Telecom Saint-Étienne
FISE 3

Co-Design
– Rapport de TP –

Étude en Filière Ingénieur sous Statut Étudiant
28.01.2026

Bastien DESCOS

Table des Matières

I. Notebook exemple cybersécurité

1. Présentation du dataset

Le dataset que nous allons utiliser est le UNSW-NB15, ce dataset a initialement 2,540,044 entrées réparties en 4 fichiers principaux. Nous allons utiliser 175,341 enregistrements en tant que base de données d'entraînement ainsi que 82,332 en tant que base de données de test.
Il contient différent types d'attaque permettant d'entraîner le modèle pour reconnaître ces attaques:

- Fuzzers
- Analysis
- Backdoors
- DoS
- Exploits
- Generic
- Reconnaissance
- Shellcode
- Worms

En l'occurrence, la version que nous téléchargeons est une version pré-quantifiée ce qui nous permettra un temps de téléchargement beaucoup plus court.

La sortie du réseau est simplement un retour qui indique si l'entrée est suspicieuse ou si elle est seine grâce à une probabilité allant de 0 (l'entrée est sûre) à 1 (l'entrée est suspecte).

2. Entrainement MLP

2.a) Modèle

Le réseau est un MLP avec 593 entrées, 3 couches cachées de 64 neurones, et une couche de sortie de 1 neurone. On retrouve donc 1 sortie qui indique la probabilité que l'entrée soit une attaque ou non.
Un modèle MLP (Multi-Layer Perceptron) est un réseau de neurones artificiels composé de plusieurs couches de neurones entièrement connectées. Chaque neurone dans une couche est connecté à tous les neurones de la couche suivante.
Cela nous fait un total de $593\times64 + 64\times64 + 64\times64 + 64\times1 = 38{,}081$ poids.
Le modèle est entraîné avec une fonction de perte binaire cross-entropy et l'optimiseur Adam. L'entraînement est effectué sur 10 époques avec un batch size de 256.

2.b) Quantification

Nous utilisons une quantification pour nos poids ainsi que nos ReLu de 2 bits.

Il existe plusieurs types de quantification, nous utilisons ici une quantification binaire. Pour la quantification binaire, les poids prennent seulement 2 valeurs possibles: 0 et 1.

Pour la quantification, il exite plusieurs méthodes: QAT (quantization-aware training) et PTQ (post-training quantization). La première méthode consiste à quantifier durant l'entraînement,

la seconde consiste à quantifier après l'entraînement. La quantification utilisée dans notre projet est une quantification QAT.

2.c) FINN

Le framework FINN est un framework open-source développé par Xilinx pour la conception et le déploiement de réseaux de neurones quantifiés sur des FPGA. Il sert d'interface entre le ONNX et Vitis. Il nous permet donc de ne pas avoir à créer le HLS à la main et donc nous simplifie grandement la tâche.
FINN utilise dans notre cas une quantifiaction binaire pour les poids et les activations, ce qui permet de réduire considérablement la taille du modèle et d'améliorer la vitesse d'inférence sur le FPGA.
Le modèle quantifié est ensuite converti en une représentation compatible avec le matériel FPGA à l'aide de FINN, qui génère du code HDL (Hardware Description Language) pour l'implémentation sur le FPGA.
FINN offre également des outils pour l'optimisation du modèle, la génération de bitstreams pour le FPGA.

Pour l'importation dans FINN, on ajoute 7 zéros afin de passer d'un nb premier 593, à un nb facilement découpable (600): W˙new = np.pad (W˙orig, [(0,0), (0,7)]).

Pour que FINN fonctionne, il necessite une quantification binaire codée entre {-1, +1}, c'est pourquoi nous avons dû adapter notre modèle Brevitas pour qu'il corresponde à cette contrainte. Pour ce faire nous avons utilisé un wrapper Brevitas qui convertit les poids et les activations de {0, 1} à {-1, +1}.

FINN sort beaucoup de fichiers de sortie, certains peuvent être très imposant comme le bitfile, c'est pourquoi il y a des paramètres afin de gérer les fichiers de sortie:

- **ESTIMATE_REPORTS**: fourni les rapports des ressources attendues et des performances par couche et pour l'ensemble du réseau sans synthèse Vivado complète;
- **STITCHED_IP**: crée un design IP stream-in stream-out qui peut être intégré dans d'autres designs Vivado IPI ou RTL;
- **RTLSIM_PERFORMANCE**: utiliser PyVerilator pour effectuer un test de performance/latence du design STITCHED˙IP;
- **OOC_SYNTH**: exécuter une synthèse hors contexte (juste l'accélérateur lui-même, sans aucun système l'entourant) sur le design STITCHED˙IP pour obtenir les ressources FPGA post-synthèse et la fréquence d'horloge réalisable;
- **BITFILE**: intégrer l'accélérateur dans un shell pour produire un bitfile autonome;
- **PYNQ_DRIVER**: générer un pilote Python PYNQ qui peut être utilisé pour lancer l'accélérateur;
- **DEPLOYMENT_PACKAGE**: créer un dossier contenant les sorties BITFILE et PYNQ˙DRIVER, prêt à être copié vers la plateforme FPGA cible.
- **OUTPUT_DIR**: indique le dossier dans lequel l'ensemble des sorties du programmes seront écrites.
- **STEPS**: indique la liste des étapes prédefinie ou personnalisée que FINN va faire pour le build de l'accélérateur.

Pour le déploiement sur la carte nous aurons besoin du BITFILE ainsi que du PYNQ_DRIVER. L'ESTIMATE_REPORTS peut être utile en amont pour savoir les ressources et les performances de notre accélérateur.

3. Comparaison des performances

Ici je ne sais pas encore quoi mettre.

4. Estimations implémentations FINN

Ici je dois mettre les résultats d'estimations FINN, avec les ressources utilisées, la fréquence max, etc.

5. Synthèse Vivado

Ici je dois mettre les résultats de la synthèse Vivado, avec les ressources utilisées, la fréquence max, etc.

6. Carte

Ici je dois mettre ce que le prof envoi en sceenshot et aussi faire un équivalent entre FPGA et CPU/GPU.

II. Modification MLP pour MNIST

1. MNIST dans FINN

Le dataset MNIST est un ensemble de données utilisé pour la reconnaissance de chiffres manuscrits. Il contient 60,000 images d'entraînement et 10,000 images de test, chaque image étant une image en niveaux de gris de 28×28 pixels représentant un chiffre de 0 à 9.
Afin de nous servir de MNIST, nous allons utiliser à nouveau le framework FINN.

2. Impact de la quantification

Ici je dois mettre les résultats d'impact de la quantification sur MNIST / CIFFAR10. Afin de réaliser la quantification, nous allons utiliser Brevitas qui est une bibliothèque de quantification pour PyTorch. Pour ce faire, nous avons utilisé des couches de Brevitas pour remplacer les couches standards de PyTorch. Cela s'effectue comme suit:

```
import brevitas.nn as qnn
qnn.QuantConv2d(3, 6, kernel_size=5, bias=True, padding = 2, weight_bit_width=16)
```

Nous voyons dans cette ligne la définition d'une couche de convolution quantifiée avec des poids sur 16 bits (weight_bit_width=16). Le reste ne change pas par rapport à une couche de convolution standard de PyTorch.

Pour réaliser l'impact de la quantification, nous avons entraîné plusieurs modèles avec des poids et des activations de différentes tailles (2, 4, 8, 16 bits) et nous avons comparé les performances de ces modèles sur le dataset CIFFAR10. Nous avons également comparé ces modèles avec un modèle non quantifié (poids et activations en 32 bits flottants).
Pour les modèles utilisés nous avons utilisé un modèle simple de CNN avec 2 couches de convolution suivies de 2 couches fully connected type LeNet-5.
Nous avons également utilisé un modèle de MLP avec 3 couches soit 1 couche d'entrée (100

neurones), une couche cachée (50 neurones), et une couche de sortie (10 neurones).
Les métriques pour la mesure seront la précision (accuracy) ainsi que le temps d'inférence.

Modèle	CNN	
Nb quantification	5	6
Accuracy (%)	8	9
Avg Loss	10	11
Temps CPU (s)	12	13
Temps inférence (ms)	14	15
Nb Inférences/s	16	17

3. Performances estimées

Ici je dois mettre les résultats d'estimations FINN, avec les ressources utilisées, la fréquence max, etc. Ainsi que la synthèse Vivado avec les ressources utilisées, la fréquence max, etc.

III. Bibliographie

Liens vers les références utilisées pour la réalisation du rapport:

https://www.kaggle.com/code/mrwellsdavid/unsw-nb15-dataset-mlp-classifier/notebook
https://xilinx.github.io/brevitas/v0.12.1/tutorials/tvmcon2021.html
https://github.com/Xilinx/brevitas?tab=readme-ov-file
https://arxiv.org/pdf/2103.13630