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

# Hiperparametreler
veri_yigin_boyu = 128  # Daha büyük batch size
ogrenme_orani = 0.0005  # Daha küçük öğrenme oranı
epoch_sayisi = 10  # Daha fazla epoch

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

egitim_veriseti = datasets.MNIST(root='./data', train=True, transform=veri_donusumu
test_veriseti = datasets.MNIST(root='./data', train=False, transform=veri_donusumu,

egitim_yukleyici = torch.utils.data.DataLoader(dataset=egitim_veriseti, batch_size=
test_yukleyici = torch.utils.data.DataLoader(dataset=test_veriseti, batch_size=veri

# Yeni CNN Modeli
class GuncellenmisCNN(nn.Module):
    def __init__(self):
        super(GuncellenmisCNN, self).__init__()
        self.oznitelikler = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=5, stride=1, padding=0),  # Daha fazla fil
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=0), # Daha fazla fil
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.siniflandirma = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 4 * 4, 512),  # Daha büyük tam bağlantılı katman (512)
            nn.ReLU(),
            nn.Dropout(p=0.4),          # Dropout eklendi
            nn.Linear(512, 256),        # Ekstra katman (256)
            nn.ReLU(),
            nn.Linear(256, 10)          # Çıkış katmanı
        )

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

# Model, Kayıp Fonksiyonu, Optimizasyon
model = GuncellenmisCNN()
kayip_fonksiyonu = nn.CrossEntropyLoss()
```

```
    optimizasyon = optim.Adam(model.parameters(), lr=ogrenme_orani)

    # Eğitim Döngüsü
    for epoch in range(epoch_sayisi):
        model.train()
        toplam_kayip = 0
        for veri_indeksi, (veri, hedef) in enumerate(egitim_yukleyici):
            optimizasyon.zero_grad()
            cikti = model(veri)
            kayip = kayip_fonksiyonu(cikti, hedef)
            kayip.backward()
            optimizasyon.step()
            toplam_kayip += kayip.item()

        print(f"Epoch {epoch+1}/{epoch_sayisi}, Loss: {toplam_kayip / len(egitim_yukley

    # Test Döngüsü
    model.eval()
    dogru_sayisi = 0
    with torch.no_grad():
        for veri, hedef in test_yukleyici:
            cikti = model(veri)
            tahmin = cikti.argmax(dim=1, keepdim=True)
            dogru_sayisi += tahmin.eq(hedef.view_as(tahmin)).sum().item()

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