Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Лабораторная работа №2
по дисциплине
"Математический анализ и основы вычислений"

Семестр II

Выполнили:
студенты
Бровкин Аким Алексеевич
гр. J3110
ИСУ 465282

Воробьев Андрей Павлович
гр. J3110
ИСУ 465440

Шакина Анна Сергеевна
гр. J3110
ИСУ 396675

Отчёт сдан:
2.06.2025

Санкт-Петербург
2025

1 Введение

Целью данной лабораторной работы является изучение и реализация методов обработки изображений, таких как работа с каналами изображения, свертка, фильтрация и преобразование Фурье. Задачи работы включают:

- Загрузку и отображение изображения;
- Разделение изображения на каналы и их визуализацию;
- Применение свертки с различными ядрами;
- Использование пороговой и медианной фильтрации;
- Применение гауссовского и боксового размытия;
- Выделение краев с помощью фильтра Собеля;
- Изменение размера изображения с помощью DCT;
- Применение быстрого преобразования Фурье (FFT) для анализа и фильтрации изображения.

2 Теоретическая часть

Обработка изображений включает в себя различные методы для улучшения, анализа и преобразования изображений. В данной работе рассматриваются следующие ключевые концепции:

2.1 Каналы изображения

Изображения часто представлены в виде многоканальных данных. Например, RGB-изображения состоят из красного, зеленого и синего каналов. Анализ отдельных каналов позволяет изучить вклад каждого цвета в общее изображение.

2.2 Свёртка

Свёртка — это операция, применяемая к изображению с использованием ядра (фильтра) для выполнения задач, таких как размытие или выделение краев. Свёртка вычисляется путем скольжения ядра по изображению и вычисления взвешенной суммы пикселей.

2.3 Фильтрация

Фильтрация используется для удаления шума, сглаживания или выделения особенностей изображения. В работе применяются пороговая фильтрация, медианная фильтрация, гауссовское размытие и боксовое размытие.

2.4 Ряды Фурье и аппроксимация изображений

Ряды Фурье можно использовать для аппроксимации и обработки изображений, например, для увеличения их разрешения с сохранением гладкости линий. Для этого изображение представляется как сеточная функция $f(x, y)$, где каждая точка имеет яркость (например, от 0 до 255). Эта функция аппроксимируется частичными суммами двойного ряда Фурье по косинусам:

$$S_{mn}(x, y) = \sum_{j=-m}^{j=m} \sum_{k=-n}^{k=n} a_{jk} \cos \frac{\pi j x}{l_1} \cos \frac{\pi k y}{l_2}, \quad (1.1)$$

где l_1 и l_2 — ширина и высота рисунка, а коэффициенты a_{jk} находятся по формуле:

$$a_{jk} = \frac{1}{l_1 l_2} \int_0^{l_1} \int_0^{l_2} f(x, y) \cos \frac{\pi j x}{l_1} \cos \frac{\pi k y}{l_2} dy dx. \quad (1.2)$$

Числа m и n определяют пределы суммирования и, соответственно, количество слагаемых в сумме, влияя на качество аппроксимации. Однако, при слишком большом числе слагаемых качество может ухудшаться из-за накопления погрешностей округления.

2.5 Эффект Гиббса

При аппроксимации функций с разрывами (например, резкие переходы цвета в изображениях) с помощью конечного числа членов ряда Фурье возникает так называемый **эффект Гиббса**. Он проявляется в виде характерных всплесков или осцилляций в окрестности точек разрыва. Этот эффект вызван тем, что при вычислении частичной суммы ряда Фурье пренебрегаются слагаемые с более высокой частотой. В изображениях это может выглядеть как периодический «шум» или артефакты, особенно заметные в областях с высокой контрастностью или на границах объектов. Наличие резких границ в исходном изображении, таких как черная рамка, может усиливать этот эффект.

2.6 Дискретное косинусное преобразование (DCT) и JPEG

Методы, родственные разложению в ряд Фурье, такие как **дискретное косинусное преобразование (DCT)**, активно используются в стандартах сжатия изображений, например, JPEG. В JPEG DCT обычно применяется к блокам пикселей (например, 8×8). При таком подходе эффект Гиббса также может проявляться в виде волнообразных искажений вблизи резких цветовых переходов, особенно при высокой степени сжатия. Кодирование поблочно помогает локализовать эти искажения внутри блоков. Знание коэффициентов DCT позволяет восстанавливать изображение, в том числе с изменением разрешения.

2.7 Быстрое преобразование Фурье (FFT)

Быстрое преобразование Фурье (FFT) — это численный алгоритм, который позволяет значительно сократить количество вычислительных операций при расчете преобразования Фурье или коэффициентов ряда Фурье. Вместо порядка N^2 операций, FFT требует порядка $N \log_2 N$ операций, где N — количество точек данных, что дает существенную экономию времени при больших объемах вычислений. Существуют различные варианты FFT, например, метод Кули-Тьюки.

3 Реализация методов

3.1 Загрузка и отображение изображения

Изображение загружается с помощью библиотеки OpenCV и преобразуется из формата BGR в RGB для корректного отображения с использованием Matplotlib:

```
image_path = 'tipovikmatan.jpg'
color_image = cv2.imread(image_path)
color_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)
```

3.2 Разделение на каналы

Изображение разделяется на красный, зеленый и синий каналы:

```
r_channel = color_image[:, :, 0]
g_channel = color_image[:, :, 1]
b_channel = color_image[:, :, 2]
```

3.3 Свёртка

Реализована функция свёртки, принимающая изображение и ядро:

```
def convolve(image, kernel):
    h, w = image.shape
    kh, kw = kernel.shape
    pad_h = kh // 2
    pad_w = kw // 2
    padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)), mode='
   constant', constant_values=0)
    output = np.zeros_like(image, dtype=float)
    for i in range(h):
        for j in range(w):
            output[i, j] = np.sum(kernel * padded_image[i:i+kh, j:j+kw])
    return output
```

3.4 Пороговая фильтрация

Реализована функция для бинаризации изображения:

```
def threshold_filter(image, threshold_value, high_val=255, low_val=0):
    thresholded_image = np.zeros_like(image)
    thresholded_image[image > threshold_value] = high_val
    thresholded_image[image <= threshold_value] = low_val
    return thresholded_image
```

3.5 Медианная фильтрация

Реализована функция медианной фильтрации:

```
def median_filter(image, kernel_size=3):
    if kernel_size % 2 == 0:
        raise ValueError("Размер ядра должен бут  нечетнум числом")
    img_height, img_width = image.shape
    pad_size = kernel_size // 2
    padded_image = np.pad(image, pad_size, mode='reflect')
```

```
    output_image = np.zeros_like(image)
    for y in range(img_height):
        for x in range(img_width):
            neighborhood = padded_image[y : y + kernel_size, x : x + kernel_
size]
            output_image[y, x] = np.median(neighborhood)
    return output_image.astype(image.dtype)
```

3.6 Гауссовское размытие

Реализована функция для создания гауссовского ядра:

```
def gaussian_kernel(size, sigma=1.0):
    if size % 2 == 0:
        raise ValueError("Размер ядра должен бут  нечетнум числом")
    ax = np.arange(-size // 2 + 1.0, size // 2 + 1.0)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-(xx**2 + yy**2) / (2.0 * sigma**2))
    kernel = kernel / np.sum(kernel)
    return kernel
```

3.7 Боксовое размытие

Реализована функция для создания ядра боксового размытия:

```
def box_blur_kernel(size):
    if size <= 0:
        raise ValueError("Размер ядра должен бут  положител нум числом")
    return np.ones((size, size), dtype=np.float32) / (size * size)
```

3.8 Фильтр Собеля

Реализована функция для применения фильтра Собеля:

```
def sobel_filter(image):
    sobel_x_kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=np.
float32)
    sobel_y_kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=np.
float32)
    gradient_x = convolve(image.astype(np.float32), sobel_x_kernel)
    gradient_y = convolve(image.astype(np.float32), sobel_y_kernel)
    gradient_magnitude = np.abs(gradient_x) + np.abs(gradient_y)
    if np.max(gradient_magnitude) > 0:
        gradient_magnitude = (gradient_magnitude / np.max(gradient_magnitude) *
 255).astype(np.uint8)
    else:
        gradient_magnitude = gradient_magnitude.astype(np.uint8)
    return gradient_magnitude, gradient_x, gradient_y
```

3.9 Изменение размера с помощью DCT

Реализованы функции для изменения размера изображения с помощью DCT:

```
def resize_dct_raw(image, new_shape, keep_coeff=(None, None), smooth_t=0.0, a2
=1 / 500000.0):
    # Приводим изображение к float
```

```
    img = np.asarray(image, dtype=float)
    N, M = img.shape  # Исходные размеры
    H, W = new_shape  # Новые размеры

    # Вспомогательная функция для коэффициента c(n)
    def c(n):
        return 1.0 / np.sqrt(2.0) if n == 0 else 1.0

    # Прямое DCT
    F_full = np.zeros((N, M), dtype=float)
    norm = 2.0 / np.sqrt(N * M)
    for k in range(N):
        ck = c(k)
        for l in range(M):
            cl = c(l)
            S_k_l = 0.0
            for j in range(N):
                T_j_l = 0.0
                cos_j_factor = np.cos(np.pi * k * (j + 0.5) / N)
                for m_val in range(M):
                    cos_m_factor = np.cos(np.pi * l * (m_val + 0.5) / M)
                    T_j_l += img[j, m_val] * cos_m_factor
                S_k_l += cos_j_factor * T_j_l
            F_full[k, l] = norm * ck * cl * S_k_l

    # Усечение коэффициентов
    m_coeffs, n_coeffs = N - 1, M - 1
    if keep_coeff[0] is not None and keep_coeff[1] is not None:
        m_coeffs, n_coeffs = keep_coeff
        if not (0 <= m_coeffs < N and 0 <= n_coeffs < M):
            raise ValueError("keep_coeff должно удовлетворят  0 <= m < N, 0 <=
n < M")

    F_cut = np.zeros_like(F_full)
    F_cut[:m_coeffs + 1, :n_coeffs + 1] = F_full[:m_coeffs + 1, :n_coeffs + 1]


    # Сглаживание эффекта Гиббса
    if smooth_t > 0.0:
        for k in range(m_coeffs + 1):
            for l in range(n_coeffs + 1):
                lam = (np.pi * k / N) ** 2 + (np.pi * l / M) ** 2
                F_cut[k, l] *= np.exp(-a2 * lam * smooth_t)

    # Обратное DCT для интерполяции
    resized = np.zeros((H, W), dtype=float)
    inv_norm = 2.0 / np.sqrt(N * M) # N и M исходные измерения
    for x in range(H):
        for y in range(W):
            sum_val = 0.0
            for k in range(m_coeffs + 1):
                ck = c(k)
                cos_k_factor = np.cos(np.pi * k * (x + 0.5) / H) # Испол зуем н
овое измерение H
                for l in range(n_coeffs + 1):
                    cl = c(l)
                    cos_l_factor = np.cos(np.pi * l * (y + 0.5) / W) #  Испол з
уем новое измерение W
                    sum_val += ck * cl * F_cut[k, l] * cos_k_factor * cos_l_
factor
```

```
            resized[x, y] = inv_norm * sum_val

    return resized
```

```
def resize_dct_fast(image, new_shape, keep_coeff=(None, None), smooth_t=0.0, a2=1 / 500000.0):
    img = np.asarray(image, dtype=float)
    N_orig, M_orig = img.shape # Исходные измерения
    H_new, W_new = new_shape   # Новые измерения

    c_k_orig = np.ones(N_orig, dtype=float)
    c_k_orig[0] = 1.0 / np.sqrt(2.0)
    c_l_orig = np.ones(M_orig, dtype=float)
    c_l_orig[0] = 1.0 / np.sqrt(2.0)

    m_idx_orig = np.arange(M_orig)
    l_idx_orig = np.arange(M_orig)
    angles_M_orig = np.pi * np.outer(m_idx_orig + 0.5, l_idx_orig) / M_orig
    cos_m_table_orig = np.cos(angles_M_orig)

    j_idx_orig = np.arange(N_orig)
    k_idx_orig = np.arange(N_orig)
    angles_N_orig = np.pi * np.outer(j_idx_orig + 0.5, k_idx_orig) / N_orig
    cos_j_table_orig = np.cos(angles_N_orig)

    T = img.dot(cos_m_table_orig)
    B = cos_j_table_orig.T.dot(T)
    norm_factor = 2.0 / np.sqrt(N_orig * M_orig)
    F_full = norm_factor * (c_k_orig[:, None] * B * c_l_orig[None, :])

    m_coeffs, n_coeffs = N_orig - 1, M_orig - 1
    if keep_coeff[0] is not None and keep_coeff[1] is not None:
        m_coeffs, n_coeffs = keep_coeff
        if not (0 <= m_coeffs < N_orig and 0 <= n_coeffs < M_orig):
            raise ValueError("keep_coeff должен удовлетворят  0 <= m < N_orig, 0 <= n < M_orig")

    F_cut = np.zeros_like(F_full)
    F_cut[:m_coeffs + 1, :n_coeffs + 1] = F_full[:m_coeffs + 1, :n_coeffs + 1]

    if smooth_t > 0.0:
        for k_loop in range(m_coeffs + 1):
            for l_loop in range(n_coeffs + 1):
                lam = (np.pi * k_loop / N_orig) ** 2 + (np.pi * l_loop / M_orig) ** 2
                F_cut[k_loop, l_loop] *= np.exp(-a2 * lam * smooth_t)

    # Обратный DCT для новых измерений H_new , W_new
    # Коэффициенту c_k и c_l для усеченных измерений, использ уемые при суммировании
    c_k_trunc = np.ones(m_coeffs + 1, dtype=float)
    if m_coeffs >=0: c_k_trunc[0] = 1.0 / np.sqrt(2.0)
    c_l_trunc = np.ones(n_coeffs + 1, dtype=float)
    if n_coeffs >=0: c_l_trunc[0] = 1.0 / np.sqrt(2.0)

    x_idx_new = np.arange(H_new)
    k_idx_sum = np.arange(m_coeffs + 1) # Суммирование по k до m_coeffs
    angles_H_new = np.pi * np.outer(x_idx_new + 0.5, k_idx_sum) / H_new
    cos_k_table_new = np.cos(angles_H_new)
    # Поелементное умножение cos_k_table_new на c_k_trunc
```

```
    # Размеру D_H: H_new x (m_coeffs + 1)
    D_H = cos_k_table_new * c_k_trunc[None, :]


    y_idx_new = np.arange(W_new)
    l_idx_sum = np.arange(n_coeffs + 1) # Суммирование по l до n_coeffs
    angles_W_new = np.pi * np.outer(y_idx_new + 0.5, l_idx_sum) / W_new
    cos_l_table_new = np.cos(angles_W_new)
    # Поелементное умножение cos_l_table_new на c_l_trunc
    # Размеру D_W: W_new x (n_coeffs + 1)
    D_W = cos_l_table_new * c_l_trunc[None, :]


    F_sub = F_cut[:m_coeffs + 1, :n_coeffs + 1] # Подматрица F_cut

    # inv_norm_factor - то же самое, что и norm_factor, основанный на исходных N, M
    inv_norm_factor = 2.0 / np.sqrt(N_orig * M_orig)
    resized = inv_norm_factor * (D_H.dot(F_sub).dot(D_W.T))

    return resized
```

3.10 Преобразование Фурье

Реализованы функции для FFT и фильтрации в частотной области:

```
def fft2d(img):
    h = 2 ** np.ceil(np.log2(img.shape[0])).astype(int)  # Дополняем до степени двойки
    w = 2 ** np.ceil(np.log2(img.shape[1])).astype(int)
    padded_img = np.pad(img, ((0, h - img.shape[0]), (0, w - img.shape[1])), mode='constant')
    rows = np.array([fft1d(row) for row in padded_img], dtype=complex)  # FFT по строкам
    cols = np.array([fft1d(col) for col in rows.T], dtype=complex).T  # FFT по столбцам
    return cols
```

```
def apply_frequency_filter(img, filter_type='low', radius=30):
    img_arr = np.asarray(img, dtype=float)
    F = fft2d(img_arr)  # Прямое FFT
    H, W = F.shape
    F_shifted = fftshift2d(F)  # Сдвигаем спектр

    # Создаём маску в зависимости от типа фил тра
    Y, X = np.ogrid[:H, :W]
    center_y, center_x = H // 2, W // 2
    distance = np.sqrt((Y - center_y) ** 2 + (X - center_x) ** 2)
    mask = np.zeros((H,W), dtype=bool) # Инициализация маски
    if filter_type == 'low':
        mask[distance <= radius] = True  # Низкие частоту
    elif filter_type == 'high':
        mask[distance >= radius] = True  # Вусокие частоту
    else:
        raise ValueError("filter_type должен бут  'low' или 'high'")

    F_filtered = F_shifted * mask  # Применяем маску
    F_unshifted = ifftshift2d(F_filtered)  # Обратнуй сдвиг
    img_filtered = np.real(ifft2d(F_unshifted))  # Обратное FFT, берём вещественную част
```

```
    # Обрезаем изображение до исходнух размеров перед возвратом
    return np.clip(img_filtered[:img_arr.shape[0], :img_arr.shape[1]], 0, 255).
astype(np.uint8)
```

4 Экспериментальная часть

4.1 Отображение каналов

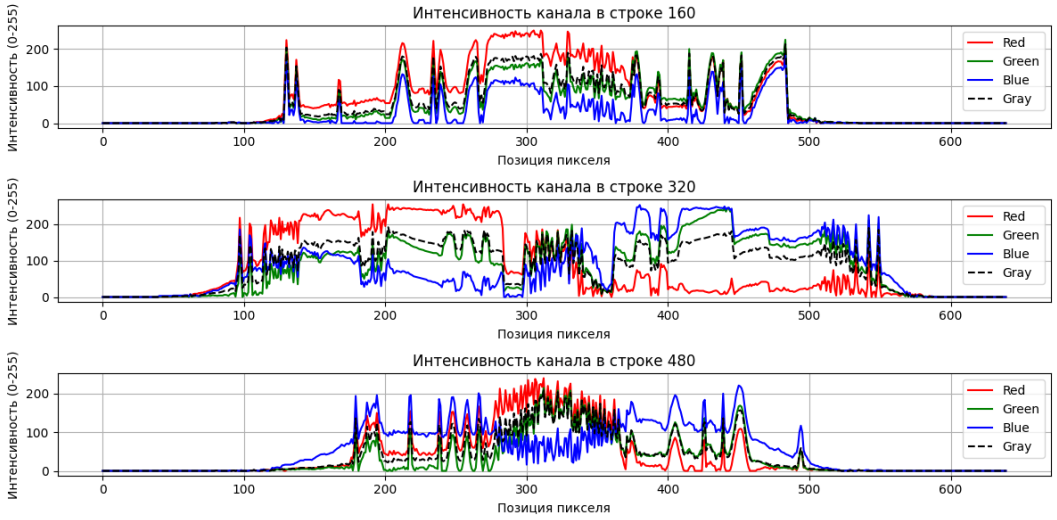

Рис. 1: Красный, зелёный и синий каналы изображения. Каждый канал представлен в градациях серого, где яркость пикселя соответствует интенсивности цвета. Это позволяет проанализировать вклад каждого цвета в общее изображение.

4.2 Свёртка

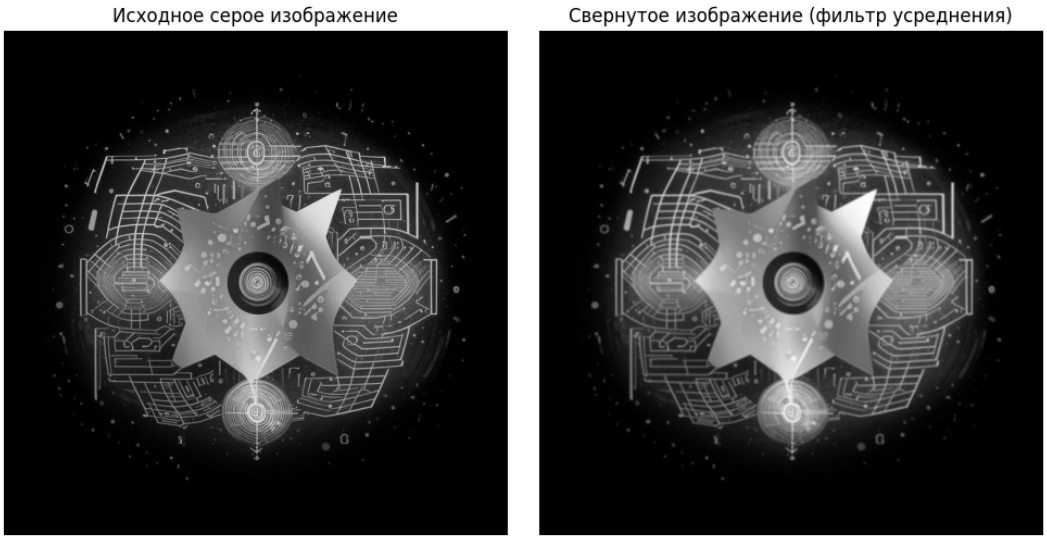

Рис. 2: Слева — исходное серое изображение, справа — результат свёртки с ядром усреднения 3x3. Свёртка сглаживает изображение, уменьшая резкие перепады яркости.

4.3 Пороговая фильтрация

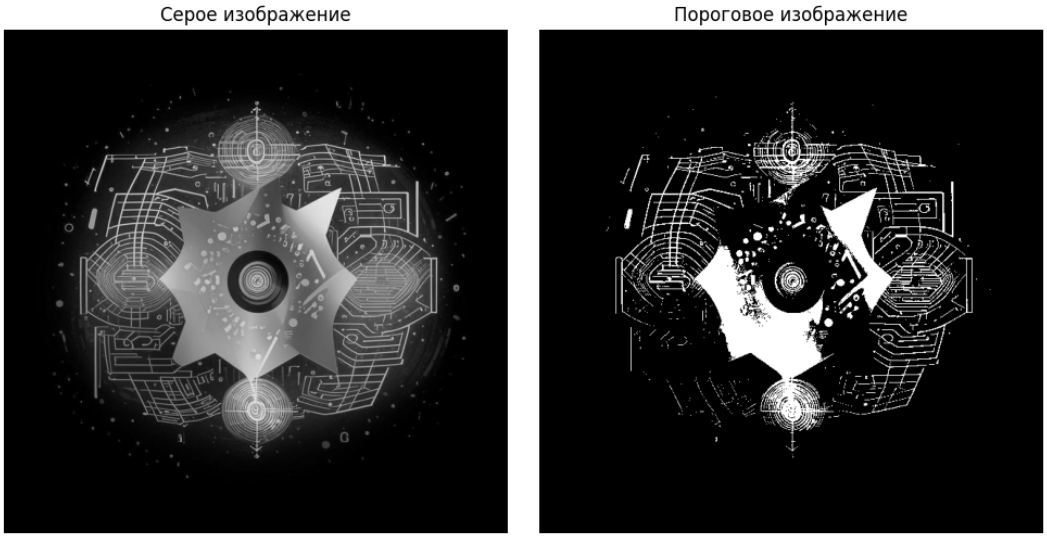

Рис. 3: Слева — серое изображение, справа — бинарное изображение после пороговой фильтрации с порогом 127. Пиксели с интенсивностью выше порога становятся белыми, ниже — чёрными.

4.4 Медианная фильтрация

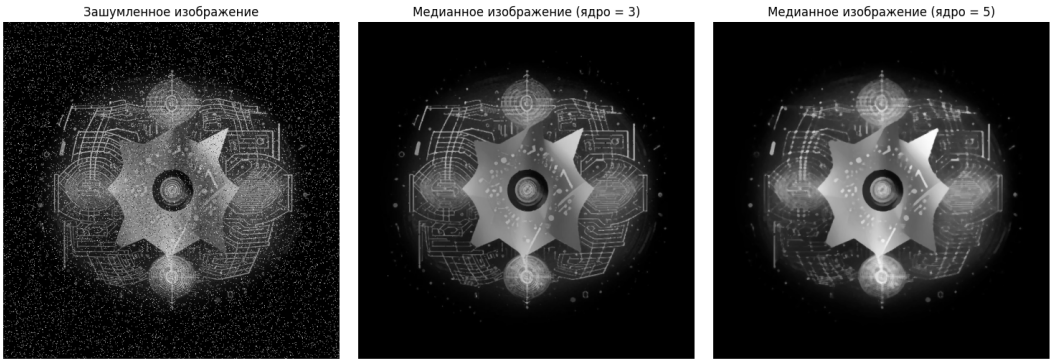

Рис. 4: Слева — зашумленное изображение с шумом 'соль и перец', в центре — результат медианной фильтрации с ядром 3x3, справа — с ядром 5x5. Медианный фильтр эффективно удаляет шум, сохраняя края объектов.

4.5 Гауссовское размытие

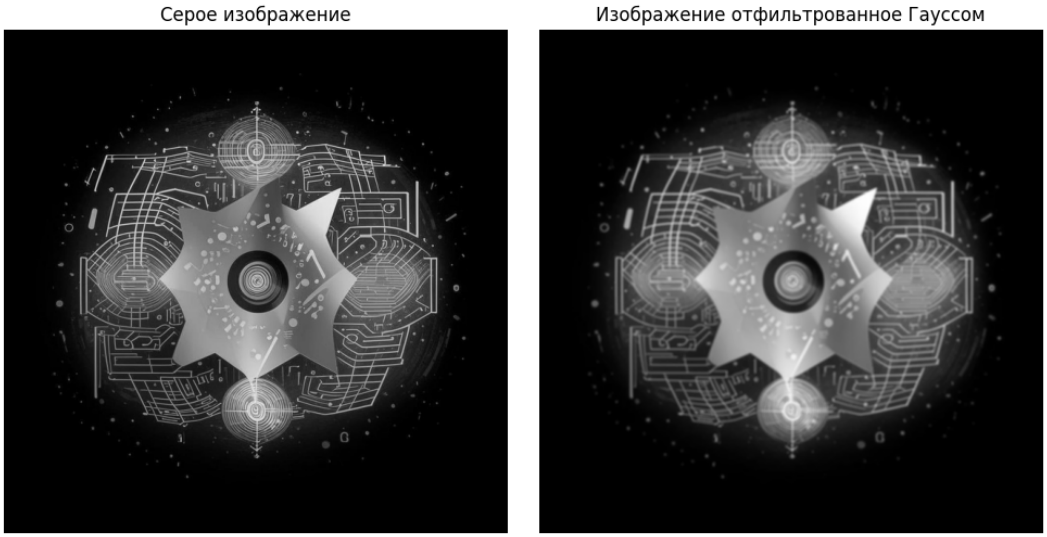

Рис. 5: Слева — серое изображение, справа — изображение после гауссовского размытия с ядром 5x5 и sigma=1.5. Размытие сглаживает изображение, уменьшая высокочастотные детали.

4.6 Боксовое размытие

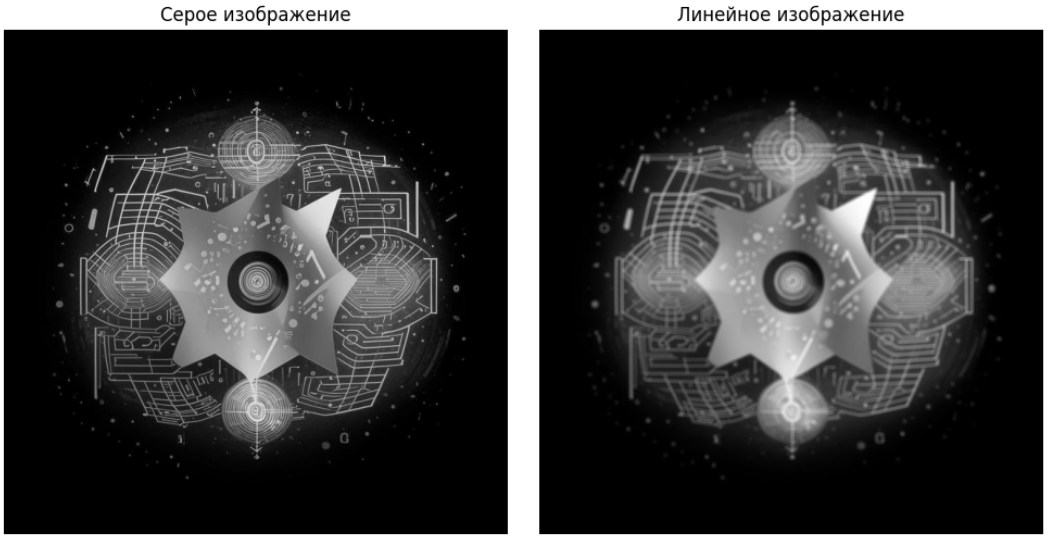

Рис. 6: Слева — серое изображение, справа — изображение после боксового размытия с ядром 5x5. Боксовое размытие также сглаживает изображение, но менее плавно, чем гауссовское.

4.7 Фильтр Собеля

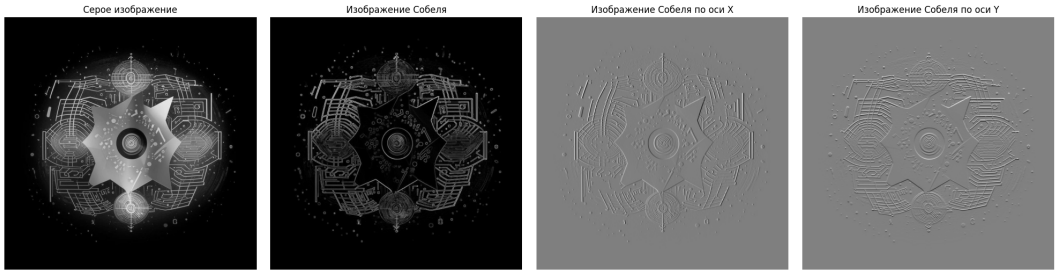

Рис. 7: Слева — серое изображение, справа — изображение с выделенными краями после применения фильтра Собеля. Фильтр выявляет контуры объектов, подчёркивая перепады яркости.

4.8 Изменение размера с помощью DCT

Рис. 8: Слева — оригинальное изображение, в центре — уменьшенное изображение (200x200), справа — увеличенное изображение (1000x1000) с использованием DCT. DCT позволяет масштабировать изображение с сохранением качества.

4.9 Преобразование Фурье

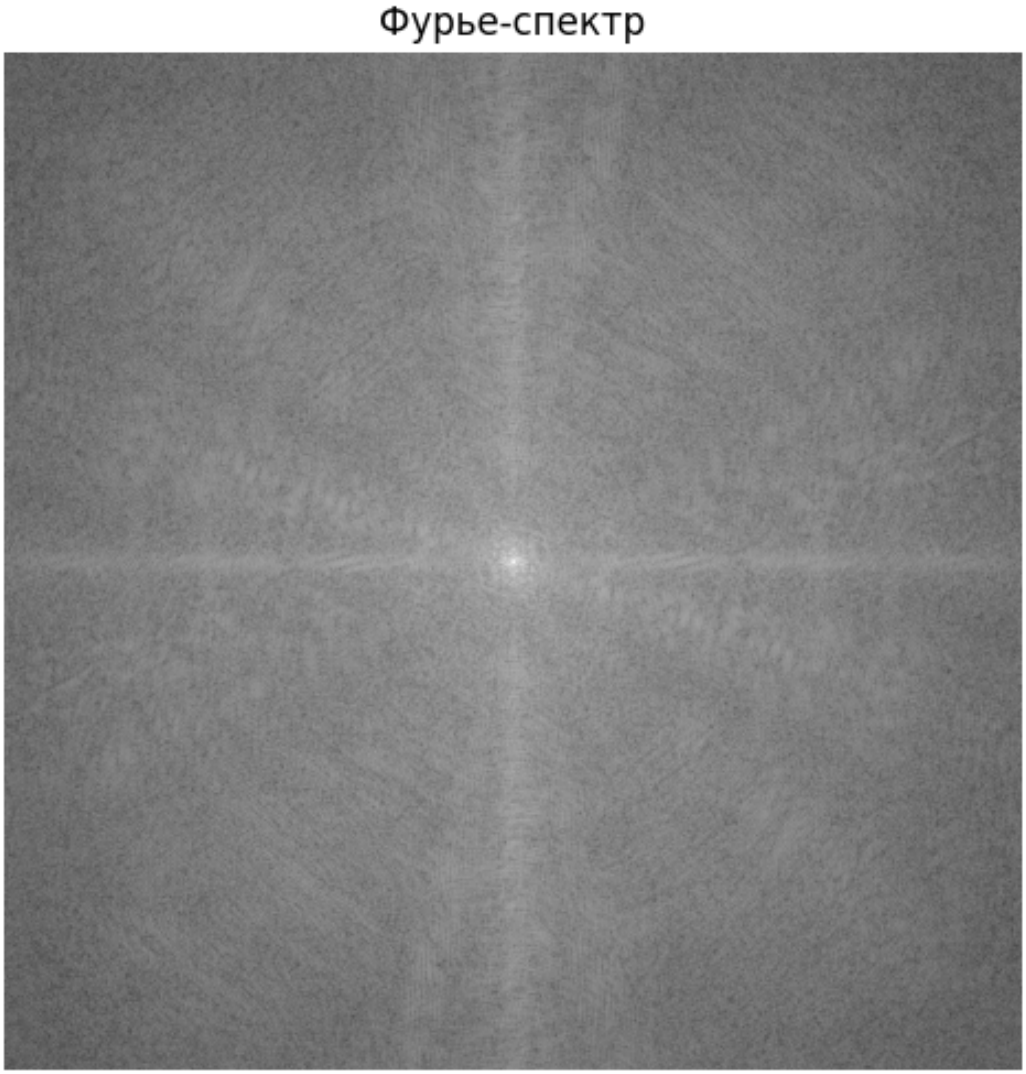

Рис. 9: Фурье-спектр изображения показывает его частотный состав, то есть какие типы и интенсивности частотных компонент (например, линий, краев) присутствуют в изображении.

4.10 Преобразование Фурье

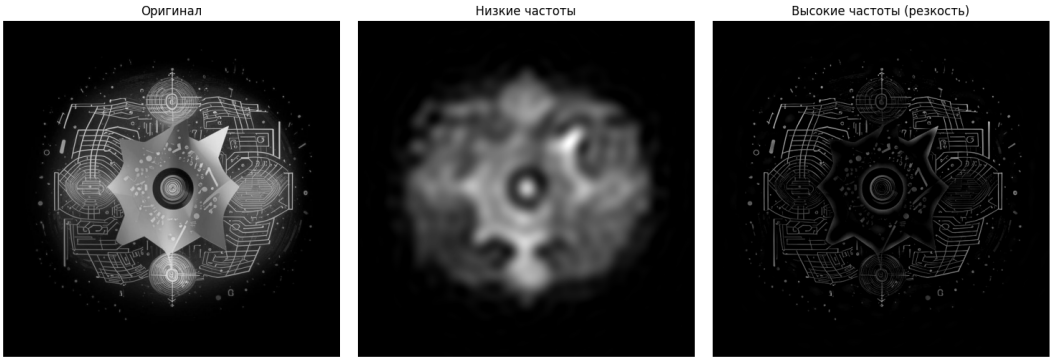

Рис. 10: Слева - оригинал изображения, в центре - после низкочастотной фильтрации, справа — после высокочастотной фильтрации. Низкочастотная фильтрация сглаживает изображение, а высокочастотная выделяет мелкие детали.

5 Анализ результатов

В ходе работы были реализованы и исследованы различные методы обработки изображений. Разделение на каналы позволило увидеть вклад каждого цвета в изображение. Свёртка с различными ядрами показала, как можно размывать изображение или выделять края. Фильтрация (пороговая и медианная) продемонстрировала способы улучшения качества изображения. Преобразование Фурье и фильтрация в частотной области показали возможности анализа и модификации частотных характеристик изображения.

6 Заключение

В данной лабораторной работе были изучены и реализованы методы обработки изображений, включая работу с каналами, свёртку, фильтрацию и преобразование Фурье. Полученные результаты демонстрируют эффективность этих методов для различных задач обработки изображений.

7 Приложение

Ссылка на код