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»

Выполнила:

Студентка 4 курса

Группы ИИ-21

Соболева П.С.

Проверил:

Крощенко А.А.

Брест 2024

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

Ход работы:

Вариант 14

№ варианта	Выборка	Размер исходного изображения	Оптимизатор
14	CIFAR-100	32X32	Adadelta

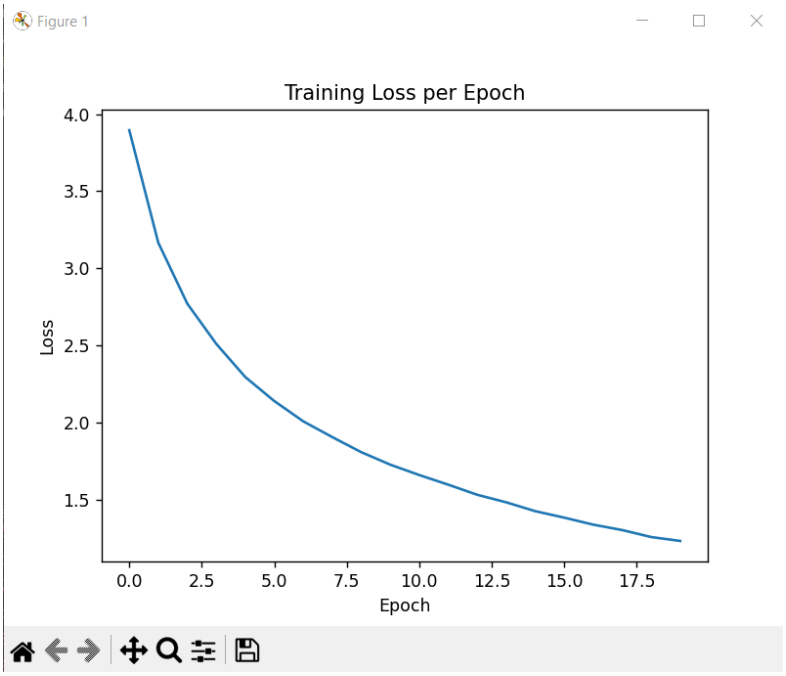
Код:

```
import torch
import torchvision
import torch.nn as nn
from tqdm import tqdm
import matplotlib.pyplot as plt

batch_size_train = 256
batch_size_test = 1000

# Аугментация данных для обучения
transform_train = torchvision.transforms.Compose([
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.1),  #
Уменьшена степень искажения
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761))
# Проверка нормализации
])

# Нормализация данных для теста
transform_test = torchvision.transforms.Compose([
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize((0.5071, 0.4867, 0.4408), (0.2675, 0.2565, 0.2761))
# Проверка нормализации
])

# Загрузка тренировочных данных
train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR100(root='C:\\Users\\user\\PycharmProjects\\ОИвИС\\data',
train=True, download=False,
                                  transform=transform_train), batch_size=batch_size_train,
shuffle=True
)

# Загрузка тестовых данных
test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.CIFAR100(root='C:\\Users\\user\\PycharmProjects\\ОИвИС\\data',
train=False, download=False,
                                  transform=transform_test), batch_size=batch_size_test,
shuffle=False
)

# Определение модели CNN
class Net(nn.Module):
```

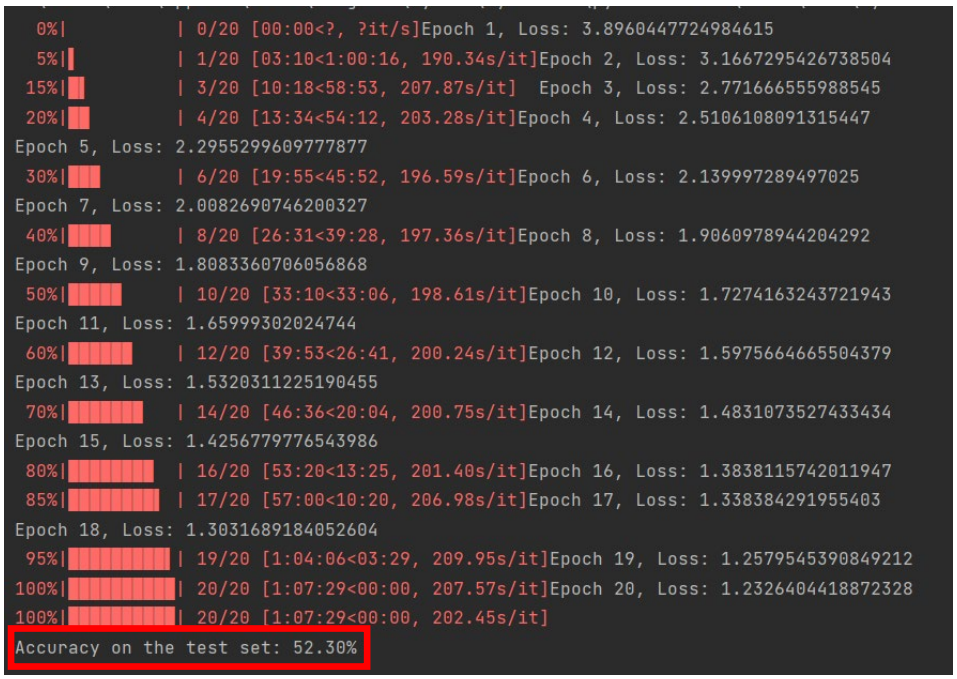
```
    def __init__(self):
        super().__init__()
        self.conv_block1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.3)
        )
        self.conv_block2 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1,
padding=1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.3)
        )
        self.conv_block3 = nn.Sequential(
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, stride=1,
padding=1),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2),
            nn.MaxPool2d(2, 2),
            nn.Dropout(0.4)
        )
        self.flc_block = nn.Sequential(
            nn.Flatten(),
            nn.Linear(256 * 4 * 4, 512),
            nn.LeakyReLU(0.2),
            nn.Linear(512, 100)
        )

    def forward(self, x):
        x = self.conv_block1(x)
        x = self.conv_block2(x)
        x = self.conv_block3(x)
        x = self.flc_block(x)
        return x

# Функция обучения модели
def train(device, model, train_loader, learning_rate=1.0, epochs=20):
    loss_fn = nn.CrossEntropyLoss().to(device)
    optimizer = torch.optim.Adadelta(model.parameters(), lr=learning_rate,
weight_decay=1e-4)
    history = []

    for epoch in tqdm(range(epochs)):
        epoch_loss = 0.0
        model.train()

        for x, y in train_loader:
            optimizer.zero_grad()
            x, y = x.to(device), y.to(device)
            pred = model(x)

            loss = loss_fn(pred, y)
            epoch_loss += loss.item()
            loss.backward()
            optimizer.step()

        average_loss = epoch_loss / len(train_loader)
        history.append(average_loss)
```

```
        print(f'Epoch {epoch + 1}, Loss: {average_loss}')

    plt.plot(range(0, epochs), history)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Loss per Epoch')
    plt.show()

# Функция тестирования модели
def test(model, device, test_loader):
    model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = correct / total
    print(f"Accuracy on the test set: {accuracy:.2%}")

# Устройство для вычислений
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Net().to(device)

# Обучение и тестирование модели
train(device, model, train_loader, learning_rate=1.0, epochs=20)
test(model, device, test_loader)
```

График изменения ошибки:



Среднее значение потерь для каждой эпохи и точность на тестовой выборке:



Увеличение количества эпох, увеличение количества сверточных слоев может помочь улучшить точность на тестовой выборке, но также увеличит время обучения.

Вывод: в ходе выполнения лабораторной работы научилась конструировать нейросетевые классификаторы и выполнять их обучение на одной из известных выборок компьютерного зрения.