

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	Adadelata

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), #
    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(
padding=1),
        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(
padding=1),
        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()

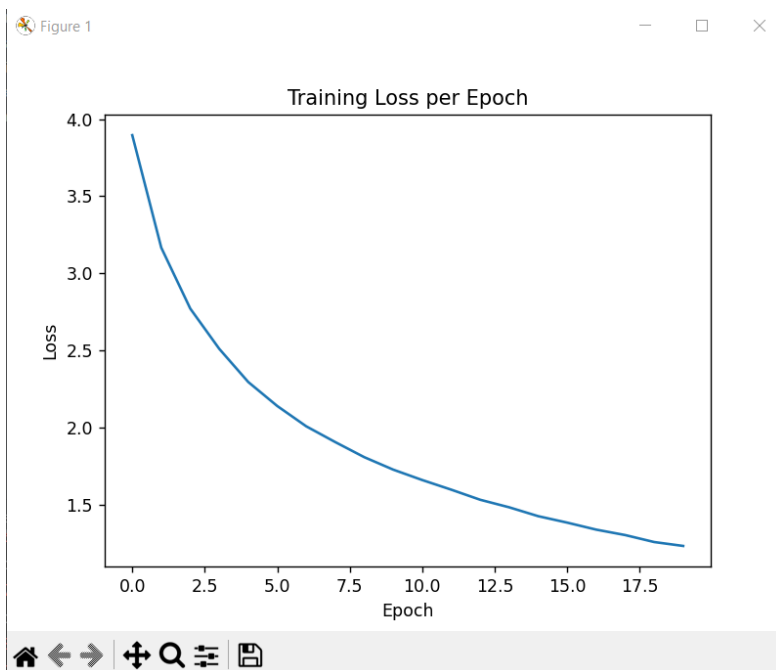
    accuracy = correct / total
    print(f"Accuracy on the test set: {accuracy:.2%}")

# 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:

```
0%|          | 0/20 [00:00<?, ?it/s]Epoch 1, Loss: 3.8960447724984615
5%|          | 1/20 [03:10<1:00:16, 190.34s/it]Epoch 2, Loss: 3.1667295426738504
15%|         | 3/20 [10:18<58:53, 207.87s/it] Epoch 3, Loss: 2.771666555988545
20%|         | 4/20 [13:34<54:12, 203.28s/it]Epoch 4, Loss: 2.5106108091315447
Epoch 5, Loss: 2.2955299609777877
30%|        | 6/20 [19:55<45:52, 196.59s/it]Epoch 6, Loss: 2.139997289497025
Epoch 7, Loss: 2.0082690746200327
40%|        | 8/20 [26:31<39:28, 197.36s/it]Epoch 8, Loss: 1.9060978944204292
Epoch 9, Loss: 1.8083360706056868
50%|        | 10/20 [33:10<33:06, 198.61s/it]Epoch 10, Loss: 1.7274163243721943
Epoch 11, Loss: 1.65999302024744
60%|        | 12/20 [39:53<26:41, 200.24s/it]Epoch 12, Loss: 1.5975664665504379
Epoch 13, Loss: 1.5320311225190455
70%|        | 14/20 [46:36<20:04, 200.75s/it]Epoch 14, Loss: 1.4831073527433434
Epoch 15, Loss: 1.4256779776543986
80%|        | 16/20 [53:20<13:25, 201.40s/it]Epoch 16, Loss: 1.3838115742011947
85%|        | 17/20 [57:00<10:20, 206.98s/it]Epoch 17, Loss: 1.338384291955403
Epoch 18, Loss: 1.3031689184052604
95%|        | 19/20 [1:04:06<03:29, 209.95s/it]Epoch 19, Loss: 1.2579545390849212
100%|       | 20/20 [1:07:29<00:00, 207.57s/it]Epoch 20, Loss: 1.2326404418872328
100%|       | 20/20 [1:07:29<00:00, 202.45s/it]
Accuracy on the test set: 52.30%
```

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.