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

Выполнила:
студентка 4 курса
группы ИИ-24
Алешко А. 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
1	MNIST	28X28	SGD

Код программы(вариант 1):

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

# Нормализация для MNIST
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,
download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=64, shuffle=True)
```

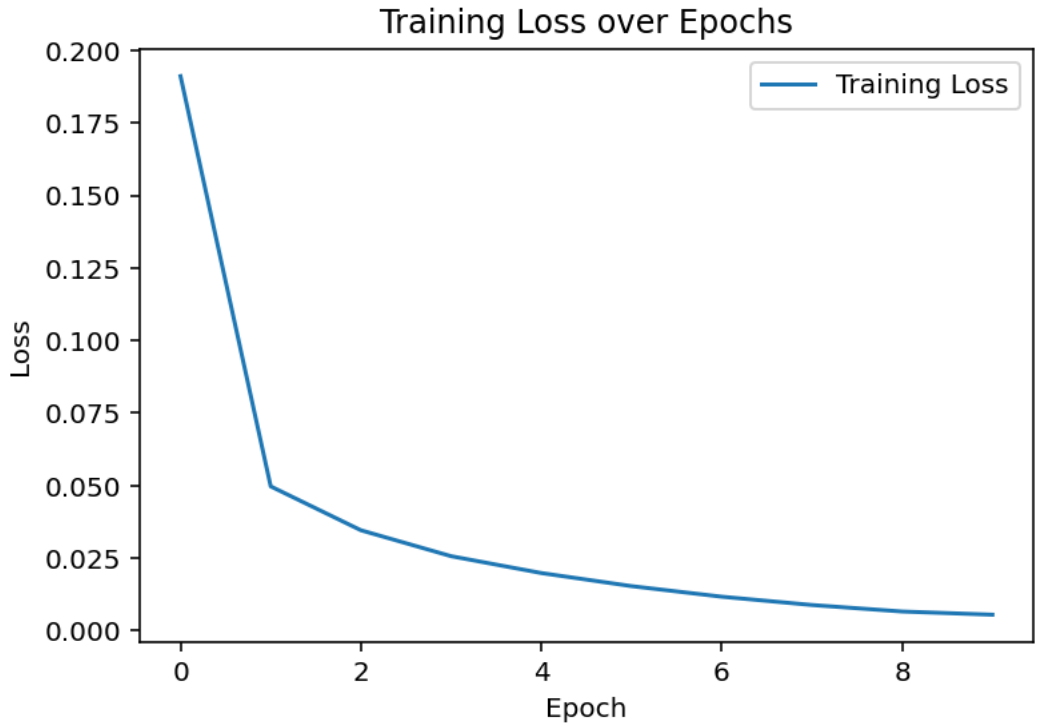
```
test_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=1000, shuffle=False)

# CNN архитектура
class SimpleCNN(nn.Module):

    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3,
padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3,
padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)  # Выход на 10 классов

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.flatten(x)
        x = self.relu3(self.fc1(x))
        x = self.fc2(x)
        return x

# Инициализация модели, лосса и оптимизатора
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()  # CrossEntropyLoss
optimizer = optim.SGD(model.parameters(), lr=0.01,
momentum=0.9)  # SGD оптимизатор

# Обучение модели
num_epochs = 10
train_losses = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for data, target in train_loader:
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    epoch_loss = running_loss / len(train_loader)
    train_losses.append(epoch_loss)
```

```
    print(f'Epoch {epoch+1}/{num_epochs}, Loss:
{epoch_loss:.4f}')

# Построение графика изменения ошибки
plt.plot(train_losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.legend()
plt.show()

# Оценка на тестовой выборке
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for data, target in test_loader:
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
        total += target.size(0)

accuracy = 100. * correct / total
print(f'Test Accuracy: {accuracy:.2f}%')

# Визуализация работы модели на произвольном изображении
data_iter = iter(test_loader)
images, labels = next(data_iter)
img = images[0]
true_label = labels[0]

with torch.no_grad():
    output = model(img.unsqueeze(0))
    pred_label = output.argmax().item()

plt.imshow(img.squeeze(), cmap='gray')
plt.title(f'Predicted: {pred_label}, True: {true_label}')
plt.show()
```

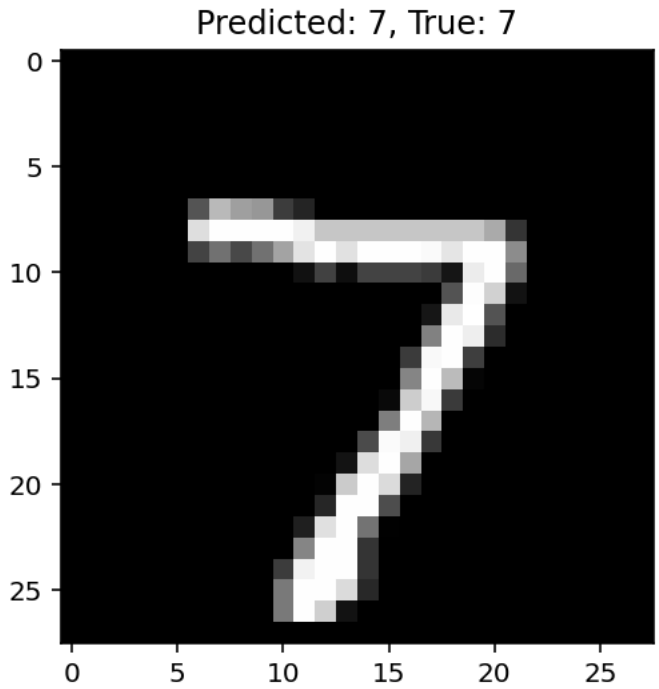
Результат работы программы:

100%|  | 9.91M/9.91M [00:08<00:00, 1.24MB/s]
100%|  | 28.9k/28.9k [00:00<00:00, 228kB/s]
100%|  | 1.65M/1.65M [00:02<00:00, 698kB/s]
100%|  | 4.54k/4.54k [00:00<00:00, 4.49MB/s]
Epoch 1/10, Loss: 0.1911
Epoch 2/10, Loss: 0.0497
Epoch 3/10, Loss: 0.0346
Epoch 4/10, Loss: 0.0257
Epoch 5/10, Loss: 0.0198

Epoch 6/10, Loss: 0.0153
Epoch 7/10, Loss: 0.0117
Epoch 8/10, Loss: 0.0088
Epoch 9/10, Loss: 0.0066
Epoch 10/10, Loss: 0.0055



Test Accuracy: 99.30%



State-of-the-art результаты для MNIST: Согласно доступным источникам(https://www.researchgate.net/publication/384853923_State-of-the-Art_Results_with_the_Fashion-MNIST_Dataset), 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 99% (точное значение зависит от запуска, но типично 98-99%), 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: научилась конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.