Computer Vision

Class 4

Deep Learning

Artificial neural networks

- Framework for learning a nonlinear predictive function (classification or regression)

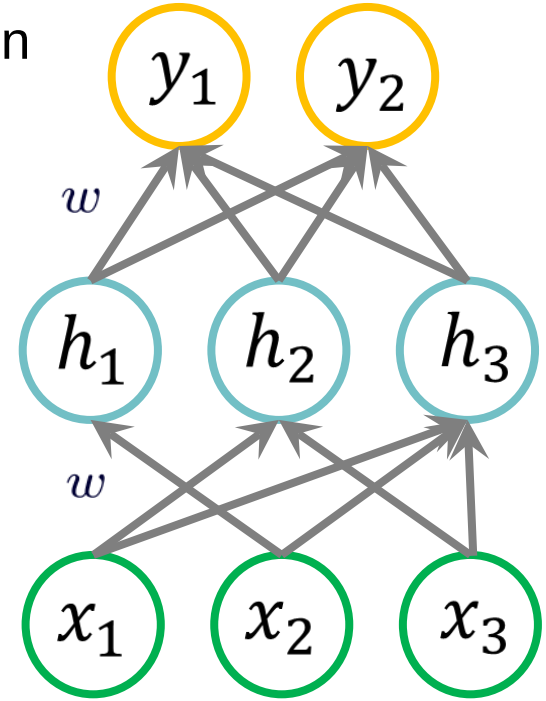
- Input: a set of data-target pairs $D = \{(\mathbf{x}_i, t_i)\}$
  Sought: weights $\mathbf{w}$

- Learning is an optimization problem, e.g., **squared loss** for regression:

$$L(\mathbf{w}; D) = \sum_i \|y(\mathbf{w}, \mathbf{x}_i) - t_i\|^2 \rightarrow \min$$

- Optimization with a variant of gradient descent (**back-propagation**)

- Multiple hidden layers allow for a hierarchical representation that gets more and more abstract towards the top → **deep learning**

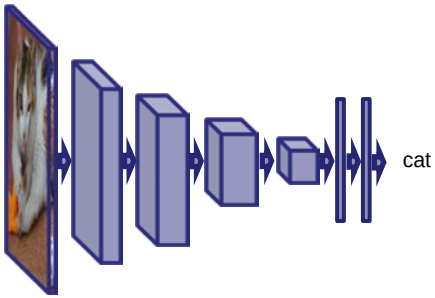


x - input data
h - hidden layer(s)
y - output layer

History of neural networks

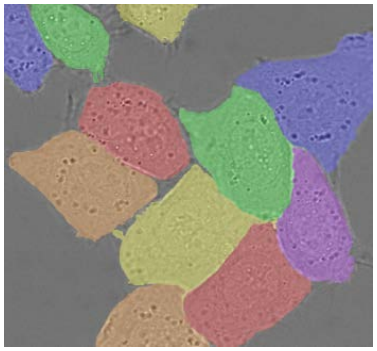
- 1958: Rosenblatt’s Perceptron
- 1974: First version of backpropagation (Werbos)
- 1980: Fukushima’s Neocogniton
- 1986: Backpropagation (Rumelhart et al.)
- 1989: Convolutional Network (LeCun)
- 1995: Support Vector Machines with soft-margin and kernel-trick
- 2007: Unsupervised pre-training (Hinton)
- 2012: AlexNet (Krishevsky et al.)
- 2020: Scaling it up (GPT3)

Most computer vision tasks can be formulated as mapping problems



- Image classification

- Object detection



- Semantic segmentation

- Pose estimation

- Optical flow estimation

- Image super-resolution

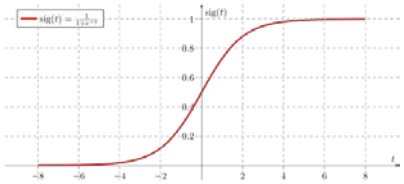
- Other new applications, such as image generation or visual question answering

General concept of a feed-forward network

- The connections between successive layers $k-1$ and k are modeled by a learnable linear function

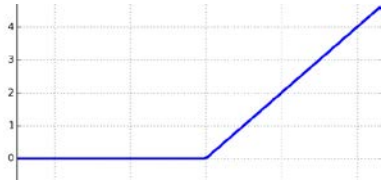
$$\mathbf{h}^k = \sigma\left(\mathbf{W}^{k\top}\mathbf{h}^{k-1} + \mathbf{w}_0^k\right)$$

followed by a fixed nonlinear function $\sigma(\cdot)$



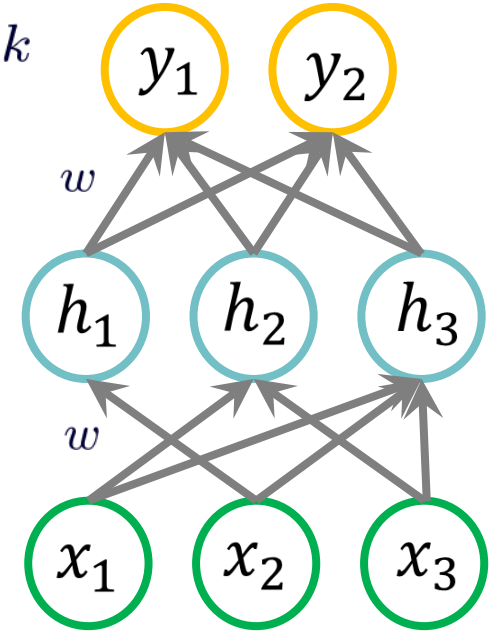
sigmoid

$$\sigma(s) = \frac{e^s}{1+e^s}$$



ReLU

$$\sigma(s) = \max(0, s)$$



x - input data
h - hidden layer(s)
y - output layer

- Nonlinearity needed to model nonlinear functions
  - Intuition: threshold on whether a unit is activated or not
  - Consider: we must compute the derivative of this nonlinearity and the resulting gradient should be informative

- All information of the network about the predictive function is in the learned weights

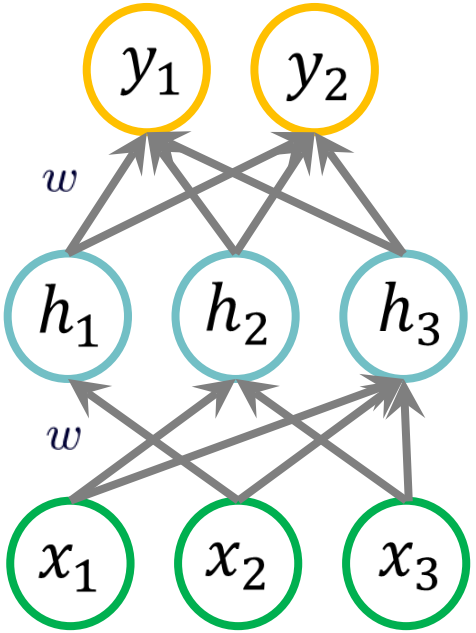
Forward propagation

- A forward pass through the network computes the predictive output $y(\mathbf{w}, \mathbf{x})$ for an input $\mathbf{x}$ given the learned weights $\mathbf{w}$

- Mainly a matrix-vector multiplication for each layer
→ fast on parallel hardware (GPUs)

- Even large networks yield real-time outputs on large GPUs

- Specialized hardware for these forward passes is getting available on the market (cloud computing, embedded hardware)



x - input data
h - hidden layer(s)
y - output layer

Training the network with backpropagation

- Goal: adapt the weights such that the prediction error on the training set is minimized

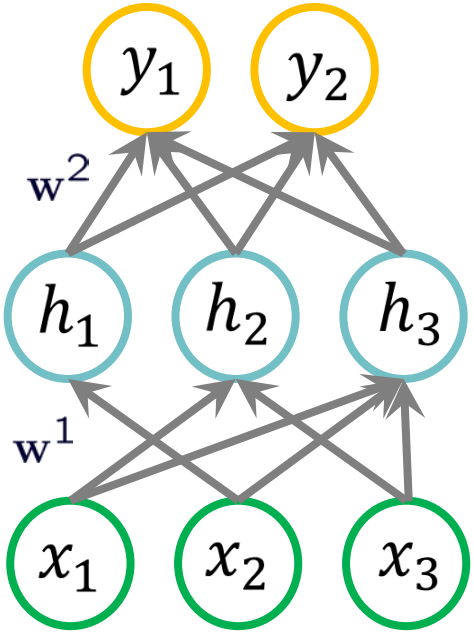
$$\mathcal{L}(\mathbf{w}; D) = \sum_i \|\mathbf{y}(\mathbf{w}, \mathbf{x}_i) - \mathbf{t}_i\|^2 \qquad \mathbf{w}^* = \text{argmin}_{\mathbf{w}} \mathcal{L}$$

- For gradient descent we need the gradient

$$\nabla \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

- For a particular weight in a certain layer, we obtain the derivative via the chain rule; example:

$$\frac{\partial \mathcal{L}}{\partial w_1^1} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial w_1^1}$$



x - input data
h - hidden layer(s)
y - output layer

- Previous computations can be reused when back-propagating the error to weights deeper in the network

- Gradient computation requires one forward and backward pass for each sample

Stochastic gradient descent

- Datasets for deep learning consist of thousands or millions of samples

  → Computing the gradient on the whole dataset is slow

- Consequence: compute the gradient only on a small random subset of samples (**minibatch**)

- Each gradient step is computed on a different random subset
  → stochastic gradient descent

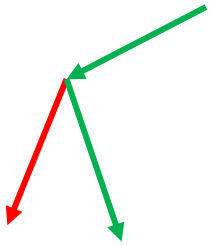
- Two relevant hyper-parameters:
  - Batch size
  - Step size (**learning rate**)

- Stochasticity due to small batches is important for regularization

Adding inertia to gradient descent

- Gradient descent tends to zigzagging
- Especially true for stochastic gradient descent due to changing minibatches
- Assuming that the direction in the previous step was also a good descent direction, we want to keep some of it in the current step
- Classical gradient descent:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \tau \nabla \mathcal{L}$$



- Gradient descent with **momentum**:

$$\mathbf{v}^{k+1} = \mu v^k - \tau \nabla \mathcal{L}$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mathbf{v}^{k+1}$$

- Momentum allows for smaller batches (why?)

Weight initialization

- Gradient descent requires an initialization:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \tau \nabla \mathcal{L}$$

- Non-convex problem → initialization influences the solution

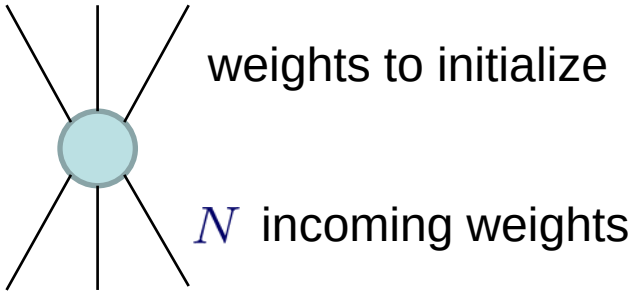
- Stochastic gradient descent avoids local minima to a large degree, but bad initialization can still lead to unsuccessful or slow training.

  → How should we initialize the weights?

- **Xavier initialization**: sample the weights from a Gaussian with zero mean and variance

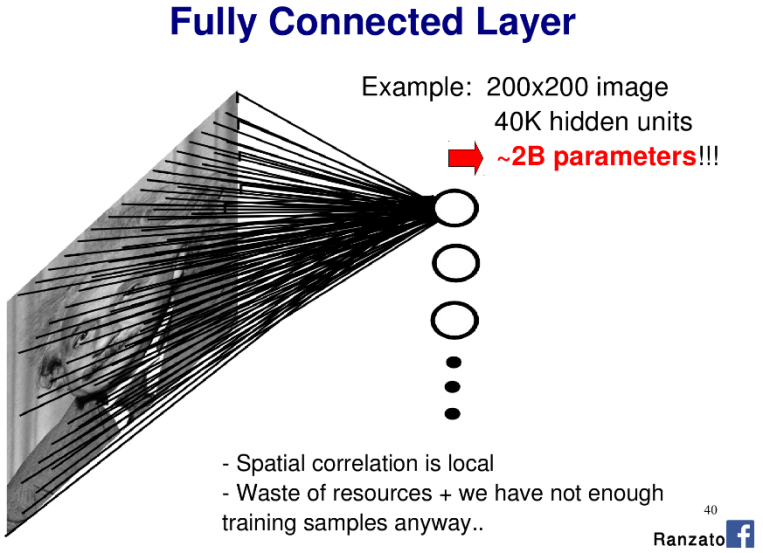
$$\sigma^2 = \frac{2}{N}$$

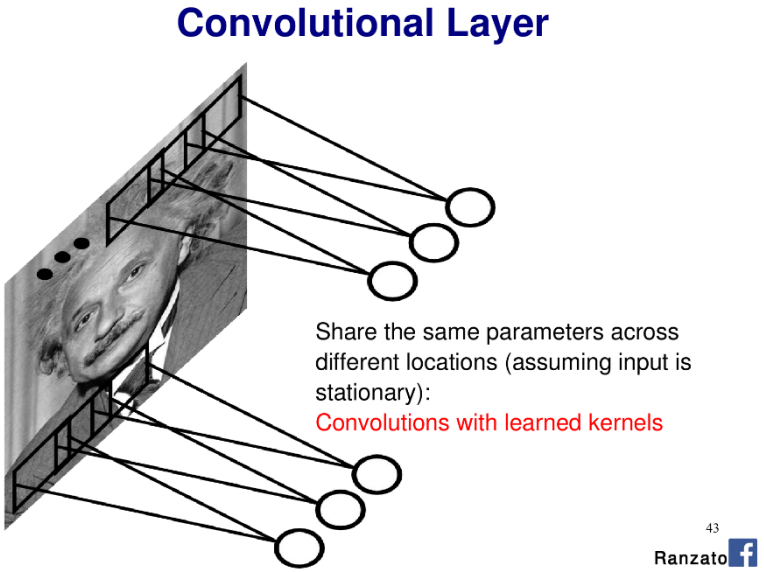
  (for ReLU activations)



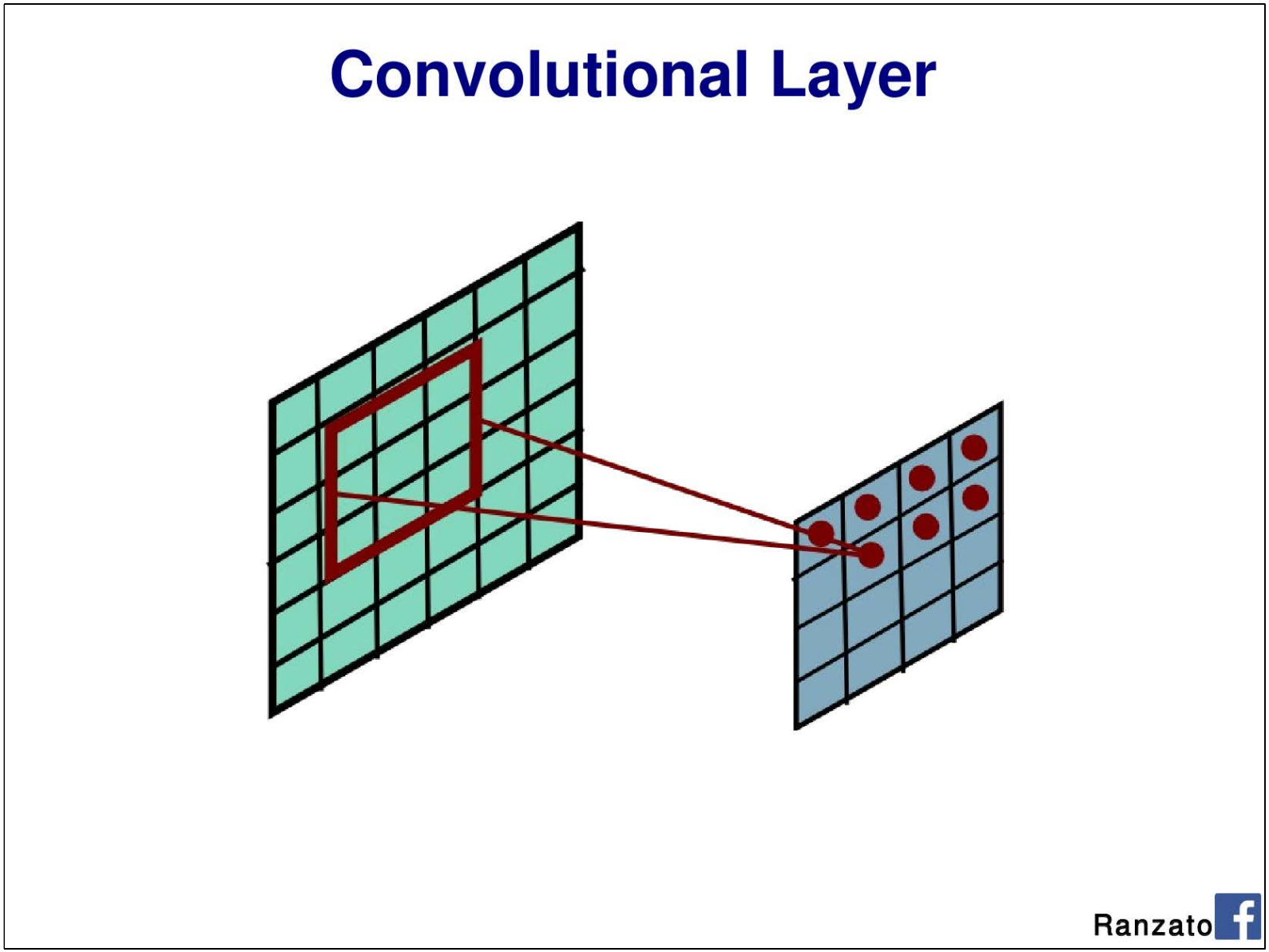
Networks for images: Convolutional networks

- Fully connected networks are inefficient on image inputs
  - Image content is largely local
  - It is mostly stationary (a certain structure can be anywhere in the image)
- Convolutional networks exploit both properties to save on weights
  - A unit is affected only by a local region (**receptive field**)
  - All regions use the same weights (**weight sharing**)
- Downsampling to consider larger receptive fields at higher abstraction levels

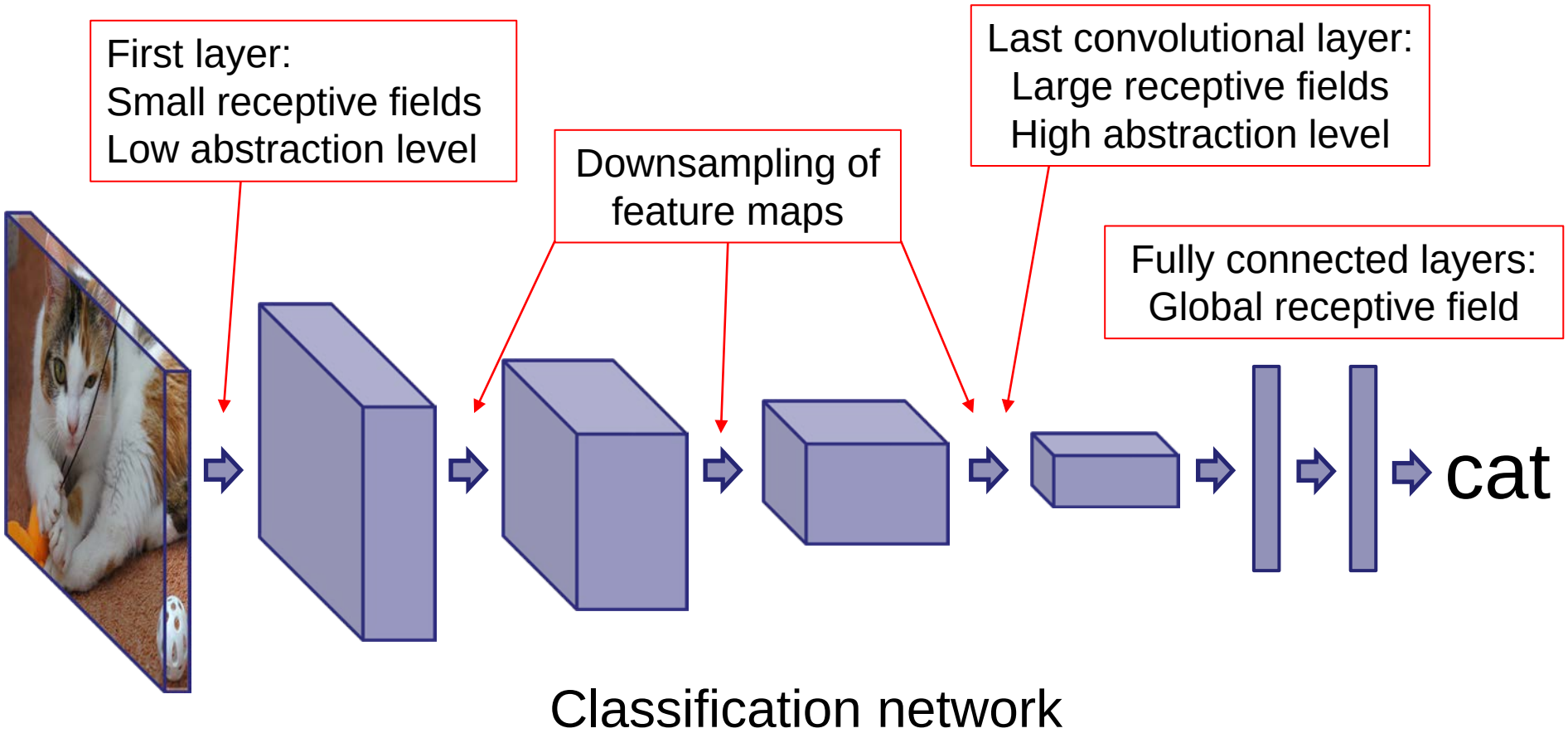




Forward propagation by convolution with a local filter

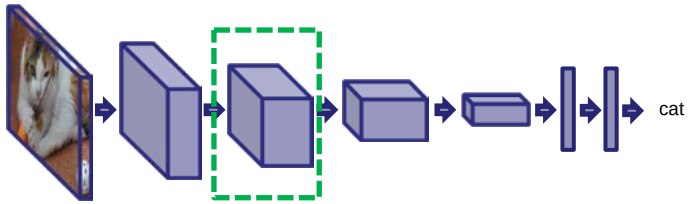


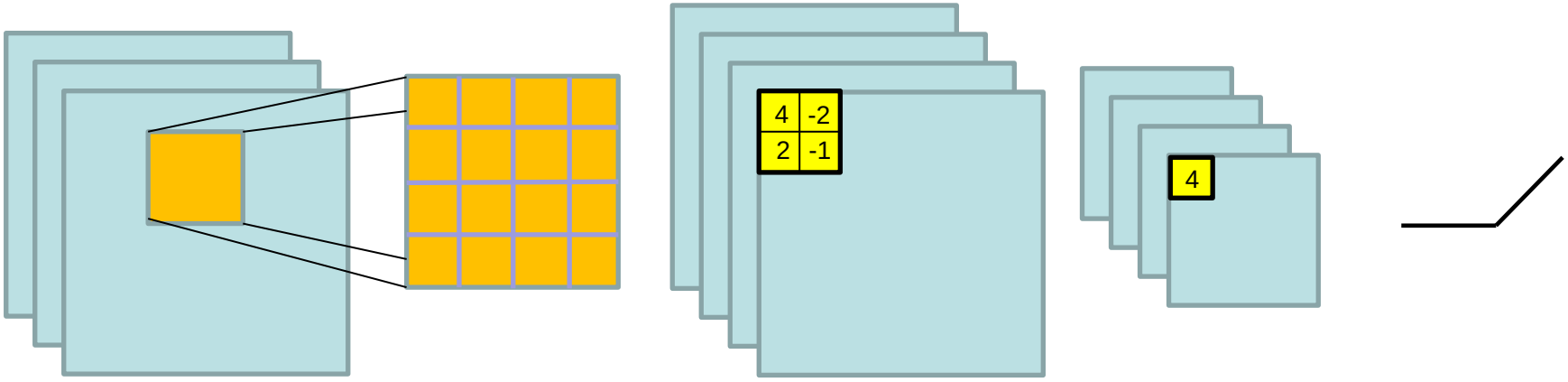
Convolutional network architecture for image classification



Classification network

Single layer module of a convolutional network





Input
(image or feature map from previous layer)

m filter masks
(learned)

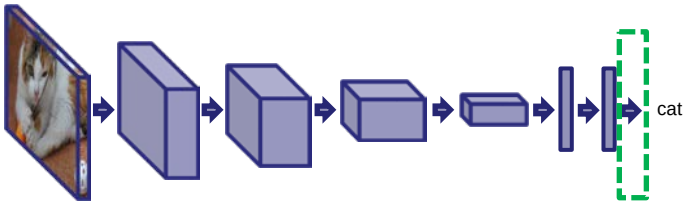
Feature map
(m filtering outputs)

Max-pooling

Nonlinearity
ReLU

- **Max-pooling** replaces a small local area of the feature map by the maximum value in that area (= downsampling)
  → data reduction, loss of localization accuracy
  → allows for larger receptive fields in the next layer

Softmax layer and cross-entropy loss



- The softmax function

$$\text{softmax}(a_j) = \frac{\exp a_j}{\sum_{k=1}^{|\mathcal{C}|} \exp a_k}$$

emphasizes the strongest activations a_j within an activation vector $\mathbf{a}$, makes all components positive, and normalizes its sum to 1

- The regression loss

$$\mathcal{L}(\mathbf{w}; D) = \sum_{i \in D} \|\mathbf{y}(\mathbf{w}, \mathbf{x}_i) - \mathbf{t}_i\|^2$$

is not optimal if the target values $\mathbf{t}_i$ are one-hot vectors for class labels

- Cross-entropy loss (ground truth vs. estimated class distribution):

$$\mathcal{L}(\mathbf{w}; D) = \sum_{i \in D} \sum_{c=1}^{|\mathcal{C}|} t_{ci} \log(\mathbf{y}_c(\mathbf{w}, \mathbf{x}_i))$$

Convolutional networks for classification

- Many layers with increasing receptive field sizes are perfect for learning abstract features (parts, attributes) from weakly labeled data (class output)

  → Powerful feature representation of image content

- The network shares features when trained for multiple classes

  → more efficient on multi-class problems than on small problems

- Large multi-class networks transfer well to similar classification tasks

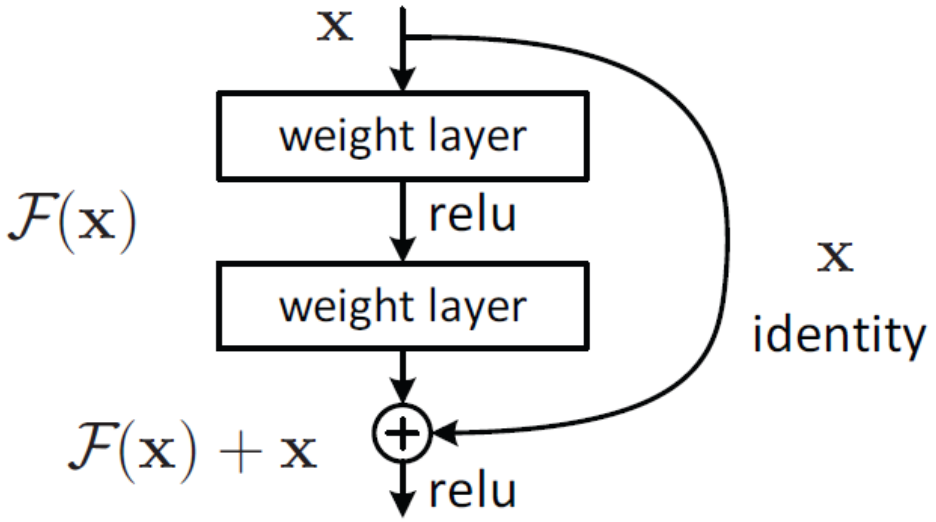
  → Adapt pre-trained network by running gradient descent on new samples (**fine-tuning, few-shot learning**)

- Deeper networks with smaller filters are more efficient (a bit harder to train)

  → Replace one layer with 7x7 filters by 3 layers with 3x3 filters

Residual networks (He et al. 2016)

- Idea: incremental features
- Incremental learning is possible but relatively hard for conventional networks
- Residual layers allow the network to learn directly the residual that is added to the representation from the previous layer



Residual layer consisting of two standard layers and the identity bypass

- Residual networks can be made very deep without negative effects (>100 layers)
- It was shown that residual networks can act as ensembles of smaller networks (Veit et al. 2016)

Classification examples

GT: mountain tent

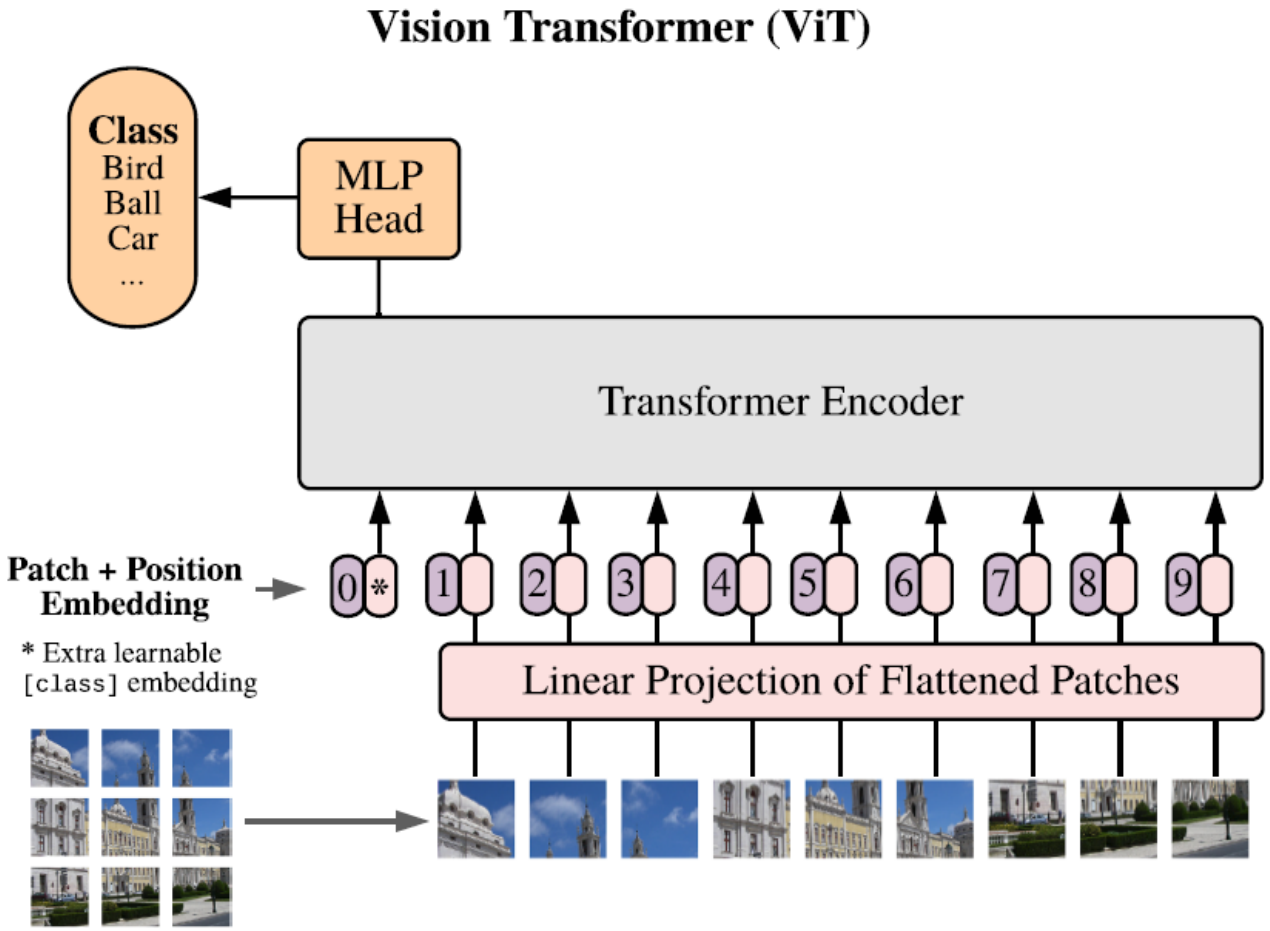
1: sleeping bag
2: mountain tent
3: parachute
4: ski
5: flagpole

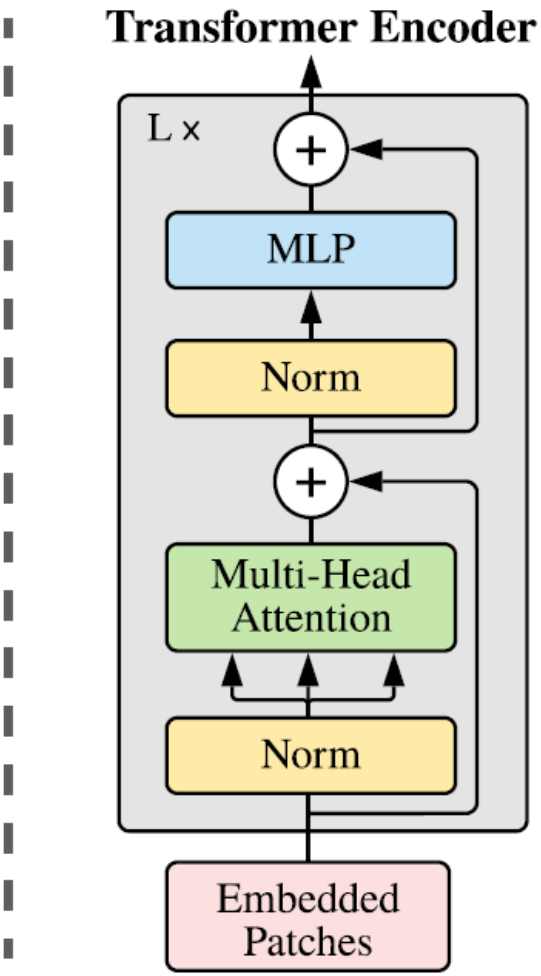
GT: geyser

1: geyser
2: volcano
3: sandbar
4: breakwater
5: leatherback turtle

Training and testing on ImageNet with 1000 object classes and more than a million images in the training set

Vision transformer architecture

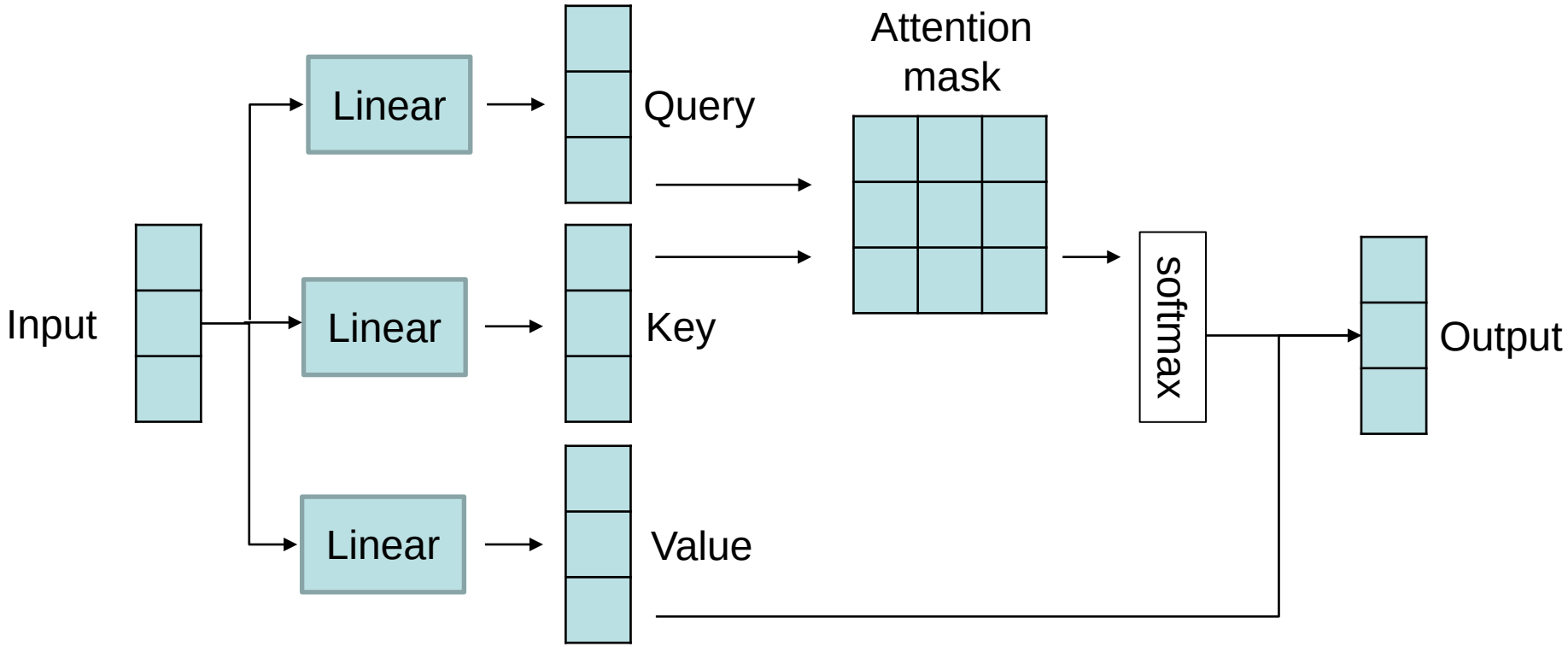




Dosovitskiy et al. 2021

Self-attention (more details in Deep Learning class 11)

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Learned parameters

Costly: $O(N^2)$

Vaswani et al. 2017

Notes about transformers

- Motivation from language processing I:
  Inputs with varying number of tokens

- Global, fully connected interaction between all tokens already in first layer
  → Full receptive field from the beginning (has pros and cons)

- Permutation invariance: tokens are not bound to a grid position
  → positional encoding

- Motivation from language processing II:
  Self-attention determines similarities between tokens
  → learned feature similarity to establish grouping
  → grouping can be extracted from attention mask

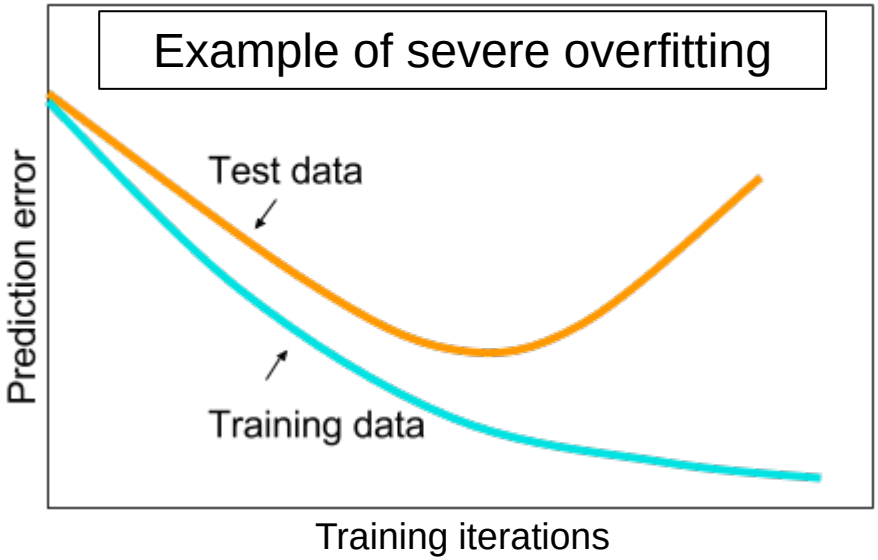
- Attention mask can be interpreted as input-dependent convolution

- Transformers include graph convolutional networks

Overfitting

- Deep networks have millions of parameters and are prone to overfitting

→ Always verify that your network does not overfit



1. Check training and validation curves

2. Evaluate your network on a different dataset (cross-dataset generalization)

3. Visualize your results and do a qualitative sanity check
   Don’t trust the numbers alone

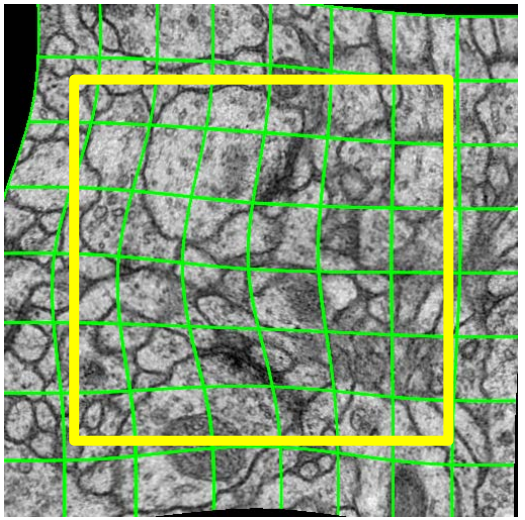
What can I do, if the network overfits?

- Try with a smaller network (fewer parameters)
- Increase the number of training samples
- Add artificial variation to the dataset → **data augmentation**
- Try standard regularization techniques
  - Dropout
  - Weight decay (penalty on size of weights)
  - Smaller batches
  - Ensembles

Data augmentation

- Training data is valuable
  → create additional data by applying standard transformations to the data

- Typical transformations
  - Rotation
  - Scaling
  - Left-right flip
  - Brightness, contrast, color
  - Blur
  - Non-rigid deformation



- Transformations depend on the task
  Which transformation counts as information, which as nuisance variation?

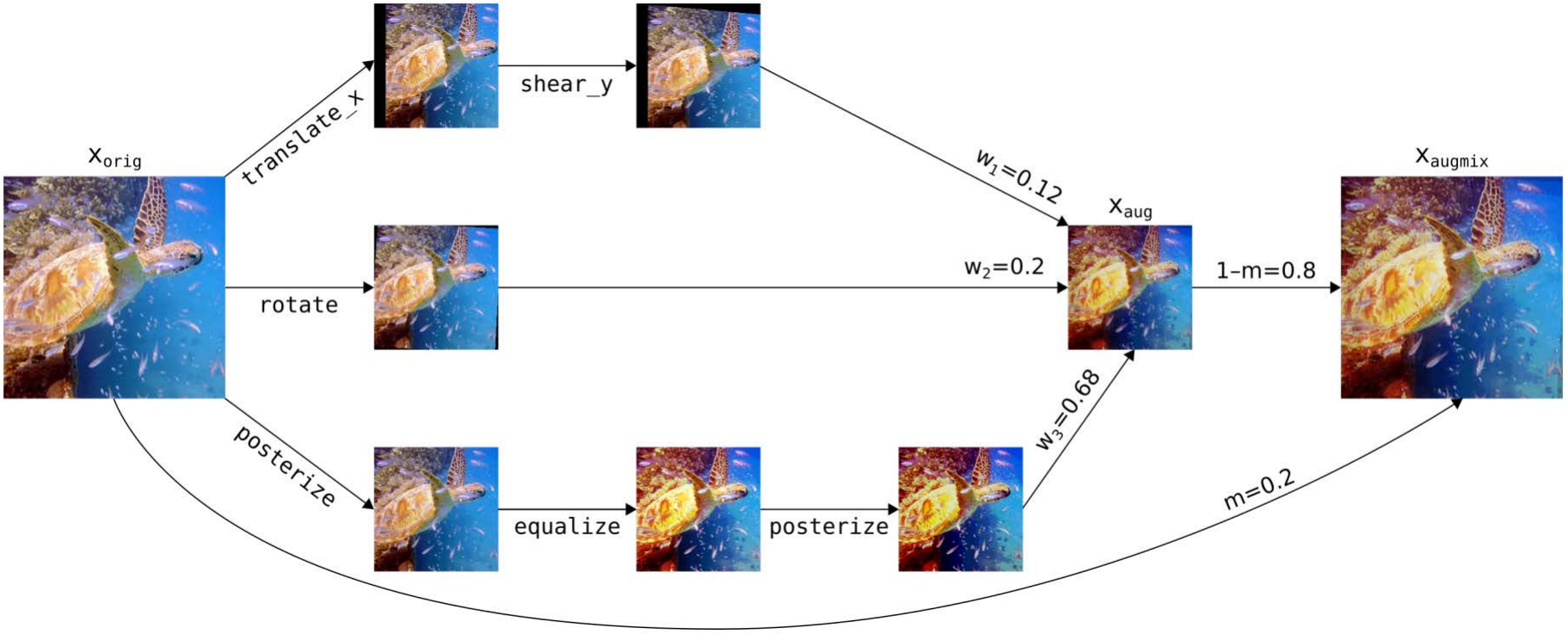
- <u>Example:</u>
  If you want to distinguish red apples from green apples, you should not apply extreme color transformations.

Heavy data augmentation, AugMix

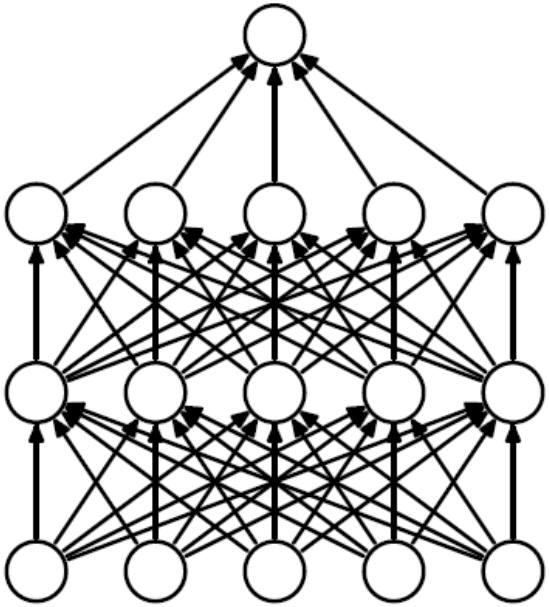
- Early deep learning methods used only flipping and cropping

- Data augmentation is the best lever for generalization
  → use more diverse data augmentation transformations

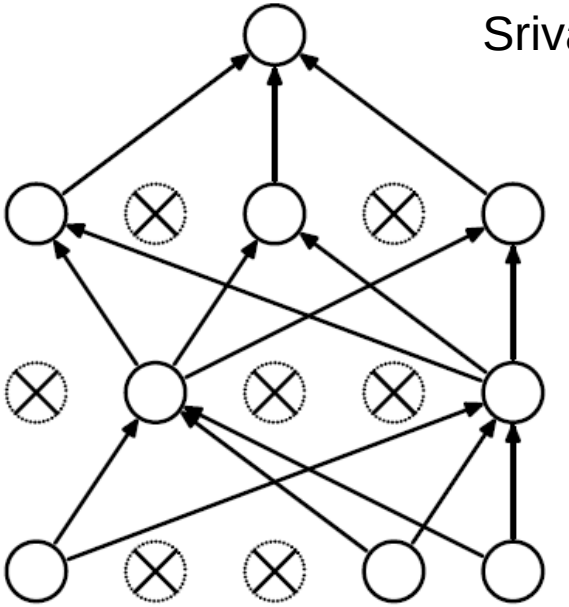
- AugMix (Hendricks et al. 2020)



Regularization with dropout



Network without dropout



Network with dropout

Srivastava et al. 2014

- During training, switch off each unit of a layer with a certain probability
- During testing, all units are present and all weights are multiplied by the probability
- Effect: multiple units must take over a task collectively

Adapting a network to another dataset: finetuning

- Training deep networks requires large training sets

- What if I have a new dataset that is small?

→ Use a network **pre-trained** on a large dataset (e.g. ImageNet) and **finetune** it on the new dataset

Idea: most features from the large dataset are relevant also for the new dataset → only the high-level features must be adapted to the task

Problem: if the new dataset is too small, finetuning will **overfit** to the training data eventually

Solutions:

- Only finetune for a small number of iterations (**early stopping**)
- Keep most layers of the pre-trained network fixed (**freezing**)

Catastrophic forgetting and knowledge distillation

- Incremental learning (incl. finetuning) on new data (new classes, new domains) leads to collapse of the learned representation

  → Affects old knowledge (it gets forgotten)
  → Also affects the new classes, as old knowledge cannot be exploited

- Common remedy: additional knowledge distillation loss

$$\mathcal{L}_{\mathcal{X}} = \mathcal{L}_{\mathcal{X}}^{CE} + \lambda * \mathcal{L}_{\mathcal{X}}^{KD}$$

- Forces the activations (often before the final softmax) of the new model on the new data to be close to the output of the old model

  → Keeps a large part of the original representation

Batch normalization and other normalization techniques

- The normal training procedure does not put constraints on the size and compatibility of the unit activations

  → Some units dominate, and it will take many iterations to correct this later in training

- Remedy: a normalization procedure during training that ensures comparable activation magnitudes

- Different flavors
  - Batch normalization
  - Layer normalization
  - Group normalization
  - [...]

Batch normalization

- Observation: the activations depend on the activations of the input units

- In batch normalization, the statistics for the normalization are computed from each mini-batch of size m:
  - mean: $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$
  - variance: $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$

- Activations are normalized by these statistics:

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

- Additional learnable parameters ensure that the model keeps its representation power:

$$y_i \leftarrow \gamma \widehat{x}_i + \beta$$

Issues with batch normalization

- Small minibatches (extreme case m=1) yield weak statistics

  → After training, compute final statistics on multiple minibatches
  → Alternatives: layer norm, group norm

- Batch normalization increases sensitivity to distribution shift between training and test data

  → Recompute statistics on target data (if available)

- Batch normalization generally problematic when data is not i.i.d. (e.g. in reinforcement learning)

The success of deep learning: some reasons for surprise

1. The objective function is non-convex and should be hard to optimize (local minima, saddle points)

   Why does optimization yield such good minima?

2. There are far more parameters in the network than there is training data. Why do networks generalize? Where is the regularization?

3. The larger the network, the better the performance, even for small tasks. Why is there no bias-variance tradeoff?

Summary

- Deep networks can learn a (potentially very complex) function from a (large) set of input-output pairs
- Learning is an optimization problem that is solved with stochastic gradient descent (or variants of it)
- For high-dimensional inputs, such as images, convolutional networks are the dominant architecture (together with transformers)
- When applied to image classification, deep networks learn a feature hierarchy with increasingly abstract features near the class output
- Overfitting is a constant danger in deep learning and must be monitored carefully
- Data augmentation generates additional artificial data

References

- A. Krizhevsky, I. Sutskever, G. Hinton: Imagenet classification with deep convolutional neural networks, *Neural Information Processing Systems (NIPS)*, 2012.

- N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research* 15:1929-1958, 2014.

- K. He, X. Zhang, S. Ren, J. Sun: Deep residual learning for image recognition, *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

- A. Veit, M. Wilber, S. Belongie: Residual networks behave like ensembles of relatively shallow networks, *Neural Information Processing Systems (NIPS)*, 2016.

- A. Dosovitskiy et al.: An image is worth 16x16 words: transformers for image recognition at scale, *International Conference on Learning Representations (ICLR)*, 2021.