Formation école doctorale :

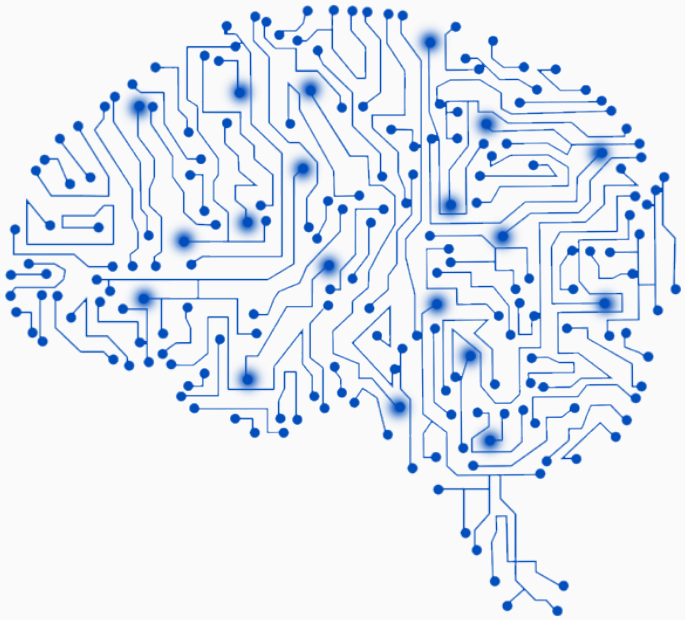
Introduction au traitement d'images

Introduction to image processing

Course IV – Introduction to Artificial Neural Networks for image classification

Bruno Galerne

Mardi 28 juin 2022



Credits

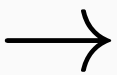
Most of the slides from **Charles Deledalle's** course "UCSD ECE285 Machine learning for image processing" (30 $\times$ 50 minutes course)

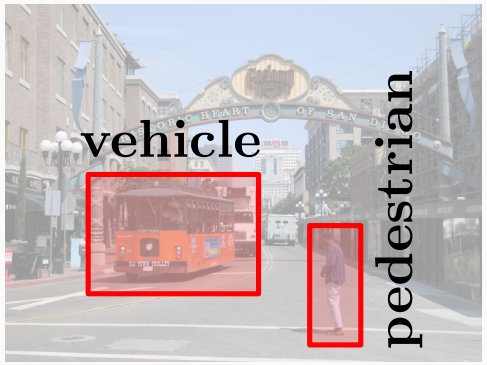
`www.charles-deledalle.fr/`

`https://www.charles-deledalle.fr/pages/teaching.php#learning`

Computer Vision and Machine Learning

Image



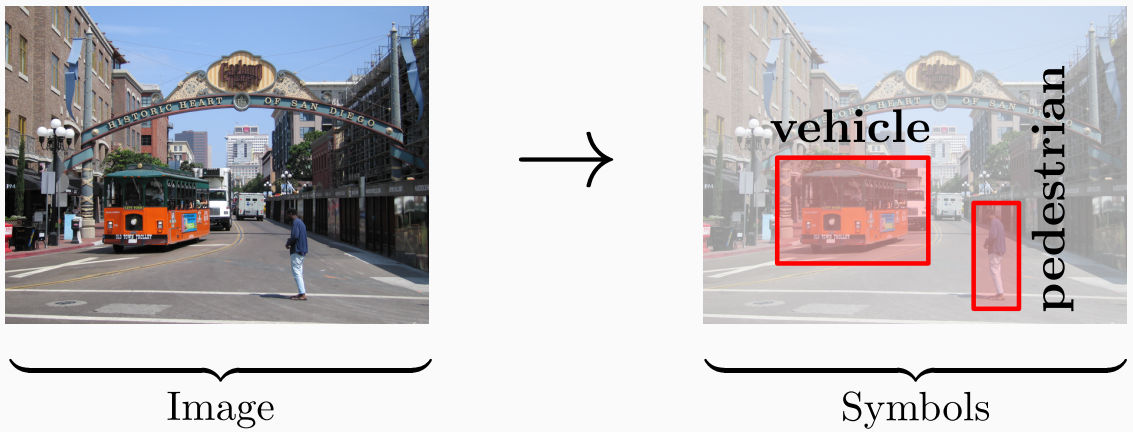


Symbols

Computer vision – Artificial Intelligence – Machine Learning

Definition (The British Machine Vision Association)

Computer vision (CV) is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images.



CV is a subfield of Artificial Intelligence.

Definition (Oxford dictionary)

Artificial Intelligence, *noun*: the theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation.

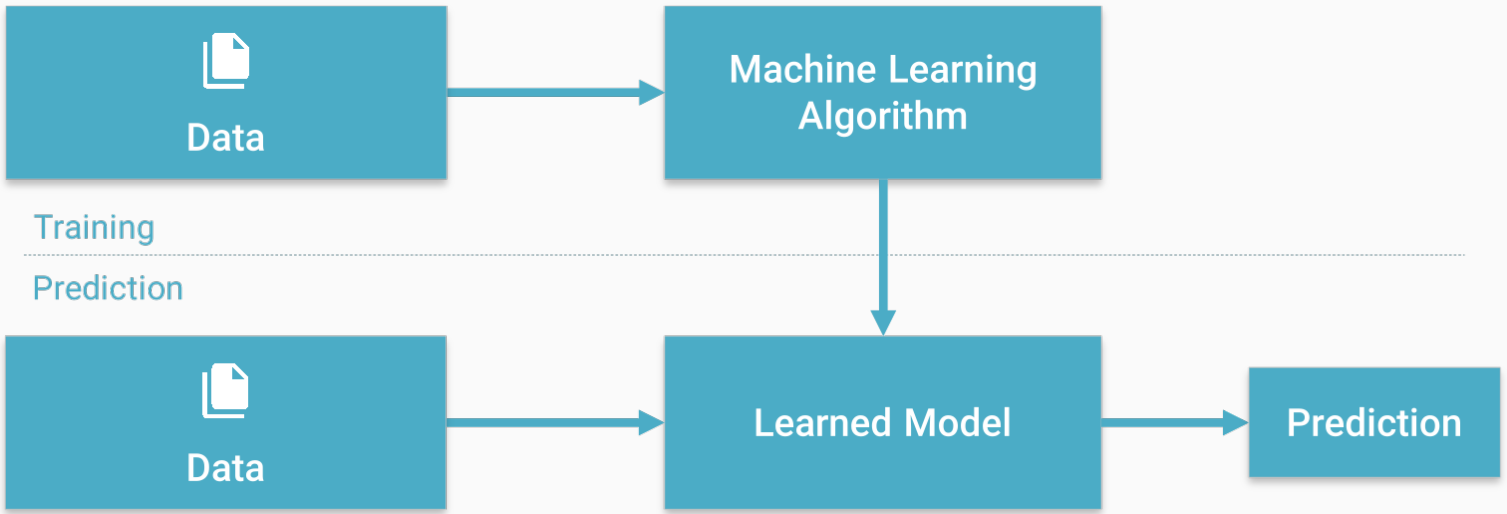
Computer vision – Artificial Intelligence – Machine Learning

CV is a subfield of AI, CV's new very best friend is machine learning (ML), ML is also a subfield of AI, but not all computer vision algorithms are ML.

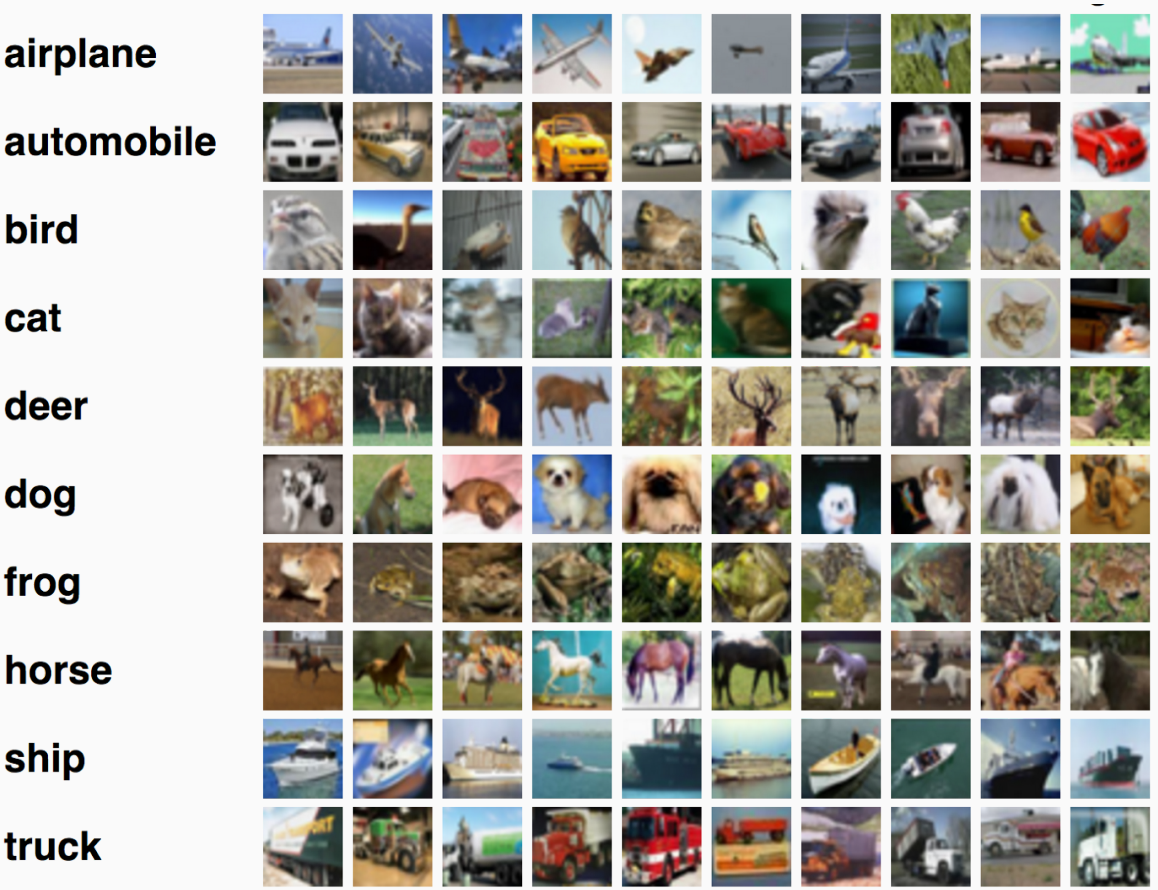
Definition

Machine Learning, *noun*: type of Artificial Intelligence that provides computers with the ability to learn without being explicitly programmed.

ML provides various techniques that can learn from and make predictions on data. Most of them follow the same general structure:

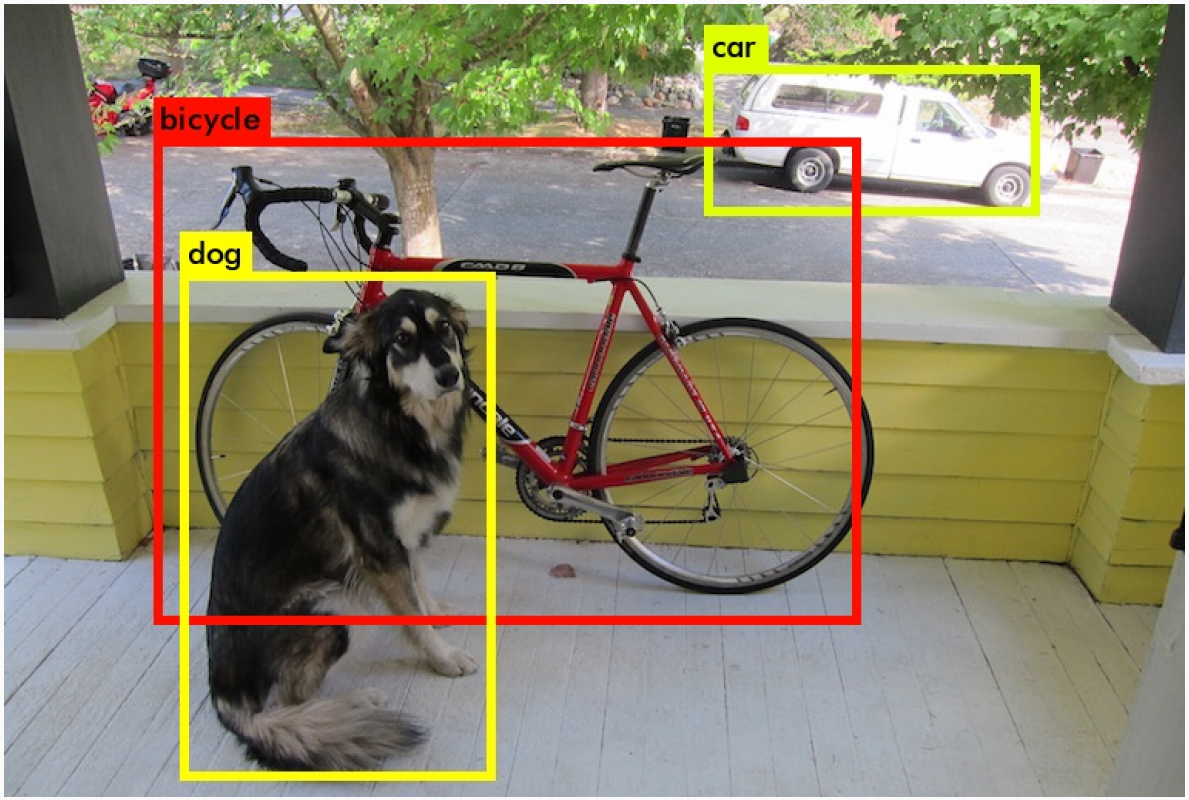


Computer vision – Image classification



Goal: to assign a given image into one of the predefined classes.

Computer vision – Object detection



(Source: Joseph Redmon)

Goal: to detect instances of objects of a certain class (such as human).

Computer vision – Image segmentation

(Source: Abhijit Kundu)

Goal: to partition an image into multiple segments such that pixels in a same segment share certain characteristics (color, texture or semantic).

Computer vision – Image captioning

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

"little girl is eating piece of cake."

"baseball player is throwing ball in game."

"woman is holding bunch of bananas."

"black cat is sitting on top of suitcase."

(Karpathy, Fei-Fei, CVPR, 2015)

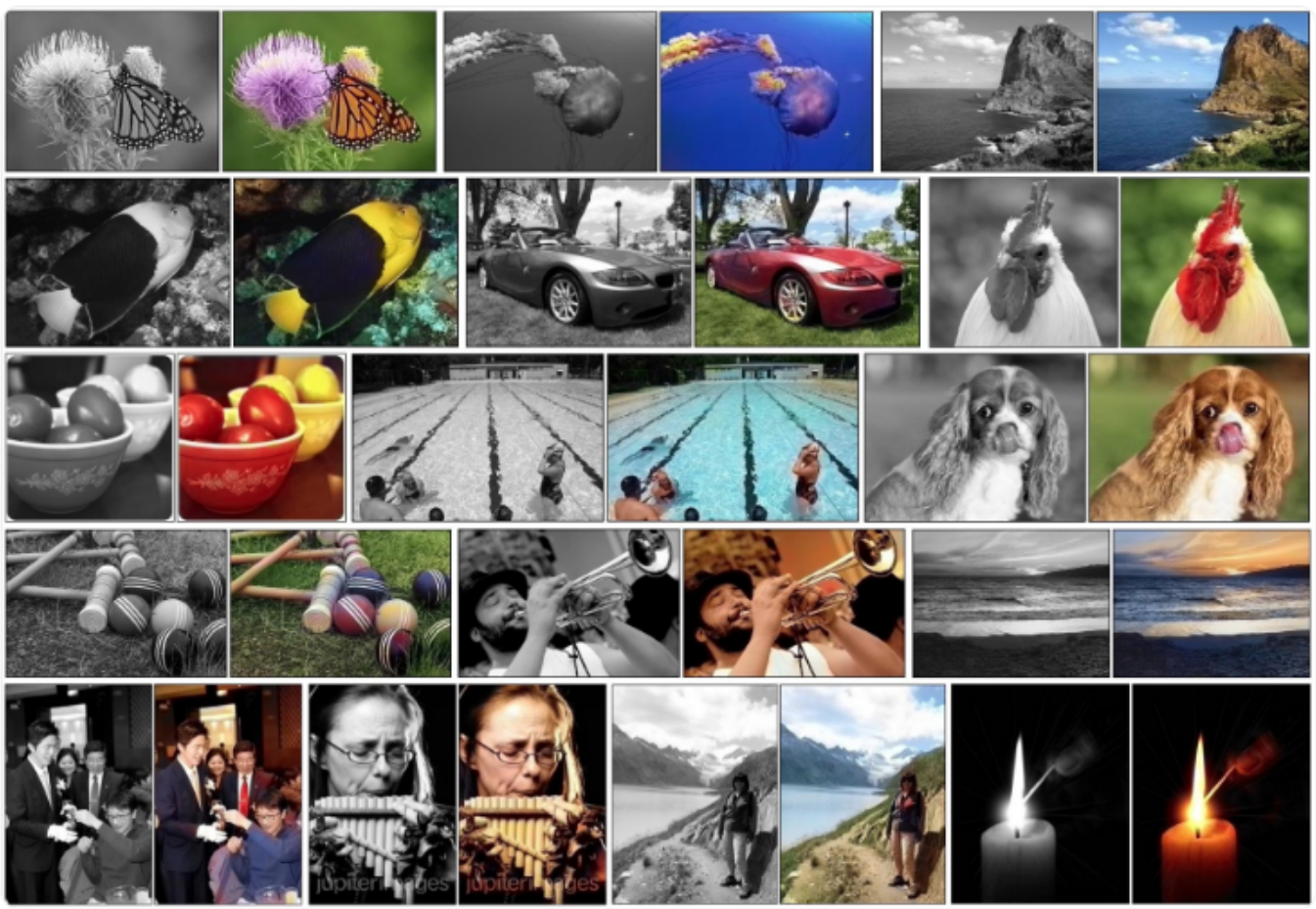
Goal: to write a sentence that describes what is happening.

Computer vision – Depth estimation

(Stereo-vision: from two images acquired with different views.)

Goal: to estimate a depth map from one, two or several frames.

Image colorization



(Source: Richard Zhang, Phillip Isola and Alexei A. Efros, 2016)

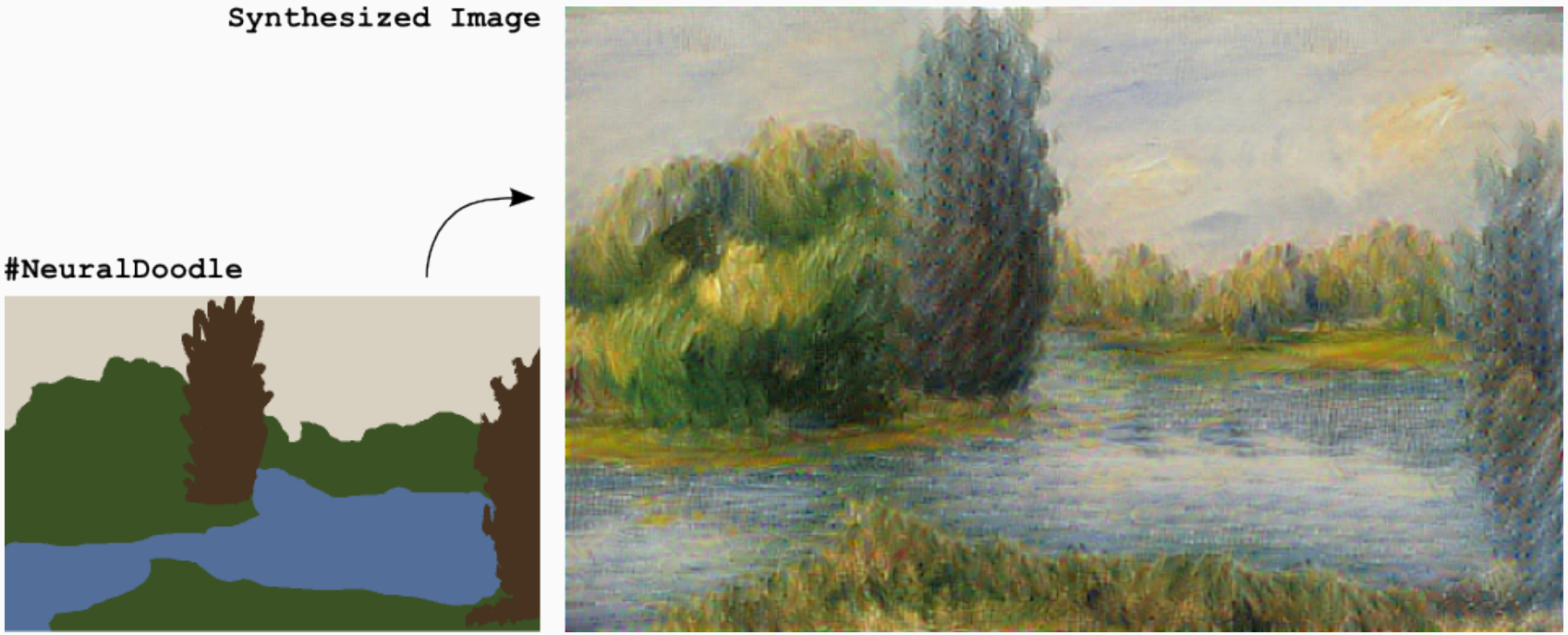
Goal: to add color to grayscale photographs.

Image generation

Generated images of bedrooms (Source: Alec Radford, Luke Metz, Soumith Chintala, 2015)

Goal: to automatically create realistic pictures of a given category.

Image stylization



(Source: Neural Doodle, Champandard, 2016)

Goal: to create stylized images from rough sketches.

Style transfer

(Source: Gatys, Ecker and Bethge, 2015)

Goal: transfer the style of an image into another one.

Learning from examples

Learning from examples

3 main ingredients

1. Training set / examples:

$$\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$$

2. Machine or model:

$$\boldsymbol{x} \rightarrow \underbrace{f(\boldsymbol{x}; \theta)}_{\text{function / algorithm}} \rightarrow \underbrace{\boldsymbol{y}}_{\text{prediction}}$$

θ: parameters of the model

3. Loss, cost, objective function / energy:

$$\underset{\theta}{\operatorname{argmin}}\ E(\theta; \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N)$$

Learning from examples

Tools:

$$\begin{cases} \text{Data} & \leftrightarrow & \text{Statistics} \\ \text{Loss} & \leftrightarrow & \text{Optimization} \end{cases}$$

Goal: to extract information from the training set

- relevant for the given task,
- relevant for other data of the same kind.

Terminology

Sample (Observation or Data): item to process (*e.g.*, classify). *Example: an individual, a document, a picture, a sound, a video. . .*

Features (Input): set of distinct traits that can be used to describe each sample in a quantitative manner. Represented as a multi-dimensional vector usually denoted by $\boldsymbol{x}$. *Example: size, weight, citizenship, . . .*

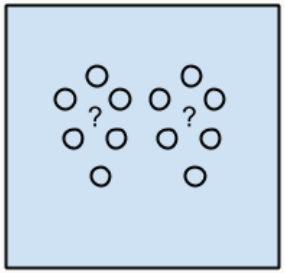
Training set: Set of data used to discover potentially predictive relationships.

Validation set: Set used to adjust the model hyperparameters.

Testing set: Set used to assess the performance of a model.

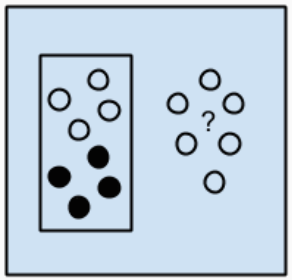
Label (Output): The class or outcome assigned to a sample. The actual prediction is often denoted by $\boldsymbol{y}$ and the desired/targeted class by $\boldsymbol{d}$ or $\boldsymbol{t}$. *Example: man/woman, wealth, education level, . . .*

Learning approaches



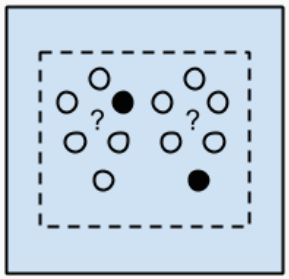
Unsupervised Learning Algorithms

Unsupervised learning: Discovering patterns in unlabeled data. *Example: cluster similar documents based on the text content.*



Supervised Learning Algorithms

Supervised learning: Learning with a labeled training set. *Example: email spam detector with training set of already labeled emails.*



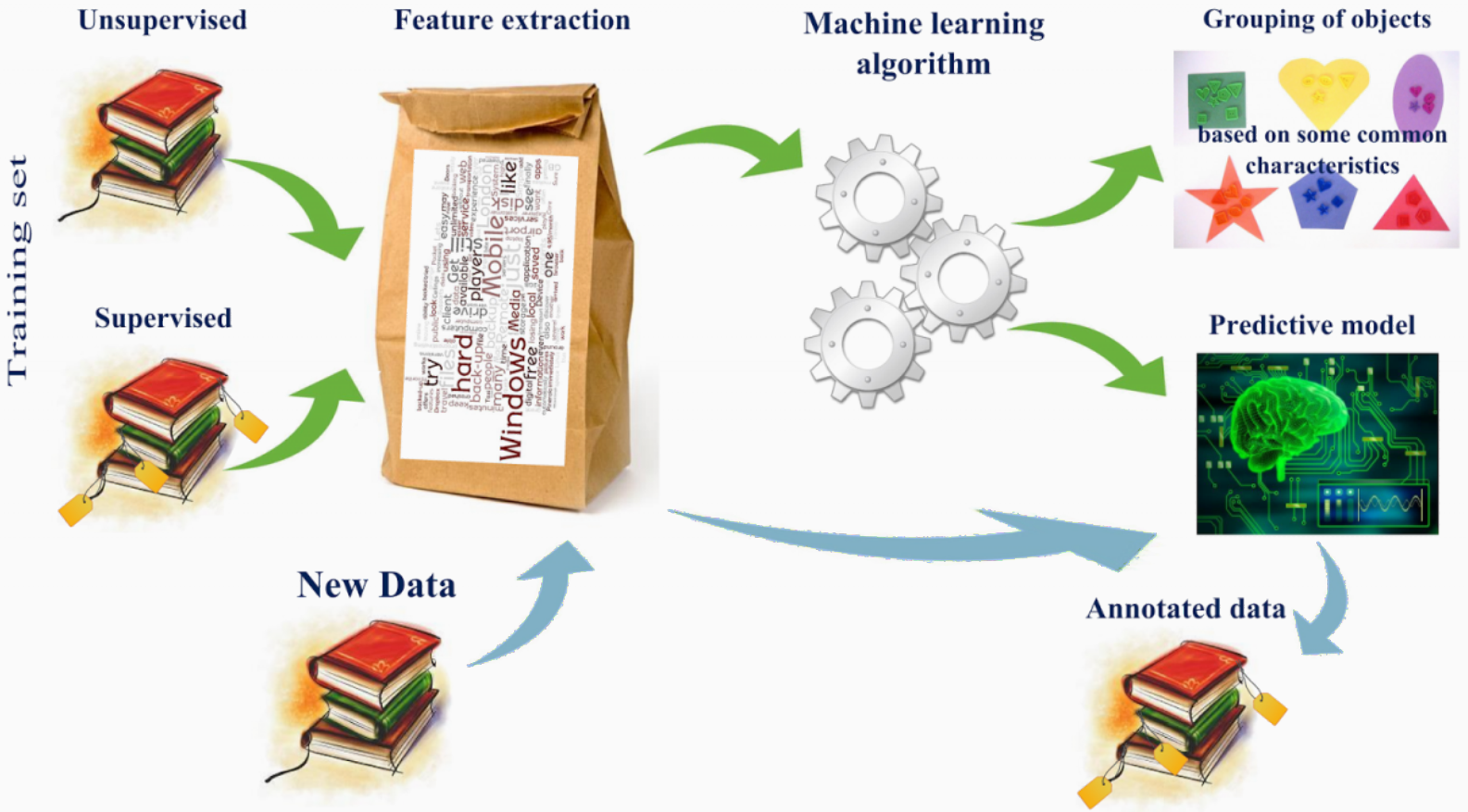
Semi-supervised Learning Algorithms

Semisupervised learning: Learning with a small amount of labeled data and a large amount of unlabeled data. *Example: web content and protein sequence classifications.*

Reinforcement learning: Learning based on feedback or reward. *Example: learn to play chess by winning or losing.*

(Source: Jason Brownlee and Lucas Masuch)

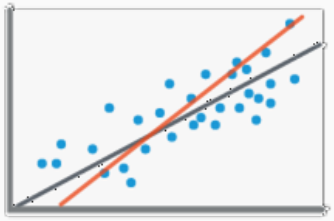
Machine learning workflow



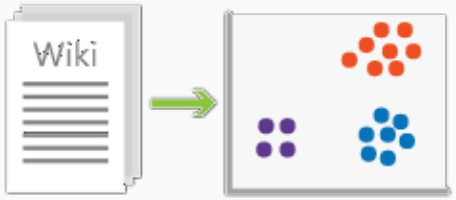
(Source: Michael Walker)

Problem types

Classification
(supervised – predictive)



Regression
(supervised – predictive)



Clustering
(unsupervised – descriptive)

Anomaly Detection
(unsupervised– descriptive)

(Source: Lucas Masuch)

What is deep learning?

- Part of the machine learning field of learning representations of data. Exceptionally effective at learning patterns.
- Utilizes learning algorithms that derive meaning out of data by using a hierarchy of multiple layers that mimic the neural networks of our brain.
- If you provide the system tons of information, it begins to understand it and respond in useful ways.
- Rebirth of artificial neural networks.

(Source: Lucas Masuch)

Deep learning: Academic actors

- Popularized by Hinton in 2006 with Restricted Boltzmann Machines

Geoffrey Hinton: University of Toronto & Google

- Developed by different actors:

Yann LeCun: New York University & Facebook

Andrew Ng: Stanford & Baidu

Yoshua Bengio: University of Montreal

Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

and many others...

Deep learning: Academic actors

- Popularized by Hinton in 2006 with Restricted Boltzmann Machines

Geoffrey Hinton: University of Toronto & Google

- Developed by different actors:

Yann LeCun: New York University & Facebook

Andrew Ng: Stanford & Baidu

Yoshua Bengio: University of Montreal

Jürgen Schmidhuber: Swiss AI Lab & NNAISENSE

and many others...

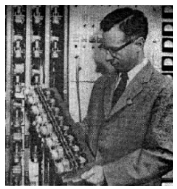
- Yoshua Bengio, Geoffrey Hinton, and Yann LeCun recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.

Actors and applications

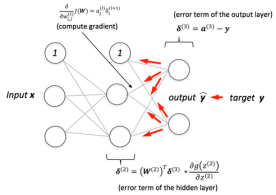
- Very active technology adopted by big actors

- Success story for many different academic problems
  - Image processing
  - Computer vision
  - Speech recognition
  - Natural language processing
  - Translation
  - etc
- Today all industries wonder if DL can improve their process.

Timeline of (deep) learning



1958 Perceptron



1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition

1998

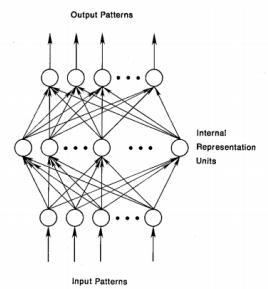
Google Brain Project on 16k Cores

2012

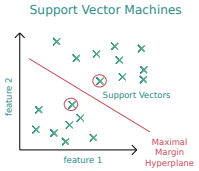
awkward silence (AI winter)

1969 Perceptrons book

~1980 Multilayer network



1995 SVM reigns

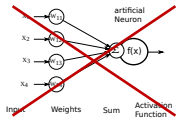


2006 Restricted Boltzmann Machine

2012 AlexNet wins ImageNet

IMAGENET

Perceptron criticized



(Source: Lucas Masuch & Vincent Lepetit)

Plan of the course

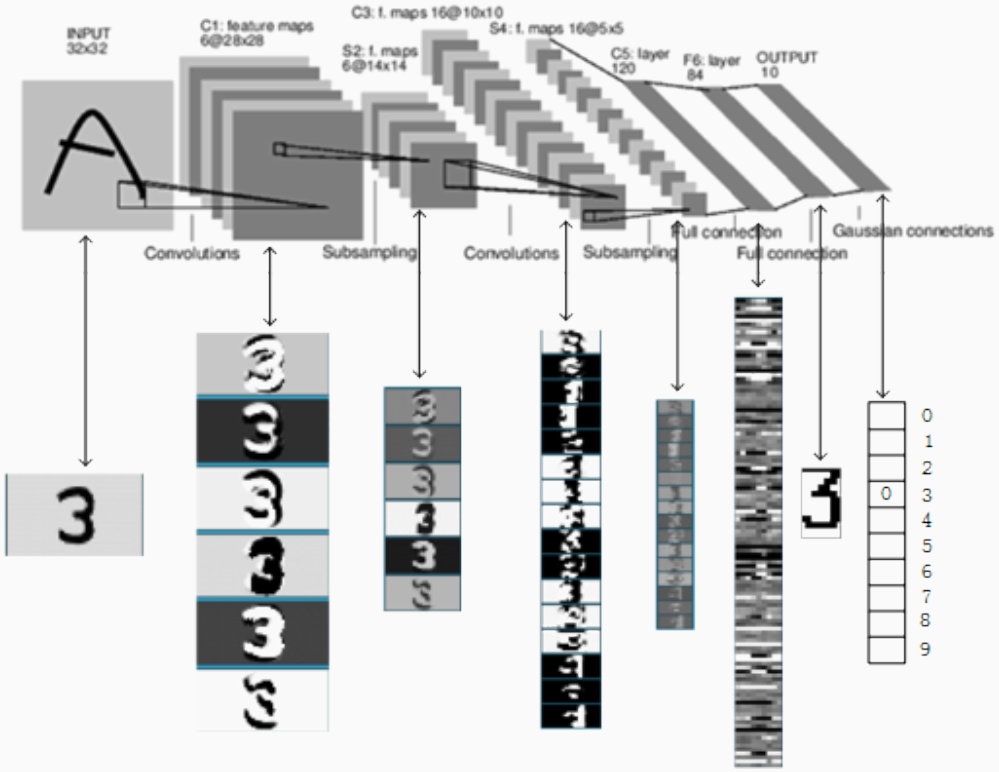
1. Introduction to neural networks
2. Convolutional neural networks for image classification
3. Convolutional neural networks for image segmentation

Software: Python + PyTorch

Remark: Focus on image processing and computer vision, but deep learning works for many other applications:

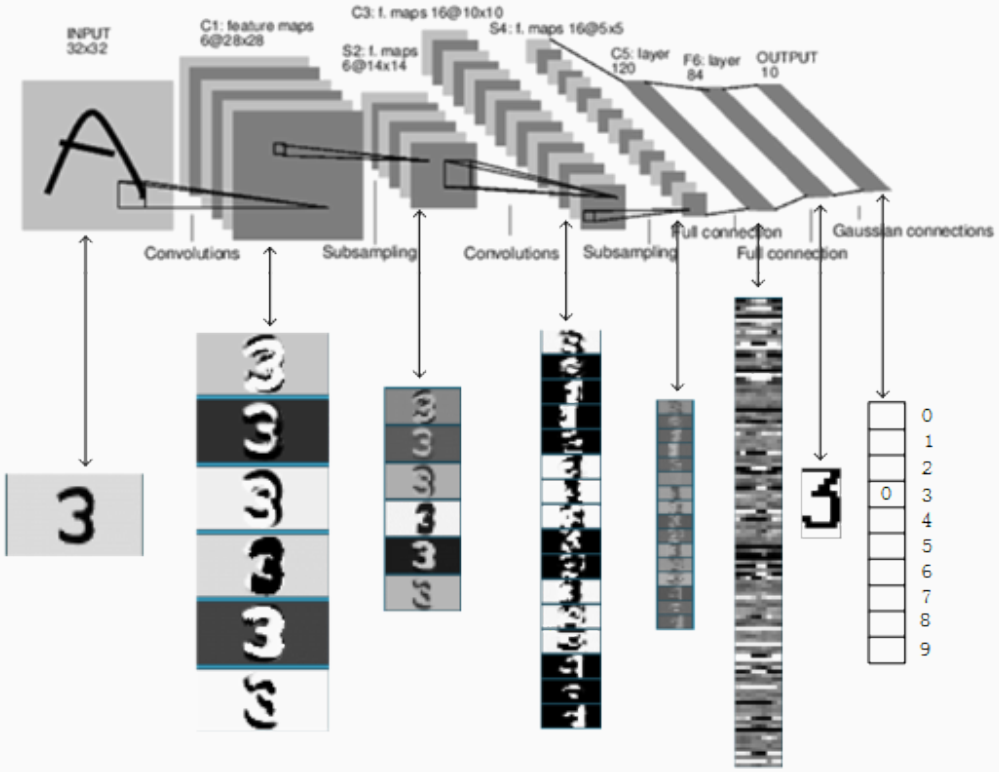
- Signal processing, speech recognition,...
- Text processing
- Graph processing (discrete geometry, social networks,...)
- Physics, chemistry,...

Neural networks for image classification



- **Goal:** Train a convolutional neural network for image classification

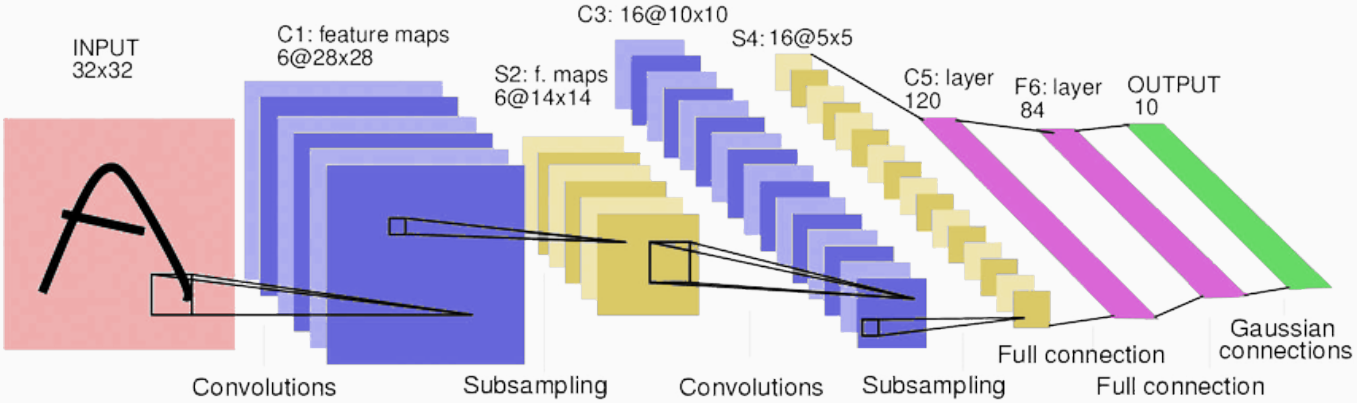
Neural networks for image classification



- **Goal:** Train a convolutional neural network for image classification
- **Goal: Understand the training** of a convolutional neural network for image classification

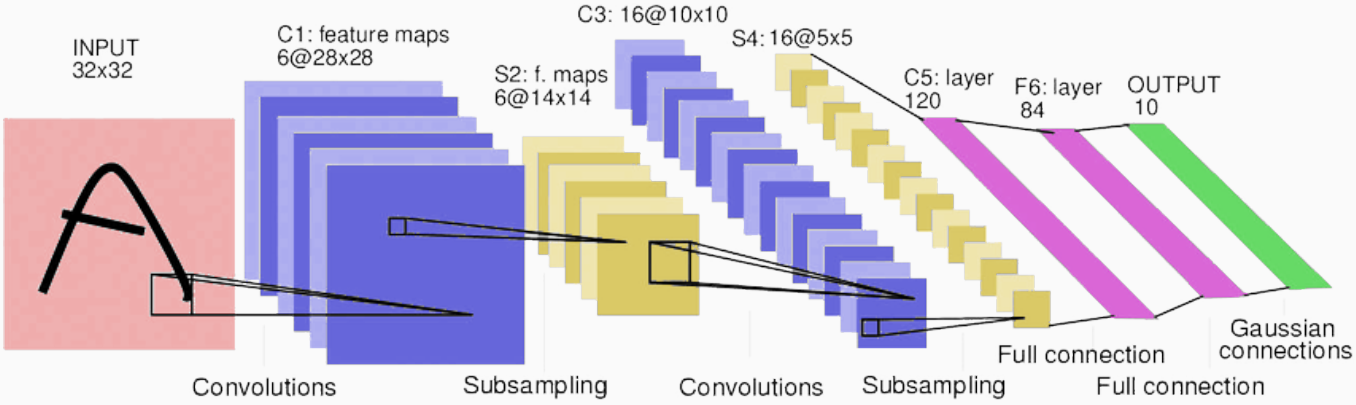
Goal of this course

Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



Goal of this course

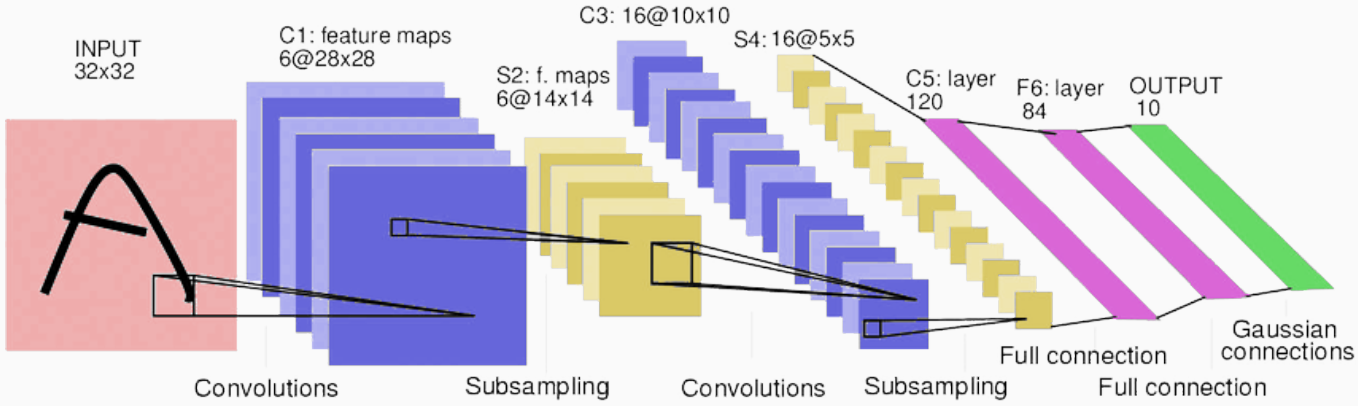
Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.

Goal of this course

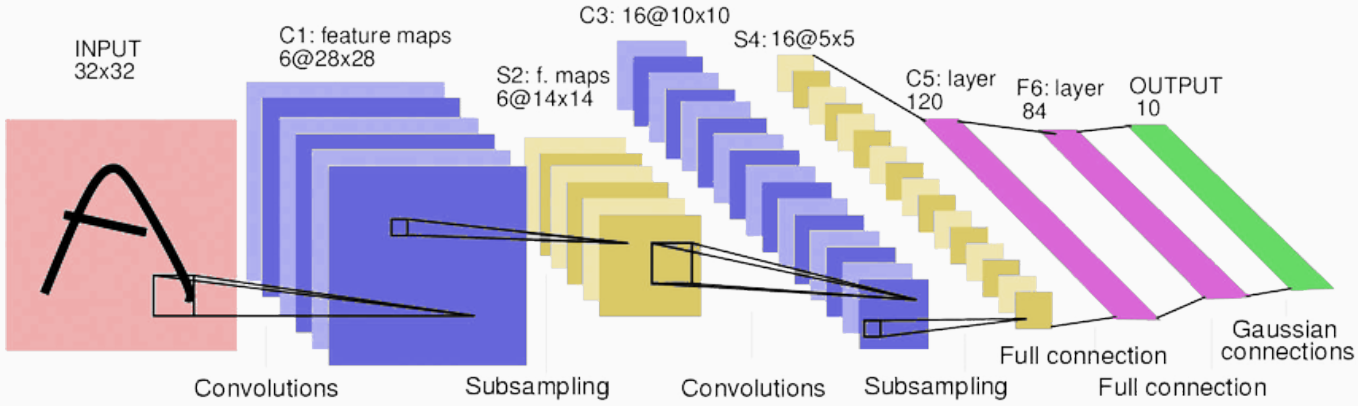
Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.

Goal of this course

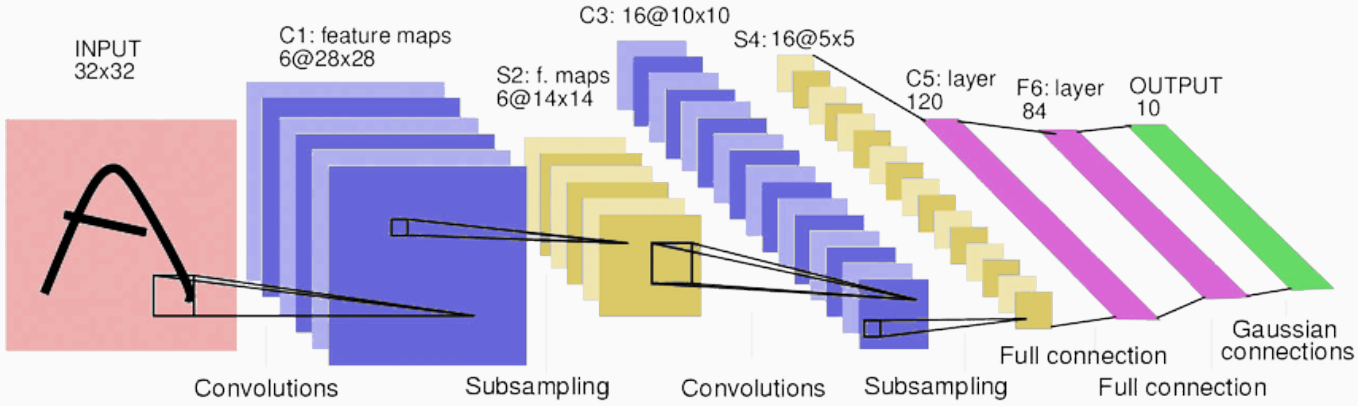
Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done by optimizing a **classification loss** $L(W)$ on a training dataset: Typically this is **a linear classifier using cross-entropy**.

Goal of this course

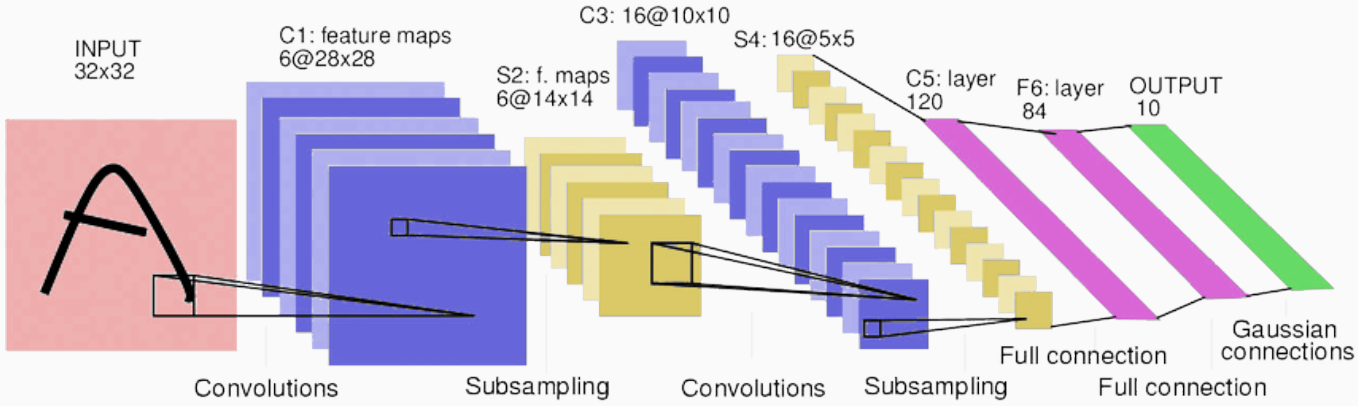
Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...



- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done by optimizing a **classification loss** $L(W)$ on a training dataset: Typically this is **a linear classifier using cross-entropy**.
- The optimization of the classification loss is done using **stochastic gradient descent** on **batches of training data**.

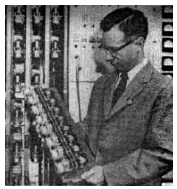
Goal of this course

Understand the training of a convolutional neural network for image classification: **A lot of notions:** going backwards...

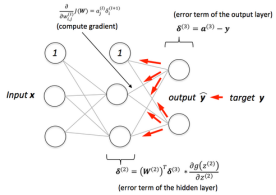


- **Convolutional neural networks:** Special neural networks for images that uses local convolutions (e.g. 3×3 filters) for the first layers.
- **Neural network:** A specific architecture to compute a classifier (or regression) having parameters=weights W to train at each layers.
- **Training** is done by optimizing a **classification loss** $L(W)$ on a training dataset: Typically this is **a linear classifier using cross-entropy**.
- The optimization of the classification loss is done using **stochastic gradient descent** on **batches of training data**.
- The gradient $\nabla L(W)$ is computed using **backpropagation**.

Timeline of (deep) learning



1958 Perceptron



1974 Backpropagation

Convolution Neural Networks for Handwritten Recognition

1998

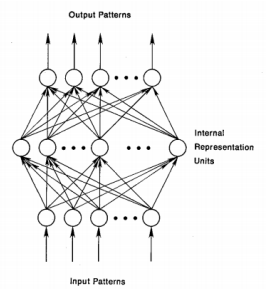
Google Brain Project on 16k Cores

2012

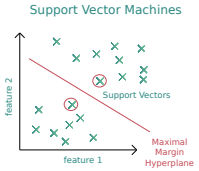
awkward silence (AI winter)

1969 Perceptrons book

~1980 Multilayer network



1995 SVM reigns

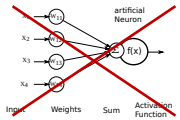


2006 Restricted Boltzmann Machine

2012 AlexNet wins ImageNet

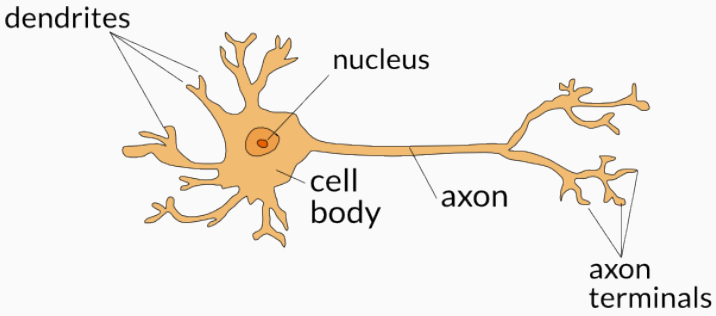
IMAGENET

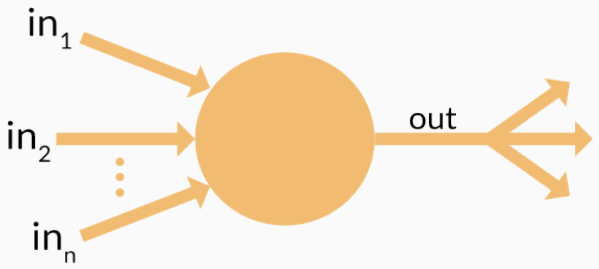
Perceptron criticized



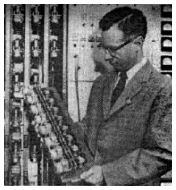
(Source: Lucas Masuch & Vincent Lepetit)

Perceptron


dendrites
nucleus
cell
body
axon
axon
terminals


in1
in2
inn
out

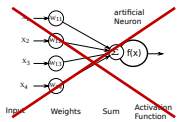
Perceptron



1958 Perceptron

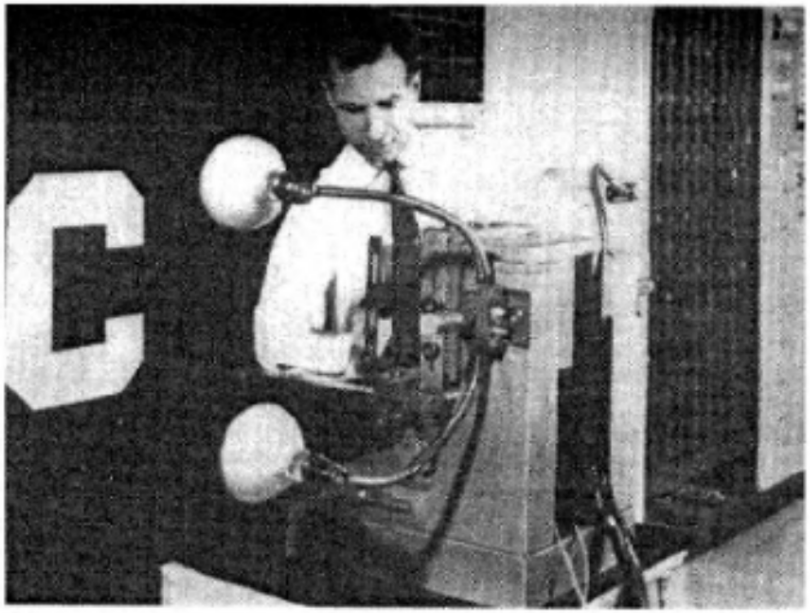
1969
Perceptrons
book

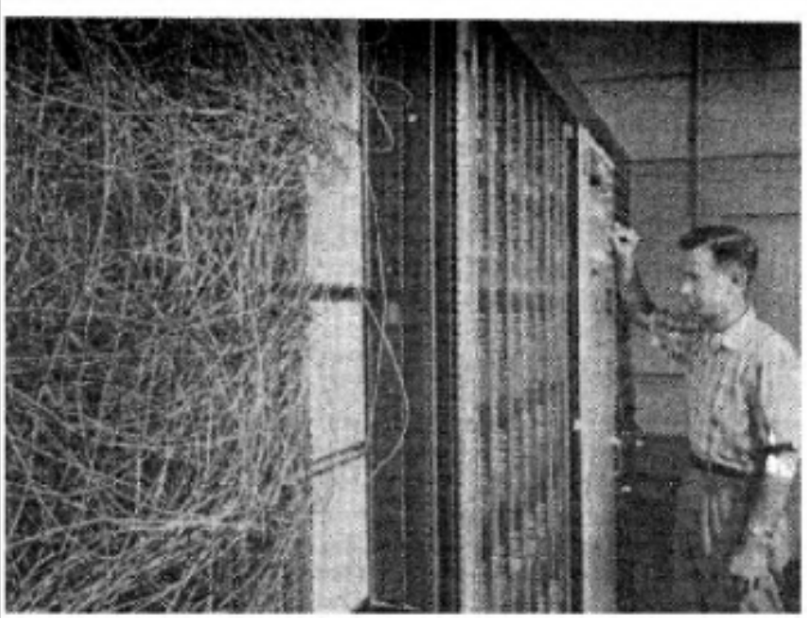
Perceptron criticized



(Source: Lucas Masuch & Vincent Lepetit)

Perceptron (Frank Rosenblatt, 1958)



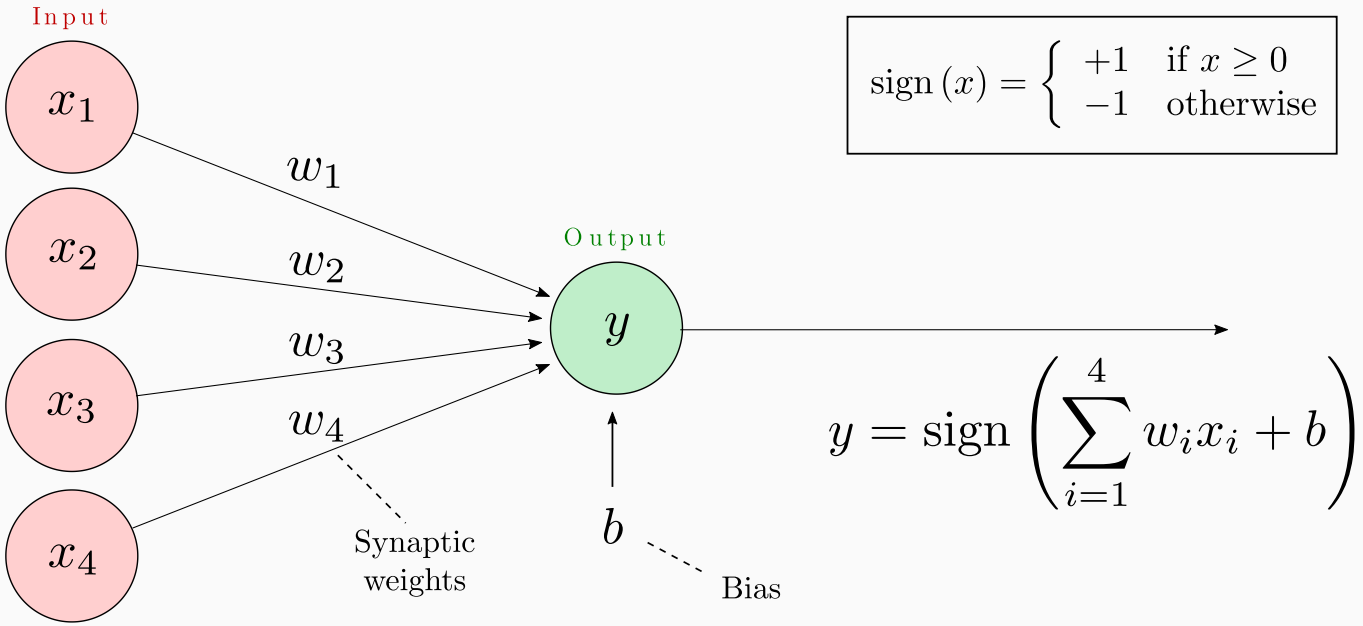


First binary classifier based on supervised learning (discrimination).

Foundation of modern artificial neural networks.

At that time: technological, scientific and philosophical challenges.

Representation of the Perceptron



$$\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$y = \text{sign}\left(\sum_{i=1}^{4} w_i x_i + b\right)$$

Parameters of the perceptron

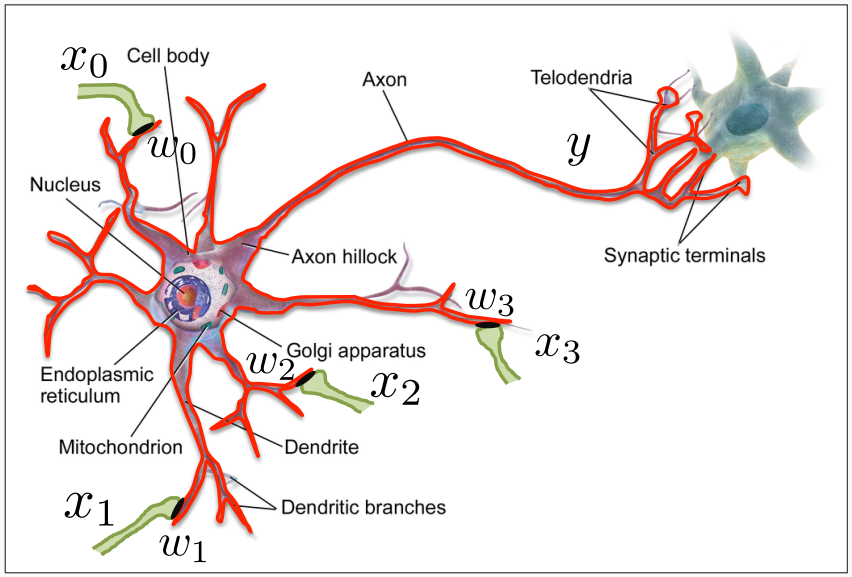
- w_k: synaptic weights
- b: bias

$\Longleftarrow$ real parameters to be estimated.

Training = adjusting the weights and biases

The origin of the Perceptron

Takes inspiration from the visual system known for its ability to learn patterns.



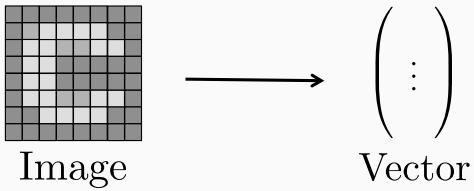
- When a neuron receives a stimulus with high enough voltage, it emits an action potential (aka, nerve impulse or spike). It is said to fire.
- The perceptron mimics this activation effect: it fires only when

$$\sum_i w_i x_i + b > 0$$

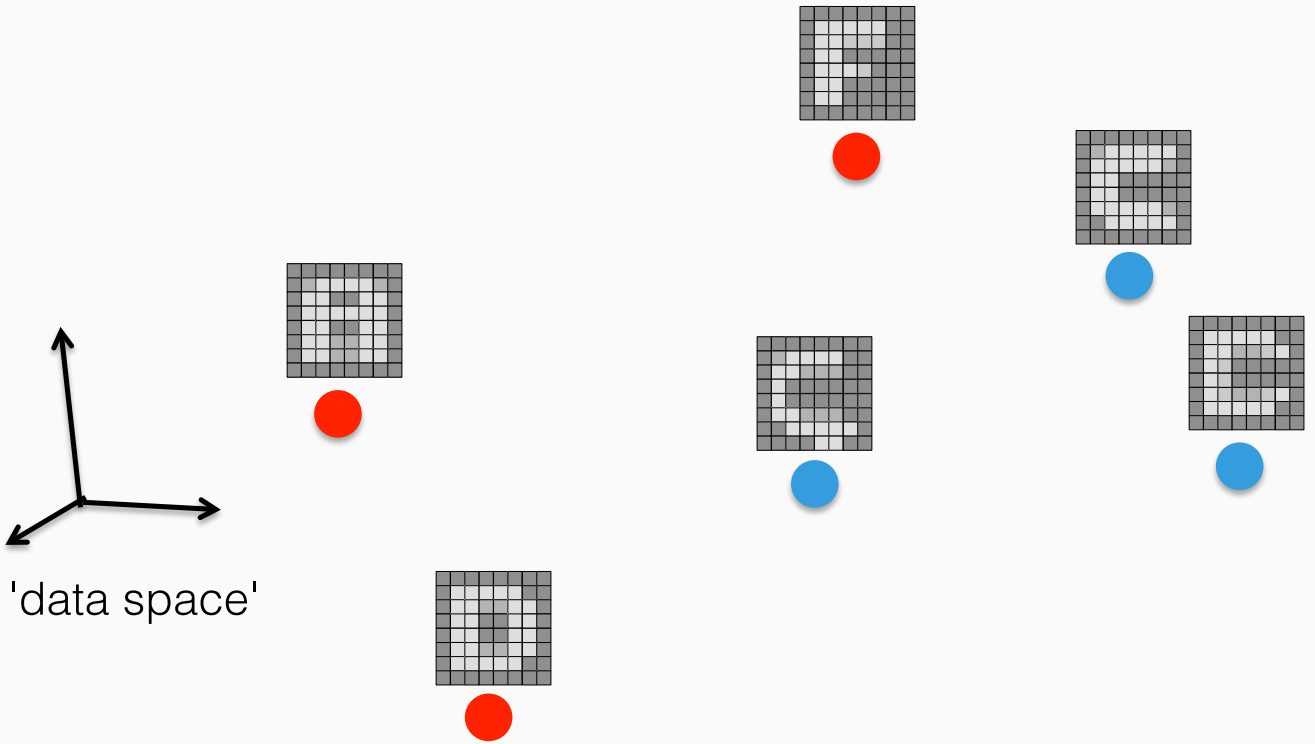
$$y = \underbrace{\operatorname{sign}(w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + b)}_{f(\boldsymbol{x};\boldsymbol{w})} = \begin{cases} +1 & \text{for the first class} \\ -1 & \text{for the second class} \end{cases}$$

Machine learning – Perceptron – Principle

1. Data are represented as vectors:



2. Collect training data with positive and negative examples:



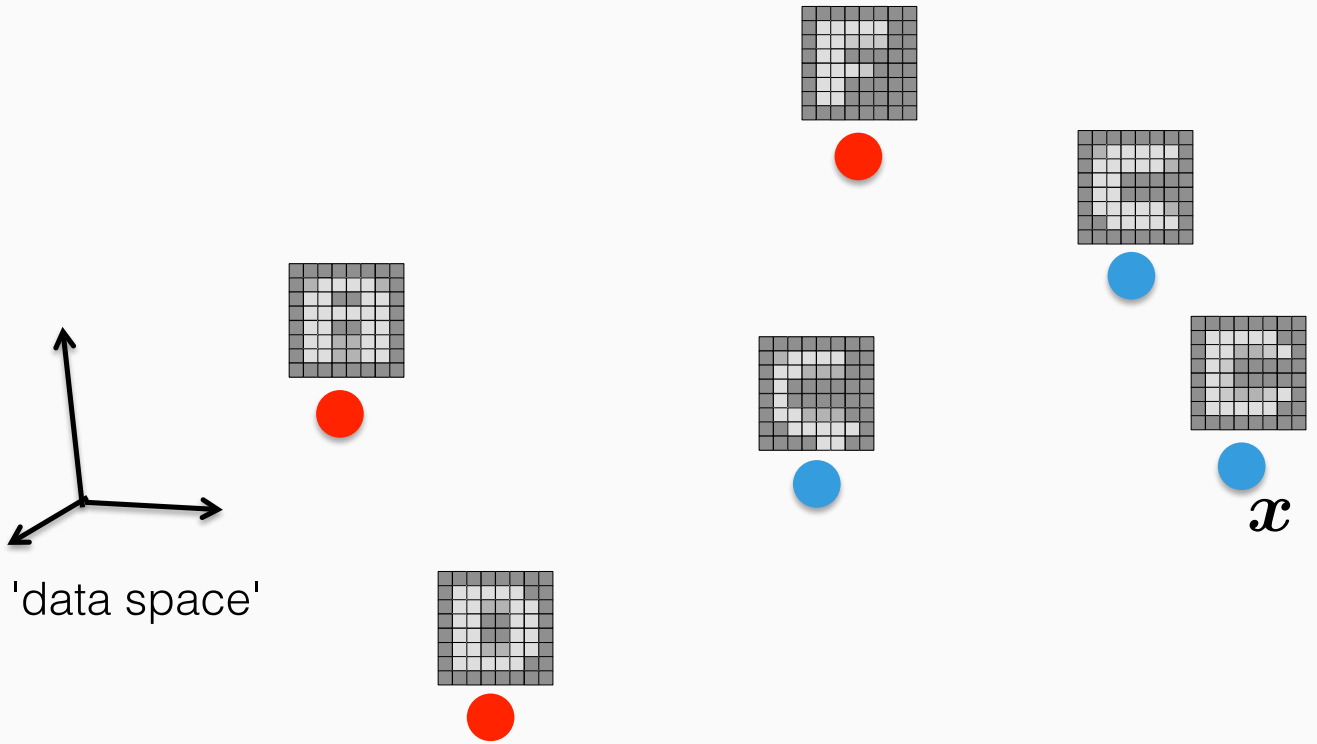
(Source: Vincent Lepetit)

❸ **Training:** find $\boldsymbol{w}$ and b so that:

- $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive for positive samples $\boldsymbol{x}$,
- $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is negative for negative samples $\boldsymbol{x}$.

Dot product:

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{i=1}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$



(Source: Vincent Lepetit)

❸ **Training:** find $\boldsymbol{w}$ and b so that:

- $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive for positive samples $\boldsymbol{x}$,
- $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is negative for negative samples $\boldsymbol{x}$.

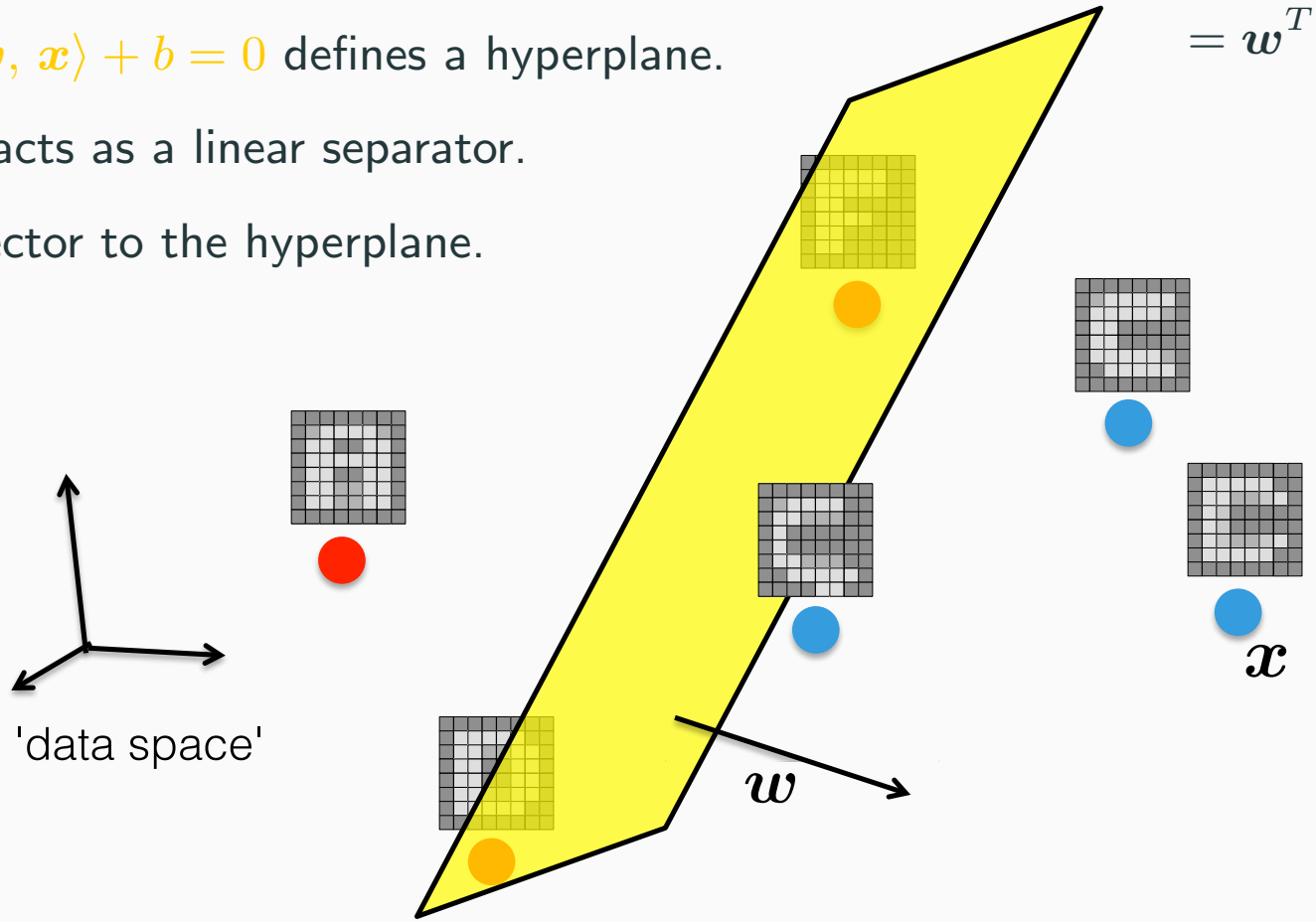
Dot product:

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{i=1}^{d} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

The equation $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0$ defines a hyperplane.

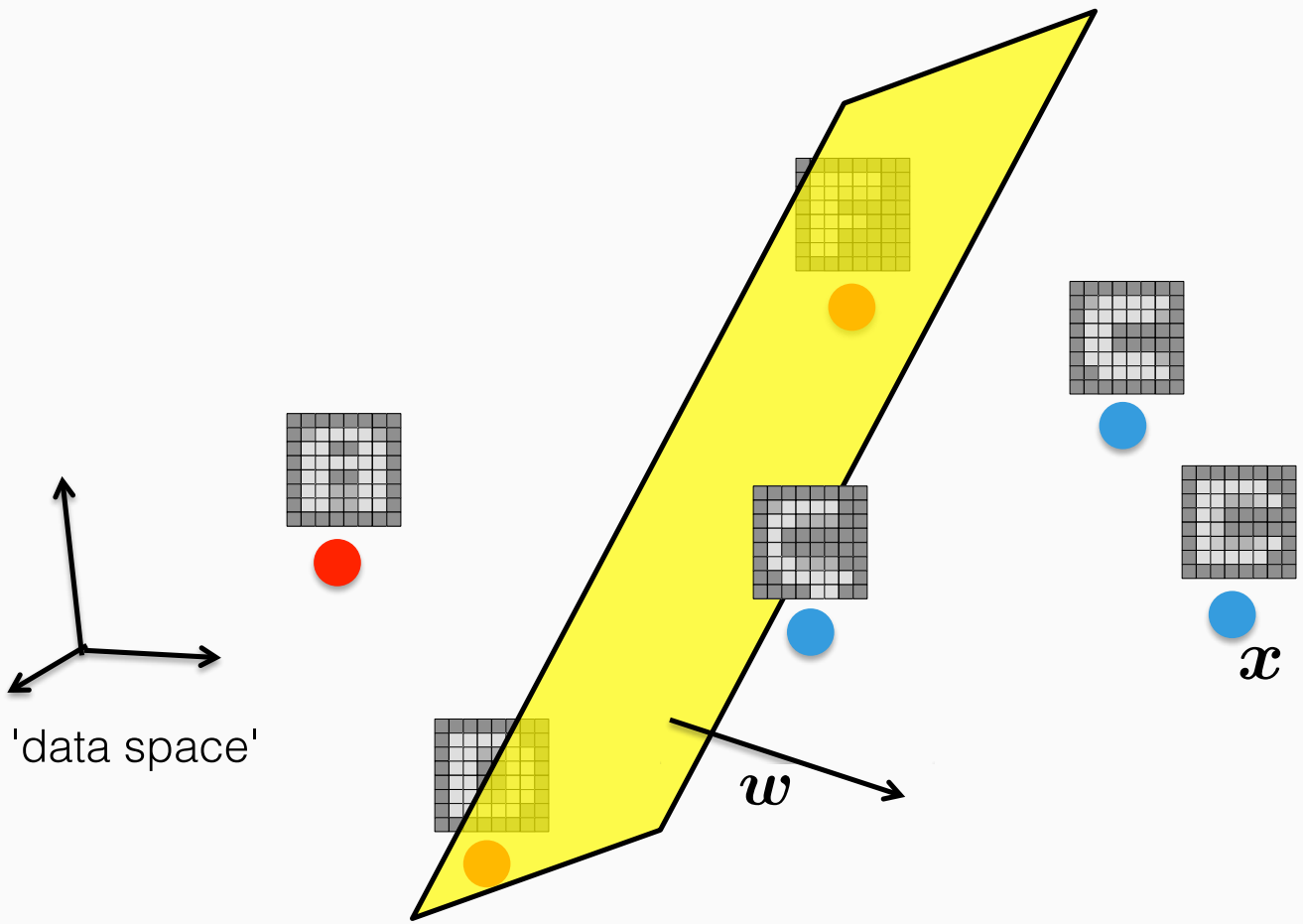
The hyperplane acts as a linear separator.

$\boldsymbol{w}$ is a normal vector to the hyperplane.



(Source: Vincent Lepetit)

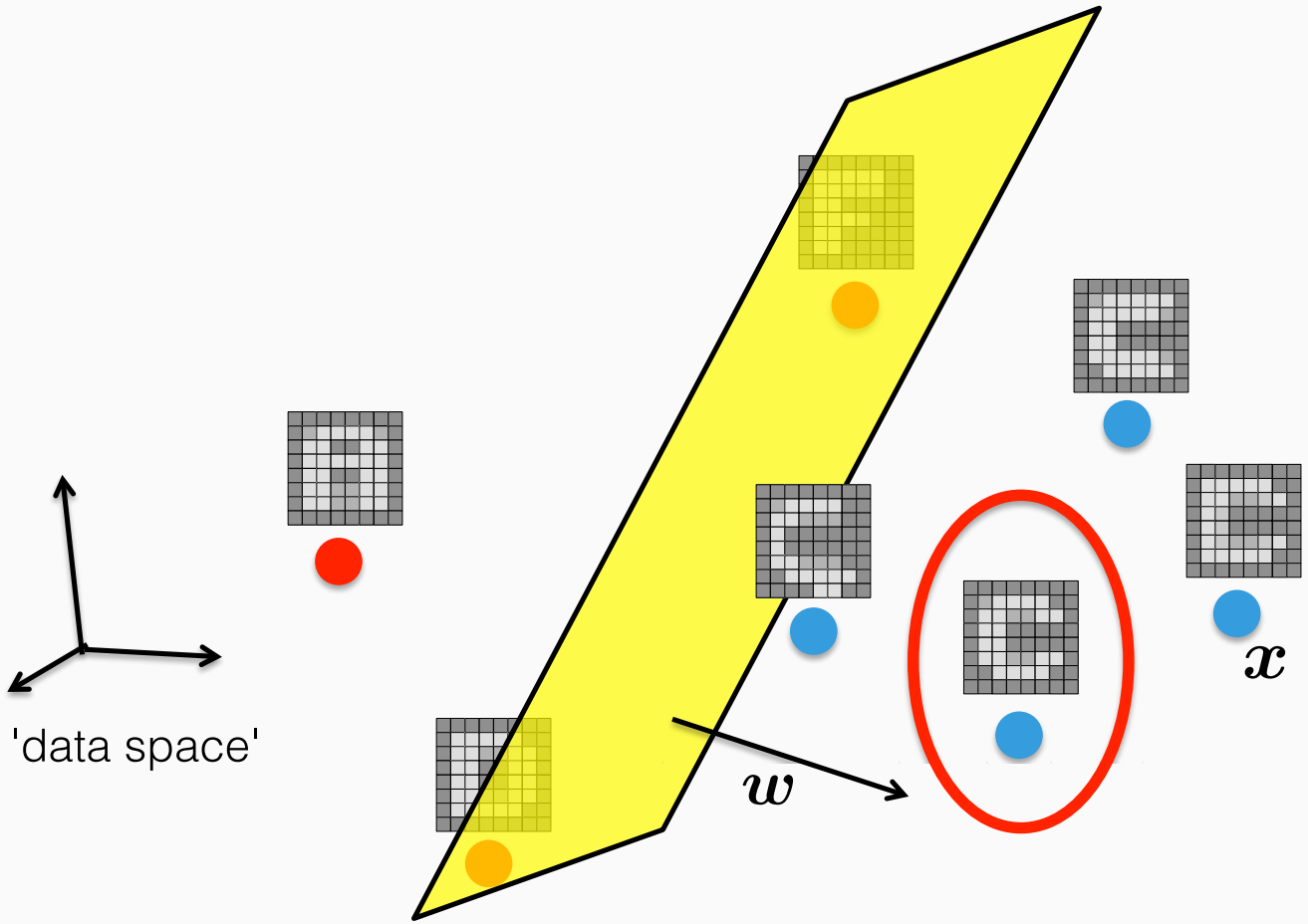
❹ **Testing:** the perceptron can now classify new examples.



(Source: Vincent Lepetit)

❹ **Testing:** the perceptron can now classify new examples.

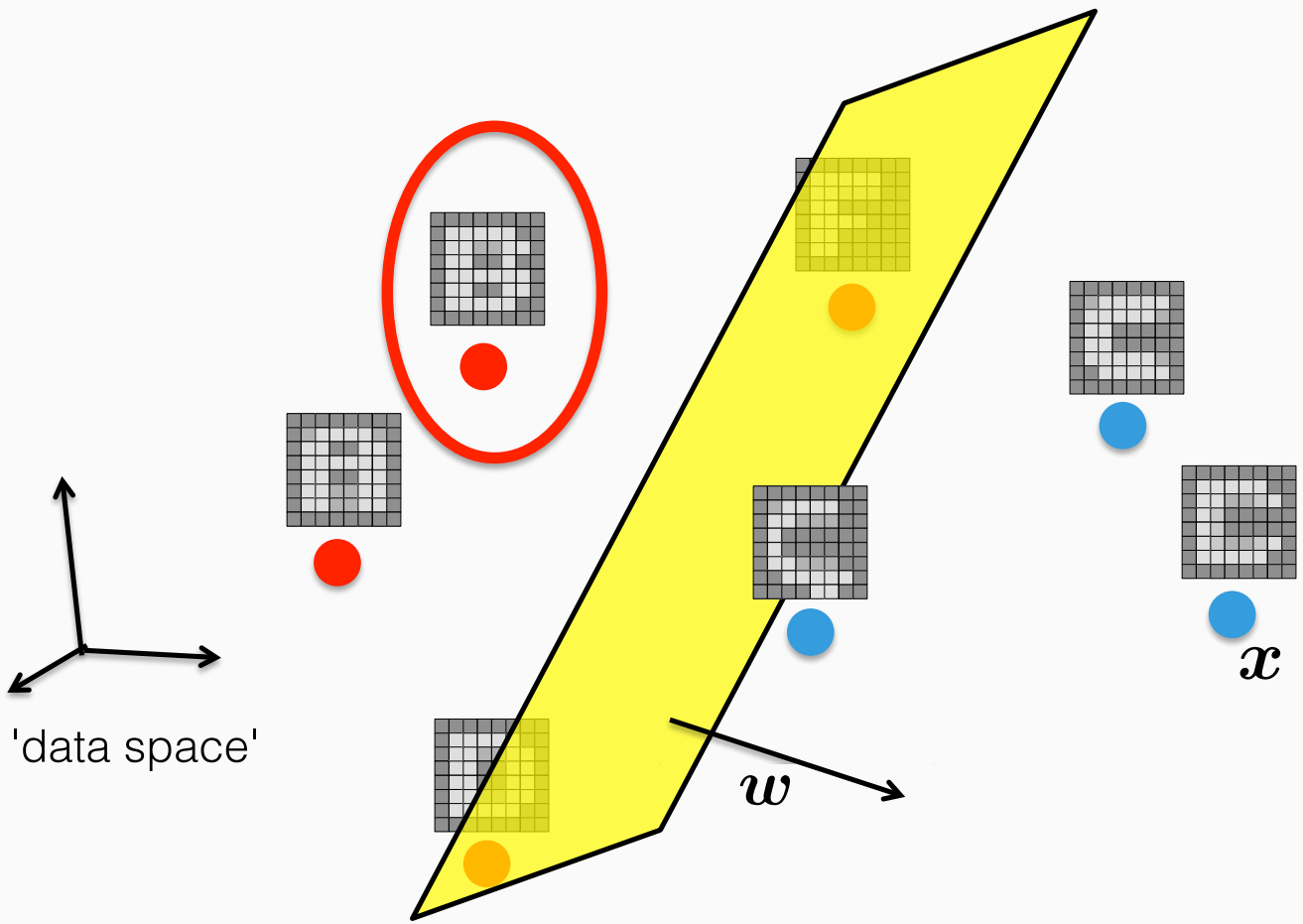
- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,



(Source: Vincent Lepetit)

❹ **Testing:** the perceptron can now classify new examples.

- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,
- and negative if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is negative.



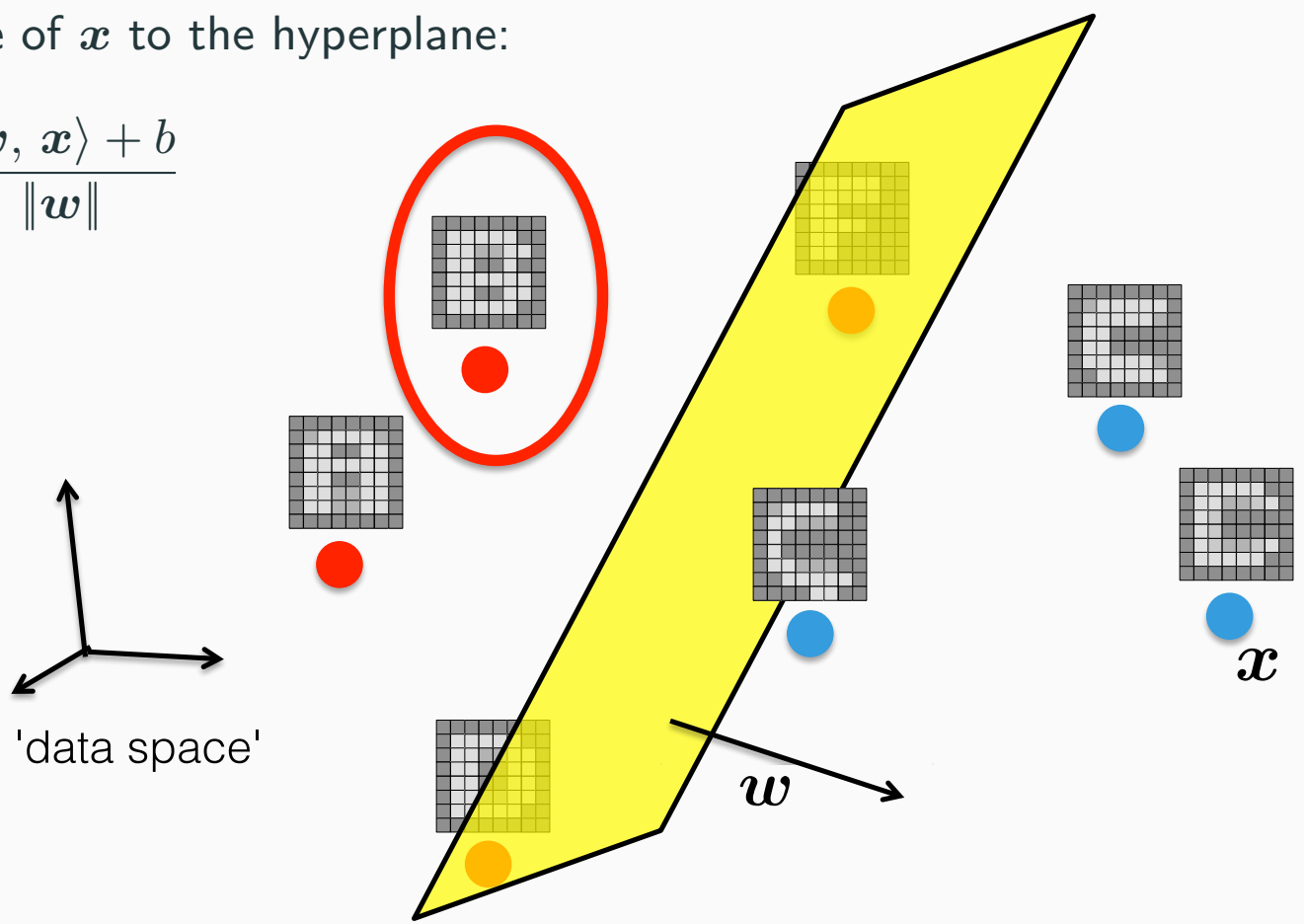
(Source: Vincent Lepetit)

❹ **Testing:** the perceptron can now classify new examples.

- A new example $\boldsymbol{x}$ is classified positive if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is positive,
- and negative if $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is negative.

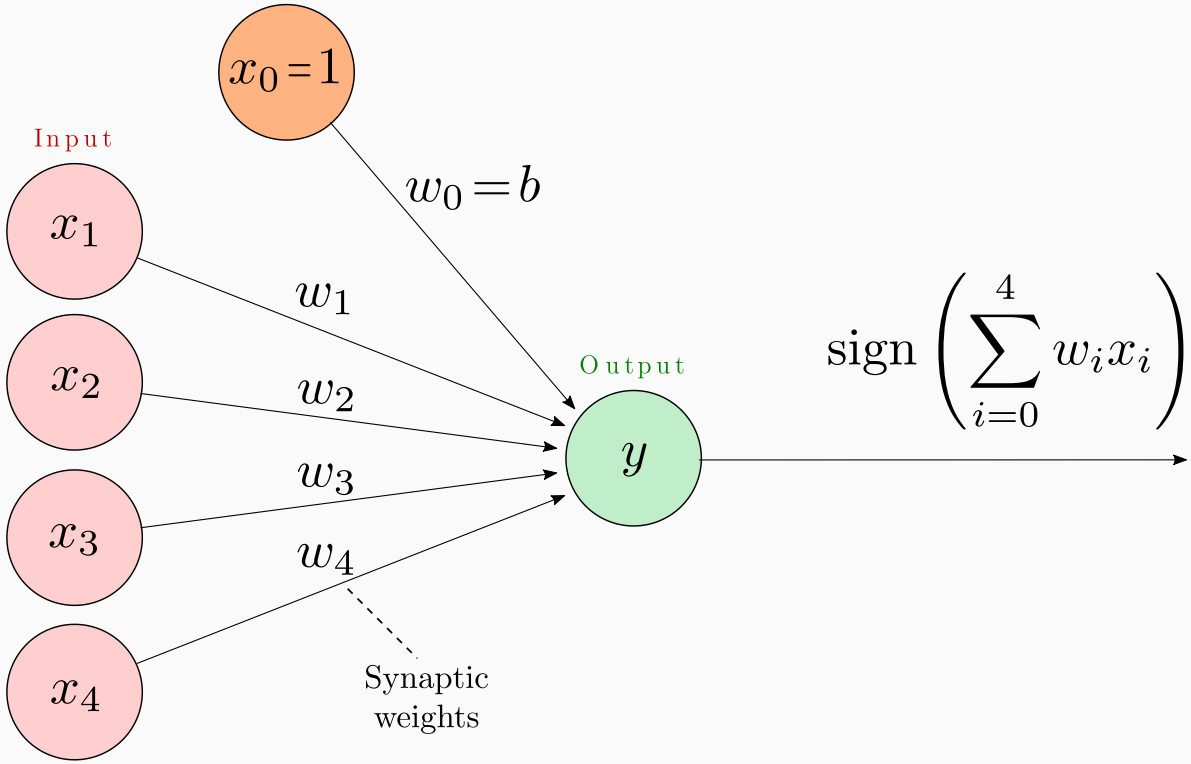
(signed) distance of $\boldsymbol{x}$ to the hyperplane:

$$r = \frac{\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b}{\|\boldsymbol{w}\|}$$



(Source: Vincent Lepetit)

Alternative representation



Use the zero-index to encode the bias as a synaptic weight.

Simplifies algorithms as all parameters can now be processed in the same way.

Perceptron algorithm

Goal: find the vector of weights $\boldsymbol{w}$ from a labeled training dataset $\mathcal{T}$

$$\mathcal{T} = \{(\boldsymbol{x}^i, d^i)\}_{i=1..N}$$

- $\boldsymbol{x}^i$: i-th training example
- d^i: desired output for sample i, $\{-1, +1\}$
- N: number of training samples

How: minimize classification errors

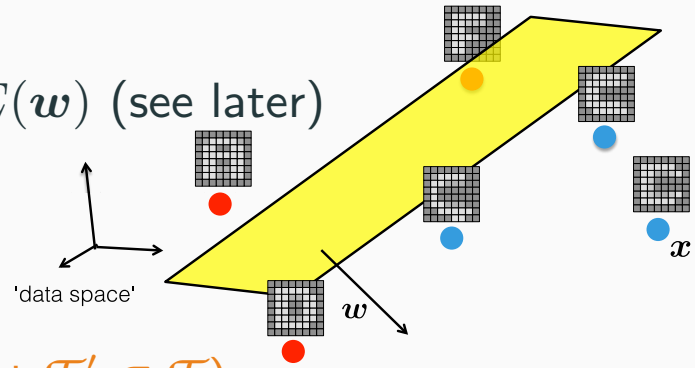
$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = -\sum_{\substack{(\boldsymbol{x},d)\in\mathcal{T} \\ \text{st } y\neq d}} d \times \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{(\boldsymbol{x},d)\in\mathcal{T}} \max(-d \times \langle \boldsymbol{w}, \boldsymbol{x} \rangle, 0)$$

- penalize only misclassified samples ($y \neq d$) for which $d \times \langle \boldsymbol{w}, \boldsymbol{x} \rangle < 0$,
- zero if all samples are correctly classified.

Perceptron algorithm

- We assume that $\max(0, t)$ is derivable with derivative 1 if $t > 0$, 0 if $t <= 0$.

Algorithm: (stochastic) gradient descent for $E(\boldsymbol{w})$ (see later)



- Initialize $\boldsymbol{w}$ randomly
- Repeat until convergence
  - For all $(\boldsymbol{x}, d) \in \mathcal{T}$ (or a random subset $\mathcal{T}' \subset \mathcal{T}$)
    - Compute: $y = \text{sign} \langle \boldsymbol{w}, \boldsymbol{x} \rangle$
    - If $y \neq d$:

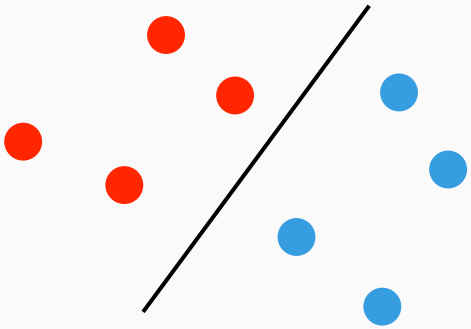
      Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \gamma d \boldsymbol{x}$

- Converges to some solution if the training data are linearly separable,
- But may pick any of many solutions of varying quality.

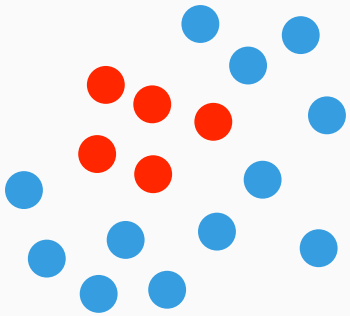
  $\Rightarrow$ Poor generalization error, compared with SVM and logistic loss.

Perceptrons book (Minsky and Papert, 1969)

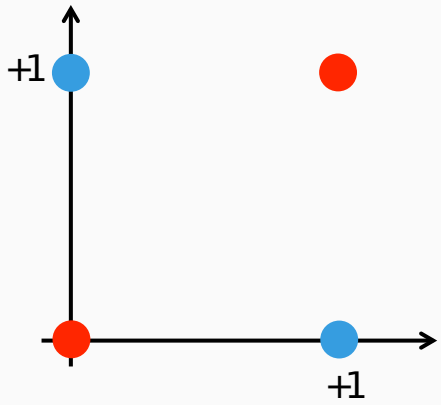
A perceptron can only classify data points that are linearly separable:



Linearly separable



Nonlinearly separable



The xor function

Seen by many as a justification to stop research on perceptrons.

(Source: Vincent Lepetit)

Artificial neural network

Artificial neural network

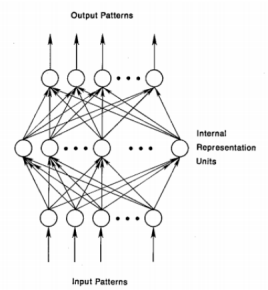
1958 Perceptron

1969
Perceptrons
book

~1980
Multilayer
network

1989
Universal
Approximation
Theorem

Perceptron criticized



(Source: Lucas Masuch & Vincent Lepetit)

Arti cial neural network

Supervised learning method initially inspired by the behavior of the human brain.

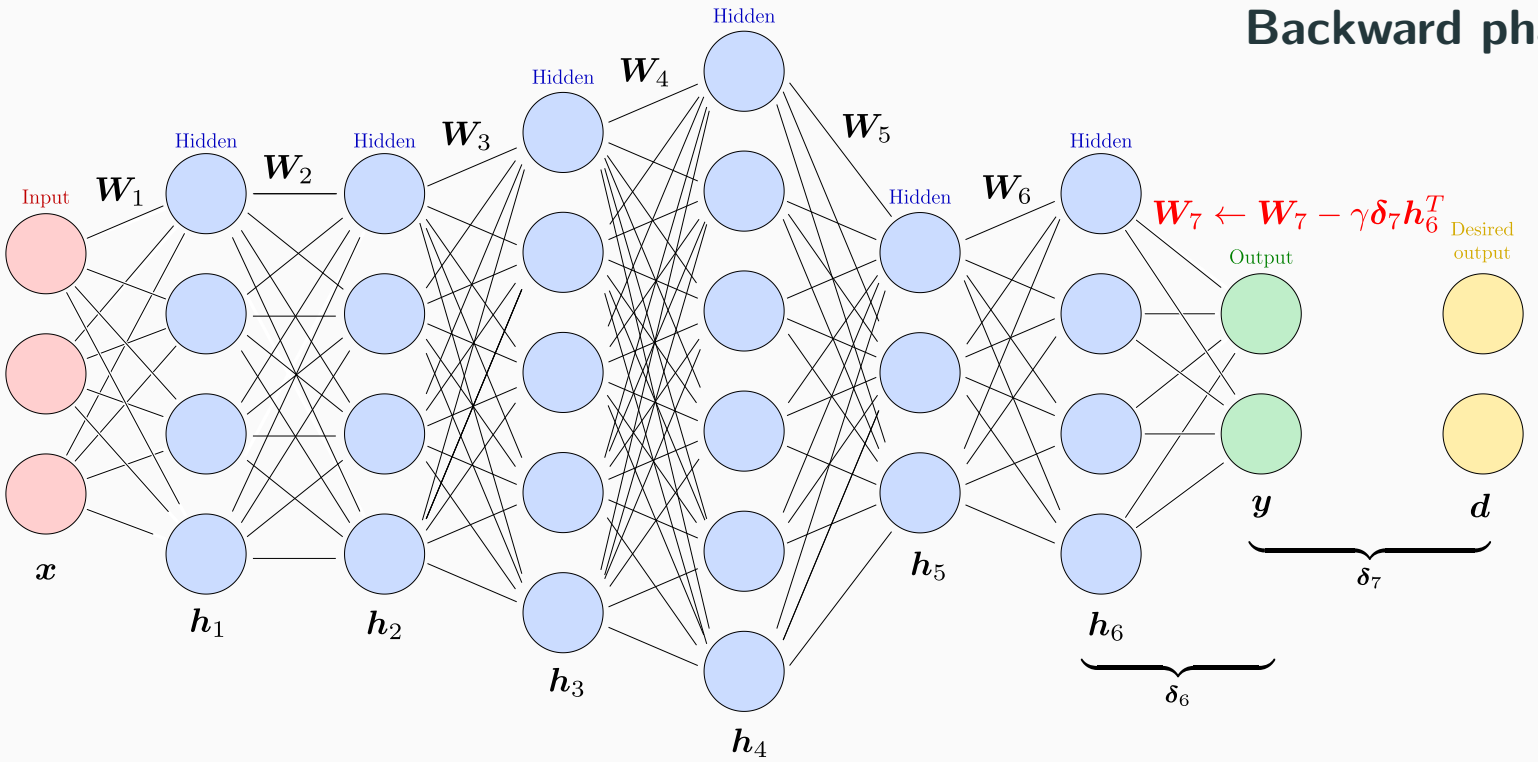
Consists of the inter-connection of several small units (just like in the human brain).

Introduced in the late 50s, very popular in the 90s, reappeared in the 2010s with deep learning.

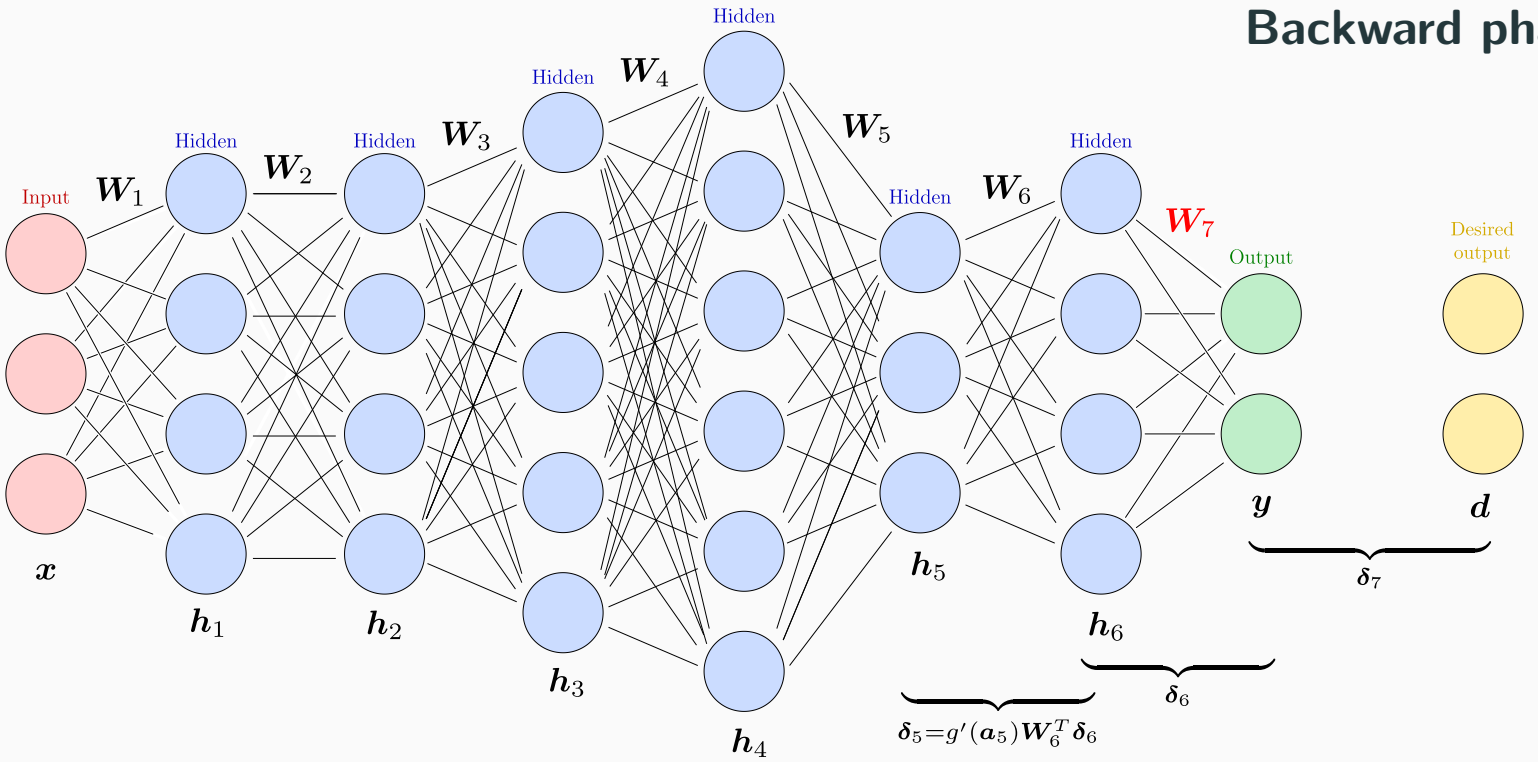
Also referred to as Multi-Layer Perceptron (MLP).

Historically used after feature extraction.

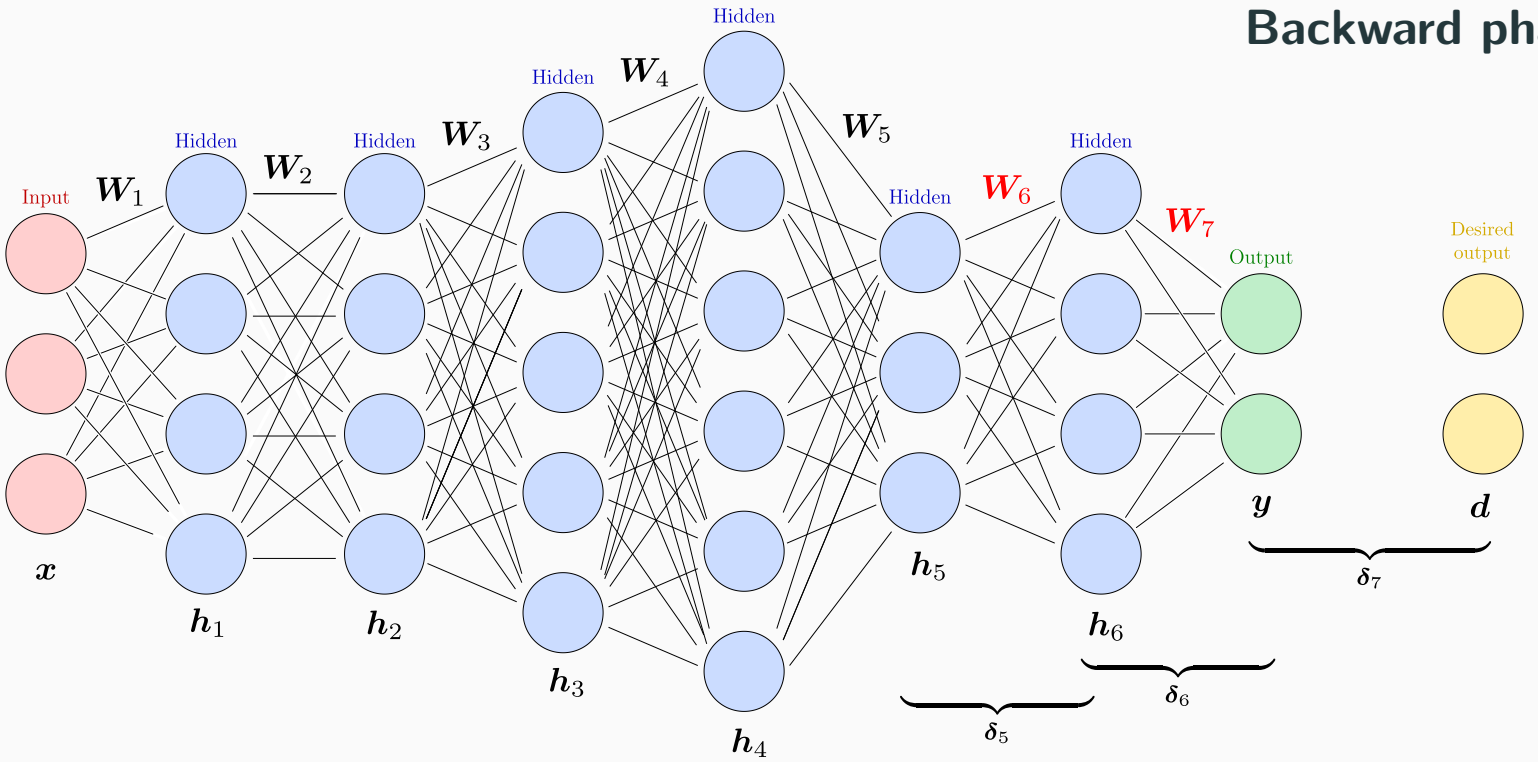
Backpropagation algorithm



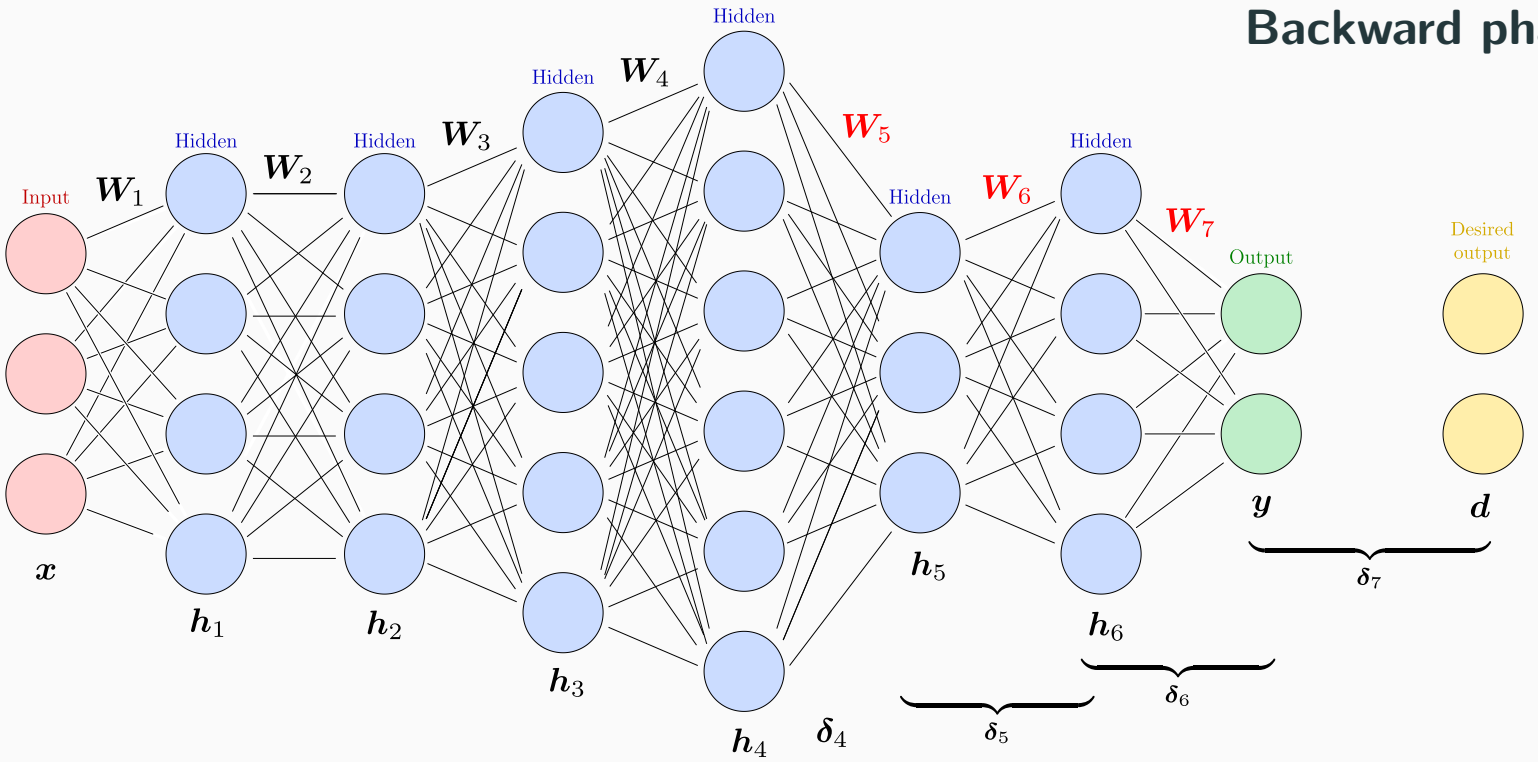
Backpropagation algorithm



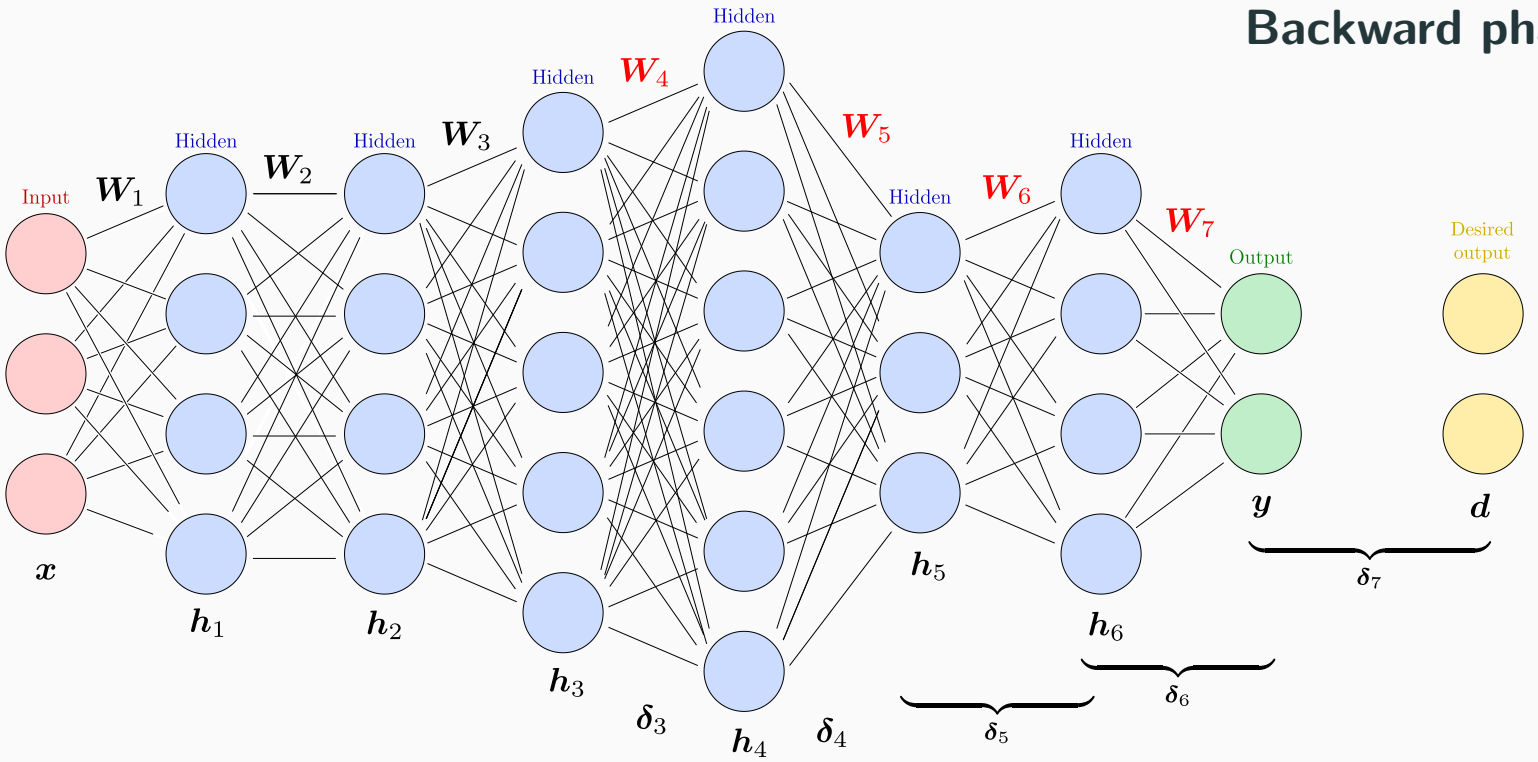
Backpropagation algorithm



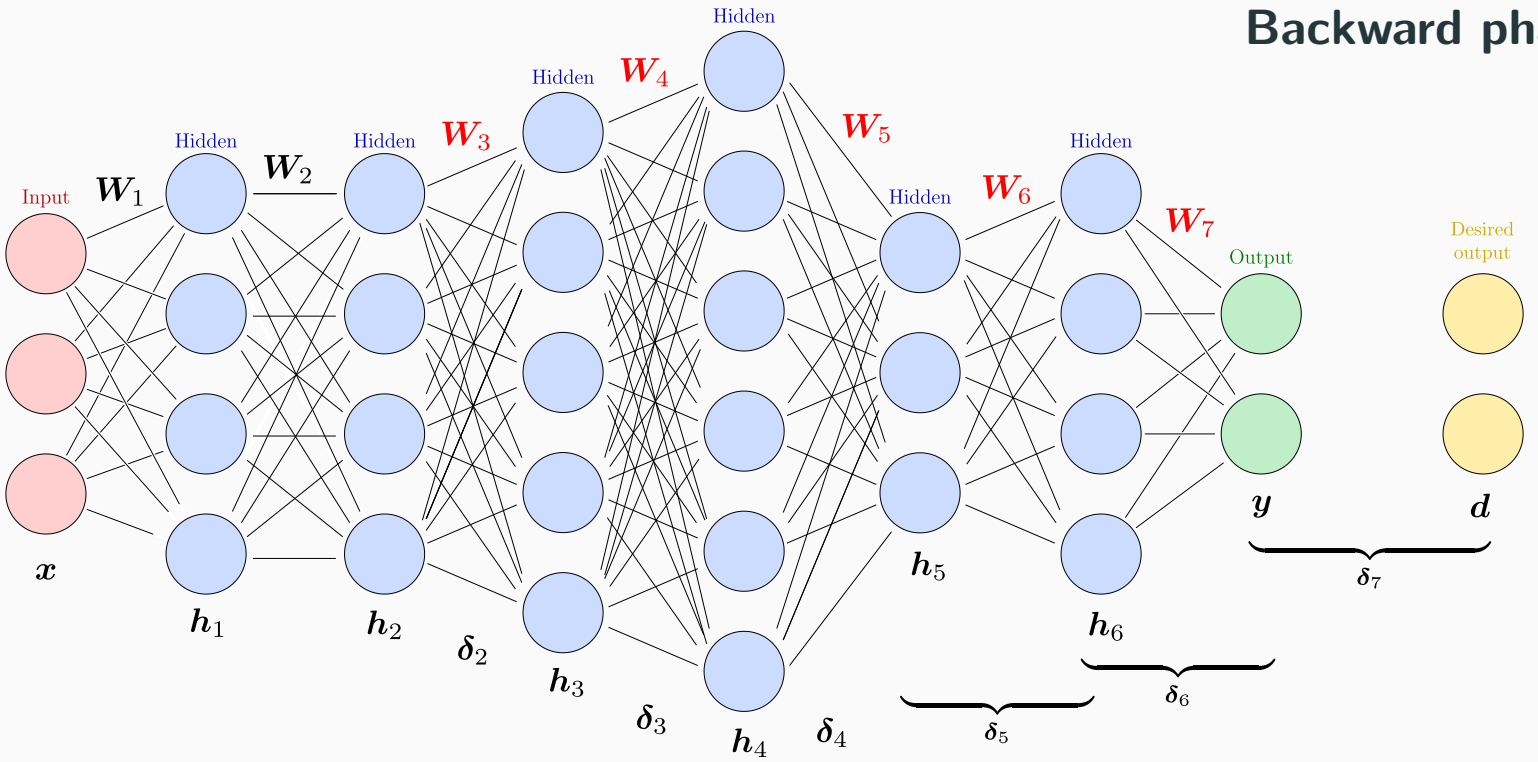
Backpropagation algorithm



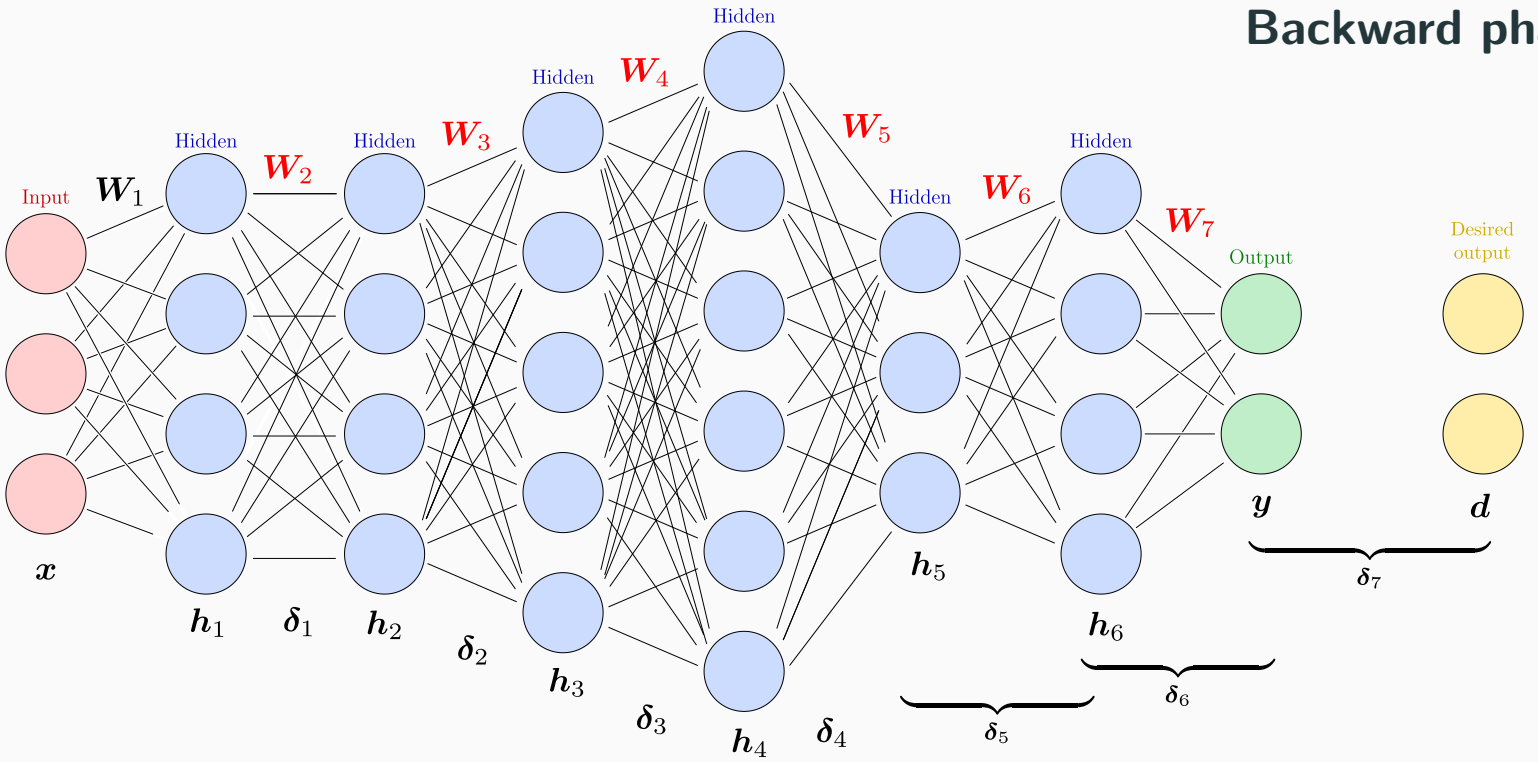
Backpropagation algorithm



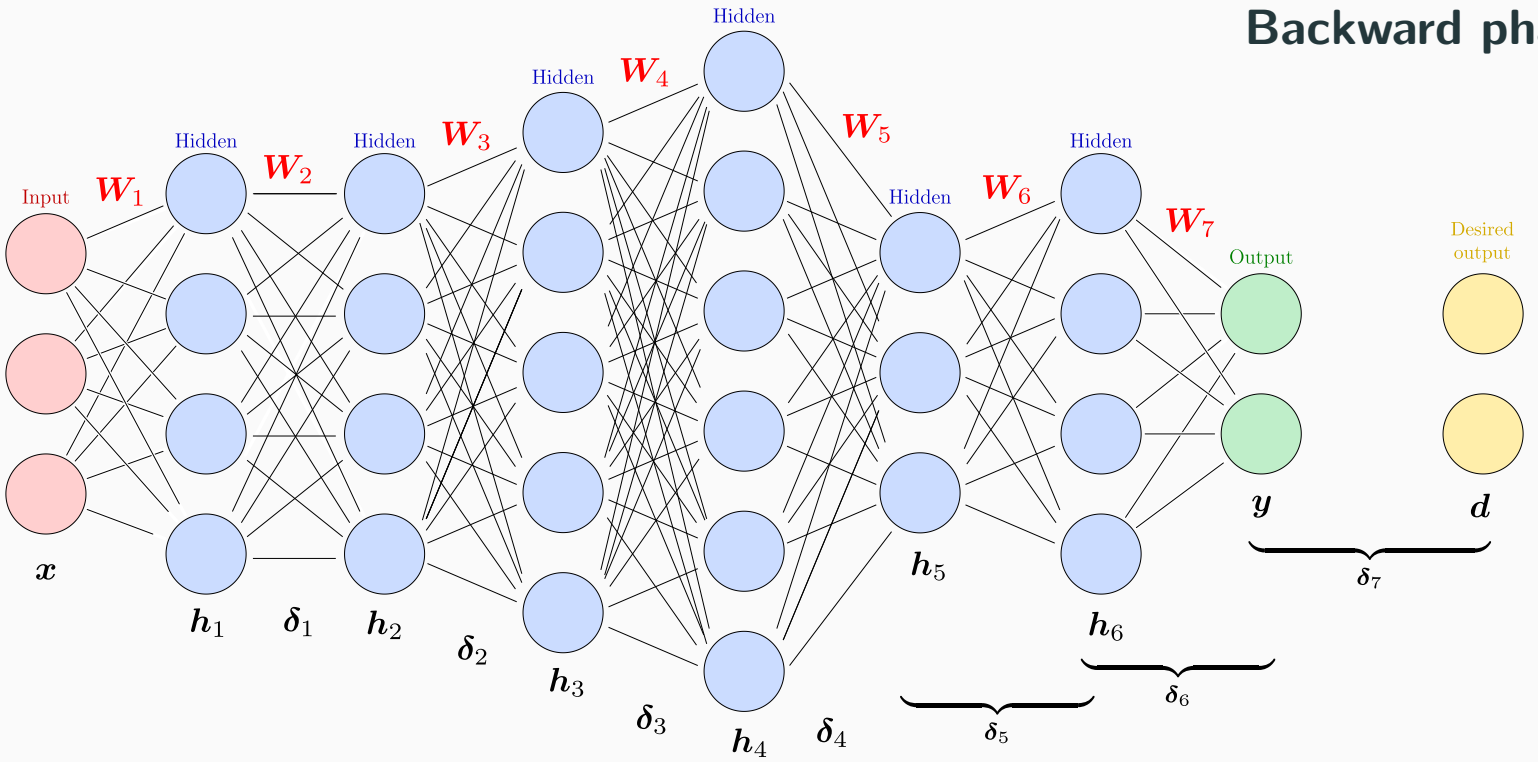
Backpropagation algorithm



Backpropagation algorithm



Backpropagation algorithm



Deep learning

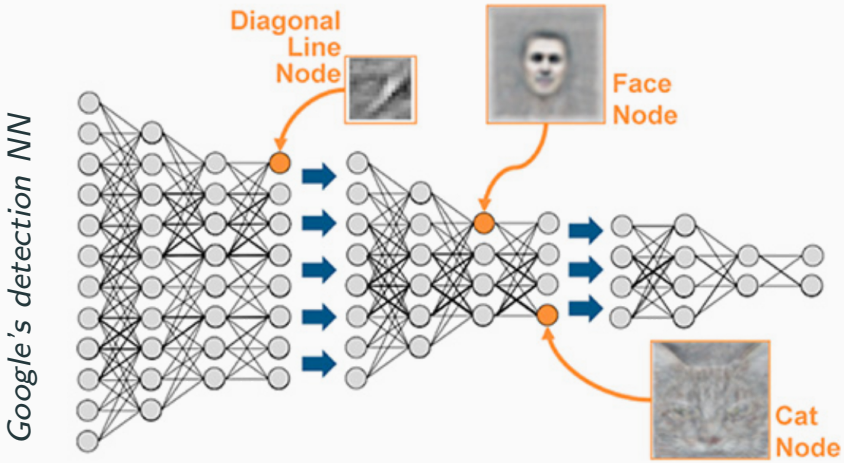
(Source: Dan Page)

ANNs recap

- Interconnections of neurons (summations and activations),
- Feedforward architecture: input $\rightarrow$ hidden layer(s) $\rightarrow$ output,
- Parameterized by a collection of weights $\boldsymbol{W}_k$ (and biases),
- Backprop: parameters learned from a collection of labeled data.

What is deep learning?

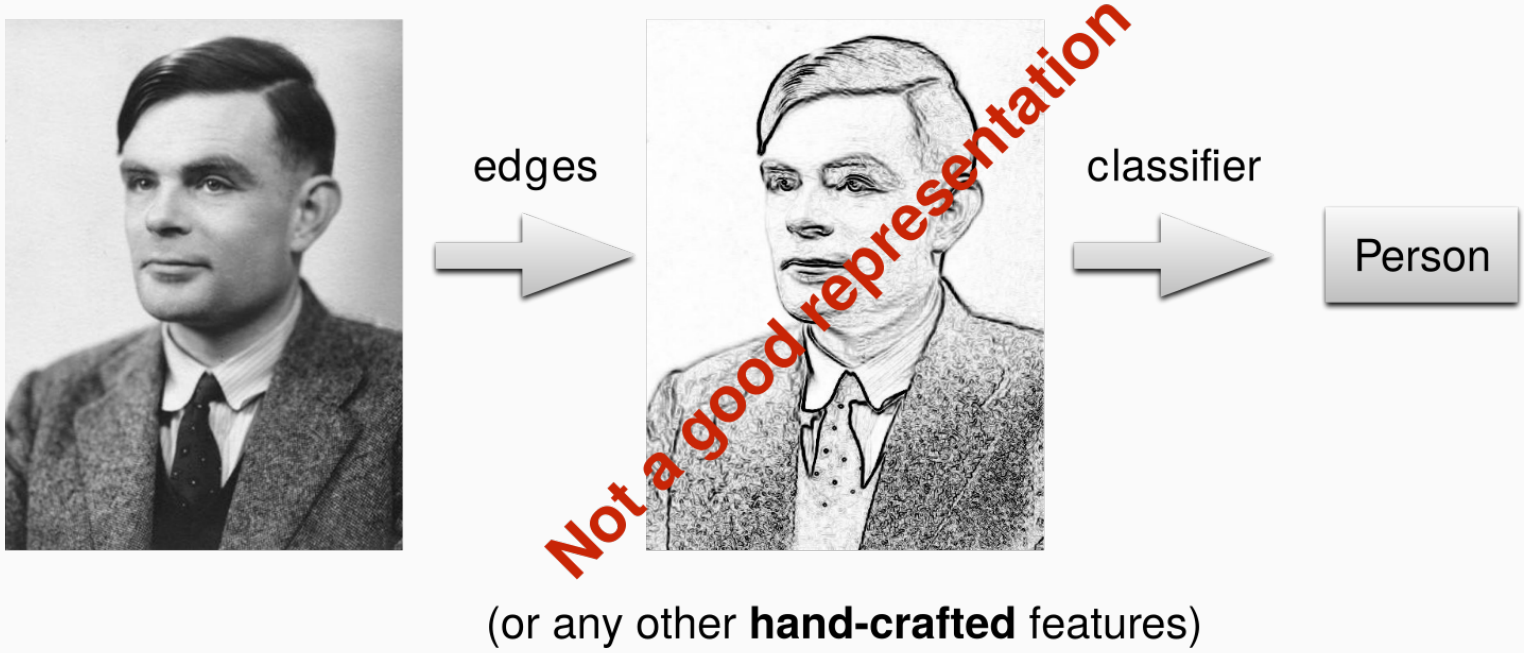
- Representation learning using artificial neural networks
  - → Learning good features automatically from raw data.
  - → Exceptionally effective at learning patterns.
- Learning representations of data with multiple levels of abstraction
  - → hierarchy of layers that mimic the neural networks of our brain,
  - → cascade of non-linear transforms.



- If you provide the system with tons of information, it begins to understand it and responds in useful ways.

(Source: Caner Hazırbaş & Lucas Masuch)

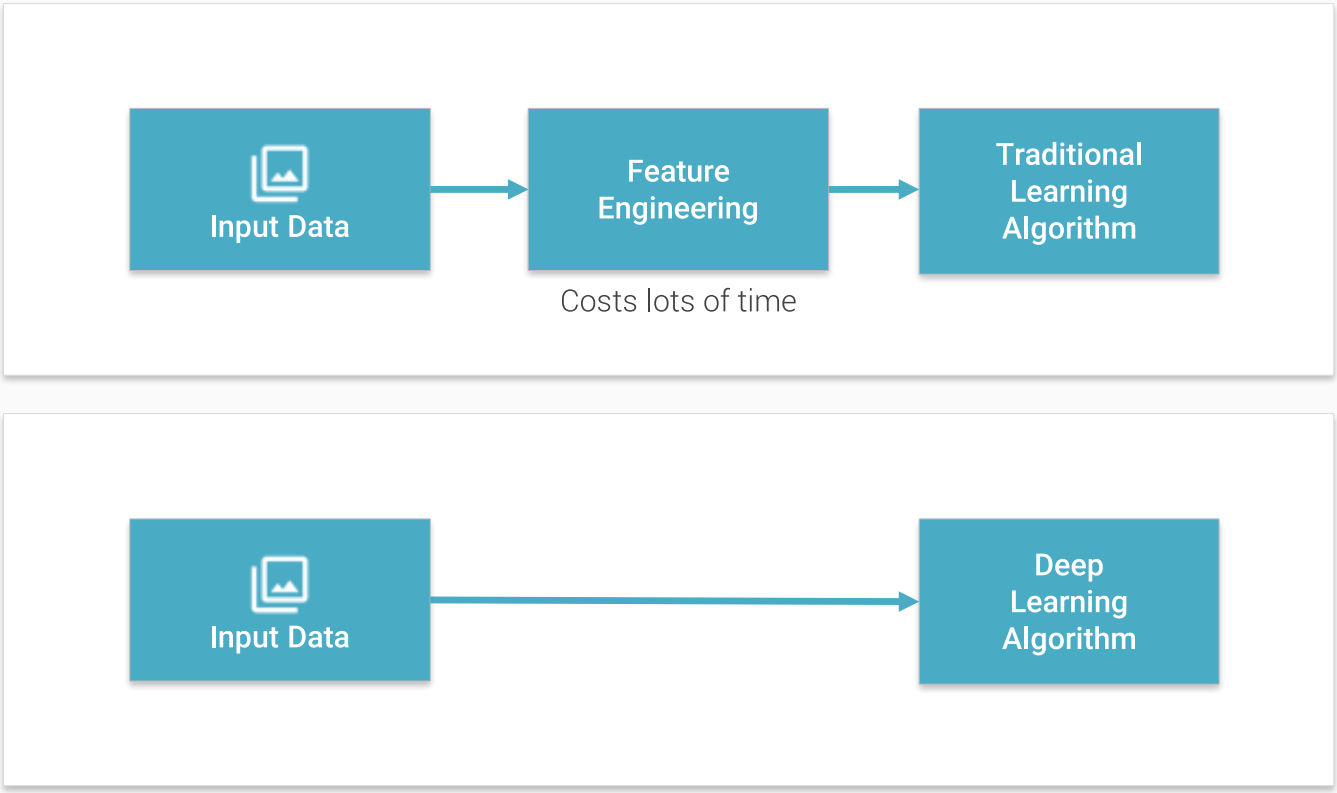
How to teach a machine?



Good representations are often very complex to define.

(Source: Caner Hazırbaş)

Deep learning: No more feature engineering



Learn the latent factors/features behind the data generation

(Source: Lucas Masuch)

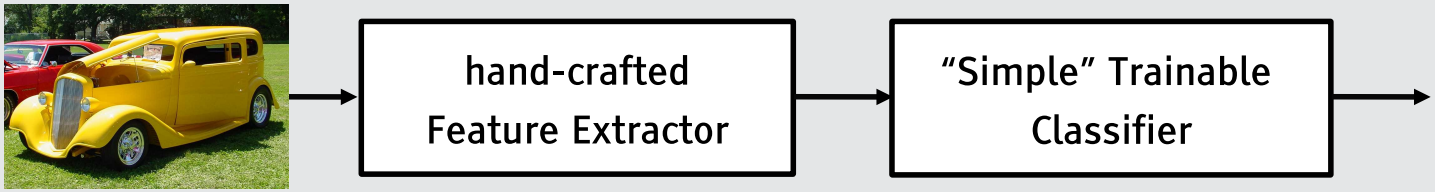
Inspired by the Brain

- The first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.
- Our brain has lots of neurons connected together and the strength of the connections between neurons represents long term knowledge.
- One learning algorithm hypothesis: all significant mental algorithms are learned except for the learning and reward machinery itself.

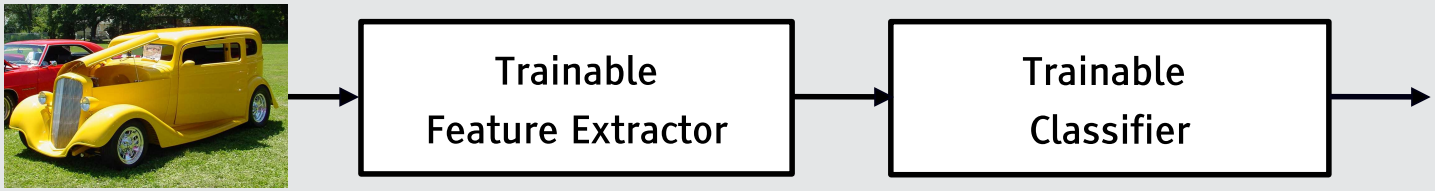
(Source: Lucas Masuch)

Deep learning: No more feature engineering

- The traditional model of pattern recognition (since the late 50's)
  - Fixed/engineered features (or fixed kernel) + trainable classifier



- End-to-end learning / Feature learning / Deep learning
  - Trainable features (or kernel) + trainable classifier

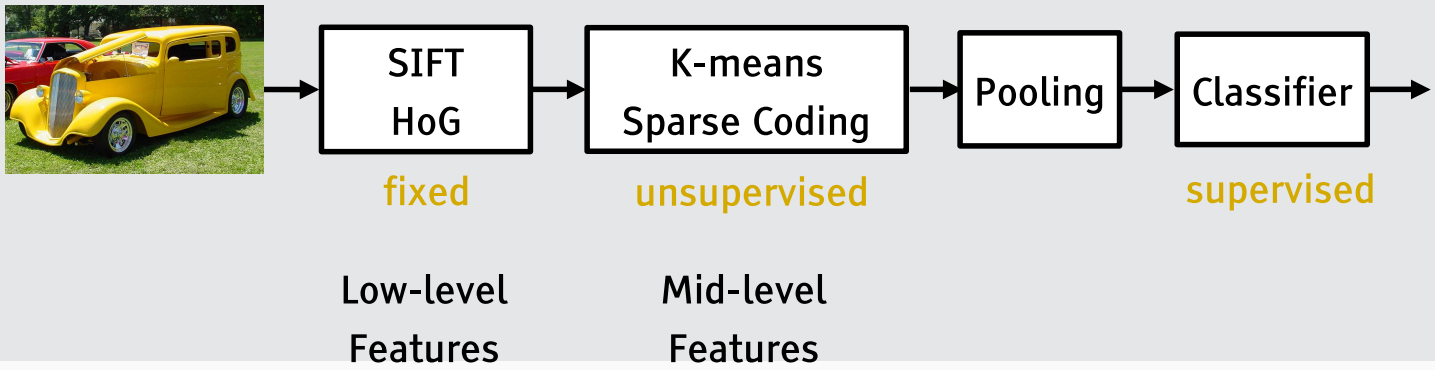


(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning: No more feature engineering

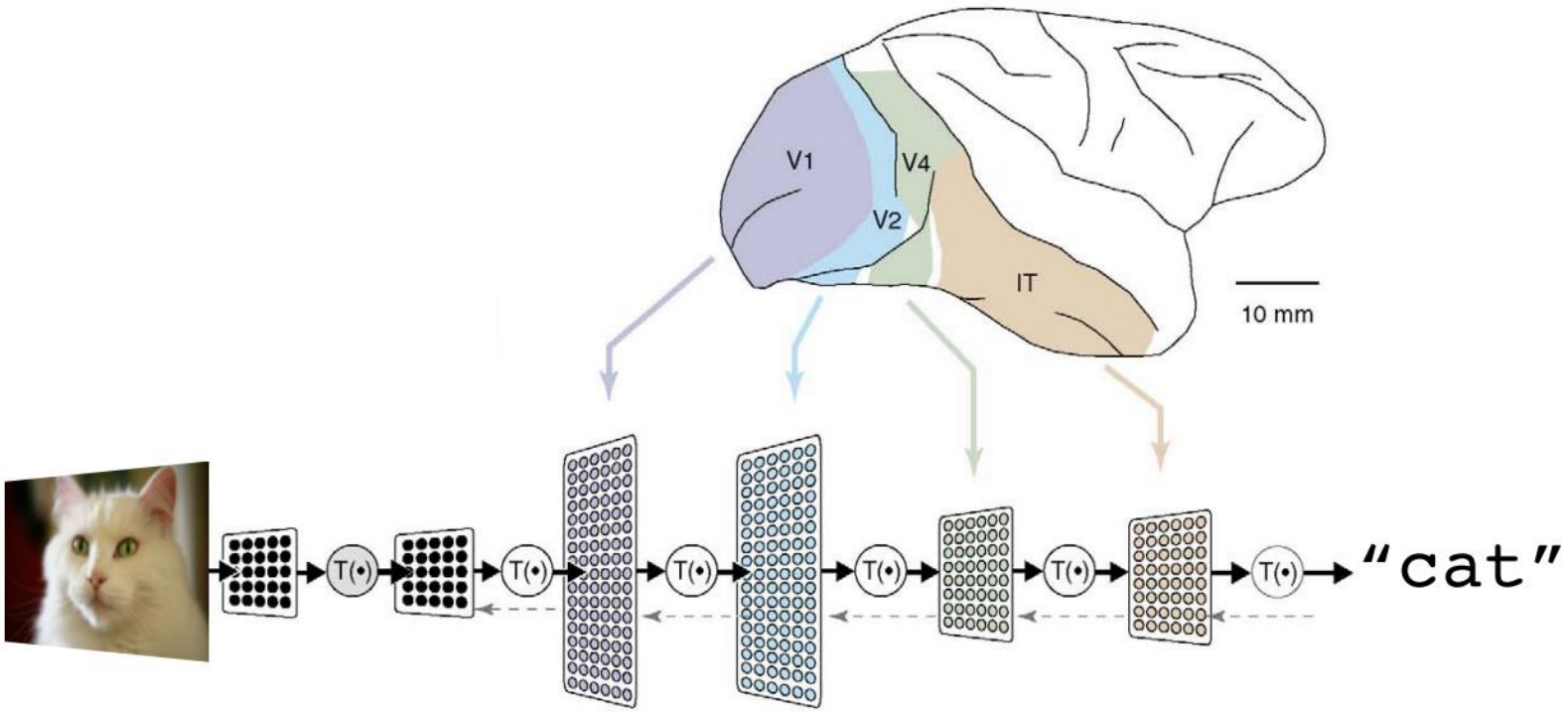
- Non-deep learning architecture for pattern recognition
  - Speech recognition: early 90's – 2011

  - Object Recognition: 2006 – 2012



(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning – Basic architecture



A deep neural network consists of a hierarchy of layers, whereby each layer transforms the input data into more abstract representations (e.g. edge -> nose -> face). The output layer combines those features to make predictions.

(Source: Lucas Masuch)

Trainable feature hierarchy

- Hierarchy of representations with increasing levels of abstraction.
- Each stage is a kind of trainable feature transform.

Image recognition

- Pixel $\rightarrow$ edge $\rightarrow$ texton $\rightarrow$ motif $\rightarrow$ part $\rightarrow$ object

Text

- Character $\rightarrow$ word $\rightarrow$ word group $\rightarrow$ clause $\rightarrow$ sentence $\rightarrow$ story

Speech

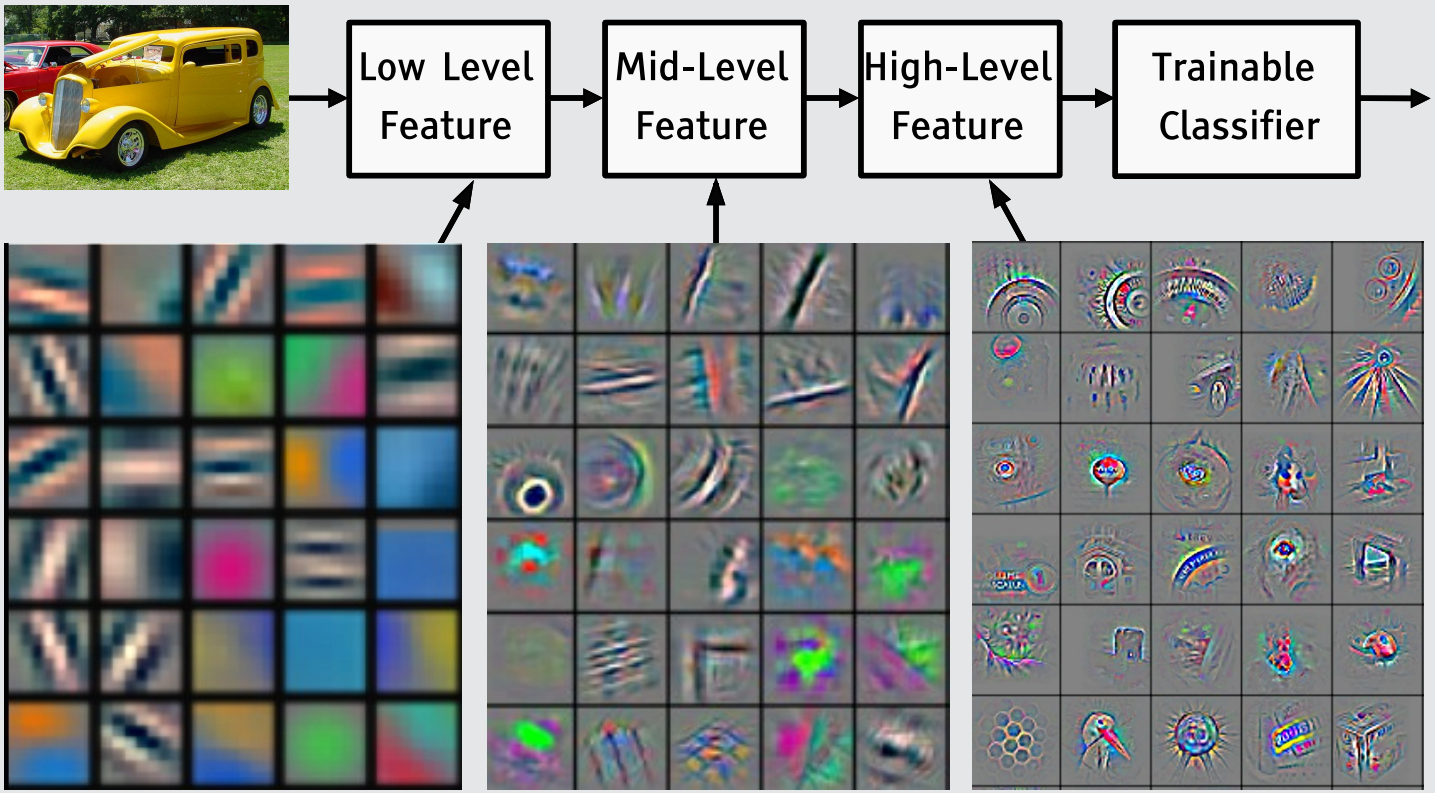
- Sample $\rightarrow$ spectral band $\rightarrow$ sound $\rightarrow$ ... $\rightarrow$ phone $\rightarrow$ phoneme $\rightarrow$ word

Deep Learning addresses the problem of learning hierarchical representations.

(Source: Yann LeCun & Marc'Aurelio Ranzato)

Deep learning – Feature hierarchy

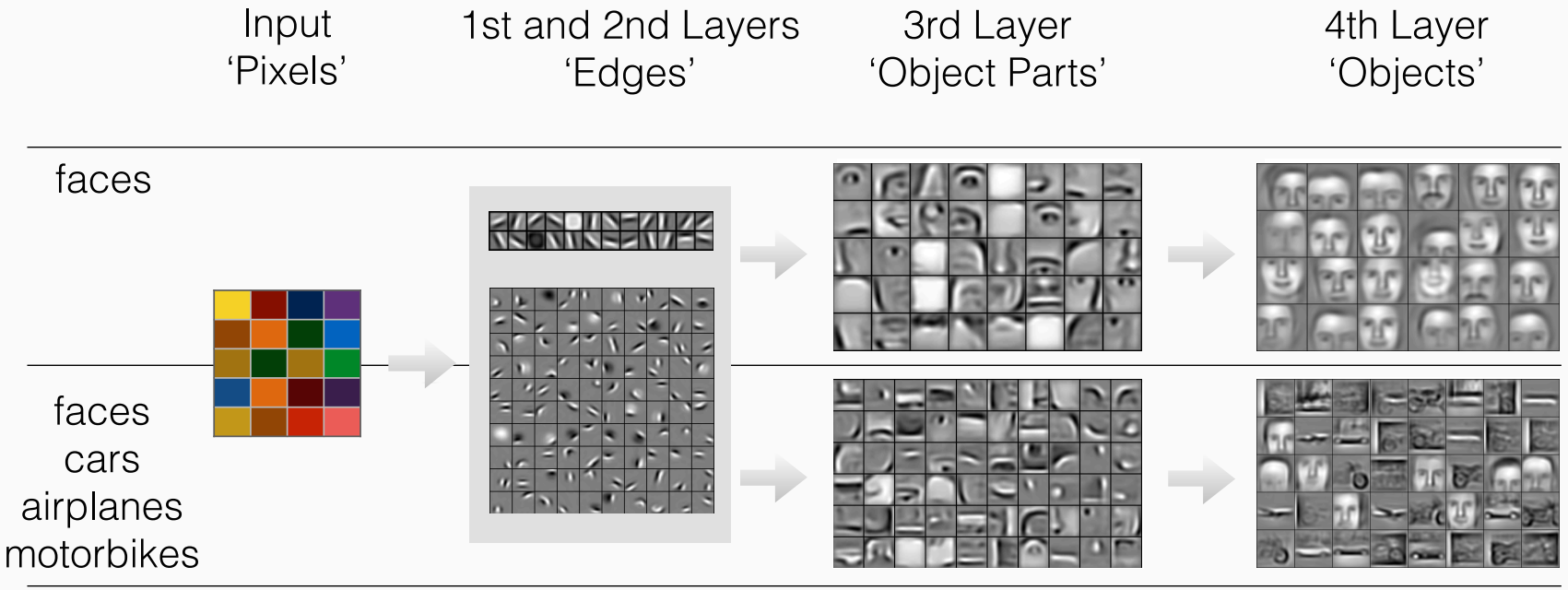
- It's deep if it has more than one stage of non-linear feature transformation



Feature visualization of convolutional net
trained on ImageNet from (Zeiler & Fergus, 2013)

(Source: Yann LeCun & Marc'Aurelio Ranzato)

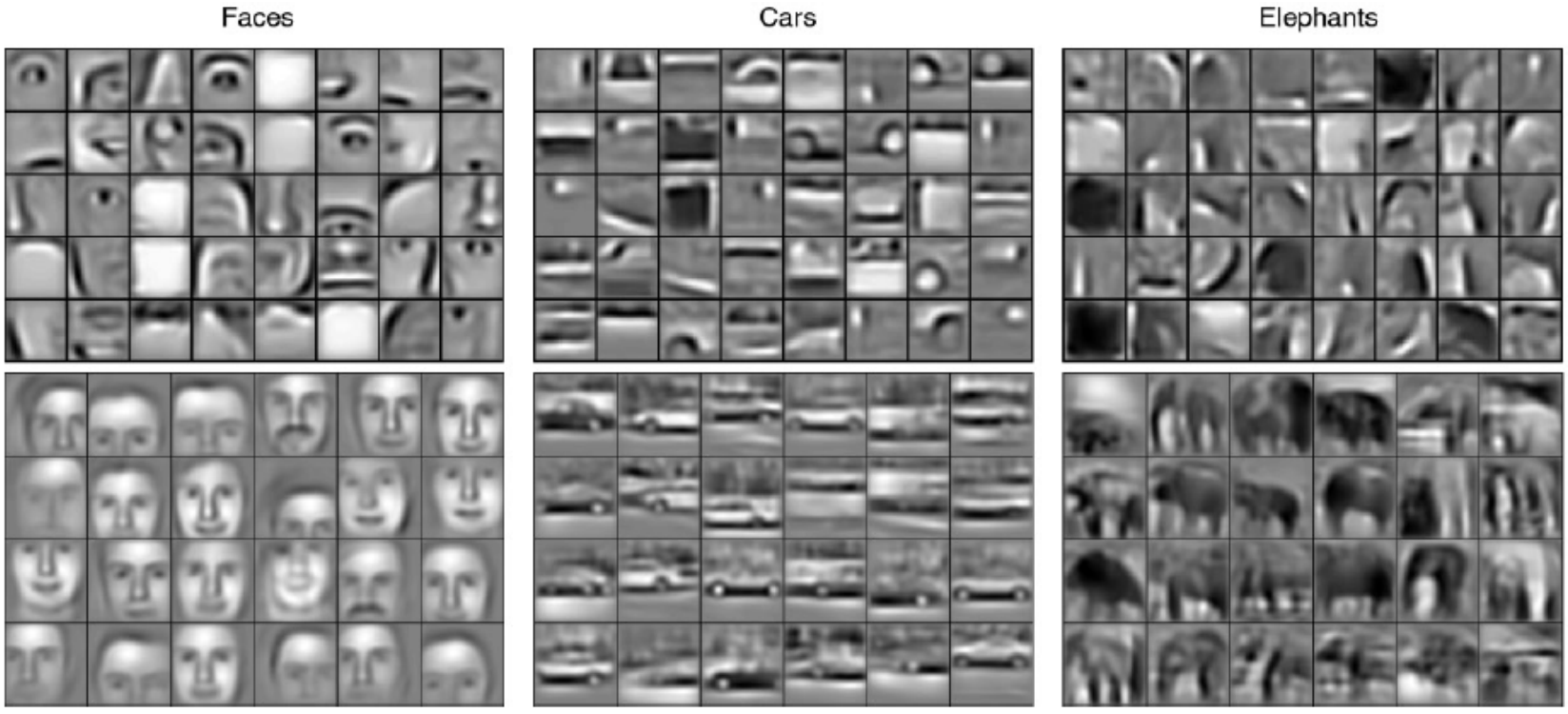
Deep learning – Feature hierarchy



Each layer progressively extracts higher level features of the input until the final layer essentially makes a decision about what the input shows. The more layers the network has, the higher level features it will learn.

(Source: Andrew Ng & Lucas Masuch & Caner Hazırbaş)

Deep learning – Feature hierarchy



- Mid- to high-level features are specific to the given task.
- Low-level representations contain less specific features.
  (can be shared for many different applications)

Deep learning – Training

Today's trend: make it deeper and deeper

- 2012: 8 layers (AlexNet – Krizhevsky *et al.*, 2012)
- 2014: 19 layers (VGG Net – Simonyan & Zisserman, 2014)
- 2014: 22 layers (GoogLeNet – Szegedy *et al.*, 2014)
- 2015: 152 layers (ResNet – He *et al.*, 2015)
- 2016: 201 layers (DenseNet – Huang *et al.*, 2017)

But remember, with back-propagation:

- We got stuck at local optima or saddle points
- The learning time does not scale well
  - it is very slow for deep networks and can be unstable.

How did networks get so deep? First, why does backprop fail?

Deep learning – Gradient vanishing problems

Back-propagation and gradient vanishing problems

$$\text{Update} \quad w_{i,j} \leftarrow w_{i,j} - \gamma \frac{\partial E(\boldsymbol{W})}{\partial w_{i,j}} \quad \text{with} \quad \frac{\partial E(\boldsymbol{W})}{\partial w_{i,j}} = \sum_{(\boldsymbol{x},\boldsymbol{d})\in\mathcal{T}} \delta_i h_j$$

$$\text{where} \quad \delta_i = g'(a_i) \times \begin{cases} y_i - d_i & \text{if } i \text{ is an output node} \\ \sum_l w_{l,i}\delta_l & \text{otherwise} \end{cases}$$

- With deep networks, the gradient vanishes quickly.
- Unfortunately, this arises even though we are far from a solution.
- The updates become insignificant, which leads to slow training rates.
- This strongly depends on the shape of $g'(a)$.

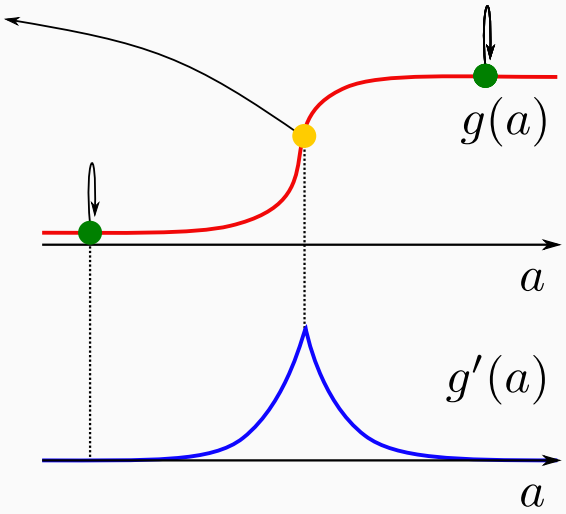
- The gradient may also explode leading to instabilities:

  $\rightarrow$ gradient exploding problem.

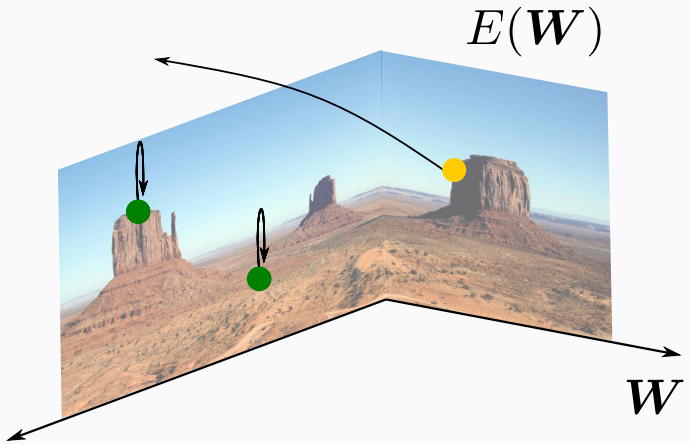
Deep learning – Gradient vanishing problem

As the network gets deeper, the landscape of E becomes:

- very hilly → lots of stationary points,
- with large plateaus → gradient vanishing problem,
- and delimited by cliffs → gradient exploding problem.



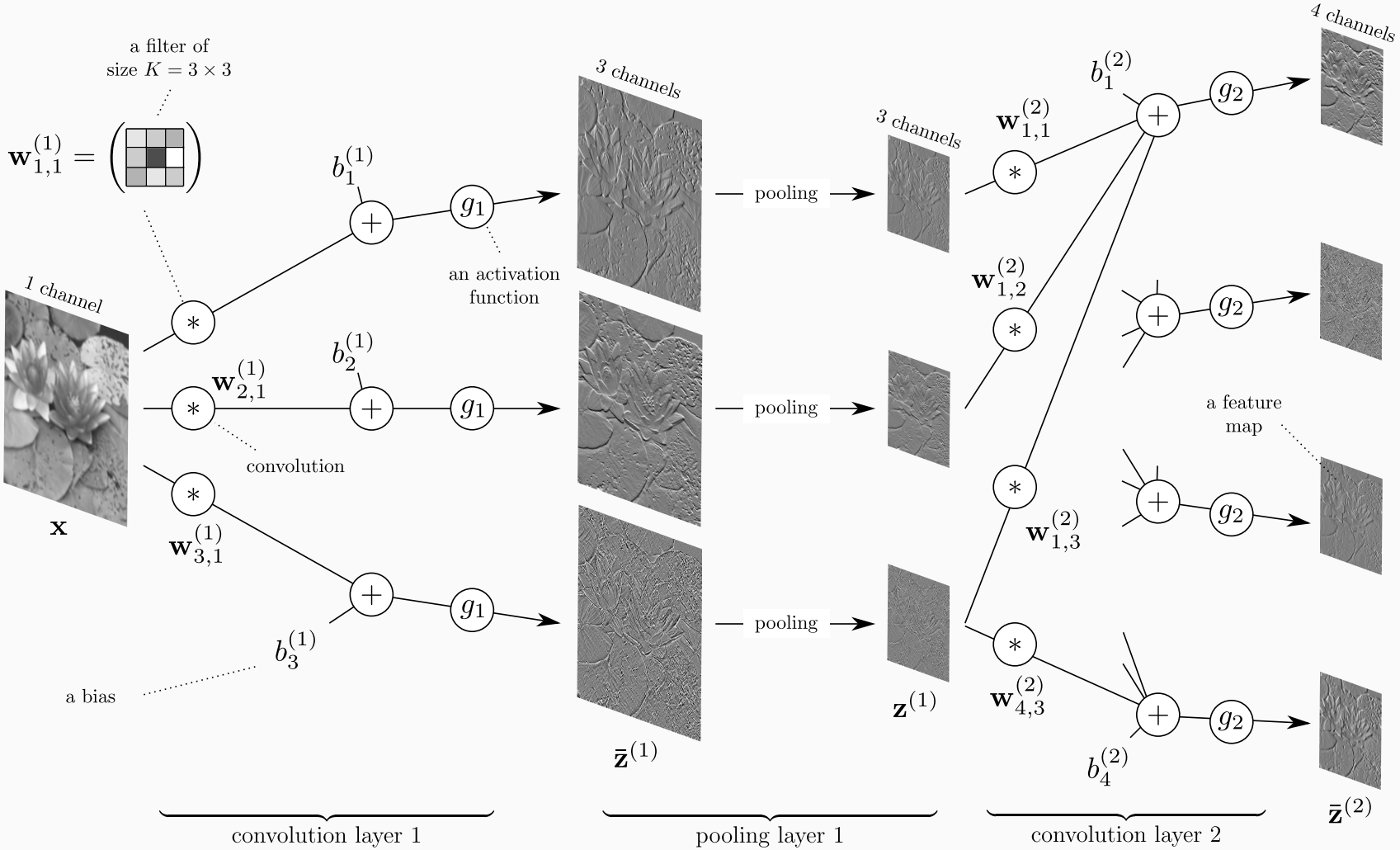
Activation function



Cost landscape

So, what has changed?

CNN for image processing



Convolution: One chanel

For an image $x = x(i, j)$ having **one chanel**, the convolution with a kernel κ of size $[-s, s] \times [-s, s]$

- the **convolution** is:

$$x * \kappa(i, j) = \sum_{(k,\ell)\in[-s,s]\times[-s,s]} \kappa(k, \ell)x(i - k, j - \ell)$$

  $x * \kappa =$ local average of x with the weight of κ in **opposite position**

- the **cross-correlation** is:

$$x \otimes \kappa(i, j) = \sum_{(k,\ell)\in[-s,s]\times[-s,s]} \kappa(k, \ell)x(i + k, j + \ell).$$

  $x \otimes \kappa =$ local average of x with the weight of κ in **same position**

- Need to deal with boundary issues for pixels at the border: zero-padding (0 if outside border), valid positions only (do not compute at border, smaller output images), mirror symmetry at border...

Convolution

- Images have generally several chanels (eg RGB).
- By computing several convolutions we can stack the result as a single multi-channel output.

In machine learning
“convolution”
means
“cross-correlation + bias”

- Recall that “linear” layers are affine map $x \mapsto Wx + b$, so convolution layers are a specific case where $x \mapsto Wx$ is a cross-correlation.

Convolution layer

Convolution layer with c_{in} input chanels and c_{out} output chanels:

- Input image $\boldsymbol{x}$ with c_{in} **chanels**: values $\boldsymbol{x}(i,j) \in \mathbb{R}^{c_{\text{in}}}$
- Output image $\boldsymbol{y}$ with c_{out} **chanels**.
- Kernel: $\boldsymbol{\kappa}$ such that for all $(k,\ell) \in [-s,s] \times [-s,s]$

$$\boldsymbol{\kappa}(k,\ell) \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}, \quad \text{is a } c_{\text{out}} \times c_{\text{in}} \text{ matrix}$$

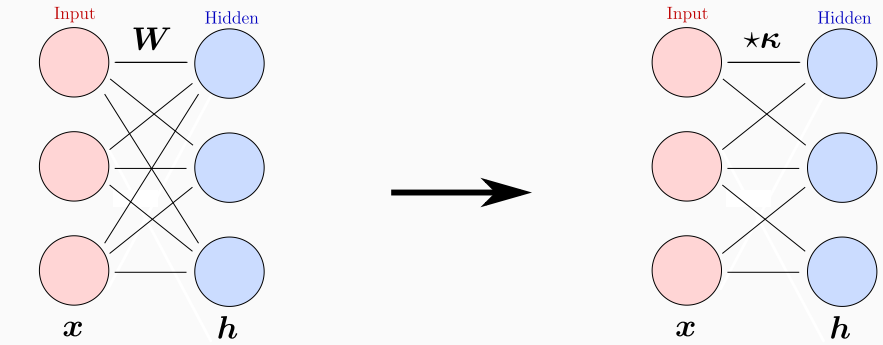
- Bias: $\boldsymbol{b} \in \mathbb{R}^{c_{\text{out}}}$

$$\boldsymbol{y} = \text{Conv}(\boldsymbol{x}; \boldsymbol{\kappa}, \boldsymbol{b})(i,j) = \sum_{(k,\ell) \in [-s,s] \times [-s,s]} \boldsymbol{\kappa}(k,\ell)\boldsymbol{x}(i+k, j+\ell) + \boldsymbol{b} \in \mathbb{R}^{c_{\text{out}}}$$

- Number of parameters: $(2s+1)^2 \times c_{\text{in}} \times c_{\text{out}}$ for $\boldsymbol{\kappa}$ and c_{out} for $\boldsymbol{b}$

What are CNNs?

- Essentially neural networks that use convolution in place of general matrix multiplications at least for the first layers.



Fully connected (FC) layer

Convolutional layer

- CNNs are designed to process the data in the form of multidimensional arrays/tensors (*e.g.*, 2D images, 3D video/volumetric images).
- Composed of series of stages: convolutional layers and pooling layers.
- Units connected to local regions in the feature maps of the previous layer.
- Do not only mimic the brain connectivity but also the visual cortex.

Convolutional neural networks

CNNs are composed of three main ingredients:

1. Local receptive fields
   - hidden units connected only to a small region of their input,
2. Shared weights
   - same weights and biases for all units of a hidden layer,
3. Pooling
   - condensing hidden layers.

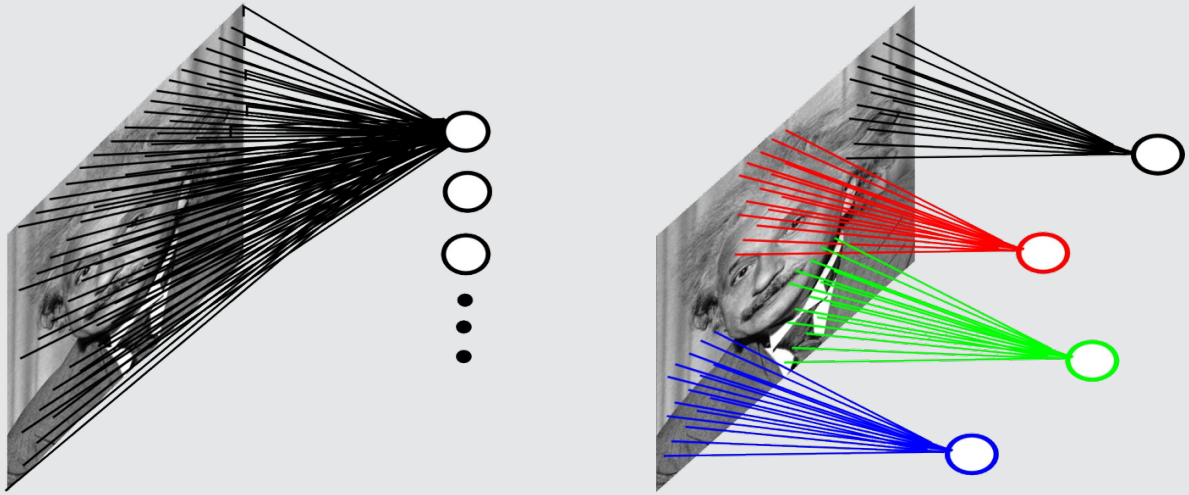
but also

4. Redundancy: more units in a hidden layer than inputs,
5. Sparsity: units should not all fire for the same stimulus.

All take inspiration from the visual cortex.

Local receptive fields $\rightarrow$ Locally connected layer

- Each unit in a hidden layer can see only a small neighborhood of its input,
- Captures the concept of spatiality.



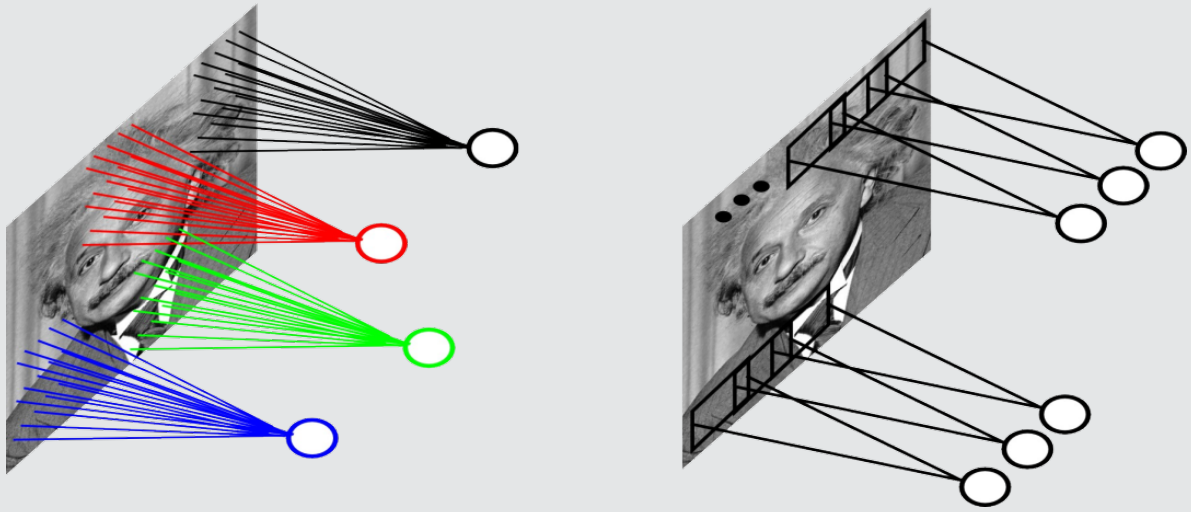
Fully connected | Locally connected

For a 200×200 image and 40,000 hidden units

- Fully connected: 1.6 billion parameters,
- Locally connected (10×10 fields): 4 million parameters.

Self-similar receptive fields $\rightarrow$ Shared weights

- Detect features regardless of position (translation invariance),
- Use convolutions to learn simple input patterns.



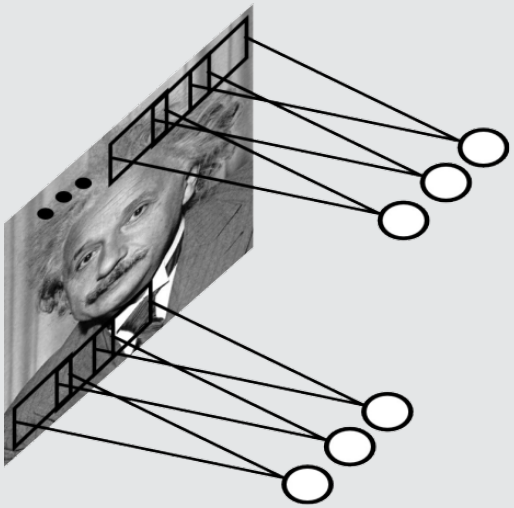
Locally connected | Shared weights

For a 200×200 image and 40,000 hidden units

- Locally connected (10×10 fields): 4 million parameters,
- & Shared weights: 100 parameters (independent of image size).

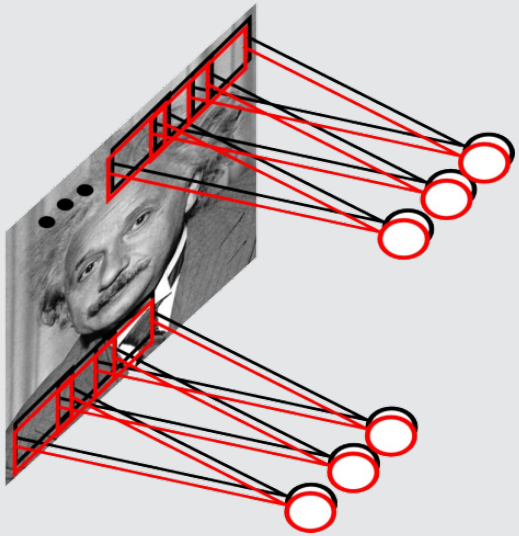
Specialized cells $\rightarrow$ Filter bank

- Use a filter bank to detect multiple patterns at each location,
- Multiple convolutions with different kernels,
- Result is a 3d array, where each slice is a feature map.



Shared weights

(1 input $\rightarrow$ 1 feature map)



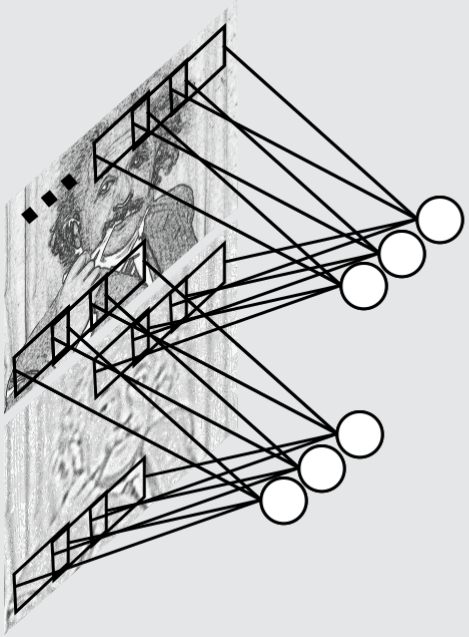
Filter bank

(1 input $\rightarrow$ 2 feature maps)

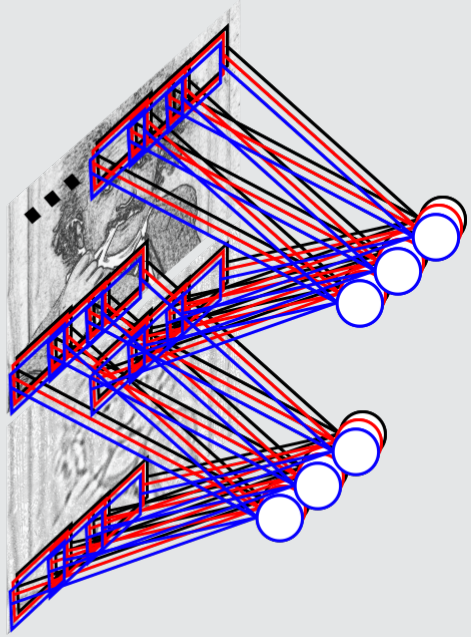
- 10×10 fields & 10 output features: 1,000 parameters.

Hierarchy $\rightarrow$ inputs of deep layers are themselves 3d arrays

- Learn to filter each channel such that their sum detects a relevant feature,
- Repeat as many times as the desired number of output features should be.



Multi-input filter
(2 inputs $\rightarrow$ 1 feature map)

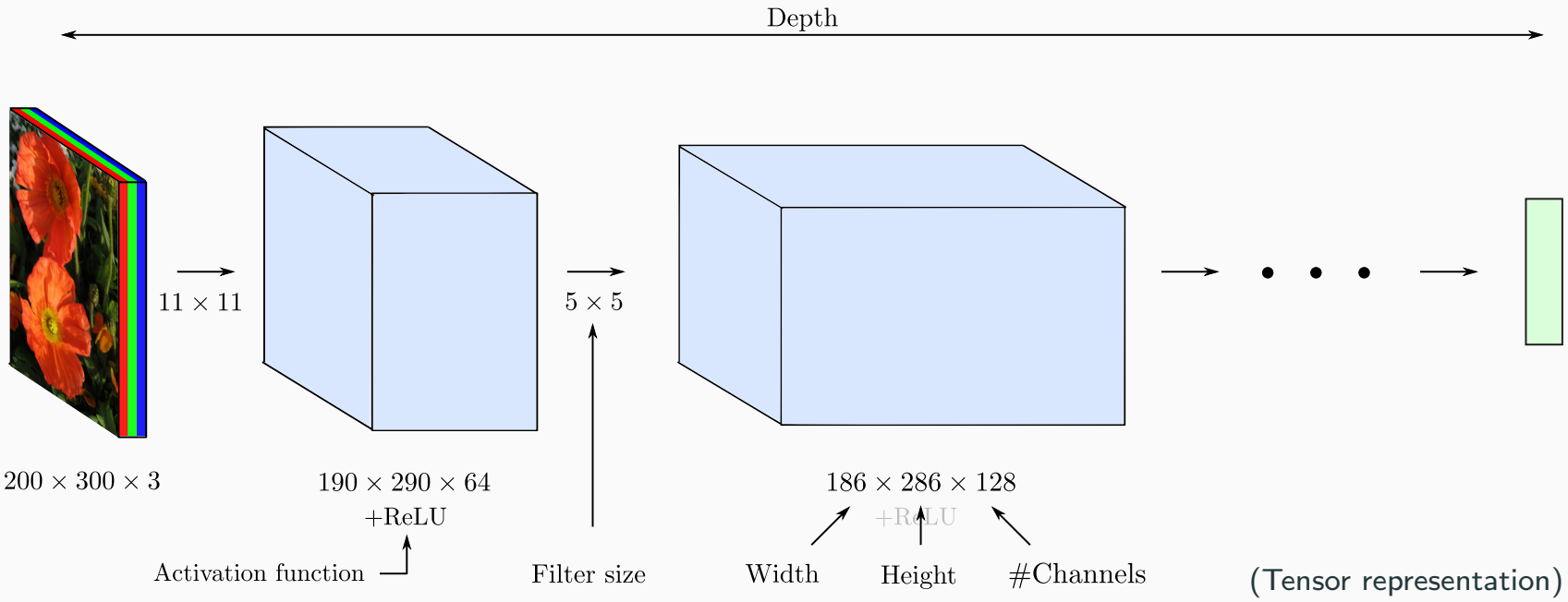


Multi-input filter bank
(2 inputs $\rightarrow$ 3 feature maps)

- **Remark:** these are not 3d convolutions, but sums of 2d convolutions.

- 10×10 fields & 10 inputs & 10 outputs: 10,000 parameters.

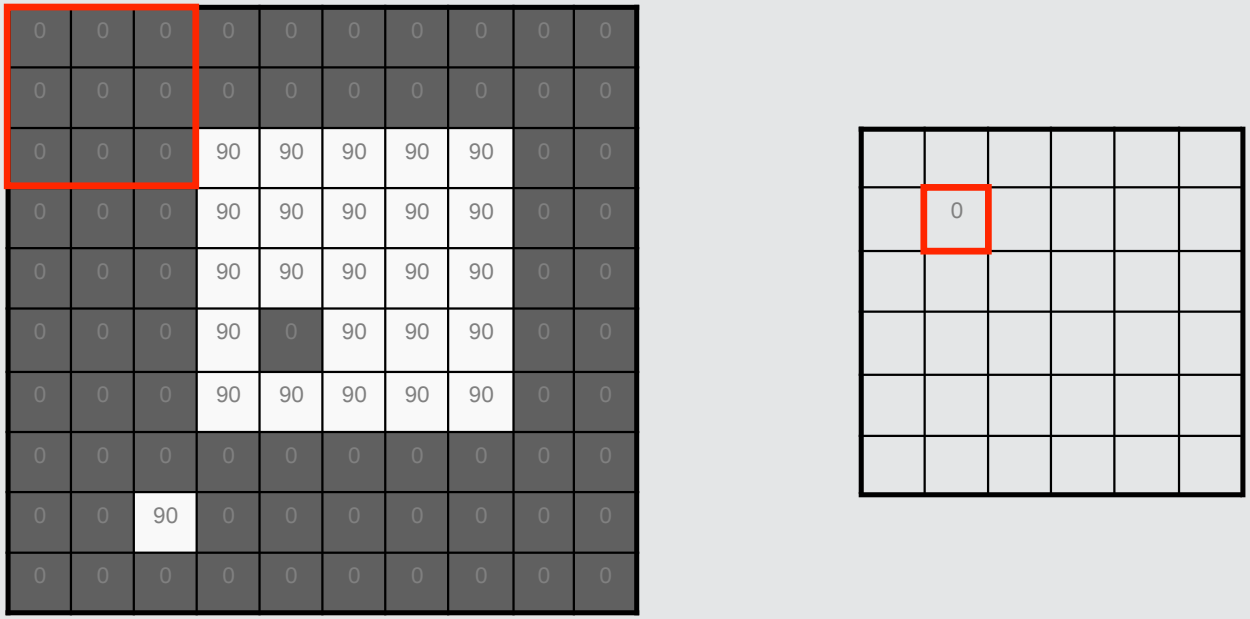
Overcomplete $\rightarrow$ increase the number of channels



- **Redundancy**: increase the number of channels between layers.
- **Padding**: $n \times n$ conv + *valid* $\rightarrow$ width and height decrease by $n - 1$.
- Can we control even more the number of simple cells?

Controlling the number of simple cells $\rightarrow$ Stride

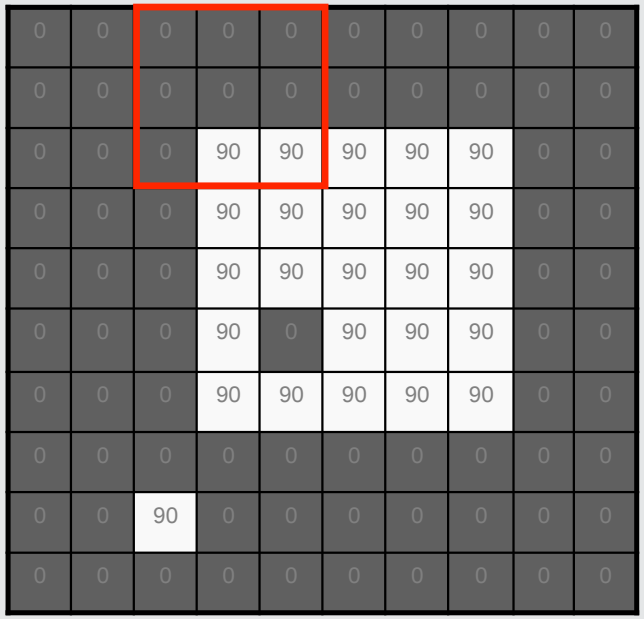
3×3 boxcar strided convolution with stride $s = 2$

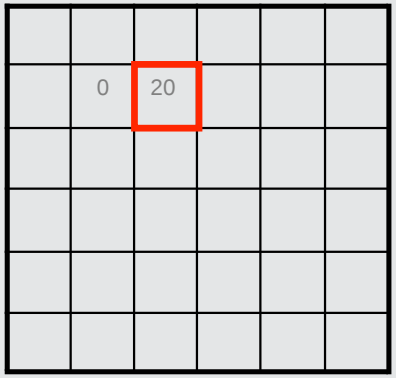


- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* $\rightarrow$ width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells $\rightarrow$ Stride

3×3 boxcar strided convolution with stride $s = 2$

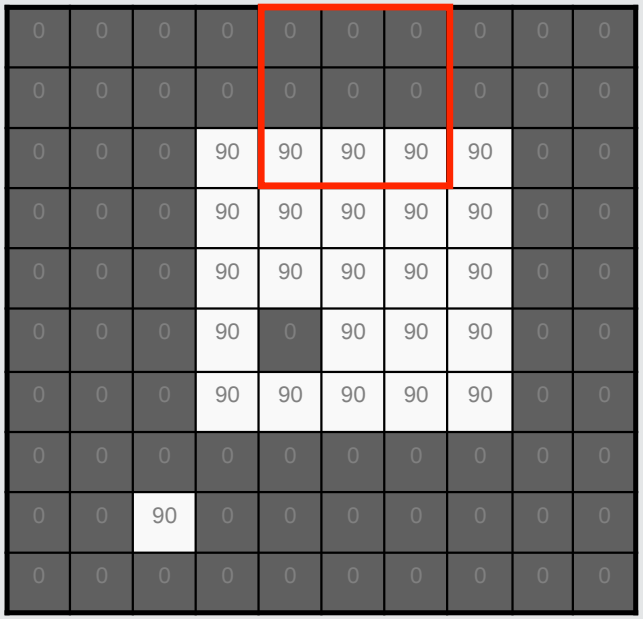


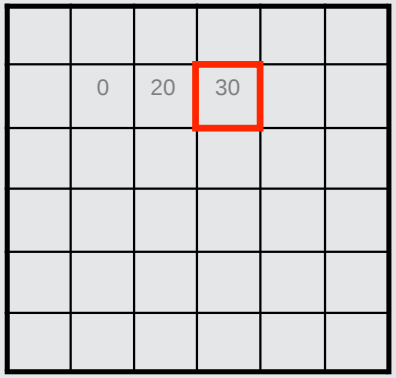


- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* $\rightarrow$ width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells $\rightarrow$ Stride

3×3 boxcar strided convolution with stride $s = 2$

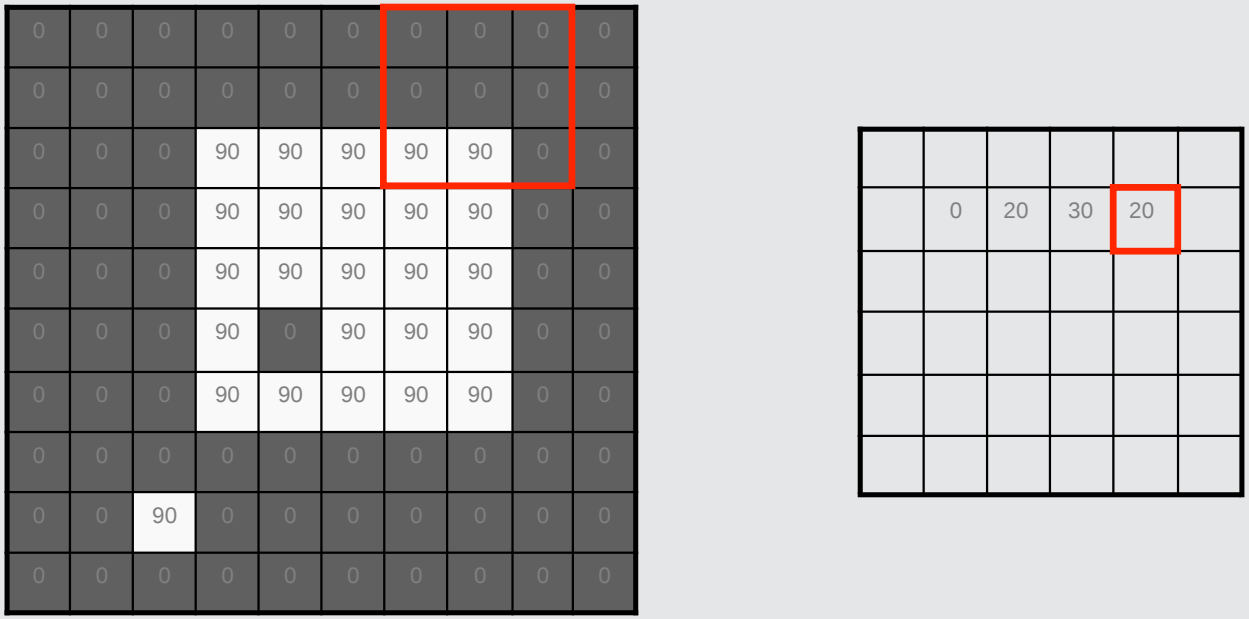




- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* $\rightarrow$ width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells $\rightarrow$ Stride

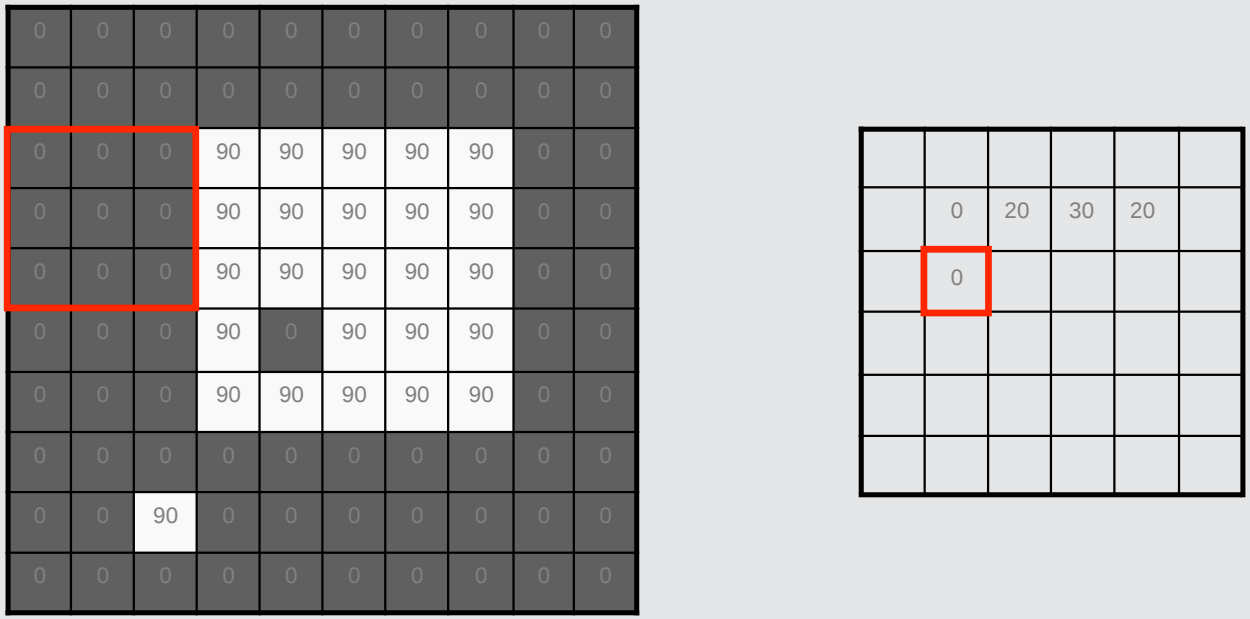
3×3 boxcar strided convolution with stride $s = 2$



- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* $\rightarrow$ width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells $\rightarrow$ Stride

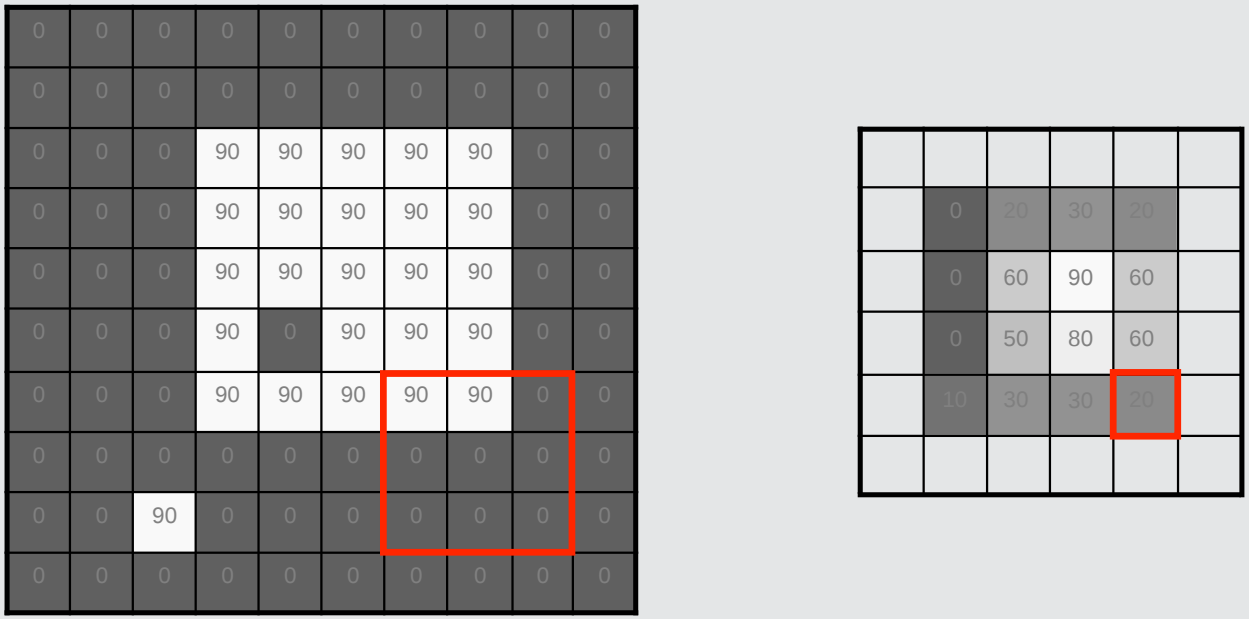
3×3 boxcar strided convolution with stride $s = 2$



- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* $\rightarrow$ width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Controlling the number of simple cells $\rightarrow$ Stride

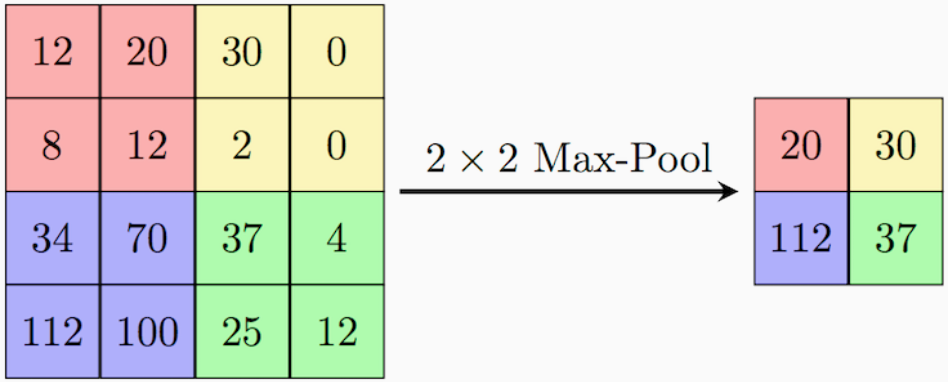
3×3 boxcar strided convolution with stride $s = 2$



- Slide the filter by s pixels step by step, not one by one,
- The interval s is called stride (usually $s = 2$),
- $n \times n$ conv + *valid* $\rightarrow$ width/height decrease to $\lceil \frac{w-n+1}{s} \rceil$ and $\lceil \frac{h-n+1}{s} \rceil$,
- Equivalent in subsampling/decimating the standard convolution,
- Trade-off between computation and degradation of performance.

Pooling layer

- Used after each convolution layer to mimic complex cells,
- Unlike striding, reduce the size by aggregating inputs:
  - Partition the image in a grid of $z \times z$ windows (usually $z = 2$),
  - max-pooling: take the $\max$ in the window

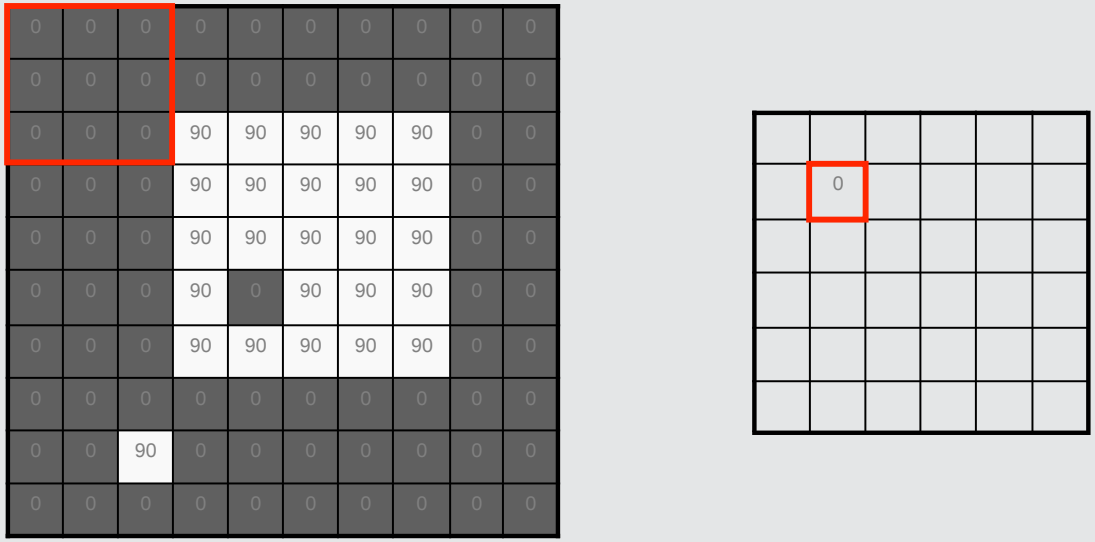


  - average-pooling: take the average

Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:

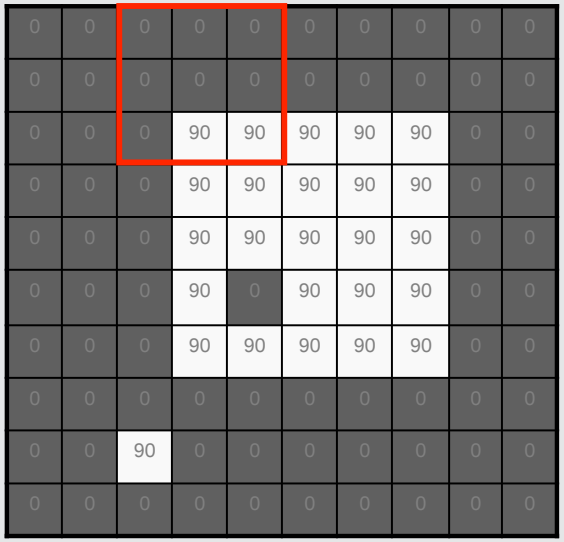


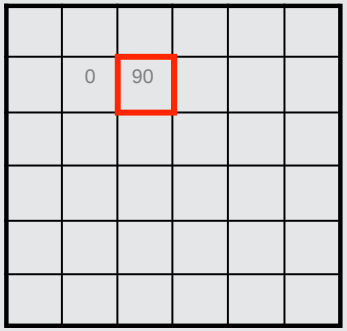
- When $z = s$: standard pooling (non-overlapping),

Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



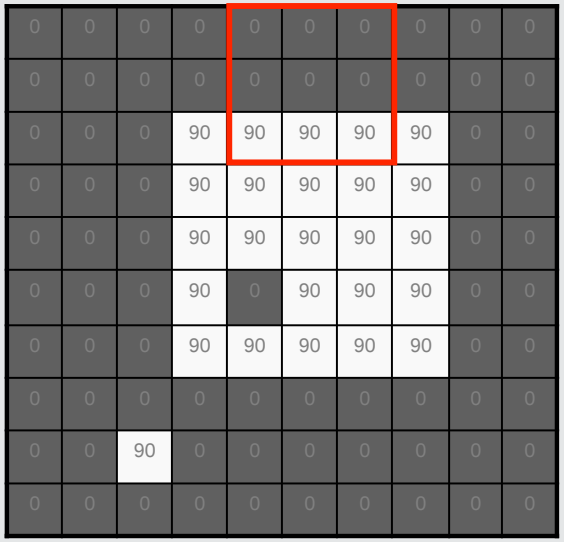


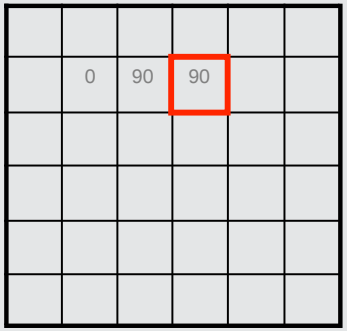
- When $z = s$: standard pooling (non-overlapping),

Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



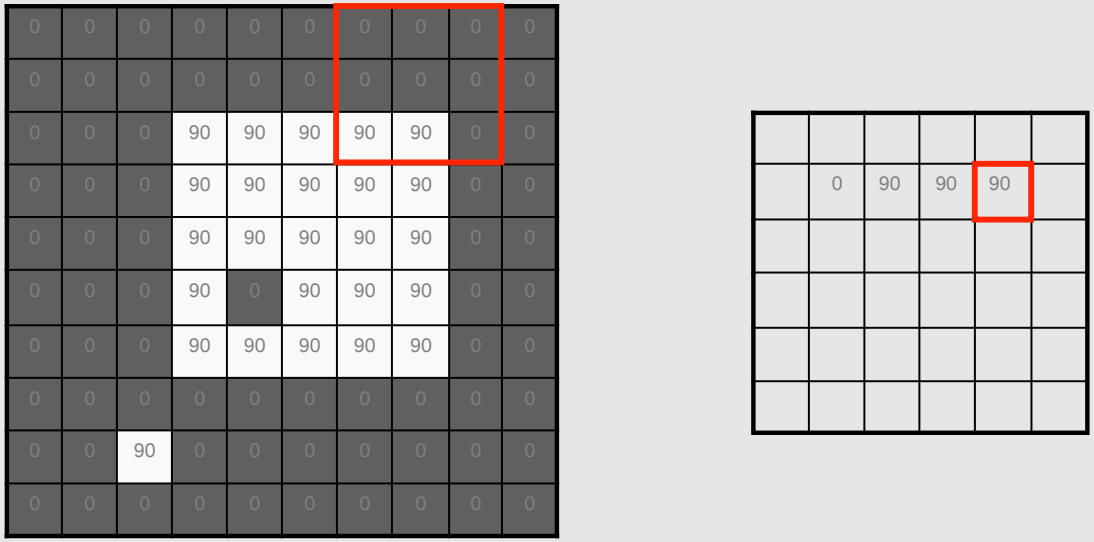


- When $z = s$: standard pooling (non-overlapping),

Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:

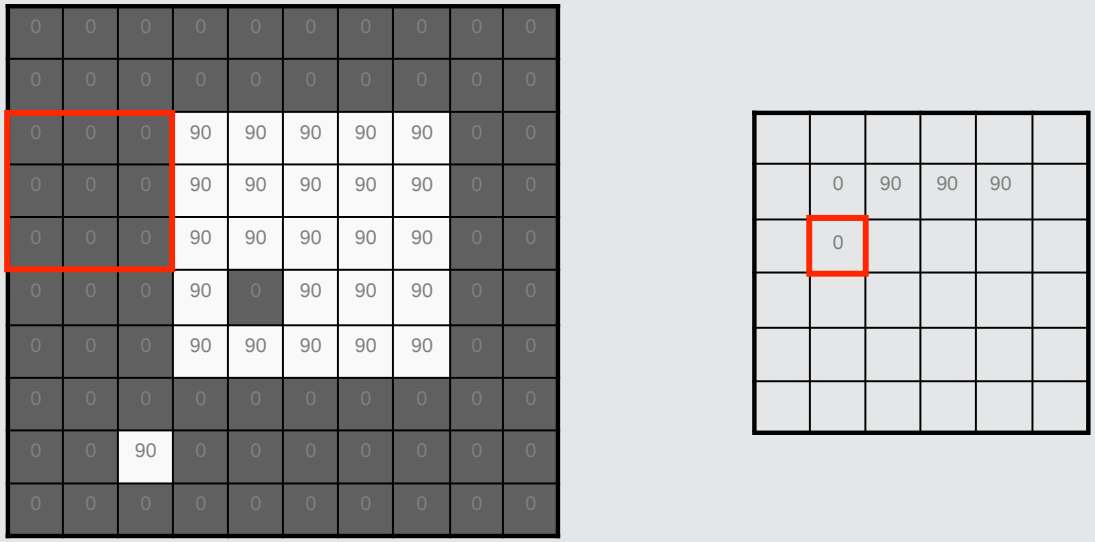


- When $z = s$: standard pooling (non-overlapping),

Pooling layer

Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:

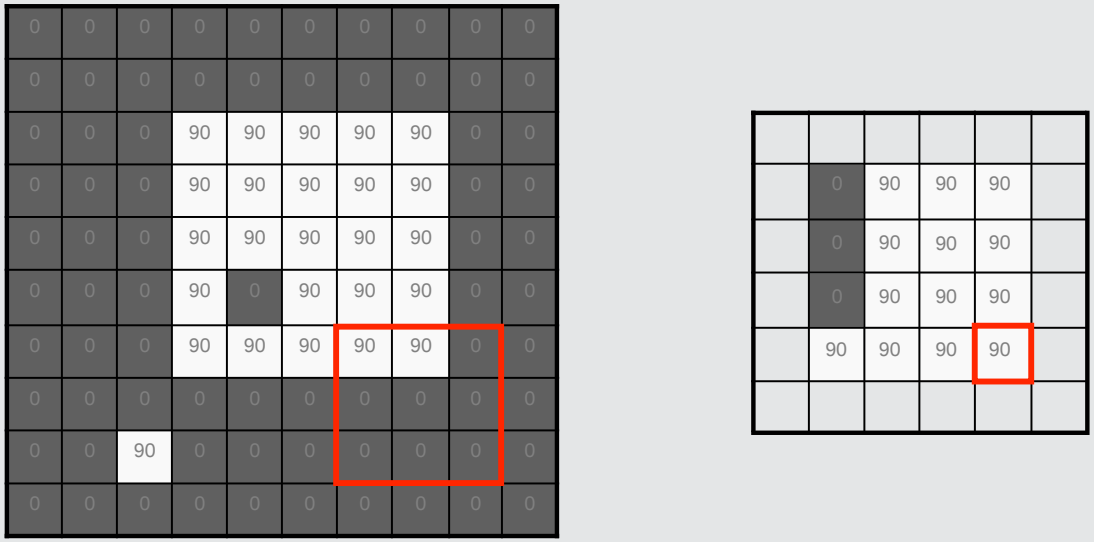


- When $z = s$: standard pooling (non-overlapping),

Pooling layer

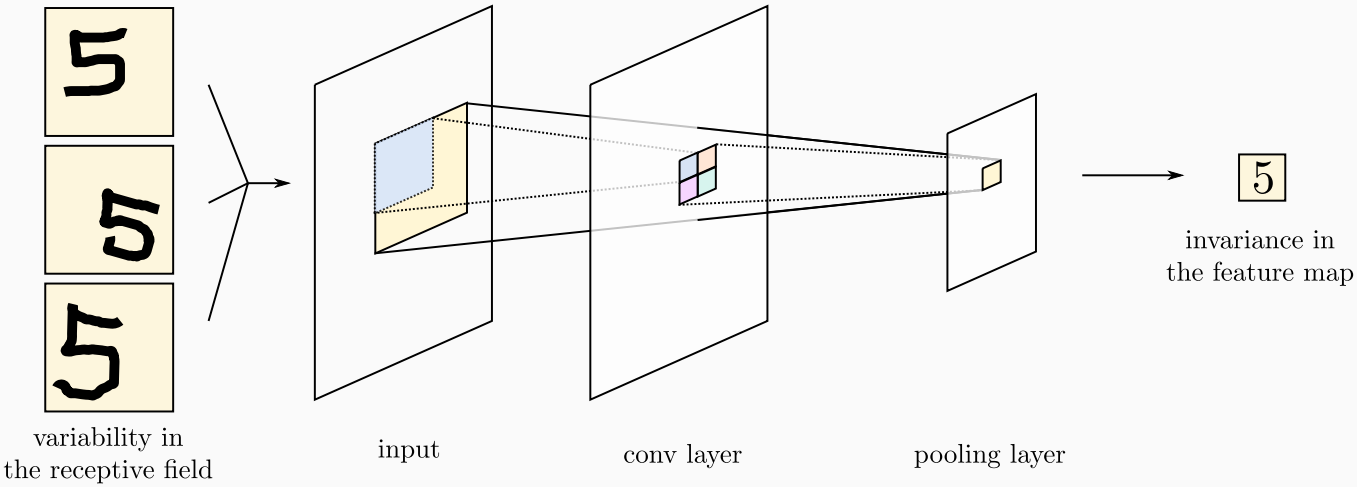
Variant: **Overlapping pooling** (Krizhevsky, Sutskever, Hinton, 2012)

- Perform pooling in a $z \times z$ sliding window,
- Use striding every s pixels,
- Typically: use max-pooling with $z = 3$ and $s = 2$:



- When $z = s$: standard pooling (non-overlapping),

Pooling layer



- Makes the output unchanged even if the input is a little bit changed,
- Allows some invariance/robustness with respect to the exact position,
- Simplifies/Condenses/Summarizes the output from hidden layers,
- Increases the effective receptive fields (with respect to the first layer.)

CNNs parameterization

Setting up a convolution layer requires choosing

- Filter size: $n \times n$
- #output channels: C
- Stride: s
- Padding: p

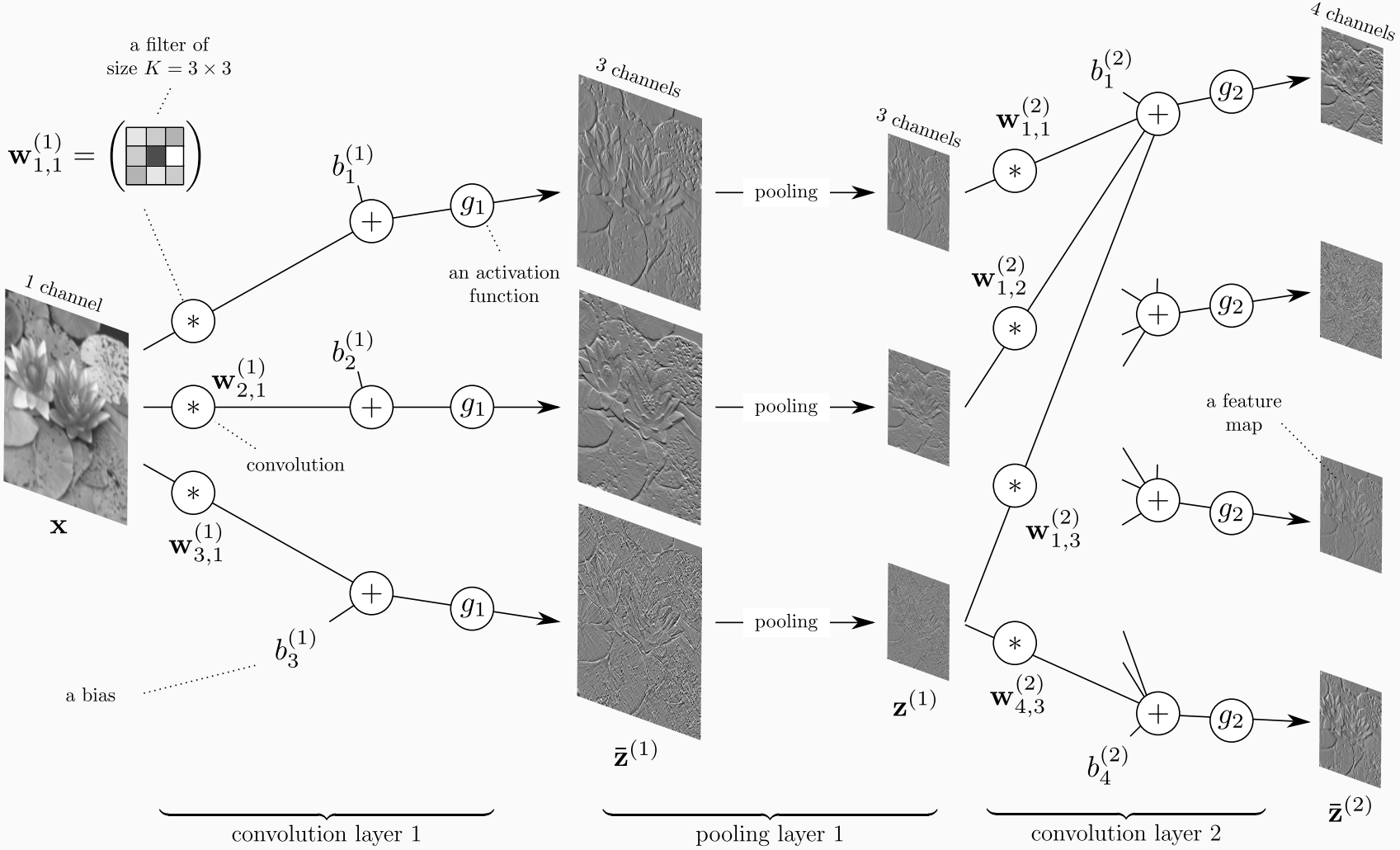
The filter weights $\boldsymbol{\kappa}$ and the bias b are learned by backprop.

Setting up a pooling layer requires choosing

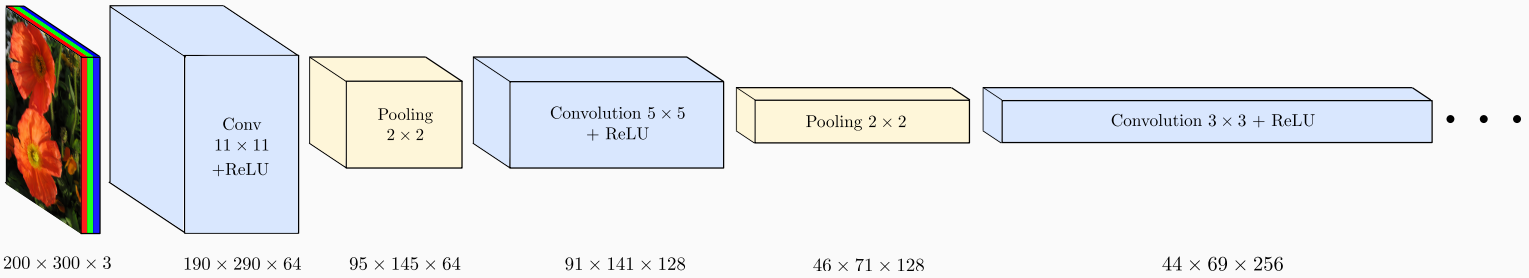
- Pooling size: $z \times z$
- Aggregation rule: max-pooling, average-pooling, ...
- Stride: s
- Padding: p

No free parameters to be learned here.

All concepts together



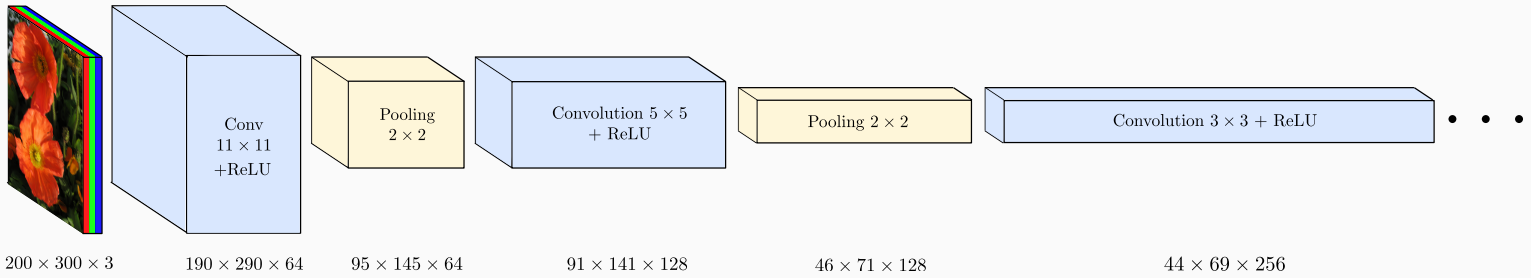
All concepts together with tensor representation



CNN: Alternate:

Conv + ReLU + pooling

All concepts together with tensor representation



CNN: Alternate:

Conv + ReLU + pooling

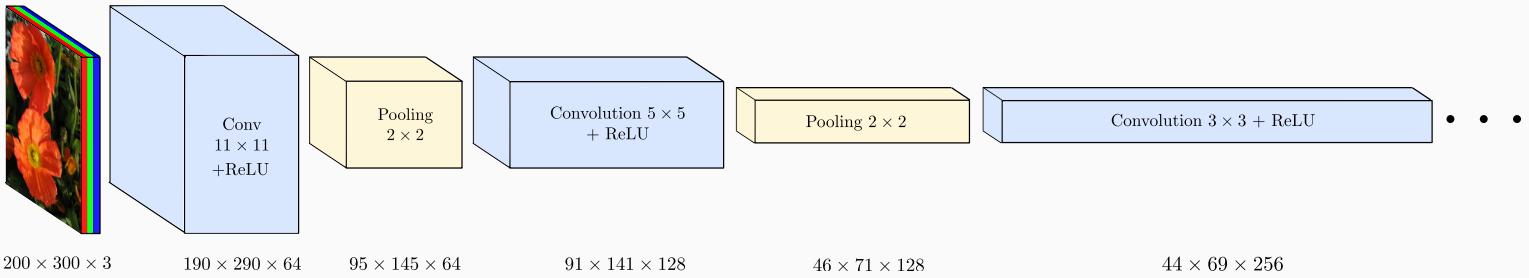
End of network:

Plug a standard neural network:

Fully connected hidden layers

(linear) + ReLU

All concepts together with tensor representation



CNN: Alternate:
Conv + ReLU + pooling

End of network:
Plug a standard neural network:
Fully connected hidden layers
(linear) + ReLU

Full network:

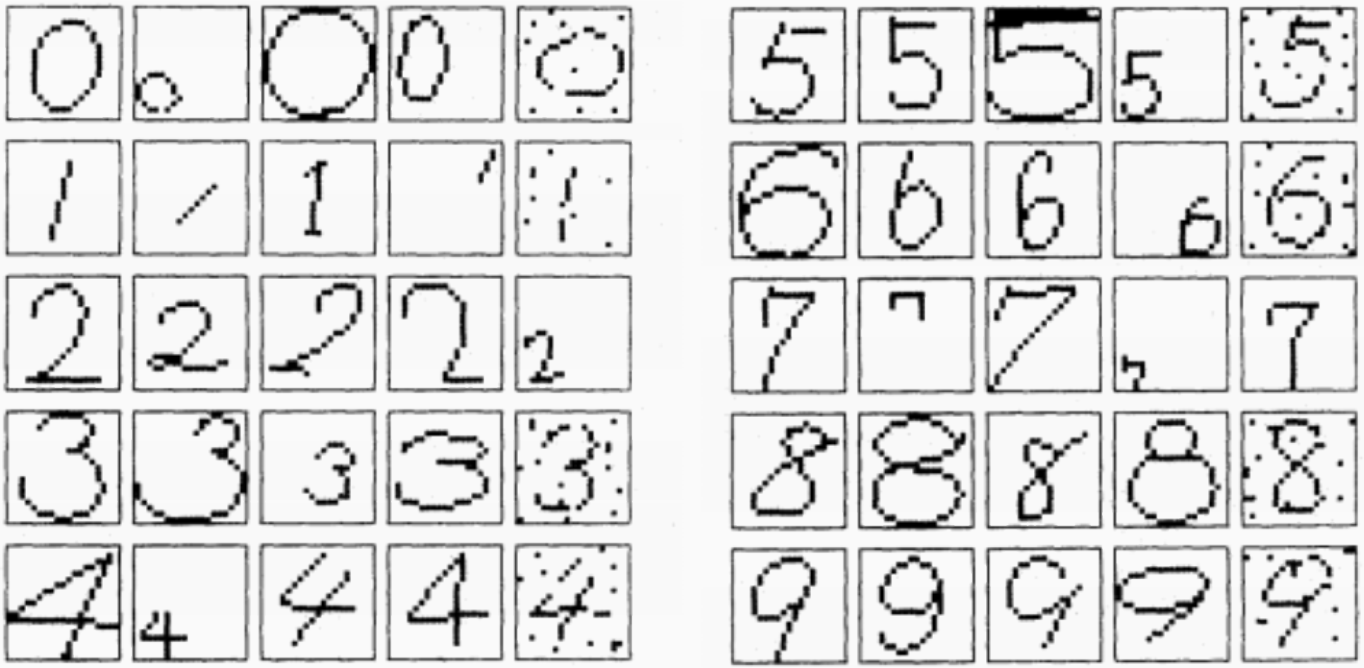
- **CNN:** Extract features specific to spatial data
- **Fully connected part:** Use CNN features for specific regression/classification task
- **Training:** Learn regression/classification and feature extraction **jointly**

Successful CNNs architectures

Neocognitron (Fukushima, 1979, 1980, 1987)

In the history of Optical Character Recognition (OCR), first system working

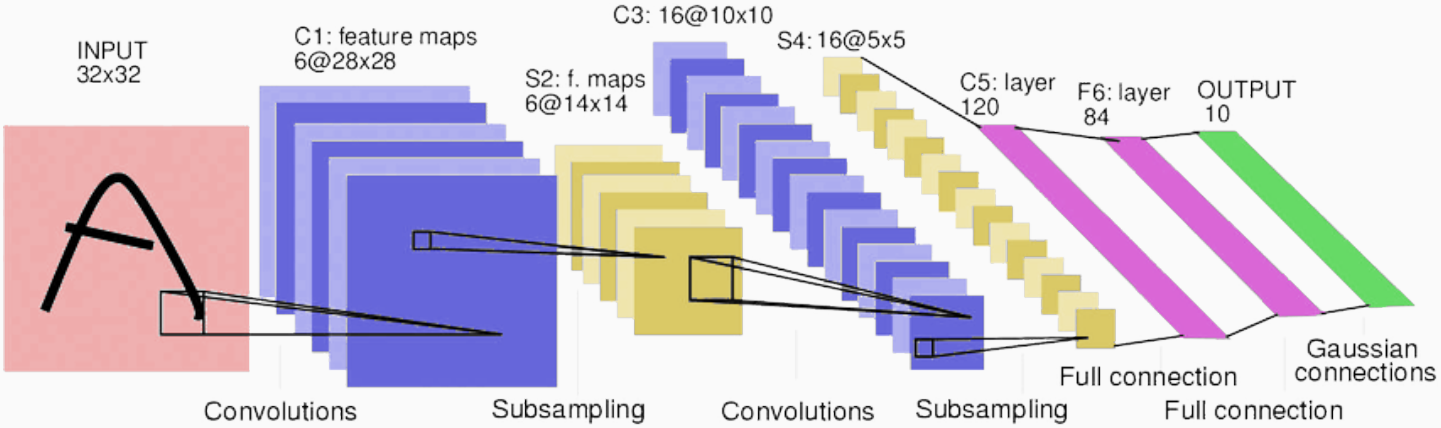
- regardless of where the digits were placed in the field of view,
- high degree of tolerance to distortion of the character,
- fairly insensitive to the size of the character.



Some digits correctly recognized

Successful CNNs architectures

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)



- Classifier for 32×32 hand-written digits,
- Was used for reading $\approx 10\%$ of the checks in North America,
- Demo: `http://yann.lecun.com/exdb/lenet/`,
- Used hyperbolic tangent connection after convolutions,
- Plugged a linear classifier using Gaussian connections (RBF kernels):

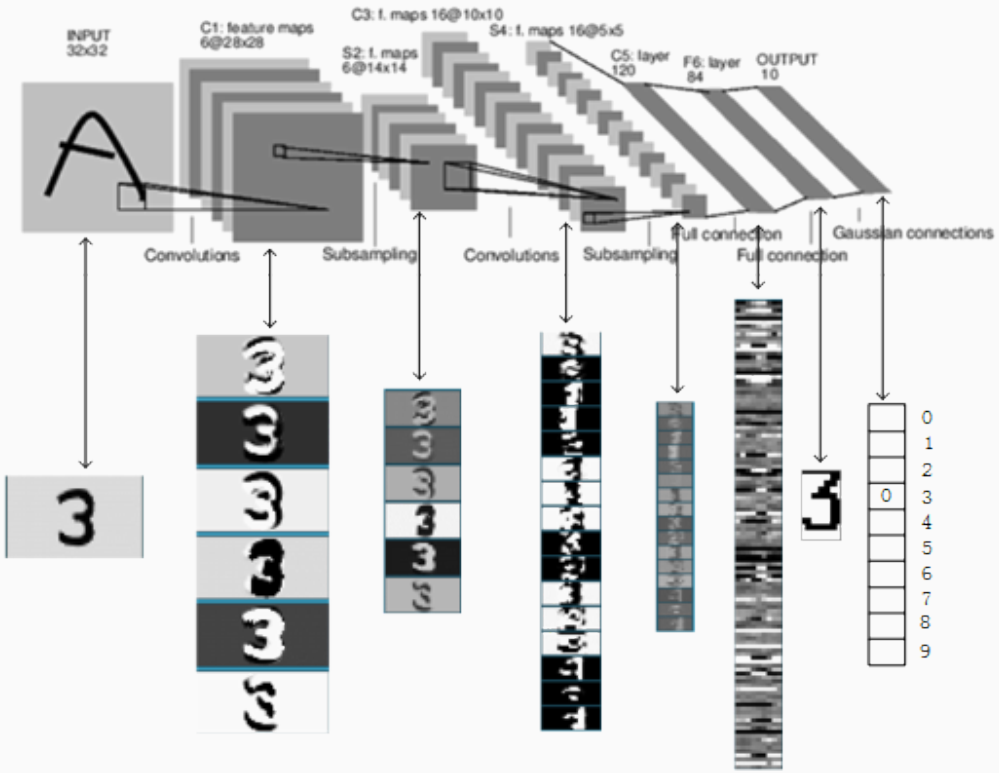
$$y_j = \sum_i (h_i - w_{ij})^2 \text{ instead of classical dot-products } y_j = \sum_i w_{ij} h_i.$$

First deep architecture (5 layers) that was successfully trained.

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)

- Trained on MNIST:
  - 28×28 images,
  - 60,000 for training,
  - 10,000 for testing.
- Used data augmentation:
  - generated random distortions,
  - extended to 32×32px.
  - 600,000 training samples,
- Backprop with SGD:
  - only 20 epochs,
  - no mini-batches,
  - 60,000 parameters,
  - about 3 days (at that time).

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)



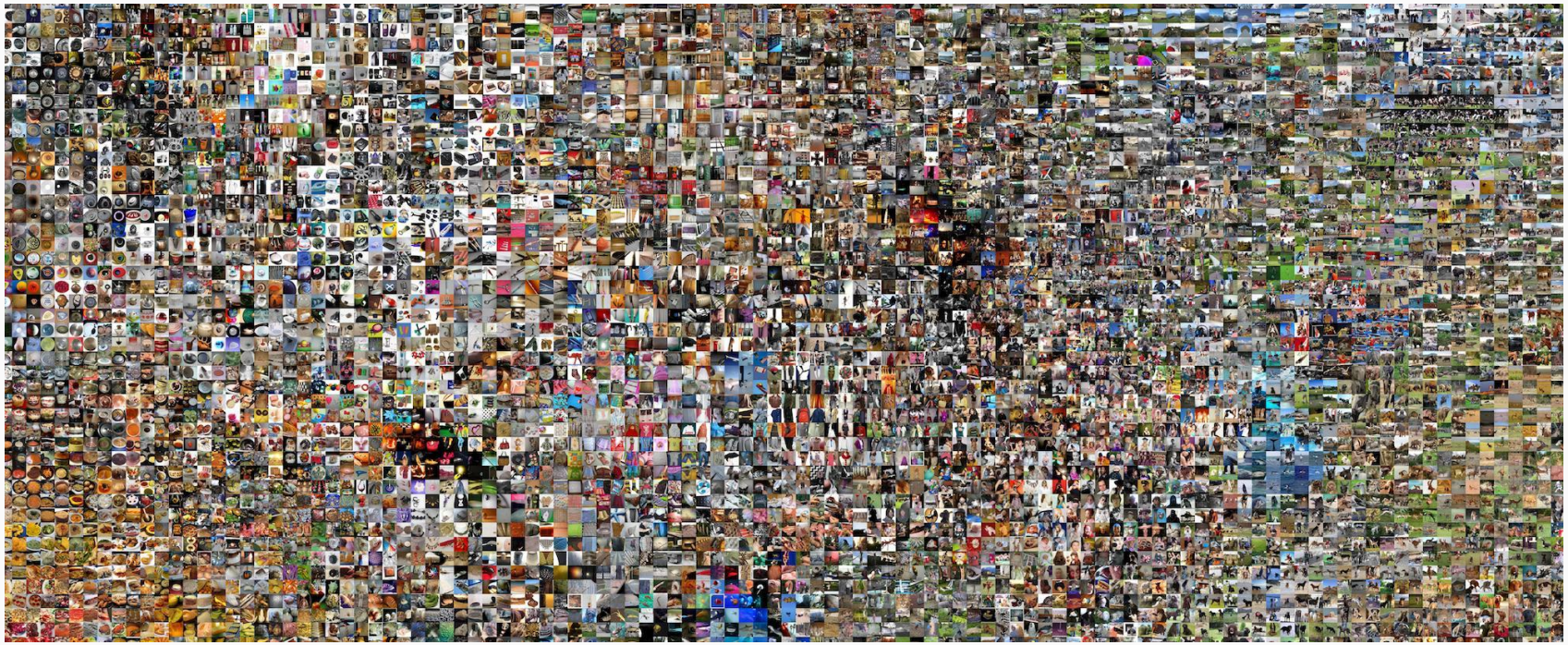
What did it learn? Look at the evolution of the features in the network.

LeNet-5 (LeCun, Bottou, Bengio and P. Haffner, 1998)

Made only 82 errors among the 10,0000 samples of the testing dataset.
Nowadays, the best results on MNIST are of 23 errors (Cireșan *et. al*, 2012).

But with the proliferation of multimedia, MNIST challenge became outdated.

ImageNet



ImageNet:

- 12 million labeled images,
- 22,000 classes,
- Labeled by crowd-sourcing (Amazon Mechanical Turk).

ImageNet challenge (ILSVRC)

ILSVRC:

- Annual challenges since 2010,
- Limited to 1,000 classes,
- 1.2 million of 256×256 images for training,
- 50,000 images for validation and 100,000 for testing.

ImageNet classification challenge (ILSVRC)

Ranking: top-5 error

Make five guesses – Correct if the desired label is within these guesses.

top-5 error rate: percentage of incorrect such answers.

why? desired labels can be ambiguous

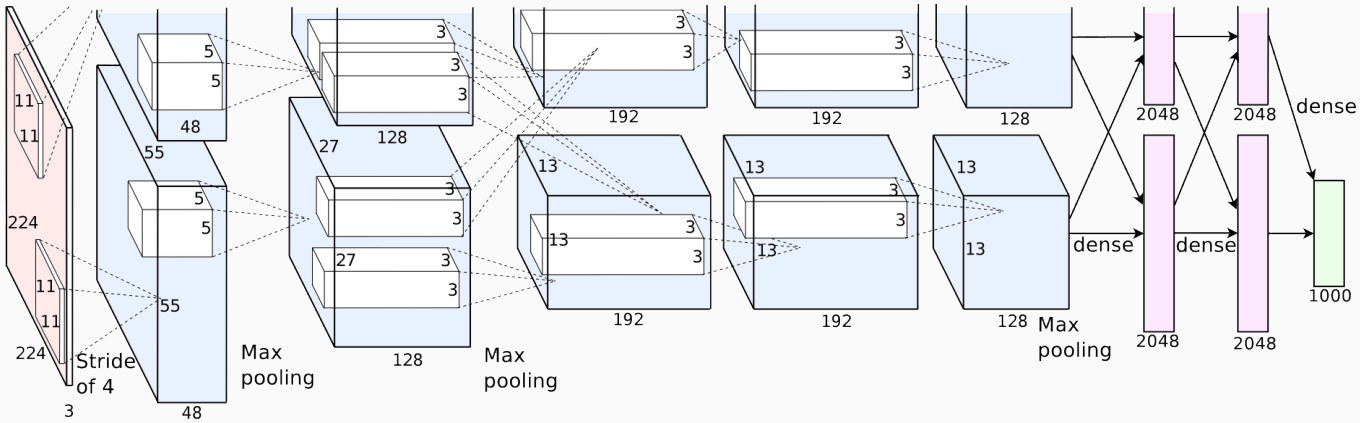
(a) Siberian husky

(b) Eskimo dog

And the 2012 winner was...

Successful CNNs architectures

AlexNet (Krizhevsky, Sutskever, Hinton, 2012)



- Similar to LeNet-5, a bit deeper (8 layers), but much larger.
- Use ReLU after convolutions, use softmax for the output layer.

Introduced concepts:

- **Specific GPU architecture:** split the channels in two stages in order to be trained simultaneously on two Nvidia Geforce GTX 580 GPUs.
- **Local Response Normalization:** after applying the first 2 filter banks, they normalize each feature map with respect to their adjacent ones.
- **Overlapping pooling:** use striding also after pooling with $z = 3$ and $s = 2$.

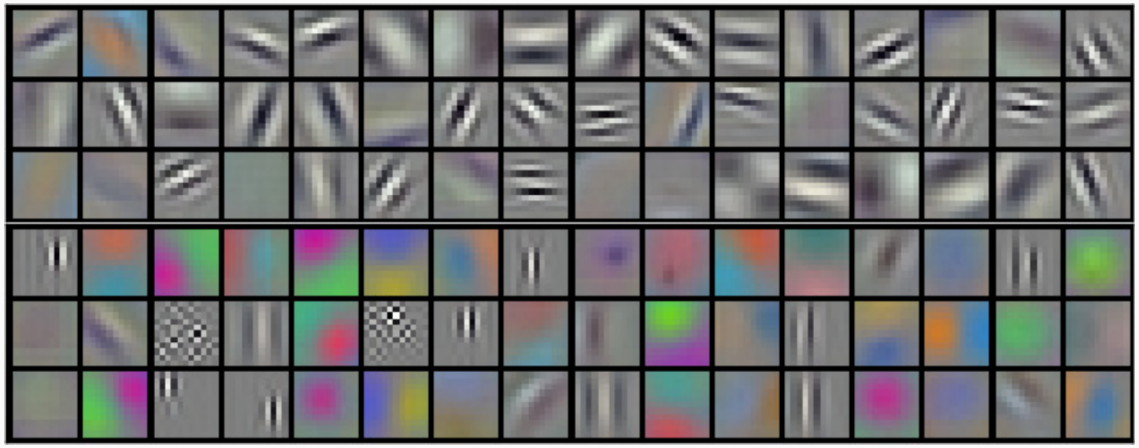
AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

Other settings:

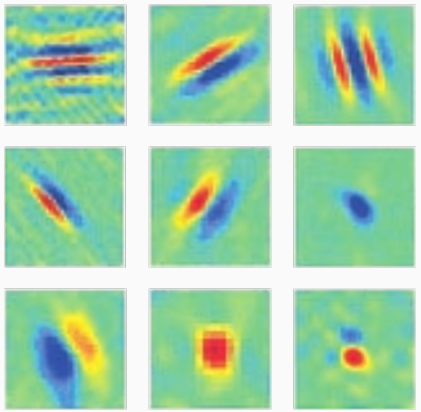
- **Number of parameters**: 60 million,
- **Data augmentation:** horizontal reflection, 224×224 subimages from 256×256 images, altering RGB channels. At test time, average the prediction obtained on 10 subimages extracted from the original one.
- **Dropout:** used for hidden neurons with probability $.5$.
- **Optimization:** SGD with mini-batch size 128 + weight decay + momentum $\rightarrow$ Took 6 days for 90 epochs.
- **Combination:** Trained 7 such networks and average their predictions (some of them were pretrained on other datasets and fine-tuned.)

AlexNet (Krizhevsky, Sutskever, Hinton, 2012)

What did it learn? Look at the filters that have been learned.



receptive fields estimated by reverse correlation:

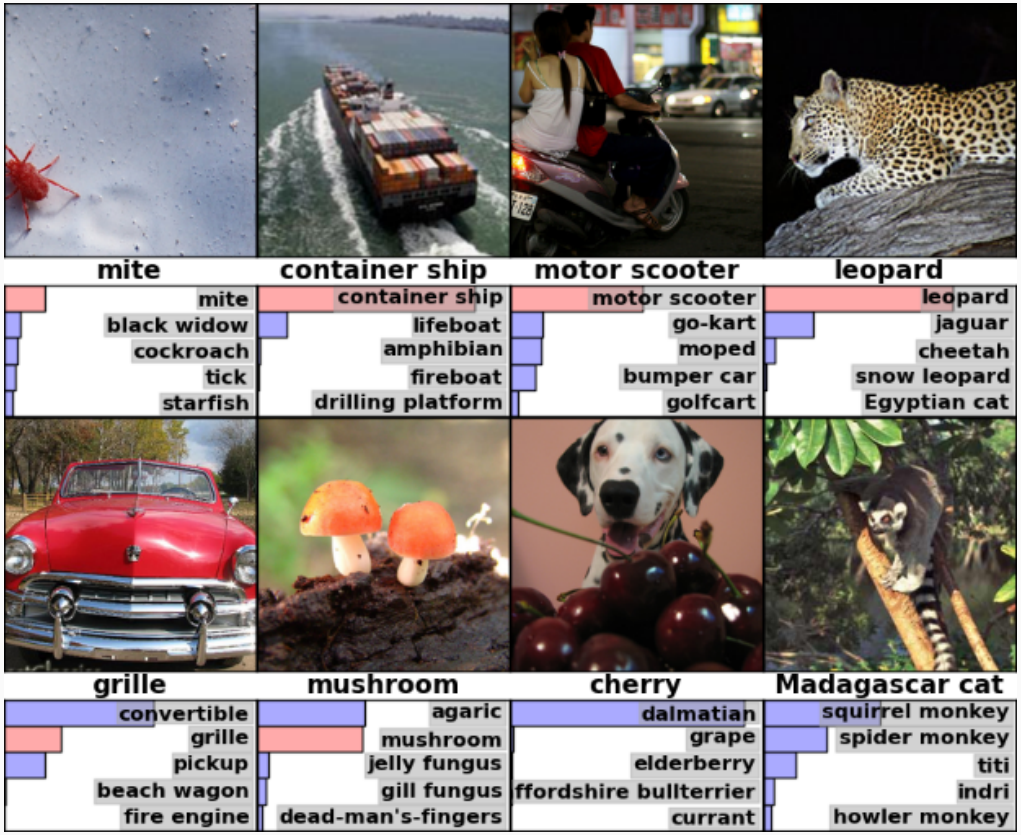


(left) 96 filters learned in the first convolution layer in AlexNet.

Many filters turn out to be edge detectors, similar to Gabor filters, common to human visual systems (right).

Successful CNNs architectures

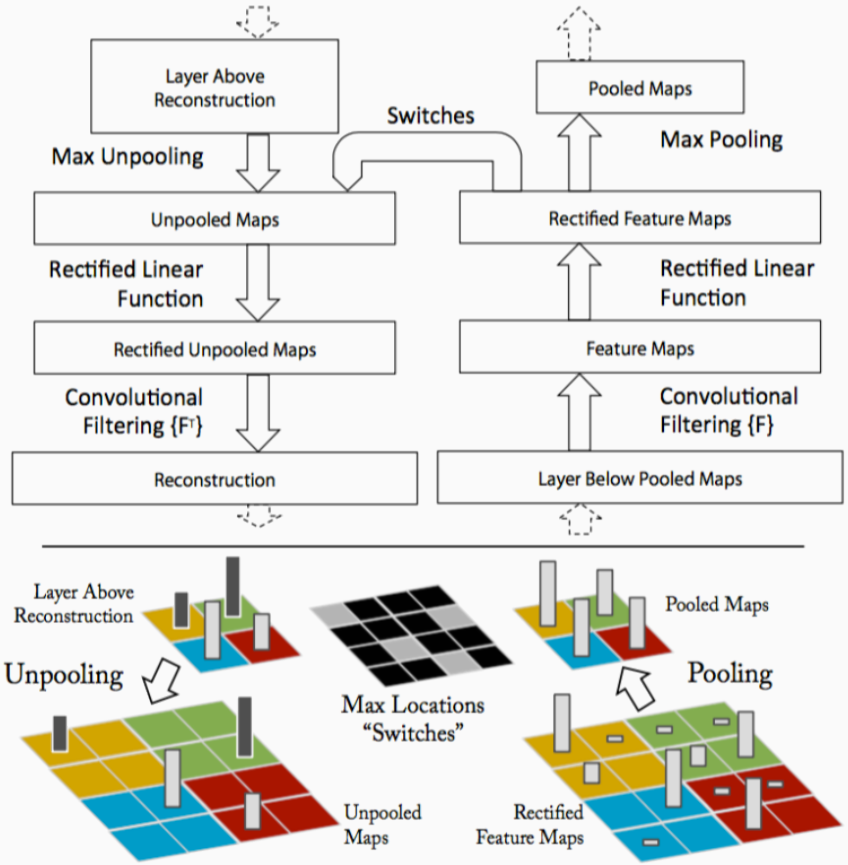
AlexNet (Krizhevsky, Sutskever, Hinton, 2012)



- Won ILSVRC 2012 by reducing the top-5 error to 16.4%.
- The second place, which was not a CNN, was around 26.2%.

ZFNet (Zeiler & Fergus, 2013)

Visualizing the evolution of features, or the learned filters is not informative enough to understand what has been learned.

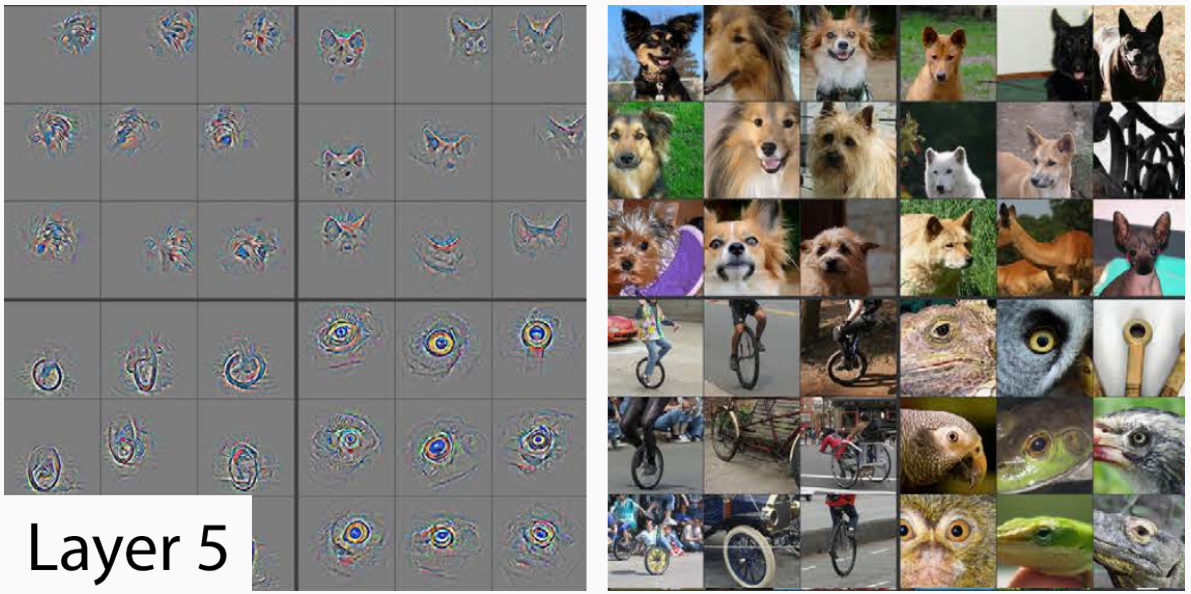


Introduced concept

Visualization with DeconvNet:
Offered a way to project an intermediate feature back to the image: starting from this feature, browse the network backward, replace pooling by unpooling, and convolutions by transposed convolutions (*i.e.*, flip the filters, aka, deconvolution layer).

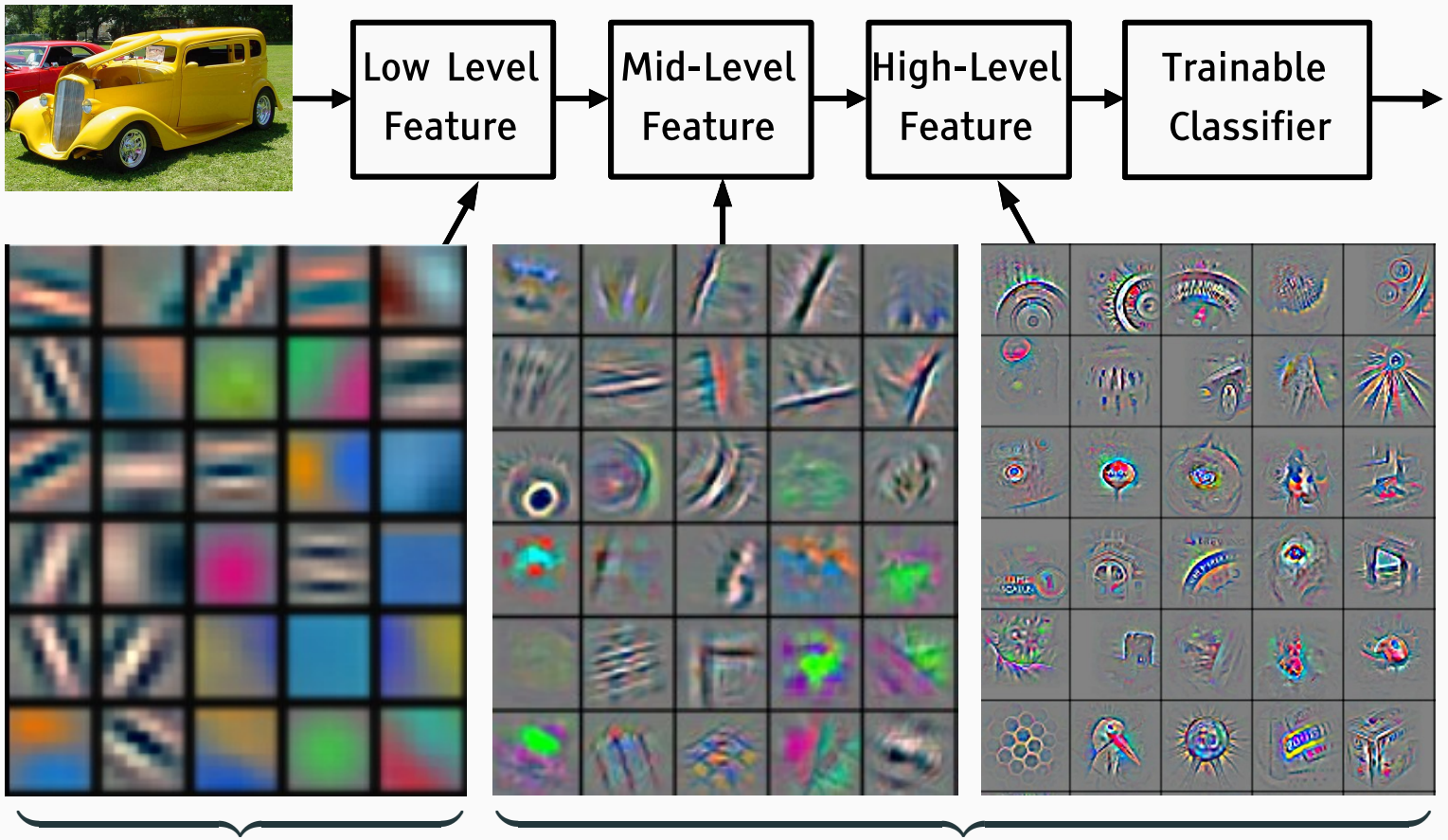
We will go back to this later.

ZFNet (Zeiler & Fergus, 2013)



- Look at the intermediate features of AlexNet at each layer,
- Inspect the features and make some changes accordingly:
  - Layer 1: Conv 11×11 s4 $\rightarrow$ Conv 7×7 s2,
  - Layer 2: Conv 5×5 s1 $\rightarrow$ Conv 5×5 s2,
  - Use also Local Response Normalization,
  - Do not use a split architecture in two stages for GPU.

ZFNet (Zeiler & Fergus, 2013)

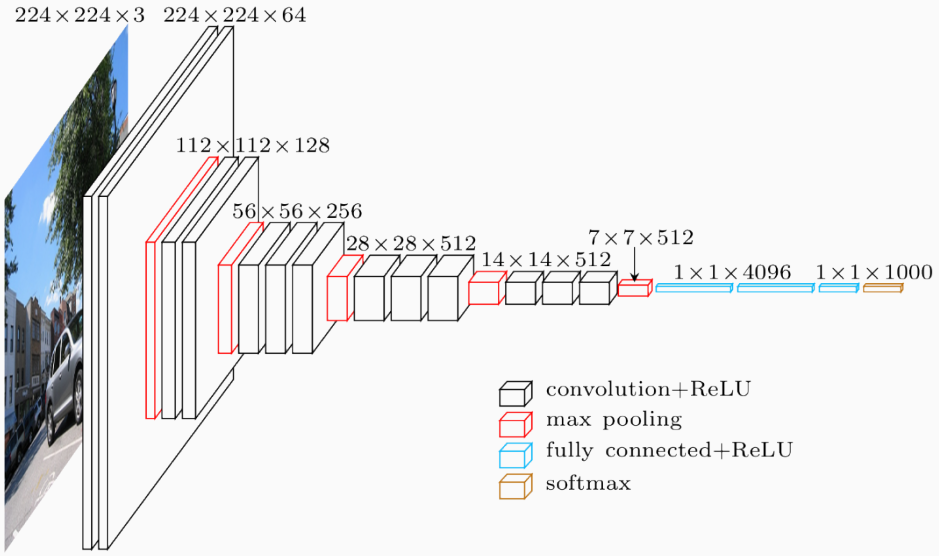


That's how we can produce such beautiful illustrations!

ZFNet (Zeiler & Fergus, 2013)

- Other settings:
  - **Number of parameters**: < 60 million,
  - **Data-augmentation**: crops and flips,
  - **Dropout**: used for FC layers with probability $.5$,
  - **Filter normalization**: when their norm exceed a threshold,
  - **Optimization**: SGD with mini-batch size 128 + momentum
    $\rightarrow$ Took 12 days for 70 epochs on a single Nvidia GTX 580 GPU.
  - **Combination:** Average the prediction of 5 such CNNs and another one with a slightly different architecture:
    - Layer 3,4,5: #Channels 384, 384, 256 $\rightarrow$ 512, 1024, 512.
- Won ILSVRC 2013 with 11.7% top-5 error (AlexNet: 16.4%).

VGG (Simonyan & Zisserman, 2014)



Introduced concept

Deep and simple:

- 16 conv filters, 3×3 s1,
- 5 max pool, 2×2 s2,
- 3 FC layers,
- No need of local response normalization.

Why does it work?

- Two first 3×3 conv layers: effective receptive field is 5×5,
- Three first 3×3 conv layers: effective receptive field is 7×7,
- Why is it better than ZFNet which uses 7×7?
  - More discriminant: 3 ReLUs instead of 1 ReLU,
  - Less parameters: $3 \times (3 \times 3) = 27$ vs $1 \times (7 \times 7) = 49$.
- Next, apply max-pooling and the effective receptive field double!

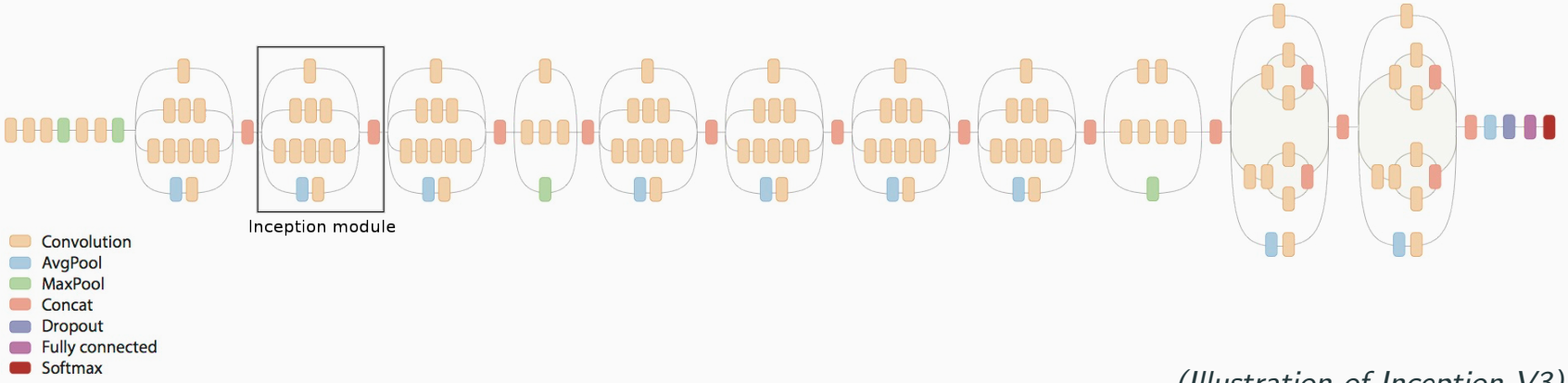
VGG (Simonyan & Zisserman, 2014)

Other settings:

- **Number of parameters**: 140 million,
- **Data-augmentation**: crops, flips, RGB color shift and scaling jittering.
- **Dropout**: used for the first two FC layers with probability $.5$,
- **Optim**: SGD with mini-batch size 256 + momentum + weight decay
  $\rightarrow$ Took about 3 weeks for 74 epochs on 4 Nvidia Titan Black GPUs.
- **Combination:** Combined 7 such CNNs.

Runner-up of ILSVRC 2014 with 7.3% top-5 error (ZFNet: 11.7%).
Not winner but very influential: "Make it deep. Keep it simple".

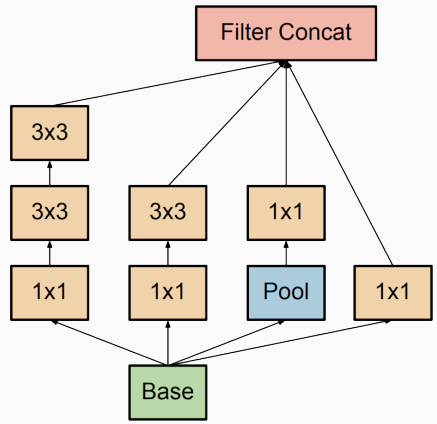
Inception/GoogLeNet (Szegedy *et al.*, 2014), V2 (2015), V3 (2016), V4 (2017)



(Illustration of Inception V3)

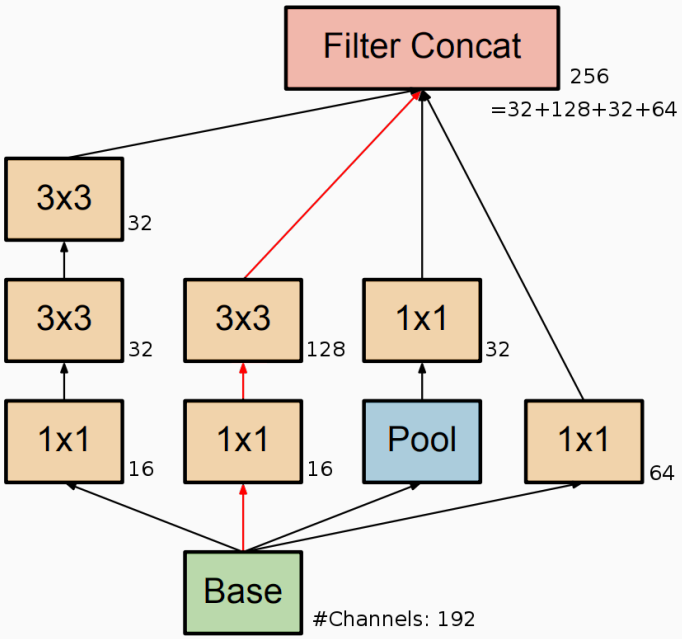
Introduced concept:

- **Inception module:** apply filters of different sizes in parallel and concatenate their answers. Each module captures in parallel details at different scales and has thus different effective receptive fields.
- **Deeper with fewer parameters:** introduced "1×1" convolutions for dimensionality reduction. The first version had 22 layers and 4 million parameters! (AlexNet has 8 layers and 60 million parameters).



(An inception module)

Inception/GoogLeNet (Szegedy *et al.*, 2014), V2 (2015), V3 (2016), V4 (2017)



What are 1×1 convolutions?

- 1×1 (multi) convolutions: outputs are simply element-wise weighted sums of the inputs. Producing less outputs than inputs allows for dimensionality reduction.

Why do they perform dimensionality reduction?

- They can take 192 feature maps and reduce the information to 16 feature maps (with same width and height).

Why do they allow for reducing the number of parameters?

- Consider the branch in red:
  - Without: $192 \times 3 \times 3 \times 128 \approx 221,000$ parameters,
  - With: $192 \times 16 + 16 \times 3 \times 3 \times 128 \approx 21,500$ parameters.

They also decrease by the same factor the computation cost!

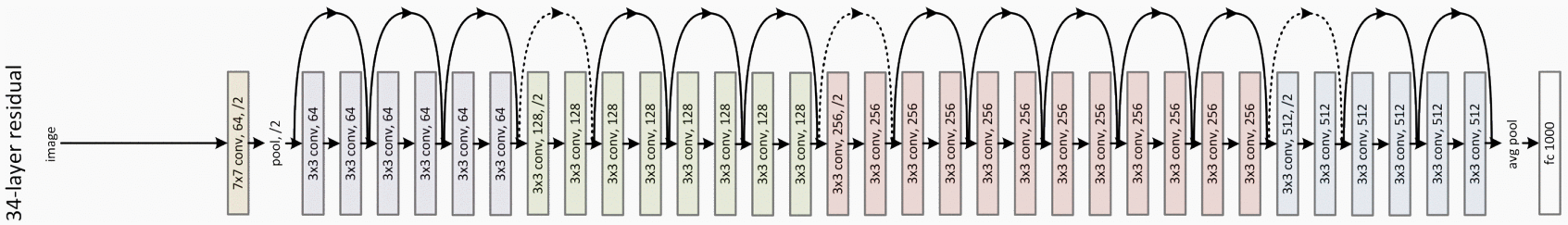
Inception/GoogLeNet (Szegedy *et al.*, 2014), V2 (2015), V3 (2016), V4 (2017)

Other settings:

- **Number of parameters**: 6.8 million (VGG: 140M),
- **Data-augmentation**: crops of different sizes, flips, random interpolations.
- **Dropout**: used for one FC layer with probability $.4$,
- **Optimization**: SGD + momentum + learning rate schedule
  $\rightarrow$ Authors suggest it may take about a week with few GPUs.
- **Combination:** Average the prediction of 7 CNNs over 144 crops per images (shifted, rescaled and flipped).

Won ILSVRC 2014 with 6.7% top-5 error (VGG: 7.3%).

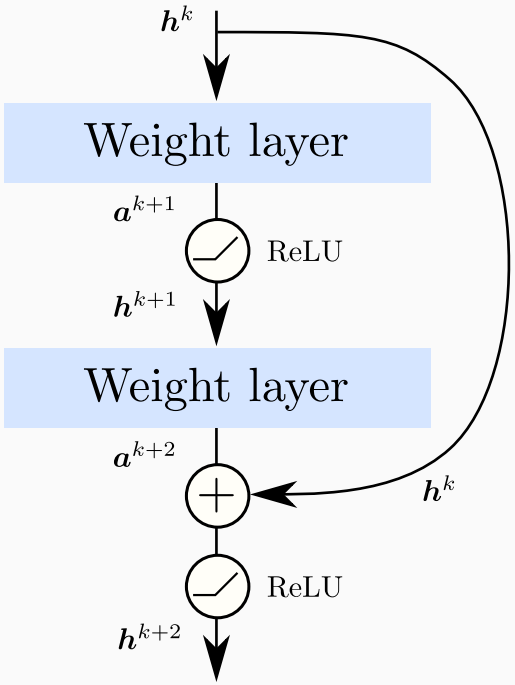
ResNet (Microsoft – He *et al.*, 2016)



(short version only 34 layers here!)

Introduced concept:

- **Be ultra deep:** 152 layers! First layer uses 7×7 convolutions and everything else is 3×3 with only two pooling layers.
- **Residual layer:** learn how to change the input instead of learning what the output should be (using shortcut connections). Refer to the previous class.



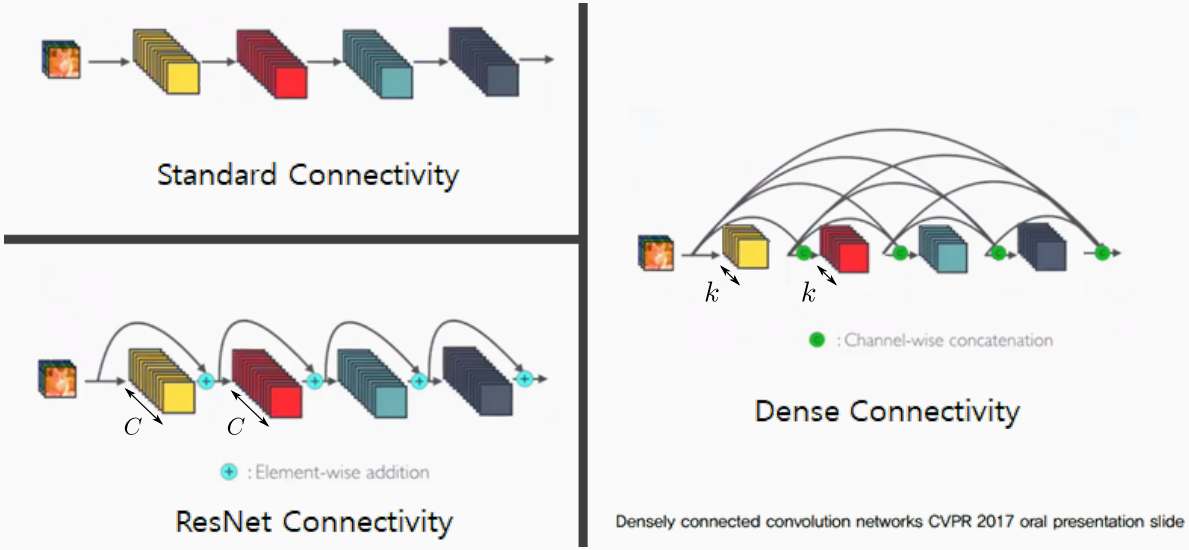
ResNet (Microsoft – He *et al.*, 2015)

Other settings:

- **Number of parameters**: 60 million,
- **Data-augmentation**: resized, cropped, flipped.
- **Batch-normalization**: between convolutions and ReLU (no dropout).
- **Optimization**: SGD with mini-batch size 256 + momentum
  + weight decay + learning rate schedule
  + He initialization (He *et al.*, 2015. Same guy)
  → Trained for about two weeks using 8 GPUs.
- **Combination:** average predictions of 6 CNNs (only two with 152 layers) over ten crops and multiple scales.

Won ILSVRC 2015 with 3.57% top-5 error (GoogLeNet, 6.7%).
First time a system beats human level prediction on ImageNet (about 5.1%).

DenseNet (Huang *et al.*, 2017)



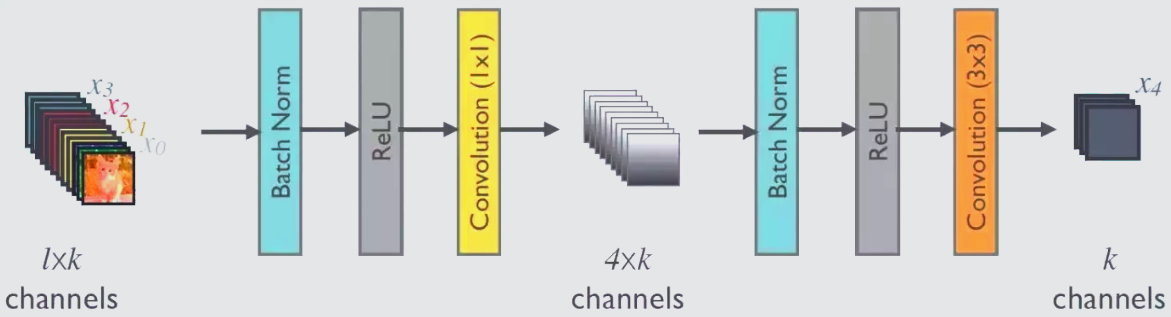
Introduced concept:

- **Dense connectivity:** Each layer at level l received the concatenation of all previous feature maps of size k as input: $l \times k$ channels . Advantages:
  1. aggregate information from different levels of abstraction,
  2. layers can be slim (k small), and the network can be deep (l large),
  3. reduce even more the number of parameters to learn:

$$\text{ResNet: } C \times C \quad \text{vs} \quad \text{DenseNet: } l \times k \times k \quad \text{with} \quad k \ll C$$

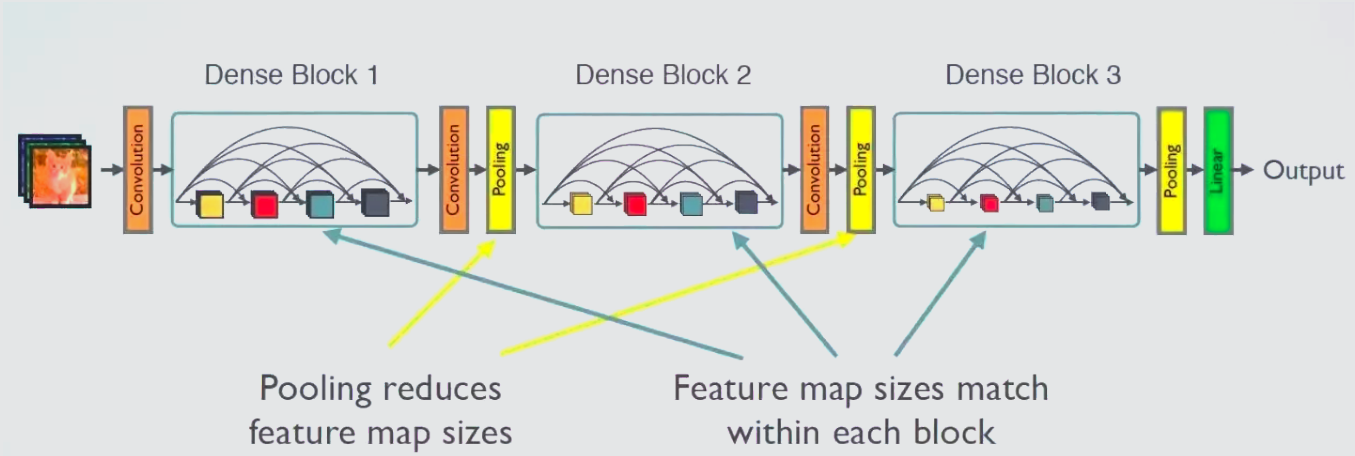
DenseNet (Huang *et al.*, 2017)

Since l can be large, use 1×1 convolution to reduce dimension



Since each layer needs to have the same spatial dimension.

$\rightarrow$ Use a succession of blocks with dense connectivity separated by pooling:



DenseNet (Huang *et al.*, 2017)

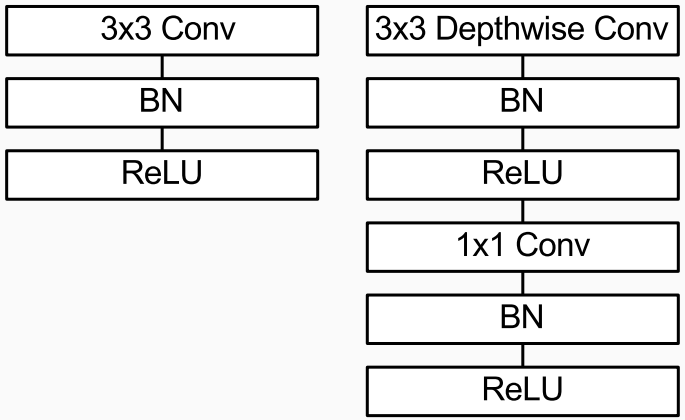
Other settings:

- **Depth:** 201 layers,
- **Number of parameters**: 20 million,
- **Data-augmentation**: crops and flips.
- **Batch-normalization**: BN, next ReLU and next Conv.
  Note that ReLU is used before Conv → pre-activation (He et al., 2016).
- **Optim**: SGD with mini-batch size 256 and 90 epochs + Nesterov + weight decay + learning rate schedule.

Same top-1 error rates as ResNet but with 3 times less parameters.
Won the CVPR 2017 best paper award.

MobileNet (Google – Howard *et al.*, 2016-17)

Goal: to develop small and fast CNNs for mobile vision applications.



Introduced concept: to factorize $n \times n$ conv. with M inputs and N outputs by:

1. **Depthwise convolution:** one filter per channel $n \times n$ ($\times M$).
2. **Pointwise convolution:** 1×1 conv. to generate N new features from M.

Separable convolution

MobileNet (Google – Howard *et al.*, 2016-17)

Other settings:

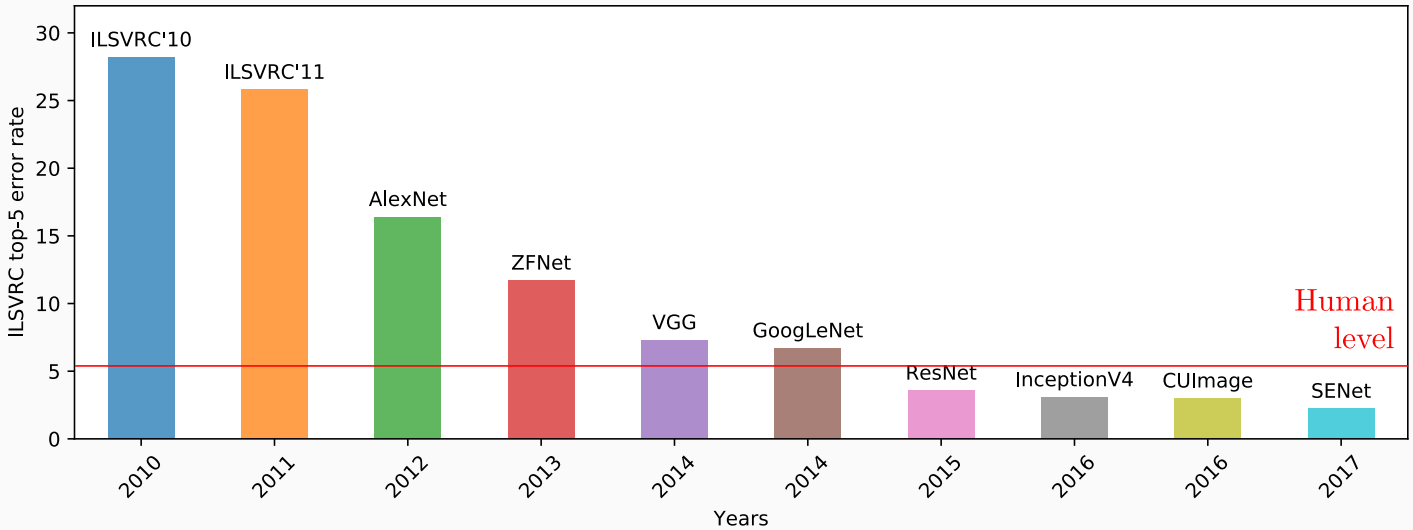
- **Depth:** 28 layers (depthwise + pointwise counted as 2)
- **Number of parameters**: 4.2 million (GoogleNet: 6.8M),
- **Data-augmentation**: less than other models,
- **Batch-normalization**: after each convolution,
- **Optimization**: SGD with no weight-decay.

Similar performance than GoogleNet but 2/3 less parameters.

Offers the possibility to reduce the width and resolution to trade accuracy and latency for specific devices.

Latest CNNs

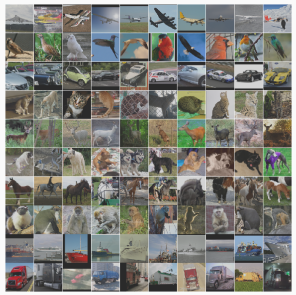
- **Inception v4** (Szegedy et al, 2016) – top-5 error: 3.08%
- **CUImage** (Zeng *et al.*, 2016) – top-5 error: 2.99% (winner ILSVRC 2016)
- **Xception** (Google – Chollet, 2016) – Similar to MobileNet
- **FractalNet** (Larson *et al.*, 2016) – Ultra-deep alternative to ResNet
- **SqueezeNet** (Iandola, *et al*, 2016) – as AlexNet but 50× fewer parameters
- **SENet** (Hu et al, 2017) – top-5 error: 2.23% (winner ILSVRC 2017)
- and many others. . .



Other ways to compare them?

Other standard datasets

CIFAR-10



STL-10

SVHN

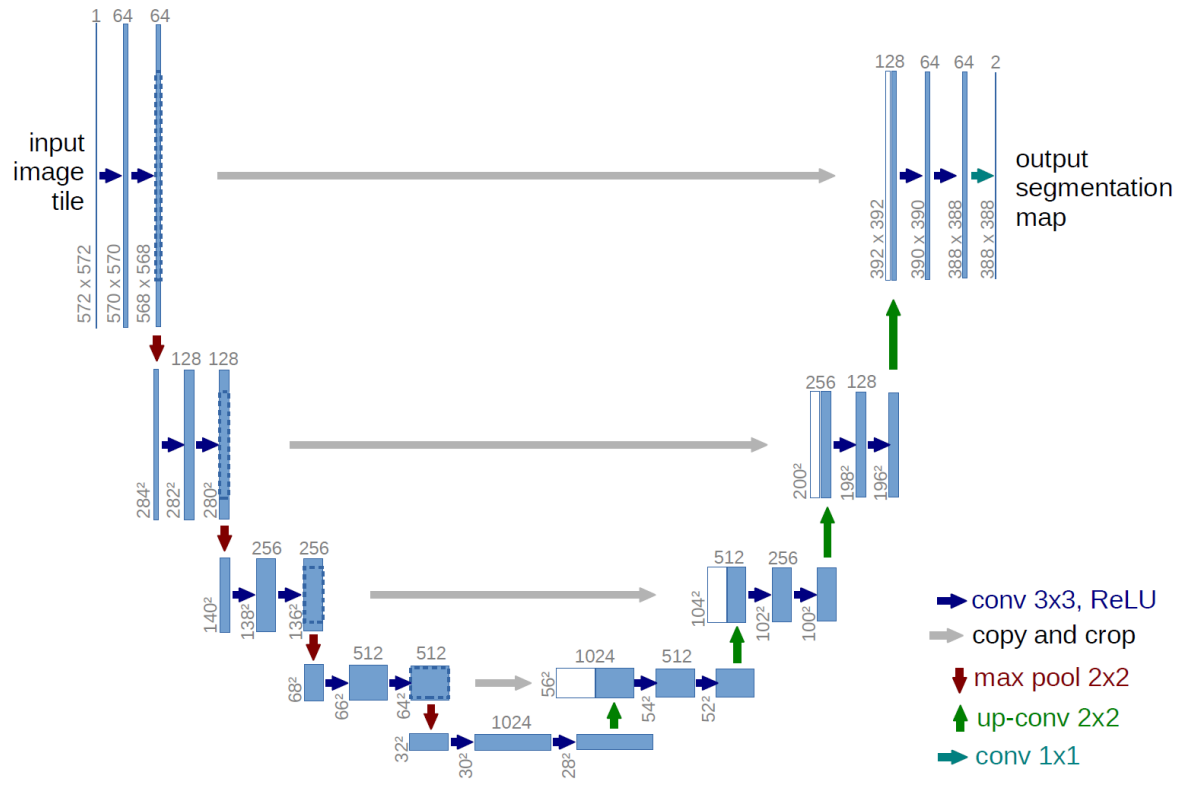
- CIFAR-10 datasets
  - 60000 32x32 color images,
  - 10 classes, with 6000 images per class,
  - 50000 training images and 10000 testing images.
- CIFAR-100 datasets
  - Same but with 100 classes,
  - Each image as a fine and coarse label.
- STL-10
  - Similar as CIFAR-10 but images are 96x96.
- SVHN (Street View House Numbers)
  - Similar to MNIST but with 600,000 color digit images.

Evaluation on these datasets usually comes with other metrics.

Image segmentation using CNN

Network for segmentation

U-net architecture (O. Ronneberger et al, 2015)



- Idea: Classify each pixel
- Condense spatial information as for image classification.
- Re-affine spatially the classification step by step with mirror upsampling steps (transpose of conv2D with padding) and concatenation.

Network for segmentation

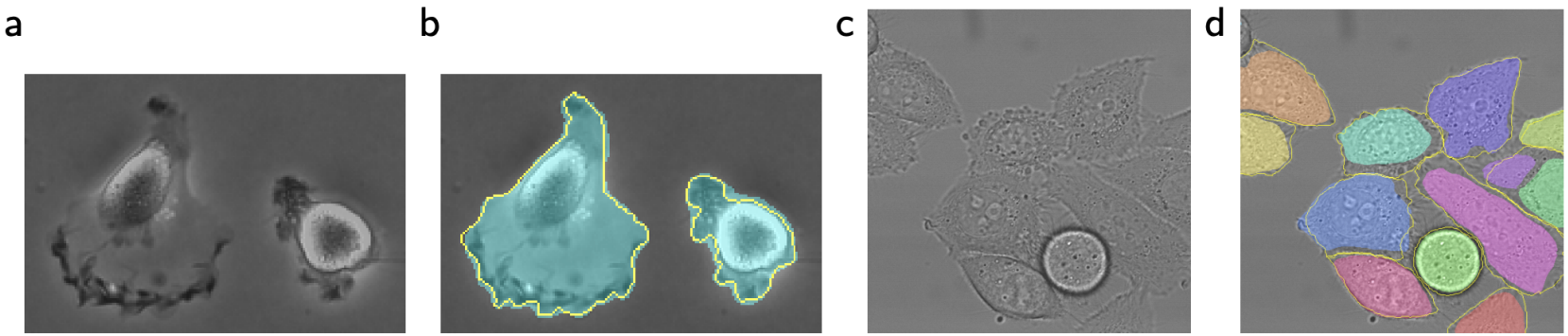
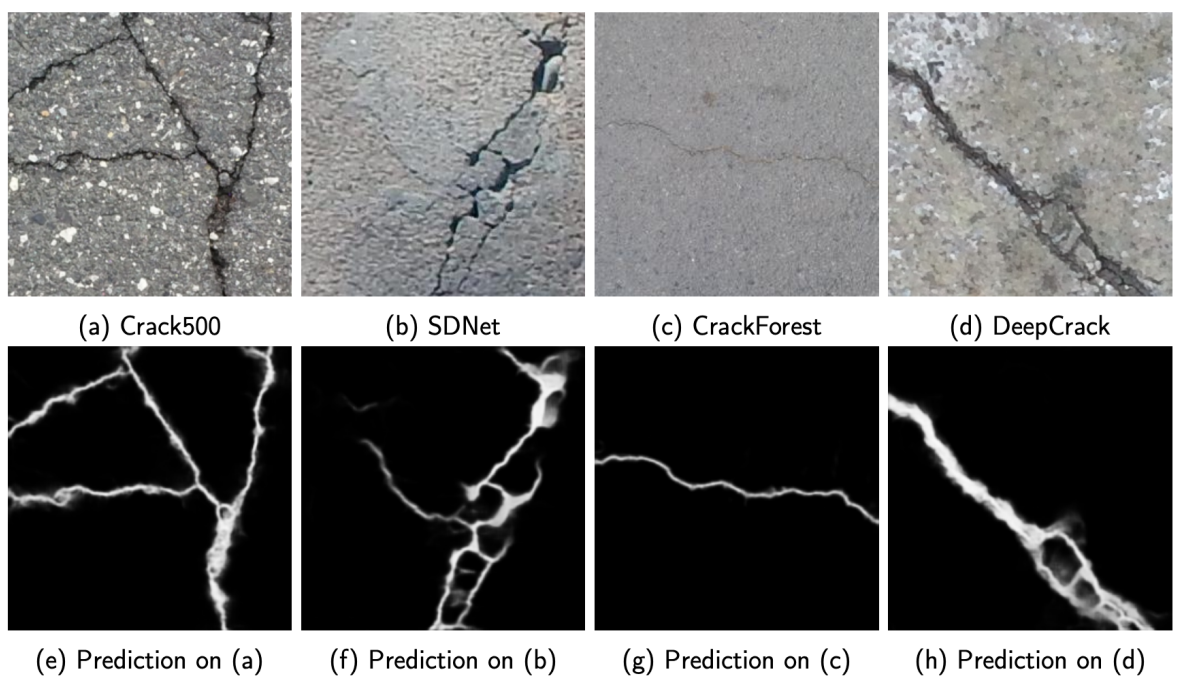


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

- Improved state-of-the-art in cell-tracking.
- Can be extended to very different contexts provided enough labeled data.

U-net architecture (O. Ronneberger et al, 2015)



(S. Drouillet, 2020)

- Example usage: Crack detection
- The network outputs the probability that each pixel belongs to a crack.

Questions?

Slides from Charles Deledalle

Sources, images courtesy and acknowledgment

K. Chatfield

P. Gallinari,

C. Hazırbaş

A. Horodniceanu

Y. LeCun

V. Lepetit

L. Masuch

A. Ng

M. Ranzato

Practical session – Convolutional neural networks

Go through the PyTorch tutorial:
"Deep Learning with PyTorch: A 60 Minute Blitz"

`https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`

Each part is a notebook with a "Run in Google Colab" button.

Chalenge: Adapt the tutorial to the STL10 database (larger images).

`https://pytorch.org/docs/stable/torchvision/datasets.html#torchvision.datasets.STL10`