Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №1

По дисциплине: «ОИвИС»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнила:
Студентка 4 курса
Группы ИИ-23
Тутина Е.Д.
Проверила:
Андренко К. В.

Брест 2025

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

Вариант 11

Выборка: MNIST
Размер исходного изображения: 28x28
Оптимизатор: Adadelta

Задание:

1. 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);

2. Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (из материалов в сети Интернет). Сделать выводы о результатах обучения СНС из п. 1;

3. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);

4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Код работы:

```
import argparse
import os
import time
import json

import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

import numpy as np
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

class SimpleCNN(nn.Module):
   def __init__(self):
       super().__init__()
```

```
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)
        self.pool  = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.fc1   = nn.Linear(32 * 7 * 7, 128)
        self.fc2   = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))    # 28x28 -> 28x28
        x = self.pool(x)             # -> 14x14
        x = F.relu(self.conv2(x))    # -> 14x14
        x = self.pool(x)             # -> 7x7
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

def train_one_epoch(model, device, loader, opt, criterion):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for data, target in loader:
        data, target = data.to(device), target.to(device)
        opt.zero_grad()
        out = model(data)
        loss = criterion(out, target)
        loss.backward()
        opt.step()
        running_loss += loss.item() * data.size(0)
        _, pred = out.max(1)
        total += target.size(0)
        correct += pred.eq(target).sum().item()
    return running_loss / total, 100.0 * correct / total

def evaluate(model, device, loader, criterion):
    model.eval()
    loss_sum = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in loader:
            data, target = data.to(device), target.to(device)
            out = model(data)
            loss = criterion(out, target)
            loss_sum += loss.item() * data.size(0)
            _, pred = out.max(1)
            total += target.size(0)
            correct += pred.eq(target).sum().item()
    return loss_sum / total, 100.0 * correct / total

def save_plots(history, save_dir):
    os.makedirs(save_dir, exist_ok=True)
    epochs = list(range(1, len(history['train_loss']) + 1))

    plt.figure(figsize=(6,4))
    plt.plot(epochs, history['train_loss'], marker='o', label='Train loss')
    plt.plot(epochs, history['test_loss'], marker='o', label='Test loss')
    plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Loss')
    plt.legend(); plt.tight_layout()
    loss_path = os.path.join(save_dir, 'loss.png')
    plt.savefig(loss_path); plt.close()

    plt.figure(figsize=(6,4))
    plt.plot(epochs, history['test_acc'], marker='o', label='Test acc')
    plt.xlabel('Epoch'); plt.ylabel('Accuracy (%)'); plt.title('Test Accuracy')
    plt.legend(); plt.tight_layout()
    acc_path = os.path.join(save_dir, 'acc.png')
```

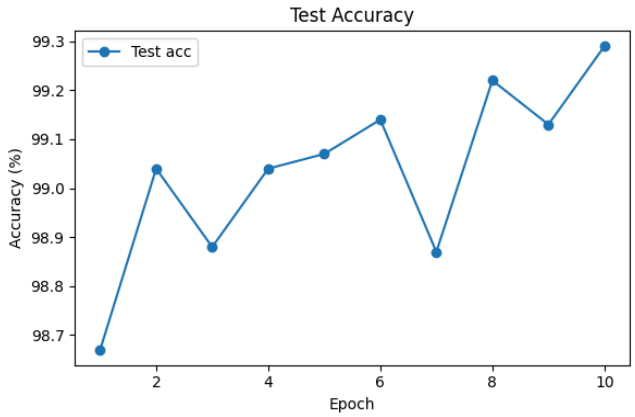
```
    plt.savefig(acc_path); plt.close()

    return loss_path, acc_path

def predict_image(model, device, img_path):
    img = Image.open(img_path).convert('L').resize((28,28))
    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize((0.1307,), (0.3081,))])
    x = transform(img).unsqueeze(0).to(device)
    model.eval()
    with torch.no_grad():
        logits = model(x)
        probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
        pred = int(probs.argmax())
    return img, pred, probs

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--epochs', type=int, default=10)
    parser.add_argument('--batch-size', type=int, default=64)
    parser.add_argument('--lr', type=float, default=1.0)    # Adadelta lr default 1.0
    parser.add_argument('--save-dir', type=str, default='results_mnist')
    parser.add_argument('--use-cuda', action='store_true')
    parser.add_argument('--predict', type=str, default=None, help='path to image to predict')
    args = parser.parse_args()

    device = torch.device('cuda' if args.use_cuda and torch.cuda.is_available() else 'cpu')
    os.makedirs(args.save_dir, exist_ok=True)

    transform = transforms.Compose([transforms.ToTensor(),
                                    transforms.Normalize((0.1307,), (0.3081,))])

    train_ds = datasets.MNIST('./data', train=True, download=True, transform=transform)
    test_ds  = datasets.MNIST('./data', train=False, download=True, transform=transform)

    train_loader = DataLoader(train_ds, batch_size=args.batch_size, shuffle=True, num_workers=2)
    test_loader  = DataLoader(test_ds,  batch_size=1000, shuffle=False, num_workers=2)

    model = SimpleCNN().to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adadelta(model.parameters(), lr=args.lr)

    history = {'train_loss': [], 'test_loss': [], 'test_acc': []}
    best_acc = 0.0
    start_time = time.time()

    for epoch in range(1, args.epochs + 1):
        tr_loss, tr_acc = train_one_epoch(model, device, train_loader, optimizer, criterion)
        te_loss, te_acc = evaluate(model, device, test_loader, criterion)

        history['train_loss'].append(tr_loss)
        history['test_loss'].append(te_loss)
        history['test_acc'].append(te_acc)

        print(f"Epoch {epoch}/{args.epochs}  TrainLoss={tr_loss:.4f}  TestLoss={te_loss:.4f}
TestAcc={te_acc:.2f}%")

        if te_acc > best_acc:
            best_acc = te_acc
            torch.save({'model_state': model.state_dict(), 'acc': best_acc, 'epoch': epoch},
                       os.path.join(args.save_dir, 'best.pth'))

    total_time = time.time() - start_time
    print(f"Training finished in {total_time/60:.2f} minutes. Best test acc: {best_acc:.2f}%")

    # Save model + history
    torch.save({'model_state': model.state_dict()}, os.path.join(args.save_dir, 'final.pth'))
```

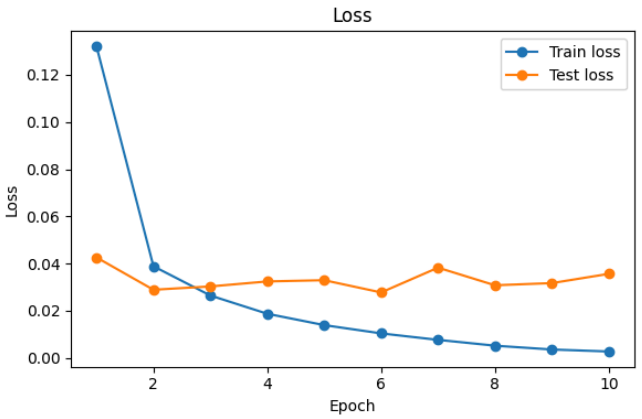
```
    with open(os.path.join(args.save_dir, 'history.json'), 'w') as f:
        json.dump(history, f)

    loss_path, acc_path = save_plots(history, args.save_dir)
    print("Saved plots:", loss_path, acc_path)

    if args.predict:
        img, pred, probs = predict_image(model, device, args.predict)
        out_path = os.path.join(args.save_dir, 'prediction.png')
        import matplotlib.pyplot as plt
        plt.figure(figsize=(3,3))
        plt.imshow(img, cmap='gray'); plt.axis('off')
        plt.title(f'Pred: {pred} ({probs[pred]*100:.1f}%)')
        plt.savefig(out_path, dpi=150); plt.close()
        print("Saved prediction image to", out_path)

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

Результат работы программы:

```
Epoch 1/10  TrainLoss=0.1321  TestLoss=0.0426  TestAcc=98.67%
Epoch 2/10  TrainLoss=0.0389  TestLoss=0.0289  TestAcc=99.04%
Epoch 3/10  TrainLoss=0.0265  TestLoss=0.0303  TestAcc=98.88%
Epoch 4/10  TrainLoss=0.0187  TestLoss=0.0324  TestAcc=99.04%
Epoch 5/10  TrainLoss=0.0139  TestLoss=0.0330  TestAcc=99.07%
Epoch 6/10  TrainLoss=0.0104  TestLoss=0.0278  TestAcc=99.14%
Epoch 7/10  TrainLoss=0.0077  TestLoss=0.0383  TestAcc=98.87%
Epoch 8/10  TrainLoss=0.0052  TestLoss=0.0308  TestAcc=99.22%
Epoch 9/10  TrainLoss=0.0036  TestLoss=0.0317  TestAcc=99.13%
Epoch 10/10  TrainLoss=0.0027  TestLoss=0.0357  TestAcc=99.29%
```

Графики акураси и потерь:





Сравнение с результатами SOTA : научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

State-of-the-Art (SOTA) по MNIST:

Классические CNN (LeNet-5): ~99.2–99.3%
ResNet, VGG, более сложные CNN: ~99.5–99.7%
Capsule Networks (Hinton, 2017): ~99.75%
Ансамбль моделей: ~99.8–99.9%

Результаты, полученные при обучении модели в ходе лабораторной работы: `TestAcc=99.29%` Отставание от лучших моделей (99.8–99.9%) небольшое (~0.5–0.6%), связано с тем, что SOTA использует:

1. очень глубокие/широкие архитектуры,
2. специальные техники регуляризации,
3. ансамбли нескольких сетей,
4. кастомные аугментации данных.

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