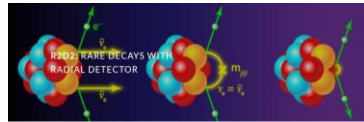


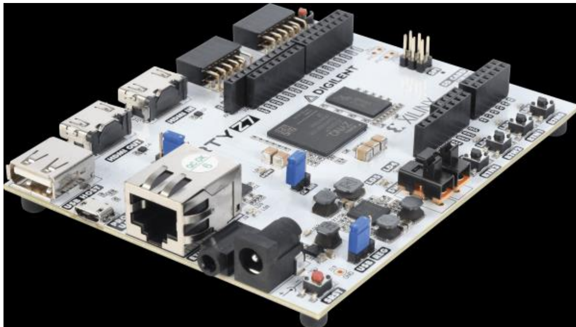
IA embarquée sur Xilinx pour l'électronique de lecture de R2D2 (Capteurs)

Frédéric Druillole, CENBG

19 Novembre 2021

Méthodologie et test de portage de réseaux de neurones sur FPGA Xilinx Zynq

ARTY Z7



ZCU102

CENBG LP2i

Sommaire

Processus d'intégration d'IA en FPGA

Utilisation de l'outil HLS4ML (IA embarqué)

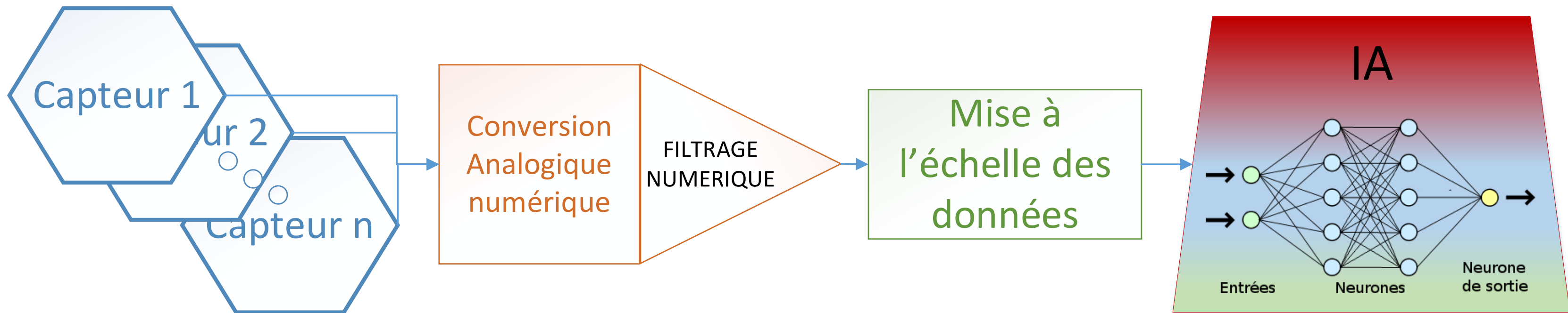
- Sur les challenges #1 #3 #5 #7
- Sur R2D2-OWEN
- Sur MNIST

Demonstration de Vitis AI (edge computing)

- Utilisation de reseaux entrainés Resnet50 et YOLO

Notre objectif

Nœud IA Level 0



Nœud IA L2

Nœud IA L1

Nœud IA L1

Nœud IA L0

Nœud IA L0

Nœud IA L0

Nœud IA L0

Nœud IA L0

Nœud IA L0

Nœud IA L0

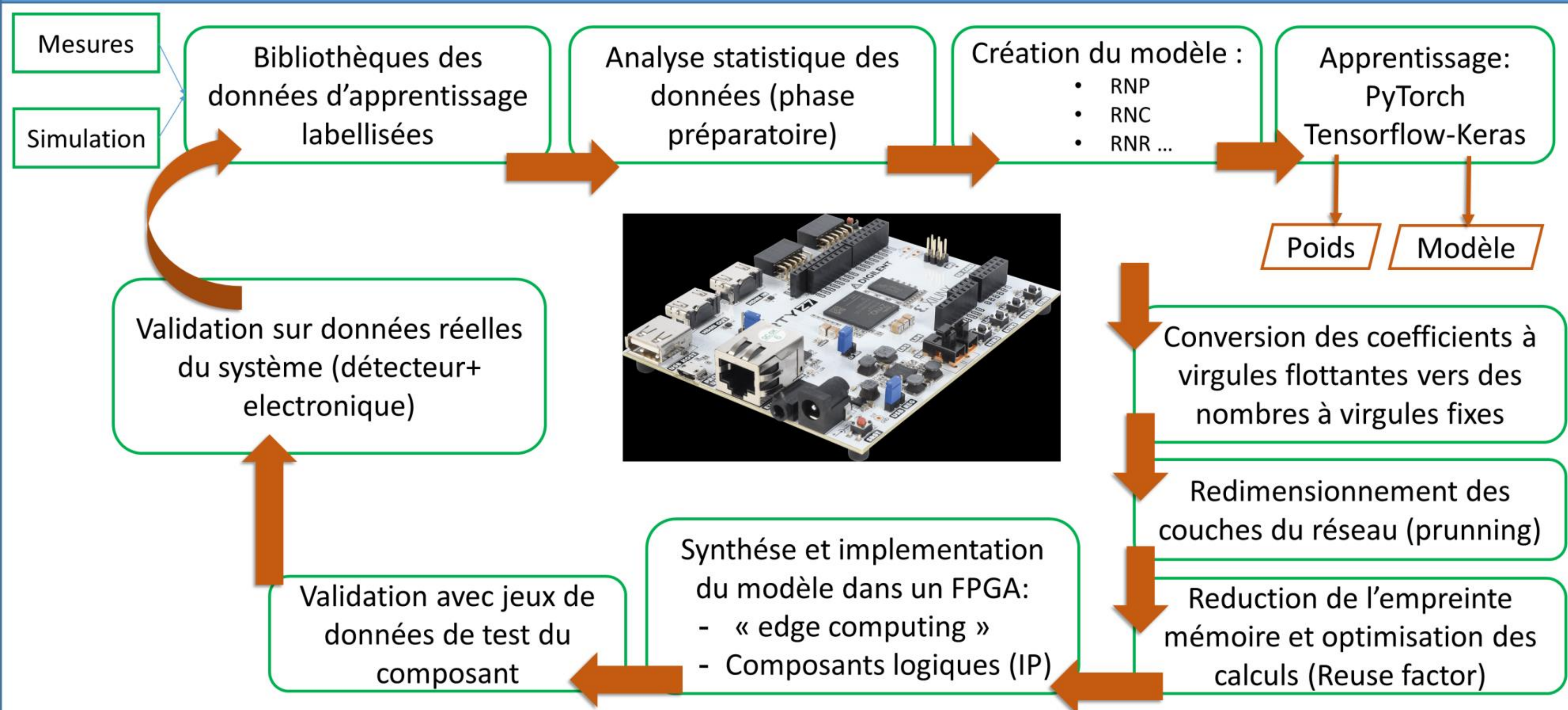
Nœud IA L0

CENBG LP2i R2D2 OWEN THINK

Pourquoi Utiliser des FPGA pour l'IA ?

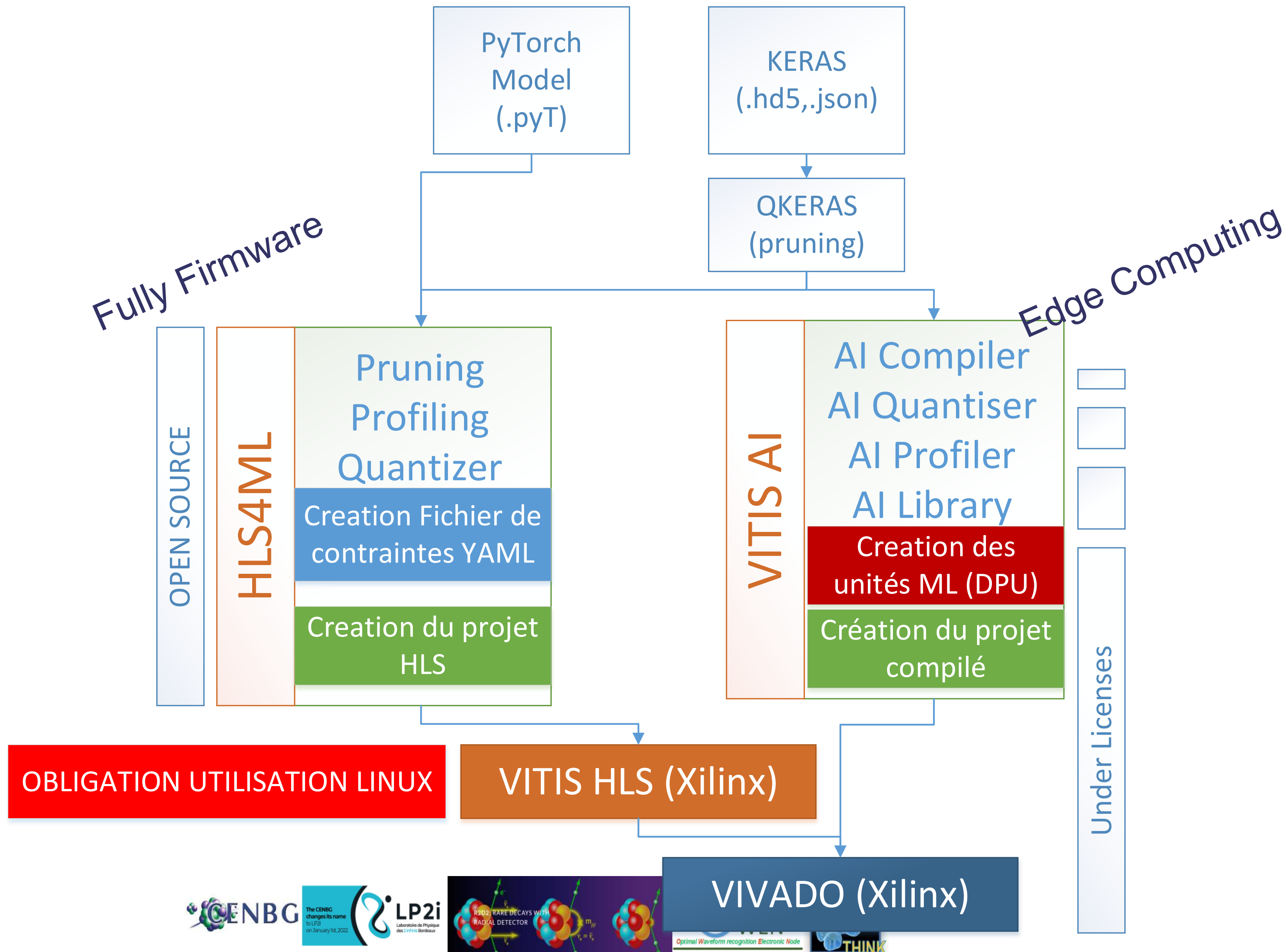
- **Efficacité des calculs**
  - Adaptation, flexibilité et personnalisation des fonctions
  - Nombreux blocs arithmétiques dédiés
  - Garantie des archirtecture IA performants (basse latence)
- **La puissance consommé**
  - Haute performance par rapport aux Watt consommées
  - Efficacité energiétique des systèmes embarqués
  - Simplification du système de refroidissement
- **À l'épreuve du future**
  - Nouveaux algorithmes d'IA arrivent rapidement
  - Architecture FPGA s'adapte facilement
  - Associe les algorithmes d'IA avec les autres tâches
  - Accélère l'ensemble des fonctions

INTEGRATION CONTINUE



CENBG LP2i WEN THINK

2 façon d'intégrer l'IA en SE

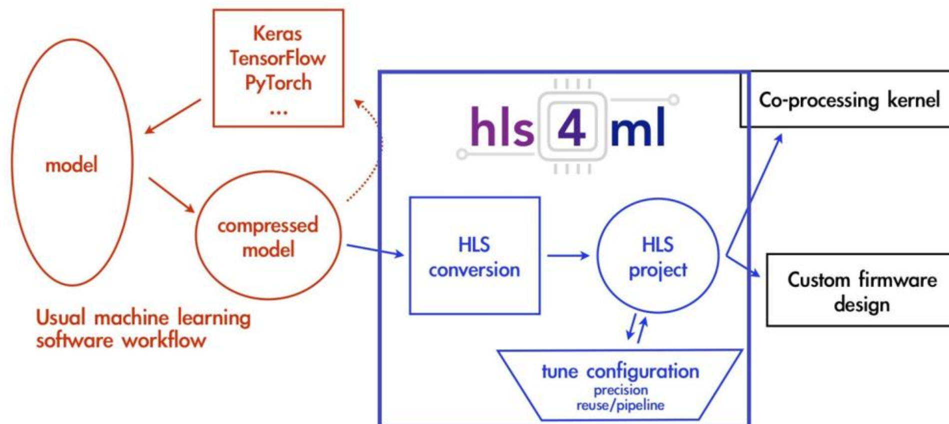


Principe HLS4ML (fully firmware)

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

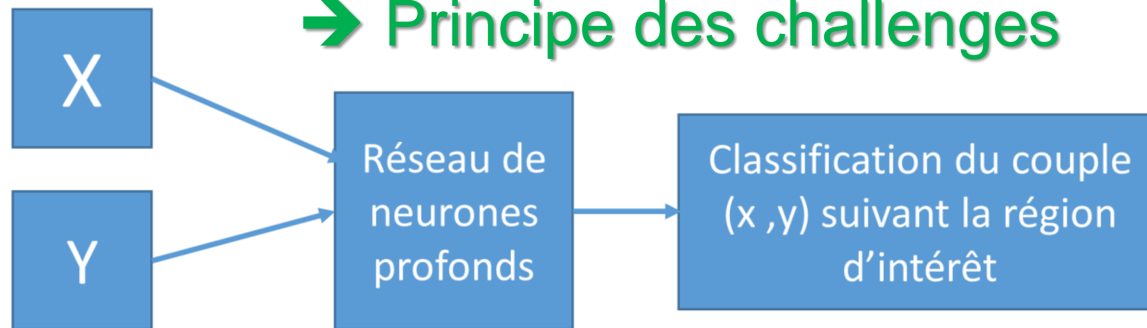
Optimize design reuse / pipeline/io

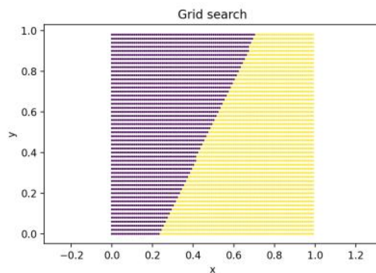
Generate a Vivado IP

Add DNN/CNN IP in your Vivado project

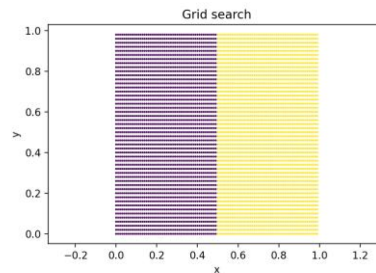
CENBG LP2i OWEN THINK

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

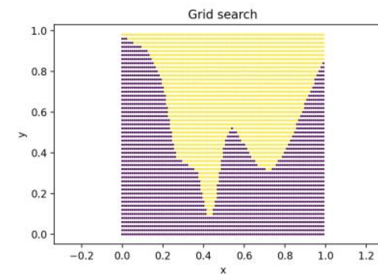




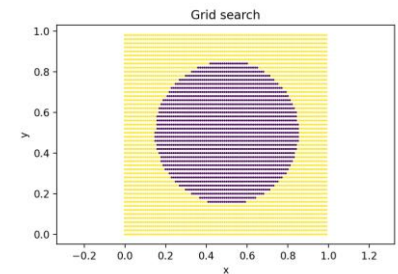
CH1



CH3



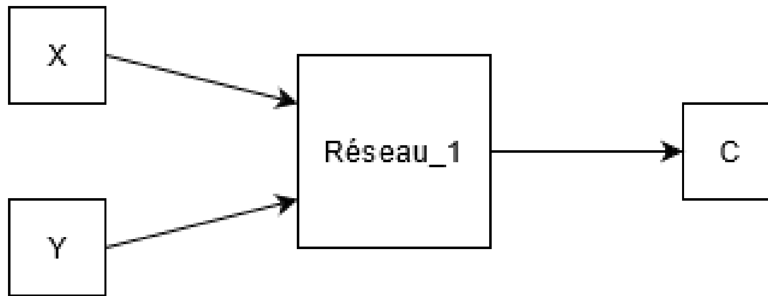
CH5



CH7

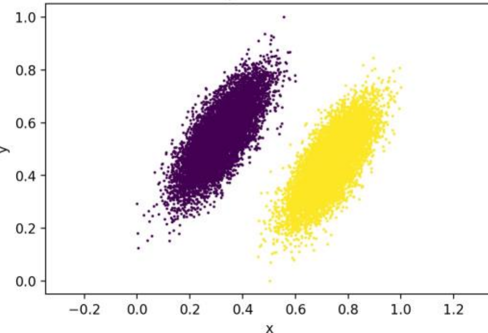
CENBG LP2i R2D2 OWEN THINK

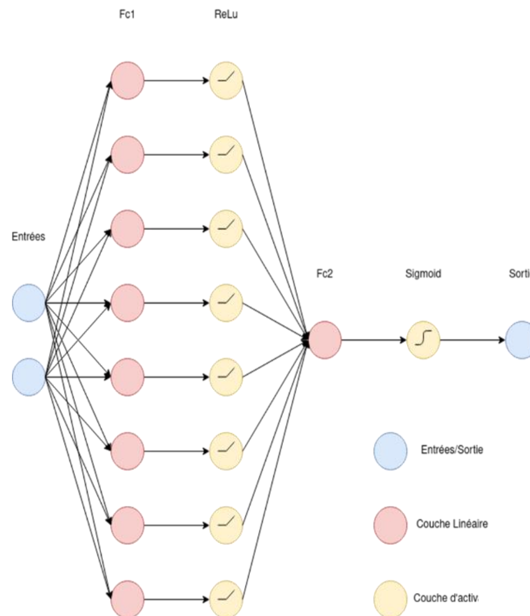
Un exemple de calcul: réseau de neurone peu profond

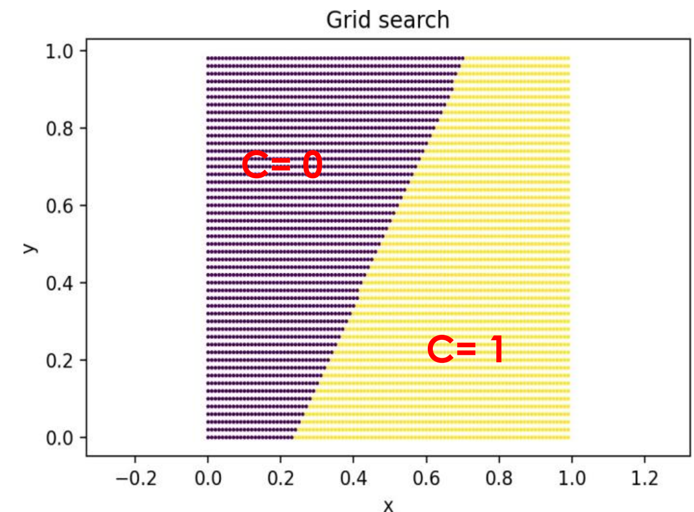


Principe: on fournit en entrée des coordonnées (x,y). Le réseau nous classifie les coordonnées selon la région d'appartenance. Ici, c'est une simple régression logistique avec une séparation de deux région linéaire.

Données d'apprentissage et de test





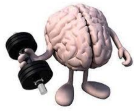


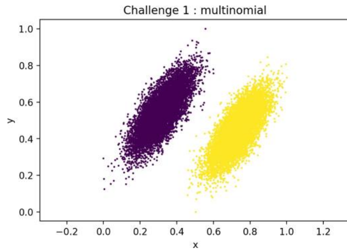
Il faut calculer

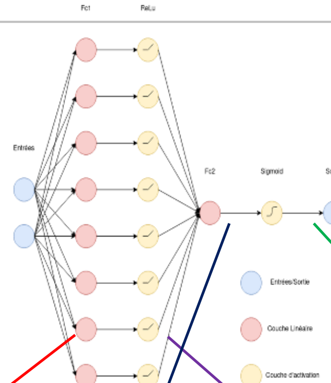
$$W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{pmatrix} \text{ et } B = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

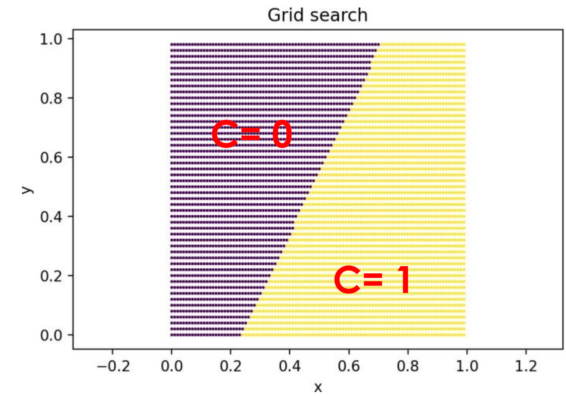
$$s = \sigma(\mathbf{W}x + \mathbf{B})$$

Un exemple de calcul: réseau de neurone peu profond









X=0,2
Y=0,7

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

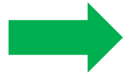
$$\begin{pmatrix} -0.4275 \\ -0.4294 \\ 0.5951 \\ -0.6608 \\ 1.7938 \\ -0.7465 \\ -0.3293 \\ 1.2358 \end{pmatrix} \xrightarrow{ReLu} \begin{pmatrix} 0 \\ 0 \\ 0.5951 \\ 0 \\ 1.7938 \\ 0 \\ 0 \\ 1.2358 \end{pmatrix}$$

$$(2.0441\ 2.5193 - 0.6255\ 0.0535\ - 1.5080\ 3.7406\ 0.1052\ - 1.2330)\ \text{X} \begin{pmatrix} 0 \\ 0 \\ 0.5951 \\ 0 \\ 1.7938 \\ 0 \\ 0 \\ 1.2358 \end{pmatrix} + (-1.4529) = (-6.0539)$$

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

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

Réseaux décrits en PyTorch



```
dims=[2,8] nb_epoch=50
class mlp(torch.nn.Module):
        def __init__(self,dims):
            super(mlp, self).__init__()
            self.fc1=torch.nn.Linear(dims[0],dims[1])
            self.relu=torch.nn.ReLU()
            self.fc2=torch.nn.Linear(dims[1],1)
            self.sigmoid=torch.nn.Sigmoid()
        def forward(self,input):
            a = self.fc1(input);
            b = self.relu(a);
            c = self.fc2(b);
            d = self.sigmoid(c);
            return d;
```

- - Ordre des couches est obligatoire pour HLS4ML
- Quantification des poids

Configuration pour la conversion en HLS

fichier .yml

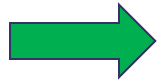
```
PytorchModel: ch1.pth
InputData:
OutputPredictions:
OutputDir: /ch1
ProjectName: reseau_ch1
XilinxPart: xc7z020clg400-1
ClockPeriod: 8

IOType: io_parallel # options: io_serial/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,7>
    ReuseFactor: 1
    Strategy: Latency
```

Les paramètres importants pour HLS4ML

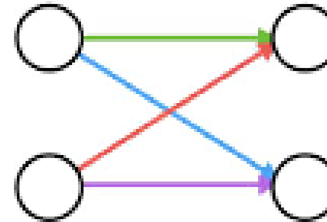
La **précision** de la quantification (Vigule flottante → Virgule fixe)

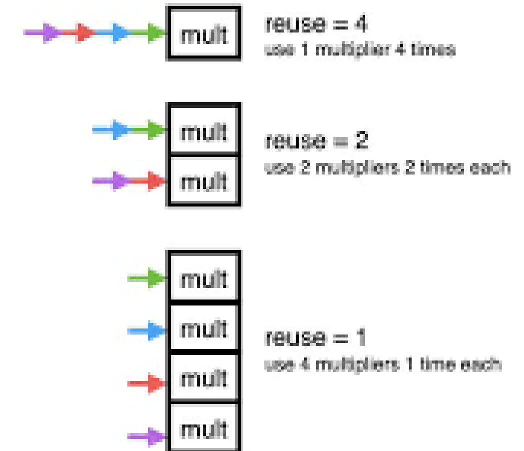
Exemple : ap_fixed<16,6>

15	0.75
0 0 1 1 1 1	1 1 0 0 0 0 0 0 0 0



Reuse Factor ➔ Nombre de fois que l'IP boucle sur la ressource de calcul du FPGA



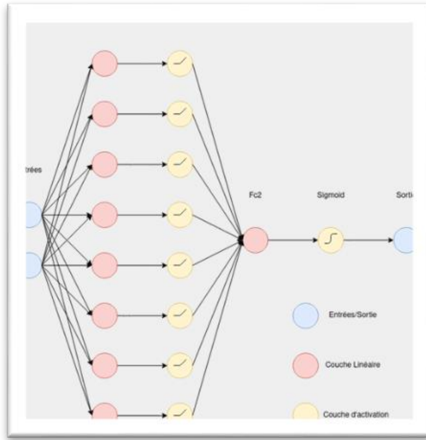


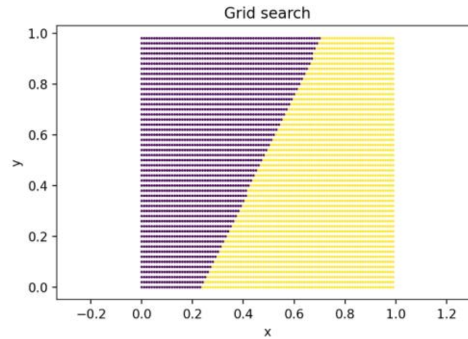
Strategy :

➔ Latence : privilégie la vitesse d'écution

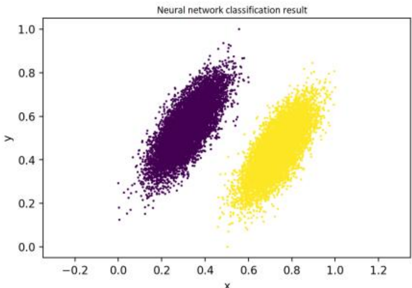
➔ Minimise les ressources au détriment de la latence

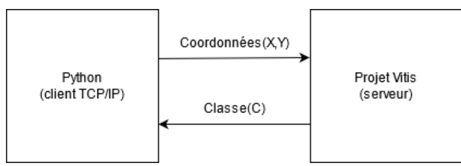
Resultats CH1

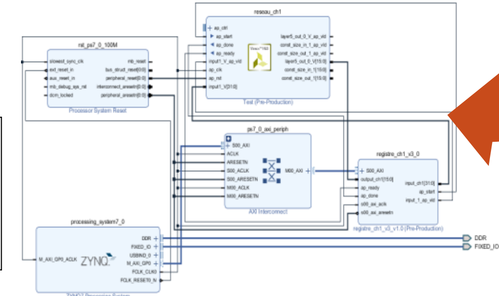




Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
9	9	72.000 ns	72.000 ns	1	1	function





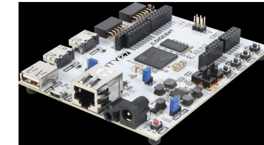


Test de l'IP en client/serveur entre le PC et la carte **ARTY Z7**

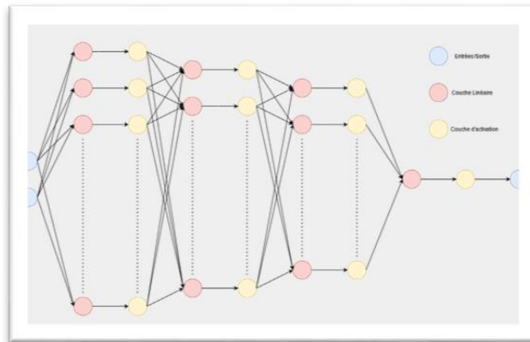
- 1 couche avec ReLu
  1 couche avec Sigmoïde

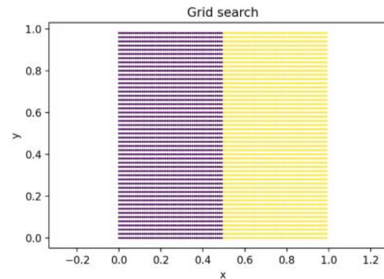
- Evaluation sur PC (python) : 100%
  Evaluation sur Arty Z7 : 100%

Resultats CH3



python



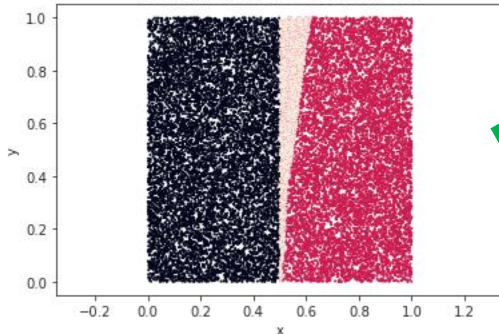


HLS

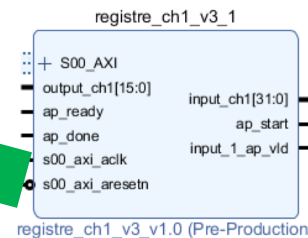
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	2824	-
FIFO	0	-	3515	17932	-
Instance	17	26	14233	38297	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	6300	-
Register	-	-	700	-	-
Total	17	26	18448	65353	0
Available	280	220	106400	53200	0
Utilization (%)	6	11	17	122	0

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 Sigmoïde

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

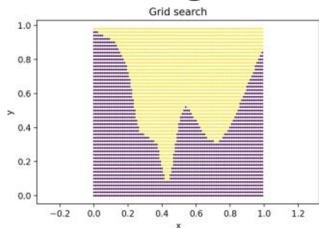
CENBG LP2i WEN THINK

Test avec CH5 et CH7

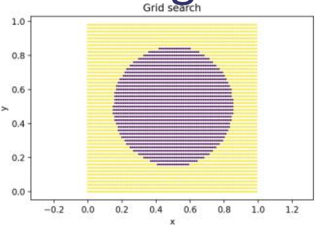
réseau	nombre de paramètres
CH5/CH7	313 016
CH3	51 263

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

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)

    lef 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
```

```
model = Sequen
model.add(QDe
          inpu
          kerr
          bias
          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"))
```

```
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"))
```

Elagage et Reuse Factor

Qkeras ➔ Entraine le réseau avec des coefficient en virgules fixes avant 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	141743	224683	0
Available	1824	2520	548160	274080	0
Utilization (%)	4	6	25	81	0

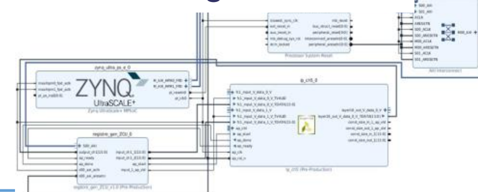
a) Ressource pour CH5

Rf = 10000

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	0	-	9005	50428	-
Instance	96	15	124972	167433	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	96	15	133983	217929	0
Available	1824	2520	548160	274080	0
Utilization (%)	5	~0	24	79	0

b) Ressource pour CH7

VIVADO design Valide

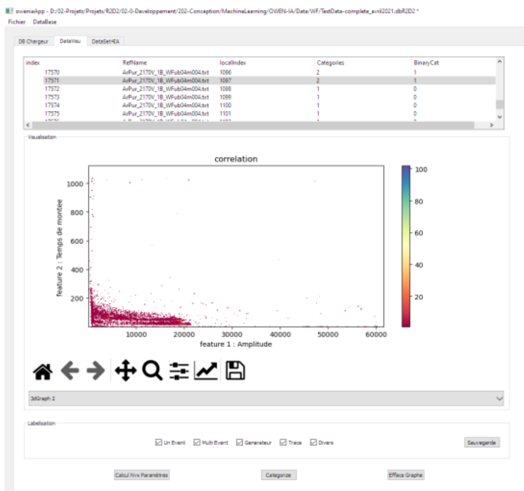


VIVADO crashe lors de la synthèse de CH5 et CH7 ➔ ??

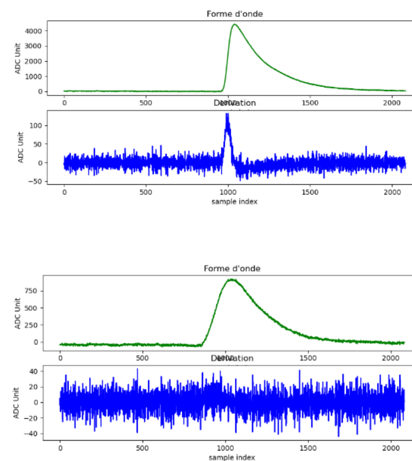
CENBG LP2i R2D2 WEN THINK

HLS4ML pour le réseau R2D2 (Rappel)

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



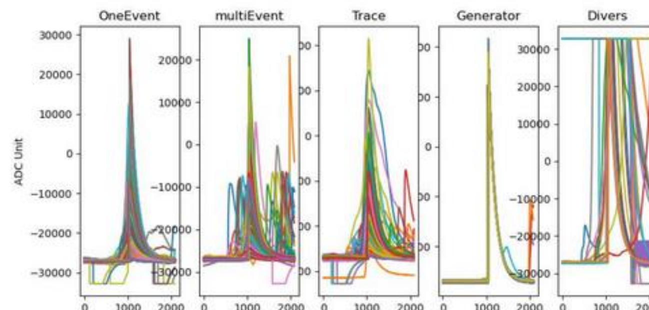
➔ préamplificateur de charge (intégration), la dérivée ➔ l'évènement physique (détecteur).



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

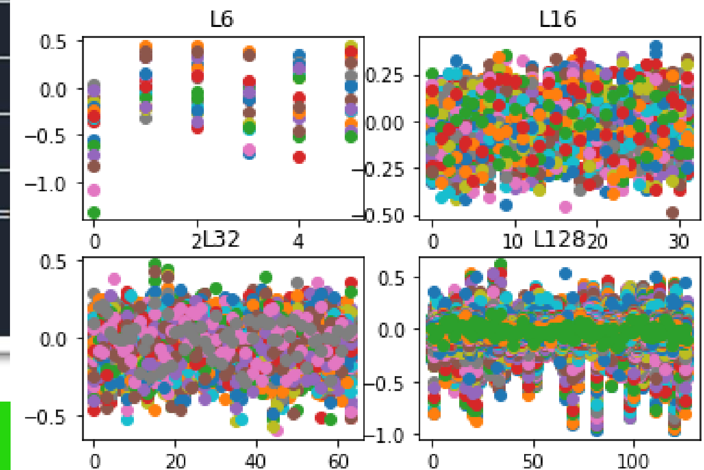


HLS4ML pour le réseau R2D2 (Apprentissage)

➔ Moins de 20000 formes d'ondes

Réseau 128x64x32x16x6

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               266752
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)
_________________________________________________________________
dense_2 (Dense)              (None, 32)
_________________________________________________________________
dense_3 (Dense)              (None, 16)
_________________________________________________________________
dense_4 (Dense)              (None, 6)
=================================================================
Total params: 277,718
Trainable params: 277,718
Non-trainable params: 0
```



Elimination des colonnes inutiles

Mise à l'échelle des données

Apprentissage DNN

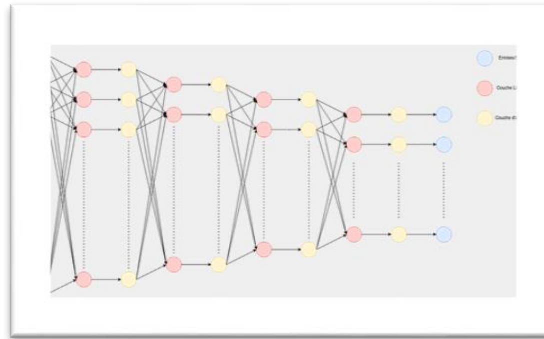
La fonction de coût sur l'échantillon test vaut 0.503.
Le taux de bonnes réponses est 0.8

CENBG LP2i R2D2 OWEN THINK

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



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

fc1_input_V_data_2078_V
fc1_input_V_data_2079_V
fc1_input_V_data_2080_V
fc1_input_V_data_2081_V
fc1_input_V_data_2082_V
fc1_input_V_data_2082_V_TVALID
fc1_input_V_data_2082_V_TREADY
fc1_input_V_data_2082_V_TDATA[15:0]
ap_ctrl
ap_start
ap_done
ap_ready
ap_idle
ap_clk
ap_rst_n
ip_r2d2_sigmoid (Pre-Production)

IP avec 2083 entrées de 16 bits

Récriture de l'IP avec une seule entrée

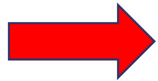
ip_r2d2_sigmoid_norm_0
input_data[33327:0]
input_vld[2082:0]
ap_clk
ap_rst_n
ap_start
layer13_out_V_data_0_V
layer13_out_V_data_0_V_TDATA[15:0]
layer13_out_V_data_0_V_TVALID
layer13_out_V_data_0_V_TREADY
layer13_out_V_data_1_V
layer13_out_V_data_2_V
layer13_out_V_data_3_V
layer13_out_V_data_4_V
layer13_out_V_data_5_V
const_size_in_1[15:0]
const_size_out_1[15:0]
const_size_in_1_ap_vld
const_size_out_1_ap_vld
ap_done
ap_ready
ap_idle
ip_r2d2_sigmoid_norma_1000_16_v1_0

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)

CENBG LP2i WEN THINK

Evaluation du réseau MLP de R2D2



Dernière couche **SOFTMAX**

SOFMAX en virgule flottante <16,6>

data	classe	sortie 0	sortie 1	sortie 2	sortie 3	sortie 4	sortie 5
0	4	0	2 436	0	69	64 056	0

SOFMAX dépasse l'intervalle [0;1]

data	classe	sortie 0	sortie 1	sortie 2	sortie 3	sortie 4	sortie 5
0	4	0.0004	2.378	0	0.067	-30,55	0

Changement de la couche SoftMax par une fonction d'activation « Sigmoïde »

[0,1024] ➔ [0,1], précision <16,6>

data	classe	sortie 0	sortie 1	sortie 2	sortie 3	sortie 4	sortie 5
0	4	0	1007	950	1023	1023	484
1	1	0	1023	919	1023	1023	484
2	1	0	1023	939	1023	1023	484
3	1	0	1023	961	1023	0	484
4	1	0	1023	931	1023	0	484
5	1	0	023	924	1023	1023	484
66	3	0	1023	982	1023	1023	484
285	4	0	1001	934	1023	1023	484
222	5	1002	0	512	0	4	484
211	2	1023	1023	1019	0	0	484

TABLE 6 – Sorties de 10 échantillons de test du réseau R2D2

Conclusion:

- Réseau IA R2D2 intégré dans un Zynq puissant (ZCU102)
- Mauvais résultats de classification

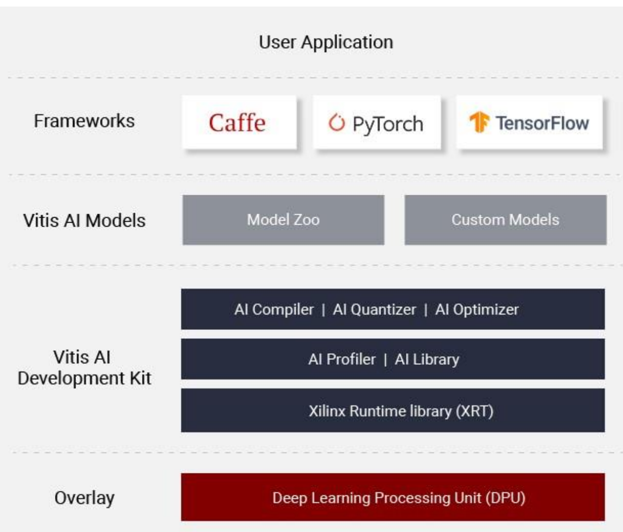
➔ **Dû au passage en nombre en virgule fixe (à comprendre)**

Synthése

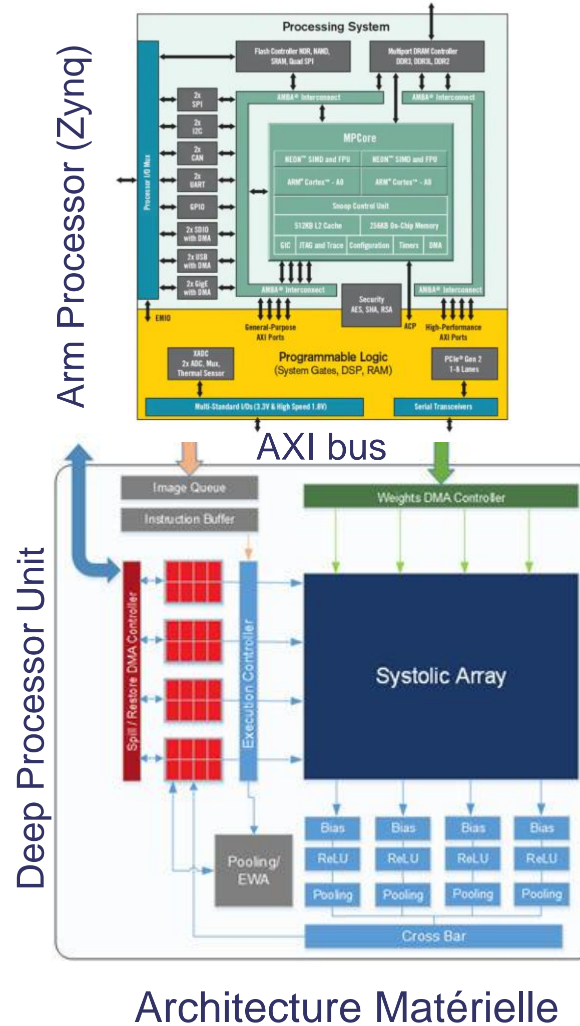
- Le modèle du DNN
  - < 100 000 coefficients
  - < entrées simple
- Travailler sur la précision
  - Comprendre l'influence de la transformation virgule flottante/virgule fixe
  - Influence de l'élagage
- Besoin en puissance de calcul
  - Etude des ressources des FPGA (DSP, LUT) par rapport au modèle
  - RAM pour la synthèse sous VIVADO
  - Interface d'entrées des IPs

Architecture Vitis AI (edge computing)

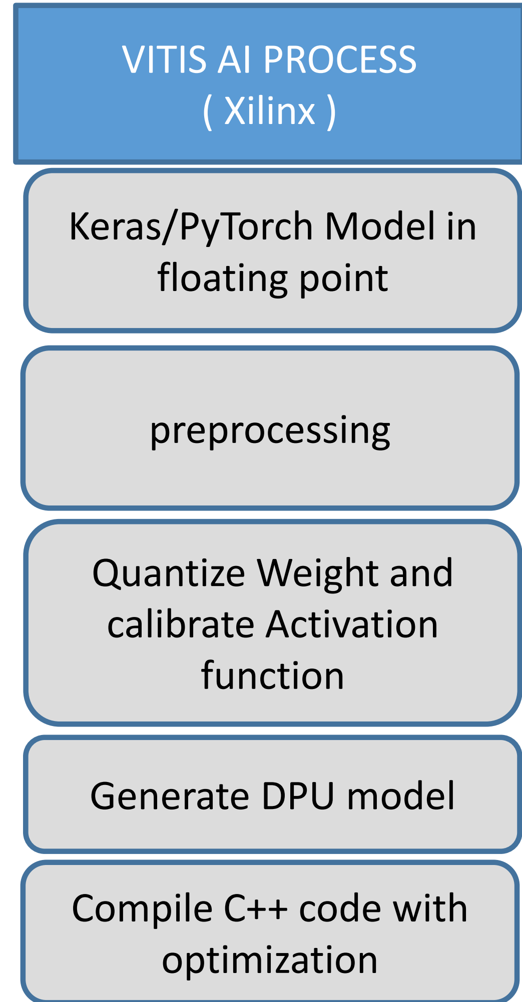
https://www.xilinx.com/html_docs/vitis_ai/1_4/zmw1606771874842.html



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



Architecture Matérielle

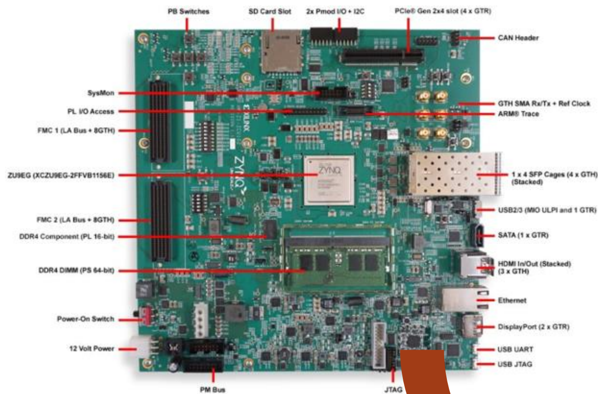


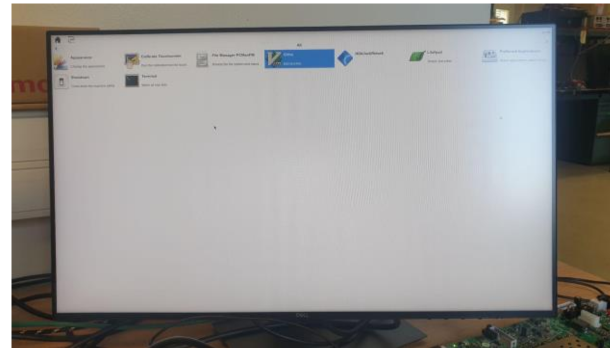
Télécharger le bloc Docker associé aux outils Tensorflow/Keras/Vitis-AI:

- **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



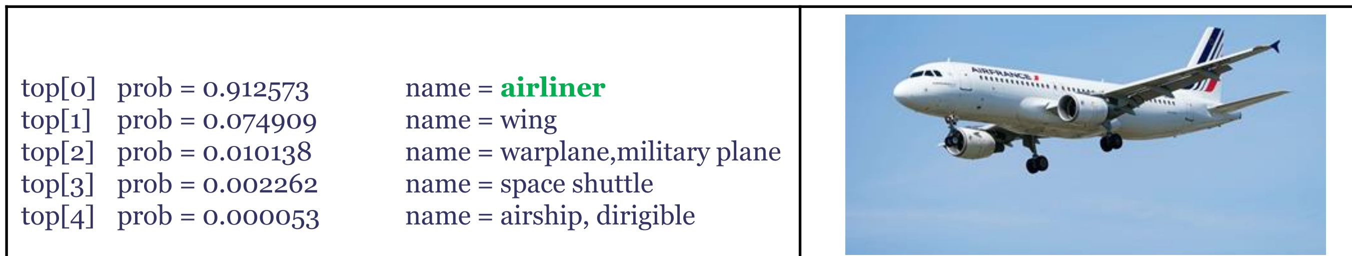


PetaLinux

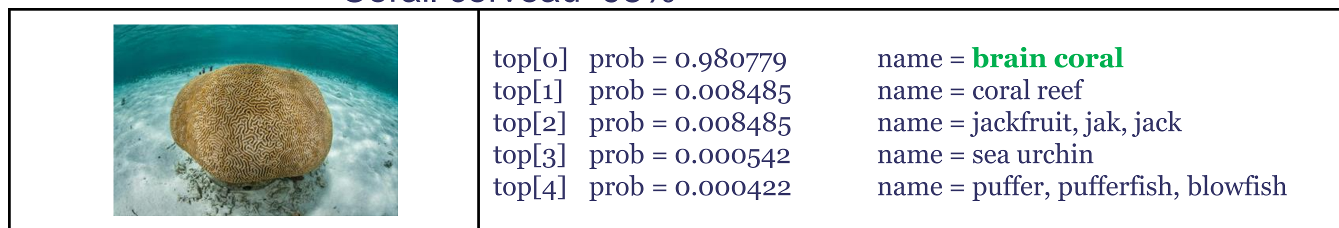
- TEST DE RESNET50 SUR DES IMAGES
- TEST DE YOLO IMAGES ET VIDEO

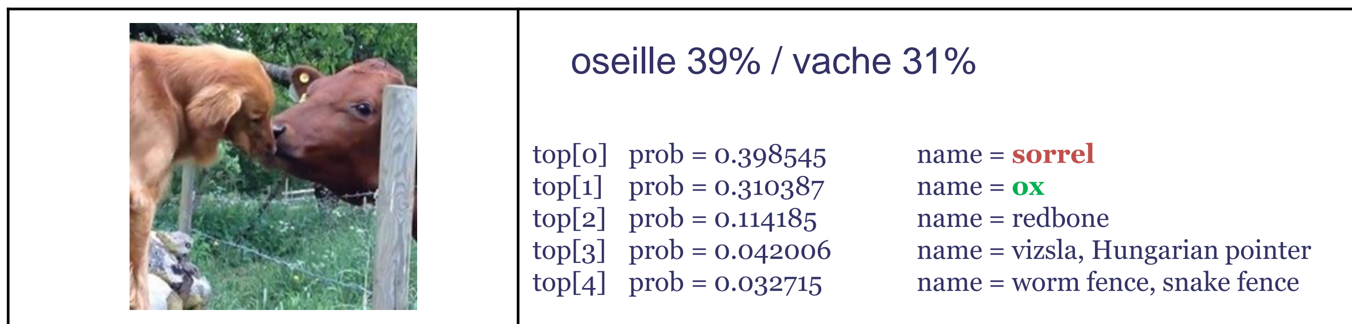
Démonstration avec le “model Zoo” : Resnet50

Avion 91%



Corail cerveau 98%





CENBG LP2i WEN THINK

Démonstration avec le “model Zoo” : Resnet50

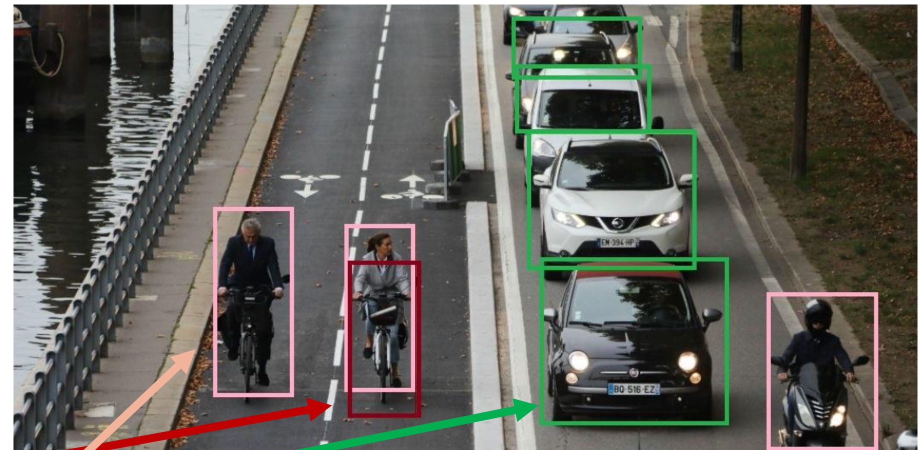
```
top[0]   prob = 0.951893        name = teddy, teddy bear
top[1]   prob = 0.010575        name = Chihuahua
top[2]   prob = 0.006414        name = Airedale
top[3]   prob = 0.003890        name = corboy hat, ten-gallon hat
top[4]   prob = 0.002360        name = toy poodle
```

➔ **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	395.669	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

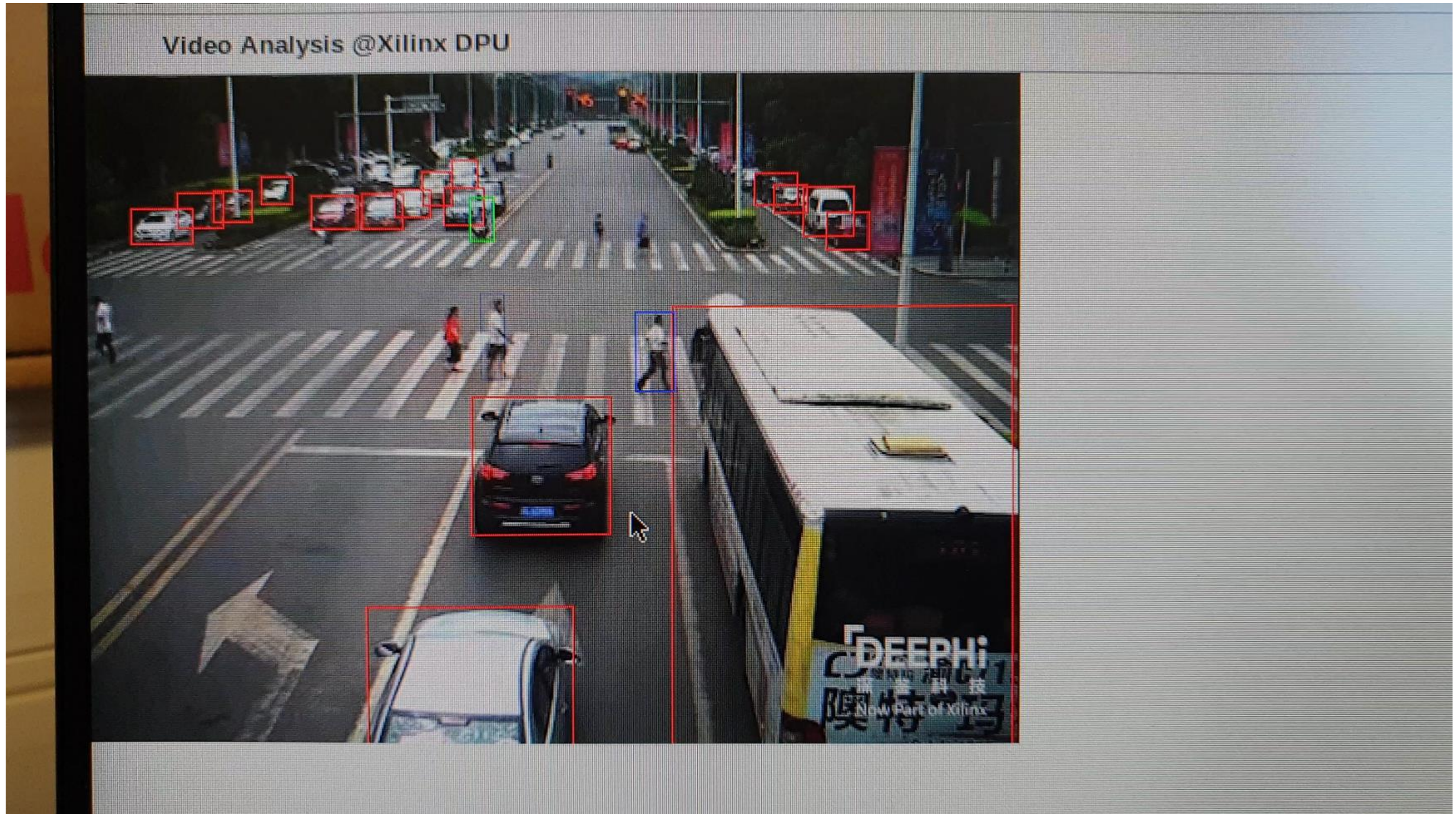
CENBG
The CENBG changes its name to LP2i on January 1st, 2022
LP2i Laboratoire de Physique des 2 Infinis Bordeaux
R2D2: RARE DECAYS WITH RADIAL DETECTOR
OWEN Optimal Waveform recognition Electronic Node
THINK

YOLO : traitement multicible sur video

Traitement vidéo en ligne sur ZCU102

CENBG LP2i WEN THINK

YOLO : traitement multicible sur video



Traitement vidéo en ligne sur ZCU102

CENBG LP2i WEN THINK

Conclusion

Fully Firmware:

- Comprendre la génération du code vhdl
- Developper/Utiliser des outils de monitorage de l'élagage
- Améliorer la synthèse en utilisant un CC

Edge Computing

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

CENBG The CENBG changes its name to LP2i on January 1st, 2022 LP2i Laboratoire de Physique des 2 Infinis Bordeaux
R2D2: RARE DECAYS WITH RADIAL DETECTOR
OWEN Optimal Waveform recognition Electronic Node
THINK