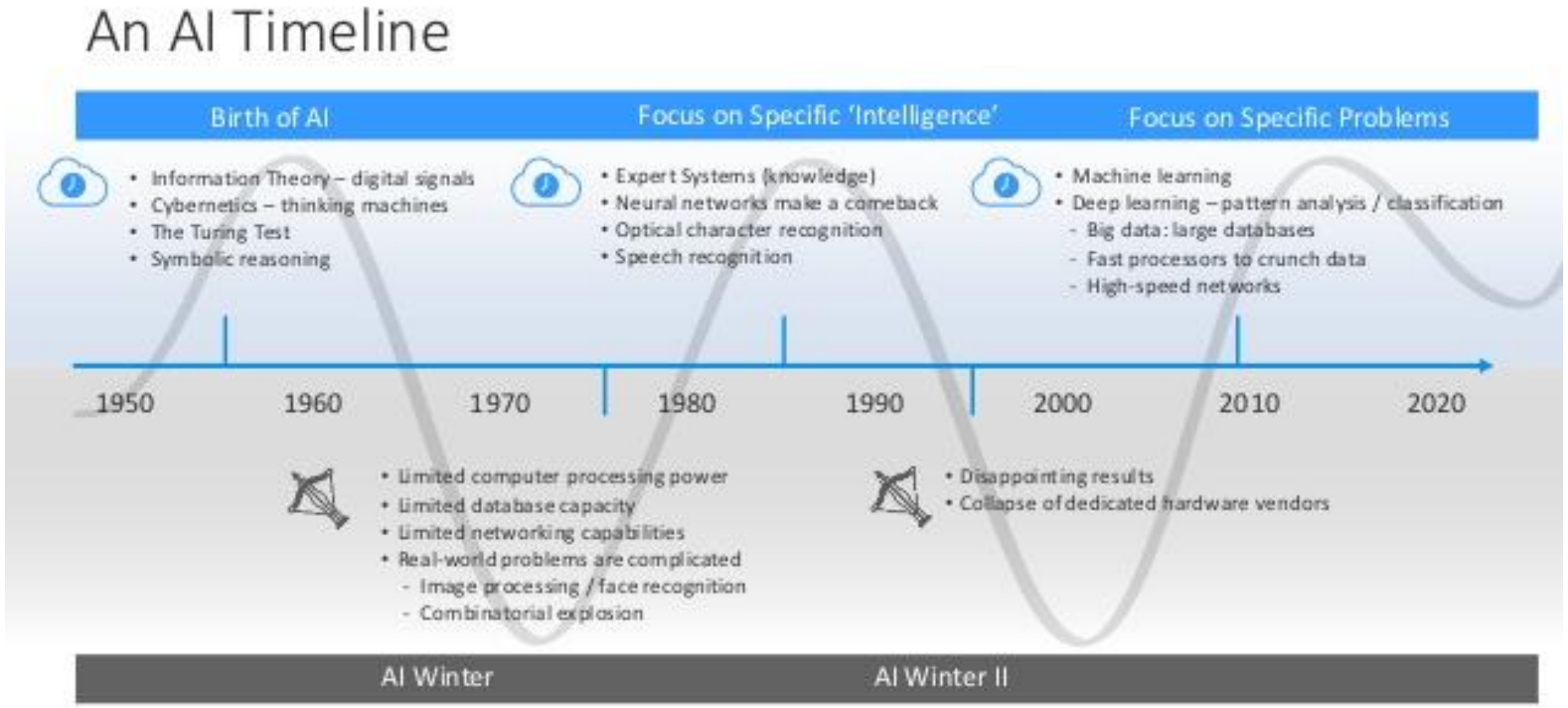
Embedded AI

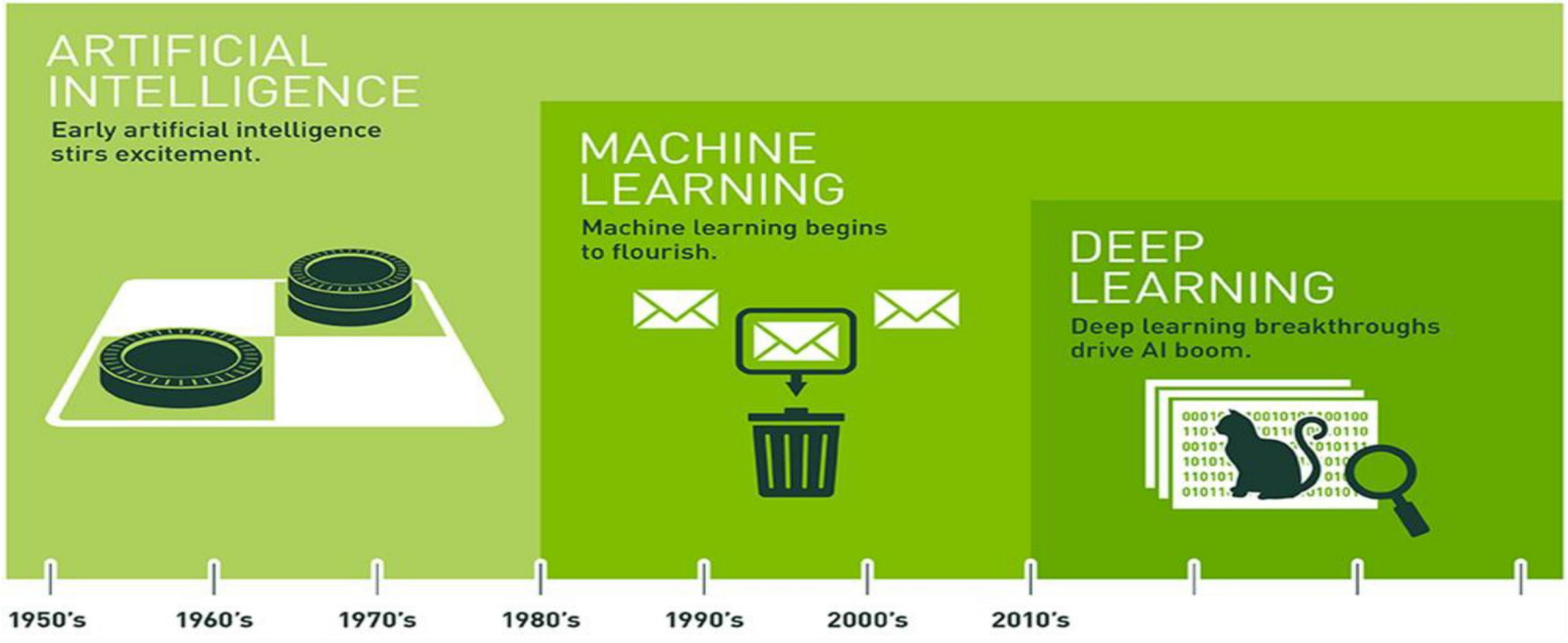
Outline of the course

- Introduction to Convolutional Neural Networks
  - Machine learning
  - Neural networks
  - Convolutional neural networks
  - Learning and inference
- Embedded Artificial Intelligence
  - Embedded targets
  - CNN on MCU with STM CubeAI
- Goals of the course
  - Labs
  - MNIST
  - UCI HAR

Timeline of AI from 1950



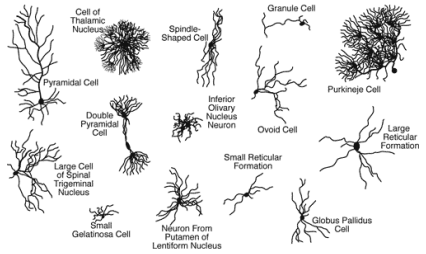
Deep Learning, the last trend of AI

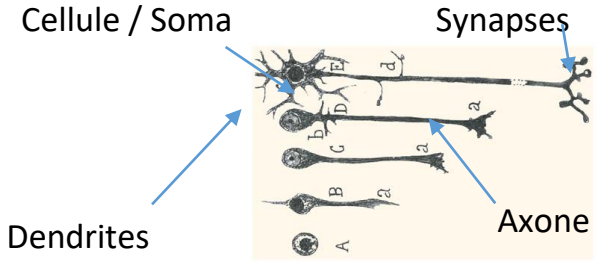


The short story of neural networks

1890, Ramon Y Cajal

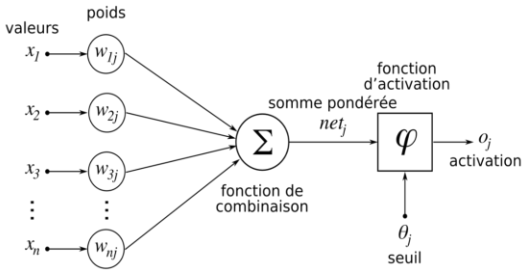
Diversity of morphology and behaviour





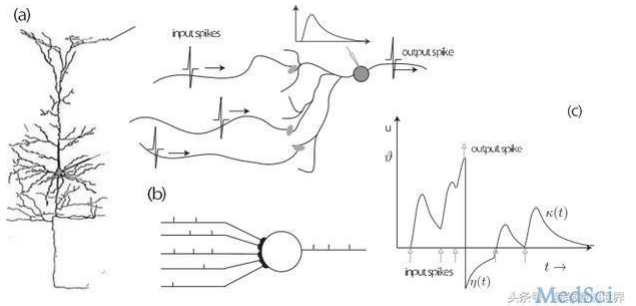
1950, Warren McCulloch et Walter Pitts

Binary inputs/outputs



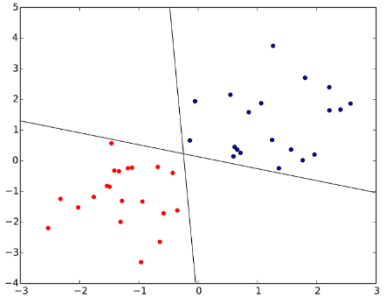
1952, Hodgkin Huxley

Spike dynamics



1957, Frank Rosebblatt

real input, linearly separable data

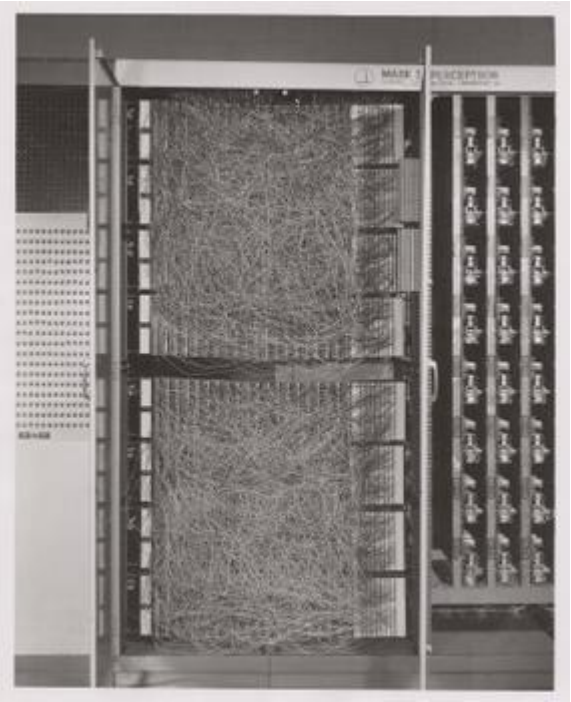


Chemical Communication → Electrical Transmission

Cajal, R. S. Recollections of My Life (translated by E. H. Craigie with the assistance of . Cano) (Am. Phil. Soc., Philadelphia, 1937; reprinted by MIT Press, Cambridge, Massachusetts, 1989)

Three generations of neural networks

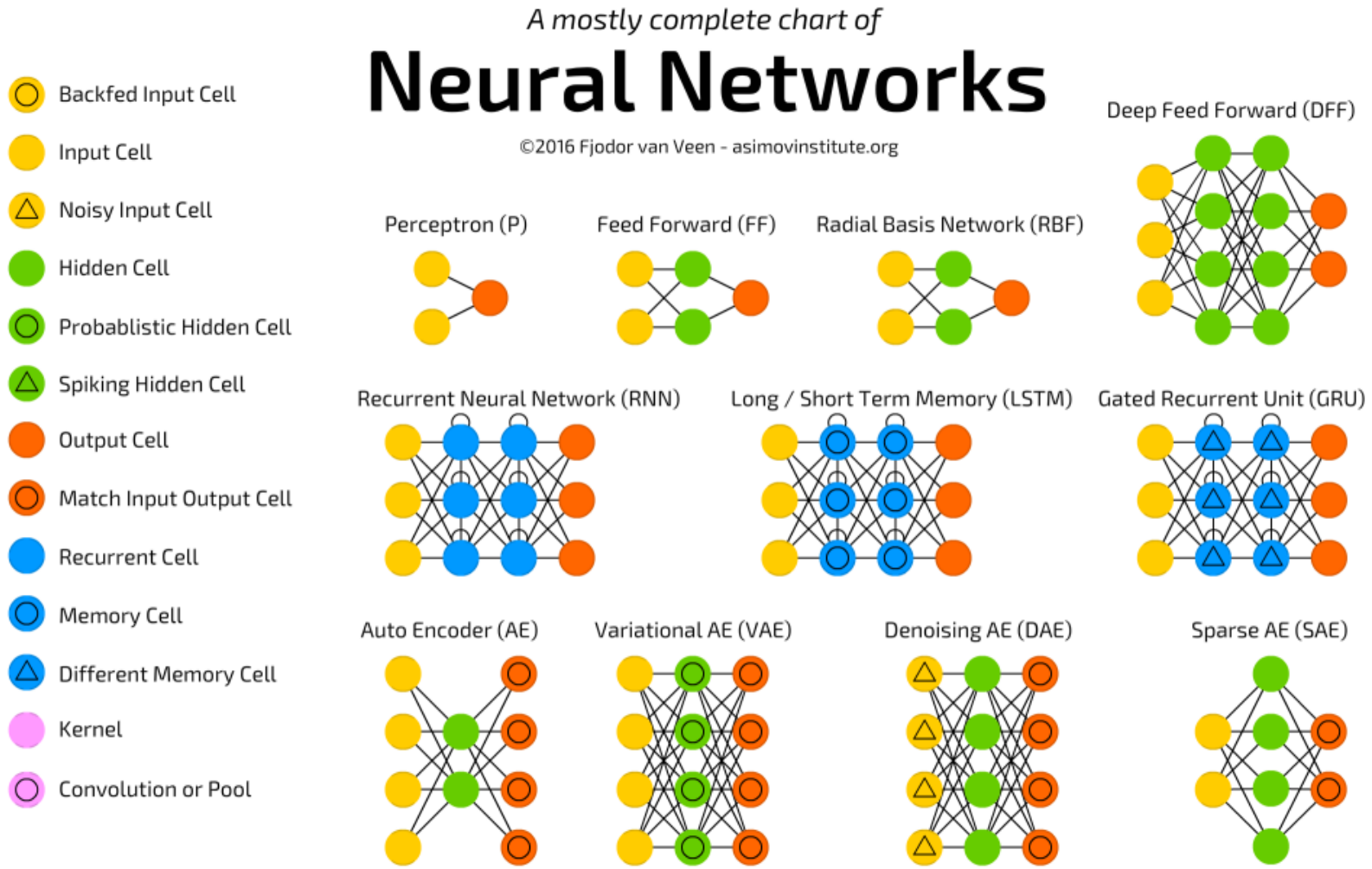
- **First generation of neural networks - Perceptron**
  - Formal neuron (Mc Culloh & Pitts)
    - Binary input / output
    - threshold gates
    - Linearly separable data
  - Perceptron (Rosenblatt)
    - Real input/output
    - Single-layer perceptrons
    - Linearly separable data
    - Learning algorithm
- **Second generation – Back Propagation**
  - Multi-layer perceptron
  - the output layer would give a probability value for a givenoutcome
  - Rumelhart backpropagation (BP) trainin galgorithm (Gradient descent), 1986
  - Requires computational units with derivable activation function
- **Third generation - Spiking neuron**
  - Bio-inspired modeling
  - Discrete representation of inputs/outputs
  - Time representation
  - Internal state



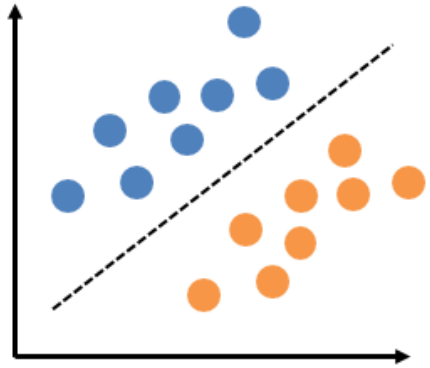
1958, Mark I Perceptron machine, Cornell

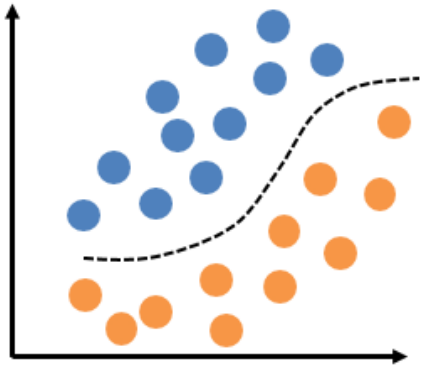
W. Mass, Networks of Spiking Neurons: The Third Generation of Neural Network Models, 1996

ANN, one method of Machine Learning

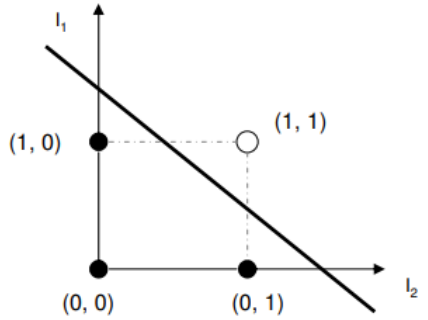


Linearly separable data

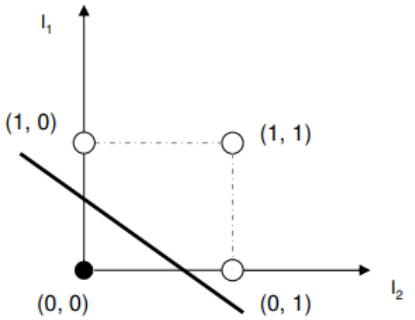




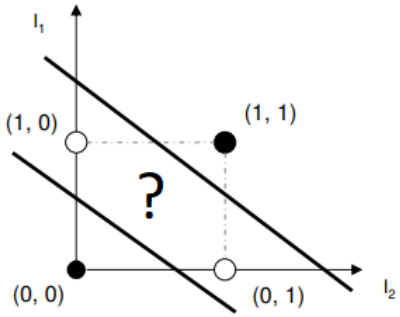
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

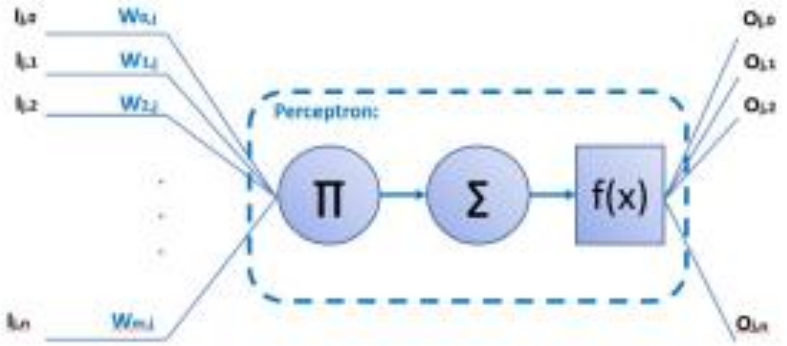


Perceptron algorithm

- Each change of **w** decreases the error on a specific point. However, changes for several points are correlated, that is different points could change the weights in opposite directions. Thus, this iterative algorithm requires several loops to converge.
- Guarantee to find a separating hyperplane if one exists–if data is linearly separable
- If data are not linearly separable, then this algorithm loops indefinitely

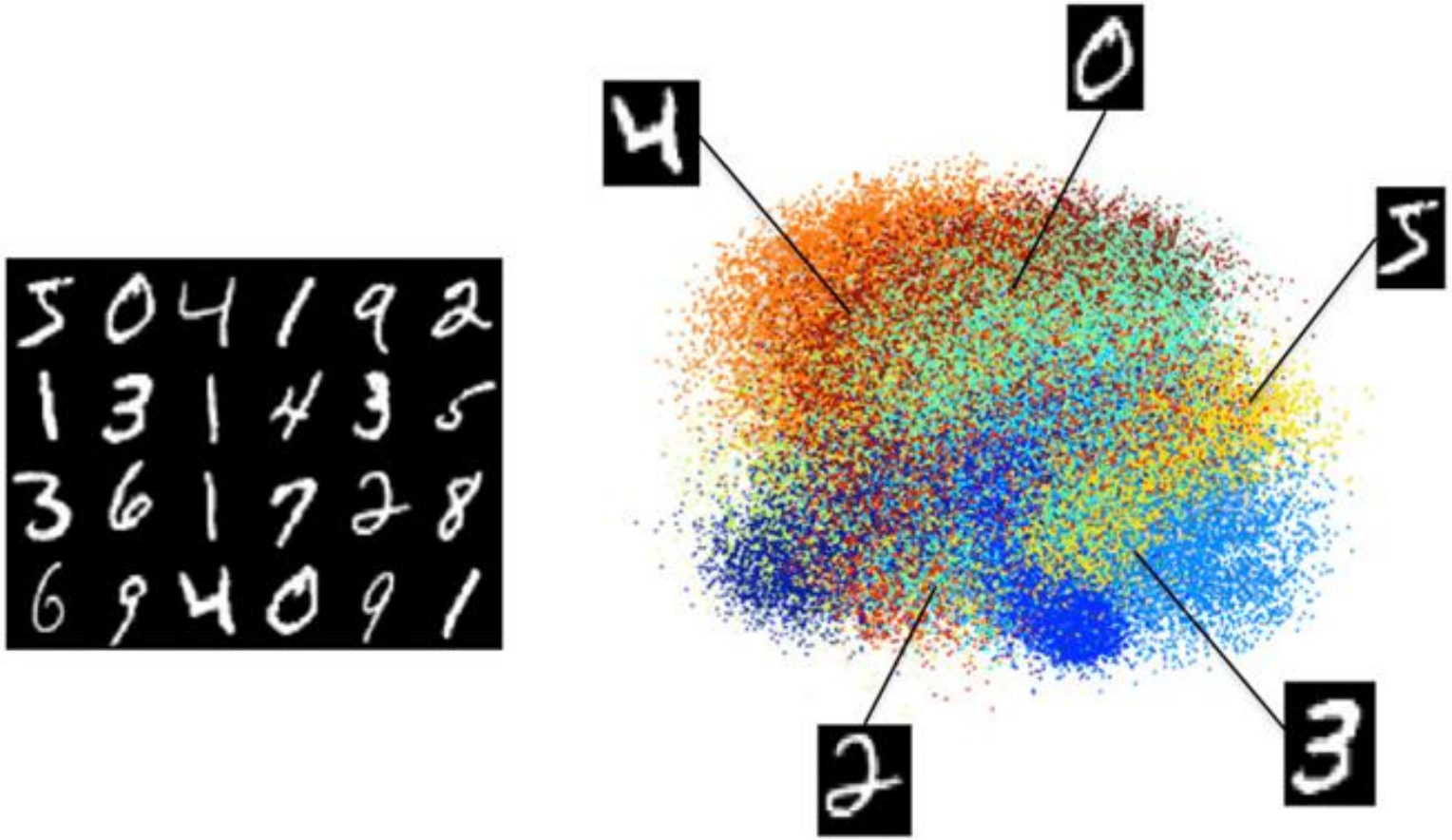
$$s_j^l(t) = \sum_{i=0}^{N_{l-1}} w_{ij}^l(t) \times y_i^{l-1}(t)$$

$$y_j^l(t) = f(s_j^l(t)),$$



Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6):386.

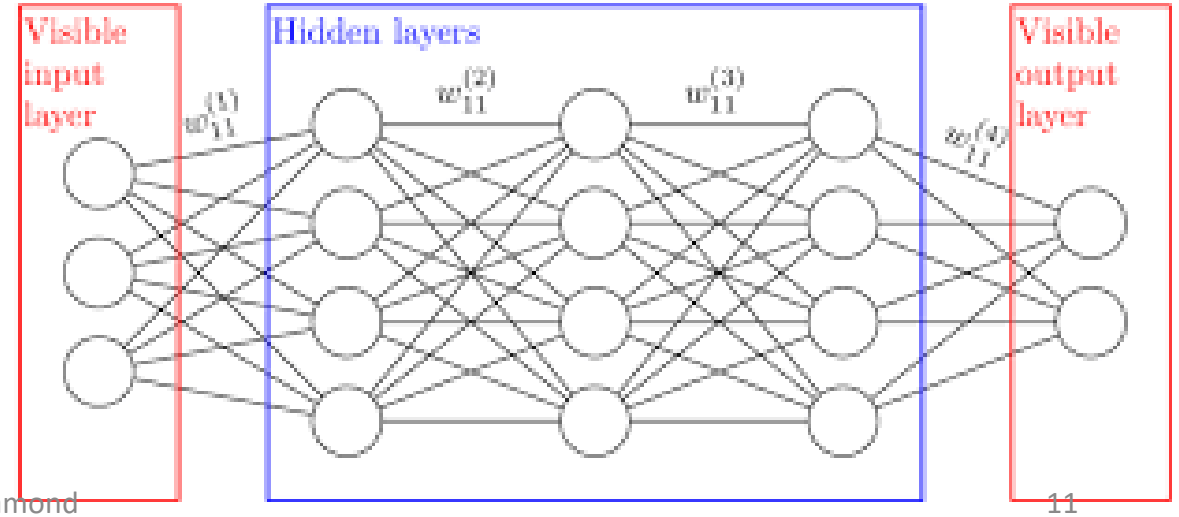
Example of data, the MNIST dataset non-linearly separable data



MLP: Multi-layer perceptron

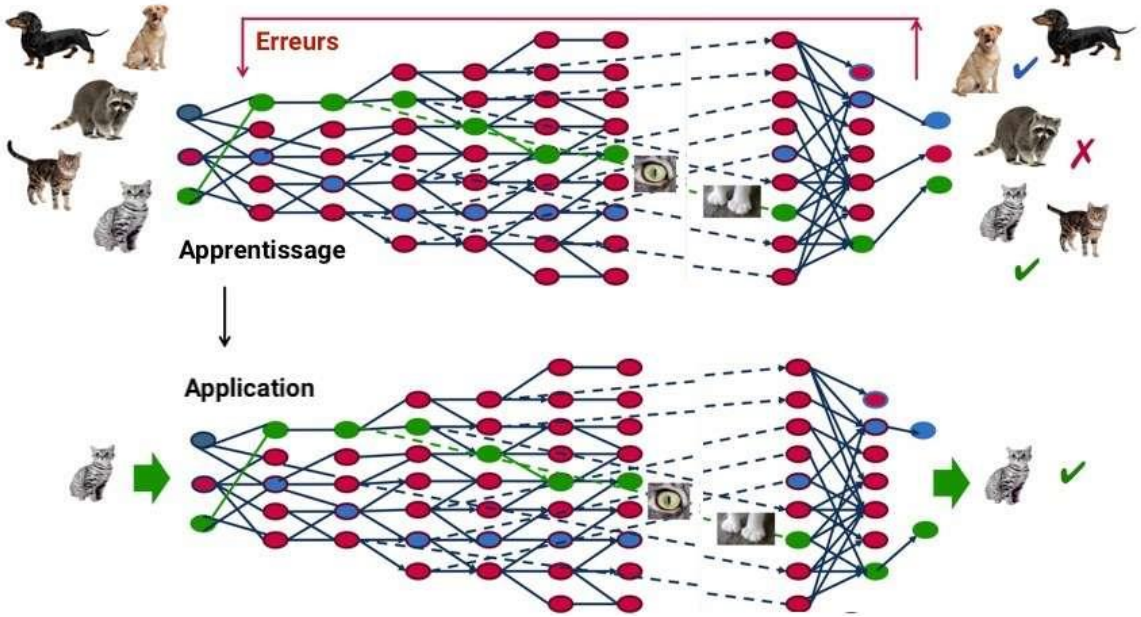
- **Solution:** Combine multiple linear separators.
- Introduction of "hidden" units into NN make them much more powerful: they are no longer limited to linearly separable problems.
- Earlier layers transform the problem into more tractable problems for the latter layers.
- Learning takes place by adjusting

the weights in the network, so that the desired output is produced whenever a training instance is presented.

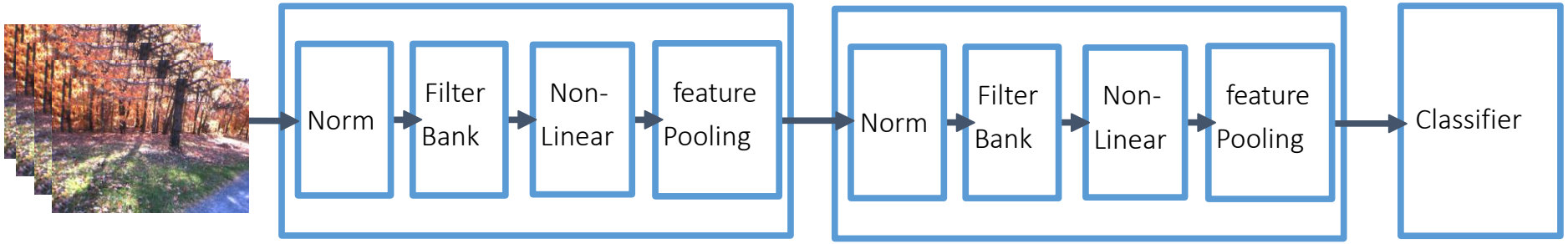


Deep learning

- Often: inputs **x** are **raw signals** or **feature vectors**,
- Often: outputs **y** are vectors which **highest value** indicate the **category of the input**.
- Instead of directly mapping **x** to **y**, constructs a graph of intermediate representations, associated through very simple mathematical functions called **layers**,
- Training: Backpropagate the gradient of the loss throughout the architecture.

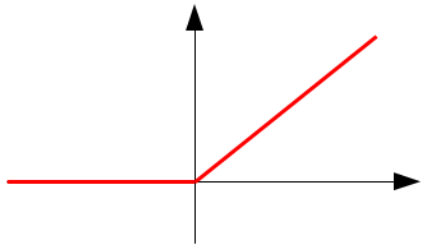


Overall architecture: multiple stages of
Normalization → filter bank → non-linearity → pooling



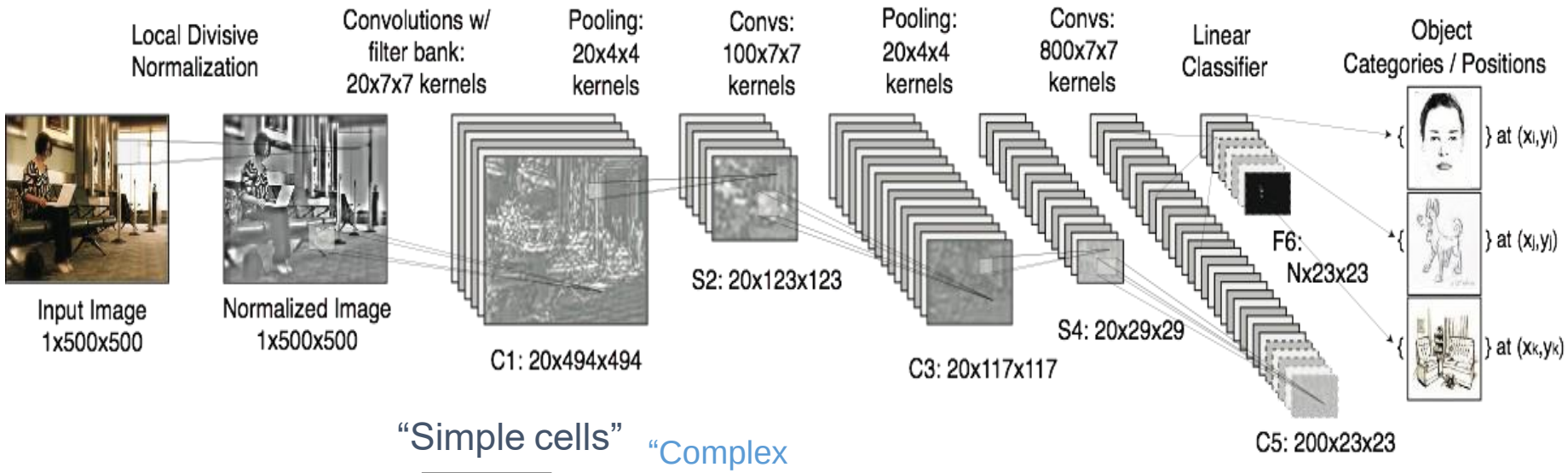
- Normalization: variations on whitening
  - Subtractive: average removal, high pass filtering
  - Divisive: local contrast normalization, variance normalization
- Filter bank: dimension expansion, projection on overcomplete basis
- Non-linearity: sparsification, saturation, lateral inhibition....
  - Rectification (relu), component-wise shrinkage, tanh,..

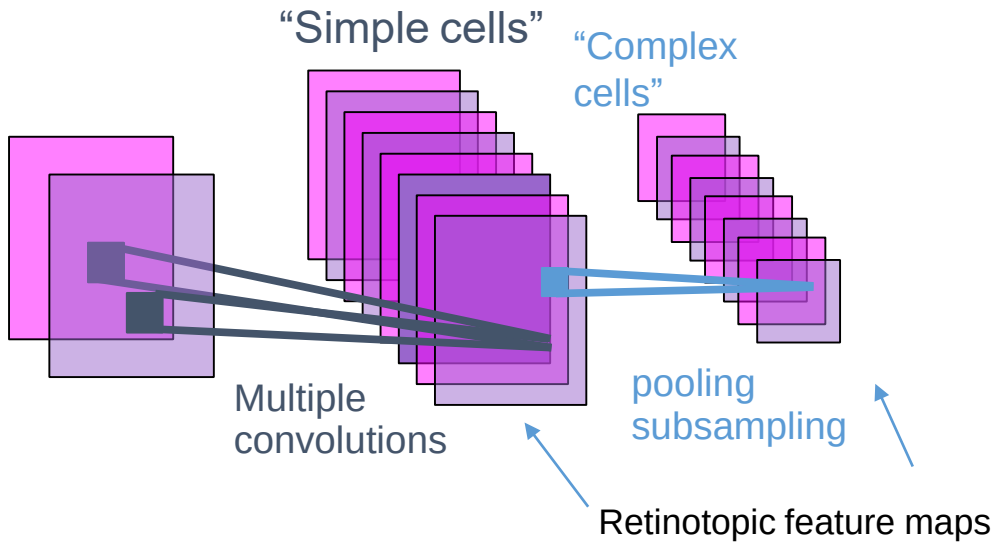
    ReLU (x)=max (x , 0)
- Pooling: aggregation over space or feature type
  - Max, Lp norm, log prob.



$$X_i; \quad L_p: \sqrt[p]{X_i^p}; \quad PROB: \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

The convolutional net model
(Multistage Hubel-Wiesel system)





- Training is supervised
- With stochastic gradient descent

[LeCun et al. 89]
[LeCun et al. 98]

Supervised learning

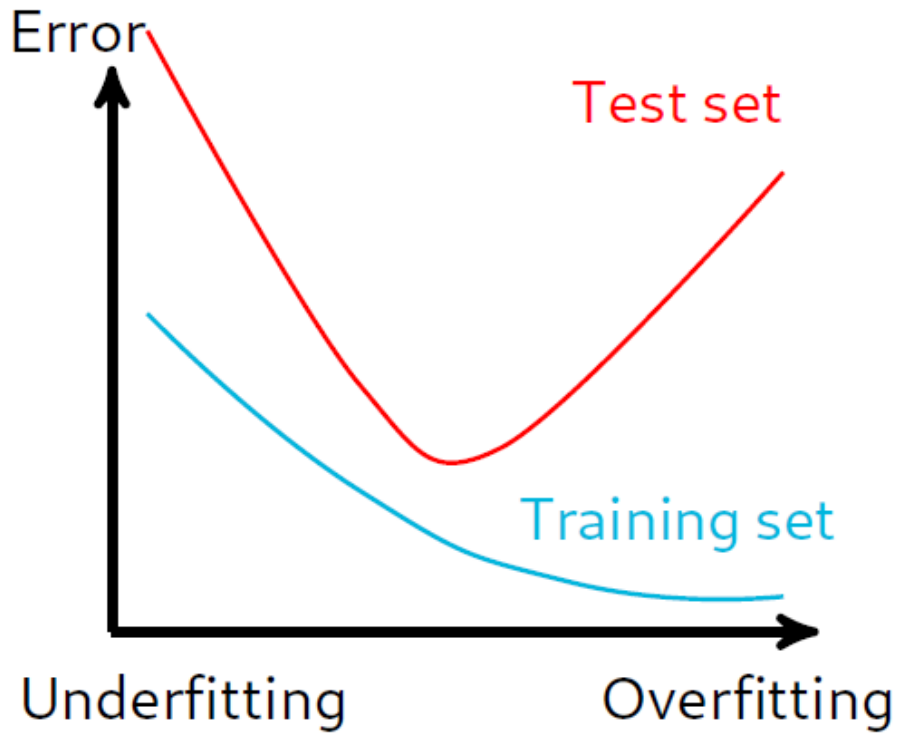
- Supervised learning methods use **labels y** associated with examples **x** to learn a function f such as **y ~=** f(**x**), with the aim of **generalizing** (!=memorizing) to unlabeled examples.

- Labels are encoded as one-hot-bit vectors and

called **targets**,

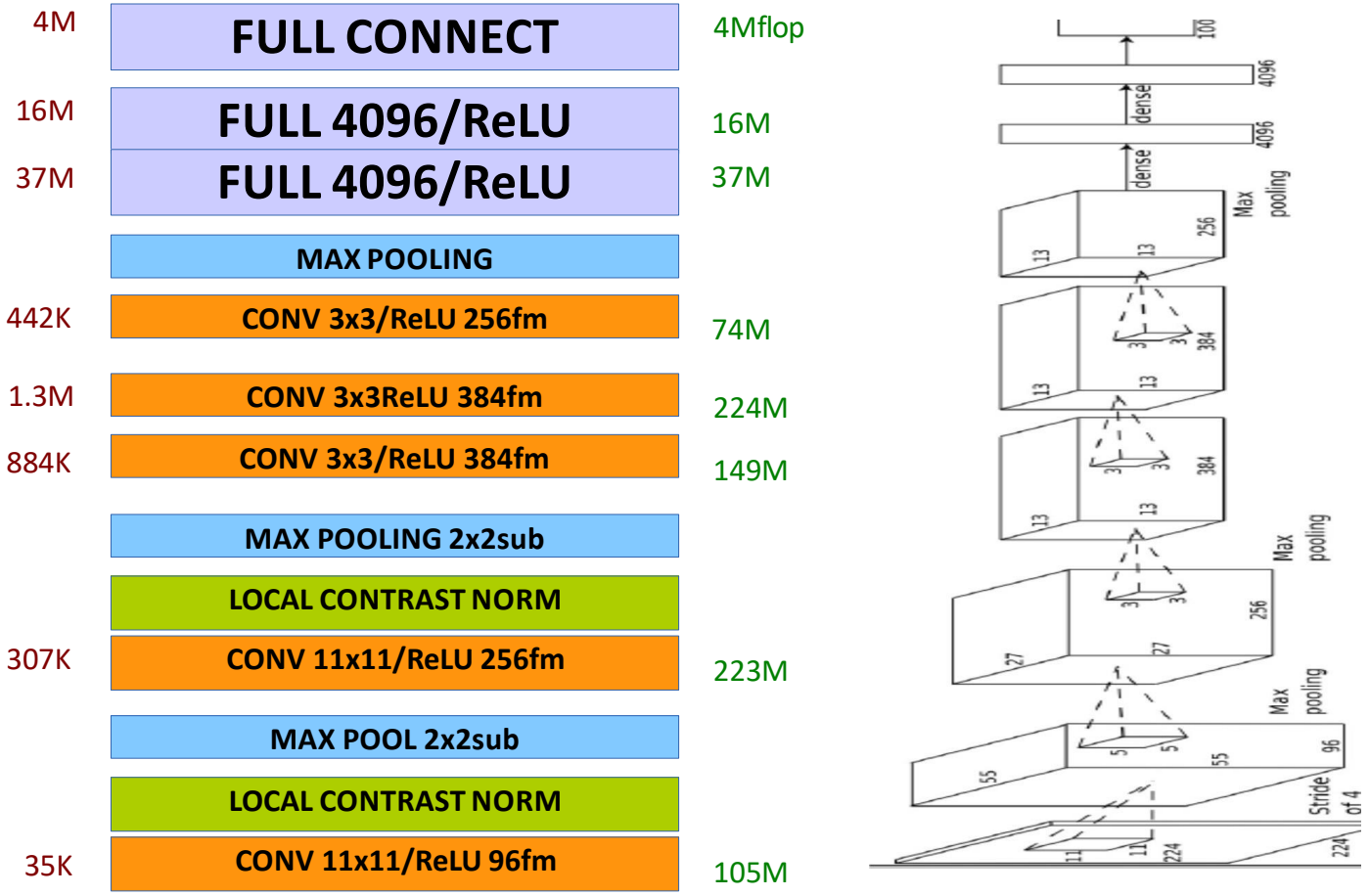
- Outputs are **softmaxed**: **y**i = exp(**y**i)/$\sum$ exp(**y**i)
- Loss is typically **cross-entropy**

- To detect overfitting, split training dataset in two parts:
  - 1 A first part is used to train,
  - 2 A second part is used to validate/test,

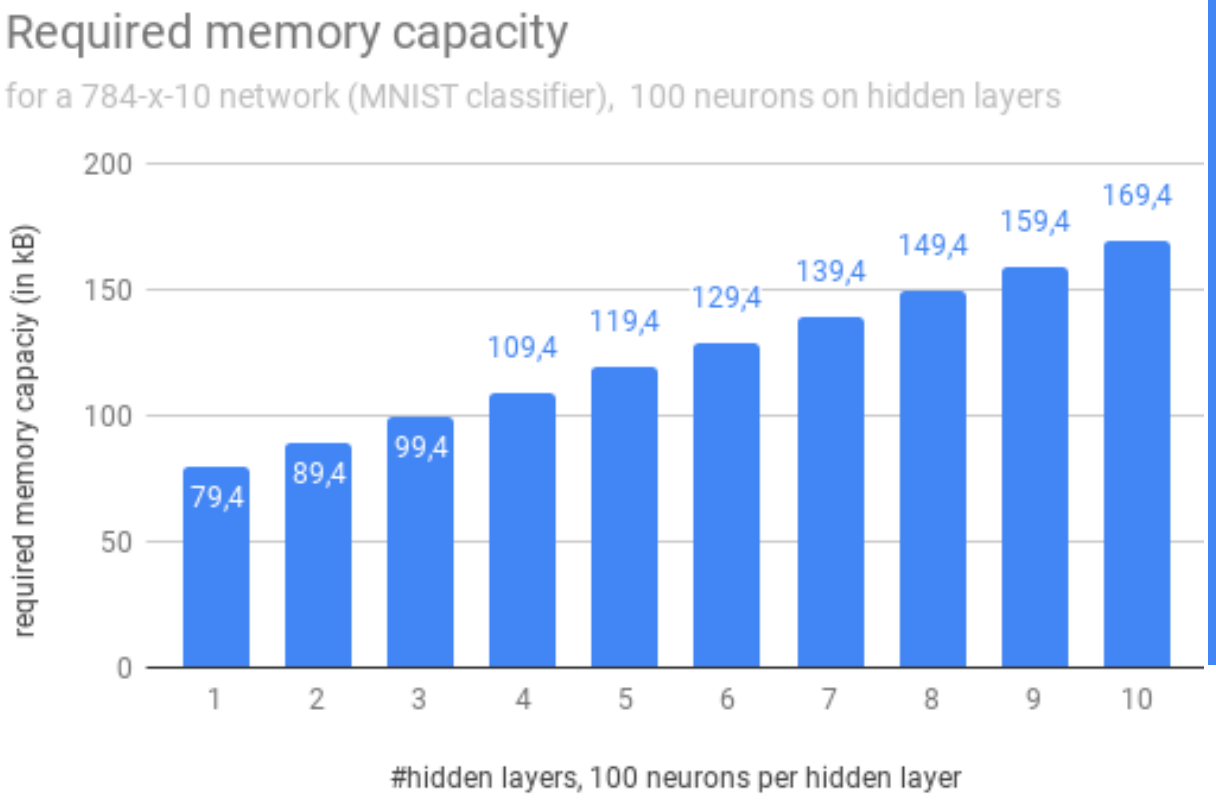


Object Recognition [Krizhevsky, Sutskever, Hinton 2012]

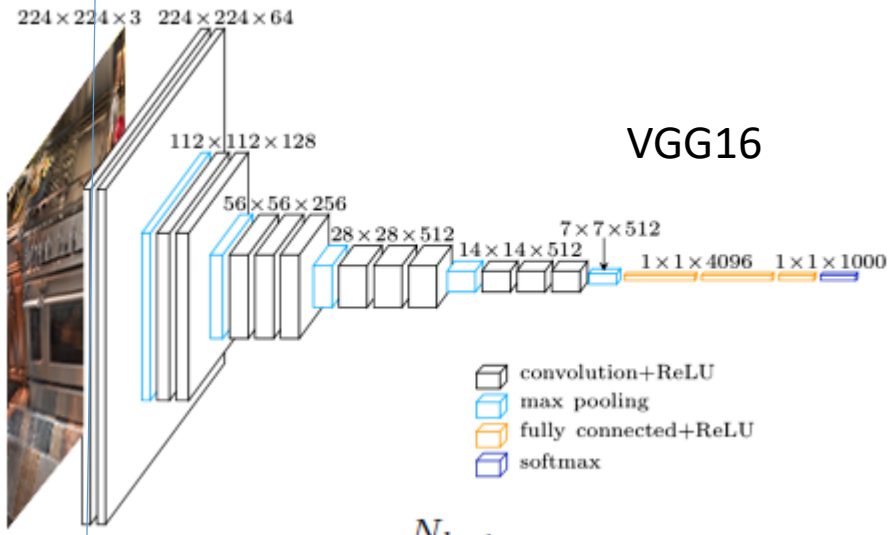
– Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops



The memory wall



230 MB

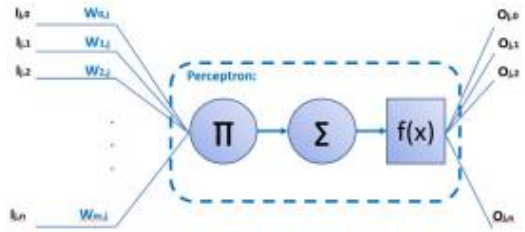


VGG16

$$s_j^l(t) = \sum_{i=0}^{N_{l-1}} w_{ij}^l(t) \times y_i^{l-1}(t)$$

W : Weigths

$$y_j^l(t) = f(s_j^l(t)),$$



Embedded AI on MCU

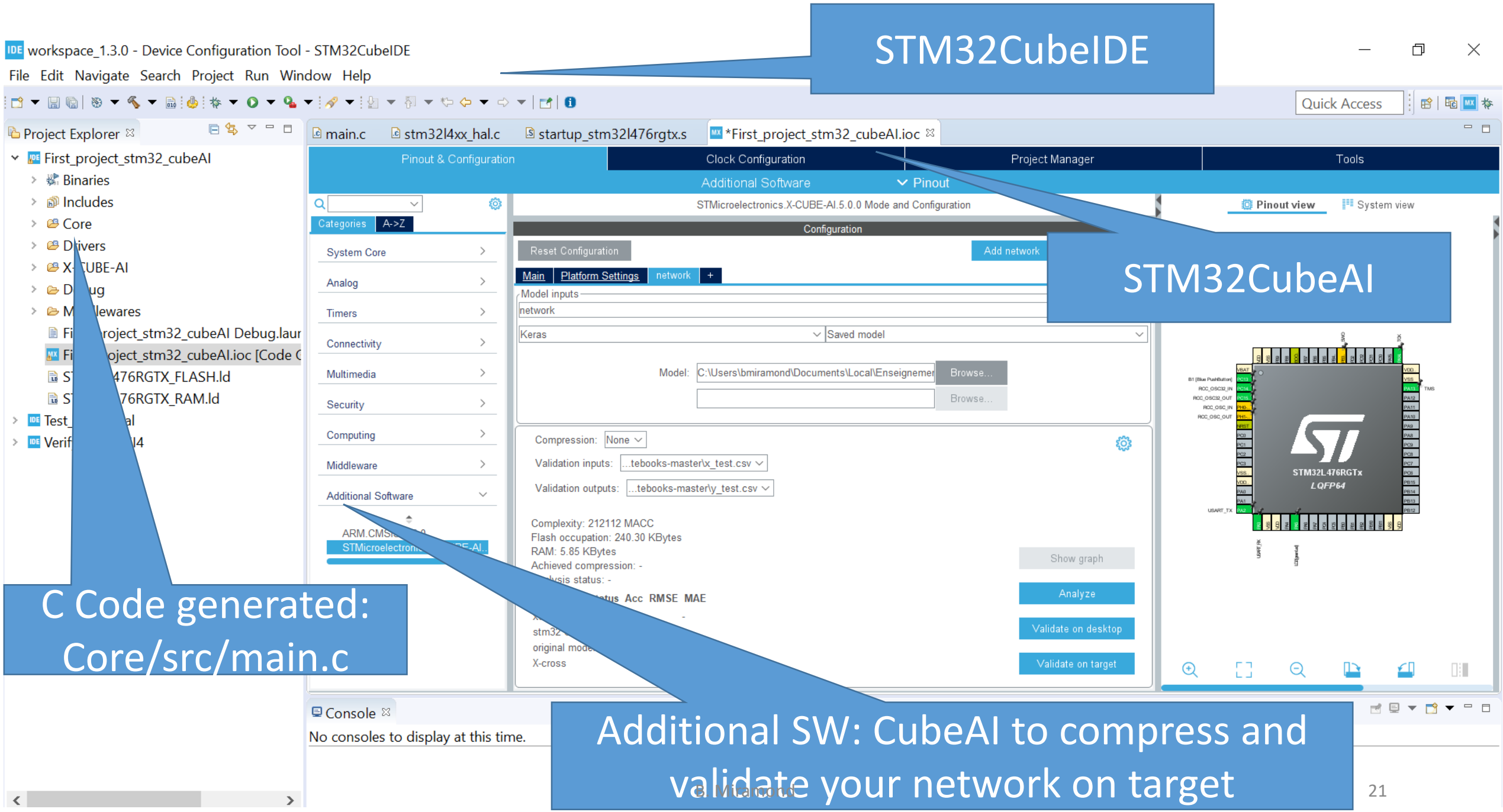
The target Micro Controler Unit (MCU)

STMicroelectronics Nucleo-L476RG

- STM32L476RG MCU
- Cortex-M4F (with FPU) at 80MHz
- 128kB RAM
- 1MB Flash
- 50mW run power consumption (worst case)
- 39µA/MHz (55µA/MHz in reality)
- 6.6µA (7.34µA in reality)
- 3.42 CoreMark/MHz

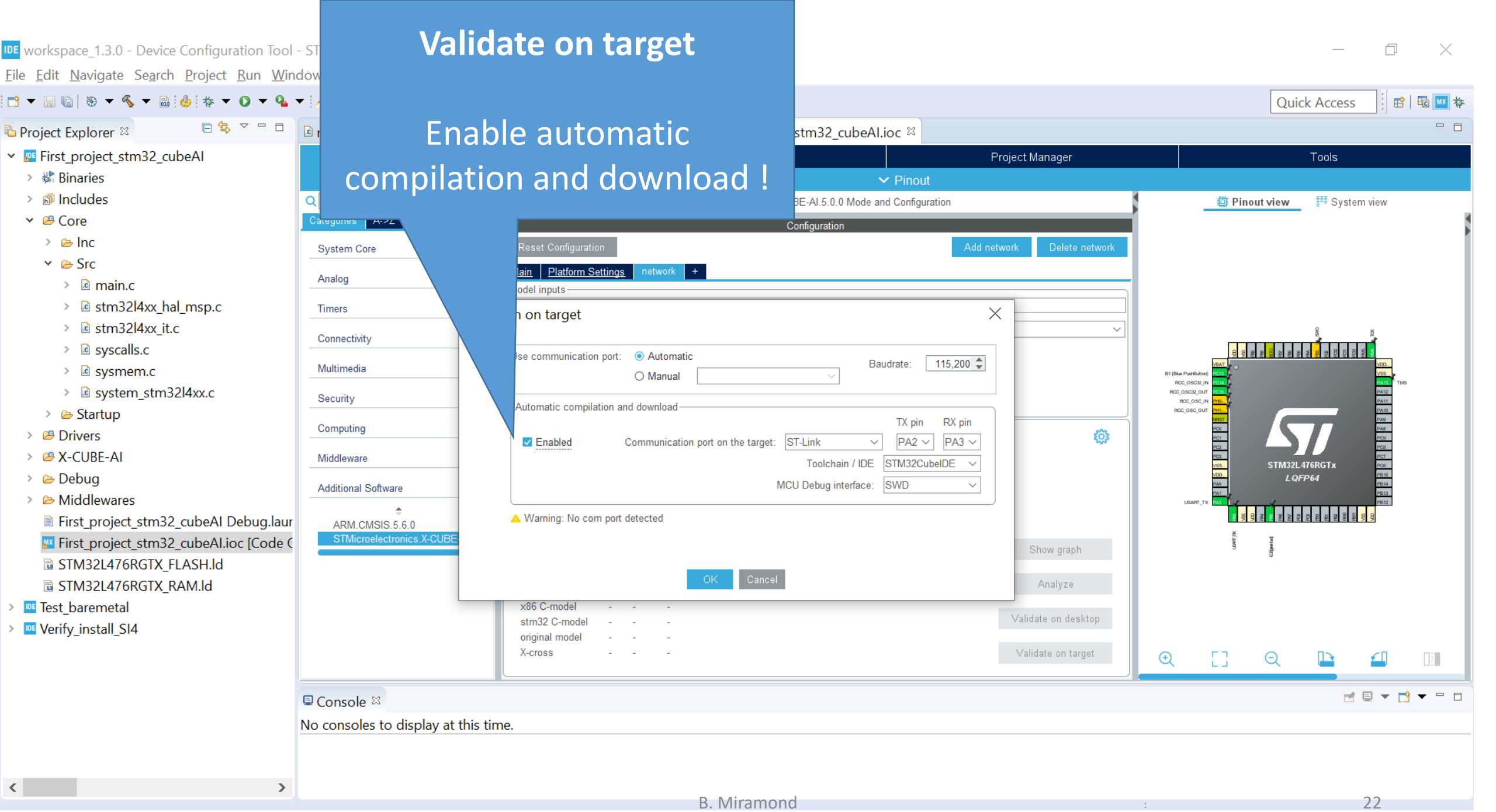
STM32CubeAI

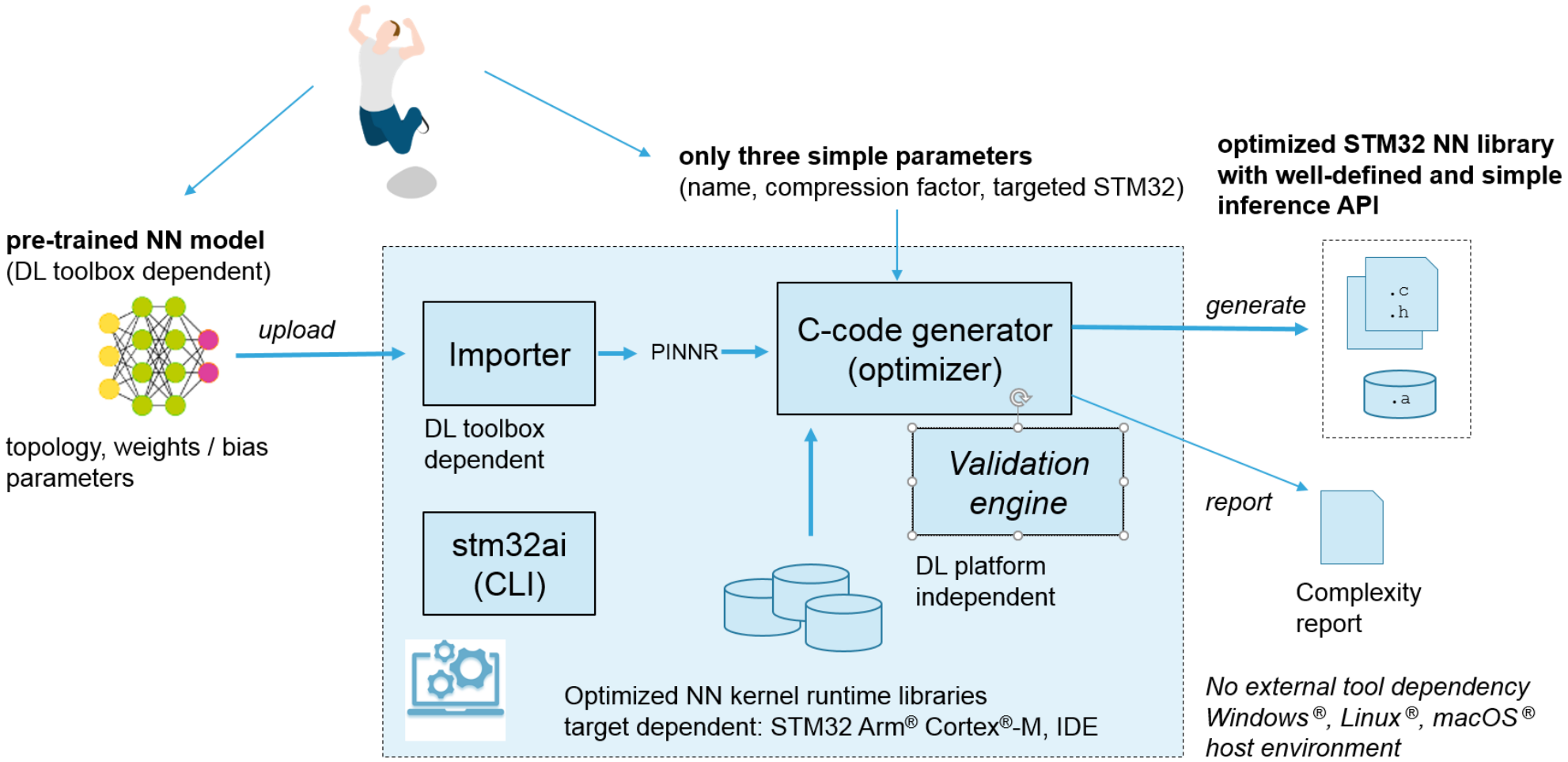
- STMicroelectronics proprietary solution
- Offline learning
- Supports compression
- Supports Keras, TFLite, Lasagne, Caffe, ConvNetJS models
- Limited support for models, no clear details on what is supported
- RAM/ROM occupation, inference time, accuracy after compression metrics
- Integrated to STM32CubeIDE with validation application
- Limited to STM32 MCU


STM32CubeIDE
workspace_1.3.0 - Device Configuration Tool - STM32CubeIDE
File Edit Navigate Search Project Run Window Help
Quick Access
Project Explorer
First_project_stm32_cubeAI
Binaries
Includes
Core
Drivers
Debug
Middlewares
First_project_stm32_cubeAI Debug.laur
First_project_stm32_cubeAI.ioc [Code G
STM32L476RGTX_FLASH.ld
STM32L476RGTX_RAM.ld
Test_cubeAI
Verify_cubeAI
main.c
stm32l4xx_hal.c
startup_stm32l476rgtx.s
*First_project_stm32_cubeAI.ioc
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Additional Software
Pinout
STMicroelectronics.X-CUBE-AI.5.0.0 Mode and Configuration
Categories
A->Z
System Core
Analog
Timers
Connectivity
Multimedia
Security
Computing
Middleware
Additional Software
Configuration
Reset Configuration
Add network
Main
Platform Settings
network
Model inputs
network
Keras
Saved model
Model:
C:\Users\bmiramond\Documents\Local\Enseignemer
Browse...
Browse...
Compression:
None
Validation inputs:
...tebooks-master\x_test.csv
Validation outputs:
...tebooks-master\y_test.csv
Complexity: 212112 MACC
Flash occupation: 240.30 KBytes
RAM: 5.85 KBytes
Achieved compression: -
Acc RMSE MAE
X-cross
Show graph
Analyze
Validate on desktop
Validate on target
Pinout view
System view
STM32L476RGTx
LQFP64
Console
No consoles to display at this time.
STM32CubeAI
C Code generated:
Core/src/main.c
Additional SW: CubeAI to compress and validate your network on target

Validate on target

Enable automatic compilation and download !

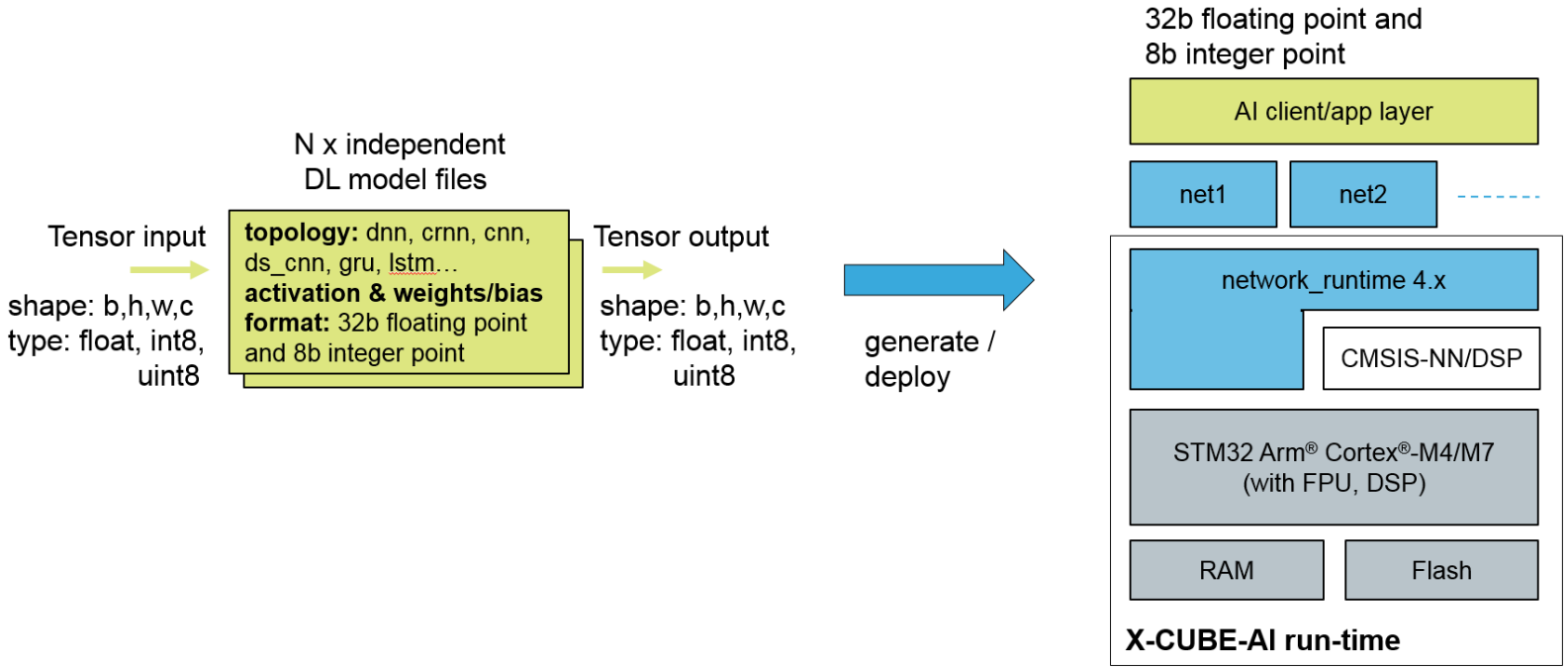




PINNR: Platform-independent Neural Network representation is a file generated by the front end (X-CUBE-AI core importer) to have a common and portable internal representation of the uploaded DL model for the next stages (optimizer and C-code generator).

Floating and fixed point implementation

- Only simple tensor input and simple tensor output are supported
  - 4-dim shape: batch, height, width, channel (*"channel-last"* format)
  - Floating-point (32b) and fixed-point (8b) types
- Generated C models are fully optimized for STM32 Arm® Cortex®-M4/M7 cores with FPU and DSP extensions
- The objective of this technique is to reduce the model size
- while also improving the CPU and hardware accelerator latency (including power consumption aspects) with little degradation in model accuracy.



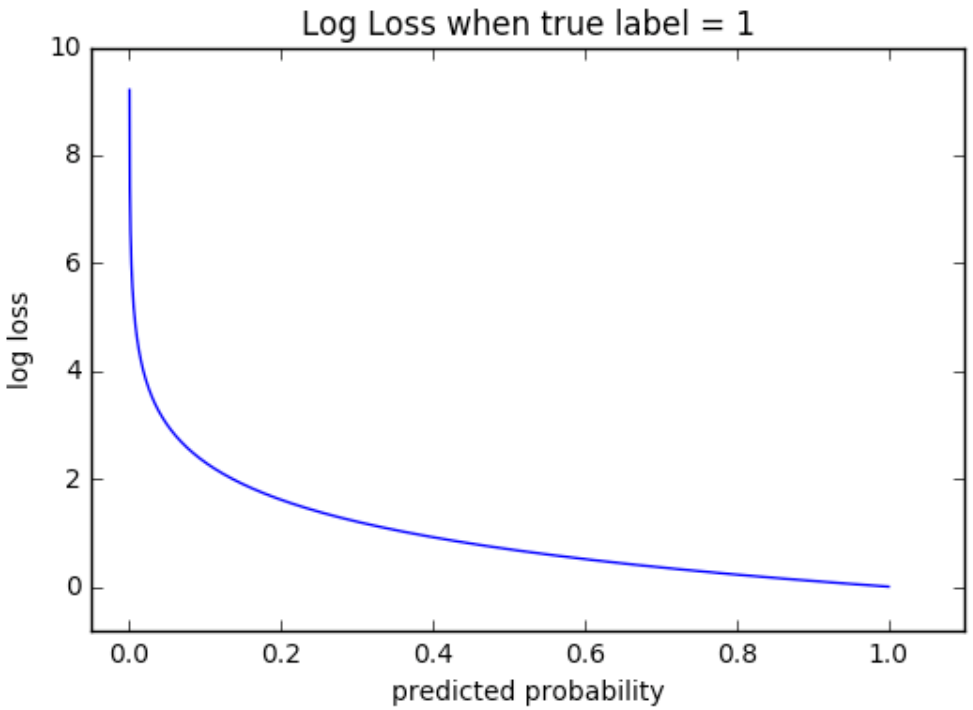
for a 32-bit floating-point C model:

- STM32 Arm® Cortex®-M4: ~9 cycles/MACC
- STM32 Arm® Cortex®-M7:- ~6 cycles/MACC

Validation of models

Metric	Meaning	Purpose
ACC	Classification accuracy (in %)	Recognition
Loss	nonnegative measure of the discrepancy between expected output and obtained output. Example: output should be [0; 1] but is [0:2; 0:8]	Training only
RMSE / MSE	Root Mean Square Error	
MAE	Mean Absolute Error	
Latency	Duration of inference on the complete Test Dataset	
ROM	Memory footprint of the model parameters in ROM	
RAM	Memory footprint of the model activities in RAM	

Example of cross entropy



Statistical number of validations/inferences

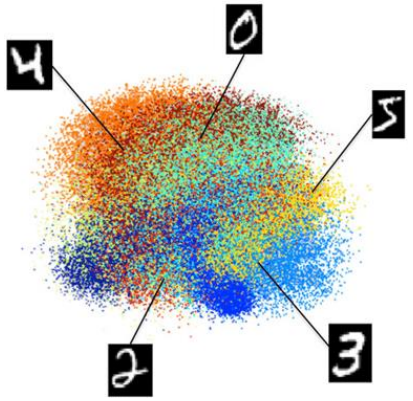
	Number of learning	Number of test validation	Result	
Learning under TF	3 for each configuration	1 per learning	Select the best configuration for next steps	
Validation on STMCubeAI	NO	1 per compression factor		
Validation on target	NO	1 per compression factor		

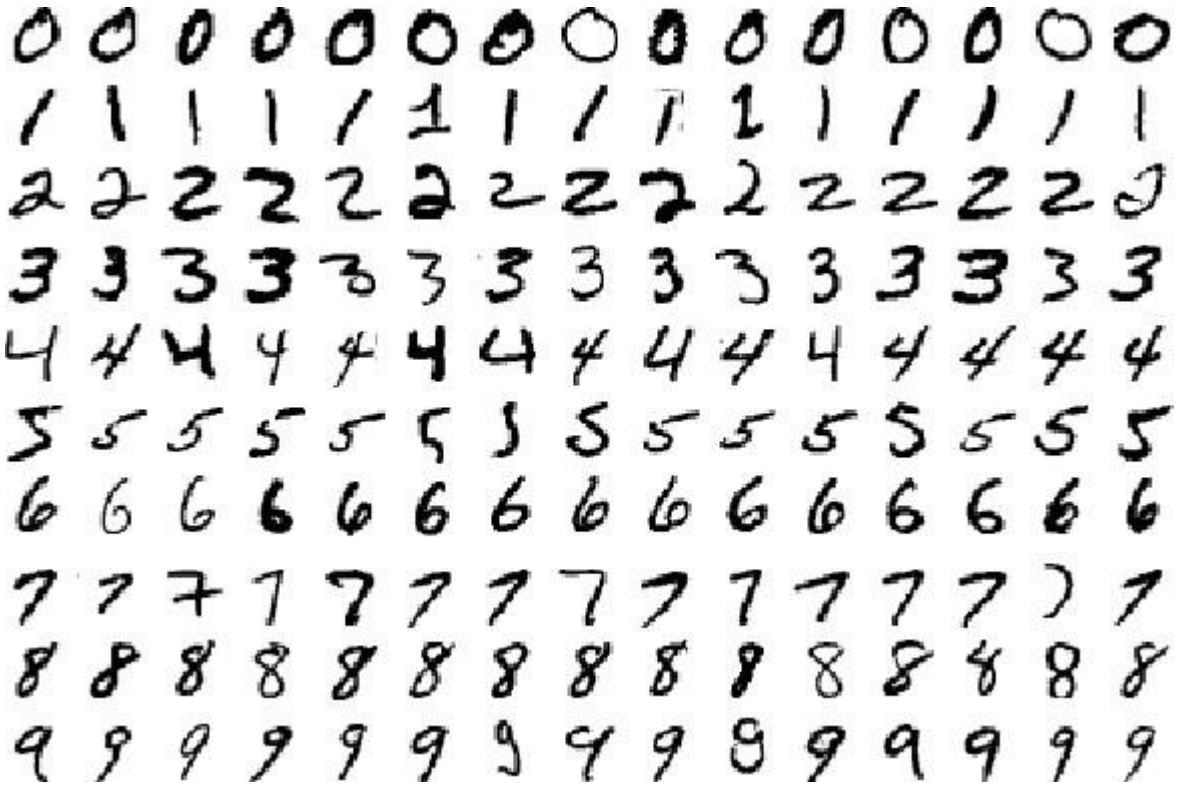
- A network has to be statistically validated by several learnings
- The inference of this same network can be made only once for each learning.

Dataset

MNIST Dataset

- Mixed National Institute of Standards and Technology
- Manuscript Digits from 0 to 9
- Black&White Images 28x28
- 60 000 training data
- 10 000 test data





Example of results on a simple CNN

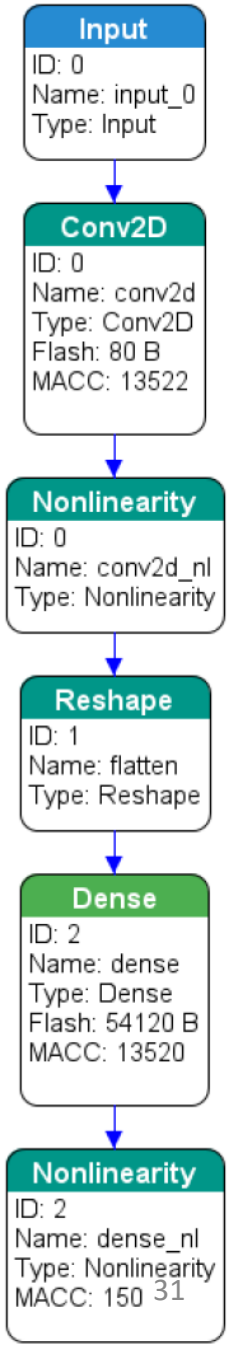
Results from tensorflow

Layer	Output shape	Number of parameters	Kernel (If conv2D)
Input	(28, 28, 1)	-	
Conv2D layers	(26, 26, 2)	20	3x3
Flatten	(1352)	–	
Dense layers	(10)	13,530	
Total trainable parameters	13,550		
Number of Epochs	3		

Results from STMCubeAI

	Type	Param #	MACC	MACC (%)	ROM (bytes)	ROM (%)	Bytes per Param
Layer 1	Conv2D	20	13 522	49,7	80	0,1	4
Layer 2	Dense	13530	13 520	49,7	54,120	99,9	4
…	DenseNL		150	0,6		0	
TOTAL			**27192**	**100**	**54,200**	**100**	**4**

Total memory
128kB RAM
1MB Flash



Validation on desktop

Results during inference	Number of inferences (test dataset)	Accuracy	RMSE	MAE	MACC	ROM (bytes)
Original model (result from TF)	250	96,3	0.069338	0.0131	27,192	54,200
Without compression		97,2	0.069338	0.013095	27,192	54,200
Compression X4		97,2	0.069323	0.013082	27,192	14,664
Compression X8		97,2	0.070925	0.013352	27,192	6,944

Validation on target

Results during inference	Number of inferences (test dataset)	Accuracy without compression	RMSE	Total latency (ms)	CPU cycles	Cycles / MACC
Original model (result from TF)	250	97,2	[redacted]			
Model validated on Laptop without compression		97,2				
Model validated on Target without compression		97,2	0.069338	5.413	433025	15.92
Model validated on Target with compression X4		97,2	0.069323	5.758	460646	16.94
Model validated on Target with compression X4		97,2	0.070925	5.462	436952	16.07

Labs sessions

- **Lab 1: installation and first execution of CNN on MCU**
  - **Validation on MNIST dataset**
  - **Learn a model and optimize it**
- Lab 2: optimization of CNN on MCU v1
  - Propose a model, learn and test
  - Application on MNIST and HAR dataset
- Break
- Lab 3: Apply to real data
  - Get data either from the shield or from the smartphone
  - Constitute a specific test set
  - Test onto the MCU
- Lab 4: Introduction to MicroAI
  - Directly export and generate code from TF