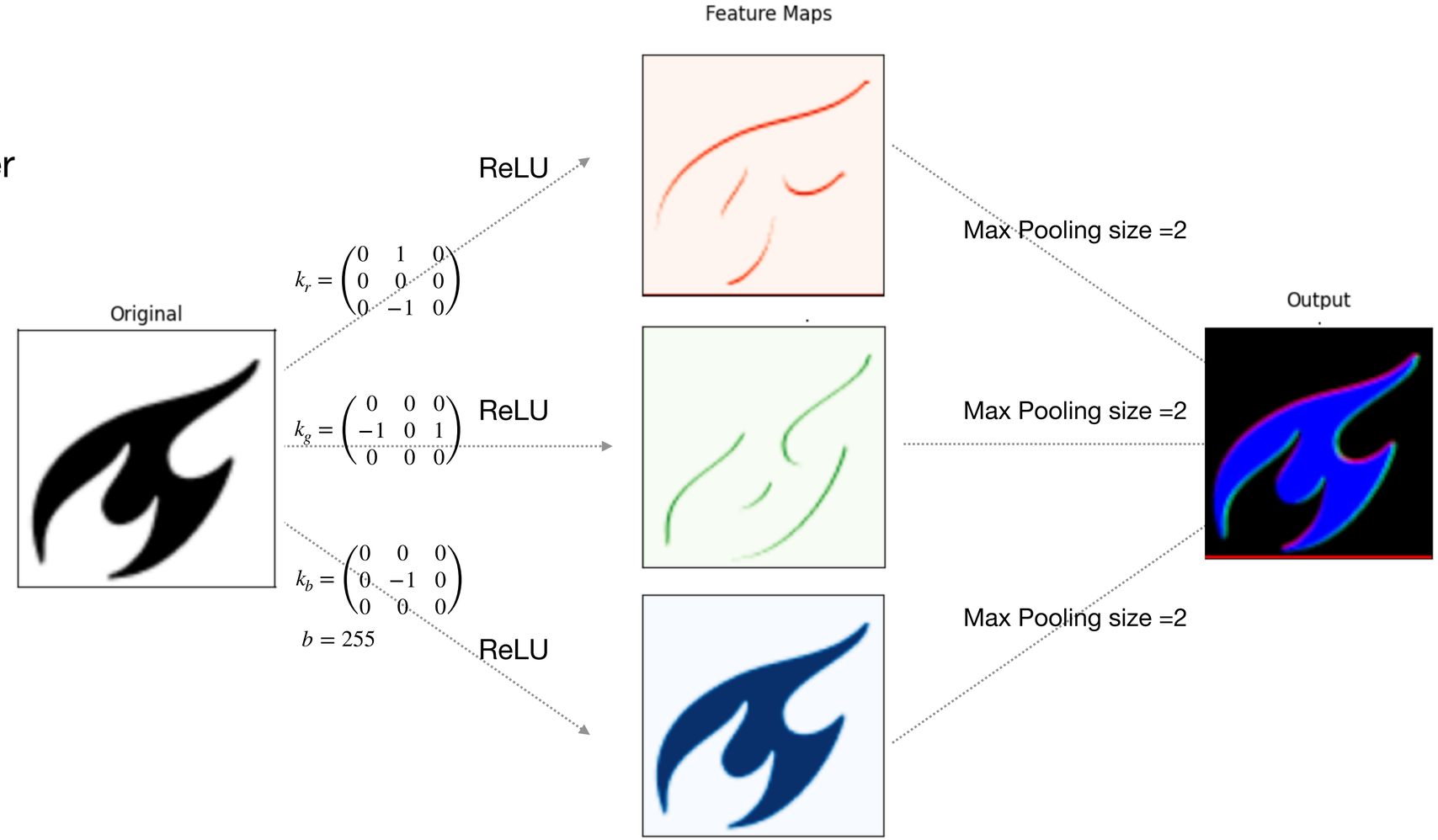


Max Pooling preserves the features of the cat

- Representation invariance to small translations
- Reduce memory

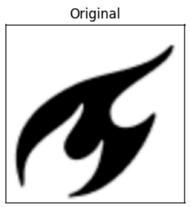
Convolutional layer

- Padding
- Moving Average Filter
- Detector kernel
- ReLU activation
- Input channels
- Feature Maps
- Pooling



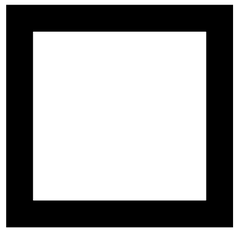
Convolutional Layer

Input



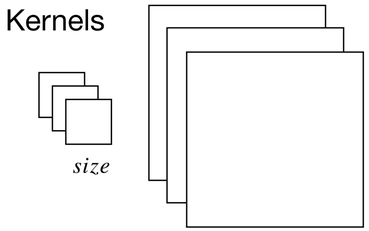
Learnable parameters

Padding



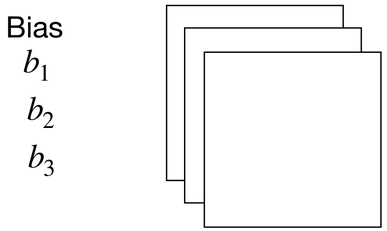
0

Convolution



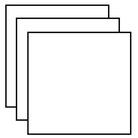
size x size x #feature maps

Bias and activation (pixel-wise)



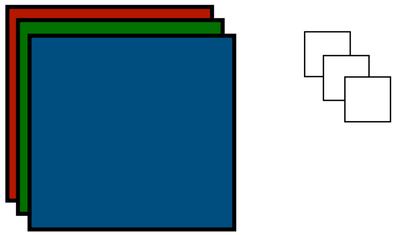
#feature maps (bias)

Pooling layer



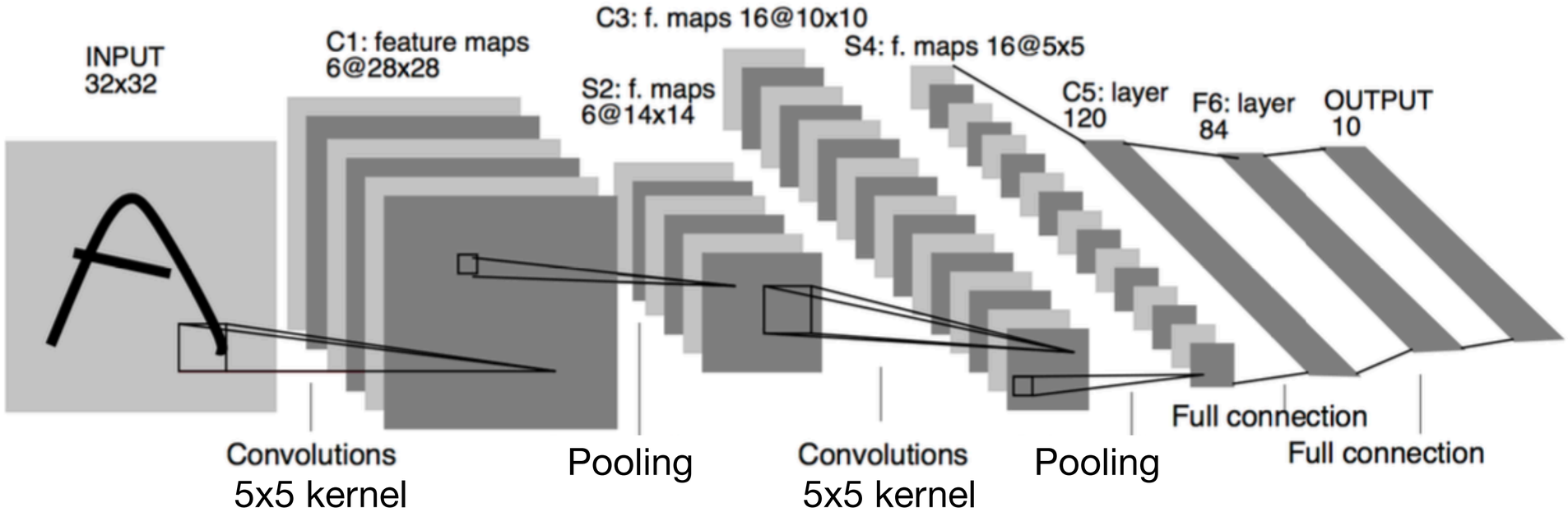
0

Output



Typically **different** kernels for every channel

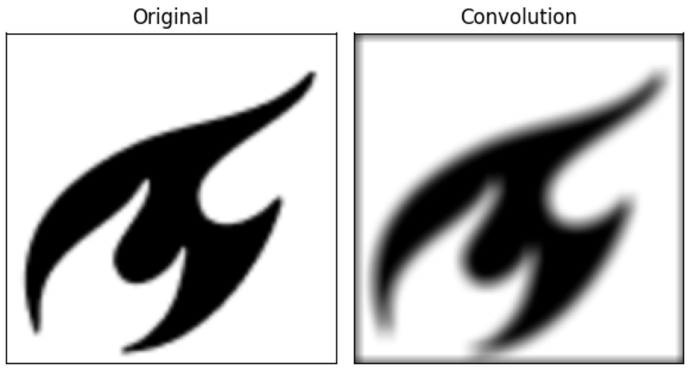
CNN Architecture



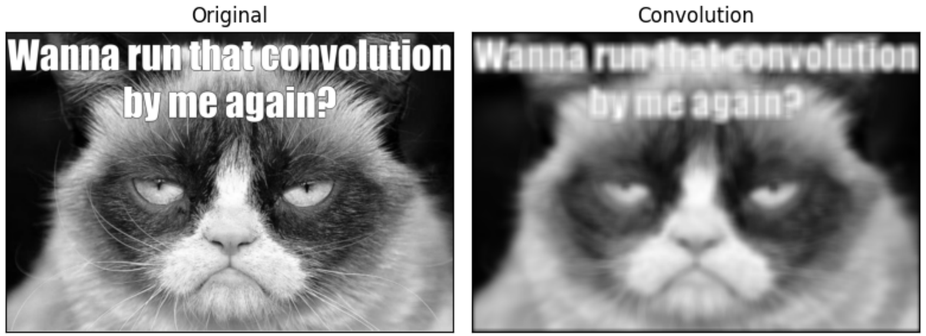
LeNet - 5 (1998)

Stride and Dilation

Bigger images require larger kernel size ⟶ more parameters to learn



kernel size = 10



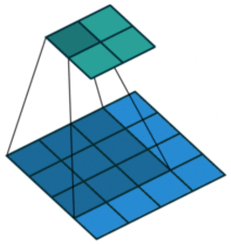
kernel size = 30

**Animation from* Dumoulin, Vincent, and Francesco Visin. "*A guide to convolution arithmetic for deep learning.*" *(2016)* https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Stride and Dilation

Bigger images require larger kernel size ⟶ more parameters to learn

Output



Original

kernel size = 3 (3x3)

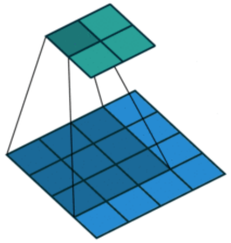
stride = 1

**Animation from* Dumoulin, Vincent, and Francesco Visin. "*A guide to convolution arithmetic for deep learning.*" *(2016)* https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Stride and Dilation

Bigger images require larger kernel size ⟶ more parameters to learn

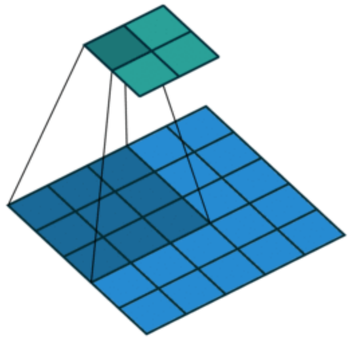
Output



Original

kernel size = 3

stride = 1



kernel size = 3

stride = 2

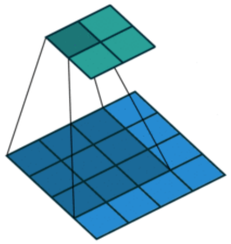
Remember, we have already seen stride for pooling!

**Animation from* Dumoulin, Vincent, and Francesco Visin. "*A guide to convolution arithmetic for deep learning.*" *(2016)* https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Stride and Dilation

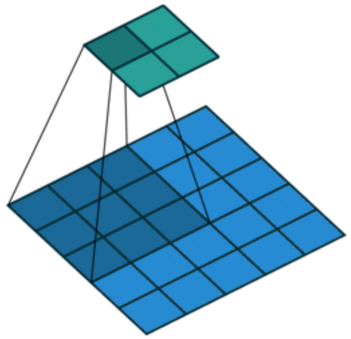
Bigger images require larger kernel size ⟶ more parameters to learn

Output

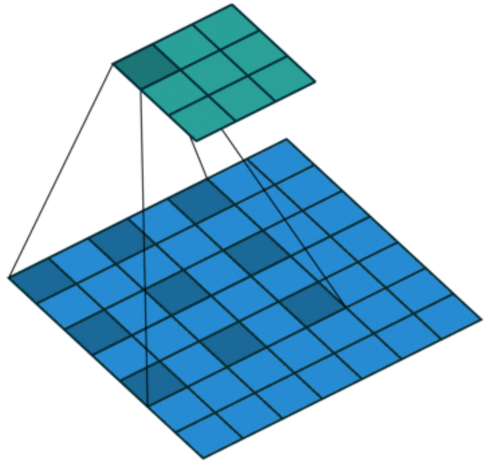


Original

kernel size = 3
stride = 1
dilation = 1



kernel size = 3
stride = 2
dilation = 1

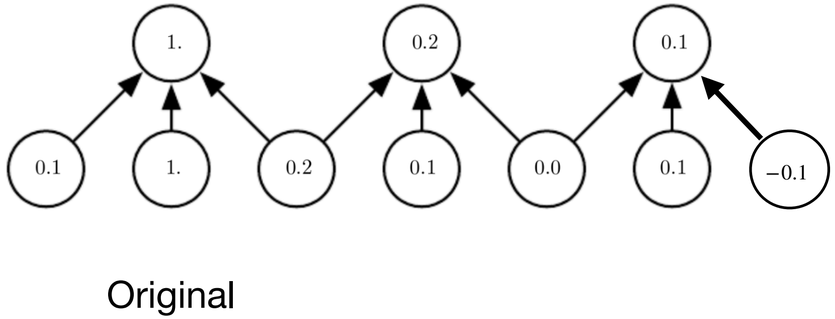


kernel size = 3
stride = 1
dilation = 2

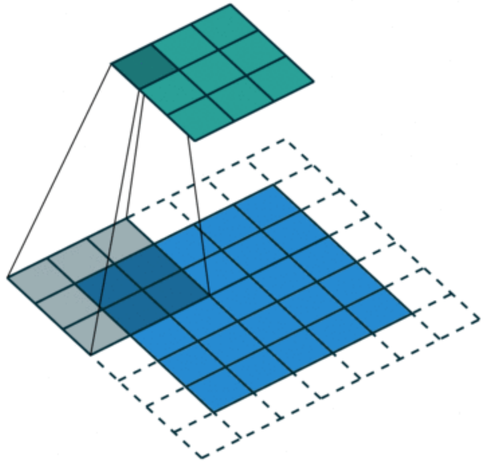
**Animation from* Dumoulin, Vincent, and Francesco Visin. "*A guide to convolution arithmetic for deep learning.*" *(2016)* https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Stride and Dilation

Exercise



operation = ?
size =?
stride = ?
dilation = ?
padding = ?

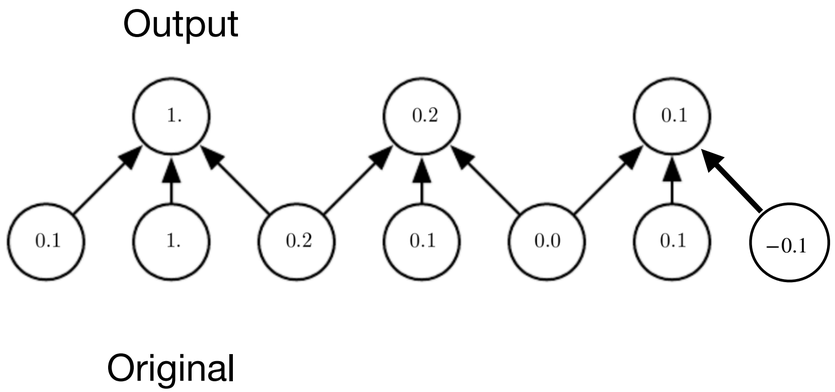


operation = ?
size=?
stride = ?
dilation = ?
padding = ?

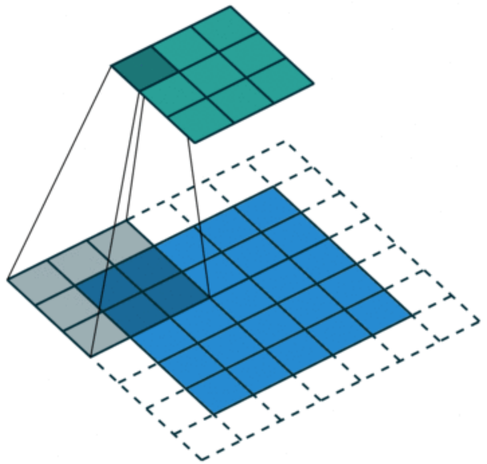
**Animation from* Dumoulin, Vincent, and Francesco Visin. "*A guide to convolution arithmetic for deep learning.*" *(2016)* https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Stride and Dilation

Solution



max pooling
size = 3
stride = 2
dilation = 1
padding = no



pooling or convolution
size = 3
stride = 2
dilation = 1
padding = yes

**Animation from* Dumoulin, Vincent, and Francesco Visin. "*A guide to convolution arithmetic for deep learning.*" *(2016)* https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

Convolutional vs fully connected layers

Images properties:

1. High dimensionality
2. Spacial structure
3. Stability under transformations

Convolutional vs fully connected layers

Images properties:

1. High dimensionality
2. Spacial structure
3. Stability under transformations

Convolution is a linear operation represented by a sparse matrix with shared parameters and a local structure

$$x * (-1,0,1) = x \cdot \begin{pmatrix} -1 & 0 & 0 & 0 & \\ 0 & -1 & 0 & 0 & \\ 1 & 0 & -1 & 0 & \dots \\ 0 & 1 & 0 & -1 & \\ 0 & 0 & 1 & 0 & \\ \dots & & & & \end{pmatrix}^{*}$$

*Convince yourself that this is true and the statement holds for any kernel.

Convolutional vs fully connected layers

Images properties:

1. High dimensionality
2. Spacial structure
3. Stability under transformations

Convolution is a linear operation represented by a sparse matrix with shared parameters and a local structure

$$x * (-1,0,1) = x \cdot \begin{pmatrix} -1 & 0 & 0 & 0 & \\ 0 & -1 & 0 & 0 & \\ 1 & 0 & -1 & 0 & \dots \\ 0 & 1 & 0 & -1 & \\ 0 & 0 & 1 & 0 & \\ \dots & & & & \end{pmatrix}$$

#convolution parameters = 3

#fully connected parameters = $H \times W$

Convolutional vs fully connected layers

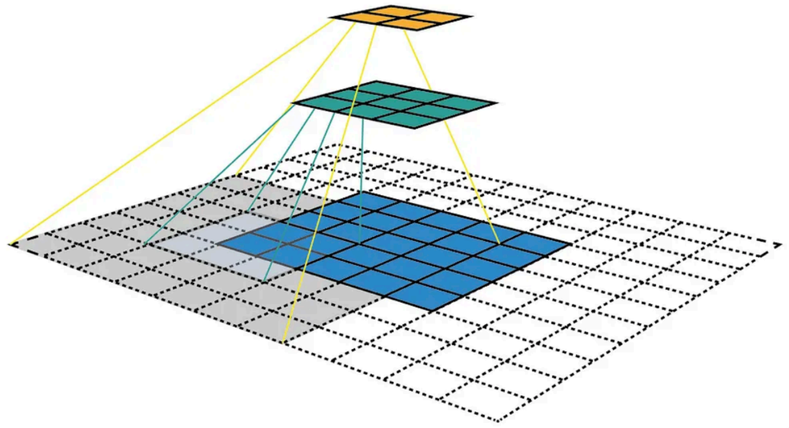
Images properties:

1. High dimensionality
2. Spacial structure
3. Stability under transformations

Convolution is a linear operation represented by a sparse matrix with shared parameters and a local structure

$$x * (-1,0,1) = x \cdot \begin{pmatrix} -1 & 0 & 0 & 0 & \\ 0 & -1 & 0 & 0 & \\ 1 & 0 & -1 & 0 & \dots \\ 0 & 1 & 0 & -1 & \\ 0 & 0 & 1 & 0 & \\ \dots & & & & \end{pmatrix}$$

Receptive field, the region of the input which affects a specific output unit is **local.**



Convolutional vs fully connected layers

Images properties:

1. High dimensionality
2. Spacial structure
3. Stability under transformations

Convolution is a linear operation represented by a sparse matrix with shared parameters and a local structure

$$x * (-1,0,1) = x \cdot \begin{pmatrix} -1 & 0 & 0 & 0 & \\ 0 & -1 & 0 & 0 & \\ 1 & 0 & -1 & 0 & \dots \\ 0 & 1 & 0 & -1 & \\ 0 & 0 & 1 & 0 & \\ \dots & & & & \end{pmatrix}$$

Convolution is a translation equivariant operation* and pooling enhances translation invariance

*Convince yourself that this is true

Exercise

Multi-armed bandit

Possible reward y: 0, +1 , +100

Bandit x: 0, 1

$$x^{(1)}, x^{(2)}, \ldots x^{(10)} = 0, 1, 1, 1, 0, 1, 0, 1, 0, 0$$
$$y^{(1)}, y^{(2)}, \ldots y^{(10)} = 1, 0, 0, 0, 1, 0, 0, 100, 0, 0$$

Neural network setup

1. What is the learning task? What is the input and output?
2. How many input, hidden and output units?
3. Which is the last layer activation function?
4. What is the loss function?

Multiclass Classification

Training set $\{x^{(i)}, y^{(i)}\}_{i=1,\ldots,n}$ of i.i.d input/output pairs, $y^{(i)}$ is a class $\{1,\ldots,K\}$

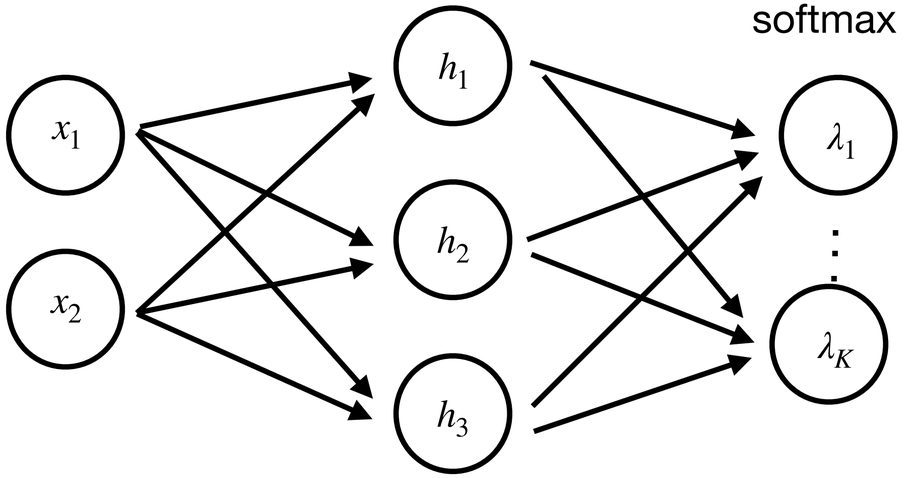
- One-hot vectors $y = (0,\ldots,0,1,0,\ldots,0)$
- K output features
- λ_k is the probability assigned to class k
- Softmax last layer:

$$\lambda_k = \text{softmax}(z_1, \ldots, z_K) = \frac{e^{z_k}}{\sum_{k'} e^{z'_k}}$$

- Cross entropy loss

$$L_{CE}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\sum_{k=1}^{K} - y_k^{(i)} \log(\lambda_k^{(i)})}_{l^{(i)}}$$

- The predicted class predicted $\hat{y} = argmax_k\{\lambda_k\}$



Summary

Topics

- Structured Data
- Convolutions
- Convolutional modules: kernels, padding, feature maps, pooling, stride, dilatation.
- Convolutional vs fully connected networks

Reading material

- *Deep Learning book* - Chapter 9 (until 9.4 excluded)
- *Understanding Deep Learning* - Chapter 10 (10.4 excluded)

Additional resources

- *Neural Networks and Deep Learning*, Michael Nielsen - Chapter 6 Introducting Convolutional Neural Networks (up to "Convolutional neural network in practice" excluded)
- Convolutional layer notebook