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

Павлюкович И.М.

Проверил:

Крощенко А.А.

Брест 2024

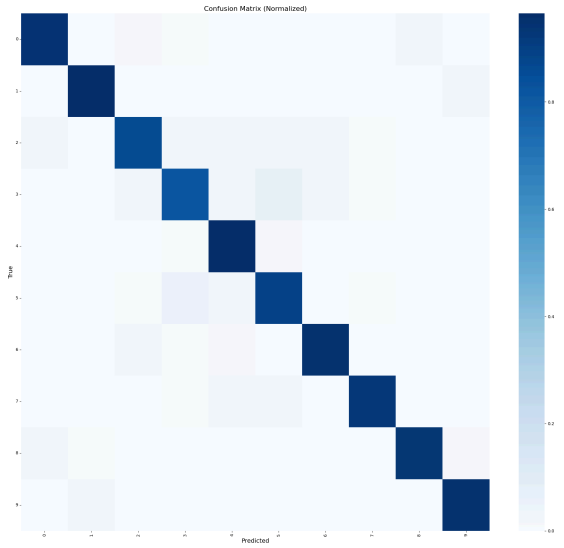
**Цель**: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

| | | | |
|---|---|---|---|
| 14 | CIFAR-100 | Adadelta | ResNet18 |

Код программы:

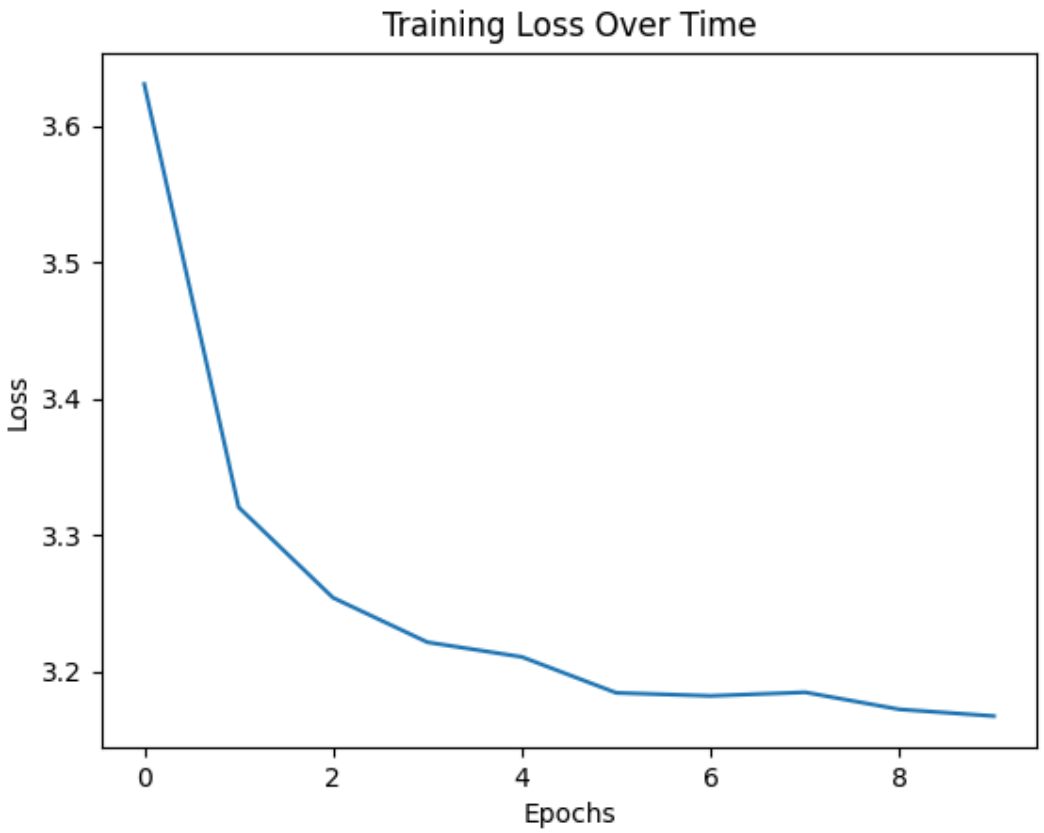
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import random
from torchvision import models

# Преобразование для CIFAR-100 (изменение размера до 32x32 и
нормализация)
transform = transforms.Compose([
    transforms.Resize((32, 32)),  # Изменение размера до
32x32
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
# Нормализация для трех каналов (RGB)
])

# Загрузка тренировочного и тестового набора CIFAR-100
train_set = torchvision.datasets.CIFAR100(root='./data',
train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_set,
batch_size=64, shuffle=True)

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

# Использование предобученной модели ResNet-18
model = models.resnet18(pretrained=True)

# Замена последнего слоя для предсказания 100 классов вместо
1000
model.fc = nn.Linear(model.fc.in_features, 100)

# Замораживаем начальные слои, чтобы не изменять
предобученные веса
for param in model.parameters():
    param.requires_grad = False

# Размораживаем последний слой для обучения
for param in model.fc.parameters():
    param.requires_grad = True

# Функция потерь и оптимизатор
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.fc.parameters())


# Функция для тренировки модели
def train_model(model, train_loader, criterion, optimizer,
epochs=10):
    model.train()
    loss_history = []

    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            optimizer.zero_grad()

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

            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_loss = running_loss / len(train_loader)
        loss_history.append(avg_loss)
        print(f"Epoch [{epoch + 1}/{epochs}], Loss:
{avg_loss:.4f}")

    return loss_history


# Функция для тестирования модели
def test_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

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


# Функция для визуализации предсказанных изображений и
сравнения с истинными метками
def visualize_predictions(model, dataset, num_images=5):
    model.eval()
    fig, axes = plt.subplots(1, num_images, figsize=(12, 3))
    random_indices = random.sample(range(len(dataset)),
num_images)

    for i, index in enumerate(random_indices):
        image, label = dataset[index]

        # Визуализация исходного изображения
        img = image.numpy().transpose((1, 2, 0)) * 0.5 + 0.5
# Обратная нормализация
        axes[i].imshow(img)

        # Подготовка изображения для модели
        image = image.unsqueeze(0)

        with torch.no_grad():
            output = model(image)
            _, predicted = torch.max(output, 1)

        # Отображение истинного и предсказанного класса
        axes[i].set_title(f"True: {label}, Pred:
{predicted.item()}")
        axes[i].axis('off')

    plt.show()


# Обучение модели
loss_history = train_model(model, train_loader, criterion,
optimizer, epochs=10)

# Тестирование модели
test_accuracy = test_model(model, test_loader)

# Построение графика обучения
```

```
plt.plot(loss_history)
plt.title('Training Loss Over Time')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```




```
# Визуализация предсказаний
visualize_predictions(model, test_set, num_images=5)
```

```
Epoch [1/10], Loss: 3.6310
Epoch [2/10], Loss: 3.3203
Epoch [3/10], Loss: 3.2539
Epoch [4/10], Loss: 3.2212
Epoch [5/10], Loss: 3.2104
Epoch [6/10], Loss: 3.1841
Epoch [7/10], Loss: 3.1818
Epoch [8/10], Loss: 3.1845
Epoch [9/10], Loss: 3.1719
Epoch [10/10], Loss: 3.1670
Test Accuracy: 24.66%
```




**Вывод**: научился осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.