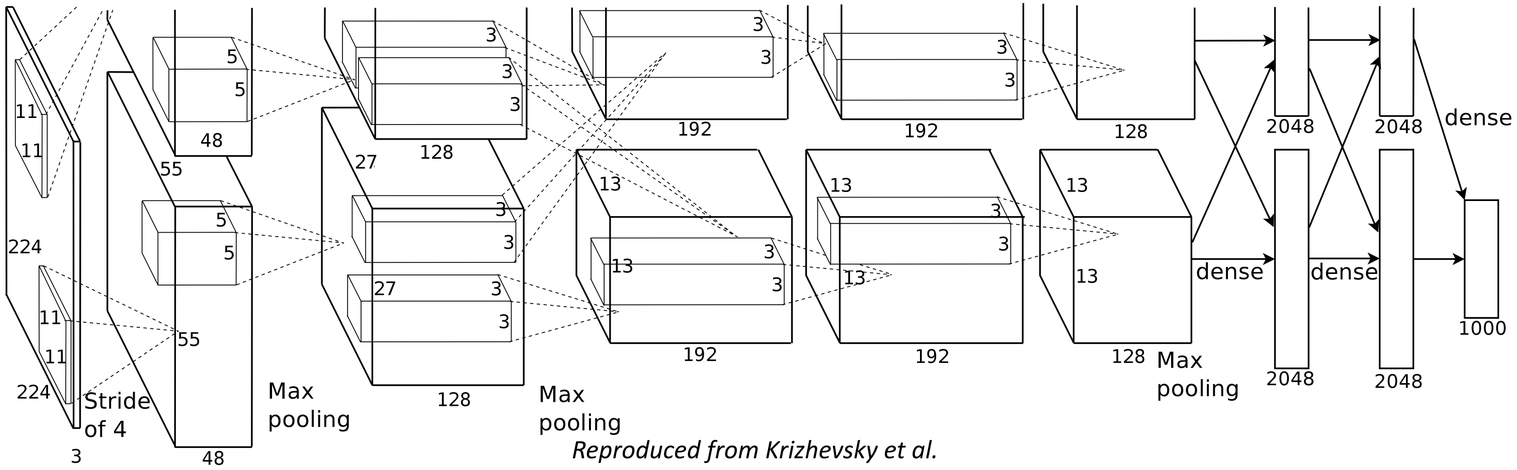
Artificial Intelligence & Data Analysis

Lecture 1: Convolutional Neural Networks



Reproduced from Krizhevsky et al.

Lyon 1

Pierre Marza

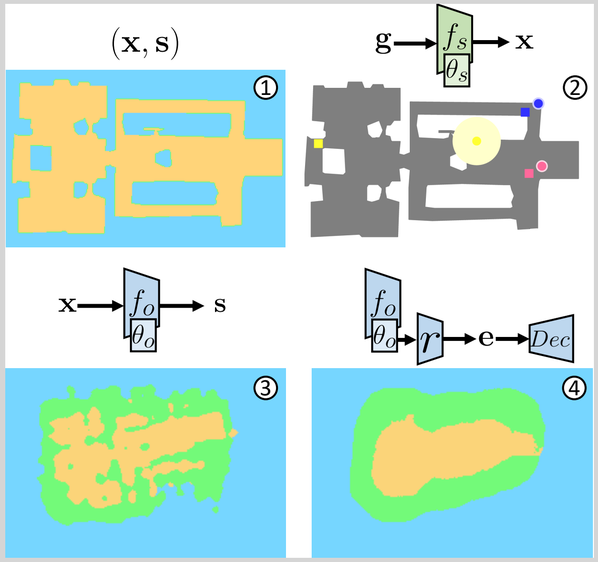
Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NeurIPS 2012

About me

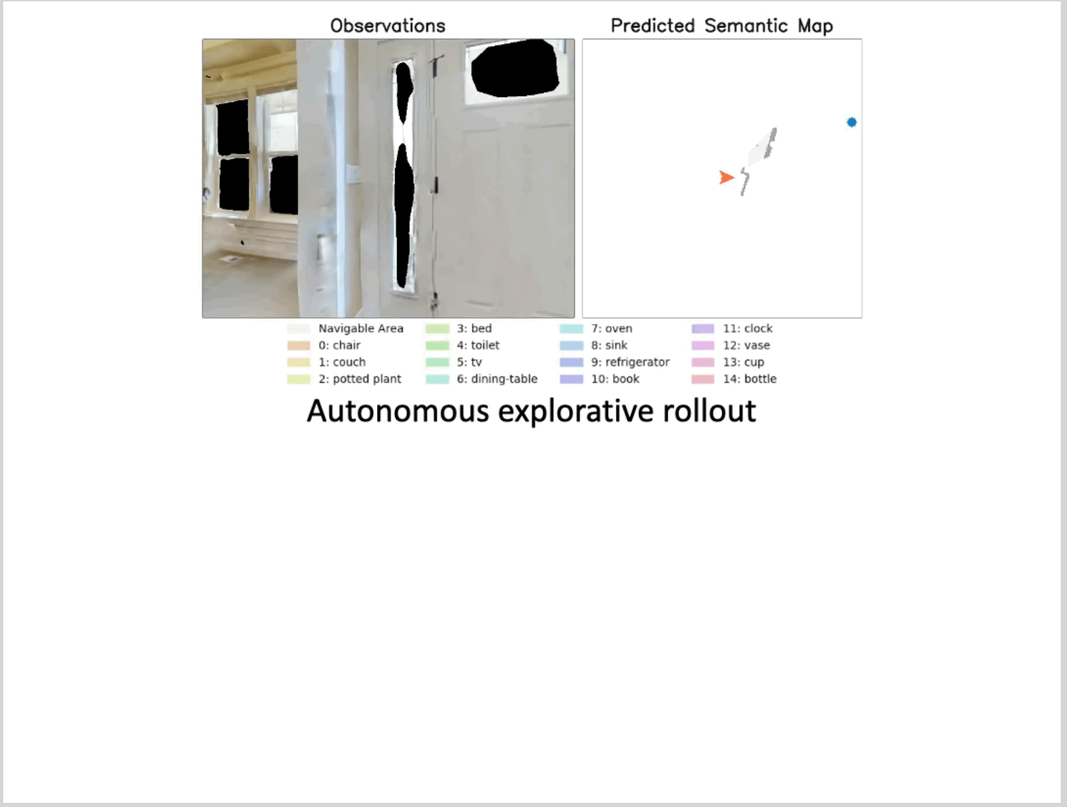
Pierre Marza: PhD student at INSA Lyon

Embodied AI, Visual Navigation, Deep Learning, Computer Vision, Reinforcement Learning

Teaching Agents how to Map: Spatial Reasoning for Multi-Object Navigation



Multi-Object Navigation with dynamically learned neural implicit representations



AutoNeRF: Training Implicit Scene Representations with Autonomous Agents

Course Overview

1. Convolutional Neural Networks (1h lecture + 2h practical)
2. Recurrent Neural Networks (1h lecture + 2h practical)
3. Transformers and Attention (1h lecture + 2h practical)

4. Reinforcement Learning 1 (Johan Peralez)
5. Reinforcement Learning 2 (Johan Peralez)

6. Project (15h)

Evaluation

1. Project oral/written presentation

2. Final exam (Multiple-choice questions)

Useful resources

Ian Goodfellow, Yoshua Bengio, Aaron Courville, **Deep Learning**, 2017, MIT Press (https://www.deeplearningbook.org/)

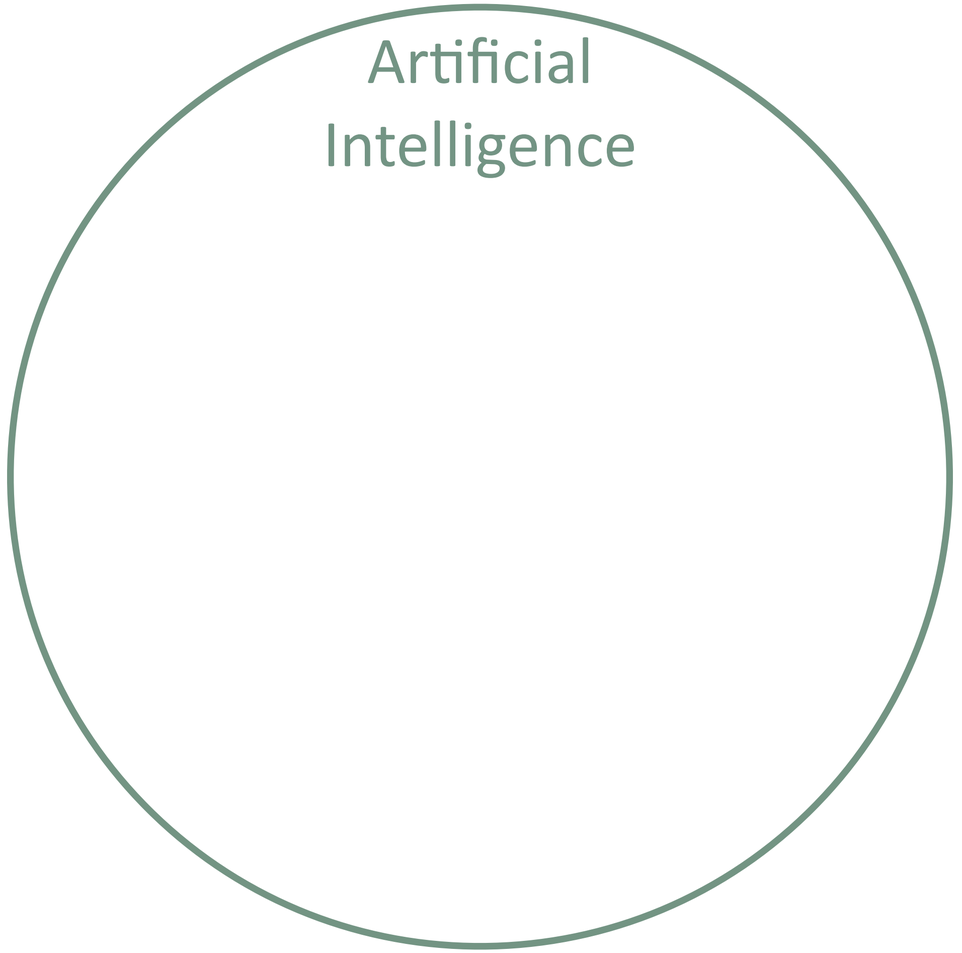
Pytorch Tutorials: https://pytorch.org/tutorials/

Christopher M. Bishop, **Pattern recognition and machine learning**, 2006, Springer (Machine Learning resource, not only Deep Learning)

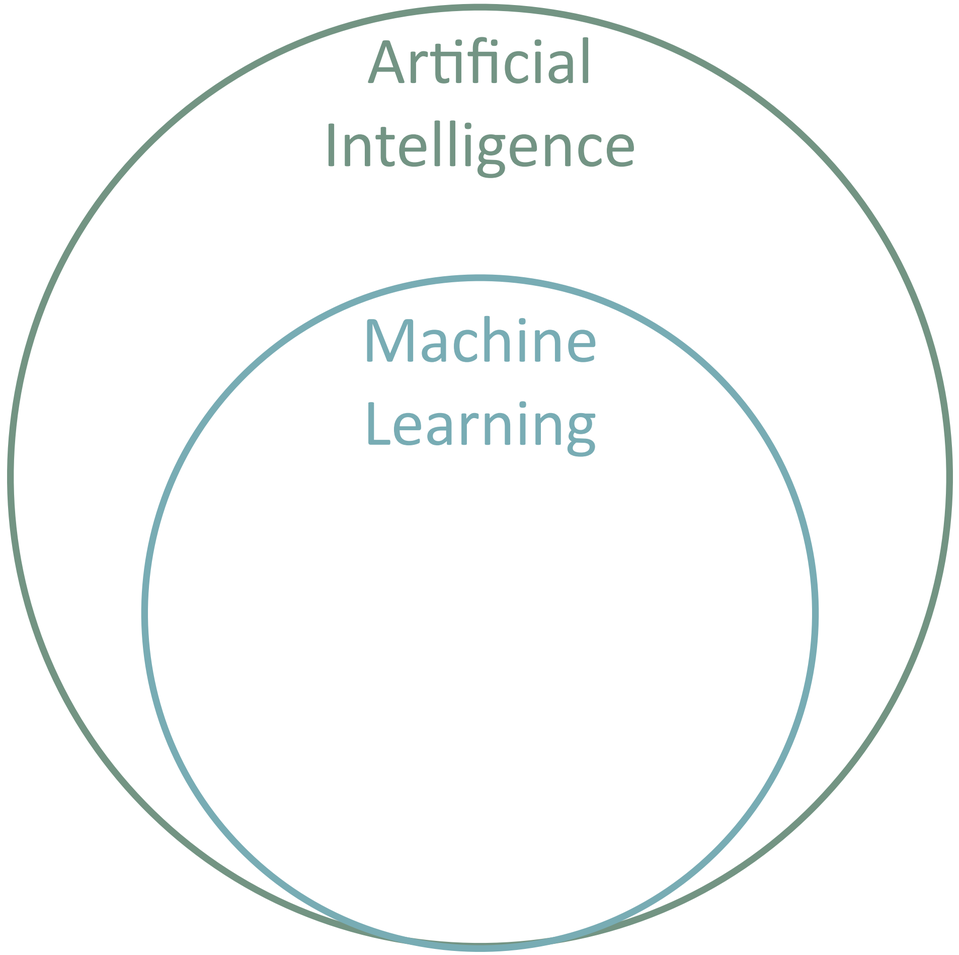
Interesting read: Rich Sutton, **The Bitter Lesson**, 2019 (http://www.incompleteideas.net/IncIdeas/BitterLesson.html)

Some of the material of this course is inspired by the lectures given by Christian Wolf (my PhD advisor) at INSA Lyon. You can find his full course here: https://chriswolfvision.github.io/www/teaching/index.html

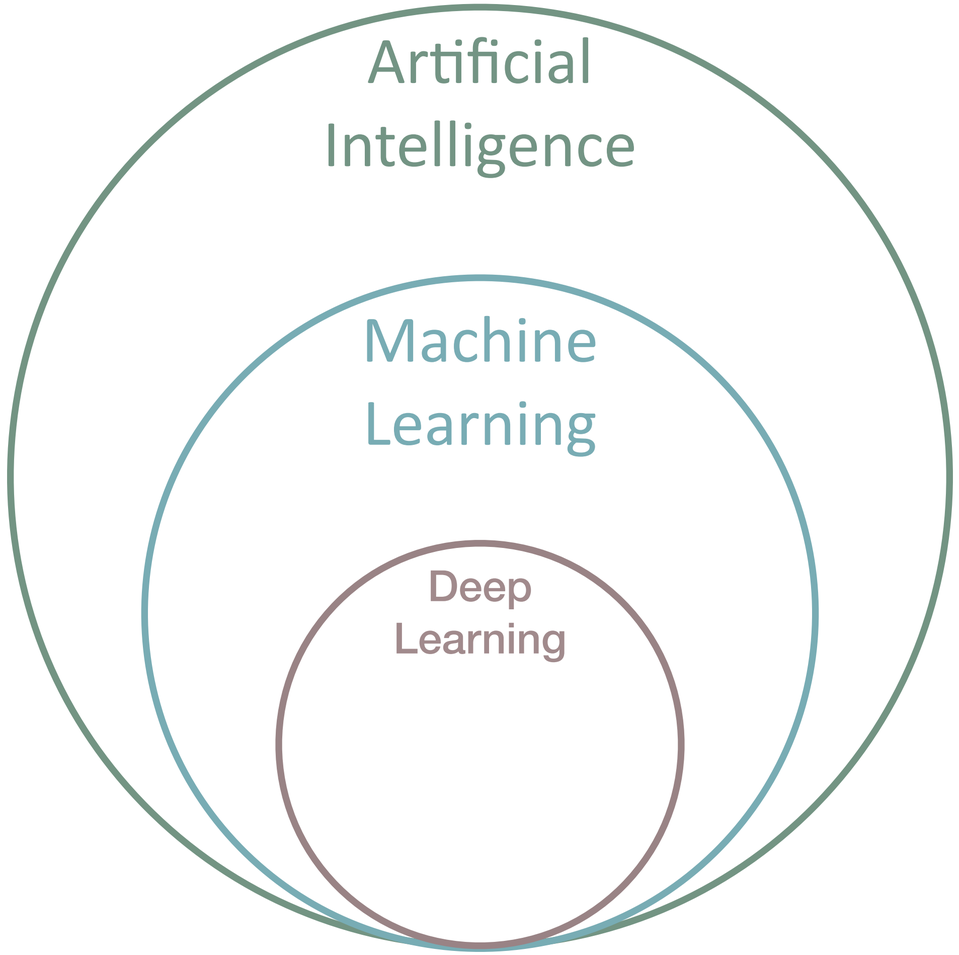
Artificial Intelligence



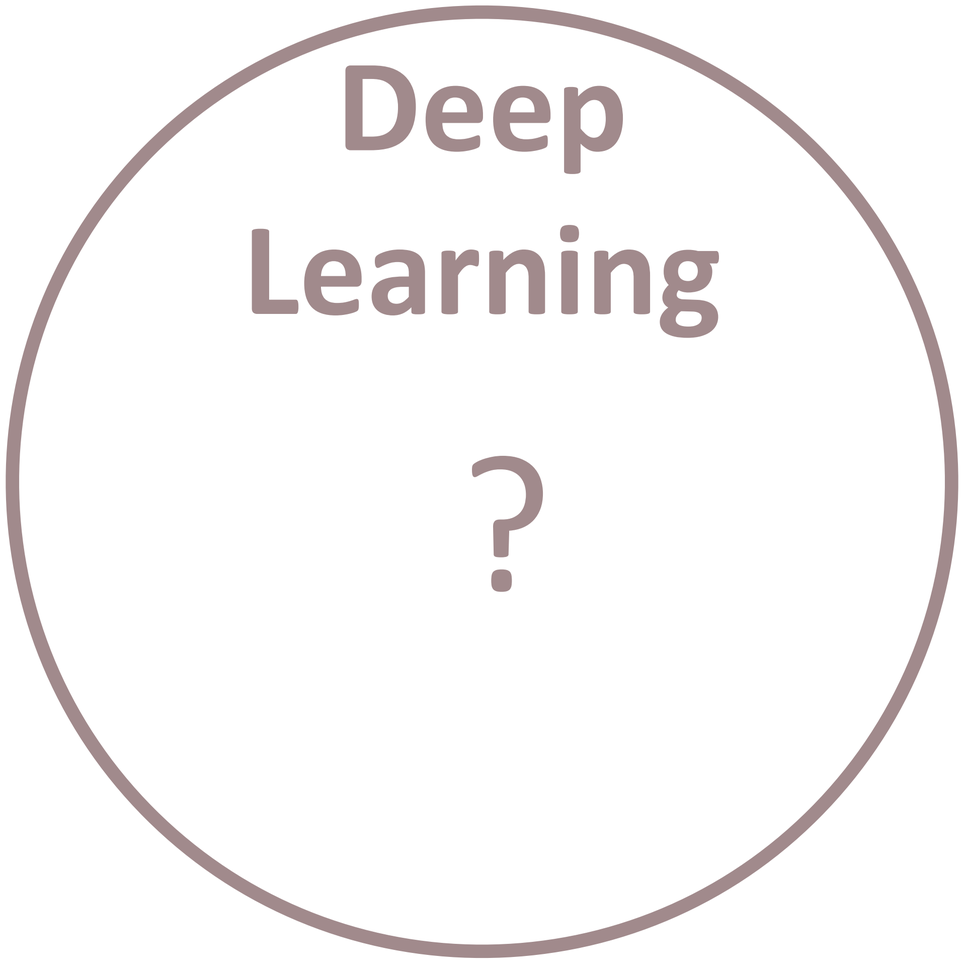
Artificial Intelligence



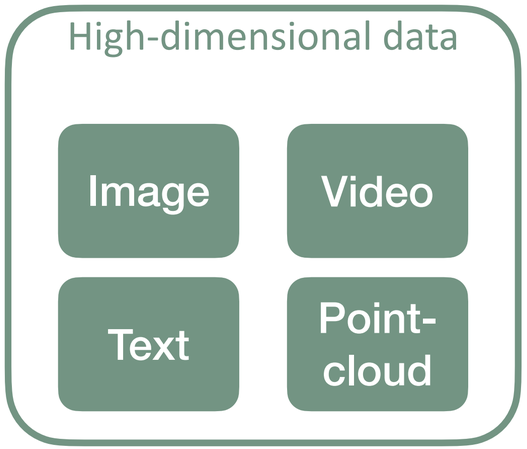
Artificial Intelligence



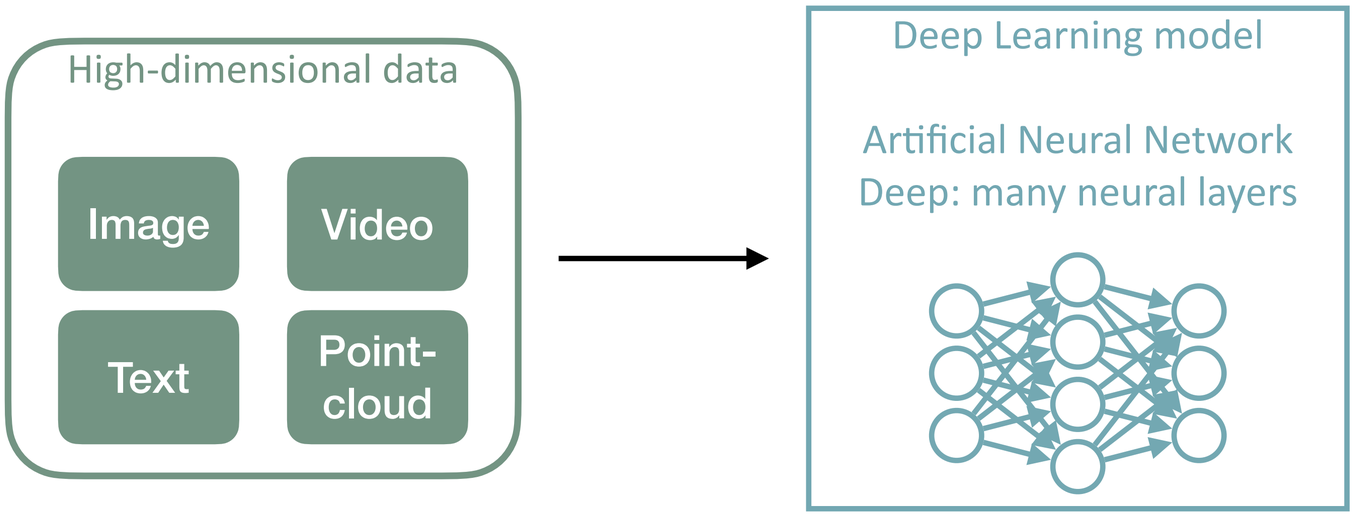
Artificial Intelligence



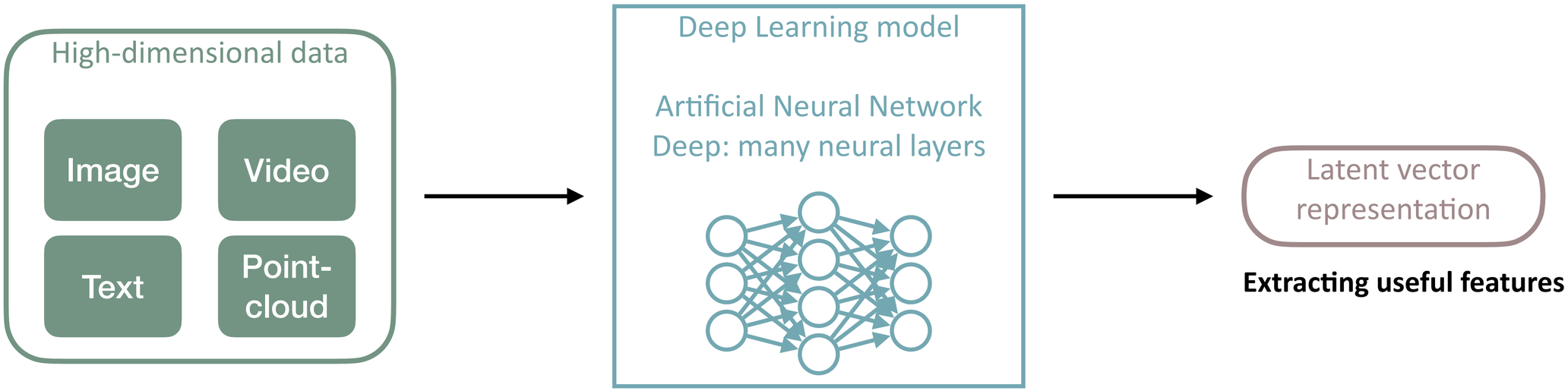
What is Deep Learning?



What is Deep Learning?



What is Deep Learning?

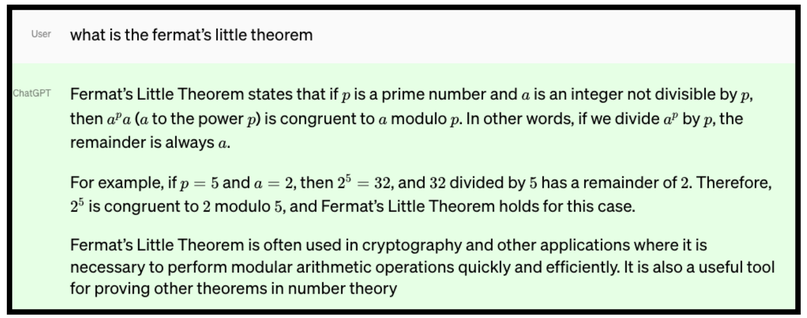


What is Deep Learning?

Image Generation

An astronaut riding a horse in photorealistic style.

(Reproduced from DALLE 2 website)



(Reproduced from ChatGPT website)

Text Generation

Latent vector representation

Classification

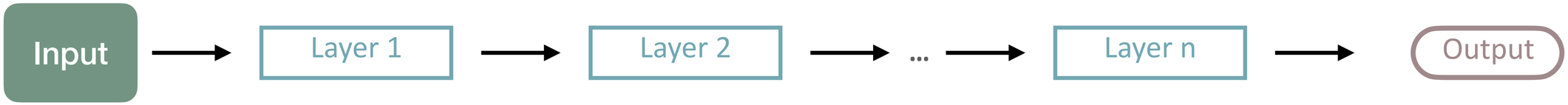
Cat

Dog

Detection

What is Deep Learning?

What is an Artificial Neural Network?

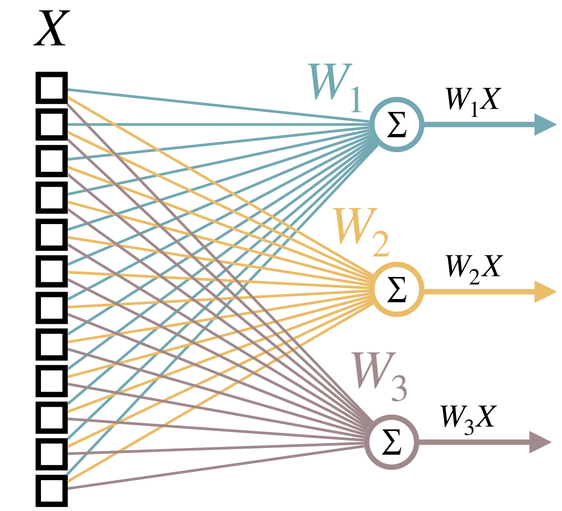


- **Parameters**/Weights: each layer is composed of **neurons** that are **parametrized by weights** that we want to **optimize**. This is the **learning** part.

- **Hyperparameters**:
  - Type and number of layers
  - Number of neurons per layer
  - Activation functions, etc.

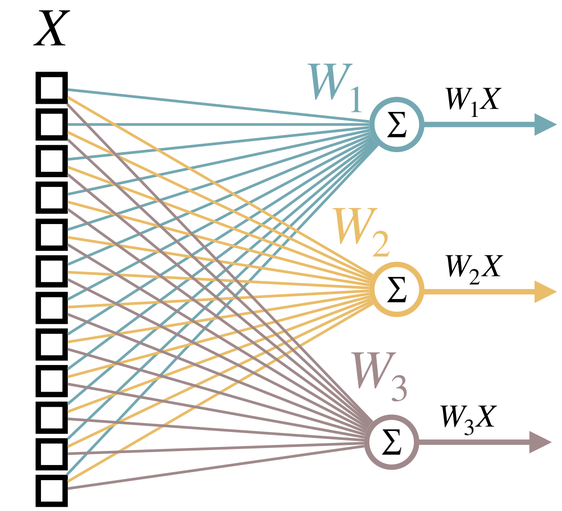
What is Deep Learning?

Recap (?): The Linear Layer



What is Deep Learning?

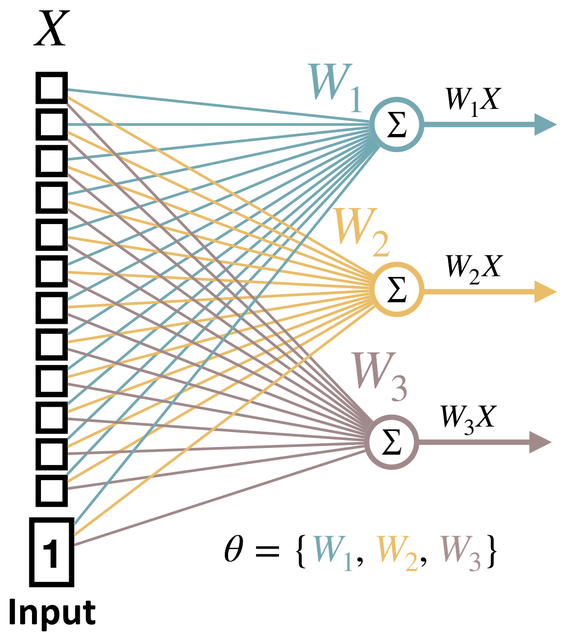
Recap (?): The Linear Layer



But, here we can just learn a **linear** function ($y = ax$) and not an **affine** function ($y = ax + b$) —> We need to **add a bias term**.

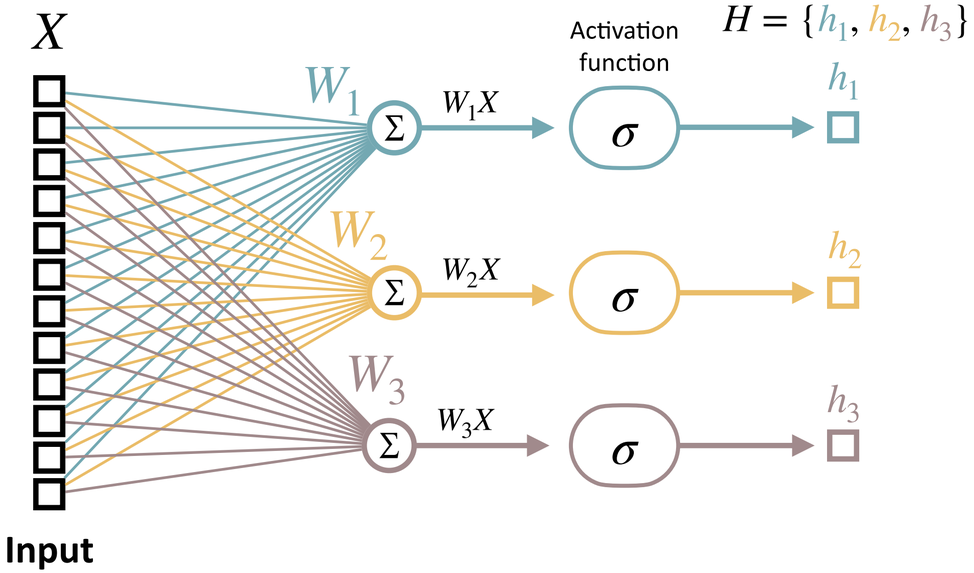
What is Deep Learning?

Recap (?): The Linear Layer



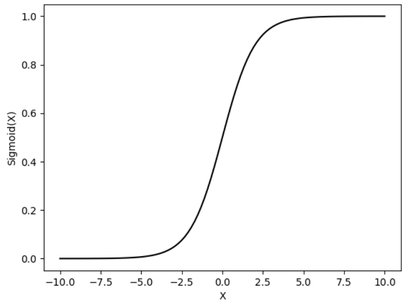
What we often do is that we **add a '1' at the end of the input**.

Recap (?): Activation Functions

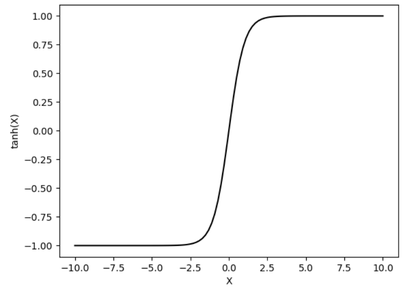


Activation Functions

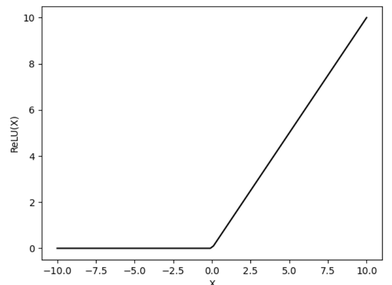
Sigmoid



tanh



ReLU



Recap (?): Loss Functions

To train your neural network, you need to **compare its output with the ground-truth labels**. This is where you need to use a **loss function**.

Example 1: Let's consider you have n training samples and are solving a **regression problem**. For a given (input, ground-truth) pair (I_i, T_i), the output of your neural network is $\hat{t}_i$. Then, you can use the **Mean Squared Error** as your loss function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(T_i - \hat{t}_i\right)^2$$

Recap (?): Loss Functions

To train your neural network, you need to **compare its output with the ground-truth labels**. This is where you need to use a **loss function**.

Example 2: Let's consider you have n training samples and are solving a **classification problem** that involves M classes. For a given (input, ground-truth) pair (I_i, T_i), the output of your neural network is $\hat{t}_i$. More specifically, for a given class c, $T_{i,c}$ is a binary indicator of whether class c is the correct one for sample i, and $\hat{t}_{i,c}$ if the predicted probability for sample i to belong to class c.

Then, you can use the **Cross-Entropy Loss** as your loss function:

$$\mathrm{CE} = -\frac{1}{n}\sum_{i=1}^{N}\sum_{c=1}^{M} T_{i,c}\log\left(\hat{t}_{i,c}\right)$$

Recap (?): Loss Functions

To train your neural network, you need to **compare its output with the ground-truth labels**. This is where you need to use a **loss function**.

Example 2: Let's consider you have n training samples and are solving a **classification problem** that involves M classes. For a given (input, ground-truth) pair (I_i, T_i), the output of your neural network is $\hat{t}_i$. More specifically, for a given class c, $T_{i,c}$ is a binary indicator of whether class c is the correct one for sample i, and $\hat{t}_{i,c}$ if the predicted probability for sample i to belong to class c.

When solving a classification problem, the output of the neural network will generally be a probability distribution over possible classes.

Standard Deep Learning Pipeline

1. **Find a dataset**, i.e. pairs of (input, output)


Cat


Dog


Cat


Dog


Cat


Dog


Cat


Cat


Cat


Dog


Dog


Dog

Standard Deep Learning Pipeline

1. Find a dataset, i.e. pairs of (input, output)
2. **Split your data** into 3 sets: train, validation, test

Train


Cat


Dog


Dog


Cat


Dog


Cat

Validation


Cat


Cat


Dog

Test


Dog


Dog


Cat

Standard Deep Learning Pipeline

1. Find a dataset, i.e. pairs of (input, output)
2. **Split your data** into 3 sets: train, validation, test

Train

To t**rain your neural network**, i.e. optimise its weights/ parameters based on the difference between its predictions and ground-truth outputs.

Validation

To evaluate the choice of hyper parameters, e.g. number of layers, neurons, type of activations, loss functions…
This set is very important: allows to evaluate the performance on data not used during training but different from the final test set (to avoid choosing hyper params that only work on this specific test set).

Test

To evaluate the final performance of you neural network (after hyper params have been chosen and weight values optimised)

Standard Deep Learning Pipeline

1. Find a dataset, i.e. pairs of (input, output)
2. Split your data into 3 sets: train, validation, test
3. **Build your neural network** (this process is often iterative based on validation performance).
    1. Types of layers (linear, convolutional, recurrent, etc.)
    2. Activation functions
    3. Number of layers, neurons, etc.

Standard Deep Learning Pipeline

1. Find a dataset, i.e. pairs of (input, output)
2. Split your data into 3 sets: train, validation, test
3. Build your neural network (this process is often iterative based on validation performance).
4. **Train your neural network**

Stochastic Gradient Descent (SGD)

Epoch: a full iteration over your train set.

Training batch: a set of samples from the train set.

f_θ: neural network parametrised by weights θ.

$\mathscr{L}$: chosen loss function (e.g. Cross-Entropy for classification problems).

Iterate over n epochs

  Iterate over m training batches

    Sampling training batch $\mathscr{B}_i = \{I_i, T_i\}$ from train set, where I_i and T_i are inputs and ground-truth outputs respectively.

    Forward pass: $\hat{t}_i = f_\theta(I_i; \theta)$, where $\hat{t}_i$ is the prediction from the neural network.

    Error computation: $e_i = \mathscr{L}(\hat{t}_i, , T_i)$ where e_i is the error between the predictions and the ground truth.

    Backward pass: Computing gradient of e_i with respect to network weights θ, and updating θ.

Course Overview

1. Convolutional Neural Networks (1h lecture + 2h practical)
2. Recurrent Neural Networks (1h lecture + 2h practical)
3. Transformers and Attention (1h lecture + 2h practical)

4. Reinforcement Learning 1 (Johan Peralez)
5. Reinforcement Learning 2 (Johan Peralez)

6. Project (15h)

Course Overview

1. Convolutional Neural Networks (1h lecture + 2h practical)
2. Recurrent Neural Networks (1h lecture + 2h practical)
3. Transformers and Attention (1h lecture + 2h practical)

4. Reinforcement Learning 1 (Johan Peralez)
5. Reinforcement Learning 2 (Johan Peralez)

6. Project (15h)

Useful resources about CNNs

- AlexNet paper (winner of the Imagenet competition): Krizhevsky et al. ImageNet Classification with Deep Convolutional Neural Networks, NeurIPS 2012

- ResNet paper (a method that is still used a lot in Computer Vision): He et al., Deep Residual Learning for Image Recognition, CVPR 2016

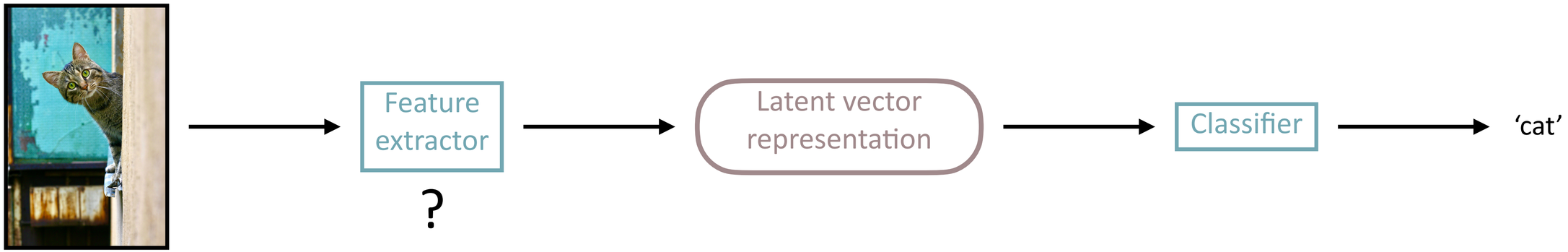
- A nice blog post from my PhD advisor (Christian Wolf) about shift equivariance: https://chriswolfvision.medium.com/what-is-translation-equivariance-and-why-do-we-use-convolutions-to-get-it-6f18139d4c59

Convolutional Neural Networks — Motivation

How to extract a feature representation of an image?

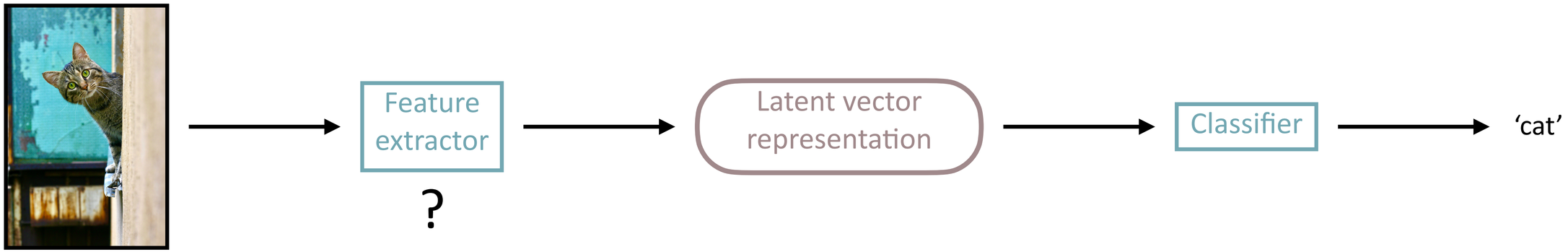
Convolutional Neural Networks — Motivation

How to extract a feature representation of an image?



Convolutional Neural Networks — Motivation

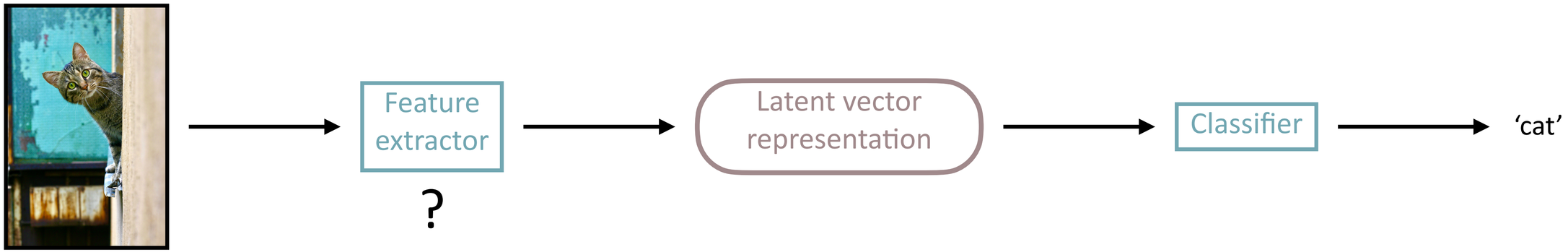
How to extract a feature representation of an image?



Before Deep Learning: SIFT, HoG features, etc.

Convolutional Neural Networks — Motivation

How to extract a feature representation of an image?



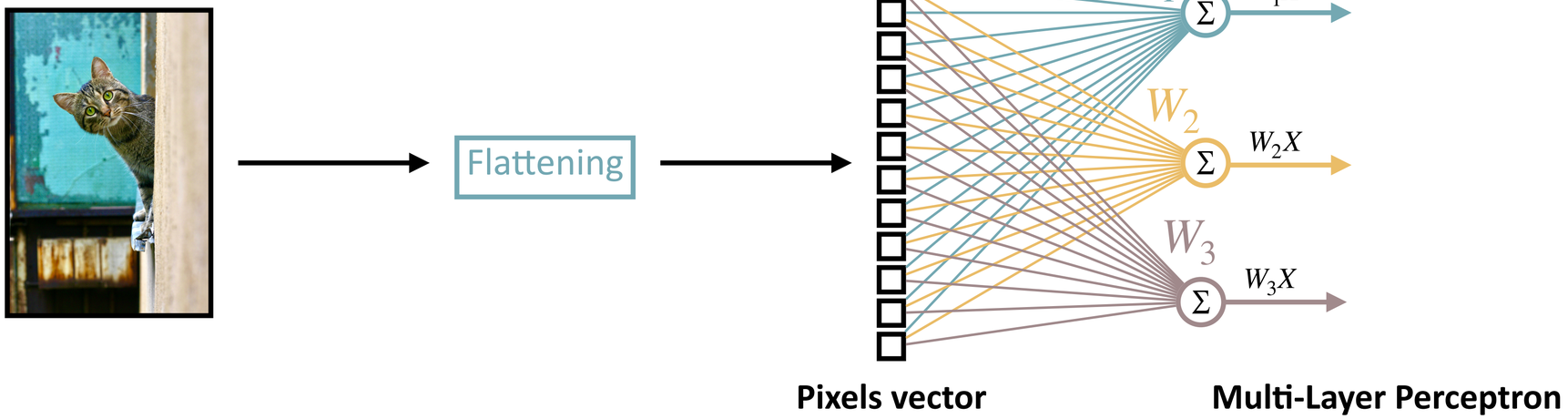
Before Deep Learning: SIFT, HoG features, etc.

Artificial Neural Network: Couldn't we use a simple Multi-Layer Perceptron?

Convolutional Neural Networks — Motivation

Artificial Neural Network: Couldn't we use a simple Multi-Layer Perceptron?

Answer: Maybe, but it can be hard to train… even sometimes impossible…



Pixels vector **Multi-Layer Perceptron**

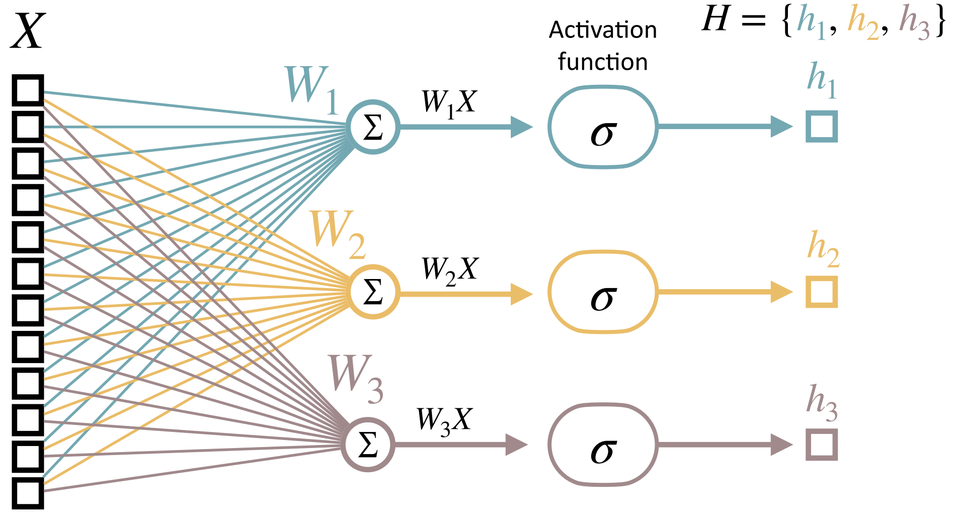
$\theta = \{W_1, W_2, W_3\}$

Convolutional Neural Networks — Motivation

Artificial Neural Network: Couldn't we use a simple Multi-Layer Perceptron?

Answer: Maybe, but it can be hard to train… even sometimes impossible…

Flattening



Pixels vector

Multi-Layer Perceptron

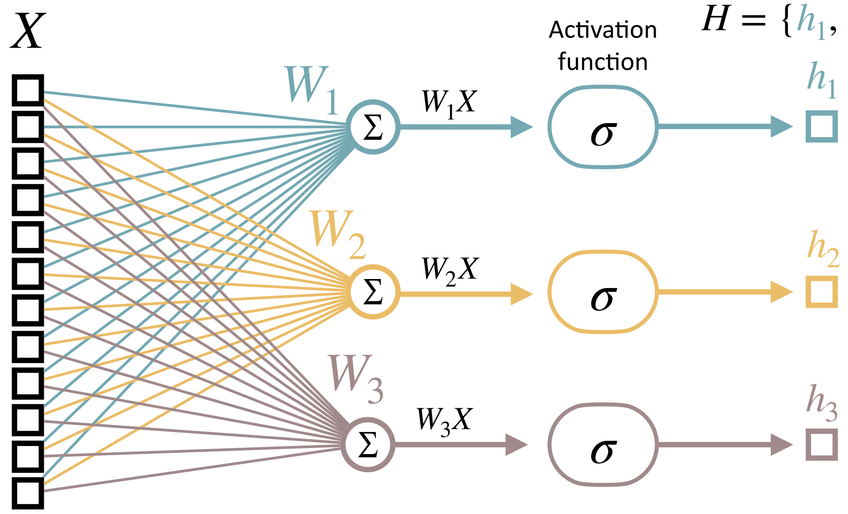
$\theta = \{W_1, W_2, W_3\}$

Convolutional Neural Networks — Motivation

Artificial Neural Network: Couldn't we use a simple Multi-Layer Perceptron?

Answer: Maybe, but it can be hard to train… even sometimes impossible…

Flattening



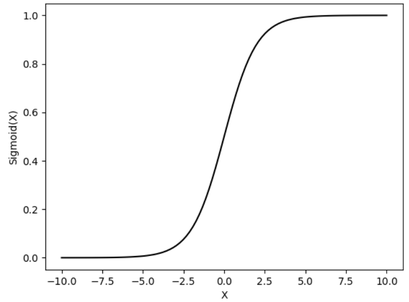
Pixels vector

Multi-Layer Perceptron

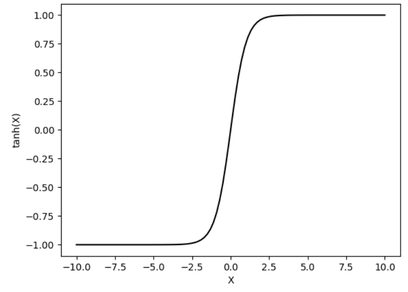
$\theta = \{W_1, W_2, W_3\}$

Activation Functions

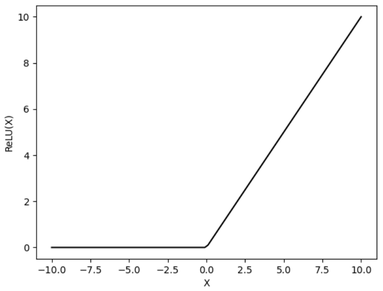
Sigmoid



tanh



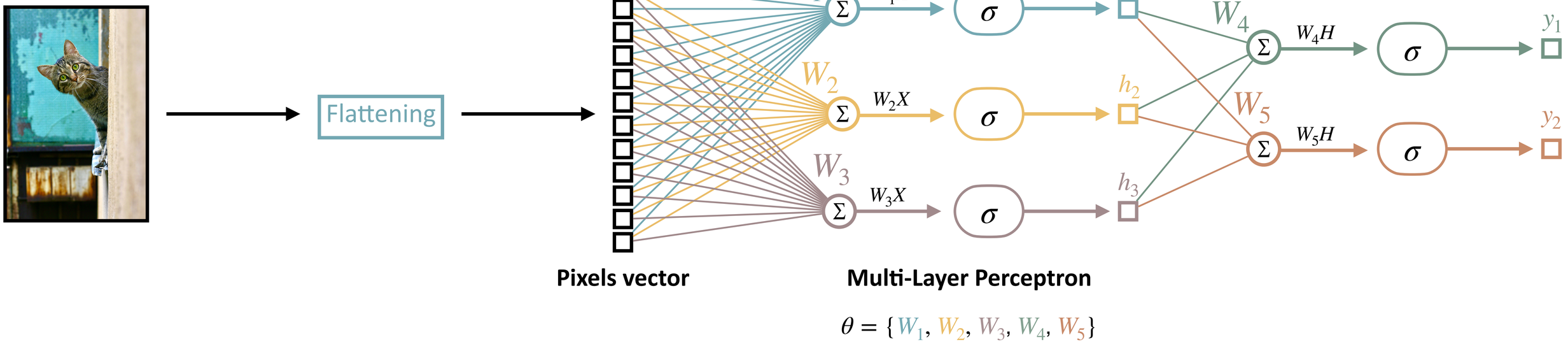
ReLU



Convolutional Neural Networks — Motivation

Artificial Neural Network: Couldn't we use a simple Multi-Layer Perceptron?

Answer: Maybe, but it can be hard to train… even sometimes impossible…



$\theta = \{W_1, W_2, W_3, W_4, W_5\}$

Convolutional Neural Networks — Motivation

Issues with MLP

- Spatial information is lost when flattening the image. We loose information about whether a given input pixel was close to others in the original image.

Convolutional Neural Networks — Motivation

Issues with MLP

- Spatial information is lost when flattening the image. We loose information about whether a given input pixel was close to others in the original image.

- The number of parameters to learn quickly grows with the size of the input image.

If we have a **224*224*3** input image —> **150528 input values**

Now let's say we want **a single linear layer** with **256 neurons**
—> 256*(150528 + 1) = 38535424 ~ **38M parameters for a single layer!**

Bias term

Convolutional Neural Networks — Motivation

Issues with MLP

- Spatial information is lost when flattening the image. We loose information about whether a given input pixel was close to others in the original image.

- The number of parameters to learn quickly grows with the size of the input image.

- It is not translation invariant (outputs a different representation for a given input image and its shifted version)

Convolutional Neural Networks — Motivation

Inductive biases:

- The set of assumptions about the target function to approximate.
- A way to constrain the search space for suitable functions.
- Incorporates prior knowledge about the task to solve.

Convolutional Neural Networks — Motivation

Inductive biases:

- The set of assumptions about the target function to approximate.
- A way to constrain the search space for suitable functions.
- Incorporates prior knowledge about the task to solve.

There are 2 important inductive biases when dealing with images:

- Locality: pixels spatially close share information
- Translation equivariance

Convolutional Neural Networks — Motivation

An operation ϕ is known to be **shift/translation equivariant** if, given an input X and a translation operator T, we have the following:

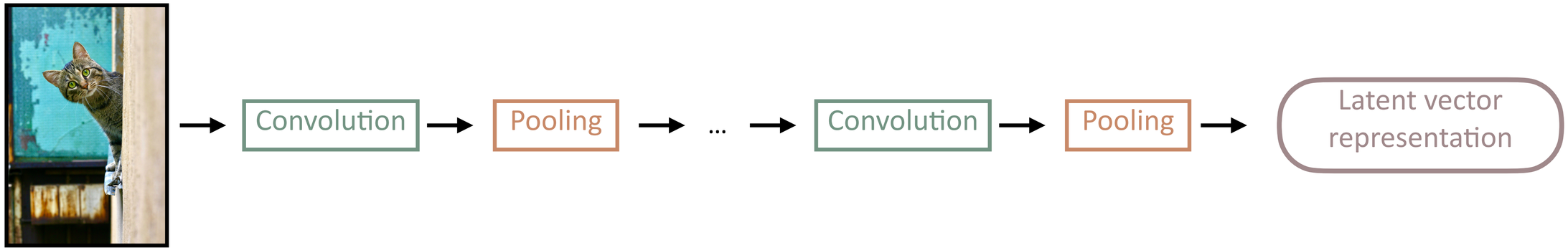
$$\phi(T(X)) = T(\phi(X))$$

Why is it interesting? If we consider shifting an image slightly, we might want the final pixel-wise representation of this image to be the same, just translated by the applied shift.

Convolutional Neural Networks — Standard Architecture

Two important operations:

1. Convolution
2. Pooling



Convolutional Neural Networks — Convolution

Let's consider an input tensor with size $H \times W \times C$,

- H is the height ($H = 5$ in the example below)
- W is the width ($W = 5$ in the example below)
- C is the number of channels ($C = 3$ in the example below as is the case for an RGB image)

Input tensor
$(H = 5, W = 5, C = 3)$

Channel 0 (e.g. R in RGB)

4	112	34	20	5
55	40	1	8	27
70	87	2	190	25
240	101	67	42	12
44	109	4	1	87

Channel 1 (e.g. G in RGB)

25	37	115	60	88
177	24	37	92	145
15	50	108	124	44
5	71	133	74	58
27	39	82	118	243

Channel 2 (e.g. B in RGB)

12	250	194	145	117
37	26	129	107	99
87	75	60	191	230
20	11	48	61	72
87	33	71	48	100

Convolutional Neural Networks — Convolution

Let's consider a convolution kernel with size $H \times W \times C$,

- H is the height ($H = 3$ in the example below)
- W is the width ($W = 3$ in the example below)
- C is the number of channels ($C = 3$ in the example below to match input channels)

Input tensor
$(H = 5, W = 5, C = 3)$

Convolution kernel
$(H = 3, W = 3, C = 3)$

Channel 0 (e.g. R in RGB)

4	112	34	20	5
55	40	1	8	27
70	87	2	190	25
240	101	67	42	12
44	109	4	1	87

3	2	4
0	1	8
2	7	6

Channel 1 (e.g. G in RGB)

25	37	115	60	88
177	24	37	92	145
15	50	108	124	44
5	71	133	74	58
27	39	82	118	243

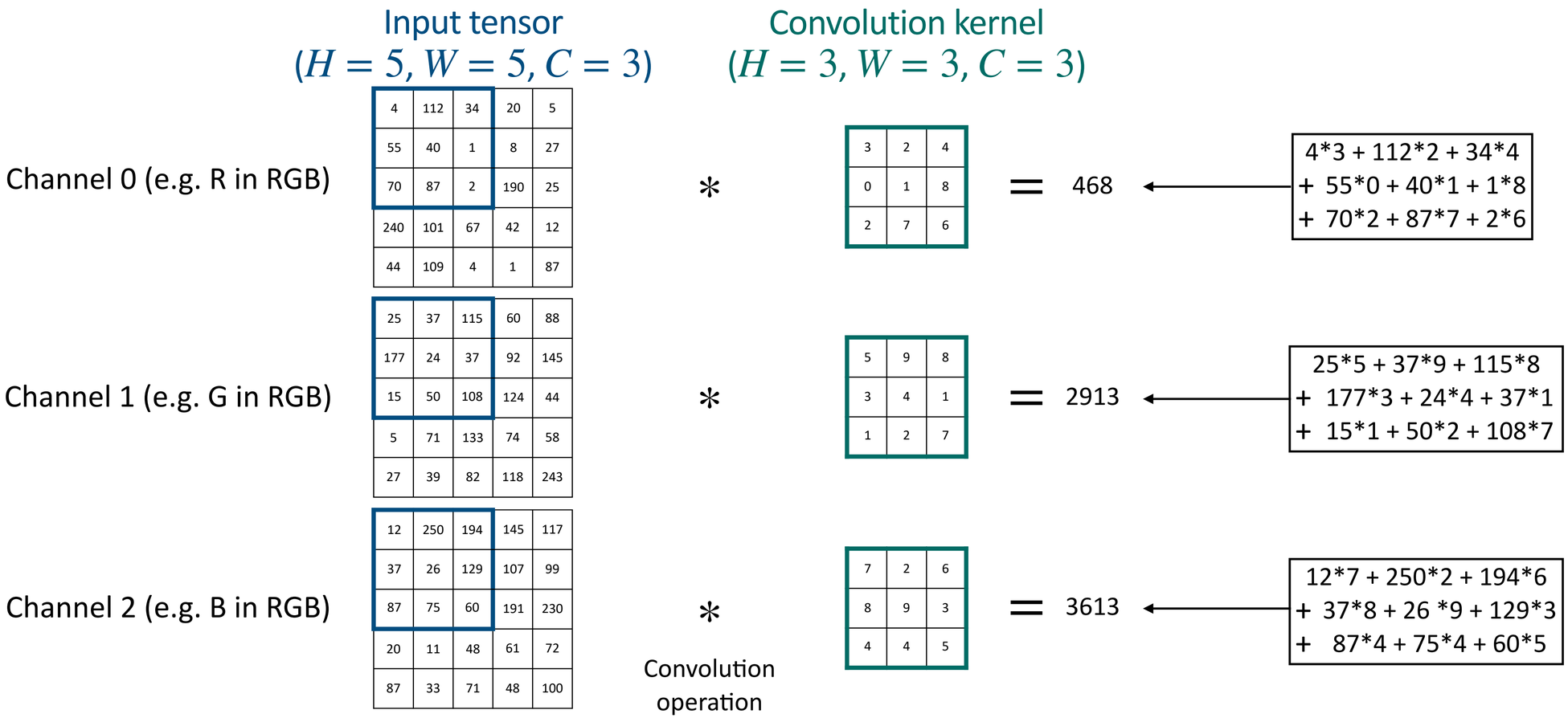
5	9	8
3	4	1
1	2	7

Channel 2 (e.g. B in RGB)

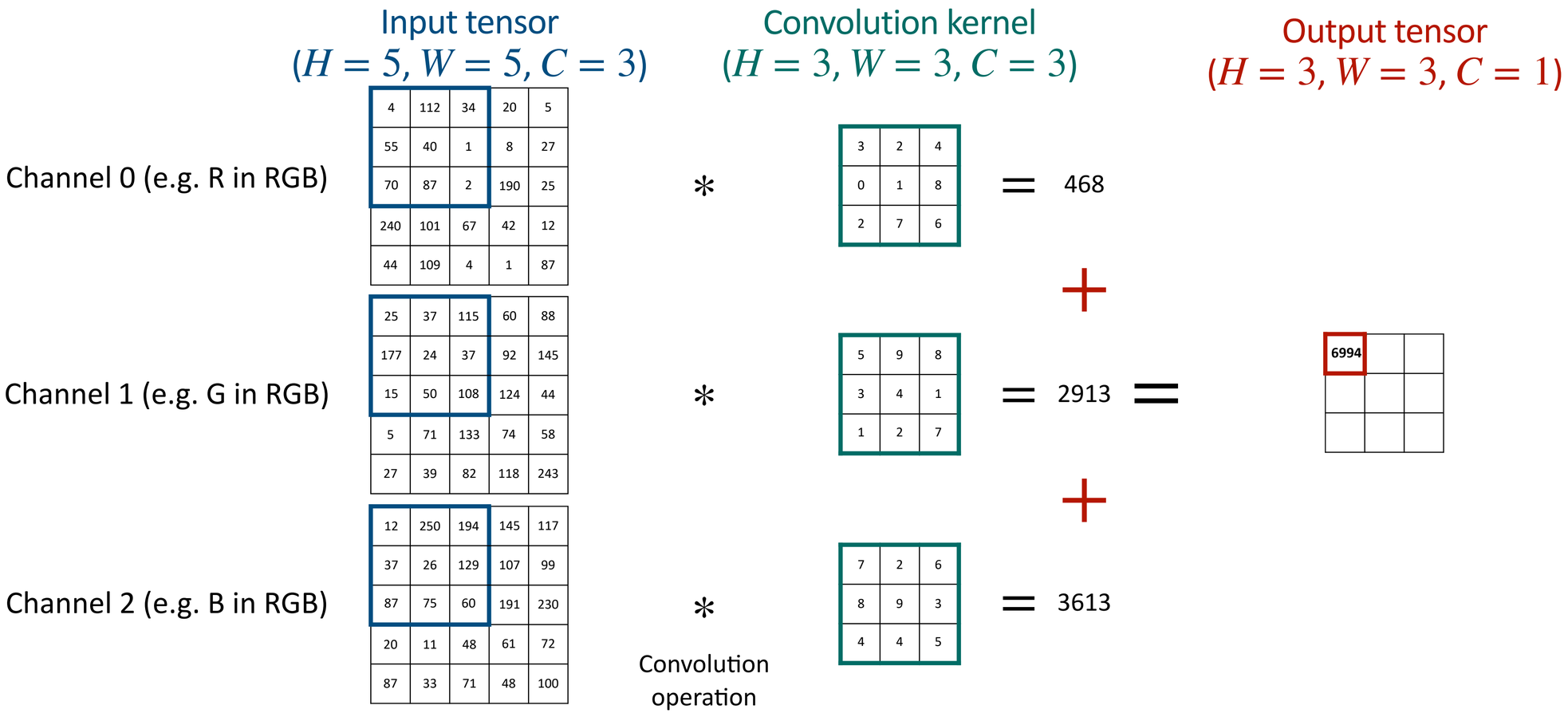
12	250	194	145	117
37	26	129	107	99
87	75	60	191	230
20	11	48	61	72
87	33	71	48	100

7	2	6
8	9	3
4	4	5

Convolutional Neural Networks — Convolution



Convolutional Neural Networks — Convolution



Convolutional Neural Networks — Convolution

Input tensor
$(H = 5, W = 5, C = 3)$

Convolution kernel
$(H = 3, W = 3, C = 3)$

Output tensor
$(H = 3, W = 3, C = 1)$

Channel 0 (e.g. R in RGB)

4	112	34	20	5
55	40	1	8	27
70	87	2	190	25
240	101	67	42	12
44	109	4	1	87

$*$

3	2	4
0	1	8
2	7	6

$= 468$

$+$

Channel 1 (e.g. G in RGB)

25	37	115	60	88
177	24	37	92	145
15	50	108	124	44
5	71	133	74	58
27	39	82	118	243

$*$

5	9	8
3	4	1
1	2	7

$= 2913$

$=$

$+$

Channel 2 (e.g. B in RGB)

12	250	194	145	117
37	26	129	107	99
87	75	60	191	230
20	11	48	61	72
87	33	71	48	100

$*$

7	2	6
8	9	3
4	4	5

$= 3613$

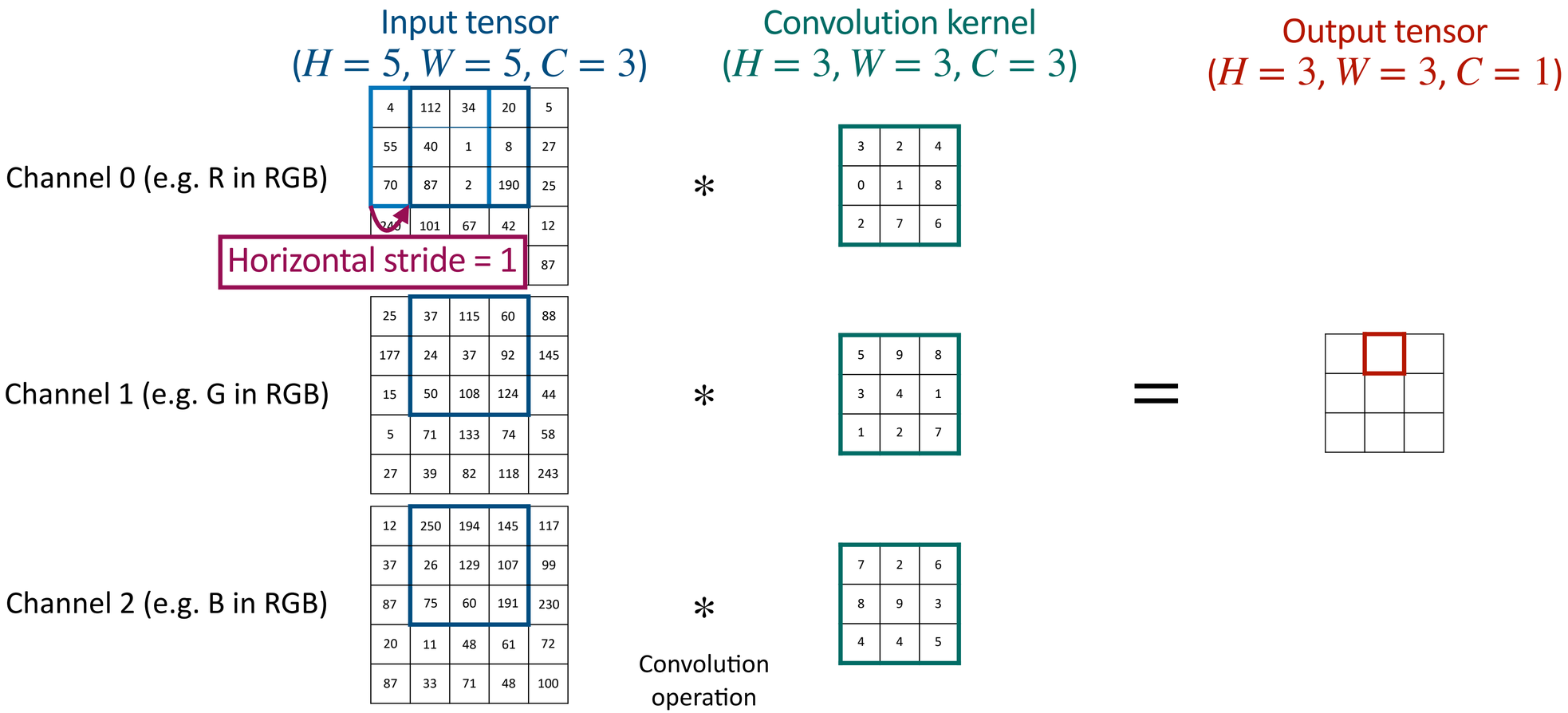
Convolution operation

6994		

We will call such tensor a **feature map**

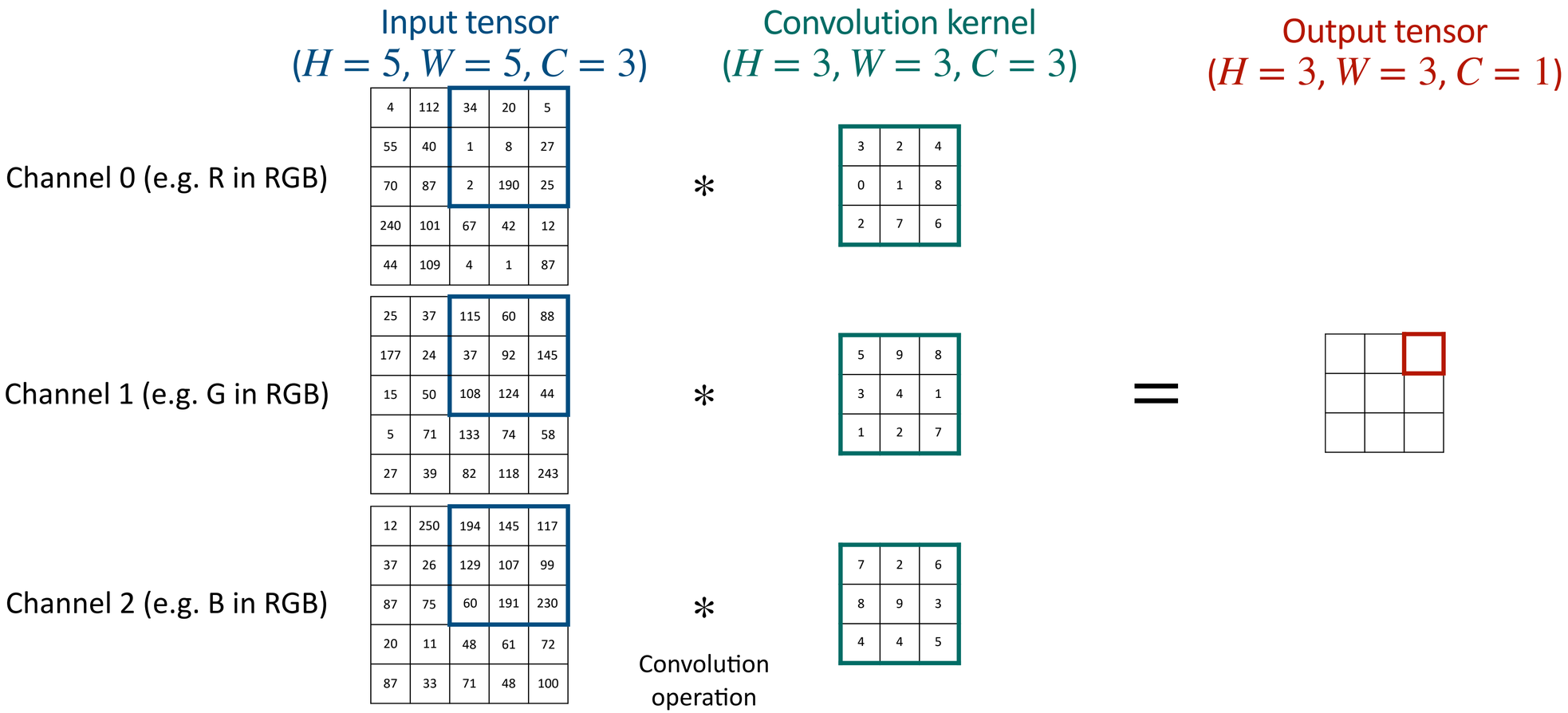
Convolutional Neural Networks — Convolution

Additional to the kernel size ($H \times W \times C$), there is another important hypeparamater of the convolution kernel: the stride, i.e. **by how much to slide the kernel**. In the example below, horizontal and vertical strides are set to 1.



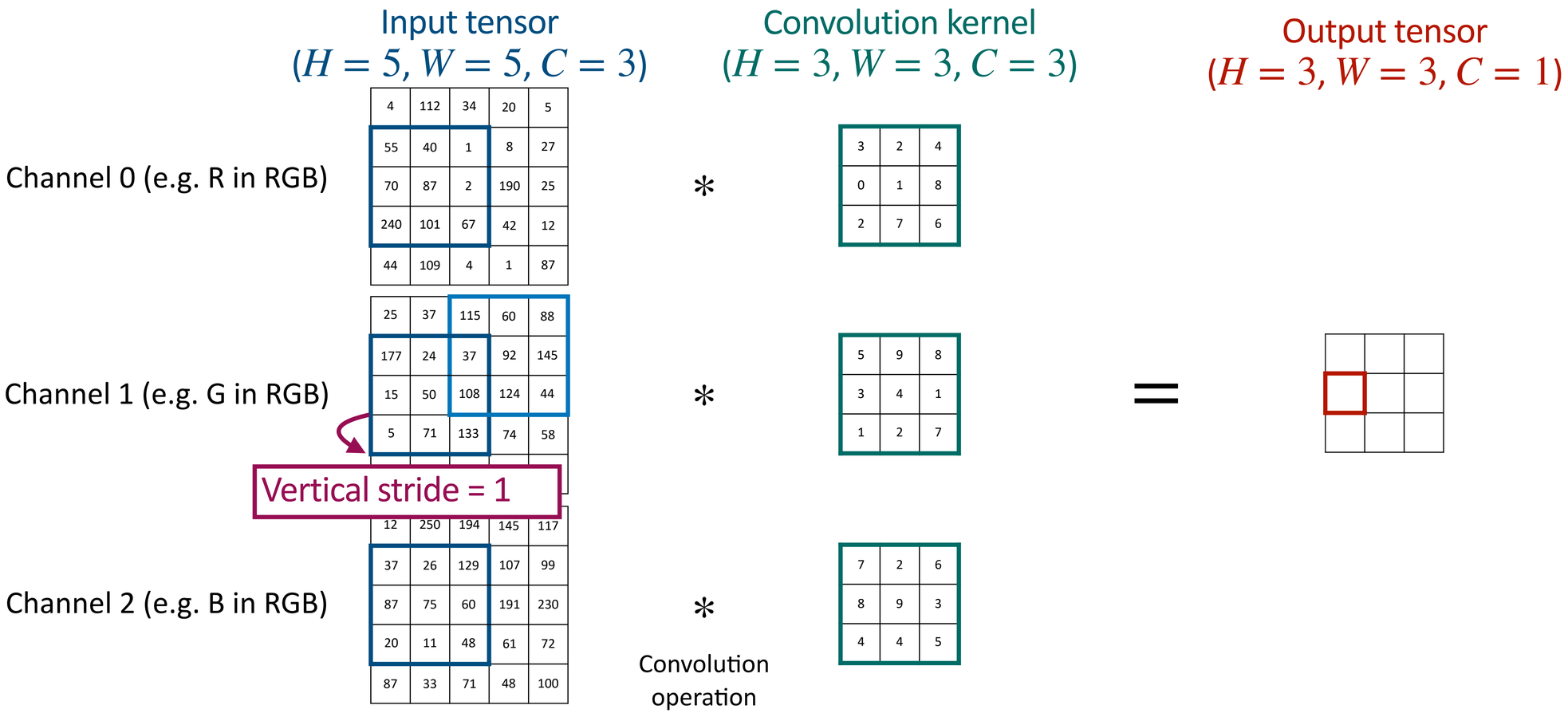
Convolutional Neural Networks — Convolution

Additional to the kernel size ($H \times W \times C$), there is another important hypeparamater of the convolution kernel: the stride, i.e. **by how much to slide the kernel**. In the example below, horizontal and vertical strides are set to 1.



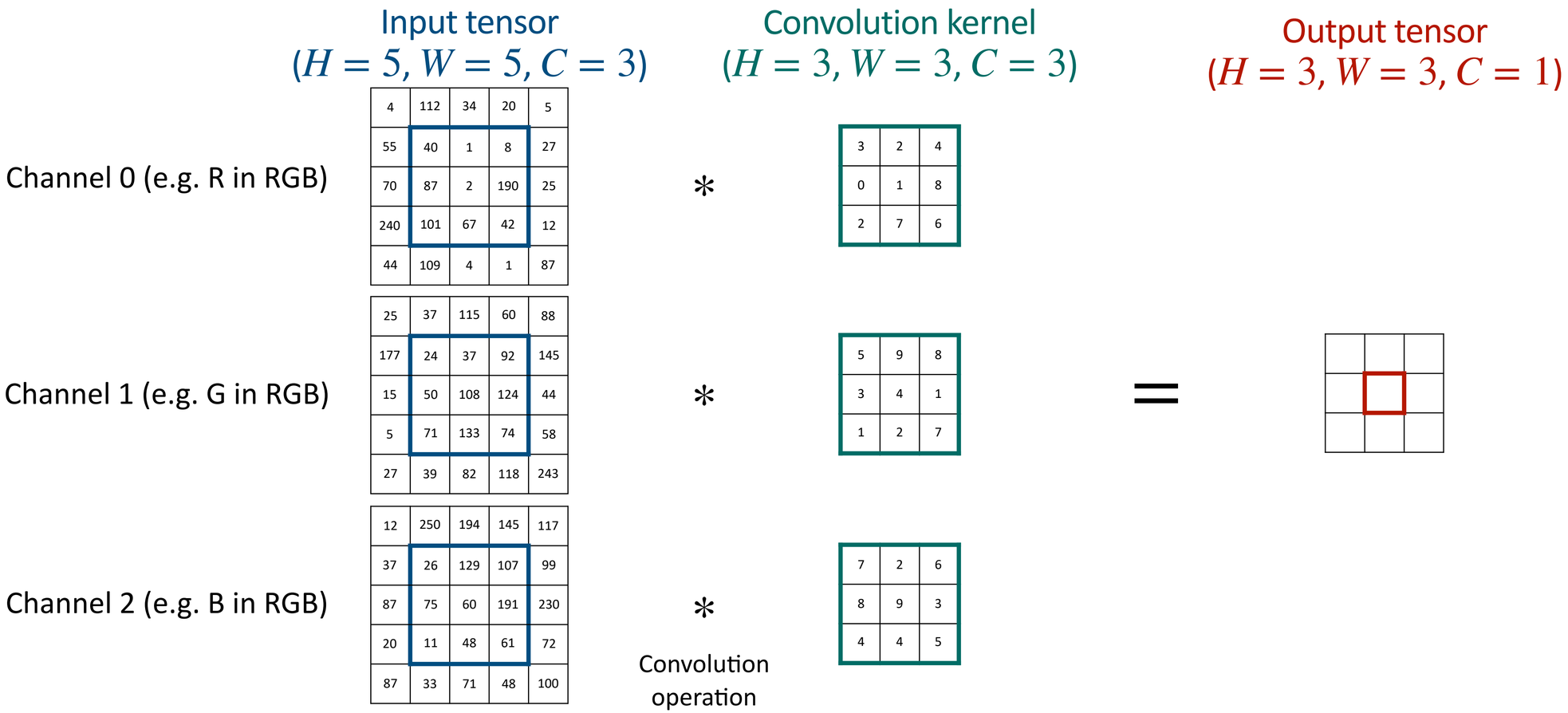
Convolutional Neural Networks — Convolution

Additional to the kernel size ($H \times W \times C$), there is another important hypeparamater of the convolution kernel: the stride, i.e. **by how much to slide the kernel**. In the example below, horizontal and vertical strides are set to 1.



Convolutional Neural Networks — Convolution

Additional to the kernel size ($H \times W \times C$), there is another important hypeparamater of the convolution kernel: the stride, i.e. **by how much to slide the kernel**. In the example below, horizontal and vertical strides are set to 1.



Convolutional Neural Networks — Convolution

Another hyper parameter of the convolution layer is **padding.** Sometimes you might want to **pad the input tensor with a border of zeros**, e.g. to avoid border values in the input to be less important in the final result. The width of this border is called the **padding size**.

Input tensor

$(H = 5, W = 5, C = 3)$

Channel 0 (e.g. R in RGB)

4	112	34	20	5
55	40	1	8	27
70	87	2	190	25
240	101	67	42	12
44	109	4	1	87

Channel 1 (e.g. G in RGB)

25	37	115	60	88
177	24	37	92	145
15	50	108	124	44
5	71	133	74	58
27	39	82	118	243

Channel 2 (e.g. B in RGB)

12	250	194	145	117
37	26	129	107	99
87	75	60	191	230
20	11	48	61	72
87	33	71	48	100

→ Padding (padding size = 1) →

0	**0**	**0**	**0**	**0**	**0**	**0**
0	4	112	34	20	5	**0**
0	55	40	1	8	27	**0**
0	70	87	2	190	25	**0**
0	240	101	67	42	12	**0**
0	44	109	4	1	87	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

0	**0**	**0**	**0**	**0**	**0**	**0**
0	25	37	115	60	88	**0**
0	177	24	37	92	145	**0**
0	15	50	108	124	44	**0**
0	5	71	133	74	58	**0**
0	27	39	82	118	243	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

0	**0**	**0**	**0**	**0**	**0**	**0**
0	12	250	194	145	117	**0**
0	37	26	129	107	99	**0**
0	87	75	60	191	230	**0**
0	20	11	48	61	72	**0**
0	87	33	71	48	100	**0**
0	**0**	**0**	**0**	**0**	**0**	**0**

Convolutional Neural Networks — Convolution

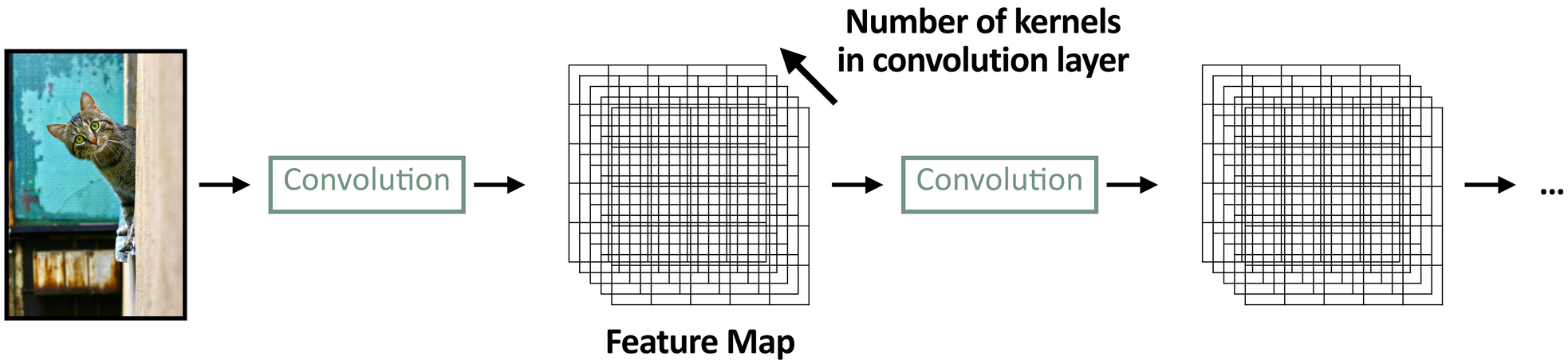
- **All weights** in a convolutional kernel are **learnt**.

- CNN is **parameter efficient**: weights are **shared across the whole input** (only sliding the kernel with fixed weights). **Much better than our MLP!**

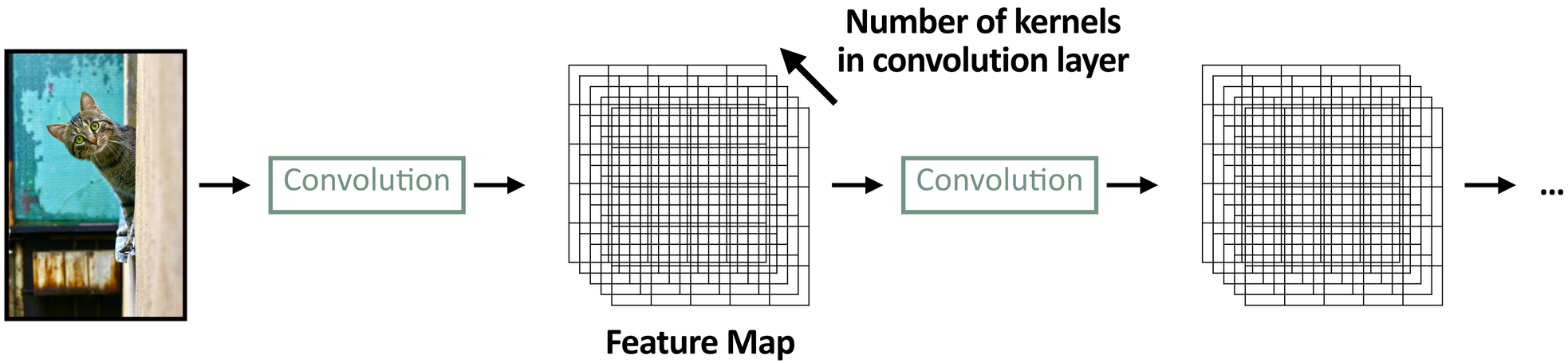
Convolutional Neural Networks — Convolution

- **All weights** in a convolutional kernel are **learnt**.
- CNN is **parameter efficient**: weights are **shared across the whole input** (only sliding the kernel with fixed weights). **Much better than our MLP!**
- One kernel provides one scalar for each position in the output tensor.
- To increase the capacity of a convolutional layer, **we learn several kernels** (output channel dimension is equal to the number of kernels).

Convolutional Neural Networks — Convolution



Convolutional Neural Networks — Convolution



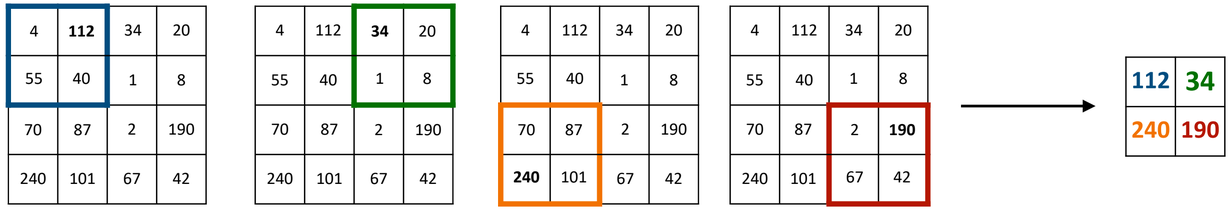
But, CNNs are not only a sequence of convolution layers! There is another operation involved: **pooling**.

Convolutional Neural Networks — Pooling

- The pooling operation allows to **reduce the spatial resolution of a tensor**.
- This is a **non-learned operation**, i.e. not parametrized by any learnable weight.

- Sliding a kernel as for the convolution operation, but instead of computing the weighted sum of inputs, it takes:
  - The max value (Max Pooling)
  - The min value (Min Pooling)
  - The average value (Mean Pooling)

Example of Max Pooling Operation with **kernel size** 2×2 and **stride** 2 (both horizontal and vertical strides):



Convolutional Neural Networks — Pooling

Why do we need pooling?

1. **Makes the network less sensitive to small shifts** in the input by discarding some non-important details. Pooling provides a form of **local shift invariance**.

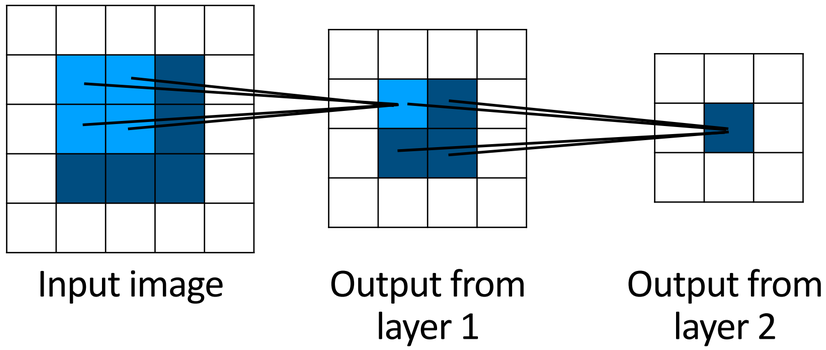
2. With lower resolution maps, **convolution computations are faster**.

3. **Increases the size of the receptive field** of each neuron.

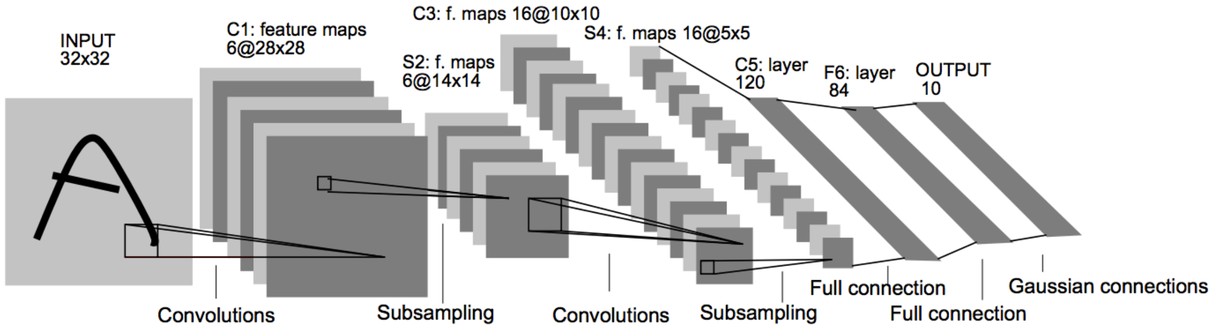
Convolutional Neural Networks — Pooling

The **receptive field of a neuron** is the **part of the input image that brings information to the given neuron.**

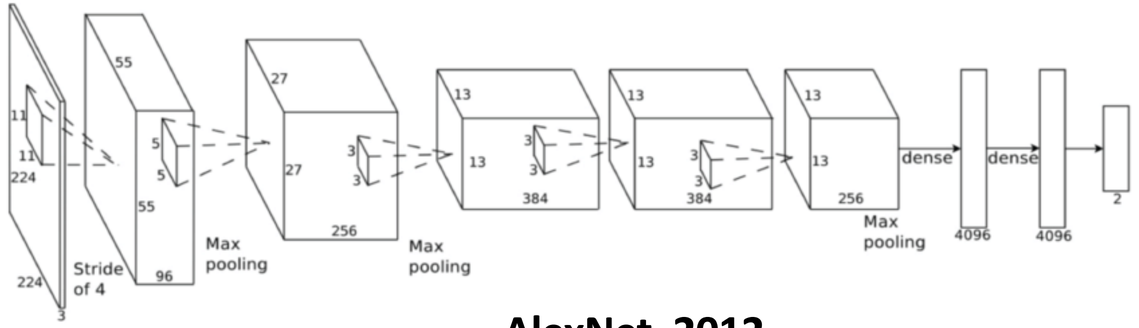
By applying a pooling operation, i.e. downsampling a feature map, we increase the receptive field of next neurons.



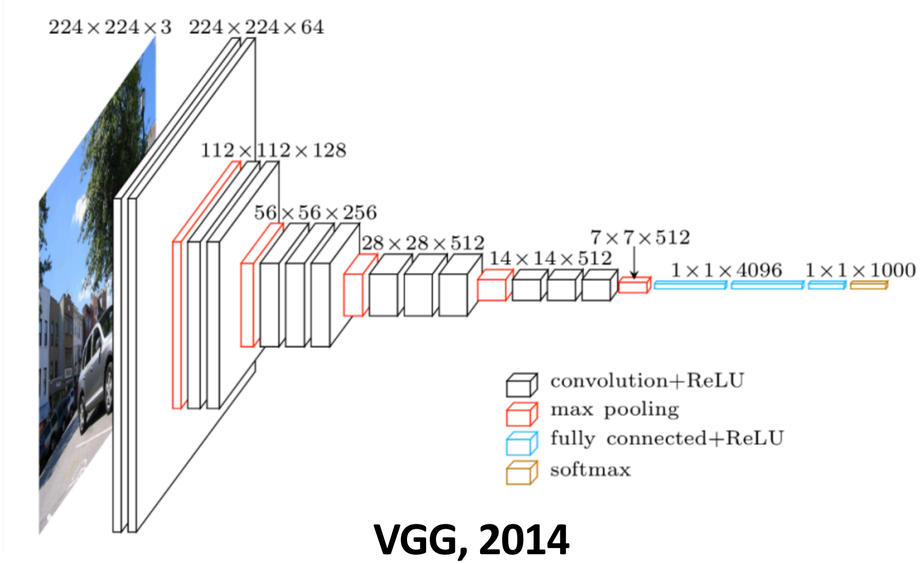
Convolutional Neural Networks — Standard Architectures



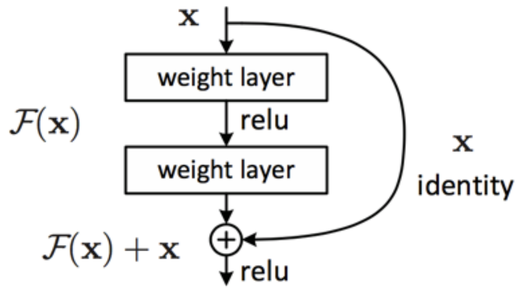
LeNet, 1998



AlexNet, 2012
Winning entry in the Imagenet Competition



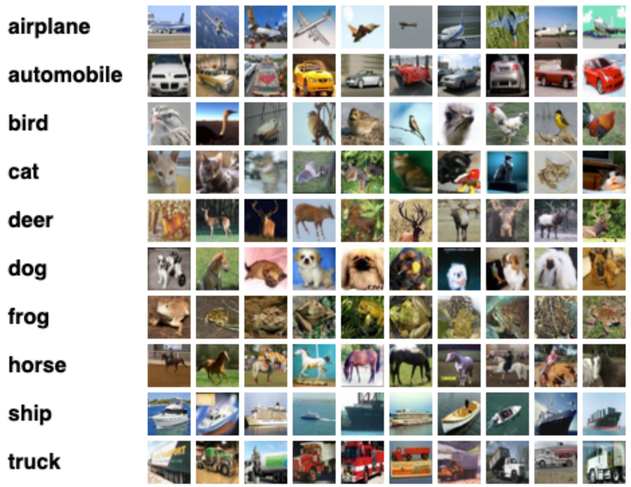
VGG, 2014
Deeper Network



ResNet, 2015
Even deeper thanks to residual connections

Convolutional Neural Networks — Practical

Goals:
1. Implementing a Convolutional Neural Network
2. Understanding the involved computations
3. Building a full Deep Learning pipeline in PyTorch to train a model on a given dataset



CIFAR-10

PyTorch