Projet CNN

BEMELINGUE Wilfried
DUCROCQ Tanguy

21 mai 2025

Table des matières

1 Introduction

L'analyse d'images astronomiques est devenue un sujet face à la quantité de données produites par les télescopes modernes. Une étape importante est la classification des galaxies selon leur morphologie pour mieux comprendre la formation et l'évolution de l'univers. Ce projet s'inscrit dans ce contexte, en explorant l'utilisation d'un réseau de neurones convolutif (CNN) pour reconnaître automatiquement différents types de galaxies à partir d'images.

L'objectif est de concevoir un CNN capable de classifier efficacement ces images dans la bonne catégorie. Cette approche s'appuie sur les performances des CNN dans les tâches de vision par ordinateur, en particulier, pour extraire automatiquement des caractéristiques visuelles complexes.

2 État de l'art

La classification d'images de galaxies a d'abord été abordée à l'aide de méthodes classiques de traitement d'image et d'algorithmes de machine learning comme les SVM ou les random forest. Cependant, ces approches nécessitaient une extraction manuelle des caractéristiques visuelles. Aujourd'hui, les réseaux de neurones convolutifs (CNN) sont devenus la norme pour ce type de tâche.

3 Construction du modèle

3.1 Formalisation du problème

Le but de ce projet est de résoudre un problème de classification supervisée à partir d'images. On dispose d'un jeu de données constitué d'images RGB de galaxies, chacune associée à un label entier entre 0 et 9 représentant une des dix classes possibles. On note $x \in \mathbf{R}^{3\times256\times256}$ une image d'entrée, et $y \in \{0, ..., 9\}$ le label associé.
Le modèle à concevoir est une fonction $f_\theta : \mathbf{R}^{3\times256\times256} \to \mathbf{R}^{10}$, paramétrée par θ, qui prend en entrée une image et retourne un vecteur de scores (logits) pour chaque classe. Le label prédit est obtenu par $\hat{y} = argmax(f_\theta(x))$
Avant d'entraîner notre modèle, les images subissent plusieurs transformations :

- Redimensionnement en 256x256 pour garantir une taille uniforme des entrées
- Conversion en tenseur PyTorch, pour pouvoir les traiter par les couches du modèle
- Normalisation des canaux RGB selon la moyenne et l'écart type de l'ensemble, afin d'accélérer et stabiliser l'entraînement

Les données sont ensuite divisées en trois sous-ensembles : entraînement, validation et test, à l'aide d'un **random_state=42** pour garantir la reproductibilité. On utilise un **batch_size=32**, ce qui donne des tenseurs de forme :

Images : torch.Size([32, 3, 256, 256]) (float32)
Labels : torch.Size([32]) (int64, conforme aux exigences de CrossEntropyLoss)

3.2 Architecture

Le modèle conçu suit une architecture CNN classique en plusieurs blocs. Chaque bloc est composé de trois opérations successives :

1. Une convolution 2D qui extrait des motifs visuels dans les images
2. Une fonction d'activation ReLU, qui introduit la non-linéarité et empêche l'effondrement des gradients.
3. Une opération de MaxPooling, qui réduit la dimension spatiale tout en conservant les caractéristiques les plus importantes.

Notre modèle implémente une architecture convolutive en 4 blocs successifs, suivis d'un classifieur entièrement connecté. Cette structure permet une extraction progressive des caractéristiques visuelles, depuis des motifs simples jusqu'à des représentations abstraites, adaptées à la classification de galaxies.

Partie convolutive (feature extractor) Cette partie est composée de 4 blocs. Chaque bloc contient :

- Une couche de convolution : elle applique plusieurs filtres à l'entrée, pour détecter des motifs locaux. Les tailles de noyaux utilisées sont 5×5 pour la première couche (avec stride=2, ce qui réduit immédiatement la résolution de moitié), puis 3×3 pour les suivantes, avec un padding de 1 pour conserver les dimensions spatiales.
- Une fonction d'activation ReLU : elle introduit de la non-linéarité, en annulant les valeurs négatives.
- Une opération de MaxPooling avec une fenêtre 2×2 : elle réduit de moitié les dimensions spatiales (hauteur et largeur) à chaque bloc, ce qui diminue la complexité tout en conservant l'essentiel de l'information.

Voici l'évolution des dimensions spatiales d'une image d'entrée (initialement 3×256×256) à travers les couches :

Couche	**Type**	**Dimensions en sortie**
Conv2d(3, 32, kernel=5, stride=2, padding=2) MaxPool2d(2)	+ ReLU	32 × 128 × 128 32 × 64 × 64
Conv2d(32, 64, kernel=3, padding=1) MaxPool2d(2)	+ ReLU	64 × 64 × 64 64 × 32 × 32
Conv2d(64, 128, kernel=3, padding=1) MaxPool2d(2)	+ ReLU	128 × 32 × 32 128 × 16 × 16
Conv2d(128, 128, kernel=3, padding=1) MaxPool2d(2)	+ ReLU	128 × 16 × 16 128 × 8 × 8

TABLE 1 – Architecture des couches convolutionnelles avec les dimensions de sortie

Partie classifieur (fully connected) Après l'extraction de caractéristiques, les tenseurs de dimension 128×8×8 sont aplaties en un vecteur de dimension 8192, puis passent dans deux couches linéaires :

— Une couche entièrement connectée : Linear(8192, 256) suivie d'une ReLU.
— Un dropout avec un taux de 0.5 : pour régulariser et réduire l'overfitting.
— Une dernière couche linéaire : Linear(256, 10) qui produit les logits pour les 10 classes.

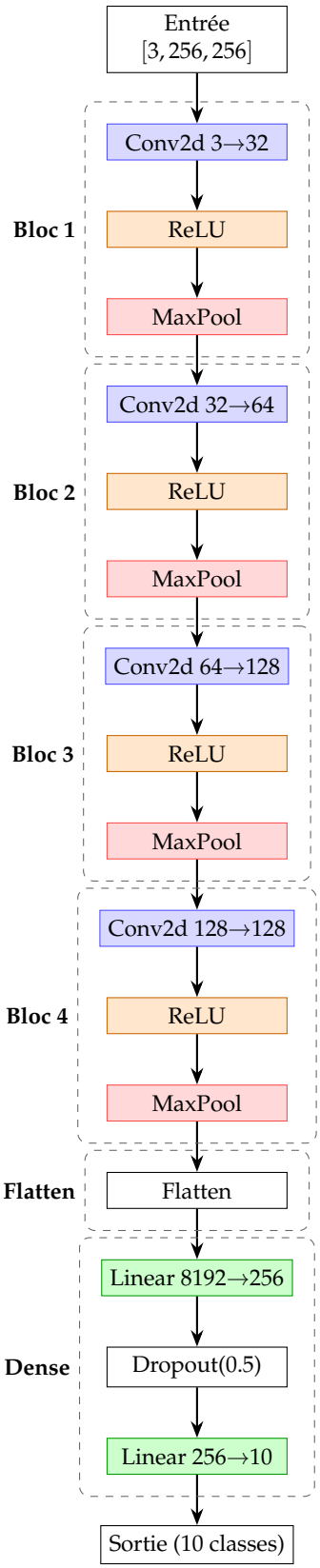

FIGURE 1 – Architecture CNN avec blocs espacés et encadrés

3.3 Fonction de coût

Nous utilisons la fonction de perte d'entropie croisée (CrossEntropyLoss), adaptée aux problèmes de classification multi-classes. Cette fonction combine une couche de softmax et la perte logarithmique. Mathématiquement, pour une sortie du modèle $z = f_\theta(x)$ et un label réel y, la fonction de coût est donnée par :

$$L(z, y) = -log(\frac{e^{z_y}}{\sum_{k=1}^{10} e^{z_k}})$$

Elle permet d'évaluer la distance entre la distribution de probabilité prédite et la distribution réelle.

4 Expérimentation et résultats

4.1 Présentation des données

Le jeu de données utilisé est **Galaxy10 DECals**, accessible sur la plateforme Hugging Face [1]. Il s'agit d'un ensemble d'images astronomiques RGB représentant différentes photos de galaxie. Chaque image est associée à un label entier compris entre 0 et 9, représentant l'une des dix classes définies dans le cadre du challenge Galaxy Zoo.

Afin de garantir une taille uniforme des entrées, les images sont redimensionnées en **256×256 pixels**. Un ensemble de transformations a été appliqué en phase d'entraînement pour augmenter la robustesse du modèle :

- Redimensionnement fixe : `Resize(256, 256)`
- Flip horizontal aléatoire : `RandomHorizontalFlip()`
- Rotation aléatoire : `RandomRotation(15)`
- Jitter couleur : `ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2)`
- Normalisation : moyenne et écart-type à 0.5 pour chaque canal RGB

Le dataset est divisé de la manière suivante :

- **Entraînement** : 80% des images
- **Validation** : 10%
- **Test** : 10%

Cette répartition a été effectuée avec un `random_state=42` pour assurer la reproductibilité des expériences. Le chargement des données s'effectue par batchs de 32 images, soit des tenseurs de forme `[32, 3, 256, 256]` pour les images et `[32]` pour les labels.

Cette préparation permet d'assurer une bonne qualité d'apprentissage et de tester efficacement la généralisation du modèle sur des données jamais vues.

4.2 Optimisation

Pour entraîner notre réseau de neurones convolutif, nous avons comparé plusieurs méthodes d'optimisation couramment utilisées en deep learning :

- **SGD (Stochastic Gradient Descent)** avec momentum
- **Adagrad** : un optimiseur adaptatif sensible à la fréquence des paramètres

1. https://huggingface.co/datasets/matthieulel/galaxy10_decals

— **Adam** : combinaison d'Adagrad et RMSProp, très efficace en pratique

Chaque optimiseur a été utilisé avec ses hyperparamètres standards :

— SGD : `lr=0.01`, `momentum=0.9`

— Adagrad : `lr=0.01`

— Adam : `lr=0.001`

Le modèle a été entraîné pendant **20 epoch** dans chaque cas, et les performances ont été comparées sur les ensembles de validation et de test.

Optimiseur	Test Loss	Test Accuracy
SGD	0.7232	0.7527
Adagrad	0.8130	0.7276
Adam	**0.6895**	**0.7708**

TABLE 2 – Performance comparative des optimiseurs sur le jeu de test

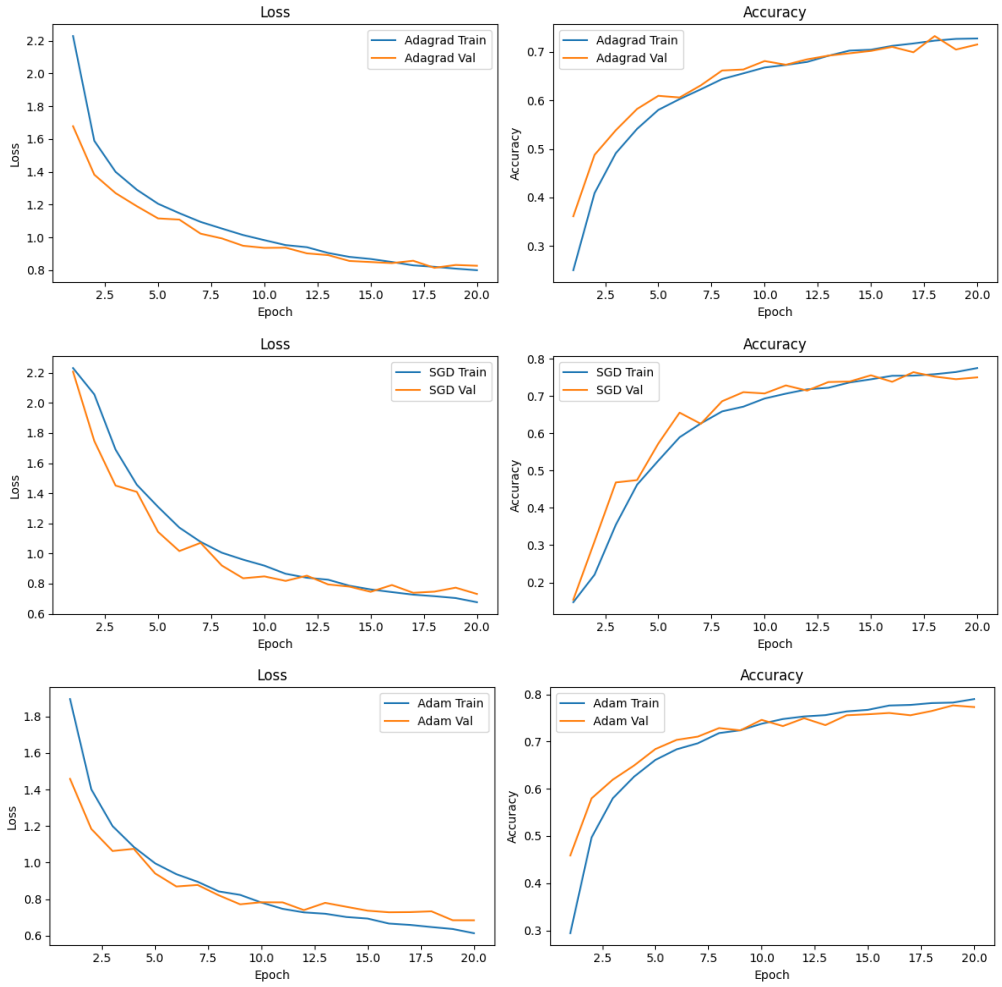

FIGURE 2 – Comparaison des courbes d'apprentissage pour SGD, Adagrad et Adam

Résultat principal : l'optimiseur **Adam** s'est révélé être le plus performant en termes de rapidité de convergence et de précision finale sur l'ensemble de test. Il a donc été retenu pour les expérimentations ultérieures.

4.3 Réglage des hyperparamètres

Après avoir sélectionné l'optimiseur **Adam**, nous avons entrepris un réglage plus fin des hyperparamètres via une recherche par grille (*grid search*) sur deux paramètres essentiels :

— Le taux d'apprentissage (`lr`) : {1e-3, 1e-4, 1e-5}

— Le `weight_decay` (régularisation L2) : {0, 1e-4, 1e-5}

lr	weight_decay	Test Loss	Test Accuracy
1e-3	0	**0.7188**	**0.7677**
1e-3	1e-4	0.7377	0.7464
1e-3	1e-5	0.7272	0.7489
1e-4	0	0.8985	0.6944
1e-4	1e-4	0.9399	0.6788
1e-4	1e-5	0.9288	0.6756
1e-5	0	1.5072	0.4721
1e-5	1e-4	1.5495	0.4421
1e-5	1e-5	1.5550	0.4515

TABLE 3 – Résultats de la recherche par grille sur les hyperparamètres (Adam)

Conclusion : la configuration optimale retenue est `lr=0.001` et `weight_decay=0`, atteignant une précision de **76.77%** sur le jeu de test. Elle a servi pour l'entraînement final du modèle.

4.4 Résultats

Le modèle final a été entraîné avec l'optimiseur **Adam**, un taux d'apprentissage `lr=0.001` et un `weight_decay=0`. L'entraînement a été effectué sur **20 epoch**.

Les performances sur le jeu de test sont les suivantes :

— **Test Loss** : 0.7100

— **Test Accuracy** : 76.71%

On observe une bonne convergence du modèle dès les premières époques, avec une stabilisation de la précision autour de la quinzième. L'écart modéré entre les courbes d'entraînement et de validation traduit une capacité de généralisation correcte.

Analyse critique : Bien que les performances soient honorables, certaines confusions persistent entre classes proches.

5 Conclusions / Perspectives

Dans ce projet, nous avons conçu et entraîné un réseau de neurones convolutif (CNN) pour la classification automatique d'images de galaxies issues du dataset *Galaxy10 DECals*. Après avoir défini une architecture adaptée et appliqué une série de transformations pour améliorer la robustesse du modèle, nous avons procédé à une évaluation rigoureuse des performances à travers différents optimiseurs et réglages d'hyperparamètres.

Notre contribution principale repose sur l'analyse comparative de trois méthodes d'optimisation (SGD, Adagrad, Adam), suivie d'un réglage fin des hyperparamètres par recherche en grille. Le modèle final, entraîné avec `Adam`, `lr=0.001` et `weight_decay=0`, a atteint une précision de **76.71%** sur le jeu de test. Ce résultat montre que l'approche retenue est capable de capturer des motifs visuels pertinents pour distinguer les morphologies galactiques.

Cependant, plusieurs perspectives peuvent être envisagées pour améliorer encore les performances :

- **Utiliser des modèles pré-entraînés** (ex. ResNet, EfficientNet) via du *transfer learning*
- **Explorer des architectures plus profondes** ou utilisant des mécanismes d'attention
- **Automatiser l'optimisation** avec des outils comme *Optuna* ou *Bayesian Optimization*

En somme, ce travail constitue une base solide pour l'automatisation de la classification galactique, et peut être enrichi par des approches plus sophistiquées dans le cadre de recherches futures.