Introduction

Ce Projet est un exemple d'implémentation d'un réseau de neurones convolutifs (CNN) utilisant Keras pour classifier des images de chiffres manuscrits du dataset MNIST. Le code comprend la préparation des données, la construction du modèle, l'entraînement, et la visualisation des métriques de performance.

Préparation des données

- ✓ `mnist.load_data()`: Charge le dataset MNIST, qui contient des images de chiffres manuscrits et leurs labels correspondants.
- ✓ `reshape`: Redimensionne les images pour qu'elles aient une profondeur de 1, ce qui est nécessaire pour un CNN traitant des images en niveaux de gris.
- ✓ Normalisation: Les valeurs des pixels sont normalisées de 0-255 à 0-1 pour faciliter l'apprentissage du réseau.
- ✓ `to_categorical`: Convertit les labels en catégories one-hot, une représentation binaire nécessaire pour la classification multi-classes.

Construction du modèle

L'architecture CNN que vous avez utilisée pour le jeu de données MNIST est une version simplifiée du modèle LeNet-5. Voici une description de chaque couche de votre modèle :

1. **C1 - Couche de convolution** : Cette couche utilise 6 filtres de taille (5 \times 5) avec un stride de 1 et un padding 'same', ce qui signifie que la sortie aura la même dimension que l'entrée. L'activation 'relu' est utilisée pour introduire la non-linéarité.
2. **S2 - Sous-échantillonnage / Couche de pooling** : Ici, vous avez une couche de pooling moyenne (AveragePooling) de taille (2 \times 2) avec un stride de 2 et un padding 'valid', ce qui réduit la dimension spatiale de la sortie.
3. **C3 - Couche de convolution** : Une autre couche de convolution avec 16 filtres de taille (5 \times 5), un stride de 1 et un padding 'valid', ce qui réduit encore la dimension de la sortie puisque le padding 'same' n'est pas utilisé.
4. **S4 - Sous-échantillonnage / Couche de pooling** : Similaire à S2, cette couche de pooling moyenne réduit davantage la dimension spatiale de la sortie.
5. **C5 - Couche de convolution** : Cette couche a 120 filtres de taille (2 \times 2) avec un stride de 1 et un padding 'valid'. C'est une grande augmentation du nombre de filtres par rapport à la couche précédente, ce qui permet de capturer plus de caractéristiques avant le passage à des couches entièrement connectées.
6. **Flatten** : La sortie de la dernière couche de convolution est aplatie en un vecteur unidimensionnel pour être traitée par les couches entièrement connectées.
7. **F6 - Couche entièrement connectée** : Cette couche dense a 84 neurones et utilise également l'activation 'relu'.
8. **Couche de sortie** : La dernière couche dense a 10 neurones (un pour chaque classe de chiffres de 0 à 9) et utilise l'activation 'softmax' pour la classification multiclasse.

Le modèle est compilé avec la fonction de perte 'categorical_crossentropy' et l'optimiseur 'SGD'. Les métriques de performance incluent la 'précision'.

Notre classe `MetricsHistory` est un rappel personnalisé pour enregistrer la précision, la précision de validation, la perte logistique et la perte logistique de validation à la fin de chaque époque. Vous avez également inclus du code pour visualiser ces métriques à l'aide de matplotlib, à la fois sous forme d'animation et de graphiques statiques.

- ✓ `Sequential()`: Initialise un modèle séquentiel, permettant d'ajouter des couches l'une après l'autre.

- ✓ `Conv2D`: Ajoute une couche de convolution qui extrait des caractéristiques des images.
- ✓ `AveragePooling2D`: Réduit la dimensionnalité des données tout en préservant les caractéristiques importantes.
- ✓ `Flatten()`: Aplatit les données multidimensionnelles en un vecteur unidimensionnel pour les couches entièrement connectées.
- ✓ `Dense`: Ajoute une couche entièrement connectée qui apprend des combinaisons non linéaires des caractéristiques.
- ✓ `compile`: Configure le modèle pour l'entraînement en définissant la fonction de perte, l'optimiseur et les métriques.

Explication des paramètres utilisés dans les fonctions Conv2D, AveragePooling2D, Dense, compile et fit :

1. **Conv2D** :
   - `filters` (6, 16, 120) : Nombre de filtres de convolution, détermine le nombre de caractéristiques extraites.
   - `kernel_size` ((5, 5), (2, 2)) : Taille de chaque filtre de convolution, affecte la zone de l'image analysée.
   - `strides` ((1, 1)) : Le déplacement du filtre sur l'image, des strides plus grands peuvent réduire la dimensionnalité de sortie.
   - `activation` ('relu') : Fonction d'activation utilisée pour introduire la non-linéarité dans le processus d'apprentissage.
   - `input_shape` ((28,28,1)) : Forme des données d'entrée, nécessaire uniquement pour la première couche de convolution.
   - `padding` ('same', 'valid') : 'same' garde la dimensionnalité originale, 'valid' réduit la dimensionnalité sans ajouter de zéros.
2. **AveragePooling2D** :
   - `pool_size` ((2, 2)) : La taille de la fenêtre de pooling, réduit la dimensionnalité en prenant la moyenne.
   - `strides` ((2, 2)) : Le déplacement de la fenêtre de pooling sur les caractéristiques, souvent égal à `pool_size`.
3. **Dense** :
   - `units` (84, 10) : Nombre de neurones dans la couche, détermine la complexité du modèle.
   - `activation` ('relu', 'softmax') : 'relu' pour les couches cachées, 'softmax' pour la couche de sortie qui normalise les scores en probabilités.
4. **compile** :
   - `loss` ('categorical_crossentropy') : Fonction de perte pour évaluer l'erreur du modèle, adaptée à la classification multi-classes.
   - `optimizer` ('SGD') : Méthode d'optimisation, Stochastic Gradient Descent, pour mettre à jour les poids du réseau.
   - `metrics` (['accuracy']) : Métriques de performance à surveiller pendant l'entraînement.
5. **fit** :
   - `epochs` (5) : Nombre de fois que l'ensemble des données d'entraînement est passé à travers le réseau.
   - `batch_size` (128) : Nombre d'échantillons traités avant de mettre à jour les poids du modèle.

- `verbose` (1) : Affiche la progression de l'entraînement.
- `validation_data` ((test_images, test_labels)) : Données utilisées pour évaluer la performance du modèle après chaque époque.
- `callbacks` ([history]) : Liste des callbacks à appliquer pendant l'entraînement pour des actions personnalisées.

Chaque paramètre joue un rôle spécifique dans la construction et l'entraînement du modèle, influençant directement la performance et l'efficacité du réseau de neurones convolutifs.

Entraînement du modèle

- `MetricsHistory`: Une classe personnalisée pour enregistrer les métriques de performance pendant l'entraînement.
- `model.fit`: Entraîne le modèle sur les données avec des callbacks pour enregistrer les métriques.

Visualisation des métriques

- `plt.subplots`: Crée des figures pour les graphiques.
- `FuncAnimation`: Crée une animation montrant l'évolution des métriques au fil des époques.
- `plt.show()`: Affiche les graphiques.

Conclusion

Le code fournit un exemple complet de la mise en œuvre d'un CNN avec Keras, de l'entraînement du modèle et de la visualisation des performances. Chaque fonction et paramètre joue un rôle crucial dans le processus d'apprentissage automatique, de la préparation des données à l'évaluation des résultats. La visualisation des métriques permet de suivre les progrès et d'ajuster les paramètres pour améliorer les performances du modèle.