МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙГОСУДАРСТВЕННЫЙТЕХНИЧЕСКИЙУНИВЕРСИТЕТ» ФАКУЛЬТЕТ

ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №1

Специальность ИИ-22

Выполнила
Леваневская Н.И.
студентка группы ИИ-22

Проверил
А.А. Крощенко,
ст. преп.
кафедры ИИТ,
«—» ———— 2024 г.

Брест 2024

Вариант 11

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

Задания:

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. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата).
3. Критерии обучения можно подобрать самостоятельно, рекомендуется использовать CrossEntropyLoss.

Выборка	**Размер исходного изображения**	**Оптимизатор**
MNIST	28X28	Adadelta

Конструирование своей модели СНС:

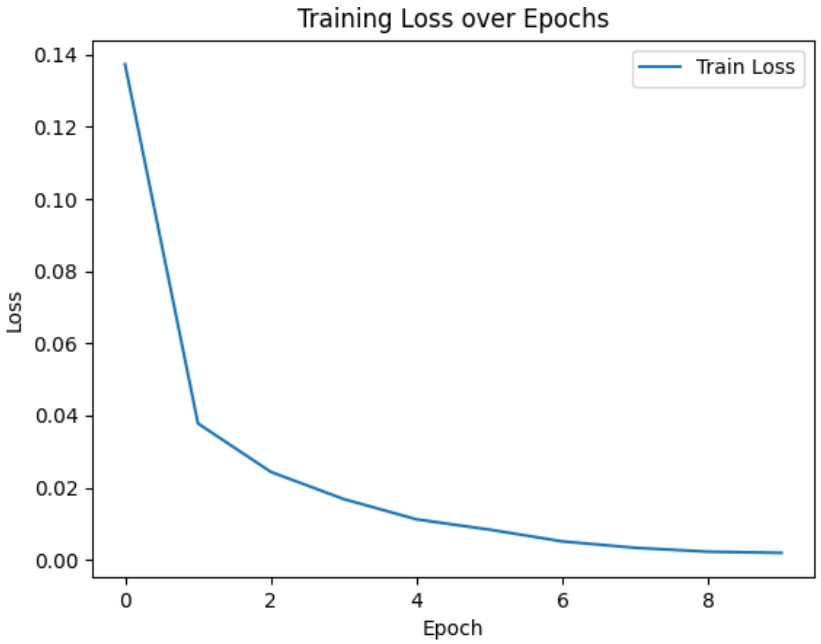
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')
print(torch.version.cuda)

batch_size = 64
learning_rate = 1.0
num_epochs = 10

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,),
(0.5,))])

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

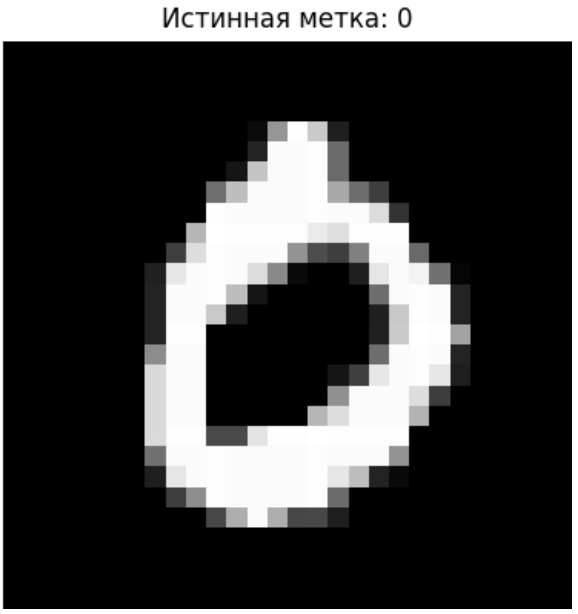
```
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size,
shuffle=False)

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 64 * 7 * 7)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

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

def train(model, train_loader, criterion, optimizer, num_epochs):
    train_losses = []
    for epoch in range(num_epochs):
        running_loss = 0.0
        for i, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        train_losses.append(running_loss / len(train_loader))
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {running_loss /
len(train_loader):.4f}")

    return train_losses

def test(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
```

```
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Accuracy on the test set: {100 * correct / total}%')

def visualize_random_image(model, test_loader, device):

    model.eval()

    images, labels = next(iter(test_loader))
    rand_idx = np.random.randint(0, len(labels))
    image, label = images[rand_idx], labels[rand_idx]

    plt.imshow(image.squeeze(), cmap='gray')
    plt.title(f"Истинная метка: {label.item()}")
    plt.axis('off')
    plt.show()

    with torch.no_grad():
        image = image.unsqueeze(0).to(device)
        output = model(image)
        _, predicted = torch.max(output.data, 1)

    print(f"Предсказанная метка: {predicted.item()}")

train_losses = train(model, train_loader, criterion, optimizer, num_epochs)
test(model, test_loader)

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

visualize_random_image(model, test_loader, device)
```

Результаты обучения:

```
PS C:\Users\Ника> & C:/Users/Ника/AppData/Local/Programs/Python/Python310/python.exe c:/Users/Ника/Desktop/oiis/oiis_lab_1.py
Using device: cuda
11.7
Epoch [1/10], Loss: 0.1500
Epoch [2/10], Loss: 0.0386
Epoch [3/10], Loss: 0.0250
Epoch [4/10], Loss: 0.0164
Epoch [5/10], Loss: 0.0129
Epoch [6/10], Loss: 0.0091
Epoch [7/10], Loss: 0.0063
Epoch [8/10], Loss: 0.0044
Epoch [9/10], Loss: 0.0030
Epoch [10/10], Loss: 0.0017
Accuracy on the test set: 99.2%
Предсказанная метка: 0
```





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