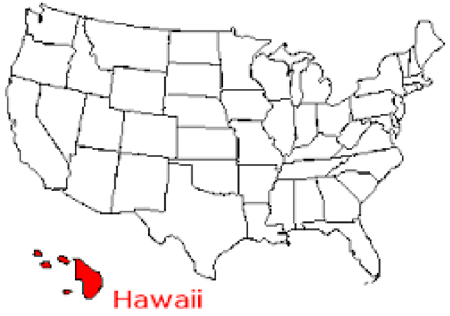
A general introduction to machine-learning and deep learning methods

Adeline Paiement, LIS lab, University of Toulon

Introducing myself…



2016-2018 : Research lecturer

- Department of Computer Science, Visual Computing research group
- IA for (physical) sciences

2013-2016 : Postdoc

- Machine learning
- Internet-of-things

2009-2013 : PhD

- Computer vision
- Medical image analysis

2008-2009 : Developer

- Robotics
- Computer vision
- Mars exploration rover

2008 : Engineering degree + astrophysics MSc

2018 - : Maître de conférences

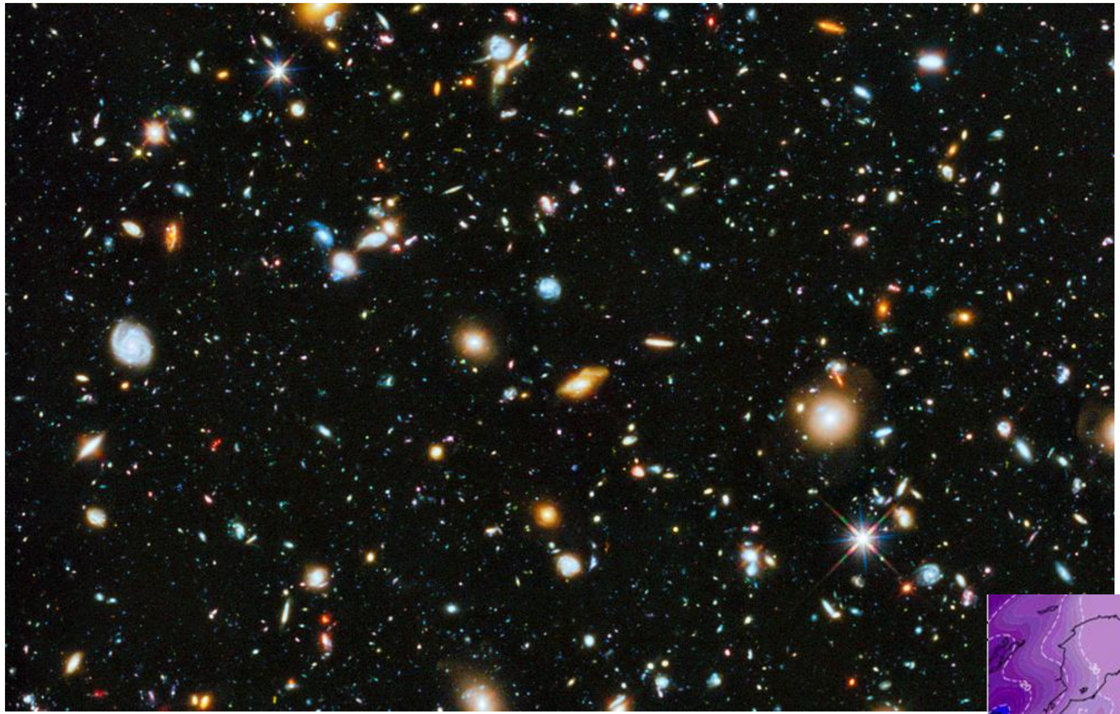
- Laboratoire d'Informatique et des Systèmes (LIS), pôle Science des Données
- AI for (physical) sciences

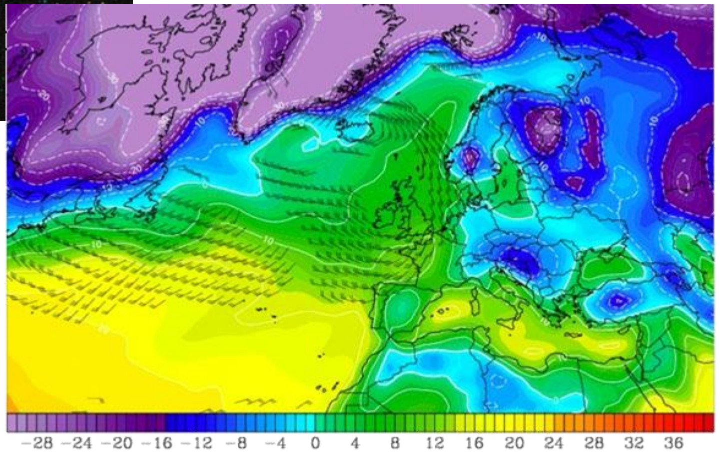
Outline:

- Introduction to machine learning
  - Unsupervised learning
  - Supervised learning
  - Example application to classifying images
  - Training in practice

- Neural networks and deep learning
  - Introduction to neural networks and deep learning
  - Application to images

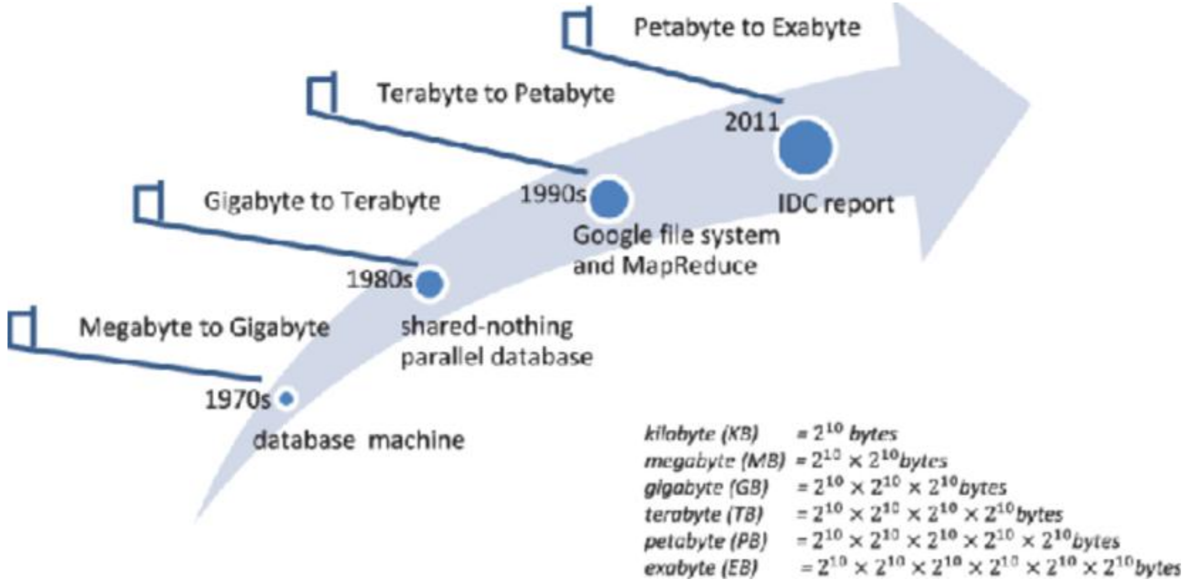
A deluge of data





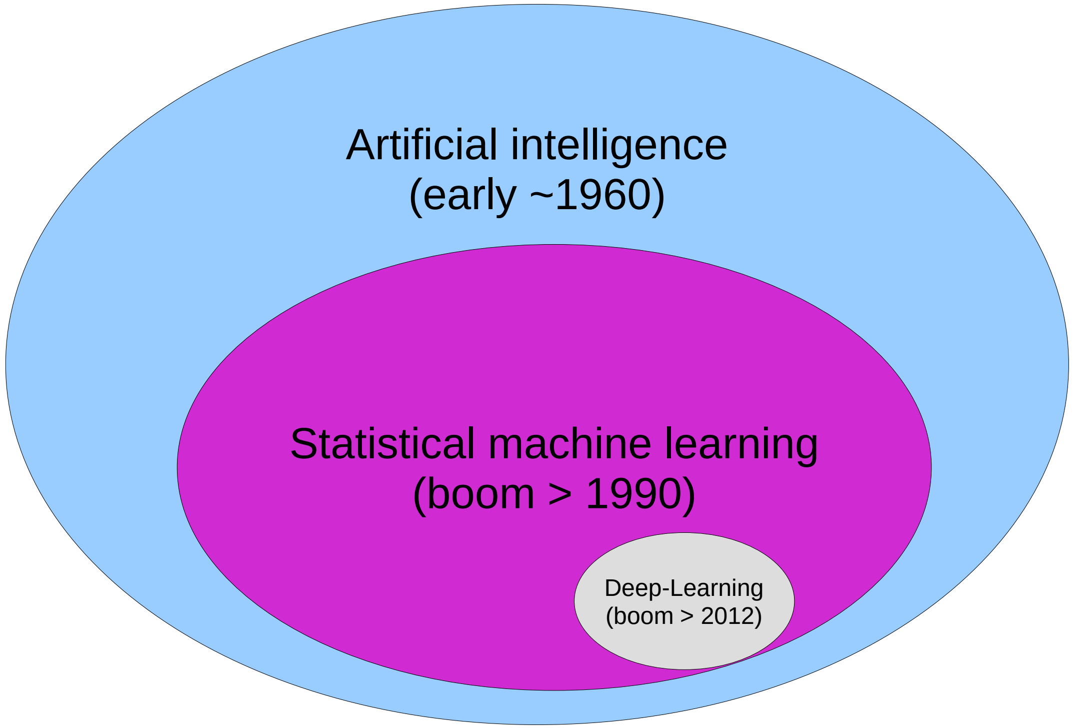
A deluge of data

Fact: The amount of data we produce increases exponentially



Data-oriented statistical model: perceived as a solution when:

- Weak or non-existing prior knowledge to build a mathematical/physics model that is robust to all possible noise and transformations
- Plenty of data available
- Complex relationship between raw input data and the information to be extracted



Statistical machine learning algorithms have increased in popularity since the 90s.

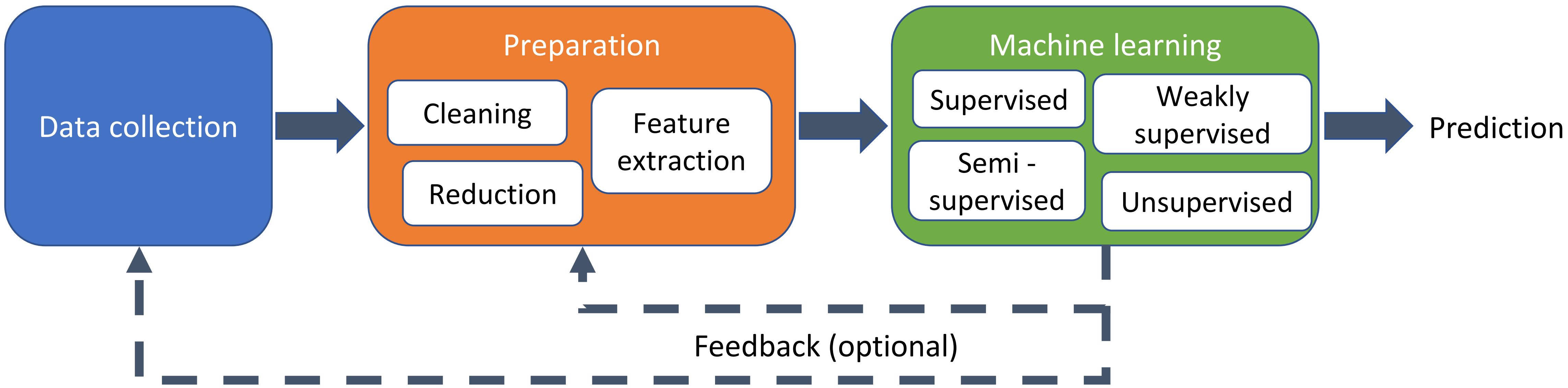
Objective: Using a large amount of data to mine/synthetize/index the information contained in the data automatically

→ Decision helping

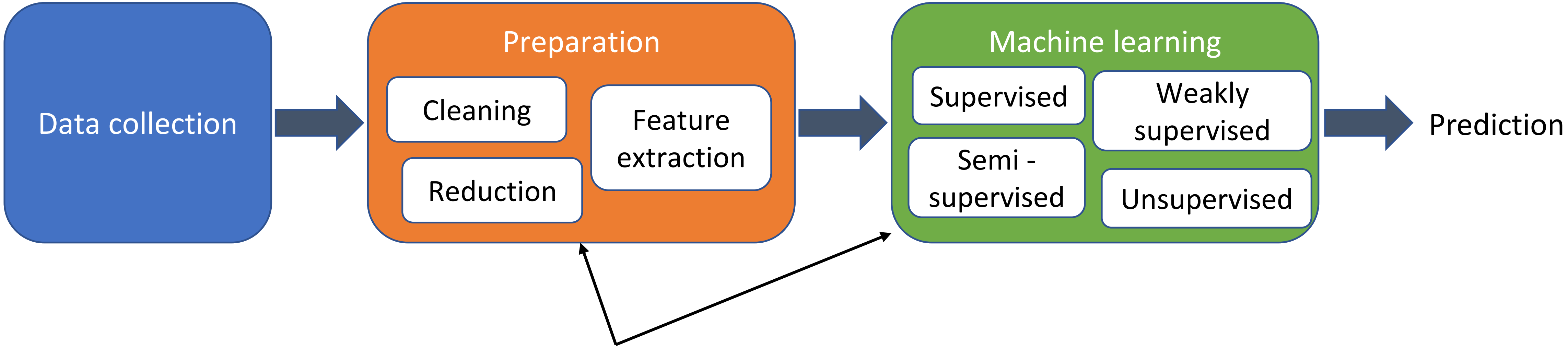
Data challenges

- Nature:
  - images
  - sound
  - text
  - graph
  - sequences
  - …
- Source:
  - telescopes
  - spectrographs
  - LASER
  - SONAR
  - accelerometers/gyroscopes
  - Highly integrated complex information systems (smart-cities, flux web, XML, etc...)
  - …
- Large dimensionality → Curse of dimensionality
- Multi-modal: multiple sensors with various properties, not necessarily pre-aligned nor synchronised
- Defects:
  - Measurement noise, outliers
  - Missing values
  - …

Generic data processing pipeline



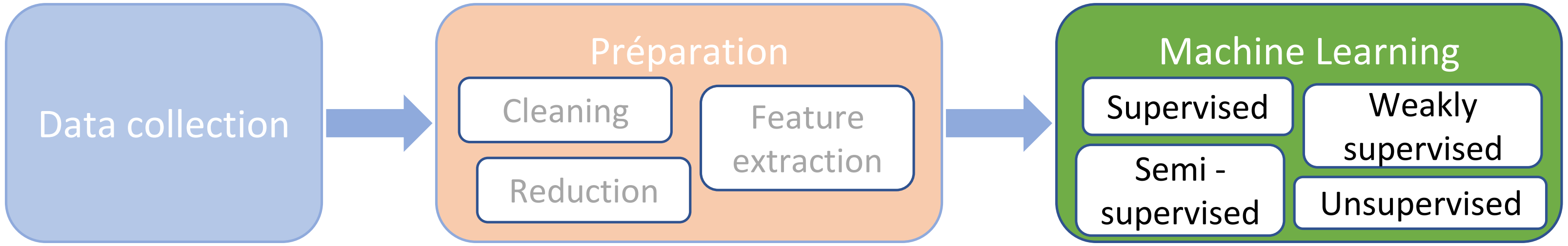
Generic data processing pipeline



Methods and steps are chosen based on:

- The data
- The problem

Four main types of problems



- ❑ **Association pattern mining**

  Finding patterns in the data

- ❑ **Clustering**

  Identifying groups with similar properties

- ❑ **Classification**

  Associating objects with predefined classes

- ❑ **Outlier detection**

  Finding objects/patterns which do not match the rest of the dataset

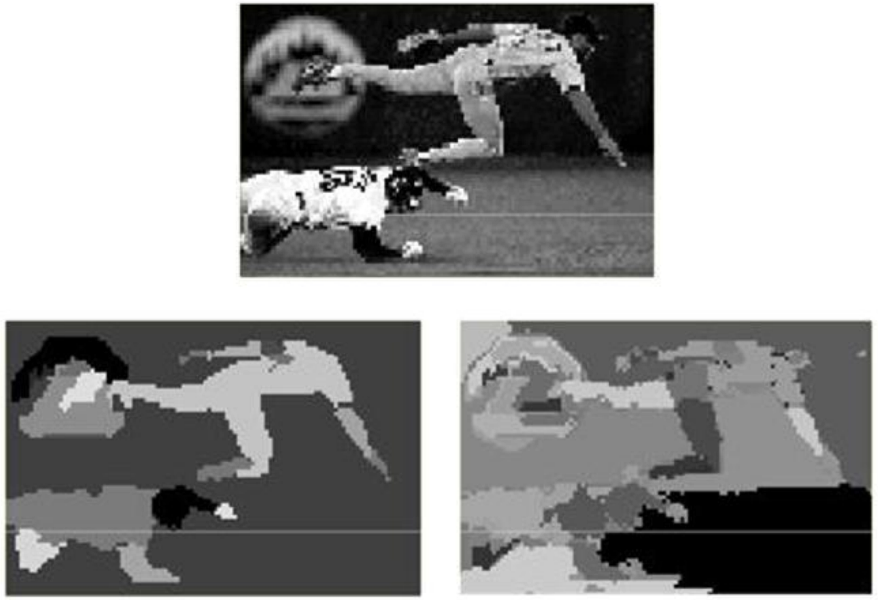
Note: there are usually multiple ways to tackle a same problem

Four main types of problems

- **Clustering**

  Identifying groups with similar properties

Clustering (pixel) values



Clustering observations (images, spectra…)

Four main types of problems

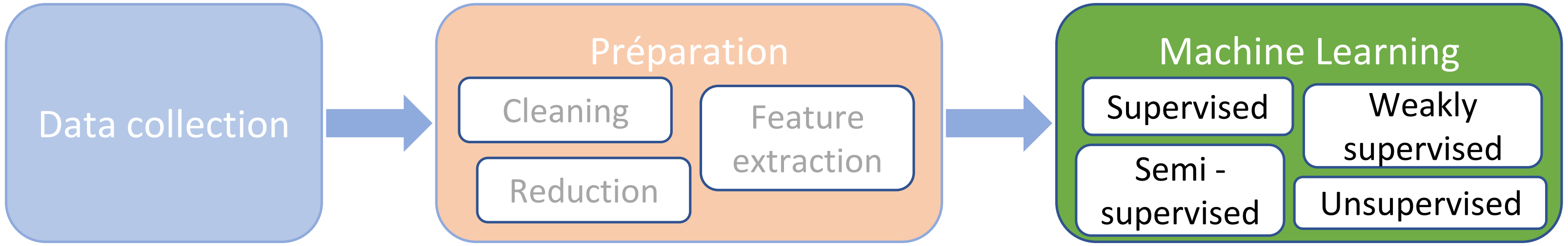
- ❑ **Classification**

  Associating objects with predefined classes

"Which kind of bird is it?"

Which pixels belong to the dog? → **Segmentation**

Main machine learning tasks



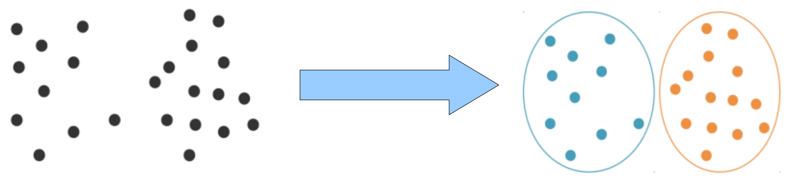
- ❑ **Unsupervised learning**
  Only input data is available. No labels.
- ❑ **Supervised learning**
  Each piece of data comes with an annotation → typically costly
- ❑ **Semi-supervised learning**
  Only part of the dataset is annotated
- ❑ **Weakly Supervised learning**
  For each piece of data, a (simple) annotation is available, but it only contains part of the relevant information

Unsupervised learning

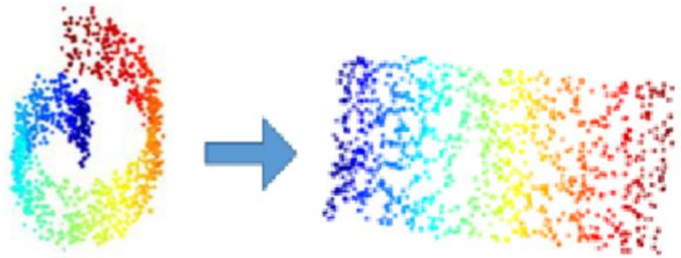
More about this in the next lecture

Main tasks consist in:

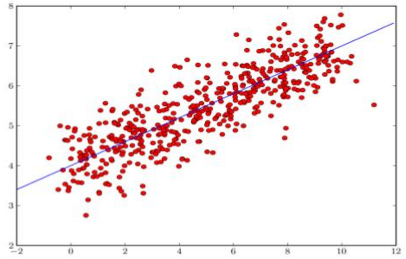
- Finding groups (clustering)



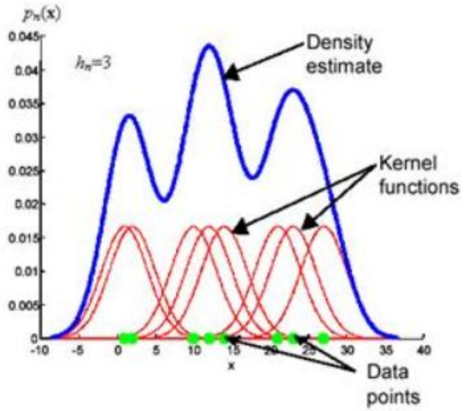
- Finding a lower-dimensionality representation of the data



- Finding interesting trends in the data



- Approximating density functions



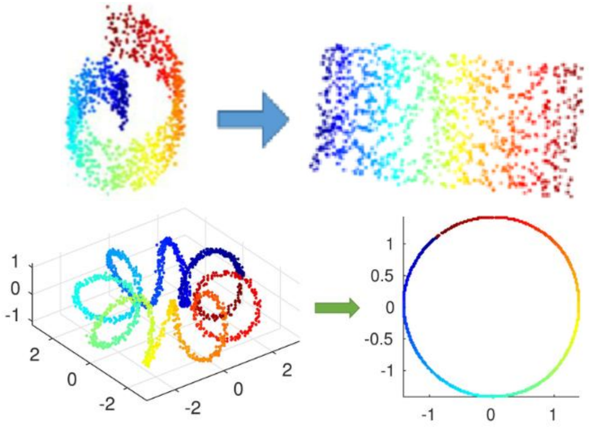
Dimensionality reduction / Manifold learning

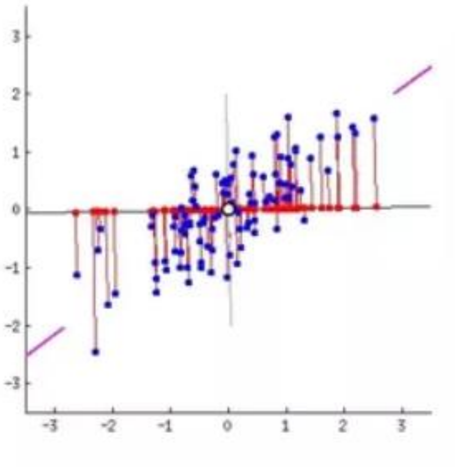
Aims:

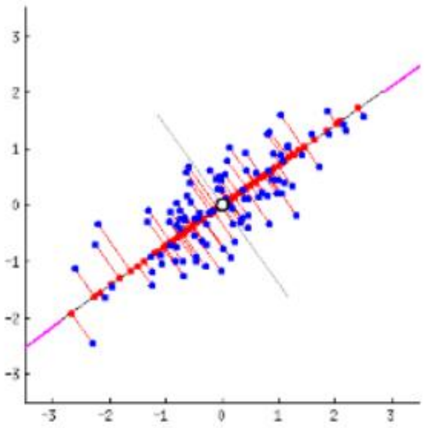
- Project the data in a different space where it is better structured

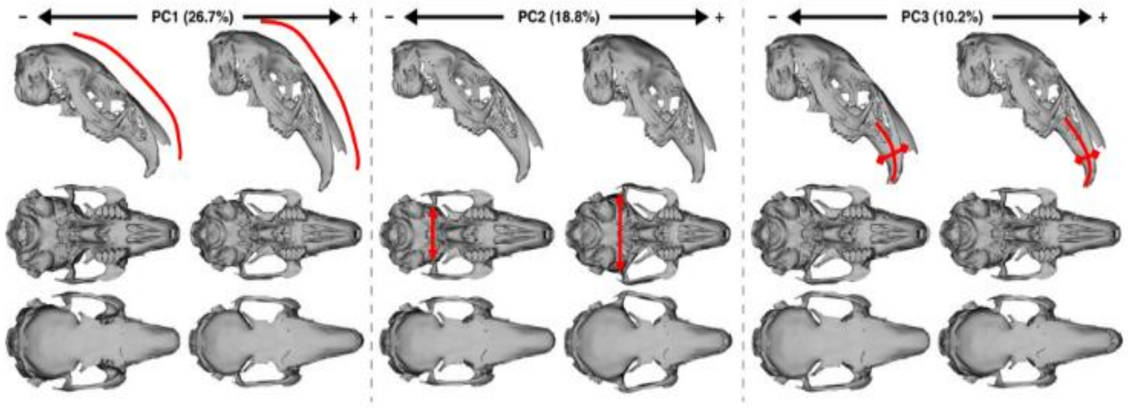
- Reduce dimensionality

- Identify main modes of variation









Supervised learning

IMAGENET

Supervised learning

- Feature selection
- Feature extraction
- Dimensionality reduction

may be first steps of data preparation for a further analysis with supervised learning.

Supervised learning is based on:

- Data

<u>and</u>

- Annotations / labels → costly

Assumption: The data is **representative** of the process to model

Large image databases

In the last 20 years, the number and size of annotated image databases have increased dramatically

- 100s → several hundred millions image (e.g. ImageNet)
- 10s → several thousands classes
- Annotations at several levels (classes, bounding boxes, contours, etc…)

Some popular image databases: ImageNet, COCO (Common Objects in Context), CIFAR, Pascal VOC, etc…

IMAGENET

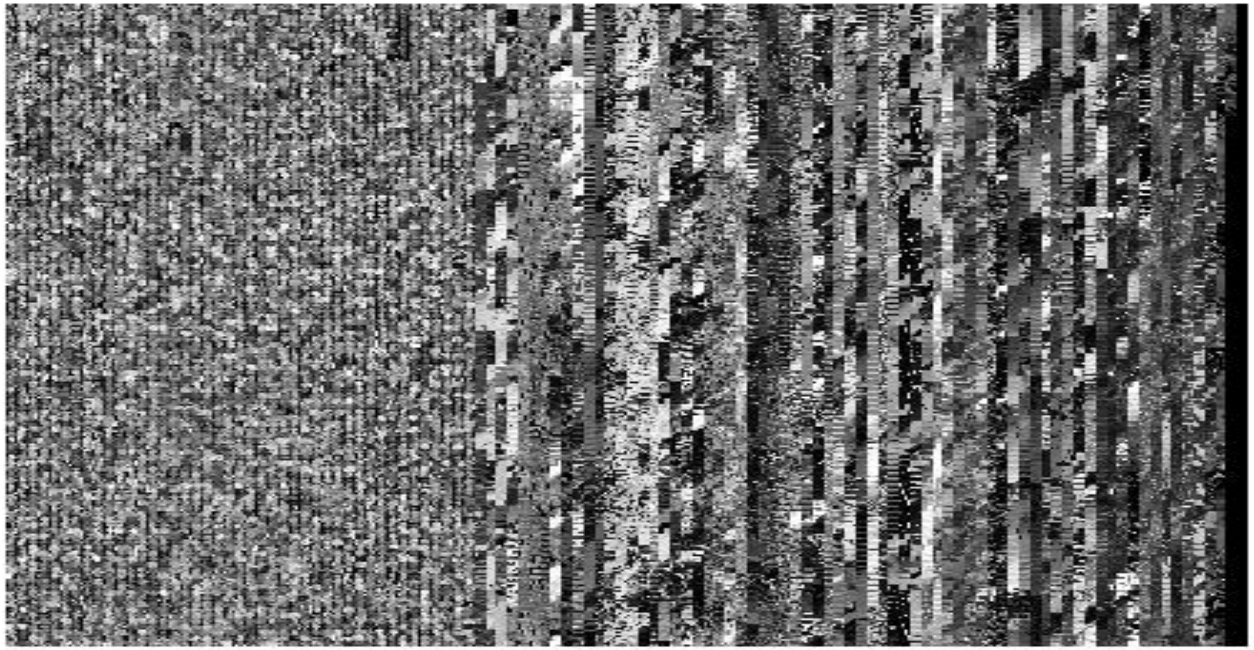
(Semi-, weakly-, fully-) supervised learning has developed dramatically in computer vision

Example of classification for images

Example task: telling if the image shows a face

Annotation: one scalar label per image

image shows a face | image doesn't



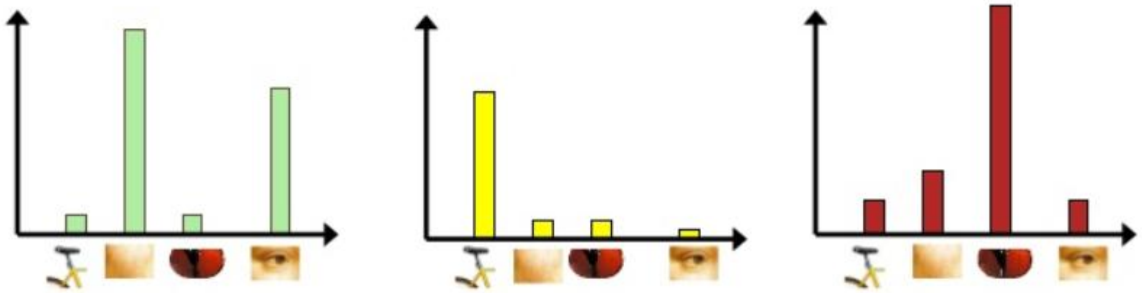
$y_i = 1$ | $y_i = -1$

Classification for images

In general, learning is not performed directly on the images,

but from **features / descriptors** $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$:

- Reduction of the dimensionality
- First stage of information extraction



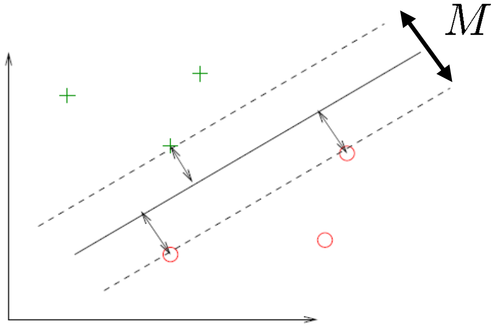
The classification task

- Find the function $h(\boldsymbol{x}) \; : \; \boldsymbol{\mathcal{X}} \mapsto \mathcal{Y}$ which associates a label y to the feature $\boldsymbol{x}$.

h often has a set of parameters θ: $h(\boldsymbol{x}, \theta)$

- For a new feature $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$, use h predict the label $\tilde{y}$

SVM: $h(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$ and $y \in \{-1; 1\}$



This task must be achieved **from a subset of all possible data samples**

Risk and performance

Total risk is defined as: $\mathcal{R}_{\boldsymbol{z}}(\boldsymbol{\theta}) = \int\limits_{\boldsymbol{x},y \in \boldsymbol{\mathcal{X}} \times \mathcal{Y}} \mathcal{L}(y, h(\boldsymbol{x};\boldsymbol{\theta}))p(\boldsymbol{x},y)d\boldsymbol{x}dy$ $\quad z = (\boldsymbol{x}, y)$

with $\mathcal{L}(y, h(\boldsymbol{x};\boldsymbol{\theta}))$ a **loss function** that measures the **cost** of difference between the model's prediction and the true label y

In practice, $p(\boldsymbol{x}, y)$ is not know, and we can only compute the **empirical risk**:

$$\mathcal{R}^N_{\boldsymbol{z}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mathcal{L}(y_i, h(\boldsymbol{x}_i;\boldsymbol{\theta})),\ \boldsymbol{x}_i \in \boldsymbol{X}, y_i \in Y.$$

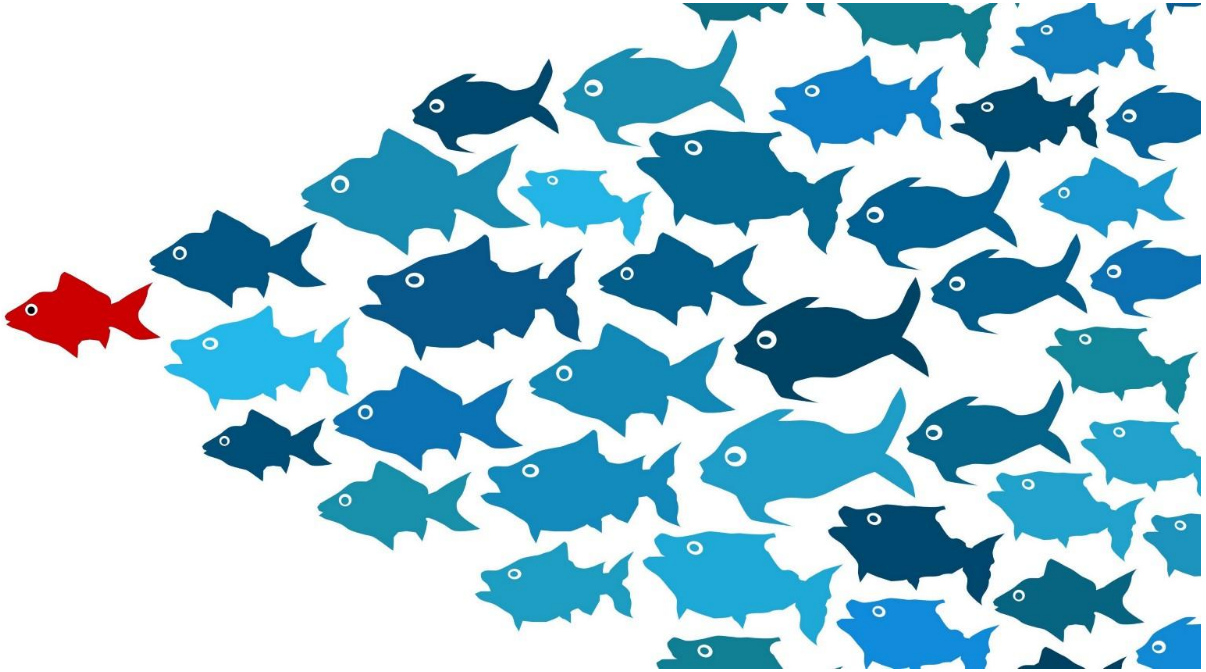
Training of the classifier:

Optimisation (e.g. gradient descent) on **minimizing the empirical risk**

Training a classifier, measuring performance

Problem: we only have finite number of data samples in our dataset

→ How do we know that the empirical risk is **representative** of the real risk?



Training a classifier, measuring performance

Problem: we only have finite number of data samples in our dataset

→ How do we know that the empirical risk is **representative** of the real risk?

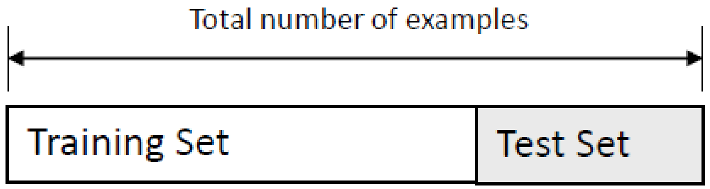
It can be shown that for a sufficiently large dataset:

- the empirical risk tends towards the real risk
- the model's parameters θ tend towards the optimal model parameters

→ How large is large enough?

Training a classifier, measuring performance

If we divide our dataset into two subsets



It can be shown that:

- the empirical risk *computed on the test set* tends towards the real risk
- the model's parameters θ, *learnt on the training set*, tend towards the optimal model parameters

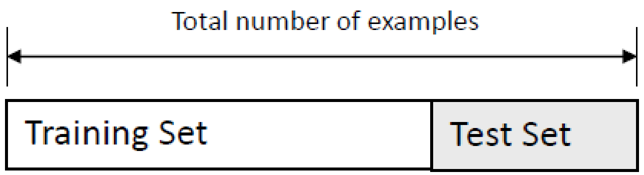
→ We can test **generalisation**

→ How large is large enough?

In practice, the empirical risk on the test set is an **estimator** of the real risk

Training a classifier, measuring performance

If we divide our dataset into two subsets

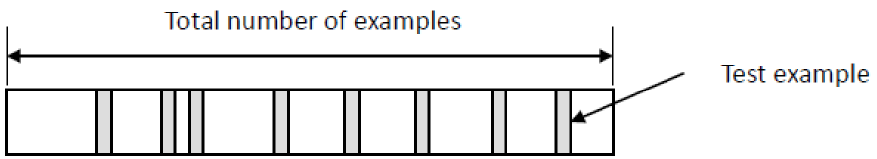


Caveats:

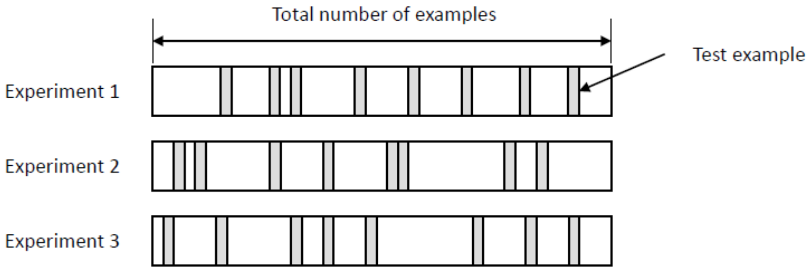
If the test set is too small or badly chosen, the empirical risk deviates from the real risk

Some (very recommended) solutions:

- **Random sampling**:



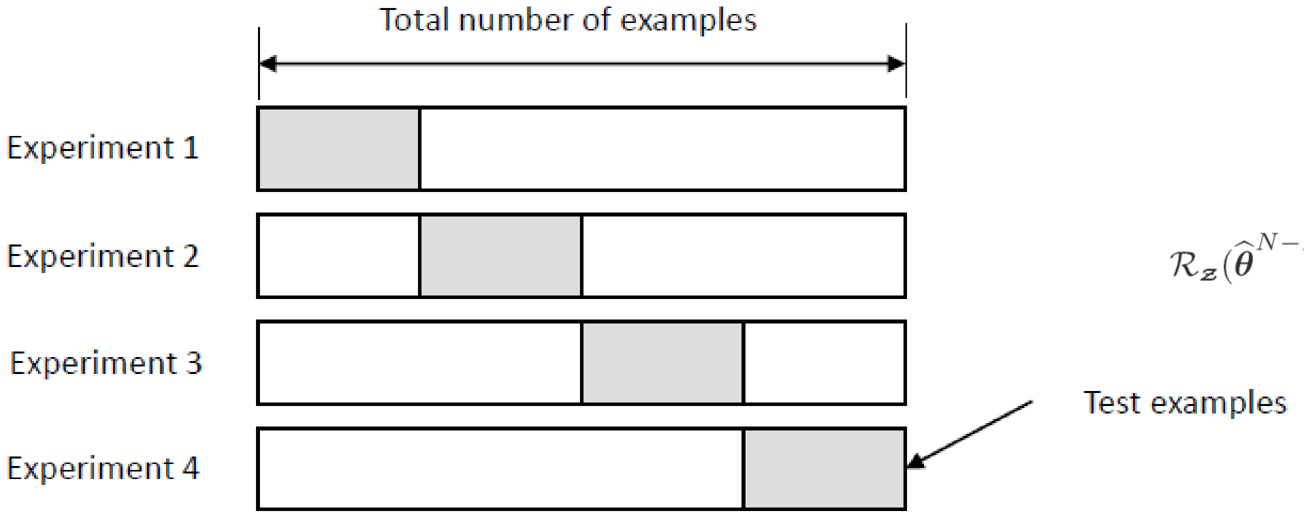
- **Averaging** over several trainings:



$$\mathcal{R}_{\boldsymbol{z}}(\widehat{\boldsymbol{\theta}}^{N_1}) \approx \frac{1}{K}\sum_{i=1}^{K}\mathcal{R}_{\overline{\boldsymbol{z}}}^{N_2,i}(\widehat{\boldsymbol{\theta}}^{N_1,i})$$

Training a classifier, measuring performance

Better, more systematic approach: **K-fold cross validation**



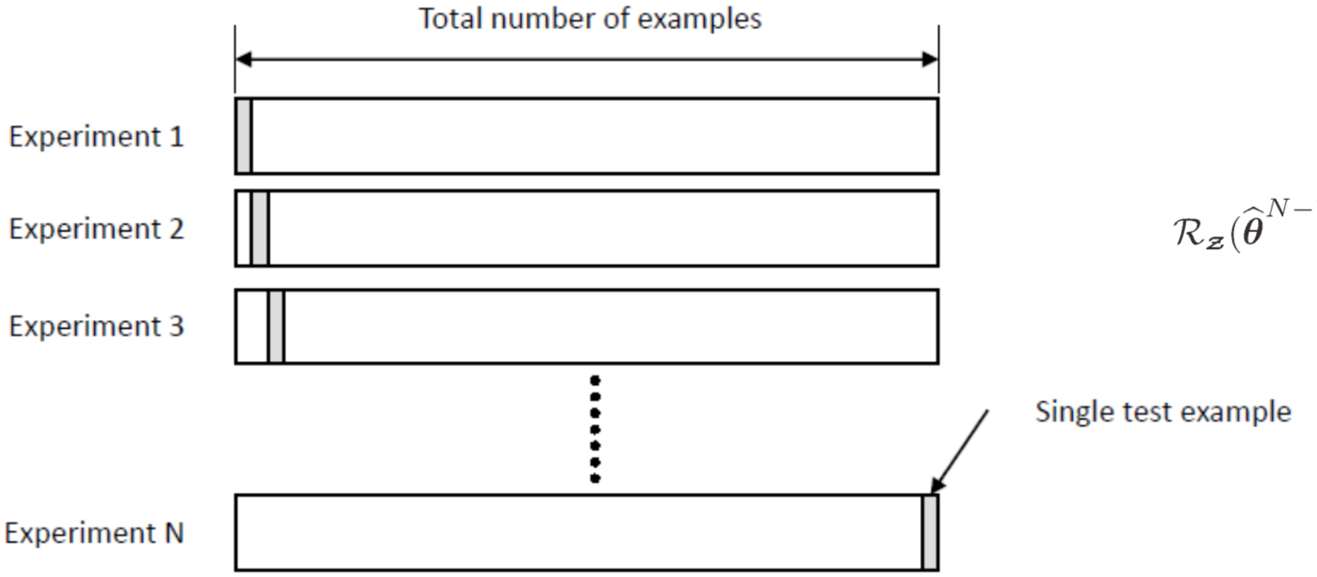
$$\mathcal{R}_{\boldsymbol{z}}(\widehat{\boldsymbol{\theta}}^{N-K}) \approx \frac{1}{K}\sum_{i=1}^{K} \mathcal{R}_{\overline{\boldsymbol{z}}}^{K,i}(\widehat{\boldsymbol{\theta}}^{N-K,i})$$

Benefits on small datasets:

- All data samples are used (once) for testing
- The training set can be larger
- More folds → smaller bias of the risk estimator (but larger standard deviation)

Training a classifier, measuring performance

For very small datasets: **Leave-one-out cross validation**



$$\mathcal{R}_{\boldsymbol{z}}(\widehat{\boldsymbol{\theta}}^{N-1}) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{R}_{\boldsymbol{z}}^{1,i}(\widehat{\boldsymbol{\theta}}^{N-1,i})$$

Benefits on small datasets:

- Same as K-fold cross validation +
- Maximises the training set size

Cons:

- Lots of computations for training

Training and model selection

In addition to optimising the θ parameters of the model,

several algorithms need to optimise hyper-parameters: choice of model

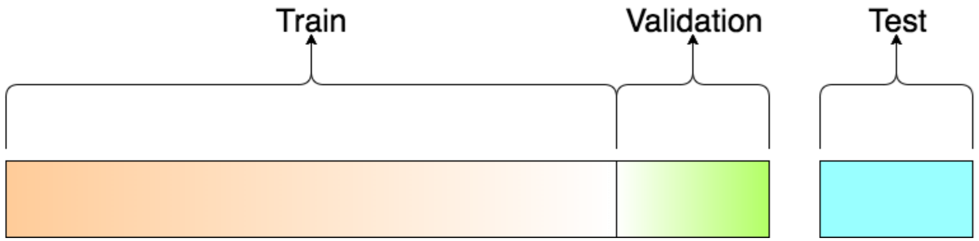
Examples:

- Neural networks: numbers of layers and neurons
- SVM: regularisation constant
- Kernel-SVM: regularisation constant, parameters of the kernel

Never optimise these hyper-parameters on the same training set as the model's parameters θ

Training and model selection

Three subsets are needed:



Proceed in 7 steps:

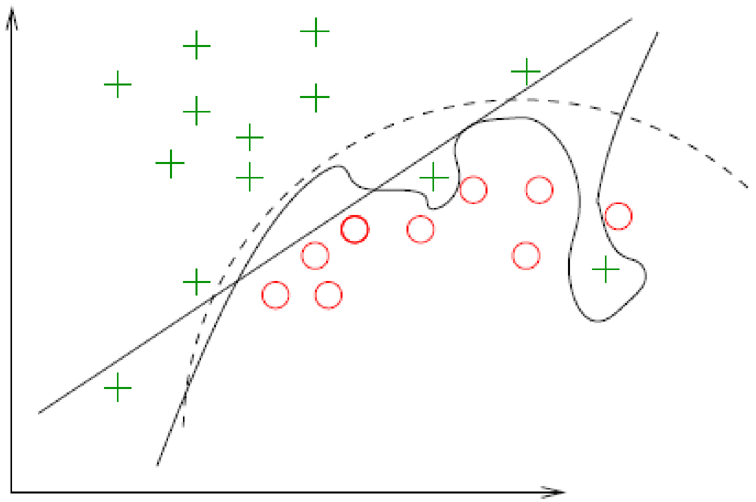
1. Divide the dataset into training, testing, and validation sets
2. Pick a model (e.g. architecture of neural network)
3. Train the model on the training set
4. Evaluate the model on the validation set
5. Repeat steps 2 to 4 with several models / architectures
6. Select the best performing model
7. Do a classic K-fold cross validation with the training and testing sets

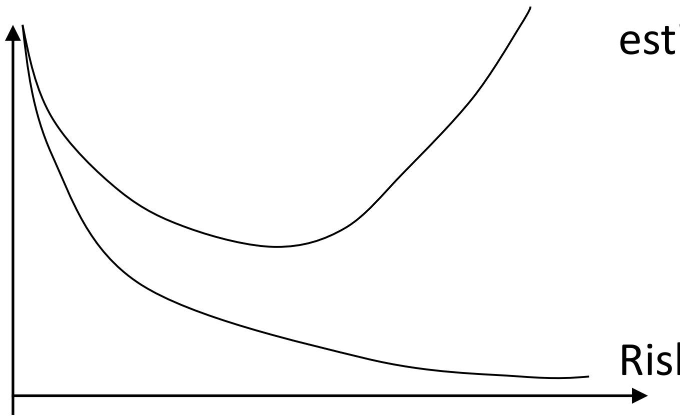
Optimisation of the hyper-parameters

Optimisation of the parameters

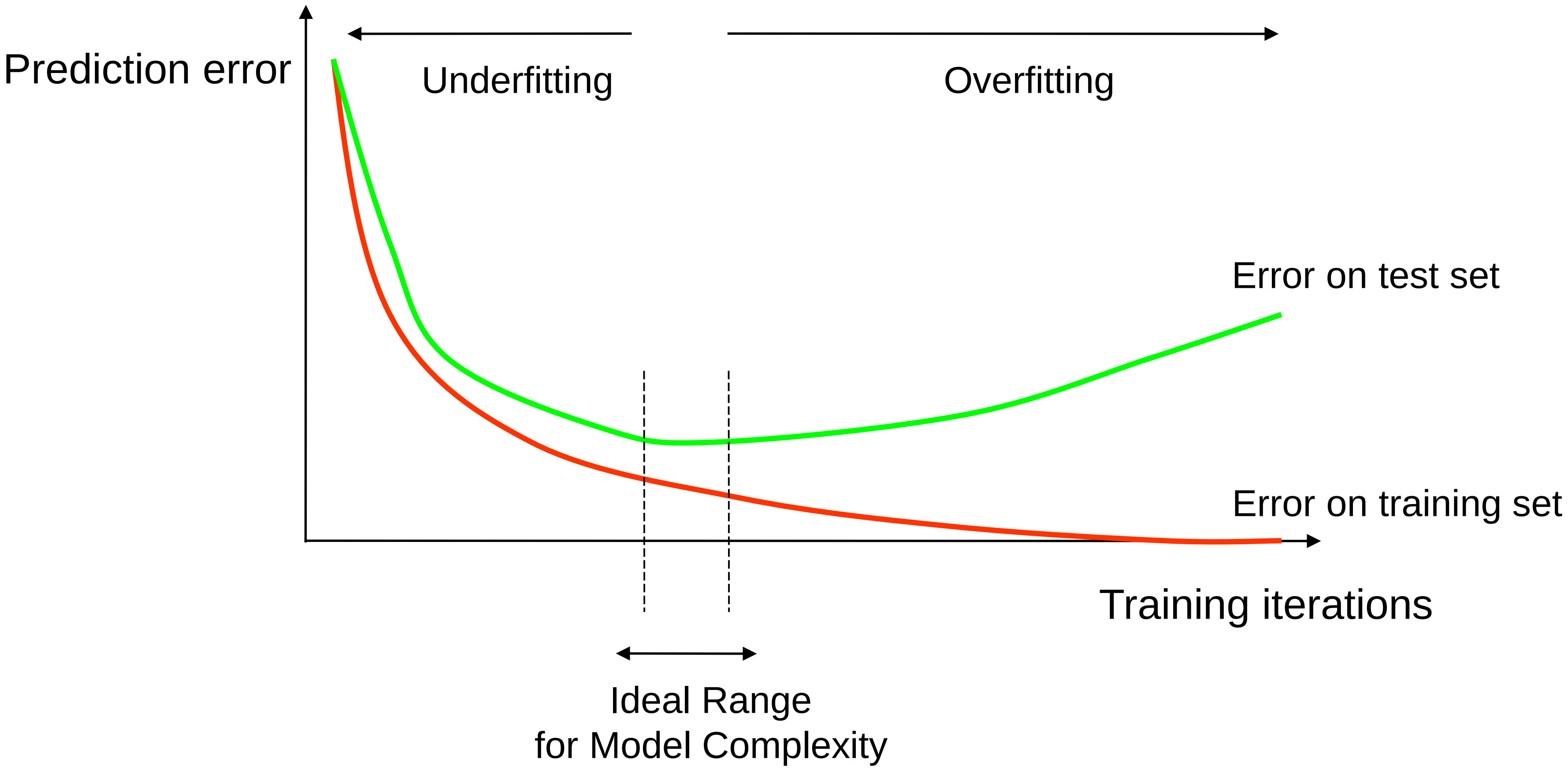
Empirical risk minimisation

The more complex the model, the higher the risk of **over-fitting**





Empirical risk minimisation

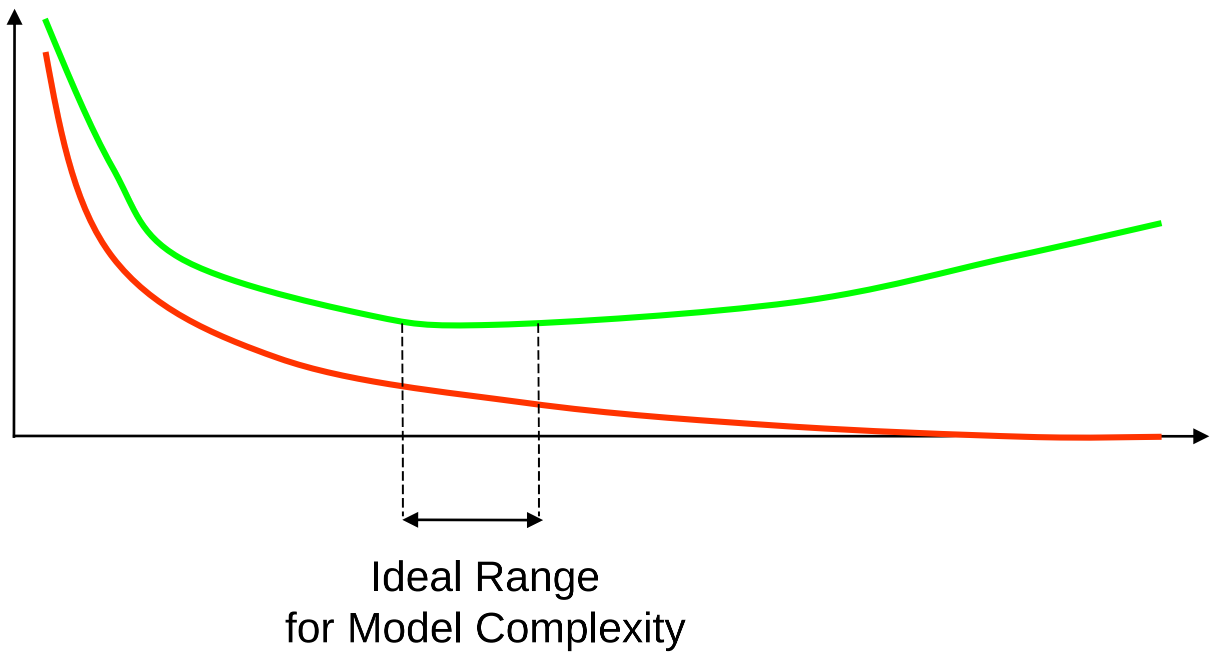


When the classifier's memory increases, it relies more on memory and it becomes harder to classify unknown samples. It's ability to generalise decreases.

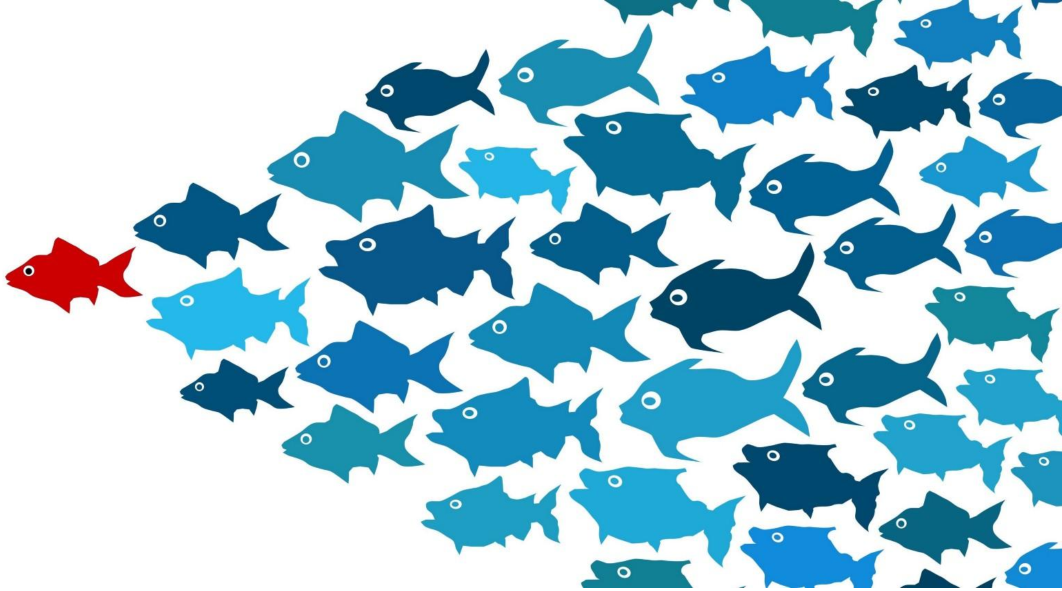
Empirical risk minimisation

2 common solutions:

- Stop training when the error computed on an additional validation set starts increasing
- “Early stopping”: stop training while the testing error is still decreasing a bit



Unbalanced datasets

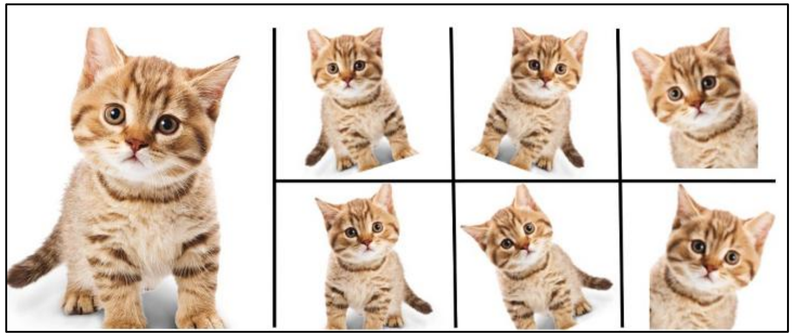


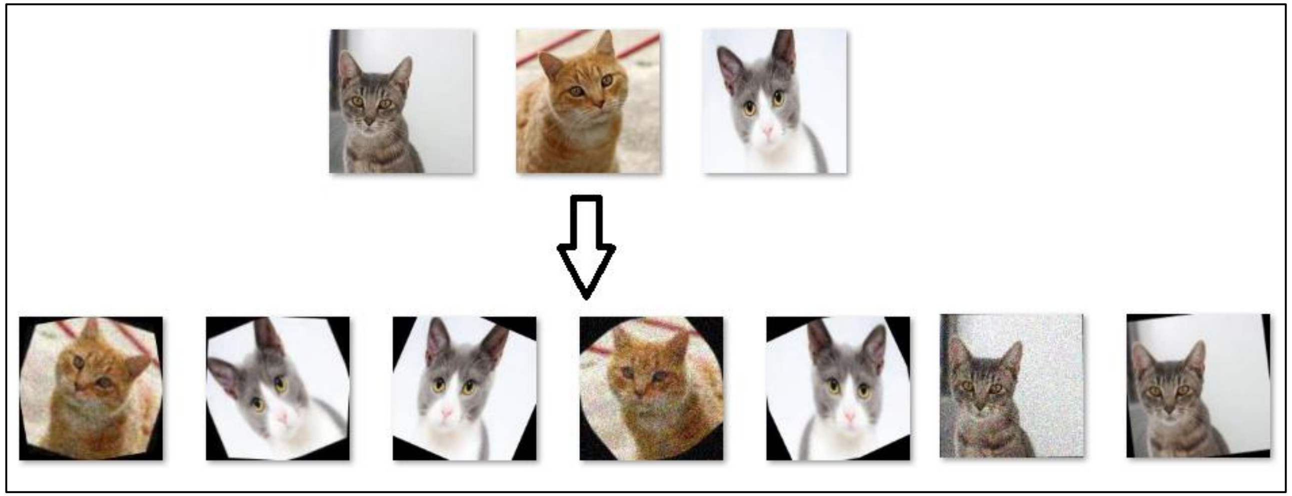
- A very common problem with real world datasets!

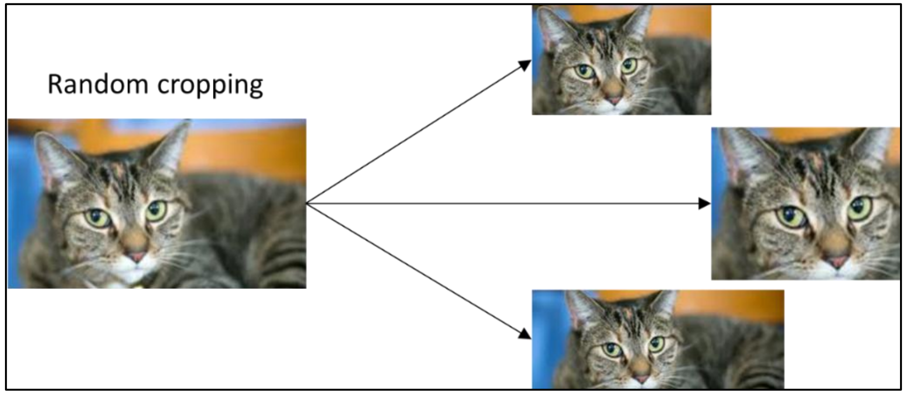
- A very important problem to consider:
    - Overfitting to the common class
    - Ignoring the rare class

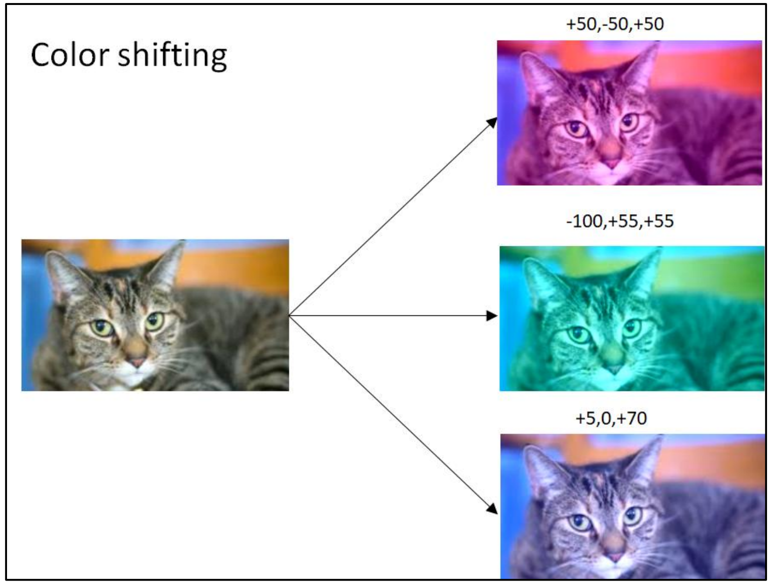
- Possible solutions:
    - Getting more data
    - Re-sampling (stratified sampling, over-sampling)
    - Generate synthetic samples?
    - Try different algorithms
    - Try a different perspective (outlier detection? Optimising a different cost?)

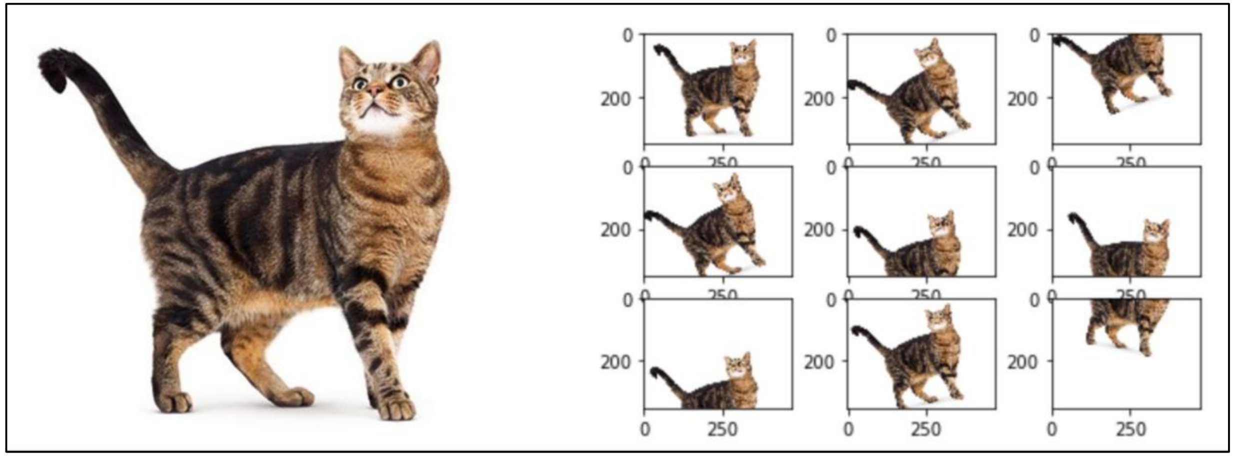
Data augmentation











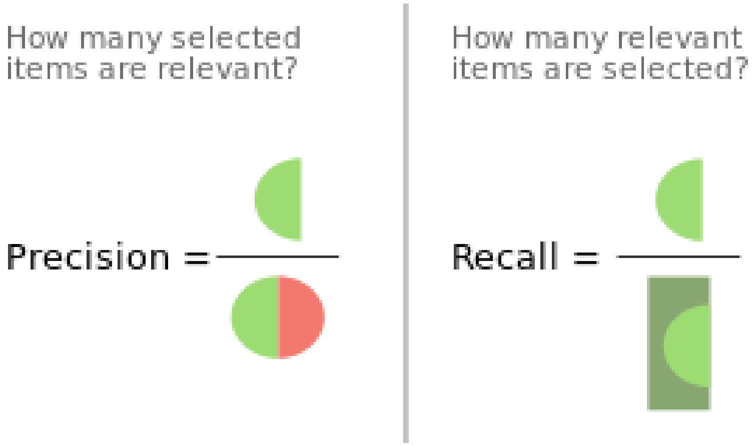
Data augmentation

Measuring the performance of a binary classifier

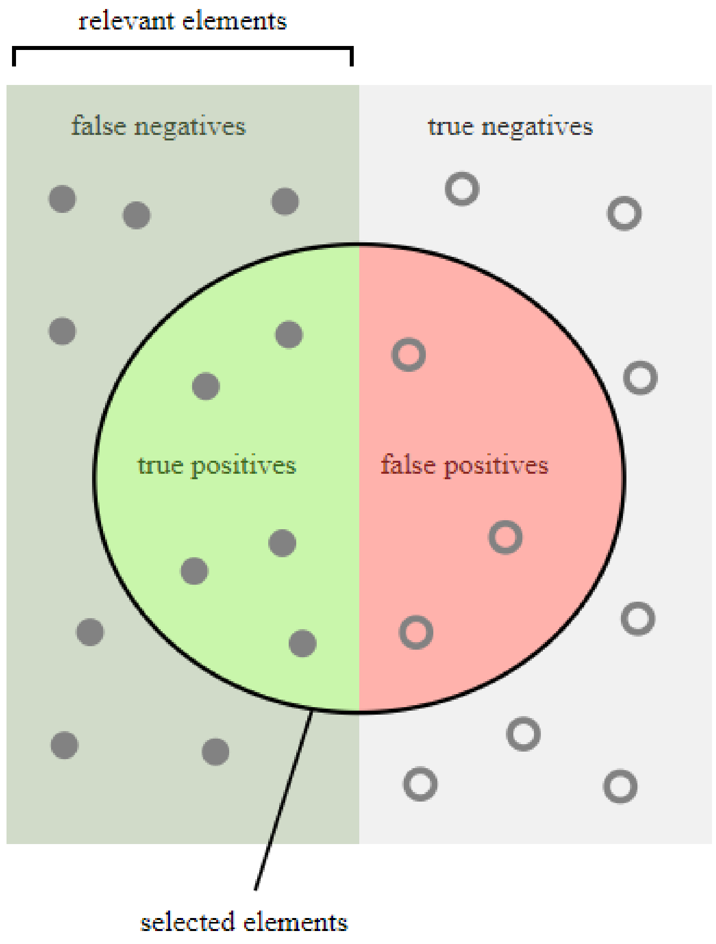
Standard measures:

- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$

complementary



- F1-score: combines the previous two
- Accuracy: biased on unbalanced data, avoid it
- Etc.

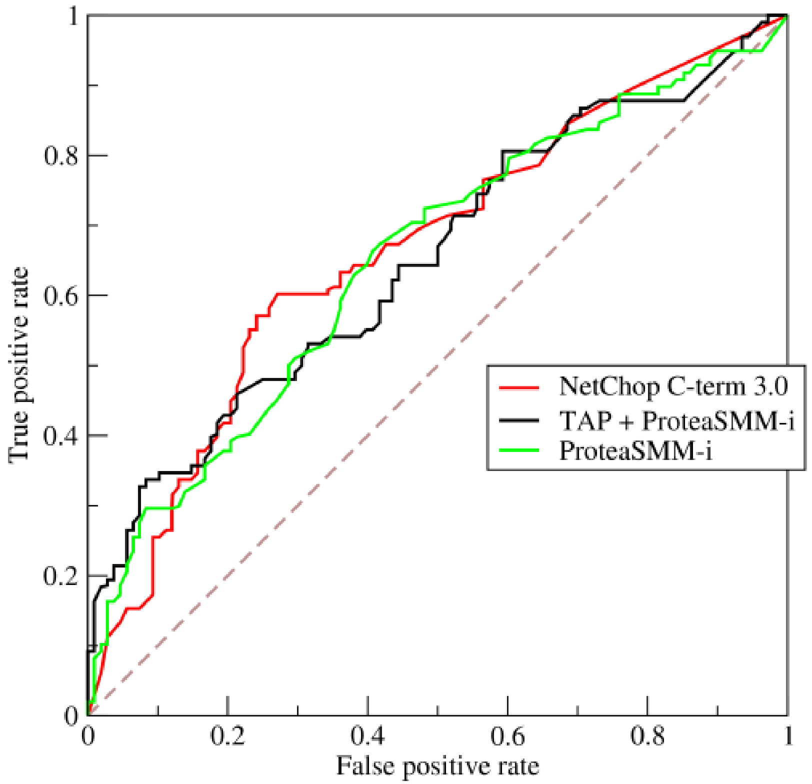


https://en.wikipedia.org/wiki/Precision_and_recall

ROC curve

Standard way to presenting performance measures for a binary classifier

- True positive rate (sensitivity) as a function of false positive rate (1 – specificity)
- Area under curve (AUC) measures the difference to a random classifier (AUC = 0.5)



Where can we find learning algorithms?

Python

Nowadays the standard language for data mining and machine learning

Libraries:

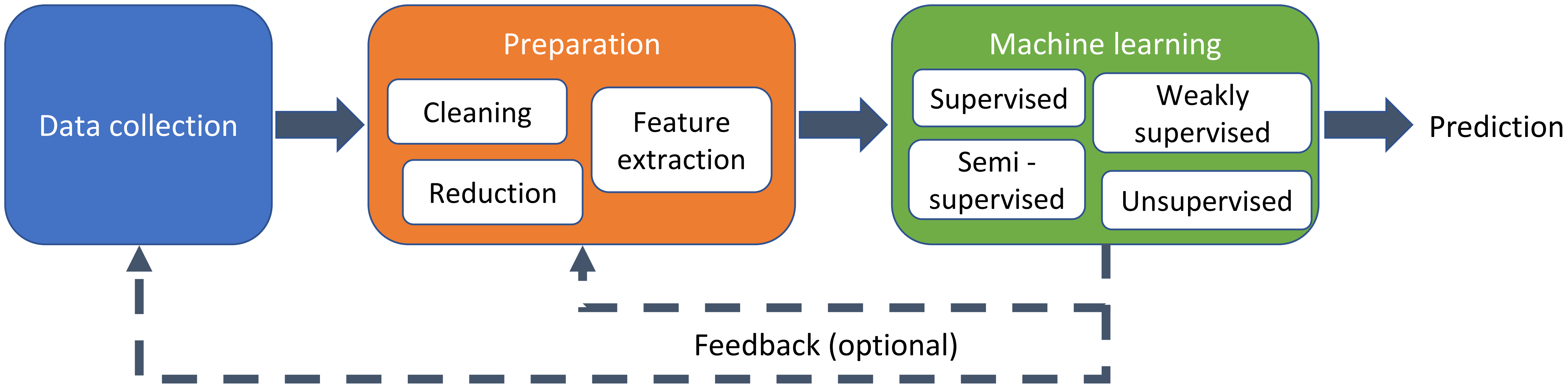
- SciKit-Learn: many supervised and unsupervised algorithms

- PyTorch: Most popular frameworks for deep learning
  - including high-level wrapper such as PyTorch Lightning

Neural networks and deep learning

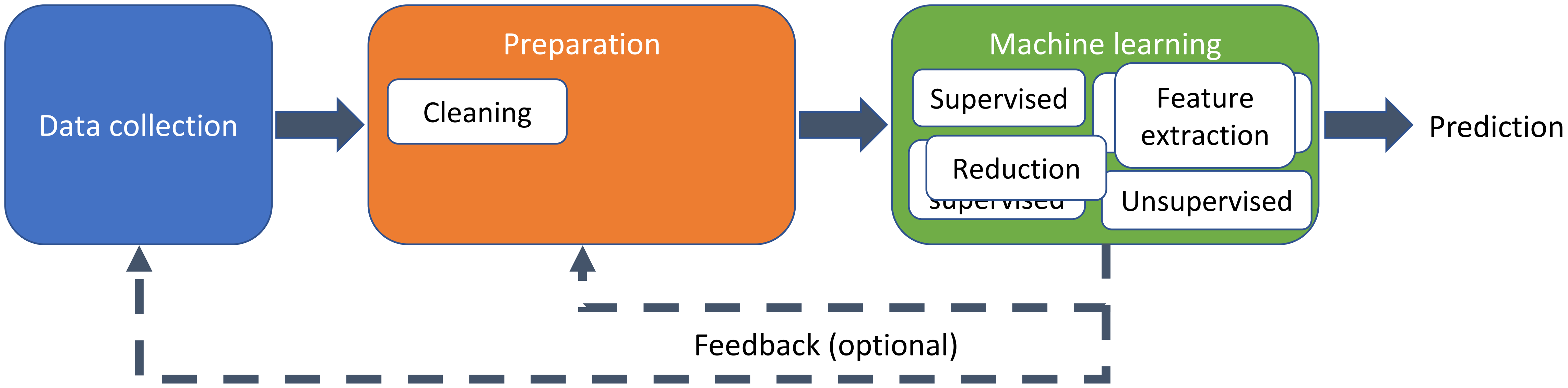
Why have neural networks become so popular?

Generic data processing pipeline



Generic data processing pipeline

Deep learning version:

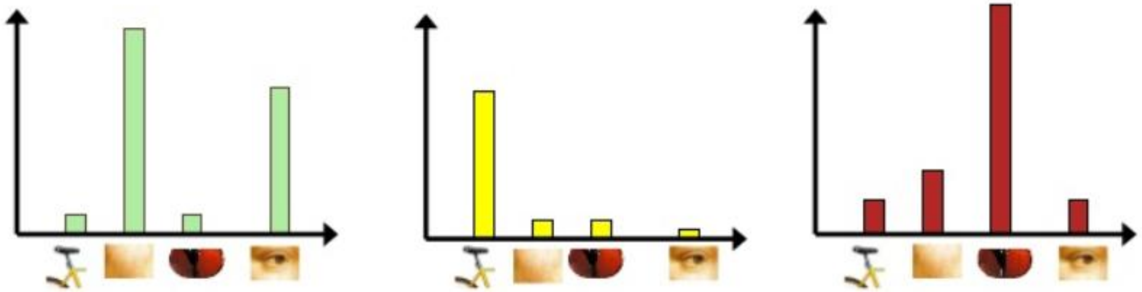


Integrates the feature extraction and reduction steps into the learning

→ Very popular in computer vision!

“In-the-box” feature extraction

- Classical computer vision:



Machine learning

Supervised

Weakly supervised

Semi - supervised

Unsupervised

Prediction

- Computer vision with deep learning:

Machine learning

Supervised

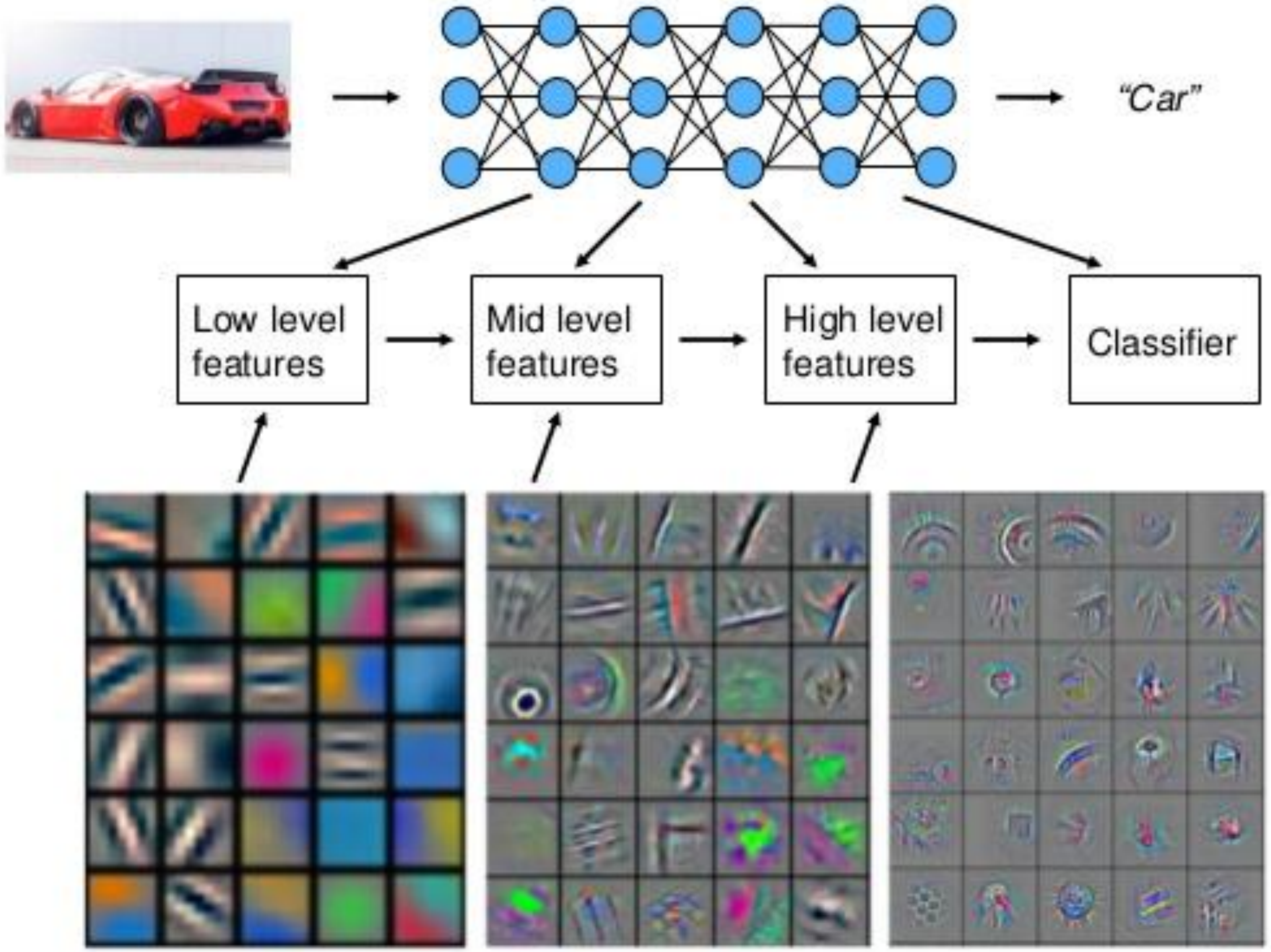
Weakly supervised

Semi - supervised

Unsupervised

Prediction

“In-the-box” feature extraction

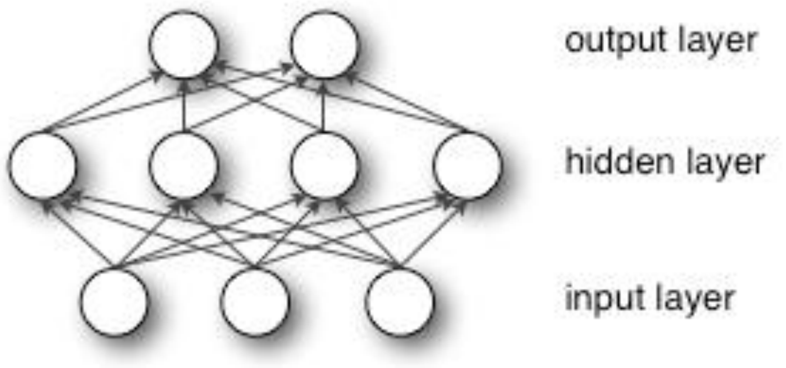


How do neural networks work?

Supervised learning scenario:

Find the function $h(\boldsymbol{x}) : \boldsymbol{\mathcal{X}} \mapsto \mathcal{Y}$ which associates a label y to the data sample $\boldsymbol{x}$.

- h is implemented by a set of neurones, organised into layers
- θ are the parameters of the neurones

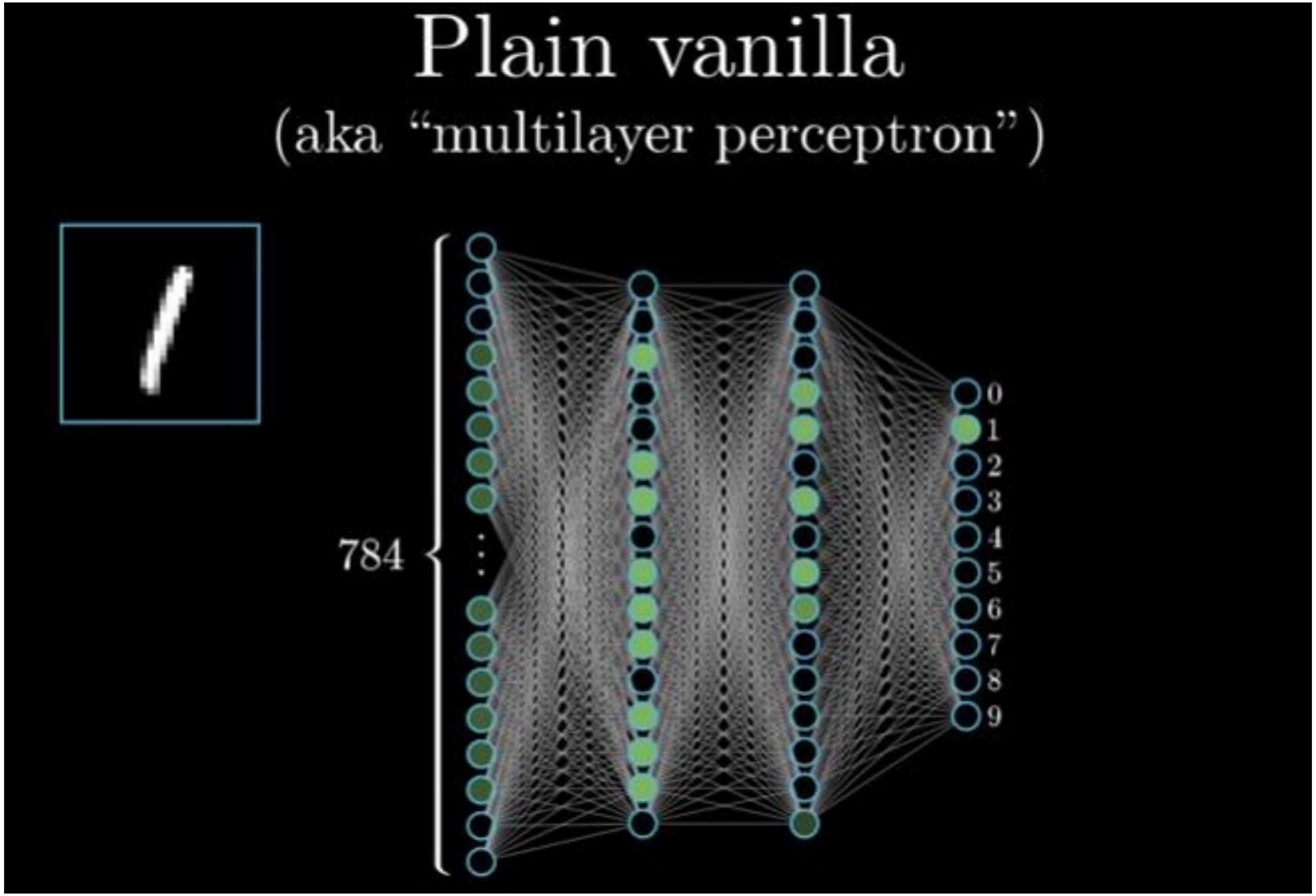


- h and its parameters θ are optimised by gradient descent to minimize an empirical risk

Neural network are “normal” machine learning algorithms

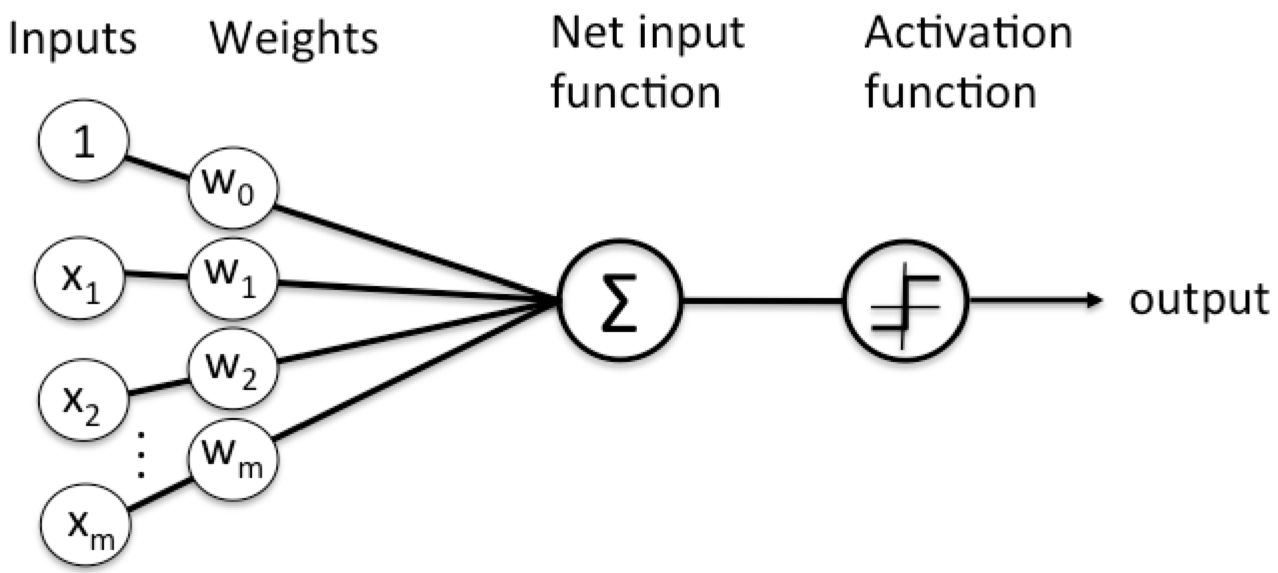
How do neural networks work?

3Blue1Brown



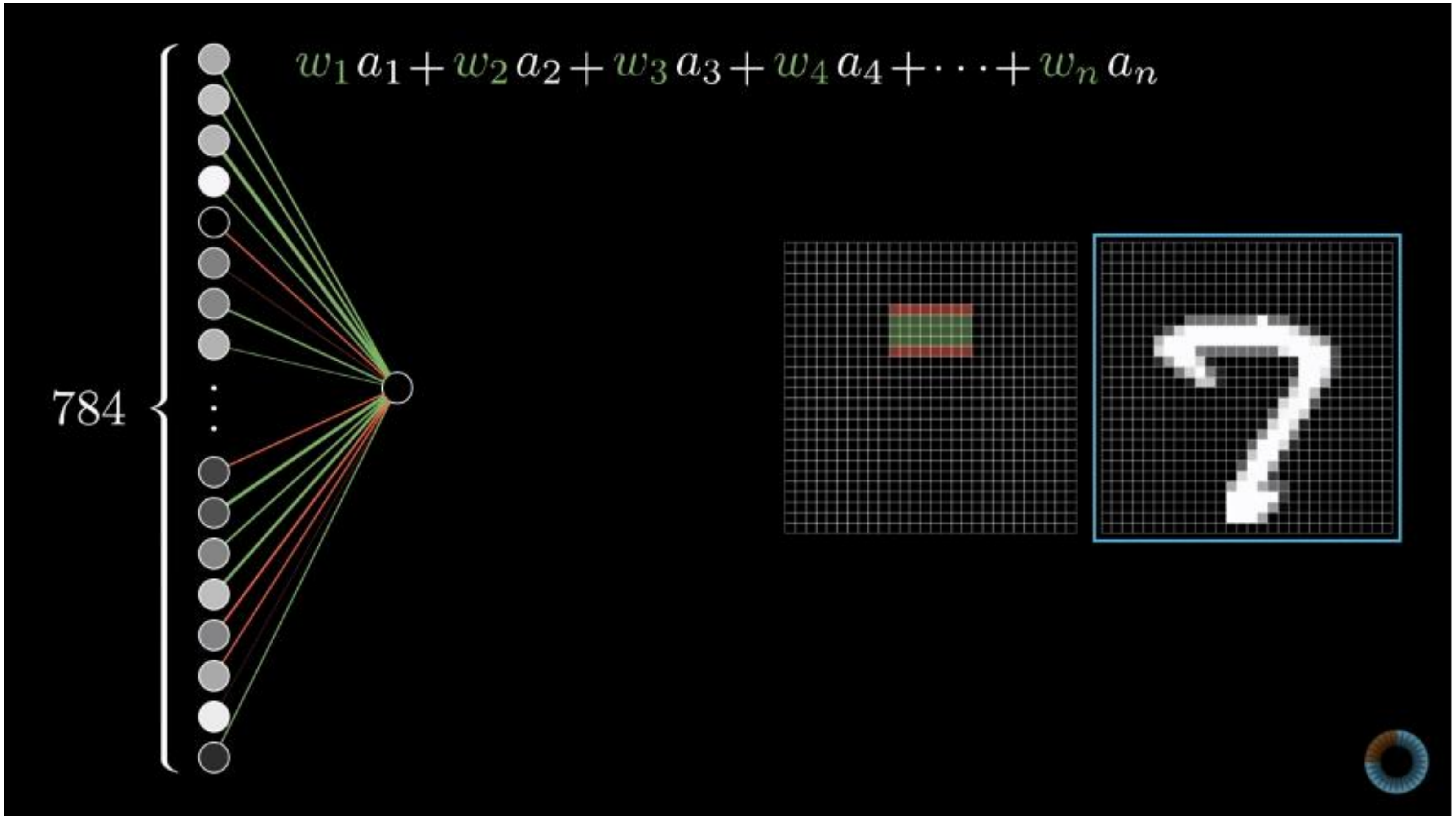
https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

What is a neurone?

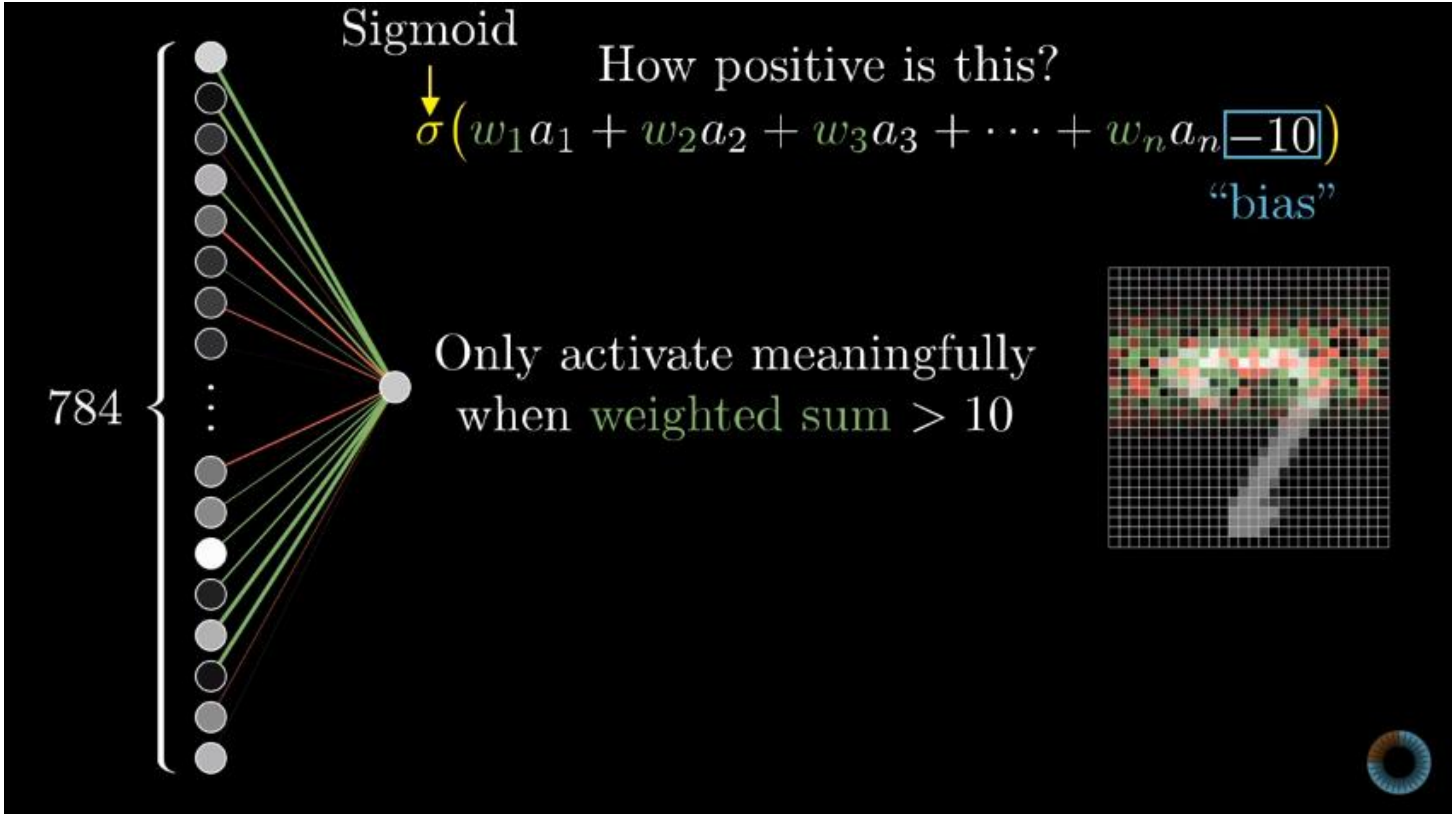


A neuron as an edge detector

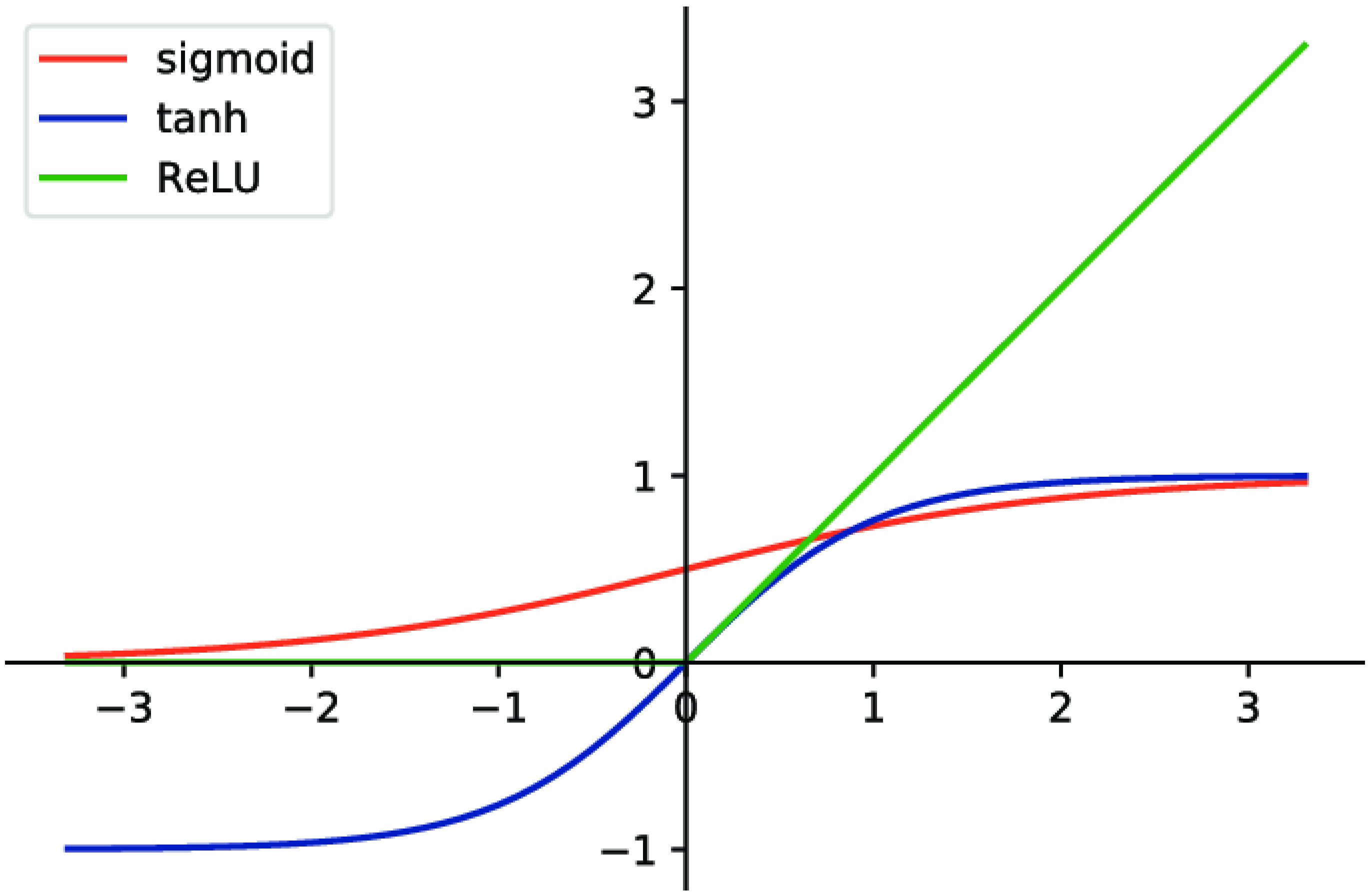
This is only an example of what could trigger a neuron!



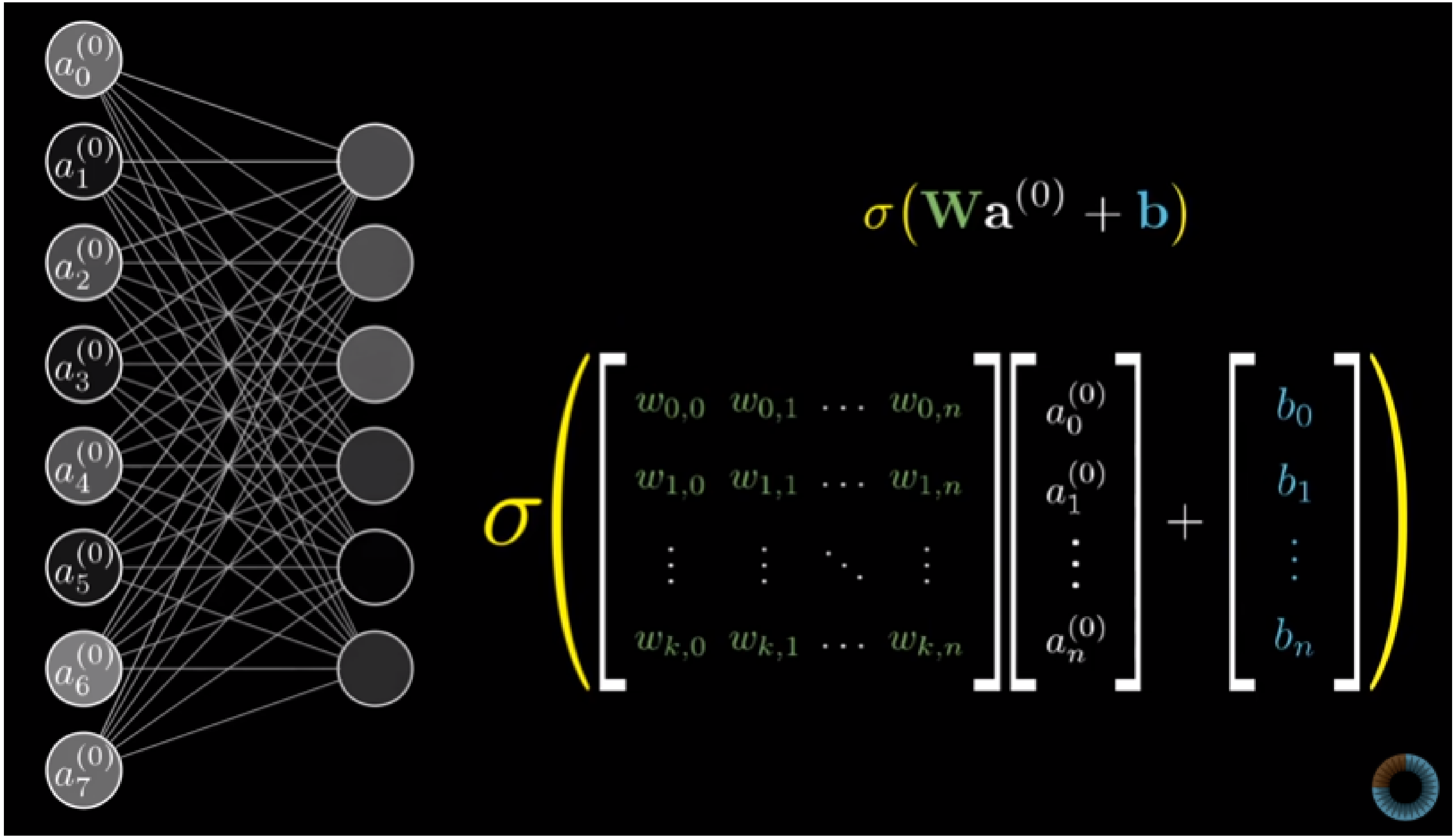
Triggering of a neuron



Some activation functions

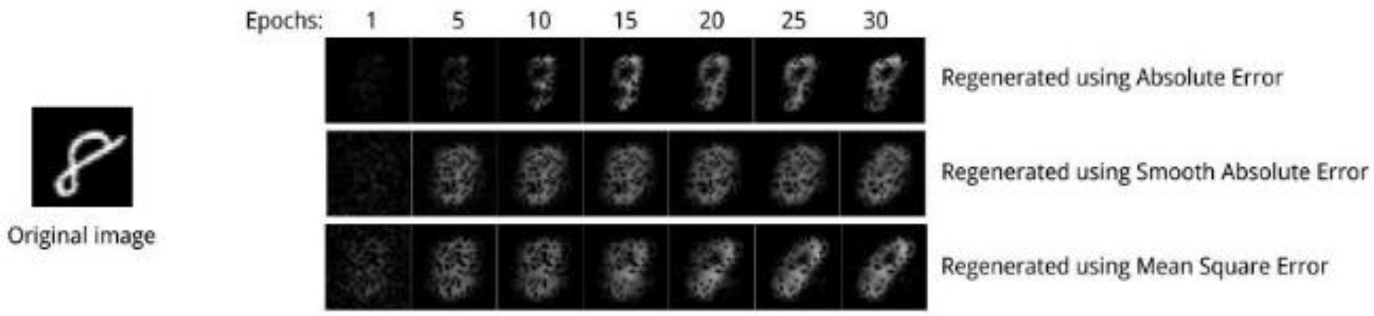


What is a neurone?



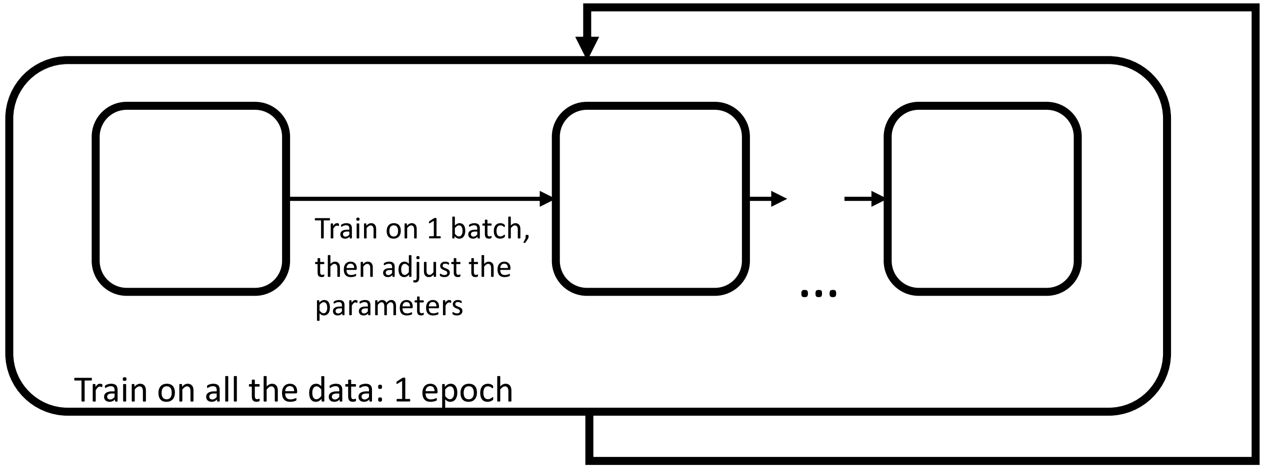
How to train a neural network

- Minimisation of the empirical risk $\mathcal{R}_{\boldsymbol{z}}^N(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mathcal{L}(y_i, h(\boldsymbol{x}_i; \boldsymbol{\theta})),\ \boldsymbol{x}_i \in \boldsymbol{X}, y_i \in Y.$
- Choice of loss function:
  - For classification:
    - (Binary) Cross Entropy loss (“softmax”)
    - Negative Log Likelihood loss
    - Margin loss
    - Soft Margin loss
    - Kullback Leibler (KL) Divergence
    - Etc.
  - For regression:
    - Euclidean error
    - Mean Absolute error
    - Mean Squared Error (Quadratic loss)
    - Mean Squared Logarithmic Error
    - Etc.
  - For autoencoders:
    - Absolute loss
    - Mean Square loss
    - Smooth Absolute loss
    - Etc.

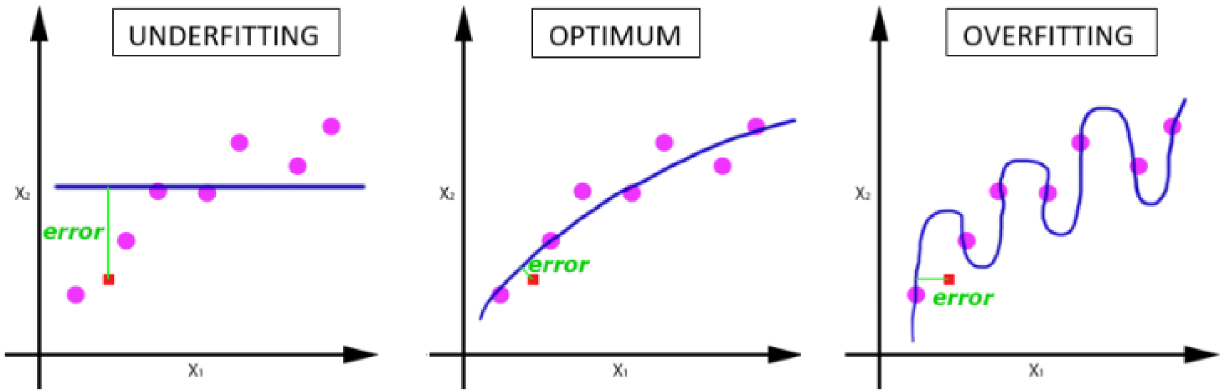


How to train a neural network

Procedure for training by gradient descent:



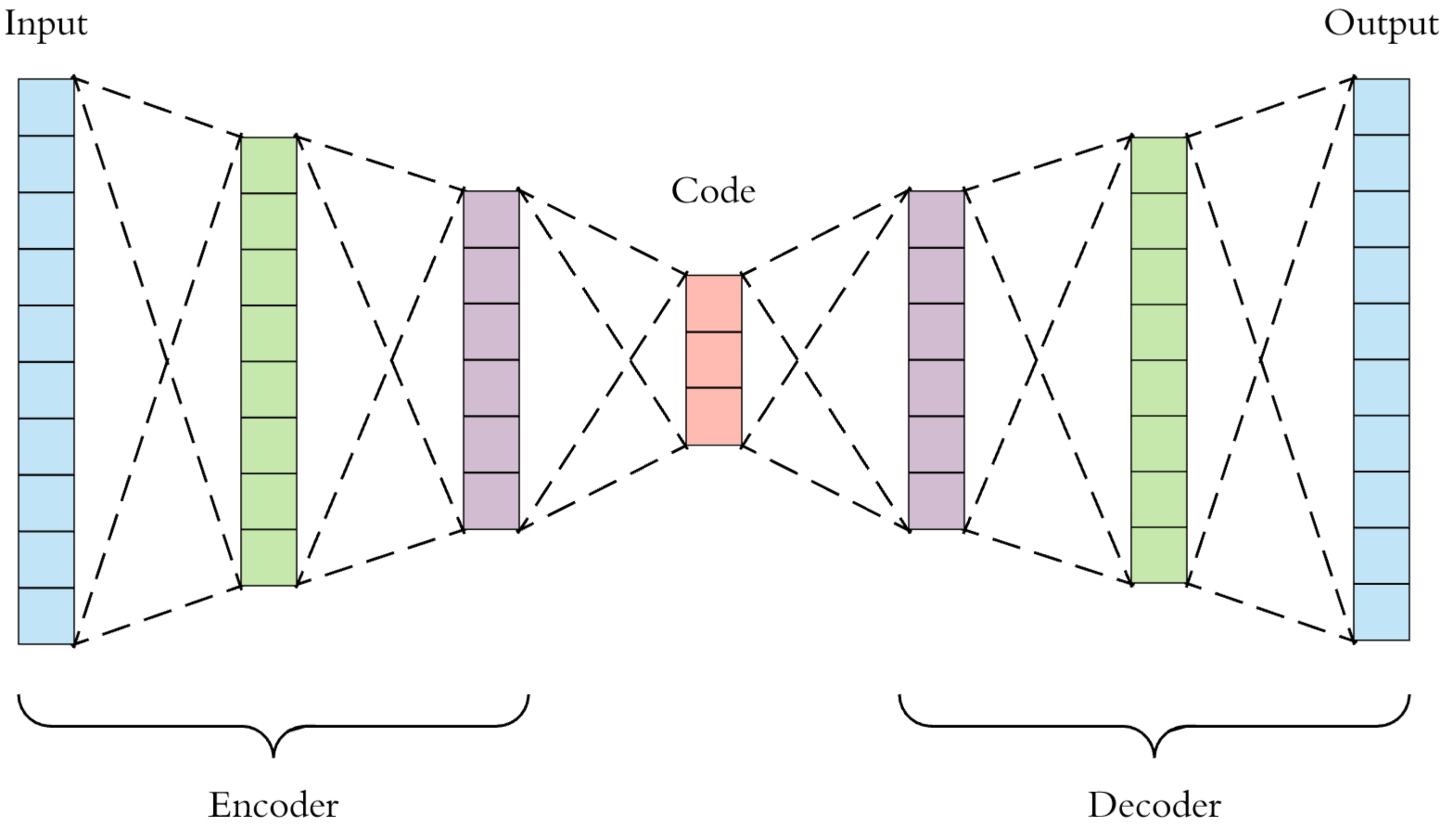
- Gradual and iterative updates



- Smoothed by the use of batches

Some types of neural networks

- Autoencoders

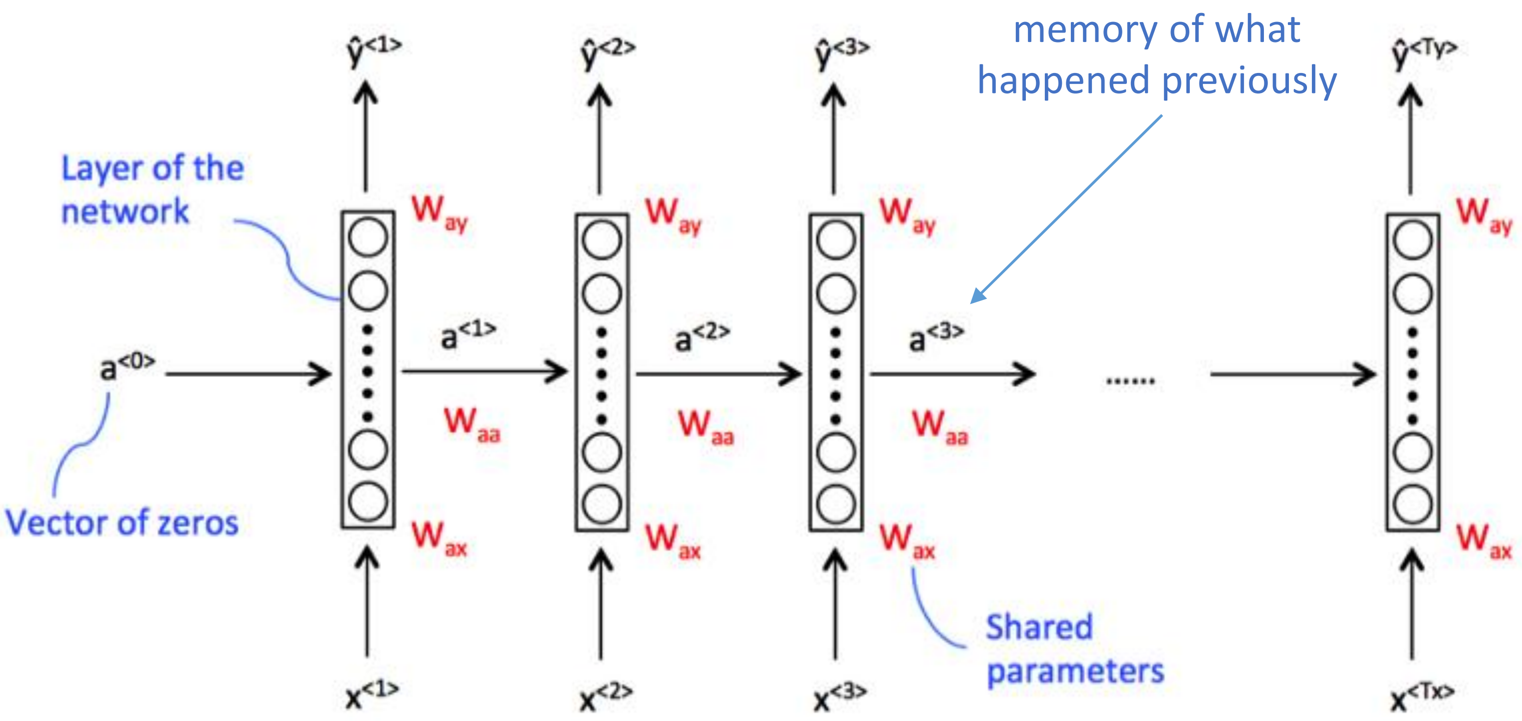


Possible uses: data reduction, denoising…

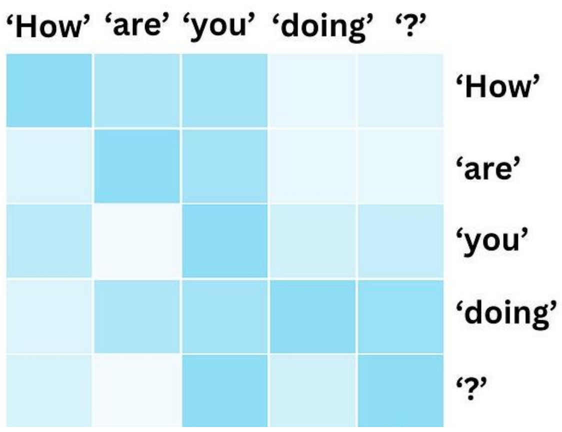
Some types of neural networks

- Recurrent neural networks

Used for sequences of data (e.g. temporal series)



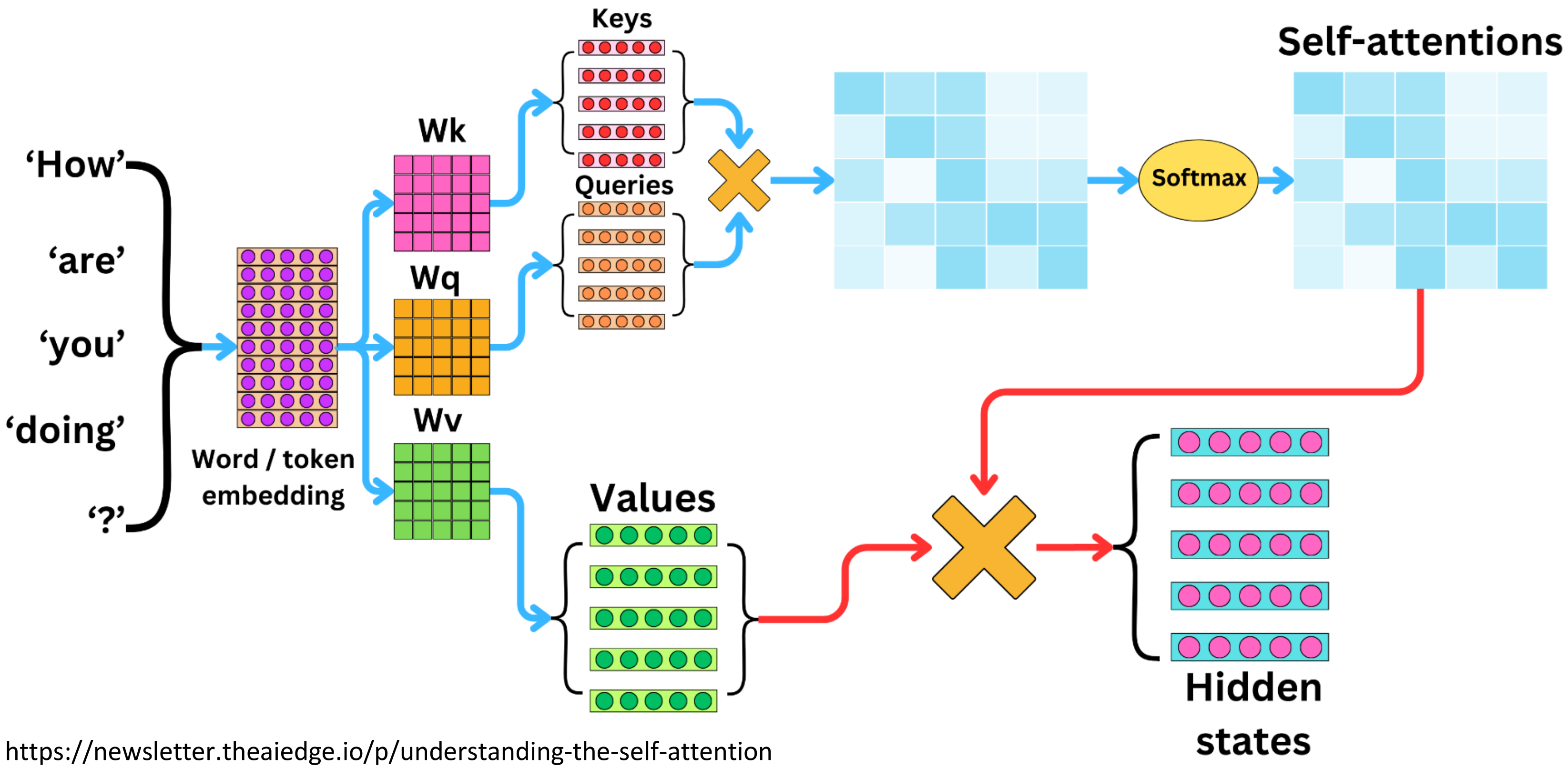
Some types of neural networks



- Transformers

Used for sequences of data, may be adapted to visual data

Self-attention modules capture relationships within the data

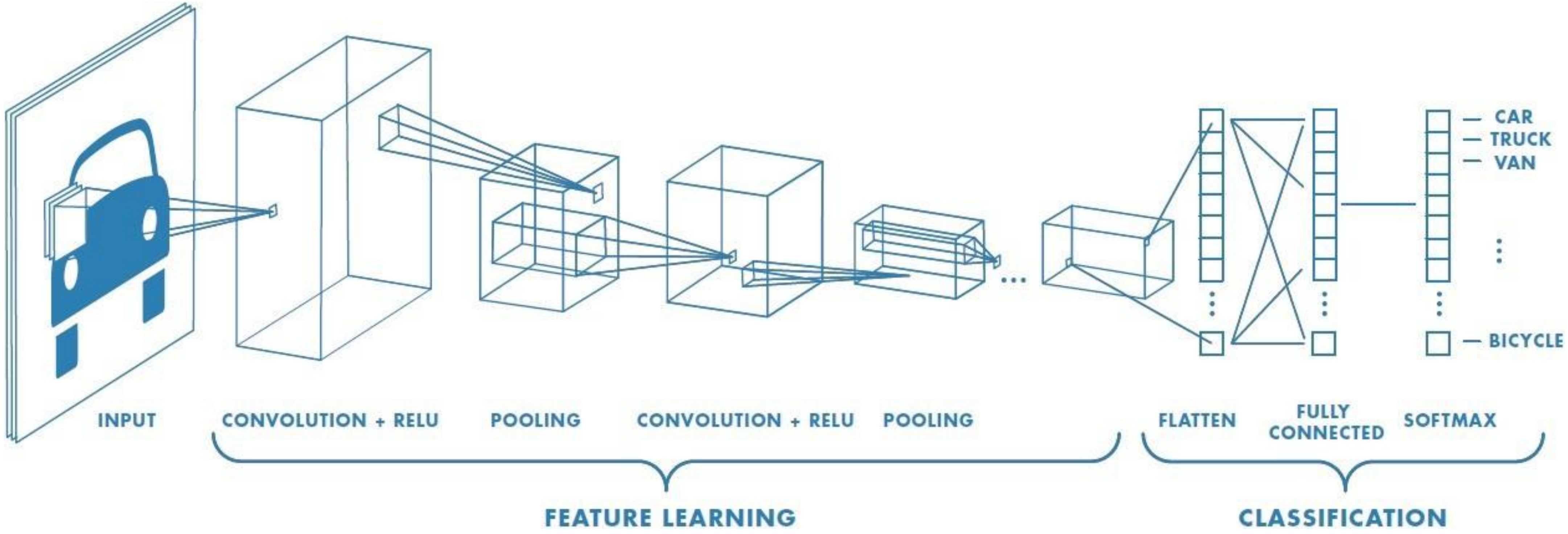


https://newsletter.theaiedge.io/p/understanding-the-self-attention

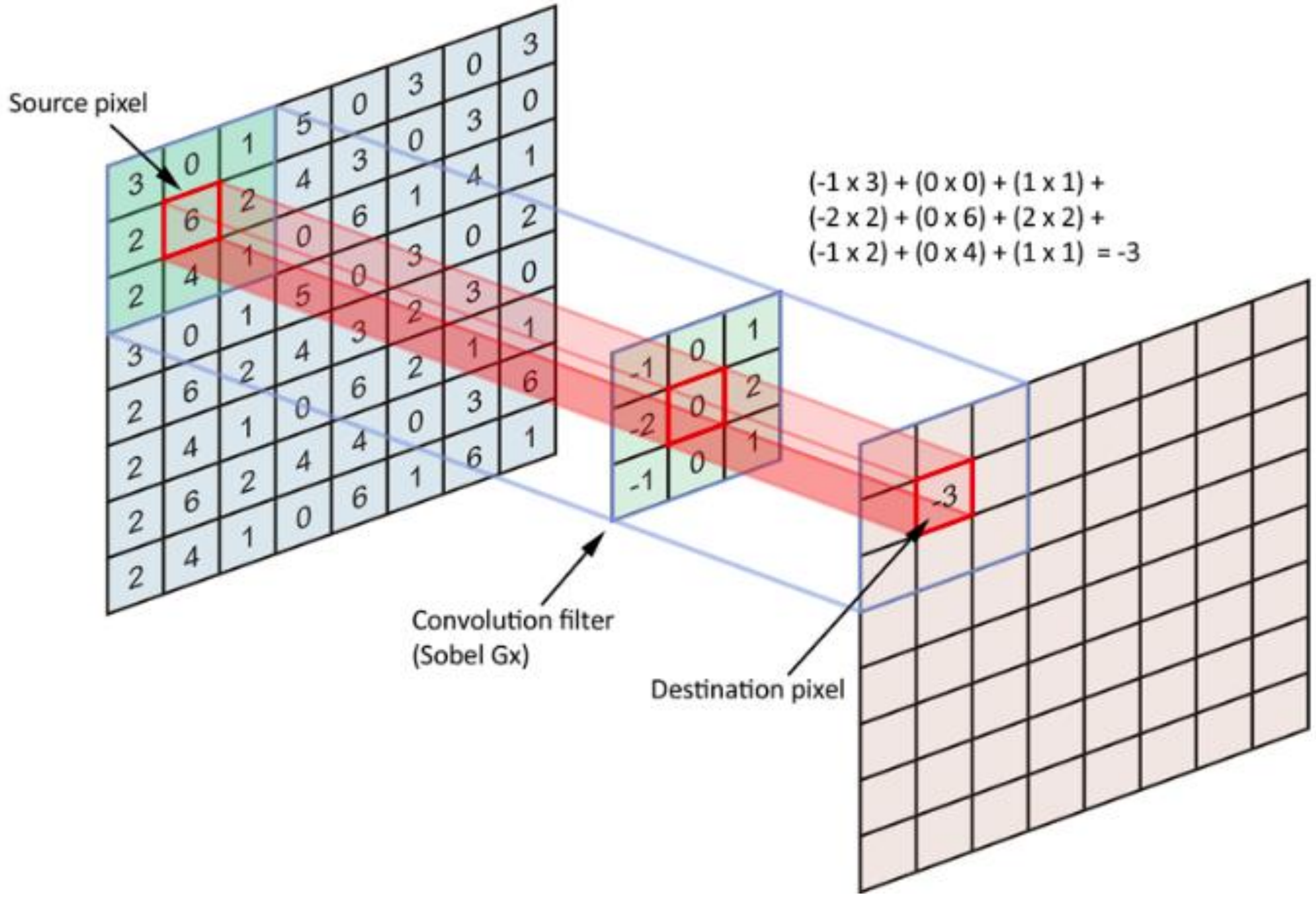
Some types of neural networks

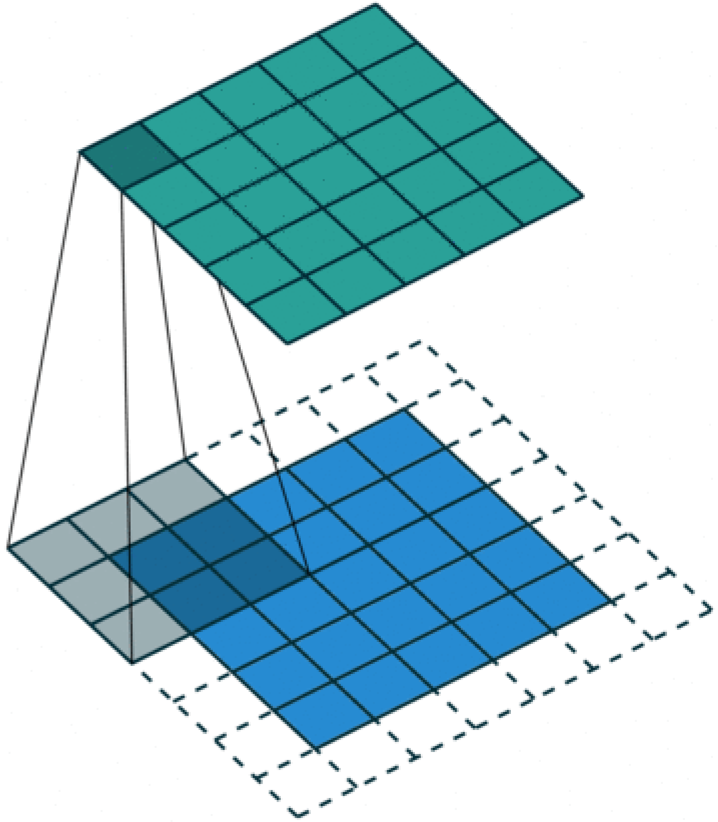
- Convolutional neural networks (CNNs)

Used for images



Convolution operation

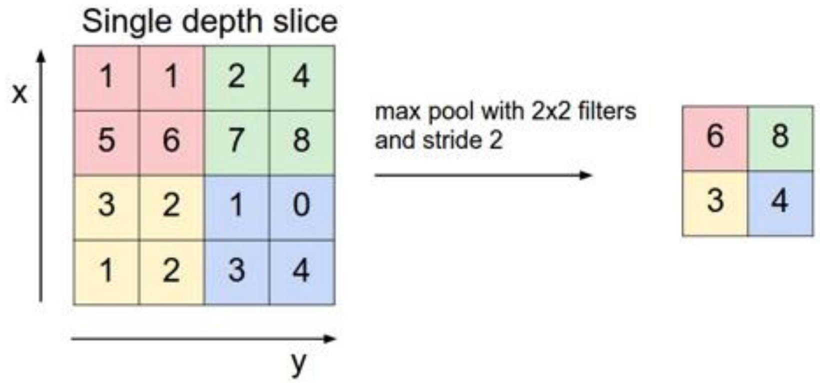




Same filter (neuron) applied to **all spatial locations** → translational invariance, and fewer parameters to learn

Pooling operation

- Max pooling

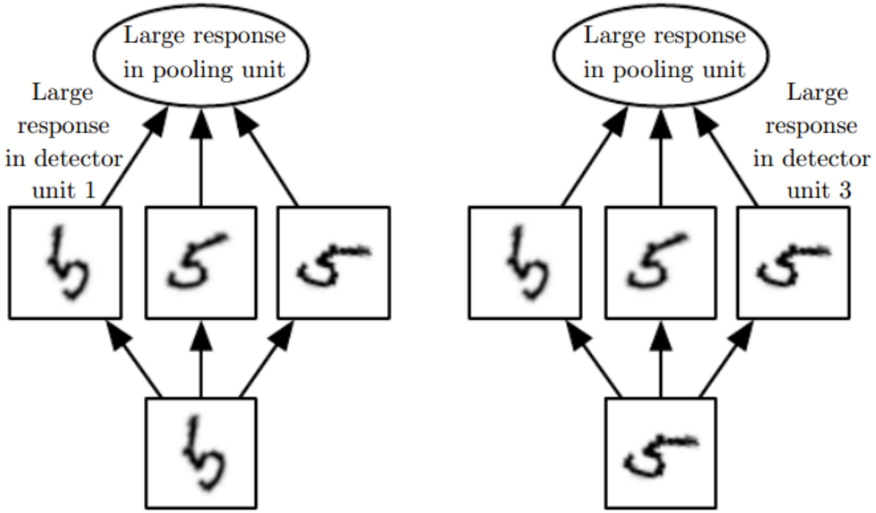


- Average pooling

❑ Reduce the number of parameters → makes the training faster and help reduce overfitting

❑ Introduce invariance to translation

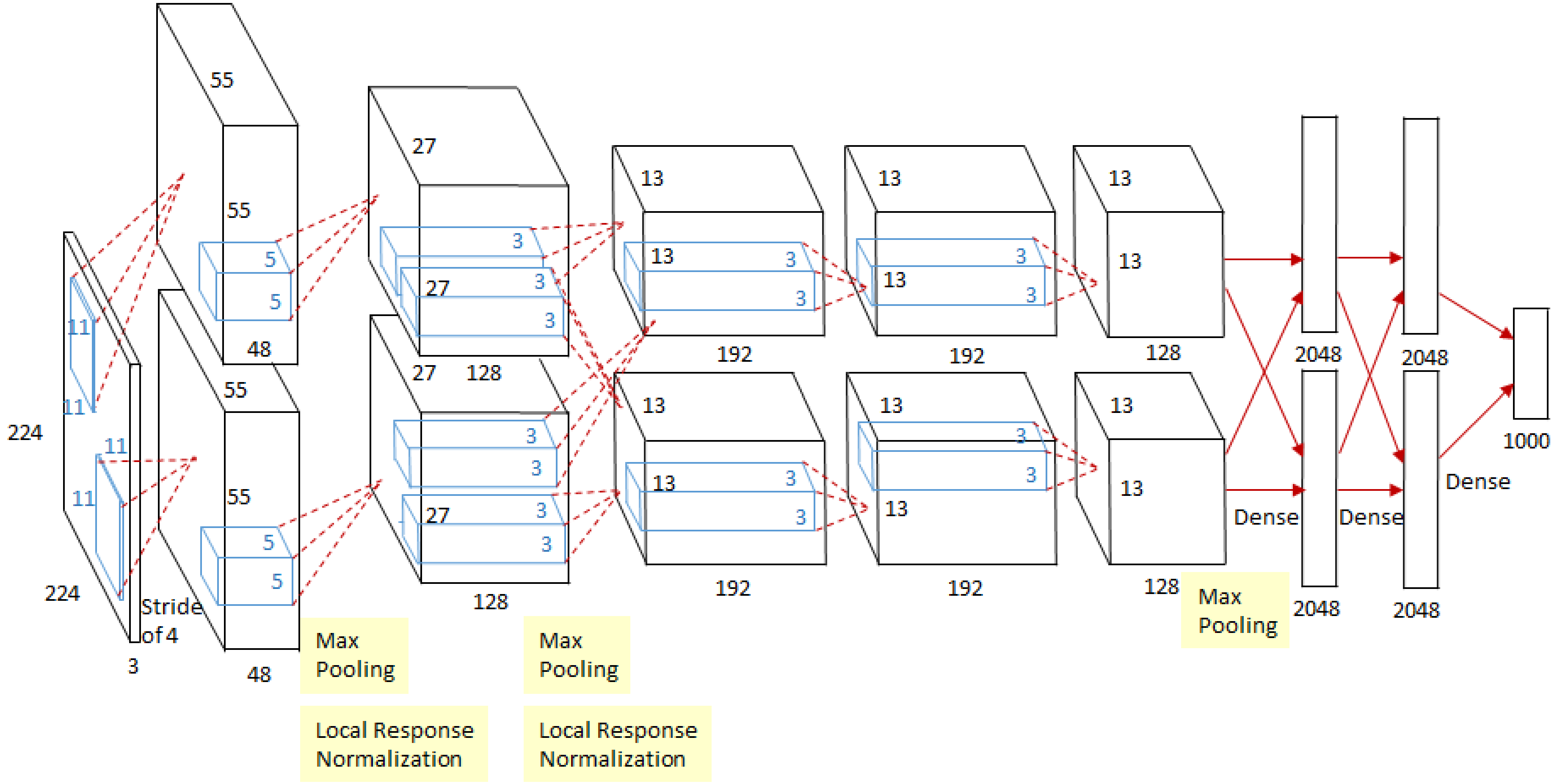
❑ May introduce other invariances, e.g. to rotation



Some popular CNN architectures

- AlexNet: historical **classification** on ImageNet

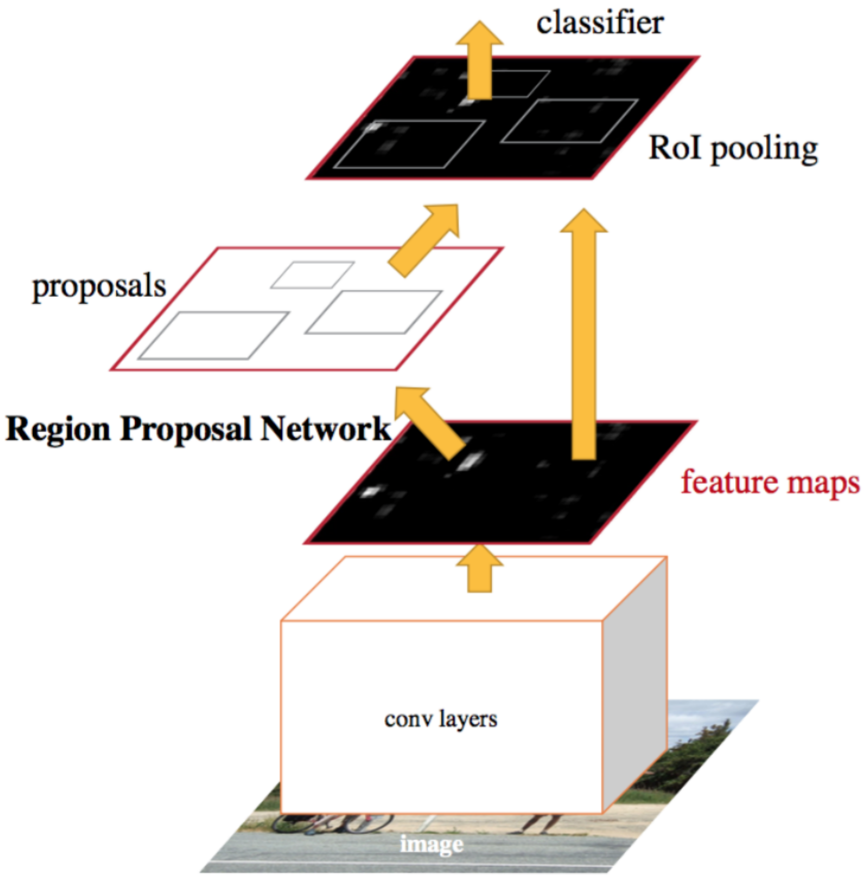
2 branches for 2 GPUs

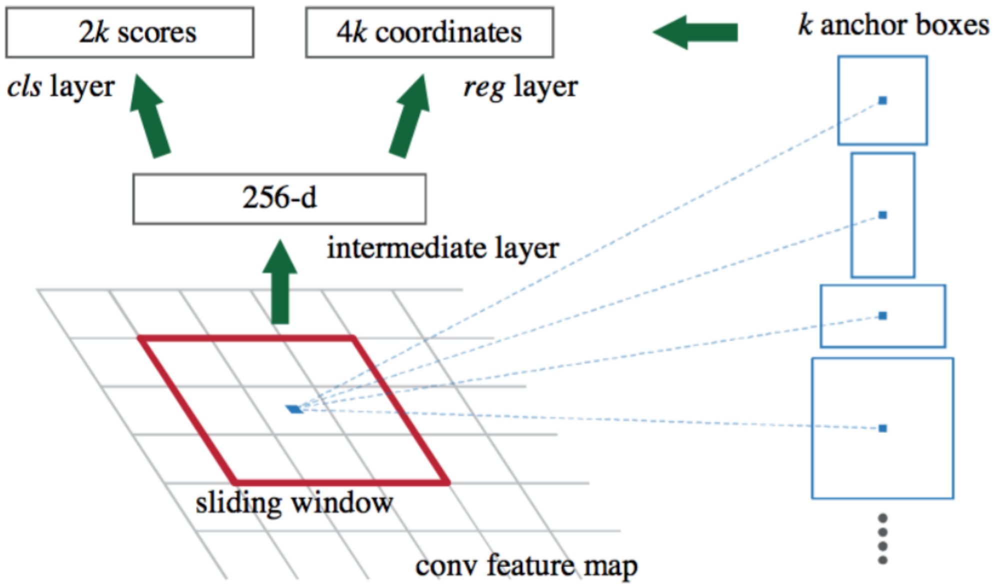


Some popular CNN architectures

- Faster-RCNN: **detection** task

The feature map is used for 2 tasks

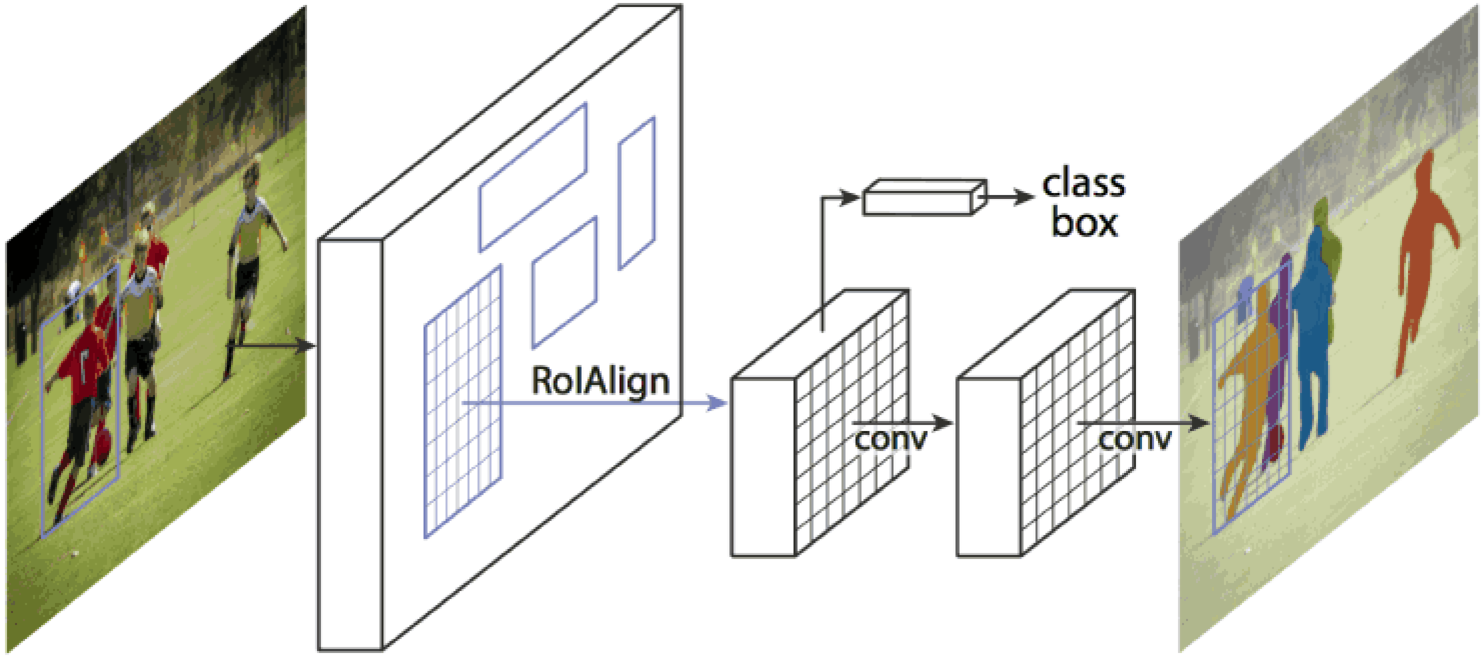




Some popular CNN architectures

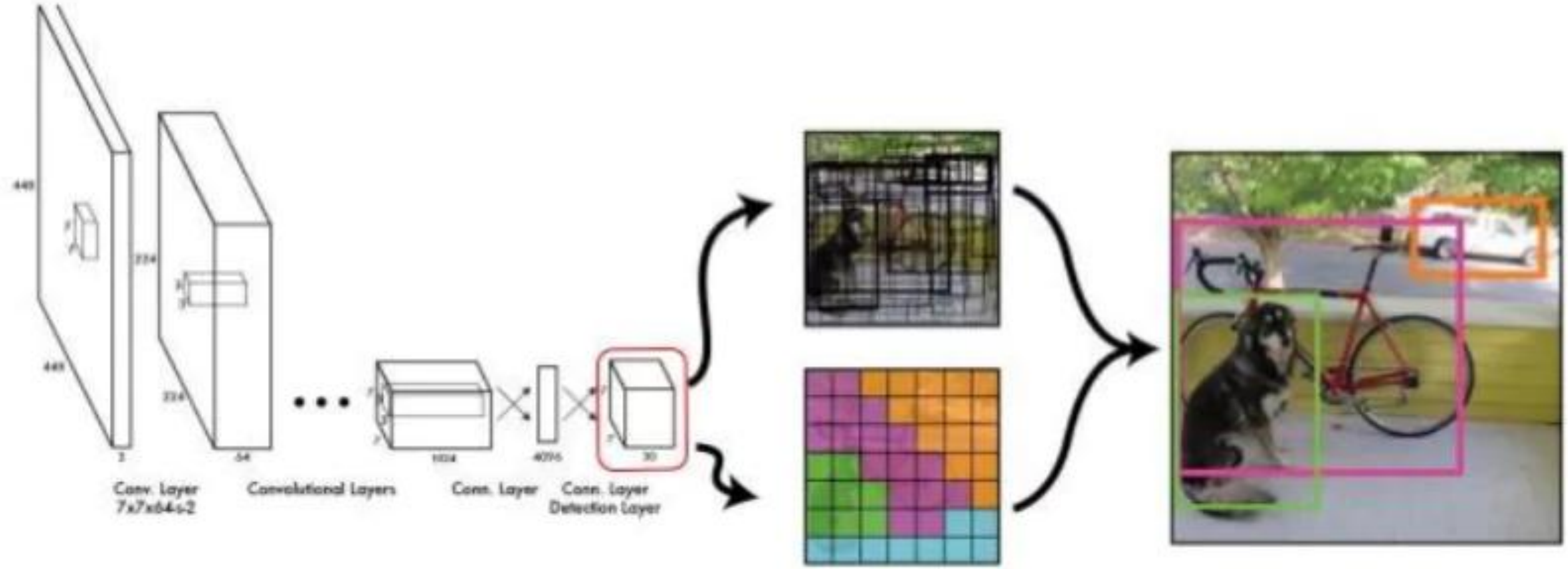
- Mask-RCNN: **detection** and **segmentation**

The feature map is used for 3 tasks



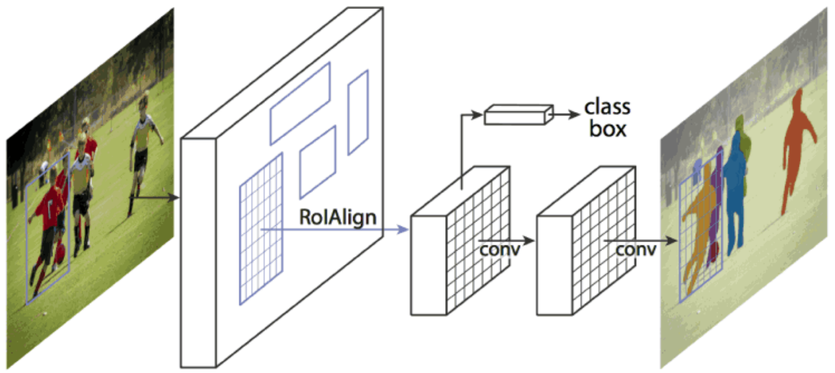
Some popular CNN architectures

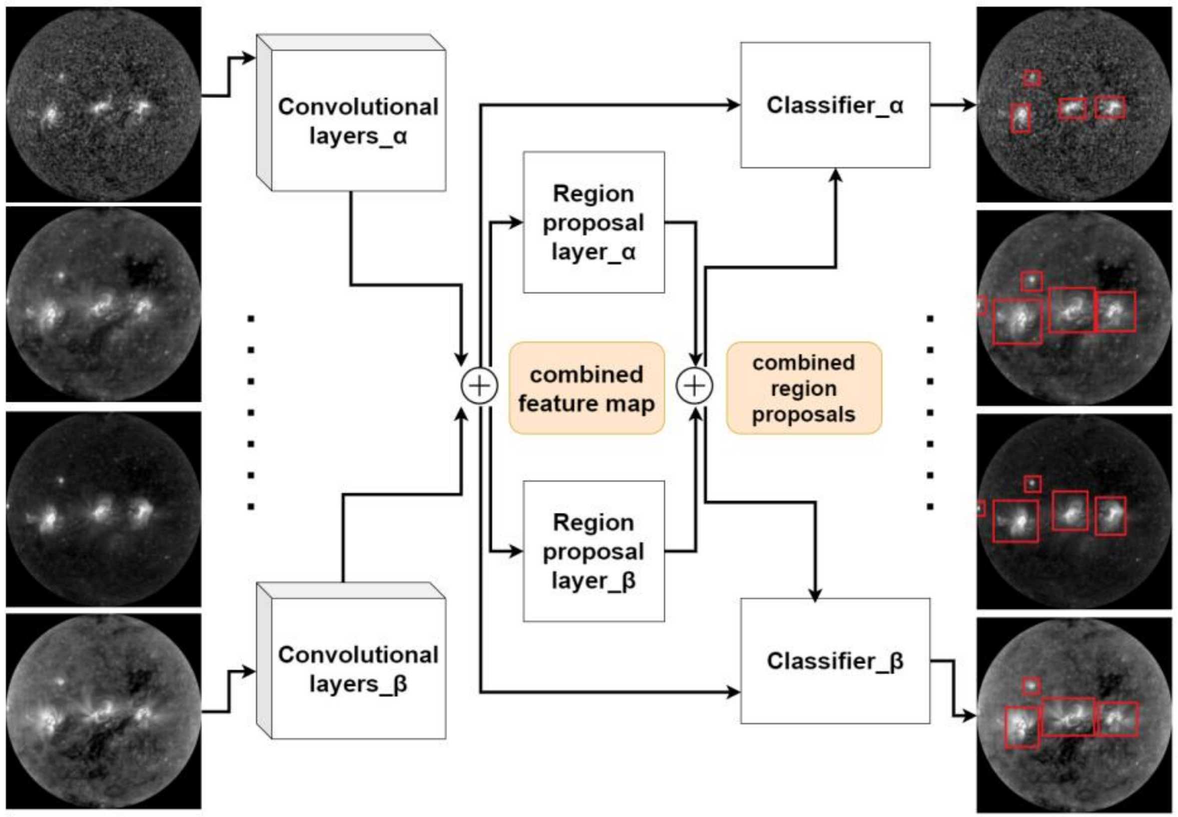
- YOLO: fast and lightweight **detection**



Multi-task neural networks

Shared feature map → features need to have more meaning





How should we interpret the predictions?

Classification layers: Predict the most likely **class**

The output is not necessarily a probability!

- Cross-entropy loss:

Computed on softmax outputs which are between 0 and 1 and are often interpreted as a pseudo-probability distribution

- Negative Log Likelihood loss:

Measures the accuracy if the model tries to output a probability for each class

- Cosine proximity:

Compares two vectors, e.g. $\begin{bmatrix} \sim 0 \\ \sim 1 \\ \sim 0 \end{bmatrix}$ and $\begin{bmatrix} \sim 1 \\ \sim 0 \\ \sim 0 \end{bmatrix}$

- Hinge loss (max margin):

Compares the signs of the output and true label

How much can we trust the model and its predictions?

Remember:

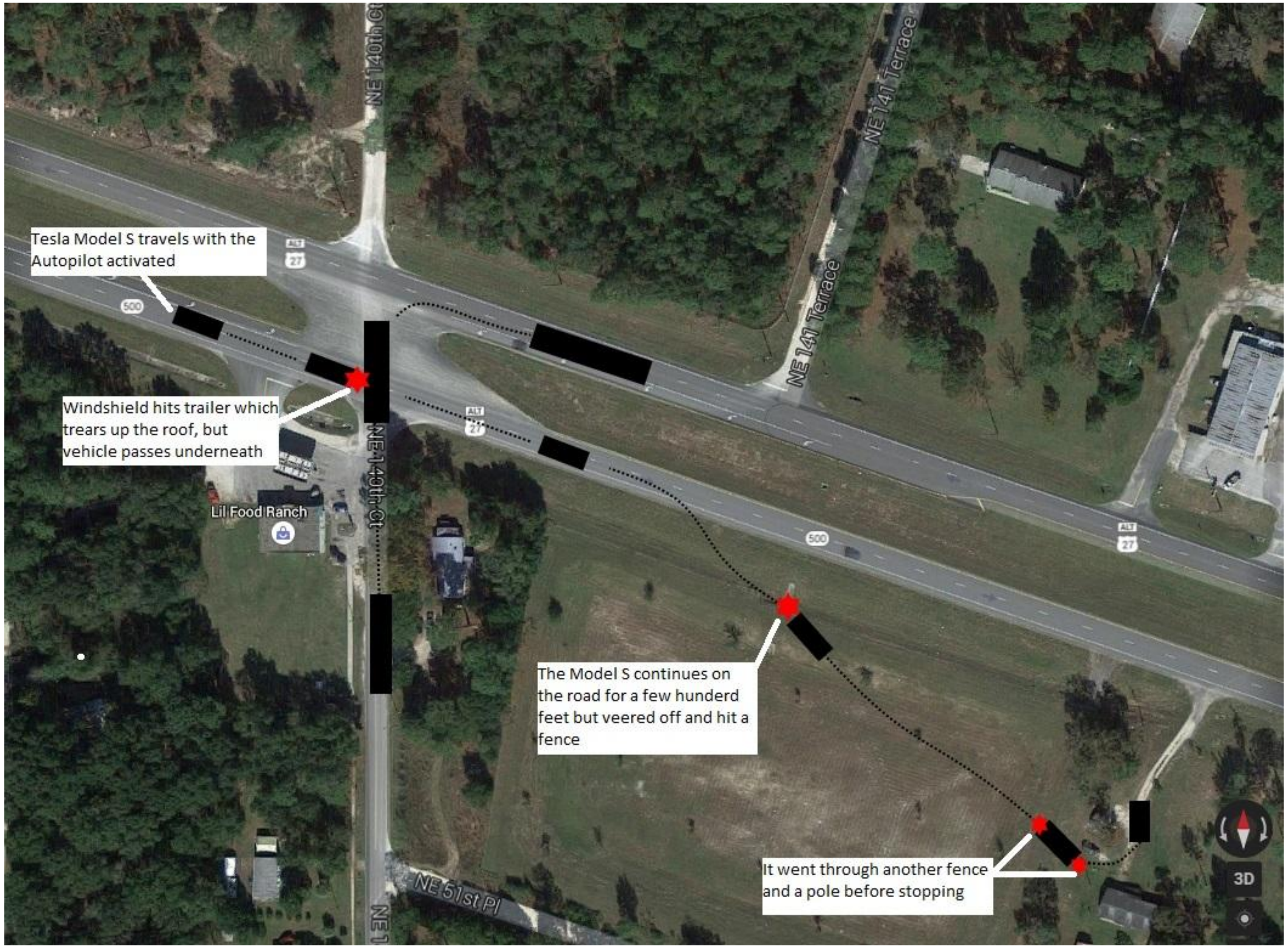
- We don’t have all the possible data in the world
- Data may be noisy, uncertain

→ Strong generalisation testing

→ and…?

How much can we trust these “end-to-end” trained black-box algorithms?

How much can we trust the model and its predictions?

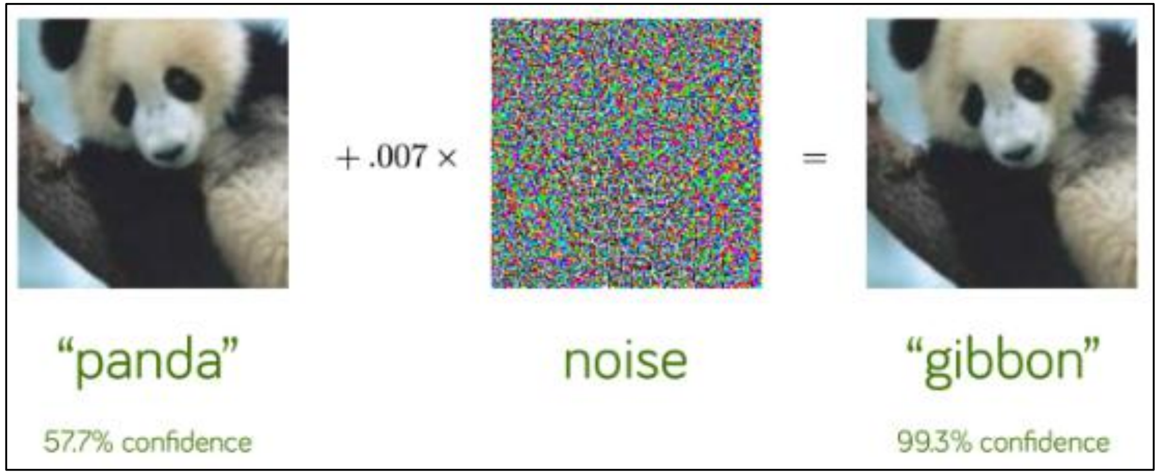


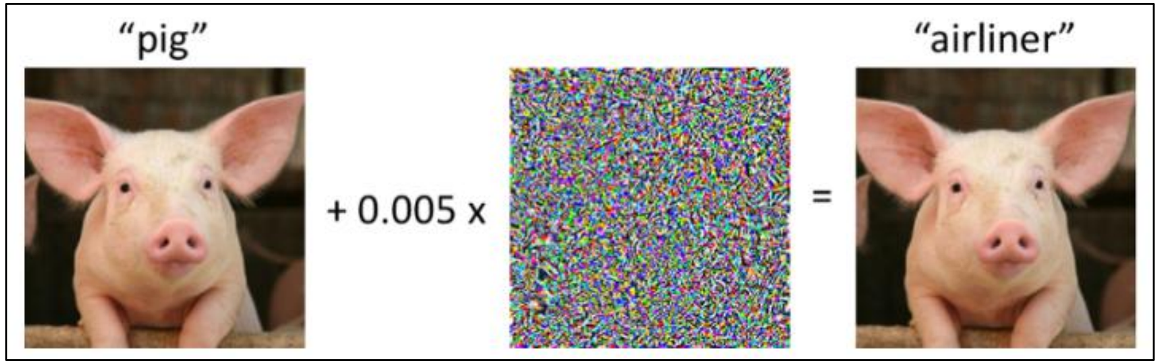
How much can we trust the model and its predictions?

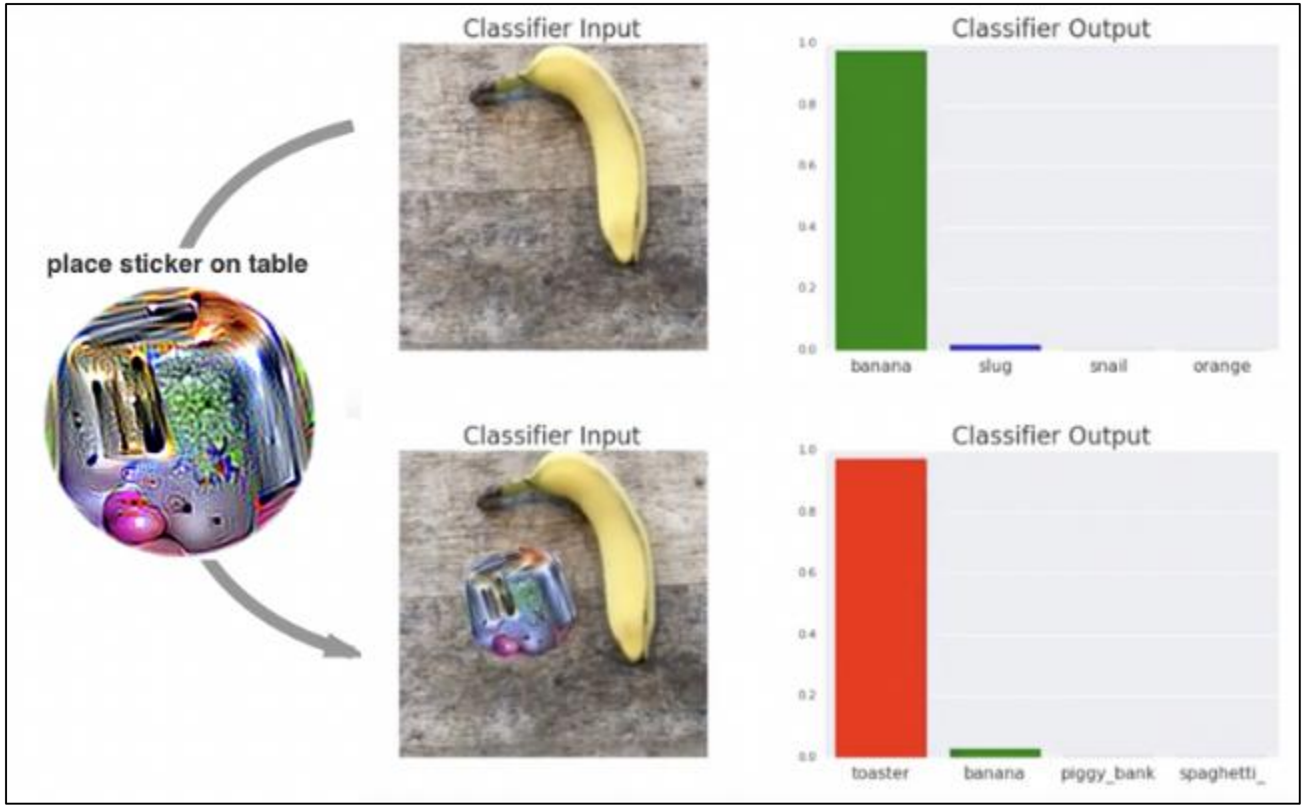
- Biased data → biased models!

How much can we trust the model and its predictions?

- Adversarial examples

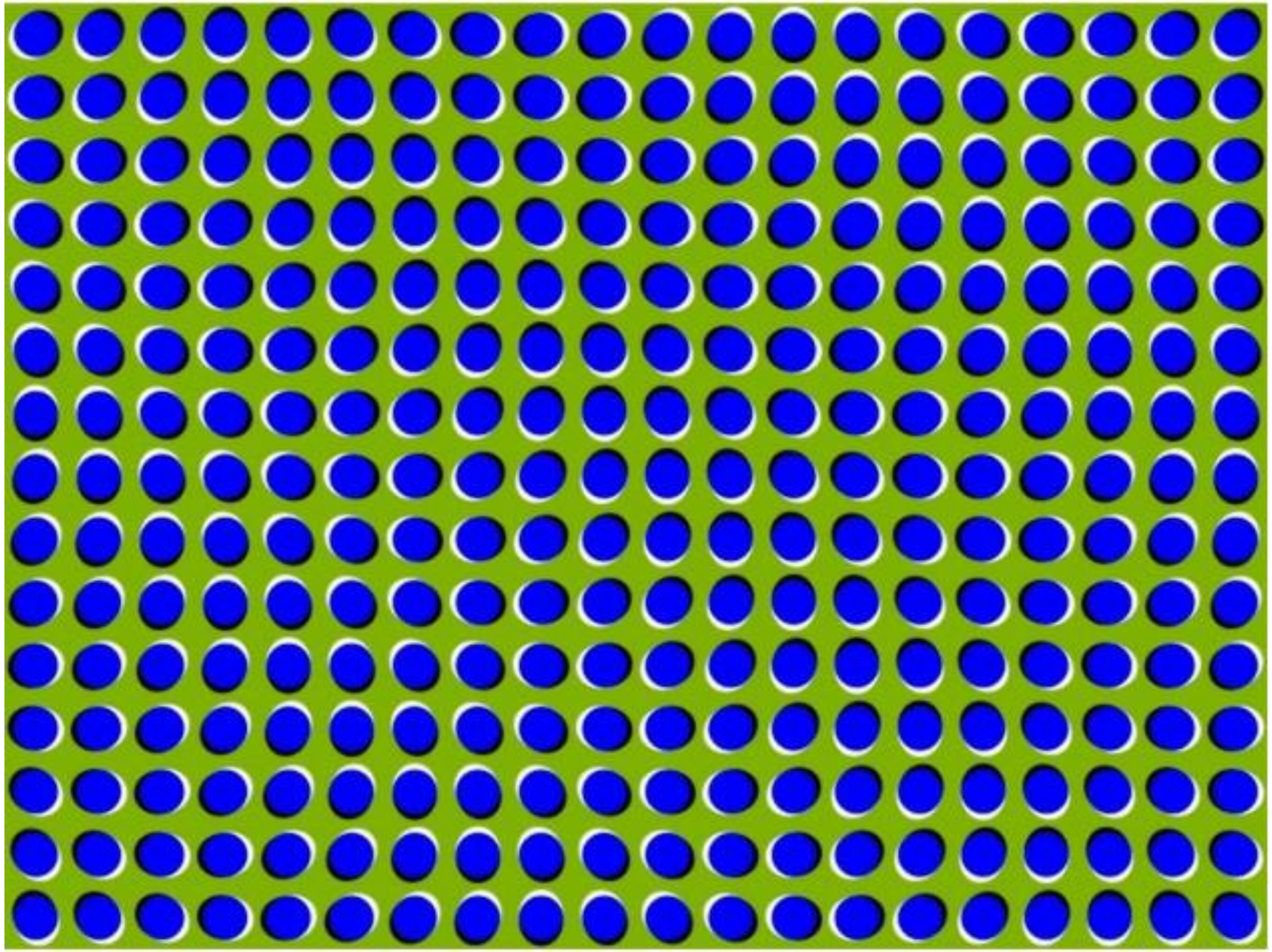






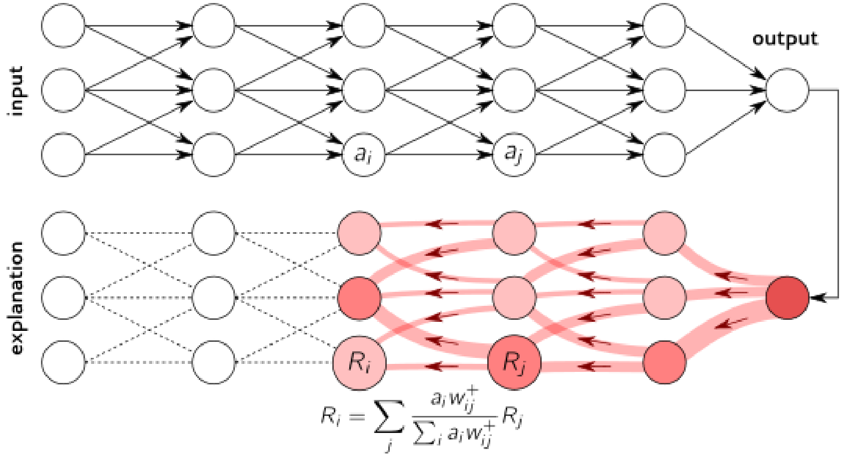
Adversarial examples

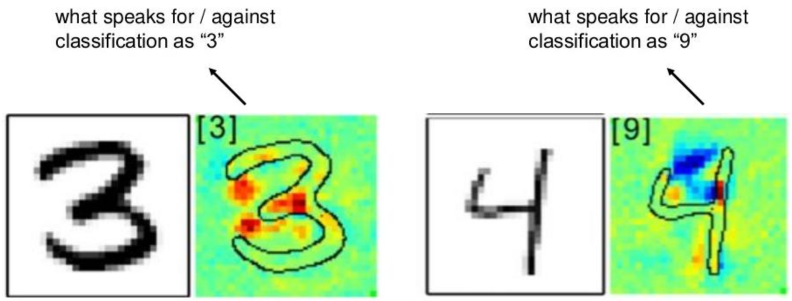
Like optical illusions for CNNs…



How do we know what neural networks actually do?

Visualising what activates the CNN's neurones: Layer-wise Relevance Propagation (LRP)





[number]: explanation target class

red color: evidence for prediction

blue color: evidence against prediction

(Binder et al., ICML Visualization Workshop, 2016)

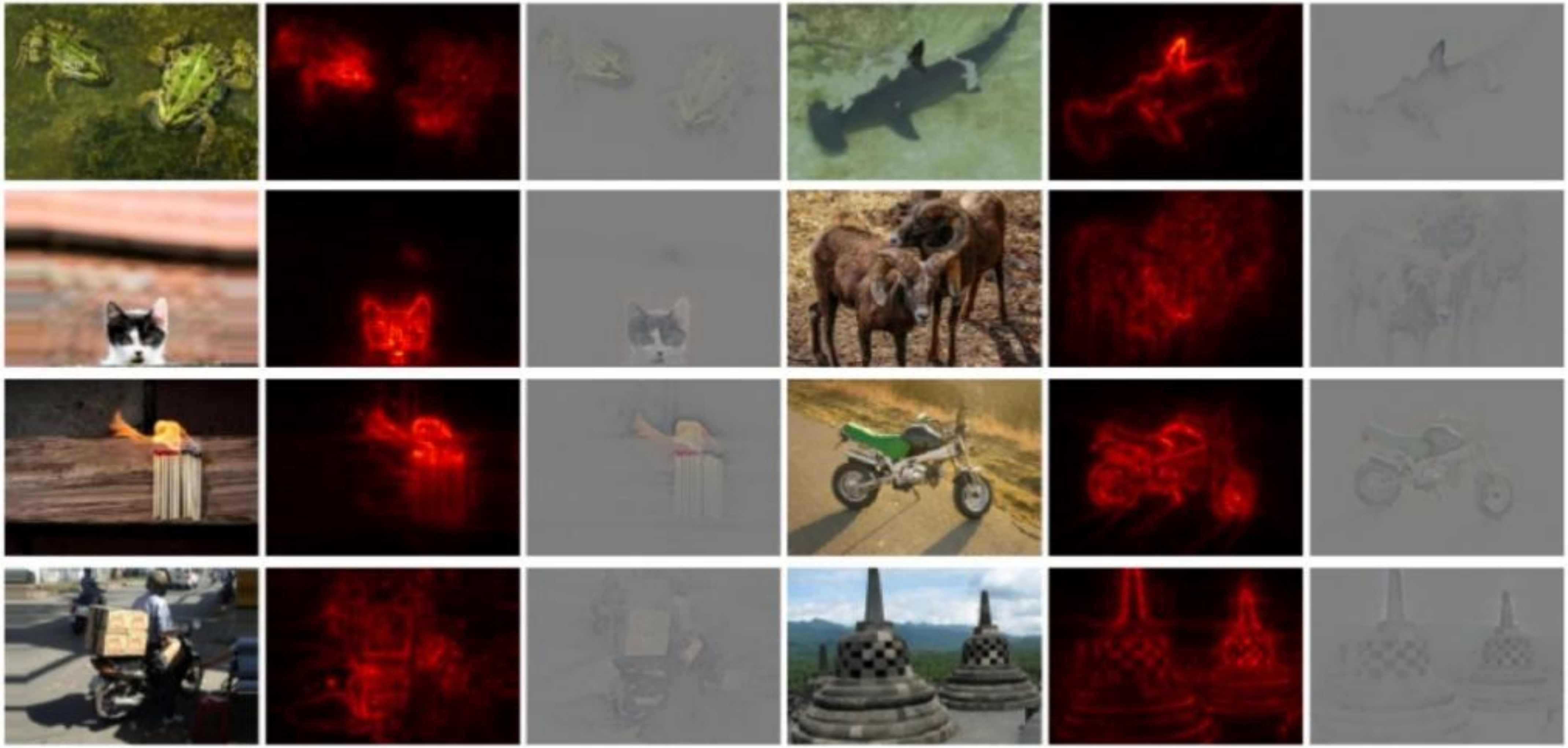
GoogleNet focuses on faces of animals

→ suppresses background noise, but doesn't use context

S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. **On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation**. PLOS ONE, 10(7), 2015
A. Binder, G. Montavon, S. Lapuschkin, K.-R. Müller, and W. Samek: **Layer-wise Relevance Propagation for Neural Networks with Local Renormalization Layers.** *ICANN 2016, 2016*

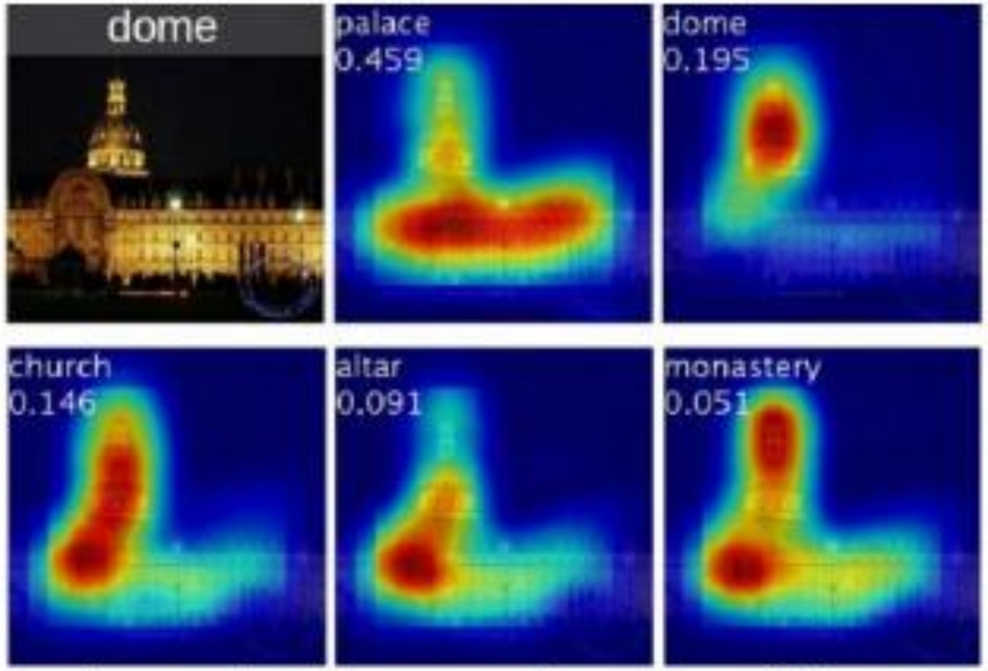
How do we know what neural networks actually do?

Visualising what activates the CNN’s neurones:



How do we know what neural networks actually do?

Another method: Class Activation Mapping (CAM) and numerous derivatives



Class activation maps of top 5 predictions

Class activation maps for one object class

Requires the use of Global Average Pooling (GAP)

➢ Not as general as LRP

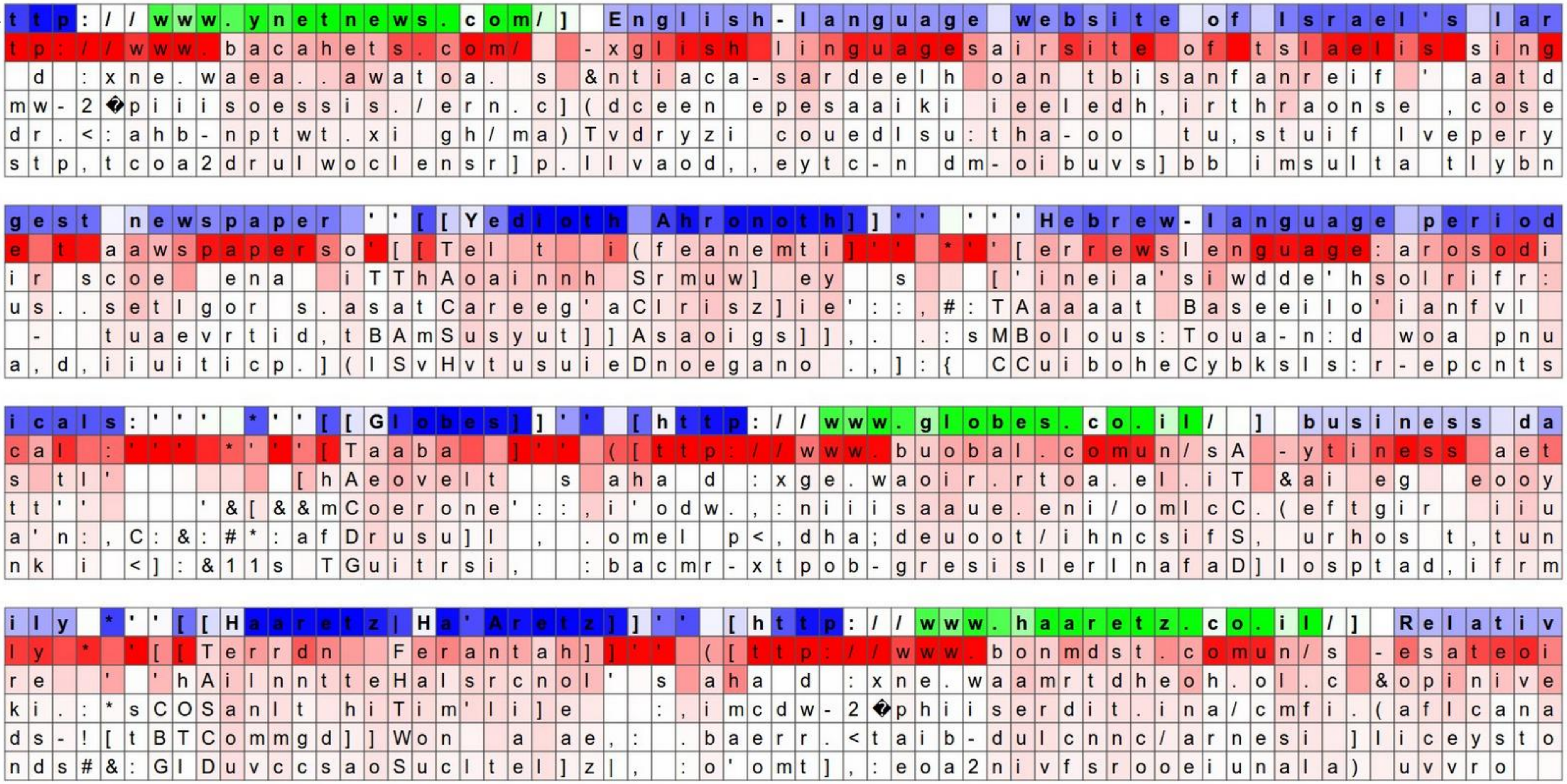
B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. **Learning Deep Features for Discriminative Localization**. CVPR'16 (arXiv:1512.04150, 2015).

How do we know what neural networks actually do?

Visualising RNN neurones' activations:

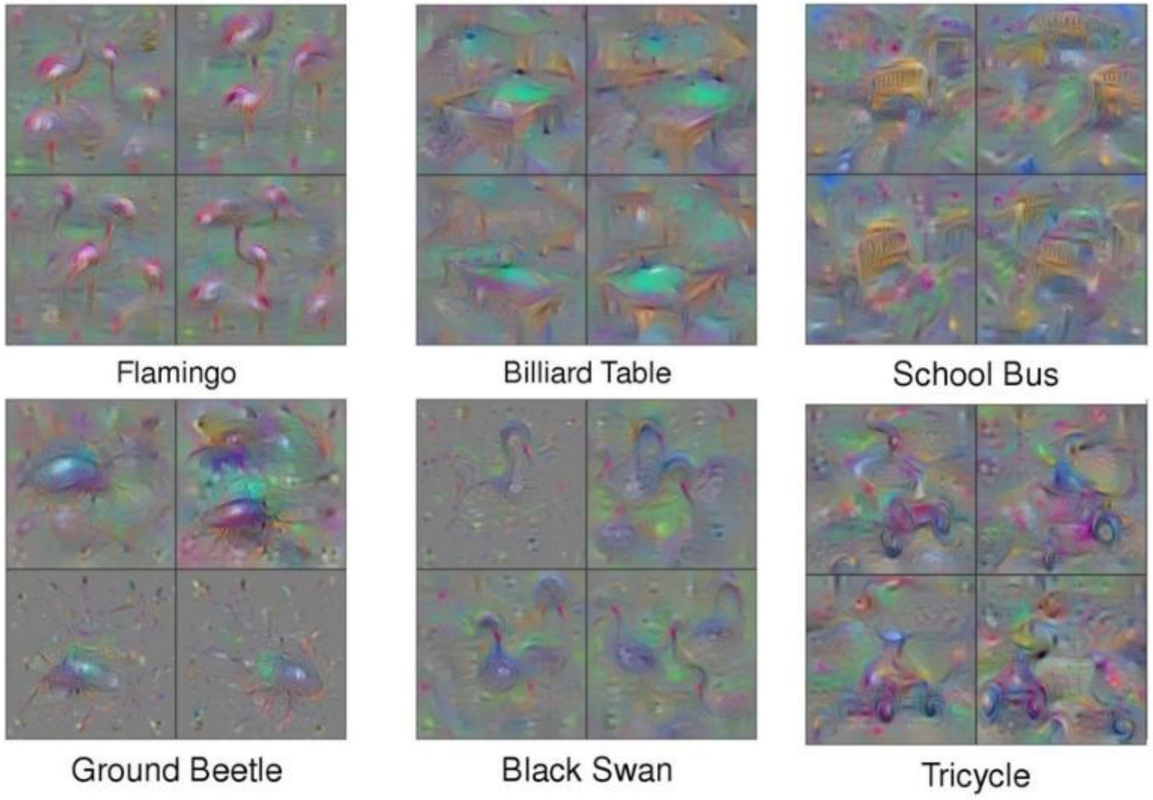
Activation of one neuron: blue → weak (-1), green → strong (+1)

Prediction: 5 most likely options for the next character



How do we know what neural networks actually do?

Ideal images that maximally activate a given CNN neurone for different classes:



K. Simonyan, A. Vedaldi, A. Zisserman: **Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Map**. ICLR Workshop 2014

A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, J. Clune: **Synthesizing the preferred inputs for neurons in neural networks via deep generator networks**. NIPS 2016

The next (foreseeable) big developments in AI…

- Explainable neural networks



- Physics inspired neural networks

- Hybrid data- and knowledge-driven models