# Rapport de Travaux Pratiques 3

## Réseaux de Neurones Convolutifs et Vision par Ordinateur

Joseph BOUIYODA
Département Génie Informatique, ENSPY
bouiyodajoseph@gmail.com

21 janvier 2026

# Table des matières

# 1 Introduction

Ce troisième rapport de Travaux Pratiques marque une transition fondamentale des réseaux de neurones denses vers les Réseaux de Neurones Convolutifs (CNNs), l'architecture de choix pour les tâches de vision par ordinateur. L'objectif de ce TP est de comprendre, implémenter et analyser les briques de base des CNNs, d'explorer des architectures avancées comme les ResNets, et d'aborder conceptuellement des applications complexes telles que la segmentation, la détection d'objets et le transfert de style.

# Première partie
# Fondamentaux des CNNs

## 2 Concepts Théoriques

### 2.1 Convolution

**Rôle du filtre (kernel) :** Un filtre est une petite matrice de poids (ex : 3x3) qui glisse sur l'image pour détecter une caractéristique spécifique (un bord, une courbe, une couleur).

**Rôle du pas (stride) :** Le pas définit le nombre de pixels de décalage du filtre à chaque étape. Un pas plus grand réduit la dimension de la sortie.

**Objectif principal :** L'objectif d'une couche convolutive est de transformer une image en un ensemble de **cartes de caractéristiques** (*feature maps*), où chaque carte met en évidence la présence d'un motif particulier dans l'image.

### 2.2 Pooling

Le pooling est une opération de sous-échantillonnage qui réduit la dimension spatiale des cartes de caractéristiques.

- **Max Pooling :** Conserve uniquement la valeur maximale dans une fenêtre (ex : 2x2). Il préserve la caractéristique la plus forte et offre une invariance aux petites translations.
- **Average Pooling :** Calcule la moyenne des valeurs dans la fenêtre, ce qui lisse la représentation.

Son rôle est de réduire la charge de calcul et de rendre la détection de caractéristiques plus robuste.

### 2.3 De l'Image à la Classification

Après l'extraction de caractéristiques par les couches convolutives, la transition vers une classification finale s'effectue via une couche `Flatten`. Son rôle est de prendre la carte de caractéristiques 3D finale et de l'aplatir en un long vecteur 1D. Ce vecteur peut alors servir d'entrée à des couches `Dense` classiques pour effectuer la classification.

### 2.4 Réseaux Résiduels (ResNets)

Les ResNets résolvent le problème de la **disparition du gradient** (*vanishing gradient*) dans les réseaux très profonds. Ce problème empêche les premières couches d'apprendre efficacement. La solution est l'ajout de **connexions résiduelles** (*skip connections*), qui créent un raccourci pour le gradient, lui permettant de se propager plus facilement à travers le réseau. Cela permet d'entraîner des réseaux de centaines de couches sans dégradation des performances.

Deuxième partie

# Implémentations Pratiques

## 3 Exercice 1 : Architecture CNN Classique

### 3.1 Objectif

Construire, entraîner et évaluer un premier CNN pour la classification sur le jeu de données CIFAR-10, plus complexe que MNIST.

### 3.2 Résultats de l'entraînement

Le modèle a été entraîné sur 10 époques. La précision finale sur l'ensemble de test est de **71.16%**. L'analyse de l'entraînement révèle un problème majeur : à la 10ème époque, la précision sur les données d'entraînement atteint **96.51%** alors que celle sur les données de validation n'est que de **73.34%**. Cet écart de plus de 23% est un signe clair de **surapprentissage (variance élevée)**. Le modèle mémorise les données d'entraînement mais ne parvient pas à généraliser correctement.

## 4 Exercice 2 : Blocs Résiduels (ResNets)

### 4.1 Objectif

Implémenter une architecture plus avancée basée sur les blocs résiduels pour tenter d'améliorer la performance et de réduire le surapprentissage observé.

### 4.2 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` (la *skip connection*) à la sortie des convolutions `y` permet au bloc d'apprendre une fonction **résiduelle**. Au lieu d'apprendre une transformation complexe `H(x)`, le bloc apprend `F(x) = H(x) - x`. Il est beaucoup plus facile pour le réseau d'apprendre à annuler cette fonction (`F(x) = 0`) que d'apprendre directement l'identité (`H(x) = x`). Cela facilite grandement la propagation du gradient dans les réseaux profonds et combat le problème de dégradation des performances.

### 4.3 Implémentation et Résultats

Une première implémentation du bloc résiduel a conduit à des résultats décevants (précision de 51.71%), inférieurs à ceux du CNN de base.

```
--- COMPARAISON FINALE (AVEC BUG INITIAL) ---
Précision du CNN de base : 0.7116
Précision du Mini-ResNet : 0.5171
```

### 4.4 Analyse du Bug et Correction

Ce résultat contre-intuitif a révélé un bug subtil dans l'implémentation du bloc résiduel, où l'ordre des opérations n'était pas optimal. Une version corrigée, respectant mieux l'architecture ResNet originale, a été implémentée.

```
1 def residual_block(x, filters, stride=1):
2     shortcut = x
3
```

```
    # 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)

    z = keras.layers.Add()([shortcut, y])
    z = keras.layers.Activation('relu')(z)
    return z
```

Listing 1 – Fonction residual_block corrigée

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

# Troisième partie
# Applications Avancées (Conceptuel)

## 5 Exercice 3 : Reconnaissance et Détection

### 5.1 Segmentation d'Image (U-Net)

— **Sortie d'un modèle de segmentation :** La sortie est une **carte de segmentation** (ou masque) de la même taille que l'image d'entrée, où chaque pixel est assigné à une classe.
— **Rôle de l'upsampling :** Dans une architecture U-Net, la partie ascendante (décodeur) utilise l'upsampling (ou convolutions transposées) pour augmenter progressivement la résolution spatiale. Cela permet de passer des caractéristiques sémantiques ("quoi") à une localisation précise ("où"), afin de reconstruire la carte de segmentation finale.

### 5.2 Détection d'Objet (Bounding Boxes)

Un CNN peut prédire la position d'un objet en utilisant une **tête à sorties multiples** après les couches d'extraction de caractéristiques :

1. **Une tête de classification** (sortie Softmax) pour prédire la classe de l'objet.
2. **Une tête de régression** (sortie linéaire avec 4 neurones) pour prédire les 4 coordonnées de la boîte englobante : `(x, y, w, h)`.

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

## 6 Exercice 4 : Transfert de Style Neuronal

### 6.1 Objectif

Préparer une tâche de transfert de style en chargeant un modèle VGG16 pré-entraîné et en définissant les couches d'extraction.

### 6.2 Implémentation

Le script charge le modèle VGG16 sans ses couches de classification, le gèle (`trainable=False`), et crée un modèle "extracteur" pour récupérer les activations des couches de contenu et de style.

```
# Charger les images (non montr )
content_image = load_and_process_image('content.jpg')
style_image = load_and_process_image('style.jpg')

# Charger le mod le VGG16 pr -entra n
vgg = keras.applications.VGG16(include_top=False, weights='imagenet')
vgg.trainable = False

# D finir les couches d'int r t
content_layers = ['block5_conv2']
style_layers = ['block1_conv1', 'block2_conv1', 'block3_conv1',
                'block4_conv1', 'block5_conv1']
```

Listing 2 – Préparation pour le transfert de style

### 6.3 Analyse des Fonctions de Perte

**Perte de Contenu (Content Loss) :** Assure 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 (sémantiques) du CNN.

**Perte de Style (Style Loss) :** Assure que l'image générée adopte la texture 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'optimisation vise à minimiser une somme pondérée de ces deux pertes.

## 7 Conclusion Générale

Ce TP a permis d'acquérir une compréhension pratique et théorique des Réseaux de Neurones Convolutifs. L'implémentation d'un CNN simple sur CIFAR-10 a mis en évidence le défi du surapprentissage. L'introduction aux ResNets, malgré un bug initial instructif, a démontré la puissance des architectures modernes pour l'entraînement de réseaux profonds. Enfin, l'exploration conceptuelle de la segmentation, de la détection et du transfert de style ouvre la voie à des applications de vision par ordinateur plus complexes et spécialisées.