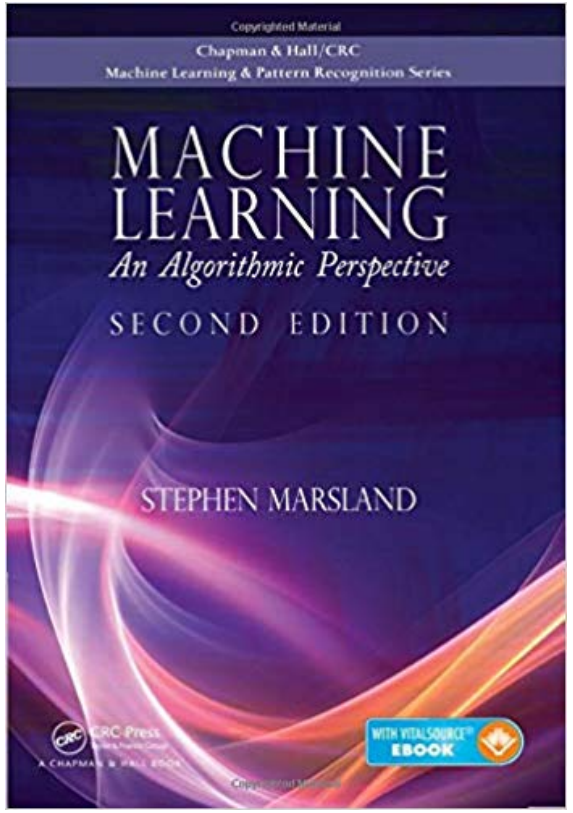
Introduction to Machine Learning and Artificial Neural Networks

David Cornu

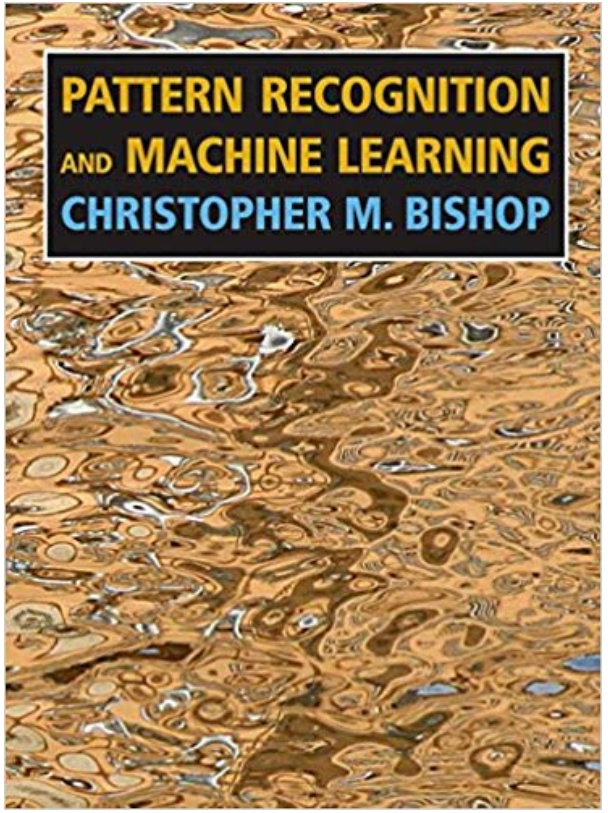
LUX, Observatoire de Paris, PSL

M2 OSAE 2025/2026

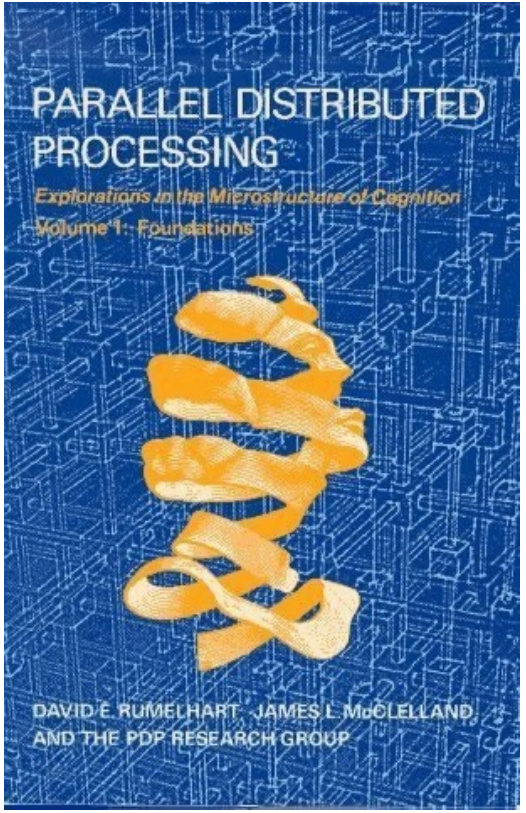
Some learning materials and references



Stephen Marsland.
Machine Learning an Algorithmic Perspective.
Chapman and Hall/Crc, 2015



Christopher M. Bishop.
Pattern Recognition and Machine Learning.
Springer, 2011



David E. Rumelhart et al.
Parallel Distributed Processing. MIT press, 1989

Some learning materials and references

- My PhD Thesis: https://arxiv.org/abs/2010.01431
  or its updated version: https://share.obspm.fr/s/X3xN6NxJbnZDBWX
  Chapters 2, 4 and 11 can be considered as a handout for this course

- **The Fidle** (Formation Introduction au Deep Learning) **course from the CNRS:**
  https://gricad-gitlab.univ-grenoble-alpes.fr/talks/fidle/

- The Neural Network CS231n course from Stanford:
  https://cs231n.github.io/

- Some more references on a dedicated page on the MINERVA website
  https://minerva-astro.obspm.fr/learning/

Take everything you read about ML and AI with a grain of salt! (This course included)

Lots of materials still contain practical or theoretical approximations and errors.
Crossing references is most of the time necessary.

1) What is Machine Learning ?

Any method that extracts statistical information about a dataset through a learning process.
Another appropriate name is « **statistical learning** » .
But how to define learning ? (still a relatively open question depending on the research field)

And what is **Artificial Intelligence** then ? Tricky and subjective question …
→ Most of the time ML is considered as a sub-field of a larger AI domain.

2) Why is it interesting ?

Learns directly from data:
- Can learn unknown relations that are too difficult to formulate analytically
- Can work efficiently in high dimensional parameter spaces and for large volumes of data
- Can approximate very complex relations (methods that are Universal Function Approximators)

3) For which applications ?

Almost anything you can think of !
BUT it is limited by the data availability and computing power.

Are ML methods better for any application ? Not at all !
Warning: AI is often used as a selling argument, while it is not automatically better than classical data analysis

**in a 2019 report from the London venture capital firm MMC, 40% of « AI startups » don't really use AI or ML*

Attempt of learning definition

The online Cambridge dictionary:

« The process of getting an understanding of something by experience »

The *Tresor de la langue Française informatisé:*

« Acquérir la connaissance d'une chose par l'exercice de l'intelligence, de la mémoire, des mécanismes gestuels, appropriés, etc. »

« Acquire the knowledge of something through the practice of intelligence, memory, appropriate gestures, etc. »

But theses definitions are biased toward a human definition of the learning process...

Attempt of learning definition

In a more generic way, **animal learning** is usually summarized by these capabilities:

- Sensitivity **- Memory** **- Adaptation** **- Generalization**

By adopting the proposed **statistical learning** definition, any algorithm that works using the following elements can be considered as **Machine Learning**:

- An adaptable input construction, which represents the sensitivity of the method.
- A form of memory to remember either the previous situations, or a reduced version of them.
- A way to estimate if its behavior (output prediction) is appropriate.
- A way to adapt its behavior during the learning process.
- A way to generalize its behavior to new outputs.

Keywords to explore the notion of Artificial Intelligence from there (beyond the scope of this course) :

- Symbolic and sub-symbolic representations - Artificial General Intelligence - Strong AI

Types of Machine Learning

Supervised

Learn from labeled examples.
Try to find a generalization to get correct prediction most of the time.

Semi-Supervised

Usually a mix of two algorithms.
A un-supervised one first that minimize the definition biases.
A supervised one that remap to the expected groups.

Un-Supervised

Dataset with examples is provided but with no labels.
Try to find similarities in the input in order to create groups.

Self-supervised

Dataset with no external labels.
Use the data themselves to define labels and learn using a supervised formalism.
(e.g., auto-encoders)

Reinforced

Based on an reward function.
Used when there is no easy error function. Corrections are applied based on an external performance measurement.

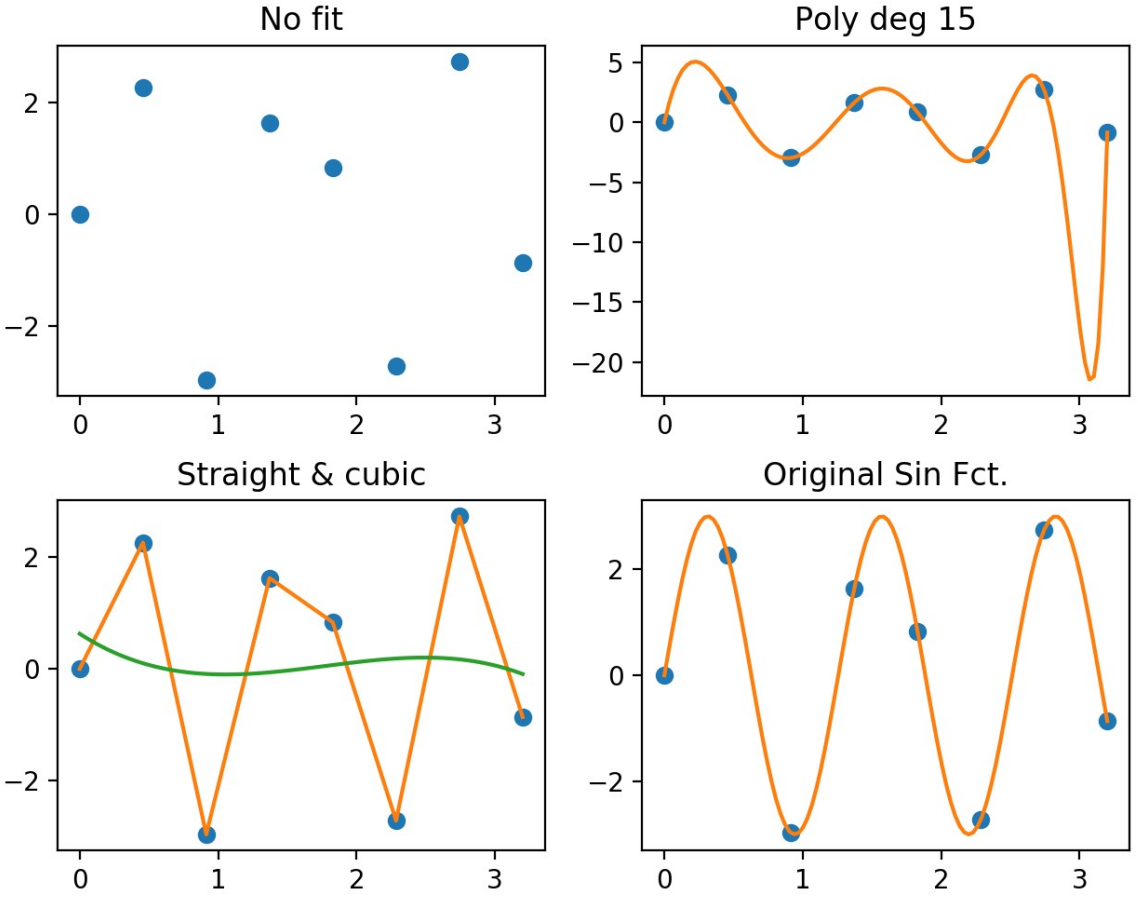
Evolutionary

Similar to reinforced, explore the possibility space based on biological evolution processes.
Selection effects, reproduction, mutation, etc.

Use cases

Simple example of regression

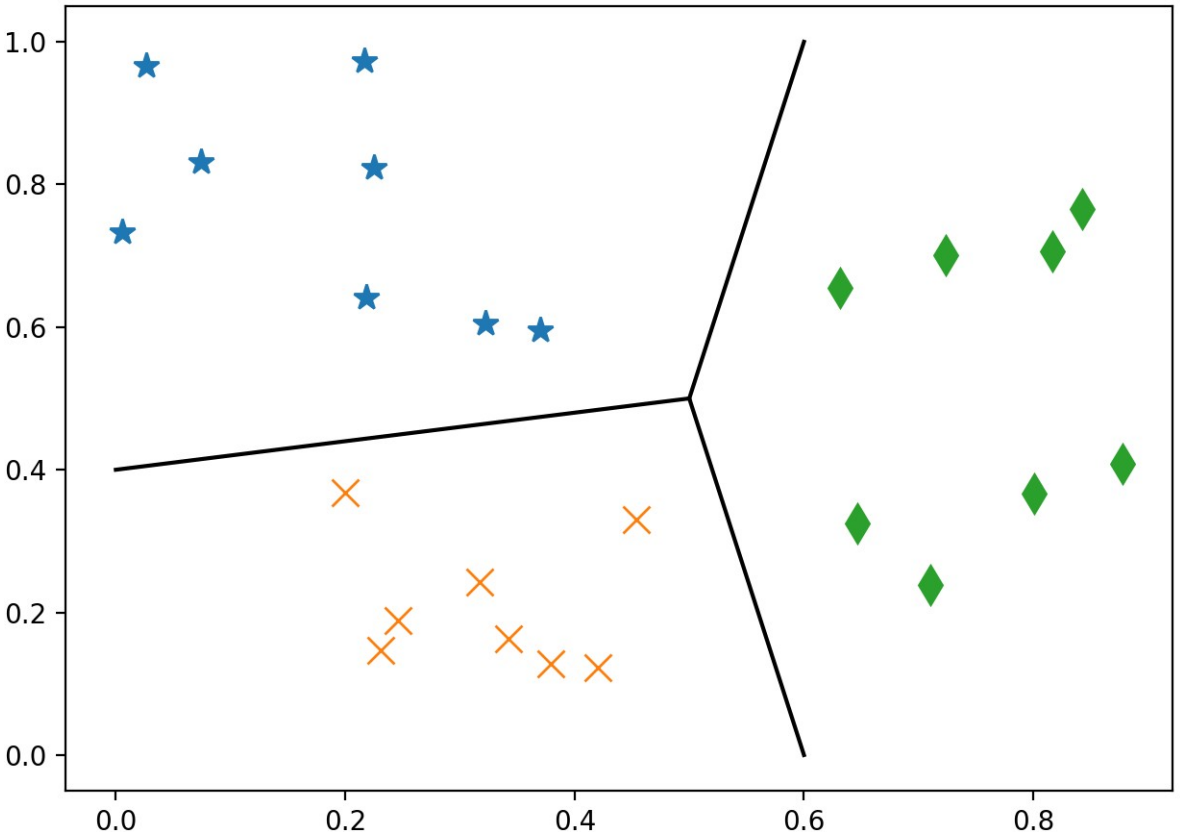
Reconstruct functions from a sample of data points (in any number of dimensions)



Use cases

Simple example of classification

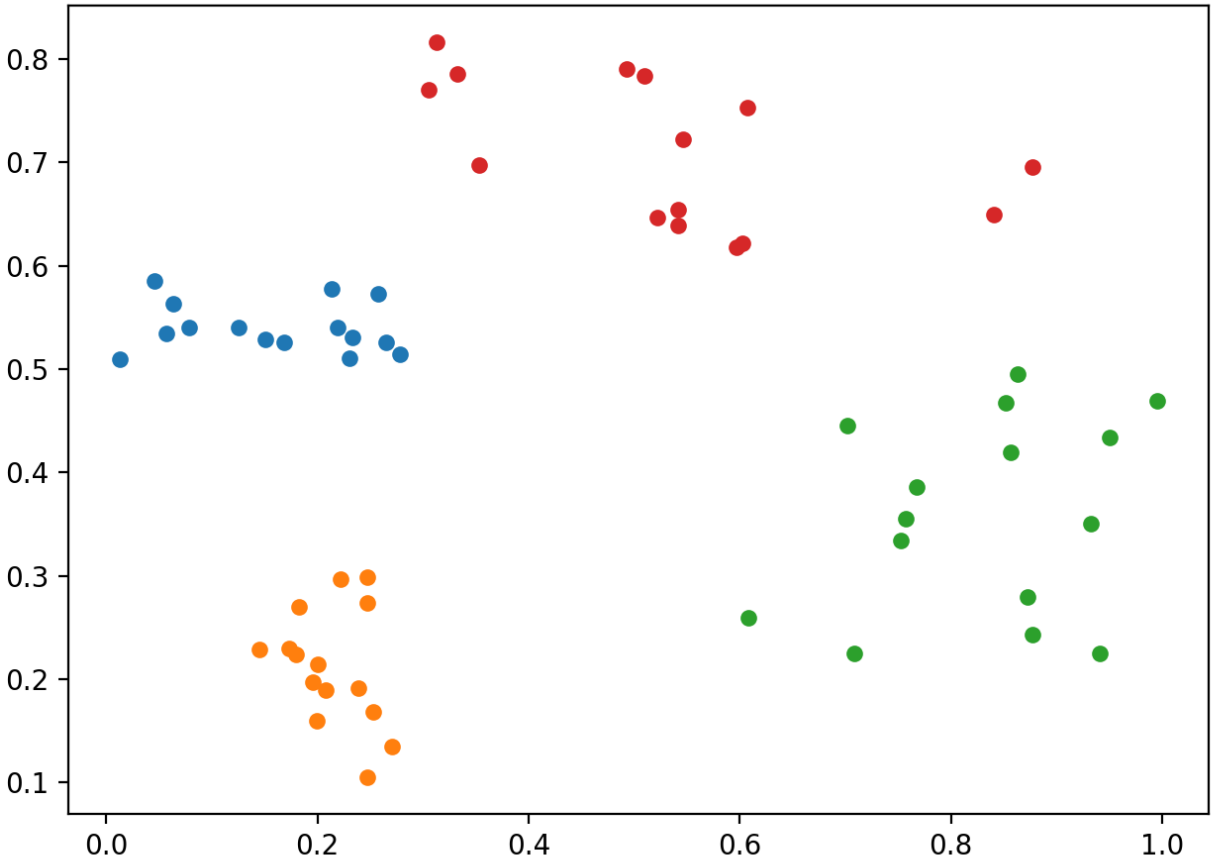
Decide from an input vector with several dimensions to which class it should be associated.

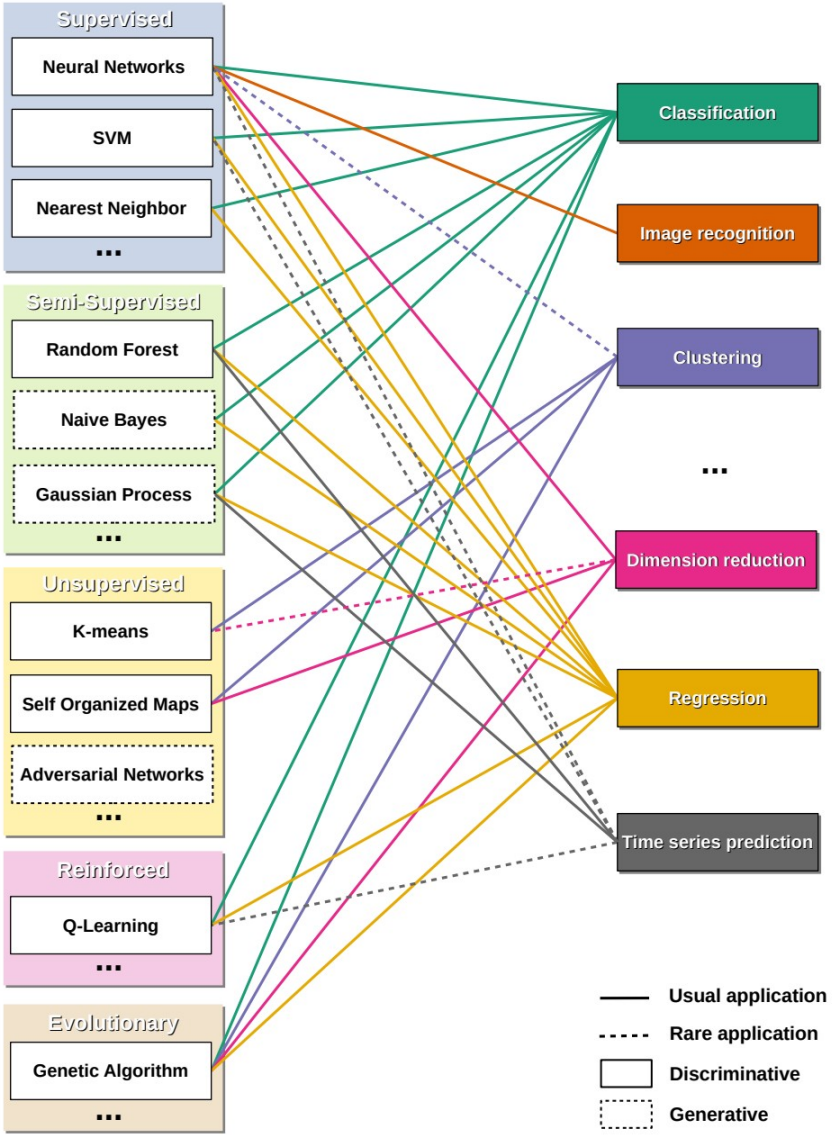


Use cases

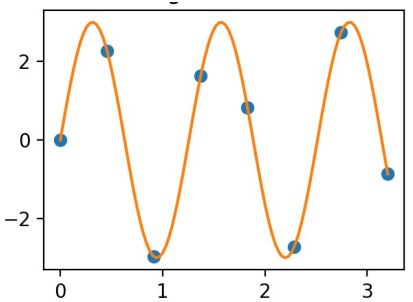
Simple example of clustering

Find groups in data based on a predefined strategy (e.g geometric distance) without prior knowledge on the number of groups, nor the solution for any example.

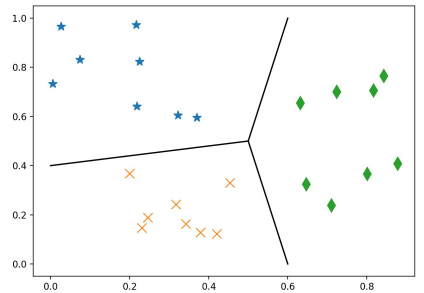




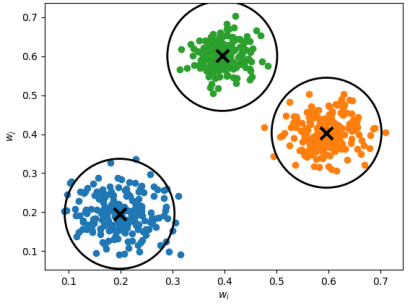
A wide variety of applications ...



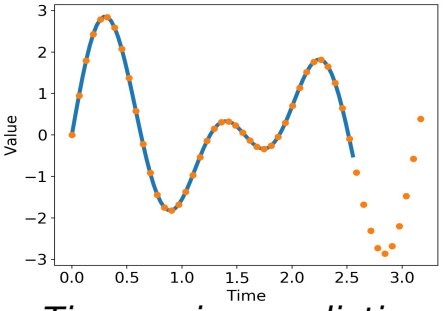
Regression



Classification



Clustering



Time series prediction

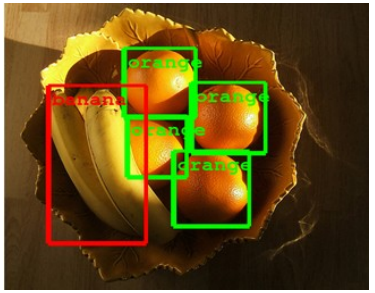
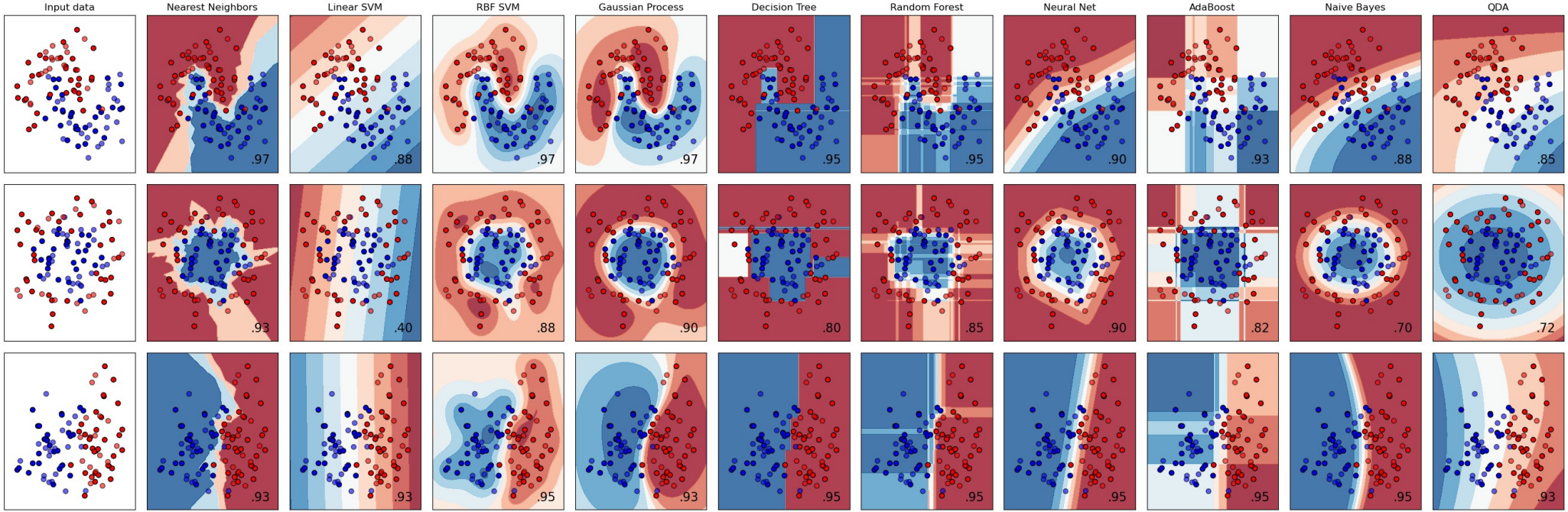


Image recognition

And much more ...

Methods comparison regarding data distribution



Some methods are more suited for specific data distributions, while others work well enough for most data distributions you might encounter.

In any case, every method has its weaknesses that you must identify and understand.

Choosing a method is selecting a balance between:

- The method capability on your data
- The computational efficiency of the method
- The required time of development / integration
- Your own proficiency with the various methods
- ...

How to use Machine Learning ?

Software

Nowadays, there are dedicated ML frameworks for almost every programming language.

They all have specific advantages, but any of them is enough to start learning and using ML.

Hardware

ML methods require a lot of data and computing power during the **training phase**.

However, these methods are very compute efficient when making **predictions**, meaning that it is often possible to deploy already trained models on light hardware (laptops or even modern smartphones)

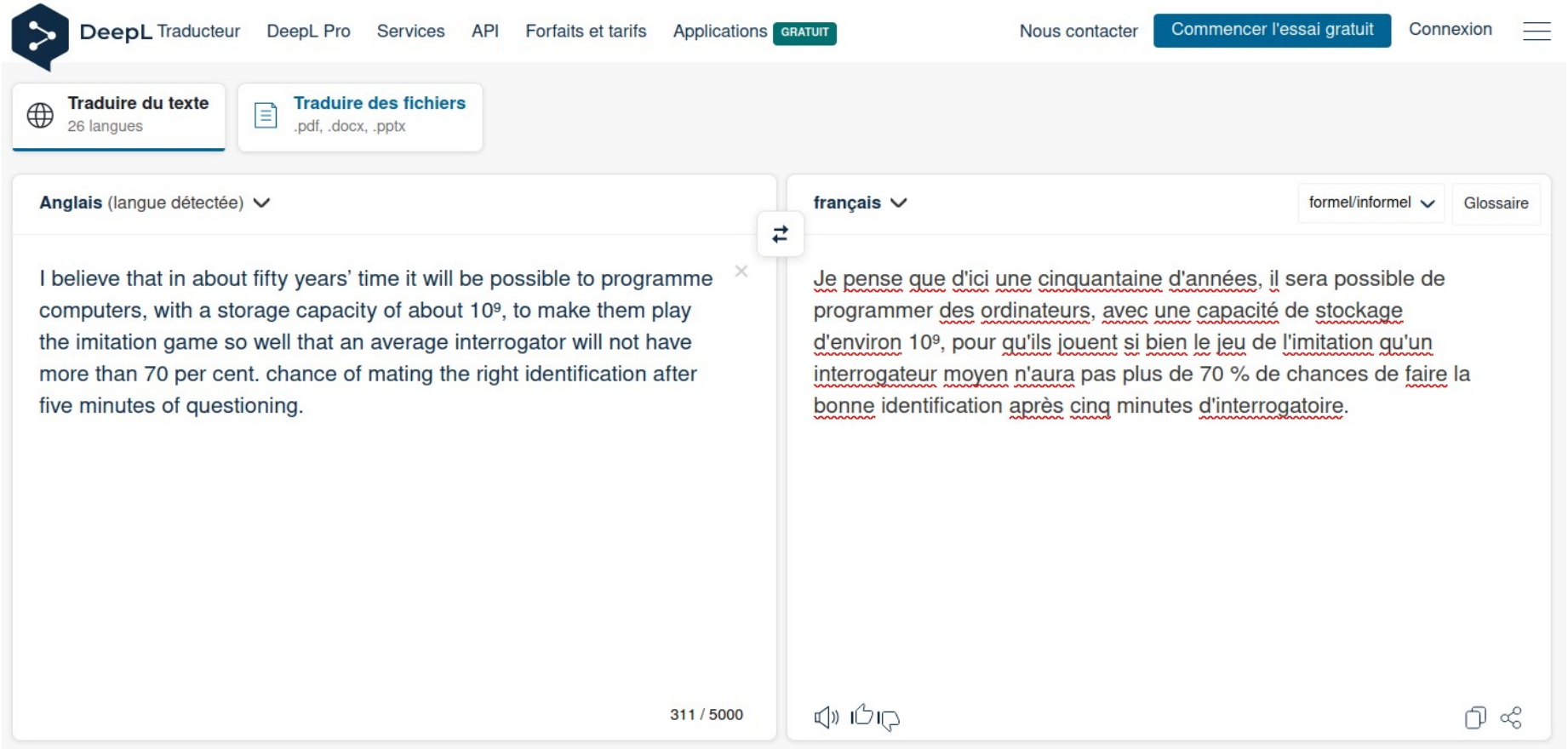
Still, light resources are usually enough to explore relatively simple ML methods, and many complex problems can be expressed in a simple form, allowing preliminary studies on low-end hardware resources to be conducted.

When training powerful models that work on images, it is helpful to use GPU hardware that is very efficient for most modern ML approaches.

Note that **hardware dedicated to ML methods** are more and more present and accessible.

(Tensor Cores in CPUs and GPUs, Neural Processing Units (NPU), etc ...)

Use cases
Examples of modern applications



**Private Convolutional Neural Network implementation, trained on the linguee database.*

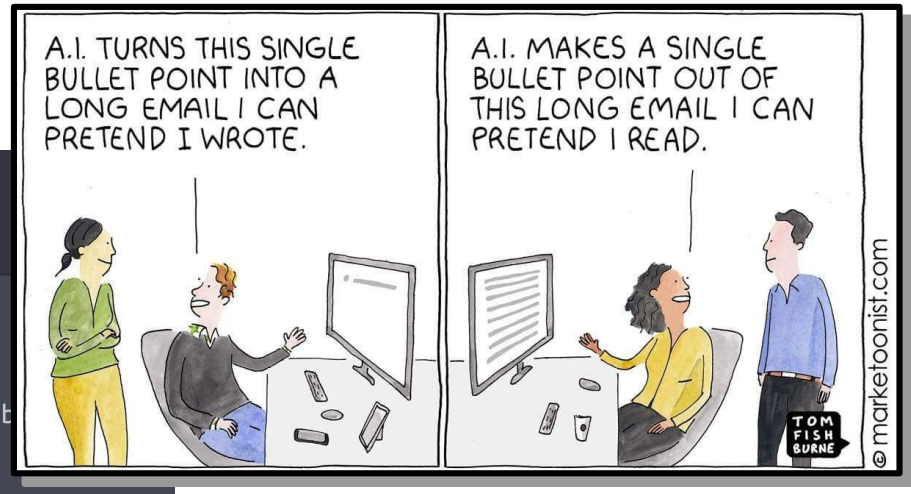
Use cases
Examples of modern applications

TH

Got any creative ideas for a 10 year old's birthday?

Sure! Here are some creative birthday ideas for a 10-year-old:

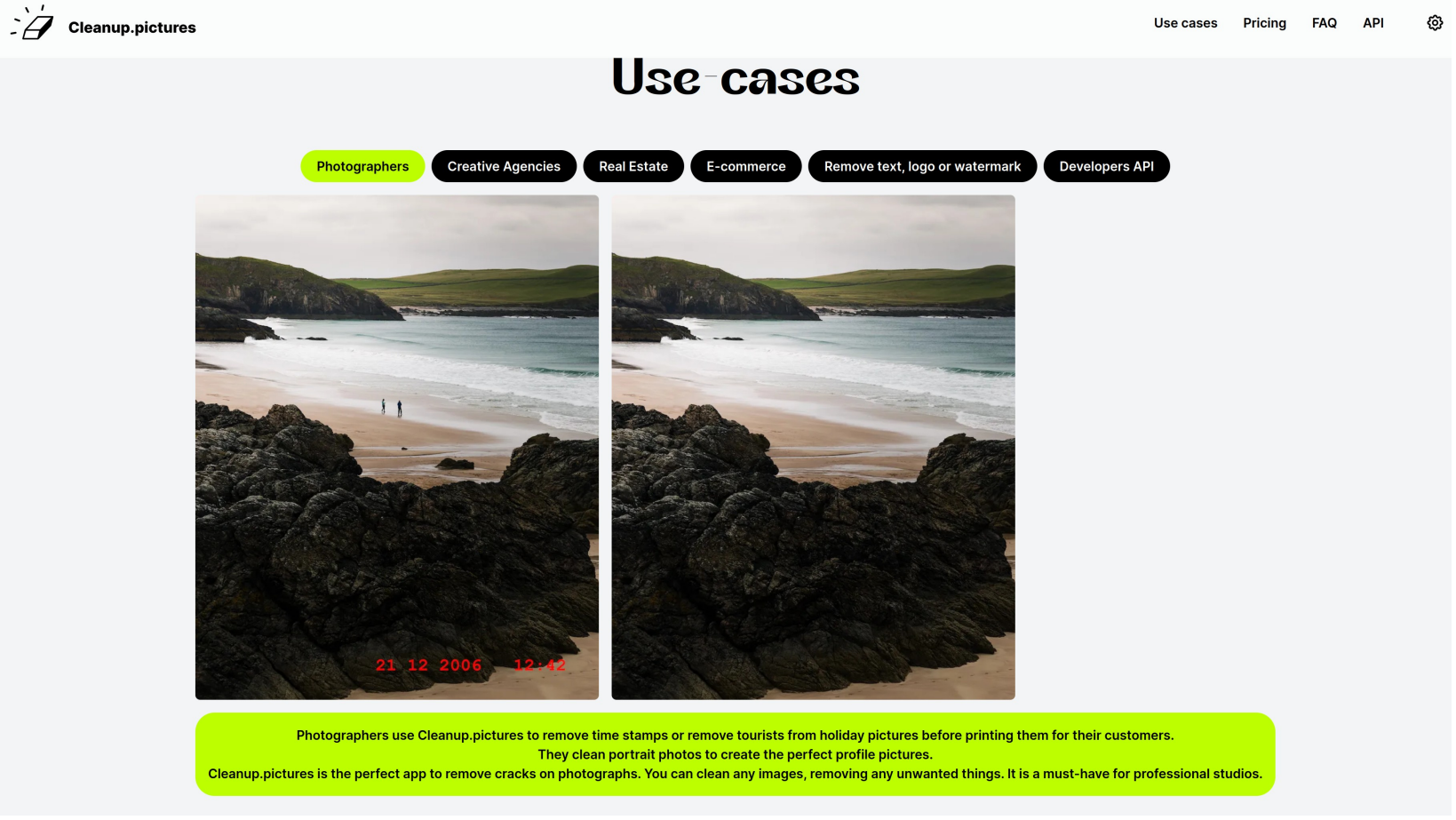
1. A themed birthday party such as a movie or video game character or a favorite hobb
2. A scavenger hunt with friends, either indoors or outdoors
3. A craft or DIY project, like creating tie-dye shirts or friendship bracelets
4. An outing to an amusement park, trampoline park, or laser tag center
5. A baking or cooking class, where the birthday child and friends can make and enjoy treats together
6. A sleepover with friends, complete with games, movies, and a special breakfast
7. A science or art experiment-themed party, where children can participate in hands-on activities
8. A sporty day with a visit to a rock climbing gym, batting cage, or mini golf course
9. A virtual escape room or game experience, played together with friends over video chat
10. A wildlife or nature exploration outing, such as a hike or a trip to a local zoo or aquarium.



Modern AI LLMs generated text is now quite convincing for human reader, but it remains detectable with automated tools based on ... AI LLMs (adversarial attack)!

**ChatGPT, based on GPT3 (Generative Pre-trained Transformer) => https://arxiv.org/abs/2005.14165*

Use cases
Examples of modern applications



**Based on the LaMa: Resolution-robust Large Mask Inpainting with Fourier Convolutions method → https://arxiv.org/abs/2109.07161 and for fun https://cleanup.pictures*

Use cases

Examples of modern applications

These persons do not exist !

**Based on the Style Generative Adversarial Network model*
→ *https://arxiv.org/abs/1812.04948 and for fun https://thispersondoesnotexist.com/*

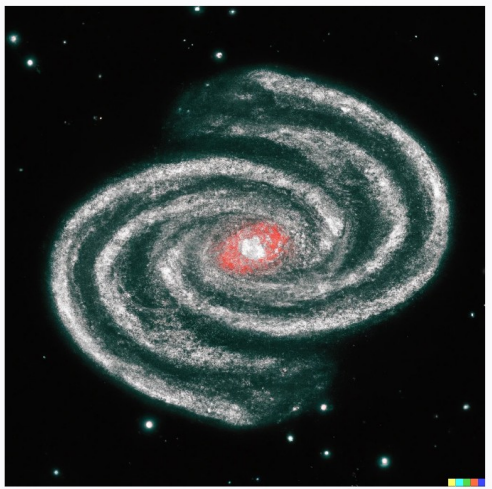
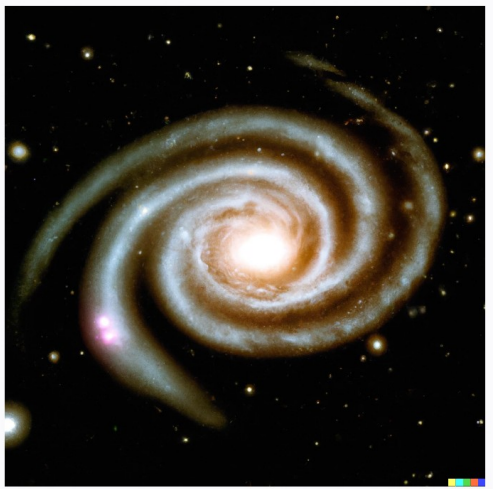
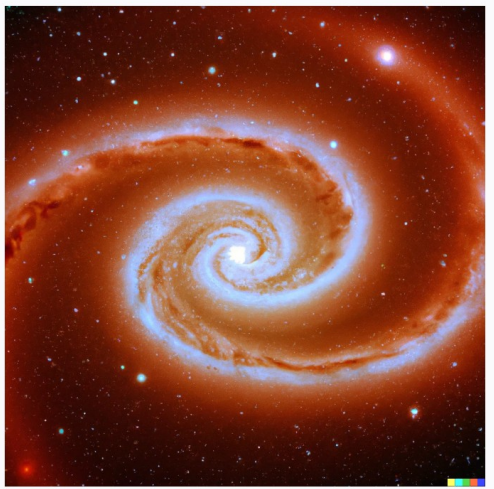
Use cases
Examples of modern applications

Edit the detailed description

Surprise me | Upload

An artistic view of a barred spiral galaxy observed in the infrared

Generate

**Dall-E, based on CLIP word embending and diffusion decodeur => https://arxiv.org/abs/2204.06125*

Use cases
Examples of modern applications

Generate an image of "An artistic view of a barred spiral galaxy observed in the infrared"

Here are the artistic views of a barred spiral galaxy observed in the infrared. The images highlight the central bar structure, swirling spiral arms, and vibrant infrared colors with distant stars and cosmic dust in the background.

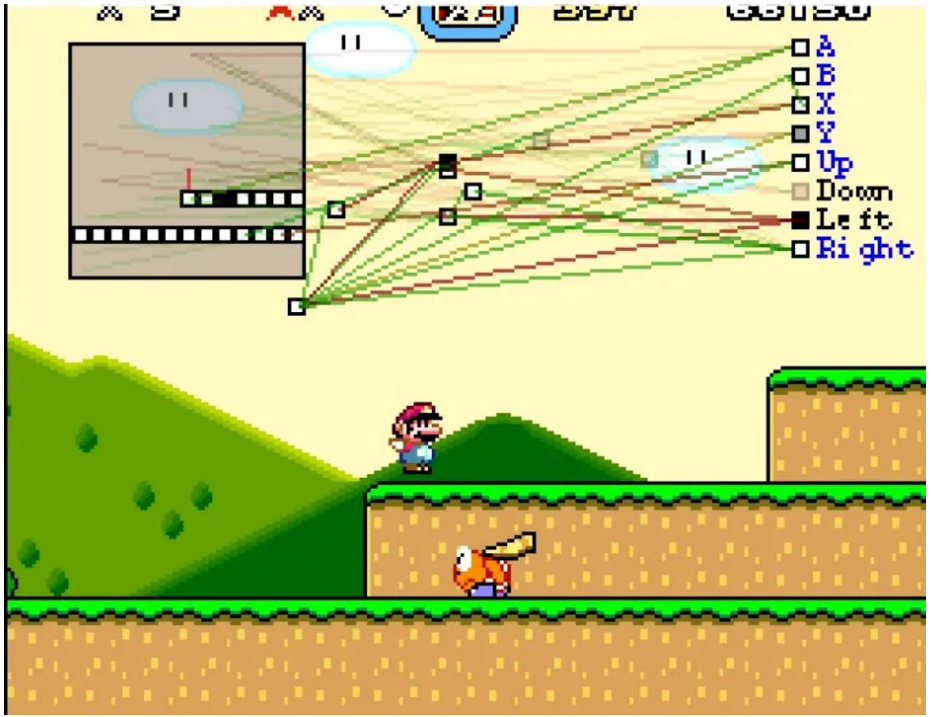
Generate a more realistic image that would resemble an astronomical observation

Here are the more realistic images of a barred spiral galaxy observed in the infrared, resembling actual astronomical observations.

With DALL-E 3 through ChatGPT 4 interface

Use cases
Examples of modern applications

The same processes are used to train robots to walk. But can also be used to infer any function from which you can only define a global posterior fitness score.



*Based on Genetic Neural Evolution
→ *https://www.youtube.com/watch?v=qv6UVOQ0F44*

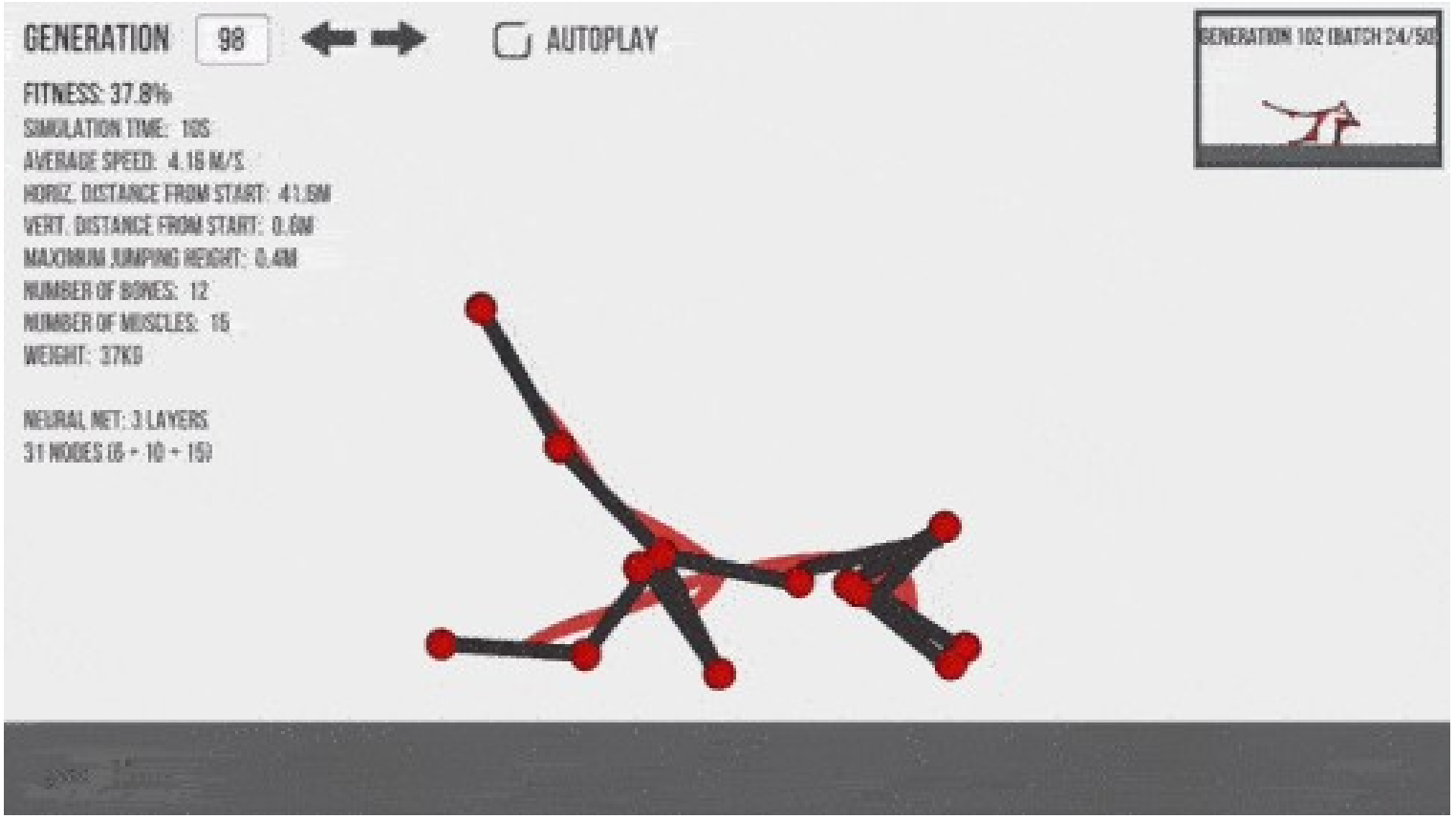
*Based on Deep Reinforced Learning method AC3
→ *https://arxiv.org/abs/1602.01783v2*
→ *https://youtu.be/nMR5mjCFZCw*

Use cases

Examples of modern applications

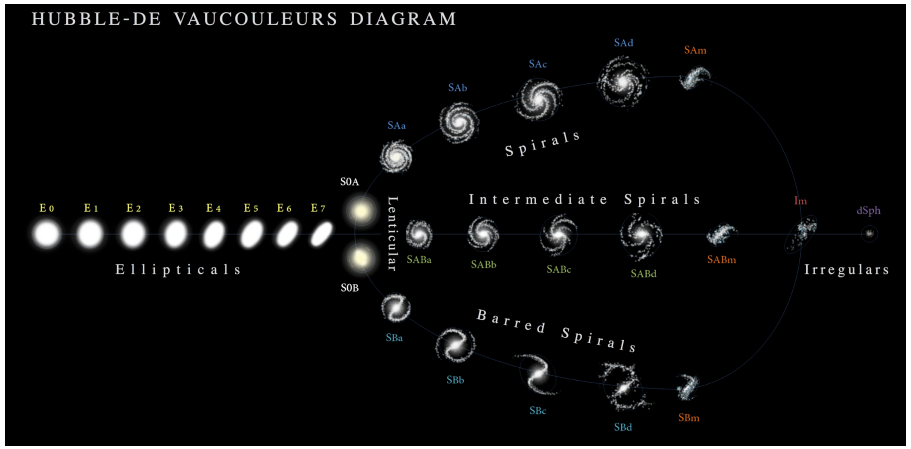
*Based on the YOLO Convolutional Neural Network architecture for object detection
→ https://www.youtube.com/watch?v=MPU2HistivI and https://arxiv.org/abs/1804.02767

Use cases
Examples of modern applications

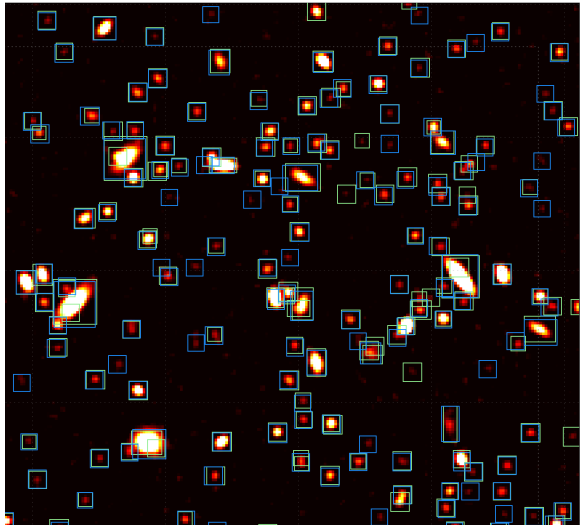


**Based on Genetic Neural Evolution*
→ Interactive creature creation at https://keiwan.itch.io/evolution

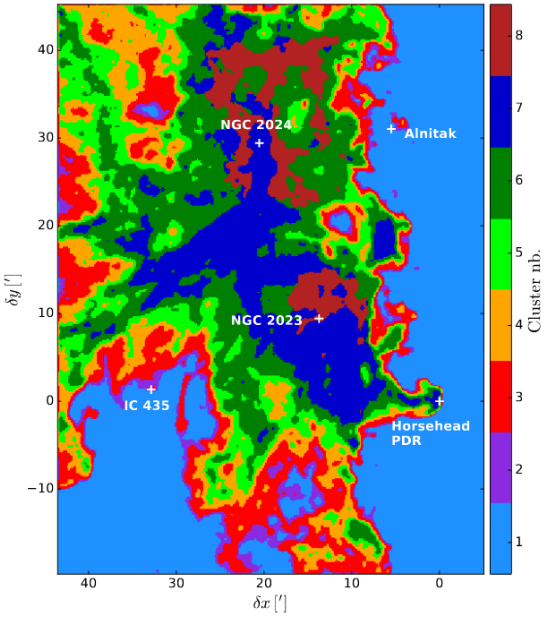
What about astronomical applications ?



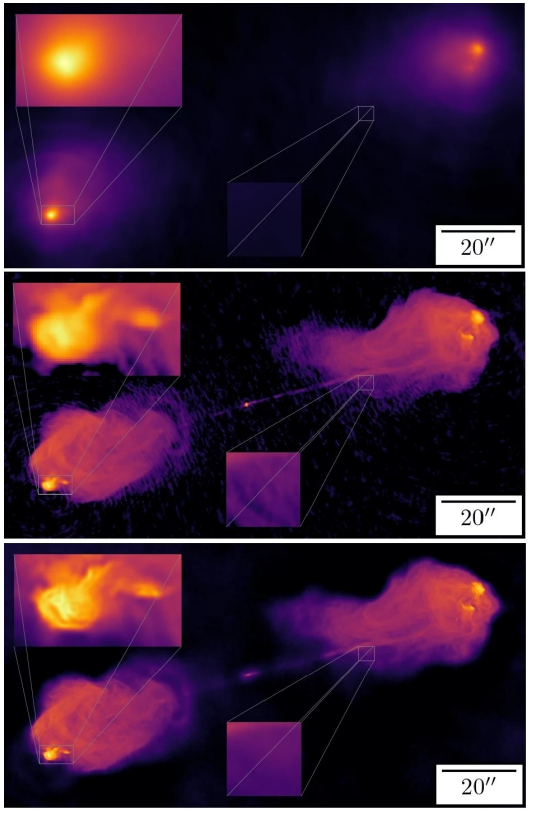
Morphological classification of galaxies



Detection and characterization of galaxies



Molecular cloud emission clustering



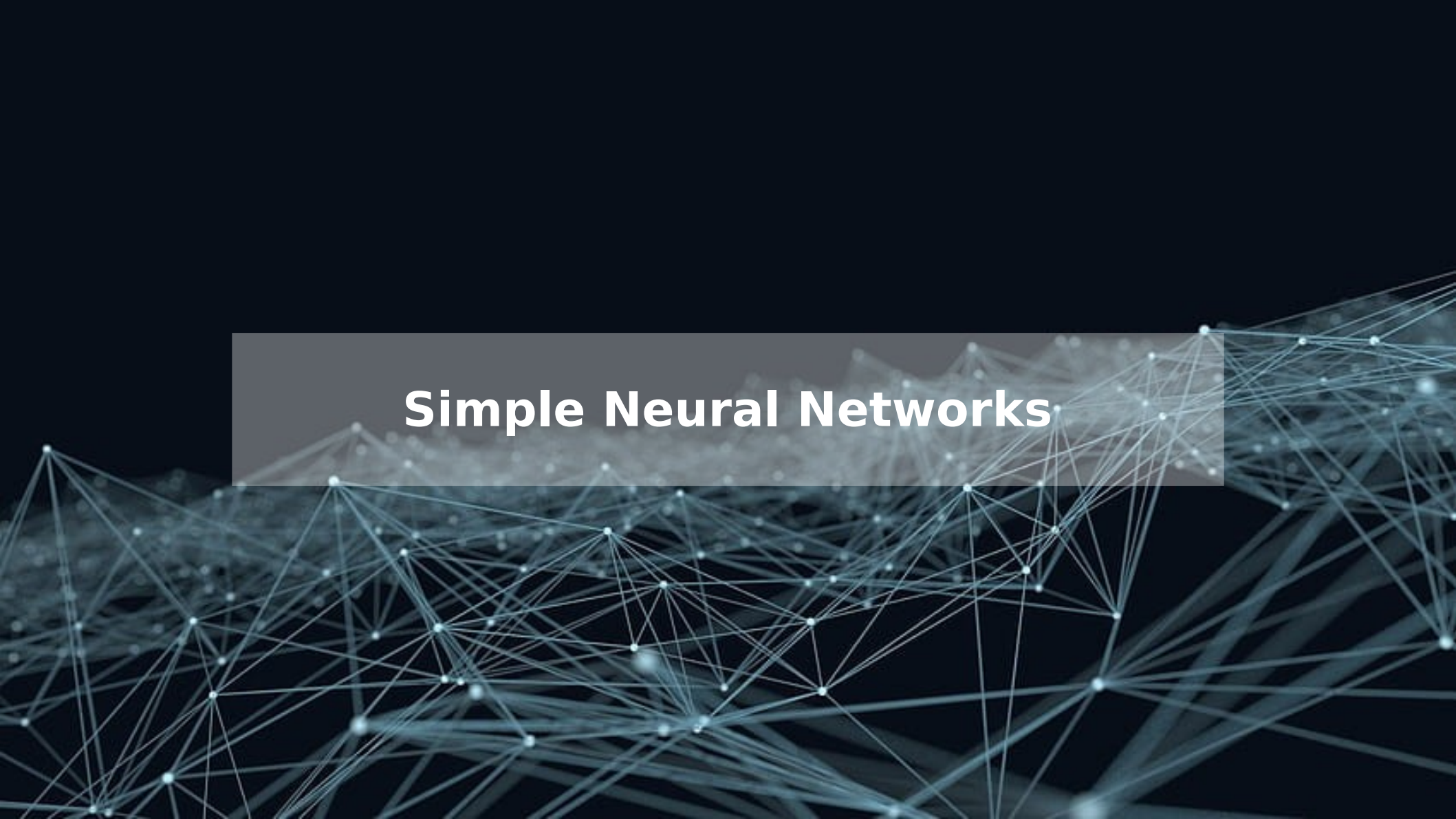
Radio-astronomical image enhancing

And much more :

- *Cosmological simulation acceleration*
- *Inverse modeling for cosmological models*
- *Stellar classification based on spectra*
- *ISM turbulence regime classification*
- *Transient events time series prediction*
- *...*

Wide diversity of objects, methods, scales, data representation, etc...

Simple Neural Networks



Neural Network: The brain as model

« There is a fantastic existence proof that learning is possible, which is the bag of water and electricity (together with a few trace chemicals) sitting between your ears [...] which is the squishy thing that your skull protects »

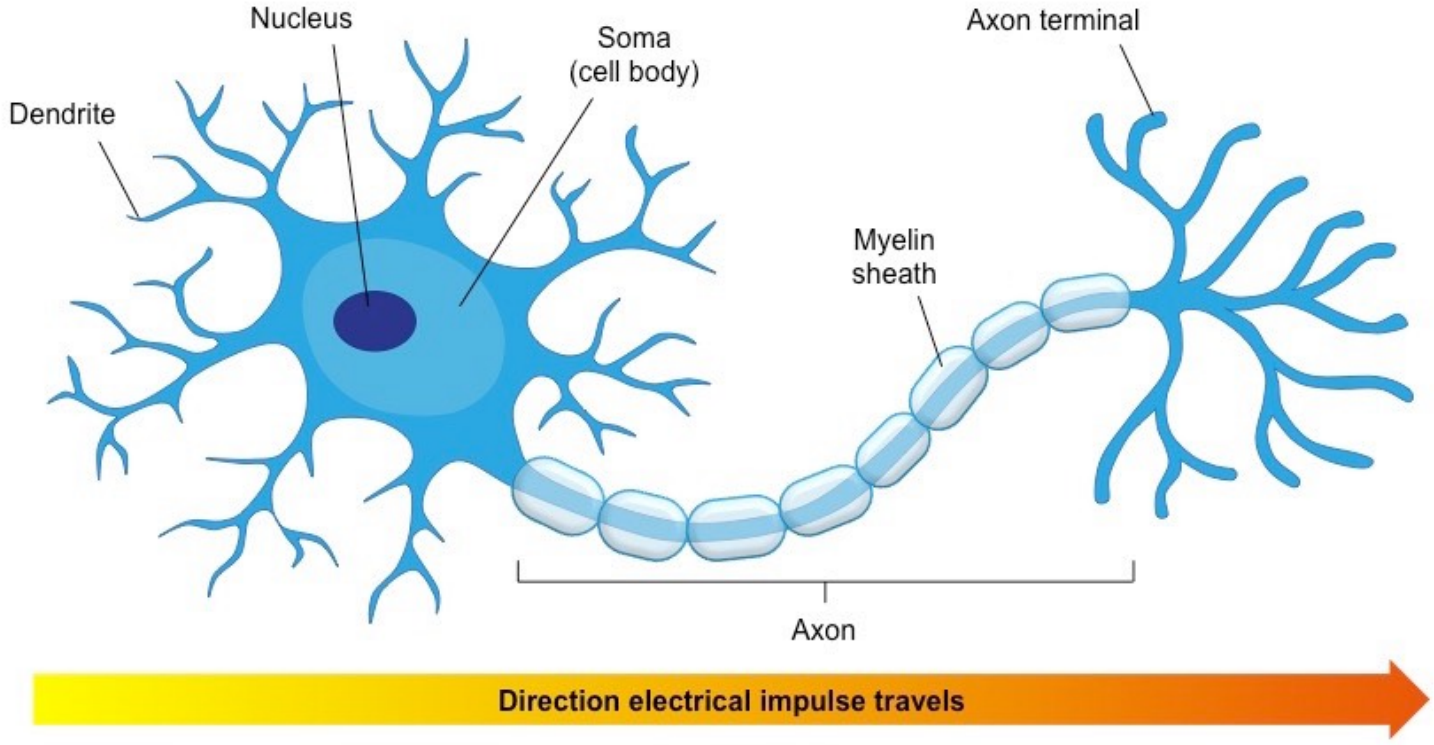
**Stephen Marsland*

The brain does exactly what we want for data analysis:

- Extract complex information in a compressed form
- Deal with noisy and/or inconsistent data
- Work in highly-dimensional spaces
- Give the appropriate answer most of the time
- Provide results very quickly
- Remain robust through aging (neuron loss)

The biological neuron

Elementary brick of a biological brain (10^{11} in the human brain).
The idea will be to use it as a model to emulate learning capabilities.



Perform the sum of various electrochemical input signals. If this total signal is sufficient, it sends a new signal through its axon to transfer information (toward other neurons).

Biological Neural Networks

A connection between two neurons is called a synapse (10^{14} in the human brain).
A neuron is a binary compute unit that either “fire” or “not-fire” in response to a signal.

➡ **In this view, a brain is a massively parallel super-computer of 10^{11} processing units.**

A simplified view of how it learns

The synapses represent the “strength” of the connection between two neurons.

Learning = modifying this connection (either positive or negative and changing its intensity)

➡ **This is called plasticity**

The Hebb law defines a simplified rule for learning:

If two neurons “fire” at the same time, there must be some **correlation** between them, and their connection must be strengthened.

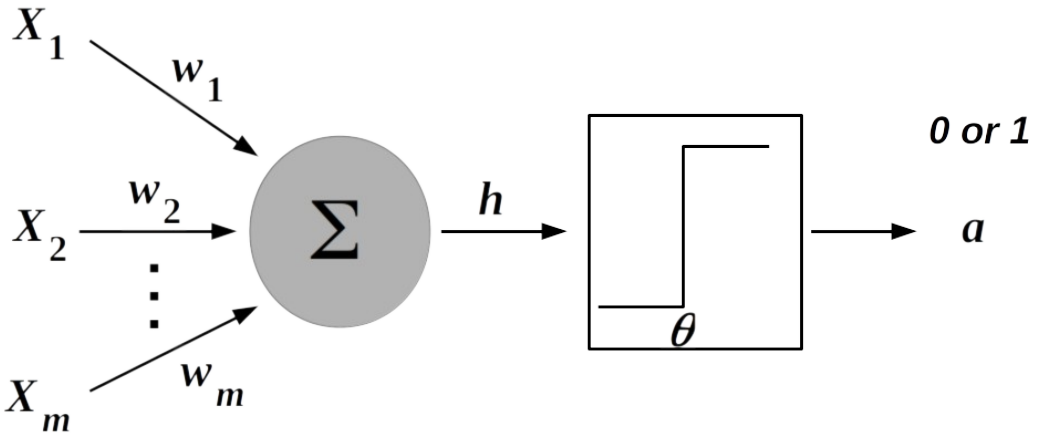
➡ **This is called conditioning**

**These rules are not enough to train a neural system but illustrate the processes that occur in the biological brain.*

To create an algorithm from these biological concepts, one first need to create a mathematical model.

Model of a Neuron

Mathematical model from McCulloch and Pitts, inspired by the biological neuron.

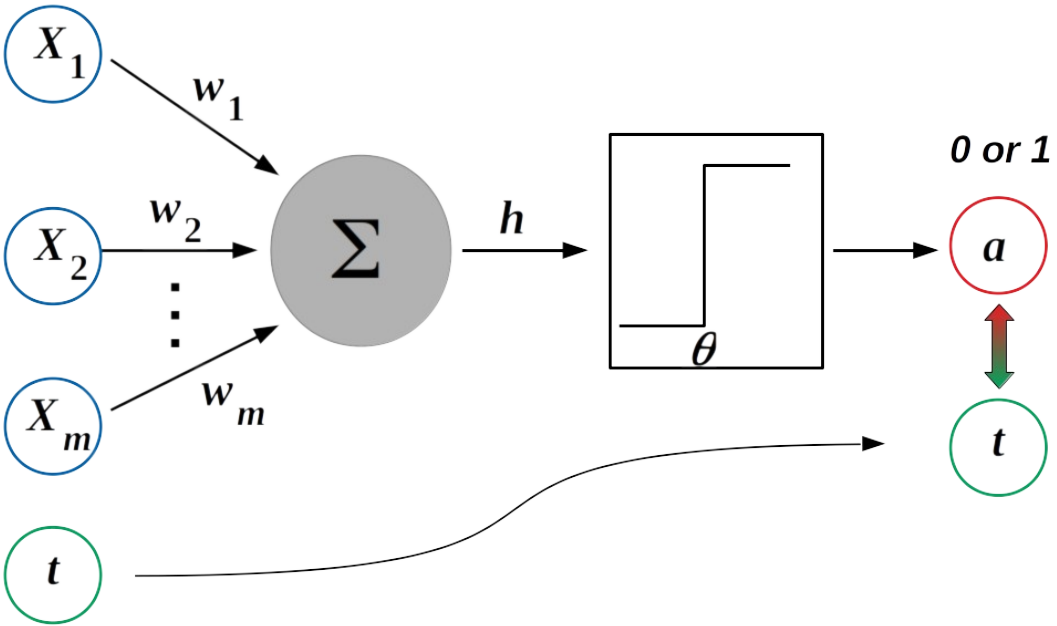


Its main components are:

- An input vector X_i that represents the dimensions of a given object
- A set of weights w_i that links the various input dimensions to the neuron
- A sum function h that defines how these weights are combined with the input dimensions
- An activation function $g(h)$ that defines if the network should remain in a “0” state or should be activated to a “1” state, depending on the results of the sum.

$$h = \sum_{i=1}^{m} X_i w_i \qquad a = g(h) = \begin{cases} 1 & \text{if} \quad h > \theta \\ 0 & \text{if} \quad h \leq \theta \end{cases}$$

Training a Neuron



$$h = \sum_{i=1}^{m} X_i w_i \qquad a = g(h) = \begin{cases} 1 & \text{if} \quad h > \theta \\ 0 & \text{if} \quad h \leq \theta \end{cases}$$

The learning process for this model is supervised
→ Each input vector is associated to a target value

So what is the purpose of the neuron if the expected value is already known ?

➡ **Generalization**

Find patterns in the parameter space so it can perform prediction on vectors with unseen (but close) values.

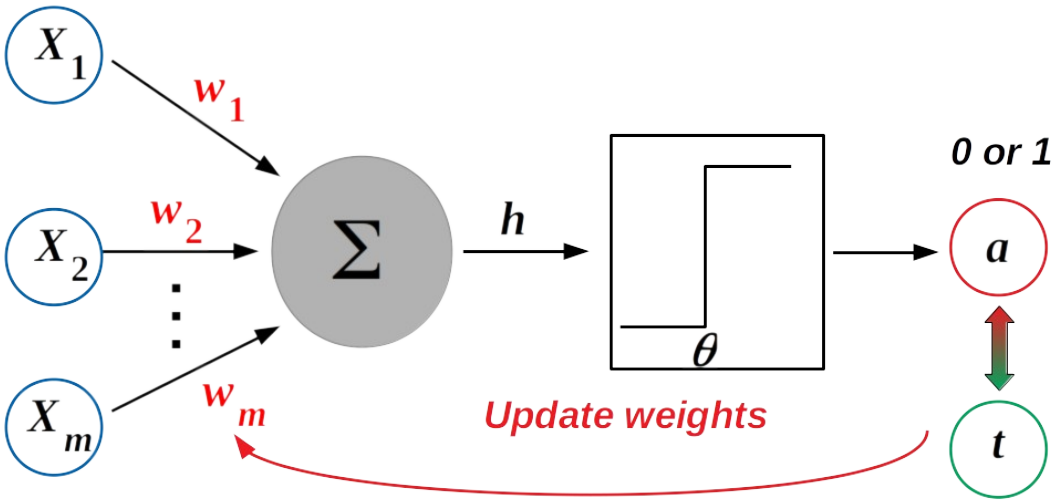
In practice, such a simple neuron optimizes a linear separation in the parameter space (**hyperplane**).

How does this model learn ?

Which parameters can or cannot be modified ?

The input, output, and targets are fixed, so the learning relies on modifying the **weights and the activation threshold**.

Training a Neuron



$$\omega_i \leftarrow \omega_i - \eta\ (a - t)\ x_i$$

$$h = \sum_{i=1}^{m} X_i w_i \qquad a = g(h) = \begin{cases} 1 & \text{if} \quad h > \theta \\ 0 & \text{if} \quad h \leq \theta \end{cases}$$

How to proceed when the output does not correspond to the target ?

There are m weights w_i associated with the network corresponding to the input dimensions.

How to modify the weights?

- If the output state is 1 while it should be 0, the weights need to be lowered
- If the output state is 0 while it should be 1, the weights need to be increased

To quantify this modification, one needs to choose an error function E.

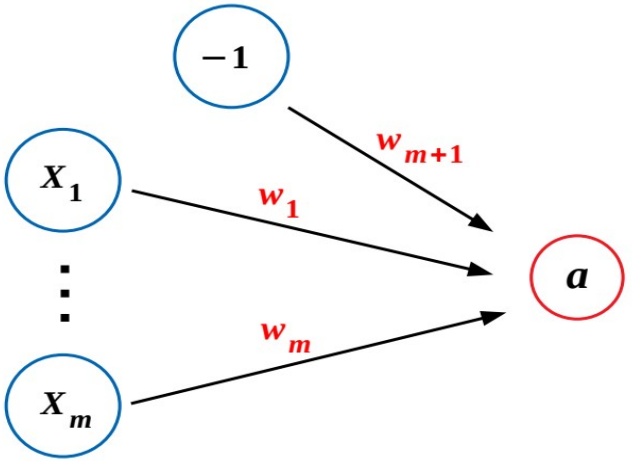
$$E = 0.5 \times (a - t)^2$$

In the end, the weight update is defined as proportional to the input value, to the derivative of the error function for each dimension, and scaled by a learning rate* factor.

$$\omega_i \leftarrow \omega_i - \eta\ (a - t)\ x_i$$

*Typically between 0.1 and 0.4, details will be given later

The bias node

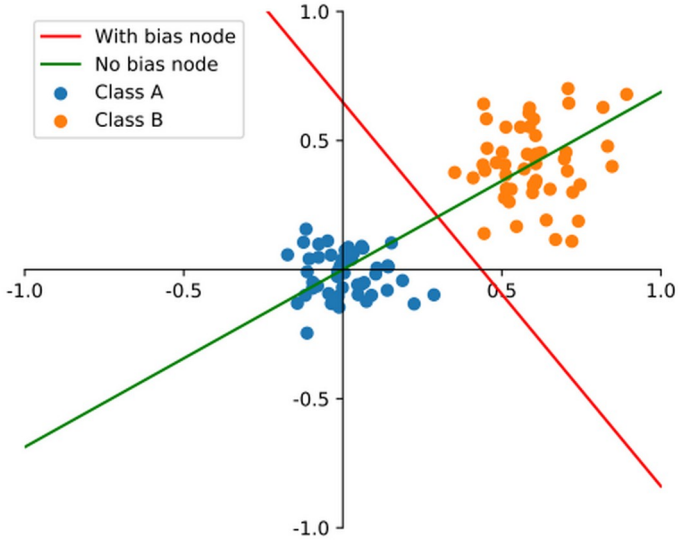


Problem with the previous formalism

The linear separation presents a fixed *f(0) = 0 point*.
The weight correction is also 0, regardless of the input value.

Solution

Add a bias input node that acts as an additional input with a constant value (usually -1). It has its own **variable weight** so it can learn the shift of the intercept position. The size of both the input vector and the weights vector is now *m+1*.



To plot the found linear separation of a neuron, we need to find where in the parameter space the neuron changes state, which corresponds to:

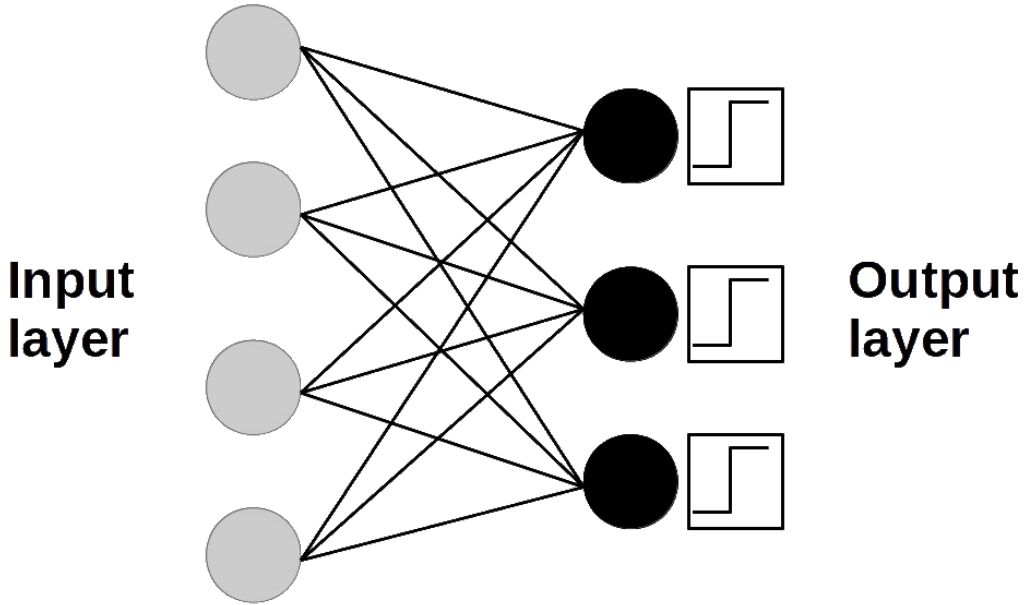
$$\sum_{i=1}^{m} X_i w_i = h = 0$$

For a 2D parameter space it results in:

$$X_1 = (W_b - X_0 W_0)/W_1$$

Note that the direction of activation is orthogonal to the linear separation.

The Perceptron algorithm



A single neuron only performs a linear separation, which is insufficient for many applications.

The simplest way to combine neurons on a single problem is to **stack them independently**. Each neuron is connected to the input vector with its own set of weights.

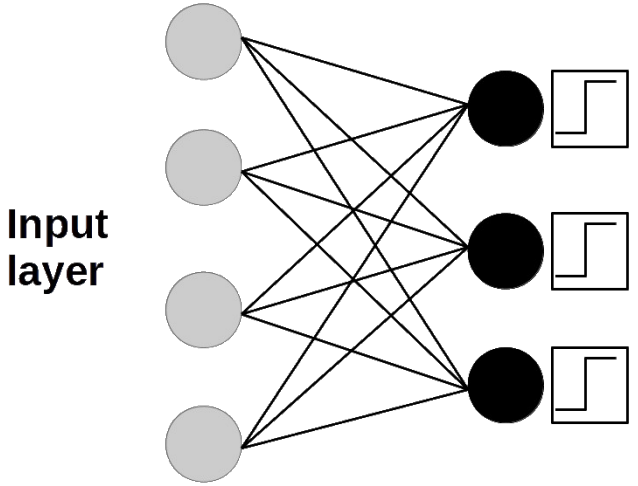
The training procedure is identical, but this time there is an index j to represent the neurons, and the weights are now in the form of a 2D matrix.

$$h_j = \sum_{i=1}^{m+1} x_i \omega_{ij} \qquad a_j = g(h_j) = \begin{cases} 1 & \text{if} \quad h_j > \theta \\ 0 & \text{if} \quad h_j \leq \theta \end{cases}$$

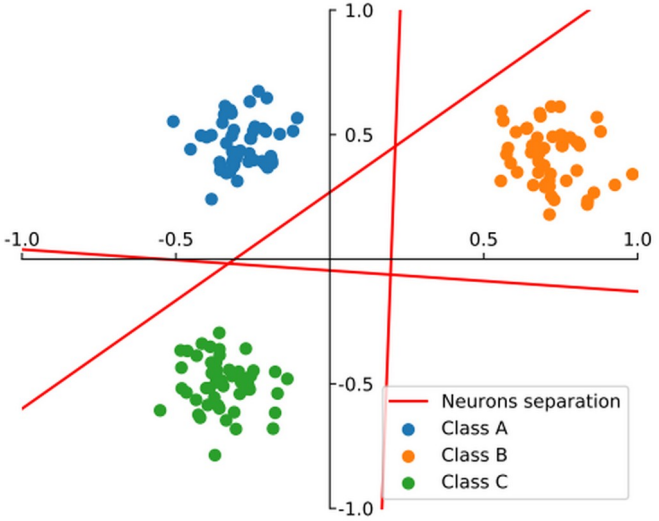
$$\omega_{ij} \leftarrow \omega_{ij} - \eta \left(a_j - t_j \right) \times x_i$$

The combination of this training procedure and this neuron connection scheme is called the single layer "Perceptron" (Rosenblatt 1958).

The Perceptron algorithm



C1	C2	C3
1	0	0
0	1	0
0	0	1



How to have multiple neurons work on the same problem ?

1) Encode the output vector (e.g in binary format)
2) Have one neuron per output class and target a format with a 1 only in the appropriate class, e.g [1,0,0]-[0,1,0]-[0,0,1]

**Warning: Since each neuron only performs a linear separation, each class must be linearly separable from all the others.*

The Perceptron algorithm

- **Initialization**
  - Set the starting weights to small random values (positive and negative). Can be drawn from a uniform or Gaussian distribution centered on zero.

- **Training**
  - For a given number of steps T, or until the output is “correct”
    - For each input vector
      - Compute the activation of each neuron *j* using the activation function *g :*

$$a_j = g\left(\sum_{i=0}^{m+1} w_{ij}x_i\right) = \begin{cases} 1 & \text{if} \quad \sum_{i=0}^{m+1} w_{ij}x_i > 0 \\ 0 & \text{if} \quad \sum_{i=0}^{m+1} w_{ij}x_i \leq 0 \end{cases}$$

      - Update each weight individually using :

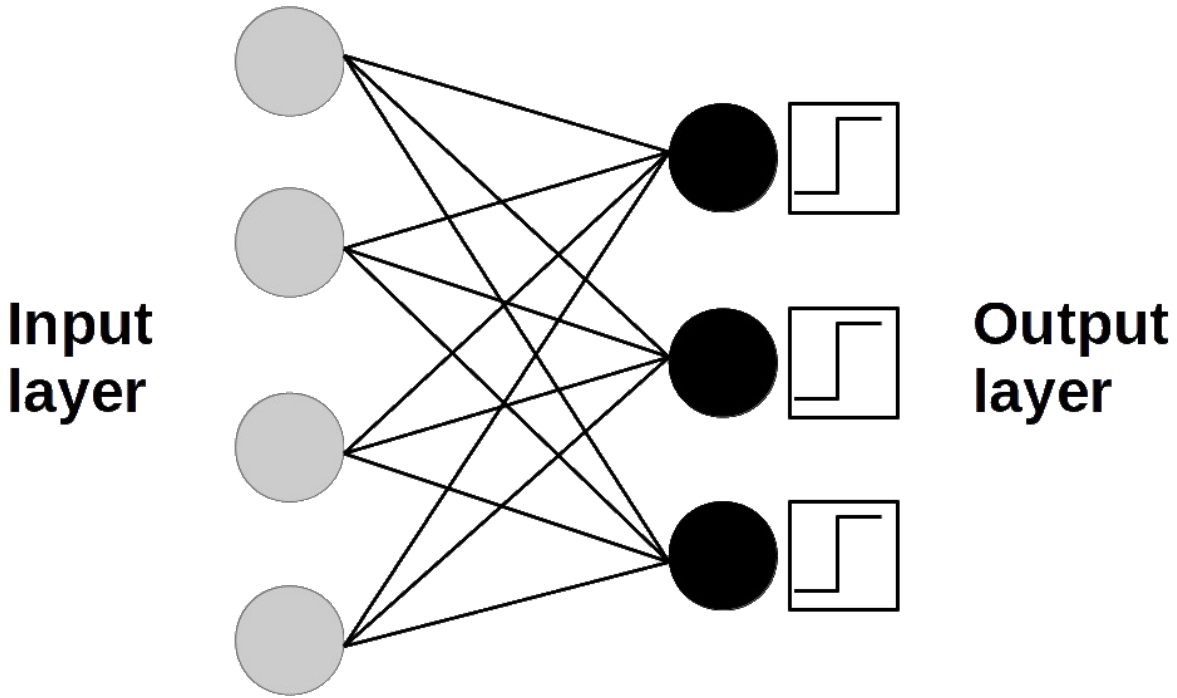
$$\omega_{ij} \leftarrow \omega_{ij} - \eta\left(a_j - t_j\right) \times x_i$$

- **Inference**
  - Compute the final activation of each neuron *j* for each input vector to test

Constructing a “Deep” Neural Network

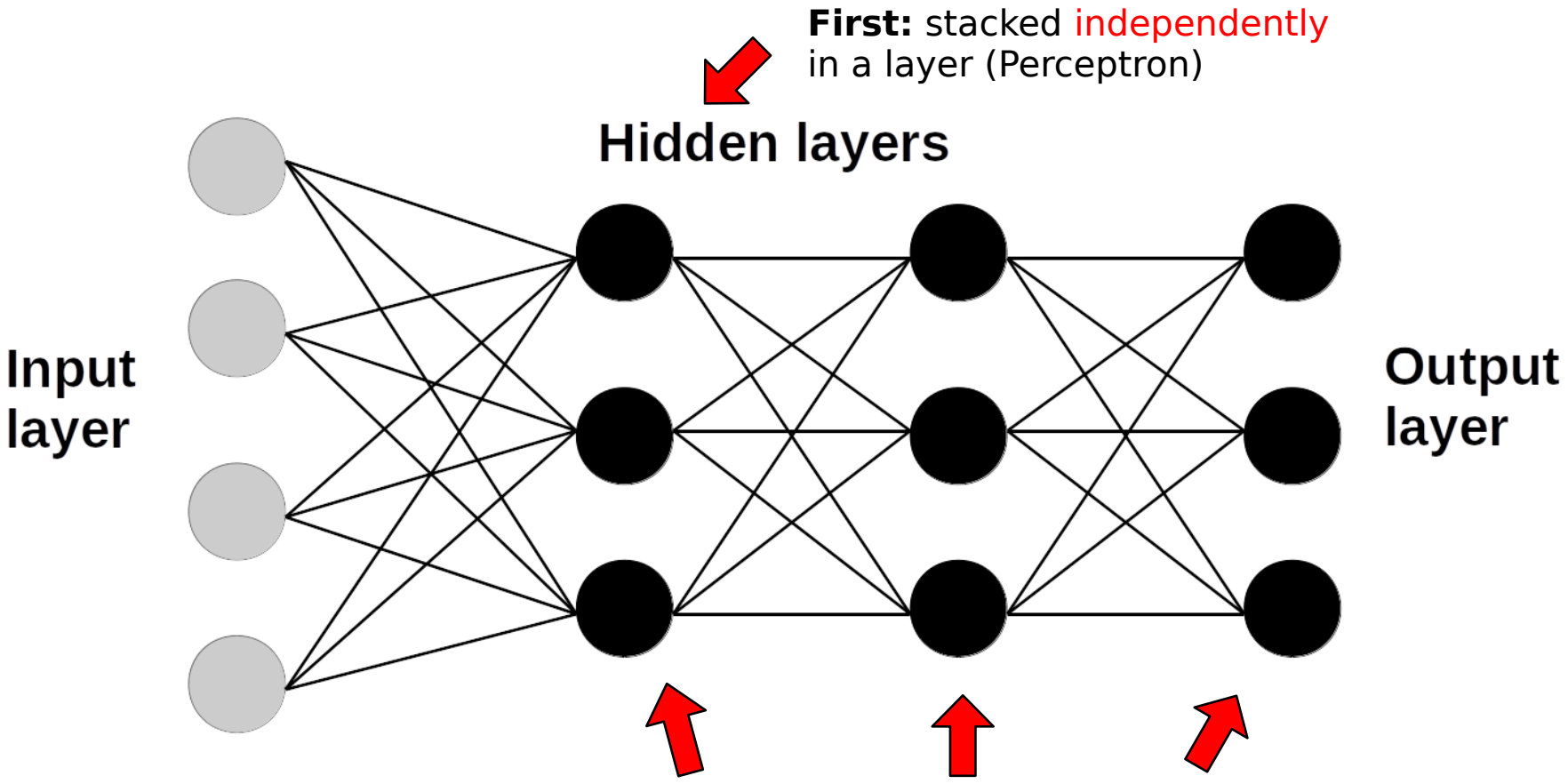
For more complex problems more neurons must be used.

First: stacked independently in a layer (Perceptron)



Constructing a “Deep” Neural Network

For more complex problems more neurons must be used.



Constructing a “Deep” Neural Network

In addition to the MLP architecture, it is necessary to change the activation to allow non-linear combinations

➡ **Change for a Sigmoid (logistic) function**

$$g(h) = \frac{1}{1 + \exp(-\beta h)}$$

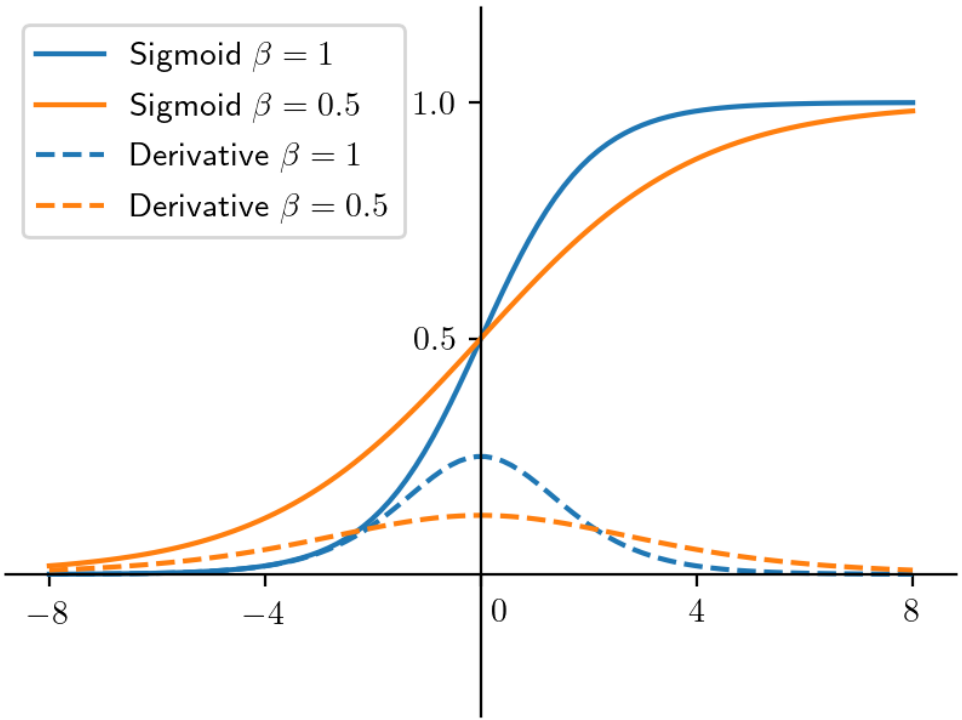
Beta represents the slope of the function.

The sigmoïd is continuous while preserving the desired global binary behavior in its extremity.

It has a nice derivative form :

$$\frac{\partial a_j}{\partial h_j} = \beta a_j (1 - a_j)$$

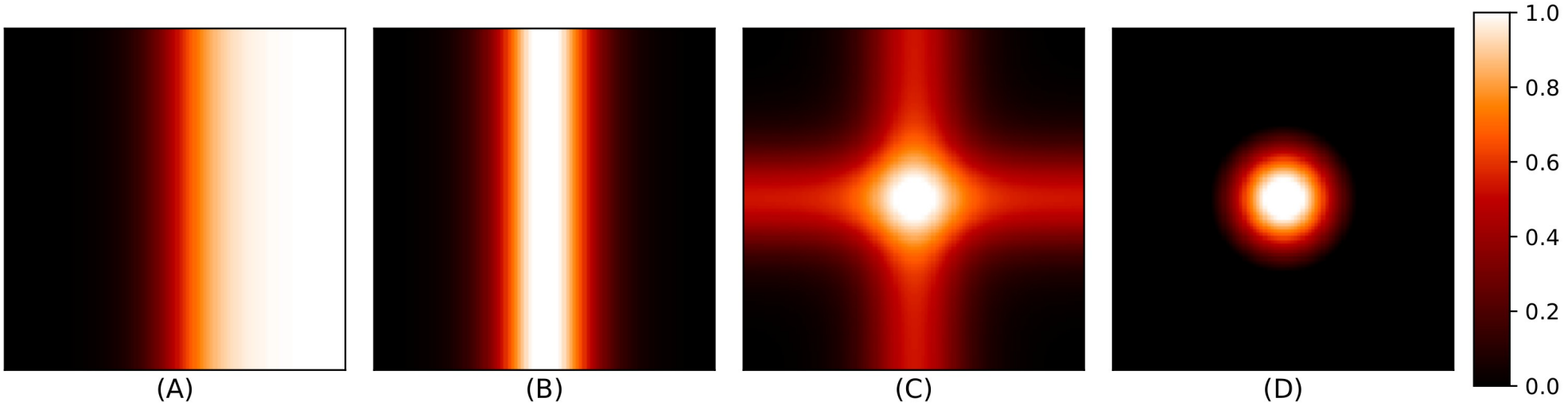
For regression problems, the output layer can either use sigmoïds or simple linear activation functions since the previous layers will handle non-linearity.



Constructing a “Deep” Neural Network

Stacking layers that use a sigmoid activation allows the construction of more and more complex functions. The image below demonstrates that it is possible to reconstruct hill shapes (B), bumps (C), and point-like functions.

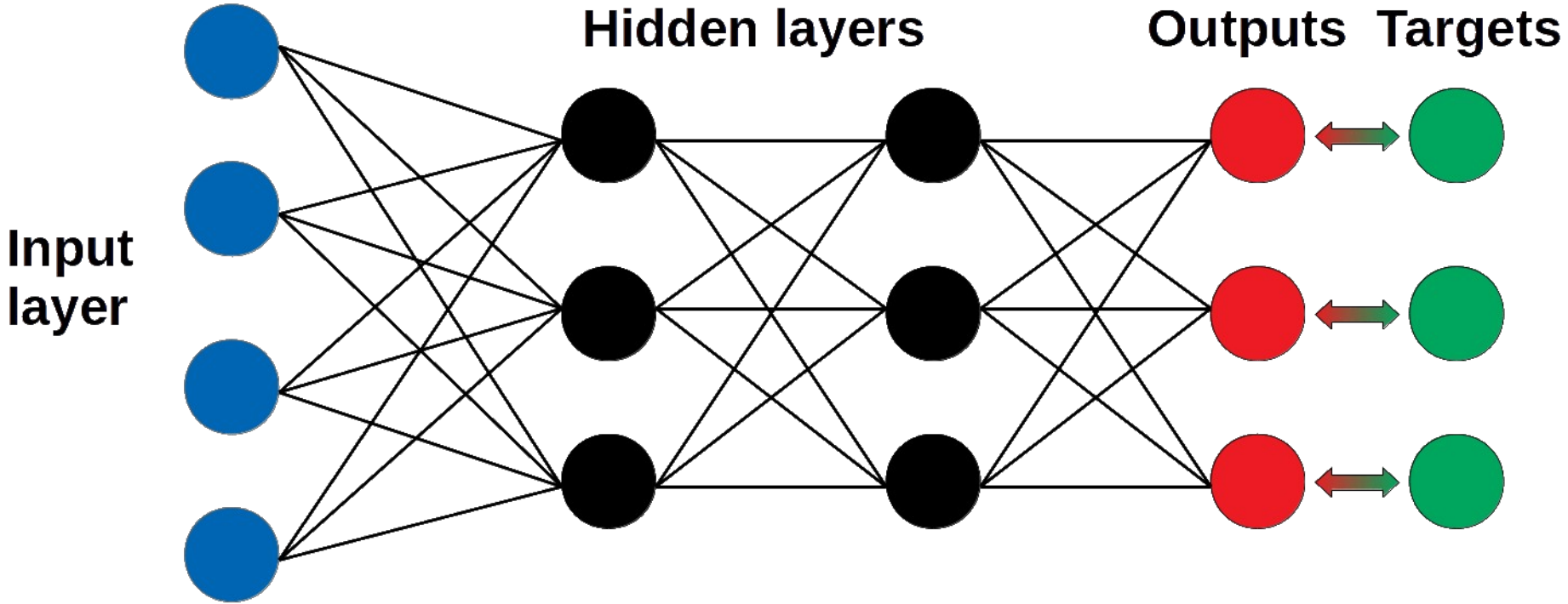
➡ With these improvements, the MLP is a **Universal Function Approximator**



In practice, based on the universal approximation theorem, a MLP with a single hidden layer remains a UFA at the condition of having a large enough number of neurons.

Error gradient backpropagation

Problem : how to correct the output of neurons in the hidden layers ?



$$\omega_{ij} \leftarrow \omega_{ij} - \eta \frac{\partial E}{\partial \omega_{ij}} \quad \Rightarrow \quad \frac{\partial E}{\partial \omega_{ij}} = \delta_l(j) \frac{\partial h_j}{\partial \omega_{ij}} \quad \text{with} \quad \delta_l(j) \equiv \frac{\partial E}{\partial h_j} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial h_j} = \frac{\partial a_j}{\partial h_j} \sum_k \omega_{kj} \delta_{l+1}(k)$$

Actual Network equations for an MLP with a single hidden layer

Changes in the Perceptron algorithm required to obtain the Multi-Layer Perceptron algorithm :

- Change the activation function of all neurons, with $\beta \geq 1$

$$g(h) = \frac{1}{1 + \exp(-\beta h)}$$

- Add a hidden layer with its own bias node (connect input → hidden, and hidden → output)
- Implement the "back-propagation" with $E(a^o, t) = \frac{1}{2} \sum_{k=1}^{N_o} (a_k^o - t_k)^2$ using the following equations:

$$\delta_o(k) = \beta a_k^o (1 - a_k^o)(a_k^o - t_k)$$

$$\delta_h(j) = \beta a_j^h (1 - a_j^h) \sum_{k=1}^{N_o} \delta_o(k) \omega_{jk}$$

$$\omega_{jk} \leftarrow \omega_{jk} - \eta \delta_o(k) a_j^h$$

$$v_{ij} \leftarrow v_{ij} - \eta \delta_h(j) x_i$$

Example with classification

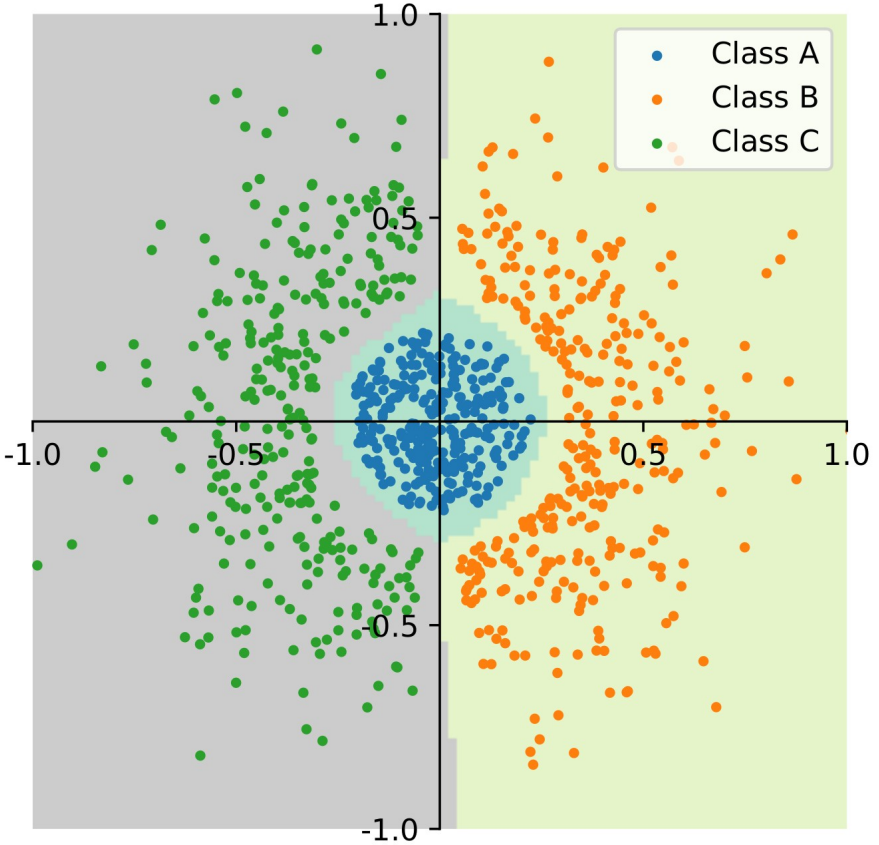


Illustration of a three-class separation in a two-dimensional feature space using a trained MLP with 3 output neurons and 8 hidden neurons, all with sigmoid activations. The light background colors indicate the regions of the feature space that the network has attributed to each class.

How to measure classification performance ?

Confusion Matrix

- Uses "**observational proportions**"
- Provides confusion between classes
- Is much more precise than the average accuracy

$$\text{Recall} = \frac{TP}{TP + FN}$$

Actual \ Predicted	Positive	Negative	Recall
Positive	93	7	93.0%
Negative	46	954	95.4%
Precision	66.9%	99.27%	95.18%

$$\text{Precision} = \frac{TP}{TP + FP}$$

$TP \equiv$ True Positive $\quad TN \equiv$ True Negative

$FP \equiv$ False Positive $\quad FN \equiv$ False Negative

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

ML methods usually learn better on balanced dataset **BUT**

the results must always be represented using "observational proportions", with imbalance.

Example: The Pima Indian dataset

Dataset from the UCI Machine Learning repository.

Provides 8 physiological measurements for 768 female native Americans and provides a class depending on whether the individual has developed diabetes.

The objective is to **predict if a person has diabetes** based solely on the 8 input measurements.

A simple **Perceptron** can reach **70% accuracy**, while an **MLP** with 8 hidden neurons with can reach **90% accuracy.**

```
Iteration : 1400
**************************************************
Confmat :
                31          2          93.94
                 3         14          82.35

  Precision     91.18     87.50 Accu    90.00
**************************************************
Average test set quadratic error: 0.158614
Average training dataset quadratic error : 0.135860
```

Attribute Number	Attribute
1	Patient age
2	Body mass index (kg/m2)
3	Concentration of plasma glucose
4	2-h serum insulin (mu U/mL)
5	Thickness of triceps skin-fold (mm)
6	Pedigree function of diabetes
7	Number of times patient pregnant
8	Diastolic blood pressure (mmHg)
9	Class 0 or 1

Input dimension : 8 (+1)

Output dimension : 1
(or 2 if using one neuron per class)

Number of example : 768

Class distribution : 500 class 0; and 268 class 1

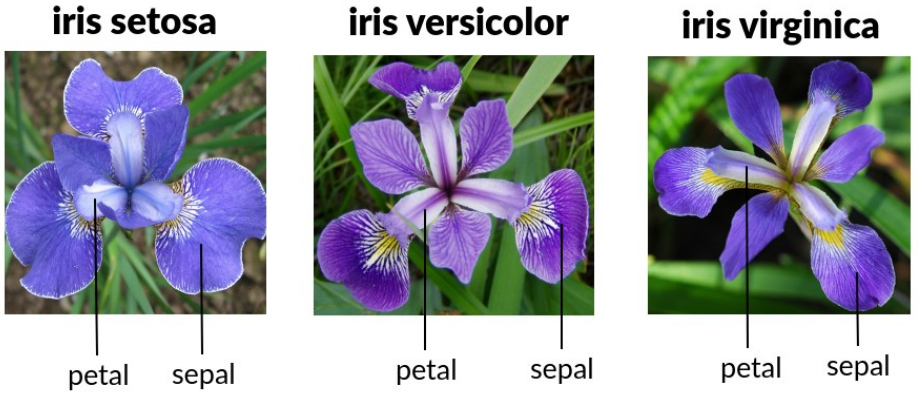
Example: The Iris dataset

Dataset from the UCI Machine Learning repository.

Provides 4 sizes for a set of 150 Iris flowers and provides membership to one of 3 classes.

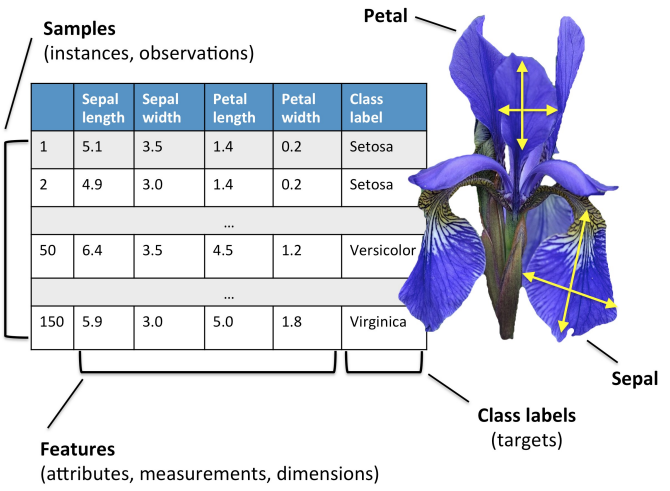
The objective is to **predict the corresponding class** based on the 4 input measurements for each flower.

A simple **Perceptron** can reach **80% accuracy,** while an **MLP** with 4 hidden neurons can reach **100% accuracy**.




```
Iteration : 800
**************************************************
Confmat :                                       Re
                18          0          0        100.00
                 0         17          1         94.44
                 0          0         14        100.00

  Precision    100.00     100.00      93.33 Accu   98.00
**************************************************
```



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

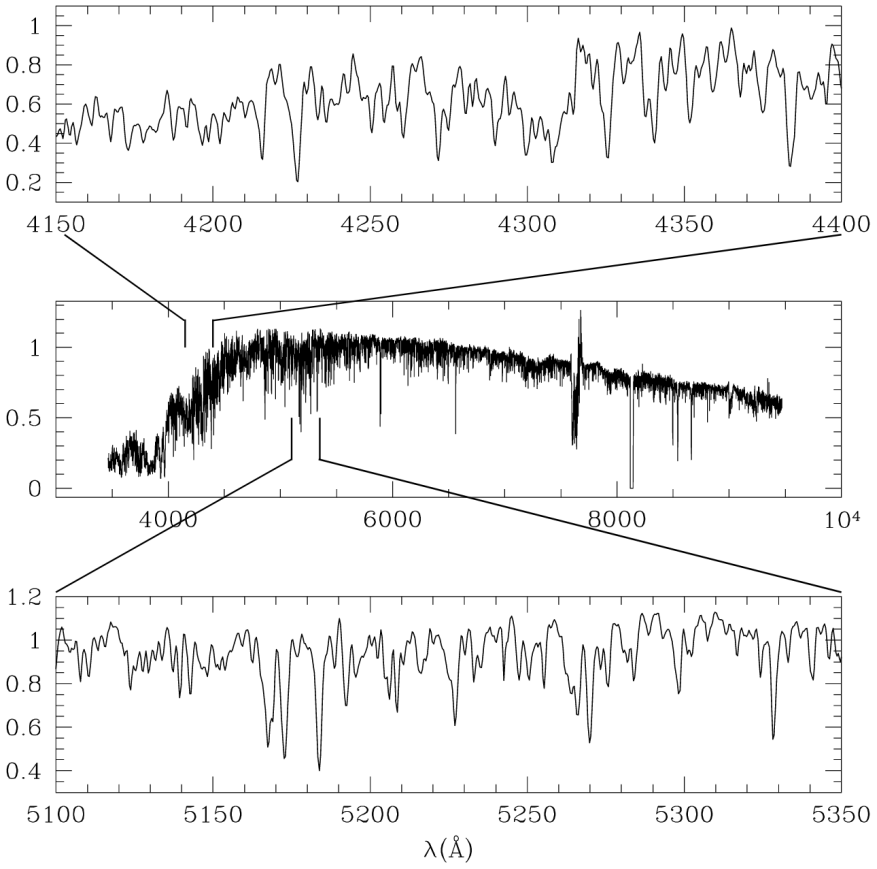
Input dimension : 4 (+1)
Output dimension : 3
Number of example : 150
Class distribution : 50 examples per class

Example: The Spectra dataset

This dataset contains stellar spectra obtained with the 0.9m Coudé Feed telescope at Kitt Peak National Observatory, classified into various spectral types (*F. Valdes et al. 2004*).

The provided dataset here is a simplification that only keeps 1115 homogeneous spectra with half the resolution (3753 “pixels” remains).

The target only provides the 7 regular spectral types for classification to increase the number of examples per class. The dataset is highly imbalanced which require rebalancing the training dataset!



Input dimension : 3753 (+1)
Output dimension : 7
Number of example : 1115
Class distribution : strong imbalance

```
Iteration : 1500
*********************************************************************
Confmat :                                          Recall
                5         0         0         0         0         0         0       100.00
                1        37         5         0         0         0         0        86.05
                1         0        11         1         0         0         0        84.62
                1         0         1        16         1         0         4        69.57
                0         0         1         2        44         6         1        81.48
                0         0         0         2         2        37         7        77.08
                0         0         0         0         0         2        12        85.71

  Precision     62.50    100.00     61.11     76.19     93.62     82.22     50.00 Accu   81.00
*********************************************************************
```

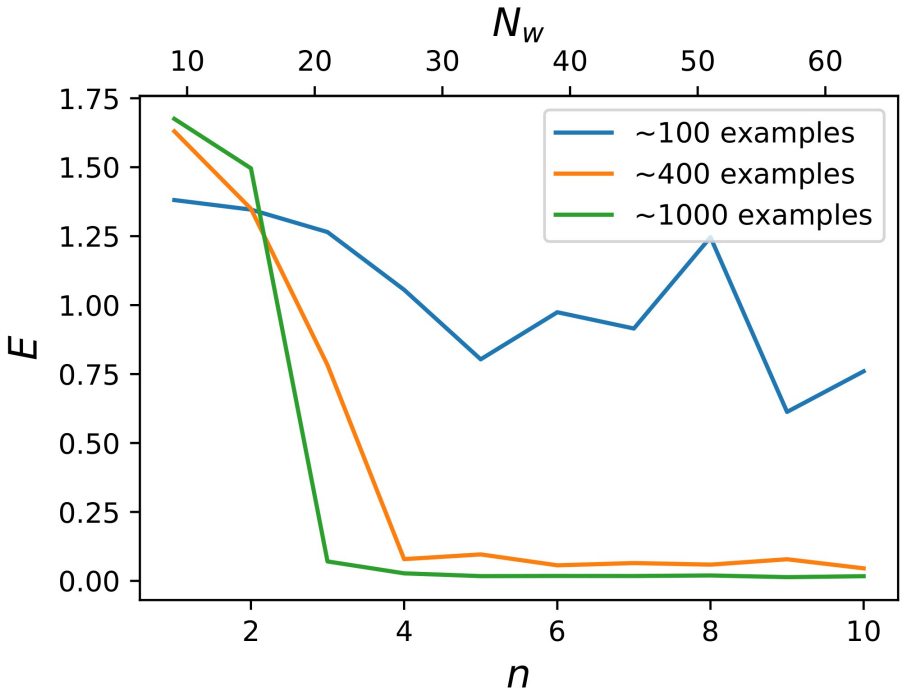
Network depth and size

How to choose the network architecture?
In the case of the MLP, how many layers and neurons should be used per layer?

In this context, the objective is to find a **trade-off between:**
The strength of the network (its expressive power) and the difficulty of training the model

A few ideas that help explore the architectures:

- More data allows to constrain more parameters
  Rule of thumb: need roughly 10 times the number of weights as training examples
- More neurons increase the strength but add parameters to fit.
  Reminder: a neuron is a linear separation in the parameter space → how many would you need by eye?
  (starting point before exploration)
- More layers allow the construction of complex functions with fewer parameters, but increases training difficulty.
  Reminder: a single hidden layer can approximate any function, but it is not always the optimal solution
  (nb. of parameters, time to converge, etc.)



Finding the appropriate architecture is always an exploratory and iterative process.
Training different networks on the same data is almost always necessary to obtain good results.

Learning rate

Choosing the appropriate learning rate for the task is very important and must consider several aspects of the error space.

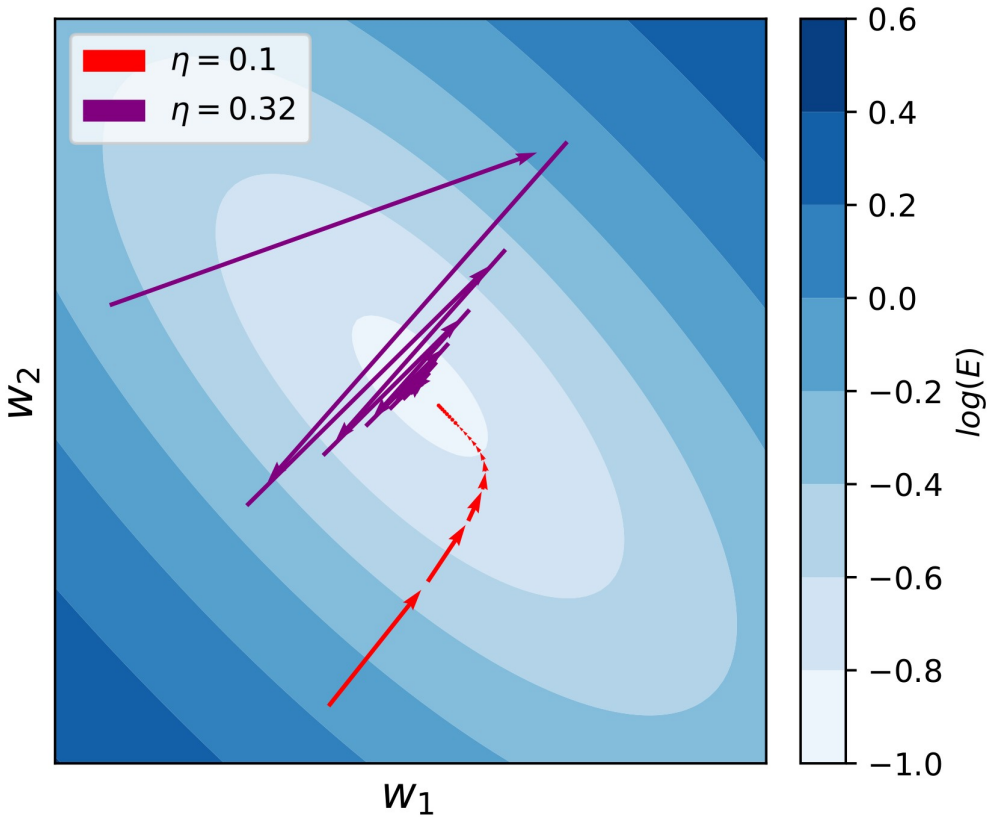
	Small η	**Large η**
Pros	• Increase training stability • Can find narrower minima	• Faster training • Easily switch from a local minima to another
Cons	• Slow training • Might get stuck on local minima	• Might fail to find a narrow minima • Training can be very unstable

The learning rate does not have to be a single value, it can change during the training phase.

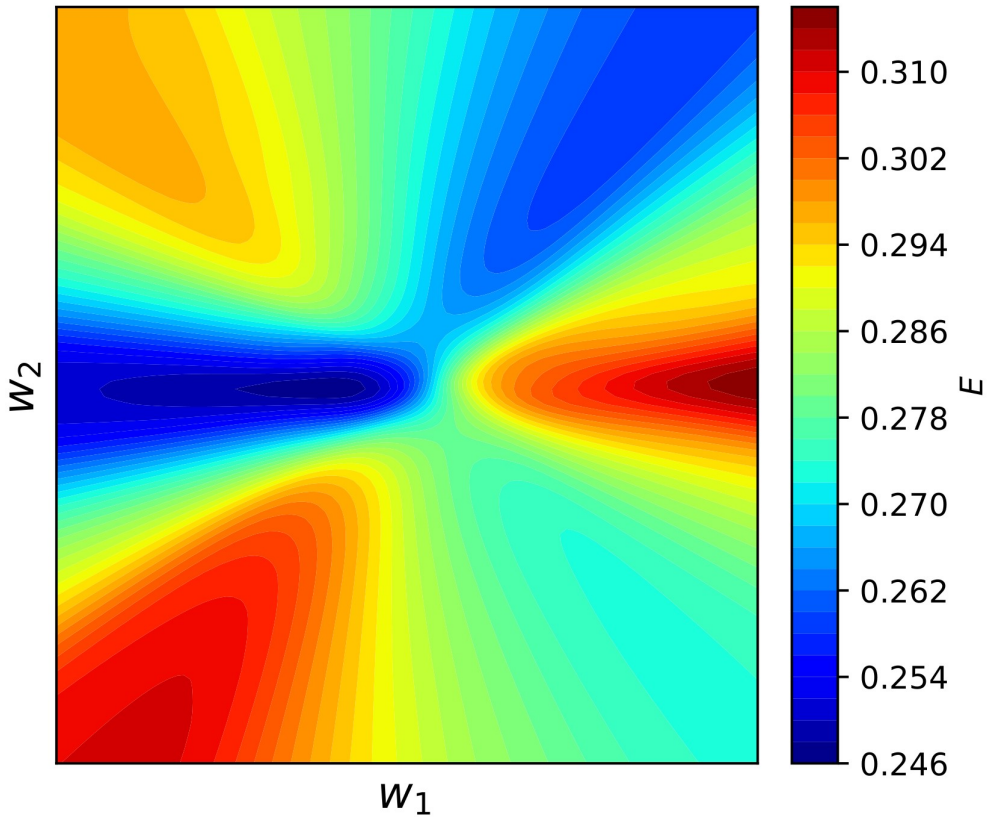
The simple approaches are: pre-established fixed values, linear decay, exponential decay, ...

Most frameworks allow the use of advanced gradient descent scheme that also automatically adapts the learning rate: Adam, RMSprop, Adagrad, etc ...

Learning rate



Logarithm of the error term in the weight space of a simple binary neuron with a two dimensional input parameter. The arrows represent the correction after each epoch for two different learning rates.



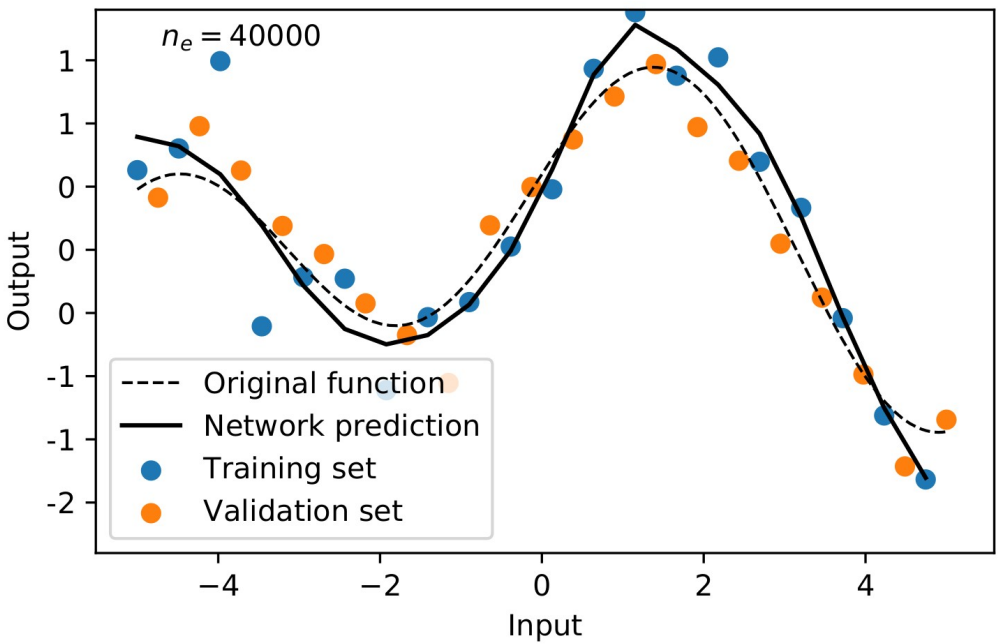
Error term in a 2D weight space of neuron in the hidden layer of a 3 class output MLP. The other weights of the network are frozen while exploring this 2D weight space.

Overtraining

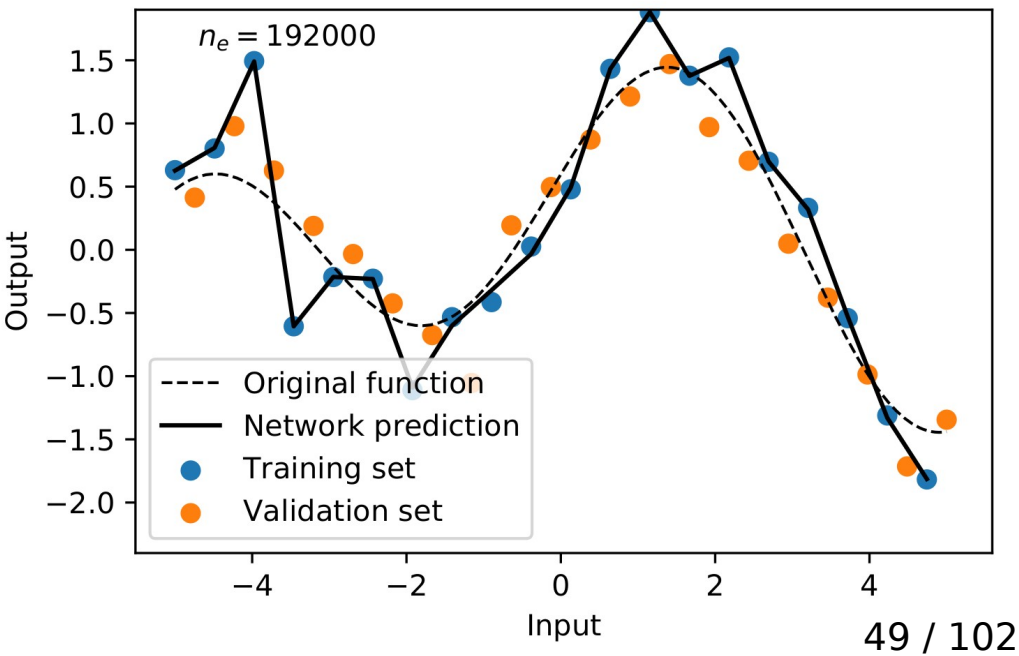
Overtraining: At some point during the training phase, the network that looks at the same data again and again, might start to over-fit!

It means that the network is learning specificities of the dataset that are no longer related to the average function that it should approximate.

Proper fit



Over-fitting



Monitoring overtraining

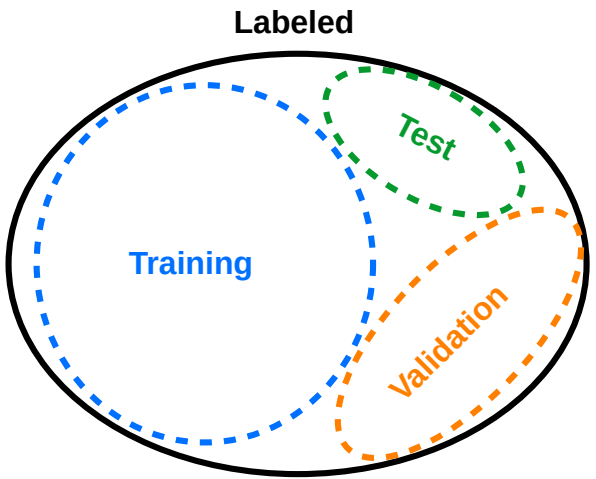
While several techniques can be used to moderate overtraining (like regularization), it is always necessary to monitor it to stop the training at the appropriate time.

Split **labeled dataset** into 3 subsets

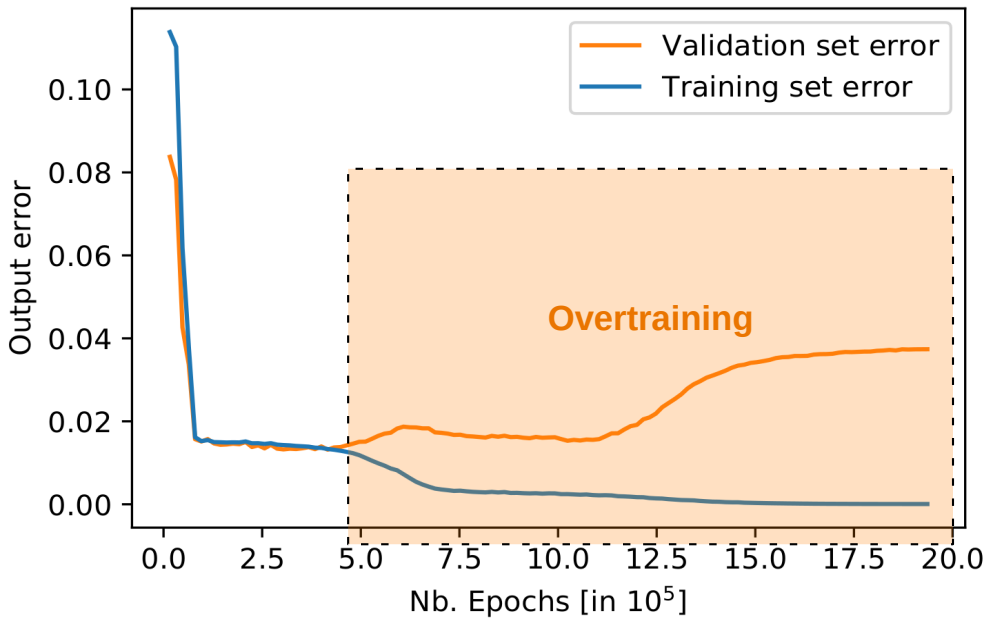
Training set ➤ To train the network

Validation set ➤ To monitor the training process

Test set ➤ To assess the quality of the prediction



Typical proportions

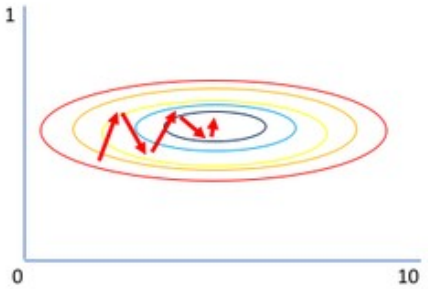


Normalization

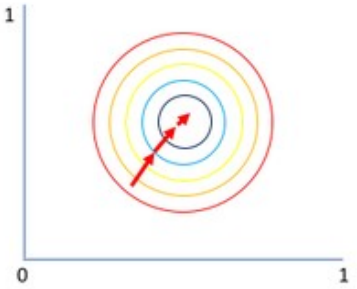
Why normalize ?

It is always necessary to **normalize each dimension of the dataset,** usually in the range [0,1] or [-1,1].

Otherwise, some dimensions might dominate the weight update gradient, eclipsing the information contained in the others



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Different approaches to rescaling

Centered: $X - \text{mean}$

Standardized: $\frac{X - \text{mean}}{\text{sd}}$

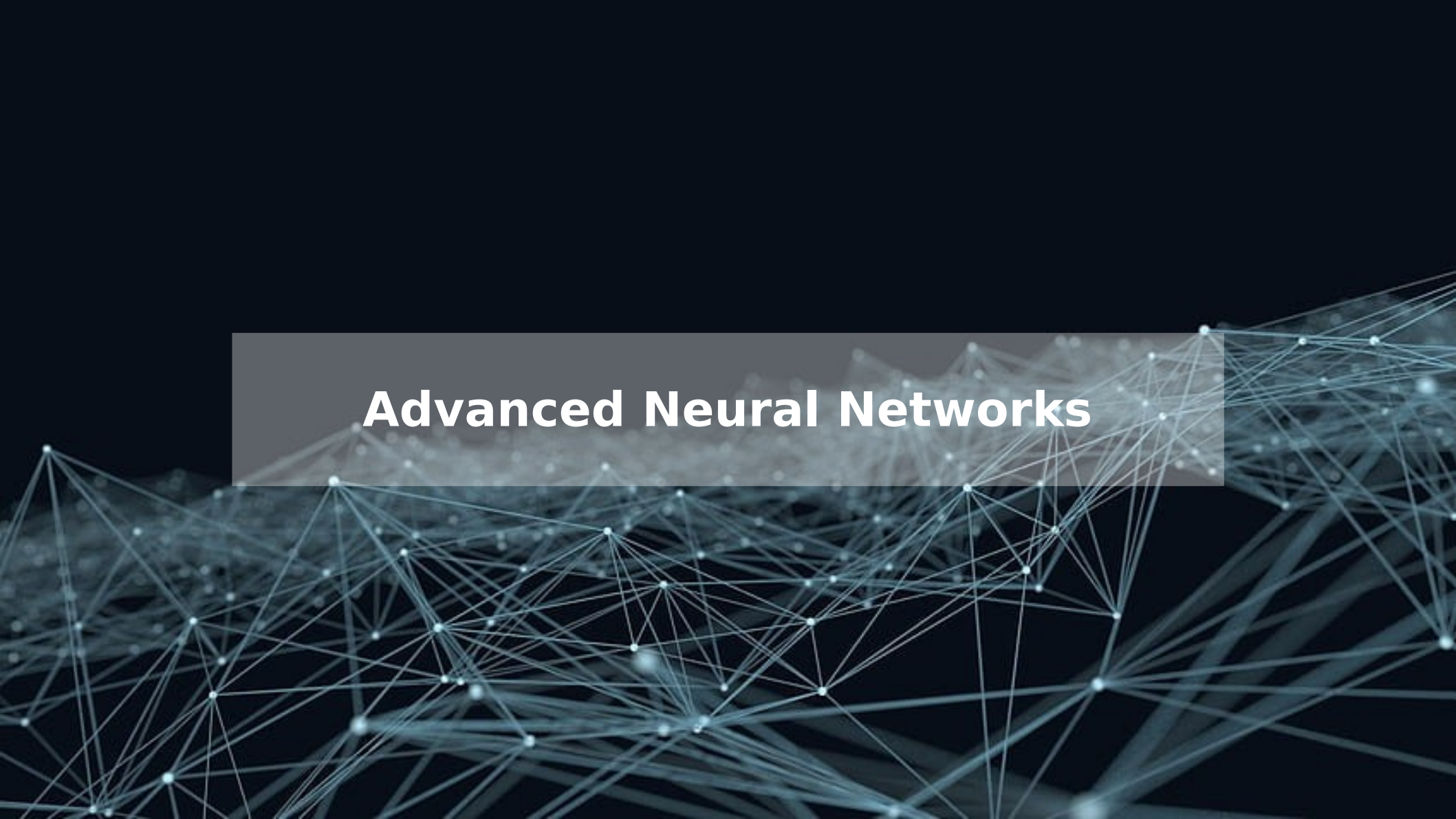
Normalized: $\frac{X - \min(X)}{\max(X) - \min(X)}$

WARNING !

All labeled data must be normalized with the same values (not only with the same equation)!

This include both the training and testing samples but also the data used for prediction at deployment time!

Advanced Neural Networks



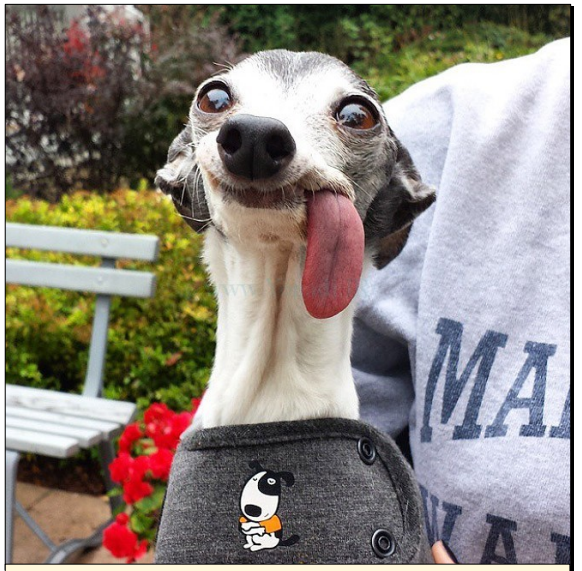
Neural Networks for images

Fully connected networks has shown one weakness

➡ **They are inefficient for handling images !**

- Images are highly dimensional (lots of pixels!)
- They have a very high degree of invariance
  (mainly translation but also luminosity, color, rotation, ...)

Classical ANN can deal with images by considering each pixel of an image as an individual input but it is STRONGLY inefficient.



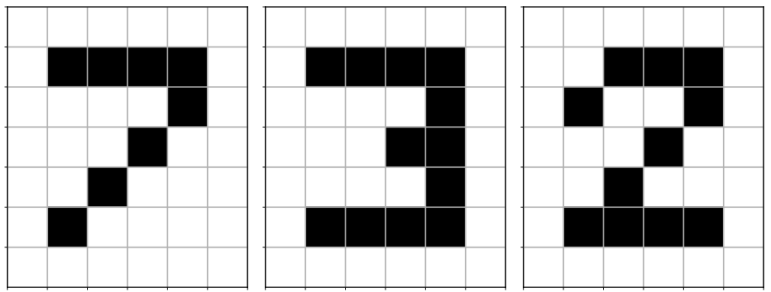
*A **highly dimensional** "dog" with ~0.5 Million pixels.*
Quite difficult to classify ...

Driven by the computer vision and pattern recognition community these issues have found a solution in the 90s with:

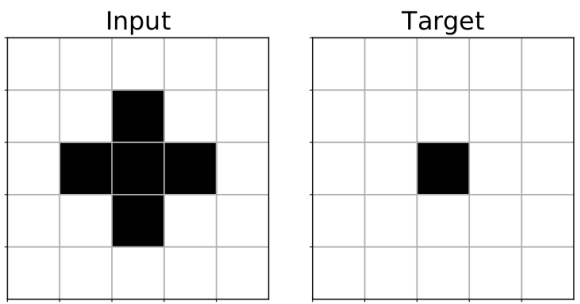
➡ **Convolutional Neural Networks** !

Spatially coherent information

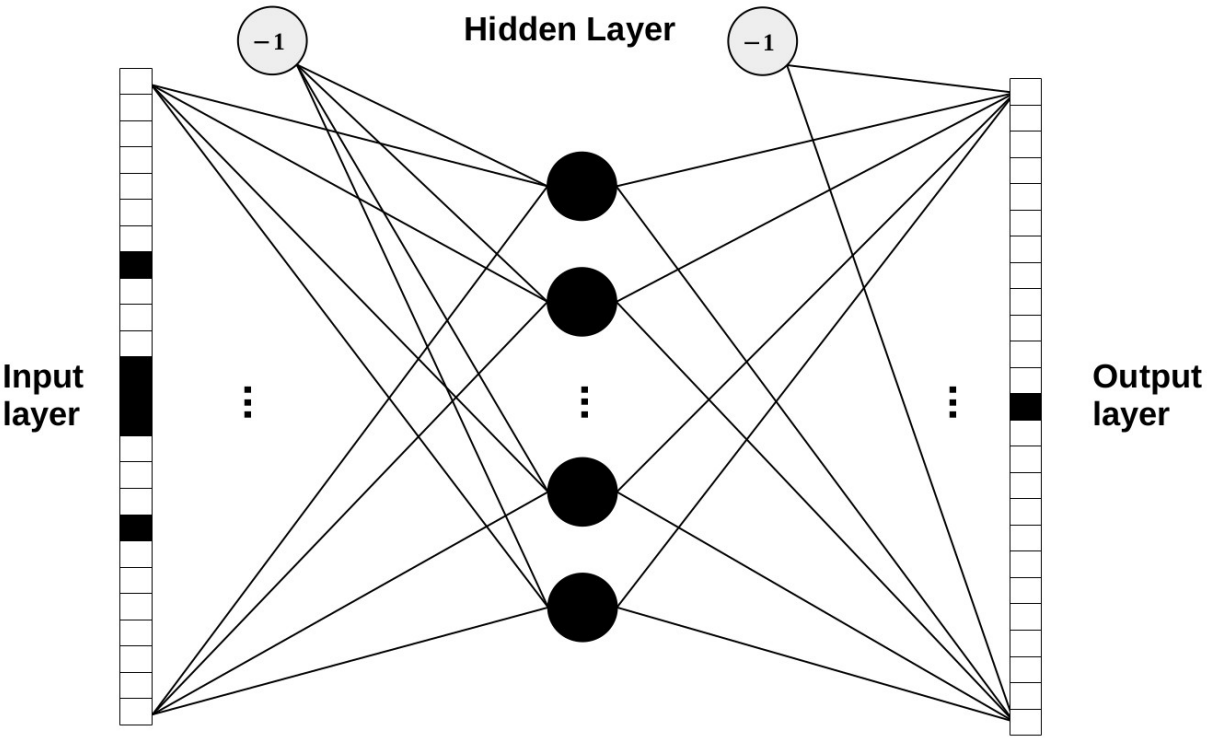
*Classical ANN can deal with images by considering each pixel of an image as an individual input but it is **STRONGLY inefficient**.*



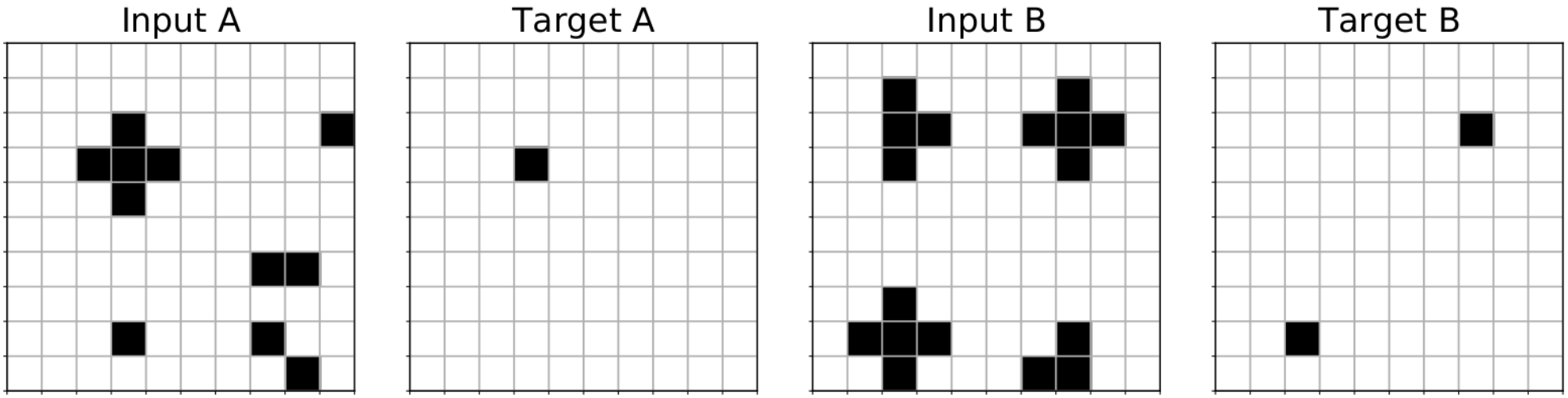
Simple digit representation as a 6x7 binary pixel image



Representation of a simple cross pattern on a 5 × 5 image as input, and the corresponding localization prediction on an equivalent size output image



Spatially coherent information : Pattern recognition

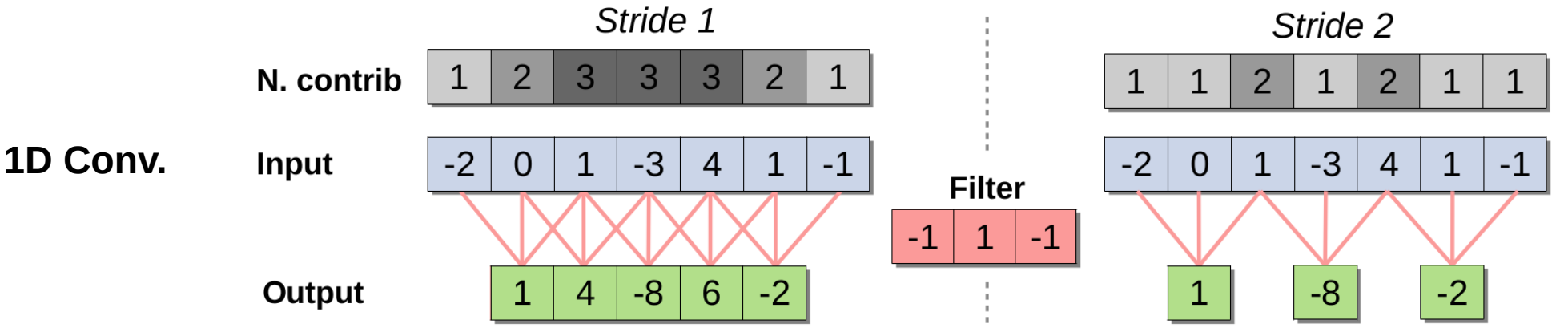


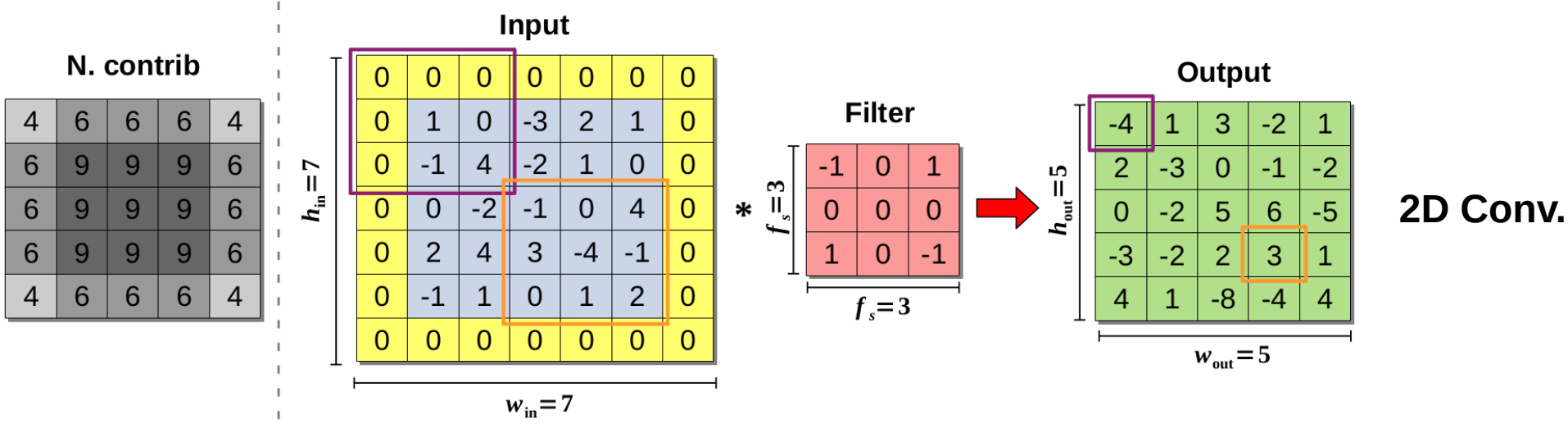
Looking for specific patterns can be automatized by scanning all the possible positions in the image.

In contrast, training a fully connected network to do the same task would require learning the presence or non-presence of the pattern at every possible position instead of learning the pattern once and only checking its presence at every position.

How to circumvent this behavior ? ➡ Use **Convolutional layers** !

Convolution filter

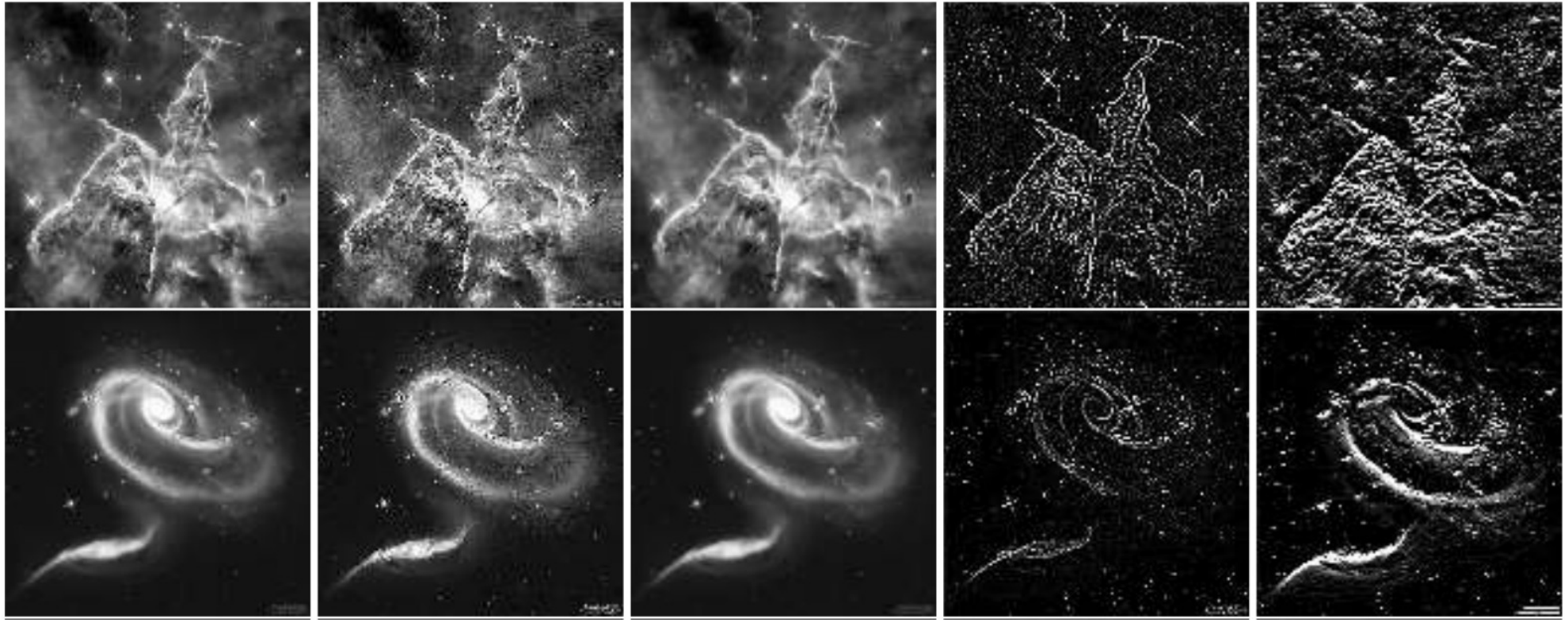




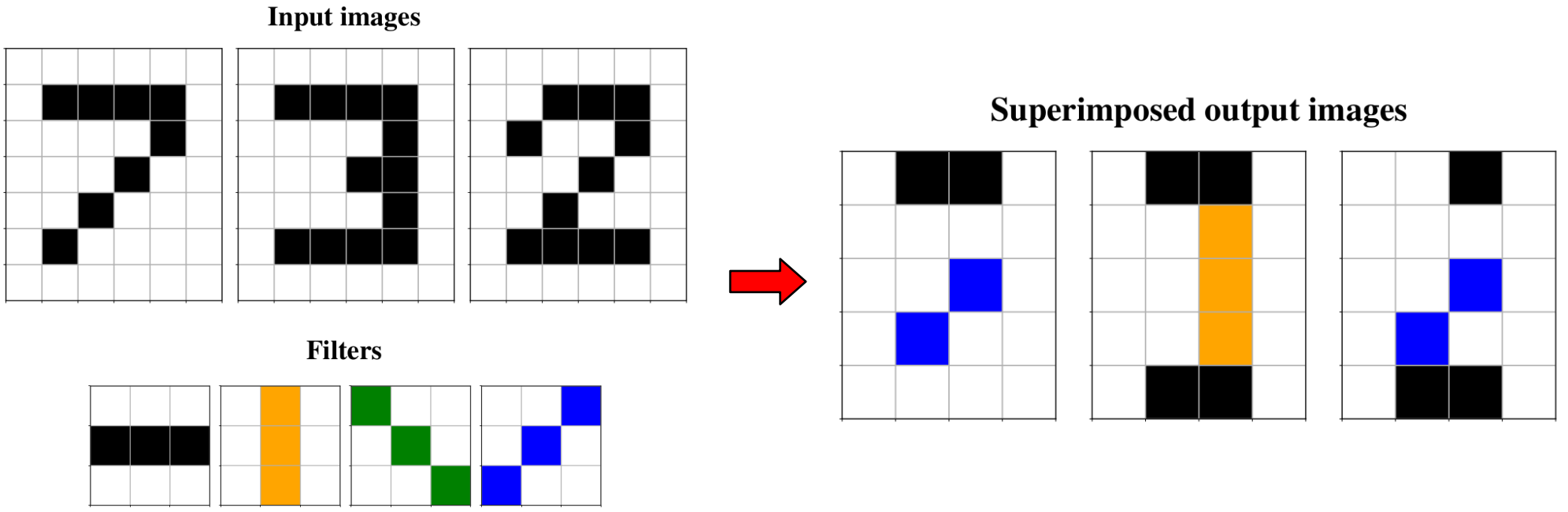
Stride 1 and Padding 1

Filter effect examples

No filter	Sharpen	Gaussian blur	Edge detector	Axis elevation
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$



Pattern recognition with several filters

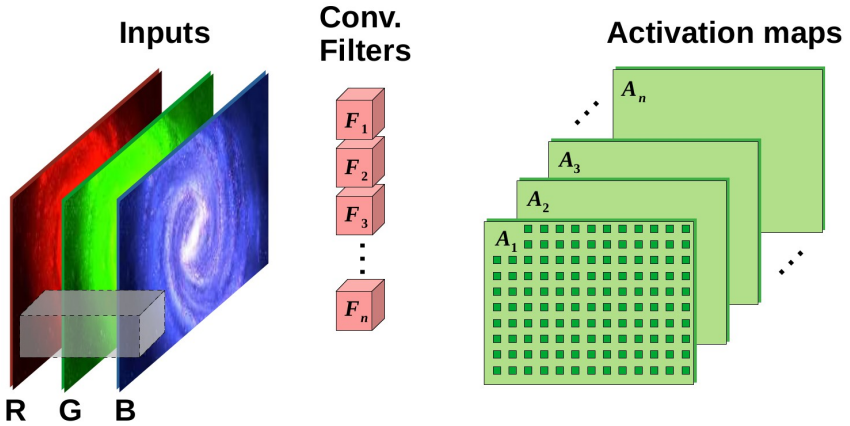


Each filter will produce its own activation map.

Combining the information from different activation maps allows to construct more complex patterns.

**Here the different activation maps are superimposed using color coding per filter*

Convolutional Neural Networks



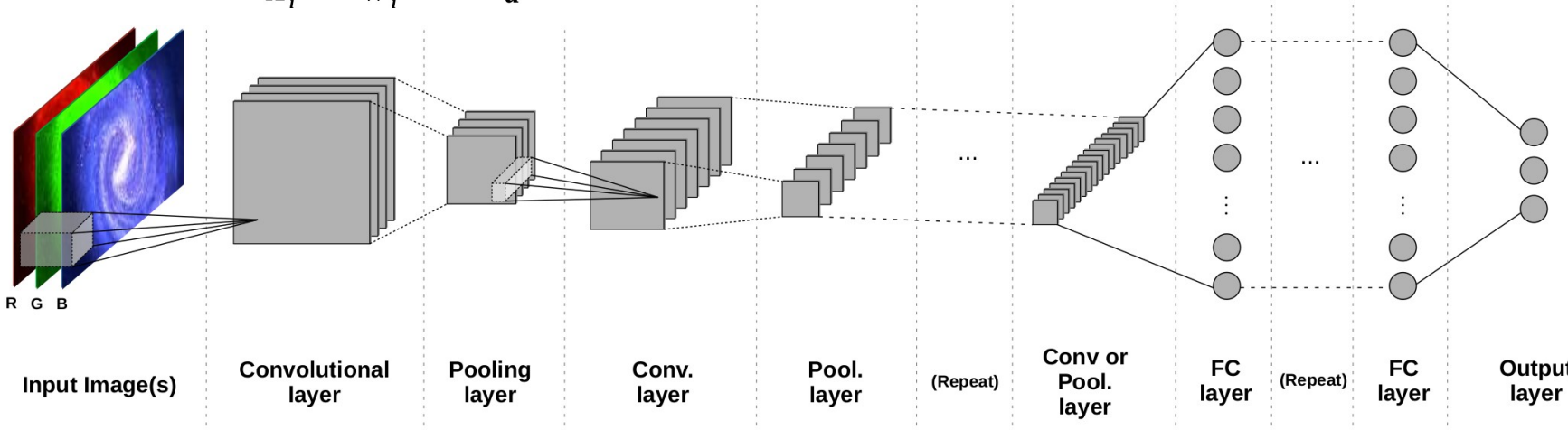
Each filter can be seen as a **single neuron** with one weight per input dimension in the filter.

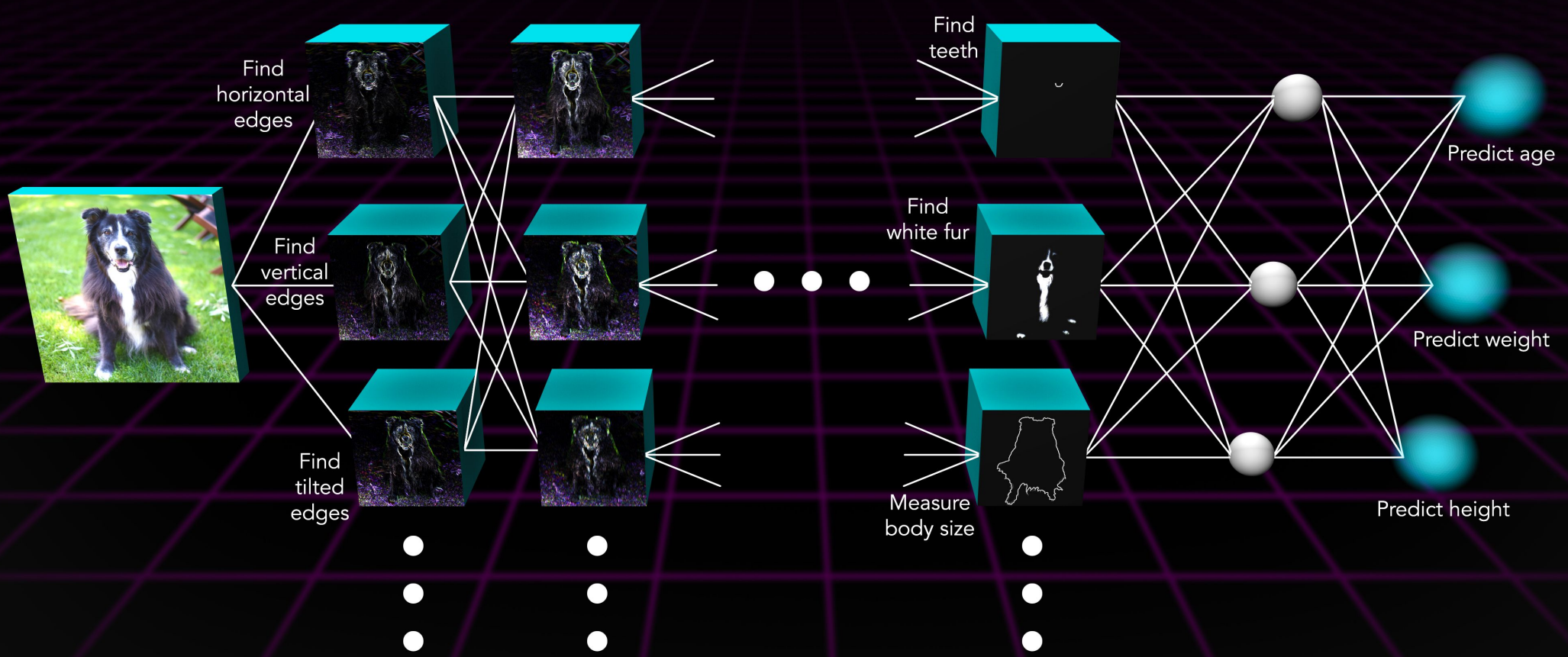
BUT, the same weights are used at every position and the outputs are independent.

➡ **Translational equivariance !**

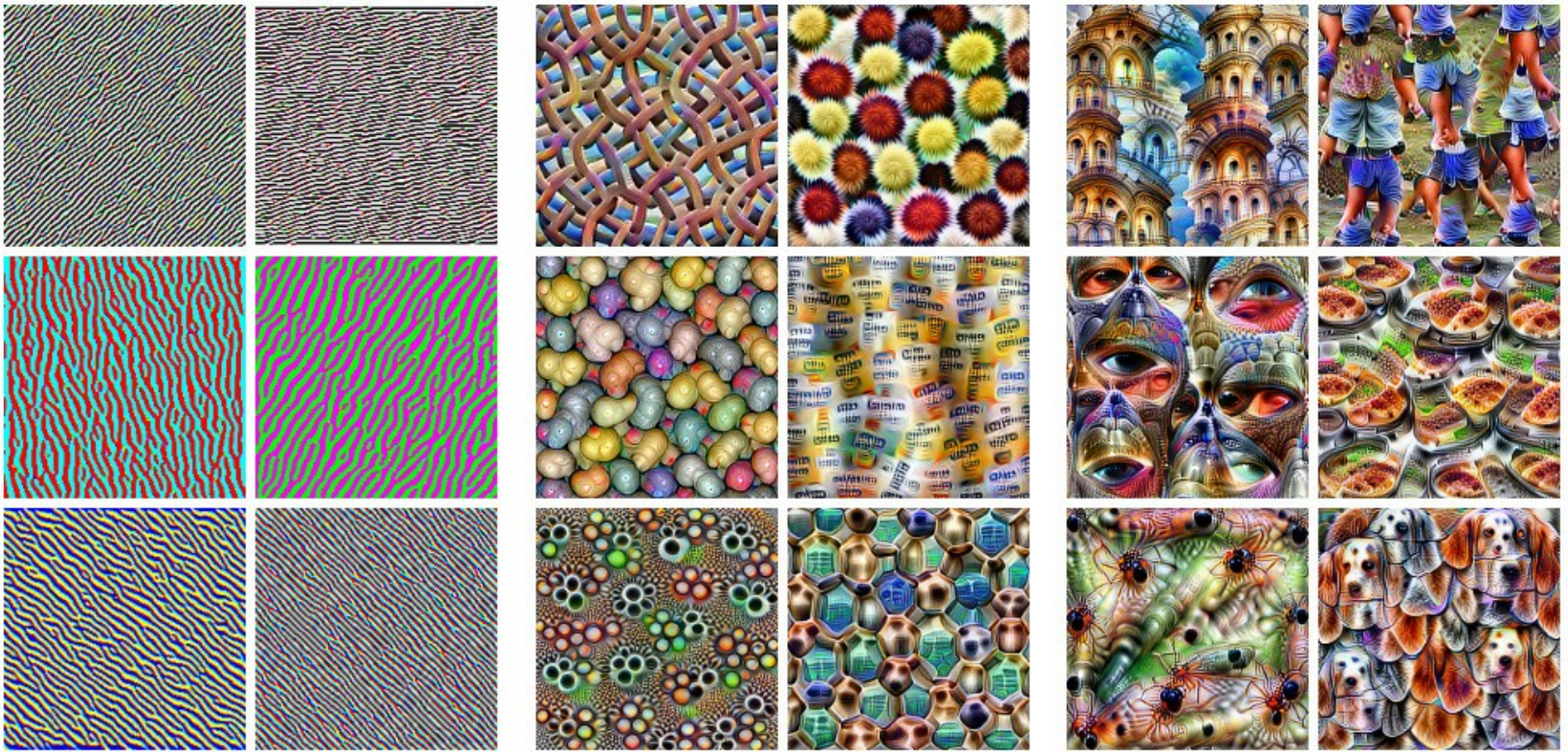
A network made of stacked convolutional layers can be tuned for Translation invariance.

$$g\left(\sum_i X_i \circ W_i\right) = a$$




Find horizontal edges
Find vertical edges
Find tilted edges
Find teeth
Find white fur
Measure body size
Predict age
Predict weight
Predict height

Examples of filter maximization



Edges (layer conv2d0)

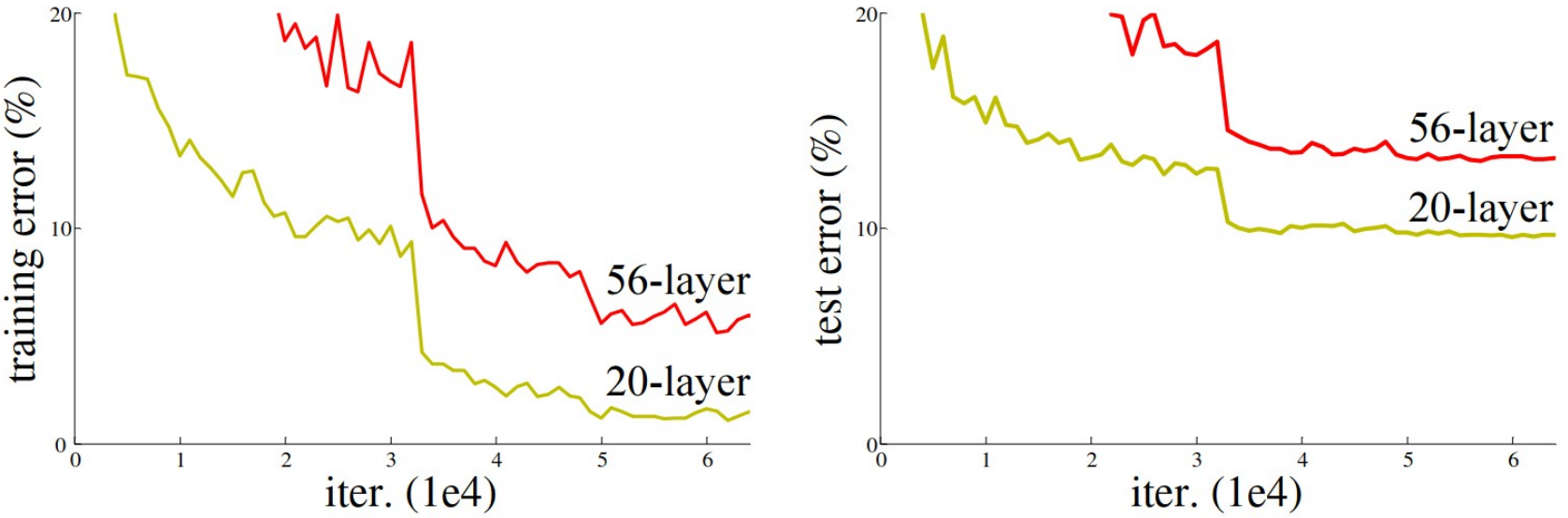
Patterns (layer mixed4a)

Objects (layers mixed4d & mixed4e)

Example of input images that maximize specific filters activation at different depth in a classification network

Vanishing Gradient Issue

From He et al. 2015



In principle, the deeper the network, the higher its expressivity should be as long as it is trained with enough data. However, it is nos the case in practice due to the gradient slowly getting smaller and smaller as it goes through more layers.

Several techniques can mitigate this issue to construct networks with hundreds of layers (e.g., changing the activation or having pass through connections between layers that are far away in the network).

The Rectified Linear Unit (ReLU)

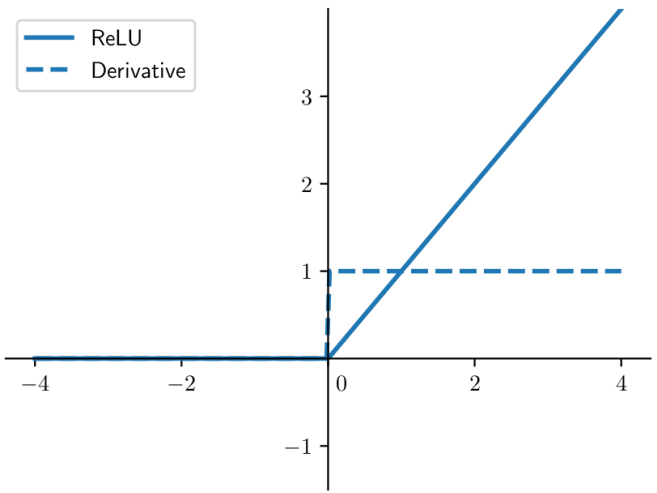
The ReLU (or its variance, the leaky-ReLU) has proven more efficient for CNN. It preserves a form of non-linearity and its constant derivative reduces **vanishing gradient problems**.

It is scale-invariant, and much faster to compute than other activation functions.

Using this activation, it becomes possible to construct much deeper networks.

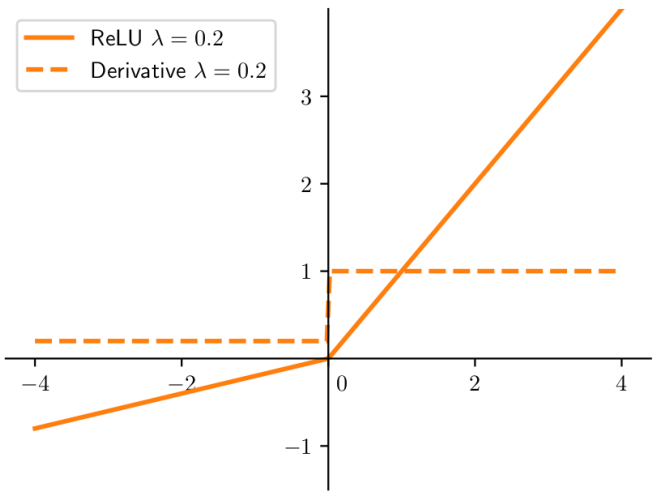
$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ 0 & \text{if} \quad h_j < 0 \end{cases}$$

or $$a_j = g(h_j) = \max(0, h_j)$$



$$a_j = g(h_j) = \begin{cases} h_j & \text{if} \quad h_j \geq 0 \\ \lambda h_j & \text{if} \quad h_j < 0 \end{cases}$$

or $$a_j = g(h_j) = \max(0, h_j) + \min(0, \lambda h_j)$$

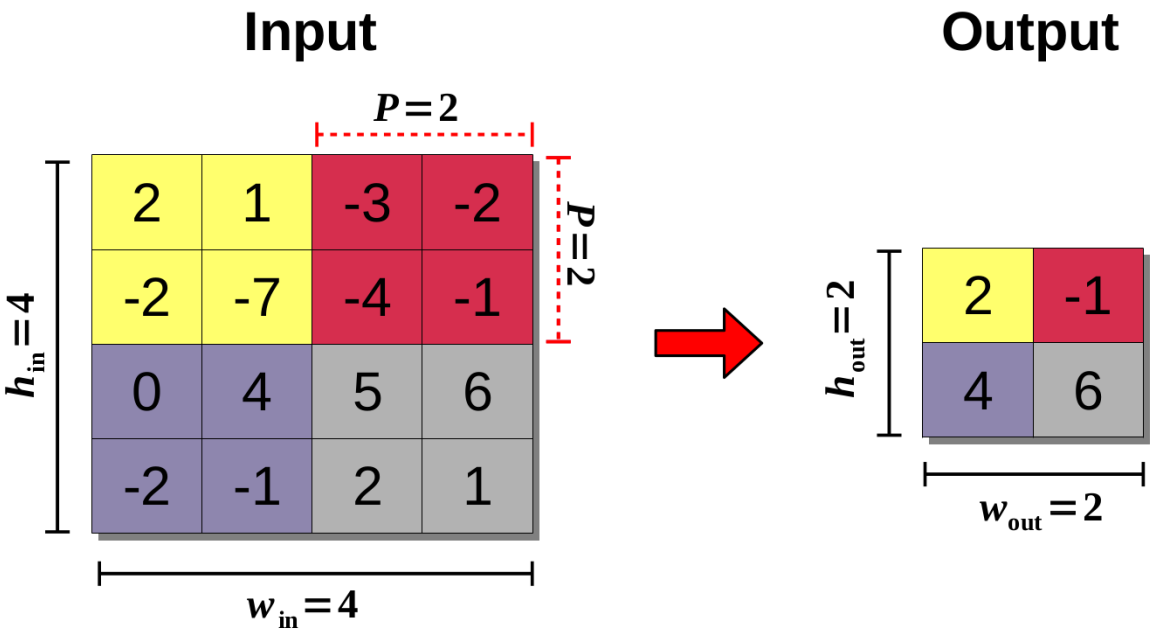


Dimensionality reduction: Pooling

A classical convolution operation is tuned to preserve the spatial dimensionality.
Still, it is most of the time necessary to **reduce the “image” size progressively**.

For classification tasks, the output layer is often reduced to a dense layer with a few neurons.

One way to reduce the spatial dimensionality is to use **Pooling layers** !

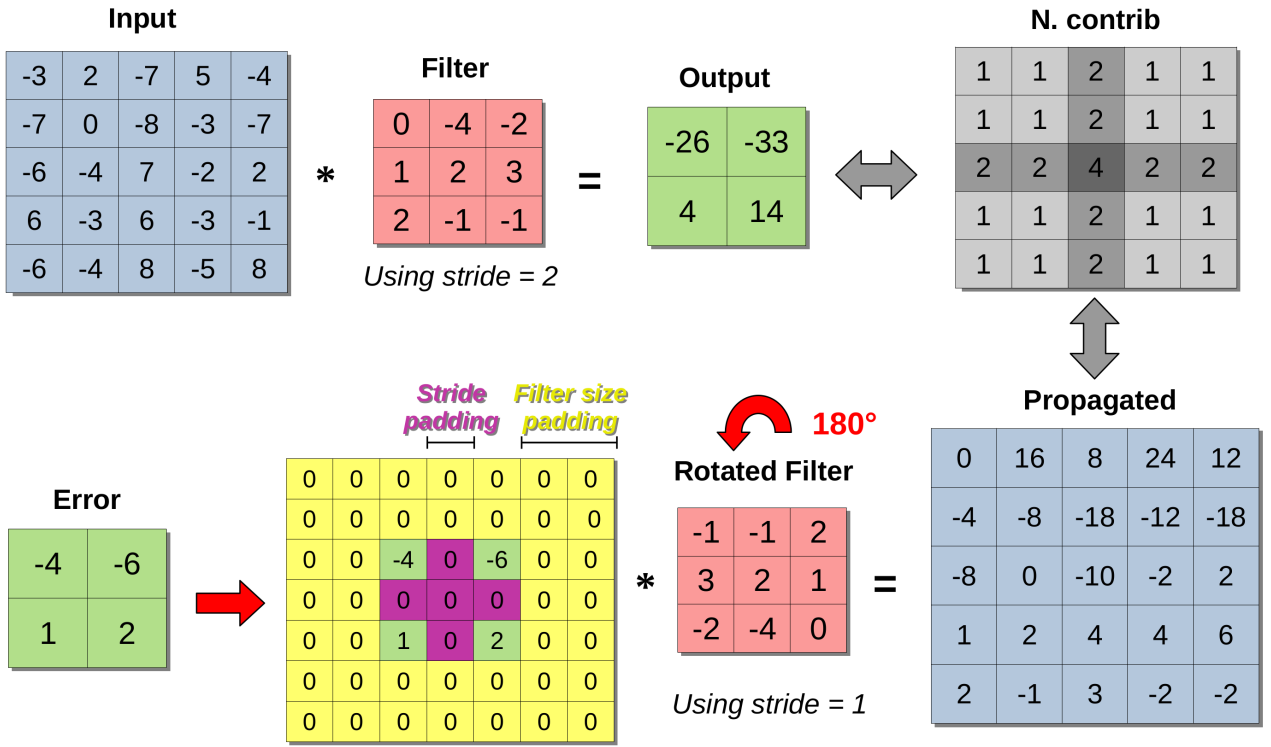


Different pooling methods exist, the most common being Max-Pooling and Average-Pooling. Pooling size can be modified, but most network architectures reduce spatial dimensions by a factor of two.

Learning the filters

Like fully connected layers, the **convolutional filters** can be learned using **backpropagation** of the error measured at the output layer.

The error is propagated using a transposed convolution operation, which can be expressed with a classical convolution operation using simple transformations on specific layer elements.



*Learning the convolutional filters is often considered to be the definition of "**Deep Learning**".*

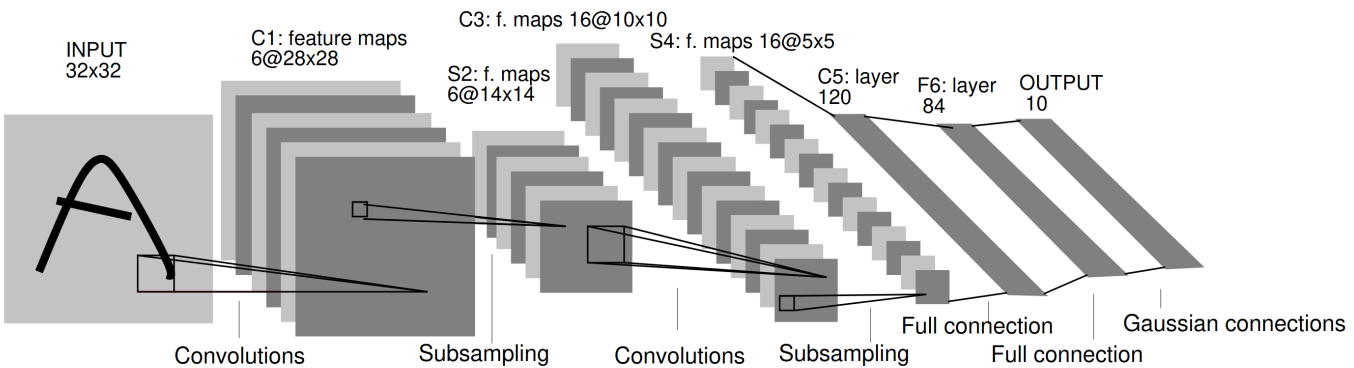
Simple example: MNIST

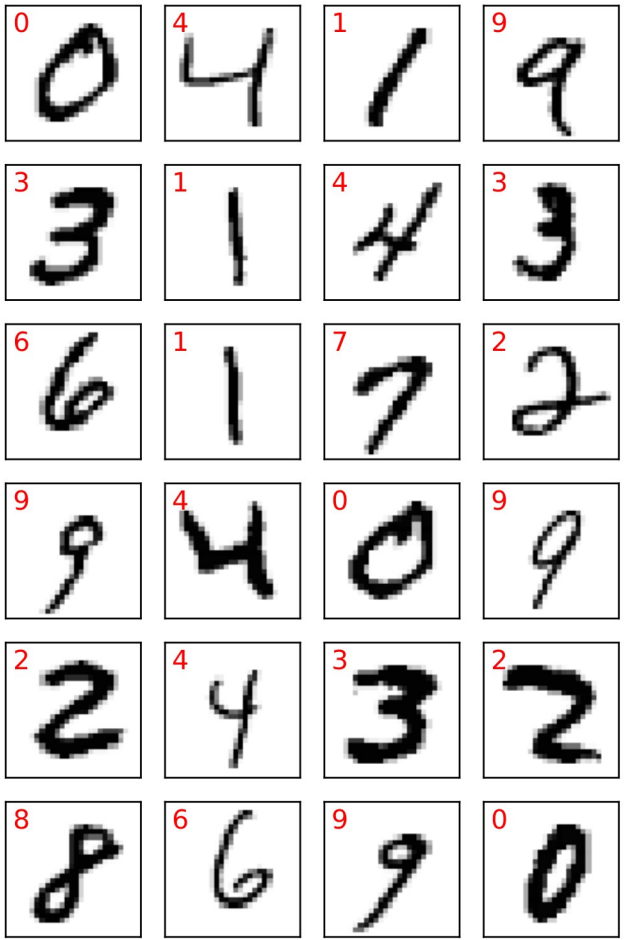
The well-known **MNIST** (Modified NIST's special dataset) dataset consists of handwritten digits from 500 different writers expressed as 28x28 grayscale images.

It is freely accessible in the form of a 60000 image training set, 10000 images validation, set and a 10000 image test set.

Very simple CNN architectures can achieve over **99.3 classification accuracy** on this dataset.

LeNet 5 network architecture (from LeCun et al. 1998):





Example results on MNIST

Actual \ Predicted	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	Recall
C0	976	0	1	0	0	0	1	1	1	0	99.6%
C1	0	1132	1	0	1	0	0	0	1	0	99.7%
C2	1	1	1027	0	1	0	0	1	1	0	99.5%
C3	0	0	1	1004	0	3	0	1	1	0	99.4%
C4	0	0	1	0	972	0	1	0	1	7	99.0%
C5	0	0	0	4	0	886	1	0	0	1	99.3%
C6	3	2	0	0	1	2	949	0	1	0	99.1%
C7	0	2	3	0	0	0	0	1020	1	2	99.2%
C8	0	0	1	1	0	1	1	1	968	1	99.4%
C9	0	0	0	0	3	1	0	4	0	1001	99.2%
Precision	99.6%	99.6%	99.2%	99.5%	99.4%	99.2%	99.6%	99.2%	99.3%	98.9%	99.35%

Obtained with the following architecture
=> I-28.28, C-6.5, P-2, C-16.5, P-2, C-48.3, D-1024_d0.5, D-256_d0.2, D10

All inner layers uses leaky ReLU activation (factor 0.1) and the output uses a Softmax activation.
The batch size is b=64, the learning rate is $\eta=2x10^{-4}$ with a small decay up to $\eta=1x10^{-4}$, and a momentum of $\alpha=0.9$.

The network is trained for 40 epochs.

1D CNN for spectra

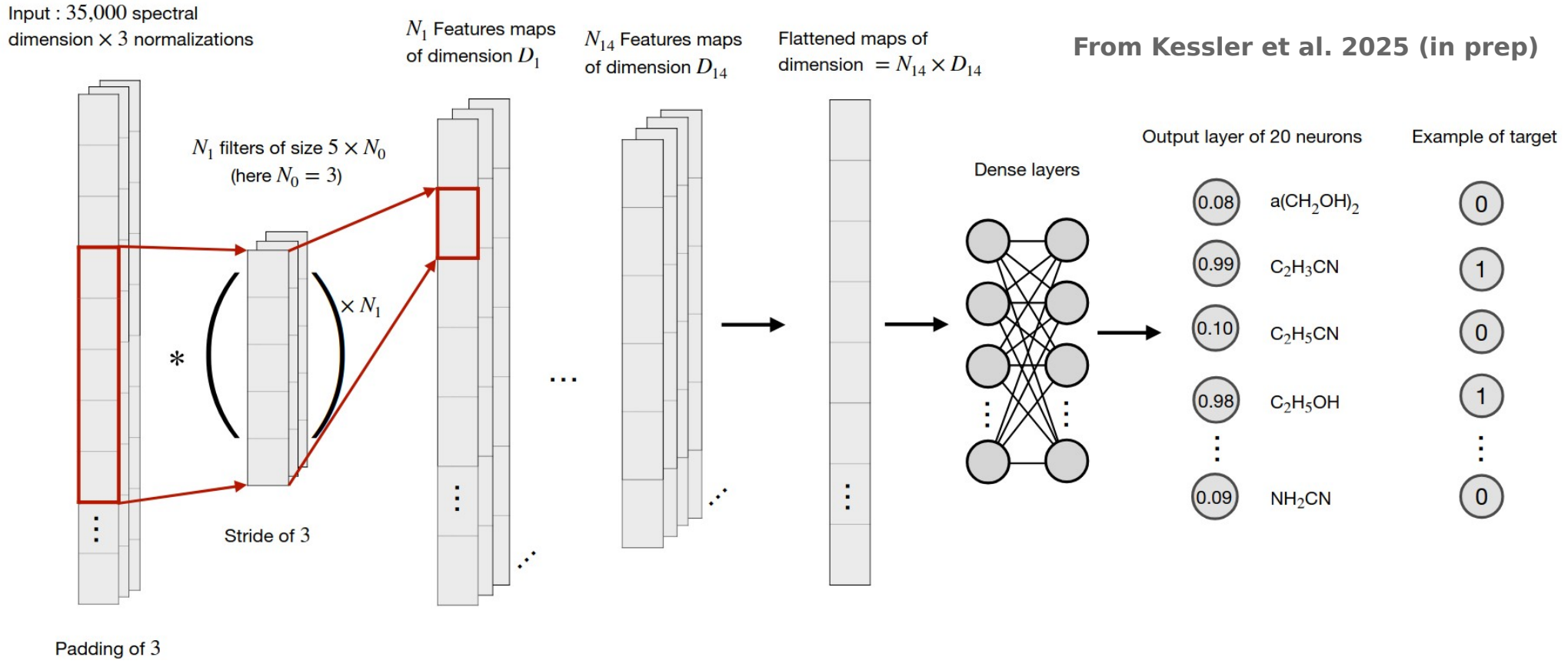
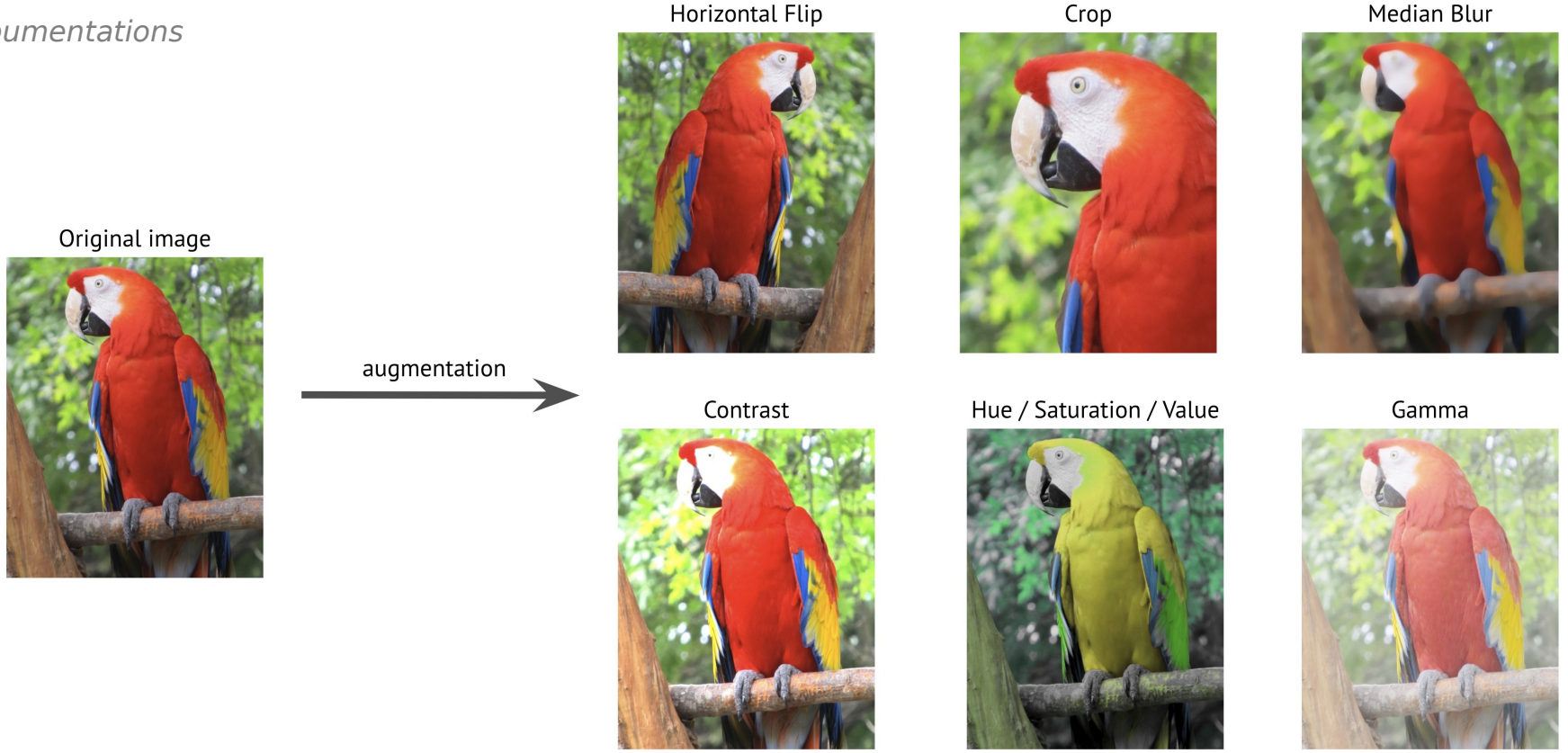


Fig. 5. Scheme of the ANN architecture. Filters are applied to the input data to convolve the information and produce features maps. This operation is done for each of the convolutional layers. Dense layers then combine the extracted features and learn how to label the spectra depending on the provided target. The output layer is composed of one neuron per class giving a score between 0 and 1 independent between each other.

Data augmentation

From Albumentations



Depending on the task, one can choose a set of transforms that do not alter the labeling corresponding to the image (here the class). This allow to increase the coverage of the feature space without the need of new data.

Augmentation does not solve all the dataset size issues, it usually **cannot create new contexts** nor it can generate features that are simply missing in the original dataset.

More advanced image classification

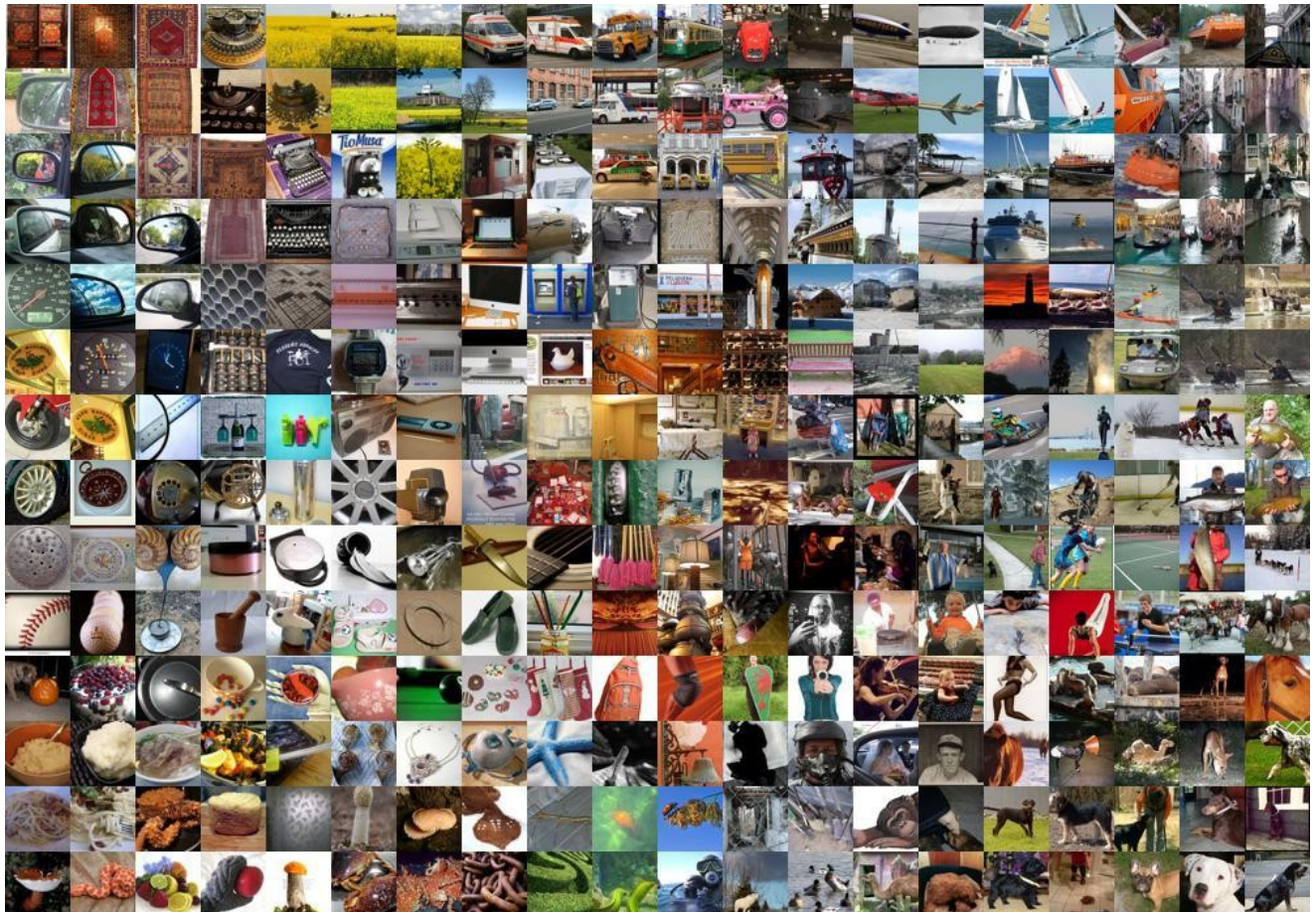
ASIRRA (Animal Species Image Recognition for Restricting Access): 25000 images, 50 % cats, 50 % dogs.

Typical of a **CAPTCHA** task or **HIP** (Human Interactive Proof) because the task is easy and quick for humans.

➡ Nowadays modern CNN tools have more than **90% accuracy** on ASIRRA !

We can generate standardize and augmented images from the dataset using dynamic image augmentation.

The canonical ImageNet-2012 dataset



ImageNet is a famous dataset as it is large and diverse enough to pre-train large models for a large variety of applications (classification, generation, detection, etc.)

- 1,281,167 training images (150 GB)
- 50,000 validation images
- 100,000 test images,
- Each image is associated with one of 1000 possible classes.

Images are of variable resolution with an average around 500x400.

The canonical ImageNet-2012 dataset

Result over ImageNet using a darknet-19 backbone, **91.7 Top5 Accuracy** over the 1000 classes, at a 448p resolution. The network run at 740 ips on an RTX 4090.

Explaining a model decision with occlusion analysis

An occlusion analysis is done by replacing a portion of the image by noise and measuring the impact it has on the model prediction. An occlusion map can be created by repeating moving the occlusion window over the original image.

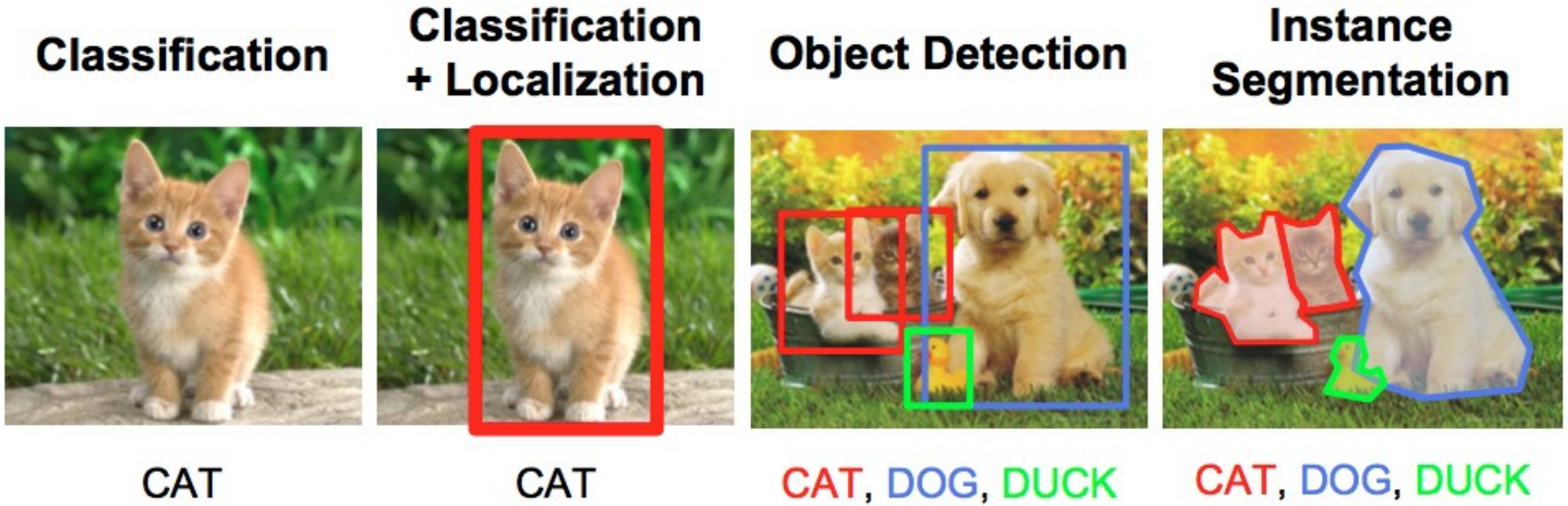
Allow to identify image features that contribute positively or negatively to a given prediction.

Here the occlusion map is done with the ImageNet trained model for the « malamute » class.

Object detection

“Object detection” types

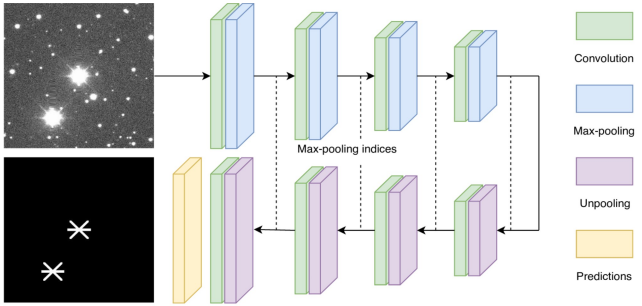
*Image from Stanford Deep Learning course cs224



CNN architectures for object detection

U-Nets

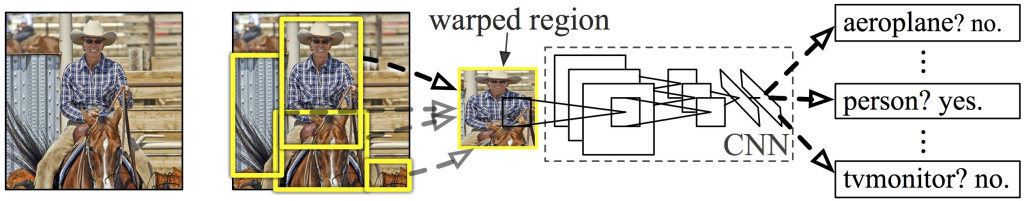
Pros: segmentation maps, shallow latent space, ...



Region-based

Methods : R-CNN (Fast and Faster), SPP-net, Mask R-CNN, …

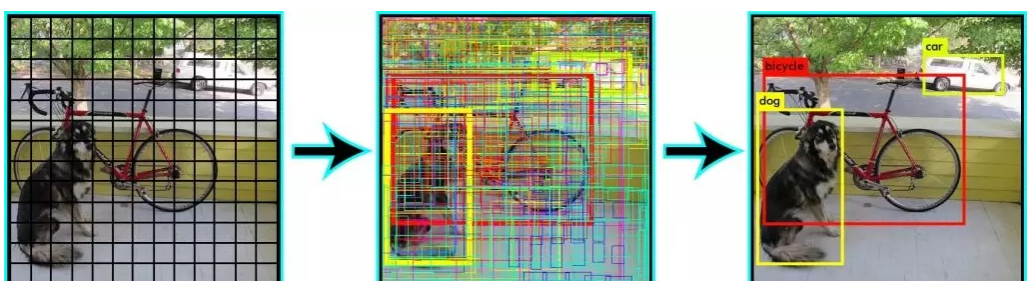
Pros: Best accuracy, ...



Regression-based

Methods : SSD (Single Shot Detector), YOLO (You Only Look Once), …

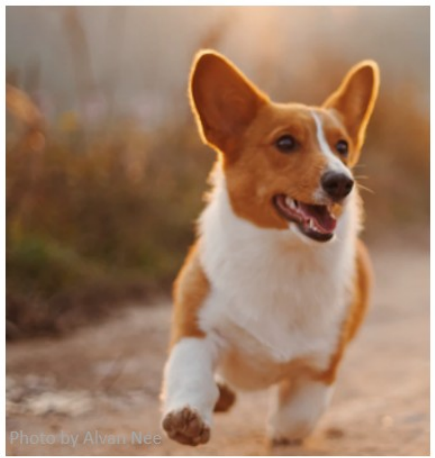
Pros: Very Fast, straightforward architecture,...



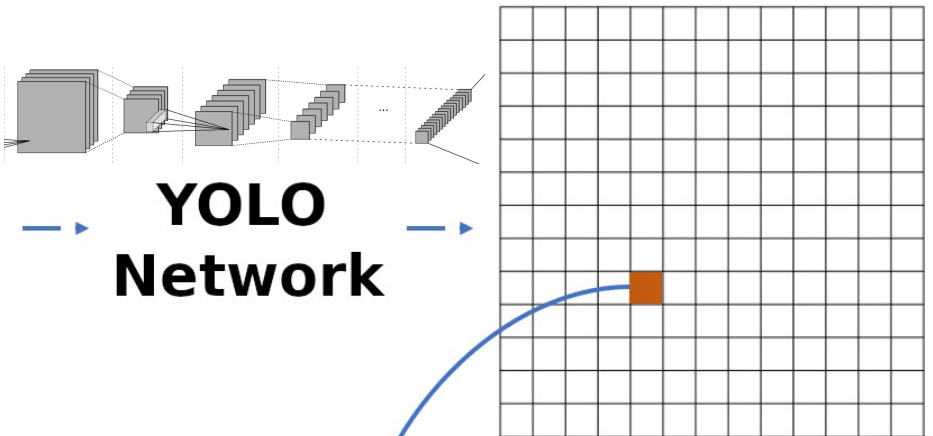
The You Only Look Once (YOLO) detector

Originally introduced in Redmon et al. 2015 (V1), 2016 (V2), 2018 (V3)

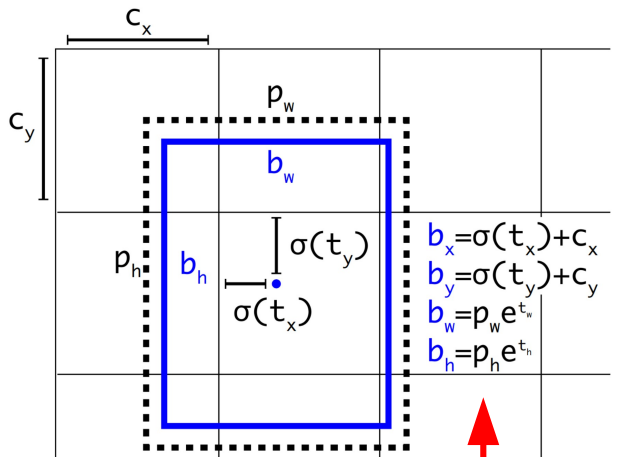
**Images from blog post and Redmon papers*

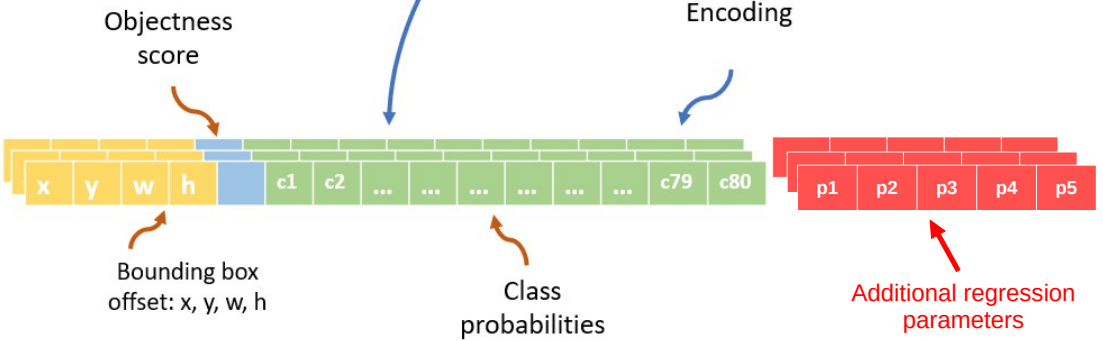


Pre-processing Image



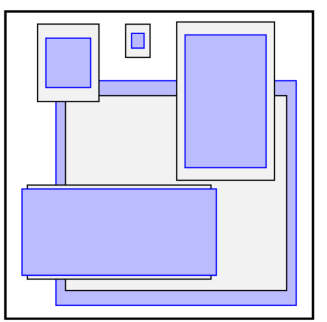
Encoding





p_w
p_h

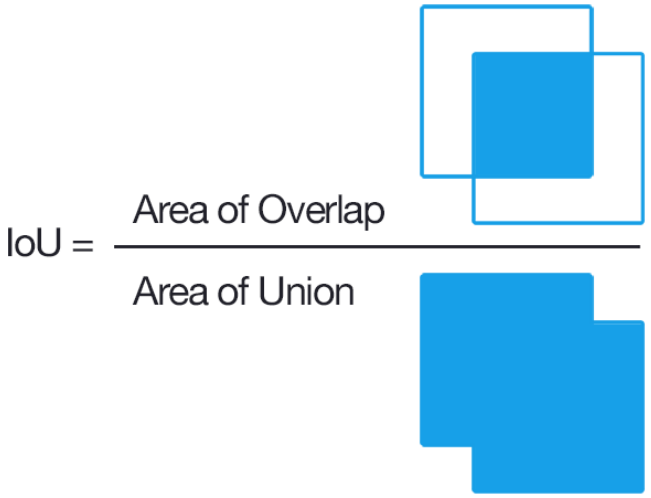
Box size priors

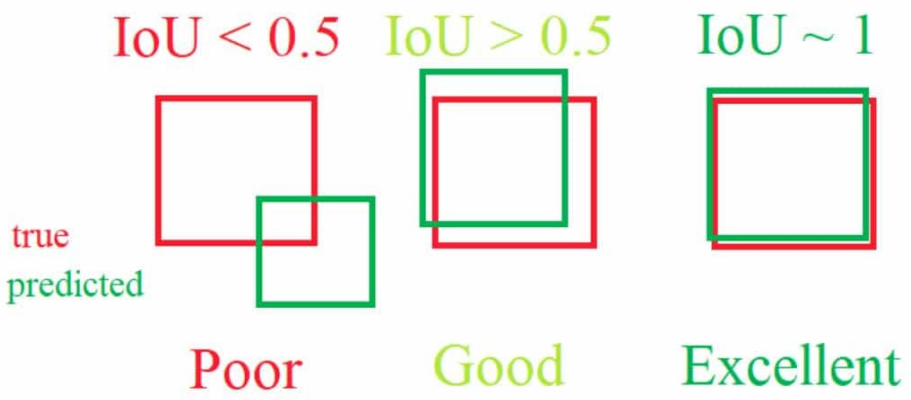


The last layer is conv. = the boxes « share » weights spatially.

The output is a 3D cube encoding all possible boxes on the output grid.

How to estimate box « correctness » ?

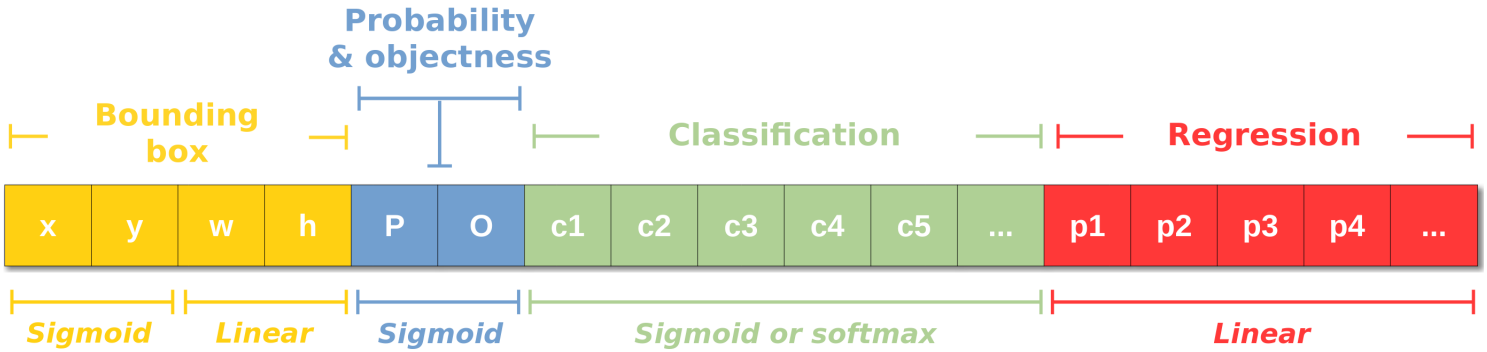




For each box the network will predict an “objectness” score defined as:

$$Pr(\text{object}) * IOU(b, \text{object}) = \sigma(t_o)$$

The YOLO activation/ association process

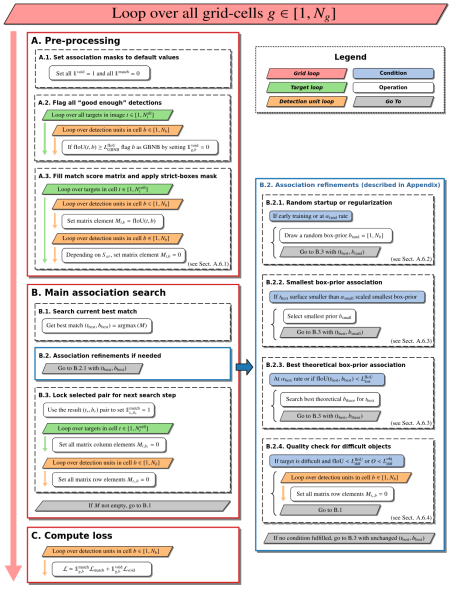


*All error terms uses MSE loss

Simplified Training procedure:

- For each target, find the corresponding grid element
- Search for the best predicted box in this grid element
- Exclude “good but not best” boxes (using an IoU threshold)
- **For the best box only :**
  - Update the position and size
  - Update objectness with target probability of 1
  - Update parameters and class according to targets
- For all the predictions with no associated target, update only the objectness with a target 0

*See the full association function in Cornu et al 2024



The YOLO activation / Loss function

$$\mathcal{L} = \sum_{i=0}^{N_g} \sum_{j=0}^{N_b} \mathbb{1}_{ij}^{\text{match}} \Big(\lambda_{\text{pos}} \left[(o_{ij}^x - \hat{o}_{ij}^x)^2 + (o_{ij}^y - \hat{o}_{ij}^y)^2 \right]$$
$$+\lambda_{\text{size}} \left[(o_{ij}^w - \hat{o}_{ij}^w)^2 + (o_{ij}^h - \hat{o}_{ij}^h)^2 \right]$$
$$+\lambda_{\text{class}} \mathbb{1}_{ij}^{C} \sum_{k}^{N_C} \left(-\hat{C}_{ij}^k \log(C_{ij}^k) \right)$$
$$+\lambda_{\text{param}} \mathbb{1}_{ij}^{p} \sum_{k}^{N_p} \gamma^k \left(p_{ij}^k - \hat{p}_{ij}^k \right)^2$$
$$+\lambda_{\text{prob}} \mathbb{1}_{ij}^{P} \left(P_{ij} - 1 \right)^2$$
$$+\lambda_{\text{obj}} \mathbb{1}_{ij}^{O} \left(O_{ij} - \text{fIoU}_{ij} \right)^2 \Big)$$

/!\ No error term for « Good but not best » boxes

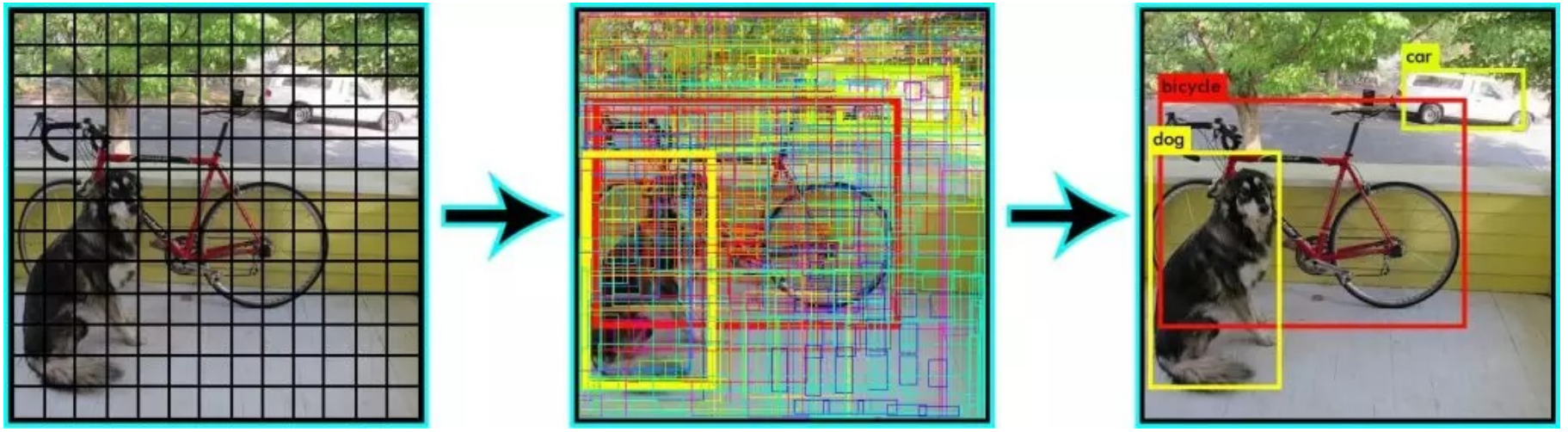
Only for the best box corresponding to each target

$$+ \sum_{i=0}^{N_g} \sum_{j=0}^{N_b} \mathbb{1}_{ij}^{\text{void}} \lambda_{\text{void}}^{j} \Big(\lambda_{\text{prob}} \left(P_{ij} - 0 \right)^2 + \lambda_{\text{obj}} \left(O_{ij} - 0 \right)^2 \Big).$$

Only for background (no match) predicted boxes

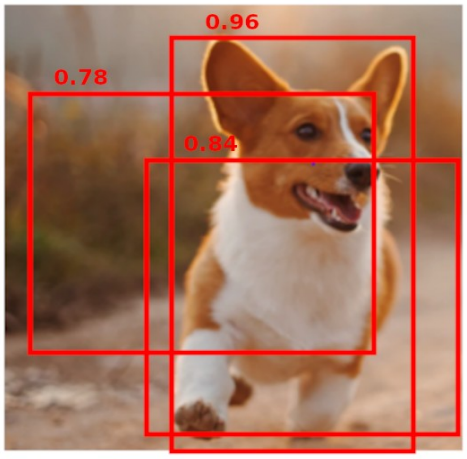
Post-processing: Non Max Suppression

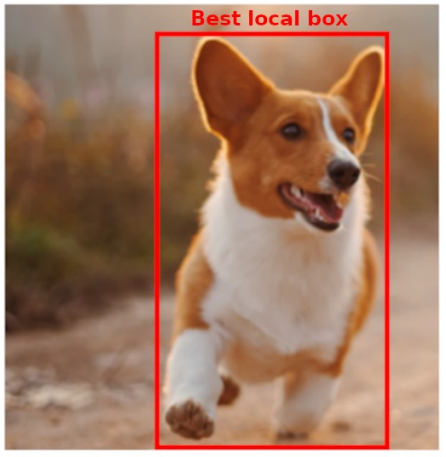
Due to the fixed size nature of its output layer, a YOLO network always predict all the possible boxes



1) Most probable boxes are kept using a threshold in objectness

2) NMS takes the most probable box and removes overlapping ones based on an IoU threshold. Repeat.



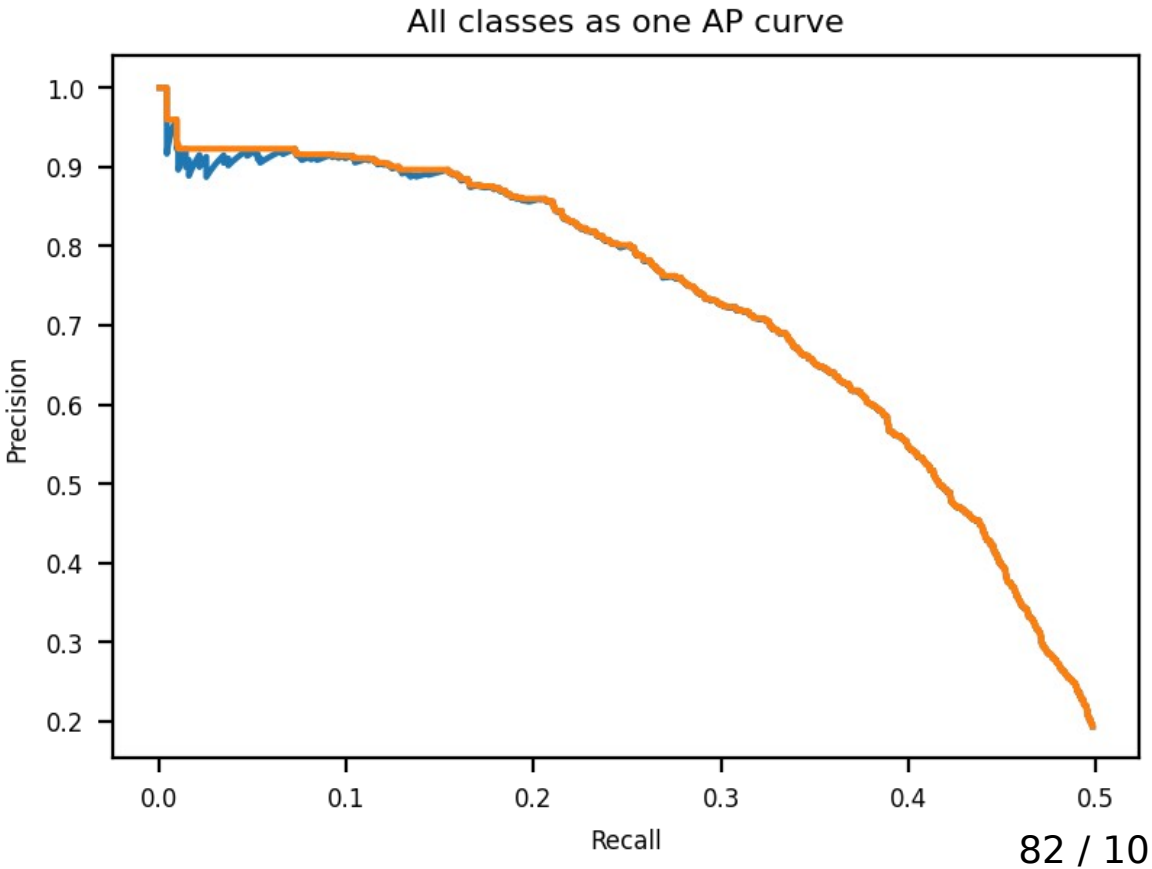


Detection performance metric

Match if IoU > V between prediction and target → Usually V = 0.5

The precision-recall curve

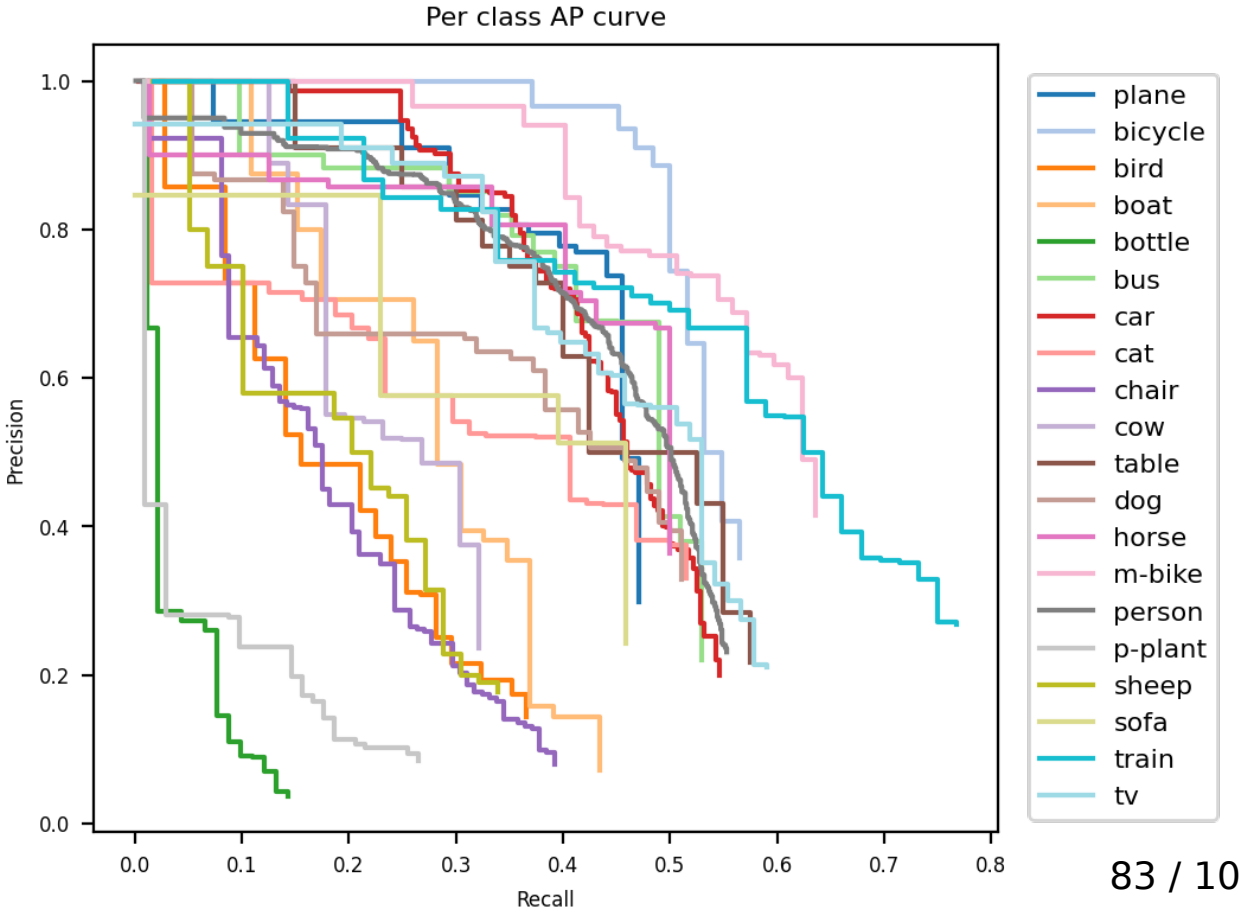
- Predictions are sorted by score
- For each data point, compute the precision and recall using all the predictions that are scored higher, using **IoU@0.5** and correct classification as “True” criteria
- The curve is then smoothed so it can only monotonically decrease (precision is always equal to its maximum value for any higher recall)
- The area under the curve defines the global metric called: **Averaged-Precision AP@50**



Per class AP and Mean AP

Each class gets its own Recall-Precision curve and AP score.
The final metric is the mean of the AP values.

Here mAP@50 ~ 32%

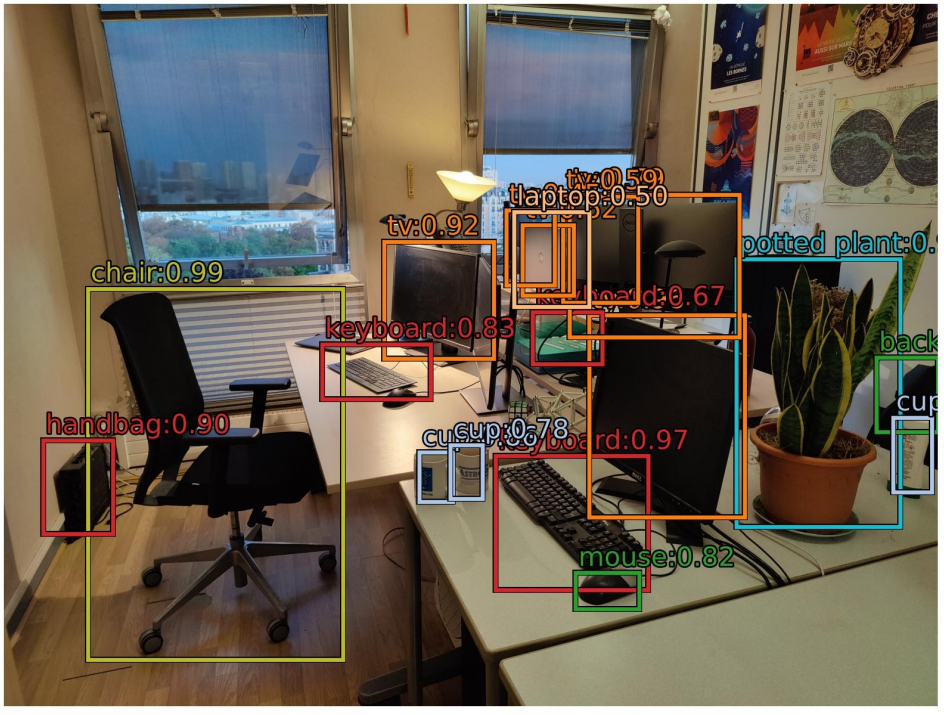


Warning: definition of AP and mAP might change depending on the challenge or dataset

More advanced YOLO detection

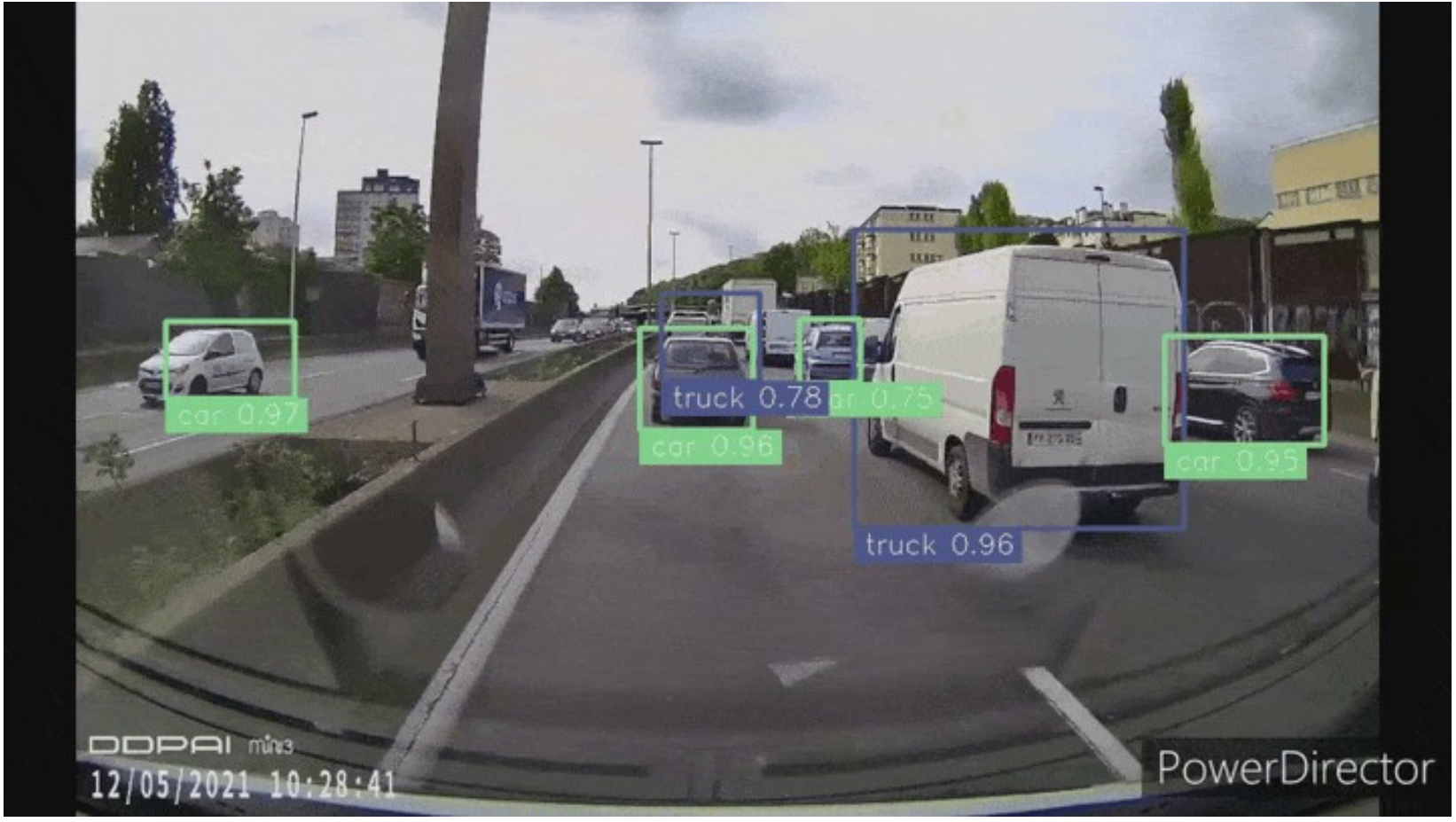
In the CIANNA git repository (https://github.com/Deyht/CIANNA), we provide several detection models using the custom YOLO-CIANNA method with backbone very similar to the darknet-19 architecture and applied to several datasets.

The model is pre-trained on ImageNet and then trained on the PASCAL and COCO datasets independently.



Results from a YOLO network trained on COCO

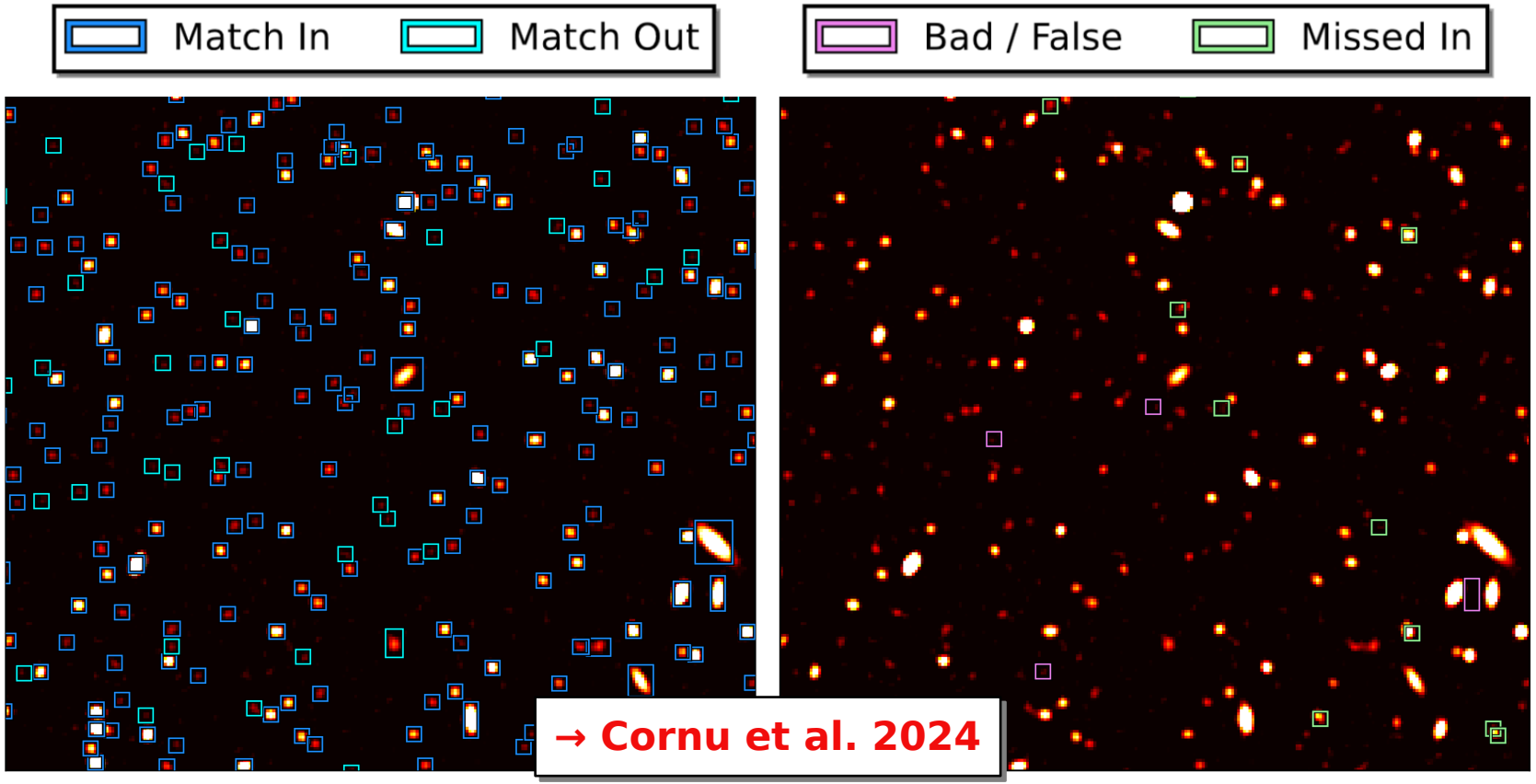
Result using a YOLO detector with a darknet-19 “like” backbone. 40.1 mAP over the 1000 classes, at a 416p resolution. The network run at 690 ips on an RTX 4090.



Application to galaxy detection - SKAO SDC1

Simulated continuum observation at 560 MHz for 1000h integration time with SKA

Objective: detect (Ra, Dec) and characterize (Flux, Bmaj, Bmin, PA, ...) the sources

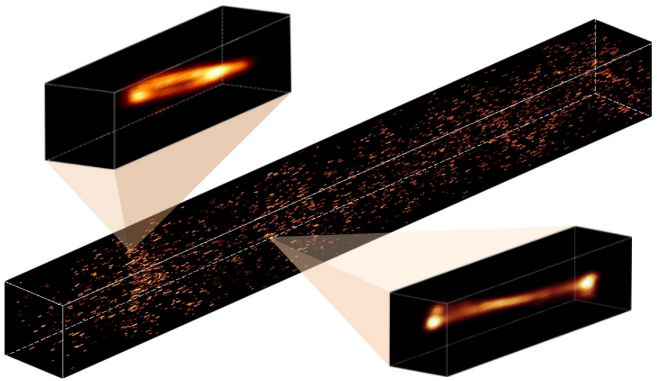


Fast (~130 Mpix/s - RTX 4090) and light network (17 layers, 13M param)

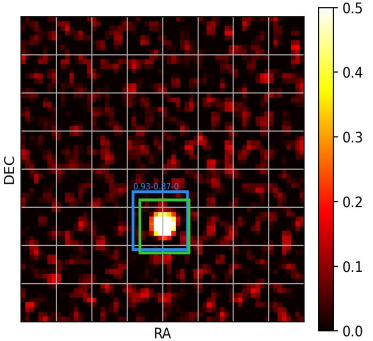
Application to galaxy detection - SKAO SDC2

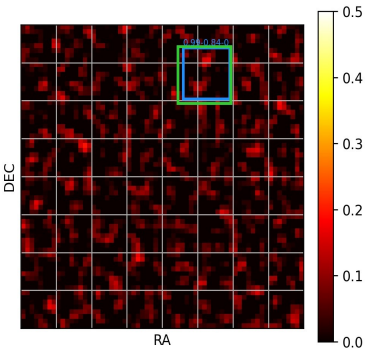
Simulated HI cube for 2000h integration time with SKA

Objective: detect (Ra, Dec, Freq) and characterize (Line flux, HI size, PA, I) the sources

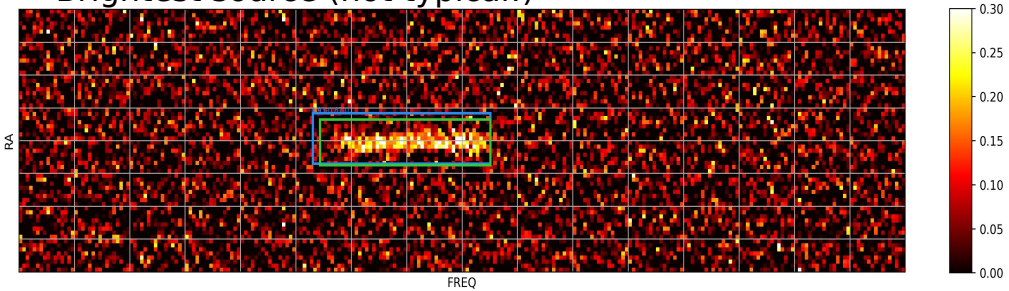


True boxes vs Predicted boxes

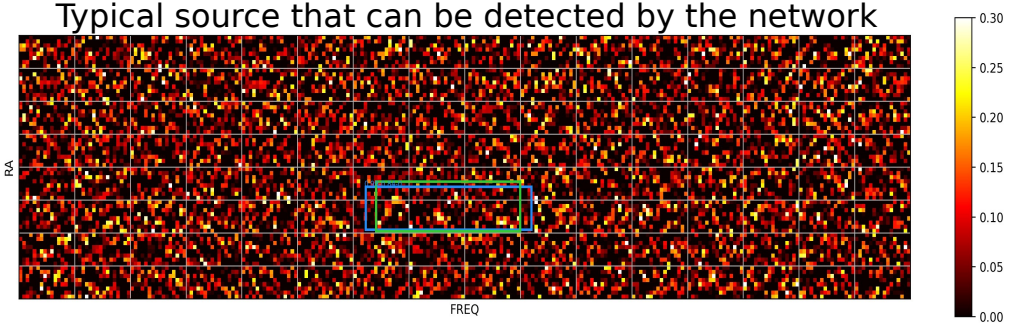




Brightest source (not typical!)



Typical source that can be detected by the network



*averaged over 20 channels in FREQ and 20 pixels in DEC respectively

→ Harley et al. 2023
→ Cornu et al. In prep

Fully 3D, fast (~120 Mpix/s - RTX 4090) and light network (23 layers, 4M param)

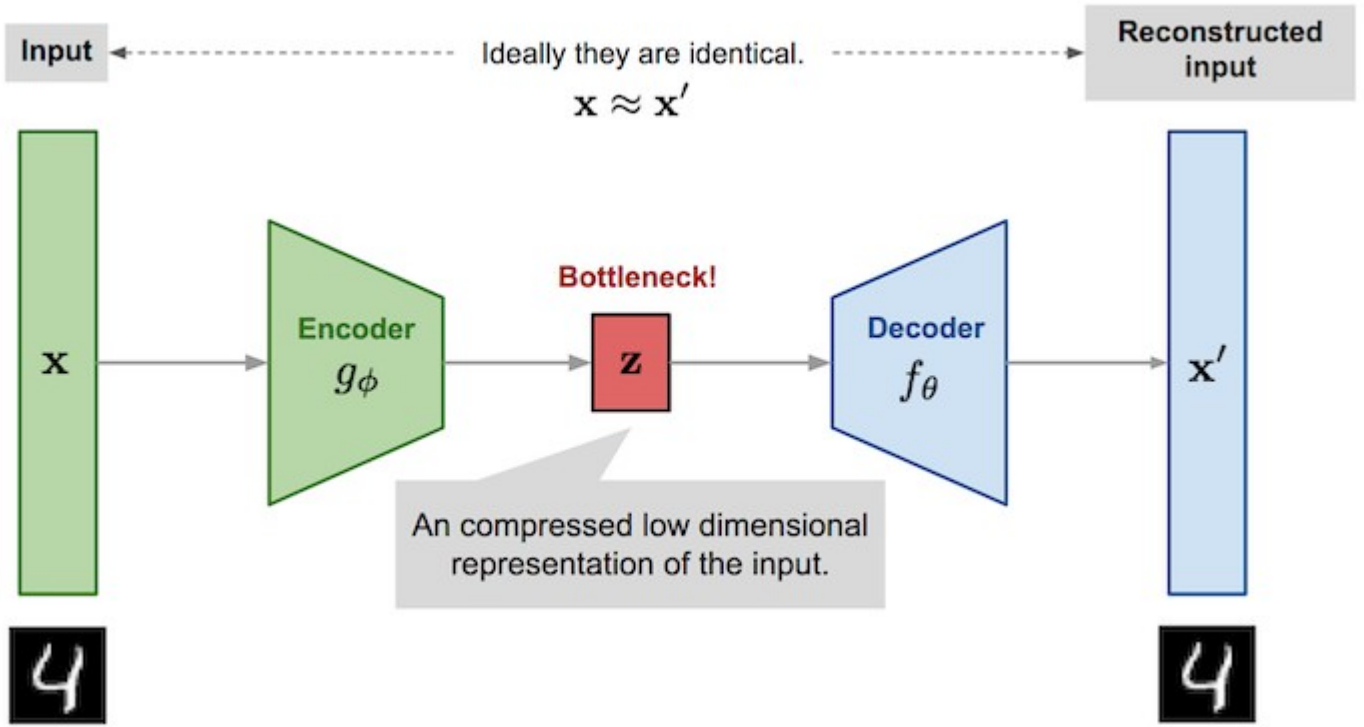
Want to learn more ?
Spe: Numerical Methods

- Code a Multi Layer Perceptron from scratch
- Use dedicated framework to build advanced network
- Understand all the hyper-parameters of a network
- See more complex network structures (normalization layers, residual blocks, etc.)
- More hindsight on how to adapt a network architecture to a given problem
- A glimpse on trending architectures like generative models (adversarial network and diffusion models)

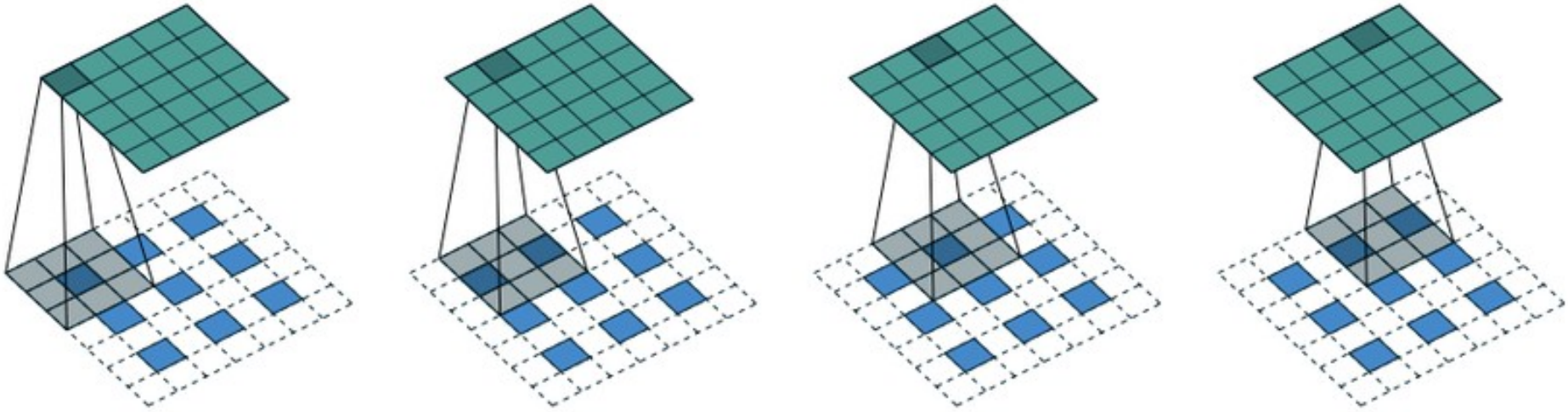
Self-supervised training with Auto-Encoders

AE networks aims to reconstruct their input (self-supervied learning). They are composed of an **encoder** that compresses the information up to a **latent space,** and of a **decoder** that reconstructs the input from the latent representation.

AE are used for denoising, data compression, and even data generation.



Transposed convolution for upsampling



To upsample the activation maps of a given layer, the common approach is transposed convolution.

By adding an internal padding between elements of the activation maps and then doing a classical convolution, we obtain output maps that are larger than the input ones.

→ *This is the operation used when backpropagating the gradient through layers with a stride >1*

This structure allows the construction of expanding decodeurs that are employed in many architectures.

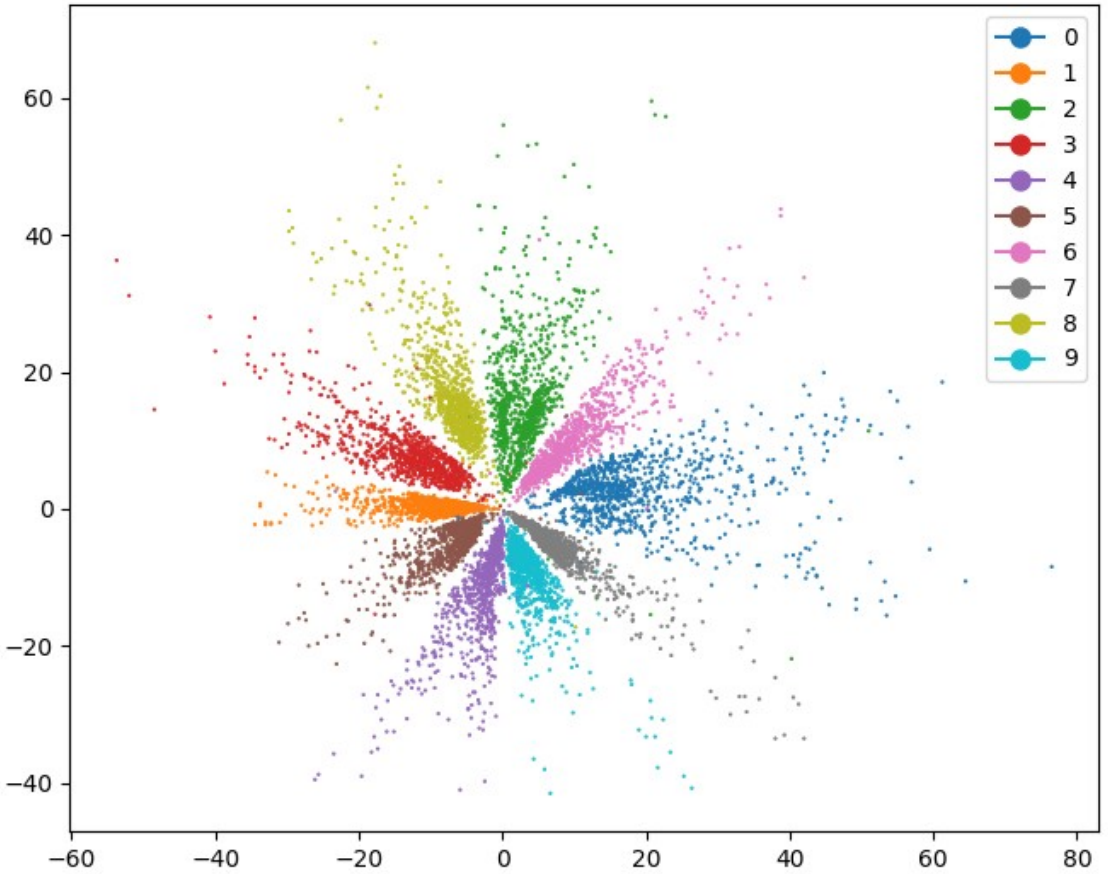
Visualizing the latent space

After the AE is trained, we can use the encoder on labeled data to visualize their distribution **in the latent space**.

AE are very similar to PCA in their principle but can separate more complex objects thank to the learned non-linearity allowed by the network structure.

AE can be used for data generation by selecting new points in the latent space and using the decoder to generated the associated image.

Most modern applications use Variational AE in order to produce more compact and distinct distributions in the latent space, which also improve the realism of generated images when drawing in between class distribution in the latent space.



Generated digits from an MNIST AutoEncoder

Generated using a regular sampling in polar coordinates in the latent space from last slide.

Practical work:

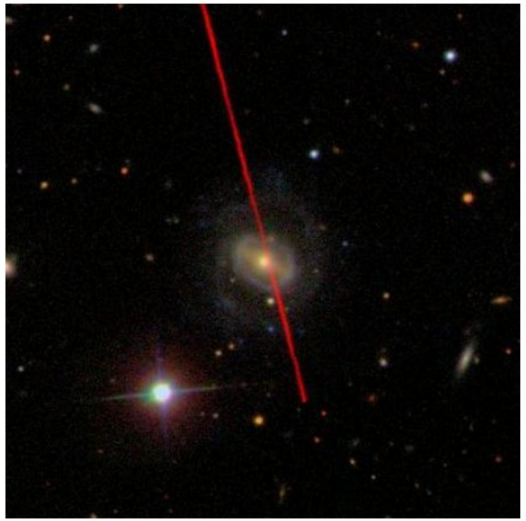
Using the provided starting notebook, create an AE architecture on the MNIST dataset.

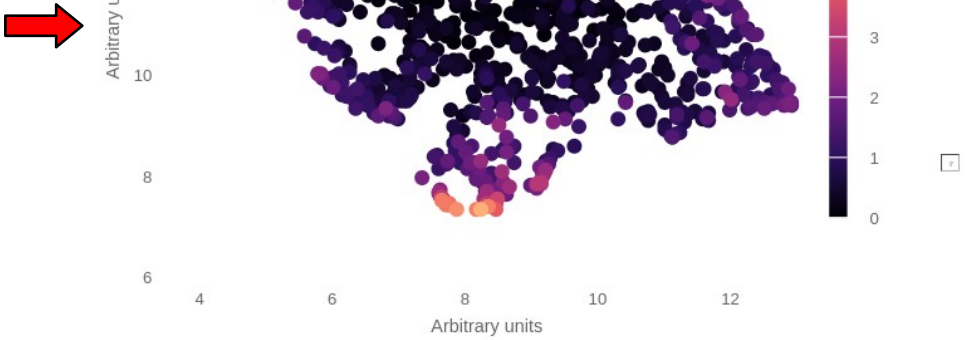
You will evaluate the performance of your reconstruction, visualize the latent space, and use it to generated new digits like above.

Advanced:

→ How could you reduce class confusion in the latent space ?

Semi-supervised recommendation rating
Example with Astronomaly





0 — Index in list

701930 — Object name

HOW INTERESTING IS THIS OBJECT?

0 | 1 | 2 | 3 | 4 | 5

Raw anomaly s... — Scoring method to sort by

Show unlabelled objects first

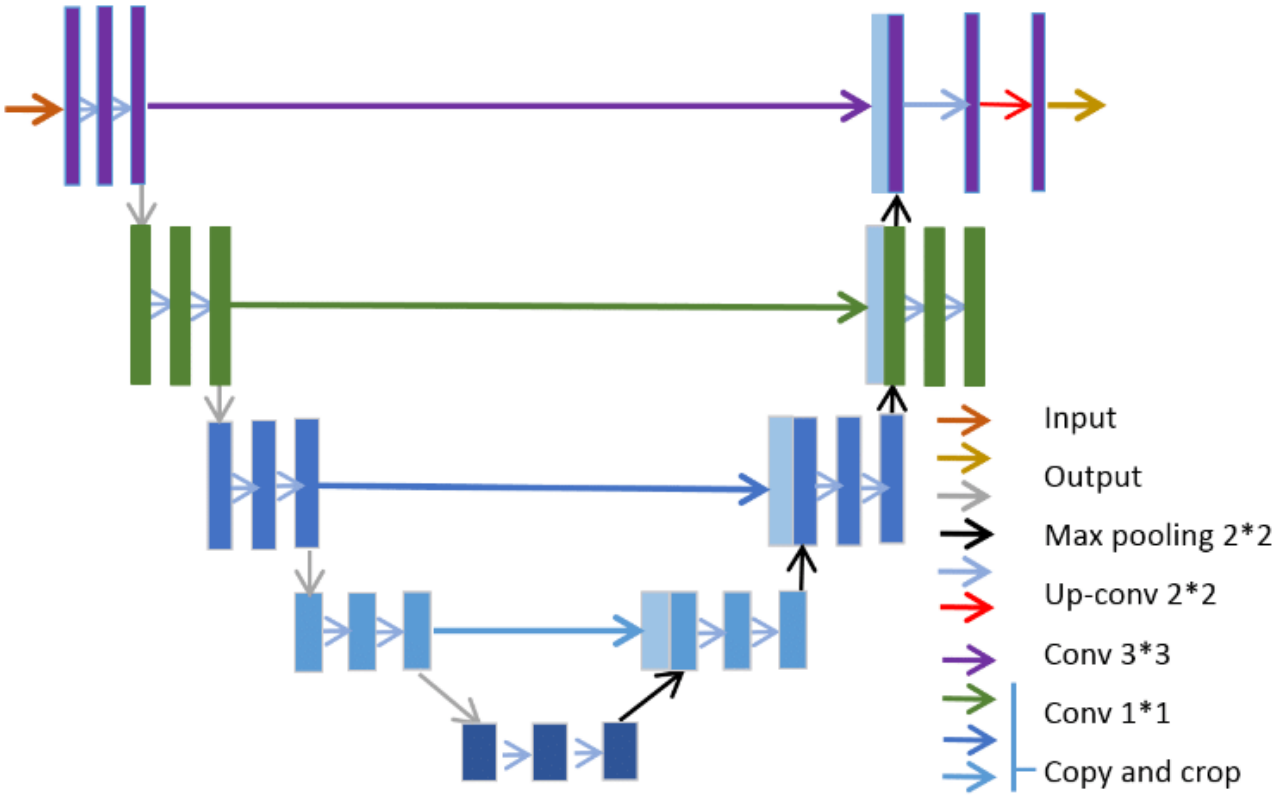
DELETE LABELS | RETRAIN

Raw anomaly score — Scoring method to colour by

Shortcut connections and the U-Net architecture

Shortcut/skip connections (or concatenate connections) are used to **adjunct the activation maps of a given layer to another layer** with the same spatial dimensions.

They are mostly used to pass high resolution maps from early stages of the network to later stage, usually from an encoder part to a decoder part like in the U-Net architecture.

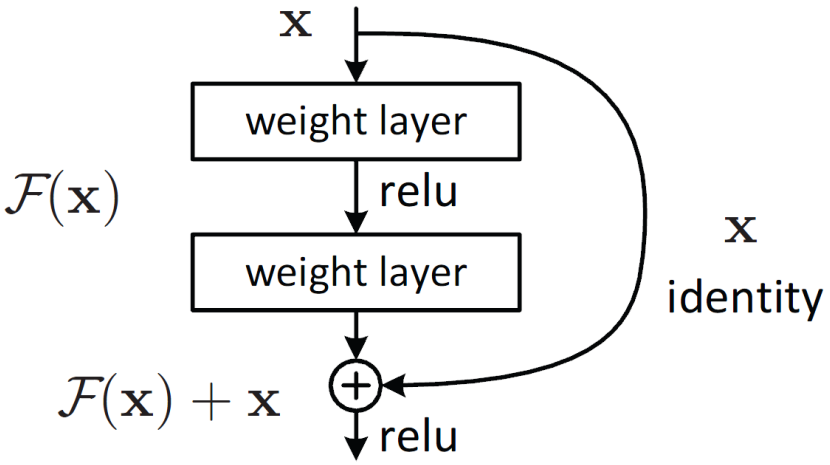


During training, the gradient flows both to the direct previous layer and through the shortcut connection. At the origin of the shortcut the two gradients are summed.

Using shortcut connections reduce the vanishing gradient issue as the gradient can flow directly to upper parts of the network. This principle can be used to construct deeper networks!

This type of architecture can be used for many application, for example denoising, image segmentation, or image generation, and it can be trained in a supervised or self-supervise way depending on the application.

Very deep networks: Residual connections

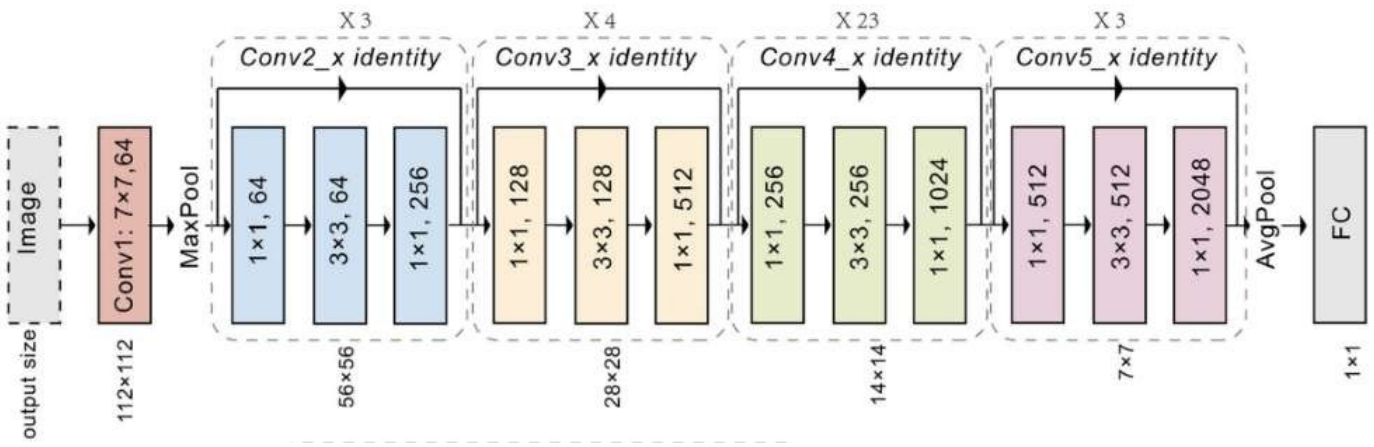


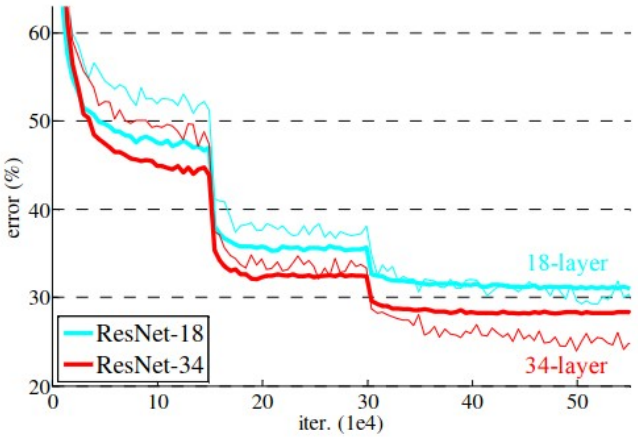
Residual connections are a specific type of shortcut connection where the feature maps of two distant layers are added element wise.

By construction, it means that an identity mapping flows through the residual connection and that all layers in between are tasked to learn a residue to add to this identity.

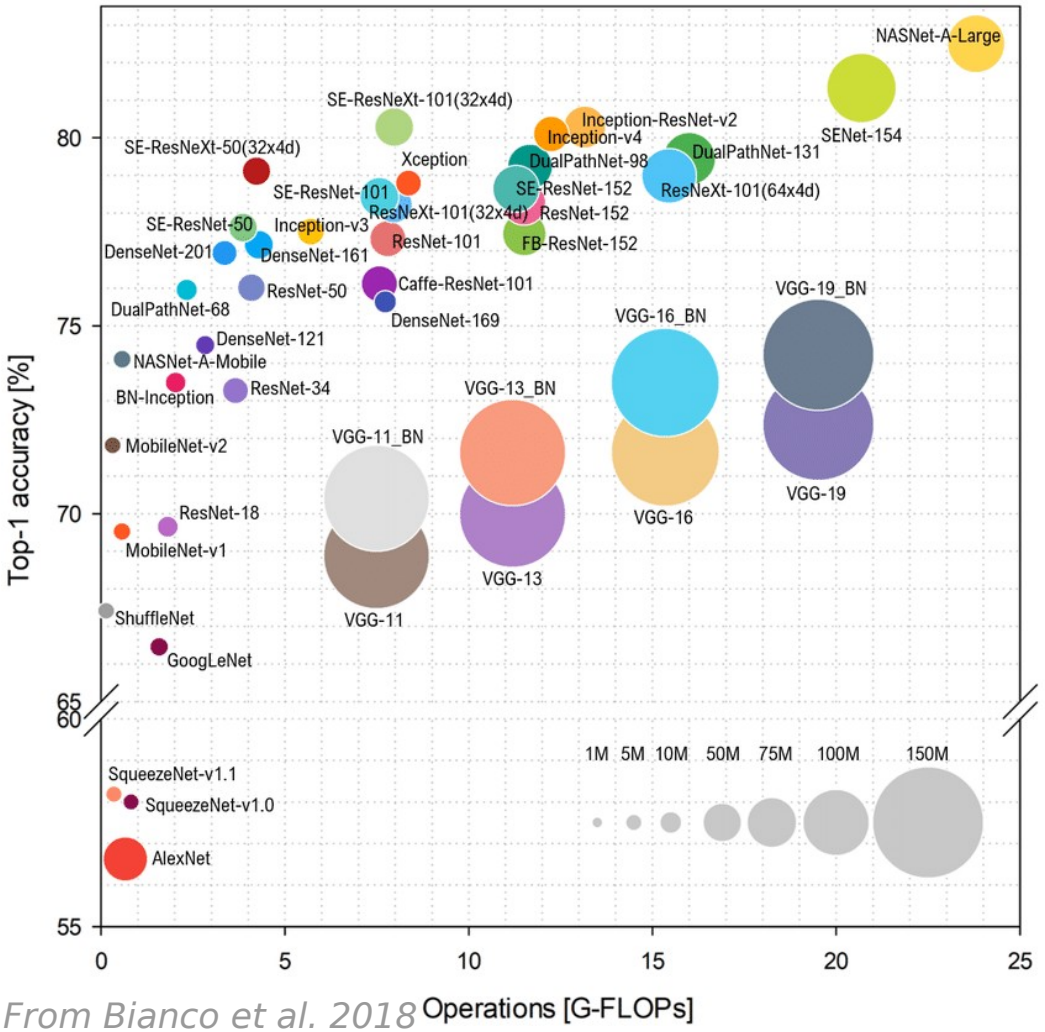
Residual connections are the main innovation that allowed the construction of networks with **hundreds of layers** without suffering from the vanishing gradient effect (Res-Net).

From He et al. 2015





Green AI vs Red AI



From Bianco et al. 2018

← Obtained form Image-Net 2012

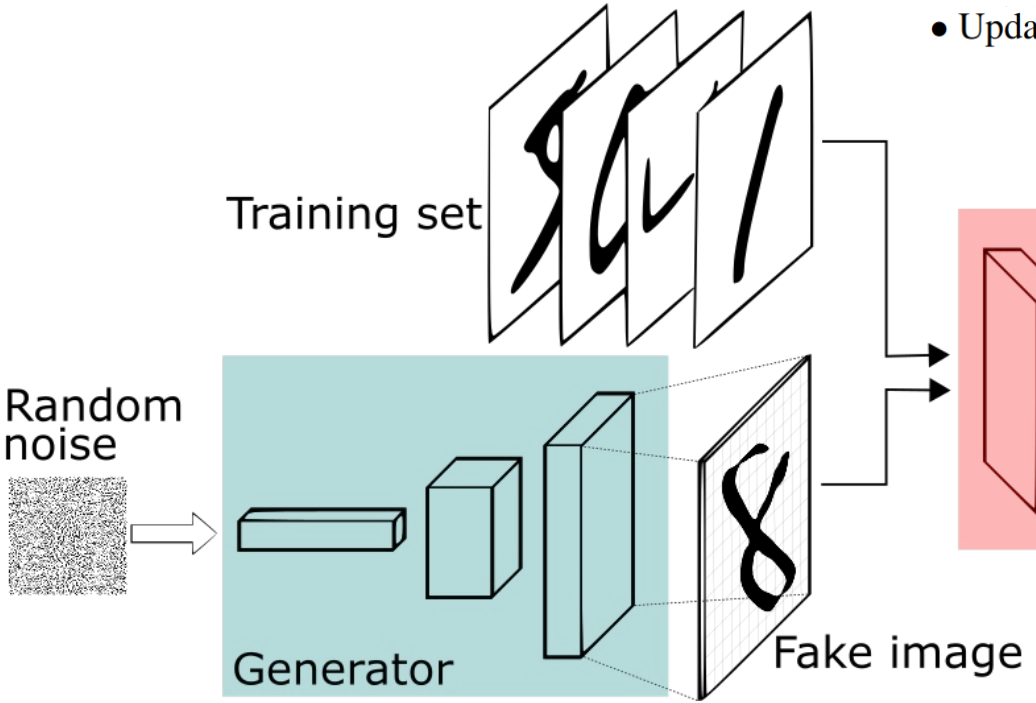
There is a **clear relation between accuracy and compute cost and model size.**

Finding efficient models that reach high accuracy is usually possible but more difficult to design and train.

Exploring architectures and training procedures require computing time !!

- When optimizing a **single prediction**, the total impact is mostly dominated by training time. (e.g., simulation based inference in research). Therefore optimizing the architecture might be irrelevant, or counter-productive.
  → Problem: High accuracy result obtained this way encourage the use of sub-optimized model structures!
- When the model is planned to be **deployed** optimization is way more important as any % gained in efficiency scale with the size of the user base!

Generative Adversarial Networks



- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

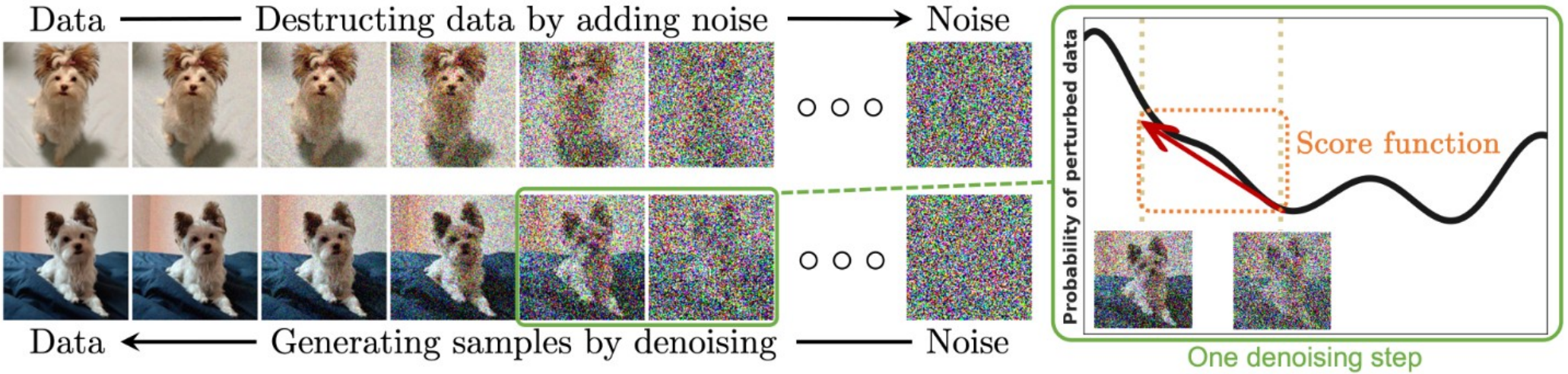
Two « independent » networks plays a min-max adversarial game. At each training step the discriminator make a decision that can then be used to motivate the update of the generator so it can fool the discriminator.

GANs comes in many flavors and can be used to perform a wide variety of tasks.

GANs can generate very convincing images, but they are usually considered more difficult to train than other generative approaches like diffusion models.

Denoising Diffusion Probabilistic models

From Yang et al. 2022



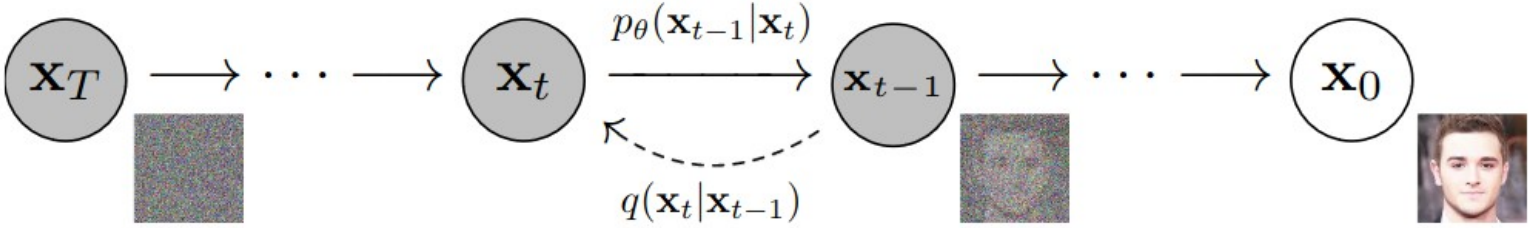
Denoising diffusion models have revolutionized the world of AI image generation.
It is the technic behind most modern AI generator you have encountered.

Training a network to denoise an image with no prior information about its content is a special case of self supervised learning. The very simplicity of this task makes it suitable for almost any dataset, and it will automatically preserve the inherent diversity (or entropy) of the training database (in contrast with GAN that can collapse to a restrained parameter space).
A DDPM can be built from any denoising network architecture, e.g., U-Net or transformers.

With this approach it is possible to train very large models that can be used as foundation models pour other more specific tasks.

Denoising Diffusion Probabilistic models

From Ho et al. 2020



The forward process (diffusion process) is fixed to a Markov Chain that gradually adds Gaussian noise to the data based on a beta variance schedule.

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

The reverse process is a Markov Chain with learned Gaussian transitions.

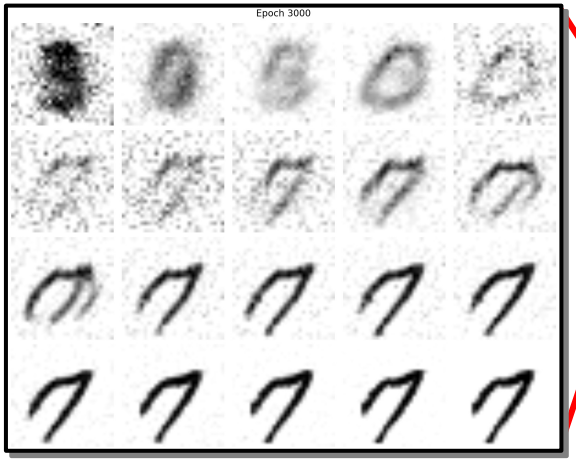
$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

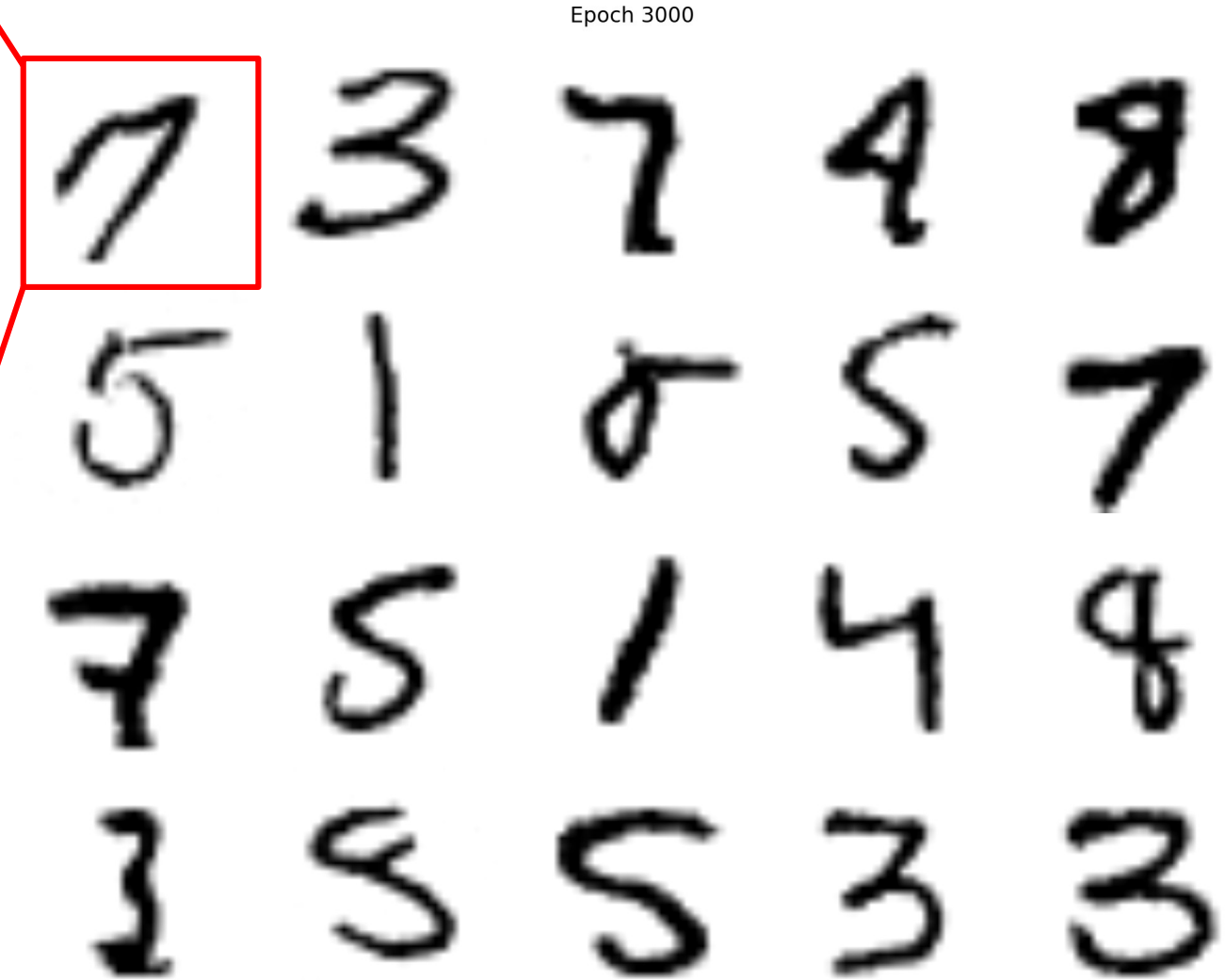
In practice you can use the following relation in both cases:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$$

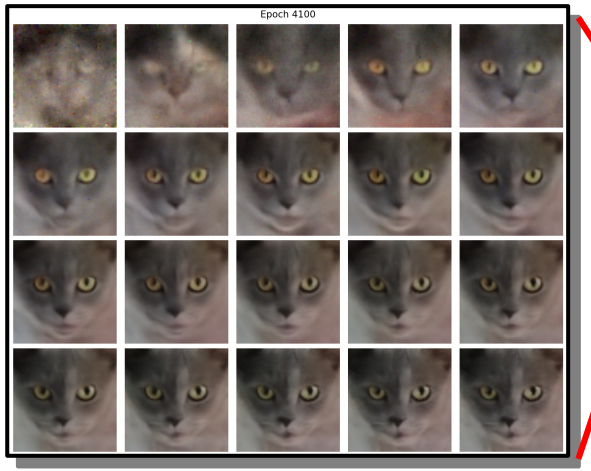
With → $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=0}^{t} \alpha_s$ $\qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

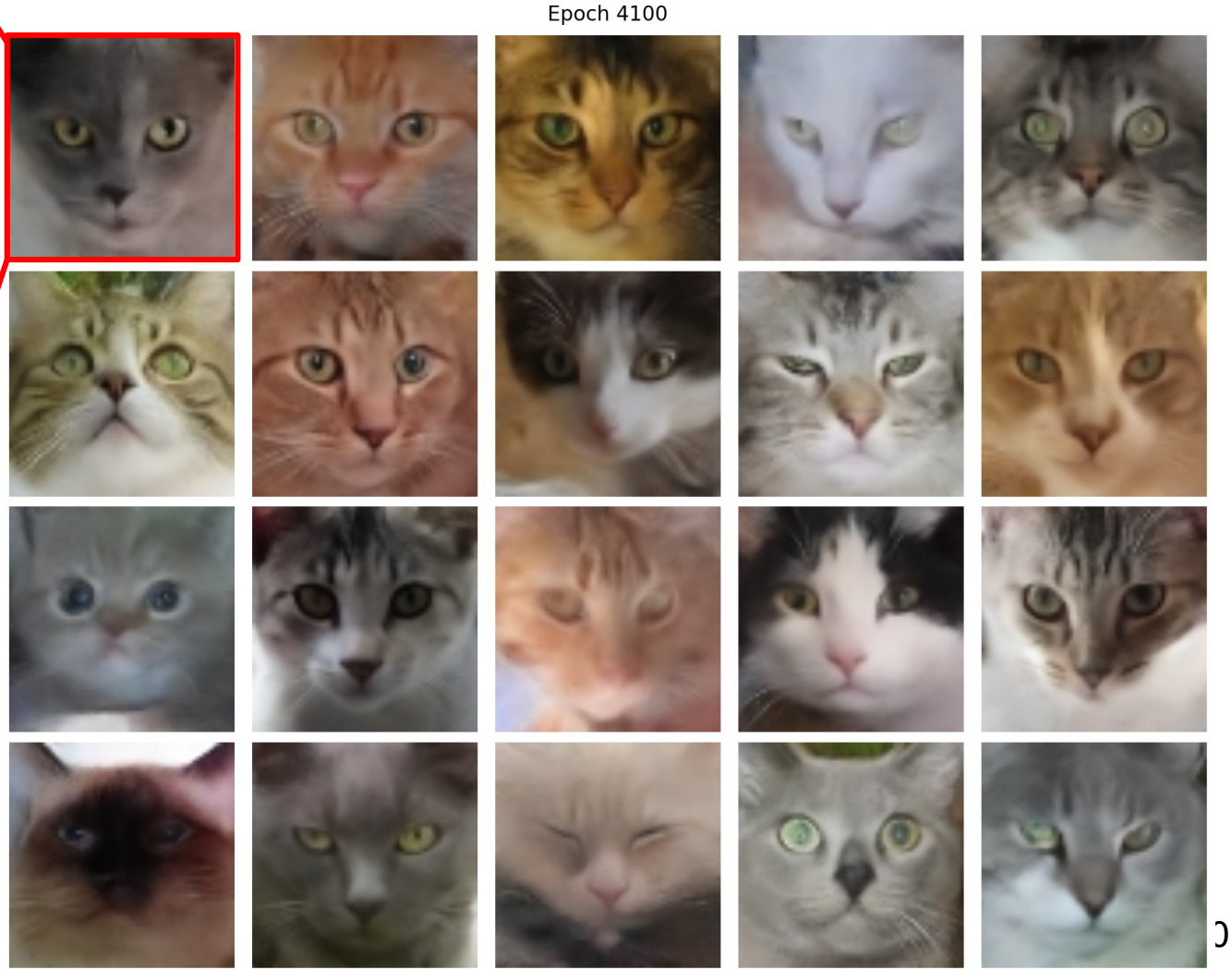
DDPM: MNIST Image generation





DDPM: Cat faces Image generation





DDPM: GZ2 SDSS image generation