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
Поддубный Ю. 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.

№ варианта	Выборка	Размер исходного изображения	Оптимизатор
14	CIFAR-100	32X32	Adadelta

Код программы:

```
import torch import 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 SimpleCIFAR100CNN(nn.Module):
    def __init__(self, num_classes=100):
        super().__init__()
        self.features = 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.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 4 * 4, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

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


def train_one_epoch(model, loader, criterion, optimizer,
device):
    model.train()
    running_loss = 0.0
    for images, targets in tqdm(loader, desc='Train batches',
leave=False):
        images, targets = images.to(device),
targets.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * images.size(0)
    return running_loss / len(loader.dataset)


def evaluate(model, loader, criterion, device):
    model.eval()
    running_loss = 0.0
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for images, targets in loader:
            images, targets = images.to(device),
targets.to(device)
            outputs = model(images)
            loss = criterion(outputs, targets)
            running_loss += loss.item() * images.size(0)
            preds = outputs.argmax(dim=1)
            correct += (preds == targets).sum().item()
            total += images.size(0)
    avg_loss = running_loss / len(loader.dataset)
    acc = correct / total
    return avg_loss, acc


def visualize_image_prediction(model, img_path, class_names,
device, img_size=32):
    model.eval()
    img = Image.open(img_path).convert('RGB')
    transform = T.Compose([
        T.Resize((img_size, img_size)),
        T.ToTensor(),
        T.Normalize(mean=[0.4914, 0.4822, 0.4465],
std=[0.2023, 0.1994, 0.2010])
    ])
    input_tensor = transform(img).unsqueeze(0).to(device)
    with torch.no_grad():
        outputs = model(input_tensor)
        probs = F.softmax(outputs, dim=1).cpu().numpy()[0]
        top5_idx = probs.argsort()[-5:][::-1]

    plt.figure(figsize=(4,4))
    plt.imshow(img)
    plt.axis('off')
    plt.title('Top-1: {}
({:.2f}%)'.format(class_names[top5_idx[0]],
probs[top5_idx[0]]*100))
    plt.show()

    print('\nTop-5 predictions:')
    for i in top5_idx:
        print(f"{class_names[i]}: {probs[i]*100:.2f}%")


def main(batch_size=128, epochs=30, lr=1.0, use_cuda=True,
```

```
save_dir='.'):
    device = torch.device('cuda' if torch.cuda.is_available()
and use_cuda else 'cpu')
    print('Device:', device)

    transform_train = 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])
    ])
    transform_test = T.Compose([
        T.ToTensor(),
        T.Normalize(mean=[0.4914, 0.4822, 0.4465],
std=[0.2023, 0.1994, 0.2010])
    ])

    train_set = datasets.CIFAR100(root='./data', train=True,
download=True, transform=transform_train)
    test_set = datasets.CIFAR100(root='./data', train=False,
download=True, transform=transform_test)

    train_loader = DataLoader(train_set,
batch_size=batch_size, shuffle=True, num_workers=4,
pin_memory=True)
    test_loader = DataLoader(test_set, batch_size=batch_size,
shuffle=False, num_workers=4, pin_memory=True)

    class_names = train_set.classes

    # Модель, критерий, оптимизатор Adadelta
    model =
SimpleCIFAR100CNN(num_classes=len(class_names)).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adadelta(model.parameters(), lr=lr)

    train_losses = []
    test_losses = []
    test_accs = []

    best_acc = 0.0
    os.makedirs(save_dir, exist_ok=True)

    for epoch in range(1, epochs+1):
        print(f'\nEpoch {epoch}/{epochs}')
        train_loss = train_one_epoch(model, train_loader,
```

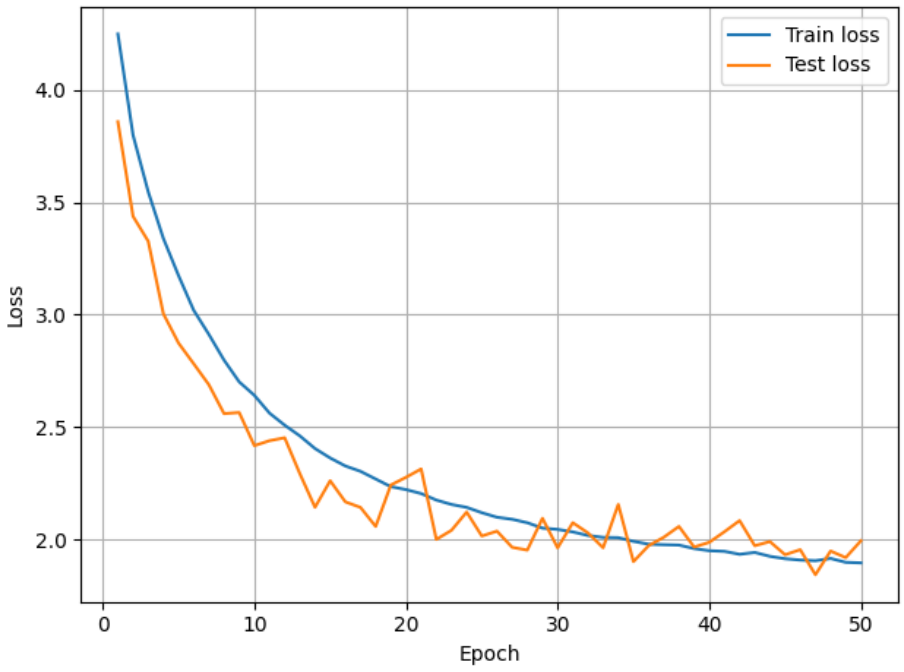
```
criterion, optimizer, device)
        test_loss, test_acc = evaluate(model, test_loader,
criterion, device)

        train_losses.append(train_loss)
        test_losses.append(test_loss)
        test_accs.append(test_acc)

        print(f'Train loss: {train_loss:.4f} | Test loss:
{test_loss:.4f} | Test acc: {test_acc*100:.2f}%')

        if test_acc > best_acc:
            best_acc = test_acc
            torch.save(model.state_dict(),
os.path.join(save_dir, 'model_cifar100_adadelta.pth'))

    plt.figure()
    plt.plot(range(1, epochs+1), train_losses, label='Train
loss')
    plt.plot(range(1, epochs+1), test_losses, label='Test
loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plot_path = os.path.join(save_dir, 'train_plot.png')
    plt.savefig(plot_path)
    print('Saved loss plot to', plot_path)

    final_loss, final_acc = evaluate(model, test_loader,
criterion, device)
    print(f'Final test accuracy: {final_acc*100:.2f}%')
    print('Model saved to', os.path.join(save_dir,
'model_cifar100_adadelta.pth'))

    example_path = os.path.join('.', 'example.jpg')
    if os.path.exists(example_path):
        try:
            visualize_image_prediction(model, example_path,
class_names, device)
        except Exception as e:
            print('Ошибка при визуализации:', e)
    else:
        print("Файл 'example.jpg' не найден в текущей
директории. Поместите изображение для теста и повторно
запустите скрипт, или используйте функцию
```

```
visualize_image_prediction отдельно.")


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--batch-size', type=int, default=128)
    parser.add_argument('--epochs', type=int, default=5)
    parser.add_argument('--lr', type=float, default=1.0,
help='Initial lr for Adadelta')
    parser.add_argument('--no-cuda', action='store_true')
    parser.add_argument('--save-dir', type=str, default='.')
    args = parser.parse_args()

    main(batch_size=args.batch_size, epochs=args.epochs,
lr=args.lr, use_cuda=not args.no_cuda, save_dir=args.save_dir)
```

Результат работы программы:

Epoch 50/50
Batch-size 256
Train loss: 1.8967 | Test loss: 1.9950 | Test acc: 48.89%
Final test accuracy: 48.89%

Top-5 predictions:
cup: 20.24%
bed: 11.05%
table: 6.49%
baby: 4.75%
wardrobe: 4.63%



State-of-the-art результаты для Cifar100: Простая мелкая CNN, обычно даёт существенно более низкую точность, чем современные SOTA (которые часто используют глубокие остаточные сети, масштабирование, сложные аугментации и/или предобучение). Для небольшой CNN на CIFAR-100 разумные ожидания — точность в пределах десятков процентов (в реальности обычно 40–70% в зависимости от настроек и эпох). Конкретные высокие числа в литературе встречаются (например, работы типа PyramidNet+ShakeDrop и результаты крупных ансамблей/масштабных моделей дают очень высокие значения точности). В научных статьях и докладах 2024–2025 появляются новые методы (например, методы дифференцированной обучения/усиления, distillation, а также масштабирование трансформеров и foundation-models), которые увеличивают точность ещё сильнее.

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