

Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А. Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 4
по дисциплине «Основы криптографии и стеганографии»
Стеганографическое встраивание информации в пространственную область
цифровых изображений

Студентка гр. БИБ235
_____ А.Н. Ворощук
«11» мая 2024 г.

Руководитель
Заведующий кафедрой информационной
безопасности киберфизических систем
канд. техн. наук, доцент
_____ О. О. Евсютин
«__» _____ 2024 г.

Москва 2024

Оглавление

1 Задание на практическую работу	3
2 Краткая теоретическая часть	4
2.1 Метод LSB	4
2.2 Метод PM1	4
2.3 Метод QIM	5
2.4 Метод PVD	5
2.5 Метод NMI	6
2.6 Оценка эффективности встраивания	7
3 Результаты работы программы	9
3.1 Функции для реализации метода NMI.....	9
3.2 Функции для расчёта показателей качества.....	12
3.3 Итоговая программа	13
4 Результаты вычислительных экспериментов.....	15
4.1 Встраивание максимально возможного для данного контейнера объема информации.....	15
4.2 Встраивание разного количества информации.....	17
4.3 Извлечение встроенной информации из стегоизображений	18
4.4 Сравнение гистограмм изображений до и после встраивания.....	19
5 Выводы о проделанной работе.....	21
6 Используемая литература	22

1 Задание на практическую работу

1) Написать программную реализацию одного из следующих стеганографических методов для цифровых изображений по выбору студента: QIM, PVD, NMI;

2) Провести вычислительные эксперименты с полученной программной реализацией и сделать выводы об эффективности рассмотренного метода встраивания;

3) Подготовить отчет о выполнении работы.

Программа должна обладать следующей функциональностью:

1) при встраивании:

принимать на вход цветное (RGB) изображение-контейнер;

принимать на вход сообщение для встраивания;

рассчитывать показатели качества встраивания;

2) при извлечении:

принимать на вход цветное стегоизображение;

3) осуществлять встраивание или извлечение информации по выбору пользователя.

Вычислительные эксперименты с полученной программной реализацией должны включать следующее:

1) встраивание максимально возможного для данного контейнера объема информации с оценкой незаметности встраивания;

2) встраивание разного количества информации и сравнение незаметности встраивания;

3) извлечение встроенной информации из стегоизображений в условиях отсутствия постобработки и при наличии различных операций обработки изображений, оценка робастности;

4) сравнение гистограмм изображений до и после встраивания.

Отчет должен содержать следующие составные части:

1) раздел с заданием;

2) раздел с краткой теоретической частью;

3) раздел с результатами работы программы;

4) раздел с результатами вычислительных экспериментов;

5) раздел с выводами о проделанной работе.

2 Краткая теоретическая часть

Стеганография – это наука о скрытой передаче и хранении информации таким образом, чтобы сам факт наличия этой информации был тайной для третьих лиц. Цифровая стеганография работает с цифровой информацией. В качестве контейнеров для секретных сообщений могут выступать самые разные цифровые объекты: мультимедиа-данные, исполняемые файлы, интернет-трафик, данные с датчиков «интернета вещей» и т.д.

Одним из наиболее распространенных типов контейнеров для стеганографического встраивания являются цифровые изображения. В первую очередь это связано с их высокой избыточностью: наиболее распространенная цветовая модель RGB содержит 2^{24} различных цветов, поэтому небольшое изменение оттенка отдельных пикселей остается незаметным для глаза человека. Кроме того, цифровые изображения обладают так называемой пространственной избыточностью, которая состоит в том, что соседние пиксели изображения в большинстве случаев похожи друг на друга. Это свойство также широко используется в методах обработки цифровых изображений.

Существует большое количество различных методов встраивания дополнительной информации в цифровые изображения, отличающихся разными особенностями. Все существующие методы делятся на два больших класса: методы пространственного встраивания и методы частотного встраивания. Методы пространственного встраивания напрямую изменяют значения пикселей изображения, в то время как методы частотного встраивания вносят изменения в значения частотных коэффициентов, полученных после применения некоторого частотного преобразования к матрице пикселей.

2.1 Метод LSB

Одним из самых простых методов пространственного встраивания является метод замены наименее значимых битов (Least Significant Bits, LSB) [1]. Для встраивания секретного сообщения необходимо преобразовать в двоичную последовательность. Метод LSB предполагает замену одного или нескольких младших битов пикселей битами секретного сообщения при встраивании информации. Чтобы извлечь встроенную информацию, необходимо последовательно обойти пиксели изображения и сформировать последовательность из нужного количества младших битов каждого пикселя.

2.2 Метод PM1

Метод «плюс-минус один» (Plus-Minus One, PM1) является модификацией метода LSB. Данный метод меняет младший бит пикселя на бит сообщения посредством

случайного уменьшения или увеличения значения пикселя на единицу в следующих случаях:

- если значение пикселя четное, и бит секретного сообщения равен единице;
- если значение пикселя нечетное, и бит секретного сообщения равен нулю.

Соответствующая формула встраивания имеет вид:

$$P'_i = \begin{cases} P_i + (-1)^r, & \text{если } P_i \bmod 2 \neq m_i; \\ P_i, & \text{иначе,} \end{cases}$$

где P_i – значение пикселя изображения до встраивания, P'_i – значение пикселя после встраивания, r – случайное число из множества $\{0, 1\}$.

Чтобы извлечь встроенную информацию, необходимо, как и в методе LSB, последовательно обойти пиксели изображения и сформировать последовательность из нужного количества младших битов каждого пикселя.

2.3 Метод QIM

Метод модуляции индекса квантования (Quantization Index Modulation, QIM) [3] состоит в изменении значений пикселей изображения в зависимости от значений встраиваемых битов сообщения. Данная операция называется модуляцией. Встраивание бита m_i осуществляется в пиксель контейнера P_i по формуле

$$P'_i = q \cdot \left\lfloor \frac{P_i}{q} \right\rfloor + \frac{q}{2} \cdot m_i,$$

где q – шаг квантования (четное число), $\lfloor \dots \rfloor$ – взятие целой части от деления.

Для извлечения моделируется ситуация встраивания в пиксель P'_i нулевого и единичного битов, после чего результирующий бит определяется на основании того, к какому из двух значений ближе имеющееся значение P'_i . Формула извлечения одного бита выглядит следующим образом:

$$m_i = \arg \min_{e \in \{0,1\}} |P'_i - [P'_i]_e|, [P'_i]_e = q \cdot \left\lfloor \frac{P'_i}{q} \right\rfloor + \frac{q}{2} \cdot e.$$

2.4 Метод PVD

В методе разности значений пикселей (Pixel Value Difference, PVD) [4] изображение разбивается на непересекающиеся пары пикселей (P_i, P_{i+1}) и для каждой пары считается разность их значений $d_k = P_i - P_{i+1}$, $k = \frac{i+1}{2}$. Абсолютное значение разности пары пикселей $|d_k|$ попадает в один из шести отрезков вида $[l_k, u_k]$. Эти отрезки заданы заранее

и обычно определяются следующим образом: [0, 7], [8, 15], [16, 31], [32, 63], [64, 127], [128, 255].

Количество битов сообщения, которое может быть встроено в пару пикселей (P_i, P_{i+1}) , определяется по формуле

$$n_k = \log_2(u_k - l_k + 1).$$

Далее фрагмент сообщения длиной n_k битов представляется в виде целого числа m_k и вычисляется новое значение разности d_k^* для данной пары пикселей по формуле

$$d_k^* = \begin{cases} l_k + m_k, & \text{если } d_k \geq 0, \\ -(l_k + m_k), & \text{если } d_k < 0. \end{cases}$$

Встраивание фрагмента сообщения m_k в пару пикселей осуществляется по формуле:

$$(P'_i, P'_{i+1}) = \begin{cases} \left(P_i + \left\lfloor \frac{(d_k^* - d_k)}{2} \right\rfloor, P_{i+1} - \left\lfloor \frac{(d_k^* - d_k)}{2} \right\rfloor \right), & \text{если } d_k - \text{нечетное,} \\ \left(P_i + \left\lfloor \frac{(d_k^* - d_k)}{2} \right\rfloor, P_{i+1} - \left\lfloor \frac{(d_k^* - d_k)}{2} \right\rfloor \right), & \text{если } d_k - \text{четное.} \end{cases}$$

Извлечение сообщения выполняется по формуле

$$m_k = |P'_i - P'_{i+1}| - l_k.$$

2.5 Метод NMI

Метод встраивания на основе интерполяции по среднему значению соседей (Neighbor Mean Interpolation, NMI) устроен следующим образом. Исходное изображение C разрешением $m \times n$ преобразуется в изображение P разрешением $2m \times 2n$ по схеме, представленной на рисунке 1. Именно изображение P выступает в качестве контейнера для встраивания сообщения.

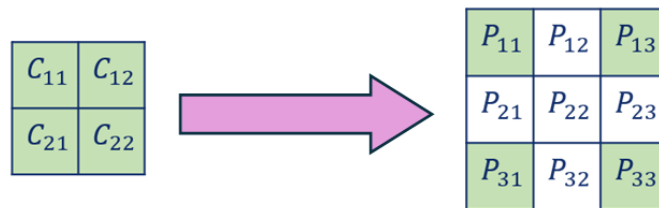


Рисунок 1 – Интерполяция изображения на этапе встраивания

Значения пикселей изображения P рассчитываются по следующим формулам:

$$P_{11} = C_{11};$$

$$P_{13} = C_{12};$$

$$P_{31} = C_{21};$$

$$P_{33} = C_{22};$$

$$P_{12} = \left\lfloor \frac{P_{11} + P_{13}}{2} \right\rfloor;$$

$$P_{21} = \left\lfloor \frac{P_{11} + P_{31}}{2} \right\rfloor;$$

$$P_{22} = \left\lfloor \frac{P_{11} + P_{12} + P_{21}}{3} \right\rfloor.$$

После этого для каждого из непересекающихся блоков пикселей размером 2×2 интерполированного изображения P вычисляется три разностных значения по следующей формуле:

$$d_{ij} = P_{ij} - P_d,$$

где P_d – левый верхний пиксель блока.

Фрагмент сообщения длиной $n_k = \lfloor \log_2 |d_{ij}| \rfloor$ битов представляется в виде целого числа m_k и встраивается в пиксель P_{ij} по формуле

$$P'_{ij} = P_{ij} + m_k.$$

Таким образом, в каждый блок пикселей размером 2×2 может быть встроено три фрагмента сообщения.

Чтобы извлечь сообщение из стегоконтейнера P' , сначала необходимо восстановить изображение C и посредством интерполяции сформировать изображение P , как это показано на рисунке 2.

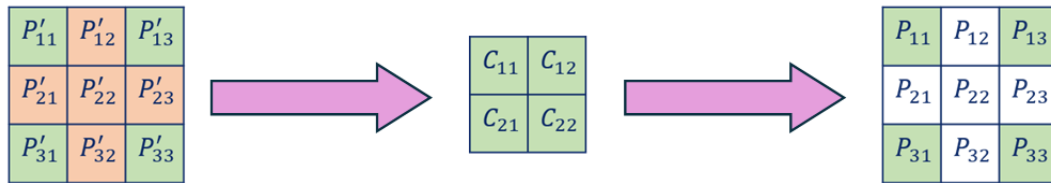


Рисунок 2 – Интерполяция изображения на этапе извлечения

После этого стегоизображение P' разбивается на блоки размером 2×2 пикселя и из каждого блока извлекается три фрагмента встроеного сообщения по формуле

$$m_k = P'_{ij} - P_{ij}.$$

2.6 Оценка эффективности встраивания

Для оценки качества стеганографического встраивания информации в изображения используются различные характеристики. Наиболее распространенными из них являются незаметность и емкость встраивания. Незаметность отражает различие между изображениями до и после встраивания и обычно оценивается метриками PSNR, MSE, RMSE, SSIM:

$$\text{PSNR (dB)} = 10 \times \log_{10} \left(\frac{255^2}{\text{MSE}} \right),$$

$$\text{MSE} = \frac{1}{M \times N} \sum_{i=1}^M \sum_{j=1}^N (P_{ij} - P'_{ij})^2,$$

$$\text{RMSE} = \sqrt{\text{MSE}},$$

где P_{ij} – значение пикселя изображения-контейнера, P'_{ij} – значение пикселя изображения с вложением;

$$\text{SSIM} = \frac{(2\mu_P\mu_S + K_1) \times (2\sigma_{PS} + K_2)}{(\mu_P^2 + \mu_S^2 + K_1) \times (\sigma_P^2 + \sigma_S^2 + K_2)},$$

где μ_P – среднее значение пикселей изображения-контейнера, μ_S – среднее значение пикселей изображения с вложением, σ_P^2 – дисперсия пикселей изображения-контейнера, σ_S^2 – дисперсия пикселей изображения с вложением, σ_{PS} – ковариация между пикселями обоих изображений, K_1 и K_2 – константы.

Емкость показывает, сколько битов встроенной информации приходится на каждый пиксель изображения, и выражается следующей формулой:

$$\text{EC (bpp)} = \frac{B}{M \times N},$$

где $M \times N$ – размеры изображения, B – это общее число битов, встроенных в изображение.

Для оценки устойчивости встраивания к деструктивным воздействиям, таким как изменение яркости, JPEG-сжатие и т.д. обычно применяются метрики BER и NCC:

$$\text{BER} = \frac{B_e}{B},$$

где B_e – количество ошибок, возникших при извлечении (измененных битов);

$$\text{NCC} = \frac{\sum_{x=1}^M \sum_{y=1}^N (W(x, y) \times W_{ext}(x, y))}{\sqrt{\sum_{x=1}^M \sum_{y=1}^N (W^2(x, y))} \sqrt{\sum_{x=1}^M \sum_{y=1}^N (W_{ext}^2(x, y))}},$$

где $W(x, y)$ – исходное сообщение, $W_{ext}(x, y)$ – извлеченное сообщение.

Метрика NCC обычно применяется, когда встроенное сообщение является изображением.

Важным показателем качества стеганографического встраивания является устойчивость перед стегоанализом. Существует множество схем стегоанализа для пространственной области изображений. Простейшие из них анализируют сходство гистограмм изображений до и после встраивания.

3 Результаты работы программы

3.1 Функции для реализации метода NMI

Для программной реализации был выбран метод встраивания на основе интерполяции по среднему значению соседей (Neighbor Mean Interpolation, NMI). Для перевода текста в двоичную строку были написаны функции, изображенные на рисунке 3.

```
def str_to_bitstring(s: str) -> str:
    """
    Исходная строка преобразуется в строку бит

    Args:
        s (str): Исходная текстовая строка

    Returns:
        str: Строка из нулей и единиц
    """

    bitstring = ''.join(format(ord(c), '08b') for c in s)
    return bitstring

def bitstring_to_str(bitstring: str) -> str:
    """
    Строка бит преобразуется в текстовую строку

    Args:
        s (str): Строка из нулей и единиц

    Returns:
        str: Текстовая строка
    """

    byte_strings = [bitstring[i:i + 8] for i in range(0, len(bitstring), 8)]
    string = ''.join(chr(int(bs, 2)) for bs in byte_strings)
    return string
```

Рисунок 3 – Функции для перевода строки в нужный формат

Работа метода встраивания NMI основана на интерполяции изображений. Функция `interpolation`, увеличивающая исходное изображение в 4 раза с помощью интерполяции изображена на рисунке 4.

```

def interpolation(image: Image) -> Image:
    """
    Исходное изображение m x n преобразуется в изображение 2m x 2n с помощью интерполяции

    Args:
        image (Image): Оригинальное изображение

    Returns:
        Image: Увеличенное изображение
    """
    width, height = image.size
    C = np.array(image)
    P = np.zeros((height*2, width*2, 3))
    #Переносу в большое изображение старые пиксели
    for i in range (height):
        for j in range (width):
            for k in range (3):
                P[2*i][2*j][k] = C[i][j][k]
    #Генерирую новые пиксели
    for i in range(2*height - 1):
        for j in range(2*width - 1):
            if (i%2 == 0 and j%2 == 1):
                for k in range(3):
                    P[i][j][k] = int((P[i][j-1][k] + P[i][j+1][k])/2)
            elif(i%2 == 1 and j%2 == 0):
                for k in range(3):
                    P[i][j][k] = int((P[i-1][j][k] + P[i+1][j][k])/2)
            elif(i%2 == 1 and j%2 == 1):
                for k in range(3):
                    P[i][j][k] = int((P[i-1][j-1][k] + P[i-1][j+1][k] + P[i+1][j-1][k])/3)
    #Генерирую новые пиксели для краевых участков
    for i in range(2*height - 1):
        for k in range(3):
            P[i][-1][k] = int((P[i][-2][k] + P[i][-3][k])/2)
    for j in range(2*width - 1):
        for k in range(3):
            P[-1][j][k] = int((P[-2][j][k] + P[-3][j][k])/2)
    for k in range(3):
        P[-1][-1][k] = int((P[-2][-1][k] + P[-1][-2][k] + P[-2][-2][k])/3)

    new_image = Image.fromarray(P.astype('uint8'), 'RGB')
    return(new_image)

```

Рисунок 4 – Функция, увеличивающая изображение с помощью интерполяции

Функция compression, уменьшающая интерполированное изображение до исходного состояния изображена на рисунке 5.

```

def compression(image: Image) -> Image:
    """
    Исходное изображение 2m x 2n преобразуется в изображение m x n

    Args:
        image (Image): Увеличенное изображение

    Returns:
        Image: Сжатое изображение
    """
    width, height = image.size
    P = np.array(image)
    C = np.zeros((height//2, width//2, 3))
    for i in range (height//2):
        for j in range (width//2):
            for k in range (3):
                C[i][j][k] = P[2*i][2*j][k]
    new_image = Image.fromarray(C.astype('uint8'), 'RGB')
    return new_image

```

Рисунок 5 – Функция, сжимающая изображение

Основная функция encode, которая встраивает текст в выбранное изображение показана на рисунке 6.

```
def encode(image: Image, message: str) -> Image:
    """Встраивание информации в изображение, метод Neighbor Mean Interpolation

    Args:
        image (Image): Изображение
        message (_type_): Сообщение, которое необходимо встроить
    Returns:
        Image: Стегаизображение
    """

    bitstring = str_to_bitstring(message)
    #буду работать в красном цвете, k = 0
    p_image = interpolation(image)
    width, height = image.size
    P = np.array(p_image)

    #проход по блокам 2 x 2
    break_flag = False
    for i in range (height):
        for j in range (width):

            Pd = int(P[2*i][2*j][0]) #основной пиксель

            for x, y in [(2*i, 2*j + 1), (2*i + 1, 2*j), (2*i + 1, 2*j + 1)]:
                d = abs(int(P[x][y][0]) - Pd)
                if (d == 0 or d ==1):
                    continue
                n = int(math.log2(d))

                if (len(bitstring) == 0):
                    break_flag = True
                    break
                elif (len(bitstring) < n):
                    m = int(bitstring, 2)
                    P[x][y][0] += m
                    bitstring = ""
                    break
                else:
                    m = int(bitstring[-n:], 2)
                    P[x][y][0] += m
                    bitstring = bitstring[0 : -n]

            if break_flag:
                break

        if break_flag:
            break

    s_image = Image.fromarray(P.astype('uint8'), 'RGB') #стеганоизображение
    return(s_image)
```

Рисунок 6 – Функция, встраивающая текст в изображение

Функция decode, возвращающая встроенный в стеганоизображение текст изображена на рисунке 7.

```

def decode(s_image:Image) -> str:
    image = compression(s_image)#исходное изображение
    width, height = image.size
    p_image = interpolation(image)#увеличенное изображение
    P = np.array(p_image)
    S = np.array(s_image)

    message = ""
    bitstring = ""
    #проход по блокам 2 x 2
    for i in range (height):
        for j in range (width):
            Pd = int(P[2*i][2*j][0]) #основной пиксель
            for x, y in [(2*i, 2*j + 1), (2*i + 1, 2*j), (2*i + 1, 2*j + 1)]:
                d = abs(int(P[x][y][0]) - Pd)
                m = S[x][y][0] - P[x][y][0]

                if (d == 0 or d == 1):
                    n = 0
                else:
                    n = int(math.log2(d))
                    bitstring = bin(m)[2:].zfill(n) + bitstring
    #make_multiple_of_8(bitstring)
    while len(bitstring) % 8 != 0:
        bitstring = bitstring[1:]
    message = bitstring_to_str(bitstring)
    return message

```

Рисунок 7 – Функция, извлекающая встроенную информацию из стегоизображения

3.2 Функции для расчёта показателей качества

На рисунке 8 изображены функции, рассчитывающие метрики MSE, RMSE, PSNR.

```

def MSE(image1, image2):
    c = 0
    width, height = image1.size
    P = np.array(image1).astype(float)
    S = np.array(image2).astype(float)
    for i in range (height):
        for j in range (width):
            c += (P[i][j][0] - S[i][j][0]) ** 2
    mse = c / (width * height)
    return mse

def RMSE(image1, image2):
    mse = MSE(image1, image2)
    rmse = math.sqrt(mse)
    return rmse

def PSNR(image1, image2):
    mse = MSE(image1, image2)
    psnr = 10*math.log10(255**2/mse)
    return psnr

```

Рисунок 8 – Расчёт метрик MSE, RMSE, PSNR

На рисунке 9 изображена функция для расчёта метрики SSIM.

```

def SSIM(image1, image2):
    K1 = (0.01 * 255)**2
    K2 = (0.03 * 255)**2
    width, height = image1.size
    P = np.array(image1).astype(float)
    S = np.array(image2).astype(float)

    #Вычисляю средние значения
    p_sum = np.sum(P)
    s_sum = np.sum(S)
    p_mean = p_sum / (width * height)
    s_mean = s_sum / (width * height)

    #Вычисляю дисперсии и ковариацию
    p_deviation_sum = np.sum((P - p_mean)**2)
    s_deviation_sum = np.sum((S - s_mean)**2)
    ps_deviation_sum = np.sum((P - p_mean)*(S - s_mean))
    p_variance = p_deviation_sum / (width * height)
    s_variance = s_deviation_sum / (width * height)
    ps_covariance = ps_deviation_sum / (width * height)

    #Вычисляю метрику
    numerator = (2*p_mean*s_mean + K1)*(2*ps_covariance + K2)
    denominator = (p_mean**2 + s_mean**2 + K1)*(p_variance + s_variance + K2)
    ssim = numerator/denominator
    return ssim

```

Рисунок 9 – Расчёт метрики SSIM

3.3 Итоговая программа

На рисунке 10 изображена итоговая программа.

```

from PIL import Image
import numpy as np
import math
from nmi import*
from analysis import*

choice = input("Для встраивания информации выберите E, для извлечения D:\n")
image_number = int(input("Выберите номер изображения (1-3): \n"))
if choice.upper() == 'E':
    image = Image.open(f"{image_number}.bmp")
    message = input("Введите сообщение для встраивания: \n")
    image1 = interpolation(image)
    image2 = encode(image, message)
    image2.save(f"{image_number}_new.bmp")
    print(f"Ваше стегоизображение {image_number}_new.bmp сохранено.\n")
    mse = MSE(image1, image2)
    rmse = RMSE(image1, image2)
    psnr = PSNR(image1, image2)
    ssim = SSIM(image1, image2)
    print(f"Ниже представлены полученные метрики:\n")
    print("MSE = ", mse, "\n")
    print("RMSE = ", rmse, "\n")
    print("PSNR = ", psnr, "\n")
    print("SSIM = ", ssim, "\n")
elif choice.upper() == 'D':
    new_image = Image.open(f"{image_number}_new.bmp")
    message = decode(new_image)
    print(f"Из стегоизображения {image_number}_new.bmp был получено сообщение: \n", message)

```

Рисунок 10 – Итоговая программа, объединение реализованных функций

Рассмотрим работу программы. При запуске пользователю предлагается выбрать встраивание или извлечение информации методом NMI. Далее необходимо выбрать одно из загруженных изображений. Встроим предложение «Quick brown fox jumps over the lazy dog.» в первый рисунок. На рисунке 11 показан результат работы программы.

```
Для встраивания информации выберите E, для извлечения D:  
E  
Выберите номер изображения (1-3):  
1  
Введите сообщение для встраивания:  
Quick brown fox jumps over the lazy dog.  
Ваше стегоизображение 1_new.bmp сохранено.  
  
Ниже представлены полученные метрики:  
  
MSE = 4.743891851668113e-05  
RMSE = 0.006887591633995233  
PSNR = 91.36945581177628  
SSIM = 0.9999999998301942
```

Рисунок 11 – Работа программы, встраивание текста.

Рассмотрим исходное растровое изображение 1.bmp на рисунке 12.



Рисунок 12 – Исходное растровое изображение

Стегоизображение с встроенным текстом представлено на рисунке 13. Внешние различия незаметны для человеческого глаза. Размер изображения увеличен с помощью интерполяции.



Рисунок 13 – Стегоизображение 1_new.bmp

Теперь рассмотрим обратное преобразование. На рисунке 14 представлен результат работы программы при извлечении информации из изображения 1_new.bmp.

```
Для встраивания информации выберите E, для извлечения D:
D
Выберите номер изображения (1-3):
1
Из стегоизображения 1_new.bmp был получено сообщение:
Quick brown fox jumps over the lazy dog.
```

Рисунок 14 – Работа программы, извлечение информации из стегоизображения

В результате была получена текстовая строка, совпадающая с встроенной. Программа сработала верно.

4 Результаты вычислительных экспериментов

4.1 Встраивание максимально возможного для данного контейнера объема информации

Будем работать со вторым изображением. На рисунке 14 изображена функция, возвращающая количество символов, которое можно вставить в изображение методом NMI.

```
def size(image: Image):
    """Длина максимально возможного сообщения

    Args:
        image (Image): Изображение
    Returns:
        float: Максимальная длина сообщения для встраивания
    """
    size = 0
    #буду работать в красном цвете, k = 0
    p_image = interpolation(image)
    width, height = image.size
    P = np.array(p_image)

    #проход по блокам 2 x 2
    for i in range (height):
        for j in range (width):
            Pd = int(P[2*i][2*j][0]) #основной пиксель

            for x, y in [(2*i, 2*j + 1), (2*i + 1, 2*j), (2*i + 1, 2*j + 1)]:
                d = abs(int(P[x][y][0]) - Pd)
                if (d == 0 or d == 1):
                    continue
                n = int(math.log2(d))
                size += n
            size = size // 8
    return(size)
```

Рисунок 15 – Расчёт длины максимального сообщения

Для изображения 2.bmp было получено число 143955. Создадим текстовый файл long_message.txt с нужным числом символов. В основной программе заменим ввод текста с клавиатуры на ввод из файла. На рисунке 16 показан результат встраивания информации.

```
19 choice = input("Для встраивания информации выберите E, для извлечения D:\n")
20 image_number = int(input("Выберите номер изображения (1-3): \n"))
21 if choice.upper() == 'E':
22     image = Image.open(f"{image_number}.bmp")
23     #message = input("Введите сообщение для встраивания: \n")
24     with open('long_message.txt') as f:
25         message = f.read()
26     image1 = interpolation(image)
```

PROBLEMS OUTPUT DEBUG CONSOLE PORTS 6

▼ TERMINAL

```
Для встраивания информации выберите E, для извлечения D:
E
Выберите номер изображения (1-3):
2
Ваше стегоизображение 2_new.bmp сохранено.

Ниже представлены полученные метрики:

MSE = 1.650285484146898

RMSE = 1.2846343776136842

PSNR = 45.95521281220677

SSIM = 0.9999960496355114
```

Рисунок 16 – Встраивание максимально длинного сообщения, метрики
На рисунке 17 показано исходное изображение.



Рисунок 17 – Изображение 2.bmp до встраивания
На рисунке 18 показано стегоизображение.



Рисунок 18 – Стегоизображение 2_mew.bmp

Проверим правильность встраивания. На рисунке 19 изображена часть извлеченной информации и ее длина.


```

30 elif choice.upper() == 'D':
31     new_image = Image.open(f"{image_number}_new.bmp")
32     message = decode(new_image)
33     print(f"Из стегоизображения {image_number}_new.bmp был получено сообщение: \n", message)
34     print(len(message))

```

PROBLEMS OUTPUT **DEBUG CONSOLE** PORTS 6

▼ TERMINAL

```

Summoning up each argument to aid;

As was behooveful for such questioner,

And such profession: "As good Christian ought,

Declare thee, What is faith?" Whereat I rais'd

My forehead to the light, whence this had breath'd,

Then turn'd to Beatrice, and in her looks

Approval met, that from their inmost fount

I should unlock the waters. "May the grace,

That giveth me the captain of the church

For confessor," said I, "vouchsafe to me

Apt utterance for my thoughts!" then added: "Sire!

E'en a
143955

```

Рисунок 19 – Результат извлечения информации

Длина полученного сообщения 143955 совпала с исходной длиной встроенной информации. Строки текста, выведенного в терминале, совпадают с строками из файла long_message.txt. Из чего можно сделать вывод о корректной работе реализованной программы.

4.2 Встраивание разного количества информации

Создадим файлы с разным количеством символов. Встроим информацию во второе изображение по аналогии с встраиванием 100% информации в разделе 4.1. Полученные метрики внесем в таблицу 1 (значения были округлены).

Таблица 1 – Зависимость метрик от количества встроенного текста

Процент встроенного текста	Количество символов	MSE	RMSE	PSNR	SSIM
100%	143955	1.6503	1.2846	45.9552	0.99999605
80%	115164	1.3596	1.1660	46.7967	0.99999684
60%	86373	1.0485	1.0239	47.9253	0.99999764
40%	57582	0.7699	0.8775	49.2664	0.99999832
20%	28791	0.4310	0.6565	51.7863	0.99999909
10%	14395	0.2203	0.4694	54.7008	0.99999954

Проанализируем результаты. Показатель MSE уменьшается с уменьшением встроенного текста. Среднеквадратичная ошибка (MSE) рассчитывается как среднее значение квадратов разностей между прогнозируемыми и фактически наблюдаемыми

значениями пикселей. При уменьшении встроенного текста увеличивается количество совпадающих друг с другом пикселей в исходном (интерполированном) изображении и полученном стегоизображении.

Показатель RMSE является квадратным корнем среднеквадратичной ошибки. По этой причине он также падает при уменьшении встроенной информации.

Метрика PSNR - пиковое отношение сигнал/шум. Этот показатель измеряет качество обработанного изображения путем сравнения его с оригиналом, предполагая, что оба изображения имеют одинаковое разрешение. Соответственно, чем выше среднеквадратичная ошибка, тем меньше показатель PSNR. По результатам видим, что при уменьшении встроенной информации качество изображения улучшается.

Теперь рассмотрим метрику SSIM (structure similarity). SSIM-индекс — это метод полного сопоставления изображений. Он проводит измерение качества на основе исходного изображения (не сжатого или без искажений). Значения SSIM не превосходят 1. По полученным результатам можно оценить, что сходство изображение увеличивается при уменьшении количества встроенной информации.

4.3 Извлечение встроенной информации из стегоизображений

Рассмотрим изображение 3.bmp на рисунке 20.



Рисунок 20 – Изображение 3.bmp




Встроим в это изображение строку «Yesterday all my troubles seemed so far away, now it looks as though they're here to stay. Oh, I believe in yesterday». Повысим яркость полученного стегоизображения на один процент. Попробуем извлечь из нового стегоизображения информацию. В результате извлечения была получена строка из 47910 символов, хотя изначальная строка содержала всего 117 символов. При таком различии в

извлеченных сообщениях нельзя корректно оценивать робастность метода встраивания. Можно сделать вывод, что встроенная информация оказалась неустойчивой к деструктивным воздействиям в виде изменения яркости изображения.

4.4 Сравнение гистограмм изображений до и после встраивания

Рассмотрим изображение номер 3. Встроим в него текст, состоящий из 14395 символов. В таблице 2 приведены исходное изображение 3.bmp, увеличенное изображение 3_big.bmp и стегоизображение 3_new.bmp.

Таблица 2 – Изменение третьего изображения в процессе встраивания информации

3.bmp	3_big.bmp	3_new.bmp
		

С помощью программной реализации, изображенной на рисунке 21 создадим гистограммы для каждого изображения из таблицы. Будем рассматривать гистограммы только для красного цвета, т.к. изменения производились в красном канале.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

image = Image.open("3.bmp")
S = np.array(image)
Red = [S[i][j][0] for i in range(len(S)) for j in range(len(S[0]))]
fig, ax = plt.subplots(figsize=(16,10), layout = "constrained")
n, bins, patches = ax.hist(Red, range(min(Red), max(Red)), facecolor = 'darkred', alpha = 0.75)
ax.set_xlabel("Значение красного цвета в RGB")
ax.set_ylabel("Частота данного значения среди пикселей изображения")
ax.set_title("Значения красного для изображения 3.bmp")
ax.grid(True)
plt.savefig("gr3.jpg")
```

Рисунок 21 – Программная реализация для создания гистограмм

На рисунках 22-24 изображены полученные гистограммы.

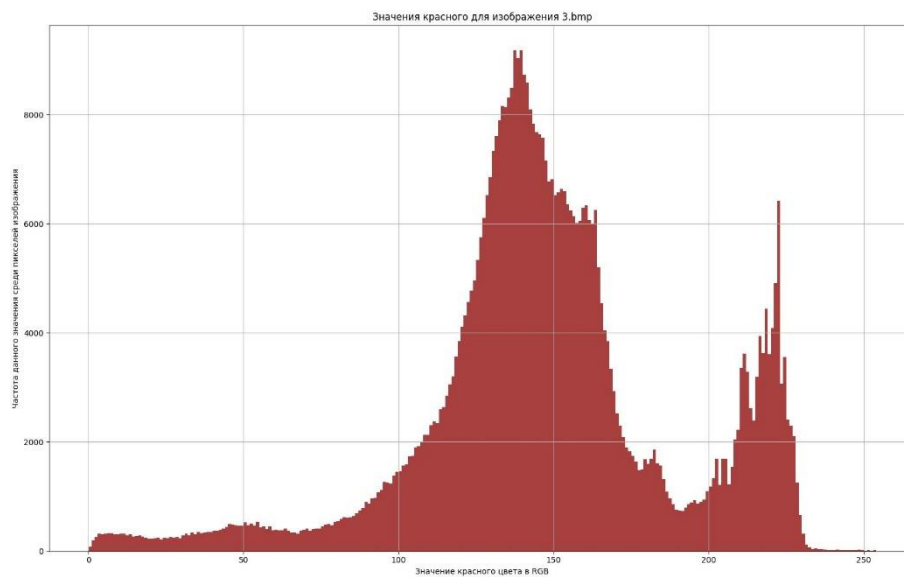


Рисунок 22 – Гистограмма частоты красных значений для изображения 3.bmp

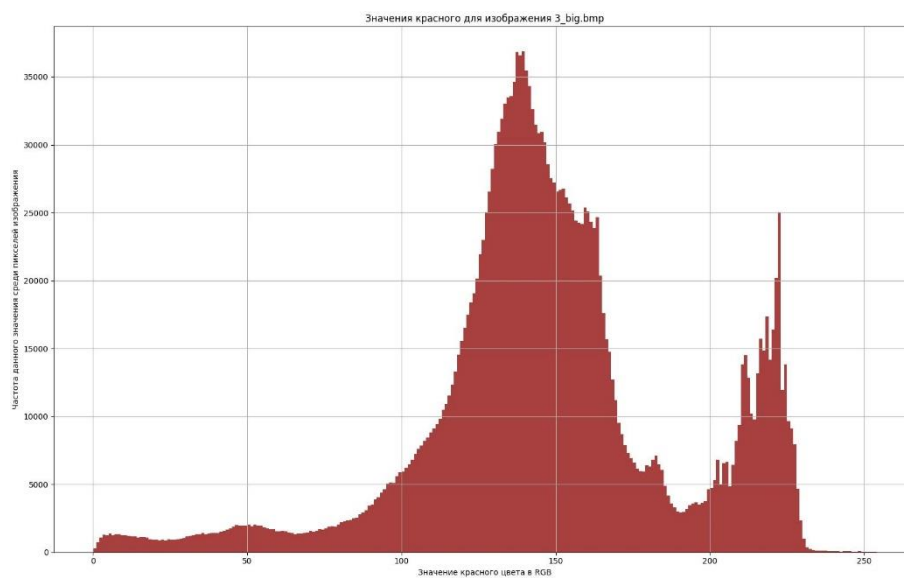


Рисунок 23 – Гистограмма частоты красных значений для изображения 3_big.bmp

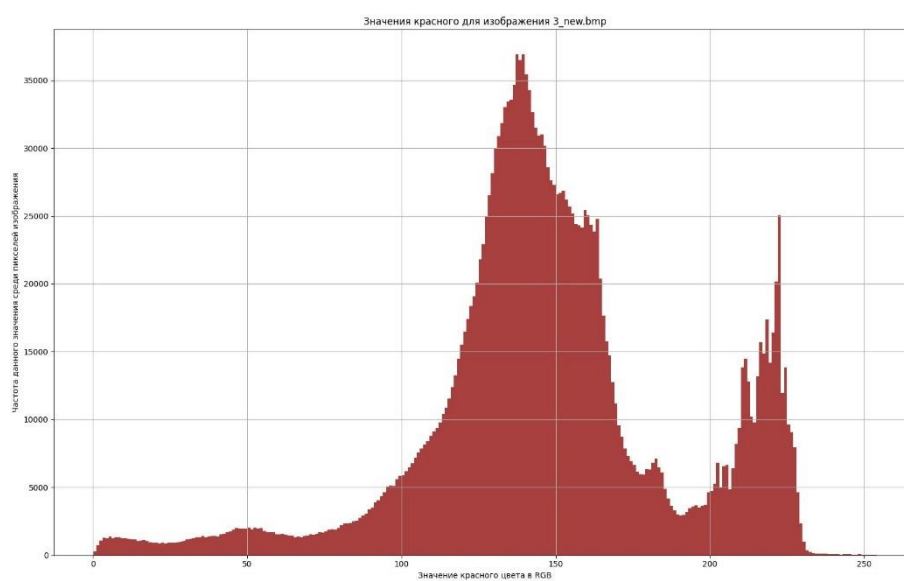


Рисунок 24 – Гистограмма частоты красных значений для изображения 3_new.bmp

Несмотря на различие в количестве пикселей у исходного и увеличенного изображений, их гистограммы выглядят похожими. Это говорит о значительной внешней схожести исходного и интерполированного изображений.

Сравним гистограммы для интерполированного и стегоизображения. Визуально они не имеют отличий. Чтобы увидеть разницу, изобразим гистограммы на одном графике. Будем использовать полупрозрачный красный цвет для гистограммы 3_big.bmp и полупрозрачный синий цвет для гистограммы 3_new.bmp. Таким образом, фиолетовая область будет отвечать за одинаковые значения, а синяя и красная – за различия между гистограммами. Полученный график изображен на рисунке 25.

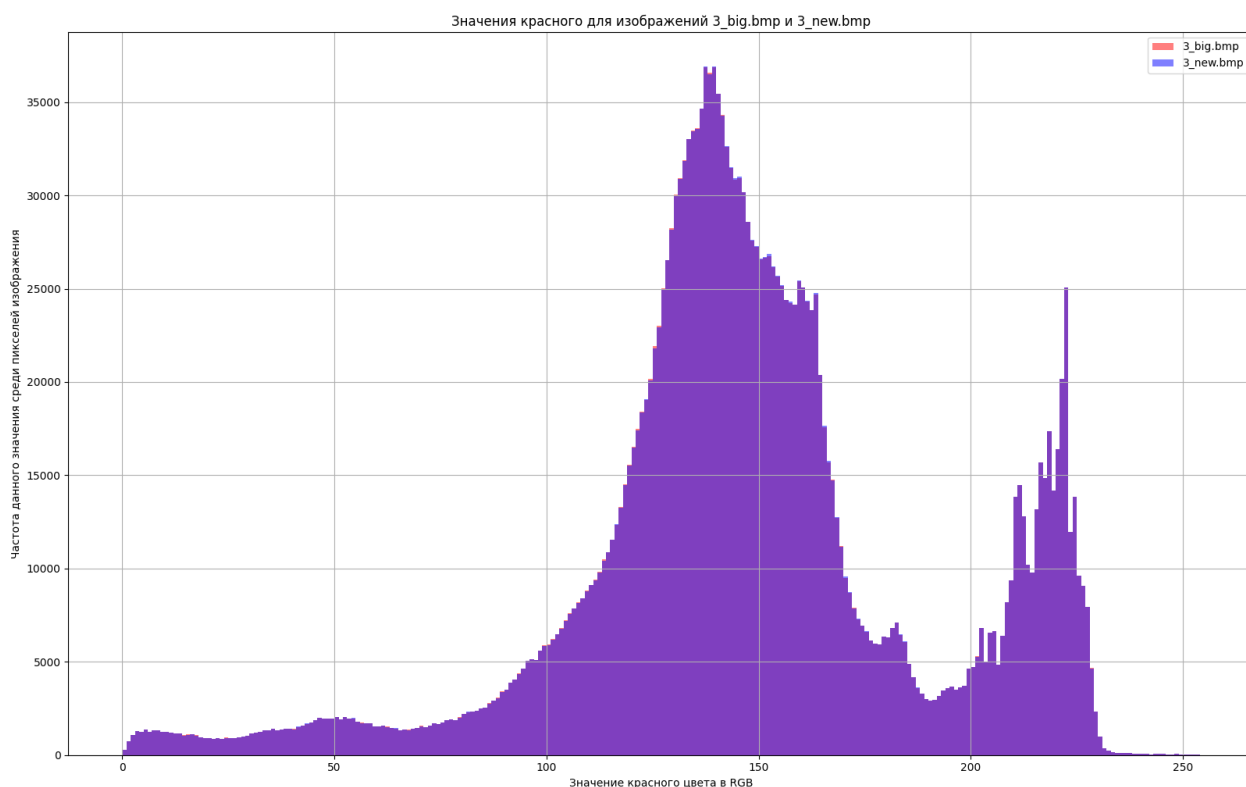


Рисунок 25 – Объединенная гистограмма

По объединенной гистограмме видны незначительные различия между изображениями 3_big.bmp и 3_new.bmp.

5 Выводы о проделанной работе

В ходе проделанной работы были изучены различные методы пространственного встраивания информации в цифровые изображения, а также метрики для оценки эффективности встраивания. На языке python был реализован метод пространственного встраивания Neighbor Mean Interpolation (NMI). С помощью написанного кода были проведены вычислительные эксперименты с встраиванием разного количества информации, изменением полученного стегоизображения.

В результате можно сделать вывод о том, что метод NMI позволяет встроить большое количество информации в цифровое изображение незаметно для человеческого глаза, но при этом не защищает информацию от внешних деструктивных воздействий.

6 Используемая литература

1. О.О. Евсютин – Курс лекций по дисциплине «Основы криптографии и стеганографии»
2. PSNR and SSIM Metric: Python Implementation – CV Notes, URL: <https://cvnote.ddlee.cc/2019/09/12/psnr-ssim-python>
3. Structural similarity index measure – Wikipedia, URL: https://en.wikipedia.org/wiki/Structural_similarity_index_measure
4. Nearest Neighbour Interpolation – Wojik, URL: <https://kwojcicki.github.io/blog/NEAREST-NEIGHBOUR>
5. Mean squared error – Wikipedia, URL: https://en.wikipedia.org/wiki/Mean_squared_error
6. Peak signal-to-noise ratio – Wikipedia, URL: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio
7. SSIM – Wikipedia, URL: <https://ru.wikipedia.org/wiki/SSIM>