Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине: «ОИИС»

# Тема: «Обучение классификаторов средствами библиотеки PyTorch»

Выполнила:

Студентка 4 курса

Группы ИИ-23

Осипова А.Е.

Проверила:

Андренко К.В.

Брест 2025

Цель: 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.

**Вариант:**

****

**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 numpy as np

from tqdm import tqdm

train\_transform = transforms.Compose([

    transforms.RandomHorizontalFlip(),

    transforms.RandomRotation(10),

    transforms.ToTensor(),

    transforms.Normalize((0.5,), (0.5,))

])

test\_transform = transforms.Compose([

    transforms.ToTensor(),

    transforms.Normalize((0.5,), (0.5,))

])

train\_loader = torch.utils.data.DataLoader(

    torchvision.datasets.FashionMNIST('/files/', train=True, download=True,

                                  transform=train\_transform),

    batch\_size=128, shuffle=True)

test\_loader = torch.utils.data.DataLoader(

    torchvision.datasets.FashionMNIST('/files/', train=False, download=True,

                                  transform=test\_transform),

    batch\_size=256, shuffle=False)

class NN(nn.Module):

    def \_\_init\_\_(self):

        super(NN, self).\_\_init\_\_()

        self.conv1 = nn.Conv2d(1, 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.pool = nn.MaxPool2d(2, 2)

        self.conv3 = nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1)

        self.bn3 = nn.BatchNorm2d(128)

        self.conv4 = nn.Conv2d(128, 256, kernel\_size=3, stride=1, padding=1)

        self.bn4 = nn.BatchNorm2d(256)

        self.conv5 = nn.Conv2d(256, 512, kernel\_size=3, stride=1, padding=1)

        self.bn5 = nn.BatchNorm2d(512)

        self.conv6 = nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1)

        self.bn6 = nn.BatchNorm2d(512)

        self.global\_avg\_pool = nn.AdaptiveAvgPool2d((1, 1))

        self.fc1 = nn.Linear(512, 256)

        self.fc2 = nn.Linear(256, 128)

        self.fc3 = nn.Linear(128, 10)

        self.relu = nn.LeakyReLU()

        self.dropout4 = nn.Dropout(0.5)

    def forward(self, x):

        x = self.pool(self.relu(self.bn1(self.conv1(x))))

        x = self.pool(self.relu(self.bn2(self.conv2(x))))

        x = self.pool(self.relu(self.bn3(self.conv3(x))))

        x = self.relu(self.bn4(self.conv4(x)))

        x = self.pool(self.relu(self.bn5(self.conv5(x))))

        x = self.relu(self.bn6(self.conv6(x)))

        x = self.global\_avg\_pool(x)

        x = x.view(-1, 512)

        x = self.relu(self.fc1(x))

        x = self.dropout4(x)

        x = self.relu(self.fc2(x))

        x = self.fc3(x)

        return x

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model = NN().to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.00003)

def train(model, loader, criterion, optimizer, device):

    model.train()

    running\_loss = 0.0

    correct, total = 0, 0

    progress = tqdm(loader, desc="Training", leave=False)

    for images, labels in progress:

        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)

        progress.set\_postfix(loss=loss.item())

    accuracy = 100 \* correct / total

    return running\_loss / len(loader), accuracy

def test(model, loader, criterion, device):

    model.eval()

    running\_loss = 0.0

    correct, total = 0, 0

    progress = tqdm(loader, desc="Testing", leave=False)

    with torch.no\_grad():

        for images, labels in progress:

            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)

            progress.set\_postfix(loss=loss.item())

    accuracy = 100 \* correct / total

    return running\_loss / len(loader), accuracy

train\_losses, test\_losses = [], []

train\_accuracies, test\_accuracies = [], []

num\_epochs = 50

for epoch in range(num\_epochs):

    print(f"\nEpoch {epoch+1}/{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"Train Loss: {train\_loss:.4f}, Train Acc: {train\_accuracy:.2f}% | "

          f"Test Loss: {test\_loss:.4f}, Test Acc: {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("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("Accuracy")

plt.legend()

plt.show()

classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot')

def imshow(img):

    img = img.cpu().numpy().transpose((1, 2, 0))

    mean = np.array([0.5])

    std = np.array([0.5])

    img = std \* img + mean

    img = np.clip(img, 0, 1)

    plt.imshow(img, cmap='gray')

    plt.axis("off")

dataiter = iter(test\_loader)

images, labels = next(dataiter)

images, labels = images.to(device), labels.to(device)

model.eval()

with torch.no\_grad():

    outputs = model(images)

    \_, predicted = torch.max(outputs, 1)

for i in range(8):

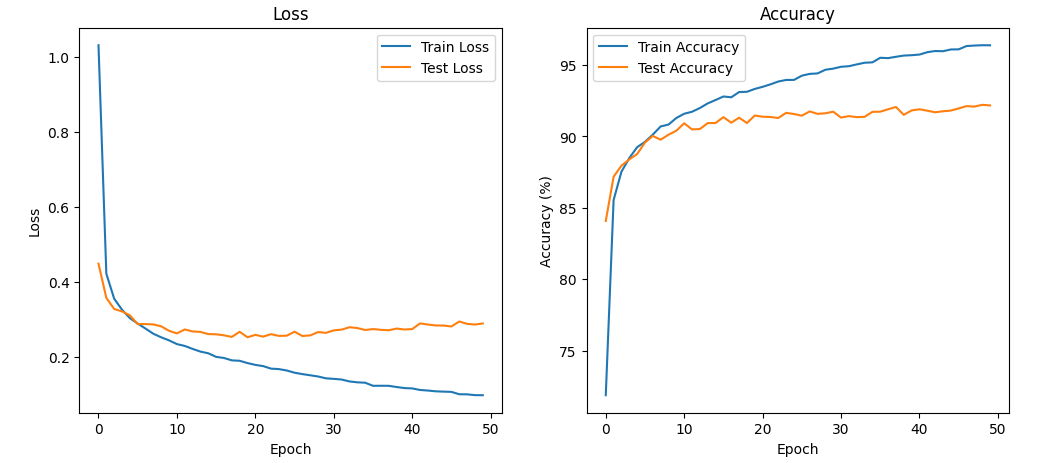
    plt.subplot(2,4,i+1)

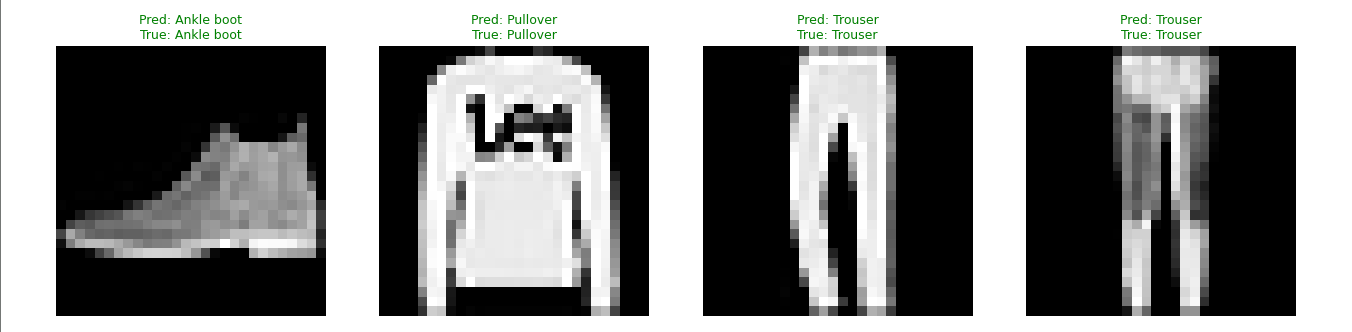
    imshow(images[i])

    plt.title(f"Pred: {classes[predicted[i]]}\nTrue: {classes[labels[i]]}",

              fontsize=9, color=("green" if predicted[i]==labels[i] else "red"))

plt.show()



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