Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Обработка изображений в ИС»

###### Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнил:

Студент 4 курса

Группы ИИ-21

Пучинский А.А.

Проверил:

Крощенко А.А.

Брест 2024

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

Код программы:

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import numpy as np

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

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

transform = 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)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64, shuffle=True)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=64, shuffle=False)

class SimpleCNN(nn.Module):

def \_\_init\_\_(self):

super(SimpleCNN, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(3, 32, 3, padding=1)

self.conv2 = nn.Conv2d(32, 64, 3, padding=1)

self.pool = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(64 \* 8 \* 8, 512)

self.fc2 = nn.Linear(512, 10)

self.relu = nn.ReLU()

def forward(self, x):

x = self.pool(self.relu(self.conv1(x)))

x = self.pool(self.relu(self.conv2(x)))

x = x.view(-1, 64 \* 8 \* 8)

x = self.relu(self.fc1(x))

x = self.fc2(x)

return x

net = SimpleCNN().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

def train\_model(epochs=10):

train\_losses = []

for epoch in range(epochs):

running\_loss = 0.0

for i, data in enumerate(trainloader, 0):

inputs, labels = data

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

optimizer.zero\_grad()

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

train\_losses.append(running\_loss / len(trainloader))

print(f'Epoch {epoch+1}, Loss: {running\_loss / len(trainloader)}')

return train\_losses

epochs = 10

train\_losses = train\_model(epochs)

plt.plot(range(epochs), train\_losses, label='Training Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Training Loss Curve')

plt.legend()

plt.show()

def evaluate\_model():

correct = 0

total = 0

with torch.no\_grad():

for data in testloader:

images, labels = data

images, labels = images.to(device), labels.to(device) # Перенос данных на устройство

outputs = net(images)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

correct += (predicted == labels).sum().item()

accuracy = 100 \* correct / total

print(f'Accuracy on test set: {accuracy:.2f}%')

evaluate\_model()

def imshow(img):

img = img / 2 + 0.5

npimg = img.numpy()

plt.imshow(np.transpose(npimg, (1, 2, 0)))

plt.show()

dataiter = iter(testloader)

images, labels =next(dataiter)

imshow(torchvision.utils.make\_grid(images))

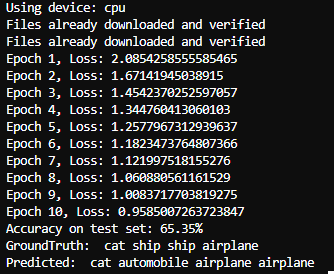
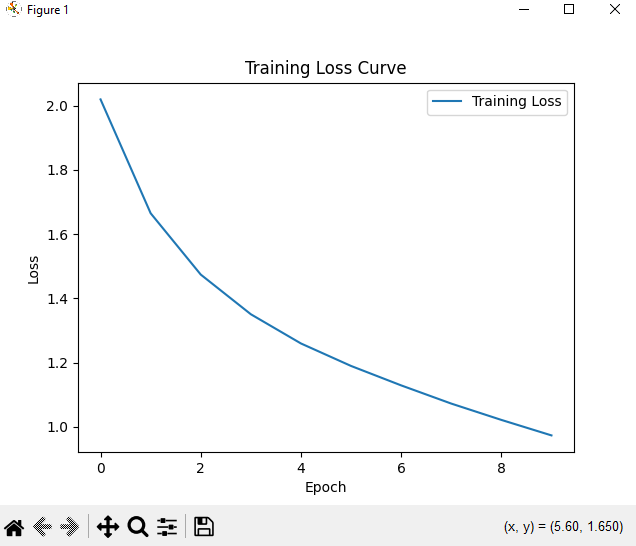
print('GroundTruth: ', ' '.join(f'{trainset.classes[labels[j]]}' for j in range(4)))

images = images.to(device)

outputs = net(images)

\_, predicted = torch.max(outputs, 1)

print('Predicted: ', ' '.join(f'{trainset.classes[predicted[j]]}' for j in range(4)))



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