МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ИИ(з)

Выполнил А. Ю. Кураш,

студент группы ИИ-24

Проверил Андренко К.В,

Преподаватель-стажер кафедры ИИТ,

« k 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.

# Выполнение:

Код программы

import os

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

import torchvision

import torchvision.transforms as transforms

from torchvision import models

import matplotlib.pyplot as plt

import numpy as np

from PIL import Image

# --- параметры ---

batch\_size = 128

num\_epochs = 10

lr = 1e-3

num\_classes = 10

mean = (0.4914, 0.4822, 0.4465)

std  = (0.2470, 0.2435, 0.2616)

# путь к модели из ЛР1

custom\_model\_path = r"C:\Users\User\OneDrive\Desktop\IP\_AI\_24\reports\Kurash\lab1\cifar\_simple\_cnn.pth"

# --- архитектура кастомной CNN (ЛР1) ---

class SimpleCNN(nn.Module):

    def \_\_init\_\_(self, num\_classes=10):

        super().\_\_init\_\_()

        self.features = nn.Sequential(

            nn.Conv2d(3, 32, kernel\_size=3, padding=1),

            nn.BatchNorm2d(32),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(2),

            nn.Conv2d(32, 64, kernel\_size=3, padding=1),

            nn.BatchNorm2d(64),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(2),

            nn.Conv2d(64, 128, kernel\_size=3, padding=1),

            nn.BatchNorm2d(128),

            nn.ReLU(inplace=True),

            nn.MaxPool2d(2),

        )

        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

# --- функция оценки точности ---

def evaluate(model, loader, device):

    model.eval()

    correct, total = 0, 0

    with torch.no\_grad():

        for X, y in loader:

            X, y = X.to(device), y.to(device)

            out = model(X)

            preds = out.argmax(dim=1)

            correct += (preds == y).sum().item()

            total += y.size(0)

    return correct / total

# --- денормализация ---

def denormalize(img\_tensor, mean, std):

    img = img\_tensor.clone().cpu().numpy()

    img = img \* np.array(std)[:, None, None] + np.array(mean)[:, None, None]

    img = np.clip(img, 0, 1)

    return img

# --- визуализация ---

def visualize\_predictions(model, loader, classes, mean, std, device, fname="pred\_grid.png"):

    model.eval()

    Xb, yb = next(iter(loader))

    Xb = Xb.to(device)

    with torch.no\_grad():

        out = model(Xb)

        preds = out.argmax(dim=1).cpu().numpy()

    grid = torchvision.utils.make\_grid(Xb[:64], nrow=8, padding=2)

    img = denormalize(grid, mean, std)

    img = np.transpose(img, (1, 2, 0))

    plt.figure(figsize=(8, 8))

    plt.imshow(img)

    plt.axis("off")

    plt.title("Model predictions on test images")

    plt.savefig(fname, dpi=150)

    plt.close()

    print(f"✅ Predictions visualization saved to {fname}")

# --- основная функция ---

def main():

    device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

    print(f"✅ Using device: {device}")

    # --- загрузка CIFAR-10 ---

    transform = transforms.Compose([

        transforms.ToTensor(),

        transforms.Normalize(mean, std)

    ])

    train\_set = torchvision.datasets.CIFAR10(root="./data", train=True, download=True, transform=transform)

    test\_set = torchvision.datasets.CIFAR10(root="./data", train=False, download=True, transform=transform)

    test\_loader = DataLoader(test\_set, batch\_size=batch\_size, shuffle=False, num\_workers=0)

    classes = train\_set.classes

    # --- загрузка кастомной CNN ---

    custom\_model = SimpleCNN(num\_classes=num\_classes).to(device)

    if os.path.exists(custom\_model\_path):

        custom\_model.load\_state\_dict(torch.load(custom\_model\_path, map\_location=device))

        print(f"✅ Custom SimpleCNN loaded from {custom\_model\_path}")

    else:

        print("⚠️ Custom model not found, training skipped.")

    acc\_custom = evaluate(custom\_model, test\_loader, device)

    print(f"📊 Custom SimpleCNN test accuracy: {acc\_custom \* 100:.2f}%")

    # --- загрузка предобученной MobileNetV3 ---

    mobilenet = models.mobilenet\_v3\_small(weights=models.MobileNet\_V3\_Small\_Weights.IMAGENET1K\_V1)

    mobilenet.classifier[3] = nn.Linear(mobilenet.classifier[3].in\_features, num\_classes)

    mobilenet = mobilenet.to(device)

    print("✅ MobileNetV3 initialized with ImageNet weights")

    # --- обучение MobileNetV3 ---

    criterion = nn.CrossEntropyLoss()

    optimizer = optim.Adam(mobilenet.parameters(), lr=lr)

    train\_loader = DataLoader(train\_set, batch\_size=batch\_size, shuffle=True, num\_workers=0)

    train\_losses, test\_losses, test\_accs = [], [], []

    for epoch in range(num\_epochs):

        mobilenet.train()

        running\_loss = 0.0

        for X, y in train\_loader:

            X, y = X.to(device), y.to(device)

            optimizer.zero\_grad()

            out = mobilenet(X)

            loss = criterion(out, y)

            loss.backward()

            optimizer.step()

            running\_loss += loss.item() \* X.size(0)

        train\_loss = running\_loss / len(train\_loader.dataset)

        test\_acc = evaluate(mobilenet, test\_loader, device)

        train\_losses.append(train\_loss)

        test\_accs.append(test\_acc)

        print(f"Epoch {epoch+1}/{num\_epochs} | Loss: {train\_loss:.4f} | Test Acc: {test\_acc\*100:.2f}%")

    # --- сравнение ---

    acc\_mobilenet = evaluate(mobilenet, test\_loader, device)

    print(f"\n📊 Results comparison:")

    print(f"  • Custom SimpleCNN accuracy: {acc\_custom\*100:.2f}%")

    print(f"  • Fine-tuned MobileNetV3 accuracy: {acc\_mobilenet\*100:.2f}%")

    # --- графики ---

    plt.figure(figsize=(6, 4))

    plt.plot(range(1, len(train\_losses) + 1), train\_losses, label="Train loss")

    plt.plot(range(1, len(test\_accs) + 1), test\_accs, label="Test acc")

    plt.xlabel("Epoch")

    plt.ylabel("Value")

    plt.title("MobileNetV3 Training Progress")

    plt.legend()

    plt.tight\_layout()

    plt.savefig("mobilenet\_training.png", dpi=150)

    print("✅ Saved training curve to mobilenet\_training.png")

    plt.close()

    # --- визуализация предсказаний ---

    visualize\_predictions(mobilenet, test\_loader, classes, mean, std, device, fname="mobilenet\_preds.png")

    visualize\_predictions(custom\_model, test\_loader, classes, mean, std, device, fname="customcnn\_preds.png")

    # --- классификация произвольного изображения ---

    def predict\_image(image\_path, model, model\_name):

        model.eval()

        if not os.path.exists(image\_path):

            print(f"⚠️ Image not found: {image\_path}")

            return

        img = Image.open(image\_path).convert("RGB").resize((32, 32))

        t = transforms.Compose([

            transforms.ToTensor(),

            transforms.Normalize(mean, std)

        ])

        x = t(img).unsqueeze(0).to(device)

        with torch.no\_grad():

            logits = model(x)

            probs = torch.softmax(logits, dim=1).cpu().numpy()[0]

        top\_idx = probs.argsort()[::-1][:3]

        print(f"\n🔍 Predictions for {model\_name}:")

        for i in top\_idx:

            print(f"  {classes[i]:10s}: {probs[i]\*100:.2f}%")

    example\_img = r"C:\Users\User\OneDrive\Desktop\IP\_AI\_24\reports\Kurash\lab1\src\image.png"

    predict\_image(example\_img, custom\_model, "Custom SimpleCNN")

    predict\_image(example\_img, mobilenet, "Fine-tuned MobileNetV3")

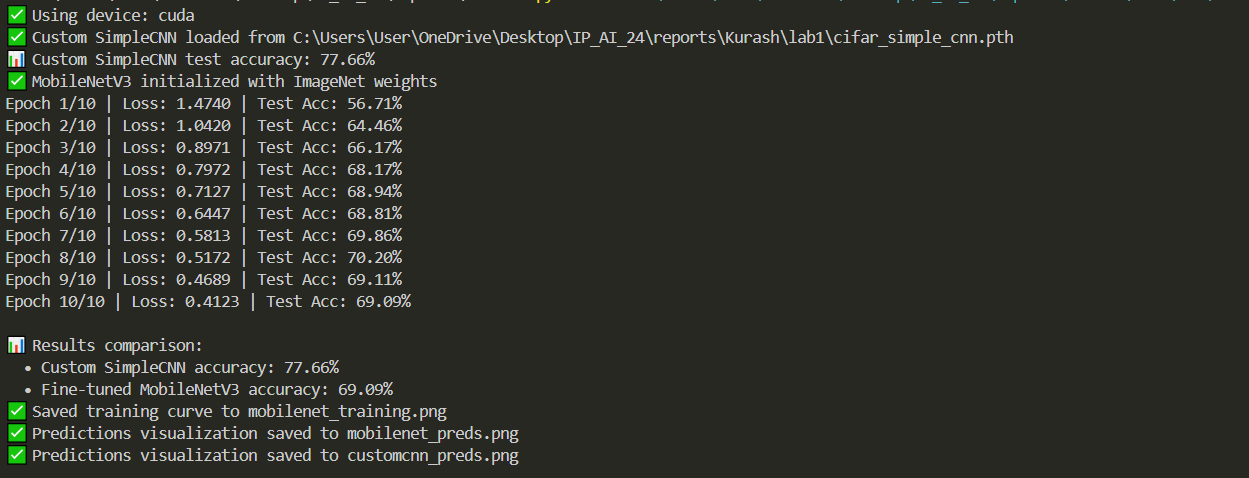
# ✅ безопасный запуск для Windows

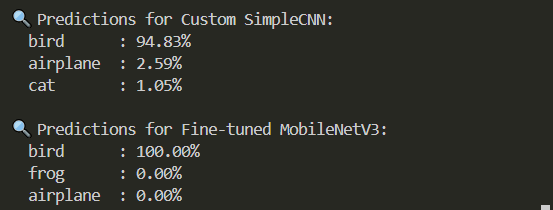
if \_\_name\_\_ == "\_\_main\_\_":

    import multiprocessing

    multiprocessing.freeze\_support()

    main()





**Вывод:** Я изучил построение модели на базе предобученных моделей .