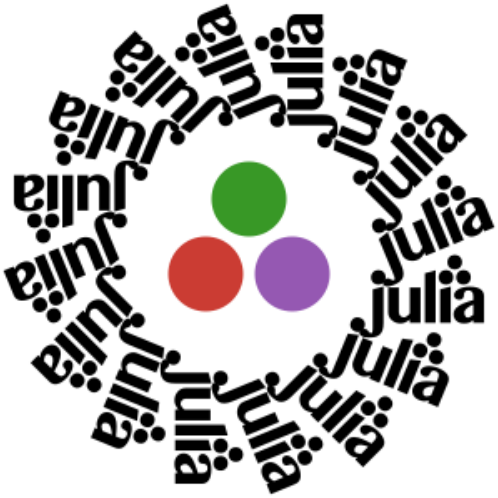
Projet Julia et R-Shiny

2025-02-11

Table of contents

Projet Julia et R-Shiny

Exploration et Évaluation des Réseaux de Neurones Convolutifs pour la Classification d'Images MNIST



Réalisé par :

- **BENCHEIKH El Amira Cerine**
- **BELOUARDA Manal**
- **MEKKAOUI Sara**

Date : 14/11/2024

Introduction

Le langage Julia

Julia est un langage de programmation relativement récent, lancé en 2012. Il a été conçu pour répondre aux besoins de calcul scientifique intensif, combinant la rapidité des langages compilés comme C et Fortran avec la facilité d'utilisation et de prototypage de langages interprétés comme Python. Julia est particulièrement apprécié dans le domaine du calcul numérique, de l'analyse de données et de l'intelligence artificielle. Voici quelques points forts de Julia :

- **Performance :** Julia est un langage compilé avec des performances proches de celles des langages bas-niveau, notamment grâce à son compilateur JIT (Just-In-Time).
- **Syntaxe intuitive :** La syntaxe de Julia est proche de celle de Python, ce qui rend le code plus lisible et accessible.
- **Support des opérations sur matrices et calcul scientifique :** Julia est optimisé pour les calculs sur matrices, les opérations vectorielles et le calcul parallèle, ce qui le rend idéal pour le développement d'algorithmes d'apprentissage automatique et de réseaux de neurones.

Dans ce projet, nous avons utilisé Julia pour la construction d'un modèle de réseau de neurones convolutifs (CNN) avec le package `Flux.jl`, spécialisé dans les applications de machine learning.

Réseaux de Neurones Convolutifs

Les réseaux de neurones convolutifs, ou CNNs, sont une classe de réseaux de neurones spécialement conçus pour le traitement des données sous forme d'images. Ils sont composés de couches convolutives, qui détectent les caractéristiques des images comme les bords, les textures et les formes, et de couches de pooling qui réduisent la dimensionnalité des données tout en conservant leurs informations essentielles.

Explication des Couches Utilisées dans le Modèle

Voici une brève explication des couches que nous utilisons dans notre modèle :

- **Couches de Convolution :** Ces couches appliquent des filtres sur les images pour détecter des caractéristiques spécifiques (comme des bords ou des textures).
- **Couches de Pooling :** Les couches de pooling réduisent la résolution des images, réduisant ainsi le nombre de paramètres et la complexité computationnelle du modèle.
- **Couches Denses :** Ces couches se trouvent en fin de modèle et permettent de faire une classification en combinant les caractéristiques extraites par les couches précédentes.

L'image ci-dessous montre le fonctionnement de base d'un CNN :

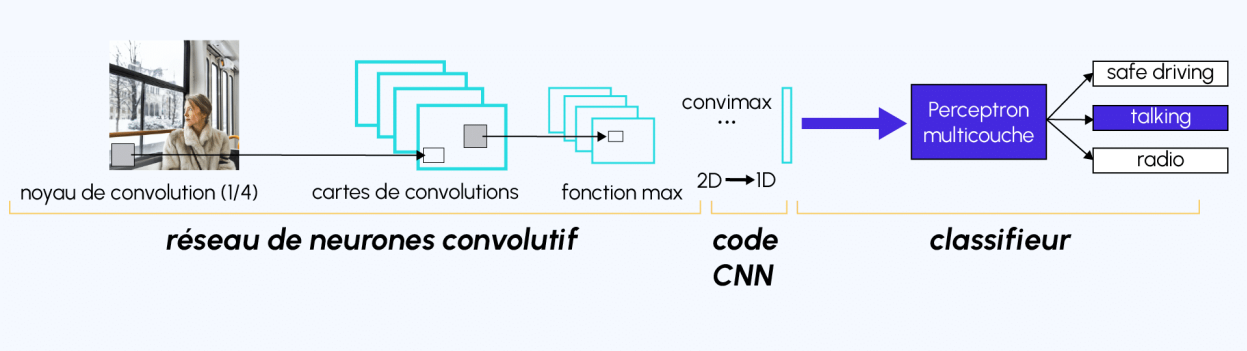


Figure 1: Schéma d'un réseau de neurones convolutifs

Les CNNs sont particulièrement efficaces pour la reconnaissance d'images, car ils apprennent à détecter automatiquement les caractéristiques pertinentes sans nécessiter de prétraitement manuel.

Contexte et Objectif du Projet

Le projet consiste à utiliser un modèle de réseau de neurones convolutifs pour classifier des images du dataset MNIST, qui est un ensemble d'images de chiffres manuscrits. Le but est de construire et d'entraîner un modèle capable de reconnaître chaque chiffre avec une haute précision.

Dans ce projet, nous avons divisé le code en plusieurs parties :

- **Préparation des données :** Nous avons chargé et pré-traité les données pour qu'elles soient prêtes pour l'apprentissage. Cela inclut la normalisation et le redimensionnement des images ainsi que l'encodage des labels en one-hot.
- **Construction du modèle CNN :** Le modèle utilise des couches convolutives et des couches de pooling pour extraire les caractéristiques des images, suivies de couches denses pour la classification finale.
- **Entraînement et Évaluation :** Nous avons configuré une fonction d'entraînement qui utilise un optimiseur `Adam` pour ajuster les poids du modèle en minimisant la fonction de perte de cross-entropie. Nous avons également calculé des métriques de précision, de rappel et de F1-score pour évaluer les performances du modèle.

Ce projet illustre comment les réseaux de neurones convolutifs peuvent être appliqués pour résoudre des problèmes de classification d'images et comment Julia et `Flux.jl` facilitent leur implémentation.

Les Bénéfices de Julia dans le Machine Learning

Julia présente plusieurs avantages par rapport à d'autres langages comme Python dans le contexte du Machine Learning :

- **Performance :** Julia est extrêmement rapide, proche du langage C, ce qui permet une exécution plus rapide lors de l'entraînement des modèles de Machine Learning.
- **Facilité d'utilisation :** Julia combine la simplicité de Python avec la performance des langages compilés.
- **Interopérabilité :** Julia peut facilement interagir avec d'autres langages comme Python, ce qui permet de profiter des bibliothèques existantes tout en tirant parti de la rapidité de Julia.

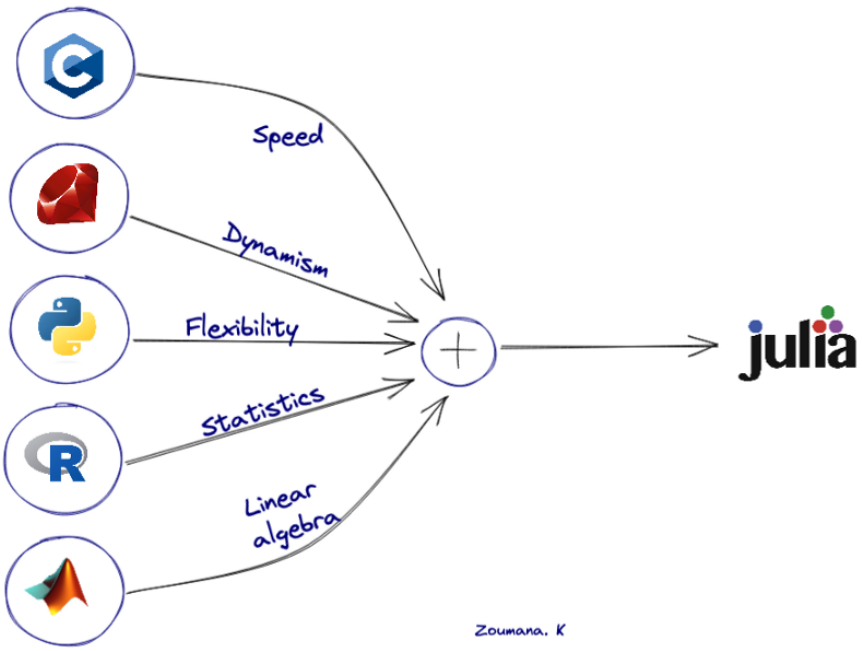


Figure 2: Graphique comparatif des performances

Comparaison entre Julia et Python

Bien que Python soit largement utilisé dans le Machine Learning, Julia offre des avantages significatifs :

- **Python** dispose d'un écosystème riche, avec des bibliothèques comme TensorFlow et PyTorch.
- **Julia**, bien qu'ayant un écosystème plus petit, est beaucoup plus rapide, surtout pour les calculs numériques lourds.

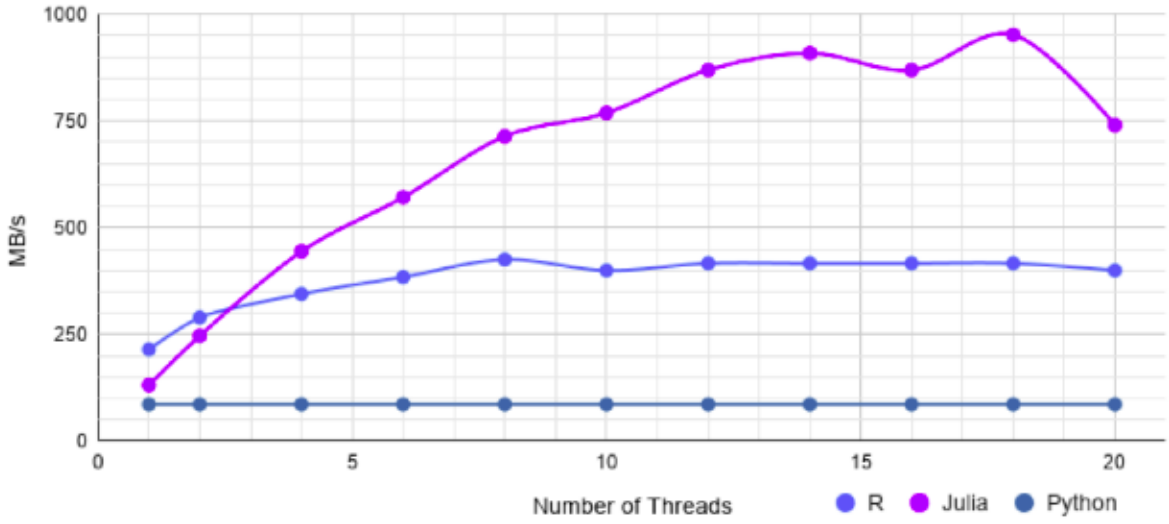


Figure 3: Comparaison des temps d'exécution entre Julia et Python

Inconvénients de Julia

Bien que Julia offre de nombreux avantages, il existe quelques inconvénients :

- L'écosystème est encore en développement et ne rivalise pas encore avec Python en termes de nombre de bibliothèques disponibles.
- La communauté est plus petite, ce qui signifie moins de ressources et d'exemples disponibles.

Préparation des Données

Décortiquer les Données

Avant de pouvoir entraîner notre modèle, nous devons charger et examiner les données pour comprendre leur structure.

Normalisation des Données

La normalisation consiste à ajuster les valeurs des pixels (qui sont souvent entre 0 et 255 dans le cas des images) pour qu'elles soient dans une gamme plus petite, typiquement entre 0 et 1. Cela permet de stabiliser l'entraînement en réduisant les fluctuations importantes des gradients.

La normalisation doit être effectuée avant de passer les données dans le réseau de neurones, car les modèles de Machine Learning, et en particulier les réseaux de neurones, fonctionnent mieux lorsque les données sont sur une échelle comparable. Typiquement, cela se fait juste après le chargement des données et avant la phase d'entraînement.

Pourquoi la normalisation est importante ?

- **Stabilité du Gradient** : Lors de l'entraînement du réseau de neurones, des valeurs de pixels très élevées (comme 255 pour les images) peuvent entraîner une explosion des gradients pendant la rétropropagation. Normaliser ces valeurs à une échelle plus petite comme [0, 1] permet de maintenir les gradients plus stables et favorise une convergence plus rapide.
- **Performance de l'Optimiseur** : Les optimiseurs comme Adam dans notre cas fonctionnent mieux lorsque les entrées sont sur des échelles similaires, ce qui accélère l'entraînement et peut conduire à de meilleures performances.

Canaux des Données

Dans le contexte des réseaux de neurones pour la reconnaissance d'images, les canaux (ou channels) font référence aux différentes couches de couleurs d'une image. Prenons l'exemple des images RGB (Rouge, Vert, Bleu), qui sont les plus courantes pour les images colorées.

Qu'est-ce que les canaux d'une image ?

- **Image en niveaux de gris** : Une image en noir et blanc a un seul canal, car chaque pixel est décrit par une seule valeur représentant l'intensité lumineuse (de noir à blanc).
- **Image RGB** : Une image en couleur est souvent composée de trois canaux : Rouge (R), Vert (G) et Bleu (B). Chaque pixel dans une image RGB a donc trois valeurs, une pour chaque couleur. Chaque couleur est représentée par une intensité (souvent entre 0 et 255, où 0 est l'absence de couleur et 255 est la couleur maximale).

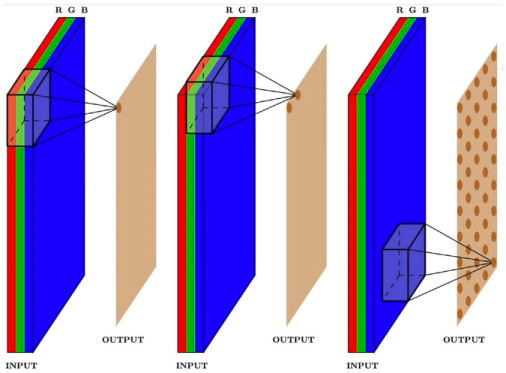


Figure 4: Comprendre le réseau CNN

Dans notre projet, notre modèle de réseau de neurones convolutifs (CNN) travaille avec des images en RGB. Chaque couche de convolution traitera les trois canaux de l'image. Lorsqu'on passe les images dans le modèle, il est crucial que les dimensions des données (en particulier le nombre de canaux) soient correctement prises en compte, car chaque couche de convolution appliquera des filtres à chacun des canaux (R, G, B) pour extraire des caractéristiques.

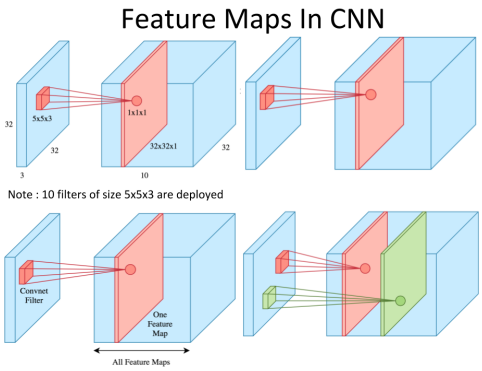


Figure 5: How channels RGB effect convolutional neural network

Encodage des Labels

Le terme "labels" fait référence aux étiquettes de classe pour chaque image dans les ensembles de données d'entraînement et de test. Ces étiquettes sont essentielles pour la classification d'images, car elles représentent la classe correcte à laquelle une image appartient. Par exemple, dans le cas du dataset MNIST, chaque image représente un chiffre (0 à 9), et l'étiquette correspond à ce chiffre.

Chargement des labels :

Lorsque vous chargez les données MNIST avec la fonction MNIST.traindata() et MNIST.testdata(), vous obtenez à la fois les images (train_X et test_X) et leurs labels correspondants (train_y et test_y). Ces labels indiquent quel chiffre est représenté sur chaque image.

Par exemple, si une image correspond au chiffre "3", alors l'étiquette pour cette image sera "3".

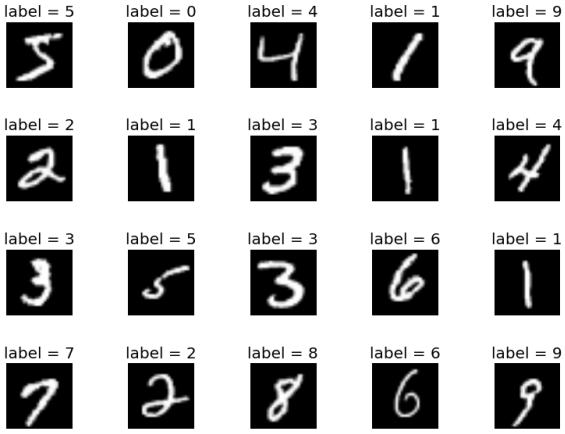


Figure 6: Labels des images

Construction du Modèle

Définir les Couches de Convolution

Ces couches sont responsables de l'extraction des caractéristiques à partir des images d'entrée (les chiffres de MNIST) et de la réduction de la dimension des données tout en conservant les informations importantes pour la classification. Dans notre modèle, nous avons défini deux couches de convolution.

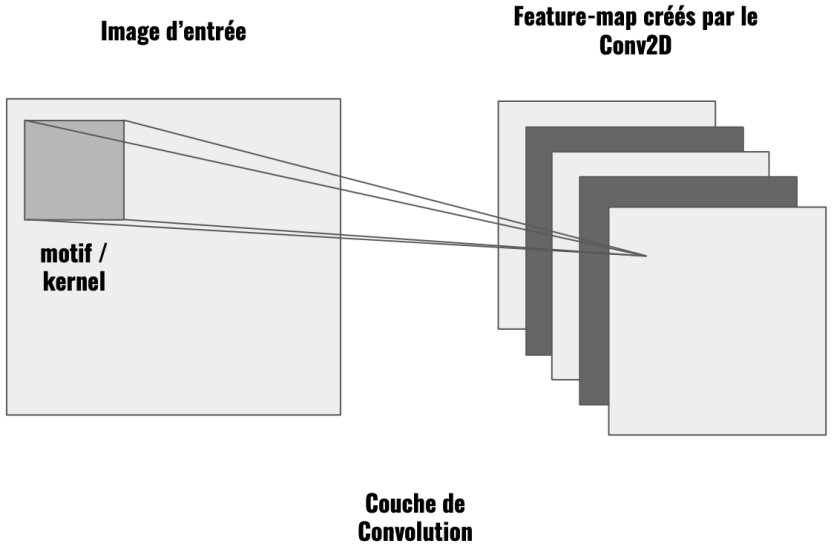


Figure 7: Couche de convolution

Filtres dans les Couches de Convolution

Les filtres appliquent des transformations spécifiques pour détecter des caractéristiques comme des bords ou des textures.

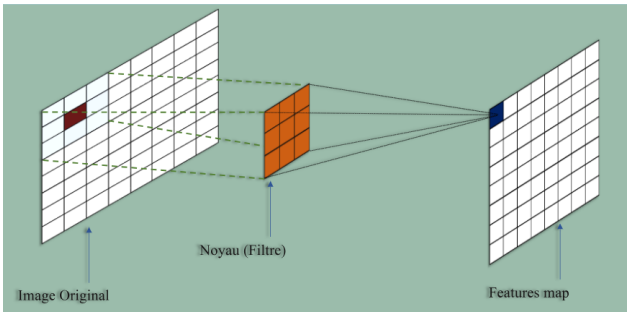


Figure 8: fonctionnement des filtres

MaxPooling

La couche de MaxPool est utilisée pour réduire la taille de l'image tout en conservant les informations les plus importantes, et la sortie des convolutions est aplatie (avec Flux.flatten) pour être passée dans des couches entièrement connectées (les couches Dense), qui feront la classification finale.

Dense et Softmax

Les couches denses sont utilisées pour combiner les caractéristiques extraites par les couches précédentes, et la fonction Softmax convertit les résultats en probabilités de classe.

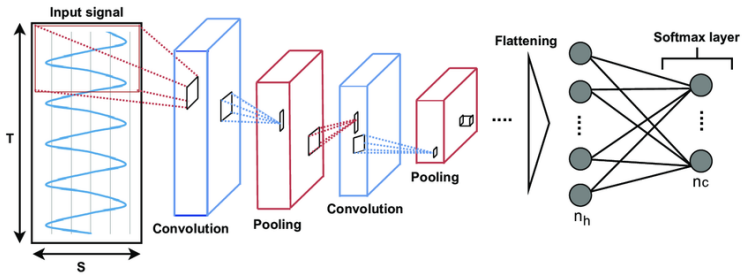


Figure 9: Illustation sur le fonctionnement de softmax

Entraînement du Modèle

Fonction de Perte

La fonction de perte, ou fonction de coût, est utilisée pour mesurer à quel point le modèle est éloigné de la vérité (les valeurs réelles) lorsqu'il fait des prédictions. L'objectif lors de l'entraînement d'un modèle de machine learning est de minimiser cette fonction de perte, en ajustant les paramètres du modèle (comme les poids dans un réseau de neurones) pour que l'erreur entre les prédictions et les valeurs réelles soit aussi faible que possible.

Entropie croisée (Cross-Entropy)

L'entropie croisée est une fonction de perte couramment utilisée pour les tâches de classification, en particulier pour les problèmes de classification multi-classes, où l'objectif est de prédire une catégorie parmi plusieurs.

Fonction d'Optimisation

Dans un réseau de neurones, l'optimiseur joue un rôle crucial dans l'apprentissage du modèle. Il est responsable de mettre à jour les poids du réseau de manière à ce que la fonction de perte (loss function) soit minimisée, c'est-à-dire que les prédictions du modèle soient de plus en plus proches des valeurs réelles au fur et à mesure de l'entraînement. Dans notre code, nous avons utilisé l'optimiseur Adam.

Calcul de la Précision (Accuracy)

Le calcul de la précision (ou accuracy) est une métrique couramment utilisée pour évaluer la performance d'un modèle de classification. Elle mesure la proportion des prédictions correctes parmi l'ensemble des prédictions effectuées par le modèle. Plus précisément, la précision calcule combien de fois le modèle a correctement classé les données par rapport au nombre total d'exemples.

$$\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}} = \frac{\sum_{i=1}^{n} \mathbf{1}\left(\hat{y}_i = y_i\right)}{n}$$

Métriques d'Évaluation

Recall, F1-Score, Precision, et Matrice de Confusion

Les métriques telles que le *Recall*, le *F1-Score*, et la *Precision* nous aident à évaluer la performance de notre modèle. La *Matrice de Confusion* permet de visualiser les faux positifs, faux négatifs, vrais positifs et vrais négatifs.

Recall (Rappel)

Le *Recall* mesure la capacité du modèle à identifier toutes les instances positives. Plus il est élevé, plus le modèle maximise le nombre de vrais positifs et minimise le nombre de faux négatifs.

La formule du *Recall* est la suivante :

$$\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}} = \frac{\sum_{i=1}^{n} \mathbf{1}\left(\hat{y}_i = y_i\right)}{n}$$

où : - (TP) : Vrais Positifs (True Positives) - (FN) : Faux Négatifs (False Negatives)

Precision (Précision)

La *Precision* mesure la proportion de prédictions positives correctes parmi toutes les prédictions positives. Une précision élevée signifie que les prédictions positives du modèle sont majoritairement correctes.

La formule de la *Precision* est la suivante :

$$\text{Precision} = \frac{TP}{TP + FP}$$

où : - (TP) : Vrais Positifs (True Positives) - (FP) : Faux Positifs (False Positives)

F1-Score

Le *F1-Score* est la moyenne harmonique de la précision et du rappel. Il permet de trouver un équilibre entre la précision et le rappel, en particulier lorsque les classes sont déséquilibrées.

La formule du *F1-Score* est la suivante :

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Matrice de Confusion

La *Matrice de Confusion* est une table qui permet de visualiser la performance du modèle en comparant les prédictions avec les vérités réelles. Elle permet de voir où le modèle se trompe le plus souvent. Elle est définie comme suit :

$$\begin{bmatrix} \text{TN} & \text{FP} \\ \text{FN} & \text{TP} \end{bmatrix}$$

où : - (TN) : Vrais Négatifs (True Negatives) - (FP) : Faux Positifs (False Positives) - (FN) : Faux Négatifs (False Negatives) - (TP) : Vrais Positifs (True Positives)

La matrice de confusion permet de mieux comprendre où le modèle se trompe (par exemple, s'il classe beaucoup de vrais positifs comme faux positifs).

Notre Modèle

Dans cette section, nous présentons la conception de notre modèle de réseau de neurones convolutifs (CNN), que nous avons créé de manière manuelle pour mieux comprendre les mécanismes sous-jacents des réseaux de neurones. Nous expliquons également les principales fonctions utilisées pour entraîner et évaluer notre modèle, et discutons des résultats obtenus.

Préparation des Données

La préparation des données est une étape essentielle avant d'entraîner un modèle d'apprentissage profond, car des données mal préparées peuvent influencer négativement la performance du modèle. Nous avons suivi plusieurs étapes clés pour préparer nos données :

- **Chargement des données** : Nous avons utilisé le jeu de données MNIST, qui contient des images de chiffres manuscrits. Les données d'apprentissage et de test ont été chargées respectivement à l'aide des fonctions `MNIST.traindata()` et `MNIST.testdata()`.
- **Normalisation des données** : Les images sont initialement représentées avec des valeurs allant de 0 à 255. Afin d'améliorer l'efficacité de l'apprentissage, nous avons normalisé ces valeurs en les divisant par 255, ce qui les amène dans l'intervalle [0, 1].
- **Encodage One-Hot** : Les étiquettes des images ont été converties en vecteurs binaires à l'aide de l'encodage one-hot. Chaque étiquette de chiffre a ainsi été transformée en un vecteur binaire de 10 dimensions, avec une valeur de 1 à la position correspondant à la classe, et 0 ailleurs.

Construction du Modèle CNN

Nous avons implémenté un réseau de neurones convolutif (CNN) avec deux couches de convolution, suivies de couches de max pooling, puis une couche dense pour la classification. Les principales fonctions utilisées pour construire ce modèle sont la convolution et le max pooling, que nous avons implémentées manuellement.

Fonction de Convolution

La fonction de convolution est au cœur du réseau de neurones convolutifs. Dans notre implémentation manuelle, nous avons conçu un algorithme de convolution qui applique un noyau (ou filtre) à une image pour extraire des caractéristiques locales telles que des bords ou des textures. Cette opération permet de transformer l'image d'entrée en une carte de caractéristiques, qui sera ensuite utilisée pour des tâches de classification ou de détection.

La convolution est mathématiquement définie comme une opération entre l'image et le noyau. Chaque pixel de la carte de caractéristiques résultante est calculé en appliquant une multiplication élément par élément entre un sous-ensemble de l'image et le noyau, suivi d'une somme des résultats. En termes mathématiques, cela peut être formulé comme suit :

$$y(i,j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \sum_{c=0}^{C-1} x(i+m, j+n, c) \cdot w(m,n,c)$$

où : - (y(i,j)) est la valeur de la carte de caractéristiques en la position ((i,j)). - (x(i+m, j+n, c)) est la valeur de l'image d'entrée (x) au pixel décalé par (m) et (n) dans la dimension spatiale (hauteur et largeur) et (c) dans la dimension des canaux. - (w(m,n,c)) est le noyau de convolution pour le canal (c), de taille (k_h ×k_w), et qui est appliqué à l'image (x). - (C) est le nombre de canaux de l'image d'entrée, (k_h) et (k_w) sont respectivement la hauteur et la largeur du noyau.

Explication du Code

La fonction `convolution2d` dans le code prend en entrée une image (input) de dimensions ((h, w, c, n)) où :

- (h) et (w) sont la hauteur et la largeur de l'image.
- (c) est le nombre de canaux (1 pour des images en niveau de gris).
- (n) est le nombre d'images dans le lot (batch).

De plus, la fonction prend un noyau (kernel) de dimensions ((k_h, k_w, c)), où (k_h) et (k_w) sont la hauteur et la largeur du noyau, et (c) est le nombre de canaux du noyau, qui doit correspondre au nombre de canaux de l'image d'entrée.

Le processus de convolution s'effectue par les étapes suivantes :

1. **Vérification des dimensions** : Nous vérifions si le nombre de canaux de l'image d'entrée (c) et celui du noyau (kc) sont égaux. Si ce n'est pas le cas, une erreur est levée.
2. **Calcul de la taille de sortie** : La taille de la sortie

Max Pooling

Le `max_pooling` est appliqué après chaque couche de convolution pour réduire la taille de la carte de caractéristiques tout en conservant les informations importantes. Cela permet de réduire la complexité du modèle et d'améliorer son efficacité en extrayant les informations les plus significatives.

Modèle Complet

Le modèle complet combine ces opérations de convolution, max pooling et couches denses pour effectuer une classification des chiffres manuscrits.

Entraînement du Modèle

Le modèle a été entraîné en utilisant la descente de gradient pour minimiser la fonction de perte (cross-entropy). La fonction d'entraînement `train_model` itère sur les données d'apprentissage pendant plusieurs époques et ajuste les poids du modèle pour améliorer les prédictions. L'accuracy (précision) du modèle est évaluée à chaque époque sur les données de test.

Évaluation du Modèle

Après l'entraînement, nous avons utilisé plusieurs métriques pour évaluer les performances du modèle, telles que la précision, le rappel et le F1-score. Nous avons également calculé la matrice de confusion pour visualiser la capacité du modèle à classer correctement ou incorrectement les images.

Résultats et Visualisations

Nous avons obtenu une précision de 98% pour la classification des chiffres, ce qui montre que le modèle fonctionne bien pour le jeu de données MNIST.

Conclusion

Dans ce projet, nous avons créé un modèle de réseau de neurones convolutifs en utilisant des fonctions manuelles pour mieux comprendre les mécanismes de base. Les résultats obtenus montrent une précision élevée, indiquant que notre modèle est capable de classer correctement les images de chiffres manuscrits. Les visualisations obtenues permettent également de mieux comprendre la performance du modèle. Ces connaissances peuvent être étendues à d'autres domaines nécessitant l'analyse d'images, telles que la reconnaissance faciale ou la détection d'anomalies dans les systèmes de surveillance.

Rshiny

Introduction à RShiny

RShiny est un framework open-source de la bibliothèque R, qui permet de créer des applications web interactives directement à partir de R. Grâce à RShiny, il est possible de construire des applications visuelles et dynamiques, en facilitant la mise en œuvre de modèles statistiques, d'analyses de données et de visualisations interactives.

Shiny simplifie la création d'interfaces utilisateur (UI) et de serveurs web grâce à une syntaxe simple et intuitive, permettant aux utilisateurs de R, même ceux avec peu de compétences en développement web, de créer des applications web efficaces. Les applications Shiny peuvent être déployées localement ou publiées sur des serveurs en ligne comme `shinyapps.io` ou un serveur Shiny hébergé par un utilisateur.

Les principales caractéristiques de RShiny incluent :

- **Interactivité** : Les utilisateurs peuvent interagir avec les graphiques, tableaux et autres visualisations, ce qui permet d'explorer les données de manière dynamique.
- **Flexibilité** : Shiny permet de créer des interfaces utilisateurs complexes qui intègrent des graphiques, des tables, des formulaires, et bien d'autres composants.
- **Simplicité** : Grâce à sa syntaxe facile à comprendre, il est possible de transformer rapidement un script R en une application interactive, sans nécessiter de connaissances approfondies en développement web (comme HTML, CSS ou JavaScript).
- **Extensibilité** : RShiny est compatible avec de nombreux autres packages R pour l'analyse de données, comme ggplot2, plotly, leaflet, et bien plus encore, permettant une large gamme de visualisations et de traitements des données.

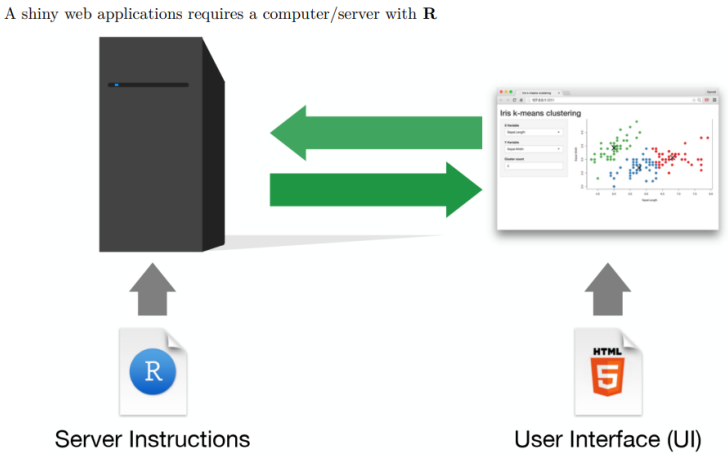


Figure 10: Fonctionnement de Rshiny

Tâches et Contributions de chaque membre

Dans le cadre de notre projet, chaque membre du groupe a contribué à une partie spécifique de RShiny. Voici les détails de chaque contribution :

Contribution 1 :

Le travail réalisé se divise en plusieurs étapes essentielles, dont l'entraînement du modèle, le stockage de ce modèle entraîné, et l'intégration avec RShiny via l'utilisation de Rulia.

Entraînement du Modèle

On a d'abord réccupérer notre modèle entraîné . Le processus d'entraînement a été effectué une seule fois pour obtenir un modèle optimisé qui sera ensuite utilisé pour faire des prédictions dans l'application RShiny.

Stockage du Modèle

Une fois l'entraînement terminé, le modèle a été stocké dans un fichier `.jld2`, ce qui permet de le sauvegarder pour une utilisation future. Le format `.jld2` est utilisé pour stocker des objets Julia de manière compacte et efficace.

Vérification de l'Appel du Modèle

Ensuite on a vérifié la possibilité d'appeler le modèle à partir du fichier `.jld2`. Cette étape est cruciale pour garantir que le modèle peut être utilisé dans l'application sans nécessiter un nouvel entraînement.

Installation de Rulia pour l'Intégration entre R et Julia

Afin de faciliter l'intégration entre R et Julia,on a installé le package `Rulia`. Ce package permet de faire le lien entre les deux langages et d'exécuter du code Julia directement depuis R. L'installation et la configuration de `Rulia` ont permis de rendre l'interaction entre l'application Shiny et le modèle Julia fluide et efficace.

Création de l'Application RShiny

Une fois le modèle chargé et l'intégration avec Julia mise en place, on a créé une application RShiny nommée `app.R`. Cette application permet à l'utilisateur de dessiner un chiffre via une interface graphique, de faire appel au modèle, et de recevoir la prédiction en retour.

matrix

L'application Shiny présente plusieurs éléments interactifs, tels qu'un champ de saisie pour l'utilisateur et un bouton pour lancer la prédiction. Lorsqu'un utilisateur entre un chiffre et clique sur le bouton, l'application appelle le modèle stocké dans le fichier `.jld2`, effectue la prédiction et affiche le résultat à l'utilisateur.

matrix ## Prédiction via le Modèle

Lorsque l'utilisateur entre un chiffre dans l'interface, l'application envoie cette valeur à Julia, qui utilise le modèle pour effectuer la prédiction. Le modèle prédit un chiffre en fonction des données d'entrée, et cette prédiction est ensuite renvoyée à l'application Shiny pour être affichée à l'utilisateur. Cette étape utilise le modèle stocké dans le fichier `.jld2`.

Figure 11: Processus de prédiction dans l'application Shiny

Ainsi, on a réussi à créer une chaîne complète allant de l'entraînement du modèle à son intégration dans une application web interactive. Le processus commence par l'entraînement du modèle, suivi de son stockage, de son appel via Julia, et de l'intégration de la fonctionnalité de prédiction dans une application RShiny. Cette approche a permis de rendre l'application dynamique, en permettant aux utilisateurs de réaliser des prédictions en temps réel.

Contribution 2 :

Suite à notre motivation de rendre l'évaluation de notre modèle plus interactive et accessible, nous avons décidé d'intégrer un tableau de bord interactif dans notre application `RShiny`.

On a pris en charge l'utilisation des résultats des quatre métriques (précision, rappel, F1-score, etc.) obtenus lors de l'évaluation des modèles. Le travail a consisté à afficher ces résultats sous forme d'un tableau de bord interactif, permettant aux utilisateurs de visualiser facilement la performance du modèle.

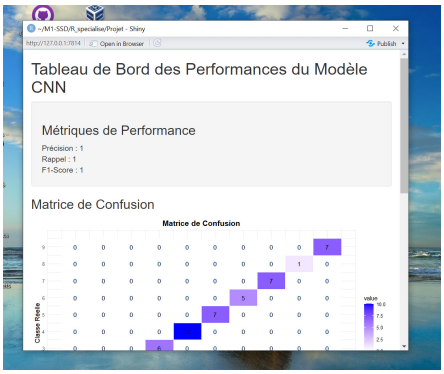


Figure 12: l'application en RShiny

Calcul des Métriques

Les métriques utilisées pour évaluer la performance du modèle sont essentielles pour comprendre sa capacité à faire des prédictions correctes. On a extrait les résultats de quatre métriques principales, telles que la précision, le rappel, le F1-score, et d'autres, afin de donner une vue d'ensemble de la performance du modèle. Ces résultats sont calculés sur la base des prédictions réalisées par le modèle sur les données de test.

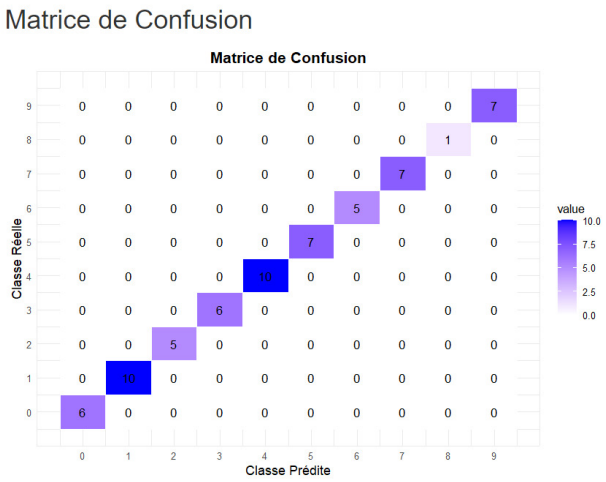


Figure 13: Exemple de calcul des métriques de performance

Création du Tableau de Bord

Une fois les métriques calculées, On a créé un tableau de bord interactif pour afficher les résultats de manière claire et concise. Le tableau de bord permet aux utilisateurs de voir les valeurs de chaque métrique pour chaque modèle testé, facilitant ainsi l'analyse comparative des performances des différents modèles.

Ce tableau de bord est intégré à l'application principale et permet aux utilisateurs d'interagir avec les résultats.

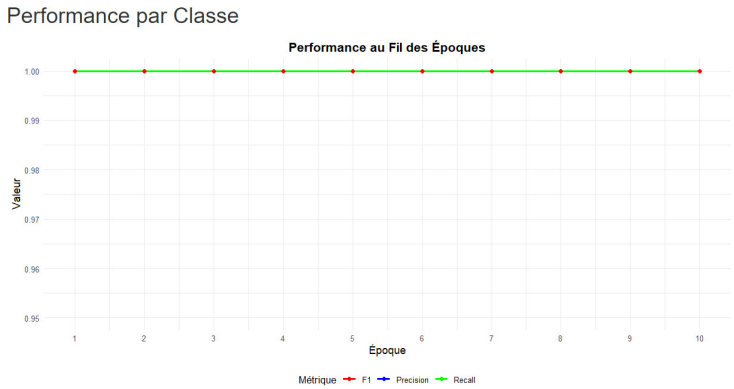


Figure 14: Tableau de bord affichant les résultats des métriques

Affichage Interactif des Résultats

Le tableau de bord créé offre des fonctionnalités interactives qui permettent de visualiser les résultats de manière dynamique. Les utilisateurs peuvent, par exemple, ajuster des filtres pour afficher

les métriques de performance d'un modèle particulier ou comparer les résultats entre plusieurs modèles.

En résumé, on a effectué l'intégration des résultats des métriques de performance dans un tableau de bord interactif, permettant de visualiser les performances du modèle de manière claire et intuitive. Ce tableau de bord est un outil précieux pour les utilisateurs, car il facilite l'analyse et la comparaison des différents modèles en fonction de leurs performances sur plusieurs métriques.

Contribution 3:

Application Shiny pour la Visualisation des Données MNIST

Dans cette section, on présente une application Shiny développée pour la visualisation et l'exploration des données MNIST. Cette application permet de filtrer les images par classe (chiffre manuscrit de 0 à 9) et de générer deux types de visualisations interactives : un histogramme de la distribution des valeurs des pixels et un nuage de points représentant la répartition spatiale des pixels dans l'image.

Objectifs de l'Application

L'objectif principal de l'application est de fournir un moyen interactif pour explorer les images MNIST. L'utilisateur peut : - Sélectionner une ou plusieurs classes de chiffres (0-9) pour visualiser les images correspondantes. - Afficher un **histogramme** représentant la distribution des valeurs des pixels des images filtrées. - Afficher un **nuage de points** représentant la position des pixels dans l'image.

Filtrage par Classe

L'application permet à l'utilisateur de filtrer les images par classe, ce qui signifie qu'il peut choisir de visualiser uniquement les images d'une certaine classe de chiffres. Le filtre utilise un menu déroulant où l'utilisateur peut sélectionner plusieurs classes à la fois.

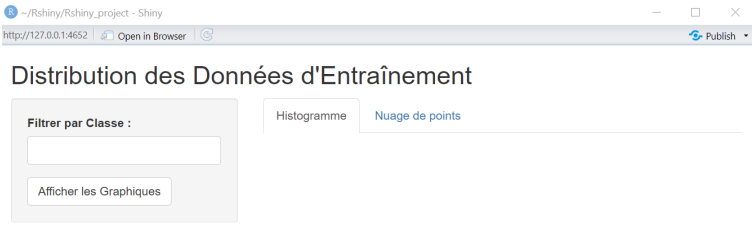


Figure 15: menu du filtre

Histogramme des Pixels

L'histogramme montre la **répartition des valeurs des pixels** pour les images de la classe sélectionnée. Les pixels d'une image ont des valeurs d'intensité allant de 0 (blanc) à 255 (noir). Cet histogramme permet de visualiser la distribution des intensités dans les images, et ainsi, comprendre la nature de l'éclairage et du contraste des pixels de chaque classe.

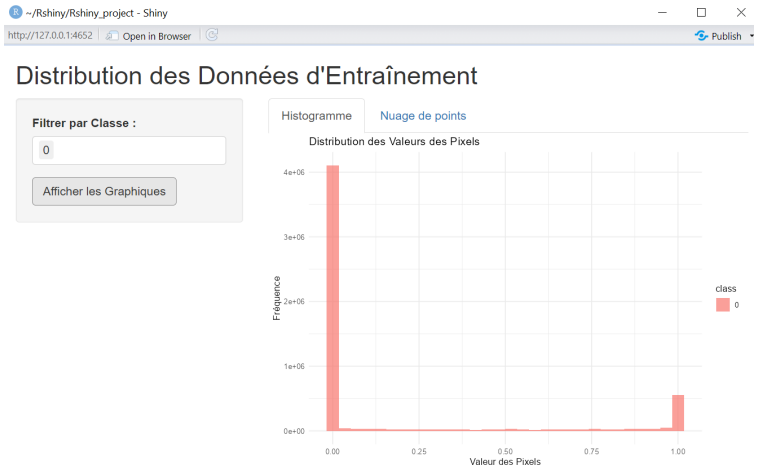


Figure 16: répartition des valeurs des pixels

Commentaire sur l'Histogramme

- L'histogramme permet d'observer la répartition des pixels sombres et clairs dans les images. Par exemple, pour le chiffre "0", on peut voir une concentration de pixels sombres aux bords de l'image, tandis que le centre de l'image peut être plus clair.

Nuage de Points

Le nuage de points montre la **répartition spatiale des pixels** dans l'image. Chaque point correspond à un pixel, et sa couleur est déterminée par l'intensité du pixel, ce qui permet de visualiser comment les pixels sont disposés dans l'image en fonction de leur position dans les coordonnées x et y.

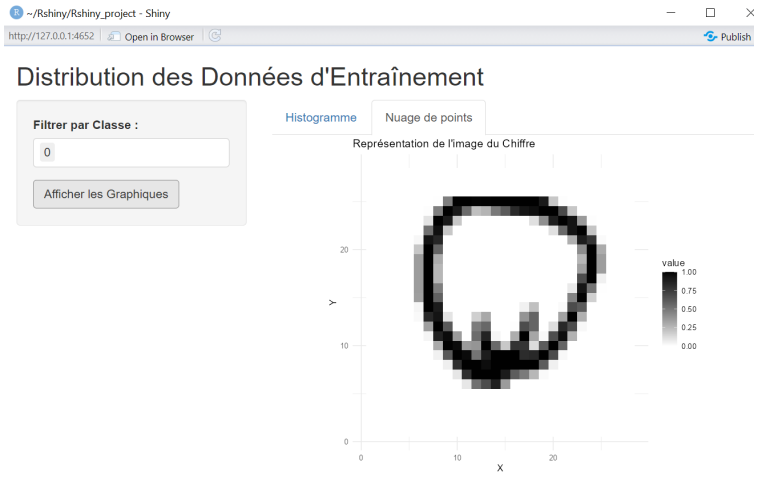
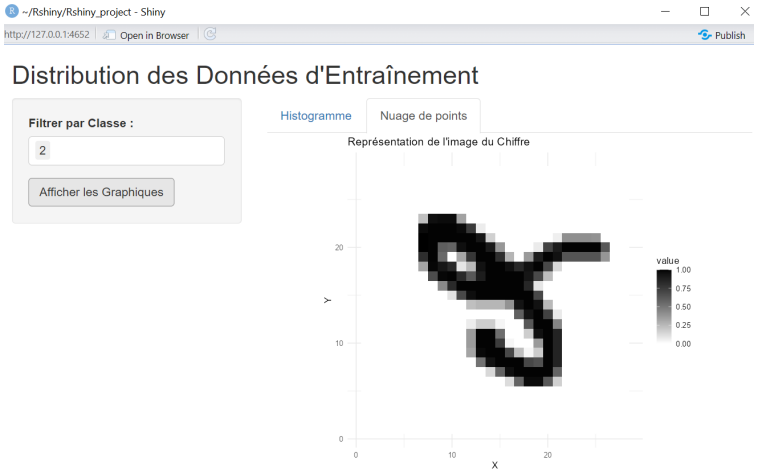


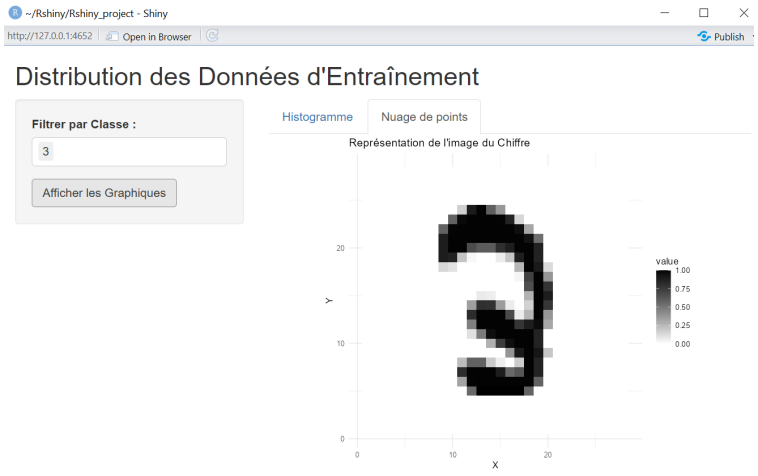
Figure 17: nuage des points

Commentaire sur le Nuage de Points

- Le nuage de points permet de visualiser la forme générale du chiffre. Par exemple, pour le chiffre “0”, les points sont disposés de manière circulaire, reflétant la forme du chiffre manuscrit.

Quelques autres exemples de test:





Conclusion

L'application Shiny permet une exploration interactive des images MNIST, facilitant la visualisation de la distribution des valeurs des pixels ainsi que leur disposition spatiale. Cette approche permet une meilleure compréhension des caractéristiques visuelles des chiffres manuscrits, ce qui est essentiel pour les tâches de classification d'images.

Commentaires générales

Pour l'exemple d'Image MNIST

Pour valider le chargement des données, une image d'exemple de la classe "0" a été affichée (voir code R fichier test). Cela permet de vérifier que la représentation des pixels et la reconstruction de l'image sont correctes.

À propos du package julia

Le fichier src_ver1.jl contient le code utilisant les fonctions du package Flux, tandis que le fichier src_ver2.jl présente le code où les fonctions des réseaux de neurones convolutifs (CNN) sont écrites manuellement. Voici les principales observations et améliorations apportées :

1. Vérification et suppression des impressions intermédiaires

- J'ai initialement ajouté des impressions print pour vérifier que les étapes intermédiaires fonctionnaient correctement, notamment les dimensions des images après chaque couche de convolution et de max-pooling.

- Cependant, ces impressions ralentissaient considérablement l'exécution du code. Elles ont été supprimées une fois les vérifications terminées.

2. Écriture manuelle et conception des fonctions CNN

- La partie la plus complexe n'était pas seulement d'écrire manuellement les fonctions nécessaires pour les CNN, mais également de les concevoir de manière à reproduire le fonctionnement de Flux.
- Cela a permis d'intégrer ces fonctions dans un modèle d'entraînement CNN utilisé pour classifier les images du dataset MNIST.

3. Dynamisation de la fonction `load_prepare_data`

- Au départ, la fonction `load_prepare_data()` était statique et spécifique au dataset MNIST. Pour rendre le code plus dynamique, elle a été modifiée en : julia : load_prepare_data(train_X, train_y, test_X, test_y; num_train, num_test)

Cette nouvelle version permet de charger et préparer n'importe quel dataset, avec des options pour limiter le nombre d'exemples utilisés (`num_train` et `num_test`). Cela est particulièrement utile pour éviter de tester sur les 60 000 images de MNIST, car les fonctions écrites manuellement demandent plus de temps.

4. Utilisation du multiple dispatch

- La fonction `metriques` illustre bien le multiple dispatch. Elle dispose de plusieurs méthodes, Julia choisissant dynamiquement celle correspondant aux types des arguments fournis.
- Cette flexibilité rend le code plus performant et expressif, en exploitant pleinement l'une des principales forces de Julia.

5. Convention de nommage des fonctions et variables

- Les noms de fonctions et de variables étaient initialement écrits en `snake_case` (exemple : `train_loader`, `compute_accuracy`), car je suis habituée à coder en Python.
- Cependant, bien que ce style soit autorisé dans Julia, il est moins courant. Julia privilégie le `CamelCase` pour les noms de types et modules, et recommande de suivre cette convention.
- J'aurais renommé les fonctions et variables pour adopter le style `CamelCase`, conformément aux standards des bibliothèques Julia, comme Flux.

6. Commentaires dans le code

- Tous les commentaires ont été rédigés en anglais afin de garantir que le code soit compréhensible par un public international.

7. Optimisation des performances

- Le code a été optimisé pour améliorer les temps d'exécution, notamment grâce à l'utilisation de la parallélisation avec `@threads` pour les boucles.
- Julia offre une excellente gestion du parallélisme, ce qui a permis d'accélérer les calculs.
- J'ai également réduit la taille du dataset pour des tests rapides. Plutôt que d'utiliser les 60 000 images de MNIST, j'ai travaillé sur des sous-ensembles plus petits, ce qui permet de valider la logique sans attendre trop longtemps.

8. Résultats des tests

Pour valider les performances, j'ai testé le modèle avec différents nombres d'images et j'ai opté pour des paramètres (`epochs = 10`, `batch_size = 64`) : - 500 images : 2 minutes, précision de 73,6 % - 1000 images : 4 minutes, précision de 77,1 % - 2500 images : 12 minutes, précision de 79,4 % - 5000 images : 38 minutes, précision de 84 %

9. Observations :

- L'augmentation de la taille du dataset améliore la précision, mais augmente également les temps d'exécution.
- Cette lenteur est attendue pour plusieurs raisons :
  - Absence de GPU sur ma machine, ce qui limite la vitesse d'entraînement.
  - Les fonctions écrites manuellement sont moins optimisées que celles des bibliothèques comme Flux.
  - Le traitement parallèle n'est pas suffisant pour compenser l'augmentation des données.

Conclusion

Les améliorations apportées au fichier `src_ver2.jl` permettent : - Une meilleure modularité grâce à des fonctions dynamiques comme `loadPrepareData`. - Une optimisation des performances via la parallélisation. - Une précision croissante avec des ensembles de données plus importants, bien que les temps d'exécution restent élevés en l'absence de GPU.

Ces efforts ont permis de rendre le code plus robuste, flexible et en phase avec les bonnes pratiques en programmation Julia.

Mécanisme d'Entraînement

Tout d'abord, nous avons mis en place un mécanisme d'entraînement où le modèle est entraîné sur les données d'entraînement. Nous utilisons une fonction `entrainement_cp()` qui gère l'entraînement par lots, avec un `batch_size` de 64.

En plus de cela, nous avons ajouté une fonctionnalité de sauvegarde du modèle à chaque époque à l'aide de la fonction `jldsave()`. Cela permet de conserver l'état du modèle après chaque itération, afin de pouvoir analyser ou reprendre l'entraînement à tout moment, avec le code suivant :

julia: model_state = Flux.state(model) jldsave("mymodel_epoch_$epoch.jld2", model_state)

Cela est essentiel pour sauvegarder non seulement le modèle final, mais aussi les versions intermédiaires de celui-ci, au cas où nous voudrions tester différentes configurations ou reprendre l'entraînement après un arrêt.

Prédiction Finale

À la fin de l'entraînement, nous utilisons la fonction `prediction_final()` pour effectuer des prédictions sur un sous-ensemble des données de test. Cette étape permet d'évaluer la capacité du modèle à prédire correctement les classes des images non vues lors de l'entraînement.

Points à Développer pour D'autres Problèmes pour Généraliser les Projets

- **Dimension des Entrées** : Le modèle est configuré pour des images de dimension (28, 28, 1), ce qui est spécifique aux données comme MNIST. Pour le rendre générique et utilisable avec d'autres types de données, il serait nécessaire d'adapter la dimension d'entrée en fonction des données spécifiques. Par exemple, pour des images RGB de taille 64x64, il serait nécessaire d'ajuster la fonction de redimensionnement à la forme (64, 64, 3, N). Ce mécanisme peut être rendu dynamique en permettant à l'utilisateur de spécifier la taille des images et le nombre de canaux (par exemple, en passant ces informations en arguments à la fonction de création du dataset).
- **Adaptation du Code pour Différents Modèles et Types de Données** : Le code peut être facilement réutilisé pour différents modèles et entrées. Par exemple, pour un modèle de classification de texte, il suffira de remplacer le modèle CNN par un modèle RNN ou Transformer, et d'adapter les entrées à des séquences de texte au lieu d'images.

Pour rendre le code encore plus générique, voici quelques suggestions : 1. Permettre à l'utilisateur de spécifier la taille d'entrée des données (`input_size`), le nombre de canaux (pour des images RGB, par exemple), et la taille des lots. 2. Modifier le modèle en fonction du type de données (par exemple, CNN pour des images, RNN pour des séquences de texte). 3. Intégrer un mécanisme de pré-traitement des données (scaling, normalisation, tokenisation, etc.) selon le type de données.

Cela permettrait à ce code de s'adapter à différents types de tâches (classification d'images, classification de texte, etc.).

Conclusion et Perspectives d'Amélioration sur Notre Modèle

Ce code est une base solide pour entraîner des modèles de machine learning, en particulier pour la classification d'images. En adaptant la gestion des données et la structure du modèle, il peut être réutilisé pour une variété de tâches et de datasets différents. Les principaux points à développer seraient l'adaptabilité des dimensions d'entrée, la possibilité de traiter différents types de données, et l'amélioration de la gestion des hyperparamètres pour optimiser les performances du modèle.

Enfin, en ajoutant des fonctionnalités comme l'évaluation après chaque époque (par exemple, en calculant la précision sur les données de validation) et l'optimisation des hyperparamètres, ce code pourrait encore être amélioré en termes de performance et de flexibilité.

Application et Intégration du Code R pour la Prédiction

L'intégration de l'application R a été un travail essentiel pour simplifier l'interface utilisateur et rendre le processus de prédiction plus accessible. Grâce à cette application, l'utilisateur peut désormais effectuer des prédictions de manière fluide, sans avoir à se soucier du chemin du modèle ou du processus de chargement. Cette automatisation et cette centralisation des fonctionnalités sont des améliorations majeures qui rendent l'ensemble du processus beaucoup plus simple et efficace.

En gros, ce que fait notre application :

1. L'utilisateur dessine un chiffre sur un canevas.
2. L'image est capturée et envoyée à R sous forme de données base64.
3. R décode l'image, la redimensionne à 28x28 pixels et la sauvegarde.
4. La fonction de prédiction en Julia est appelée (`Rulia`) pour prédire le chiffre basé sur l'image traitée avec le modèle entraîné et stocké dans le fichier `.jld2`.
5. La classe prédite est affichée dans la console Julia.

Conclusion

Ce projet a été une aventure fascinante dans l'univers des réseaux de neurones, où nous avons exploré leur puissance et leurs implications concrètes dans le monde actuel. À travers cette expérience, nous avons non seulement approfondi nos connaissances théoriques sur ces modèles, mais également découvert leur lien étroit avec les défis et les opportunités de notre société moderne :

de la reconnaissance d'images à l'automatisation, ces outils transforment le monde qui nous entoure.

Nous avons également eu l'opportunité de travailler avec plusieurs langages de programmation tels que Python, R et Julia. Si chacun d'eux a ses points forts, ce projet a clairement mis en évidence la supériorité de Julia en termes de performances et d'efficacité. En nous plongeant dans Julia, nous avons compris à quel point le langage pouvait simplifier les tâches complexes tout en offrant une rapidité d'exécution impressionnante, un atout indéniable pour des projets de grande envergure.

Ce travail a été bien plus qu'un simple exercice académique pour nous. Il a éveillé une réelle passion pour la recherche et le développement dans le domaine des technologies de l'intelligence artificielle. C'est avec une curiosité et une ambition renouvelées que nous envisageons de poursuivre dans cette voie, convaincus que les compétences et les enseignements tirés de cette expérience nous serviront dans nos futures réalisations.

Enfin, ce projet a également été une leçon sur la collaboration, la résilience et l'importance de rester ouverts aux nouvelles technologies. Nous sommes fières d'avoir pu allier innovation technique et créativité dans une solution accessible et pratique grâce à l'intégration de RShiny. Ce travail n'est pas une fin, mais un point de départ pour explorer davantage les possibilités infinies offertes par ces outils extraordinaires.

Cette expérience a renforcé notre conviction que l'apprentissage ne s'arrête jamais, et que chaque ligne de code est une nouvelle étape vers la réalisation de nos rêves technologiques.

Enfin, nous tenons à remercier sincèrement notre professeur, M. Rémi, pour son encadrement et son dévouement tout au long de ce projet. Sa disponibilité, ses conseils précieux et sa passion pour l'enseignement nous ont guidées et motivées à donner le meilleur de nous-mêmes.