Deep Learning Basics

Dr. Pedro Casas

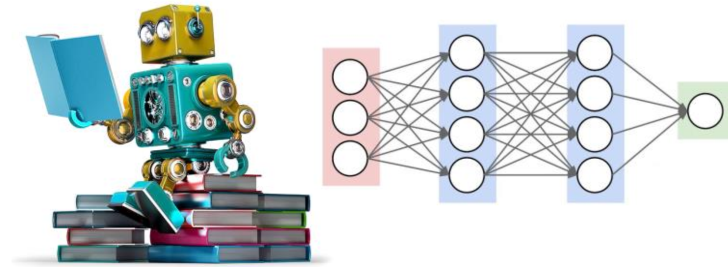
Data Science @Digital Insight Lab

AIT Austrian Institute of Technology @Vienna

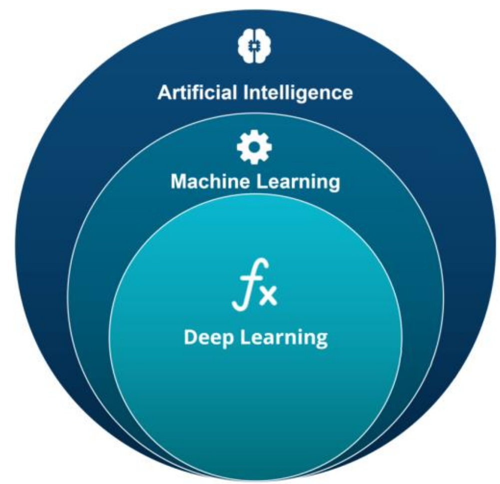
IIE – FING – ARTES

December 2019

Deep Learning Basics

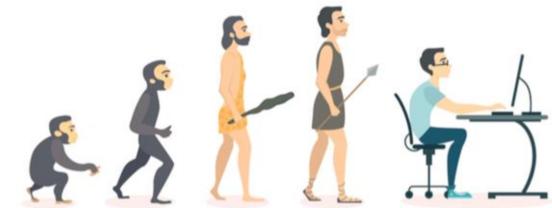


- Deep Learning 101
- Definitions and main Components
- Training Deep Neural Networks
- Convolutional Neural Networks (CNNs)

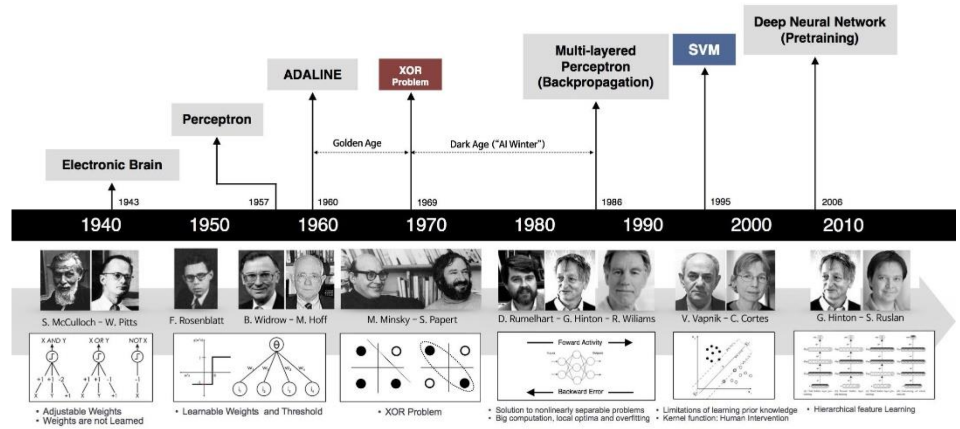


Deep Learning (http://www.deeplearningbook.org), by I. Goodfellow, Y. Bengio, A. Courville

Deep Learning – a bit of History



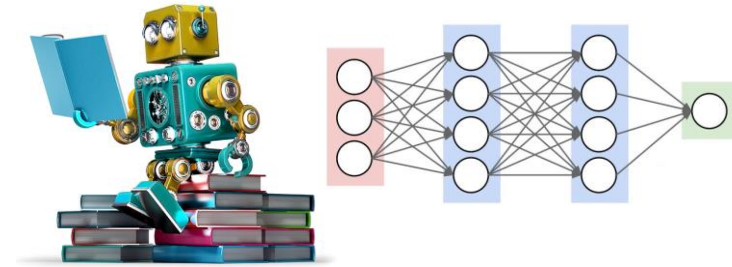
- 1943: Neural networks
- 1957: Perceptron
- 1974-86: Backpropagation, RBM, RNN
- 1989-98: CNN, MNIST, LSTM, Bidirectional RNN



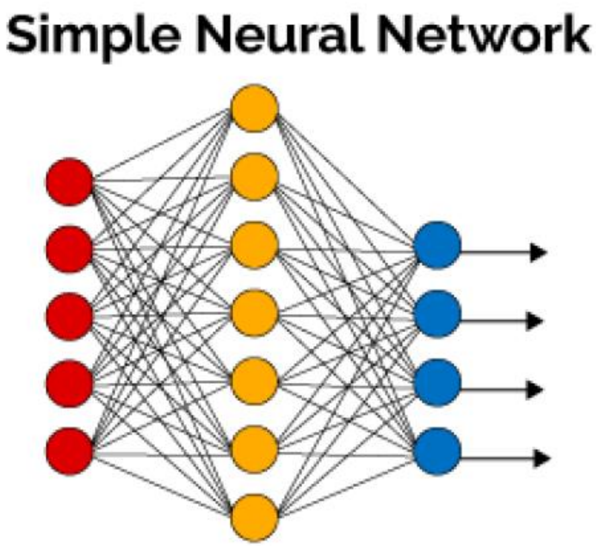
- 2006: "Deep Learning"
- 2009: ImageNet
- 2012: AlexNet, Dropout
- 2014: GANs
- 2014: DeepFace

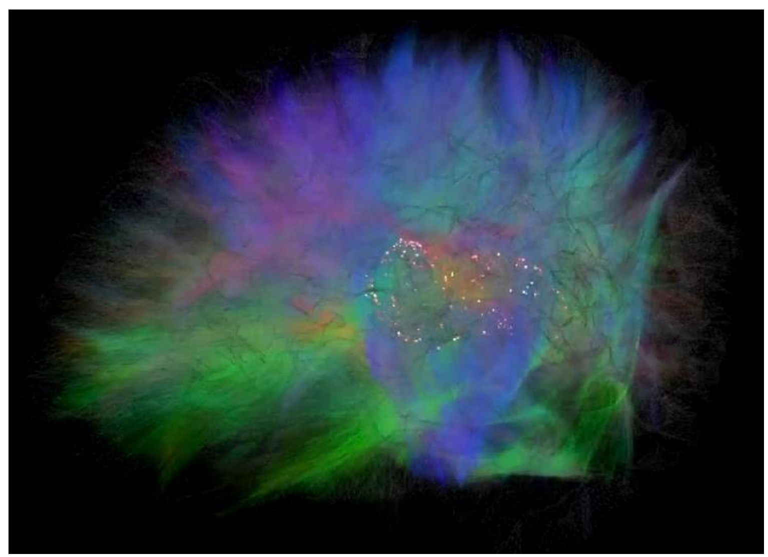
- 2016: AlphaGo
- 2017: AlphaZero, Capsule Networks (CapsNets)
- 2018: BERT transformers

Deep Learning 101



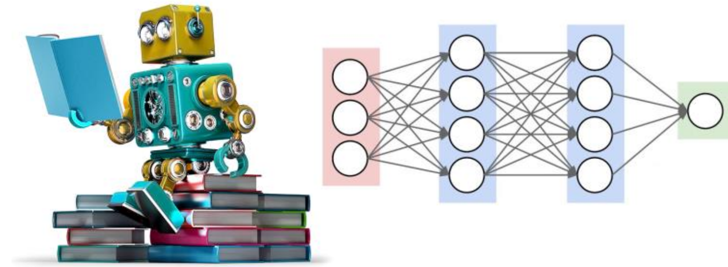
- What is Deep Learning (DL)?
- **Recall:** a feedforward network with a single layer is sufficient to represent (approximate) **any function**...
- ...but **the layer may be infeasibly large** and may fail to learn and generalize correctly...





- Visualization of the human brain:
- **3% of the human brain neurons**
- **0.0001% of neural synapses**

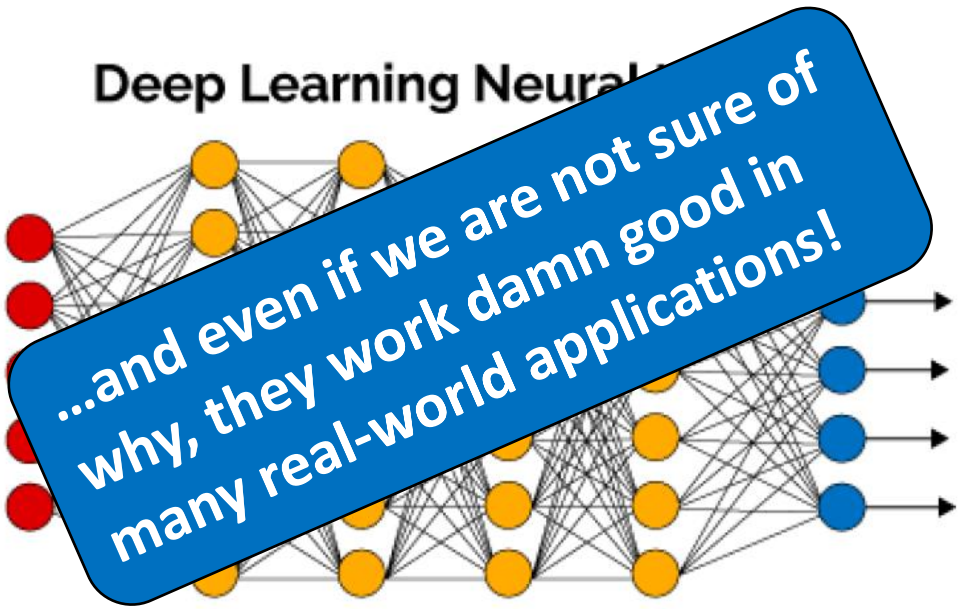
Deep Learning 101



- What is Deep Learning (DL)?

- **In simple terms:** using a neural network with **several layers** of nodes between input and output.

- **Deep Neural Networks (DNNs):**

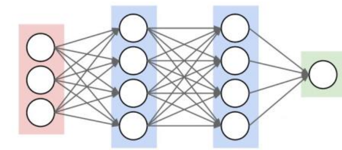


- exceptionally **effective at learning patterns.**

- **hierarchical structure**…

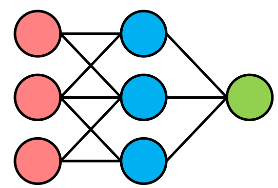
- …can learn the hierarchies of **knowledge** that seem to be **useful in solving real-world problems**…

Deep Learning 101

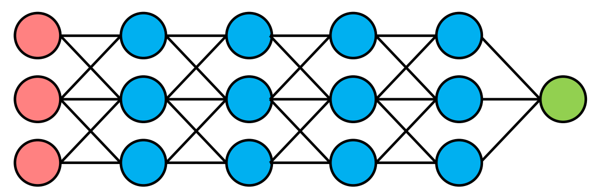


- hmmm…OK, but: multilayer neural networks have been around for 25 years. **What's actually new?**

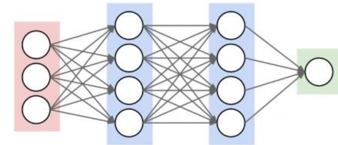
- We have always had **good algorithms** to **learn** the **weights** in **networks with 1 hidden layer…**



- …but **these algorithms** are **not good** at learning the weights for **networks with more hidden layers**

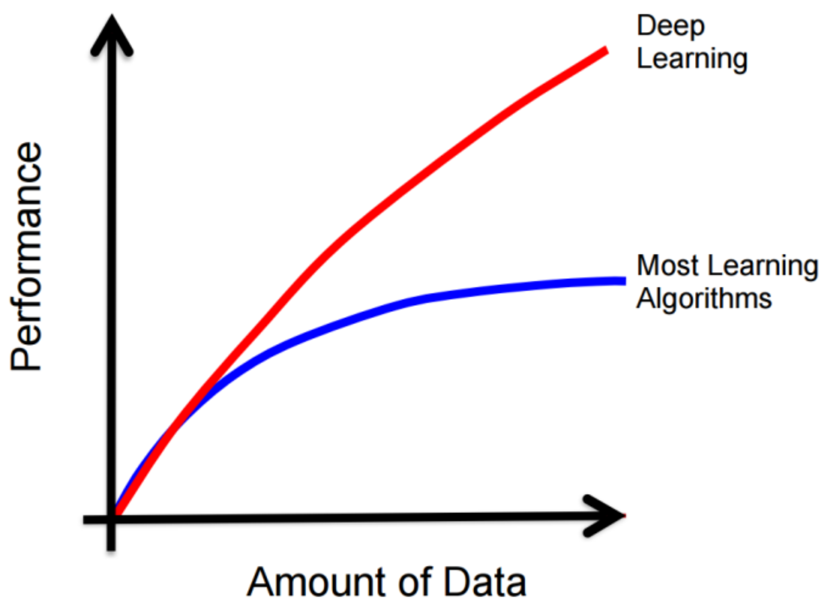


- **What's new is**: algorithms to **train many-later networks**



Deep Learning 101

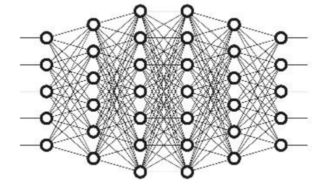
- Why Deep Learning: **Scalable** Machine Learning
- The **more** the training **data**, the **better** the **performance**
- **Why now: data, hardware**, community, tools, investment



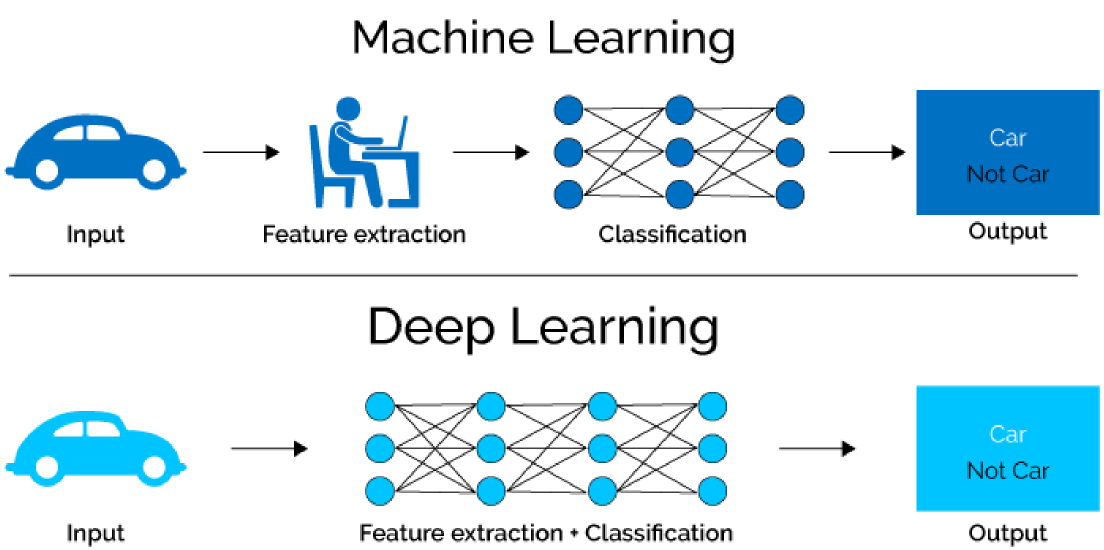
Exciting progress:

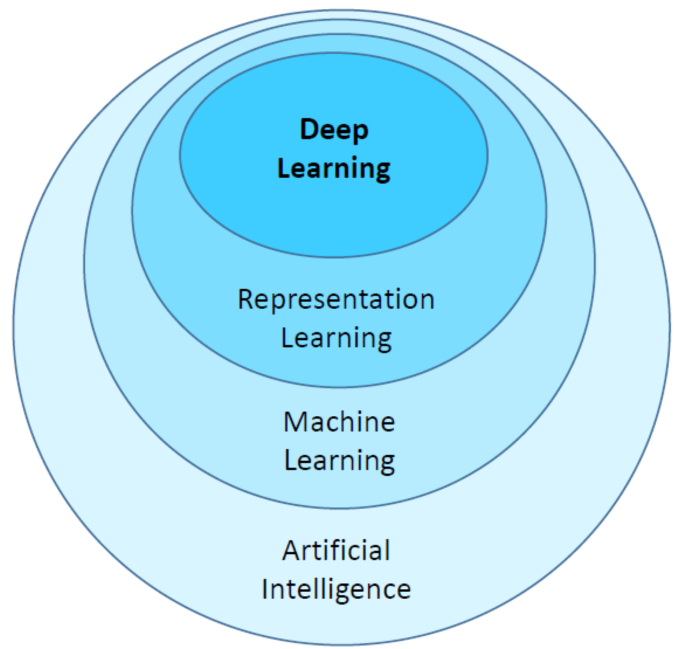
- Face recognition
- Image classification
- Speech recognition
- Text-to-speech generation
- Handwriting transcription
- Machine translation
- Medical diagnosis
- Cars: drivable area, lane keeping
- Digital assistants
- Ads, search, social recommendations
- Gaming

Deep Learning 101

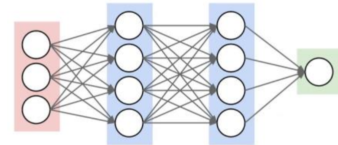


- One of the keys behind DL is the automatic **learning of data representations**
- DL algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**





Deep Learning – Why is it Useful?



- **Manually designed features** are often **over-specified**, **incomplete** and **take a long time** to design and validate.

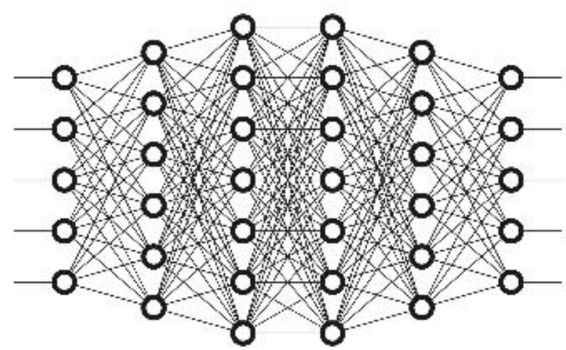
- **Learned Features** are **easy to adapt**, **fast to learn**.

- **Deep learning** provides a very flexible, (almost?) universal, learnable framework to **represent world, visual and linguistic information**.

- Can learn both **unsupervised** and **supervised.**

- Effective **end-to-end** joint system **learning**.

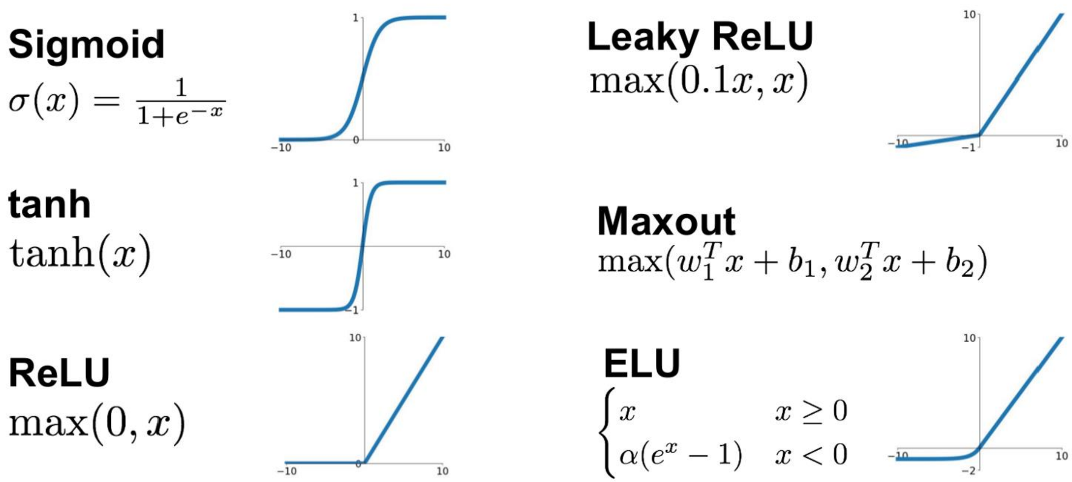
- **Use massive amounts of training data.**

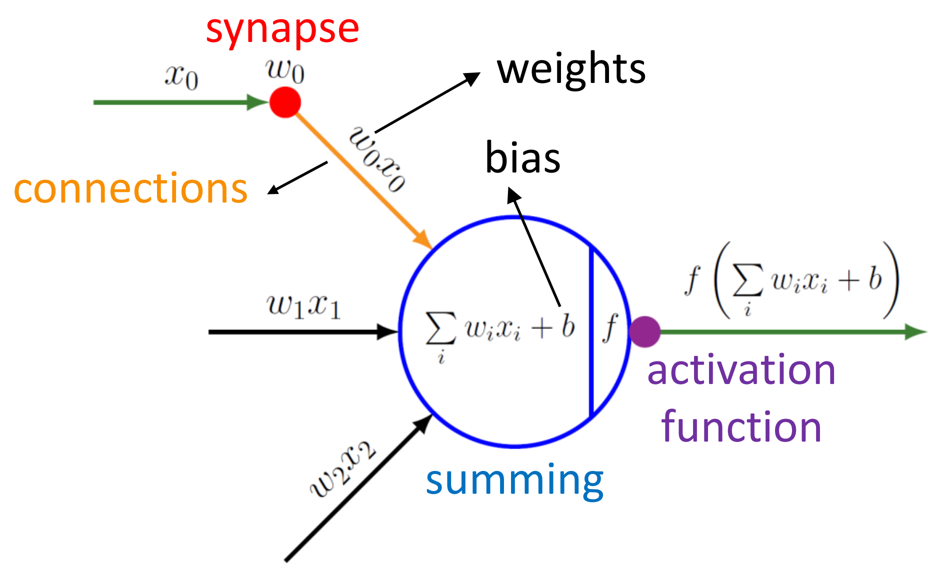


Neural Networks – Neuron Model

- Artificial neurons are the computational building blocks for **Artificial Neural Networks** (ANNs)

- *Inspired* by natural brain neurons…



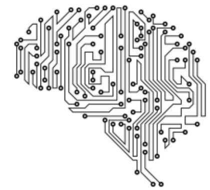


- …but **natural neurons** and the **human brain** have probably **nothing to do with ANNs**!

ANNs (DNNs) vs the Human Brain

Human Brain

- Humans can **learn from very few examples** (embedded past knowledge)
- 100 billion neurons, 1.000 trillion synapses (**DNNs x 10 M**)
- Human brain has **no layers**, brain **works asynchronously**
- **No clue how it learns,** certainly **NOT** through **backpropagation**
- **Life-long learning** (non-stop learning), unsupervised and through exploration
- **Energy-efficient** (very little power)

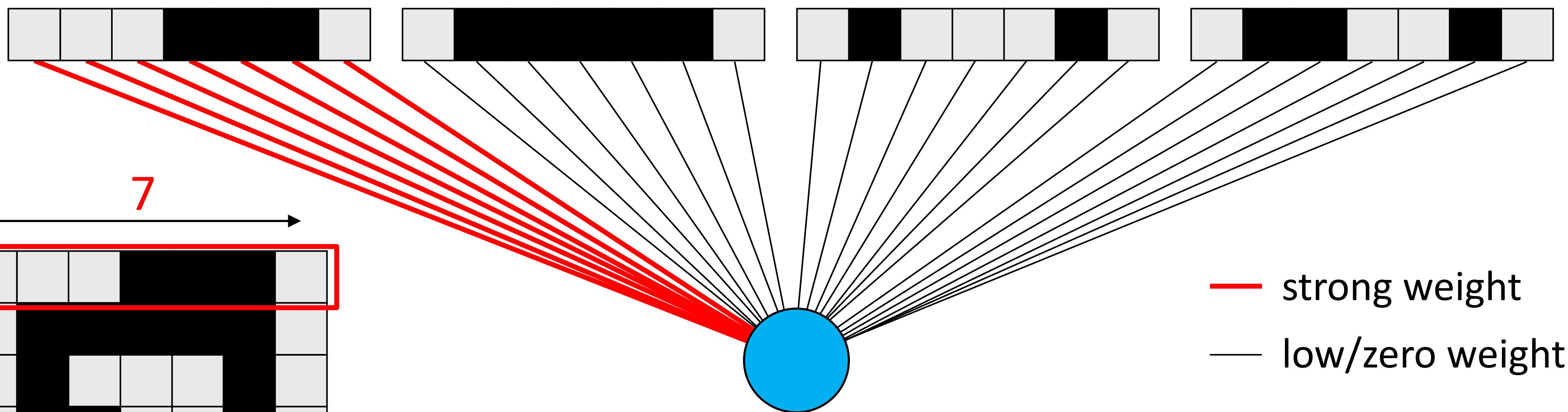


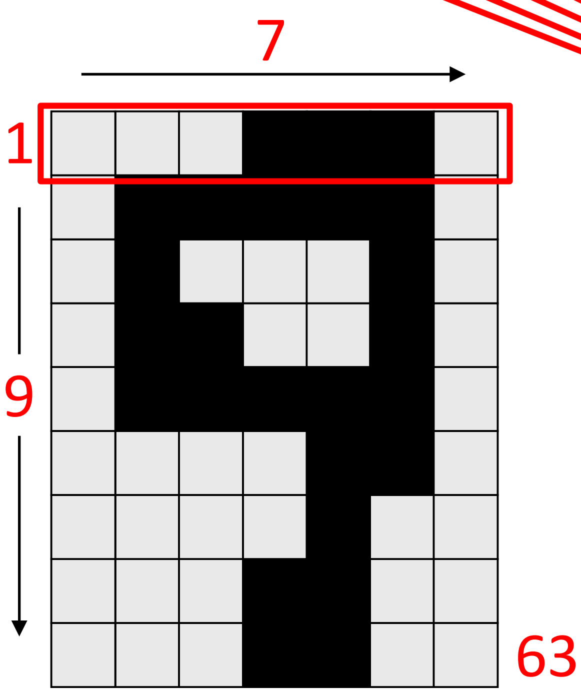
DNNs

- DNNs need **thousands/millions of examples**, even to **learn basic** mappings
- **ResNet 152**: **60 million connections** (weights)
- DNNs are **synchronous**
- Learning by gradient-descent (**backpropagation**)
- Mostly on **supervised learning**
- **Get ready to pay the energy bill!**

Deep Learning – Intuitive Example

- Automatic **learning** of **data representations – how?**
- **Input raw data**, connection **weights learn** to detect specific **feature maps**

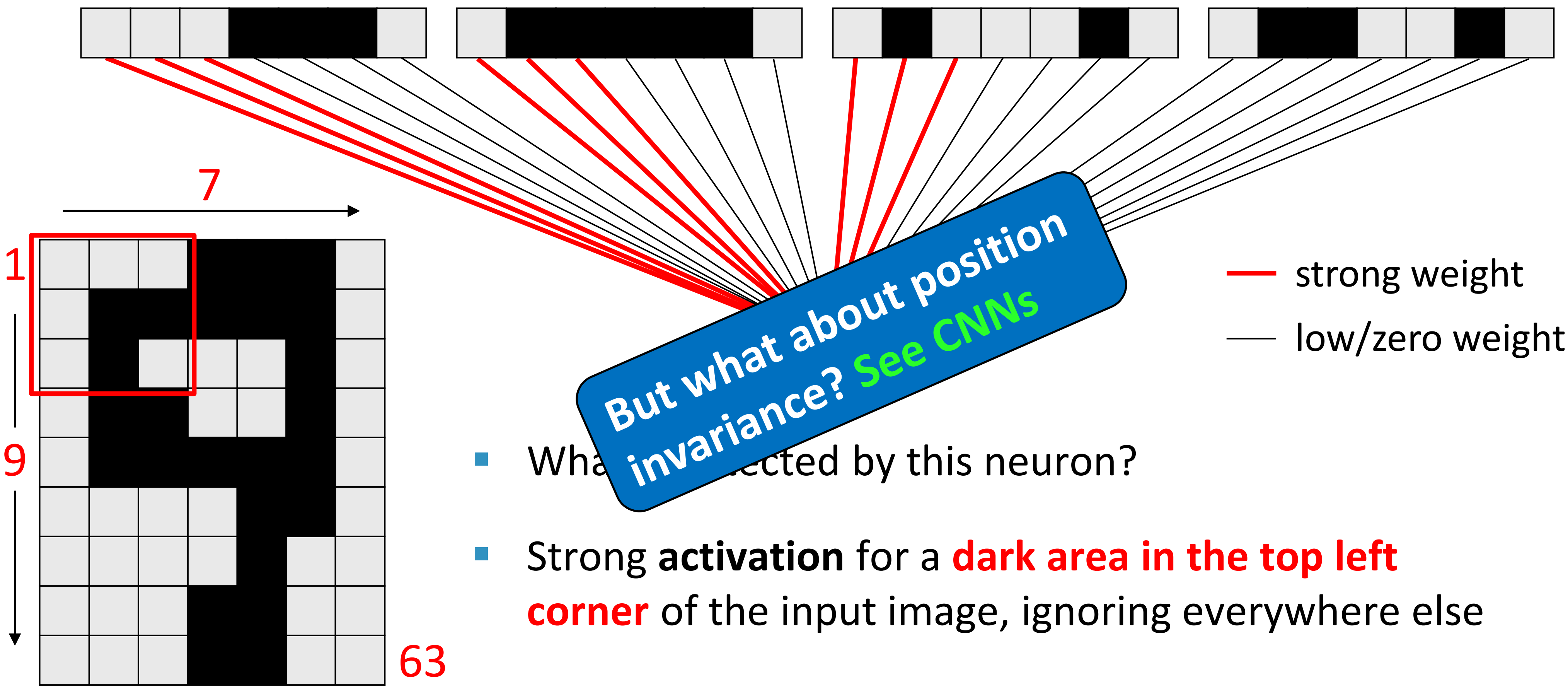




- What is detected by this neuron?
- Strong **activation** for a **horizontal line in the top row** of the input image, ignoring everywhere else

Deep Learning – Intuitive Example

- Automatic **learning** of **data representations – how?**
- **Input raw data**, connection **weights learn** to detect specific **feature maps**



- Wha… …ected by this neuron?
- Strong **activation** for a **dark area in the top left corner** of the input image, ignoring everywhere else

Deep Learning – Training

- **Training a NN** means **setting**/tuning **all the free-parameters** (weights and bias)

- This is achieved by solving an optimization problem, minimizing a certain **loss function**, which quantifies the **gap between prediction and ground truth**:

- **Regression**
  - Mean Squared Error (MSE)

Prediction

$$MSE = \frac{1}{N}\sum(t_i - s_i)^2$$

Ground Truth

- **Classification**
  - Cross Entropy Loss (CE)

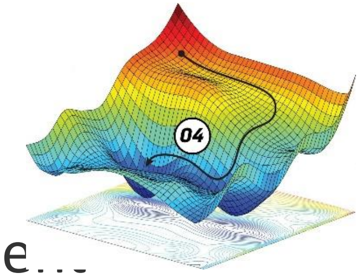
Classes

Prediction (p)

$$CE = -\sum_{i}^{C} t_i log(s_i)$$

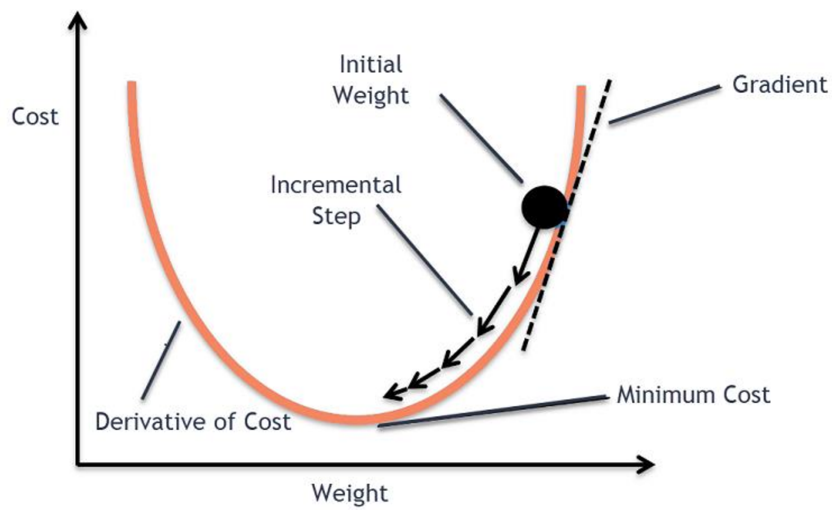
binary {0,1} correct class

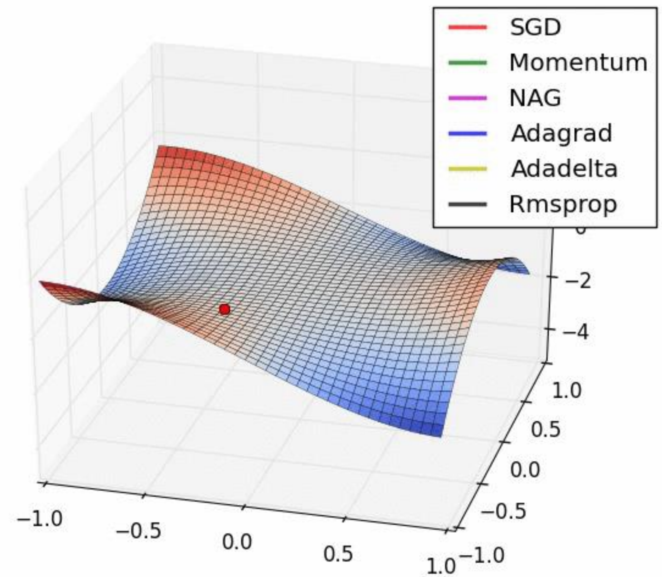
Optimization by Gradient Descent



- How to iteratively minimize the loss function?
  - e.g.: **Stochastic Gradient Descent (SGD),** Adaptive Mome… Estimation (Adam), RMSprop, etc.

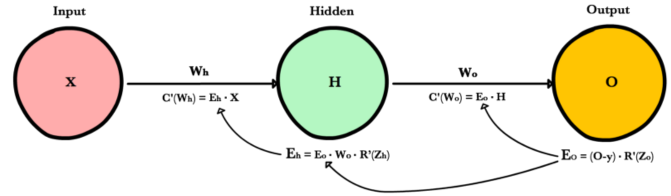
- Update the weights and bias by moving in the negative direction of the **loss function derivate**





- **Gradient** is computed for the loss function **w.r.t. weights/bias** ($\boldsymbol{\theta}$)
- e.g., for MSE → $J_n(\boldsymbol{\theta}) = || \boldsymbol{\theta}^T \boldsymbol{x}_n - \boldsymbol{y}_n ||^2$ → $\nabla_\theta J_n(\boldsymbol{\theta}) = \boldsymbol{\theta}^T \left(\boldsymbol{\theta}\ \boldsymbol{x}_n - \boldsymbol{y}_n\right) = \boldsymbol{\theta}^T \boldsymbol{\theta} \boldsymbol{x}_n - \boldsymbol{\theta}^T \boldsymbol{y}_n$

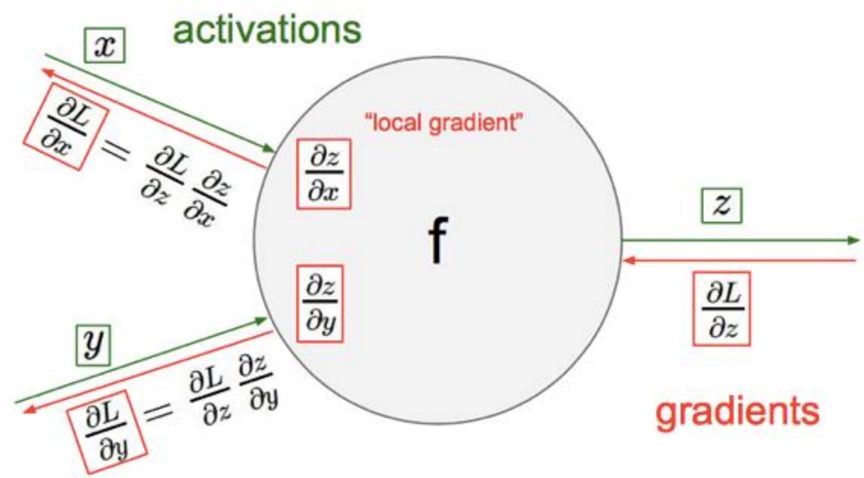
Backpropagation



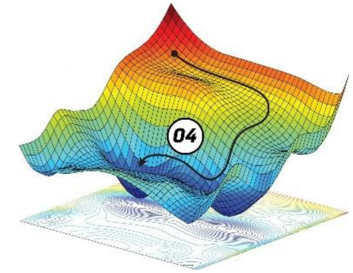
- Update each element of θ → $\theta_j^{new} = \theta_j^{old} - \alpha \frac{d}{d\theta_j^{old}} J(\theta)$
- Matrix notation for all parameters → $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

  learning rate

- Computing the **analytical expression** for the gradient is **straightforward**
- …but **numerically evaluating** the gradient is **computationally expensive**
- Solution: **backpropagation**
- Use the **chain rule** to sequentially compute the gradient through each node, re-using previous computations



Batch Gradient Descent



- **How often** and based on **which data** do we **update weights?**
  - **Epoch:** represents one iteration over the entire training data (**size *m***)
  - **Batch:** if data is too big, we split it in **batches**
  - **Iteration:** an epoch is composed of **data-size/batch-size** iterations

(1) **Batch gradient descent:** take **all the training data** to take one gradient decent step. This is very slow if you have large data set.

(2) **Online-training /stochastic gradient descent:** each training example (or few of them) is a batch in itself. Weights are updated for each training example.

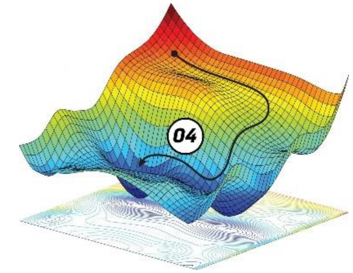
(3) **Mini-batch gradient descent:** split the available data in batches of fixed size. Each gradient descent step takes batch-size of data samples to take one gradient decent step. Faster than batch gradient decent.

batch size

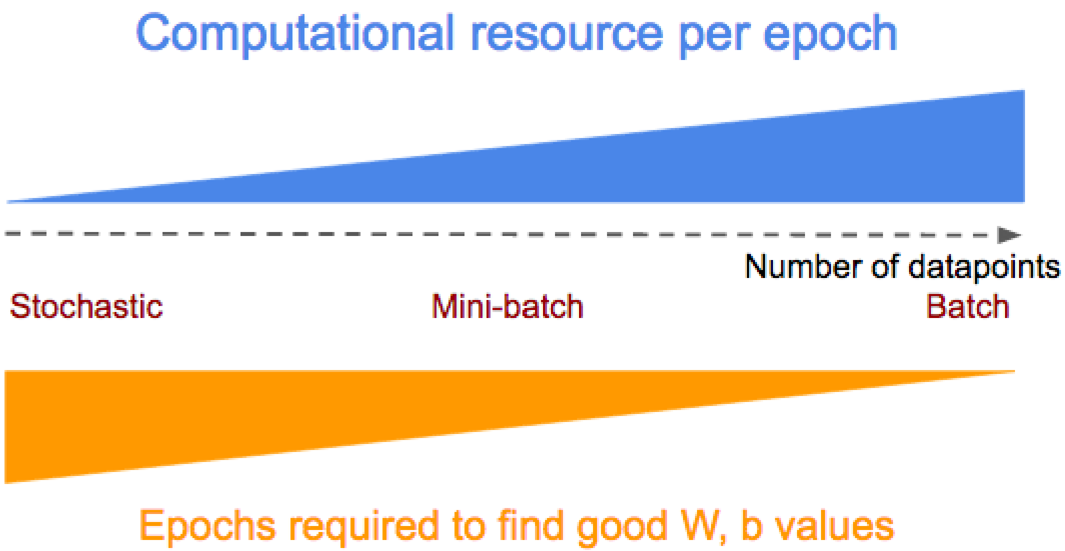
(2) (3) (1)

1 ⟷ *m*

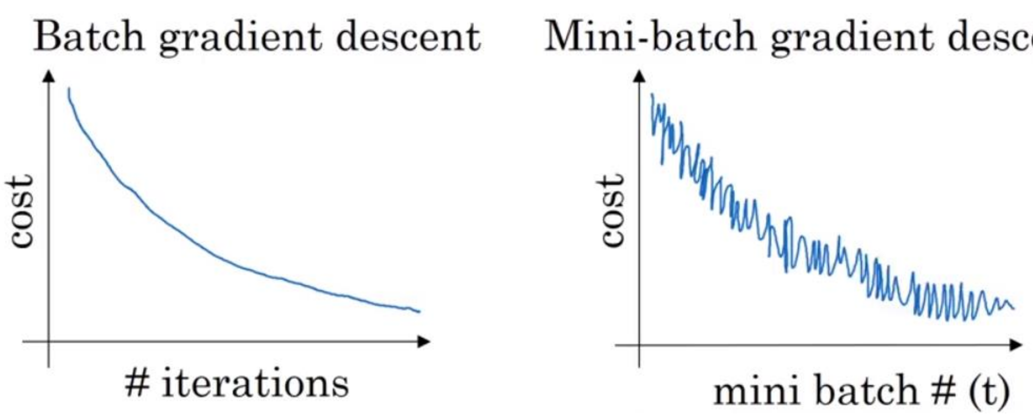
Batch Gradient Descent – Tradeoffs

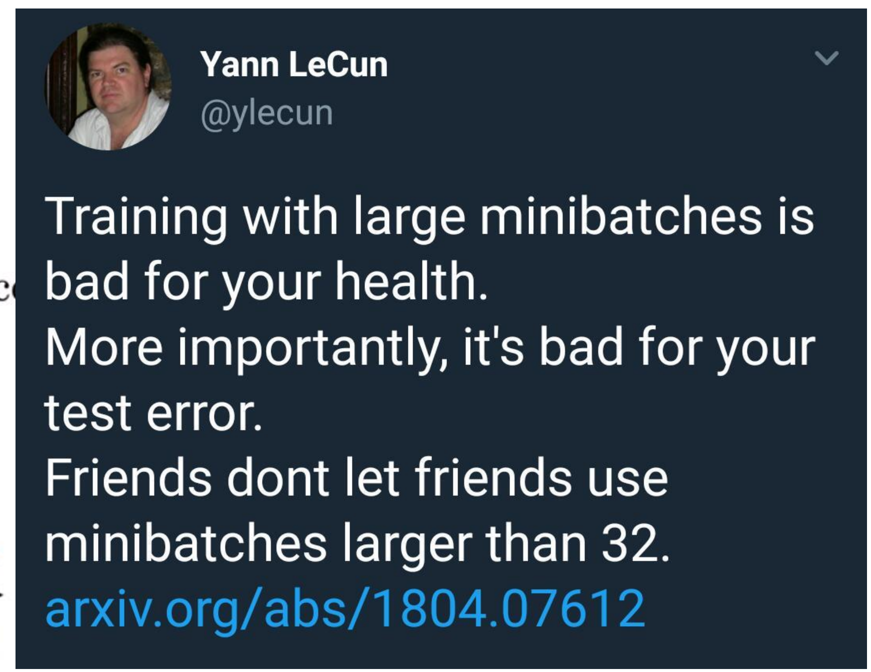


- What's better, **smaller** or **larger** **batch size**?

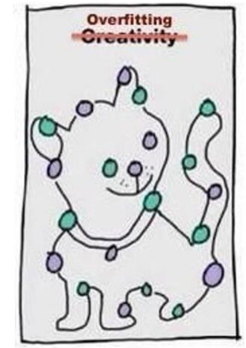


- Larger batch size = needs more computational resources
- Smaller batch size = (empirically) better generalization





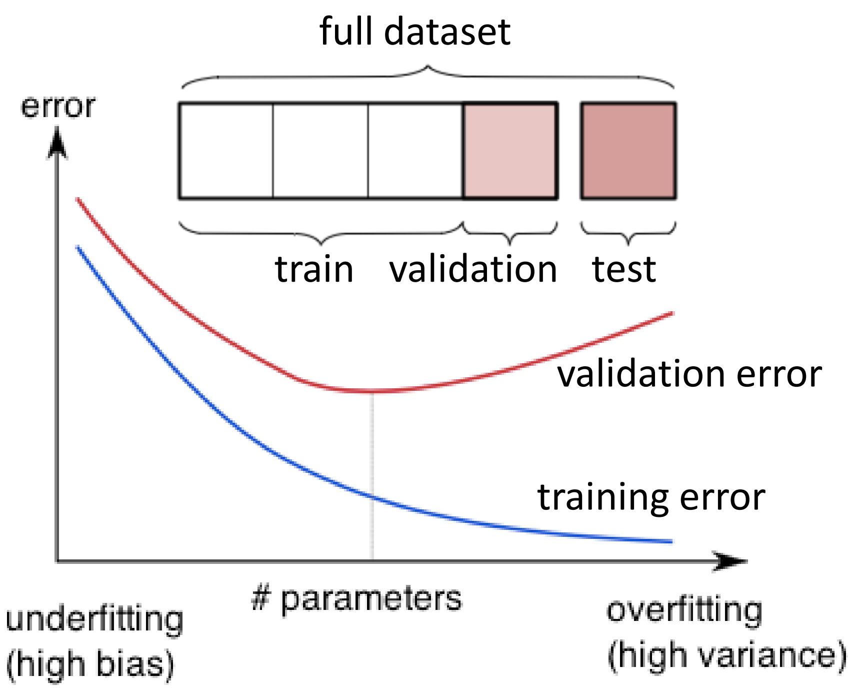
Regularization – fighting overfitting



Most important part of learning: generalize to unseen data

(1) **Early stopping:** stop the training when the algorithm stops learning the underlying model

(2) **Dropout:** randomly drop units (along with their connections) during training. **At each iteration**, each unit is retained with fixed **probability *p*** (usually $p >$ 0.5), independent of other units



(3) **Weight penalty/decay** (e.g., L2): prevent big weights. Results in smoother models. e.g., (*w/2*; w/2) is better than (*w*; *0*)

(4) **L1 weight decay:** allows for a few weights to remain large

Data Normalization

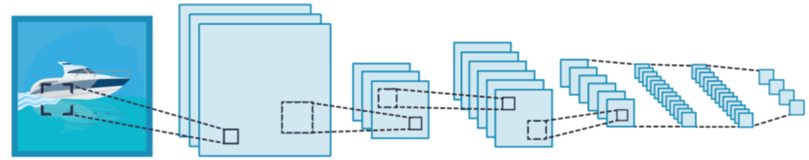
- Data normalization helps to **speed up the learning process**, by keeping activations from going too high/too low.

- **Input normalization:** normalize network inputs, e.g.: normalize to [0,1], or according to mean & var., etc.

- **Batch normalization (BN):** normalize hidden layer inputs to mini-batch mean & var. During training, the distribution of each layer's inputs changes as the parameters of the previous layers change. BN reduces impact of earlier layers on later layers.

- Many other alternatives:
  - Layer normalization (LN) – conceived for RNNs
  - Instance normalization (IN) – conceived for Style Transfer
  - Group normalization (GN) – conceived for CNNs

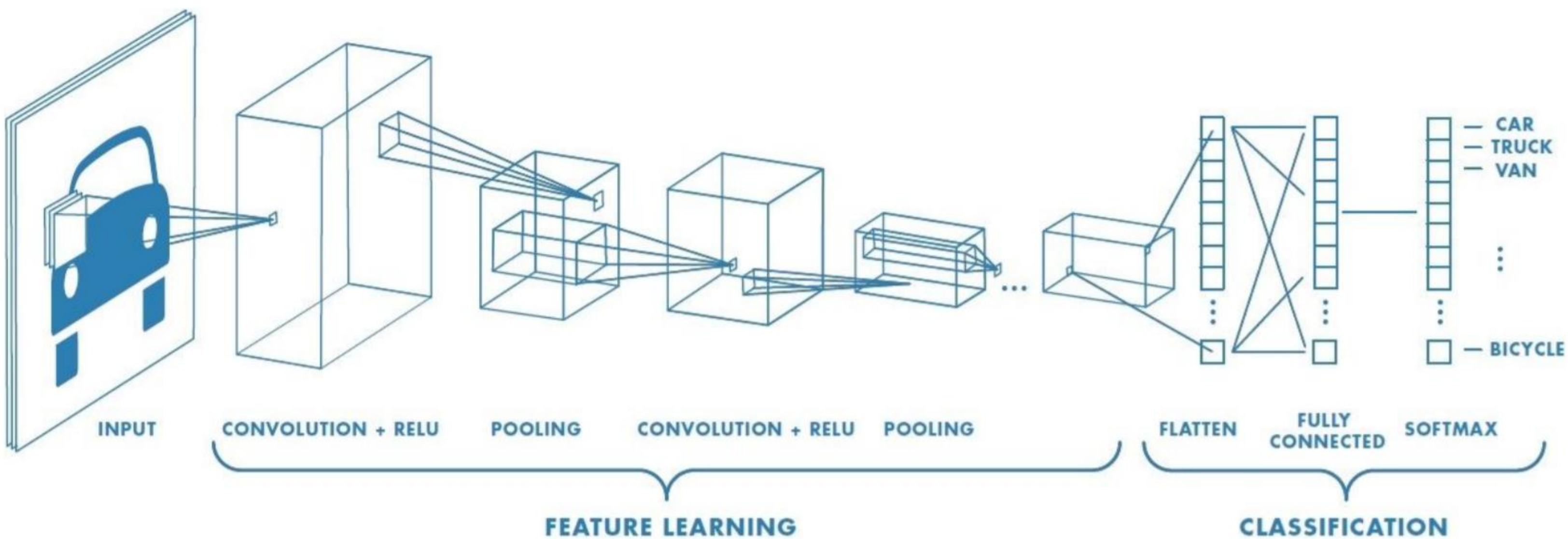
Convolutional Networks (CNN)



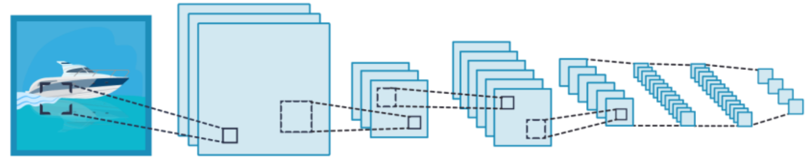
- **Convolutional Neural Networks**: build spatial features, reducing the number of parameters needed for image processing

- **CNNs** are specially conceived for **image processing tasks**, their success is the **primary reason why deep learning is so popular**

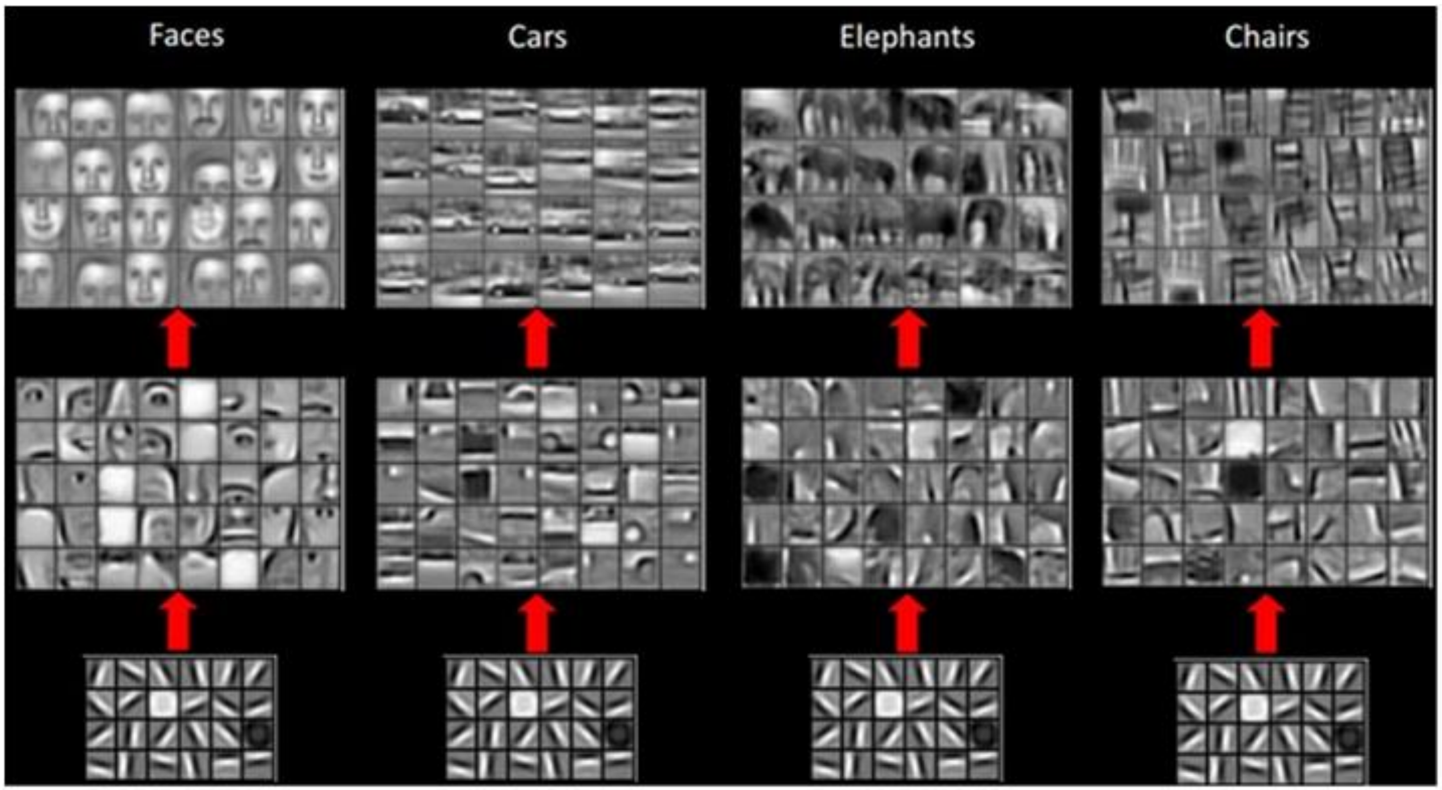
- Convolutional neural networks are composed by **a set of layers with specific functionality**



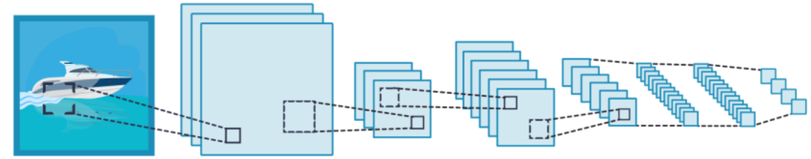
Convolutional Networks (CNN)



- CNNs detect features in images and learn how to recognize objects with them
- **Layers near the start** detect **simple features** like edges
- **Deeper layers** can detect more **complex features** like eyes, noses, or an entire face



Convolutional Layer



- **Key idea:** we change weights by ***feature detectors*** or filters, and drastically reduce connections
- **Learning in CNNs** is about calibrating the feature detector values
- In a nutshell: we learn **new filters**, which **discover specific characteristics** of the image
- Convolutional layers work as **feature detectors**, generating the so-called **activation maps**

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

Input Image

$\otimes$

0	0	1
1	0	0
0	1	1

Feature Detector

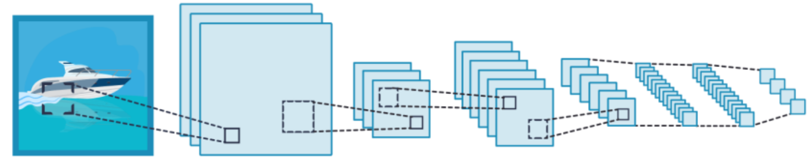
1 ×1	1 ×0	1 ×1	0	0
0 ×0	1 ×1	1 ×0	1	0
0 ×1	0 ×0	1 ×1	1	1
0	0	1	1	0
0	1	1	0	0

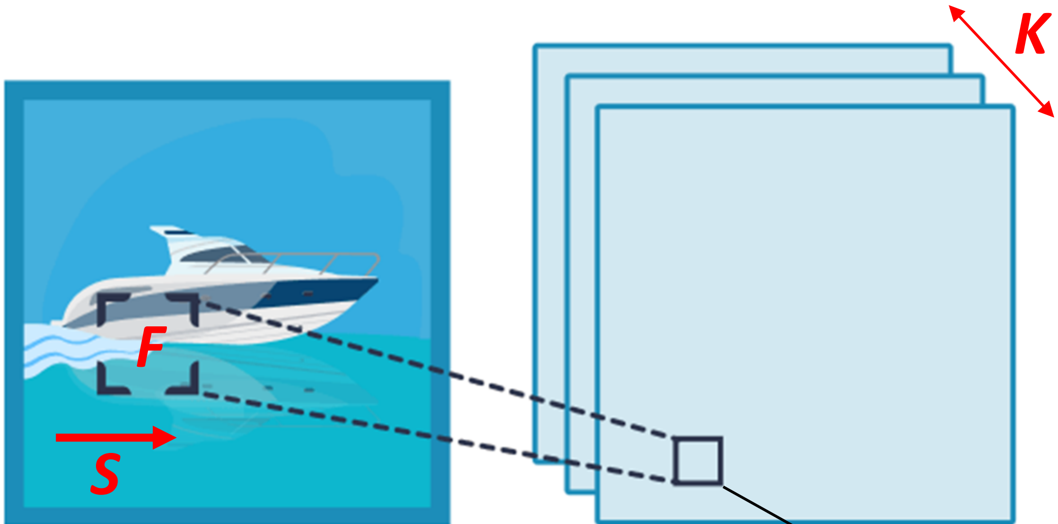
Image

4		

Convolved Feature

Convolutional Layer + ReLU

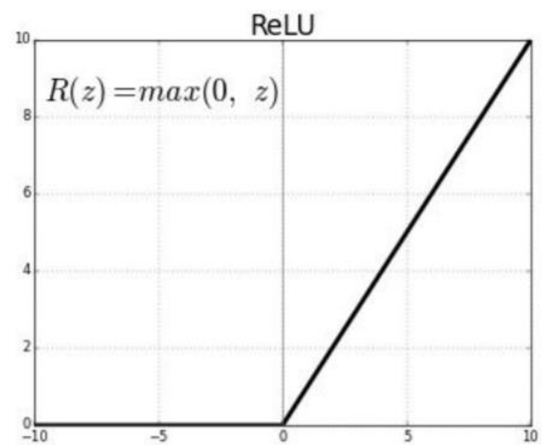




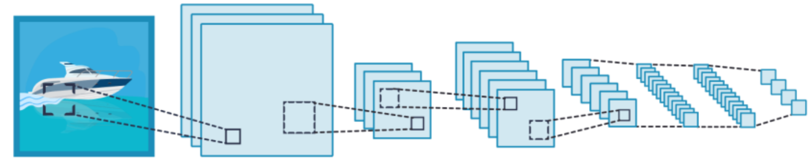
- **Hyper-parameters :**
  - the number of filters ***K***
  - the size of the filters ***F***
  - the stride ***S***

- The convolutional step is combined with an **activation layer**, usually ReLU – Rectifier Linear Unit

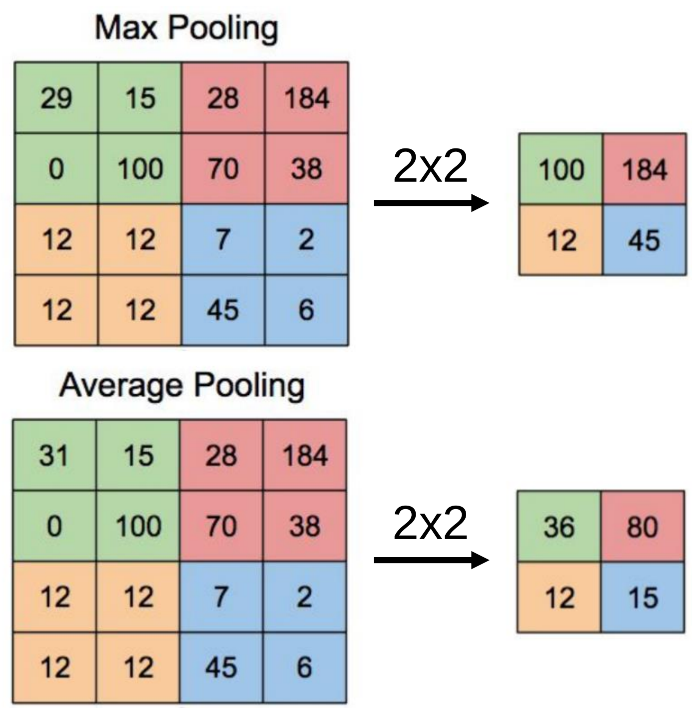
- Used to **increase non-linearity** of the network without affecting receptive fields of convolutional layers
- Prefer ReLU, results in faster training
- **LeakyReLU** addresses the vanishing gradient problem



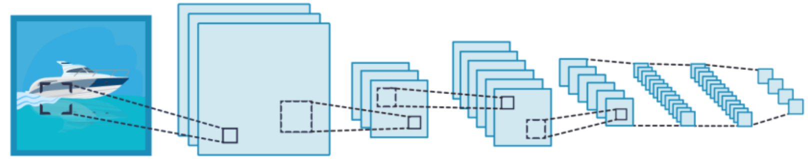
Pooling Layer



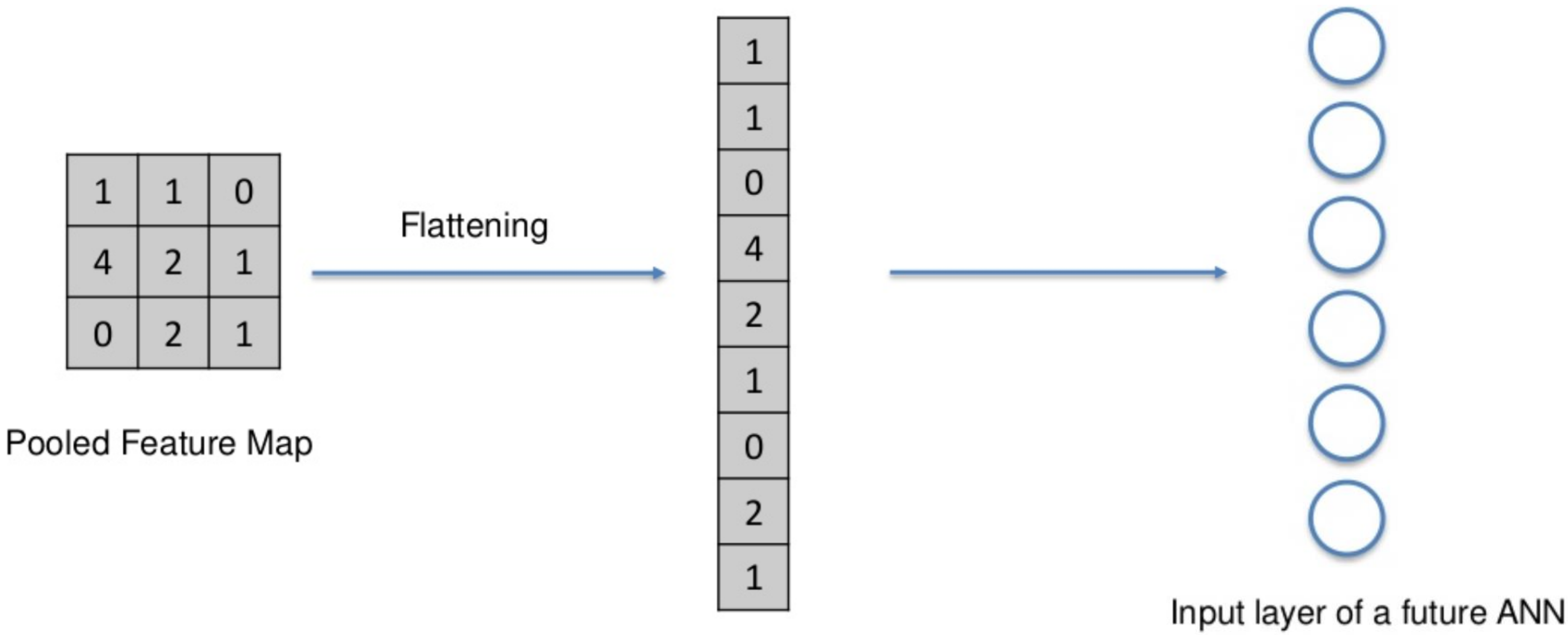
- The main goal of the pooling function is to **progressively reduce the spatial size of the representation** to **reduce the amount of parameters** and computation in the network
- Hence, it **also controls overfitting**
- A **pooling function** replaces the output of the network at a certain location with a **summary statistic of the nearby outputs**
- **Pooling layers** apply **non-linear down-sampling** on activation maps
- Pooling is very aggressive (discard info)
- The **trend** now is to **use smaller filter size** and abandon pooling



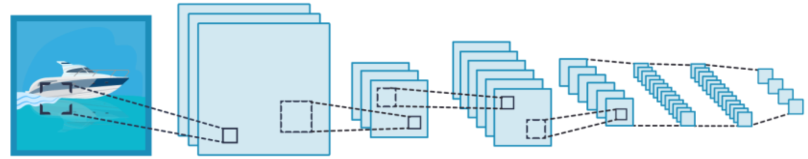
Flattening Layer



- Flattening is **converting the data into a 1-dimensional array**
- We flatten the **output of the pooling layers** to create a single long feature vector
- And it is connected to the final classification model, which is called a **fully-connected layer**



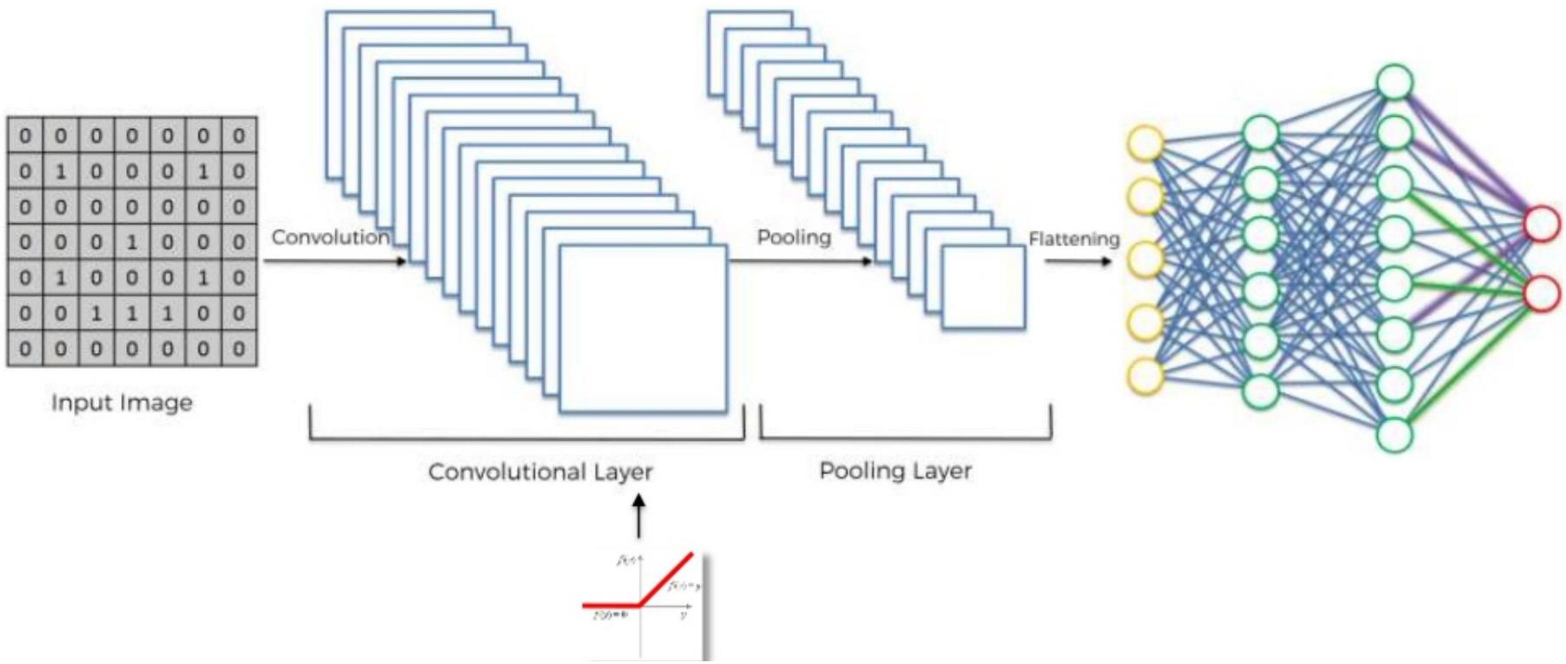
Fully-Connected Layer



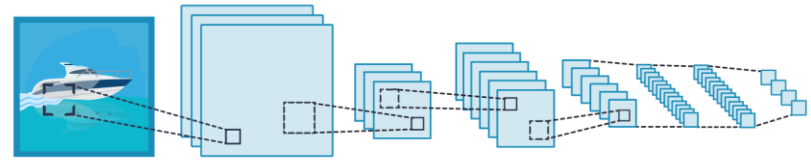
- Fully connected layer = **Regular neural network**

- It corresponds to the **final learning phase**, which maps extracted visual features to desired outputs (e.g., classification)

- Common output is a vector, which is then **passed through a *softmax* function** to represent **confidence of classification**



Softmax Layer

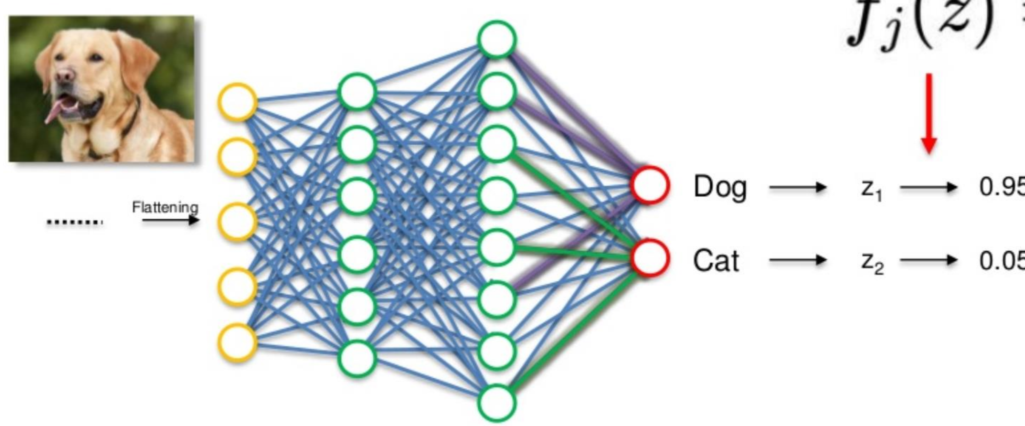


- A special kind of **activation layer**, usually **used at the end of FC layer outputs**
- Can be viewed as a normalizer, producing a **discrete probability distribution** vector
- The Softmax is used as the **activation function in the output layer of the FC Layer**, and ensures that the sum of the outputs is 1.
- The Softmax function takes a **vector of arbitrary real-valued scores** and **squashes it to a vector of values between zero and one that sum to one**

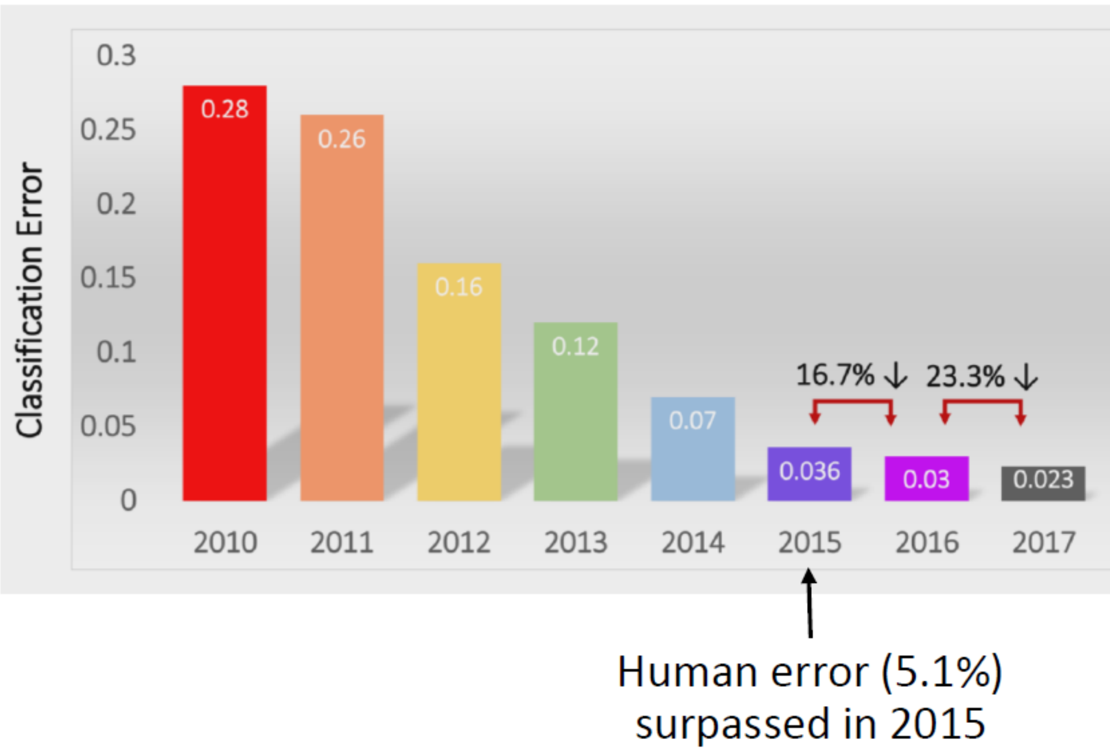
$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Given sample vector input **x** and weight vectors {$\mathbf{w}_j$}, the predicted probability of y = j

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



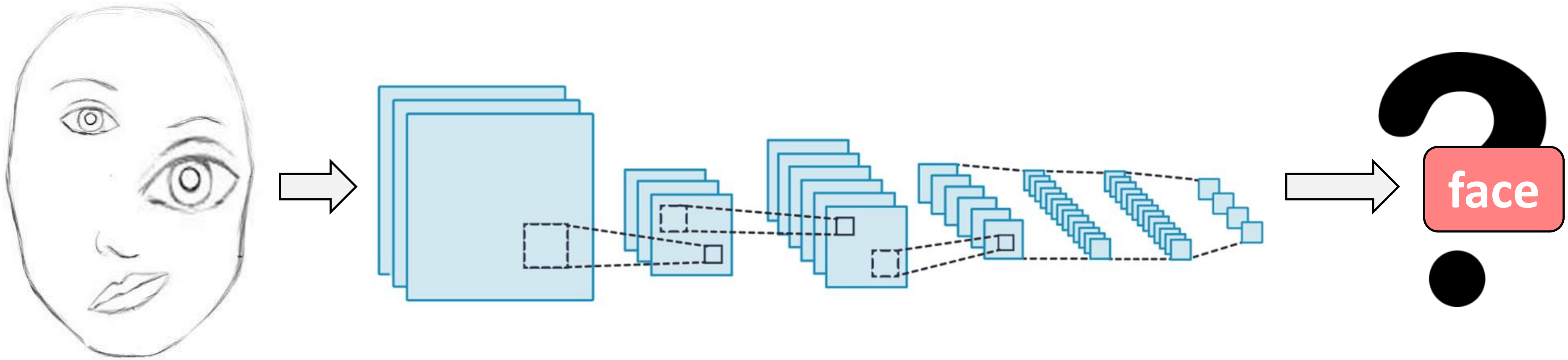
ImageNet Large Scale Visual Recognition Challenge

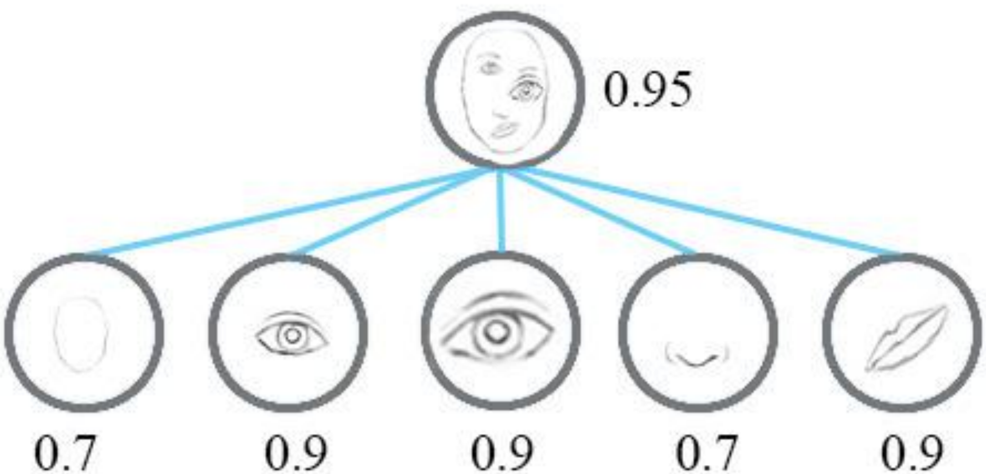


- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models
- **SENet (2017): 2.99% to 2.251%**
  - Squeeze and excitation block: network is allowed to adaptively adjust the weighting of each feature map in the convolutional block.

CNN Quiz

- **What would this CNN detect?**





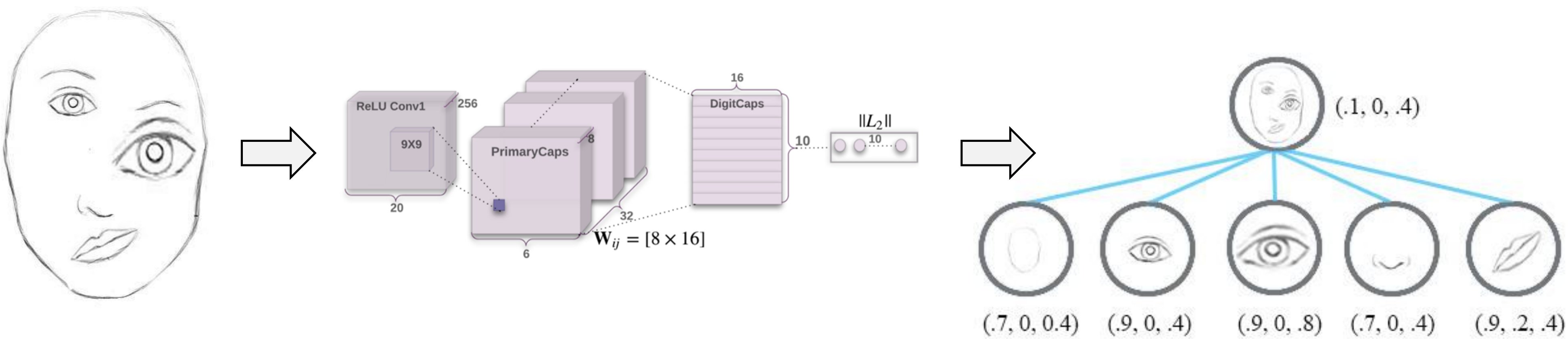
- CNNs have drawbacks in their basic architecture…
- …causing them to not work very well for some tasks
- **there is no spatial location information in a CNN!**

Capsule Networks (CapsNets)

- **In a nutshell:** why not adding such information within the network?

- **Capsule networks**: encode not only probability of an object being present, but also spatial information

- **Capsules**: groups of neurons that encode spatial information (object position and orientation) as well as the probability of an object being present.



- **Capsule nets** are still in a **research and development phase** and not reliable enough to be used in commercial tasks

AUSTRIAN INSTITUTE OF TECHNOLOGY
TOMORROW TODAY

Thanks

Dr. Pedro Casas

Data Science @Digital Insight Lab

AIT Austrian Institute of Technology @Vienna

pedro.casas@ait.ac.at

http://pcasas.info