Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Обработка изображений в ИС»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-21

Корнейчук А.И.

Проверил:

Крощенко А.А.

Брест 2024

Вариант 11

11	MNIST	28X28	Adadelta

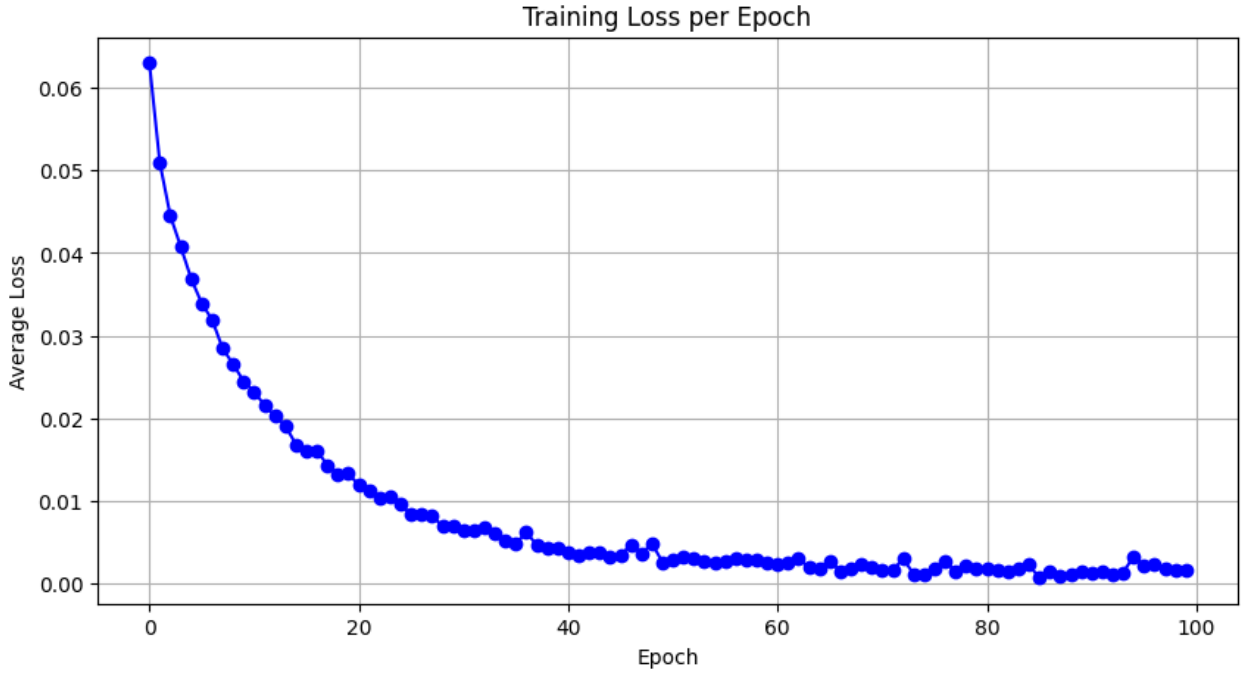
Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

Код программы:

```
import torch
import torchvision  # type: ignore
import torch.nn as nn
from tqdm import tqdm
import matplotlib.pyplot as plt
batch_size_train = 128
batch_size_test = 1000
train_loader = torch.utils.data.DataLoader(
        torchvision.datasets.MNIST(root='./data', train=True, download=True,
                                   transform=torchvision.transforms.Compose([
                                       torchvision.transforms.ToTensor(),
                                       torchvision.transforms.Normalize((0.1307,), (0.3081,))
                                   ])), batch_size=batch_size_train, shuffle=True
    )
test_loader = torch.utils.data.DataLoader(
        torchvision.datasets.MNIST(root='./data', train=False, download=True,
                                   transform=torchvision.transforms.Compose([
                                       torchvision.transforms.ToTensor(),
                                       torchvision.transforms.Normalize((0.1307,), (0.3081,))
                                   ])), batch_size=batch_size_test, shuffle=True
    )
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.act = nn.LeakyReLU(0.2)
        self.dropout = nn.Dropout(0.5)
        self.maxpool = nn.MaxPool2d(2, 2)
        self.conv_0 = nn.Conv2d(1, 8, 3, 1)
        self.conv_1 = nn.Conv2d(8, 16, 3, 1)
        self.conv_2 = nn.Conv2d(16, 32, 3, 1)

        self.flatten = nn.Flatten()
        self.linear_1 = nn.Linear(32, 25)
        self.linear_2 = nn.Linear(25, 15)
        self.linear_3 = nn.Linear(15, 10)

    def forward(self, x):
        out = self.conv_0(x)
        out = self.act(out)
        out = self.maxpool(out)

        out = self.conv_1(out)
        out = self.act(out)
        out = self.maxpool(out)
        out = self.dropout(out)

        out = self.conv_2(out)
        out = self.act(out)
        out = self.maxpool(out)
        out = self.dropout(out)

        out = self.flatten(out)
        out = self.linear_1(out)
        out = self.act(out)
        out = self.linear_2(out)
        out = self.act(out)
        out = self.linear_3(out)
```

```
        return out
def train(device, model, train_loader, learning_rate=1, epochs=5):
    loss_fn = nn.CrossEntropyLoss().to(device)
    optimizer = torch.optim.Adadelta(model.parameters(), lr=learning_rate)
    loss_history = []

    for epoch in tqdm(range(epochs), desc="Training Progress"):
        total_loss = 0.0

        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)
            optimizer.zero_grad()

            predictions = model(inputs)
            loss = loss_fn(predictions, targets)
            total_loss += loss.item()

            loss.backward()
            optimizer.step()

        avg_loss = total_loss / len(train_loader)
        loss_history.append(avg_loss)

        print(f"Epoch {epoch + 1}/{epochs}, Loss: {avg_loss:.4f}")

    plt.figure(figsize=(10, 5))
    plt.plot(loss_history, marker='o', linestyle='-', color='b')
    plt.xlabel('Epoch')
    plt.ylabel('Average Loss')
    plt.title('Training Loss per Epoch')
    plt.grid(True)
    plt.show()
def test(model, device, test_loader):
    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, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = correct / total
    print(f"Accuracy on the test set: {accuracy:.2%}")
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Net().to(device)
train(device, model, train_loader, epochs=100)
test(model, device, test_loader)
```

График изменения ошибки при обучении



Точность на тестовой выборке

```
test(model, device, test_loader)
```

```
Accuracy on the test set: 98.54%
```

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