МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ИИ(з)

Выполнил
Д. Д. Крупич,
студент группы ИИ-24

Проверил
Андренко К.В,
Преподаватель-стажер кафедры ИИТ,
«___k _____2025 г.

Брест 2025

Цель: осуществлять обучение НС, сконструированных на базе предобученных архитектур НС

Общее 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;

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

3. Ознакомиться с state-of-the-art результатами для предлагаемых выборок (по материалам в сети Интернет). Сделать выводы о результатах обучения НС из п. 1 и 2;

4. Реализовать визуализацию работы предобученной СНС и кастомной (из ЛР 1). Визуализация осуществляется посредством выбора и подачи на сеть произвольного изображения (например, из сети Интернет) с отображением результата классификации;

5. Оформить отчет по выполненной работе, залить исходный код и отчет в соответствующий репозиторий на github.

Вариант:

7	Fashion-MNIST	Adam	ResNet34

Код программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
import torchvision
import torchvision.transforms as transforms
from torchvision.models import resnet34, ResNet34_Weights
import matplotlib.pyplot as plt
from tqdm import tqdm
import time

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Device: {device}')

train_transform = transforms.Compose([
```

```
    transforms.Grayscale(num_output_channels=3),
    transforms.Resize(224),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),
    transforms.Resize(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

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

batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=2)

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

print(f'Train size: {len(train_dataset)}')
print(f'Test size: {len(test_dataset)}')

model = resnet34(weights=ResNet34_Weights.IMAGENET1K_V1)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)
model = model.to(device)

def train_epoch(model, loader, criterion, optimizer, device):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in tqdm(loader, desc='Training', leave=False):
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
```

```
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    epoch_loss = running_loss / len(loader)
    epoch_acc = 100. * correct / total
    return epoch_loss, epoch_acc

def test_model(model, loader, criterion, device):
    model.eval()
    test_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for inputs, labels in tqdm(loader, desc='Testing', leave=False):
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            test_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    test_loss = test_loss / len(loader)
    test_acc = 100. * correct / total
    return test_loss, test_acc

num_epochs = 10
learning_rate = 0.001
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

train_losses = []
train_accs = []
test_losses = []
test_accs = []

print('\n=== Training ResNet34 ===')
start_time = time.time()
```

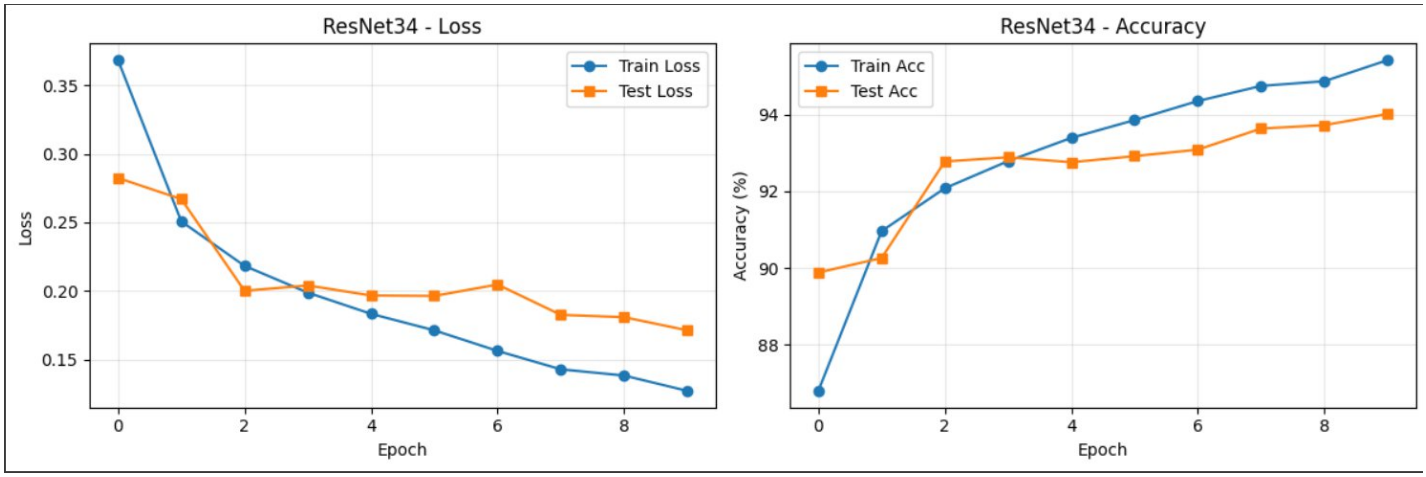
```
for epoch in range(num_epochs):
    print(f'\nEpoch {epoch+1}/{num_epochs}')

    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer, device)
    train_losses.append(train_loss)
    train_accs.append(train_acc)

    test_loss, test_acc = test_model(model, test_loader, criterion, device)
    test_losses.append(test_loss)
    test_accs.append(test_acc)

    print(f'Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%')
    print(f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}%')

training_time = time.time() - start_time
print(f'\nTraining time: {training_time:.2f} seconds')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

ax1.plot(train_losses, label='Train Loss', marker='o')
ax1.plot(test_losses, label='Test Loss', marker='s')
ax1.set_title('ResNet34 - Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.legend()
ax1.grid(True, alpha=0.3)

ax2.plot(train_accs, label='Train Acc', marker='o')
ax2.plot(test_accs, label='Test Acc', marker='s')
ax2.set_title('ResNet34 - Accuracy')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy (%)')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('resnet34_training.png', dpi=300, bbox_inches='tight')
plt.show()

print('\n' + '='*70)
print('RESULTS')
print('='*70)
print(f'Final Test Accuracy: {test_accs[-1]:.2f}%')
print(f'Training Time: {training_time:.2f}s')
```

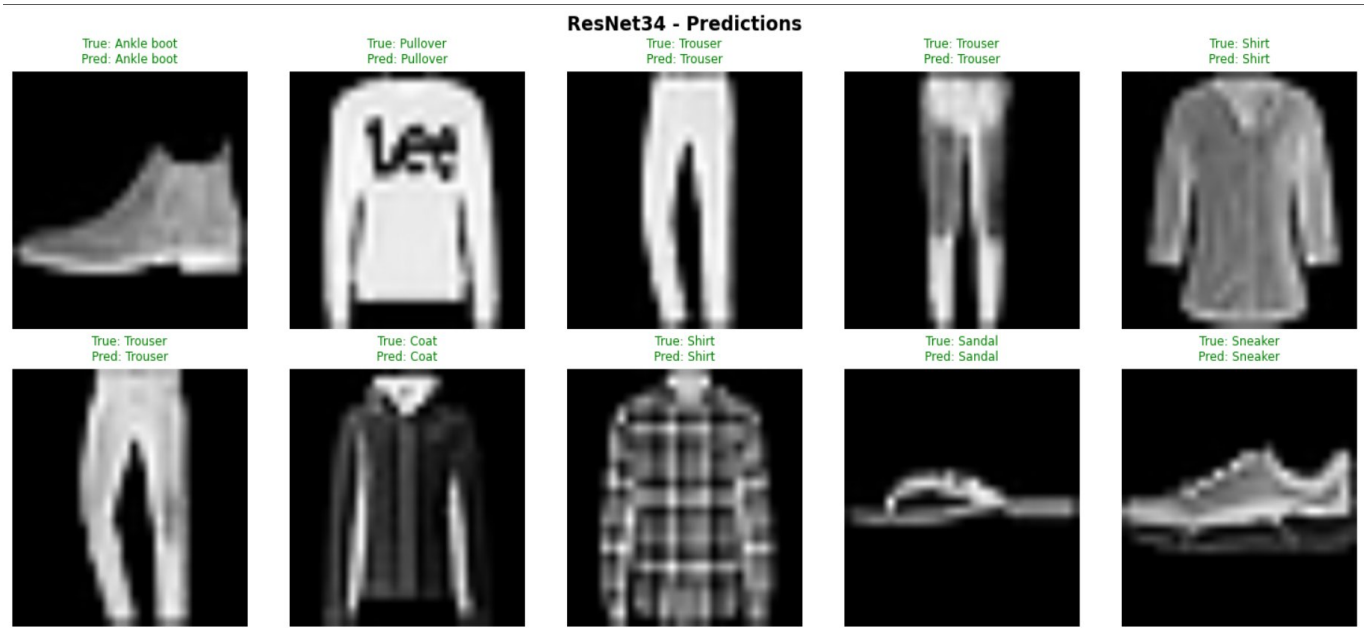
```
torch.save(model.state_dict(), 'resnet34_fashion_mnist.pth')
print('\nModel saved: resnet34_fashion_mnist.pth')

def visualize_predictions(model, loader, num_images=10):
    model.eval()

    fig, axes = plt.subplots(2, 5, figsize=(15, 6))
    axes = axes.ravel()

    dataiter = iter(loader)
    images, labels = next(dataiter)
    images, labels = images.to(device), labels.to(device)

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

    for idx in range(num_images):
        img = images[idx].cpu()
        mean = torch.tensor([0.485, 0.456, 0.406]).view(3, 1, 1)
        std = torch.tensor([0.229, 0.224, 0.225]).view(3, 1, 1)
        img = img * std + mean
        img = img[0].numpy()

        axes[idx].imshow(img, cmap='gray')
        axes[idx].axis('off')

        true_label = classes[labels[idx]]
        pred_label = classes[predicted[idx]]
        color = 'green' if labels[idx] == predicted[idx] else 'red'

        axes[idx].set_title(f'True: {true_label}\nPred: {pred_label}', color=color, fontsize=9)

    plt.tight_layout()
    return fig

fig = visualize_predictions(model, test_loader, 10)
fig.suptitle('ResNet34 - Predictions', fontsize=14, fontweight='bold', y=1.02)
plt.savefig('resnet34_predictions.png', dpi=300, bbox_inches='tight')
plt.show()

def evaluate_per_class(model, loader, device):
    model.eval()
    class_correct = [0] * 10
    class_total = [0] * 10
```

```
    with torch.no_grad():
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = outputs.max(1)

            for i in range(labels.size(0)):
                label = labels[i].item()
                class_correct[label] += (predicted[i] == labels[i]).item()
                class_total[label] += 1

    return class_correct, class_total

class_correct, class_total = evaluate_per_class(model, test_loader, device)

print('\n' + '='*70)
print('PER-CLASS ACCURACY')
print('='*70)
print(f'{"Class":<20} {"Accuracy":<15}')
print('-'*35)
for i, class_name in enumerate(classes):
    acc = 100 * class_correct[i] / class_total[i] if class_total[i] > 0 else 0
    print(f'{class_name:<20} {acc:>13.2f}%')

print('\nFiles saved:')
print('- resnet34_fashion_mnist.pth')
print('- resnet34_training.png')
print('- resnet34_predictions.png')
```

Вывод программы:

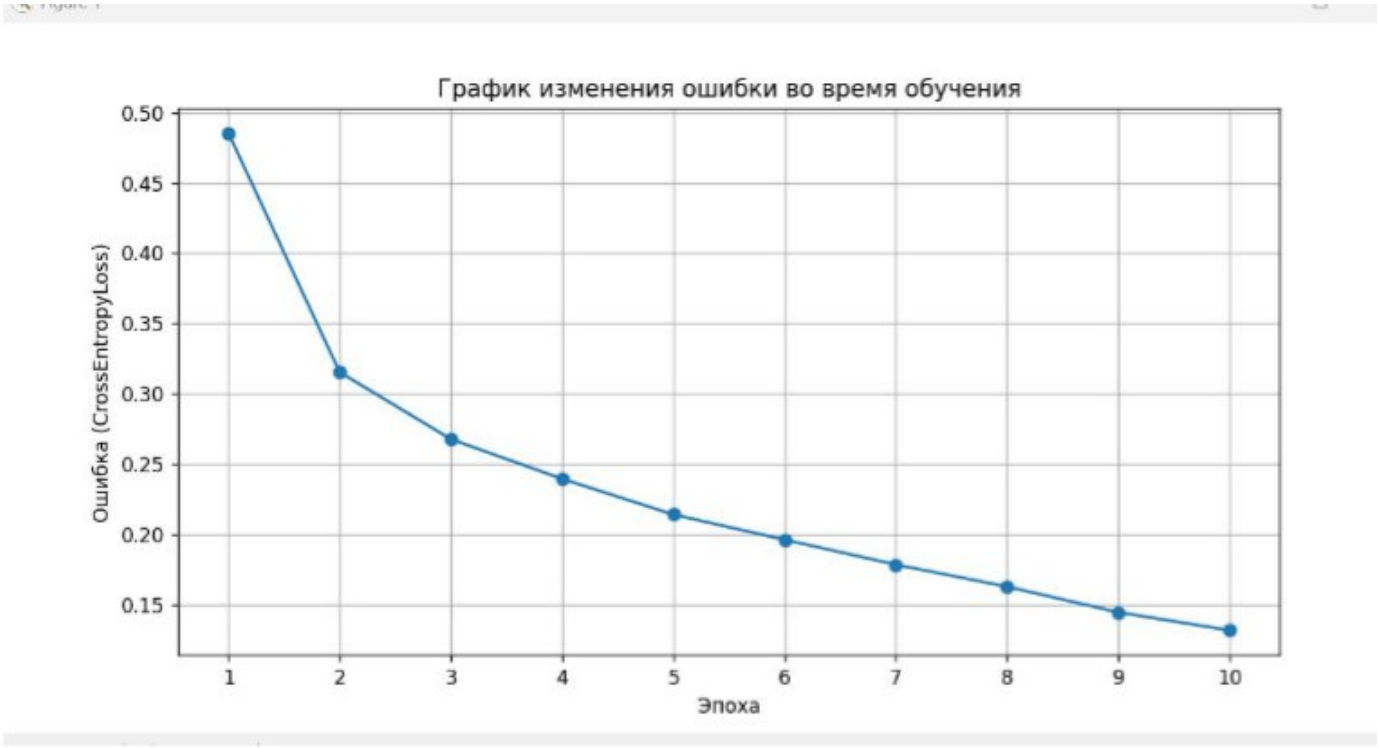
```
=== Training ResNet34 ===

Epoch 1/10
Train Loss: 0.3684, Train Acc: 86.79%
Test Loss: 0.2824, Test Acc: 89.88%

Epoch 2/10
Train Loss: 0.2507, Train Acc: 90.96%
Test Loss: 0.2670, Test Acc: 90.26%

Epoch 3/10
Train Loss: 0.2183, Train Acc: 92.08%
Test Loss: 0.2001, Test Acc: 92.78%

Epoch 4/10
Train Loss: 0.1987, Train Acc: 92.79%
Test Loss: 0.2040, Test Acc: 92.89%

Epoch 5/10
Train Loss: 0.1833, Train Acc: 93.40%
Test Loss: 0.1966, Test Acc: 92.76%

Epoch 6/10
Train Loss: 0.1712, Train Acc: 93.86%
Test Loss: 0.1963, Test Acc: 92.92%

Epoch 7/10
Train Loss: 0.1562, Train Acc: 94.36%
Test Loss: 0.2046, Test Acc: 93.09%

Epoch 8/10
Train Loss: 0.1428, Train Acc: 94.75%
Test Loss: 0.1825, Test Acc: 93.64%

Epoch 9/10
Train Loss: 0.1382, Train Acc: 94.88%
Test Loss: 0.1808, Test Acc: 93.73%

Epoch 10/10
Train Loss: 0.1270, Train Acc: 95.43%
Test Loss: 0.1712, Test Acc: 94.02%

Training time: 3230.03 seconds
```

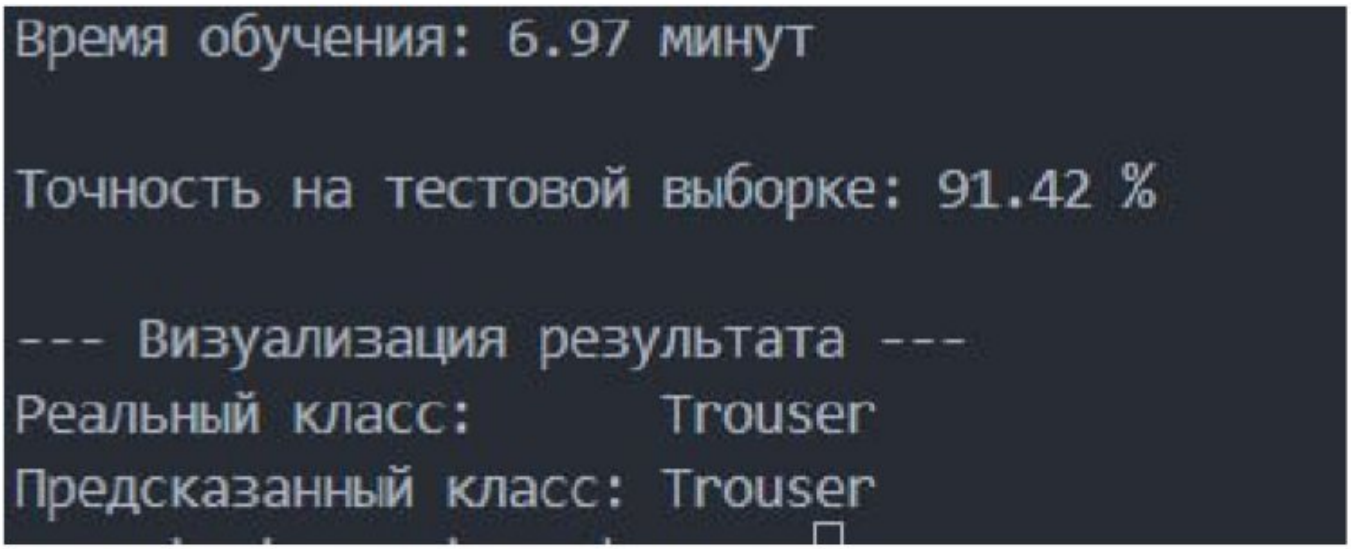





```
================================================================================
PER-CLASS ACCURACY
================================================================================
Class                Accuracy
--------------------------------------
T-shirt/top                90.50%
Trouser                    98.70%
Pullover                   94.20%
Dress                      95.50%
Coat                       92.80%
Sandal                     98.50%
Shirt                      76.20%
Sneaker                    98.70%
Bag                        99.40%
Ankle boot                 95.70%
```

Сравнение с 1 ЛР:





Сравнение моделей:

1. **Предобученная ResNet34 (ЛР2):** показала точность **94.02%** на тестовой выборке Fashion-MNIST за время обучения **53.8 минут** (3230 секунд, 10 эпох).
2. **Кастомная CNN (ЛР1):** достигла точности **91,42%** на той же выборке.

Основные выводы:

- Предобученная ResNet34 превзошла кастомную модель на **2.02%**, показав преимущество transfer learning и глубокой архитектуры.
- ResNet34 достигла результата, близкого к state-of-the-art для Fashion-MNIST, что подтверждает эффективность предобученных моделей.
- Transfer learning оказался эффективным даже для данных, сильно отличающихся от ImageNet (одежда vs. общие объекты), благодаря универсальности низкоуровневых признаков.

- Для задач с высокими требованиями к точности предобученные модели предпочтительнее кастомных архитектур.

SOTA-результат из статьи MDPI "State-of-the-Art Results with the Fashion-MNIST Dataset" (https://doi.org/10.3390/math12203174) — точность до 99.65% с моделью CNN-3-128, использующей 3 сверточных слоя, dropout, и аугментацию данных.

Вывод: Я осуществил обучение НС, сконструированных на базе предобученных архитектур НС