**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра систем автоматизированного проектирования**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Реализация алгоритма JPEG»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3352 |  | Шушков В.А. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2025

# Введение

**Цель:** реализация алгоритма , а также его составных частей, таких как: AC/DC кодирование, квантование матриц и так далее.

**Задачи:**

1. Реализовать преобразование цветового пространства из в и обратно.
2. Выполнить субдискретизацию (downsampling) хроминансных компонент изображения.
3. Разделить изображение на блоки размером N×N, заполняя неполные блоки заданным значением.
4. Изучить и реализовать прямое и обратное двумерное дискретное косинусное преобразование для блоков .
5. Разработать механизм адаптивного изменения квантовочной матрицы в зависимости от заданного уровня качества сжатия.
6. Реализовать зигзагообразный обход коэффициентов в блоке .
7. Изучить и реализовать дифференциальное кодирование DC-компонент и переменное кодирование AC и DC коэффициентов.
8. Реализовать RLE кодирование AC коэффициентов.
9. Реализовать кодирование разностей DC и пар Run/Size для AC коэффициентов с использованием таблиц Хаффмана и упаковку закодированных данных в байтовую строку.
10. Собрать полный алгоритм кодирования и декодирования изображений в формате JPEG.
11. Построить график зависимости размера сжатого файла от коэффициента качества с шагом 2.

# 

# Теоретическая часть

## Цветовые пространства и

1. :

Цветовая модель – это аддитивная модель, применяемая для цифрового представления и воспроизведения цветов с использованием трёх базовых компонентов: красного (Red), зелёного (Green) и синего (Blue).

Модель можно представить как трёхмерное пространство, в котором каждая ось соответствует одному из цветовых каналов. Значения в каждом канале варьируются от 0 до 255, что обусловлено использованием 8-битного представления (1 байт) на канал. Точка (0, 0, 0) соответствует чёрному цвету, а (255, 255, 255) — белому.

В изображениях цвета формируются путём наложения этих трёх каналов. Интенсивность каждого из них определяет вклад соответствующего цвета: чем выше значение, тем ярче компонент. Например, значение 255 в синем канале означает его максимальную насыщенность, тогда как 0 означает полное его отсутствие



.

1. :

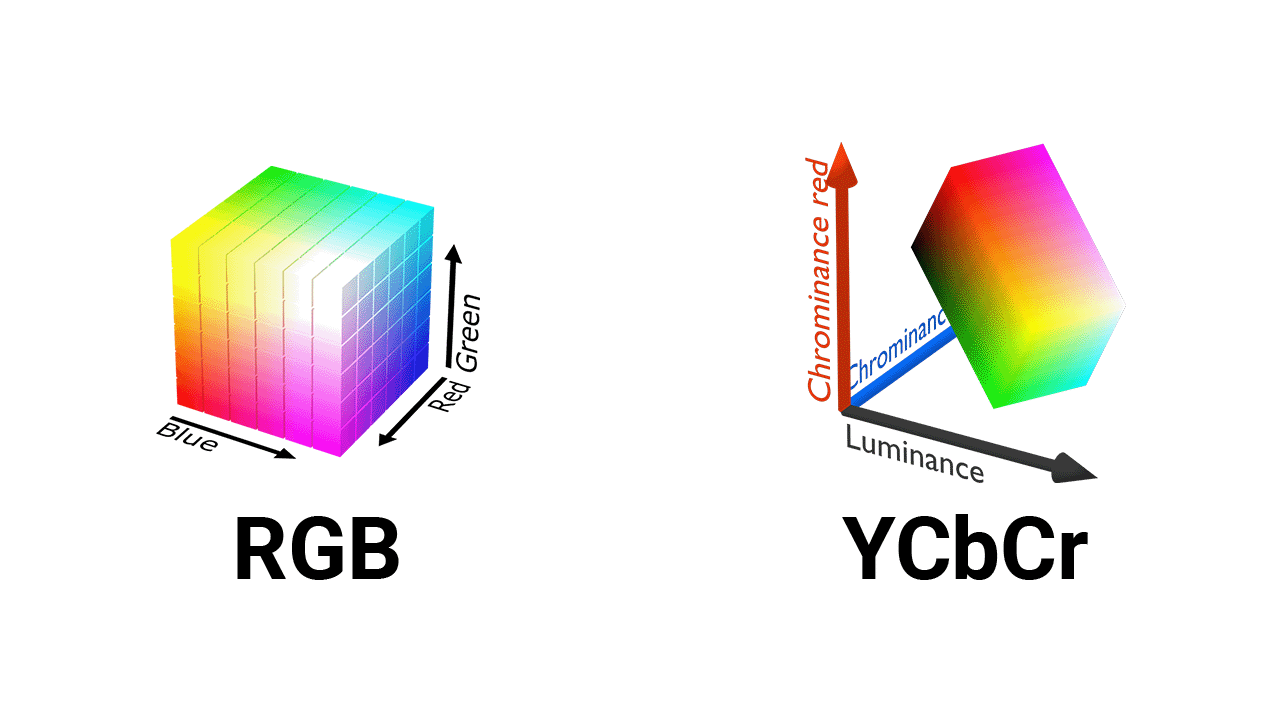
– это семейство цветовых пространств, широко применяемых для хранения и передачи цветных изображений в цифровом виде, включая видео и фотографии. Оно основано на разложении изображения на отдельные компоненты:

* **Y** — яркостная составляющая (luma),
* **Cb** — разность между синим и яркостью (blue-difference chroma),
* **Cr** — разность между красным и яркостью (red-difference chroma).

В отличие от RGB-модели, где цвет задаётся как комбинация трёх основных цветов, пространство отделяет яркость от цветовой информации, что делает его более удобным для сжатия и обработки.

Часто представляют в виде двумерной плоскости с осями Cb и Cr, где каждая точка отражает цветовой оттенок, а яркость задаётся отдельно.

Рисунок 1.2 – Простейшее представление



1. Переходы цветовых пространств:

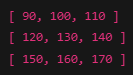
Для преобразования цветового изображения из пространства RGB в YCbCr применяются заранее определённые формулы. При этом значения каждого цветового компонента предварительно нормализуются, то есть приводятся к диапазону от 0 до 1 путём деления на 255. Нормализация необходима для стандартизации данных и корректного выполнения вычислений. Формулы преобразования приведены ниже.

Для перехода из в используются обратные формулы:

1. Downsampling и Upsampling:

Downsampling – это процесс уменьшения разрешения изображения путём снижения количества пикселей. Это достигается путём группировки соседних пикселей и замены каждой группы одним пикселем, значение которого рассчитывается как среднее арифметическое по всем пикселям в группе.

Например, при коэффициенте **N = 3** берутся блоки размером 3×3 пикселя, и каждый такой блок преобразуется в один пиксель с усреднённым значением



Сумма значений:  
90 + 100 + 110 + 120 + 130 + 140 + 150 + 160 + 170 = **1170**

Среднее значение:  
1170 / 9 = **130**

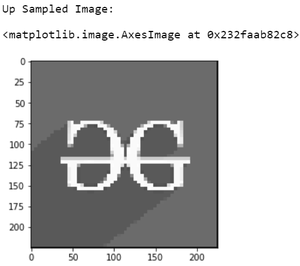
**Результат:**  
Блок 3×3 заменяется одним пикселем со значением **130**

С помощью Downsampling изображение теряет значительную часть своей информации, но общие очертания изображения остаются узнаваемыми. Поэтому downsampling применяют только к слоям Cb и Cr так как изменения качества на них наименее восприимчивы человеком.

Upsampling – это процесс увеличения размера изображения в раз. Данное действие происходит путем добавления новых пикселей на основе существующих. Таким образом, находится крайний пиксель и расширяется в квадрат на основе этого. Так, например, для 1 пиксель в исходном изображении дублируется в группу пикселей 2х2 с одним и тем же набором значений.

С помощью Upsampling изображение не восстанавливает исходное изображение, но увеличивается до его размеров.

Рисунок 1.4 – Upsampling преобразование

1. DCT-II 2D и iDCT-II 2D:

В алгоритме JPEG дискретное косинусное преобразование (ДКП) выполняет ключевую функцию перевода изображения из пространственного представления (где каждый пиксель описывается своим цветом и яркостью) в частотное. Это преобразование перераспределяет информационную энергию изображения, концентрируя её преимущественно в низкочастотных составляющих.

Результирующая матрица коэффициентов после ДКП имеет важные особенности:

* Низкочастотные компоненты (расположенные в верхнем левом углу матрицы) содержат основную информацию о плавных цветовых переходах и общей яркости блока.
* Высокочастотные компоненты (в правом нижнем углу) отражают резкие цветовые и яркостные переходы:
* Положительные значения коэффициентов указывают на резкие цветовые скачки
* Амплитуда коэффициентов прямо пропорциональна степени резкости перехода
* Чем выше значение коэффициента, тем более контрастный переход он представляет
* Плавные градиенты и мягкие цветовые изменения кодируются малыми по величине коэффициентами.

Такое распределение информации позволяет эффективно сжимать изображение - малозначимые высокочастотные компоненты могут быть отброшены или сильно квантованы с минимальным визуальным ущербом для восприятия изображения.

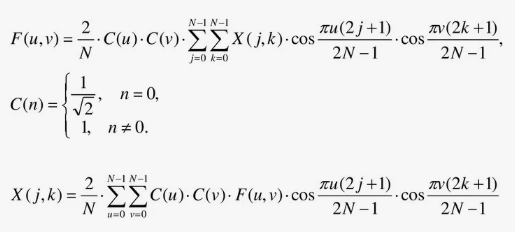
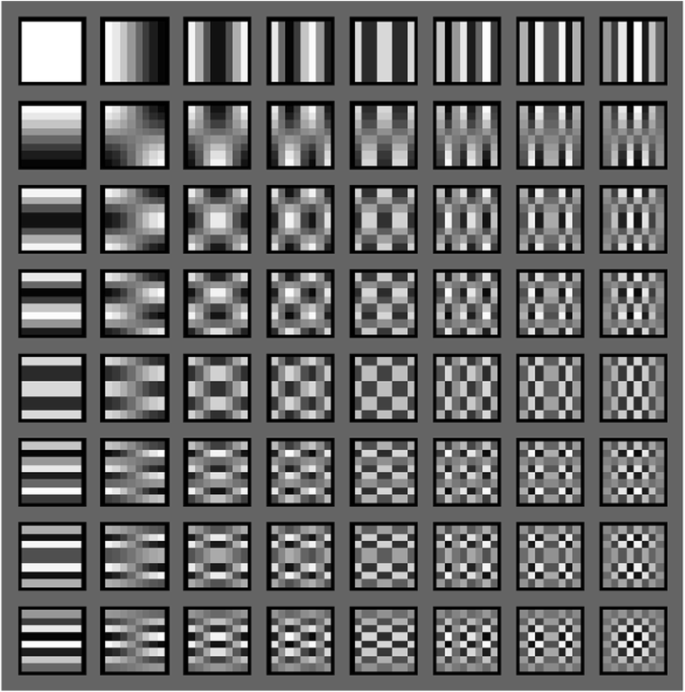


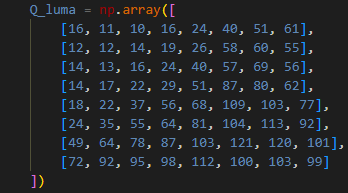
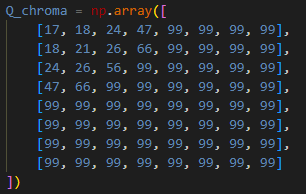
Рисунок 1.5 – ДКП



1. Матрица квантования:

Матрица квантования применяется для увеличения числа одинаковых значений в блоке. После применения ДКП наиболее значимая информация содержится в районе ячеек (0, 0), при этом, чем ближе к другому краю матрицы, тем менее ценная информация там хранится, таким образом, необходимо максимально сохранить значения около ячеек (0, 0), а остальными можно пожертвовать в пользу сжатия. В связи с этим матрицы квантования и формируются определенным образом. Суть матрицы в том, чтобы поэлементно делить значения из блока ДКП на значения из матрицы.

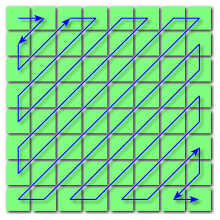
Рисунок 1.6 – Квантование преобразование

1. Зигзаг обход:

Для дальнейшего кодирования необходимо перевести значения матрицы в строку тем более, что из-за особенностей преобразования рядом будут находится равные или похожие значения. Из-за такой структуры было предложено проводить обход зигзагом, таким образом выходная строка будет содержать больше последовательностей одинаковых символов.

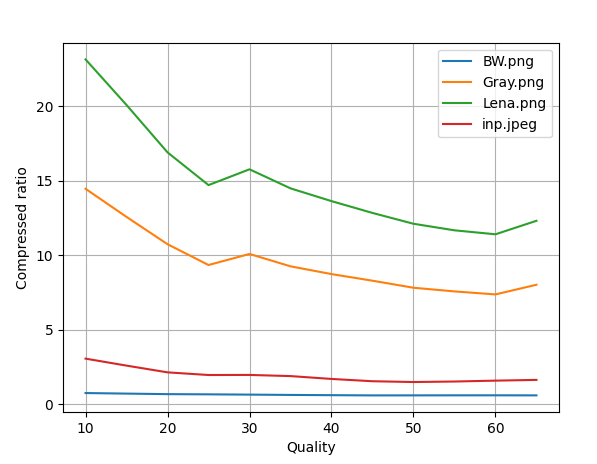
Рисунок 1.6 – Обход зигзагом



# Практическая часть.

Рассмотрев результаты сжатия, отображенные на графике, можно сказать, что лучший результат был получен на цветном изображении, при этом более маленькое изображение получилось сжать сильнее. Результаты на малом и большом изображении отличаются, скорее всего это связано с изначальным форматом хранения данных и наличие канала прозрачности в картинках формата png.

Рисунок 2.1 – Результаты сжатия



Далее будут приведены примеры сжатых изображений с разным уровнем качества.

Рисунок 2.2 – Примеры изображений цветных



Рисунок 2.3 – Примеры изображений цветных

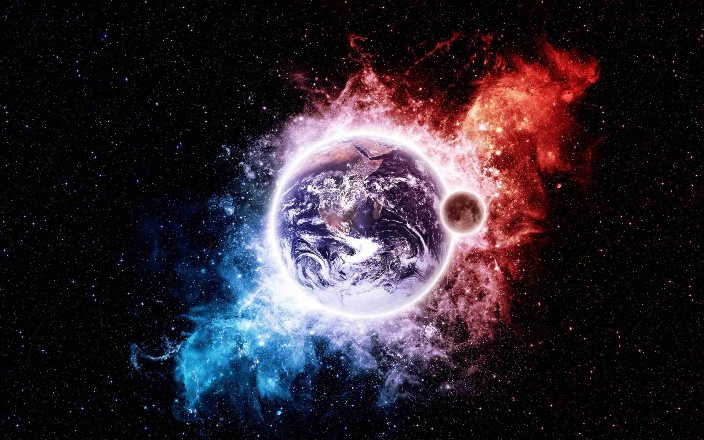
 

Рисунок 2.4 – Примеры ЧБ изображений



Рисунок 2.5 – Примеры изображений оттенков серого

Данные изображения были приведены в следующем порядке: слева оригинал, который передавался функции компрессии, справа изображение после обработки с качеством 20. На более маленьком изображении артефакты сжатия видны сильнее, чем на изображении высокого качества.

# Код программы

**Write.py**

import struct

import heapq

import ast

def huffman(data):

if not data:

return {}

# 1. Подсчёт частот символов

freq = {value:data.count(value) for value in set(data)}

# 2. Создание мин-кучи (частот, символ)

heap = [[f, [char, ""]] for char, f in freq.items()]

heapq.heapify(heap)

# 3. Построение кодов без дерева

while len(heap) > 1:

# Извлекаем два элемента с наименьшей частотой

left = heapq.heappop(heap)

right = heapq.heappop(heap)

# Добавляем '0' и '1' к их кодам

for pair in left[1:]:

pair[1] = '0' + pair[1]

for pair in right[1:]:

pair[1] = '1' + pair[1]

# Объединяем узлы и добавляем обратно в кучу

merged\_freq = left[0] + right[0]

merged\_items = left[1:] + right[1:]

heapq.heappush(heap, [merged\_freq] + merged\_items)

# 4. Извлекаем коды из последнего элемента кучи

codes = {}

if heap:

for char, code in heap[0][1:]:

codes[char] = code

cache = ""

#print(codes)

for char in data:

cache+=codes[char]

return cache, codes

def unhuffman(encoded\_data, huff\_codes):

reverse\_codes = {v: k for k, v in huff\_codes.items()}

current\_code = ""

decoded\_data = bytearray()

for bit in encoded\_data:

current\_code += bit

if current\_code in reverse\_codes:

decoded\_data.append(reverse\_codes[current\_code])

current\_code = ""

return decoded\_data

def repr\_to\_ndarray(s):

from numpy import array

return eval(s)

def write\_huffman\_encoded(combined\_data, filename):

# 3. Кодируем данные

encoded\_data, codes = huffman(combined\_data)

# 4. Записываем в файл

with open(filename, 'wb') as f:

# Записываем заголовок (размер данных и таблицу кодов)

header = {

'codes': codes,

'data\_length': len(encoded\_data),

'original\_length': len(combined\_data)

}

header\_bytes = str(header).encode('utf-8')

f.write(struct.pack('I', len(header\_bytes)))

f.write(header\_bytes)

# Записываем закодированные данные

padded\_data = encoded\_data + '0' \* (8 - len(encoded\_data) % 8)

for i in range(0, len(padded\_data), 8):

byte = padded\_data[i:i+8]

f.write(bytes([int(byte, 2)]))

def read\_huffman\_encoded(filename):

with open(filename, 'rb') as f:

# Читаем заголовок

header\_size = struct.unpack('I', f.read(4))[0]

header = eval(f.read(header\_size).decode('utf-8'))

# Читаем закодированные данные

encoded\_bits = ""

while True:

byte = f.read(1)

if not byte:

break

encoded\_bits += f"{ord(byte):08b}"

# Обрезаем лишние биты дополнения

encoded\_bits = encoded\_bits[:header['data\_length']]

# Декодируем Хаффмана

decoded\_data = unhuffman(encoded\_bits, header['codes'])

# Восстанавливаем структуру данных

temp = decoded\_data.decode('utf-8')

combined\_data = eval(temp)

return combined\_data

**functions.py**

import math

import sys

import numpy as np

from PIL import Image

import scipy

import os

import matplotlib.pyplot as plt

import heapq

from write import \*

from decimal import Decimal, getcontext

getcontext().prec = 20

def progress\_bar(iteration, total, length=40):

# Вычисляем процент завершения

percent = (iteration / total)

# Вычисляем количество символов для заполнения прогресс-бара

filled\_length = int(length \* percent)

# Создаем строку прогресс-бара

bar = '█' \* filled\_length + '-' \* (length - filled\_length)

# Выводим прогресс-бар

sys.stdout.write(f'\r|{bar}| {percent:.2%} Complete')

sys.stdout.flush()

def rgb\_to\_ycbcr(image):

"""Конвертация RGB в YCbCr цветовое пространство"""

# Коэффициенты преобразования

matrix = np.array([

[0.299, 0.587, 0.114],

[-0.1687, -0.3313, 0.5],

[0.5, -0.4187, -0.0813]

])

ycbcr = np.dot(image, matrix.T)

ycbcr[:,:,[1,2]] += 128

return np.clip(ycbcr, 0, 255)

def downsample(chroma):

"""Уменьшение разрешения цветоразностных компонентов (2x2)"""

h, w = chroma.shape

# Убедимся, что размеры четные

h = h // 2 \* 2

w = w // 2 \* 2

return (chroma[:h:2,:w:2] + chroma[1:h:2,:w:2] + chroma[:h:2,1:w:2] + chroma[1:h:2,1:w:2]) / 4

def dct2(block):

"""Дискретное косинусное преобразование (ДКП) для 2D блока"""

return scipy.fft.dct(scipy.fft.dct(block.T, norm='ortho').T, norm='ortho')

def idct2(block):

"""Обратное дискретное косинусное преобразование для 2D блока"""

return scipy.fft.idct(scipy.fft.idct(block.T, norm='ortho').T, norm='ortho')

def quantize(block, quantization\_table):

"""Квантование блока с использованием таблицы квантования"""

# Убедимся, что размеры блока и таблицы совпадают

if block.shape != quantization\_table.shape:

raise ValueError(f"Размеры блока {block.shape} и таблицы квантования {quantization\_table.shape} не совпадают")

return np.round(block / quantization\_table)

def dequantize(block, quantization\_table):

"""Обратное квантование"""

if block.shape != quantization\_table.shape:

raise ValueError(f"Размеры блока {block.shape} и таблицы квантования {quantization\_table.shape} не совпадают")

return block \* quantization\_table

def RLE(inp):

count = 0

let = inp[0]

out=""

for i in inp:

if let == i and count < 255:

count+=1

else:

#print(let)

out+=f"{chr(count)}{chr(int(let)+128)}"

let = i

count = 1

if count != 255:

out+=f"{chr(count)}{chr(int(let)+128)}"

return out

def unRLE(inp):

out=[]

for i in range(1, len(inp), 2):

out+=[ord(inp[i])-128]\*(ord(inp[i-1]))

return out

def zigzag(block):

row, col = 0, 0

line = []

while row<block.shape[0] and col<block.shape[1]:

line.append(block[row][col])

if (row+col)% 2 == 0:

if col == 0 and row != block.shape[0]-1:

row+=1

elif row == block.shape[0]-1:

col+=1

else:

col-=1

row+=1

else:

if col == block.shape[1]-1:

row+=1

elif row == 0 and col != block.shape[1]-1:

col+=1

else:

col+=1

row-=1

return line

def unzigzag(line, shape, block\_size = 8):

row, col = 0, 0

i = 0

block = np.zeros(shape=shape, dtype=int)

while row<block.shape[0] and col<block.shape[1]:

block[row][col] = line[i]

i+=1

if (row+col)% 2 == 0:

if col == 0 and row != block.shape[0]-1:

row+=1

elif row == block.shape[0]-1:

col+=1

else:

col-=1

row+=1

else:

if col == block.shape[1]-1:

row+=1

elif row == 0 and col != block.shape[1]-1:

col+=1

else:

col+=1

row-=1

return block

def process\_channel(channel, quant\_table, block\_size=8):

"""Обработка одного канала (Y, Cb или Cr)"""

height, width = channel.shape

# Дополнение изображения до кратного block\_size

pad\_h = block\_size - height % block\_size

pad\_w = block\_size - width % block\_size

padded = np.pad(channel, ((0, pad\_h), (0, pad\_w)), mode='edge')

compressed = np.zeros\_like(padded)

comp=""

for i in range(0, padded.shape[0], block\_size):

for j in range(0, padded.shape[1], block\_size):

block = padded[i:i+block\_size, j:j+block\_size] - 128

dct\_block = dct2d\_matrix(block)

quant\_block = quantize(dct\_block, quant\_table)

compressed[i:i+block\_size, j:j+block\_size] = quant\_block

comp += RLE(zigzag(quant\_block))

return comp, (height, width)

def reconstruct\_channel(compressed, quant\_table, original\_shape, block\_size=8):

"""Восстановление одного канала"""

pad\_h = block\_size - original\_shape[0] % block\_size

pad\_w = block\_size - original\_shape[1] % block\_size

line = unRLE(compressed)

reconstructed = np.zeros(shape = (original\_shape[0]+pad\_h, original\_shape[1]+pad\_w))

for i in range(0, reconstructed.shape[0], block\_size):

for j in range(0, reconstructed.shape[1], block\_size):

quant\_block = unzigzag(line[:64], (8,8))

line = line[64:]

dct\_block = dequantize(quant\_block, quant\_table)

idct\_block = idct2d\_matrix(dct\_block)

reconstructed[i:i+block\_size, j:j+block\_size] = idct\_block + 128

# Обрезка до исходного размера

return reconstructed[:original\_shape[0], :original\_shape[1]]

def jpeg\_compress(image\_path, output\_path, quality=50):

"""Основная функция сжатия JPEG"""

# 1. Загрузка изображения

img = Image.open(image\_path)

if img.mode == 'RGBA':

img = img.convert('RGB')

img\_array = np.array(img, dtype=np.float32)

# 2. Конвертация в YCbCr

ycbcr = rgb\_to\_ycbcr(img\_array)

y = ycbcr[:,:,0]

cb = ycbcr[:,:,1]

cr = ycbcr[:,:,2]

# 3. Уменьшение цветовых компонентов (chroma subsampling)

cb\_down = downsample(cb)

cr\_down = downsample(cr)

# 4. Таблица квантования для яркости

Q\_luma = 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]

])

# Масштабирование таблицы квантования

if quality < 50:

scale = 5000 / quality

else:

scale = 200 - 2 \* quality

Q\_luma = np.clip(np.floor((Q\_luma \* scale + 50) / 100), 1, 255)

# Таблица квантования для цветности (обычно более грубая)

Q\_chroma = np.array([

[17, 18, 24, 47, 99, 99, 99, 99],

[18, 21, 26, 66, 99, 99, 99, 99],

[24, 26, 56, 99, 99, 99, 99, 99],

[47, 66, 99, 99, 99, 99, 99, 99],

[99, 99, 99, 99, 99, 99, 99, 99],

[99, 99, 99, 99, 99, 99, 99, 99],

[99, 99, 99, 99, 99, 99, 99, 99],

[99, 99, 99, 99, 99, 99, 99, 99]

])

Q\_chroma = np.clip(np.floor((Q\_chroma \* scale + 50) / 100), 1, 255)

# 5. Обработка каналов

compressed\_y, y\_shape = process\_channel(y, Q\_luma)

compressed\_cb, cb\_shape = process\_channel(cb\_down, Q\_chroma)

compressed\_cr, cr\_shape = process\_channel(cr\_down, Q\_chroma)

combined = {

'y': compressed\_y,

'cb': compressed\_cb,

'cr': compressed\_cr,

'y\_shape': y\_shape,

'cb\_shape': cb\_shape,

'cr\_shape': cr\_shape,

'quant\_luma': repr(Q\_luma),

'quant\_chroma': repr(Q\_chroma),

'original\_shape': img\_array.shape,

}

data = str(combined).encode('utf-8')

write\_huffman\_encoded(data, output\_path)

return {

'compressed\_y': compressed\_y,

'compressed\_cb': compressed\_cb,

'compressed\_cr': compressed\_cr,

'quant\_luma': Q\_luma,

'quant\_chroma': Q\_chroma,

'original\_size': img\_array.size,

'compressed\_size': len(compressed\_y) + len(compressed\_cr) + len(compressed\_cb),

'original\_shape': img\_array.shape,

'y\_shape': y\_shape,

'cb\_shape': cb\_shape,

'cr\_shape': cr\_shape

}

def jpeg\_decompress(input\_path):

"""Декомпрессия JPEG"""

compressed\_data = read\_huffman\_encoded(input\_path)

compressed\_data['quant\_luma'] = repr\_to\_ndarray(compressed\_data['quant\_luma'])

compressed\_data['quant\_chroma'] =repr\_to\_ndarray(compressed\_data['quant\_chroma'])

# 1. Восстановление каналов

y = reconstruct\_channel(

compressed\_data['y'],

compressed\_data['quant\_luma'],

compressed\_data['y\_shape']

)

cb = reconstruct\_channel(

compressed\_data['cb'],

compressed\_data['quant\_chroma'],

compressed\_data['cb\_shape']

)

cr = reconstruct\_channel(

compressed\_data['cr'],

compressed\_data['quant\_chroma'],

compressed\_data['cr\_shape']

)

# 2. Восстановление цветовых компонентов (интерполяция)

cb\_upsampled = np.repeat(np.repeat(cb, 2, axis=0), 2, axis=1)

cr\_upsampled = np.repeat(np.repeat(cr, 2, axis=0), 2, axis=1)

# Обрезка до исходного размера

original\_height, original\_width = compressed\_data['original\_shape'][:2]

cb\_upsampled = cb\_upsampled[:original\_height, :original\_width]

cr\_upsampled = cr\_upsampled[:original\_height, :original\_width]

# 3. Обратное преобразование в RGB

ycbcr = np.stack([y, cb\_upsampled, cr\_upsampled], axis=-1)

ycbcr[:,:,[1,2]] -= 128

matrix = np.array([

[1, 0, 1.402],

[1, -0.34414, -0.71414],

[1, 1.772, 0]

])

rgb = np.dot(ycbcr, matrix.T)

return np.clip(rgb, 0, 255).astype(np.uint8)

def create\_cos\_table(N):

"""Создает таблицу косинусов для N-точечного ДКП"""

cos\_table = [[0.0] \* N for \_ in range(N)]

for k in range(N):

for n in range(N):

cos\_table[k][n] = math.cos(float(Decimal(math.pi) \* Decimal(k) \* Decimal(2\*n + 1) / Decimal(2\*N)))

return cos\_table

# Глобальная таблица для блоков 8x8 (стандарт JPEG)

COS\_TABLE\_8x8 = create\_cos\_table(8)

def dct1d\_manual(signal, cos\_table):

"""1D DCT с предвычисленной таблицей косинусов"""

N = len(signal)

result = [0.0] \* N

for k in range(N):

sum\_val = 0.0

for n in range(N):

sum\_val += signal[n] \* cos\_table[k][n]

ck = math.sqrt(1/N) if k == 0 else math.sqrt(2/N)

result[k] = ck \* sum\_val

return result

def dct2d\_manual(matrix, cos\_table):

"""2D DCT с оптимизацией через таблицу косинусов"""

rows = len(matrix)

cols = len(matrix[0]) if rows > 0 else 0

# Применяем 1D DCT к каждой строке

temp = []

for i in range(rows):

temp.append(dct1d\_manual(matrix[i], cos\_table))

# Применяем 1D DCT к каждому столбцу

result = []

for j in range(cols):

column = [temp[i][j] for i in range(rows)]

transformed\_col = dct1d\_manual(column, cos\_table)

if not result:

result = [[] for \_ in range(rows)]

for i in range(rows):

result[i].append(transformed\_col[i])

return np.array(result)

def idct1d\_manual(coefficients, cos\_table):

"""1D обратное ДКП с таблицей косинусов"""

N = len(coefficients)

result = [0.0] \* N

for n in range(N):

sum\_val = 0.0

for k in range(N):

ck = math.sqrt(1/N) if k == 0 else math.sqrt(2/N)

sum\_val += ck \* coefficients[k] \* cos\_table[k][n]

result[n] = sum\_val

return result

def idct2d\_manual(matrix, cos\_table):

"""2D обратное ДКП с оптимизацией"""

rows = len(matrix)

cols = len(matrix[0]) if rows > 0 else 0

# Обработка столбцов

temp = []

for j in range(cols):

column = [matrix[i][j] for i in range(rows)]

transformed\_col = idct1d\_manual(column, cos\_table)

if not temp:

temp = [[] for \_ in range(rows)]

for i in range(rows):

temp[i].append(transformed\_col[i])

# Обработка строк

return np.array([idct1d\_manual(row, cos\_table) for row in temp])

def matrix\_mult(A, B):

"""Умножение матриц (A × B)"""

return [

[

sum(A[i][k] \* B[k][j] for k in range(len(B)))

for j in range(len(B[0]))

]

for i in range(len(A))

]

def dct2d\_matrix(block):

"""2D DCT через матричное умножение: C × block × C'"""

# Умножаем на матрицу C слева (C × block)

temp = matrix\_mult(C, block)

# Умножаем на C' справа (result × C\_transpose)

return np.array(matrix\_mult(temp, C\_transpose))

def idct2d\_matrix(coeffs):

"""2D IDCT: C' × coeffs × C"""

temp = matrix\_mult(C\_transpose, coeffs)

return np.array(matrix\_mult(temp, C))

def compute\_dct\_matrix(N):

"""Создает матрицу ДКП для N-точечного преобразования"""

C = [[0.0] \* N for \_ in range(N)]

for k in range(N):

for n in range(N):

if k == 0:

C[k][n] = math.sqrt(1/N)

else:

C[k][n] = math.sqrt(2/N) \* math.cos(math.pi \* k \* (2\*n + 1) / (2\*N))

return C

C = compute\_dct\_matrix(8)

C\_transpose = np.array([list(row) for row in zip(\*C)])

**Main.py**

from function import \*

import pandas as pd

# Пример использования

if \_\_name\_\_ == "\_\_main\_\_":

files = [

#"BW.png",

#"Gray.png",

#"Lena.png",

"inp.jpeg"

]

i = 0

columns = ["files"]+[quality for quality in range(10, 70, 5)]

results = []

for input\_path in files:

output\_path = "compressed.bin"

# Сжатие

temp = []

for quality in columns[1:]:

compressed = jpeg\_compress(input\_path, output\_path, quality)

i+=1

progress\_bar(i, len(files)\*len(columns)\*2)

# Декомпрессия

reconstructed\_img = jpeg\_decompress(output\_path)

# Сохранение и отображение результатов

original = Image.open(input\_path)

result = Image.fromarray(reconstructed\_img)

result.save("photo/"+input\_path+str(quality)+".jpg")

# Расчет степени сжатия

compression\_ratio = os.path.getsize(input\_path) / os.path.getsize(output\_path)

temp.append(compression\_ratio)

i+=1

progress\_bar(i, len(files)\*len(columns)\*2)

results.append([input\_path]+temp)

print(pd.DataFrame(results, columns=columns))

# Заключение

Результаты сжатия представлены в таблице.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Files | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 |
| BW.png | 0.75 | 0.71 | 0.68 | 0.66 | 0.64 | 0.62 | 0.61 | 0.59 | 0.59 | 0.60 | 0.60 | 0.59 |
| Gray.png | 14.44 | 12.56 | 10.73 | 9.33 | 10.08 | 9.25 | 8.73 | 8.28 | 7.81 | 7.56 | 7.36 | 8.01 |
| Lena.png | 23.12 | 20.06 | 16.89 | 14.69 | 15.75 | 14.47 | 13.62 | 12.83 | 12.10 | 11.66 | 11.39 | 12.30 |
| inp.jpg | 3.06 | 2.59 | 2.14 | 1.96 | 1.96 | 1.89 | 1.69 | 1.54 | 1.49 | 1.52 | 1.58 | 1.63 |

Таким образом в рамках лабораторной работы был изучен алгоритм работы компрессора jpeg. Были воссозданы функции для промежуточных этапов работы с изображениями и реализован JPEG-инспирированный компрессор, который сжимает изображение с приемлемой потерей качества.

# Ссылка на репозиторий GitHub

https://github.com/K4taLizator/aisdlab2.git