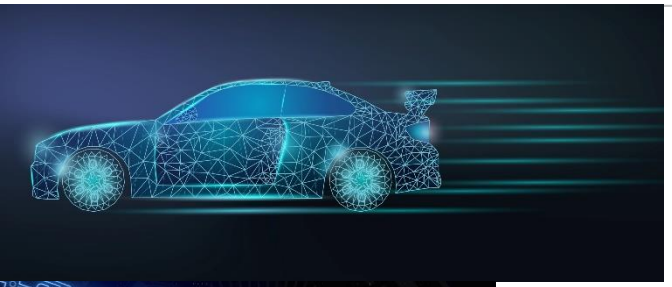
Le projet THINK:
Testing Hardware Instantiation of Neurals Kernels

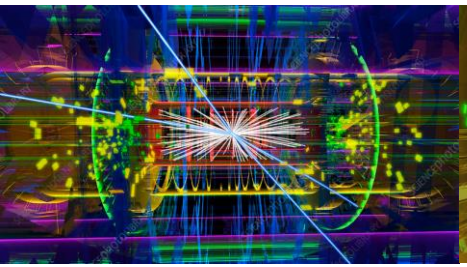
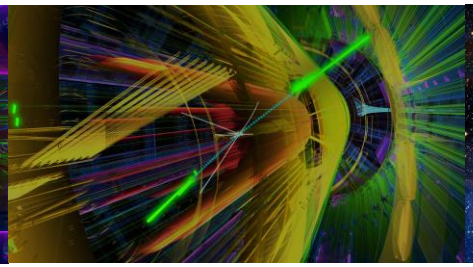
➔Nouveaux paradigmes
➔Technologies non matures (Recherche)
➔Applications innovantes en cours de développement

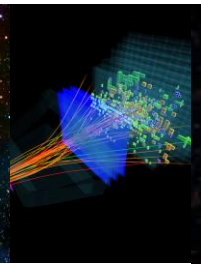
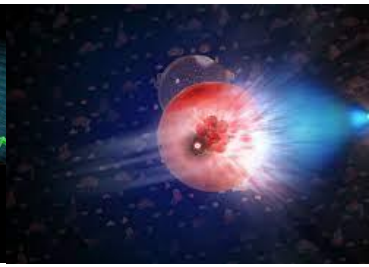


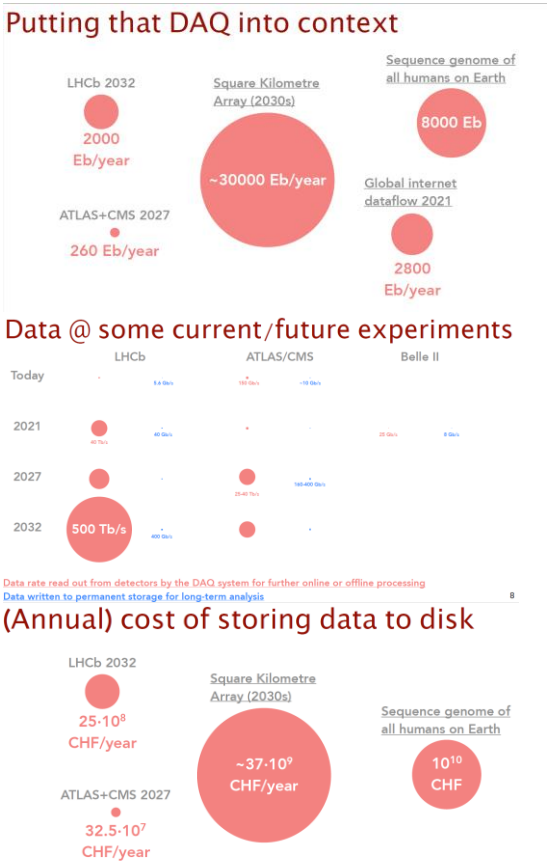
- Motivation
- La technologie
- Le statuts du projet
- Les travaux du SEA

J.-P. Cachemiche (CPPM)
V. Gligorov, O. Ledortz (LPNHE)
D. Etasse, J. Hommet (LPC Caen)
F. Bellachia, S. Lafrasse (LAPP
R. Bouet, F. Druillole, A. Rebii (CENBG)
F. Magniette, E. Sauvan (LLR)
G. Aad, Y. Boursier, T. Calvet, E. Fortin, E. Monnier (CPPM)
J. Frontera (CEA IRFU)

Les évolutions des détecteurs



- **Augmentation de la luminosité dans les détecteurs, la recherche d'évènements rares, sous l'effet du bruit de fond et des phénomènes d'empilement, complexifie la tâche des algorithmes de reconnaissances.**

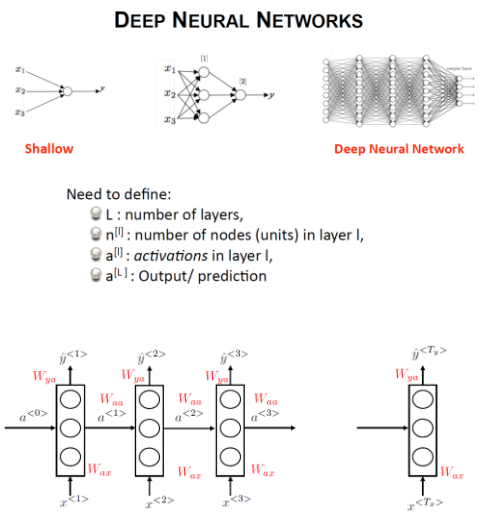
- **Les détecteurs requièrent plus d'intelligence pour filtrer les données**

- **Apparitions des architectures triggerless à base de co-processeurs accélérateurs et GPU**

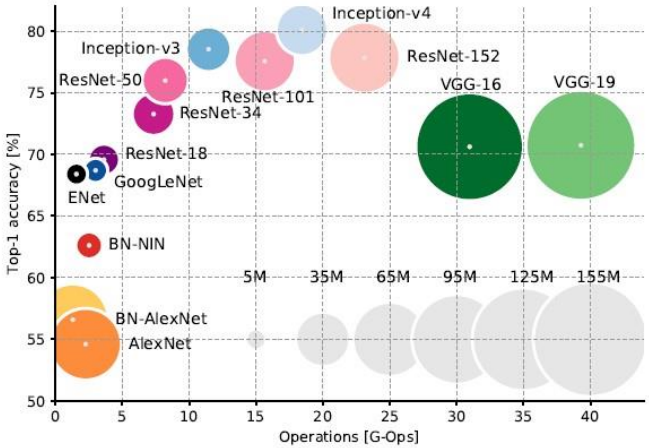
Enjeux : Réduction des données en temps réel en augmentant l'intelligence au plus prêt des sources de données :

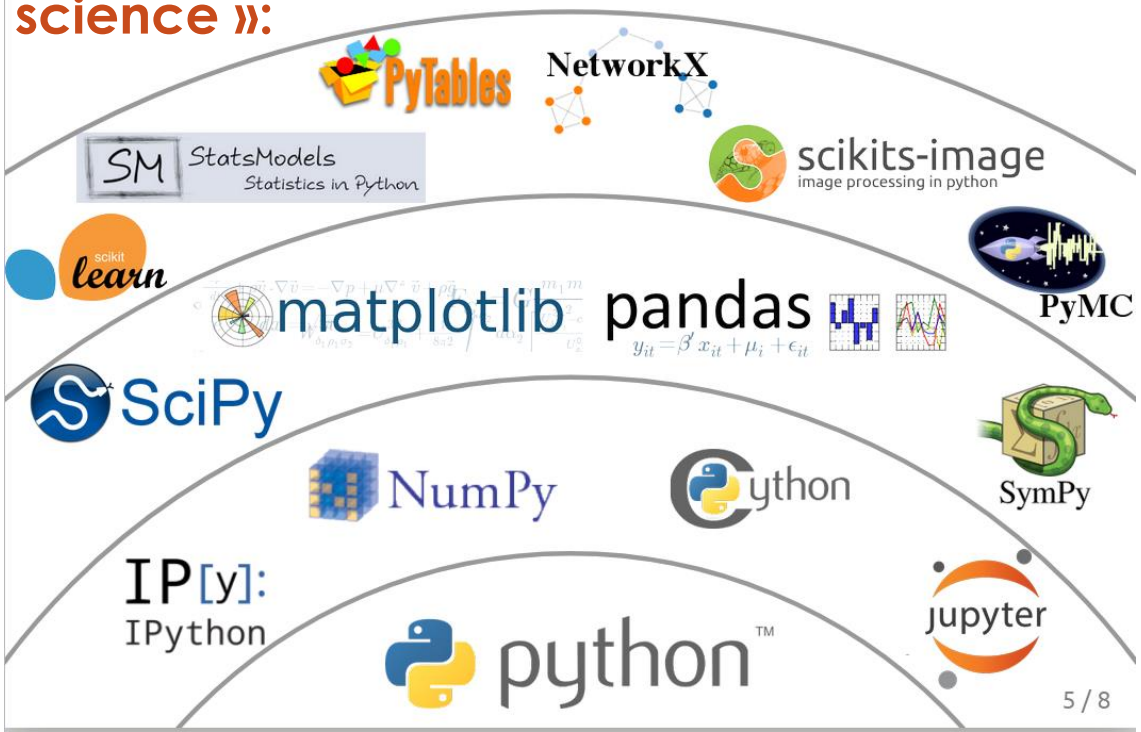
➔Embarqué les algorithmes dans les nœuds de decisions
➔Optimizer les traitements (classifications….DL)
➔Utiliser un mixte GPU, MPPA, FPGA

SAVOIR FAIRE ACTUEL



- Techniques neuronales et méthodologies connues de certains physiciens mais peu connu des ingénieurs
- Implémenté majoritairement dans les fermes de calcul mais pratiquement pas au niveau matériel prêt des détécteurs
- Implique la maîtrise de nombreux outils de la « data science »:





OBJECTIFS DE THINK

❑ PROPOSER UNE SOLUTION IA EMBARQUEE POUR LES FUTURS INSTRUMENTS

HIERARCHISATION DES PERFORMANCES ENTRE SOLUTIONS MATERIELS

- Un GPU est-il plus rapide qu'un FPGA ou qu'un MPPA pour une application donnée ?
- Rapport Investissement/ Maîtrise de la technologie
- Entrées/Sorties Disponibles

DETERMINER LES LIMITES DE CHAQUE SOLUTION

- Taille et type de réseaux implémentables
- Quelles applications pour quelles techniques ?

AVOIR UN MODELE DE COÛT DE CHAQUE SOLUTION

- Matériels, outillage, licences
- Cout d'apprentissage des outils
- Efficacité Solution/temps de developpement
- Facilité d'usage

RENDRE ACCESSIBLE LES OUTILS

➔ Determinant pour definir l'architecture en debut de programme

Objectifs du projet

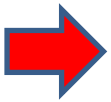
Le projet se déroule en 7 étapes réparties sur 36 mois:

- Une phase de formation pour les ingénieurs et techniciens chargés de la mise en œuvre technique des différentes implantations.

- Dans une seconde phase, on identifiera un minimum de deux applications typiques qui serviront de benchmark aux implémentations hardware.

- Une troisième phase consistera à définir une ou plusieurs structures de réseaux et à effectuer un apprentissage sur ces dernières. Cette phase peut se faire en simulation et ne dépend normalement pas de l'implémentation matérielle future.

- Quatre implémentation matérielles seront effectuées en parallèle sur respectivement FPGA, processeur MPPA, processeur neuromorphique et GPU.



- La phase suivante consistera à comparer les performances en terme de coût, de vitesse d'exécution, de consommation, etc … La facilité d'évolution algorithmique et par conséquent la facilité de mise en œuvre des outils de portage fera également l'objet d'une comparaison.

- Enfin le projet se terminera par une phase de diffusion du savoir éventuellement soutenue par plusieurs workshops, ainsi qu'une mise à disposition des outils ou des blocs utilisés dans un espace commun.

Applications en support du projet

- Le projet Amidex **OWEN** (Optimal Waveform recognition Electronic Node) qui consiste à développer un nouvel instrument pour traiter le signal venant d'un détecteur innovant, une TPC sphérique à haute pression. Son but est la recherche d'un phénomène rare tel que la détection directe de matière noire et l'observation de la décroissance double béta sans neutrino. Dans ce contexte, il s'agit de développer un système d'acquisition intégrant un algorithme de problème inverse basé sur les réseaux de neurones pour l'identification des formes d'ondes

- Le projet **RTA** (Real-Time Analysis) dans l'expérience LHCb qui consiste à traiter 40 Tb des donnees par seconde pour n'en garder que 80Gb/s pour une analyse plus profonde offline. Pour ce faire RTA doit à la fois utiliser efficacement les architectures modernes de calcul, et mettre en place des algorithmes avancés sur GPUs tels que les réseaux neurones.

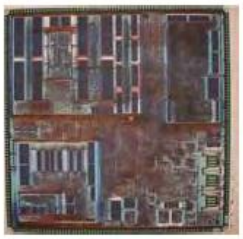
- Le projet Amidex **AIDAQ** qui consiste à implémenter des algorithmes de reconnaissance neuronale sur FPGA dans le calorimètre à argon liquide d'ATLAS pour réaliser les fonctions de trigger de premier niveau en environnement fortement bruité et avec des niveaux de pile-up variables.

- Le projet **HGCNN** qui consiste à développer des outils d'analyse neuronale pour les données des calorimètres à haute granularité (comme le futur calorimètre HGCal de CMS). Ces outils doivent être intégrés dans des FPGA et fournir des primitives de déclenchement avec des latences de l'ordre de la microseconde.

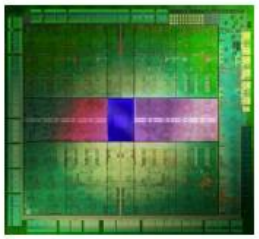
- Le projet **imXgam** d'imagerie médicale par tomographie. On se propose de débruiter les images par des techniques neuronales

DE NOUVEAUX COMPOSANTS PERFORMANTS

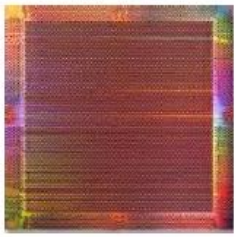
CPUs



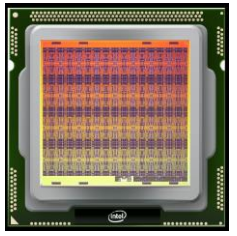
DSPs



GPUs

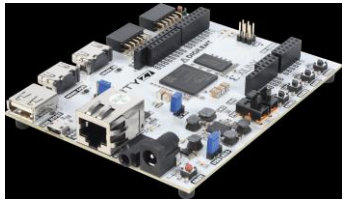


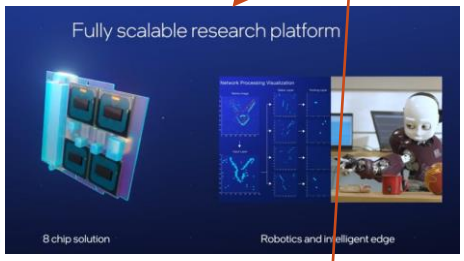
FPGAs



NMC

MPPA





DE NOUVEAUX LOGICIELS DE DEVELOPPEMENT

XILINX VITIS™ AI

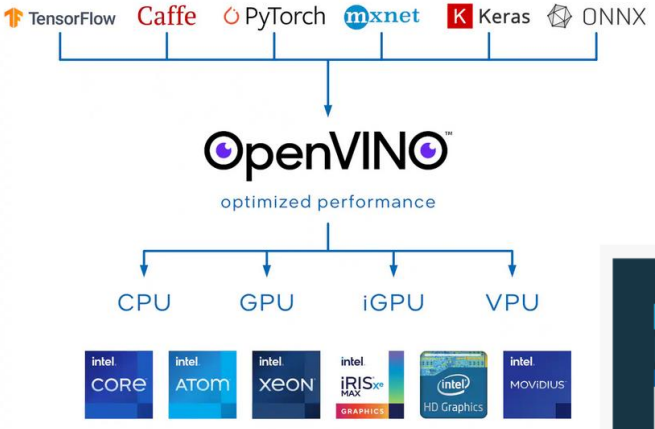
hls 4 ml

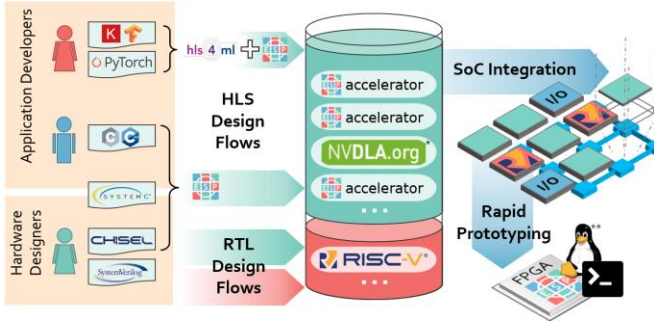
OpenCL

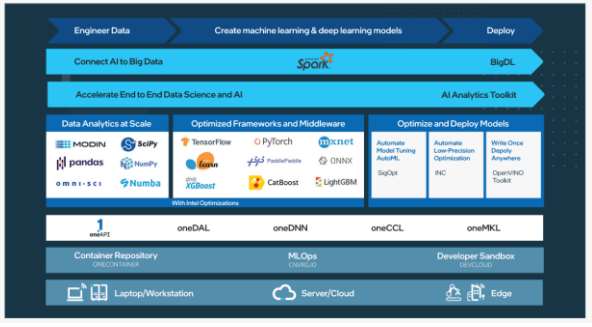
OpenMP

NVIDIA CUDA

Vivado™ HLS





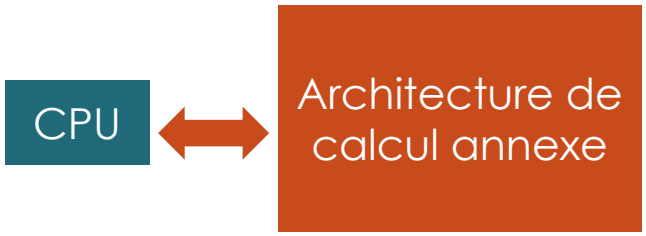


Sommaire

- **Processus d'intégration d'IA en FPGA**
- **Utilisation de l'outil HLS4ML (IA embarqué)**

- **Demonstration de Vitis AI (edge computing)**
  - Utilisation de reseaux entrainés Resnet50 et YOLO

IA embarqué : 2 solutions



Edge computing

-- Reste basé sur de la programmation logicielle

MMPA
GPU
FPGA
PU redesigned pour l'IA (TPU, KPU…)

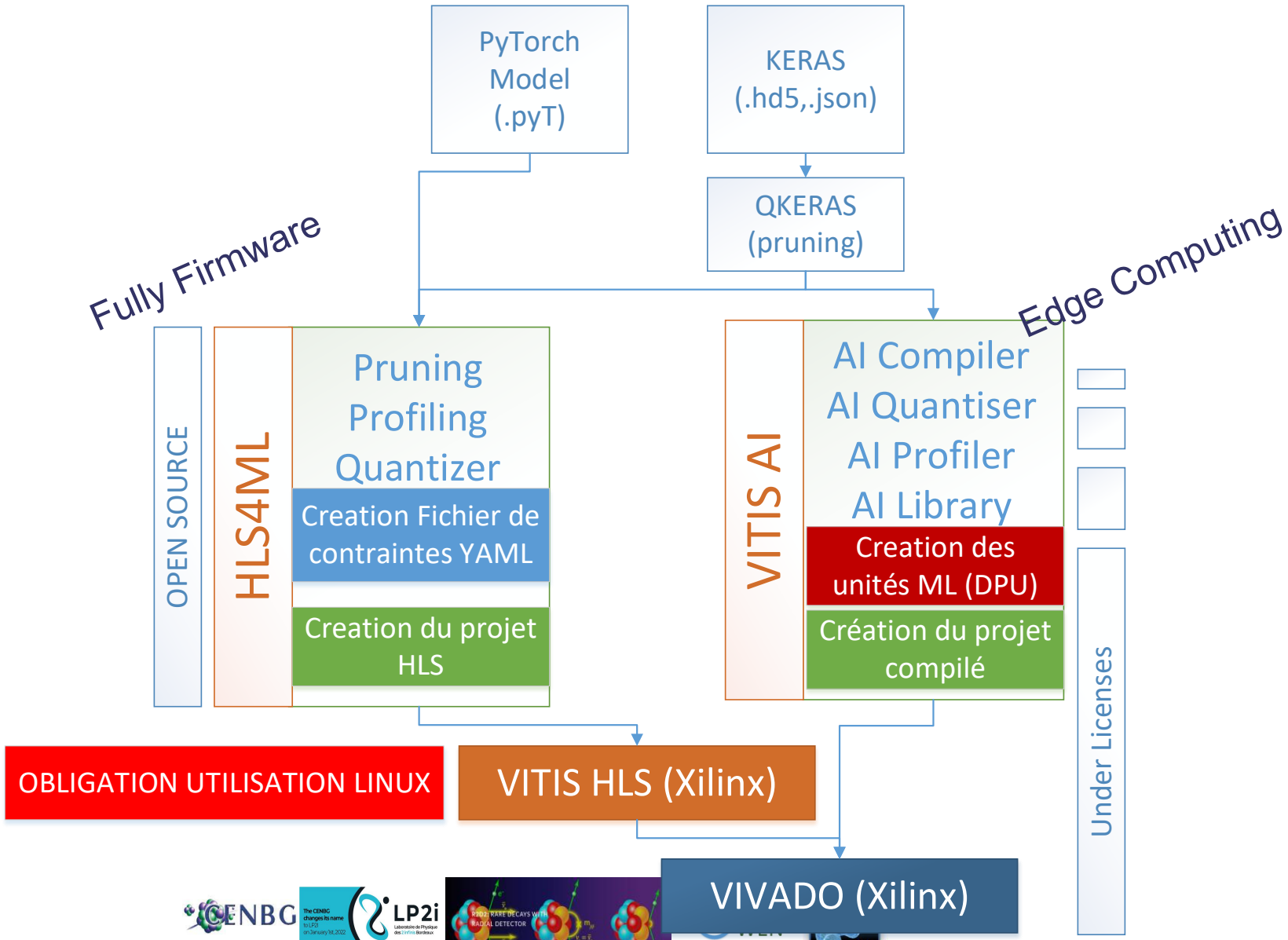
1 Pilote les développements actuels

-- basé sur des fonctions matérielles dédiées aux calculs sans logiciel (calcul matriciel)

ASIC
Circuit neuromorphique
FPGA

2 En cours de développement

2 façon d'intégrer l'IA en SE



PROCESSUS D'INTEGRATION D'UN RESEAU DE NEURONE PROFOND DANS UN FPGA

Mesures

Simulation

Bibliothèques des données d'apprentissage labellisées

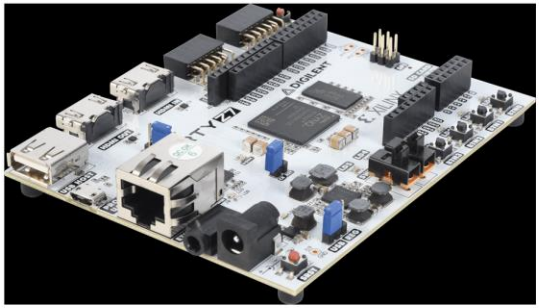
Analyse statistique des données (phase préparatoire)

Création du modèle :
- RNP
- RNC
- RNR …

Apprentissage: PyTorch Tensorflow-Keras

Poids

Modèle



Conversion des coefficients à virgules flottantes vers des nombres à virgules fixes

Redimensionnement des couches du réseau (prunning)

Reduction de l'empreinte mémoire et optimisation des calculs (Reuse factor)

Synthése et implementation du modèle dans un FPGA:
- « edge computing »
- Composants logiques (IP)

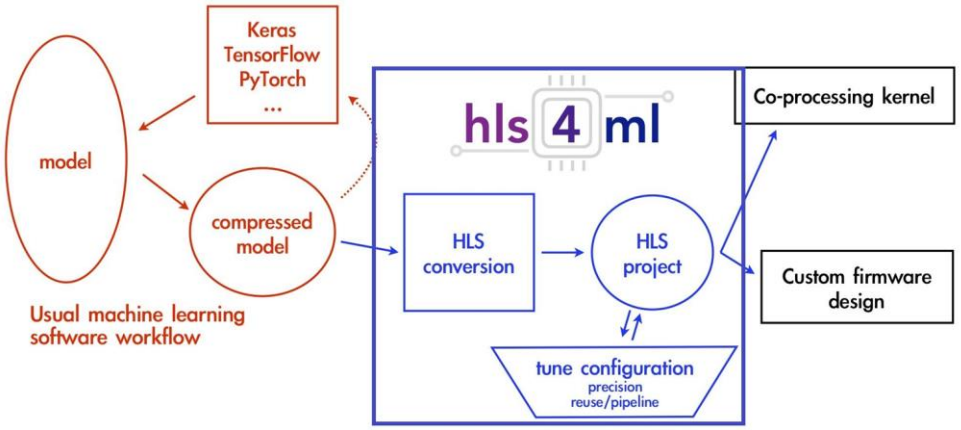
Validation avec jeux de données de test du composant

Validation sur données réelles du système (détecteur+ electronique)

Principe HLS4ML (fully firmware)

Bibliothèque python
Bibliothéque HLS (nnet.h)

Permet de passer d'un réseau PyTorch/Keras à un projet HLS



HLS4ML PROCESS
(Cern Open Source)

Keras/PyTorch Model in fixed point

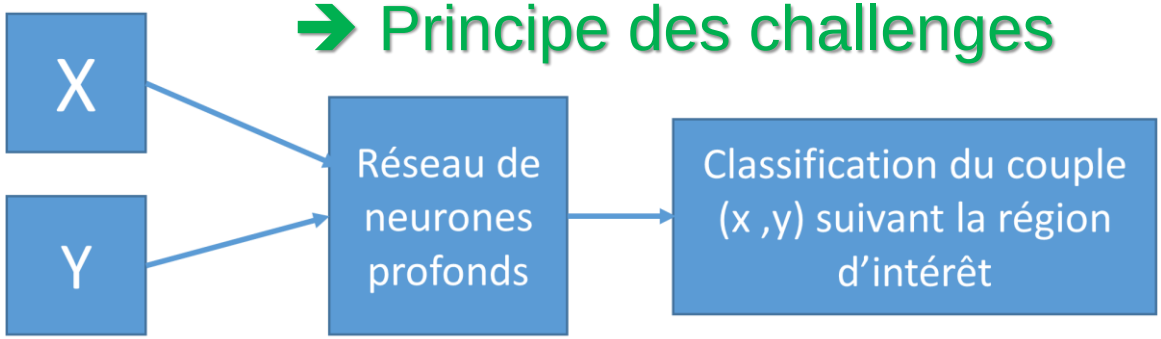
Convert to HLS project

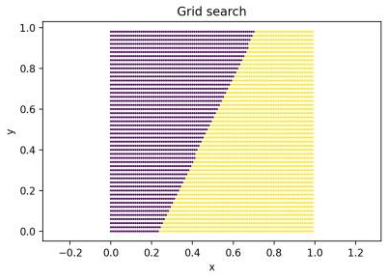
Optimize design reuse / pipeline/io

Generate a Vivado IP

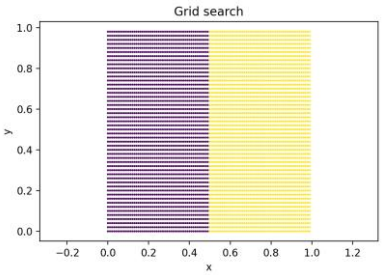
Add DNN/CNN IP in your Vivado project

Test de HLS4ML avec les réseaux « Challenge » fournis par Frédéric Magniette (LLR)

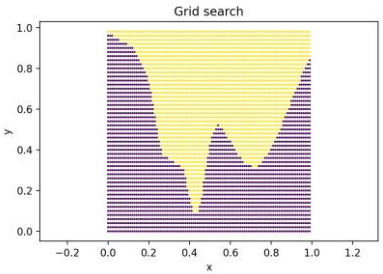




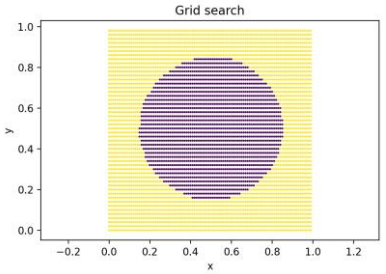
CH1



CH3

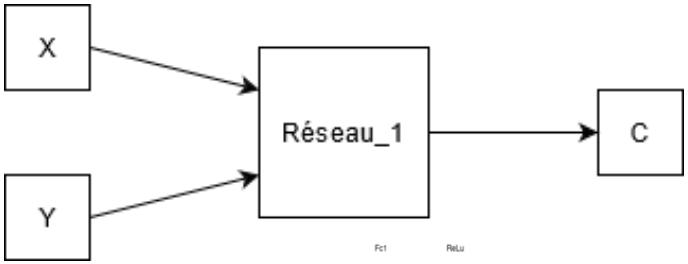


CH5

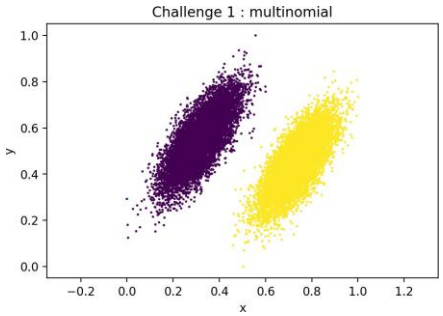


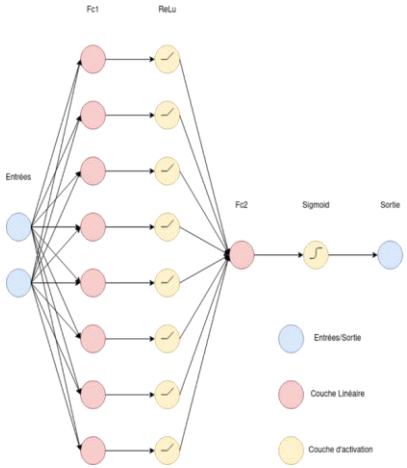
CH7

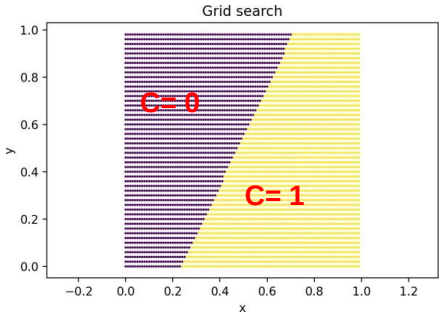
Un exemple de calcul: réseau de neurone peu profond



Principe: on fournit en entrée des coordonnées (x,y). Le réseau nous classifie les coordonnées selon la région d'appartenance. Ici, c'est une simple régression logistique avec une séparation de deux région linéaire.





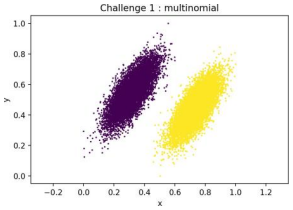


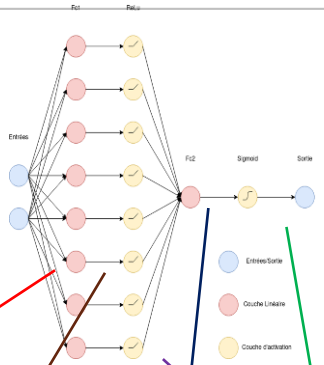
Il faut calculer

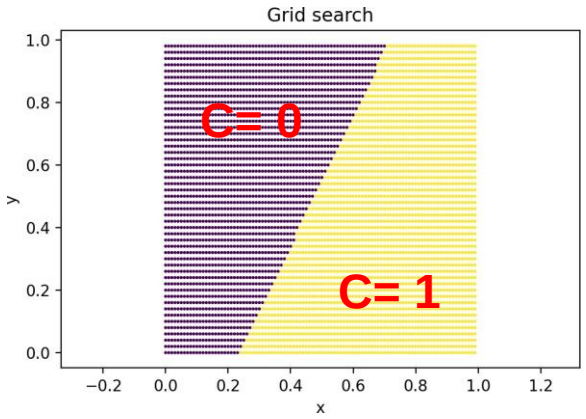
$$W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} \text{ et } B = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

$$s = \sigma(\mathbf{W}x + \mathbf{B})$$

Un exemple de calcul: réseau de neurone peu profond







X=0,2
Y=0,7

$$\begin{pmatrix} 1.7856 & -1.2345 \\ 2.4834 & -0.5797 \\ -0.3130 & 0.2658 \\ 0.3110 & -0.4116 \\ -0.4458 & 0.8010 \\ 3.4282 & -1.7166 \\ -0.6808 & 0.1631 \\ -0.5633 & 0.5694 \end{pmatrix} \text{X} \begin{pmatrix} 0.2 \\ 0.7 \end{pmatrix} + \begin{pmatrix} 0.0795 \\ -0.5202 \\ 0.4716 \\ -0.4349 \\ 1.3222 \\ -0.2305 \\ -0.3073 \\ 0.9498 \end{pmatrix} = \begin{pmatrix} -0.4275 \\ -0.4294 \\ 0.5951 \\ -0.6608 \\ 1.7938 \\ -0.7465 \\ -0.3293 \\ 1.2358 \end{pmatrix}$$

$$\begin{pmatrix} -0.4275 \\ -0.4294 \\ 0.5951 \\ -0.6608 \\ 1.7938 \\ -0.7465 \\ -0.3293 \\ 1.2358 \end{pmatrix} \xrightarrow{ReLu} \begin{pmatrix} 0 \\ 0 \\ 0.5951 \\ 0 \\ 1.7938 \\ 0 \\ 0 \\ 1.2358 \end{pmatrix}$$

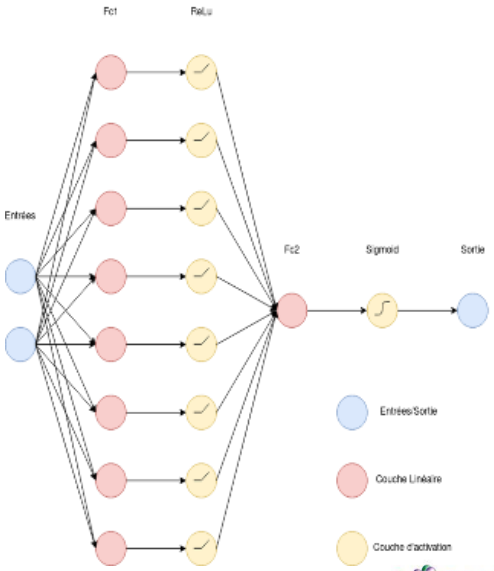
$$(-6.0539) \xrightarrow{sigmoïde} (0.0023) \sim 0$$

$$.0441\ 2.5193 - 0.6255\ 0.0535\ - 1.5080\ 3.7406\ 0.1052\ - 1.2330)\ \text{X} \begin{pmatrix} 0 \\ 0 \\ 0.5951 \\ 0 \\ 1.7938 \\ 0 \\ 0 \\ 1.2358 \end{pmatrix} + (-1.4529) = (-6.0539)$$

X=0,2
Y=0,7 Appartient à la zone **C= 0**

CH1

```
##############################################
#  creation reseau en KERAS                  #
##############################################
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l1
from tensorflow.keras.layers import Dense, Activation, Flatten
from tensorflow.keras import *


keras_model = Sequential()
keras_model.add(Input(shape=(2,)))
keras_model.add(Dense(8, name="fc1",kernel_initializer="lecun_uniform"))
keras_model.add(Activation(activations.relu, name="relu1") )
keras_model.add(Dense(1, name="fc2",kernel_initializer="lecun_uniform"))
keras_model.add(Activation(activations.sigmoid, name="sigmoid") )

keras_model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
print(keras_model.summary())
```

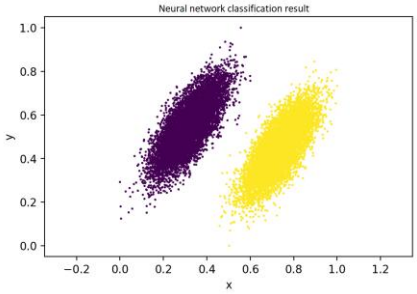


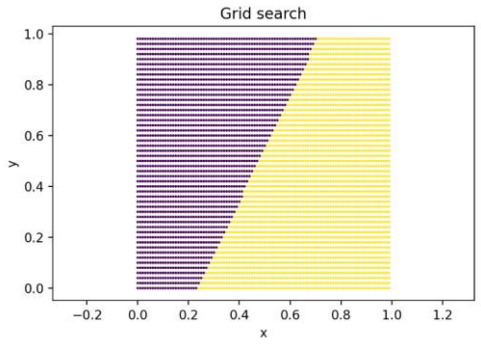
Input input_16	input: ? / output: (2,)	output (2,): ap_fixed<16,6>
Dense fc1	input: (2,) / output: (8,)	weight (2, 8): ap_fixed<16,6> / bias (8,): ap_fixed<16,6> / output (8,): ap_fixed<16,6>
Activation relu1	input: (8,) / output: (8,)	output (8,): ap_fixed<16,6>
Dense fc2	input: (8,) / output: (1,)	weight (8, 1): ap_fixed<16,6> / bias (1,): ap_fixed<16,6> / output (1,): ap_fixed<16,6>
Activation sigmoid	input: (1,) / output: (1,)	output (1,): ap_fixed<16,6>

Resultats CH1

```
import plotting
import hls4ml
config = hls4ml.utils.config_from_keras_model(keras_model, granularity='model')
print("-----------------------------------------------------------------")
print("configuration")
print("-----------------------------------------------------------------")
plotting.print_dict(config)
hls_model = hls4ml.converters.convert_from_keras_model(keras_model, \
                                hls_config=config,\
                                output_dir='model_ch1/hls4ml_prj',
                                part="xc7z020clg400-1")
hls4ml.utils.plot_model(hls_model,show_shapes=True, show_precision=True, to_file="./test")
hls_model.compile()
```

```
================================================================
== Utilization Estimates
================================================================
* Summary:
+-----------------+---------+-------+--------+-------+-----+
|       Name      | BRAM_18K| DSP48E|   FF   |  LUT  | URAM|
+-----------------+---------+-------+--------+-------+-----+
|DSP              |        -|      -|       -|      -|    -|
|Expression       |        -|      -|       0|      6|    -|
|FIFO             |        -|      -|       -|      -|    -|
|Instance         |        1|     21|    2290|    981|    -|
|Memory           |        -|      -|       -|      -|    -|
|Multiplexer      |        -|      -|       -|     36|    -|
|Register         |        -|      -|     321|      -|    -|
+-----------------+---------+-------+--------+-------+-----+
|Total            |        1|     21|    2611|   1023|    0|
+-----------------+---------+-------+--------+-------+-----+
|Available        |      280|    220|  106400|  53200|    0|
+-----------------+---------+-------+--------+-------+-----+
|Utilization (%)  |    ~0   |      9|       2|      1|    0|
+-----------------+---------+-------+--------+-------+-----+
```

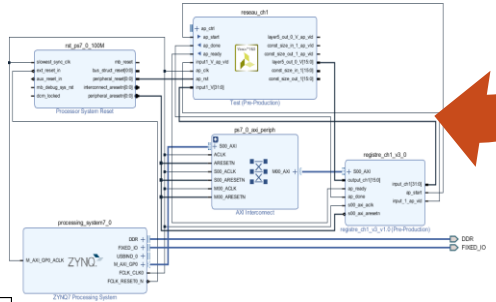




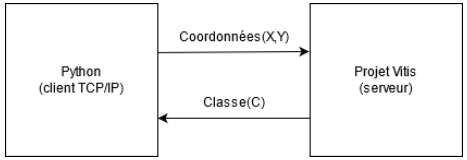
HLS

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
9	9	72.000 ns	72.000 ns	1	1	function

VIVADO/VITIS



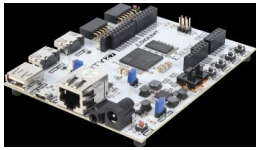
Test de l'IP en client/serveur entre le PC et la carte **ARTY Z7**



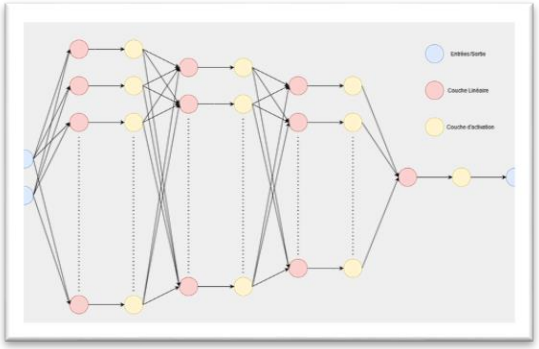
- 1 couche avec ReLu
- 1 couche avec Sigmoïde

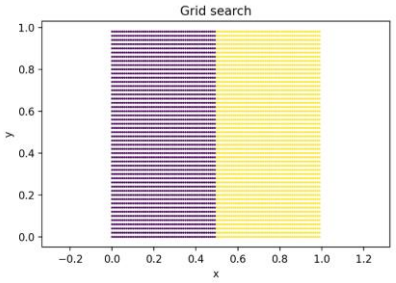
- Evaluation sur PC (python) : 100%
- Evaluation sur Arty Z7 : 100%

Resultats CH3



python



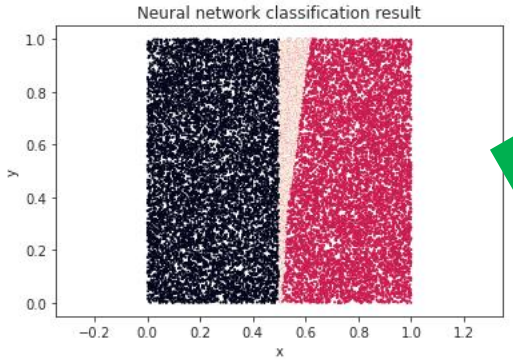


HLS

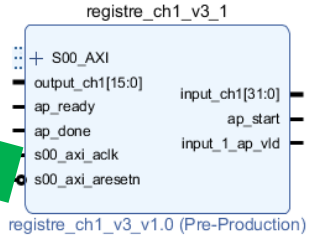
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2824	-
FIFO	0	-	3515	17932	-
Instance	17	26	14233	38297	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	6300	-
Register	-	-	700	-	-
Total	17	26	18448	65353	0
Available	280	220	106400	53200	0
Utilization (%)	6	11	17	122	0

Elagage / Pruning

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2	-
FIFO	-	-	-	-	-
Instance	15	27	27656	46180	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	101	-
Register	-	-	11269	-	-
Total	15	27	38925	46283	0
Available	280	220	106400	53200	0
Utilization (%)	5	12	36	86	0



Test de l'IP en client/serveur entre le PC et la carte **ARTY Z7**



- 1 couche avec ReLu
- 1 couche avec Signoïde

- Evaluation sur PC (python) : 100%
- Evaluation sur Arty Z7 : 96%

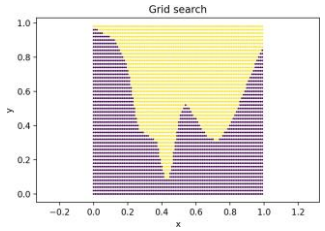
Test avec CH5 et CH7

- ➔ Réseaux RNP avec plus de couches
- ➔ Impossible à intégrer à ARTY Z7
- ➔ Utilisation de ZCU102

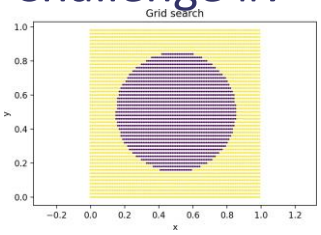
réseau	nombre de paramètres
CH5/CH7	313 016
CH3	51 263

```
model.add(QDense(500,
                 input_shape=(2,),
                 kernel_quantizer=quantized_bits(8,2,alpha=1),
                 bias_quantizer=quantized_bits(8,2,alpha=1),
                 name="fc1"))
model.add(QActivation("quantized_relu(8,2)",
                      name="relu1"))
```

Challenge #5



Challenge #7



pyTorch

```
class mlp(torch.nn.Module):
    def __init__(self,dims):
        super(mlp, self).__init__()
        self.relu=torch.nn.ReLU()
        self.sigmoid=torch.nn.Sigmoid()
        self.fc1=torch.nn.Linear(dims[0],dims[1])
        self.fc2=torch.nn.Linear(dims[1],dims[2])
        self.fc3=torch.nn.Linear(dims[2],dims[3])
        self.fc4=torch.nn.Linear(dims[3],dims[4])
        self.fc5=torch.nn.Linear(dims[4],1)

    def forward(self,input):
        x=self.relu(self.fc1(input))
        x=self.relu(self.fc2(x))
        x=self.relu(self.fc3(x))
        x=self.relu(self.fc4(x))
        x=self.sigmoid(self.fc5(x))
        return x
```

QKERAS

```
model = Sequen
model.add(QDe
          inp
          ker
          bia
          nam
model.add(QAct
          n
model.add(QDense(250,
                 kernel_quantizer=quantized_bits(8,2,alpha=1),
                 bias_quantizer=quantized_bits(8,2,alpha=1),
                 name="fc2"))
model.add(QActivation("quantized_relu(8,2)",
                      name="relu2"))
model.add(QDense(100,
                 kernel_quantizer=quantized_bits(8,2,alpha=1),
                 bias_quantizer=quantized_bits(8,2,alpha=1),
                 name="fc3"))
model.add(QActivation("quantized_relu(8,2)",
                      name="relu3"))
model.add(QDense(50,
                 kernel_quantizer=quantized_bits(8,2,alpha=1),
                 bias_quantizer=quantized_bits(8,2,alpha=1),
                 name="fc4"))
model.add(QActivation("quantized_relu(8,2)",
                      name="relu4"))
model.add(QDense(1,
                 kernel_quantizer=quantized_bits(8,2,alpha=1),
                 bias_quantizer=quantized_bits(8,2,alpha=1),
                 name="fc5"))
model.add(Activation("sigmoid", name="sigmoid"))
```

Elagage et Reuse Factor

HLS

Rf = 1000

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	0	-	9005	50428	-
Instance	84	157	132732	174187	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	84	157	141747	224683	0
Available	1824	2520	548160	274080	0
Utilization (%)	4	6	25	81	0

a) Ressource pour CH5

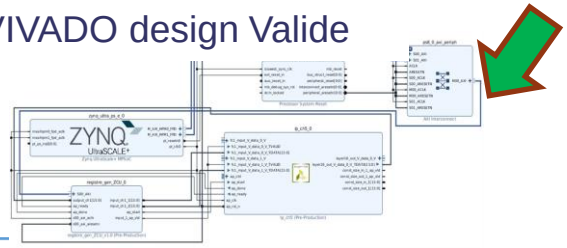
Rf = 10000

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	0	-	9005	50428	-
Instance	96	151	249721	67433	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	96	151	339832	17929	0
Available	1824	2520	548160	274080	0
Utilization (%)	5	~0	24	79	0

b) Ressource pour CH7

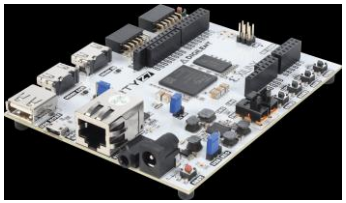
Qkeras ➔ Entraine le réseau avec des coefficient en virgules fixes avant HLS

VIVADO design Valide



☹ **VIVADO crashe lors de la synthèse de CH5 et CH7 ➔ ??**

CH7

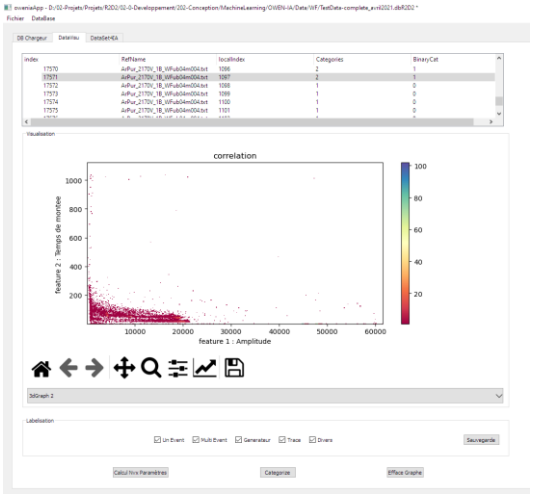


```
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  | 5.00 ns | 9.410 ns |   0.62 ns  |
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-------+-------+----------+
    |  Latency (cycles) |  Latency (absolute) |    Interval   | Pipeline |
    |   min   |   max   |    min   |    max   |  min  |  max  |   Type   |
    +---------+---------+----------+----------+-------+-------+----------+
    |    15020|    15021| 0.141 ms | 0.141 ms |  15006|  15007| dataflow |
    +---------+---------+----------+----------+-------+-------+----------+

    ================================================================
    == Utilization Estimates
    ================================================================
    * Summary:
    +-----------------+---------+-------+--------+--------+-----+
    |       Name      | BRAM_18K| DSP48E|    FF  |   LUT  | URAM|
    +-----------------+---------+-------+--------+--------+-----+
    |DSP              |        -|      -|       -|       -|    -|
    |Expression       |        -|      -|       0|      34|    -|
    |FIFO             |        0|      -|    9005|   50428|    -|
    |Instance         |      199|     15|  169262|  174887|    -|
    |Memory           |        -|      -|       -|       -|    -|
    |Multiplexer      |        -|      -|       -|      36|    -|
    |Register         |        -|      -|       6|       -|    -|
    +-----------------+---------+-------+--------+--------+-----+
    |Total            |      199|     15|  178273|  225385|    0|
    +-----------------+---------+-------+--------+--------+-----+
    |Available        |      280|    220|  106400|   53200|    0|
    +-----------------+---------+-------+--------+--------+-----+
    |Utilization (%)  |       71|      6|     167|     423|    0|
    +-----------------+---------+-------+--------+--------+-----+
```

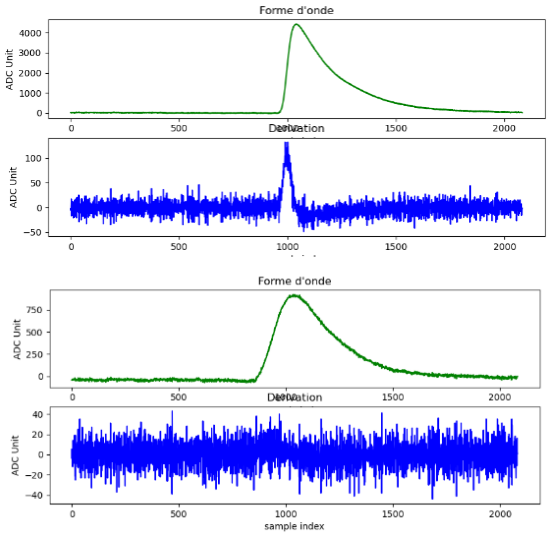
```
+ Timing:
    * Summary:
    +--------+---------+----------+------------+
    |  Clock |  Target | Estimated| Uncertainty|
    +--------+---------+----------+------------+
    |ap_clk  | 5.00 ns | 4.369 ns |   0.62 ns  |
    +--------+---------+----------+------------+

+ Latency:
    * Summary:
    +---------+---------+----------+----------+-------+-------+----------+
    |  Latency (cycles) |  Latency (absolute) |    Interval   | Pipeline |
    |   min   |   max   |    min   |    max   |  min  |  max  |   Type   |
    +---------+---------+----------+----------+-------+-------+----------+
    |    13008|    13009| 65.040 us | 65.045 us |  13003|  13004| dataflow |
    +---------+---------+----------+----------+-------+-------+----------+

    ================================================================
    == Utilization Estimates
    ================================================================
    * Summary:
    +-----------------+---------+-------+--------+--------+-----+
    |       Name      | BRAM_18K| DSP48E|    FF  |   LUT  | URAM|
    +-----------------+---------+-------+--------+--------+-----+
    |DSP              |        -|      -|       -|       -|    -|
    |Expression       |        -|      -|       0|      32|    -|
    |FIFO             |        0|      -|    9005|   50428|    -|
    |Instance         |      150|     15|  124975|  167435|    -|
    |Memory           |        -|      -|       -|       -|    -|
    |Multiplexer      |        -|      -|       -|      36|    -|
    |Register         |        -|      -|       6|       -|    -|
    +-----------------+---------+-------+--------+--------+-----+
    |Total            |      150|     15|  133986|  217931|    0|
    +-----------------+---------+-------+--------+--------+-----+
    |Available        |     1824|   2520|  548160|  274080|    0|
    +-----------------+---------+-------+--------+--------+-----+
    |Utilization (%)  |        8|     ~0|      24|      79|    0|
    +-----------------+---------+-------+--------+--------+-----+
```

HLS4ML pour le réseau R2D2

OWEN-IA:
Un logiciel de labellisation et de création d'une base de données des signaux



- ➔ préamplificateur de charge (intégration)
- ➔ la dérivée
- ➔ l'évènement physique (détécteur).

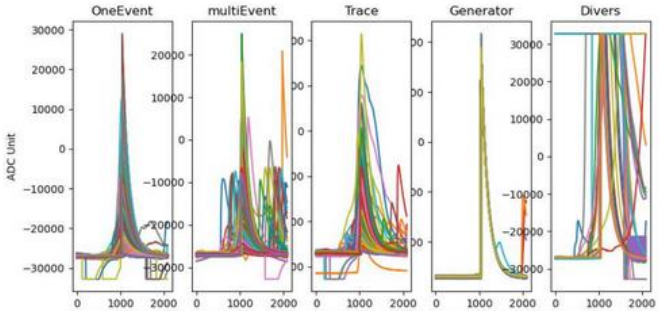


On calcule pour le signal et sa dérivée :

- Temps de montée
- Amplitude Pic à Pic
- Amplitude maximum
- Largeur à 50% d'amplitude maximum
- Largeur à 20% d'amplitude maximum
- Largeur à 80 % d'amplitude maximum
- Valeur de l'intégrale du signal
- Ligne de base moyenne
- Ligne de base écart type

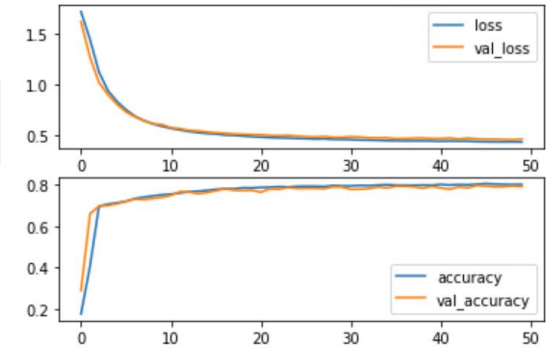
Classification de signaux selon 5 catégories

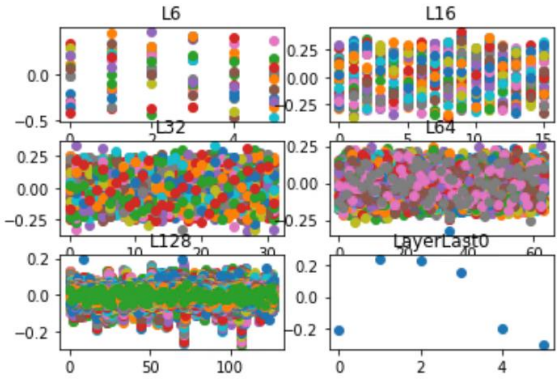
- **1 event**
- **multiEvent**
- **Trace**
- **Generateur**
- **Divers**

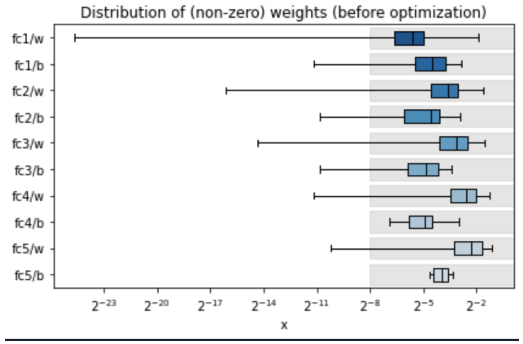


DNN pour R2D2

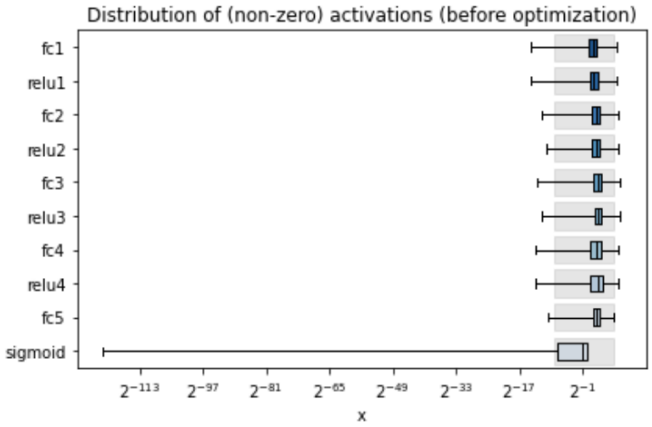
Input input_18	input: ? / output: (2083,)	output (2083,): ap_fixed<16,8>
Dense fc1	input: (2083,) / output: (128,)	weight (2083, 128): ap_fixed<16,8> / bias (128,): ap_fixed<16,8> / output (128,): ap_fixed<16,8>
Activation relu1	input: (128,) / output: (128,)	output (128,): ap_fixed<16,8>
Dense fc2	input: (128,) / output: (64,)	weight (128, 64): ap_fixed<16,8> / bias (64,): ap_fixed<16,8> / output (64,): ap_fixed<16,8>
Activation relu2	input: (64,) / output: (64,)	output (64,): ap_fixed<16,8>
Dense fc3	input: (64,) / output: (32,)	weight (64, 32): ap_fixed<16,8> / bias (32,): ap_fixed<16,8> / output (32,): ap_fixed<16,8>
Activation relu3	input: (32,) / output: (32,)	output (32,): ap_fixed<16,8>
Dense fc4	input: (32,) / output: (16,)	weight (32, 16): ap_fixed<16,8> / bias (16,): ap_fixed<16,8> / output (16,): ap_fixed<16,8>
Activation relu4	input: (16,) / output: (16,)	output (16,): ap_fixed<16,8>
Dense fc5	input: (16,) / output: (6,)	weight (16, 6): ap_fixed<16,8> / bias (6,): ap_fixed<16,8> / output (6,): ap_fixed<16,8>
Activation sigmoid	input: (6,) / output: (6,)	output (6,): ap_fixed<16,8>







Le taux de bonnes réponses est **0.80**



Problème de precision sur les poids du réseau pour l'implementation dans un FPGA

Etude de la conversion en FP<16,6>

[9.9999952e-01 7.8708261e-02 9.9373168e-01 2.3431629e-02 4.9352646e-04 2.3432046e-02] ➔ [1. 0. 0. 0. 0. 0.]

[9.9960995e-01 9.7909123e-01 9.9998963e-01 7.0114225e-02 4.3409372e-07 1.2318639e-04] ➔ [0. 0. 1. 0. 0. 0.]

[9.996357e-01 9.892076e-01 9.991980e-01 1.000000e+00 9.983493e-01 9.450185e-08] ➔ [0. 0. 0. 1. 0. 0.]

[9.9931806e-01 9.6154308e-01 6.2342167e-01 3.2636523e-04 4.7685789e-05 5.0149560e-02] ➔ [1. 0. 0. 0. 0. 0.]

[0.9999528 0.34945738 0.9880831 0.08217603 0.01645651 0.02990997] ➔ [1. 0. 0. 0. 0. 0.]

[9.9996746e-01 9.9999130e-01 9.9988866e-01 1.8101977e-06 1.3024892e-10 7.6915210e-05] ➔ [0. 1. 0. 0. 0. 0.]

Conversion en virgule fixe : <16,6>

Keras layer 'fc1', first sample:[10.888223 3.535251 -1.1810297 -3.8457754 0.44802344 2.7766297 -4.0835686 -0.17144588 -4.477542 2.7125385 -3.9044225 -3.6908937 1.9270804 -1.6103764 -0.39574364 -5.2935867

hls4ml layer 'fc1', first sample:[6.67578125 -0.67578125 -5.2734375 -8.07421875 -3.72265625 -1.3125 -8.21484375 -4.296875 -8.6171875 -1.47265625 -8.0546875 -7.828125 -2.28515625 -5.609375 -4.5390625

Conversion en HLS

Keras Accuracy: 0.791602266872746
hls4ml Accuracy: 0.6620298815043791

[**0.99609375** 0.1796875 0.80078125 0.1640625 0.02734375 0.23828125] ==> [1. 0. 0. 0. 0. 0.]

[0.99609375 0.95703125 1. 0.05859375 0. 0.00390625] ==> [0. 0. 1. 0. 0. 0.]

[0.97265625 0.94921875 0.9296875 **0.99609375** 0.9375 0.] ==> [0. 0. 0. 1. 0. 0.]

[**0.921875** 0.80859375 0.6875 0.0390625 0.01953125 0.34765625] ==> [1. 0. 0. 0. 0. 0.]

[**0.89453125** 0.33984375 0.66015625 0.40234375 0.26171875 0.37890625] ==> [1. 0. 0. 0. 0. 0.]

[1. 1. 0.99609375 0. 0. 0.] ==> [0. 1. 0. 0. 0. 0.]

[0.99609375 0.27734375 0.95703125 0.1328125 0.00390625 0.078125] ==> [0. 0. 1. 0. 0. 0.]

[**0.91796875 0.4375** 0.7265625 0.23046875 0.1328125 0.24609375] ==> [1. 0. 0. 0. 0. 0.]

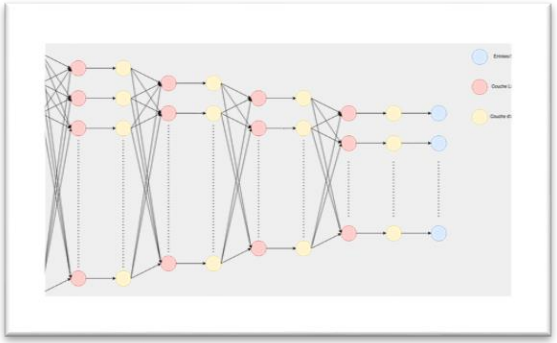
[0.75 0.5 0.6484375 0.44921875 0.4296875 0.390625] ==> [0. 1. 0. 0. 0. 0.]

[**0.98828125** 0.44140625 0.84765625 0.078125 0.015625 0.1328125] ==> [1. 0. 0. 0. 0. 0.]

HLS4ML pour le réseau R2D2 (Intégration dans ZCU102)

Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC

System Logic Cells (K)	600
Memory (Mb)	32.1
DSP Slices	2,520
Maximum I/0 Pins	328



IP avec 2083 entrées de 16 bits

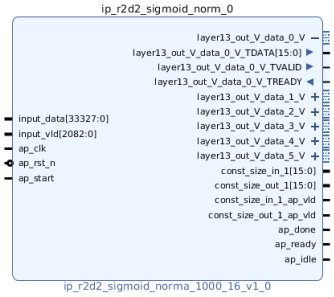
Effet du Reuse factor

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	0	-	2110	11816	-
Instance	318	138	159946	126469	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	318	138	162062	138353	0
Available	1824	2520	548160	274080	0
Utilization (%)	17	5	29	50	0

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	0	-	2110	11816	-
Instance	153	35	126988	134768	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	153	35	129104	146652	0
Available	1824	2520	548160	274080	0
Utilization (%)	8	1	23	53	0

÷2

Récriture de l'IP avec une seule entrée



☹ Les tableaux de tableaux non supportés en tant qu'entrées/sorties des entités sur les IP

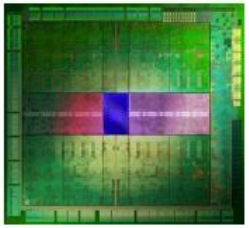
→ Solution ➔ array of std_logic ➔ Transformation ➔ vector (2083*16 downto 0)

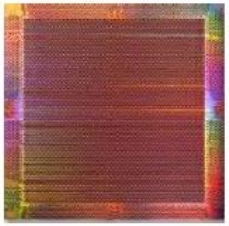
☺

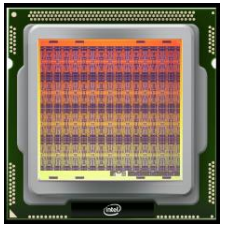
Les nouvelles architectures matérielles : Conclusion

MPPA


ASICs


GPUs


FPGAs


NMC

- Concevoir des systèmes IA embarqués nécessite :
- Une connsaissance en IA classique
- Une excellente appréhension des architectures matérielles
- Une spécialisation par solution matériel
  - Utilisation des ressources
  - Utilisation des outils d'optimisation du réseau

Conclusion

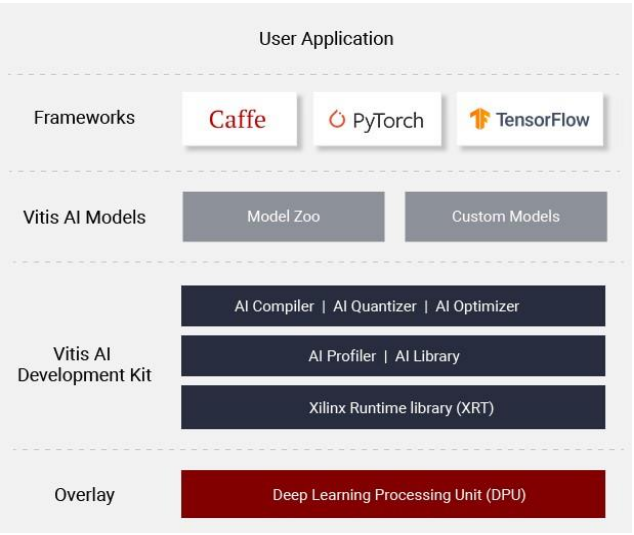
Fully Firmware:

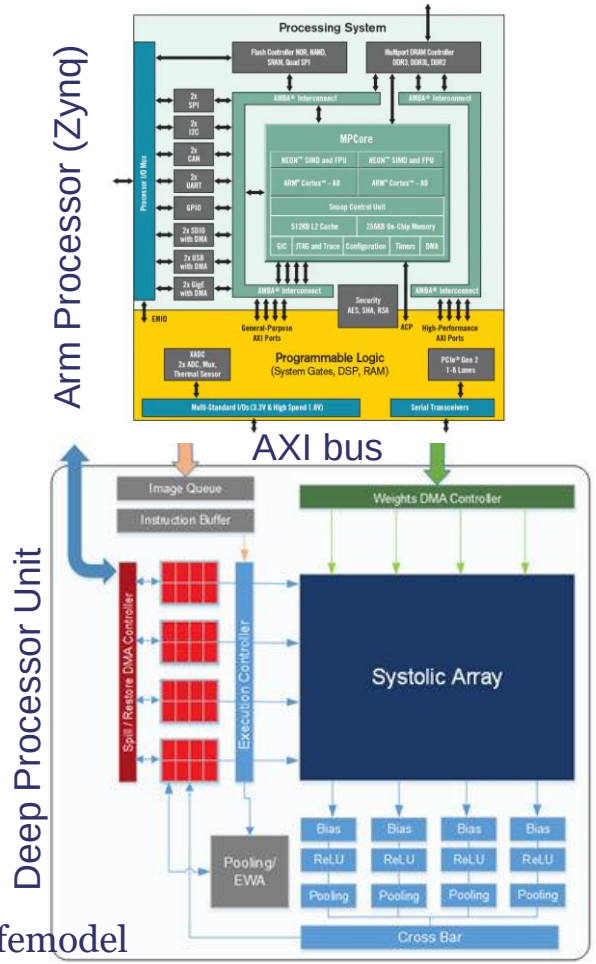
- Comprendre la generation du code vhdl
- Developper des outils de monitorage de l'élagage
- Améliorer la synthése en utilisant un CC

Edge Computing

- Ameliorer le linux embarqué
- Developper son propre modèle de réseaux
- Tester les circuits VERSAL

Architecture Vitis AI (edge computing)





- **tensorflow** : model.pb
- **pytorch** : model.xmodel
- **caffe** : model.prototxt/model.caffemodel

VITIS AI PROCESS
(Xilinx)

Keras/PyTorch Model in floating point

preprocessing

Quantize Weight and calibrate Activation function

Generate DPU model

Compile C++ code with optimization

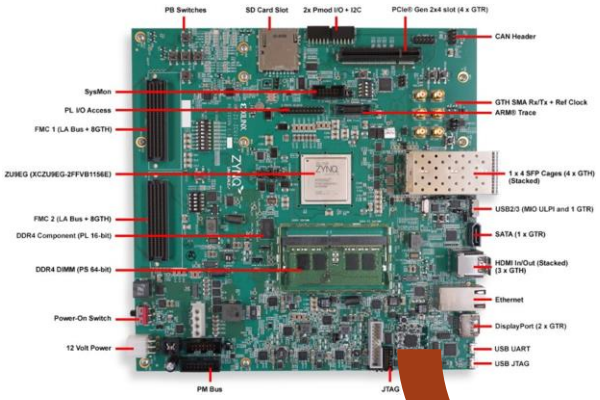
Outils sous Linux

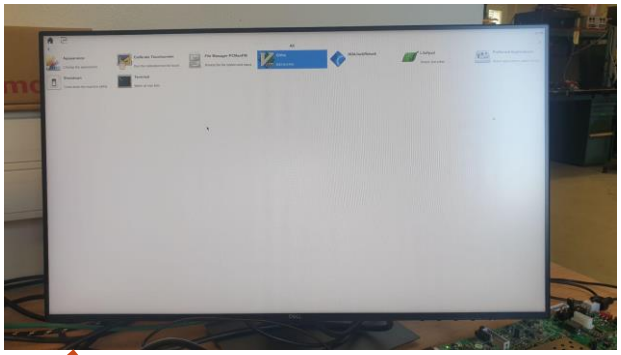
Télécharger le bloc Docker associé aux outils Tensorflow/Keras/Vitis-AI:

- **https://hub.docker.com/r/xilinx/vitis-ai**
- **docker pull xilinx/vitis-ai**

- Extraire les données (qq 10 milliers)
  - Données d'entrainements
  - Données de validation
  - Données de test
- Choix et création de l'architecture du RN
- Apprentissage du réseau (entrainement)
- Optimisation du réseau
- Conversion et gel du réseau
- Quantification des coefficients
- Compilation du RN pour le DPU
- Execution du réseau sur le DPU (ZCU102/ZCU104/Versal)

Principe de Vitis AI sur ZCU102





PetaLinux

- TEST DE RESNET50 SUR DES IMAGES
- TEST DE YOLO IMAGES ET VIDEO

Démonstration avec le “model Zoo” : Resnet50

Avion 91%

top[0] prob = 0.912573 name = **airliner**
top[1] prob = 0.074909 name = wing
top[2] prob = 0.010138 name = warplane,military plane
top[3] prob = 0.002262 name = space shuttle
top[4] prob = 0.000053 name = airship, dirigible

Corail cerveau 98%

top[0] prob = 0.980779 name = **brain coral**
top[1] prob = 0.008485 name = coral reef
top[2] prob = 0.008485 name = jackfruit, jak, jack
top[3] prob = 0.000542 name = sea urchin
top[4] prob = 0.000422 name = puffer, pufferfish, blowfish

oseille 39% / vache 31%

top[0] prob = 0.398545 name = **sorrel**
top[1] prob = 0.310387 name = **ox**
top[2] prob = 0.114185 name = redbone
top[3] prob = 0.042006 name = vizsla, Hungarian pointer
top[4] prob = 0.032715 name = worm fence, snake fence

Démonstration avec le “model Zoo” : Resnet50

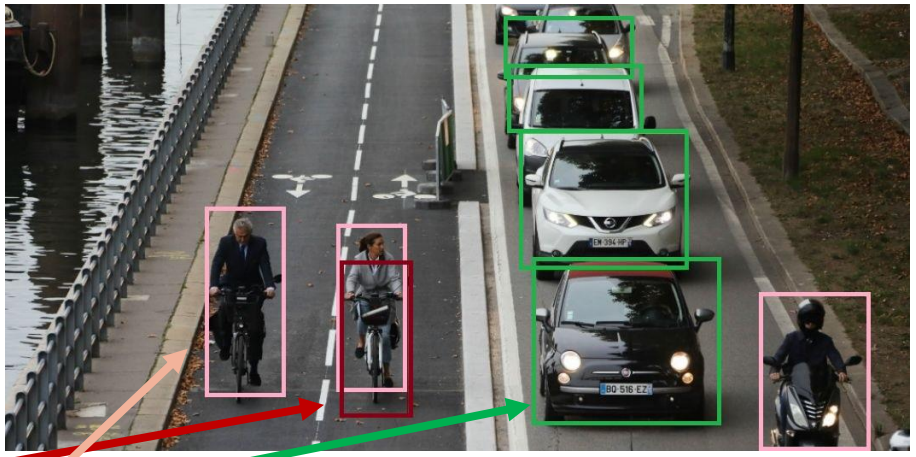
top[0]	prob = 0.951893	name = **teddy, teddy bear**
top[1]	prob = 0.010575	name = Chihuahua
top[2]	prob = 0.006414	name = Airedale
top[3]	prob = 0.003890	name = corboy hat, ten-gallon hat
top[4]	prob = 0.002360	name = toy poodle

➔ **main.cc** de **Resnet50** :

- **ListImages()** : fonction qui charge les images que l’on veut traiter.
- **LoadWords()** : fonction qui charge les différentes catégories que connait Resnet.
- **CPUCalcSoftmax()** : fonction qui calcule la fonction mathématique Softmax.
- **TopK()** : fonction qui renvoie le top K (dans notre exemple K=5) en terme de probabilité pour une image.
- **runResnet50()** : fonction qui fait le lien entre toutes les fonctions précédentes. Elle commence par charger les données nécessaires.

```
auto job_id = runner->execute_async(inputsPtr, outputsPtr);
runner->wait(job_id.first, -1);
for (unsigned int i = 0; i < runSize; i++) {
                    cout << "\nImage : " << images[n + i] << endl;
                     /* Calculate softmax on CPU and display TOP-5 classification results */
                    CPUCalcSoftmax(&FCResult[i * outSize], outSize, softmax);
TopK(softmax, outSize, 5, kinds);
 /* Display the impage */
cv::imshow("Classification of ResNet50", imageList[i]);
cv::waitKey(10000);
```

YOLO : traitement multicible sur images

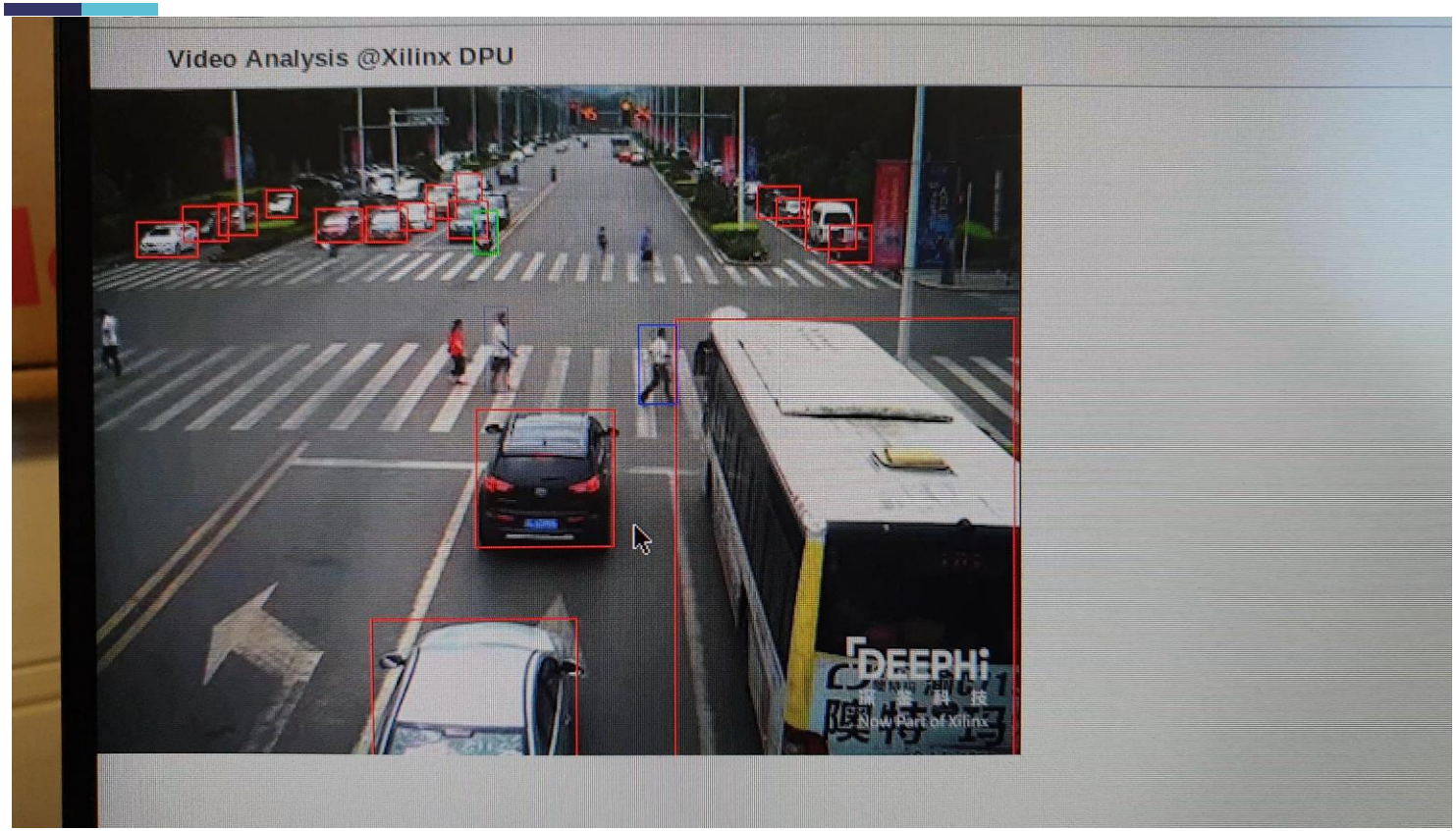


RESULT : 1 395.969 302.705 471.263 483.167 0.779274
RESULT : 6 586.168 16.2274 734.577 84.4693 0.979318
RESULT : 6 604.624 151.448 797.66 310.705 0.968632
RESULT : 6 618.412 299.663 837.151 491.764 0.928414
RESULT : 6 589.709 72.6563 747.69 149.985 0.607446
RESULT : 14 236.443 240.04 327.627 457.719 0.998934
RESULT : 14 390.115 260.296 470.584 452.397 0.99409
RESULT : 14 885.904 341.999 1010.54 522.461 0.989938

YOLO : traitement multicible sur video

Traitement vidéo en ligne sur ZCU102

YOLO : traitement multicible sur video



Traitement vidéo en ligne sur ZCU102