Université Sultan Moulay Slimane
FP – Khouribga
Département : Mathématiques et Informatique
Filière : Systèmes d'Information et Intelligence Artificielle
Module : Deep Learning
Atelier : Deep Learning
Année Universitaire : 2024/2025
Pr. Ibtissam Bakkouri

Atelier - 2

Objectifs :

1. Comprendre les composants clés des réseaux de neurones convolutionnels.
2. Illustrer l'effet du padding, du stride et de la taille des filtres sur la sortie d'une couche convolutionnelle.
3. Construire et explorer différentes couches CNN avec Python.

Prérequis :

- Python installé (version 3.x).
- Bibliothèques nécessaires : **numpy**, **tensorflow** ou **pytorch**, **matplotlib**.

1- Couche de convolution :

Exemple : Calculer la taille de la sortie

- **Formule pour la taille de sortie :**

$$\text{Sortie} = \frac{\text{Entrée} + 2 \times \text{Padding} - \text{Taille du filtre}}{\text{Stride}} + 1$$

```
import numpy as np

def convolution_output_size(input_size, filter_size, stride, padding):
    return (input_size + 2 * padding - filter_size) // stride + 1

# Exemple
input_size = 32  # Taille de l'entrée (32x32 image)
filter_size = 3  # Taille du filtre (3x3)
stride = 1   # Pas
padding = 1          # Padding
output_size = convolution_output_size(input_size, filter_size, stride, padding)
print(f"Taille de sortie après convolution : {output_size}x{output_size}")
```

Exemple : Convolution avec et sans padding

```
import tensorflow as tf

# Entrée fictive (image 5x5 avec une seule couche)
input_data = tf.constant([
    [1, 2, 3, 0, 1],
    [0, 1, 2, 3, 4],
    [4, 5, 6, 1, 0],
    [0, 1, 0, 2, 3],
    [3, 4, 1, 2, 1]
], dtype=tf.float32)
input_data = tf.reshape(input_data, (1, 5, 5, 1))  # (Batch, Height, Width, Channels)
```

```
# Filtre 3x3
filter_data = tf.constant([
    [1, 0, -1],
    [1, 0, -1],
    [1, 0, -1]
], dtype=tf.float32)
filter_data = tf.reshape(filter_data, (3, 3, 1, 1))  # (Height, Width, Input Channels, Output Channels)
# Convolution sans padding
output_no_padding = tf.nn.conv2d(input_data, filter_data, strides=1, padding='VALID')
# Convolution avec padding
output_with_padding = tf.nn.conv2d(input_data, filter_data, strides=1, padding='SAME')
print("Sortie sans padding :")
print(output_no_padding.numpy().squeeze())
print("\nSortie avec padding :")
print(output_with_padding.numpy().squeeze())
```

2- Couche d'activation :

```
from tensorflow.keras.layers import ReLU
# Exemple de données
data = tf.constant([[-1.0, 0.0, 1.0], [2.0, -3.0, 4.0]], dtype=tf.float32)
# Appliquer la fonction ReLU
relu = ReLU()
```

```
output_relu = relu(data)

print("Activation ReLU :\n", output_relu.numpy())
```

3- Couche de sous-échantillonnage (Pooling):

```
# Exemple de Max Pooling

input_data = tf.constant([

    [1, 3, 2, 1],

    [4, 6, 5, 3],

    [7, 8, 9, 4],

    [2, 3, 1, 0]

], dtype=tf.float32)

input_data = tf.reshape(input_data, (1, 4, 4, 1))

# Max Pooling 2x2 avec stride 2

output_pooling = tf.nn.max_pool2d(input_data, ksize=2, strides=2, padding='VALID')

print("Résultat après Max Pooling :")

print(output_pooling.numpy().squeeze())
```

4- Couche de normalisation :

```
from tensorflow.keras.layers import BatchNormalization

# Exemple de données

data = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]], dtype=tf.float32)

# Normalisation Batch

batch_norm = BatchNormalization()

normalized_data = batch_norm(data, training=True)

print("Données après normalisation batch :\n", normalized_data.numpy())
```

5- Couche de fusion :

```
# Exemple de fusion par addition

input1 = tf.constant([[1.0, 2.0, 3.0]])

input2 = tf.constant([[0.5, 1.5, 2.5]])

merged = tf.add(input1, input2)

print("Résultat de la fusion (addition) :\n", merged.numpy())
```

6- Couche entièrement connectée :

```
from tensorflow.keras.layers import Dense

# Exemple d'entrée

input_data = tf.constant([[1.0, 2.0, 3.0]], dtype=tf.float32)

# Couche dense avec 2 neurones

dense_layer = Dense(2, activation='relu')

output_dense = dense_layer(input_data)

print("Sortie de la couche entièrement connectée :\n", output_dense.numpy())
```

Travail à faire :

1- Écrire une fonction Python qui calcule la taille de sortie d'une couche convolutionnelle. La fonction doit prendre en entrée :

- La taille de l'entrée (largeur et hauteur).
- La taille du filtre (ex : 3x3 ou 5x5).
- Le stride.
- Le padding.

2- Tester la fonction avec différents paramètres :

- Entrée : 32×32, Filtre : 3×3, Stride : 1, Padding : 0.
- Entrée : 64×64, Filtre : 5×5, Stride : 2, Padding : 1.

3- Implémenter une convolution avec un filtre 3×3 appliqué à une image fictive (par exemple, une matrice 5×5).

4- Comparer les résultats de la convolution avec et sans padding (VALID et SAME).

5- Quelle est la formule du padding **P** à utiliser pour garantir que la taille de sortie après une convolution reste identique à celle de l'entrée ? Justifier votre réponse.

6- Visualiser l'entrée, le filtre et la sortie avec matplotlib.

7- Implémenter manuellement les fonctions d'activation suivantes et tester leur effet sur un jeu de données fictif :

- ReLU
- Sigmoïde
- Tanh

8- Tracer les courbes des fonctions d'activation dans un graphique (intervalle : $[-3,3]$).

9- Appliquer un Max Pooling 2×2 avec un stride de 2 sur une image fictive 4×4.

10- Appliquer un Average Pooling 2×2 sur la même image et comparer les résultats.

11- Visualiser les résultats avec matplotlib.

12- Générer un ensemble de données aléatoires 10×3 (10 exemples avec 3 caractéristiques).

13- Appliquer une normalisation batch sur cet ensemble et observer les résultats avant et après normalisation.

14- Expliquer pourquoi la normalisation batch est importante pour l'entraînement des CNN.

15- Générer deux matrices 3×3 représentant deux cartes de caractéristiques différentes.

16- Appliquer une fusion par :

- Addition.
- Multiplication.
- Concaténation.