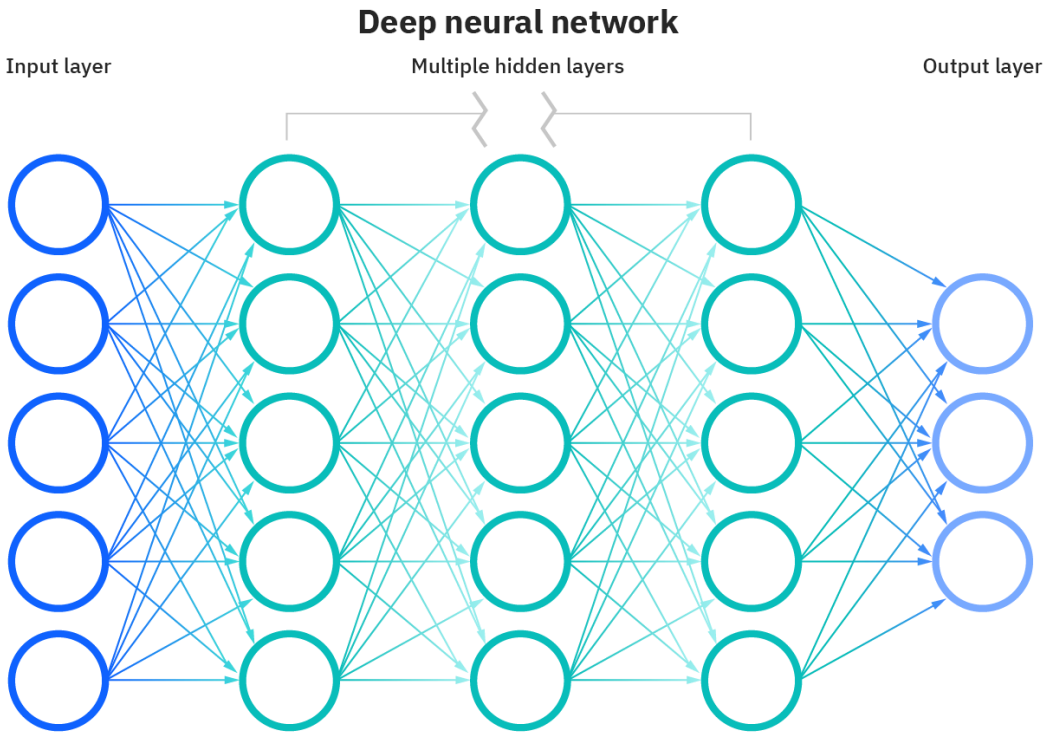
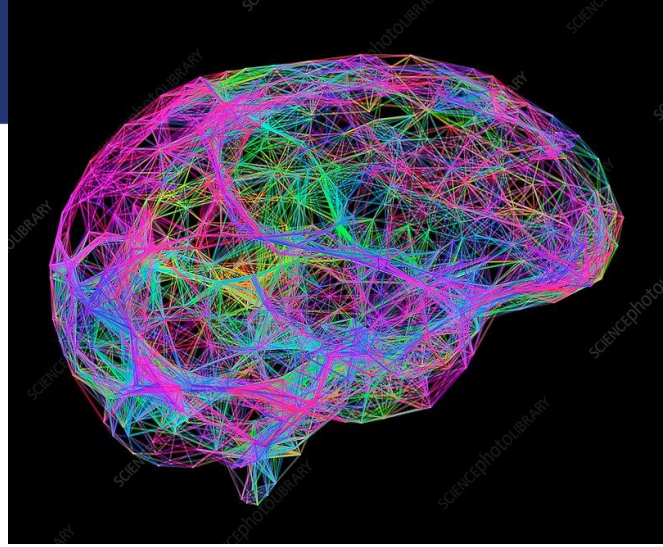
imc FH KREMS
UNIVERSITY OF APPLIED SCIENCES

Deep Learning

Neural Networks

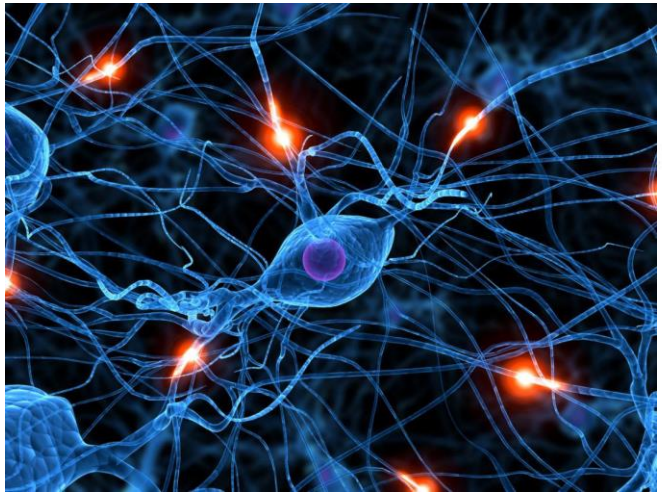


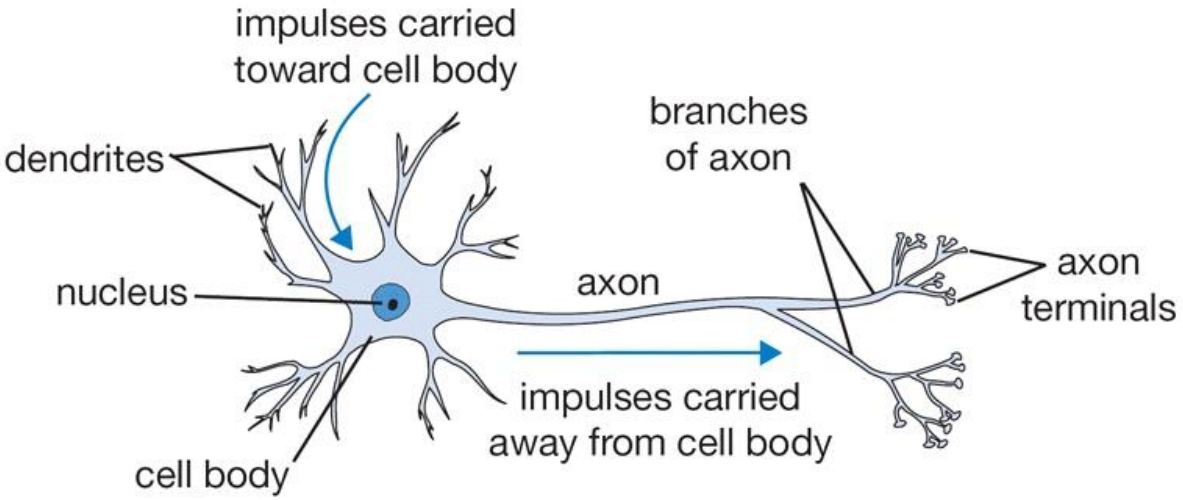
DEEP LEARNING – A NEW PROMISE



Biological Motivation

- There are ~86B neurons in our brain
- Neurons are connected to other neurons through an axon
- The structure resemble to a network, where information is passed from one neuron to another





Geoffrey Hinton (2016)

2021

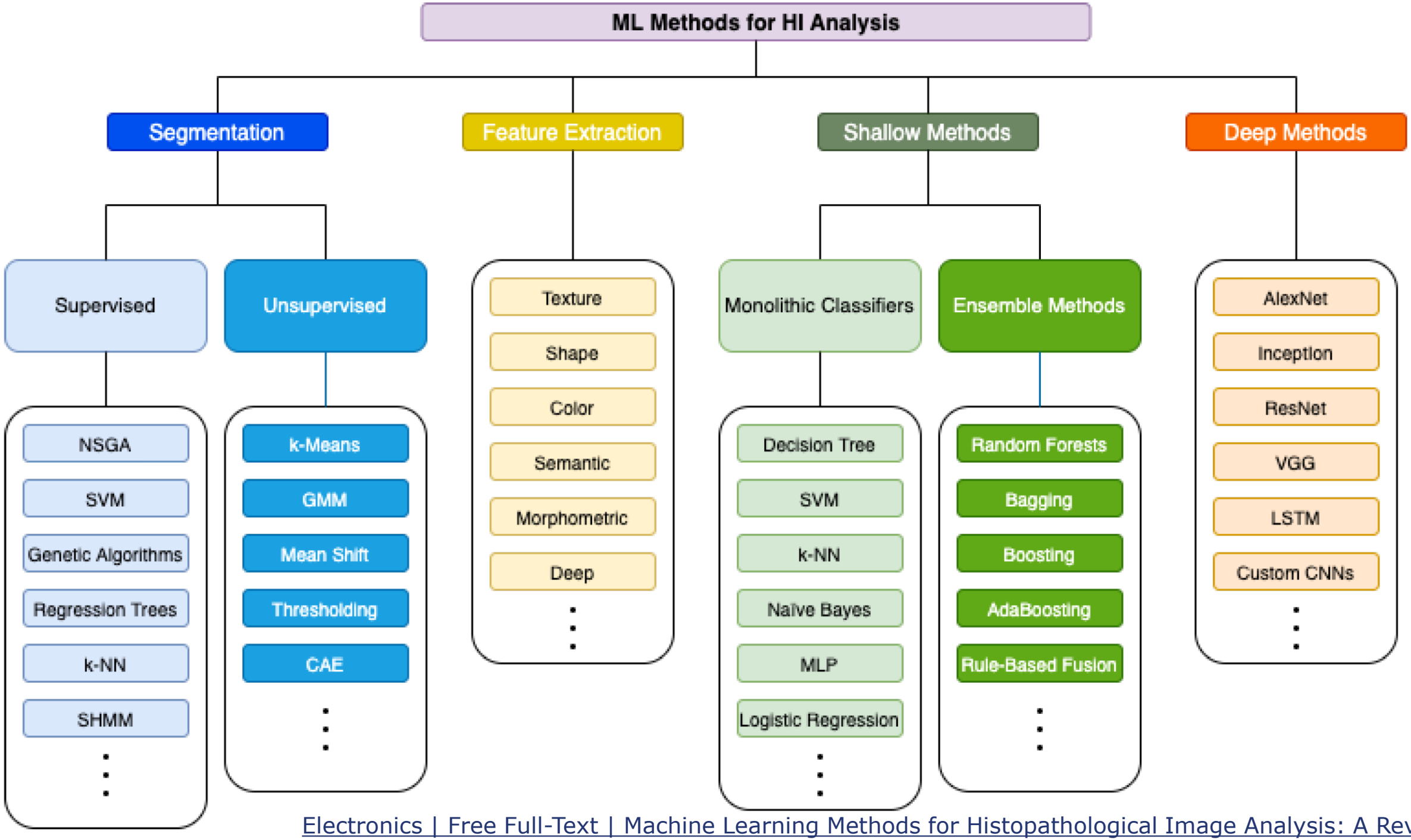
FAULTY IMAGE

AI has a long way to go before doctors can trust it with your life

Image: REUTERS/Luke MacGregor

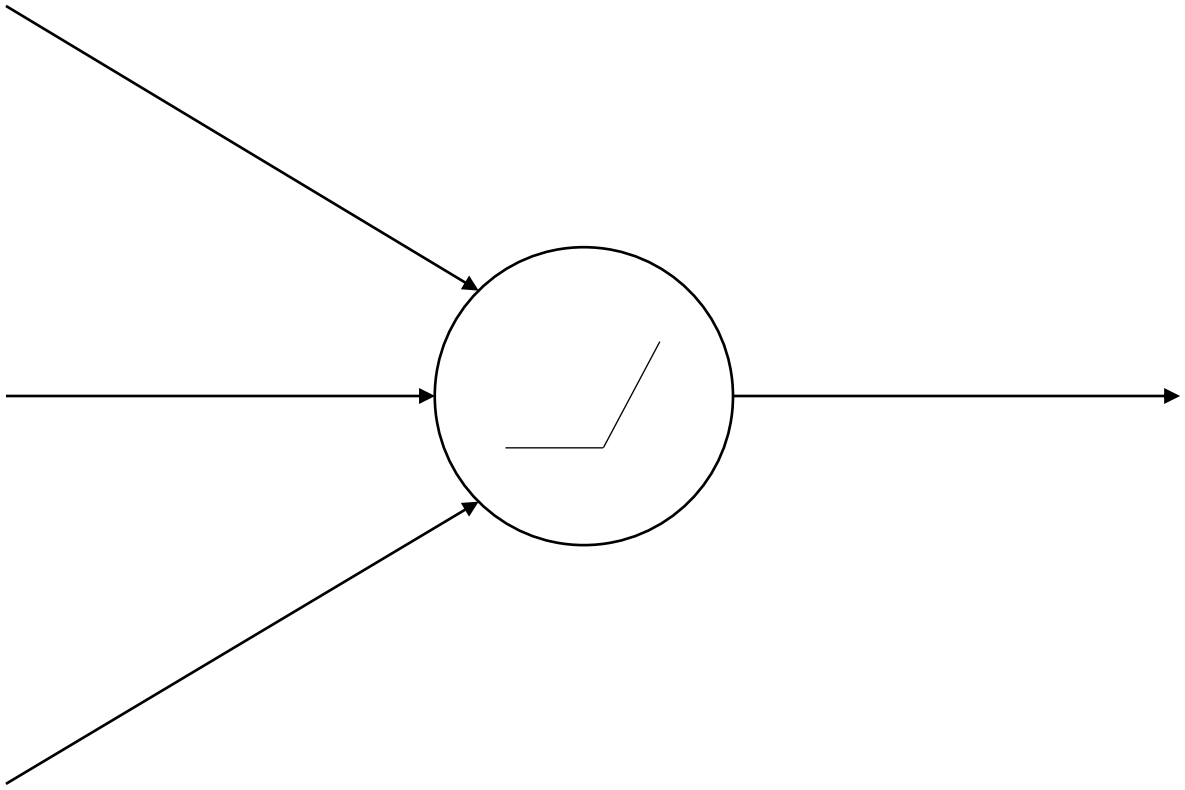
Job security.

By **Gary Smith and Jeffrey Funk** | Published June 4, 2021



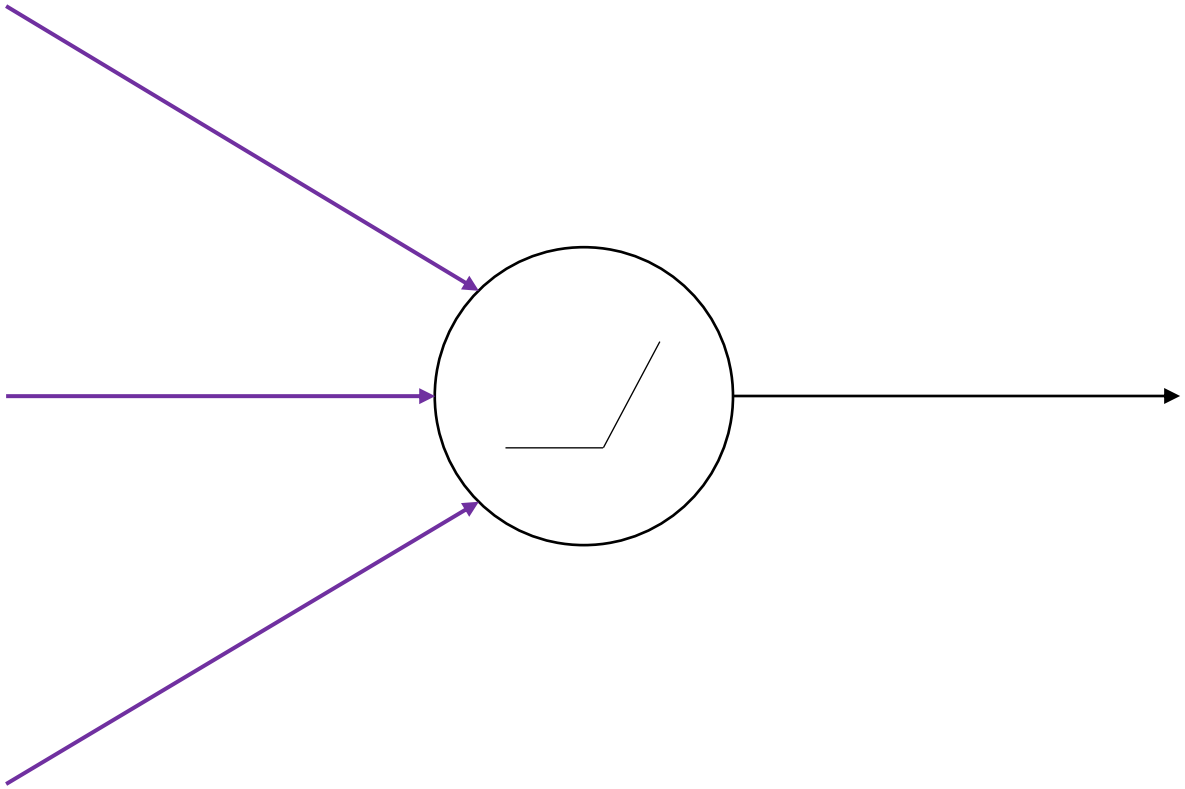
Electronics | Free Full-Text | Machine Learning Methods for Histopathological Image Analysis: A Review

Neuron (Perceptron)



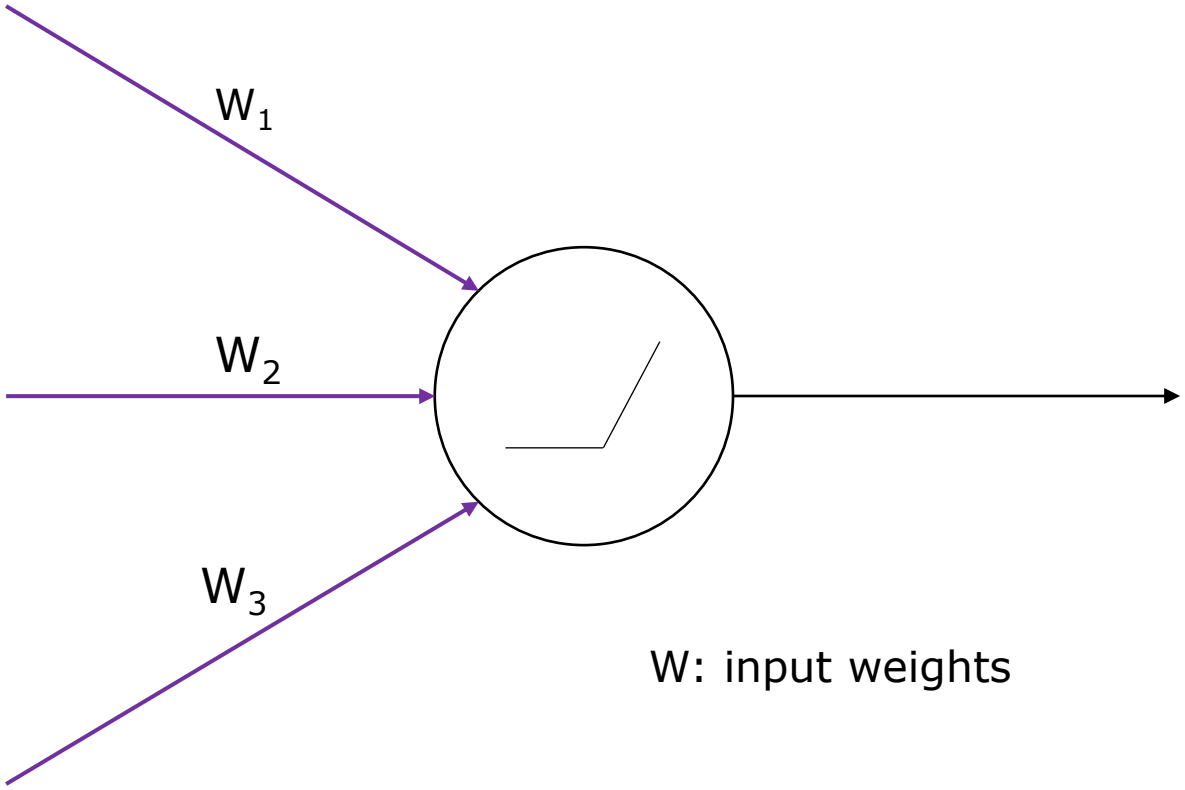
Neuron (Perceptron)

Input synapses



Neuron (Perceptron)

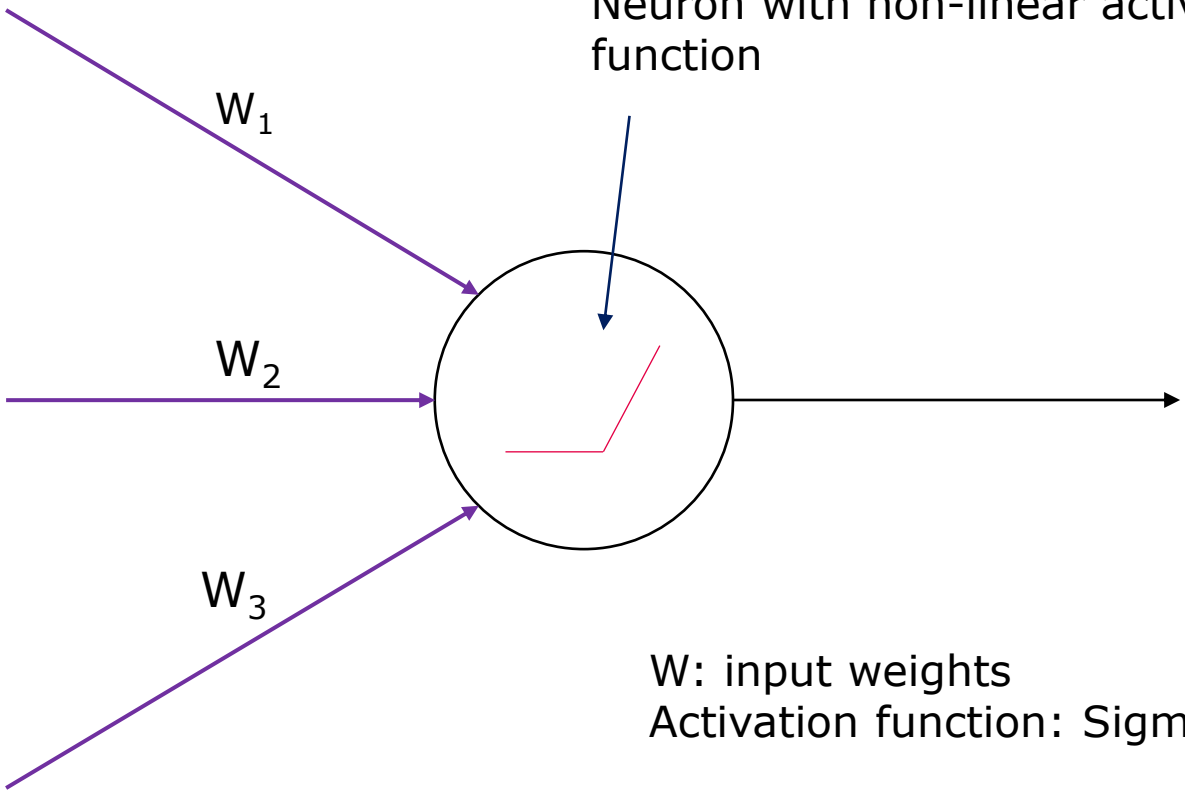
Input synapses



W: input weights

Neuron (Perceptron)

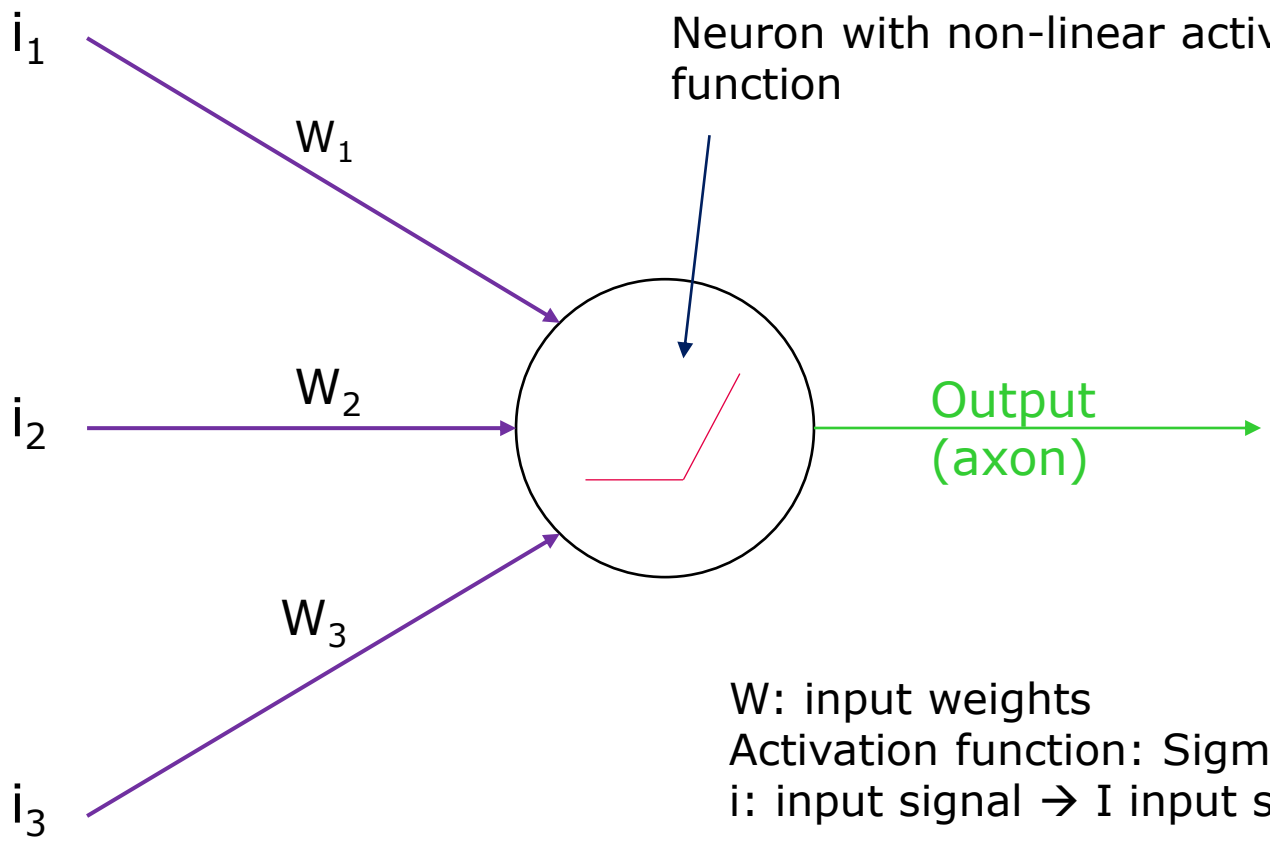
Input synapses



W: input weights
Activation function: Sigmoid // max(0, value)

Neuron (Perceptron)

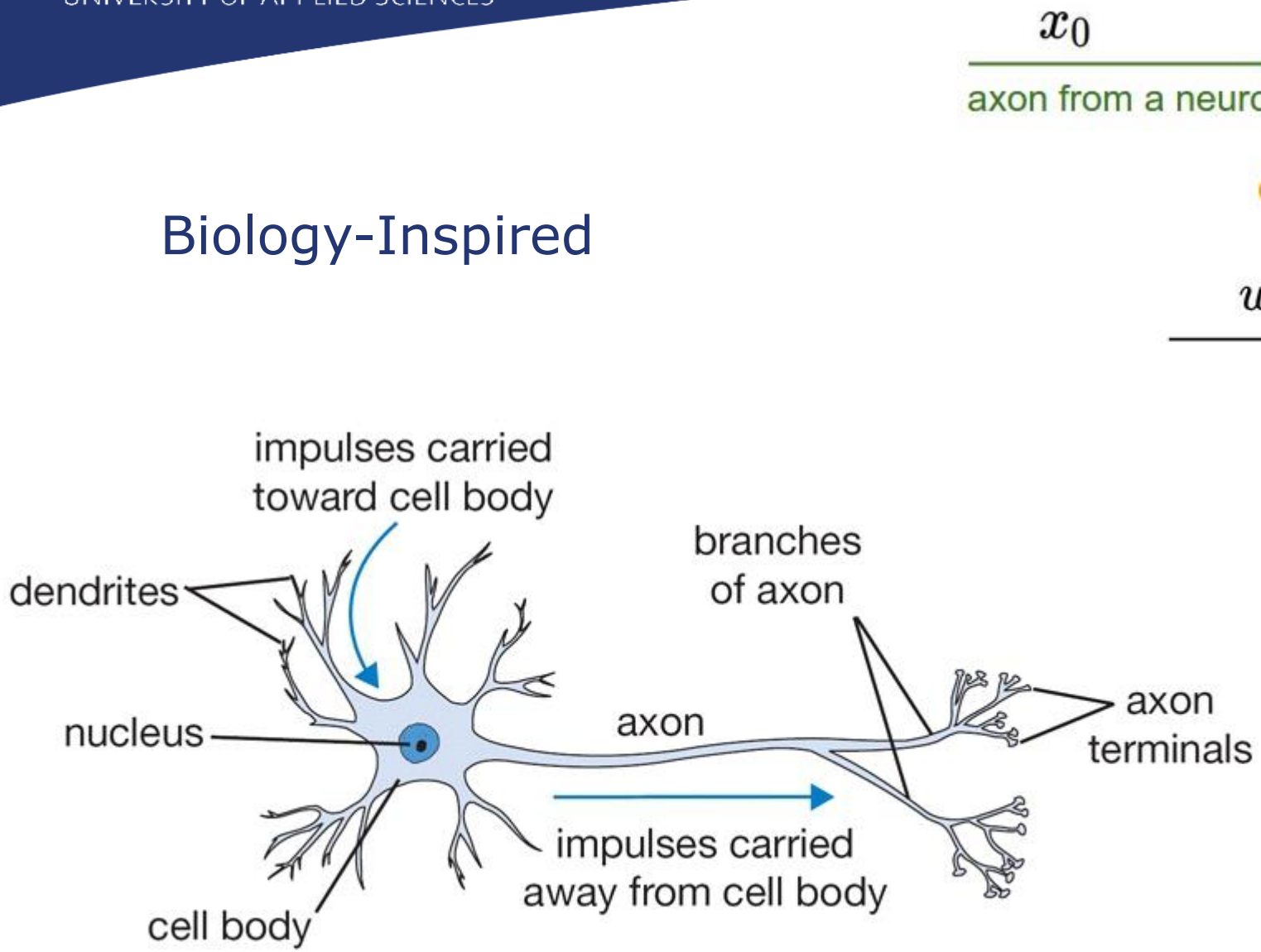
Input synapses

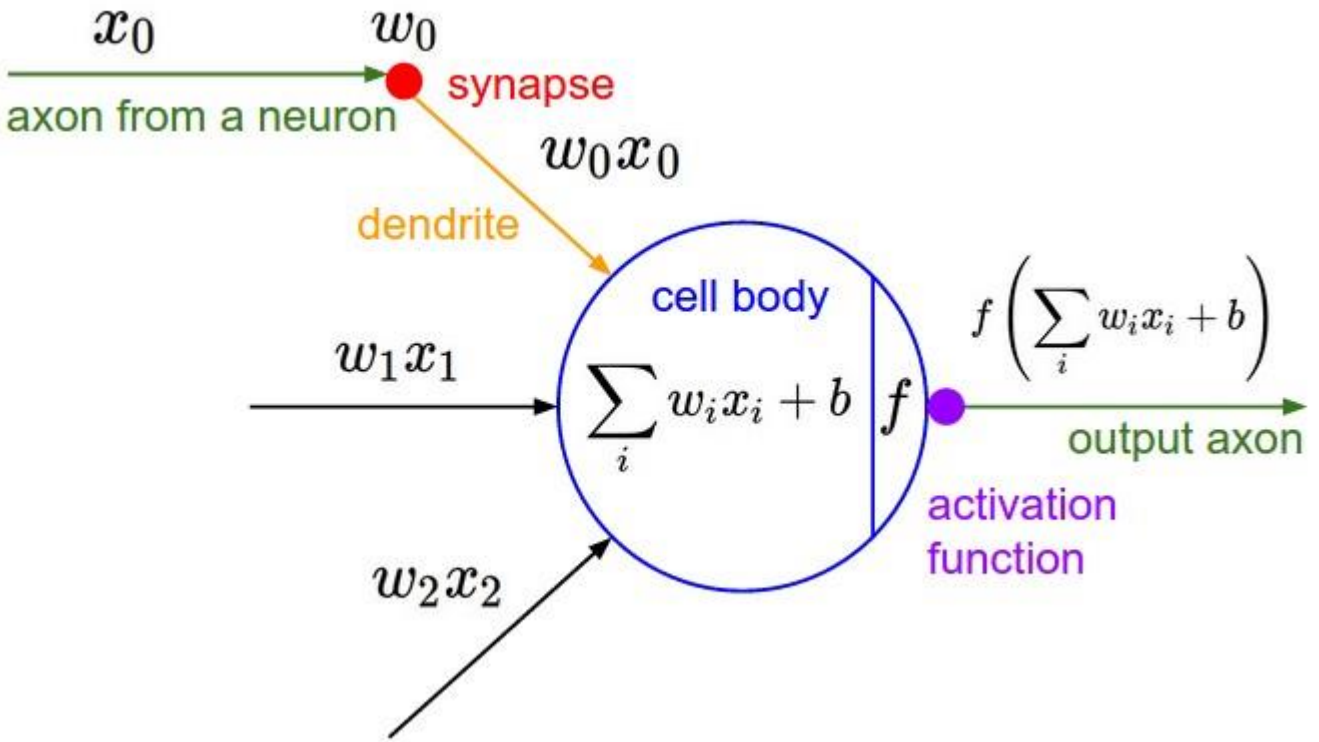


W: input weights
Activation function: Sigmoid // max(0, value)
i: input signal → I input signal vector

$$Output = \max(0, I \cdot W)$$

Biology-Inspired



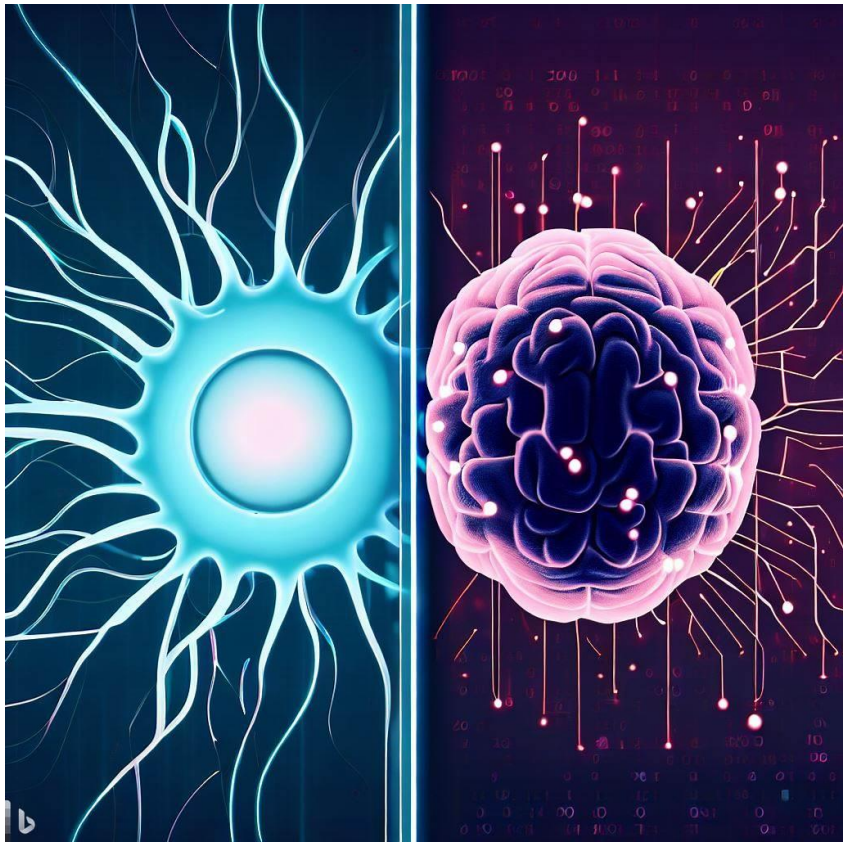


Neuron (Perceptron)

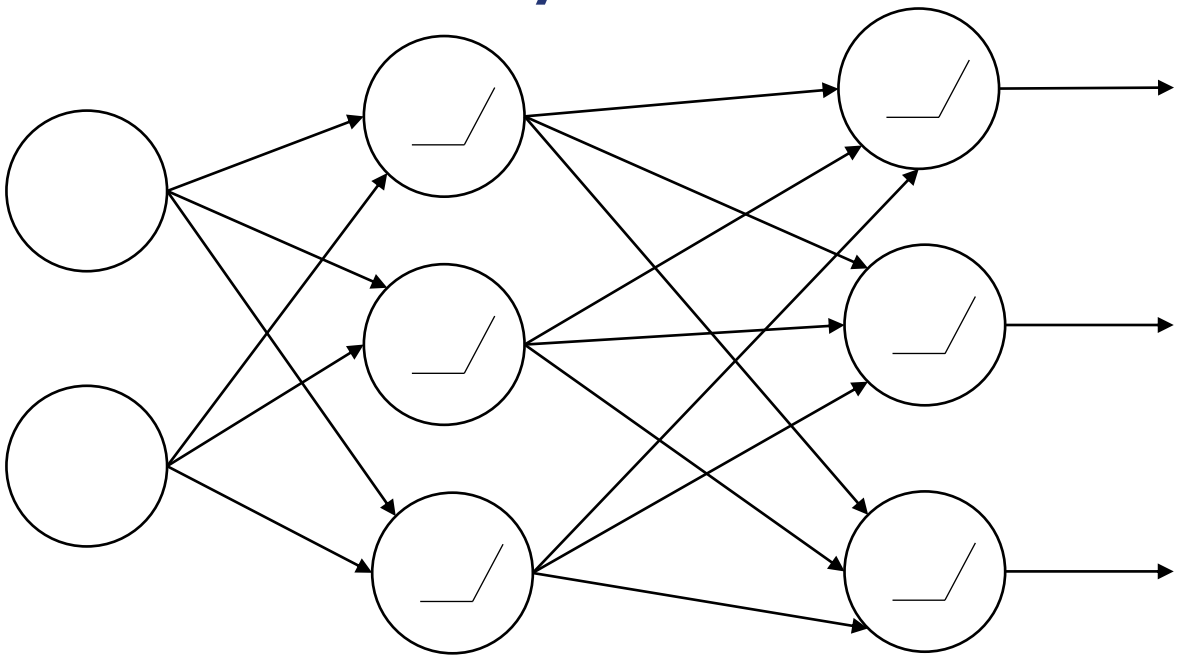
Differences from real brain neurons

- The perceptron model is quite different from the biological neurons:
  - Those communicate by sending spike signals at various frequencies
  - The learning in brains seems also quite different

- It would be better to think of artificial neural networks as non-linear projections of data (and not as a model of brain)



Neural Network Layers

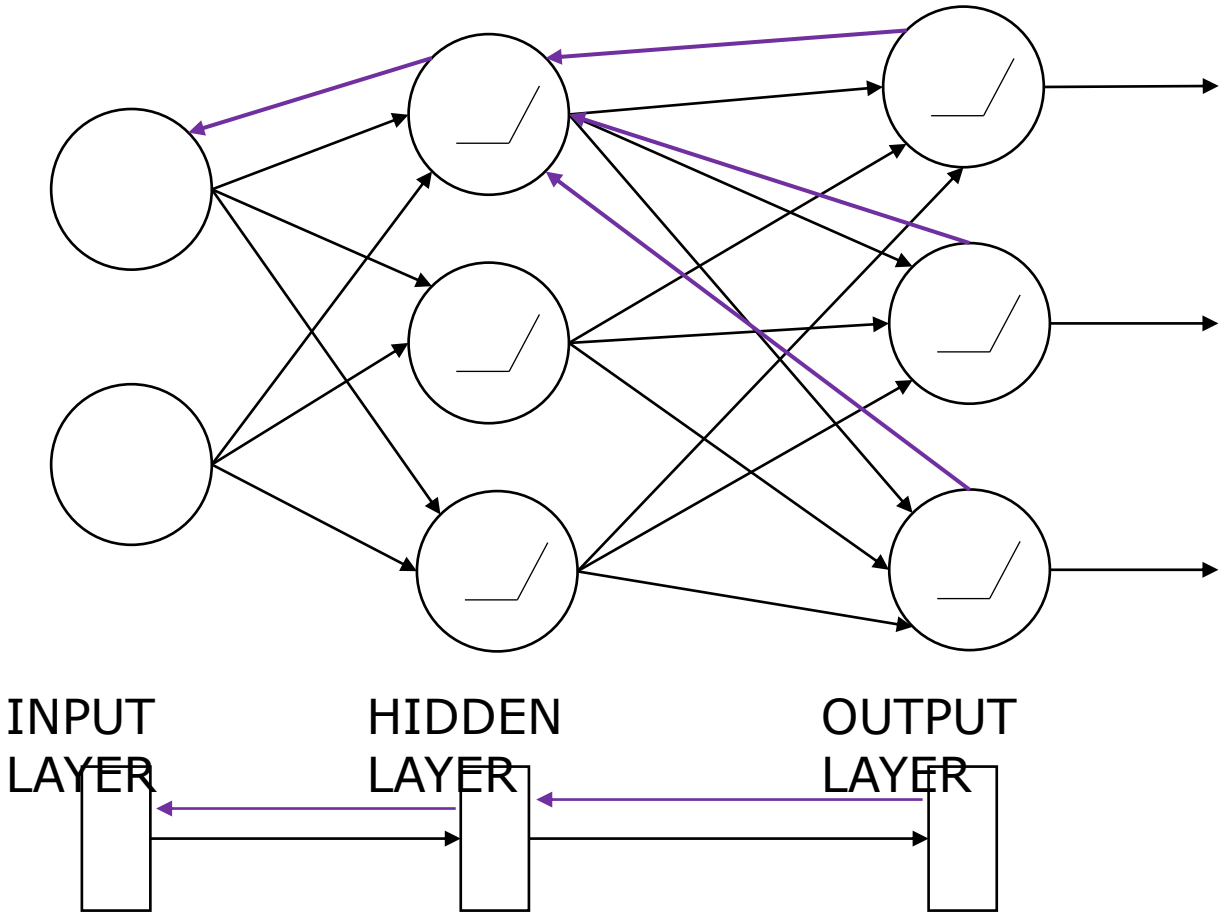


INPUT LAYER

HIDDEN LAYER

OUTPUT LAYER

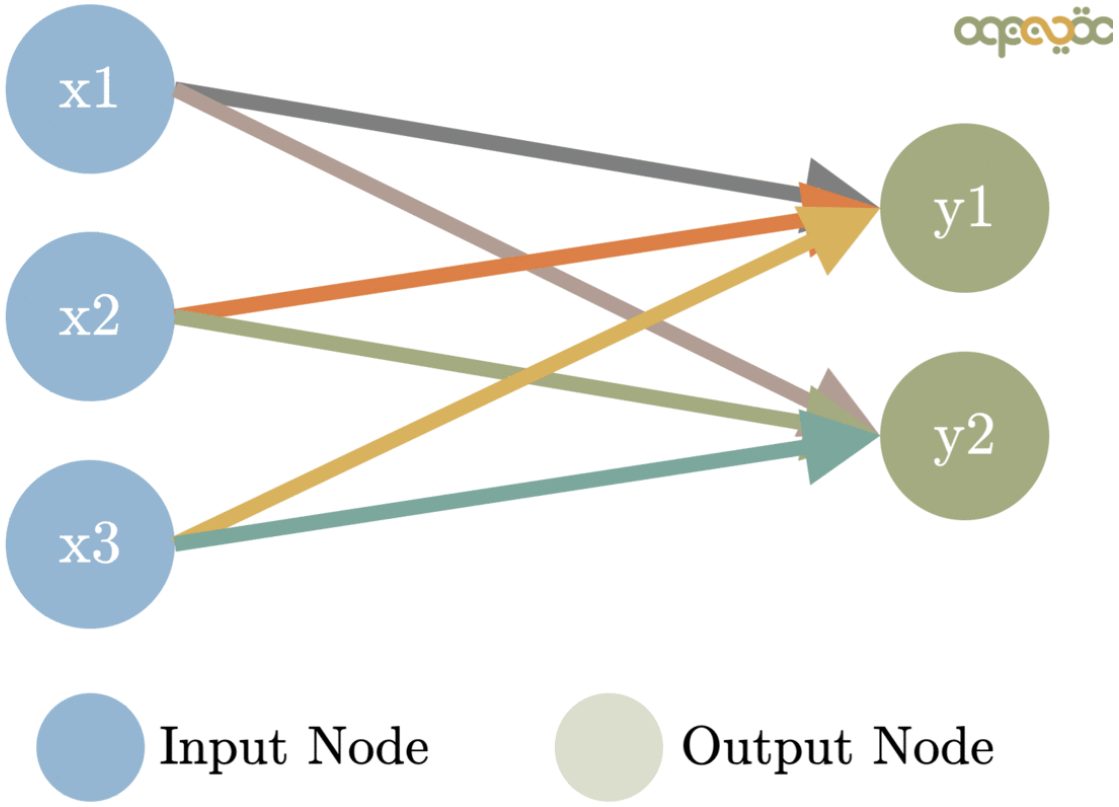
Neural Network Layers



- During the training, the network computes the values for a certain input through the network and compares the result to the given output.
- The gradient of the error is used to correct the neuron weights.

Neural Network

Simple network



Error calculation

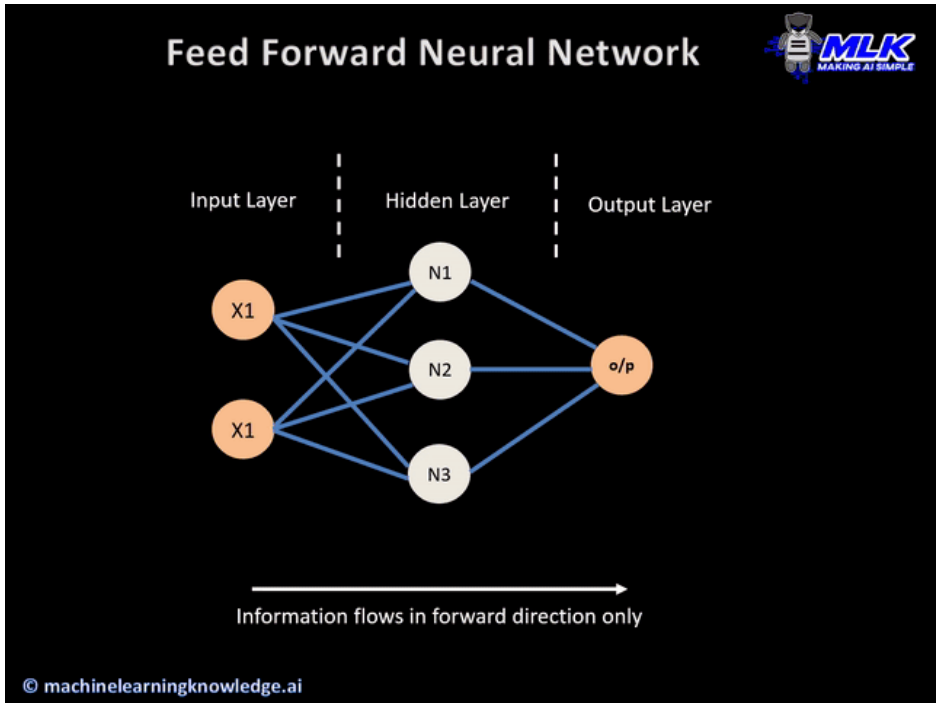
Loss Function

- Calculates the distance between the network output and the desired value.
- You decide which loss function to use, based on the task:
  - Mean Squared Error (MSE) – regression
  - Binary Cross Entropy – binary classification
  - Categorical Cross Entropy Loss – multiclass (one-hot-vectors)
  - Negative Log-Likelihood loss – multiclass
  - Multilabel margin loss – multilabel

And many, many more…
torch.nn — PyTorch 2.0 documentation

The Inference Part – Feed Forward

Given an input, predicting the output by calculating the values through the network



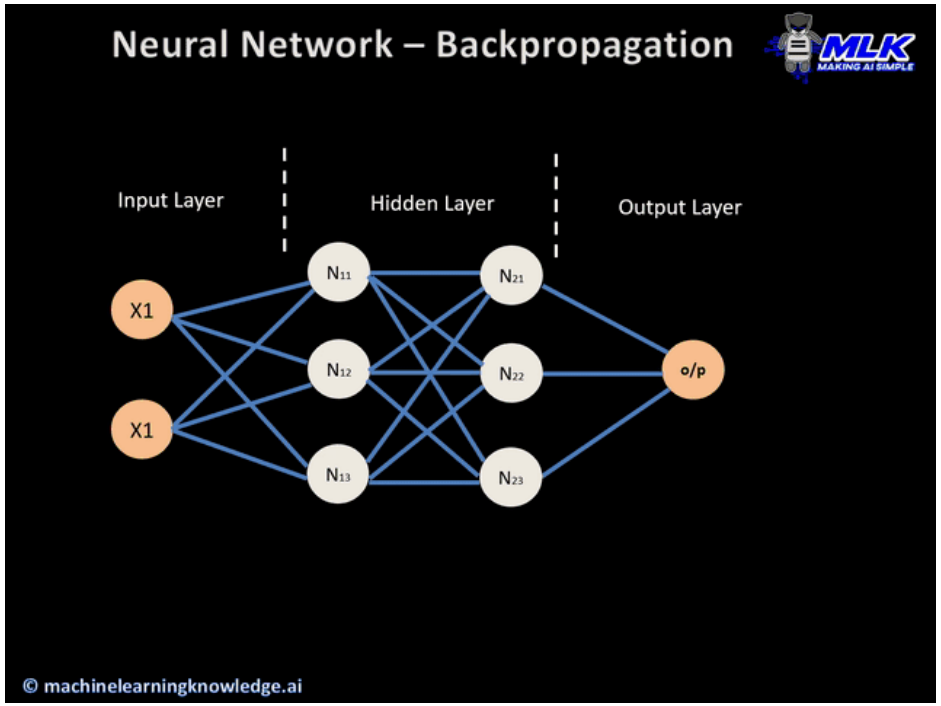
Backpropagation – Training Networks to Learn

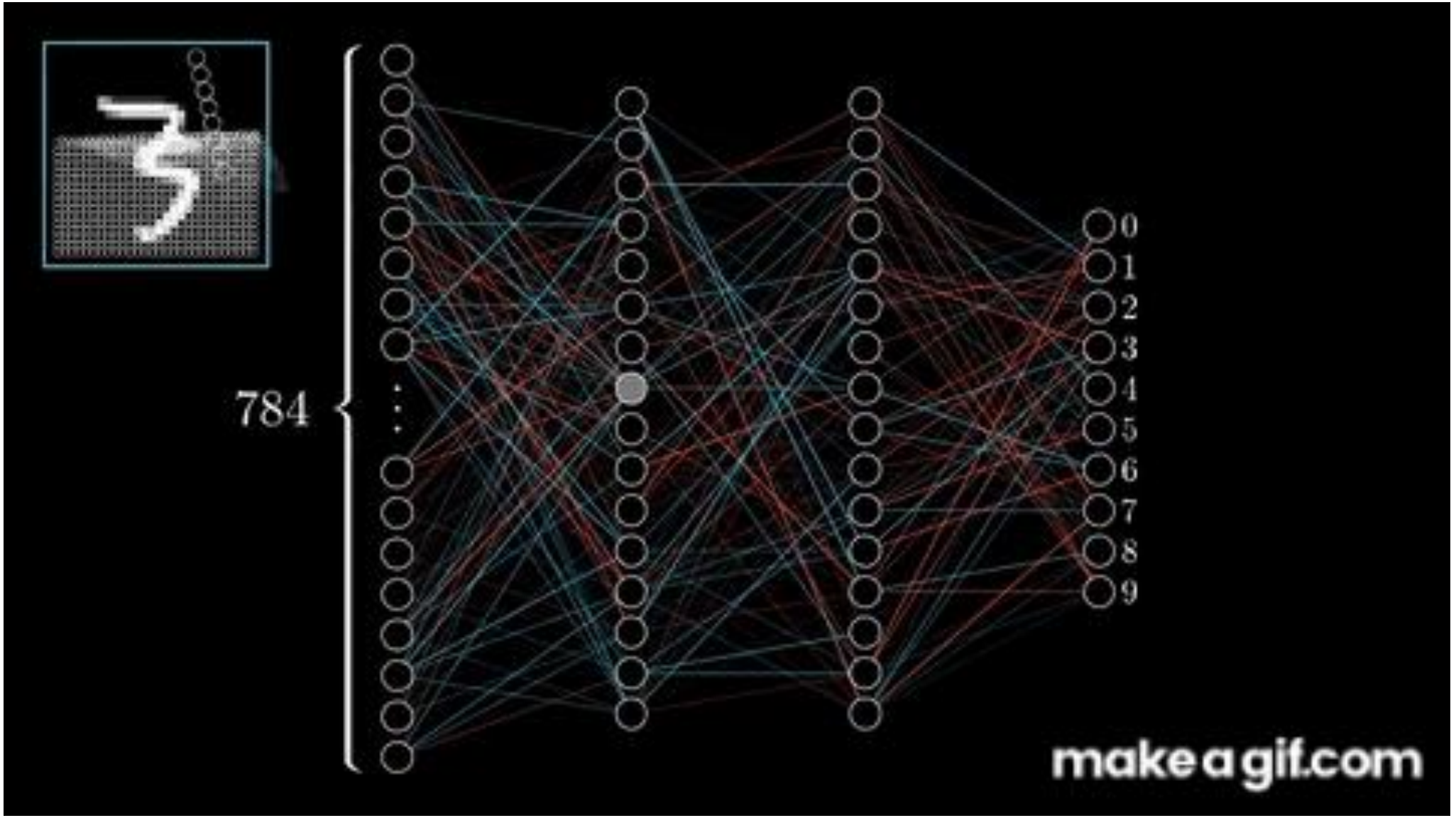
Backpropagation occurs **only during the training.**

The network calculates the error (**loss function**)

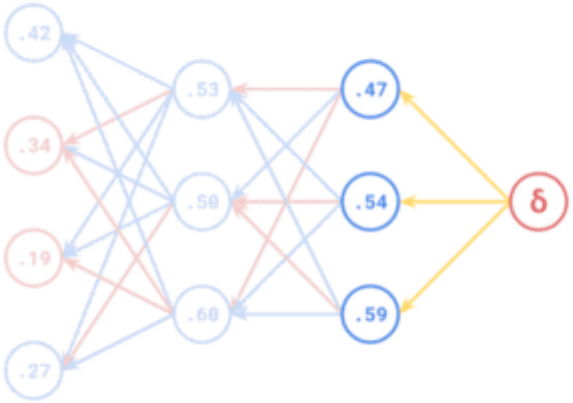
Then gently updates the network weights

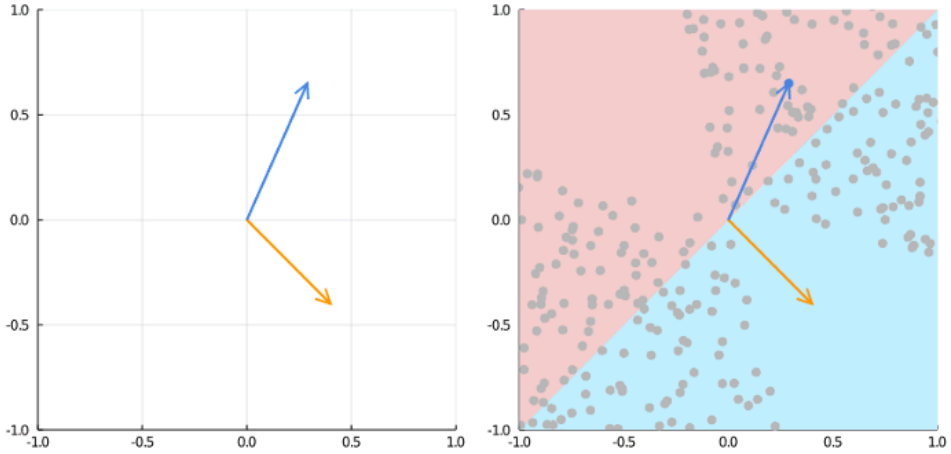
Next time, when they "see" this sample, the prediction is closer to the desired result.




784
0
1
2
3
4
5
6
7
8
9
makeagif.com

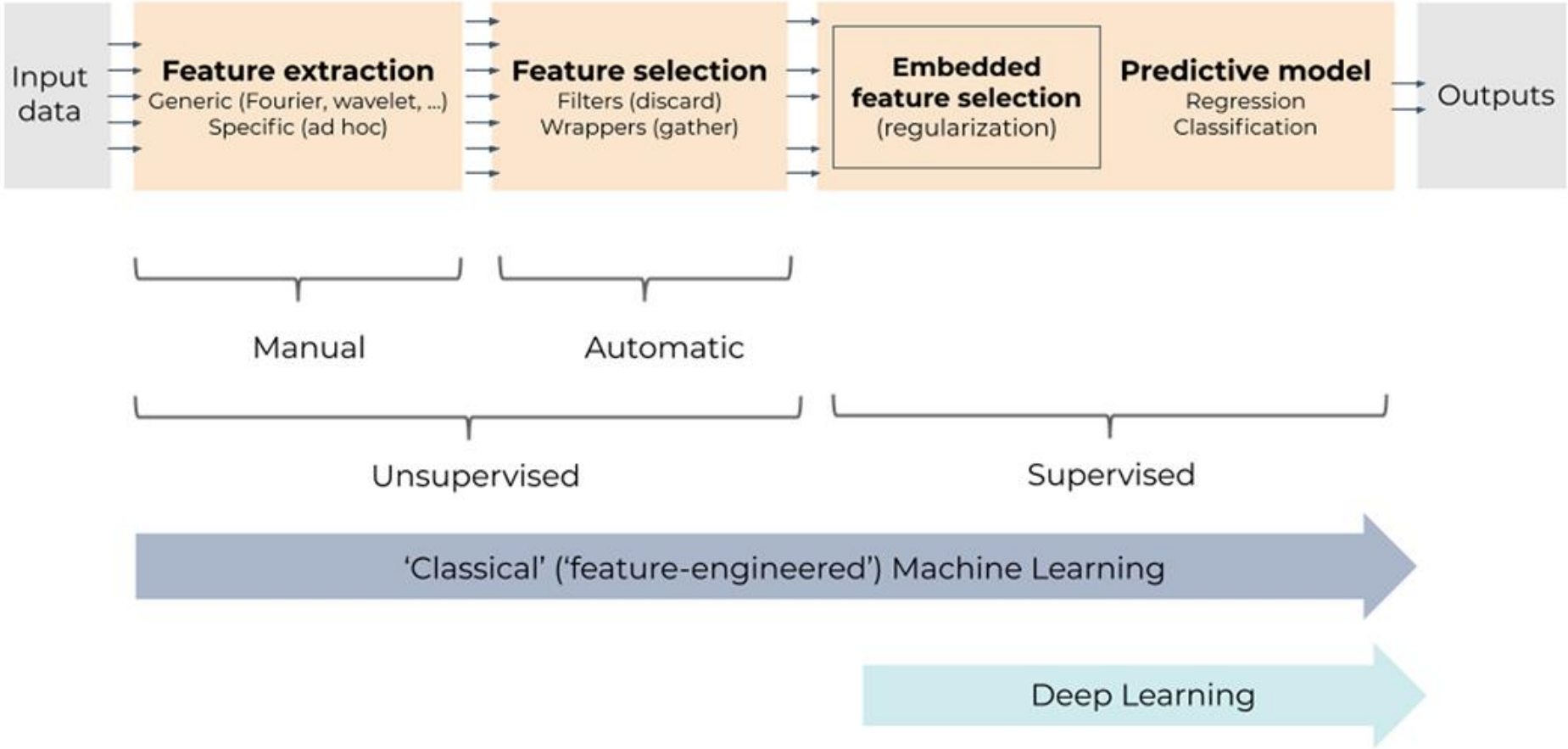
For a Deeper Dive



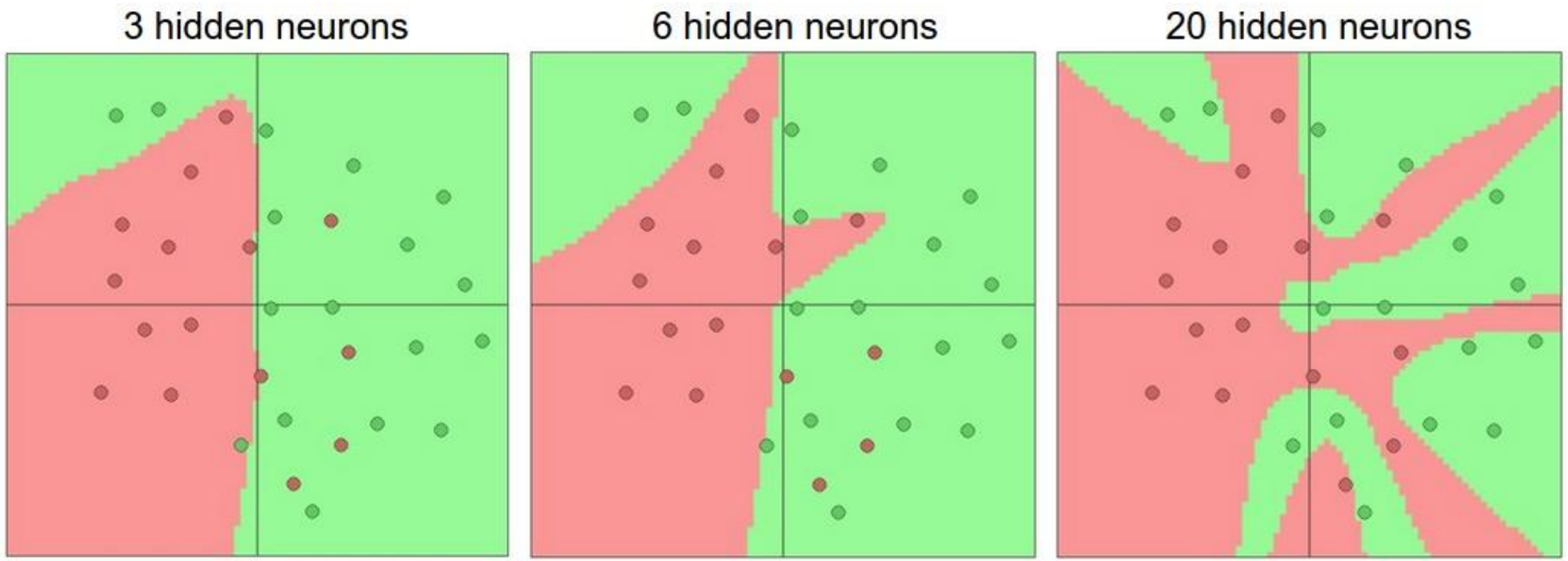


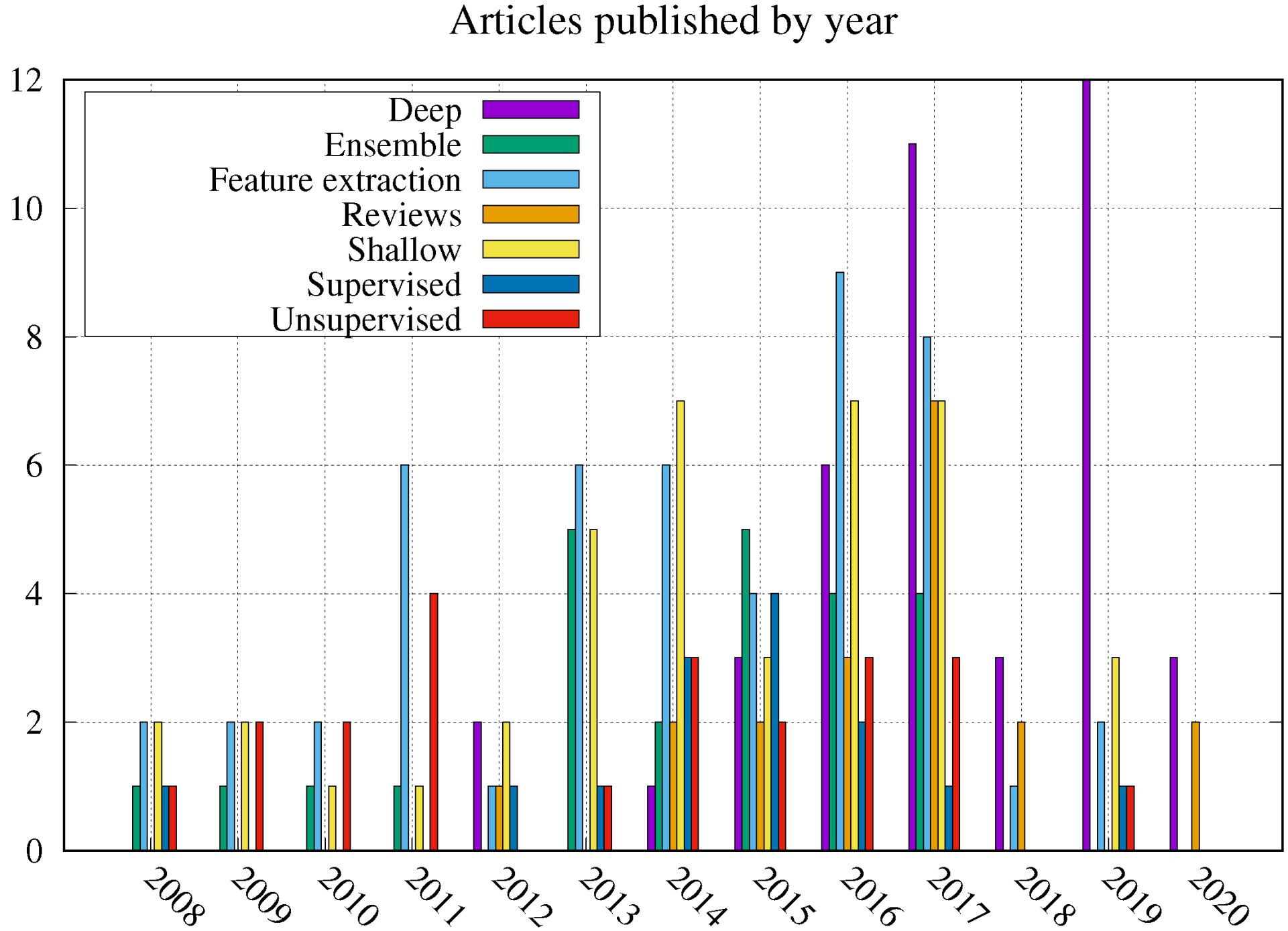
- The Building Blocks of Deep Learning | by Tyron Jung | The Feynman Journal | Medium
- What Makes Backpropagation So Elegant? | by Tyron Jung | The Feynman Journal | Medium

Manual Machine Learning vs Deep Learning



Degrees of freedom




Articles published by year
Deep
Ensemble
Feature extraction
Reviews
Shallow
Supervised
Unsupervised
0
2
4
6
8
10
12
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020

Large-scale Medical Image Datasets

Challenges and Competitions:

- Kaggle: numerous competitions
- Grand-Challenge.org: numerous competitions
- MICCAI (Medical Image Computing and Computer-Assisted Intervention) conference

Collections and Directories

- NBIA/NCIA/NTIA: numerous datasets
- XNAT: numerous datasets
- ACR DSI Dataset Directory
- Aylward open access medical image repository list

Individual Datasets

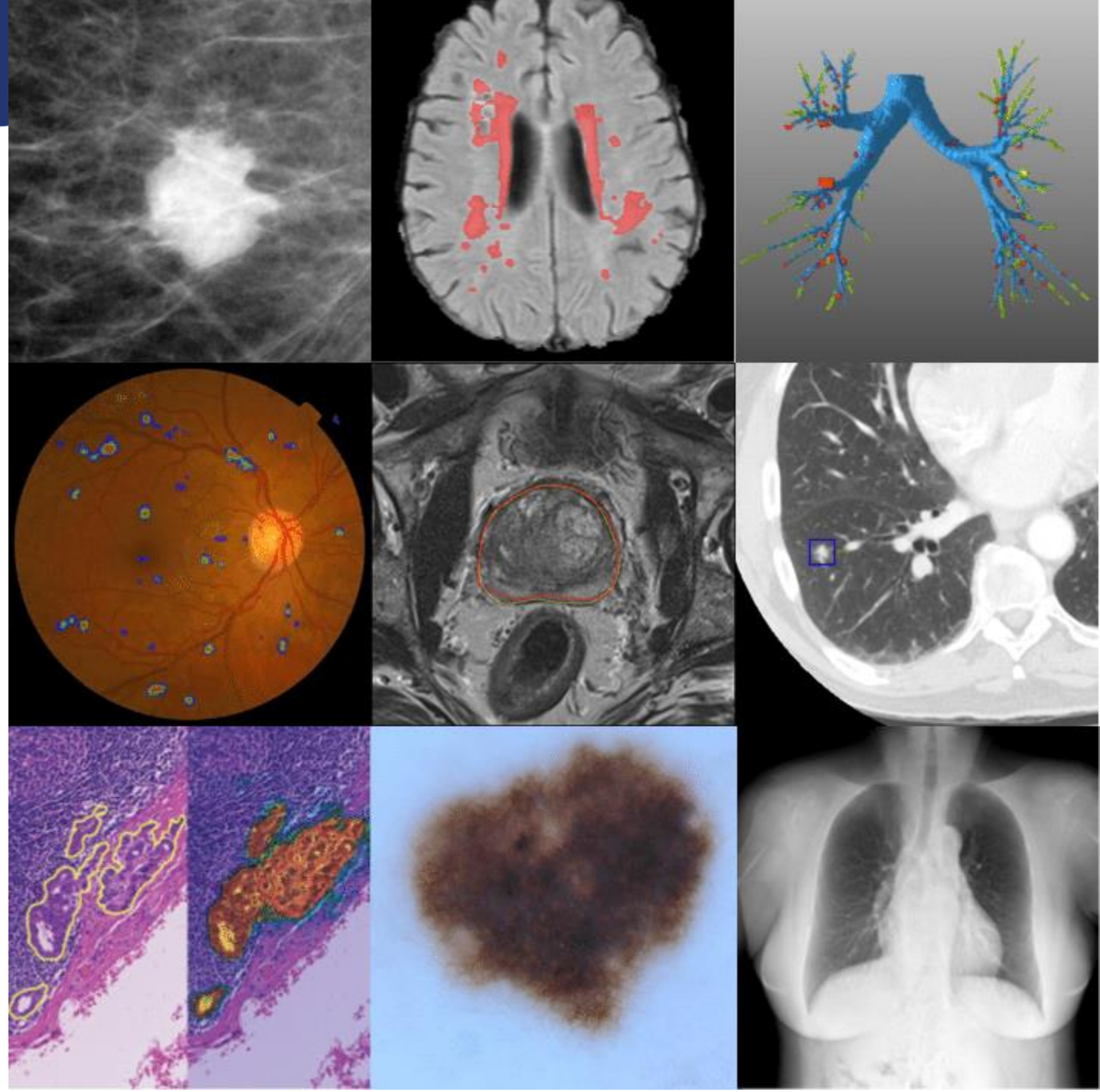
- Osteoarthritis Initiative: 431k X-ray & MR studies
- CheXpert: 224k CXR
- NIH CXR: 112k CXR
- MURA: 40k MSK X-rays
- DeepLesion: 33k bookmarked CT images
- DDSM: 2.5k mammography studies
- fastMRI: 1.5k knee MRI
- MRNet: 1.3k knee MR
- BraTs: Brain Tumor MRI

imc FH KREMS
UNIVERSITY OF APPLIED SCIENCES

Tasks

Where DL can help

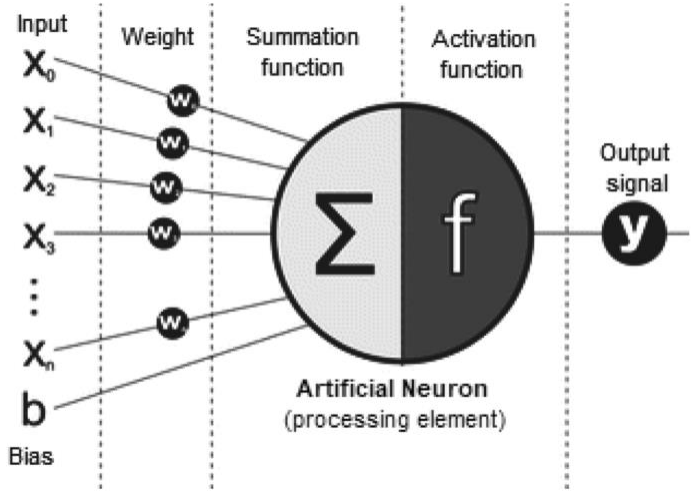
- Classification
- Detection
- Segmentation
- Visualization

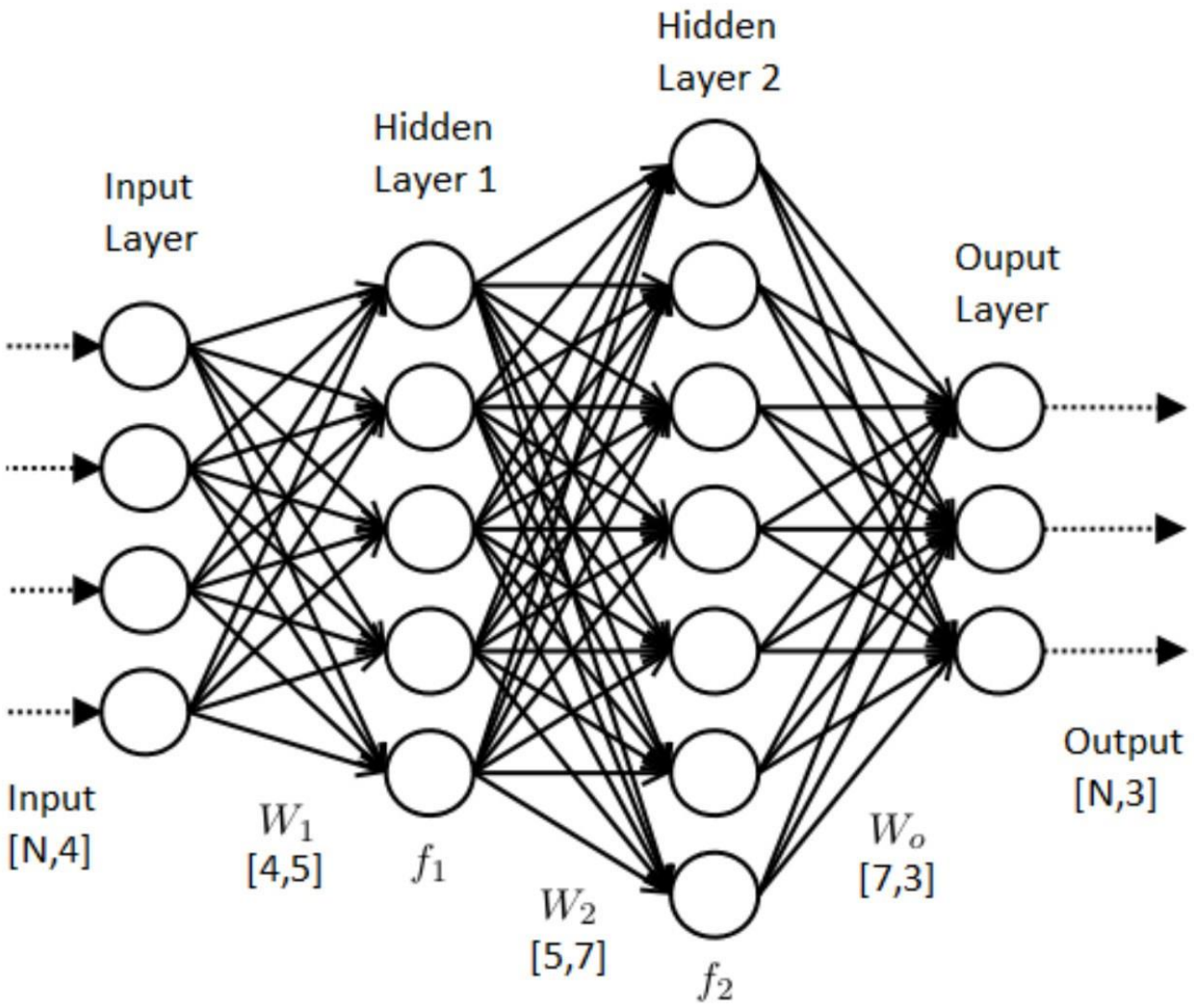


FF Neural Networks

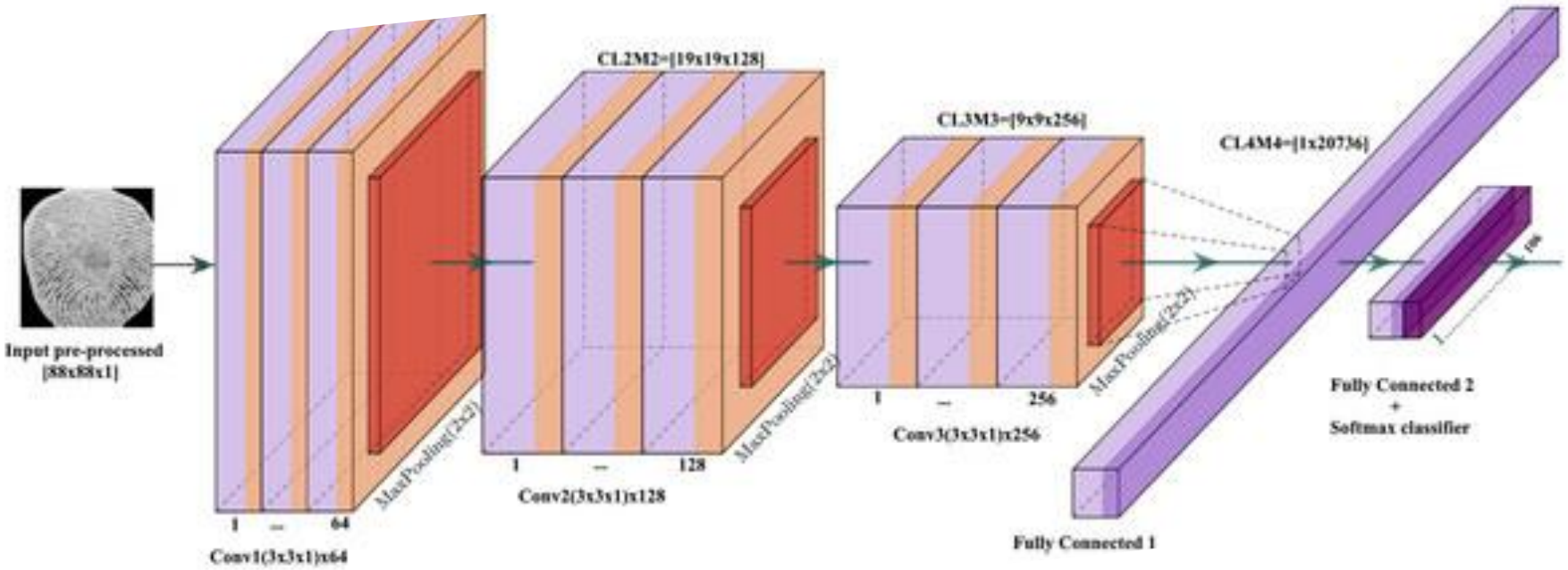
Generic Architecture

- Vector input
- Non-linear mapping (activation) function



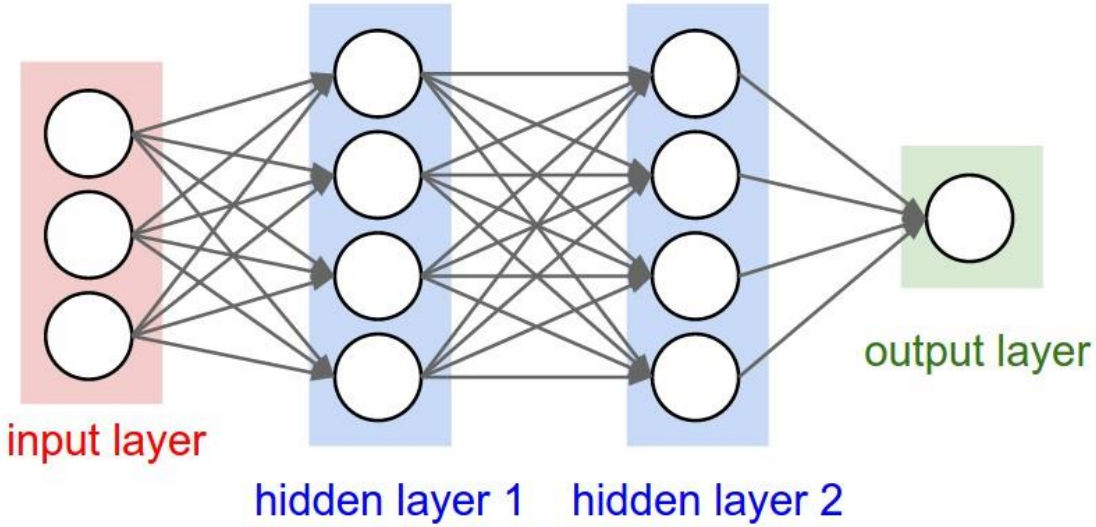


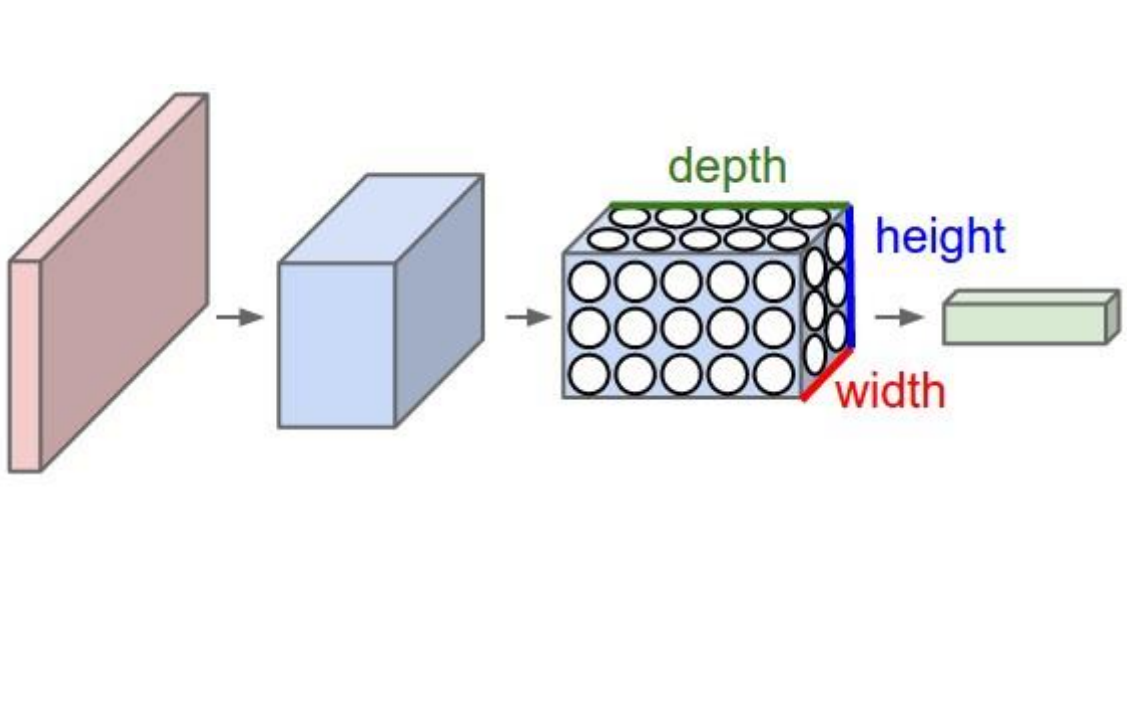
Convolutional Neural Networks (CNN)



From FF to CNN

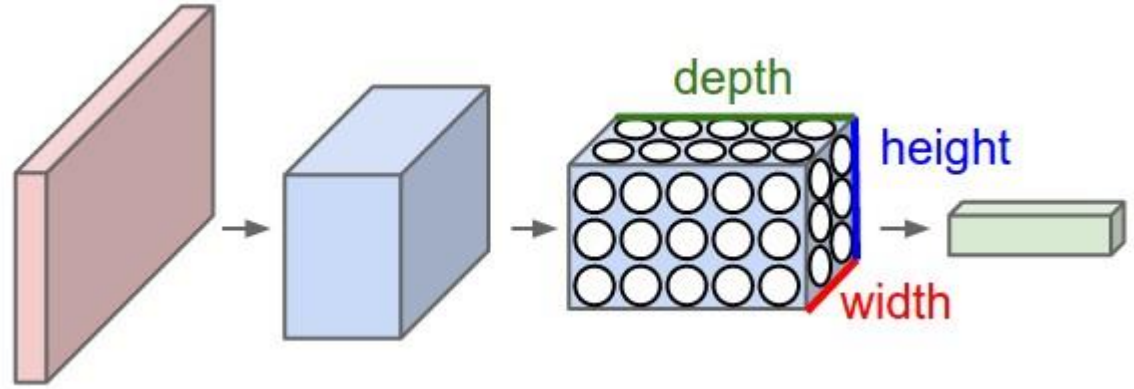
Spatial view of the input





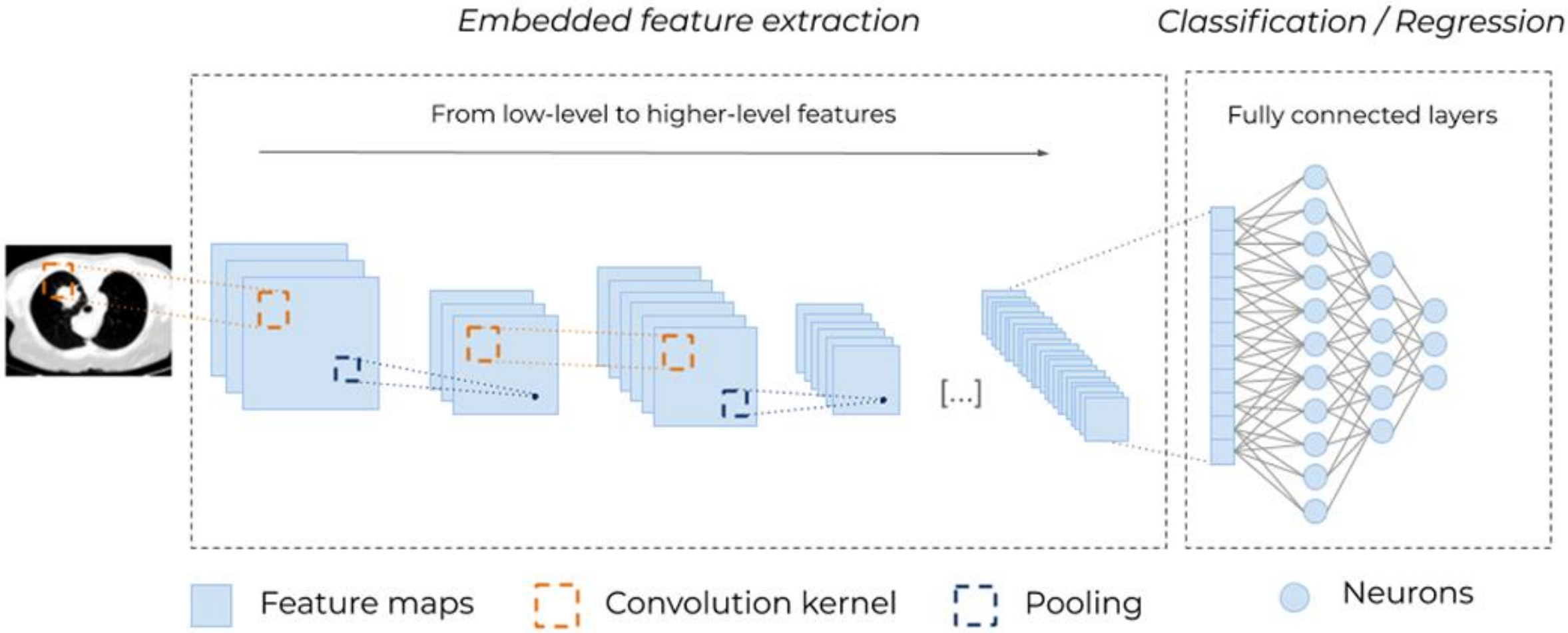
CNN

Convolutional Neural Network



- CNNs are also composed of layers, but those layers are **not** fully connected:
- CNN layers have filters, sets of cube-shaped weights that are applied throughout the image.
- Each 2D slice of the filters are called kernels.
- These filters introduce translation invariance and parameter sharing.
- How are they applied? Convolutions!

Convolutional Neural Network (CNN)



Artificial intelligence and machine learning for medical imaging: A technology review - PubMed

Key characteristics of CNNs

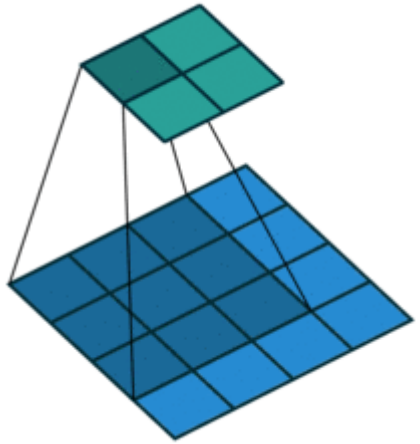
Why do they work so well on images?

- They learn a **hierarchical representation** of structures in the data
  - (i.e., more **complex** representations are expressed in terms of **simpler** representations).
- Preserve the **spatial structure** of the features extracted.

Convolutions

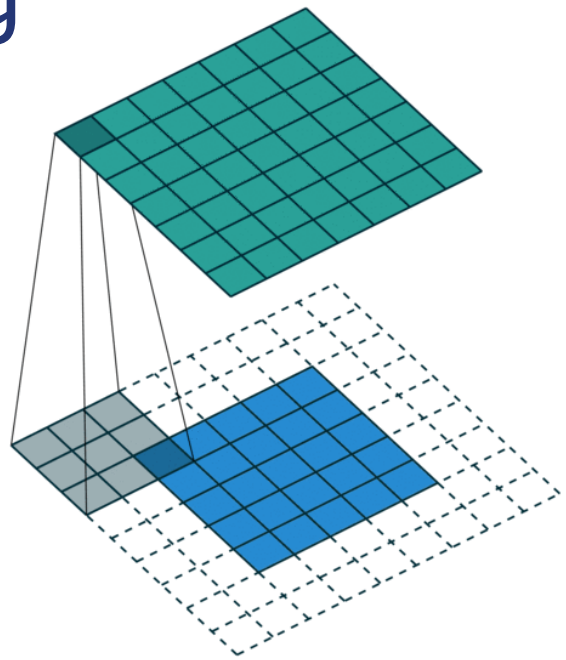
What about the edges?

- If we apply convolutions on a normal image, the result is down-sampled by an amount depending on the size of the filter:

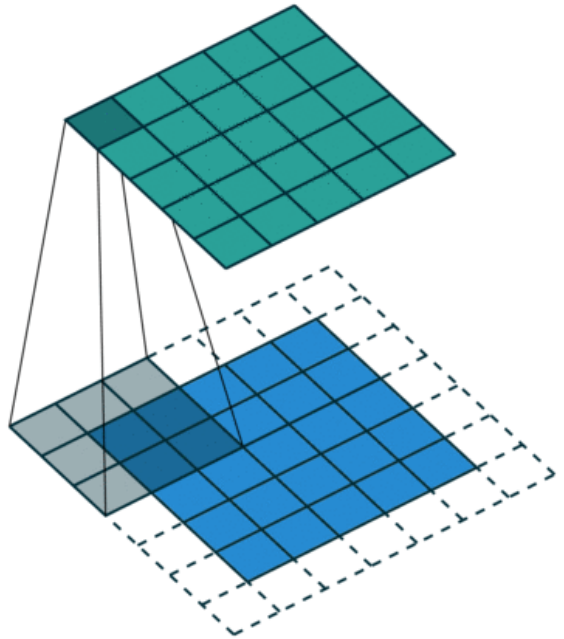


- We can avoid this by **padding** the edges in different ways.

Padding



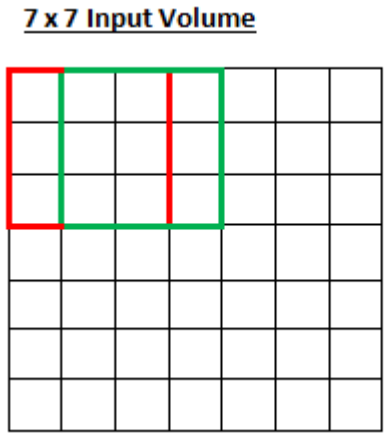
Full padding. Adds a border of zeros such that all image pixels are visited the same number of times by the filter. Increases size of output.

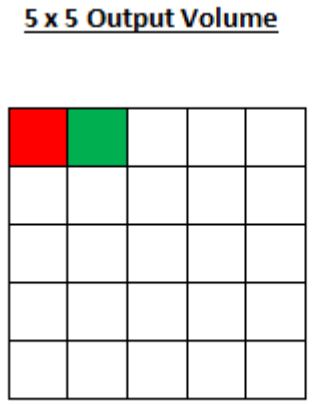


Same padding. Ensures that the output has the same size as the input.

Stride

"Moving Window" Size

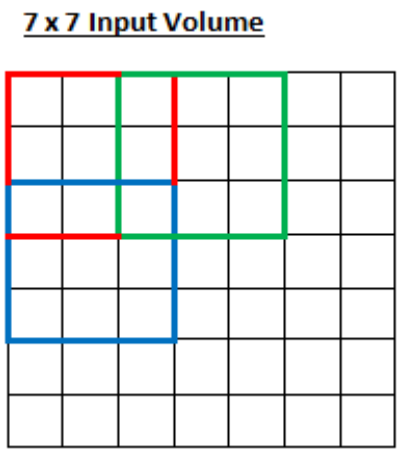


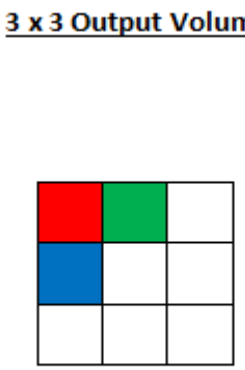


- **Stride** controls how the filter convolves around the input volume.
- Calculating the output size:

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

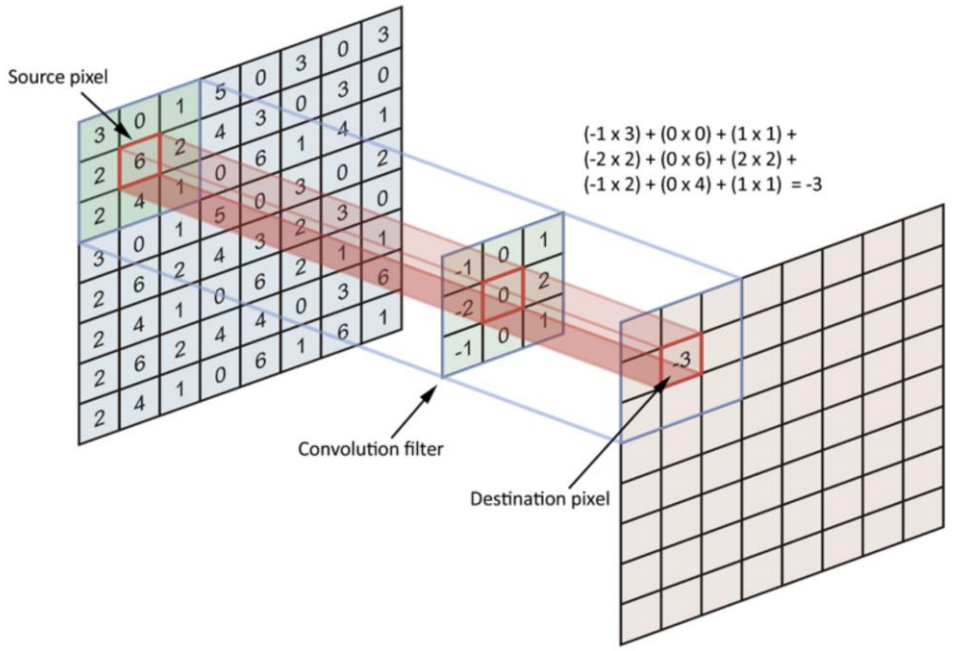
- Where n_{out} is output dim,
  n_{in} is the input dim,
  k is the filter size,
  p is padding
  s the stride

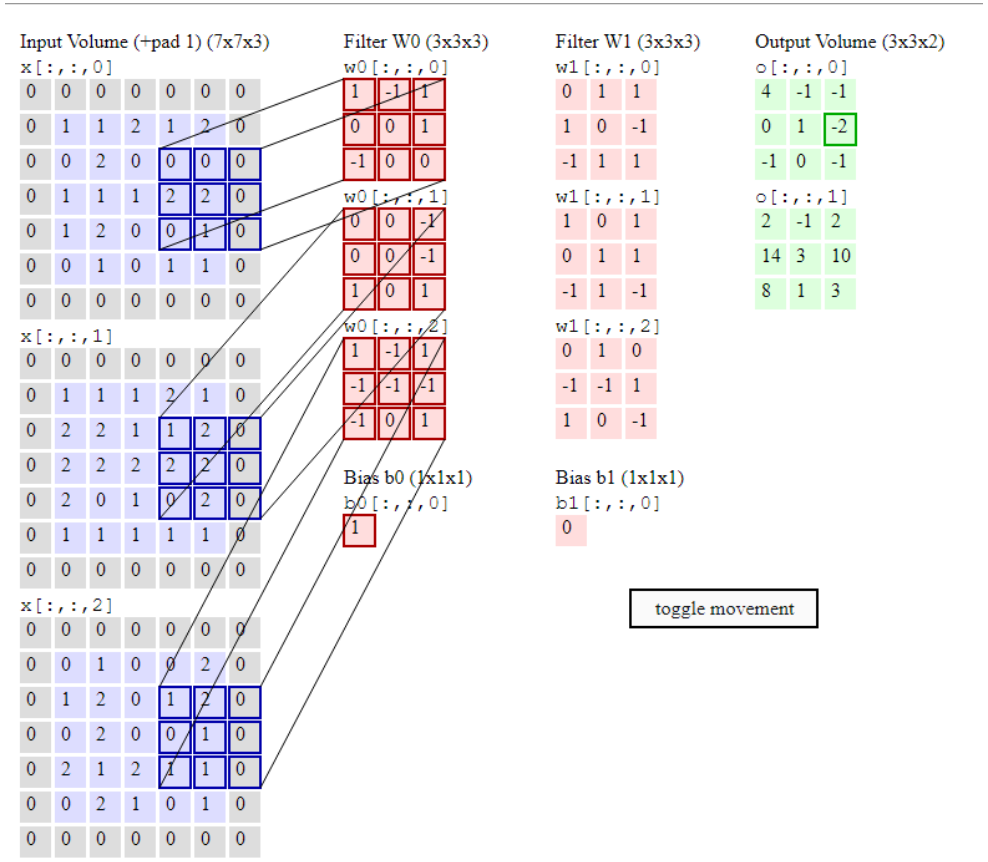




Convolutional Neural Networks

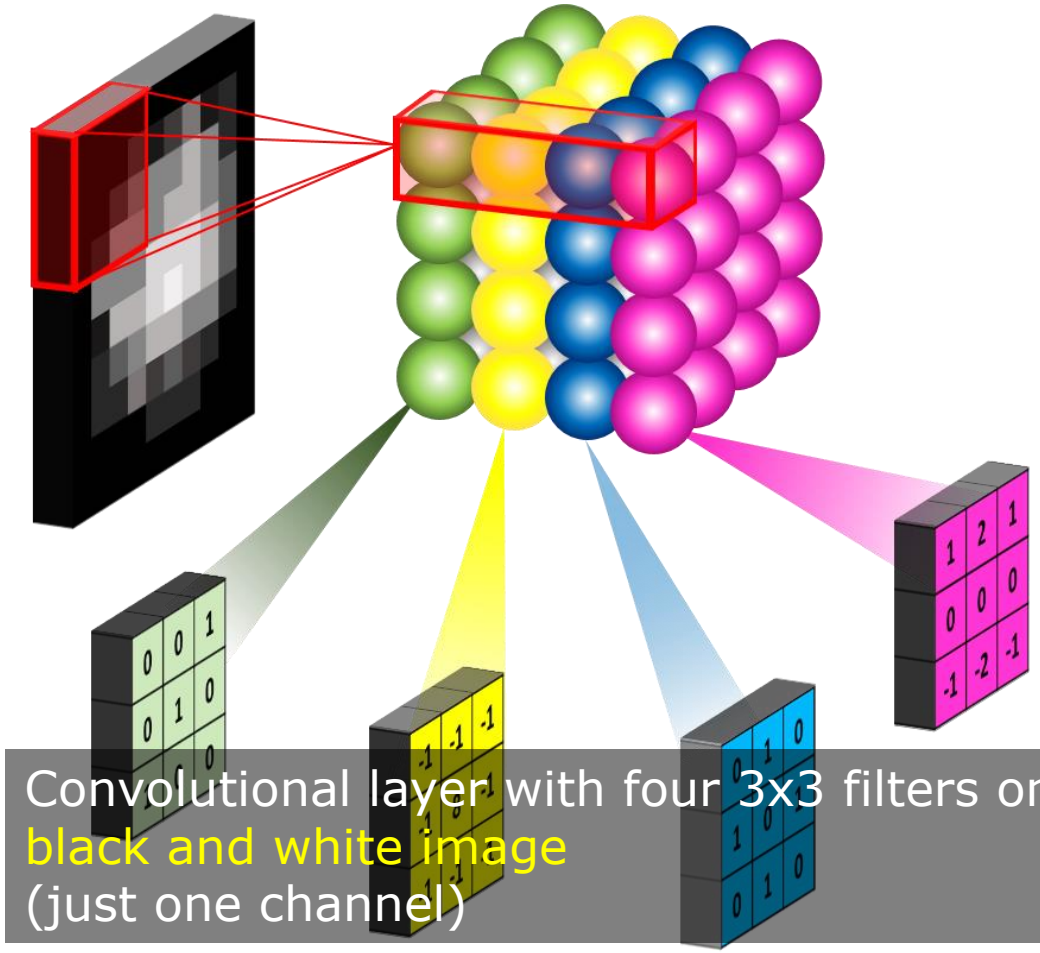
Convolution Filter



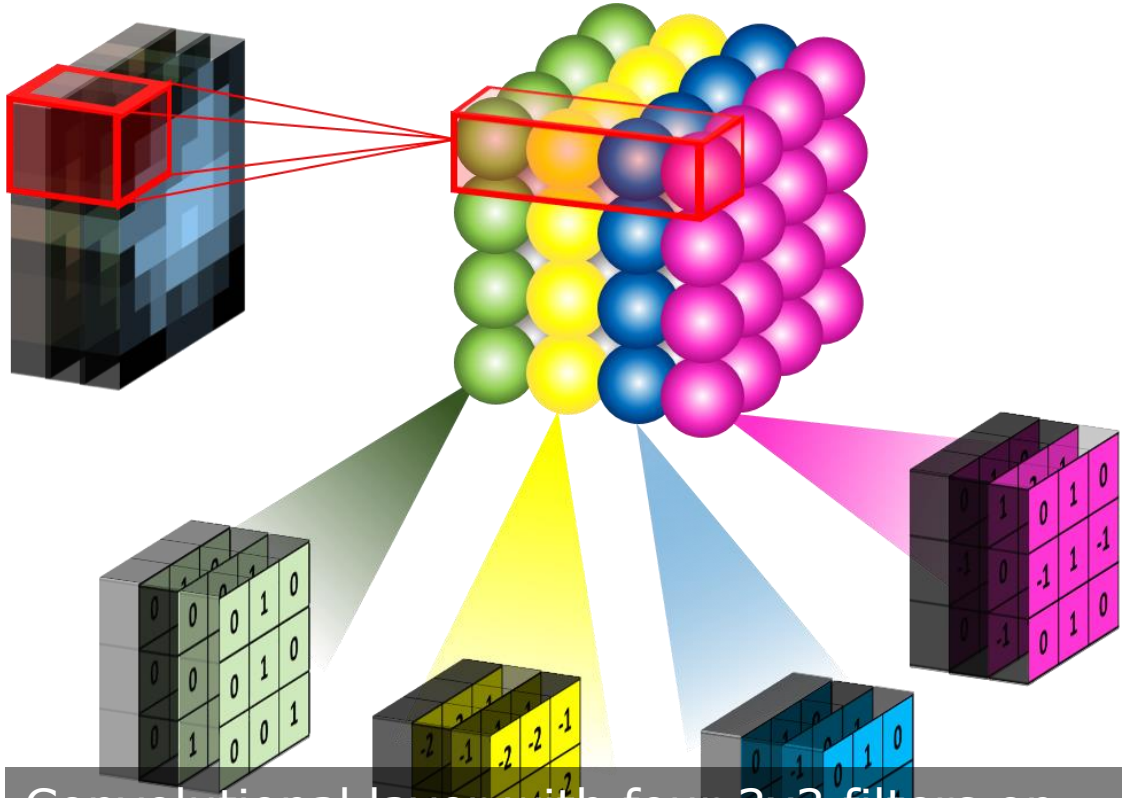


- Demo: https://cs231n.github.io/convolutional-networks/

Convolutional Neural Network - Layers



Convolutional layer with four 3x3 filters on a black and white image (just one channel)

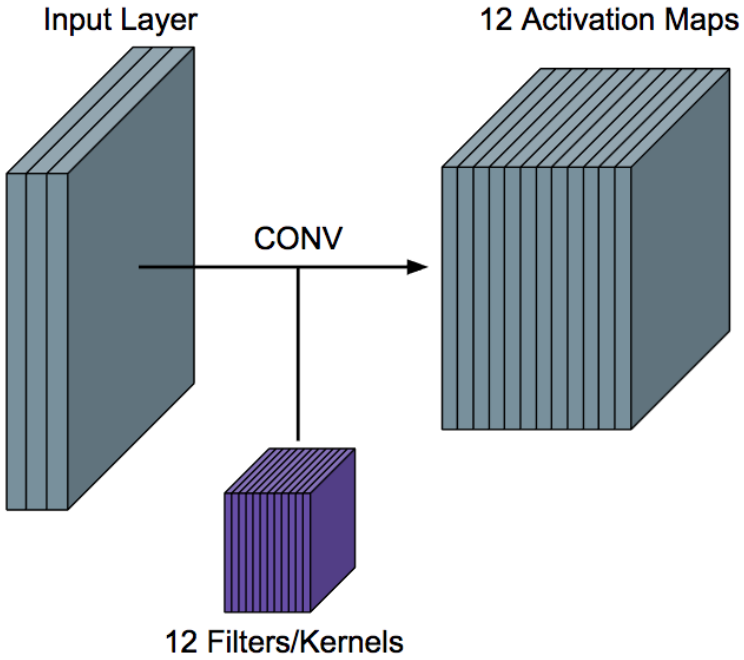


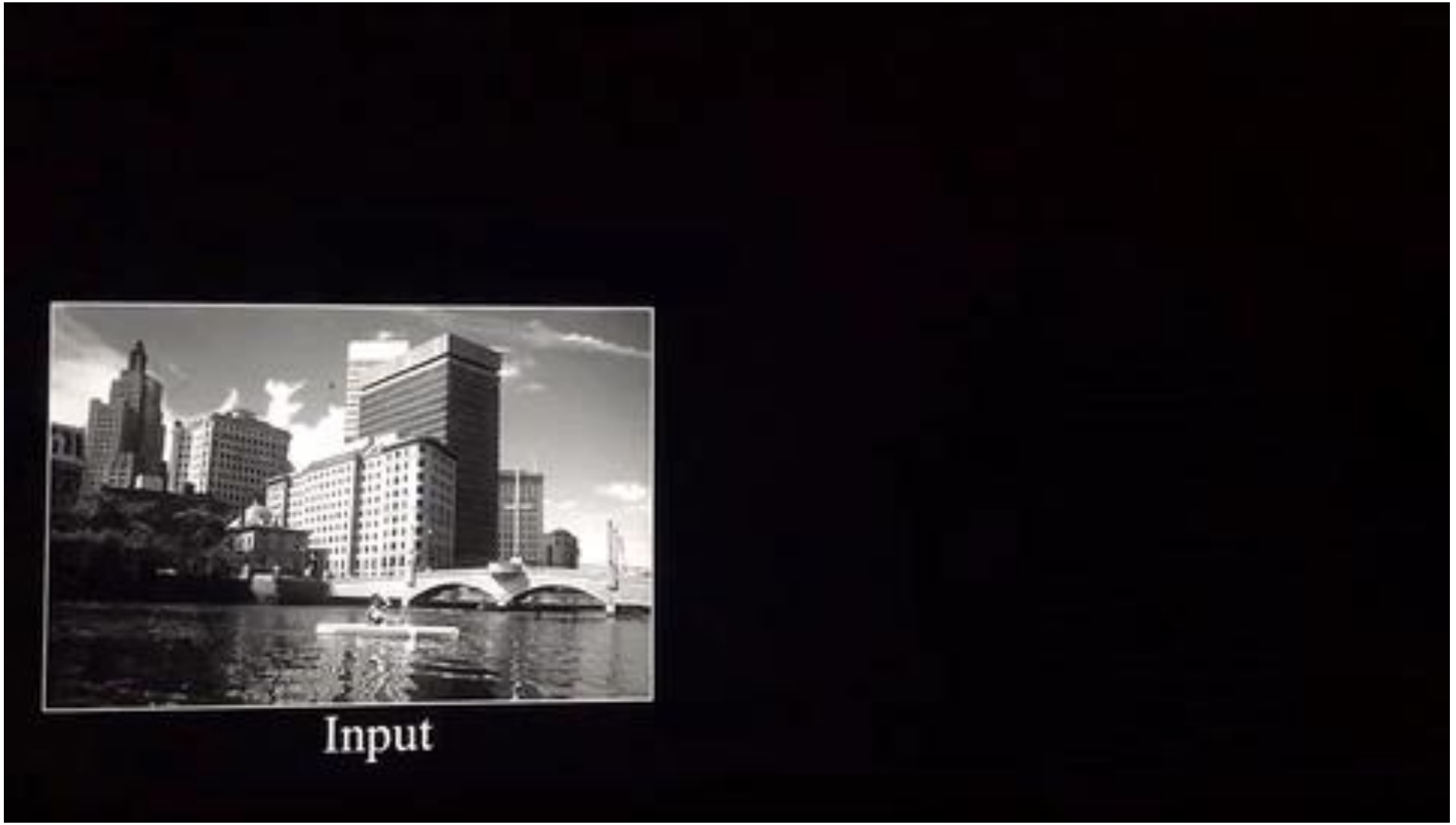
Convolutional layer with four 3x3 filters on an RGB image. Here, the filters are **cubes**, which are applied on the full depth of the image.

Convolutional Neural Network

Filter Convolution

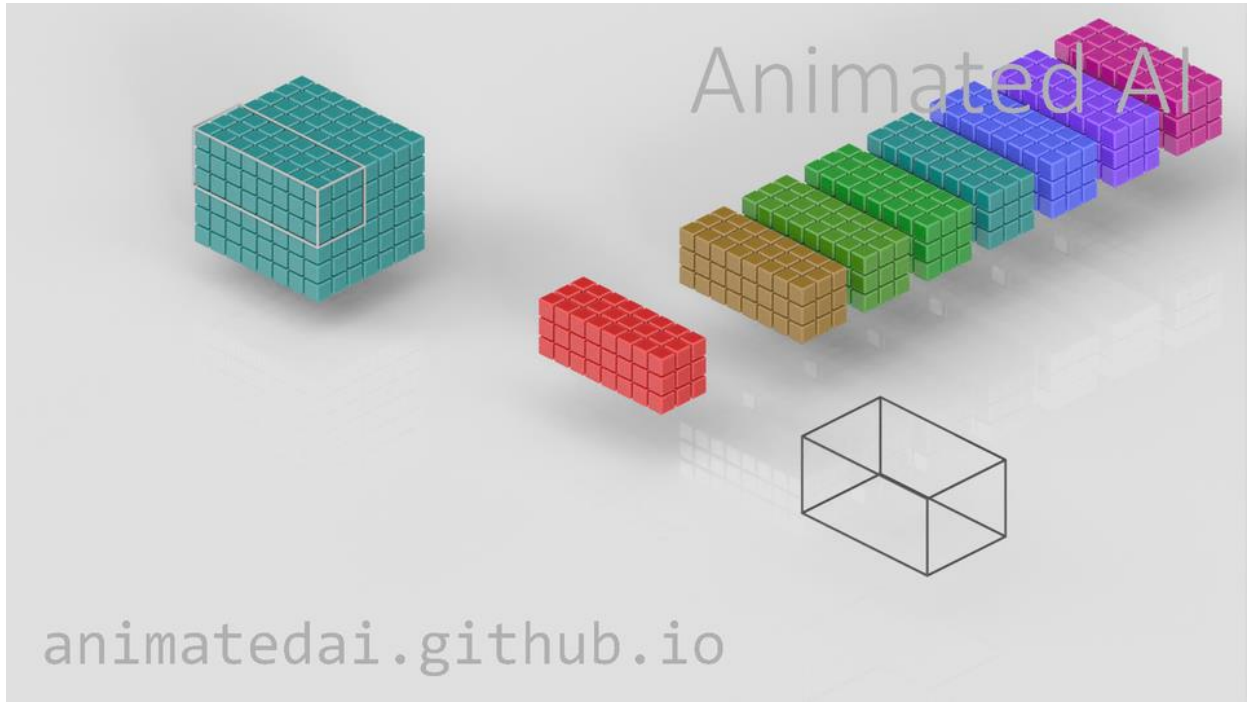
- Filters convolve with the 3D input cube to create *2D feature maps*.
- When using multiple filters, we get a 3D output, with one 2D feature map per filter
- The feature map dimension can vary significantly between convolutional Layers:
  - E.g., a layer input can be 32x32x16 and its output 32x32x128, if that layer has 128 filters.




Input

Basic Convolution

Fundamental Algorithm of Convolution in Neural Networks (youtube.com)

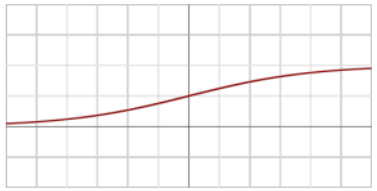


Animated AI

Activation Functions

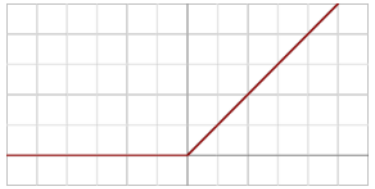
Desired properties

- Cheap to compute
- Non-linear
- Differentiable
- Monotonic
- Bounded output



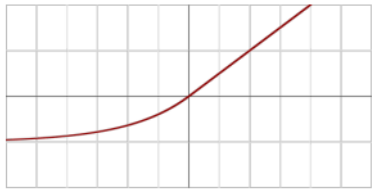
Logistic/Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$



ReLU

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$



ELU

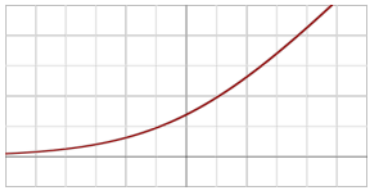
$$f(x) = \begin{cases} x & if\ x \geq 0 \\ \alpha(e^x - 1) & if\ x < 0 \end{cases}$$

Linear

$$f(x) = x$$

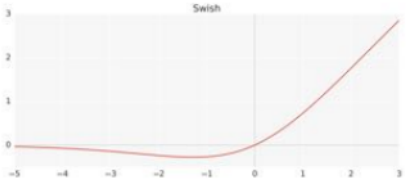
Softmax

$$f(x) = \frac{1}{Z}e^x$$



Softplus

$$f(x) = \ln(1 + e^x)$$

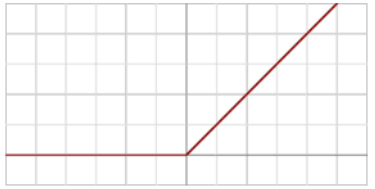


Swish

$$f(x) = \frac{x}{1 + e^{-x}}$$

Activation Functions

Rectified Non-Linear unit (ReLU)



ReLU

$$f(x) = \begin{cases} x & if\ x \geq 0 \\ 0 & if\ x < 0 \end{cases}$$

- Solves the vanishing gradient problem that occurs with *sigmoid*
- Easy to compute:

$$Output = \max(0, input)$$

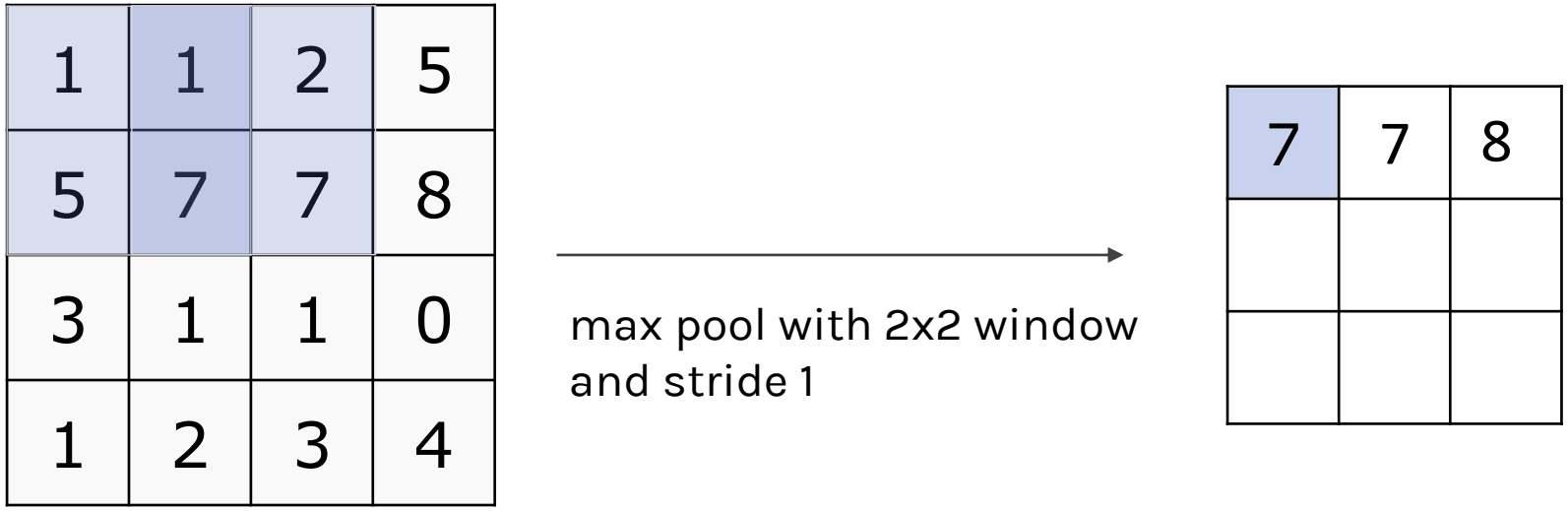
- Generates sparsity (not always beneficial)

Convolutional Layers - Recap

- A convolutional layer convolves each of its filters with the input.
- Input -
  - a 3D tensor: Width × Height × Channels (or Feature Maps)
- Output -
  - a 3D tensor: Width × Height × Feature Maps (one for each filter)
- Applies non-linear activation function (usually ReLU) over each value of the output.
- Multiple (hyper)parameters to be defined:
  - # of filters
  - Filters size
  - Stride
  - Padding
  - Activation function
  - Regularization.

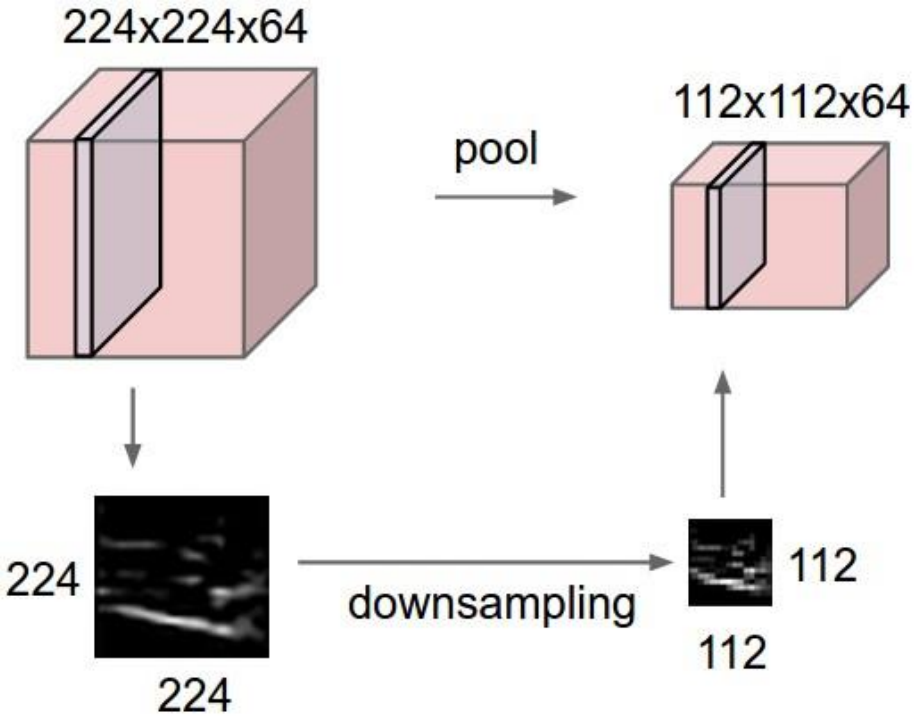
Pooling

- Invariant to small, "local transitions"
- Reduces input size to final fully connected layers
- No learnable parameters



Pooling Layer

Max or Average → down sample layers



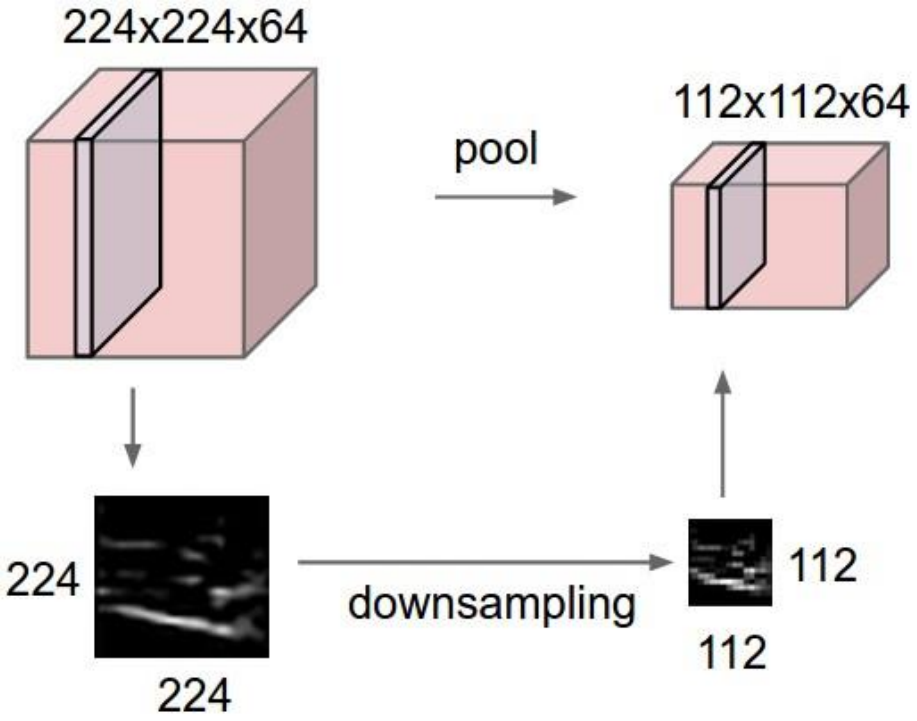
1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

max pool with 2x2 window and stride 2

7	8
3	4

Pooling Layer

Max or Average → down sample layers

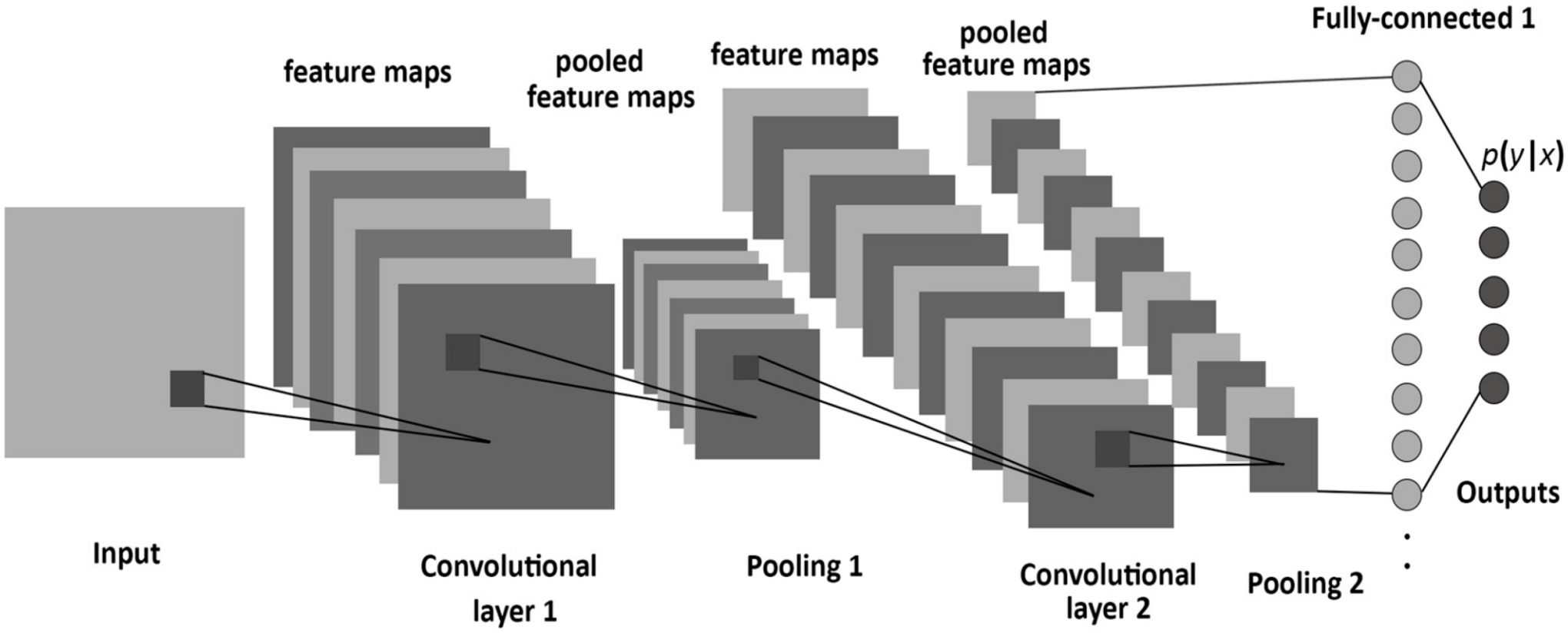


1	1	2	5
5	7	7	8
3	1	1	0
1	2	3	4

mean pool with 2x2 window and stride 2

4	6
2	2

A Convolutional Network



CNN Building Blocks

CNN is built by stacking layers, typically of 3 types

Convolutional Layers

Pooling Layers

Fully Connected Layers

Convolutional Layers

Action

- Apply filters to extract features
- Filters are composed of small kernels, learned.
- One bias per filter.
- Apply activation function on every value of feature map

Parameters

- Number of kernels
- Size of kernels (W and H only, D is defined by input cube)
- Activation function
- Stride
- Padding
- Regularization type and value

I/O

- Input:

  3D cube -

  previous set of feature maps
- Output:

  3D cube -

  one 2D map per filter

CNN Building Blocks

CNN is built by stacking layers, typically of 3 types

Convolutional Layers

Pooling Layers

Fully Connected Layers

Pooling Layers

Action

- Reduce dimensionality
- Extract maximum of average of a region
- Sliding window approach

Parameters

- Stride
- Size of window

I/O

- Input:

  3D cube, previous set of feature maps

- Output:

  3D cube,
  one 2D map per filter,
  reduced spatial dimensions

CNN Building Blocks

CNN is built by stacking layers, typically of 3 types

Convolutional Layers

Pooling Layers

Fully Connected Layers

Fully Connected Layers

Action

- Aggregate information from final feature maps
- Generate final classification

Parameters

- Number of nodes
- Activation function: changes depending on role of layer.
- If aggregating info, use ReLU.
- If producing final classification, use Softmax.

I/O

- Input: FLATTENED 3D cube, previous set of feature maps
- Output: 3D cube, one 2D map per filter

CNN Building Blocks

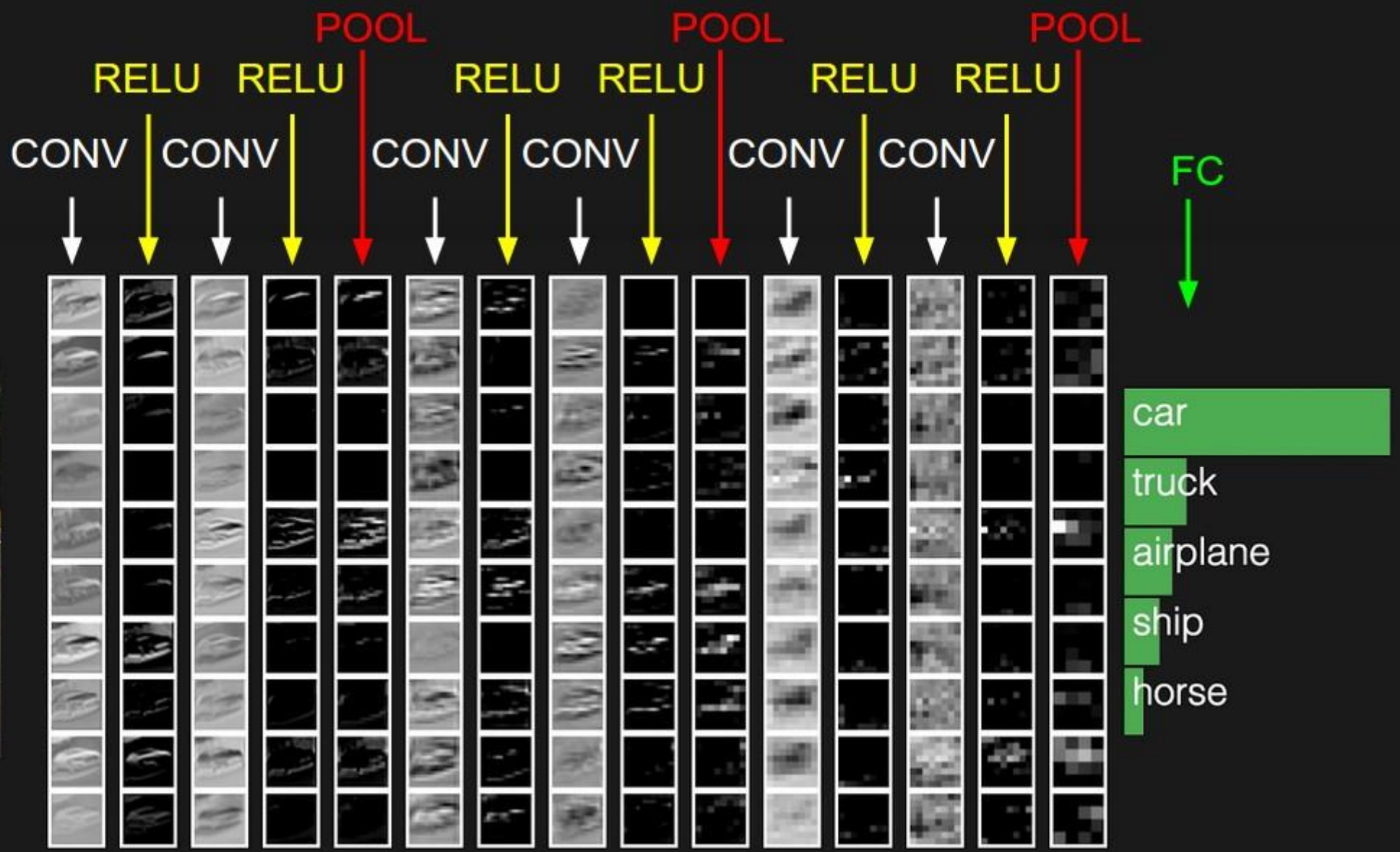
CNN is built by stacking layers, typically of 3 types

Convolutional Layers

Pooling Layers

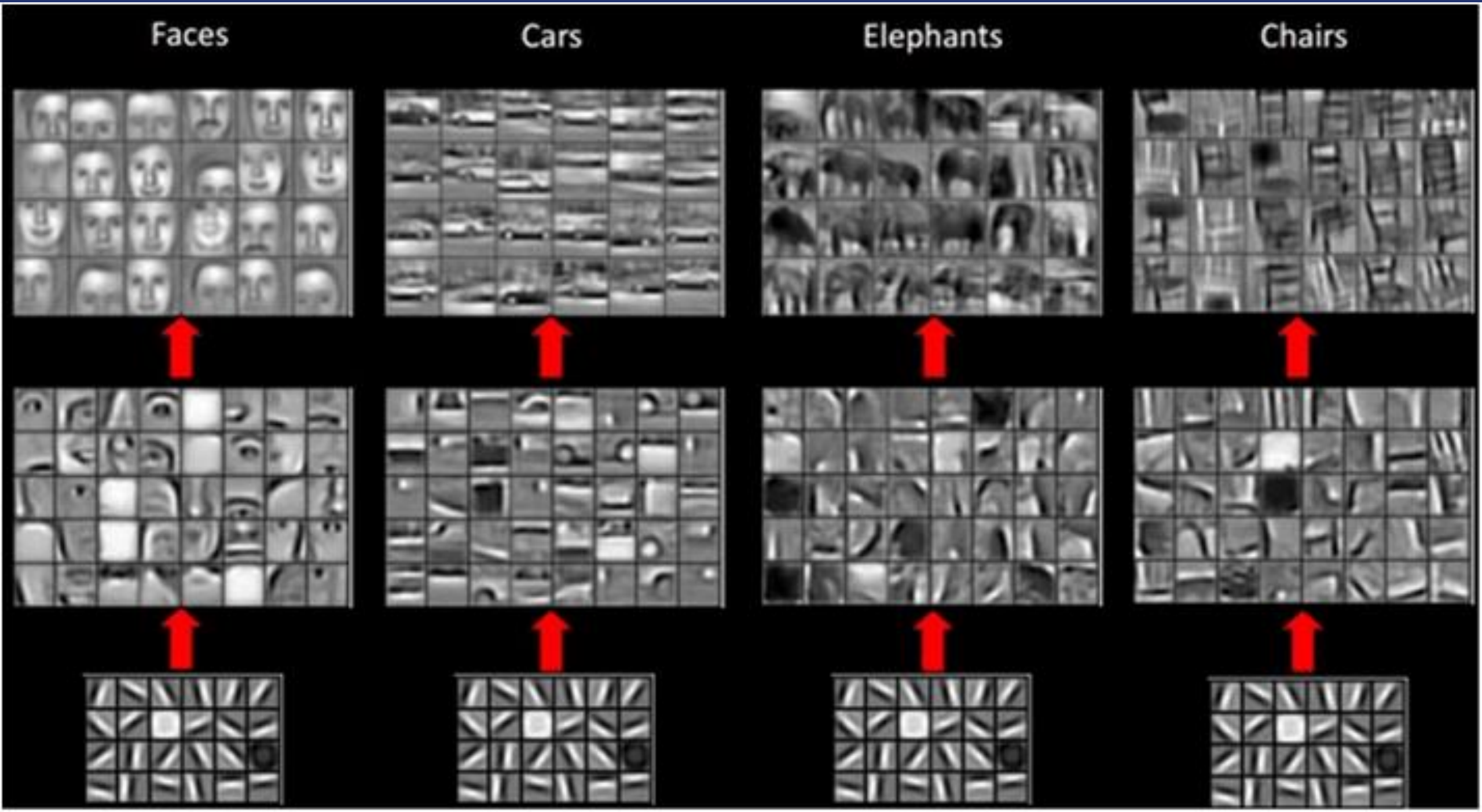
Fully Connected Layers

Live example: http://cs231n.stanford.edu/



What Do CNN Layers Learn?

- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters:
  - edges, corners, etc.
- The middle layers learn filters that detect complex textures and parts of objects.
  - E.g., for faces, they might learn to respond to eyes, noses, etc.
- The last layers have higher representations:
  - they learn to recognize full objects, in different shapes and positions.


Faces
Cars
Elephants
Chairs

Simple CNN in PyTorch

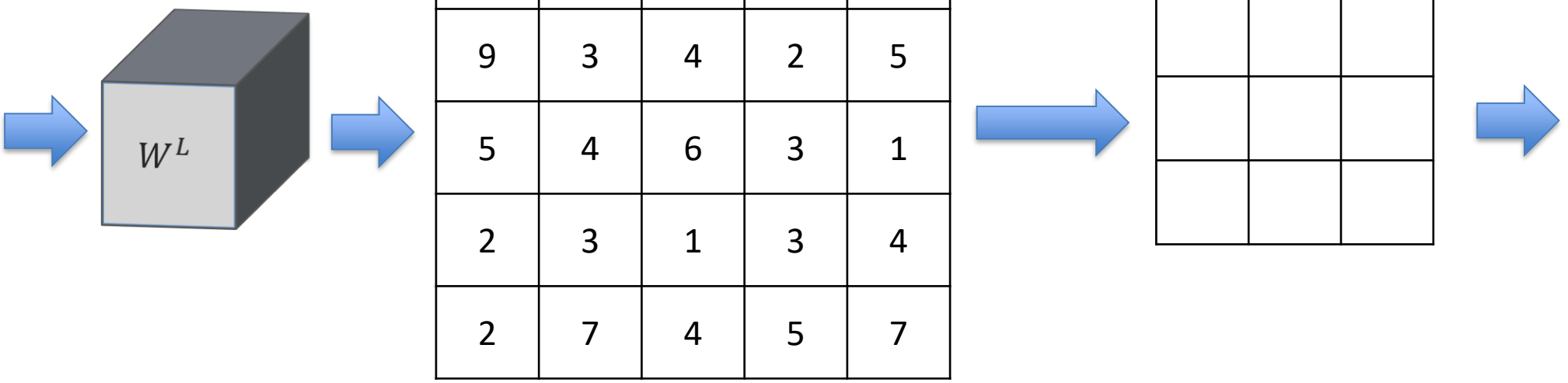
Training a Classifier — PyTorch Tutorials

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

Backward propagation of Maximum Pooling Layer

Forward mode, 3x3 stride 1

rest of the network

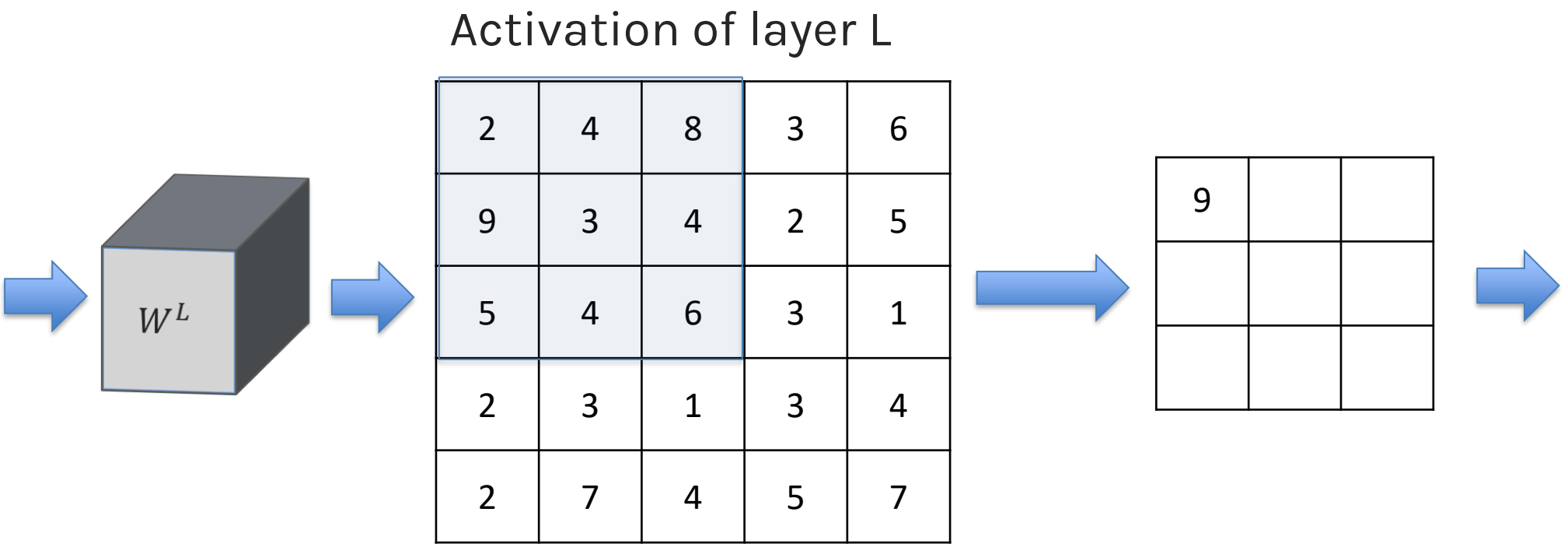


rest of the network

Backward propagation of Maximum Pooling Layer

Forward mode, 3x3 stride 1

rest of the network

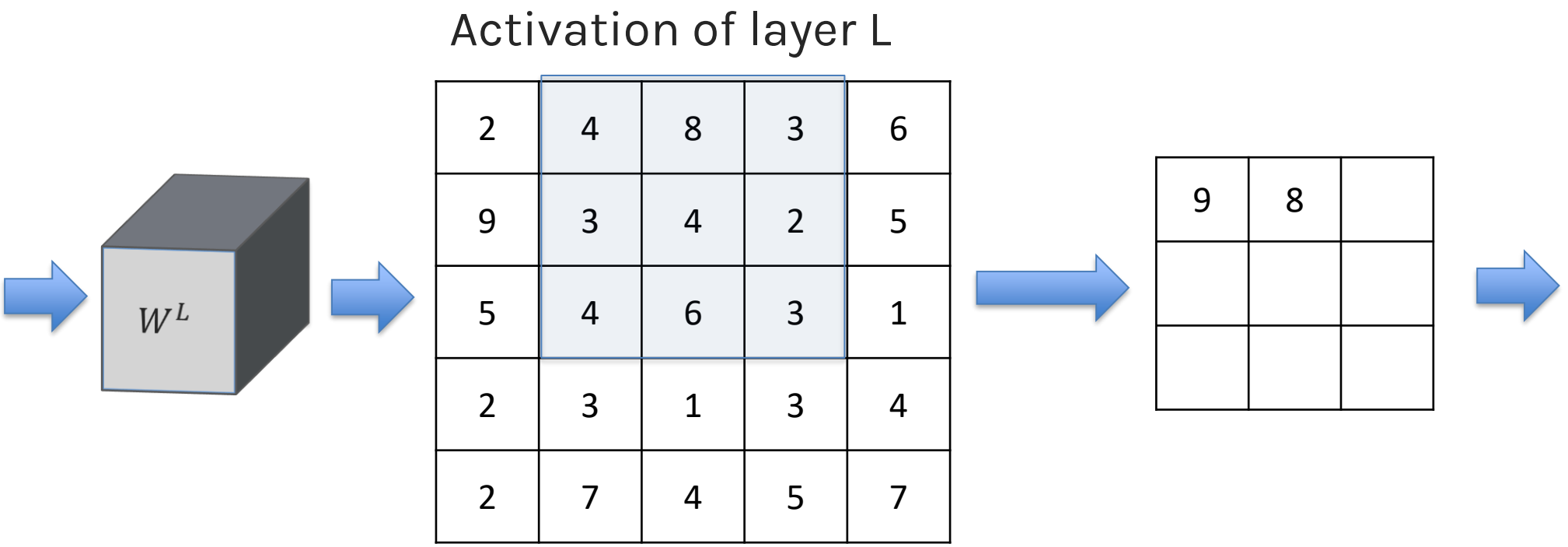


rest of the network

Backward propagation of Maximum Pooling Layer

Forward mode, 3x3 stride 1

rest of the network



rest of the network

Backward propagation of Maximum Pooling Layer

Forward mode, 3x3 stride 1

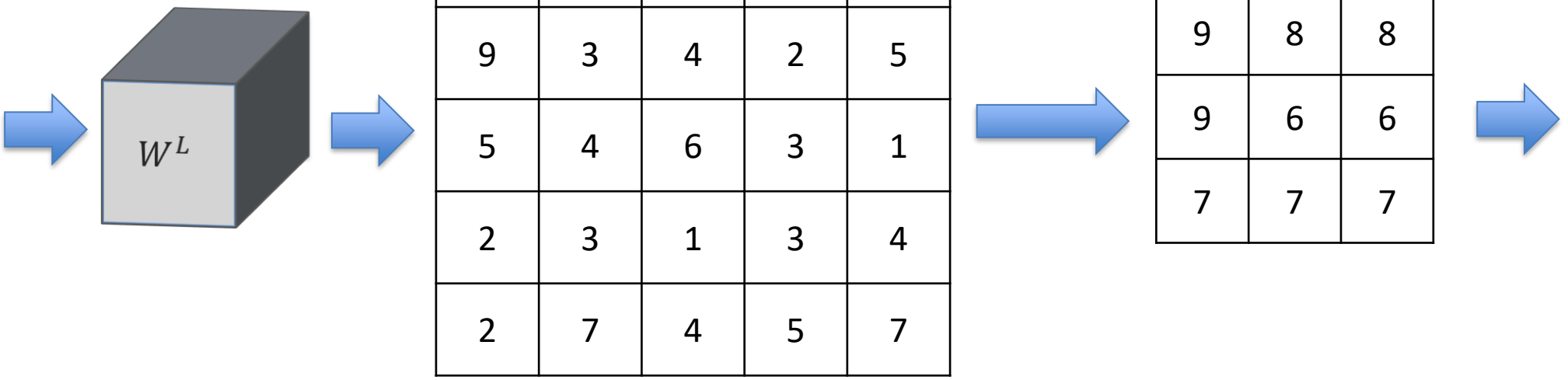
rest of the network

rest of the network

Backward propagation of Maximum Pooling Layer

Forward mode, 3x3 stride 1

rest of the network



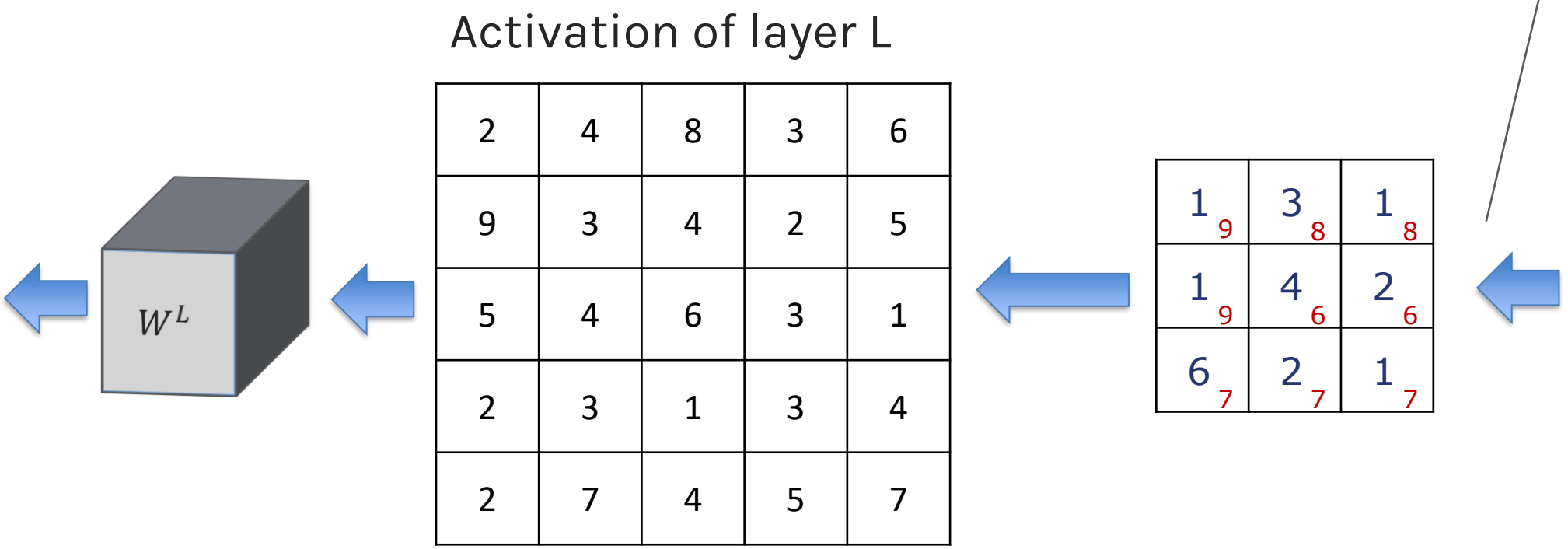
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



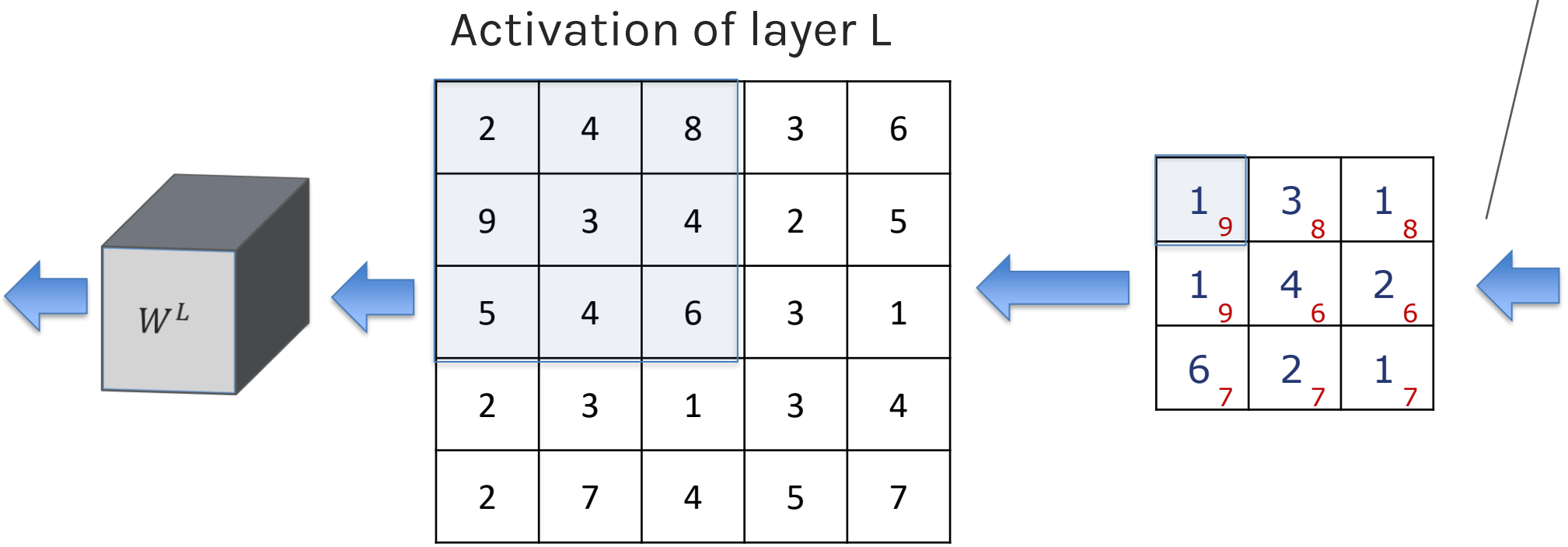
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



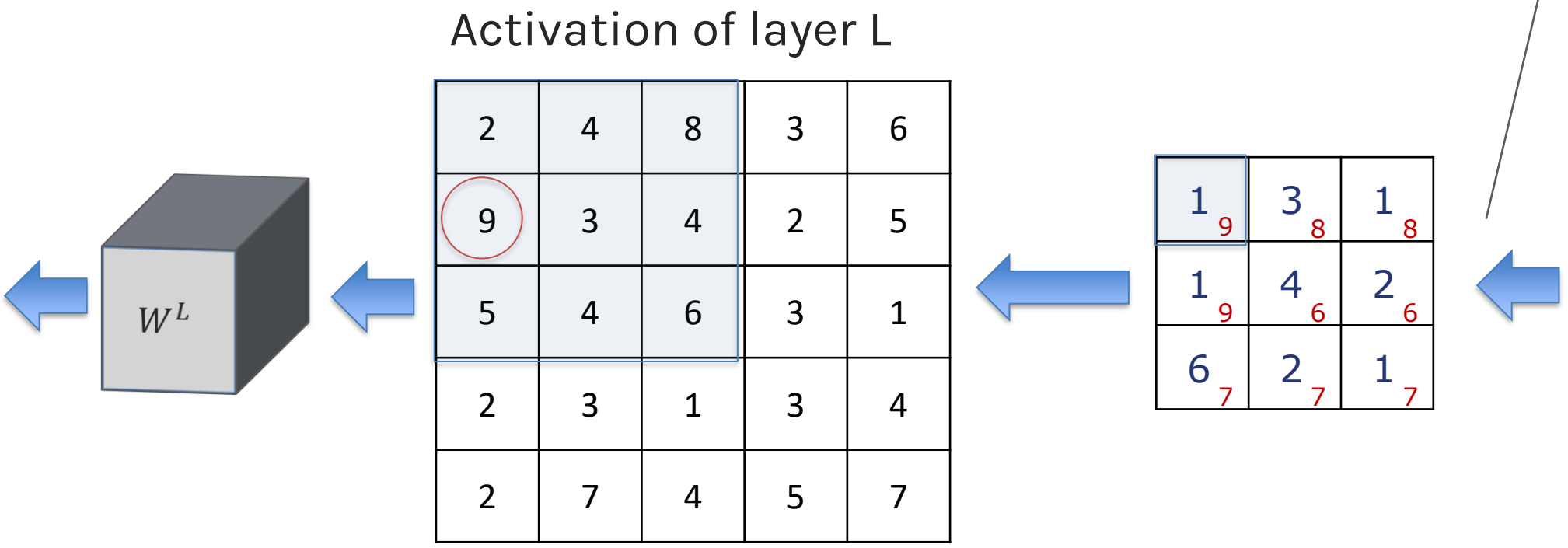
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



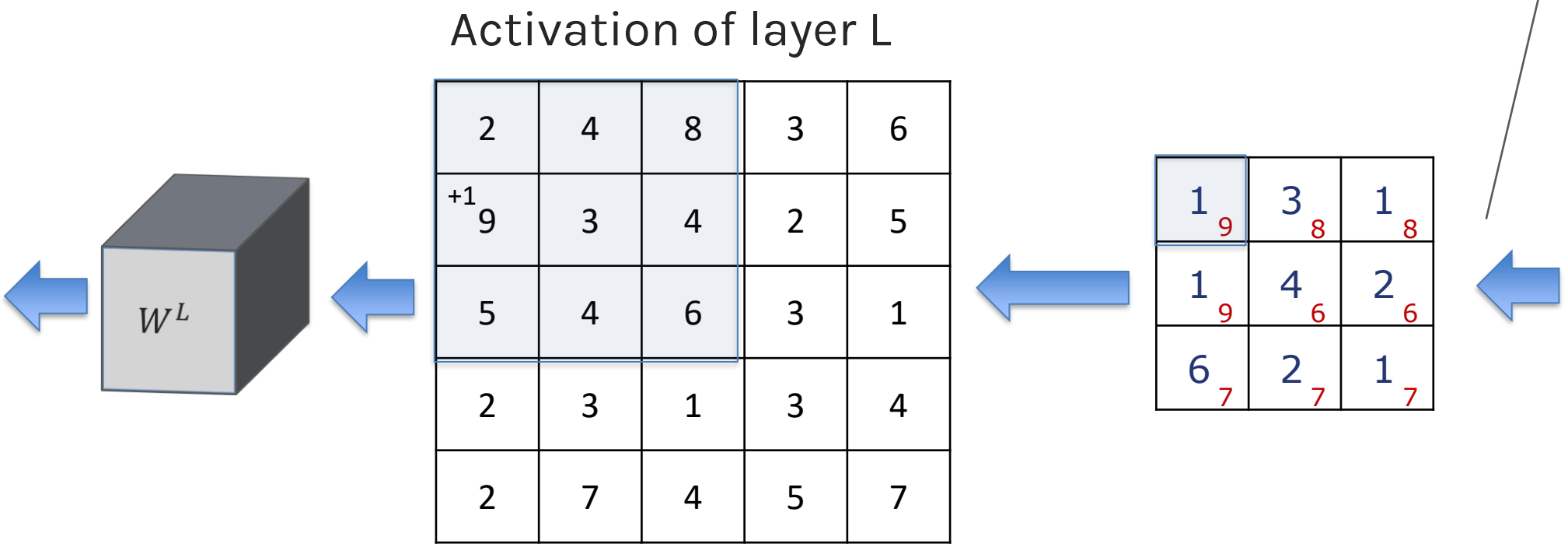
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network

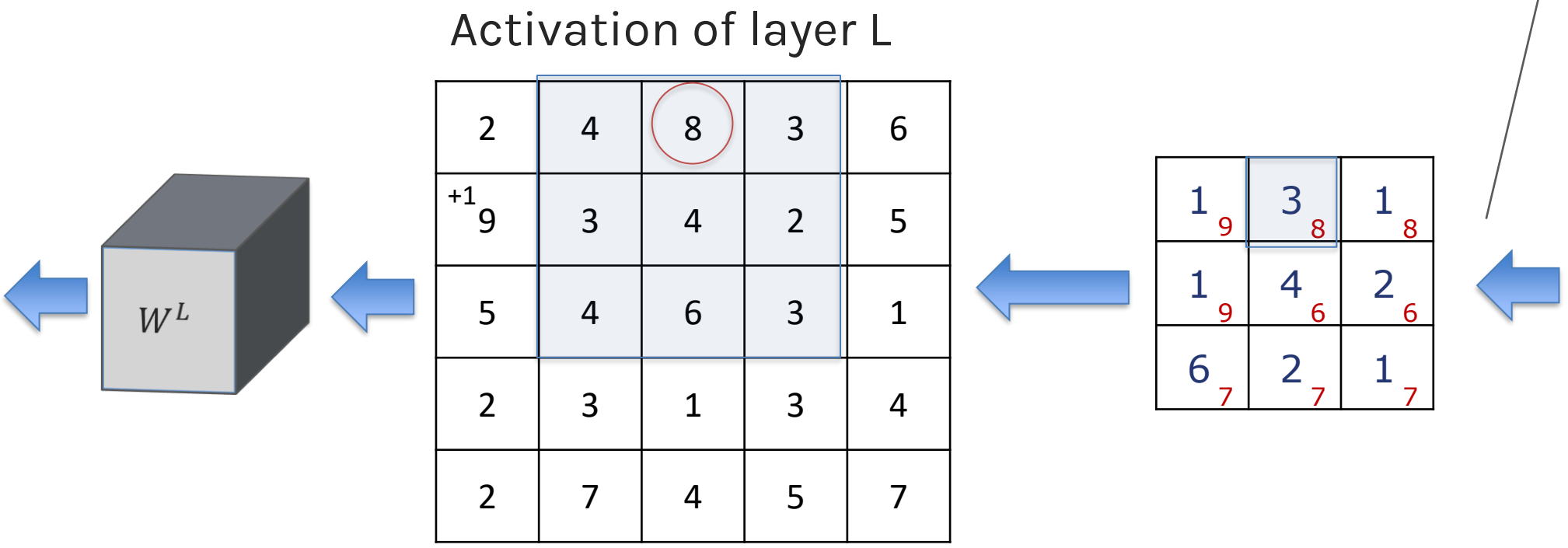
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network

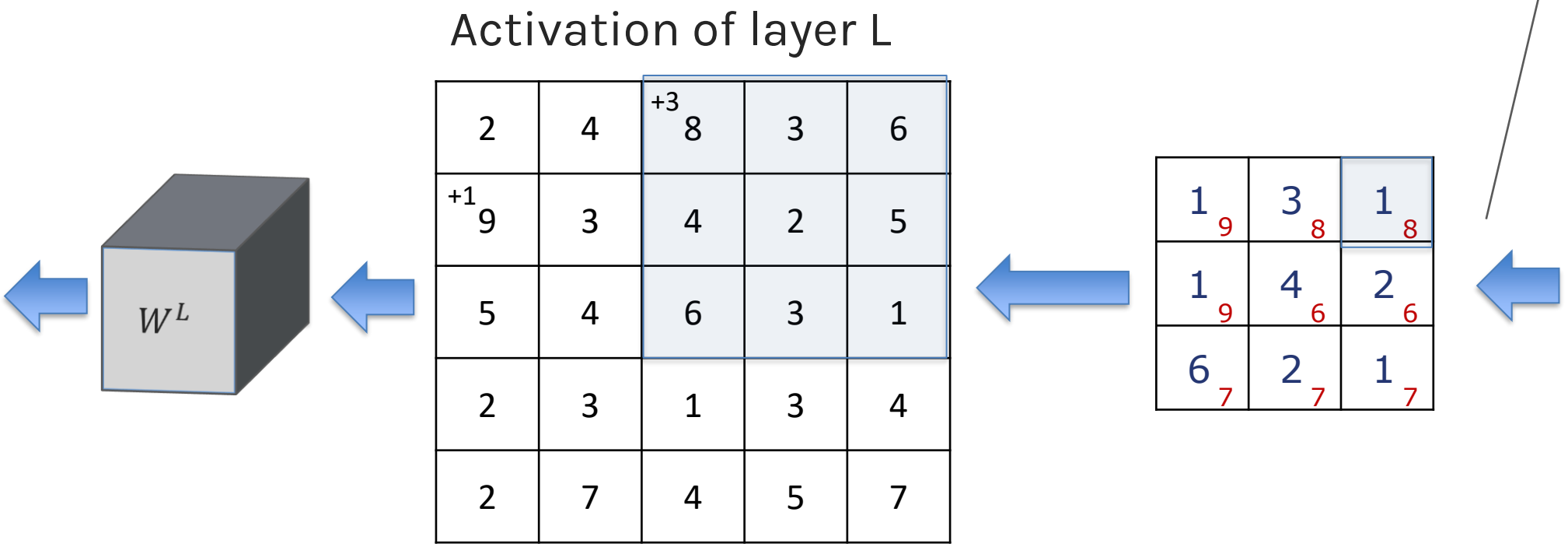
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



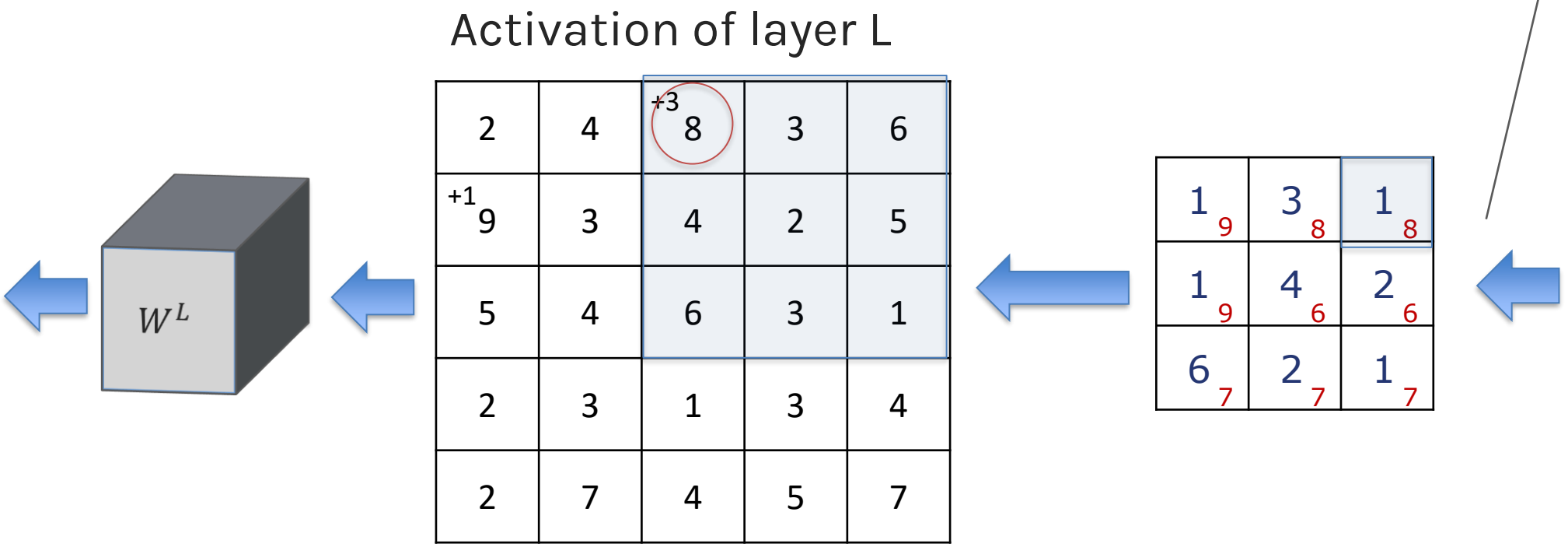
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



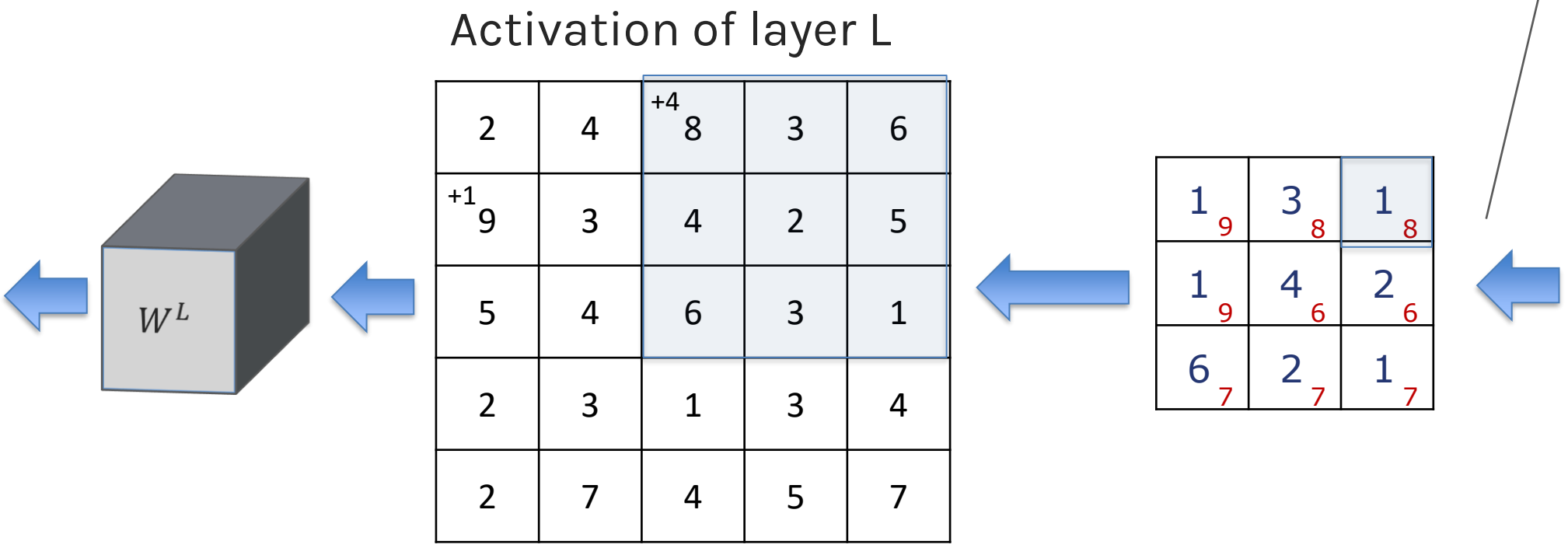
rest of the network

Blue: the current layer's derivatives (max-pool)
Red: the corresponding value of the previous layer.

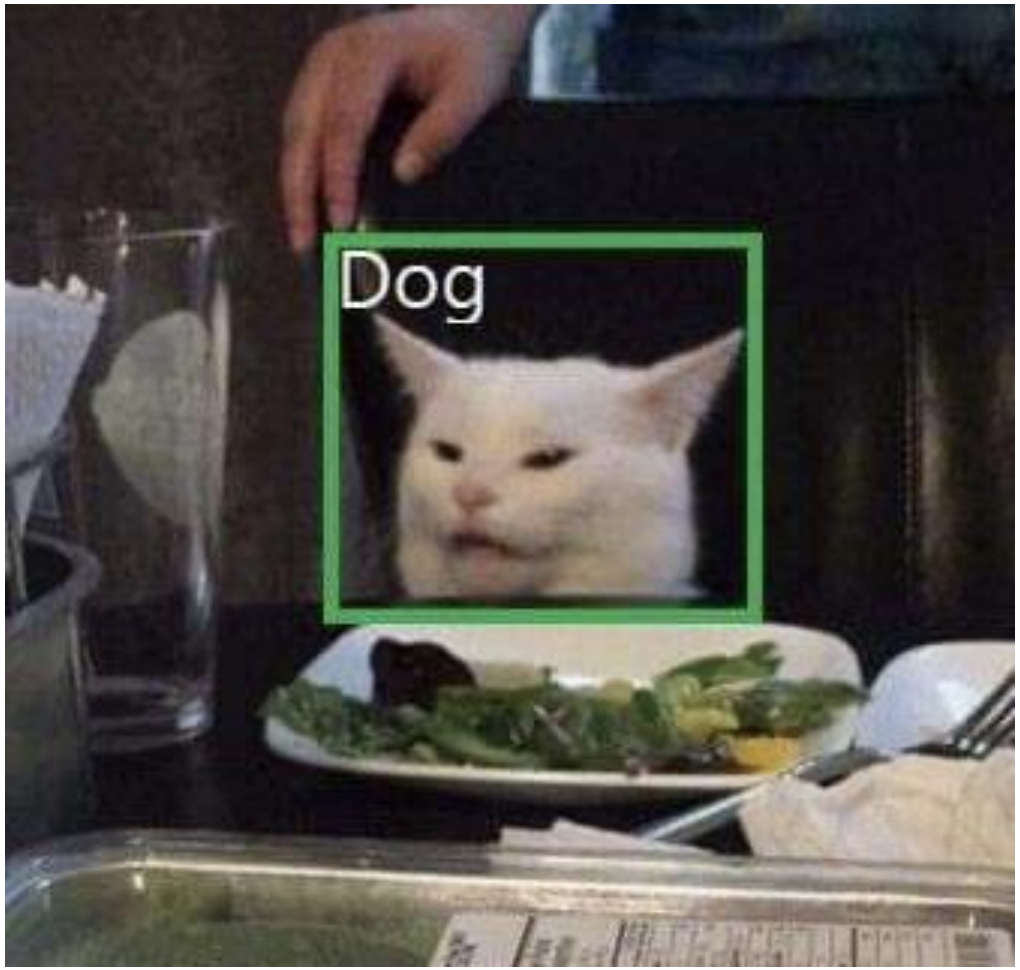
Backward propagation of Maximum Pooling Layer

Backward mode.

rest of the network



rest of the network


Dog

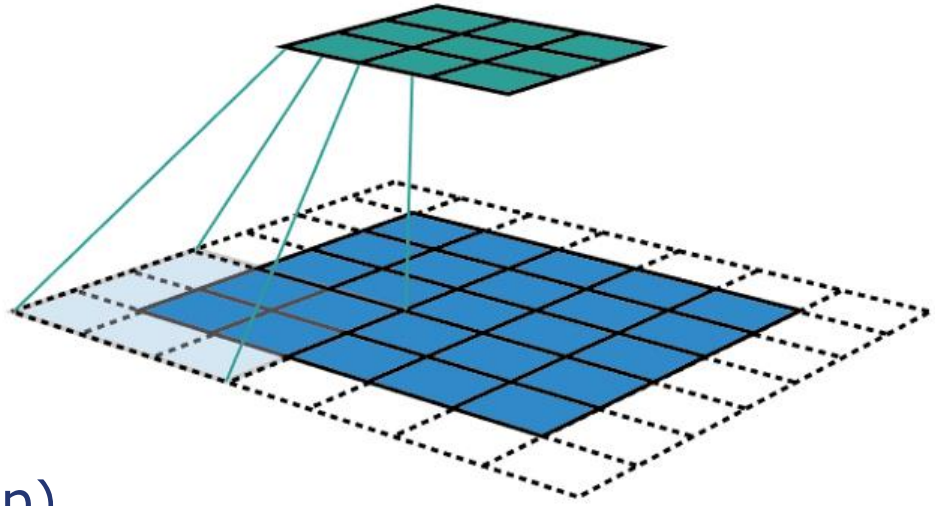
Layer's **Receptive Field**

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: # of input features
n_{out}: # of output features
k: convolution **kernel** size
p: convolution **padding** size
s: convolution **stride** size

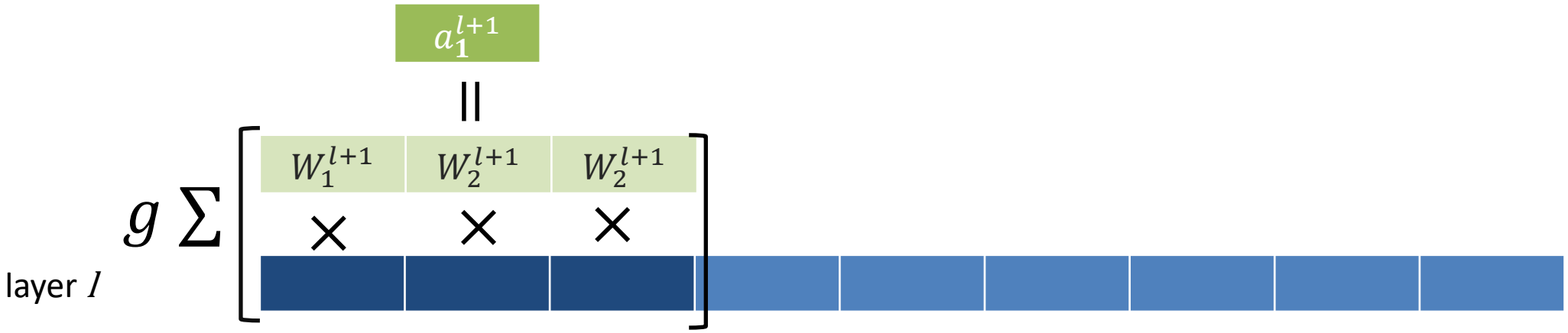
- The **receptive field** is the region in the input space that a particular CNN's feature is looking at (i.e. is affected by).

- Apply a convolution C with
  kernel size ***k* = 3x3**,
  padding size ***p* = 1x1**,
  stride ***s* = 2x2**
  on an input map **5x5**,
  we will get an output feature map **3x3** (green map).



Layer's Receptive Field

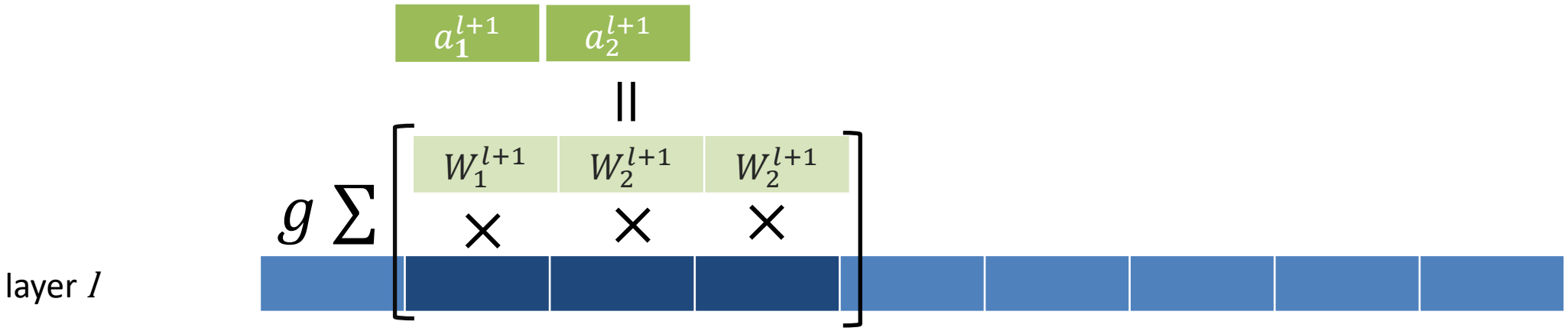
Receptive field in 1D: no padding, stride 1 and kernel 3x1



How to Calculate Receptive Field Size in CNN | Baeldung on Computer Science

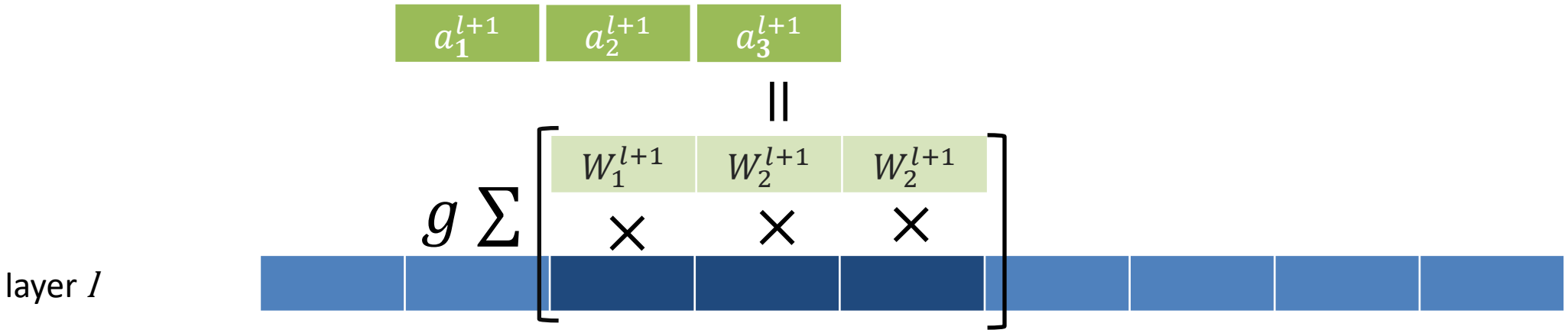
Layer's Receptive Field

Receptive field in 1D: no padding, stride 1 and kernel 3x1



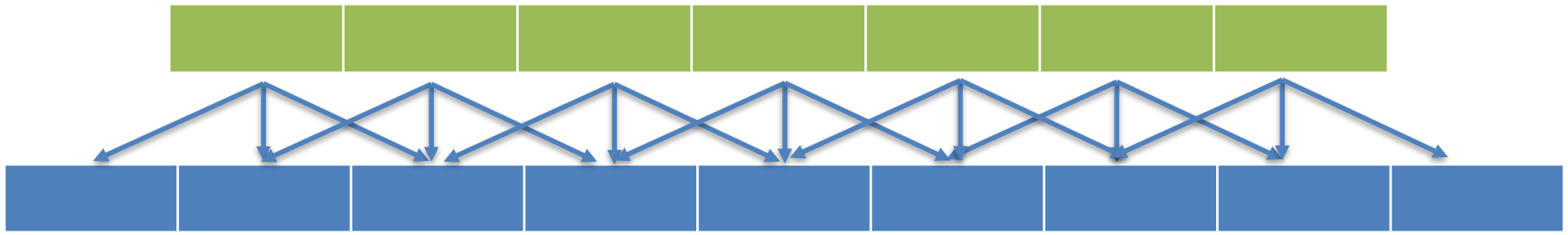
Layer's Receptive Field

Receptive field in 1D: no padding, stride 1 and kernel 3x1



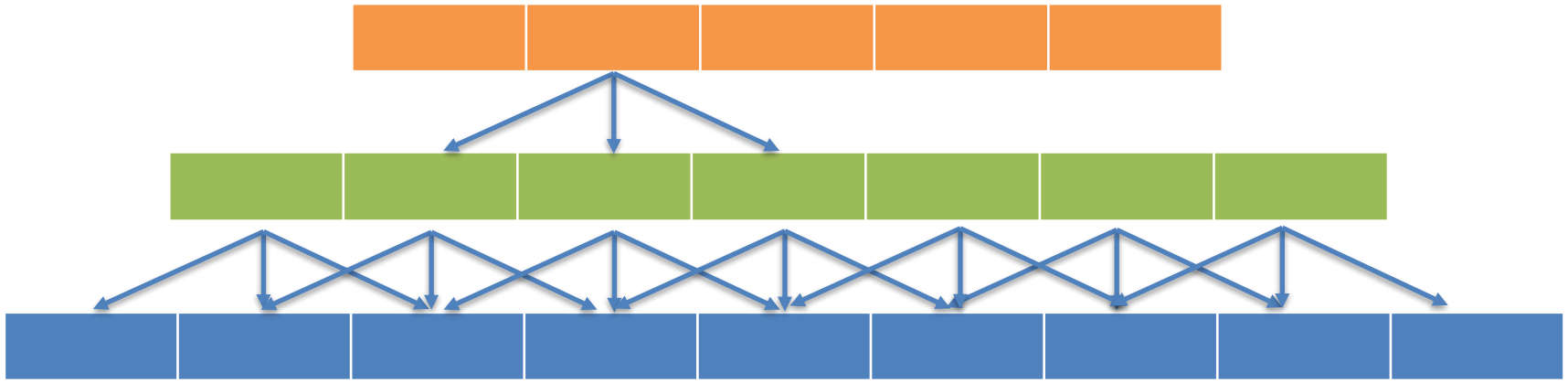
Layer's Receptive Field

Receptive field in 1D: no padding, stride 1 and kernel 3x1



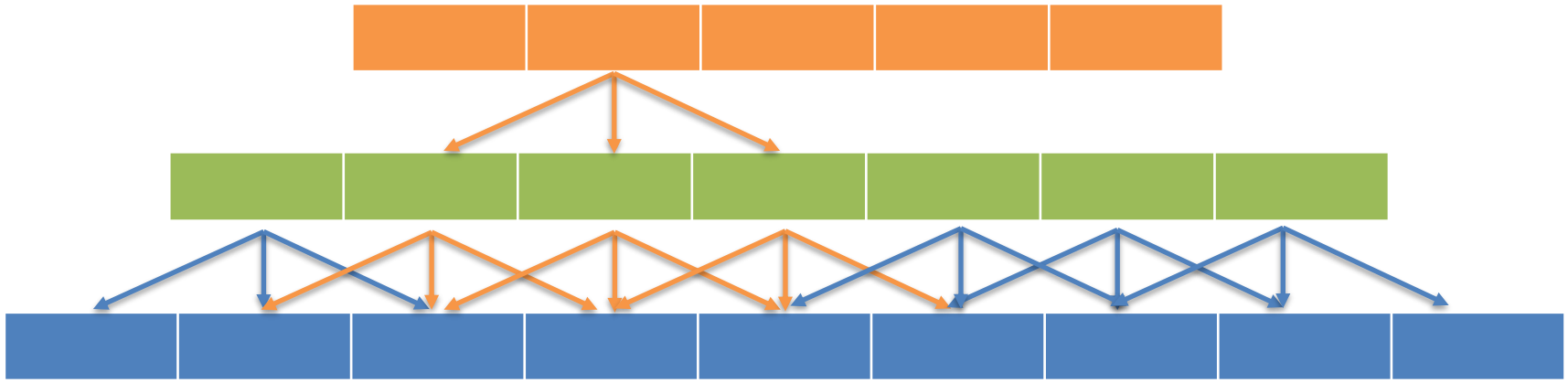
Layer's Receptive Field

Receptive field in 1D: no padding, stride 1 and kernel 3x1



Layer's Receptive Field

Receptive field in 1D: no padding, stride 1 and kernel 3x1



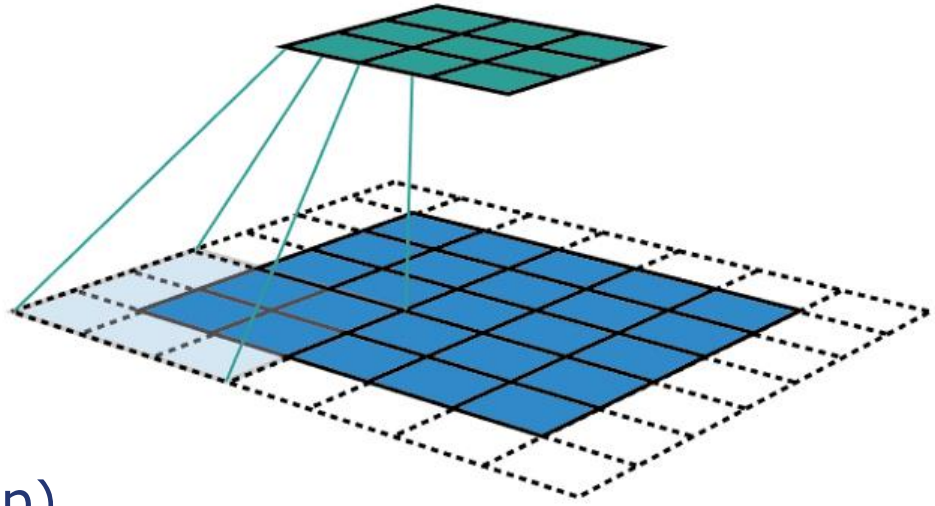
Layer's **Receptive Field**

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: # of input features
n_{out}: # of output features
k: convolution **kernel** size
p: convolution **padding** size
s: convolution **stride** size

- The **receptive field** is the region in the input space that a particular CNN's feature is looking at (i.e. is affected by).

- Apply a convolution C with
  kernel size ***k* = 3x3**,
  padding size ***p* = 1x1**,
  stride ***s* = 2x2**
  on an input map **5x5**,
  we will get an output feature map **3x3** (green map).



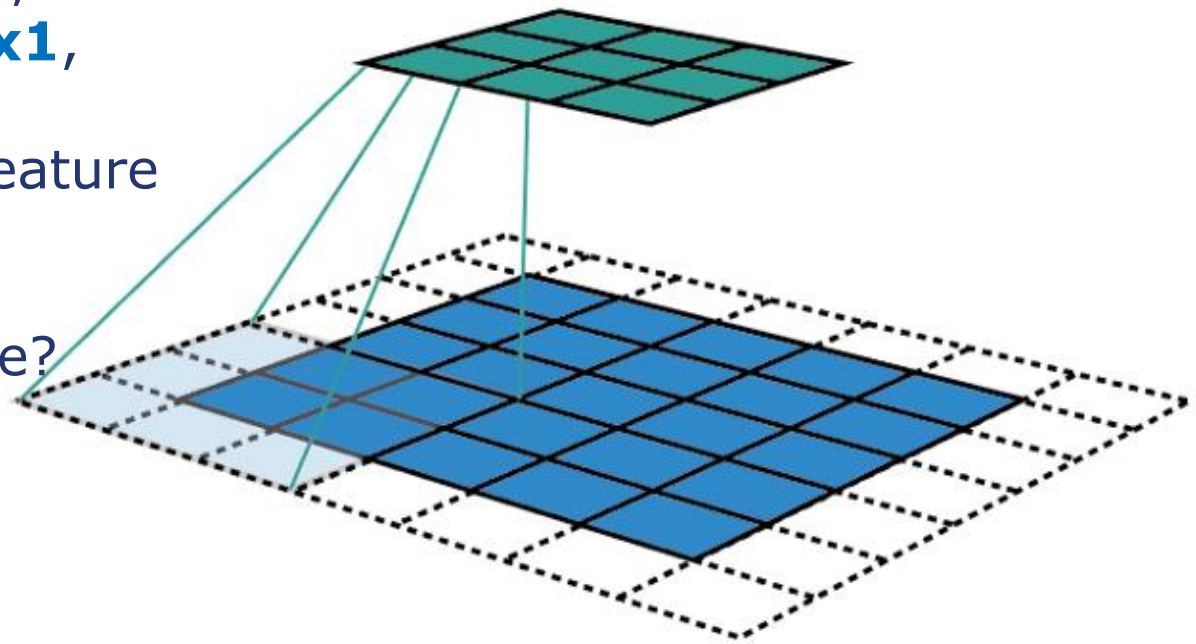
Layer's **Receptive Field**

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: # of input features
n_{out}: # of output features
k: convolution **kernel** size
p: convolution **padding** size
s: convolution **stride** size

We apply the convolution C with kernel size ***k* = 3x3**, padding size ***p* = 1x1**, stride ***s* = 2x2** on the **3x3** green feature map.

What is the new size?

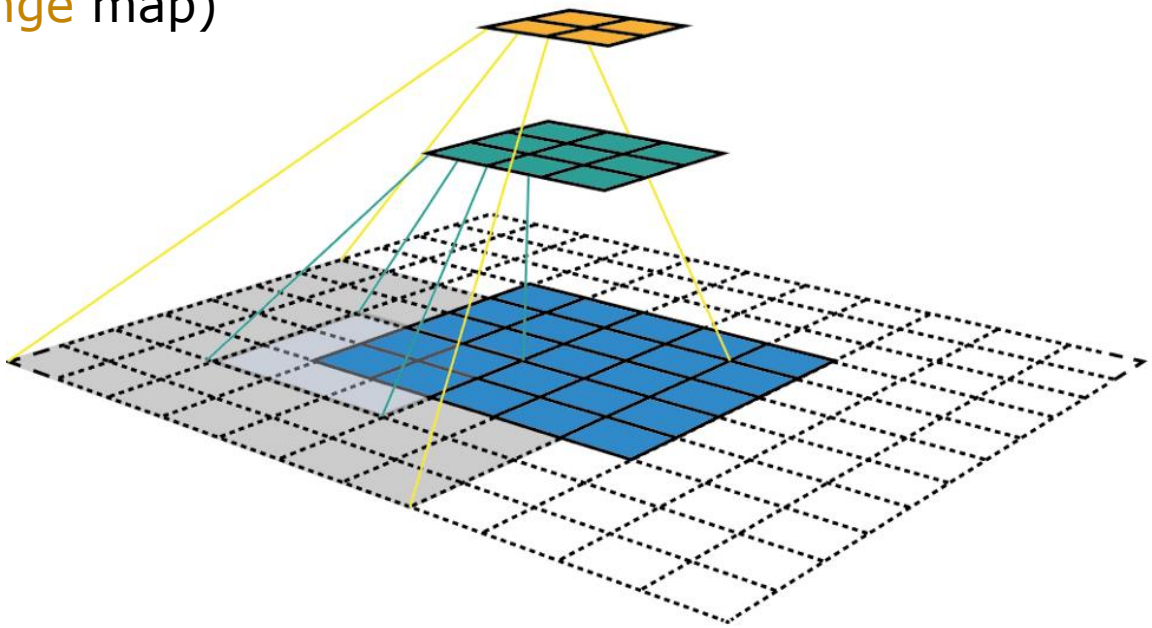


Layer's **Receptive Field**

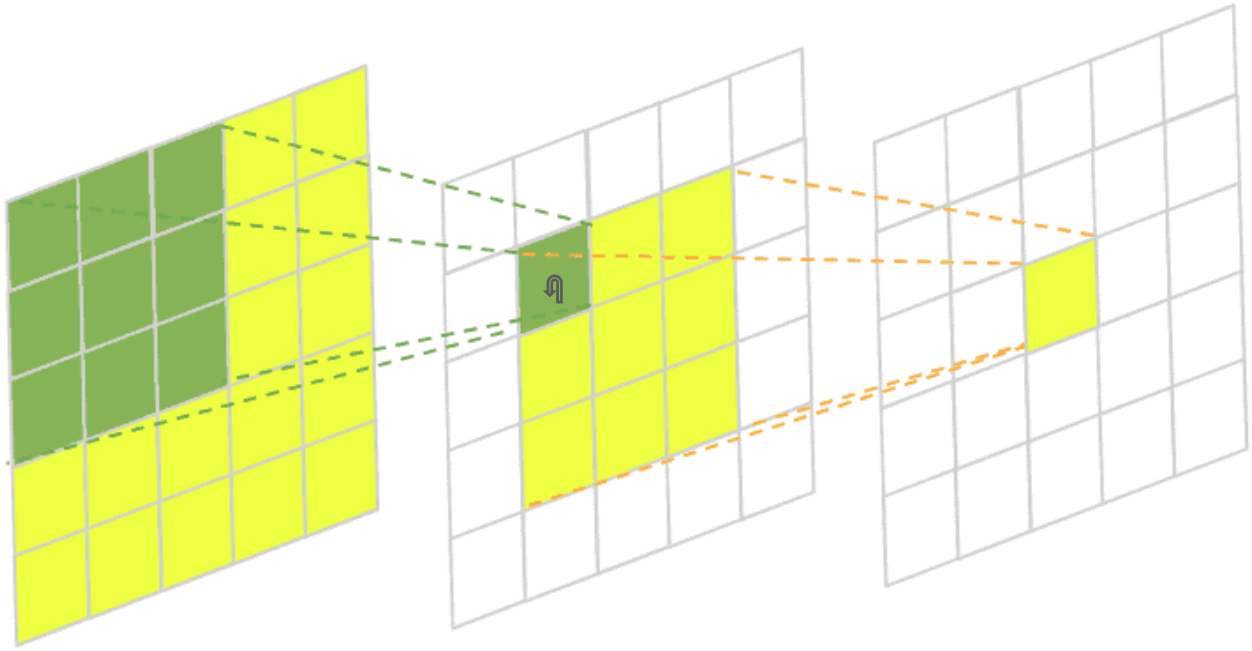
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: # of input features
n_{out}: # of output features
k: convolution **kernel** size
p: convolution **padding** size
s: convolution **stride** size

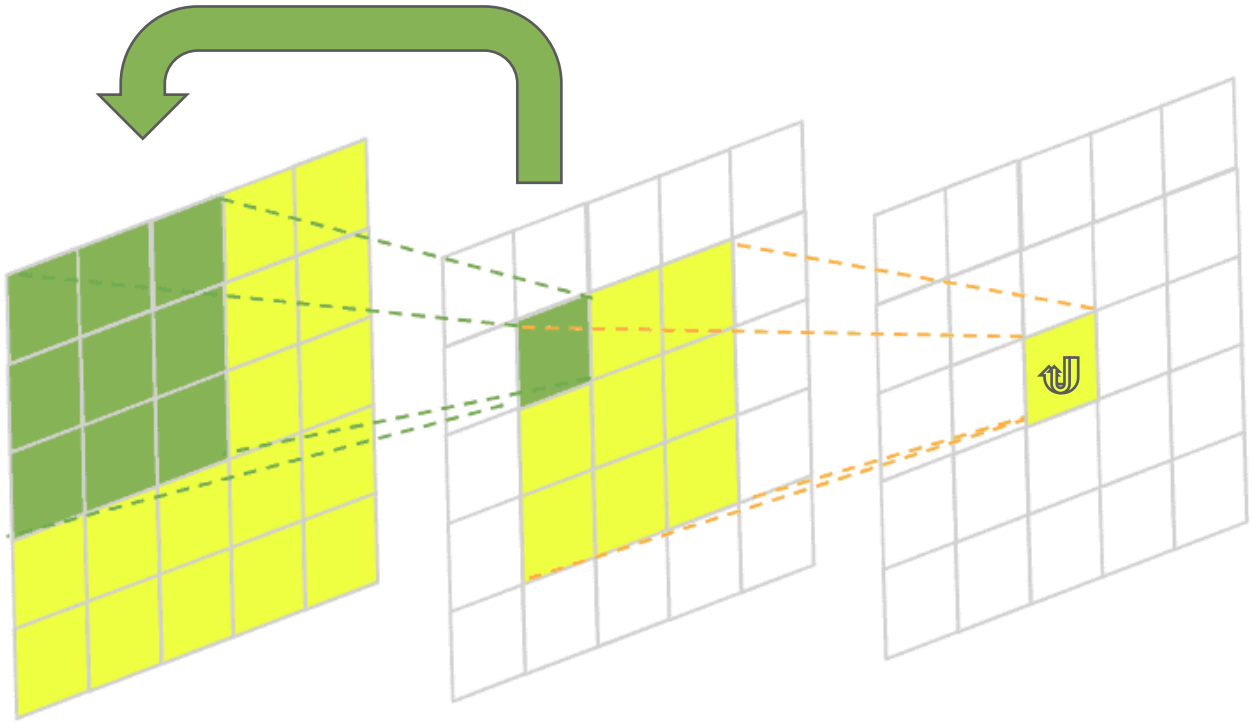
After applying the same convolution on top of the 3x3 feature map, we will get a **2x2** feature map (orange map)



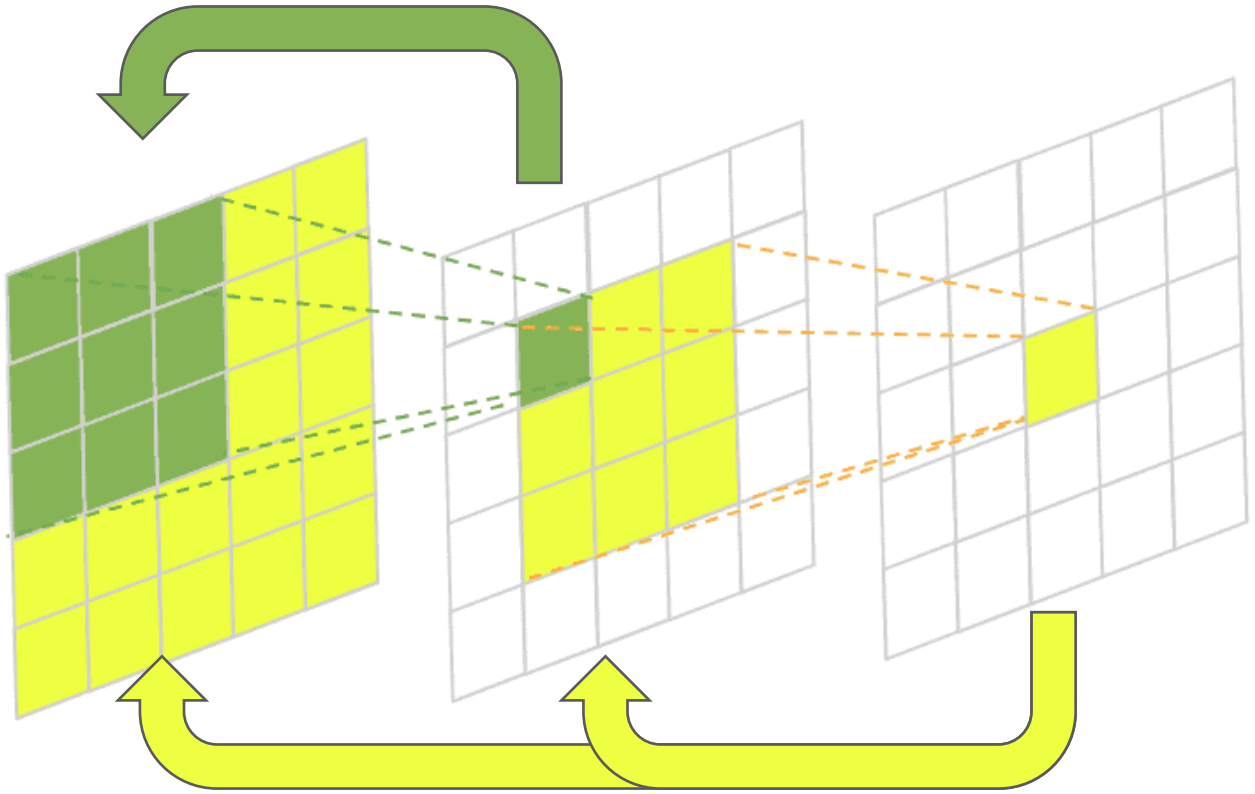
Summary: Receptive Field



Summary: Receptive Field

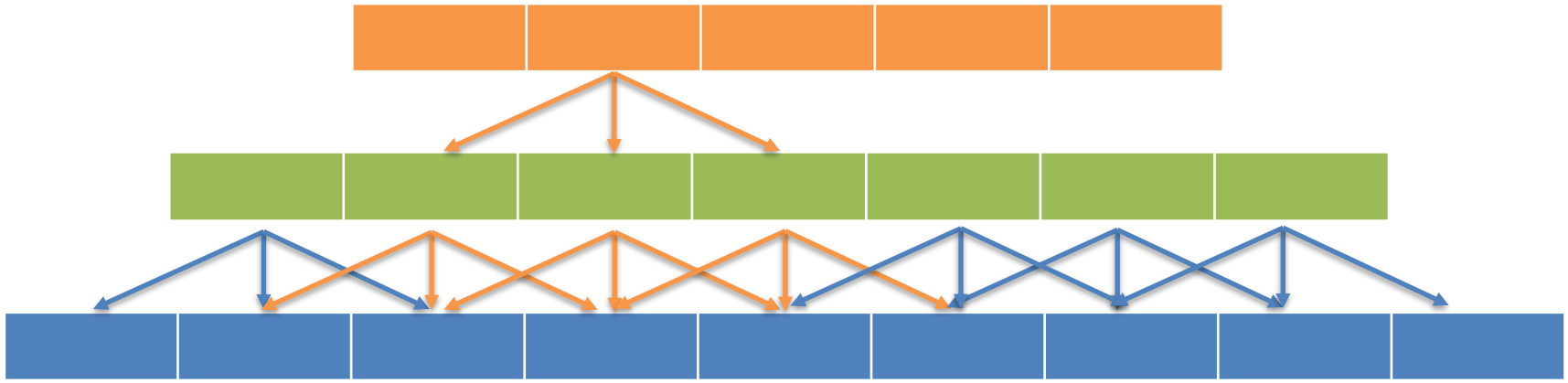


Summary: Receptive Field



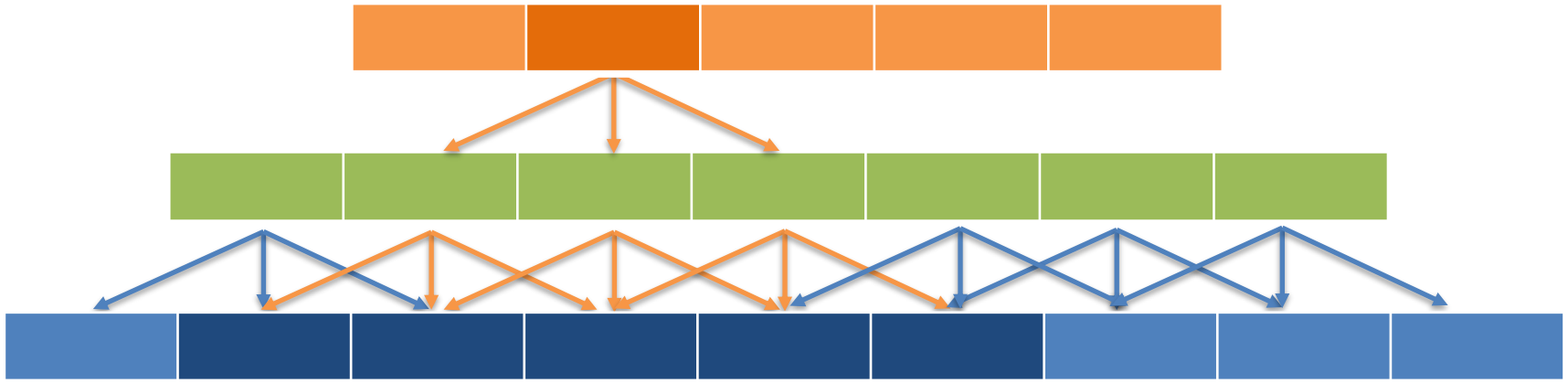
Layer's Receptive Field

Receptive field in 1D: no padding, stride 1 and kernel 3x1



Layer's Receptive Field

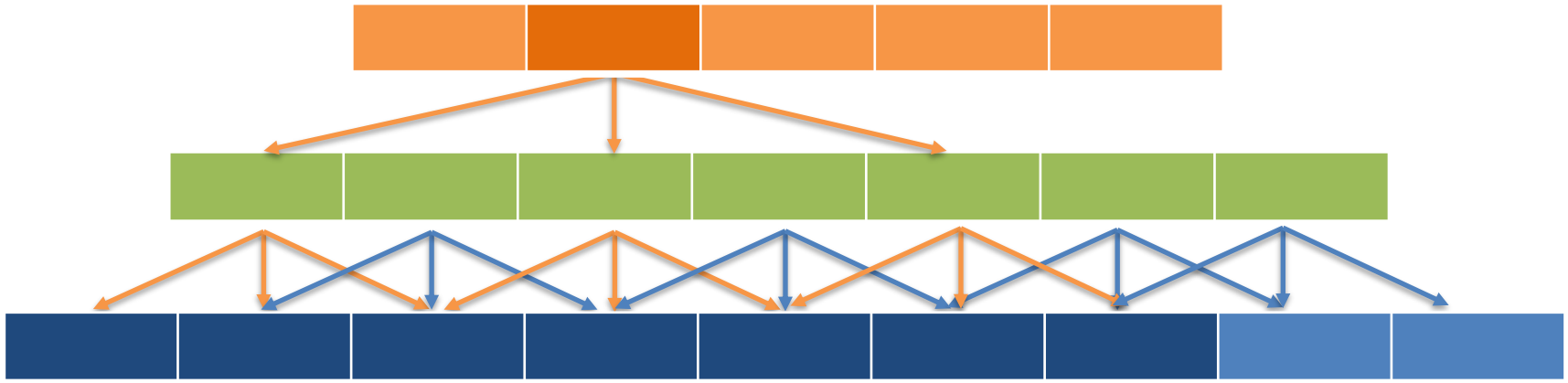
Receptive field in 1D: no padding, stride 1 and kernel 3x1



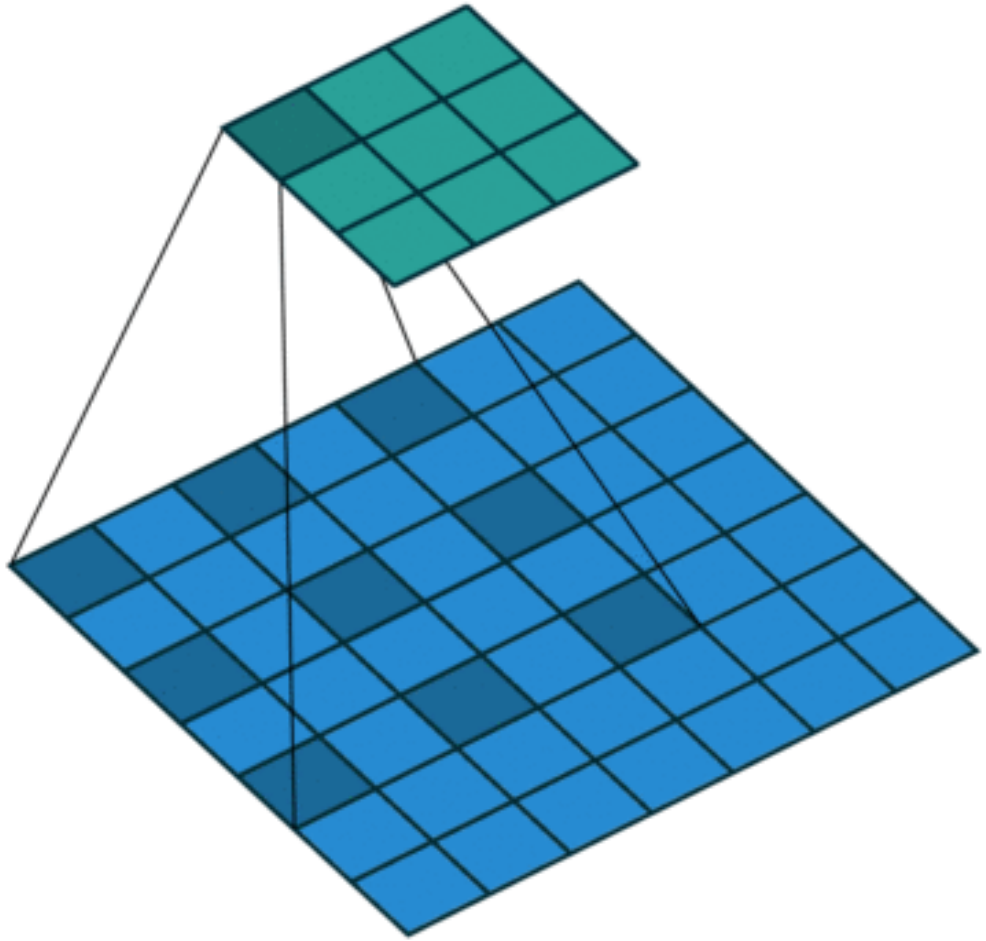
Dilated CNN

Receptive field in 1D: no padding, stride 1 and kernel 3x1

And skip some of the connections:



Dilated CNN



Understanding the receptive field of deep convolutional networks | AI Summer (theaisummer.com)

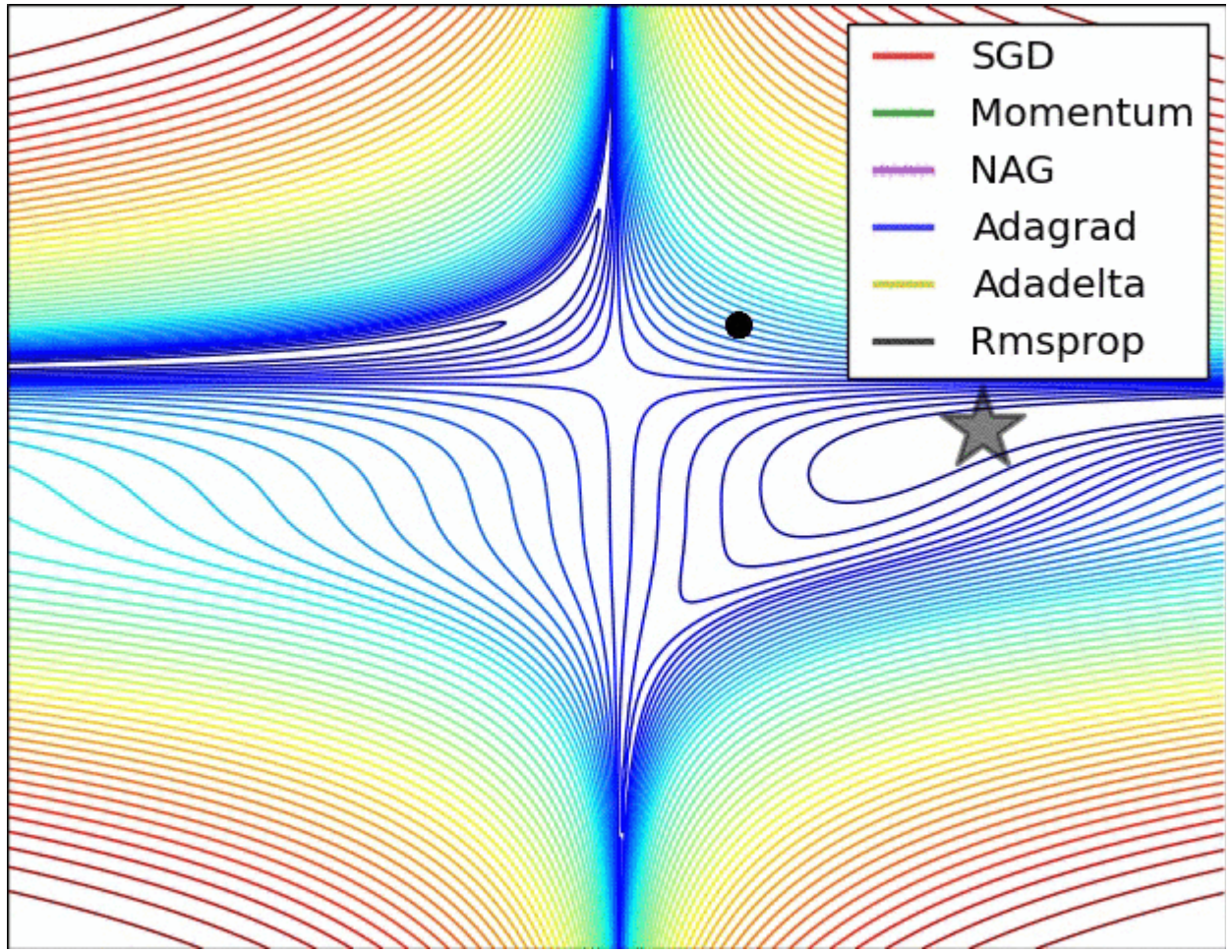
Loss / Cost calculation

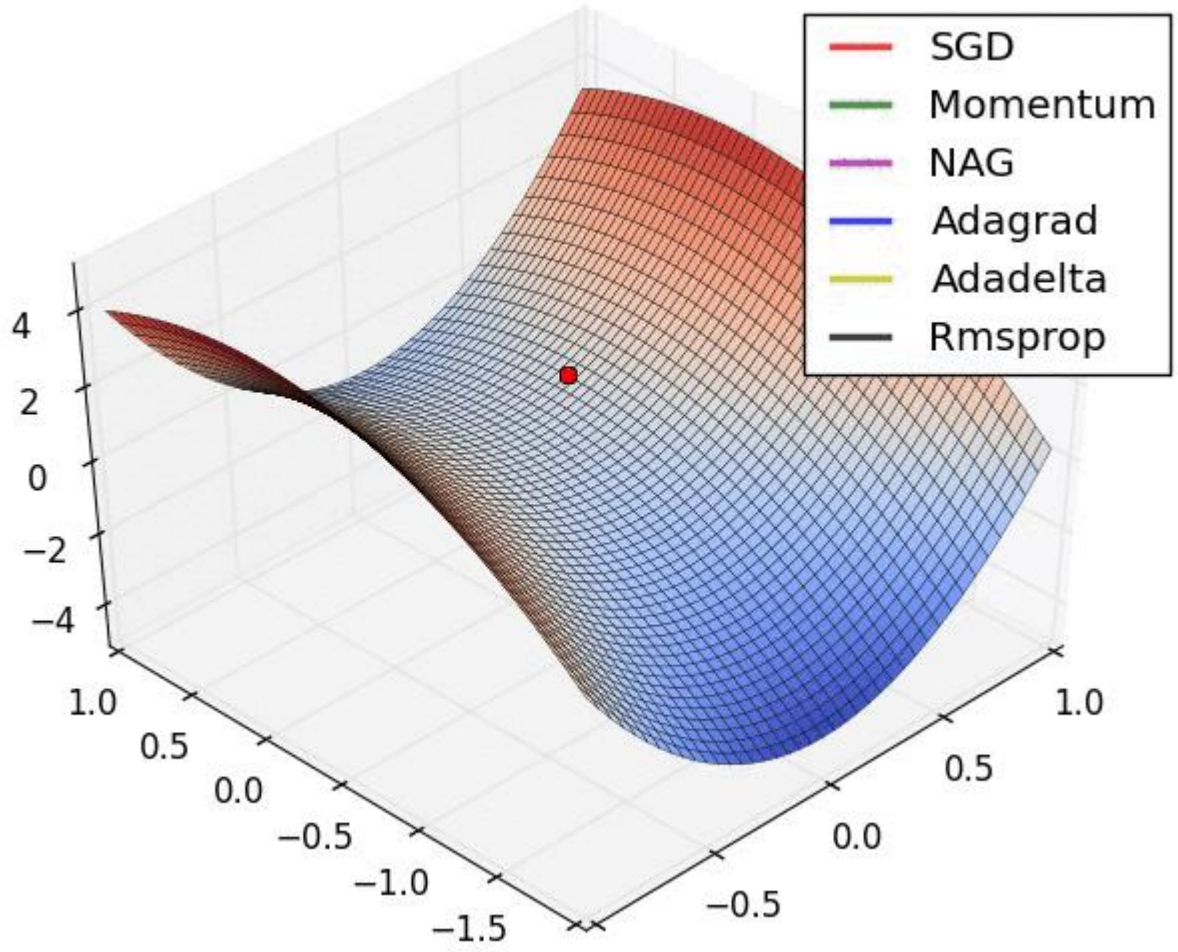
L1 Distance

- $d_1(I_1, I_2) = \sum_P \left| I_1^P - I_2^P \right|$ Where the sum is taken over all pixels:

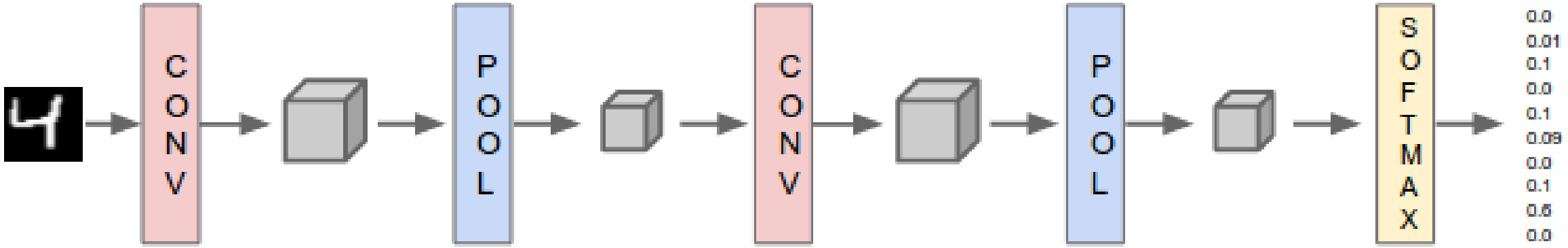
test image					training image					pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	=	46	12	14	1	→ 456
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

Optimization

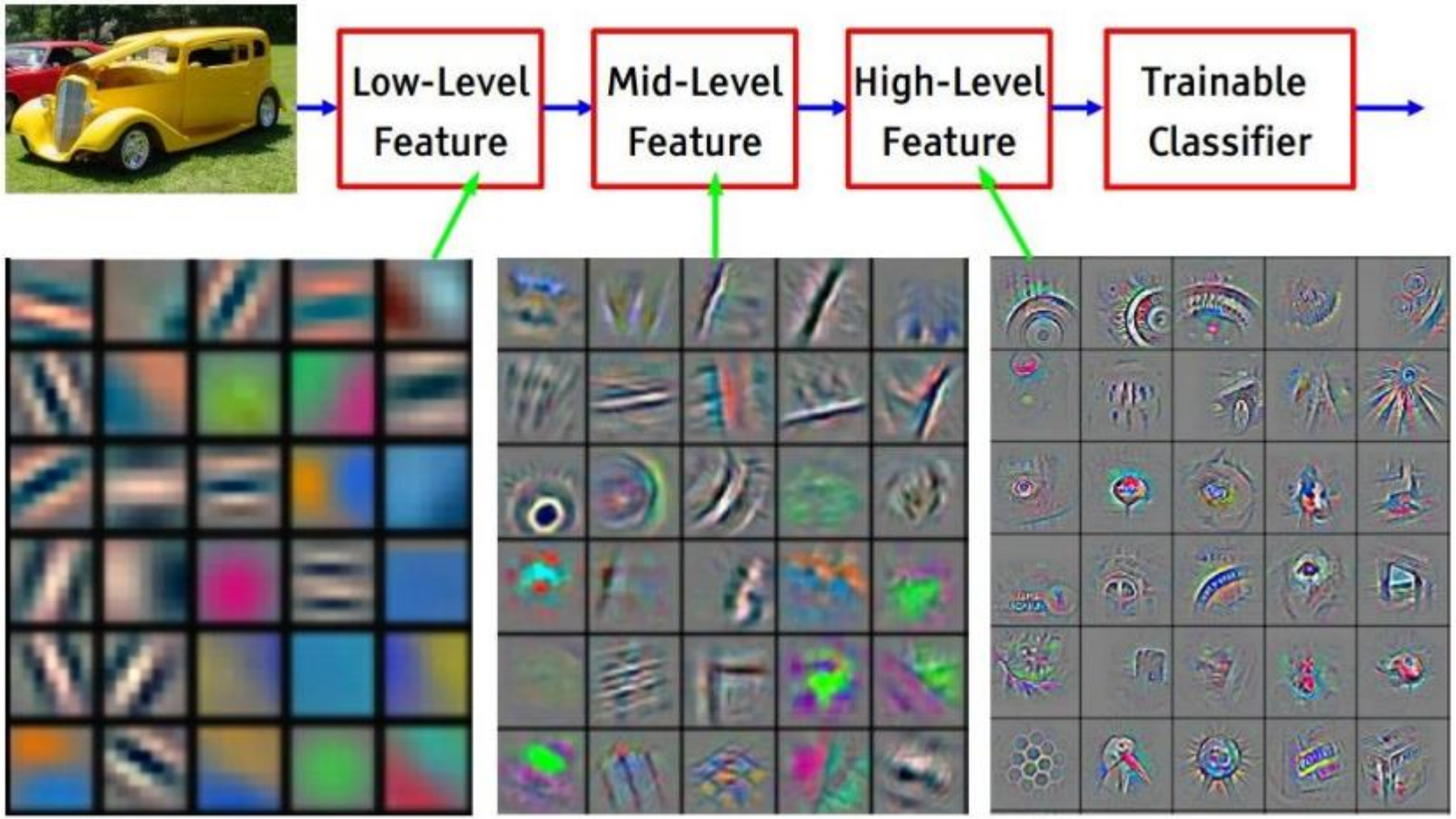




Putting it all together

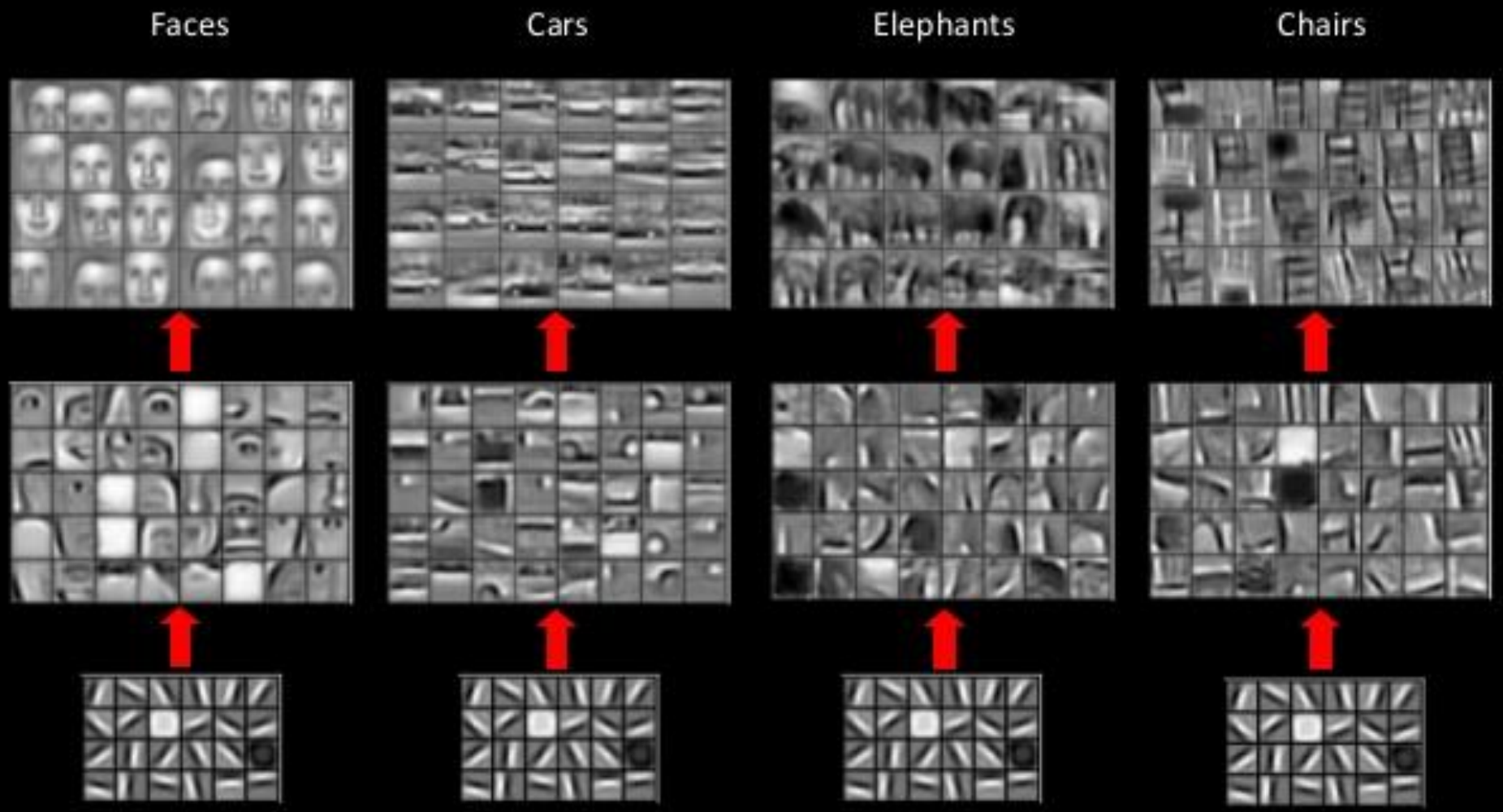


Neural Networks Extract Features



Learning of object parts

Examples of learned object parts from object categories



Demos

- https://playground.tensorflow.org/
- ConvNetJS: Deep Learning in your browser (stanford.edu)

imc FH KREMS
UNIVERSITY OF APPLIED SCIENCES

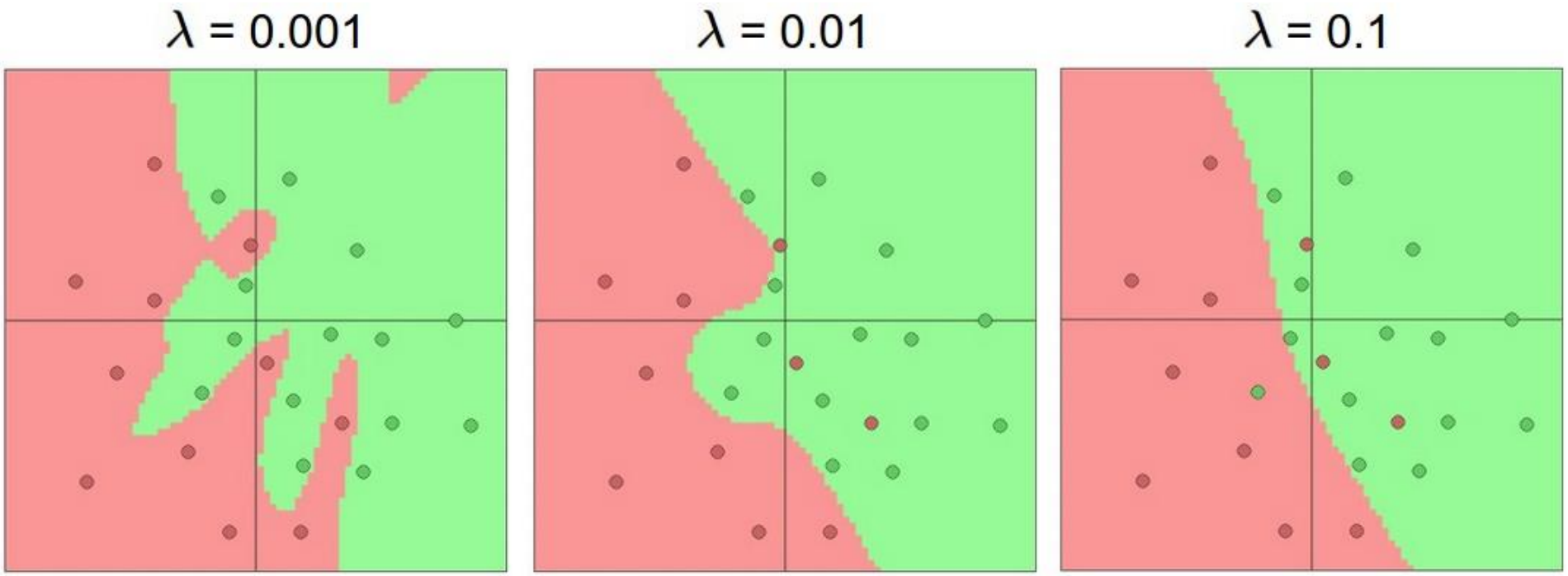
REGULARIZATION TECHNIQUES

To prevent overfitting

Overfitting

(Recap)

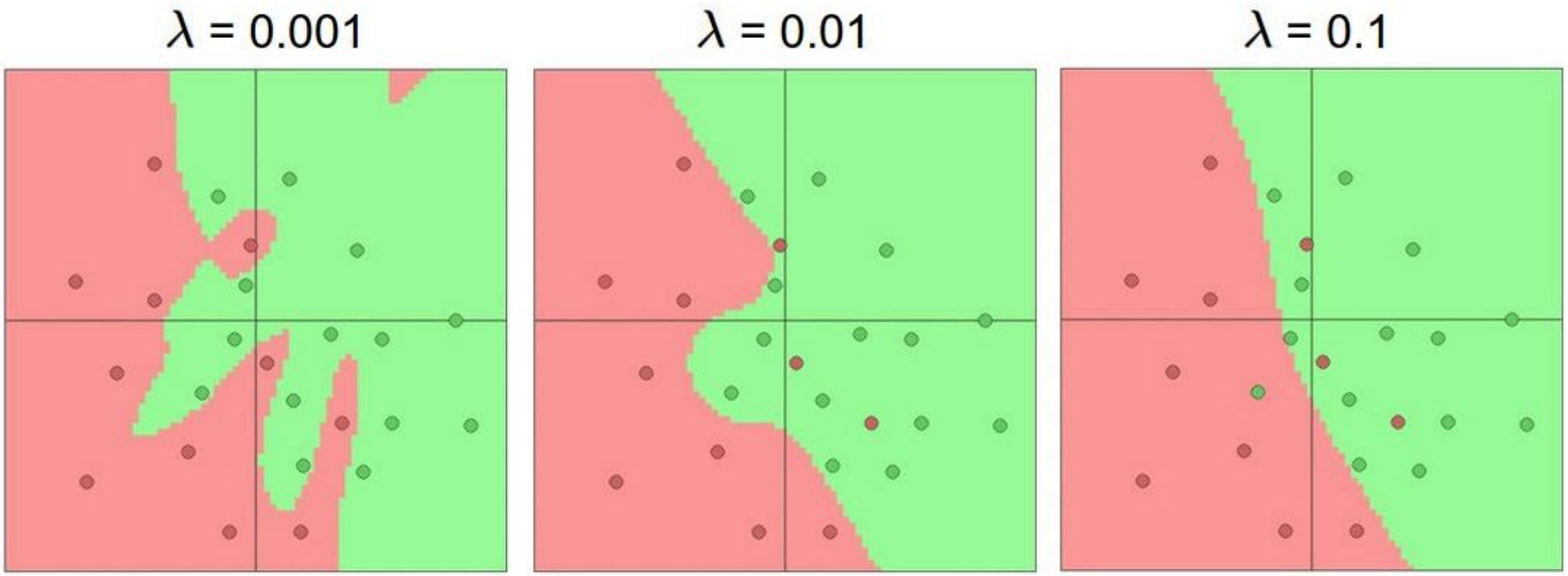
$$L(w) = \sum_{i=1}^{n} \left(y^i - wx^i\right)^2$$



Overfitting

(Recap)

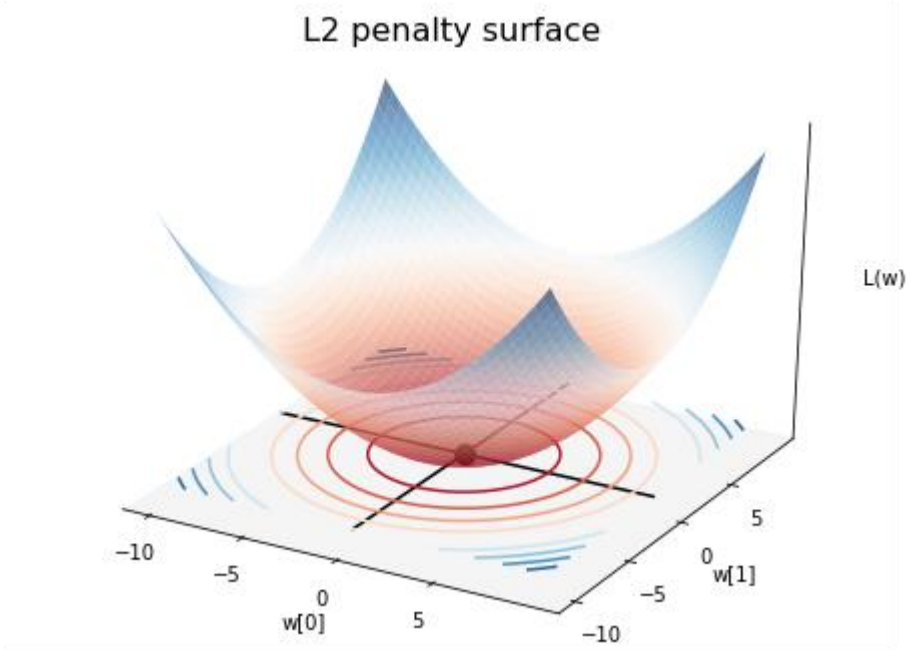
$$L(w) = \sum_{i=1}^{n} \left(y^i - wx^i\right)^2 + \lambda \sum_{j=0}^{d} w_j^2$$

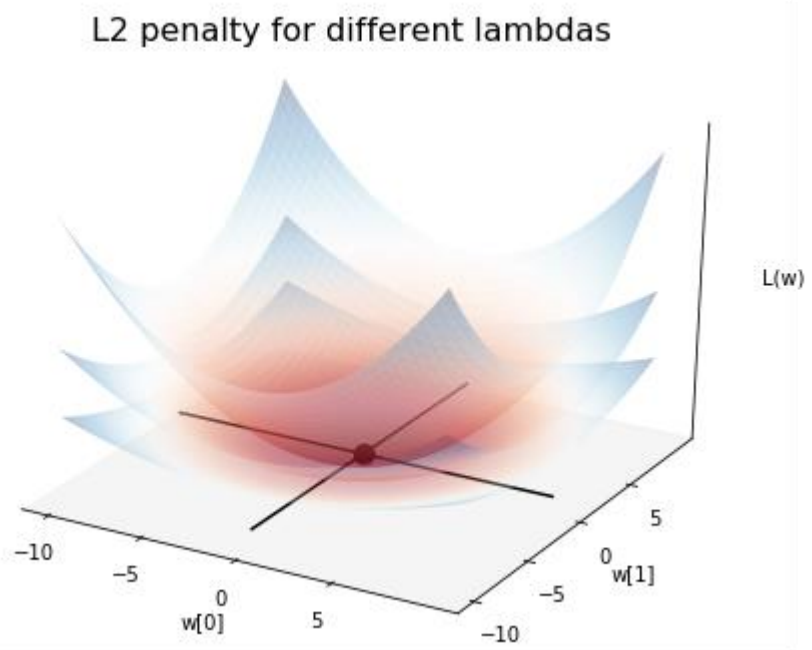


Overfitting

(Recap)

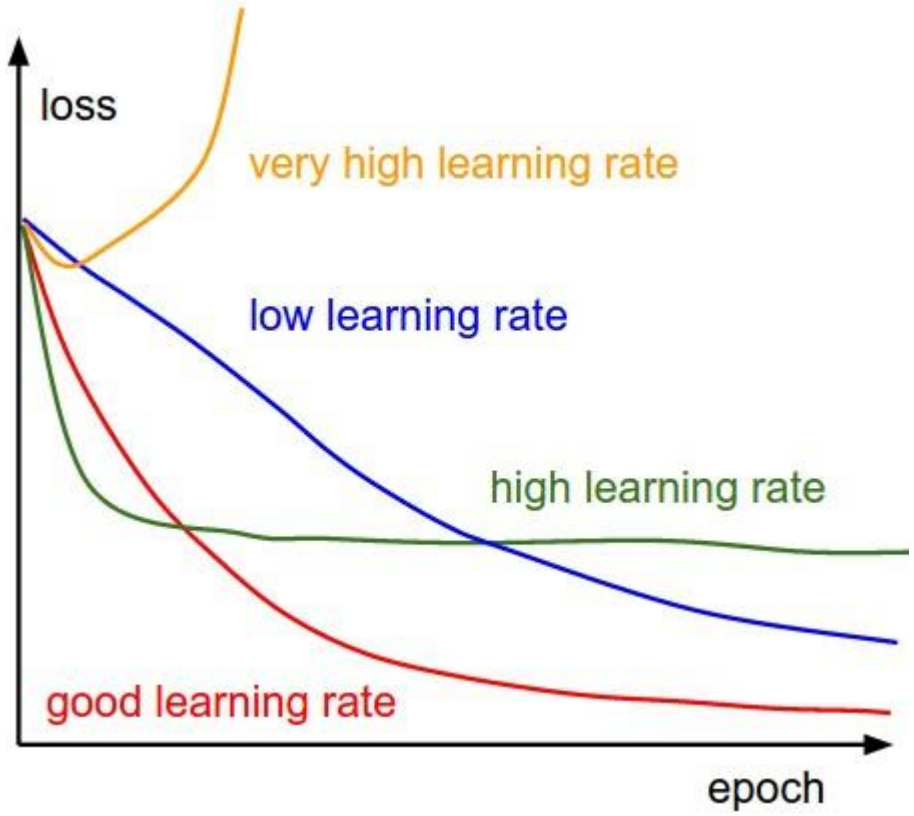
$$L(w) = \sum_{i=1}^{n} \left(y^i - wx^i\right)^2 \qquad +\lambda \sum_{j=0}^{d} w_j^2$$

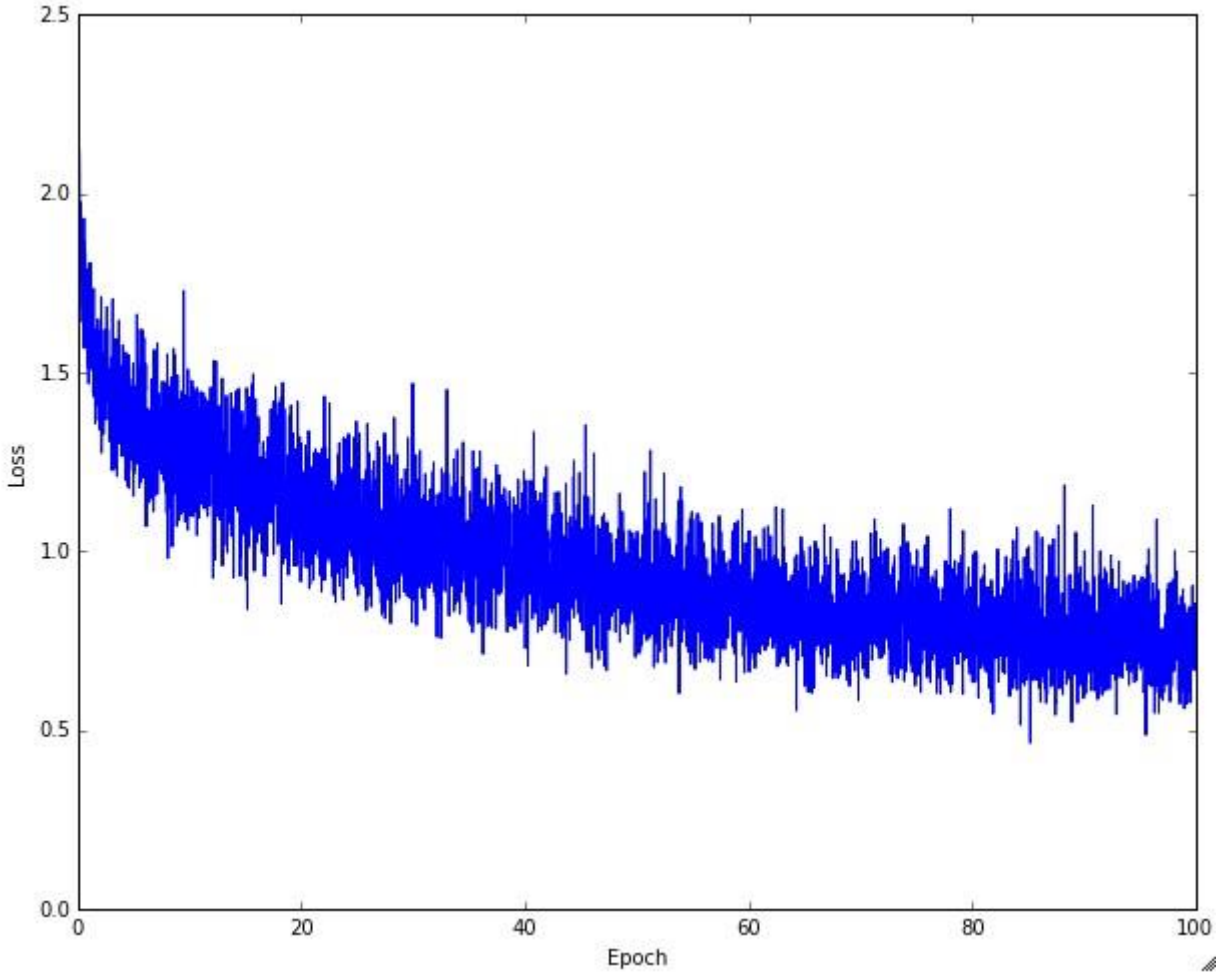




Linear Regression with Regularization (aunnnn.github.io)

Learning Rate Setting





Regularization

How to prevent overfitting?

- Choosing a proper Learning Rate
- Drop-out (elapsing neuron weights)
- Normalization (Batch Normalization, Gradient Accumulation)

imc FH KREMS
UNIVERSITY OF APPLIED SCIENCES

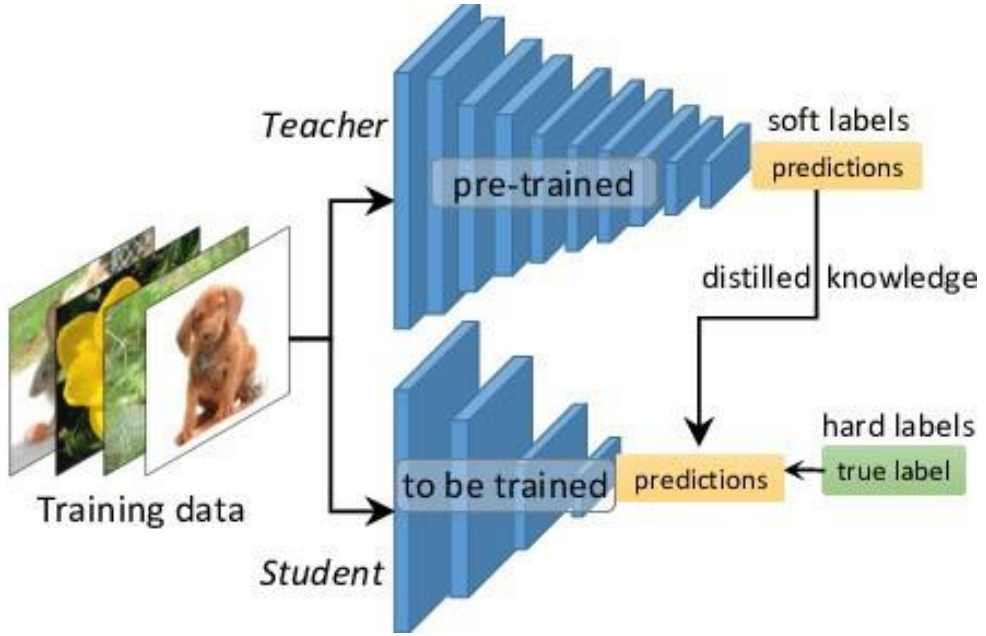
MODEL DISTILLATION

Teacher-Student

Model Distillation

Basic Idea (Example with Multi-Class Classification Network)

- Train a model with many parameters:
  - Multiple Layers
  - Multiple Neurons in each Layer
- Create a smaller Network
  - With significantly **fewer** parameters
- Train it on the outputs of the large network
  - On the float numbers, not the rounded softmax/true labels



Model Distillation

Result Network

- The new network learns with the outputs of the original, NOT with a cost function of 1s and 0s
- The network generalizes better
- Annotated Code: Distilling the Knowledge in a Neural Network (labml.ai)

imc FH KREMS
UNIVERSITY OF APPLIED SCIENCES

CNN – DEEPER DIVE

CS231n Convolutional Neural Networks for Visual Recognition

imc FH KREMS
UNIVERSITY OF APPLIED SCIENCES

MODEL INTERPRETATION

Why should we trust a model?!

Do you trust your doctor?

Imaginary Scenario

- You feel unwell
- Your doctor diagnoses you and gives you a prognosis
  - Based only on your historical medical data
  - Without meeting you in person

- Would you seek a second opinion?

- When and why would you trust your doctor?

Model Explainability / Prediction Reasoning

Tools to explain computer vision predictions

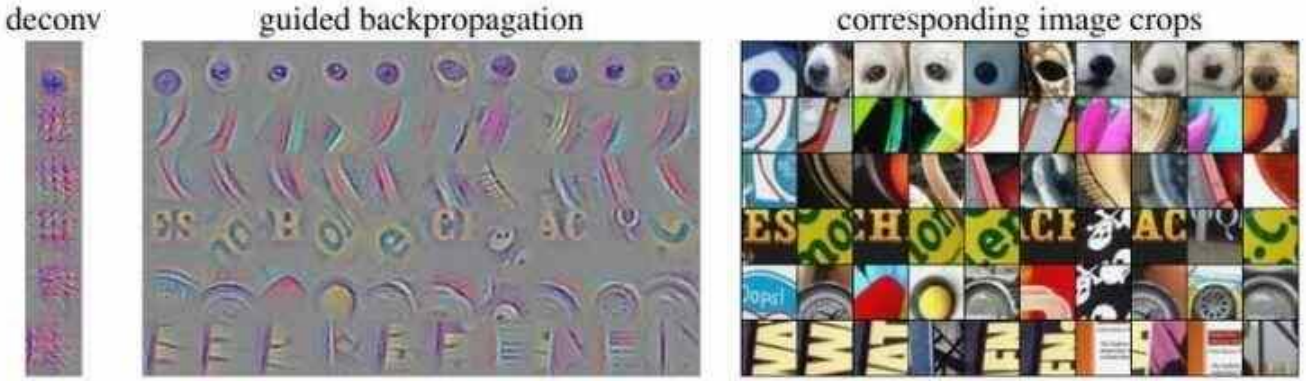
- Saliency Maps
  - *Deconvolutional Network Approach* (DeconvNet)
  - *Gradient-Based Approach* (Vanilla Gradient)
  - *Guided Backpropagation Algorithm*
  - *Class Activation Mapping (CAM)*
  - *Grad-CAM*
- *Explanatory Tools*
  - *SHAP*
  - *LIME*
- *Causal Inference*

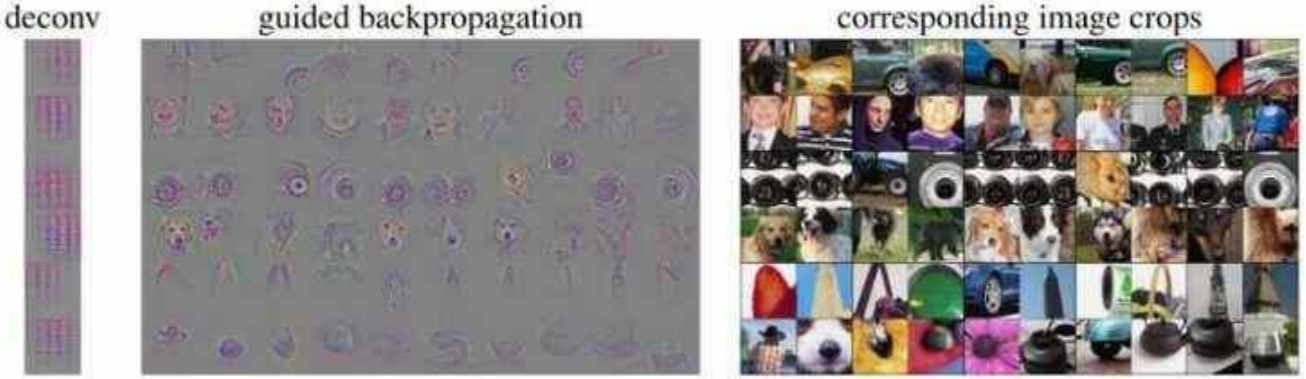
Saliency Maps

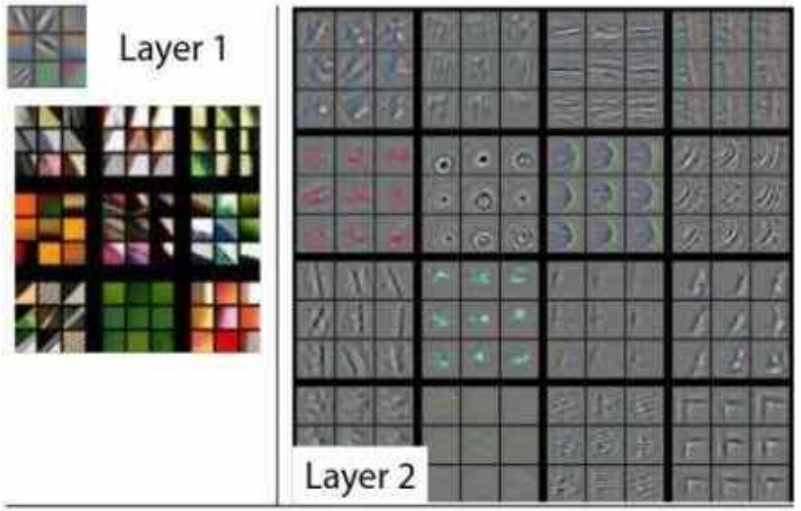
What did CNN focus on?

- Given an image, visualize the excited CNN areas
- Can be applied on different layers
- Normally visualized as a heatmap layer on top of the original image

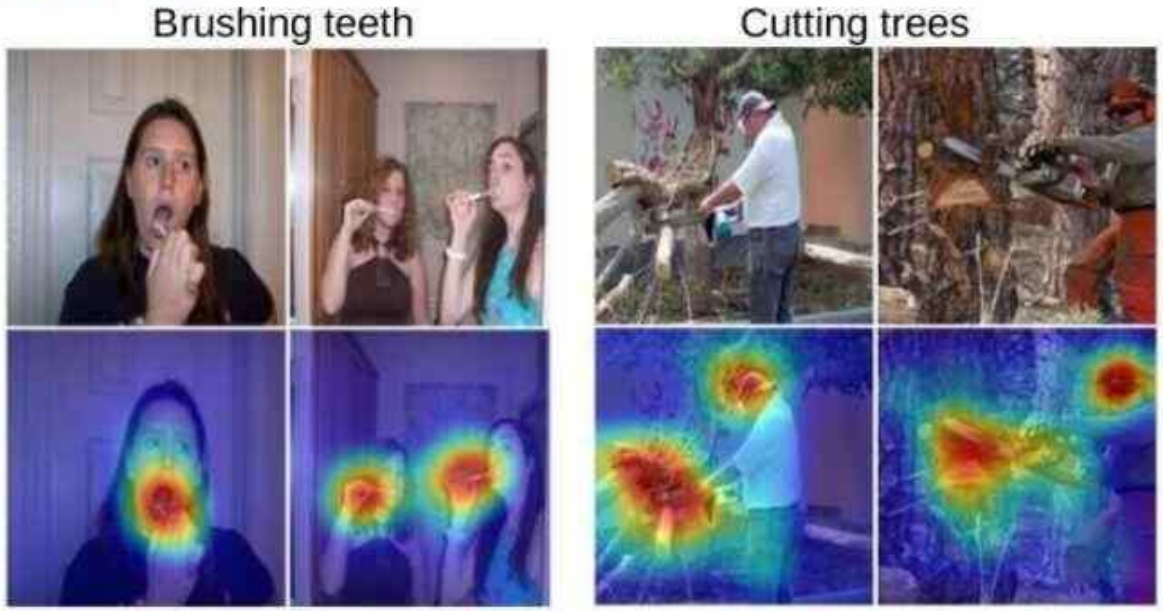
- Tools:
  - iNNvestigate neural networks' predictions!
  - DeepExplain








(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat'

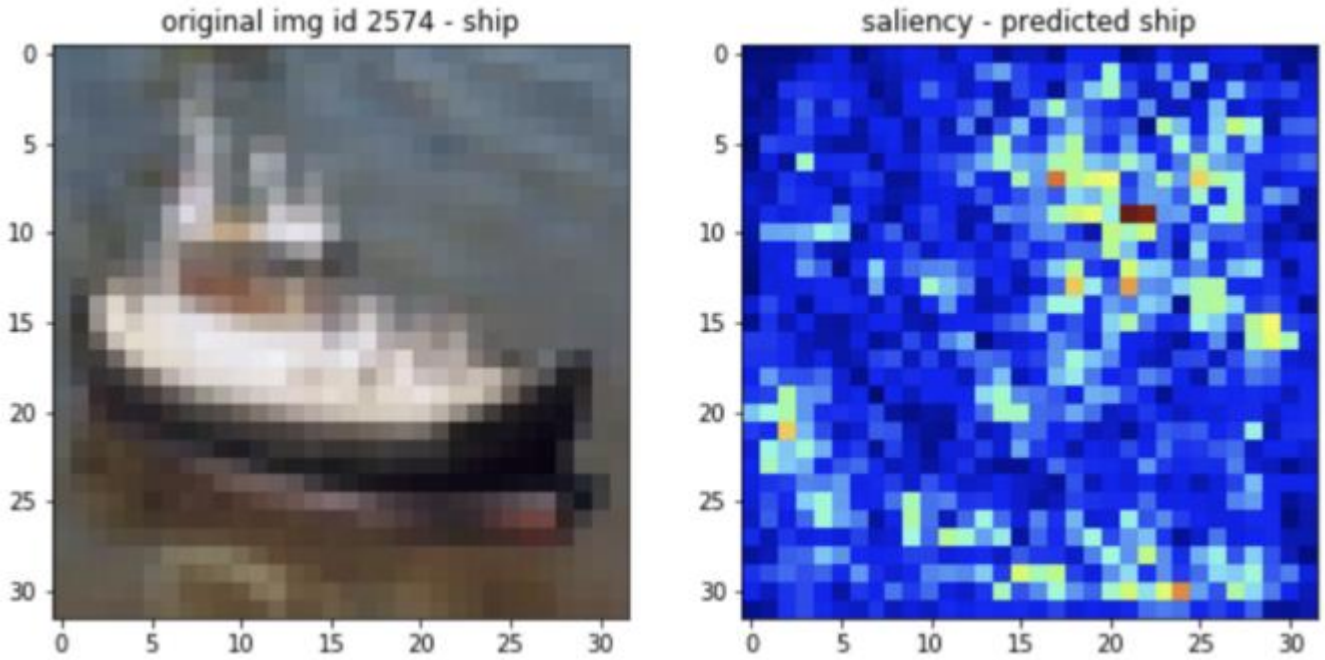


Saliency Maps

Example

The water around the ship is the reason for the prediction.

The model won’t do well if the ship is in a different background.

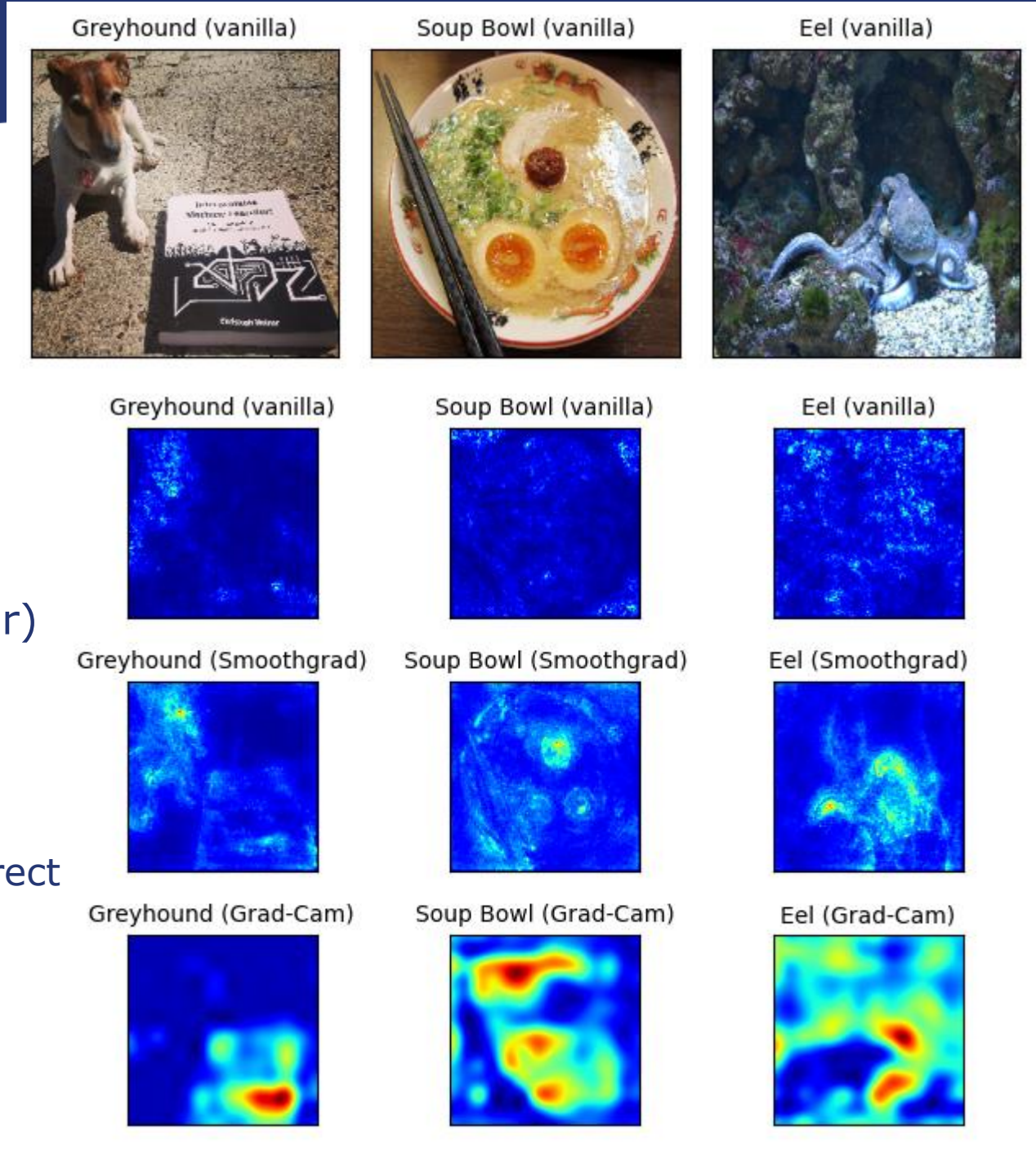


Saliency Maps

Comparison

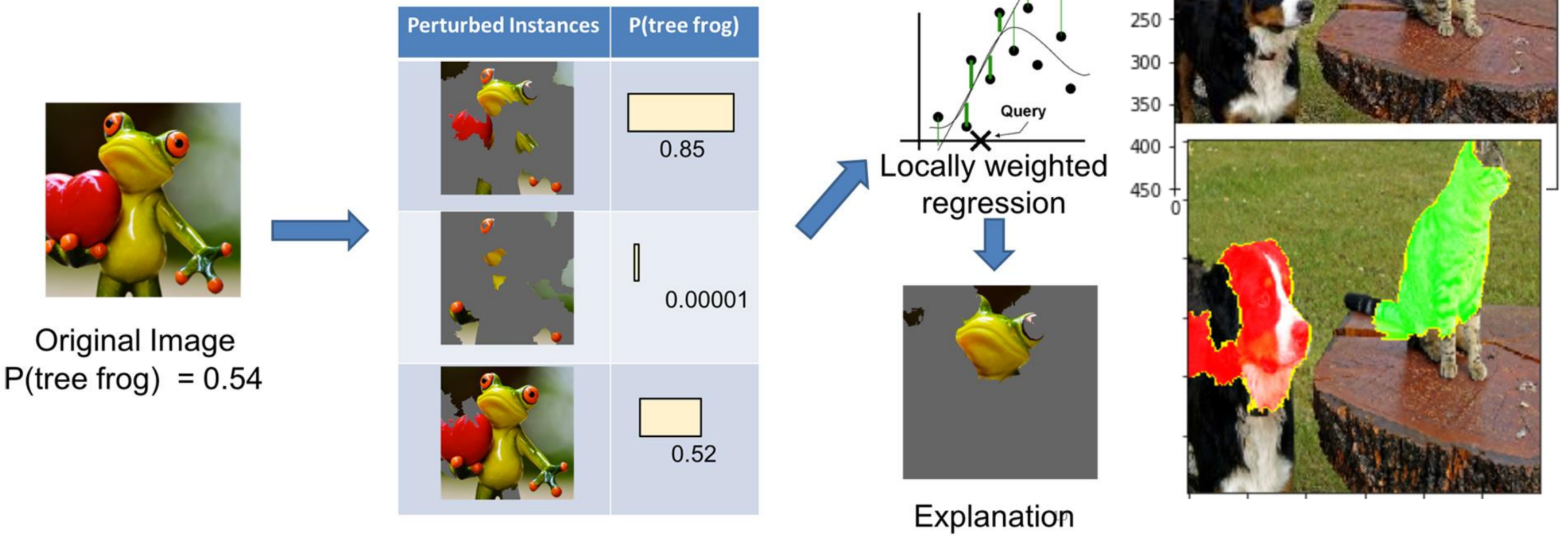
- Relatively effective
- Fast to compute (GradCAM - only last layer)
- Gives good results

- But…
  - Hard to interpret if an interpretation is correct
  - Fragile / Sensitive
  - May be unreliable



Explanatory Tools

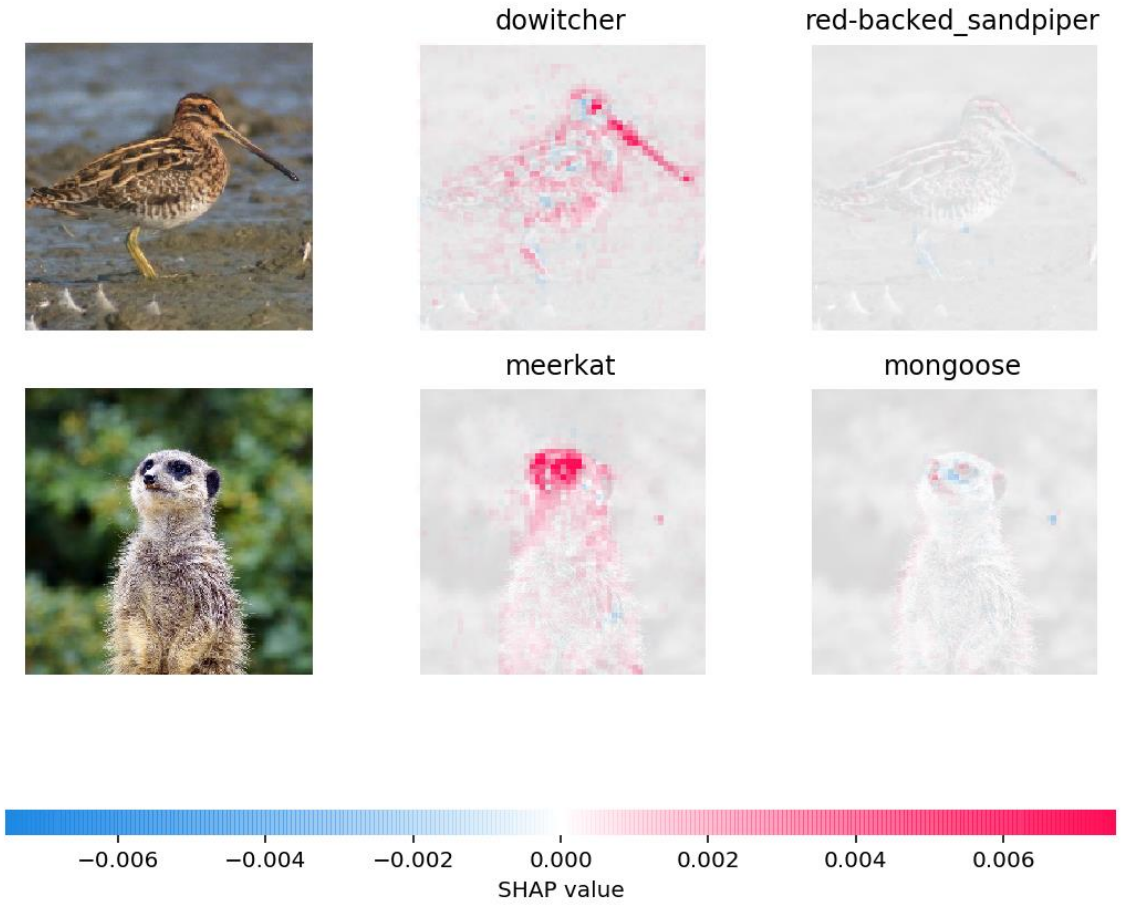
LIME - Local Surrogate



Explanatory Tools

SHAP

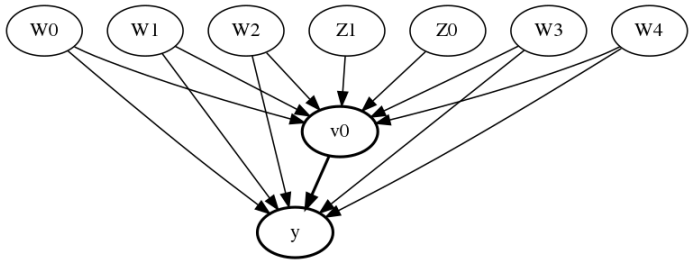
- Shapley values
  - Game Theory
  - How much each player (feature) in a coalition contributed to the result (prediction)
- SHAP
  - Combines LIME, Shapley Vals and others.
  - “Coalition” of super-pixels

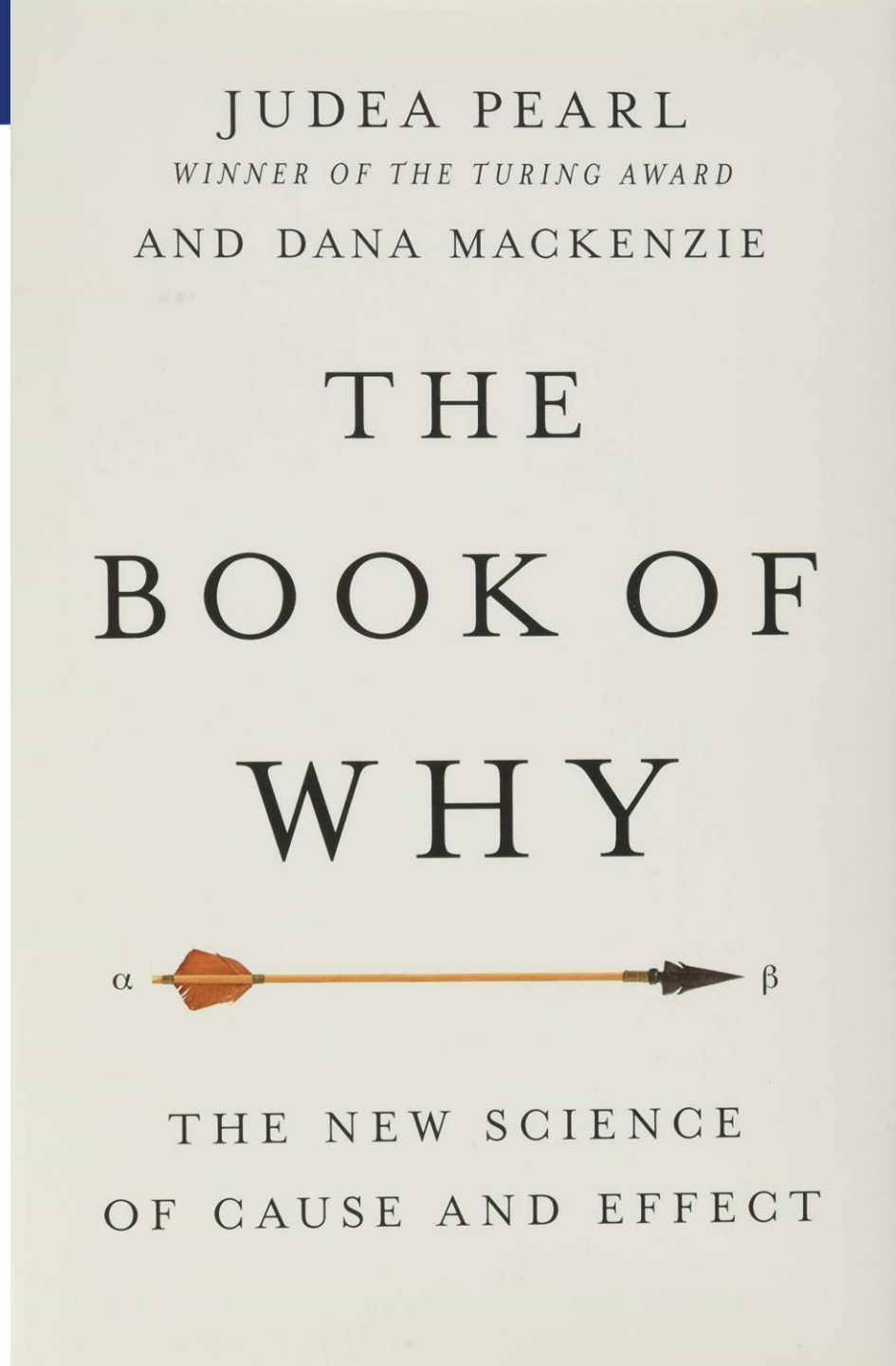


Causal Inference

80% of the population are X… but Why?

- Statistics examine samples and deduct for whole populations
  - But lacks the (hidden) reasons why.
- A new science
  - New math:
    P(y|x) →P(y|do(x))
- Applications:
  - Personalized medicine





TRANSFER LEARNING

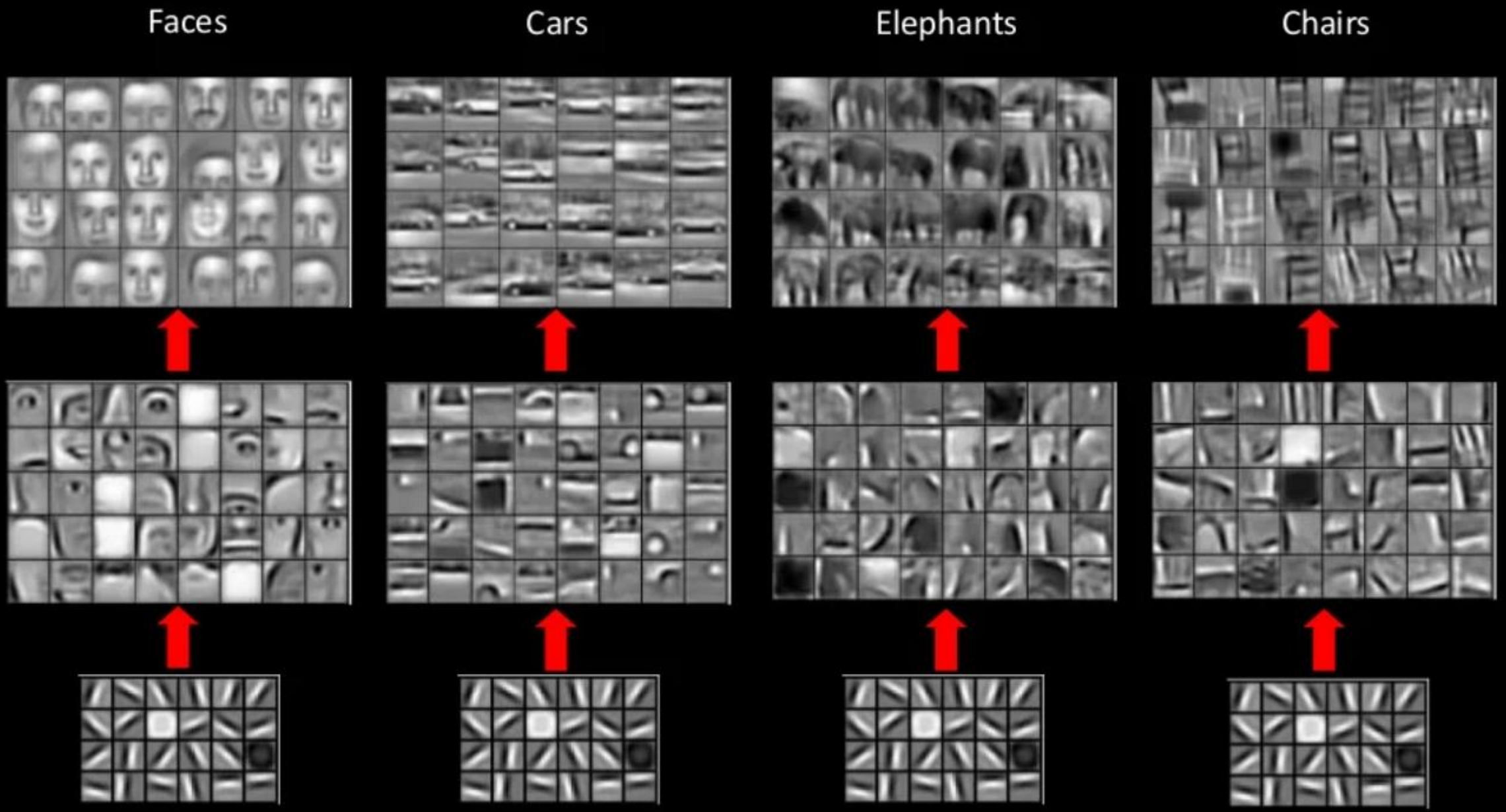
Transfer Learning

How is it done?

- Shared weights
  - Set a lower LR for layers
- Self-Supervised Learning
  - Training on a large amount of data

- Then, fine-tune on a down-stream task

Examples of learned object parts from object categories

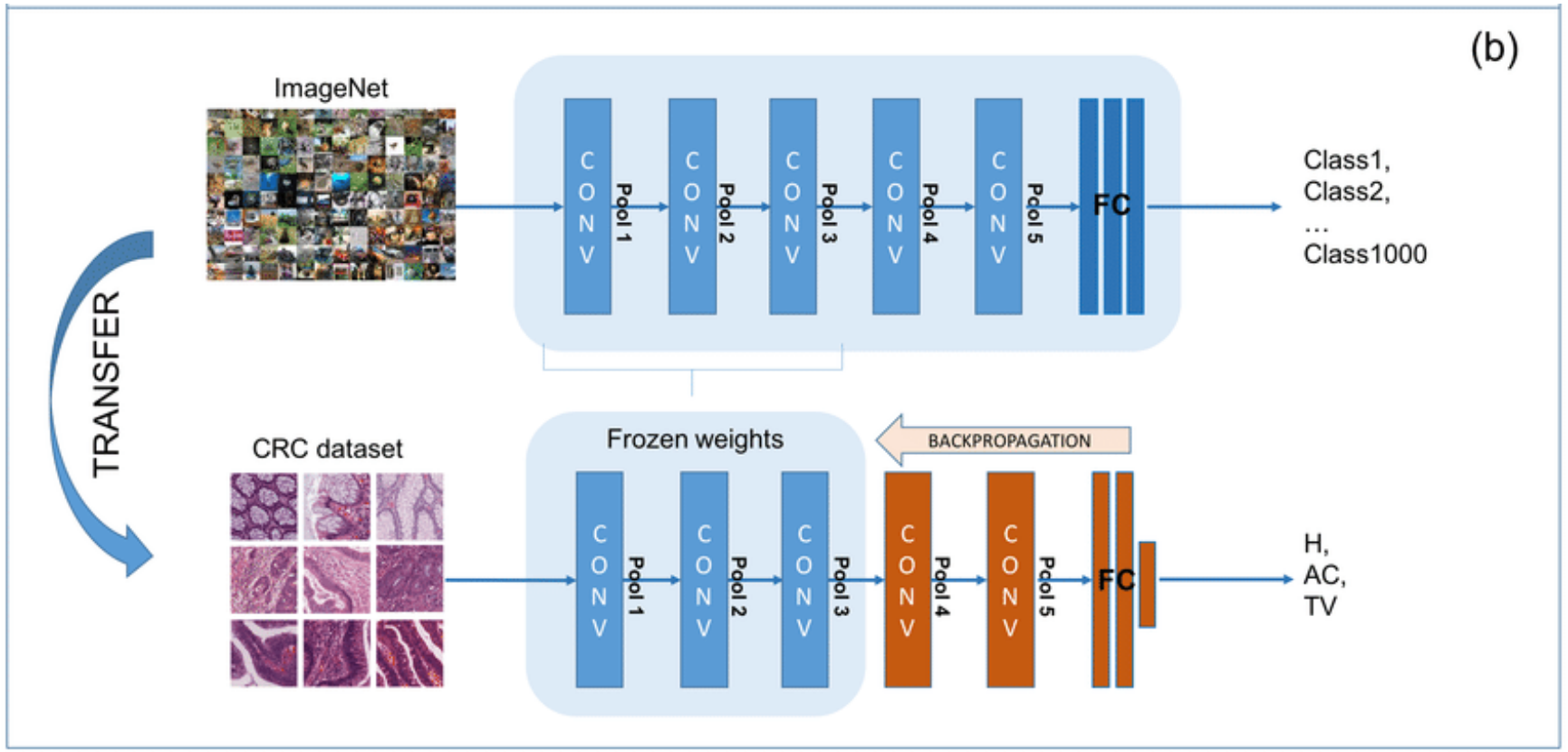


Transfer Learning

Re-using layer-weights

Setting a different LR for each layer

[More info]



Transfer Learning

Training

- We often use already pre-trained networks
  - Training such models is expensive
  - Hard to find large amounts of data

- Better from the same domain
  - E.g., train on all X-Rays
  - Fine-tune on chest X-ray for specific use-case (e.g., knee)

Transfer Learning

PyTorch example

- https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

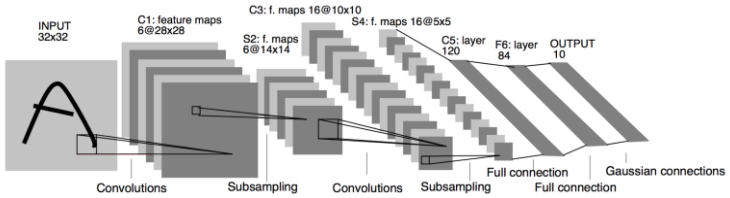
- Learning rate:
  - Can be set to give more weight to recent layers – close to the SoftMax

CNNs in Mammography (cont'd)

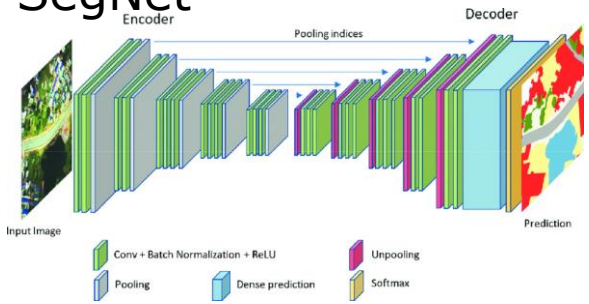
- **Best practices** to improve CNN performance:
  - Pre-processing images (especially mammograms)
  - Combination of learned and hand-crafted features
  - Data augmentation
  - Transfer learning
  - Batch normalization
  - Dropout
- Attention-based modules
- Multi-view images
- Balanced data distributions
- Context and patient information
- Multi-stage **vs** end-to-end (E2E) methods

CNN Architectures

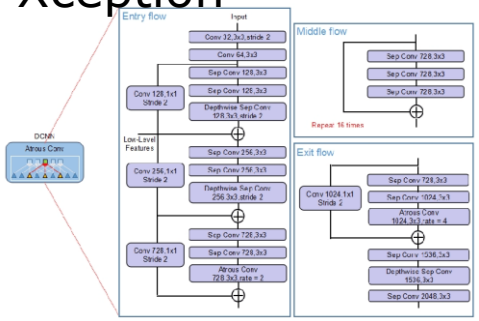
LeNet



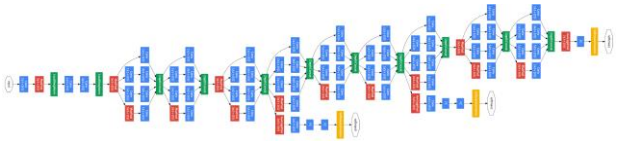
SegNet



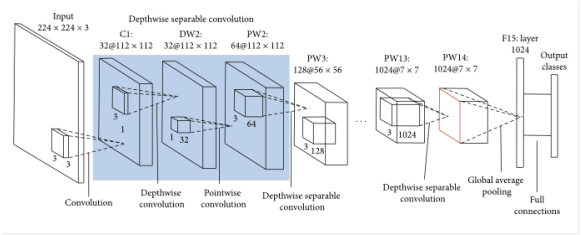
Xception



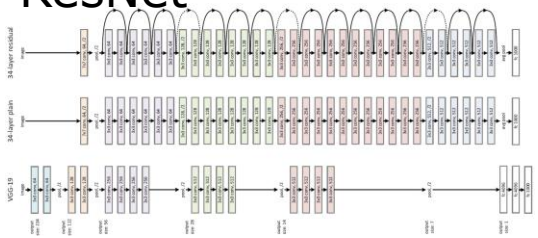
GoogLeNet



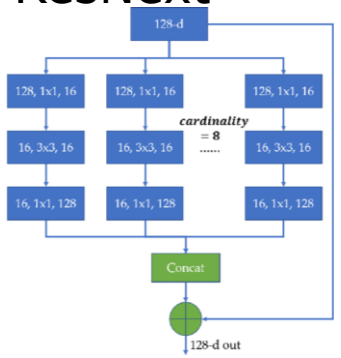
MobileNet



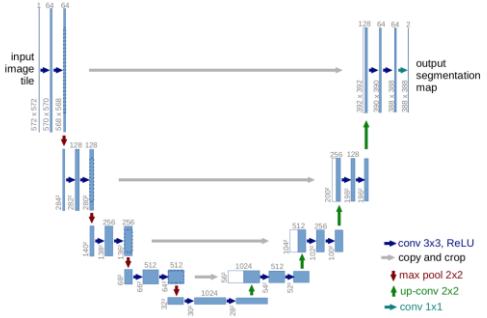
ResNet



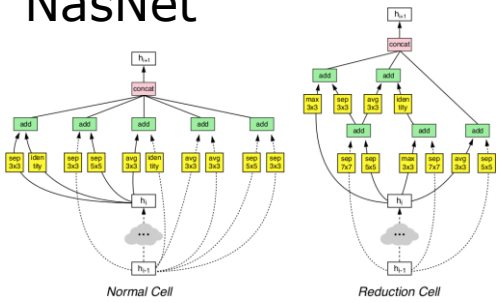
ResNext



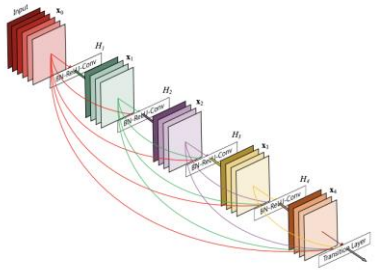
U-Net



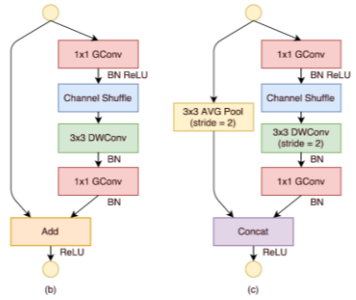
NasNet



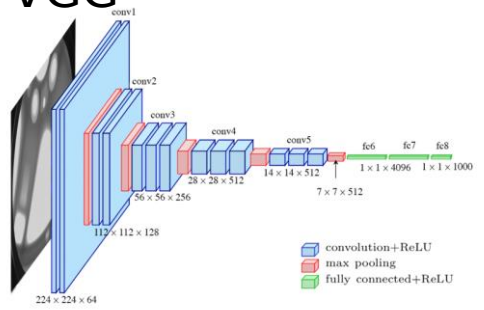
DenseNet



ShuffleNet



VGG



CNN Architectures

PyTorch

- TIMM: timm · PyPI
- Now also in HuggingFace:
  GitHub - huggingface/pytorch-image-models: PyTorch image models, scripts, pretrained weights -- ResNet, ResNeXT, EfficientNet, NFNet, Vision Transformer (ViT), MobileNet-V3/V2, RegNet, DPN, CSPNet, Swin Transformer, MaxViT, CoAtNet, ConvNeXt, and more