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

Обработка изображений в ИС

Лабораторная работа №2

Конструирование моделей на базе предобученных нейронных сетей

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

Брест-2025

Цель работы: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

Общее задание:
1. Для заданной выборки и архитектуры предобученной нейронной организовать процесс обучения НС, предварительно изменив структуру слоев, в соответствии с предложенной выборкой. Использовать тот же оптимизатор, что и в ЛР №1. Построить график изменения ошибки и оценить
эффективность обучения на тестовой выборке;
2. Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;
3. Ознакомиться с state-of-the-art результатами для предлагаемых выборок (по материалам в сети Интернет). Сделать выводы о результатах обучения НС из п. 1 и 2;
4. Реализовать визуализацию работы предобученной СНС и кастомной (из ЛР 1). Визуализация осуществляется посредством выбора и подачи на сеть произвольного изображения (например, из сети Интернет) с отображением результата классификации;
5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

№ варианта	Выборка	Предобученная архитектура	Оптимизатор
14	CIFAR-100	SqueezeNet 1.1	Adadelta

Код программы:

```
import os
import argparse
from tqdm import tqdm
from PIL import Image
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision.transforms as T
import torchvision.datasets as datasets
import torchvision.models as models
import matplotlib.pyplot as plt
import torch.nn.functional as F
```

```
def get_squeezenet(num_classes=100, pretrained=True,
freeze_features=False, device='cpu'):
    try:
        weights = models.SqueezeNet1_1_Weights.DEFAULT if
pretrained else None
        model = models.squeezenet1_1(weights=weights)
    except Exception:
        model = models.squeezenet1_1(pretrained=pretrained)

    final_conv = nn.Conv2d(512, num_classes, kernel_size=1)
    model.classifier = nn.Sequential(
        nn.Dropout(p=0.5),
        final_conv,
        nn.ReLU(inplace=True),
        nn.AdaptiveAvgPool2d((1, 1))
    )

    if freeze_features:
        for param in model.features.parameters():
            param.requires_grad = False

    return model.to(device)

def train_one_epoch(model, loader, criterion, optimizer,
device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 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)
        outputs = outputs.view(outputs.size(0), -1)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * images.size(0)
        preds = outputs.argmax(dim=1)
        correct += (preds == targets).sum().item()
        total += images.size(0)
    return running_loss / len(loader.dataset), correct / total
```

```
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)
            outputs = outputs.view(outputs.size(0), -1)
            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)
    return running_loss / len(loader.dataset), correct / total


def visualize_image_prediction(model, img_path, class_names,
device, img_size=224):
    model.eval()
    img = Image.open(img_path).convert('RGB')
    transform = T.Compose([
        T.Resize((img_size, img_size)),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
    ])
    input_tensor = transform(img).unsqueeze(0).to(device)
    with torch.no_grad():
        outputs = model(input_tensor)
        outputs = outputs.view(outputs.size(0), -1)
        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(f"Top-1: {class_names[top5_idx[0]]}
({probs[top5_idx[0]]*100:.2f}%)")
    plt.show()

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


def main(batch_size=64, epochs=20, lr=1.0,
```

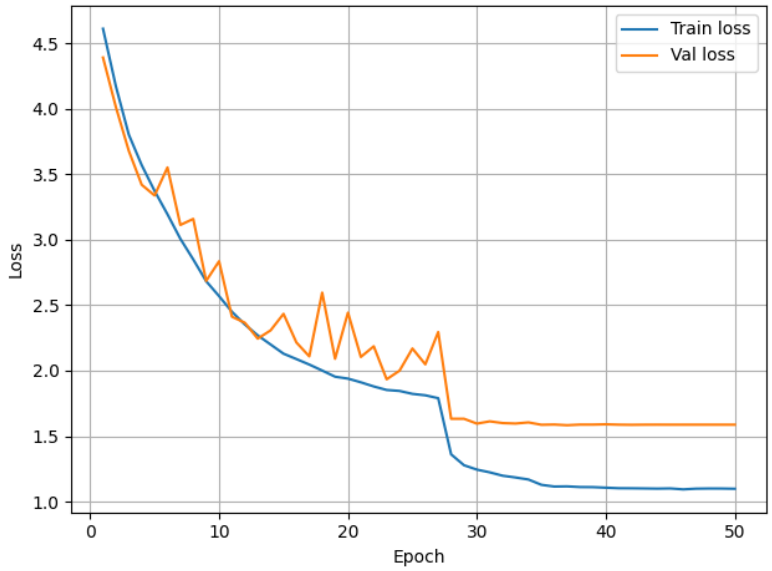
```
freeze_features=False, use_cuda=True, save_dir='.'):
    device = torch.device('cuda' if torch.cuda.is_available()
and use_cuda else 'cpu')
    print('Device:', device)

    transform_train = T.Compose([
        T.Resize((224,224)),
        T.RandomHorizontalFlip(),
        T.RandomCrop(224, padding=4),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
    ])
    transform_test = T.Compose([
        T.Resize((224,224)),
        T.ToTensor(),
        T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
    ])

    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

    model = get_squeezenet(num_classes=len(class_names),
pretrained=True, freeze_features=freeze_features,
device=device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adadelta(filter(lambda p:
p.requires_grad, model.parameters()), lr=lr)

    scheduler =
optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode="min",
patience=3, factor=0.1)

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

    train_losses = []
    val_losses = []
```

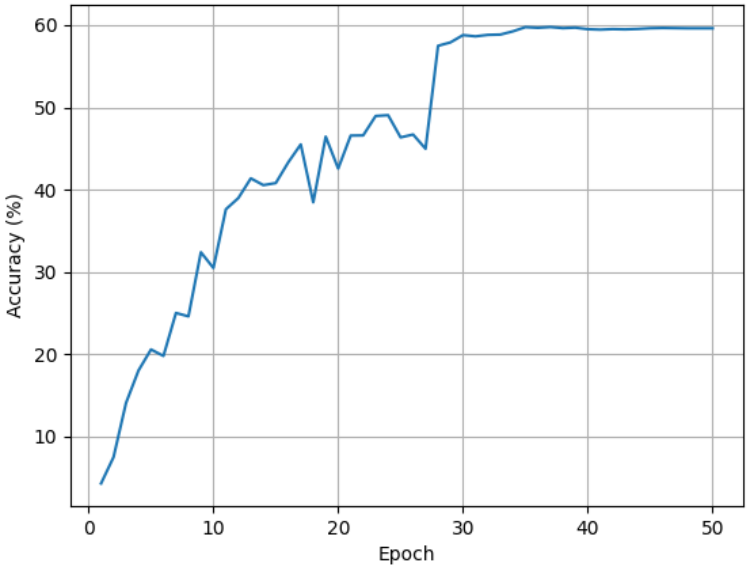
```
    val_accs = []

    for epoch in range(1, epochs+1):
        print(f"\nEpoch {epoch}/{epochs}")
        train_loss, train_acc = train_one_epoch(model,
train_loader, criterion, optimizer, device)
        val_loss, val_acc = evaluate(model, test_loader,
criterion, device)

        train_losses.append(train_loss)
        val_losses.append(val_loss)
        val_accs.append(val_acc)

        print(f"Train loss: {train_loss:.4f}, Train acc:
{train_acc*100:.2f}%")
        print(f"Val loss:   {val_loss:.4f}, Val acc:
{val_acc*100:.2f}%")

        scheduler.step(val_loss)

        if val_acc > best_acc:
            best_acc = val_acc
            torch.save(model.state_dict(),
os.path.join(save_dir, 'squeezenet_cifar100_adadelta.pth'))

    plt.figure()
    plt.plot(range(1, epochs+1), train_losses, label='Train
loss')
    plt.plot(range(1, epochs+1), val_losses, label='Val loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(os.path.join(save_dir, 'loss_plot.png'))

    plt.figure()
    plt.plot(range(1, epochs+1), [a*100 for a in val_accs],
label='Val Acc (%)')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.grid(True)
    plt.savefig(os.path.join(save_dir, 'acc_plot.png'))

    print('Best val acc:', best_acc)
    print('Model saved to', os.path.join(save_dir,
'squeezenet_cifar100_adadelta.pth'))

    example_path = os.path.join('.', 'example.jpg')
```

```
    if os.path.exists(example_path):
        visualize_image_prediction(model, example_path,
class_names, device)
    else:
        print("Put 'example.jpg' in the current folder to run
a sample prediction.")


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--batch-size', type=int, default=128)
    parser.add_argument('--epochs', type=int, default=30)
    parser.add_argument('--lr', type=float, default=1.0)
    parser.add_argument('--freeze-features',
action='store_true')
    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, freeze_features=args.freeze_features, use_cuda=not
args.no_cuda, save_dir=args.save_dir)
```

Результат работы программы:

Epoch 50/50
Train loss: 1.0982, Train acc: 68.94%
Val loss: 1.5874, Val acc: 59.61%
Best val acc: 0.5976





State-of-the-art результаты для Cifar100: По данным из интернета предобученный SqueezeNet 1.1 на CIFAR-100 показывает топ-1 accuracy около 62.2% и F1-меру 0.62, что уступает более сложным современным моделям по качеству, но превосходит их по скорости и компактности. Модель занимает всего около 2.95 МБ и отличается быстрой инференцией, что важно для задач на устройствах с ограниченными ресурсами

Вывод: осуществил обучение НС, сконструированных на базе предобученных архитектур НС.