`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Конструирование моделей на базе предобученных нейронных сетей»

Выполнил:

Студент 4 курса

Группы ИИ-21

Карагодин Д. Л.

Проверил:

Крощенко А. А.

Брест 2024

`

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

Ход работы:

Вариант 1

В-т	Выборка	Оптимизатор	Предобученная архитектура
1	MNIST	SGD	AlexNet

Код программы:

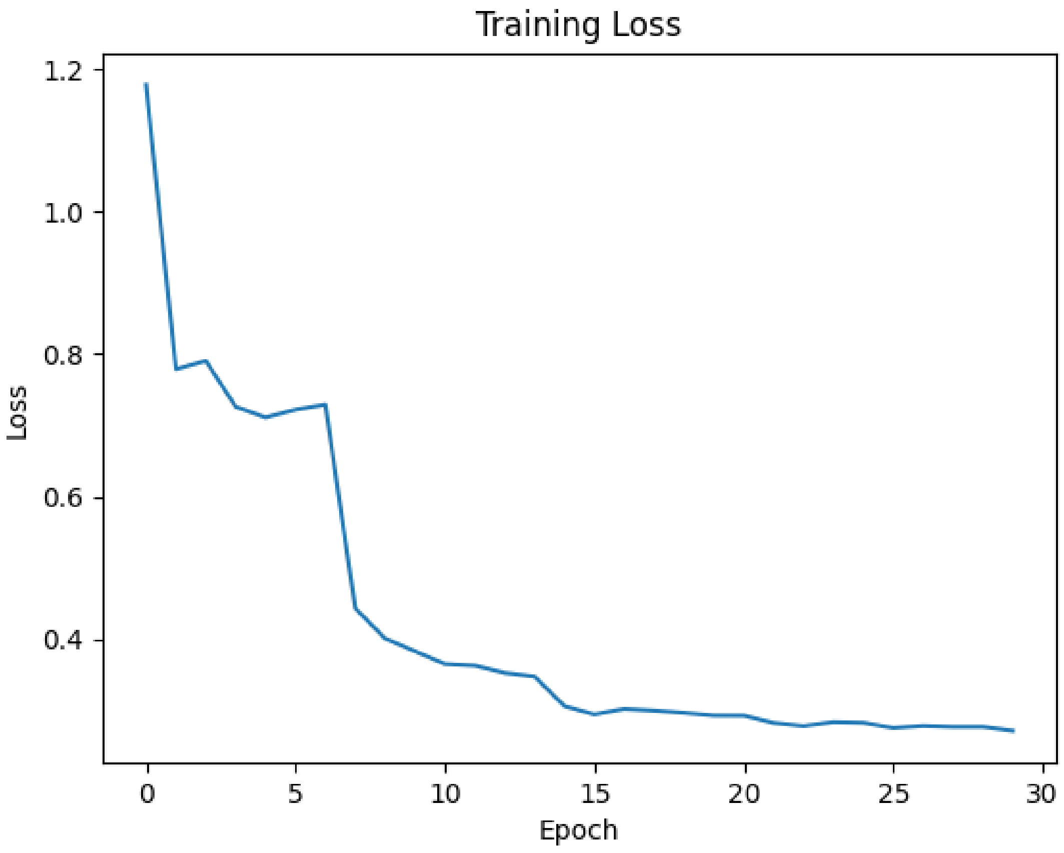
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import matplotlib.pyplot as plt
from torchvision.models import AlexNet_Weights

print(torch.cuda.is_available() )
device = torch.device("cuda")
print(f"Using device: {device}")

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(num_output_channels=3),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])
])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False,
transform=transform, download=True)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=1000, shuffle=False)


model = models.alexnet(weights=AlexNet_Weights.IMAGENET1K_V1)

for param in model.parameters():
    param.requires_grad = False

model.classifier[6] = nn.Linear(4096, 10)


model = model.to(device)

criterion = nn.CrossEntropyLoss()
```

`

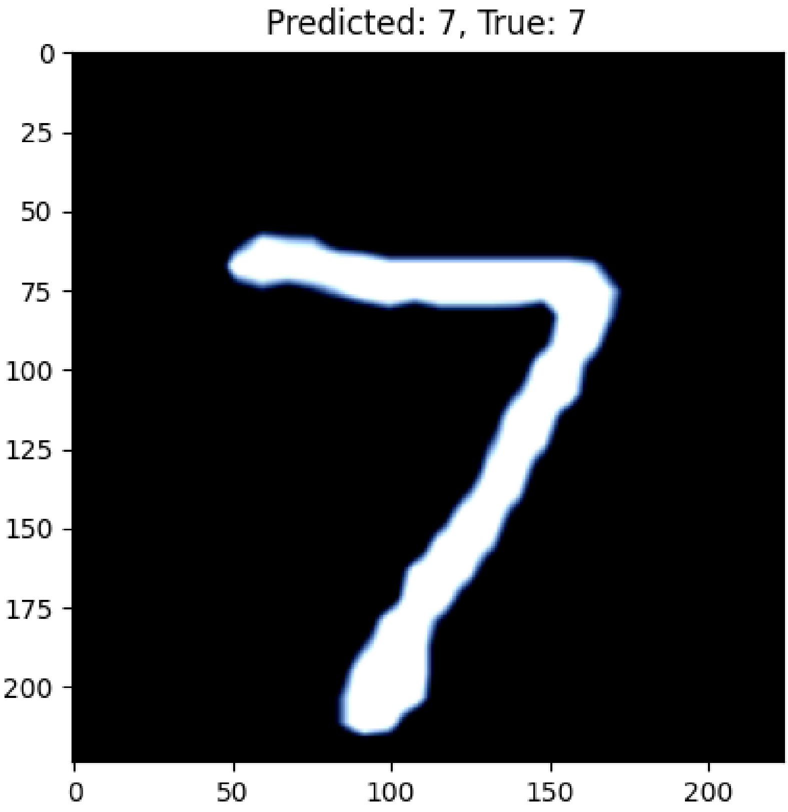
```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.825)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=7,
gamma=0.25)

def train_model(num_epochs):
    model.train()
    train_loss_history = []
    for epoch in range(num_epochs):
        running_loss = 0.0
        for i, (images, labels) in enumerate(train_loader):

            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            # for name, param in model.named_parameters():
            #     if param.requires_grad:
            #         print(f'{name}: {param.grad}')
            running_loss += loss.item()

        epoch_loss = running_loss / len(train_loader)
        train_loss_history.append(epoch_loss)
        scheduler.step()
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}')

    return train_loss_history

def test_model():
    model.eval()
    correct = 0
    total = 0
    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.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Accuracy on the test set: {accuracy:.2f}%')
    return accuracy

def plot_loss_history(loss_history):
    plt.plot(loss_history)
    plt.title('Training Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.show()

num_epochs = 30
loss_history = train_model(num_epochs)
```

```
`

test_model()
plot_loss_history(loss_history)

def visualize_prediction(image_index):
    image, label = test_dataset[image_index]
    model.eval()
    with torch.no_grad():

        image = image.to(device).unsqueeze(0)
        output = model(image)
        _, predicted = torch.max(output.data, 1)

    plt.imshow(image.cpu().squeeze().permute(1, 2, 0), cmap='gray')
    plt.title(f'Predicted: {predicted.item()}, True: {label}')
    plt.show()

visualize_prediction(0)
```

Результат программы:



`



```
Epoch [1/30], Loss: 1.1780
Epoch [2/30], Loss: 0.7788
Epoch [3/30], Loss: 0.7904
Epoch [4/30], Loss: 0.7261
Epoch [5/30], Loss: 0.7112
Epoch [6/30], Loss: 0.7220
Epoch [7/30], Loss: 0.7290
Epoch [8/30], Loss: 0.4430
Epoch [9/30], Loss: 0.4005
Epoch [10/30], Loss: 0.3826
Epoch [11/30], Loss: 0.3647
Epoch [12/30], Loss: 0.3625
Epoch [13/30], Loss: 0.3523
Epoch [14/30], Loss: 0.3472
Epoch [15/30], Loss: 0.3056
Epoch [16/30], Loss: 0.2940
Epoch [17/30], Loss: 0.3017
Epoch [18/30], Loss: 0.2993
Epoch [19/30], Loss: 0.2961
Epoch [20/30], Loss: 0.2925
Epoch [21/30], Loss: 0.2923
Epoch [22/30], Loss: 0.2820
Epoch [23/30], Loss: 0.2776
Epoch [24/30], Loss: 0.2832
Epoch [25/30], Loss: 0.2822
Epoch [26/30], Loss: 0.2749
Epoch [27/30], Loss: 0.2779
```

`

```
Epoch [28/30], Loss: 0.2765
Epoch [29/30], Loss: 0.2765
Epoch [30/30], Loss: 0.2710
Accuracy on the test set: 96.96%
```

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