Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине: «Обработка изображений в интеллектуальных системах»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-23

Бусень А.Д.

Проверила:

Андренко К.В.

Брест 2025

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

Вариант 1.

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

Код программы:

```
import random
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt

trainset = torchvision.datasets.MNIST(
    root="./data",
    train=True,
    download=True,
    transform=transforms
)

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))  # нормализация для MNIST
])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=1000, shuffle=False)

class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, 3, padding=1)   # 28x28 -> 28x28
        self.pool = nn.MaxPool2d(2, 2)              # 28x28 -> 14x14
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1) # 14x14 -> 14x14
        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
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = SimpleCNN().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)

epochs = 100
train_losses = []
test_losses = []

for epoch in range(epochs):
    net.train()
    running_loss = 0.0
    for images, labels in trainloader:
        images, labels = images.to(device), labels.to(device)

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

        running_loss += loss.item()
    train_losses.append(running_loss / len(trainloader))

    net.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for images, labels in testloader:
            images, labels = images.to(device), labels.to(device)
            outputs = net(images)
            test_loss += criterion(outputs, labels).item()
            pred = outputs.argmax(dim=1)
            correct += pred.eq(labels).sum().item()
    test_losses.append(test_loss / len(testloader))

    acc = 100. * correct / len(testset)
    print(f"Epoch {epoch+1}: Train loss={train_losses[-1]:.4f}, Test loss={test_losses[-1]:.4f},
Accuracy={acc:.2f}%")

plt.plot(train_losses, label="Train loss")
plt.plot(test_losses, label="Test loss")
plt.legend()
plt.show()

net.eval()
idx = random.randint(0, len(testset)-1)
image, label = testset[idx]
with torch.no_grad():
```

```
    output = net(image.unsqueeze(0).to(device))
    pred = output.argmax(dim=1).item()

plt.imshow(image.squeeze(), cmap="gray")
plt.title(f"True: {label}, Pred: {pred}")
plt.show()
```

Результат работы программы:

Epoch 92: Train loss=0.0000, Test loss=0.0417, Accuracy=99.13%

Epoch 93: Train loss=0.0000, Test loss=0.0418, Accuracy=99.13%

Epoch 94: Train loss=0.0000, Test loss=0.0418, Accuracy=99.12%

Epoch 95: Train loss=0.0000, Test loss=0.0418, Accuracy=99.13%

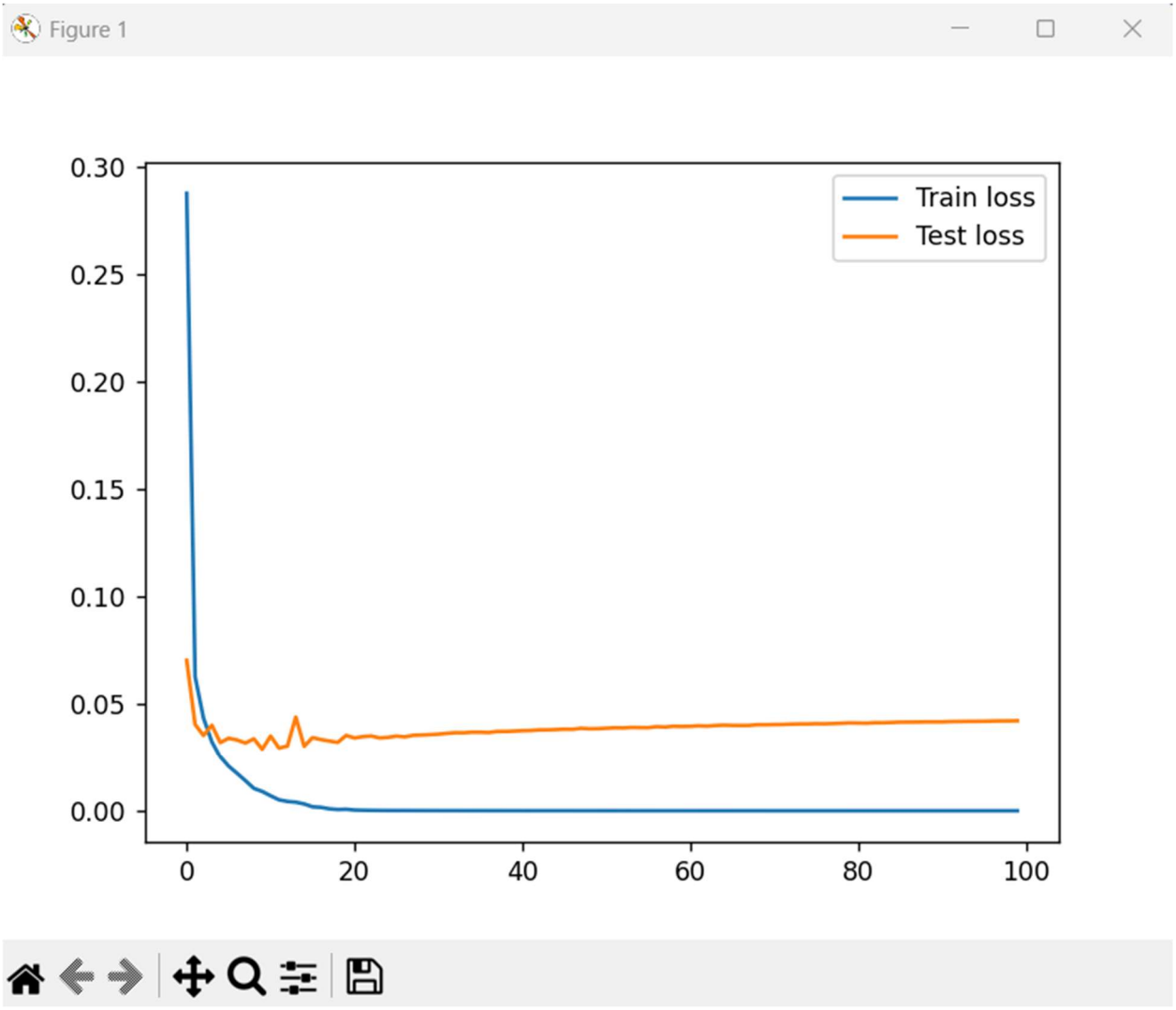
Epoch 96: Train loss=0.0000, Test loss=0.0419, Accuracy=99.13%

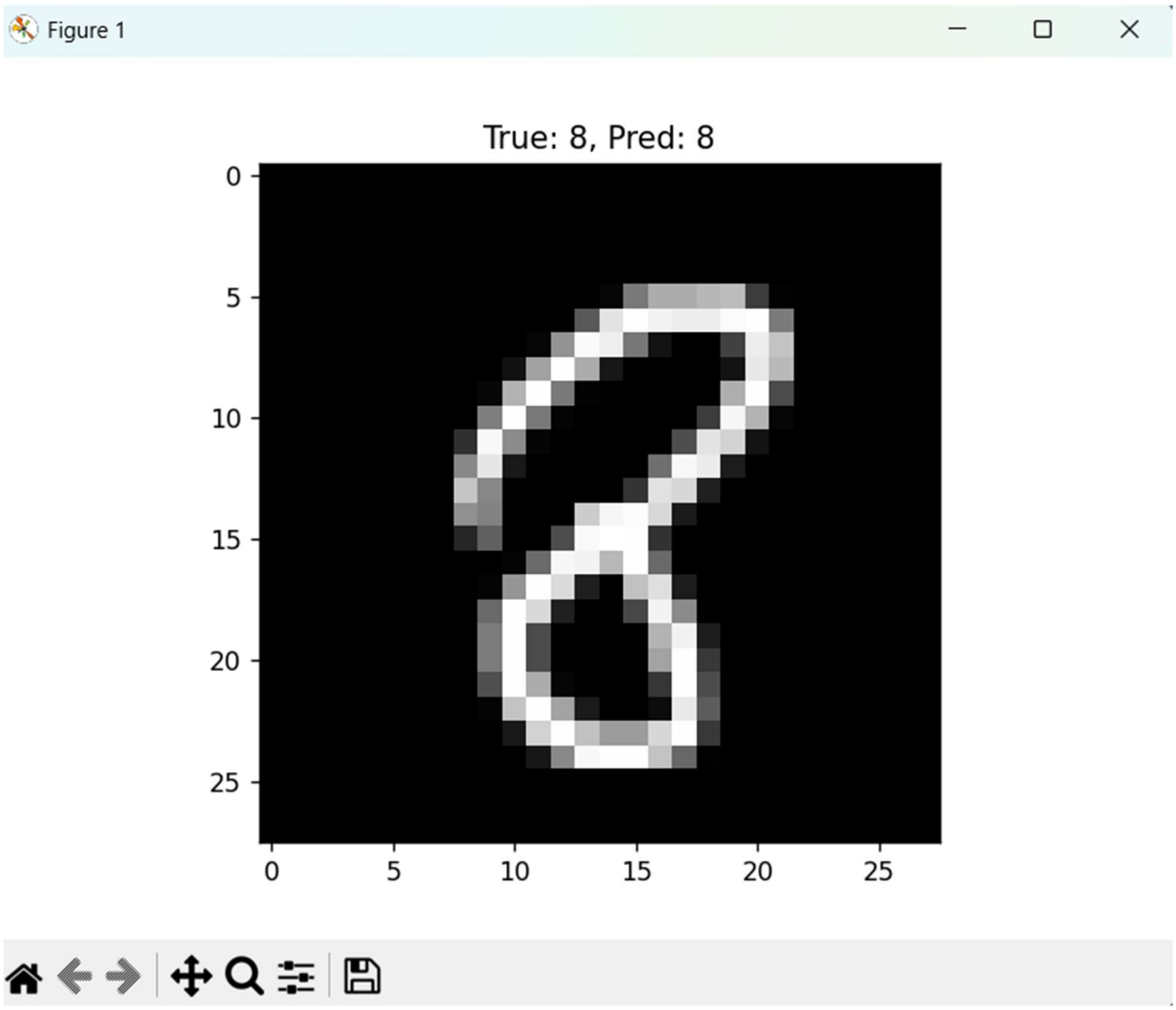
Epoch 97: Train loss=0.0000, Test loss=0.0420, Accuracy=99.12%

Epoch 98: Train loss=0.0000, Test loss=0.0420, Accuracy=99.13%

Epoch 99: Train loss=0.0000, Test loss=0.0421, Accuracy=99.13%

Epoch 100: Train loss=0.0000, Test loss=0.0421, Accuracy=99.13%





LeNet-5	~60K	99%	Классическая архитектура от LeCun
SimpleCNN (твоя)	~60K	98.5–99%	Быстрая, лёгкая, обучается на CPU

ResNet-18 адаптированный	~11M	99.6–99.7%	Skip connections, BatchNorm
DenseNet-121 адаптированный	~8M	99.7%	Плотные соединения, более глубокая CNN
CapsuleNet (Hinton)	~8M	99.75%	Специализирована для распознавания цифр
Vision Transformer (ViT-Small)	~22M	99.8%	Современная SOTA, требует больше данных

Моя модель отлично подходит для учебных задач, быстрых экспериментов и CPU-обучения.

Для SOTA результатов (~99.7–99.8%) требуется более глубокая архитектура, регуляризация (BatchNorm, Dropout), data augmentation и иногда ансамбли моделей.

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