Exp 4

import tensorflow as tf

from tensorflow.keras import layers, models

import matplotlib.pyplot as plt

import time

import pandas as pd

import os

def load\_and\_preprocess\_data(dataset\_name):

    if dataset\_name == 'mnist':

        (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

        x\_train = x\_train[..., tf.newaxis] / 255.0

        x\_test = x\_test[..., tf.newaxis] / 255.0

        input\_shape = (28, 28, 1)

        num\_classes = 10

    elif dataset\_name == 'cifar10':

        (x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

        x\_train = x\_train / 255.0

        x\_test = x\_test / 255.0

        y\_train = y\_train.flatten()

        y\_test = y\_test.flatten()

        input\_shape = (32, 32, 3)

        num\_classes = 10

    return x\_train, y\_train, x\_test, y\_test, input\_shape, num\_classes

def build\_cnn\_model(input\_shape, num\_classes):

    model = models.Sequential([

        layers.Conv2D(32, (3, 3), activation='relu', input\_shape=input\_shape),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation='relu'),

        layers.MaxPooling2D((2, 2)),

        layers.Flatten(),

        layers.Dense(64, activation='relu'),

        layers.Dense(num\_classes, activation='softmax')

    ])

    return model

def train\_and\_save(dataset\_name):

    x\_train, y\_train, x\_test, y\_test, input\_shape, num\_classes = load\_and\_preprocess\_data(dataset\_name)

    model = build\_cnn\_model(input\_shape, num\_classes)

    model.compile(optimizer='adam',

                  loss='sparse\_categorical\_crossentropy',

                  metrics=['accuracy'])

    print(f"\n Training on {dataset\_name.upper()} dataset...\n")

    start\_time = time.time()

    history = model.fit(x\_train, y\_train, epochs=10,

                        validation\_data=(x\_test, y\_test),

                        verbose=2)

    end\_time = time.time()

    training\_time = end\_time - start\_time

    # Save model weights

    # Save model weights (filename must end in `.weights.h5`)

    weight\_file = f'{dataset\_name}.weights.h5'

    model.save\_weights(weight\_file)

    print(f" Model weights saved to: {weight\_file}")

    # Export metrics to CSV

    metrics\_df = pd.DataFrame(history.history)

    csv\_file = f'{dataset\_name}\_training\_log.csv'

    metrics\_df.to\_csv(csv\_file, index=False)

    print(f" Training history saved to: {csv\_file}")

    # Plot accuracy

    plt.figure()

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

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

    plt.title(f'{dataset\_name.upper()} Accuracy vs Epochs')

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

    plt.legend()

    plt.grid(True)

    plt.show()

    return training\_time

# Make output directory

os.makedirs("outputs", exist\_ok=True)

os.chdir("outputs")

# Run for MNIST and CIFAR-10

mnist\_time = train\_and\_save('mnist')

cifar\_time = train\_and\_save('cifar10')

# Compare training time

print("\n Training Time Comparison:")

print(f"MNIST Training Time   : {mnist\_time:.2f} seconds")

print(f"CIFAR-10 Training Time: {cifar\_time:.2f} seconds")







