Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

# Лабораторная работа №1

По дисциплине: «Обработка изображений в интеллектуальных системах»

Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-23 Шурина А.А.

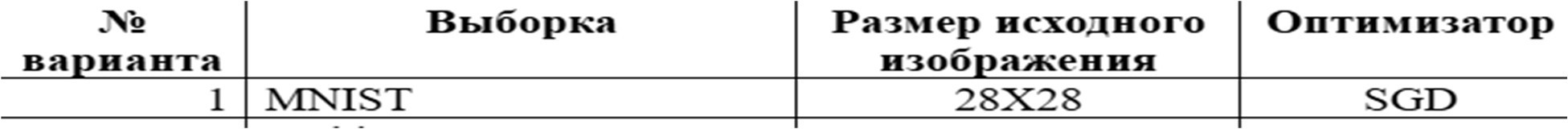
Проверила:

Андренко К.В.

# Брест 2025

Цель работы: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

Вариант 13.


Код программы:

import argparse  
import os  
import time  
from pathlib import Path  
import matplotlib.pyplot as plt  
import numpy as np  
import torch  
import torch.nn as nn  
import torch.nn.functional as F  
import torch.optim as optim  
import torchvision  
from PIL import Image  
from torch.utils.data import DataLoader  
from torchvision import transforms  
from tqdm import tqdm  
CLASS\_NAMES = ('airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck')  
base\_dir = Path(r"C:\\Users\\User\\Desktop\\Studing-7sem\\OIIS\\lab\_1")  
models\_dir = base\_dir / 'models'  
results\_dir = base\_dir / 'results'  
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.ReLU(inplace=True),  
 nn.MaxPool2d(2),  
 nn.Conv2d(32, 64, kernel\_size=3, padding=1),  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(2),  
 nn.Conv2d(64, 128, kernel\_size=3, padding=1),  
 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  
def get\_dataloaders(batch\_size=128, augment=True, num\_workers=4):  
 mean = (0.4914, 0.4822, 0.4465)  
 std = (0.2470, 0.2435, 0.2616)  
 train\_transforms = [transforms.ToTensor(), transforms.Normalize(mean, std)]  
 if augment:  
 train\_transforms = [  
 transforms.RandomHorizontalFlip(),  
 transforms.RandomCrop(32, padding=4),  
 ] + train\_transforms  
 transform\_train = transforms.Compose(train\_transforms)  
 transform\_test = transforms.Compose([transforms.ToTensor(), transforms.Normalize(mean, std)])  
 train\_set = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform\_train)  
 test\_set = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform\_test)  
 train\_loader = DataLoader(train\_set, batch\_size=batch\_size, shuffle=True, num\_workers=num\_workers)  
 test\_loader = DataLoader(test\_set, batch\_size=batch\_size, shuffle=False, num\_workers=num\_workers)  
 return train\_loader, test\_loader  
def train\_epoch(model, device, loader, criterion, optimizer):  
 model.train()  
 running\_loss = 0.0  
 correct = 0  
 total = 0  
 for inputs, targets in tqdm(loader, desc='train', leave=False):  
 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)  
 \_, predicted = outputs.max(1)  
 total += targets.size(0)  
 correct += predicted.eq(targets).sum().item()  
 epoch\_loss = running\_loss / total  
 epoch\_acc = 100.0 \* correct / total  
 return epoch\_loss, epoch\_acc  
def eval\_epoch(model, device, loader, criterion):  
 model.eval()  
 running\_loss = 0.0  
 correct = 0  
 total = 0  
 with torch.no\_grad():  
 for inputs, targets in tqdm(loader, desc='eval', leave=False):  
 inputs, targets = inputs.to(device), targets.to(device)  
 outputs = model(inputs)  
 loss = criterion(outputs, targets)  
 running\_loss += loss.item() \* inputs.size(0)  
 \_, predicted = outputs.max(1)  
 total += targets.size(0)  
 correct += predicted.eq(targets).sum().item()  
  
 epoch\_loss = running\_loss / total  
 epoch\_acc = 100.0 \* correct / total  
 return epoch\_loss, epoch\_acc  
def plot\_metrics(history):  
 results\_dir.mkdir(parents=True, exist\_ok=True)  
 epochs = len(history['train\_loss'])  
 plt.figure(figsize=(10,4))  
 plt.subplot(1,2,1)  
 plt.plot(range(1, epochs+1), history['train\_loss'], label='train\_loss')  
 plt.plot(range(1, epochs+1), history['test\_loss'], label='test\_loss')  
 plt.xlabel('Epoch')  
 plt.ylabel('Loss')  
 plt.legend()  
  
 plt.subplot(1,2,2)  
 plt.plot(range(1, epochs+1), history['train\_acc'], label='train\_acc')  
 plt.plot(range(1, epochs+1), history['test\_acc'], label='test\_acc')  
 plt.xlabel('Epoch')  
 plt.ylabel('Accuracy (%)')  
 plt.legend()  
 plt.tight\_layout()  
 out\_path = results\_dir / 'training\_metrics.png'  
 plt.savefig(out\_path)  
 print('Saved plot to', out\_path)  
 plt.close()  
def predict\_image(model, device, img\_path):  
 model.eval()  
 mean = (0.4914, 0.4822, 0.4465)  
 std = (0.2470, 0.2435, 0.2616)  
 transform = transforms.Compose([  
 transforms.Resize((32,32)),  
 transforms.ToTensor(),  
 transforms.Normalize(mean, std)  
 ])  
 img = Image.open(img\_path).convert('RGB')  
 inp = transform(img).unsqueeze(0).to(device)  
 with torch.no\_grad():  
 out = model(inp)  
 probs = F.softmax(out, dim=1).cpu().numpy()[0]  
 pred = int(out.argmax(1).cpu().numpy()[0])  
 plt.imshow(np.array(img))  
 plt.title(f'Pred: {CLASS\_NAMES[pred]} ({probs[pred]\*100:.1f}%)')  
 plt.axis('off')  
 plt.show()  
 return pred, probs

Результат работы программы:

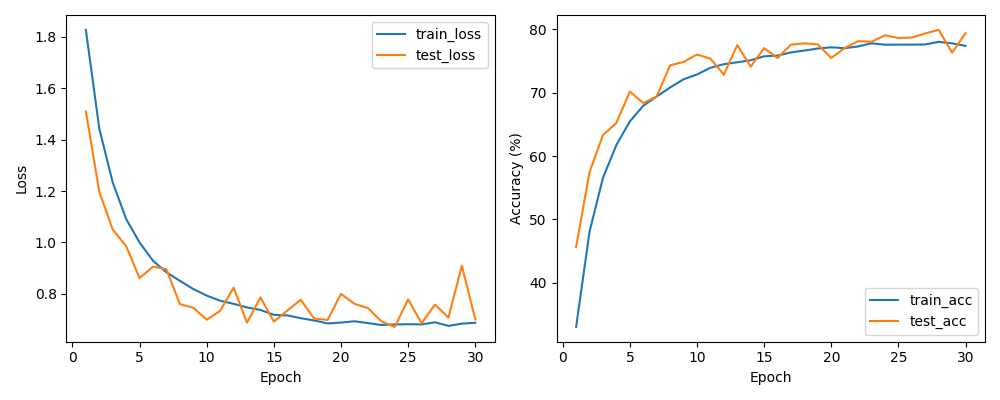
Epoch 25/30 Train loss 0.6814 acc 77.60% | Test loss 0.7783 acc 78.64% (62.5s)

Epoch 26/30 Train loss 0.6807 acc 77.59% | Test loss 0.6852 acc 78.72% (53.6s)

Epoch 27/30 Train loss 0.6889 acc 77.62% | Test loss 0.7577 acc 79.37% (51.7s)

Epoch 28/30 Train loss 0.6749 acc 78.04% | Test loss 0.7070 acc 79.93% (53.5s)

Epoch 29/30 Train loss 0.6838 acc 77.80% | Test loss 0.9092 acc 76.33% (55.0s)





Полученные результаты сравнивались с современными моделями, такими как ResNet и DenseNet, которые достигают точности свыше 95% на той же выборке. Однако простая сеть имеет значительно меньшую глубину и сложность, что делает её удобной для изучения базовых принципов работы сверточных нейронных сетей.

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