Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра интеллектуально-информационных технологий

Лабораторная работа №2
По дисциплине «Обработка изображений в ИС»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнила:
студентка 4 курса
группы ИИ-24
Коцуба Е.М.
Проверила:
Андренко К.В.

Брест 2025

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

Вариант 5

В-т	Выборка	Оптимизатор	Предобученная архитектура
5	STL-10 (размеченная часть)	SGD	DenseNet121

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.

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torchvision.models import densenet121, DenseNet121_Weights
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import numpy as np
import requests
from io import BytesIO
from PIL import Image
import sys
import os

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

```
# === Трансформации для предобученной модели (224x224) ===
pretrained_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])

# === Трансформации для кастомной модели (96x96) ===
custom transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# === Датасеты ===
train_dataset_pretrained = torchvision.datasets.STL10(root='./data',
split='train', transform=pretrained_transform, download=True)
test_dataset_pretrained = torchvision.datasets.STL10(root='./data',
split='test', transform=pretrained_transform, download=True)

train_dataset_custom = torchvision.datasets.STL10(root='./data',
split='train', transform=custom_transform, download=True)
test_dataset_custom = torchvision.datasets.STL10(root='./data', split='test',
transform=custom transform, download=True)

# === Лоадеры ===
batch_size = 64
train loader pretrained = DataLoader(train dataset pretrained,
batch_size=batch_size, shuffle=True, num_workers=2, pin_memory=True)
test_loader_pretrained = DataLoader(test_dataset_pretrained,
batch_size=batch_size, shuffle=False, num_workers=2, pin_memory=True)

train_loader_custom = DataLoader(train_dataset_custom, batch_size=batch_size,
shuffle=True, num_workers=2, pin_memory=True)
test_loader_custom = DataLoader(test_dataset_custom, batch_size=batch_size,
shuffle=False, num_workers=2, pin_memory=True)

# === Кастомная модель (для 96x96) ===
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).  init  ()
        self.conv1 = nn.Conv2d(3, 16, kernel size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 12 * 12, 512)
        self.relu4 = nn.ReLU()
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = x.view(-1, 64 * 12 * 12)
        x = self.relu4(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
# === Предобученная модель ===
for param in
densenet121(weights=DenseNet121_Weights.IMAGENET1K_V1).parameters():
    param.requires_grad = False
model = densenet121(weights=DenseNet121_Weights.IMAGENET1K_V1)
model.classifier = nn.Linear(model.classifier.in_features, 10)
for param in model.classifier.parameters():
    param.requires_grad = True
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.classifier.parameters(), lr=0.01, momentum=0.9)

# === Обучение предобученной модели ===
num_epochs = 20
train losses, test losses = [], []
train_accuracies, test_accuracies = [], []

total_steps = len(train_loader_pretrained) * num_epochs
step = 0
print("0.0%", end="")

for epoch in range(num_epochs):
    model.train()
    running loss = 0.0
    correct = 0
    total = 0
    for images, labels in train_loader_pretrained:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        step += 1
        progress = step / total_steps * 100
        print(f"\r{progress:.1f}%", end="")
        sys.stdout.flush()

    train_loss = running_loss / len(train_loader_pretrained)
    train_accuracy = 100 * correct / total
    train losses.append(train loss)
    train_accuracies.append(train_accuracy)

    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader_pretrained:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
             , predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    test_loss = test_loss / len(test_loader_pretrained)
```

```
    test_accuracy = 100 * correct / total
    test_losses.append(test_loss)
    test_accuracies.append(test_accuracy)

    print(f"\rЭпоха {epoch + 1}/{num_epochs}, Тренировочная потеря:
{train_loss:.4f}, "
          f"Тренировочная точность: {train_accuracy:.2f}%, "
          f"Тестовая потеря: {test_loss:.4f}, Тестовая точность:
{test_accuracy:.2f}%")

print("\r100.0%")

# === Графики ===
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Тренировочная потеря')
plt.plot(test_losses, label='Тестовая потеря')
plt.xlabel('Эпоха')
plt.ylabel('Потеря')
plt.legend()
plt.title('График изменения ошибки')

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Тренировочная точность')
plt.plot(test_accuracies, label='Тестовая точность')
plt.xlabel('Эпоха')
plt.ylabel('Точность (%)')
plt.legend()
plt.title('График изменения точности')
plt.savefig('pretrained_training_plot.png')
plt.close()

torch.save(model.state_dict(), 'stl10_densenet121.pth')

# === Визуализация предобученной ===
def visualize_prediction(model, dataset, device, filename, is_custom=False):
    model.eval()
    classes = dataset.classes
    idx = np.random.randint(0, len(dataset))
    image, label = dataset[idx]
    image = image.unsqueeze(0).to(device)
    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output, 1)

    image = image.squeeze().cpu().numpy().transpose(1, 2, 0)
    if is_custom:
        image = image * 0.5 + 0.5
    else:
        image = (image * np.array([0.229, 0.224, 0.225]) + np.array([0.485,
0.456, 0.406])).clip(0, 1)
    plt.imshow(image)
    plt.title(f'Истинный класс: {classes[label]}\nПредсказанный класс:
{classes[predicted.item()]}')
    plt.axis('off')
    plt.savefig(filename)
    plt.close()

visualize_prediction(model, test_dataset_pretrained, device,
'pretrained prediction.png')

# === Обучение кастомной модели (96x96) ===
print("Обучение кастомной модели (SimpleCNN)...")
custom_model = SimpleCNN().to(device)
```

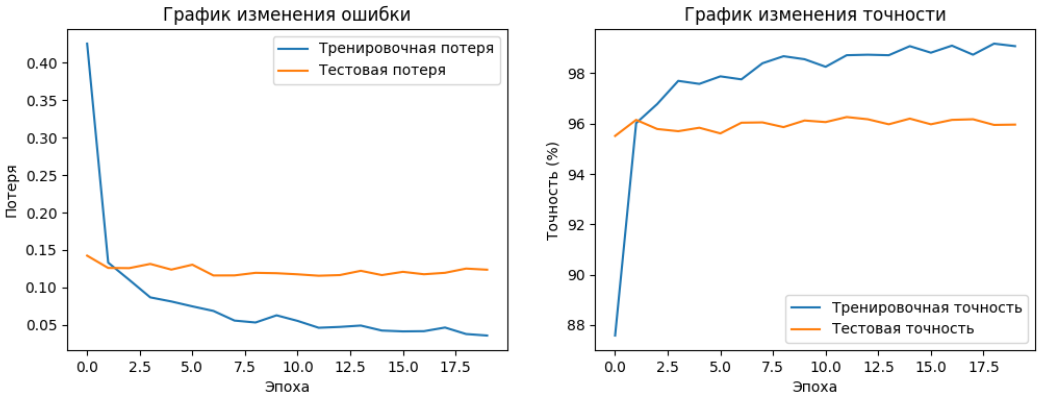
```
custom_optimizer = optim.SGD(custom_model.parameters(), lr=0.01,
momentum=0.9)
custom_epochs = 10

for epoch in range(custom_epochs):
    custom_model.train()
    running loss = 0.0
    for images, labels in train_loader_custom:
        images, labels = images.to(device), labels.to(device)
        custom_optimizer.zero_grad()
        outputs = custom model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        custom_optimizer.step()
        running_loss += loss.item()
    print(f"Кастомная модель: эпоха {epoch+1}/{custom epochs}, потеря:
{running_loss/len(train_loader_custom):.4f}")

torch.save(custom_model.state_dict(), 'stl10_model.pth')
print("stl10_model.pth сохранён")

visualize_prediction(custom_model, test_dataset_custom, device,
'custom_prediction.png', is_custom=True)

# === Классификация по URL ===
def classify_external_image(url, model, transform, classes, device):
    response = requests.get(url)
    img = Image.open(BytesIO(response.content)).convert('RGB')
    img = transform(img).unsqueeze(0).to(device)
    model.eval()
    with torch.no_grad():
        output = model(img)
        _, predicted = torch.max(output, 1)
    print(f'Предсказанный класс для изображения из URL:
{classes[predicted.item()]}')

example_url = 'https://www.avianews.com/wp-
content/uploads/2022/06/30_a321xlr_first_flight.jpg'
classify_external_image(example_url, model, pretrained_transform,
test_dataset_pretrained.classes, device)
```

Результат работы программы:

```
Эпоха 1/20, Тренировочная потеря: 0.4255, Тренировочная точность: 87.58%, Тестовая потеря: 0.1424, Тестовая точность: 95.51%
Эпоха 2/20, Тренировочная потеря: 0.1331, Тренировочная точность: 96.02%, Тестовая потеря: 0.1260, Тестовая точность: 96.15%
Эпоха 3/20, Тренировочная потеря: 0.1101, Тренировочная точность: 96.78%, Тестовая потеря: 0.1258, Тестовая точность: 95.79%
Эпоха 4/20, Тренировочная потеря: 0.0867, Тренировочная точность: 97.70%, Тестовая потеря: 0.1314, Тестовая точность: 95.70%
Эпоха 5/20, Тренировочная потеря: 0.0813, Тренировочная точность: 97.58%, Тестовая потеря: 0.1238, Тестовая точность: 95.84%
Эпоха 6/20, Тренировочная потеря: 0.0747, Тренировочная точность: 97.88%, Тестовая потеря: 0.1304, Тестовая точность: 95.61%
Эпоха 7/20, Тренировочная потеря: 0.0686, Тренировочная точность: 97.76%, Тестовая потеря: 0.1160, Тестовая точность: 96.04%
Эпоха 8/20, Тренировочная потеря: 0.0558, Тренировочная точность: 98.40%, Тестовая потеря: 0.1161, Тестовая точность: 96.05%
Эпоха 9/20, Тренировочная потеря: 0.0532, Тренировочная точность: 98.68%, Тестовая потеря: 0.1195, Тестовая точность: 95.86%
Эпоха 10/20, Тренировочная потеря: 0.0628, Тренировочная точность: 98.56%, Тестовая потеря: 0.1190, Тестовая точность: 96.12%
Эпоха 11/20, Тренировочная потеря: 0.0553, Тренировочная точность: 98.26%, Тестовая потеря: 0.1175, Тестовая точность: 96.06%
Эпоха 12/20, Тренировочная потеря: 0.0462, Тренировочная точность: 98.72%, Тестовая потеря: 0.1156, Тестовая точность: 96.26%
Эпоха 13/20, Тренировочная потеря: 0.0473, Тренировочная точность: 98.74%, Тестовая потеря: 0.1165, Тестовая точность: 96.17%
Эпоха 14/20, Тренировочная потеря: 0.0491, Тренировочная точность: 98.72%, Тестовая потеря: 0.1222, Тестовая точность: 95.97%
Эпоха 15/20, Тренировочная потеря: 0.0424, Тренировочная точность: 99.08%, Тестовая потеря: 0.1165, Тестовая точность: 96.20%
Эпоха 16/20, Тренировочная потеря: 0.0413, Тренировочная точность: 98.82%, Тестовая потеря: 0.1208, Тестовая точность: 95.97%
Эпоха 17/20, Тренировочная потеря: 0.0416, Тренировочная точность: 99.10%, Тестовая потеря: 0.1176, Тестовая точность: 96.15%
Эпоха 18/20, Тренировочная потеря: 0.0465, Тренировочная точность: 98.74%, Тестовая потеря: 0.1195, Тестовая точность: 96.17%
Эпоха 19/20, Тренировочная потеря: 0.0378, Тренировочная точность: 99.18%, Тестовая потеря: 0.1252, Тестовая точность: 95.95%
Эпоха 20/20, Тренировочная потеря: 0.0358, Тренировочная точность: 99.08%, Тестовая потеря: 0.1237, Тестовая точность: 95.96%
```

```
Обучение кастомной модели (SimpleCNN)...
Кастомная модель: эпоха 1/10, потеря: 2.1886
Кастомная модель: эпоха 2/10, потеря: 1.7882
Кастомная модель: эпоха 3/10, потеря: 1.5679
Кастомная модель: эпоха 4/10, потеря: 1.3925
Кастомная модель: эпоха 5/10, потеря: 1.2403
Кастомная модель: эпоха 6/10, потеря: 1.0751
Кастомная модель: эпоха 7/10, потеря: 0.9074
Кастомная модель: эпоха 8/10, потеря: 0.7240
Кастомная модель: эпоха 9/10, потеря: 0.5373
Кастомная модель: эпоха 10/10, потеря: 0.3702
stl10_model.pth сохранён
Предсказанный класс для изображения из URL: airplane
```

Изображение, использованное для проверки (по ссылке):



Вывод: трансферное обучение с использованием предобученной DenseNet121 позволило достичь высокой точности (до 96.26%) на тестовом наборе STL-10 всего за 20 эпох с минимальными вычислительными затратами. Fine-tuning только классификатора — эффективная стратегия при ограниченных данных и ресурсах. Кастомная модель SimpleCNN, обученная с нуля, показала устойчивую сходимость, но уступает по точности предобученной модели. Использование GPU (CUDA) обеспечило быстрое выполнение всех этапов.