# Import necessary libraries

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from torch.utils.tensorboard import SummaryWriter

import datetime

import numpy as np

# Load and preprocess the Iris dataset

data = load\_iris()

X, y = data.data, data.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Convert data to PyTorch tensors

X\_train = torch.tensor(X\_train, dtype=torch.float32)

X\_test = torch.tensor(X\_test, dtype=torch.float32)

y\_train = torch.tensor(y\_train, dtype=torch.long)

y\_test = torch.tensor(y\_test, dtype=torch.long)

# Define the model

class IrisModel(nn.Module):

def \_\_init\_\_(self):

super(IrisModel, self).\_\_init\_\_()

self.fc1 = nn.Linear(4, 64)

self.fc2 = nn.Linear(64, 32)

self.fc3 = nn.Linear(32, 16)

self.fc4 = nn.Linear(16, 3)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.relu(self.fc2(x))

x = torch.relu(self.fc3(x))

x = torch.softmax(self.fc4(x), dim=1)

return x

model = IrisModel()

# Define loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# Set up TensorBoard

log\_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

writer = SummaryWriter(log\_dir)

# Train the model

epochs = 100

batch\_size = 8

n\_batches = int(np.ceil(len(X\_train) / batch\_size))

for epoch in range(epochs):

model.train()

running\_loss = 0.0

for i in range(n\_batches):

start = i \* batch\_size

end = start + batch\_size

inputs = X\_train[start:end]

labels = y\_train[start:end]

# Zero the parameter gradients

optimizer.zero\_grad()

# Forward + backward + optimize

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

# Log training loss to TensorBoard

avg\_loss = running\_loss / n\_batches

writer.add\_scalar('Loss/train', avg\_loss, epoch)

# Evaluate the model on the test set

model.eval()

with torch.no\_grad():

outputs = model(X\_test)

loss = criterion(outputs, y\_test)

\_, predicted = torch.max(outputs, 1)

accuracy = (predicted == y\_test).sum().item() / len(y\_test)

writer.add\_scalar('Accuracy/test', accuracy, epoch)

print(f"Epoch [{epoch+1}/{epochs}], Loss: {avg\_loss:.4f}, Test Accuracy: {accuracy:.4f}")

print(f"Final Test Accuracy: {accuracy:.4f}")

# Close the TensorBoard writer

writer.close()

# Instructions to run TensorBoard in Google Colab

# Load the TensorBoard extension

%load\_ext tensorboard

# Launch TensorBoard

%tensorboard --logdir logs/fit

