



# Projet de Développement d'une intelligence artificielle légère sur un système embarqué

Option TSI, Filière Électronique  
Année universitaire 2025–2026

**Binôme :**

Amer SAIDI  
Bilal HADDOU

**Encadrant :**

Frederic CHATRIE

ENSEIRB-MATMECA  
Bordeaux INP

# Table des matières

<b>1 Présentation générale du projet</b>	<b>3</b>
1.1 Cahier des charges . . . . .	3
1.2 Architecture globale du système . . . . .	3
<b>2 Données et préparation</b>	<b>4</b>
2.1 Base de données MNIST . . . . .	4
2.2 Base de données personnalisée . . . . .	4
<b>3 Modèles de reconnaissance</b>	<b>5</b>
3.1 Perceptron multicouche (MLP) . . . . .	5
3.2 Réseau de neurones convolutionnel (CNN) . . . . .	5
3.3 Impact du prétraitement des données sur les performances des modèles . . . . .	6
<b>4 Entraînement des modèles</b>	<b>7</b>
4.1 Principe général de l'entraînement . . . . .	7
4.2 Entraînement et test du MLP . . . . .	8
4.3 Entraînement et test du CNN . . . . .	9
<b>5 Comparaison MLP et CNN</b>	<b>11</b>
<b>6 Implémentation embarquée</b>	<b>12</b>
6.1 Export des poids du modèle . . . . .	12
6.2 Implémentation du réseau en C . . . . .	13
6.2.1 MLP . . . . .	13
6.2.2 CNN . . . . .	13
6.3 Validation locale du MLP avant déploiement embarqué . . . . .	14
6.4 Validation locale du CNN avant déploiement embarqué . . . . .	14
6.5 Validation de l'implémentation embarquée . . . . .	15
6.6 Prétraitement des images sur Raspberry Pi . . . . .	15
6.7 Intégration du flux vidéo et inférence en temps réel . . . . .	16
6.8 Environnement logiciel . . . . .	16
<b>7 Mesures de performances</b>	<b>16</b>
7.1 Temps d'inférence du MLP . . . . .	16
7.2 Temps d'inférence du CNN . . . . .	17
7.3 Performances en temps réel (FPS) . . . . .	18
7.4 Validation visuelle des résultats . . . . .	18
<b>8 Analyse critique et perspectives d'amélioration</b>	<b>19</b>
<b>9 Conclusion</b>	<b>20</b>

## Introduction

La reconnaissance automatique de chiffres manuscrits constitue un problème fondamental en intelligence artificielle et en apprentissage automatique. Elle est largement étudiée en raison de ses nombreuses applications concrètes, telles que la lecture automatique de formulaires, la reconnaissance de codes postaux, la numérisation de documents manuscrits ou encore les interfaces homme-machine. Ce sujet représente également un excellent cas d'étude pour l'implémentation et l'évaluation de réseaux de neurones artificiels.

L'objectif de ce projet est de concevoir et d'implémenter un système complet de reconnaissance de chiffres manuscrits sur une plateforme embarquée de type Raspberry Pi, équipée d'une caméra. Le projet couvre l'ensemble de la chaîne de traitement, depuis l'acquisition d'images en conditions réelles jusqu'à la prédiction finale du chiffre reconnu. Cette approche impose des contraintes spécifiques liées à l'environnement embarqué, notamment en termes de ressources mémoire, de puissance de calcul et de temps d'exécution.

Dans un premier temps, des modèles de reconnaissance ont été développés et entraînés à partir de la base de données MNIST, qui constitue une référence dans le domaine de la reconnaissance de chiffres manuscrits. Deux architectures de réseaux de neurones ont été étudiées : un perceptron multicouche (MLP) et un réseau de neurones convolutif (CNN). Ces modèles ont permis d'analyser les compromis entre complexité, précision et coût computationnel. Les performances ont ensuite été évaluées sur une base de données personnalisée, construite à partir d'images réelles, afin d'étudier la capacité de généralisation des modèles.

Une attention particulière a été portée à l'implémentation embarquée du perceptron multicouche. Les poids du modèle entraîné ont été exportés et intégrés directement dans un programme en langage C, permettant une inférence autonome sur la Raspberry Pi sans dépendance à un framework d'apprentissage. Le prétraitement des images issues de la caméra a quant à lui été réalisé à l'aide de la bibliothèque OpenCV en C++, afin de garantir une gestion robuste des formats d'images et une cohérence avec les données utilisées lors de l'entraînement.

Ce rapport présente de manière détaillée les différentes étapes du projet, depuis la préparation des données et la conception des modèles jusqu'à leur intégration sur une plateforme embarquée. Il met également en évidence les résultats obtenus, les limites rencontrées en conditions réelles, ainsi que les perspectives d'amélioration possibles.

# 1 Présentation générale du projet

## 1.1 Cahier des charges

Le projet a pour objectif principal de concevoir un système capable de reconnaître automatiquement des chiffres manuscrits à partir d'images acquises par une caméra, puis d'exécuter cette reconnaissance sur une plateforme embarquée de type Raspberry Pi. Le système doit être capable de fonctionner de manière autonome, avec un temps de réponse compatible avec une utilisation en temps réel.

Les principales fonctionnalités attendues sont les suivantes :

- Acquisition d'images de chiffres manuscrits à l'aide d'une caméra connectée à la Raspberry Pi.
- Prétraitement des images afin de les rendre compatibles avec les données utilisées lors de l'entraînement des modèles.
- Reconnaissance automatique du chiffre présent sur l'image à l'aide d'un réseau de neurones.
- Affichage du chiffre prédict par le système.

Le projet est soumis à plusieurs contraintes. D'un point de vue matériel, la Raspberry Pi dispose de ressources limitées en termes de mémoire et de puissance de calcul, ce qui impose de limiter la complexité des modèles utilisés. D'un point de vue logiciel, l'inférence du modèle doit être réalisée sans dépendre de bibliothèques lourdes de type frameworks d'apprentissage automatique, afin de garantir la portabilité et la légèreté de l'implémentation embarquée.

## 1.2 Architecture globale du système

Le système développé repose sur une architecture modulaire, organisée en plusieurs étapes successives formant une chaîne de traitement complète. Cette architecture permet de séparer clairement les différentes responsabilités du système et de faciliter le débogage ainsi que les évolutions futures.

Le pipeline global peut être résumé comme suit :

**Acquisition de l'image** Une image est capturée à l'aide de la caméra connectée à la Raspberry Pi. Cette image peut être issue d'un flux vidéo ou d'une capture ponctuelle.

**Prétraitement de l'image** L'image acquise est traitée afin d'extraire le chiffre manuscrit et de le mettre sous une forme normalisée. Cette étape comprend notamment la conversion en niveaux de gris, la binarisation, le recadrage autour du chiffre, le redimensionnement en  $28 \times 28$  pixels et la normalisation des valeurs des pixels.

**Inférence du réseau de neurones** Le vecteur d'entrée issu du prétraitement est transmis au réseau de neurones embarqué. Celui-ci calcule les activations successives des couches du modèle et produit un vecteur de sortie correspondant aux probabilités associées à chaque chiffre possible.

**Décision et affichage du résultat** Le chiffre reconnu est déterminé à partir de la sortie du réseau (classe ayant la valeur maximale) et affiché à l'utilisateur.

Cette architecture a été conçue de manière à isoler le prétraitement image du cœur du modèle de reconnaissance. Le prétraitement est réalisé à l'aide de la bibliothèque OpenCV en C++, tandis que le réseau de neurones est implémenté en langage C afin de garantir une exécution efficace et portable sur la plateforme embarquée.

## 2 Données et préparation

### 2.1 Base de données MNIST

La base de données MNIST (*Modified National Institute of Standards and Technology*) est une base de référence largement utilisée pour l'évaluation des algorithmes de reconnaissance de chiffres manuscrits. Elle est composée de 70 000 images de chiffres manuscrits, réparties en 60 000 images d'entraînement et 10 000 images de test. Chaque image représente un chiffre compris entre 0 et 9 et est stockée sous la forme d'une image en niveaux de gris de taille  $28 \times 28$  pixels.

Les valeurs des pixels sont comprises entre 0 et 255, où 0 correspond à un pixel noir et 255 à un pixel blanc. Cette base présente l'avantage d'être normalisée et relativement propre, avec des chiffres bien centrés et peu de bruit. MNIST constitue ainsi un point de départ idéal pour l'entraînement et la comparaison de différents modèles de réseaux de neurones.

Dans le cadre de ce projet, la base MNIST a été utilisée pour entraîner et évaluer les modèles de reconnaissance, notamment le perceptron multicouche et le réseau de neurones convolutif. Elle a également servi de référence pour mesurer les performances des modèles en termes de précision.

### 2.2 Base de données personnalisée

En complément de la base MNIST, une base de données personnalisée a été créée afin de rapprocher le projet d'un cas d'usage réel. Pour chaque chiffre de 0 à 9, le même chiffre a été écrit vingt fois sur une seule page numérique. Une capture d'écran de cette page a ensuite été réalisée, fournissant une image contenant l'ensemble des occurrences du chiffre.

À partir de cette image, chaque chiffre manuscrit a été automatiquement extrait puis traité à l'aide d'un programme python afin d'obtenir des images individuelles. Les chiffres ont été isolés, nettoyés et redimensionnés pour correspondre au format de la base MNIST, à savoir des images en niveaux de gris de taille  $28 \times 28$  pixels avec un chiffre centré sur fond noir.

Cette méthode a permis de constituer une base de données personnalisée de 200 images, soit vingt images par chiffre. Bien que cette base soit de taille limitée, elle permet d'évaluer la capacité des modèles à généraliser sur des données non standardisées et reflétant davantage les conditions réelles d'acquisition, notamment dans un contexte embarqué.

### 3 Modèles de reconnaissance

Dans ce projet, deux architectures de réseaux de neurones ont été étudiées pour la reconnaissance de chiffres manuscrits : un perceptron multicouche (MLP) et un réseau de neurones convolutionnel (CNN). Ces modèles ont été choisis afin de comparer une approche simple basée sur des couches entièrement connectées à une approche plus adaptée au traitement d'images.

Le MLP traite l'image sous forme de vecteur et présente une implémentation simple ainsi qu'un faible coût de calcul, ce qui le rend adapté aux systèmes embarqués. En revanche, il ne tient pas compte de la structure spatiale des images, ce qui peut limiter ses performances. Le CNN, quant à lui, exploite des couches de convolution pour extraire automatiquement des caractéristiques visuelles, offrant généralement de meilleures performances au prix d'une complexité plus élevée.

Les deux modèles ont été entraînés et évalués sur la base MNIST ainsi que sur une base de données personnalisée, afin d'analyser leurs performances et leur capacité de généralisation dans un contexte embarqué.

#### 3.1 Perceptron multicouche (MLP)

Le perceptron multicouche (MLP) est un réseau de neurones artificiels composé de plusieurs couches entièrement connectées. Dans ce projet, l'image d'entrée de taille  $28 \times 28$  pixels est d'abord aplatie afin de former un vecteur de 784 valeurs, correspondant aux niveaux de gris normalisés de l'image.

L'architecture du MLP utilisée comporte une couche d'entrée, deux couches cachées et une couche de sortie. Les couches cachées utilisent la fonction d'activation ReLU, tandis que la couche de sortie comporte dix neurones, correspondant aux dix chiffres possibles de 0 à 9. La sortie du réseau représente les scores associés à chaque classe, la classe prédite étant celle dont le score est maximal.

#### 3.2 Réseau de neurones convolutionnel (CNN)

Un réseau de neurones convolutionnel (Convolutional Neural Network, CNN) est un modèle de deep learning particulièrement adapté au traitement des images. Contrairement aux réseaux entièrement connectés (MLP), un CNN exploite la structure spatiale des images en appliquant des opérations de convolution locales. Chaque couche convolutionnelle applique un ou plusieurs filtres (ou noyaux) qui parcourrent l'image afin d'extraire des caractéristiques pertinentes telles que les contours, les formes ou les motifs locaux.

Mathématiquement, une couche de convolution réalise une opération de la forme :

$$y(i, j) = \sum_{m,n} x(i + m, j + n) w(m, n) + b,$$

où  $x$  représente l'image d'entrée,  $w$  le filtre de convolution et  $b$  un biais. L'application successive de couches convolutionnelles permet d'apprendre des représentations de plus en plus abstraites de l'image.

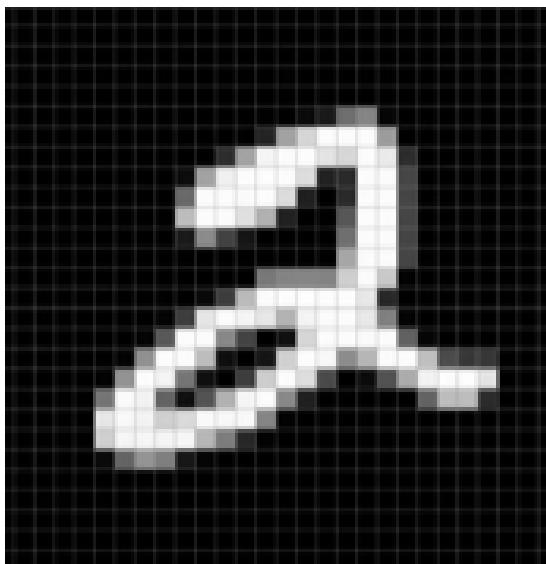
Dans ce projet, lors de l'implémentation du CNN en Python (PyTorch) et en C++, un choix volontaire a été fait de n'utiliser qu'un seul canal de sortie par couche convolutionnelle. Cela signifie que chaque couche applique un seul filtre de convolution. Ce choix simplifie fortement l'architecture du réseau et permet de rendre l'implémentation en C plus directe et maîtrisable, sans dépendre de bibliothèques spécialisées.

En effet, avec un seul canal de sortie, les opérations de convolution se réduisent à des calculs matriciels simples, ce qui facilite la lecture, la validation et le déploiement du modèle sur une plateforme embarquée. Cette approche permet de conserver le principe fondamental des réseaux convolutionnels tout en assurant une implémentation efficace et compréhensible dans un contexte embarqué.

### 3.3 Impact du prétraitement des données sur les performances des modèles

Les premiers modèles de reconnaissance ont été entraînés directement à partir de la base de données MNIST, sans modification du prétraitement initial. Bien que ces modèles aient obtenu de bonnes performances lors des tests sur MNIST, leur déploiement sur la plateforme embarquée Raspberry Pi a révélé une précision de reconnaissance très faible lors de l'utilisation d'images issues de la caméra.

Cette dégradation des performances s'explique par une différence notable entre les images de la base MNIST et celles prétraitées sur la Raspberry Pi. Comme illustré à la Figure 1, les chiffres de MNIST sont généralement centrés et n'occupent pas toute l'image de taille  $28 \times 28$  pixels. En revanche, le prétraitement appliqué aux images acquises par la caméra, implémenté en C++ avec OpenCV, produit des chiffres fortement recadrés, occupant presque toute la surface de l'image.



(a) Image issue de la base MNIST



(b) Image après prétraitement sur Raspberry Pi

FIGURE 1 – Comparaison entre une image MNIST et une image prétraitée issue de la caméra

Afin de réduire cet écart entre les données d'entraînement et les données utilisées en conditions réelles, le prétraitement développé pour la Raspberry Pi a été appliqué à l'ensemble de la base

MNIST. Les images ainsi obtenues reproduisent fidèlement le format et la représentation des chiffres observés lors de l'inférence embarquée.

Les modèles ont ensuite été réentraînés sur cette version modifiée de MNIST. Les résultats obtenus ont montré une amélioration très significative des performances sur la Raspberry Pi, avec un taux de reconnaissance des chiffres très élevé. Cette expérience met en évidence l'importance essentielle de l'adéquation entre le prétraitement des données d'entraînement et celui appliqué en phase d'inférence, en particulier dans un contexte embarqué.

## 4 Entraînement des modèles

### 4.1 Principe général de l'entraînement

L'entraînement des modèles a été conçu en tenant compte de leur déploiement final sur une plateforme embarquée. L'objectif n'est pas uniquement d'obtenir de bonnes performances en environnement de développement, mais surtout d'assurer un comportement robuste et précis lors de l'exécution en temps réel sur la carte embarquée.

Dans le système embarqué, l'extraction du chiffre à partir du flux caméra repose sur une chaîne de prétraitement d'image spécifique. Cette chaîne comprend une réduction du bruit par filtrage gaussien de taille  $3 \times 3$ , suivie d'un seuillage binaire inverse. Les contours sont ensuite extraits, et seul le contour correspondant à l'aire maximale est conservé, celui-ci étant supposé représenter le chiffre d'intérêt. La région correspondante est alors recadrée, centrée dans une image carrée, redimensionnée à une taille de  $28 \times 28$  pixels, puis normalisée avant d'être fournie au modèle de reconnaissance.

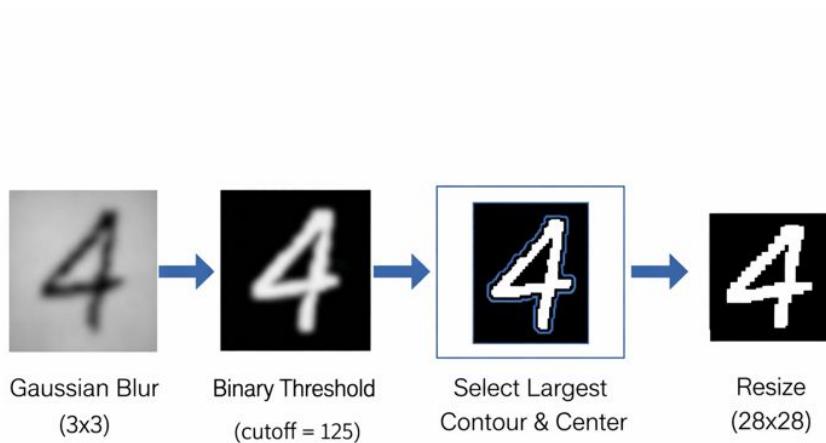


FIGURE 2 – prétraitement utilisé

Afin de garantir la cohérence entre la phase d'entraînement et l'exécution embarquée, exactement

le même prétraitement a été appliqué aux images de la base de données MNIST ainsi qu'à notre jeu de données personnalisé. Cette homogénéité entre les données d'apprentissage et les données traitées en temps réel permet d'améliorer la capacité de généralisation des modèles et d'obtenir de bonnes performances lors du déploiement embarqué.

L'optimisation des paramètres des modèles est réalisée à l'aide de l'algorithme Adam, choisi pour sa rapidité de convergence et sa robustesse. La fonction de coût utilisée est l'entropie croisée catégorielle (*Cross Entropy Loss*), adaptée aux problèmes de classification multi-classes tels que la reconnaissance de chiffres.

## 4.2 Entraînement et test du MLP

Dans un premier temps, le réseau de neurones entièrement connecté (MLP) est entraîné uniquement sur la base de données MNIST. Toutes les images subissent le même prétraitement que celui utilisé dans le système embarqué, incluant la réduction de bruit par filtrage gaussien, le seuillage binaire inverse, l'extraction du contour principal, le recadrage dans une image carrée, le redimensionnement en  $28 \times 28$  pixels et la normalisation. Ce choix garantit une cohérence entre la phase d'apprentissage et l'exécution en conditions réelles sur la carte embarquée.

La Figure 3 présente, côté à côté, l'évolution de la fonction de coût lors de l'entraînement ainsi que la matrice de confusion obtenue sur la base de test MNIST. On observe une diminution rapide de la loss durant les premières époques, suivie d'une convergence progressive, indiquant une bonne optimisation des paramètres du modèle. La matrice de confusion est fortement diagonale, ce qui montre que la majorité des chiffres sont correctement classifiés. Les erreurs restantes concernent principalement des chiffres présentant des similarités visuelles. L'accuracy obtenue sur MNIST est de **95.11%**.

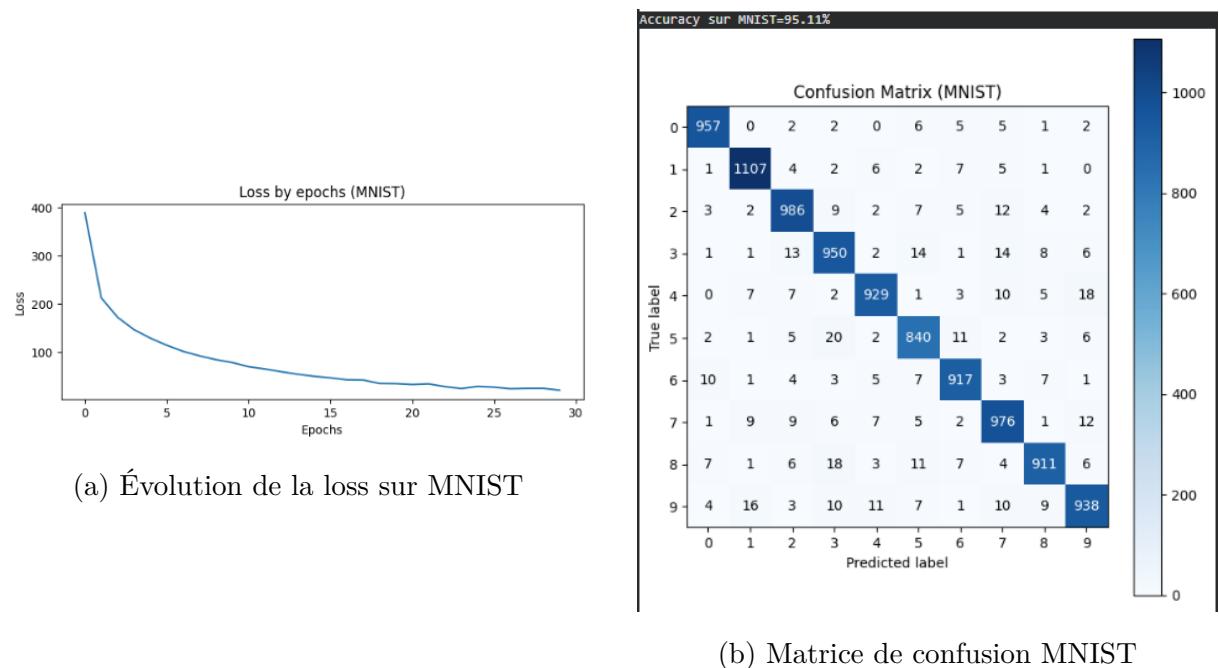


FIGURE 3 – Résultats du MLP sur la base MNIST : courbe de loss et matrice de confusion (accuracy = 95.11%).

Dans un second temps, le MLP est entraîné et évalué sur notre jeu de données personnalisé. Les données sont séparées en deux parties distinctes : 50% pour l'entraînement et 50% pour le test. Le même prétraitement que pour MNIST est appliqué, afin de garantir une cohérence entre les données d'apprentissage et celles traitées en conditions embarquées.

La Figure 4 présente la matrice de confusion obtenue sur le jeu de test de notre dataset. La diagonale reste dominante, ce qui montre que le modèle parvient à généraliser correctement sur des données acquises en conditions réelles. L'accuracy obtenue sur notre dataset est d'environ 85%, ce qui est satisfaisant compte tenu de la taille plus réduite du jeu de données et de la variabilité des acquisitions.

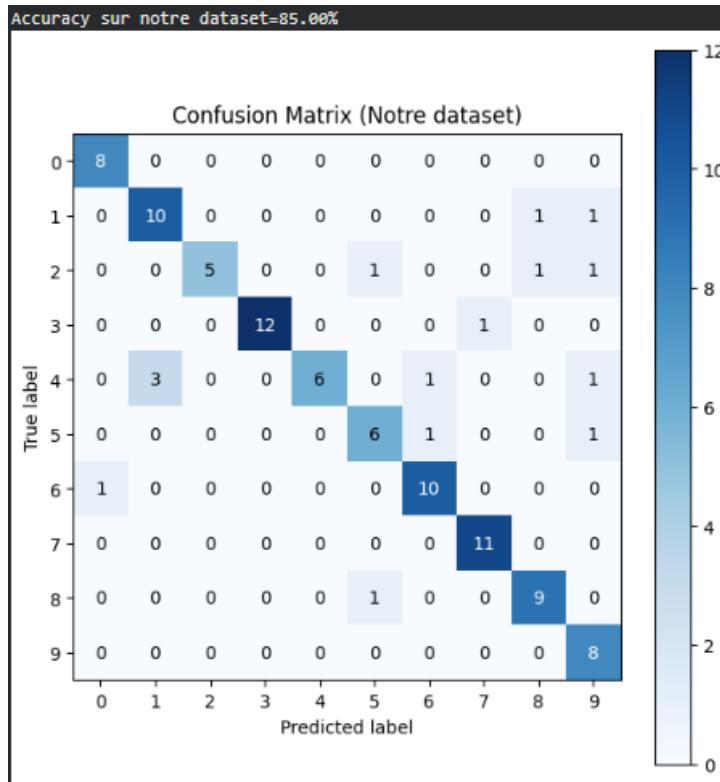


FIGURE 4 – Matrice de confusion du MLP sur notre dataset (accuracy  $\approx 85\%$ ).

### 4.3 Entrainement et test du CNN

Le réseau de neurones convolutionnel (CNN) est entraîné selon le même principe général que le MLP, en appliquant strictement le même prétraitement aux images de la base MNIST et à notre jeu de données personnalisé. Cette cohérence garantit que le modèle apprend à partir de données représentatives de celles traitées ultérieurement en temps réel sur la plateforme embarquée.

Contrairement à une architecture CNN classique, un choix volontaire a été fait de limiter le nombre de canaux de sortie à un seul par couche convolutionnelle, ce qui revient à appliquer un unique filtre à chaque couche. Ce choix a été motivé par la volonté de simplifier l'implémentation en C et de rendre le modèle directement embarquable sans dépendre de bibliothèques optimisées. En contrepartie, la capacité de représentation du modèle est réduite, ce qui nécessite un entraînement sur un nombre d'époques plus élevé afin d'atteindre des performances satisfaisantes.

La Figure 5 présente, côté à côté, l'évolution de la fonction de coût lors de l'entraînement du CNN sur MNIST ainsi que la matrice de confusion obtenue sur le jeu de test. La loss diminue progressivement au fil des époques, indiquant une convergence stable malgré une capacité limitée du modèle. La matrice de confusion est majoritairement diagonale, ce qui montre que le CNN parvient à distinguer correctement la majorité des chiffres. L'accuracy obtenue sur la base MNIST est de **91.22%**.

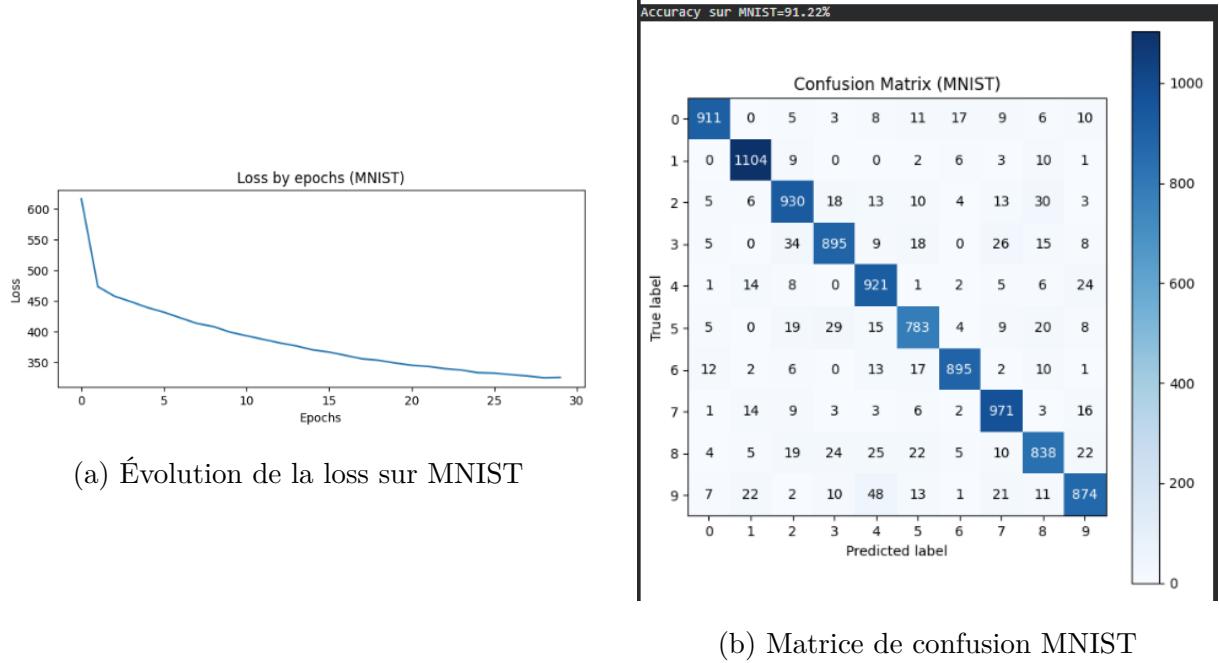


FIGURE 5 – Résultats du CNN sur la base MNIST : courbe de loss et matrice de confusion (accuracy = 91.22%).

Dans un second temps, le CNN est évalué sur notre jeu de données personnalisé. Les données sont séparées en deux ensembles distincts, avec 50% des images utilisées pour l'entraînement et 50% pour le test. Le même prétraitement que pour MNIST est appliqué, assurant ainsi une cohérence entre l'apprentissage et l'exécution embarquée.

La Figure 6 présente la matrice de confusion obtenue sur le jeu de test de notre dataset. Bien que les performances soient inférieures à celles obtenues sur MNIST, la diagonale reste dominante, montrant que le modèle généralise correctement malgré la taille réduite du jeu de données et la variabilité des acquisitions. L'accuracy obtenue sur notre dataset est d'environ **74%**, ce qui reste satisfaisant compte tenu de la simplicité de l'architecture convolutionnelle retenue.

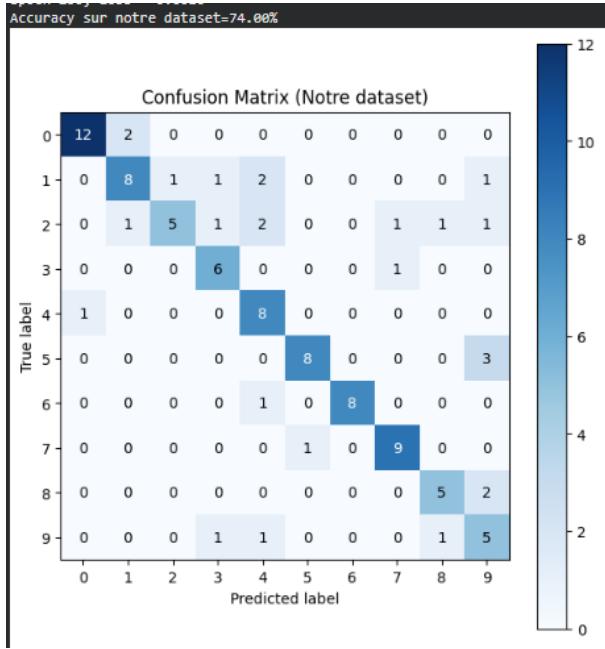


FIGURE 6 – Matrice de confusion du CNN sur notre dataset (accuracy  $\approx 74\%$ , séparation entraînement/test 50/50).

## 5 Comparaison MLP et CNN

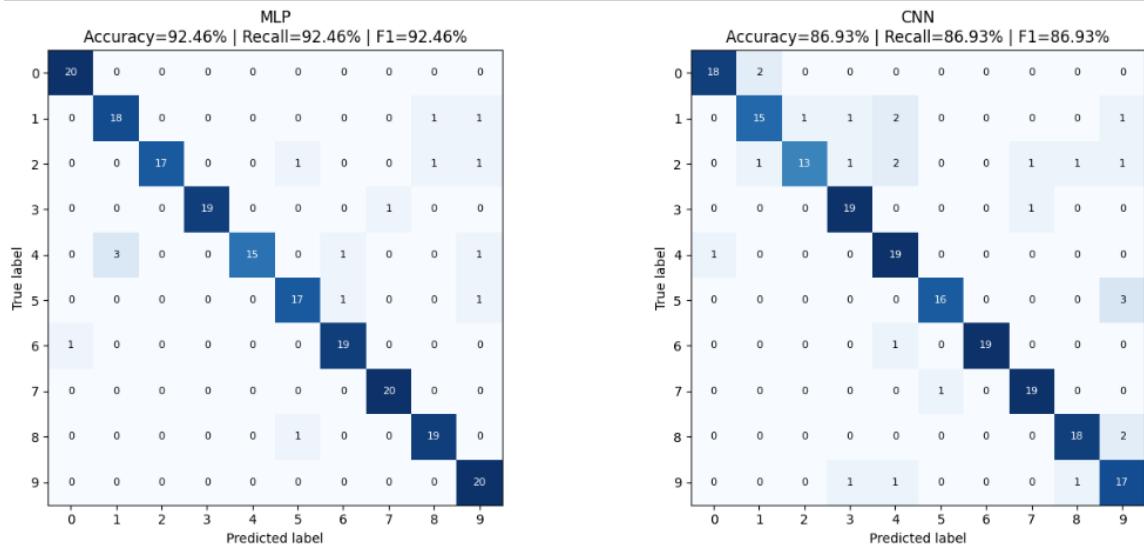


FIGURE 7 – Comparaison des matrices de confusion du MLP et du CNN évalués sur l'ensemble de notre dataset.

La Figure 7 présente une comparaison des performances du MLP et du CNN sur l'ensemble de notre jeu de données. Dans ce type de tâche de reconnaissance d'images, les réseaux de neurones convolutionnels sont en général plus performants que les réseaux entièrement connectés, car ils exploitent la structure spatiale des images et apprennent des caractéristiques locales pertinentes. Cependant, ce comportement théorique n'est pas observé dans notre cas expérimental.

Le MLP obtient une accuracy de **92.46%**, supérieure à celle du CNN (**86.93%**), ce qui indique

une meilleure robustesse dans le contexte de données limitées et de prétraitement embarqué. Cette différence de performance s’explique principalement par le choix volontaire d’une architecture convolutionnelle très simplifiée, dans laquelle chaque couche ne possède qu’un seul canal de sortie, c’est-à-dire un unique filtre de convolution. Cette contrainte, imposée afin de faciliter l’implémentation en C et le déploiement sur la plateforme embarquée, limite fortement la capacité de représentation du CNN.

Ainsi, malgré l’avantage théorique des CNN pour les tâches de vision par ordinateur, le MLP s’avère plus adapté dans notre contexte spécifique, offrant un meilleur compromis entre simplicité d’implémentation, robustesse et performances en conditions embarquées.

## 6 Implémentation embarquée

### 6.1 Export des poids du modèle

Afin de permettre l’exécution du réseau de neurones sur une plateforme embarquée ne disposant pas de bibliothèques de *deep learning*, les poids et biais des modèles entraînés ont été exportés depuis l’environnement Python vers un format exploitable en langage C++. Les paramètres ont été extraits sous forme de fichiers texte, puis regroupés dans le fichier (`model_weights.h`) sous forme de tableaux statiques.

Cette méthode permet d’intégrer directement les paramètres du réseau dans le code embarqué, sans dépendance externe, tout en garantissant la cohérence entre le modèle entraîné et l’inférence réalisée sur la Raspberry Pi.

Dans le cas du réseau de neurones convolutionnel (CNN), les poids appris lors de l’entraînement sont exportés sous forme de fichiers binaires afin de pouvoir être utilisés directement dans l’implémentation en C++. Pour chaque couche du réseau, deux fichiers binaires distincts sont générés : l’un contenant les poids et l’autre les biais associés.

Les fichiers binaires utilisés pour le CNN sont les suivants :

- `layer1.0.weight.bin` et `layer1.0.bias.bin` pour la première couche convolutionnelle,
- `layer2.0.weight.bin` et `layer2.0.bias.bin` pour la deuxième couche convolutionnelle,
- `layer3.weight.bin` et `layer3.bias.bin` pour la couche entièrement connectée de sortie.

Ces fichiers sont ensuite lus lors de l’exécution du programme embarqué et les valeurs qu’ils contiennent sont chargées en mémoire dans le code C++. Les poids et biais sont stockés dans des structures de données définies dans le fichier `neural_net_cnn.cpp`, permettant ainsi de reproduire fidèlement le calcul du passage avant (*forward pass*) du CNN sans dépendre de bibliothèques de deep learning externes. Cette approche garantit une implémentation maîtrisée, légère et compatible avec les contraintes de la plateforme embarquée.

## 6.2 Implémentation du réseau en C

### 6.2.1 MLP

Le réseau de neurones a été implémenté manuellement en langage C++ afin de permettre son exécution sur la Raspberry Pi. Chaque couche entièrement connectée est représentée par une structure contenant les dimensions d'entrée et de sortie, ainsi que les tableaux de poids et de biais associés.

La propagation avant est réalisée par une fonction calculant, pour chaque neurone, la somme pondérée des entrées à laquelle est ajouté le biais correspondant. Les fonctions d'activation sont implémentées explicitement : la fonction ReLU est appliquée aux couches intermédiaires, tandis que la fonction softmax est utilisée en sortie afin d'obtenir des probabilités associées à chaque classe.

L'interface du réseau est définie dans un fichier d'en-tête, assurant une séparation claire entre la définition des structures et leur implémentation. Cette organisation facilite l'intégration du réseau dans le programme principal et garantit la compatibilité avec le code C++ utilisé pour le prétraitement des images.

### 6.2.2 CNN

Le réseau convolutionnel a été implémenté en C++ sous la forme d'un *forward pass* explicite, sans dépendre de bibliothèques de deep learning. L'objectif est de reproduire le calcul effectué en Python, tout en restant compatible avec les contraintes d'exécution sur une plateforme embarquée.

Les images prétraitées (format  $28 \times 28$ ) sont représentées en mémoire par des tableaux linéaires, manipulés via des pointeurs (`double*`). Ainsi, un pixel en position  $(y, x)$  est accédé par l'indice  $y \cdot W + x$ , ce qui permet des parcours efficaces en mémoire lors des opérations de convolution.

La convolution 2D (`Conv2d`) est réalisée en balayant tous les pixels de l'image, puis en accumulant la somme pondérée des valeurs du voisinage défini par le noyau (ici  $5 \times 5$ ), à laquelle on ajoute un biais. Un *zero-padding* est appliqué : lorsque l'indice sort des limites de l'image, la contribution est considérée nulle. Après chaque convolution, une activation ReLU (`ReLU`) est appliquée élément par élément afin d'introduire la non-linéarité du modèle.

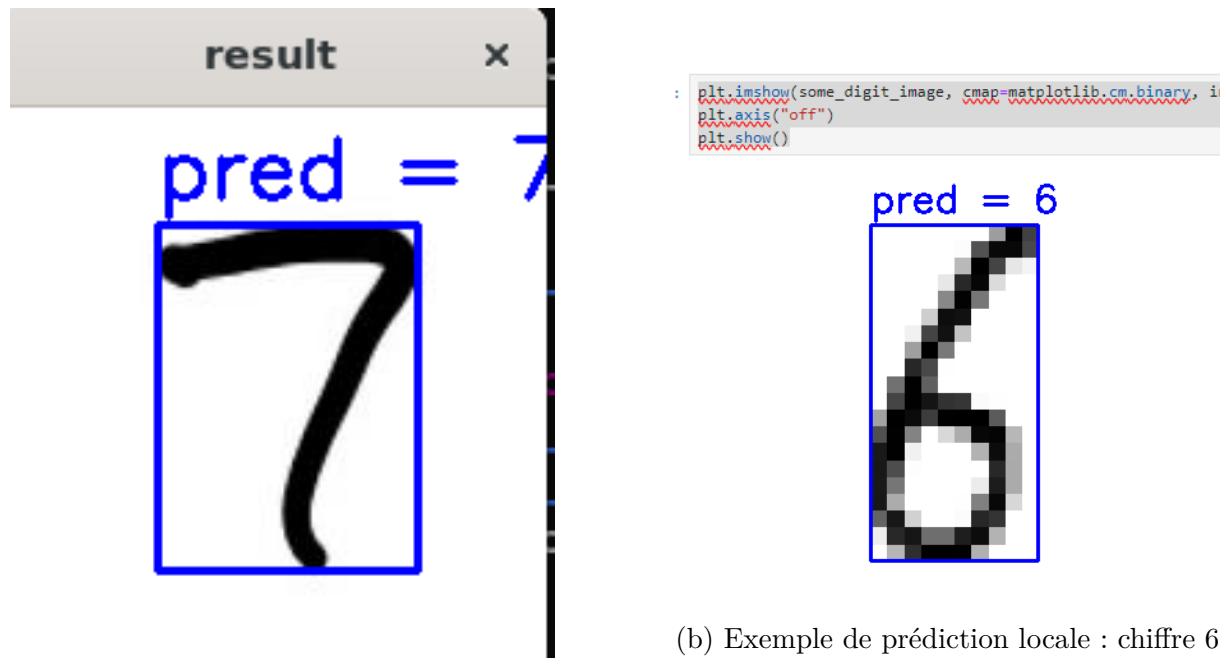
Enfin, la couche de sortie est une couche entièrement connectée (`Linear`) qui transforme le tenseur aplati en un vecteur de scores (*logits*) de taille égale au nombre de classes. Les poids appris en Python sont chargés depuis des fichiers binaires (`.bin`) au démarrage (fonction `load_weights`), puis stockés en mémoire dans des buffers utilisés par les différentes fonctions. Cette approche permet de garder une implémentation simple, contrôlable et directement exécutable sur la carte, tout en restant fidèle au modèle entraîné.

### 6.3 Validation locale du MLP avant déploiement embarqué

Avant le déploiement du modèle sur la plateforme embarquée, une phase de validation a été réalisée en local sur un ordinateur. Cette étape permet de vérifier le bon fonctionnement du réseau de neurones, ainsi que la cohérence entre le prétraitement, l'inférence et la prédiction finale, dans un environnement maîtrisé.

La Figure 8 présente deux exemples de chiffres traités localement. Pour chaque image, le chiffre est détecté, recadré, normalisé puis transmis au MLP, qui fournit une prédiction correcte. Ces résultats confirment que l'ensemble de la chaîne de traitement (prétraitement + inférence) fonctionne comme attendu.

Cette validation locale montre que le modèle MLP est stable, précis et prêt à être déployé sur la carte embarquée. Elle constitue une étape essentielle avant le passage à l'exécution en temps réel sur la Raspberry Pi.



(a) Exemple de prédiction locale : chiffre 7

(b) Exemple de prédiction locale : chiffre 6

FIGURE 8 – Validation locale du MLP sur PC avant déploiement embarqué.

### 6.4 Validation locale du CNN avant déploiement embarqué

Après l'implémentation du réseau convolutionnel en C++, une phase de validation a également été réalisée en local sur un ordinateur. Cette étape permet de vérifier le bon fonctionnement du modèle, ainsi que la cohérence entre le prétraitement des images, le calcul du *forward pass* et la prédiction finale, avant tout déploiement sur la plateforme embarquée.

La Figure 9 présente plusieurs exemples de chiffres traités localement par le CNN. Les chiffres sont détectés dans l'image, recadrés, normalisés puis transmis au réseau convolutionnel, qui fournit une prédiction correcte. Ces résultats montrent que le CNN est fonctionnel et que son

implémentation en C++ reproduit correctement le comportement du modèle entraîné en Python.

À ce stade du projet, la carte embarquée n'étant pas disponible, le CNN n'a pas encore pu être testé en conditions réelles sur la plateforme cible. Cependant, l'ensemble des poids du modèle validé est exporté sous forme de fichiers binaires et déposé sur le dépôt Git du projet. Il suffira donc de charger ces poids sur la carte pour procéder au déploiement et à l'évaluation embarquée lorsque le matériel sera disponible.

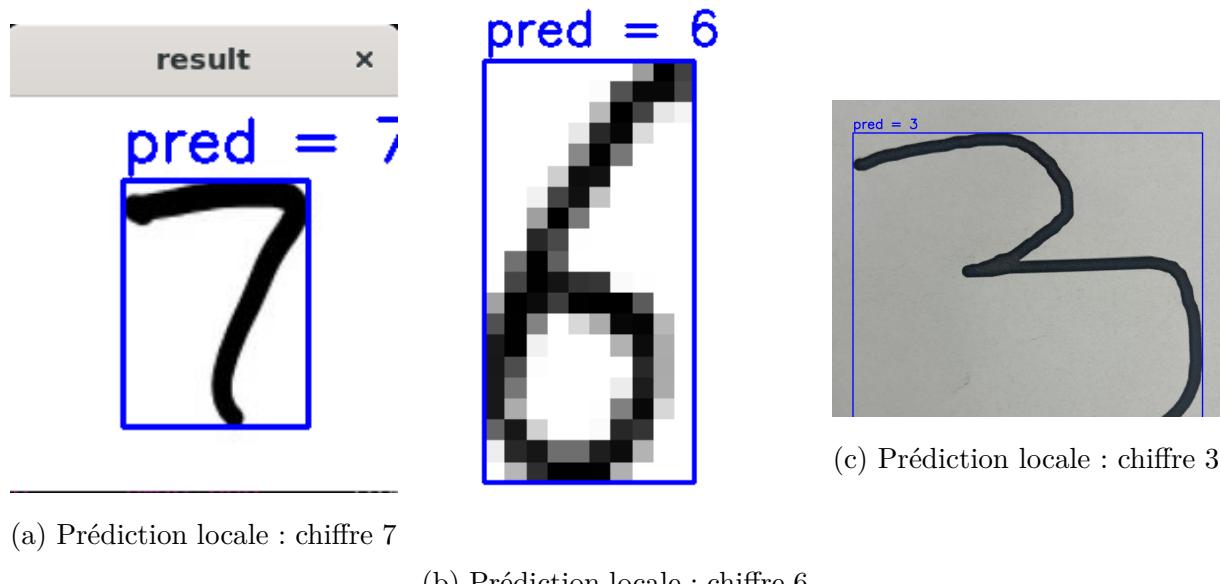


FIGURE 9 – Validation locale du CNN en C++ avant déploiement embarqué.

## 6.5 Validation de l'implémentation embarquée

Avant l'intégration du réseau de neurones dans la chaîne complète de traitement embarquée, une étape de validation a été réalisée afin de vérifier la conformité de l'implémentation en langage C par rapport au modèle entraîné sous Python.

Un échantillon issu de l'ensemble de test MNIST a été utilisé comme entrée du modèle MLP sous PyTorch, puis exécuté avec l'implémentation équivalente en C en utilisant les mêmes valeurs d'entrée et les poids exportés. Les sorties numériques ainsi que la classe prédite ont été comparées.

Les résultats obtenus sont strictement identiques dans les deux environnements, confirmant la validité de l'export des poids, la justesse des calculs réalisés en C et la fiabilité de l'implémentation avant son intégration complète sur la Raspberry Pi.

## 6.6 Prétraitement des images sur Raspberry Pi

Le prétraitement des images est une étape clé de la chaîne de reconnaissance, car il conditionne directement les performances du modèle. Dans ce projet, la bibliothèque OpenCV a été utilisée en C++ pour sa robustesse, ses fonctions optimisées de traitement d'images et sa compatibilité avec la Raspberry Pi. Son intégration facilite également l'interfaçage avec le réseau de neurones implémenté en langage C, tout en garantissant de bonnes performances d'exécution.

Le pipeline de prétraitement appliqué aux images issues de la caméra vise à produire des entrées compatibles avec le format MNIST. Il comprend la conversion de l'image en niveaux de gris, suivie d'une binarisation afin de séparer le chiffre du fond et de limiter l'influence des variations d'éclairage. L'image est ensuite redimensionnée en  $28 \times 28$  pixels.

À l'issue de ce traitement, l'image obtenue présente un chiffre centré et normalisé, directement exploitable par le réseau de neurones embarqué.

## 6.7 Intégration du flux vidéo et inférence en temps réel

La reconnaissance de chiffres en temps réel s'appuie sur l'exploitation du flux vidéo fourni par la caméra de la Raspberry Pi. Le flux, encodé en H.264, est décodé à l'aide de GStreamer puis traité image par image via OpenCV, offrant une solution efficace pour la manipulation de vidéo sur une plateforme embarquée.

Chaque image du flux est récupérée sous forme de matrice OpenCV, puis soumise au pipeline de prétraitement avant d'être transmise au réseau de neurones embarqué. Le modèle MLP effectue alors l'inférence et produit la prédiction du chiffre présent dans l'image.

Les images peuvent ensuite être enrichies avec la classe prédite et renvoyées vers un client distant. Le programme intègre également une mesure du nombre d'images traitées par seconde, permettant d'évaluer les performances du système en temps réel et de valider la faisabilité d'une reconnaissance continue sur Raspberry Pi.

## 6.8 Environnement logiciel

Le développement et les tests ont été réalisés dans un environnement logiciel maîtrisé afin d'assurer la reproductibilité des résultats. L'utilisation de conteneurs Docker a permis d'isoler les dépendances et de garantir une configuration logicielle cohérente entre les phases de développement et de test.

Le code en langage C et C++ a été compilé et testé dans un environnement proche de celui de la Raspberry Pi, facilitant le débogage et la validation. La compilation croisée a également été utilisée afin de générer des exécutables compatibles avec l'architecture cible, tout en réduisant le temps de compilation et en simplifiant l'intégration des bibliothèques externes.

# 7 Mesures de performances

L'objectif est de vérifier que le système de reconnaissance fonctionne en temps réel sur Raspberry Pi. Les mesures portent sur le temps d'inférence du MLP et la fluidité du traitement vidéo.

## 7.1 Temps d'inférence du MLP

Le temps d'inférence correspond au délai nécessaire pour que le MLP produise une prédiction. Sur Raspberry Pi, la propagation avant en C présente un temps très court et stable, environ **0,10 ms** par image, avec des valeurs comprises entre **0,095 ms** et **0,103 ms**.

```
Temps d'inférence MLP : 0.100055 ms
FPS: 30
Temps d'inférence MLP : 0.099351 ms
FPS: 30
Temps d'inférence MLP : 0.098981 ms
FPS: 29
Temps d'inférence MLP : 0.099666 ms
FPS: 30
Temps d'inférence MLP : 0.102555 ms
FPS: 29
Temps d'inférence MLP : 0.098444 ms
FPS: 30
Temps d'inférence MLP : 0.095758 ms
```

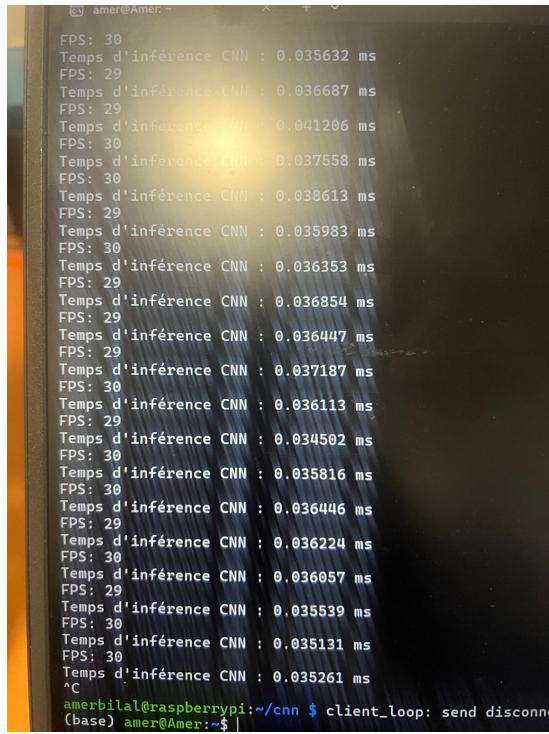
FIGURE 10 – Mesure du temps d’inférence et FPS

## 7.2 Temps d’inférence du CNN

Le temps d’inférence correspond au délai nécessaire pour que le réseau convolutionnel produise une prédiction à partir d’une image prétraitée. Sur la Raspberry Pi, l’implémentation du CNN en C++ permet d’obtenir un temps de calcul très court pour la propagation avant, de l’ordre de quelques dizaines de microsecondes par image.

Les mesures expérimentales montrent un temps d’inférence moyen d’environ **0,036 ms**, avec des valeurs comprises entre **0,034 ms** et **0,041 ms**, et une cadence d’exécution proche de **30 FPS**. Ces résultats indiquent que, d’un point de vue purement computationnel, l’architecture convolutionnelle retenue est compatible avec une exécution en temps réel sur la plateforme embarquée.

Cependant, malgré ces performances temporelles satisfaisantes, le CNN n’a pas permis d’obtenir une reconnaissance fiable des chiffres sur la carte embarquée. Les performances en termes de précision sont restées nettement inférieures à celles du MLP, en raison notamment de la simplicité volontaire de l’architecture (un seul canal de sortie par couche convolutionnelle). Le CNN a pu être exécuté sur la Raspberry Pi pour des mesures de temps d’inférence, mais n’a pas été retenu comme solution finale en raison de performances de reconnaissance insuffisantes.



```
amer@Amer:~$ client_loop: send disconnect
FPS: 30
Temps d'inférence CNN : 0.035632 ms
FPS: 29
Temps d'inférence CNN : 0.036687 ms
FPS: 29
Temps d'inférence CNN : 0.041206 ms
FPS: 30
Temps d'inférence CNN : 0.037558 ms
FPS: 30
Temps d'inférence CNN : 0.038613 ms
FPS: 29
Temps d'inférence CNN : 0.035983 ms
FPS: 30
Temps d'inférence CNN : 0.036353 ms
FPS: 29
Temps d'inférence CNN : 0.036854 ms
FPS: 29
Temps d'inférence CNN : 0.036447 ms
FPS: 29
Temps d'inférence CNN : 0.037187 ms
FPS: 30
Temps d'inférence CNN : 0.036113 ms
FPS: 29
Temps d'inférence CNN : 0.034502 ms
FPS: 30
Temps d'inférence CNN : 0.035816 ms
FPS: 30
Temps d'inférence CNN : 0.036446 ms
FPS: 29
Temps d'inférence CNN : 0.036224 ms
FPS: 30
Temps d'inférence CNN : 0.036057 ms
FPS: 29
Temps d'inférence CNN : 0.035539 ms
FPS: 30
Temps d'inférence CNN : 0.035131 ms
FPS: 30
Temps d'inférence CNN : 0.035261 ms
^C
amerbilal@raspberrypi:~/cnn $ client_loop: send disconnect
(base) amer@Amer:~$ |
```

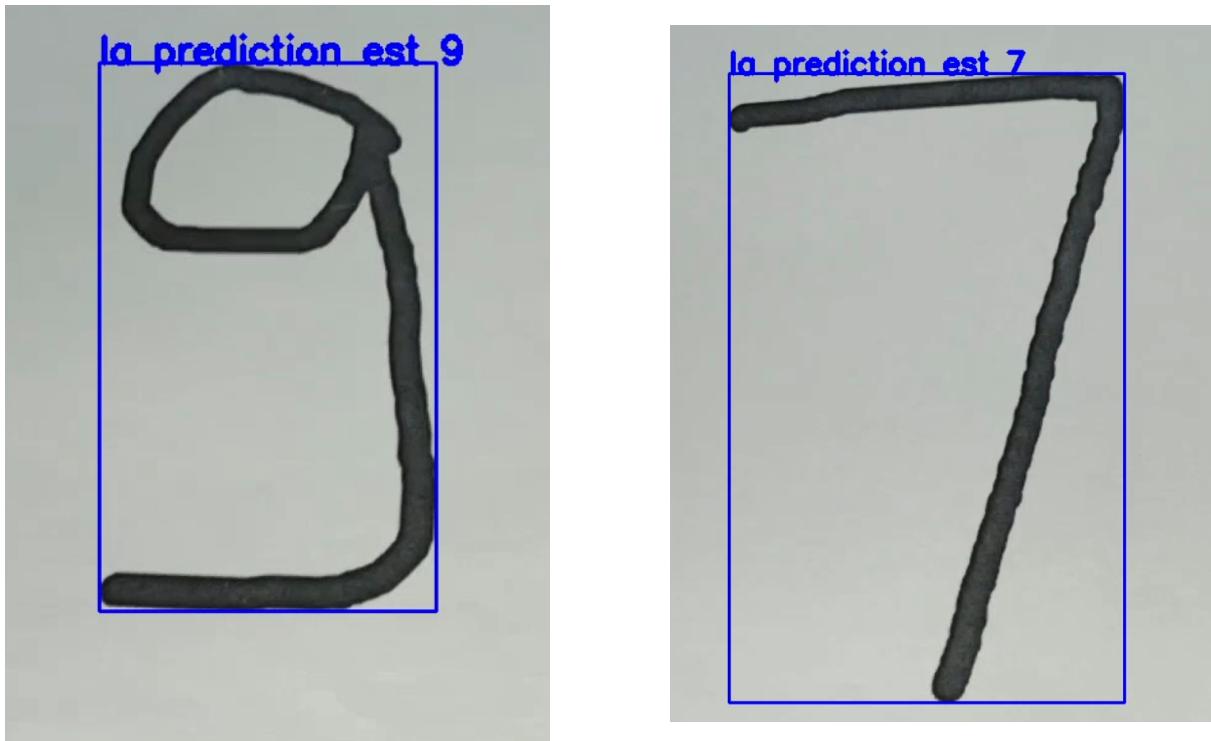
FIGURE 11 – Mesure du temps d’inférence et du nombre d’images par seconde (FPS) pour le CNN sur Raspberry Pi.

### 7.3 Performances en temps réel (FPS)

L’exécution complète du pipeline vidéo (capture, prétraitement, inférence et affichage) atteint une cadence stable d’environ **30 FPS**, permettant un traitement fluide et continu.

### 7.4 Validation visuelle des résultats

Des tests visuels en temps réel montrent que les chiffres manuscrits sont correctement détectés et encadrés, et que la prédiction est affichée sur l’image.



(a) Prédiction : 9

(b) Prédiction : 7

FIGURE 12 – Exemples de prédictions de chiffres manuscrits

Ces résultats valident la faisabilité d'un système de reconnaissance de chiffres manuscrits en temps réel sur Raspberry Pi, basé sur un MLP optimisé en C.

## 8 Analyse critique et perspectives d'amélioration

Les résultats obtenus au cours de ce projet montrent que la mise en œuvre d'un système de reconnaissance de chiffres manuscrits sur une plateforme embarquée est réalisable et performante, notamment avec un perceptron multicouche. Toutefois, certaines limites ont également été mises en évidence, ouvrant la voie à plusieurs pistes d'amélioration.

L'une des principales difficultés rencontrées concerne l'implémentation du réseau de neurones convolutionnel en langage C. Bien que le CNN offre de meilleures performances en termes de précision lors de l'entraînement et de l'évaluation sous Python, son portage vers un environnement embarqué s'est avéré plus complexe. La gestion des couches de convolution, du sous-échantillonnage et de la mémoire associée nécessite une implémentation plus avancée et plus coûteuse en ressources, ce qui a limité son intégration complète dans le système embarqué dans le cadre de ce projet.

Par ailleurs, bien que le MLP présente d'excellentes performances en temps d'inférence, il reste plus sensible aux variations d'écriture et aux conditions d'acquisition que le CNN. Certaines erreurs de classification observées sont liées à des chiffres visuellement proches ou à des pré-traitements imparfaits, ce qui souligne l'importance d'un pipeline de prétraitement robuste et

cohérent avec les données d'entraînement.

Plusieurs perspectives d'amélioration peuvent être envisagées. L'intégration d'un CNN optimisé pour l'embarqué, par exemple à l'aide de bibliothèques dédiées ou de modèles quantifiés, permettrait d'améliorer la précision tout en maîtrisant la complexité. L'augmentation de la base de données personnalisée, en diversifiant les styles d'écriture et les conditions d'éclairage, pourrait également renforcer la capacité de généralisation des modèles. Enfin, l'optimisation du prétraitement et l'exploration de techniques telles que la quantification ou la réduction de précision pourraient améliorer davantage les performances globales du système.

## 9 Conclusion

Ce projet avait pour objectif de concevoir et d'implémenter un système de reconnaissance de chiffres manuscrits sur une plateforme embarquée, depuis l'entraînement des modèles jusqu'à leur intégration en temps réel sur une Raspberry Pi. Les différentes étapes ont permis de comparer plusieurs architectures de réseaux de neurones, d'évaluer leurs performances et de valider la faisabilité d'une exécution embarquée. Les résultats obtenus montrent que le perceptron multicouche offre un excellent compromis entre précision et rapidité d'exécution, tandis que le réseau convolutionnel, bien que plus performant en théorie, présente des contraintes d'implémentation plus importantes dans un contexte embarqué.

Ce projet a permis d'acquérir de nombreuses compétences techniques, notamment en apprentissage automatique, en traitement d'images et en programmation embarquée. Le portage des modèles vers le langage C, leur intégration avec une caméra et l'optimisation des performances ont renforcé la compréhension des contraintes liées aux systèmes embarqués. L'utilisation de Docker a également permis de maîtriser la gestion d'environnements de développement reproductibles, de simplifier les phases de compilation et de test, et d'assurer une meilleure portabilité du code entre la machine de développement et la Raspberry Pi.

Enfin, ce projet présente un apport pédagogique important en mettant en lien des notions théoriques et leur application pratique. Il illustre concrètement les difficultés rencontrées lors du passage d'un modèle entraîné dans un environnement haut niveau vers un système embarqué réel. Cette expérience met en évidence les compromis nécessaires entre précision, complexité et performances, et constitue une base solide pour aborder des projets plus avancés dans le domaine de l'intelligence artificielle embarquée.