Compte rendu du TP 3 – 2024/2025
Étudiant : FALL Madou, classe M2 TSI

Rapport de TP : Autoencodeur Débruitage - Modèles de Bruit

Introduction
Ce rapport présente l'implémentation et l'analyse d'un autoencodeur convolutionnel destiné à la débruitisation d'images issues du dataset MNIST. L'objectif est d'entraîner un modèle capable de reconstruire les images originales à partir de leur version bruitée. Le projet s'inscrit dans le cadre de l'apprentissage profond (deep learning) et vise à explorer les capacités des autoencodeurs pour la réduction de bruit dans les images.

Importation des Bibliothèques
Le script utilise plusieurs bibliothèques essentielles pour l'apprentissage automatique et le traitement d'images :
- **TensorFlow & Keras** : Pour définir et entraîner le modèle.
- **NumPy & Pandas** : Pour la manipulation de données.
- **Matplotlib** : Pour la visualisation des images et des performances.
- **OS** : Pour la gestion des fichiers et dossiers.

Prétraitement des Images
a. Chargement des images
Deux fonctions sont utilisées pour charger les images et les prétraiter :
- `load_img_old(path_to_img)` : Utilise TensorFlow pour charger, normaliser et redimensionner les images.
- `load_img(path_to_img)` : Utilise Matplotlib pour charger les images, avec une normalisation simple.

b. Fonction `imageShowFormat(img, image_size)`
Cette fonction reconvertit une image normalisée en un format standard en multipliant les valeurs par 255 et en les transformant en entier.

Architecture du Modèle
a. Encodeur
- 2 couches **Conv2D** pour extraire les caractéristiques.
- **Flatten** et **Dense** pour réduire la dimensionnalité.

b. Décodeur
- **Dense** suivi de **Reshape** pour restaurer la structure spatiale.
- 2 couches **Conv2DTranspose** pour reconstruire l'image.

Ajout de Bruit
Le bruit est ajouté aux images d'entraînement et de test en utilisant :
- `np.random.normal()` : Bruit gaussien.
- `np.random.poisson()` : Bruit de Poisson.

Entraînement du Modèle
L'entraînement est réalisé avec :
- **EarlyStopping** : Arrête l'entraînement si la performance stagne.
- **ModelCheckpoint** : Sauvegarde le modèle optimal.

Visualisation des Résultats
a. `Plot_Normalized_Images`
Affiche 8 paires d'images originales et bruitées.

b. `Plot_Result_Images`
Compare les images originales, bruitées et débruitées par le modèle.

c. `plot_hist`
Affiche les courbes d'apprentissage (accuracy et loss).

Fonctions Mathématiques et Optimisation

- Fonctions Mathématiques
  - `sigmoid(x, A, C)` : Calcule la valeur de la fonction sigmoïde.
  - `Gauss_Function(x, Gauss_Shift)` : Calcule la valeur d'une fonction gaussienne centrée en Gauss_Shift.
  - `Sigmoid_Function(x, Parameters, a)` : Combine une sigmoïde et une gaussienne pour créer une fonction de mappage.
  - `Power_Function(x, Power_Param_Vector, a)` : Combine une fonction puissance et une gaussienne pour créer une autre fonction de mappage.

- Création des Paramètres
  - `Create_Parameters_Ternary_GWO()` : Initialise les paramètres pour le mappage ternaire (OptionMap et OptionBinary).

- Mappage Ternaire
  - `Ternary_Map(y, a, OptionMap, OptionBinary)` : Utilise les fonctions de mappage (**Sigmoid** ou **Power**) pour transformer une valeur continue y en une valeur discrète (0, 1 ou 2). Si **OptionBinary** est activé, les valeurs sont binaires (0 ou 1).

- Calcul de la Contribution du Leader
  - `Compute_Leader_Contribution(Positionsij, Leader_posj, a)` : Calcule la contribution du leader (Alpha, Beta ou Delta) pour mettre à jour la position d'un loup.

- Sélection du Leader et Contribution
  - `Leader_Selection_and_Contribution(...)` : Sélectionne le leader (par défaut, Alpha) et calcule sa contribution pour mettre à jour la position d'un loup.

- Mise à Jour des Positions
  - `Update_Position_TGWO(...)` : Met à jour la position d'un loup en combinant les contributions de deux leaders et en appliquant le mappage ternaire.

- `Update_Position_ContinuousGWO` : Met à jour la position d'un loup dans l'espace de recherche en utilisant les positions des leaders (Alpha, Beta, Delta) et deux positions supplémentaires (rho1 et rho2).

Classe Solution

La classe solution sert à encapsuler toutes les informations relatives à une exécution d'un algorithme d'optimisation. Elle permet de :
- Stocker les résultats (meilleure solution, convergence, etc.).
- Enregistrer les paramètres de l'optimisation (bornes, dimension, population, etc.).
- Faciliter l'analyse et la comparaison des performances de différents algorithmes.

Fonctions Supplémentaires

- `dynamic(r, x)` :
  - Calcule la prochaine valeur de la séquence en utilisant l'équation de la `logistic map`.
  - Paramètres :
  - **r** : Paramètre de contrôle (dans ce code, fixé à 2.8).
  - **x** : Valeur actuelle de la séquence.
  - Retourne : La valeur suivante xn+1.

- `logistic_map(max_iter)` :
  - Génère une séquence de valeurs en utilisant la logistic map.
  - Paramètres :
  - max_iter : Nombre maximal d'itérations (longueur de la séquence).
  - Retourne : Une liste contenant la séquence générée.

- `CrossOver(X1, X2, X3)` :
  - Sélectionne aléatoirement l'une des trois valeurs X1, X2 ou X3 avec des probabilités égales.

- `Update_Position_BGWO`(...) :
  - Met à jour la position d'un loup pour les variables binaires en utilisant une combinaison de la sigmoïde et des contributions des leaders (Alpha, Beta, Delta).

- `GWO_continuous_ternary_binary(...)` :
  - Fonction principale qui implémente l'algorithme BGWO.
  - Gère les variables continues, ternaires et binaires en utilisant des mécanismes de mise à jour spécifiques.
  - Utilise des séquences chaotiques pour améliorer l'exploration.

Fonctions d'Affichage et de Sauvegarde

- `My_Vector(mini, maxi, step)` :
  - Crée un vecteur colonne de valeurs allant de mini à maxi avec un pas de step.

- `Display_Numerical_Results(…)` :
  - Affiche les résultats numériques d'une optimisation.

- `Display_Visual_Results(…)` :
  - Affiche et sauvegarde un graphique de la courbe de convergence.

Fonctions Principales

- `setup()` :
  - Définit les paramètres de l'optimisation (dimension, bornes, taille de la population, etc.).

- `GWO_continuous_ternary_binary()` :
  - Exécute l'algorithme GWO pour optimiser les paramètres de l'autoencodeur.

- `My_Autoencoder()` :
  - Évalue la performance de l'autoencodeur avec les paramètres donnés.

- `Plot_Result_Images()`:
  - Trace les images originales, bruyantes et débruitées pour visualiser les résultats.

- `run_denoise()` :
  - Exécute un autoencodeur avec des paramètres prédéfinis pour débruiter des images.

- `run_continuous_ternary_binary()` :
  - Orchestre l'exécution d'un algorithme d'optimisation hybride pour optimiser un autoencodeur.

Conclusion

Ce rapport détaille l'implémentation d'un autoencodeur convolutionnel pour le débruitage d'images, ainsi que les méthodes d'optimisation utilisées pour améliorer ses performances. Les résultats montrent que l'autoencodeur est capable de reconstruire efficacement les images originales à partir de versions bruitées, démontrant ainsi son utilité dans les tâches de réduction de bruit.

Les résultats obtenus après avoir compilé le code sont : **accuracy** = 0.8077, **loss** = 0.0234, **val_accuracy** = 0.8086, **val_loss** = 0.0209.