```
!pip install kaggle
!mkdir ~/.kaggle
```

```
Requirement already satisfied: kaggle in
/usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in
/usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2024.8.30)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from kaggle) (4.66.6)
Requirement already satisfied: python-slugify in
/usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in
/usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: webencodings in
/usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in
/usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle)
(1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle)
(3.4.0)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

```
!kaggle datasets download -d meowmeowmeowmeowmeow/gtsrb-german-
traffic-sign
!unzip -q gtsrb-german-traffic-sign.zip -d /content/
```

```
Dataset URL:
https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-
traffic-sign
License(s): CC0-1.0
Downloading gtsrb-german-traffic-sign.zip to /content
100% 609M/612M [00:06<00:00, 83.0MB/s]
100% 612M/612M [00:06<00:00, 98.4MB/s]
```

```
import os
import pandas as pd
from PIL import Image
from torch.utils.data import Dataset
```

```
class GTSRB_Dataset(Dataset):
    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (str): Path to the CSV file with annotations.
            root_dir (str): Directory with all the images.
            transform (callable, optional): Optional transform to be
applied on a sample.
        """
        self.data_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

    def __len__(self):
        return len(self.data_frame)

    def __getitem__(self, idx):
        # Fetch the image path and label
        img_path = os.path.join(self.root_dir,
self.data_frame.iloc[idx, 7])
        image = Image.open(img_path)
        label = self.data_frame.iloc[idx, 6]

        # Apply transformations if provided
        if self.transform:
            image = self.transform(image)

        return image, label

import torch
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

# Define the transformation
transform = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor()
])

# Define paths
train_csv = '/content/Train.csv'
test_csv = '/content/Test.csv'
root_dir = '/content/'

# Initialize datasets
train_dataset = GTSRB_Dataset(csv_file=train_csv, root_dir=root_dir,
transform=transform)
test_dataset = GTSRB_Dataset(csv_file=test_csv, root_dir=root_dir,
transform=transform)
```

```
# Create DataLoaders
train_loader = DataLoader(dataset=train_dataset, batch_size=64,
shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=10,
shuffle=False)

import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# Check device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using {device} device')

# Define the MLP Model
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(32*32*3, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 128),
            nn.ReLU(),
            nn.Linear(128, 43)
        )

    def forward(self, x):
        x = x.view(x.size(0), -1)  # Flatten the input
        return self.layers(x)

# Initialize the model, loss, and optimizer
model = MLP().to(device)
print(model)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training function
def train_model(model, train_loader, criterion, optimizer,
num_epochs=5):
    loss_values = []

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0.0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)  #
```

```
Move data to device

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward pass
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        loss_values.append(avg_loss)
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")

    # Plot loss
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, num_epochs + 1), loss_values, label='Training
Loss', marker='o')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training Loss Over Epochs')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
Using cpu device
MLP(
  (layers): Sequential(
    (0): Linear(in_features=3072, out_features=1024, bias=True)
    (1): ReLU()
    (2): Linear(in_features=1024, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=128, bias=True)
    (5): ReLU()
    (6): Linear(in_features=128, out_features=43, bias=True)
  )
)
```

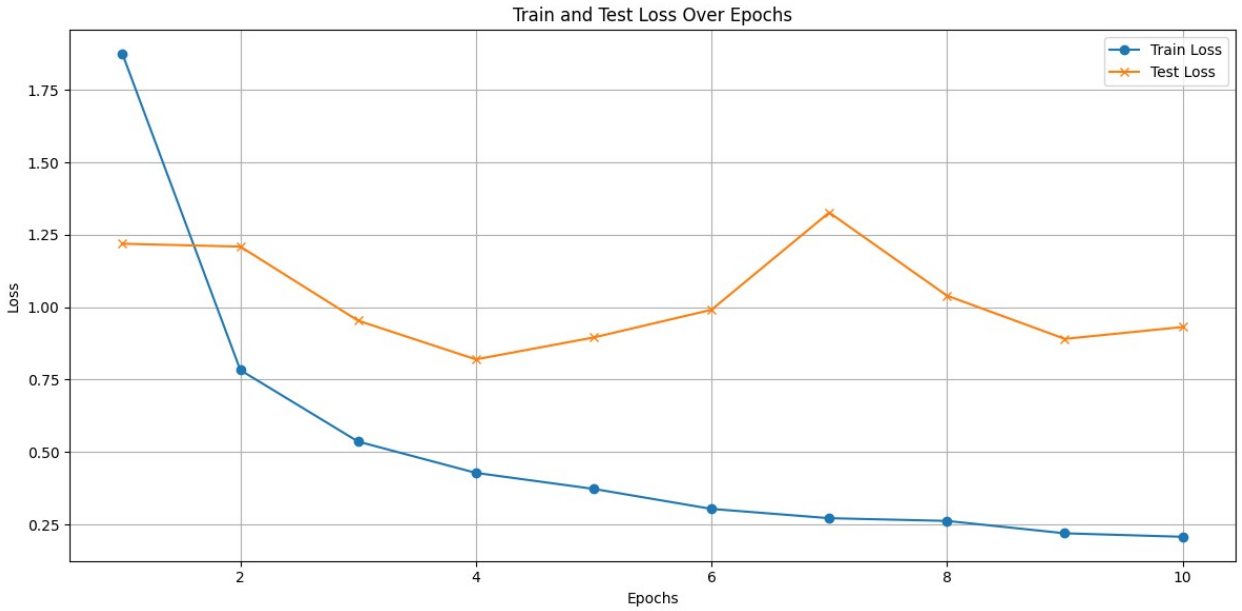
```
import matplotlib.pyplot as plt
import torch

def train_and_evaluate(model, train_loader, test_loader, criterion,
optimizer, epochs):
    train_losses, test_losses, accuracies = [], [], []

    for epoch in range(epochs):
```

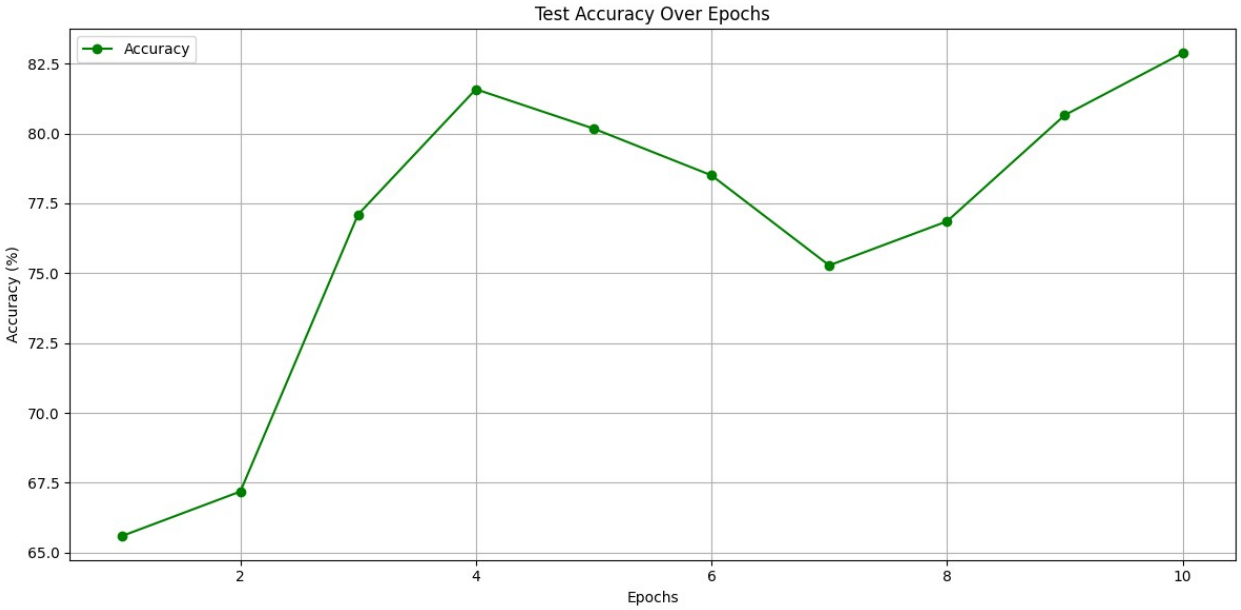
```
        # Training phase
        model.train()
        running_train_loss = 0.0

        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)  #
Move data to device

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_train_loss += loss.item()

        avg_train_loss = running_train_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        # Evaluation phase
        model.eval()
        running_test_loss, correct, total = 0.0, 0, 0

        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                running_test_loss += loss.item()

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

        avg_test_loss = running_test_loss / len(test_loader)
        accuracy = 100 * correct / total
        test_losses.append(avg_test_loss)
        accuracies.append(accuracy)

        print(f"Epoch [{epoch + 1}/{epochs}], Train Loss:
{avg_train_loss:.4f}, Test Loss: {avg_test_loss:.4f}, Accuracy:
{accuracy:.2f}%")

    # Plot Training and Test Loss
    plt.figure(figsize=(12, 6))
    plt.plot(range(1, epochs + 1), train_losses, label='Train Loss',
marker='o')
    plt.plot(range(1, epochs + 1), test_losses, label='Test Loss',
marker='x')
    plt.xlabel('Epochs')
```

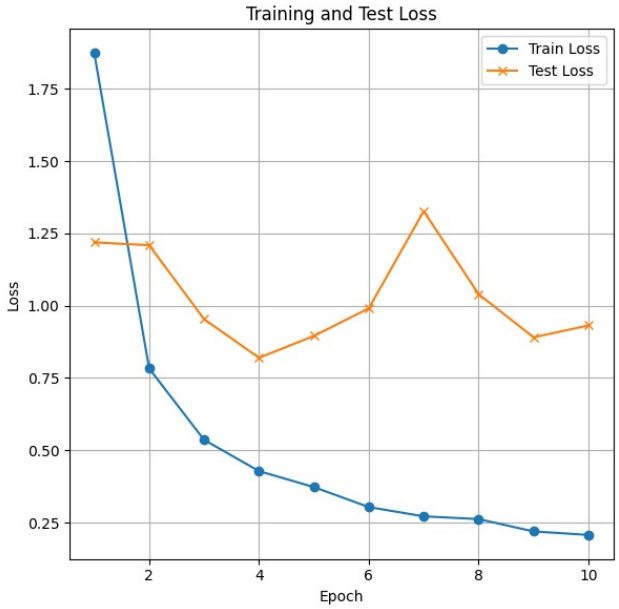
```
    plt.ylabel('Loss')
    plt.title('Train and Test Loss Over Epochs')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    # Plot Accuracy
    plt.figure(figsize=(12, 6))
    plt.plot(range(1, epochs + 1), accuracies, label='Accuracy',
marker='o', color='g')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy (%)')
    plt.title('Test Accuracy Over Epochs')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    return train_losses, test_losses, accuracies

# Run training and evaluation
epochs = 10
train_losses, test_losses, accuracies = train_and_evaluate(model,
train_loader, test_loader, criterion, optimizer, epochs=epochs)

# Plotting Losses and Accuracy
plt.figure(figsize=(12, 6))

# Train and Test Loss Plot
plt.subplot(1, 2, 1)
plt.plot(range(1, epochs + 1), train_losses, label='Train Loss',
marker='o')
plt.plot(range(1, epochs + 1), test_losses, label='Test Loss',
marker='x')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()
plt.grid(True)

# Accuracy Plot
plt.subplot(1, 2, 2)
plt.plot(range(1, epochs + 1), accuracies, label='Test Accuracy',
marker='o', color='green')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Test Accuracy')
plt.legend()
plt.grid(True)
```

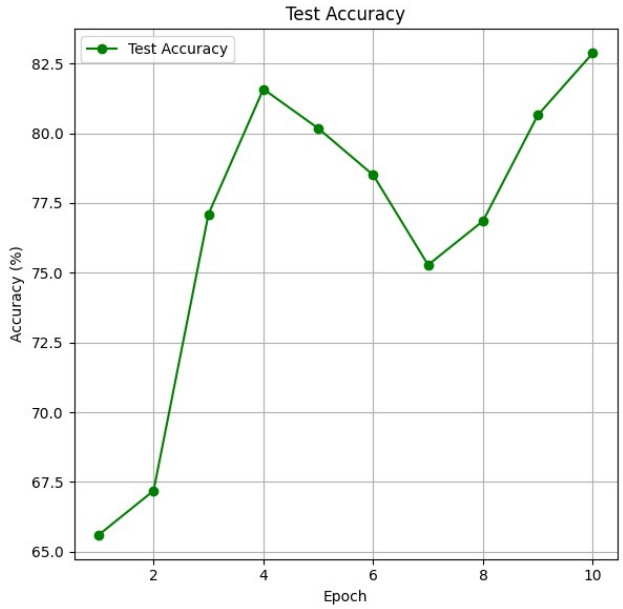
```
# Ensure everything fits well
plt.tight_layout()
plt.show()
```

```
Epoch [1/10], Train Loss: 1.8734, Test Loss: 1.2191, Accuracy: 65.61%
Epoch [2/10], Train Loss: 0.7829, Test Loss: 1.2092, Accuracy: 67.19%
Epoch [3/10], Train Loss: 0.5362, Test Loss: 0.9536, Accuracy: 77.10%
Epoch [4/10], Train Loss: 0.4282, Test Loss: 0.8200, Accuracy: 81.58%
Epoch [5/10], Train Loss: 0.3729, Test Loss: 0.8955, Accuracy: 80.17%
Epoch [6/10], Train Loss: 0.3039, Test Loss: 0.9910, Accuracy: 78.50%
Epoch [7/10], Train Loss: 0.2718, Test Loss: 1.3268, Accuracy: 75.28%
Epoch [8/10], Train Loss: 0.2624, Test Loss: 1.0399, Accuracy: 76.86%
Epoch [9/10], Train Loss: 0.2196, Test Loss: 0.8907, Accuracy: 80.66%
Epoch [10/10], Train Loss: 0.2076, Test Loss: 0.9316, Accuracy: 82.87%
```




Test Accuracy Over Epochs
Accuracy
Accuracy (%)
Epochs


Training and Test Loss
Train Loss
Test Loss
Loss
Epoch


Test Accuracy
Test Accuracy
Accuracy (%)
Epoch