`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-21

Карагодин Д. Л.

Проверил:

Крощенко А. А.

Брест 2024

`

Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

Ход работы:

Вариант 1

№ варианта	Выборка	Размер исходного изображения	Оптимизатор
1	MNIST	28X28	SGD

Код программы:

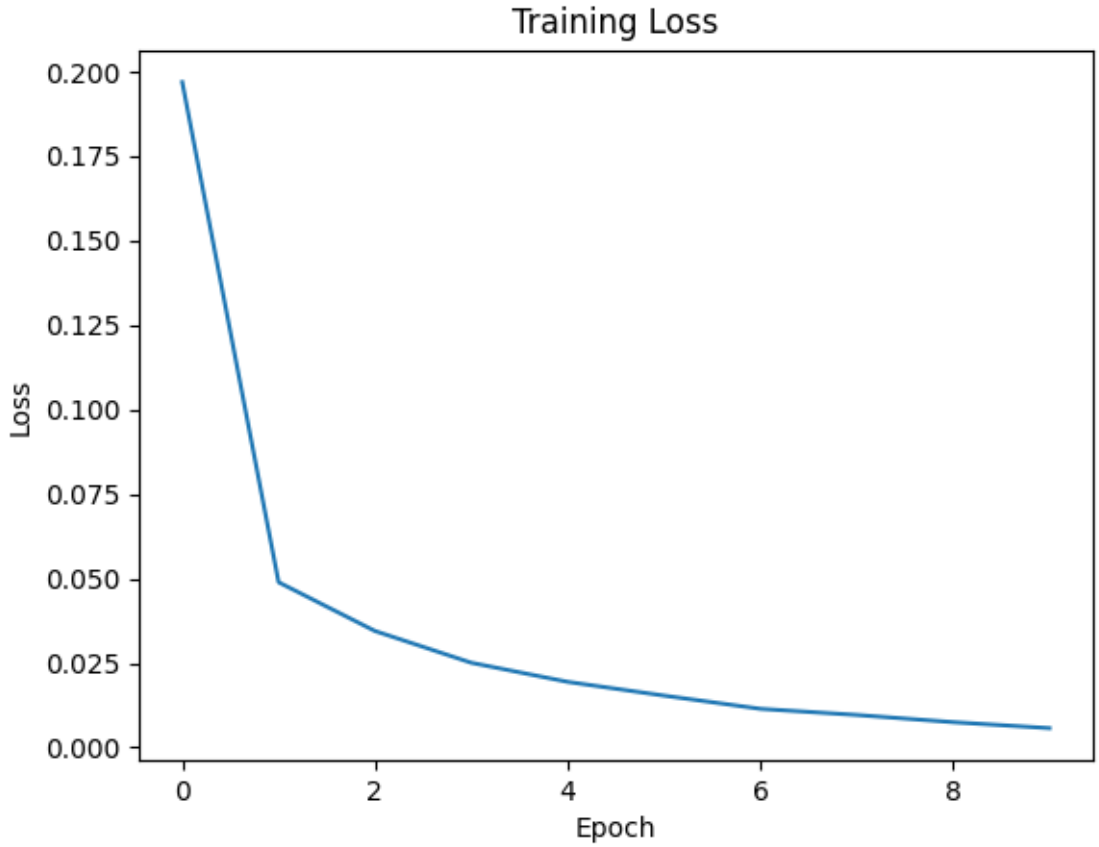
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))  # Нормализация для выборки
MNIST
])
train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
transform=transform, download=True)
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=1000, shuffle=False)
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=5, padding=2)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5, padding=2)
        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 32 * 7 * 7)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
def train_model(num_epochs):
    train_loss_history = []
    for epoch in range(num_epochs):
```

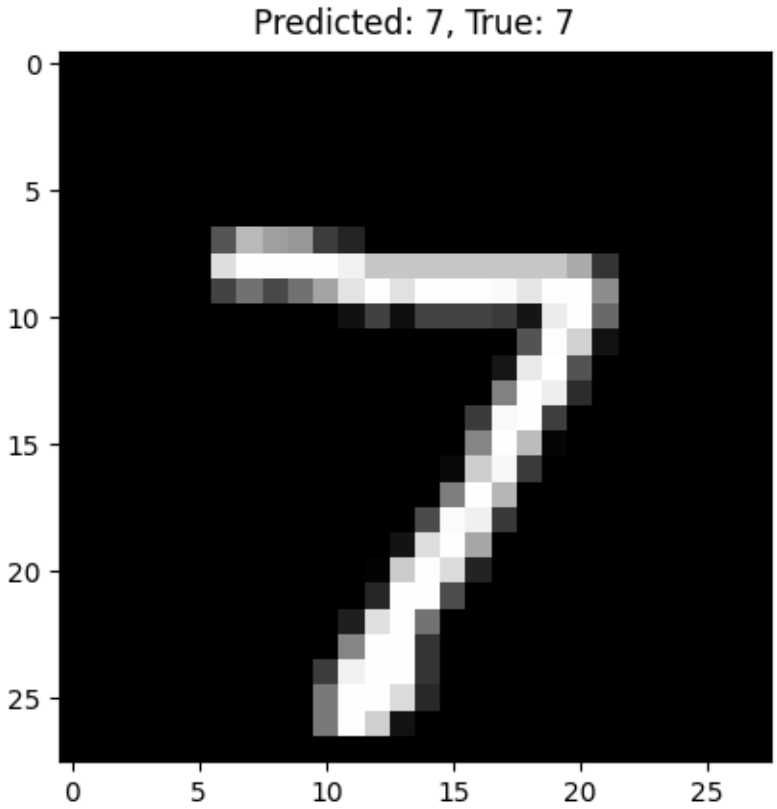
`

```
        running_loss = 0.0
        model.train()
        for i, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        epoch_loss = running_loss / len(train_loader)
        train_loss_history.append(epoch_loss)
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}')
    return train_loss_history
def test_model():
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print(f'Accuracy on the test set: {accuracy:.2f}%')
    return accuracy
def plot_loss_history(loss_history):
    plt.plot(loss_history)
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()
num_epochs = 10
loss_history = train_model(num_epochs)
test_model()
plot_loss_history(loss_history)
def visualize_prediction(image_index):
    image, label = test_dataset[image_index]
    model.eval()
    with torch.no_grad():
        image = image.unsqueeze(0).to(device)
        output = model(image)
        _, predicted = torch.max(output.data, 1)
    plt.imshow(image.cpu().squeeze(), cmap='gray')
    plt.title(f'Predicted: {predicted.item()}, True: {label}')
    plt.show()
visualize_prediction(0)
```

`

Результат программы:





`

```
Epoch [1/10], Loss: 0.1969
Epoch [2/10], Loss: 0.0489
Epoch [3/10], Loss: 0.0345
Epoch [4/10], Loss: 0.0251
Epoch [5/10], Loss: 0.0194
Epoch [6/10], Loss: 0.0153
Epoch [7/10], Loss: 0.0114
Epoch [8/10], Loss: 0.0096
Epoch [9/10], Loss: 0.0075
Epoch [10/10], Loss: 0.0057
Accuracy on the test set: 99.17%
```

Цель: научился конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.