Rapport Apprentissage profond sur système embarqué

Nicolas MICHA, Guillaume MARANINCHI

decembre 2025

Table des matières

1 Introduction

L'apprentissage profond (Deep Learning) nécessite généralement une puissance de calcul importante. Cependant, le déploiement de ces algorithmes sur des systèmes embarqués, comme les FPGA, impose des contraintes strictes en termes de ressources (mémoire, logique, consommation).

L'objectif de ce projet est d'implémenter un Réseau de Neurones Profond (DNN) sur une carte **Nexys A7-100T** équipée d'un FPGA **Artix-7 (xc7a100tcsg324-1)**. Le but est de classifier des coordonnées (X, Y) appartenant à deux régions distinctes d'un plan.

Le projet se déroule en deux phases. Premièrement, l'étude et l'optimisation théorique des modèles via **HLS4ML** (Quantization/Pruning). Deuxièmement, l'intégration système dans **Vivado**. Il est à noter que pour cette seconde phase d'implémentation physique et de validation sur carte, nous avons utilisé une IP de référence fournie par l'encadrant, nos propres modèles générés présentant des instabilités lors de la synthèse finale. Cela nous a permis de nous concentrer sur la validation de l'architecture système (Microblaze, ILA).

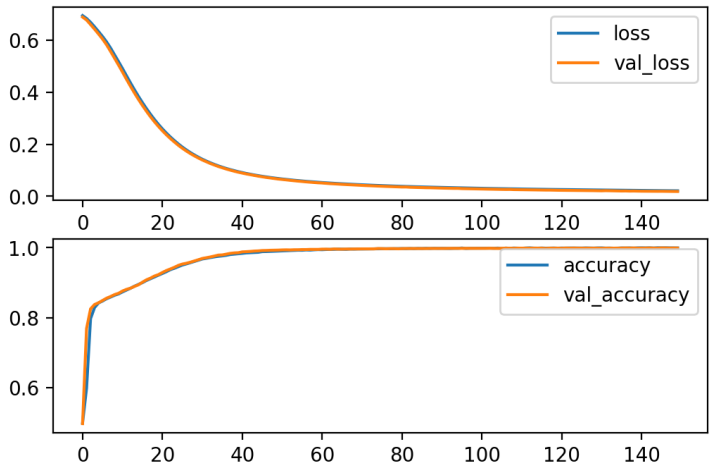
2 Analyse et Implémentation des Modèles

2.1 Modèle 1 : Architecture de Référence

2.1.1 Définition et Apprentissage

Le premier modèle sert de référence ("Baseline"). Il s'agit d'un Perceptron Multicouche (MLP) défini avec l'API Keras. L'architecture imposée est la suivante : une couche d'entrée de dimension 2, suivie de trois couches cachées de 200, 100 et 50 neurones (activation ReLU), et une sortie sigmoïde.

Les données d'entraînement (10 000 échantillons) sont générées synthétiquement pour séparer le plan en deux zones (gauche/droite). L'entraînement est réalisé sur plusieurs époques avec l'optimiseur Adam. La précision obtenue est proche de 100%.



2.1.2 Conversion HLS et Limites

Pour la conversion vers le FPGA Artix-7, nous avons configuré HLS4ML avec un `ReuseFactor` de 100 et une précision de `ap_fixed<8,2>` L'analyse du rapport de synthèse montre que ce modèle est très coûteux en ressources. Le nombre élevé de paramètres (plus

de 25 000 coefficients) combiné à une mauvaise précision sature les blocs DSP (Digital Signal Processing) et les LUTs (voir Table 1) disponibles sur l'Artix-7. Ces résultats sont aussi liés aux pragmas utilisés pour la synthèse (voir 2.4). Ce modèle n'est pas viable pour une intégration système complète sans optimisation.

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	–	–	–	–	–
Expression	–	–	0	2858	–
FIFO	0	–	3515	14302	–
Instance	59	255	30243	79279	–
Memory	–	–	–	–	–
Multiplexer	–	–	–	6336	–
Register	–	–	706	–	–
Total	**59**	**255**	**34464**	**102775**	**0**
Available	**270**	**240**	**126800**	**63400**	**0**
Utilization (%)	**21**	**106**	**27**	**162**	**0**

TABLE 1 – Ressources FPGA utilisées

2.2 Modèle 2 : Sparsité et QKeras

2.2.1 Stratégie d'Optimisation

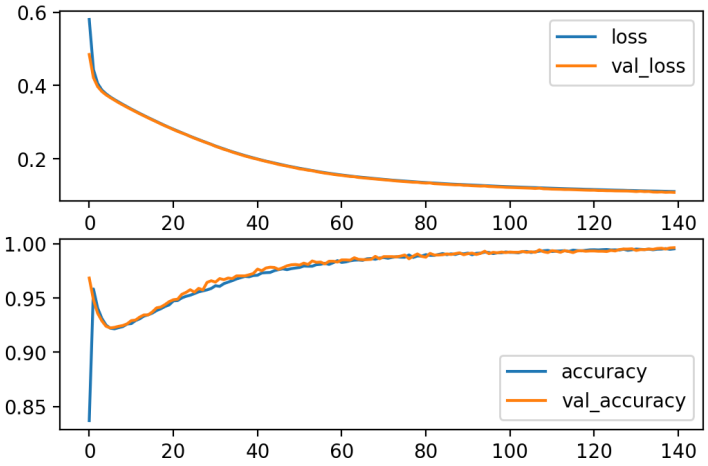
Le second modèle vise à réduire l'empreinte matérielle via deux techniques :

— **Quantification (QKeras) :** Utilisation de couches `QDense` avec des poids quantifiés sur 16 bits (`quantized_bits(16,2)`) et des activations sur 9 bits (`quantized_relu(9,2)`).

— **Élagage (Pruning) :** Application d'une sparsité cible de **75%** durant l'entraînement. Cela signifie que 75% des poids sont forcés à zéro.

2.2.2 Synthèse et Résultats

L'outil HLS4ML exploite la sparsité : lors de la génération du code C++, les multiplications par zéro sont détectées et les opérateurs matériels associés sont supprimés. Le rapport de synthèse confirme une réduction significative des LUTs (Look-Up Tables) et des DSPs par rapport au Modèle 1, tout en conservant une précision supérieure à 98%. Néanmoins par manque de temps, nous n'avons pas pu créer un modèle qui s'intègre dans le FPGA.



Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	–	–	–	–	–
Expression	–	–	0	4270	–
FIFO	0	–	5265	27210	–
Instance	114	255	48041	92905	–
Memory	–	–	–	–	–
Multiplexer	–	–	–	9486	–
Register	–	–	1056	–	–
Total	**114**	**255**	**54362**	**133871**	**0**
Available	**270**	**240**	**126800**	**63400**	**0**
Utilization (%)	**42**	**106**	**42**	**211**	**0**

TABLE 2 – Ressources FPGA utilisées

2.3 Modèle 3 : Optimisation Hybride Post-Training

2.3.1 Configuration HLS

Ce modèle reprend l'architecture élaguée à 75%, mais pousse l'optimisation via la configuration de HLS4ML suite au profilage des poids. Nous avons forcé une précision plus agressive pour les poids et biais, passant à `ap_fixed<8,2>` (8 bits dont 2 entiers), tout en gardant une précision intermédiaire plus large pour éviter la saturation.

2.3.2 Performances

Cette configuration offre le meilleur compromis. L'utilisation de poids sur 8 bits réduit la largeur des bus de données et l'utilisation mémoire (BRAM/Logic), permettant au design de s'insérer confortablement dans l'Artix-7 avec une précision quasi-identique au modèle Keras.

2.4 Analyse de l'IP Vivado et des Pragmas

L'analyse du code C++ généré dans le répertoire `firmware` révèle l'utilisation de directives HLS (Pragmas) pour piloter la synthèse RTL :

— `#pragma HLS PIPELINE` : Cette directive est utilisée pour augmenter le débit (Throughput). Elle permet de traiter une nouvelle donnée à chaque cycle d'horloge (Initiation Interval = 1) en pipelinant les opérations.
— `#pragma HLS ALLOCATION` : Liée au paramètre `ReuseFactor`, cette directive limite le nombre d'instances physiques d'opérateurs (ex : limiteurs de DSP). Elle force le réutilisation séquentielle des ressources matérielles pour économiser la surface du FPGA.
— `#pragma HLS ARRAY_PARTITION` : Cette directive partitionne les tableaux de poids (stockés en BRAM ou LUTRAM) en registres individuels. Cela permet un accès parallèle simultané à toutes les données nécessaires pour alimenter le pipeline, évitant les goulots d'étranglement mémoire.

Le constat final sur les ressources montre que l'augmentation du `ReuseFactor` et l'application du Pruning sont indispensables pour faire passer un réseau dense sur un petit FPGA comme l'Artix-7. L'objectif étant de ne pas dépasser 70 % d'utilisation des blocs du FPGA.

3 Implémentation Matérielle et Architecture Système

L'intégration a été étudiée autour de l'IP fournie `model_nexys_pruned`. La simulation comportementale a été réalisée, et l'architecture globale nécessaire au déploiement système nous a été présentée par l'enseignant.

3.1 Simulation Comportementale

Un projet Vivado `Simulation_MonIA` a été créé ciblant la Nexys4 DDR. Pour valider l'IP, un module séquenceur VHDL nommé `master_IA` a été développé. Il pilote l'interface de contrôle de l'IP HLS (protocole `ap_ctrl_chain`) en générant les signaux de handshake (`ap_start`, `ap_done`) et en fournissant les stimuli. Les simulations confirment que l'IP respecte les délais de calcul et produit les sorties attendues.

3.2 Déploiement Physique et Debug (Démonstration ILA)

L'étape de déploiement physique consiste à charger le Bitstream sur le FPGA. La démonstration de l'enseignant s'est focalisée sur l'intérêt du débogage matériel via un cœur **ILA (Integrated Logic Analyzer)**.

L'ILA agit comme un oscilloscope numérique inséré directement dans la logique du FPGA. Nous avons pu voir comment configurer le déclenchement (Trigger) sur un événement physique (appui bouton) pour capturer les états internes du bus AXI et les signaux de contrôle de l'IP. Cela permet de vérifier que la logique séquentielle réagit correctement à la fréquence d'horloge réelle (100 MHz).

3.3 Architecture Système Complète (Microblaze)

Enfin, l'architecture nécessaire pour transformer le FPGA en un système de classification autonome connecté nous a été présentée. Faute de temps, nous n'avons pas assisté à l'exécution fonctionnelle (classification en temps réel), mais nous avons étudié le design matériel (*Block Design*) sous Vivado.

L'architecture repose sur un Système sur Puce (SoC) intégrant :

— **Le Cerveau (Microblaze) :** Un processeur soft-core instancié dans la logique programmable qui orchestre le système.
— **La Communication :** Une interface Ethernet gérée par le processeur pour recevoir les données (X, Y) depuis un PC.
— **L'Accélération (IP HLS) :** Notre réseau de neurones connecté au processeur via un bus **AXI4-Lite**.

L'étude de ce schéma montre comment le logiciel (code C sur Microblaze) décharge le calcul intensif vers le matériel (IP HLS) : le processeur écrit les données dans les registres de l'IP, lance le calcul, et lit le résultat via le bus AXI une fois le signal d'interruption levé.

4 Conclusion

Ce projet a permis d'explorer les étapes nécessaires pour porter un algorithme d'intelligence artificielle sur un système embarqué contraint.

L'étude théorique a mis en évidence que l'approche naïve (Modèle 1) est inadaptée aux cibles comme l'Artix-7, et que l'utilisation de la quantification (8 bits) et de l'élagage (75% de sparsité) est indispensable pour respecter les contraintes de ressources (DSP et LUTs).

La fin du projet nous a permis de valider la simulation de l'IP et d'appréhender l'architecture matérielle complexe requise pour un déploiement réel. L'étude du *Block Design* intégrant le processeur Microblaze et l'IP HLS a clarifié le rôle des interfaces (AXI, ILA) dans la conception d'un système intelligent complet sur FPGA, illustrant la complémentarité entre le traitement logiciel et l'accélération matérielle.