An introduction to Multi-Layer Perceptron and Convolutional Neural Network

Giuseppe Angora

Layout

Theory part (~2h):

Introduction to **Perceptron**, **Multi-Layer Perceptron**

Fundamental elements of Neural Network (e.g. **Loss function**, **optimizer**, **activation functions**)

-> Code examples (by using sk-learn and tensorflow packages, *SL_Wen_afternoon.ipynb* file)

Convolutional Neural Networks

-> Code examples (tensorflow package, *SL_Wen_afternoon.ipynb* file)

Autoencoders

-> Code examples (tensorflow package, *SL_Wen_afternoon.ipynb* file)

Coffee Break (~0.5h)

Hans-on session (~1h):

Several exercises (*exercises.ipynb* file)

to be solved **independently** (with solutions in *solved_exercises.ipynb* file)

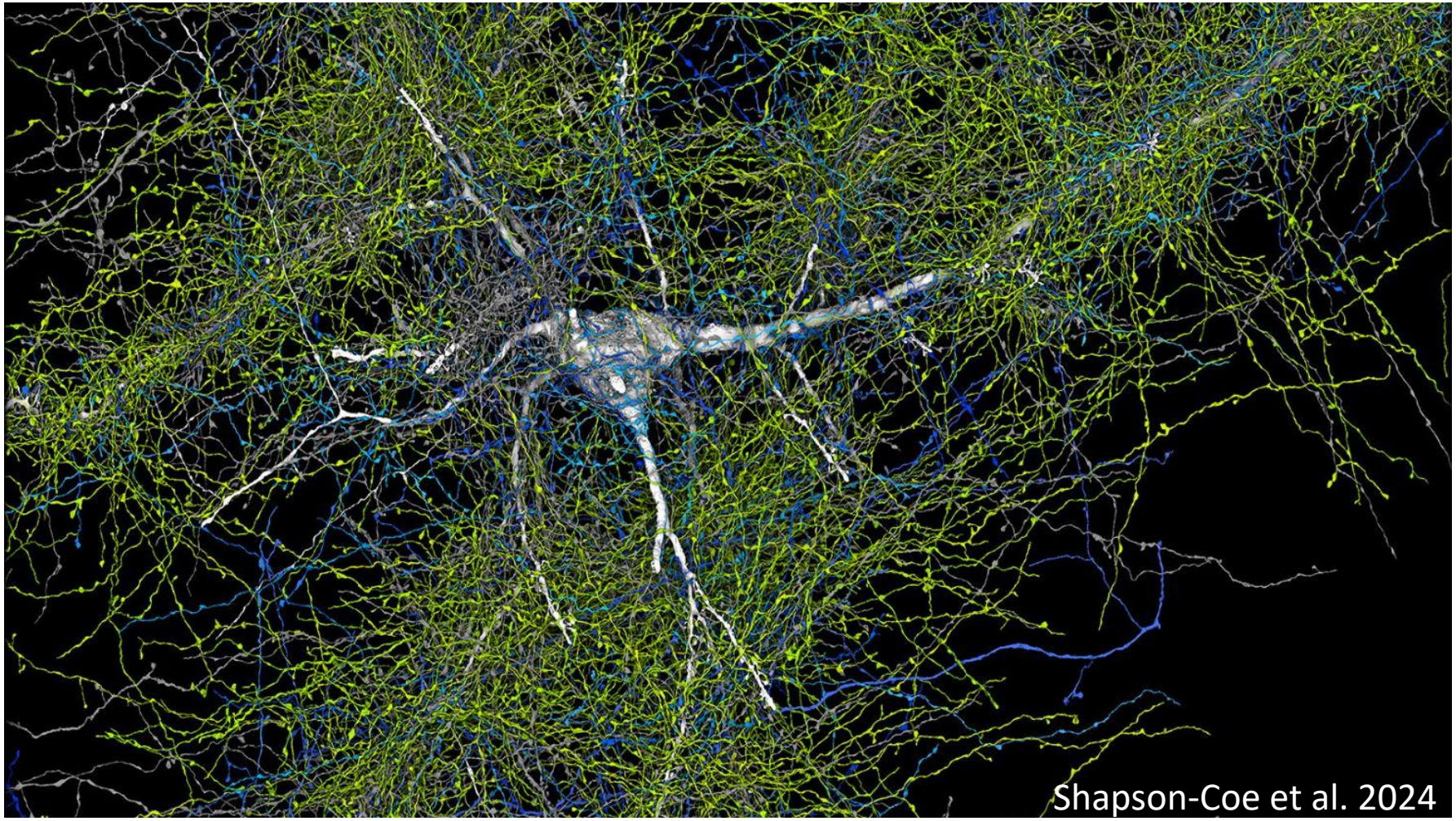
File summary:

SL_Wen_afternoon.ipynb: examples of neural network implementation

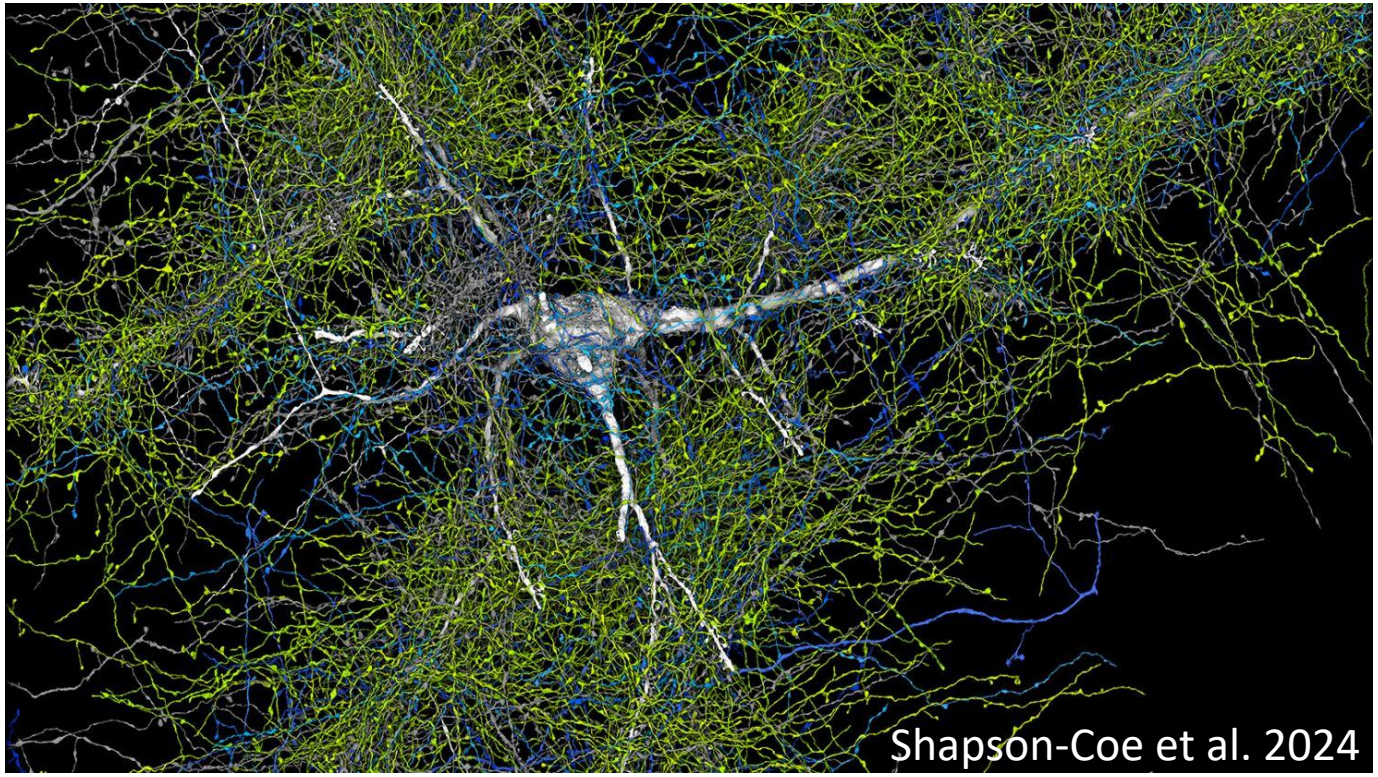
exercises.ipynb: exercises for the hands-on session

solved_exercises.ipynb: (my) solution for the proposed exercises

Starting from the biology



Starting from the biology



In homo sapiens:

≥100G neurons

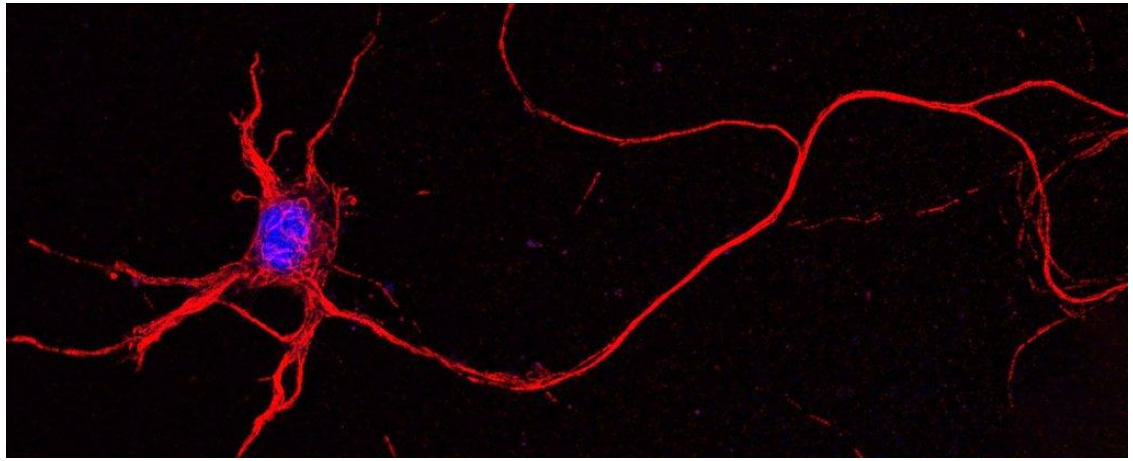
~1k neuron-neuron connections (up to 80k per neuron)

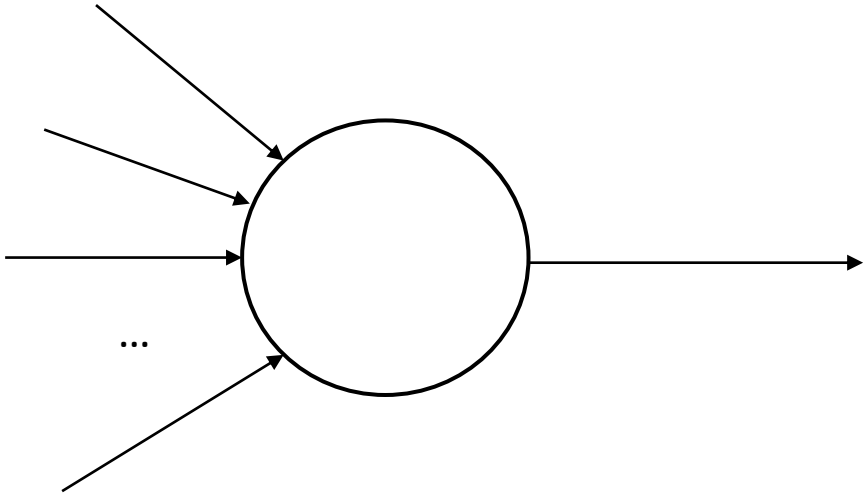
Propagation speed: (min: 500 mm/s – 100 m/s)

Data Volume ~1ZB (10^{21} byte)

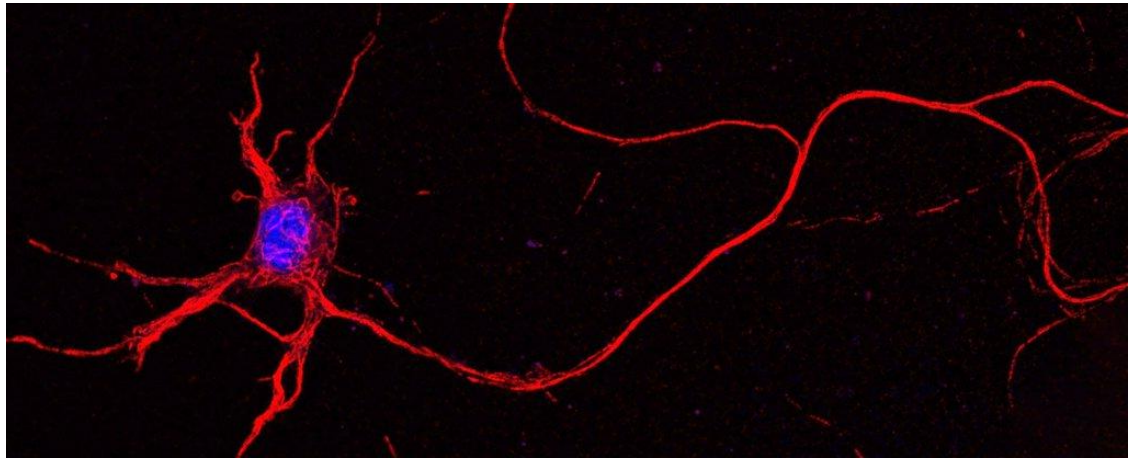
~ 50M differential equations to represent a single neuron

The Perceptron

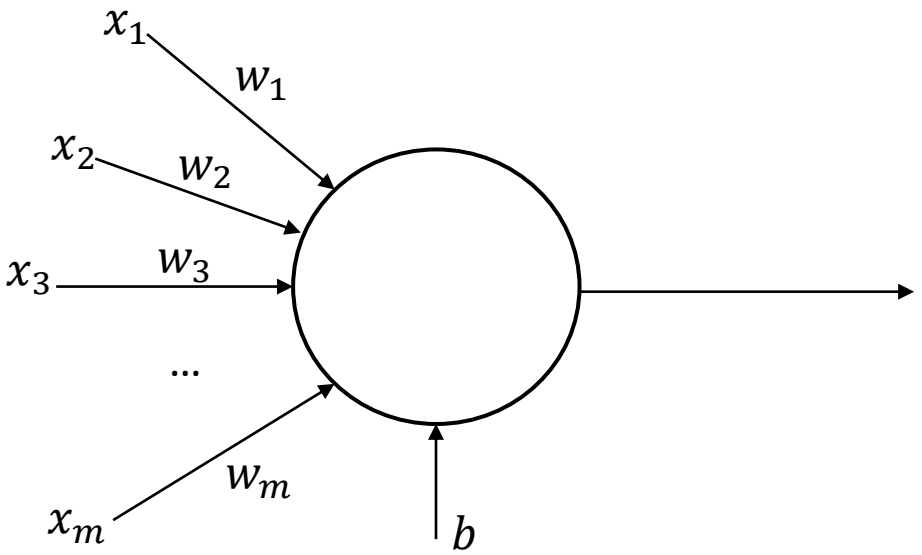




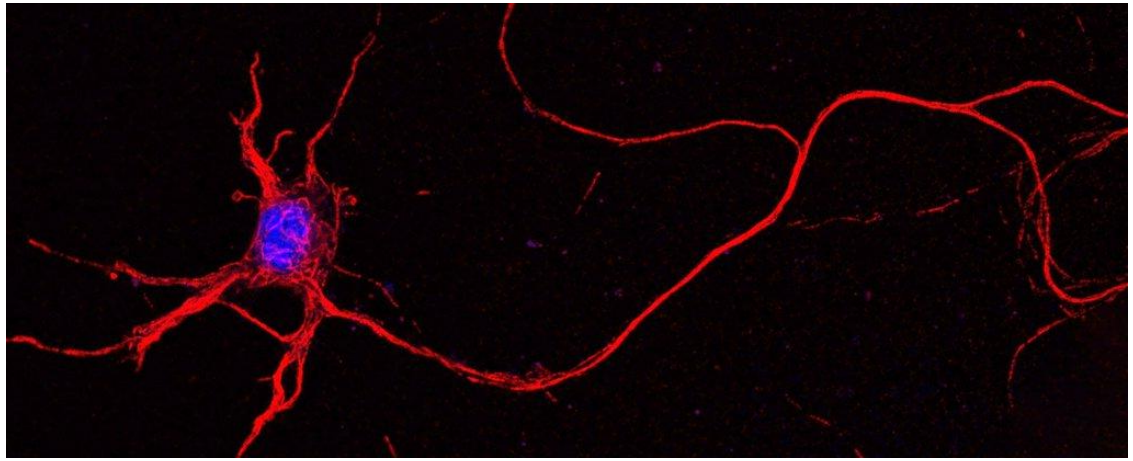
The Perceptron



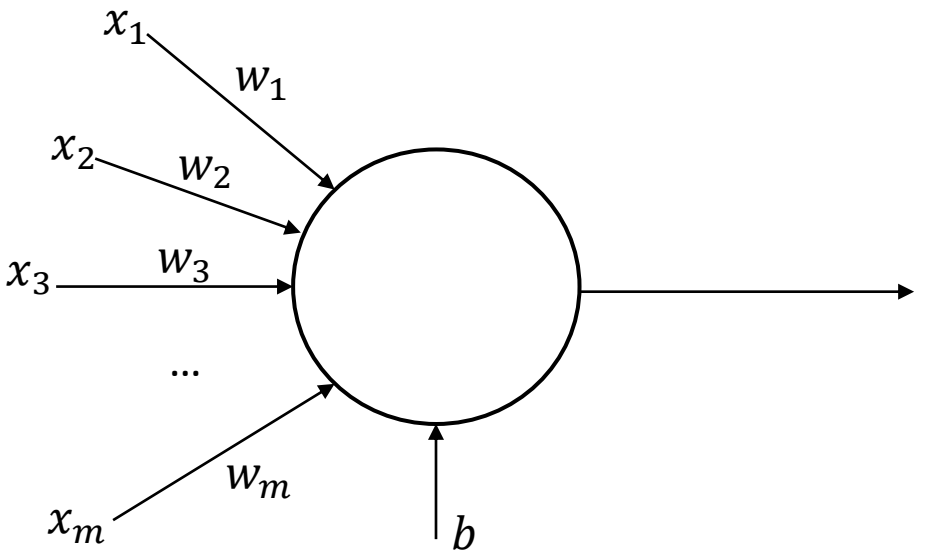
Hebbian behavior (Hebb, 1949): The variation of the synaptic connection sensibility (with respect to a certain input signal) is correlated to the learning mechanism



The Perceptron

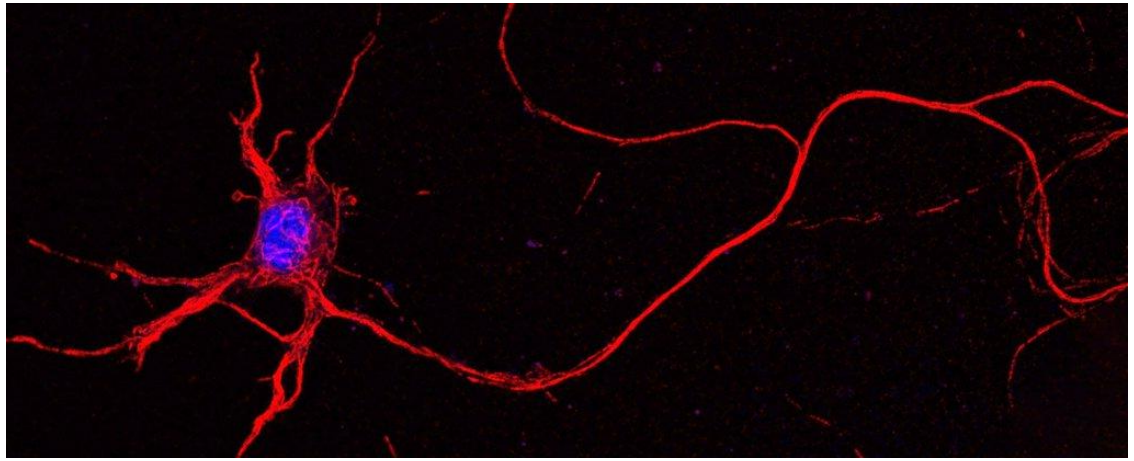


Hebbian behavior (Hebb, 1949): The variation of the synaptic connection sensibility (with respect to a certain input signal) is correlated to the learning mechanism

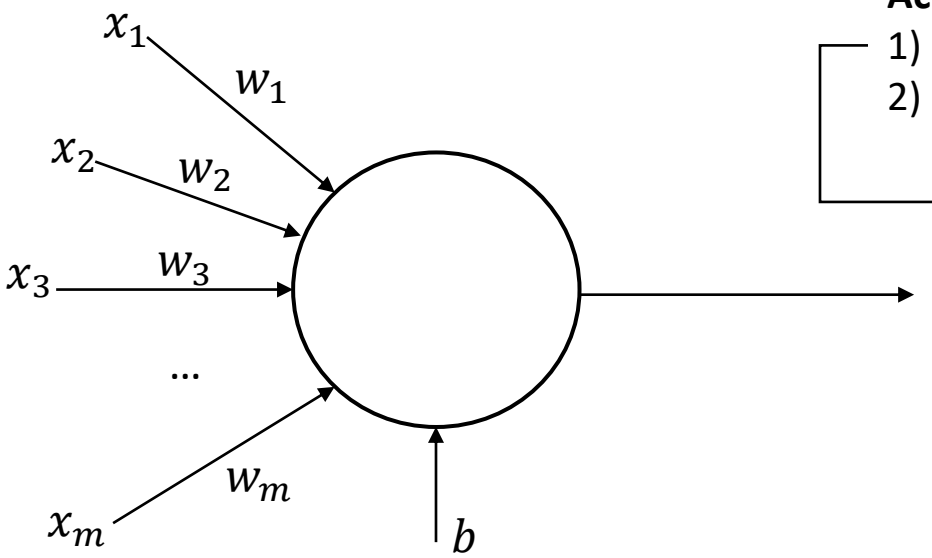


$$y = \sum_{j=1}^{m} w_j x_j + b$$

The Perceptron



Hebbian behavior (Hebb, 1949):
The variation of the synaptic connection sensibility (with respect to a certain input signal) is correlated to the learning mechanism



Activation function (inspired by biology):
1) Impose a codomain;
2) Introduce non-linear elements.

$$y = \sigma\left(\sum_{j=1}^{m} w_j x_j + b\right)$$

Activation functions

Activation function	Equation
Step[1,5]	$\sigma(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$
Rectified Linear Unit (ReLU)[1,5]	$\sigma(x) = max(0, x)$
Parametric ReLU (PReLU)[2,3]	$\sigma(x) = \begin{cases} \alpha x & x \leq 0 \\ x & x > 0 \end{cases}$
Exponential Linear Unit (ELU)[4,6]	$\sigma(x) = \begin{cases} \alpha(e^x - 1) & x \leq 0 \\ x & x > 0 \end{cases}$
Softplus[1,7,8]	$\sigma(x) = \ln(1 + e^x)$
Sigmoid[1,6,7,8]	$\sigma(x) = \frac{1}{1 + e^{-x}}$
Hyperbolic tangent[1,6]	$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Softmax[1,5,6]	$\sigma_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \quad i = 1, \ldots, M \quad \mathbf{x} = (x_1, \ldots, x_M)$

Activation functions

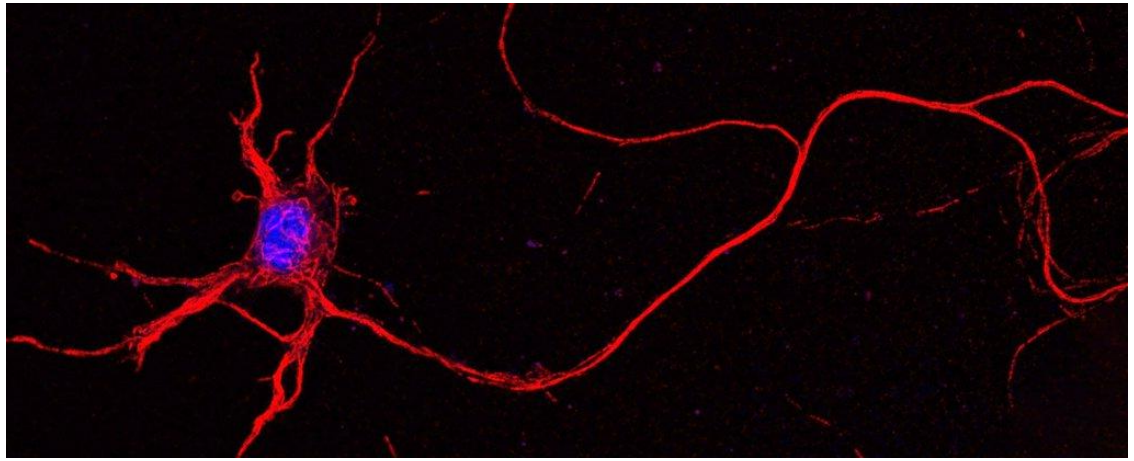
It is required for a classification problem

$\text{Softmax}^{(1,5,6)}$	$\sigma_i(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$	$i = 1, \ldots, M \quad \mathbf{x} = (x_1, \ldots, x_M)$

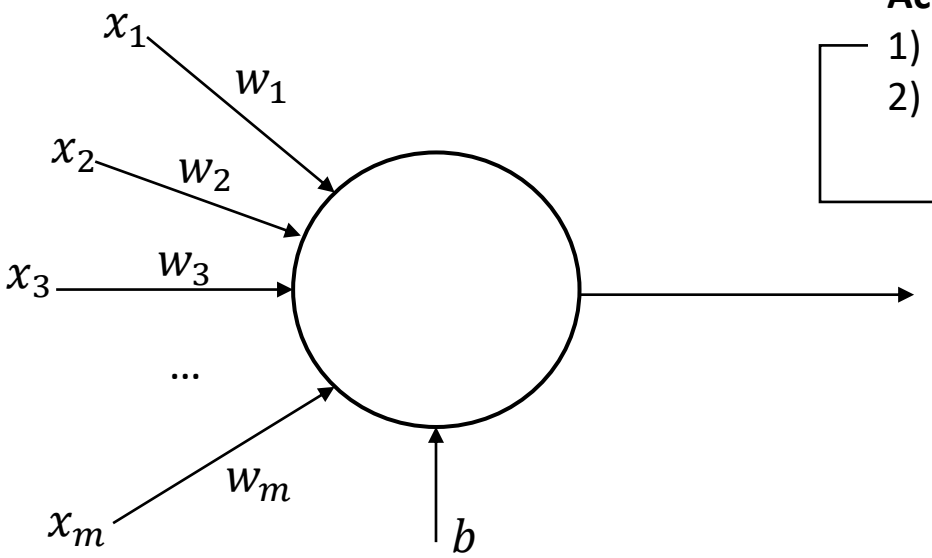
It maps floating values into probabilities:

$$\sigma(softmax): \boldsymbol{x} \in \mathbb{R}^M \rightarrow \boldsymbol{p} \in [0,1]^M: \sum_{j=1}^{M} p_j = 1$$

The Perceptron



Hebbian behavior (Hebb, 1949):
The variation of the synaptic connection sensibility (with respect to a certain input signal) is correlated to the learning mechanism

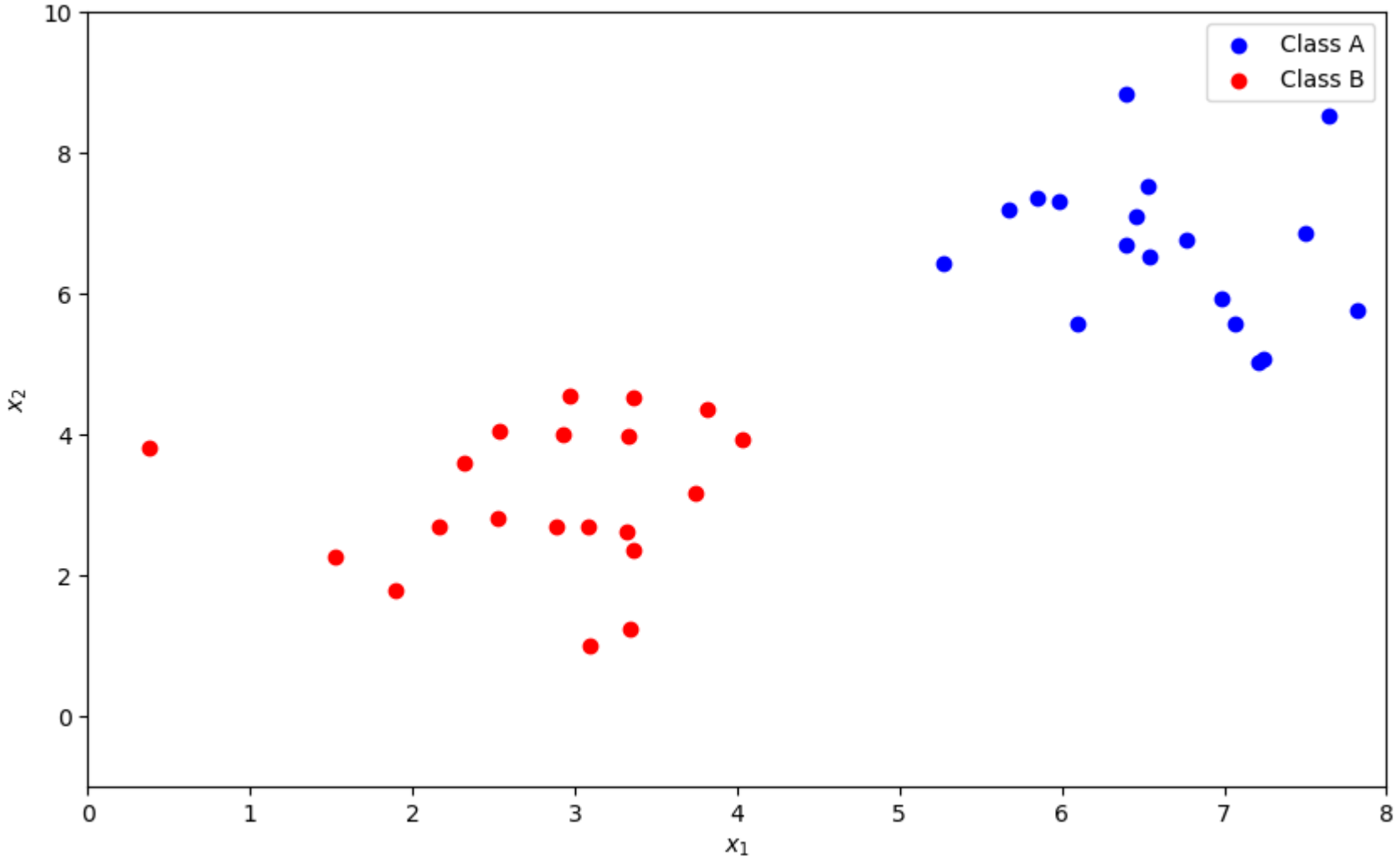


Activation function (inspired by biology):
1) Impose a codomain;
2) Introduce non-linear elements.

$$y = \sigma\left(\sum_{j=1}^{m} w_j x_j + b\right)$$

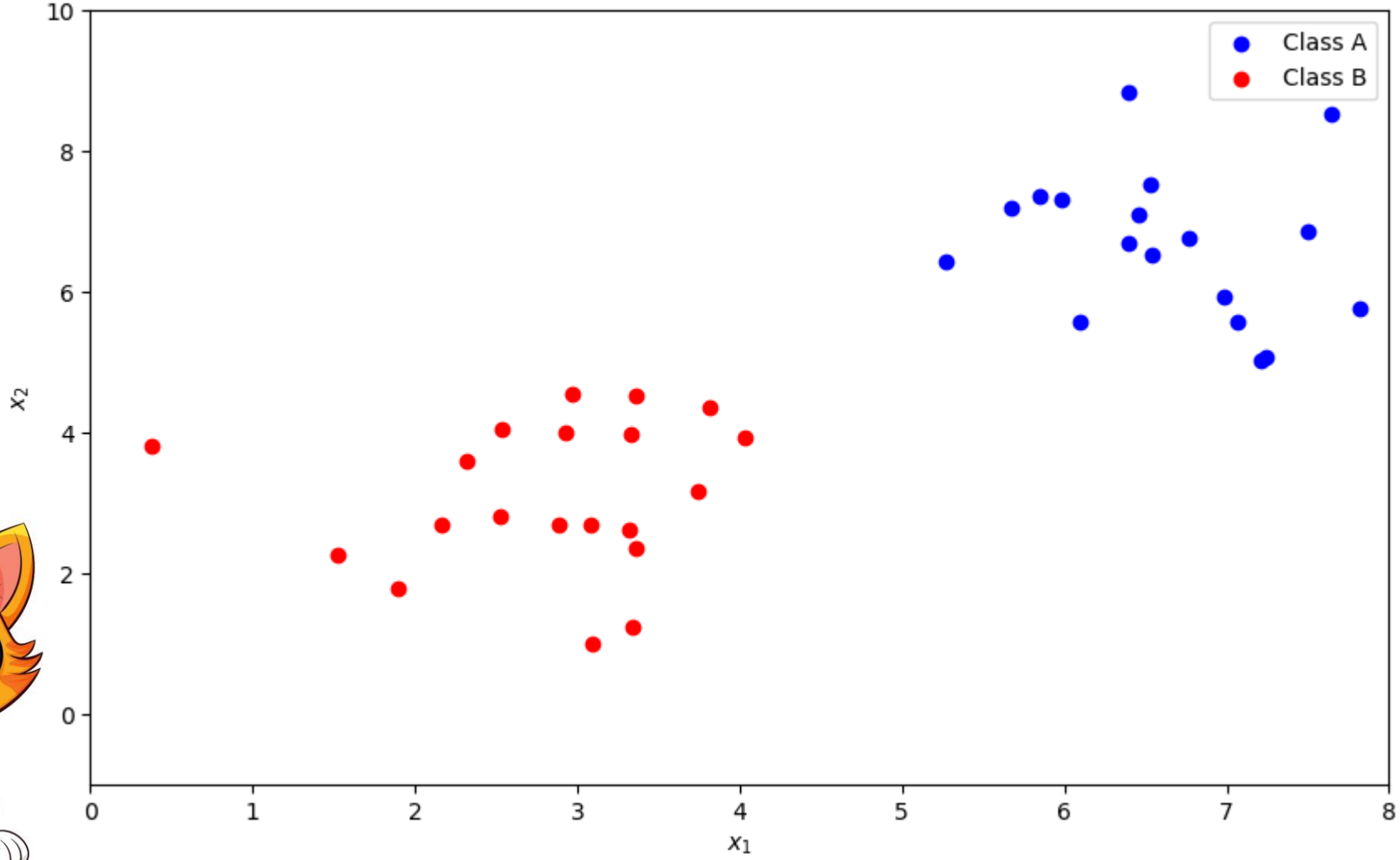
The Perceptron

Let's suppose two point distributions (class A and B)



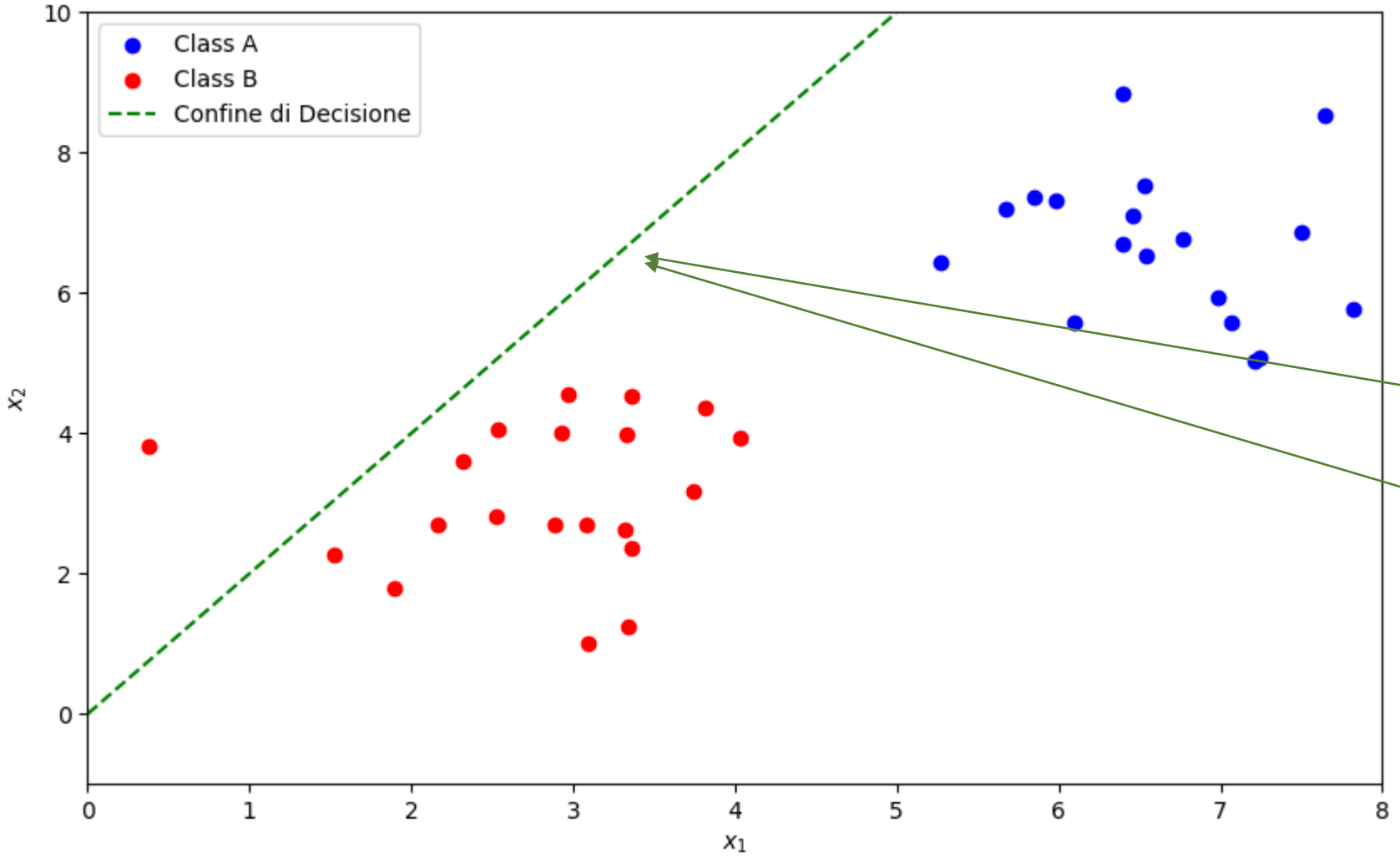
The Perceptron

Let’s suppose two point distributions (class A and B)



The Perceptron

Let's suppose two point distributions (class A and B)
Separation line is initially random (i.e. random init for weights)



$$y = \sum_{j=1}^{m} w_j x_j + b$$

$$y = w_1 x_1 + w_2 x_2 + b$$

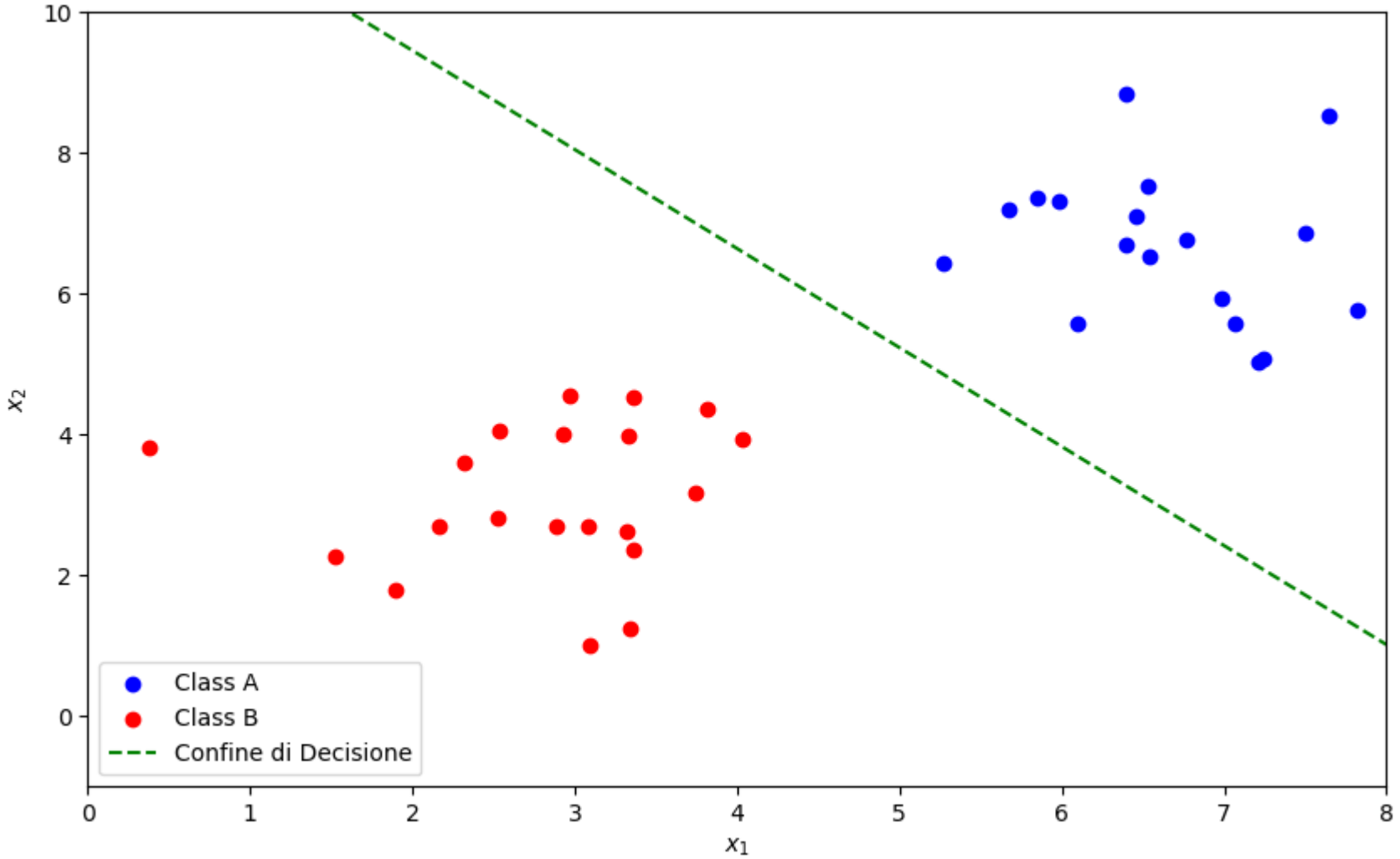
In the x_1, x_2 plane (i.e., $y = 0$)

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

That is a line!

The Perceptron

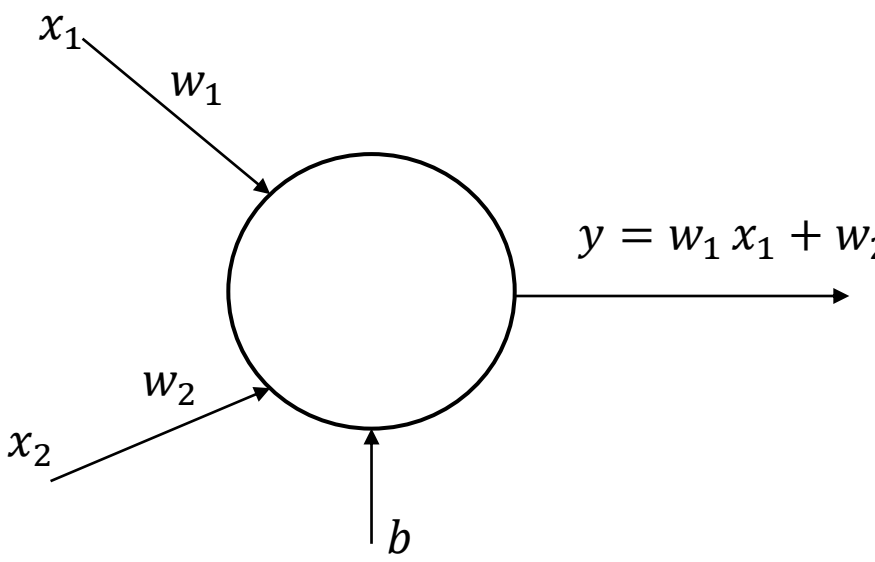
Let's suppose two point distributions (class A and B)
Separation line is initially random (i.e. random init for weights)
During the training weights (w_1 , w_2) are adapted (in order to separate the point distributions)



$$y = w_1 x_1 + w_2 x_2 + b$$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

The loss functions

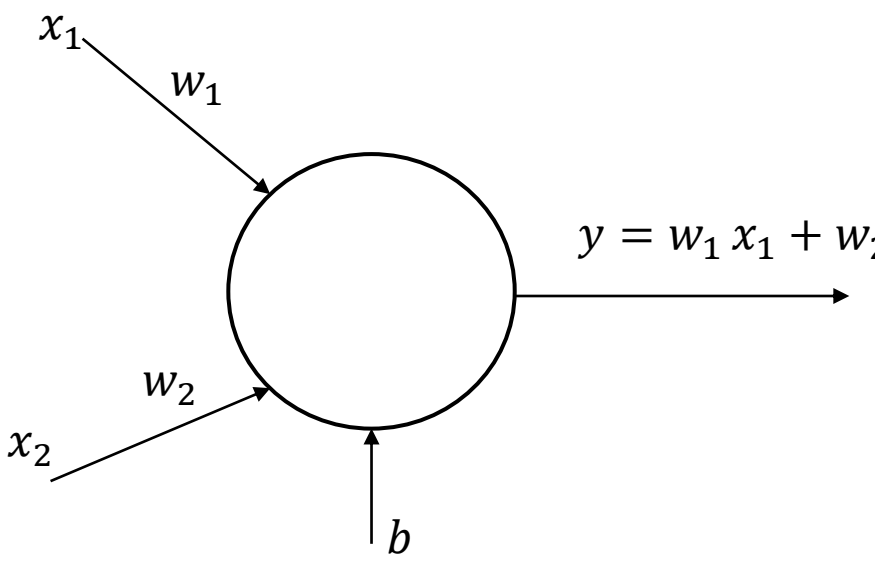


$$y = w_1 x_1 + w_2 x_2 + b$$

The output is compared with the truth (i.e. the labels)

$$y \; ? \; \bar{y}$$

The loss functions



$$y = w_1 x_1 + w_2 x_2 + b$$

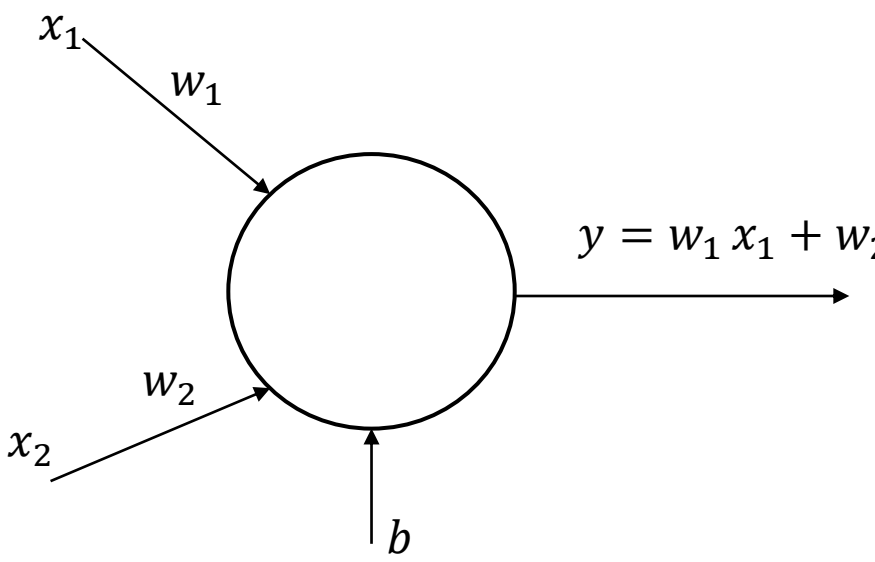
The output is compared with the truth (i.e. the labels)

$$y \; ? \; \bar{y}$$

We are demanding that *all* the perceptron outputs are equal to the labels, i.e:

$$\sum_{i=i}^{N} y_i - \bar{y}_i \rightarrow 0$$

The loss functions



$$y = w_1 x_1 + w_2 x_2 + b$$

The output is compared with the truth (i.e. the labels)

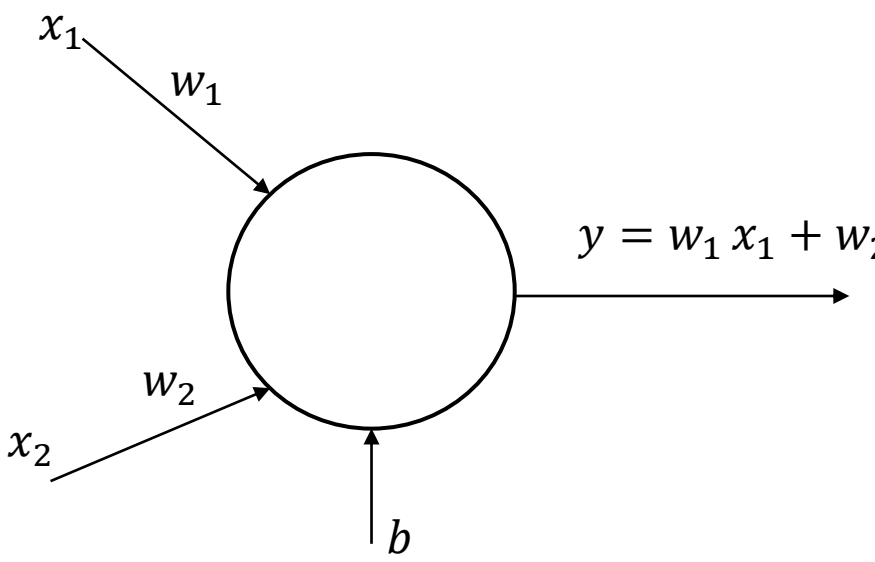
$$y \; ? \; \bar{y}$$

We are demanding that *all* the perceptron outputs are equal to the labels, i.e:

$$\sum_{i=i}^{N} (y_i - \bar{y}_i)^2 \to 0$$

$$\sum_{i=i}^{N} |y_i - \bar{y}_i| \to 0$$

The loss functions



$$y = w_1 x_1 + w_2 x_2 + b$$

The output is compared with the truth (i.e. the labels)

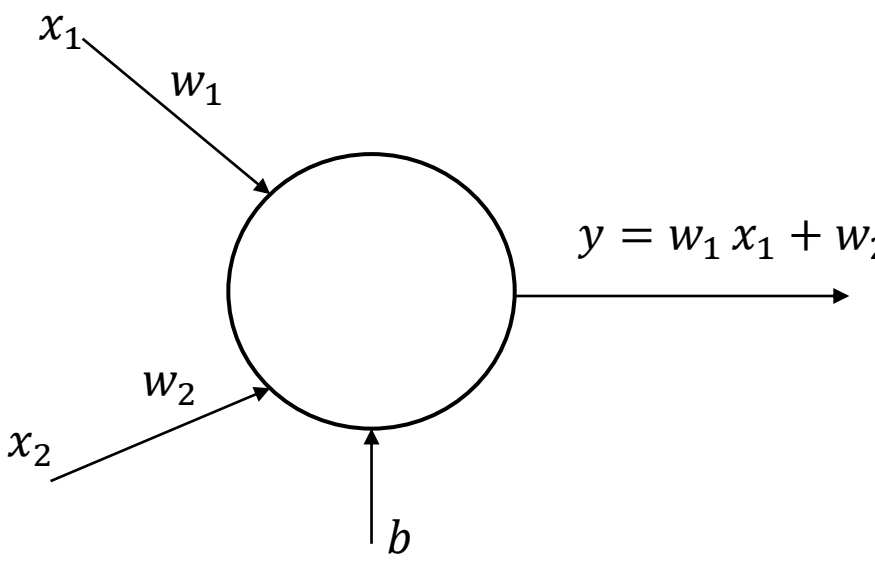
$$y \; ? \; \bar{y}$$

We are demanding that *all* the perceptron outputs are equal to the labels, i.e:

Mean Square Error (MSE): $\frac{1}{N}\sum_{i=i}^{N}(y_i - \bar{y}_i)^2 \to 0$

Mean Absolute Error (MAE): $\frac{1}{N}\sum_{i=i}^{N}|y_i - \bar{y}_i| \to 0$

The loss functions



$$y = w_1 x_1 + w_2 x_2 + b$$

The output is compared with the truth (i.e. the labels)

$$y \; ? \; \bar{y}$$

We are demanding that *all* the perceptron outputs are equal to the labels, i.e:

MSE: $\frac{1}{N}\sum_{i=i}^{N}(y_i - \bar{y}_i)^2$

MAE: $\frac{1}{N}\sum_{i=i}^{N}|y_i - \bar{y}_i|$

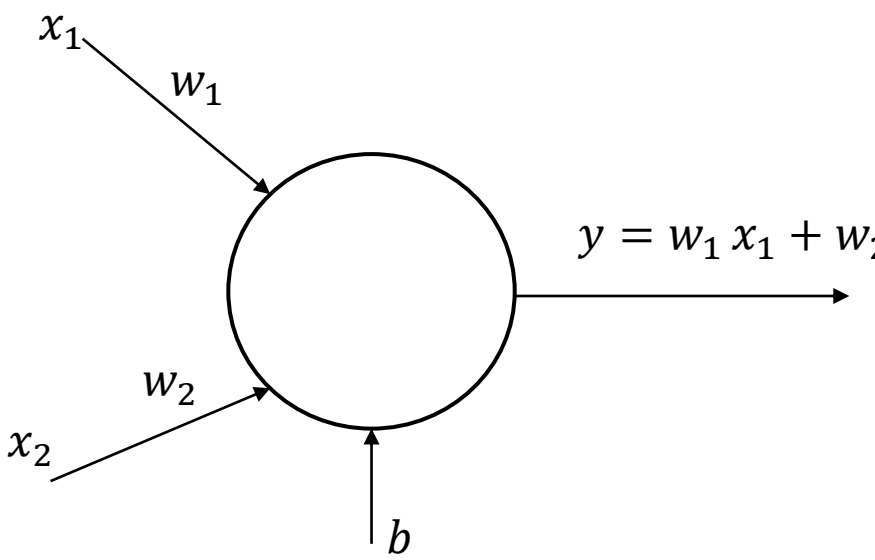
Binary cross-entropy:

$$H(y,\bar{y}) = -\frac{1}{N}\sum_{i=i}^{N}\bar{y}(x_i)\cdot\log\big(y(x_i)\big) + (1-\bar{y}(x_i))\cdot\log\big(1-y(x_i)\big)$$

M-class generalization:

$$H(y,\bar{y}) = -\frac{1}{N}\sum_{i=i}^{N}\sum_{j=1}^{M}\bar{y}_j(x_i)\cdot\log\Big(y_j(x_i)\Big)$$

The loss functions



$$y = w_1 x_1 + w_2 x_2 + b$$

The output is compared with the truth (i.e. the labels)

$$y \; ? \; \bar{y}$$

We are demanding that *all* the perceptron outputs are equal to the labels, i.e:

Regression tasks

MSE: $\frac{1}{N}\sum_{i=i}^{N}(y_i - \bar{y}_i)^2$

MAE: $\frac{1}{N}\sum_{i=i}^{N}|y_i - \bar{y}_i|$

Classification tasks

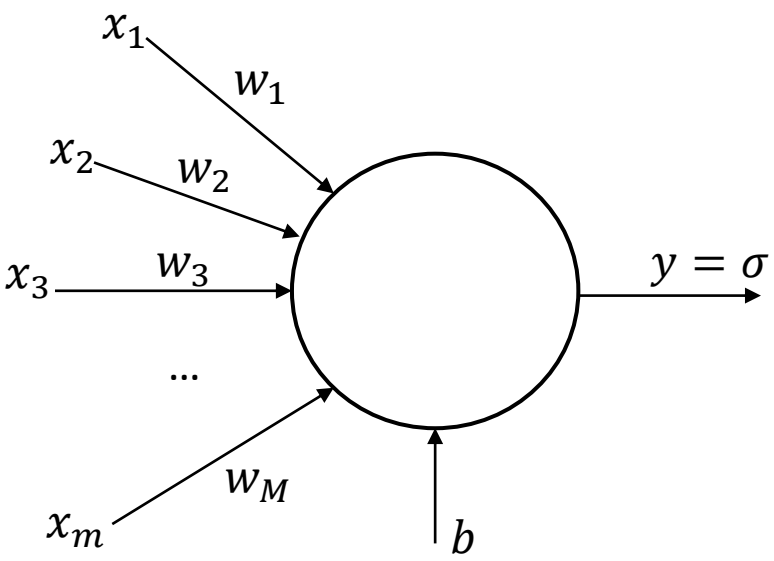
Binary cross-entropy:

$$H(y,\bar{y}) = -\frac{1}{N}\sum_{i=i}^{N}\bar{y}(x_i)\cdot\log\big(y(x_i)\big) + (1-\bar{y}(x_i))\cdot\log\big(1-y(x_i)\big)$$

M-class generalization:

$$H(y,\bar{y}) = -\frac{1}{N}\sum_{i=i}^{N}\sum_{j=1}^{M}\bar{y}_j(x_i)\cdot\log\Big(y_j(x_i)\Big)$$

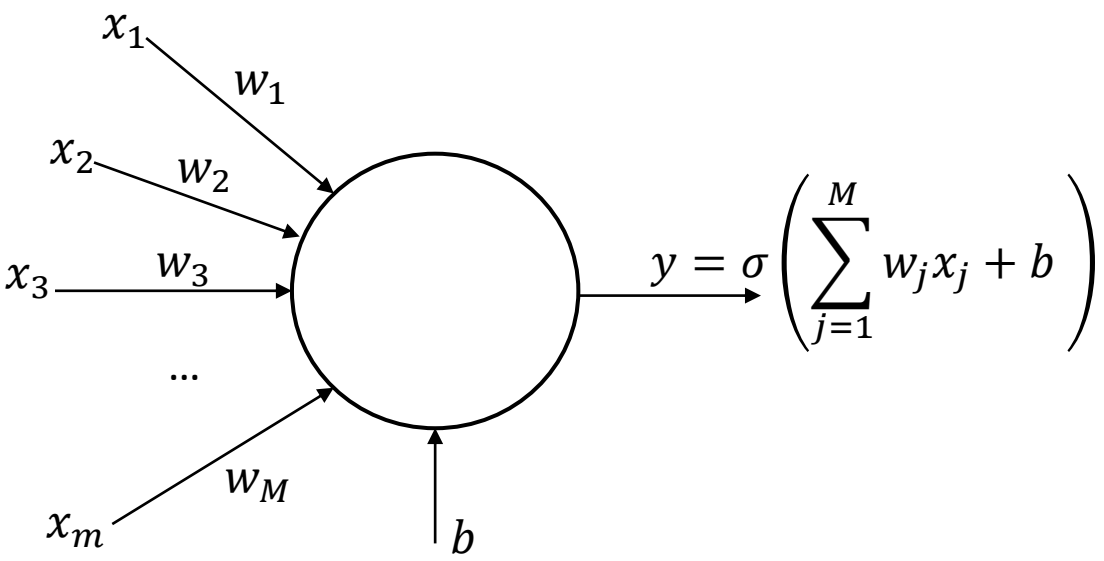
The Optimization: A Recap



$$y = \sigma\left(\sum_{j=1}^{M} w_j x_j + b\right) = f(x_j | w_j, b)$$

1) The optimizer compares the output with the so called Ground Truth $= \bar{y}$

The Optimization: A Recap

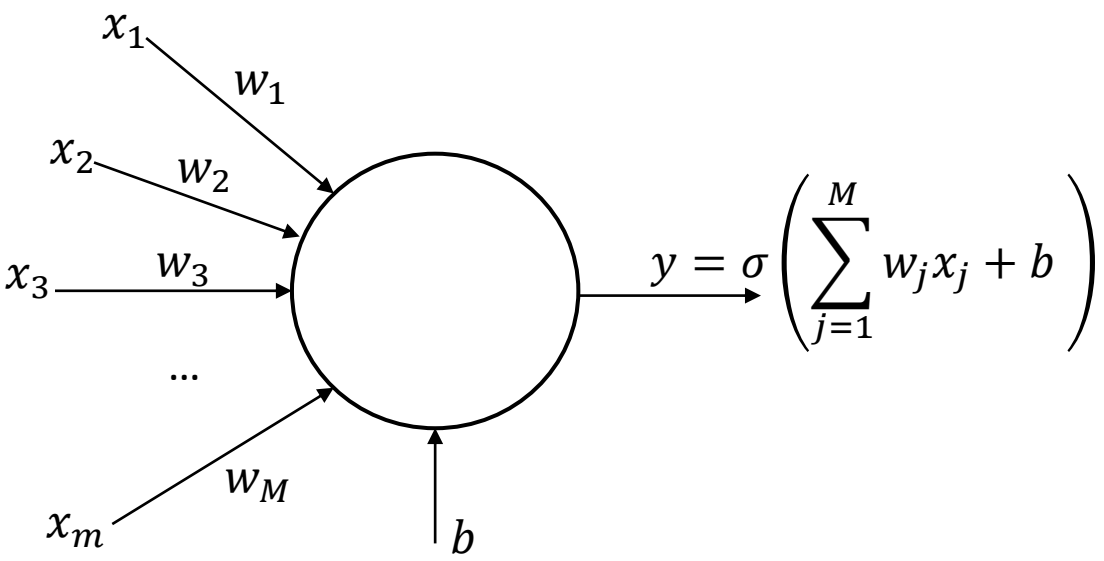


1) The optimizer compares the output with the so called Ground Truth $= \bar{y}$

2) It computes the current value (at epoch t) of the loss function:

$$C(y, \bar{y}\,|w_j^{(t)}, b^{(t)})$$ (e.g. MSE, MAE, Cross entropy)

The Optimization: A Recap



1) The optimizer compares the output with the so called Ground Truth $= \bar{y}$

2) It computes the current value (at epoch t) of the loss function:

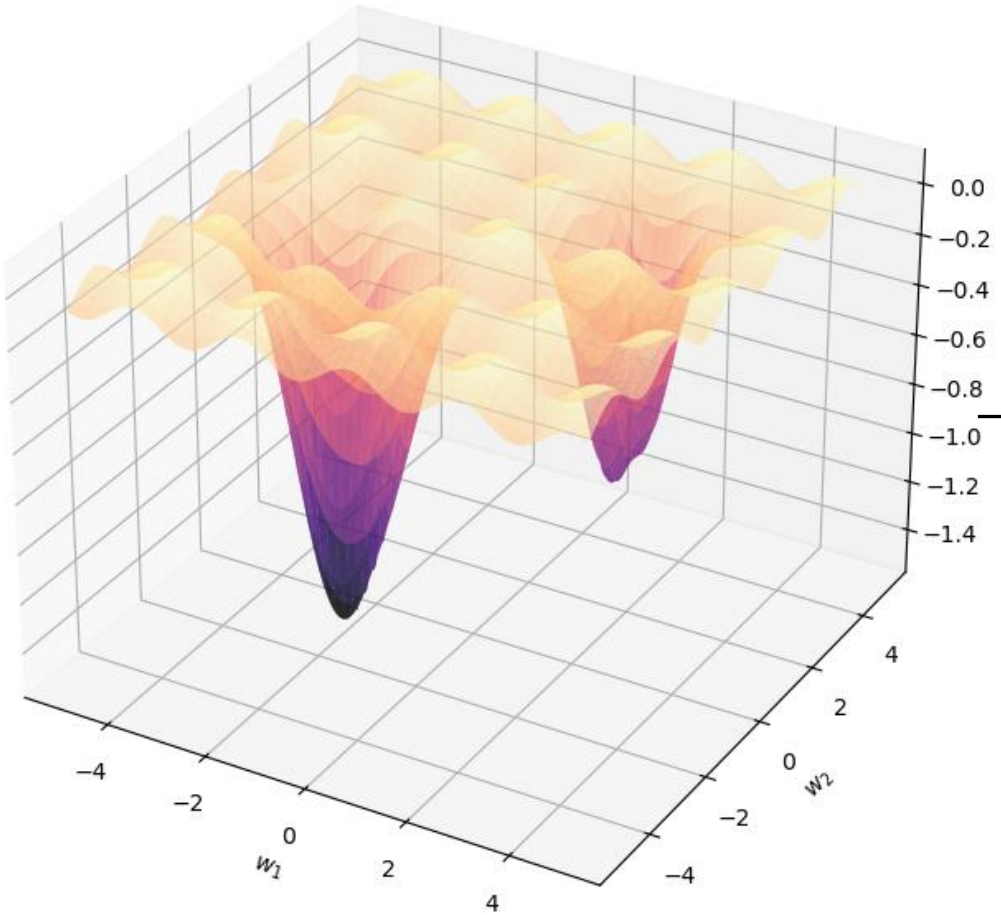
$$C(y, \bar{y}\,|w_j^{(t)}, b^{(t)})$$ (e.g. MSE, MAE, Cross entropy)

3) Given the loss function, the optimizer adjusts the weights and biases $(w_j^{(t+1)}, b^{(t+1)})$ that will be used at epoch $t+1$

The Optimization

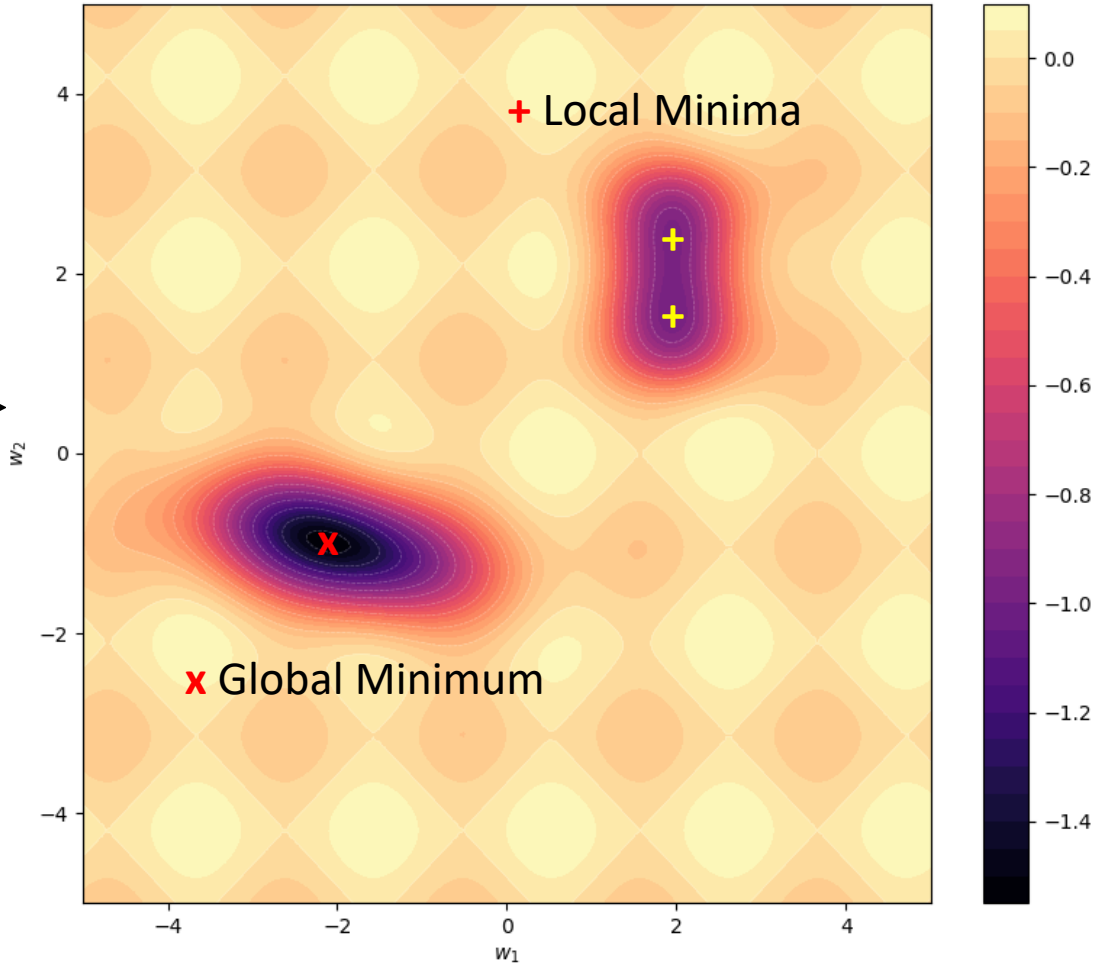
The simplest optimizer is the **Stochastic Gradient Descent (SGD)**

Let's suppose this is the loss function:



2D projection

in the w_1, w_2 plane



The Optimization

The simplest optimizer is the **Stochastic Gradient Descent (SGD)**

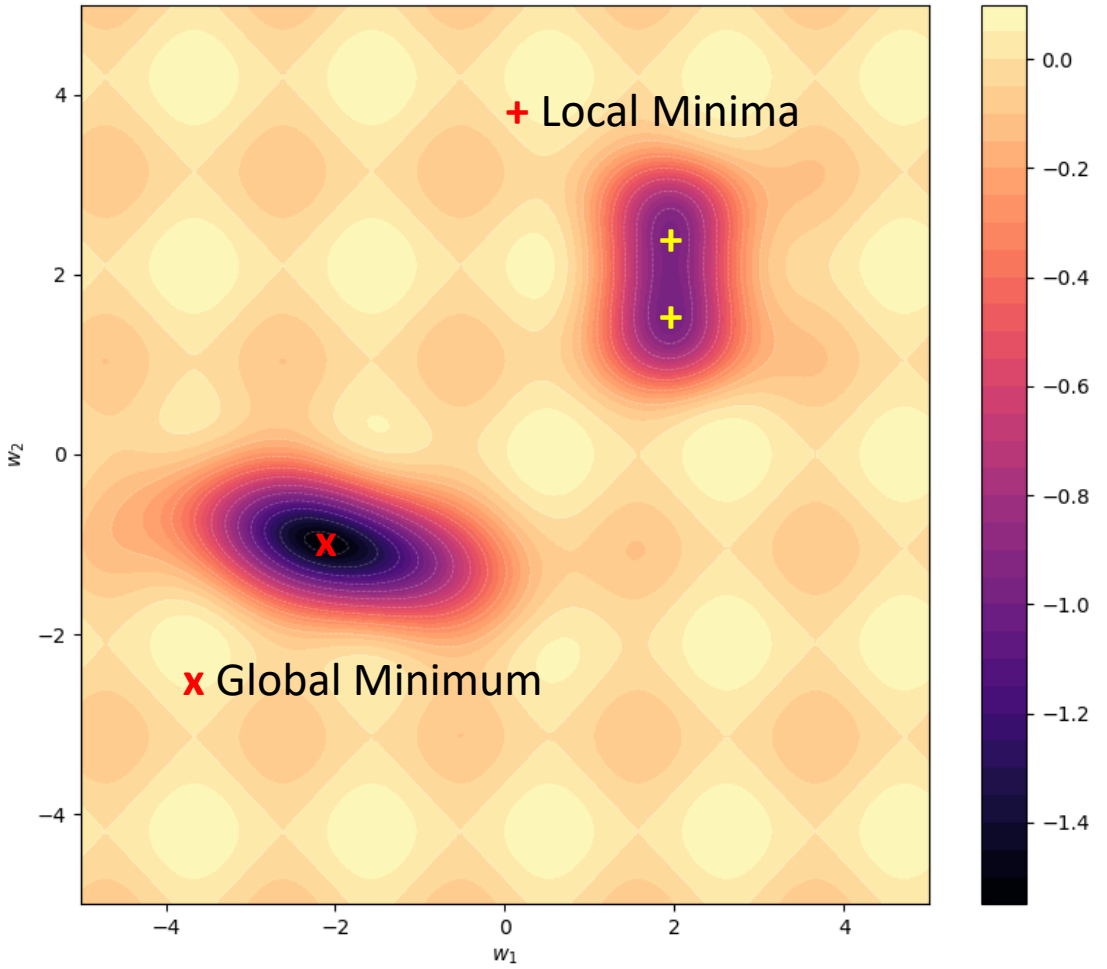
The **SGD** searches for a path toward the minimum,
by computing the gradient of the loss function:

$$C(y, \bar{y} \,|w_1^{(t)}, w_2^{(t)})$$

$$\nabla_{w_1^{(t)}} C, \nabla_{w_2^{(t)}} C$$

SGD adaptation rule:

$$w_1^{(t+1)} = w_1^{(t)} - \eta \nabla_{w_1^{(t)}} C$$
$$w_2^{(t+1)} = w_2^{(t)} - \eta \nabla_{w_2^{(t)}} C$$



The Optimization

The simplest optimizer is the **Stochastic Gradient Descent (SGD)**

The **SGD** searches for a path toward the minimum,
by computing the gradient of the loss function:

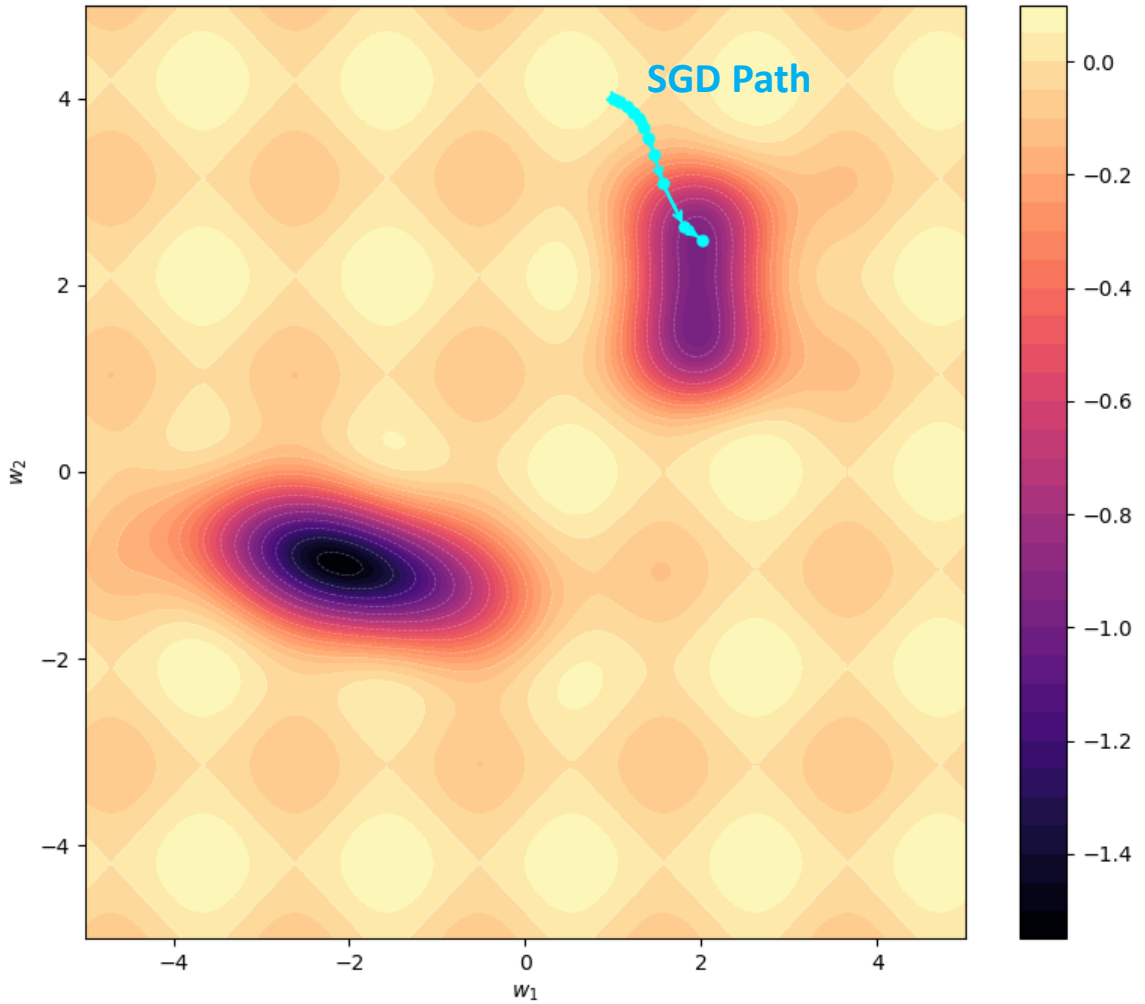
$$C(y, \bar{y}\,|w_1^{(t)}, w_2^{(t)})$$

$$\nabla_{w_1^{(t)}} C, \nabla_{w_2^{(t)}} C$$

SGD adaptation rule:

$$w_1^{(t+1)} = w_1^{(t)} - \eta \nabla_{w_1^{(t)}} C$$
$$w_2^{(t+1)} = w_2^{(t)} - \eta \nabla_{w_2^{(t)}} C$$

What could possibly go wrong?



The Optimization

The simplest optimizer is the **Stochastic Gradient Descent (SGD)**

The SGD searches for a path toward the minimum,
by computing the gradient of the loss function:

$$C(y, \bar{y}\,|w_j^{(t)}) \quad \nabla_{w_j^{(t)}} C(w_j^{(t)})$$

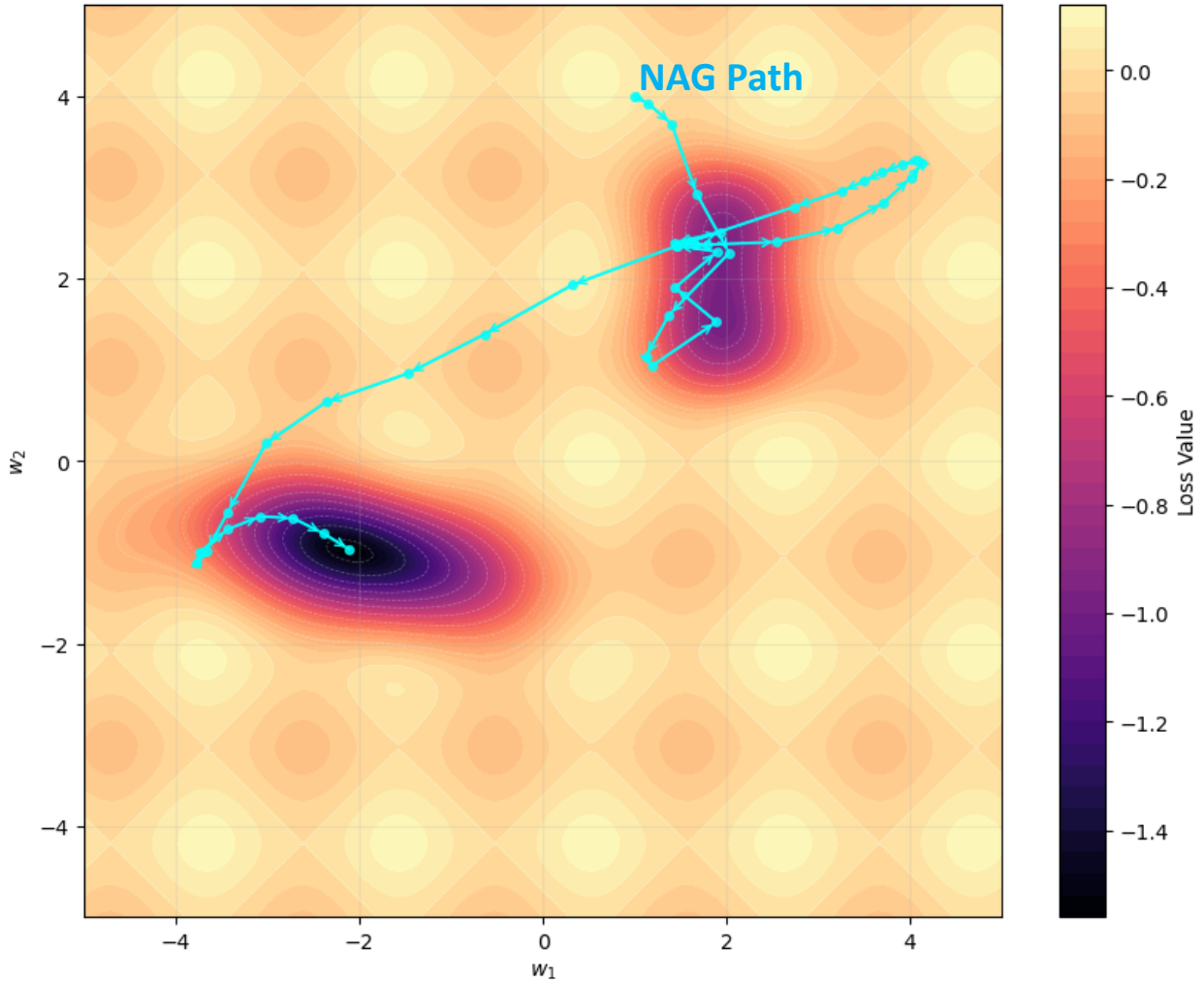
SGD adaptation rule:

$$w_j^{(t+1)} = w_j^{(t)} - \eta \nabla_{w_j^{(t)}} C(w_j^{(t)})$$

Nesterov Accelerated Gradient - NAG adaptation rule:

$$w_j^{(t+1)} = w_j^{(t)} - \eta \nabla_{w_j^{(t)}} C(w_j^{(t)} - \gamma \eta \nabla_{w_j^{(t)}} C)$$

Nesterov momentum



The Optimization

A lot of Optimizers:

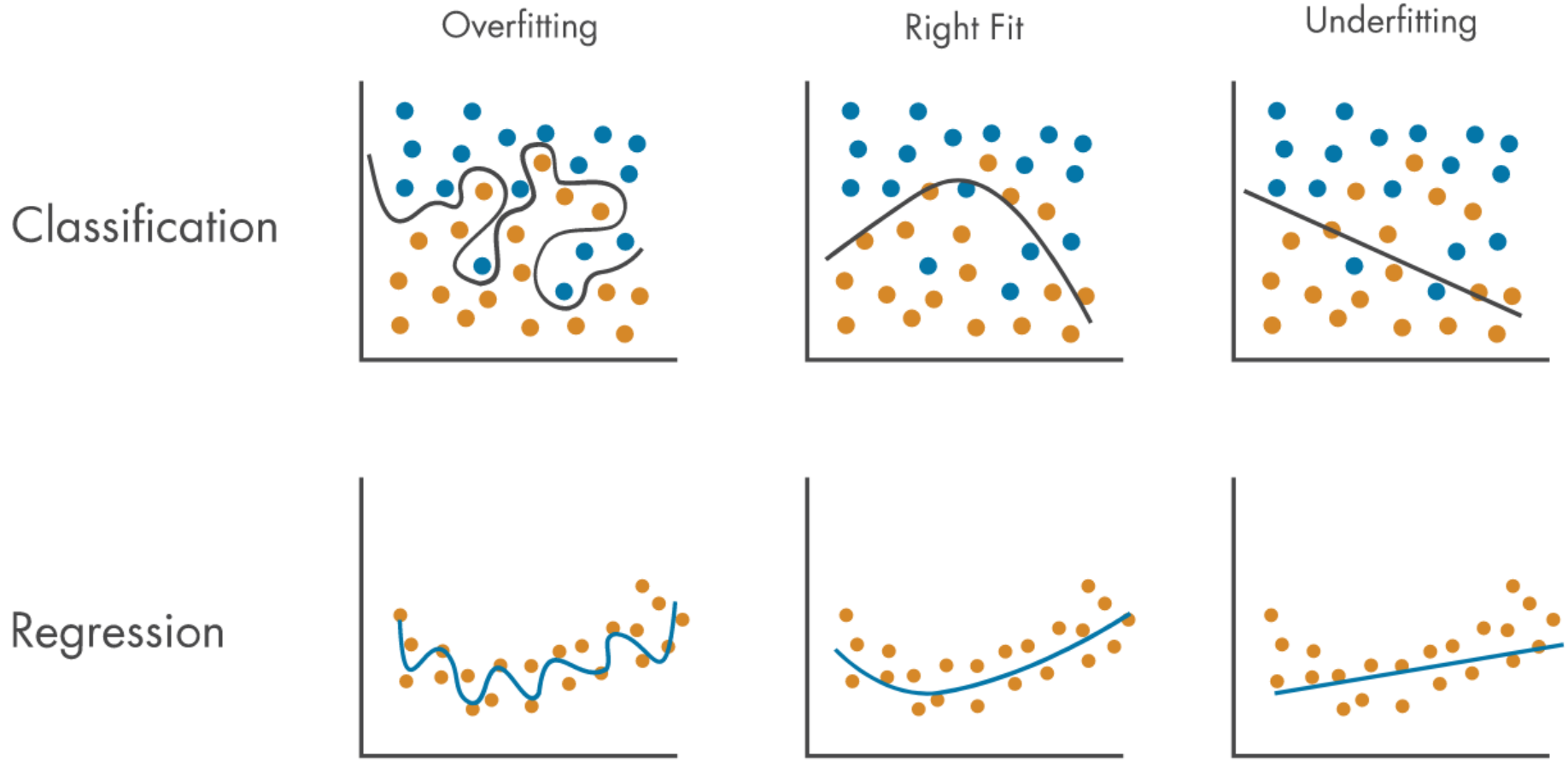
- **SGD** (Stochastic Gradient Descent)
- **NAV** (Nesterov Accelerated Gradient, i.e. SGD with Nesterov Momentum)
- **Adagrad** (Adaptive Gradient Algorithm): automatic adaptation of the learning rate during the training
- **Adam** (Adaptive Momentum Estimation): by using a decay memory of the past gradients, the adaptation is based on the mean and squared mean of the past gradients
- **L-BFGS** (Limited-memory Broyden-Fletcher-Goldfar-Shanno): it estimates and uses the Hessian

Optimizer	Pro	Con
SGD	Easy, fast	Slow Convergence, local minima problem
NAG	Stable, tries to avoid local minima	Require manual synchronization of the learning rate
Adagrad	Not require learning rate reduction	Learning rate decays fast, it could “dye”
Adam	Fast, robust	Affected by local minima problem
L-BFGS	Extremely precise	Computational expensive, not suitable for very deep networks

Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.



Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

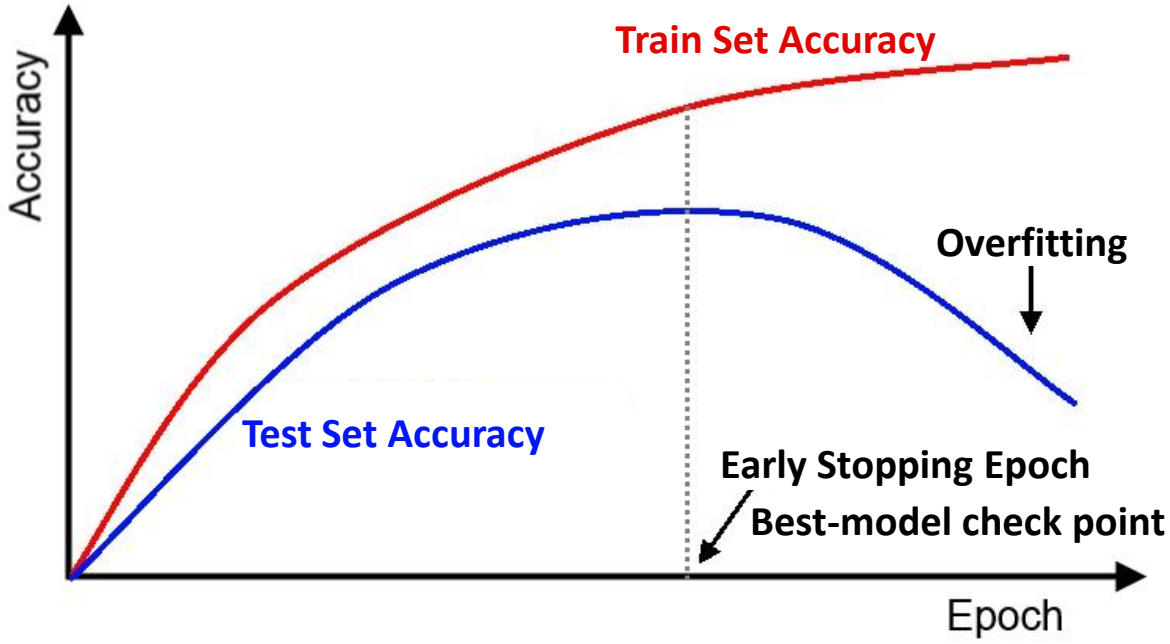
Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization)
- Dropout

- Intruding adversarial examples

- Many other approaches (see, e.g, Goodfellow +2016)



Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization) → $C(y, \bar{y}\,|w_j, b_j) = Loss(y, \bar{y}\,|w_j, b_j) + \lambda \sum_{j=1}^{M} w_j^2$
- Dropout

e.g. MAE, MSE, Cross-Entropy — **L2-term**

- Intruding adversarial examples

- Many other approaches (see, e.g, Goodfellow +2016)

Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization) →
- Dropout

e.g. MAE, MSE, Cross-Entropy **L2-term**

$$C(y, \bar{y}\,|w_j, b_j) = Loss(y, \bar{y}\,|w_j, b_j) + \lambda \sum_{j=1}^{M} w_j^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial L}{\partial w} + 2\lambda w \qquad w^{(t+1)} = w^{(t)} - \eta \frac{\partial C}{\partial w}$$

- Intruding adversarial examples

- Many other approaches (see, e.g, Goodfellow +2016)

$$w^{(t+1)} = w^{(t)} - \eta \left(\frac{\partial L}{\partial w} + \lambda' w^{(t)} \right)$$

Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization) →
- Dropout

e.g. MAE, MSE, Cross-Entropy — **L2-term**

$$C(y, \bar{y}\,|w_j, b_j) = Loss(y, \bar{y}\,|w_j, b_j) + \lambda \sum_{j=1}^{M} w_j^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial L}{\partial w} + 2\lambda w \qquad w^{(t+1)} = w^{(t)} - \eta \frac{\partial C}{\partial w}$$

- Intruding adversarial examples

- Many other approaches (see, e.g, Goodfellow +2016)

$$w^{(t+1)} = (1 - \eta\lambda')w^{(t)} - \eta \frac{\partial L}{\partial w}$$

Decay term: weights contraction, before the gradient subtraction

Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization) →
- Dropout

e.g. MAE, MSE, Cross-Entropy — **L2-term**

$$C(y, \bar{y}\,|w_j, b_j) = Loss(y, \bar{y}\,|w_j, b_j) + \lambda \sum_{j=1}^{M} w_j^2$$

$$w^{(t+1)} = (1 - \eta\lambda')w^{(t)} - \eta \frac{\partial L}{\partial w}$$

- Intruding adversarial examples

- Many other approaches (see, e.g, Goodfellow +2016)

Weight Decay:
- the weights remain small, it prevents the network from overreacting to small changes in the input
- it encourages the network to distribute the weight more evenly
- smaller weights tend to keep the network in a more “linear” regime, making the learning curve smoother

Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization)
- Dropout

- Intruding adversarial examples

- Many other approaches (see, e.g, Goodfellow +2016)

Regularization techniques

Overfitting: is the tendency of an algorithm to extract more information than necessary to capture the noise-less signal from the training set, which results to a hyper-specialization, preventing the gain of generalization.

Underfitting: is the condition that occurs when the model is not able to extract abstract properties during the training phase; in it could be the result of insufficient capacity, insufficient training, or insufficient information retention.

Underfitting and overfitting can be prevented with **regularization techniques**:

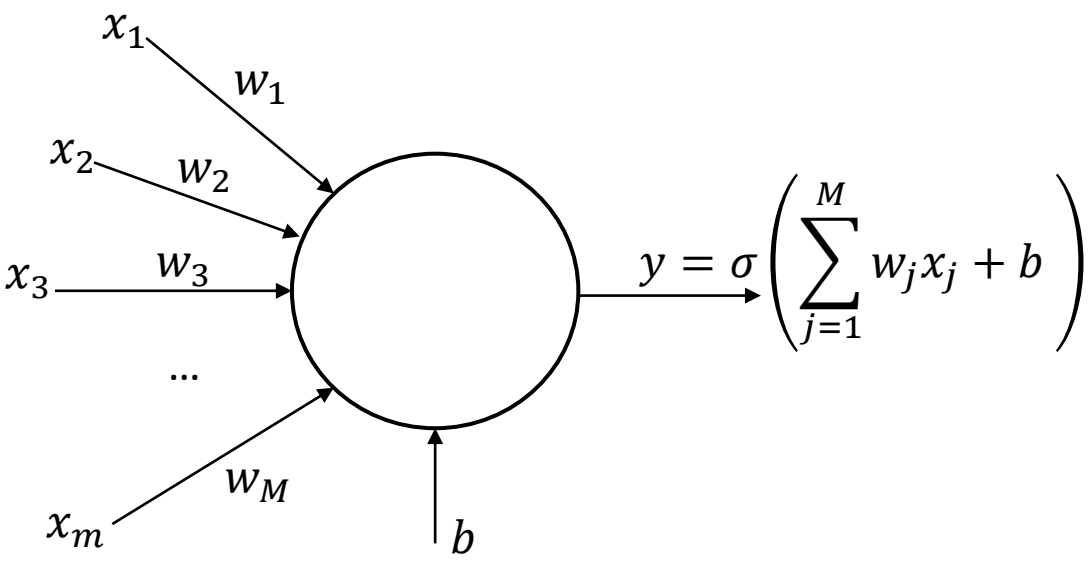
- Early stopping criteria;
- Best-model check point;
- Reducing the learning-rate (as a function of the epochs);
- Weight decay (a.k.a. L2 regularization)
- Dropout

- Intruding adversarial examples

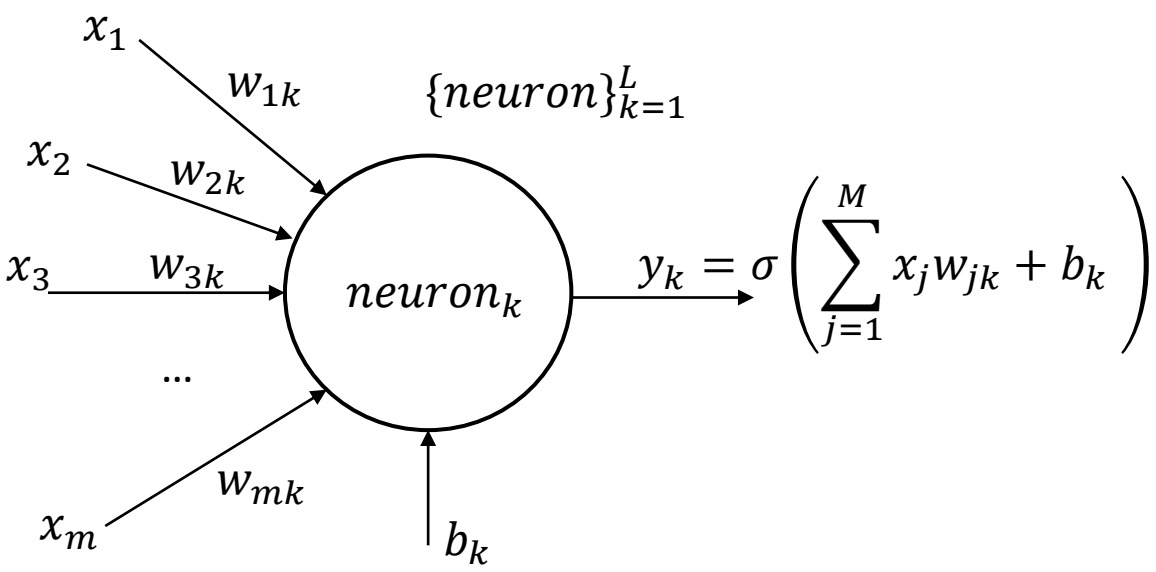
- Many other approaches (see, e.g, Goodfellow +2016)

Adversarial examples in the dataset:
- Prevent hyper specialization on the training set
- Methods are trained on real world complex situations

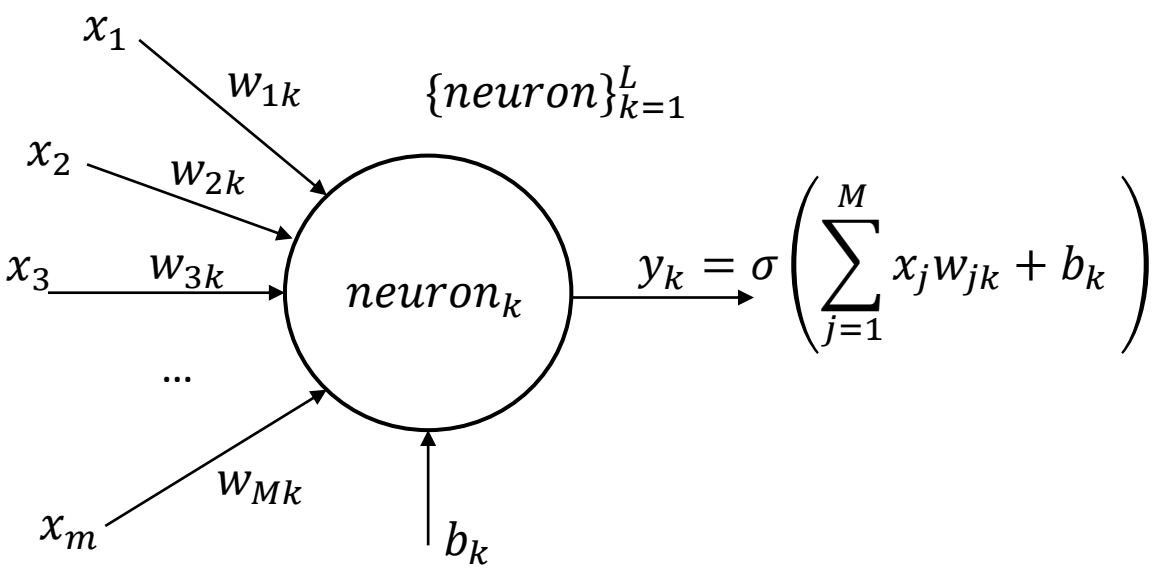
The Multi-Layer Perceptron

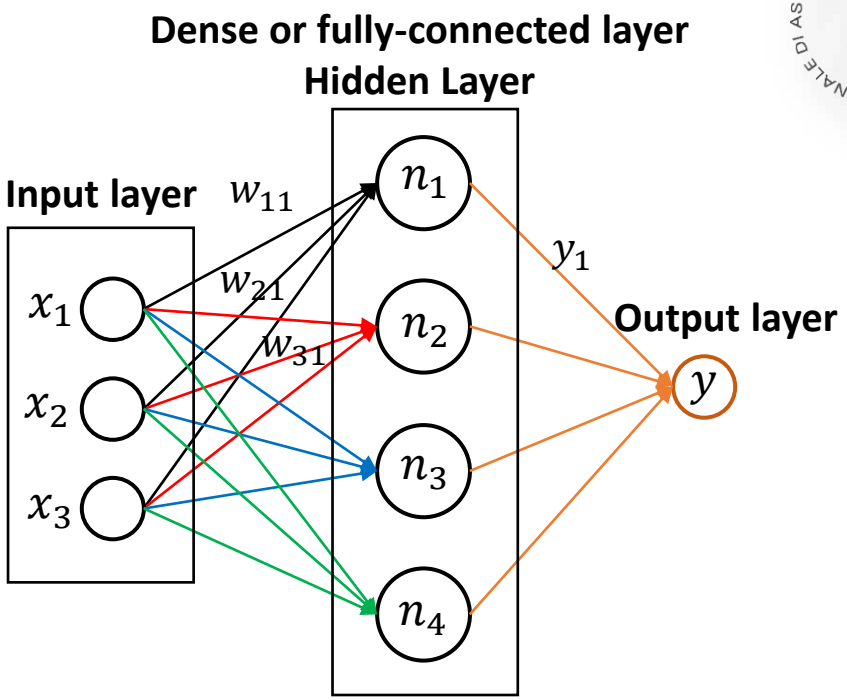


The Multi-Layer Perceptron



The Multi-Layer Perceptron



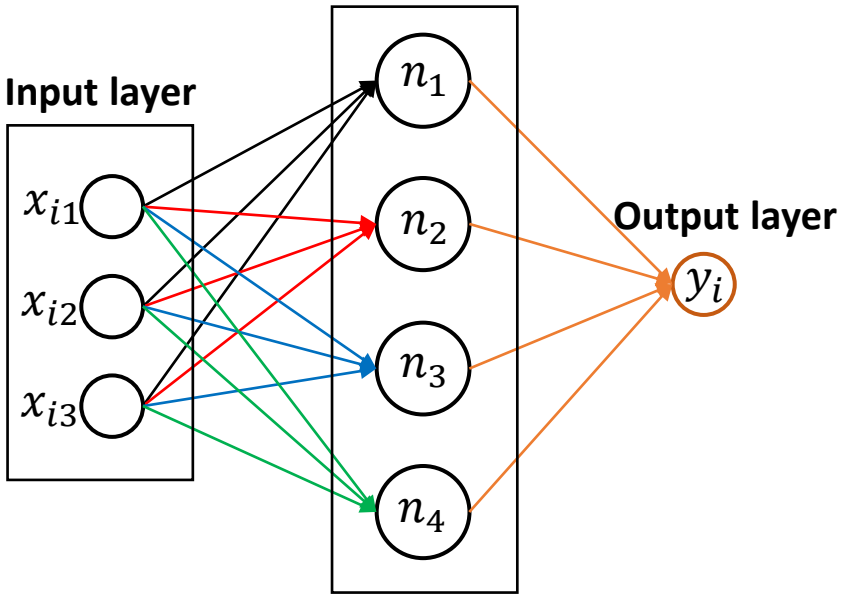


The Multi-Layer Perceptron

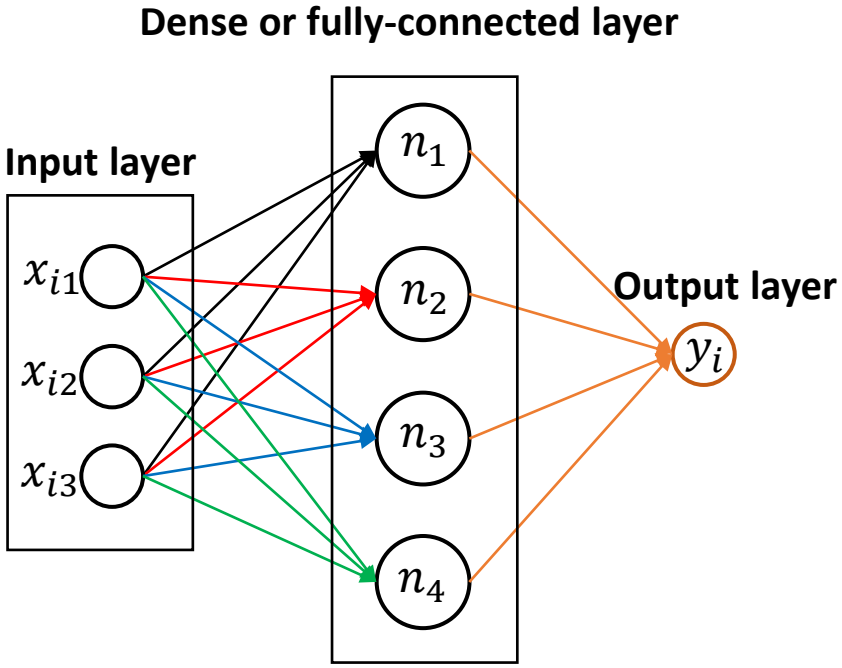
The input is represented as a matrix,
i.e. N samples, each of them is an M-dimensional vector, (N, M)

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nM} \end{pmatrix}$$

Dense or fully-connected layer



The Multi-Layer Perceptron



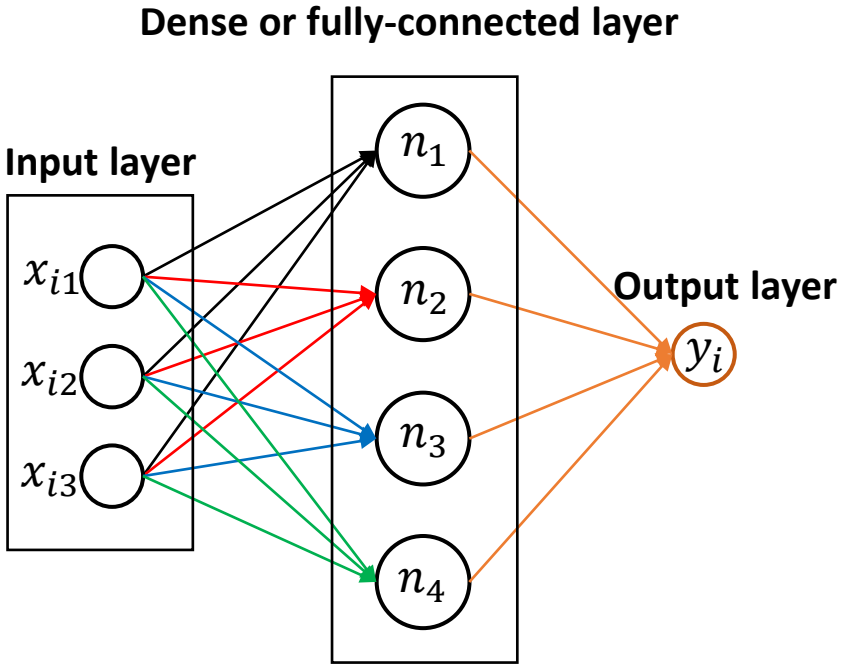
The input is represented as a matrix,
i.e. N samples, each of them is an m-dimensional vector, (N, M)

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nM} \end{pmatrix}$$

Similarly, the weights can be represented as a matrix,
i.e. L neurons, each of them is a M-dimensional vector, (M, L)

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1L} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{ML} \end{pmatrix}$$

The Multi-Layer Perceptron



The input is represented as a matrix,
i.e. N samples, each of them is an m-dimensional vector, (N, M)

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NM} \end{pmatrix}$$

Similarly, the weights can be represented as a matrix,
i.e. L neurons, each of them is a m-dimensional vector, (M, L)

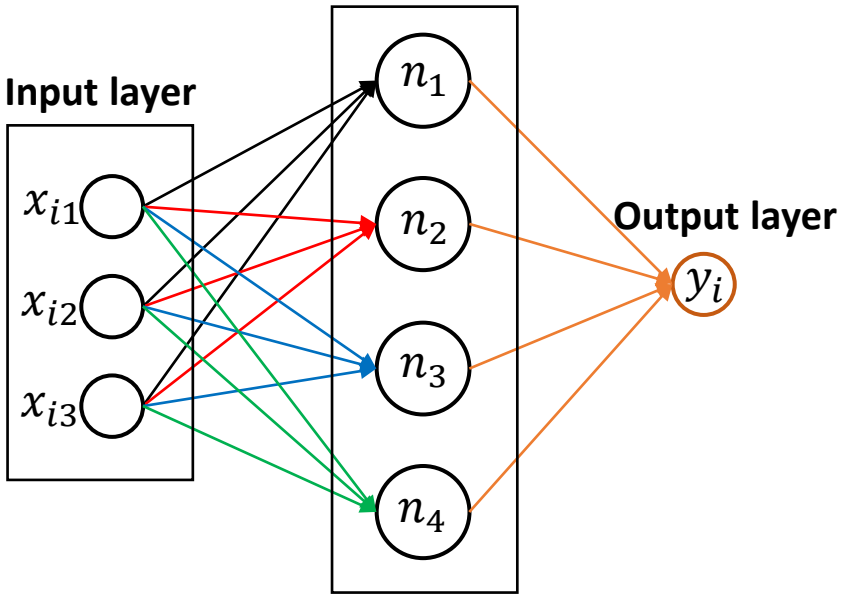
$$W = \begin{pmatrix} w_{11} & \cdots & w_{1L} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{ML} \end{pmatrix}$$

Dense layer equation:

$$y = \begin{pmatrix} y_1 \\ \cdots \\ y_i \\ \cdots \\ y_N \end{pmatrix} = X \cdot W + b = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NM} \end{pmatrix} \begin{pmatrix} w_{11} & \cdots & w_{1L} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{ML} \end{pmatrix} \oplus \begin{pmatrix} b_1 \\ \cdots \\ b_k \\ \cdots \\ b_L \end{pmatrix}$$

The Multi-Layer Perceptron

Dense or fully-connected layer



The input is represented as a matrix,
i.e. N samples, each of them is an m-dimensional vector, (N, M)

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nM} \end{pmatrix}$$

Similarly, the weights can be represented as a matrix,
i.e. L neurons, each of them is a m-dimensional vector, (M, L)

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1L} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{ML} \end{pmatrix}$$

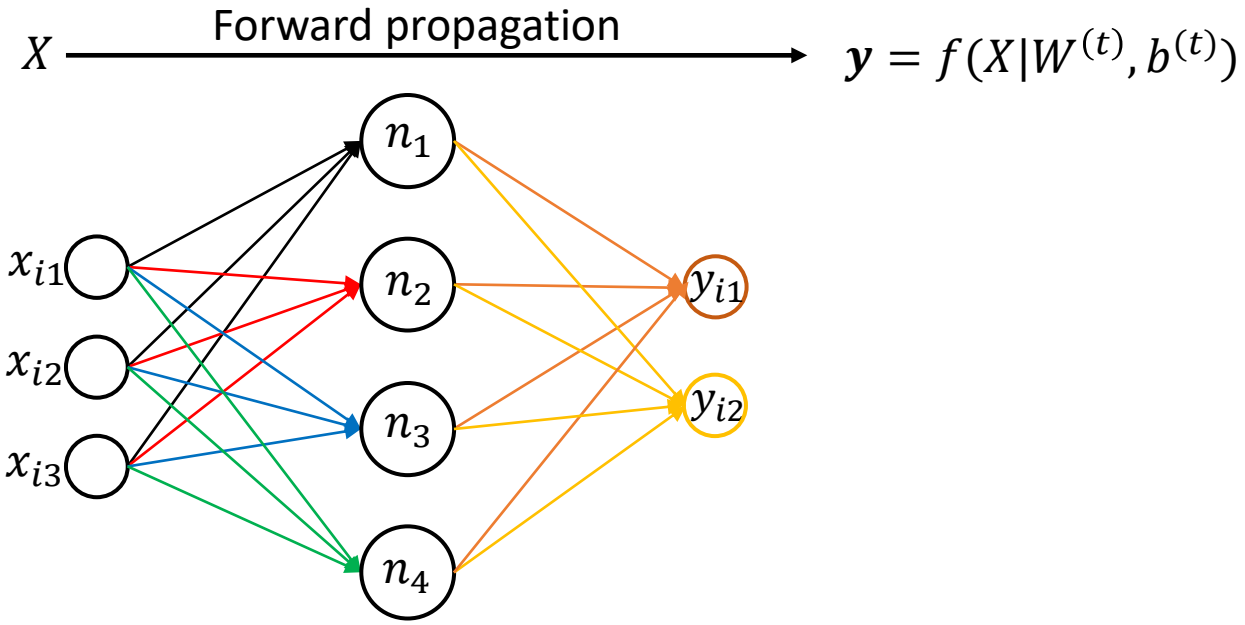
Dense layer equation:

$$y = \begin{pmatrix} y_1 \\ \cdots \\ y_i \\ \cdots \\ y_n \end{pmatrix} = X \cdot W + b = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nM} \end{pmatrix} \begin{pmatrix} w_{11} & \cdots & w_{1L} \\ \vdots & \ddots & \vdots \\ w_{M1} & \cdots & w_{ML} \end{pmatrix} \oplus \begin{pmatrix} b_1 \\ \cdots \\ b_k \\ \cdots \\ b_L \end{pmatrix}$$

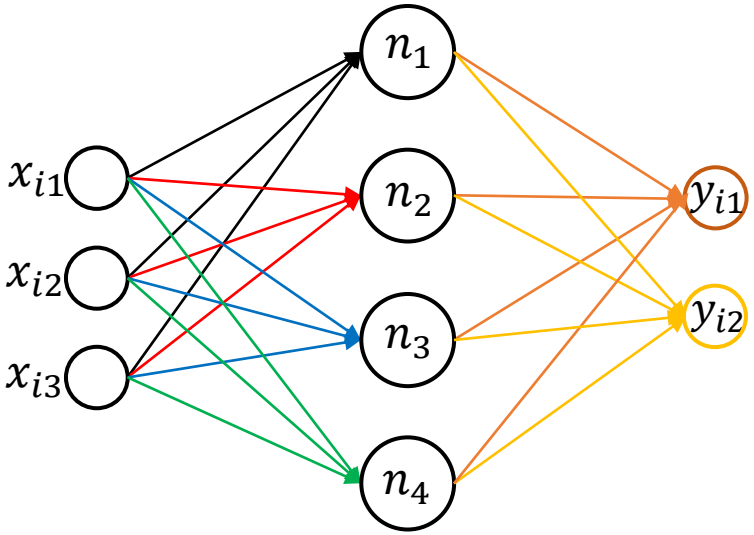
→ $y_i = x_{ij} w_{jk} + b_k$

→ $y_i = \sigma(x_{ij} w_{jk} + b_k)$ (by including the activation function, σ)

Adaptation mechanism: forward-backward propagation



Adaptation mechanism: forward-backward propagation

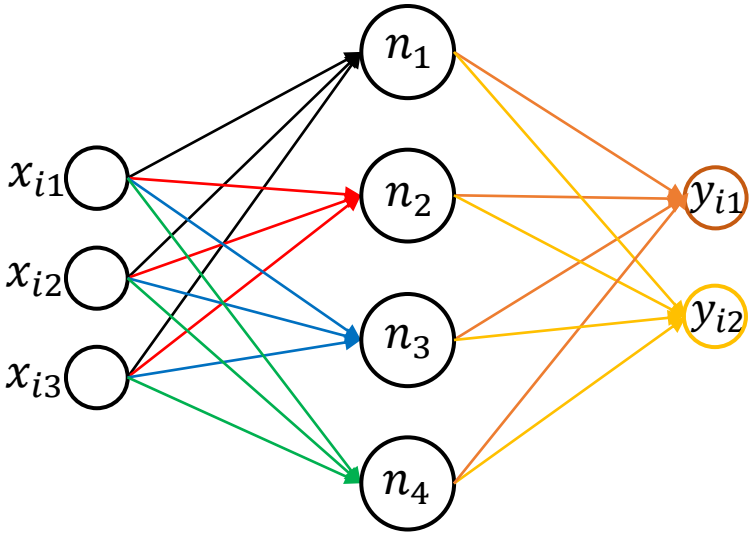


Ground Truth $= \bar{\boldsymbol{y}}$

$\boldsymbol{y} = f(X|W^{(t)}, b^{(t)}) \longrightarrow C(\boldsymbol{y}, \bar{\boldsymbol{y}}\,|W^{(t)}, b^{(t)})$

e.g. MSE, MAE,
Cross entropy

Adaptation mechanism: forward-backward propagation



X → Forward propagation → $\boldsymbol{y} = f(X|W^{(t)}, b^{(t)})$ → $C(\boldsymbol{y}, \bar{\boldsymbol{y}}\,|W^{(t)}, b^{(t)})$

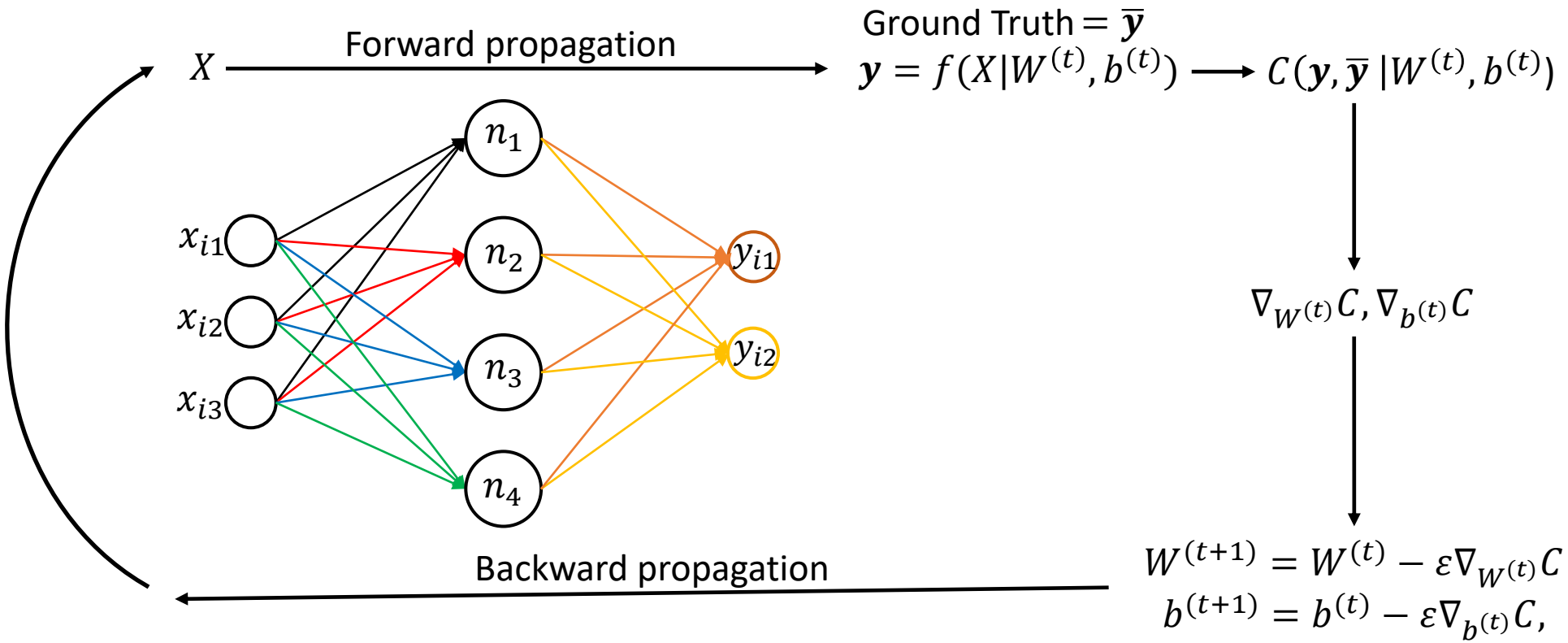
Ground Truth $= \bar{\boldsymbol{y}}$

$\nabla_{W^{(t)}} C, \nabla_{b^{(t)}} C$

$$W^{(t+1)} = W^{(t)} - \varepsilon \nabla_{W^{(t)}} C$$
$$b^{(t+1)} = b^{(t)} - \varepsilon \nabla_{b^{(t)}} C,$$

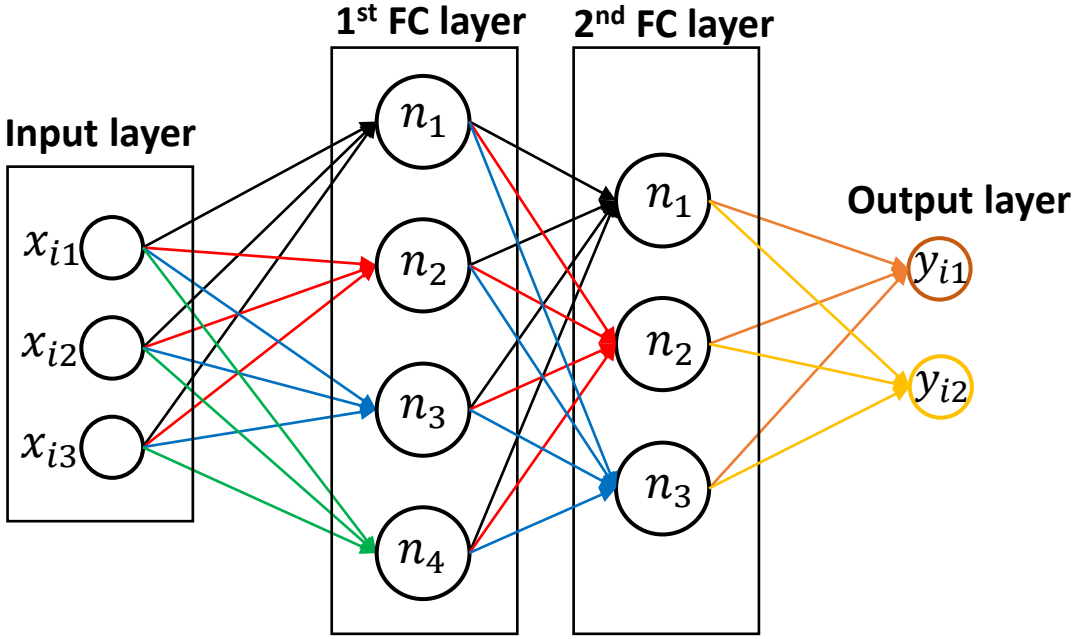
Backward propagation

Adaptation mechanism: forward-backward propagation

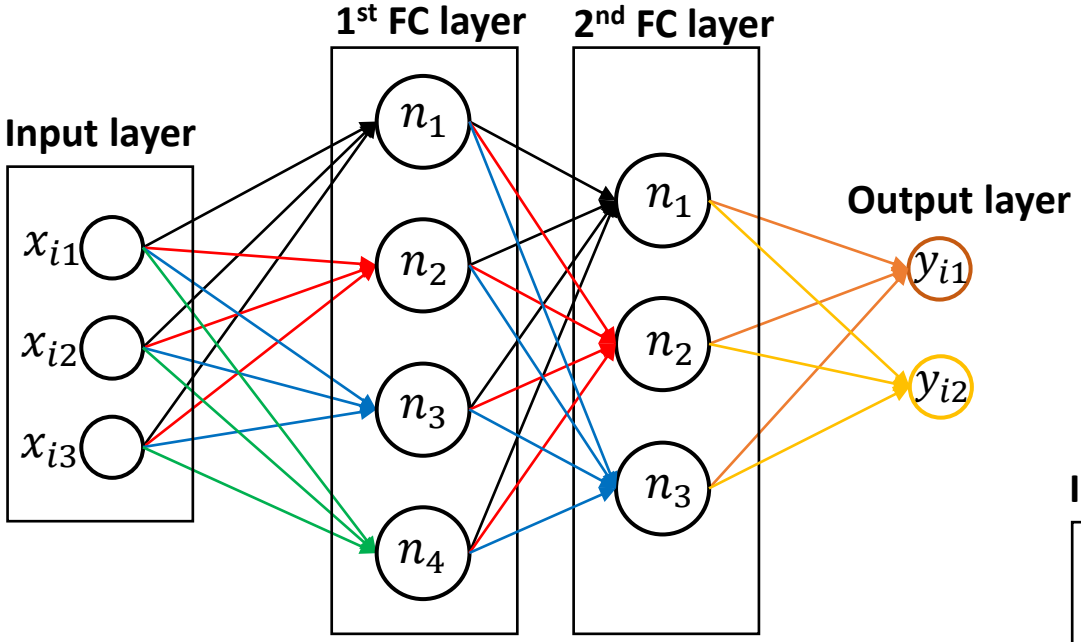


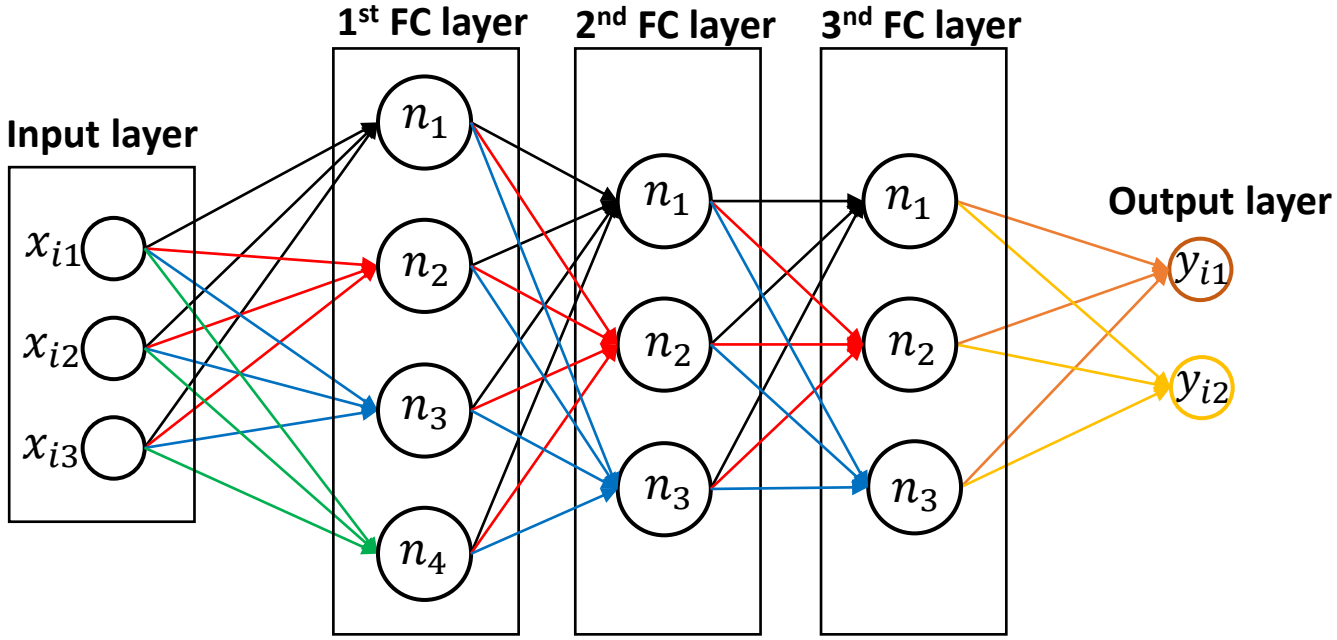
This the **optimization** process, based on which weights and biases are adapted.

The Multi-Layer Perceptron(s)

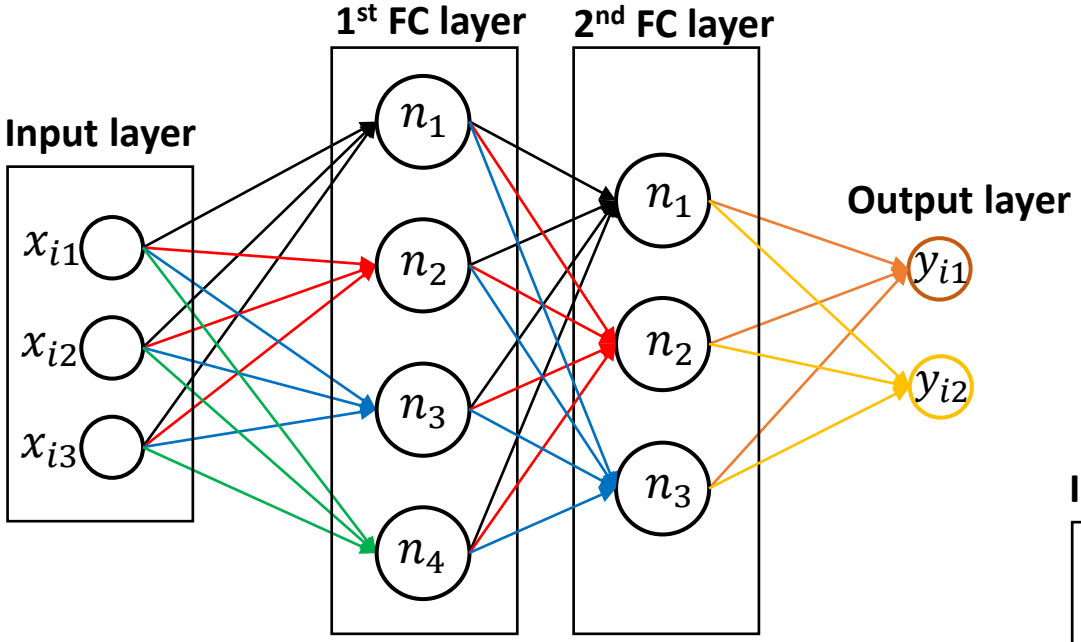


The Multi-Layer Perceptron(s)

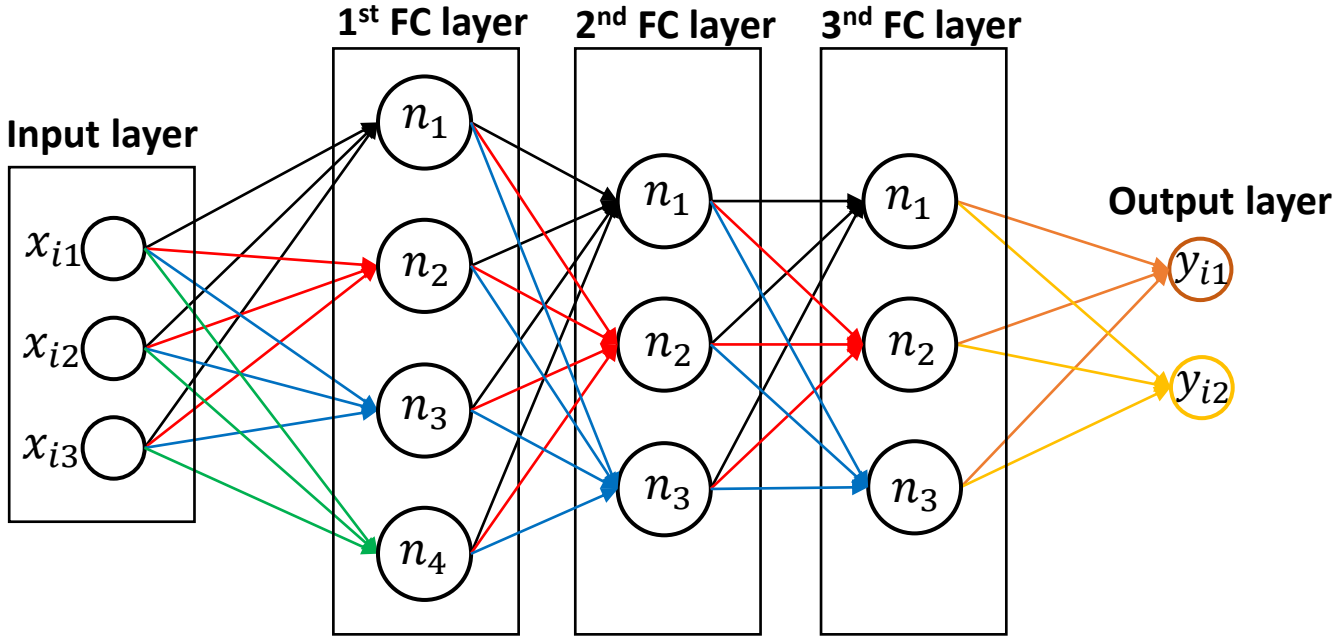




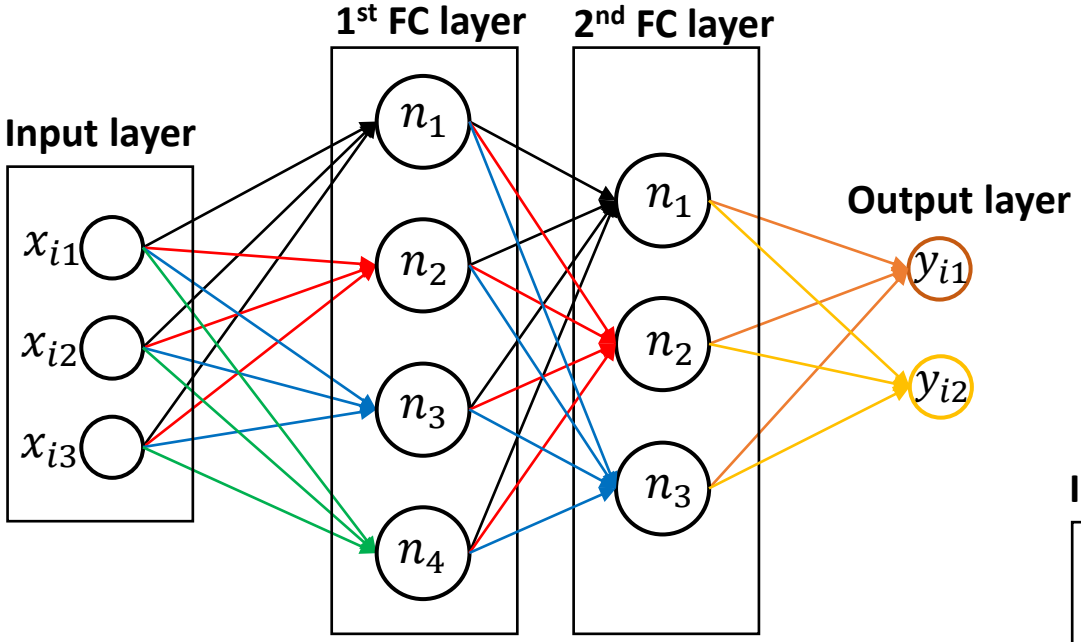
The Multi-Layer Perceptron(s)



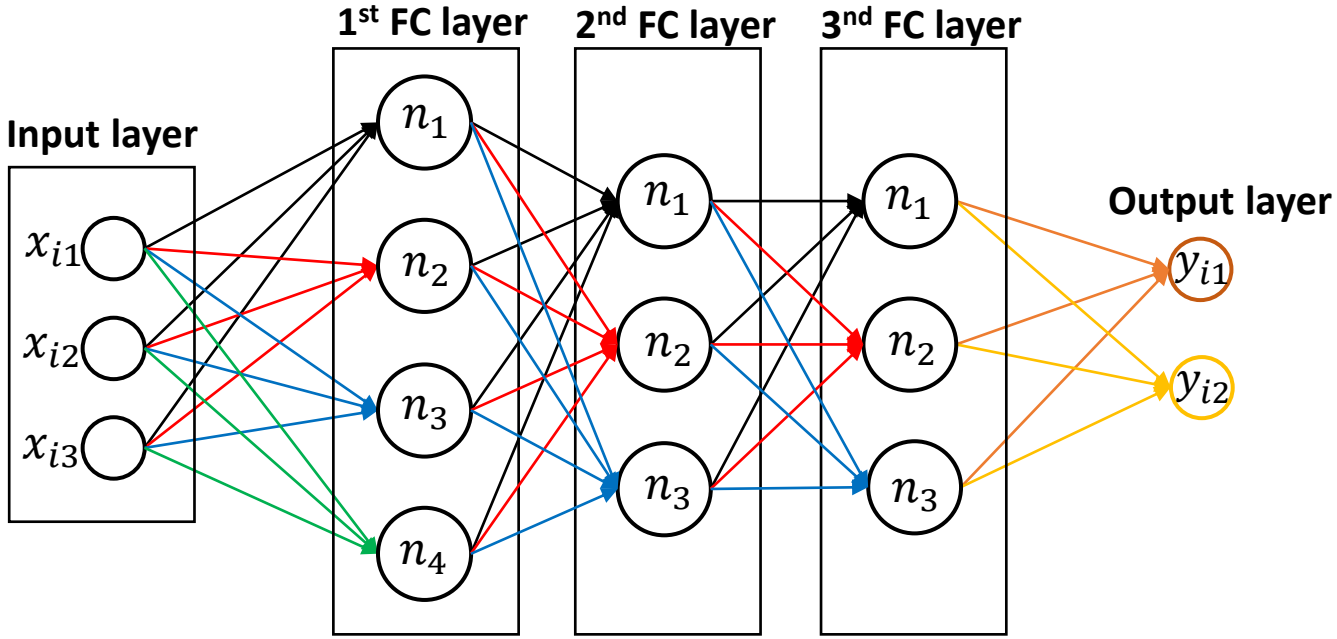
What is the correct setting?



The Multi-Layer Perceptron(s)



What is the correct setting?
There is no a universally right answer…
Only empirical rules and… a theorem



Hyper-parameter setup

Universal Approximation Theorem (applied to an MLP):
an MLP with one hidden layer can approximate any continuous function
(with a non-polynomial activation, e.g. ReLU, sigmoid)

Its corollary:
An MLP with any number of hidden layers can be approximated with an MLP composed by
only two hidden layers and an *adequate* number of neurons per layer

An empirical rule (1):
If the input has M dimensions
the first layer has: $\mathbf{2 \cdot M - 1}$
the second layer has: $\mathbf{M - 1}$

An empirical rule (2):
If the input has M dimensions, and the output has K dimensions
the first layer has: $\frac{\mathbf{2}}{\mathbf{3}} \cdot \boldsymbol{M} + \boldsymbol{K}$
the second layer has: $\frac{\mathbf{1}}{\mathbf{2}} \boldsymbol{M} + \boldsymbol{K}$

Hyper-parameter setup

Universal Approximation Theorem (applied to an MLP):
an MLP with one hidden layer can approximate any continuous function
(with a non-polynomial activation, e.g. ReLU, sigmoid)

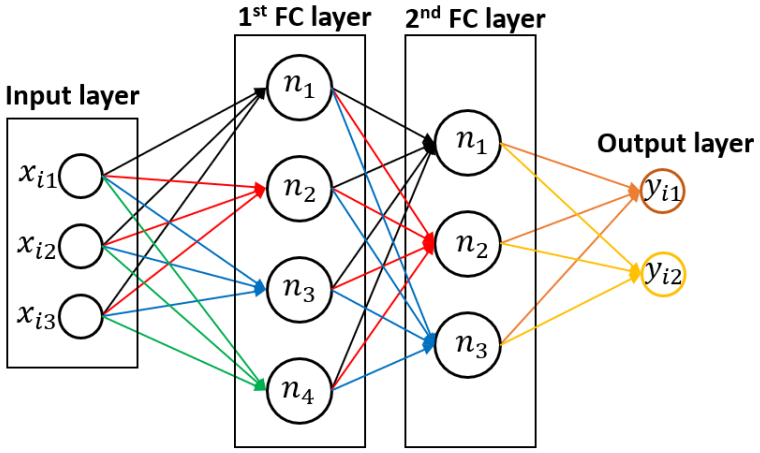
Its corollary:
An MLP with any number of hidden layers can be approximated with an MLP composed by
only two hidden layers and an *adequate* number of neurons per layer

An empirical rule (1):
If the input has M dimensions
the first layer has: $\mathbf{2 \cdot M - 1}$
the second layer has: $\mathbf{M - 1}$

An empirical rule (2):
If the input has M dimensions, and the output has K dimensions
the first layer has: $\frac{2}{3} \cdot \boldsymbol{M} + \boldsymbol{K}$
the second layer has: $\frac{1}{2} \boldsymbol{M} + \boldsymbol{K}$

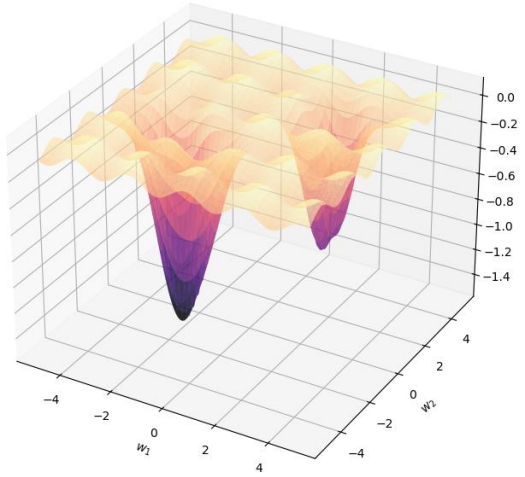
Hyper-parameter setup optimization:
- Grid search
- Bayes algorithm
- Genetic algorithm
- many others approaches

Compiling a network



Building the network architecture, e.g.:

define the number of hidden layers
the number of neuron per hidden layers,
the activation functions
the regularization technique(s)



Loss function (Supervised version):

a measure of the discrepancy between the model's prediction and the target value
(it requires the setting of a learning rate and/or a learning rate decay rule)

Optimizers:

the algorithm that minimizes the loss function
by updating the weights and biases during the training

Metrics:

measures (different from the loss function) of the discrepancy between the model's prediction and the target value

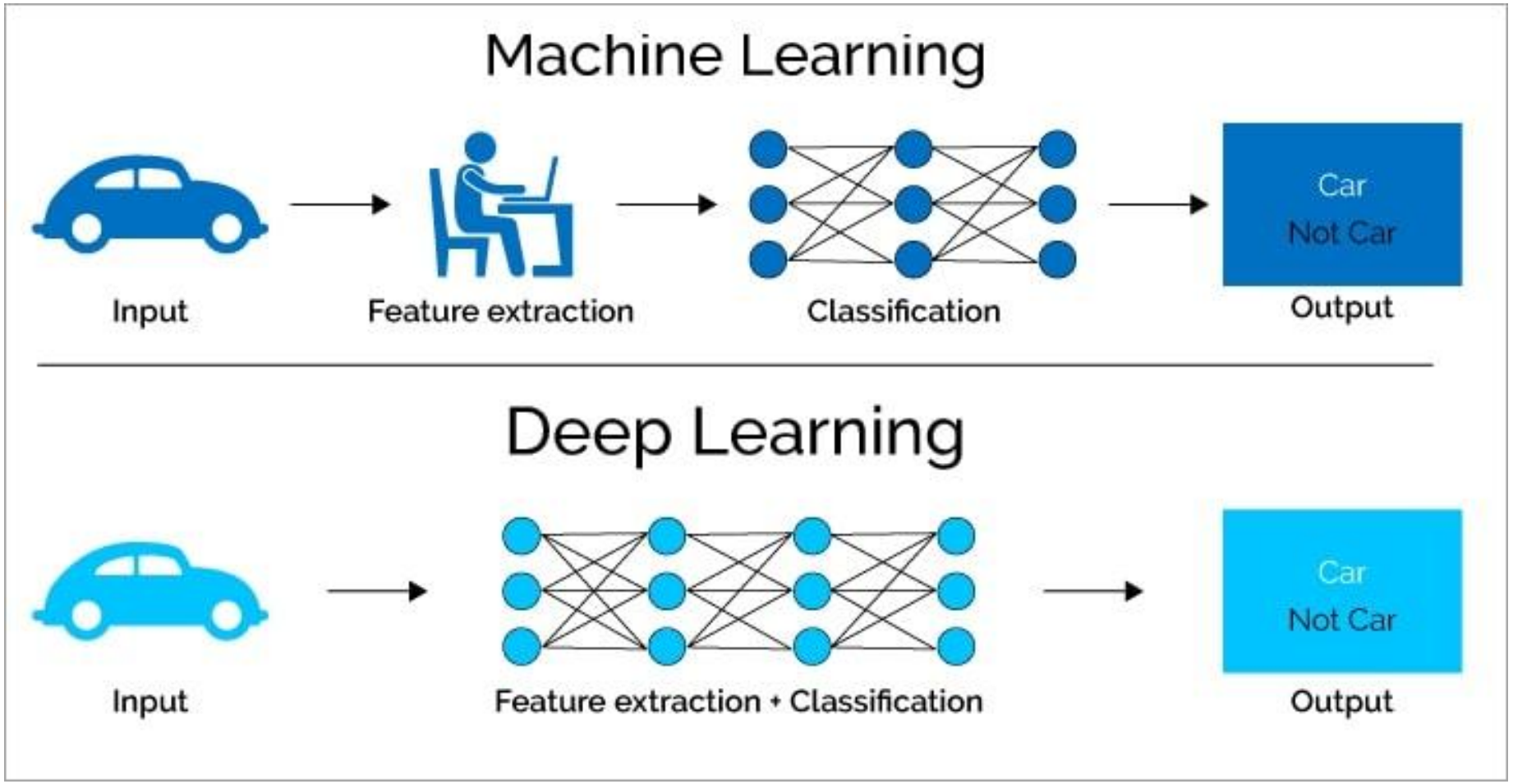
Example for a <u>classification problem</u>:

Loss function: Cross-Entropy
Metric: Accuracy
Optimizer: SGD

GO TO NOTEBOOK FOR THE MLP CODING INTRODUCTION

Deep Learning

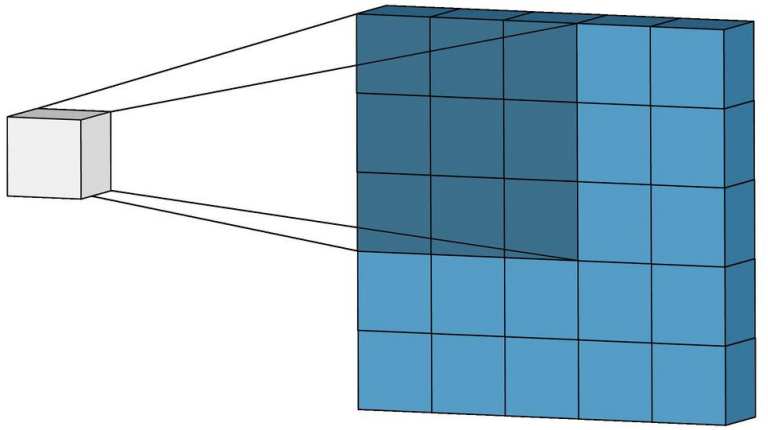
There are several differences between machine learning and deep learning (as well as similarities). The main difference is their **intrinsic capability of deep learning methods to automatically extract features** (avoiding the feature computing and selection), thus they are suitable to process raw data.



The element of Deep Learning methods: Convolution

In a convolutional neural network (CNN), **convolutions** replace the matrix products. The weights become **filters** (kernel), which suppress or emphasize some **characteristics**.

$$X * W\ [l,n]\ = \sum_{k=1}^{K} \sum_{m=1}^{M} X[k,m]W[l-k,n-m]$$



Thus the same kernel processes the input → saving memory
Multiple kernels on the same input → each kernel act as “feature extractor”

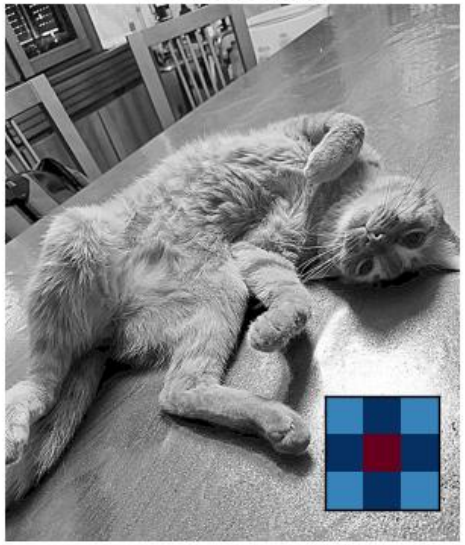
The element of Deep Learning methods: Convolution

Convolution examples

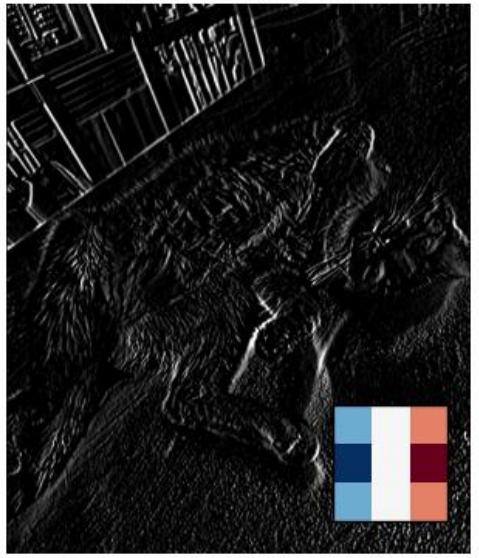
1. Original

2. Smoothing (Blur)

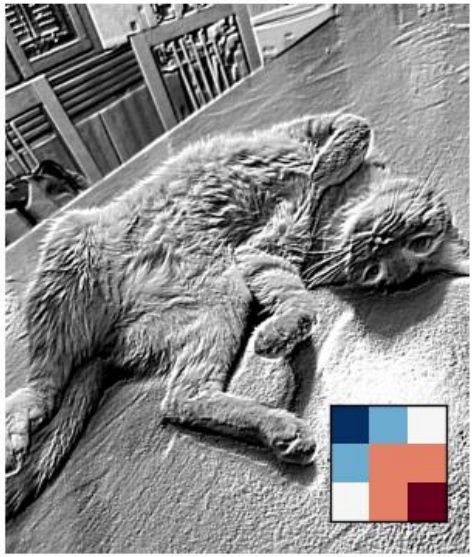
3. Sharpening



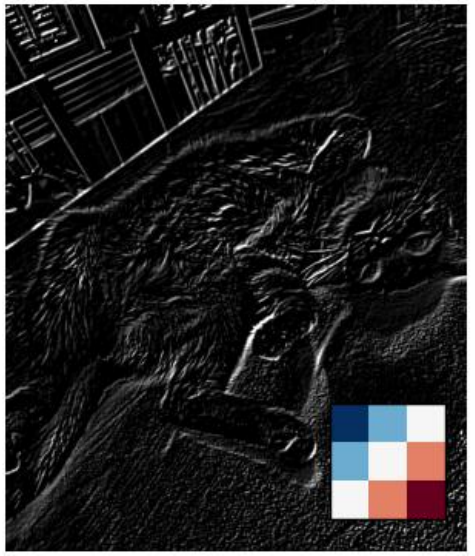
4. Vertical edges



5. Emboss



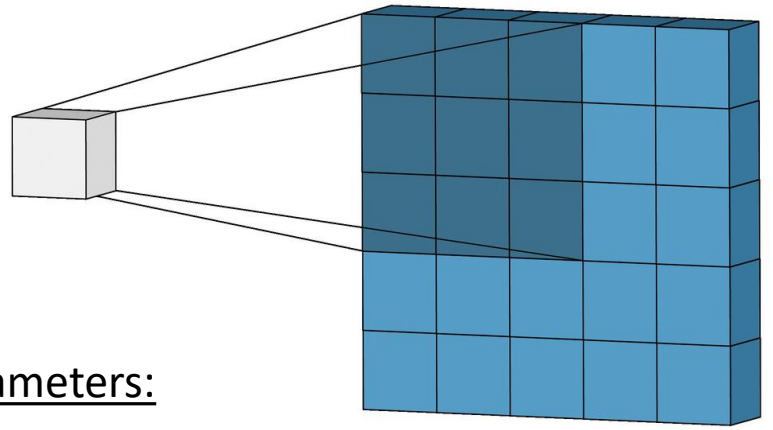
6. Diagonal edges



The element of Deep Learning methods: Convolution

In a convolutional neural network (CNN), **convolutions** replace the matrix products. The weights become **filters** (kernel), which suppress or emphasize some **characteristics**.

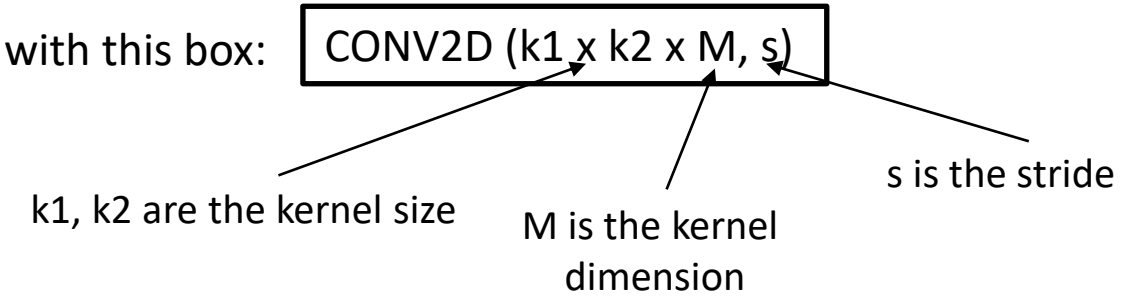
$$X * W\ [l,n]\ = \sum_{k=1}^{K} \sum_{m=1}^{M} X[k,m]W[l-k,n-m]$$



A convolution block can be represented with a set of hyperparameters:

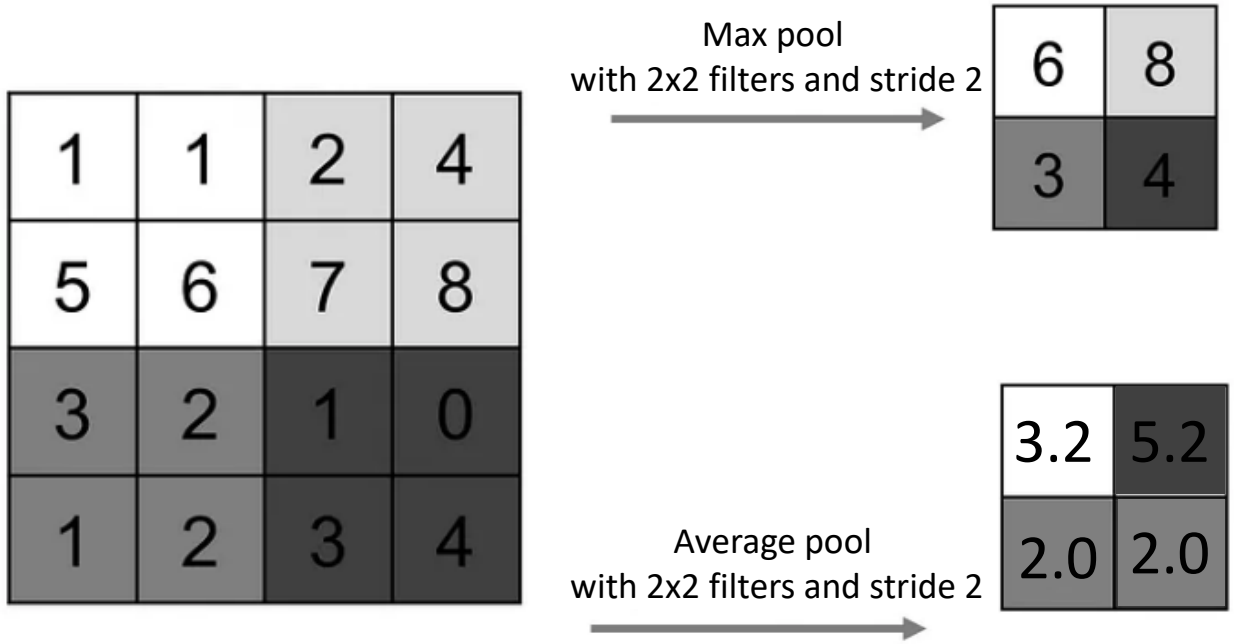
- Kernel size (typically 3x3, 5x5, sometimes 7x7)
- Kernel dimension: the number of kernels (i.e. the number of convolutions), it determines the output dimension.
- Strides: the step of the convolution
- Padding: adding (or not) a zero pad out of the image, so that input and output shape correspond (we always assume **padding = 'same')**.

We are going to represent a convolution operation with this box:



The element of Deep Learning methods: Pooling

In a convolutional neural network (CNN), **pooling** operations (i.e. a subsampling) is used to propagate only the most important **information**, suppressing noise **features** and reducing the complexity.



The element of Deep Learning methods: Pooling

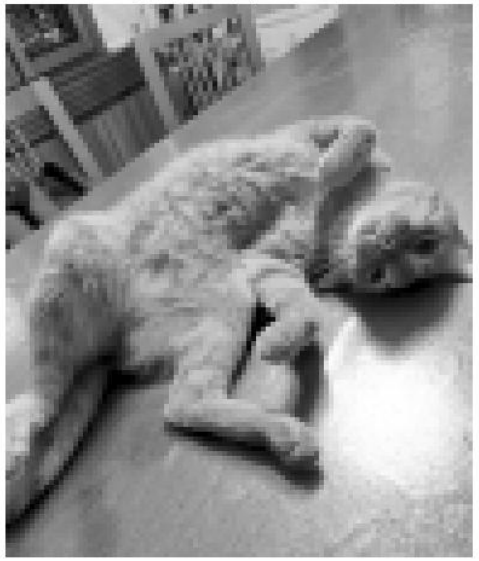
In a convolutional neural network (CNN), **pooling** operations (i.e. a subsampling) is used to propagate only the most important **information**, suppressing noise **features** and reducing the complexity.

Originale

Max Pooling (5x5, stride=5)

Max Pooling (5x5, stride=10)

Average Pooling (5x5, stride=5)



Average Pooling (5x5, stride=10)

The element of Deep Learning methods: Upsampling

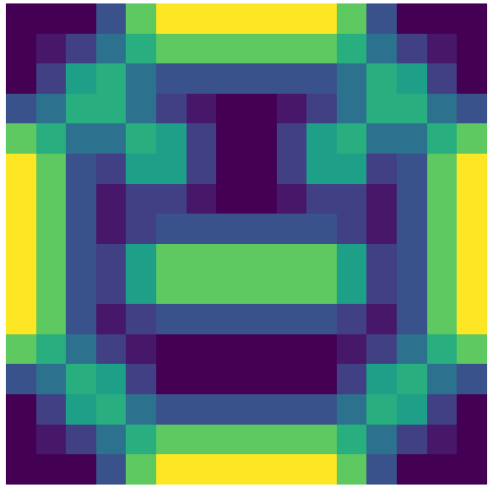
Up-sampling is the opposite mechanism of Pooling. It resize the input by interpolating the ‘new’ pixel values.

1	2	3
4	5	6
7	8	9

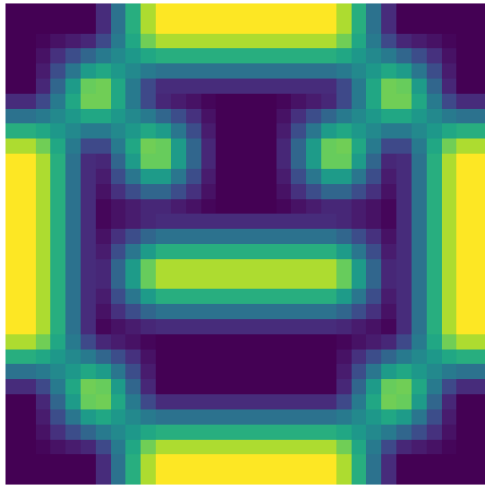
1	1	2	2	3	3
1	1	2	2	3	3
4	4	5	5	6	6
4	4	5	5	6	6
7	7	8	8	9	9
7	7	8	8	9	9

Original image

2x Upsampling

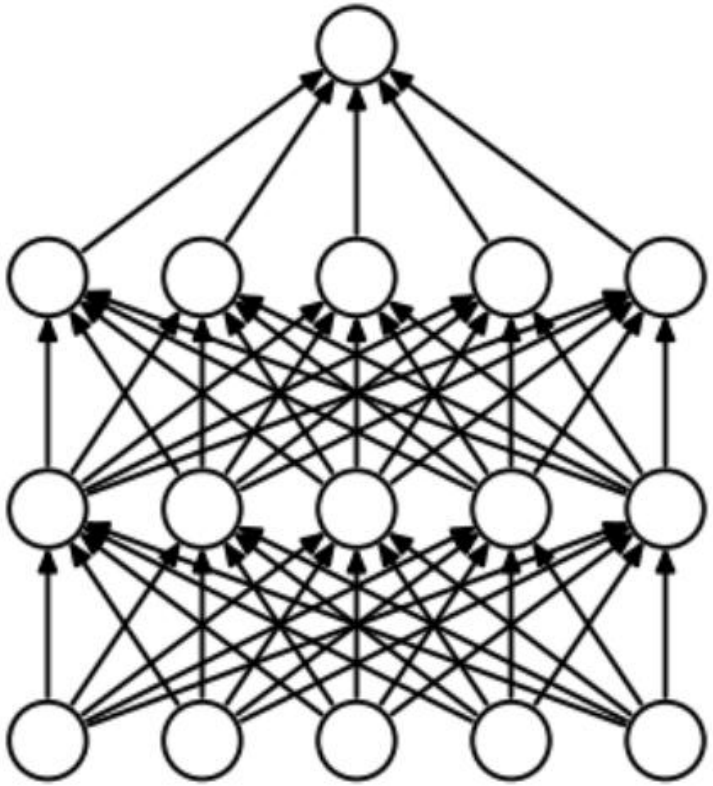


4x Upsampling

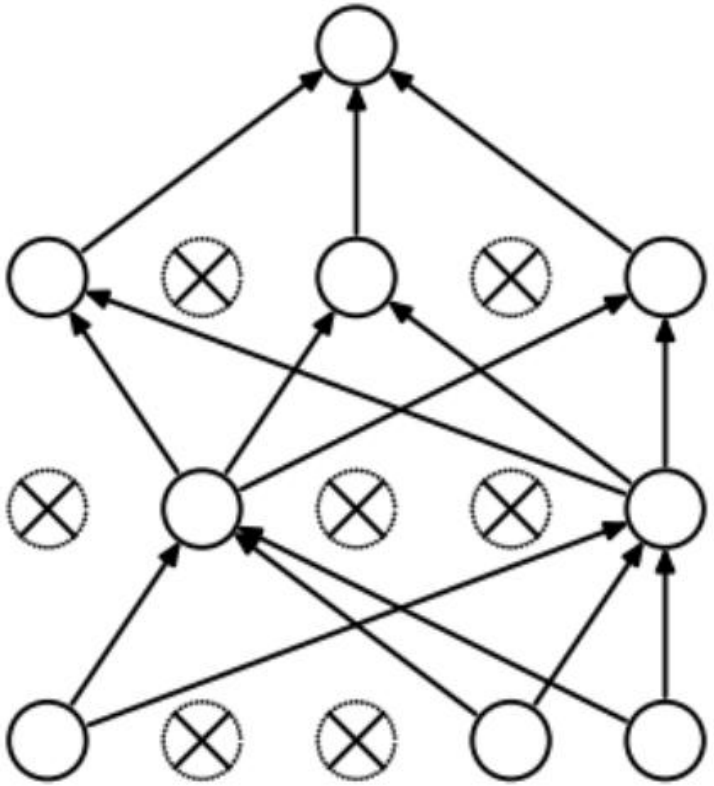


The element of Deep Learning methods: Dropout

Dropout is regularization technique which randomly ‘drops out’ (i.e. deactivate) neurons during the training. This ‘connections pruning’ prevents neurons from relying one specific neighbors (a.k.a co-adaptation problem).



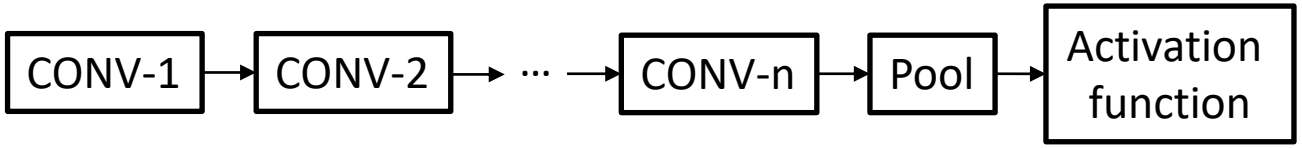
Standard Fully Connected Network



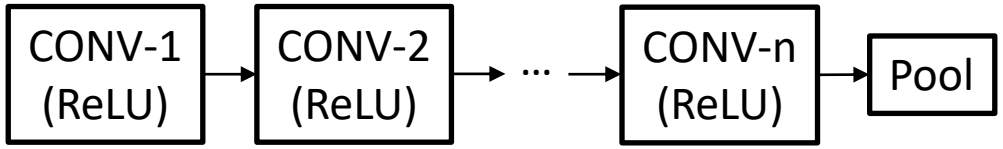
After the dropout

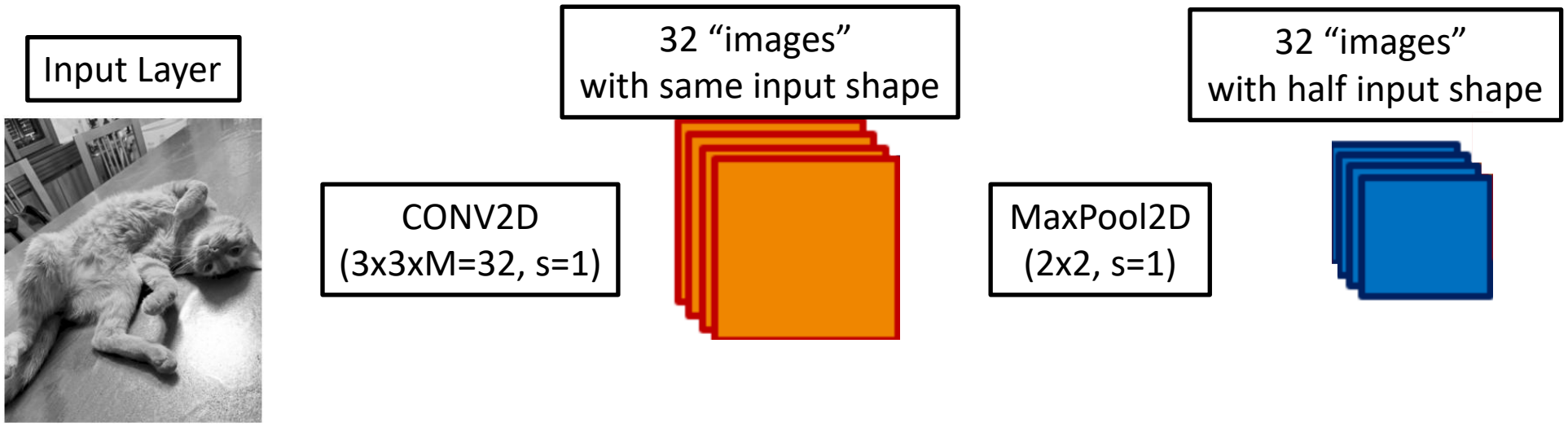
Building a Convolutional Neural Network (CNN):
the simplest block of the chain

The simplest block in a CNN is chain like this:



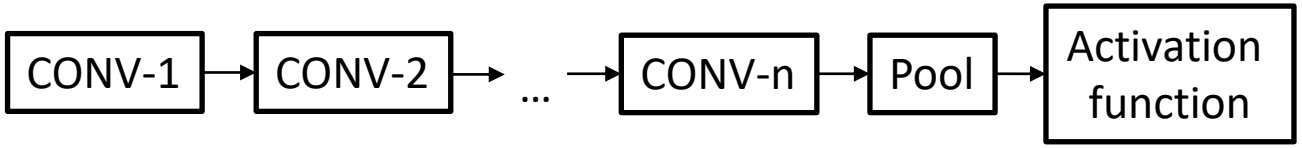
Or, the activation function can be applied after each convolution (e.g. a ReLU):



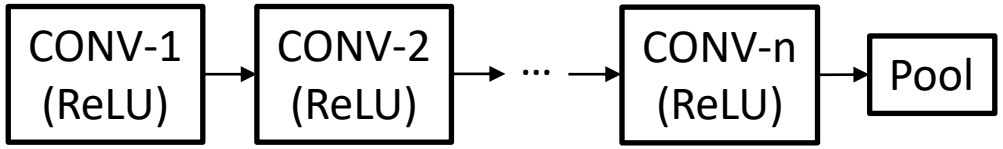


Building a Convolutional Neural Network (CNN):
the simplest block of the chain

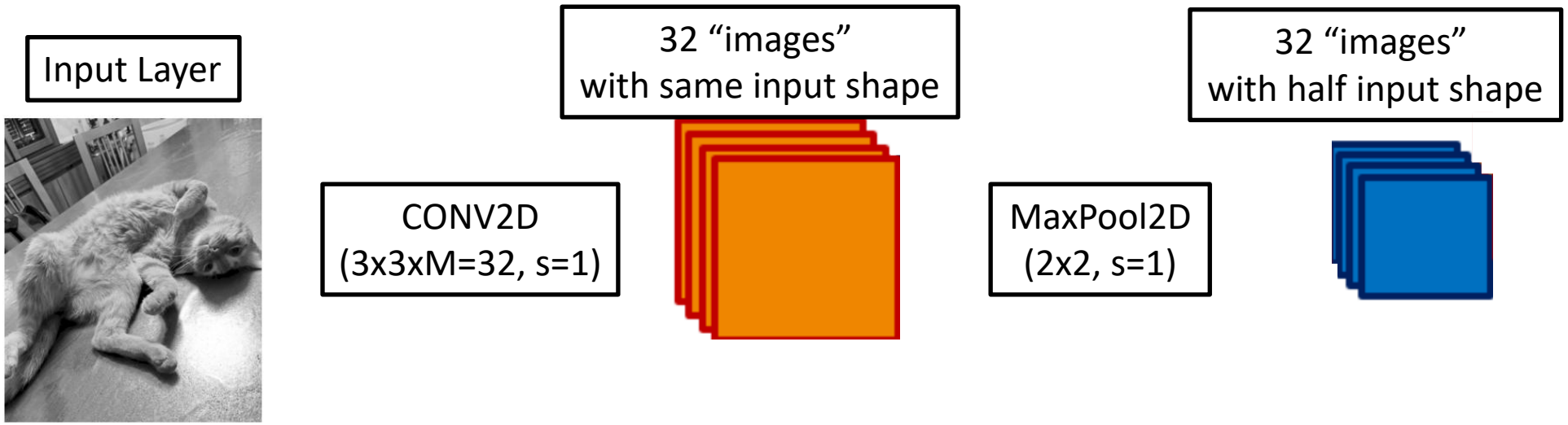
The simplest block in a CNN is chain like this:



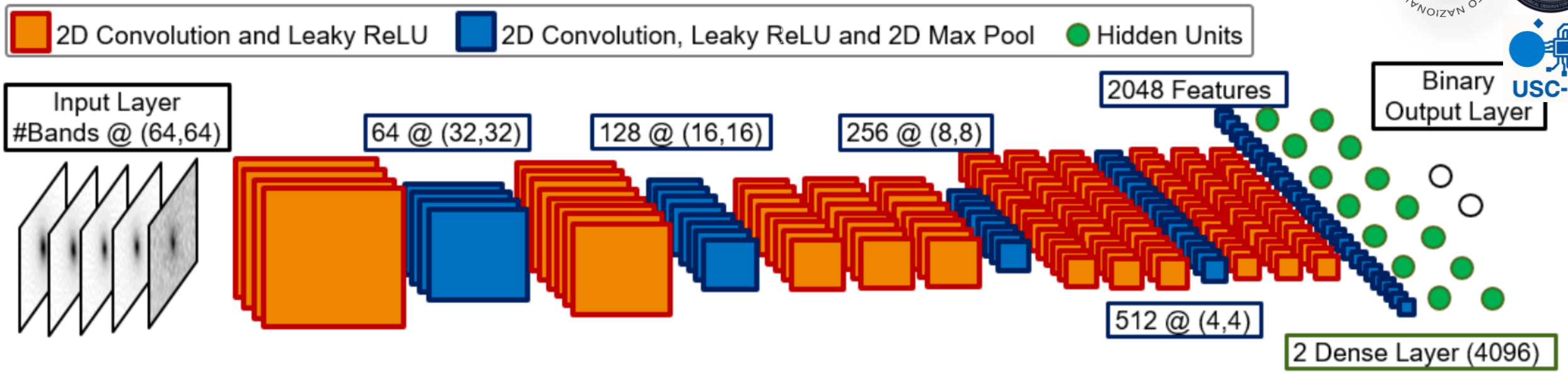
Or, the activation function can be applied after each convolution (e.g. a ReLU):



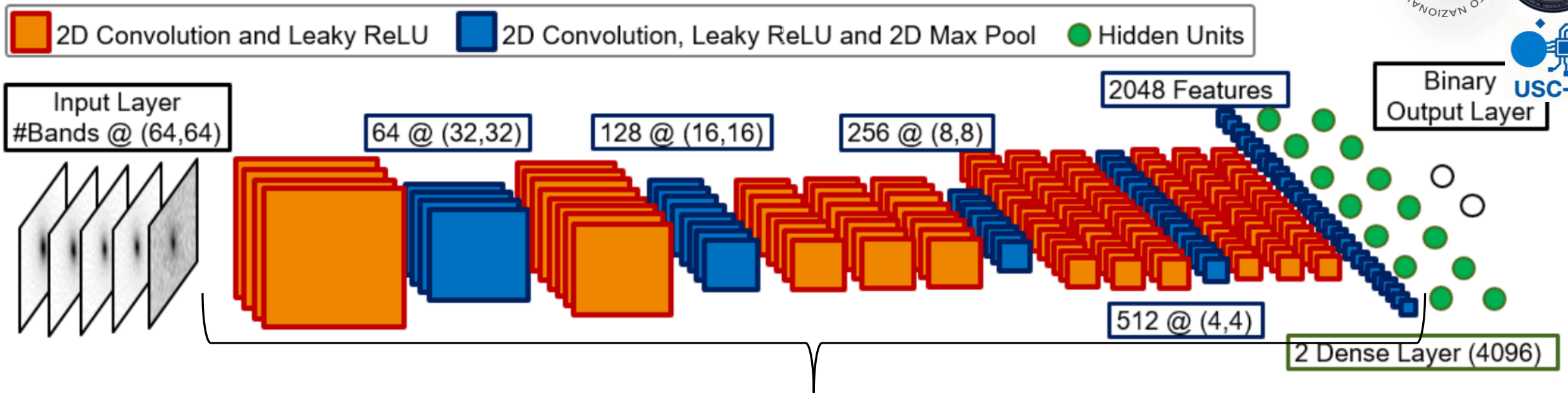
The extracted set of (sub)images is called **Feature Map** (FM):
FMs (at the end of the training) represent the set of features the network automatically extracted to solve the problem



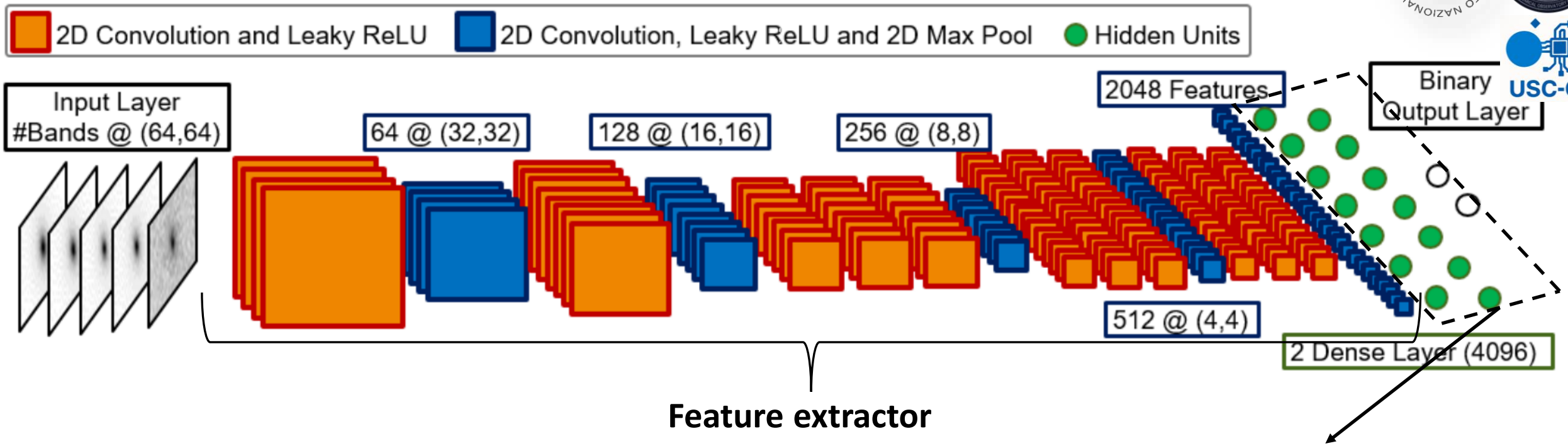
Building a Convolutional Neural Network (CNN):



Building a Convolutional Neural Network (CNN):



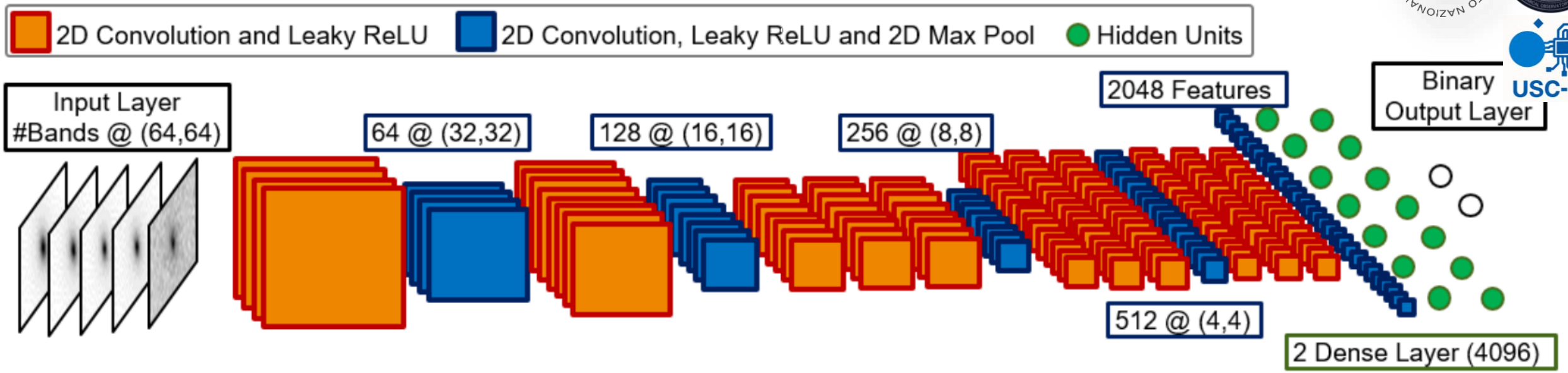
Building a Convolutional Neural Network (CNN):

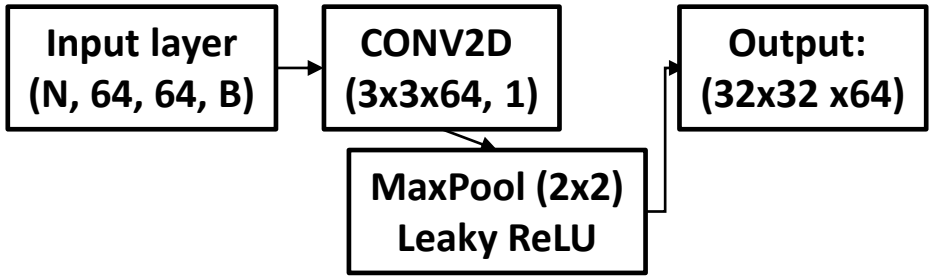


Feature extractor

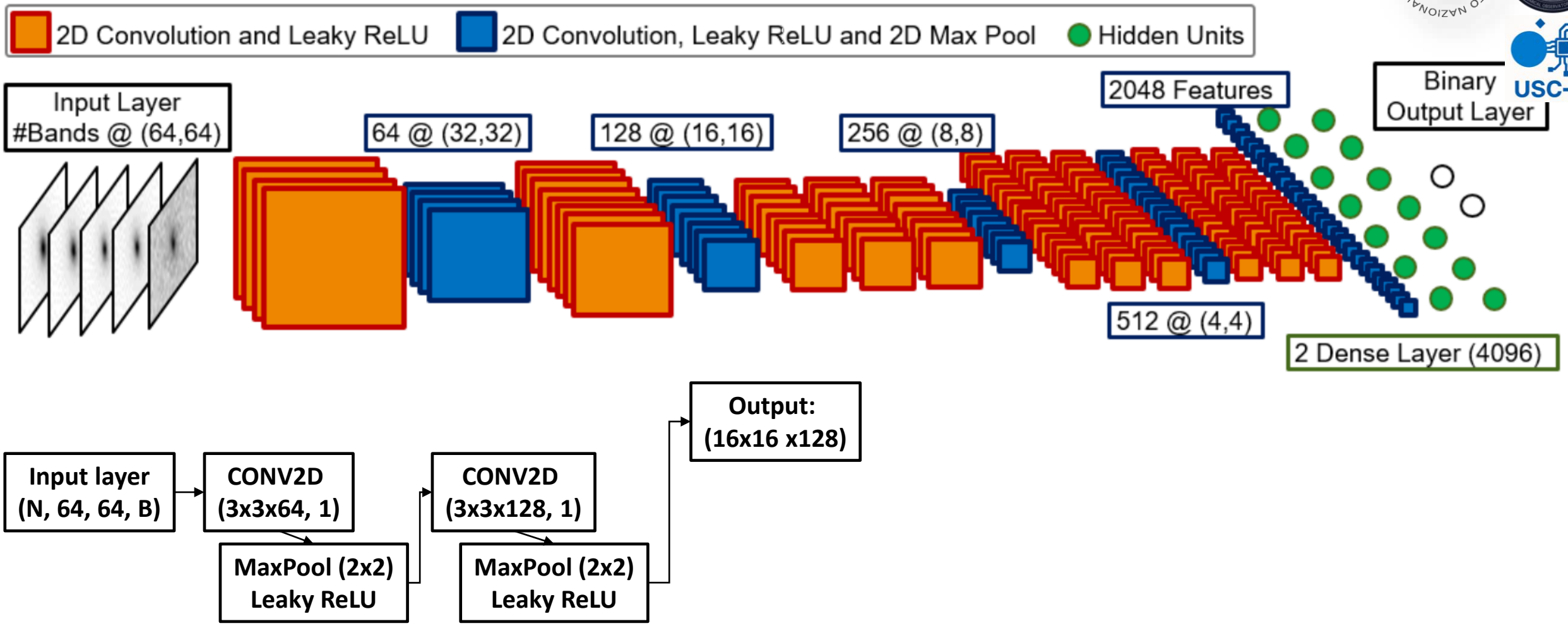
Fully Connected (Dense) Layers For the Classification / Regression task

Building a Convolutional Neural Network (CNN):

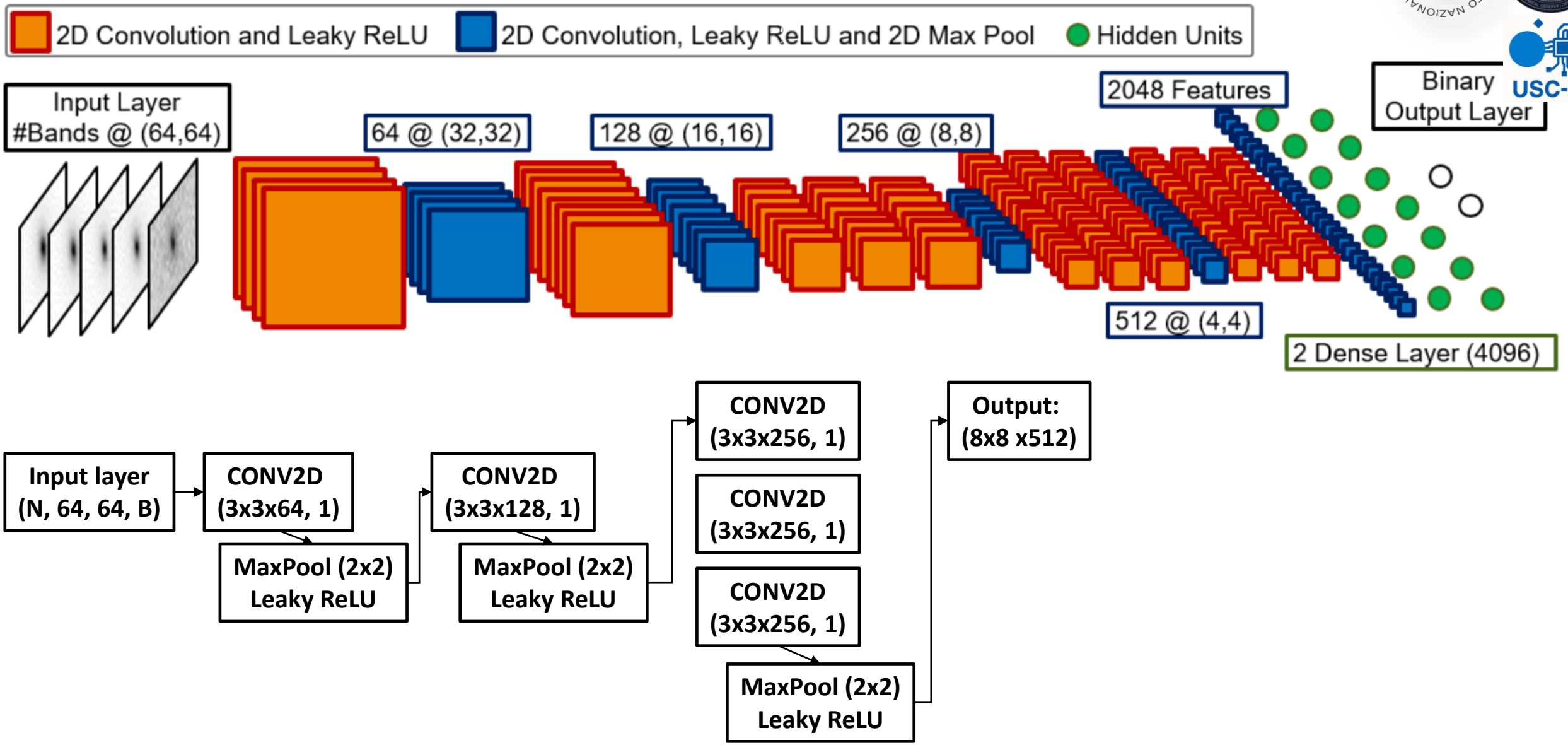




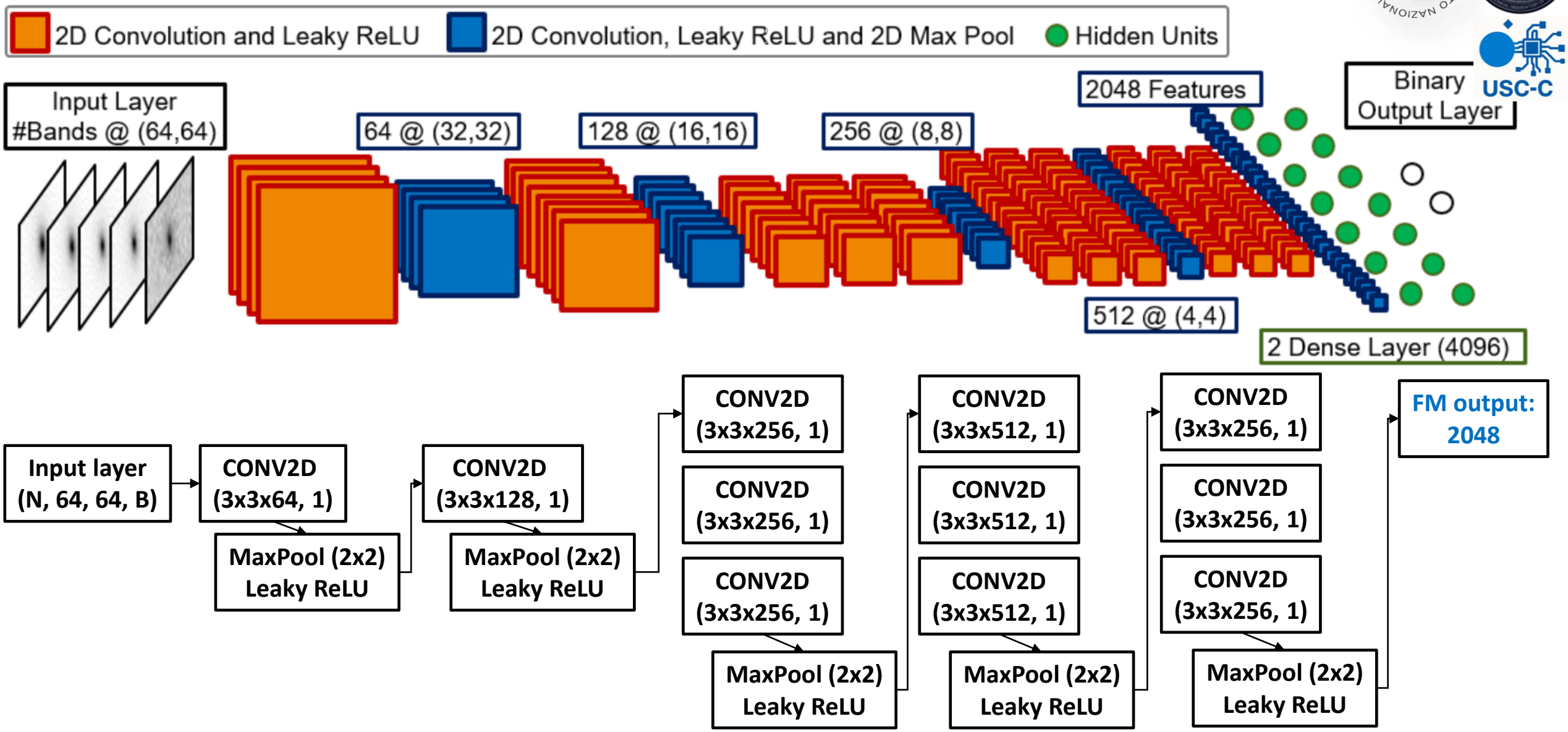
Building a Convolutional Neural Network (CNN):



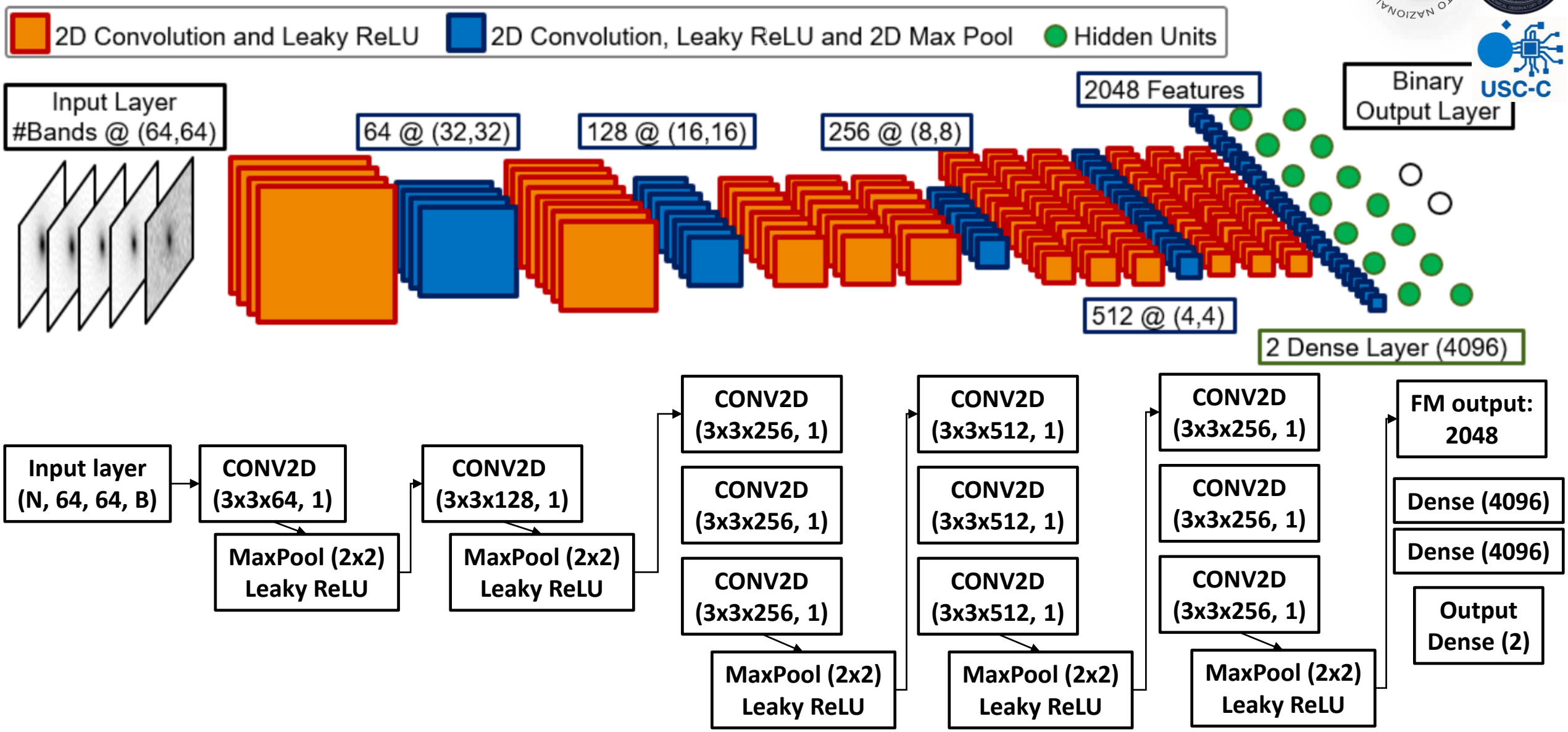
Building a Convolutional Neural Network (CNN):



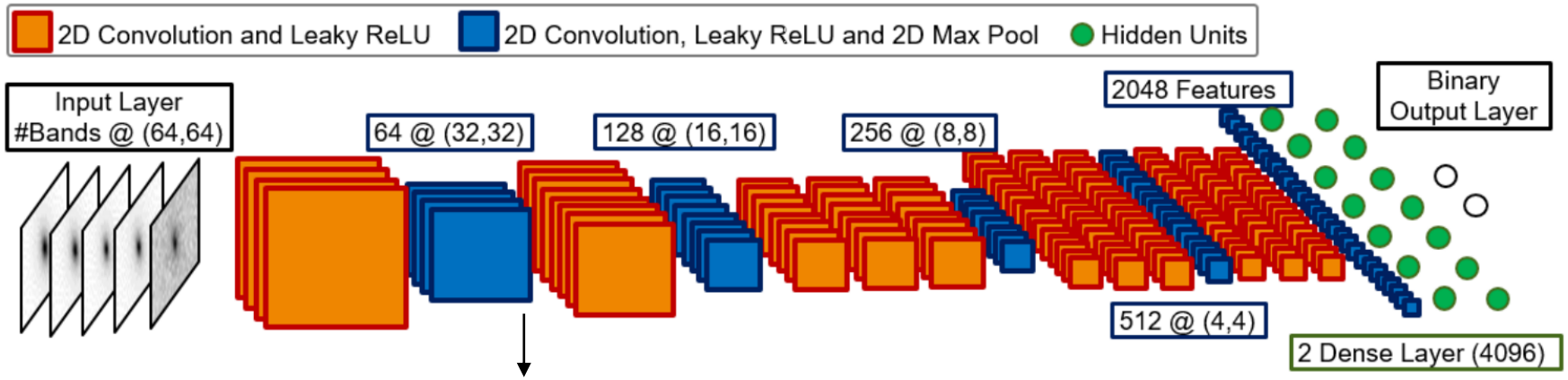
Building a Convolutional Neural Network (CNN):



Building a Convolutional Neural Network (CNN):



Deep Learning: Feature Maps

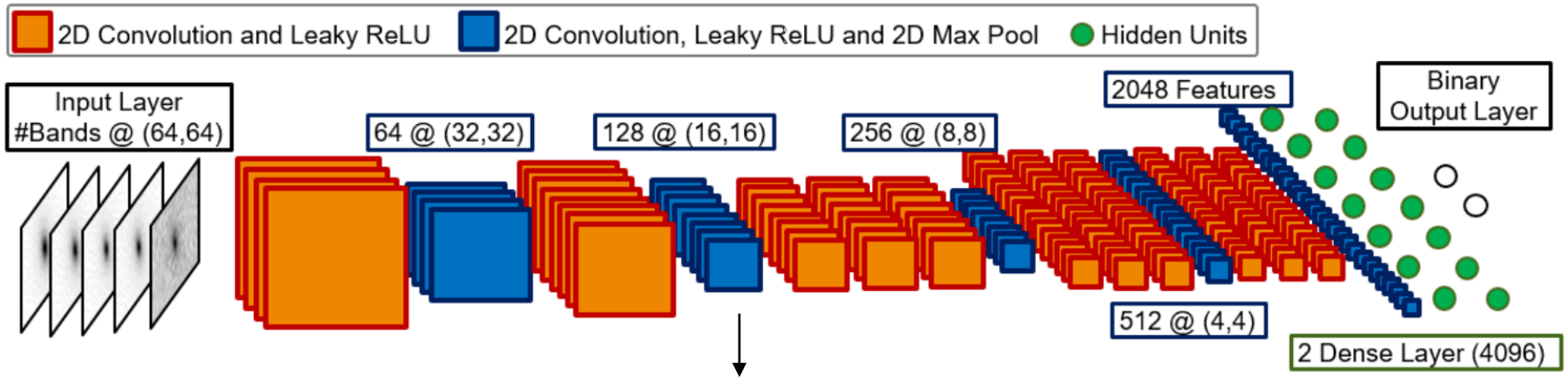


Feature extracted from after the first layer:

Originale

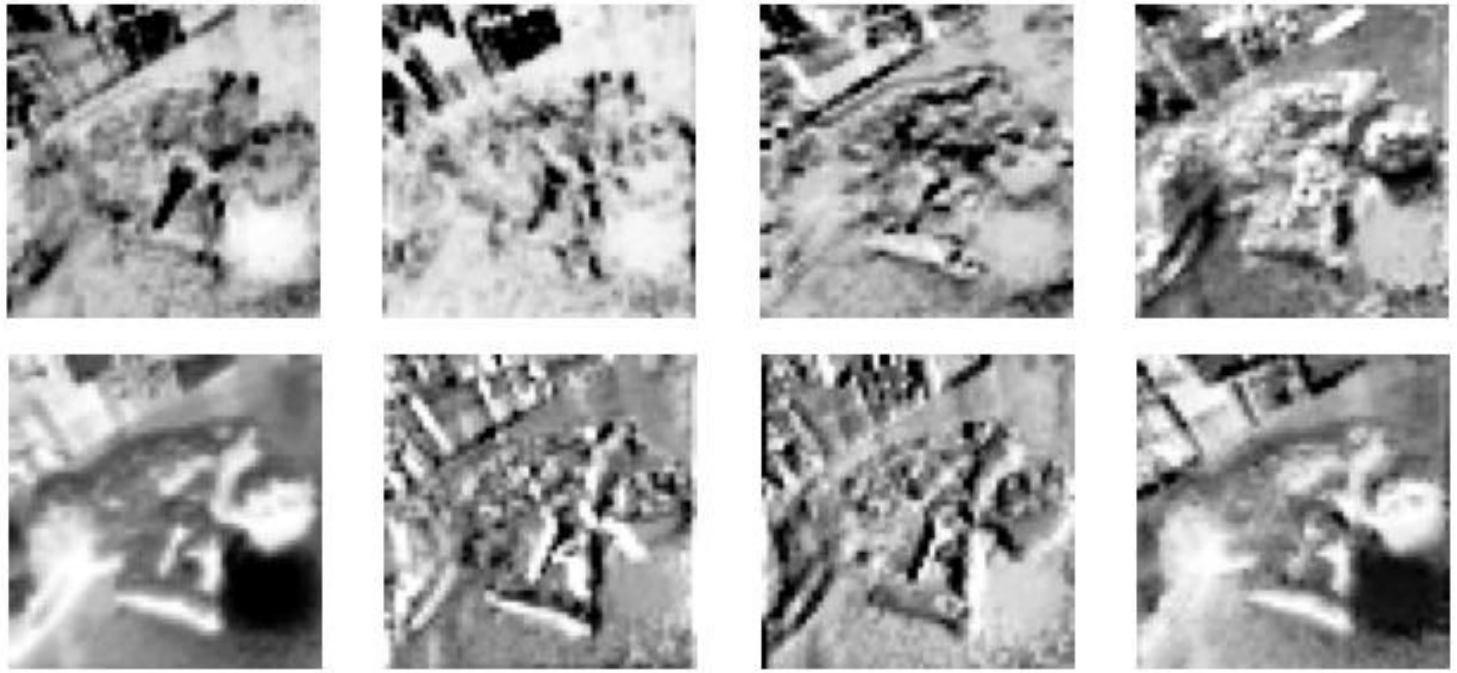
conv1_conv (size=112)

Deep Learning: Feature maps

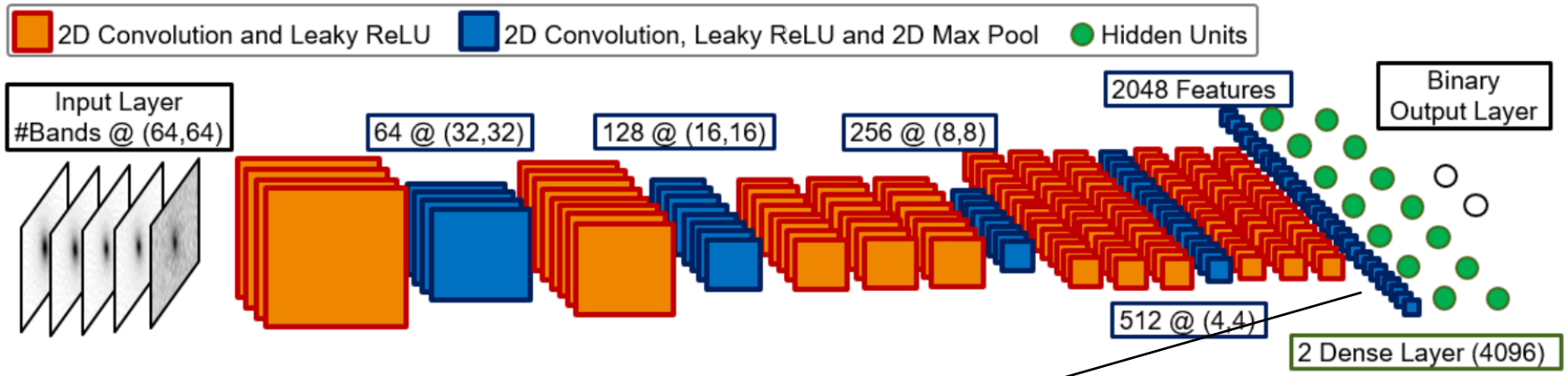


Feature extracted from after the second layer:
conv2_block1_1_conv (size=56)

Originale



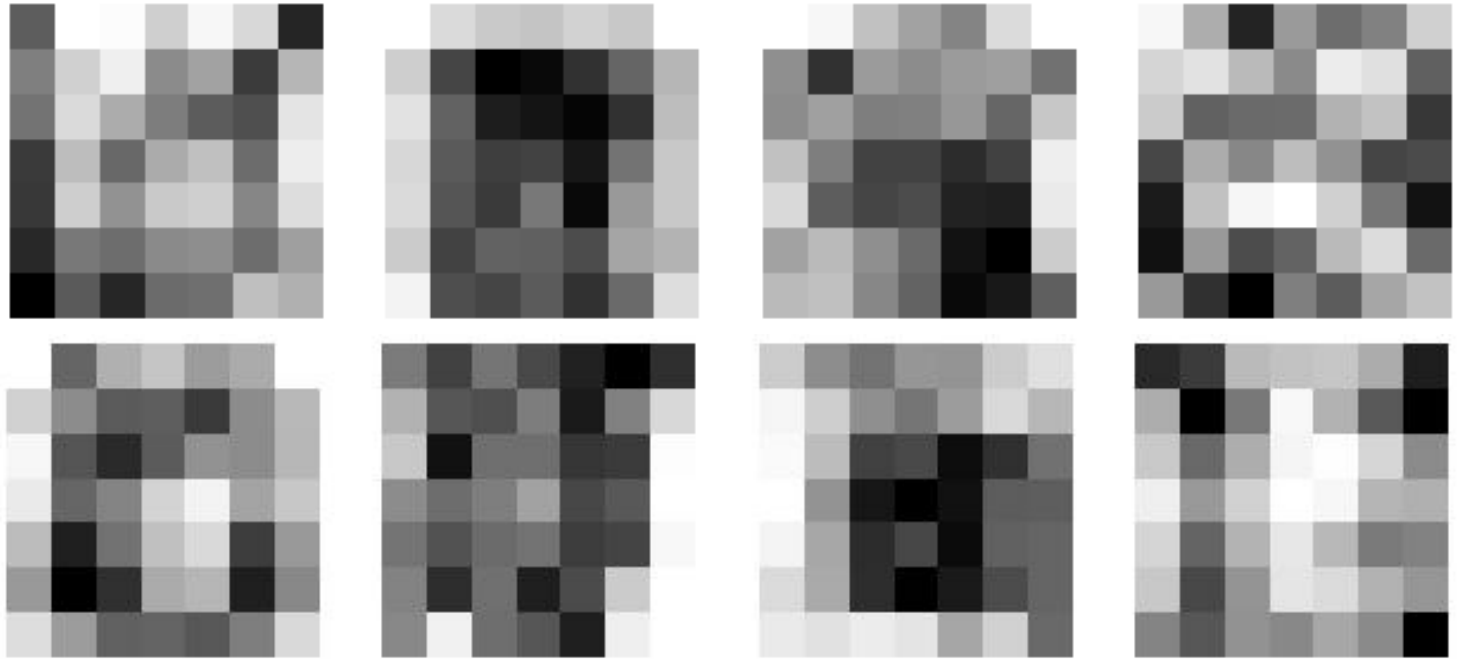
Deep Learning: Feature maps



Feature extracted from after the last layer:

Originale

conv5_block1_1_conv (size=7)



Deep Learning: Feature maps

1st layer

2nd

3rd

4th

Last layer

conv1_conv (size=112)

conv2_block1_1_conv (size=56)

conv3_block2_1_conv (size=28)

conv4_block1_1_conv (size=14)

conv5_block1_1_conv (size=7)

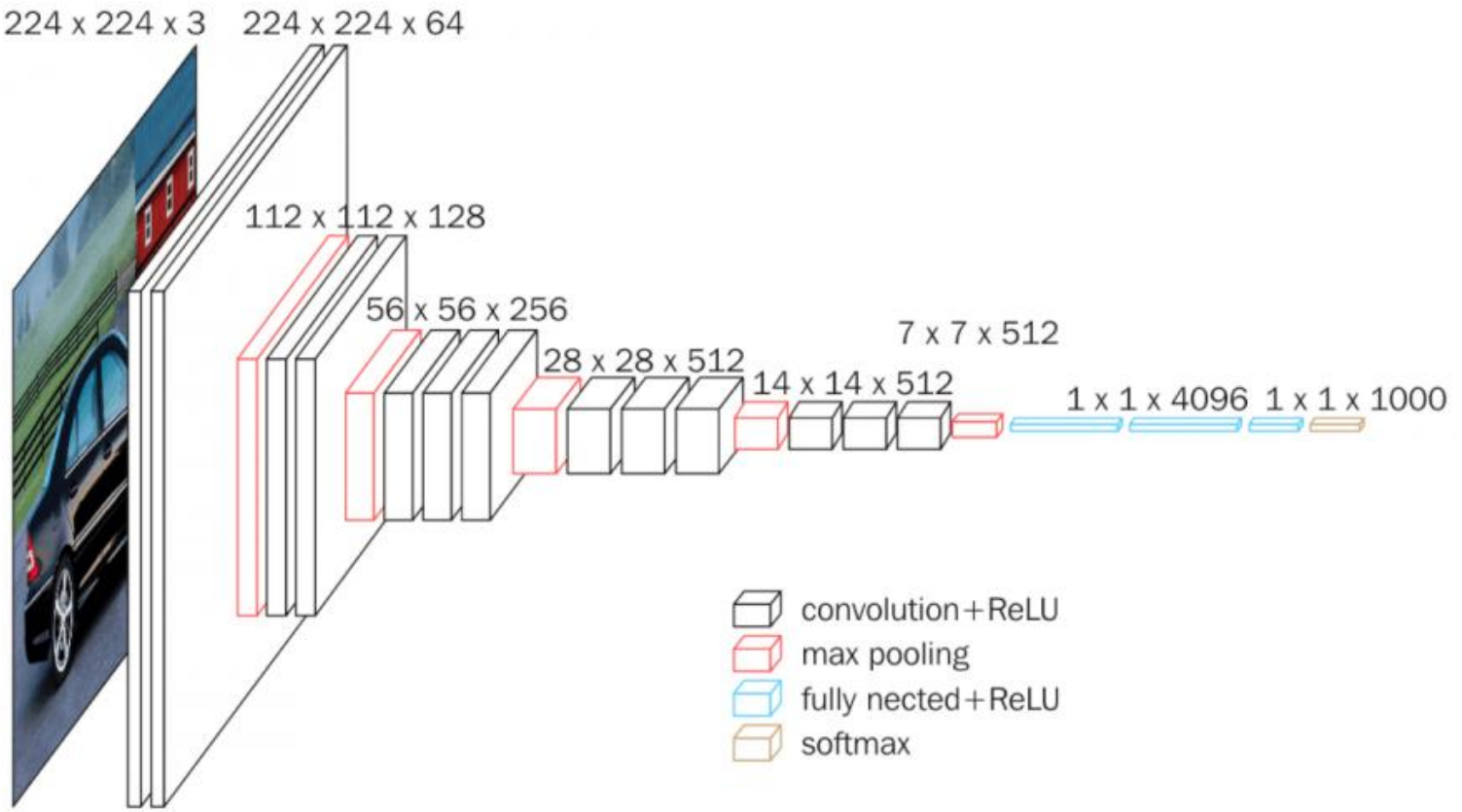
Network depth

Real image
(observed reality)

Extracted images
(Coded in features)

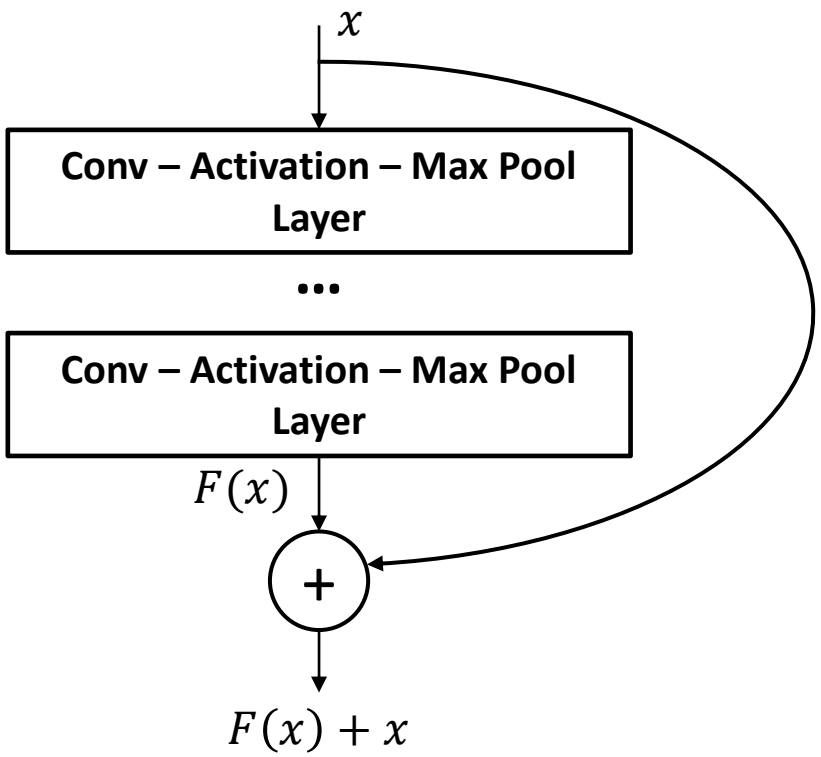
Deep Learning model(s): VGG

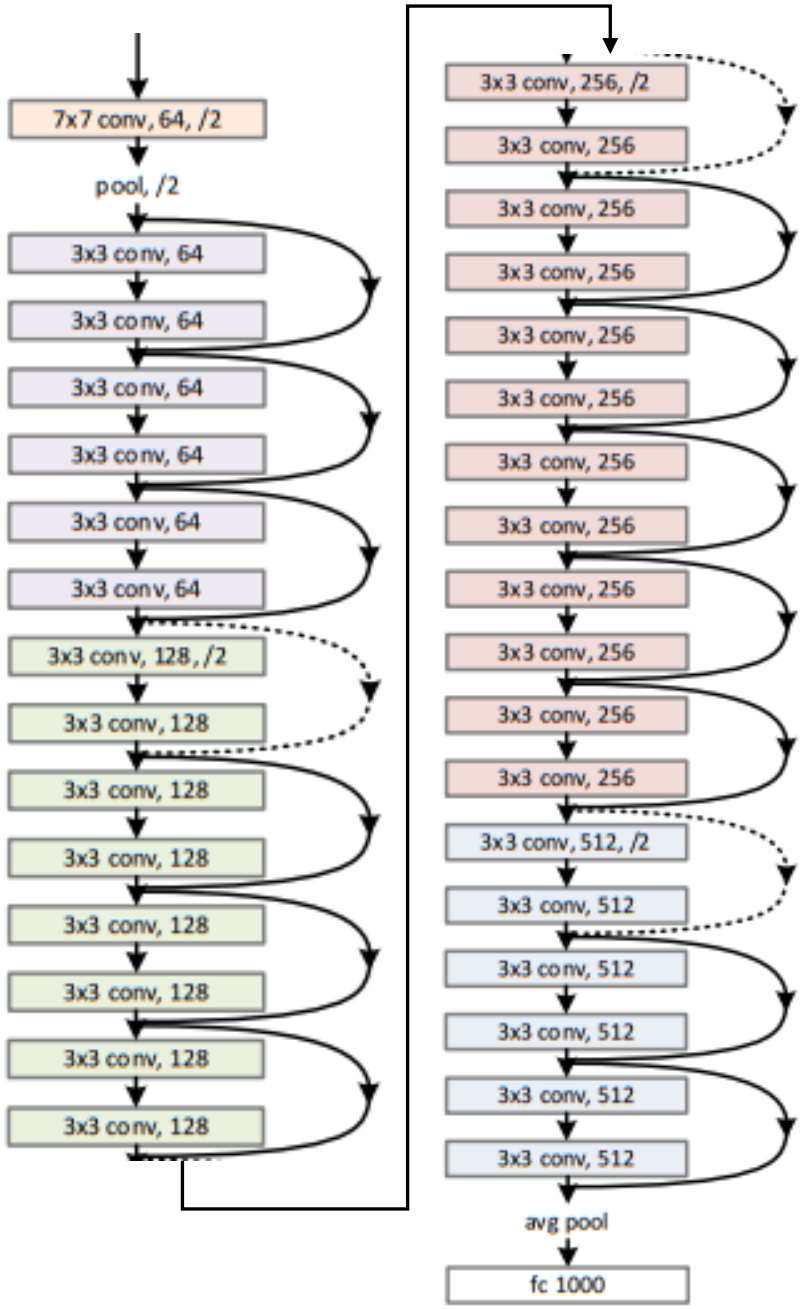
VGG: Visual Group Geometry network (*Simonyan & Zisserman 2014*):
- Only 3x3 kernel with stride 1
- ReLU activations after each convolution
- Three Dense Layers at the top
- Currently two VGG networks are available: VGG-16 and VGG-19



Deep Learning model(s): ResNet

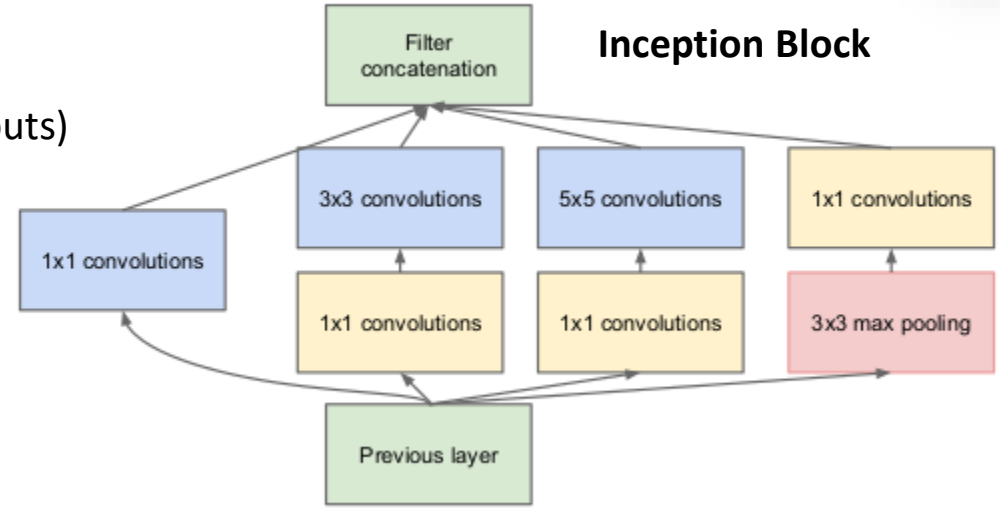
ResNet: Residual Network (*He+2014*):
- It is based on the residual block
- A residual block, the network should learn, at least, the identity
- The identity learning exploits the so-called skip-connection
- It is one of the most performative CNN
- Currently there are tens of available ResNet

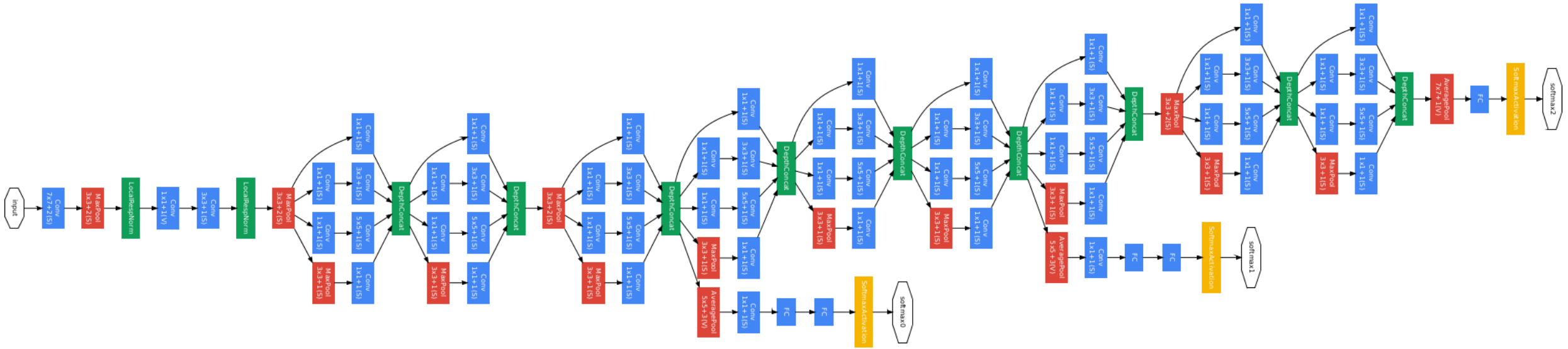




Deep Learning model(s): Inception Network

Inception Network (*Szegedy+2015*):
- It is based on the inception block
- This block performs a sort of hyper-parameters optimization
- Multi-output network: FC layers are set at different depths (auxiliar outputs)
- Weighted loss function between the 'real' output and the auxiliar ones

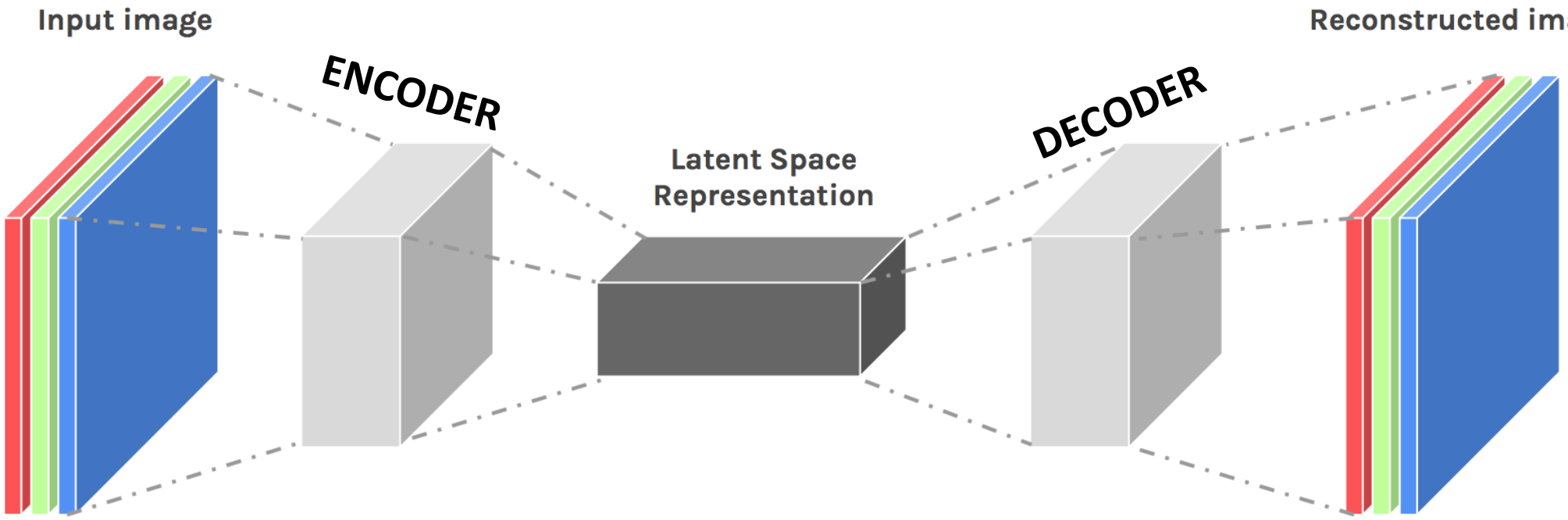




GO TO NOTEBOOK FOR THE CNN CODING INTRODUCTION

Autoencoder (in a nutshell)

Deep Autoencoder (*Hinton+2006*):
- Autoencoders are self-learning algorithms (i.e. they learn to map a function: $\mathbb{R}^n \rightarrow \mathbb{R}^n$)
- By using the same space as input and output, autoencoders find a embedded space that represents the input
- They can be used as denoiser, image-generation, data-imputation, image-segmentation, upscaling
- They also act as anomaly detectors

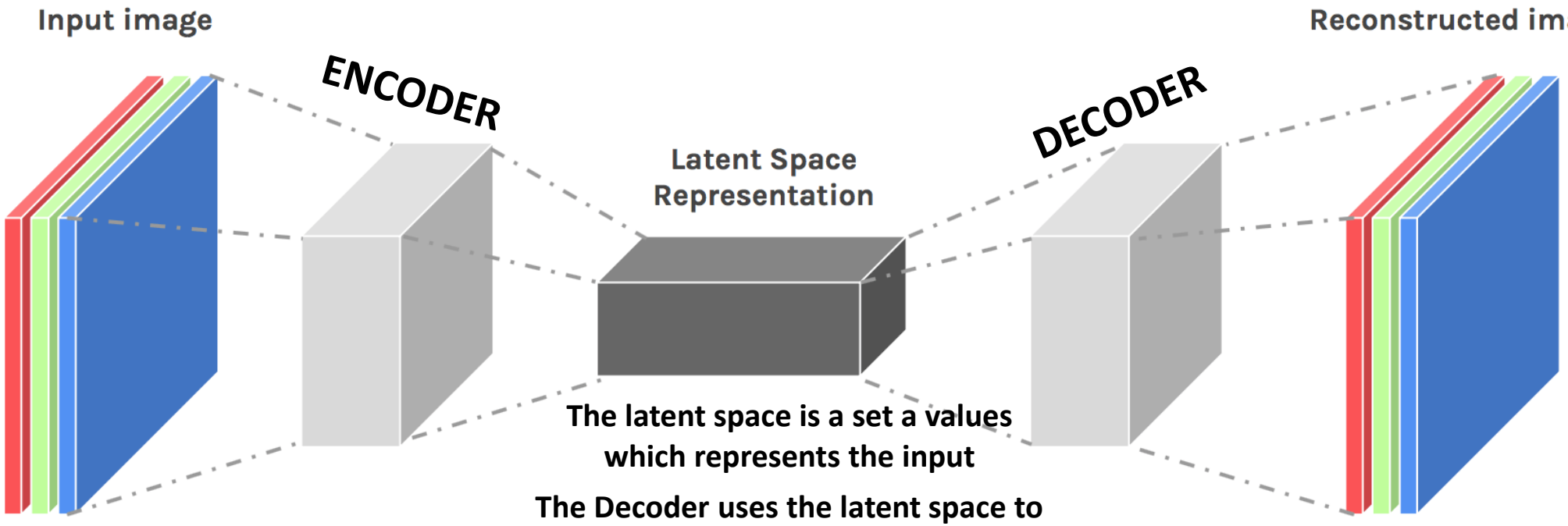


Chain of Conv, Activation, Pooling
It maps the **input** into a **Feature Map vector**

Chain of Conv, Activation, Up-sampling
It maps the **Feature Map** vector into the **input**

Autoencoder (in a nutshell)

Deep Autoencoder (*Hinton+2006*):
- Autoencoders are self-learning algorithms (i.e. they learn to map a function: $\mathbb{R}^n \rightarrow \mathbb{R}^n$)
- By using the same space as input and output, autoencoders find a embedded space that represents the input
- They can be used as denoiser, image-generation, data-imputation, image-segmentation, upscaling
- They also act as anomaly detectors



Chain of Conv, Activation, Pooling
It maps the **input** into a **Feature Map vector**

The latent space is a set a values which represents the input

The Decoder uses the latent space to reconstruct the input

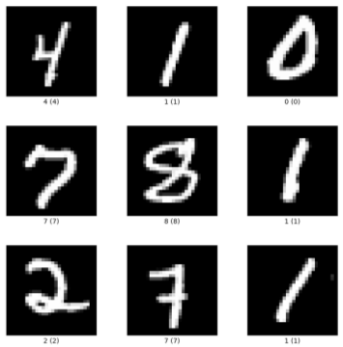
Chain of Conv, Activation, Up-sampling
It maps the **Feature Map** vector into the **input**

GO TO NOTEBOOK FOR THE AUTOENCORE CODING INTRODUCTION


INAF ISTITUTO NAZIONALE DI ASTROFISICA
INAF SCHOOL OF AI
USC-C

FIGURES USED IN THE NOTEBOOKS

LeNet Like network



(N, 28, 28, 1)

CONV2D (3x3x16, 1)

MaxPool(2x2)

ReLU

(N, 14, 14, 16)

CONV2D (3x3x32, 1)

MaxPool(2x2)

ReLU

(N, 7, 7, 32)

CONV2D (3x3x64, 1)

MaxPool(2x2)

ReLU

(N, 3, 3, 64)

Extractor (Flatten)

(N, 3*3*64)= =(N, 576)

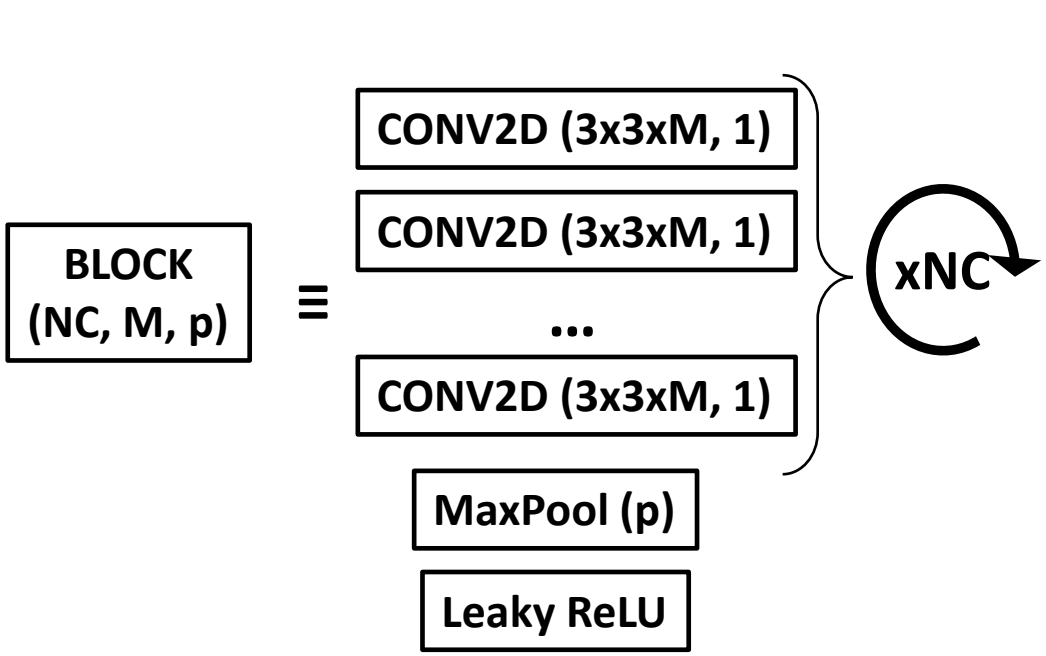
Dense (1024)

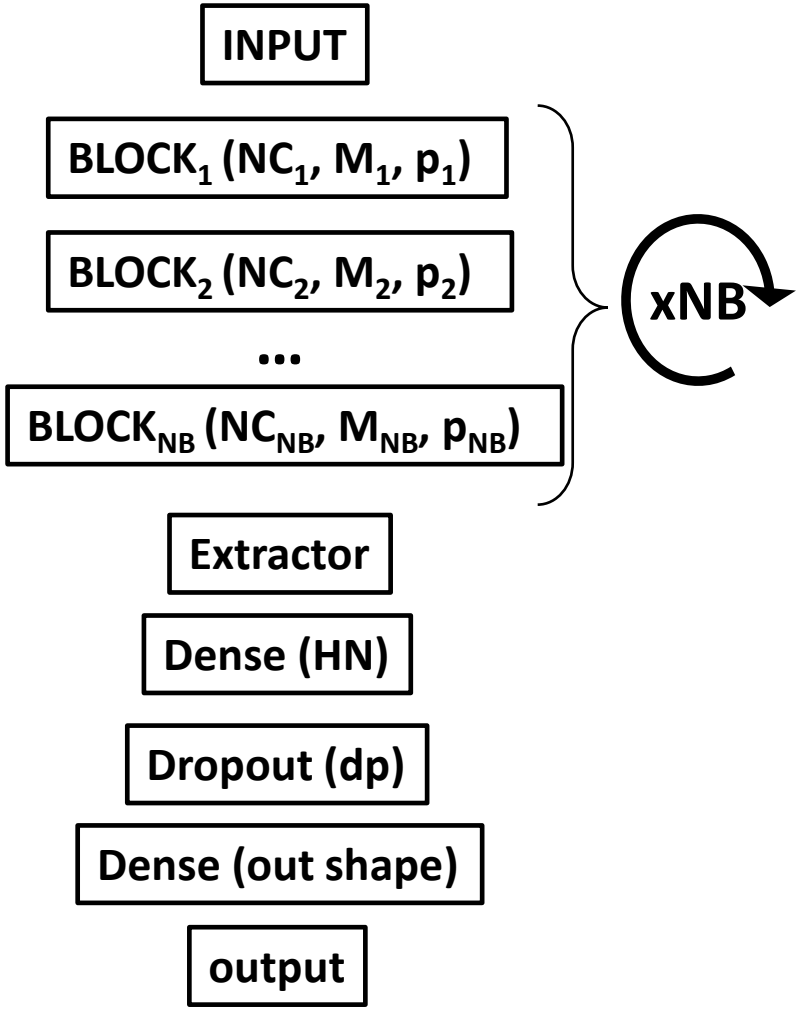
Dropout (0.5)

Dense (10)

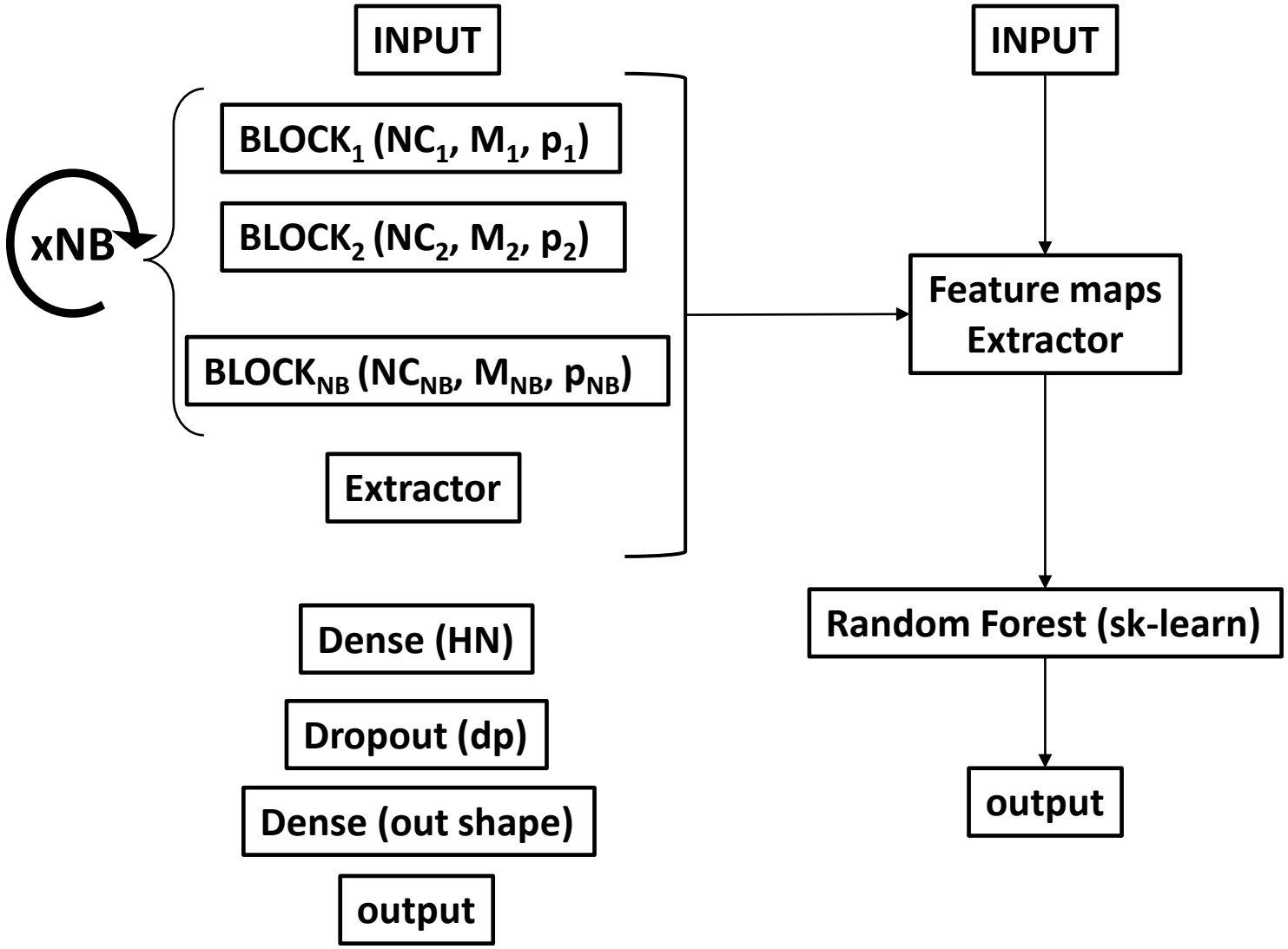
Output

VGG network

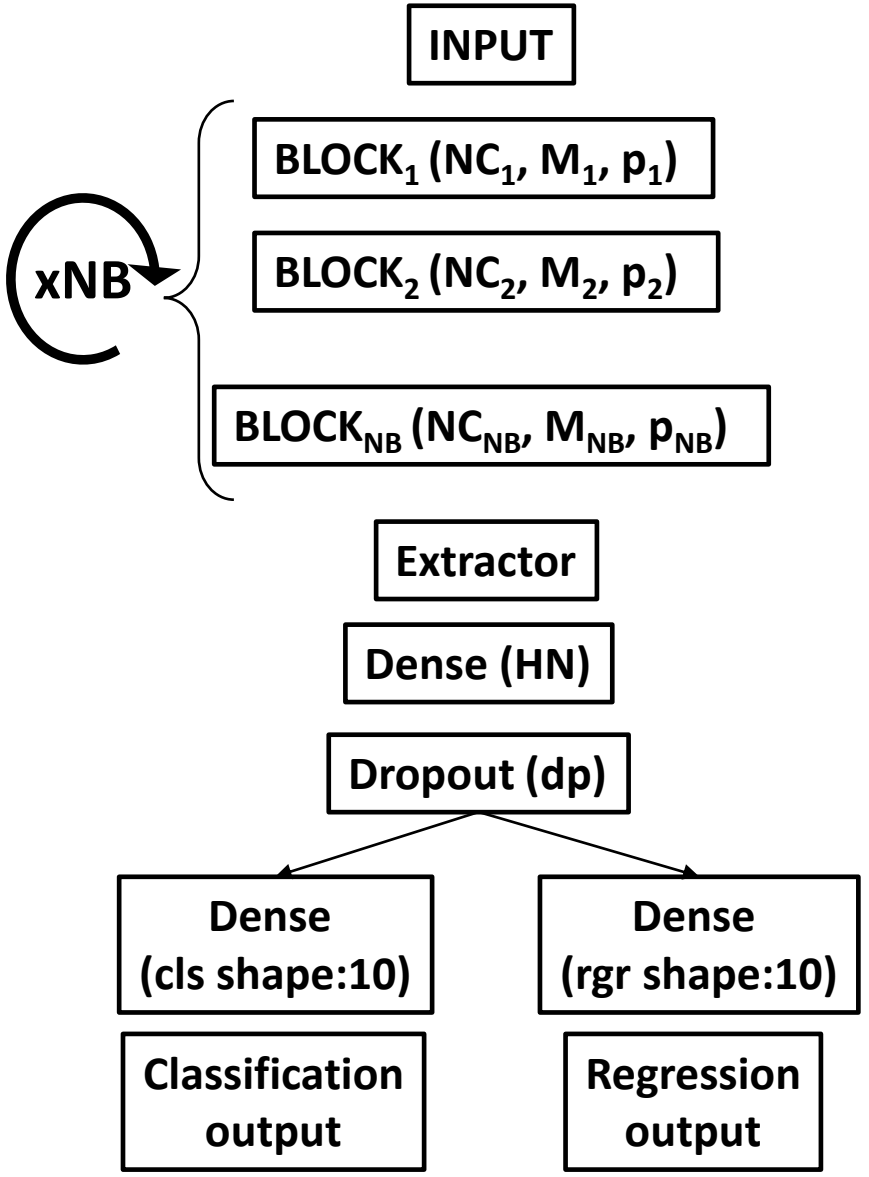

BLOCK
(NC, M, p)
≡
CONV2D (3x3xM, 1)
CONV2D (3x3xM, 1)
...
CONV2D (3x3xM, 1)
xNC
MaxPool (p)
Leaky ReLU


INPUT
BLOCK$_1$ (NC$_1$, M$_1$, p$_1$)
BLOCK$_2$ (NC$_2$, M$_2$, p$_2$)
...
BLOCK$_{NB}$ (NC$_{NB}$, M$_{NB}$, p$_{NB}$)
xNB
Extractor
Dense (HN)
Dropout (dp)
Dense (out shape)
output

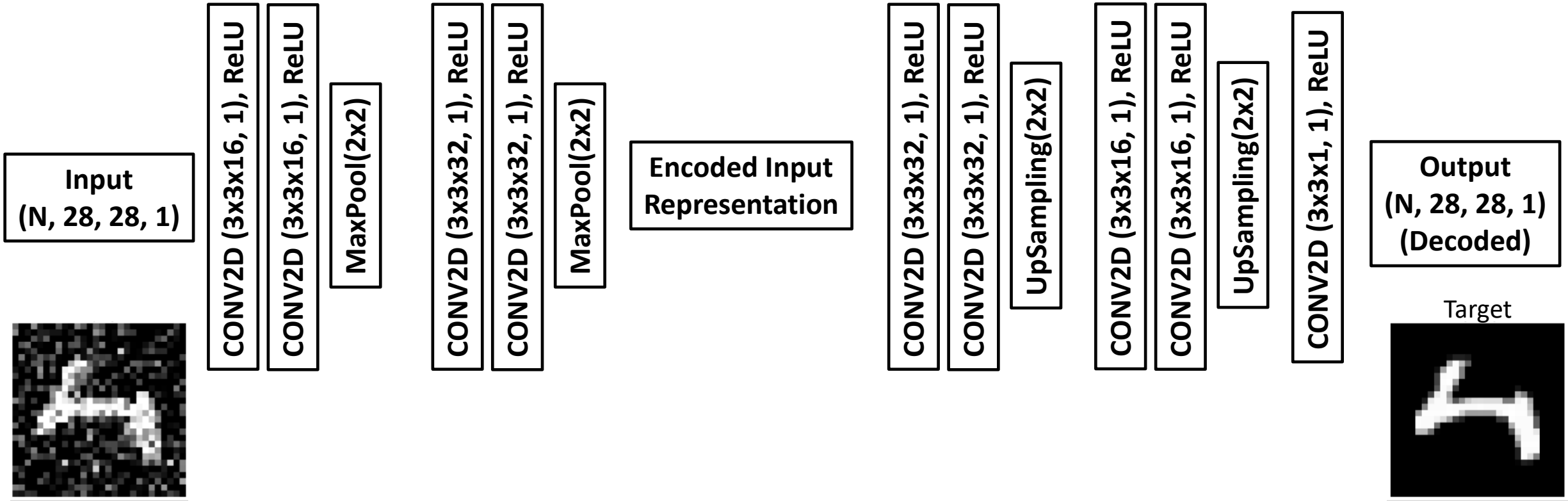
Extractor + RF



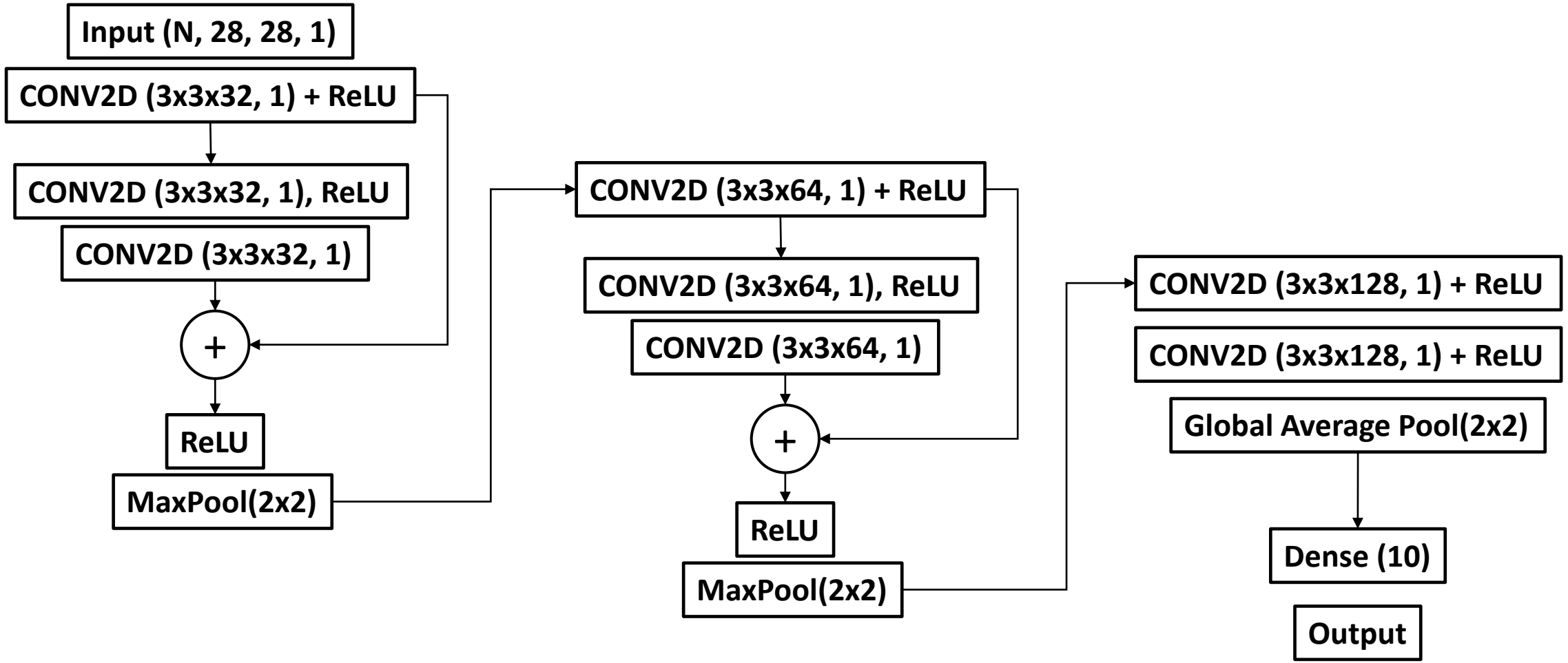
Multi Output CNN



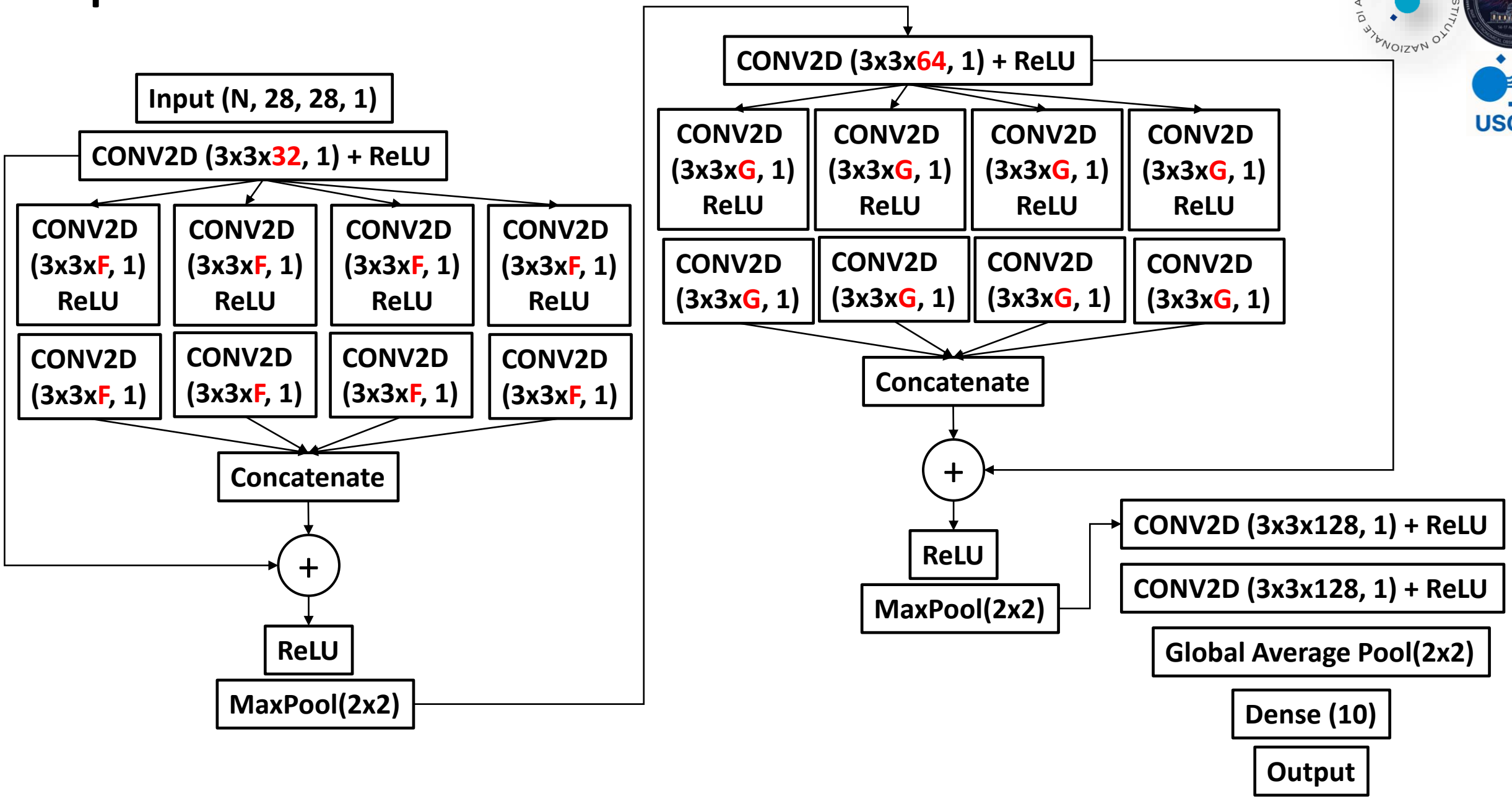
Simplified Autoencoder



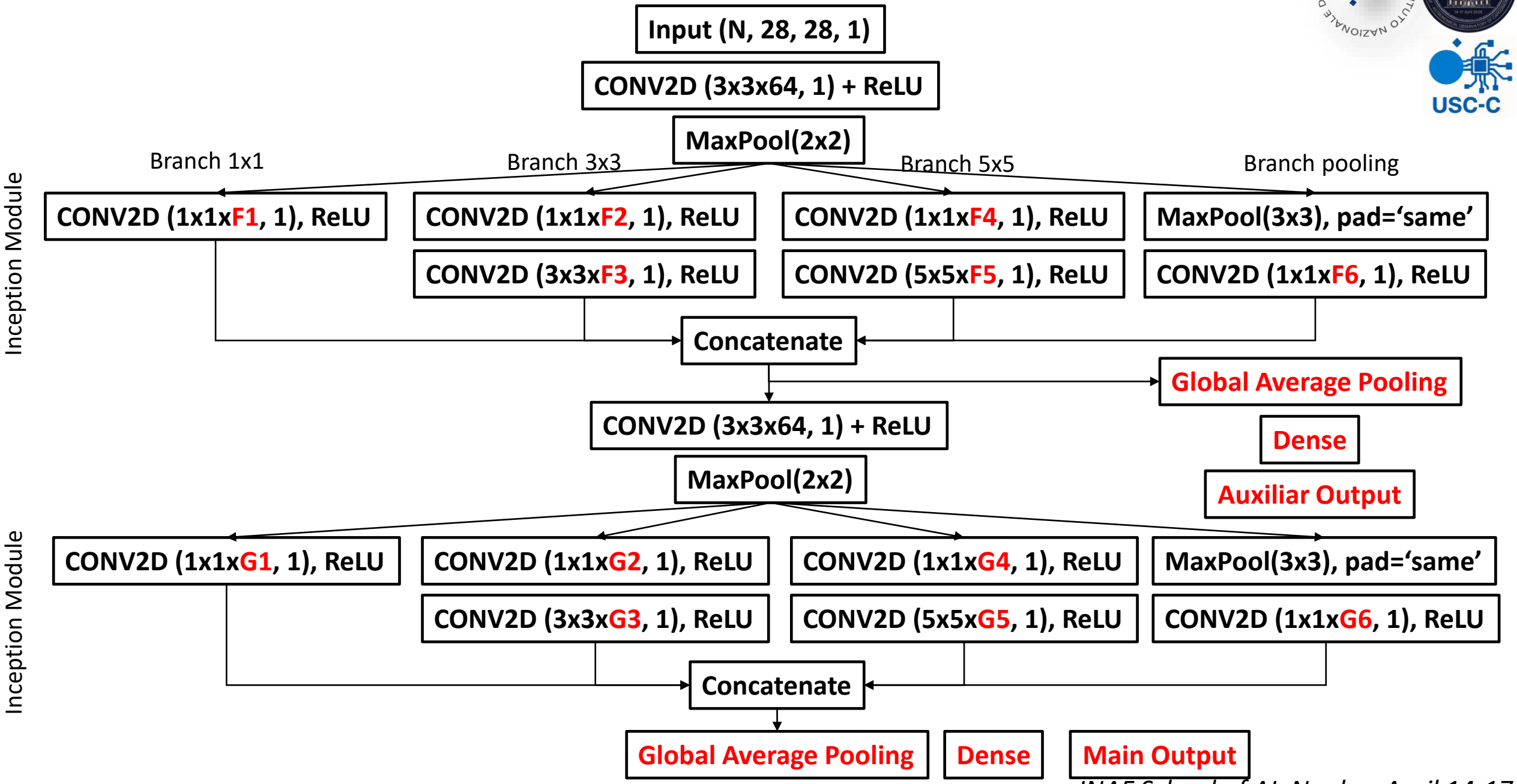
Simplified Resnet like



Simplified ResneXt like



Simplified Inception network (dual-outputs)



Channel-by-Channel (four-outputs)