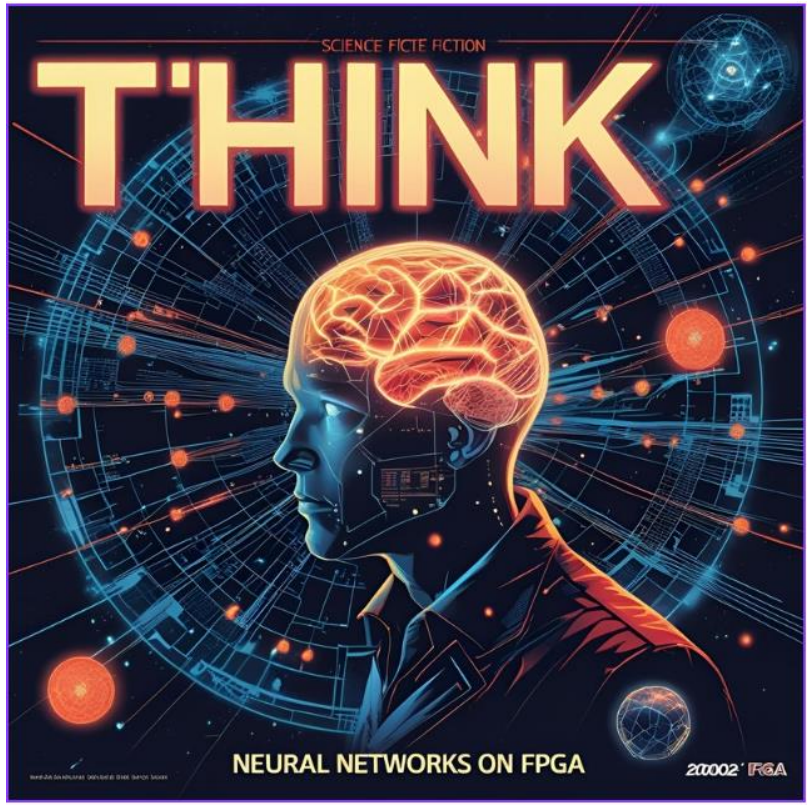
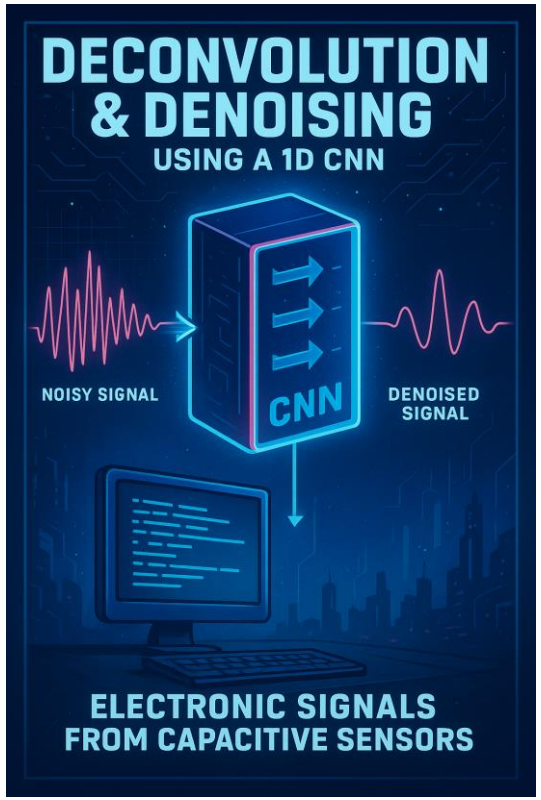
– Denoizing and deconvolution signal from 1D CNN–





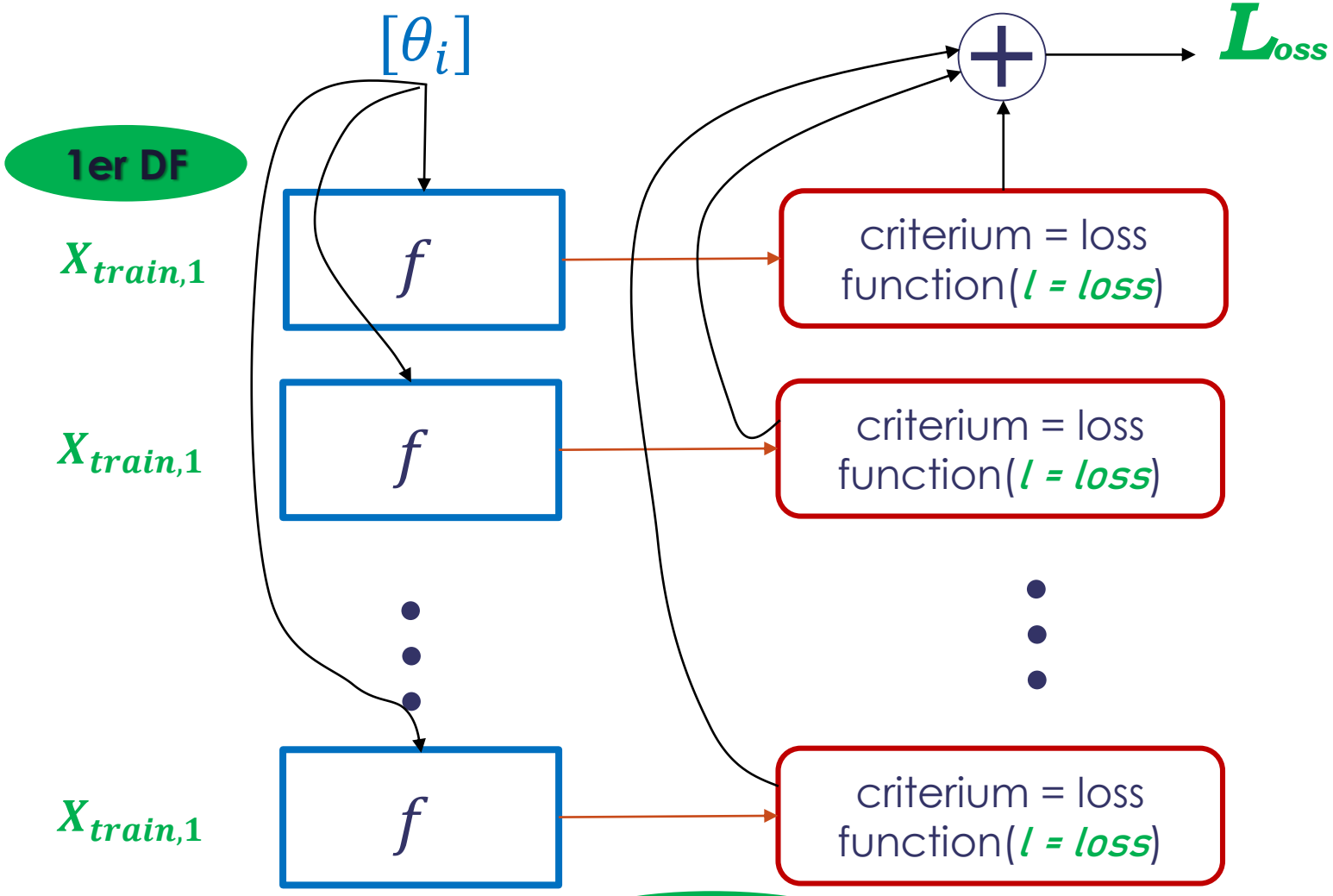
Test of Hardware Inferring Neural networK

Reseau DAQ – Hardware-Firmware-Software-

Phase 2

Neural network principles

Minimize un criterium



$$\tilde{\theta}^* = \underset{\theta}{argmin} \sum_{i=0}^{N} l(Y_{train,i}, f(X_{train,i}; \theta))$$

3em DF

$Y_{train,1}$

$Y_{train,1}$

$Y_{train,1}$

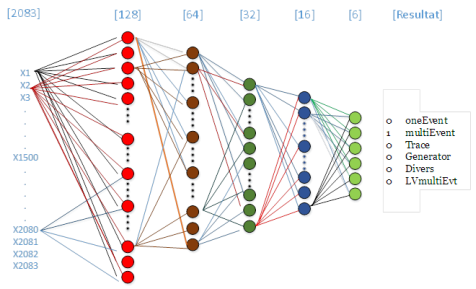
Model constraints:
Error
ODE
Etc…

4em DF

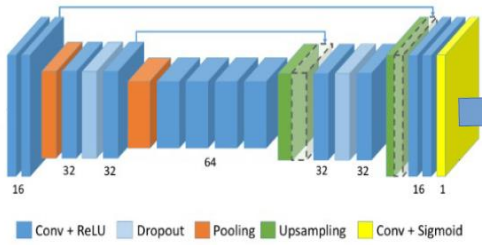
LEARNING STEP:
Gradient Descent Algo.

Les elementary bricks

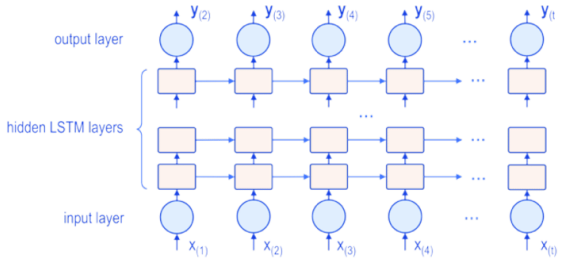
DEEP NEURAL NETWORK

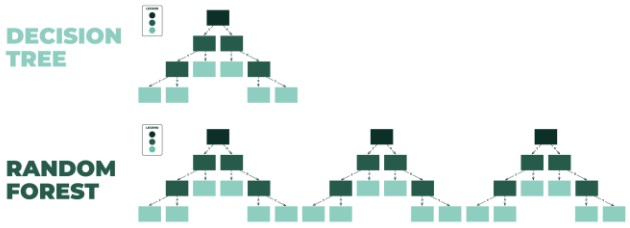


CONVOLUTIONAL NEURAL NETWORK

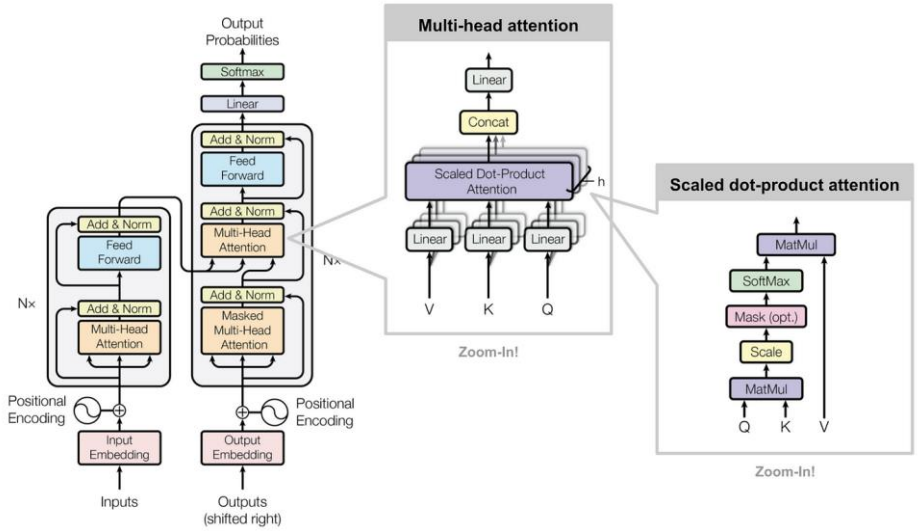


RECURRENT NEURAL NETWORK





LLM



Embedded

Off-line

- Signal generation
- Design Optimisation

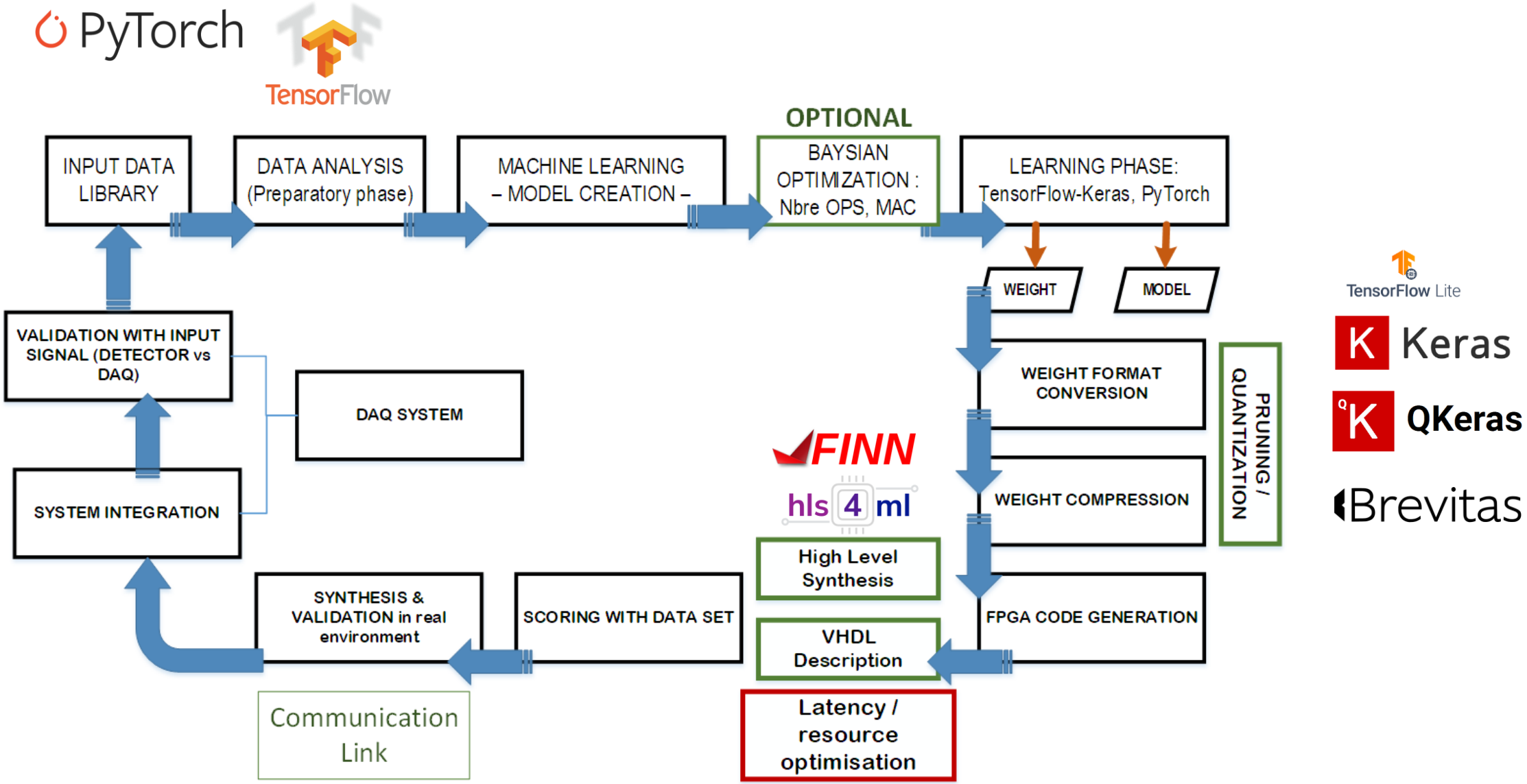
On-line

- Instruments Optimization
  - Signal recognition
  - Pile-Up recovery
  - Signal deconvolution
- Selection/ Classification/ Decision
  - Data selection
  - Data parameters prediction
  - Denoizing
- Data Compression
  - Reduced data format

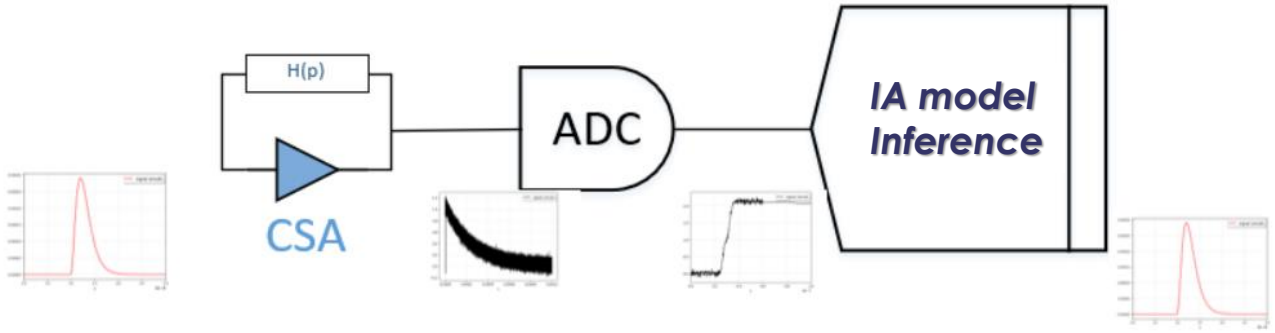
IA Components comparaison

Critères	CPU	CPU+AI-CoPro	GPU	FPGA	ASIC
Adaptability (to various model)	**High**	**High**	Moyen	**Low**	Aucune
Calculus power	**low**	Mean-High	**High**	**High**	Mean
Latency	Mean (ms)	Mean (ms)	**Faible** (µs)	**Faible**(10 ns)	**Très faible**
Input stream	**Low**	Mean	**High**	**High**	**High**
Parallelism	**Low**	Mean	**High**	**High**	**High**
Electrical power consumption	Mean	Mean	Mean	Mean	**High**
Easy to use	**Facile**	Moyen	Mean	**Complex**	**Very Complex**
Density of model	Moyen	**Low**	**Low**	**High**	**High**

Embedded ML methodology



Problem statement



Data Inputs

Data Outputs

Pre-processing scaling → **DL Model inference** → **Post-processing Scaling**

➔ Minimize latency & used ressources

- Memory BW
- Data Size

$$T_m = \frac{M}{B_{mem}}$$

- Pre-processing input size
- Norm. Elt time

$$T_{pre} = C_{pre} \cdot d$$

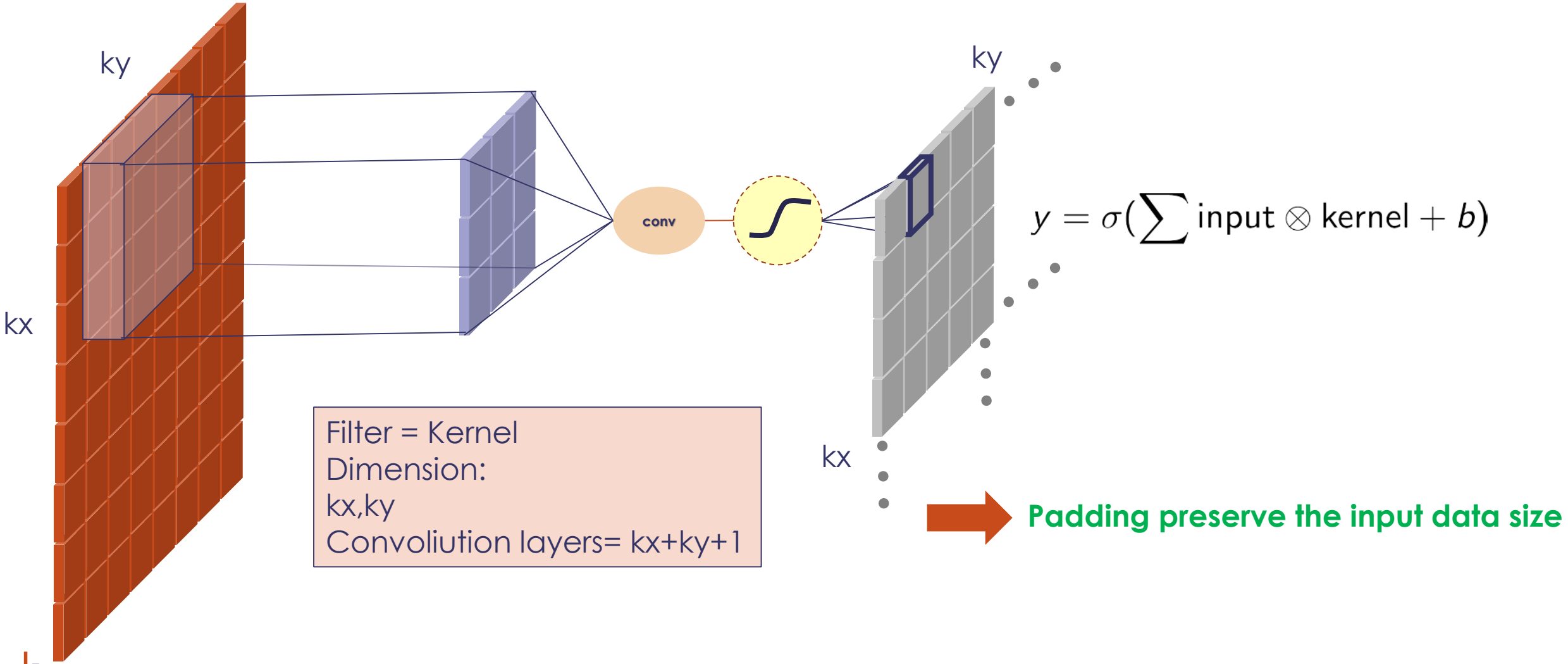
$$T_{comp} = \frac{2.m.n}{P_{ops}}$$

- One layer calculation
- MAC speed

$$T_{total} = T_{mem} + T_{pre} + T_{comp}$$

Did you say « Convolution » ?

- Jupyter lab
- https://github.com/vdumoulin/conv_arithmetic



Filter = Kernel
Dimension:
kx,ky
Convoliution layers= kx+ky+1

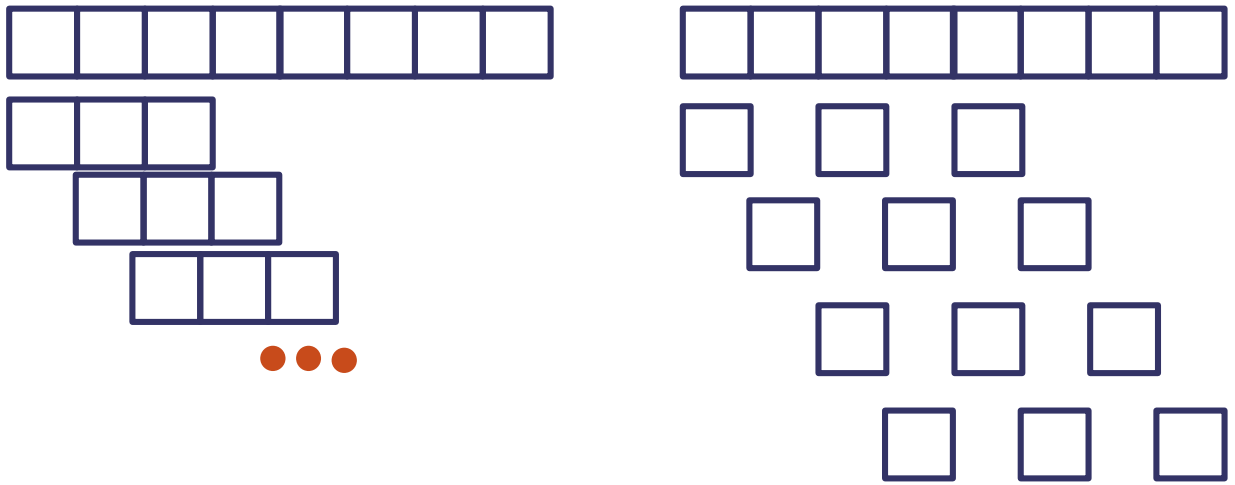
Padding preserve the input data size

Objectif : build original signal of a capacitive detector from a noisy and filtered signal.

Application : Front-End signal processing.

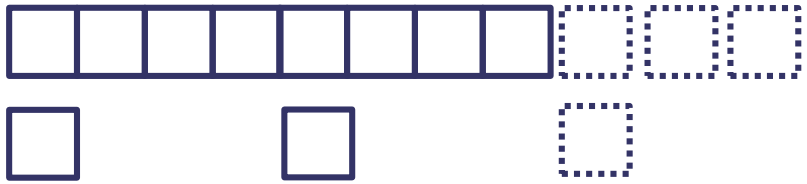
1D Classical CNN

$$y[t] = \sum_{k=0}^{K-1} w_k \, x[t-k]$$



1D Dilated CNN

$$y[t] = \sum_{k=0}^{K-1} w_k \, x[t - r \cdot k] \qquad R = 1 + (K-1)\sum_{l=0}^{L-1} r_l$$



DECONVOLUTION: Problem statement

Exemples:

- **x=[x0,x1,x2,x3,x4,x5,x6,x7]**
- Noyau de taille K=3K = 3K=3,

avec coefficients w=[w0,w1,w2]

- Pas (*stride*) = 1
- Pas de padding (donc la sortie rétrécit quand on dilate)

R = 2, Layers 2

$$y[t] = w_0 x[t] + w_1 x[t-2] + w_2 x[t-4]$$

Calcul explicite

t	indices utilisés	$y[t]$
4	[4,2,0]	$w_0x_4 + w_1x_2 + w_2x_0$
5	[5,3,1]	$w_0x_5 + w_1x_3 + w_2x_1$
6	[6,4,2]	$w_0x_6 + w_1x_4 + w_2x_2$
7	[7,5,3]	$w_0x_7 + w_1x_5 + w_2x_3$

R = 1, Layers 1

$$y[t] = w_0 x[t] + w_1 x[t-1] + w_2 x[t-2]$$

t	indices utilisés	$y[t]$
2	[2,1,0]	$w_0x_2 + w_1x_1 + w_2x_0$
3	[3,2,1]	$w_0x_3 + w_1x_2 + w_2x_1$
4	[4,3,2]	$w_0x_4 + w_1x_3 + w_2x_2$
5	[5,4,3]	$w_0x_5 + w_1x_4 + w_2x_3$
6	[6,5,4]	$w_0x_6 + w_1x_5 + w_2x_4$
7	[7,6,5]	$w_0x_7 + w_1x_6 + w_2x_5$

R = 4, Layers 3

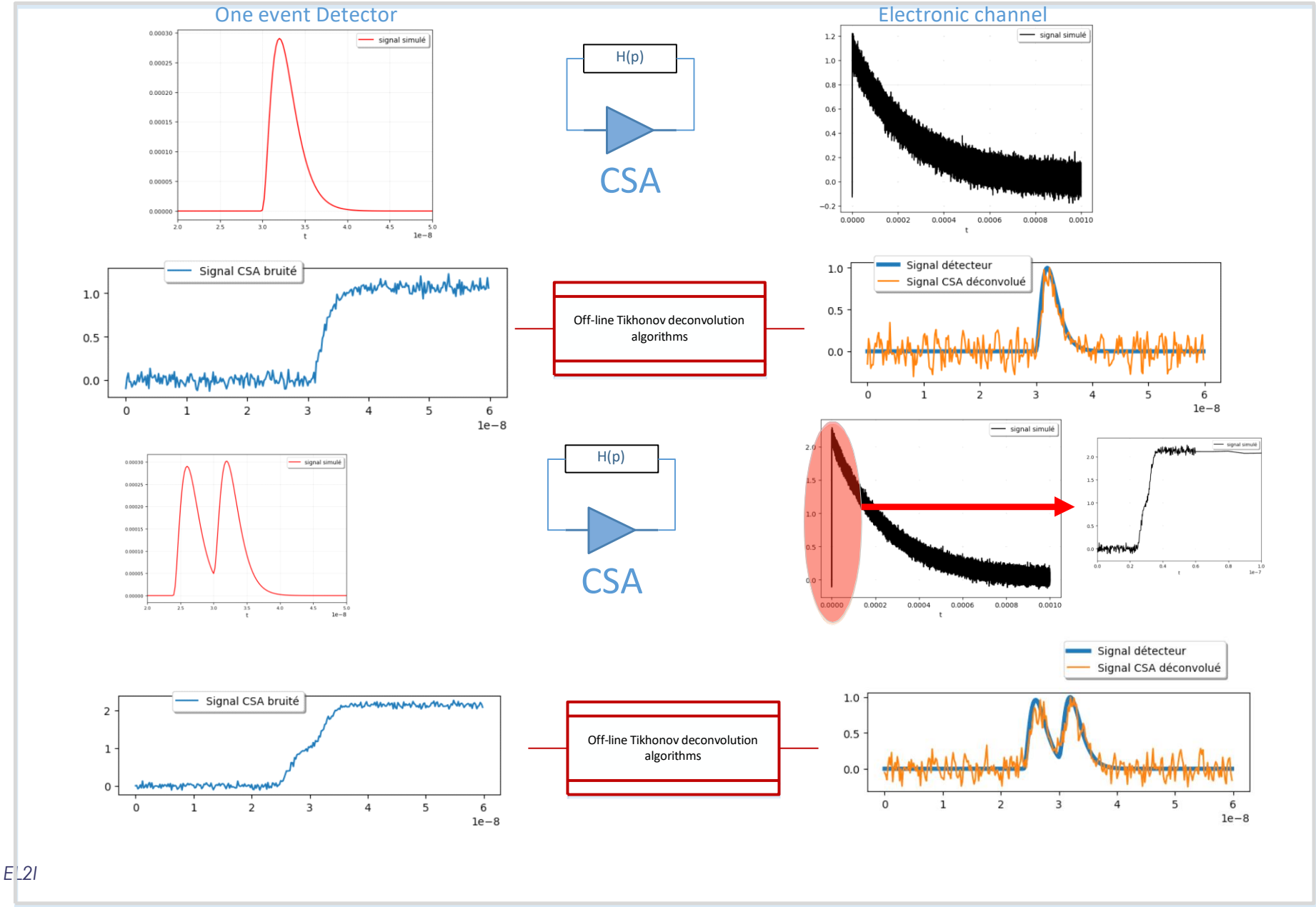
$$y[t] = w_0 x[t] + w_1 x[t-4] + w_2 x[t-8]$$

t	indices utilisés	$y[t]$
7	[7,3,−1]	$w_0x_7 + w_1x_3 + w_2x_{-1}$ → nécessite padding ou tronquage

Deconvolution off-line with Tikhonov Algorithms

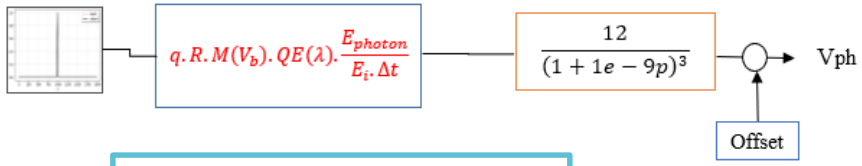
Ill-problem in Hadamard' principle:

➔ Off-Line Algorithme



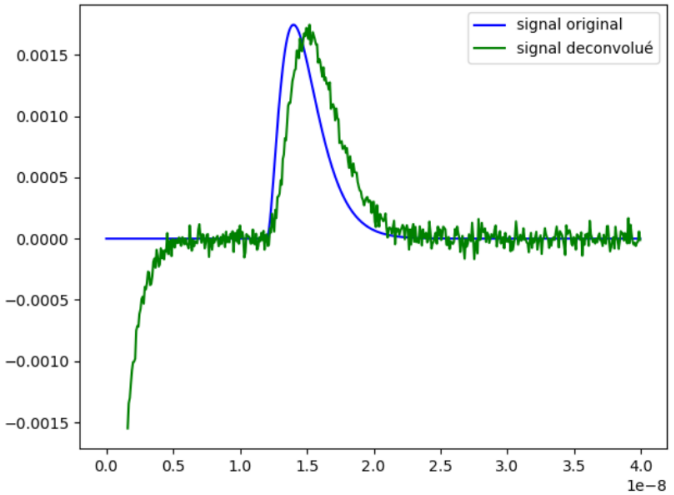
Deconvolution with IA: Test signal APD

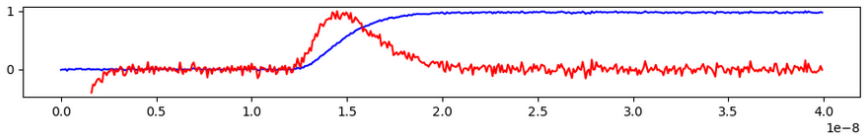
APD Signal : same as capacitif detector signal

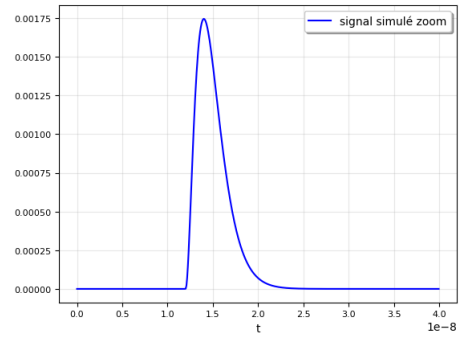


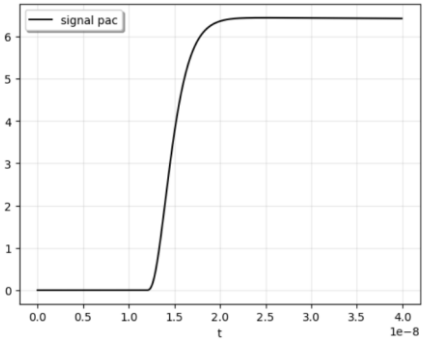
PAC : R/(1+RCp)
R=5MΩΩ Tau=5µs

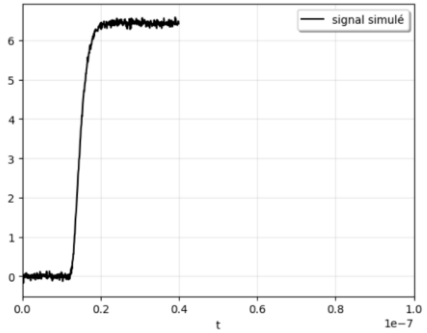
Déconvolution Tikhonov

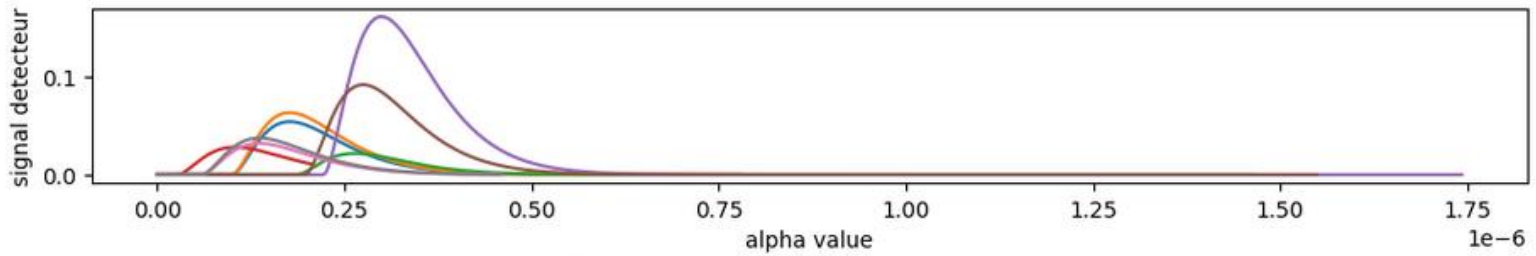


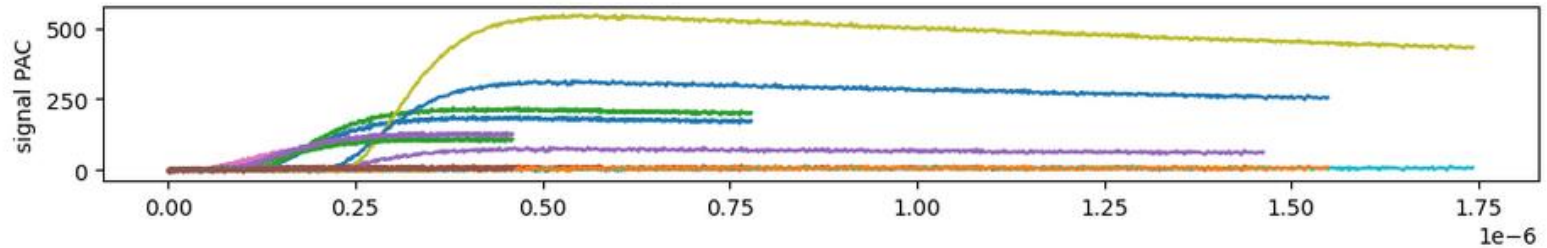


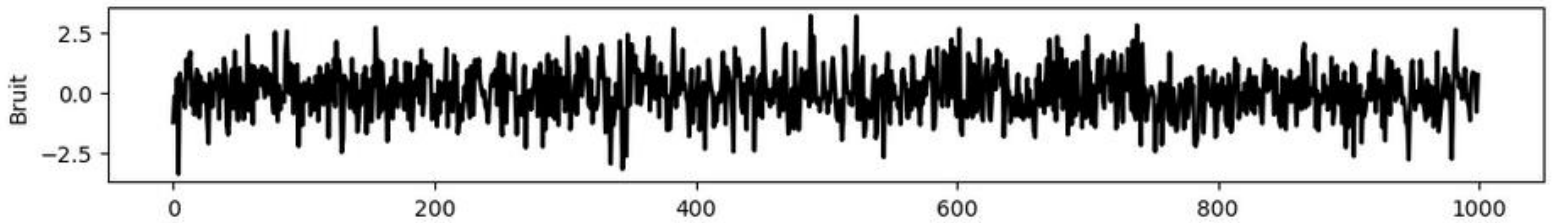




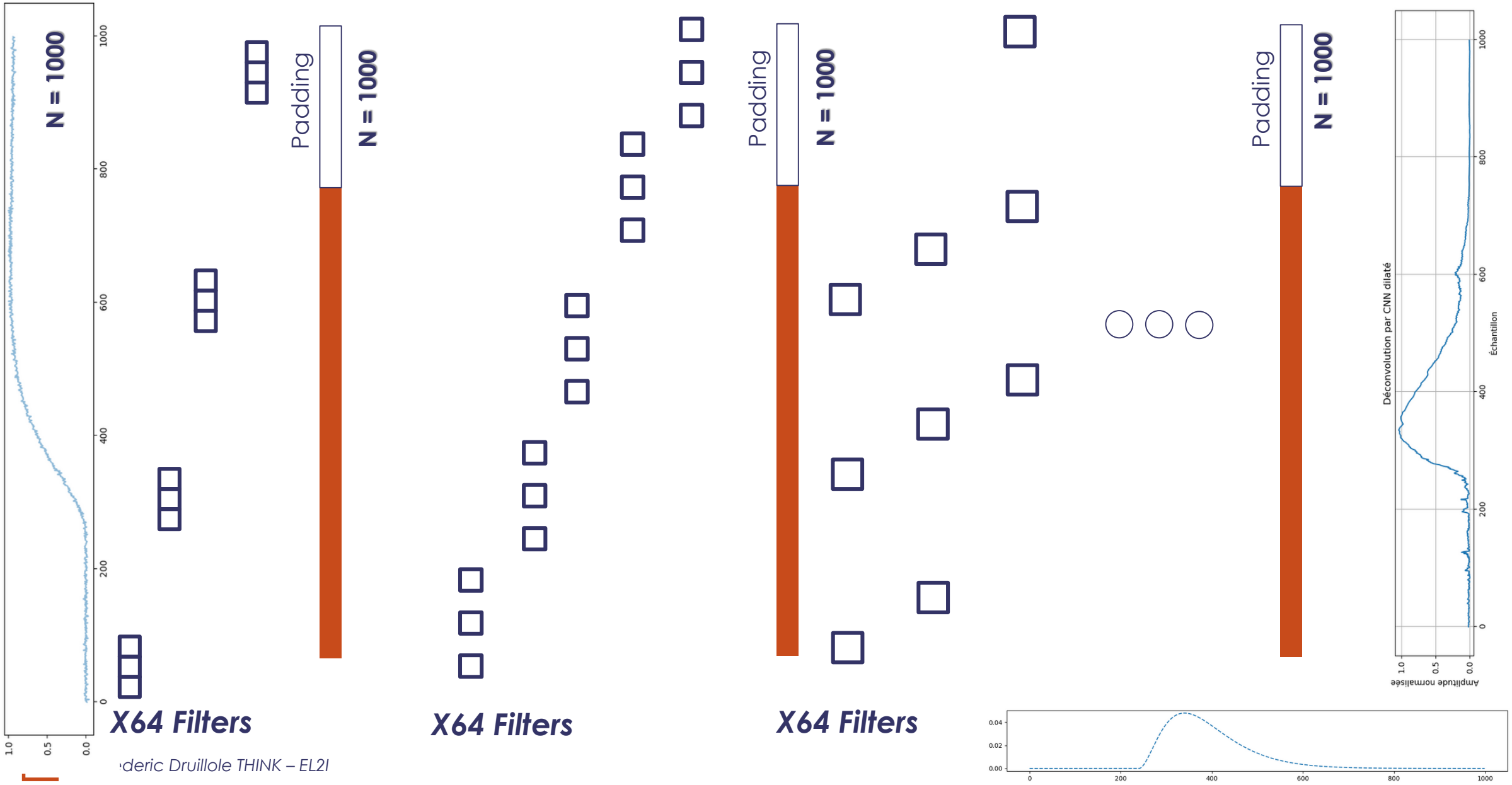








1D dilated CNN Model

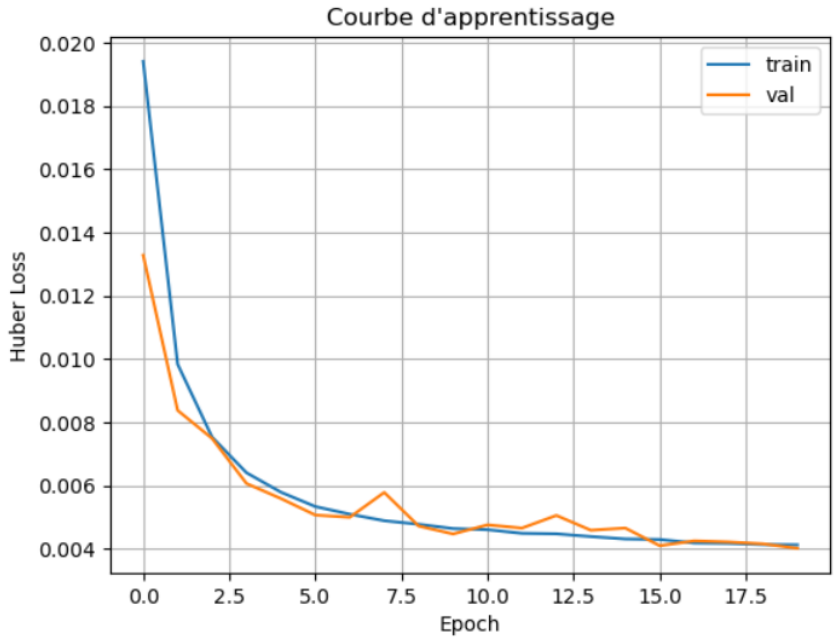


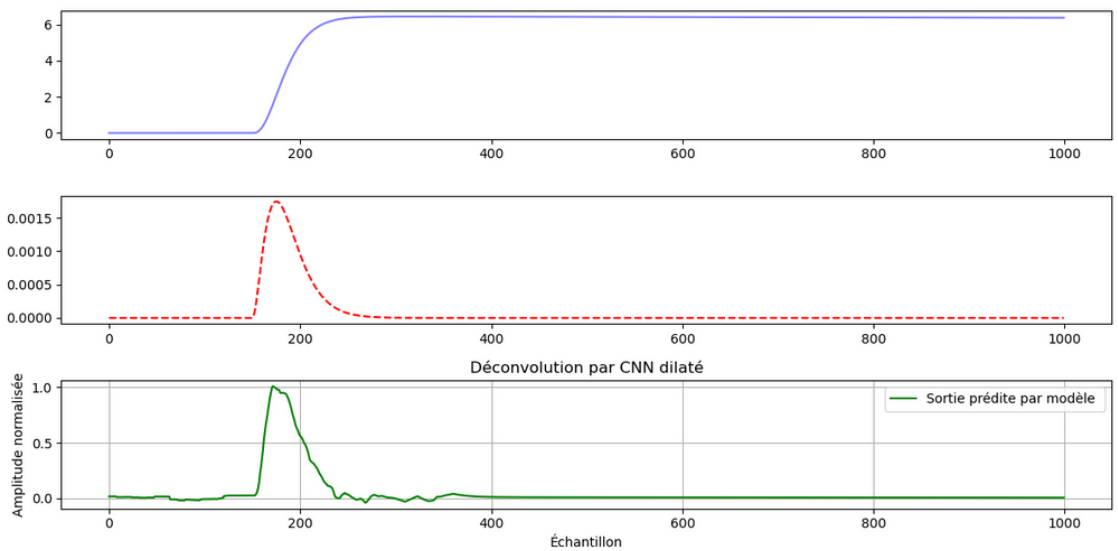
Déconvolution with IA: Modèle CNN 1D dilaté

```python
class DilatedConvBlock(layers.Layer):
    def __init__(self, filters, kernel_size, dilation_rate, **kwargs):
        super().__init__(**kwargs)
        self.conv = layers.Conv1D(
            filters=filters,
            kernel_size=kernel_size,
            dilation_rate=dilation_rate,
            padding="causal",
            activation="relu"
        )

    def call(self, inputs):
        return self.conv(inputs)

def build_custom_dilated_model(input_length=1000):
    inputs = Input(shape=(input_length, 1), name="signal_input")
    x = DilatedConvBlock(64, 3, dilation_rate=1, name="dilated_1")(inputs)
    x = DilatedConvBlock(64, 3, dilation_rate=2, name="dilated_2")(x)
    x = DilatedConvBlock(64, 3, dilation_rate=4, name="dilated_4")(x)
    x = DilatedConvBlock(64, 3, dilation_rate=8, name="dilated_8")(x)
    x = DilatedConvBlock(64, 3, dilation_rate=16, name="dilated_16")(x)
    x = DilatedConvBlock(64, 3, dilation_rate=32, name="dilated_32")(x)
    outputs = layers.Conv1D(1, kernel_size=1, activation="linear", name="final_output")(x)

    model = Model(inputs, outputs, name="DilatedCNNCustom")
    model.compile(optimizer="adam", loss=tf.keras.losses.Huber(delta=1.0))
    return model
```

Scaling; Amplitude = 1

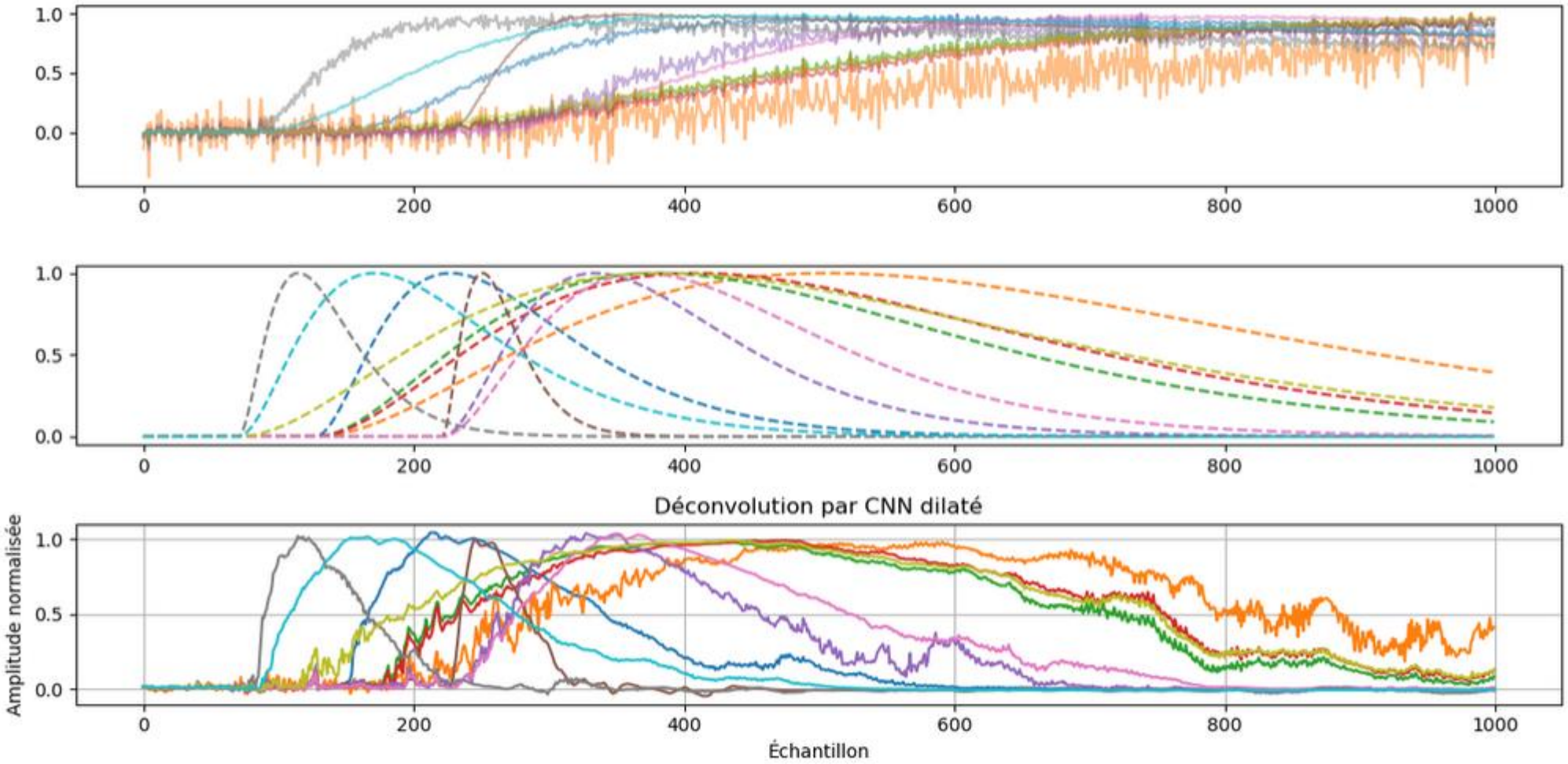
➔





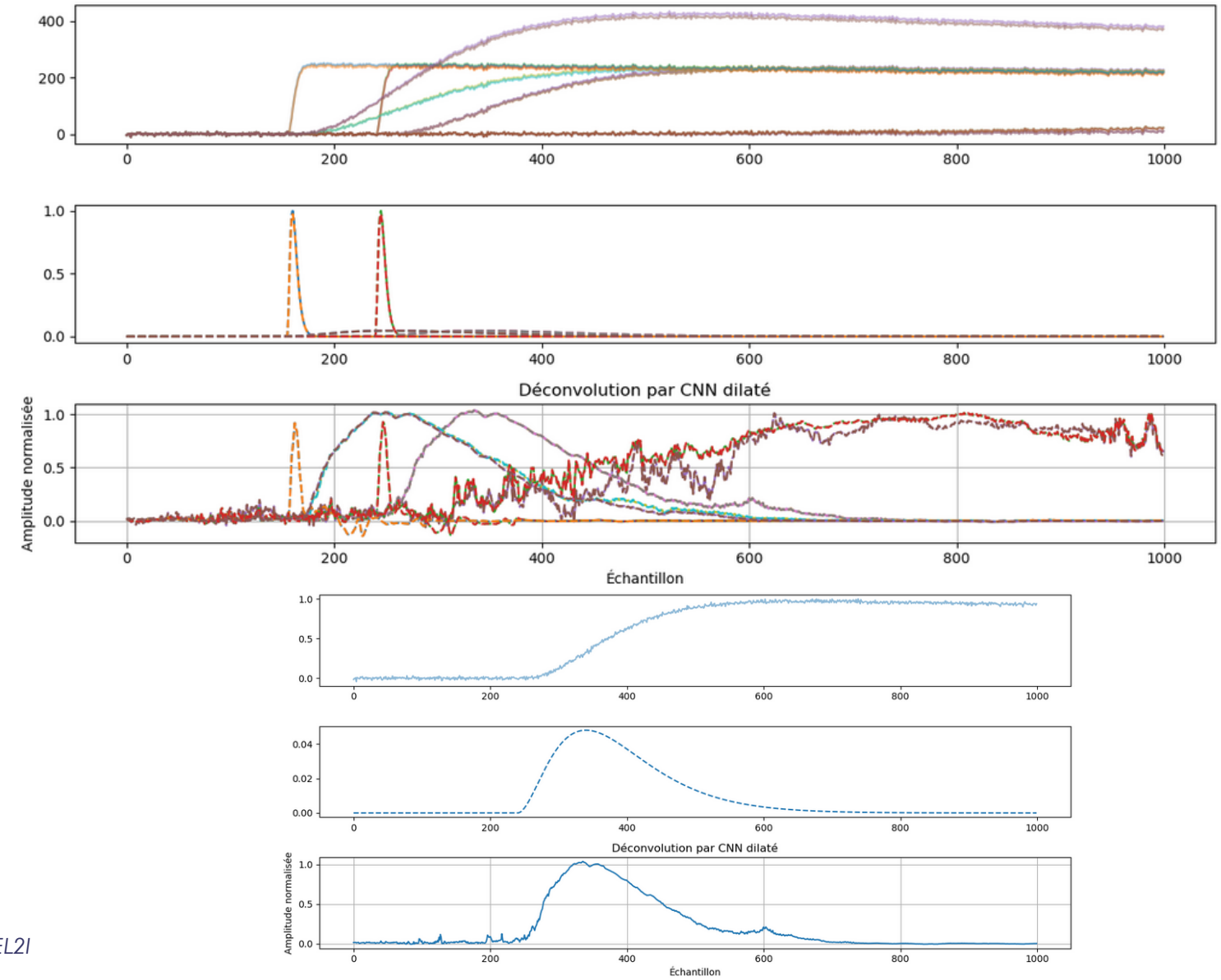
Déconvolution with IA: Modèle CNN 1D dilaté

Prédiction with known & Learning signal (Validation)



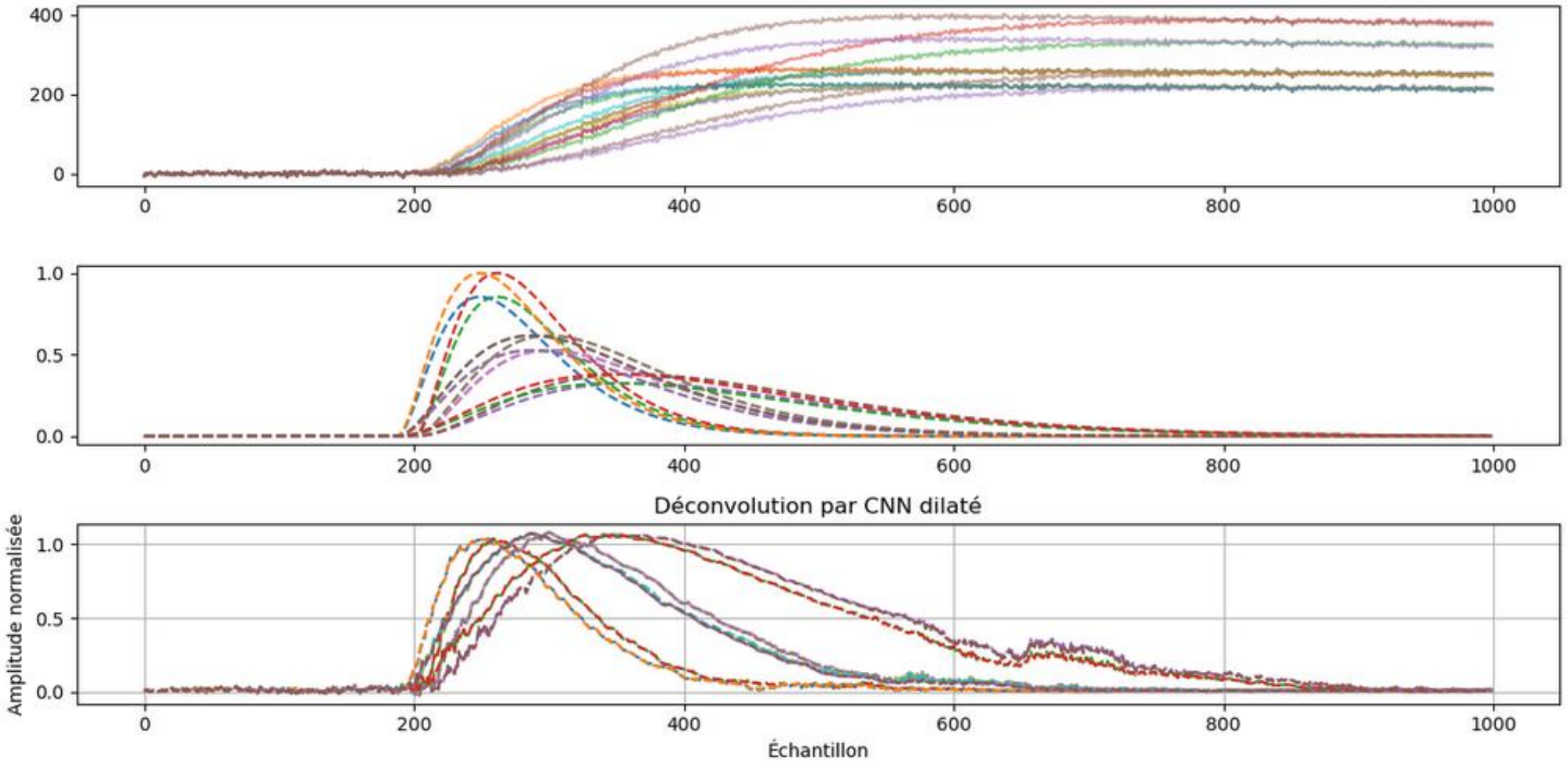
Déconvolution with IA: Modèle CNN 1D dilaté

Prediction with non-learning signal

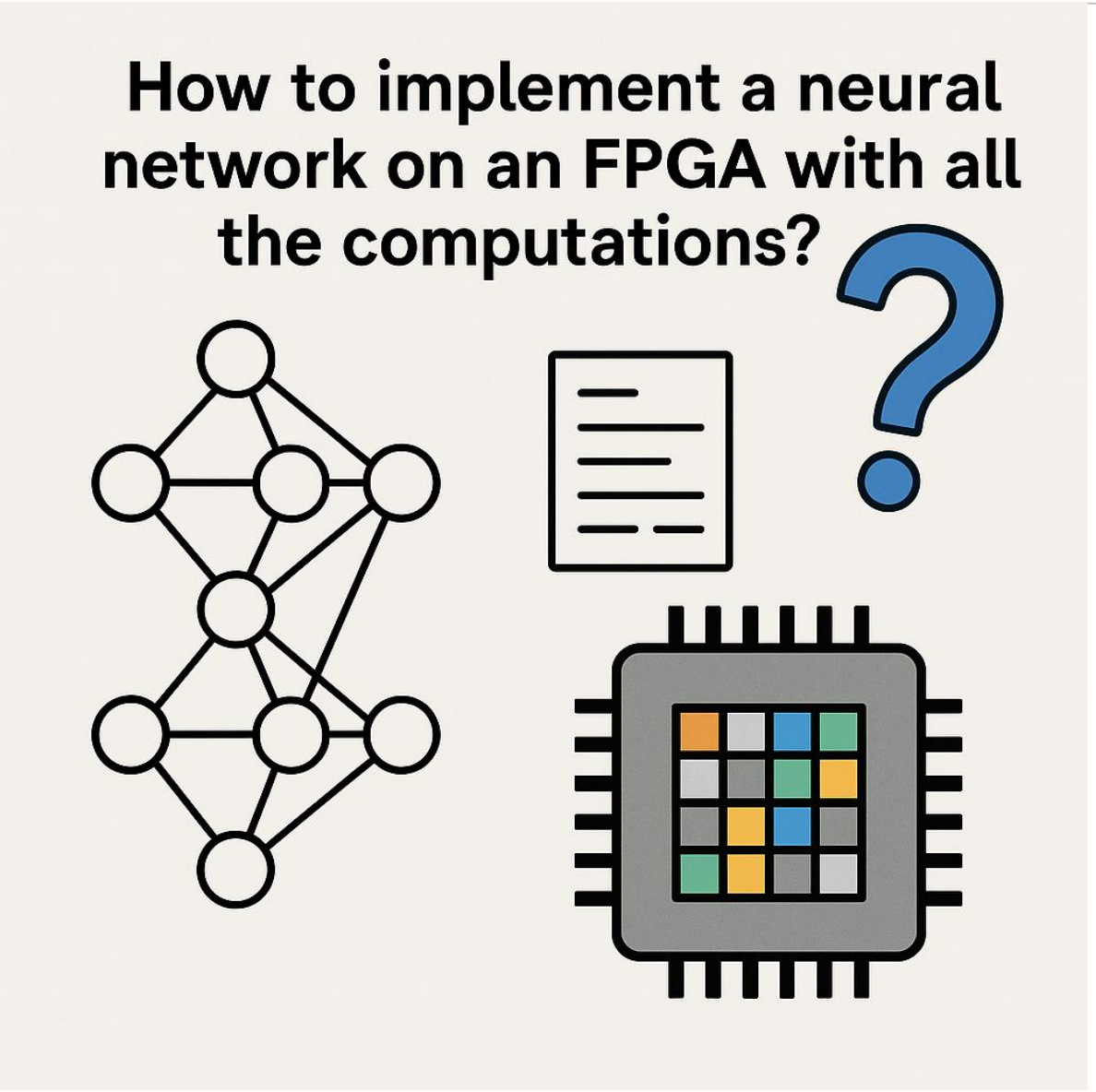


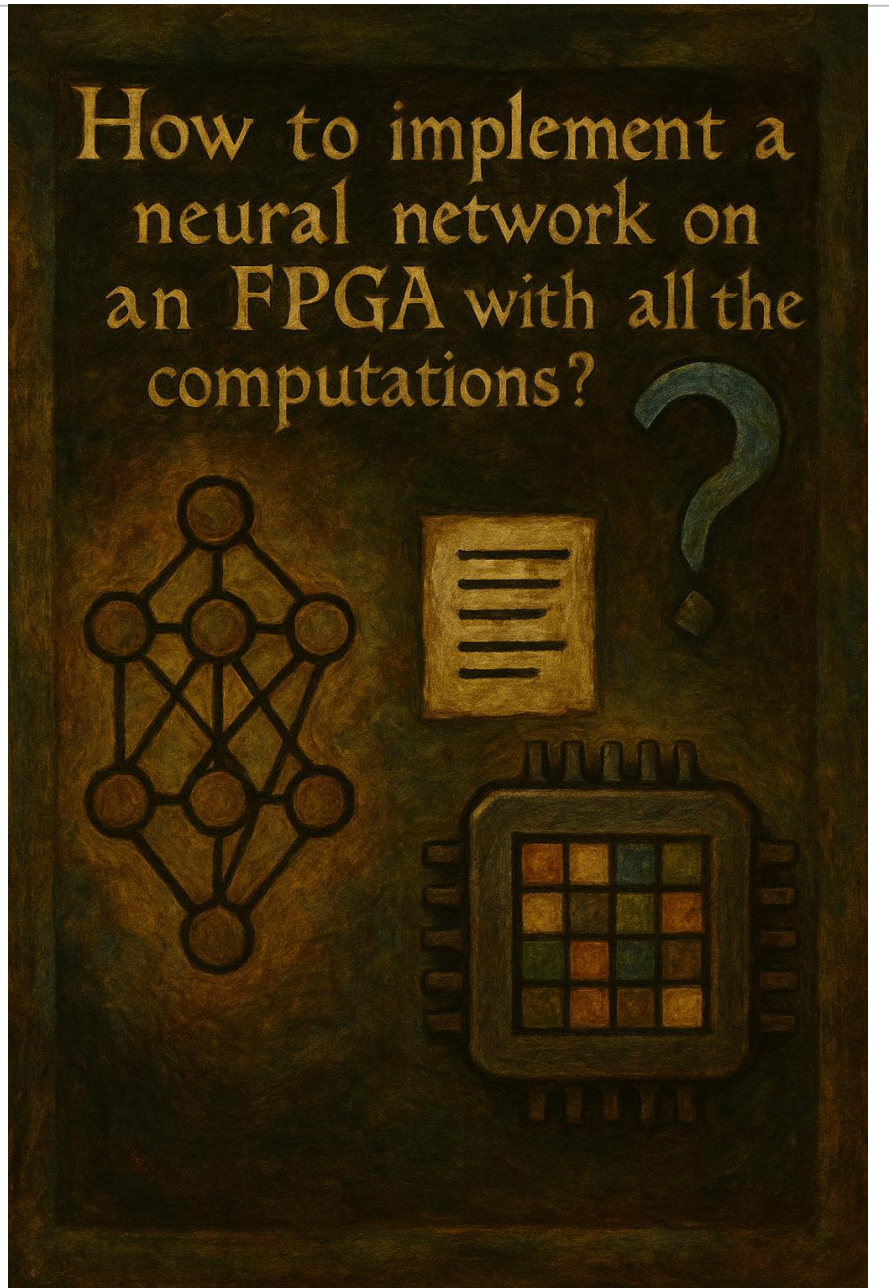
Déconvolution with IA: Modèle CNN 1D dilaté

Prediction with non-learning signal



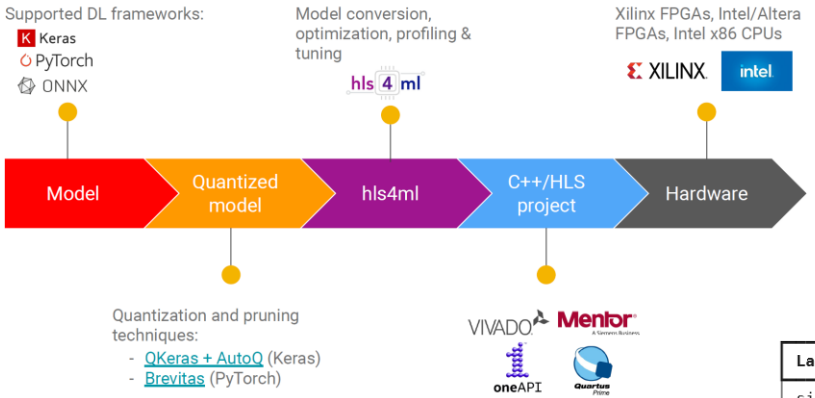
How to infer a neural network based model in an embedded component with limited ressources ?





Déconvolution with IA: Inférence sur FPGA (ARTY 100T) avec HLS4ML

hls4ml pipeline




```yaml
Backend: Vivado
ClockPeriod: 5
HLSConfig:
  LayerName:
    dilated_1:
      Precision:
        bias: fixed<16,6>
        result: fixed<16,6>
        weight: fixed<16,6>
      Trace: true
    dilated_16:
      Precision:
        bias: fixed<16,6>
        result: fixed<16,6>
        weight: fixed<16,6>
      Trace: true
    dilated_16_relu:
      Precision:
        result: fixed<16,6>
      Trace: true
    dilated_1_relu:
      Precision:
        result: fixed<16,6>
      Trace: true
  Model:
    BramFactor: 1000000000
    Precision: fixed<16,6>
    ReuseFactor: 1
    Strategy: Latency
    TraceOutput: false
IOType: io_parallel
InputData: null
KerasModel: !keras_model 'model_standard_deconv_nexys/model_nexys_hls4ml_prj_1/
OutputDir: model_standard_deconv_nexys/model_nexys_hls4ml_prj_1
OutputPredictions: null
Part: xc7a100tcsg324-1
ProjectName: myproject
Stamp: c463fEa6
```

```python
class DilatedConvBlock(layers.Layer):
    def __init__(self, filters, kernel_size, dilation_rate, **kwargs):
        super().__init__(**kwargs)
        self.conv = layers.Conv1D(
            filters=filters,
            kernel_size=kernel_size,
            dilation_rate=dilation_rate,
            padding="causal",
            activation="relu"
        )

    def call(self, inputs):
        return self.conv(inputs)
```

Layer (type)	Output Shape	Param #
signal_input (InputLayer)	(None, 1000, 1)	0
dilated_1_pruned (Conv1D)	(None, 1000, 18)	72
dilated_2_pruned (Conv1D)	(None, 1000, 64)	3,520
dilated_4_pruned (Conv1D)	(None, 1000, 64)	12,352
dilated_8_pruned (Conv1D)	(None, 1000, 64)	12,352
dilated_16_pruned (Conv1D)	(None, 1000, 64)	12,352
dilated_32_pruned (Conv1D)	(None, 1000, 64)	12,352
final_output (Conv1D)	(None, 1000, 1)	65

Total params: 53,065 (207.29 KB)

Trainable params: 53,065 (207.29 KB)

Non-trainable params: 0 (0.00 B)

```
    // ******************************************
    // NETWORK INSTANTIATION
    // ******************************************

    // hls-fpga-machine-learning insert layers

    layer2_t layer2_out[N_OUTPUTS_2*N_FILT_2];
    #pragma HLS ARRAY_PARTITION variable=layer2_out complete dim=0
    nnet::conv_1d_cl<input_t, layer2_t, config2>(signal_input, layer2_out, w2, b2); // dilated_1
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer2_t>(layer2_out, "dilated_1", N_OUTPUTS_2*N_FILT_2);
#endif

    layer3_t layer3_out[N_OUTPUTS_2*N_FILT_2];
    #pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
    nnet::relu<layer2_t, layer3_t, relu_config3>(layer2_out, layer3_out); // dilated_1_relu
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer3_t>(layer3_out, "dilated_1_relu", N_OUTPUTS_2*N_FILT_2);
#endif

    layer4_t layer4_out[N_OUTPUTS_4*N_FILT_4];
    #pragma HLS ARRAY_PARTITION variable=layer4_out complete dim=0
    nnet::conv_1d_cl<layer3_t, layer4_t, config4>(layer3_out, layer4_out, w4, b4); // dilated_2
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer4_t>(layer4_out, "dilated_2", N_OUTPUTS_4*N_FILT_4);
#endif

    layer5_t layer5_out[N_OUTPUTS_4*N_FILT_4];
    #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
    nnet::relu<layer4_t, layer5_t, relu_config5>(layer4_out, layer5_out); // dilated_2_relu
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer5_t>(layer5_out, "dilated_2_relu", N_OUTPUTS_4*N_FILT_4);
#endif

    layer6_t layer6_out[N_OUTPUTS_6*N_FILT_6];
    #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
    nnet::conv_1d_cl<layer5_t, layer6_t, config6>(layer5_out, layer6_out, w6, b6); // dilated_4
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer6_t>(layer6_out, "dilated_4", N_OUTPUTS_6*N_FILT_6);
#endif

    layer7_t layer7_out[N_OUTPUTS_6*N_FILT_6];
    #pragma HLS ARRAY_PARTITION variable=layer7_out complete dim=0
    nnet::relu<layer6_t, layer7_t, relu_config7>(layer6_out, layer7_out); // dilated_4_relu
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer7_t>(layer7_out, "dilated_4_relu", N_OUTPUTS_6*N_FILT_6);
#endif

    layer8_t layer8_out[N_OUTPUTS_8*N_FILT_8];
    #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
    nnet::conv_1d_cl<layer7_t, layer8_t, config8>(layer7_out, layer8_out, w8, b8); // dilated_8
#ifndef __SYNTHESIS__
    nnet::save_layer_output<layer8_t>(layer8_out, "dilated_8", N_OUTPUTS_8*N_FILT_8);
#endif
```

Déconvolution with IA: Inférence sur FPGA (ARTY 100T) avec HLS4ML

➔ **ERROR:** [HLS 200-70] Empty dataflow region in '*dilated_cnn_deconv_standard*'
(it may have been optimized away due to the absence of outputs).

➔ **ERROR:** [HLS 214-89] Function '*dilated_cnn_deconv_standard* ' cannot be synthesized in dataflow mode

➡ Absence de MLP dans le modèle

Type de structure		Implémentation RTL générée par HLS
Opérations arithmétiques (+, -, *, /)	➔	Mapping vers des **DSP48, LUT** ou adder trees selon la taille et le type
Boucles for, while	➔	Génération d'une **FSM** (machine d'état) pour séquencer les itérations
Tableaux (array)	➔	Mappage sur des **BRAM ou LUTRAM** selon les pragmas **(#pragma HLS RESOURCE, #pragma HLS ARRAY_PARTITION, etc.**)
Fonctions	➔	Générées comme **sous-modules RTL** ou inline selon **#pragma HLS INLINE**
Pipeline	➔	Génération de **registres** intermédiaires et d'une logique de contrôle selon **#pragma HLS PIPELINE**
Interfaces	➔	Génération d'un wrapper **AXI4, AXI-Lite, stream, BRAM** etc., selon **#pragma HLS INTERFACE**

Déconvolution with IA: Inférence sur FPGA (ARTY 100T) avec HLS4ML

- The convolutional layers (conv1, conv2, pool, etc.) are **chained together in a pipeline**.
- There is **no explicit final output** (i.e., no output vector written to an AXI interface or an output[] variable).
- HLS therefore sees a succession of functional calls, thus **activating implicit dataflow mode**.
- However, since no output is visible to the top module, it considers the **region "empty" or "unobservable**."
- Result: the error message "**Empty dataflow region**."

Solution:

- Add observable output (vector of size n=1000)

```
for (int i=0; i<LEN_OUT; i++)
    output[i] = final_output [i];
```

- Force HLS to not **activate implicit dataflow mode**

```
#pragma HLS INLINE off
#pragma HLS PIPELINE
```

The output CPP code of HLS4ML must be manipulated to account for the model's architecture, and the #pragma statements must also be modified.

➔ Write the model directly in VHDL.
➔ Understand how to optimize latency and resources in VHDL.

Déconvolution with IA: Conclusion

- We have a 1D CNN model that recreates a deconvolved transfer function signal for single-event signals.
- HLS4ML is used to optimize latency and resource usage for an FPGA target.

- HLS fails to synthesize the model due to the implicit DATAFLOW function.
- Revise the HLS code to add explicit output (not working yet).
- Understand how to optimize latency and resources directly in VHDL.

To Do List

- Create a 1D CNN entity directly in VHDL
- Find a deconvolution model for the 2-joint-event signals
- Test on NEXYS and ZCU104