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»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Лящук А. В.

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.

Задание по вариантам

№ варианта	Выборка	Оптимизатор	Предобученная архитектура
10	STL-10	Adam	SqueezeNet1.1

Код:

```
import os
import time
from datetime import datetime
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
from torchvision import transforms, models

# ======================
```

```
# ПАРАМЕТРЫ ОБУЧЕНИЯ
# ======================
# Основные параметры
EPOCHS = 20
BATCH_SIZE = 64
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-4
FREEZE_BACKBONE = False

# Пути
SAVE_DIR = 'checkpoints_stl10_squeezenet'
RESUME_TRAINING = False
RESUME_PATH = None
VISUALIZE_IMAGE = None  # Путь к изображению для предсказания

# Настройки данных
NUM_WORKERS = 4
INPUT_SIZE = 224
LOG_INTERVAL = 50  # Интервал логирования (в батчах)


# ======================
# МОДЕЛЬ И ФУНКЦИИ
# ======================
def create_squeezenet1_1(num_classes=10, freeze_backbone=False):
    """
    Создает модель SqueezeNet 1.1 с предобученными весами ImageNet
    и заменяет классификатор для нужного количества классов:cite[7]
    """
    # Загрузка предобученной модели SqueezeNet 1.1:cite[7]
    model =
models.squeezenet1_1(weights=models.SqueezeNet1_1_Weights.IMAGENET1K_V1)

    # Замена последнего сверточного слоя в классификаторе:cite[7]
    # Исходный слой: Conv2d(512, 1000, kernel_size=1)
    model.classifier[1] = nn.Conv2d(512, num_classes, kernel_size=1)

    # Инициализация нового слоя
    nn.init.normal_(model.classifier[1].weight, mean=0.0, std=0.01)

    if freeze_backbone:
        # Заморозка всех параметров, кроме классификатора:cite[4]:cite[9]
        for name, param in model.named_parameters():
            if 'classifier' not in name:
                param.requires_grad = False

    return model


def evaluate(model, loader, device, criterion):
    """Функция оценки модели на валидационной выборке"""
    model.eval()
    running_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for x, y in loader:
            x, y = x.to(device), y.to(device)
            out = model(x)
            loss = criterion(out, y)
            running_loss += loss.item() * x.size(0)
            preds = out.argmax(dim=1)
            correct += (preds == y).sum().item()
```

```
            total += x.size(0)
    return running_loss / total, 100.0 * correct / total


def predict_image(path, model, device, transform, input_size, classes):
    """Функция для предсказания на одном изображении"""
    img = Image.open(path).convert('RGB')
    img_resized = img.resize((input_size, input_size))
    x = transform(img_resized).unsqueeze(0).to(device)
    model.eval()
    with torch.no_grad():
        logits = model(x)
        probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
        pred = int(np.argmax(probs))
    return img, pred, probs


# ======================
# ОСНОВНАЯ ФУНКЦИЯ
# ======================
def main():
    # Настройка устройства
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    print(f'Using device: {device}')
    if torch.cuda.is_available():
        print(f'GPU: {torch.cuda.get_device_name(0)}')

    # Создание директории для сохранения
    os.makedirs(SAVE_DIR, exist_ok=True)

    # Параметры нормализации для ImageNet:cite[7]
    imagenet_mean = (0.485, 0.456, 0.406)
    imagenet_std = (0.229, 0.224, 0.225)

    # Преобразования для обучающей выборки:cite[1]
    train_transform = transforms.Compose([
        transforms.RandomResizedCrop(INPUT_SIZE, scale=(0.8, 1.0)),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
    ])

    # Преобразования для тестовой выборки:cite[1]
    test_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(INPUT_SIZE),
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
    ])

    # Загрузка датасета STL-10:cite[1]:cite[5]
    print("Loading STL-10 dataset...")
    train_set = torchvision.datasets.STL10(
        root='./data',
        split='train',
        download=True,
        transform=train_transform
    )
    test_set = torchvision.datasets.STL10(
```

```
        root='./data',
        split='test',
        download=True,
        transform=test_transform
    )

    # Создание загрузчиков данных:cite[8]
    train_loader = DataLoader(
        train_set,
        batch_size=BATCH_SIZE,
        shuffle=True,
        num_workers=NUM_WORKERS,
        pin_memory=True
    )
    test_loader = DataLoader(
        test_set,
        batch_size=BATCH_SIZE,
        shuffle=False,
        num_workers=NUM_WORKERS,
        pin_memory=True
    )

    # Классы STL-10:cite[1]
    classes = [
        'airplane', 'bird', 'car', 'cat', 'deer',
        'dog', 'horse', 'monkey', 'ship', 'truck'
    ]

    # Создание модели
    print("Creating SqueezeNet 1.1 model...")
    model = create_squeezenet1_1(
        num_classes=10,
        freeze_backbone=FREEZE_BACKBONE
    ).to(device)

    # Вывод информации о модели
    print(f"Model: SqueezeNet 1.1")
    print(f"Trainable parameters: {sum(p.numel() for p in model.parameters()
if p.requires_grad)}")
    print(f"Total parameters: {sum(p.numel() for p in model.parameters())}")

    # Функция потерь и оптимизатор Adam:cite[6]:cite[10]
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(
        [p for p in model.parameters() if p.requires_grad],
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY,
        betas=(0.9, 0.999)  # Стандартные параметры для Adam:cite[6]
    )
    scheduler = optim.lr_scheduler.StepLR(
        optimizer,
        step_size=max(5, EPOCHS // 2),
        gamma=0.1
    )

    # Возобновление обучения (если нужно)
    start_epoch = 1
    history = {'train_loss': [], 'test_loss': [], 'test_acc': []}

    if RESUME_TRAINING or RESUME_PATH:
        ckpt_path = RESUME_PATH
```

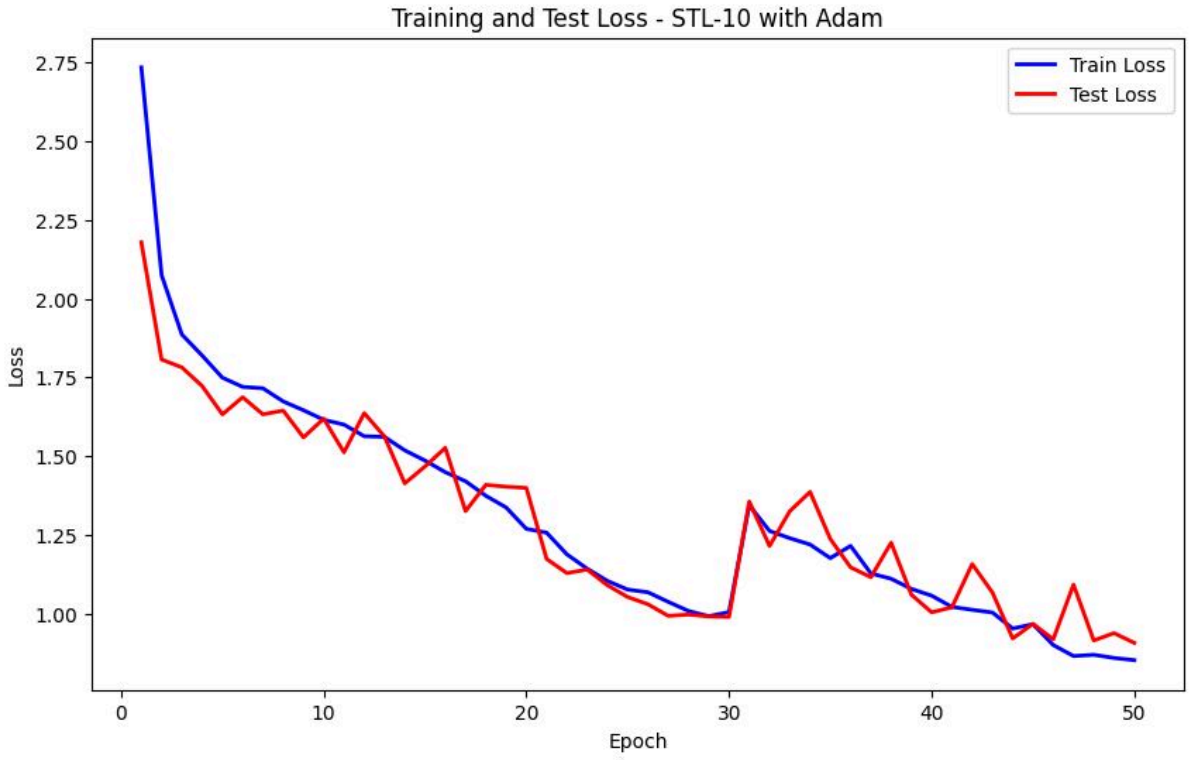
```
        if RESUME_TRAINING and ckpt_path is None:
            files = [f for f in os.listdir(SAVE_DIR) if f.endswith('.pth')]
            if files:
                files = sorted(files, key=lambda x:
int(x.split('epoch')[-1].split('.')[0]))
                ckpt_path = os.path.join(SAVE_DIR, files[-1])

        if ckpt_path and os.path.isfile(ckpt_path):
            print(f"Loading checkpoint {ckpt_path} ...")
            ckpt = torch.load(ckpt_path, map_location=device)
            model.load_state_dict(ckpt['model_state'])
            optimizer.load_state_dict(ckpt['optimizer_state'])
            start_epoch = ckpt['epoch'] + 1
            if 'history' in ckpt:
                history = ckpt['history']
            print(f"Resumed from epoch {ckpt['epoch']}")
        else:
            print("⚠️ Checkpoint not found, starting from scratch")

    # Обучение модели
    print(f"Starting training for {EPOCHS} epochs...")
    global_start_time = time.time()

    for epoch in range(start_epoch, EPOCHS + 1):
        epoch_start_time = time.time()
        model.train()
        running_loss = 0.0

        # Логирование времени для батчей
        batch_times = []
        batch_start_time = time.time()

        for batch_idx, (xb, yb) in enumerate(train_loader, 1):
            xb, yb = xb.to(device), yb.to(device)
            optimizer.zero_grad()

            # Прямой проход
            logits = model(xb)
            loss = criterion(logits, yb)

            # Обратный проход
            loss.backward()
            optimizer.step()

            # Статистика
            running_loss += loss.item() * xb.size(0)

            # Логирование каждые LOG_INTERVAL батчей
            if batch_idx % LOG_INTERVAL == 0:
                batch_time = time.time() - batch_start_time
                batch_times.append(batch_time)

                # Прогноз оставшегося времени эпохи
                avg_batch_time = np.mean(batch_times[-10:]) if
len(batch_times) > 10 else batch_time
                remaining_batches = len(train_loader) - batch_idx
                eta_seconds = avg_batch_time * remaining_batches
                eta_str = time.strftime("%H:%M:%S", time.gmtime(eta_seconds))

                current_time = datetime.now().strftime("%H:%M:%S")
                print(f"[{current_time}] Epoch {epoch}/{EPOCHS} | "
```

```
                        f"Batch {batch_idx}/{len(train_loader)} | "
                        f"Loss: {loss.item():.6f} | "
                        f"Batch Time: {batch_time:.3f}s | "
                        f"ETA: {eta_str}")

                batch_start_time = time.time()

        # Статистика эпохи
        epoch_time = time.time() - epoch_start_time
        train_loss = running_loss / len(train_loader.dataset)
        test_loss, test_acc = evaluate(model, test_loader, device, criterion)

        history['train_loss'].append(train_loss)
        history['test_loss'].append(test_loss)
        history['test_acc'].append(test_acc)
        scheduler.step()

        print(f"Epoch {epoch}/{EPOCHS} completed in {epoch_time:.2f}s | "
              f"TrainLoss {train_loss:.4f} | TestLoss {test_loss:.4f} | "
              f"TestAcc {test_acc:.2f}%")

        # Сохранение контрольной точки
        checkpoint_path = os.path.join(SAVE_DIR,
f'squeezenet_epoch{epoch}.pth')
        torch.save({
            'epoch': epoch,
            'model_state': model.state_dict(),
            'optimizer_state': optimizer.state_dict(),
            'history': history,
            'test_acc': test_acc
        }, checkpoint_path)
        print(f"Checkpoint saved: {checkpoint_path}")

    # Итоговое время обучения
    total_time = time.time() - global_start_time
    print(f'Training finished in {total_time / 60:.2f} minutes')

    # Построение графиков обучения
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(range(1, len(history['train_loss']) + 1), history['train_loss'],
label='train')
    plt.plot(range(1, len(history['test_loss']) + 1), history['test_loss'],
label='test', linestyle='--')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Test Loss')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(range(1, len(history['test_acc']) + 1), history['test_acc'],
label='test acc', color='green')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.title('Test Accuracy')
    plt.legend()

    plt.tight_layout()
    history_path = os.path.join(SAVE_DIR, 'training_history.png')
    plt.savefig(history_path, dpi=150)
    print(f'Saved history plot to {history_path}')
```

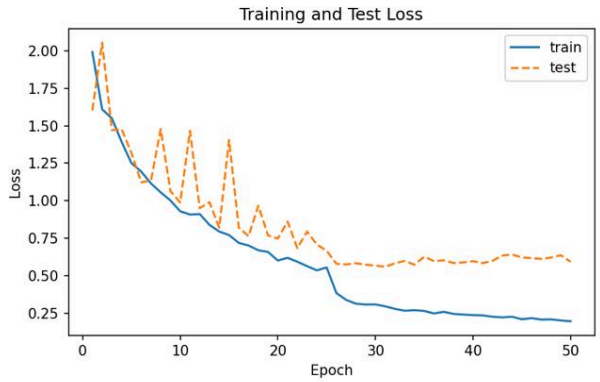
```
    # Визуализация (если указан путь к изображению)
    if VISUALIZE_IMAGE and os.path.exists(VISUALIZE_IMAGE):
        print(f"Making prediction for image: {VISUALIZE_IMAGE}")
        img, pred_idx, probs = predict_image(
            VISUALIZE_IMAGE,
            model,
            device,
            test_transform,
            INPUT_SIZE,
            classes
        )
        plt.figure(figsize=(6, 6))
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Prediction: {classes[pred_idx]} ({probs[pred_idx] *
100:.1f}%)')
        pred_path = os.path.join(SAVE_DIR, 'prediction.png')
        plt.savefig(pred_path, dpi=150, bbox_inches='tight')
        print(f'Saved prediction visualization to {pred_path}')
    elif VISUALIZE_IMAGE:
        print(f"Image {VISUALIZE_IMAGE} not found, skipping prediction.")


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

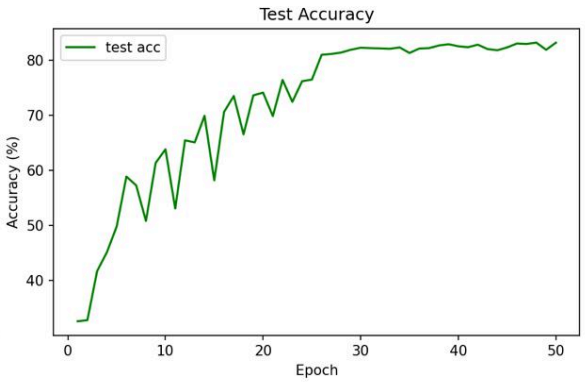
Вывод:

Л. Р. №1



Л. Р. № 2





State-of-art:

Family	Architecture	ImageNet Acc. (%)	Depth	MFLOPS	Skip-conn.	Branch	1×1 conv.	Batch norm
VGG	VGG-11 [23]	69.020	11	7637				
	VGG-16 [23]	71.592	16	15517				
	VGG-16_bn [23]	73.360	16	15544				✓
	VGG-19 [23]	72.376	19	19682				
ResNet	ResNet-18 [24]	69.758	18	1827	✓			✓
	ResNet-50 [24]	76.130	50	4143	✓		✓	✓
	ResNeXt-50 32x4d [60]	77.618	50	4298	✓	✓	✓	✓
Inception	GoogLeNet [25]	69.778	22	1516		✓	✓	✓
	InceptionNet v3 [61]	77.294	49	2850		✓	✓	✓
EfficientNet	EfficientNet_B0 [62]	77.692	81	407	✓			✓
	EfficientNet_B7 [62]	78.642	271	5308	✓			✓
Other	AlexNet [22]	56.522	8	717				
	DenseNet-121 [59]	74.434	121	2899	✓		✓	✓
	SqueezeNet 1.1 [63]	58.178	18	360		✓	✓	

Ссылка на статью

Вывод: осуществила обучение НС, сконструированных на базе предобученных архитектур НС