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
import requests
from io import BytesIO
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
```

```python
# ======================
# ПАРАМЕТРЫ
# ======================
EPOCHS = 10
BATCH_SIZE = 64
LEARNING_RATE = 1e-3
WEIGHT_DECAY = 1e-4
INPUT_SIZE = 64
NUM_WORKERS = 4
SAVE_DIR = 'results_lab2'

# Ссылка на произвольное изображение из интернета для теста (Пункт 4)
TEST_IMAGE_URL =
"
RYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU3Ojo6Iys/RD84QzQ5OjcBCgoKDQwNGg
8PGjclHyU3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3N//
AABEIAJQAvwMBIgACEQEDEQH/xAAcAAEAAQUBAQAAAAAAAAAAAAAAAQMFBgcIAgT/xABDEAACAQID
BAYECggHAQAAAAAAAQIDBAUGEQchMUESE1FhcYEiMnShFSMmM0JicpGxwRYkVHOCoqPCJTZDRFJT0
RT/xAAVAQEBAAAAAAAAAAAAAAAAAAAAAf/EABQRAQAAAAAAAAAAAAAAAAAAAAD/2gAMAwEAAhEDEQ
A/ANyAAAAAAAAAAAzEsybRcvYBKVKpcu7ul/t7TSb173wXmwMt13jyND4hthzHVu5Ts6FlbW7ekK
UqXWNL60m978EinHbDmnlTwqXjbT/ACmBvwGg5bYc06/N4Wvs20/zmRbbXs0U7mFSvGwr00/SpOj0
NV4p6rx3gb95gwnLe07L2NShQrVZYddyWnVXW6Lfp8H7jNYSU4qUWpRa1Uo70/ACQAAAAAAAAAAAAA
AAAAAAAMBzNtUwbCZVbfDU8SuoNxbpyUaSf2+flqWHbNnSrbSll7DKrhPoJ3tSnLRrXhTXlvfc0u0
1J6q0W5ctOQGS5lzzj2Ym4XXwtbZuEP4ucvPd3IxmWmjSjSO07BxIlvRRDab7lvNjZa2Y0scwC
yxN4tOhO5+udNUul5xcy82bcxYXZ0rTDsNuNZeuEAhuMJK/aUUL4UERL4ucE8PDAPvc5D
9mwOmtHOLMhBNEJTHnj/+dN/zlQ88uWdCGGXTx5ALDyR9wzZMbDFVZ9g+wDsnZVr6RIVu3GMxBGrb
0jWhQzvXXdEhxV+NhUG0iqAVsFFo8Gnc7m/jqiHQHQ/7uuE8vafKmF9jy4YtXXxMSt+0fxCPOgERl
GpuP+UPj4u+Mst4Gwyrh10tIVsWwO4QtGoeHqkJoBK8VH/wnV7M+nG3iCT1gEnfQjBsACzsg8Mv0L
aS8LpTYFa8gq/93K/mmNC/6WiP8KH5LNDq2zT+dT72+wDjHbSnEPUbQ2fMHNgwPFnXhwpeTJGDyEzm
AAAAAAAAAAAOfSgDQSh/Xcy/wi3VnLLK+fpP5UP+LG9bSTJ+Ih0IvPrCIDEb3/RhgP9WZ+YW9+o6Ln
+M0GRaNl4xbkFqUcSfNC2Fk2ztd0EiwlqA==gD7n+bYyNYovvDNUJ/2OzZYyyKQIlVw1GmGdvE4RXvw
ipEG1Wq+tLuRFJnNWP60cFpGAPVHjFWdKJ9QiGW4dfqfrL0VeE0dqIJ8vMApwL4fqM6TYyQC/nCQFE
AAAAAAAAAAAAAAAAAAAAAAAAAA5x2oy12gYz+8
pr+lAxVmS7Snrn3G/36X8kTGeZQAAEoEACQyAAAAv+Qv87YH7XH8GWAveSXpnDBmv2uBB08uLA/H
UAAAAAAAAAAAAAAAHNW0jX9O8b9p/tRjTMk2kP5dY37T+SMaKJIBaqAAAASAAAAAAAAAAH//2Q==" # Пример футболки

# Устройство
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

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

```
# ======================
# 1. МОДЕЛИ
# ======================

# --- Модель из ЛР 2 (DenseNet121) ---
def create_densenet121(num_classes=10):
    print("Creating DenseNet121...")
    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)
    return model

# --- Модель из ЛР 1 (Кастомная архитектура) ---
# ВАЖНО: Сюда лучше вставить класс твоей модели из первой лабы.
# Я написал универсальный пример, который работает с размером 224x224.
class CustomCNN_Lab1(nn.Module):
    def __init__(self, num_classes=10):
        super(CustomCNN_Lab1, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1), # Вход 3 канала (т.к.
трансформ общий)
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((4, 4)) # Адаптивный пулинг, чтобы не
зависеть от размера входа
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

# ======================
# 2. ФУНКЦИИ ОБУЧЕНИЯ
# ======================

def train_one_epoch(model, loader, optimizer, criterion):
    model.train()
    running_loss = 0.0
    for inputs, labels in loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
```

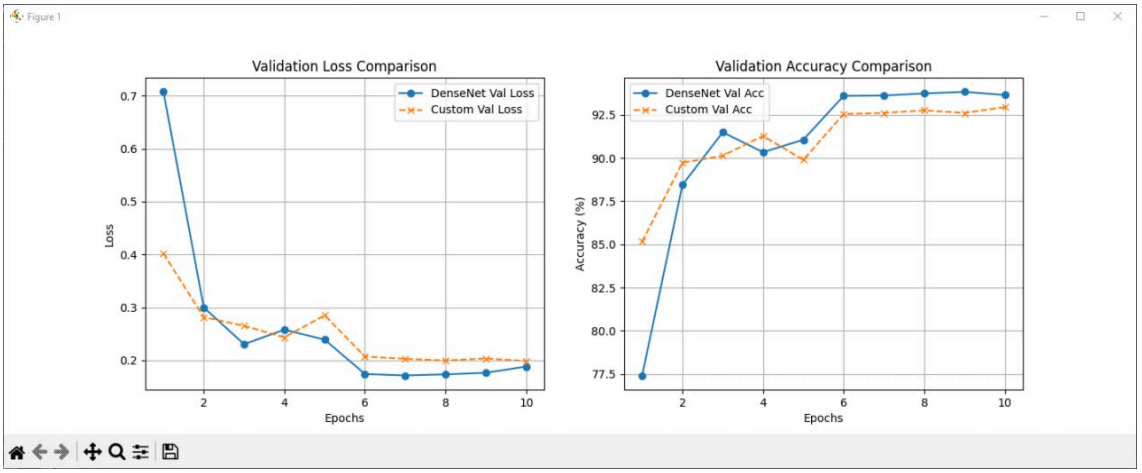
```
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
    return running_loss / len(loader.dataset)

def evaluate(model, loader, criterion):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            running_loss += loss.item() * inputs.size(0)
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    return running_loss / len(loader.dataset), 100. * correct / total

def train_model(model, train_loader, test_loader, model_name="Model"):
    print(f"\nTraining {model_name} on {device}...")

    # Оптимизатор RMSprop (по заданию)
    optimizer = optim.RMSprop(model.parameters(), lr=LEARNING_RATE,
weight_decay=WEIGHT_DECAY)
    criterion = nn.CrossEntropyLoss()
    scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

    history = {'train_loss': [], 'test_loss': [], 'test_acc': []}

    start_time = time.time()

    for epoch in range(EPOCHS):
        t_loss = train_one_epoch(model, train_loader, optimizer, criterion)
        v_loss, v_acc = evaluate(model, test_loader, criterion)
        scheduler.step()

        history['train_loss'].append(t_loss)
        history['test_loss'].append(v_loss)
        history['test_acc'].append(v_acc)

        print(f"Epoch {epoch+1}/{EPOCHS} | Train Loss: {t_loss:.4f} | Val
Loss: {v_loss:.4f} | Val Acc: {v_acc:.2f}%")

    total_time = time.time() - start_time
    print(f"{model_name} training finished in {total_time:.1f}s")

    return history, total_time

# ======================
# 3. ВИЗУАЛИЗАЦИЯ
# ======================

def download_image(url):
    try:
        response = requests.get(url)
```

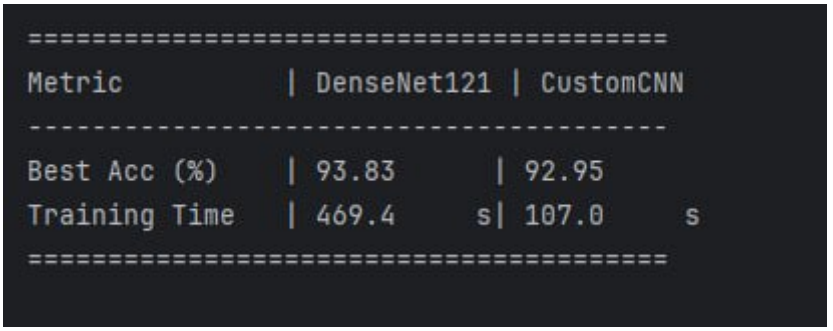
```
        img = Image.open(BytesIO(response.content)).convert('RGB')
        return img
    except Exception as e:
        print(f"Error downloading image: {e}")
        return None

def visualize_comparison(model_densenet, model_custom, transform):
    print("\n--- Visualizing Predictions ---")

    # Скачиваем изображение
    img_orig = download_image(TEST_IMAGE_URL)
    if img_orig is None:
        return

    # Подготовка изображения
    img_tensor = transform(img_orig).unsqueeze(0).to(device)

    model_densenet.eval()
    model_custom.eval()

    with torch.no_grad():
        # DenseNet Prediction
        out1 = model_densenet(img_tensor)
        prob1 = torch.softmax(out1, dim=1)
        score1, pred1 = prob1.max(1)
        label1 = CLASSES[pred1.item()]

        # Custom CNN Prediction
        out2 = model_custom(img_tensor)
        prob2 = torch.softmax(out2, dim=1)
        score2, prob2 = prob2.max(1)
        label2 = CLASSES[prob2.item()]

    # Отображение
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(img_orig)
    plt.axis('off')
    plt.title(f"DenseNet121 Prediction:\n{label1} ({score1.item():.2%})",
color='green')

    plt.subplot(1, 2, 2)
    plt.imshow(img_orig)
    plt.axis('off')
    plt.title(f"Custom CNN (Lab 1) Prediction:\n{label2}
({score2.item():.2%})", color='blue')

    plt.tight_layout()
    plt.savefig(os.path.join(SAVE_DIR, 'visualization_comparison.png'))
    plt.show()

# ======================
# MAIN
# ======================
def main():
    os.makedirs(SAVE_DIR, exist_ok=True)

    # 1. Данные
    # Используем одинаковые преобразования для чистоты эксперимента (размер
224 для DenseNet)
```

```
    # Grayscale(3) нужен, так как DenseNet ожидает 3 канала (RGB)
    imagenet_mean = [0.485, 0.456, 0.406]
    imagenet_std = [0.229, 0.224, 0.225]

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

    train_set = torchvision.datasets.FashionMNIST(root='./data', train=True,
download=True, transform=transform)
    test_set = torchvision.datasets.FashionMNIST(root='./data', train=False,
download=True, transform=transform)

    train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True,
num_workers=NUM_WORKERS)
    test_loader = DataLoader(test_set, batch_size=BATCH_SIZE, shuffle=False,
num_workers=NUM_WORKERS)

    # 2. Создание моделей
    dense_model = create_densenet121().to(device)
    custom_model = CustomCNN_Lab1().to(device)

    # 3. Обучение
    dense_hist, dense_time = train_model(dense_model, train_loader,
test_loader, "DenseNet121")
    custom_hist, custom_time = train_model(custom_model, train_loader,
test_loader, "CustomCNN")

    # 4. Сравнение результатов (Графики)
    epochs_range = range(1, EPOCHS + 1)

    plt.figure(figsize=(14, 5))

    # Loss comparison
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, dense_hist['test_loss'], label='DenseNet Val
Loss', marker='o')
    plt.plot(epochs_range, custom_hist['test_loss'], label='Custom Val Loss',
marker='x', linestyle='--')
    plt.title('Validation Loss Comparison')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    # Accuracy comparison
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, dense_hist['test_acc'], label='DenseNet Val Acc',
marker='o')
    plt.plot(epochs_range, custom_hist['test_acc'], label='Custom Val Acc',
marker='x', linestyle='--')
    plt.title('Validation Accuracy Comparison')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy (%)')
    plt.legend()
    plt.grid(True)

    plt.savefig(os.path.join(SAVE_DIR, 'comparison_plots.png'))
```

```
    print(f"Comparison plots saved to {SAVE_DIR}")
    plt.show()

    # Вывод текстовой таблицы сравнения
    print("\n" + "="*40)
    print(f"{'Metric':<15} | {'DenseNet121':<10} | {'CustomCNN':<10}")
    print("-" * 40)
    print(f"{'Best Acc (%)':<15} | {max(dense_hist['test_acc']):<10.2f} |
{max(custom_hist['test_acc']):<10.2f}")
    print(f"{'Training Time':<15} | {dense_time:<10.1f}s|
{custom_time:<10.1f}s")
    print("="*40 + "\n")

    # 5. Визуализация работы на произвольном изображении
    visualize_comparison(dense_model, custom_model, transform)

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

Вывод:





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