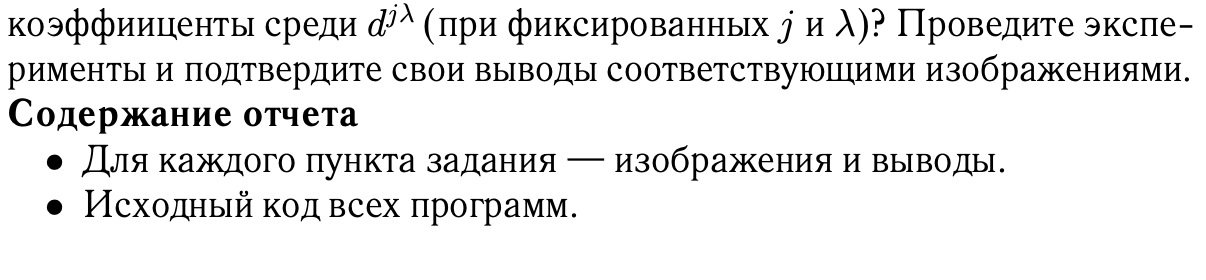
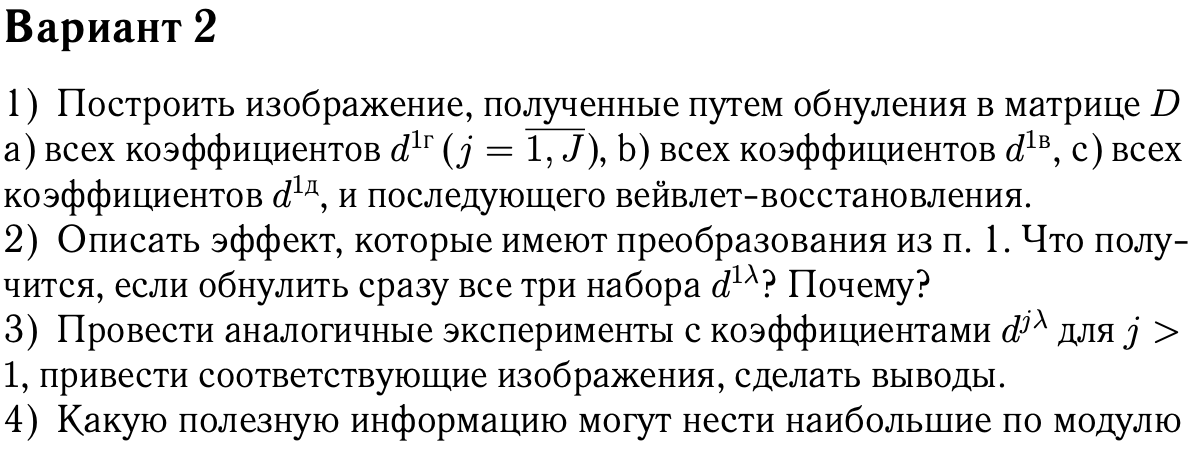
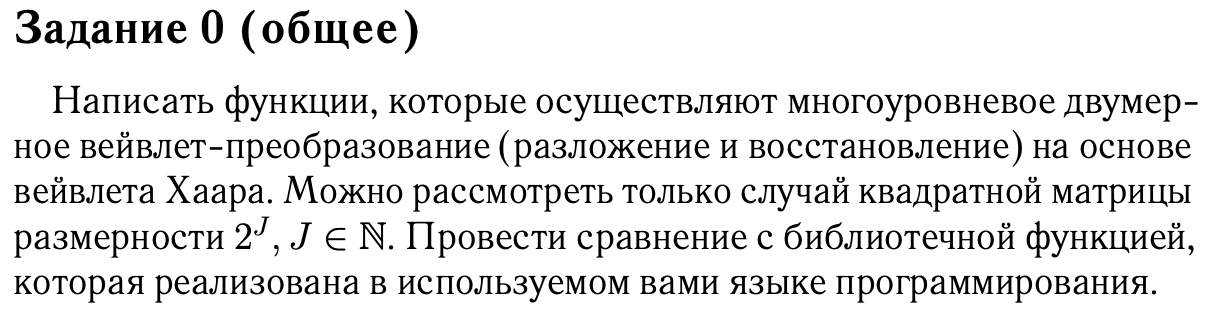
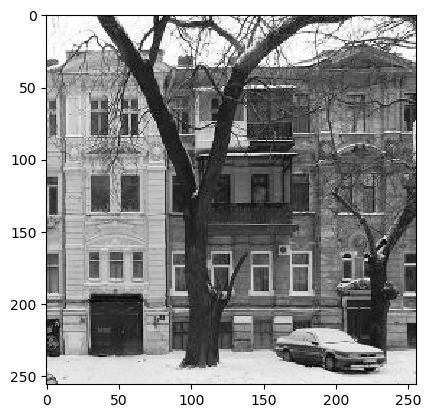
**Лабораторная работа 3**Двумерное дискретное преобразование Хаара

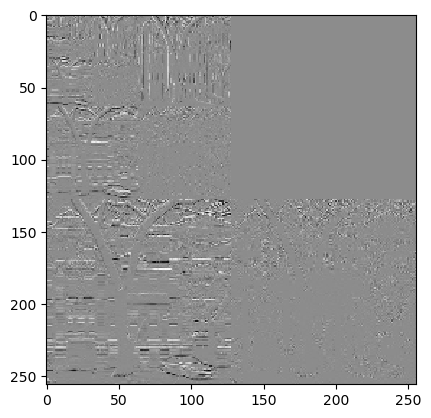
Выполнил Казачинский Глеб, 3 курс 6 группа

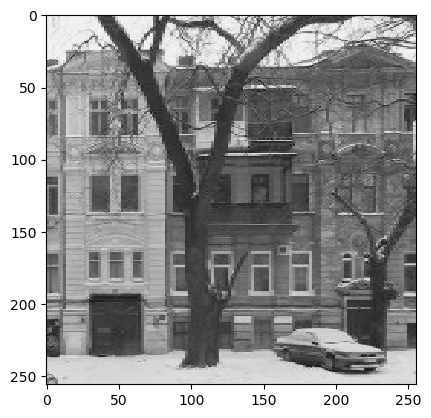


Исходное изображение 256х256:

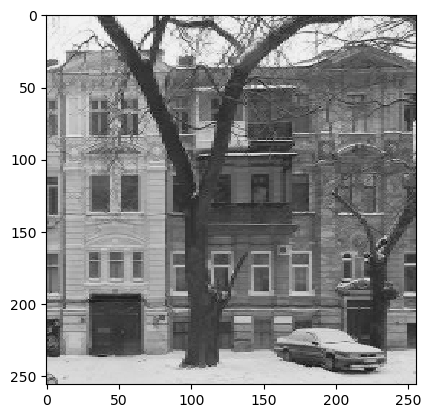


**1)  
a)**

разложение и зануление соответсвующей матрицы d:  


восстановление:  


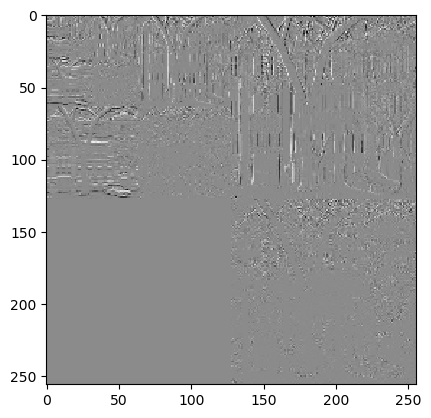
используя библиотечные разложение и восстановление:

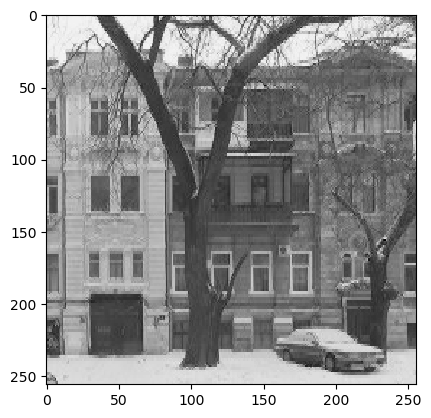


Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

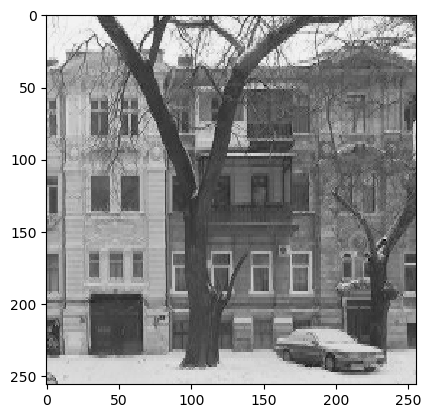
Посчитаем норму разности полученных изображений:



**б)** разложение и зануление соответсвующей матрицы d:  


восстановление:  


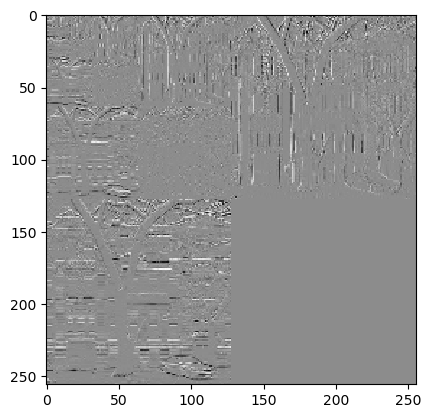
используя библиотечные разложение и восстановление:

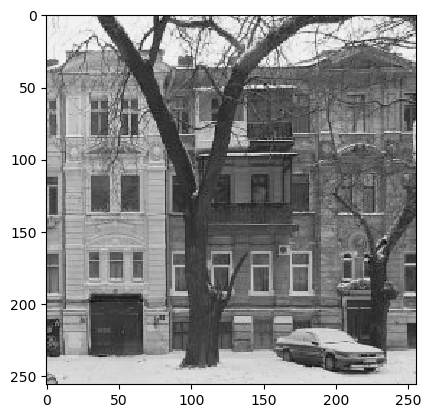


Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

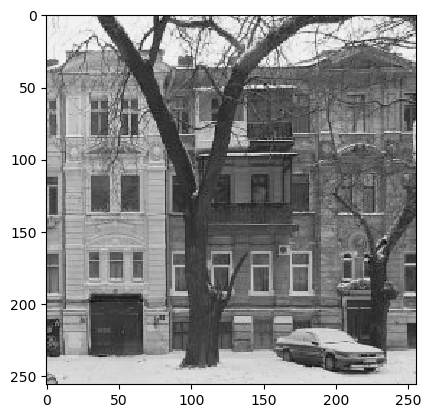
Посчитаем норму разности полученных изображений:



**с)** разложение и зануление соответсвующей матрицы d:  


восстановление:  


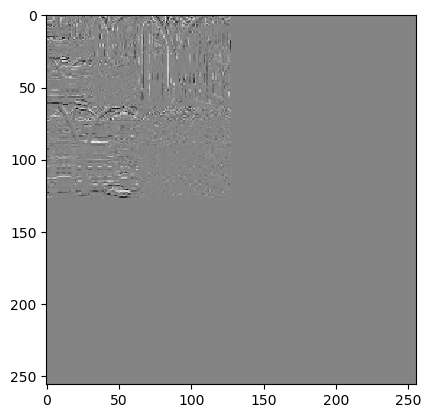
используя библиотечные разложение и восстановление:

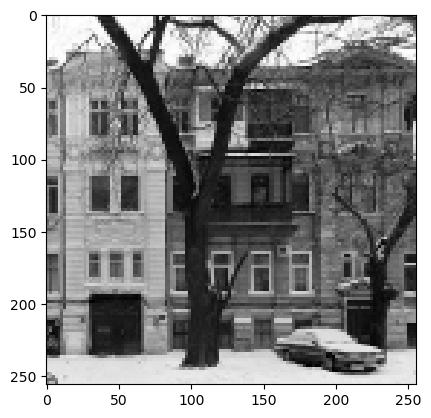


Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

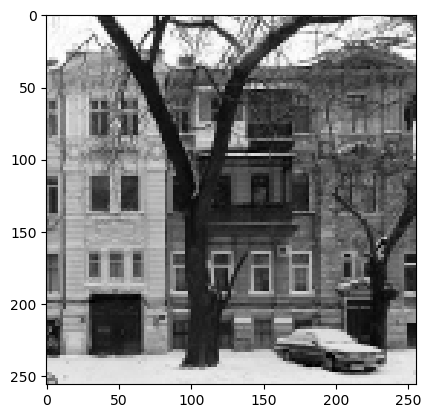
Посчитаем норму разности полученных изображений:



**2)** разложение и зануление всех 3-ёх матрицы d:  


восстановление:  


используя библиотечные разложение и восстановление:

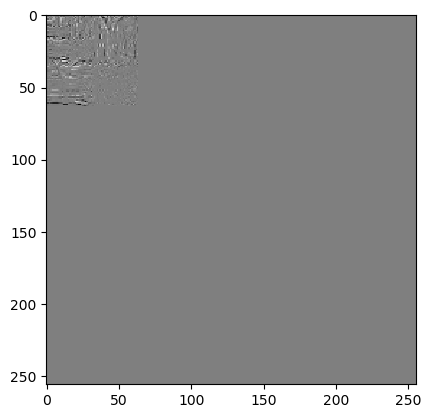


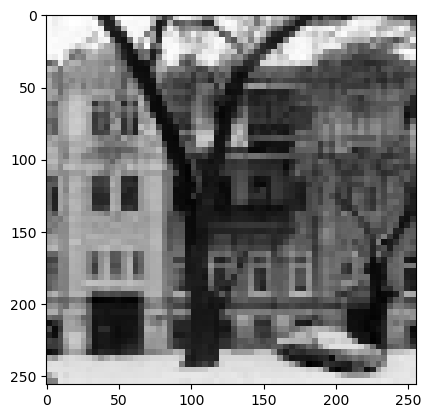
Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

Посчитаем норму разности полученных изображений:

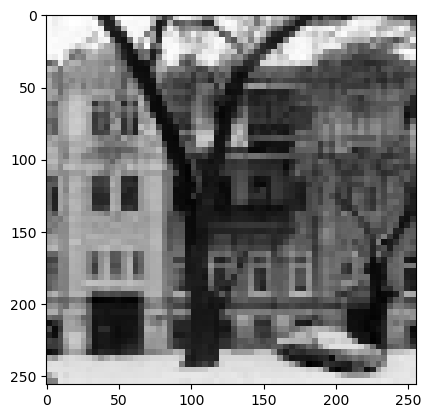


Занулив все три набора  (остаётся только матрица низкочастотных коэффициентов) получили более грубое почти в 4 раза уменьшеное изображение

**3)** разложение и зануление всех d1г, d1в, d1д, d1г, d1в, d1д:  


восстановление:  


используя библиотечные разложение и восстановление:



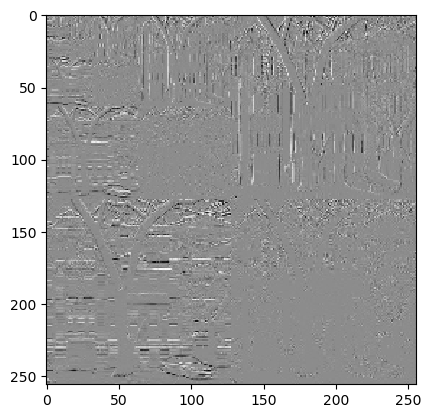
Получили одинаковые изображения используя запрограммированный алгоритм и разложение(pywt.wavedec2) и восстановление(pywt.waverec2)

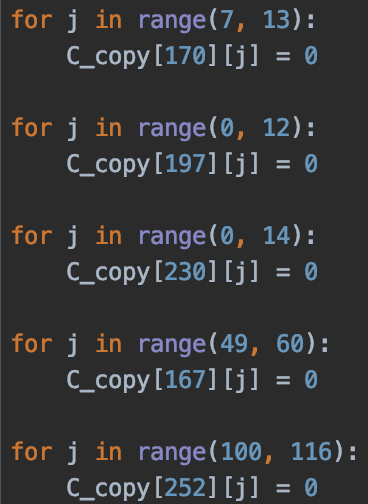
Посчитаем норму разности полученных изображений:

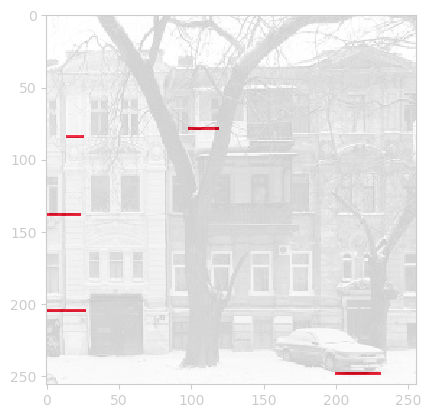


Занулив все 6 наборов  получили ещё более сглаженное изображение

**4)** разложение :



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.  
 

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):



**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\_, 'С\_')

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)

135. save\_img(C\_with\_d\_1\_v\_0, 'task\_1\_b\_C\_')

136. save\_img(C\_new, 'task\_1\_b')

137.

138. # b) true

139. C\_true\_array, coeff\_slices = pywt.coeffs\_to\_array(C\_true)

140. C\_true\_array = set\_d\_to\_0(C\_true\_array, 'v', 1)

141. C\_new\_true = pywt.waverec2(pywt.array\_to\_coeffs(C\_true\_array, coeff\_slices, output\_format='wavedec2'), 'haar')

142. **print**('||dwt\_i(C\_) - dwt\_i\_true(C\_true)|| b:', LA.norm(C\_new\_true - C\_new))

143. save\_img(C\_new\_true, 'task\_1\_b\_true')

144.

145. # c)

146. C\_with\_d\_1\_d\_0 = set\_d\_to\_0(C\_, 'd', 1)

147. C\_new = dwt\_i(C\_with\_d\_1\_d\_0)

148. save\_img(C\_with\_d\_1\_d\_0, 'task\_1\_c\_C\_')

149. save\_img(C\_new, 'task\_1\_c')

150.

151. # c) true

152. C\_true\_array, coeff\_slices = pywt.coeffs\_to\_array(C\_true)

153. C\_true\_array = set\_d\_to\_0(C\_true\_array, 'd', 1)

154. C\_new\_true = pywt.waverec2(pywt.array\_to\_coeffs(C\_true\_array, coeff\_slices, output\_format='wavedec2'), 'haar')

155. **print**('||dwt\_i(C\_) - dwt\_i\_true(C\_true)|| c:', LA.norm(C\_new\_true - C\_new))

156. save\_img(C\_new\_true, 'task\_1\_c\_true')

157.

158. # task 2

159. C\_copy = C\_.copy()

160. C\_ = set\_d\_to\_0(C\_, 'g', 1)

161. C\_ = set\_d\_to\_0(C\_, 'v', 1)

162. C\_ = set\_d\_to\_0(C\_, 'd', 1)

163. C\_new = dwt\_i(C\_)

164. save\_img(C\_, 'task\_2\_c\_C\_')

165. save\_img(C\_new, 'task\_2')

166.

167. # task 2 true

168. C\_true\_array, coeff\_slices = pywt.coeffs\_to\_array(C\_true)

169. C\_true\_array = set\_d\_to\_0(C\_true\_array, 'g', 1)

170. 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):

207. C\_copy[167][j] = 0

208.

209. **for** j **in** range(100, 116):

210. C\_copy[252][j] = 0

211.

212. C\_new = dwt\_i(C\_copy)

213. save\_img(C\_new, 'task\_4')