Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине «ОИвИС»

Тема: “ Конструирование моделей на базе предобученных нейронных сетей”

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.
Предобученная модель: AlexNet
Цель: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС.

Задание 1. Для заданной выборки и архитектуры предобученной нейронной организовать процесс обучения НС, предварительно изменив структуру слоев, в соответствии с предложенной выборкой. Использовать тот же оптимизатор, что и в ЛР №1. Построить график изменения ошибки и оценить эффективность обучения на тестовой выборке;

Задание 2. Сравнить полученные результаты с результатами, полученными на кастомных архитектурах из ЛР №1;
Код программы:

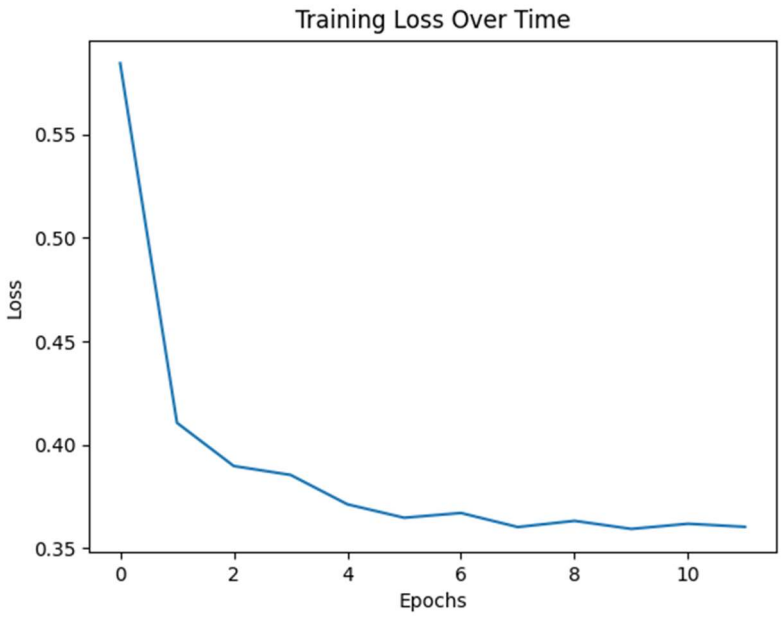
```
 import torchvision.transforms as transforms

import matplotlib.pyplot as plt

import random

from torchvision import models


# Преобразование для FashionMNIST (изменение размера до 224x224 и преобразование в трехканальное изображение)

transform = transforms.Compose([

    transforms.Resize((224, 224)),  # Изменение размера до 224x224

    transforms.Grayscale(num_output_channels=3),  # Преобразование в трехканальное изображение

    transforms.ToTensor(),

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

])


# Загрузка тренировочного и тестового набора

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)


# Использование предобученной модели AlexNet

model = models.alexnet(pretrained=True)
```

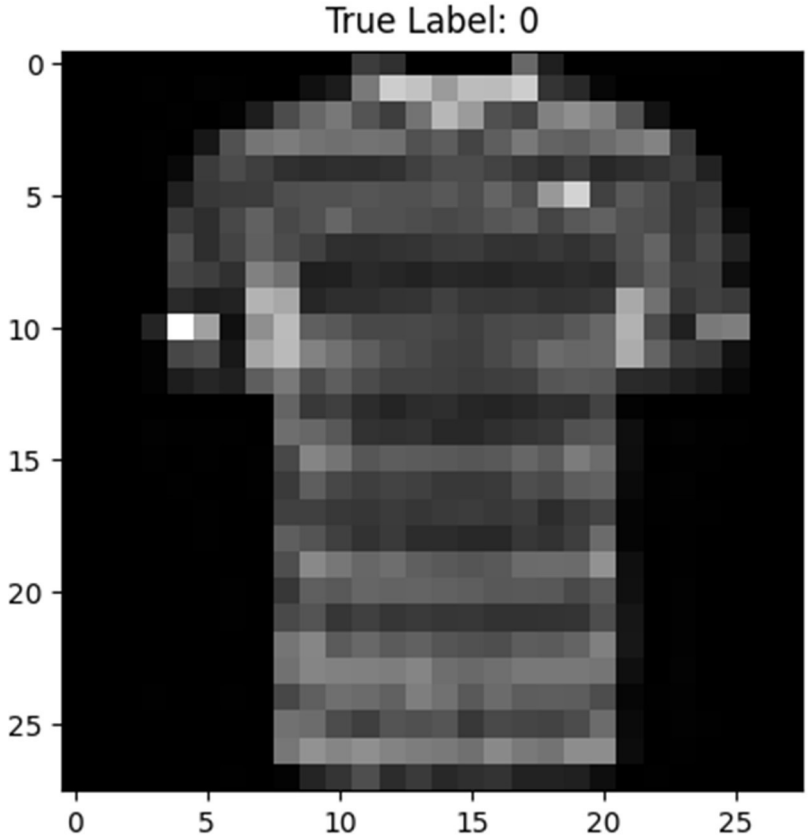
```
# Замена последнего слоя для предсказания 10 классов вместо 1000
model.classifier[6] = nn.Linear(model.classifier[6].in_features, 10)

# Замораживаем начальные слои, чтобы не изменять предобученные веса
for param in model.features.parameters():
    param.requires_grad = False

# Функция потерь и оптимизатор
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters())

# Функция для тренировки модели
def train_model(model, train_loader, criterion, optimizer, epochs=10):
    model.train()
    loss_history = []

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

            outputs = model(inputs)
            loss = criterion(outputs, labels)

            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

# Функция для тестирования модели
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

# Функция для визуализации 5 предсказанных изображений и сравнения с истинными метками
def visualize_predictions(model, dataset, num_images=5):
    model.eval()
    fig, axes = plt.subplots(1, num_images, figsize=(12, 3))
    random_indices = random.sample(range(len(dataset)), num_images)

    for i, index in enumerate(random_indices):
        image, label = dataset[index]

        # Визуализация исходного изображения
        img = image.numpy().transpose((1, 2, 0)).squeeze()
        axes[i].imshow(img, cmap='gray')

        # Подготовка изображения для модели
        image = image.unsqueeze(0)

        with torch.no_grad():
            output = model(image)
            _, predicted = torch.max(output, 1)

        # Отображение истинного и предсказанного класса
        axes[i].set_title(f"True: {label}, Pred: {predicted.item()}")
        axes[i].axis('off')

    plt.show()

# Обучение модели
```

```
loss_history = train_model(model, train_loader, criterion, optimizer, epochs=13)

# Тестирование модели
test_accuracy = test_model(model, test_loader)

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

# Визуализация предсказаний
visualize_predictions(model, test_set, num_images=5)
```

```
Epoch [1/13], Loss: 0.5844
Epoch [2/13], Loss: 0.4106
Epoch [3/13], Loss: 0.3897
Epoch [4/13], Loss: 0.3854
Epoch [5/13], Loss: 0.3712
Epoch [6/13], Loss: 0.3681
Epoch [7/13], Loss: 0.3647
Epoch [8/13], Loss: 0.3670
Epoch [9/13], Loss: 0.3602
Epoch [10/13], Loss: 0.3632
Epoch [11/13], Loss: 0.3593
Epoch [12/13], Loss: 0.3618
Epoch [13/13], Loss: 0.3603
Test Accuracy: 87.74%
```






```
Predicted Class: 0
```

При сравнении результатов с лабораторной работой №1 было выяленно что результаты имеют худший результат, результат этой лабораторной работы отличается на 5%.