Лабораторная работа 3

Двумерное дискретное преобразование Хаара

Выполнил Казачинский Глеб, 3 курс 6 группа

Задание 0 (общее)

Написать функции, которые осуществляют многоуровневое двумерное вейвлет-преобразование (разложение и восстановление) на основе вейвлета Хаара. Можно рассмотреть только случай квадратной матрицы размерности 2^J, $J \in \mathbb{N}$. Провести сравнение с библиотечной функцией, которая реализована в используемом вами языке программирования.

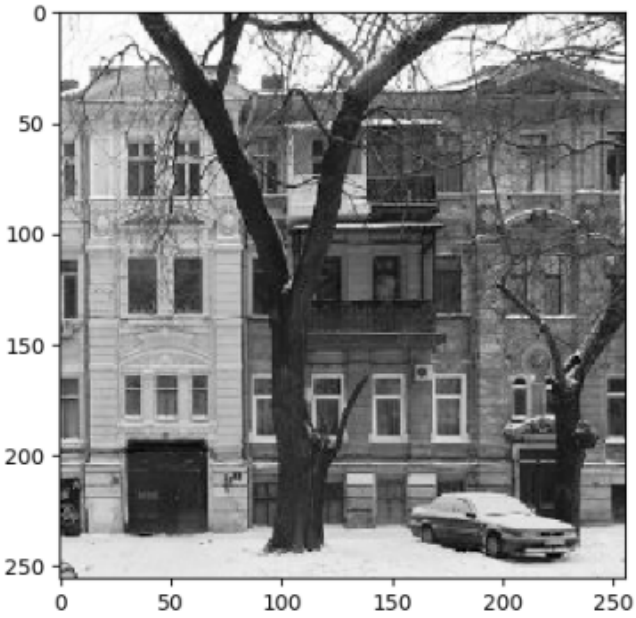
Вариант 2

1) Построить изображение, полученные путем обнуления в матрице D a) всех коэффициентов $d^{1г}$ $(j = \overline{1, J})$, b) всех коэффициентов $d^{1в}$, c) всех коэффициентов $d^{1д}$, и последующего вейвлет-восстановления.
2) Описать эффект, которые имеют преобразования из п. 1. Что получится, если обнулить сразу все три набора $d^{1\lambda}$? Почему?
3) Провести аналогичные эксперименты с коэффициентами $d^{j\lambda}$ для $j > 1$, привести соответствующие изображения, сделать выводы.
4) Какую полезную информацию могут нести наибольшие по модулю коэффициенты среди $d^{j\lambda}$ (при фиксированных j и λ)? Проведите эксперименты и подтвердите свои выводы соответствующими изображениями.

Содержание отчета

- Для каждого пункта задания — изображения и выводы.
- Исходный код всех программ.

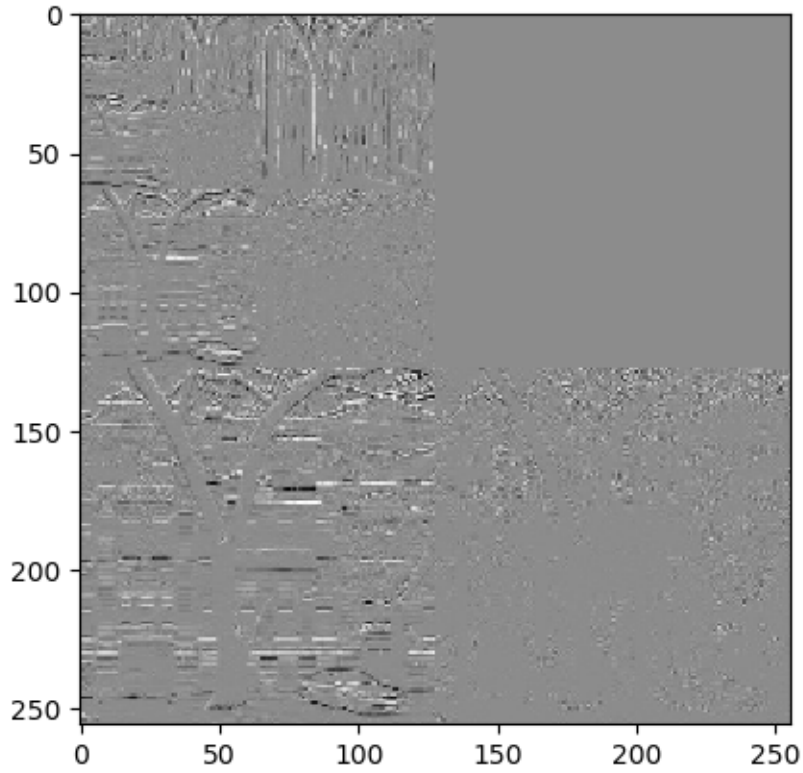
Исходное изображение 256x256:



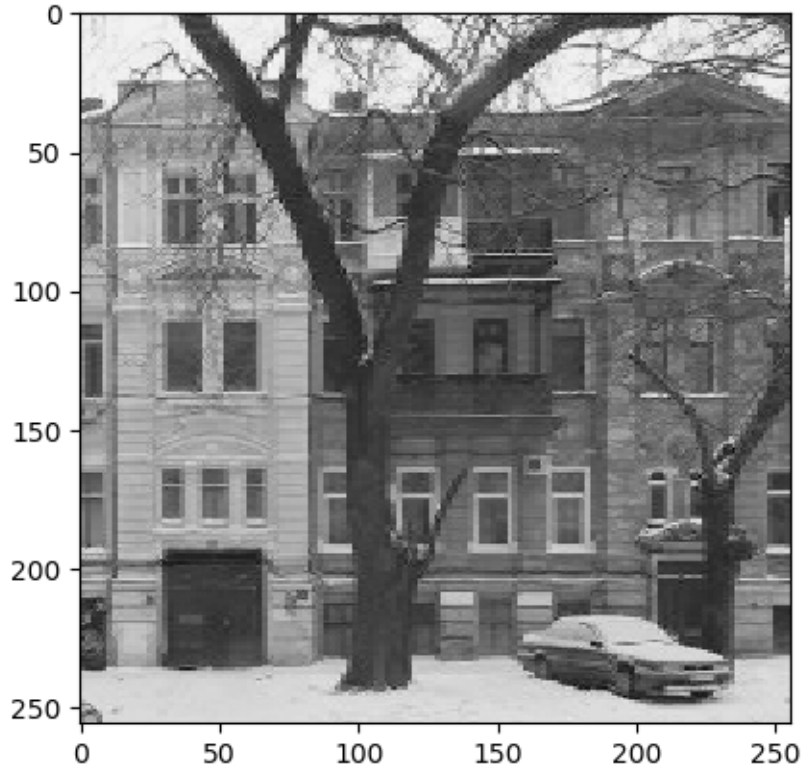
1)

a)

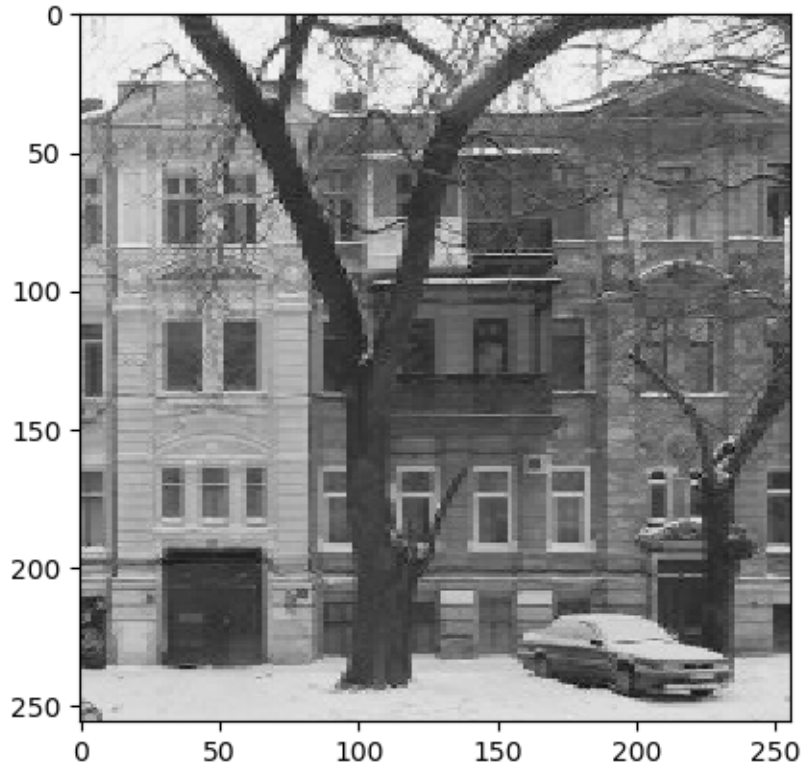
разложение $\tilde{C}$ и зануление соответсвующей матрицы d:



восстановление:



используя библиотечные разложение и восстановление:

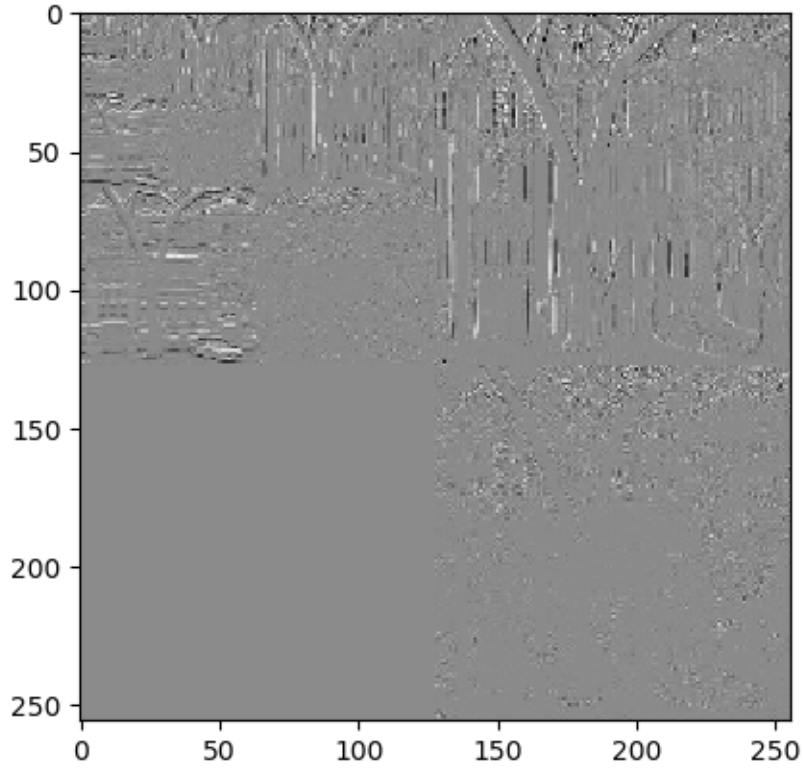


Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

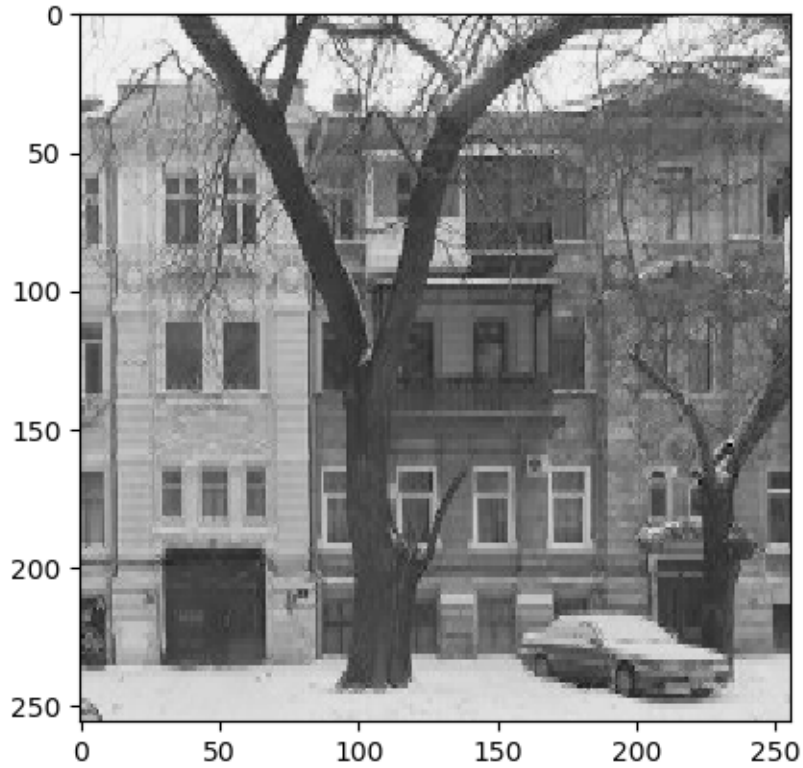
Посчитаем норму разности полученных изображений:

```
||dwt_i(C_) - dwt_i_true(C_true)|| a: 6.416193608158252e-11
```

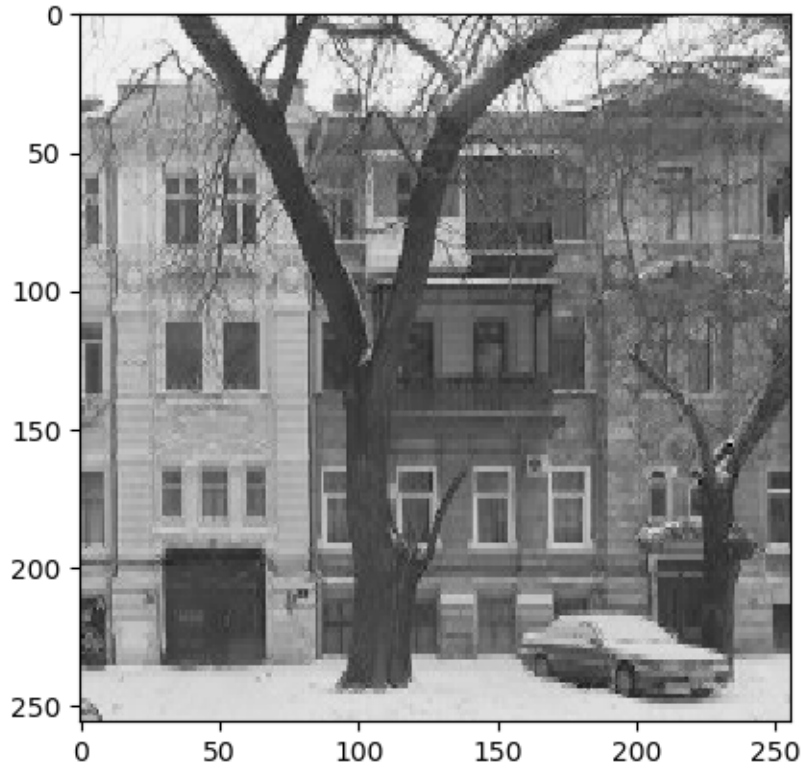
б) разложение $\tilde{C}$ и зануление соответсвующей матрицы d:



восстановление:



используя библиотечные разложение и восстановление:

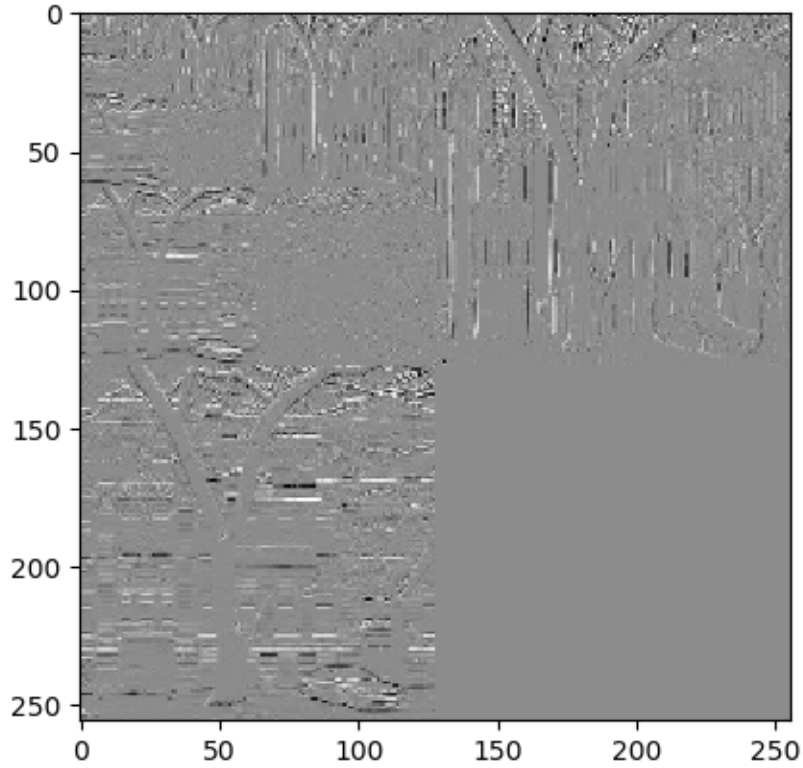


Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

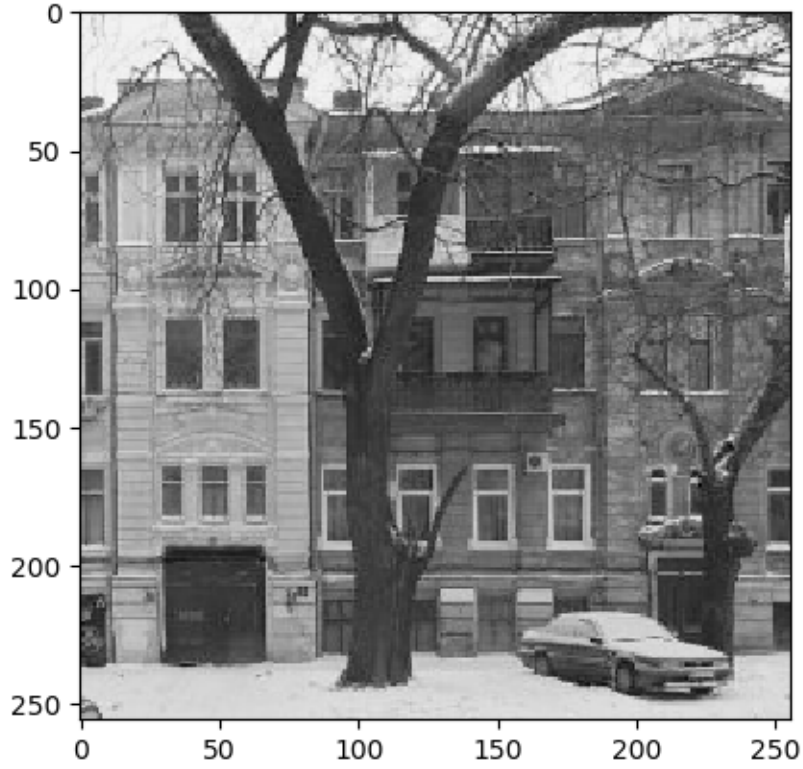
Посчитаем норму разности полученных изображений:

```
||dwt_i(C_) - dwt_i_true(C_true)|| b: 6.394113172614066e-11
```

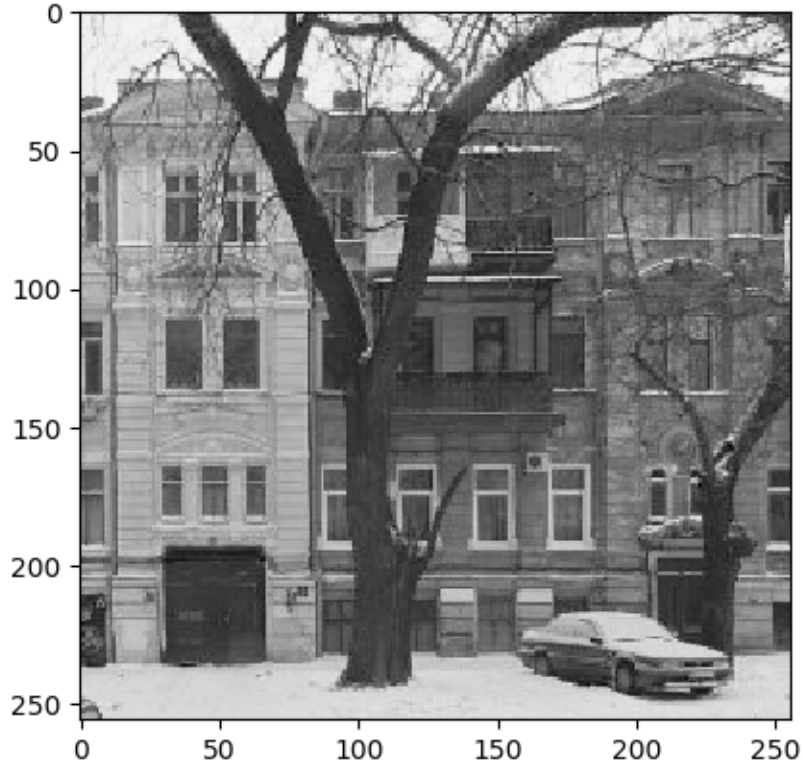
c) разложение $\tilde{C}$ и зануление соответсвующей матрицы d:



восстановление:



используя библиотечные разложение и восстановление:

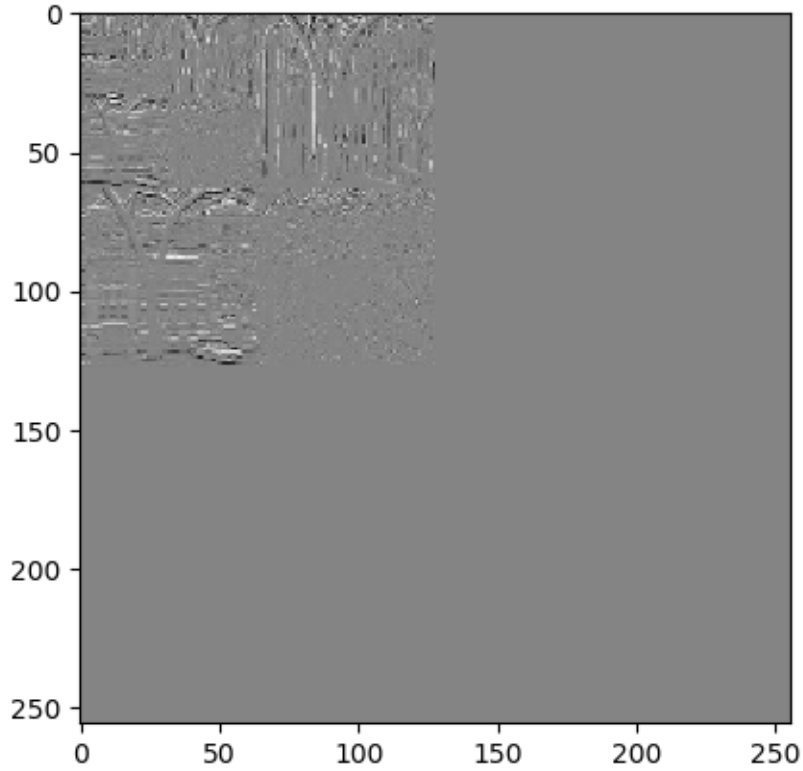


Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

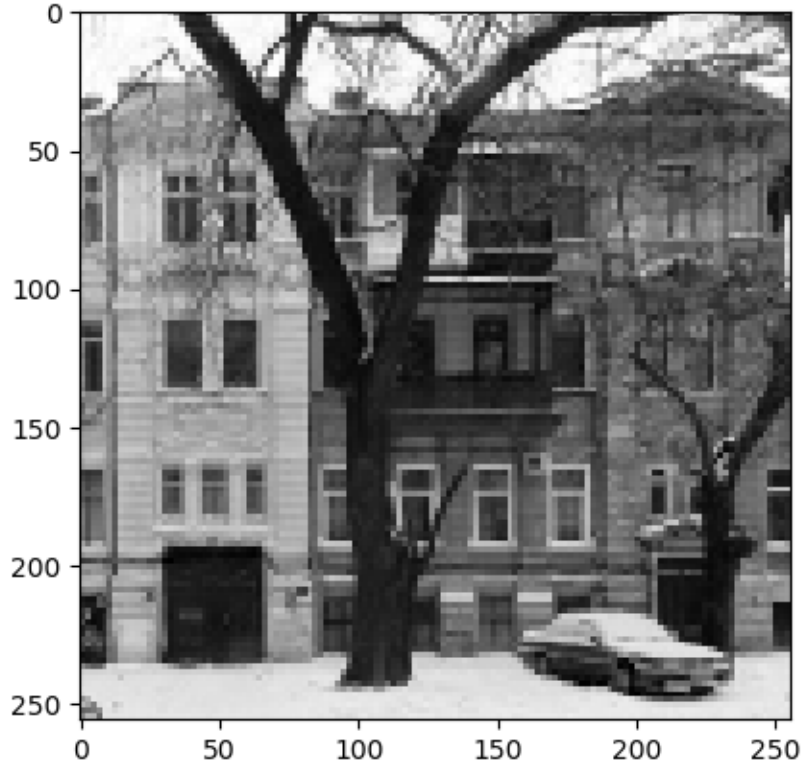
Посчитаем норму разности полученных изображений:

```
||dwt_i(C_) - dwt_i_true(C_true)|| c: 6.395198726443195e-11
```

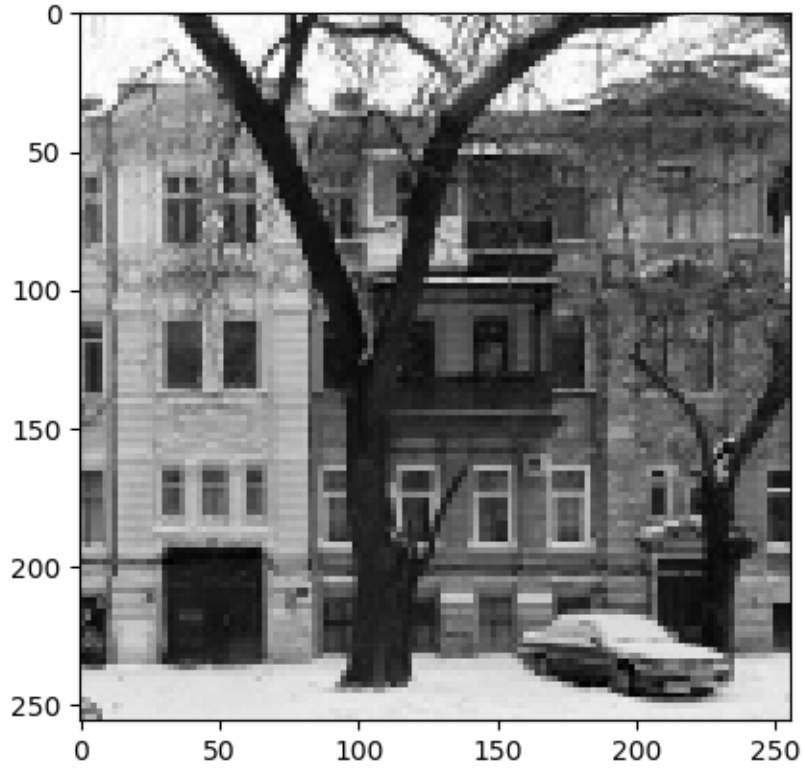
2) разложение $\tilde{C}$ и зануление всех 3-ёх матрицы d:



восстановление:



используя библиотечные разложение и восстановление:



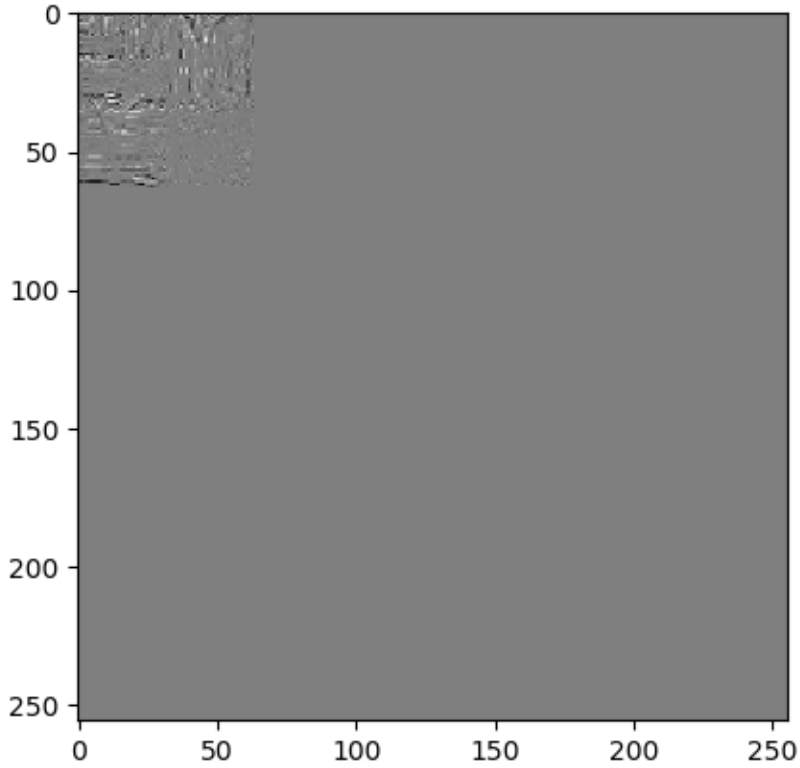
Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

Посчитаем норму разности полученных изображений:

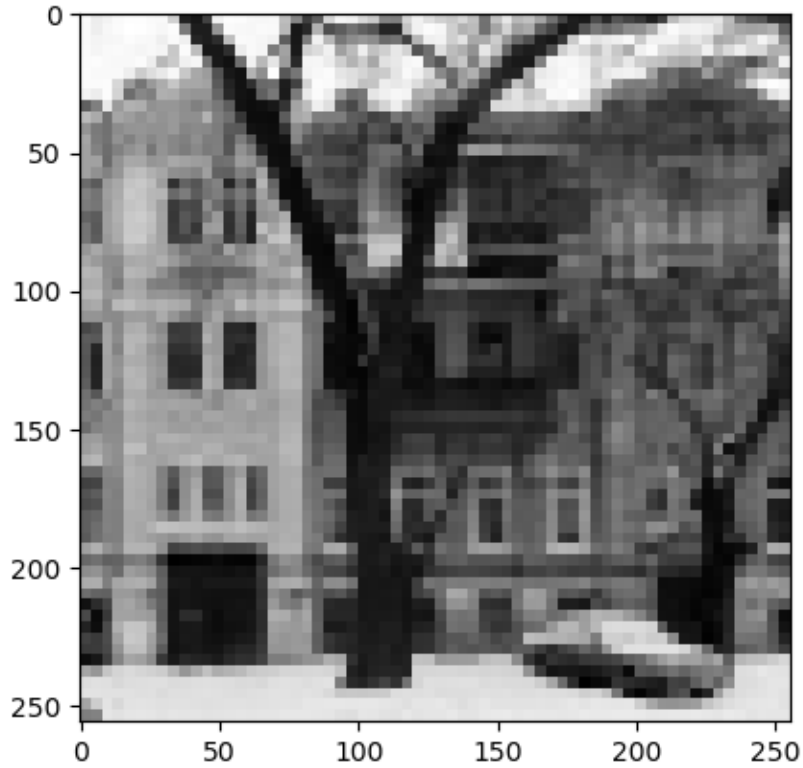
```
||dwt_i(C_) - dwt_i_true(C_true)|| task_2: 6.414470316016621e-11
```

Занулив все три набора $d^{j\lambda}$ (остаётся только матрица низкочастотных коэффициентов) получили более грубое почти в 4 раза уменьшеное изображение

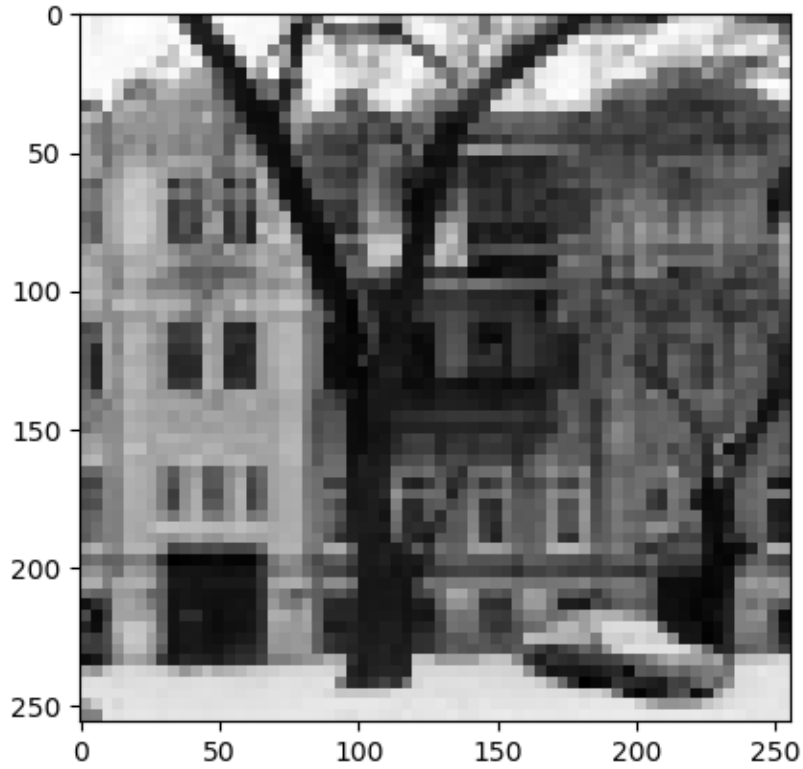
3) разложение $\tilde{C}$ и зануление всех $d^{1г}$, $d^{1в}$, $d^{1д}$, $d^{1г}$, $d^{1в}$, $d^{1д}$:



восстановление:



используя библиотечные разложение и восстановление:



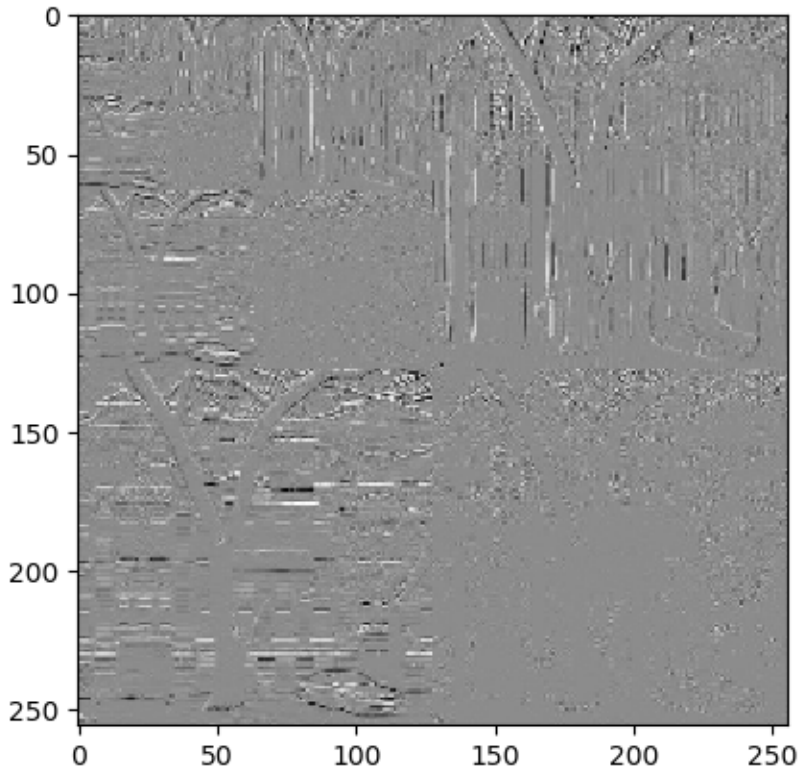
Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

Посчитаем норму разности полученных изображений:

```
||dwt_i(C_) - dwt_i_true(C_true)|| task_3: 6.357210410245638e-11
```

Занулив все 6 наборов $d^{j\lambda}$ получили ещё более сглаженное изображение

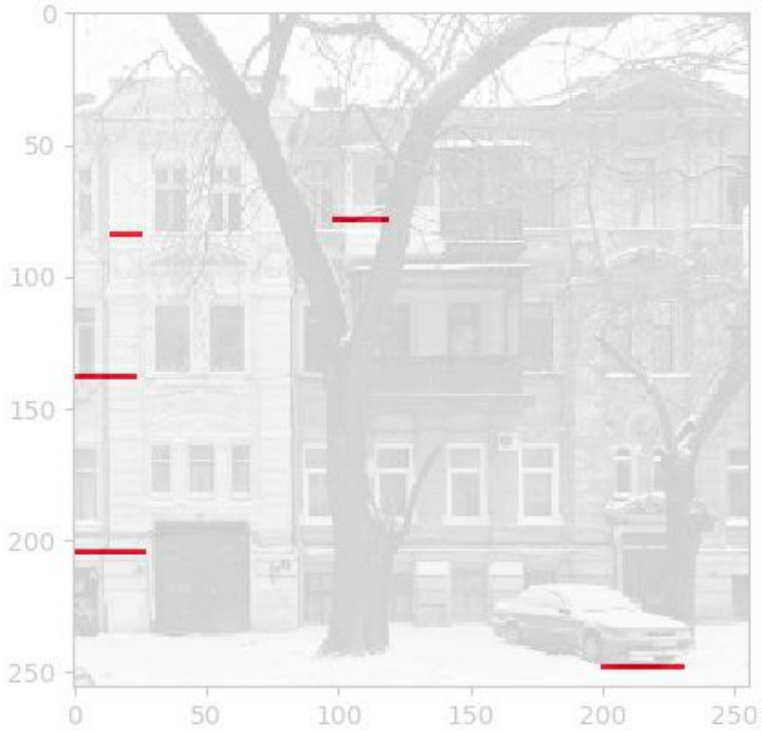
4) разложение $\tilde{C}$:



На картинке можно увидеть значимые резкие переходы например на 170-ой строке и т.д, занулим некоторые из них и сравним исходное изображение с изображением, восстановленым по матрице, в которой мы занулим указанные резкие переходы.

```
for j in range(7, 13):
    C_copy[170][j] = 0

for j in range(0, 12):
    C_copy[197][j] = 0

for j in range(0, 14):
    C_copy[230][j] = 0

for j in range(49, 60):
    C_copy[167][j] = 0

for j in range(100, 116):
    C_copy[252][j] = 0
```

Отличия плохо видны визуально, воспользуемся сторонней утилитой сравнения изображений:



Я занулял некоторые элементы по строкам, поэтому получил “сглаживание”, различия с исходным рисунком в местах, где резкие переходы “по вертикали”

Список всех полученных изображений с их размерами(весом):

C_.png	Сегодня, 22:54	122 КБ	PNG
source_black.png	Сегодня, 22:54	130 КБ	PNG
task_1_a_C_.png	Сегодня, 22:54	96 КБ	PNG
task_1_a_true.png	Сегодня, 22:54	118 КБ	PNG
task_1_a.png	Сегодня, 22:54	118 КБ	PNG
task_1_b_C_.png	Сегодня, 22:54	97 КБ	PNG
task_1_b_true.png	Сегодня, 22:54	115 КБ	PNG
task_1_b.png	Сегодня, 22:54	115 КБ	PNG
task_1_c_C_.png	Сегодня, 22:54	100 КБ	PNG
task_1_c_true.png	Сегодня, 22:54	117 КБ	PNG
task_1_c.png	Сегодня, 22:54	117 КБ	PNG
task_2_C_.png	Сегодня, 22:54	42 КБ	PNG
task_2_true.png	Сегодня, 22:54	42 КБ	PNG
task_2.png	Сегодня, 22:54	42 КБ	PNG
task_3_C_.png	Сегодня, 22:54	19 КБ	PNG
task_3_true.png	Сегодня, 22:54	20 КБ	PNG
task_3.png	Сегодня, 22:54	20 КБ	PNG
task_4_diff.jpg	Сегодня, 22:31	79 КБ	JPEG
task_4.png	Сегодня, 22:54	130 КБ	PNG

Код программы

```
1. import numpy as np
2. import pywt
3. import matplotlib.pyplot as plt
4. from skimage.color import rgb2gray
5. from numpy import linalg as LA
6.
7.
8. def save_img(C, name):
9.     plt.imshow(C, cmap='Greys_r')
10.     plt.savefig('compressed_img/{0}.png'.format(name), bbox_inches='tight')
11.
12.
13. def for_rows(C_, n):
14.     for i in range(n):
15.         s, m = [], []
16.         for k in range(0, n - 1, 2):
17.             s.append((C_[i][k] + C_[i][k + 1]) / 2)
18.             m.append((C_[i][k] - C_[i][k + 1]) / 2)
19.         C_[i, :n] = s + m
20.
21.
22. def for_columns(C_, n):
23.     for j in range(n):
24.         s, m = [], []
25.         for k in range(0, n - 1, 2):
26.             s.append((C_[k][j] + C_[k + 1][j]) / 2)
27.             m.append((C_[k][j] - C_[k + 1][j]) / 2)
28.         C_[:n, j] = s + m
29.
30.
```

```
31. def dwt(C):
32.     if len(C) != len(C[0]):
33.         raise ValueError('Wrong matrix dimensions')
34.     n = len(C)
35.     C_ = np.array(C.copy(), dtype=float)
36.     while n != 1:
37.         for_rows(C_, n)
38.         for_columns(C_, n)
39.         n = n // 2
40.     return C_
41.
42.
43. def for_rows_i(C, n, iteration):
44.     for i in range(n):
45.         s = []
46.         for k in range(0, n - iteration, 1):
47.             s.append(C[i][k] + C[i][k + iteration])
48.             s.append(C[i][k] - C[i][k + iteration])
49.         C[i, :n] = s
50.
51.
52. def for_columns_i(C, n, iteration):
53.     for j in range(n):
54.         s = []
55.         for k in range(0, n - iteration, 1):
56.             s.append(C[k][j] + C[k + iteration][j])
57.             s.append(C[k][j] - C[k + iteration][j])
58.         C[:n, j] = s
59.
60.
```

```
61. def dwt_i(C_):
62.     if len(C_) != len(C_[0]):
63.         raise ValueError('Wrong matrix dimensions')
64.     n = len(C_)
65.     C = np.array(C_.copy(), dtype=float)
66.     c = 2
67.     iteration = 1
68.     while c != n * 2:
69.         for_columns_i(C, c, iteration)
70.         for_rows_i(C, c, iteration)
71.         c *= 2
72.         iteration *= 2
73.     return C
74.
75.
76. def set_d_to_0(C, d, i):
77.     C_ = C.copy()
78.     n = C_.shape[0]
79.     c = 0
80.     while n is not 1 and c < i:
81.         if d == 'g':
82.             C_[:n // 2, n // 2:n] = 0
83.         if d == 'v':
84.             C_[n // 2:n, :n // 2] = 0
85.         if d == 'd':
86.             C_[n // 2:n, n // 2:n] = 0
87.         n = n // 2
88.         c += 1
89.     return C_
90.
91.
92. C = np.int32(
93.     rgb2gray(plt.imread(
94.         r'/Users/fpm.kazachin/PycharmProjects/wavelet_analysis/l3/256x256.jpg')) * 255)  # матрица из интенсивностей серого цвета
95.
96. save_img(C, 'source_black')
97. n = C.shape[0]
```

```
98. # tests:
99. # C = np.array([[0, 2, 1, 2],
100. #              [1, 1, 2, 0],
101. #              [0, 1, 2, 1],
102. #              [0, 2, 1, 2]])
103. # print('C = \n', C)
104. # C_ = dwt(C)
105. # print('C_ = \n', C_)
106. # print('dwt_i(C_) = \n', dwt_i(C_))
107.
108.
109. C_ = dwt(C)
110. save_img(C_, 'C_')
111. C_new = dwt_i(C_)
112. print('|dwt_i(C_) - C| =', LA.norm(C - C_new))
113.
114. C_true = pywt.wavedec2(C, 'haar')
115. C_new_true = pywt.waverec2(C_true, 'haar')
116. print('|C_new_true - C| =', LA.norm(C_new_true - C))
117.
118. # task 1
119. # a)
120. C_with_d_1_g_0 = set_d_to_0(C_, 'g', 1)
121. C_new = dwt_i(C_with_d_1_g_0)
122. save_img(C_with_d_1_g_0, 'task_1_a_C_')
123. save_img(C_new, 'task_1_a')
124.
125. # a) true
126. C_true_array, coeff_slices = pywt.coeffs_to_array(C_true)
127. C_true_array = set_d_to_0(C_true_array, 'g', 1)
128. C_new_true = pywt.waverec2(pywt.array_to_coeffs(C_true_array, coeff_slices,
output_format='wavedec2'), 'haar')
129. print('||dwt_i(C_) - dwt_i_true(C_true)|| a:', LA.norm(C_new_true - C_new))
130. save_img(C_new_true, 'task_1_a_true')
131.
132. # b)
133. C_with_d_1_v_0 = set_d_to_0(C_, 'v', 1)
134. C_new = dwt_i(C_with_d_1_v_0)
```

```
save_img(C_with_d_1_v_0, 'task_1_b_C_')
save_img(C_new, 'task_1_b')

# b) true
C_true_array, coeff_slices = pywt.coeffs_to_array(C_true)
C_true_array = set_d_to_0(C_true_array, 'v', 1)
C_new_true = pywt.waverec2(pywt.array_to_coeffs(C_true_array, coeff_slices, output_format='wavedec2'), 'haar')
print('||dwt_i(C_) - dwt_i_true(C_true)|| b:', LA.norm(C_new_true - C_new))
save_img(C_new_true, 'task_1_b_true')

# c)
C_with_d_1_d_0 = set_d_to_0(C_, 'd', 1)
C_new = dwt_i(C_with_d_1_d_0)
save_img(C_with_d_1_d_0, 'task_1_c_C_')
save_img(C_new, 'task_1_c')

# c) true
C_true_array, coeff_slices = pywt.coeffs_to_array(C_true)
C_true_array = set_d_to_0(C_true_array, 'd', 1)
C_new_true = pywt.waverec2(pywt.array_to_coeffs(C_true_array, coeff_slices, output_format='wavedec2'), 'haar')
print('||dwt_i(C_) - dwt_i_true(C_true)|| c:', LA.norm(C_new_true - C_new))
save_img(C_new_true, 'task_1_c_true')

# task 2
C_copy = C_.copy()
C_ = set_d_to_0(C_, 'g', 1)
C_ = set_d_to_0(C_, 'v', 1)
C_ = set_d_to_0(C_, 'd', 1)
C_new = dwt_i(C_)
save_img(C_, 'task_2_c_C_')
save_img(C_new, 'task_2')

# task 2 true
C_true_array, coeff_slices = pywt.coeffs_to_array(C_true)
C_true_array = set_d_to_0(C_true_array, 'g', 1)
C_true_array = set_d_to_0(C_true_array, 'v', 1)
```

```
171. C_true_array = set_d_to_0(C_true_array, 'd', 1)
172. C_new_true = pywt.waverec2(pywt.array_to_coeffs(C_true_array, coeff_slices,
output_format='wavedec2'), 'haar')
173. print('||dwt_i(C_) - dwt_i_true(C_true)|| task_2:', LA.norm(C_new_true - C_new))
174. save_img(C_new_true, 'task_2_true')
175.
176. # task 3
177. C_ = set_d_to_0(C_, 'g', 2)
178. C_ = set_d_to_0(C_, 'v', 2)
179. C_ = set_d_to_0(C_, 'd', 2)
180. C_new = dwt_i(C_)
181. save_img(C_, 'task_3_C_')
182. save_img(C_new, 'task_3')
183.
184. # task 3 true
185. C_true_array, coeff_slices = pywt.coeffs_to_array(C_true)
186. C_true_array = set_d_to_0(C_true_array, 'g', 1)
187. C_true_array = set_d_to_0(C_true_array, 'v', 1)
188. C_true_array = set_d_to_0(C_true_array, 'd', 1)
189. C_true_array = set_d_to_0(C_true_array, 'g', 2)
190. C_true_array = set_d_to_0(C_true_array, 'v', 2)
191. C_true_array = set_d_to_0(C_true_array, 'd', 2)
192. C_new_true = pywt.waverec2(pywt.array_to_coeffs(C_true_array, coeff_slices,
output_format='wavedec2'), 'haar')
193. print('||dwt_i(C_) - dwt_i_true(C_true)|| task_3:', LA.norm(C_new_true - C_new))
194. save_img(C_new_true, 'task_3_true')
195.
196. # task 4
197. for j in range(7, 13):
198.     C_copy[170][j] = 0
199.
200. for j in range(0, 12):
201.     C_copy[197][j] = 0
202.
203. for j in range(0, 14):
204.     C_copy[230][j] = 0
205.
206. for j in range(49, 60):
```

```
    C_copy[167][j] = 0

for j in range(100, 116):
    C_copy[252][j] = 0

C_new = dwt_i(C_copy)
save_img(C_new, 'task_4')
```