Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный университет имени М. В. Ломоносова»

Кафедра вычислительной механики

Отчёт по задаче на работу с изображениями по теме: Фрактальное сжатие изображений

Преподаватель: Почеревин Роман Владимирович
Студент 223 группы: Скворцов Андрей Сергеевич

Москва
2024

Отчёт по работе с BMP-изображениями в Python-3

Задание. Реализовать алгоритмы фрактального сжатия и восстановления изображения.
Решение. Используя библиотеки numpy и scipy, можно решить эту задачу проще. Кроме того используем библиотеки multiprocessing для распараллеливания программы:

1 Загрузка изображений

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy import ndimage
import numpy as np
from PIL import Image
import multiprocessing as mp
import datetime
```

Пусть изображения заданы внутри программы. Выгрузим его как черно-белое при помощи функции get_greyscale_image(img):

```
def get_greyscale_image(img):
        return np.mean(img[:,:,:2], 2)
```

Далее разобьем изображение на нужные нам блоки размера 4 на 4 вместо 8 на 8 при помощи функции reduce(img, factor):

```
def reduce(img, factor):
        result = np.zeros((img.shape[0] // factor, img.shape[1] // factor))
        for i in range(result.shape[0]):
                for j in range(result.shape[1]):
                        result[i, j] = np.mean(img[i * factor:(i + 1) * factor,
                        j * factor:(j + 1) * factor])

        return result
```

2 Алгоритм фрактального сжатия

2.1 Генерация преобразованных блоков

Теперь рассмотрим сам алгоритм фрактального сжатия изображения. Оно происходит при вызове функции compress. Сначала мы генерируем всевозможные блоки, которые мы можем получить при помощи нашего отображения, при помощи функции generate_all_transformed_blocks:

```
def generate_all_transformed_blocks(img, source_size_block,
destination_size_block, step):
        factor = source_size_block // destination_size_block
        transformed\_blocks = []
        for k in range((img.shape[0] - source_size_block) // step + 1):
                for l in range((img.shape[1] - source_size_block) // step + 1):
                        S = reduce(img[k * step:k * step + source_size_block,
                        l * step:l * step + source_size_block], factor)
                        for direction, angle in candidates:
                                transformed_blocks.append((k, l, direction,
                                angle, apply_transformation(S, direction, angle)

        return transformed_blocks
```

В этой функции мы используем функции поворота на угол кратный пи / 2
и отзеркаливание блока с последующим примением трансформации:

```
def rotate(img, angle):
        return ndimage.rotate(img, angle, reshape = False)

def flip(img, direction):
        return img[::direction,:]

def apply_transformation(img, direction, angle, contrast = 1.0,
brightness = 0.0):
        return contrast * rotate(flip(img, direction), angle) + brightness
```

2.2 Сжатие

Далее для каждого блока размером 8 на 8 ищем наиболее похожий на
него блок размера 4 на 4 и сохраняем параметры преобразования этого
блока, а также координаты исходного блока:

```
def compress(img, source_size_block, destination_size_block, step):
        transformations = []
        transformed_blocks = generate_all_transformed_blocks(img,
        source_size_block, destination_size_block, step)

        i_count = img.shape[0] // destination_size_block
        j_count = img.shape[1] // destination_size_block
        for i in range(i_count):
                transformations.append([])
                for j in range(j_count):
                        transformations[i].append(None)
                        min_d = float('inf')
```

```
                D = img[i * destination_size_block:
                (i + 1) * destination_size_block,
                j * destination_size_block:
                (j + 1) * destination_size_block]

                for k, l, direction, angle, S in transformed_blocks:
                        contrast, brightness =
                        find_contrast_and_brightness2(D, S)

                        S = contrast * S + brightness
                        d = np.sum(np.square(D - S))
                        if d < min_d:
                                min_d = d
                                transformations[i][j] = (k, l, direction
                                angle, contrast, brightness)

                                transformations_no_fratal[i][j] = S

        np.save('save_data_{}'.format(number), transformations_no_fratal)
        np.save('save_data_fractal_{}'.format(number), transformations)

        return transformations
```

Для наибольшей схожести блоков подбираем яркость и контрастность при помощи функции find_contrast_and_brightness2, причем схожесть определяется методом наименьших квадратов, реализованным в библиотеке numpy:

```
def find_contrast_and_brightness2(D, S):
        A = np.concatenate((np.ones((S.size, 1)),
        np.reshape(S, (S.size, 1))), axis = 1)

        b = np.reshape(D, (D.size,))
        x, _, _, _ = np.linalg.lstsq(A, b, rcond=None)

        return x[1], x[0]
```

2.3 Восстановление изображения после фрактального сжатия

Теперь рассмотрим алгоритм восстановления описанный в функции decompress. Самое полезное для нас в сжимающем отображении это наличие неподвижных точек в каждом блоке, поэтому для восстановления изображения нужно просто применить это отображение несколько раз к случайному изображению:

```
def decompress(transformations, source_size_block,
destination_size_block, step, nb_iter):

        factor = source_size_block // destination_size_block
        height = len(transformations) * destination_size_block
        width = len(transformations[0]) * destination_size_block
        iterations = [np.random.randint(0, 256, (height, width))]
        cur_img = np.zeros((height, width))
        for i_iter in range(nb_iter):
                for i in range(len(transformations)):
                        for j in range(len(transformations[i])):
                                k, l, flip, angle,
                                contrast, brightness = transformations[i][j]

                                k = int(k)
                                l = int(l)
                                flip = int(flip)
                                S = reduce(iterations[-1][k * step:
                                k * step + source_size_block,l * step:
                                l * step + source_size_block], factor)

                                D = apply_transformation(S, flip, angle,
                                contrast, brightness)

                                cur_img[i * destination_size_block:
                                (i + 1) * destination_size_block,
                                j * destination_size_block:
                                (j + 1) * destination_size_block] = D

                iterations.append(cur_img)
                cur_img = np.zeros((height, width))

        return iterations
```

Сохраним изображение:

```
        plt.imsave('result_{}.bmp'.format(number1),
        iterations[nb_iter - 1], cmap='gray')
```

2.4 Восстановление размеров изображения

После восстановления изображения мы получили его уменьшенную версию, по сравнению с исходным вариантом. Масштабируем его при помощи функции scale_image:

```
def scale_image(input_file, output_file, scale):
        try:
                img = Image.open(input_file)
                width, height = img.size
                new_width = width * scale
                new_height = height * scale
                new_img = Image.new('RGB',(new_width, new_height))

                for k in range(new_width):
                        for l in range(new_height):
                                i = k // scale
                                j = l // scale
                                pixel_sum = [0,0,0]
                                for x in range(2):
                                        for y in range(2):
                                                if i + x < width and j + y < hei
                                                  pixel = img.getpixel
                                                  ((i + x, j + y))

                                                  pixel_sum[0] += pixel[0]
                                                  pixel_sum[1] += pixel[1]
                                                  pixel_sum[2] += pixel[2]
                                pixel_avg=(pixel_sum[0] // 4, pixel_sum[1] // 4,
                                pixel_sum[2] // 4)

                                new_img.putpixel((k, l), pixel_avg)
                                new_img.save(output_file)
        except Exception as e:
                print(f"An error occurred: {e}")
```

3 Дополнительный способ сжатия

Помимо метода фрактального сжатия я рассмотрел вариант сохранения наиболее похожих блоков, для уменьшения потери качества изображения. Он реализуется вместе с методом фрактального сжатия и, в некотором смысле, идет параллельно ему.
Алгоритм восстановления прост: считывается NDarray и преобразуется в изображение

```
def decompress_ultra_mega_sposob(transformations,
source_size_block, destination_size_block):

        height = len(transformations) * destination_size_block
        width = len(transformations[0]) * destination_size_block
```

```
    cur_img = np.zeros((height, width))

    for i in range(len(transformations)):
            for j in range(len(transformations[i])):
                    cur_img[i * destination_size_block:
                    (i + 1) * destination_size_block,
                    j * destination_size_block:
                    (j + 1) * destination_size_block] = transformations[i][j

    return cur_img
```

4 Анализ подбора коэффициентов сжатия

Для меньшей потери качества было проведено исследрование на коэффициенты сжатия: какого размера брать ранговые и доменные блоки.
Я рассматривал сжатия 4 в 2, 6 в 3, 8 в 4 и 10 в 5 и получил следующие результаты(в данном лучае сжатие k в l означает, что мы превращаем блоки k * k в блоки l * l):

(a) Исходная маленькая картинка

(b) Исходная большая картинка

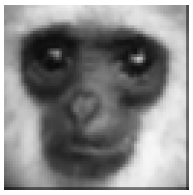
4.1 Малое изображение

(a) Фрактальное сжатие 4 в 2

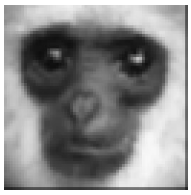
(b) Способ сохранения 4 в 2

(c) Фрактальное сжатие 6 в 3

(d) Способ сохранения 6 в 3



(a) Фрактальное сжатие 8 в 4



(b) Способ сохранения 8 в 4

(c) Фрактальное сжатие 10 в 5

(d) Способ сохранения 10 в 5

4.2 Большое изображение

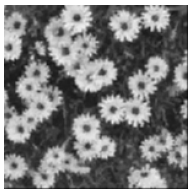
Из-за долгого времени работы алгоритма, на большом изображения такое маленькое разбиение не тестировалось.

(a) Фрактальное сжатие 6 в 3

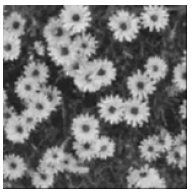
(b) Способ сохранения 6 в 3

(c) Фрактальное сжатие 8 в 4

(d) Способ сохранения 8 в 4



(a) Фрактальное сжатие 10 в 5



(b) Способ сохранения 10 в 5

4.3 Опрос

Для выявления лучших коэффициентов сжатия было опрошено 8 человек и получены следующие результаты:
Для маленького изображения:
Из результатов опроса видно, что для малых изображений лучше всего использовать сжатие 6 в 3, т.к. оно не сильно затратно по времени и имеет хорошую четкость. Для больших изображений лучше использовать

Обезьянка	Респондент 1	Респондент 2	Респондент 3	Респондент 4	Респондент 5	Респондент 6	Респондент 7	Респондент 8	Среднее	Время	Размер сжатый(Кб)	Исходный размер(Кб)
4 в 2 фрак	6,5	6	6	6	9	9	9	5	7,0625	0:14:09.624059	193	193
4 в 2 сохр	5	3	3	5	7	9	9	5,5	5,8125	0:14:09.624059	129	193
8 в 4 фрак	3,5	6	1	3	5	8	7	1	4,3125	0:00:03.945229	13	193
8 в 4 сохр	2	2	1	3	3	8	7	1,5	3,4375	0:00:03.945229	33	193
10 в 5 фрак	2,5	4	1	2	4	6	5	0,5	3,125	0:00:01.207121	5	193
10 в 5 сохр	1	3	1	2	3	6	5	1	2,75	0:00:01.207121	20	193
6 в 3 фрак	5	5	5	5	6	7	6	4	5,375	0:00:28.290035	37	193
6 в 3 сохр	3	3	2	4	5	7	6	4,5	4,3125	0:00:28.290035	56	193

(a) Маленькое изображение

Цветочки	Респондент 1	Респондент 2	Респондент 3	Респондент 4	Респондент 5	Респондент 6	Респондент 7	Респондент 8	Среднее	Время	Размер сжатый(Кб)	Исходный размер(Кб)
4 в 2 фрак	-	-	-	-	-	-	-	-	-	-	-	209
4 в 2 сохр	-	-	-	-	-	-	-	-	-	-	-	209
8 в 4 фрак	3,5	6	1	3	5	8	7	1	4,3125	0:03:39.547086	100	209
8 в 4 сохр	2	2	1	3	3	8	7	1,5	3,4375	0:03:39.547086	265	209
10 в 5 фрак	1,5	3	1	1	3	4	3	0,5	2,125	0:00:38.118574	43	209
10 в 5 сохр	1	2	1	1	2	4	3	1	1,875	0:00:38.118574	176	209
6 в 3 фрак	5	5	7	5	6	7	6	4	5,625	0:34:20.081393	324	209
6 в 3 сохр	3	3	6	4	6	7	6	4,5	4,9375	0:34:20.081393	485	209

(b) Большое изображение

сжатие 8 в 4, хотя при таком сжатия существенно падает качество изображения, но время затраченное на это не столь велико. Именно такие коэффициенты сжатия будем считать оптимальными.

5 Сжатие других изображений(цветных)

Рис. 7: Исходная картинка 3926 Кб

На данном изображении получились следующие результаты по времени(отсутствует результат 6 в 3 т.к. в программе выбор между максимальным сжатием/максимальным сохранением качества/оптимальное сжатие):
1. 10 в 5: 0:02:15.822430
2. 8 в 4: 0:12:21.176291

(a) Фрактальное сжатие 8 в 4

(b) Способ сохранения 8 в 4

(c) Фрактальное сжатие 10 в 5

(d) Способ сохранения 10 в 5