Convolutional Neural Networks

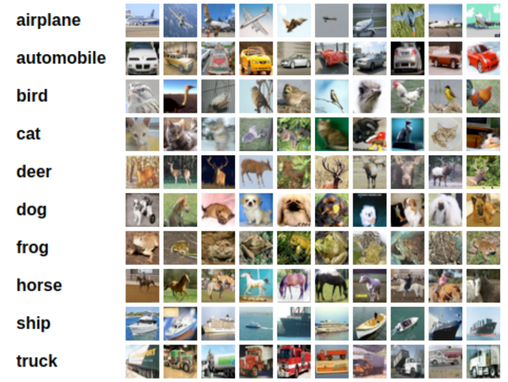
Theo Heimel
May 2025

CP3, UCLouvain

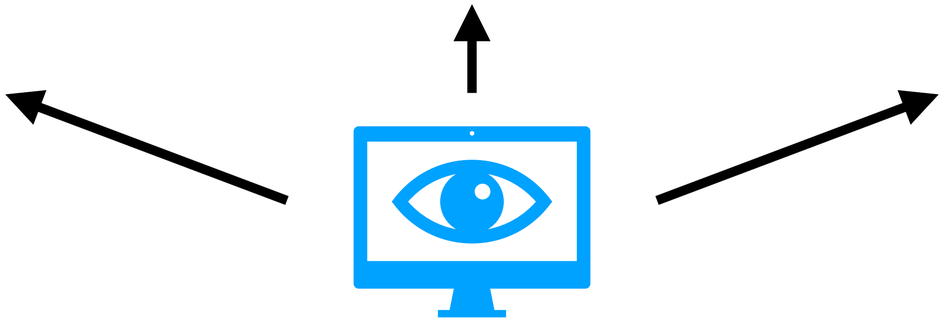
Introduction

Tagging

Segmentation



Classification



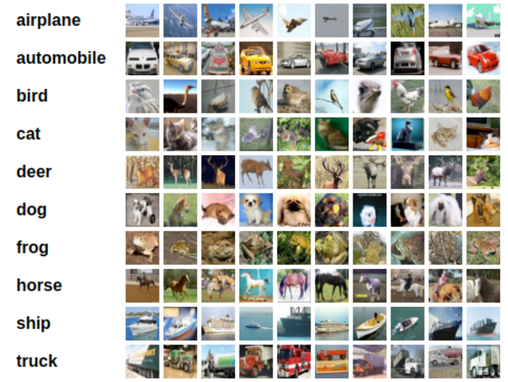
Computer vision tasks appear in many areas
of science, engineering and technology:
self-driving cars, medicine, astronomy, and many more...

Images from https://indiaai.gov.in/article/image-segmentation-the-deep-learning-approach, https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html, https://glassboxmedicine.com/2020/01/21/segmentation-u-net-mask-r-cnn-and-medical-applications/

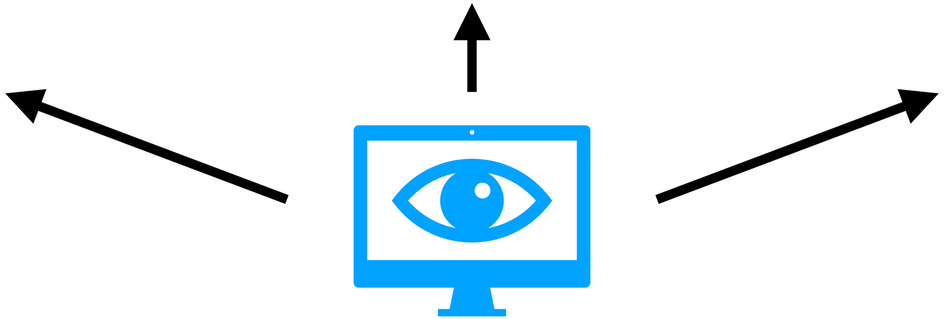
Introduction

Tagging

Segmentation



Classification



Underlying most neural networks for these tasks:

convolutional neural networks

Images from https://indiaai.gov.in/article/image-segmentation-the-deep-learning-approach, https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html, https://glassboxmedicine.com/2020/01/21/segmentation-u-net-mask-r-cnn-and-medical-applications/

Outline

① **Challenges of image processing**

② **Introduction to convolutions**

③ **Convolutional layers**

④ **CNN building blocks**

⑤ **A short history of CNNs**

⑥ **Outlook**

Slides based on previous CNN lectures at ERuM schools, especially those by Sascha Diefenbacher.

More lectures on CNNs:
ErUM DataHub Material Collection

Outline

① **Challenges of image processing**

② Introduction to convolutions

③ Convolutional layers

④ CNN building blocks

⑤ A short history of CNNs

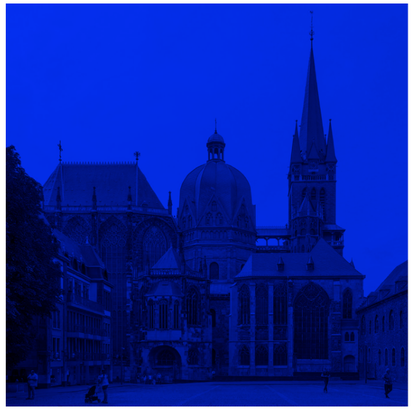
⑥ Outlook

What is an image?

color image
↓
red, green,
blue channels
↓
matrices of
intensities

33	13	76	47	69	3
68	99	37	2	16	25
3	1	48	52	52	16
12	87	24	21	44	86
19	48	0	27	64	69
66	88	31	15	16	47

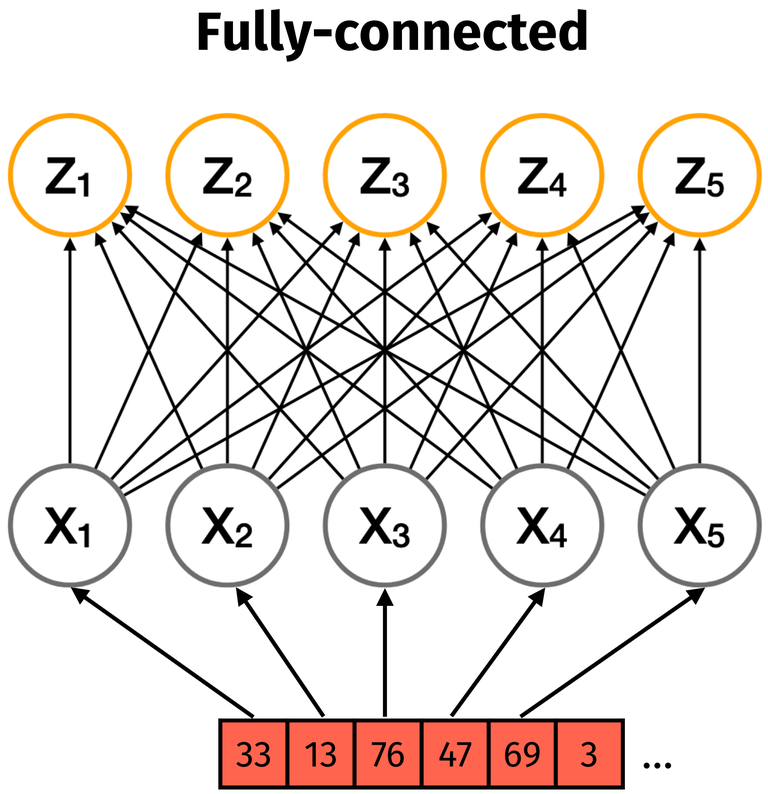
52	15	8	82	37	53
32	20	85	40	74	77
62	30	74	78	36	90
58	95	58	7	69	55
65	8	66	69	94	8
0	65	80	41	26	60



10	21	45	48	15	19
33	93	77	57	75	71
24	9	1	45	75	1
41	71	68	95	71	38
43	33	74	34	86	56
61	88	21	79	95	30

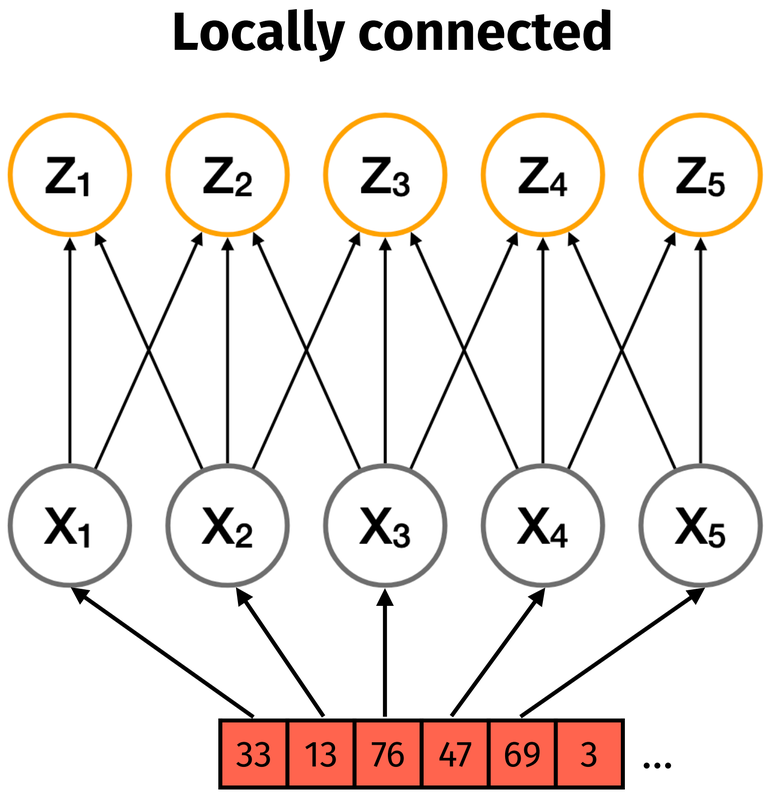
Training neural nets on images

- Naive idea: flatten all values, build fully-connected network
- Problem: network would too large, even at low resolution
- Example: CIFAR-10 RGB image
  → 32 x 32 x 3 = 3072 inputs
  → 3072 x 3072 = 9440256 parameters/layer



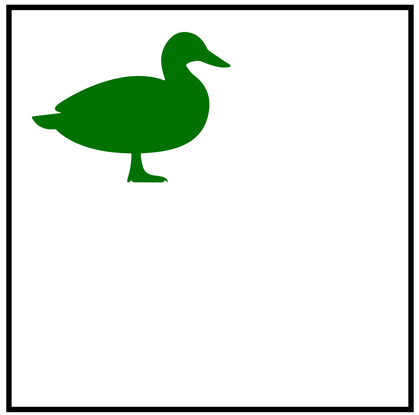
Training neural nets on images

- Problem: network would too large, even at low resolution
- Solution: image features have limited size
  → don’t need to see whole image at once
  → use **locally connected network**
- Much lower number of parameters
  → can we reduce them even more?

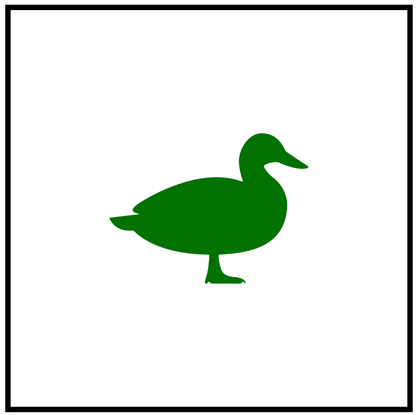


Translation invariance

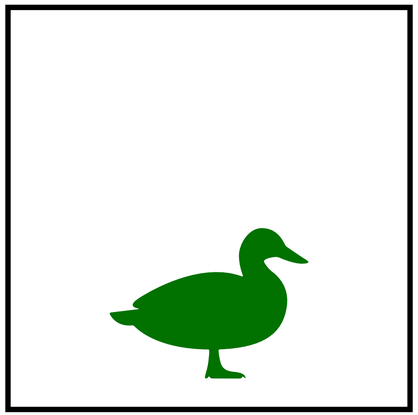
A duck



Also a duck



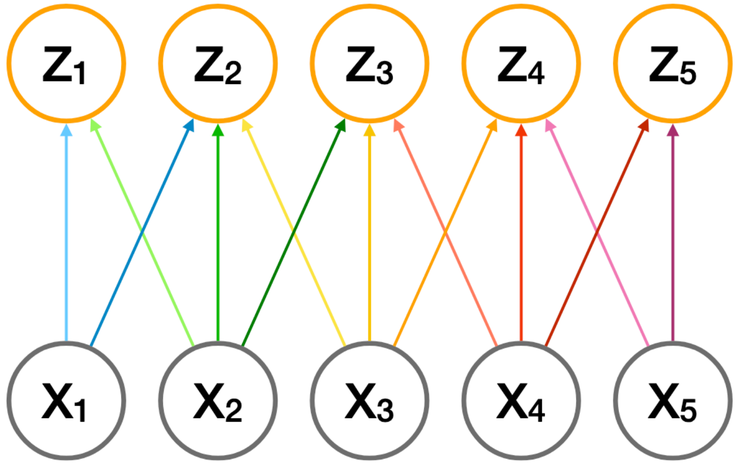
Another duck



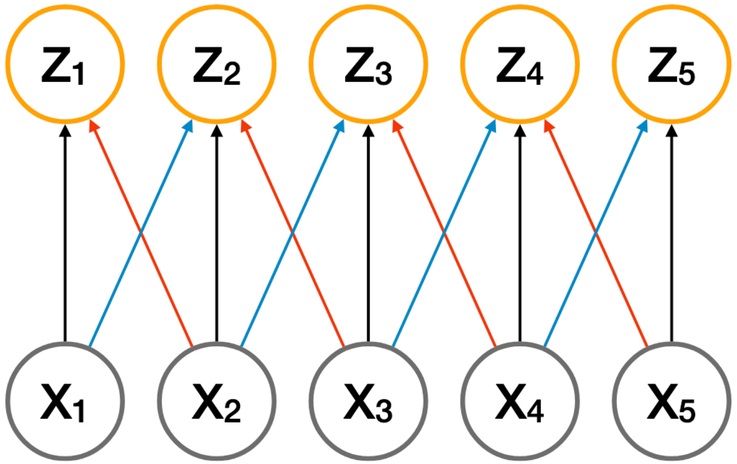
- Classification of object is independent of position in image
  → translation invariance
- With locally connected network:
  have to relearn task for every position

Parameter sharing

Locally connected



Convolution



- Classification of object is independent of position in image
  → translation invariance
- With locally connected network:
  have to relearn task for every position
- Solution: convolution

  → **shared parameters for different positions**

Outline

① Challenges of image processing

② **Introduction to convolutions**

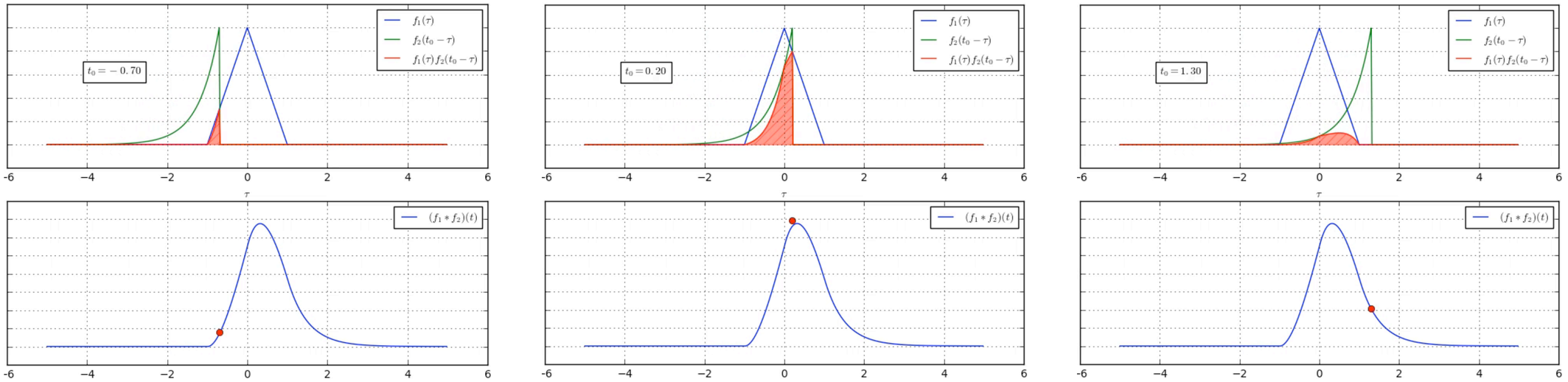
③ Convolutional layers

④ CNN building blocks

⑤ A short history of CNNs

⑥ Outlook

Mathematical convolutions



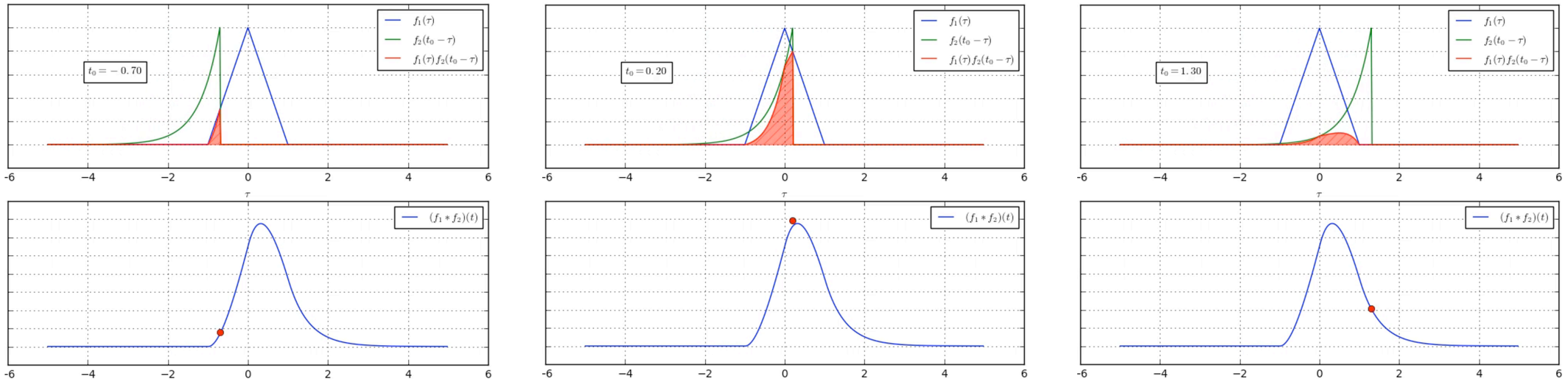
Animated illustration:
https://dspillustrations.com/pages/posts/misc/convolution-examples-and-the-convolution-integral.html

- Convolution in mathematics: defined for real-valued functions f, g

$$(f * g)(t) = \int_{-\infty}^{\infty} \mathrm{d}\tau\, f(\tau)\, g(t-\tau)$$

- Applications in statistics, signal processing, … and many more

Mathematical convolutions



Animated illustration:
https://dspillustrations.com/pages/posts/misc/convolution-examples-and-the-convolution-integral.html

- To apply to images, first discretize: $(I * K)(i) = \sum_{m=-\infty}^{\infty} I(m)K(i-m)$
- Then add a second dimension:

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m, j-n)$$

Image convolutions

Image I

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1 0	1 1	1 0	0	0
0 1	1 2	1 1	1	0
0 0	0 1	1 0	1	1
0	0	1	1	0
0	1	1	0	0

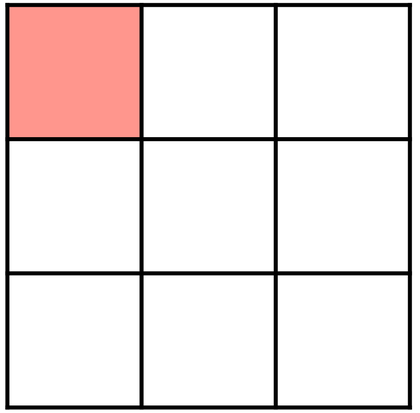
$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$



$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1 (0)	1 (1)	1 (0)	0	0
0 (1)	1 (2)	1 (1)	1	0
0 (0)	0 (1)	1 (0)	1	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4		

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1 $_0$	1 $_1$	0 $_0$	0
0	1 $_1$	1 $_2$	1 $_1$	0
0	0 $_0$	1 $_1$	1 $_0$	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4		

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1 $_0$	1 $_1$	0 $_0$	0
0	1 $_1$	1 $_2$	1 $_1$	0
0	0 $_0$	1 $_1$	1 $_0$	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4	6	

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1	1 $_0$	0 $_1$	0 $_0$
0	1	1 $_1$	1 $_2$	0 $_1$
0	0	1 $_0$	1 $_1$	1 $_0$
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4	6	

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1	1 $_0$	0 $_1$	0 $_0$
0	1	1 $_1$	1 $_2$	0 $_1$
0	0	1 $_0$	1 $_1$	1 $_0$
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4	6	4

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4	6	4

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4	6	4
2		

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Image convolutions

Image I

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$*$

Kernel K

0	1	0
1	2	1
0	1	0

$=$

Convolution $I * K$

4	6	4
2	5	6
2	5	4

$$(I * K)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(m,n)K(i-m,j-n)$$

Example: edge detection

Image I

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$*$

Kernel K

$=$

Convolution $I * K$

Example: edge detection

Image I

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

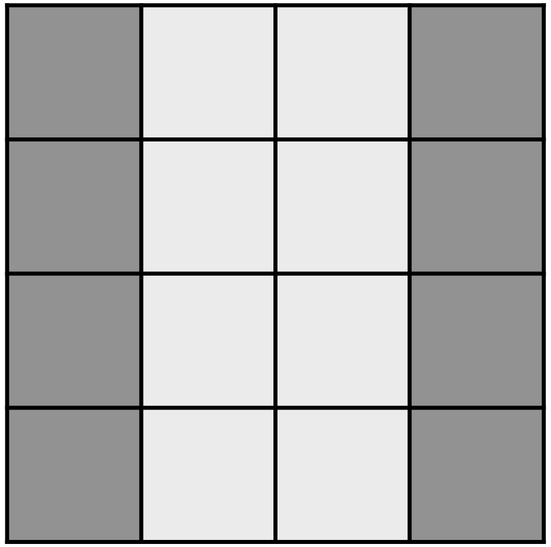
$*$

Kernel K

?	?	?
?	?	?
?	?	?

$=$

Convolution $I * K$



Example: edge detection

Image I

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$*$

Kernel K

1	0	-1
1	0	-1
1	0	-1

$=$

Convolution $I * K$

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

Example: edge detection

Image I

10	10	10	0	0	0
10					0
10					0
10					0
10					0
10	10	10	0	0	0

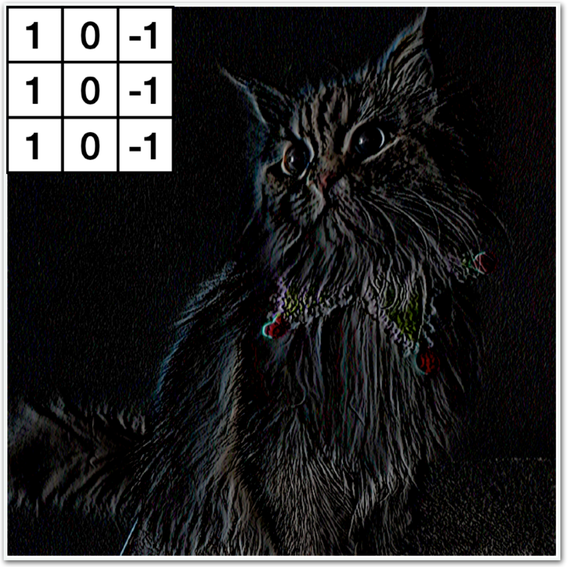
$*$

Kernel K

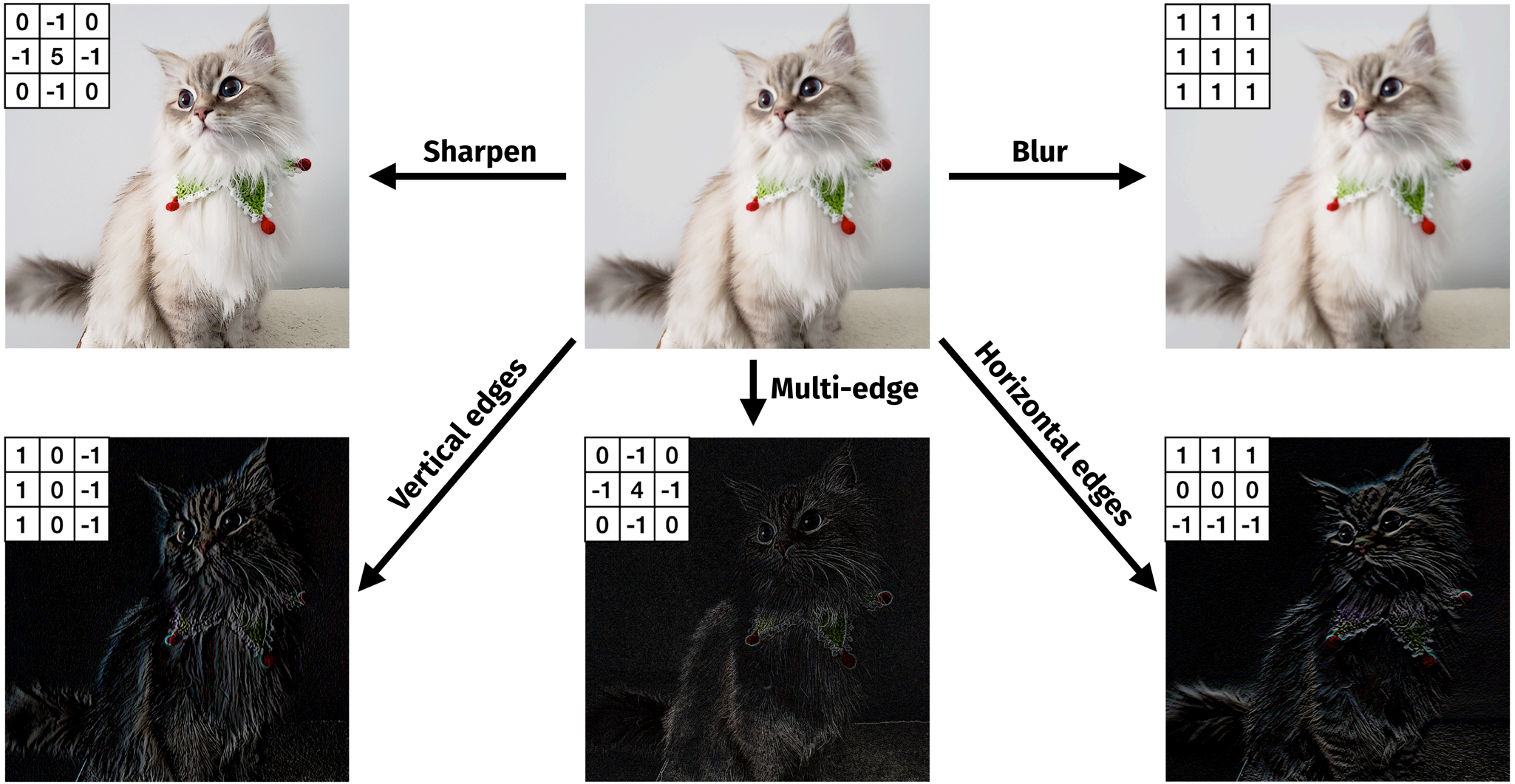
1	0	-1
1	0	-1
1	0	-1

$=$

Convolution $I * K$

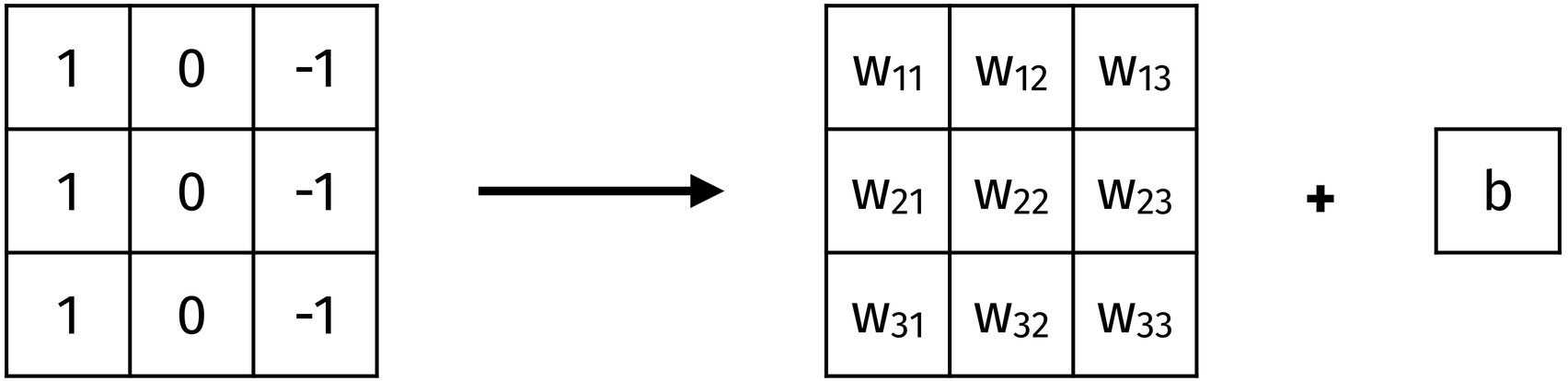


Filters for specific purposes



Machine-learned convolutions

Idea: Generalize using machine learning



Convolutional neural network layer

Replace fixed kernel entries with

learnable weight and bias

Outline

① Challenges of image processing

② Introduction to convolutions

③ **Convolutional layers**

④ CNN building blocks

⑤ A short history of CNNs

⑥ Outlook

CNN layers

```
torch.nn.Conv2d(
    in_channels,
    out_channels,
    kernel_size,
    stride=1,
    padding=0,
    dilation=1,
    groups=1,
    bias=True,
    padding_mode='zeros',
    device=None,
    dtype=None
)
```

Most important settings

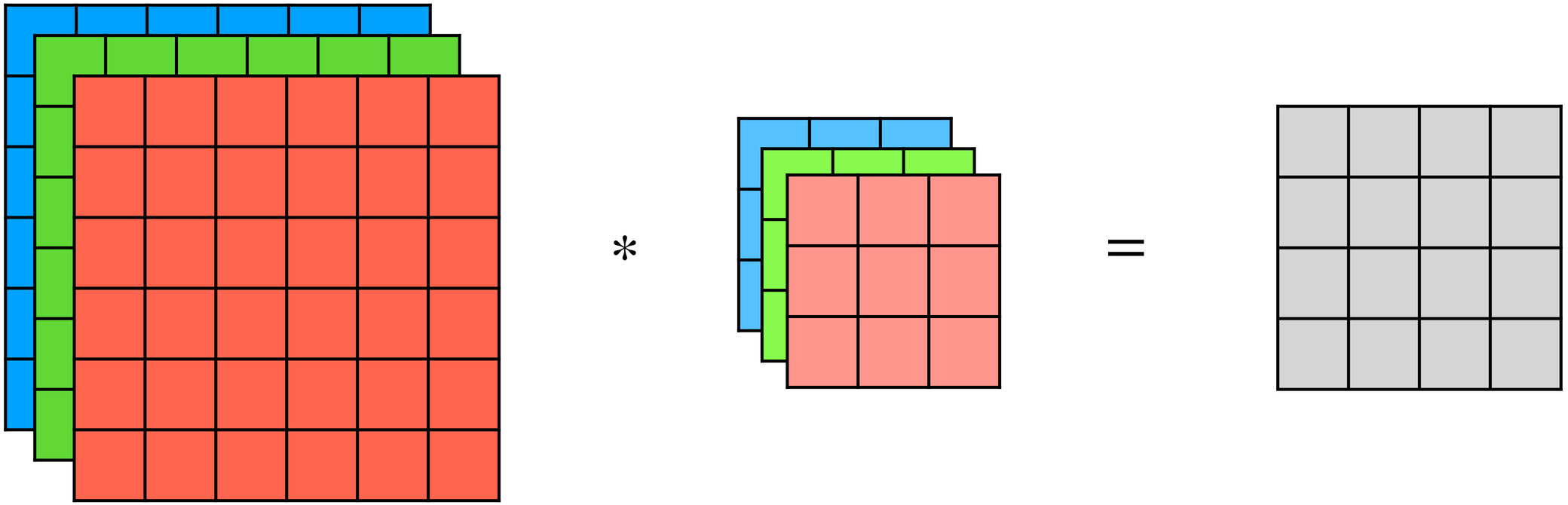
① Channels

② Kernel size

③ Padding

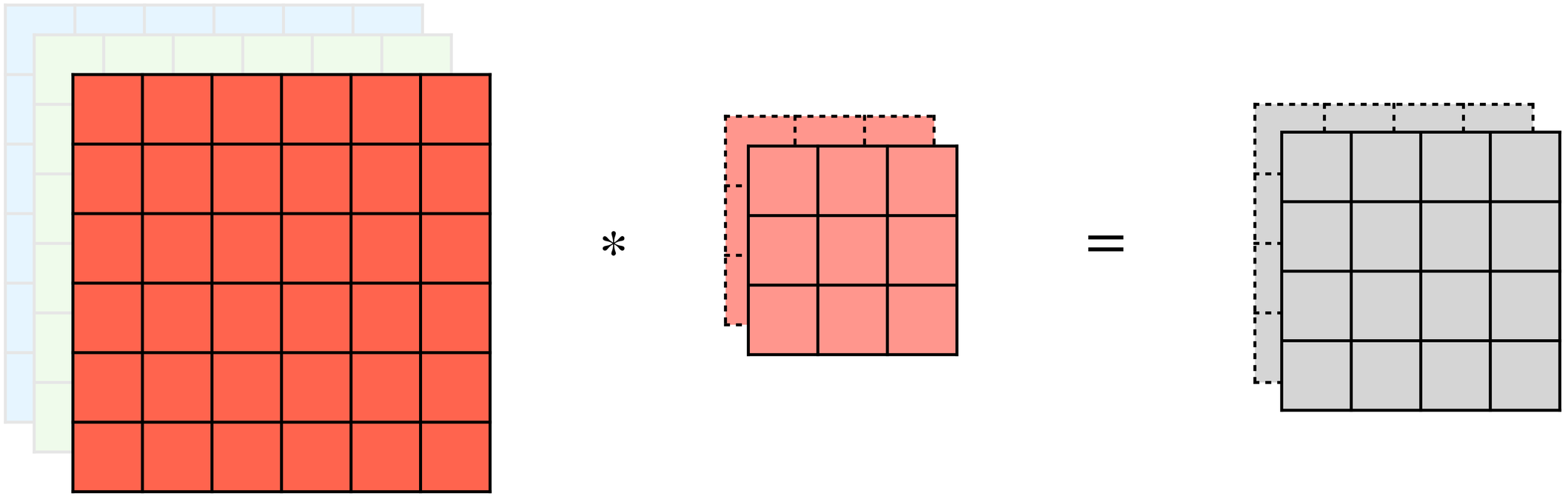
④ Strides

Channels



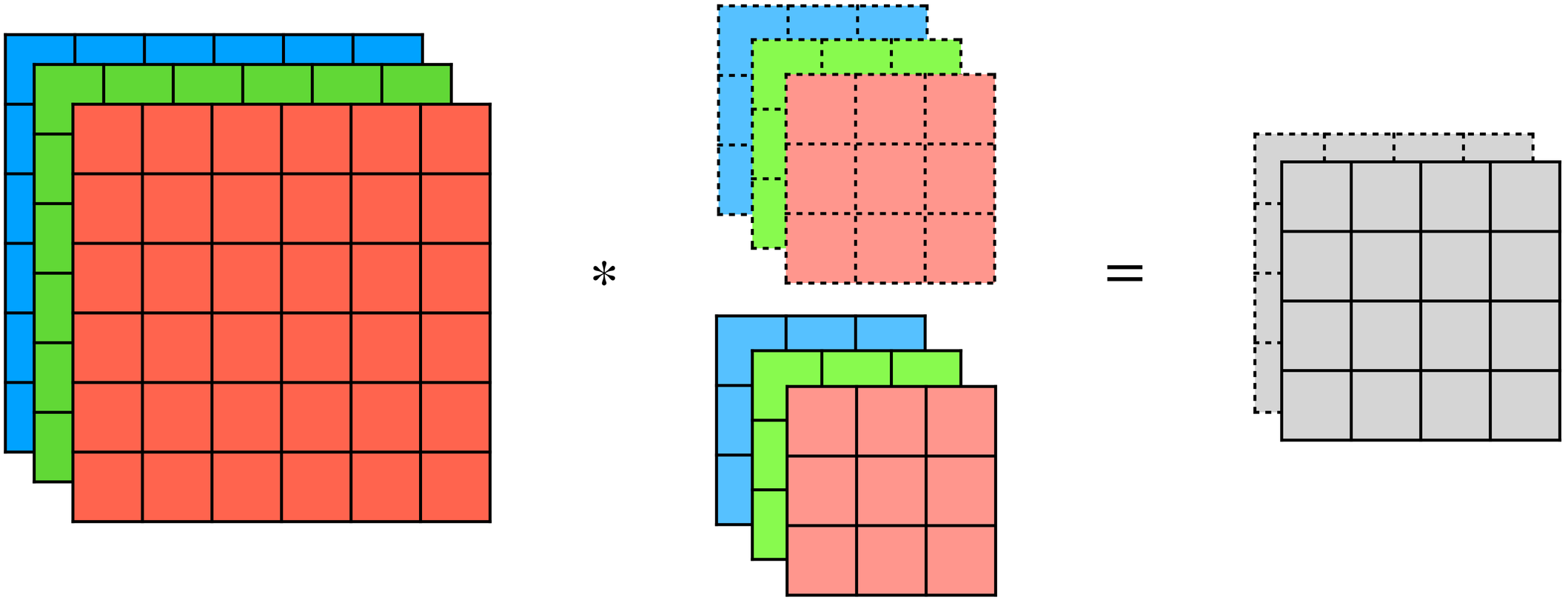
- can have multiple inputs, for example red, green, blue channels
- separate **kernel for each input, result is sum**

Channels



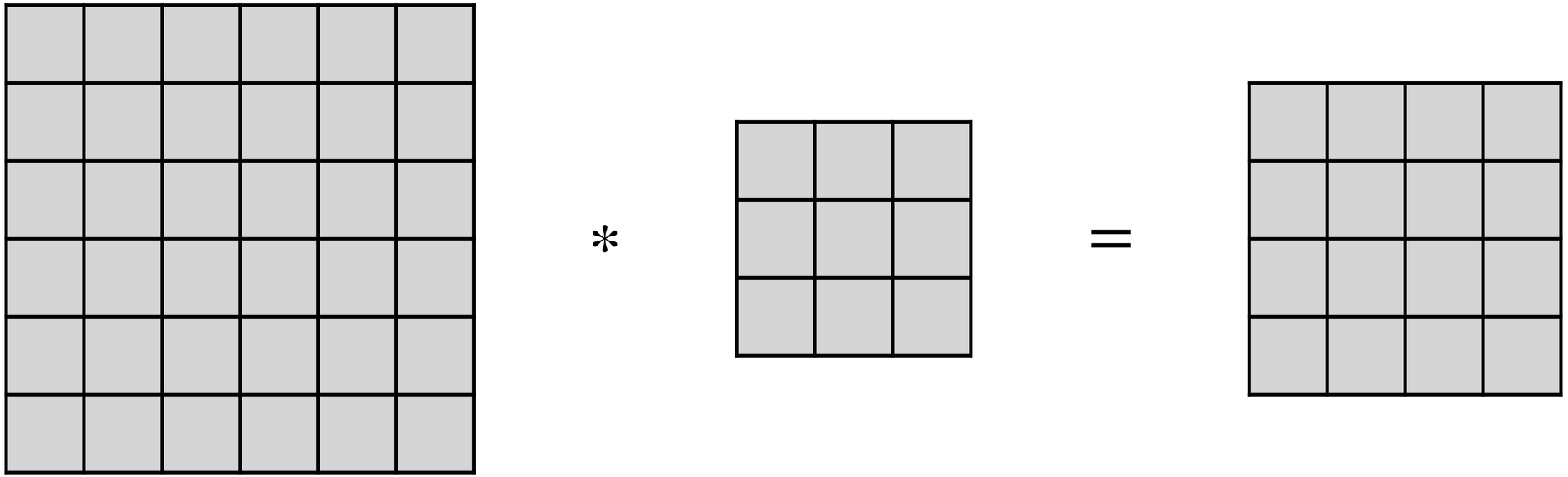
- one kernel may not be enough to capture all important features
- multiple **kernels with independent weights**
- new output channel for each kernel

Channels



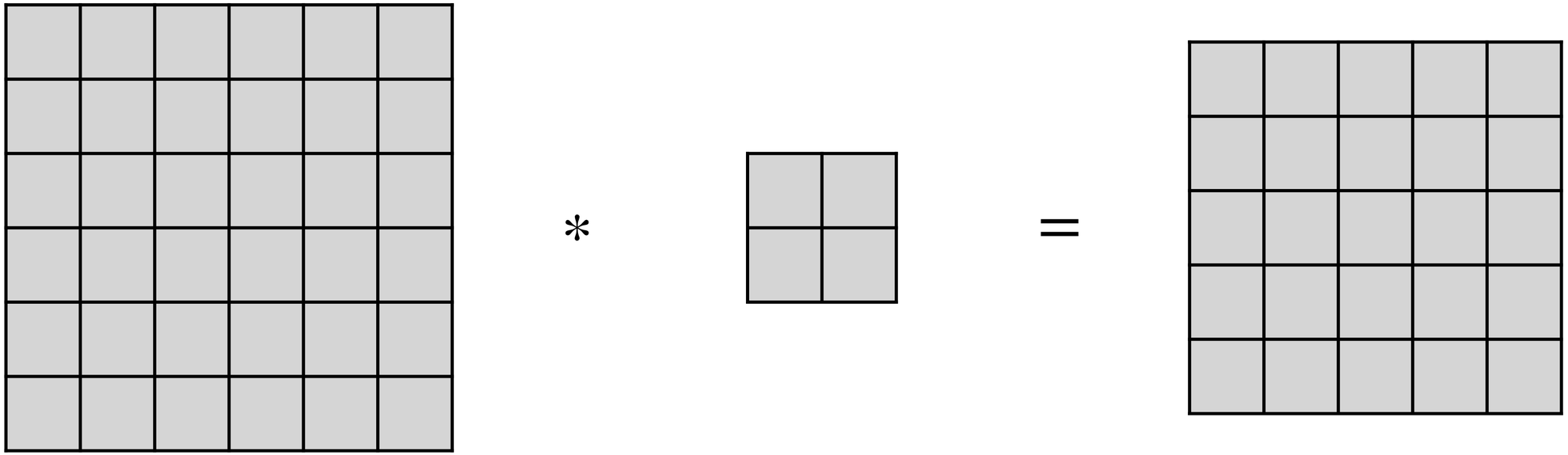
- Combine both: $O_i = I_j * K_{ij}$ with $\mathrm{N_{in}} \times \mathrm{N_{out}}$ independent kernels
- Sum over input channels j

Kernel size



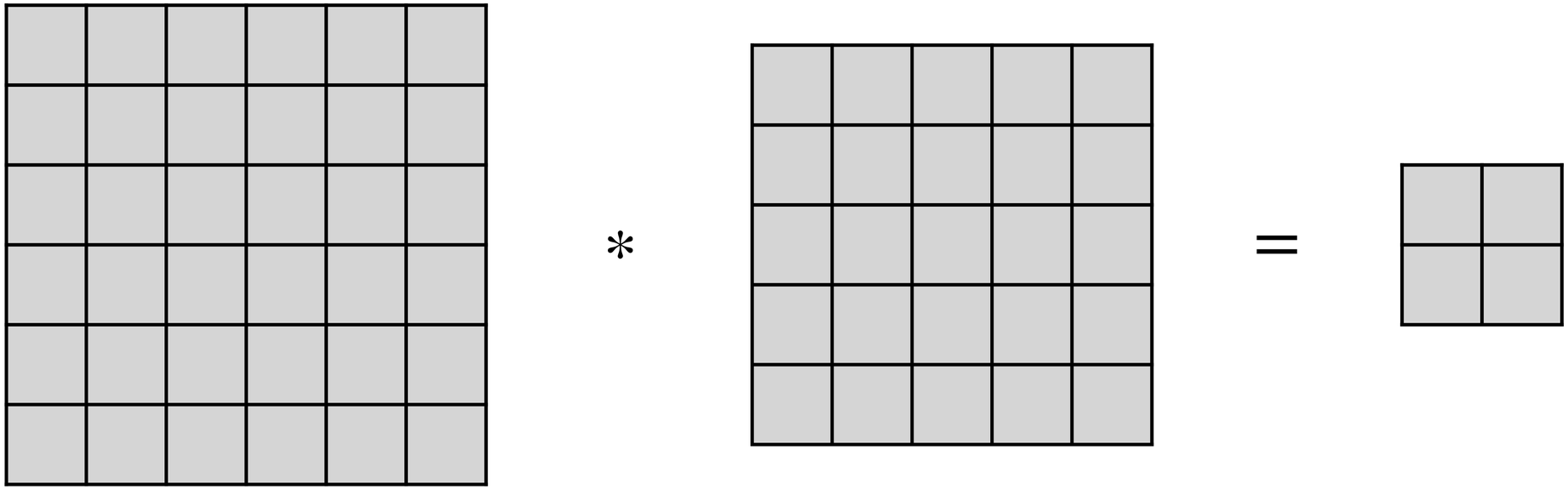
- kernel size **determines how far pixels are connected**

Kernel size



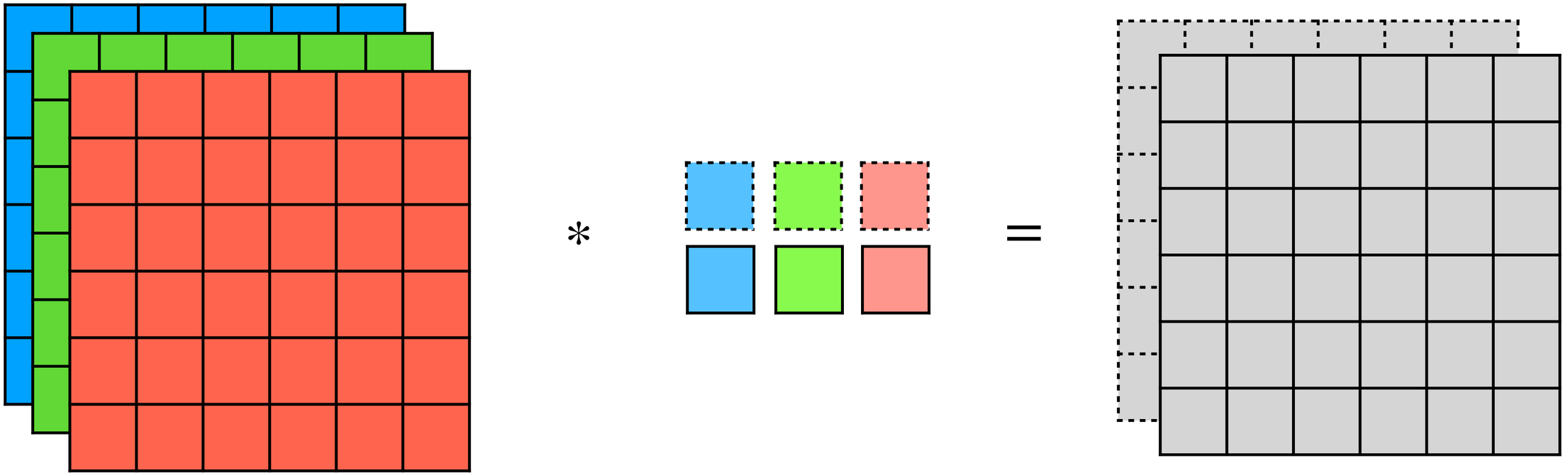
- kernel size **determines how far pixels are connected**
- too small: can’t cover interesting features

Kernel size



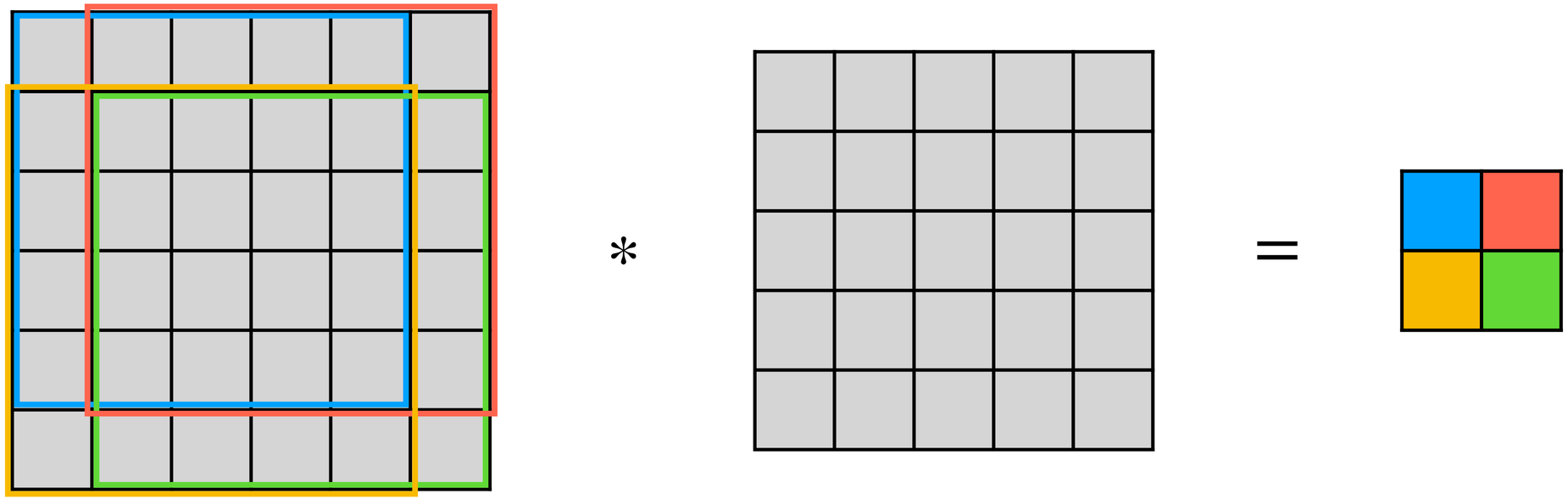
- kernel size **determines how far pixels are connected**
- too small: can’t cover interesting features
- too large: don’t profit from weight sharing any more

1x1 Convolutions



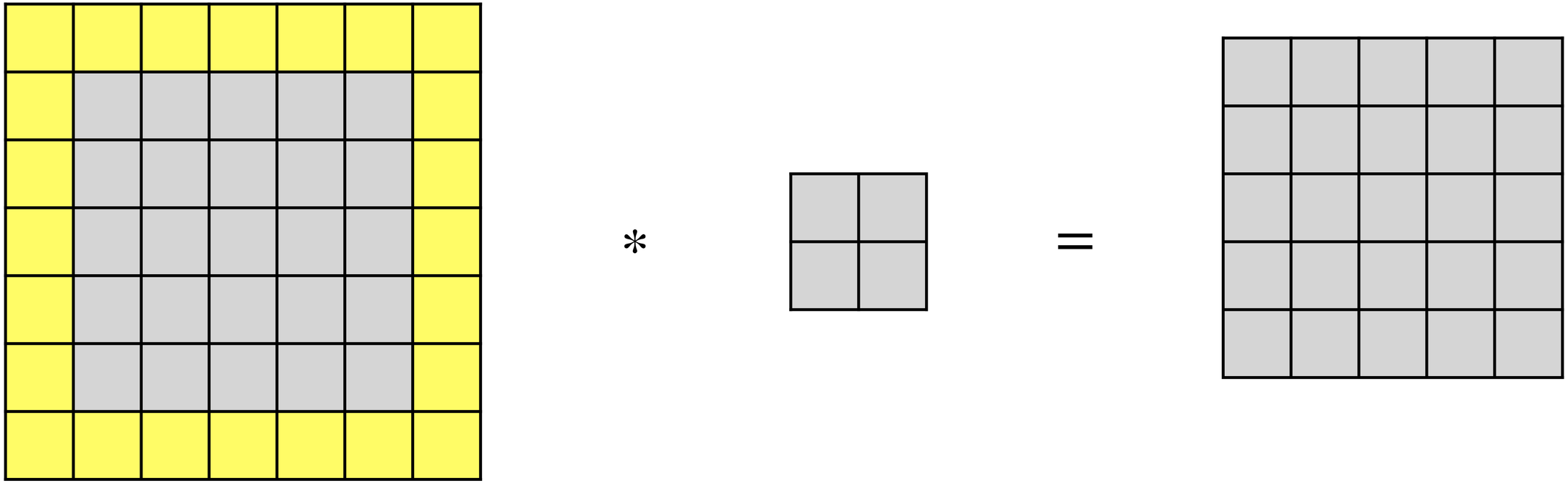
- Interesting special case: 1x1 convolution
- Acts like **linear layer applied to every pixel** individually

Output size



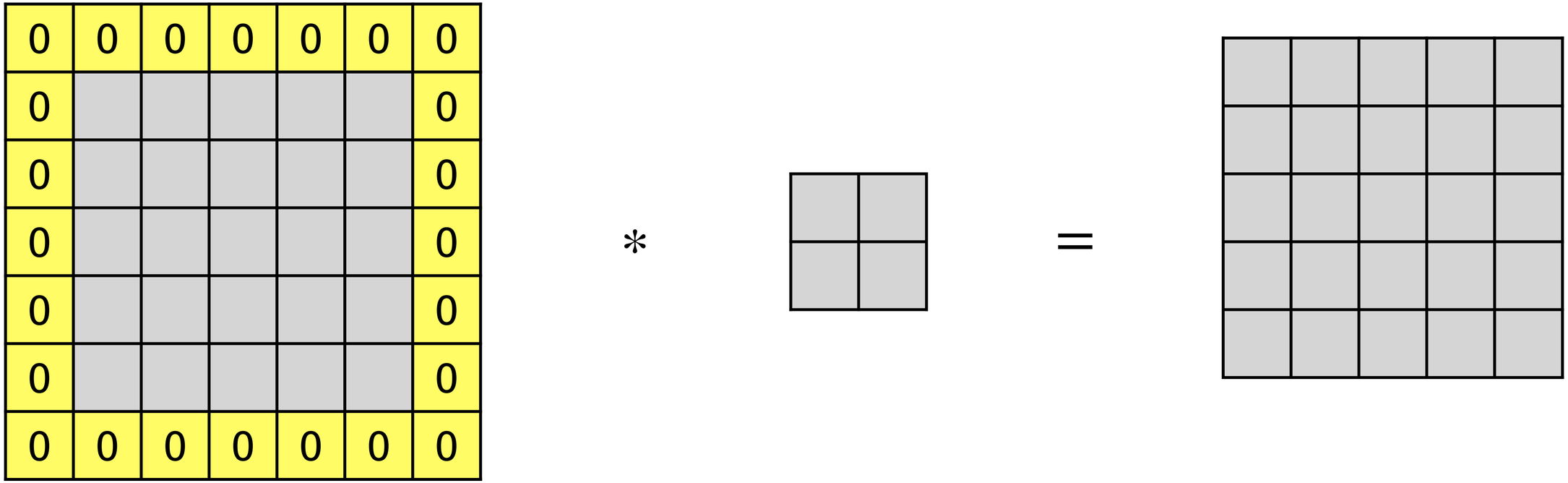
- kernel size affects output size
- image and kernel need to fully overlap
  → output always smaller than input
- output size = input size - kernel size + 1

Padding



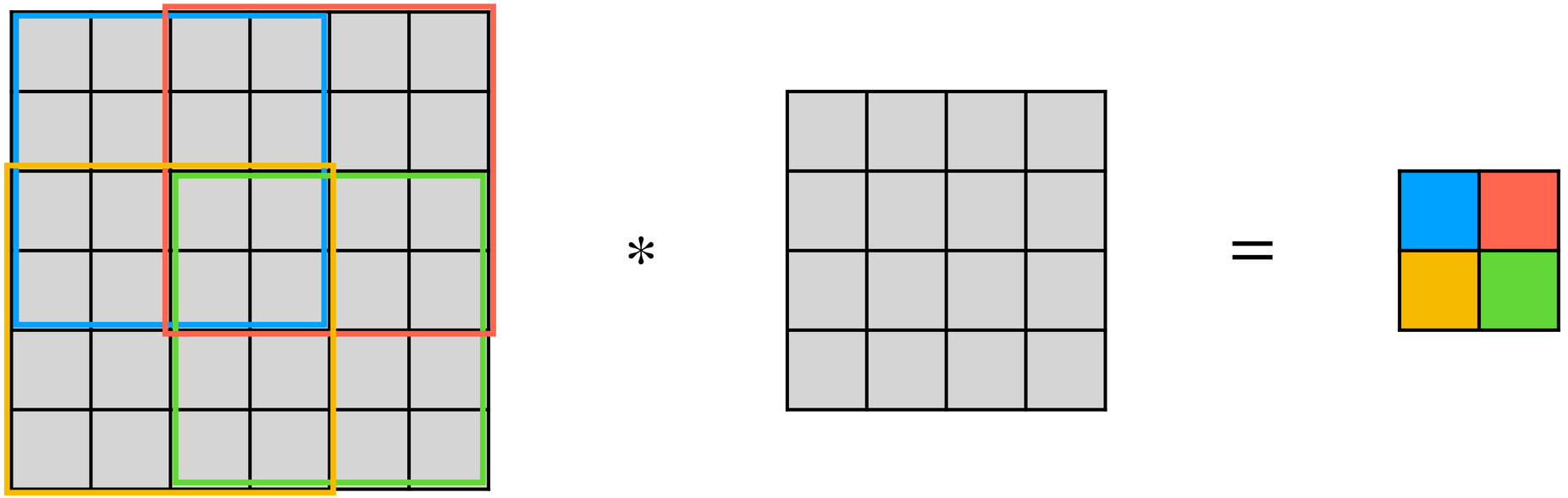
- use padding to **prevent reduction in output size**
- PyTorch: `padding="same"` such that output size = input size
- Also possible: fixed value for padding size

Padding



- use padding to **prevent reduction in output size**
- PyTorch: `padding="same"` such that output size = input size
- Also possible: fixed value for padding size
- Default mode: use zeros for padding

Strides



- strides: amount that kernel moves per step
- can be greater than one

  → **combined convolution and downsampling**

CNN layers

```
torch.nn.Conv2d(
    in_channels,
    out_channels,
    kernel_size,
    stride=1,
    padding=0,
    dilation=1,
    groups=1,
    bias=True,
    padding_mode='zeros',
    device=None,
    dtype=None
)
```

Most important settings

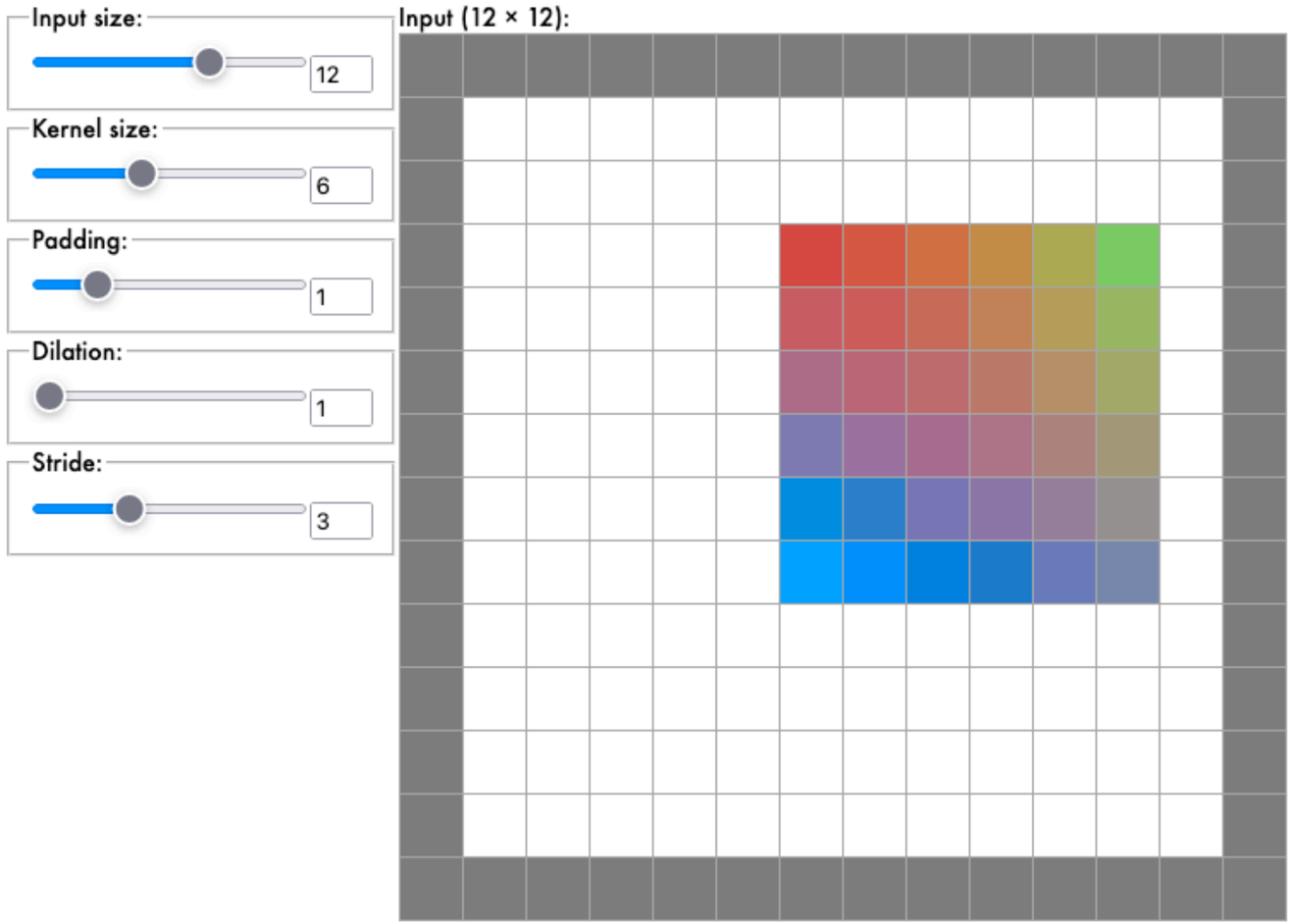
① **Channels**
number of input/output feature maps

② **Kernel size**
choose according to size of features

③ **Padding**
add pixels to change output size

④ **Strides**
step size, allows for downsampling

Interactive visualization



https://ezyang.github.io/convolution-visualizer/

Outline

① Challenges of image processing

② Introduction to convolutions

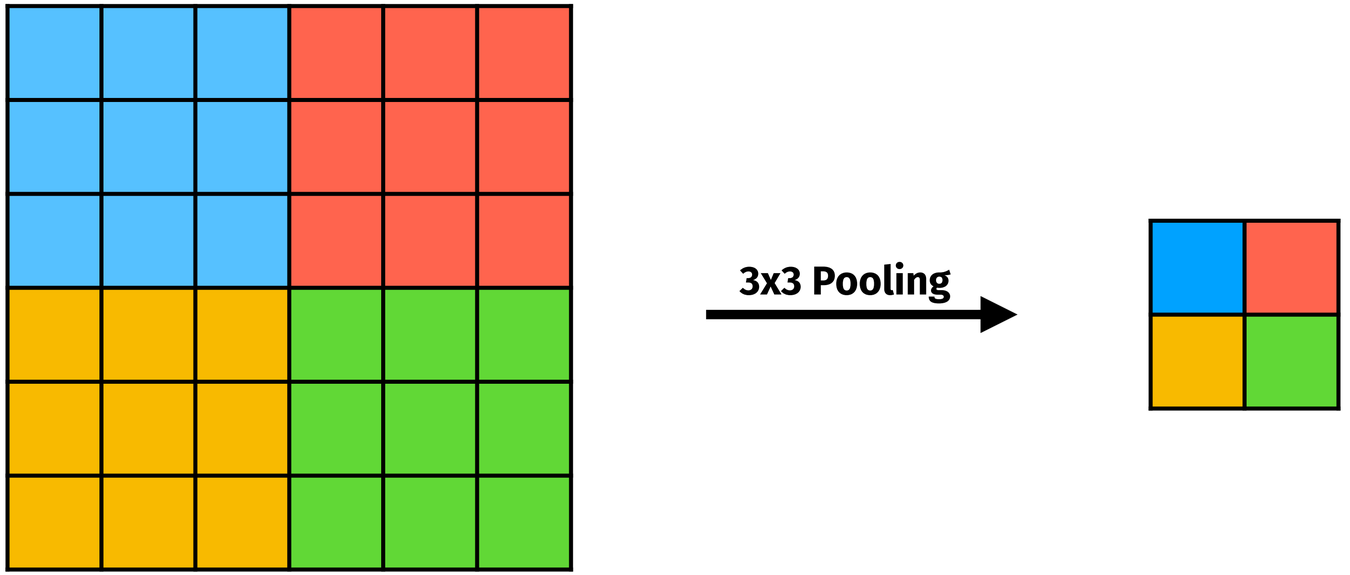
③ Convolutional layers

④ **CNN building blocks**

⑤ A short history of CNNs

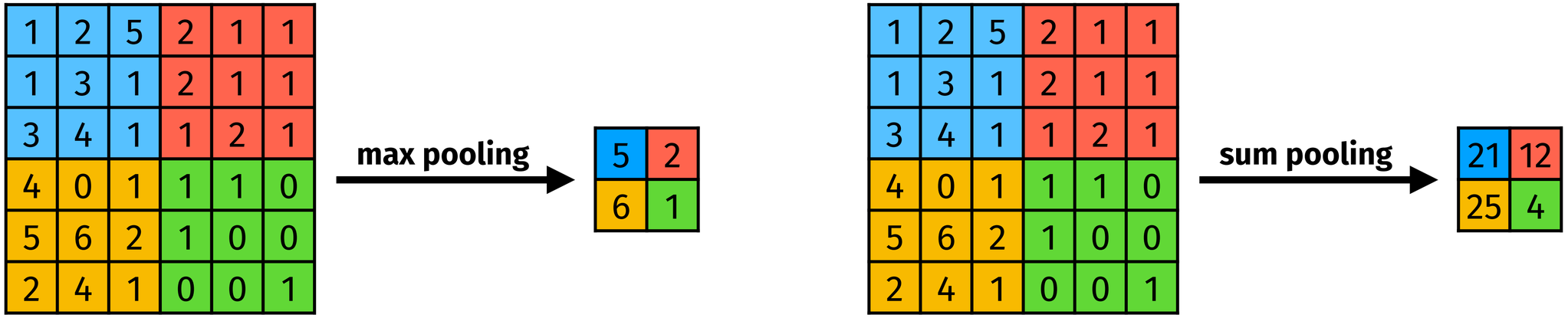
⑥ Outlook

Pooling



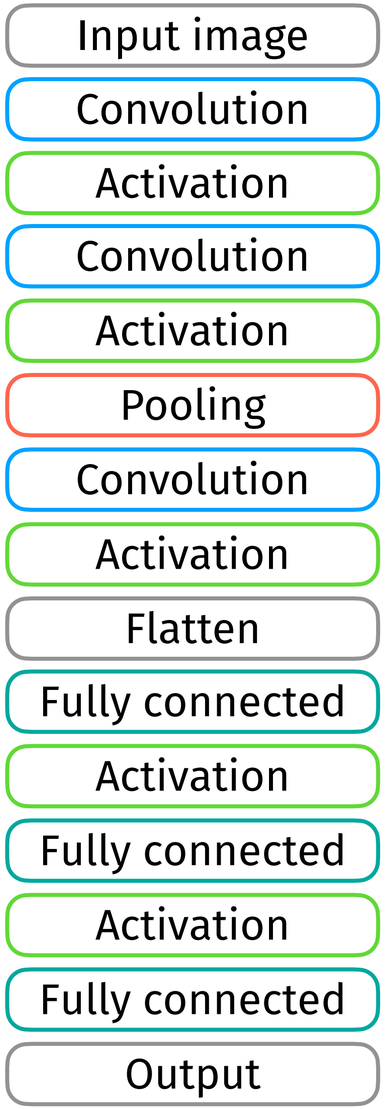
- Often need to **reduce image size**
- Seen before: convolution with strides
- Alternative: combine values from multiple pixels to one
- output size = input size / kernel size

Pooling



- Different functions can be used for pooling
  → max pooling: focus on high value pixels
  → sum pooling: preserve total intensity
  → average pooling: preserve average intensity
- PyTorch: `torch.nn.MaxPool2d`, `torch.nn.AvgPool2d`

Typical CNN architecture



Convolution typically followed by activation function

Often multiple convolutional layers in a row

Pooling layer after one or more convolutions

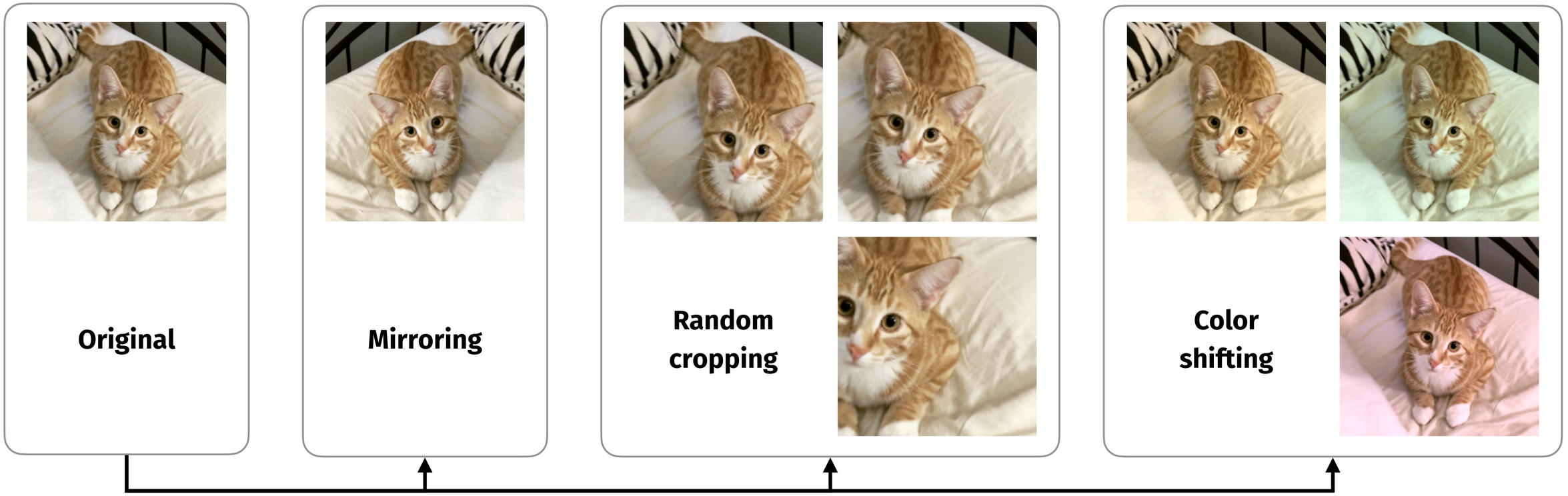
Then more convolutional layers

Last step: task-specific “head”

For example: flatten output and apply fully connected network for classification

Data augmentation

- CNNs: translation-invariant up to boundary effects
- want network output to be invariant under other transformations
  → cropping, mirroring, rotations, shearing, warping, color shifting
- simple solution: **increase training statistics by augmenting data**



Outline

① Challenges of image processing

② Introduction to convolutions

③ Convolutional layers

④ CNN building blocks

⑤ **A short history of CNNs**

⑥ Outlook

Early CNN: LeNet

Input image: 28 x 28 x 1

Convolution, 5x5 kernel, 2 padding: 28 x 28 x 6

Sigmoid activation

Average pooling, 2x2 kernel, no pad: 14 x 14 x 6

Convolution, 5x5 kernel, no pad: 10 x 10 x 16

Sigmoid activation

Average pooling, 2x2 kernel, no pad: 5 x 5 x 16

Flatten

Fully connected, 120 nodes

Activation

Fully connected, 84 nodes

Activation

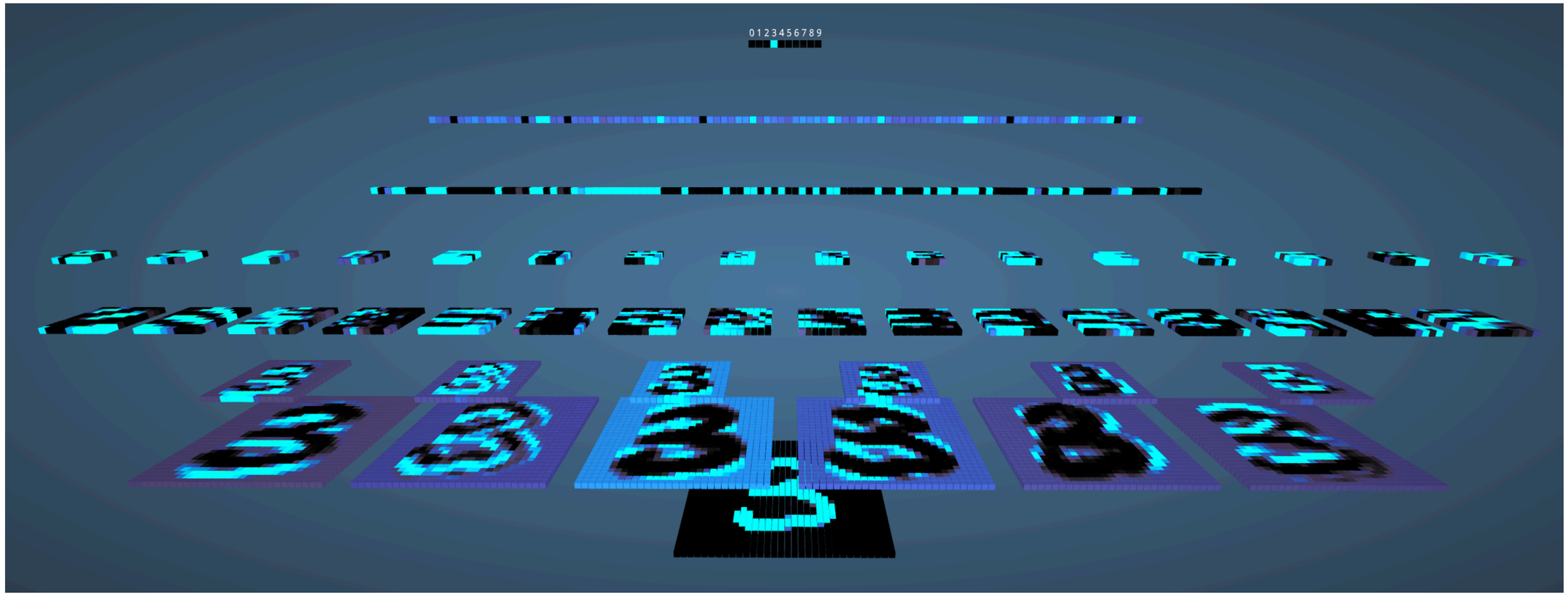
Fully connected, 10 nodes

Output: 1 of 10 classes

- Multiple versions developed by Yann LeCun between 1989 - 1995
- First **CNN trained using backpropagation**
- Classification of handwritten digits
- 1994: MNIST dataset created
  → still common toy problem today

- Left: LeNet-5 architecture (1995)

Interactive 3D visualization



https://adamharley.com/nn_vis/cnn/3d.html

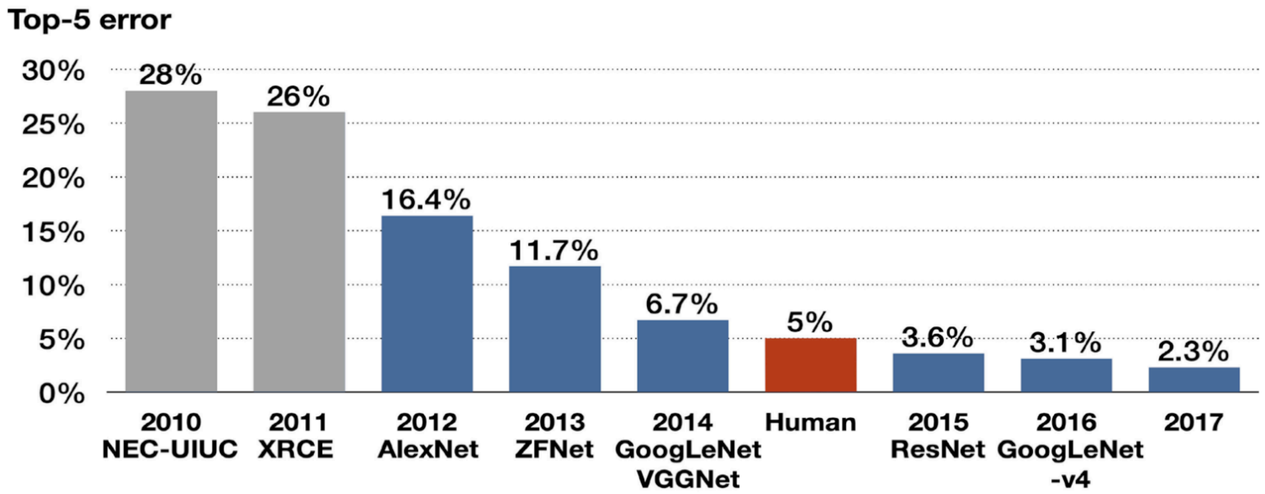
A short history of CNNs

LeNet
First CNN trained
using backpropagation

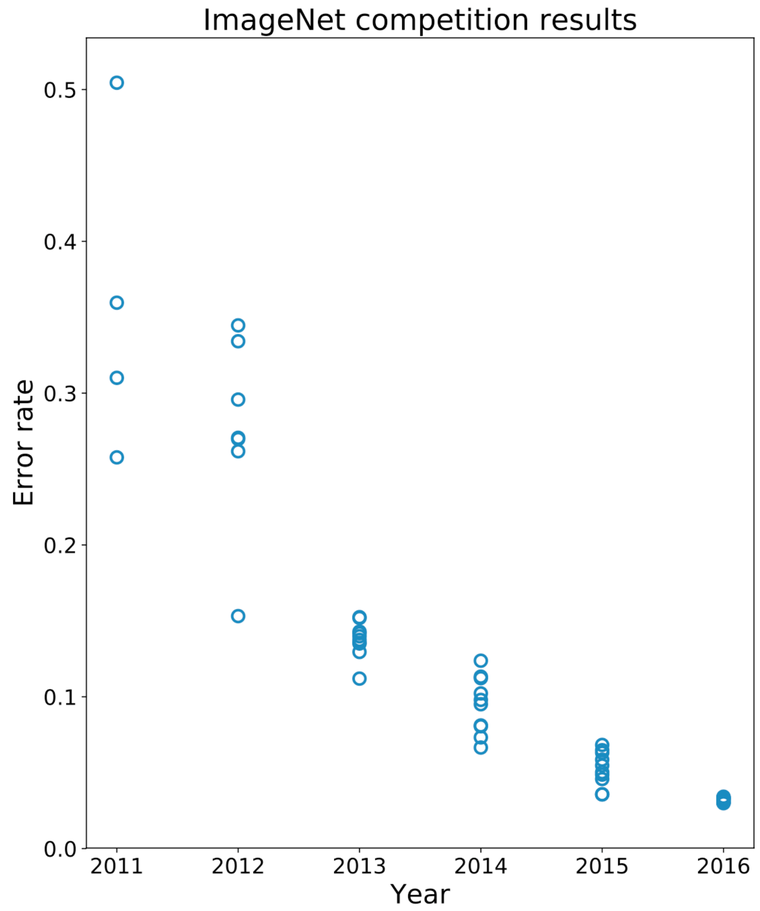
1989 - 1995

ImageNet challenge

- ImageNet dataset: more than 14M images
- hand-labeled, more than 10k categories
- Since 2010: annual ImageNet challenge
  → classification with 1000 categories
- Metric: true label must be within top 5 of predicted labels



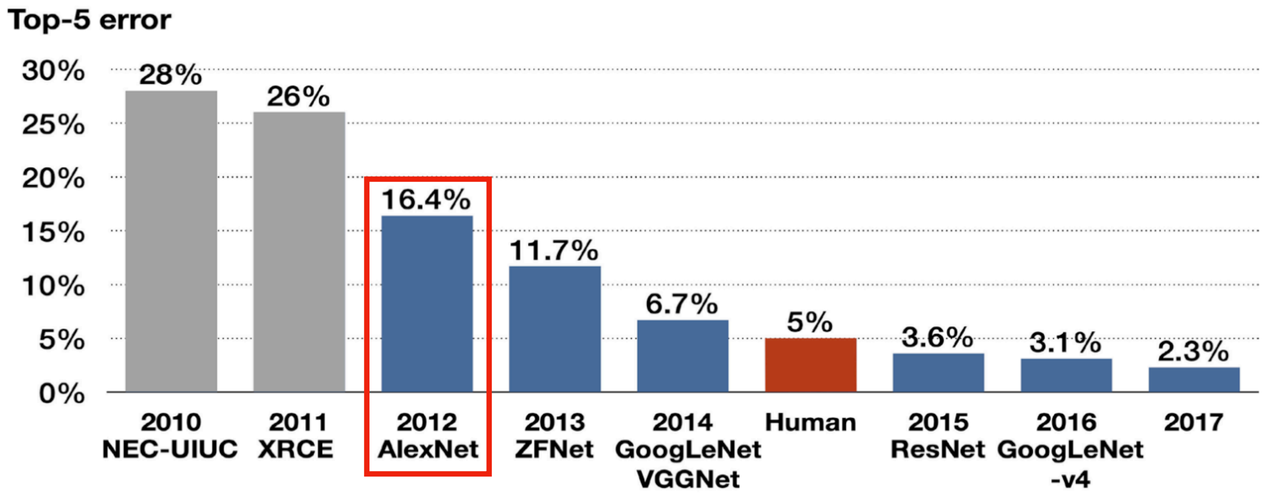
Dae-Young Kang et. al.: Application of Deep Learning in Dentistry and Implantology, 10.32542



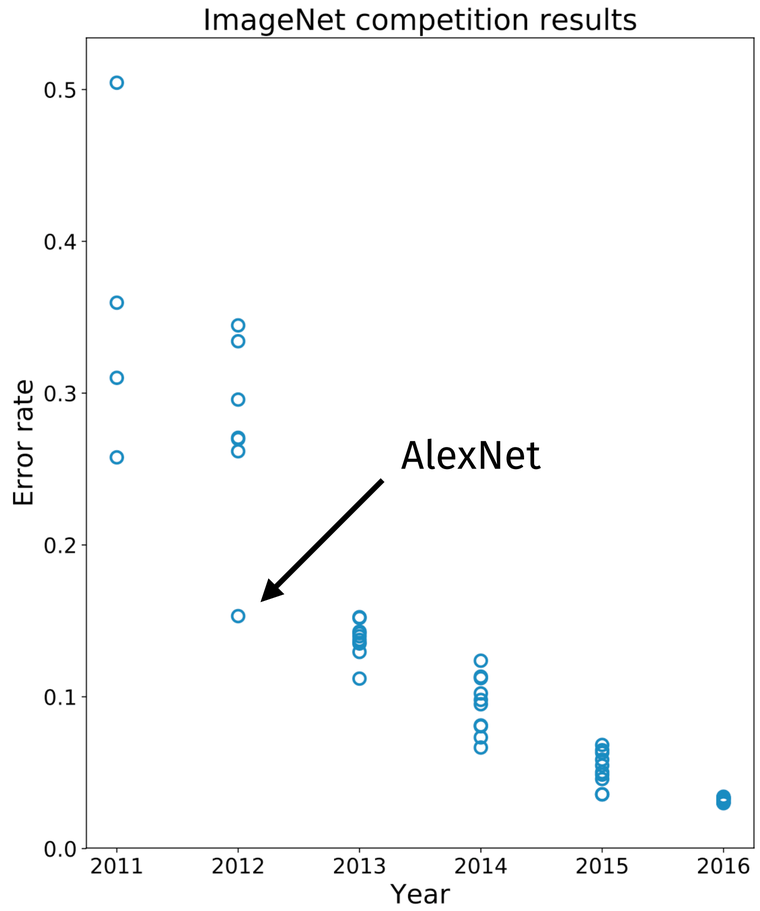
By Gkrusze - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=69750373

ImageNet challenge

- ImageNet dataset: more than 14M images
- hand-labeled, more than 10k categories
- Since 2010: annual ImageNet challenge
  → classification with 1000 categories
- Metric: true label must be within top 5 of predicted labels



Dae-Young Kang et. al.: Application of Deep Learning in Dentistry and Implantology, 10.32542

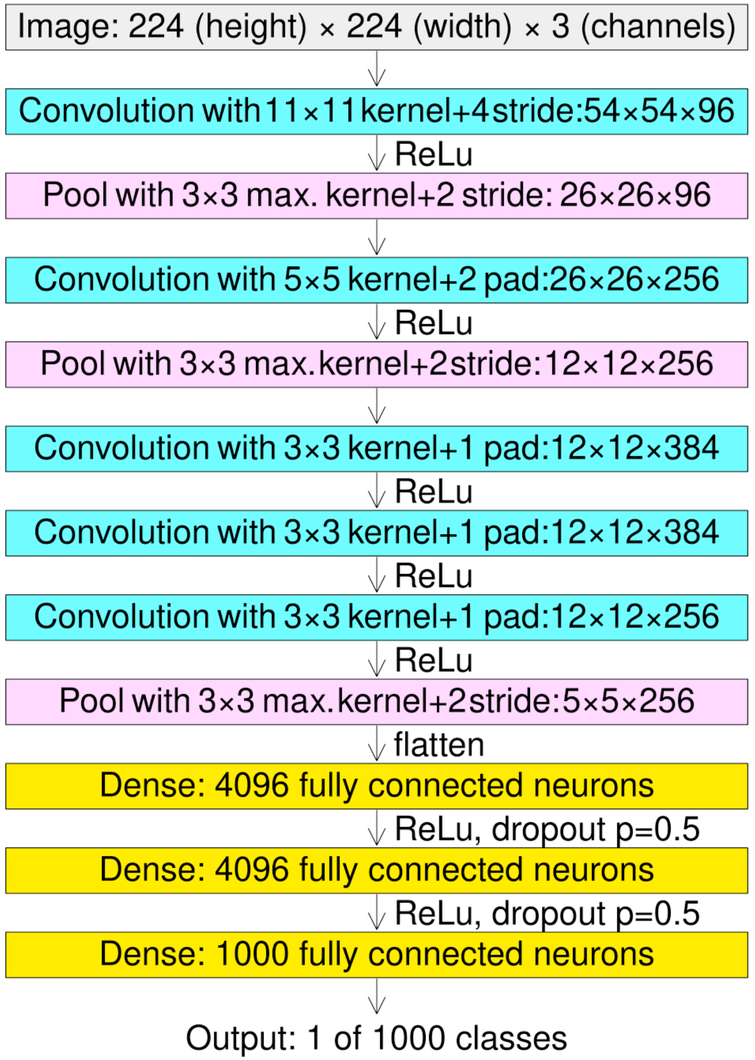


By Gkrusze - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=69750373

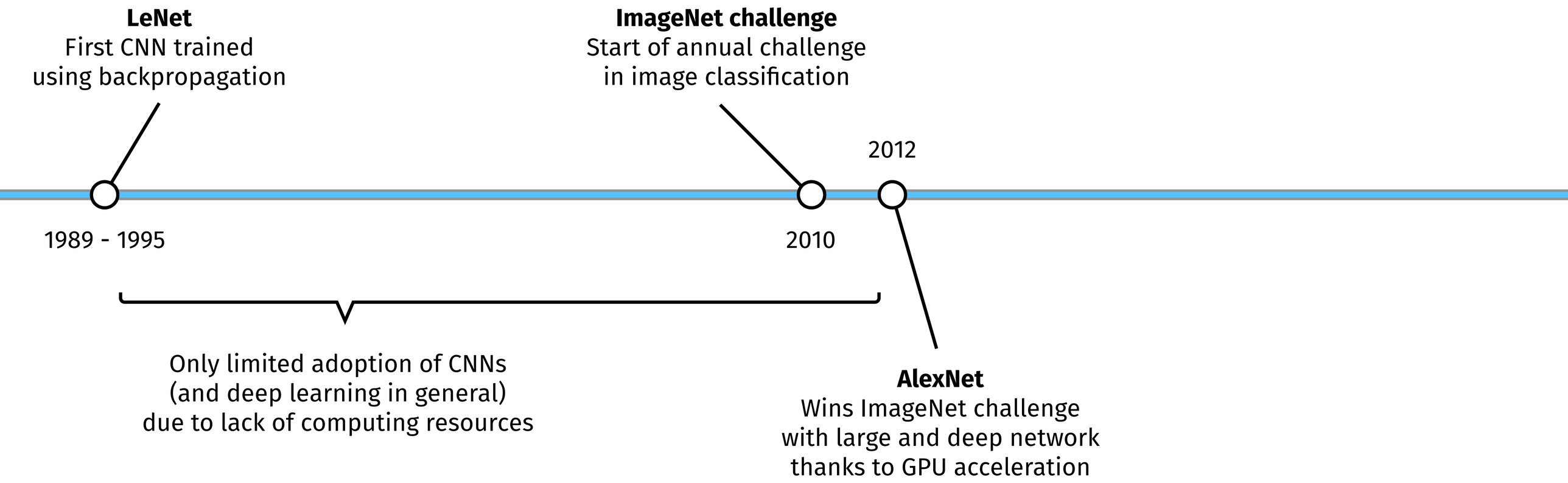
AlexNet

- Developed by Alex Krizhevsky et al
- Similar to LeNet structurally, but much wider and deeper
- Among the **first to use GPUs for NNs**

  → **much larger network possible**
- Won 2012 ImageNet challenge with much lower error rate than competitors
- Highly influential, often claimed to have started today's deep learning boom

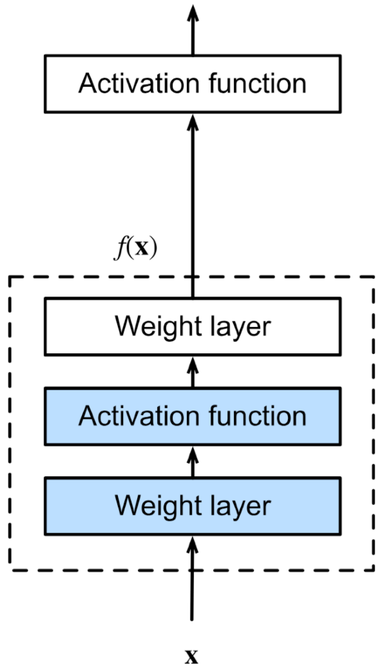


A short history of CNNs



ResNet

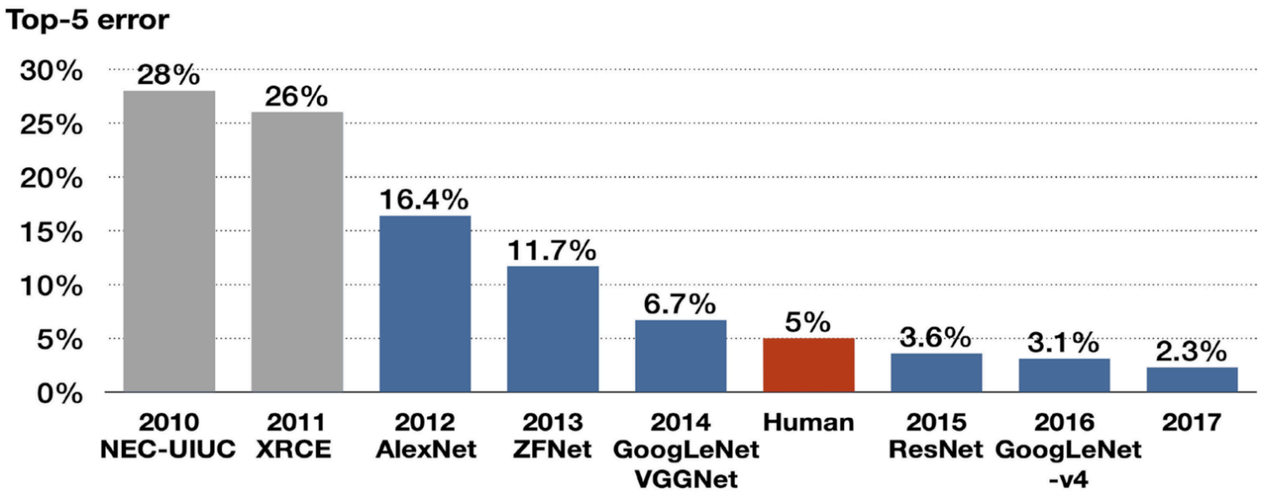
- Standard CNN: **depth limited** due to **vanishing gradients problem**
- The issue:
  → negative weights
  → negative outputs
  → ReLU activation yields 0
  → gradient is 0
  → optimizer does not adapt weights
- More likely to happen for deep networks



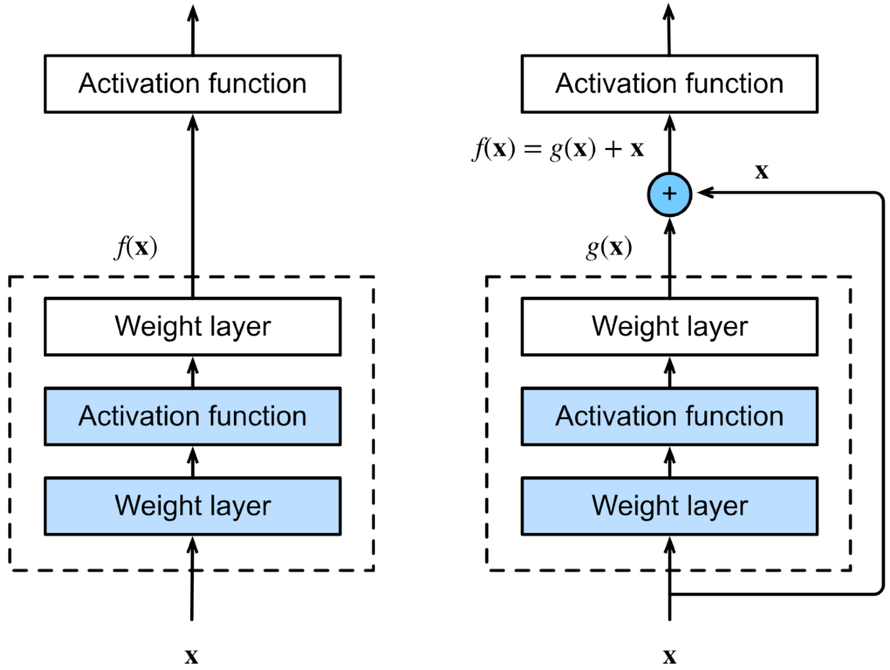
https://github.com/d2l-ai/d2l-en

ResNet

- Solution: add residual/skip connections
- Convolutions learn modification to input
- Gradients can propagate easily
  → deeper networks possible
  → can build more expressive models
- Successful in 2015 ImageNet challenge



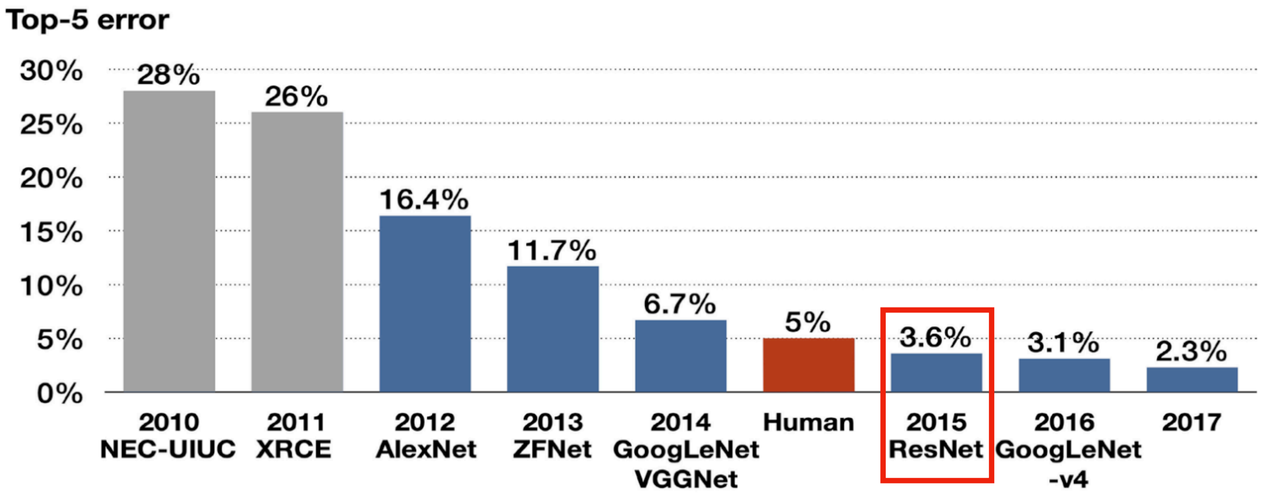
Dae-Young Kang et. al.: Application of Deep Learning in Dentistry and Implantology, 10.32542



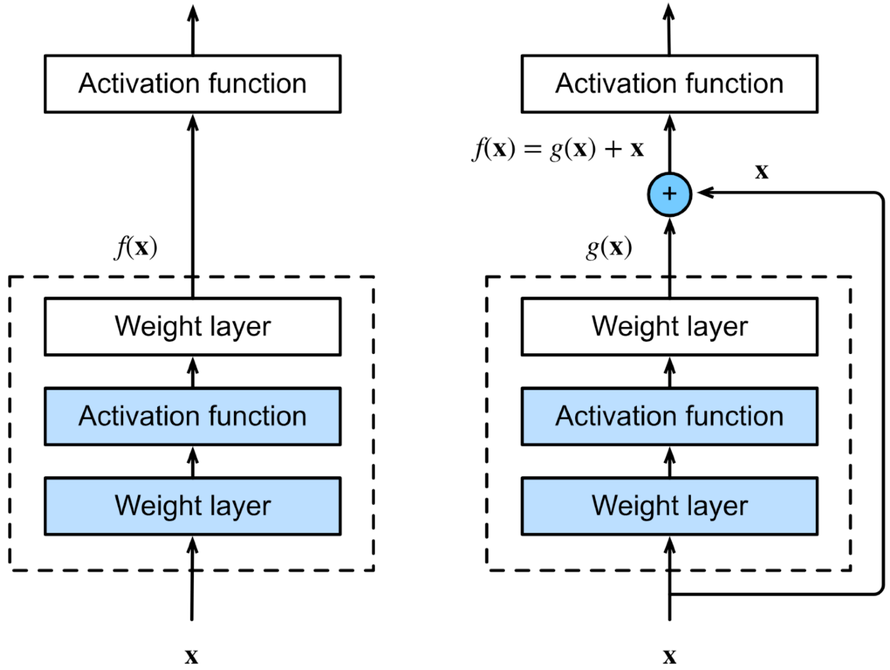
https://github.com/d2l-ai/d2l-en

ResNet

- Solution: add residual/skip connections
- Convolutions learn modification to input
- Gradients can propagate easily
  → deeper networks possible
  → can build more expressive models
- Successful in 2015 ImageNet challenge

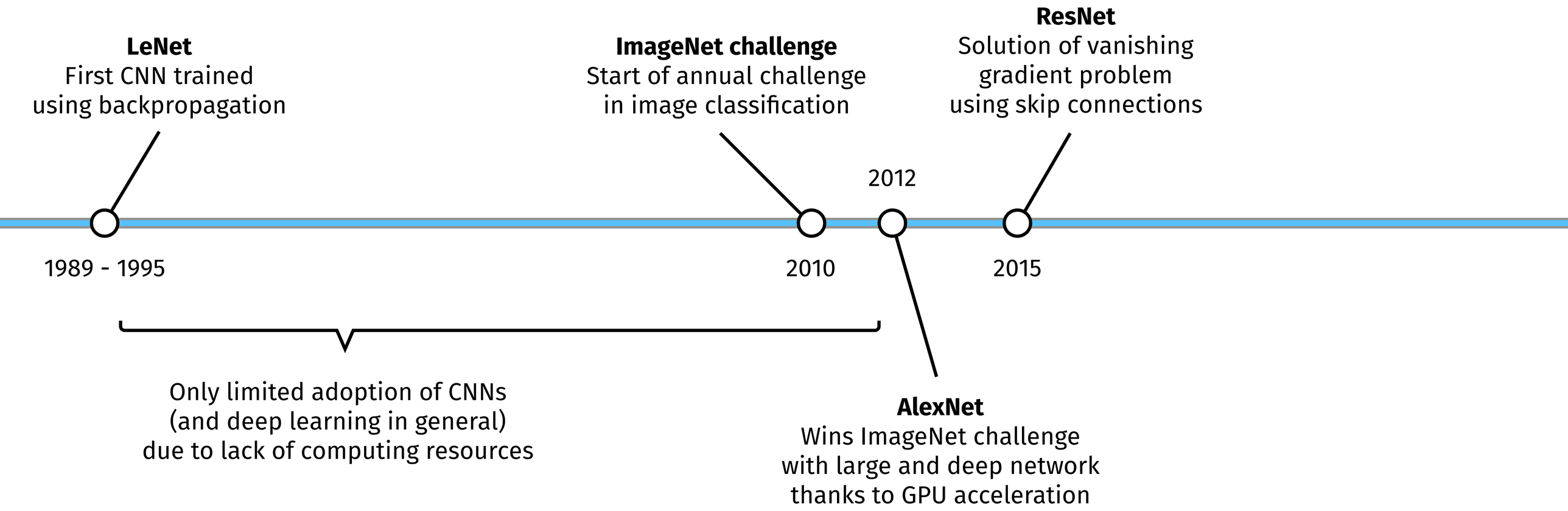


Dae-Young Kang et. al.: Application of Deep Learning in Dentistry and Implantology, 10.32542

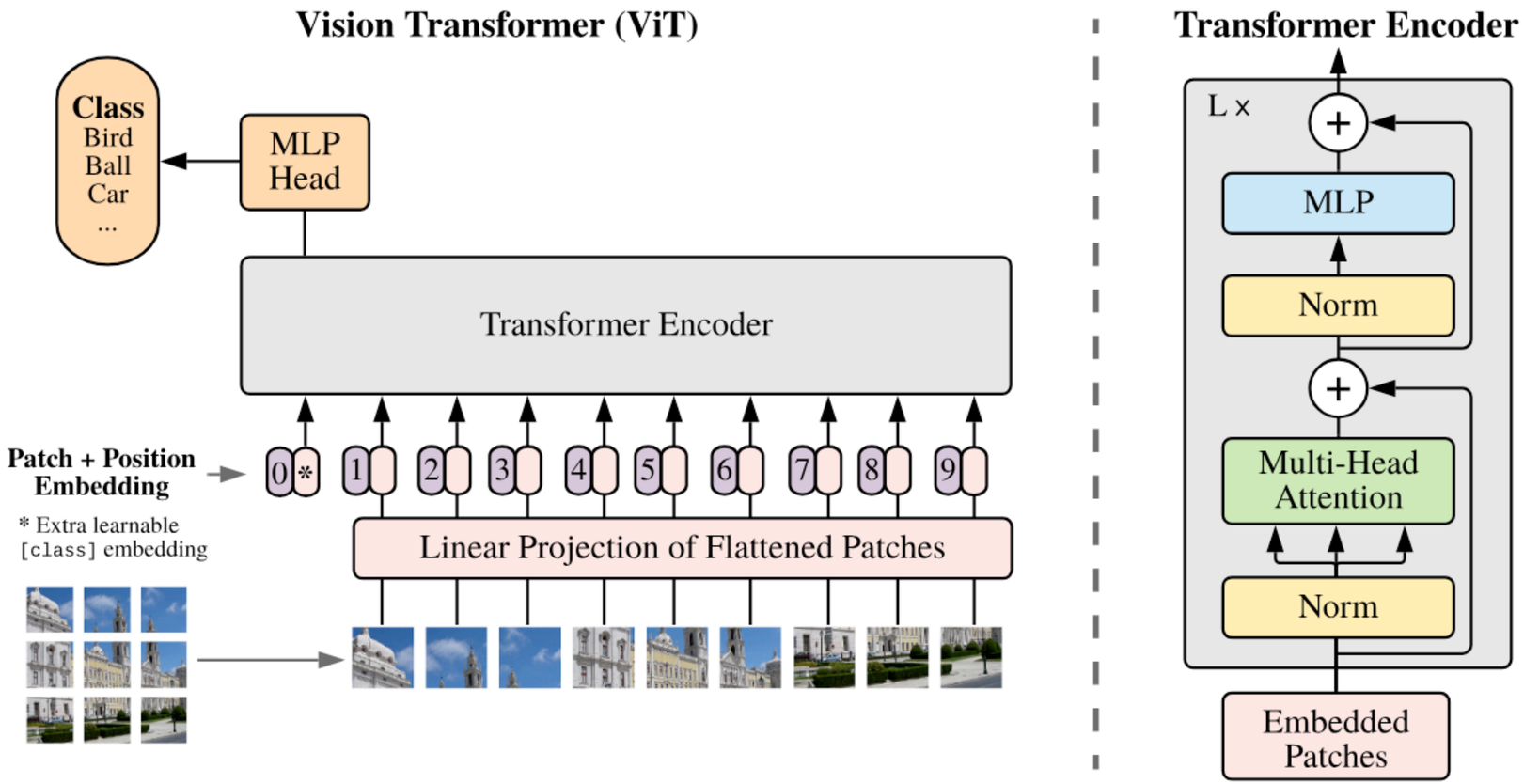


https://github.com/d2l-ai/d2l-en

A short history of CNNs



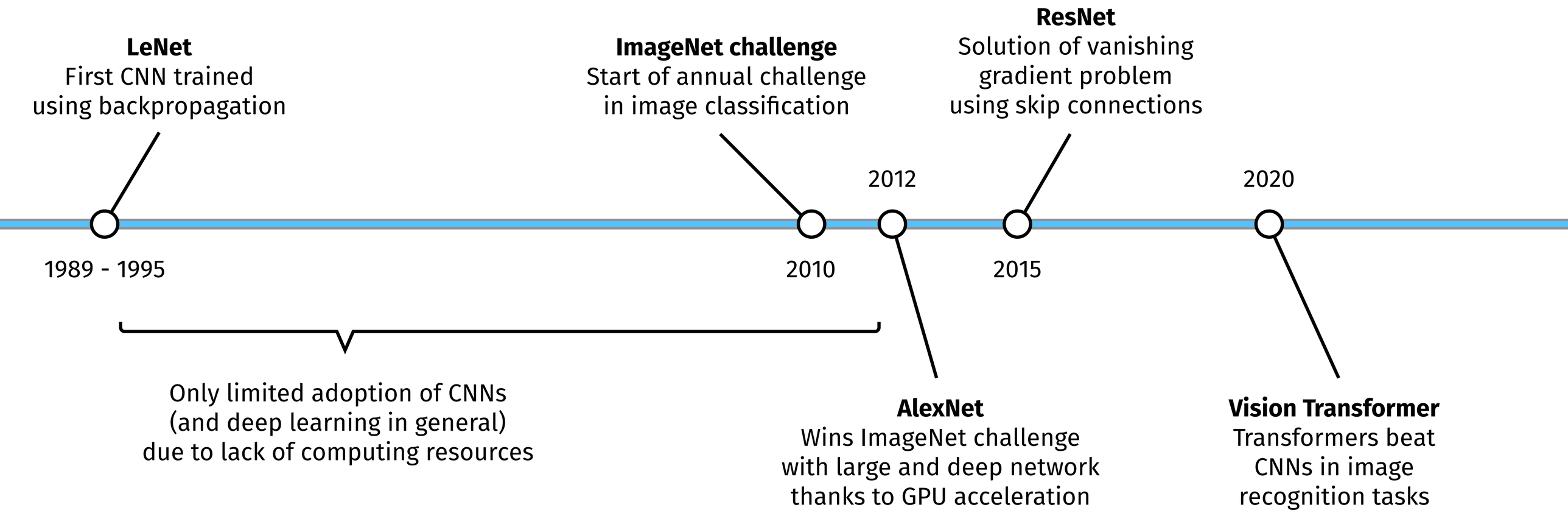
Vision transformers



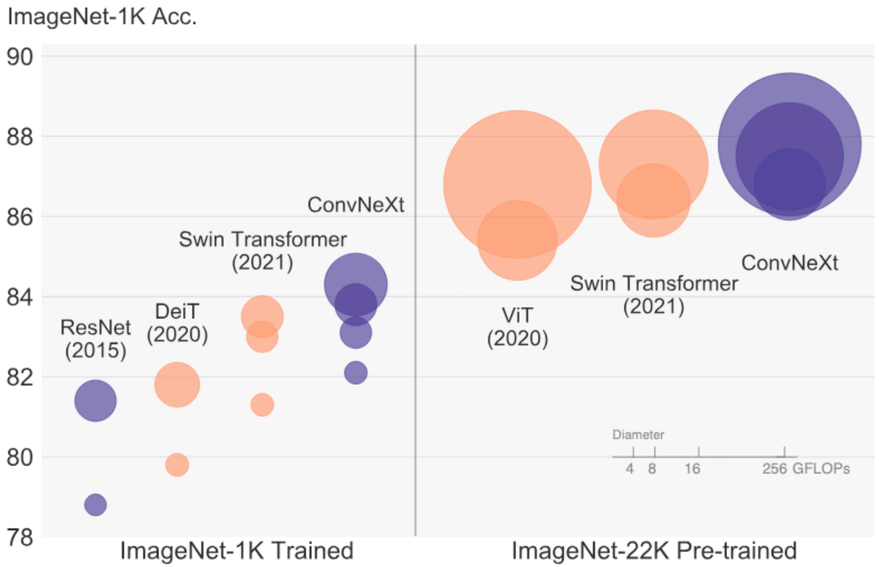
- Cut image into 16x16 patches, use as input sequence to transformer
- like CNNs: localization + weight sharing
- beats CNNs in common tasks at time of publication

Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale" arXiv:2010.11929 (2020)

A short history of CNNs

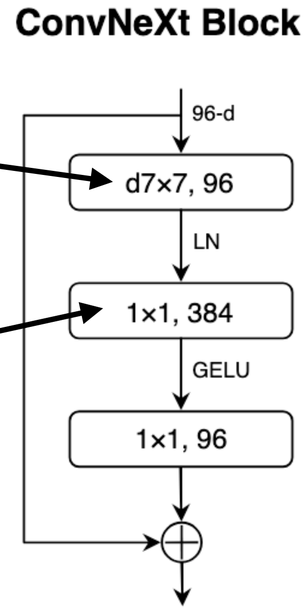


ConvNext



depth-wise convolutions
only one kernel per channel

channels interact via
1x1 convolutions

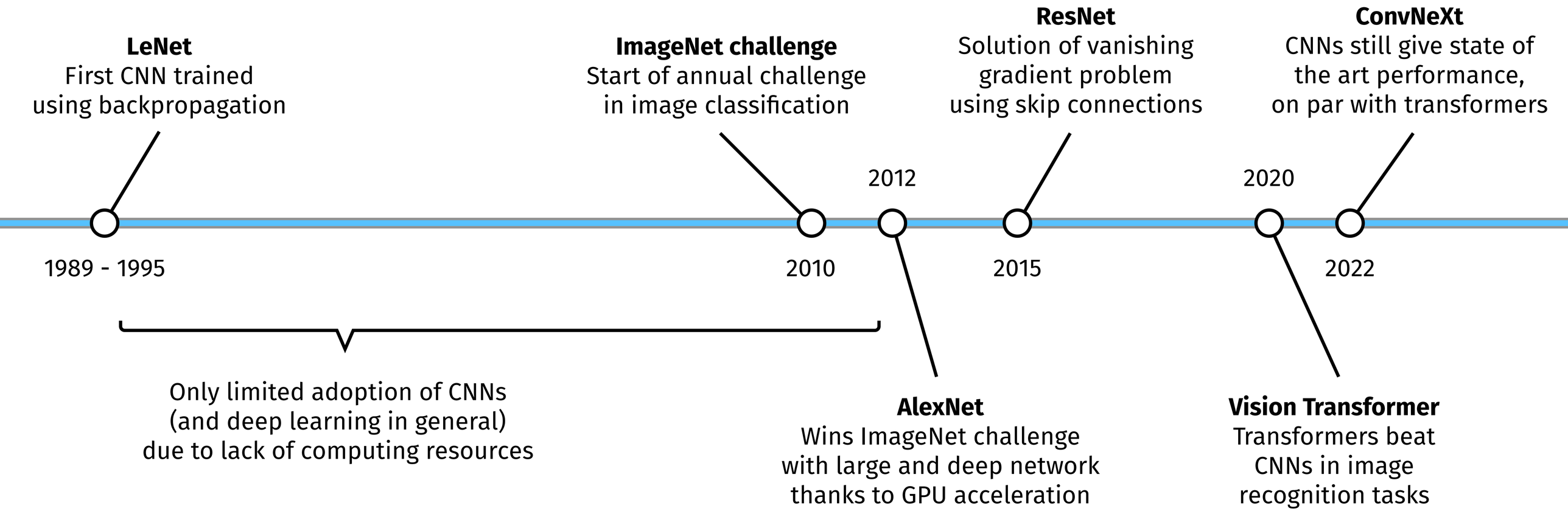


	output size	• ConvNeXt-T
stem	56×56	4×4, 96, stride 4
res2	56×56	$\begin{bmatrix} \text{d7}\times\text{7, 96} \\ \text{1}\times\text{1, 384} \\ \text{1}\times\text{1, 96} \end{bmatrix} \times 3$
res3	28×28	$\begin{bmatrix} \text{d7}\times\text{7, 192} \\ \text{1}\times\text{1, 768} \\ \text{1}\times\text{1, 192} \end{bmatrix} \times 3$
res4	14×14	$\begin{bmatrix} \text{d7}\times\text{7, 384} \\ \text{1}\times\text{1, 1536} \\ \text{1}\times\text{1, 384} \end{bmatrix} \times 9$
res5	7×7	$\begin{bmatrix} \text{d7}\times\text{7, 768} \\ \text{1}\times\text{1, 3072} \\ \text{1}\times\text{1, 768} \end{bmatrix} \times 3$
FLOPs		4.5×10^9
# params.		28.6×10^6

- Modernized ResNet-like architecture
- Pure convolutional network → simple and efficient
- Tuned taking inspiration from transformer-based models
- state of the art performance

Liu, Zhuang, et al. "A convnet for the 2020s", 2022

A short history of CNNs



Outline

① Challenges of image processing

② Introduction to convolutions

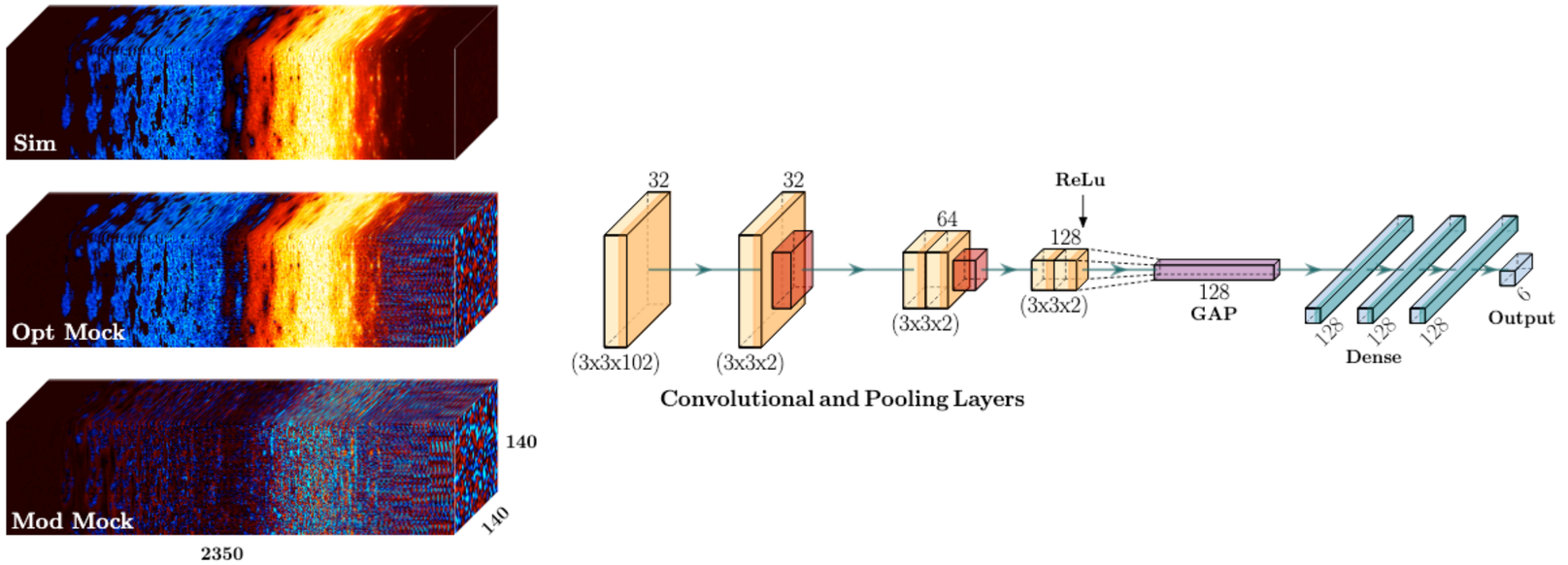
③ Convolutional layers

④ CNN building blocks

⑤ A short history of CNNs

⑥ **Outlook**

Different dimensions

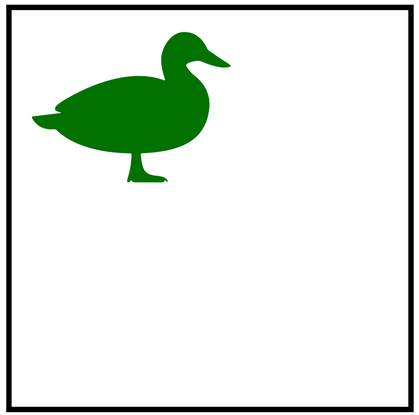


Application in astrophysics: 21cm light-cones [arXiv:2201.07587]

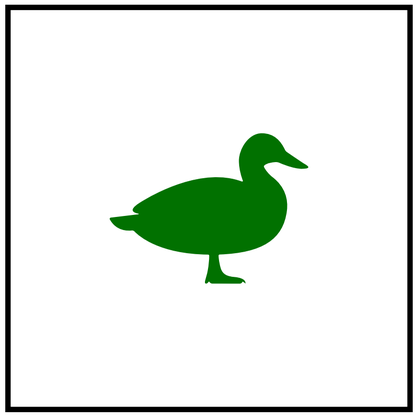
- CNNs are not restricted to two dimensions
- one-dimensional data: time series, …
- three-dimensional data:
  → CT and MRT scans in medicine
  → observations from radio telescopes

Invariance and equivariance

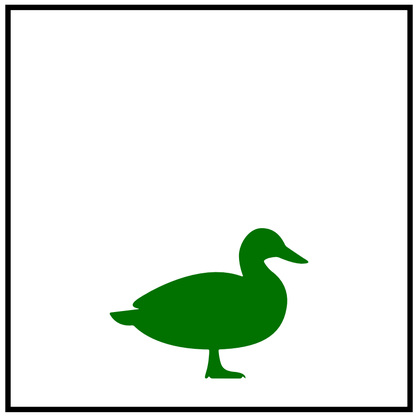
A duck



Also a duck



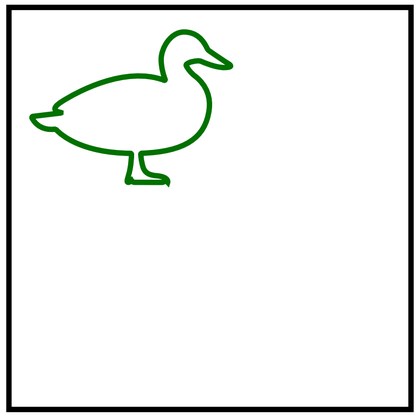
Another duck



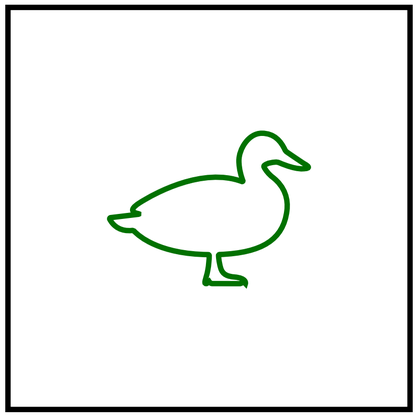
- Main motivation for introducing convolutions:
  want classifier result to be **permutation invariant**

Invariance and equivariance

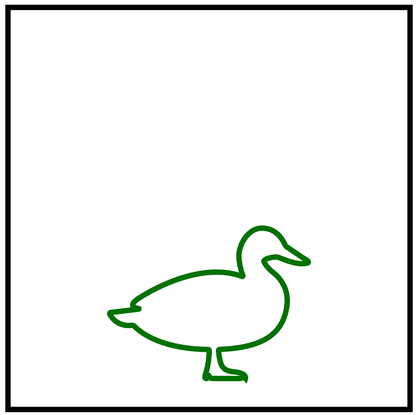
A duck



Also a duck



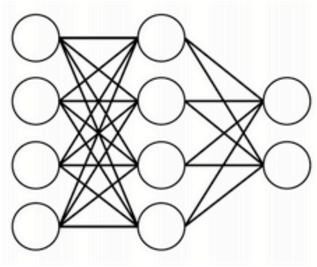
Another duck



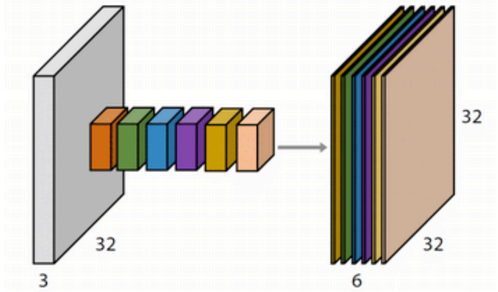
- Main motivation for introducing convolutions:
  want classifier result to be **permutation invariant**
- Solution: CNN layer is **permutation equivariant**
  → transformation acts the same at different positions
- Symmetry of our task leads to specific architecture

Geometric deep learning

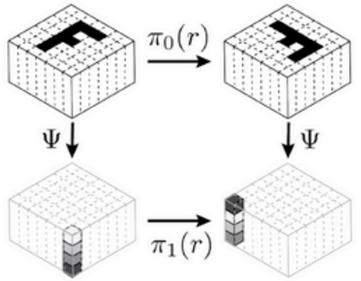
Many network architectures follow from a symmetry!



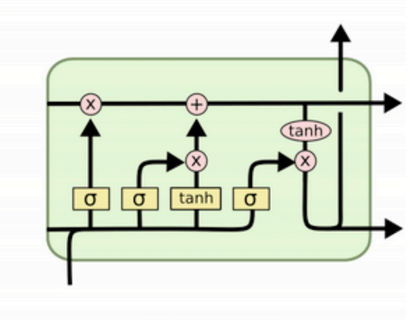
Perceptrons
Function regularity



CNNs
Translation



Group-CNNs
Translation+Rotation,
Global groups



LSTMs
Time warping

DeepSets / Transformers
Permutation

GNNs
Permutation

Intrinsic CNNs
Isometry / Gauge choice

Book and lectures on the topic:
https://geometricdeeplearning.com/

Exercises

Now it’s time for a coffee,

and then some exercises!

You can find the material here:
https://github.com/theoheimel/erum-cnn-2025