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

Шпак И.С.

Проверил:

Крощенко А.А.

Брест 2024

Вариант 16

16	MNIST	28X28	RMSprop

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

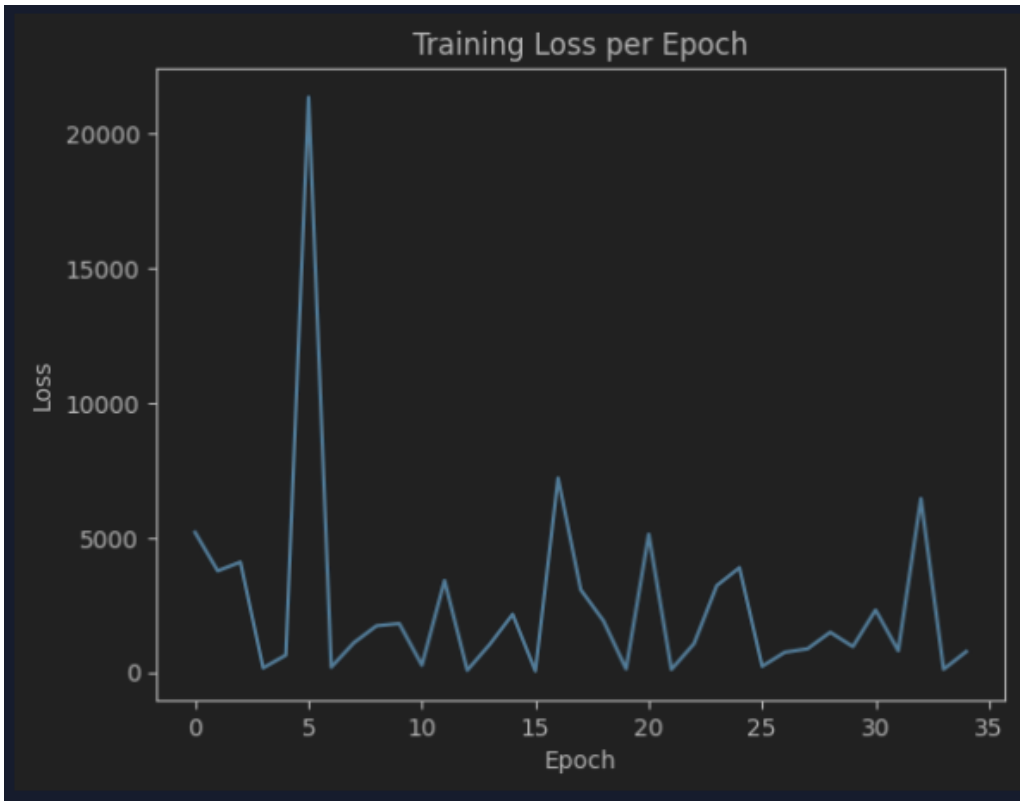
Код программы:

```
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.conv_block1 = nn.Sequential(
         nn.Conv2d(in_channels=1, out_channels=8, kernel_size=5, stride=1, padding=1),
         nn.LeakyReLU(0.2),
         nn.MaxPool2d(2, 2),
         nn.Dropout(0.5)
      )
      self.conv_block2 = nn.Sequential(
         nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, stride=1),
         nn.LeakyReLU(0.2),
         nn.MaxPool2d(2, 2),
```

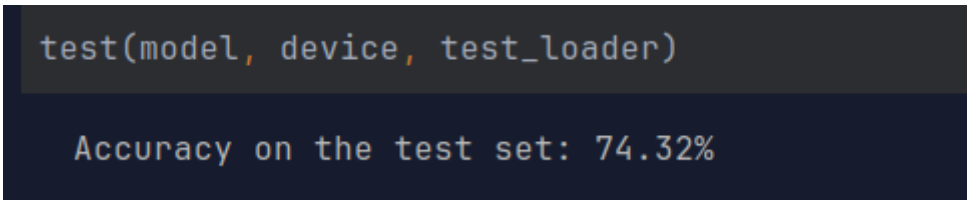
```
            nn.Dropout(0.5)
        )
        self.conv_block3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1),
            nn.LeakyReLU(0.2),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.5)
        )
        self.flc_block = nn.Sequential(
            nn.Flatten(),
            nn.Linear(32, 25),
            nn.Linear(25, 15),
            nn.Linear(15, 10)
        )

    def forward(self, x):
        x = self.conv_block1(x)
        x = self.conv_block2(x)
        x = self.conv_block3(x)
        x = self.flc_block(x)

        return x

def train(device, model, train_loader, learning_rate=0.05, epochs=5):
    loss_fn = nn.CrossEntropyLoss().to(device)
    optimizer = torch.optim.RMSprop(model.parameters(), lr=learning_rate)
    history = []
    for epoch in tqdm(range(epochs)):
        epoch_loss = 0.0
        for x, y in train_loader:
            optimizer.zero_grad()
            x, y = x.to(device), y.to(device)
            pred = model(x)

            loss = loss_fn(pred, y)
            epoch_loss += loss.item()
            loss.backward()
            optimizer.step()

        average_loss = epoch_loss / len(train_loader)
        history.append(average_loss)

        print(f'Epoch {epoch + 1}, Loss: {average_loss}')
    plt.plot(range(0, epochs), history)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
```

```
    plt.title('Training Loss per Epoch')
    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)
```

График изменения ошибки при обучении



Точность на тестовой выборке



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