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»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Лящук А. В.

Проверила:

Андренко К. В.

Брест 2025

Цель: 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.

Задание по вариантам

№ варианта	Выборка	Класс	Оптимизатор
10	STL-10	96X96	Adam

Код:

```
import os
import time
from PIL import Image
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib

# Используем бэкенд, не требующий дисплея
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from torch.utils.data import Subset

# ========== ПАРАМЕТРЫ (можно менять прямо здесь) ==========
EPOCHS = 10
BATCH_SIZE = 64  # Уменьшил batch size для STL-10 (изображения больше)
LR = 0.001  # Стандартный learning rate для Adam
WEIGHT_DECAY = 1e-4
```

```
USE_CUDA = False
RESUME = True
SAVE_DIR = 'checkpoints_stl10'
VISUALIZE_IMAGE = None
DATA_RATIO = 1.0  # Использовать 100% данных
# ============================================================

# Проверяем доступность CUDA
if USE_CUDA:
    USE_CUDA = torch.cuda.is_available()
    print(f"CUDA доступна: {USE_CUDA}")

device = torch.device('cuda' if USE_CUDA else 'cpu')
print(f"Используемое устройство: {device}")
os.makedirs(SAVE_DIR, exist_ok=True)


# -----------------------
# CNN для STL-10 (изображения 96x96, 3 канала)
# -----------------------
class STL10CNN(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.features = nn.Sequential(
            # блок 1
            nn.Conv2d(3, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),  # 48x48

            # блок 2
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),  # 24x24

            # блок 3
            nn.Conv2d(128, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),  # 12x12

            # блок 4
            nn.Conv2d(256, 512, 3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, 3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2, 2),  # 6x6
        )
        self.classifier = nn.Sequential(
```

```
            nn.Flatten(),
            nn.Linear(512 * 6 * 6, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(1024, 512),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x


def main():
    print("Начинаем загрузку данных STL-10...")

    # -----------------------
    # Data (STL-10 + аугментация)
    # -----------------------
    # Стандартные значения нормализации для STL-10
    transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(96, padding=8),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
hue=0.1),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Для RGB
    ])

    transform_test = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Для RGB
    ])

    try:
        # Загружаем размеченную часть STL-10
        full_trainset = torchvision.datasets.STL10(
            root='./data',
            split='train',
            download=True,
            transform=transform_train
        )

        # Выбираем только часть данных
        num_samples = int(len(full_trainset) * DATA_RATIO)
        indices = torch.randperm(len(full_trainset))[:num_samples]
        trainset = Subset(full_trainset, indices)

        trainloader = torch.utils.data.DataLoader(
            trainset,
            batch_size=BATCH_SIZE,
            shuffle=True,
            num_workers=0
        )

        testset = torchvision.datasets.STL10(
            root='./data',
            split='test',
```

```
            download=True,
            transform=transform_test
        )
        testloader = torch.utils.data.DataLoader(
            testset,
            batch_size=BATCH_SIZE,
            shuffle=False,
            num_workers=0
        )

        print(f"Используется {num_samples} из {len(full_trainset)}
тренировочных образцов ({DATA_RATIO * 100}%)")
        print("Данные STL-10 успешно загружены!")
    except Exception as e:
        print(f"Ошибка при загрузке данных: {e}")
        return

    classes = ['airplane', 'bird', 'car', 'cat', 'deer',
               'dog', 'horse', 'monkey', 'ship', 'truck']

    # -----------------------
    # Модель/оптимизатор/критерий
    # -----------------------
    print("Инициализация модели...")
    model = STL10CNN(num_classes=10).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(  # Заменяем на Adam
        model.parameters(),
        lr=LR,
        weight_decay=WEIGHT_DECAY
    )
    scheduler = optim.lr_scheduler.CosineAnnealingLR(
        optimizer,
        T_max=EPOCHS
    )

    # -----------------------
    # Загрузка из чекпоинта (если нужно)
    # -----------------------
    start_epoch = 1
    best_acc = 0.0
    history = {'train_loss': [], 'test_loss': [], 'test_acc': []}

    if RESUME:
        checkpoint_path = os.path.join(SAVE_DIR, 'best.pth')
        if os.path.isfile(checkpoint_path):
            print(f"Загрузка модели из {checkpoint_path} ...")
            try:
                checkpoint = torch.load(checkpoint_path, map_location=device)
                model.load_state_dict(checkpoint['model_state'])
                best_acc = checkpoint.get('acc', 0.0)
                start_epoch = checkpoint.get('epoch', 0) + 1
                # Загружаем историю если есть
                if 'history' in checkpoint:
                    history = checkpoint['history']
                print(f"Модель загружена. Лучший acc={best_acc:.2f}% (эпоха
{start_epoch - 1})")
                print(f"История загружена: {len(history['train_loss'])}
эпох")
            except Exception as e:
                print(f"Ошибка при загрузке чекпоинта: {e}")
```

```
        else:
            print("Чекпоинт не найден, начинаем обучение с нуля.")

    # -----------------------
    # Функция валидации
    # -----------------------
    def evaluate(loader):
        model.eval()
        correct = 0
        total = 0
        running_loss = 0.0
        with torch.no_grad():
            for inputs, targets in loader:
                inputs, targets = inputs.to(device), targets.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, targets)
                running_loss += loss.item() * inputs.size(0)
                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()
        return running_loss / total, 100.0 * correct / total

    # -----------------------
    # Обучение
    # -----------------------
    print("Начинаем обучение...")
    start_time = time.time()

    # Проверяем, нужно ли проводить обучение
    if start_epoch <= EPOCHS:
        for epoch in range(start_epoch, EPOCHS + 1):
            model.train()
            running_loss = 0.0
            for i, (inputs, targets) in enumerate(trainloader, 1):
                inputs, targets = inputs.to(device), targets.to(device)
                optimizer.zero_grad()
                outputs = model(inputs)
                loss = criterion(outputs, targets)
                loss.backward()
                optimizer.step()
                running_loss += loss.item() * inputs.size(0)

                if i % 20 == 0:  # Увеличил интервал вывода из-за меньшего
количества батчей
                    print(
                        f'Эпоха {epoch}, Батч {i}/{len(trainloader)}, Loss:
{loss.item():.4f}, Время: {time.time() - start_time:.2f}c')

            train_loss = running_loss / len(trainloader.dataset)
            test_loss, test_acc = evaluate(testloader)
            scheduler.step()

            history['train_loss'].append(train_loss)
            history['test_loss'].append(test_loss)
            history['test_acc'].append(test_acc)

            print(
                f'Epoch {epoch}/{EPOCHS}  TrainLoss={train_loss:.4f}
TestLoss={test_loss:.4f}  TestAcc={test_acc:.2f}%')

            if test_acc > best_acc:
```

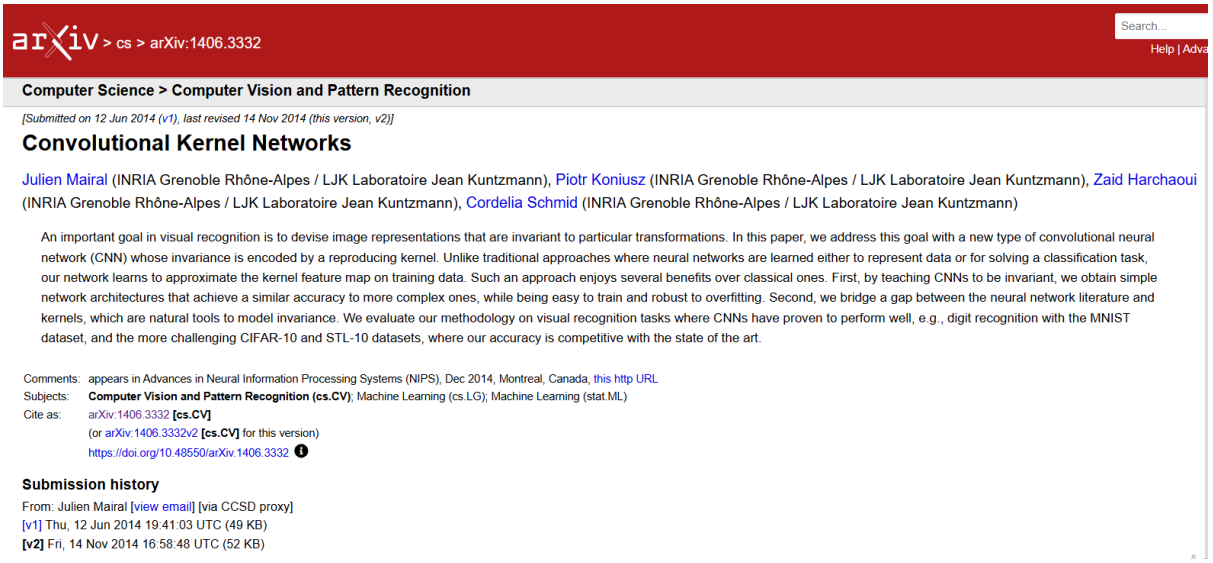
```
                best_acc = test_acc
                checkpoint = {
                    'model_state': model.state_dict(),
                    'acc': best_acc,
                    'epoch': epoch,
                    'history': history  # Сохраняем историю вместе с моделью
                }
                torch.save(checkpoint, os.path.join(SAVE_DIR, 'best.pth'))
                print(f"Новая лучшая модель сохранена с точностью
{best_acc:.2f}%")

        total_time = time.time() - start_time
        print(f'Обучение завершено за {total_time / 60:.2f} минут. Лучшая
точность: {best_acc:.2f}%')
    else:
        print(f"Пропускаем обучение, так как начальная эпоха {start_epoch}
превышает EPOCHS {EPOCHS}")

    # -----------------------
    # График изменения ошибки
    # -----------------------
    if history['train_loss'] and history['test_loss']:
        print("Создание графика изменения ошибки...")

        # Создаем простой текстовый файл с данными для отладки
        debug_path = os.path.join(SAVE_DIR, 'loss_data.txt')
        with open(debug_path, 'w') as f:
            f.write("Epoch,Train_Loss,Test_Loss\n")
            for i in range(len(history['train_loss'])):
                f.write(f"{i +
1},{history['train_loss'][i]},{history['test_loss'][i]}\n")
        print(f"Данные для отладки сохранены в {debug_path}")

        # Создаем график с минимальными настройками
        try:
            # Создаем новую фигуру
            fig, ax = plt.subplots(figsize=(10, 6))

            # Данные для оси X (эпохи)
            epochs = range(1, len(history['train_loss']) + 1)

            # Простые линии без сложного форматирования
            ax.plot(epochs, history['train_loss'], 'b-', linewidth=2,
label='Train Loss')
            ax.plot(epochs, history['test_loss'], 'r-', linewidth=2,
label='Test Loss')

            # Простые подписи на английском
            ax.set_xlabel('Epoch')
            ax.set_ylabel('Loss')
            ax.legend()
            ax.set_title('Training and Test Loss - STL-10 with Adam')

            # Сохраняем разными способами для надежности
            loss_path_png = os.path.join(SAVE_DIR, 'training_loss.png')
            loss_path_pdf = os.path.join(SAVE_DIR, 'training_loss.pdf')

            plt.savefig(loss_path_png, dpi=100, bbox_inches='tight')
            plt.savefig(loss_path_pdf, bbox_inches='tight')
            plt.close(fig)  # Явно закрываем фигуру
```

```
            print(f'График ошибки сохранен в {loss_path_png} и
{loss_path_pdf}')

        except Exception as e:
            print(f"Ошибка при создании графика: {e}")
            # Альтернативный способ - сохранить данные в CSV
            csv_path = os.path.join(SAVE_DIR, 'loss_data.csv')
            try:
                import pandas as pd
                df = pd.DataFrame({
                    'epoch': range(1, len(history['train_loss']) + 1),
                    'train_loss': history['train_loss'],
                    'test_loss': history['test_loss']
                })
                df.to_csv(csv_path, index=False)
                print(f"Данные сохранены в CSV: {csv_path}")
            except:
                print("Не удалось сохранить данные в CSV")

        # Выводим финальные значения ошибок
        print(f"Финальная ошибка обучения: {history['train_loss'][-1]:.4f}")
        print(f"Финальная ошибка тестирования:
{history['test_loss'][-1]:.4f}")
    else:
        print("Нет данных для построения графика ошибки")

    # -----------------------
    # Визуализация предсказания для отдельного изображения (опционально)
    # -----------------------
    def predict_image(img_path):
        img = Image.open(img_path).convert('RGB').resize((96, 96))  # STL-10:
цветные 96x96
        x = transform_test(img).unsqueeze(0).to(device)
        model.eval()
        with torch.no_grad():
            logits = model(x)
            probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
            pred = int(np.argmax(probs))
        return img, pred, probs

    if VISUALIZE_IMAGE and os.path.exists(VISUALIZE_IMAGE):
        print(f"Визуализация изображения: {VISUALIZE_IMAGE}")
        img, pred_idx, probs = predict_image(VISUALIZE_IMAGE)
        plt.figure(figsize=(4, 4))
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Prediction: {classes[pred_idx]}\nConfidence:
{probs[pred_idx] * 100:.1f}%')

        single_pred_path = os.path.join(SAVE_DIR, 'single_prediction.png')
        plt.savefig(single_pred_path, dpi=150, bbox_inches='tight')
        plt.close()
        print(f"Визуализация предсказания сохранена в {single_pred_path}")

    # -----------------------
    # Финальная оценка модели
    # -----------------------
    print("Финальная оценка модели на тестовом наборе...")
    final_test_loss, final_test_acc = evaluate(testloader)
    print(f"Финальные результаты - Loss: {final_test_loss:.4f}, Accuracy:
{final_test_acc:.2f}%")
```

```
if __name__ == "__main__":
    main()
    print("Программа завершена!")
```

Вывод:

State-of-art:



arXiv > cs > arXiv:1406.3332

Search... Help | Adva

Computer Science > Computer Vision and Pattern Recognition

[Submitted on 12 Jun 2014 (v1), last revised 14 Nov 2014 (this version, v2)]

Convolutional Kernel Networks

Julien Mairal (INRIA Grenoble Rhône-Alpes / LJK Laboratoire Jean Kuntzmann), Piotr Koniusz (INRIA Grenoble Rhône-Alpes / LJK Laboratoire Jean Kuntzmann), Zaid Harchaoui (INRIA Grenoble Rhône-Alpes / LJK Laboratoire Jean Kuntzmann), Cordelia Schmid (INRIA Grenoble Rhône-Alpes / LJK Laboratoire Jean Kuntzmann)

An important goal in visual recognition is to devise image representations that are invariant to particular transformations. In this paper, we address this goal with a new type of convolutional neural network (CNN) whose invariance is encoded by a reproducing kernel. Unlike traditional approaches where neural networks are learned either to represent data or for solving a classification task, our network learns to approximate the kernel feature map on training data. Such an approach enjoys several benefits over classical ones. First, by teaching CNNs to be invariant, we obtain simple network architectures that achieve a similar accuracy to more complex ones, while being easy to train and robust to overfitting. Second, we bridge a gap between the neural network literature and kernels, which are natural tools to model invariance. We evaluate our methodology on visual recognition tasks where CNNs have proven to perform well, e.g., digit recognition with the MNIST dataset, and the more challenging CIFAR-10 and STL-10 datasets, where our accuracy is competitive with the state of the art.

Comments: appears in Advances in Neural Information Processing Systems (NIPS), Dec 2014, Montreal, Canada, this http URL
Subjects: **Computer Vision and Pattern Recognition (cs.CV)**; Machine Learning (cs.LG); Machine Learning (stat.ML)
Cite as: arXiv:1406.3332 **[cs.CV]**
(or arXiv:1406.3332v2 **[cs.CV]** for this version)
https://doi.org/10.48550/arXiv.1406.3332

Submission history

From: Julien Mairal [view email] [via CCSD proxy]
[v1] Thu, 12 Jun 2014 19:41:03 UTC (49 KB)
[v2] Fri, 14 Nov 2014 16:58:48 UTC (52 KB)

[Ссылка на статью]

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