



Федеральное государственное автономное образовательное учреждение высшего образования «Национальный Исследовательский Университет ИТМО»

**ЛАБОРАТОРНАЯ РАБОТА №6
ПРЕДМЕТ «ЧАСТОТНЫЕ МЕТОДЫ»
ТЕМА «ОБРАБОТКА ИЗОБРАЖЕНИЙ»**

Лектор: Перегудин А. А.
Практик: Пашенко А. В.
Студент: Румянцев А. А.
Поток: ЧАСТ.МЕТ. 1.3

Факультет: СУиР
Группа: R3241

Санкт-Петербург
2024

Содержание

1 Задание 1. Фильтрация изображений с периодичностью	2
1.1 Используемые программы	4
2 Задание 2. Размытие изображения	7
2.1 Используемые программы	9
3 Задание 3. Увеличение резкости	11
3.1 Используемые программы	12
4 Задание 4. Выделение краев	14
4.1 Используемые программы	15

1 Задание 1. Фильтрация изображений с периодичностью

Скачаем изображение под номером 4 с данного [гугл-диска](#).



Рис. 1: Изображение, выбранное с представленного гугл-диска

Далее проделаем следующие шаги:

1. Загрузим это изображение в `python` с помощью библиотеки `pillow`.
2. Преобразуем полученный массив к вещественному типу с помощью инструментов библиотеки `numpy`. Поделим все значения на 255 – максимальное значение яркости цвета.
3. Найдем двумерный Фурье-образ массива и сдвинем его в центр с помощью команд `fftshift(fft2(my_image))`. Так как изображение цветное, выполним преобразование для каждого из цветовых каналов по отдельности.
4. Разделим полученные образы каждого канала на массивы модулей и аргументов с помощью функций библиотеки `numpy abs` и `angle`.
5. Для удобства работы, найдем логарифм от каждого массива модулей и нормализуем его значения в диапазон от 0 до 1. Чтобы избежать неопределенности в логарифме, предварительно прибавим ко всем значениям 1. Для этого понадобится команда `pr.log1p()`. Для обратной операции `pr.expm1()`.
6. Объединим преобразованные каналы. Сохраним полученный массив (нормализованный логарифм модуля Фурье-образа) как изображение командой `save`, вызываемой от объекта `Image`.
7. Проанализируем полученное на предыдущем шаге изображение. Найдем пики, соответствующие периодичности на исходной картинке.
8. В программе редактирования изображений исправим полученный Фурье-образ: сгладим все ненужные цветовые пики, отвечающие за гармоники, от которых мы хотим избавиться.

9. Восстановим картинку из отредактированного образа, проделав обратные шаги.

Цветной нормализованный логарифм модуля Фурье-образа данного изображения представлен на следующем рисунке.

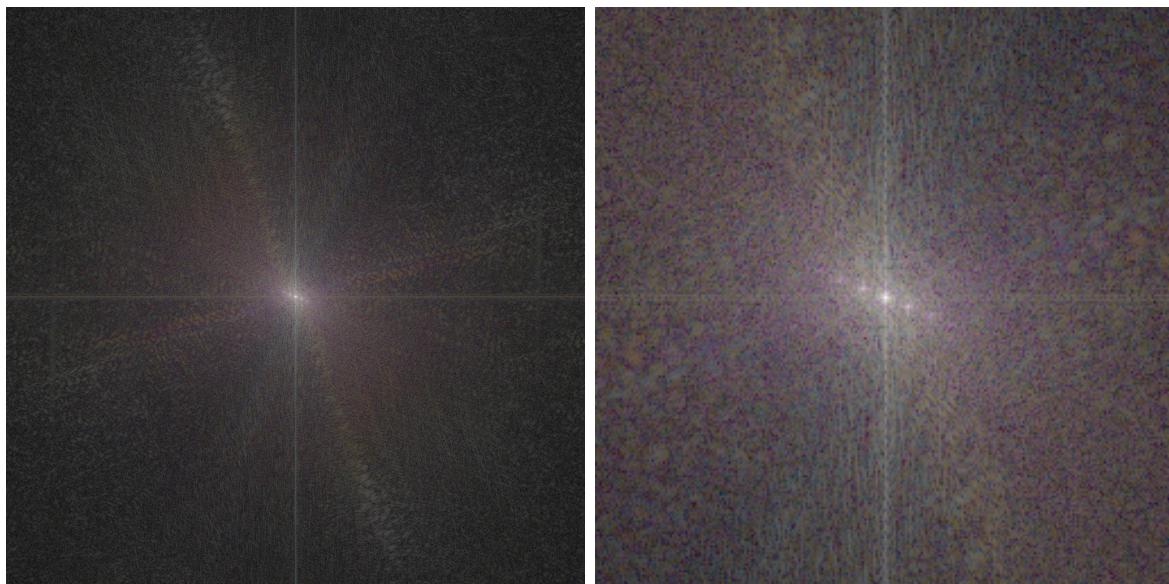


Рис. 2: Нормализованный логарифм модуля Фурье-образа

Видим на рис. 2b около центра цветовые пики во второй и четвертой четвертях системы координат, если задать ее в центре изображения. Нам необходимо сгладить эти цветовые пики, отвечающие за синусоидальный шум на картинке. Сделаем это в редакторе изображений *paint* – наложим «пластиры» из наиболее близких к пикам незасветленных областей с пикселями на места гармонического шума. Результат исправления расположен ниже.

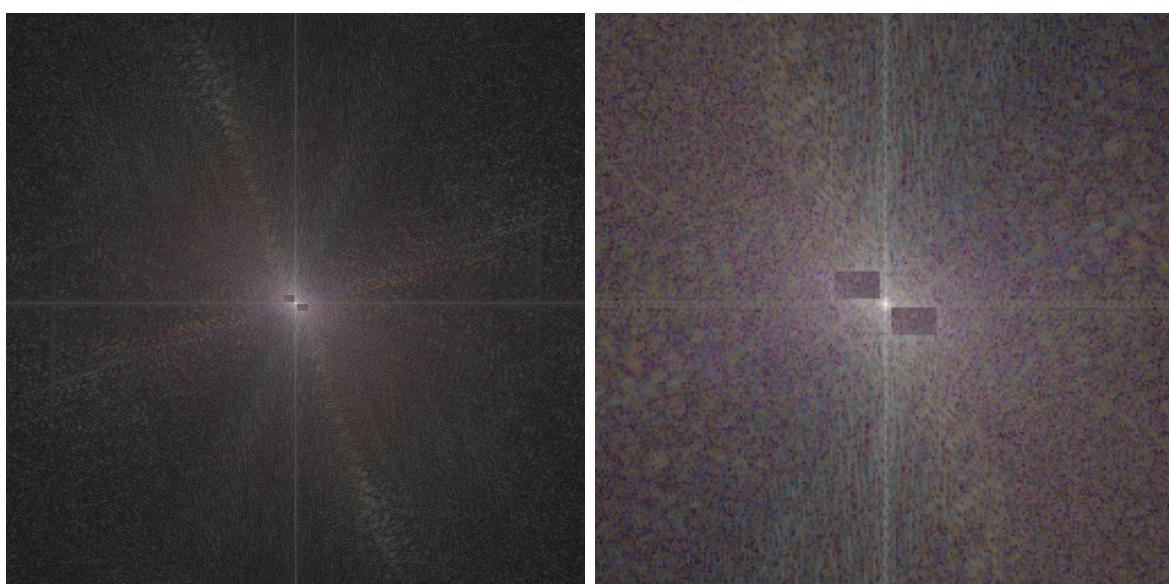


Рис. 3: Исправленный нормализованный логарифм модуля Фурье-образа

Ниже представлен сравнительный рисунок – оригинальное изображение и фильтрованное. Изображение без синусоидального шума (или с минимальным) выглядит более «мягко». Море и небо стали более естественными.



Рис. 4: Сравнение исходного и фильтрованного изображений

Для восстановления изображения понадобилась формула

$$\hat{X} = |\hat{X}|_{new} \cdot e^{i\angle\hat{X}},$$

где \hat{X} – Фурье-образ изображения, $|\hat{X}|_{new}$ фильтрованный массив модулей, $e^{i\angle\hat{X}}$ комплексное число с аргументом $\angle\hat{X}$, представляющим массив углов Фурье-образа. $|\hat{X}|_{new}$ отвечает за новые амплитуды частотных компонент, а $e^{i\angle\hat{X}}$ за фазы исходных амплитуд. Без массива углов мы бы не смогли привести изображение к изначальной «структуре».

1.1 Используемые программы

Для реализации программ, необходимых для выполнения задания 1, были использованы язык программирования `python` с библиотеками `pillow` и `numpy`.

```
from PIL import Image
import numpy as np

# Method to read image along the specified path
def read_img(path: str):
    return Image.open(path)

# Method to save image along the specified path
def save_img(img: Image, path: str):
    img.save(path)

# Method to convert an array to an image
def convert_arr_to_img(arr):
    return Image.fromarray(arr)

# Method to normalize data (ex. logarithm of abs(image))
```

```

# Returns min and max to inverse the normalization
def normalize(data):
    min_ = np.min(data)
    max_ = np.max(data)
    nz = (data - min_) / (max_ - min_)
    return nz, min_, max_

# Method to remove normalization from data
# Requires min and max to inverse normalization
def denormalize(nz, min_, max_):
    return nz * (max_ - min_) + min_

# Method to split rgb image array into 3 channels
def rgb_channels(img: Image):
    img_arr = np.array(img)
    r = img_arr[:, :, 0]
    g = img_arr[:, :, 1]
    b = img_arr[:, :, 2]
    return r, g, b

# Method to split rgb image angles array into 3 for each channel
def rgb_angles(angles):
    ar = angles[:, :, 0]
    ag = angles[:, :, 1]
    ab = angles[:, :, 2]
    return ar, ag, ab

# Method to apply fft2 to a channel
# Uses special algorithm for convenience
def fft2_channel(channel):
    channel = channel.astype(np.float64) / 255
    fft_ = np.fft.fftshift(np.fft.fft2(channel))
    abs_ = np.abs(fft_)
    angle = np.angle(fft_)
    log = np.log1p(abs_)
    nz, min_, max_ = normalize(log)
    return nz, min_, max_, angle

# Method to apply ifft2 to a channel
# Inverts all the steps of the special algorithm
def ifft2_channel(channel, angle, min_, max_):
    channel = channel.astype(np.float64) / 255
    denor = denormalize(channel, min_, max_)
    abs_ = np.expm1(denor)
    col_res = abs_ * np.exp(1j * angle)
    ifft_ = np.fft.ifft2(np.fft.ifftshift(col_res))
    return np.real(ifft_)

# Method to apply fft2 to an image. Processes 3 channels at once
def fft2(img: Image):
    r, g, b = rgb_channels(img)

    nzr, min_r, max_r, ar = fft2_channel(r)
    nzg, min_g, max_g, ag = fft2_channel(g)
    nzb, min_b, max_b, ab = fft2_channel(b)

    res = np.stack((nzr, nzg, nzb), axis=2)
    res = (res * 255).astype(np.uint8)

    ang = np.stack((ar, ag, ab), axis=2)

```

```

nz_min_max = [(min_r, max_r), (min_g, max_g), (min_b, max_b)]
return res, ang, nz_min_max

# Method to apply ifft2 to an image. Processes 3 channels at once
def ifft2(img: Image, angles, nz_min_max):
    r, g, b = rgb_channels(img)
    ar, ag, ab = rgb_angles(angles)

    nz_r_min, nz_r_max = nz_min_max[0][0], nz_min_max[0][1]
    nz_g_min, nz_g_max = nz_min_max[1][0], nz_min_max[1][1]
    nz_b_min, nz_b_max = nz_min_max[2][0], nz_min_max[2][1]

    new_r = ifft2_channel(r, ar, nz_r_min, nz_r_max)
    new_g = ifft2_channel(g, ag, nz_g_min, nz_g_max)
    new_b = ifft2_channel(b, ab, nz_b_min, nz_b_max)

    res = np.stack((new_r, new_g, new_b), axis=2)
    res = normalize(res)[0] * 255
    return res.astype(np.uint8)

# Helper method for external use of fft2
def fft2_2img(img: Image):
    res, ang, nz_min_max = fft2(img)
    return convert_arr_to_img(res), ang, nz_min_max

# Helper method for external use of ifft2
def ifft2_2img(img: Image, angles, nz_min_max):
    return convert_arr_to_img(ifft2(img, angles, nz_min_max))

```

Листинг 1: Программные методы, необходимые для задания 1

Все вышеперечисленные наработки используются в программе, представленной ниже.

```

import img_utils as iu

# Specify the image source
src = 'fm_lab6/src'
img_path = f'{src}/4.png'

# Specify where to render the result
render_to = 'fm_lab6/renderers/task1'
rimg_path = f'{render_to}/fft2.png'
reimg_path = f'{render_to}/new4.png'

# Specify the corrected image source
# Parameter "corrected" if that source does exist
corr_im_path = f'{src}/corr_fft2.png'
corrected = True

# Reading image, applying fft2 and saving the result
img = iu.read_img(img_path)
ans, ang, nzmm = iu.fft2_2img(img)
iu.save_img(ans, rimg_path)

# Reading corr. im., applying ifft2 to recover im. and saving the res.
if corrected:
    img2 = iu.read_img(corr_im_path)
    ans2 = iu.ifft2_2img(img2, ang, nzmm)
    iu.save_img(ans2, reimg_path)

```

Листинг 2: Реализация задания 1

2 Задание 2. Размытие изображения

Оставим то же изображение, что и в предыдущем пункте. Проделаем следующие шаги:

1. Загрузим изображение в `python`. Будем обрабатывать каждый цветовой канал по отдельности.
2. Выберем три нечетных значения $n \geq 3$. Пусть $n = 9, 13, 21$.
3. *Блоchное размытие.* Создадим три матрицы ядра блочного размытия. Для этого используем команду `np.ones((n,n))/(n**2)`.
4. *Размытие по Гауссу.* Создадим три матрицы ядра гауссовского размытия. Для этого заполним матрицу $n \times n$ значениями функции

$$f(x, y) = e^{-\frac{9}{n^2} \left(\left(x - \frac{n+1}{2} \right)^2 + \left(y - \frac{n+1}{2} \right)^2 \right)}$$

и поделим все значения на сумму полученных элементов. Важно, чтобы значения матрицы были центрально-симметричны.

5. Выполним свертку каждого канала исходного изображения с каждым из ядер для каждого из размытий командой `convolve2d`. Для этого понадобится библиотека `scipy`. Должно получиться шесть результатов.
6. Проанализируем полученные изображения. Если размытие будет слабо заметно, то возьмем большее значение n .
7. Найдем Фурье-образы от каждого канала исходного изображения и от каждого из ядер, заполнив пропуски нулями. Для этого используем дополнительный параметр функции `fft2(my_image, s=(h, w))`, где h и w – высота и ширина исходного изображения.
8. Поэлементно перемножим Фурье-образы каждого канала изображения с образами каждого из ядер.
9. Выполним обратное преобразование Фурье от полученных произведений.
10. Сравним изображения, полученные с помощью свертки и Фурье-преобразования.
11. Сравним качество блочного и гауссовского размытия.



(a) Блоchное при $n = 9$

(b) Блоchное при $n = 13$

(c) Блоchное при $n = 21$

Рис. 5: Блоchные размытия сверткой для $n = 9, 13, 21$

(a) Гаусс при $n = 9$ (b) Гаусс при $n = 13$ (c) Гаусс при $n = 21$ Рис. 6: Размытия по Гауссу сверткой для $n = 9, 13, 21$ (a) Блочное при $n = 9$ (b) Блочное при $n = 13$ (c) Блочное при $n = 21$ Рис. 7: Блочные размытия произведением Фурье-образов для $n = 9, 13, 21$ (a) Гаусс при $n = 9$ (b) Гаусс при $n = 13$ (c) Гаусс при $n = 21$ Рис. 8: Размытия по Гауссу произведением Фурье-образов для $n = 9, 13, 21$

Сравнивая рисунки 5 с 7 и 6 с 8 видим, что результаты свертки совпадают с результатами произведения Фурье-образов. Это можно объяснить теоремой о свертке, представляемой формулой

$$\mathcal{F}\{g\} \mathcal{F}\{f\} = \mathcal{F}\{g * f\},$$

то есть после произведения Фурье-образа изображения g с образом ядра f и обратного преобразования Фурье мы получили свертку $g * f$. Следовательно, выполнилась теорема о свертке.

На всех изображениях размытие хорошо видно. При увеличении значения параметра n размытие увеличивается. Блочное размытие более резкое – сильнее размывает детали на изображении по сравнению с размытием по Гауссу при тех же значениях параметра n (детализация корабля см. рис. 5c и 6c). Следовательно, размытие по Гауссу более качественное.

2.1 Используемые программы

Для выполнения данного задания были написаны новые методы, а также использованы некоторые старые, представленные в предыдущем листинге 1. Добавилась библиотека `scipy`.

```

import scipy.signal as sps

# Method to apply fft2 to each channel
def raw_fft2(img: Image):
    r, g, b = rgb_channels(img)

    new_r = np.fft.fftshift(np.fft.fft2(r))
    new_g = np.fft.fftshift(np.fft.fft2(g))
    new_b = np.fft.fftshift(np.fft.fft2(b))

    return np.stack((new_r, new_g, new_b), axis=2)

# Method to apply ifft2 to each channel
def raw_ifft2(arr):
    r, g, b = rgb_channels(arr)

    new_r = np.fft.ifft2(np.fft.ifftshift(r))
    new_g = np.fft.ifft2(np.fft.ifftshift(g))
    new_b = np.fft.ifft2(np.fft.ifftshift(b))

    return np.stack((np.clip(new_r.real, 0, 255), np.clip(
        new_g.real, 0, 255), np.clip(new_b.real, 0, 255)),
                    axis=2).astype(np.uint8)

# Method to apply convolve2d to each channel
# Uses mode='same' as default to save image side dimensions
def convolve2d(img: Image, a, mode='same'):
    r, g, b = rgb_channels(img)

    c2r = sps.convolve2d(r, a, mode)
    c2g = sps.convolve2d(g, a, mode)
    c2b = sps.convolve2d(b, a, mode)

    return (np.stack((np.clip(c2r, 0, 255),
                      np.clip(c2g, 0, 255), np.clip(c2b, 0, 255)),
                     axis=2)).astype(np.uint8)

# Method that returns block kernel
def block_kernel(n: int):
    return np.ones((n, n)) / (n**2)

# Method that calculates and returns gaussian kernel
def gaussian_kernel(n: int):
    a = np.zeros((n, n), dtype=np.float32)
    sum_ = 0
    for x in range(n):

```

```

        for y in range(n):
            pow_ = -9 / (n**2) * \
                   ((x - (n + 1) / 2)**2 + (y - (n + 1) / 2)**2)
            res = np.exp(pow_)
            a[x][y] = res
            sum_ += res
        a /= sum_
    return a

# Method that gets block kernel and returns the convolve2d result
def block_blur_conv2(img: Image, n: int, mode='same'):
    a = block_kernel(n)
    return convolve2d(img, a, mode)

# Method that gets gaussian kernel and returns the convolve2d result
def gaussian_blur_conv2(img: Image, n: int, mode='same'):
    a = gaussian_kernel(n)
    return convolve2d(img, a, mode)

# Method to apply kernel to each channel
def apply_kernel(arr3, ker):
    r, g, b = rgb_channels(arr3)

    new_r = r * ker
    new_g = g * ker
    new_b = b * ker

    return np.stack((new_r, new_g, new_b), axis=2)

# Method to blur images with fft2
# Uses abstract kernel
def fft2_blur(img: Image, kernel, n):
    a = kernel(n)
    w, h = img.size

    fft2_img = raw_fft2(img)
    fft2_ker = np.fft.fftshift(np.fft.fft2(a, s=(h, w)))
    img_ker = apply_kernel(fft2_img, fft2_ker)

    return raw_ifft2(img_ker)

# Method that passes block ker. and returns the blurred fft2 res.
def block_blur_fft2(img: Image, n: int):
    return fft2_blur(img, block_kernel, n)

# Method that passes gaussian ker. and returns the blurred fft2 res.
def gaussian_blur_fft2(img: Image, n: int):
    return fft2_blur(img, gaussian_kernel, n)

# Helper method for external use of block blur with conv2
def block_blur_conv2_2img(img: Image, n: int, mode='same'):
    return convert_arr_to_img(block_blur_conv2(img, n, mode))

# Helper method for external use of gaussian blur with conv2
def gaussian_blur_conv2_2img(img: Image, n: int, mode='same'):
    return convert_arr_to_img(gaussian_blur_conv2(img, n, mode))

# Helper method for external use of block blur with fft2
def block_blur_fft2_2img(img: Image, n: int):
    return convert_arr_to_img(block_blur_fft2(img, n))

```

```
# Helper method for external use of gaussian blur with fft2
def gaussian_blur_fft2_2img(img: Image, n: int):
    return convert_arr_to_img(gaussian_blur_fft2(img, n))
```

Листинг 3: Программные методы, необходимые для задания 2

Все вышеперечисленные доработки используются в программе, представленной ниже.

```
import img_utils as iu

# Specify the image source and render output
src = 'fm_lab6/src'
img_path = f'{src}/4.png'
render_to = 'fm_lab6/render/task2',
r1 = f'{render_to}/block_blur'
r2 = f'{render_to}/gaussian_blur'

# Starting cycle to check different n values
img = iu.read_img(img_path)
n = [9, 13, 21]
for i in n:
    # Rendering block blur with conv2
    conv2_i = iu.block_blur_conv2_2img(img, i)
    rimg_path1 = f'{r1}/bl_c2_n={i}.png'
    iu.save_img(conv2_i, rimg_path1)

    # Rendering gaussian blur with conv2
    conv2_i2 = iu.gaussian_blur_conv2_2img(img, i)
    rimg_path3 = f'{r2}/g_c2_n={i}.png'
    iu.save_img(conv2_i2, rimg_path3)

    # Rendering block blur with fft2
    fft2_i = iu.block_blur_fft2_2img(img, i)
    rimg_path2 = f'{r1}/bl_fft2_n={i}.png'
    iu.save_img(fft2_i, rimg_path2)

    # Rendering gaussian blur with fft2
    fft2_i2 = iu.gaussian_blur_fft2_2img(img, i)
    rimg_path4 = f'{r2}/g_fft2_n={i}.png'
    iu.save_img(fft2_i2, rimg_path4)
```

Листинг 4: Реализация задания 2

3 Задание 3. Увеличение резкости

Для выполнения данного задания используем то же изображение, что и в предыдущем пункте. Проделаем следующие шаги:

1. Загрузим изображение в python. Будем обрабатывать каждый цветовой канал по отдельности.
2. Зададим матрицу ядра увеличения резкости

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

3. Найдем свертку каждого из трех каналов исходного изображения с ядром командой `convolve2d` из библиотеки `scipy`.
4. Проанализируем полученные изображения. Если увеличение резкости будет слабо заметно, применим свертку с ядром еще несколько раз.
5. Найдем Фурье-образы каждого канала изображения и ядра, как в задании 2.
6. Поэлементно перемножим Фурье-образы каждого канала изображения и ядра и выполним обратное преобразование Фурье от их произведений. Объединим преобразованные каналы.
7. Сравним изображения, полученные двумя способами.



Рис. 9: Увеличения резкости сверткой и произведением Фурье-образов

Видим, что изображения в обоих случаях стали более четкими, применять преобразование еще раз не требуется. Разницу с оригинальным хорошо видно при сравнении детализации корабля и облаков. Результат сверткой и произведением Фурье-образов совпадают, следовательно выполнилась теорема о свертке.

3.1 Используемые программы

Для выполнения данного задания понадобилась написать новые методы, а также воспользоваться уже сделанными, приведенными в предыдущих листингах 1 и 3.

```
# Method to sharpen image with conv2
def sharpen_conv2(img: Image, n: int = 1, K=None):
    if n <= 0:
        return img

    if K is None:
        K = [[0, -1, 0], [-1, 5, -1], [0, -1, 0]]

    ans = convolve2d(img, K)
```

```

for i in range(n - 1):
    ans = convolve2d(ans, K)

return ans

# Method to sharpen image with fft2
def sharpen_fft2(img: Image, n: int = 1, K=None):
    if n <= 0:
        return img

    if K is None:
        K = [[0, -1, 0], [-1, 5, -1], [0, -1, 0]]

    w, h = img.size

    fft2_img = raw_fft2(img)
    fft2_ker = np.fft.fftshift(np.fft.fft2(K, s=(h, w)))

    img_ker = apply_kernel(fft2_img, fft2_ker)
    for i in range(n - 1):
        img_ker = apply_kernel(img_ker, fft2_ker)

    return raw_ifft2(img_ker)

# Helper method for external use of sharpen with conv2
def sharpen_conv2_2img(img: Image, n: int = 1, K=None):
    return convert_arr_to_img(sharpen_conv2(img, n, K))

# Helper method for external use of sharpen with fft2
def sharpen_fft2_2img(img: Image, n: int = 1, K=None):
    return convert_arr_to_img(sharpen_fft2(img, n, K))

```

Листинг 5: Программные методы, необходимые для задания 3

Ниже приведена программа, применяющая написанные методы.

```

import img_utils as iu

# Specify the image source and render output
src = 'fm_lab6/src'
img_path = f'{src}/4.png'
render_to = 'fm_lab6/renderers/task3'

# Starting cycle to check different n values
img = iu.read_img(img_path)
n = [1, 2]
for i in n:
    # Rendering sharpen with conv2
    ans = iu.sharpen_conv2_2img(img, i)
    iu.save_img(ans, f'{render_to}/sharp_conv2_n={i}.png')

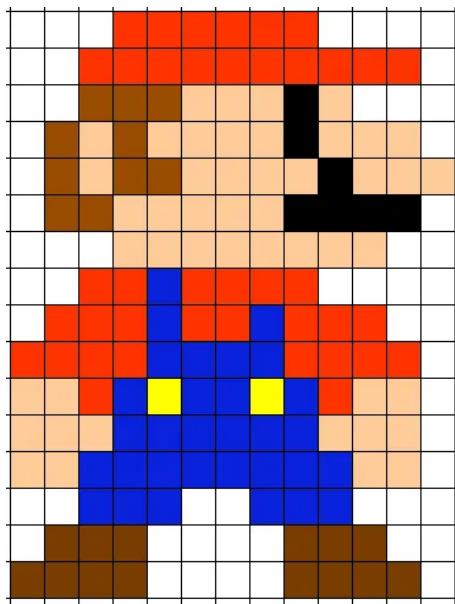
    # Rendering sharpen with fft2
    ans2 = iu.sharpen_fft2_2img(img, i)
    iu.save_img(ans2, f'{render_to}/sharp_fft2_n={i}.png')

```

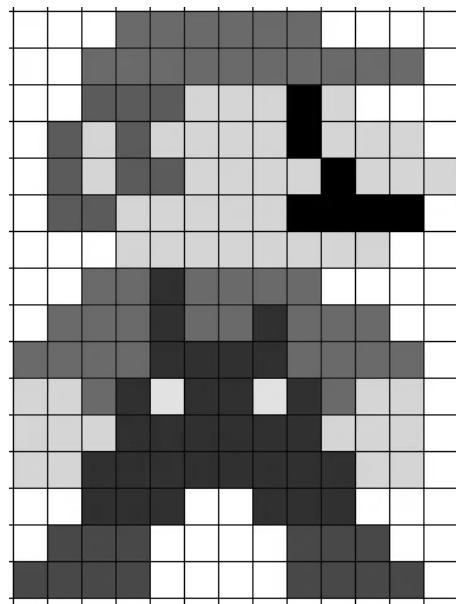
Листинг 6: Реализация задания 3

4 Задание 4. Выделение краев

Найдем в интернете качественное изображение малого размера в стиле pixel art. Пусть это будет изображение, представленное ниже.



(a) Исх. изобр. для задания 4



(b) Черно-белое изобр. для задания 4

Рис. 10: Исходное и преобразованное к черно-белому изображения для задания 4

Далее проделаем следующие шаги:

1. Загрузим изображение в python, преобразуем его в черно-белое (команда `im2gray`) и поделим все элементы на 255.
2. Зададим матрицу ядра выделения краев
$$K = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$
3. Найдем свертку исходного изображения с ядром командой `convolve2d`.
4. Нормализуем значения полученного массива в диапазон от 0 до 1.
5. Проанализируем полученные изображения.
6. Найдем Фурье-образы исходного изображения и ядра.
7. Поэлементно перемножим Фурье-образы изображения и ядра и выполним обратное преобразование Фурье от их произведения.
8. Сравним изображения, полученные двумя способами.

Далее сразу приведены изображения, полученные двумя способами.

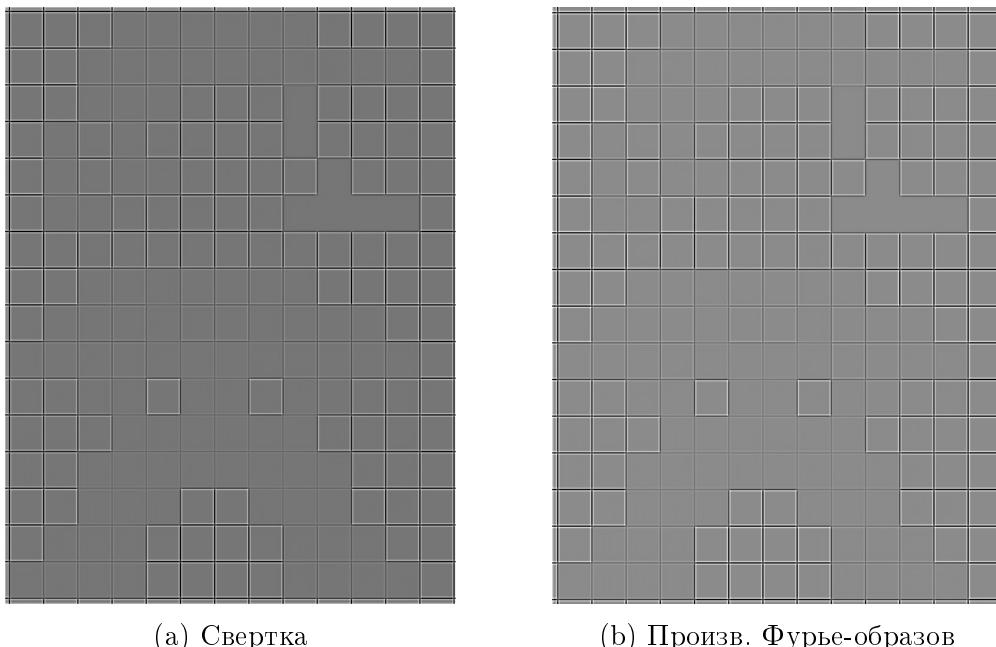


Рис. 11: Выделение краев сверткой и произведением Фурье-образов

Видим, что остались только выделенные края пикселей. В некоторых местах края четче, чем в других. Изображения получились одинаковыми, но одно светлее другого, что, скорее всего, вызвано ошибками в вычислениях. Следовательно, выполняется теорема о свертке.

4.1 Используемые программы

Для реализации данного задания потребовалось написать новые методы и воспользоваться некоторыми другими, приведенными ранее в листингах 1, 3, 5.

```
# Method to convert rgb image to gray image
def img2gray(img: Image):
    return img.convert('L')

# Method to edgen image with conv2
def edgen_conv2(img: Image, K = None):
    img2gray_ = img2gray(img)
    if K is None:
        K = [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]
    c2d = sps.convolve2d(img2gray_, K, mode='same')
    nc2d = (normalize(c2d)[0] * 255).astype(np.uint8)
    return nc2d

# Method to edgen image with fft2
def edgen_fft2(img: Image, K = None):
    img2gray_ = img2gray(img)
    if K is None:
        K = [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]
    w, h = img.size
    fft2_img = np.fft.fftshift(np.fft.fft2(img2gray_))
    fft2_ker = np.fft.fftshift(np.fft.fft2(K, s=(h, w)))
    img_ker = fft2_img * fft2_ker
    res = np.fft.ifft2(np.fft.ifftshift(img_ker)).real
    res = (normalize(res)[0] * 255).astype(np.uint8)
    return res
```

```
# Helper method for external use of edgen with conv2
def edgen_conv2_2img(img: Image, K = None):
    return convert_arr_to_img(edgen_conv2(img, K))

# Helper method for external use of edgen with fft2
def edgen_fft2_2img(img: Image, K = None):
    return convert_arr_to_img(edgen_fft2(img, K))
```

Листинг 7: Программные методы, необходимые для задания 4

Написанные методы были использованы в программе, приведенной ниже.

```
import img_utils as iu

# Specify the image source and render output
src = 'fm_lab6/src'
img_path = f'{src}/pa.png'
render_to = 'fm_lab6/renders/task4'

# Read image and convert to gray
img = iu.read_img(img_path)
gray = iu.img2gray(img)
iu.save_img(gray, f'{render_to}/img2gray.png')

# Rendering edgen image with conv2
ans = iu.edgen_conv2_2img(img)
iu.save_img(ans, f'{render_to}/edgen_conv2.png')

# Rendering edgen image with fft2
ans2 = iu.edgen_fft2_2img(img)
iu.save_img(ans2, f'{render_to}/edgen_fft2.png')
```

Листинг 8: Реализация задания 4