Université de Bretagne Occidentale

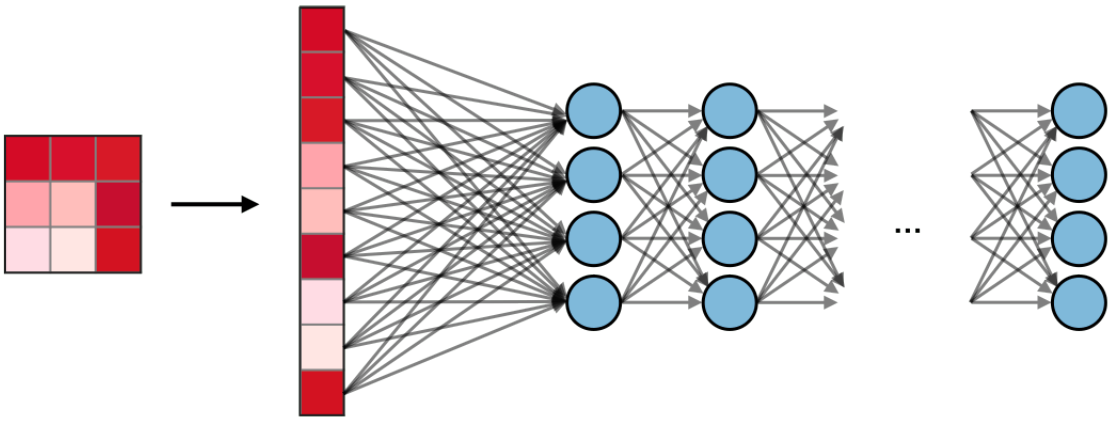
Faculté des Sciences & Techniques

Compte rendu de TAL n°4 Implémentation C de réseaux de neurones convolutifs sur systèmes embarqués

Ewann DELACRE

12 mai 2025

Professeur référent : Hai Nam Tran



1ère année de Master Informatique
Parcours Logiciels pour Systèmes Embarqués
Promotion 2025
Université de Bretagne Occidentale, Brest

Table des matières

1 Introduction

1.1 Du 5 au 9 mai

Depuis le dernier rapport, l'objectif principal est resté le même : effectuer une inférence d'image via un réseau de neurones convolutif (CNN) sur un environnement embarqué léger. Cependant, plusieurs pivots méthodologiques ont eu lieu. Après une tentative difficile d'intégration de TensorFlow Lite Micro (TFLM) dans Zephyr RTOS, nous avons repris le développement autour de TensorFlow C API en environnement desktop afin de stabiliser le modèle d'inférence. Ce rapport présente les étapes majeures de ce travail : finalisation d'une inférence C simple, puis son évolution vers un traitement réel d'image.

2 Retour sur TensorFlow Lite Micro et Zephyr

Avant de poursuivre avec TensorFlow pur, nous avons tenté d'intégrer un exemple complet basé sur le module tflite-micro/hello_world dans Zephyr, ciblant la carte Raspberry Pi Pico WH. Malgré une configuration correcte de west.yml et de l'environnement Zephyr SDK, l'intégration du module échouait systématiquement à la compilation, notamment à cause de l'absence de certains fichiers source (micro_string.cc) et de références symboliques manquantes (MicroInterpreterGraph, MicroAllocator, etc.). Ces difficultés, nombreuses et peu documentées, ont confirmé qu'une implémentation complète avec TFLM dans Zephyr nécessiterait un effort d'intégration avancé, difficile à justifier à ce stade du projet.

3 Inférence v2 : finalisation de l'inférence simple avec TensorFlow C

Une deuxième version du code tf_inference.c a été produite, cette fois intégralement construite autour de l'API C officielle de TensorFlow (hors Zephyr). Cette version fonctionne avec un jeu de poids prédéfini (3 filtres convolutifs 2x2) appliqués à une entrée 4x4. Le traitement suit toutes les étapes classiques :
- Placeholder d'entrée 4D [1, 4, 4, 1]

- Convolution 2D avec biais

- Activation ReLU

- Global Average Pooling

- Softmax et prédiction finale (argmax)

4 Inférence v3 : traitement d'une image réelle

Pour progresser vers un cas d'usage concret, une version img_tf_inference.c a été développée. Cette fois, le réseau doit prendre en paramètre une image d'entrée 28x28 (format MNIST). Le pipeline suit une logique identique à celle de l'inférence précédente, mais avec chargement dynamique de l'image.

Nous avons initialement exploré les bibliothèques stb_image et stb_image_resize pour effectuer la lecture d'image .png et la conversion en matrice de float. Cependant, leur taille importante (plus de 18k lignes cumulées) remet en question leur pertinence pour l'embarqué.

Un plan alternatif consiste à utiliser un format .raw ou .pgm (non compressé) plus simple à parser manuellement, sans dépendance externe. Dans cette optique, un convertisseur .png → .raw externe serait intégré dans la chaîne d'outils (hors embarqué), et le binaire analyserait ensuite directement les valeurs uint8_t (0 à 255) représentant les intensités de pixels.

5 Bilan et suite du projet

Le code d'inférence fonctionne désormais correctement dans deux contextes :

- Sur des données codées en dur (v2) embarqué ;

- À partir d'une image d'entrée réelle (v3) mais le résultat de prédiction est très mauvais.

Les prochaines étapes visent à :

- Corriger la phase d'entrainement pour que la prédiction soit correcte.

- Comparer (entre C pur et TF API) le temps d'exécution, la taille du fichier exécutable et la consommation de mémoire vive.