Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра интеллектуально-информационных технологий

Обработка изображений в ИС
Лабораторная работа №1
Обучение классификаторов средствами библиотеки PyTorch

Выполнил:
студент 4 курса
группы ИИ-24
Штыхно Д. В.
Проверила:
Андренко К. В.

Брест-2025

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

Общее задание:

1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);
2. Ознакомьтесь с state-of-the-art результатами для предлагаемых выборок (из материалов в сети Интернет). Сделать выводы о результатах обучения СНС из п. 1;
3. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);
4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

№ варианта	Выборка	Размер исходного изображения	Оптимизатор
20	STL-10 (размеченная часть)	96X96	RMSprop

Код программы(вариант 20):

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4469, 0.4393, 0.4066), (0.2240,
0.2210, 0.2239))
])

train_dataset = datasets.STL10(root='./data', split='train',
download=True, transform=transform)
test_dataset = datasets.STL10(root='./data', split='test',
download=True, transform=transform)
```

```
train_loader = torch.utils.data.DataLoader(train_dataset,
batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset,
batch_size=1000, shuffle=False)


class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Входные данные теперь 3 канала (RGB)
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)   # 96 -> 48

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3,
padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)   # 48 -> 24

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3,
padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(2)   # 24 -> 12

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 12 * 12, 256)
        self.relu4 = nn.ReLU()
        self.fc2 = nn.Linear(256, 10)  # 10 классов в STL-10

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = self.flatten(x)
        x = self.relu4(self.fc1(x))
        x = self.fc2(x)
        return x


device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=0.001,
alpha=0.9)


num_epochs = 10
```

```
train_losses = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    epoch_loss = running_loss / len(train_loader)
    train_losses.append(epoch_loss)
    print(f'Epoch {epoch+1}/{num_epochs}, Loss:
{epoch_loss:.4f}')


plt.plot(train_losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss over Epochs')
plt.legend()
plt.show()


model.eval()
correct = 0
total = 0
with torch.no_grad():
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
        total += target.size(0)

accuracy = 100. * correct / total
print(f'Test Accuracy: {accuracy:.2f}%')


data_iter = iter(test_loader)
images, labels = next(data_iter)
```

```
img = images[0]
true_label = labels[0]

model.eval()
with torch.no_grad():
    output = model(img.unsqueeze(0).to(device))
    pred_label = output.argmax().item()


img_show = img.permute(1, 2, 0).cpu().numpy()
img_show = (img_show * np.array((0.2240, 0.2210, 0.2239))) +
np.array((0.4469, 0.4393, 0.4066))
img_show = np.clip(img_show, 0, 1)

plt.imshow(img_show)
plt.title(f'Predicted: {pred_label}, True:
{true_label.item()}')
plt.axis('off')
plt.show()
```

Результат работы программы:

100%|[redacted]| 2.64G/2.64G [06:35<00:00, 6.67MB/s]

Epoch 1/10, Loss: 2.0040

Epoch 2/10, Loss: 1.4648

Epoch 3/10, Loss: 1.1583

Epoch 4/10, Loss: 0.8576

Epoch 5/10, Loss: 0.6013

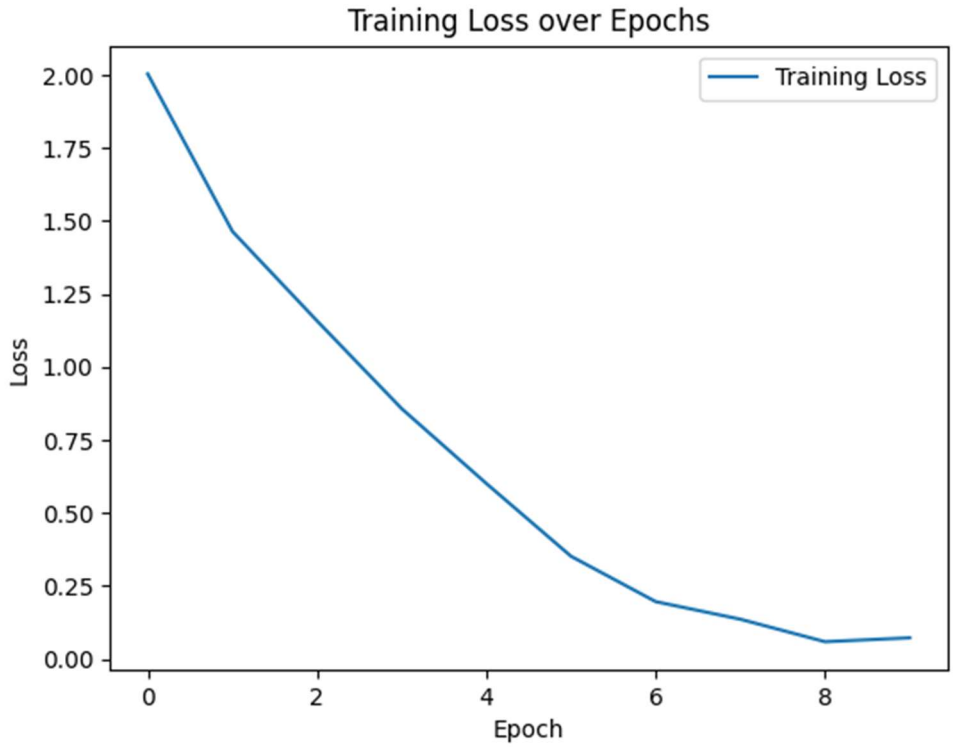
Epoch 6/10, Loss: 0.3522

Epoch 7/10, Loss: 0.1968

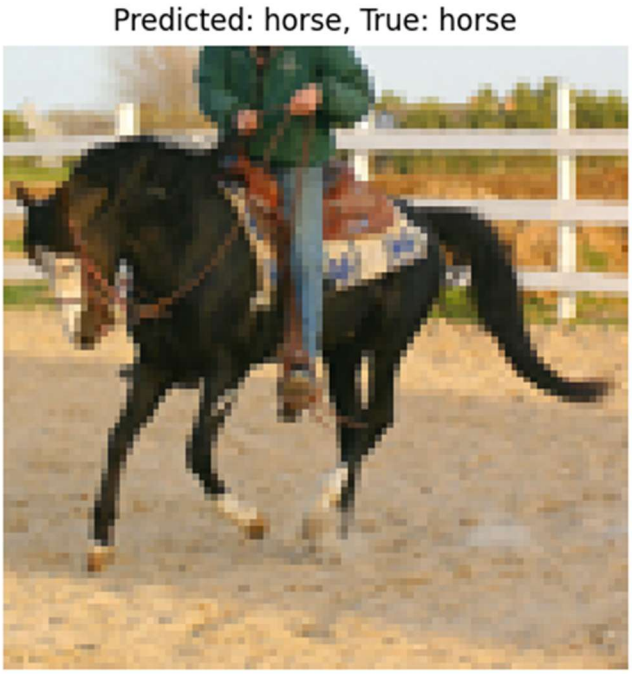
Epoch 8/10, Loss: 0.1366

Epoch 9/10, Loss: 0.0596

Epoch 10/10, Loss: 0.0729



Test Accuracy: 59.09%



Согласно современным источникам (Papers with Code – STL-10, 2024), state-of-the-art результаты на наборе данных **STL-10** достигают **98–99% accuracy** при использовании глубоких архитектур (ResNet, Vision Transformer, ConvNeXt) и предобучения на больших выборках (например, ImageNet). Без предобучения, но с аугментациями и регуляризацией, модели показывают **80–90% точности**, тогда как простые CNN, обученные с нуля, обычно достигают **50–65%**. Разработанная в лабораторной работе простая сверточная нейронная сеть, обученная на размеченной части STL-10 без аугментаций, показала **59.09% точности**, что соответствует ожидаемому диапазону для базовых архитектур. Это подтверждает, что модель успешно обучается, однако для приближения к state-of-the-art результатам необходимо использовать более глубокие сети, предобучение и современные методы регуляризации.

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