МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет
по дисциплине
«Обработка изображений в интеллектуальных системах»
по лабораторной работе № 1
«Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:
студент 4 курса
группы ИИ-22
Сидоренко А.А
Проверил:
Крощенко А.А.

Брест 2024

Цель: научиться конструировать нейросетевые классификаторы и выпол-нять их обучение на известных выборках компьютерного зрения.

Ход работы:

Для выполнения лабораторной работы требуется:

- Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых ти-пах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);
- Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (https://paperswithcode.com/task/image-classification). Сделать выводы о результатах обучения СНС из п. 1;
- Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата).

Вариант:

№	Выборка	Размер изображения	Оптимизатор
16	MNIST	28X28	RMSprop

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as opt
import torchvision as tvs
import torchvision.transforms as tfs
import torch.utils.data as data
from tqdm import tqdm

# Вариант 16

image_size = 28 * 28
lr = 0.001
epochs = 20
hidden_size = 512
hidden_size_2 = 128
batch_size = 128
output = 10

transform = tfs.Compose([tfs.ToTensor(), tfs.Normalize((0.5,), (0.5,))])
train_dst = tvs.datasets.MNIST('./data', train=True, transform=transform,
download=True)
test_dst = tvs.datasets.MNIST('./data', train=False, transform=transform,
download=True)


class NN(nn.Module):
    def __init__(self, input_dim, num_hidden, num_hidden2, output_dim):
```

```
        super().__init__()
        self.layer_1 = nn.Linear(input_dim, num_hidden)
        self.layer_2 = nn.Linear(num_hidden, num_hidden2)
        self.layer_3 = nn.Linear(num_hidden2, output_dim)

    def forward(self, x):
        x = self.layer_1(x)
        x = nn.functional.relu(x)
        x = self.layer_2(x)
        x = nn.functional.relu(x)
        x = self.layer_3(x)
        return x


def train(model, data_train, loss_f, optimizer):
    for i in range(epochs):
        loss_mean = 0
        lm_count = 0
        train_tqdm = tqdm(data_train, leave=True)
        for x_train, y_train in train_tqdm:
            x_train = x_train.view(-1, image_size)
            predict = model(x_train)
            loss = loss_f(predict, y_train)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            lm_count += 1
            loss_mean = 1 / lm_count * loss.item() + (1 - 1 / lm_count) *
loss_mean

            train_tqdm.set_description(f"Epoch[{i + 1}/{epochs}],
loss_mean={loss_mean:.3f}")


def test(model, data_test):
    correct = 0
    total = 0
    for x_test, y_test in data_test:
        with torch.no_grad():
            x_test = x_test.view(-1, image_size)
            p = model(x_test)
            p = torch.argmax(p, dim=1)

            correct += (p == y_test).sum().item()
            total += y_test.size(0)

    accuracy = correct / total
    return accuracy


def main():
    model = NN(image_size, hidden_size, hidden_size_2, output)
    optimizer = opt.RMSprop(params=model.parameters(), lr=lr)
    loss_f = nn.CrossEntropyLoss()
    data_train = data.DataLoader(train_dst, batch_size=batch_size, shuffle=True)
    data_test = data.DataLoader(test_dst, batch_size=batch_size, shuffle=False)
    model.train()
    train(model, data_train, loss_f, optimizer)
    model.eval()
    accuracy = test(model, data_test)
    print(f"Точность: {accuracy:.4f}")


if __name__ == "__main__":
    main()
```

Результат:

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