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»

Выполнил:

Студент 4 курса

Группы ИИ-21

Романко Н. А.

Проверил:

Крощенко А. А.

Брест 2024

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

Ход работы:

№	Выборка	Оптимизатор	Предобученная архитектура
4	MNIST	SGD	ResNet18

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.

Код программы:

```
import torch
import torch.utils.data
import torch.nn as nn
from torch.optim.sgd import SGD
import torchvision
from torchvision.transforms import v2
from torchvision.models import resnet18, ResNet18_Weights
import matplotlib.pyplot as plt
import numpy as np
import os
import keyboard

model_path = 'ОИ\\лаба 2\\model_resnet.pth'

def save_model(model, path=model_path):
    while True:
        save = input("Save model? (y/n): ")
        if save.lower() == 'y':
            torch.save(model.state_dict(), model_path)
            print(f"Model saved to {path}.")
            break
        elif save.lower() == 'n':
            print("Model not saved.")
            break
        else:
            print("Invalid input. Please enter 'y' or 'n'.")

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

def stop_callback(event):
    global stop_training
    if event.name == 'p' or event.name == 'з':
        stop_training = True
        print("Training stoped by user.")

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'Accuracy on test set: {accuracy:.2f}%')

def visualize_random_prediction(testdata, visualisedata):
```

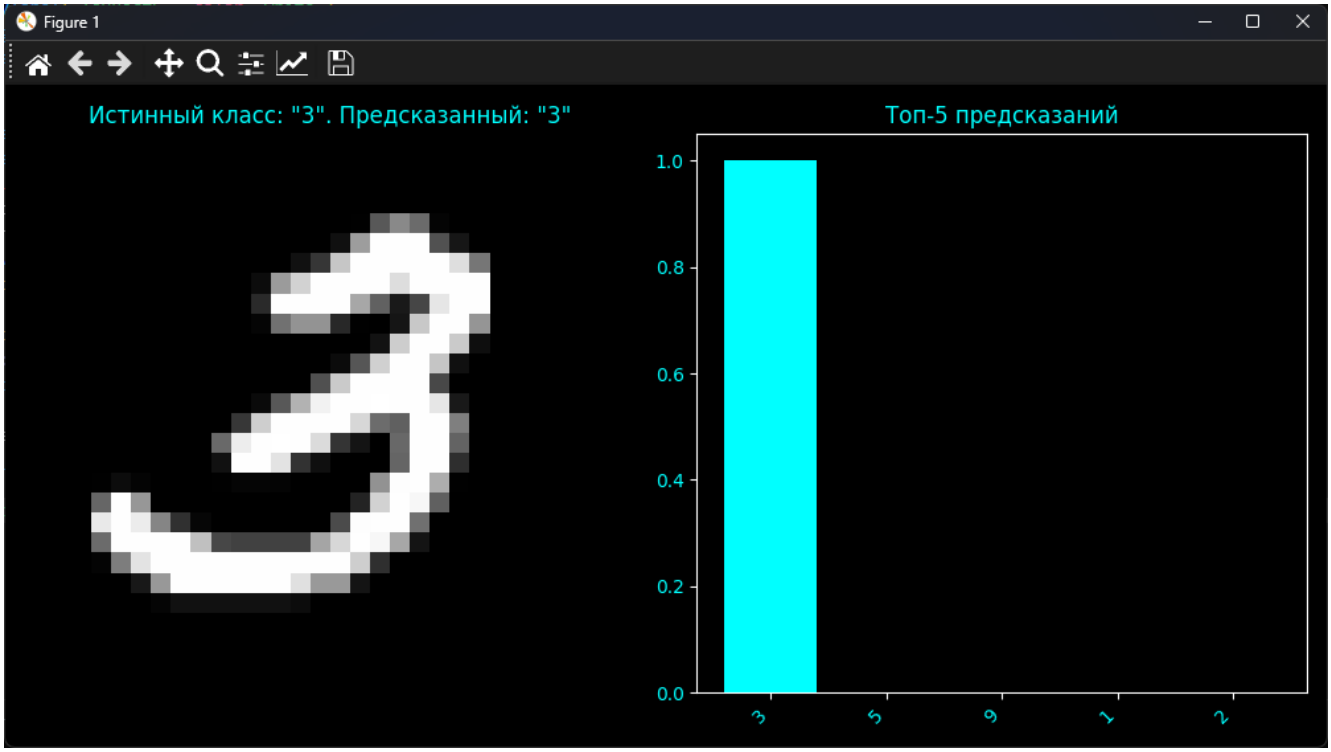
```
    testset = testdata

    net.eval()

    random_index = int(torch.randint(0, len(testset), (1,)).item())
    image, label = testset[random_index]
    visulise_image, _ = visualisedata[random_index]

    image_tensor = image.unsqueeze(0).to(device)

    with torch.no_grad():
        output = net(image_tensor)

    _, predicted = torch.max(output, 1)
    classes = [str(i) for i in range(10)]

    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.style.use('dark_background')

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(visulise_image, cmap='gray')
    plt.title(f'Истинный класс: \"{classes[label]}\". Предсказанный: \"{classes[predicted.item()]}\"',
color='cyan')  # type:ignore
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.bar(top_5_classes, top_5_prob.cpu().numpy(), color='cyan')
    plt.title('Топ-5 предсказаний', color='cyan')
    plt.xticks(rotation=45, ha='right', color='cyan')
    plt.yticks(color='cyan')

    plt.gcf().set_facecolor('black')

    plt.tight_layout()
    plt.show()

class ResNet18_MNIST(nn.Module):
    def __init__(self):
        super(ResNet18_MNIST, self).__init__()
        self.resnet = resnet18(weights=ResNet18_Weights.DEFAULT)

        # for param in self.resnet.parameters():
        #     param.requires_grad = False

        self.resnet.avgpool = nn.AdaptiveAvgPool2d((1, 1))

        num_features = self.resnet.fc.in_features
        self.resnet.fc = nn.Sequential(  #type: ignore
            nn.Linear(num_features, 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        return self.resnet(x)

train_transform = v2.Compose([
    v2.Grayscale(num_output_channels=3),
    v2.RandomHorizontalFlip(),
    v2.RandomRotation(degrees=(0, 25)),
    v2.Compose([v2.ToImage(), v2.ToDtype(torch.float32, scale=True)]),
    v2.Normalize((0.5,), (0.5,))
])

test_transform = v2.Compose([
```

```
    v2.Grayscale(num_output_channels=3),
    v2.Compose([v2.ToImage(), v2.ToDtype(torch.float32, scale=True)]),
    v2.Normalize((0.5,), (0.5,))
])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=train_transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=test_transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=32, shuffle=False)
visualiseset = torchvision.datasets.MNIST(root='./data', train=False, download=True)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
net = ResNet18_MNIST().to(device)

if os.path.exists(model_path):
    while True:
        use_saved_model = input("Saved model found. Use it? (y/n): ").lower()
        if use_saved_model == 'y':
            net = load_model(net, device)
            is_saved_model = True
            break
        elif use_saved_model == 'n':
            is_saved_model = False
            break
        else:
            print("Invalid input. Please enter 'y' or 'n'.")

if not is_saved_model:
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.AdamW(net.parameters(), lr=0.001, weight_decay=1e-4)  #type:ignore
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)

    accuracies = []
    num_epoch = 150
    total_batches = len(trainloader)

    plt.ion()
    fig, ax = plt.subplots()
    fig.patch.set_facecolor('black')
    ax.set_facecolor('black')
    ax.set_xlabel('Итерации', color='white')
    ax.set_ylabel('Точность', color='white')
    line, = ax.plot(accuracies, color='cyan')
    plt.show()

    text_box = ax.text(0.5, 0.1, '', fontsize=12, color='white', ha='center', transform=ax.transAxes)

    stop_training = False
    keyboard.on_press(stop_callback)

    for epoch in range(num_epoch):
        if stop_training:
            break

        scheduler.step()
        correct = 0
        total = 0

        for i, data in enumerate(trainloader, 0):
            if stop_training:
                break

            inputs, labels = data[0].to(device), data[1].to(device)
            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
```

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

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

            if i % 100 == 99:
                accuracy = 100 * correct / total
                accuracies.append(accuracy)
                correct = 0
                total = 0

                line.set_ydata(accuracies)
                line.set_xdata(np.arange(len(accuracies)))
                text_box.set_text(f'Epoch: {epoch + 1}/{num_epoch}\nBatch: {i +
1}/{total_batches}\nAccuracy: {accuracy:.2f}%')
                ax.relim()
                ax.autoscale_view()
                fig.canvas.draw()
                fig.canvas.flush_events()
                fig.canvas.draw()

    plt.ioff()
    plt.plot(accuracies, color='cyan')
    plt.title('Изменение точности', color='white')
    plt.xlabel('Итерации', color='white')
    plt.ylabel('Точность', color='white')
    plt.gca().set_facecolor('black')
    plt.show()

test_model(net, testloader)
visualize_random_prediction(testset, visualiseset)
while True:
    choice = input("Check another image? (y/n): ").lower()
    if choice == 'y':
        visualize_random_prediction(testset, visualiseset)
        continue
    elif choice == 'n':
        break
    else:
        print("Invalid input. Please enter 'y' or 'n'.")

if not is_saved_model:
    save_model(net)

print("Программа завершена.")
```

Результат:

```
Accuracy on test set: 97.27%
```



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