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

Швороб В.А.

Проверила:

Андренко К.В.

Брест 2025

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

Вариант 12.

| **Выборка** | **Размер исходного изображения** | **Оптимизатор** |
|---|---|---|
| Fashion-MNIST | **28x28** | Adadelta |

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 (https://paperswithcode.com/task/image-classification). 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:

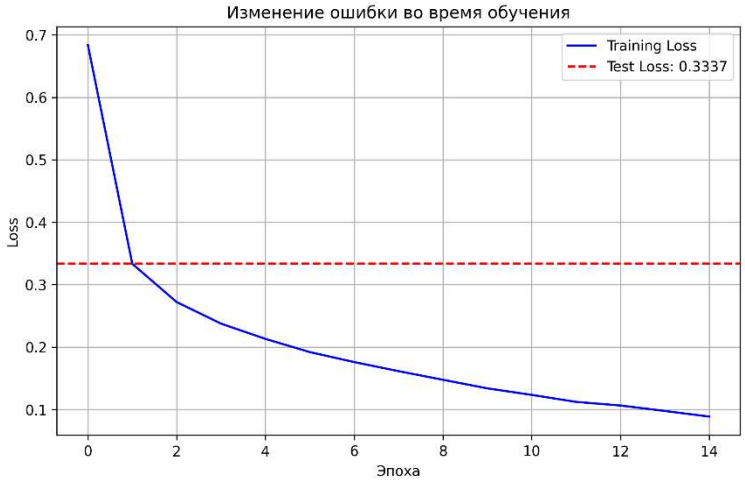
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import DataLoader
import requests
from PIL import Image
import io
import os

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Используемое устройство: {device}")

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.FashionMNIST(
    root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.FashionMNIST(
    root='./data', train=False, download=True, transform=transform)

batch_size = 128
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

classes = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 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 * 3 * 3, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.25)

    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 * 3 * 3)

        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
```

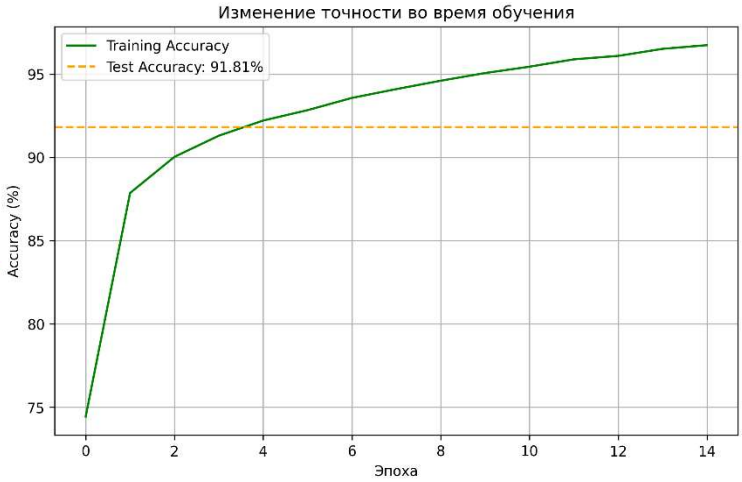
```
        return x


model = SimpleCNN().to(device)
print(model)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters())


def train_model(model, train_loader, criterion, optimizer, num_epochs=15):
    train_losses = []
    train_accuracies = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for batch_idx, (data, target) in enumerate(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()
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

            if batch_idx % 100 == 0:
                print(
                    f'Epoch: {epoch + 1}/{num_epochs} | Batch: {batch_idx}/{len(train_loader)} | Loss: {loss.item():.4f}')

        epoch_loss = running_loss / len(train_loader)
        epoch_accuracy = 100 * correct / total

        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_accuracy)

        print(f'Epoch {epoch + 1}/{num_epochs} | Loss: {epoch_loss:.4f} | Accuracy: {epoch_accuracy:.2f}%')

    return train_losses, train_accuracies


print("Начало обучения...")
train_losses, train_accuracies = train_model(model, train_loader, criterion, optimizer, num_epochs=15)


def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    test_loss = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
```

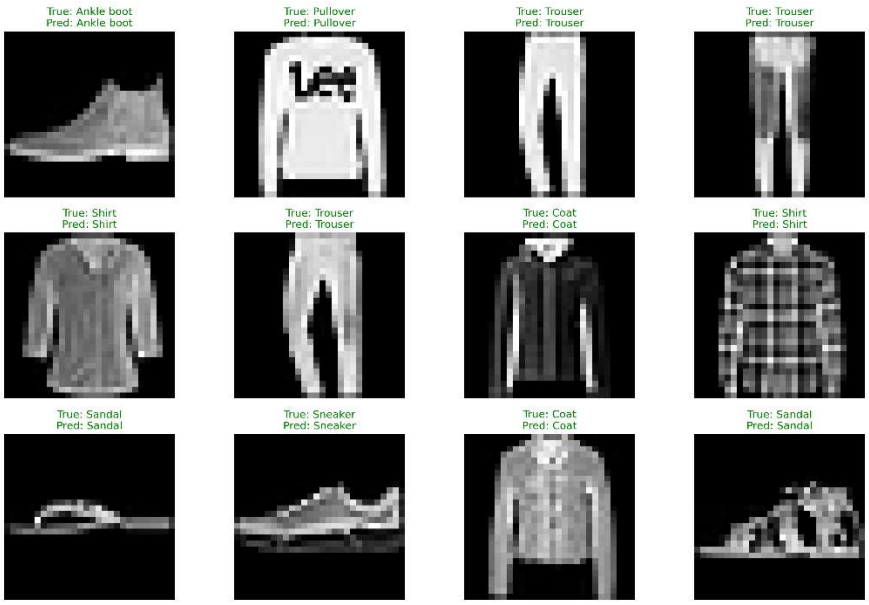
```
            output = model(data)
            test_loss += criterion(output, target).item()
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

    accuracy = 100 * correct / total
    avg_loss = test_loss / len(test_loader)

    print(f'Test Loss: {avg_loss:.4f} | Test Accuracy: {accuracy:.2f}%')
    return accuracy, avg_loss


print("\nОценка на тестовой выборке:")
test_accuracy, test_loss = evaluate_model(model, test_loader)

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.plot(train_losses, 'b-', label='Training Loss')
plt.axhline(y=test_loss, color='r', linestyle='--', label=f'Test Loss: {test_loss:.4f}')
plt.title('Изменение ошибки во время обучения')
plt.xlabel('Эпоха')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, 'g-', label='Training Accuracy')
plt.axhline(y=test_accuracy, color='orange', linestyle='--', label=f'Test Accuracy: {test_accuracy:.2f}%')
plt.title('Изменение точности во время обучения')
plt.xlabel('Эпоха')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.savefig('training_results.png', dpi=300, bbox_inches='tight')
plt.show()


def visualize_predictions(model, test_loader, num_images=12):
    model.eval()
    data_iter = iter(test_loader)
    images, labels = next(data_iter)
    images, labels = images.to(device), labels.to(device)

    with torch.no_grad():
        outputs = model(images[:num_images])
        _, predicted = torch.max(outputs, 1)

    images = images.cpu()

    fig, axes = plt.subplots(3, 4, figsize=(15, 10))
    for i, ax in enumerate(axes.flat):
        if i < num_images:
            img = images[i].squeeze() * 0.5 + 0.5
            ax.imshow(img, cmap='gray')
            ax.set_title(f'True: {classes[labels[i]]}\nPred: {classes[predicted[i]]}',
                         color='green' if labels[i] == predicted[i] else 'red')
            ax.axis('off')

    plt.tight_layout()
```

```
    plt.savefig('predictions_visualization.png', dpi=300, bbox_inches='tight')
    plt.show()


print("\nВизуализация предсказаний на тестовых изображениях:")
visualize_predictions(model, test_loader)


def predict_single_image(model, image_path=None, url=None):
    model.eval()

    if url:
        response = requests.get(url)
        image = Image.open(io.BytesIO(response.content))
    else:
        image = Image.open(image_path)

    transform_single = transforms.Compose([
        transforms.Grayscale(),
        transforms.Resize((28, 28)),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])

    image_tensor = transform_single(image).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(image_tensor)
        probabilities = torch.nn.functional.softmax(output[0], dim=0)
        _, predicted = torch.max(output, 1)

    plt.figure(figsize=(8, 4))

    plt.subplot(1, 2, 1)
    plt.imshow(image.convert('L'), cmap='gray')
    plt.title(f'Предсказание: {classes[predicted.item()]}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    y_pos = np.arange(len(classes))
    plt.barh(y_pos, probabilities.cpu().numpy())
    plt.yticks(y_pos, classes)
    plt.xlabel('Вероятность')
    plt.title('Распределение вероятностей')
    plt.tight_layout()

    plt.savefig('single_prediction.png', dpi=300, bbox_inches='tight')
    plt.show()

    return classes[predicted.item()], probabilities.cpu().numpy()
```

Вывод программы:




Используемое устройство: cuda

```
SimpleCNN(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=1152, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=10, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.25, inplace=False)
)
```

Начало обучения...
Epoch: 1/15 | Batch: 0/469 | Loss: 2.3062
Epoch: 1/15 | Batch: 100/469 | Loss: 0.8360
Epoch: 1/15 | Batch: 200/469 | Loss: 0.5722
Epoch: 1/15 | Batch: 300/469 | Loss: 0.5620
Epoch: 1/15 | Batch: 400/469 | Loss: 0.5357
Epoch 1/15 | Loss: 0.6835 | Accuracy: 74.44%
Epoch: 2/15 | Batch: 0/469 | Loss: 0.5111
Epoch: 2/15 | Batch: 100/469 | Loss: 0.4009
Epoch: 2/15 | Batch: 200/469 | Loss: 0.3256
Epoch: 2/15 | Batch: 300/469 | Loss: 0.3406

Epoch: 2/15 | Batch: 400/469 | Loss: 0.2522
Epoch 2/15 | Loss: 0.3333 | Accuracy: 87.86%
Epoch: 3/15 | Batch: 0/469 | Loss: 0.2596
Epoch: 3/15 | Batch: 100/469 | Loss: 0.3253
Epoch: 3/15 | Batch: 200/469 | Loss: 0.2618
Epoch: 3/15 | Batch: 300/469 | Loss: 0.1694
Epoch: 3/15 | Batch: 400/469 | Loss: 0.2985
Epoch 3/15 | Loss: 0.2720 | Accuracy: 90.03%
Epoch: 4/15 | Batch: 0/469 | Loss: 0.2252
Epoch: 4/15 | Batch: 100/469 | Loss: 0.3716
Epoch: 4/15 | Batch: 200/469 | Loss: 0.2311
Epoch: 4/15 | Batch: 300/469 | Loss: 0.1865
Epoch: 4/15 | Batch: 400/469 | Loss: 0.2617
Epoch 4/15 | Loss: 0.2375 | Accuracy: 91.30%
Epoch: 5/15 | Batch: 0/469 | Loss: 0.1319
Epoch: 5/15 | Batch: 100/469 | Loss: 0.1148
Epoch: 5/15 | Batch: 200/469 | Loss: 0.1236
Epoch: 5/15 | Batch: 300/469 | Loss: 0.1984
Epoch: 5/15 | Batch: 400/469 | Loss: 0.2695
Epoch 5/15 | Loss: 0.2131 | Accuracy: 92.21%
Epoch: 6/15 | Batch: 0/469 | Loss: 0.1952
Epoch: 6/15 | Batch: 100/469 | Loss: 0.2494
Epoch: 6/15 | Batch: 200/469 | Loss: 0.1623
Epoch: 6/15 | Batch: 300/469 | Loss: 0.1756
Epoch: 6/15 | Batch: 400/469 | Loss: 0.2763
Epoch 6/15 | Loss: 0.1919 | Accuracy: 92.83%
Epoch: 7/15 | Batch: 0/469 | Loss: 0.1915
Epoch: 7/15 | Batch: 100/469 | Loss: 0.1260
Epoch: 7/15 | Batch: 200/469 | Loss: 0.1104
Epoch: 7/15 | Batch: 300/469 | Loss: 0.1695
Epoch: 7/15 | Batch: 400/469 | Loss: 0.1616
Epoch 7/15 | Loss: 0.1759 | Accuracy: 93.57%
Epoch: 8/15 | Batch: 0/469 | Loss: 0.0877
Epoch: 8/15 | Batch: 100/469 | Loss: 0.1579
Epoch: 8/15 | Batch: 200/469 | Loss: 0.1194
Epoch: 8/15 | Batch: 300/469 | Loss: 0.1869
Epoch: 8/15 | Batch: 400/469 | Loss: 0.2609
Epoch 8/15 | Loss: 0.1614 | Accuracy: 94.10%
Epoch: 9/15 | Batch: 0/469 | Loss: 0.1102
Epoch: 9/15 | Batch: 100/469 | Loss: 0.2526
Epoch: 9/15 | Batch: 200/469 | Loss: 0.1013
Epoch: 9/15 | Batch: 300/469 | Loss: 0.1045
Epoch: 9/15 | Batch: 400/469 | Loss: 0.2434
Epoch 9/15 | Loss: 0.1476 | Accuracy: 94.60%
Epoch: 10/15 | Batch: 0/469 | Loss: 0.1906
Epoch: 10/15 | Batch: 100/469 | Loss: 0.2170
Epoch: 10/15 | Batch: 200/469 | Loss: 0.2072
Epoch: 10/15 | Batch: 300/469 | Loss: 0.1484
Epoch: 10/15 | Batch: 400/469 | Loss: 0.1255
Epoch 10/15 | Loss: 0.1338 | Accuracy: 95.06%
Epoch: 11/15 | Batch: 0/469 | Loss: 0.1078
Epoch: 11/15 | Batch: 100/469 | Loss: 0.0606
Epoch: 11/15 | Batch: 200/469 | Loss: 0.0769
Epoch: 11/15 | Batch: 300/469 | Loss: 0.1342
Epoch: 11/15 | Batch: 400/469 | Loss: 0.1094
Epoch 11/15 | Loss: 0.1234 | Accuracy: 95.44%
Epoch: 12/15 | Batch: 0/469 | Loss: 0.1341
Epoch: 12/15 | Batch: 100/469 | Loss: 0.0569

Epoch: 12/15 | Batch: 200/469 | Loss: 0.0865
Epoch: 12/15 | Batch: 300/469 | Loss: 0.0725
Epoch: 12/15 | Batch: 400/469 | Loss: 0.0969
Epoch 12/15 | Loss: 0.1122 | Accuracy: 95.89%
Epoch: 13/15 | Batch: 0/469 | Loss: 0.0481
Epoch: 13/15 | Batch: 100/469 | Loss: 0.1102
Epoch: 13/15 | Batch: 200/469 | Loss: 0.1152
Epoch: 13/15 | Batch: 300/469 | Loss: 0.1049
Epoch: 13/15 | Batch: 400/469 | Loss: 0.1253
Epoch 13/15 | Loss: 0.1064 | Accuracy: 96.09%
Epoch: 14/15 | Batch: 0/469 | Loss: 0.0942
Epoch: 14/15 | Batch: 100/469 | Loss: 0.0730
Epoch: 14/15 | Batch: 200/469 | Loss: 0.1689
Epoch: 14/15 | Batch: 300/469 | Loss: 0.0878
Epoch: 14/15 | Batch: 400/469 | Loss: 0.1255
Epoch 14/15 | Loss: 0.0976 | Accuracy: 96.52%
Epoch: 15/15 | Batch: 0/469 | Loss: 0.0739
Epoch: 15/15 | Batch: 100/469 | Loss: 0.0395
Epoch: 15/15 | Batch: 200/469 | Loss: 0.0684
Epoch: 15/15 | Batch: 300/469 | Loss: 0.1451
Epoch: 15/15 | Batch: 400/469 | Loss: 0.0474
Epoch 15/15 | Loss: 0.0886 | Accuracy: 96.74%

Оценка на тестовой выборке:
Test Loss: 0.3337 | Test Accuracy: 91.81%

Точность нашей модели на тестовой выборке: 91.81%

State-of-the-Art результаты для Fashion-MNIST:
- Современные модели достигают точности 96-97%
- Лучшие результаты обычно показывают:
  * ResNet-архитектуры: ~96.7%
  * EfficientNet: ~97.0%
  * Transformer-based модели: ~97.2%
  * Ансамбли моделей: до 97.5%

Выводы о нашей модели:
1. Наша простая CNN достигла точности 91.81%
2. Архитектура использует только базовые слои:
   - 3 сверточных слоя
   - MaxPooling для уменьшения размерности
   - 3 полносвязных слоя
   - ReLU активации и Dropout для регуляризации
3. Оптимизатор: Adadelta
4. Функция потерь: CrossEntropyLoss

Модель показывает хорошие результаты для простой архитектуры

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