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

Супрунович И. С.

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
17	Fashion-MNIST	RMSprop	DenseNet121

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
# ======================
EPOCHS = 20
BATCH_SIZE = 64
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-4
FREEZE_BACKBONE = False

# Пути
SAVE_DIR = 'checkpoints_fashion_mnist'
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_densenet121(num_classes=10, freeze_backbone=False):
    model =
models.densenet121(weights=models.DenseNet121_Weights.IMAGENET1K_V1)

    # Замена классификатора
    in_features = model.classifier.in_features
    model.classifier = nn.Linear(in_features, num_classes)

    if freeze_backbone:
        for name, param in model.named_parameters():
            if 'classifier' not in name:
                param.requires_grad = False
    return model


def evaluate(model, loader, device, criterion):
    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):
    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
    imagenet_mean = (0.485, 0.456, 0.406)
    imagenet_std = (0.229, 0.224, 0.225)

    # Преобразования с конвертацией в 3 канала
    train_transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=3),
        transforms.RandomResizedCrop(INPUT_SIZE, scale=(0.8, 1.0)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
    ])

    test_transform = transforms.Compose([
        transforms.Grayscale(num_output_channels=3),
        transforms.Resize((INPUT_SIZE, INPUT_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
    ])

    # Загрузка данных
    print("Loading Fashion-MNIST dataset...")
    train_set = torchvision.datasets.FashionMNIST(
        root='./data',
        train=True,
        download=True,
        transform=train_transform
    )
    test_set = torchvision.datasets.FashionMNIST(
        root='./data',
        train=False,
        download=True,
        transform=test_transform
    )

    # Создание загрузчиков данных
    train_loader = DataLoader(
        train_set,
        batch_size=BATCH_SIZE,
        shuffle=True,
        num_workers=NUM_WORKERS,
        pin_memory=True
    )
    test_loader = DataLoader(
```

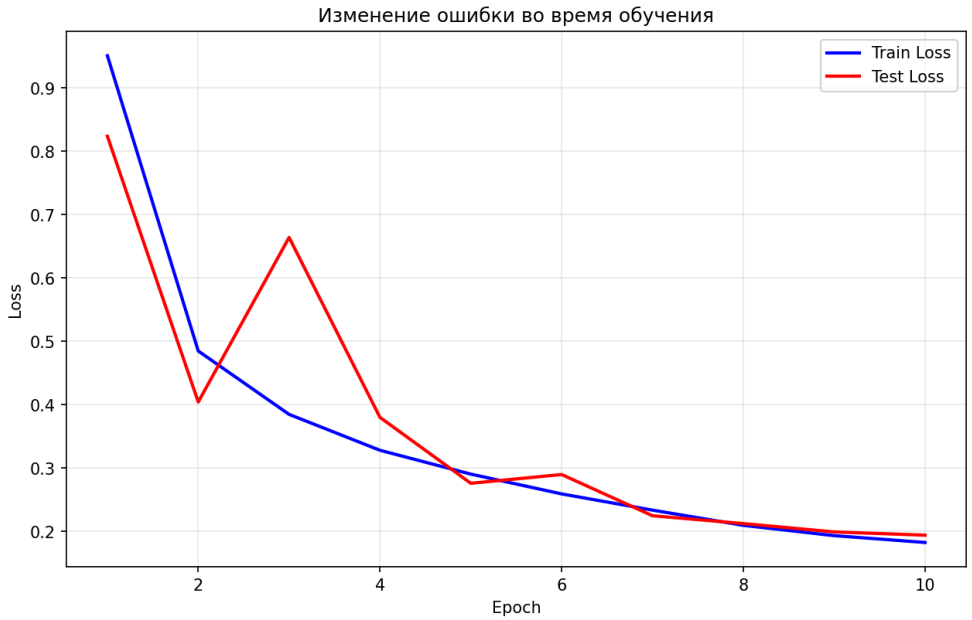
```
        test_set,
        batch_size=BATCH_SIZE,
        shuffle=False,
        num_workers=NUM_WORKERS,
        pin_memory=True
    )

    # Классы Fashion-MNIST
    classes = [
        'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
    ]

    # Создание модели
    print("Creating DenseNet121 model...")
    model = create_densenet121(
        num_classes=10,
        freeze_backbone=FREEZE_BACKBONE
    ).to(device)

    # Функция потерь и оптимизатор
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.RMSprop(
        [p for p in model.parameters() if p.requires_grad],
        lr=LEARNING_RATE,
        weight_decay=WEIGHT_DECAY
    )
    scheduler = optim.lr_scheduler.StepLR(
        optimizer,
        step_size=max(5, EPOCHS // 2),
        gamma=0.1
    )
    # Возобновление обучения
    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
            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'densenet121_epoch{epoch}.pth')
        torch.save({
            'epoch': epoch,
            'model_state': model.state_dict(),
            'optimizer_state': optimizer.state_dict()
        }, 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, EPOCHS + 1), history['train_loss'], label='train')
    plt.plot(range(1, EPOCHS + 1), history['test_loss'], label='test',
linestyle='--')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss')
```

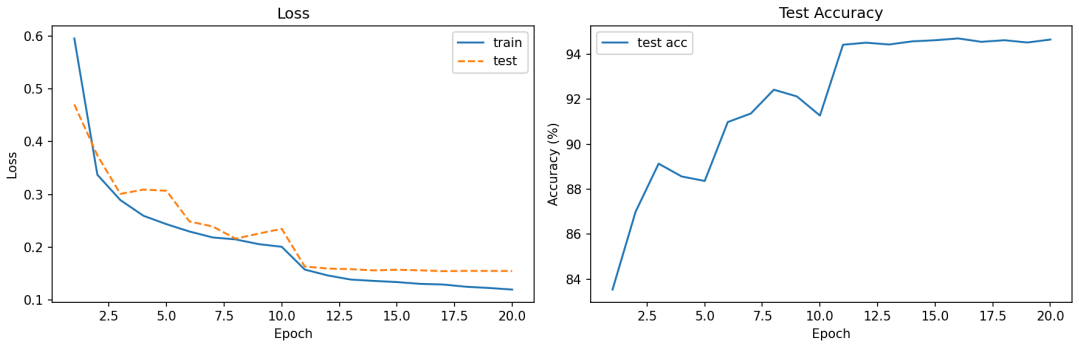
```
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(range(1, EPOCHS + 1), history['test_acc'], label='test acc')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.title('Test Accuracy')
    plt.legend()
    plt.tight_layout()
    history_path = os.path.join(SAVE_DIR, 'training_history.png')
    plt.savefig(history_path, dpi=150)
    print(f'Saved history plot to {history_path}')
    # Визуализация (если указан путь к изображению)
    if VISUALIZE_IMAGE:
        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=(4, 4))
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Pred: {classes[pred_idx]} ({probs[pred_idx] *
100:.1f}%)')
        plt.show()
if __name__ == "__main__":
    main()
```

Вывод:

Л. Р. №1



Л. Р. № 2



State-of-art:

This study aims to explore the effectiveness of a hybrid model combining the VGG16 and DenseNet121 architectures for image classification tasks on the Fashion MNIST dataset. This model is designed to leverage the advantages of both architectures to produce richer feature representations. In this study, the performance of the hybrid model is compared with several other architectures, including LeNet-5, VGG-16, ResNet-20, ResNet-50, EfficientNet-B0, and DenseNet-121, using various optimizers such as Adam, RMSProp, AdaDelta, AdaGrad and SGD. The test results indicate that the Adam and SGD optimizers deliver excellent results. The VGG16 + DenseNet121 hybrid model achieved perfect training accuracy 100%, the highest validation accuracy 94.65%, and excellent test accuracy 94.16%. Confusion matrix analysis confirms that this model is capable of correctly classifying the majority of images, although there is some confusion between classes with visual similarities. These findings affirm that a hybrid approach and the appropriate selection of optimizers can significantly enhance model performance in image classification tasks.

Ссылка на статью

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