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-21

Корпач Д.Р.

Проверил:

Крощенко А.А.

Брест 2024

Вариант 12.
Выборка: Fashion-MNIST.
Размер исходного изображения: 28*28
Оптимизатор: Adadelta.
Цель: научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения
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);

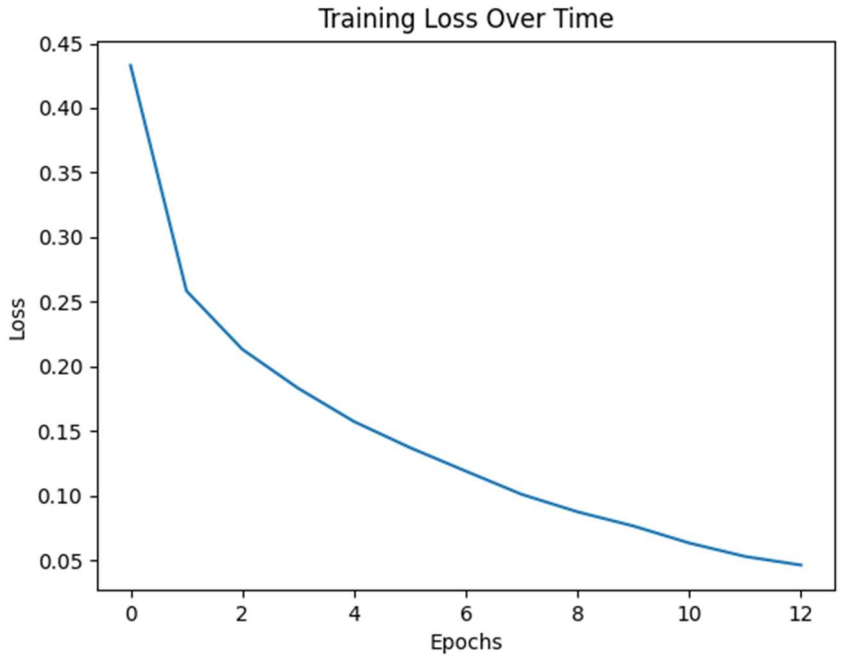
Задание 2. Реализовать визуализацию работы СНС из пункта 1 (выбор и подачу на архитектуру произвольного изображения с выводом результата);
Код программы:

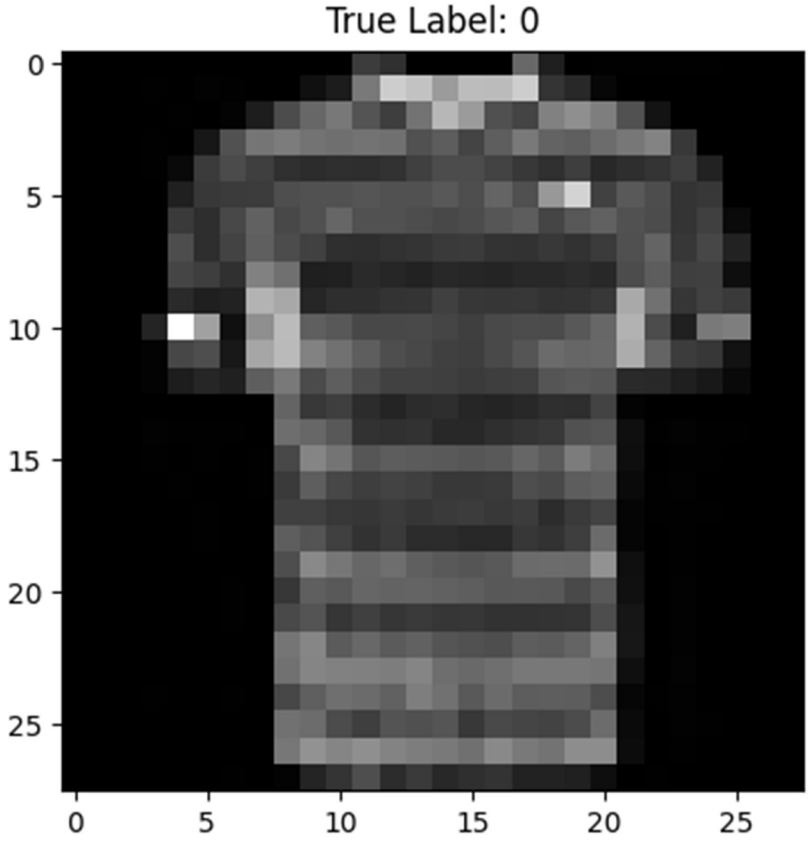
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import random
import numpy as np

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

train_set = torchvision.datasets.FashionMNIST(root='./data', train=True,
download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_set, batch_size=64, shuffle=True)

test_set = torchvision.datasets.FashionMNIST(root='./data', train=False,
download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=64, shuffle=False)

# Определение CNN модели
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        # Первый сверточный слой
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Второй сверточный слой
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        # Полносвязный слой
        self.fc1 = nn.Linear(64 * 7 * 7, 128)
        self.relu3 = nn.ReLU()

        # Выходной слой
```

```
        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 = x.view(-1, 64 * 7 * 7)  # Векторизация
        x = self.relu3(self.fc1(x))
        x = self.fc2(x)
        return x

# Инициализация модели, функции потерь и оптимизатора
model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters())

def train_model(model, train_loader, criterion, optimizer, epochs=13):
    model.train()
    loss_history = []

    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            # Обнуляем градиенты
            optimizer.zero_grad()

            # Прямой проход
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            # Обратный проход и оптимизация
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        avg_loss = running_loss / len(train_loader)
        loss_history.append(avg_loss)
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}")

    return loss_history

def test_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy:.2f}%')
    return accuracy

def visualize_and_classify_image(model, dataset, index=None):
    if index is None:
        index = random.randint(0, len(dataset) - 1)

    image, label = dataset[index]
```

```
    img = image.numpy().squeeze()

    plt.imshow(img, cmap='gray')
    plt.title(f"True Label: {label}")
    plt.show()
    image = image.unsqueeze(0)

    model.eval()
    with torch.no_grad():
        output = model(image)
        _, predicted = torch.max(output, 1)
    print(f"Predicted Class: {predicted.item()}")
loss_history = train_model(model, train_loader, criterion, optimizer, epochs=13)

test_accuracy = test_model(model, test_loader)

plt.plot(loss_history)
plt.title('Training Loss Over Time')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

visualize_and_classify_image(model, test_set, index=35)
```

```
Epoch [1/13], Loss: 0.4329
Epoch [2/13], Loss: 0.2585
Epoch [3/13], Loss: 0.2133
Epoch [4/13], Loss: 0.1832
Epoch [5/13], Loss: 0.1574
Epoch [6/13], Loss: 0.1373
Epoch [7/13], Loss: 0.1191
Epoch [8/13], Loss: 0.1011
Epoch [9/13], Loss: 0.0876
Epoch [10/13], Loss: 0.0766
Epoch [11/13], Loss: 0.0634
Epoch [12/13], Loss: 0.0530
Epoch [13/13], Loss: 0.0464
Test Accuracy: 91.66%
```






```
Predicted Class: 0
```