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

Лабораторная работа №1

По дисциплине «обработка изображений в ИС»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:
Студент 4 курса
Группы ИИ-24
Якимовец Е. Г.
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). 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:

```
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 MNISTtransform = 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)
```

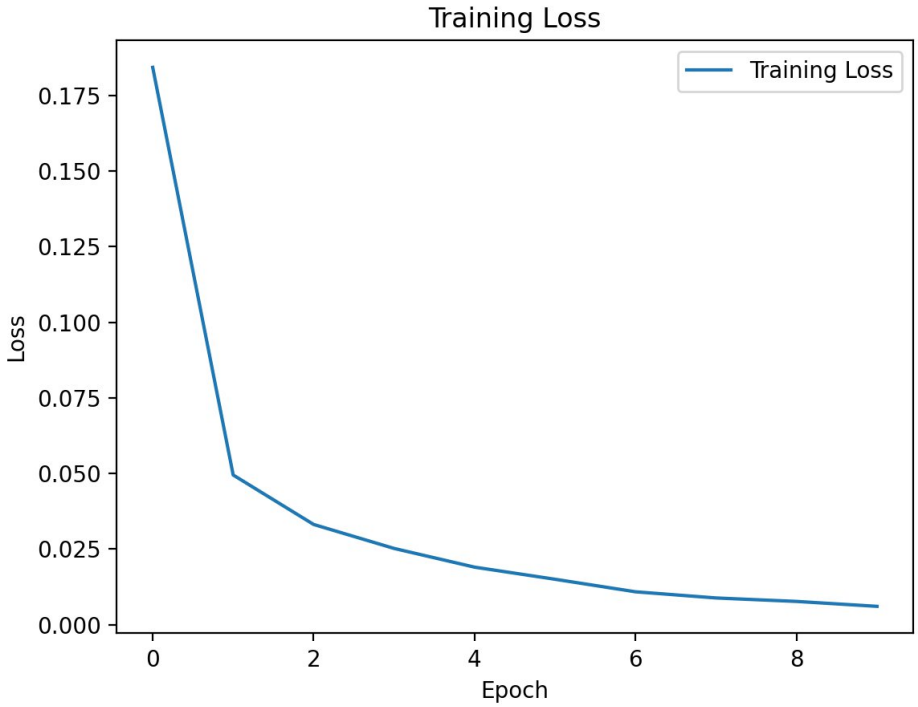
```
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=1000, shuffle=False)


# CNN архитектураclass MNIST_CNN(nn.Module):

    def __init__(self):
        super(MNIST_CNN, self).__init__()
        self.conv_block1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.act1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)

        self.conv_block2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.act2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)

        self.flatten = nn.Flatten()
        self.fc_block1 = nn.Linear(64 * 7 * 7, 128)
        self.act3 = nn.ReLU()
        self.fc_output = nn.Linear(128, 10)

    def forward(self, x):
        x = self.pool1(self.act1(self.conv_block1(x)))
        x = self.pool2(self.act2(self.conv_block2(x)))
        x = self.flatten(x)
        x = self.act3(self.fc_block1(x))
        x = self.fc_output(x)
        return x


# Инициализация модели, лосса и оптимизатораcnn_model = MNIST_CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(cnn_model.parameters(), lr=0.01, momentum=0.9)

# Обучение моделиnum_epochs = 10train_losses = []

for epoch in range(num_epochs):
    cnn_model.train()
    running_loss = 0.0    for inputs, labels in train_loader:
        optimizer.zero_grad()
```

```
        logits = cnn_model(inputs)
        loss = criterion(logits, labels)

        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 ошибокplt.plot(train_losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()
plt.show()

# Оценка точности на тестовой выборкеcnn_model.eval()
correct = 0total = 0with torch.no_grad():
    for inputs, labels in test_loader:
        logits = cnn_model(inputs)
        predictions = logits.argmax(dim=1, keepdim=True)

        correct += predictions.eq(labels.view_as(predictions)).sum().item()
        total += labels.size(0)

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

# Визуализация работы моделиdata_iter = iter(test_loader)
images, labels = next(data_iter)

sample_img = images[0]
true_label = labels[0]

with torch.no_grad():
    logits = cnn_model(sample_img.unsqueeze(0))
```

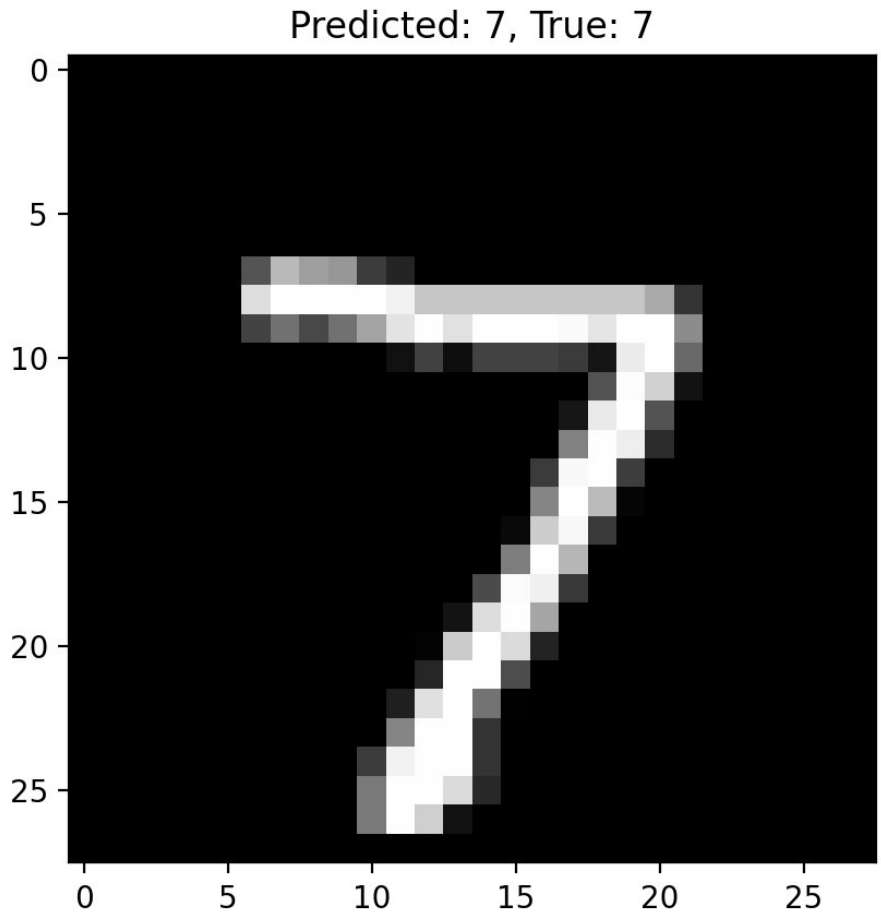
```
    predicted_label = logits.argmax().item()

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

Результат программы:

```
Epoch 1/10, Loss: 0.1843
Epoch 2/10, Loss: 0.0495
Epoch 3/10, Loss: 0.0331
Epoch 4/10, Loss: 0.0252
Epoch 5/10, Loss: 0.0190
Epoch 6/10, Loss: 0.0150
Epoch 7/10, Loss: 0.0108
Epoch 8/10, Loss: 0.0088
Epoch 9/10, Loss: 0.0077
Epoch 10/10, Loss: 0.0060
```




```
Test Accuracy: 99.11%
```



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