Compte rendu du TP 2 – 2024/2025
Étudiant : FALL Madou, classe M2 TSI

Rapport de TP : MNIST Multi class classification with MLP

Introduction
Ce rapport présente une analyse du code fourni pour la classification des chiffres manuscrits du jeu de données MNIST à l'aide d'un réseau de neurones multi-couches (MLP). L'objectif est de comprendre le fonctionnement du code, des bibliothèques utilisées et du processus d'entraînement du modèle.

Bibliothèques utilisées
Les bibliothèques suivantes sont utilisées pour le traitement des données, la création du modèle et l'affichage des résultats :

- **NumPy** : Manipulation efficace des tableaux de données numériques.
- **Pandas** : Gestion et affichage des jeux de données sous forme de DataFrame.
- **Matplotlib (pyplot)** : Visualisation des données sous forme de graphiques.
- **TensorFlow/Keras** : Création et entraînement des réseaux de neurones profonds.
  - layers : Contient des couches prédéfinies pour les modèles.
  - Conv2D, Conv2DTranspose : Couches de convolution utilisées pour l'extraction de caractéristiques.
  - BatchNormalization : Améliore la convergence du modèle.
  - MaxPooling2D : Réduction de la taille des images pour extraire les caractéristiques principales.
  - max_norm : Contrainte sur les poids du réseau pour éviter l'explosion des gradients.
  - backend (K) : Interface pour interagir avec TensorFlow à bas niveau.
- **OpenCV (cv2)** : Traitement et affichage des images.
- **Google Colab patches (cv2_imshow)** : Affichage des images dans un notebook Google Colab.

Paramétrage supplémentaire
- **Options Pandas** : Ajuste l'affichage des lignes et la précision des nombres flottants.
- **Options NumPy** : Améliore la lisibilité des tableaux affichés.

Ces outils permettent de prétraiter les images, d'entraîner un réseau de neurones et d'évaluer ses performances.

Chargement du jeu de données MNIST
Le jeu de données est chargé avec la commande suivante :

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

Description du jeu de données
- MNIST contient **70 000 images** de chiffres manuscrits en niveaux de gris (0-255).
- Chaque image a une **taille de 28x28 pixels**.
- Les étiquettes (y_train et y_test) sont des entiers de **0 à 9** correspondants aux chiffres représentés.

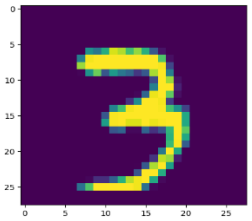
Répartition des données

- **60 000 images d'entraînement** → `(x_train, y_train)`
- **10 000 images de test** → `(x_test, y_test)`

Visualisation des données

Quelques commandes permettent d'examiner visuellement des images spécifiques du jeu MNIST :

```
x_train[2917]       # Affiche l'image à l'indice 2917 (matrice 28x28)
```



```
x_train[2917][10]   # Accède à la 10e ligne de cette image (tableau de 28 pixels)
      array([0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
      58, 254, 216, 11,  0,  0,  0,  0,  0,  0,  0,  0], dtype=uint8)
```

`x_train[2917][10][16]` # Accède au 16e pixel de la 10e ligne de l'image (valeur entre 0 et 255), ici c'est : `0`

Prétraitement des données

Le code applique une **normalisation** pour améliorer la performance du modèle :

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Cela ramène les valeurs des pixels dans l'intervalle [0, 1], facilitant ainsi l'apprentissage du réseau.

Fonction de visualisation

Une fonction `plot_curve` est définie pour tracer les courbes d'évolution des métriques d'entraînement (précision et perte).

Création du modèle de classification

Le modèle est défini dans la fonction `create_model(my_learning_rate)`, qui :

- **Crée un réseau de neurones** avec :
  - Une couche d'entrée (aplatissement des images 28x28 en un vecteur de 784 valeurs).
  - Une couche cachée de **256 neurones** avec activation **ReLU**.
  - Une régularisation **Dropout (20%)** pour éviter le sur-apprentissage.
  - Une couche de sortie de **10 neurones** avec activation **Softmax** pour la classification.
- **Compile le modèle** avec :
  - L'optimiseur `Adam`.
  - La fonction de perte `sparse_categorical_crossentropy`.
  - La métrique `accuracy`.

Entraînement du modèle

La fonction `train_model(model, train_features, train_labels, epochs, batch_size, validation_split)` :

- Entraîne le modèle sur les données normalisées pendant **50 époques**.
- Enregistre l'historique des métriques (accuracy, loss) à chaque époque.
- Utilise **20% des données** pour la validation.

Suivi des performances
À la fin de l'entraînement, un graphique affiche l'évolution de la précision (accuracy) et de la précision de validation (val_accuracy).

Évaluation du modèle
Le modèle est ensuite évalué sur les données de test pour mesurer sa performance sur des données jamais vues.

Prétraitement des images pour la prédiction
Le code implémente plusieurs fonctions pour préparer de nouvelles images à être classifiées par le modèle :

- `Rescale_Image(img)` : Met à l'échelle l'image pour que les pixels soient entre **0 et 1**.
- `load_image(File_Name, K)` :
    1. Charge une image depuis un fichier.
    2. Convertit l'image en **niveaux de gris**.
    3. Redimensionne l'image à **28x28 pixels**.
    4. Met l'image à l'échelle et l'adapte au format d'entrée du modèle.

Ces fonctions garantissent que les images fournies au modèle respectent le même format que les données MNIST.

Prédiction avec le modèle
La fonction `Predict_Class(test_image)` permet de :

- Charger une image et la normaliser.
- Effectuer une prédiction avec le modèle.
- Retourner et afficher la **classe prédite** ainsi que les **probabilités associées**.

Expérimentation avec Google Drive
Le code permet d'accéder à Google Drive depuis Google Colab pour :

- Charger plusieurs images.
- Effectuer des prédictions sur ces images avec le modèle entraîné.
- Comparer les résultats avec les classes attendues.
- Tester l'impact du bruit sur les images (prédictions avant et après débruitage).

Conclusion
Ce projet met en œuvre un modèle de réseau de neurones pour la classification des chiffres manuscrits du jeu MNIST. Le processus inclut :

- Le chargement et la préparation des données.
- La création et l'entraînement d'un modèle MLP.
- L'évaluation et l'utilisation du modèle sur de nouvelles images.

Ce travail permet de mieux comprendre les concepts fondamentaux des réseaux de neurones et du traitement d'images en deep learning.

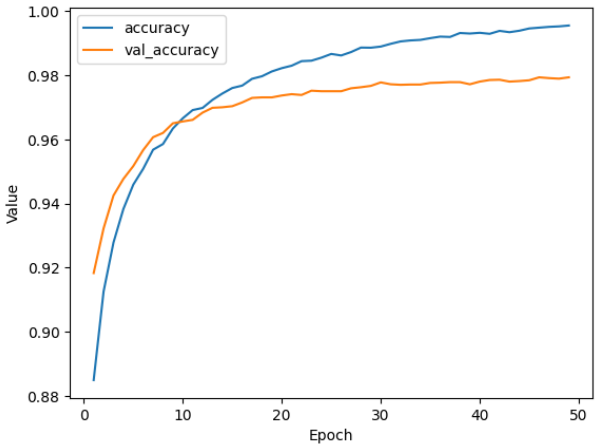
Hyperparamètres
Les paramètres utilisés pour l'entraînement du modèle sont les suivants :

- **Taux d'apprentissage** : 0.003
- **Nombre d'époques** : 50
- **Taille des lots** : 4000
- **Fraction de validation** : 20 %

Analyse des Résultats
Le modèle a été évalué sur plusieurs échantillons et a montré une bonne capacité de classification sur des images propres. Cependant, une image bruitée a entraîné une erreur de prédiction. Après débruitage, la classe correcte a été restaurée, mettant en évidence l'efficacité du prétraitement.

Image	Classe Attendue	Prédiction	Observations
Image 1	2	2	✅ Prédiction correcte
Image 2	7	7	✅ Prédiction correcte
Image 3	0	0	✅ Prédiction correcte
Image 4 (bruitée)	0	5	⚠ Erreur due au bruit
Image 5 (débruitée)	0	0	🔄 Correction après débruitage

Courbes d'entraînement
L'évolution de l'**accuracy** et de l'**accuracy de validation** pendant l'entraînement est représentée dans le graphique ci-dessous :



Observations :
• L'**accuracy** atteint environ 99 %, indiquant une bonne convergence du modèle.
• L'**accuracy de validation** se stabilise à 97 %, ce qui montre que le modèle apprend efficacement.
• Aucune tendance évidente de **surapprentissage** n'est observée, la courbe de validation restant proche de celle de l'entraînement.

Conclusions principales
✅ **Bonne performance sur les images propres** : le modèle classe correctement les données non altérées.
⚠ **Sensibilité au bruit** : la classification est perturbée par le bruit, entraînant des erreurs.
🔄 **Efficacité du débruitage** : le traitement des images bruitées permet de restaurer la prédiction correcte.