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

Литвинюк Т. В.

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

import seaborn as sns

from sklearn.metrics import confusion\_matrix

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

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=128, shuffle=True, num\_workers=4)

testset = torchvision.datasets.CIFAR10(

root='./data', train=False, download=True, transform=transform)

testloader = torch.utils.data.DataLoader(

testset, batch\_size=128, shuffle=False, num\_workers=4)

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.Adadelta(net.parameters())

# Обучение модели

num\_epochs = 10

train\_loss\_list = []

for epoch in range(num\_epochs):

running\_loss = 0.0

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

inputs, labels = data

inputs, labels = inputs.to(device), labels.to(device) # <-- Важный шаг

optimizer.zero\_grad()

outputs = net(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

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

print(f"Epoch {epoch+1}, Loss: {running\_loss / len(trainloader):.3f}")

print('Training Finished')

correct = 0

total = 0

all\_predictions = []

all\_labels = []

num\_classes = 10

with torch.no\_grad():

for data in testloader:

images, labels = data

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

outputs = net(images)

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

total += labels.size(0)

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

all\_predictions.extend(predicted.cpu().numpy()) # Сохранение предсказаний

all\_labels.extend(labels.cpu().numpy())

print(f'Accuracy of the network on the 10000 test images: {100 \* correct / total:.2f}%')

cm = confusion\_matrix(all\_labels, all\_predictions)

cm\_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(20, 18))

sns.heatmap(cm\_normalized, annot=False, fmt='.2f', cmap='Blues', cbar=True)

plt.xlabel('Predicted', fontsize=14)

plt.ylabel('True', fontsize=14)

plt.title('Confusion Matrix (Normalized)', fontsize=16)

plt.xticks(np.arange(num\_classes) + 0.5, labels=np.arange(num\_classes), rotation=90, fontsize=10)

plt.yticks(np.arange(num\_classes) + 0.5, labels=np.arange(num\_classes), rotation=0, fontsize=10)

plt.tight\_layout()

plt.show()

plt.plot(train\_loss\_list, label='Training Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

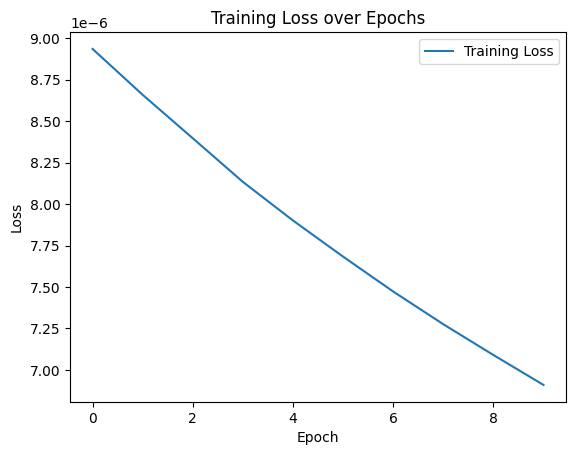
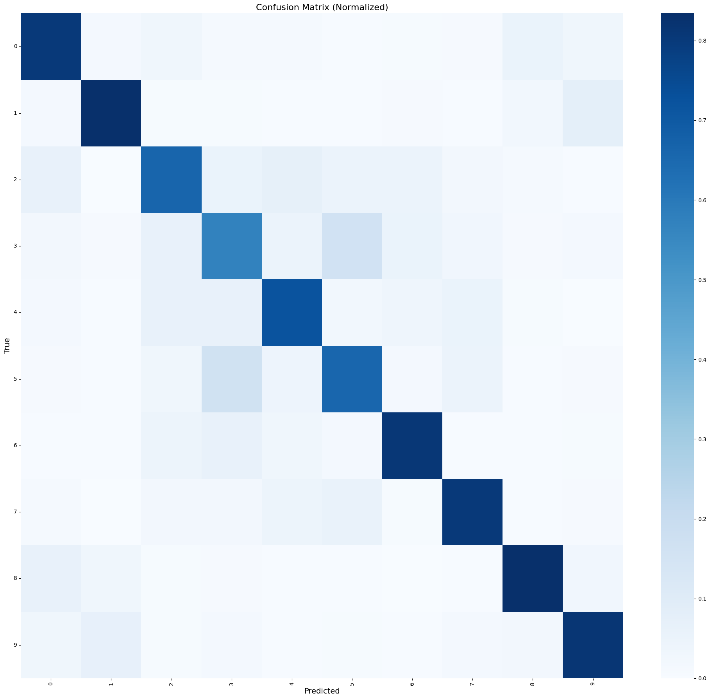
plt.title('Training Loss over Epochs')

plt.legend()

plt.show()

dataiter = iter(testloader)

images, labels = next(dataiter)





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