Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «Обработка изображений в ИС»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнила:

Студентка 4 курса

Группы ИИ-21

Соболева П.С.

Проверил:

Крощенко А.А.

Брест 2024

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

Ход работы:

Вариант 14

В-т	Выборка	Оптимизатор	Предобученная архитектура
14	CIFAR-100	Adadelta	ResNet18

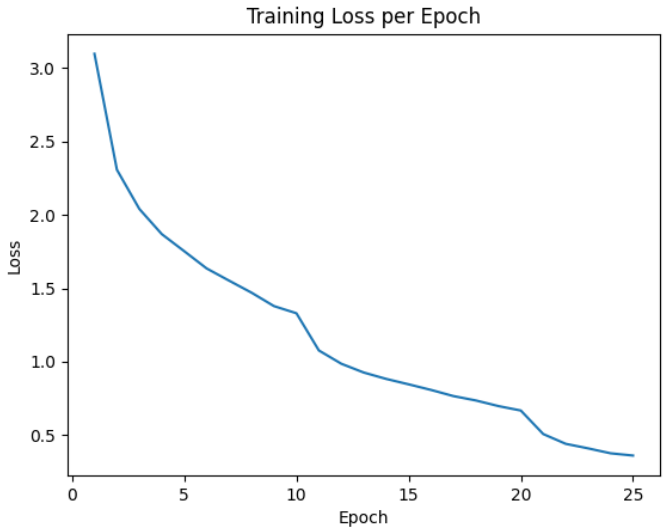
Код:

```
import torch
import torchvision
from torchvision import transforms
import torch.nn as nn
import numpy as np
import seaborn as sns
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Параметры загрузки данных
batch_size_train = 256
batch_size_test = 100

# Преобразования для CIFAR-100 с аугментацией данных
preprocess = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomCrop(32, padding=4),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
])

# Загрузка данных CIFAR-100
train_loader = torch.utils.data.DataLoader(

torch-
vi-
sion.datasets.CIFAR100(root='C:\\Users\\user\\PycharmProjects\\ОИвИС\\data',
train=True, download=True,
                                   transform=preprocess),
    batch_size=batch_size_train, shuffle=True
)

test_loader = torch.utils.data.DataLoader(

torch-
vi-
sion.datasets.CIFAR100(root='C:\\Users\\user\\PycharmProjects\\ОИвИС\\data',
train=False, download=True,
                                   transform=preprocess),
    batch_size=batch_size_test, shuffle=False
)

# Загрузка предобученной ResNet18 и адаптация под CIFAR-100
model = torch-
vi-
```

```
si-
on.models.resnet18(weights=torchvision.models.ResNet18_Weights.IMAGENET1K_V1)
model.fc = nn.Linear(model.fc.in_features, 100)  # Изменение выходного слоя
для 100 классов


# Функция для обучения модели
def train(device, model, train_loader, learning_rate=1.0, epochs=50, mod-
el_save_path='best_model.pth'):
    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adadelta(model.parameters(), lr=learning_rate)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gam-
ma=0.5)  # Понижение lr каждые 10 эпох
    history = []
    best_loss = float('inf')

    for epoch in tqdm(range(epochs), desc="Training Progress"):
        model.train()
        epoch_loss = 0.0
        for batch_idx, (x, y) in enumerate(train_loader):
            x, y = x.to(device), y.to(device)
            optimizer.zero_grad()
            pred = model(x)
            loss = criterion(pred, y)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()

            # Промежуточный вывод для отслеживания прогресса по мини-батчам
            if batch_idx % 10 == 0:
                print(f"Epoch [{epoch + 1}/{epochs}], Batch
[{batch_idx}/{len(train_loader)}], Loss: {loss.item()}")

        average_loss = epoch_loss / len(train_loader)
        history.append(average_loss)
        scheduler.step()  # Обновление learning rate

        if average_loss < best_loss:
            best_loss = average_loss
            torch.save(model.state_dict(), model_save_path)
            print(f'Model saved with loss {best_loss:.4f} at epoch {epoch +
1}')
        print(f'Epoch {epoch + 1}, Average Loss: {average_loss:.4f}')

    plt.plot(range(1, epochs + 1), history)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Loss per Epoch')
    plt.show()


# Функция для тестирования модели и построения матрицы ошибок
def test(model, device, test_loader):
    model.eval()
    correct = 0
    total = 0
    all_labels = []
    all_predictions = []
    num_classes = 100
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
```

```
            _, predicted = torch.max(outputs, 1)

            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_labels.extend(labels.cpu().numpy())
            all_predictions.extend(predicted.cpu().numpy())

    accuracy = correct / total
    print(f"Accuracy on the test set: {accuracy:.2%}")

    cm = confusion_matrix(all_labels, all_predictions)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=(20, 18))
    sns.heatmap(cm_normalized, annot=False, fmt='.2f', cmap='Blues',
cbar=True)

    plt.xlabel('Predicted', fontsize=14)
    plt.ylabel('True', fontsize=14)
    plt.title('Confusion Matrix (Normalized)', fontsize=16)

    plt.xticks(np.arange(num_classes) + 0.5, labels=np.arange(num_classes),
rotation=90, fontsize=10)
    plt.yticks(np.arange(num_classes) + 0.5, labels=np.arange(num_classes),
rotation=0, fontsize=10)

    plt.tight_layout()
    plt.show()


# Инициализация и запуск обучения и тестирования
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Запуск обучения
train(device, model, train_loader, learning_rate=1.0, epochs=25)

# Тестирование модели
model.load_state_dict(torch.load('best_model.pth'))
test(model, device, test_loader)
```

График изменения ошибки при обучении:



Результат:

```
Training Progress:   0%|          | 0/25 [00:00<?, ?it/s]Epoch [1/25], Batch [0/196], Loss: 5.013242721557617
Epoch [1/25], Batch [10/196], Loss: 4.516826629638672
Epoch [1/25], Batch [20/196], Loss: 3.9552435874938965
```

…

```
Training Progress:  52%|█████     | 13/25 [36:13<32:58, 164.89s/it]Epoch [14/25], Batch [0/196], Loss: 0.9354202747344971
Epoch [14/25], Batch [10/196], Loss: 0.8009625673294067
Epoch [14/25], Batch [20/196], Loss: 0.8338631987571716
Epoch [14/25], Batch [30/196], Loss: 0.9106451272964478
```

…

```
Epoch [25/25], Batch [180/196], Loss: 0.39431649446487427
Epoch [25/25], Batch [190/196], Loss: 0.3345850110054016
Training Progress: 100%|██████████| 25/25 [1:03:48<00:00, 153.14s/it]
Model saved with loss 0.3597 at epoch 25
Epoch 25, Average Loss: 0.3597
```

Точность на предобученной модели:

```
Accuracy on the test set: 57.67%
```

Точность на непредобученной модели (по предыдущей работе):

```
Accuracy on the test set: 52.30%
```

Вывод: в ходе выполнения лабораторной работы осуществила обучение НС, сконструированных на базе предобученных архитектур НС.