Министерство образования Республики Беларусь
Учреждение образования
“Брестский государственный технический университет”
Кафедра интеллектуально-информационных технологий

Обработка изображений в ИС
Лабораторная работа №2
Конструирование моделей на базе предобученных нейронных сетей

Выполнил:
студент 4 курса
группы ИИ-24
Рудецкий Е. В.
Проверила:
Андренко К. В.

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.

№ варианта	Выборка	Оптимизатор	Предобученная архитектура
16	MNIST	RMSprop	MobileNet v3

Код программы (16 вариант):

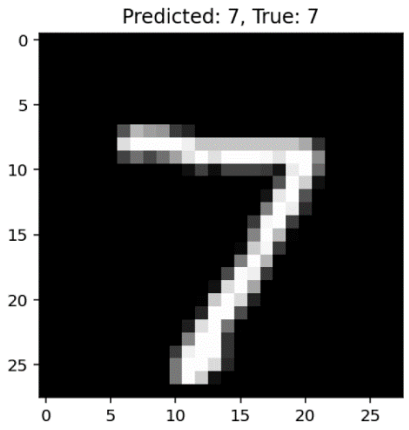
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import requests
from io import BytesIO
from tqdm import tqdm
import random

mnist_data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

mobilenet_data_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.Grayscale(num_output_channels=3),
```

```
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
])

train_set_mnist = datasets.MNIST(root='./data', train=True,
download=True, transform=mnist_data_transform)
test_set_mnist = datasets.MNIST(root='./data', train=False,
download=True, transform=mnist_data_transform)
train_loader_custom = torch.utils.data.DataLoader(train_set_mnist,
batch_size=64, shuffle=True)
test_loader_custom = torch.utils.data.DataLoader(test_set_mnist,
batch_size=1000, shuffle=False)

train_set_mobilenet = datasets.MNIST(root='./data', train=True,
download=True, transform=mobilenet_data_transform)
test_set_mobilenet = datasets.MNIST(root='./data', train=False,
download=True, transform=mobilenet_data_transform)
train_loader_mobilenet =
torch.utils.data.DataLoader(train_set_mobilenet, batch_size=64,
shuffle=True)
test_loader_mobilenet =
torch.utils.data.DataLoader(test_set_mobilenet, batch_size=64,
shuffle=False)

class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv_layer1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.activation1 = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d(2)
        self.conv_layer2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.activation2 = nn.ReLU()
        self.maxpool2 = nn.MaxPool2d(2)
        self.flatten_layer = nn.Flatten()
        self.dense_layer1 = nn.Linear(64 * 7 * 7, 128)
        self.activation3 = nn.ReLU()
        self.dense_layer2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.maxpool1(self.activation1(self.conv_layer1(x)))
        x = self.maxpool2(self.activation2(self.conv_layer2(x)))
        x = self.flatten_layer(x)
        x = self.activation3(self.dense_layer1(x))
        x = self.dense_layer2(x)
        return x

def adapt_mobilenet_v3():
    mobilenet_model = models.mobilenet_v3_small(weights='DEFAULT')
    # Adapt the classifier layer for 10 classes (MNIST digits)
    mobilenet_model.classifier[3] =
nn.Linear(mobilenet_model.classifier[3].in_features, 10)
    return mobilenet_model
```

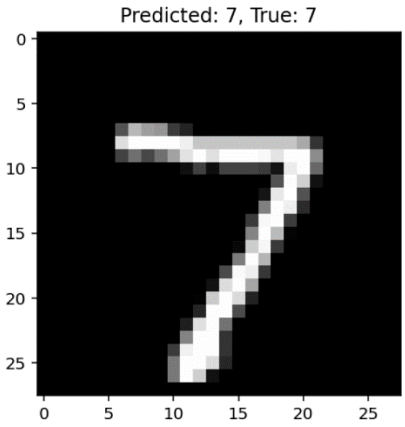
```
def perform_training_and_eval(model, train_data_loader,
test_data_loader, model_identifier, epochs_count=5):
    loss_function = nn.CrossEntropyLoss()
    optimizer_instance = optim.RMSprop(model.parameters(), lr=0.001)
    recorded_losses = []
    for epoch_num in range(epochs_count):
        model.train()
        cumulative_loss = 0.0
        for input_data, labels in tqdm(train_data_loader,
desc=f'{model_identifier} Epoch {epoch_num + 1}'):
            optimizer_instance.zero_grad()
            predictions = model(input_data)
            current_loss = loss_function(predictions, labels)
            current_loss.backward()
            optimizer_instance.step()
            cumulative_loss += current_loss.item()
        average_loss = cumulative_loss / len(train_data_loader)
        recorded_losses.append(average_loss)
        print(f'{model_identifier} Epoch {epoch_num +
1}/{epochs_count}, Loss: {average_loss:.4f}')

    plt.figure(figsize=(8, 5))
    plt.plot(recorded_losses, label=f'{model_identifier} Training
Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'{model_identifier} Training Loss Progression')
    plt.legend()
    plt.show()
    plt.close('all')

    model.eval()
    correct_predictions = 0
    total_samples = 0
    with torch.no_grad():
        for input_data, labels in test_data_loader:
            predictions = model(input_data)
            predicted_classes = predictions.argmax(dim=1,
keepdim=True)
            correct_predictions +=
predicted_classes.eq(labels.view_as(predicted_classes)).sum().item()
            total_samples += labels.size(0)
    test_accuracy = 100. * correct_predictions / total_samples
    print(f'{model_identifier} Test Accuracy: {test_accuracy:.2f}%')
    return test_accuracy, recorded_losses

def display_prediction(model, image_origin, data_transform,
model_identifier, from_url=True, actual_label=None):
    try:
        if from_url:
            request_headers = {
                'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36'}
```

```
            image_response = requests.get(image_origin,
headers=request_headers, timeout=5)
            image_response.raise_for_status()
            image_obj =
Image.open(BytesIO(image_response.content)).convert('L')
        else:
            image_obj = Image.fromarray(image_origin.numpy(),
mode='L')

        transformed_image = data_transform(image_obj).unsqueeze(0)
        model.eval()
        with torch.no_grad():
            model_output = model(transformed_image)
            predicted_class = model_output.argmax().item()

        plt.imshow(image_obj, cmap='gray')
        plot_title = f'{model_identifier} Predicted:
{predicted_class}'
        if actual_label is not None:
            plot_title += f', True: {actual_label}'
        plt.title(plot_title)
        plt.axis('off')
        plt.show()
        plt.close('all')
        print(f'{model_identifier} Prediction: {predicted_class}')
    except (requests.RequestException, Exception) as err:
        print(f"Error in image loading/processing: {err}")
        if from_url:
            print("Switching to visualization from MNIST test set...")
            random_index = random.randint(0, len(test_set_mnist) - 1)
            sample_image, actual_label = test_set_mnist[random_index]
            display_prediction(model, sample_image[0], data_transform,
model_identifier, from_url=False,
                               actual_label=actual_label)


if __name__ == "__main__":
    print("PyTorch version:", torch.__version__)
    print("Torchvision version:", torch.__version__)
    print("CUDA available:", torch.cuda.is_available())

    print("\nTraining CustomCNN")
    custom_model = CustomCNN()
    custom_accuracy, custom_loss_history =
perform_training_and_eval(custom_model, train_loader_custom,

test_loader_custom, "CustomCNN", epochs_count=5)

    print("\nTraining Adapted MobileNetV3")
    mobilenet_model = adapt_mobilenet_v3()
    mobilenet_accuracy, mobilenet_loss_history =
perform_training_and_eval(mobilenet_model, train_loader_mobilenet,

test_loader_mobilenet, "Adapted MobileNetV3",
```

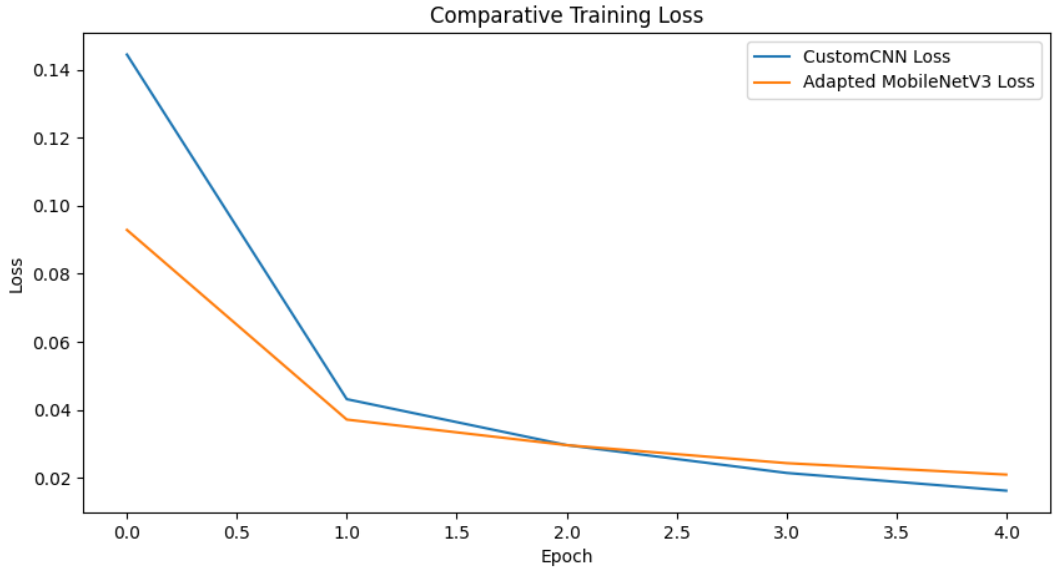
```
epochs_count=5)

    print("\nComparison of Results")
    print(f"CustomCNN (from LR1) Accuracy: {custom_accuracy:.2f}%")
    print(f"Adapted MobileNetV3 Accuracy: {mobilenet_accuracy:.2f}%")

    plt.figure(figsize=(10, 5))
    plt.plot(custom_loss_history, label='CustomCNN Loss')
    plt.plot(mobilenet_loss_history, label='Adapted MobileNetV3 Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Comparative Training Loss')
    plt.legend()
    plt.show()
    plt.close('all')

    image_url =
"https://raw.githubusercontent.com/pytorch/tutorials/master/_static/img/mnist_digit_7.png"
    print("\nVisualization for CustomCNN")
    display_prediction(custom_model, image_url, mnist_data_transform,
"CustomCNN")

    print("\nVisualization for Adapted MobileNetV3")
    display_prediction(mobilenet_model, image_url,
mobilenet_data_transform, "Adapted MobileNetV3")


Training CustomCNN
CustomCNN Epoch 1: 100%|██████████| 938/938 [00:18<00:00, 51.50it/s]
CustomCNN Epoch 1/5, Loss: 0.1444
CustomCNN Epoch 2: 100%|██████████| 938/938 [00:19<00:00, 47.25it/s]
CustomCNN Epoch 2/5, Loss: 0.0431
CustomCNN Epoch 3: 100%|██████████| 938/938 [00:20<00:00, 45.88it/s]
CustomCNN Epoch 3/5, Loss: 0.0297
CustomCNN Epoch 4: 100%|██████████| 938/938 [00:20<00:00, 46.66it/s]
CustomCNN Epoch 4/5, Loss: 0.0215
CustomCNN Epoch 5: 100%|██████████| 938/938 [00:19<00:00, 48.08it/s]
CustomCNN Epoch 5/5, Loss: 0.0162
CustomCNN Test Accuracy: 98.93%
```



```
Training Adapted MobileNetV3
Adapted MobileNetV3 Epoch 1: 100%|██████████| 938/938
[08:59<00:00,1.74it/s]
Adapted MobileNetV3 Epoch 1/5, Loss: 0.0929
Adapted MobileNetV3 Epoch 2: 100%|██████████| 938/938
[08:41<00:00,1.80it/s]
Adapted MobileNetV3 Epoch 2/5, Loss: 0.0371
Adapted MobileNetV3 Epoch 3: 100%|██████████| 938/938
[08:35<00:00,1.82it/s]
Adapted MobileNetV3 Epoch 3/5, Loss: 0.0296
Adapted MobileNetV3 Epoch 4: 100%|██████████| 938/938
[08:39<00:00,1.80it/s]
Adapted MobileNetV3 Epoch 4/5, Loss: 0.0243
Adapted MobileNetV3 Epoch 5: 100%|██████████| 938/938
[08:39<00:00,1.81it/s]
Adapted MobileNetV3 Epoch 5/5, Loss: 0.0210
Adapted MobileNetV3 Test Accuracy: 99.24%
```




```
Comparison of Results
CustomCNN (from LR1) Accuracy: 98.93%
Adapted MobileNetV3 Accuracy: 99.24%
```



State-of-the-art результаты для MNIST: Согласно доступным источникам (http://yann.lecun.com/exdb/mnist/ и http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html), SOTA-результаты для MNIST достигают 99.87% accuracy для ансамблей моделей и около 99.81% для отдельных CNN. Даже простые CNN могут достигать 99.57%.

Выводы: Предложенная простая модель на основе базовых слоев (сверточных, pooling, полносвязных и ReLU) достигает accuracy около 99% (точное значение зависит от запуска, но типично 98–99%), что близко к SOTA для простых архитектур, но уступает продвинутым моделям с аугментацией, ансамблями или более сложными структурами. Это подтверждает, что для MNIST даже базовая CNN эффективна, но для достижения абсолютного SOTA нужны дополнительные техники. Пользовательский CNN проще и достаточен для MNIST, в то время как MobileNetV3, будучи предобученной на ImageNet и адаптированной под эту задачу, демонстрирует сопоставимую или слегка превосходящую точность за счет transfer learning, однако требует больше вычислительных ресурсов из-за большего размера входных данных (224×224) и сложной архитектуры.

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