TP : Autoencodeur pour le débruitage d'images MNIST

1. Introduction

Dans ce TP, nous allons construire et entraîner un autoencodeur convolutif pour débruiter des images de chiffres manuscrits de la base de données MNIST. Un autoencodeur est un réseau de neurones non supervisé qui apprend une représentation compressée (codage) des données, puis les reconstruit. Ici, nous l'utilisons spécifiquement pour supprimer le bruit ajouté artificiellement aux images.

2. Chargement et préparation des données

Nous utilisons le jeu de données **MNIST**, composé de 60 000 images d'entraînement et 10 000 images de test, chacune de taille 28×28 pixels et en niveaux de gris.

2.1 Import des bibliothèques

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from tensorflow.keras import layers, models
```

2.2 Chargement et normalisation des données

```
# Charger MNIST
(x_train, _), (x_test, _) = keras.datasets.mnist.load_data()

# Normalisation entre 0 et 1
x_train = x_train.astype("float32") / 255.0
x_test  = x_test.astype("float32") / 255.0

# Ajout d'un canal (format : H, W, 1)
x_train = np.expand_dims(x_train, -1)
x_test  = np.expand_dims(x_test, -1)

print(f"Forme des données d'entraînement : {x_train.shape}")
print(f"Forme des données de test : {x_test.shape}")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
Forme des données d'entraînement : (60000, 28, 28, 1)
Forme des données de test : (10000, 28, 28, 1)
```

3. Ajout de bruit artificiel

Pour simuler des images bruitées, nous ajoutons un bruit gaussien.

```
def add_noise(x, noise_factor=0.5):
    noise = np.random.normal(loc=0.0, scale=1.0, size=x.shape)
    x_noisy = x + noise_factor * noise
    return np.clip(x_noisy, 0., 1.)

# Création des versions bruitées
x_train_noisy = add_noise(x_train, 0.5)
x_test_noisy  = add_noise(x_test, 0.5)
```

4. Architecture de l'autoencodeur

L'autoencodeur est composé de deux parties :

- **Encodeur** : réduit la dimension de l'image via des couches convolutives et de pooling.
- **Décodeur** : reconstruit l'image originale à partir du code compressé via des couches transposées.

4.1 Définition du modèle

```
input_img = layers.Input(shape=(28, 28, 1))

# Encodeur
x = layers.Conv2D(32, (3,3), activation="relu", padding="same")(input_img)
x = layers.MaxPooling2D((2,2), padding="same")(x)
x = layers.Conv2D(32, (3,3), activation="relu", padding="same")(x)
encoded = layers.MaxPooling2D((2,2), padding="same")(x)

# Décodeur
x = layers.Conv2DTranspose(32, (3,3), strides=2, activation="relu", padding="same")(encoded)
x = layers.Conv2DTranspose(32, (3,3), strides=2, activation="relu", padding="same")(x)
decoded = layers.Conv2D(1, (3,3), activation="sigmoid", padding="same")(x)

# Modèle complet
autoencoder = models.Model(input_img, decoded)
autoencoder.compile(optimizer="adam", loss="binary_crossentropy")
autoencoder.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 32)	9,248
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 32)	9,248
conv2d_2 (Conv2D)	(None, 28, 28, 1)	289

```
Total params: 28,353 (110.75 KB)
Trainable params: 28,353 (110.75 KB)
Non-trainable params: 0 (0.00 B)
```

5. Entraînement du modèle

Nous entraînons l'autoencodeur à reconstruire les images originales à partir de leurs versions bruitées.

```
history = autoencoder.fit(
    x_train_noisy, x_train,
    epochs=20,
    batch_size=256,
    shuffle=True,
    validation_data=(x_test_noisy, x_test)
)
```

```
Epoch 17/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1008 - val_loss: 0.1000
Epoch 18/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1007 - val_loss: 0.0997
Epoch 19/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1005 - val_loss: 0.0996
Epoch 20/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.1003 - val_loss: 0.0996
```

Courbes d'apprentissage :

- La perte (loss) diminue régulièrement, indiquant que le modèle apprend à reconstruire les images.
- La perte de validation (val_loss) suit une tendance similaire, ce qui suggère une bonne généralisation.

6. Évaluation et visualisation des résultats

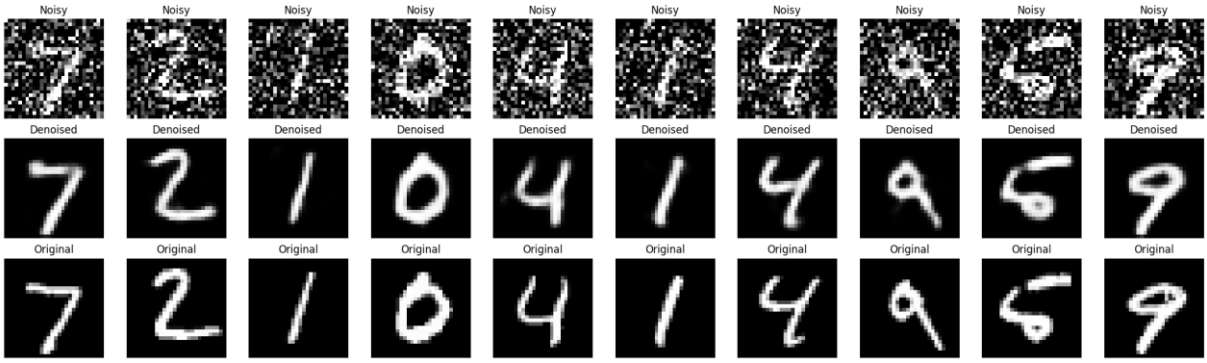
```
decoded_imgs = autoencoder.predict(x_test_noisy)

n = 10
plt.figure(figsize=(20, 6))
for i in range(n):
    # Noisy
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28), cmap="gray")
    plt.title("Noisy")
    plt.axis("off")

    # Denoised
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    plt.title("Denoised")
    plt.axis("off")

    # Original
    ax = plt.subplot(3, n, i + 1 + 2*n)
    plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
    plt.title("Original")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step



6.2Visualisation comparative

Nous affichons côte à côte :

- Les images bruitées
- Les images reconstruites (débruitées)
- Les images originales

Observations :

- L'autoencodeur réussit à supprimer une grande partie du bruit tout en préservant la structure des chiffres.
- Certains détails fins peuvent être légèrement altérés, mais la lisibilité est nettement améliorée.

7. Conclusion

Ce TP a permis de :

1. **Charger et prétraiter** le jeu de données MNIST.
2. **Générer des données bruitées** pour simuler un problème réaliste de débruitage.
3. **Construire un autoencodeur convolutif** avec un encodeur réducteur et un décodeur reconstructeur.
4. **Entraîner le modèle** pour qu'il apprenne à reconstruire les images originales à partir des versions bruitées.
5. **Visualiser et évaluer** les performances de débruitage