Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Отчет по лабораторной работе 1**

Специальность ИИ-23

Выполнила:

Студентка 4 курса

Группы ИИ-23

Маслакова К.Ю.

Проверил:

Андреенко К.В

Преподаватель –стажер

«\_\_»\_\_\_\_\_\_\_\_2025г.

Брест 2025

**Вариант 4**

Выборка: CIFAR-100

Размер исходного изображения: 32X32

Оптимизатор: SGD

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.

**Код программы:**

# train\_cifar100\_simplecnn.py

import os

import time

import copy

import argparse

from tqdm import tqdm

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

import torchvision

from torchvision import transforms, datasets

from PIL import Image

import numpy as np

import matplotlib.pyplot as plt

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

BATCH\_SIZE = 128

LR = 0.01

MOMENTUM = 0.9

WEIGHT\_DECAY = 5e-4

EPOCHS = 5

NUM\_WORKERS = 4

CHECKPOINT\_DIR = "checkpoints"

os.makedirs(CHECKPOINT\_DIR, exist\_ok=True)

class SimpleCIFAR100CNN(nn.Module):

def \_\_init\_\_(self, num\_classes=100):

super().\_\_init\_\_()

self.features = nn.Sequential(

nn.Conv2d(3, 64, kernel\_size=3, padding=1), # 32x32

nn.BatchNorm2d(64),

nn.ReLU(inplace=True),

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

nn.BatchNorm2d(64),

nn.ReLU(inplace=True),

nn.MaxPool2d(2), # 16x16

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

nn.BatchNorm2d(128),

nn.ReLU(inplace=True),

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

nn.BatchNorm2d(128),

nn.ReLU(inplace=True),

nn.MaxPool2d(2), # 8x8

nn.Conv2d(128, 256, kernel\_size=3, padding=1),

nn.BatchNorm2d(256),

nn.ReLU(inplace=True),

nn.MaxPool2d(2), # 4x4

)

self.classifier = nn.Sequential(

nn.Flatten(),

nn.Linear(256 \* 4 \* 4, 512),

nn.ReLU(inplace=True),

nn.Dropout(0.5),

nn.Linear(512, num\_classes)

)

def forward(self, x):

x = self.features(x)

x = self.classifier(x)

return x

train\_transform = transforms.Compose([

transforms.RandomCrop(32, padding=4),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565, 0.2761]) # CIFAR-100 stats

])

test\_transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565, 0.2761])

])

train\_dataset = datasets.CIFAR100(root="./data", train=True, download=True, transform=train\_transform)

test\_dataset = datasets.CIFAR100(root="./data", train=False, download=True, transform=test\_transform)

train\_loader = DataLoader(train\_dataset, batch\_size=BATCH\_SIZE, shuffle=True, num\_workers=NUM\_WORKERS)

test\_loader = DataLoader(test\_dataset, batch\_size=BATCH\_SIZE, shuffle=False, num\_workers=NUM\_WORKERS)

classes = train\_dataset.classes

def train\_one\_epoch(model, loader, criterion, optimizer, device):

model.train()

running\_loss = 0.0

correct = 0

total = 0

loop = tqdm(loader, desc="Train", leave=False)

for inputs, targets in loop:

inputs, targets = inputs.to(device), targets.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, targets)

loss.backward()

optimizer.step()

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

\_, preds = outputs.max(1)

correct += preds.eq(targets).sum().item()

total += targets.size(0)

loop.set\_postfix(loss=running\_loss/total, acc=100.\*correct/total)

epoch\_loss = running\_loss / total

epoch\_acc = 100. \* correct / total

return epoch\_loss, epoch\_acc

def evaluate(model, loader, criterion, device):

model.eval()

running\_loss = 0.0

correct = 0

total = 0

with torch.no\_grad():

loop = tqdm(loader, desc="Eval ", leave=False)

for inputs, targets in loop:

inputs, targets = inputs.to(device), targets.to(device)

outputs = model(inputs)

loss = criterion(outputs, targets)

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

\_, preds = outputs.max(1)

correct += preds.eq(targets).sum().item()

total += targets.size(0)

loop.set\_postfix(loss=running\_loss/total, acc=100.\*correct/total)

epoch\_loss = running\_loss / total

epoch\_acc = 100. \* correct / total

return epoch\_loss, epoch\_acc

def main():

model = SimpleCIFAR100CNN(num\_classes=100).to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=LR, momentum=MOMENTUM, weight\_decay=WEIGHT\_DECAY)

scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=20, gamma=0.1)

best\_acc = 0.0

history = {"train\_loss":[], "train\_acc":[], "val\_loss":[], "val\_acc":[]}

for epoch in range(1, EPOCHS+1):

print(f"Epoch {epoch}/{EPOCHS}")

t0 = time.time()

train\_loss, train\_acc = train\_one\_epoch(model, train\_loader, criterion, optimizer, device)

val\_loss, val\_acc = evaluate(model, test\_loader, criterion, device)

scheduler.step()

history["train\_loss"].append(train\_loss)

history["train\_acc"].append(train\_acc)

history["val\_loss"].append(val\_loss)

history["val\_acc"].append(val\_acc)

elapsed = time.time() - t0

print(f" train loss: {train\_loss:.4f}, train acc: {train\_acc:.2f}%")

print(f" val loss: {val\_loss:.4f}, val acc: {val\_acc:.2f}% (epoch time {elapsed:.1f}s)")

if val\_acc > best\_acc:

best\_acc = val\_acc

torch.save({

"model\_state": model.state\_dict(),

"optimizer\_state": optimizer.state\_dict(),

"epoch": epoch,

"best\_acc": best\_acc,

}, os.path.join(CHECKPOINT\_DIR, "best\_checkpoint.pth"))

print(f" Saved best model (acc={best\_acc:.2f}%)")

epochs = list(range(1, EPOCHS+1))

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)

plt.plot(epochs, history["train\_loss"], label="train loss")

plt.plot(epochs, history["val\_loss"], label="val loss")

plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend(); plt.title("Loss")

plt.subplot(1,2,2)

plt.plot(epochs, history["train\_acc"], label="train acc")

plt.plot(epochs, history["val\_acc"], label="val acc")

plt.xlabel("Epoch"); plt.ylabel("Accuracy (%)"); plt.legend(); plt.title("Accuracy")

plt.tight\_layout()

plt.savefig("training\_plots.png", dpi=150)

print("Графики сохранены в training\_plots.png")

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

main()

def predict\_image(image\_path, checkpoint\_path=os.path.join(CHECKPOINT\_DIR, "best\_checkpoint.pth"), topk=5):

from torchvision.transforms.functional import to\_pil\_image

net = SimpleCIFAR100CNN(num\_classes=100).to(device)

ck = torch.load(checkpoint\_path, map\_location=device)

net.load\_state\_dict(ck["model\_state"])

net.eval()

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

prep = transforms.Compose([

transforms.Resize((32,32)),

transforms.ToTensor(),

transforms.Normalize(mean=[0.5071, 0.4867, 0.4408], std=[0.2675, 0.2565, 0.2761])

])

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

with torch.no\_grad():

out = net(x)

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

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

for i in top\_idx:

print(f"{classes[i]:30s} prob={probs[i]:.4f}")

**Результат работы программы:**

Epoch 1/5 train loss: 3.9294, train acc: 9.20%

val loss: 3.3993, val acc: 17.35% (epoch time 5403.7s)

Saved best model (acc=17.35%)

Epoch 2/5

train loss: 3.3692, train acc: 17.61%

val loss: 2.9787, val acc: 25.08% (epoch time 2931.9s)

Saved best model (acc=25.08%)

Epoch 3/5

train loss: 3.0409, train acc: 23.44%

val loss: 2.7062, val acc: 30.72% (epoch time 2494.2s)

Saved best model (acc=30.72%)

Epoch 4/5

train loss: 2.7912, train acc: 28.28%

val loss: 2.5568, val acc: 33.59% (epoch time 2552.5s)

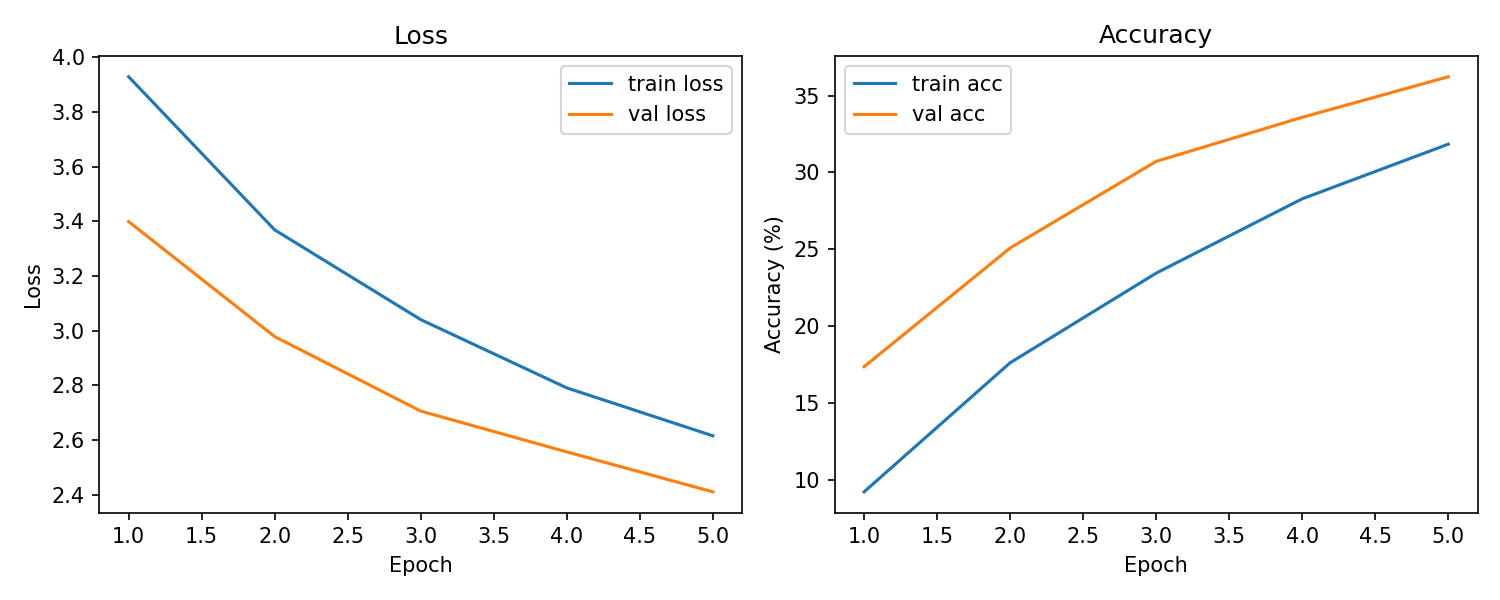
Saved best model (acc=33.59%)

Epoch 5/5

train loss: 2.6157, train acc: 31.84%

val loss: 2.4108, val acc: 36.23% (epoch time 2461.1s)

Saved best model (acc=36.23%)

Графики:

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