Министерство науки и высшего образования Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого (СПбПУ). Институт компьютерных наук и Кибербезопасности

**КУРСОВАЯ РАБОТА**

Перенос стилей между изображениями

По дисциплине «Архитектура ЭВМ. Часть 2»

Выполнил студент в5130904/30022 Лютов А.В.

Руководитель

проф. д.т.н. Молодяков С.А.

Санкт-Петербург

2024

Оглавление

[Введение 3](#_Toc198153709)

[Алгоритм 4](#_Toc198153710)

[Блок-схема 5](#_Toc198153711)

[Описание использованных функций 6](#_Toc198153712)

[Текст программы 7](#_Toc198153713)

[Скриншоты 10](#_Toc198153714)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 11](#_Toc198153715)

# Введение

Современные методы обработки изображений с использованием глубокого обучения открывают широкие возможности для художественной трансформации визуального контента. Одним из таких методов является нейронная стилизация изображений (Neural Style Transfer), которая позволяет переносить стиль одного изображения (например, картины художника) на содержание другого изображения (например, фотографии). Этот процесс осуществляется с помощью сверточных нейронных сетей (CNN), которые способны извлекать и анализировать как высокоуровневые признаки контента, так и характерные текстуры и цветовые шаблоны, соответствующие стилю.

Данная работа реализует метод нейронной стилизации на основе предобученной модели VGG19, обученной на ImageNet, с использованием библиотеки PyTorch. Программа объединяет содержание одного изображения с художественным стилем другого, генерируя новый визуальный результат, сохраняющий структуру оригинального изображения и визуальные особенности стиля.

Целью данной работы является реализация и исследование алгоритма нейронной стилизации изображений с применением предобученной сверточной нейронной сети VGG19. В ходе выполнения ставятся следующие задачи:

* Загрузить и подготовить входные изображения: изображение содержимого и изображение стиля.
* Выделить признаки содержимого и стиля с помощью сверточной нейронной сети.
* Реализовать функцию потерь, объединяющую потери по содержанию и стилю.
* Выполнить оптимизацию изображения с учетом выбранной функции потерь.
* Получить и сохранить результат стилизации изображения.
* Визуализировать и проанализировать полученное стилизованное изображение.

В результате будет получено изображение, визуально сочетающее содержание одной сцены и художественные особенности другой, что демонстрирует возможности использования нейросетей в области цифрового искусства.

## Алгоритм

1. **Инициализация параметров**  
   Устанавливаются параметры: размер изображения, количество шагов, веса потерь, выбор оптимизатора и скорость обучения.
2. **Определение устройства**  
   Определяется использование GPU или CPU.
3. **Загрузка предобученной модели VGG19**  
   Загружается модель VGG19, обученная на ImageNet. Используется только часть features.
4. **Определение параметров нормализации**  
   Устанавливаются mean и std для нормализации изображений.
5. **Загрузка и нормализация изображений**  
   Загружаются и масштабируются контент- и стиль-изображения, нормализуются.
6. **Подготовка изображения для генерации (target)**  
   Клонируется изображение контента, включается requires\_grad.
7. **Выбор оптимизатора**  
   В зависимости от флага USE\_ADAM используется Adam или LBFGS.
8. **Определение слоёв признаков**  
   Указываются слои для потерь контента и стиля.
9. **Извлечение признаков с помощью VGG19**  
   Определяется функция для получения признаков с указанных слоёв.
10. **Вычисление грам-матриц для оценки стиля**  
    Стиль сравнивается через грам-матрицы.
11. **Определение функции потерь (closure)**  
    Функция возвращает общее значение потерь и считает градиенты.
12. **Запуск оптимизации**  
    Запускается обучение с выбранным оптимизатором на указанное количество шагов.
13. **Обратная нормализация и сохранение результата**  
    Результат денормализуется, сохраняется как output.jpg.
14. **Визуализация результата**  
    Отображается стилизованное изображение с помощью matplotlib.

## Блок-схема

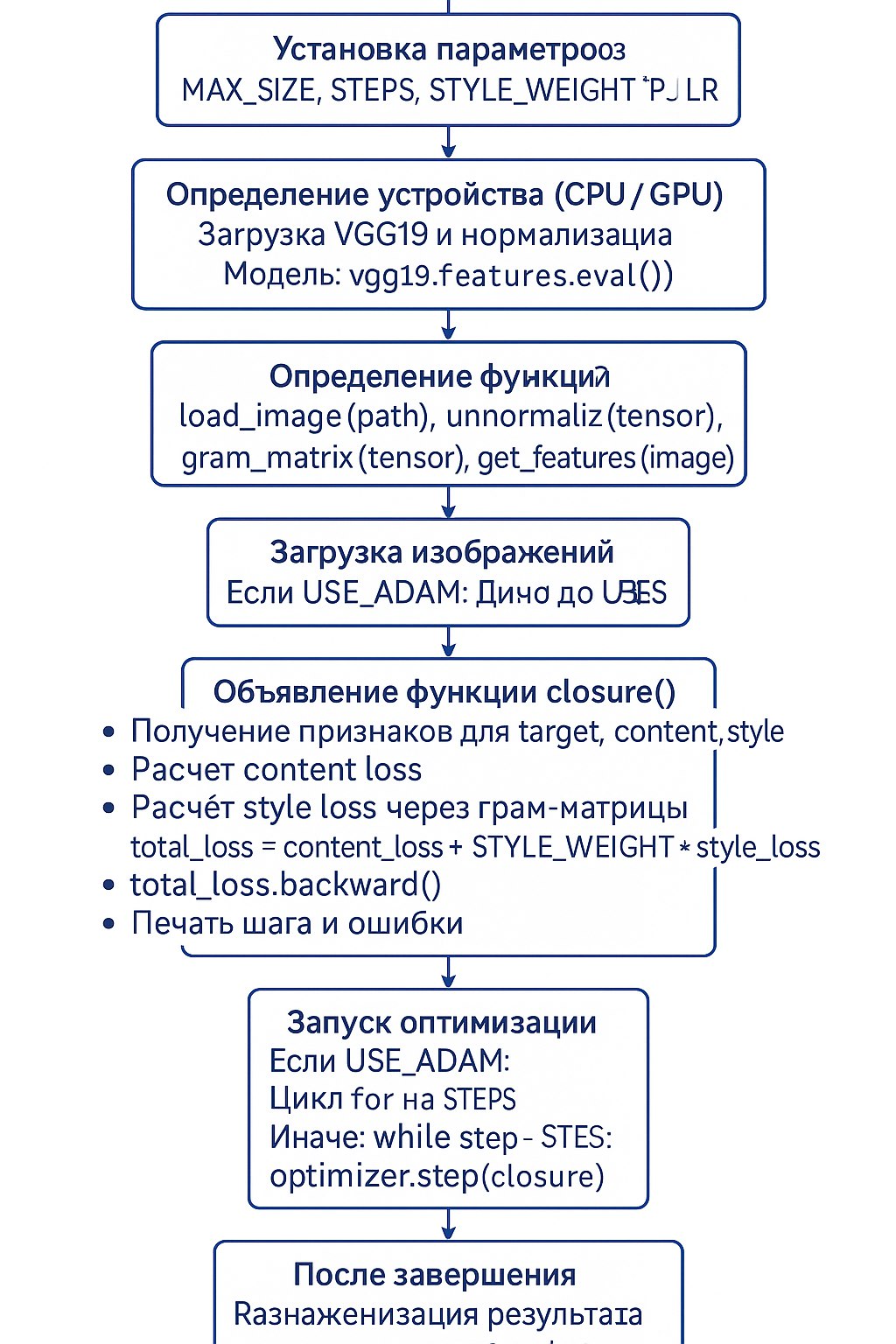


Рис. 1 блок схема работы программы

## Описание использованных функций

| **Функция / Метод** | **Назначение** |
| --- | --- |
| torch.device() | Определяет устройство (CPU или CUDA) |
| models.vgg19(...).features | Загружает сверточную часть VGG19 |
| transforms.Compose() | Последовательность преобразований изображения |
| Image.open() | Открытие изображения через PIL |
| tensor.unsqueeze(0) | Добавляет батч размерность |
| tensor.to(device) | Перемещает тензор на GPU/CPU |
| gram\_matrix() | Вычисление грам-матрицы для оценки стиля |
| torch.mm() | Матричное перемножение |
| optimizer.zero\_grad() | Обнуляет градиенты |
| loss.backward() | Вычисляет градиенты |
| optimizer.step() | Обновляет параметры |
| transforms.ToPILImage() | Преобразует тензор обратно в PIL изображение |
| plt.imshow() | Отображает изображение |

## Текст программы

import torch

import torch.optim as optim

from torchvision import transforms, models

from PIL import Image

import matplotlib.pyplot as plt

# ========== ПАРАМЕТРЫ ==========

MAX\_SIZE = 256

STEPS = 300

STYLE\_WEIGHT = 1e6

USE\_ADAM = False

LR = 0.01

# ========== УСТРОЙСТВО ==========

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

print(f"Используется устройство: {device}")

# ========== МОДЕЛЬ ==========

cnn = models.vgg19(weights=models.VGG19\_Weights.IMAGENET1K\_V1).features.to(device).eval()

# ========== НОРМАЛИЗАЦИЯ ==========

mean = torch.tensor([0.485, 0.456, 0.406]).to(device)

std = torch.tensor([0.229, 0.224, 0.225]).to(device)

# ========== ЗАГРУЗКА ИЗОБРАЖЕНИЯ ==========

def load\_image(path, max\_size=MAX\_SIZE):

    image = Image.open(path).convert('RGB')

    transform = transforms.Compose([

        transforms.Resize(max\_size),

        transforms.ToTensor(),

        transforms.Normalize(mean.tolist(), std.tolist())

    ])

    return transform(image).unsqueeze(0).to(device)

# ========== РАЗНОРМАЛИЗАЦИЯ ==========

def unnormalize(tensor):

    mean\_ = mean.view(3, 1, 1).to(tensor.device)

    std\_ = std.view(3, 1, 1).to(tensor.device)

    return tensor \* std\_ + mean\_

# ========== ГРАМ-МАТРИЦА ==========

def gram\_matrix(tensor):

    b, c, h, w = tensor.size()

    tensor = tensor.view(c, h \* w)

    return torch.mm(tensor, tensor.t()) / (c \* h \* w)

# ========== СЛОИ ==========

content\_layer = '21'  # conv4\_2

style\_layers = ['0', '5', '10', '19', '28']  # conv1\_1, conv2\_1, ...

# ========== ФУНКЦИЯ ИЗВЛЕЧЕНИЯ ПРИЗНАКОВ ==========

def get\_features(image):

    features = {}

    x = image

    for name, layer in cnn.named\_children():

        x = layer(x)

        if name in style\_layers + [content\_layer]:

            features[name] = x

    return features

# ========== ЗАГРУЗКА ИЗОБРАЖЕНИЙ ==========

content\_img = load\_image("content.jpg")

style\_img = load\_image("style.jpg")

target\_img = content\_img.clone().requires\_grad\_(True)

# ========== ОПТИМИЗАТОР ==========

optimizer = optim.Adam([target\_img], lr=LR) if USE\_ADAM else optim.LBFGS([target\_img])

# ========== ОБУЧЕНИЕ ==========

print("Стилизация началась...")

step = [0]  # обёртка для мутабельности

def closure():

    optimizer.zero\_grad()

    target\_features = get\_features(target\_img)

    content\_features = get\_features(content\_img)

    style\_features = get\_features(style\_img)

    # Content loss

    content\_loss = torch.mean((target\_features[content\_layer] - content\_features[content\_layer]) \*\* 2)

    # Style loss

    style\_loss = 0

    for layer in style\_layers:

        gram\_t = gram\_matrix(target\_features[layer])

        gram\_s = gram\_matrix(style\_features[layer])

        style\_loss += torch.mean((gram\_t - gram\_s) \*\* 2)

    total\_loss = content\_loss + STYLE\_WEIGHT \* style\_loss

    total\_loss.backward()

    if step[0] % 50 == 0:

        print(f"Step {step[0]} / {STEPS} | Loss: {total\_loss.item():.2f}")

    step[0] += 1

    return total\_loss

if USE\_ADAM:

    for \_ in range(STEPS):

        closure()

        optimizer.step()

else:

    while step[0] <= STEPS:

        optimizer.step(closure)

# ========== СОХРАНЕНИЕ ==========

print("Стилизация завершена.")

final\_img = unnormalize(target\_img.squeeze(0).detach()).clamp(0, 1).cpu()

output = transforms.ToPILImage()(final\_img)

output.save("output.jpg")

print("Результат сохранен как output.jpg")

# ========== ВИЗУАЛИЗАЦИЯ ==========

plt.imshow(output)

plt.axis("off")

plt.title("Стилизованное изображение")

plt.show()

## Скриншоты



Рис. 2. Исходное фото №1



Рис. 3. Исходное фото №2



Рис. 4. Результат

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. TorchVision documentation — <https://pytorch.org/vision/stable/index.html>
2. Официальная документация Python — <https://docs.python.org/>
3. Neural Style Transfer — PyTorch Tutorials, https://pytorch.org/tutorials/advanced/neural\_style\_tutorial.html
4. VGG19 — ImageNet Classification with Deep Convolutional Neural Networks, <https://arxiv.org/abs/1409.1556>
5. NumPy — <https://numpy.org>
6. Matplotlib — https://matplotlib.org/stable/index.html
7. Pillow (PIL Fork) Documentation — <https://pillow.readthedocs.io/en/stable/>