МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Студент гр. 2301 _____ Комиссаров П.Е.

Преподаватель: _____ Пестерев Д.О.

Санкт-Петербург
2024

1. Экспериментальная часть с результатами исследования эффективности сжатия, как всего алгоритма в целом, так и отдельных частей

В лабораторной работе мы рассматривали сжатие фотографий с помощью алгоритма, состоящего из следующих этапов:

1. Преобразование в цветовое пространство YCbCr
2. Сабсэмплинг матриц с цветовыми каналами Cb и Cr
3. Дискретное косинусное преобразование
4. Квантование
5. Кодирование с помощью алгоритма RLE

Сжатие производилось на различных входных данных, таких как:

1. Фото размером 1600x1600
2. Фото, состоящее из пикселей случайного цвета
3. Фото, состоящее из пикселей одного цвета

В таблице 1 представлены коэффициенты сжатия для каждого этапа процесса с Q=50 (для квантования). (запись производилась в текстовые файлы)

Эти коэффициенты были рассчитаны путем сравнения размеров данных на начальном этапе с размерами на текущем этапе, что позволяет оценить вклад каждого этапа в общее сжатие данных.

В таблице 1.1 представлены размеры файлов для каждого этапа процесса с Q=50 (для квантования). (без текстовых файлов). Также сравнение коэффициентов DCT сравниваются с файлом с типом данных int16. (тк файл DCT может быть записан только в int16).

На таблице 2 представлены размеры каждого из полученных файлов в КБ

Таблица 1

	begin	YCbCr	subsempling	DCT	quantation	zigzag	RLE
Image	25579	28539	5186	4310	2988	2988	1459
Random	27565	28993	5188	5352	3756	3756	6581
Solid	20000	27504	5179	3002	2839	2839	353

Таблица 2

	begin	YCbCr	subsempling	DCT	quantation	zigzag	RLE
Image	1	0.896	4.932	5.935	8.560	8.560	17.532
Random	1	0.948	5.313	5.150	7.339	7.339	4.186
Solid	1	0.727	3.862	6.664	7.047	7.047	56.657

Таблица 1.1

	Begin (uint8)	Begin (int16)	YCbCr (uint8)	Subsampling (uint8)	DCT (int16)	Quantation (int16)	Zigzag (int16)	RLE
Image	7500	15000	7500	3750	7500	7500	7500	1458
Random	7500	15000	7500	3750	7500	7500	7500	6581
Solid	7500	15000	7500	3750	7500	7500	7500	352

Таблица 2.1

	Begin (uint8)	Begin (int16)	YCbCr (uint8)	Subsampl. (uint8)	DCT (int16)	Quantation (int16)	Zigzag (int16)	RLE
Image	1	1	1	2	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	5.144
Random	1	1	1	2	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	1.140
Solid	1	1	1	2	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	$1_{(c\ int8)}$ $/2_{(c\ int16)}$	21.307

Из таблицы 1 видно, что после преобразования фото в цветовое пространство YCbCr размер фото увеличился в ~ 0.9 раз.

После сабсемплинга каналов Cb и Cr, файлы сильно сократились в своем размере. Это вызвано тем, что матрицы каналов Cb, Cr уменьшились в 4 раза.

DCT сжал файлы (1 таблица), так как большинство значений массива после преобразования стали равны 0, либо маленькому значению

Квантование также сжало изображения (1 таблица), так как мы просто уменьшили размер матриц коэффициентов DCT

Обход зиг-загом не изменил размер фото, так как мы просто перевели

данные массивов в строку

RLE сильно сжал фото, так как мы за прошлые этапы увеличили кол-во повторяющихся друг за другом символов.

Сравнение фото до и после кодирования. Даунсемплинг = 2, Q = 50. Рисунок 1

Из таблицы 1.1 и 2.1 можно заметить, что размер файла изменился только после сабсемплинга и рле. После сабсеплинга размер стал меньше в 2 раза, так как 2 из 3 каналов уменьшились в 4 раза.

Кроме того, после DCT размер стал больше. Причиной этого является другой тип данных, так как коэффициенты могут быть отрицательными и быть больше типа данных int8, поэтому они записаны в типе данных int16, который в 2 раза больше типа данных int8

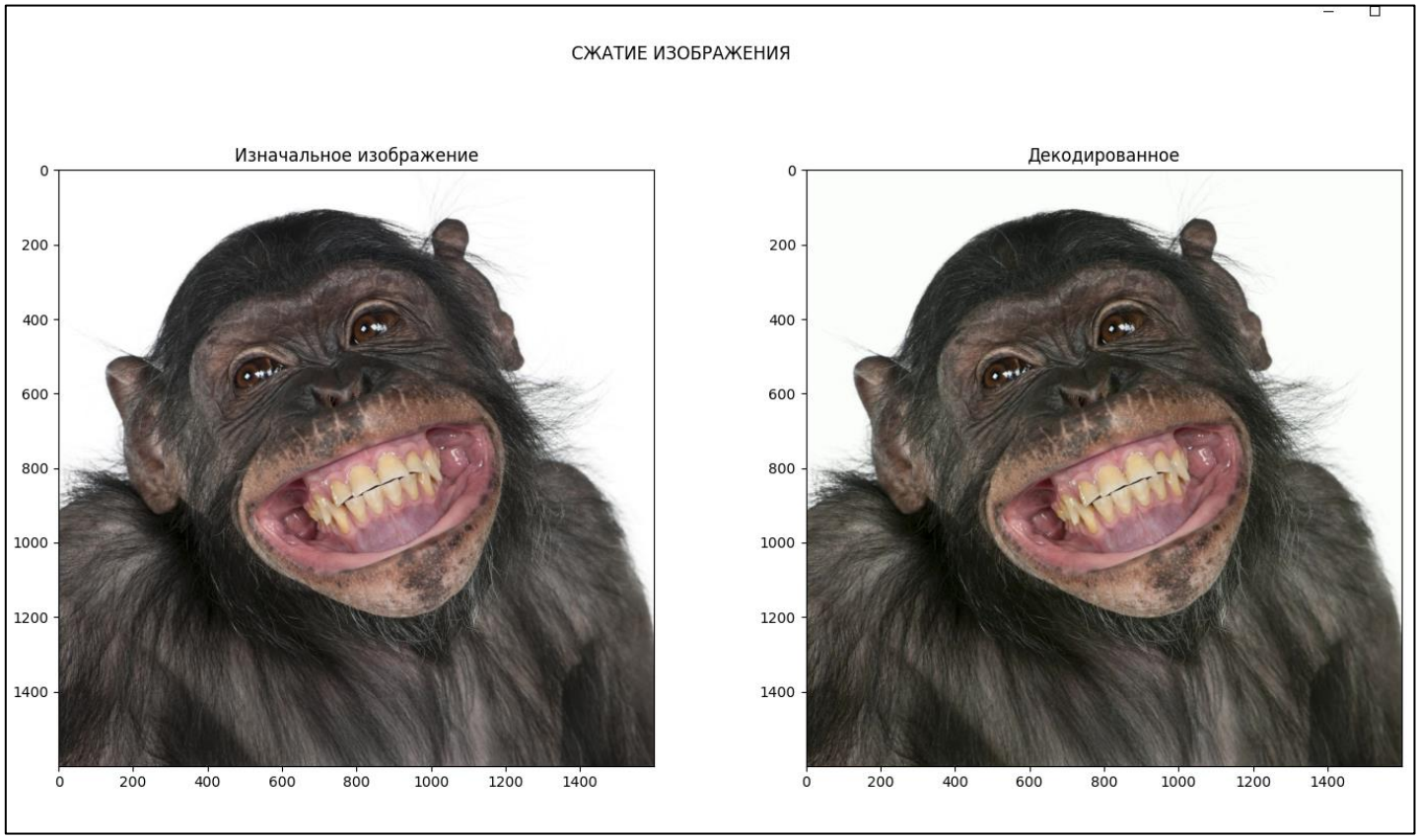

Рисунок 1 – Сравнение фото

На таблице 3 представлены коэффициенты сжатия для следующего метода сабсемплинга: сначала делаем даунсемплинг, после чего сразу делаем апсемплинг. Получились такие результаты (Таблица 3) (на самом деле я просто ошибся в реализации алгоритма, но таблицу уже сделал, поэтому решил оставить,

как + эксперимент😊)

Таблица 3

	begin	YCbCr	subsempling	DCT	quantation	zigzag	RLE
Image	1	0.896	0.896	1.606	1.678	1.678	14.625
Random	1	0.948	0.944	1.470	1.741	1.741	4.752
Solid	1	0.727	0.737	1.306	1.323	1.323	28.329

Сабсемплинг не оказал влияния на размер изображений в первом и третьем файлах, так как процесс включает в себя сначала даунсемплинг, который вычисляет среднее значение для блока пикселей в цветовом пространстве Cb, Cr, а затем апсемплинг, который заменяет все пиксели в этом блоке на вычисленное среднее. В случае изображений с однородным цветом и обычной картинке изменения в цвете происходят плавно, что не приводит к изменению размера. Однако в изображении с случайными цветами пикселей среднее значение может быть выше, что приводит к замене пикселей с низкими значениями на более высокие, что увеличивает общий размер изображения

Также я проверил, что будет, если произвести сжатие, исключая какой-либо из этапов. Результаты приведены в таблице 4 (коэффициенты сжатия в итоге). Сверху записан этап, исключенный из сжатия.

Таблица 4

	YCbCr	subsempling	DCT	quantation	-
Image	22.636	14.274	5.096	7.287	17.532
Random	8.457	3.025	5.124	3.172	4.186
Solid	338.983	28.329	4.942	48.543	56.657

Из таблицы видно, что каждый из этапов сжатия важен для качественного сжатия любого из файлов.

Также можно заметить, что если убрать этап с преобразованием из RGB в YCbCr размер становится меньше, но без этого этапа на фото появляется много артефактов. Рисунок 2.

Также без ДКТ фото теряет свое качество. Это вызвано тем, что после квантования появляется очень большая погрешность. (Так как квантация должна применяться к коэффициентам ДКТ) Рисунок 3.

Если убрать subsampling, то сжатие будет чуть менее эффективным.

Самый большой вес в сжатии несет DCT, если его убрать, то сжатие всех фото будет уменьшено в несколько раз. Это вызвано тем, что с помощью DCT можно занулить большинство значений матрицы.

Если убрать квантирование, сжатие также становится хуже для всех трех файлов.

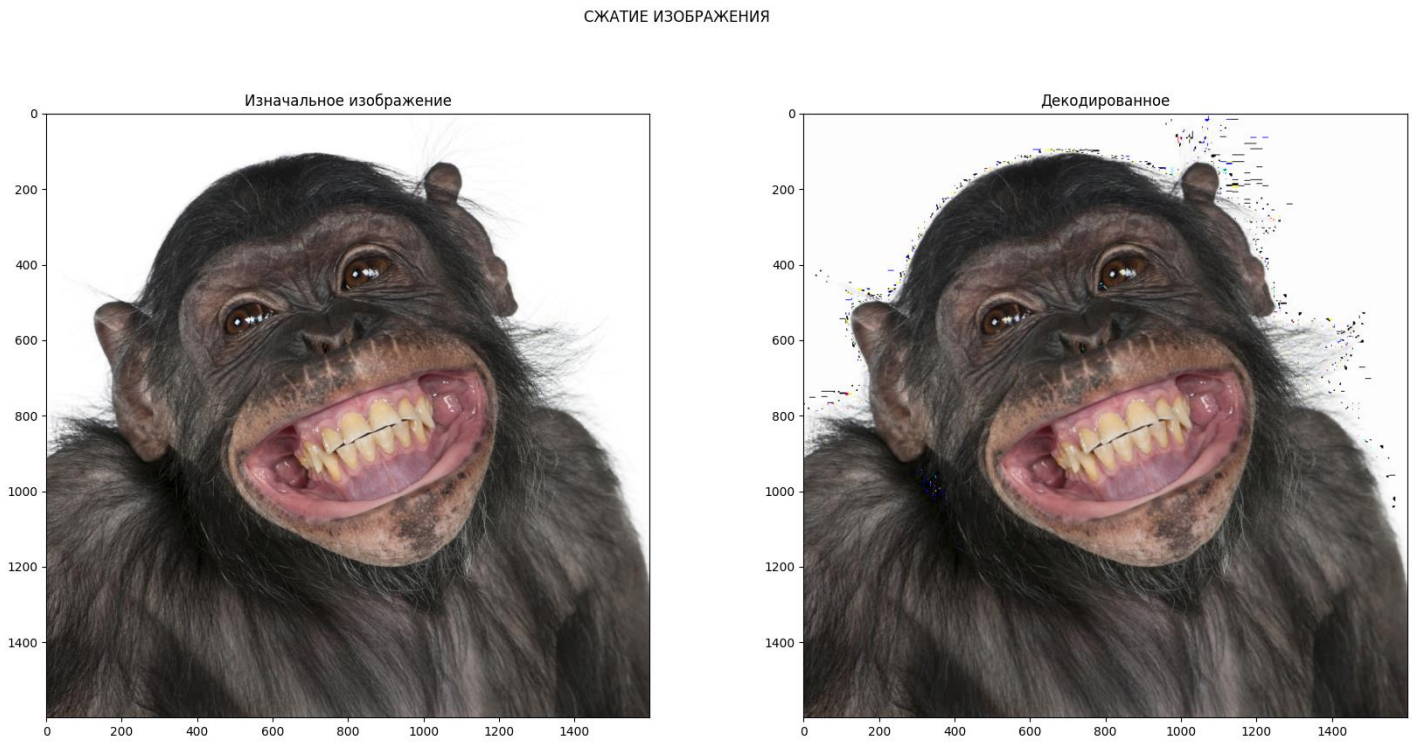

Рисунок 2 – Артефакты на фото

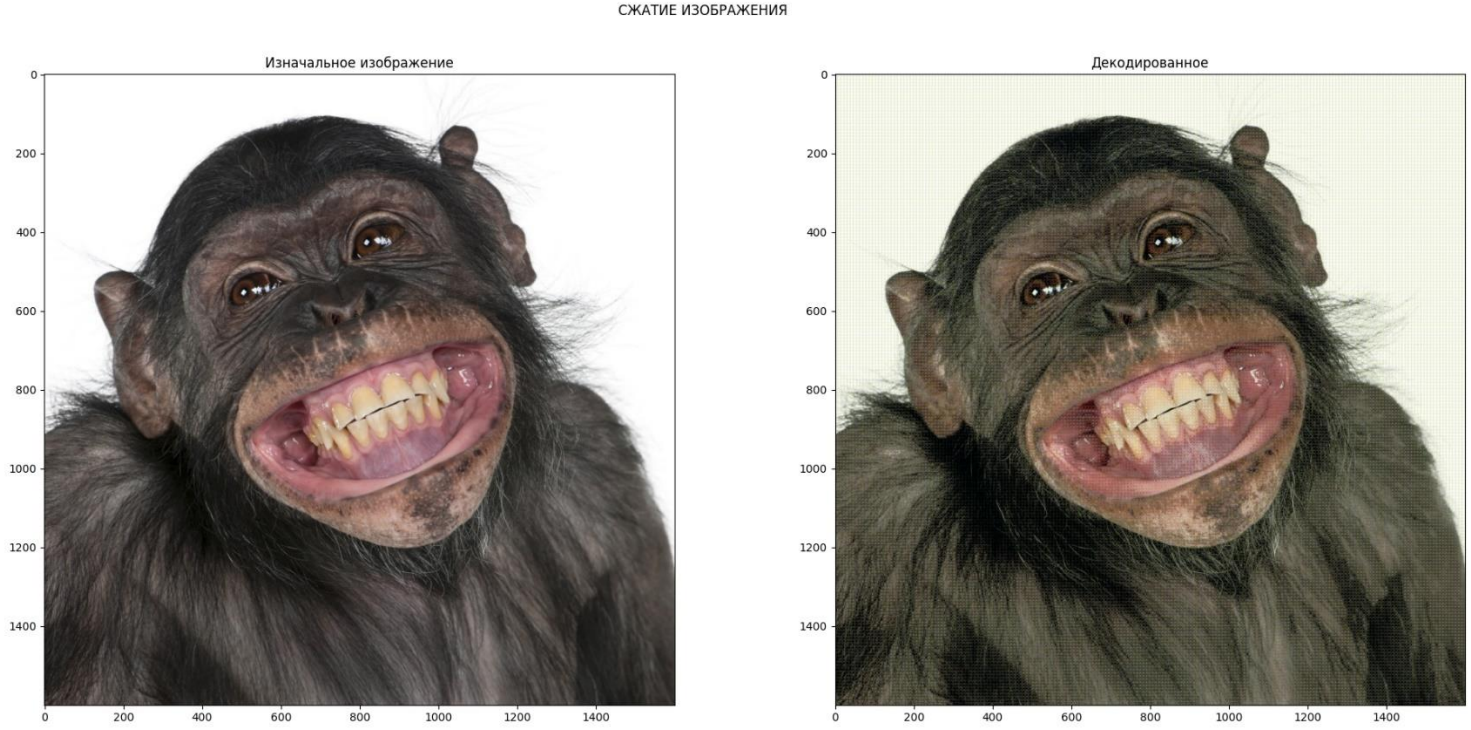

Рисунок 3 – Без ДКТ

Следующее, что я проверил – связь между коэффициентом качества квантирования и размером файла на выходе. Результаты сжатия приведены в таблице 4 и рисунке 2.

Таблица 4

	Картинка	Случайное	Заливка
95	7.924	3.230	53.619
85	9.834	3.411	53.619
75	12.632	3.545	55.096
65	14.517	3.646	55.096
55	16.513	3.779	55.096
50	17.532	4.187	56.657
45	18.323	4.556	56.657
35	20.779	4.623	56.657
25	25.102	5.646	56.657
15	33.480	8.458	56.657
10	42.846	11.912	56.657

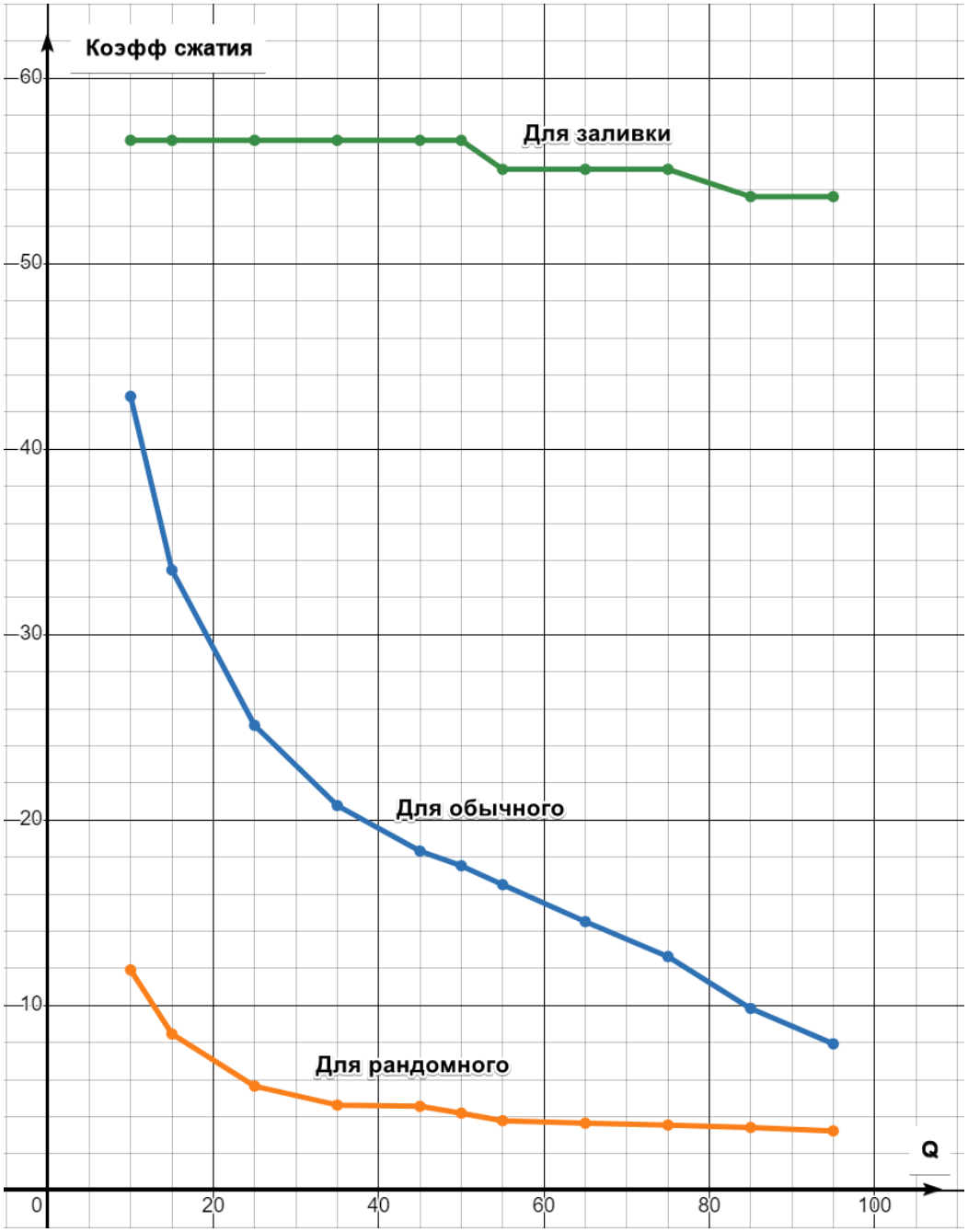

Рисунок 4 – График сжатия от параметра Q

Из рисунка видно, что коэффициент сжатия больше всего меняется на промежутке Q=[10;35], на промежутке Q=[40;55] коэффициент слабо колеблется, а на остальном промежутке Q=[55;95] коэффициент снова начинает меняться, но с меньшей скоростью.

Также можно заметить, что Q сильнее всего влияет на коэффициент сжатия для обычного изображения.

Но нужно учитывать, что при изменении Q качество фото меняется. Можно, что качество фото для Q=[95;50] слабо заметно. А далее, соответствуя динамике графика, качество сильно ухудшается. Для доказательства представлены следующие изображения

Также можно заметить, что качество от 100-60 практически не меняется. Поэтому по умолчанию q=50, тк с 40 уже виднеется ухудшение качества фото

Изменение обычного фото при изменении качества Q:

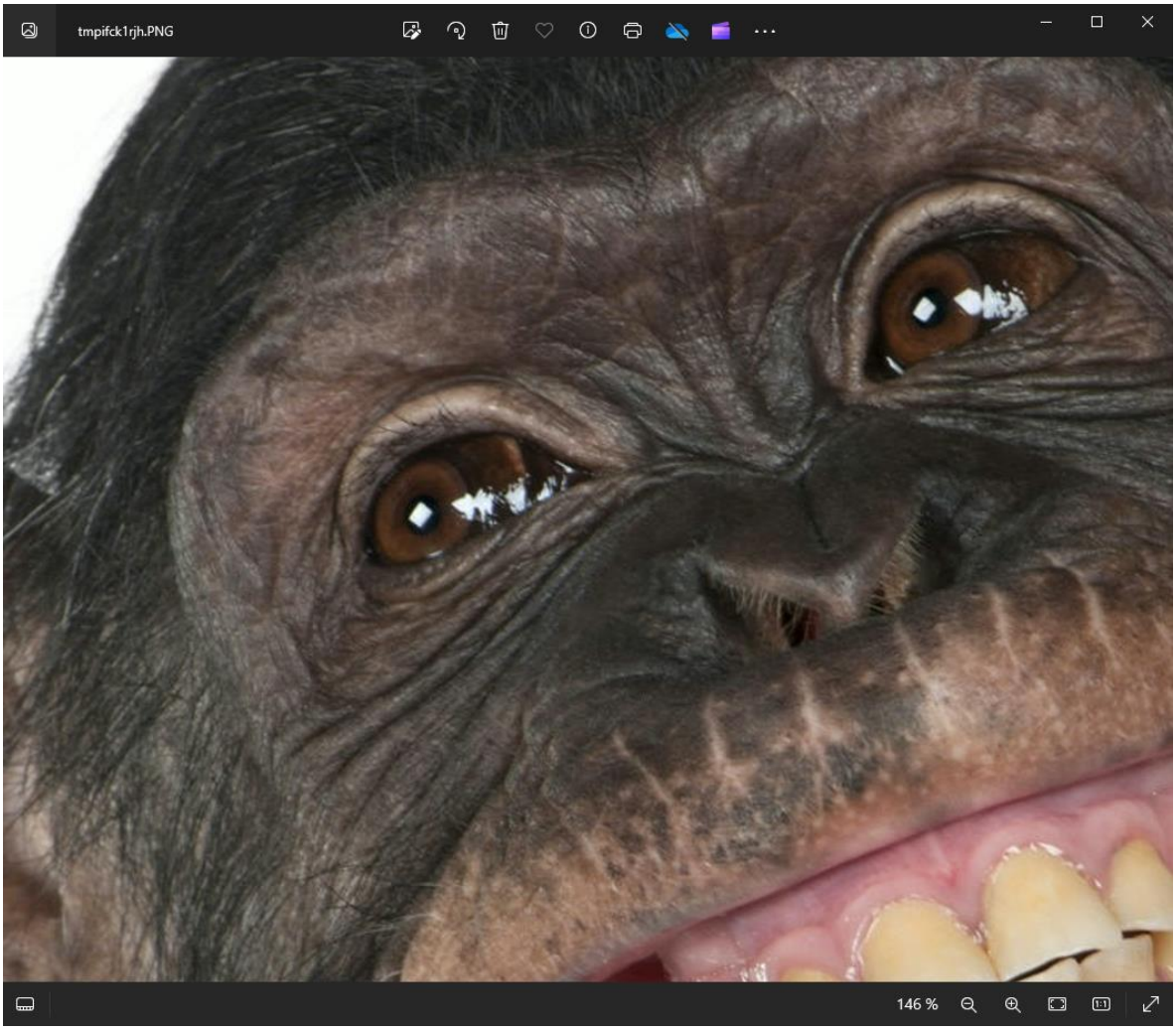

Рисунок 5 – Q=80

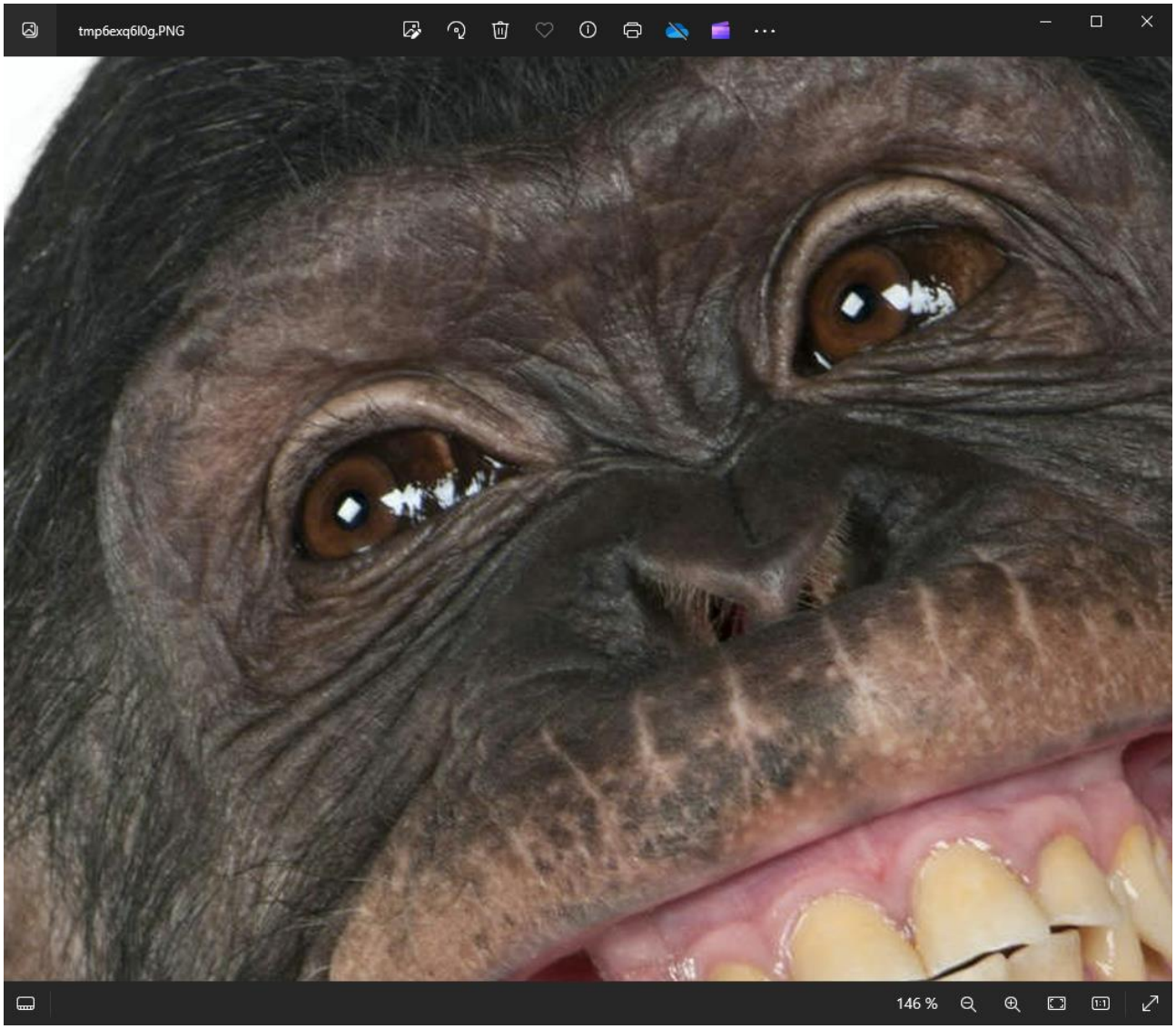

Рисунок 6 – Q=60

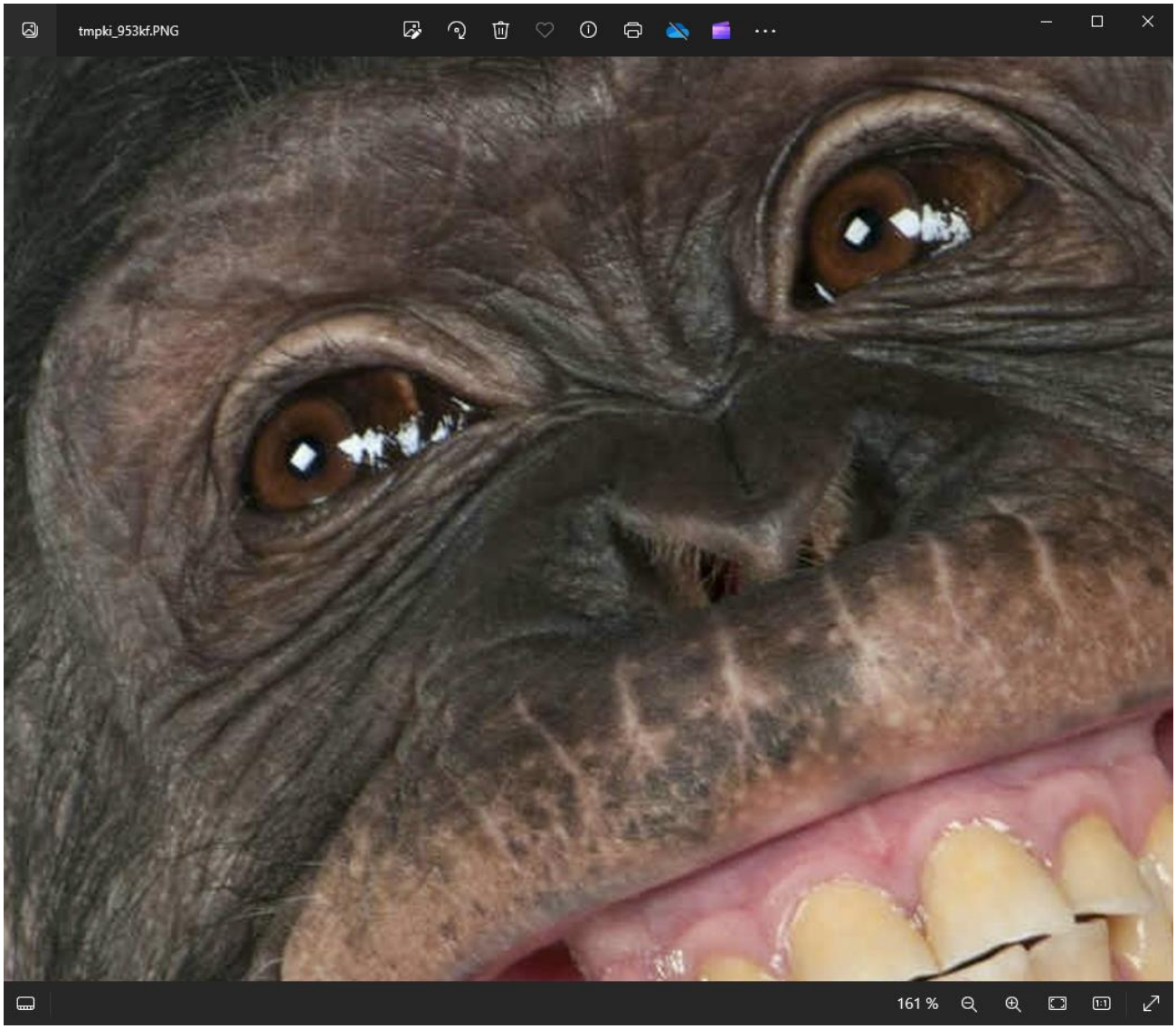

Рисунок 7 – Q=40

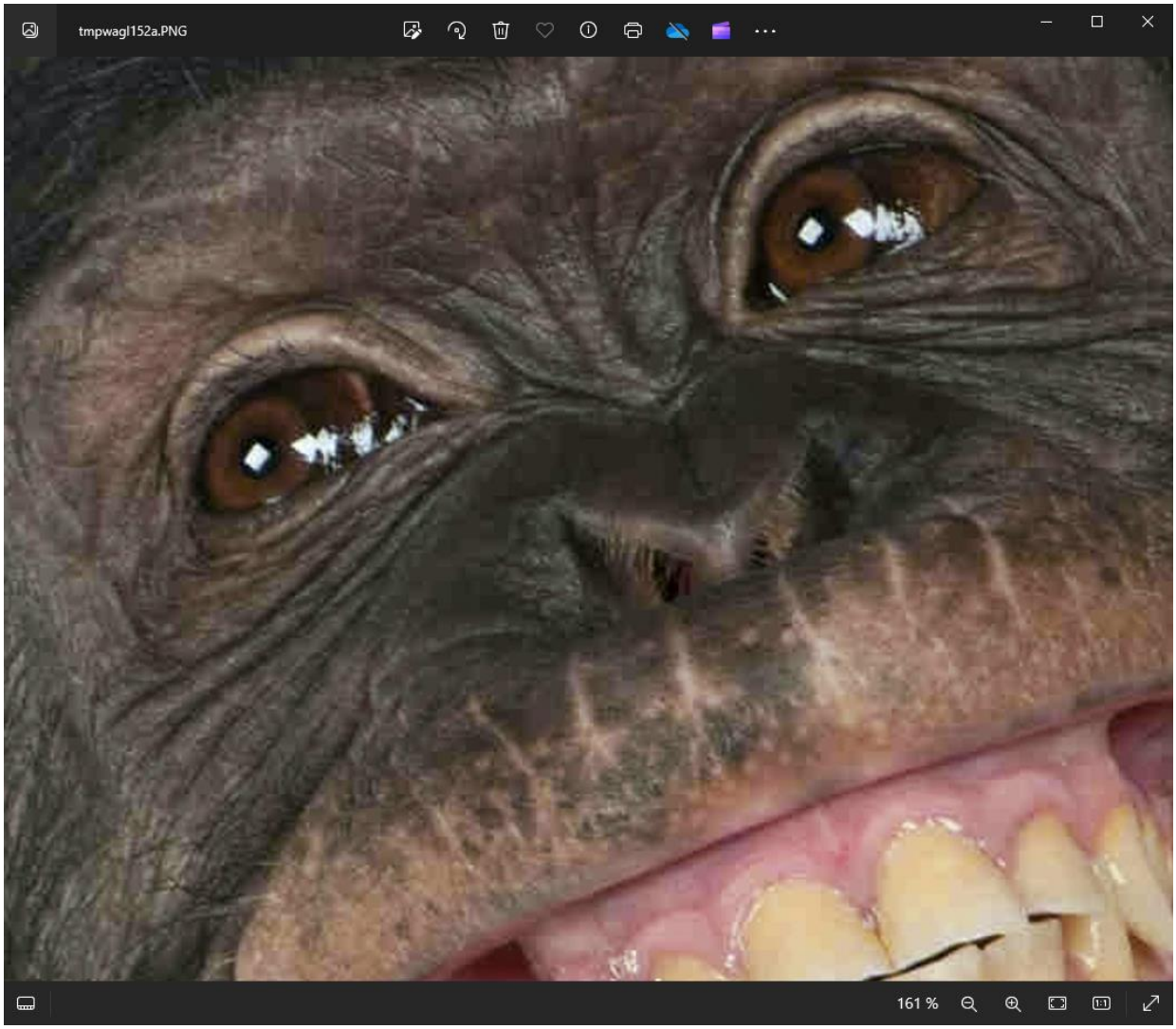

Рисунок 8 – Q=20

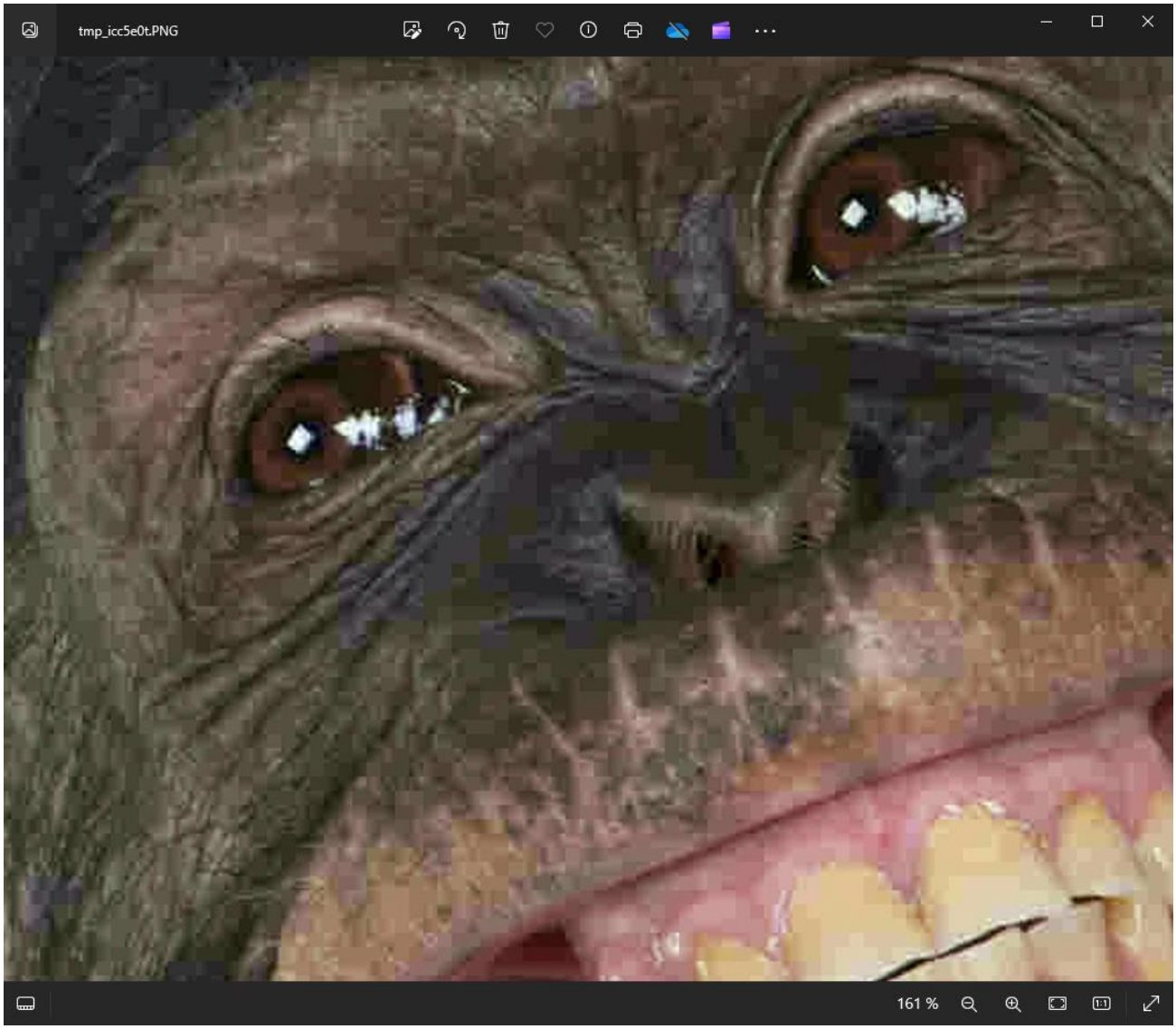

Рисунок 9 – Q=10

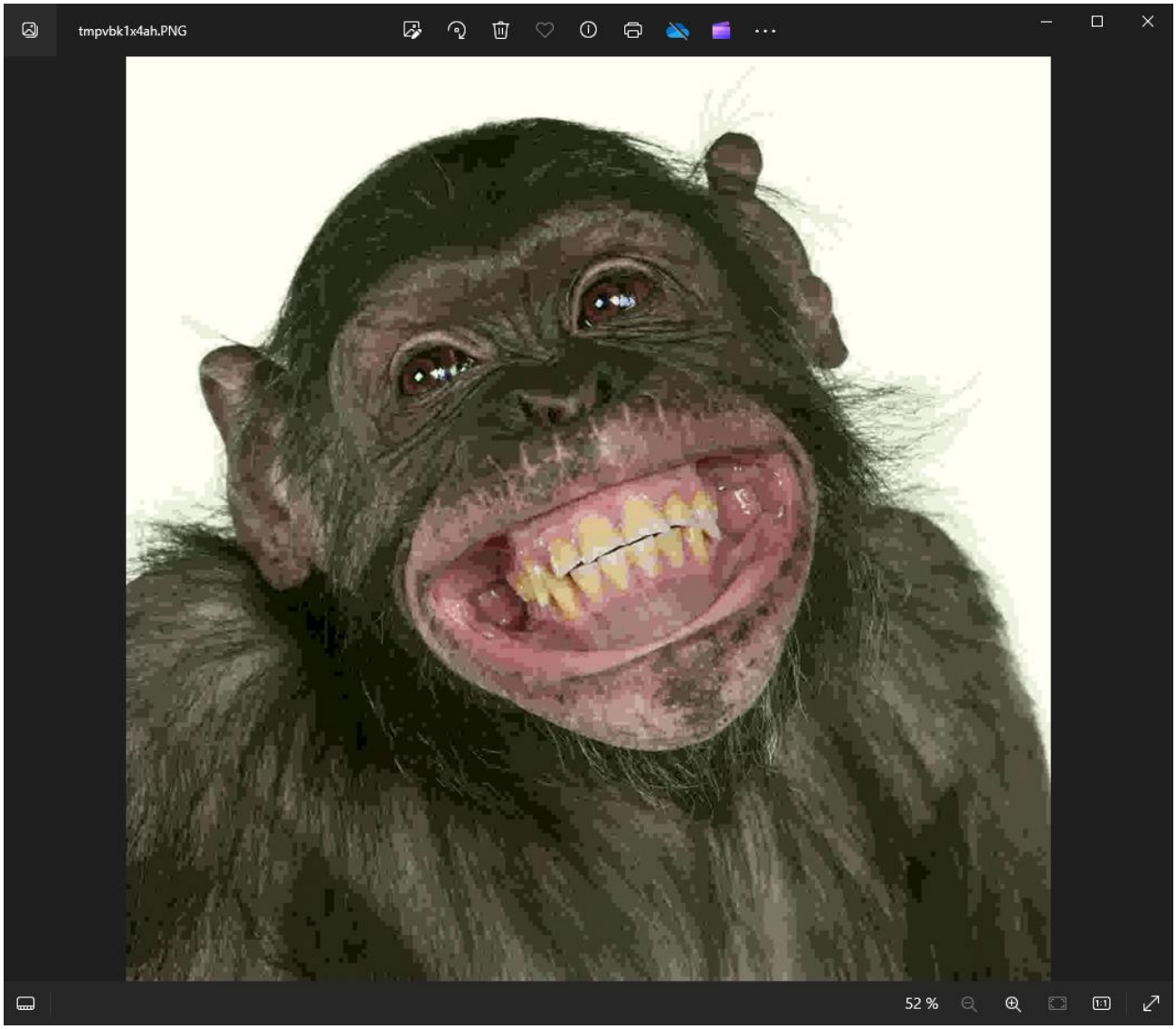

Рисунок 10 – Q=5

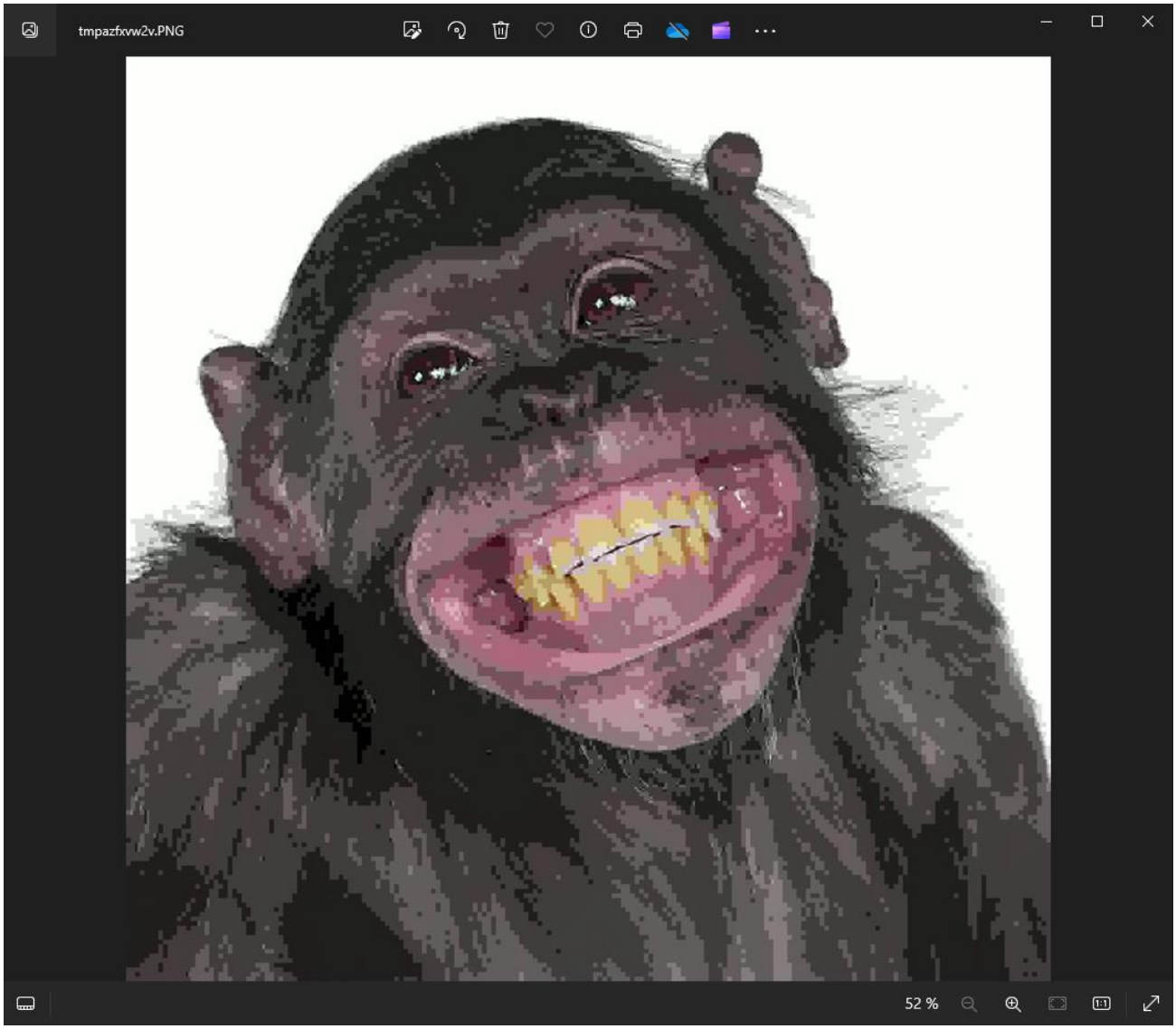

Рисунок 11 – Q=1

Изменение фото из одного цвета при изменении качества Q:

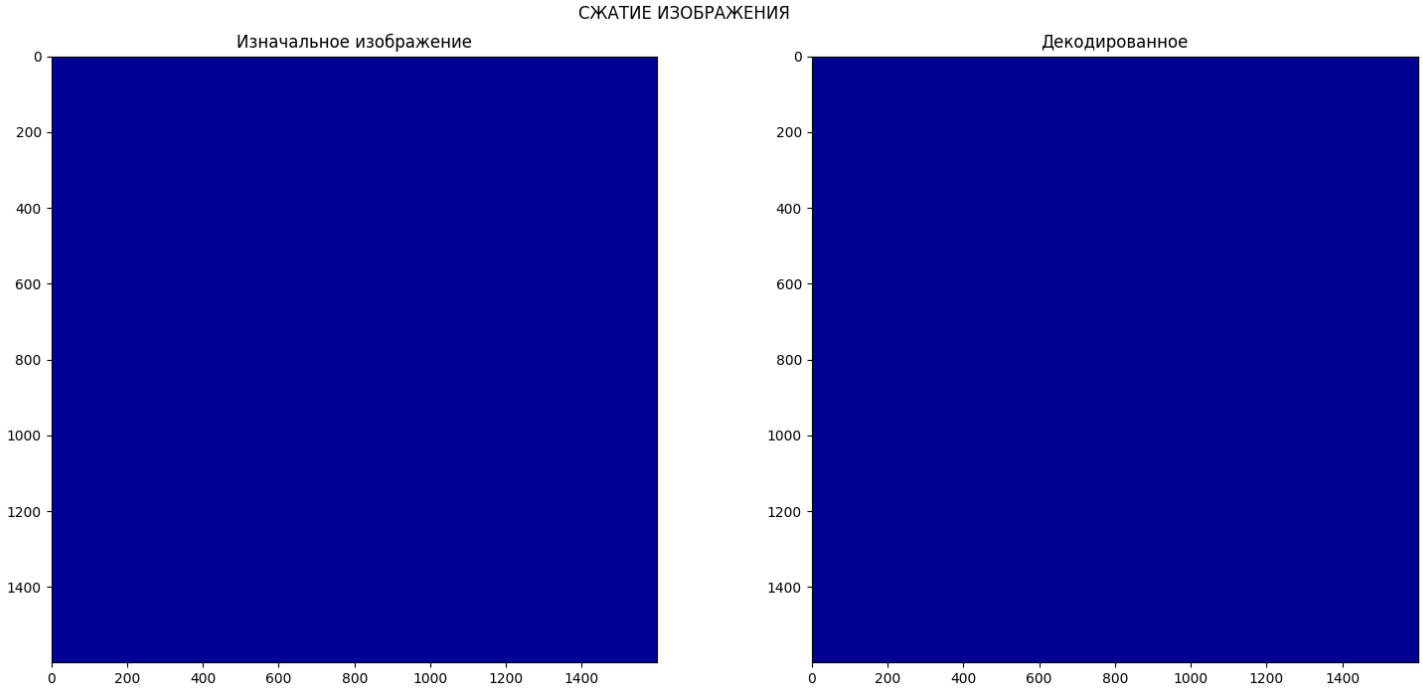

Рисунок 12 – Q=80

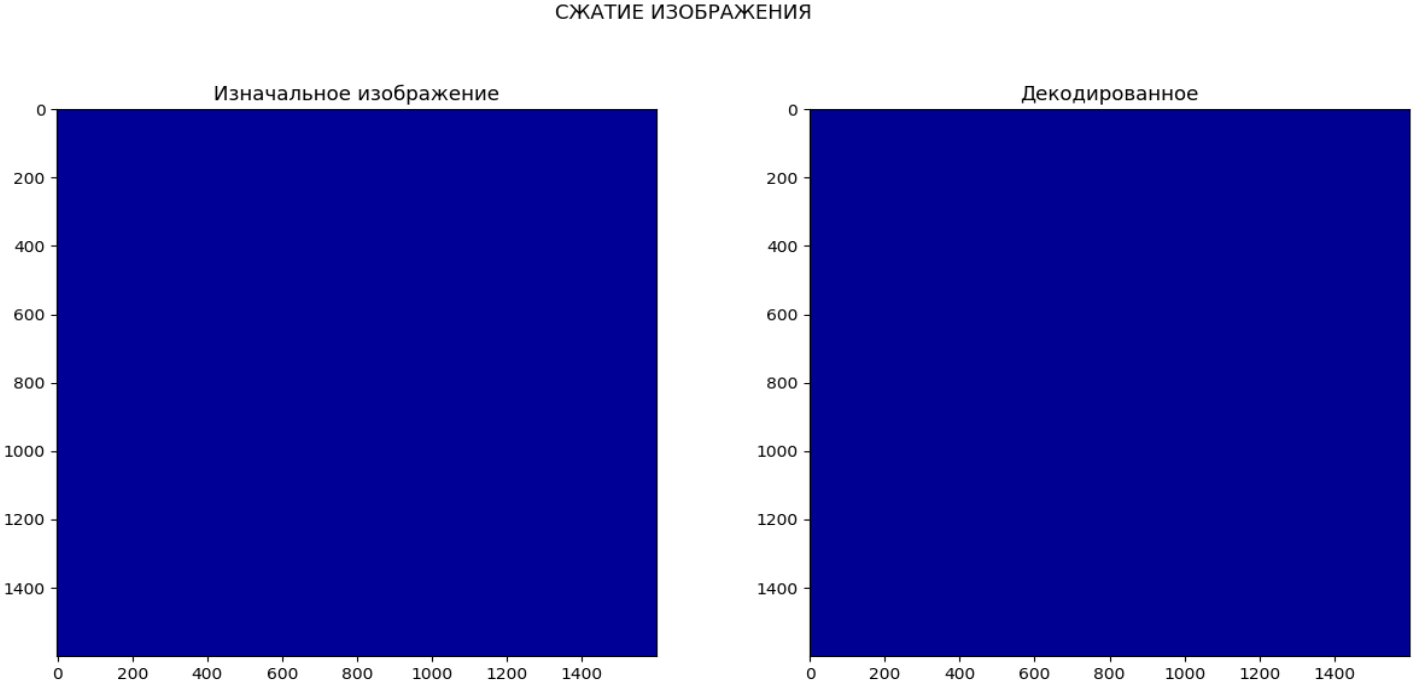

Рисунок 14 – Q=30

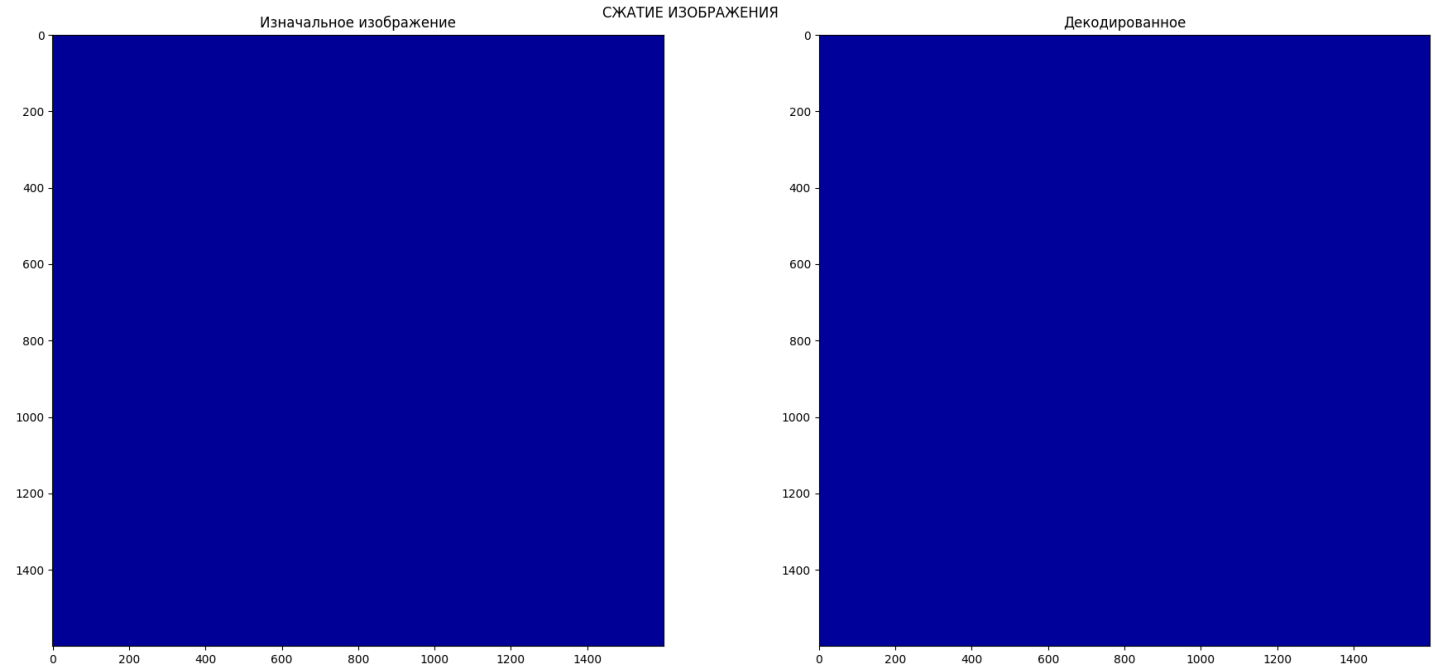

Рисунок 15 – Q=15

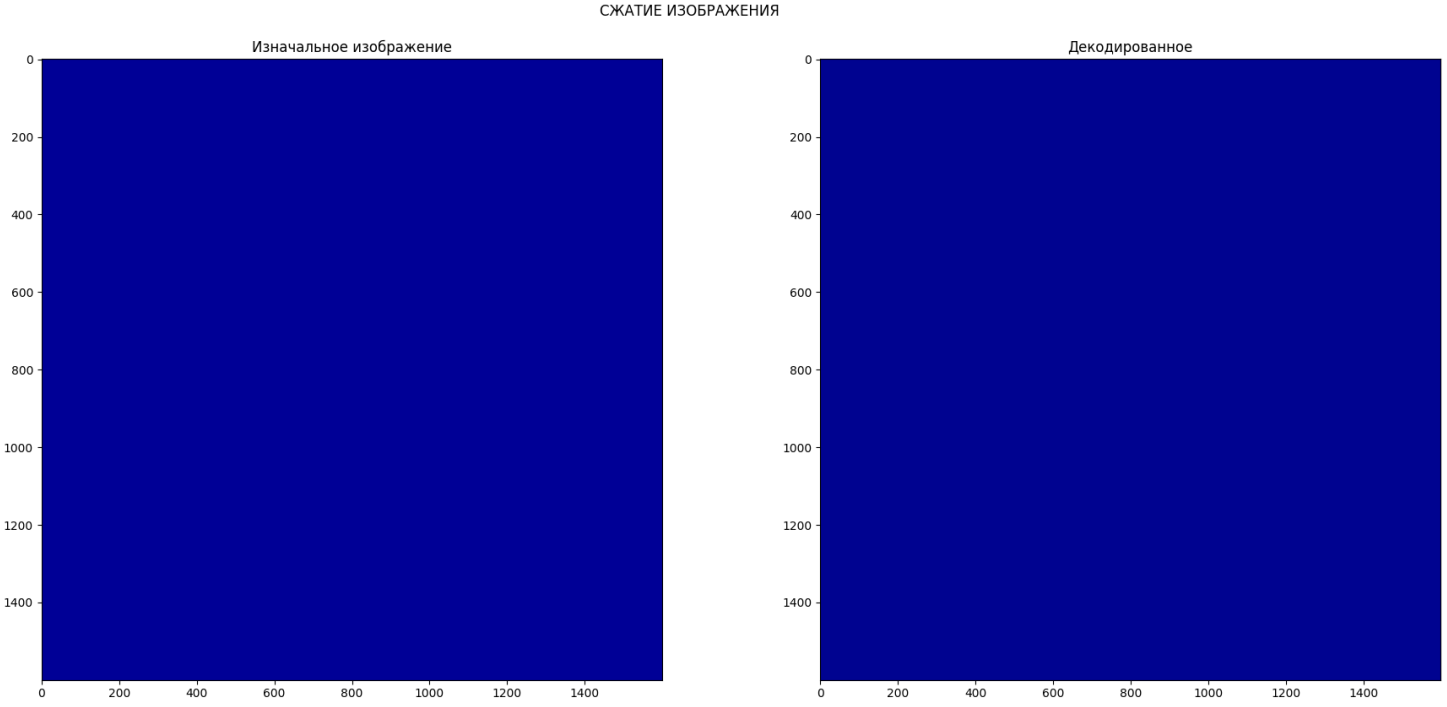

Рисунок 15 – Q=5

Изменение фото из случайного цвета при изменении качества Q:

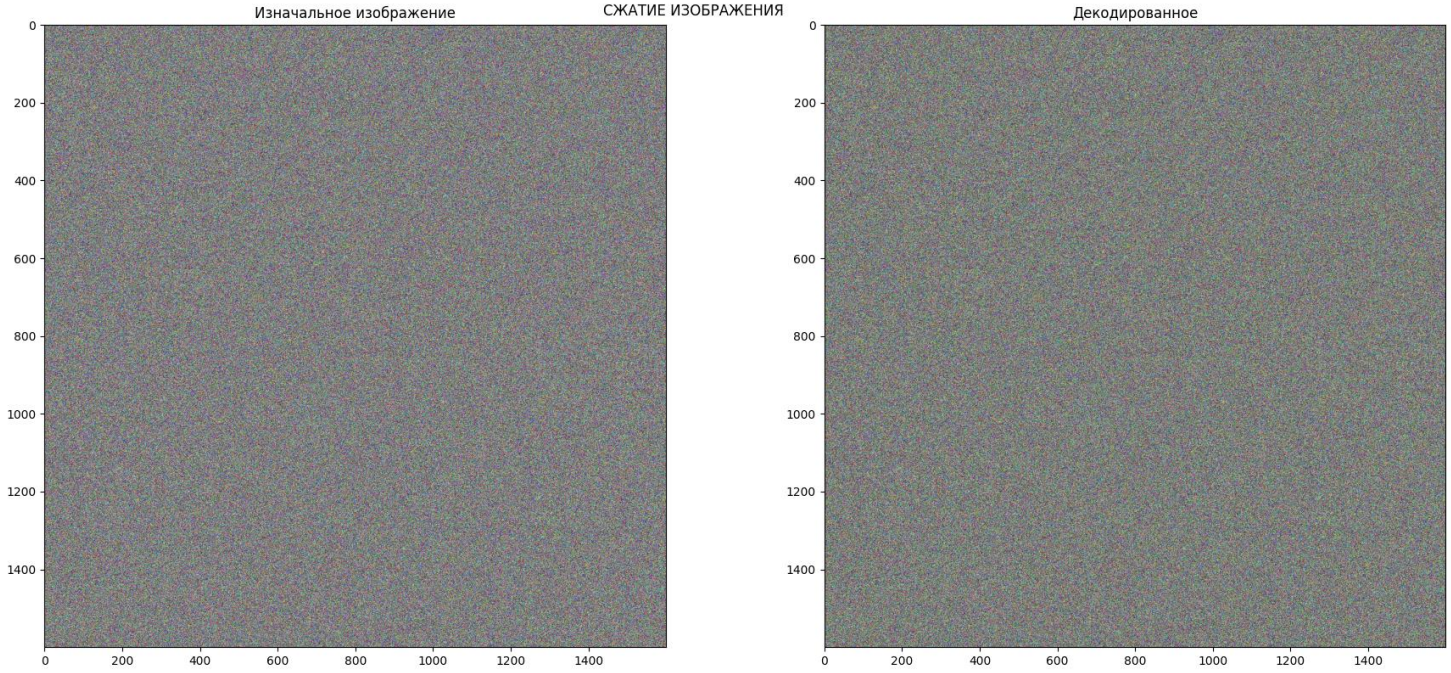

Рисунок 12 – Q=80

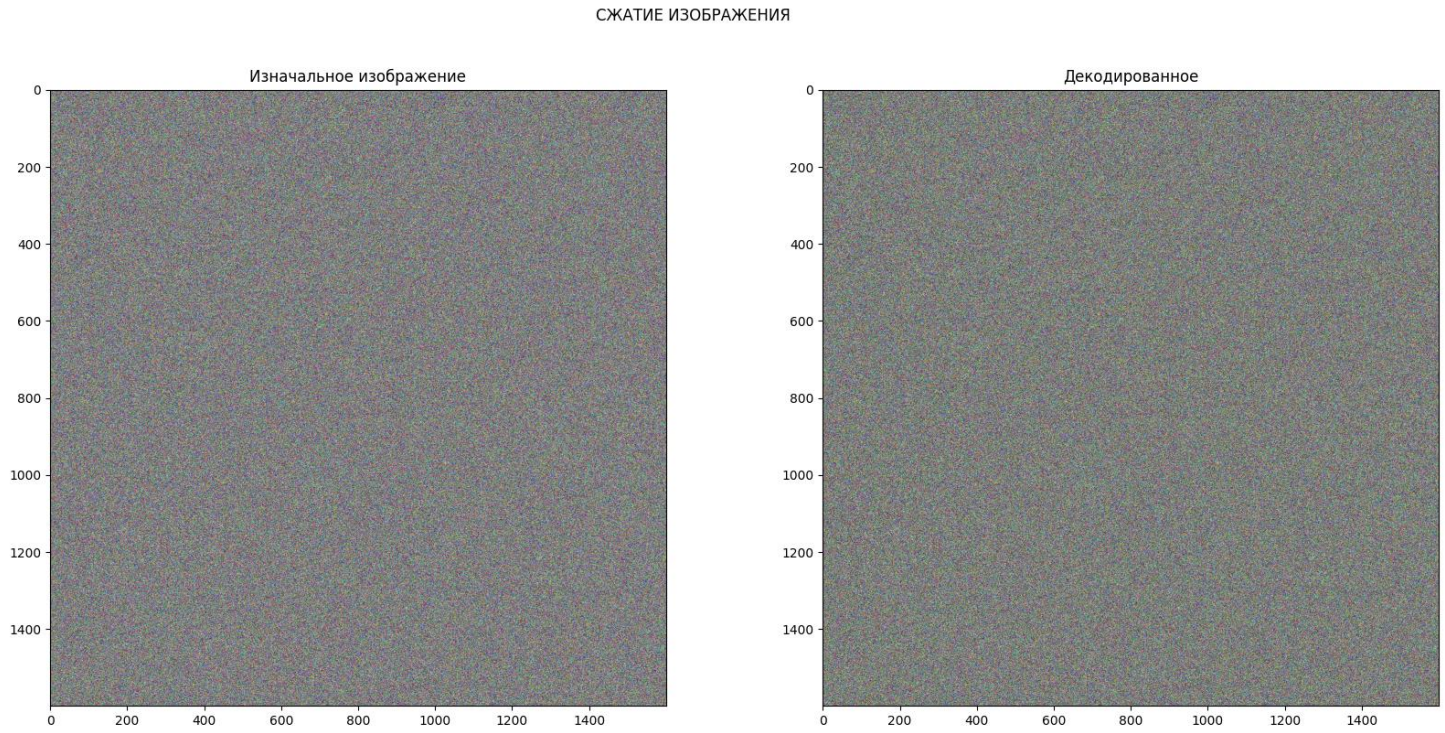

Рисунок 14 – Q=30

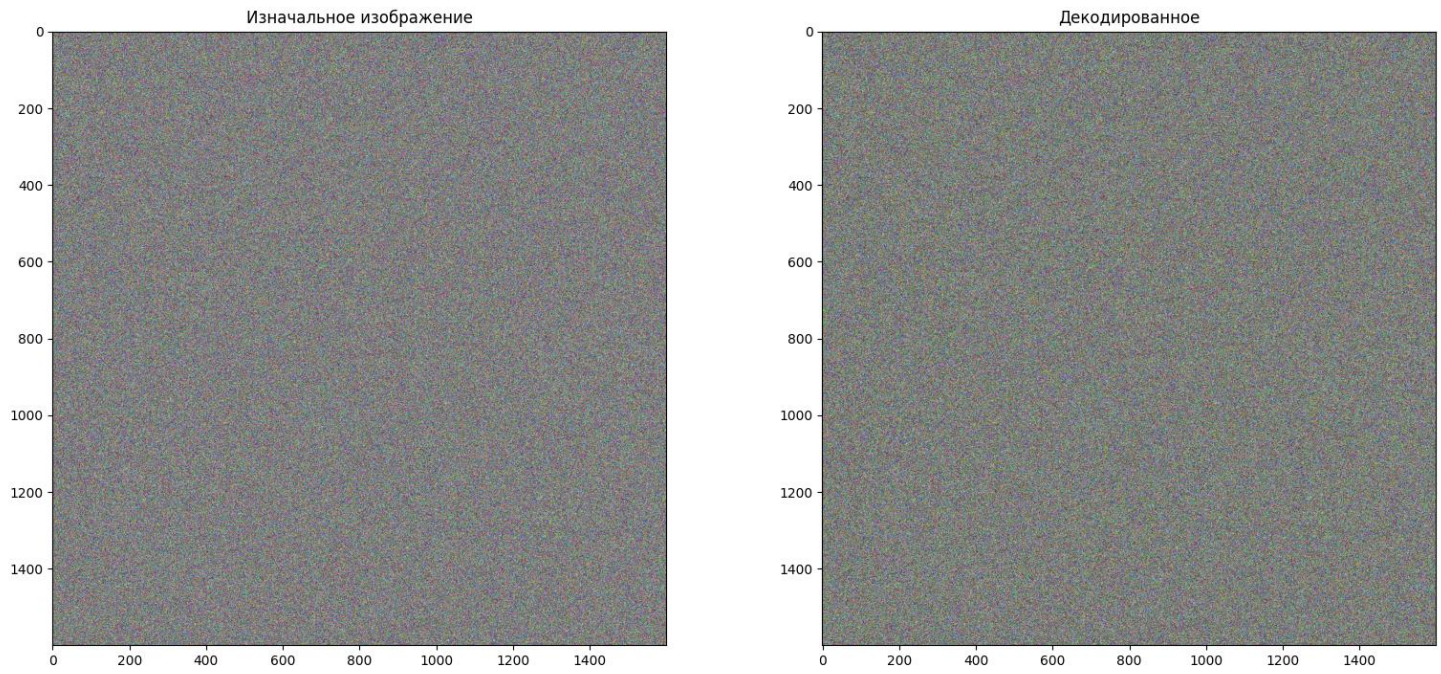

Рисунок 15 – Q=15

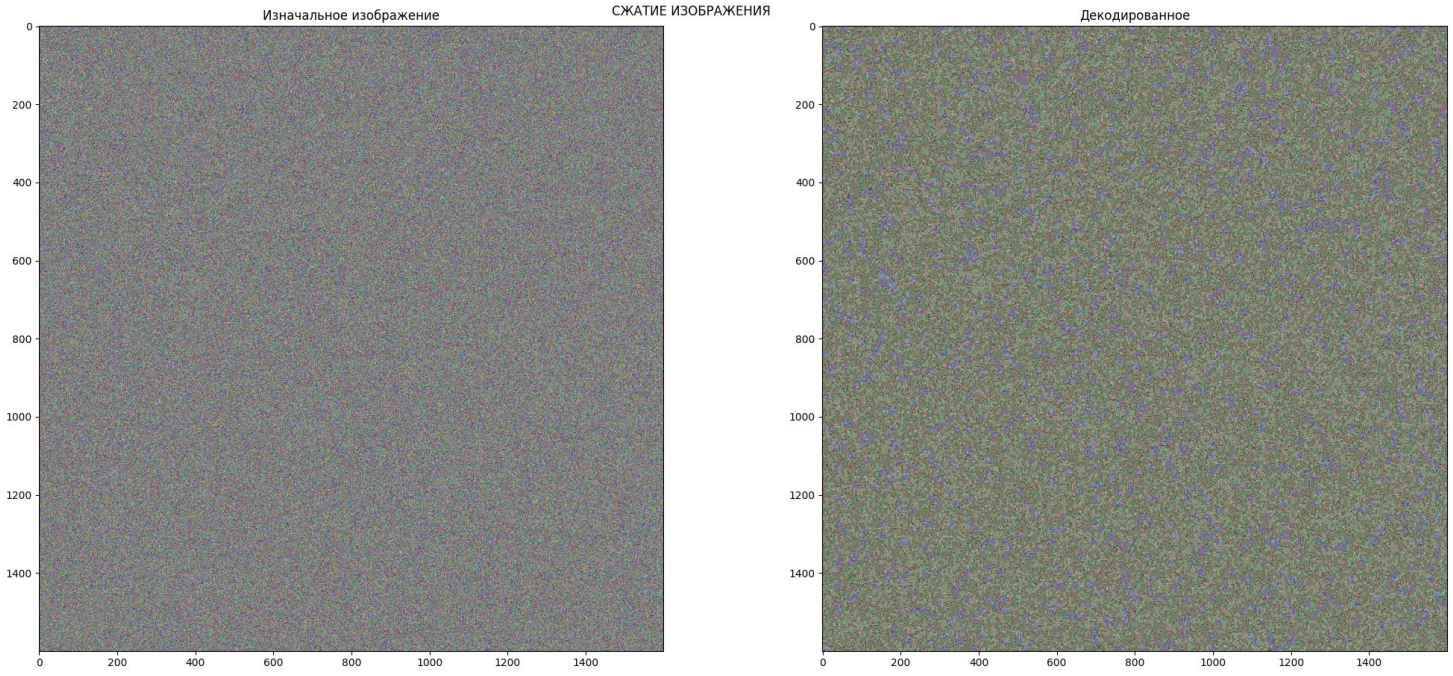

Рисунок 15 – Q=5

Ссылка на ГИТ:

https://github.com/pavlichek121/lr2_compress_image.git

2. Весь написанный код

2.1. ДКТ с помощью матрицы

//ДКТ

```
def dctmtx(M):

    T = np.zeros((M, M))
    for p in range(M):
        for q in range(M):
            if p == 0:
                T[p, q] = 1 / np.sqrt(M)
            else:
                T[p, q] = np.sqrt(2 / M) * np.cos(np.pi * (2 * q + 1) * p / (2 * M))
    return T
def dct_matrix(image):

    # Получаем размеры изображения
    height, width = image.shape

    # Создаем матрицу ДКП для строк и столбцов
    T_height = dctmtx(height)
    T_width = dctmtx(width)

    # Выполняем преобразование: T * image * T'
    dct_transformed = np.dot(np.dot(T_height, image), T_width.T)

    return dct_transformed.astype(np.int16)
```

//ОБРАТНОЕ ДКТ

```
def idct(dct_coeffs):

    # Получаем размеры коэффициентов ДКТ
    height, width = dct_coeffs.shape

    # Создаем матрицу ДКП для строк и столбцов
    T_height = dctmtx(height)
    T_width = dctmtx(width)

    # Выполняем обратное преобразование: T' * dct_coeffs * T
    idct_transformed = np.dot(np.dot(T_height.T, dct_coeffs), T_width)

    return idct_transformed.astype(np.int16)
```

2.2. Разделение и объединение на блоки 8x8

//РАЗДЕЛЕНИЕ НА БЛОКИ

```
def split_image(image):
    """
    Разделяет изображение на блоки размером 8x8.
    """
    height, width = image.shape
```

```
    blocks = []

    for y in range(0, height, 8):
        for x in range(0, width, 8):
            blocks.append(image[y:y+8, x:x+8])
    return np.array(blocks)
```

//ОБЪЕДИНЕНИЕ БЛОКОВ

```
def merge_blocks(blocks, image_shape):
    """
    Объединяет блоки обратно в изображение.
    """
    height, width, _ = image_shape

    image = np.zeros((height, width), dtype=int)
    block_idx = 0
    for y in range(0, height, 8):
        for x in range(0, width, 8):
            image[y:y+8, x:x+8] = blocks[block_idx]
            block_idx += 1
    return image
```

2.3. РЛЕ

//РЛЕ

```
def rle(input_array):
    def append_rle(rle_line, current_char, count):
        if count > 1:
            rle_line.append(' ')
            rle_line.append(chr(count))
        rle_line.append(current_char)

    rle_line = []
    count = 1

    # Преобразуем первый элемент массива в символ
    if input_array[0] < 0:
        current_char = 'ε' + chr(-input_array[0])
    else:
        current_char = chr(input_array[0])

    for i in range(1, len(input_array)):
        if input_array[i] < 0:
            next_char = 'ε' + chr(-input_array[i])
        else:
            next_char = chr(input_array[i])

        if next_char == current_char:
            count += 1
        else:
            append_rle(rle_line, current_char, count)
            current_char = next_char
            count = 1

    # Обрабатываем последнюю последовательность символов
```

```
    append_rle(rle_line, current_char, count)

    return ''.join(rle_line)
```

//ОБРАТНОЕ РЛЕ

```
def rle_decoder(encoded_string):
    decoded_array = []
    i = 0
    length = len(encoded_string)

    while i < length:
        if encoded_string[i] == '▮':
            # Читаем количество повторений
            count = ord(encoded_string[i + 1])
            i += 2
        else:
            count = 1

        if encoded_string[i] == 'ε':
            # Отрицательное число
            number = -ord(encoded_string[i + 1])
            i += 2
        else:
            # Положительное число
            number = ord(encoded_string[i])
            i += 1

        decoded_array.extend([number] * count)

    # Преобразование списка в ndarray
    return np.array(decoded_array)
```

2.4. Квантование

//ПОЛУЧЕНИЕ МАТРИЦЫ КВАНТОВАНИЯ

```
def get_quantization_matrix(quality):
    quantization_matrix = np.array([
        [16, 11, 10, 16, 24, 40, 51, 61],
        [12, 12, 14, 19, 26, 58, 60, 55],
        [14, 13, 16, 24, 40, 57, 69, 56],
        [14, 17, 22, 29, 51, 87, 80, 62],
        [18, 22, 37, 56, 68, 109, 103, 77],
        [24, 35, 55, 64, 81, 104, 113, 92],
        [49, 64, 78, 87, 103, 121, 120, 101],
        [72, 92, 95, 98, 112, 100, 103, 99]
    ])
    # Проверка, что уровень качества находится в диапазоне от 1 до 100
    if quality < 1 or quality > 100:
        raise ValueError("Уровень качества должен быть в диапазоне от 1 до 100.")
```

```
    # Масштабирование базовой матрицы квантования в соответствии с уровнем качества
    if quality<=50:
        scaling_factor=5000 / quality
    else:
        scaling_factor=200-2*quality


    scaled_matrix = np.floor((quantization_matrix * scaling_factor+50)/100)
    scaled_matrix= np.clip(scaled_matrix, 1, 255)

    return scaled_matrix.astype(np.uint8)
```

// КВАНТОВАНИЕ КОЭФ ДКТ

```
def quantization_dct(dct_matrix, quantization_matrix):
    quantized_matrix = np.round(dct_matrix / quantization_matrix)
    return quantized_matrix.astype(int)
```

// ОБРАТНОЕ КВАНТОВАНИЕ КОЭФ ДКТ

```
def de_quantization_dct(dct_matrix, quantization_matrix):
    quantized_matrix = np.round(dct_matrix * quantization_matrix)
    return quantized_matrix.astype(int)
```

2.5. Работа с YCbCr

// ИЗ RGB В YCBCR

```
def rgb_to_ycbcr(mas):
    #создаем отдельные массивы по цветам
    R, G, B = mas[:, :, 0], mas[:, :, 1], mas[:, :, 2]

    #переводим в YCbCr
    Y = 0.299 * R + 0.587 * G + 0.114 * B
    Cb = -0.1687 * R - 0.3313 * G + 0.5 * B + 128
    Cr = 0.5 * R - 0.4187 * G - 0.0183 * B + 128

    # Объединение массивов Y, Cb и Cr в один трехмерный массив
    ycbcr_image_array = np.dstack((Y, Cb, Cr))
    # Ограничение значений компонент в диапазоне [0, 255]
    ycbcr_image_array = np.clip(ycbcr_image_array, 0, 255)
    # Преобразование массива к типу данных np.uint8 (беззнаковые 8-битные целые числа)
    ycbcr_image_array = ycbcr_image_array.astype(np.uint8)

    return ycbcr_image_array
```

// ИЗ YCBCR В RGB

```
def ycbcr_to_rgb(Y, Cb, Cr):
    # Разделение входного массива изображения на его цветовые компоненты: Y, Cb и Cr


    #меняем тип данных массивов, чтобы в них можно было хранить отрицательные числа
    Y=Y.astype(np.int16)
    Cb = Cb.astype(np.int16)
    Cr = Cr.astype(np.int16)
```

```
    # Вычисление компонент R, G и B в цветовом пространстве RGB
    R = Y + 1.402 * (Cr - 128)
    G = Y - 0.34414 * (Cb - 128) - 0.71414 * (Cr - 128)
    B = Y + 1.772 * (Cb - 128)

    # Преобразование массивов R, G и B к типу данных np.uint8 (беззнаковые 8-битные целые числа)
    R = np.uint8(np.clip(R, 0, 255))
    G = np.uint8(np.clip(G, 0, 255))
    B = np.uint8(np.clip(B, 0, 255))

    # Объединение массивов R, G и B в один трехмерный массив
    rgb_image_array = np.dstack((R, G, B))

    return rgb_image_array
```

2.6. Обход ZigZag

// ОБХОД ЗИГЗАГОМ

```
def zigzag(matrix):
    result = []  # Список для хранения результата
    row, col = len(matrix), len(matrix[0])  # Количество строк и столбцов в матрице
    total = row + col - 1  # Общее количество диагоналей

    for current_sum in range(total):
        if current_sum % 2 == 0:
            for i in range(min(current_sum, row-1), max(-1, current_sum-col), -1):
                result.append(matrix[i][current_sum - i])
        else:
            for i in range(max(0, current_sum-col+1), min(current_sum+1, row)):
                result.append(matrix[i][current_sum - i])

    return result
```

// ОБРАТНЫЙ ОБХОД ЗИГЗАГОМ

```
def inverse_zigzag(array, rows, cols):
    matrix = np.zeros((rows, cols), dtype=int)  # Создаем пустую матрицу нужного размера
    current_index = 0  # Индекс для элементов входного массива
    total = rows + cols - 1  # Общее количество диагоналей


    for current_sum in range(total):
        if current_sum % 2 == 0:
            for i in range(min(current_sum, rows-1), max(-1, current_sum-cols), -1):
                matrix[i][current_sum - i] = array[current_index]
                current_index += 1
```

```
        else:
            for i in range(max(0, current_sum-cols+1), min(current_sum+1, rows)):
                matrix[i][current_sum - i] = array[current_index]
                current_index += 1


    return matrix
```

2.7. Сабсемплинг

// ДАУНСЕМПЛИНГ 3 ВИДАМИ

```
def downsampling(image, factor_h, factor_w, downsampling_type):
    height, width = image.shape
    # Проверка входных данных

    if downsampling_type==1:

    #рассчитываем количество строк, столбцов, которые необходимо исключить
        height_out, width_out = np.floor(height/factor_h), np.floor(width/factor_w);
        delete_lines_h, delete_lines_w= height- height_out, width-width_out
    #индекс начала
        start_index = 0
    #пока нужно удалить строки
        while delete_lines_h > 0:
            # Находим индексы четных строк, учитывая текущий размер матрицы
            even_indices = list(range(start_index, image.shape[0], 2))
            #count нужен, так как при удалении строки из матрицы индексы сдвигаются
            count_delete=0
            for index in even_indices:
                if delete_lines_h > 0:
                    #удаляем четную стркоу (с учетом удаленных строк)
                    image = np.delete(image, index-count_delete, axis=0)
                    count_delete+=1
                    delete_lines_h -= 1

            # Обновляем start_index, чтобы начать с начала, если достигнут конец матрицы
            start_index = max(0, start_index - image.shape[0])

        start_index = 0
        while delete_lines_w > 0:
            # Находим индексы четных столбцов, учитывая текущий размер матрицы
            even_indices = list(range(start_index, image.shape[1], 2))
            # count нужен, так как при удалении строки из матрицы индексы сдвигаются
            count_delete = 0
            for index in even_indices:
                if delete_lines_w > 0:
                    # удаляем четную стркоу (с учетом удаленных строк)
                    image = np.delete(image, index-count_delete, axis=1)
                    count_delete += 1
                    delete_lines_w -= 1
```

```
            # Обновляем start_index, чтобы начать с начала, если достигнут конец матрицы
            start_index = max(0, start_index - image.shape[1])
        return image

    if downsampling_type == 2:

        #считаем новые высоту и длину
        height_out, width_out = int((height / factor_h)), int((width / factor_w))
        #размеры блоков
        block_h, block_w=int(np.floor(height/height_out)), int(np.floor(width / width_out))
        # Создание нового массива для уменьшенного изображения
        downsampling_image = np.zeros((height_out, width_out), dtype=np.uint8)

        # Проход по уменьшенному изображению
        for i in range(height_out):
            for j in range(width_out):
                # Вычисление среднего значения блока (все, что с i - работа со строками, что с j - работа с столбцами)
                # Выделяем блоки
                block = image[i * block_h: (i + 1) * block_h, j * block_w: (j + 1) * block_w]
                #среднее значение
                average = np.mean(block)
                # Запись среднего значения в уменьшенное изображение
                downsampling_image[i, j] = int(average)
        return downsampling_image

    if downsampling_type == 3:
        # считаем новые высоту и длину
        height_out, width_out = int((height / factor_h)), int((width / factor_w))
        # размеры блоков
        block_h, block_w = int(np.floor(height / height_out)), int(np.floor(width / width_out))

        # Создание нового массива для уменьшенного изображения
        downsampling_image = np.zeros((height_out, width_out), dtype=np.uint8)

        # Проход по уменьшенному изображению
        for i in range(height_out):
            for j in range(width_out):
                # Вычисление среднего значения блока (все, что с i - работа со строками, что с j - работа с столбцами)
                # Выделяем блоки
                block = image[i * block_h: (i + 1) * block_h, j * block_w: (j + 1) * block_w]
                # среднее значение
                average = np.mean(block)
                #матрица разницы элементов блока и среднего значения
                diff_matrix=abs(block- average)
                #индекс элемента, разница со средним которого минимальна
                index_of_nearest_pixel = diff_matrix.argmin()
                #находим элемент по индексу
                nearest_pixel = block.flat[index_of_nearest_pixel]
                # Запись среднего значения в уменьшенное изображение
                downsampling_image[i, j] = int(nearest_pixel)
        return downsampling_image
```

// АПСЕМПЛИНГ

```
def upsampling_image(image, factor_h, factor_w):
    height, width = image.shape
```

```
    # считаем новые высоту и длину
    height_out, width_out = int((height * factor_h)), int((width * factor_w))
    # размеры блоков
    block_h, block_w = int(np.floor(height_out / height)), int(np.floor(width_out / width))
    # Создание нового массива для увеличенного изображения
    upsampling_image = np.zeros((height_out, width_out), dtype=np.uint8)

    # Проход по увеличенному изображению
    for i in range(height):
        for j in range(width):
            # Выделяем блок из исходного изображения
            block = image[i,j]
            # Копируем блок в увеличенное изображение
            upsampling_image[i * factor_h: (i + 1) * factor_h, j * factor_w: (j + 1) * factor_w] = block
    return upsampling_image
```

2.8. Сохранение файла

```
def save_image_to_txt(image_array, filename):
    """Сохраняет изображение, представленное массивом NumPy, в текстовый файл. """

    # Открываем файл для записи
    with open(filename, 'w') as file:
        # Перебираем все пиксели в изображении
        for row in image_array:
            for pixel in row:
                # Записываем значения каналов YCbCr в файл, разделяя их пробелами
                file.write(f"{pixel[0]} {pixel[1]} {pixel[2]} ")
            # Добавляем перенос строки после каждой строки пикселей
            file.write("\n")
```

2.9. Полное кодирование и декодирование

```
def full_compress_and_decode_image(image_path, target_folder):
    "основная часть"

#########################################################################################################
#########################################################################

    'Считывание фото'
    ###############################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
    image_array_rgb = np.array(image_rgb)
    width, height = image.size
    ###############################################################################################
    'Записываем массив фото RGB в файл'
```

```
#########################################################################################
save_image_to_txt(image_array_rgb, '0_RGB_image.txt')
#########################################################################################

print(f"1. Шаг: Преобразование из RGB в YCbCr")
'YCbCr'
#########################################################################################
'массив в формате YCbCr'
image_array_ycbcr = rgb_to_ycbcr(image_array_rgb)
#########################################################################################
'Записываем массив фото YCbCrв файл'
#########################################################################################
save_image_to_txt(image_array_ycbcr, '1_YCbCr_image.txt')
#########################################################################################

'YCbCr по массивам'
#########################################################################################
Y, Cb, Cr = image_array_ycbcr[:, :, 0], image_array_ycbcr[:, :, 1], image_array_ycbcr[:, :, 2]
#########################################################################################

print(f"2. Шаг: Сабсемплинг матриц")
'сабсемплинг'
#########################################################################################
"коэф. сжатия"
compression_h = 10
compression_w = 10

"даунсемплинг Cb Cr 2 методом даунсемплинга"
downsampling_matrix_Cb = downsampling(Cb, compression_h, compression_w, 2)
downsampling_matrix_Cr = downsampling(Cr, compression_h, compression_w, 2)

"апсемплинг Cb Cr, чтобы вернуть к размеру Y"
Cb = upsampling_image(downsampling_matrix_Cb, compression_h, compression_w)
Cr = upsampling_image(downsampling_matrix_Cr, compression_h, compression_w)
#########################################################################################
'Записываем массивы фото после сабсемплинга в файл'
#########################################################################################
ycbcr_sub = np.dstack((Y, Cb, Cr))
save_image_to_txt(ycbcr_sub, '2_subsempling_image.txt')
#########################################################################################

'разделение изображения на блоки 8x8'
#########################################################################################
blocks_Y = split_image(Y)
blocks_Cb = split_image(Cb)
blocks_Cr = split_image(Cr)
#########################################################################################

print(f"3. Шаг: Применение ДКТ матрицей")
'ДКТ матрицей'
#########################################################################################

'применяем дкт'
coeffs_blocks_Y = np.array([dct_matrix(block) for block in blocks_Y])
coeffs_blocks_Cb = np.array([dct_matrix(block) for block in blocks_Cb])
coeffs_blocks_Cr = np.array([dct_matrix(block) for block in blocks_Cr])
#########################################################################################
'Записываем массивы фото после ДКТ в файл'
#########################################################################################
```

```
'Объединяем блоки'
Y_dct = merge_blocks(coeffs_blocks_Y, image_array_rgb.shape)
Cb_dct = merge_blocks(coeffs_blocks_Cb, image_array_rgb.shape)
Cr_dct = merge_blocks(coeffs_blocks_Cr, image_array_rgb.shape)

'Сохраняем txt'
coeffs_blocks = np.dstack((Y_dct, Cb_dct, Cr_dct))
save_image_to_txt(coeffs_blocks, '3_DCT_image.txt')
########################################################################################

print(f"4. Шаг: Квантование матриц с Q=50")
'Квантование'
########################################################################################
'Ставим качество 50 и создаем матрицу квантования'
quality = 50
quantization_matrix_quality = get_quantization_matrix(quality)

'Квантование'
quantization_coeffs_Y = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Y])
quantization_coeffs_Cb = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cb])
quantization_coeffs_Cr = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cr])
########################################################################################
'Объединение блоков'
########################################################################################
Y_merge = merge_blocks(quantization_coeffs_Y, image_array_rgb.shape)
Cb_merge = merge_blocks(quantization_coeffs_Cb, image_array_rgb.shape)
Cr_merge = merge_blocks(quantization_coeffs_Cr, image_array_rgb.shape)
########################################################################################
'Записываем массивы фото после Квантования в файл'
########################################################################################
merge_blocks_ = np.dstack((Y_merge, Cb_merge, Cr_merge))
save_image_to_txt(merge_blocks_, '4_Квантование_image.txt')
########################################################################################

print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
########################################################################################
zigzag_Y = zigzag(Y_merge)
zigzag_Cb = zigzag(Cb_merge)
zigzag_Cr = zigzag(Cr_merge)
########################################################################################
'Записываем фото после Зиг Зага в файл'
########################################################################################
zigzag_blocks = np.dstack((zigzag_Y, zigzag_Cb, zigzag_Cr))
save_image_to_txt(zigzag_blocks, '5_ZigZag_image.txt')
########################################################################################

print(f"6. Шаг: Сжатие с помощью RLE")
'RLE'
########################################################################################
rle_Y = rle(zigzag_Y)
rle_Cb = rle(zigzag_Cb)
rle_Cr = rle(zigzag_Cr)
########################################################################################
'Записываем фото после РЛЕ в файл'
########################################################################################
```

```
    rle_finish = rle_Y + "p" + rle_Cb + "p" + rle_Cr
    with open("6_RLE_code.txt", 'w', encoding='utf-8') as file:
        file.write(rle_finish)
    ##########################################################################################
    print(f"Сжатие завершено!")

    'Переписываем файлы txt в отдельную папку'
    ##########################################################################################
    'Путь к исходной папке'
    source_folder = 'C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv'

    'Создаем целевую папку, если её нет'
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    'Перебираем все файлы в исходной папке'
    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым файлом (например, имеет ли он расширение .txt)
        if filename.endswith('.txt'):
            # Формируем полные пути к файлу в исходной и целевой папках
            source_path = os.path.join(source_folder, filename)
            target_path = os.path.join(target_folder, filename)

            # Копируем файл из исходной папки в целевую
            shutil.copy2(source_path, target_path)

    print("Копирование завершено.")

    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым
        if filename.endswith('.txt'):
            # Формируем полный путь к файлу
            file_path = os.path.join(source_folder, filename)
            # Удаляем файл
            os.remove(file_path)
    ##########################################################################################


print(f"_____
_____\n\n")

    print(f"Проверка кодирования декодированием:")
    print(f"1. Шаг: Разжатие RLE")
    'декодер RLE'
    ##########################################################################################
    de_rle_Y = rle_decoder(rle_Y)
    de_rle_Cb = rle_decoder(rle_Cb)
    de_rle_Cr = rle_decoder(rle_Cr)
    ##########################################################################################

    print(f"2. Шаг: Обрантный зиг заг")
    'декодер RLE'
    ##########################################################################################
    de_zigzag_Y = inverse_zigzag(de_rle_Y, height, width)
    de_zigzag_Cb = inverse_zigzag(de_rle_Cb, height, width)
    de_zigzag_Cr = inverse_zigzag(de_rle_Cr, height, width)
    ##########################################################################################

    'разделение изображения на блоки 8x8'
```

```
#################################################################################################
de_blocks_Y = split_image(de_zigzag_Y)
de_blocks_Cb = split_image(de_zigzag_Cb)
de_blocks_Cr = split_image(de_zigzag_Cr)
#################################################################################################

print(f"3. Шаг: Обратное квантование")
'Обратное квантование матриц'
#################################################################################################
de_quantization_Y = de_quantization_dct(de_blocks_Y, quantization_matrix_quality)
de_quantization_Cb = de_quantization_dct(de_blocks_Cb, quantization_matrix_quality)
de_quantization_Cr = de_quantization_dct(de_blocks_Cr, quantization_matrix_quality)
#################################################################################################

print(f"4. Шаг: Обратное ДКТ")
'Обратное квантование матриц'
#################################################################################################
de_DCT_Y = np.array([idct(block) for block in de_quantization_Y])
de_DCT_Cb = np.array([idct(block) for block in de_quantization_Cb])
de_DCT_Cr = np.array([idct(block) for block in de_quantization_Cr])
#################################################################################################

'Объединение блоков матриц'
#################################################################################################
de_Y_merge = merge_blocks(de_DCT_Y, image_array_rgb.shape)
de_Cb_merge = merge_blocks(de_DCT_Cb, image_array_rgb.shape)
de_Cr_merge = merge_blocks(de_DCT_Cr, image_array_rgb.shape)
#################################################################################################

print(f"5. Шаг: Декодирование в RGB")
'Обратное кодирование в RGB'
#################################################################################################
rgb_array = ycbcr_to_rgb(de_Y_merge, de_Cb_merge, de_Cr_merge)
#################################################################################################

image_rgb_decoder = Image.fromarray(rgb_array, 'RGB')
image_rgb_decoder.save('decode_image.jpg')

# проверка изображений на кодирование
fig, axs = plt.subplots(1, 2)
# Отображение изображений на Axes
axs[0].imshow(image)
axs[0].set_title("Изначальное изображение")
axs[1].imshow(image_rgb_decoder)
axs[1].set_title("Декодированное")
# Установка расстояния между изображениями
plt.tight_layout()
# Отображение Figure
fig.suptitle("СЖАТИЕ ИЗОБРАЖЕНИЯ")

plt.show()
```

2.10. Кодирование и декодирование без YCbCr

```
def without_YCbCr_compress_and_decode_image(image_path, target_folder):
    "основная часть"

##########################################################################################################
#########################################################################

    'Считывание фото'
    ###############################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
    image_array_rgb = np.array(image_rgb)
    width, height = image.size
    ###############################################################################################
    'Записываем массив фото RGB в файл'
    ###############################################################################################
    save_image_to_txt(image_array_rgb, '0_RGB_image.txt')
    ###############################################################################################

    'YCbCr по массивам'
    ###############################################################################################
    Y, Cb, Cr = image_array_rgb[:, :, 0], image_array_rgb[:, :, 1], image_array_rgb[:, :, 2]
    ###############################################################################################

    print(f"2. Шаг: Сабсемплинг матриц")
    'сабсемплинг'
    ###############################################################################################
    "коэф. сжатия"
    compression_h = 2
    compression_w = 2

    "даунсемплинг Cb Cr 2 методом даунсемплинга"
    downsampling_matrix_Y = downsampling(Y, compression_h, compression_w, 2)
    downsampling_matrix_Cb = downsampling(Cb, compression_h, compression_w, 2)
    downsampling_matrix_Cr = downsampling(Cr, compression_h, compression_w, 2)

    "апсемплинг Cb Cr, чтобы вернуть к размеру Y"
    Y = upsampling_image(downsampling_matrix_Y, compression_h, compression_w)
    Cb = upsampling_image(downsampling_matrix_Cb, compression_h, compression_w)
    Cr = upsampling_image(downsampling_matrix_Cr, compression_h, compression_w)
    ###############################################################################################
    'Записываем массивы фото после сабсемплинга в файл'
    ###############################################################################################
    ycbcr_sub = np.dstack((Y, Cb, Cr))
    save_image_to_txt(ycbcr_sub, '2_subsempling_image.txt')
    ###############################################################################################

    'разделение изображения на блоки 8x8'
    ###############################################################################################
    blocks_Y = split_image(Y)
    blocks_Cb = split_image(Cb)
    blocks_Cr = split_image(Cr)
    ###############################################################################################

    print(f"3. Шаг: Применение ДКТ матрицей")
    'ДКТ матрицей'
```

```
##########################################################################################

'применяем дкт'
coeffs_blocks_Y = np.array([dct_matrix(block) for block in blocks_Y])
coeffs_blocks_Cb = np.array([dct_matrix(block) for block in blocks_Cb])
coeffs_blocks_Cr = np.array([dct_matrix(block) for block in blocks_Cr])
##########################################################################################
'Записываем массивы фото после ДКТ в файл'
##########################################################################################
'Объединяем блоки'
Y_dct = merge_blocks(coeffs_blocks_Y, image_array_rgb.shape)
Cb_dct = merge_blocks(coeffs_blocks_Cb, image_array_rgb.shape)
Cr_dct = merge_blocks(coeffs_blocks_Cr, image_array_rgb.shape)

'Сохраняем txt'
coeffs_blocks = np.dstack((Y_dct, Cb_dct, Cr_dct))
save_image_to_txt(coeffs_blocks, '3_DCT_image.txt')
##########################################################################################

print(f"4. Шаг: Квантование матриц с Q=50")
'Квантование'
##########################################################################################
'Ставим качество 50 и создаем матрицу квантования'
quality = 50
quantization_matrix_quality = get_quantization_matrix(quality)

'Квантование'
quantization_coeffs_Y = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Y])
quantization_coeffs_Cb = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cb])
quantization_coeffs_Cr = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cr])
##########################################################################################
'Объединение блоков'
##########################################################################################
Y_merge = merge_blocks(quantization_coeffs_Y, image_array_rgb.shape)
Cb_merge = merge_blocks(quantization_coeffs_Cb, image_array_rgb.shape)
Cr_merge = merge_blocks(quantization_coeffs_Cr, image_array_rgb.shape)
##########################################################################################
'Записываем массивы фото после Квантования в файл'
##########################################################################################
merge_blocks_ = np.dstack((Y_merge, Cb_merge, Cr_merge))
save_image_to_txt(merge_blocks_, '4_Квантование_image.txt')
##########################################################################################

print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
##########################################################################################
zigzag_Y = zigzag(Y_merge)
zigzag_Cb = zigzag(Cb_merge)
zigzag_Cr = zigzag(Cr_merge)
##########################################################################################
'Записываем фото после Зиг Зага в файл'
##########################################################################################
zigzag_blocks = np.dstack((zigzag_Y, zigzag_Cb, zigzag_Cr))
save_image_to_txt(zigzag_blocks, '5_ZigZag_image.txt')
##########################################################################################
```

```
    print(f"6. Шаг: Сжатие с помощью RLE")
    'RLE'
    ##################################################################################################
    rle_Y = rle(zigzag_Y)
    rle_Cb = rle(zigzag_Cb)
    rle_Cr = rle(zigzag_Cr)
    ##################################################################################################
    'Записываем фото после РЛЕ в файл'
    ##################################################################################################
    rle_finish = rle_Y + "р" + rle_Cb + "р" + rle_Cr
    with open("6_RLE_code.txt", 'w', encoding='utf-8') as file:
        file.write(rle_finish)
    ##################################################################################################
    print(f"Сжатие завершено!")

    'Переписываем файлы txt в отдельную папку'
    ##################################################################################################
    'Путь к исходной папке'
    source_folder = 'C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv'

    'Создаем целевую папку, если её нет'
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    'Перебираем все файлы в исходной папке'
    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым файлом (например, имеет ли он расширение .txt)
        if filename.endswith('.txt'):
            # Формируем полные пути к файлу в исходной и целевой папках
            source_path = os.path.join(source_folder, filename)
            target_path = os.path.join(target_folder, filename)

            # Копируем файл из исходной папки в целевую
            shutil.copy2(source_path, target_path)

    print("Копирование завершено.")

    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым
        if filename.endswith('.txt'):
            # Формируем полный путь к файлу
            file_path = os.path.join(source_folder, filename)
            # Удаляем файл
            os.remove(file_path)
    ##################################################################################################


print(f"_____
_____\n\n")

    print(f"Проверка кодирования декодированием:")
    print(f"1. Шаг: Разжатие RLE")
    'декодер RLE'
    ##################################################################################################
    de_rle_Y = rle_decoder(rle_Y)
    de_rle_Cb = rle_decoder(rle_Cb)
    de_rle_Cr = rle_decoder(rle_Cr)
    ##################################################################################################
```

```
print(f"2. Шаг: Обрантный зиг заг")
'декодер RLE'
#############################################################################################
de_zigzag_Y = inverse_zigzag(de_rle_Y, height, width)
de_zigzag_Cb = inverse_zigzag(de_rle_Cb, height, width)
de_zigzag_Cr = inverse_zigzag(de_rle_Cr, height, width)
#############################################################################################

'разделение изображения на блоки 8x8'
#############################################################################################
de_blocks_Y = split_image(de_zigzag_Y)
de_blocks_Cb = split_image(de_zigzag_Cb)
de_blocks_Cr = split_image(de_zigzag_Cr)
#############################################################################################

print(f"3. Шаг: Обратное квантование")
'Обратное квантование матриц'
#############################################################################################
de_quantization_Y = de_quantization_dct(de_blocks_Y, quantization_matrix_quality)
de_quantization_Cb = de_quantization_dct(de_blocks_Cb, quantization_matrix_quality)
de_quantization_Cr = de_quantization_dct(de_blocks_Cr, quantization_matrix_quality)
#############################################################################################

print(f"4. Шаг: Обратное ДКТ")
'Обратное квантование матриц'
#############################################################################################
de_DCT_Y = np.array([idct(block) for block in de_quantization_Y])
de_DCT_Cb = np.array([idct(block) for block in de_quantization_Cb])
de_DCT_Cr = np.array([idct(block) for block in de_quantization_Cr])
#############################################################################################

'Объединение блоков матриц'
#############################################################################################
de_Y_merge = merge_blocks(de_DCT_Y, image_array_rgb.shape)
de_Cb_merge = merge_blocks(de_DCT_Cb, image_array_rgb.shape)
de_Cr_merge = merge_blocks(de_DCT_Cr, image_array_rgb.shape)
#############################################################################################


rgb_array_ = np.dstack((de_Y_merge, de_Cb_merge, de_Cr_merge))
rgb_array_ = rgb_array_.astype('uint8')
image_rgb_decoder = Image.fromarray(rgb_array_, 'RGB')
image_rgb_decoder.save('decode_image.jpg')

# проверка изображений на кодирование
fig, axs = plt.subplots(1, 2)
# Отображение изображений на Axes
axs[0].imshow(image)
axs[0].set_title("Изначальное изображение")
axs[1].imshow(image_rgb_decoder)
axs[1].set_title("Декодированное")
# Установка расстояния между изображениями
plt.tight_layout()
# Отображение Figure
fig.suptitle("СЖАТИЕ ИЗОБРАЖЕНИЯ")

plt.show()
```

2.11. Кодирование и декодирование без Сабсемплинга

```
def without_Subsempling_compress_and_decode_image(image_path, target_folder):
    "основная часть"

##########################################################################################################
#####################################################################
    'Считывание фото'
    ###############################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
    image_array_rgb = np.array(image_rgb)
    width, height = image.size
    ###############################################################################################
    'Записываем массив фото RGB в файл'
    ###############################################################################################
    save_image_to_txt(image_array_rgb, '0_RGB_image.txt')
    ###############################################################################################

    print(f"1. Шаг: Преобразование из RGB в YCbCr")
    'YCbCr'
    ###############################################################################################
    'массив в формате YCbCr'
    image_array_ycbcr = rgb_to_ycbcr(image_array_rgb)
    ###############################################################################################
    'Записываем массив фото YCbCrв файл'
    ###############################################################################################
    save_image_to_txt(image_array_ycbcr, '1_YCbCr_image.txt')
    ###############################################################################################

    'YCbCr по массивам'
    ###############################################################################################
    Y, Cb, Cr = image_array_ycbcr[:, :, 0], image_array_ycbcr[:, :, 1], image_array_ycbcr[:, :, 2]
    ###############################################################################################


    'разделение изображения на блоки 8x8'
    ###############################################################################################
    blocks_Y = split_image(Y)
    blocks_Cb = split_image(Cb)
    blocks_Cr = split_image(Cr)
    ###############################################################################################

    print(f"3. Шаг: Применение ДКТ матрицей")
    'ДКТ матрицей'
    ###############################################################################################

    'применяем дкт'
    coeffs_blocks_Y = np.array([dct_matrix(block) for block in blocks_Y])
    coeffs_blocks_Cb = np.array([dct_matrix(block) for block in blocks_Cb])
    coeffs_blocks_Cr = np.array([dct_matrix(block) for block in blocks_Cr])
    ###############################################################################################
    'Записываем массивы фото после ДКТ в файл'
```

```
##################################################################################################
'Объединяем блоки'
Y_dct = merge_blocks(coeffs_blocks_Y, image_array_rgb.shape)
Cb_dct = merge_blocks(coeffs_blocks_Cb, image_array_rgb.shape)
Cr_dct = merge_blocks(coeffs_blocks_Cr, image_array_rgb.shape)

'Сохраняем txt'
coeffs_blocks = np.dstack((Y_dct, Cb_dct, Cr_dct))
save_image_to_txt(coeffs_blocks, '3_DCT_image.txt')
##################################################################################################

print(f"4. Шаг: Квантование матриц с Q=50")
'Квантование'
##################################################################################################
'Ставим качество 50 и создаем матрицу квантования'
quality = 50
quantization_matrix_quality = get_quantization_matrix(quality)

'Квантование'
quantization_coeffs_Y = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Y])
quantization_coeffs_Cb = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cb])
quantization_coeffs_Cr = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cr])
##################################################################################################
'Объединение блоков'
##################################################################################################
Y_merge = merge_blocks(quantization_coeffs_Y, image_array_rgb.shape)
Cb_merge = merge_blocks(quantization_coeffs_Cb, image_array_rgb.shape)
Cr_merge = merge_blocks(quantization_coeffs_Cr, image_array_rgb.shape)
##################################################################################################
'Записываем массивы фото после Квантования в файл'
##################################################################################################
merge_blocks_ = np.dstack((Y_merge, Cb_merge, Cr_merge))
save_image_to_txt(merge_blocks_, '4_Квантование_image.txt')
##################################################################################################

print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
##################################################################################################
zigzag_Y = zigzag(Y_merge)
zigzag_Cb = zigzag(Cb_merge)
zigzag_Cr = zigzag(Cr_merge)
##################################################################################################
'Записываем фото после Зиг Зага в файл'
##################################################################################################
zigzag_blocks = np.dstack((zigzag_Y, zigzag_Cb, zigzag_Cr))
save_image_to_txt(zigzag_blocks, '5_ZigZag_image.txt')
##################################################################################################

print(f"6. Шаг: Сжатие с помощью RLE")
'RLE'
##################################################################################################
rle_Y = rle(zigzag_Y)
rle_Cb = rle(zigzag_Cb)
rle_Cr = rle(zigzag_Cr)
##################################################################################################
'Записываем фото после РЛЕ в файл'
```

```
    ##########################################################################################
    rle_finish = rle_Y + "p" + rle_Cb + "p" + rle_Cr
    with open("6_RLE_code.txt", 'w', encoding='utf-8') as file:
        file.write(rle_finish)
    ##########################################################################################
    print(f"Сжатие завершено!")

    'Переписываем файлы txt в отдельную папку'
    ##########################################################################################
    'Путь к исходной папке'
    source_folder = 'C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv'

    'Создаем целевую папку, если её нет'
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    'Перебираем все файлы в исходной папке'
    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым файлом (например, имеет ли он расширение .txt)
        if filename.endswith('.txt'):
            # Формируем полные пути к файлу в исходной и целевой папках
            source_path = os.path.join(source_folder, filename)
            target_path = os.path.join(target_folder, filename)

            # Копируем файл из исходной папки в целевую
            shutil.copy2(source_path, target_path)

    print("Копирование завершено.")

    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым
        if filename.endswith('.txt'):
            # Формируем полный путь к файлу
            file_path = os.path.join(source_folder, filename)
            # Удаляем файл
            os.remove(file_path)
    ##########################################################################################


print(f"_____
_____\n\n")

    print(f"Проверка кодирования декодированием:")
    print(f"1. Шаг: Разжатие RLE")
    'декодер RLE'
    ##########################################################################################
    de_rle_Y = rle_decoder(rle_Y)
    de_rle_Cb = rle_decoder(rle_Cb)
    de_rle_Cr = rle_decoder(rle_Cr)
    ##########################################################################################

    print(f"2. Шаг: Обрантный зиг заг")
    'декодер RLE'
    ##########################################################################################
    de_zigzag_Y = inverse_zigzag(de_rle_Y, height, width)
    de_zigzag_Cb = inverse_zigzag(de_rle_Cb, height, width)
    de_zigzag_Cr = inverse_zigzag(de_rle_Cr, height, width)
    ##########################################################################################
```

```
'разделение изображения на блоки 8x8'
#################################################################################################
de_blocks_Y = split_image(de_zigzag_Y)
de_blocks_Cb = split_image(de_zigzag_Cb)
de_blocks_Cr = split_image(de_zigzag_Cr)
#################################################################################################

print(f"3. Шаг: Обратное квантование")
'Обратное квантование матриц'
#################################################################################################
de_quantization_Y = de_quantization_dct(de_blocks_Y, quantization_matrix_quality)
de_quantization_Cb = de_quantization_dct(de_blocks_Cb, quantization_matrix_quality)
de_quantization_Cr = de_quantization_dct(de_blocks_Cr, quantization_matrix_quality)
#################################################################################################

print(f"4. Шаг: Обратное ДКТ")
'Обратное квантование матриц'
#################################################################################################
de_DCT_Y = np.array([idct(block) for block in de_quantization_Y])
de_DCT_Cb = np.array([idct(block) for block in de_quantization_Cb])
de_DCT_Cr = np.array([idct(block) for block in de_quantization_Cr])
#################################################################################################

'Объединение блоков матриц'
#################################################################################################
de_Y_merge = merge_blocks(de_DCT_Y, image_array_rgb.shape)
de_Cb_merge = merge_blocks(de_DCT_Cb, image_array_rgb.shape)
de_Cr_merge = merge_blocks(de_DCT_Cr, image_array_rgb.shape)
#################################################################################################

print(f"5. Шаг: Декодирование в RGB")
'Обратное кодирование в RGB'
#################################################################################################
rgb_array = ycbcr_to_rgb(de_Y_merge, de_Cb_merge, de_Cr_merge)
#################################################################################################

image_rgb_decoder = Image.fromarray(rgb_array, 'RGB')
image_rgb_decoder.save('decode_image.jpg')

# проверка изображений на кодирование
fig, axs = plt.subplots(1, 2)
# Отображение изображений на Axes
axs[0].imshow(image)
axs[0].set_title("Изначальное изображение")
axs[1].imshow(image_rgb_decoder)
axs[1].set_title("Декодированное")
# Установка расстояния между изображениями
plt.tight_layout()
# Отображение Figure
fig.suptitle("СЖАТИЕ ИЗОБРАЖЕНИЯ")

plt.show()
```

2.12. Кодирование и декодирование без ДКТ

```
def without_DCT_compress_and_decode_image(image_path, target_folder):
    "основная часть"

##########################################################################################################
#####################################################################

    'Считывание фото'
    ###############################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
    image_array_rgb = np.array(image_rgb)
    width, height = image.size
    ###############################################################################################
    'Записываем массив фото RGB в файл'
    ###############################################################################################
    save_image_to_txt(image_array_rgb, '0_RGB_image.txt')
    ###############################################################################################

    print(f"1. Шаг: Преобразование из RGB в YCbCr")
    'YCbCr'
    ###############################################################################################
    'массив в формате YCbCr'
    image_array_ycbcr = rgb_to_ycbcr(image_array_rgb)
    ###############################################################################################
    'Записываем массив фото YCbCrв файл'
    ###############################################################################################
    save_image_to_txt(image_array_ycbcr, '1_YCbCr_image.txt')
    ###############################################################################################

    'YCbCr по массивам'
    ###############################################################################################
    Y, Cb, Cr = image_array_ycbcr[:, :, 0], image_array_ycbcr[:, :, 1], image_array_ycbcr[:, :, 2]
    ###############################################################################################

    print(f"2. Шаг: Сабсемплинг матриц")
    'сабсемплинг'
    ###############################################################################################
    "коэф. сжатия"
    compression_h = 10
    compression_w = 10

    "даунсемплинг Cb Cr 2 методом даунсемплинга"
    downsampling_matrix_Cb = downsampling(Cb, compression_h, compression_w, 2)
    downsampling_matrix_Cr = downsampling(Cr, compression_h, compression_w, 2)

    "апсемплинг Cb Cr, чтобы вернуть к размеру Y"
    Cb = upsampling_image(downsampling_matrix_Cb, compression_h, compression_w)
    Cr = upsampling_image(downsampling_matrix_Cr, compression_h, compression_w)
    ###############################################################################################
    'Записываем массивы фото после сабсемплинга в файл'
    ###############################################################################################
```

```
ycbcr_sub = np.dstack((Y, Cb, Cr))
save_image_to_txt(ycbcr_sub, '2_subsempling_image.txt')
#########################################################################################

'разделение изображения на блоки 8x8'
#########################################################################################
blocks_Y = split_image(Y)
blocks_Cb = split_image(Cb)
blocks_Cr = split_image(Cr)
#########################################################################################

print(f"4. Шаг: Квантование матриц с Q=50")
'Квантование'
#########################################################################################
'Ставим качество 50 и создаем матрицу квантования'
quality = 50
quantization_matrix_quality = get_quantization_matrix(quality)

'Квантование'
quantization_coeffs_Y = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in blocks_Y])
quantization_coeffs_Cb = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in blocks_Cb])
quantization_coeffs_Cr = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in blocks_Cr])
#########################################################################################
'Объединение блоков'
#########################################################################################
Y_merge = merge_blocks(quantization_coeffs_Y, image_array_rgb.shape)
Cb_merge = merge_blocks(quantization_coeffs_Cb, image_array_rgb.shape)
Cr_merge = merge_blocks(quantization_coeffs_Cr, image_array_rgb.shape)
#########################################################################################
'Записываем массивы фото после Квантования в файл'
#########################################################################################
merge_blocks_ = np.dstack((Y_merge, Cb_merge, Cr_merge))
save_image_to_txt(merge_blocks_, '4_Квантование_image.txt')
#########################################################################################

print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
#########################################################################################
zigzag_Y = zigzag(Y_merge)
zigzag_Cb = zigzag(Cb_merge)
zigzag_Cr = zigzag(Cr_merge)
#########################################################################################
'Записываем фото после Зиг Зага в файл'
#########################################################################################
zigzag_blocks = np.dstack((zigzag_Y, zigzag_Cb, zigzag_Cr))
save_image_to_txt(zigzag_blocks, '5_ZigZag_image.txt')
#########################################################################################

print(f"6. Шаг: Сжатие с помощью RLE")
'RLE'
#########################################################################################
rle_Y = rle(zigzag_Y)
rle_Cb = rle(zigzag_Cb)
rle_Cr = rle(zigzag_Cr)
#########################################################################################
'Записываем фото после РЛЕ в файл'
```

```
    ##########################################################################################
    rle_finish = rle_Y + "p" + rle_Cb + "p" + rle_Cr
    with open("6_RLE_code.txt", 'w', encoding='utf-8') as file:
        file.write(rle_finish)
    ##########################################################################################
    print(f"Сжатие завершено!")

    'Переписываем файлы txt в отдельную папку'
    ##########################################################################################
    'Путь к исходной папке'
    source_folder = 'C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv'

    'Создаем целевую папку, если её нет'
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    'Перебираем все файлы в исходной папке'
    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым файлом (например, имеет ли он расширение .txt)
        if filename.endswith('.txt'):
            # Формируем полные пути к файлу в исходной и целевой папках
            source_path = os.path.join(source_folder, filename)
            target_path = os.path.join(target_folder, filename)

            # Копируем файл из исходной папки в целевую
            shutil.copy2(source_path, target_path)

    print("Копирование завершено.")

    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым
        if filename.endswith('.txt'):
            # Формируем полный путь к файлу
            file_path = os.path.join(source_folder, filename)
            # Удаляем файл
            os.remove(file_path)
    ##########################################################################################


print(f"_____
_____\n\n")

    print(f"Проверка кодирования декодированием:")
    print(f"1. Шаг: Разжатие RLE")
    'декодер RLE'
    ##########################################################################################
    de_rle_Y = rle_decoder(rle_Y)
    de_rle_Cb = rle_decoder(rle_Cb)
    de_rle_Cr = rle_decoder(rle_Cr)
    ##########################################################################################

    print(f"2. Шаг: Обрантный зиг заг")
    'декодер RLE'
    ##########################################################################################
    de_zigzag_Y = inverse_zigzag(de_rle_Y, height, width)
    de_zigzag_Cb = inverse_zigzag(de_rle_Cb, height, width)
    de_zigzag_Cr = inverse_zigzag(de_rle_Cr, height, width)
    ##########################################################################################
```

```
'разделение изображения на блоки 8x8'
#########################################################################################
de_blocks_Y = split_image(de_zigzag_Y)
de_blocks_Cb = split_image(de_zigzag_Cb)
de_blocks_Cr = split_image(de_zigzag_Cr)
#########################################################################################

print(f"3. Шаг: Обратное квантование")
'Обратное квантование матриц'
#########################################################################################
de_quantization_Y = de_quantization_dct(de_blocks_Y, quantization_matrix_quality)
de_quantization_Cb = de_quantization_dct(de_blocks_Cb, quantization_matrix_quality)
de_quantization_Cr = de_quantization_dct(de_blocks_Cr, quantization_matrix_quality)
#########################################################################################


'Объединение блоков матриц'
#########################################################################################
de_Y_merge = merge_blocks(de_quantization_Y, image_array_rgb.shape)
de_Cb_merge = merge_blocks(de_quantization_Cb, image_array_rgb.shape)
de_Cr_merge = merge_blocks(de_quantization_Cr, image_array_rgb.shape)
#########################################################################################

print(f"5. Шаг: Декодирование в RGB")
'Обратное кодирование в RGB'
#########################################################################################
rgb_array = ycbcr_to_rgb(de_Y_merge, de_Cb_merge, de_Cr_merge)
#########################################################################################

image_rgb_decoder = Image.fromarray(rgb_array, 'RGB')
image_rgb_decoder.save('decode_image.jpg')

# проверка изображений на кодирование
fig, axs = plt.subplots(1, 2)
# Отображение изображений на Axes
axs[0].imshow(image)
axs[0].set_title("Изначальное изображение")
axs[1].imshow(image_rgb_decoder)
axs[1].set_title("Декодированное")
# Установка расстояния между изображениями
plt.tight_layout()
# Отображение Figure
fig.suptitle("СЖАТИЕ ИЗОБРАЖЕНИЯ")

plt.show()
```

2.13. Кодирование и декодирование без Квантирования

```
def without_quant_compress_and_decode_image(image_path, target_folder):
    "основная часть"

##########################################################################################################
#########################################################################
    'Считывание фото'
    ###############################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
    image_array_rgb = np.array(image_rgb)
    width, height = image.size
    ###############################################################################################
    'Записываем массив фото RGB в файл'
    ###############################################################################################
    save_image_to_txt(image_array_rgb, '0_RGB_image.txt')
    ###############################################################################################

    print(f"1. Шаг: Преобразование из RGB в YCbCr")
    'YCbCr'
    ###############################################################################################
    'массив в формате YCbCr'
    image_array_ycbcr = rgb_to_ycbcr(image_array_rgb)
    ###############################################################################################
    'Записываем массив фото YCbCrв файл'
    ###############################################################################################
    save_image_to_txt(image_array_ycbcr, '1_YCbCr_image.txt')
    ###############################################################################################

    'YCbCr по массивам'
    ###############################################################################################
    Y, Cb, Cr = image_array_ycbcr[:, :, 0], image_array_ycbcr[:, :, 1], image_array_ycbcr[:, :, 2]
    ###############################################################################################

    print(f"2. Шаг: Сабсемплинг матриц")
    'сабсемплинг'
    ###############################################################################################
    "коэф. сжатия"
    compression_h = 10
    compression_w = 10

    "даунсемплинг Cb Cr 2 методом даунсемплинга"
    downsampling_matrix_Cb = downsampling(Cb, compression_h, compression_w, 2)
    downsampling_matrix_Cr = downsampling(Cr, compression_h, compression_w, 2)

    "апсемплинг Cb Cr, чтобы вернуть к размеру Y"
    Cb = upsampling_image(downsampling_matrix_Cb, compression_h, compression_w)
    Cr = upsampling_image(downsampling_matrix_Cr, compression_h, compression_w)
    ###############################################################################################
    'Записываем массивы фото после сабсемплинга в файл'
    ###############################################################################################
    ycbcr_sub = np.dstack((Y, Cb, Cr))
    save_image_to_txt(ycbcr_sub, '2_subsempling_image.txt')
    ###############################################################################################

    'разделение изображения на блоки 8x8'
```

```
##########################################################################################
blocks_Y = split_image(Y)
blocks_Cb = split_image(Cb)
blocks_Cr = split_image(Cr)
##########################################################################################

print(f"3. Шаг: Применение ДКТ матрицей")
'ДКТ матрицей'
##########################################################################################

'применяем дкт'
coeffs_blocks_Y = np.array([dct_matrix(block) for block in blocks_Y])
coeffs_blocks_Cb = np.array([dct_matrix(block) for block in blocks_Cb])
coeffs_blocks_Cr = np.array([dct_matrix(block) for block in blocks_Cr])
##########################################################################################
'Записываем массивы фото после ДКТ в файл'
##########################################################################################
'Объединяем блоки'
Y_dct = merge_blocks(coeffs_blocks_Y, image_array_rgb.shape)
Cb_dct = merge_blocks(coeffs_blocks_Cb, image_array_rgb.shape)
Cr_dct = merge_blocks(coeffs_blocks_Cr, image_array_rgb.shape)

'Сохраняем txt'
coeffs_blocks = np.dstack((Y_dct, Cb_dct, Cr_dct))
save_image_to_txt(coeffs_blocks, '3_DCT_image.txt')
##########################################################################################

print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
##########################################################################################
zigzag_Y = zigzag(Y_dct)
zigzag_Cb = zigzag(Cb_dct)
zigzag_Cr = zigzag(Cr_dct)
##########################################################################################
'Записываем фото после Зиг Зага в файл'
##########################################################################################
zigzag_blocks = np.dstack((zigzag_Y, zigzag_Cb, zigzag_Cr))
save_image_to_txt(zigzag_blocks, '5_ZigZag_image.txt')
##########################################################################################

print(f"6. Шаг: Сжатие с помощью RLE")
'RLE'
##########################################################################################
rle_Y = rle(zigzag_Y)
rle_Cb = rle(zigzag_Cb)
rle_Cr = rle(zigzag_Cr)
##########################################################################################
'Записываем фото после РЛЕ в файл'
##########################################################################################
rle_finish = rle_Y + "р" + rle_Cb + "р" + rle_Cr
with open("6_RLE_code.txt", 'w', encoding='utf-8') as file:
    file.write(rle_finish)
##########################################################################################
print(f"Сжатие завершено!")

'Переписываем файлы txt в отдельную папку'
##########################################################################################
'Путь к исходной папке'
source_folder = 'C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv'
```

```
    'Создаем целевую папку, если её нет'
    if not os.path.exists(target_folder):
        os.makedirs(target_folder)

    'Перебираем все файлы в исходной папке'
    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым файлом (например, имеет ли он расширение .txt)
        if filename.endswith('.txt'):
            # Формируем полные пути к файлу в исходной и целевой папках
            source_path = os.path.join(source_folder, filename)
            target_path = os.path.join(target_folder, filename)

            # Копируем файл из исходной папки в целевую
            shutil.copy2(source_path, target_path)

    print("Копирование завершено.")

    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым
        if filename.endswith('.txt'):
            # Формируем полный путь к файлу
            file_path = os.path.join(source_folder, filename)
            # Удаляем файл
            os.remove(file_path)
    ##########################################################################################


print(f"_____
_____\n\n")

    print(f"Проверка кодирования декодированием:")
    print(f"1. Шаг: Разжатие RLE")
    'декодер RLE'
    ##########################################################################################
    de_rle_Y = rle_decoder(rle_Y)
    de_rle_Cb = rle_decoder(rle_Cb)
    de_rle_Cr = rle_decoder(rle_Cr)
    ##########################################################################################

    print(f"2. Шаг: Обрантный зиг заг")
    'декодер RLE'
    ##########################################################################################
    de_zigzag_Y = inverse_zigzag(de_rle_Y, height, width)
    de_zigzag_Cb = inverse_zigzag(de_rle_Cb, height, width)
    de_zigzag_Cr = inverse_zigzag(de_rle_Cr, height, width)
    ##########################################################################################

    'разделение изображения на блоки 8x8'
    ##########################################################################################
    de_blocks_Y = split_image(de_zigzag_Y)
    de_blocks_Cb = split_image(de_zigzag_Cb)
    de_blocks_Cr = split_image(de_zigzag_Cr)
    ##########################################################################################

    print(f"4. Шаг: Обратное ДКТ")
    'Обратное квантование матриц'
    ##########################################################################################
    de_DCT_Y = np.array([idct(block) for block in de_blocks_Y])
```

```
    de_DCT_Cb = np.array([idct(block) for block in de_blocks_Cb])
    de_DCT_Cr = np.array([idct(block) for block in de_blocks_Cr])
    ##########################################################################################

    'Объединение блоков матриц'
    ##########################################################################################
    de_Y_merge = merge_blocks(de_DCT_Y, image_array_rgb.shape)
    de_Cb_merge = merge_blocks(de_DCT_Cb, image_array_rgb.shape)
    de_Cr_merge = merge_blocks(de_DCT_Cr, image_array_rgb.shape)
    ##########################################################################################

    print(f"5. Шаг: Декодирование в RGB")
    'Обратное кодирование в RGB'
    ##########################################################################################
    rgb_array = ycbcr_to_rgb(de_Y_merge, de_Cb_merge, de_Cr_merge)
    ##########################################################################################

    image_rgb_decoder = Image.fromarray(rgb_array, 'RGB')
    image_rgb_decoder.save('decode_image.jpg')

    # проверка изображений на кодирование
    fig, axs = plt.subplots(1, 2)
    # Отображение изображений на Axes
    axs[0].imshow(image)
    axs[0].set_title("Изначальное изображение")
    axs[1].imshow(image_rgb_decoder)
    axs[1].set_title("Декодированное")
    # Установка расстояния между изображениями
    plt.tight_layout()
    # Отображение Figure
    fig.suptitle("СЖАТИЕ ИЗОБРАЖЕНИЯ")

    plt.show()
```

2.14. Кодирование и декодирование, с целью посмотреть размер фото в зависимости от Q

```
def quantization_compress_and_decode_image(image_path, target_folder):
    "основная часть"

##########################################################################################################
#########################################################################
    'Считывание фото'
    ##########################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
```

```
image_array_rgb = np.array(image_rgb)
width, height = image.size
#########################################################################################
'Записываем массив фото RGB в файл'
#########################################################################################
save_image_to_txt(image_array_rgb, '0_RGB_image.txt')
#########################################################################################

'YCbCr по массивам'
#########################################################################################
Y, Cb, Cr = image_array_rgb[:, :, 0], image_array_rgb[:, :, 1], image_array_rgb[:, :, 2]
#########################################################################################

print(f"2. Шаг: Сабсемплинг матриц")
'сабсемплинг'
#########################################################################################
"коэф. сжатия"
compression_h = 2
compression_w = 2

"даунсемплинг Cb Cr 2 методом даунсемплинга"
downsampling_matrix_Y = downsampling(Y, compression_h, compression_w, 2)
downsampling_matrix_Cb = downsampling(Cb, compression_h, compression_w, 2)
downsampling_matrix_Cr = downsampling(Cr, compression_h, compression_w, 2)

"апсемплинг Cb Cr, чтобы вернуть к размеру Y"
Y = upsampling_image(downsampling_matrix_Y, compression_h, compression_w)
Cb = upsampling_image(downsampling_matrix_Cb, compression_h, compression_w)
Cr = upsampling_image(downsampling_matrix_Cr, compression_h, compression_w)
#########################################################################################

'разделение изображения на блоки 8x8'
#########################################################################################
blocks_Y = split_image(Y)
blocks_Cb = split_image(Cb)
blocks_Cr = split_image(Cr)
#########################################################################################

print(f"3. Шаг: Применение ДКТ матрицей")
'ДКТ матрицей'
#########################################################################################

'применяем дкт'
coeffs_blocks_Y = np.array([dct_matrix(block) for block in blocks_Y])
coeffs_blocks_Cb = np.array([dct_matrix(block) for block in blocks_Cb])
coeffs_blocks_Cr = np.array([dct_matrix(block) for block in blocks_Cr])
#########################################################################################
quality = 95
for i in range(0, 18):

    print(f"4. Шаг: Квантование матриц с Q={quality}",)
    'Квантование'
    #####################################################################################
    'Ставим качество 50 и создаем матрицу квантования'
    quantization_matrix_quality = get_quantization_matrix(quality)

    'Квантование'
    quantization_coeffs_Y = np.array(
        [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Y])
```

```
quantization_coeffs_Cb = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cb])
quantization_coeffs_Cr = np.array(
    [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cr])
########################################################################################################
'Объединение блоков'
########################################################################################################
Y_merge = merge_blocks(quantization_coeffs_Y, image_array_rgb.shape)
Cb_merge = merge_blocks(quantization_coeffs_Cb, image_array_rgb.shape)
Cr_merge = merge_blocks(quantization_coeffs_Cr, image_array_rgb.shape)
########################################################################################################


print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
########################################################################################################
zigzag_Y = zigzag(Y_merge)
zigzag_Cb = zigzag(Cb_merge)
zigzag_Cr = zigzag(Cr_merge)
########################################################################################################

print(f"6. Шаг: Сжатие с помощью RLE")
'RLE'
########################################################################################################
rle_Y = rle(zigzag_Y)
rle_Cb = rle(zigzag_Cb)
rle_Cr = rle(zigzag_Cr)
########################################################################################################
'Записываем фото после РЛЕ в файл'
########################################################################################################
rle_finish = rle_Y + "p" + rle_Cb + "p" + rle_Cr
name = f"RLE_{quality}_code.txt"
with open(name, 'w', encoding='utf-8') as file:
    file.write(rle_finish)
########################################################################################################
print(f"Сжатие завершено!")
quality-=5
'Переписываем файлы txt в отдельную папку'
########################################################################################################
'Путь к исходной папке'
source_folder = 'C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv'

'Создаем целевую папку, если её нет'
if not os.path.exists(target_folder):
    os.makedirs(target_folder)

'Перебираем все файлы в исходной папке'
for filename in os.listdir(source_folder):
    # Проверяем, является ли файл текстовым файлом (например, имеет ли он расширение .txt)
    if filename.endswith('.txt'):
        # Формируем полные пути к файлу в исходной и целевой папках
        source_path = os.path.join(source_folder, filename)
        target_path = os.path.join(target_folder, filename)

        # Копируем файл из исходной папки в целевую
        shutil.copy2(source_path, target_path)

print("Копирование завершено.")
```

```
    for filename in os.listdir(source_folder):
        # Проверяем, является ли файл текстовым
        if filename.endswith('.txt'):
            # Формируем полный путь к файлу
            file_path = os.path.join(source_folder, filename)
            # Удаляем файл
            os.remove(file_path)
#########################################################################################
```

2.15. Кодирование и декодирование, с целью посмотреть качество фото в зависимости от Q

```
def compress_and_decode_image(image_path, target_folder, Q):
    "основная часть"

#########################################################################################################
#########################################################################
    'Считывание фото'
    ###########################################################################################
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
    image_array_rgb = np.array(image_rgb)
    width, height = image.size
    ###########################################################################################


    print(f"1. Шаг: Преобразование из RGB в YCbCr")
    'YCbCr'
    ###########################################################################################
    'массив в формате YCbCr'
    image_array_ycbcr = rgb_to_ycbcr(image_array_rgb)
    ###########################################################################################


    'YCbCr по массивам'
    ###########################################################################################
    Y, Cb, Cr = image_array_ycbcr[:, :, 0], image_array_ycbcr[:, :, 1], image_array_ycbcr[:, :, 2]
    ###########################################################################################

    print(f"2. Шаг: Сабсемплинг матриц")
    'сабсемплинг'
    ###########################################################################################
    "коэф. сжатия"
    compression_h = 10
    compression_w = 10

    "даунсемплинг Cb Cr 2 методом даунсемплинга"
```

```
downsampling_matrix_Cb = downsampling(Cb, compression_h, compression_w, 2)
downsampling_matrix_Cr = downsampling(Cr, compression_h, compression_w, 2)

"апсемплинг Cb Cr, чтобы вернуть к размеру Y"
Cb = upsampling_image(downsampling_matrix_Cb, compression_h, compression_w)
Cr = upsampling_image(downsampling_matrix_Cr, compression_h, compression_w)
###############################################################################################


'разделение изображения на блоки 8x8'
###############################################################################################
blocks_Y = split_image(Y)
blocks_Cb = split_image(Cb)
blocks_Cr = split_image(Cr)
###############################################################################################

print(f"3. Шаг: Применение ДКТ матрицей")
'ДКТ матрицей'
###############################################################################################

'применяем дкт'
coeffs_blocks_Y = np.array([dct_matrix(block) for block in blocks_Y])
coeffs_blocks_Cb = np.array([dct_matrix(block) for block in blocks_Cb])
coeffs_blocks_Cr = np.array([dct_matrix(block) for block in blocks_Cr])
###############################################################################################

print(f"4. Шаг: Квантование матриц с Q=50")
'Квантование'
###############################################################################################
'Ставим качество 50 и создаем матрицу квантования'

quantization_matrix_quality = get_quantization_matrix(Q)

'Квантование'
quantization_coeffs_Y = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Y])
quantization_coeffs_Cb = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cb])
quantization_coeffs_Cr = np.array(
   [quantization_dct(block, quantization_matrix_quality) for block in coeffs_blocks_Cr])
###############################################################################################
'Объединение блоков'
###############################################################################################
Y_merge = merge_blocks(quantization_coeffs_Y, image_array_rgb.shape)
Cb_merge = merge_blocks(quantization_coeffs_Cb, image_array_rgb.shape)
Cr_merge = merge_blocks(quantization_coeffs_Cr, image_array_rgb.shape)
###############################################################################################


print(f"5. Шаг: Преобразоваие матриц в строку с помощью обхода зиг-загом")
'Зиг заг'
###############################################################################################
zigzag_Y = zigzag(Y_merge)
zigzag_Cb = zigzag(Cb_merge)
zigzag_Cr = zigzag(Cr_merge)
###############################################################################################


print(f"6. Шаг: Сжатие с помощью RLE")
```

```
    'RLE'
    ##########################################################################################
    rle_Y = rle(zigzag_Y)
    rle_Cb = rle(zigzag_Cb)
    rle_Cr = rle(zigzag_Cr)
    ##########################################################################################

    print(f"Сжатие завершено!")

    'Переписываем файлы txt в отдельную папку'
    ##########################################################################################

    ##########################################################################################


print(f"_____
_____\n\n")

    print(f"Проверка кодирования декодированием:")
    print(f"1. Шаг: Разжатие RLE")
    'декодер RLE'
    ##########################################################################################
    de_rle_Y = rle_decoder(rle_Y)
    de_rle_Cb = rle_decoder(rle_Cb)
    de_rle_Cr = rle_decoder(rle_Cr)
    ##########################################################################################

    print(f"2. Шаг: Обрантный зиг заг")
    'декодер RLE'
    ##########################################################################################
    de_zigzag_Y = inverse_zigzag(de_rle_Y, height, width)
    de_zigzag_Cb = inverse_zigzag(de_rle_Cb, height, width)
    de_zigzag_Cr = inverse_zigzag(de_rle_Cr, height, width)
    ##########################################################################################

    'разделение изображения на блоки 8x8'
    ##########################################################################################
    de_blocks_Y = split_image(de_zigzag_Y)
    de_blocks_Cb = split_image(de_zigzag_Cb)
    de_blocks_Cr = split_image(de_zigzag_Cr)
    ##########################################################################################

    print(f"3. Шаг: Обратное квантование")
    'Обратное квантование матриц'
    ##########################################################################################
    de_quantization_Y = de_quantization_dct(de_blocks_Y, quantization_matrix_quality)
    de_quantization_Cb = de_quantization_dct(de_blocks_Cb, quantization_matrix_quality)
    de_quantization_Cr = de_quantization_dct(de_blocks_Cr, quantization_matrix_quality)
    ##########################################################################################

    print(f"4. Шаг: Обратное ДКТ")
    'Обратное квантование матриц'
    ##########################################################################################
    de_DCT_Y = np.array([idct(block) for block in de_quantization_Y])
    de_DCT_Cb = np.array([idct(block) for block in de_quantization_Cb])
    de_DCT_Cr = np.array([idct(block) for block in de_quantization_Cr])
    ##########################################################################################

    'Объединение блоков матриц'
```

```
    ##########################################################################################
    de_Y_merge = merge_blocks(de_DCT_Y, image_array_rgb.shape)
    de_Cb_merge = merge_blocks(de_DCT_Cb, image_array_rgb.shape)
    de_Cr_merge = merge_blocks(de_DCT_Cr, image_array_rgb.shape)
    ##########################################################################################

    print(f"5. Шаг: Декодирование в RGB")
    'Обратное кодирование в RGB'
    ##########################################################################################
    rgb_array = ycbcr_to_rgb(de_Y_merge, de_Cb_merge, de_Cr_merge)
    ##########################################################################################

    image_rgb_decoder = Image.fromarray(rgb_array, 'RGB')
    image_rgb_decoder.save('decode_image.jpg')

    # проверка изображений на кодирование
    fig, axs = plt.subplots(1, 2)
    # Отображение изображений на Axes
    axs[0].imshow(image)
    axs[0].set_title("Изначальное изображение")
    axs[1].imshow(image_rgb_decoder)
    axs[1].set_title("Декодированное")
    # Установка расстояния между изображениями
    plt.tight_layout()
    # Отображение Figure
    fig.suptitle("СЖАТИЕ ИЗОБРАЖЕНИЯ")

    plt.show()
```

2.13. Проверка работы ДКТ матрицами и обычным ДКТ

```
def dct_check(image_path):
    'проверка работы ДКТ с помощью матрицы'

##########################################################################################################
#########################################################################
    """
    ---------------------------------------------------------------------------------------------
    Время выполнения dct_2: 4 секунд
    Время выполнения dct: 207 секунд
    ДКТ с помощью матрицы работает быстрее обычного ДКТ в: 47 раз
    ---------------------------------------------------------------------------------------------
    """
    print(f"Проверка работы ДКТ с помощью матрицы")
    print(f"-----------------------------------------------------------------------------------------------")

    #считываем изображение
    image = Image.open(image_path)
    image_rgb = image.convert("RGB")
```

```
# создаем массив RGB
image_array_rgb = np.array(image_rgb)

#массивы по цветам
R, G, B = image_array_rgb[:, :, 0], image_array_rgb[:, :, 1], image_array_rgb[:, :, 2]

#разделение изображения на блоки 8x8
blocks_R = split_image(R)
blocks_G = split_image(G)
blocks_B = split_image(B)

#засекаем время для ДКТ матрицей
start_time = time.time()
#применяем дкт
coeffs_blocks_R = np.array([dct_matrix(block) for block in blocks_R])
coeffs_blocks_G = np.array([dct_matrix(block) for block in blocks_G])
coeffs_blocks_B = np.array([dct_matrix(block) for block in blocks_B])

#декодируем дкт
de_coeffs_blocks_R = np.array([idct(block) for block in coeffs_blocks_R])
de_coeffs_blocks_G = np.array([idct(block) for block in coeffs_blocks_G])
de_coeffs_blocks_B = np.array([idct(block) for block in coeffs_blocks_B])
dct_matrix_time = time.time() - start_time


#засекаем время для ДКТ
start_time = time.time()

#применяем ДКТ для всех болков всех цветов
coeffs_blocks_R_1 = np.array([dct(block) for block in blocks_R])
coeffs_blocks_G_1 = np.array([dct(block) for block in blocks_G])
coeffs_blocks_B_1 = np.array([dct(block) for block in blocks_B])

#применяем декодер ДКТ для всех болков всех цветов
de_coeffs_blocks_R_ = np.array([revers_dct(block) for block in coeffs_blocks_R_1])
de_coeffs_blocks_G_ = np.array([revers_dct(block) for block in coeffs_blocks_G_1])
de_coeffs_blocks_B_ = np.array([revers_dct(block) for block in coeffs_blocks_B_1])

dct_time = time.time() - start_time


#объединение всех блоков в массивы цветов
R_=merge_blocks(de_coeffs_blocks_R, image_array_rgb.shape)
G_=merge_blocks(de_coeffs_blocks_G, image_array_rgb.shape)
B_=merge_blocks(de_coeffs_blocks_B, image_array_rgb.shape)

#объединение массивов цветов в изображение
rgb_image_array_ = np.dstack((R_, G_, B_))
rgb_image_array_=rgb_image_array_.astype(np.uint8)
image_rgb_ = Image.fromarray(rgb_image_array_, 'RGB')


#объединение всех блоков в массивы цветов
R__=merge_blocks(de_coeffs_blocks_R_, image_array_rgb.shape)
G__=merge_blocks(de_coeffs_blocks_G_, image_array_rgb.shape)
B__=merge_blocks(de_coeffs_blocks_B_, image_array_rgb.shape)

#объединение массивов цветов в изображение
```

```
    rgb_image_array__ = np.dstack((R__, G__, B__))
    rgb_image_array__=rgb_image_array__.astype(np.uint8)
    image_rgb__ = Image.fromarray(rgb_image_array__, 'RGB')

    #изображение ДКТ
    image_rgb__.show()
    #изображение ДКТ матрицей
    image_rgb_.show()

    print(f"Время выполнения dct_matrix: {int(dct_matrix_time)} секунд")
    print(f"Время выполнения dct: {int(dct_time)} секунд")

    print(f"ДКТ с помощью матрицы работает быстрее обычного ДКТ в: {int(dct_time/dct_matrix_time)} раз")
    print(f"----------------------------------------------------------------------------------------------")

#########################################################################################################
#########################################################################


if __name__ == "__main__":

    # image_array = np.random.randint(0, 256, (1600, 1600, 3), dtype=np.uint8)
    # # Создаем изображение из массива
    # image = Image.fromarray(image_array, 'RGB')
    # # Сохраняем изображение в файл
    # image.save('1600_random_image.jpg')
    #
    # # Определяем цвет (например, зеленый)
    #color = (0, 0, 150)  # RGB: (R, G, B)
     # Создаем изображение размером 1600x1600 с указанным цветом
    #image = Image.new('RGB', (1600, 1600), color)
     # Сохраняем изображение в файл
    #image.save('1600_solid_image.jpg')
    # dct_check("800.jpg")

    # compress_and_decode_image("1600.jpg", "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_full_code_txt/txt_image_1600",90)
    # compress_and_decode_image("1600.jpg", "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_full_code_txt/txt_image_1600",
    #                           80)
    # compress_and_decode_image("1600.jpg", "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_full_code_txt/txt_random_image_1600",60)
    # compress_and_decode_image("1600.jpg", "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_full_code_txt/txt_solid_image_1600",40)
    # compress_and_decode_image("1600.jpg", "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_full_code_txt/txt_image_1600",
    #                           20)
    # compress_and_decode_image("1600.jpg",
    #                           "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/0_full_code_txt/txt_random_image_1600", 10)
```

```
    # without_YCbCr_compress_and_decode_image("1600.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/0_without_YCbCr_code_txt/txt_1600")
    # without_YCbCr_compress_and_decode_image("1600_random_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_YCbCr_code_txt/txt_random_image_1600")
    # without_YCbCr_compress_and_decode_image("800_solid_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_YCbCr_code_txt/txt_solid_image_800")



    # without_Subsempling_compress_and_decode_image("1600.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/0_without_Subsempling_code_txt/txt_1600")
    # without_Subsempling_compress_and_decode_image("1600_random_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_Subsempling_code_txt/txt_random_image_1600")
    # without_Subsempling_compress_and_decode_image("800_solid_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_Subsempling_code_txt/txt_solid_image_800")


    # without_DCT_compress_and_decode_image("1600.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/0_without_DCT_code_txt/txt_1600")
    # without_DCT_compress_and_decode_image("1600_random_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_DCT_code_txt/txt_random_image_1600")
    # without_DCT_compress_and_decode_image("800_solid_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_DCT_code_txt/txt_solid_image_800")



    # without_quant_compress_and_decode_image("1600.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/0_without_quant_code_txt/txt_1600")
    # without_quant_compress_and_decode_image("1600_random_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_quant_code_txt/txt_random_image_1600")
    # without_quant_compress_and_decode_image("800_solid_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2
лаба/Отчет/.venv/0_without_quant_code_txt/txt_solid_image_800")


    # quantization_compress_and_decode_image("1600.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/1_quant_code_txt/txt_1600")
    # quantization_compress_and_decode_image("1600_random_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/1_quant_code_txt/txt_random_image_1600")
    # quantization_compress_and_decode_image("800_solid_image.jpg",
    #                                         "C:/labs/aisd/4 семестр/2 лаба/Отчет/.venv/1_quant_code_txt/txt_solid_image_800")


##########################################################################################
#########################################################################
```