Solution Atelier

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

# Charger les données MNIST

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

# Prétraitement des données : redimensionner et normaliser les images

train\_images = train\_images.reshape(-1, 28, 28, 1).astype('float32') / 255.0

test\_images = test\_images.reshape(-1, 28, 28, 1).astype('float32') / 255.0

# Convertir les labels en catégories (one-hot encoding)

train\_labels = tf.keras.utils.to\_categorical(train\_labels, 10)

test\_labels = tf.keras.utils.to\_categorical(test\_labels, 10)

# Définir le modèle

model = models.Sequential()

# Première couche de convolution

model.add(layers.Conv2D(32, (5, 5), activation='sigmoid', input\_shape=(28, 28, 1)))

model.add(layers.MaxPooling2D((2, 2))) # Max pooling 2x2

# Seconde couche de convolution

model.add(layers.Conv2D(64, (5, 5), activation='sigmoid'))

model.add(layers.MaxPooling2D((2, 2))) # Max pooling 2x2

# Mettre à plat la sortie du second bloc convolutif

model.add(layers.Flatten())

# Couche complètement connectée de taille 100

model.add(layers.Dense(100, activation='sigmoid'))

# Couche de sortie de taille 10 (classification 0-9)

model.add(layers.Dense(10, activation='softmax'))

# Compiler le modèle

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

# Afficher le résumé de l'architecture du modèle

model.summary()

# Entraîner le modèle

history = model.fit(train\_images, train\_labels, epochs=10, batch\_size=64, validation\_data=(test\_images, test\_labels), verbose=2)

# Évaluer le modèle

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

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

# Visualiser les courbes d'entraînement

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()

# Afficher les courbes d'entraînement

plot\_training\_curves(history)

# Effectuer des prédictions sur un échantillon de test

predictions = model.predict(test\_images[:10])

# Afficher les prédictions

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].argmax() # True label (le label réel de l'image)

pred\_label = predictions[i].argmax() # Prédiction du modèle

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()