Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Обработка изображений в ИС

Лабораторная работа №1

Обучение классификаторов средствами библиотеки PyTorch

Выполнил:
Студент 4-го курса
Группы ИИ-24
Терехов Н. А.
Проверила:
Андренко К. В.

Брест 2025

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

Общее 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.

19	CIFAR-100	32X32	RMSprop

Код программы:

```
import os
import argparse
from tqdm import tqdm
from PIL import Image

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as T
import torchvision.datasets as datasets
import matplotlib.pyplot as plt


class CIFAR100Classifier(nn.Module):
    def __init__(self, num_categories=100):
        super().__init__()
        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),  # 16x16
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),  # 8x8
```

```
            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),  # 4x4
        )

        self.fc_layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, num_categories)
        )

    def forward(self, x):
        x = self.conv_layers(x)
        x = self.fc_layers(x)
        return x
def run_training_epoch(network, data_loader, loss_fn, opt,
device):
    network.train()
    total_loss = 0.0
    for batch_imgs, batch_labels in tqdm(data_loader,
desc='Training', leave=False):
        batch_imgs, batch_labels = batch_imgs.to(device),
batch_labels.to(device)
        opt.zero_grad()
        predictions = network(batch_imgs)
        loss_value = loss_fn(predictions, batch_labels)
        loss_value.backward()
        opt.step()
        total_loss += loss_value.item() * batch_imgs.size(0)
    return total_loss / len(data_loader.dataset)


def test_model(network, data_loader, loss_fn, device):
    network.eval()
    total_loss = 0.0
    right_predictions = 0
    samples_count = 0
    with torch.no_grad():
        for batch_imgs, batch_labels in data_loader:
            batch_imgs, batch_labels = batch_imgs.to(device),
batch_labels.to(device)
            predictions = network(batch_imgs)
            loss_value = loss_fn(predictions, batch_labels)
            total_loss += loss_value.item() *
batch_imgs.size(0)
            predicted_classes = predictions.argmax(dim=1)
            right_predictions += (predicted_classes ==
batch_labels).sum().item()
            samples_count += batch_imgs.size(0)
    avg_loss = total_loss / len(data_loader.dataset)
    accuracy = right_predictions / samples_count
```

```
    return avg_loss, accuracy
def show_prediction(network, image_path, category_list, device,
image_dim=32):
    network.eval()
    input_image = Image.open(image_path).convert('RGB')
    preprocess = T.Compose([
        T.Resize((image_dim, image_dim)),
        T.ToTensor(),
        T.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023,
0.1994, 0.2010])
    ])
    input_tensor =
preprocess(input_image).unsqueeze(0).to(device)
    with torch.no_grad():
        output = network(input_tensor)
        probabilities = F.softmax(output,
dim=1).cpu().numpy()[0]
        top5_indices = probabilities.argsort()[-5:][::-1]

    plt.figure(figsize=(4, 4))
    plt.imshow(input_image)
    plt.axis('off')
    plt.title('Лучший результат: {} ({:.2f}%)'.format(
        category_list[top5_indices[0]],
probabilities[top5_indices[0]] * 100))
    plt.show()

    print('\nТоп-5 предсказаний:')
    for idx, class_idx in enumerate(top5_indices, 1):
        print(f"{idx}. {category_list[class_idx]}:
{probabilities[class_idx] * 100:.2f}%")


def execute_training(batch_size=128, num_epochs=15,
learning_rate=0.001, use_gpu=True, output_path='.'):
    compute_device = torch.device('cuda' if
torch.cuda.is_available() and use_gpu else 'cpu')
    print('Используемое устройство:', compute_device)

    # Преобразования данных
    train_transforms = T.Compose([
        T.RandomCrop(32, padding=4),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023,
0.1994, 0.2010])
    ])
    test_transforms = T.Compose([
        T.ToTensor(),
        T.Normalize(mean=[0.4914, 0.4822, 0.4465], std=[0.2023,
0.1994, 0.2010])
    ])
```

```
    # Загрузка данных
    training_data = datasets.CIFAR100(root='./data',
train=True, download=True, transform=train_transforms)
    validation_data = datasets.CIFAR100(root='./data',
train=False, download=True, transform=test_transforms)

    training_loader = DataLoader(training_data,
batch_size=batch_size, shuffle=True, num_workers=4,
pin_memory=True)
    validation_loader = DataLoader(validation_data,
batch_size=batch_size, shuffle=False, num_workers=4,
                                   pin_memory=True)

    category_names = training_data.classes

    # Инициализация модели
    model =
CIFAR100Classifier(num_categories=len(category_names)).to(compu
te_device)
    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.RMSprop(model.parameters(),
lr=learning_rate)

    # Трекинг метрик
    training_loss_history = []
    validation_loss_history = []
    validation_accuracy_history = []

    best_accuracy = 0.0
    os.makedirs(output_path, exist_ok=True)

    # Цикл обучения
    for epoch in range(1, num_epochs + 1):
        print(f'\nЭпоха {epoch}/{num_epochs}')

        # Обучение
        epoch_train_loss = run_training_epoch(model,
training_loader, loss_function, optimizer, compute_device)

        # Валидация
        epoch_val_loss, epoch_val_accuracy = test_model(model,
validation_loader, loss_function, compute_device)

        # Сохранение метрик
        training_loss_history.append(epoch_train_loss)
        validation_loss_history.append(epoch_val_loss)
        validation_accuracy_history.append(epoch_val_accuracy)

        print(
            f'Потери на обучении: {epoch_train_loss:.4f} |
Потери на валидации: {epoch_val_loss:.4f} | Точность:
{epoch_val_accuracy * 100:.2f}%')
```

```
        # Сохранение лучшей модели
        if epoch_val_accuracy > best_accuracy:
            best_accuracy = epoch_val_accuracy
            model_save_path = os.path.join(output_path,
'cifar100_rmsprop_model.pth')
            torch.save(model.state_dict(), model_save_path)
            print(f'Новая лучшая модель сохранена! Точность:
{best_accuracy * 100:.2f}%')

    # Визуализация результатов
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(range(1, num_epochs + 1), training_loss_history,
label='Обучение')
    plt.plot(range(1, num_epochs + 1), validation_loss_history,
label='Валидация')
    plt.xlabel('Эпоха')
    plt.ylabel('Потери')
    plt.legend()
    plt.grid(True)
    plt.title('Динамика потерь')

    plt.subplot(1, 2, 2)
    plt.plot(range(1, num_epochs + 1),
validation_accuracy_history, label='Точность', color='green')
    plt.xlabel('Эпоха')
    plt.ylabel('Точность')
    plt.legend()
    plt.grid(True)
    plt.title('Точность на валидации')

    plt.tight_layout()
    plot_file = os.path.join(output_path,
'training_results.png')
    plt.savefig(plot_file)
    print('\nГрафики сохранены в', plot_file)

    # Финальная оценка
    final_loss, final_accuracy = test_model(model,
validation_loader, loss_function, compute_device)
    print(f'\nФинальная точность: {final_accuracy * 100:.2f}%')
    print(f'Лучшая точность: {best_accuracy * 100:.2f}%')
    print(f'Модель сохранена: {model_save_path}')

    # Тест на примере изображения
    sample_image_path = os.path.join('.', 'sample_image.jpg')
    if os.path.exists(sample_image_path):
        try:
            print(f'\nТестирование на изображении:
{sample_image_path}')
            show_prediction(model, sample_image_path,
category_names, compute_device)
```

```
        except Exception as e:
            print('Ошибка при обработке изображения:', e)
    else:
        print(f"\nФайл '{sample_image_path}' не найден.")
        print("Для тестирования поместите изображение в текущую
директорию и назовите 'sample_image.jpg'")


if __name__ == '__main__':
    arg_parser = argparse.ArgumentParser(description='Обучение
классификатора CIFAR-100 с RMSprop')
    arg_parser.add_argument('--batch-size', type=int,
default=128, help='Размер батча')
    arg_parser.add_argument('--epochs', type=int, default=15,
help='Количество эпох')
    arg_parser.add_argument('--lr', type=float, default=0.001,
help='Скорость обучения для RMSprop')
    arg_parser.add_argument('--no-cuda', action='store_true',
help='Не использовать GPU')
    arg_parser.add_argument('--save-dir', type=str,
default='.', help='Директория для сохранения')
    arguments = arg_parser.parse_args()

    execute_training(
        batch_size=arguments.batch_size,
        num_epochs=arguments.epochs,
        learning_rate=arguments.lr,
        use_gpu=not arguments.no_cuda,
        output_path=arguments.save_dir
    )
```

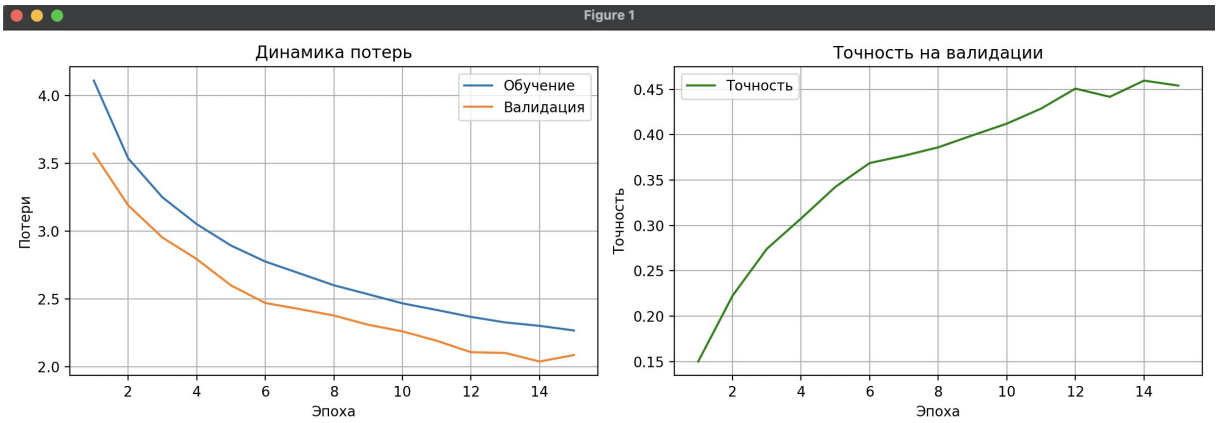
Результат работы программы:

Эпоха 14/15
Потери на обучении: 2.3018 | Потери на валидации: 2.0384 | Точность: 45.96%
Новая лучшая модель сохранена! Точность: 45.96%

Эпоха 15/15
Потери на обучении: 2.2678 | Потери на валидации: 2.0862 | Точность: 45.41%

Графики сохранены в ./training_results.png

Финальная точность: 45.41%
Лучшая точность: 45.96%
Модель сохранена: ./cifar100_rmsprop_model.pth



Тестирование на изображении:

Топ-5 предсказаний:
1. whale: 39.00%
2. dolphin: 37.51%
3. shark: 9.56%
4. turtle: 5.41%
5. seal: 5.01%

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