# 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.data import DataLoader, TensorDataset

# Import pytorch\_lightning AFTER installing it if necessary

try:

import pytorch\_lightning as pl

except ImportError:

!pip install pytorch\_lightning

import pytorch\_lightning as pl

from pytorch\_lightning.loggers import TensorBoardLogger

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)

# Create DataLoaders

train\_dataset = TensorDataset(X\_train, y\_train)

test\_dataset = TensorDataset(X\_test, y\_test)

train\_loader = DataLoader(train\_dataset, batch\_size=8, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=8, shuffle=False)

# Define the model using PyTorch Lightning

class IrisModel(pl.LightningModule):

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)

self.criterion = nn.CrossEntropyLoss()

def forward(self, x):

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

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

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

x = self.fc4(x) # No softmax in forward, it's applied in loss calculation

return x

def training\_step(self, batch, batch\_idx):

inputs, labels = batch

outputs = self(inputs)

loss = self.criterion(outputs, labels)

self.log('train\_loss', loss)

return loss

def validation\_step(self, batch, batch\_idx):

inputs, labels = batch

outputs = self(inputs)

loss = self.criterion(outputs, labels)

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

accuracy = (predicted == labels).sum().item() / len(labels)

self.log('val\_loss', loss, prog\_bar=True)

self.log('val\_accuracy', accuracy, prog\_bar=True)

return loss

def configure\_optimizers(self):

return optim.Adam(self.parameters(), lr=0.001)

# Set up TensorBoard logger

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

logger = TensorBoardLogger("logs", name="iris\_model")

# Train the model using PyTorch Lightning

model = IrisModel()

trainer = pl.Trainer(max\_epochs=100, logger=logger)

trainer.fit(model, train\_loader, test\_loader)

# Instructions to run TensorBoard in Google Colab

# Load the TensorBoard extension

%load\_ext tensorboard

# Launch TensorBoard

%tensorboard --logdir logs

