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

Лабораторная работа №1
По дисциплине: «Интеллектуальный анализ данных»
Тема: “ Обучение классификаторов средствами библиотеки PyTorch”

Выполнил:
Студент 4 курса
Группы ИИ-24
Капуза Н.А.
Проверила:
Андренко К. В.

Брест 2025

Цель: 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
4	CIFAR-100	32X32	SGD

Ход работы:

Код программы:

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
import pickle
from torch.optim.lr_scheduler import StepLR

# --- 1. Определение улучшенной архитектуры СНС ---
# Эта модель глубже и использует Batch Normalization для стабилизации обучения
# и Dropout для борьбы с переобучением.
class ImprovedCNN(nn.Module):
```

```
    def __init__(self):
        super(ImprovedCNN, self).__init__()
        # Блок 1
        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        # Блок 2
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(2, 2) # Размер 32x32 -> 16x16
        # Блок 3
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        # Блок 4
        self.conv4 = nn.Conv2d(256, 512, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        # Размер после второго пулинга 16x16 -> 8x8

        # Полносвязные слои
        self.fc1 = nn.Linear(512 * 8 * 8, 1024)
        self.dropout = nn.Dropout(0.5) # Dropout с вероятностью 50%
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, 100) # 100 классов CIFAR-100

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = F.relu(self.bn4(self.conv4(x)))

        # Выпрямление данных для полносвязного слоя
        x = x.view(-1, 512 * 8 * 8)

        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# --- 2. Вспомогательная функция для отображения изображений ---
def imshow(img):
    img = img / 2 + 0.5     # денормализация
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()
```

```
# --- 3. Основной блок выполнения кода ---
# Обертка if __name__ == '__main__': обязательна для Windows
# при использовании multiprocessing в DataLoader (num_workers > 0).
if __name__ == '__main__':
    # --- 3.1 Подготовка данных и аугментация ---

    # Аугментация данных - один из ключевых способов улучшить точность.
    # Мы применяем случайные преобразования к обучающим данным.
    transform_train = transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])

    # Для тестовых данных аугментация не применяется!
    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])

    trainset = torchvision.datasets.CIFAR100(root='./data', train=True, download=True,
transform=transform_train)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=128, shuffle=True,
num_workers=2)

    testset = torchvision.datasets.CIFAR100(root='./data', train=False, download=True,
transform=transform_test)
    testloader = torch.utils.data.DataLoader(testset, batch_size=128, shuffle=False,
num_workers=2)

    # --- 3.2 Инициализация модели, оптимизатора и планировщика ---

    # Проверка доступности GPU (CUDA) для ускорения вычислений
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print(f"Обучение будет на устройстве: {device}")

    net = ImprovedCNN()
    net.to(device) # Перемещаем модель на выбранное устройство (CPU или GPU)

    criterion = nn.CrossEntropyLoss()
    # weight_decay добавляет L2-регуляризацию
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-
4)
    # Планировщик для уменьшения скорости обучения со временем
```

```
    scheduler = StepLR(optimizer, step_size=10, gamma=0.1)

    # --- 3.3 Цикл обучения ---
    epochs = 30
    loss_history = []

    print('Начало обучения...')
    for epoch in range(epochs):
        net.train() # Переключаем модель в режим обучения
        running_loss = 0.0
        for i, data in enumerate(trainloader, 0):
            # Перемещаем данные на то же устройство, что и модель
            inputs, labels = data[0].to(device), data[1].to(device)

            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            if i % 200 == 199: # Выводим статистику каждые 200 батчей
                print(f'Эпоха: {epoch + 1}, Батч: {i + 1}, Потери: {running_loss / 200:.3f}')
                loss_history.append(running_loss / 200)
                running_loss = 0.0

        scheduler.step() # Обновляем скорость обучения в конце каждой эпохи

    print('Обучение завершено.')

    # --- 3.4 Оценка модели на тестовых данных ---
    net.eval() # Переключаем модель в режим оценки (отключает Dropout и т.д.)
    correct = 0
    total = 0
    with torch.no_grad(): # Отключаем расчет градиентов для экономии памяти
        for data in testloader:
            images, labels = data[0].to(device), data[1].to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Точность сети на 10000 тестовых изображений: {100 * correct / total:.2f}
%')
```

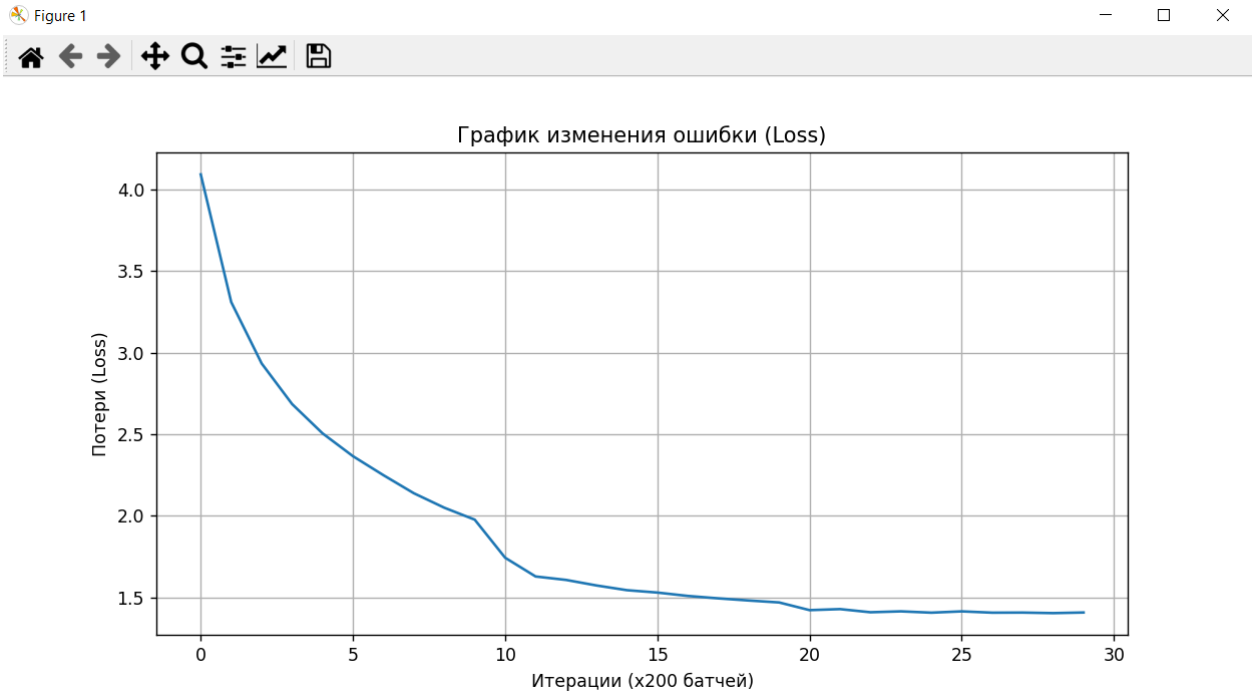
```
    # --- 3.5 Построение графика и визуализация результатов ---

    # Построение графика изменения ошибки
    plt.figure(figsize=(10, 5))
    plt.plot(loss_history)
    plt.title('График изменения ошибки (Loss)')
    plt.xlabel('Итерации (x200 батчей)')
    plt.ylabel('Потери (Loss)')
    plt.grid(True)
    plt.show()

    # Визуализация работы СНС
    meta_path = './data/cifar-100-python/meta'
    try:
        with open(meta_path, 'rb') as infile:
            data = pickle.load(infile, encoding='latin1')
            cifar100_classes = data['fine_label_names']

        dataiter = iter(testloader)
        images, labels = next(dataiter)

        # Перемещаем данные обратно на CPU для работы с matplotlib и numpy
        images_for_show = images.cpu()
        labels_for_show = labels.cpu()

        print("\nИсходные изображения:")
        imshow(torchvision.utils.make_grid(images_for_show[:4]))

        print('Настоящие метки: ', ' '.join(f'{cifar100_classes[labels_for_show[j]]:15s}' for
j in range(4)))

        # Предсказание делаем на GPU (images остаются на device)
        outputs = net(images.to(device))
        _, predicted = torch.max(outputs, 1)

        # Перемещаем предсказания на CPU для вывода
        predicted = predicted.cpu()

        print('Предсказанные метки: ', ' '.join(f'{cifar100_classes[predicted[j]]:15s}' for j in
range(4)))

    except FileNotFoundError:
        print(f"\nНе удалось найти файл с метаданными: {meta_path}")
        print("Визуализация с названиями классов пропущена.")
```

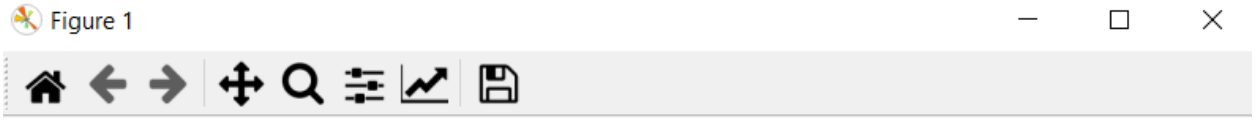
Обучение будет на устройстве: cuda:0
Начало обучения...
Эпоха: 1, Батч: 200, Потери: 4.092
Эпоха: 2, Батч: 200, Потери: 3.310
Эпоха: 3, Батч: 200, Потери: 2.934
Эпоха: 4, Батч: 200, Потери: 2.685
Эпоха: 5, Батч: 200, Потери: 2.506
Эпоха: 6, Батч: 200, Потери: 2.366
Эпоха: 7, Батч: 200, Потери: 2.249
Эпоха: 8, Батч: 200, Потери: 2.139
Эпоха: 9, Батч: 200, Потери: 2.050
Эпоха: 10, Батч: 200, Потери: 1.977
Эпоха: 11, Батч: 200, Потери: 1.744
Эпоха: 12, Батч: 200, Потери: 1.629
Эпоха: 13, Батч: 200, Потери: 1.608
Эпоха: 14, Батч: 200, Потери: 1.574
Эпоха: 15, Батч: 200, Потери: 1.545
Эпоха: 16, Батч: 200, Потери: 1.530
Эпоха: 17, Батч: 200, Потери: 1.510
Эпоха: 18, Батч: 200, Потери: 1.495
Эпоха: 20, Батч: 200, Потери: 1.470
Эпоха: 21, Батч: 200, Потери: 1.423
Эпоха: 22, Батч: 200, Потери: 1.429
Эпоха: 23, Батч: 200, Потери: 1.410
Эпоха: 24, Батч: 200, Потери: 1.415
Эпоха: 25, Батч: 200, Потери: 1.407
Эпоха: 26, Батч: 200, Потери: 1.415
Эпоха: 27, Батч: 200, Потери: 1.407
Эпоха: 28, Батч: 200, Потери: 1.408
Эпоха: 29, Батч: 200, Потери: 1.404
Эпоха: 30, Батч: 200, Потери: 1.408
Обучение завершено.
Точность сети на 10000 тестовых изображений: 60.01 %

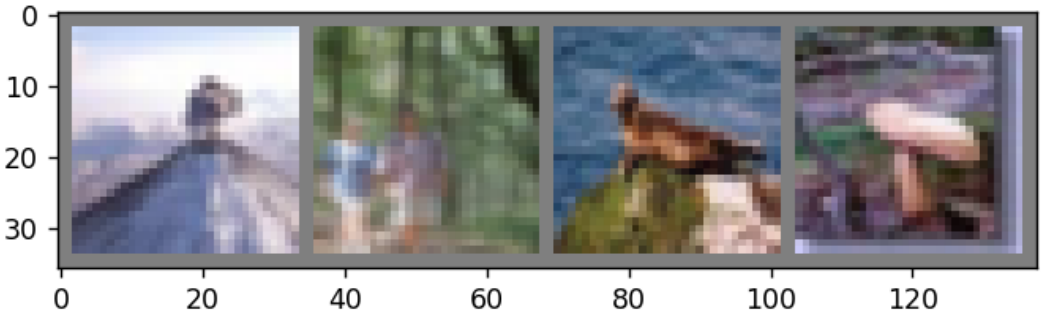
Исходные изображения:

Настоящие метки: mountain forest seal mushroom
Предсказанные метки: man squirrel otter mushroom

График:







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