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

Лабораторная работа №2
По дисциплине: «Обработка изображений в интеллектуальных системах»
Тема: “Конструирование моделей на базе предобученных нейронных сетей”

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

Брест 2025

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

Общее задание

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 (по материалам в сети Интернет). Сделать выводы о результатах обучения НС из п. 1 и 2;

4. Реализовать визуализацию работы предобученной СНС и кастомной (из ЛР 1). Визуализация осуществляется посредством выбора и подачи на сеть произвольного изображения (например, из сети Интернет) с отображением результата классификации;

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

№ варианта	Выборка	Оптимизатор	Предобученная архитектура
4	Fashion-MNIST	SGD	MobileNet v3

Ход работы:

Код программы:

```
import torch
import torchvision
import torchvision.transforms as transforms
# Импортируем классы для загрузки предобученных весов
from torchvision.models import MobileNet_V3_Large_Weights

import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
import pickle


# 1. Вспомогательная функция для отображения изображений
# ВАЖНО: Эта функция изменена для корректного отображения изображений
```

```
# после нормализации под ImageNet.
def imshow(img):
    """Функция для денормализации и отображения изображения."""
    # Стандартные значения нормализации для ImageNet
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])

    npimg = img.numpy()
    npimg = np.transpose(npimg, (1, 2, 0))  # Конвертация из (C, H, W) в (H, W, C)
    npimg = std * npimg + mean  # Денормализация
    npimg = np.clip(npimg, 0, 1)  # Обрезка значений до диапазона [0, 1]

    plt.imshow(npimg)
    plt.show()


# 2. Основной блок выполнения кода
if __name__ == '__main__':
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print(f"Обучение будет на устройстве: {device}")

    # 2.1 Подготовка данных для предобученной модели
    # Предобученные модели требуют определенного формата входных данных.

    # 1. Нормализация с использованием среднего и стд. отклонения ImageNet.
    # 2. Увеличение размера изображений с 32x32 до 224x224.
    transform_pretrained = transforms.Compose([
        transforms.Resize(224),  # Увеличиваем размер изображения
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    # Загрузка датасетов CIFAR-100 с новыми трансформациями
    trainset = torchvision.datasets.CIFAR100(root='./data', train=True, download=True,
transform=transform_pretrained)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True,
                                              num_workers=2)  # Уменьшим batch_size из-за размера картинок

    testset = torchvision.datasets.CIFAR100(root='./data', train=False, download=True,
transform=transform_pretrained)
    testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False, num_workers=2)

    # 2.2 Инициализация модели MobileNet v3

    # Загружаем MobileNet v3 Large с лучшими доступными весами (IMAGENET1K_V2)
    weights = torchvision.models.MobileNet_V3_Large_Weights.IMAGENET1K_V2
    net = torchvision.models.mobilenet_v3_large(weights=weights)

    # Заменяем последний слой (классификатор)
    # У MobileNet v3 классификатор - это net.classifier
    num_ftrs = net.classifier[-1].in_features
```

```
net.classifier[-1] = nn.Linear(num_ftrs, 100)  # Заменяем на слой для 100 классов

net.to(device)

# 2.3 Оптимизатор и критерий потерь
criterion = nn.CrossEntropyLoss()

# Используем тот же оптимизатор, что и в ЛР №1, как требует задание
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-4)

# Обучать будем меньше эпох, так как модель уже много знает
epochs = 15
loss_history = []

# 2.4 Цикл обучения (дообучения)
print('Начало дообучения (используется предобученная MobileNet v3)...')
for epoch in range(epochs):
    net.train()
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        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()

    epoch_loss = running_loss / len(trainloader)
    loss_history.append(epoch_loss)
    print(f'Эпоха: {epoch + 1}/{epochs}, Потери: {epoch_loss:.4f}')

print('Обучение завершено.')

# 2.5 Оценка модели и визуализация
net.eval()
correct = 0
total = 0
with torch.no_grad():
    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()

accuracy = 100 * correct / total
print(f'\nИтоговая точность сети на 10000 тестовых изображений: {accuracy:.2f} %')

# Построение графика
plt.figure(figsize=(10, 5))
```

```
    plt.plot(loss_history)
    plt.title('График изменения ошибки (Loss) для MobileNet v3')
    plt.xlabel('Эпоха')
    plt.ylabel('Средние потери за эпоху')
    plt.grid(True)
    plt.show()

    # Визуализация предсказаний
    # (код аналогичен предыдущим работам)
    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)

        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)))

        outputs = net(images.to(device))
        _, predicted = torch.max(outputs, 1)
        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.")
```

Графики:

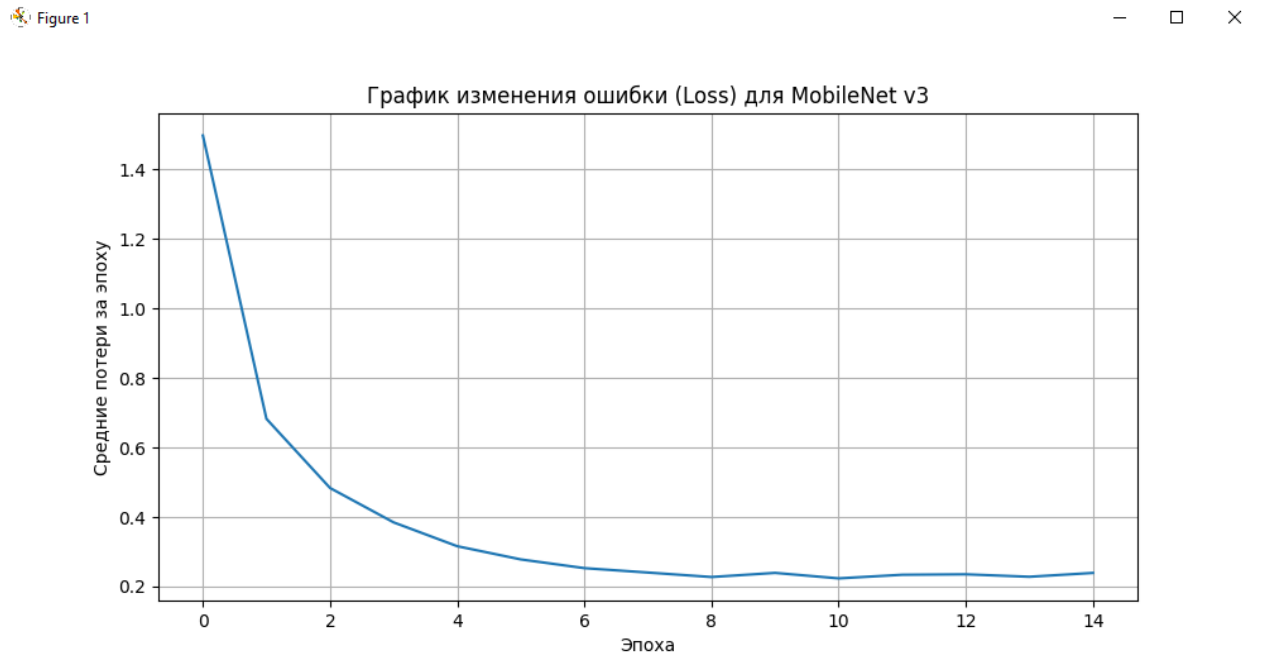
Figure 1
График изменения ошибки (Loss) для MobileNet v3
Средние потери за эпоху
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8
10
12
14
Эпоха


Figure 1
0
100
200
0
200
400
600
800

Результат:

Начало дообучения (используется предобученная MobileNet v3)...
Эпоха: 1/15, Потери: 1.4983
Эпоха: 2/15, Потери: 0.6828
Эпоха: 3/15, Потери: 0.4838
Эпоха: 4/15, Потери: 0.3848
Эпоха: 5/15, Потери: 0.3162
Эпоха: 6/15, Потери: 0.2782
Эпоха: 7/15, Потери: 0.2531
Эпоха: 8/15, Потери: 0.2404
Эпоха: 9/15, Потери: 0.2275
Эпоха: 10/15, Потери: 0.2394
Эпоха: 11/15, Потери: 0.2236
Эпоха: 12/15, Потери: 0.2340
Эпоха: 13/15, Потери: 0.2353
Эпоха: 14/15, Потери: 0.2282
Эпоха: 15/15, Потери: 0.2393
Обучение завершено.

Итоговая точность сети на 10000 тестовых изображений: 69.85 %

Исходные изображения:
Настоящие метки: mountain forest seal mushroom
Предсказанные метки: mountain forest beaver mushroom
Как видим результат обучения во второй лабораторной работе лучше, чем в первой (69.85 %>60.01%)

Вывод: осуществил обучение НС, сконструированных на базе предобученных архитектур НС.