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

Бузель С.Д.

Проверил:

Андренко К. В.

Брест 2025

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

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.

В-т	**Выборка**	**Оптимизатор**	**Предобученная архитектура**
3	CIFAR-10	SGD	ResNet34

Код программы:

1. Программа обучения:

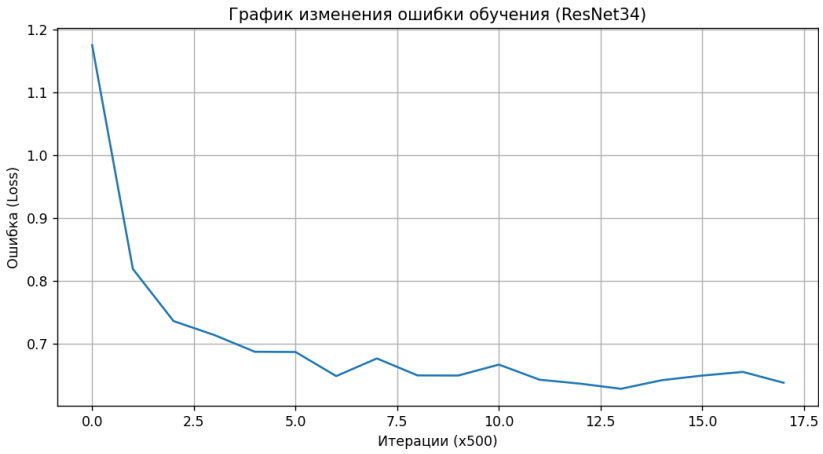
```
import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

if __name__ == '__main__':
    # 0. Настройка устройства (CPU)
    device = torch.device('cpu')
    print(f'Используется устройство: {device}')

    # 1. Адаптация данных CIFAR-10 для ResNet
    transform_resnet = transforms.Compose([
        transforms.Resize(224), # Изменяем размер изображений 32x32 на 224x224
        transforms.ToTensor(),
```

```
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # Стандартная нормализация для ImageNet
])

# Загрузка данных с новыми трансформациями
trainset = torchvision.datasets.CIFAR10(root='D:/REALOIIS/lab2', train=True,
                                        download=True, transform=transform_resnet)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=16,
                                          shuffle=True, num_workers=0)

testset = torchvision.datasets.CIFAR10(root='D:/REALOIIS/lab2', train=False,
                                       download=True, transform=transform_resnet)
testloader = torch.utils.data.DataLoader(testset, batch_size=16,
                                         shuffle=False, num_workers=0)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# 2. Загрузка и модификация предобученной модели ResNet34
# Загружаем ResNet34 с весами, обученными на ImageNet
model_resnet = models.resnet34(weights=models.ResNet34_Weights.IMAGENET1K_V1)

# "Замораживаем" все слои сети, чтобы их веса не обновлялись во время обучения.
for param in model_resnet.parameters():
    param.requires_grad = False

# Узнаем, сколько признаков подается на вход последнему слою (классификатору)
num_ftrs = model_resnet.fc.in_features

# Заменяем последний слой (fc - fully connected) на новый, который будем обучать.
model_resnet.fc = nn.Linear(num_ftrs, 10)     # У него будет 10 выходов (по числу классов в CIFAR-10)

# Перемещаем модель на выбранное устройство (CPU)
model_resnet = model_resnet.to(device)

# 3. Определение критерия и оптимизатора
criterion = nn.CrossEntropyLoss()
# Оптимизировать будем только параметры нового, незамороженного слоя
optimizer = optim.SGD(model_resnet.fc.parameters(), lr=0.001, momentum=0.9)

# 4. Процесс обучения
print("Начало обучения ResNet34...")
loss_history_resnet = []
epochs = 3 # Для transfer learning часто нужно меньше эпох

for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)

        optimizer.zero_grad()
        outputs = model_resnet(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 500 == 499: # Вывод каждые 500 батчей
```

```
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 500:.3f}')
            loss_history_resnet.append(running_loss / 500)
            running_loss = 0.0

print('Обучение ResNet34 завершено')

# Сохраняем обученную модель для использования в 4 пункте
torch.save(model_resnet.state_dict(), 'D:/REALOIIS/lab2/cifar_resnet34.pth')

# 5. Построение графика и оценка
plt.figure(figsize=(10, 5))
plt.plot(loss_history_resnet)
plt.title('График изменения ошибки обучения (ResNet34)')
plt.xlabel('Итерации (x500)')
plt.ylabel('Ошибка (Loss)')
plt.grid(True)
plt.show()

correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = model_resnet(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Точность предобученной сети на 10000 тестовых изображений: {accuracy:.2f} %')
```

Точность сети на 10000 тестовых изображений: 77.52 %
Изображения для предсказания:
Истинные метки: cat   ship ship plane
Предсказания: cat   plane truck plane

--- Анализ одного изображения ---
Истинный класс: cat
Предсказанный класс: cat



2. Программа сравнения распознования произвольного изображения:

```
import torch
import torch.nn as nn
import torchvision.models as models
import torchvision.transforms as transforms
from PIL import Image
import matplotlib.pyplot as plt

# --- ШАГ 1: Определение архитектур ---

# 1.1. Копируем класс из ЛБ1
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10) # 10 классов на выходе

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# 1.2. Создаем "скелет" для модели ResNet34 из ЛБ2
def create_resnet_model():
    model = models.resnet34(weights=None) # Загружаем архитектуру без предобученных весов
    num_ftrs = model.fc.in_features
    # Заменяем последний слой
    model.fc = nn.Linear(num_ftrs, 10)
    return model
```

```
# --- ШАГ 2: Подготовка инстументов ---

# 2.1. Определяем список классов CIFAR-10
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# 2.2. Создаем два разных набора трансформаций.
# Каждая модель получает изображение в том формате в котором она обучалась

# Трансформации для кастомной модели (32x32, нормализация 0.5)
transform_custom = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Трансформации для ResNet модели (224x224, нормализация ImageNet)
transform_resnet = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# --- ШАГ 3: Загрузка моделей ---

# 3.1. Загрузка кастомной модели
custom_model = SimpleCNN()
custom_model.load_state_dict(torch.load('D:/REALOIIS/lab2/cifar_custom_cnn.pth'))
# Переводим модель в режим оценки
custom_model.eval()

# 3.2. Загрузка ResNet модели
resnet_model = create_resnet_model()
resnet_model.load_state_dict(torch.load('D:/REALOIIS/lab2/cifar_resnet34.pth'))
```

```
resnet_model.eval()

# --- ШАГ 4: Функция для предсказания ---

def predict_image(image_path, model, transform, model_name):
    try:
        # Открываем изображение и конвертируем в RGB
        image = Image.open(image_path).convert('RGB')
    except FileNotFoundError:
        print(f"ОШИБКА: Файл не найден по пути '{image_path}'. Проверьте имя файла.")
        return

    print(f"\n--- Предсказание моделью: {model_name} ---")

    # Показываем исходное изображение
    plt.imshow(image)
    plt.title(f"Исходное изображение для {model_name}")
    plt.axis('off')
    plt.show()

    # Применяем нужные трансформации и добавляем batch размерность
    image_tensor = transform(image).unsqueeze(0)

    # Делаем предсказание (без вычисления градиентов)
    with torch.no_grad():
        outputs = model(image_tensor)
        # Применяем Softmax для получения вероятности
        probabilities = torch.nn.functional.softmax(outputs[0], dim=0)
        # Находим класс с максимальной вероятностью
        _, predicted_idx = torch.max(outputs, 1)

    predicted_class = classes[predicted_idx.item()]
    confidence = probabilities[predicted_idx.item()].item() * 100

    print(f"Результат: модель думает, что это '{predicted_class}'")
    print(f"Уверенность: {confidence:.2f}%")
```

```
# --- ШАГ 5: Запуск предсказаний ---

if __name__ == '__main__':
    path_to_image = 'D:/REALOIIS/lab2/justadog.png'

    # Вызываем функцию предсказания для каждой модели
    predict_image(path_to_image, custom_model, transform_custom, "из (ЛБ1)")
    predict_image(path_to_image, resnet_model, transform_resnet, "из (ЛБ2)")
```

--- Предсказание моделью: из (ЛБ1) ---

Результат: модель думает, что это 'cat'

Уверенность: 30.02%

--- Предсказание моделью: из (ЛБ2) ---

Результат: модель думает, что это 'dog'

Уверенность: 44.17%

Вывод: Я научился осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.