Министерство образования Республики Беларусь
Учреждение образования
"Брестский государственный технический университет"
Кафедра ИИТ

Лабораторная работа №1

По дисциплине "Обработка изображений в интеллектуальных системах"

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-24

Бузель С.Д.

Проверил:

Андренко К. В.

Брест 2025

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

Задание:

1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);

2. Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (из материалов в сети Интернет). Сделать выводы о результатах обучения СНС из п. 1;

3. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);

4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№ варианта	Выборка	Размер исходного изображения	Оптимизатор
3	CIFAR-10	32X32	SGD

Код программы:

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np

# --- Шаг 1: Загрузка и нормализация данных CIFAR-10 ---
# Преобразования для изображений: конвертация в тензоры и нормализация
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Загрузка обучающей выборки
trainset = torchvision.datasets.CIFAR10(root='D:/REALOIIS/lab1', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=0)

# Загрузка тестовой выборки
testset = torchvision.datasets.CIFAR10(root='D:/REALOIIS/lab1', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=0)
```

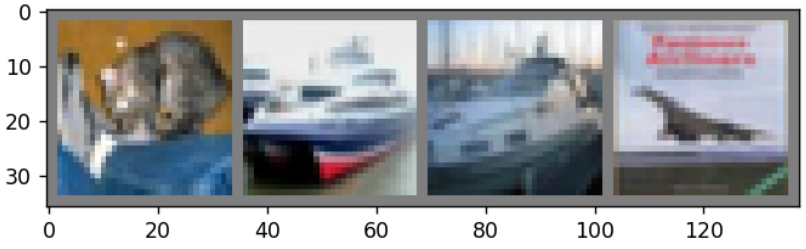
```
# Классы изображений
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# --- Шаг 2: Определение архитектуры сверточной нейронной сети ---
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Первый сверточный слой: 3 входных канала (RGB), 6 выходных, ядро 5x5
        self.conv1 = nn.Conv2d(3, 6, 5)
        # Слой подвыборки (max pooling) с окном 2x2
        self.pool = nn.MaxPool2d(2, 2)
        # Второй сверточный слой
        self.conv2 = nn.Conv2d(6, 16, 5)
        # Полносвязные слои
        self.fc1 = nn.Linear(16 * 5 * 5, 120) # 16*5*5 - размерность после conv2 и pool
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10) # 10 - количество классов

    def forward(self, x):
        # Применение слоев с функцией активации ReLU
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # Выравнивание данных для полносвязного слоя
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = SimpleCNN()

# --- Шаг 3: Определение функции потерь и оптимизатора ---
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# --- Шаг 4: Обучение нейронной сети ---
loss_history = []
epochs = 5 # Для демонстрации, можно увеличить для лучшей точности

for epoch in range(epochs):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # Получение входных данных. data - список [inputs, labels]
        inputs, labels = data

        # Обнуление градиентов
        optimizer.zero_grad()

        # Прямое распространение и обратное распространение и оптимизация
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # Сбор статистики по ошибке
        running_loss += loss.item()
        if i % 2000 == 1999:    # Вывод каждые 2000 мини-батчей
```

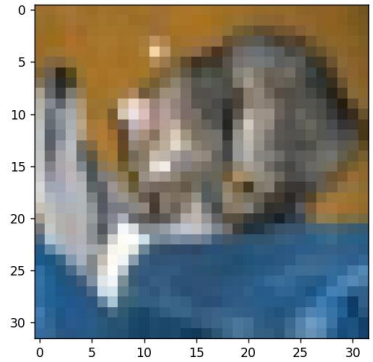
```
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            loss_history.append(running_loss / 2000)
            running_loss = 0.0

print('Обучение завершено')

# --- Шаг 5: Построение графика изменения ошибки ---
plt.figure(figsize=(10, 5))
plt.plot(loss_history)
plt.title('График изменения ошибки обучения')
plt.xlabel('Итерации (x2000)')
plt.ylabel('Ошибка (Loss)')
plt.grid(True)
plt.show()

# Шаг 6: Оценка эффективности на тестовой выборке
correct = 0
total = 0
with torch.no_grad(): # Отключаем вычисление градиентов для оценки
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Точность сети на 10000 тестовых изображений: {100 * correct // total} %')

# --- Шаг 6: Отображение изображения ---
def imshow(img):
    img = img / 2 + 0.5     # денормализация
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

# Получаем случайные изображения из тестовой выборки
dataiter = iter(testloader)
images, labels = next(dataiter)

# Показываем изображения
print("Изображения для предсказания:")
imshow(torchvision.utils.make_grid(images))

# Выводим истинные метки
print('Истинные метки: ', ' '.join(f'{classes[labels[j]]:5s}' for j in range(4)))

# Делаем предсказание
outputs = net(images)
_, predicted = torch.max(outputs, 1)

# Выводим предсказанные метки
print('Предсказания: ', ' '.join(f'{classes[predicted[j]]:5s}' for j in range(4)))

# Выбераем одно изображение для индивидуального анализа
image_index = 0
single_image = images[image_index]
true_label = labels[image_index]
```

```
# Подаем одно изображение на вход сети
with torch.no_grad():
    output = net(single_image.unsqueeze(0)) # unsqueeze(0) для добавления batch-размерности
    _, predicted_label = torch.max(output, 1)

print("\n--- Анализ одного изображения ---")
imshow(single_image)
print(f"Истинный класс: {classes[true_label]}")
print(f"Предсказанный класс: {classes[predicted_label.item()]}")
```

Точность сети на 10000 тестовых изображений: 60 %
Изображения для предсказания:
Истинные метки: cat   ship ship plane
Предсказания: cat   plane truck plane

--- Анализ одного изображения ---
Истинный класс: cat
Предсказанный класс: cat





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