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

Худик А.А.

Проверил:

Крощенко А.А.

Брест 2024

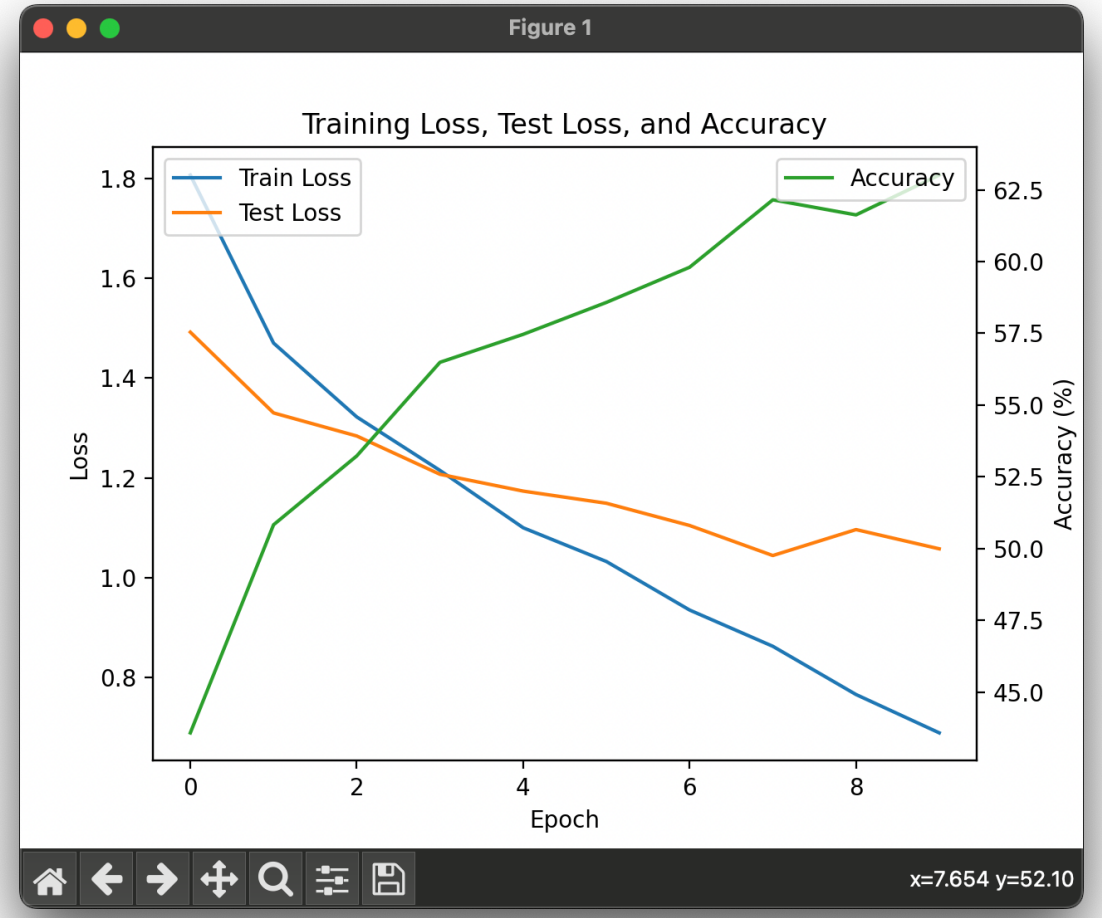
Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.

5	STL-10 (размеченная часть)	96X96	SGD

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import matplotlib

matplotlib.use("MacOSX")

# Check if a GPU is available and set device
device = torch.device("mps" if torch.mps.is_available() else "cpu")

# Parameters
batch_size = 32
learning_rate = 0.001
num_epochs = 10

# Load STL-10 data with normalization
transform = transforms.Compose([
    transforms.Resize((96, 96)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # normalization to range [-1, 1]
])

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

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Enhanced CNN architecture
class EnhancedCNN(nn.Module):
    def __init__(self):
        super(EnhancedCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(128 * 12 * 12, 256)
        self.fc2 = nn.Linear(256, 10)  # 10 classes in STL-10

    def forward(self, x):
        x = self.pool(torch.relu(self.bn1(self.conv1(x))))
        x = self.pool(torch.relu(self.bn2(self.conv2(x))))
        x = self.pool(torch.relu(self.bn3(self.conv3(x))))
        x = x.view(-1, 128 * 12 * 12)
        x = self.dropout(torch.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

# Initialize model, loss function, and optimizer
model = EnhancedCNN().to(device)  # Transfer model to GPU
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9, weight_decay=1e-4)

# Train and evaluate model
train_losses, test_losses, accuracies = [], [], []
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)  # Transfer data to GPU

        optimizer.zero_grad() # Zero the gradients
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    train_losses.append(running_loss / len(train_loader))

    # Evaluate on test set
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)  # Transfer data to GPU

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            test_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
```

```
    test_losses.append(test_loss / len(test_loader))
    accuracy = 100 * correct / total
    accuracies.append(accuracy)

    print(f"Epoch {epoch+1}/{num_epochs}, Train Loss: {train_losses[-1]:.4f}, "
          f"Test Loss: {test_losses[-1]:.4f}, Accuracy: {accuracy:.2f}%")

# Plot training loss, test loss, and accuracy
fig, ax1 = plt.subplots()

ax1.plot(train_losses, label='Train Loss', color='tab:blue')
ax1.plot(test_losses, label='Test Loss', color='tab:orange')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.legend(loc='upper left')

ax2 = ax1.twinx()
ax2.plot(accuracies, label='Accuracy', color='tab:green')
ax2.set_ylabel('Accuracy (%)')
ax2.legend(loc='upper right')

plt.title("Training Loss, Test Loss, and Accuracy")
plt.show()
```




```
Epoch 10/10, Train Loss: 0.6892, Test Loss: 1.0580, Accuracy: 63.01%
```

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