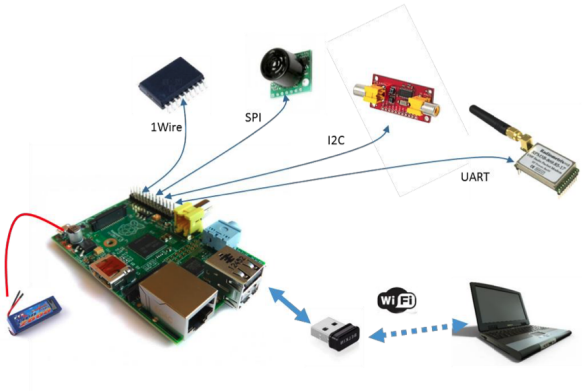
Cahier de Travaux Pratiques

Développement d'une intelligence artificielle légère sur un système embarqué



Frédéric CHATRIE
frederic.chatrie@bordeaux-inp.fr

Formation TSI

Résumé du projet

Ce projet a pour objectif de concevoir et déployer une application d'intelligence artificielle embarquée sur Raspberry Pi capable de détecter et reconnaître des chiffres manuscrits (0-9) en temps réel à partir d'une caméra. Le projet se décompose en **3 étapes principales** :

1. **Prise en main de Docker** : environnement conteneurisé Keras/TensorFlow
2. **Entraînement Python** : implémentation et comparaison MLP vs CNN sur MNIST
3. **Inférence C** : déploiement ultra-léger en C pur sur Raspberry Pi

Les étudiants compareront les performances des architectures MLP et CNN, puis implémenteront un moteur d'inférence en langage C pour un déploiement optimisé sur système embarqué.

Table des matières

1 Introduction

1.1 Contexte

Les systèmes embarqués sous Linux sont omniprésents dans les technologies modernes : véhicules autonomes, drones, robots industriels, smartphones, caméras intelligentes, objets connectés (IoT), etc. La convergence entre l'intelligence artificielle et les systèmes embarqués, connue sous le nom d'**Edge AI** ou **TinyML**, représente un enjeu technologique majeur.

L'exécution d'algorithmes d'IA directement sur des dispositifs embarqués présente de nombreux avantages :

— **Latence minimale** : traitement local sans dépendance réseau
— **Confidentialité** : les données sensibles restent sur l'appareil
— **Disponibilité** : fonctionnement même hors connexion
— **Coût réduit** : pas d'infrastructure cloud nécessaire
— **Scalabilité** : déploiement sur des milliers d'appareils sans surcharge serveur

La carte Raspberry Pi constitue un support d'apprentissage performant, très bon marché et disposant d'une forte communauté qui la place parmi les plus documentées. Elle possède des entrées/sorties puissantes permettant une connexion avec le monde physique par l'intermédiaire de capteurs (notamment une caméra) et d'actionneurs.

1.2 Objectifs pédagogiques

À l'issue de ce module, les étudiants seront capables de :

1. Maîtriser l'environnement Docker pour le développement reproductible
2. Concevoir et entraîner un **MLP** (Perceptron Multi-Couches) pour la classification
3. Concevoir et entraîner un **CNN** (Réseau de Neurones Convolutif) pour la classification d'images
4. **Comparer** les performances MLP vs CNN (précision, temps, taille)
5. Exporter les poids d'un modèle vers un format exploitable en C
6. **Implémenter un moteur d'inférence en langage C** (forward pass)
7. Développer une chaîne de traitement d'images robuste avec OpenCV
8. Déployer et tester une application d'IA temps réel sur Raspberry Pi
9. Documenter et présenter un projet technique de niveau professionnel

1.3 Description du projet

Objectif final du projet

Développer une application embarquée sur Raspberry Pi qui :

1. Capture en continu le flux vidéo de la caméra
2. Détecte automatiquement la présence de chiffres manuscrits dans le champ de vision
3. Reconnaît les chiffres (0 à 9) avec la plus grande précision possible
4. **Utilise un moteur d'inférence écrit en C** (pas Python/TFLite)
5. Affiche le résultat en temps réel et alerte l'utilisateur
6. Fonctionne à une grande cadence (grâce à l'inférence C)

Scénario d'utilisation : Un utilisateur présente un chiffre manuscrit (écrit sur papier) devant la caméra de la Raspberry Pi. L'application détecte automatiquement la présence du chiffre, le reconnaît, et informe l'utilisateur du résultat (via une API ou un message console).

2 Évaluation

2.1 Modalités d'évaluation

L'évaluation du projet se décompose comme suit :

Critère	%	**Description**
Rapport technique	60	Document PDF détaillé, structuré, avec résultats expérimentaux
Code source	40	Qualité, documentation, organisation du dépôt

TABLE 1 – Barème d'évaluation

2.2 Livrables attendus

1. **Rapport technique** (format PDF, 15-25 pages) comprenant :
   - Introduction et analyse du problème
   - Architecture MLP et CNN avec justification des choix
   - **Comparaison détaillée MLP vs CNN** (précision, temps, mémoire) avec une analyse statistique
   - Description de l'export des poids et du format utilisé
   - **Description de l'implémentation C** (structures, forward pass)
   - Description de la chaîne de prétraitement
   - Résultats expérimentaux (courbes d'apprentissage, matrices de confusion)
   - Guide de déploiement sur Raspberry Pi
   - Analyse critique et perspectives d'amélioration
2. **Code source** (dépôt) organisé comme suit :
   - `/docker/` : Dockerfile et scripts de configuration
   - `/training/` : scripts d'entraînement Keras | Tensorflow | PyTorch (MLP + CNN) et export des poids (.txt ou .h5)
   - `/models/` : modèles entraînés et poids exportés
   - `/inference_c/` : code C d'inférence (neural_network.c/.h, main.c)
   - `/utils/` : fonctions utilitaires (prétraitement, etc.)
   - `Makefile` : compilation du code C
   - `README.md` : documentation d'installation et d'utilisation
3. **Démonstration** : application fonctionnelle **en C** sur la Raspberry Pi lors de la restitution en dernière séance.

2.3 Critères de qualité

Critère	Insuffisant	Satisfaisant	Excellent
Précision MLP sur MNIST	$< 95\%$	95-97%	$> 97\%$
Précision CNN sur MNIST	$< 97\%$	97-99%	$> 99\%$
Précision MLP sur BDD personnel	$< 60\%$	60-80%	$> 80\%$
Précision CNN sur BDD personnel	$< 70\%$	70-85%	$> 85\%$
Précision avec caméra	$< 60\%$	60-80%	$> 80\%$
Temps inférence C (MLP)	> 50ms	20-50ms	< 20ms
Temps inférence C (CNN)	> 100ms	40-100ms	< 40ms
Code fonctionnel	Partiel	Complet	Optimisé

Table 2 – Critères de performance

3 Matériel et environnement

3.1 Raspberry Pi 5

3.1.1 Spécifications techniques

Composant	Spécification
Processeur	Broadcom BCM2712, Quad-core Cortex-A76 @ 2.4 GHz (64-bit)
RAM	16 Go LPDDR5-5500
GPU	VideoCore VII @ 750 MHz
Stockage	Carte microSD 64 Go (Classe 10)
Connectivité	WiFi 6 (802.11ax), Bluetooth 5.1, Gigabit Ethernet
USB	2 × USB 3.0, 2 × USB 2.0
Vidéo	2 × micro-HDMI (4Kp120), interface DSI, interface CSI
GPIO	40 pins (compatible HAT)
Alimentation	USB-C 5V/5A (25W)

Table 3 – Spécifications de la Raspberry Pi 5

3.1.2 Système d'exploitation

La carte est pré-configurée avec **Raspberry Pi OS (64 bits)** (anciennement Raspbian), une distribution basée sur Debian optimisée pour le matériel Raspberry Pi. Raspbian est livré avec plus de 45 000 paquets, c'est-à-dire des logiciels pré-compilés pour une installation facile via les gestionnaires de paquets.

3.2 Caméra Raspberry Pi Module V3

Caractéristique	Valeur
Capteur	Sony IMX708 (12 mégapixels)
Résolution photo	4056 × 3040 pixels
Résolution vidéo	4K @ 60fps, 1080p @ 120fps, 720p @ 240fps
Interface	CSI-2 (Camera Serial Interface)
Champ de vision	84° (diagonal)
Mise au point	Autofocus

Table 4 – Spécifications de la caméra Raspberry Pi Module V3

3.3 Connexion à distance

3.3.1 Protocole SSH

En général, les systèmes embarqués n'ont pas de moniteur, de clavier et de souris. Leur gestion se fait à distance par l'intermédiaire du protocole SSH (Secure Shell).

```
# Connexion SSH (remplacer <IP> par l'adresse de votre Pi)
ssh votre_login@<IP_RASPBERRY>

# Exemple
ssh dupont@192.168.1.42
```

3.3.2 Transfert de fichiers (SCP)

```
# Copier un fichier vers la Pi
scp votre_fichier votre_login@<IP>:/home/votre_login/projet/

# Copier un dossier entier (option -r)
scp -r mon_projet/ votre_login@<IP>:/home/votre_login/

# Copier depuis la Pi vers le PC
scp votre_login@<IP>:/home/votre_login/votre_fichier ./
```

4 Répartition des séances

4.1 Séance 1 : Prise en main et acquisition des données

4.1.1 Objectifs de la séance 1

- Se connecter à la Raspberry Pi et vérifier le bon fonctionnement de la caméra
- Comprendre le dataset MNIST et ses caractéristiques
- Mettre en place l'environnement Docker pour l'entraînement (plutôt sur machine locale)
- Création de la base de données personnalisée de 20 échantillons de chaque caractère manuscrit (0-9) à l'aide d'un logiciel équivalent à Paint

4.1.2 Prise en main de la Raspberry Pi

Connexion initiale Chaque groupe dispose d'une Raspberry Pi 5. Le login et le mot de passe sont à définir avec l'enseignant afin de créer un compte qui vous est attribué.

```
# Connexion
ssh -X votre_nom@<IP_FOURNIE>

# Verification de l'espace disque
df -h

# Verification de la temperature CPU
vcgencmd measure_temp
```

Test de la caméra Assurez-vous que la caméra est correctement connectée à la Raspberry Pi via le port CSI. Activez la caméra dans les paramètres de configuration si nécessaire.

```
# Verification que la camera est detectee
rpicam-hello --list-cameras

# Capture d'une image de test
rpicam-still -o test_capture.jpg

# Apercu video (5 secondes)
rpicam-hello --timeout 5000

# Capture video (10 secondes)
rpicam-vid -t 10000 -o test_video.h264
```

4.1.3 Le dataset MNIST

Présentation Le dataset MNIST (Modified National Institute of Standards and Technology) est le benchmark historique du deep learning pour la classification d'images. Il contient :

- **70 000 images** de chiffres manuscrits (0-9)
- **60 000** images d'entraînement + **10 000** images de test
- Format : images en niveaux de gris de **28 × 28 pixels**
- Valeurs des pixels : entiers de 0 (noir) à 255 (blanc)
- Chiffres centrés par calcul du centre de masse

4.1.4 Environnement Docker

Accès au conteneur Framework IA Un conteneur Docker avec PyTorchTensorFlow/Keras est disponible pour chaque groupe sous le nom de student-gpu.zip. Après avoir décompressé l'archive, vous pouvez démarrer le conteneur avec la commande suivante (à exécuter sur votre machine locale si vous avez une carte NVIDIA ou non, ou directement sur la Raspberry Pi) :

```
# Lister les conteneurs disponibles
docker ps -a

# Construire le conteneur (si necessaire)
docker build -t student_gpu_votre_nom .

# Acceder au conteneur
./student.sh

# Une fois dans le conteneur, verifier TensorFlow
python3 -c "import tensorflow as tf; print(tf.__version__)"
```

4.1.5 Gestion des fichiers avec Docker

```
# Copier un fichier depuis l'hote vers le conteneur
docker cp script.py student_gpu_votre_nom:/workspace/

# Copier depuis le conteneur vers l'hote
docker cp student_gpu_votre_nom:/workspace/model ./
```

4.1.6 Travail à réaliser - Séance 1

1. **Connexion et vérification**
   - Se connecter à la Raspberry Pi 5
   - Tester la caméra et capturer quelques images
   - Accéder au conteneur Docker et vérifier l'installation de TensorFlow
2. **Exploration de MNIST**
   - Charger et visualiser le dataset
   - Analyser la distribution des classes
   - Calculer des statistiques (moyenne, écart-type des pixels)
3. **Acquisition de données personnelles**
   - Écrire les chiffres 0-9 sur Paint (20 exemplaires de chaque)
   - Capturer ces chiffres et les stocker dans un dossier `data/custom_digits/`
   - Créer un script de prétraitement pour les convertir au format MNIST (28×28, niveaux de gris, ...)
   - Vérifier la qualité des images prétraitées (Format bmp préconisé, pour simplifier la lecture en C)
4. **Documentation**
   - Commencer le rapport : introduction, description du matériel
   - Documenter les observations sur MNIST

4.2 Séance 2 : Entraînement des modèles MLP et CNN

4.2.1 Objectifs de la séance 2

- Implémenter un MLP (Perceptron Multi-Couches)
- Implémenter un CNN (Réseau de Neurones Convolutif)
- Comparer les performances des deux architectures
- Analyser les résultats et comprendre les différences à la fois sur MNIST et sur la base de données personnalisée

4.2.2 Travail à réaliser - Séance 2

1. **Implémentation du MLP**
   - Implémenter l'architecture MLP
   - Entraîner sur MNIST et BDD personnelle et comparer les résultats
   - Valider la précision (>95% sur MNIST)
   - Sauvegarder le modèle et noter le nombre de paramètres
2. **Implémentation du CNN**
   - Implémenter l'architecture CNN
   - Entraîner sur MNIST et BDD personnelle et comparer les résultats
   - Valider la précision (>97% sur MNIST)
   - Sauvegarder le modèle
3. **Data Augmentation**
   - Appliquer la data augmentation aux deux modèles
   - Comparer les performances avec/sans augmentation

4. **Comparaison MLP vs CNN**
   — Créer un tableau comparatif (précision, temps entraînement, taille)
   — Analyser les matrices de confusion des deux modèles
   — Identifier les forces et faiblesses de chaque architecture
   — Documenter les résultats dans le rapport

4.3 Séance 3 : Export des poids et implémentation C

4.3.1 Objectifs de la séance 3

— Exporter les poids des modèles dans un format exploitable en C
— Implémentation du forward pass en C
— Implémenter le MLP en C (plus simple)
— Préparer l'implémentation du CNN en C

4.3.2 Travail à réaliser - Séance 3

1. **Export des poids**
   — Implémenter les scripts d'export
   — Exporter les poids du MLP entraîné (.txt (plus simple pour l'implémentation C) ou .h5)
   — Vérifier la cohérence des fichiers générés
2. **Implémentation C du MLP**
   — Implémenter les structures et fonctions C
   — Compiler et tester sur PC d'abord
   — Valider que les prédictions sont identiques à Python
3. **Cross-compilation pour RPi**
   — Compiler le code pour architecture ARM
   — Transférer sur la Raspberry Pi
   — Mesurer le temps d'inférence
4. **Benchmark**
   — Comparer temps inférence MLP en C vs Python
   — Documenter les gains de performance
   — Valider les prédictions sur le jeu de test MNIST et la BDD personnelle

4.4 Séance 4 : Application temps réel sur Raspberry Pi

4.4.1 Objectifs de la séance 4

— Développer la chaîne de prétraitement d'images en C/OpenCV
— Intégrer l'inférence MLP/CNN dans l'application caméra
— Tester et optimiser le système complet
— Préparer la démonstration finale

4.4.2 Travail à réaliser - Séance 4

1. **Intégration application C**
   - Compiler l'application complète sur la Raspberry Pi
   - Intégrer le prétraitement avec l'inférence MLP
   - Tester avec des chiffres manuscrits écrits sur papier
2. **Implémentation CNN en C** (optionnel, réellement apprécié)
   - Ajouter les fonctions de convolution et pooling
   - Exporter et intégrer les poids du CNN
   - Comparer performances MLP vs CNN en C
3. **Tests et optimisations**
   - Mesurer les FPS et temps d'inférence réels
   - Ajuster les seuils de binarisation selon l'éclairage
   - Documenter les performances finales
4. **Préparation de la démonstration**
   - Préparer différents cas de test
   - Corriger les derniers bugs
   - Finaliser le rapport

5 Annexes

5.1 Installation des dépendances sur Raspberry Pi

```
# Mise a jour du systeme
sudo apt update && sudo apt upgrade -y

# Installation des outils de compilation
sudo apt install -y build-essential cmake

# Installation d'OpenCV pour C++
sudo apt install -y libopencv-dev

# Verification OpenCV
pkg-config --modversion opencv4

# Installation de Python (pour l'entrainement)
sudo apt install -y python3-pip python3-opencv
pip3 install torch numpy matplotlib
```

5.2 Structure recommandée du projet

```
projet_mnist/
|-- README.md
|-- Makefile
|-- docker/
|   |-- Dockerfile
|   |-- docker-compose.yml
|-- training/
|   |-- train_mlp.py
|   |-- train_cnn.py
```

```
|   |-- compare_models.py
|   |-- export_weights.py
|-- models/
|   |-- mlp_model.txt
|   |-- cnn_model.txt
|-- inference_c/
|   |-- neural_network.h
|   |-- neural_network.c
|   |-- model_weights.h
|   |-- main.cpp
|   |-- Makefile
|-- utils/
|   |-- visualization.py
|   |-- benchmark.py
|-- docs/
|   |-- rapport.pdf
|   |-- comparaison_mlp_cnn.md
|-- data/
    |-- custom_digits/
```