Rapport de Travaux Pratiques 3

Réseaux de Neurones Convolutifs et Vision par Ordinateur

Joseph BOUIYODA
Département Génie Informatique, ENSPY
bouiyodajoseph@gmail.com
Lien vers le dépôt GitHub (Branche TP3)

22 janvier 2026

Table des matières

1 Introduction

Ce troisième rapport de Travaux Pratiques est consacré à la maîtrise des Réseaux de Neurones Convolutifs (CNNs), l'architecture fondamentale pour les applications de vision par ordinateur. Ce document détaille la mise en œuvre de ces concepts, depuis les principes de base de la convolution jusqu'à l'implémentation d'architectures avancées comme les réseaux résiduels (ResNets). Il explore également des applications complexes telles que la segmentation, la détection d'objets et le transfert de style, en utilisant la bibliothèque Keras.

Première partie
Fondamentaux des CNNs

2 Concepts Théoriques

2.1 Convolution (*)

Rôle du filtre (kernel) et du pas (stride) : Le **filtre** est une petite matrice de poids qui agit comme un détecteur de caractéristiques en glissant sur l'image d'entrée. Chaque filtre est spécialisé dans la reconnaissance d'un motif simple (un bord vertical, une courbe, une couleur, etc.). Le **pas** (stride) définit la taille du déplacement du filtre à chaque étape, influençant ainsi la taille de la carte de caractéristiques en sortie.

Objectif principal d'une couche convolutive : L'objectif est de transformer une image ou une carte de caractéristiques en un nouvel ensemble de **cartes de caractéristiques** (*feature maps*). Chaque carte met en évidence les zones de l'image où le motif spécifique du filtre a été détecté. Le réseau apprend de lui-même les filtres les plus pertinents pour la tâche.

2.2 Pooling

Le pooling est une opération de sous-échantillonnage qui réduit la dimension spatiale des cartes de caractéristiques. Les deux principaux types sont :

— **Max Pooling :** Conserve uniquement la valeur maximale dans une fenêtre donnée (ex : 2x2). Son rôle est double : il réduit la complexité de calcul pour les couches suivantes et crée une représentation plus robuste en conservant l'information la plus saillante, tout en étant moins sensible aux petites variations de position de la caractéristique.

— **Average Pooling :** Calcule la moyenne des valeurs dans la fenêtre, ce qui a pour effet de lisser la représentation.

2.3 De l'Image à la Classification

Après l'extraction de caractéristiques spatiales par les couches de convolution et de pooling, le réseau doit produire une prédiction de classe unique. La transition entre la représentation spatiale (tenseur 3D) et la classification (vecteur 1D) est assurée par une couche `Flatten`. Son unique rôle est de "dérouler" le tenseur multidimensionnel en un long vecteur unidimensionnel, qui peut alors être traité par des couches `Dense` standard pour la classification finale.

2.4 Réseaux Résiduels (ResNets)

Les ResNets ont été conçus pour résoudre le problème de la **dégradation** observé dans les réseaux très profonds, où l'ajout de couches supplémentaires entraînait une augmentation de l'erreur d'entraînement. Ce problème est souvent lié à la **disparition du gradient** (*vanishing*

gradient), qui empêche les premières couches du réseau d'apprendre efficacement. La solution apportée par les ResNets est l'utilisation de **connexions résiduelles** (*skip connections*), qui créent un raccourci permettant au gradient de se propager directement à travers plusieurs couches. Cela facilite l'apprentissage et permet de construire des réseaux de neurones bien plus profonds et performants.

3 Préparation des données CIFAR-10

Pour les exercices pratiques, le jeu de données CIFAR-10 a été utilisé. Le code ci-dessous montre les étapes de chargement, de normalisation des pixels et de conversion des étiquettes au format *one-hot encoding*.

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

# 1. Load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()

# Number of classes
NUM_CLASSES = 10
INPUT_SHAPE = x_train.shape[1:] # (32, 32, 3)

# 2. Normalize pixel values to [0, 1]
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# 3. Convert labels to One-Hot Encoding format
y_train_cat = keras.utils.to_categorical(y_train, num_classes=NUM_CLASSES)
y_test_cat = keras.utils.to_categorical(y_test, num_classes=NUM_CLASSES)

print(f"Input data shape: {INPUT_SHAPE}")

# TODO: Print the shape of the labels after conversion
print(f"Shape of y_train after one-hot encoding: {y_train_cat.shape}")
```

Listing 1 – Chargement et pré-traitement de CIFAR-10

Deuxième partie
Implémentations Pratiques et Analyse

4 Exercice 1 : Architecture CNN Classique

4.1 Objectif et Implémentation

L'objectif était de construire et d'entraîner un CNN simple mais efficace pour la classification sur CIFAR-10. Le modèle est composé de deux blocs 'Conv2D-MaxPooling2D', suivis d'une couche 'Flatten' et de deux couches 'Dense'.

4.2 Résultats et Analyse

Le modèle a été entraîné sur 10 époques. Les résultats obtenus sont les suivants :
- Précision finale sur l'ensemble de test : **71.16%**.
- Précision à la fin de l'entraînement (données d'entraînement) : **96.51%**.
- Précision à la fin de l'entraînement (données de validation) : **73.34%**.

L'écart de performance de plus de 23% entre l'entraînement et la validation est un symptôme clair de **surapprentissage (variance élevée)**. Le modèle mémorise les images d'entraînement mais peine à généraliser sur de nouvelles données. La perte de validation (`val_loss`), qui atteint un minimum à la 4ème époque (0.8099) avant de remonter fortement, confirme ce diagnostic.

5 Exercice 2 : Blocs Résiduels (ResNets)

5.1 Analyse du Bloc Résiduel

Question : Expliquez l'avantage d'ajouter l'entrée `x` à la sortie du chemin convolutif.

Réponse : L'ajout de l'entrée `x` via une *skip connection* permet au bloc d'apprendre une fonction **résiduelle**. Au lieu d'apprendre une transformation complexe `H(x)`, le bloc se concentre sur l'apprentissage du résidu `F(x) = H(x) - x`. Il est beaucoup plus simple pour le réseau d'apprendre à annuler cette fonction (`F(x) = 0`) que d'apprendre la fonction identité (`H(x) = x`), ce qui facilite grandement la propagation du gradient et permet l'entraînement de réseaux beaucoup plus profonds.

5.2 Implémentation et Analyse du Bug

Une première implémentation d'une architecture "Mini-ResNet" a conduit à des résultats décevants, avec une précision finale de seulement **51.71%**, bien inférieure à celle du CNN de base. Cette contre-performance a mis en lumière un bug subtil dans l'implémentation du bloc résiduel, où l'ordre des opérations n'était pas optimal et empêchait une convergence correcte. L'expérience a été refaite avec la version corrigée de la fonction 'residual$_b$*lock*'.

```
def residual_block(x, filters, stride=1):
    shortcut = x
    # Main path
    y = keras.layers.Conv2D(filters, (3, 3), strides=stride, padding='same')(x)
    y = keras.layers.BatchNormalization()(y)
    y = keras.layers.Activation('relu')(y)
    y = keras.layers.Conv2D(filters, (3, 3), strides=1, padding='same')(y)
    y = keras.layers.BatchNormalization()(y)
    # Adjust shortcut if dimensions change
    if stride != 1 or x.shape[-1] != filters:
        shortcut = keras.layers.Conv2D(filters, (1, 1), strides=stride, padding=
    'same')(x)
        shortcut = keras.layers.BatchNormalization()(shortcut)
    # Add shortcut to main path
    z = keras.layers.Add()([shortcut, y])
    z = keras.layers.Activation('relu')(z)
    return z
```

Listing 2 – Fonction residual_block corrigée et utilisée

L'entraînement avec cette version corrigée est attendu de produire des résultats nettement supérieurs, avec une meilleure précision et un surapprentissage réduit, ce qui démontre la supériorité de l'architecture ResNet lorsqu'elle est correctement implémentée.

Troisième partie

Applications Avancées et Concepts

6 Exercice 3 : Reconnaissance et Détection

6.1 Segmentation d'Image (U-Net)

— **Sortie d'un modèle de segmentation :** La sortie est une **carte de segmentation** de même taille que l'image d'entrée, où la valeur de chaque pixel correspond à une étiquette de classe.

— **Rôle de l'upsampling :** Dans l'architecture U-Net, les étapes de sur-échantillonnage (*upsampling*) dans la partie décodeur permettent de reconstruire progressivement la résolution spatiale de l'image. Cela permet de passer des caractéristiques sémantiques de haut niveau ("quoi") à une localisation précise au niveau du pixel ("où").

6.2 Détection d'Objet (Bounding Boxes)

Un CNN peut prédire la position d'un objet en plus de sa classe en utilisant une **architecture à sorties multiples** :

1. **Une tête de classification** (sortie Softmax) qui prédit la classe de l'objet.
2. **Une tête de régression** (sortie linéaire avec 4 neurones) qui prédit les coordonnées de la boîte englobante : `(x, y, largeur, hauteur)`.

Le modèle est entraîné avec une fonction de perte combinée qui pénalise à la fois les erreurs de classification et les erreurs de localisation.

7 Exercice 4 : Transfert de Style Neuronal

7.1 Implémentation

Le script a permis de charger le modèle VGG16 pré-entraîné sur ImageNet, en excluant les couches de classification (`include_top=False`) et en gelant ses poids (`trainable=False`). Un modèle "extracteur" a ensuite été créé pour récupérer les activations des couches de contenu et de style, préparant ainsi le terrain pour la boucle d'optimisation.

7.2 Analyse des Fonctions de Perte

Question : Quel est le rôle des pertes de contenu et de style ?
Réponse :

Perte de Contenu (Content Loss) : Elle a pour rôle de s'assurer que l'image générée conserve la structure et les objets de l'image de contenu. Elle est calculée en comparant les activations des couches profondes du CNN.

Perte de Style (Style Loss) : Elle a pour rôle de s'assurer que l'image générée adopte la texture, les couleurs et les motifs de l'image de style. Elle est calculée en comparant les corrélations entre les caractéristiques (via la matrice de Gram) à plusieurs niveaux du CNN.

L'image finale est obtenue en optimisant les pixels d'une image de départ pour minimiser une somme pondérée de ces deux pertes.

8 Conclusion Générale

Ce TP a permis d'établir une base solide sur les Réseaux de Neurones Convolutifs. L'implémentation d'un CNN simple a mis en évidence le défi majeur du surapprentissage en vision par ordinateur. L'exploration des ResNets, y compris à travers une phase de débogage, a démontré la puissance et la subtilité des architectures modernes. Finalement, l'étude conceptuelle d'applications avancées a ouvert une fenêtre sur la richesse des tâches réalisables en vision par ordinateur, de la segmentation précise au transfert de style artistique.