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-22

Павлюкович И.М.

Проверил:

Крощенко А.А.

Брест 2024

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

14	CIFAR-100	32X32	Adadelta

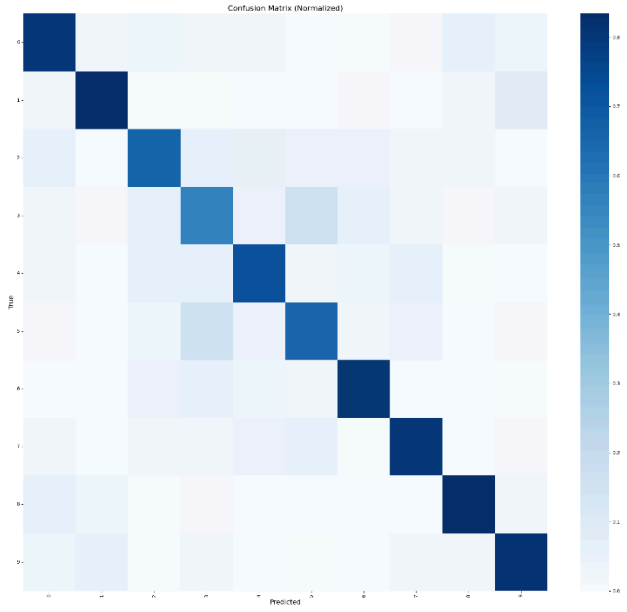
Код программы:

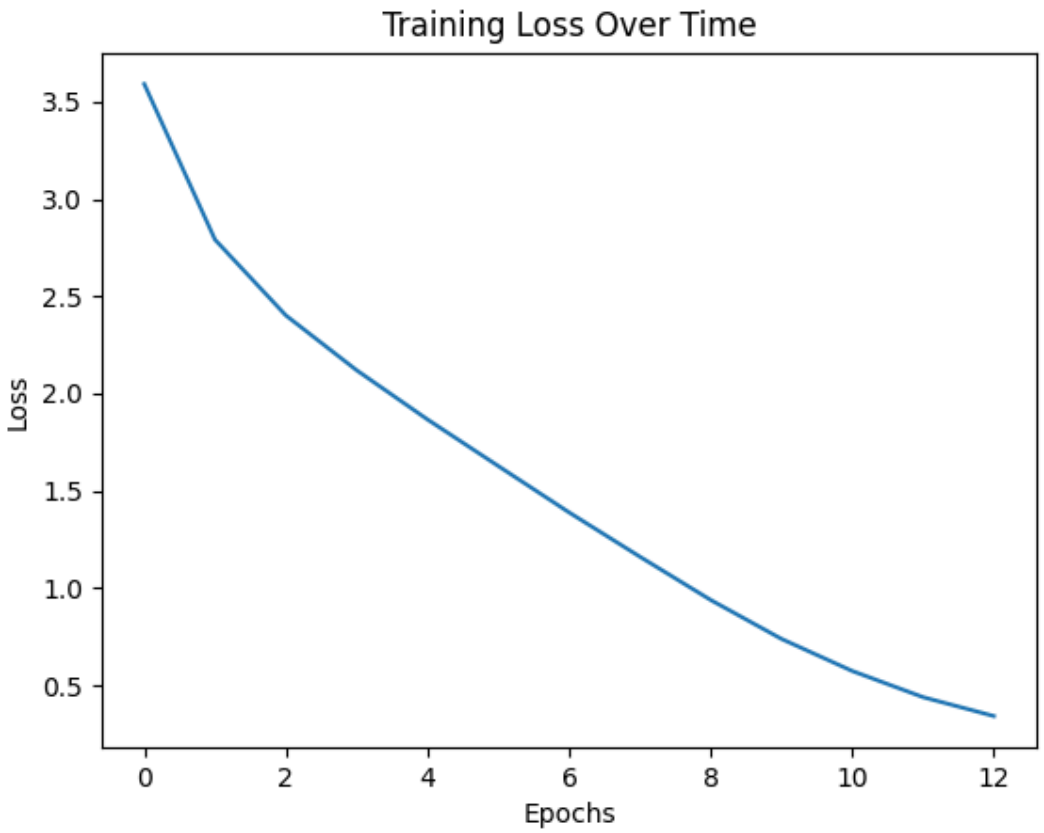
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import random

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

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)


# Определение CNN модели
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Первый сверточный слой
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3,
stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Второй сверточный слой
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3,
stride=1, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Полносвязный слой
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.relu3 = nn.ReLU()

        # Выходной слой для 100 классов
        self.fc2 = nn.Linear(128, 100)

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)  # Векторизация
        x = self.relu3(self.fc1(x))
        x = self.fc2(x)
        return x


# Инициализация модели, функции потерь и оптимизатора
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters())


# Функция для обучения модели
def train_model(model, train_loader, criterion, optimizer,
epochs=13):
    model.train()
    loss_history = []

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

            # Прямой проход
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            # Обратный проход и оптимизация
            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


def visualize_and_classify_image(model, dataset,
index=None):
    # Если индекс не указан, выбираем случайное изображение
    if index is None:
        index = random.randint(0, len(dataset) - 1)

    # Получаем изображение и его истинную метку
    image, label = dataset[index]

    # Переводим изображение в формат для отображения
    img = image.numpy().transpose((1, 2, 0)) * 0.5 + 0.5  #
Делаем обратную нормализацию для отображения

    # Показываем изображение
    plt.imshow(img)
    plt.title(f"True Label: {label}")
    plt.show()

    # Подготавливаем изображение для модели
    image = image.unsqueeze(0)  # Добавляем размер батча

    # Прогоняем изображение через модель
    model.eval()
```

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

    # Выводим предсказанный класс
    print(f"Predicted Class: {predicted.item()}")


# Обучаем модель и собираем историю ошибок
loss_history = train_model(model, train_loader, criterion,
optimizer, epochs=13)
```

```
# Тестируем модель на тестовой выборке
test_accuracy = test_model(model, test_loader)

# Строим график изменения ошибки во время обучения
plt.plot(loss_history)
plt.title('Training Loss Over Time')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

visualize_and_classify_image(model, test_set, index=35)
```

```
Epoch [1/13], Loss: 3.5916
Epoch [2/13], Loss: 2.7911
Epoch [3/13], Loss: 2.4020
Epoch [4/13], Loss: 2.1200
Epoch [5/13], Loss: 1.8672
Epoch [6/13], Loss: 1.6278
Epoch [7/13], Loss: 1.3893
Epoch [8/13], Loss: 1.1617
Epoch [9/13], Loss: 0.9392
Epoch [10/13], Loss: 0.7391
Epoch [11/13], Loss: 0.5741
Epoch [12/13], Loss: 0.4396
Epoch [13/13], Loss: 0.3425
Test Accuracy: 36.42%
Predicted Class: 64
```





Вывод: научился конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.