TP 2 : Apprentissage Profond avec TensorFlow / Keras

L'**apprentissage profond (Deep Learning)** est une branche de l'**apprentissage automatique** qui s'appuie sur des **réseaux de neurones artificiels** comportant **plusieurs couches** (*deep networks*).
Ces modèles permettent à une machine d'apprendre à partir de grandes quantités de données sans qu'on doive définir explicitement les règles.

Exemples d'applications :

- Reconnaissance d'images (vision par ordinateur)
- Reconnaissance vocale
- Traduction automatique
- Conduite autonome
- Détection de fraudes, analyse médicale, etc.

Concepts de base

Concept	Description
Neuron (Neurone)	Unité de base d'un réseau. Reçoit des entrées, effectue un calcul pondéré, applique une fonction d'activation et produit une sortie.
Couche (Layer)	Ensemble de neurones traitant une représentation des données. Les couches sont empilées pour former le réseau.
Fonction d'activation	Fonction non linéaire (comme ReLU, Sigmoid, Softmax) qui permet au modèle de capturer des relations complexes.
Fonction de perte (Loss function)	Mesure la différence entre la prédiction du modèle et la vraie valeur. L'objectif est de la minimiser.
Optimiseur (Optimizer)	Méthode mathématique (Adam, SGD, RMSprop…) qui ajuste les poids du réseau pour améliorer la performance.
Époque (Epoch)	Un passage complet sur tout le jeu de données d'entraînement.
Batch	Sous-ensemble du jeu de données utilisé à chaque itération pendant l'entraînement.
Rétropropagation (Backpropagation)	Algorithme qui calcule comment ajuster les poids en fonction de l'erreur obtenue à la sortie.

Qu'est-ce que TensorFlow ?

TensorFlow est une bibliothèque open-source développée par **Google Brain**, utilisée pour créer, entraîner et déployer des modèles de **Machine Learning** et **Deep Learning**.
Elle permet de manipuler des *tensors* (n-dimensions de données), d'exécuter des calculs sur CPU ou GPU, et d'automatiser le processus d'apprentissage.

Qu'est-ce que Keras ?

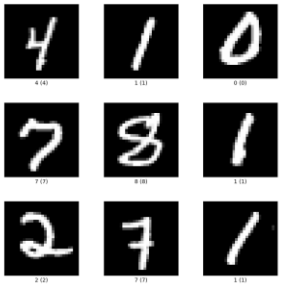
Keras est une API haut-niveau intégrée dans TensorFlow (`tf.keras`) qui simplifie la création de réseaux de neurones.
Elle est intuitive, modulaire et idéale pour les étudiants et chercheurs.
Exemple : au lieu d'écrire manuellement le calcul du gradient, Keras s'en charge automatiquement.

Pourquoi TensorFlow + Keras ?

- TensorFlow = moteur d'exécution performant
- Keras = interface simple pour construire des modèles
  Ensemble, ils offrent puissance **et** facilité d'utilisation.

Exercice 1 : Reconnaissance de chiffres manuscrits (MNIST)

L'objectif est de construire un **réseau de neurones entièrement connecté (Dense)** avec **TensorFlow/Keras** capable de reconnaître des **chiffres manuscrits (0–9)** à partir du jeu de données **MNIST**.



Dataset link : https://www.kaggle.com/datasets/hojjatk/mnist-dataset

Les étapes à suivre :

1. Charger et explorer les données
2. Normaliser et visualiser les images
3. Créer et compiler un modèle de réseau de neurones
4. L'entraîner et évaluer ses performances
5. Visualiser les courbes d'apprentissage et les prédictions

1. Import des bibliothèques

Importez les modules nécessaires :

- `tensorflow` pour la création et l'entraînement du modèle
- `matplotlib` pour la visualisation
- `numpy` pour les opérations mathématiques

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense, Flatten
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np

print("TensorFlow version:", tf.__version__)
```

Questions :

1. Quelle est la différence entre TensorFlow et Keras ?
2. À quoi sert la couche `Flatten` dans un réseau de neurones ?
3. Quelle version de TensorFlow est installée sur votre machine ?
4. À quoi servent `Sequential` et `Dense` ?

2. Chargement et préparation du dataset

Le dataset **MNIST** contient 70 000 images de chiffres manuscrits (28×28 pixels, niveaux de gris).

- `x_train` : images d'entraînement
- `y_train` : labels correspondants (0–9)
- `x_test`, `y_test` : données pour l'évaluation

Nous normalisons les valeurs des pixels (entre 0 et 1) pour accélérer la convergence.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalisation
x_train = x_train / 255.0
x_test = x_test / 255.0

print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
```

Questions :

1. Pourquoi normalise-t-on les valeurs des pixels ?
2. Quelle est la taille d'une image et son type de données ?
3. Combien d'exemples contient le jeu d'entraînement ?

3. Visualisation d'échantillons

Avant d'entraîner le modèle, observons quelques images pour vérifier que les données sont correctes.

```
plt.figure(figsize=(8,3))
for i in range(10):
    plt.subplot(2,5,i+1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

À faire :

- Identifier les chiffres affichés
- Vérifier que les labels correspondent bien aux images

Questions :

1. Quelle commande permet de modifier le nombre d'images affichées ?
2. Que représente l'argument `cmap='gray'` ?
3. Quelle utilité a cette étape dans un projet d'IA ?

4. Construction du modèle de réseau de neurones

Le modèle **Sequential** est une pile linéaire de couches :

1. `Input(shape=(28,28))` → définit la taille de l'entrée
2. `Flatten()` → transforme chaque image 2D en vecteur 1D (784 valeurs)
3. `Dense(128, activation='relu')` → couche cachée avec 128 neurones
4. `Dense(10, activation='softmax')` → couche de sortie avec 10 classes (0–9)

```
model = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.summary()
```

Questions :

1. Combien de paramètres le modèle contient-il ?
2. Quelle est la fonction d'activation utilisée dans la couche de sortie ? Pourquoi ?
3. Quelle est la différence entre `ReLU` et `Softmax` ?

5. Compilation du modèle

On définit :

- **L'optimiseur** (`adam`) : ajuste les poids pendant l'apprentissage
- **Loss function** : calcule l'erreur entre la sortie et la vérité
- **La métrique** (`accuracy`) : évalue la précision du modèle

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Questions :

1. Qu'est-ce qu'un optimiseur ?
2. Pourquoi utilise-t-on une fonction de perte ?
3. Que se passe-t-il si on change la fonction de perte ?
4. Quelle serait la conséquence d'un mauvais choix d'optimiseur ?
5. Quelle est la différence entre `loss` et `metrics` ?

6. Entraînement du modèle

On entraîne le modèle sur les données d'entraînement pendant **5 époques**.
L'entraînement ajuste les poids pour minimiser la perte.

```
history = model.fit(
    x_train, y_train,
    epochs=5,
    batch_size=32,
    validation_data=(x_test, y_test)
)
```

À observer :

- La précision d'entraînement augmente-t-elle ?
- La précision de validation suit-elle la même tendance ?

Questions :

1. Que représente une "époque" ?
2. Pourquoi utilise-t-on un jeu de validation ?
3. Que se passerait-il si on augmentait le nombre d'époques à 50 ?

7. Évaluation du modèle

On teste le modèle sur des données **jamais vues** pendant l'entraînement.

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

Questions :

1. Quelle est la précision finale de votre modèle ?
2. Pourquoi la précision du test est souvent inférieure à celle d'entraînement ?
3. Que faire si le modèle surapprend (*overfitting*) ?

8. Visualisation des performances

Les courbes permettent de voir si le modèle apprend bien ou surapprend.

```
plt.figure(figsize=(8,4))
plt.plot(history.history['accuracy'], label='Précision entraînement')
plt.plot(history.history['val_accuracy'], label='Précision validation')
plt.title('Courbe de précision')
plt.xlabel('Époques')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```

Questions :

1. Que remarquez-vous sur les deux courbes ?
2. Comment détecter visuellement un surapprentissage ?
3. Quelle stratégie pourrait-on utiliser pour y remédier ?
4. Que faire si les deux courbes divergent ?

9. Prédictions et visualisation

Nous affichons quelques images testées avec leur prédiction.

```
predictions = model.predict(x_test)

plt.figure(figsize=(10,4))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Vrai: {y_test[i]}\nPrédit: {np.argmax(predictions[i])}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Questions :

1. Le modèle fait-il des erreurs ? Sur quelles images ?
2. Comment expliquer une prédiction erronée ?
3. Quelle technique permettrait d'améliorer la précision (ex : CNN) ?

Travail demandé aux étudiants

À inclure dans le rapport final :

1. Le **code complet** du TP

2. Les **courbes de précision et de perte**
3. La **valeur de précision finale** obtenue
4. Une **analyse personnelle** (5–10 lignes) :
    - Interprétation des résultats
    - Difficultés rencontrées
    - Propositions d'amélioration (plus de couches, changement d'activation, autres architectures, hyperparamètres, etc.)

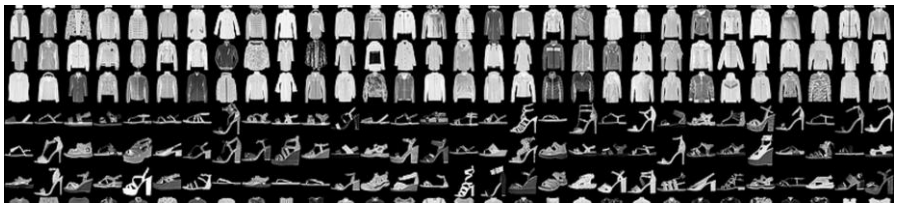
Exercice 2 : classification d'images en utilisant le dataset Fashion-MNIST

Objectif : Découvrir et expérimenter les principes du **Deep Learning** à travers la **classification d'images** en utilisant le dataset **Fashion-MNIST**.

Les étudiants apprendront à :

- Construire un **réseau de neurones artificiels (ANN)** puis un **réseau convolutif (CNN)**.
- Analyser les résultats à l'aide de métriques et de courbes.
- Comprendre le rôle des **fonctions d'activation** et expérimenter leurs effets.

Dataset : Fashion-MNIST



Dataset link : https://www.kaggle.com/datasets/zalando-research/fashionmnist/

Jeu de données : Fashion-MNIST :

- 70 000 images (60 000 pour l'entraînement, 10 000 pour le test)
- Chaque image : 28 × 28 pixels, niveaux de gris
- 10 classes de vêtements (*T-shirt, pantalon, pull, robe, manteau, sandale, chemise, basket, sac, bottine*)

Chargement via Keras :

```python
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
import matplotlib.pyplot as plt
import seaborn as sns

# Pour la reproductibilité
tf.random.set_seed(42)

# Chargement du dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

Étape 1 – Exploration du jeu de données

A. Dimensions et types

```python
print("Dimensions du train set :", X_train.shape)
print("Dimensions du test set :", X_test.shape)
```

B. Visualisation d'exemples

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(f"Label : {y_train[i]}")
    plt.axis('off')
plt.show()
```

Question :

Que remarquez-vous sur la nature des données et la taille des images ?

Étape 2 – Prétraitement des données

- Mise à l'échelle des pixels entre 0 et 1
- Conversion en float
- Encodage des labels

```python
from tensorflow.keras.utils import to_categorical

# Normalisation des pixels entre 0 et 1
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Encodage one-hot des labels
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)
```

Questions :

1. Pourquoi faut-il normaliser les images ?
2. Quelle est l'importance du one-hot encoding dans un problème de classification multiclasse ?

Étape 3 – Création d'un modèle dense (ANN)

Création du modèle de base :

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Flatten, Dense, Dropout

model = Sequential([
    Input(shape=(28, 28)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.summary()
```

Compilation et entraînement

```
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    X_train, y_train_cat,
    epochs=10,
    batch_size=128,
    validation_split=0.2
)
```

Visualisation

```
plt.plot(history.history['accuracy'], label='Train acc')
plt.plot(history.history['val_accuracy'], label='Val acc')
plt.legend()
plt.title('Évolution de la précision (ANN)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
```

Questions :

1. Pourquoi utilise-t-on la fonction d'activation **ReLU** ici ?
2. Quelle est la différence entre `accuracy` et `val_accuracy` ?
3. Que se passe-t-il si l'on augmente le nombre d'époques à 50 ?

Évaluation finale :

```
test_loss, test_acc = model.evaluate(X_test, y_test_cat)
print("Précision sur le test set :", test_acc)
```

Étape 4 — Introduction aux Fonctions d'Activation

Objectif :

Comparer les performances des fonctions : **sigmoïde**, **tanh**, **ReLU**, et **LeakyReLU**.

```python
from tensorflow.keras.layers import LeakyReLU

activations = ['sigmoid', 'tanh', 'relu']
results = {}

for act in activations:
    model_act = Sequential([
        Input(shape=(28,28)),
        Flatten(),
        Dense(128, activation=act),
        Dense(64, activation=act),
        Dense(10, activation='softmax')
    ])

    model_act.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    h = model_act.fit(X_train, y_train_cat, epochs=5, batch_size=128, validation_split=0.2, verbose=0)
    results[act] = h.history['val_accuracy'][-1]

# Leaky ReLU
model_leaky = Sequential([
    Input(shape=(28,28)),
    Flatten(),
    Dense(128),
    LeakyReLU(0.1),
    Dense(64),
    LeakyReLU(0.1),
    Dense(10, activation='softmax')
])
model_leaky.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
h = model_leaky.fit(X_train, y_train_cat, epochs=5, batch_size=128, validation_split=0.2, verbose=0)
results['LeakyReLU'] = h.history['val_accuracy'][-1]

plt.bar(results.keys(), results.values())
plt.title("Comparaison des fonctions d'activation")
plt.ylabel("Validation Accuracy")
plt.show()
```

Questions :

1. Quelle fonction d'activation donne les meilleurs résultats ?
2. Pourquoi les fonctions **sigmoïde** et **tanh** sont moins utilisées dans les couches cachées ?
3. À quoi sert **LeakyReLU** ?

Étape 5 : Modèle CNN (Convolutional Neural Network)

Création du modèle CNN

Les étudiants créent un **réseau de neurones convolutif** pour exploiter la structure spatiale des images

```python
from tensorflow.keras.layers import Conv2D, MaxPooling2D

# Reshape pour ajouter la dimension "canal"
X_train_cnn = X_train.reshape(-1, 28, 28, 1)
X_test_cnn = X_test.reshape(-1, 28, 28, 1)

cnn = Sequential([
    Input(shape=(28, 28, 1)),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])
cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cnn.summary()
```

Entraînement

```python
history_cnn = cnn.fit(
    X_train_cnn, y_train_cat,
    epochs=10,
    batch_size=128,
    validation_split=0.2
)
```

Évaluation

```python
test_loss, test_acc = cnn.evaluate(X_test_cnn, y_test_cat)
print("Précision CNN sur test set :", test_acc)
```

Questions :

1. Quelle est la différence entre un **ANN** et un **CNN** ?
2. Pourquoi le CNN donne-t-il généralement de meilleurs résultats ?
3. Quel est le rôle des couches de **Pooling** ?

Étape 6 — Visualisation des performances

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

y_pred = np.argmax(cnn.predict(X_test_cnn), axis=1)
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, cmap="Blues")
plt.title("Matrice de confusion - CNN")
plt.show()
```

Questions :

1. Quelles classes sont le plus souvent confondues ?
2. Comment peut-on réduire ces erreurs de classification ?

Étape 7 : Data Augmentation

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    horizontal_flip=True
)

datagen.fit(X_train_cnn)

cnn.fit(
    datagen.flow(X_train_cnn, y_train_cat, batch_size=128),
    epochs=10,
    validation_data=(X_test_cnn, y_test_cat)
)
```

Questions :

1. Quel est l'intérêt de la **Data Augmentation** ?
2. Quels types de transformations semblent les plus efficaces pour ce dataset ?