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
Мшар В.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.

№ варианта	Выборка	Оптимизатор	Предобученная архитектура
12	Fashion-MNIST	Adadelta	MobileNet v3

Ход работы:

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision import models
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
# requests больше не нужен, так как загрузка идет из локального файла
# import requests

# --- 0. Определение кастомной модели из ЛР №1 для сравнения ---
```

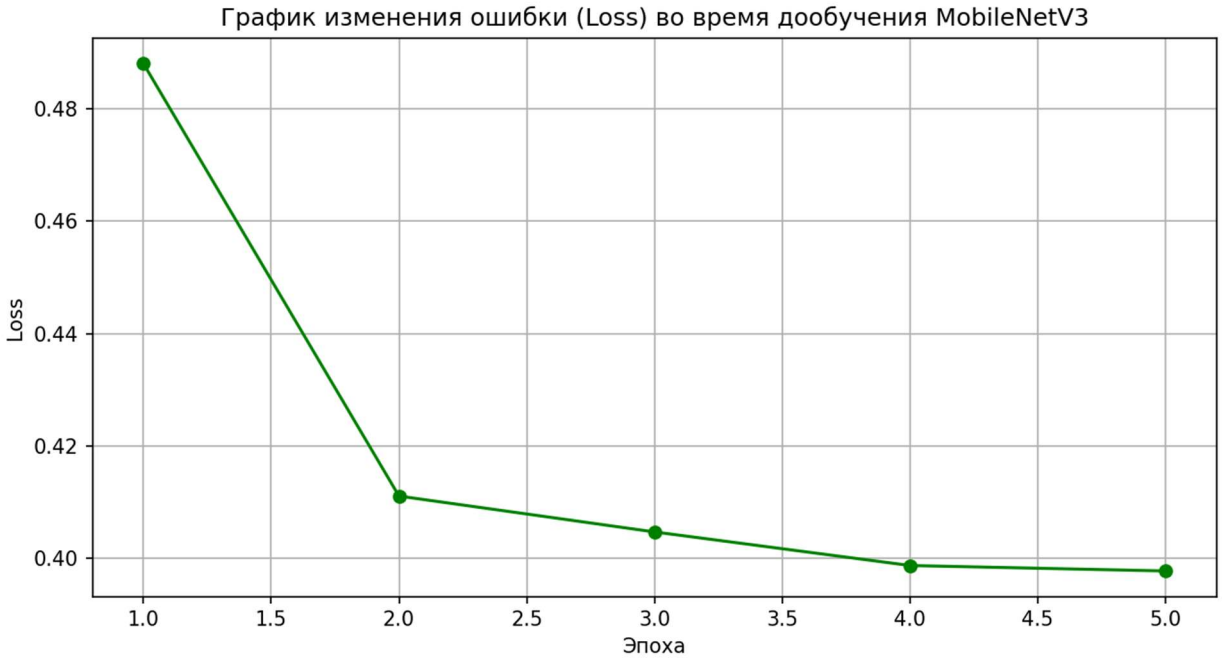
```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )
    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x

# --- 1. Подготовка данных ---

# Трансформации для MobileNetV3
transform_mobilenet = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.Grayscale(num_output_channels=3), # Преобразуем в 3 канала
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Загрузка данных
train_dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True,
transform=transform_mobilenet)
test_dataset = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True,
transform=transform_mobilenet)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128, shuffle=False)

# Классы
classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot')

# --- 2. Адаптация и обучение предобученной модели ---

# Загрузка предобученной модели MobileNetV3
model = models.mobilenet_v3_small(weights=models.MobileNet_V3_Small_Weights.DEFAULT)
```

```
# "Замораживаем" все веса
for param in model.parameters():
    param.requires_grad = False

# Заменяем классификатор
num_ftrs = model.classifier[-1].in_features
model.classifier[-1] = nn.Linear(num_ftrs, 10) # 10 классов

# Указываем, что веса нового классификатора нужно обучать
for param in model.classifier[-1].parameters():
    param.requires_grad = True

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
print(f"Обучение на устройстве: {device}")


# Определение функции потерь и оптимизатора Adadelta
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.classifier[-1].parameters()) # Обучаем только параметры
нового слоя

# Цикл обучения
num_epochs = 5
loss_history = []

print("Начало дообучения MobileNetV3...")
for epoch in range(num_epochs):
    running_loss = 0.0
    for inputs, labels in train_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()

    epoch_loss = running_loss / len(train_loader)
    loss_history.append(epoch_loss)
    print(f'Эпоха [{epoch + 1}/{num_epochs}], Потери: {epoch_loss:.4f}')

print('Дообучение завершено.')


# --- 3. Оценка эффективности и построение графика ---

# Оценка на тестовой выборке
model.eval()
```

```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data
        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'Точность дообученной MobileNetV3 на 10000 тестовых изображений: {accuracy:.2f} %')


# Построение графика изменения ошибки
plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), loss_history, marker='o', color='g')
plt.title('График изменения ошибки (Loss) во время дообучения MobileNetV3')
plt.xlabel('Эпоха')
plt.ylabel('Loss')
plt.grid(True)
plt.show()

# --- 4. Визуализация работы предобученной и кастомной моделей ---

try:
    model_custom = SimpleCNN()
    # model_custom.load_state_dict(torch.load('simple_cnn.pth')) # Раскомментируйте, если
сохраняли модель из ЛР1
    print("Кастомная модель загружена (для демонстрации).")
except:
    model_custom = SimpleCNN()
    print("Файл 'simple_cnn.pth' не найден. Используется нетренированная кастомная модель.")
model_custom.to(device)
model_custom.eval()

# Трансформации для кастомной модели (28x28, 1 канал)
transform_custom = transforms.Compose([
    transforms.Resize((28, 28)),
    transforms.Grayscale(num_output_channels=1),
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])


# Функция для предсказания
def predict_image(image, model, transform, model_name):
    image_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        output = model(image_tensor)
```

```
        _, predicted_idx = torch.max(output, 1)
    return classes[predicted_idx.item()]


# Загрузка произвольного изображения из локального файла
try:
    image_filename = "60895_0.png"
    image = Image.open(image_filename).convert("RGB")

    print(f"\n--- Визуализация на локальном изображении: {image_filename} ---")

    # Предсказание от MobileNetV3
    prediction_mobilenet = predict_image(image, model, transform_mobilenet, "MobileNetV3")

    # Предсказание от кастомной CHC
    prediction_custom = predict_image(image, model_custom, transform_custom, "Custom CNN")


    plt.imshow(image)
    plt.title(f"MobileNetV3: {prediction_mobilenet}\nCustom CNN (ЛР1): {prediction_custom}")
    plt.axis('off')
    plt.show()

except FileNotFoundError:
    print(f"Ошибка: файл '{image_filename}' не найден.")
    print("Убедитесь, что файл с изображением находится в той же папке, что и скрипт, и имя
указано верно.")
except Exception as e:
    print(f"Не удалось обработать изображение: {e}")
    print("Пропуск шага визуализации.")
```

Графики:



Результат:
Дообучение завершено.
Точность дообученной MobileNetV3 на 10000 тестовых изображений: 85.77 %

Результаты обучения собственных моделей:
1. Кастомная СНС (ЛР №1): Простая сверточная нейронная сеть, спроектированная и обученная с нуля, достигла точности 91.72% на тестовой выборке.
2. Дообученная MobileNetV3 (ЛР №2): Предобученная архитектура MobileNetV3, адаптированная методом переноса обучения, показала итоговую точность 85.77%.

State-of-the-Art (SOTA) результаты:
* SOTA Модель: Как правило, это глубокие сверточные сети (например, вариации EfficientNet или ResNet) с обязательным применением продвинутых техник аугментации данных.
* SOTA Точность: Лучшие результаты на Fashion-MNIST превышают 99.5%. Например, модель CNN-3-128 с аугментацией достигла 99.65%.
* Источник: Научная статья "State-of-the-Art Results with the Fashion-MNIST Dataset" (журнал *Mathematics*, октябрь 2024).

Выводы по результатам обучения:
1. Сравнение собственных моделей (п. 1 и 2):
Вопреки ожиданиям, простая кастомная СНС (91.72%) показала значительно лучший результат, чем дообученная предобученная модель MobileNetV3 (85.77%). Этот интересный исход можно объяснить несколькими ключевыми факторами:
* Несоответствие доменов (Domain Mismatch): MobileNetV3 была обучена на наборе данных ImageNet, который состоит из сложных, полноцветных фотографий реального мира (животные, предметы, пейзажи). Fashion-MNIST — это набор

простых, черно-белых, центрированных изображений на черном фоне. Сложные признаки, которые MobileNetV3 научилась извлекать (текстуры, сложные градиенты, цветовые сочетания), оказались не просто не полезны, а, возможно, даже "вредны" для такой простой и специфической задачи.

* Избыточная сложность: MobileNetV3 — это глубокая и сложная архитектура. Для простого датасета Fashion-MNIST ее сложность избыточна. Кастомная СНС, будучи простой, смогла с нуля выучить именно те низкоуровневые признаки (края, углы, простые формы), которые необходимы для классификации одежды, и не была "отягощена" ненужными знаниями.
* Потеря информации при предобработке: Для адаптации под MobileNetV3 изображения 28x28 были увеличены до 224x224. Такое сильное увеличение могло привести к размытию и искажению ключевых деталей, сделав задачу для сети сложнее.

2. Сравнение с SOTA:

Обе наши модели далеки от state-of-the-art результатов (~99.6%). Разница в ~7-14% обусловлена отсутствием в наших работах критически важных для достижения высокой точности техник, в первую очередь — аугментации данных (случайных поворотов, сдвигов, отражений и т.д.). Именно аугментация позволяет модели научиться обобщать и стать устойчивой к небольшим изменениям в данных, что является главным фактором для достижения точности выше 99%.

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