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-23
Вышинский А. С.
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.

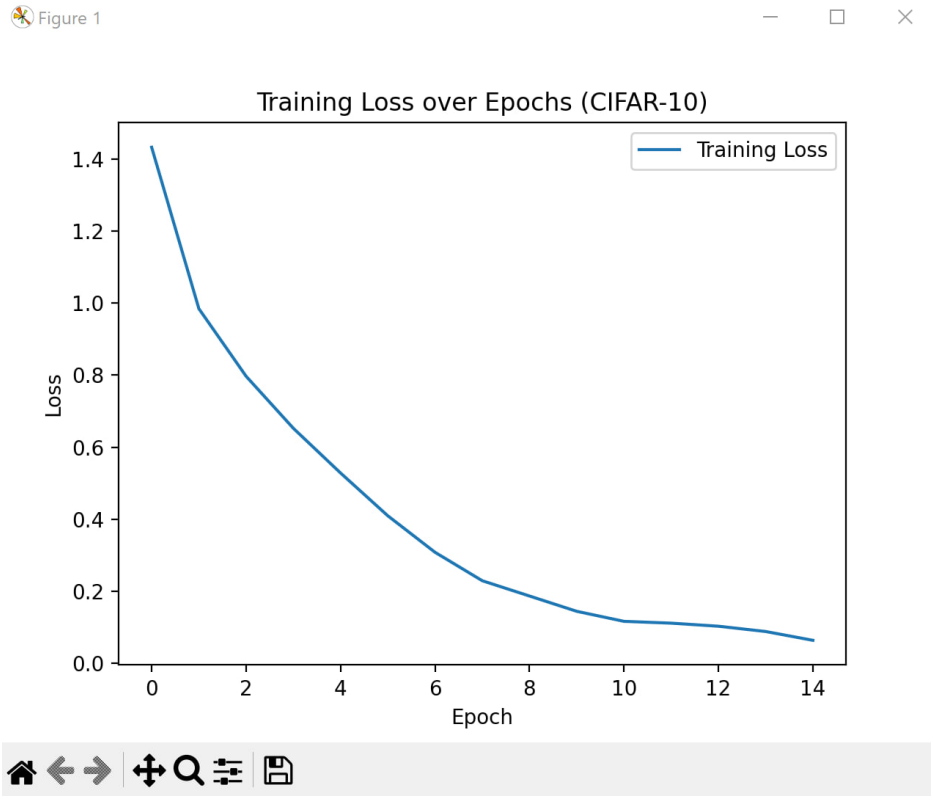
№ варианта	Выборка	Размер исходного изображения	Оптимизатор
3	CIFAR-10	32X32	SGD

Код программы:

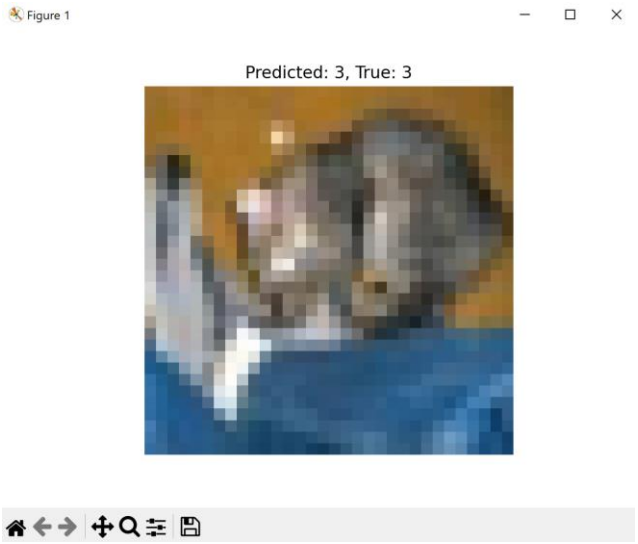
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465),
                         (0.2023, 0.1994, 0.2010))
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, 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__()
```

```
        # Вход: 3 канала (RGB)
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)  # 32x32 -> 16x16

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)  # 16x16 -> 8x8

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(64 * 8 * 8, 128)
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(128, 10)

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

model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

num_epochs = 15
train_losses = []

for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for data, target in train_loader:
        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 (CIFAR-10)')
plt.legend()
plt.show()

model.eval()
correct = 0
total = 0
with torch.no_grad():
    for data, target in test_loader:
```

```
        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]

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

def denormalize(img):
    img = img * torch.tensor((0.2023, 0.1994, 0.2010)).view(3, 1, 1)
    img = img + torch.tensor((0.4914, 0.4822, 0.4465)).view(3, 1, 1)
    return torch.clamp(img, 0, 1)

plt.imshow(np.transpose(denormalize(img).numpy(), (1, 2, 0)))
plt.title(f'Predicted: {pred_label}, True: {true_label}')
plt.axis('off')
plt.show()

Epoch 1/15, Loss: 1.4322
Epoch 2/15, Loss: 0.9844
Epoch 3/15, Loss: 0.7970
Epoch 4/15, Loss: 0.6527
Epoch 5/15, Loss: 0.5286
Epoch 6/15, Loss: 0.4099
Epoch 7/15, Loss: 0.3086
Epoch 8/15, Loss: 0.2298
Epoch 9/15, Loss: 0.1874
Epoch 10/15, Loss: 0.1449
Epoch 11/15, Loss: 0.1171
Epoch 12/15, Loss: 0.1121
Epoch 13/15, Loss: 0.1036
Epoch 14/15, Loss: 0.0889
Epoch 15/15, Loss: 0.0645
Test Accuracy: 71.14%
```



Test Accuracy: 71.14%



State-of-the-art результаты для CIFAR-10:

Согласно открытым источникам и техническим статьям (включая официальные руководства TensorFlow и PyTorch, а также материалы на *Machine Learning Mastery* и *keras.io*), точность (accuracy) современных моделей на наборе данных CIFAR-10 варьируется в широком диапазоне:

Базовые CNN, подобные обучающим примерам из официальных туториалов TensorFlow/PyTorch, показывают точность 50–70%.

Более продвинутые модели с регуляризацией, аугментацией и оптимизацией гиперпараметров достигают 85–90%.

Современные state-of-the-art (SOTA) архитектуры, такие как ResNet, DenseNet, EfficientNet, ConvMixer и другие гибриды CNN/ViT, могут достигать 96–99% точности в зависимости от глубины сети, стратегии обучения и ансамблирования.

Выводы: Полученные результаты подтверждают, что даже простая CNN без аугментации данных и сложных приёмов оптимизации способна достичь уровня около 70% точности, что соответствует диапазону типичных базовых моделей, представленных в официальных обучающих материалах по PyTorch и TensorFlow.
Тем не менее, для достижения SOTA-результатов (выше 95%) необходимы более сложные подходы: использование глубоких архитектур (ResNet, DenseNet, Vision Transformer), аугментация данных (например, random crop, flip, color jitter), применение трансферного обучения.

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