ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА"

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ

2 КУРС

Отчет по задаче №5. Фрактальное сжатие изображений

Работу выполнил: *Репин Михаил Денисович*
студент 224 группы
Семинарист: *Почеревин Роман Владимирович*
26 мая 2022 г.

Содержание

1 Условие задачи

Условие: Реализовать алгоритмы фрактального сжатия и восстановления изображения. Работать с черно-белыми изображениями.

2 Используемые библиотеки

Используются библиотеки: *imageio.v2* для открытия и записи изображений; *numpy* для удобной работы с числовыми массивами и выполнения математических операций; библиотека *matplotlib* для вывода в одном окне всех итераций распаковки изображения.

3 Алгоритм решения задачи и элементы кода

Первоначально считываем исходное изображение с помощью функции *imread* из библиотеки imageio.v2. Затем с помощью функций библиотеки *numpy* делаем изображенние черно-белым. Вызываем функцию *compress*, куда в качестве параметров передаем необходимые размеры областей разбиения (о них подробнее ниже), для удобства и быстроты операция берем их за степени двойки. Вводим разбиение матрицы изображения на постоянную сетку (так называемых ранговых квадратов), вводим разбиение на сетку перекрывающихся квадратов (так называемых доменных квадратов). С помщью функции *alltransform* генерируем всевозможные афинные преобразования для каждого из доменных блоков, предварительно сжимая их до размеров ранговых блоков. Затем, проходясь по разбиению по-

стоянной сеткой, для каждого из ранговых квадратов подбираем наиболее похожий преобразованный доменный квадрат с точки зрения метрики Фробениуса. Запоминаем координаты и коэффициенты преобразования, записываем их в массив сжатия. На этом процесс сжатия завершен.

Листинг 1: Код функции compress

```
def compress(img, dsize, rsize, step):
    com_array = []
    transformed_blocks = all_transform(img,dsize, rsize, step
        )
    i_count = img.shape[0] // rsize
    j_count = img.shape[1] // rsize
    f = open('res.bin', 'wb')
    f.write((str(i_count) + '\n').encode())
    f.write((str(j_count) + '\n').encode())
    for i in range(i_count):
        com_array.append([])
        for j in range(j_count):
            print("{}/{} ; {}/{}".format(i, i_count, j,
                j_count))
            com_array[i].append(None)
            min_d = float('inf')
            R = img[i*rsize:(i+1)*rsize,j*rsize:(j+1)*rsize]
            for k, l, val, angle, D in transformed_blocks:
                contrast, brightness = bright(R, D)
                D = contrast*D + brightness
                d = np.sum(np.square(R - D))
                if d < min_d:
                    min_d = d
                    com_array[i][j] = (k, l, val, angle,
                        contrast, brightness)
            for t in range(len(com_array[i][j])):
                if t != len(com_array[i][j])-1:
                    f.write((str(com_array[i][j][t])).encode
                        ())
                else:
                    f.write((str(com_array[i][j][t])+'\n').
                        encode())
    f.close()
    return com_array
```

Функция генерации всевозможных афинных преобразований для доменных квадратов:

Листинг 2: Функция alltransform

```
def all_transform(img, dsize, rsize, step):
    factor = dsize // rsize
    transformed_blocks = []
    for i in range((img.shape[0] - dsize) // step + 1):
        for j in range((img.shape[1] - dsize) // step + 1):
            cp_block = reduce(img[i*step:i*step+dsize, j*step
               :j*step+dsize], factor)
            for val, angle in candidates:
                transformed_blocks.append((i, j, val, angle,
                   make_transform(cp_block, val, angle)))
    return transformed_blocks
```

Алгоритм распаковки генерирует совершенно случайную картинку (в данном случае полностью черную). Затем, разбивая картинку на ранговые и доменные блоки, применяет соответствующее преобразование из сжатых данных для соответствующего доменного блока и вставляет получившийся кусок на место рангового блока. Такая система итерируемых функций применяется к картинке заданное число раз(как видно, 10 раз уже достаточно). В итоге получется исходная картинка, с допустимыми для данного алгоритма потерями.

Листинг 3: Распаковка изображения

```
def decompress(com_array, dsize, rsize, step, nb_iter=10):
    factor = dsize // rsize
    height = len(com_array) * rsize
    width = len(com_array[0]) * rsize
    iterations = [np.full((height, width), 1)]
    cur_img = np.zeros((height, width))
    for i_iter in range(nb_iter):
        print(i_iter)
        for i in range(len(com_array)):
            for j in range(len(com_array[i])):
                k, l, flip, angle, contrast, brightness =
                    com_array[i][j]
                D = reduce(iterations[-1][k*step:k*step+dsize
                    , l*step:l*step+dsize], factor)
                R = make_transform(D, flip, angle, contrast,
                    brightness)
                cur_img[i*rsize:(i+1)*rsize, j*rsize:(j+1)*
                    rsize] = R
        iterations.append(cur_img)
        cur_img = np.zeros((height, width))
    return iterations
```

4 Примеры работы алгоритма

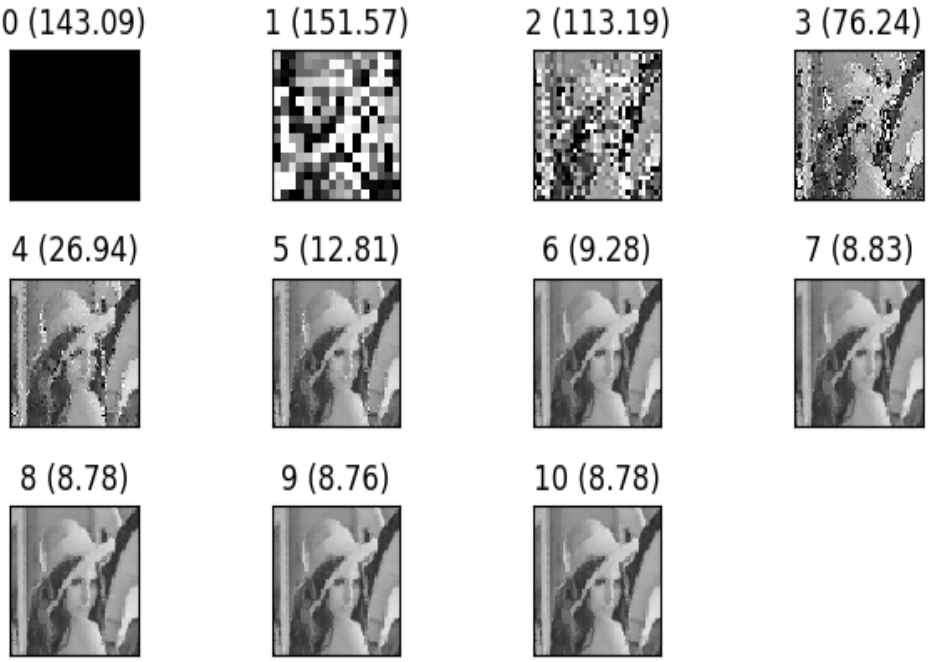

Рис. 1: Итерации распаковки изображения