```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix,
ConfusionMatrixDisplay
import csv

# Cihaz ayarı: GPU varsa onu kullan, yoksa CPU kullan.
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Cihaz: {device}")

# Özel bir veri seti sınıfı tanımlıyoruz.
class CustomMNISTDataset(Dataset):
    """MNIST veri setini özelleştirilmiş şekilde yükler."""
    def __init__(self, csv_file, transform=None, is_test=False):
        # Veri setini CSV dosyasından yükler.
        self.data_frame = pd.read_csv(csv_file)
        self.transform = transform  # Görüntü dönüşümleri
        self.is_test = is_test  # Test veri seti mi?

    def __len__(self):
        # Veri setindeki örneklerin sayısını döndürür.
        return len(self.data_frame)

    def __getitem__(self, index):
        # Bir örneği alır ve gerekli işlemleri yapar.
        item = self.data_frame.iloc[index]
        if self.is_test:
            # Test verisi için yalnızca görüntü alıyoruz.
            image = item.values.reshape(28, 28).astype(np.uint8)
            label = None
        else:
            # Eğitim verisi için hem görüntü hem etiket alıyoruz.
            image = item[1:].values.reshape(28, 28).astype(np.uint8)
            label = item.iloc[0]

        # NumPy görüntüsünü PIL formatına çeviriyoruz.
        image = transforms.ToPILImage()(image)

        # Dönüşümler varsa uyguluyoruz.
        if self.transform is not None:
            image = self.transform(image)

        # Test verisi ise sadece görüntüyü döndür, yoksa görüntü ve etiketi döndür.
        if self.is_test:
            return image
        else:
```

```
            return image, label

# Veri artırma (augmentation) için dönüşümler tanımlanır.
transform = transforms.Compose([
    transforms.RandomRotation(30),  # Rastgele döndürme
    transforms.RandomHorizontalFlip(p=0.5),  # Rastgele yatay çevirme
    transforms.RandomAffine(10, translate=(0.1, 0.1)),  # Çeviri ve döndürme
    transforms.ToTensor(),  # Tensor formatına çevir
    transforms.Normalize((0.5,), (0.5,))  # Normalizasyon
])

# Eğitim ve test veri setlerini oluşturuyoruz.
train_dataset = CustomMNISTDataset(
    csv_file='/kaggle/input/digit-recognizer/train.csv',
    transform=transform,
    is_test=False
)
test_dataset = CustomMNISTDataset(
    csv_file='/kaggle/input/digit-recognizer/test.csv',
    transform=transform,
    is_test=True
)

# Veri seti boyutlarını yazdır.
print(f"Eğitim Veri Seti Boyutu: {len(train_dataset)}, Test Veri Seti Boyutu: {len(test_dataset)}")

# Veri yükleyiciler (Dataloader) tanımlanır.
batch_size = 64
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True,
num_workers=2)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False,
num_workers=2)

# Basit bir Convolutional Neural Network (CNN) modeli tanımlıyoruz.
class SimpleCNN(nn.Module):
    """Basit bir CNN modeli"""
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)  # İlk konvolüsyon katmanı
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)  # İkinci konvolüsyon katmanı
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)  # Üçüncü konvolüsyon katmanı
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)  # Havuzlama katmanı
        self.dropout = nn.Dropout(0.3)  # Aşırı öğrenmeyi engellemek için dropout
        self.fc1 = nn.Linear(128 * 3 * 3, 256)  # Tam bağlı katman 1
        self.fc2 = nn.Linear(256, 10)  # Tam bağlı katman 2 (Çıkış katmanı)

    def forward(self, x):
        # İleri yayılım işlemi
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = self.pool(torch.relu(self.conv3(x)))
        x = x.view(-1, 128 * 3 * 3)  # Veriyi tam bağlı katmanlar için düzleştir
```

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

# Modeli tanımlıyoruz ve cihazımıza (CPU/GPU) yüklüyoruz.
model = SimpleCNN().to(device)

# Kayıp fonksiyonu (Loss) ve optimizasyon algoritması tanımlanır.
criterion = nn.CrossEntropyLoss()  # Çapraz entropi kaybı
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam optimizasyon algoritması

# Eğitim raporu için dosya adı
training_report_filename = "egitim_raporu.csv"

# Eğitim döngüsü
num_epochs = 20
train_losses, train_accuracies = [], []

# Eğitim raporu dosyasına başlık ekliyoruz.
with open(training_report_filename, "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["Epoch", "Kayıp", "Doğruluk (%)"])  # Başlık

# Eğitim işlemi
for epoch in range(num_epochs):
    model.train()
    running_loss, correct, total = 0.0, 0, 0

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

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

    epoch_loss = running_loss / len(train_loader)
    epoch_accuracy = 100 * correct / total
    train_losses.append(epoch_loss)
    train_accuracies.append(epoch_accuracy)
    print(f"Epoch {epoch + 1}/{num_epochs}, Kayıp: {epoch_loss:.4f}, Doğruluk:
{epoch_accuracy:.2f}%")

    # Her epoch sonucunu CSV dosyasına yaz
    with open(training_report_filename, "a", newline="") as file:
        writer = csv.writer(file)
        writer.writerow([epoch + 1, epoch_loss, epoch_accuracy])
```

```
# Eğitim performansı görselleştirilir.
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label="Kayıp")
plt.title("Eğitim Kaybı")
plt.xlabel("Epoch")
plt.ylabel("Kayıp")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label="Doğruluk")
plt.title("Eğitim Doğruluğu")
plt.xlabel("Epoch")
plt.ylabel("Doğruluk (%)")
plt.legend()
plt.show()

# Performans metrikleri ve değerlendirme
model.eval()
all_labels, all_predictions = [], []

with torch.no_grad():
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        all_labels.extend(labels.cpu().numpy())
        all_predictions.extend(predicted.cpu().numpy())

# Performans ölçümleri
precision = precision_score(all_labels, all_predictions, average="macro")
recall = recall_score(all_labels, all_predictions, average="macro")
f1 = f1_score(all_labels, all_predictions, average="macro")
print(f"Kesinlik (Precision): {precision:.4f}, Duyarlılık (Recall): {recall:.4f}, F1 Skoru: {f1:.4f}")

# Eğitim raporuna performans metriklerini ekliyoruz.
with open(training_report_filename, "a", newline="") as file:
    writer = csv.writer(file)
    writer.writerow([])
    writer.writerow(["Performans Metrikleri"])
    writer.writerow(["Kesinlik (Precision)", precision])
    writer.writerow(["Duyarlılık (Recall)", recall])
    writer.writerow(["F1 Skoru", f1])

# Karmaşıklık Matrisi (Confusion Matrix) görselleştirme
conf_matrix = confusion_matrix(all_labels, all_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=range(10))
disp.plot(cmap=plt.cm.Blues)
plt.title("Karmaşıklık Matrisi")
plt.show()
```

```
# Test tahminleri ve sonuçların kaydedilmesi
test_predictions = []

with torch.no_grad():
    for inputs in test_loader:
        inputs = inputs.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        test_predictions.extend(predicted.cpu().tolist())

# Test sonuçlarını CSV formatında kaydet
submission = pd.DataFrame({
    "ImageId": range(1, len(test_predictions) + 1),
    "Label": test_predictions
})
submission.to_csv('submission.csv', index=False)
```