Create and Train a Simple Neural Network using PyTorch

Table of Contents

Description

In this lab, you will learn how to create and train a simple neural network using PyTorch, a popular deep learning framework. You will work with the MNIST dataset, a collection of handwritten digits, to build a model that can recognize digits from 0 to 9.

Problem Statement

The goal is to build a neural network that can accurately classify handwritten digits from the MNIST dataset. The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits, labeled from 0 to 9. You will preprocess the data, build a neural network model, train it, evaluate its performance, and visualize the training process.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-04) is required before proceeding with Lab Guide-05.

Software Requirements

- **Python**: Version 3.11.9
- **PyTorch**: Install using `pip install torch torchvision`.
- **Tensorflow**: Install using `pip install tensorflow`.
- **Visual Studio Code (VSCode)**: A lightweight code editor that provides powerful features for Python development, including extensions for linting, debugging, and version control.
- **NumPy**: Install using `pip install numpy`.
- **Matplotlib**: Install using `pip install matplotlib`.

Hardware Requirements

- Minimum 4GB RAM.
- At least 1GB of free disk space.
- A GPU (optional, but recommended for faster training).

Setup Instructions

Setting Up a Python Environment

1. Install Python: You can download and install Python 3.11.9 from the official Python website:

- Visit the official Python website.
- Locate a reliable version of Python 3, **"Download Python 3.11.9"**.
- Choose the correct link for your device from the options provided: either Windows installer (64-bit) or Windows installer (32-bit) and proceed to download the executable file.

---

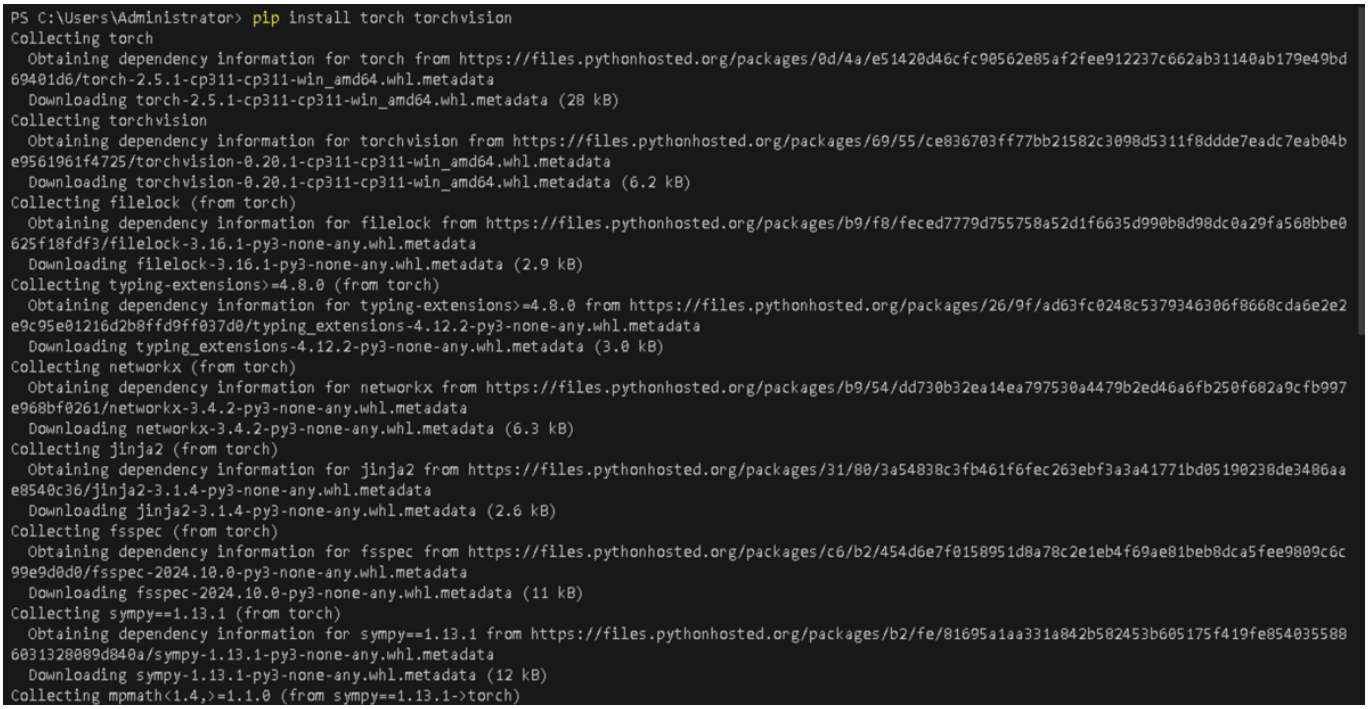
2. Create a Virtual Environment:

```
python -m venv myenv
source myenv/bin/activate  # On Windows use `myenv\Scripts\activate`
```

---

3. Install Required Packages:

```
pip install torch torchvision
```



```
PS C:\Users\Administrator> pip install torch torchvision
Collecting torch
  Obtaining dependency information for torch from https://files.pythonhosted.org/packages/0d/4a/e51420d46cfc90562e85af2fee912237c662ab31140ab179e49bd69401d6/torch-2.5.1-cp311-cp311-win_amd64.whl.metadata
  Downloading torch-2.5.1-cp311-cp311-win_amd64.whl.metadata (28 kB)
Collecting torchvision
  Obtaining dependency information for torchvision from https://files.pythonhosted.org/packages/69/55/ce836703ff77bb21582c3098d5311f8ddde7eadc7eab04be9561961f4725/torchvision-0.20.1-cp311-cp311-win_amd64.whl.metadata
  Downloading torchvision-0.20.1-cp311-cp311-win_amd64.whl.metadata (6.2 kB)
Collecting filelock (from torch)
  Obtaining dependency information for filelock from https://files.pythonhosted.org/packages/b9/f8/feced7779d755758a52d1f6635d990b8d98dc0a29fa568bbe0625f18fdf3/filelock-3.16.1-py3-none-any.whl.metadata
  Downloading filelock-3.16.1-py3-none-any.whl.metadata (2.9 kB)
Collecting typing-extensions>=4.8.0 (from torch)
  Obtaining dependency information for typing-extensions>=4.8.0 from https://files.pythonhosted.org/packages/26/9f/ad63fc0248c5379346306f8668cda6e2e2e9c95e01216d2b8ffd9ff037d0/typing_extensions-4.12.2-py3-none-any.whl.metadata
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 kB)
Collecting networkx (from torch)
  Obtaining dependency information for networkx from https://files.pythonhosted.org/packages/b9/54/dd730b32ea14ea797530a4479b2ed46a6fb250f682a9cfb997e968bf0261/networkx-3.4.2-py3-none-any.whl.metadata
  Downloading networkx-3.4.2-py3-none-any.whl.metadata (6.3 kB)
Collecting jinja2 (from torch)
  Obtaining dependency information for jinja2 from https://files.pythonhosted.org/packages/31/80/3a54838c3fb461f6fec263ebf3a3a41771bd05190238de3486aae8540c36/jinja2-3.1.4-py3-none-any.whl.metadata
  Downloading jinja2-3.1.4-py3-none-any.whl.metadata (2.6 kB)
Collecting fsspec (from torch)
  Obtaining dependency information for fsspec from https://files.pythonhosted.org/packages/c6/b2/454d6e7f0158951d8a78c2e1eb4f69ae81beb8dca5fee9809c6c99e9d0d0/fsspec-2024.10.0-py3-none-any.whl.metadata
  Downloading fsspec-2024.10.0-py3-none-any.whl.metadata (11 kB)
Collecting sympy==1.13.1 (from torch)
  Obtaining dependency information for sympy==1.13.1 from https://files.pythonhosted.org/packages/b2/fe/81695a1aa331a842b582453b605175f419fe8540355886031328089d840a/sympy-1.13.1-py3-none-any.whl.metadata
  Downloading sympy-1.13.1-py3-none-any.whl.metadata (12 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy==1.13.1->torch)
```

```
pip install matplotlib, numpy
```

```
Collecting kiwisolver>=1.3.1 (from matplotlib)
  Downloading kiwisolver-1.4.7-cp313-cp313-win_amd64.whl.metadata (6.4 kB)
Collecting packaging>=20.0 (from matplotlib)
  Downloading packaging-24.1-py3-none-any.whl.metadata (3.2 kB)
Collecting pillow>=8 (from matplotlib)
  Downloading pillow-10.4.0-cp313-cp313-win_amd64.whl.metadata (9.3 kB)
Collecting pyparsing>=2.3.1 (from matplotlib)
  Downloading pyparsing-3.1.4-py3-none-any.whl.metadata (5.1 kB)
Requirement already satisfied: scipy>=1.6.0 in c:\users\thinkpad\appdata\roaming\python\python313\site-packages (from scikit-learn) (1.
14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\thinkpad\appdata\roaming\python\python313\site-packages (from scikit-learn) (1
.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\thinkpad\appdata\roaming\python\python313\site-packages (from scikit-le
arn) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\users\thinkpad\appdata\roaming\python\python313\site-packages (from python-dateutil>=2.8.
2->pandas) (1.16.0)
Downloading matplotlib-3.9.2-cp313-cp313-win_amd64.whl (7.8 MB)
   ---------------------------------------- 7.8/7.8 MB 5.6 MB/s eta 0:00:00
Downloading seaborn-0.13.2-py3-none-any.whl (294 kB)
Downloading contourpy-1.3.0-cp313-cp313-win_amd64.whl (218 kB)
Using cached cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.54.1-cp313-cp313-win_amd64.whl (2.2 MB)
   ---------------------------------------- 2.2/2.2 MB 5.8 MB/s eta 0:00:00
Downloading kiwisolver-1.4.7-cp313-cp313-win_amd64.whl (55 kB)
Downloading packaging-24.1-py3-none-any.whl (53 kB)
Downloading pillow-10.4.0-cp313-cp313-win_amd64.whl (2.6 MB)
   ---------------------------------------- 2.6/2.6 MB 5.5 MB/s eta 0:00:00
Downloading pyparsing-3.1.4-py3-none-any.whl (104 kB)
Installing collected packages: pyparsing, pillow, packaging, kiwisolver, fonttools, cycler, contourpy, matplotlib, seaborn
  WARNING: The scripts fonttools.exe, pyftmerge.exe, pyftsubset.exe and ttx.exe are installed in 'C:\Users\Thinkpad\AppData\Roaming\Pyt
hon\Python313\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed contourpy-1.3.0 cycler-0.12.1 fonttools-4.54.1 kiwisolver-1.4.7 matplotlib-3.9.2 packaging-24.1 pillow-10.4.0 py
parsing-3.1.4 seaborn-0.13.2
```

Key Concepts

Neural Networks

Neural networks are computational models designed to simulate the way the human brain processes information. These networks consist of layers of nodes, also called neurons, which are interconnected. Each connection has a weight that adjusts as learning proceeds, guiding the strength of the signal from one neuron to another. The primary components of neural networks are:

1. **Input Layer**: The layer that receives the initial data.
2. **Hidden Layers**: Layers between the input and output layers where computation is performed through weighted connections. These layers can be numerous and form the depth of the network.
3. **Output Layer**: The layer that produces the final result or prediction.

Neural networks are used for various tasks such as image and speech recognition, natural language processing, and game playing. They excel at identifying patterns and making predictions based on data.

TensorFlow

TensorFlow is an open-source deep learning framework developed by the Google Brain team. It provides a comprehensive ecosystem for building and deploying machine learning models. TensorFlow is widely used in industry and research for tasks such as image and speech recognition, natural language processing, and reinforcement learning.

PyTorch

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab. It is known for its flexibility and ease of use, particularly due to its dynamic computation graph. PyTorch is particularly favored in the research community due to its flexibility and ease of use, making it suitable for prototyping and experimenting with new ideas.

Steps to Create and Train a Neural Network

Loading and Preparing Data

1. Import Libraries:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt
```

---

2. Load and Normalize Data:

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # Normalize to [-1, 1]
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False, download=True,
transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=64,
shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=64,
shuffle=False)
```

Output

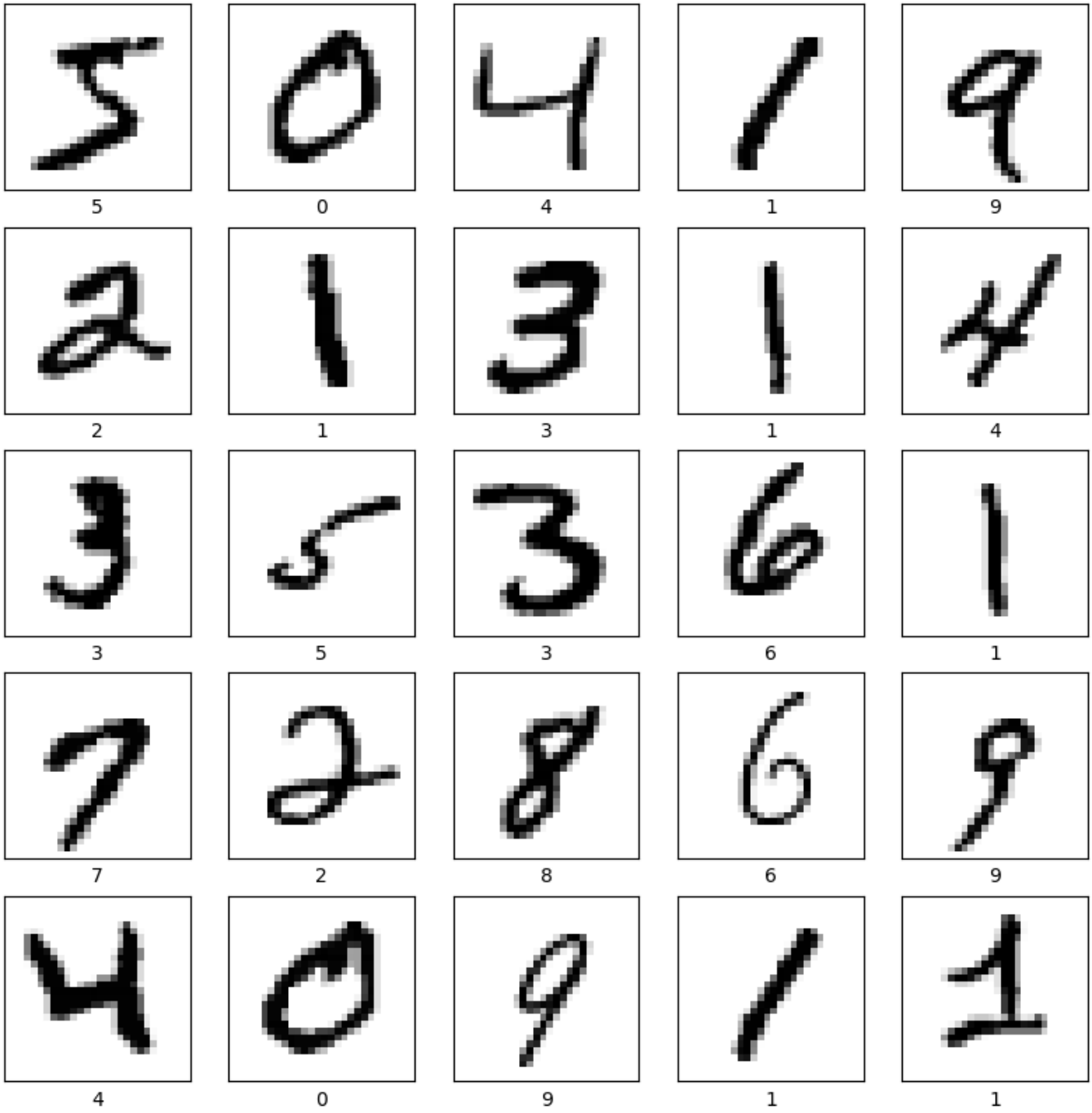
```
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 14287318.56it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 483639.34it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:02<00:00, 657300.97it/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
```

---

3. Visualize Sample Data:

```
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    image, label = train_dataset[i]
    plt.imshow(image.squeeze(), cmap=plt.cm.binary)
    plt.xlabel(label)
plt.show()
```

Output



Building the Neural Network

Define the Model:

```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.2)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
```

```
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model = SimpleNN()
```

Compiling the Model

Define the Loss Function and Optimizer:

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())
```

Training the Model

Train the Model:

```
epochs = 5
train_losses, train_accuracies = [], []

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_loss = running_loss / len(train_loader)
    accuracy = correct / total
    train_losses.append(epoch_loss)
    train_accuracies.append(accuracy)
    print(f'Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}, Accuracy:
{accuracy:.4f}')
```

Output

```
100.0%
Extracting ./data\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./data\MNIST\raw

Epoch 1/5, Loss: 0.4232, Accuracy:0.8745
Epoch 2/5, Loss: 0.2329, Accuracy:0.9313
Epoch 3/5, Loss: 0.1868, Accuracy:0.9448
Epoch 4/5, Loss: 0.1611, Accuracy:0.9509
Epoch 5/5, Loss: 0.1479, Accuracy:0.9549
```

Evaluating the Model

Evaluate the Model:

```
model.eval()
test_loss = 0.0
correct = 0
total = 0

with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        loss = criterion(outputs, labels)
        test_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

test_loss /= len(test_loader)
test_accuracy = correct / total
print(f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}')
```

Output

```
PS C:\Users\Administrator\Desktop\AIML> python neural_network.py
Test Loss: 0.1082, Test Accuracy: 0.9679
```

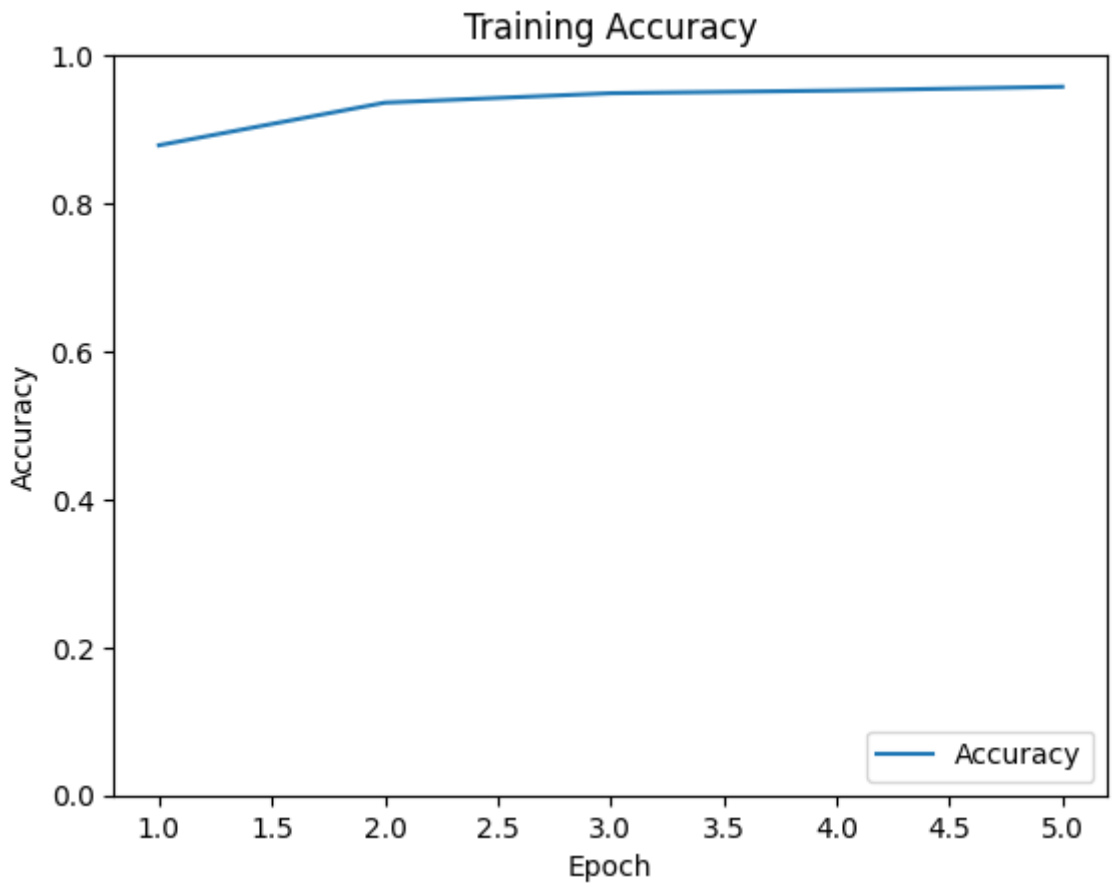
Visualizing Training Results

Plot Training Accuracy:

```
plt.plot(range(1, epochs + 1), train_accuracies, label='Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
```

```
plt.title('Training Accuracy')
plt.show()
```

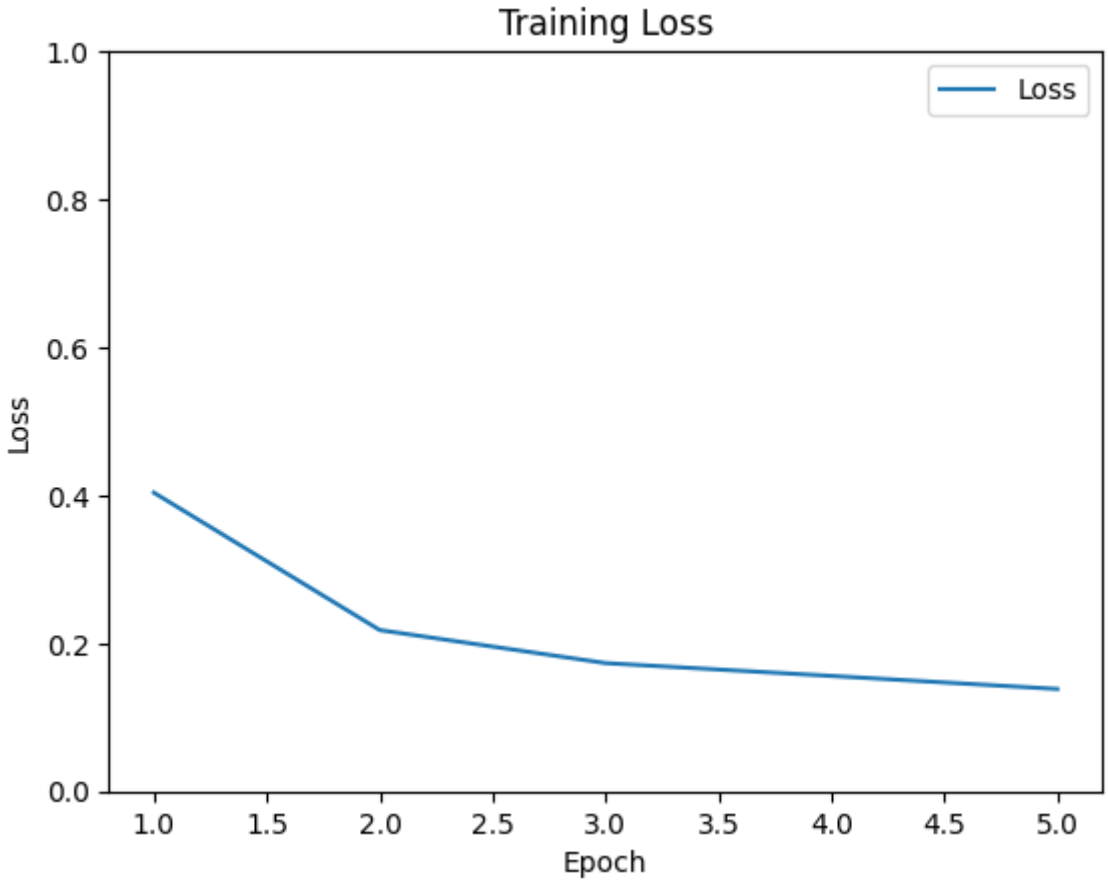
Output



Plot Training Loss:

```
plt.plot(range(1, epochs + 1), train_losses, label='Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.legend(loc='upper right')
plt.title('Training Loss')
plt.show()
```

Output



Optional TensorFlow Implementation

If you prefer to use TensorFlow instead of PyTorch, you can follow similar steps to build and train a neural network for the MNIST dataset. Here's a quick guide to implementing the same neural network using TensorFlow:

Installing TensorFlow via pip

Install TensorFlow:

```
pip install tensorflow
```

Complete Code

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout
import matplotlib.pyplot as plt

# Load and normalize the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
# Visualize sample data
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train[i], cmap=plt.cm.binary)
    plt.xlabel(y_train[i])
plt.show()

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Test accuracy: {test_acc}')

# Visualize training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

References

- Introduction to PyTorch
- Tensorflow vs. PyTorch