

# TP3 Apprentissage profond

## Introduction aux réseaux de neurones convolutifs

Jérôme Buisine  
[jerome.buisine@univ-littoral.fr](mailto:jerome.buisine@univ-littoral.fr)

7 décembre 2023

Durée : 2h

---

L'objectif de ce TP est la prise en main de la librairie Keras afin d'évaluer les performances des réseaux de neurones convolutifs.

### 1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles durant ce TP :

- 1. [Documentation](#) officielle de l'outil Git ;
- 2. [Gitlab](#) : interface web pour la gestion de projets versionnés sous Git ;
- 3. [pyenv](#) : permet la gestion d'environnement Python.
- 4. [matplotlib](#) : librairie Python qui permet un affichage rapide de données.
- 5. [jupyter](#) : un outil de travail permettant une interaction rapide et visuelle avec une console Python.
- 6. [scikit-learn](#) : documentation de l'API `scikit-learn` utile pour le *machine learning*.
- 7. [Keras](#) : documentation de l'API Keras pour le *deep learning*.

### 2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `tp3-cnn`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

---

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

---

### 3 Premier réseau de neurones convolutif

Il vous faudra tout d'abord installer les dépendances qui nous permettront d'utiliser les API Keras (via `tensorflow`), `tensorboard` pour du monitoring durant/après l'entraînement d'un model et `seaborn` pour un affichage plus rapide :

---

```
pip install tensorflow seaborn tensorboard
```

---

Puis créez un nouveau notebook nommé « `cnn_mnist.ipynb` ». L'objectif est de proposer un premier réseau de neurones convolutif pour afin d'améliorer les performance du modèle précédent.

Pour rappel, la base de données `Mnist` sur laquelle nous allons travailler est disponible via `Keras` :

---

```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
print("x_train : ", x_train.shape)
```

---

**Tâche 1 :** Comme dans le TP précédent, proposez une normalisation des données par la valeur maximale d'une images 8 bits, soit 255 afin que chaque valeur de pixel soit  $\in [0, 1]$ . Cette normalisation doit être faites sur les deux ensembles d'entrées : apprentissage et validation.

Il sera nécessaire de redimensionner vos images en entrée pour appliquer une couche convolutive :

---

```
# redimensionnement de chaque image en 28 * 28 * 1
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)
```

---

**Tâche 2 :** Créer maintenant une fonction `get_model_cnn_mnist` qui prend en paramètre la géométrie des données de celui-ci et retourne un modèle.

- Il sera composé d'un layer `Input` qui prendra en paramètre la géométrie des données d'entrée, ici  $28 \times 28 \times 1$  (un canal de couleur : niveau de gris).
- Les entrées seront passées à 2 layers `Conv2D` avec chacun un kernel de  $3 \times 3$ , un stride de 1 et pas de padding. Ces layers retourneront respectivement un nombre de 8 et 16 cartes de caractéristiques (*features maps*). Les fonctions d'activations seront des `ReLU`.
- D'un layer `Flatten` qui permet l'aplatissement des données en un vecteur afin de pouvoir connecter correctement nos couches suivantes.
- Puis, de deux layers `Dense`, tous deux composés de fonctions d'activation de type `ReLU` et respectivement 128 et 32 neurones.
- Une couche de `Dropout` après chaque couche cachée.
- Enfin, un dernier layer de type `Dense` avec un nombre de neurones correspondant au nombre de classes à prédire. Il s'agira du layer de sortie.

**Tâche 3 :** Observez le nombre de paramètres de votre modèle avec et sans couche convolutive. Puis, entraînez le avec les paramètres suivants :

- Les données d'entrée de la base d'apprentissage et les prédictions attendues.
- Un nombre d'époque à 20.
- Un batch de taille 512.
- Des données de validation (`validation_data`) qui seront les données de test.
- Une fonction callback de sauvegarde spécifique au modèle.

## Questions

- Quels sont les performances de ce modèle vis-à-vis du précédent ?
- Que pouvez vous souligner sur les performances de ce modèle relativement à son nombre de paramètres ?
- Quelle couche du réseau implique beaucoup de paramètres et pourquoi ?

## 4 Vers un modèle simplifié

Dans cette partie du TP, nous allons proposer une seconde version du modèle dans lequel nous allons utiliser des couches de « pooling ». Ces couches vont nous permettre une réduction rapides des dimensions des données.

**Tâche 4 :** Créer un nouveau modèle intégrant le principe de pooling. Vous pouvez observer le nombre de paramètres de ce modèle une fois créé. Les modifications apportées au modèle seront les suivantes :

- Une couche de MaxPooling sera ajoutée après chaque couche Conv2D.
- Chaque couche de MaxPooling aura un kernel de  $2 \times 2$ .

Avant de procéder à l'entraînement de notre nouveau modèle. Nous allons ici utiliser `tensorboard` pour visualiser rapidement les performances de notre modèle. Pour cela, vous pouvez ajouter un `callback` lors de son apprentissage :

---

```
log_dir = 'logs/cnn_mnist_v2'  
os.makedirs(log_dir, exist_ok=True)  
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=log_dir,  
                                                    histogram_freq=1)
```

---

**Note :** il est possible de spécifier un dossier de log pour chaque modèle (dans son callback). Ainsi, les modèles pourront être observés et comparés.

Pour accéder à l'interface de `tensorboard`, il est nécessaire de l'exécuter depuis le terminal en lui précisant le dossier de log :

---

```
tensorboard --logdir tp3-cnn/logs
```

---

Il est ensuite possible d'accéder à l'interface web de `tensorboard` depuis votre localhost et le port 6006 (port par défaut).

**Tâche 5 :** Nous allons maintenant proposer une 3<sup>e</sup> version de notre modèle en y ajoutant du dropout. Une fois le modèle créé, vous pouvez l'entraîner. Les modifications au modèle seront les suivantes :

- Une couche de Dropout sera ajoutée après chaque couche de MaxPooling. Le pourcentage sera fixé à hauteur de 20%.
- Une couche de BatchNormalization sera ajoutée après chaque couche Conv2D et Dense.
- Les couches Dropout après chaque couche Dense auront maintenant un pourcentage fixé à hauteur de 50%.
- Un callback spécifique pour `tensorboard` sera affecté au modèle lors de son apprentissage

## Questions

- Quelles améliorations ont été apportées aux deux dernières versions de modèles ?
- Quel est le principe d'ajout de dropout après une couche de pooling dans le cadre de l'apprentissage d'un modèle ? Cela apporte-t-il une meilleure performance dans notre cas ?

## 5 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « tp3 » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.