Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине: «Обработка изображений в ИС»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-21

Романко Н. А.

Проверил:

Крощенко А. А.

Брест 2024

Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

Ход работы:

№	Сфера применения	Размер исходного изображения	Оптимизатор
4	CIFAR-100	32x32	SGD

Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);

Код программы:

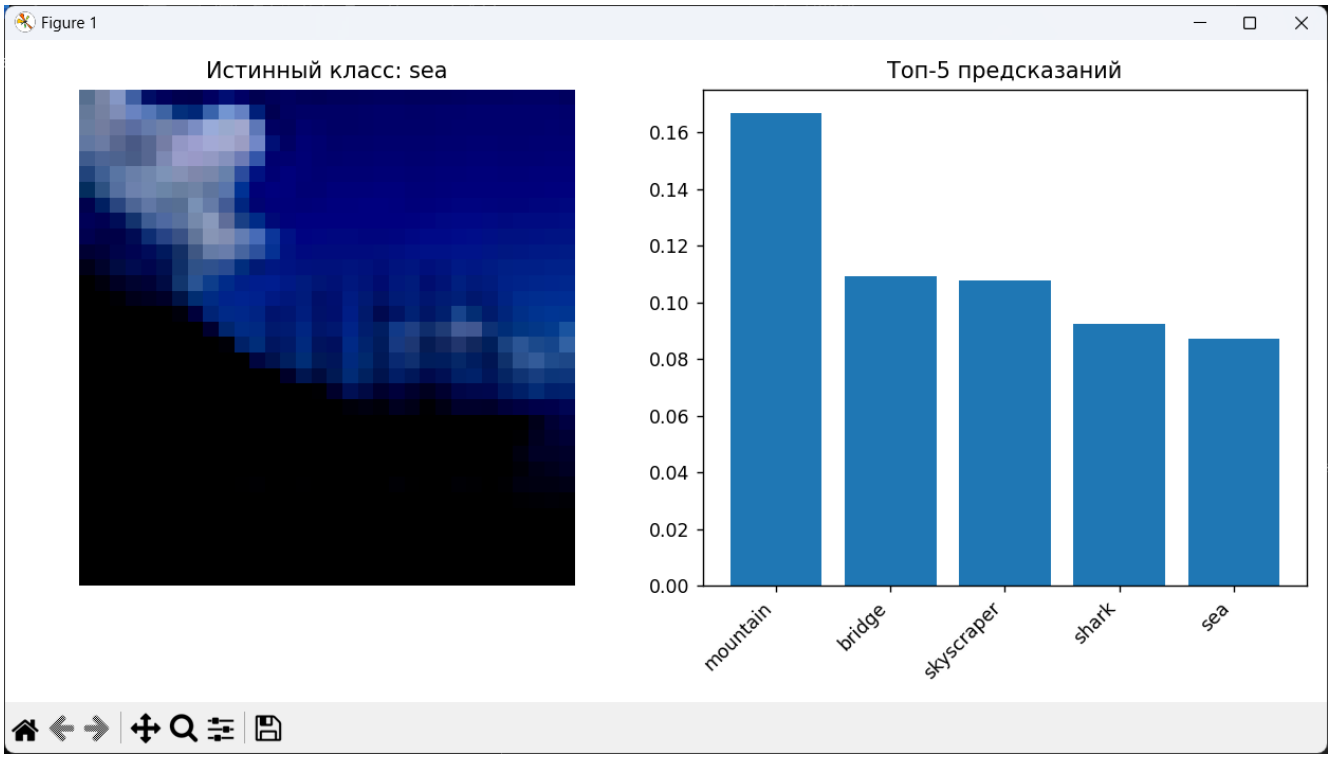
```
import torch
import torch.utils.data
import torch.nn as nn
from torch.optim.sgd import SGD
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import os

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.fc2 = nn.Linear(512, 100)

    def forward(self, x):
        x = self.pool(torch.relu(self.conv1(x)))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

def save_model(model, path='model.pth'):
    torch.save(model.state_dict(), path)
    print(f"Модель сохранена в {path}")

def load_model(model, device, path='model.pth'):
    model.load_state_dict(torch.load(path, map_location=torch.device(device)))
    print(f"Модель загружена из {path}")
    return model

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

trainset = torchvision.datasets.CIFAR100(root='./data', train=True, download=True,
transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = CNN().to(device)
if os.path.exists('model.pth'):
    use_saved_model = input("Найдена сохраненная модель. Использовать ее? (да/нет): ").lower() == 'да'
else:
    use_saved_model = False
```

```
if use_saved_model:
    net = load_model(net, device)
else:
    criterion = nn.CrossEntropyLoss()
    optimizer = SGD(net.parameters(), lr=0.001)

    losses = []
    num_epoch = 35
    total_batches = len(trainloader)
    for epoch in range(num_epoch):
        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 = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            if i % 100 == 99:
                avg_loss = running_loss / 100
                losses.append(avg_loss)
                print(f'Epoch [{epoch+1}/{num_epoch}], Batch [{i+1}/{total_batches}], Loss:
{avg_loss:.4f}')
                running_loss = 0.0

    plt.plot(losses)
    plt.title('Изменение ошибки')
    plt.xlabel('Итерации (x100)')
    plt.ylabel('Ошибка')
    plt.show()

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

    accuracy = 100 * correct / total
    print(f'Точность на тестовом наборе: {accuracy:.2f}%')

def visualize_random_prediction(testdata):
    testset = testdata

    random_index = int(torch.randint(0, len(testset), (1,)).item())
    image, label = testset[random_index]
    image_tensor = image.unsqueeze(0).to(device)

    with torch.no_grad():
        output = net(image_tensor)
    _, predicted = torch.max(output, 1)
    classes = testset.classes

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(image.permute(1, 2, 0))
    plt.title(f'Истинный класс: {classes[label]}')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    probabilities = torch.nn.functional.softmax(output[0], dim=0)
    top_5_prob, top_5_catid = torch.topk(probabilities, 5)
    top_5_classes = [classes[idx] for idx in top_5_catid]
    plt.bar(top_5_classes, top_5_prob.cpu().numpy())
    plt.title('Топ-5 предсказаний')
    plt.xticks(rotation=45, ha='right')
```

```
    plt.tight_layout()
    plt.show()

testset = torchvision.datasets.CIFAR100(root='./data', train=False, download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False)

test_model(net, testloader)
while True:
    visualize_random_prediction(testset)
    choice = input("Хотите просмотреть еще одно изображение? (да/нет): ").lower()
    if choice != 'да':
        break

save_model_choice = input("Хотите сохранить обученную модель? (да/нет): ").lower()
if save_model_choice == 'да':
    save_model(net)

print("Программа завершена.")
```

Результат:

```
Модель загружена из model.pth
Files already downloaded and verified
Точность на тестовом наборе: 19.67%
```



Вывод: в ходе выполнения лабораторной работы научился конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.