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-22

Кузюк Д. Н.

Проверил:

Крощенко А.А.

Брест 2024

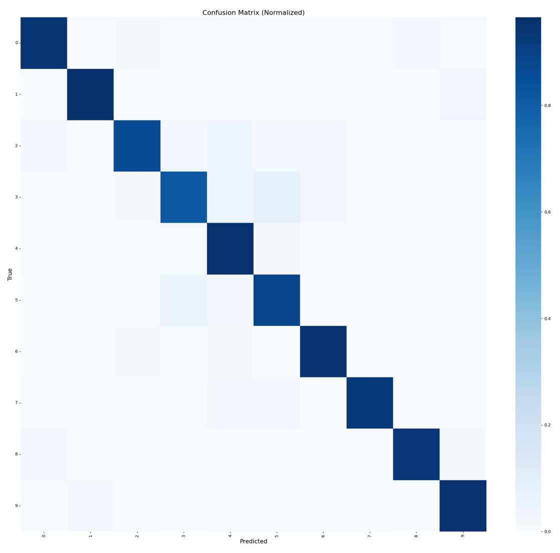
Цель: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

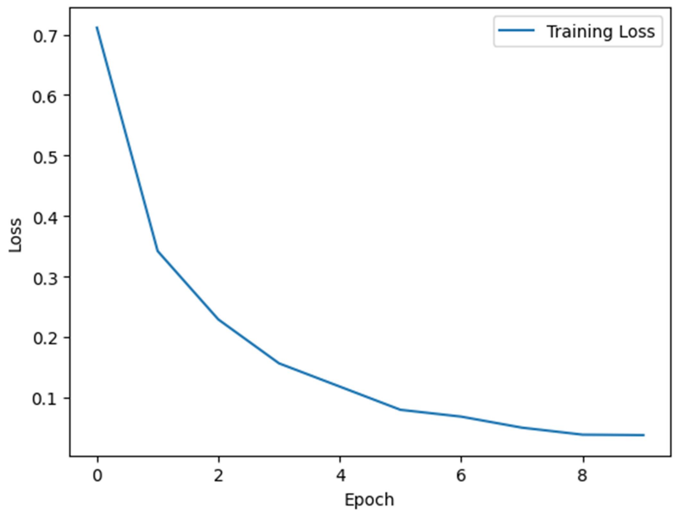
13	CIFAR-10	Adadelta	ResNet18

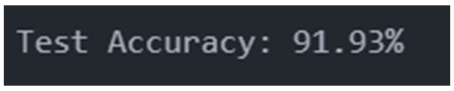
Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

from torchvision import datasets, transforms, models
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

batch_size = 64
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)
learning_rate = 1.0

model = models.resnet18(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, 10)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters(), lr=learning_rate)
num_epochs = 10

def train(model, train_loader, criterion, optimizer, num_epochs):
    model.train()
    train_loss_history = []

    for epoch in range(num_epochs):
        running_loss = 0.0
        for images, labels in train_loader:
            images, labels = images.to(device), labels.to(device)

            optimizer.zero_grad()

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

            loss.backward()
            optimizer.step()

            running_loss += loss.item()

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

    return train_loss_history
train_loss_history = train(model, train_loader, criterion, optimizer, num_epochs)
plt.plot(train_loss_history, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns

def evaluate(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    all_predictions = []
    all_labels = []
    num_classes = 10

    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, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            all_predictions.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

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

    cm = confusion_matrix(all_labels, all_predictions)
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.figure(figsize=(20, 18))
    sns.heatmap(cm_normalized, annot=False, fmt='.2f', cmap='Blues', cbar=True)

    plt.xlabel('Predicted', fontsize=14)
    plt.ylabel('True', fontsize=14)
    plt.title('Confusion Matrix (Normalized)', fontsize=16)

    plt.xticks(np.arange(num_classes) + 0.5, labels=np.arange(num_classes),
rotation=90, fontsize=10)
    plt.yticks(np.arange(num_classes) + 0.5, labels=np.arange(num_classes),
rotation=0, fontsize=10)

    plt.tight_layout()
    plt.show()
    return accuracy
test_accuracy = evaluate(model, test_loader)
def visualize_predictions(model, test_loader, num_images=5):
    model.eval()
    images_shown = 0
    class_names = test_dataset.classes

    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, 1)

            for i in range(images.size(0)):
                if images_shown == num_images:
                    return
                img = images[i].cpu().numpy().transpose((1, 2, 0))
                img = (img * 0.5 + 0.5)
                plt.imshow(img)
                plt.title(f'Predicted: {class_names[predicted[i]]}, Actual:
{class_names[labels[i]]}')
                plt.axis('off')
                plt.show()
                images_shown += 1
visualize_predictions(model, test_loader)
```







```
Test Accuracy: 91.93%
```

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