**ETIAC 2024 : ATELIER DEEP LEARNING**

**Objectifs :**

* Construire un modèle (ANN et/ou CNN).
* Entraîner et évaluer les performances.
* Tester des prédictions sur de nouvelles images.

**Avant de commencer…**

* Qu'est-ce qu'une fonction d'activation ? Pourquoi est-elle importante ?
* Quelles sont, selon vous, les étapes pour entraîner un modèle de deep learning ?
* Pourquoi devons-nous normaliser ou standardiser les données avant d'entraîner un modèle ?
* Citez deux autres architectures de Deep Learning à part les ANN et les CNN.
* Dites pour quels types de problèmes il est préférable de les utiliser.
* Lorsqu’on veut faire des prédictions futures en tenant compte des informations passées, quelle est l’architecture DL indiquée ??
* Comment choisit-on le nombre d'epochs et la taille des batchs ?
* Pourquoi utilisons-nous la fonction sigmoid dans la couche de sortie pour une tâche de classification binaire ?

**Exploration de colab**

Ajout de code

Ajout de Texte

Utilisation GPU / TPU

GEMENI intégré

Enregistrer le code

Enregistrer le notebook

Importer des fichiers

Travailler avec des fichiers en local

### ****Travail Pratique 1 : Classification des Chiffres Manuscrits avec un Modèle de Deep Learning****

#### ****Description du Problème Résolu :****

Le **jeu de données MNIST** (Modified National Institute of Standards and Technology) est l'un des jeux de données les plus populaires pour tester les modèles de machine learning, particulièrement dans le domaine de la **vision par ordinateur**. Il contient 70 000 images de chiffres manuscrits (0-9), chacune ayant une taille de 28x28 pixels, en niveaux de gris.



Dans ce TP, nous allons construire un modèle de **Deep Learning** pour classer ces chiffres. Nous allons commencer par utiliser un **modèle de réseau de neurones artificiels (ANN)** ou un **réseau neuronal convolutif (CNN)** de base, pour effectuer la classification des chiffres manuscrits.

L'objectif de ce TP est de **concevoir et entraîner un modèle simple** capable de reconnaître ces chiffres, et de comprendre les différentes étapes de l'entraînement d'un modèle de deep learning : **prétraitement des données**, **conception du modèle**, **entraîner le modèle**, et **évaluer ses performances**.

#### ****Problématique :****

Comment entraîner un modèle de Deep Learning pour effectuer la classification des chiffres manuscrits à partir du jeu de données MNIST en utilisant des architectures simples comme les réseaux de neurones artificiels (ANN) ou convolutifs (CNN) ?

#### ****Objectifs Pédagogiques :****

1. **Comprendre le prétraitement des données** : Normaliser et redimensionner les images pour les rendre compatibles avec le modèle de Deep Learning.
2. **Construire un modèle de classification d'images** avec des réseaux de neurones artificiels (ANN) ou un réseau de neurones convolutif (CNN).
3. **Entraîner un modèle de Deep Learning** sur les données de MNIST.
4. **Évaluer la performance du modèle** en utilisant des métriques comme la précision et la perte.
5. **Visualiser les courbes d'entraînement** pour observer la progression du modèle et éviter le sur-apprentissage (overfitting).
6. **Effectuer des prédictions** sur de nouvelles images et tester la capacité du modèle à reconnaître des chiffres.

#### ****Méthodologie :****

1. **Chargement et Prétraitement des Données** :
   * Charger le jeu de données MNIST.
   * Redimensionner les images en un format adapté aux modèles de deep learning (normalisation des pixels, ajout de dimensions de canal).
   * Diviser le jeu de données en ensemble d'entraînement et de test.
2. **Construction du Modèle** :
   * Choisir un modèle de base : ANN ou CNN.
   * Définir les couches du modèle, y compris les couches de convolution, de pooling, de dropout, etc.
   * Compiler le modèle en choisissant un optimiseur, une fonction de perte et une métrique.
3. **Entraînement du Modèle** :
   * Entraîner le modèle sur les données d'entraînement.
   * Utiliser des techniques comme la validation croisée pour évaluer la performance pendant l'entraînement.
4. **Évaluation du Modèle** :
   * Utiliser les données de test pour évaluer la performance du modèle.
   * Afficher les courbes de **précision** et **perte** pour comprendre l'évolution de l'entraînement.
5. **Prédictions sur de Nouvelles Images** :
   * Tester le modèle avec des images nouvelles.
   * Vérifier la capacité du modèle à faire des prédictions précises sur des images non vues pendant l'entraînement.

import tensorflow as tf

import numpy as np

# Charger le dataset MNIST

mnist = tf.keras.datasets.mnist

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

# Ajouter une dimension de canal pour les images en niveaux de gris

train\_images = train\_images.reshape(-1, 28, 28, 1) / 255.0

test\_images = test\_images.reshape(-1, 28, 28, 1) / 255.0

# Vérifier les formes des données

print(f"Train Images Shape: {train\_images.shape}")

print(f"Test Images Shape: {test\_images.shape}"

################

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Définir un générateur d'augmentation

datagen = ImageDataGenerator(

rotation\_range=10, # Rotation jusqu'à 10 degrés

width\_shift\_range=0.1, # Décalage horizontal

height\_shift\_range=0.1, # Décalage vertical

zoom\_range=0.1 # Zoom sur l'image

)

# Ajuster le générateur sur les données d'entraînement

datagen.fit(train\_images)

##################

from tensorflow.keras import layers, models

# Définir un modèle CNN

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)), # Convolution

layers.BatchNormalization(), # Normalisation

layers.MaxPooling2D((2, 2)), # Pooling

layers.Conv2D(64, (3, 3), activation='relu'), # Deuxième couche de convolution

layers.BatchNormalization(),

layers.MaxPooling2D((2, 2)),

layers.Flatten(), # Aplatir

layers.Dense(128, activation='relu'), # Couche dense

layers.Dropout(0.5), # Dropout pour éviter le sur-apprentissage

layers.Dense(10, activation='softmax') # Couche de sortie

])

# Compiler le modèle

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Entraîner avec le générateur d'augmentation

history = model.fit(

datagen.flow(train\_images, train\_labels),

epochs=30,

validation\_data=(test\_images, test\_labels),

verbose=1

)

#################### » »

import matplotlib.pyplot as plt

# Fonction pour tracer les courbes

def plot\_training\_curves(history):

plt.figure(figsize=(12, 5))

# Loss

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

# Accuracy

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

# Appeler la fonction

plot\_training\_curves(history)

############### »

# Évaluer le modèle

test\_loss, test\_accuracy = model.evaluate(test\_images, test\_labels, verbose=2)

print(f"Test Accuracy: {test\_accuracy \* 100:.2f}%")

#####################”

# Afficher des prédictions sur 10 exemples du jeu de test

predictions = model.predict(test\_images)

plt.figure(figsize=(12, 6))

for i in range(10):

plt.subplot(2, 5, i + 1)

plt.imshow(test\_images[i].reshape(28, 28), cmap='gray')

true\_label = test\_labels[i]

pred\_label = np.argmax(predictions[i])

plt.title(f"True: {true\_label}, Pred: {pred\_label}")

plt.axis('off')

plt.tight\_layout()

plt.show()

###########

from google.colab import files

from PIL import Image

# Fonction pour charger une image utilisateur

def predict\_user\_image():

print("Veuillez charger une image de chiffre (taille 28x28, en niveaux de gris).")

uploaded = files.upload()

for filename in uploaded.keys():

# Charger et préparer l'image

img = Image.open(filename).convert('L') # Convertir en niveaux de gris

img = img.resize((28, 28)) # Redimensionner à 28x28

img\_array = np.array(img) / 255.0 # Normaliser

img\_array = img\_array.reshape(1, 28, 28, 1) # Ajouter la dimension batch

# Prédire avec le modèle

prediction = model.predict(img\_array)

predicted\_label = np.argmax(prediction)

# Afficher l'image et la prédiction

plt.imshow(img\_array[0].reshape(28, 28), cmap='gray')

plt.title(f"Prédiction : {predicted\_label}")

plt.axis('off')

plt.show()

# Tester avec une image utilisateur

predict\_user\_image()

### Explications détaillées des différentes parties :

#### 1. ****Préparation des données****

* Les images et labels sont chargés depuis MNIST avec tf.keras.datasets.mnist.load\_data().
* Les images sont normalisées (pixels entre 0 et 1) et mises en forme pour un CNN avec 4 dimensions (batch\_size, 28, 28, 1).

#### 2. ****Augmentation des données****

* Utiliser un générateur de données comme ImageDataGenerator permet d'améliorer la généralisation du modèle en augmentant la diversité des échantillons vus durant l'entraînement.

#### 3. ****Construction du modèle CNN****

* Le CNN est structuré avec des couches suivantes :
  + **Couches convolutionnelles (Conv2D)** pour extraire les caractéristiques des images.
  + **Normalisation (BatchNormalization)** pour accélérer la convergence.
  + **Couches de pooling (MaxPooling2D)** pour réduire la taille spatiale tout en préservant les caractéristiques importantes.
  + **Flatten** pour aplatir les caractéristiques extraites.
  + **Dense** pour connecter pleinement les neurones.
  + **Dropout** pour réduire le surapprentissage.

#### 4. ****Compilation et entraînement****

* Le modèle est compilé avec :
  + **Adam optimizer**, un optimiseur performant.
  + **Sparse Categorical Crossentropy** comme fonction de perte (adaptée aux labels entiers).
  + metrics=['accuracy'] pour surveiller la précision.
* L'entraînement est réalisé avec des données augmentées, et les performances sont validées sur les données de test.

#### 5. ****Traçage des courbes d'entraînement****

* Une fonction dédiée trace les courbes pour analyser l'évolution de la perte et de la précision sur l'ensemble d'entraînement et de validation.

#### 6. ****Évaluation du modèle****

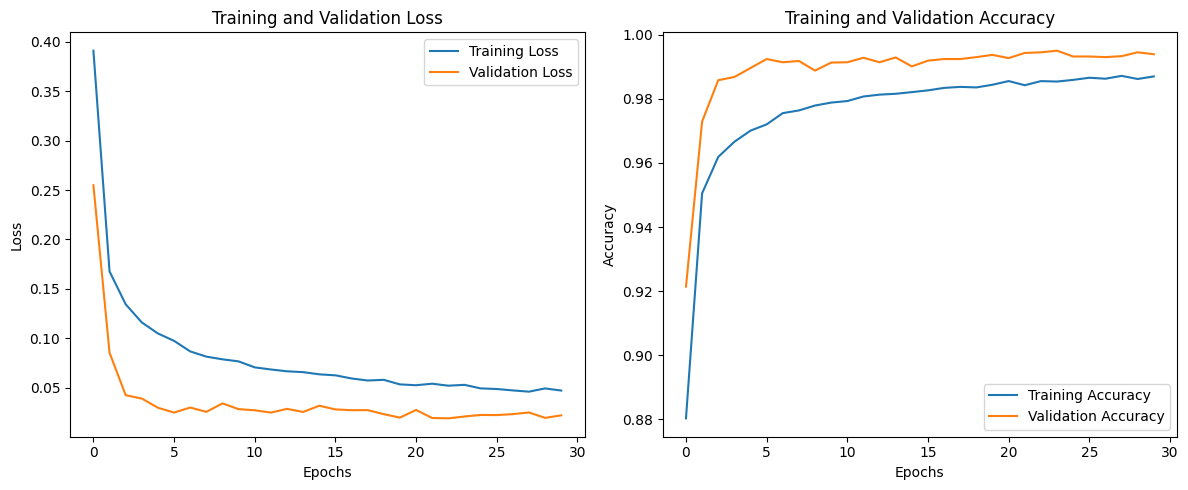
* Après l'entraînement, les performances du modèle sont évaluées sur les données de test pour calculer la précision.

#### 7. ****Visualisation des prédictions****

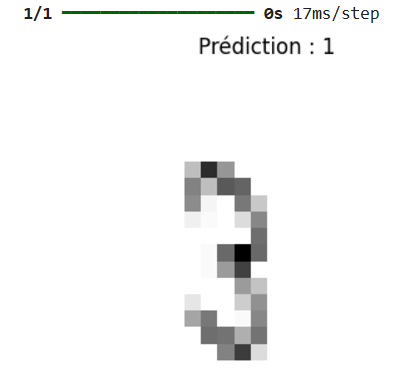
* Le modèle prédit les classes pour 10 images du jeu de test, affichées avec leurs prédictions et labels réels.

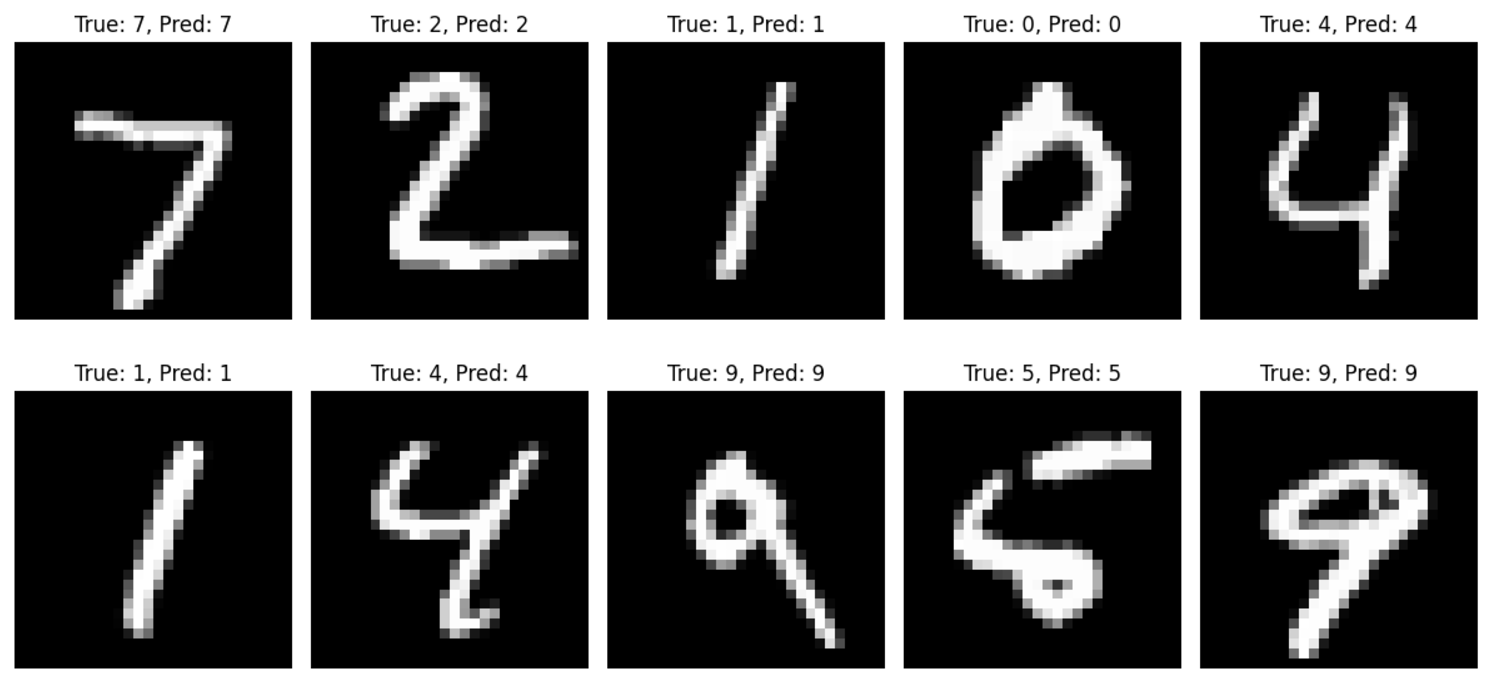
#### 8. ****Prédiction interactive d'une image utilisateur****

* L'utilisateur peut charger une image 28x28 en niveaux de gris.
* Le modèle effectue la prédiction après avoir normalisé et redimensionné l'image.



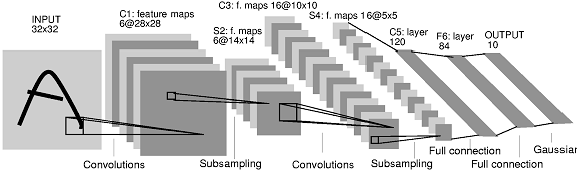






**A FAIRE**

* **Créez un script pour mettre en place un modèle pour traiter les images d’entrée de MNIST, basé sur l’architecture suivante :** 
  + Une couche de convolution avec 32 filtres de taille 5×5, suivie d’une non linéarité de type sigmoïde puis d’une couche de max pooling de taille 2×2
  + Une seconde couche de convolution avec 64 filtres de taille 5×5, suivie d’une non linéarité de type sigmoïde puis d’une couche de max pooling de taille 2×2
  + Comme dans le réseau LeNet, on considérera la sortie du second bloc convolutif comme un vecteur, ce que revient à « mettre à plat » les couches convolutives précédentes (model.add(Flatten())).
  + Une couche complètement connectée de taille 100, suivie d’une non linéarité de type sigmoïde.
  + Une couche complètement connectée de taille 10, suivie d’une non linéarité de type softmax.

[](http://cedric.cnam.fr/vertigo/cours/ml2/_images/LeNet5.png)

* Compilez le modèle. Quelle précision obtenez-vous ? améliorer votre modèle pour obtenir au moins 99% de précision.
* Quelle est le temps d’une époque avec ce modèle convolutif ?
* Quel est le nombre de paramètres de votre modèle ??
* Tester votre modèle en lui soumettant une photo d’un caractère écrit à la main.
* Enregistrez votre model afin de pouvoir le réutiliser.

**Travail pratique 2 : Problème de classification avec MLP**

Base de données d’une banque, sur 10000 clients (id, nom, pays, genre, âge, score, balance, …)

Problème à résoudre : problème de classification, prédire les clients susceptibles de quitter la banque dans 6 mois.

**Préparation des données**

Importation du jeu de données puis spécification des colonnes (variables) à considérer. Par exemple le numéro des lignes, l’identifiant, le nom du client ne seront pas considérés car on estime qu’ils n’influencent pas la probabilité qu’un client quitte ou pas une banque.)

Petite précaution pour Google colab:

* Importer les données dans votre drive
* spécifier au préalable l’emplacement des données
* Importer le jeu de données

from google.colab import drive

drive.mount('/content/drive')

dataset = pd.read\_csv('/content/drive/My Drive/Churn\_Modelling.csv')

Dans le jeu de données, il y a des variables catégoriques (**Genre, Pays**) qu’il faut réenconder correctement. Pour le genre, comme il n y a que deux catégories, cela ne pose pas de problème (1 ou 0), on peut juste utiliser le **LabelEncode**r. Par contre pour le pays, il est nécessaire de passer par l’étape de réencodage avec **ColumnTransformer** et **OneHotEncoder**

Pour que les données soient sur la même échelle, on importe le module StandardScaler qui permet de normaliser les données (entre -2 et 2 environ).

Après la préparation des données, on peut construire le réseau de neurones

# Part 1 - Data Preprocessing

# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

from google.colab import files

uploaded = files.upload()

dataset = pd.read\_csv('Churn\_Modelling.csv') # Assurez-vous que le fichier CSV est dans votre répertoire de travail

X = dataset.iloc[:, 3:13].values # Sélection des variables indépendantes (colonnes 3 à 13)

y = dataset.iloc[:, 13].values # Sélection de la variable dépendante (colonne 13)

# Encoding categorical data

from sklearn.preprocessing import LabelEncoder

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import OneHotEncoder

# Encoding categorical data

labelencoder\_X\_2 = LabelEncoder()

X[:, 2] = labelencoder\_X\_2.fit\_transform(X[:, 2]) # Encodage de la colonne 2

# OneHotEncoder pour la colonne 1

columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(), [1])], remainder='passthrough')

X = np.array(columnTransformer.fit\_transform(X), dtype = str)

X = X[:, 1:] # Supprimer la première colonne après l'encodage OneHot

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) # Appliquer la mise à l'échelle sur les données d'entraînement

X\_test = sc.transform(X\_test) # Appliquer la mise à l'échelle sur les données de test

# Part 2 - Now let's make the ANN!

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Dense

# Initializing the ANN

classifier = Sequential()

# Adding the input layer and the first hidden layer

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu', input\_dim = 11))

# Adding the second hidden layer

classifier.add(Dense(units = 6, kernel\_initializer = 'uniform', activation = 'relu'))

# Adding the output layer

classifier.add(Dense(units = 1, kernel\_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN

classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set

classifier.fit(X\_train, y\_train, batch\_size = 10, epochs = 100)

# Part 3 - Making predictions and evaluating the model

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

y\_pred = (y\_pred > 0.5) # Si la probabilité est supérieure à 0.5, on prédit 1, sinon 0

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Afficher la matrice de confusion

#print("Confusion Matrix:")

#print(cm)

**Application :**

Utilisez notre modèle ANN pour prédire si le client avec les informations suivantes va quitter la banque :

Géographie : France

Score de crédit : 600

Sexe : Homme

Age : 40 ans

Durée d'occupation : 3 ans

Solde : 60000

Nombre de produits : 2

Ce client possède-t-il une carte de crédit ? Oui

Est-ce que ce client est un membre actif : Oui

Salaire estimé : $ 50000

Alors, devons-nous dire au revoir à ce client ?

new\_prediction = classifier.predict(sc.transform(np.array([[0.0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])))

new\_prediction = (new\_prediction > 0.5)

print("Prediction: ", new\_prediction)

**Note importante 1 :** remarquez que les valeurs des caractéristiques ont toutes été saisies dans une double paire de crochets. C'est parce que la méthode "**predict"** s'attend toujours à un tableau 2D comme format de ses entrées. Et en mettant nos valeurs dans une double paire de crochets, l'entrée est exactement un tableau 2D.

**Note importante 2 :** Remarquez également que le pays "France" n'a pas été saisi comme une chaîne de caractères dans la dernière colonne mais comme "1, 0, 0" dans les trois premières colonnes. Cela s'explique par le fait que la méthode de prédiction s'attend bien sûr à ce que les valeurs de l'état soient codées en un coup, et comme nous le voyons dans la première ligne de la matrice des caractéristiques X, "France" a été codé comme "1, 0, 0". Et attention à inclure ces valeurs dans les trois premières colonnes, car les variables muettes sont toujours créées dans les premières colonnes.

**Installation de Anaconda**

**Instructions d'installation des modules**

Ouvrir le terminal avec le raccourci clavier "Win + R" et exécuter "cmd". Puis taper les commandes suivantes :

* conda create --name myenv python=3.10
* conda activate myenv
* conda install tensorflow
* python -c "import tensorflow as tf; print(tf.\_\_version\_\_)"

**Note :**Il est important de lancer Spyder à partir de l'environnement qu'on vient de créer et de s'assurer qu'on utilise bien Python 3.10 au moins. Sinon, il ne trouvera pas keras. Ça veut dire qu'à chaque fois que vous voulez lancer Spyder, vous ouvrez le terminal, vous activez l'environnement avec activate myenv, et ensuite vous lancez Spyder en tapant spyder.

