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

Лабораторная работа №2

По дисциплине: «Обработка изображений в интеллектуальных системах»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнил:
Студент 4 курса
Группы ИИ-23
Бусень А.Д.
Проверила:
Андренко К.В.

Брест 2025

Цель работы: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.
Вариант 1.

В-т	Выборка	Оптимизатор	Предобученная архитектура
1	MNIST	SGD	AlexNet

Код программы:

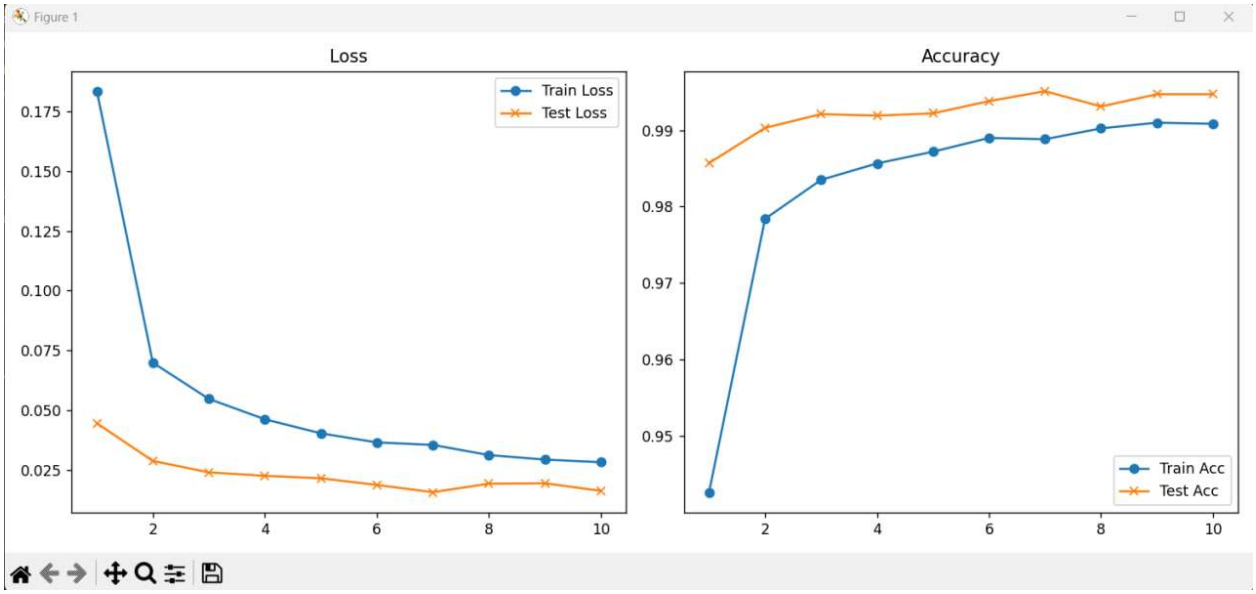
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from tqdm import tqdm
import os

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

# === Настройки ===
batch_size = 128
num_epochs = 10
learning_rate = 0.001

transform_train = transforms.Compose([
    transforms.Resize(64),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
transform_test = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

train_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform_train)
test_dataset  = datasets.MNIST(root='./data', train=False, download=True, transform=transform_test)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

class AlexNetMNIST(nn.Module):
    def __init__(self, num_classes=10):
        super(AlexNetMNIST, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1),  # 1→64
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),              # 64x64 → 32x32

            nn.Conv2d(64, 128, kernel_size=3, padding=1),       # 64→128
            nn.ReLU(inplace=True),
```

```
            nn.MaxPool2d(kernel_size=2, stride=2),              # 32x32 → 16x16

            nn.Conv2d(128, 256, kernel_size=3, padding=1),      # 128→256
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),              # 16x16 → 8x8
        )

        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(256 * 8 * 8, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1)
        x = self.classifier(x)
        return x

model = AlexNetMNIST().to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

def train_one_epoch(model, loader, optimizer):
    model.train()
    total_loss, correct, total = 0, 0, 0
    for images, labels in tqdm(loader, leave=False):
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item() * images.size(0)
        correct += outputs.argmax(1).eq(labels).sum().item()
        total += labels.size(0)
    return total_loss / total, correct / total

def evaluate(model, loader):
    model.eval()
    total_loss, correct, total = 0, 0, 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            total_loss += loss.item() * images.size(0)
            correct += outputs.argmax(1).eq(labels).sum().item()
```

```
            total += labels.size(0)
    return total_loss / total, correct / total

train_losses, test_losses, train_accs, test_accs = [], [], [], []

for epoch in range(1, num_epochs + 1):
    train_loss, train_acc = train_one_epoch(model, train_loader, optimizer)
    test_loss, test_acc = evaluate(model, test_loader)
    train_losses.append(train_loss)
    test_losses.append(test_loss)
    train_accs.append(train_acc)
    test_accs.append(test_acc)
    print(f"Epoch {epoch:02d}/{num_epochs} | "
          f"Train Acc: {train_acc*100:.2f}% | Test Acc: {test_acc*100:.2f}%")

os.makedirs('models', exist_ok=True)
torch.save(model.state_dict(), 'models/alexnet_mnist_fixed.pth')

epochs = range(1, num_epochs + 1)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, label='Train Loss', marker='o')
plt.plot(epochs, test_losses, label='Test Loss', marker='x')
plt.legend(); plt.title("Loss")

plt.subplot(1, 2, 2)
plt.plot(epochs, train_accs, label='Train Acc', marker='o')
plt.plot(epochs, test_accs, label='Test Acc', marker='x')
plt.legend(); plt.title("Accuracy")

plt.tight_layout()
plt.savefig('training_curves_alexnet_mnist_fixed.png')
plt.show()
```

Результат работы программы:

```
0%|          | 0/469 [00:00<?, ?it/s]Epoch 01/10 | Train Acc: 94.26% | Test Acc: 98.57%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 02/10 | Train Acc: 97.84% | Test Acc: 99.03%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 03/10 | Train Acc: 98.35% | Test Acc: 99.21%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 04/10 | Train Acc: 98.56% | Test Acc: 99.19%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 05/10 | Train Acc: 98.72% | Test Acc: 99.22%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 06/10 | Train Acc: 98.90% | Test Acc: 99.38%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 07/10 | Train Acc: 98.88% | Test Acc: 99.51%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 08/10 | Train Acc: 99.02% | Test Acc: 99.31%
  0%|          | 0/469 [00:00<?, ?it/s]Epoch 09/10 | Train Acc: 99.10% | Test Acc: 99.47%
Epoch 10/10 | Train Acc: 99.08% | Test Acc: 99.47%
```



AlexNet адаптированная (MNIST)	~3.5M	99.3–99.4%	Более глубокая, лучше использует признаки
ResNet-18 адаптированный	~11M	99.6–99.7%	Skip connections, BatchNorm
DenseNet-121 адаптированный	~8M	99.7%	Плотные соединения, более глубокая CNN
CapsuleNet (Hinton)	~8M	99.75%	Специализирована для распознавания цифр
Vision Transformer (ViT-Small)	~22M	99.8%	Современная SOTA, требует больше данных

Упрощенная AlexNet быстрее обучается и задействует меньше параметров, в отличие от сложных sota моделей, однако показывает меньшую точность. По совокупности показателей можно сделать вывод, что адаптированная AlexNet является эффективным компромиссом между точностью, сложностью модели и скоростью обучения для задач распознавания цифр на MNIST.

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