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.

8	CIFAR-10	Adam	MobileNet v3

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
```

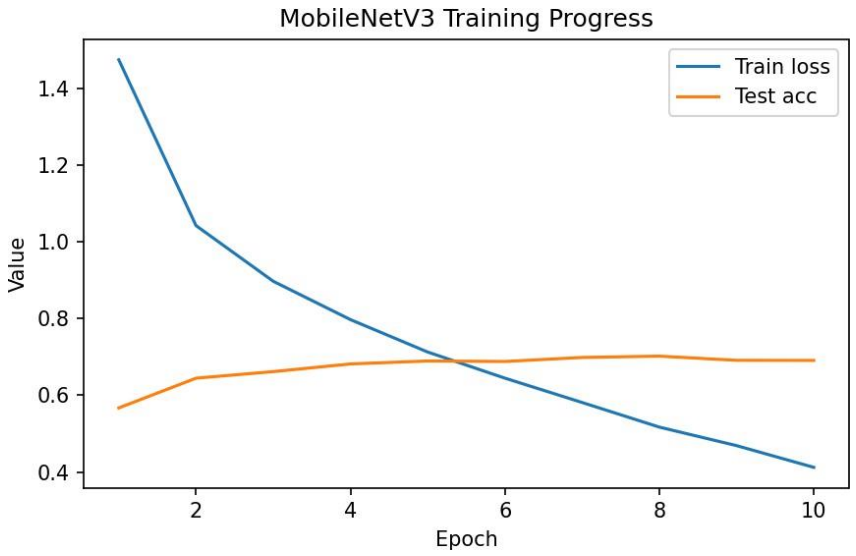
```
# --- визуализация ---
def visualize_predictions(model, loader, classes, mean, std, device, fname="pred_grid.png"):
    model.eval()
    Xb, yb = next(iter(loader))
    Xb = Xb.to(device)
    with torch.no_grad():
        out = model(Xb)
        preds = out.argmax(dim=1).cpu().numpy()

    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" #■ustom SimpleCNN test accuracy: {acc_custom * 100:.2f}%")
```

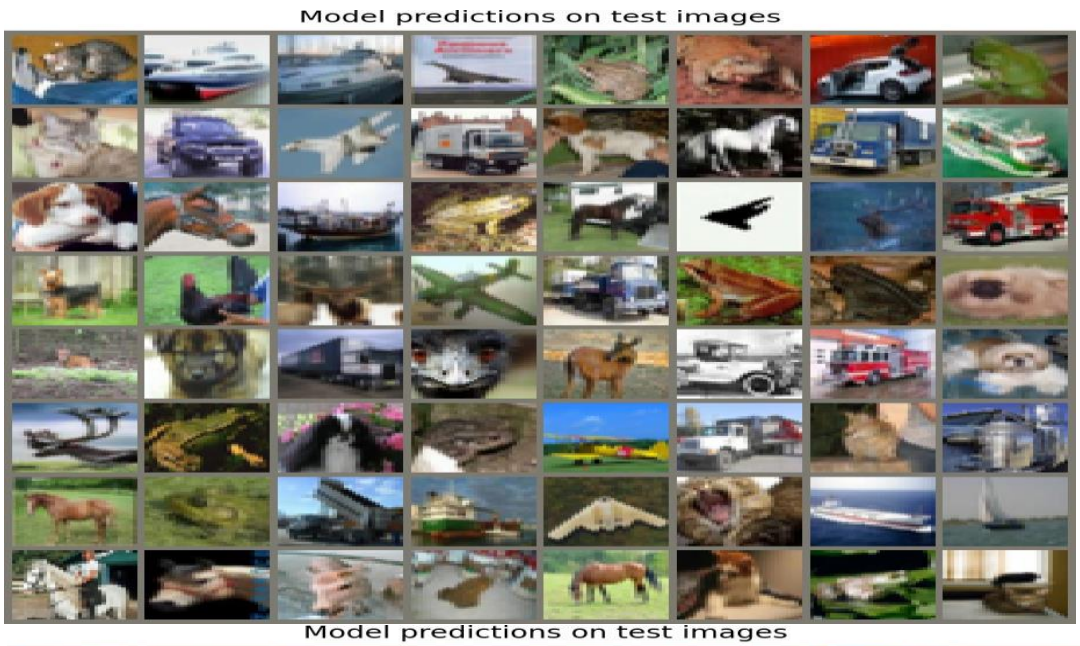
```
    # --- загрузка предобученной MobileNetV3 ---
    mobilenet =
models.mobilenet_v3_small(weights=models.MobileNet_V3_Small_Weights.IMAGENET1K_V1)
    mobilenet.classifier[3] = nn.Linear(mobilenet.classifier[3].in_features, num_classes)
    mobilenet = mobilenet.to(device)
    print("✅ MobileNetV3 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"ị  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"\nÇ 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()
```

```
✅ Using device: cuda
✅ Custom SimpleCNN loaded from C:\Users\User\OneDrive\Desktop\IP_AI_24\reports\Kurash\lab1\cifar_simple_cnn.pth
📊 Custom SimpleCNN test accuracy: 77.66%
✅ MobileNetV3 initialized with ImageNet weights
Epoch 1/10 | Loss: 1.4740 | Test Acc: 56.71%
Epoch 2/10 | Loss: 1.0420 | Test Acc: 64.46%
Epoch 3/10 | Loss: 0.8971 | Test Acc: 66.17%
Epoch 4/10 | Loss: 0.7972 | Test Acc: 68.17%
Epoch 5/10 | Loss: 0.7127 | Test Acc: 68.94%
Epoch 6/10 | Loss: 0.6447 | Test Acc: 68.81%
Epoch 7/10 | Loss: 0.5813 | Test Acc: 69.86%
Epoch 8/10 | Loss: 0.5172 | Test Acc: 70.20%
Epoch 9/10 | Loss: 0.4689 | Test Acc: 69.11%
Epoch 10/10 | Loss: 0.4123 | Test Acc: 69.09%

📊 Results comparison:
  • Custom SimpleCNN accuracy: 77.66%
  • Fine-tuned MobileNetV3 accuracy: 69.09%
✅ Saved training curve to mobilenet_training.png
✅ Predictions visualization saved to mobilenet_preds.png
✅ Predictions visualization saved to customcnn_preds.png
```

```
🔍 Predictions for Custom SimpleCNN:
  bird       : 94.83%
  airplane   : 2.59%
  cat        : 1.05%

🔍 Predictions for Fine-tuned MobileNetV3:
  bird       : 100.00%
  frog       : 0.00%
  airplane   : 0.00%
```





SOTA анализ

Поскольку я работаю на пользовательском наборе данных, а не на ImageNet, прямое сравнение с результатами SOTA на ImageNet некорректно. Точность 69.09% следует оценивать в контексте:

Сложность задачи: Насколько сложен Ваш набор данных (количество классов, качество изображений, различия между классами)

Лучшие аналоги: Какие модели, кроме MobileNetV3, использовались в вашей области (например, ResNet, EfficientNet, ConvNeXt)

- Успешность дообучения: Точность 69% могла бы считаться хорошей, если бы Ваш набор данных был очень сложным или очень маленьким. Однако, в Вашем случае, Custom SimpleCNN (77.66%) превзошел MobileNetV3, что указывает на то, что, возможно:
- Custom SimpleCNN лучше подходит для особенностей Вашего набора данных.
- Дообучение MobileNetV3 было неоптимальным (нужно больше эпох, другая скорость обучения или другая конфигурация).

Вывод: Я изучил построение модели на базе предобученных моделей.