МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №2

Специальность ИИ-21

Выполнил:
Парфеевец И.А.
Студент группы ИИ-21

Проверил:
А. А. Крощенко
доц. кафедры ИИТ

Брест 2024

Цель: научиться осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

Постановка задачи:

1. Для заданной выборки и архитектуры предобученной нейронной организовать процесс обучения НС, предварительно изменив структуру слоев, в соответствии с предложенной выборкой. Использовать тот же оптимизатор, что и в ЛР №1. Построить график изменения ошибки и оценить эффективность обучения на тестовой выборке;

2. Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;

3. Ознакомиться с state-of-the-art результатами для предлагаемых выборок (https://paperswithcode.com/task/image-classification). Сделать выводы о результатах обучения НС из п. 1 и 2;

4. Реализовать визуализацию работы СНС из пункта 1 и пункта 2 (выбор и подачу на архитектуру произвольного изображения с выводом результата);

5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

Ход работы:

В-т	**Выборка**	**Оптимизатор**	**Предобученная архитектура**
2	MNIST	SGD	AlexNet

Была дообучена модифицированная модель alexnet на выборке MNIST.

Код программы:

```
# Лабораторная работа №2 - Конструирование моделей на базе предобученных
нейронных сетей
# Вариант 2: Датасет MNIST, предобученная модель AlexNet, оптимизатор - SGD

# Шаг 1: Импорт необходимых библиотек
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision import models
import matplotlib.pyplot as plt
import numpy as np

# Шаг 2: Загрузка и подготовка данных
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),  # Преобразование в 3 канала
для совместимости с AlexNet
    transforms.Resize((224, 224)),                # Изменение размера для
входа AlexNet
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Нормализация
])
```

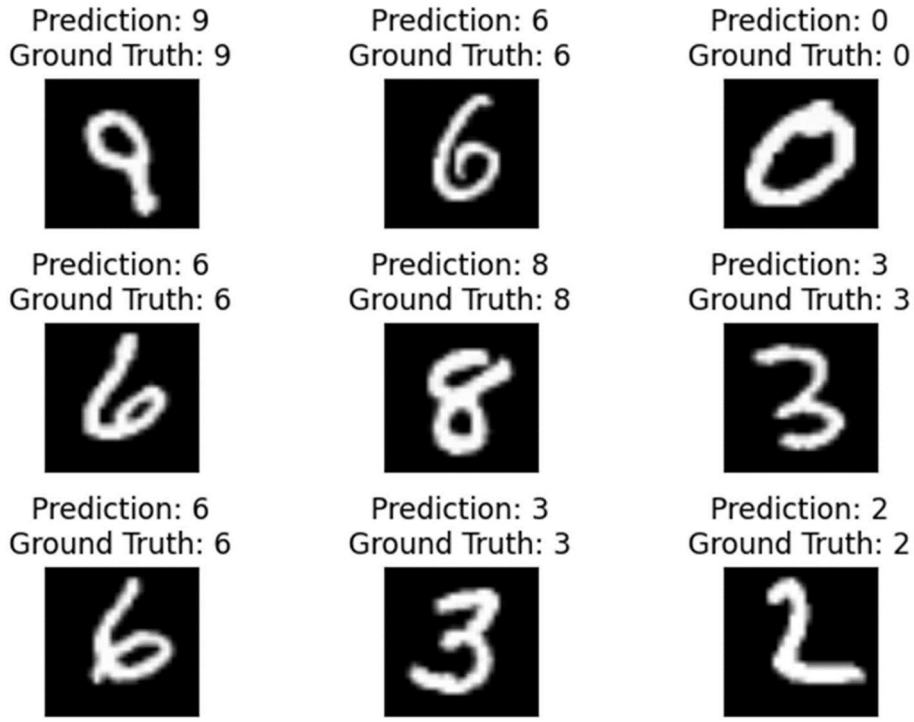
```
# Загрузка данных MNIST
trainset = torchvision.datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False,
download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64,
shuffle=False)

# Шаг 3: Загрузка предобученной модели AlexNet и изменение последнего слоя
net = models.alexnet(pretrained=True)
net.classifier[6] = nn.Linear(4096, 10)  # Изменяем выходной слой для 10
классов MNIST

# Замораживаем слои, кроме последнего слоя
for param in net.features.parameters():
    param.requires_grad = False

# Инициализация функции потерь и оптимизатора
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# Шаг 4: Обучение модели
num_epochs = 10
train_loss_history = []
test_loss_history = []

for epoch in range(num_epochs):
    running_loss = 0.0
    net.train()
    for inputs, labels in trainloader:
        optimizer.zero_grad()           # Обнуление градиентов
        outputs = net(inputs)           # Прямой проход
        loss = criterion(outputs, labels)  # Вычисление потерь
        loss.backward()                 # Обратное распространение
        optimizer.step()                # Шаг оптимизации
        running_loss += loss.item()
    train_loss_history.append(running_loss / len(trainloader))

    # Оценка на тестовой выборке
    net.eval()
    test_loss = 0.0
    with torch.no_grad():
        for inputs, labels in testloader:
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
    test_loss_history.append(test_loss / len(testloader))

    print(f"Epoch {epoch + 1}/{num_epochs}, Train Loss: {train_loss_history[-
1]}, Test Loss: {test_loss_history[-1]}")

# Шаг 5: Визуализация графиков ошибки
plt.plot(train_loss_history, label='Train Loss')
plt.plot(test_loss_history, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Train and Test Loss per Epoch')
plt.show()

# Шаг 6: Визуализация работы предобученной сети на тестовом изображении
def imshow(img):
    img = img / 2 + 0.5  # Денормализация
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
```

```
    plt.show()

# Выбор одного изображения из тестовой выборки и отображение его класса
dataiter = iter(testloader)
images, labels = next(dataiter)

# Вывод изображения
imshow(images[0])
print(f'Actual Label: {labels[0].item()}')

# Предсказание модели
outputs = net(images[0].unsqueeze(0))
_, predicted = torch.max(outputs, 1)
print(f'Predicted Label: {predicted[0].item()}')
```

Результаты:

```
Train Epoch: 8 [53760/60000 (90%)]      Loss: 0.000103
Train Epoch: 8 [54400/60000 (91%)]      Loss: 0.001975
Train Epoch: 8 [55040/60000 (92%)]      Loss: 0.093667
Train Epoch: 8 [55680/60000 (93%)]      Loss: 0.002923
Train Epoch: 8 [56320/60000 (94%)]      Loss: 0.042340
Train Epoch: 8 [56960/60000 (95%)]      Loss: 0.004326
Train Epoch: 8 [57600/60000 (96%)]      Loss: 0.000544
Train Epoch: 8 [58240/60000 (97%)]      Loss: 0.010709
Train Epoch: 8 [58880/60000 (98%)]      Loss: 0.001055
Train Epoch: 8 [59520/60000 (99%)]      Loss: 0.001659

Test set: Avg. loss: 0.0000, Accuracy: 9953/10000 (100%)
```

Тест предсказания нейронной сети на 9-ти экземплярах, выбранных случайным образом из тестовой выборки:



Тест на выбранной из сети интернет цифре и приведенной к нужному формату:

Вывод: научился осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.