`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе 2

Специальность ИИ-23

Выполнил:

Гавришук В.Р.

Студент группы ИИ-23

Проверил:

Андренко К. В.
Преподаватель-стажёр
Кафедры ИИТ,
«___» _____2025 г.

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.

Задание по вариантам

№ варианта	Выборка	Оптимизатор	Предобученная архитектура
4	CIFAR-100	SGD	MobileNet v3

Ход работы:

```
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
from torchvision import models
import warnings

warnings.filterwarnings('ignore')

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Используемое устройство: {device}")

batch_size = 64
learning_rate = 0.01
momentum = 0.9
num_epochs = 10

transform_train = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomHorizontalFlip(),
```

```
    transforms.RandomCrop(224, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5071, 0.4867, 0.4408],
                         std=[0.2675, 0.2565, 0.2761])
])

transform_test = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5071, 0.4867, 0.4408],
                         std=[0.2675, 0.2565, 0.2761])
])


def create_modified_mobilenet_v3(num_classes=100):
    model = models.mobilenet_v3_large(pretrained=True)

    for param in model.parameters():
        param.requires_grad = False

    in_features = model.classifier[3].in_features
    model.classifier[3] = nn.Linear(in_features, num_classes)

    for param in model.classifier.parameters():
        param.requires_grad = True

    return model


def train_model(model, train_loader, test_loader, criterion, optimizer, num_epochs):
    train_losses = []
    test_accuracies = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0

        for i, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

            if (i + 1) % 100 == 0:
                print(f'Epoch [{epoch + 1}/{num_epochs}], Step [{i + 1}/{len(train_loader)}],
Loss: {loss.item():.4f}')

        epoch_loss = running_loss / len(train_loader)
        train_losses.append(epoch_loss)

        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()

        accuracy = 100 * correct / total
        test_accuracies.append(accuracy)

        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {epoch_loss:.4f}, Test Accuracy:
{accuracy:.2f}%')

    return train_losses, test_accuracies


def evaluate_model(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()

    accuracy = 100 * correct / total
    print(f'Финальная точность на тестовой выборке: {accuracy:.2f}%')
    return accuracy


def visualize_prediction(model, test_dataset, image_path=None):
    if image_path is None:
        random_index = np.random.randint(0, len(test_dataset))
        image, true_label = test_dataset[random_index]
        image = image.unsqueeze(0).to(device)
    else:
        from PIL import Image
        image = Image.open(image_path).convert('RGB')
        image = transform_test(image).unsqueeze(0).to(device)
        true_label = None

    model.eval()
    with torch.no_grad():
        outputs = model(image)
        probabilities = torch.nn.functional.softmax(outputs, dim=1)
        confidence, predicted = torch.max(probabilities, 1)

    classes = test_dataset.classes

    image_to_show = image.squeeze(0).cpu()
    image_to_show = image_to_show.permute(1, 2, 0)

    mean = torch.tensor([0.5071, 0.4867, 0.4408])
    std = torch.tensor([0.2675, 0.2565, 0.2761])
    image_to_show = image_to_show * std + mean
    image_to_show = torch.clamp(image_to_show, 0, 1)
```

```
    plt.figure(figsize=(8, 6))
    plt.imshow(image_to_show)

    if true_label is not None:
        plt.title(f'Предсказание: {classes[predicted.item()]} ({confidence.item():.2f})\n'
                  f'Истинный класс: {classes[true_label]}')
    else:
        plt.title(f'Предсказание: {classes[predicted.item()]} (уверенность:
{confidence.item():.2f})')

    plt.axis('off')
    plt.show()

    top5_prob, top5_catid = torch.topk(probabilities, 5)
    print("Топ-5 предсказаний:")
    for i in range(top5_prob.size(1)):
        print(f"{classes[top5_catid[0][i].item()]}: {top5_prob[0][i].item():.4f}")


def main():
    print("Загрузка данных CIFAR-100...")

    train_dataset = torchvision.datasets.CIFAR100(
        root='./data', train=True, download=True, transform=transform_train)
    test_dataset = torchvision.datasets.CIFAR100(
        root='./data', train=False, download=True, transform=transform_test)

    train_loader = torch.utils.data.DataLoader(
        train_dataset, batch_size=batch_size, shuffle=True, num_workers=0)  # num_workers=0 для
Windows
    test_loader = torch.utils.data.DataLoader(
        test_dataset, batch_size=batch_size, shuffle=False, num_workers=0)

    print(f"Размер обучающей выборки: {len(train_dataset)}")
    print(f"Размер тестовой выборки: {len(test_dataset)}")
    print(f"Количество классов: {len(train_dataset.classes)}")

    print("Создание модели MobileNet v3...")
    model = create_modified_mobilenet_v3(num_classes=100)
    model = model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(
        filter(lambda p: p.requires_grad, model.parameters()),
        lr=learning_rate,
        momentum=momentum
    )

    print("Начало обучения...")
    train_losses, test_accuracies = train_model(
        model, train_loader, test_loader, criterion, optimizer, num_epochs
    )

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 2, 1)
    plt.plot(range(1, num_epochs + 1), train_losses, 'b-', linewidth=2)
    plt.title('Изменение ошибки во время обучения')
    plt.xlabel('Эпоха')
    plt.ylabel('Ошибка')
    plt.grid(True)
```

`

```
    plt.subplot(1, 2, 2)
    plt.plot(range(1, num_epochs + 1), test_accuracies, 'r-', linewidth=2)
    plt.title('Точность на тестовой выборке')
    plt.xlabel('Эпоха')
    plt.ylabel('Точность (%)')
    plt.grid(True)

    plt.tight_layout()
    plt.show()

    final_accuracy = evaluate_model(model, test_loader)

    print("\nВизуализация работы сети на случайном изображении:")
    visualize_prediction(model, test_dataset)

    torch.save(model.state_dict(), 'mobilenet_v3_cifar100.pth')
    print("Модель сохранена как 'mobilenet_v3_cifar100.pth'")

    print("\nДополнительная визуализация на 3 случайных изображениях:")
    for i in range(3):
        visualize_prediction(model, test_dataset)


if __name__ == '__main__':
    main()
```

Результат работы программы:

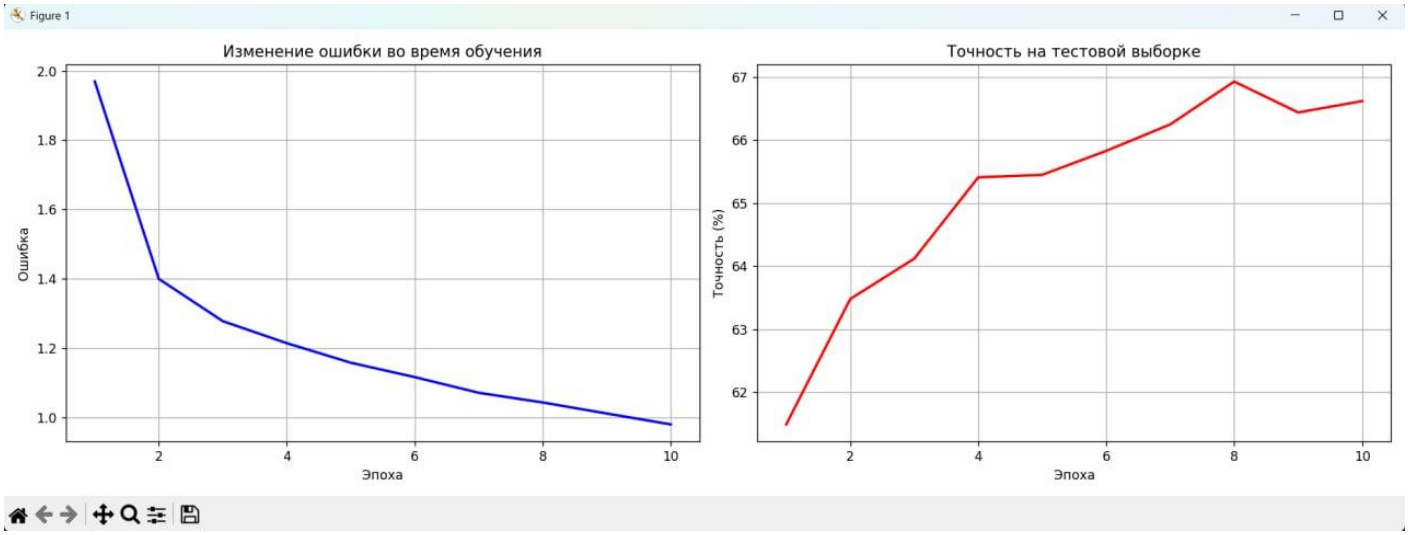
```
Начало обучения...
Epoch [1/10], Step [100/782], Loss: 2.6195
Epoch [1/10], Step [200/782], Loss: 1.7324
Epoch [1/10], Step [300/782], Loss: 1.6896
Epoch [1/10], Step [400/782], Loss: 1.4043
Epoch [1/10], Step [500/782], Loss: 1.5512
Epoch [1/10], Step [600/782], Loss: 1.2411
Epoch [1/10], Step [700/782], Loss: 1.2814
Epoch [1/10], Loss: 1.9690, Test Accuracy: 61.49%
Epoch [2/10], Step [100/782], Loss: 1.2555
Epoch [2/10], Step [200/782], Loss: 1.6513
Epoch [2/10], Step [300/782], Loss: 1.3655
Epoch [2/10], Step [400/782], Loss: 1.7121
Epoch [2/10], Step [500/782], Loss: 1.4667
Epoch [2/10], Step [600/782], Loss: 1.4659
Epoch [2/10], Step [700/782], Loss: 1.4300
Epoch [2/10], Loss: 1.3995, Test Accuracy: 63.48%
Epoch [3/10], Step [100/782], Loss: 1.0951
Epoch [3/10], Step [200/782], Loss: 1.3272
Epoch [3/10], Step [300/782], Loss: 1.1109
Epoch [3/10], Step [400/782], Loss: 0.9960
Epoch [3/10], Step [500/782], Loss: 1.2467
```

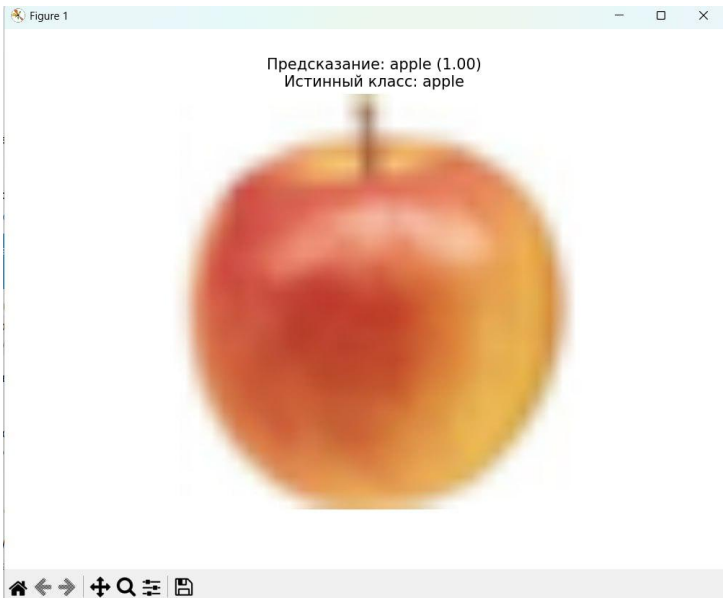
`

```
Epoch [3/10], Step [600/782], Loss: 0.9282
Epoch [3/10], Step [700/782], Loss: 1.3590
Epoch [3/10], Loss: 1.2775, Test Accuracy: 64.12%
Epoch [4/10], Step [100/782], Loss: 0.8658
Epoch [4/10], Step [200/782], Loss: 1.0138
Epoch [4/10], Step [300/782], Loss: 1.0383
Epoch [4/10], Step [400/782], Loss: 1.0455
Epoch [4/10], Step [500/782], Loss: 1.0605
Epoch [4/10], Step [600/782], Loss: 1.2260
Epoch [4/10], Step [700/782], Loss: 1.2730
Epoch [4/10], Loss: 1.2139, Test Accuracy: 65.41%
Epoch [5/10], Step [100/782], Loss: 1.1264
Epoch [5/10], Step [200/782], Loss: 1.2398
Epoch [5/10], Step [300/782], Loss: 1.3198
Epoch [5/10], Step [400/782], Loss: 1.0546
Epoch [5/10], Step [500/782], Loss: 1.3158
Epoch [5/10], Step [600/782], Loss: 0.9492
Epoch [5/10], Step [700/782], Loss: 1.0474
Epoch [5/10], Loss: 1.1575, Test Accuracy: 65.45%
Epoch [6/10], Step [100/782], Loss: 0.7920
Epoch [6/10], Step [200/782], Loss: 1.0413
Epoch [6/10], Step [300/782], Loss: 1.2118
Epoch [6/10], Step [400/782], Loss: 1.5412
Epoch [6/10], Step [500/782], Loss: 1.1261
Epoch [6/10], Step [600/782], Loss: 1.0681
Epoch [6/10], Step [700/782], Loss: 1.0386
Epoch [6/10], Loss: 1.1160, Test Accuracy: 65.83%
Epoch [7/10], Step [100/782], Loss: 1.1093
Epoch [7/10], Step [200/782], Loss: 0.9755
Epoch [7/10], Step [300/782], Loss: 0.8134
Epoch [7/10], Step [400/782], Loss: 1.1651
Epoch [7/10], Step [500/782], Loss: 1.1039
Epoch [7/10], Step [600/782], Loss: 1.2134
Epoch [7/10], Step [700/782], Loss: 0.9798
Epoch [7/10], Loss: 1.0707, Test Accuracy: 66.25%
Epoch [8/10], Step [100/782], Loss: 1.0435
Epoch [8/10], Step [200/782], Loss: 1.3462
Epoch [8/10], Step [300/782], Loss: 0.8159
Epoch [8/10], Step [400/782], Loss: 0.9329
Epoch [8/10], Step [500/782], Loss: 1.3461
Epoch [8/10], Step [600/782], Loss: 1.2480
Epoch [8/10], Step [700/782], Loss: 0.9569
Epoch [8/10], Loss: 1.0428, Test Accuracy: 66.93%
Epoch [9/10], Step [100/782], Loss: 1.1147
```

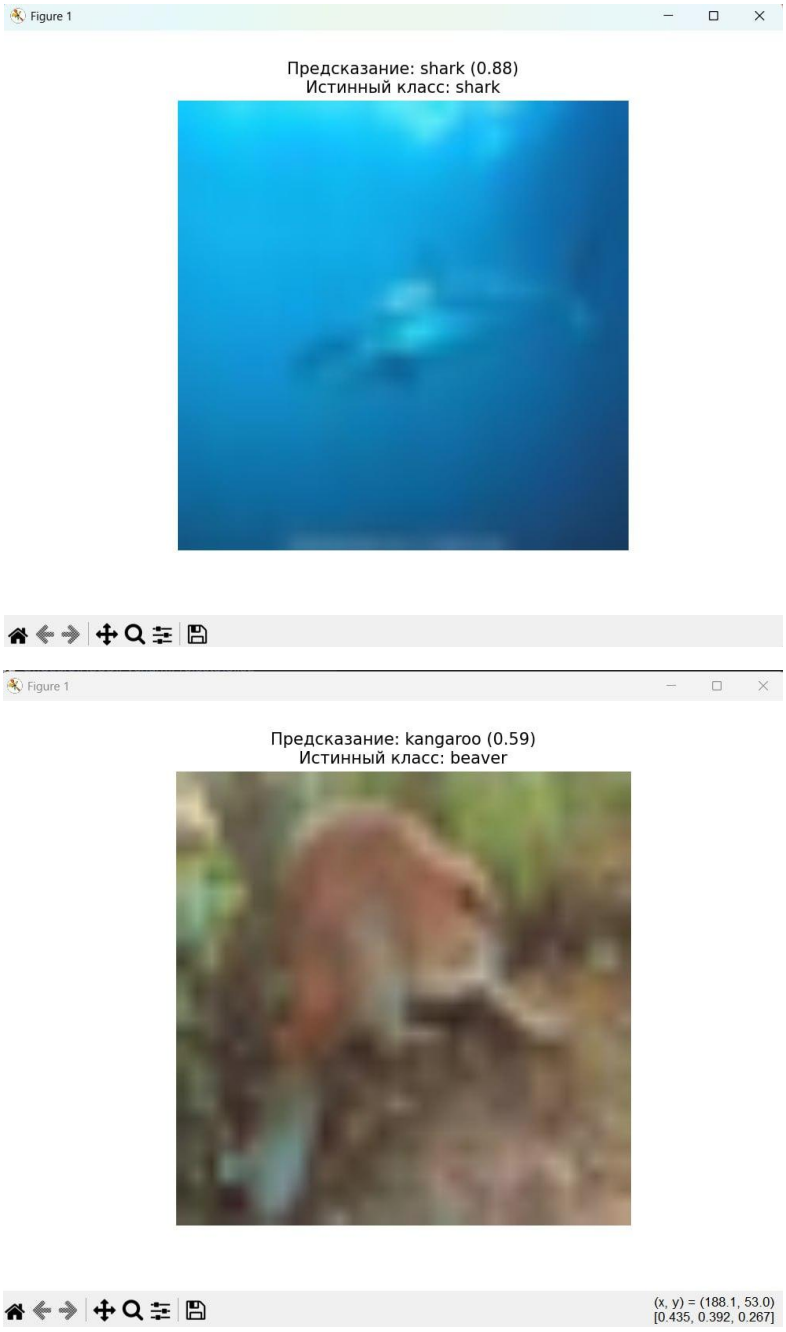
`

```
Epoch [9/10], Step [200/782], Loss: 1.1972
Epoch [9/10], Step [300/782], Loss: 1.1131
Epoch [9/10], Step [400/782], Loss: 1.0584
Epoch [9/10], Step [500/782], Loss: 1.0661
Epoch [9/10], Step [600/782], Loss: 1.0348
Epoch [9/10], Step [700/782], Loss: 1.4362
Epoch [9/10], Loss: 1.0109, Test Accuracy: 66.44%
Epoch [10/10], Step [100/782], Loss: 1.0012
Epoch [10/10], Step [200/782], Loss: 0.9766
Epoch [10/10], Step [300/782], Loss: 0.8530
Epoch [10/10], Step [400/782], Loss: 0.6413
Epoch [10/10], Step [500/782], Loss: 1.0052
Epoch [10/10], Step [600/782], Loss: 1.0237
Epoch [10/10], Step [700/782], Loss: 1.1355
Epoch [10/10], Loss: 0.9795, Test Accuracy: 66.62%
```





`



Сравнение с лр.1:

Кастомная архитектура из лр1 - точность составляет 53%

Кастомная архитектура	MobileNet v3
Обучается с нуля и использует случайные начальные веса	Модель уже обучена выделять полезные признаки
Меньшая глубина – 3 сверточных слоя	Глубокие слои, которые эффективно извлекают иерархические признаки
Базовая архитектура – обычные convolution+batch+pooling слои	Архитектура с inverted residuals, SE blocks, h-swish активациями

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