`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Отчет по лабораторной работе 2

Специальность ИИ-23

Выполнил:

Макаревич Н.Р.

Студент группы ИИ-23

Проверил:

Андренко К. В.
Преподаватель-стажёр
Кафедры ИИТ,
«___» _____ 2025 г.

Брест 2025

`

Цель: 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.

Код программы:

```
import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

from torchvision import models

import matplotlib.pyplot as plt

from tqdm import tqdm

import multiprocessing
```

`

```
def 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))
    ])

    trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform_train)
    trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                              shuffle=True, num_workers=4)

    testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform_test)
    testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                             shuffle=False, num_workers=4)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

`

```
model = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
model.maxpool = nn.Identity()
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 10)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

num_epochs = 10
train_loss = []
accuracy_graph = []

for epoch in range(num_epochs):
    print(f"\nЭпоха {epoch + 1}/{num_epochs}")
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    progress_bar = tqdm(trainloader, desc=f"Обучение эпохи {epoch + 1}", leave=False)

    for inputs, labels in progress_bar:
        inputs, labels = inputs.to(device), labels.to(device)
```

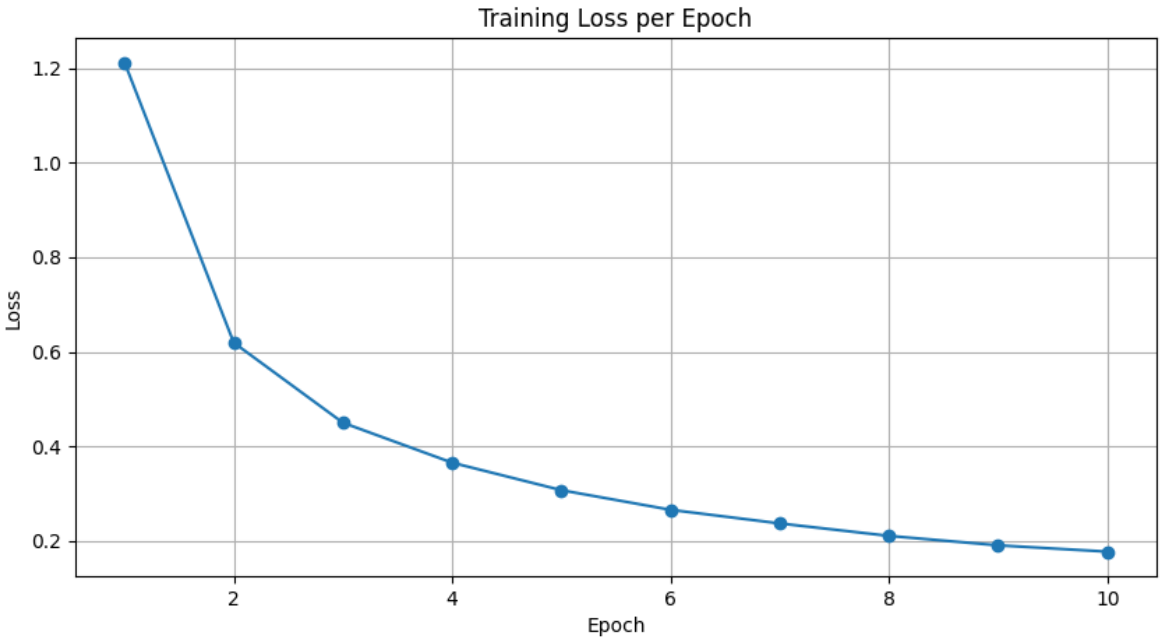
```
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

            progress_bar.set_postfix({
                "loss": f"{loss.item():.4f}",
                "acc": f"{100 * correct / total:.2f}%"
            })

        epoch_loss = running_loss / len(trainloader)
        accuracy = 100 * correct / total
        train_loss.append(epoch_loss)
        accuracy_graph.append(accuracy)

        print(f"Эпоха [{epoch + 1}/{num_epochs}] | Потеря: {epoch_loss:.4f} | Точность: {accuracy:.2f}%")

    plt.figure(figsize=(10, 5))
    plt.plot(range(1, num_epochs + 1), accuracy_graph, marker='o')
    plt.title('Training accuracy per Epoch')
```

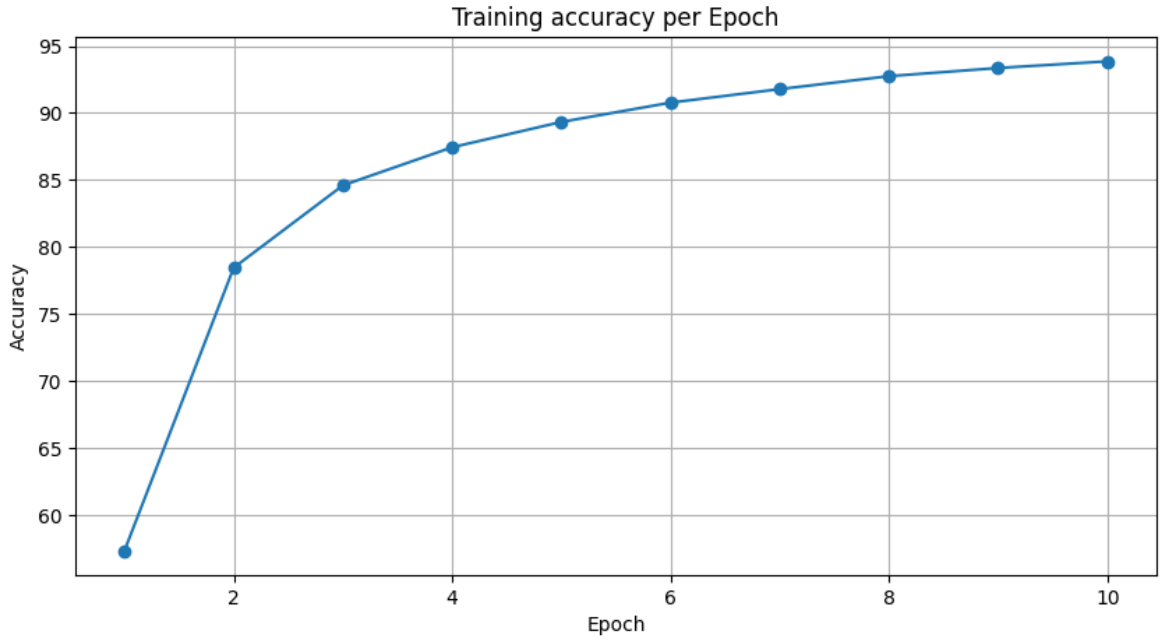
`

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 5))
plt.plot(range(1, num_epochs + 1), train_loss, marker='o')
plt.title('Training Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.show()

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

accuracy = 100 * correct / total
```

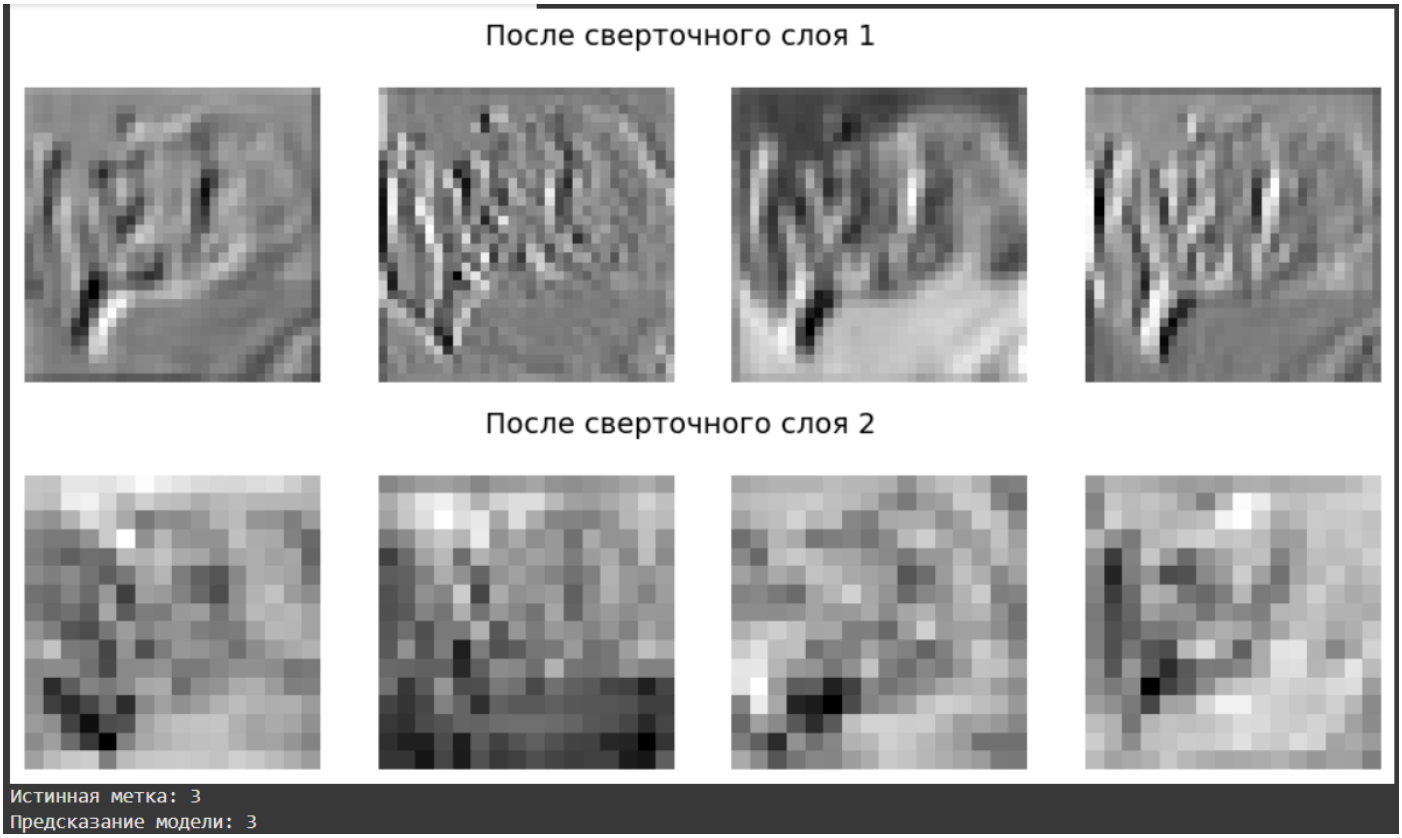
`

```
    print(f'\nAccuracy on CIFAR-10 test set: {accuracy:.2f}%')


if __name__ == '__main__':
    multiprocessing.freeze_support()
    main()
```





`



Accuracy on CIFAR-10 test set: 92.03%

Вывод:

Научился конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения. Получил значение точности 74.07% с простой моделью, 92.03% с предобученной моделью при значении state-of-art 92.38%.