Polat_Değirmenci 2023688022

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms

# Hiperparametreler
batch_size = 64
learning_rate = 0.001
num_epochs = 5

# Veri Ön İşleme ve Yükleme
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

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

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

# CNN Model Tanımı
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.oznitelikler = nn.Sequential(
            nn.Conv2d(1, 20, kernel_size=5, stride=1, padding=0),  # 28x28 -> 24x24, 20 filtre
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),                 # 24x24 -> 12x12
            nn.Conv2d(20, 50, kernel_size=5, stride=1, padding=0), # 12x12 -> 8x8, 50 filtre
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)                  # 8x8 -> 4x4
        )
        self.siniflandirma = nn.Sequential(
            nn.Flatten(),
            nn.Linear(50 * 4 * 4, 500),  # 4x4x50 -> 500 birim
            nn.ReLU(),
            nn.Linear(500, 10)           # 500 -> 10 birim (10 sınıf)
        )

    def forward(self, x):
        x = self.oznitelikler(x)
        x = self.siniflandirma(x)
        return x

# Model, Kayıp Fonksiyonu, Optimizasyon
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Eğitim Döngüsü
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {total_loss / len(train_loader):.4f}")

# Test Döngüsü
model.eval()
dogru_sayisi = 0
with torch.no_grad():
    for data, target in test_loader:
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        dogru_sayisi += pred.eq(target.view_as(pred)).sum().item()

dogruluk_orani = dogru_sayisi / len(test_loader.dataset)
print(f"Test Doğruluğu: {dogruluk_orani * 100:.2f}%")
```