Серт Серкан, группа 8

Лабораторная работа №1

**Метод LSB (Least Significant Bit)**

Вариант№3   
  
Использование метода LSB базируется на невосприимчивости человеческих органов чувств к малозначительным изменениям в контейнерах, обладающих психовизуальной избыточностью. При встраивании данных в пространственной области графических контейнеров метод LSB реализует замену наименее значимых бит (НЗБ) значений яркости или цветности отдельных пикселей контейнера битами скрываемого сообщения

Цель работы:

Реализовать LSB-алгоритм. В качестве метрик для оценки искажений заполненных контейнеров использовать **μmax D , μSNR , μPSNR** . Построить зависимости вероятности ошибок при извлечении скрытых данных от объема скрываемой информации.

Код программы:

from PIL import Image

import numpy as np

import warnings

warnings.filterwarnings(action='once')

encoding: str = 'utf-8'

class LSB:

    def \_\_init\_\_(self, old\_image\_path: str, new\_image\_path: str):

        self.\_\_empty\_image\_path: str = old\_image\_path

        self.\_\_full\_image\_path: str = new\_image\_path

        self.\_\_occupancy: int = 0

    @staticmethod

    def str\_to\_bits(message: str) -> list:

        result = []

        for num in list(message.encode(encoding=encoding)):

            result.extend([(num >> x) & 1 for x in range(7, -1, -1)])

        return result

    @staticmethod

    def bits\_to\_str(bits: list) -> str:

        chars = []

        for b in range(len(bits) // 8):

            byte = bits[b \* 8:(b + 1) \* 8]

            chars.append(chr(int(''.join([str(bit) for bit in byte]), 2)))

        return ''.join(chars)

    def embed(self, message: str,new\_image\_path: str):

        img = Image.open(self.\_\_empty\_image\_path).convert('RGB')

        picture = np.asarray(img, dtype='uint8')

        picture = picture.astype('uint8')

        img.close()

        picture\_shape = picture.shape

        height, width, depth = picture.shape[0], picture.shape[1], picture.shape[2]

        message\_bits = LSB.str\_to\_bits(message)

        if len(message\_bits) > height \* width \* depth:

            raise ValueError('Message greater than capacity!')

        message\_bits = np.asarray(message\_bits)

        bits\_length = message\_bits.shape[0]

        picture = picture.reshape(-1)

        picture[:bits\_length] = ((picture[:bits\_length] >> 1) << 1) | message\_bits

        picture = picture.reshape(picture\_shape)

        print('bits',message\_bits)

        print('bits\_leng',bits\_length)

        self.\_\_occupancy = bits\_length

        Image.fromarray(picture).save(new\_image\_path, 'PNG')  # Сохраняем новое изображение

        self.\_\_occupancy = bits\_length

        Image.fromarray(picture).save(self.\_\_full\_image\_path, 'PNG')

    def recover(self) -> str:

        img = Image.open(self.\_\_full\_image\_path).convert('RGB')

        picture = np.asarray(img, dtype='uint8')

        picture = picture.astype('uint8')

        img.close()

        recovered\_message = picture.reshape(-1)[:self.\_\_occupancy] & 0x01

        return LSB.bits\_to\_str(list(recovered\_message))

    @property

    def occupancy(self) -> int:

        return self.\_\_occupancy

def metrics(empty\_image: str, full\_image: str) -> None:

    img = Image.open(empty\_image).convert('RGB')

    empty = np.asarray(img, dtype='uint8')

    img.close()

    img = Image.open(full\_image).convert('RGB')

    full = np.asarray(img, dtype='uint8')

    img.close()

    MAX\_D = np.max(np.abs(empty.astype(int) - full.astype(int)))

    SNR = np.sum(empty \* empty) / np.sum((empty - full) \*\* 2)

    H, W = empty.shape[0], empty.shape[1]

    MSE = np.sum((empty - full) \*\* 2) / (W \* H)

    max\_pixel = 255.0

    PSNR = 20 \* np.log10(max\_pixel / np.sqrt(MSE))

    sigma = np.sum((empty - np.mean(empty)) \* (full - np.mean(full))) / (H \* W)

    UQI = (4 \* sigma \* np.mean(empty) \* np.mean(full)) / \

          ((np.var(empty) \*\* 2 + np.var(full) \*\* 2) \* (np.mean(empty) \*\* 2 + np.mean(full) \*\* 2))

    print(f'Максимальное абсолютное отклонение (MAX): {MAX\_D}')

    print(f'Отношение сигнал-шум (SNR): {SNR}')

    print(f'Среднее квадратичное отклонение (MSE): {MSE}')

    print(f'Пиковое отношение сигнал-шум (PSNR): {PSNR}')

    print(f'Универсальный индекс качества (УИК): {UQI}')

def extract\_with\_capacity(full\_image\_path, capacity):

    lsb = LSB(full\_image\_path, full\_image\_path)  # Используем тот же путь для извлечения

    message = lsb.recover()  # Извлекаем скрытое сообщение

    extracted\_message = message[:capacity]  # Получаем часть сообщения с заданным объемом

    return extracted\_message

def main():

    old\_image = 'input/SERT\_.png'

    new\_image = 'output/sert\_.png'

    with open('message.txt', mode='r', encoding=encoding) as file:

        message = file.read()

    capacities = [len(message)]  # Разные объемы сообщений для тестирования

    for capacity in capacities:

        new\_image\_path = 'output/sert\_capacity\_{}.png'.format(capacity)

        lsb = LSB(old\_image, new\_image\_path)

        lsb.embed(message[:capacity], new\_image\_path)  # Вставляем сообщение и сохраняем новое изображение

        recovered\_message = lsb.recover()

        error\_probability = sum(1 for a, b in zip(message[:capacity], recovered\_message) if a != b) / capacity

        print(f'Объем информации: {capacity}, Вероятность ошибки: {error\_probability}')

        print('сообщение:\t{}'.format(recovered\_message))

        metrics(old\_image, new\_image)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Результат работы программы:

Исходное изображение (пустой контейнер)



Изображение со встроенным сообщением (заполненный контейнер)



Результаты работы программы (введено сообщение «My secret messages»):

