Réalisé par : Chiheb ELLEFI & Hend HAMDI

1 Le principe de l'algorithme

Ce script Python utilise la bibliothèque PyTorch pour configurer, entraîner et évaluer un réseau neuronal convolutif (CNN) pour la classification d'images. Après avoir importé les bibliothèques requises, configuré les paramètres et créé l'ensemble de données, le modèle CNN est défini comme des couches convolutionnelles, de normalisation, ReLU et de pooling, suivies d'une couche entièrement connectée pour la classification. Le modèle est ensuite configuré pour la formation à l'aide d'une fonction de perte d'entropie croisée, d'un optimiseur SGD avec élan et d'un planificateur d'apprentissage.

La boucle d'entraînement itère pendant un nombre spécifié d'époques pour ajuster le modèle aux données d'entraînement et évaluer ses performances sur les données de test. Les résultats tels que la perte et la précision sont enregistrés pour analyse. Générez un graphique montrant la perte d'époque et la précision et enregistrez-le au format PDF. Enfin, les résultats finaux, y compris la perte et la précision de la formation et des tests, sont présentés. Dans l'ensemble, ce script fournit une approche complète pour créer, entraîner et évaluer des modèles CNN dans le contexte de la classification d'images.

Nature de l'apprentissage :

Apprentissage supervisé : l'algorithme est formé sur un ensemble de données étiquetées (catégories préalablement définies). Le modèle apprend de ces exemples étiquetés pour être capable de généraliser et de classer de nouvelles images qu'il n'a jamais vues auparavant.

Type de problème :

Classification : le problème est essentiellement la classification, ce qui signifie entraîner le modèle à attribuer une catégorie ou une étiquette prédéfinie à chaque image d'entrée.Les classes utilisées dans le dataset sont: Annual Crop / Forest / River / Sea Lake / Highway / Industrial / Pasture / Permanent Crop / Residential / Herbaceous Vegetation.

2 Partitionnement du dataset

on va utiliser le script suivant pour partitionner notre dataset on deux parties train_set (80%) et test_set (20%) (Vous pouvez examiner le reste du code dans le fichier 'split.py' situé dans un 'other' répertoire.).

```
import os
import shutil
import random
from sklearn.model_selection import train_test_split
# Chemin vers votre dossier de dataset
chemin_du_dossier_dataset = './EuroSAT_data'

# Liste de toutes les classes dans le dossier de dataset
classes = os.listdir(chemin_du_dossier_dataset)

# Création des répertoires pour les ensembles de train et de test
chemin_entrainement = './data/train'
chemin_test = './data/test'
.
.
.
    for img in images_test:
        shutil.copy(os.path.join(chemin_du_dossier_dataset, nom_classe, img), os.path.join(chemin_classe
```

3 Phase de training & phase de test

Phase d'entraînement :

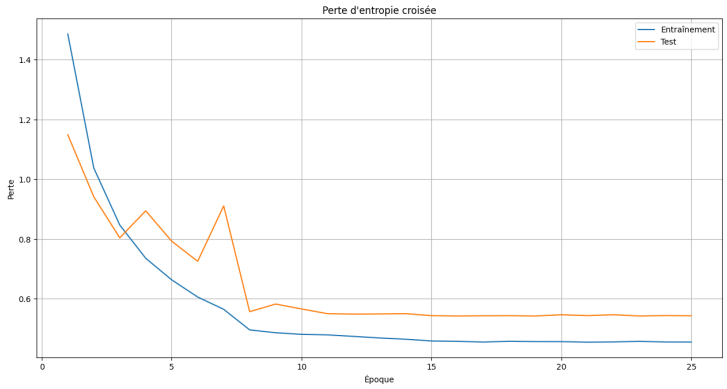
- Initialisation : Le modèle est initialisé avec des poids aléatoires.
- Forward pass : Les données d'entraînement sont propagées à travers le réseau (forward pass) pour générer des prédictions.
- Calcul de la perte : La différence entre les prédictions du modèle et les vraies étiquettes (labels) est calculée à l'aide d'une fonction de perte, ici la Cross-Entropy Loss.
- Backward pass : Les gradients sont calculés à partir de la fonction de perte par rétropropagation (backward pass), et les poids du modèle sont ajustés pour minimiser cette perte à l'aide d'un optimiseur.
- Évaluation : Pendant l'entraînement, le modèle est évalué sur les données d'entraînement pour suivre sa performance.

Phase de test :

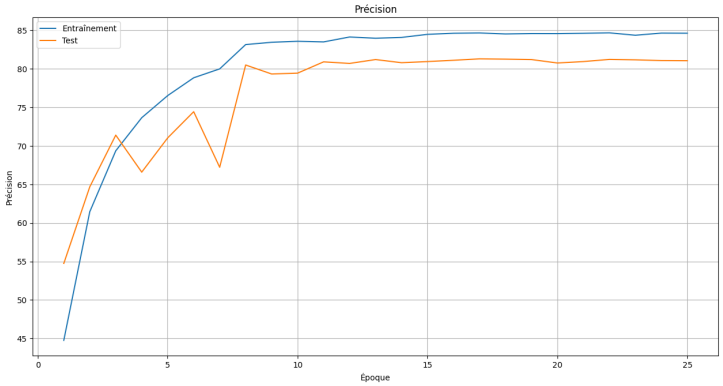
- Forward pass : Les données de test (non utilisées pour l'entraînement) sont propagées à travers le réseau.
- Calcul de la perte : Comme dans la phase d'entraînement, la perte est calculée pour évaluer la performance du modèle sur les données de test.
- Évaluation de la précision : Les prédictions du modèle sont comparées aux vraies étiquettes pour mesurer la précision du modèle sur les données de test.

4 Métriques d'évaluation

- Perte (Loss) : La perte (Cross-Entropy Loss dans ce cas) mesure la différence entre les valeurs prédites par le modèle et les valeurs réelles.



- Précision (Accuracy): Il s'agit du pourcentage d'images correctement classées par le modèle.



Tendance de la perte : La diminution progressive à la fois de la perte d'entraînement et de la perte de test au fil des époques suggère que le modèle a appris efficacement. En partant de 1,5 (entraînement) et 1,15 (test) pour descendre en dessous de 0,6 (entraînement) et 0,4 (test), cela indique une convergence améliorée.
Tendance de la précision : L'augmentation de la précision pour les ensembles d'entraînement et de test indique que le modèle apprend des représentations qui généralisent bien. Passant de 45 % (entraînement) et 55 % (test) à 85 % (entraînement) et 81 % (test), il est évident que le modèle capture davantage de subtilités dans les données à mesure que l'entraînement progresse.
Évaluation finale :

```
Perte d'entraînement finale: 0.45464267489120097
Perte de test finale: 0.5425835467436735
Précision d'entraînement finale: 84.58333587646484
Précision de test finale: 81.01851654052734
```

- Perte : Les valeurs finales de perte sont relativement faibles, indiquant que le modèle a appris à minimiser efficacement les erreurs.
- Précision : Atteindre une précision de 85 % pour l'entraînement et de 81 % pour les tests est encourageant. Cependant, il pourrait y avoir un léger surajustement étant donné que la précision d'entraînement est nettement plus élevée que celle des tests.

Choix des paramètres de configuration et leurs impacts : Les paramètres de configuration impactent significativement les performances du modèle :

- Taille du lot (Batch Size) : Une taille de lot plus grande peut accélérer l'entraînement, mais nécessite plus de mémoire. Des tailles de lot plus petites peuvent aider à la convergence du modèle, mais l'entraînement peut être plus lent.
- Nombre d'époques (Number of Epochs) : Contrôle le nombre de fois où le modèle parcourt l'ensemble de données d'entraînement complet. Trop peu d'époques peuvent entraîner un modèle non entraîné, tandis que trop d'époques peuvent conduire à un surajustement.
- Taux d'apprentissage (Learning Rate): Influence la taille des pas faits lors de la mise à jour des poids du réseau. Un taux d'apprentissage plus élevé peut accélérer l'apprentissage, mais peut aussi causer une instabilité. Un taux trop bas peut ralentir l'apprentissage.
- Momentum et Weight Decay: Le momentum peut accélérer la convergence en aidant à traverser les minima locaux, tandis que le weight decay régularise les poids pour éviter le surajustement.