Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «ОИвИС»

Тема: “Обучение классификаторов средствами библиотеки PyTorch”

Выполнил:

Студент 4 курса

Группы ИИ-23

Медведь П.В.

Проверила:

Андренко К.В.

Брест 2025

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

Вариант 5.

Выборка: STL-10 (размеченная часть)
Изображения: 96*96
Оптимизатор: SGD

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.

Код программы:

```
from torchvision import transforms
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import numpy as np
data_dir = './data'
train_transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.RandomResizedCrop(96, scale=(0.8, 1.0)),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.STL10(data_dir, split='train', download=True,
                               transform=train_transform),
    batch_size=128, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.STL10(data_dir, split='test', download=True,
                               transform=test_transform),
    batch_size=128, shuffle=False)
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(128 * 12 * 12, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
    def forward(self, x):

        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))
        x = x.view(-1, 128 * 12 * 12)
        x = self.dropout(self.relu(self.fc1(x)))
```

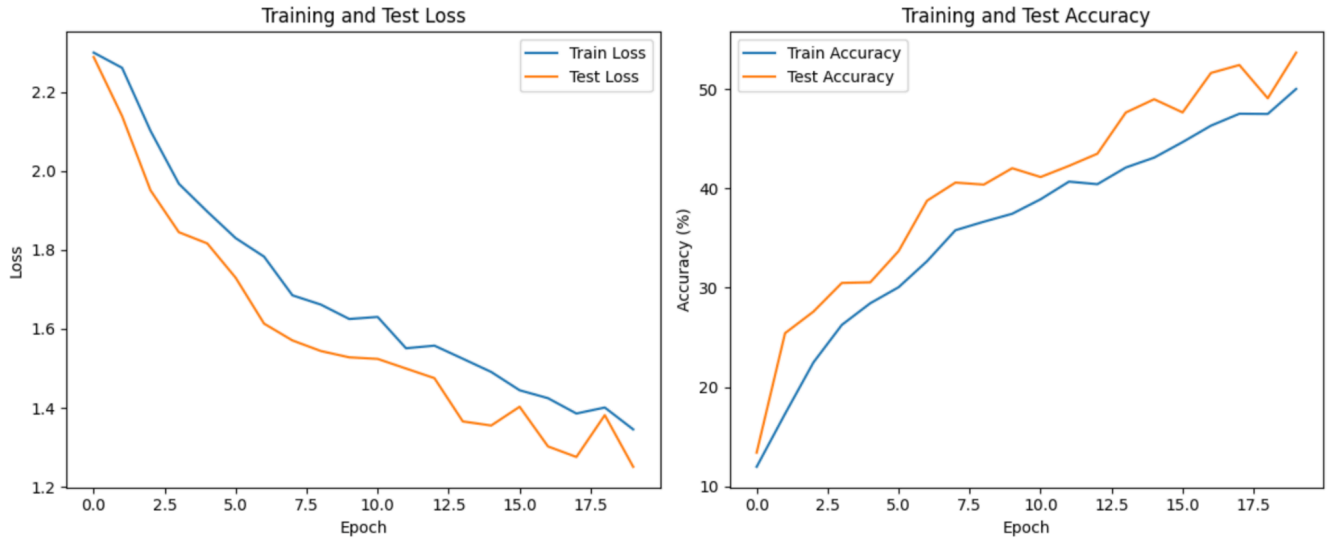
```
        x = self.dropout(self.relu(self.fc2(x)))
        x = self.fc3(x)
        return x
model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight_decay=1e-4)
def train(model, loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct += (predicted == labels).sum().item()
        total += labels.size(0)
    accuracy = 100 * correct / total
    return running_loss / len(loader), accuracy
def test(model, loader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    accuracy = 100 * correct / total
    return running_loss / len(loader), accuracy
device = torch.device('cuda')
model = model.to(device)
train_losses = []
test_losses = []
train_accuracies = []
test_accuracies = []
num_epochs = 20
for epoch in range(num_epochs):
    train_loss, train_accuracy = train(model, train_loader, criterion, optimizer, device)
    test_loss, test_accuracy = test(model, test_loader, criterion, device)
```

```
    train_losses.append(train_loss)
    test_losses.append(test_loss)
    train_accuracies.append(train_accuracy)
    test_accuracies.append(test_accuracy)
            print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.2f}%, '
        f'Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.2f}%')
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Test Loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(test_accuracies, label='Test Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.title('Training and Test Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
def imshow(img):
    img = img / 2 + 0.5
    np_img = img.numpy()
    plt.imshow(np.transpose(np_img, (1, 2, 0)))
    plt.axis('off')
    plt.show()
def test_random_image(model, loader, device):
    model.eval()
    images, labels = next(iter(loader))
    images, labels = images.to(device), labels.to(device)
    import random
    index = random.randint(0, images.size(0) - 1)
    image = images[index].unsqueeze(0)
    label = labels[index].item()
    output = model(image)
    _, predicted = torch.max(output, 1)
    predicted = predicted.item()
    imshow(image.cpu().squeeze())
    print(f'Predicted: {predicted}, Actual: {label}')
test_random_image(model, test_loader, device)
print(f"Final Train Accuracy: {train_accuracies[-1]:.2f}%")
print(f"Final Test Accuracy: {test_accuracies[-1]:.2f}%")
```

Результат работы программы:

```
Epoch 1/20, Train Loss: 2.2994, Train Accuracy: 11.96%, Test Loss: 2.2882, Test Accuracy: 13.39%
Epoch 2/20, Train Loss: 2.2610, Train Accuracy: 17.30%, Test Loss: 2.1388, Test Accuracy: 25.43%
Epoch 3/20, Train Loss: 2.1016, Train Accuracy: 22.48%, Test Loss: 1.9509, Test Accuracy: 27.60%
Epoch 4/20, Train Loss: 1.9674, Train Accuracy: 26.26%, Test Loss: 1.8448, Test Accuracy: 30.49%
Epoch 5/20, Train Loss: 1.8970, Train Accuracy: 28.44%, Test Loss: 1.8164, Test Accuracy: 30.55%
Epoch 6/20, Train Loss: 1.8298, Train Accuracy: 30.06%, Test Loss: 1.7295, Test Accuracy: 33.70%
Epoch 7/20, Train Loss: 1.7829, Train Accuracy: 32.68%, Test Loss: 1.6134, Test Accuracy: 38.77%
Epoch 8/20, Train Loss: 1.6850, Train Accuracy: 35.80%, Test Loss: 1.5708, Test Accuracy: 40.60%
Epoch 9/20, Train Loss: 1.6616, Train Accuracy: 36.66%, Test Loss: 1.5440, Test Accuracy: 40.40%
Epoch 10/20, Train Loss: 1.6251, Train Accuracy: 37.46%, Test Loss: 1.5280, Test Accuracy: 42.05%
Epoch 11/20, Train Loss: 1.6303, Train Accuracy: 38.92%, Test Loss: 1.5241, Test Accuracy: 41.16%
Epoch 12/20, Train Loss: 1.5510, Train Accuracy: 40.70%, Test Loss: 1.4996, Test Accuracy: 42.29%
Epoch 13/20, Train Loss: 1.5576, Train Accuracy: 40.44%, Test Loss: 1.4753, Test Accuracy: 43.51%
Epoch 14/20, Train Loss: 1.5245, Train Accuracy: 42.12%, Test Loss: 1.3658, Test Accuracy: 47.66%
Epoch 15/20, Train Loss: 1.4910, Train Accuracy: 43.12%, Test Loss: 1.3554, Test Accuracy: 49.00%
Epoch 16/20, Train Loss: 1.4447, Train Accuracy: 44.68%, Test Loss: 1.4029, Test Accuracy: 47.67%
Epoch 17/20, Train Loss: 1.4246, Train Accuracy: 46.34%, Test Loss: 1.3025, Test Accuracy: 51.65%
Epoch 18/20, Train Loss: 1.3858, Train Accuracy: 47.54%, Test Loss: 1.2758, Test Accuracy: 52.45%
Epoch 19/20, Train Loss: 1.4010, Train Accuracy: 47.52%, Test Loss: 1.3817, Test Accuracy: 49.10%
Epoch 20/20, Train Loss: 1.3456, Train Accuracy: 50.04%, Test Loss: 1.2509, Test Accuracy: 53.70%
```

График изменения ошибок:



SOTA-результаты для выборки:

Model Name	Percentage correct	Paper Title	Repository
NAT-M3	97.8	Neural Architecture Transfer	

Nat-M3 — это не простая сверточная сеть, а современная архитектура Transformer (специфически, вариант Vision Transformer, ViT), которая использует механизм внимания (attention). Вместо того чтобы скользить фильтрами по изображению, как это делает свёртка, она разбивает изображение на участки (патчи) и анализирует глобальные взаимосвязи между всеми частями изображения одновременно. Это позволяет ей улавливать более сложные и абстрактные зависимости.

Почему разница в точности обусловлена именно этим:

1) Архитектурное превосходство Transformer над простой CNN:

CNN (3 слоя): Это относительно мелкая сеть. Она хорошо улавливает локальные паттерны (края, текстуры), но ей не хватает "глубины" и механизмов для эффективного моделирования глобального контекста. Сложные объекты в STL-10 требуют понимания взаимосвязей между удалёнными друг от друга частями изображения.

Nat-M3 (Transformer): Механизм внимания позволяет сети "смотреть" на все части изображения сразу и взвешивать их важность для принятия решения. Например, чтобы отличить "машину" от "животного", сеть может одновременно связать информацию о колёсах, фарах и окнах, даже если они находятся далеко друг от друга в кадре. Это гораздо более мощный подход для сложных задач.

2) Масштаб модели (Количество параметров):

CNN: Имеет несколько миллионов параметров.
Nat-M3: Это большая модель с десятками или сотнями миллионов параметров. Большая ёмкость модели позволяет ей запомнить и выучить гораздо более сложные и разнообразные features (признаки) из данных.

3) Использование Продвинутых Методов Обучения:

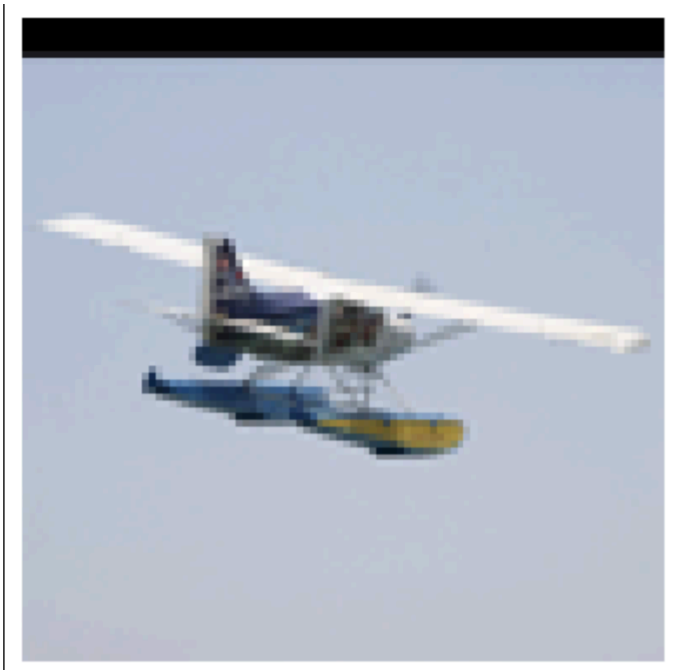
CNN: Обучалась "с нуля" на размеченных данных STL-10.
Nat-M3: Использовала предобучение на огромном наборе данных (например, ImageNet-21k), а затем дообучалась на STL-10.

4) Современные трюки оптимизации:

Модели вроде nat-M3 используют современные оптимизаторы (AdamW), сложные расписания скорости обучения, пакетную нормализацию, dropout с высокими коэффициентами и т.д. В совокупности это даёт значительный прирост.

Вывод
Разница в 45% точности — это ожидаемый разрыв между небольшой моделью, обученной с нуля, и современным state-of-the-art подходом.

Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата):



Вывод: Научился обучать классификаторы средствами библиотеки PyTorch.