Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра интеллектуально-информационных технологий

Обработка изображений в ИС
Лабораторная работа №1
Обучение классификаторов средствами библиотеки PyTorch

Выполнил:
студент 4 курса
группы ИИ-24
Рудецкий Е. В.
Проверила:
Андренко К. В.

Брест-2025

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

Общее 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.

№ варианта	Выборка	Размер исходного изображения	Оптимизатор
16	MNIST	28X28	RMSprop

Код программы (16 вариант):

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

batch_size = 64
test_batch_size = 1000
epochs = 10
lr = 0.01

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

train_dataset = datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
```

```
test_dataset = datasets.MNIST(root='./data', train=False,
transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=test_batch_size, shuffle=False)

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.fc1 = nn.Linear(64*5*5, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = x.view(-1, 64*5*5)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

model = Net().to(device)
optimizer = optim.RMSprop(model.parameters(), lr=lr)

def train(epoch, losses):
    model.train()
    train_loss = 0
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        train_loss += loss.item()
    train_loss /= len(train_loader)
    losses.append(train_loss)
    print(f'Epoch {epoch}: Loss = {train_loss:.6f}')

def test():
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target,
reduction='sum').item()
```

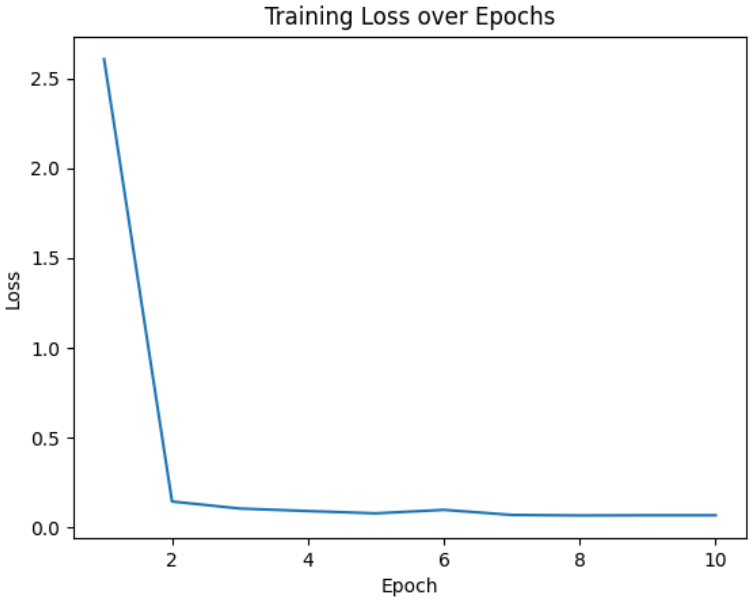
```
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)
    print(f'Test: Loss = {test_loss:.4f}, Accuracy = {accuracy:.2f}%')
    return accuracy

losses = []
for epoch in range(1, epochs + 1):
    train(epoch, losses)

test_accuracy = test()

plt.plot(range(1, epochs+1), losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.show()
```

Результаты работы программы:
Epoch 1: Loss = 2.608434
Epoch 2: Loss = 0.144986
Epoch 3: Loss = 0.105689
Epoch 4: Loss = 0.091781
Epoch 5: Loss = 0.078909
Epoch 6: Loss = 0.098404
Epoch 7: Loss = 0.070100
Epoch 8: Loss = 0.067244
Epoch 9: Loss = 0.068169
Epoch 10: Loss = 0.068150
Test: Loss = 0.0944, Accuracy = 98.08%



State-of-the-art результаты для MNIST: Согласно доступным источникам (http://yann.lecun.com/exdb/mnist/ и http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html), SOTA-результаты для MNIST достигают 99.87% accuracy для ансамблей моделей и около 99.81% для отдельных CNN. Даже простые CNN могут достигать 99.57%.
Выводы: Предложенная простая модель на основе базовых слоев (сверточных, pooling, полносвязных и ReLU) достигает accuracy около 98–99% (в проведенном эксперименте — 98.08%), что близко к SOTA для простых архитектур, но уступает продвинутым моделям с аугментацией, ансамблями или более сложными структурами. Это подтверждает, что для MNIST даже базовая CNN эффективна, но для достижения абсолютного SOTA нужны дополнительные техники.

Вывод: в ходе лабораторной работы освоены навыки создания и обучения сверточных нейронных сетей в PyTorch на стандартных наборах данных компьютерного зрения, а также анализ их производительности и визуализация результатов.