МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙГОСУДАРСТВЕННЫЙТЕХНИЧЕСКИЙУНИВЕРСИТЕТ» ФАКУЛЬТЕТ

ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ИИ-22

Выполнил:
Сидоренко А.А.
студент группы ИИ-22

Проверил:
А.А. Крощенко,
ст. преп.
кафедры ИИТ,
«__» _____ 2024 г.

Брест 2024

Вариант 16

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

Задания:

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.

Выборка	**Предобученная архитектура**	**Оптимизатор**
CIFAR-100	RMSprop	SqueezeNet 1.1

Код программы:

```
import torchvision as tvn
import torchvision.models as models
import torchvision.transforms as tfs
import torch.utils.data as data
from torchvision.datasets import CIFAR100
import torch.optim as opt
import torch.nn as nn
import torch
from tqdm import tqdm
from PIL import Image

lr = 0.001
epochs = 3
batch_size = 32


def train(model, data_train, loss_f, optz):
    for i in range(epochs):
        loss_mean = 0
        lm_count = 0

        train_tqdm = tqdm(data_train, leave=True)
        for x_data, y_data in train_tqdm:
            predict = model(x_data)
            loss = loss_f(predict, y_data)

            optz.zero_grad()
            loss.backward()
            optz.step()

            lm_count += 1
            loss_mean = 1 / lm_count * loss.item() + (1 - 1 / lm_count) * loss_mean

            train_tqdm.set_description(f"Epoch[{i + 1}/{epochs}],
loss_mean={loss_mean:.3f}")


def test(model, data_test):
    correct = 0
    total = 0

    test_tqdm = tqdm(data_test, leave=True)
```

```
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters())

def train_model(num_epochs):
    train_losses = []
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        progress_bar = tqdm(train_loader, desc=f'Epoch {epoch+1}/{num_epochs}',
leave=False)

        for inputs, labels in progress_bar:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            progress_bar.set_postfix({'loss': loss.item()})

        epoch_loss = running_loss / len(train_loader)
        train_losses.append(epoch_loss)
        print(f'Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss}')

    return train_losses

def evaluate_model():
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Accuracy on test set: {accuracy}%')

def plot_loss_curve(train_losses):
    plt.plot(train_losses)
    plt.title('Loss over Epochs')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()
```

Результаты обучения:

```
Predicted class: 91, Class name: trout, Confidence: 1.00
None
```

Вывод: на практике научилась осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.