

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 выполнять 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 типах 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:

```
C:\Users\Xiaomi\.conda\envs\ip_ai_22\python.exe C:\Users\Xiaomi\Desktop\prjs\ip_ai_22\reports\Сидорен
Epoch[1/20], loss_mean=0.413: 100%|██████████| 469/469 [00:17<00:00, 26.11it/s]
Epoch[2/20], loss_mean=0.165: 100%|██████████| 469/469 [00:19<00:00, 23.81it/s]
Epoch[3/20], loss_mean=0.115: 100%|██████████| 469/469 [00:19<00:00, 24.13it/s]
Epoch[4/20], loss_mean=0.092: 100%|██████████| 469/469 [00:18<00:00, 25.09it/s]
Epoch[5/20], loss_mean=0.073: 100%|██████████| 469/469 [00:18<00:00, 25.06it/s]
Epoch[6/20], loss_mean=0.064: 100%|██████████| 469/469 [00:18<00:00, 26.03it/s]
Epoch[7/20], loss_mean=0.056: 100%|██████████| 469/469 [00:17<00:00, 26.85it/s]
Epoch[8/20], loss_mean=0.050: 100%|██████████| 469/469 [00:17<00:00, 26.63it/s]
Epoch[9/20], loss_mean=0.044: 100%|██████████| 469/469 [00:17<00:00, 26.59it/s]
Epoch[10/20], loss_mean=0.037: 100%|██████████| 469/469 [00:17<00:00, 26.34it/s]
Epoch[11/20], loss_mean=0.034: 100%|██████████| 469/469 [00:18<00:00, 25.35it/s]
Epoch[12/20], loss_mean=0.030: 100%|██████████| 469/469 [00:17<00:00, 26.07it/s]
Epoch[13/20], loss_mean=0.028: 100%|██████████| 469/469 [00:18<00:00, 26.04it/s]
Epoch[14/20], loss_mean=0.026: 100%|██████████| 469/469 [00:20<00:00, 23.15it/s]
Epoch[15/20], loss_mean=0.024: 100%|██████████| 469/469 [00:21<00:00, 22.21it/s]
Epoch[16/20], loss_mean=0.021: 100%|██████████| 469/469 [00:22<00:00, 21.03it/s]
Epoch[17/20], loss_mean=0.022: 100%|██████████| 469/469 [00:19<00:00, 23.66it/s]
Epoch[18/20], loss_mean=0.019: 100%|██████████| 469/469 [00:20<00:00, 23.10it/s]
Epoch[19/20], loss_mean=0.019: 100%|██████████| 469/469 [00:21<00:00, 21.46it/s]
Epoch[20/20], loss_mean=0.018: 100%|██████████| 469/469 [00:20<00:00, 22.50it/s]
Точность: 0.9788
```

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