`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе 1

Специальность ИИ-23

Выполнил:

Гавришук В.Р.

Студент группы ИИ-23

Проверил:

Андренко К. В.
Преподаватель-стажёр
Кафедры ИИТ,
«___» _____2025 г.

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	Размер исходного изображения	Оптимизатор
4	CIFAR-100	32x32	SGD

Ход работы:

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


class ImprovedCNN(nn.Module):
    def __init__(self):
        super(ImprovedCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
```

`

```
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)

        self.fc1 = nn.Linear(256 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 100)

    def forward(self, x):
        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = x.view(-1, 256 * 4 * 4)
        x = self.dropout(F.relu(self.fc1(x)))
        x = self.fc2(x)
        return x


def denormalize(tensor):
    return tensor * 0.5 + 0.5


def show_predictions(net, testset, classes, device, n=8):
    net.eval()
    indices = random.sample(range(len(testset)), n)
    imgs, trues = [], []
    for idx in indices:
        img, label = testset[idx]
        imgs.append(img)
        trues.append(label)

    batch = torch.stack(imgs).to(device)
    with torch.no_grad():
        outputs = net(batch)
        _, preds = outputs.max(1)
        preds = preds.cpu().tolist()

    ncols = 4
    nrows = (n + ncols - 1) // ncols
    plt.figure(figsize=(3 * ncols, 3 * nrows))
    for i in range(n):
        ax = plt.subplot(nrows, ncols, i + 1)
        img = denormalize(imgs[i]).cpu()
        npimg = np.transpose(img.numpy(), (1, 2, 0))
        ax.imshow(npimg)
        ax.axis("off")
        true_label = classes[trues[i]]
        pred_label = classes[preds[i]]
        color = "green" if trues[i] == preds[i] else "red"
        ax.set_title(f"Pred: {pred_label}\nTrue: {true_label}", color=color, fontsize=9)
    plt.tight_layout()
    plt.show()


if __name__ == "__main__":
    transform_train = transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomCrop(32, padding=4),
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])

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

    batch_size = 64
```

`

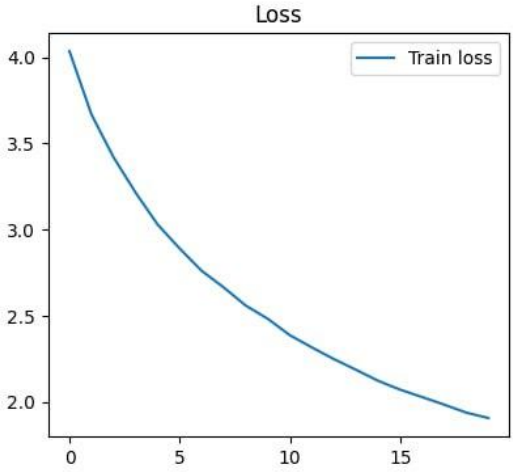
```
    trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                            download=True, transform=transform_train)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                             shuffle=True, num_workers=0)

    testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                           download=True, transform=transform_test)
    testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                            shuffle=False, num_workers=0)

    classes = trainset.classes

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    net = ImprovedCNN().to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-4)

    num_epochs = 20
    train_losses = []
    test_accuracies = []

    for epoch in range(num_epochs):
        net.train()
        running_loss = 0.0
        for inputs, labels in trainloader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        net.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for images, labels in testloader:
                images, labels = images.to(device), labels.to(device)
                outputs = net(images)
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        acc = 100 * correct / total
        test_accuracies.append(acc)
        train_losses.append(running_loss / len(trainloader))
        print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {train_losses[-1]:.3f}, Test acc:
{acc:.2f}%")

    print("Training finished!")

    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(train_losses, label="Train loss")
    plt.title("Loss")
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(test_accuracies, label="Test accuracy", color="green")
    plt.title("Accuracy")
    plt.legend()
    plt.show()
```

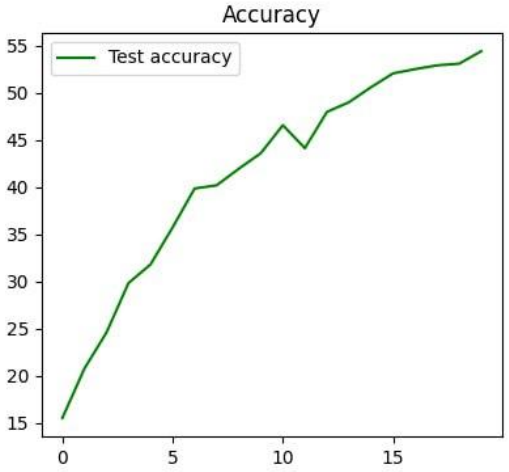
`

```
show_predictions(net, testset, classes, device, n=8)
```

Результат работы программы:

```
Epoch 1/20, Loss: 4.035, Test acc: 15.53%
Epoch 2/20, Loss: 3.668, Test acc: 20.77%
Epoch 3/20, Loss: 3.420, Test acc: 24.59%
Epoch 4/20, Loss: 3.215, Test acc: 29.84%
Epoch 5/20, Loss: 3.030, Test acc: 31.82%
Epoch 6/20, Loss: 2.891, Test acc: 35.74%
Epoch 7/20, Loss: 2.761, Test acc: 39.87%
Epoch 8/20, Loss: 2.665, Test acc: 40.21%
Epoch 9/20, Loss: 2.559, Test acc: 41.96%
Epoch 10/20, Loss: 2.483, Test acc: 43.61%
Epoch 11/20, Loss: 2.388, Test acc: 46.58%
Epoch 12/20, Loss: 2.317, Test acc: 44.15%
Epoch 13/20, Loss: 2.249, Test acc: 47.99%
Epoch 14/20, Loss: 2.187, Test acc: 49.02%
Epoch 15/20, Loss: 2.123, Test acc: 50.61%
Epoch 16/20, Loss: 2.072, Test acc: 52.08%
Epoch 17/20, Loss: 2.029, Test acc: 52.53%
Epoch 18/20, Loss: 1.985, Test acc: 52.93%
Epoch 19/20, Loss: 1.939, Test acc: 53.11%
Epoch 20/20, Loss: 1.908, Test acc: 54.43%
Training finished!
```





`

Сравнение с SOTA:

https://medium.com/data-science/cifar-100-transfer-learning-using-efficientnet-ed3ed7b89af2 - точность составляет 81.79%

Разница в точности обусловлена тем, что модели SOTA намного глубже, используют больше фильтров, продвинутые методы (такие как: аугментация – AutoAugment, RandAugment; регуляризация – MixUp, CutMix; оптимизаторы – Adamw, SAM), предобучение или дополнительные данные, а также длительное обучение и ресурсы. В то время как наша модель – простая, с ограниченным числом параметров.

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