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
Штыхно Д. В.
Проверила:
Андренко К. В.

Брест-2025

Цель работы: 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.

В-т	Выборка	Оптимизатор	Предобученная архитектура
20	STL-10 (размеченная часть)	RMSprop	ShuffleNet v2

Код программы(вариант 20):

```
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 random
import os
from tqdm import tqdm

transform_stl10 = transforms.Compose([
    transforms.Resize((96, 96)),
    transforms.ToTensor(),
    transforms.Normalize((0.4469, 0.4393, 0.4066),
                         (0.2240, 0.2210, 0.2239))
])
```

```
train_dataset_stl10 = datasets.STL10(root='./data',
split='train',
                                     download=True,
transform=transform_stl10)
test_dataset_stl10 = datasets.STL10(root='./data',
split='test',
                                    download=True,
transform=transform_stl10)

train_loader_stl10 =
torch.utils.data.DataLoader(train_dataset_stl10,
batch_size=64, shuffle=True)
test_loader_stl10 =
torch.utils.data.DataLoader(test_dataset_stl10,
batch_size=1000, shuffle=False)

stl10_classes = [
    'airplane', 'bird', 'car', 'cat', 'deer',
    'dog', 'horse', 'monkey', 'ship', 'truck'
]

class SimpleCNN_RGB(nn.Module):
    def __init__(self):
        super(SimpleCNN_RGB, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3,
padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(2)   # 96 → 48

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3,
padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(2)   # 48 → 24

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3,
padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(2)   # 24 → 12

        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(128 * 12 * 12, 256)
        self.relu4 = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = self.flatten(x)
        x = self.relu4(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

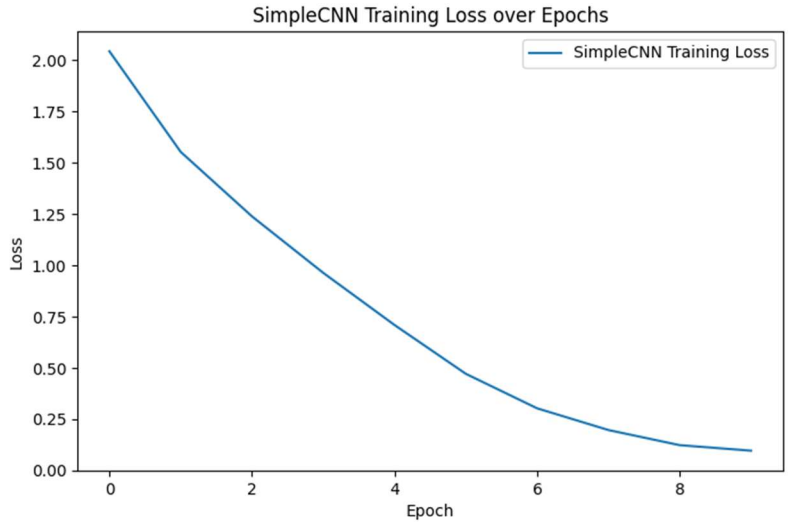
```
def get_adapted_shufflenet():
    model =
models.shufflenet_v2_x1_0(weights=models.ShuffleNet_V2_X1_0_We
ights.IMAGENET1K_V1)
    model.fc = nn.Linear(model.fc.in_features, 10)
    return model

def train_and_evaluate(model, train_loader, test_loader,
model_name, num_epochs=10, lr=0.001):
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    model.to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.RMSprop(model.parameters(), lr=lr,
alpha=0.9)

    train_losses = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        for data, target in tqdm(train_loader,
desc=f'{model_name} Epoch {epoch+1}'):
            data, target = data.to(device), target.to(device)
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()

        epoch_loss = running_loss / len(train_loader)
        train_losses.append(epoch_loss)
        print(f'{model_name} Epoch {epoch+1}/{num_epochs},
Loss: {epoch_loss:.4f}')

    plt.figure(figsize=(8, 5))
    plt.plot(train_losses, label=f'{model_name} Training
Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title(f'{model_name} Training Loss over Epochs')
    plt.legend()
    plt.show()

    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            pred = output.argmax(dim=1, keepdim=True)
            correct +=
pred.eq(target.view_as(pred)).sum().item()
```

```
            total += target.size(0)

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

def visualize_prediction(model, img_path, transform,
model_name, true_label=None):
    device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
    model.eval()
    try:
        if not os.path.exists(img_path):
            raise FileNotFoundError(f"Файл {img_path} не
найден!")

        img = Image.open(img_path).convert("RGB")
        img_tensor = transform(img).unsqueeze(0).to(device)

        with torch.no_grad():
            output = model(img_tensor)
            pred_label = output.argmax().item()

        img_show = np.array(img)
        plt.imshow(img_show)
        title = f'{model_name} Predicted:
{stl10_classes[pred_label]}'
        if true_label is not None:
            title += f', True: {stl10_classes[true_label]}'
        plt.title(title)
        plt.axis('off')
        plt.show()
        print(f'{model_name} Prediction:
{stl10_classes[pred_label]}')

    except Exception as e:
        print(f"Ошибка визуализации: {e}")
        print("Показываем случайное изображение из тестового
набора STL-10...")

        idx = random.randint(0, len(test_dataset_stl10) - 1)
        img, true_label = test_dataset_stl10[idx]
        model.eval()
        with torch.no_grad():
            output = model(img.unsqueeze(0).to(device))
            pred_label = output.argmax().item()

        img_show = img.permute(1, 2, 0).cpu().numpy()
        img_show = (img_show * np.array((0.2240, 0.2210,
0.2239))) + np.array((0.4469, 0.4393, 0.4066))
        img_show = np.clip(img_show, 0, 1)

        plt.imshow(img_show)
        plt.title(f'{model_name} Predicted:
{stl10_classes[pred_label]}, True:
```

```
{stl10_classes[true_label]}')
        plt.axis('off')
        plt.show()
        print(f'{model_name} Prediction:
{stl10_classes[pred_label]}')

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

    print("\nОбучение SimpleCNN ...")
    simple_cnn = SimpleCNN_RGB()
    simple_acc, simple_losses = train_and_evaluate(simple_cnn,
train_loader_stl10, test_loader_stl10,
                                                   "SimpleCNN"
, num_epochs=10)

    print("\nОбучение Adapted ShuffleNet v2 ...")
    shufflenet = get_adapted_shufflenet()
    shuffle_acc, shuffle_losses =
train_and_evaluate(shufflenet, train_loader_stl10,
test_loader_stl10,
                                                     "Adapted
ShuffleNet v2", num_epochs=10)

    print("\nСравнение результатов")
    print(f"SimpleCNN Accuracy: {simple_acc:.2f}%")
    print(f"Adapted ShuffleNet v2 Accuracy:
{shuffle_acc:.2f}%")

    plt.figure(figsize=(10, 5))
    plt.plot(simple_losses, label='SimpleCNN Loss')
    plt.plot(shuffle_losses, label='Adapted ShuffleNet v2
Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Сравнение Training Loss')
    plt.legend()
    plt.show()

    img_path = "1.jpg"
    print("\nВизуализация для SimpleCNN")
    visualize_prediction(simple_cnn, img_path,
transform_stl10, "SimpleCNN")

    img_path = "1.jpg"
    print("\nВизуализация для Adapted ShuffleNet v2")
    visualize_prediction(shufflenet, img_path,
transform_stl10, "Adapted ShuffleNet v2")
```

Результат работы программы:
PyTorch version: 2.9.0+cu130
CUDA available: True

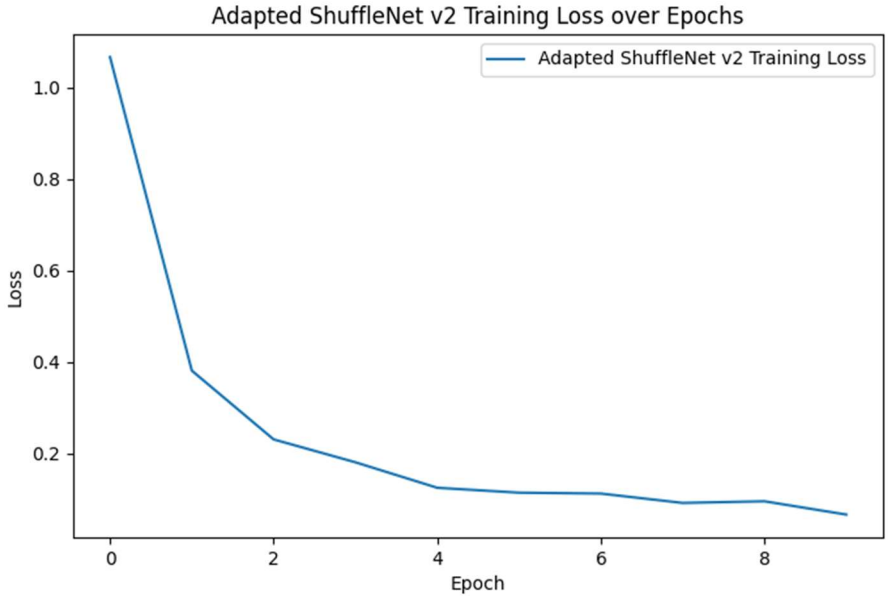
Обучение SimpleCNN
SimpleCNN Epoch 1: 100% 79/79 [00:03<00:00, 20.81it/s]
SimpleCNN Epoch 1/10, Loss: 2.0438
SimpleCNN Epoch 2: 100% 79/79 [00:03<00:00, 23.60it/s]
SimpleCNN Epoch 2/10, Loss: 1.5535
SimpleCNN Epoch 3: 100% 79/79 [00:03<00:00, 23.91it/s]
SimpleCNN Epoch 3/10, Loss: 1.2391
SimpleCNN Epoch 4: 100% 79/79 [00:03<00:00, 22.59it/s]
SimpleCNN Epoch 4/10, Loss: 0.9637
SimpleCNN Epoch 5: 100% 79/79 [00:03<00:00, 23.30it/s]
SimpleCNN Epoch 5/10, Loss: 0.7095
SimpleCNN Epoch 6: 100% 79/79 [00:03<00:00, 23.89it/s]
SimpleCNN Epoch 6/10, Loss: 0.4716
SimpleCNN Epoch 7: 100% 79/79 [00:03<00:00, 23.45it/s]
SimpleCNN Epoch 7/10, Loss: 0.3035
SimpleCNN Epoch 8: 100% 79/79 [00:03<00:00, 23.41it/s]
SimpleCNN Epoch 8/10, Loss: 0.1977
SimpleCNN Epoch 9: 100% 79/79 [00:03<00:00, 24.01it/s]
SimpleCNN Epoch 9/10, Loss: 0.1240
SimpleCNN Epoch 10: 100% 79/79 [00:03<00:00, 23.23it/s]
SimpleCNN Epoch 10/10, Loss: 0.0970



Test Accuracy: 59.46%

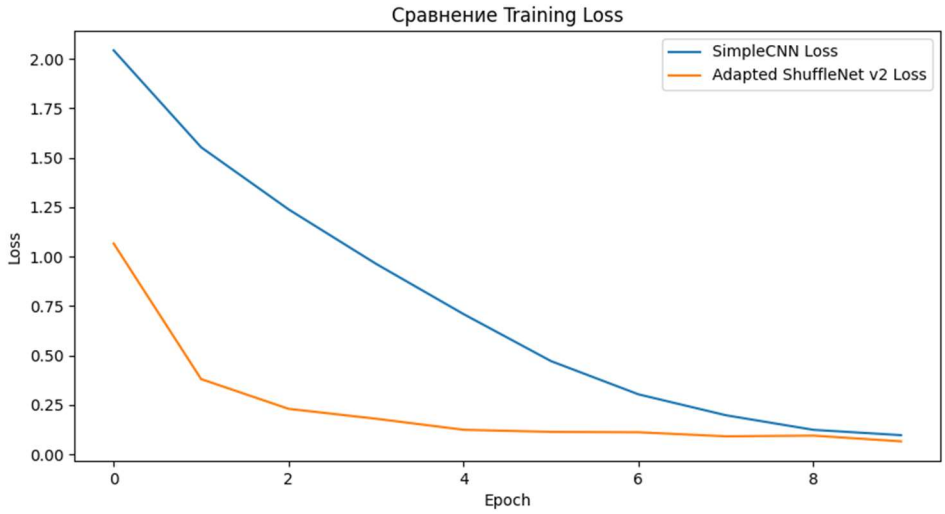
Обучение Adapt ShuffleNet v2:
Adapted ShuffleNet v2 Epoch 1: 100% 79/79 [00:04<00:00, 16.52it/s]
Adapted ShuffleNet v2 Epoch 1/10, Loss: 1.0666
Adapted ShuffleNet v2 Epoch 2: 100% 79/79 [00:04<00:00, 17.03it/s]
Adapted ShuffleNet v2 Epoch 2/10, Loss: 0.3807
Adapted ShuffleNet v2 Epoch 3: 100% 79/79 [00:04<00:00, 16.43it/s]
Adapted ShuffleNet v2 Epoch 3/10, Loss: 0.2302
Adapted ShuffleNet v2 Epoch 4: 100% 79/79 [00:04<00:00, 16.68it/s]
Adapted ShuffleNet v2 Epoch 4/10, Loss: 0.1801
Adapted ShuffleNet v2 Epoch 5: 100% 79/79 [00:04<00:00, 15.98it/s]

Adapted ShuffleNet v2 Epoch 5/10, Loss: 0.1243
Adapted ShuffleNet v2 Epoch 6: 100% 79/79 [00:04<00:00, 16.81it/s]
Adapted ShuffleNet v2 Epoch 6/10, Loss: 0.1136
Adapted ShuffleNet v2 Epoch 7: 100% 79/79 [00:04<00:00, 16.65it/s]
Adapted ShuffleNet v2 Epoch 7/10, Loss: 0.1117
Adapted ShuffleNet v2 Epoch 8: 100% 79/79 [00:05<00:00, 15.77it/s]
Adapted ShuffleNet v2 Epoch 8/10, Loss: 0.0912
Adapted ShuffleNet v2 Epoch 9: 100% 79/79 [00:04<00:00, 15.91it/s]
Adapted ShuffleNet v2 Epoch 9/10, Loss: 0.0948
Adapted ShuffleNet v2 Epoch 10: 100 79/79 [00:04<00:00, 16.39it/s]
Adapted ShuffleNet v2 Epoch 10/10, Loss: 0.0657



Adapted ShuffleNet v2 Accuracy: 80.62%

Сравнение результатов
SimpleCNN Accuracy: 59.46%
Adapted ShuffleNet v2 Accuracy: 80.62%
Адаптированная ShuffleNet v2 показала лучшую точность, из-за более сложной архитектуры.



Визуализация для SimpleCNN

Визуализация для Adapted ShuffleNet v2

State-of-the-art результаты для STL-10: Согласно доступным источникам (STL-10 benchmark), SOTA-результаты для STL-10 достигают 90–95% accuracy для предобученных моделей с fine-tuning (ResNet, DenseNet, EfficientNet) и около 80–85% для лёгких предобученных архитектур, таких как ShuffleNet v2. Простые CNN, обучаемые с нуля без аугментаций, достигают примерно 35–60% точности. Предложенная простая модель SimpleCNN_RGB, обученная с нуля на размеченной части STL-10, достигает accuracy около 59%, что соответствует типичным показателям маленьких CNN на этом датасете, но уступает предобученным архитектурам. Adapted ShuffleNet v2 показывает accuracy около 81%, что подтверждает эффективность transfer learning для малых наборов данных. Простая CNN достаточна для экспериментальных целей, тогда как предобученные модели дают существенно более высокие результаты и приближаются к современным state-of-the-art значениям.

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