Implémentation d'un modèle d'intelligence artificielle pour un système embarqué

ARTHOZOUL Thomas

February 26, 2024

1 Introduction

1.1 Contexte du projet

L'intelligence artificielle (IA) connaît un développement rapide, soutenu par les avancées technologiques récentes telles que l'utilisation accrue des cartes graphiques et l'amélioration constante des capacités de calcul des ordinateurs. Cependant, un nouveau défi émerge : celui de déployer ces algorithmes intelligents sur des systèmes embarqués disposant de ressources limitées.

1.2 Matériel

Dans ce contexte, le projet utilisera des outils d'entreprise connus :

- Utilisation et création de conteneurs Docker avec installation du framework Keras et de ses dépendances : Docker est un outil de virtualisation légère qui permet de créer et gérer des conteneurs logiciels. Dans notre cas, Docker est utilisé pour créer un environnement de développement homogène et portable, contenant le framework Keras ainsi que ses dépendances.
- Utilisation et création de modèles de réseaux de neurones dans Keras (utilisé pour le développement de modèles d'IA).
- L'ensemble du projet sera disponible sur github : Projet d'IA - Thomas A. - AP5 - ISEN

Glossary

Biais Valeur ajoutée à la somme pondérée des entrées d'un neurone avant l'application de la fonction d'activation. Le biais permet au neurone d'ajuster la sortie indépendamment de ses entrées, améliorant ainsi la flexibilité du modèle pour l'apprentissage.. 3

Dense Type de couche dans un réseau de neurones où chaque neurone reçoit des entrées de tous les neurones de la couche précédente, formant ainsi une structure « dense ». Ces couches sont également connues sous le nom de couches « fully connected ».. 3

Docker Plateforme de conteneurisation permettant de développer, déployer et exécuter des applications dans des conteneurs logiciels légers et portables.. 3

Flatten Opération utilisée dans le traitement des réseaux de neurones pour convertir la structure multidimensionnelle des données en un vecteur unidimensionnel. Ceci est souvent nécessaire pour préparer les données d'entrée pour les couches denses d'un réseau de neurones.. 3

Intelligence artificielle Ensemble de théories et de techniques visant à réaliser des machines capables de simuler l'intelligence humaine. 3

Keras Bibliothèque de programmation open source pour le deep learning qui permet la construction et l'entraînement de modèles d'intelligence artificielle de manière intuitive.. 3

MLP Acronyme de Perceptron Multicouche, une classe de réseau de neurones artificiels feedforward. Le MLP consiste en au moins trois couches de nœuds : une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Chaque nœud, à l'exception des nœuds d'entrée, est un neurone qui utilise une fonction d'activation non linéaire. Le MLP utilise une technique d'apprentissage supervisé appelée rétropropagation pour l'entraînement.. 3

Phase d'apprentissage cela se réfère à la manière dont les modèles s'ajustent et s'améliorent en traitant des données.. 3

Phase d'inférence Étape dans le processus de l'IA où le modèle entraîné est utilisé pour faire des prédictions sur de nouvelles données.. 3

Réseaux de neurones Modèles informatiques inspirés du fonctionnement du cerveau humain, utilisés pour simuler la manière dont les humains apprennent, reconnaissent des motifs et prennent des décisions.. 3

Softmax Fonction d'activation utilisée principalement dans la dernière couche d'un réseau de neurones pour la classification, qui convertit les scores de sortie en probabilités. La fonction softmax assure que la somme de toutes les probabilités de sortie est égale à 1.. 3

TensorFlow Framework open source pour le calcul numérique qui facilite la construction, l'entraînement et le déploiement de modèles d'intelligence artificielle, en particulier ceux basés sur des réseaux de neurones profonds. Développé par l'équipe de Google Brain, TensorFlow offre une flexibilité pour l'expérimentation et l'optimisation de modèles d'IA, tout en fournissant des outils pour faciliter le déploiement de solutions d'apprentissage automatique à grande échelle.. 3

Weight Dans le contexte des réseaux de neurones, un poids est une valeur numérique qui est attribuée à une entrée du neurone. Les poids sont ajustés pendant l'entraînement du modèle pour minimiser l'erreur de sortie du réseau.. 3

Intelligence artificielle Docker Keras Réseaux de neurones MLP Phase d'apprentissage Phase d'inférence TensorFlow Flatten Dense Softmax Weight Biais

Contents

2 Implémentation du projet

2.1 Base de donnée

Pour ce faire, nous devons créer notre base de données et déterminer les données qui seront utilisées pour l'apprentissage ainsi que celles destinées à tester l'efficacité de l'algorithme. En amont, chaque étudiant a créé 100 images (10 par classe) pour constituer nos jeux de données respectifs. Afin d'évaluer l'efficacité de l'algorithme, j'ai suivi les étapes suivantes :

- Création d'un répertoire contenant les images d'apprentissage.
- Collecte des images provenant de différentes sources de données.
- Assemblage de ces 400 images.
- Création d'un sous-répertoire contenant 100 images (les miennes, considérées comme les plus lisibles) pour tester l'efficacité de l'algorithme.

J'ai souhaité disposer d'un jeu de données plus conséquent afin d'améliorer la robustesse et la généralisation de l'algorithme de reconnaissance de caractères manuscrits. En travaillant avec un ensemble de données plus étendu, je cherchais à éviter les problèmes de surapprentissage, qui surviennent lorsque le modèle s'adapte trop spécifiquement aux données d'entraînement et perd en capacité de généralisation. En utilisant un jeu de données plus vaste et diversifié, j'ai pu exposer le modèle à une plus grande variété de cas, ce qui l'aide à apprendre des caractéristiques plus générales et à mieux généraliser lors de la classification de nouvelles images.

3 Architecture du modèle

Notre modèle utilise une architecture de perceptron multicouche (MLP), conçue pour traiter des images en nuances de gris de taille 28x28 pixels. Cette section détaille l'architecture du modèle en expliquant chaque couche, les choix des fonctions d'activation, et la raison derrière la dimension spécifique des données d'entrée.

3.1 Couche flatten

```
layers.Flatten(input_shape=(image_size[0], image_size[1], 1))
```

Cette couche transforme les images d'entrée 2D de 28x28 pixels avec un seul canal (nuances de gris) en vecteurs unidimensionnels de 784 éléments (28 * 28 = 784). Cette opération est cruciale, car elle prépare les données pour le traitement par les couches denses suivantes.

3.2 Couches denses avec activation de la fonction tangente hyperbolique « tanh »

```
layers.Dense(512, activation='tanh'),
```

Les couches `Dense` implémentent l'opération suivante sur les données d'entrée x:

$$\text{sortie} = \tanh(W \cdot x + b)$$

où W représente la matrice des poids, b le vecteur de biais, et tanh la fonction d'activation hyperbolique tangente. La fonction tanh est définie comme:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Elle produit des sorties dans l'intervalle $[-1, 1]$, aidant à centrer les données et potentiellement à accélérer la convergence pendant l'entraînement.

3.3 Couche de sortie avec softmax

```
layers.Dense(10, activation='softmax')
```

La dernière couche `Dense` compte 10 neurones, correspondant aux 10 classes de chiffres (0 à 9). La fonction d'activation `softmax` est utilisée pour convertir les scores de chaque neurone en probabilités. La sortie du `softmax` est calculée comme suit pour un score donné z_i parmi les scores z pour toutes les classes:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{J} e^{z_j}}$$

où i représente l'indice d'une classe spécifique et j parcourt tous les scores de classe. Cette fonction assure que la somme des probabilités de toutes les classes est égale à 1, facilitant l'interprétation des sorties du modèle comme des probabilités de classe.

3.4 Pourquoi 784 (28x28 pixels) ?

La taille d'entrée de 784 découle directement de la dimension des images d'entrée, qui sont de 28x28 pixels. Chaque pixel de l'image est traité comme une caractéristique individuelle par le modèle, donc une image de 28 par 28 pixels est convertie en un vecteur de 784 caractéristiques.

Une problématique rencontrée

Nous n'avons pas besoin de spécifier explicitement "784" dans notre code, car TensorFlow déduit cette dimension automatiquement à partir de la taille de l'image et de la configuration de la couche Flatten. La seule référence implicite à "784" est donc dans la façon dont vous configurez la taille de l'image d'entrée et décidez de l'aplatir, préparant les données pour le traitement par les couches denses du MLP.

```
image_size = (28, 28)
layers.Flatten(input_shape=(image_size[0], image_size[1], 1))
```

3.5 Synthèse

En résumé, l'architecture de notre modèle MLP est conçue pour transformer les données d'images 2D en vecteurs linéaires, les traiter à travers des couches denses avec une non-linéarité introduite par tanh, et finalement classifier les images en utilisant softmax pour produire des probabilités de classe. Cette architecture simple mais efficace est bien adaptée à la tâche de reconnaissance de chiffres sur des images en nuances de gris.

4 Choix de l'architecture : data science

Pour déterminer les paramètres de notre architecture de réseau de neurones, une séance a été dédiée consacrée à l'évaluation et à la sélection des paramètres de notre modèle. Les paramètres considérés sont les suivants :

- Le Nombre de Couches de Neurones : cet aspect définit la profondeur du réseau.
  Un plus grand nombre de couches peut potentiellement capturer des représentations plus complexes des données. Cependant, un réseau trop profond peut aussi mener à un surajustement et nécessiter plus de données pour l'entraînement. Il est crucial de trouver un équilibre qui permet au réseau de généraliser efficacement à partir des données d'apprentissage.
- Le Nombre de Neurones par Couche
  Le nombre de neurones détermine la capacité du modèle et sa capacité à apprendre des relations détaillées dans les données. Un nombre insuffisant peut ne pas capturer la complexité des données, tandis qu'un excès de neurones peut causer du surajustement et augmenter le temps d'entraînement.
- La fonction d'activation
  Les fonctions d'activation introduisent des non-linéarités dans le modèle, ce qui est essentiel pour apprendre et modéliser des relations complexes. Le choix de la fonction d'activation peut affecter la vitesse de convergence du modèle et sa capacité à traiter des problèmes non linéaires.

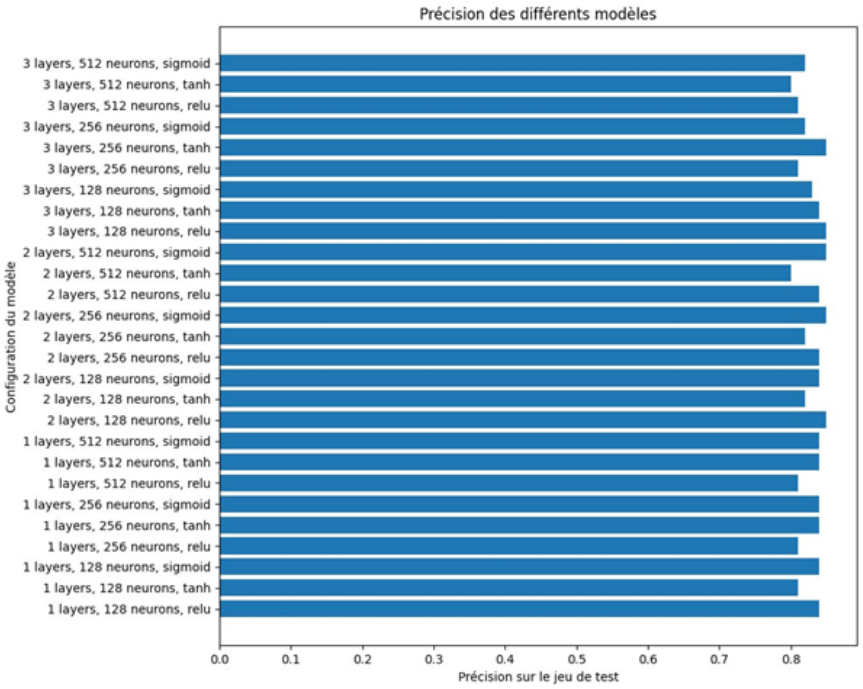

Figure 4.1: Résultat du script datascience.py version 1

L'analyse des performances révèle que plusieurs configurations du modèle présentent des niveaux de précision comparables. Face à cette proximité des résultats, notre décision s'est orientée, en partie par choix discrétionnaire, vers la configuration suivante :

Listing 1: Construction du modele avec MLP

```
model = models.Sequential([
    layers.Flatten(input_shape=(image_size[0], image_size[1], 1)),
    layers.Dense(512, activation='tanh'),
    layers.Dense(512, activation='tanh'),
    layers.Dense(10, activation='softmax')
])
```

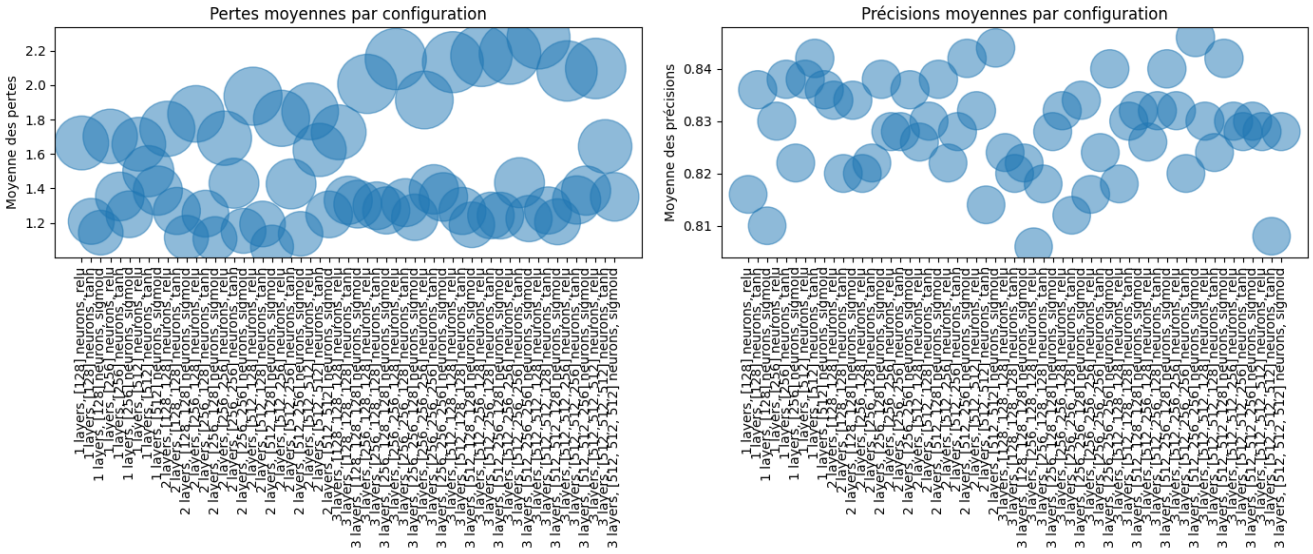

Figure 4.2: Graphique des pertes et précisions de différents modèles - datascience.py version 2

Le modèle le plus optimisé est le [512,256,256] avec la fonction d'activation sigmoid.

4.1 Analyse des résultats

Pendant une séance dédiée, plusieurs cas ont été testés et évalués. Les résultats obtenus ainsi que les remarques pertinentes sont présentés ci-dessous.

4.1.1 1er cas : mon jeu de donnée

```
base_dir = "thomasB" -> 70 images
test_dir = "thomasa" -> 30 images
```

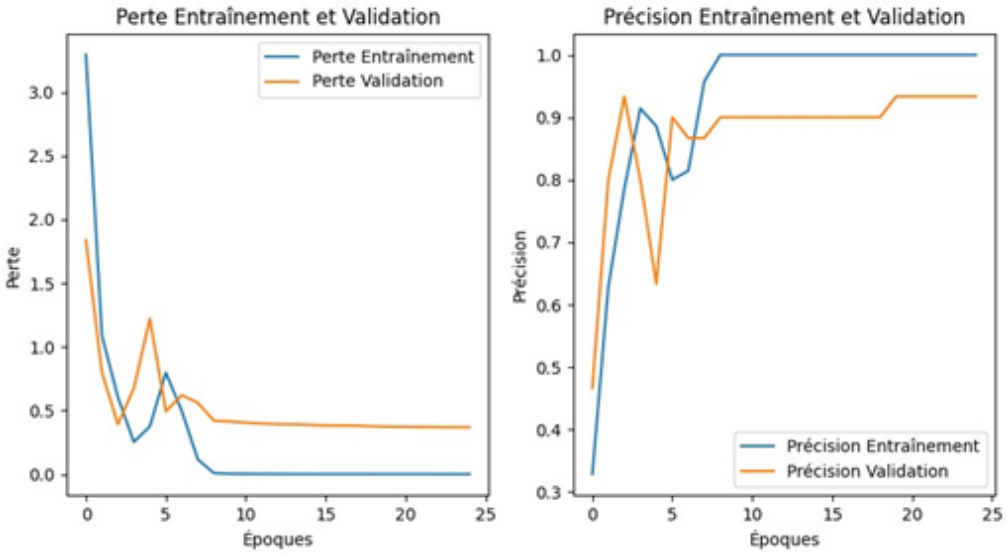

Figure 4.3: résultat du premier cas

Ces résultats indiquent que le modèle performe bien sur la tâche de reconnaissance de chiffres avec une perte faible (0.5) et une précision élevée (89%), tout en montrant des signes de bonne généralisation sans surajustement notable avec une base de test similaire à la base de données. Cependant, il est toujours recommandé de tester le modèle sur un ensemble de test indépendant pour confirmer si le modèle est robuste.

4.1.2 2ème cas : ma base de données est-elle robuste ?

```
base_dir = "thomas"   -> 100 images
test_dir = "gauthier" -> 100 images
```

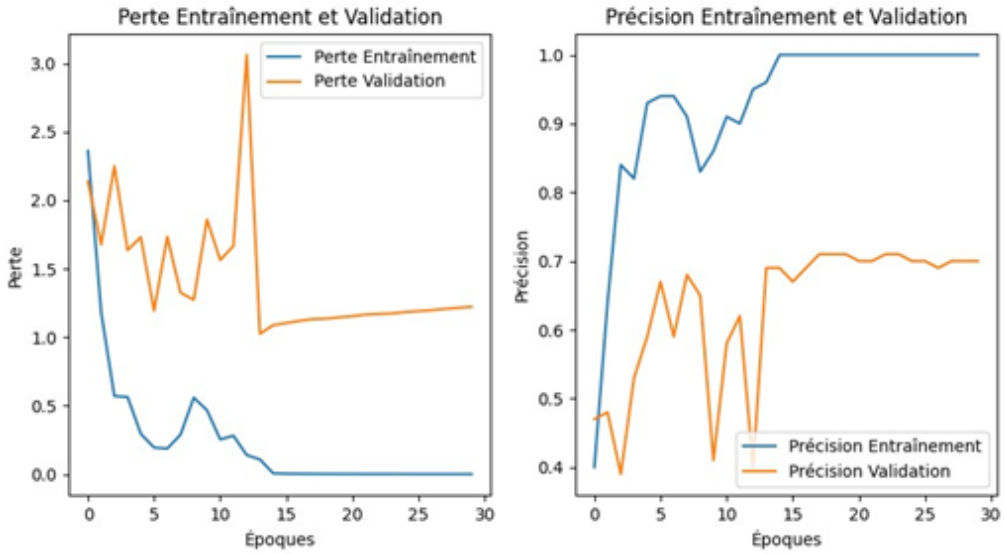

Figure 4.4: résultat de la base Gauthier

Le modèle a une certaine capacité à généraliser ce qu'il a appris à de nouvelles données, ce qui est un aspect crucial de l'apprentissage. Une précision de 70% dans ce contexte peut être un bon point de départ, et avec des ajustements supplémentaires et un affinement, il pourrait être possible d'améliorer encore cette performance.

4.1.3 3ème cas : confirmation ou infirmation de la robustesse ?

```
base_dir = "thomas"   -> 100 images
test_dir = "colin" -> 100 images
```

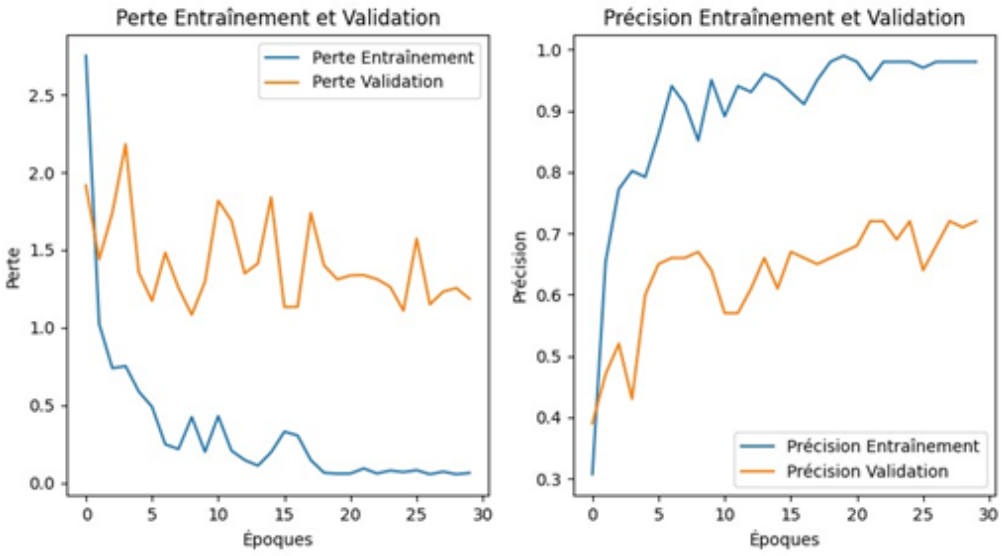

Figure 4.5: résultat de la base Colins

La baisse de précision de 70% à 65% lors du passage à un autre jeu de données pour un modèle de reconnaissance de chiffres peut être attribuée à plusieurs facteurs.

- nouveau jeu de données pourrait être moins similaire aux données sur lesquelles le modèle a été entraîné (variations de taille, d'orientation ou d'épaisseur des traits)
- la perte de validation est plus élevée et plus volatile par rapport à la perte d'entraînement

5% dans la précision peut être considérée comme assez proche, surtout si l'on prend en compte la variabilité inhérente aux jeux de données et aux conditions expérimentales.

Pour améliorer la précision sur ce nouveau jeu de données, il peut être nécessaire de revoir la représentativité des données d'entraînement, d'ajuster les hyperparamètres, d'appliquer des techniques de régularisation pour contrôler l'overfitting, ou d'explorer des méthodes d'augmentation de données pour accroître la variété des données d'entraînement et renforcer la robustesse du modèle.

4.1.4 4ème cas : base de données enrichie

```
base_dir = "data"   -> 400 images
test_dir = "thomas" -> 100 images
```

- Réduction de l'overfitting : Un plus grand ensemble de données d'entraînement peut aider à prévenir l'overfitting, car le modèle a plus d'exemples à partir desquels apprendre et est moins susceptible d'apprendre par cœur des détails spécifiques à un petit ensemble d'entraînement.
- Amélioration de la généralisation : Avec plus de données, le modèle peut mieux généraliser à de nouveaux exemples inédits, car il a été exposé à une variété plus large de cas pendant l'entraînement.
- Diversité des données : En augmentant la base de données, on peut introduire une plus grande diversité dans les types de chiffres que le modèle verra pendant l'entraînement, comme différents styles d'écriture, degrés de déformation, orientations.

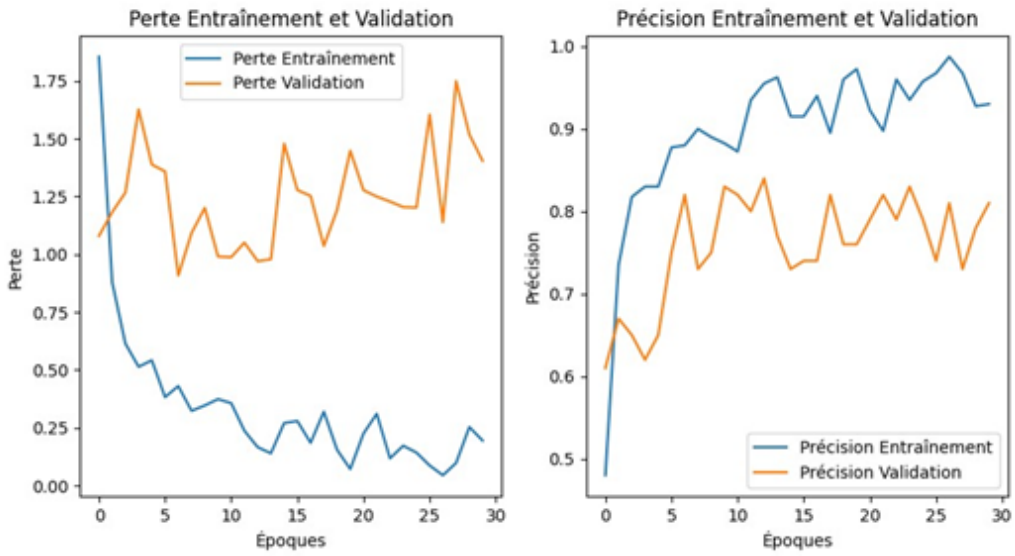

Figure 4.6: résultat de la base data

Accuracy

- La précision d'entraînement (81%) est élevée, montrant que le modèle s'ajuste bien à l'ensemble d'entraînement.
- La précision de validation présente quelques fluctuations mais reste en général sous la précision d'entraînement, ce qui est attendu puisque les modèles tendent généralement à mieux performer sur les données qu'ils ont déjà vues.

Loss

- La perte d'entraînement diminue de manière constante, ce qui indique que le modèle apprend bien sur l'ensemble d'entraînement.
- La perte de validation fluctue, mais ne présente pas de tendance à l'augmentation avec le temps, ce qui suggère qu'il n'y a pas de surajustement significatif. On pourrait par exemple faire des techniques de variations dans les images d'entraînement (data augmentation). Cependant, des risques sont présents tels que :
- Overfitting sur les transformations
- Perturbation des caractéristiques : altérer des caractéristiques clés qui sont importantes pour la classification. Par exemple, la rotation excessive d'un chiffre pourrait le rendre méconnaissable (un "6" retourné pourrait ressembler à un "9").
- Distribution des données biaisée : si certaines classes sont augmentées plus que d'autres ou si les augmentations ne sont pas bien distribuées, cela pourrait introduire un biais dans le modèle et affecter sa performance sur des données non vues.

4.2 Synthèse

Par conséquent, avec une précision de ¿=81% obtenue sur un jeu de données significatif et représentatif, le résultat est conforme à nos attentes. Cette performance indique que le modèle est capable de généraliser de manière efficace à de nouvelles données et possède une bonne capacité à reconnaître les chiffres dans des conditions variées. Avec ces résultats, nous pouvons être confiants dans la capacité du modèle à fournir des prédictions fiables et précises.

5 Implémentation pour un système embarqué (en C) : phase d'inférence

5.1 Exportation des poids et biais

Les poids et les biais servent de paramètres clés qui influencent directement la capacité du modèle à apprendre et à faire des inférences à partir des données. Définition de poids et biais Dans le contexte de l'implémentation en C d'un modèle de reconnaissance de chiffres, charger et utiliser correctement les poids et les biais pré-entraînés est crucial pour l'inférence. Ces paramètres, optimisés lors de l'entraînement, dictent comment le modèle réagit à de nouvelles entrées et détermine les classifications.

Listing 2: xtraction des poids et biais d'un modèle

```
import os
import numpy as np
from tensorflow.keras.models import load_model

model = load_model('custom_model.keras')

Creer le dossier pour enregistrer les poids et les biais
os.makedirs("weights", exist_ok=True)
os.makedirs("biaises", exist_ok=True)

for layer in model.layers:
    array = layer.get_weights()
    if len(array) != 0:
        w1,b1 = layer.get_weights()
        np.savetxt("./weights/" + layer.name + '.txt', w1)
        np.savetxt("./biaises/"+ layer.name + '.txt', b1)

print("Extraction␣des␣poids␣et␣biais␣terminee.")
```

5.2 Fonctions et structure

Le code en C pour le modèle de reconnaissance de chiffres intègre plusieurs structures et fonctions clés, essentielles pour implémenter l'inférence d'un réseau de neurones. Au cœur de cette implémentation se trouve la structure DenseLayer, qui encapsule une couche dense du réseau, incluant ses dimensions d'entrée et de sortie, ainsi que les poids et les biais qui sont les paramètres appris du modèle.

```
typedef struct {
    int input_size;
    int output_size;
    double* weights;
    double* biases;
} DenseLayer;
```

Les fonctions associées à cette structure, telles que create_dense_layer et forwardDense, facilitent respectivement l'initialisation de chaque couche avec les paramètres appropriés et l'exécution de la propagation avant, où les données d'entrée sont transformées à travers les poids et la fonction d'activation tanh pour produire des sorties intermédiaires.

```
DenseLayer* create_dense_layer(int input_size, int output_size, double* weights
    ↪ , double* biases) {
    DenseLayer* layer = (DenseLayer*)malloc(sizeof(DenseLayer));
    if (!layer) {
        fprintf(stderr, "Erreur␣d'allocation␣memoire␣pour␣la␣couche");
        exit(1);
    }
    layer->input_size = input_size;
    layer->output_size = output_size;
    layer->weights = weights;
    layer->biases = biases;

    return layer;
}
```

```
void forwardDense(DenseLayer *layer, const double *input, double *output) {
    for (int i = 0; i < layer->output_size; i++) {
        output[i] = layer->biases[i];
        for (int j = 0; j < layer->input_size; j++) {

            output[i] += input[j] * layer->weights[j * layer->output_size + i];
        }
        output[i] = tanh(output[i]);
    }
}
```

La fonction softmax est utilisée en fin de compte pour convertir les logits finaux en probabilités de classe, permettant une interprétation des prédictions du modèle.

```
void softmax(double *input, double *output, int length) {
    double sum = 0.0;
    for (int i = 0; i < length; i++) {
        output[i] = exp(input[i]);
        sum += output[i];
    }
    for (int i = 0; i < length; i++) {
        output[i] /= sum;
    }
}
```

Le traitement des images est géré par les fonctions LireBitmap, AllouerBMP, ConvertRGB2Gray, et d'autres fonctions associées, qui chargent une image BMP, allouent l'espace nécessaire pour ses données en mémoire, et la convertissent en nuances de gris pour l'entrée dans le réseau. Cette chaîne de traitement des données est cruciale pour préparer correctement les images en format compatible avec les attentes du réseau.

La gestion de la mémoire est également un aspect critique, avec des fonctions comme free_dense_layer et DesallouerBMP assurant un nettoyage approprié pour éviter les fuites de mémoire.

```
void free_dense_layer(DenseLayer* layer) {
    if (layer) {
        if (layer->weights) free(layer->weights);
        if (layer->biases) free(layer->biases);
        free(layer);
    }
}
```

6 Analyse des résultats

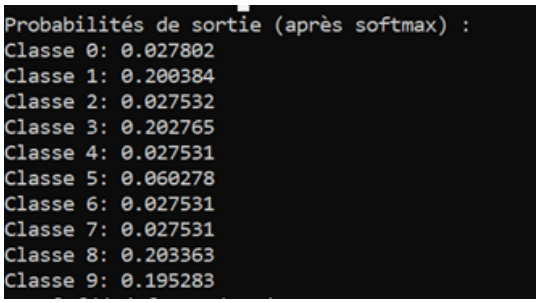

Figure 6.1: probabilité pour un 8

Figure 6.2: probabilité pour un 9

- La figure 6.1 montre les résultats du test d'une image représentant un 8, les résultats de la probabilités affichés sont trop faible (environ 20%). De plus, d'autres classes (comme les classes 1, 3 et 9) ont des probabilités similaires.
- La figure 6.2 montre les résultats du test d'une image représentant un 9, les résultats de la probabilités affichés sont très faible (environ 3%). De plus, d'autres classes (comme les classes 1, 4, 6 et 8) ont des probabilités supérieures.

Voici mon hypothèse principale pouvant expliquer ma problématique :

- Mauvaise Lecture des Poids et Biais
  Il est possible que le problème vienne de la façon dont le code charge les poids et les biais à partir des fichiers.

7 Conclusion du projet

En conclusion, bien que l'objectif initial de mise en œuvre d'un modèle d'IA pour un système embarqué en C ait rencontré des défis, le parcours à travers ce projet a été très instructif et enrichissant. L'exploration des concepts d'apprentissage profond et des subtilités de l'architecture des réseaux de neurones jusqu'à l'application pratique de tels modèles a considérablement élargi ma compréhension. Malgré le fait de ne pas avoir entièrement résolu le problème lors de l'implémentation en C, le projet a servi d'expérience d'apprentissage enrichissante, révélant la complexité des modèles en IA et de ses paramètres tels que la notion de couche, de neurone, de fonction d'activation ainsi que les phases d'apprentissages et d'inférences.

8 Webographie

References

[1] TensorFlow. *tf.keras.Model — TensorFlow v2.15.0.post1*. https://www.tensorflow.org/api_docs/python/tf/keras/Model.

[2] Keras. *The Sequential model — Keras*. https://keras.io/guides/sequential_model/.

[3] TensorFlow Core. *Keras: The high-level API for TensorFlow — TensorFlow Core*. https://www.tensorflow.org/guide/keras

[4] Machine Learnia. *Le Perceptron - Deep Learning*. https://www.youtube.com/watch?v=VlMm4VZ6lk4&t=25s&ab_channel=MachineLearnia

[5] Datagy. *Softmax Activation Function for Deep Learning: A Complete Guide*. https://datagy.io/softmax-activation-function/.

[6] DeepAI. *Softmax Function Definition*. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[7] Artemoppermann. *Activation Functions in Deep Learning: Sigmoid, tanh, ReLU*. https://artemoppermann.com/activation-functions-in-deep-learning-sigmoid-tanh-relu/.

[8] StackExchange. *Relu vs Sigmoid vs Softmax as hidden layer neurons*. https://stats.stackexchange.com/questions/218752/relu-vs-sigmoid-vs-softmax-as-hidden-layer-neurons.

9 Annexe

Listing 3: datascience.py

```
def build_model(n_layers, n_neurons, activation):
    model = models.Sequential()
    model.add(layers.Flatten(input_shape=(image_size[0], image_size[1], 1)))
    for _ in range(n_layers):
        model.add(layers.Dense(n_neurons, activation=activation))
    model.add(layers.Dense(10, activation='softmax'))

n_layers_options = [1, 2, 3]
n_neurons_options = [128, 256, 512]
activation_options = ['relu', 'tanh', 'sigmoid']
epochs = 30

model_performance = {}


for n_layers, n_neurons, activation in itertools.product(n_layers_options,
    ↪ n_neurons_options, activation_options):

    current_model = build_model(n_layers, n_neurons, activation)
    current_model.compile(optimizer='adam', loss='categorical_crossentropy',
        ↪ metrics=['accuracy'])

    print(f"Entrainement␣du␣modele␣avec␣{n_layers}␣couches,␣{n_neurons}␣
        ↪ neurones,␣activation␣{activation}")
    history = current_model.fit(
        train_generator,
        steps_per_epoch=max(1, train_generator.samples // batch_size),
        epochs=epochs,
        validation_data=test_generator,
        validation_steps=max(1, test_generator.samples // batch_size))

    test_loss, test_accuracy = current_model.evaluate(test_generator)
    print(f'Precision␣sur␣le␣jeu␣de␣test:␣{test_accuracy:.4f}␣-␣Parametres:␣
        ↪ Couches={n_layers},Neurones={n_neurons},Activation={activation}')

    model_performance[f'{n_layers}␣layers,␣{n_neurons}␣neurons,␣{activation}']
        ↪ = test_accuracy
```