Karlsruhe Institute of Technology

Machine Vision

Chapter 12: Deep Learning

Dr. Martin Lauer **Institut für Mess- und Regelungstechnik**

MULTI-LAYER PERCEPTRONS
(AS ONE KIND OF ARTIFICIAL NEURAL NETWORKS)

Multi-Layer Perceptrons (MLP)

- MLPs are highly parameterized, non-linear functions

$$f_{MLP}^{\vec{w}} : \mathbb{R}^n \rightarrow \mathbb{R}^q$$

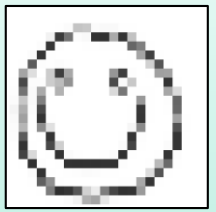
$$f_{MLP}^{\vec{w}} : \vec{x} \mapsto \vec{y}$$

parameter vector, weight vector

(input-) pattern

output

example: classification of images

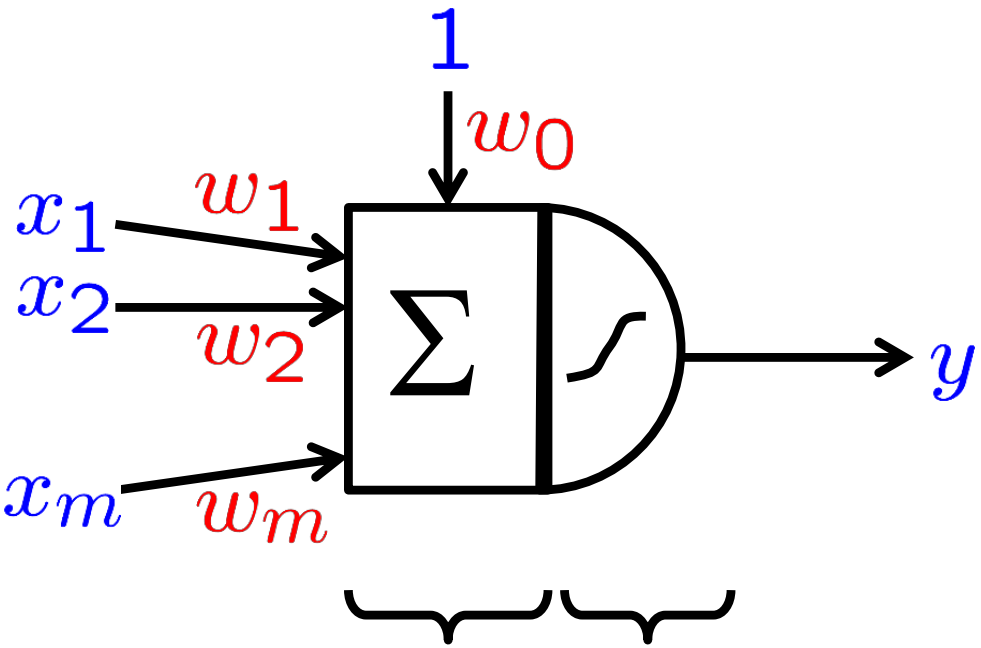


$\vec{x}$ feature vector,
e.g. vector of all gray values in image

$\vec{y}$ 1-of-q-vector that models probabilities for each of q possible categories, e.g. smiley is happy/sad/frustrated

Internal Structure of MLPs

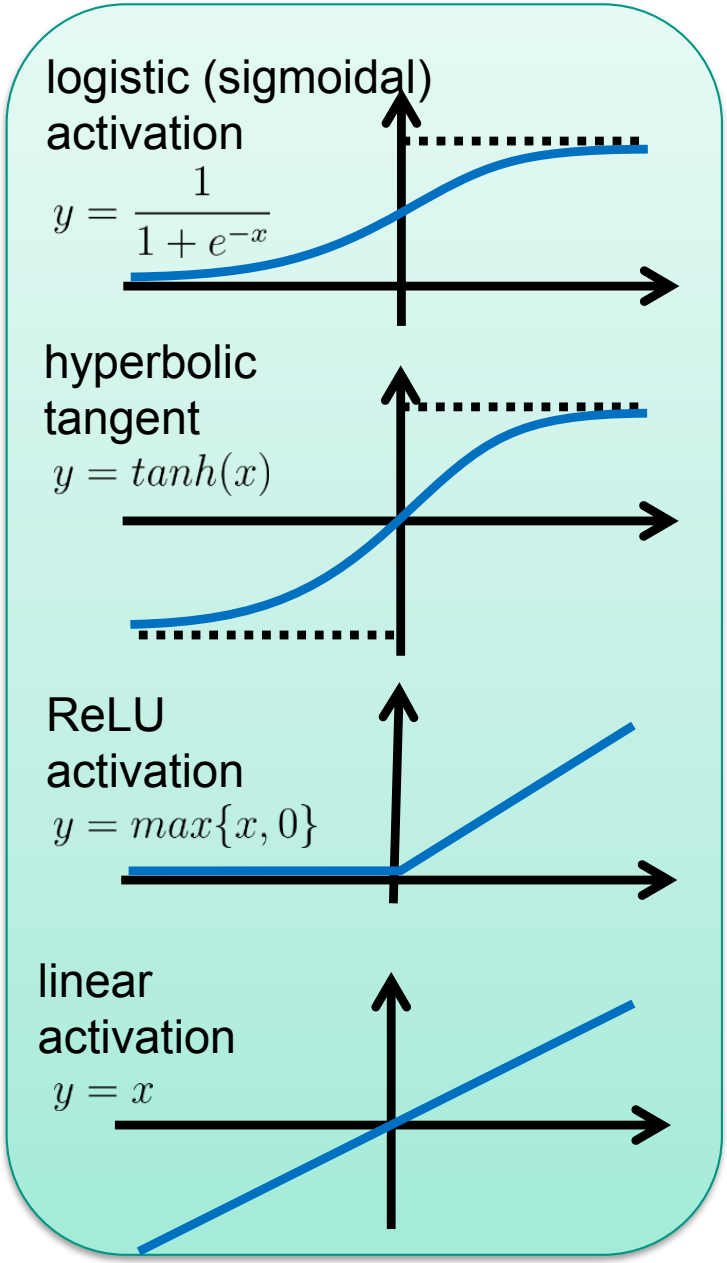
- perceptron



linear combination of inputs and weights

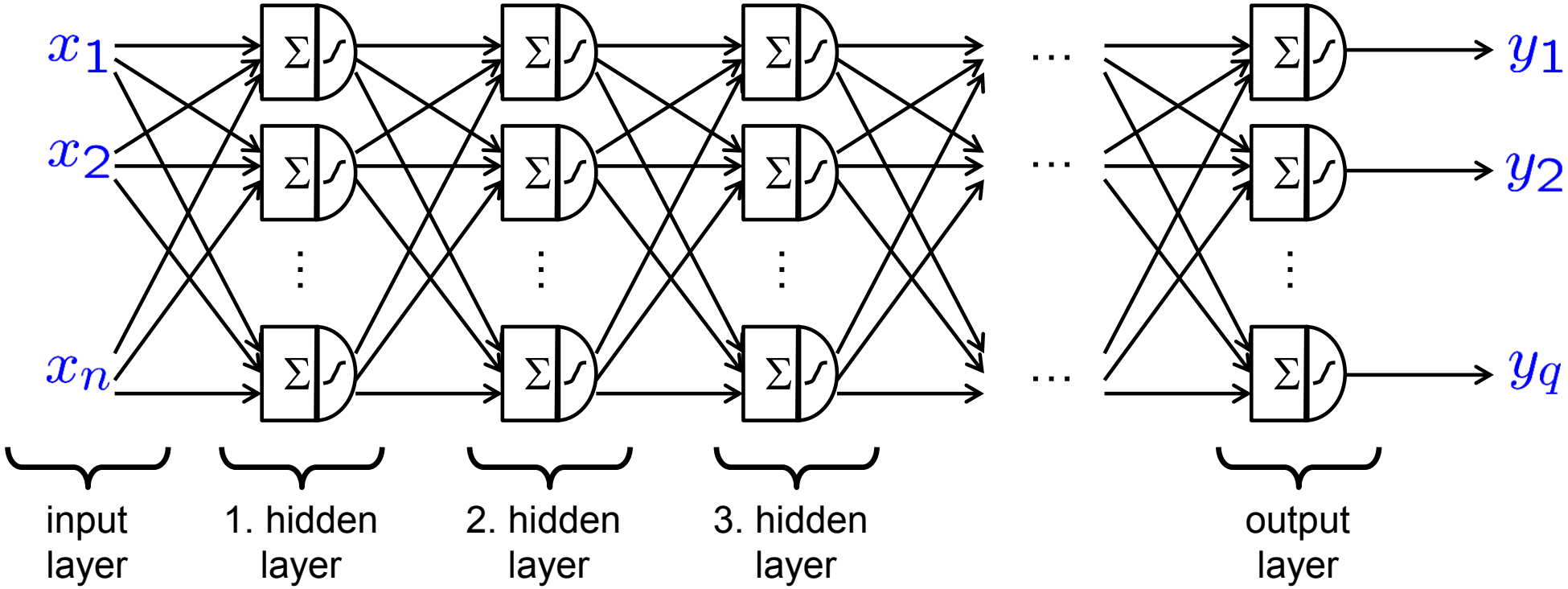
non linear activation function

$$y = f_{act}\left(w_0 + \sum_{i=1}^{m} w_i x_i\right)$$



Internal Structure of MLPs

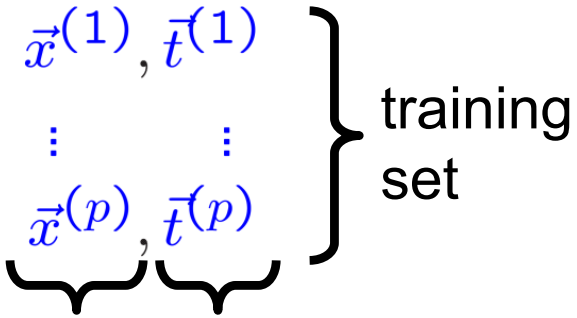
- layered arrangement of many perceptrons



  - network structure creates set of highly nonlinear function
  - many weights
  - deep architectures: typically >5 hidden layers

Training of MLPs

- how do we determine weights of MLP?
  - basic idea: minimize error for training examples



  - solve $\underset{\vec{w}}{minimize} \sum_{j=1}^{p} err\left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)$

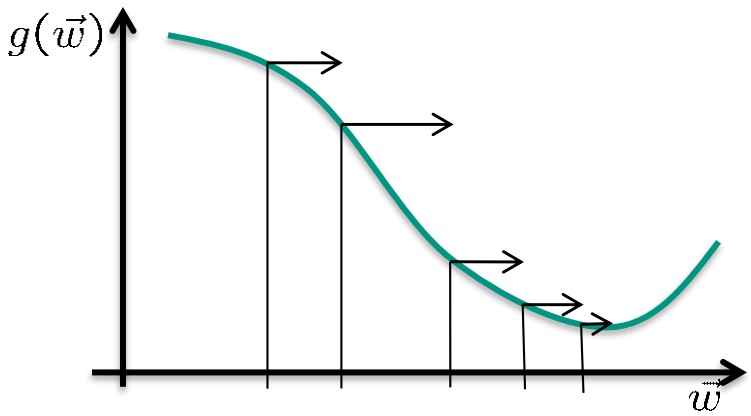
    for appropriate error measure (loss function) err
  - algorithm: gradient descent (backpropagation)

Gradient Descent (Backpropagation)

- goal:

$$\underset{\vec{w}}{minimize}\ g(\vec{w}) \quad \text{with } g(\vec{w}) := \sum_{j=1}^{p} err\left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)$$

- algorithm:
  1. initialize weights $\vec{w}$ randomly with small numbers
  2. calculate gradient $\frac{\partial g(\vec{w})}{\partial \vec{w}}$
  3. update weights $\vec{w} \leftarrow \vec{w} - \varepsilon \frac{\partial g(\vec{w})}{\partial \vec{w}}$ with small learning rate $\varepsilon > 0$
  4. goto 2 until stopping criterion reached

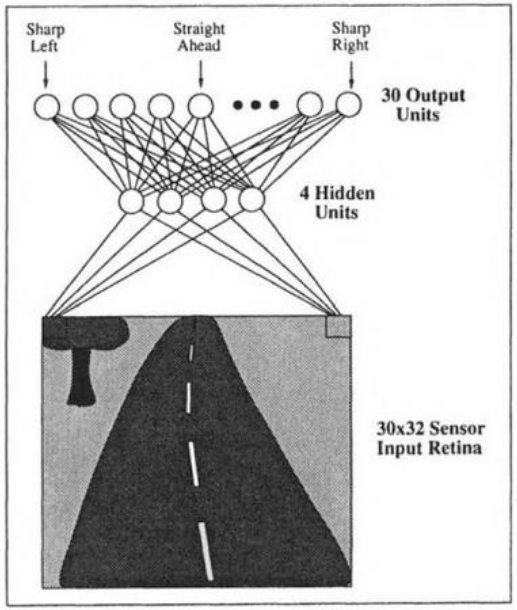


- improvements:
  – discussed later

Training MLPs (traditional methods)

- problems with traditional training methods
  - too many weight, too few training examples
  - too slow
  - numerical problems, local minima

  ⇒ overfitting, underfitting, insufficient generalization
- traditional techniques to overcome problems
  - regularization (e.g. early stopping, weight decay, Bayesian learning)
  - preprocessing of patterns, feature extraction, reduction of dimensionality
  - choose smaller MLPs, less layers, less hidden neurons, network pruning
  - replace neural networks by other methods (e.g. SVMs, boosting, etc.)



ALVINN-architecture
taken from: D. A. Pomerleau, „Neural network perception for mobile robot guidance“, 1993

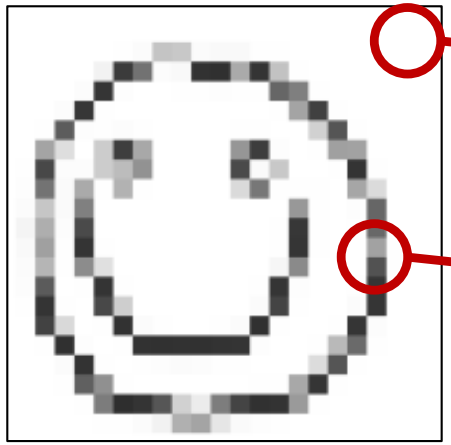
Deep Learning

- what is different in Deep Learning?
  - larger training sets (millions instead of hundreds)
  - more powerful computers, parallel implementations on multi-core CPUs and GPUs
  - special network structures
    - autoencoders
    - convolutional networks
    - recurrent networks/LSTM
    - (deep belief networks/restricted Boltzmann machines)
    - …
  - weight sharing
  - layer-wise learning
  - dropout
  - learning of useful features
  - learning from unlabeled examples

Learning of Features

- observation:
  - many pixels do not provide much information
  - neighboring pixels are highly correlated

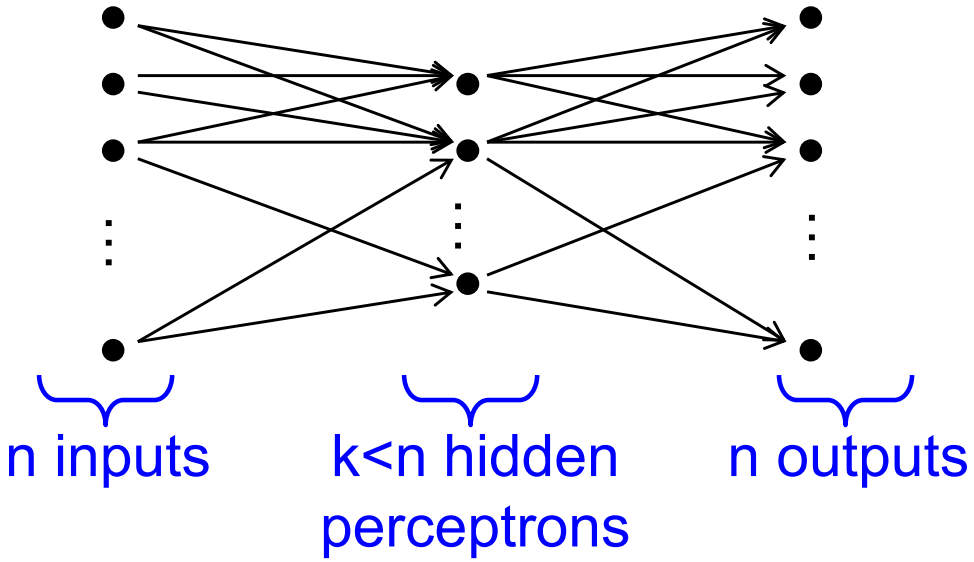
- example: smileys



- how can we seperate relevant information form irrelevant information?

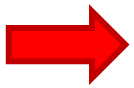
Autoencoder

- MLP with such a structure



- learn identity function

$$\underset{\vec{w}}{minimize} \sum_{j=1}^{p} \left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}) - \vec{x}^{(j)} \right)^2$$

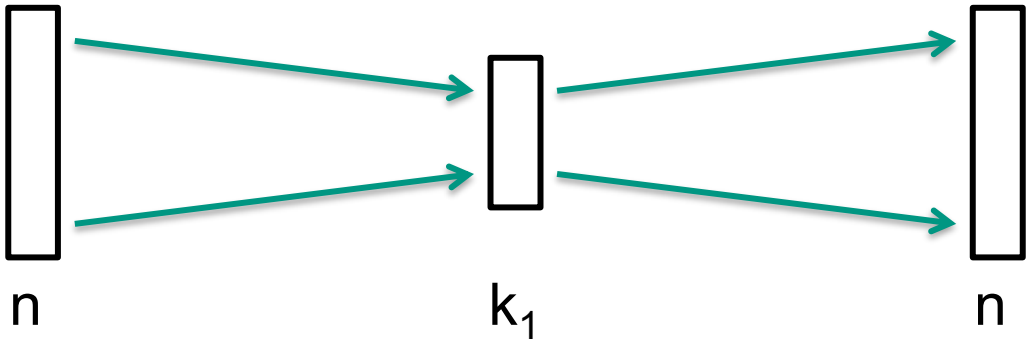


hidden layer must compress image content

kind of neural principal component analysis

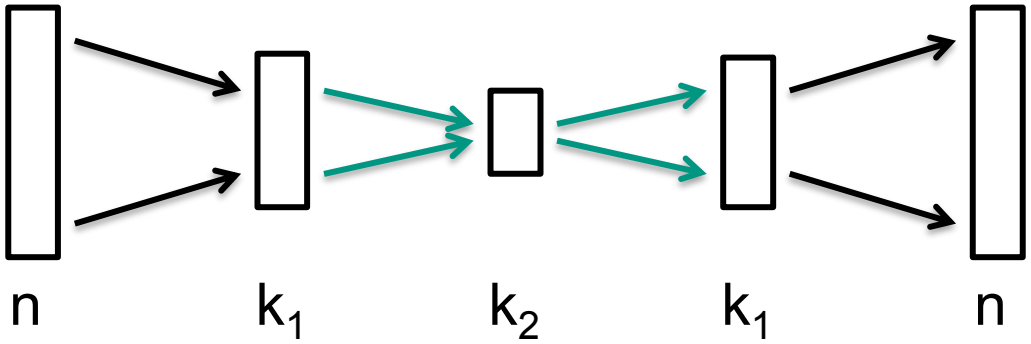
Stacked Autoencoders

- incremental training of multi-layered autoencoders
  1. train autoencoder with single hidden layer



train weights of these connections

  2. extend autoencoder by additional hidden layer



keep weights of these connections unchanged from previous step

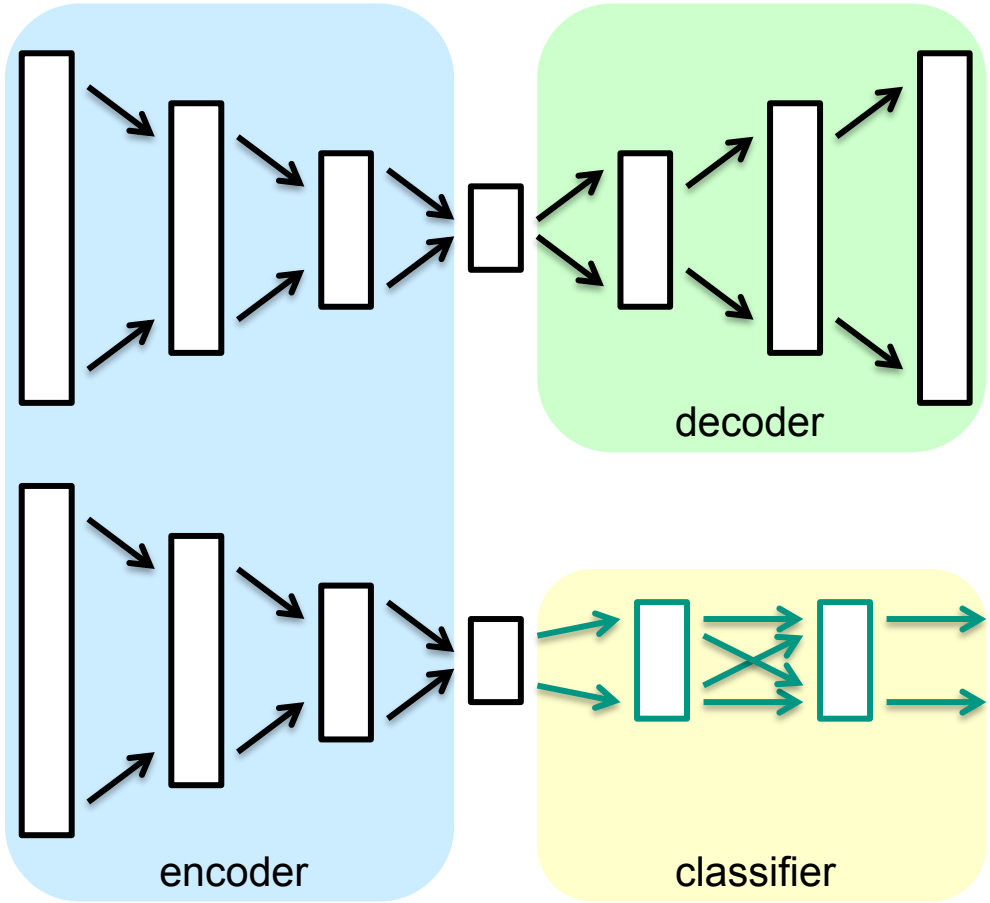
  3. repeat process analogously to add further hidden layers

compression of information increases from layer to layer

non-linear, multi-layered principal component analysis

Stacked Autoencoders for Classification

1. train stacked autoencoder
2. replace decoder network by fully-connected classifier network
3. train classifier network
4. train all weights of encoder and classifier network for a few iterations

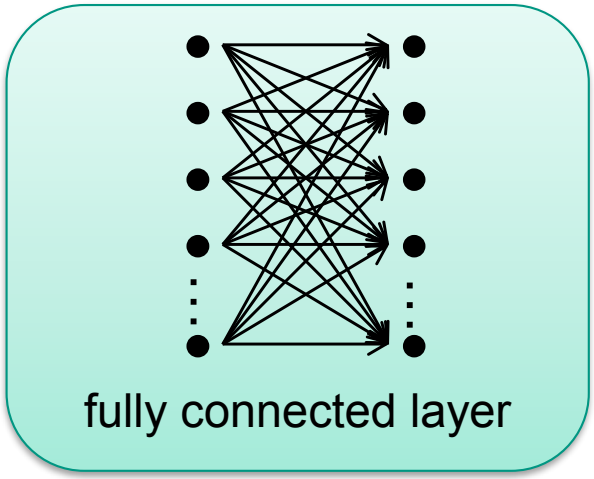


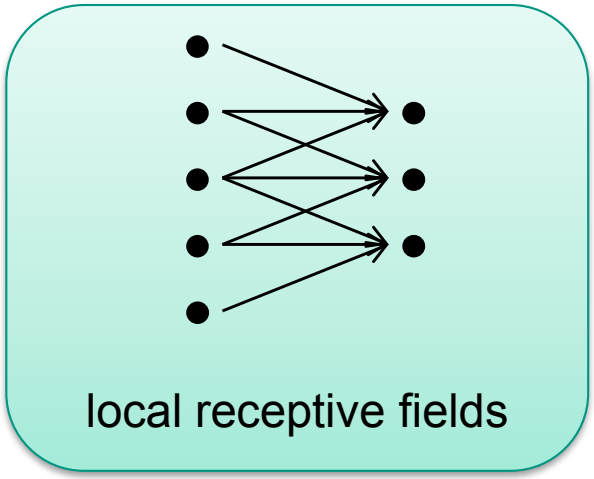
advantages:

- stacked autoencoder can be trained with unlabeled examples
- incremental training achieves better results

Local Receptive Fields

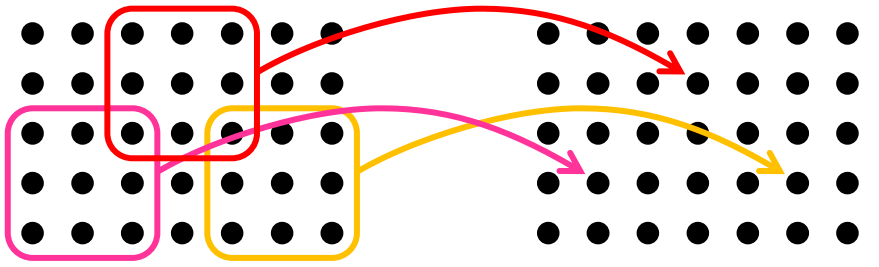
- local receptive fields for structured data





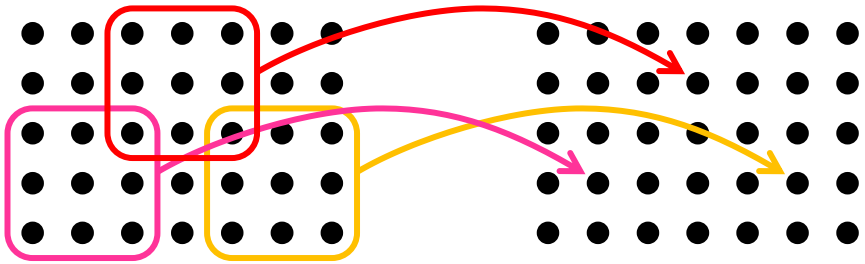
Local receptive fields force the network to process information locally.

example: local features for images



Weight Sharing

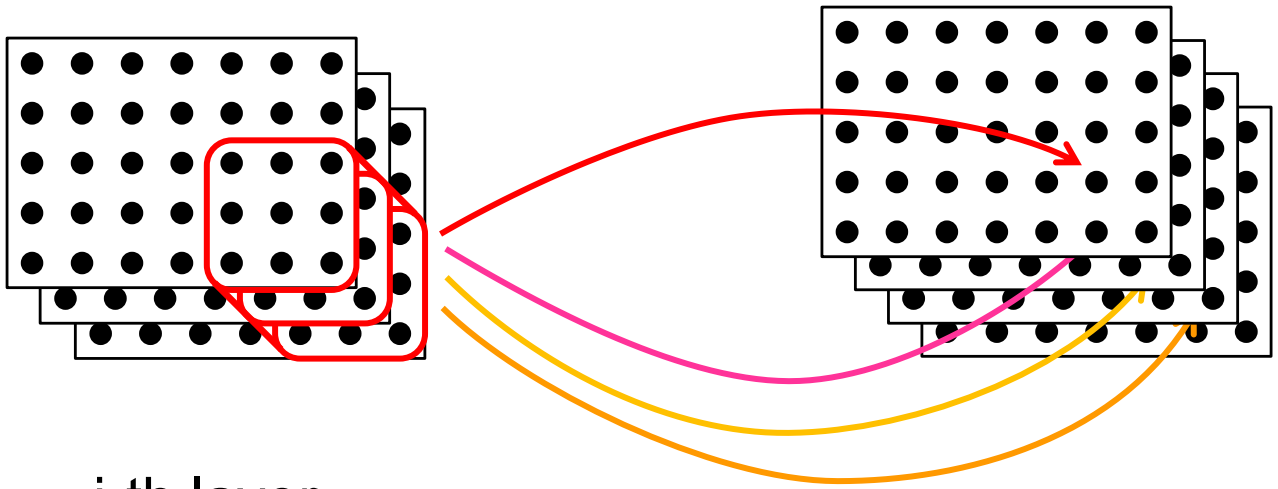
- can we generate the same local features for all pixels?
  - weight sharing: binding the weights of different perceptrons
  - convolutional layers: binding the weights of all perceptrons of one layer



$$L_{i+1} = f_{act}(L_i * \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} + \boxed{w_0})$$

Multi-Channel Feature Layers

- in each hidden layer one would like to compute several different features for each pixel → mulit-channel layers



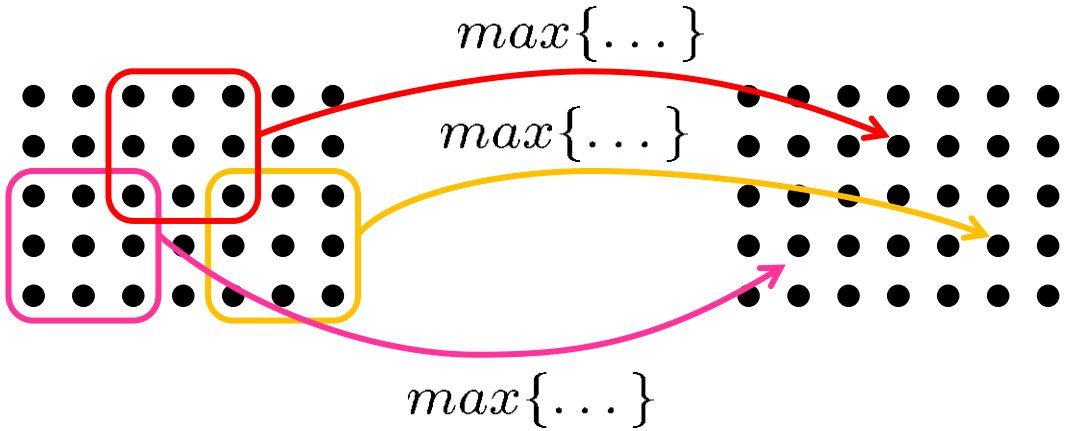
i-th layer
k channels

(i+1)-th layer
n channels

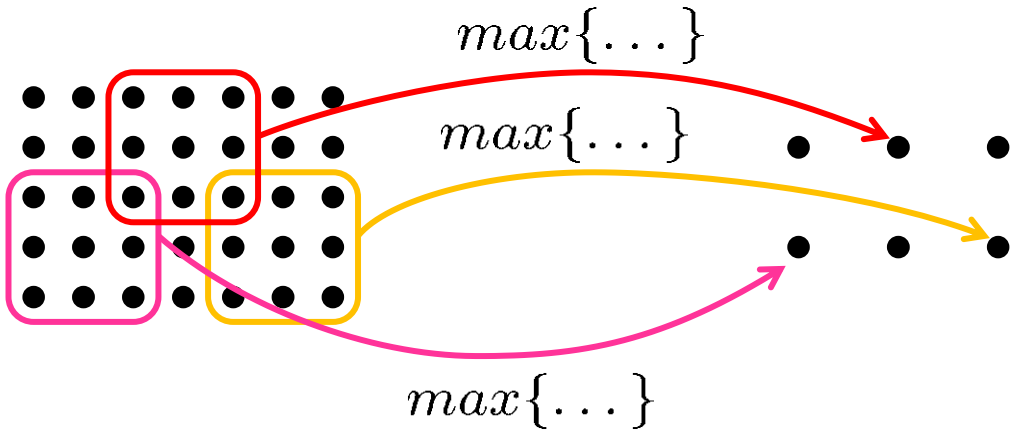
convolution kernels are tensors of size $h \times w \times k$

Max-Pooling

- pooling layers are designed to aggregate information spatially
- Max-Pooling: calculate maximum from local receptive field



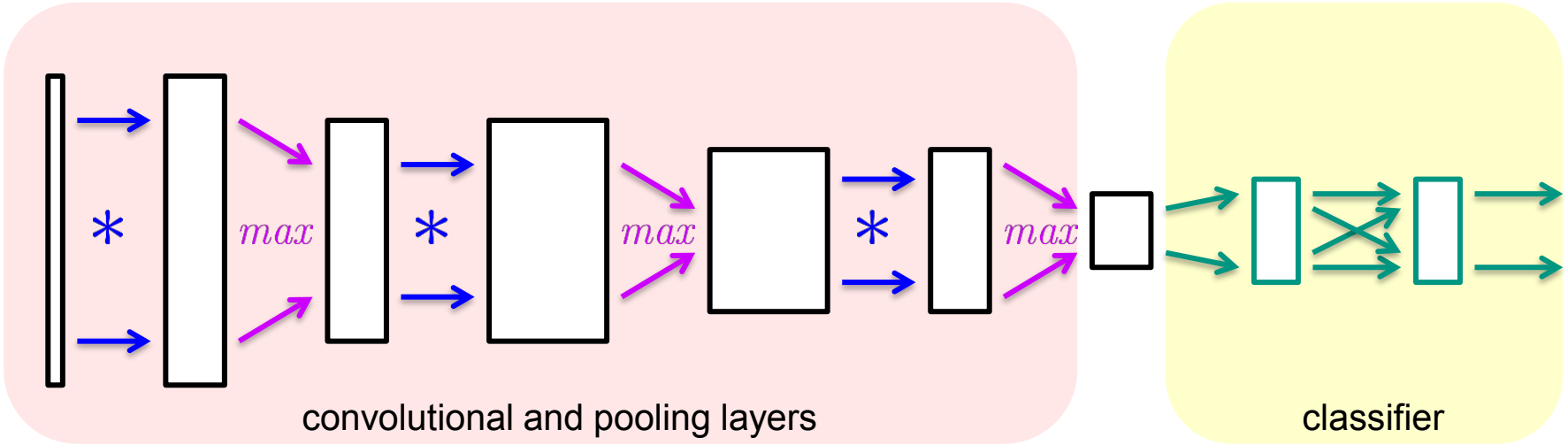
- pooling is often combined with reducing the resolution of layers



stride: resolution of previous layer/resolution of subsequent layer

Convolutional Networks

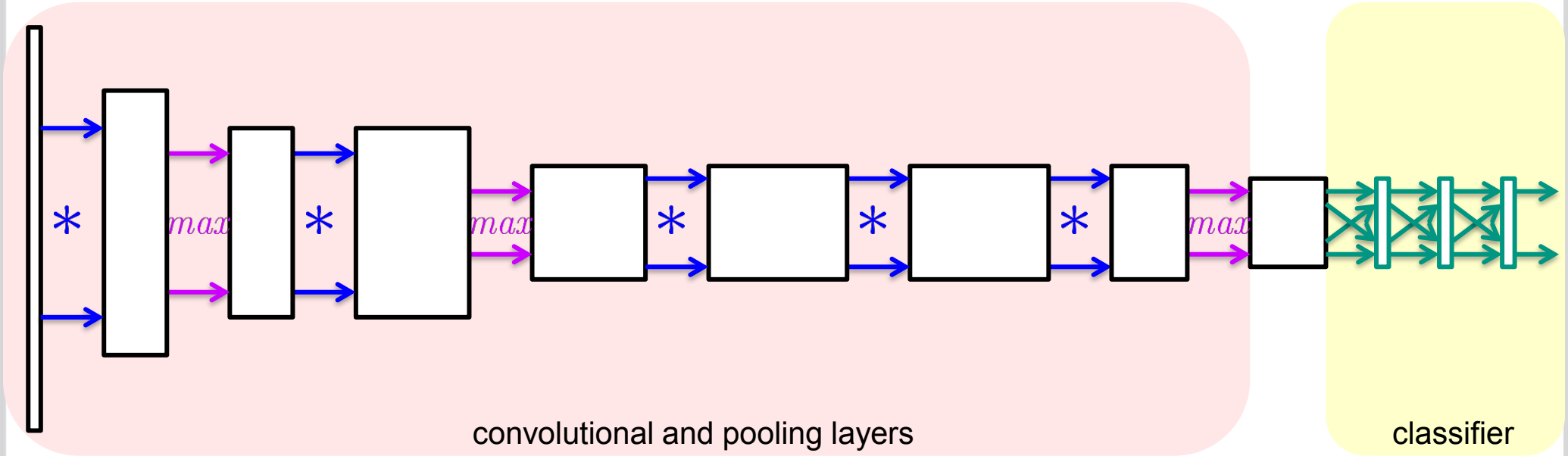
- Convolutional Neural Networks (CNNs) combine
  - convolutional layers
  - pooling layers
  - fully connected classifier network



Example: AlexNet

- A. Krizhevsky, I. Sutskever, G. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks“, NIPS, 2012

  – classification of images, 1000 categories
  – data set: 1,2 million images
  – approach: convolutional network, 60 millions weights



Example: AlexNet

- results on test set
  - top-1-error: 37%
  - top-5-error: 17%
- newer approaches:
  - top-5-error: <5%
    (better than humans)

Taken from:
http://image-net.org/challenges/LSVRC/2012/supervision.pdf

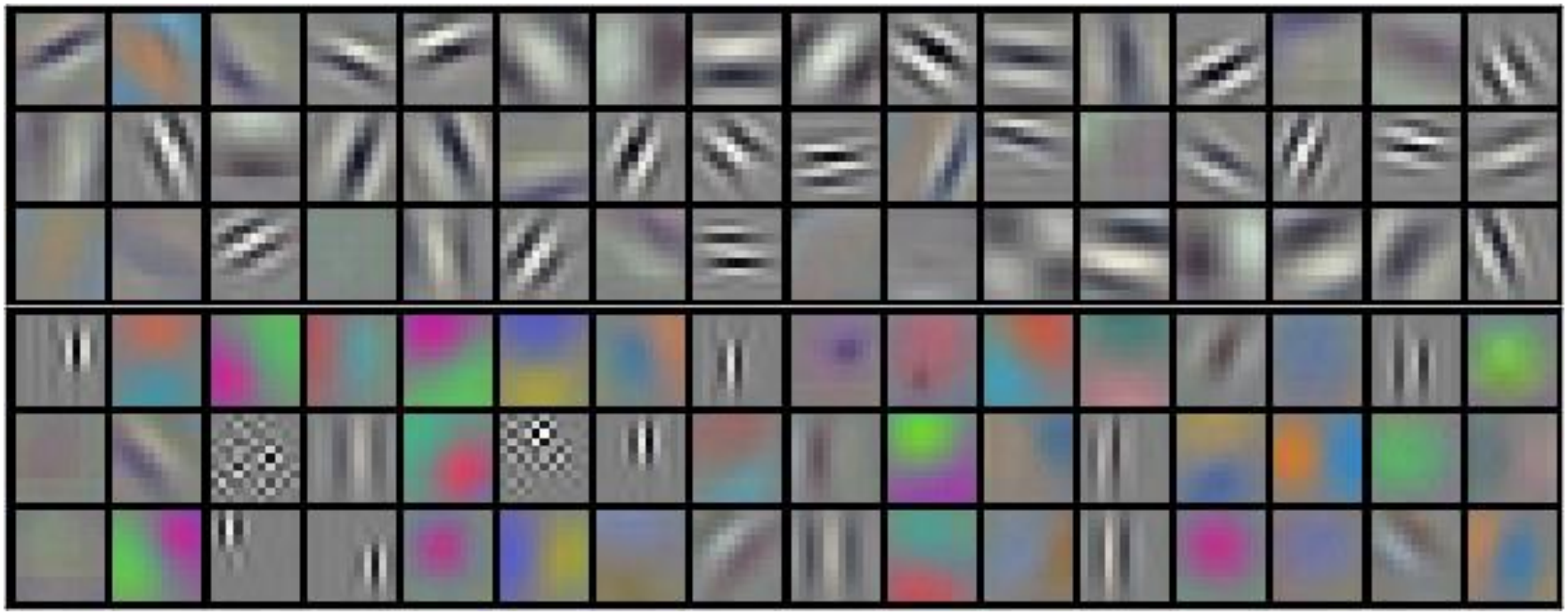
Example: AlexNet

- results on test set
  - top-1-error: 37%
  - top-5-error: 17%
- newer approaches:
  - top-5-error: <5%
    (better than humans)

Taken from:
http://image-net.org/challenges/LSVRC/2012/supervision.pdf

Convolutional Networks

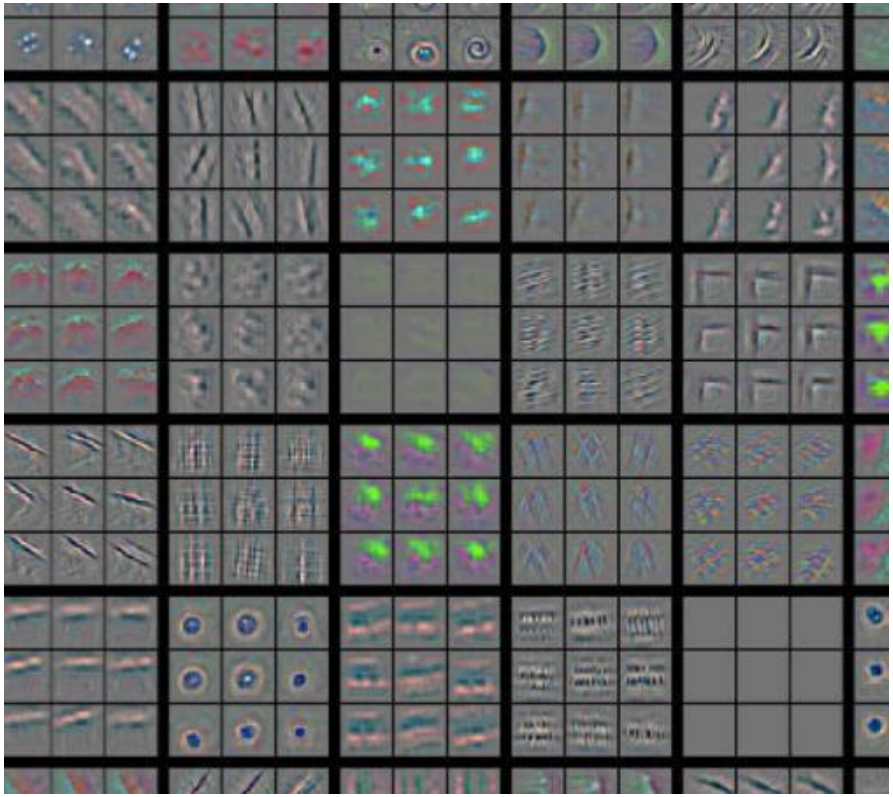
- which features are learned in hidden layers?
  - 1. layer: gray level edges, color edges, blobs



Taken from:
http://image-net.org/challenges/LSVRC/2012/supervision.pdf

Convolutional Networks

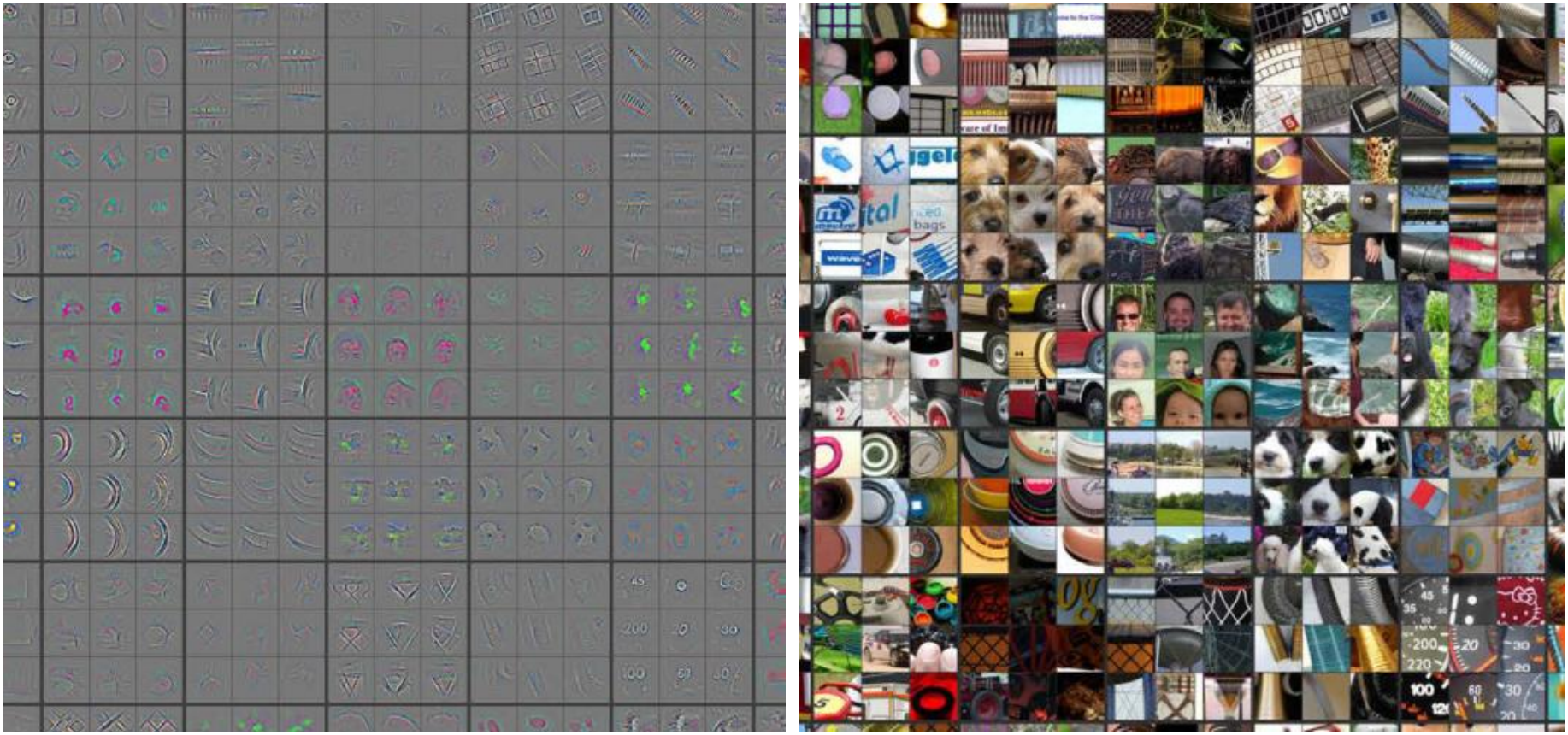
- which features are learned in hidden layers?
  - 2. layer: corners, round structures



Taken from:
M.D. Zeiler, R. Fergus, „Visualizing and Understanding Convolutional Neural Networks“, ECCV 2014

Convolutional Networks

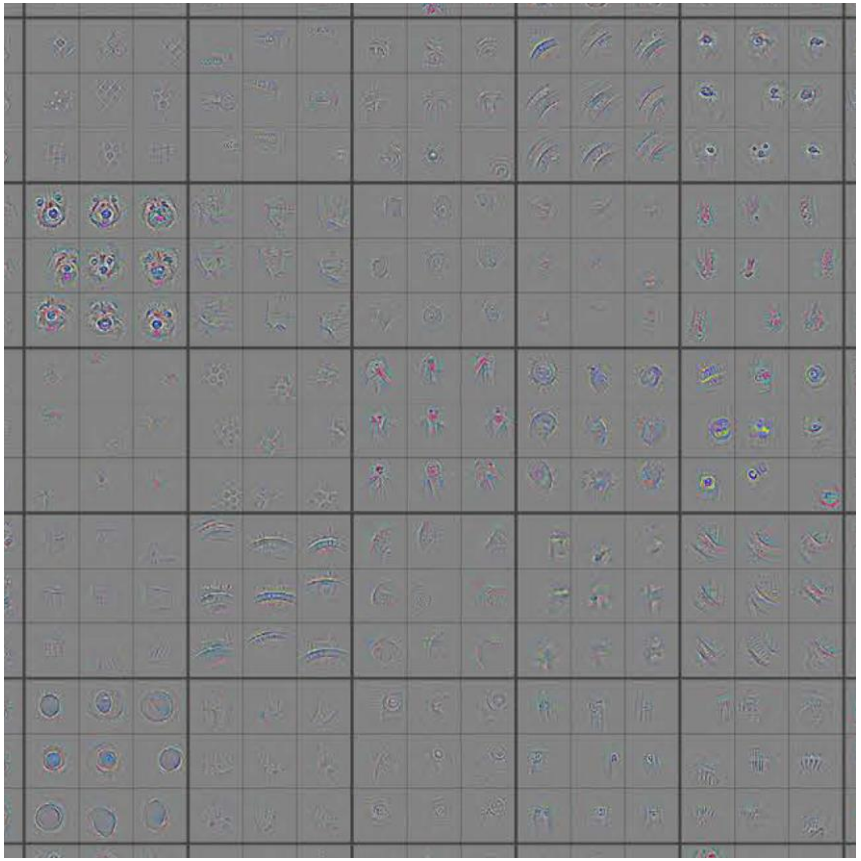
- which features are learned in hidden layers?
  - 3. layer: shapes, gratings

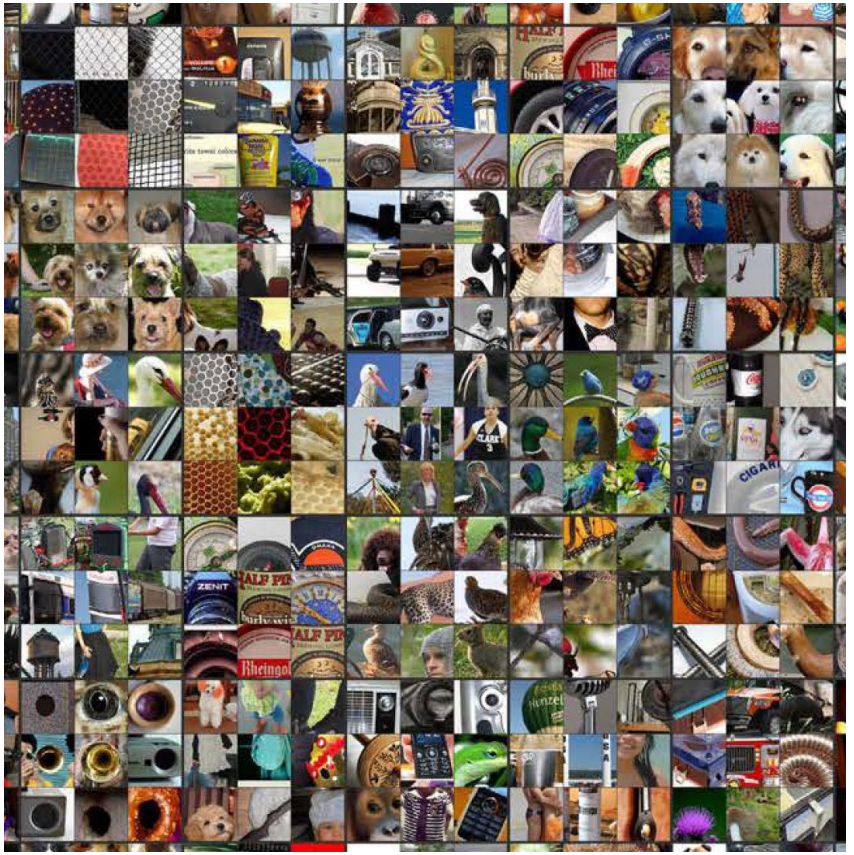


Taken from:
M.D. Zeiler, R. Fergus, „Visualizing and Understanding Convolutional Neural Networks“, ECCV 2014

Convolutional Networks

- which features are learned in hidden layers?
  - 4. layer: textured geometries

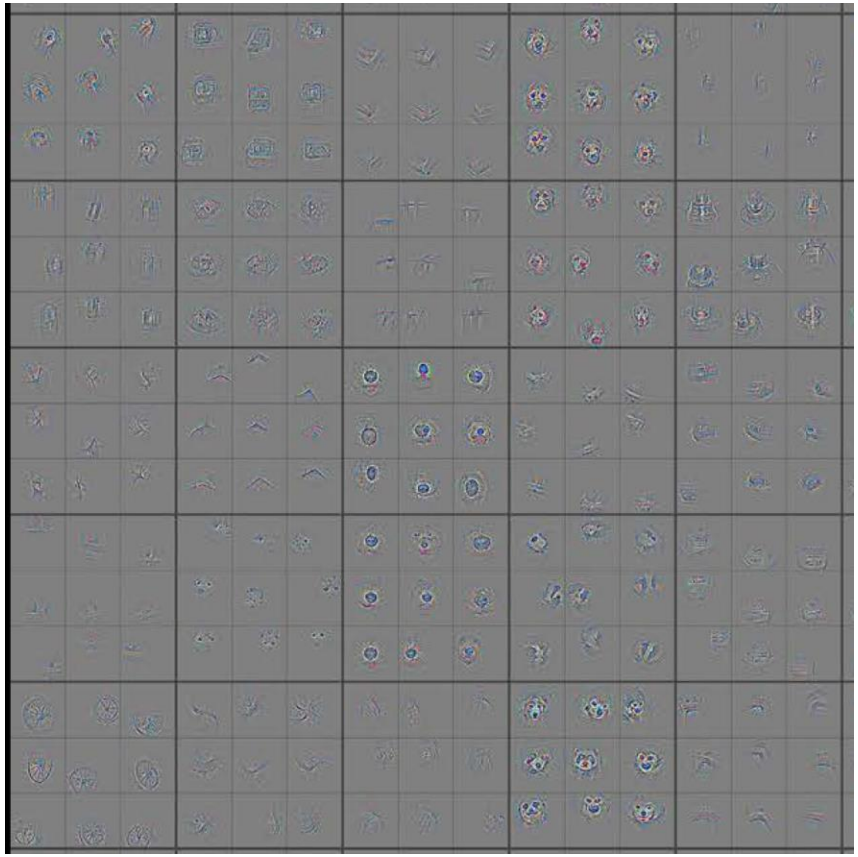




Taken from:
M.D. Zeiler, R. Fergus, „Visualizing and Understanding Convolutional Neural Networks“, ECCV 2014

Convolutional Networks

- which features are learned in hidden layers?
  - 5. layer: objects



Taken from:
M.D. Zeiler, R. Fergus, „Visualizing and Understanding Convolutional Neural Networks“, ECCV 2014

Convolutional Networks

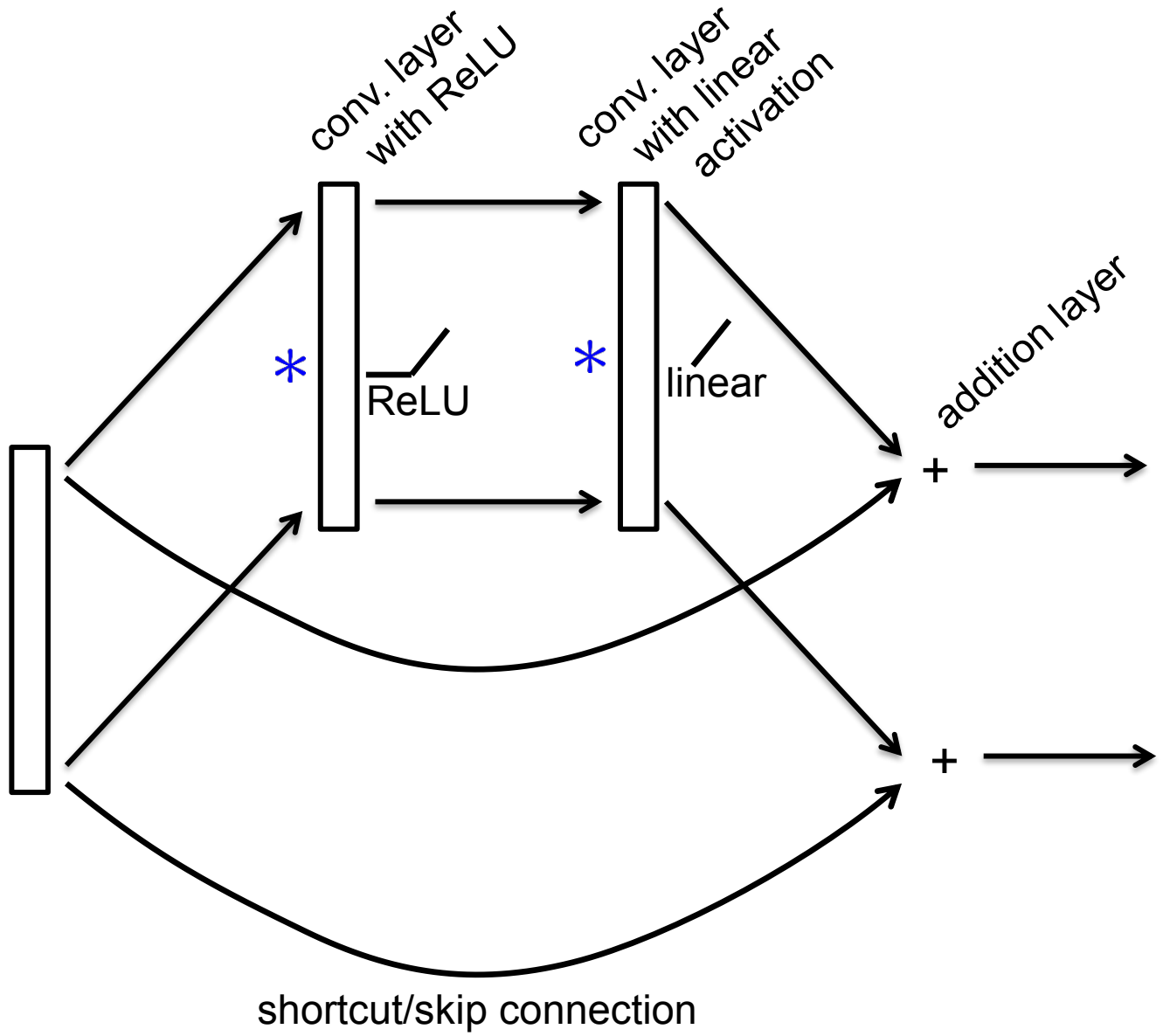
- From layer to layer…
  - features become more and more geometrically complex
  - features become more and more independent of position
  - features become more and more inpendent of pattern size
  - features become more and more specific

Popular Architectures for Image Classification

net	layers	kernel sizes	reference
LeNet5 (1998) (historical)	2 conv. layers 2 max pooling layers 2 fully conn. layers	5x5	Y. LeCun, L. Bottou, Y. Bengio, P. Haffner. *Gradient-based learning applied to document recognition*. Proc. of the IEEE, 1998
AlexNet (2012)	5 conv. layers 3 max pooling layers 3 fully conn. layers	3x3 – 11x11	A. Krizhevsky, I. Sutskever, G. E. Hinton. *Imagenet classification with deep convolutional neural networks*. NIPS 2012
VGG (2014)	13 conv. layers 5 max pooling layers 3 fully conn. layers	3x3 (1x1)	K. Simonyan, A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. arXiv 2014
ResNet (2015)	50/101/152 conv. layers (residual blocks) 2 pooling layers	3x3 (7x7)	K. He, X. Zhang, S. Ren, J. Sun. *Deep residual learning for image recognition*. CVPR 2016
GoogLeNet (2015)	2 conv. layers 9 inception layers* 4 pooling layers	1x1, 3x3, 5x5	C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, S. Erhan, V. Vanhoucke, A. Rabinovich. *Going deeper with convolutions*. CVPR 2015

* each inception layer consists of two internal layers and convolution kernels of various size

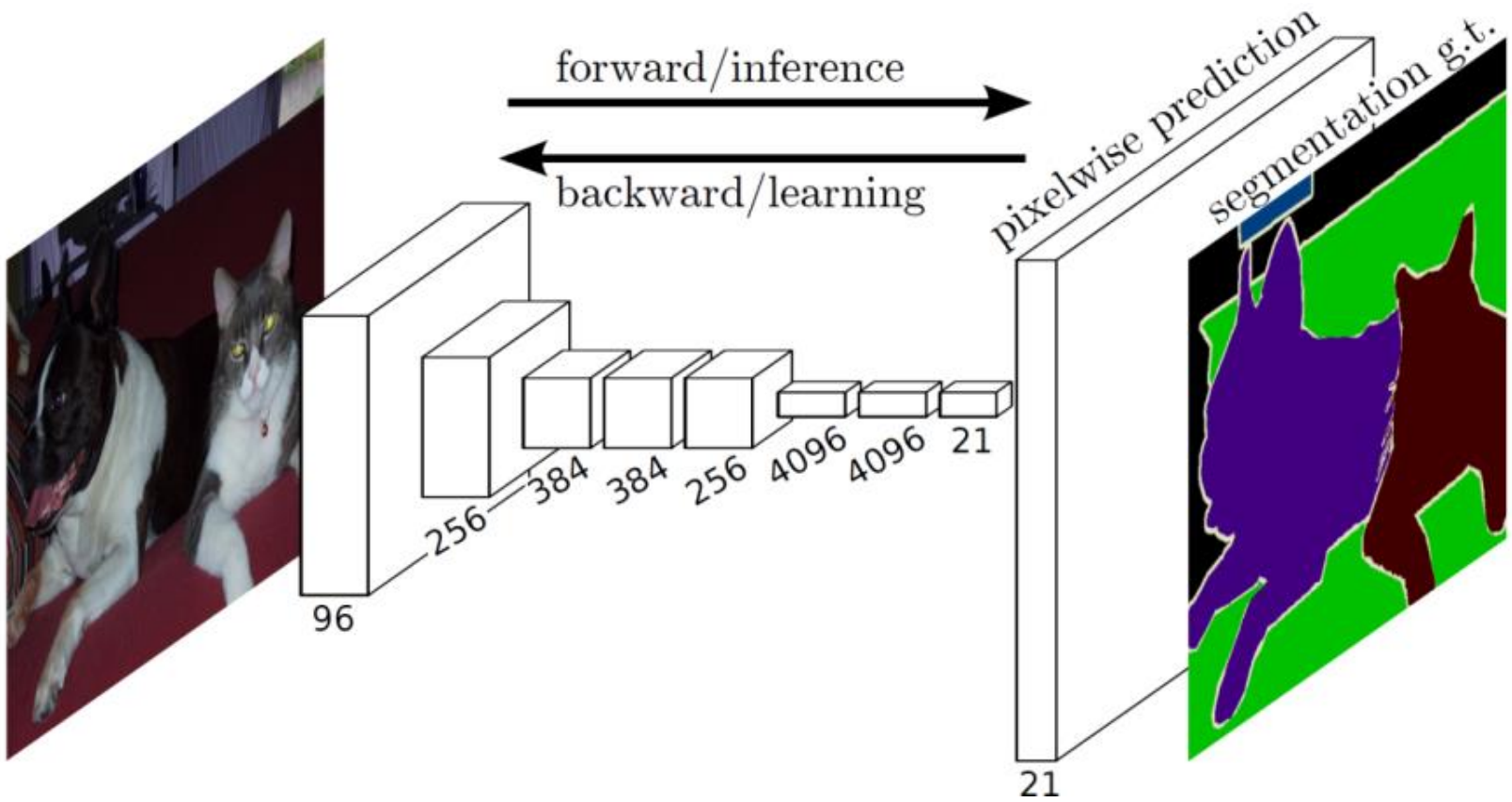
ResNet Layers



SEMANTIC SEGMENTATION AND OBJECT DETECTION

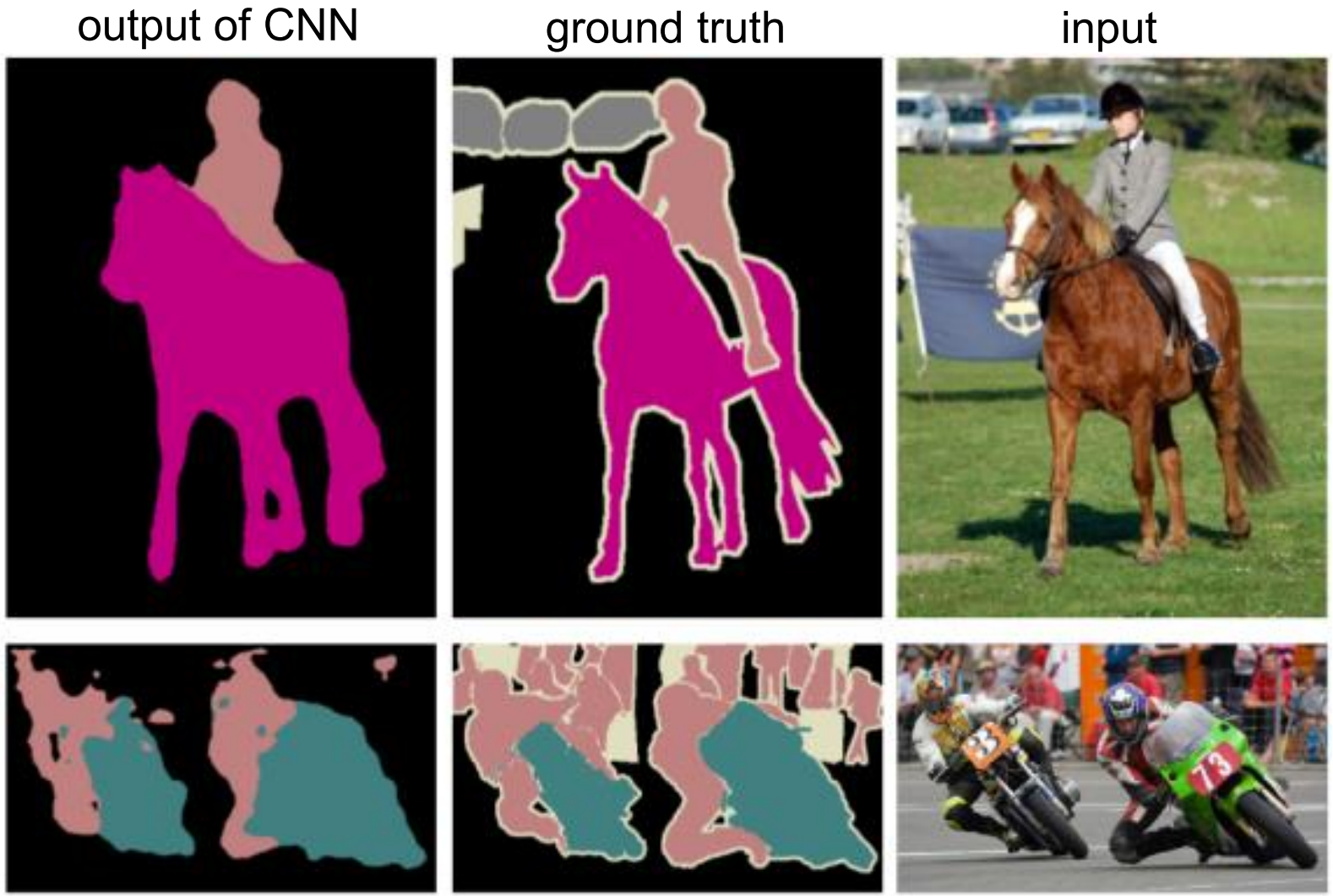
Scene Labeling

- segment the image
  - classify every pixel
  - autoencoder/decoder structure



taken from: J. Long, E. Shelhamer, T. Darrell, „Fully Convolutional Networks for Semantic Segmentation“, CVPR, 2015

Scene Labeling



taken from: J. Long, E. Shelhamer, T. Darrell, „Fully Convolutional Networks for Semantic Segmentation“, CVPR, 2015

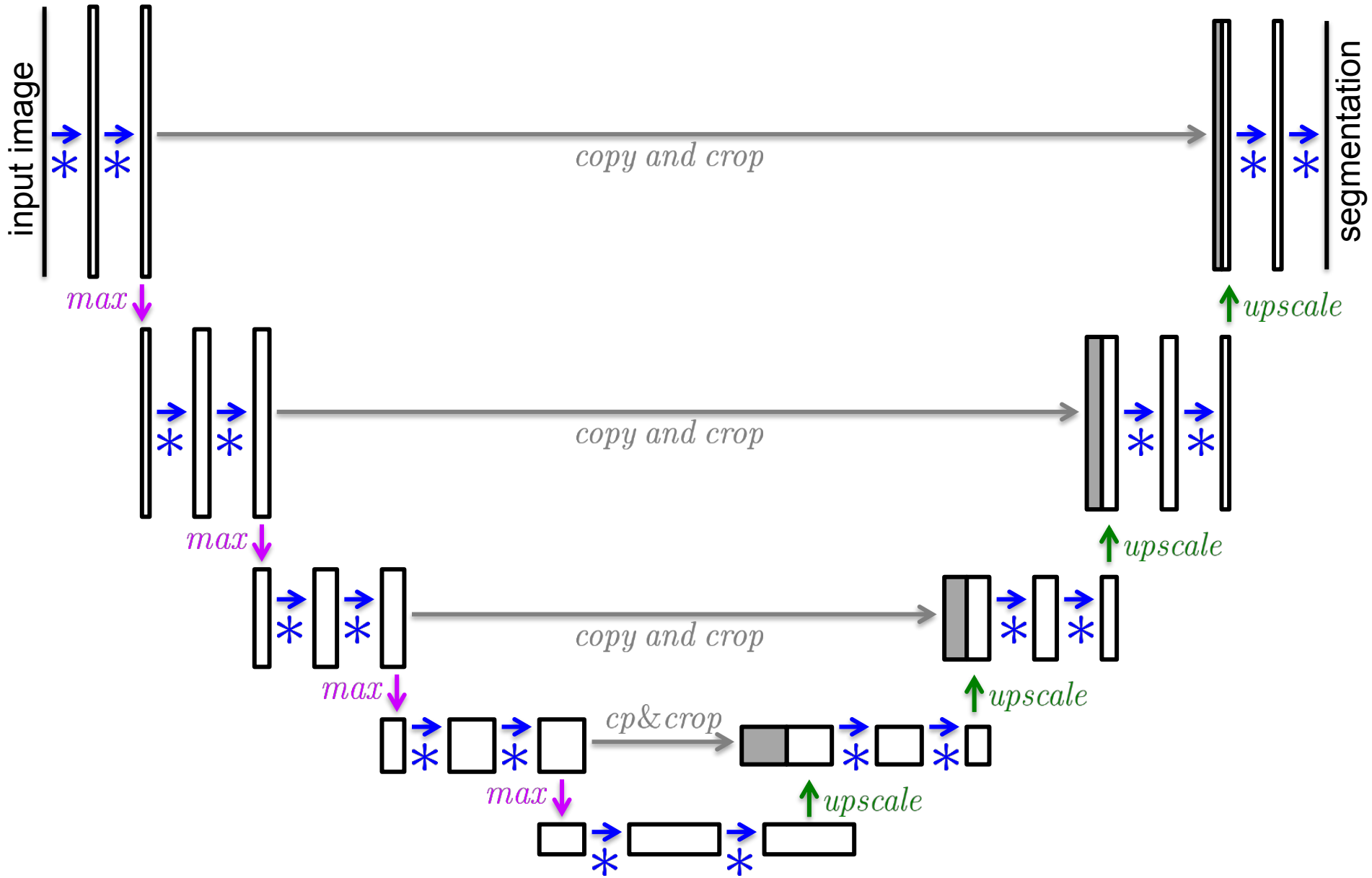
U-Net

Further improvements in U-Net:

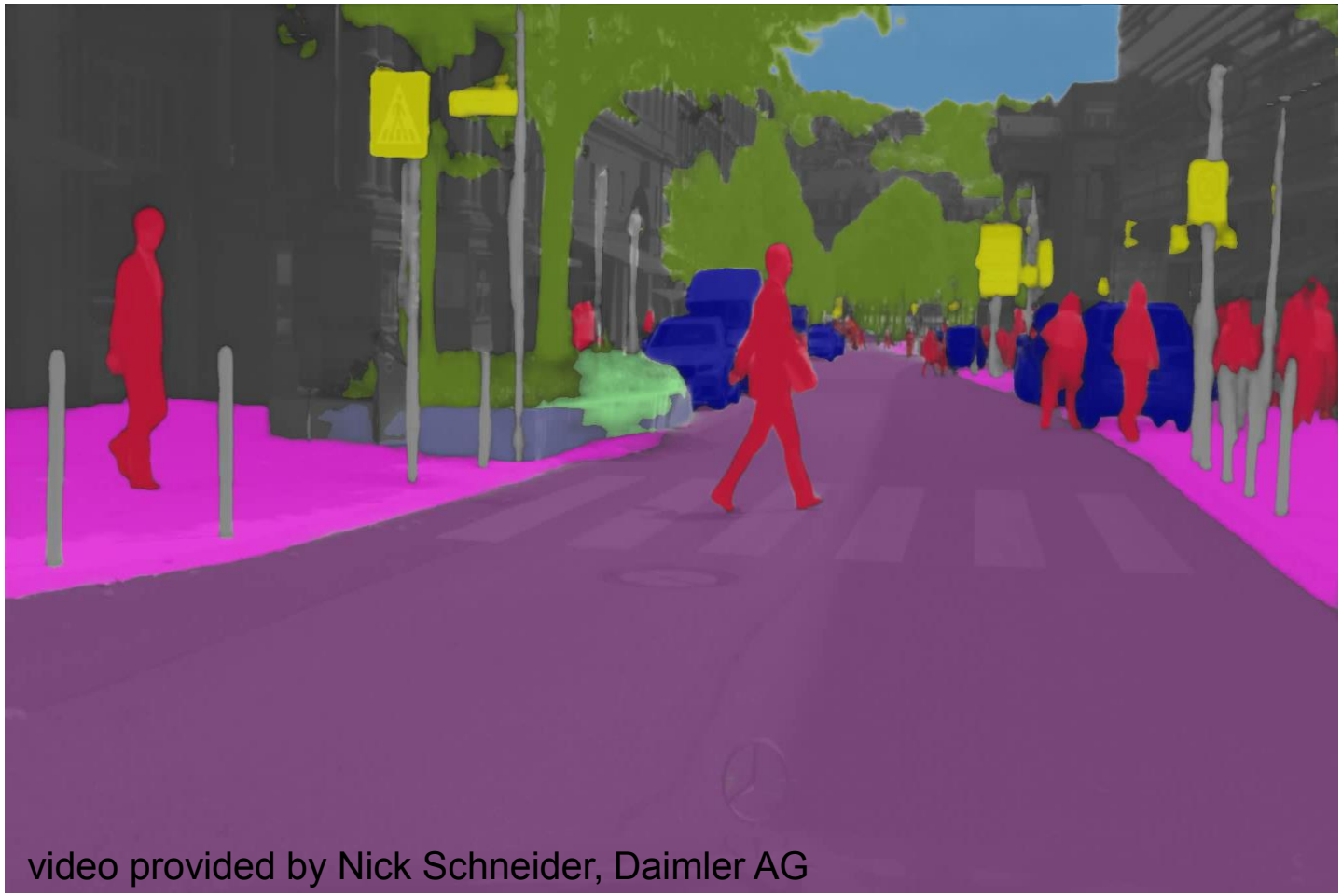
- stepwise upsampling
- skip connections that transfer knowledge on each compression layer from encoder to decoder side

O. Ronneberger, P. Fischer, T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In: MICCAI, 2015

U-Net


input image
segmentation
copy and crop
copy and crop
copy and crop
cp&crop
max
max
max
max
upscale
upscale
upscale
upscale

Scene Labeling



best performance on *Cityscapes* dataset > 80% accuracy

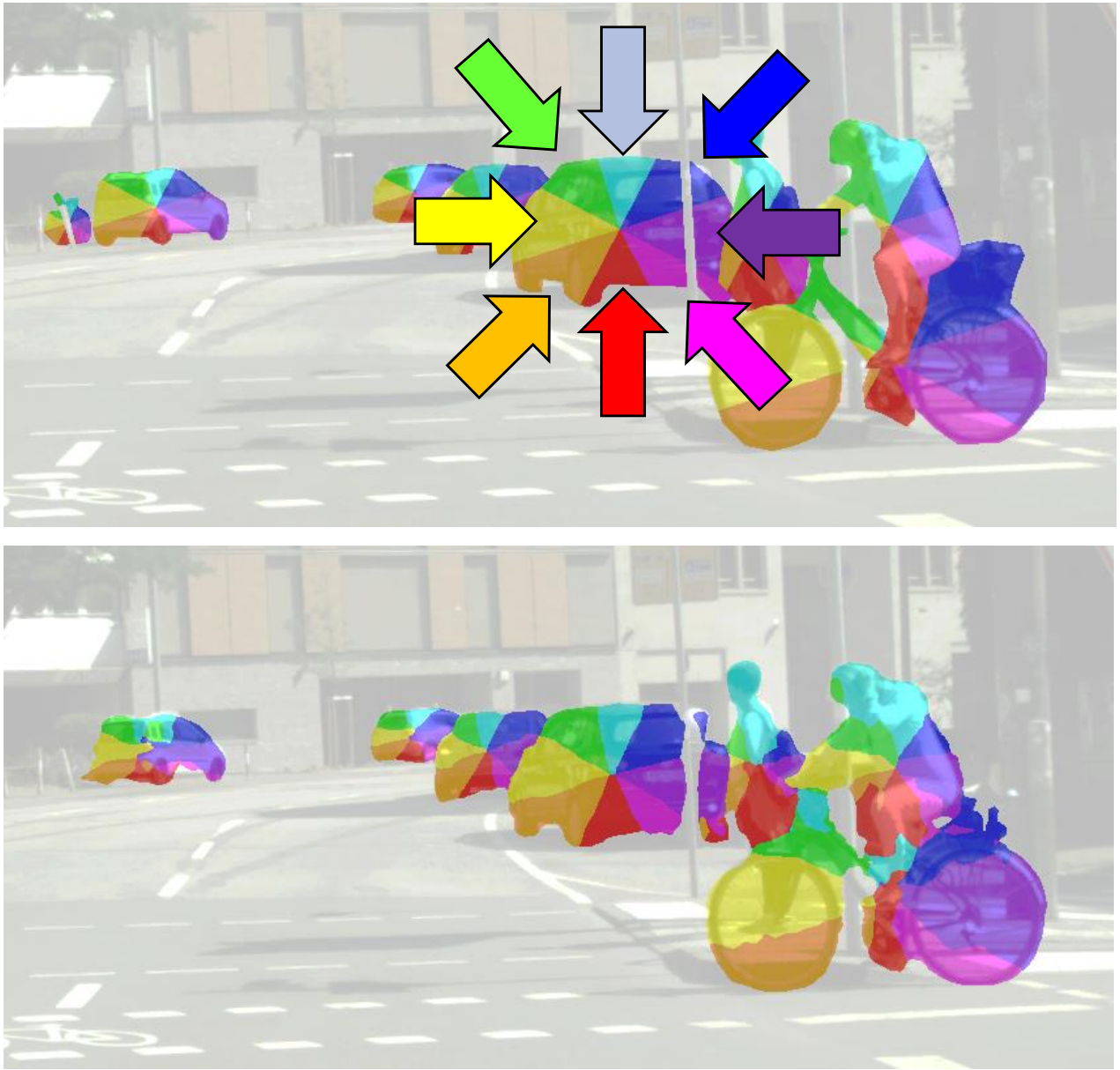
Instance Labeling

semantic labels do not provide object boundaries!

taken from: J. Uhrig, M. Cordts, U. Franke, T. Brox, Pixel-level encoding and depth layering for instance-level semantic segmentation, Germ. Conf. on Pattern Recognition, 2016/ provided by Nick Schneider, Daimler AG

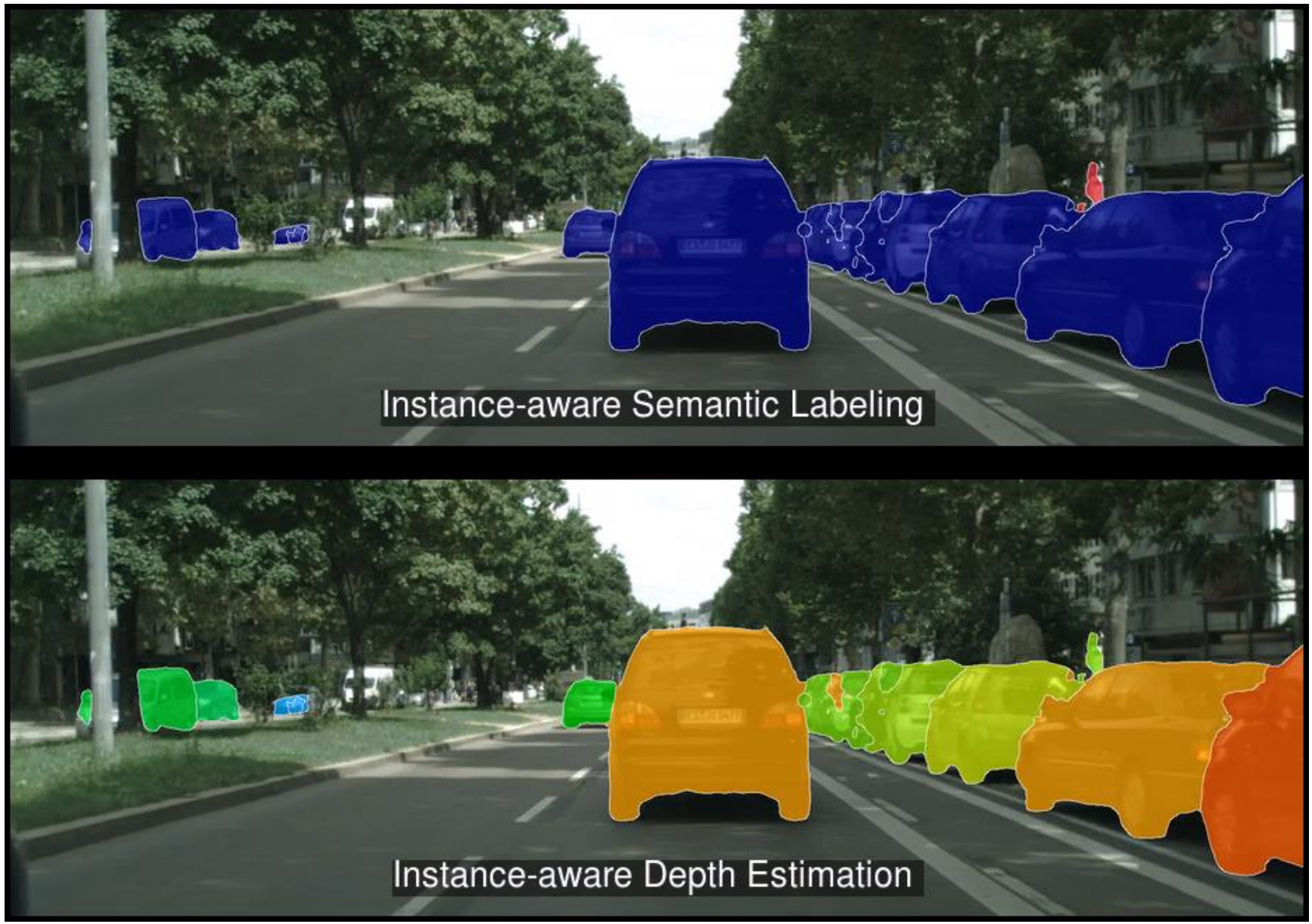
Instance Labeling

idea:
label direction
to center of object



taken from: J. Uhrig, M. Cordts, U. Franke, T. Brox, Pixel-level encoding and depth layering for instance-level semantic segmentation, Germ. Conf. on Pattern Recognition, 2016/ provided by Nick Schneider, Daimler AG

Instance Segmentation

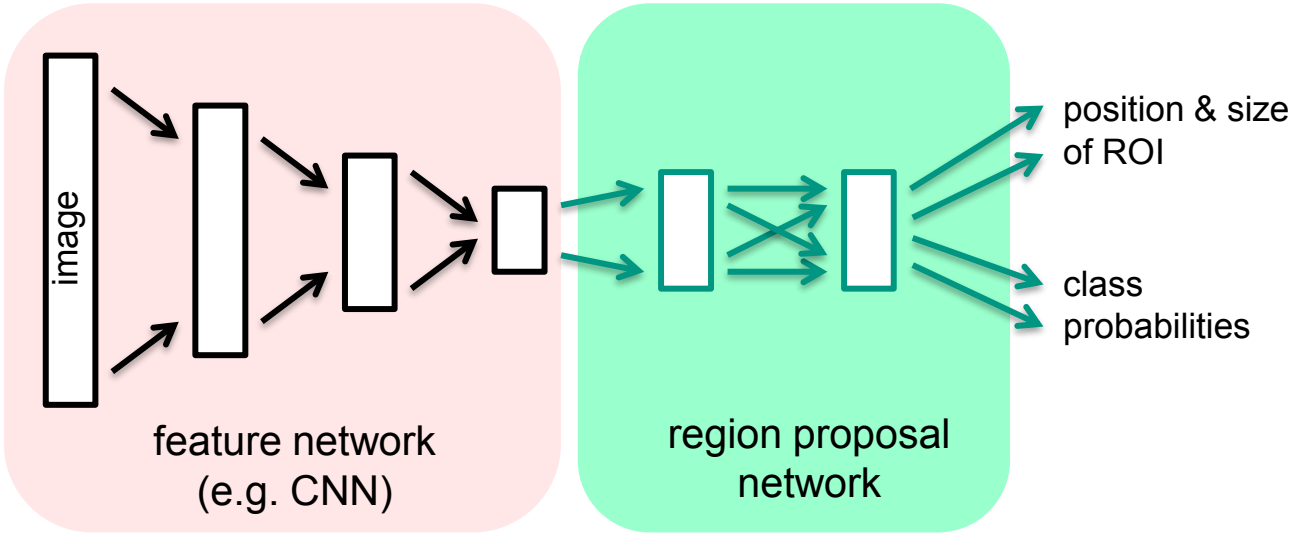


video provided by Nick Schneider, Daimler AG

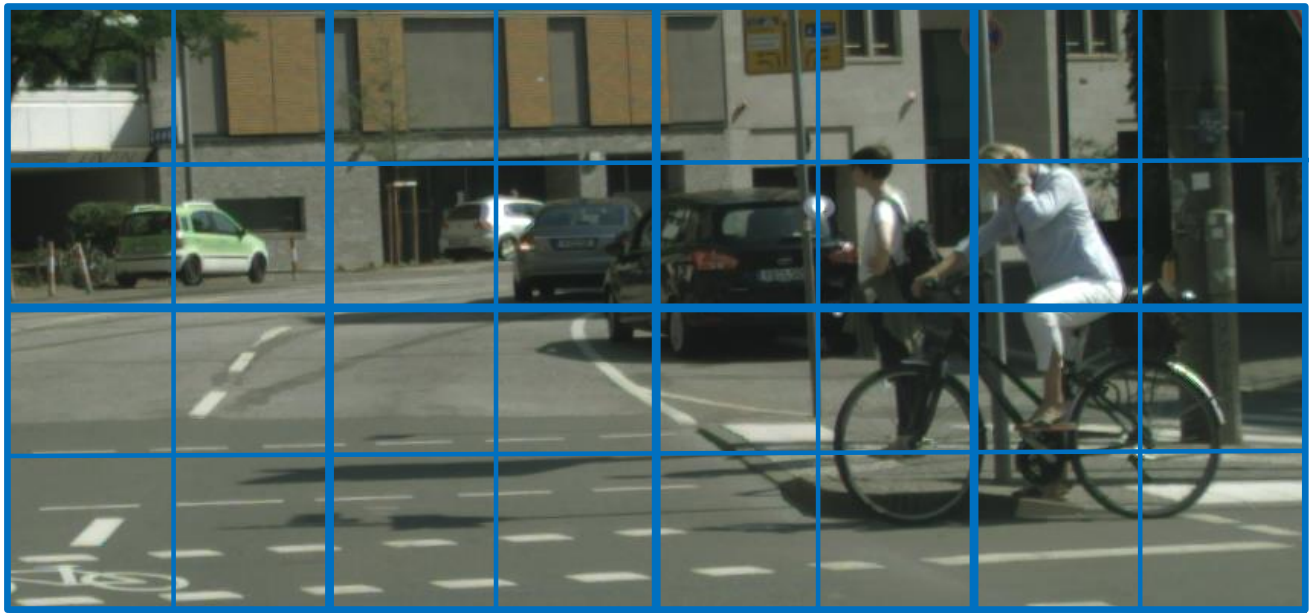
Region Proposal Networks

car: 98%
sign: 1%
vegetation: 1%
…

Does this image contain an object?
- where, which size?
- which category?



Region Proposal Networks



Where do we find which objects?

- partition image into cells
- apply region proposal network in each cell
- vary cell size of partitioning to cope with larger/smaller objects

Region Proposal Networks

work of Niels Ole Salscheider, MRT/KIT

Popular Architectures for Region Proposal Networks and Instance Segmentation

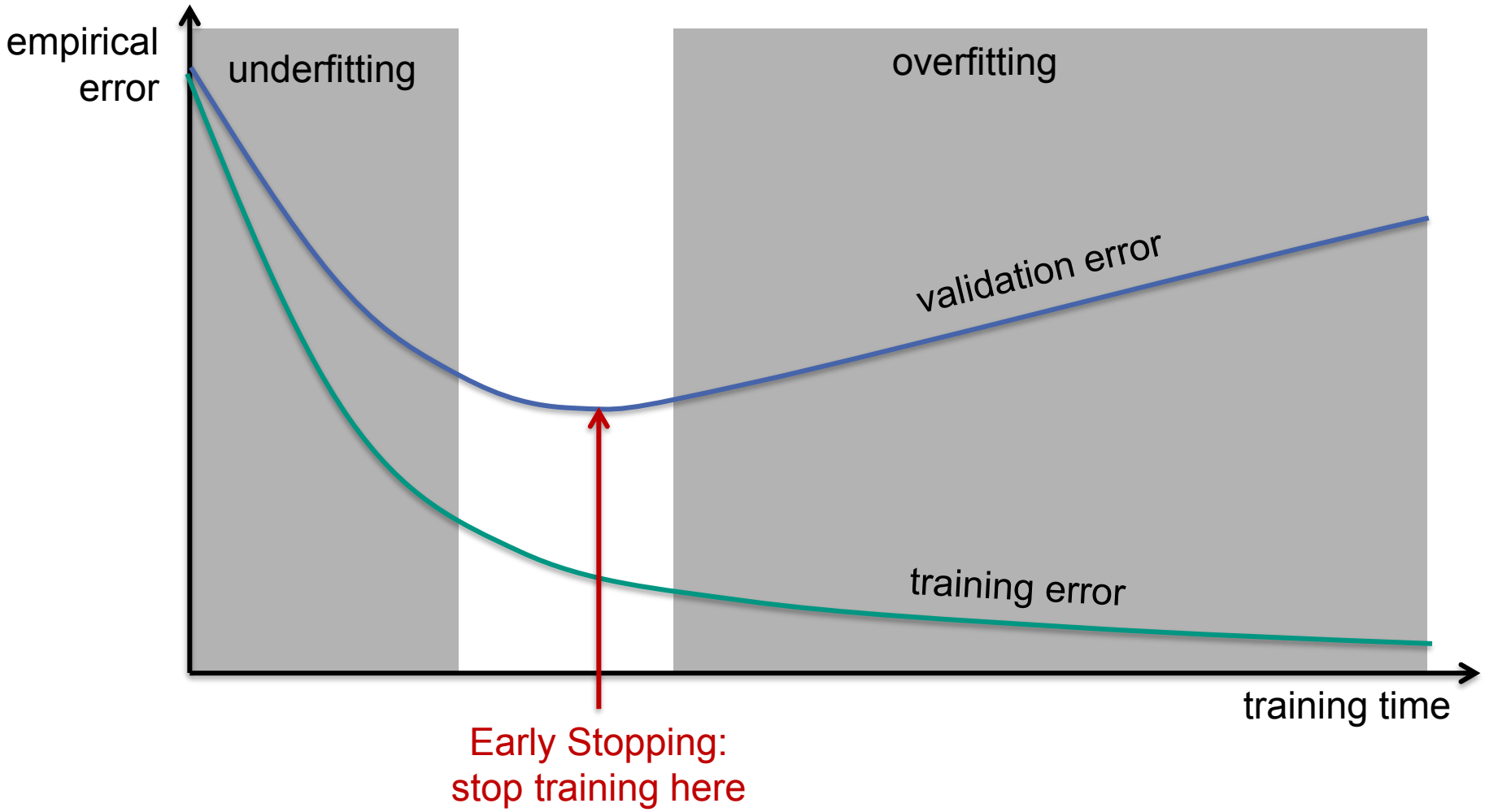
approach	provides	reference
Faster R-CNN	ROIs	S. Ren, K. He, R. Girshick, J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Netweorks"*. NIPS 2015
YOLO	ROIs	J. Redmon, S. Divvala, R. Girshick, A. Farhadi. *You Only Look Once: unified, Real-Time Object Detection*. CVPR 2016
SSD	ROIs	W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. Berg. *SSD: Single Shot MultiBox Detector.* ECCV 2016
(Daimler)	instance segmentation	J. Uhrig, M. Cordts, U. Franke, T. Brox. *Pixel-level encoding and depth layering for instance-level semantic segmentation*. GCPR 2016
Mask R-CNN	instance segmentation	K. He, G. Gkioxari, P. Dollár, R. Girshick. *Mask R-CNN.* ICCV 2017

TECHNIQUES FOR DEEP LEARNING

Principles for Training MLPs

- There‘s no data like more data!
  - remind slides 10/60-62 on data tuning
- rigorous validation of training process
- regularisation of training process
  - early stopping
  - weight decay/L2 regularisation
  - dropout
  - stochastic gradient descent
  - batch normalization
  - multi task learning
  - use pretrained networks
  - loss functions
- reuse of practical knowledge (of others)
  - successful network structures
  - successful training processes

Typical Progression of Error during Training



Why does early stopping work as regularisation technique?

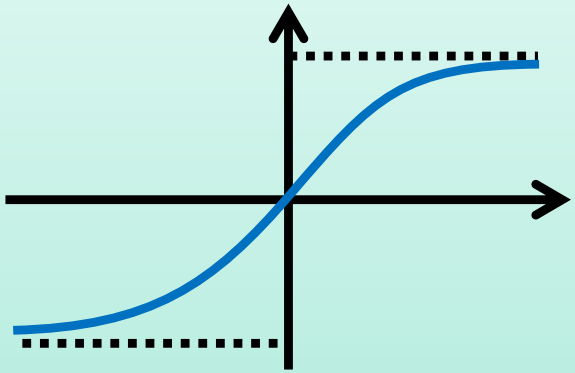
- early stopping prefers small weights
- small weights means little non-linearity (see next slide)

Regularisation by Small Weights

- assume perceptron with small absolute weights

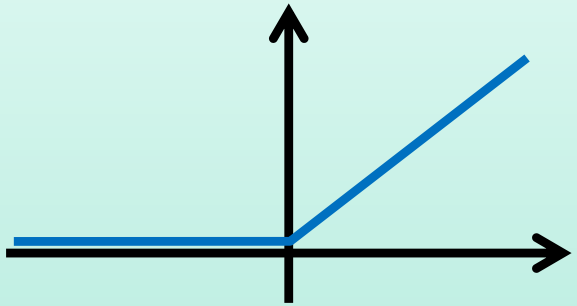
$$y = f_{act}\left(w_0 + \sum_{i=1}^{m} w_i x_i\right)$$

sigmoidal/hyperbolic tangent activation



$$y \approx w_0 + \sum_{i=1}^{m} w_i x_i$$

ReLU activation

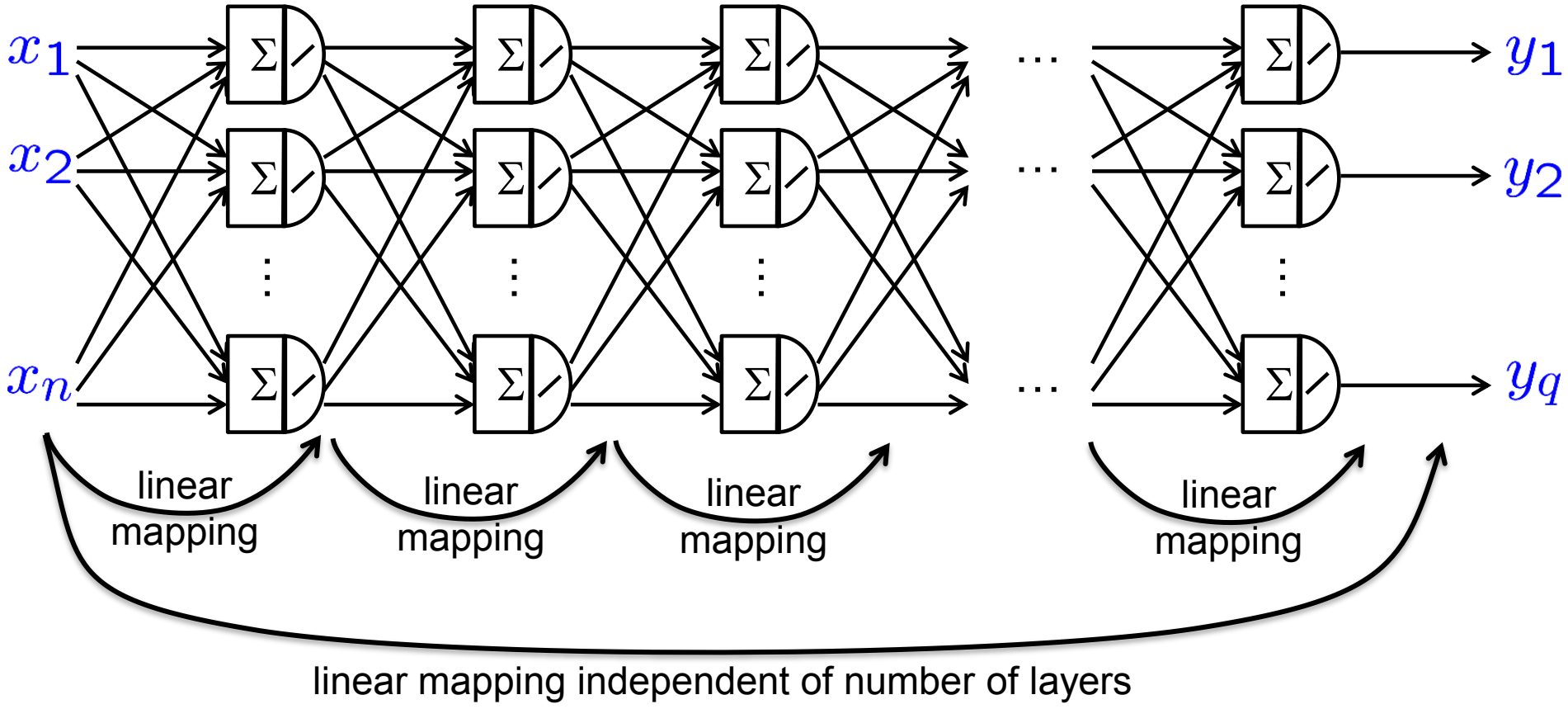


$$y \approx \begin{cases} 0 & \text{if } w_0 < 0 \\ w_0 + \sum_{i=1}^{m} w_i x_i & \text{if } w_0 > 0 \end{cases}$$

→ small weights foster linear behavior of perceptrons

Regularisation by Small Weights

- assume fully connected network with linear activation



→ linear behavior of perceptrons reduces non-linear expressiveness

→ regularisation

Weight Decay / L2-Regularisation

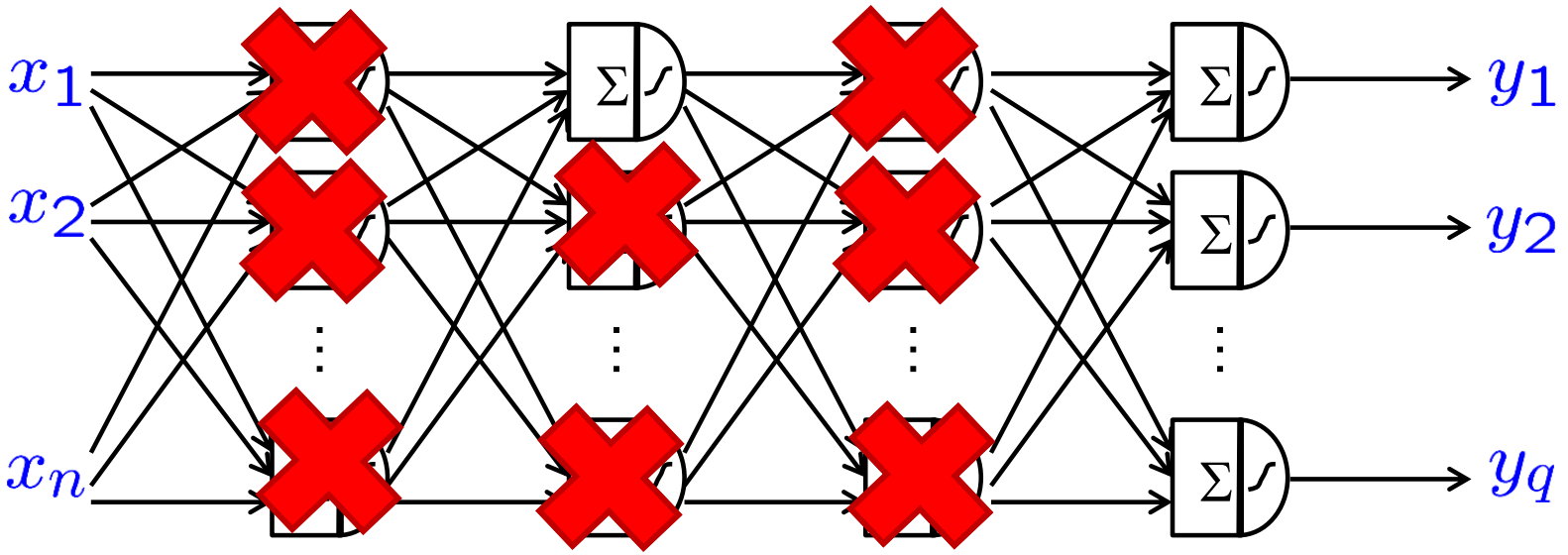
- extend goal of training by regulatisation term

$$\underset{\vec{w}}{minimize} \underbrace{\sum_{j=1}^{p} err\left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)}_{\text{1st goal: minimize error on training set}} + \underbrace{\lambda \cdot \sum_{\text{weights } i} w_i^2}_{\text{2nd goal: prefer small weights (regularisation)}}$$

$\lambda \geq 0$ is a parameter of the training algorithm that has to be chosen by trial and error

Dropout

- regularization by randomly switching off perceptrons during training



- dropout forces the neural network to store relevant information in a distributed way
- dropout reduces overfitting

Modifications of Gradient Descent

- stochastic gradient descent

$$\vec{w} \leftarrow \vec{w} - \varepsilon \cdot \frac{\partial}{\partial \vec{w}} \sum_{j=1}^{p} err\left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)$$

calculate gradient from all training examples

$$\vec{w} \leftarrow \vec{w} - \varepsilon \cdot \frac{\partial}{\partial \vec{w}} \sum_{j \in S} err\left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)$$

with $S \subseteq \{1, \ldots, p\}$

calculate gradient from subset of all training examples. Subsets typically cycle through all examples

advantages:

– speed up

– a little bit less overfitting

Modifications of Gradient Descent

- gradient descent with momentum

$$\Delta\vec{w} \leftarrow \alpha \cdot \Delta\vec{w} - \varepsilon \cdot \frac{\partial}{\partial\vec{w}} \sum_{j \in S} err\left(f^{\vec{w}}_{MLP}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)$$

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

with $\alpha \geq 0$ a parameter that controls consistency of subsequent steps

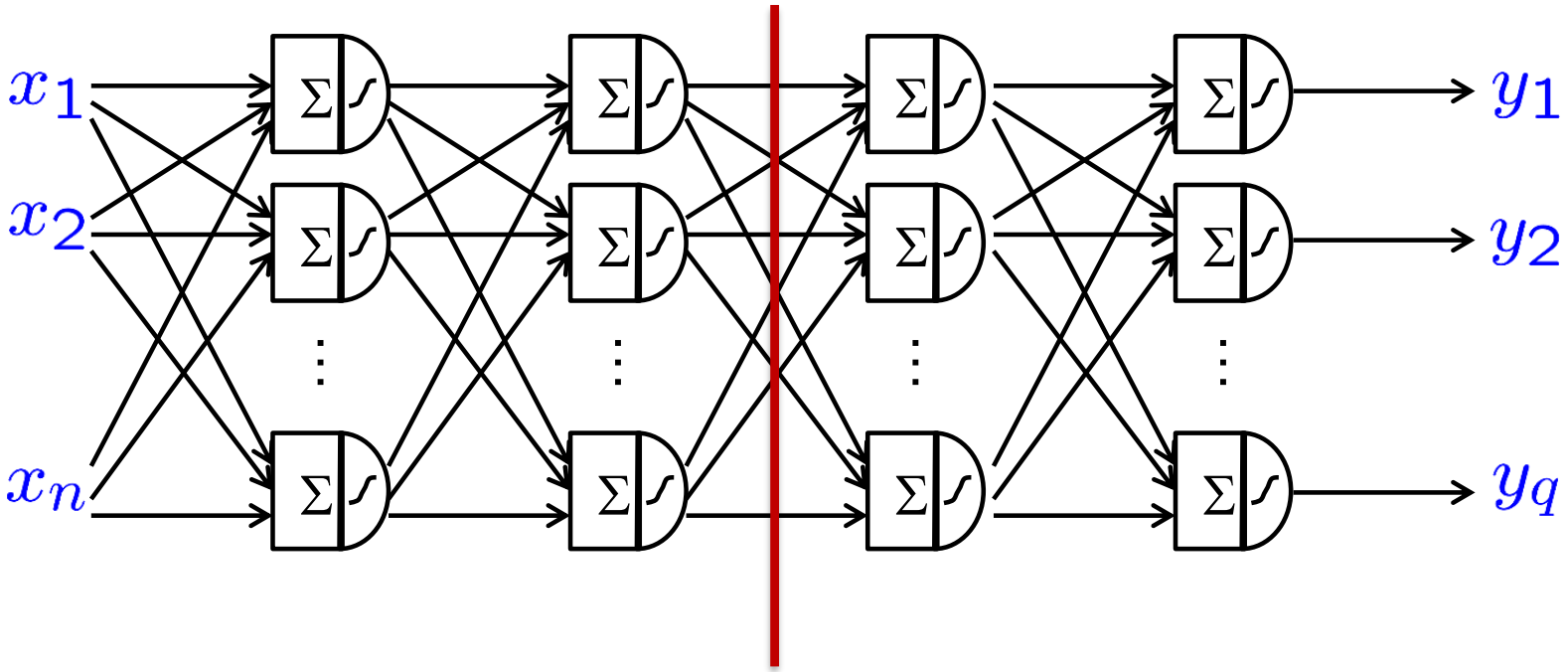
advantages:

– speed up in flat areas

– less zig zagging

Batch Normalization

- training for MLPs improves if training data are normalized (cf. slide 10-62)
- this applies also for each layer of an MLP

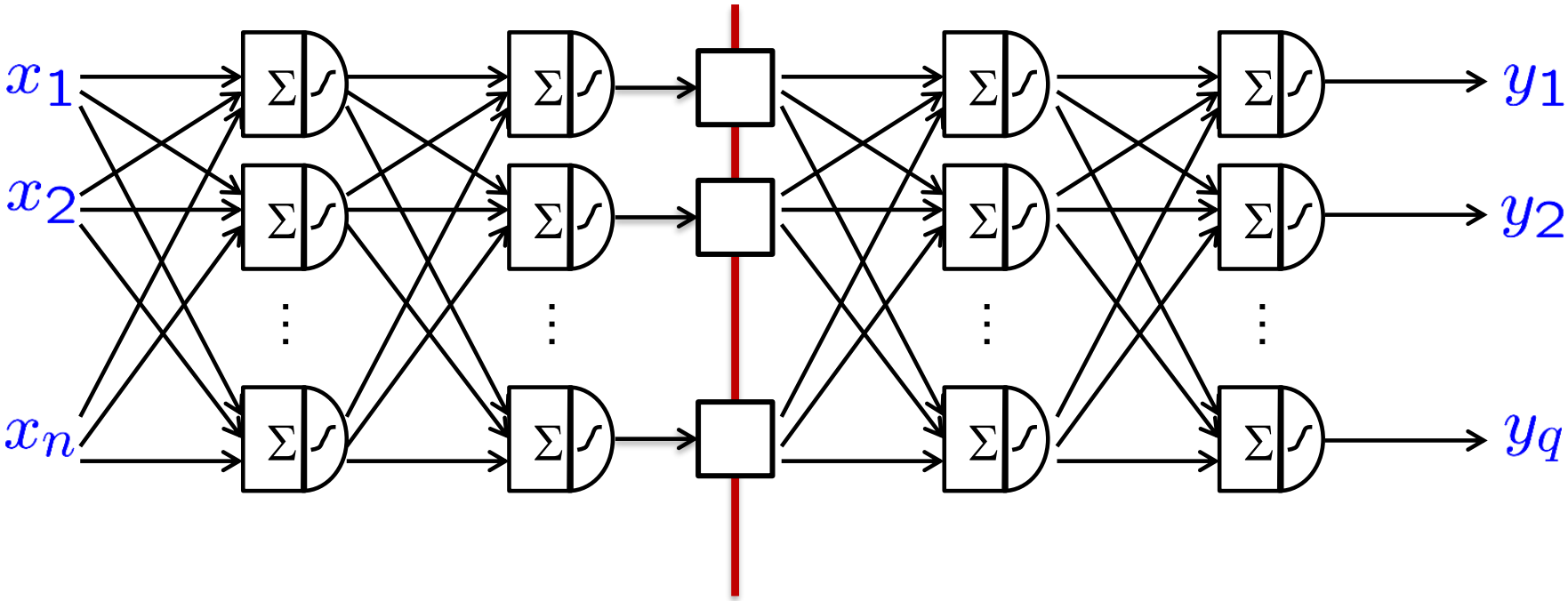


having normalized the input of a layer
improves training performance of the layer

- introduce batch normalization layers

S. Ioffe, C. Szegedy.Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: arXiv, 2015

Batch Normalization



average of inputs at this layer

$$y = \gamma \cdot \frac{x - \bar{x}}{s_x} + \beta$$

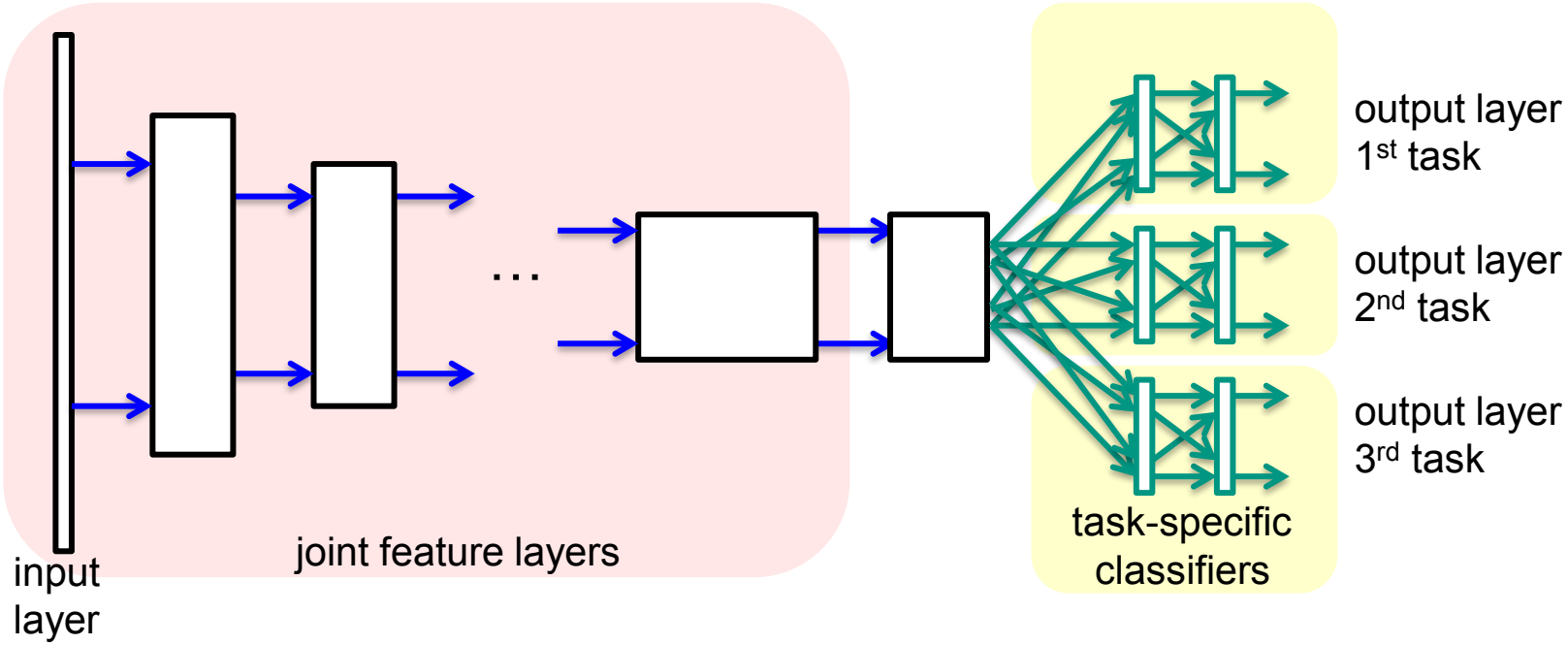
standard deviation of inputs at this layer

trainable parameters
(one for each normalization unit)

Multi Task Learning

- idea: learn several related tasks in a single network

  example: scene labeling + instance labeling + depth estimation



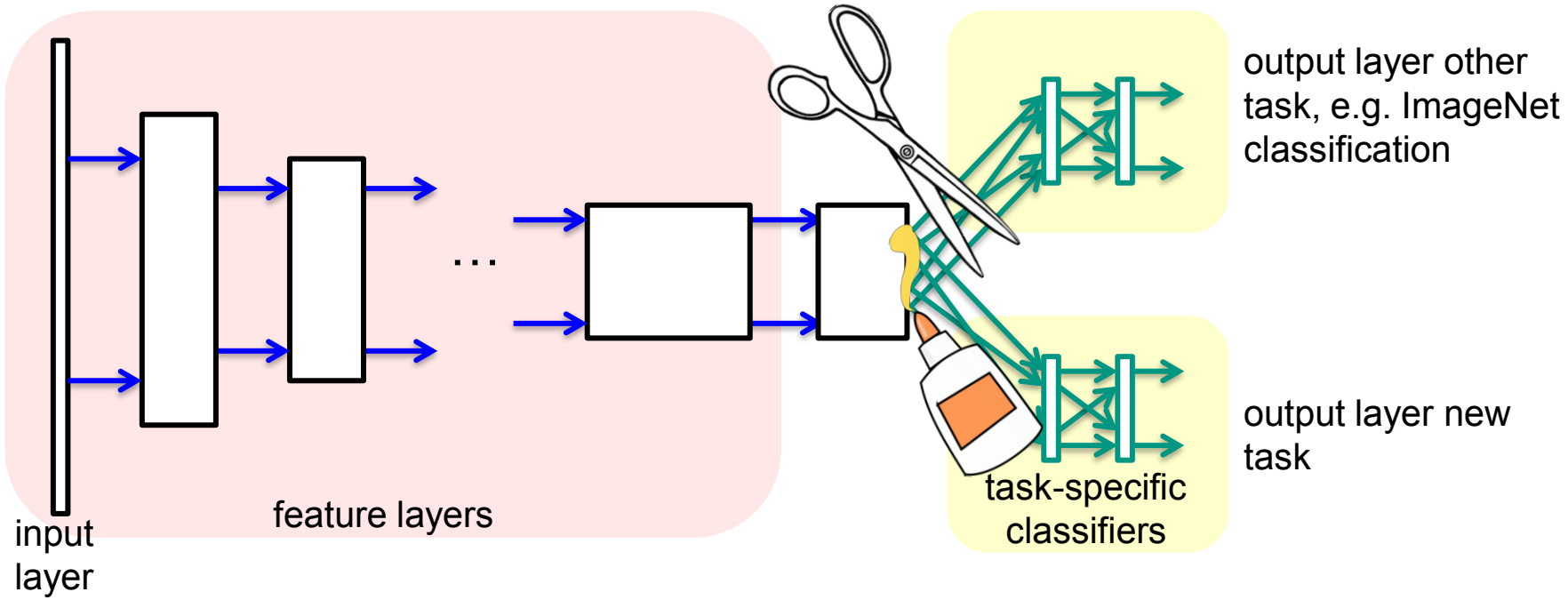
advantages:

– force network to develop common features in hidden layer

– reduce overfitting to a single task

Usage of Pre-Trained Feature Networks

- idea: reuse pre-trained network



1. train other task with large training set
2. throw away classification layers of other task
3. create new classification layers for new task
4. train weights of new classification layer while preserving feature layers

Output Layers and Loss Functions

- loss functions compare the network output with the desired output

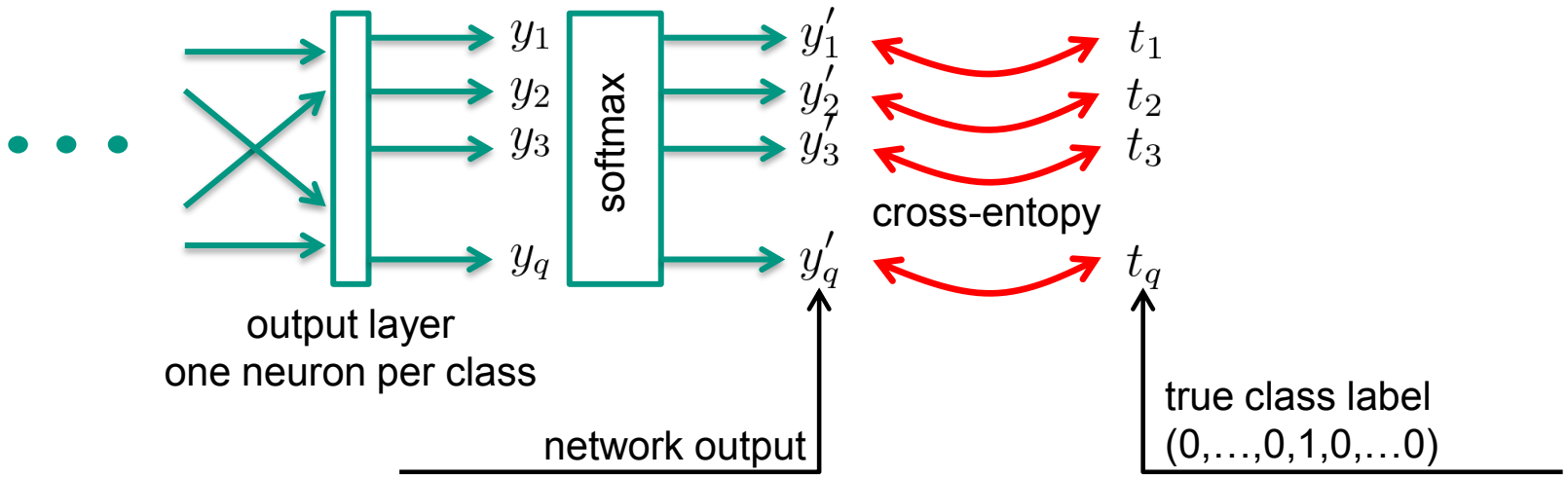
$$\underset{\vec{w}}{minimize} \sum_{j=1}^{p} err\left(f_{MLP}^{\vec{w}}(\vec{x}^{(j)}), \vec{t}^{(j)}\right)$$

1. case (regression task): desired output should be real number
   - use linear activation function in output layer
   - use squared error, i.e.

$$err(\vec{y}, \vec{t}) = ||\vec{y} - \vec{t}||^2$$

Output Layers and Loss Functions

2. case (classification task): desired output should be class label
   - use softmax activation function inoutput layer
   - use cross-entropy error



softmax operation:

$$y'_i = \frac{e^{y_i}}{\sum_{k=1}^{q} e^{y_k}}$$

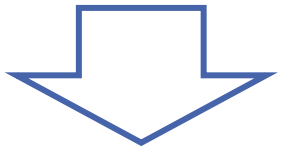
y'_i can be interpreted as class probabilities

cross entropy loss:

$$err(\vec{y'}, \vec{t}) = -\sum_{i=1}^{q} t_i \log y'_i$$

Variants for Imbalanced Training Sets

For heavily imbalanced training sets (i.e. not equal number of examples from each class) training might fail



introduce weighting factors to compensate imbalance

weighted cross entropy loss:

$$err(\vec{y}', \vec{t}) = -\sum_{i=1}^{q} \alpha_i t_i \log y'_i$$

$$\alpha_i > 0$$

one weight per class, e.g. reciprocal of number of examples

focal loss:

$$err\,(\vec{y}', \vec{t}) = -\sum_{i=1}^{q} \alpha_i |t_i - y'_i|^{\gamma_i} t_i \log y'_i$$

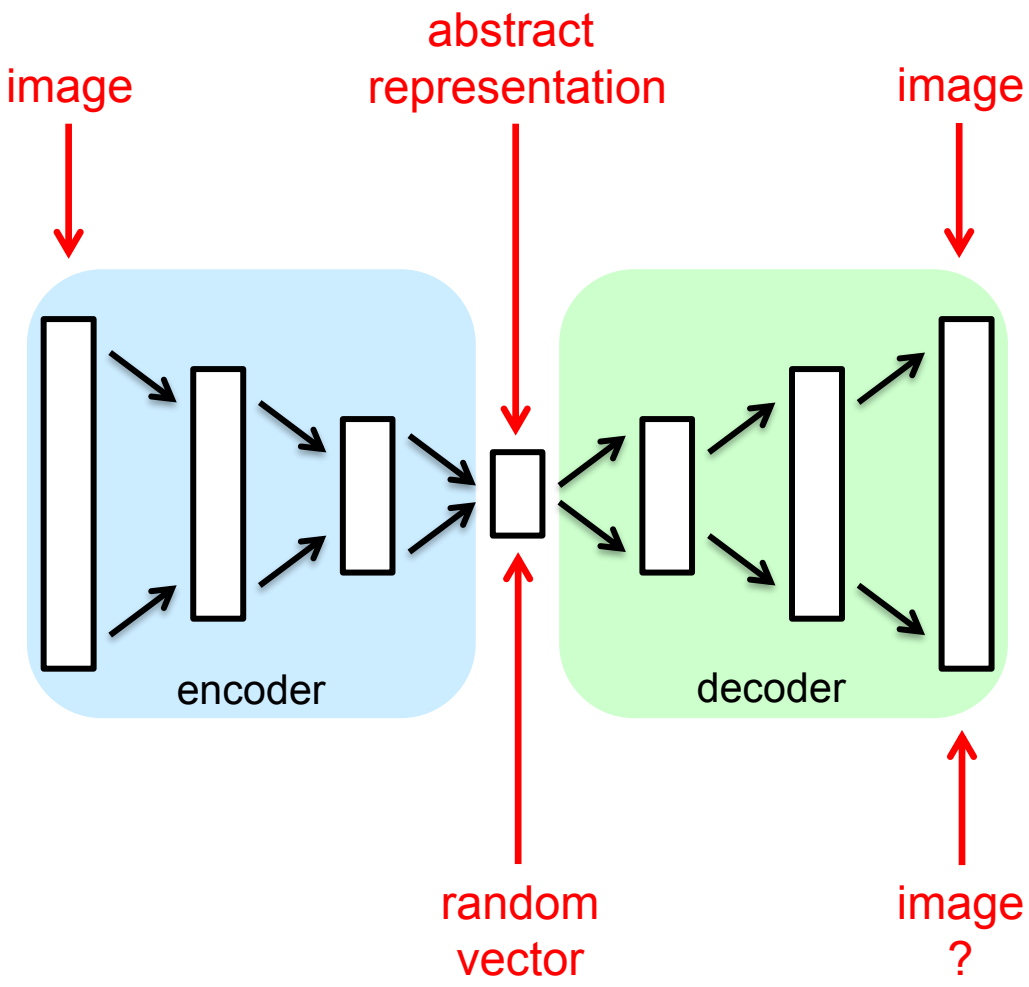
$$\gamma_i \geq 0$$

one parameter per class, e.g. 2; to focus on misclassified examples

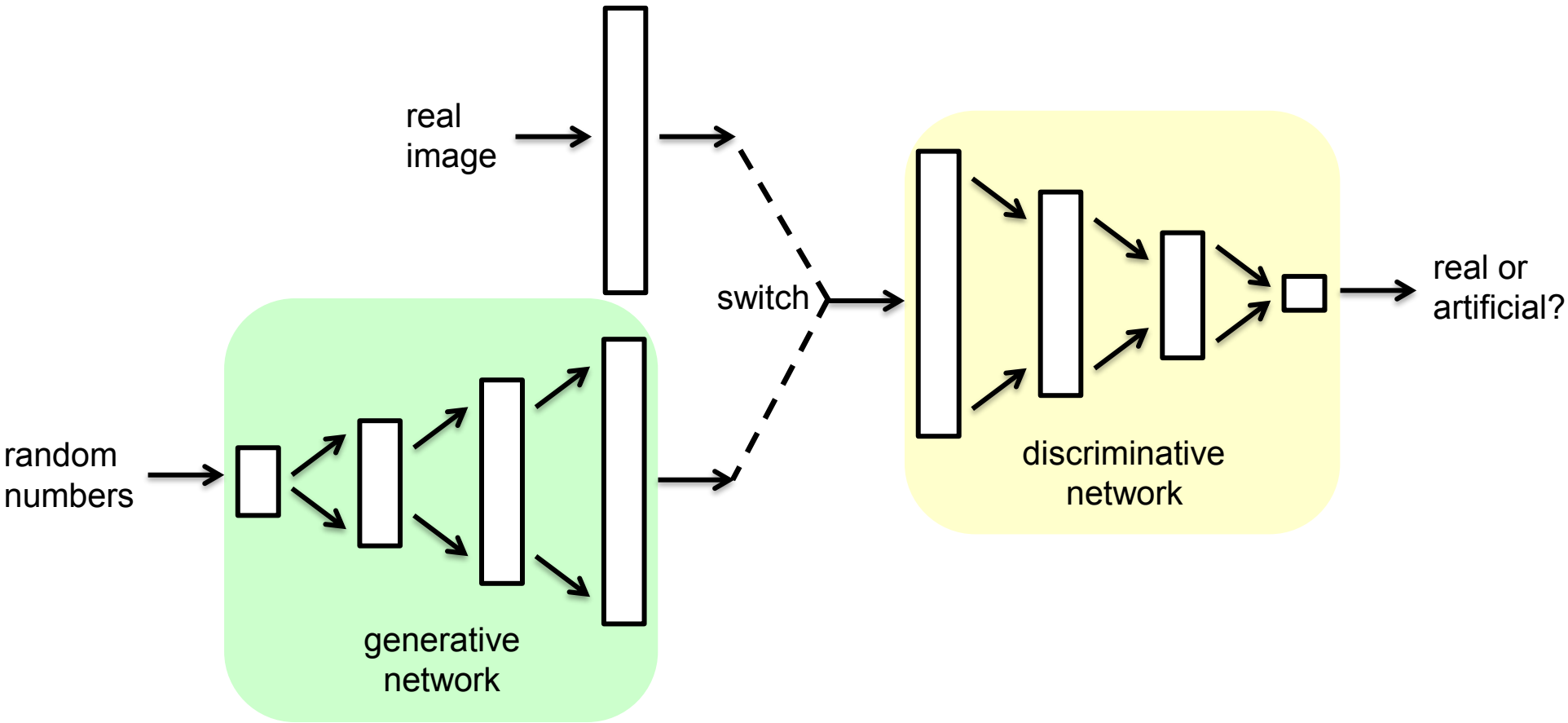
GENERATIVE NETWORKS

Generative Networks

- Can we use deep networks to generate realistic images?



Generative Adversarial Networks (GAN)



- Generative network should learn to generate photorealistic images
- Discriminative network should learn to clasify whether image is real or generated
- training: both networks are competitors in zero-sum game

Generative Adversarial Networks (GAN)

Application areas:

- image rendering
- domain adaptation
- generation of (additional) training data for classifiers

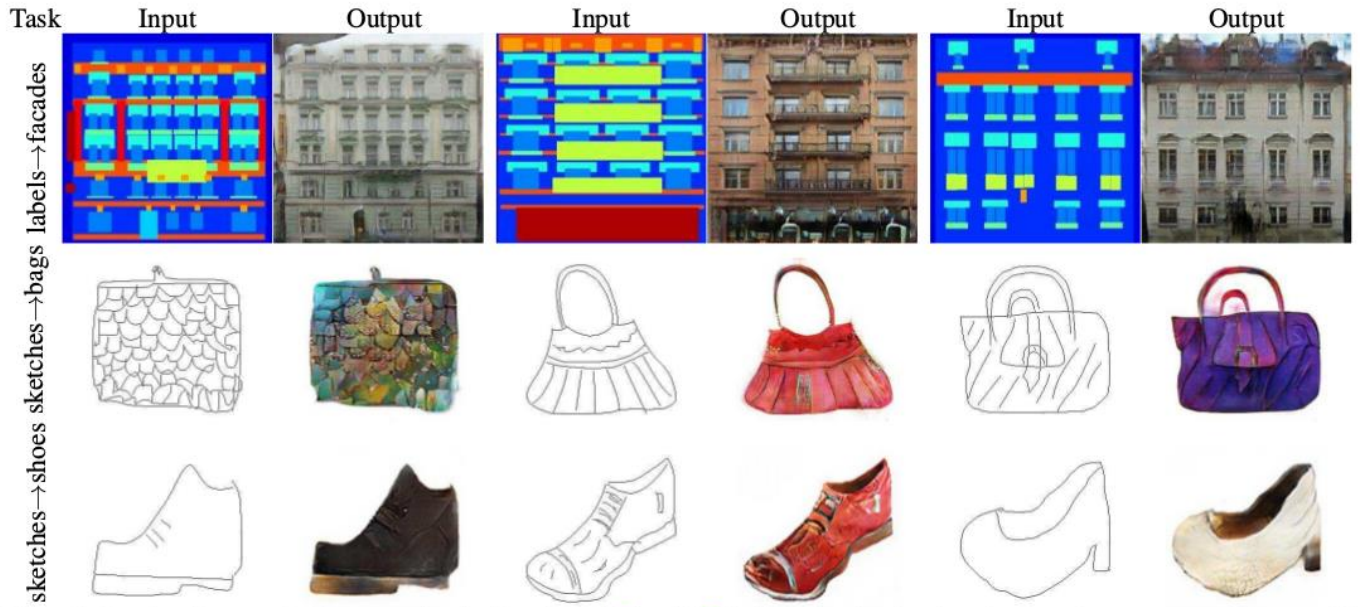


Figure 9: Results of our method on several tasks (data from [42] and [17]). Note that the sketch→photo results are generated by a model trained on automatic edge detections and tested on human-drawn sketches. Please see online materials for additional examples.

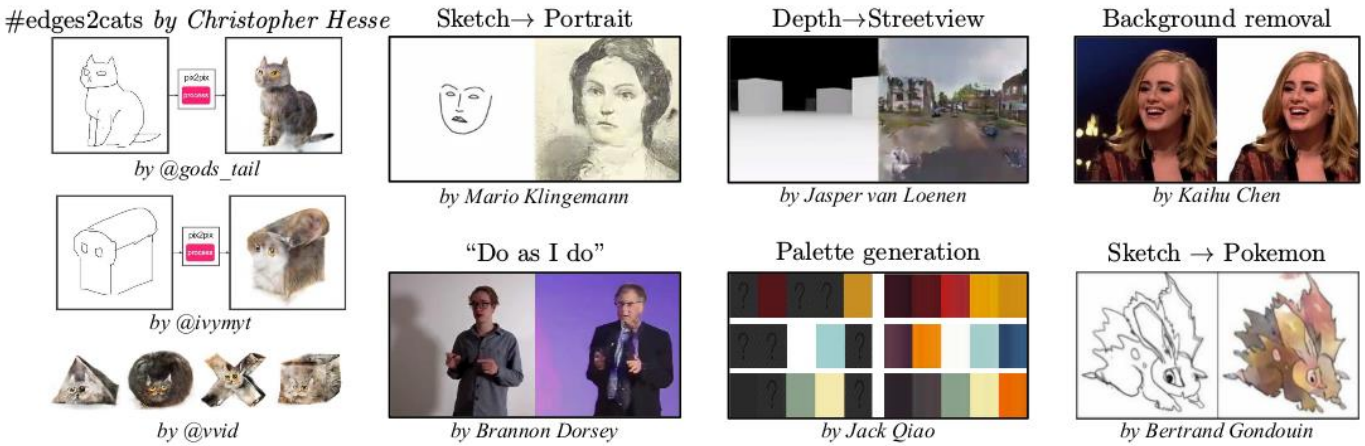


Figure 10: Example applications developed by online community based on our `pix2pix` codebase: *#edges2cats* [3] by Christopher Hesse, *Sketch → Portrait* [7] by Mario Kingemann, *"Do As I Do" pose transfer* [2] by Brannon Dorsey, *Depth→ Streetview* [5] by Jasper van Loenen, *Background removal* [6] by Kaihu Chen, *Palette generation* [4] by Jack Qiao, and *Sketch→ Pokemon* [1] by Bertrand Gondouin.

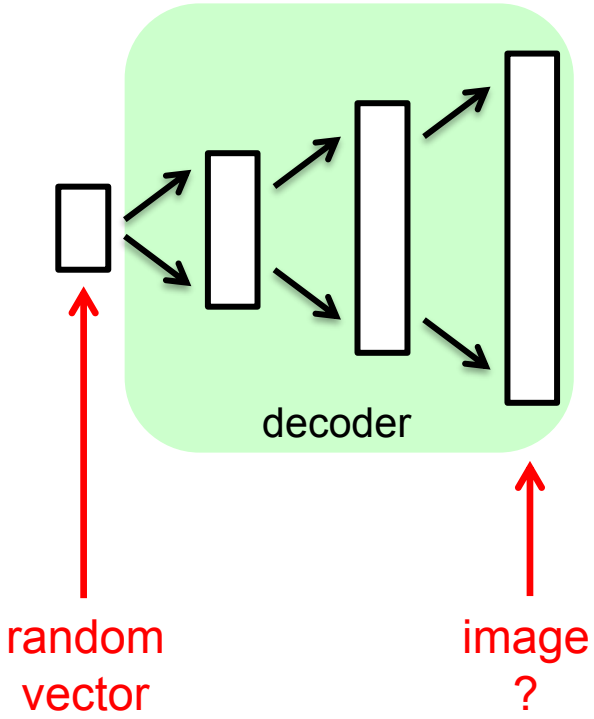
taken from: P. Isola, J.-Y. Zhu, T: Zhou, A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. CVPR 2017

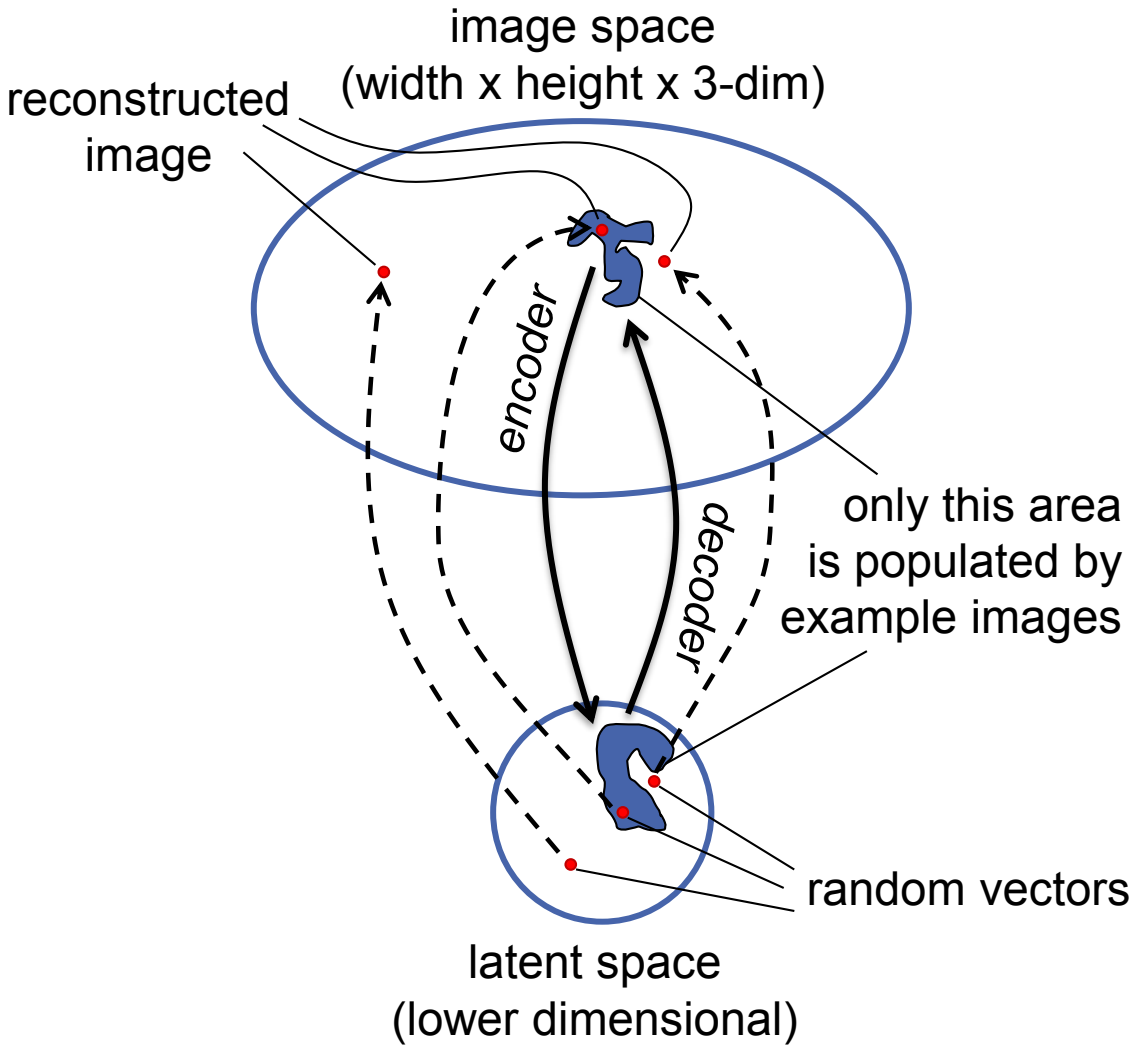
Generative Adversarial Networks (GAN)

provided by Jun-Yan Zhu on YouTube:
https://www.youtube.com/watch?v=9reHvktowLY

Generative Networks

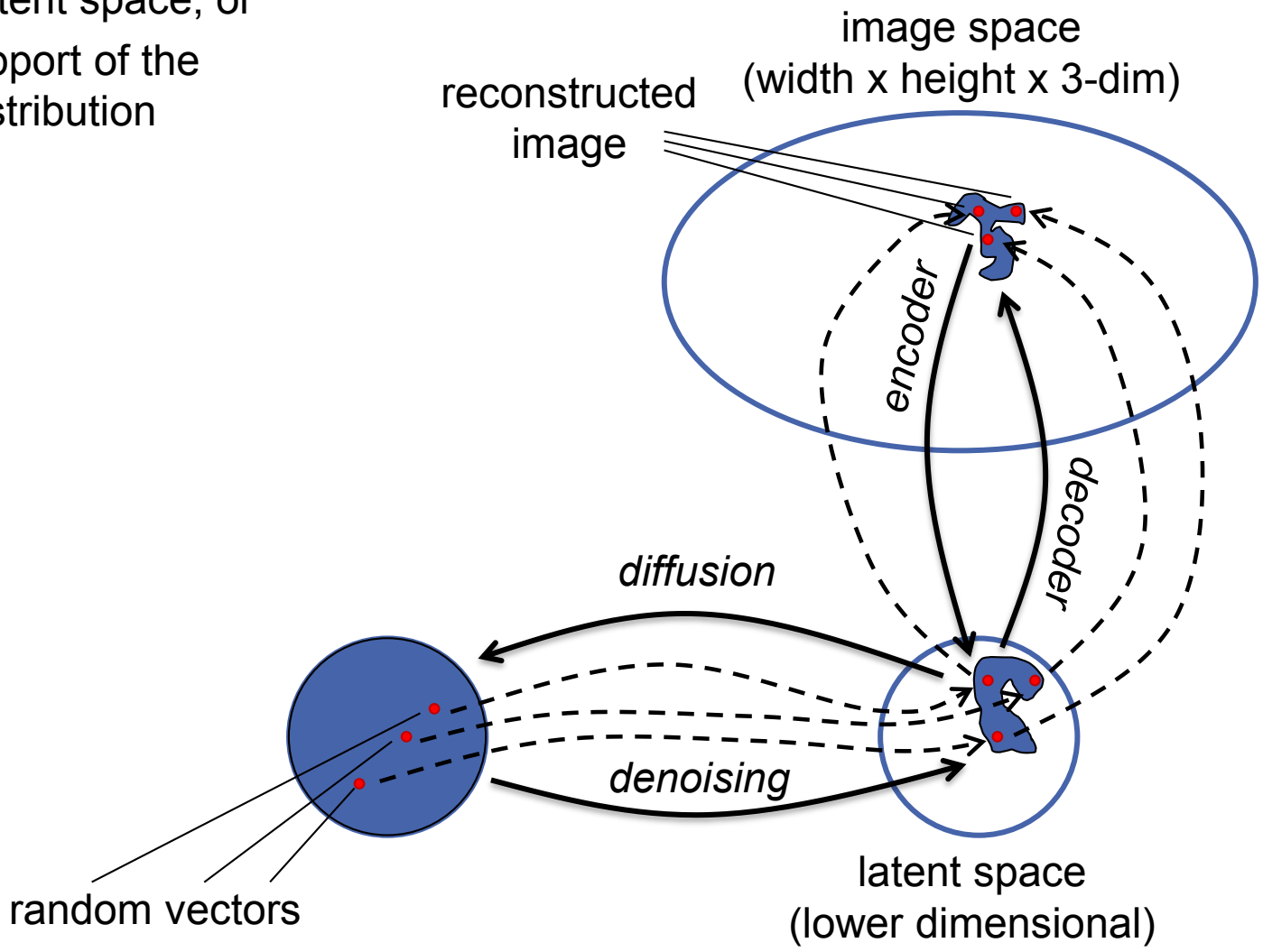
- Why doesn‘t this work?





Latent diffusion

- we need to force the encoder to fill
  - the whole latent space, or
  - to fill the support of the sampling distribution



Latent diffusion

- diffusion process
  - random process
  - incremental shift of vectors towards a target distribution
  - often, Gaussians are used as target distributions
  - physical analogon: Brownian motion
- denoising
  - predict the reverse drift with a deep network (e.g. U-net)
  - train the network by observing the diffusion process

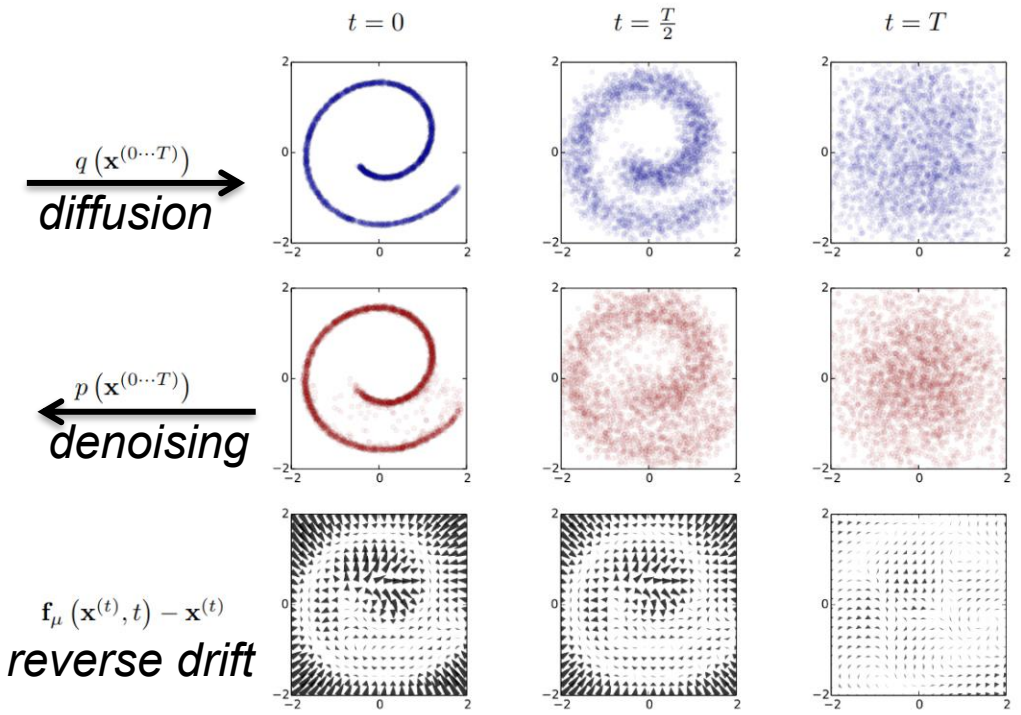
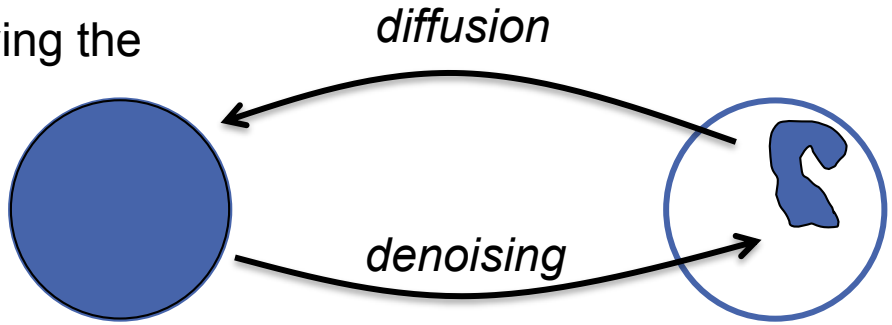
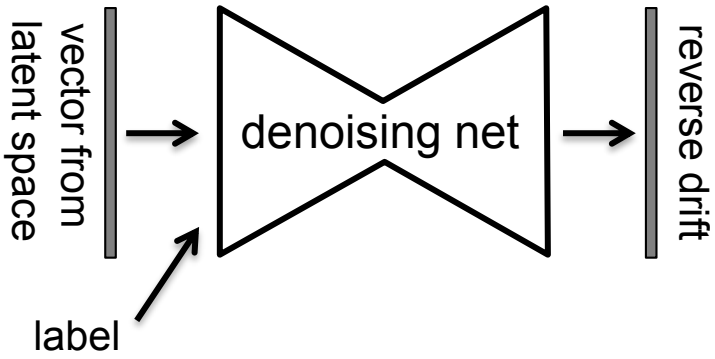


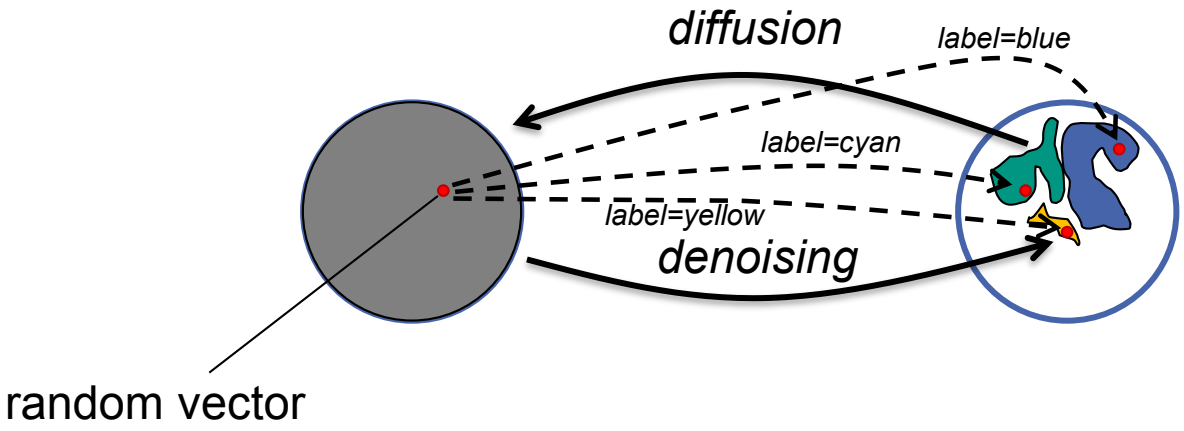
image taken from: J. Sohl-Dickstein et al. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. ICML 2015



Latent diffusion

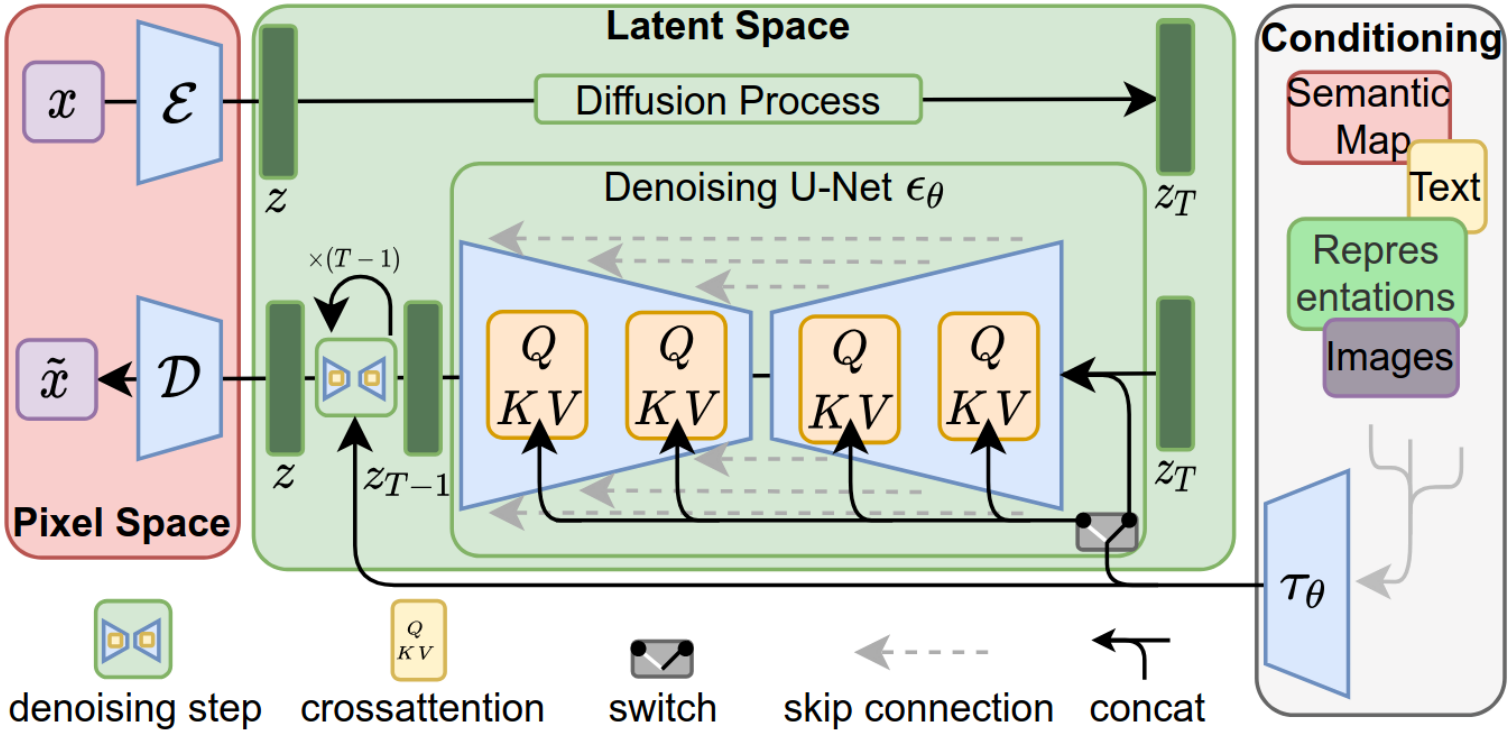
- conditional denoising
  - assume classification task
  - images of different classes are mapped to different areas in latent space
  - however, diffusion process uses the same target distribution for all images → loss of class information
  - add class label as additional input to the denoising network





Latent Diffusion

- generalization from to any kind of conditioning input possible, e.g.
  - textual descriptions
  - scene or image layout



taken from: R. Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* CVPR 2022

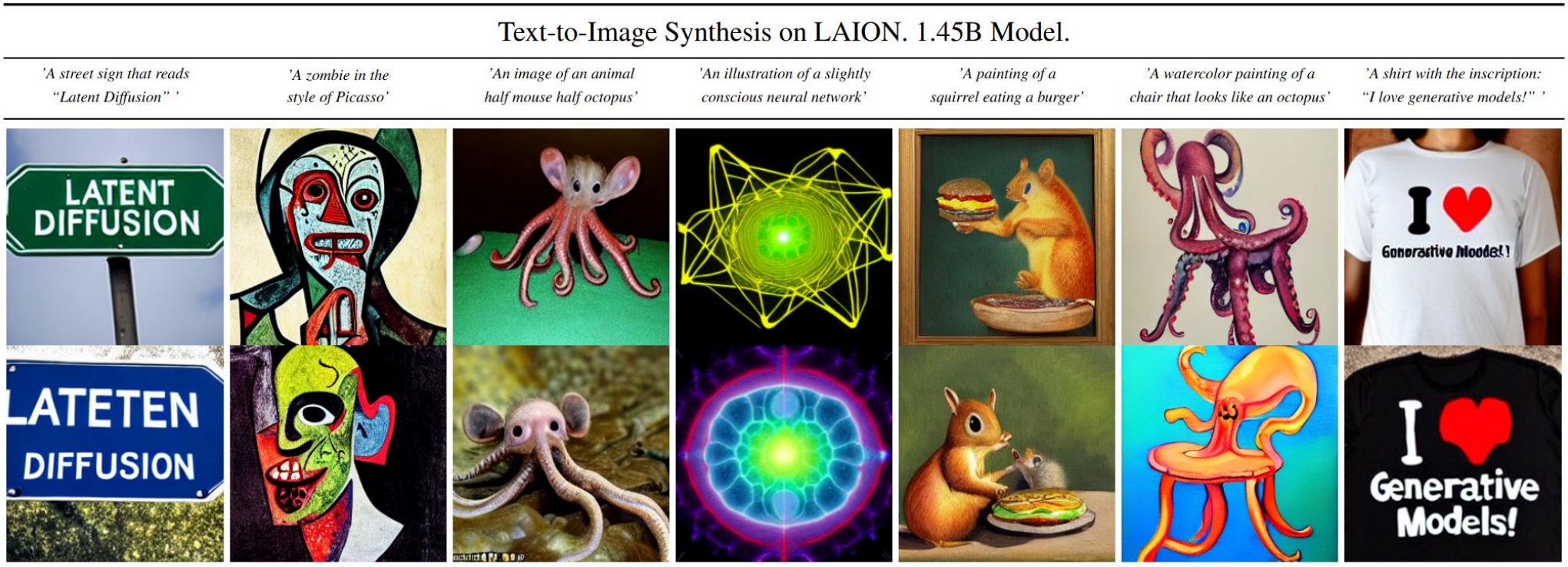
Latent Diffusion

- examples of synthesized images for various domains

taken from: R. Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* CVPR 2022

Latent Diffusion

- examples of synthesized images from textual input



taken from: R. Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* CVPR 2022

Latent Diffusion

- examples of synthesized images from image layout

taken from: R. Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* CVPR 2022

Latent Diffusion

- examples of synthesized images with removed objects

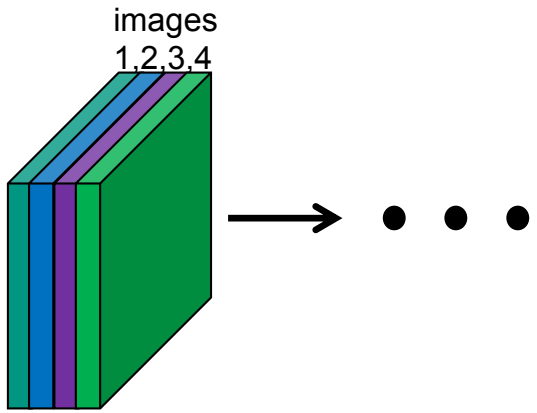
taken from: R. Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* CVPR 2022

SEQUENCE PROCESSING

Sequence Processing

- How can we process sequences, e.g. video sequences?

  (a) multi-channel input layer

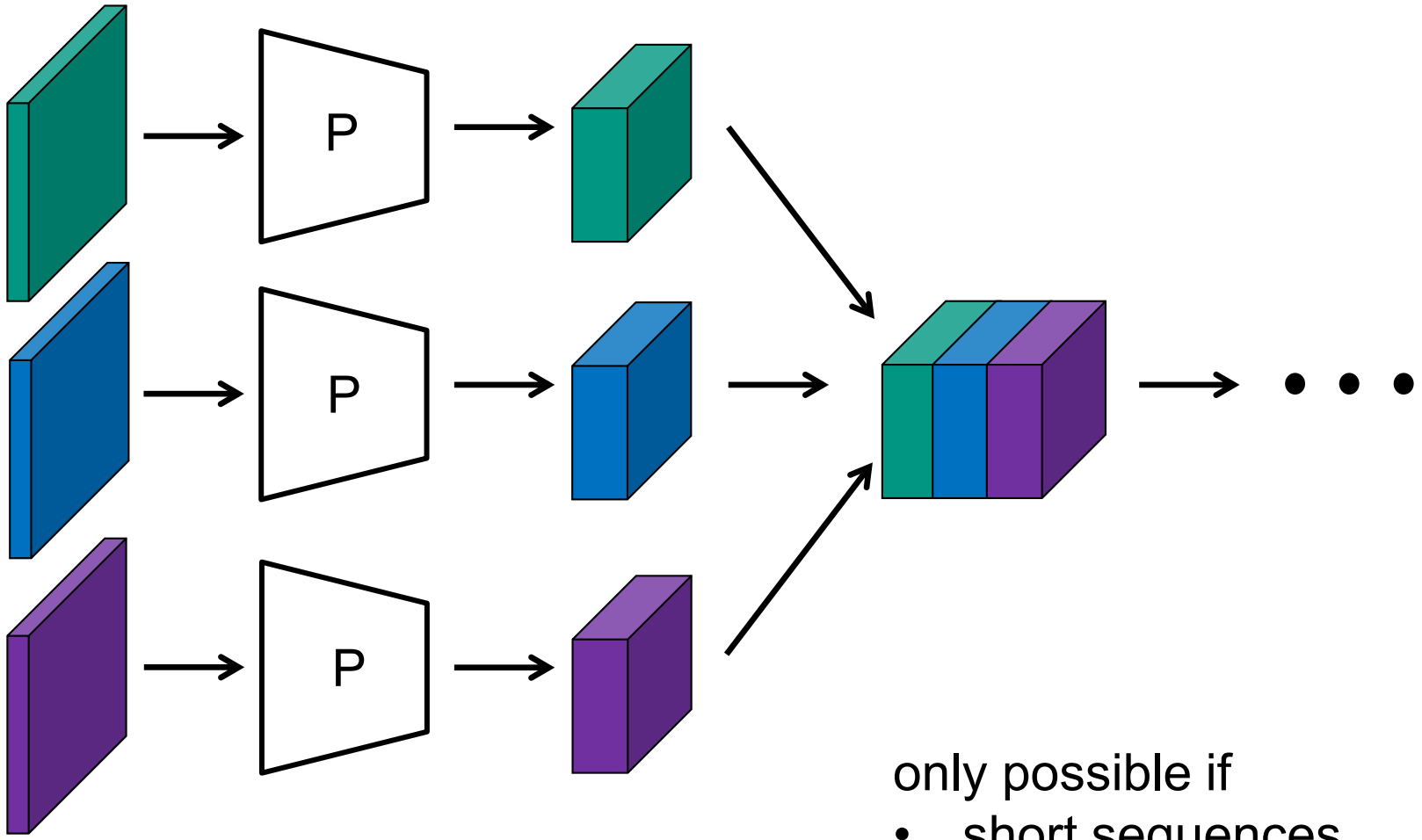


only possible if

- short sequences
- sequences of fixed length

Sequence Processing

(b) individual processing of images + concatenation layer

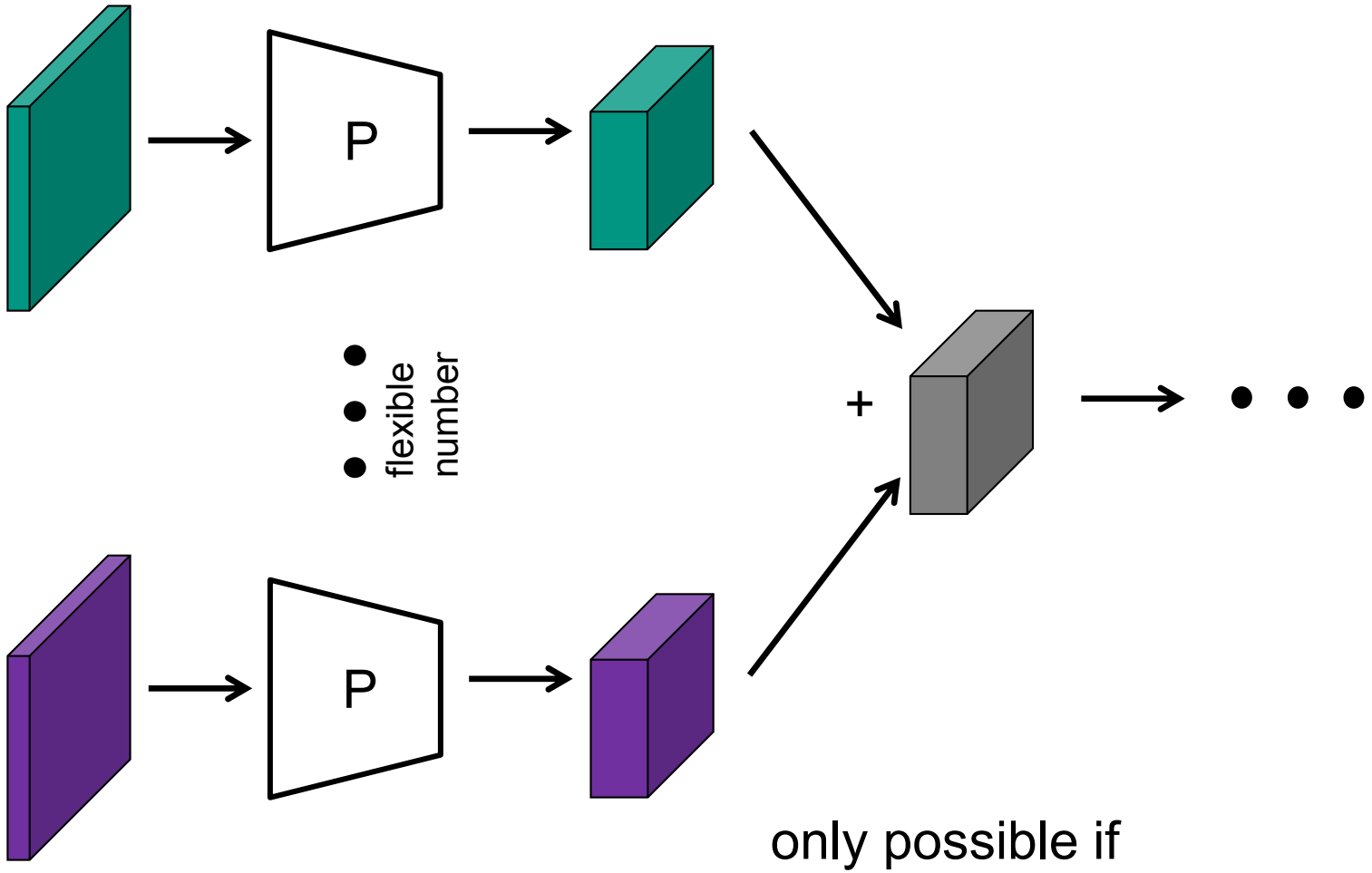


only possible if

- short sequences
- sequences of fixed length

Sequence Processing

(c) individual processing of images + addition layer (deep sets)

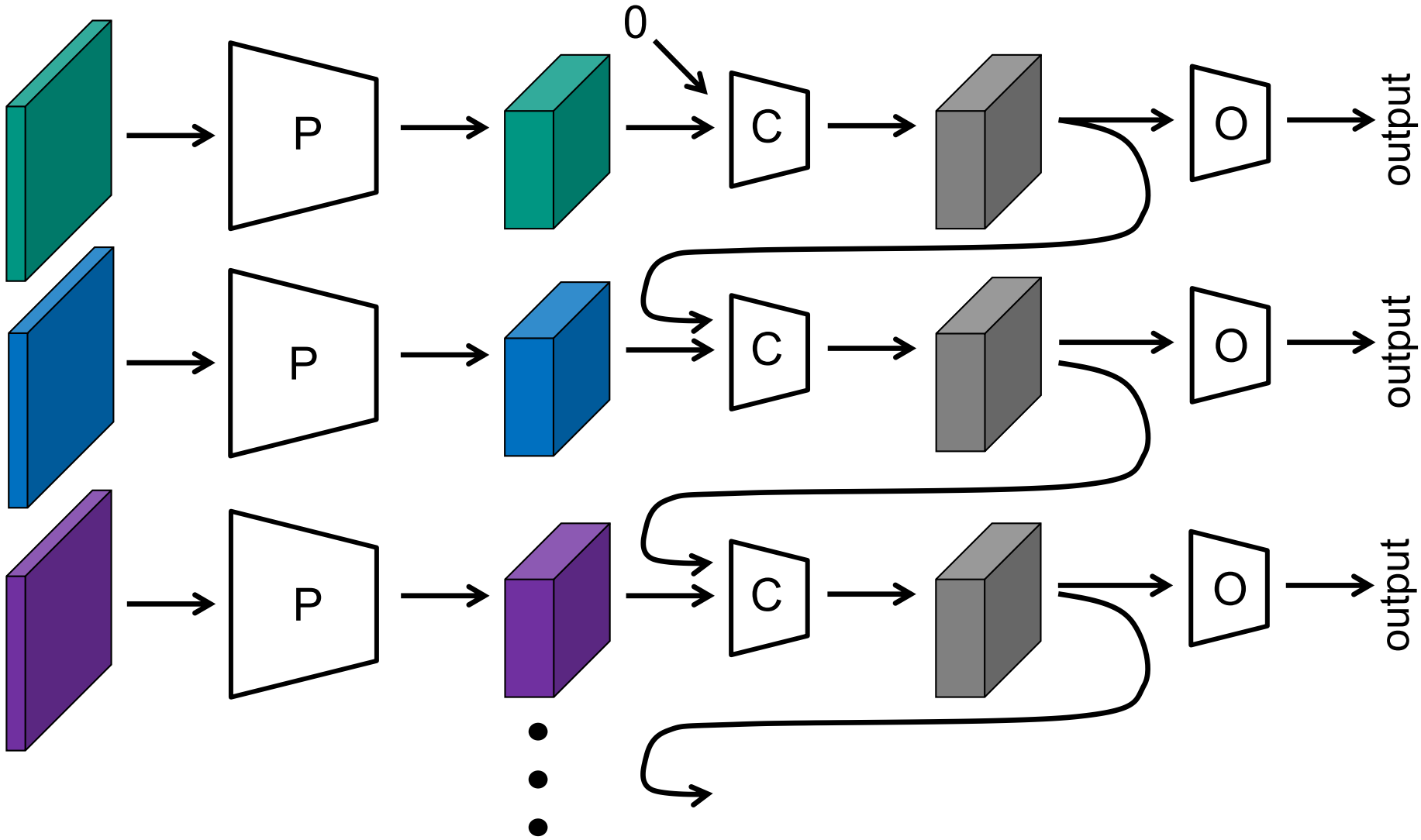


only possible if

- order of images does not matter

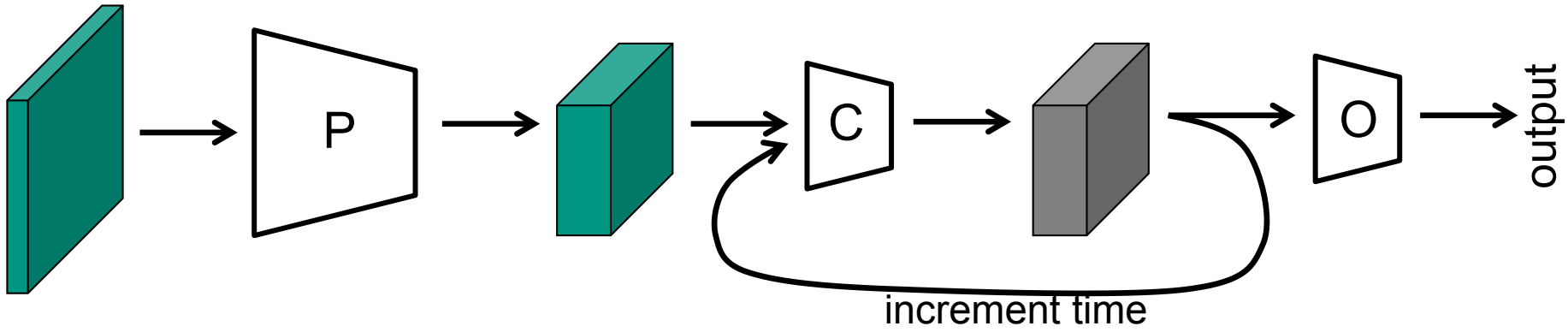
Sequence Processing

(d) recurrent network



Sequence Processing

(d) recurrent network



- learning algorithm: backpropagation through time
- problems: vanishing gradients
- solution: replace perceptrons in network C by appropriate processing units (GRU, LSTM)

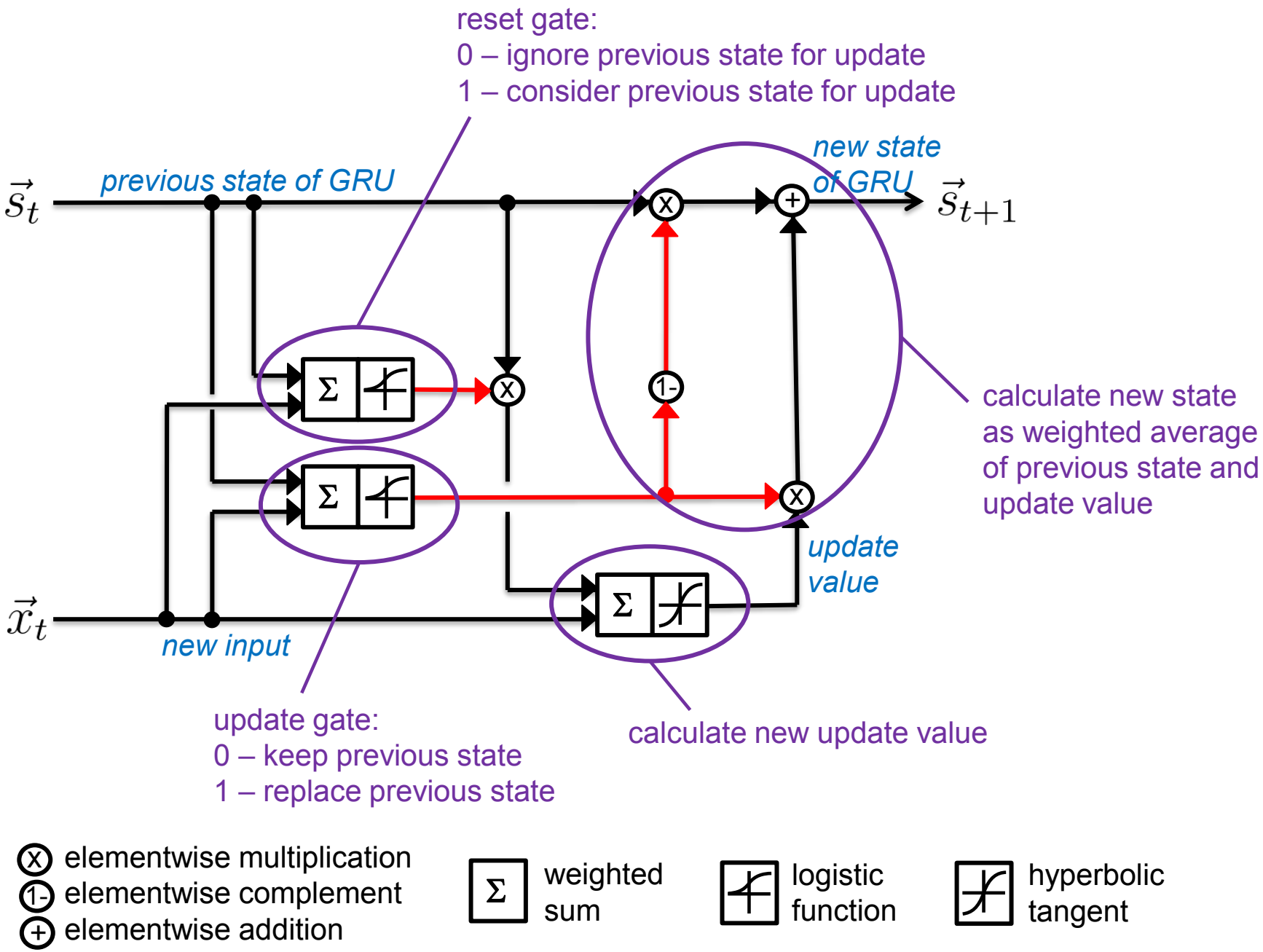
Recurrent Units (GRU+LSTM)

- specialized units that implement a simple state machine
- state is never passed through logistic function of hyperbolic tangent → no vanishing gradients
- internal structures has several *gates* that control the flow of information
- gates are switched on/off using a perceptron mechanism
- LSTM: older (1997), more complex (needs 5 perceptrons)
- GRU: newer (2014), less gates, less parameters (needs 3 perceptrons)

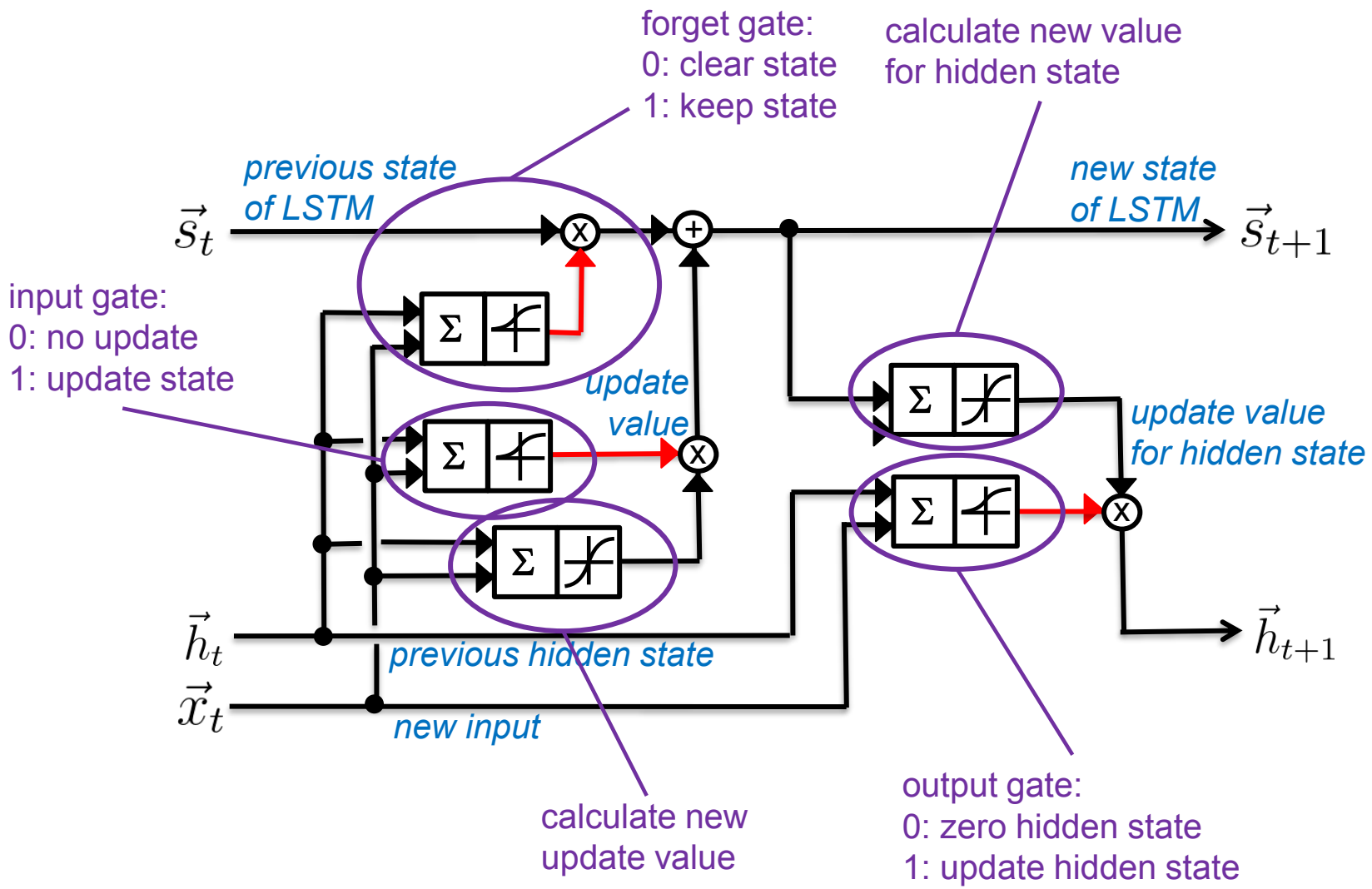
LSTM:
S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Computation 9(8), 1997
F. Gers, J. Schmidhuber, F. Cummins, *Learning to Forget: Continual Prediction with LSTM*. ICANN 1999

GRU:
K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, *Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation*, Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2014

Gated Recurrent Units (GRU)



Long Short Term Memory Units (LSTM)



elementwise multiplication
elementwise addition

weighted sum

logistic function

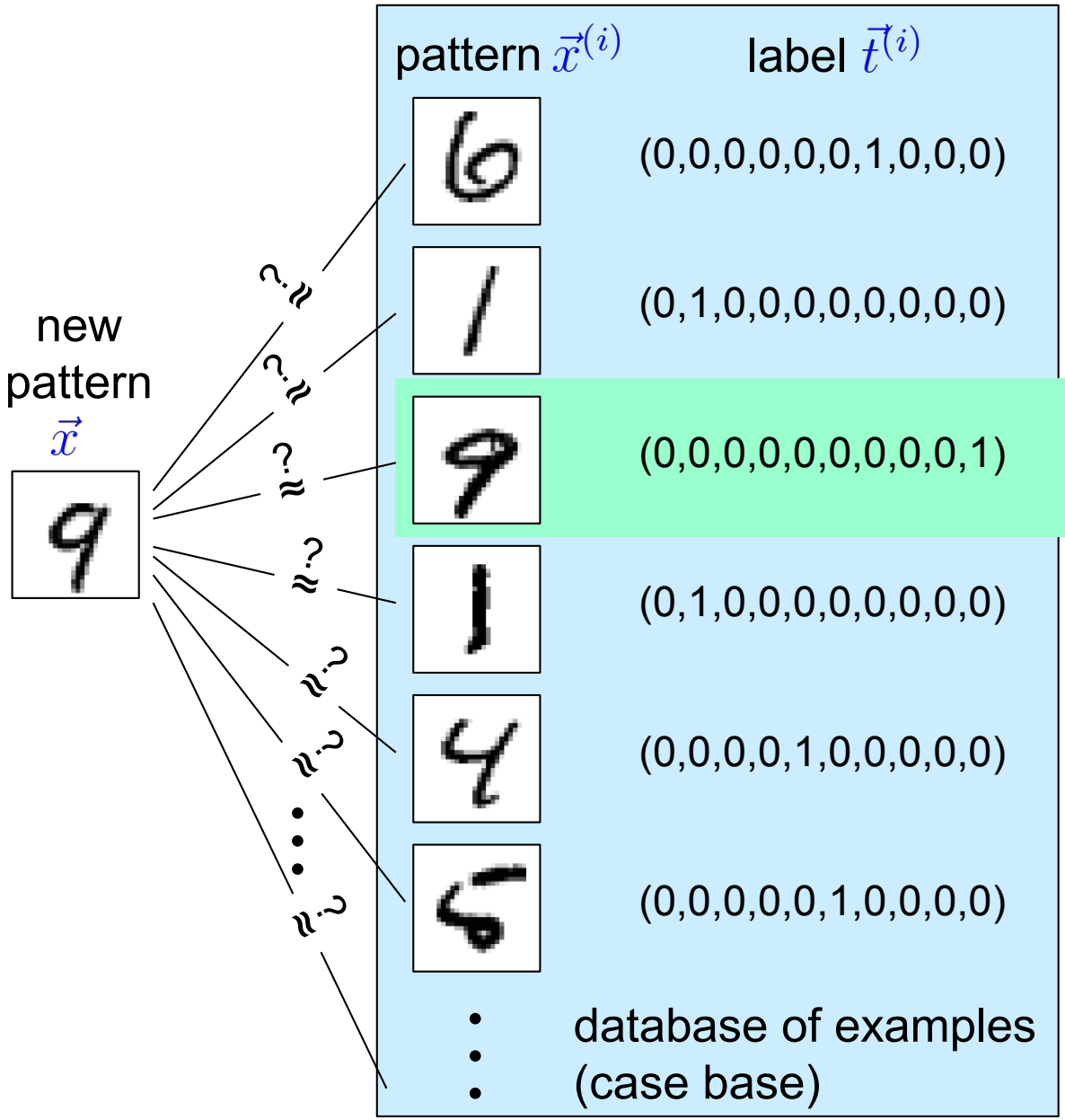
hyperbolic tangent

TRANSFORMERS

Ressources

- Subsequent presentation is based on:
  - Vaswani, Ashish, et al.
    Attention is all you need (arXiv preprint 2017 & NIPS 2017)
  - Devlin, Jacob, et al.
    BERT: Pre-training of deep bidirectional transformers for language understanding
    (arXiv preprint 2018 & NAACL-HLT 2019)
  - Dosovitskiy, Alexey, et al.
    An image is worth 16x16 words: Transformers for image recognition at scale
    (arXiv preprint 2020 & ICLR 2021)
  - Carion, Nicolas, et al.
    End-to-End Object Detection with Transformers (arXiv preprint 2020 & ECCV 2020)
  - Pascal Poupart
    CS480/680 Lecture 19: Attention and transformer Networks
    Video lecture. University of Waterloo

Nearest-Neighbor Approaches

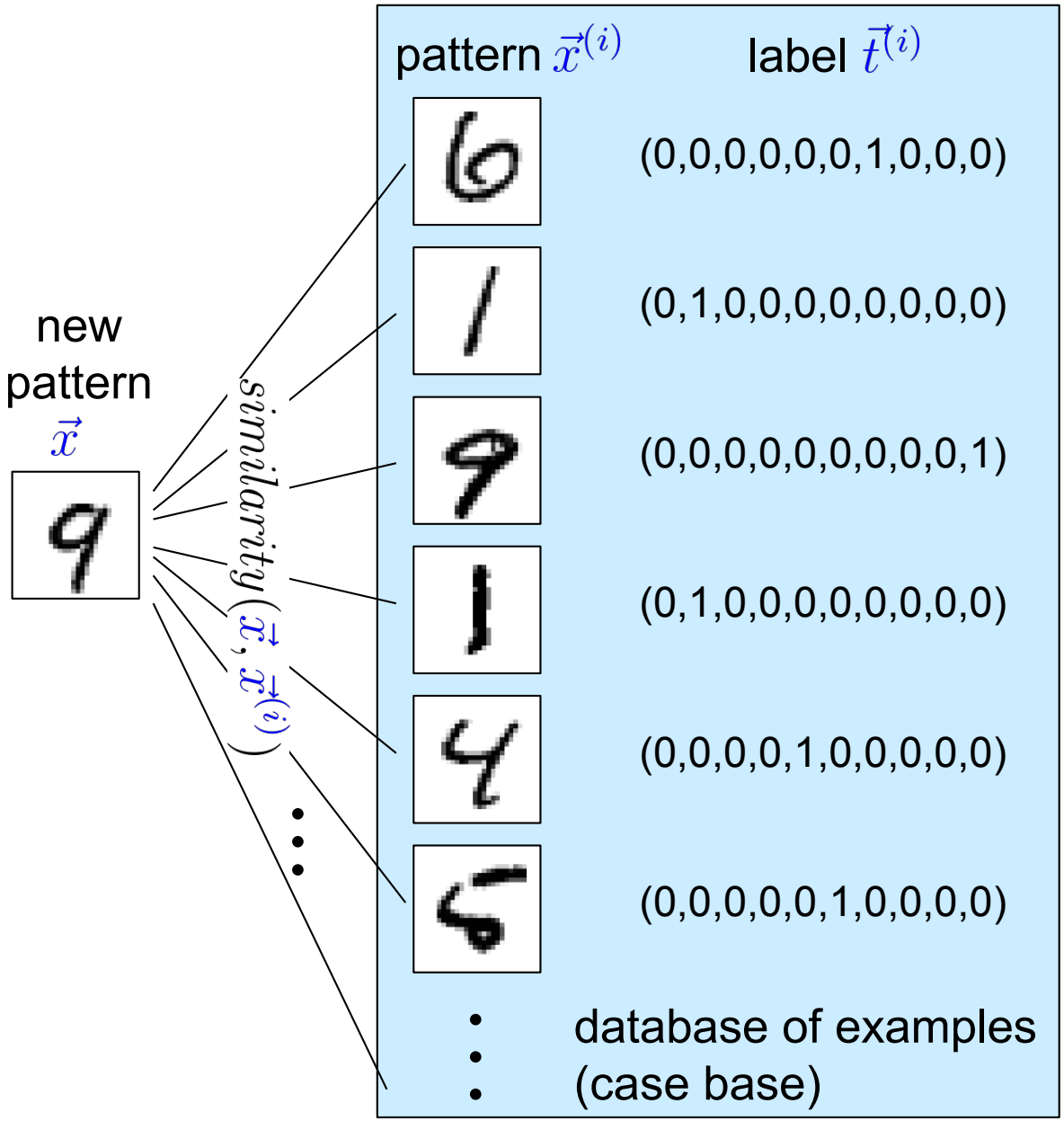


pattern most similar
use label as prediction for $\vec{x}$

Nearest-neighbor classification
Case based reasoning

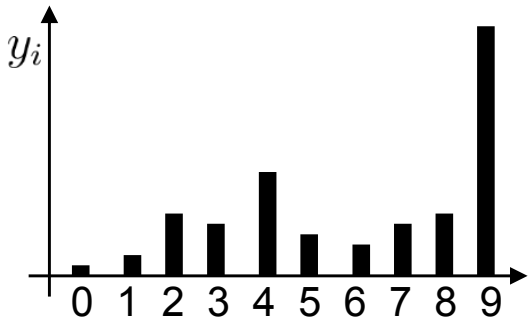
- compare new pattern $\vec{x}$ with examples in case base
- select most similar example $\vec{x}^{(i)}$
- use label $\vec{t}^{(i)}$ as prediction for $\vec{x}$

Nearest-Neighbor Approaches



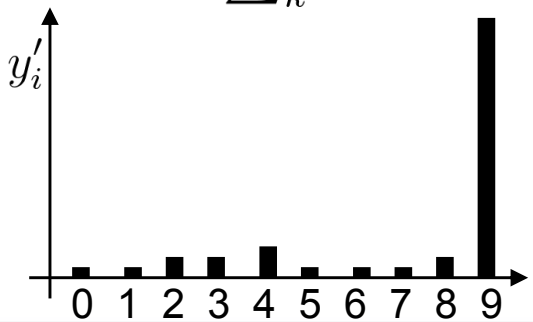
Consider all examples as weighted sum:

$$\vec{y} = \sum_i similarity(\vec{x}, \vec{x}^{(i)}) \cdot \vec{t}^{(i)}$$

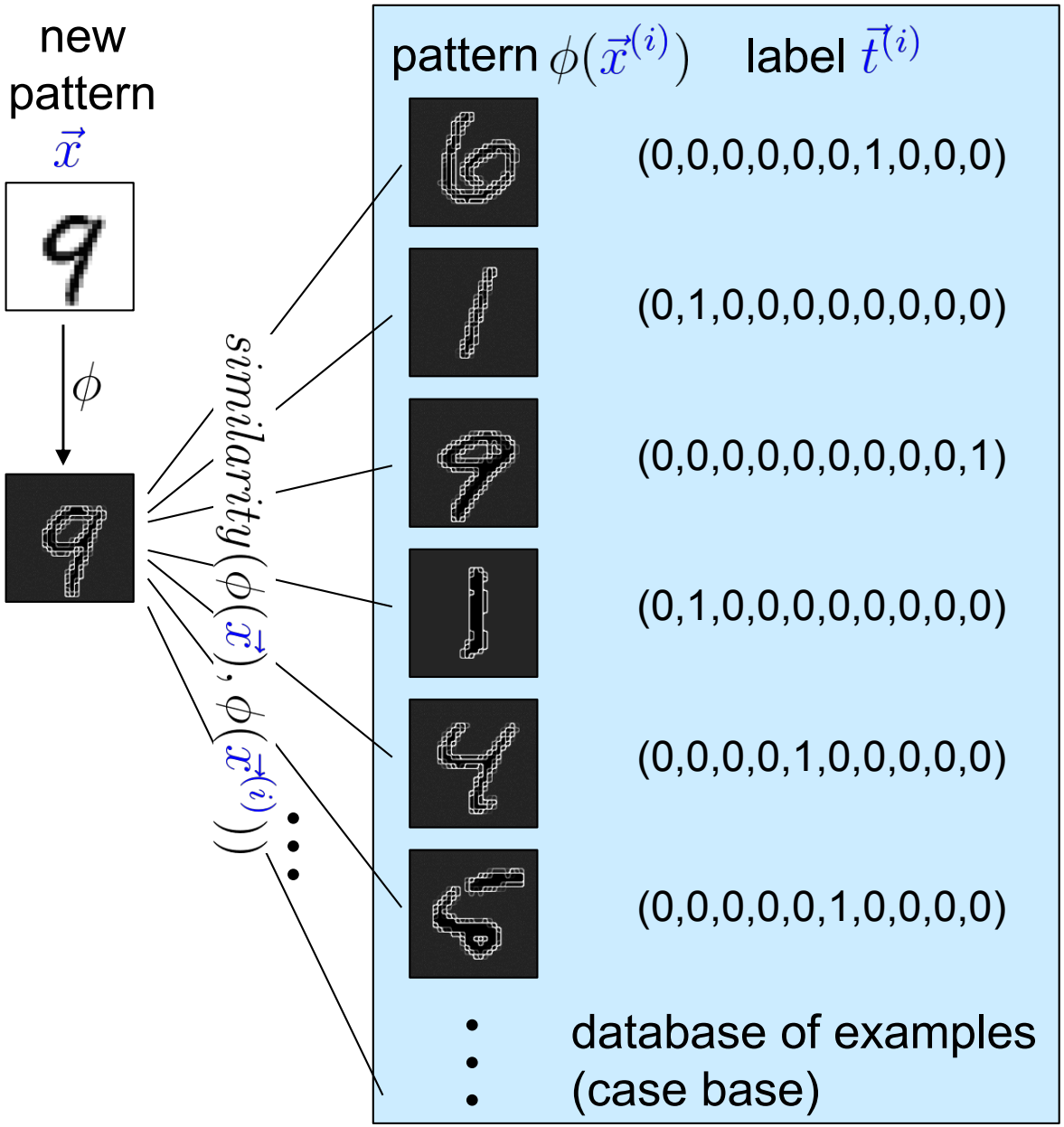


Use softmax to interpret output as probability distribution:

$$y'_j = \frac{e^{y_j}}{\sum_k e^{y_k}}$$



Nearest-Neighbor Approaches



Apply feature transform ϕ to images and perform nearest-neighbor classification in features space

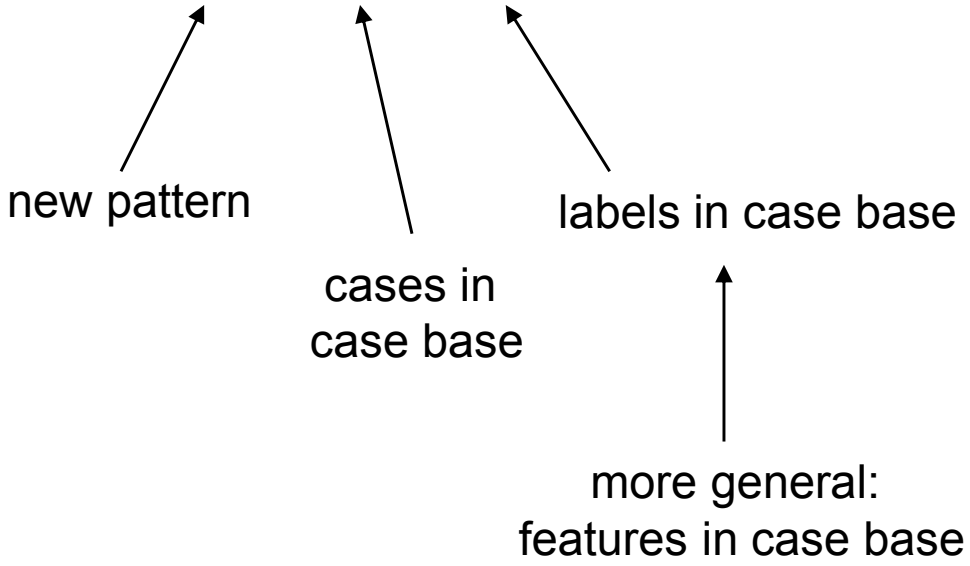
- handcrafted features
- generic features (e.g. HOG)
- a priori learned features (CNNs)
- learned features, e.g.

$$\phi_W(\vec{x}) = W \cdot \vec{x}$$

$$\vec{y}' = softmax\left(\sum_i similarity\left(\phi(\vec{x}), \phi(\vec{x}^{(i)})\right) \cdot \vec{t}^{(i)}\right)$$

Query-Key-Value Systems

- Nearest-neighbor approaches as a Query-Key-Value system



- Analogy: encyclopedia, data base, associative memory

Similarity Measures

- Properties
  - the larger the more similar arguments are
  - symmetric (?)
  - ~~non-negative (?)~~

leaving away this property allows ‚negative' evidence

in NLP words are often represented as 1-out-of-N vectors, i.e. their length is 1

- Frequent choices:
  - dot product

$$similarity(\vec{x}, \vec{y}) = \langle \vec{x}, \vec{y} \rangle = ||\vec{x}||_2 \cdot ||\vec{y}||_2 \cdot \cos\left(\angle(\vec{x}, \vec{y})\right)$$

  - scaled dot product

$$similarity(\vec{x}, \vec{y}) = \frac{\langle \vec{x}, \vec{y} \rangle}{\sqrt{dim(\vec{x})}}$$

  - general dot product

$$similarity(\vec{x}, \vec{y}) = \vec{x}^T \cdot W \cdot \vec{y}$$

learnable weight matrix

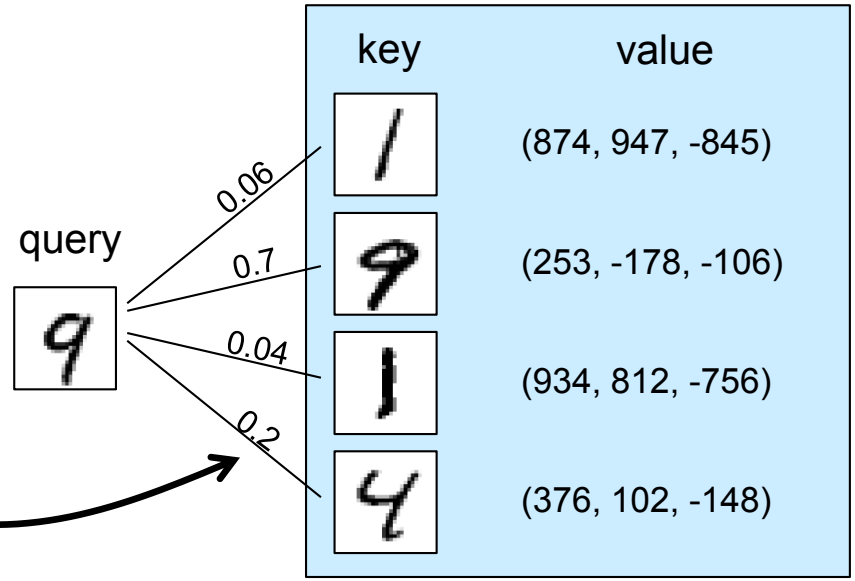
  - additive similarity

$$similarity(\vec{x}, \vec{y}) = \langle \vec{w}_{query}, \vec{x} \rangle + \langle \vec{w}_{key}, \vec{y} \rangle$$

learnable weight vectors

Interpretation as Attention Mechanism

numbers tell you which cases
“attract the attention“ of the query



distribution over cases in case base

$$\vec{y} = softmax\left(\sum_{j=1}^{n}\left(similarity(\overrightarrow{query}, \overrightarrow{key}^{(j)}) \cdot \vec{t}^{(j)}\right)\right) \cdot \begin{pmatrix} (\overrightarrow{value}^{(j)})^T \\ \vdots \\ (\overrightarrow{value}^{(n)})^T \end{pmatrix}$$

weighted sum over values in case base

with $\vec{t}^{(j)} = (0, \cdots, 0, \underset{\uparrow}{1}, 0, \cdots, 0)$
j

Tokenization

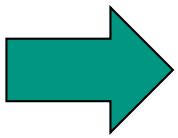
- **tokenization** in NLP: segmentation of text corpora into pieces (words/tokens/symbols)

"Which fruit do you like?"

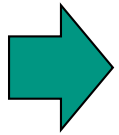
```
["which"]
["fruit"]
["do"]
["you"]
["like"]
[question mark]
```

1-out-of-n vectors

- **tokenization** in CV: splitting images into patches



split up image into patches (non-overlapping or overlapping)

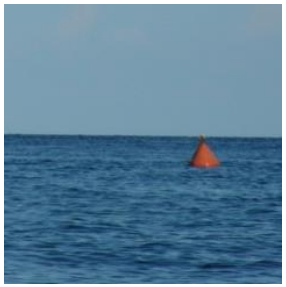


vectors

Self-Attention Mechanism

- generate query, key, and value for each token



$$\overrightarrow{query}^{(1)} = W_{query} \cdot \vec{x}^{(1)}$$
$$\overrightarrow{key}^{(1)} = W_{key} \cdot \vec{x}^{(1)}$$
$$\overrightarrow{value}^{(1)} = W_{value} \cdot \vec{x}^{(1)}$$

$$\overrightarrow{query}^{(2)} = W_{query} \cdot \vec{x}^{(2)}$$
$$\overrightarrow{key}^{(2)} = W_{key} \cdot \vec{x}^{(2)}$$
$$\overrightarrow{value}^{(2)} = W_{value} \cdot \vec{x}^{(2)}$$

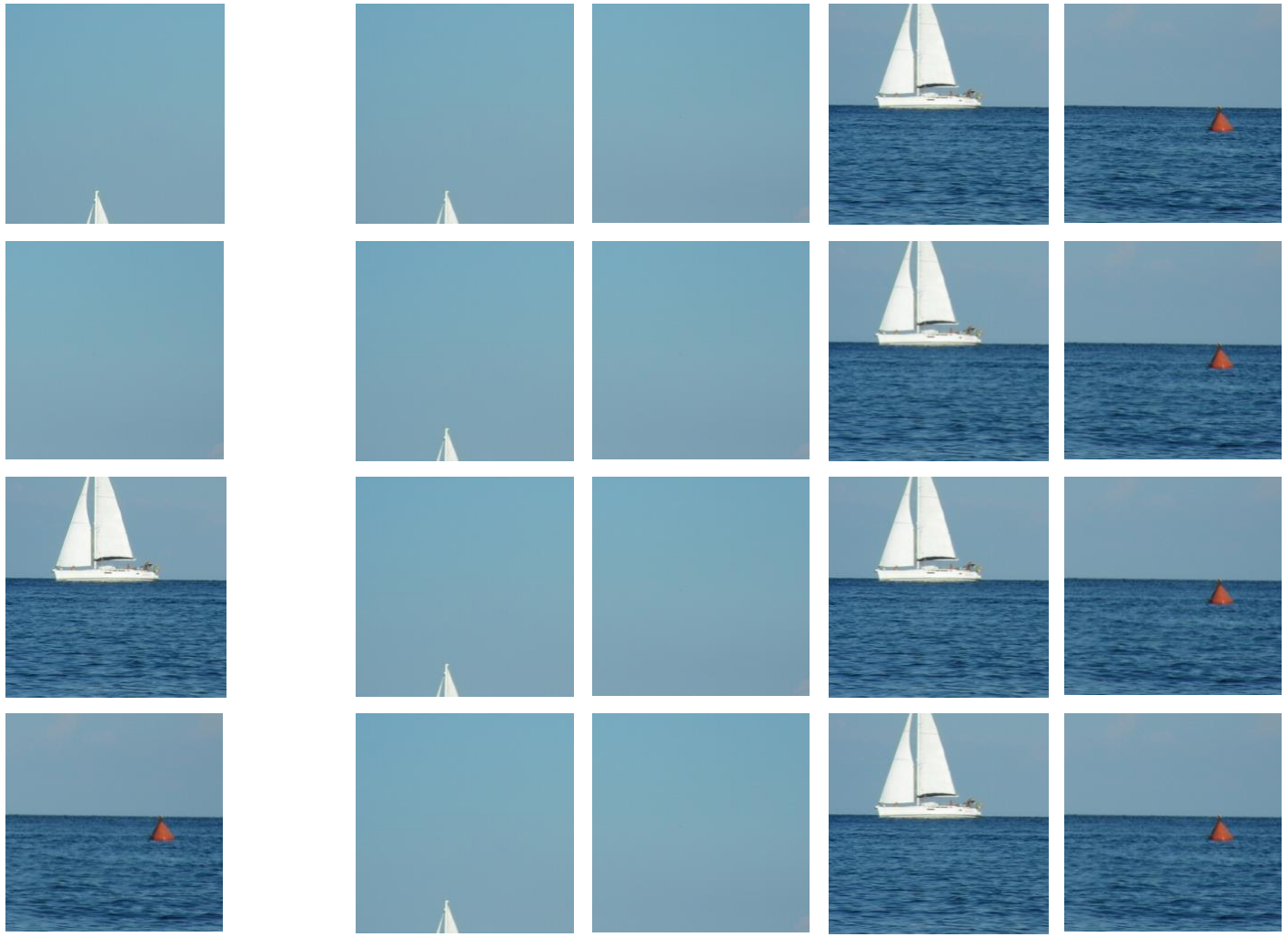
$$\overrightarrow{query}^{(3)} = W_{query} \cdot \vec{x}^{(3)}$$
$$\overrightarrow{key}^{(3)} = W_{key} \cdot \vec{x}^{(3)}$$
$$\overrightarrow{value}^{(3)} = W_{value} \cdot \vec{x}^{(3)}$$

$$\overrightarrow{query}^{(4)} = W_{query} \cdot \vec{x}^{(4)}$$
$$\overrightarrow{key}^{(4)} = W_{key} \cdot \vec{x}^{(4)}$$
$$\overrightarrow{value}^{(4)} = W_{value} \cdot \vec{x}^{(4)}$$

Self-Attention Mechanism

- apply Query-Key-Value approach in a cyclic manner

query | case base | result/attention value



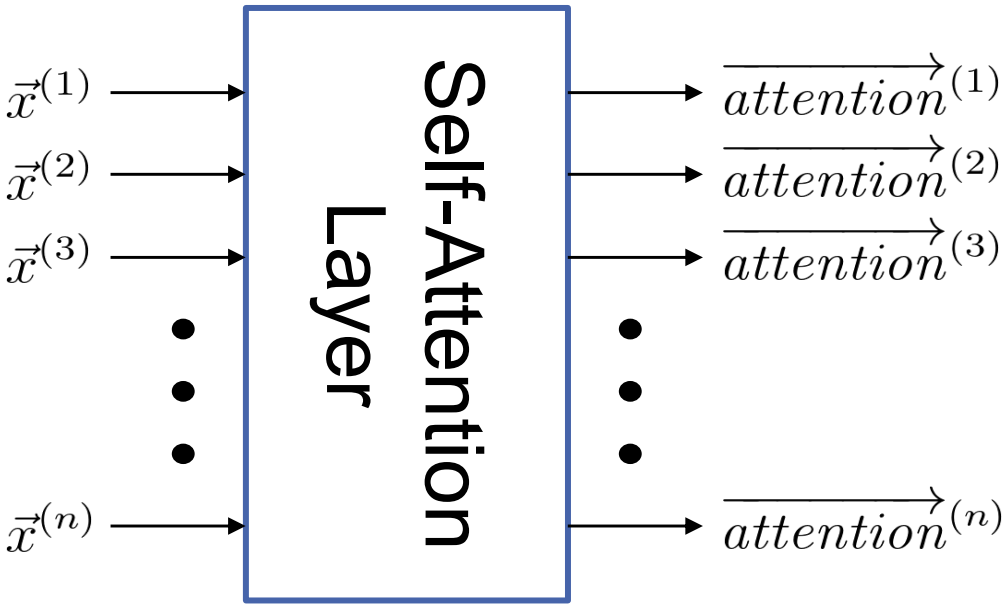
$\overrightarrow{attention}^{(1)}$

$\overrightarrow{attention}^{(2)}$

$\overrightarrow{attention}^{(3)}$

$\overrightarrow{attention}^{(4)}$

Self-Attention Mechanism



$$\overrightarrow{attention}^{(i)} = softmax\left(\sum_{j=1}^{n}\left(similarity(\underbrace{W_{query}\cdot\vec{x}^{(i)}}_{\text{query}}, \underbrace{W_{key}\cdot\vec{x}^{(j)}}_{\text{key}})\cdot\vec{t}^{(j)}\right)\right)\cdot\underbrace{\begin{pmatrix}(W_{value}\cdot\vec{x}^{(1)})^T\\ \vdots \\ (W_{value}\cdot\vec{x}^{(n)})^T\end{pmatrix}}_{\text{values}}$$

with $\vec{t}^{(j)} = (0,\cdots,0,\underset{\uparrow}{1},0,\cdots,0)$ $\quad j$

trainable parameters: $W_{query}, W_{key}, W_{value},$ parameters of $similarity$

Self-Attention Mechanism

$$\overrightarrow{attention}^{(i)} = softmax\left(\sum_{j=1}^{n}\left(similarity(W_{query}\cdot\vec{x}^{(i)}, W_{key}\cdot\vec{x}^{(j)})\cdot\vec{t}^{(j)}\right)\right)\cdot\begin{pmatrix}(W_{value}\cdot\vec{x}^{(1)})^T\\ \vdots \\ (W_{value}\cdot\vec{x}^{(n)})^T\end{pmatrix}$$

Example:

$$W_{query} = \begin{pmatrix}1 & 0\\ 2 & -1\end{pmatrix}$$

$$W_{key} = \begin{pmatrix}1 & 1\\ 0 & 1\end{pmatrix}$$

$$W_{value} = \begin{pmatrix}2 & -1\\ -1 & 1\end{pmatrix}$$

$$similarity(\vec{x},\vec{y}) = \langle\vec{x},\vec{y}\rangle$$

	input	query	key	value	probability vector	output(attention)
$\vec{x}^{(1)}$	$\begin{pmatrix}2\\-1\end{pmatrix}$	$\begin{pmatrix}2\\1\end{pmatrix}$	$\begin{pmatrix}1\\-1\end{pmatrix}$	$\begin{pmatrix}5\\-3\end{pmatrix}$	$(0.000, 0.000, 1.000)$	$\begin{pmatrix}2\\0\end{pmatrix}$
$\vec{x}^{(2)}$	$\begin{pmatrix}-1\\0\end{pmatrix}$	$\begin{pmatrix}-1\\0\end{pmatrix}$	$\begin{pmatrix}-1\\0\end{pmatrix}$	$\begin{pmatrix}-2\\1\end{pmatrix}$	$(0.118, 0.876, 0.006)$	$\begin{pmatrix}-1.150\\0.522\end{pmatrix}$
$\vec{x}^{(3)}$	$\begin{pmatrix}2\\2\end{pmatrix}$	$\begin{pmatrix}2\\2\end{pmatrix}$	$\begin{pmatrix}4\\2\end{pmatrix}$	$\begin{pmatrix}2\\0\end{pmatrix}$	$(0.000, 0.000, 1.000)$	$\begin{pmatrix}2\\0\end{pmatrix}$

similarity		key $\vec{x}^{(1)}$	key $\vec{x}^{(2)}$	key $\vec{x}^{(3)}$
query	$\vec{x}^{(1)}$	1	−2	10
query	$\vec{x}^{(2)}$	−1	1	−4
query	$\vec{x}^{(3)}$	0	−2	12

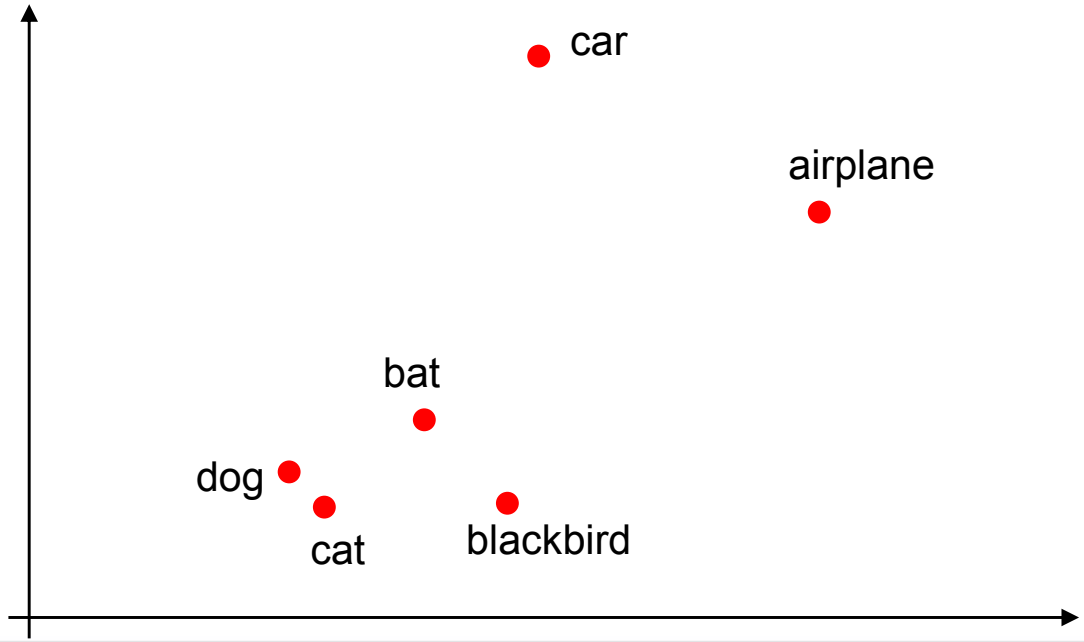
softmax

linear combination

Interpreting Self-Attention Layers as Embedding

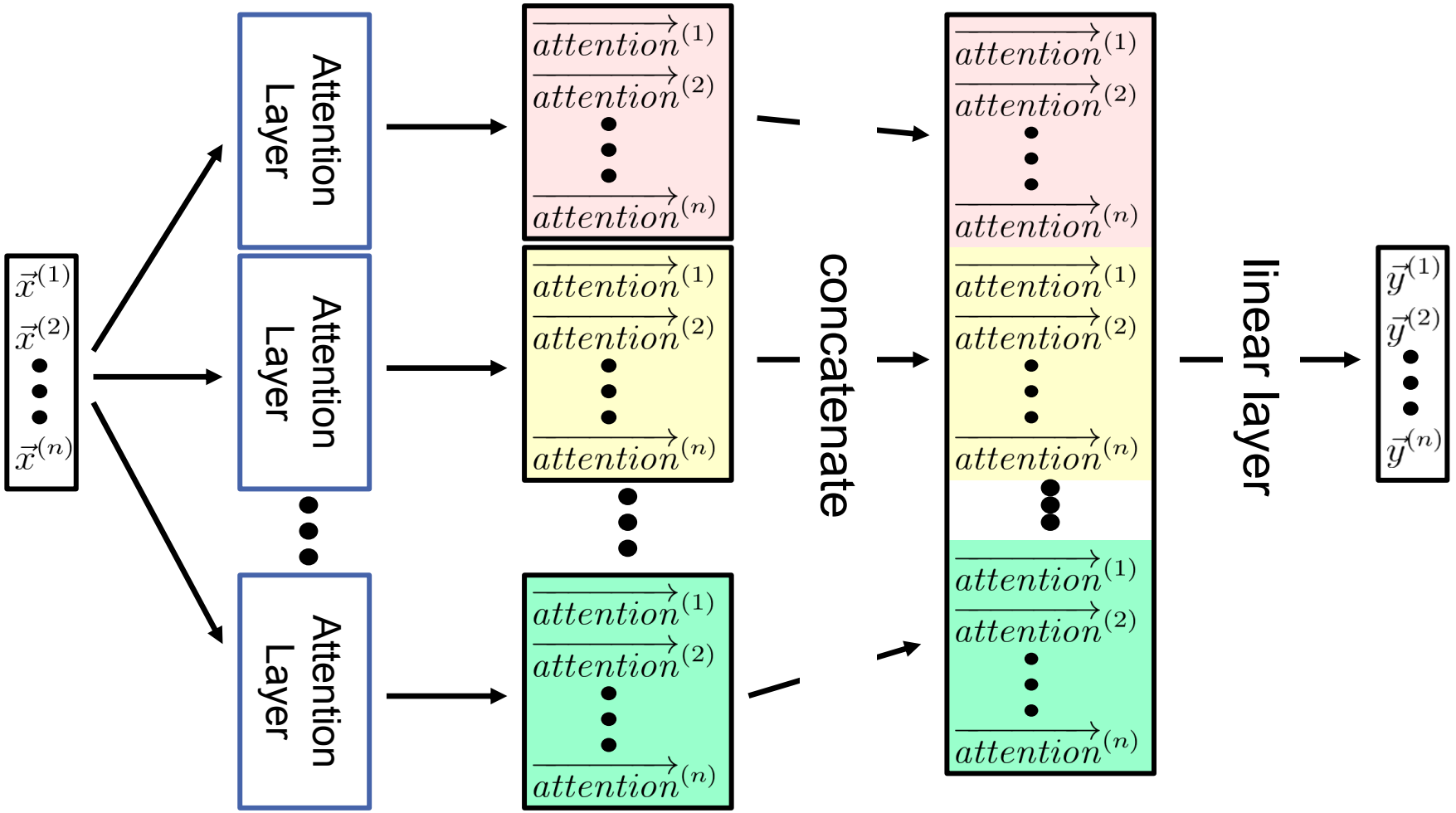
$$\overrightarrow{attention}^{(i)} = softmax\left(\sum_{j=1}^{n}\left(similarity(W_{query}\cdot\vec{x}^{(i)}, W_{key}\cdot\vec{x}^{(j)})\cdot\vec{t}^{(j)}\right)\right)\cdot\begin{pmatrix}(W_{value}\cdot\vec{x}^{(1)})^T\\ \vdots \\ (W_{value}\cdot\vec{x}^{(n)})^T\end{pmatrix}$$

- Assumption: input vectors are 1-out-of-n vectors
  e.g. representing words in a dictionary
- Then, $W_{query}\cdot\vec{x},\ W_{key}\cdot\vec{x}$ map tokens into a feature space
- $W_{value}\cdot\vec{x}$ maps tokens into (another= feature space)



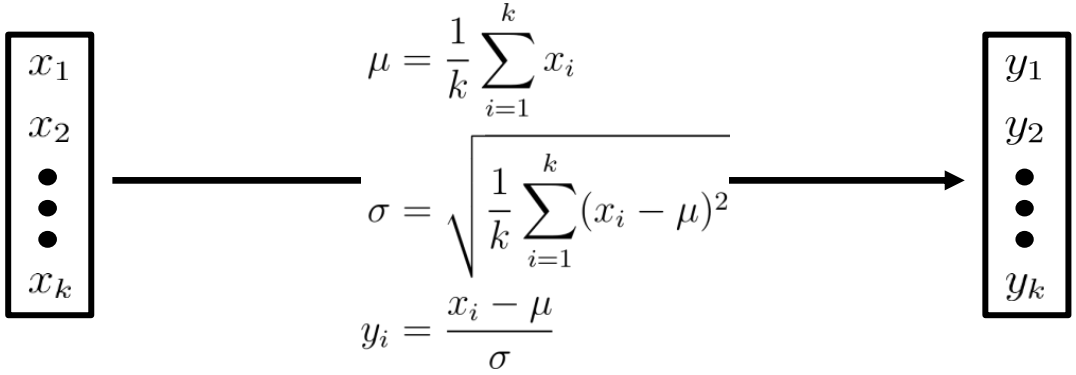
Multi-Headed Attention Layers

- like in convolutional layers one might want to create different feature respresentations in one transformer layer
  → multi-headed attention layers



Layer Normalization

- layer normalization serves to scale activation vectors to zero mean and unit standard deviation
- used to prepare outputs of one transformer layer for the subsequent layer



Position Encoding

- transformers themselves do not consider the position of tokens

- need to add position information if arrangement of tokens matters *(position embedding)*
  - encode patch number as feature vector
  - possible encoding, suggested for NLP applications

$$p \mapsto \left(\ldots, \sin \frac{p}{n^{\frac{2i}{d}}}, \cos \frac{p}{n^{\frac{2i}{d}}}, \ldots\right)$$

token number

$2i$ $2i+1$

vector of length *d*

a magic constant, chosen by user, e.g. 10000

  - add or concatenate to feature vector

BERT – Bidirectional Encoder Representation from Transformers
ViT – Vision Transformer

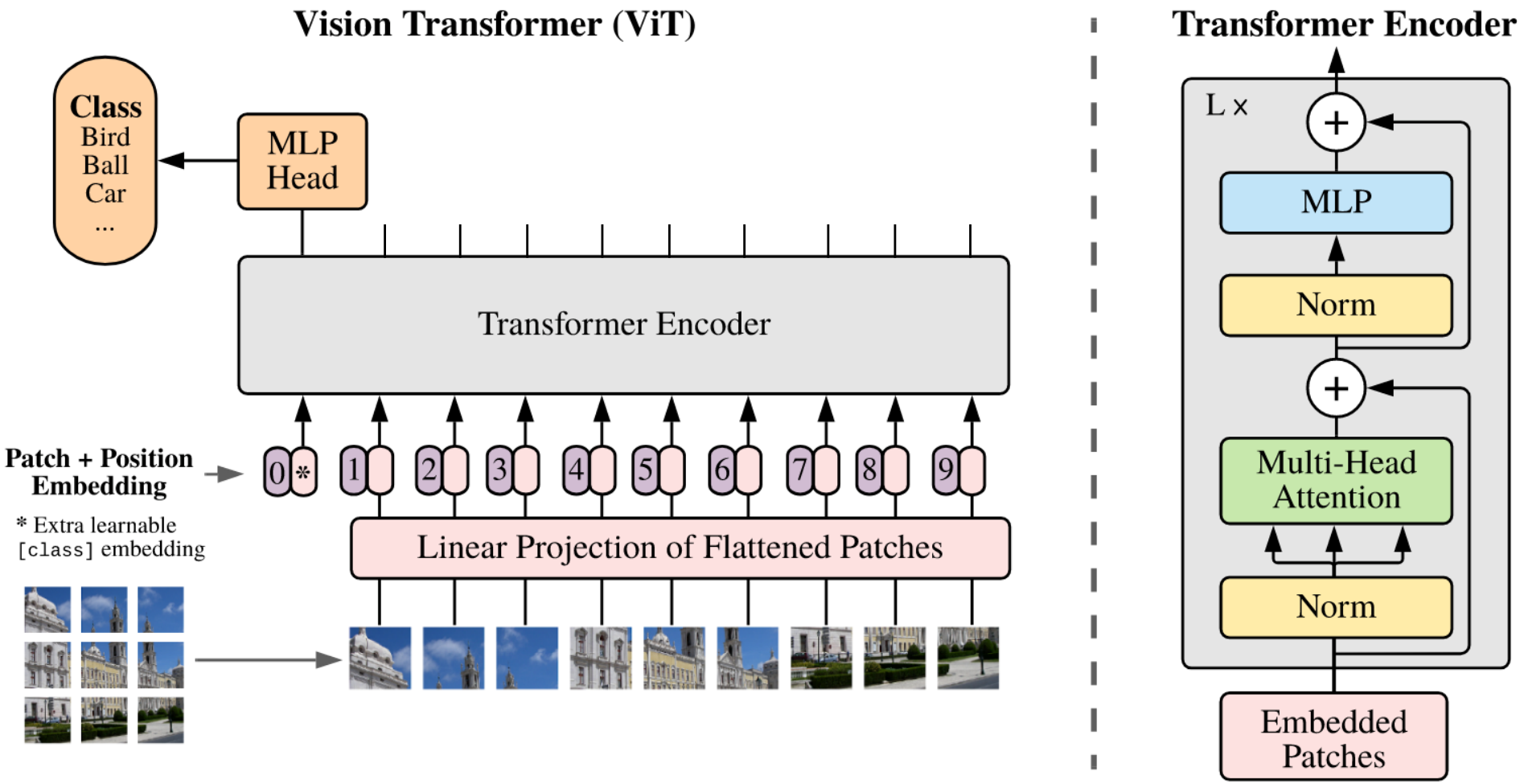


image taken from: A. Dosovitskiy et al., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, ICLR 2021

DETR: Detection Transformer

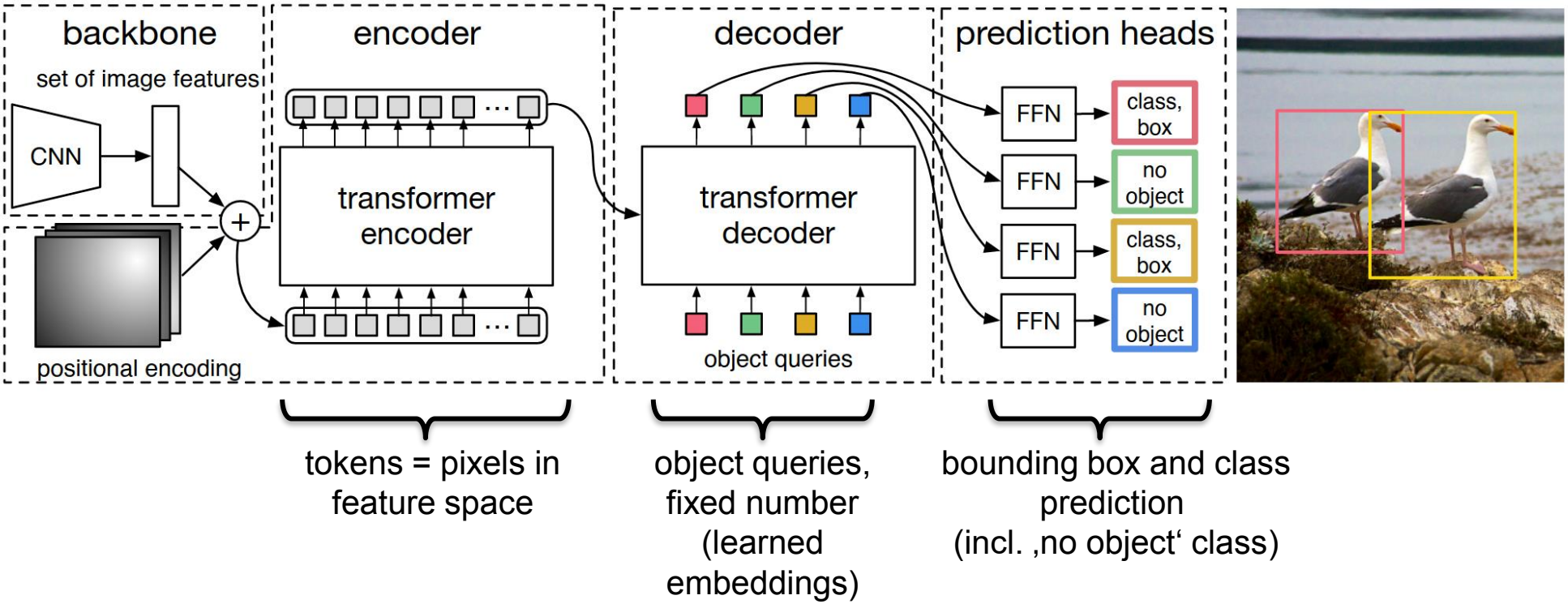
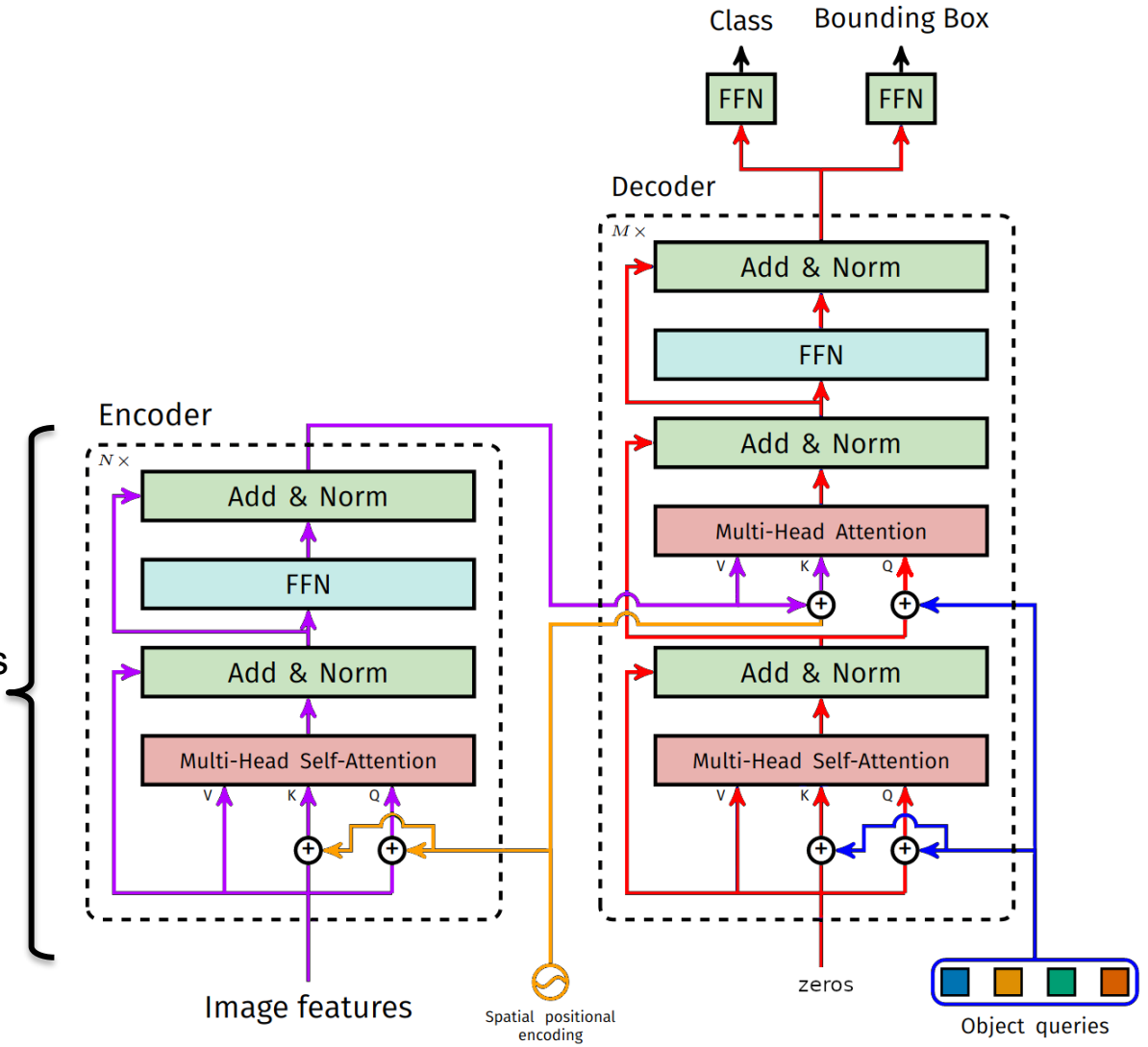


image taken from: N. Carion et al., *End-to-End Object Detection with Transformers*, ECCV 2020

DETR: Detection Transformer



prediction head: return output in desired format

process image content on layers of increasing abstraction

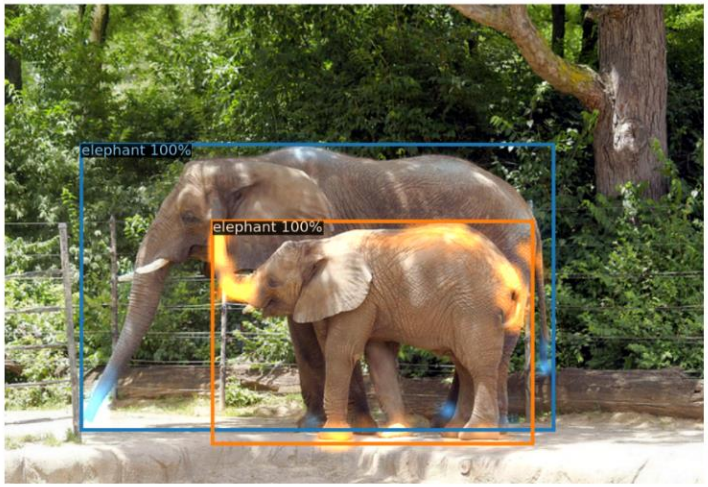
object queries:
- request 1st object
- request 2nd object
- …

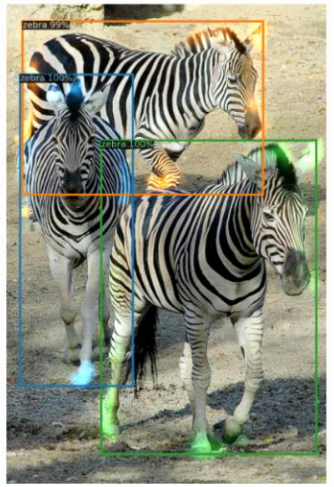
image taken from: N. Carion et al., *End-to-End Object Detection with Transformers*, ECCV 2020
adapted by M. Lauer

Results

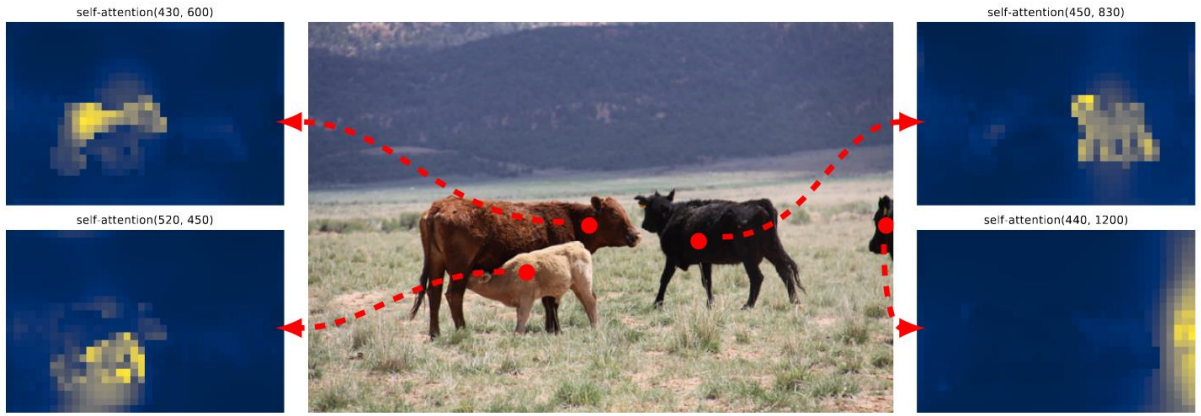
- Vision Transformers
  - comparable results to CNNs on classification benchmarks
  - require hugh amounts of training data and training time

- DETR
  - comparable results to Region-CNNs
  - high computational effort





decoder attention

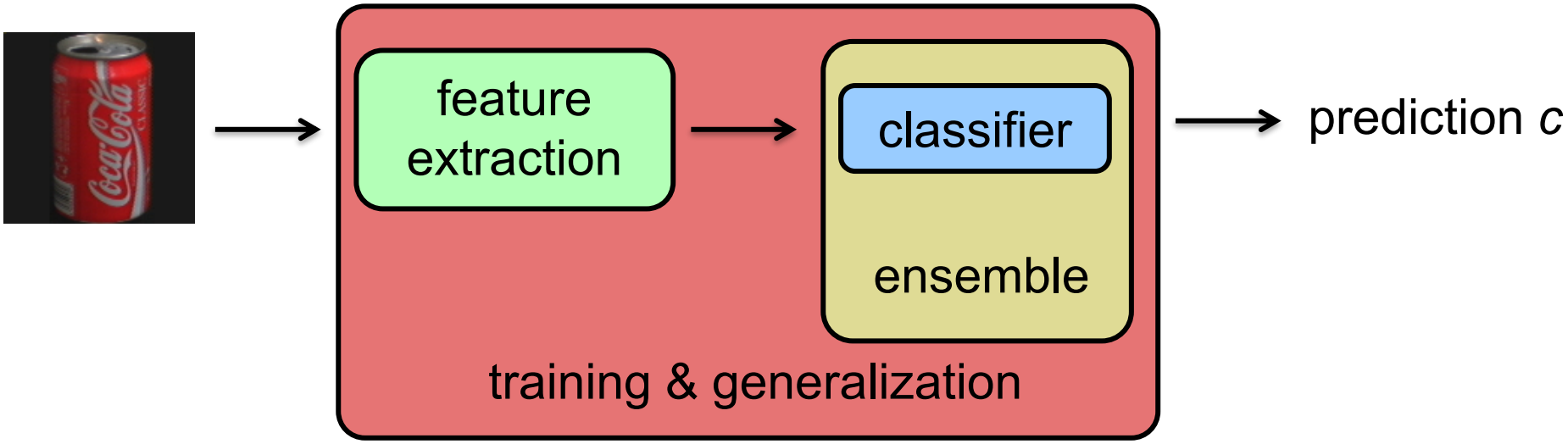


encoder attention

images taken from: N. Carion et al., *End-to-End Object Detection with Transformers*, ECCV 2020

SUMMARY: DEEP LEARNING

Pattern Recognition: the Complete Picture



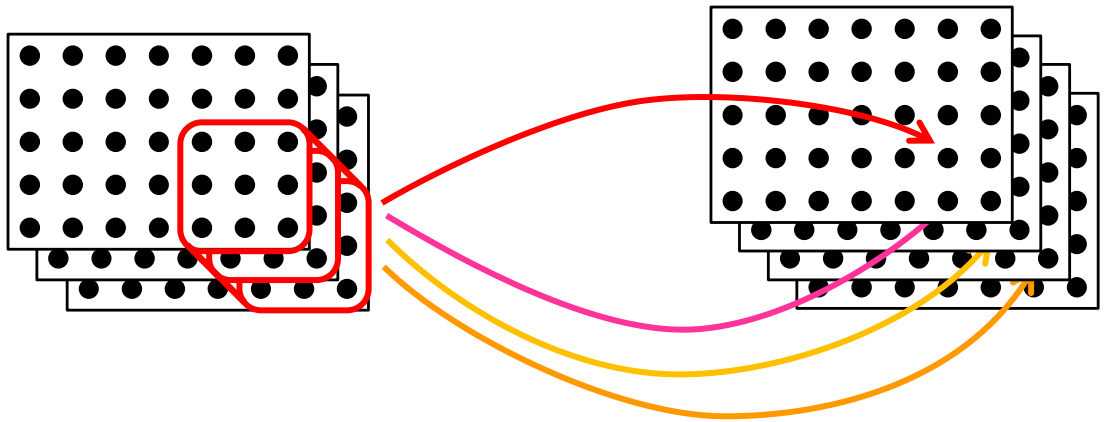
features	classifiers	ensembles	generalization
▪ smart (specific)	▪ SVM	▪ free ensemble	▪ validation
▪ HOG	▪ threshold	▪ bagging	▪ cross-validation
▪ Haar	▪ decision tree	▪ boosting	▪ data tuning
▪ LBP	▪ cascade	▪ decision forest	▪ early stopping
▪ neural features	▪ neural network	▪ multi-class	▪ randomization
▪ …	▪ …	▪ …	▪ …

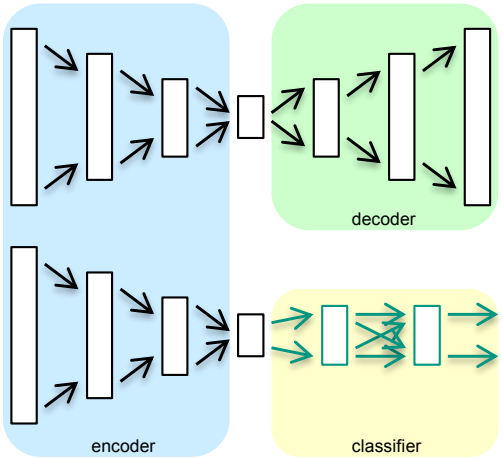
Pattern Recognition: the Complete Picture cont.

features

- smart (specific)
- HOG
- Haar
- LBP
- neural features
- …

- neural features
  - generated by autoencoder networks
  - generated by convolutional networks
  - transformers
  - neural features often outperform “traditional” features (e.g. HOG) even if trained on different images



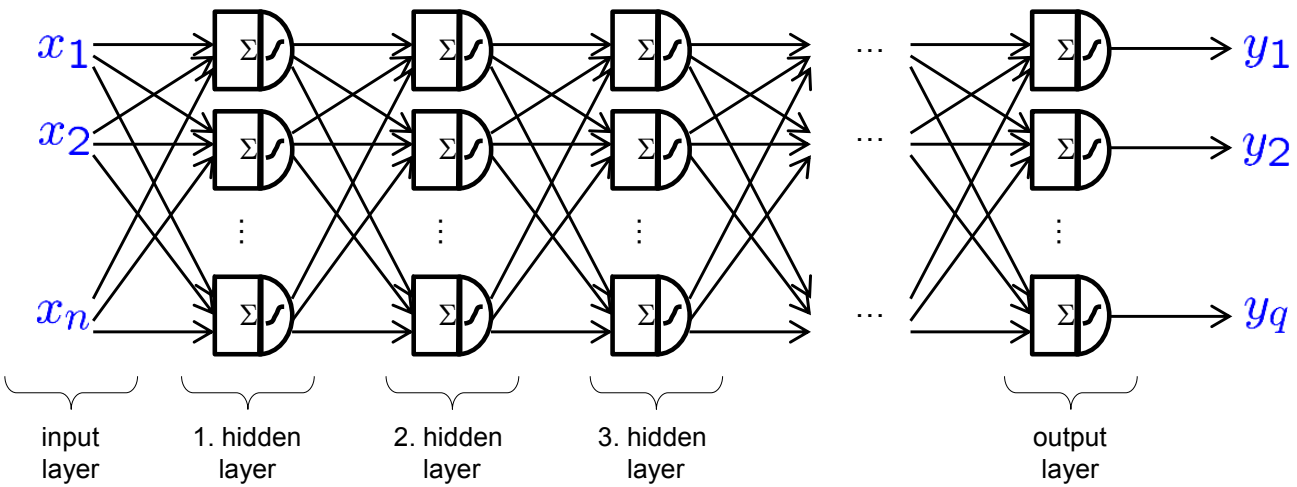


Pattern Recognition: the Complete Picture cont.

classifiers

- SVM
- threshold
- decision tree
- cascade
- neural network
- …

- artificial neural network
  - highly parameterized
  - nonlinear functions
  - build out of simple blocks (perceptrons)
  - layered layout



Pattern Recognition: the Complete Picture cont.

generalization

- validation
- cross-validation
- data tuning
- early stopping
- randomization
- …

- early stopping
  - monitor validation error
  - stop training at minimum
- weight decay/L2-regularisation
  - preference for small weights
  - punish nonlinearity
- dropout
  - randomly switch off percpetrons
  - foster distributed representation
- multi task learning
  - train related tasks with same network
  - share feature layers
- reuse of layers and networks
  - reuse trained feature layers
  - replace and retrain classification layers
  - little retraining of all weights

References

- books and articles
  - I. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. MIT Press, 2016. online: http://www.deeplearningbook.org
    extensive description of deep learning techniques
  - Y. LeCun, Y. Bengio, G. E. Hinton. *Deep Learning*. nature 521, pp. 436-444, 2015
    brief overview paper, not very deep but good to get a rough idea
  - L. Deng, D. Yu. *Deep learning: methods and applications*. Foundations and Trends in Signal Processing 7:3–4, pp. 197-387, 2014. online: http://ftp.nowpublishers.com/article/Details/SIG-039
    extensive overview and introduction, focus on natural language processing
  - Y. Bengio. Learning Deep Architectures for AI. Foundations and Trends in Machine Learning 2:1, pp. 1-127, 2009. online: http://www.nowpublishers.com/article/Details/MAL-006
    extensive introduction and overview, does not contain newer developments

References

- toolboxes
  - Tensorflow (Google), https://www.tensorflow.org
    deep learning engine that allows more extensions of existing approaches, good for experienced persons
  - Torch/PyTorch (Community/Facebook), https://pytorch.org
    comparable possibilities as Tensorflow

AI, Deep Learning, Artificial Neural Networks … All the Same?