Серт Серкан, группа 8

Лабораторная работа №4

Алгоритм Коха

ВариантNo3

В соответствие со своим вариантом реализовать стеганографический алгоритм скрытия данных в пространственной области контейнеров- изображений. Оценить уровень вносимых искажений заполненных контейнеров с использованием объективных метрик и устойчивость встроенной информации по отношению негативному воздействию на заполненный контейнер

Цель работы:

Реализовать алгоритм Сох. В качестве метрик для оценки искаженийmаркированных контейнеров использовать **μ PSNR, μMSE , μUQI**. Оценить устойчивость встроенных ЦВЗ по отношению к повороту с последующим восстановлением и обрезке с заменой данными из исходного контейнера.

Код программы:

```
import io
import numpy as np
import scipy
from PIL import Image

#задачи 3
def string_to_binary(string: str) -> str:
    binary_list = []
    for c in string:
        binary = bin(ord(c))[2:].zfill(8)
        binary_list.append(binary)
    return ''.join([b for b in binary_list])


def binary_to_string(binary: str) -> str:
    characters = []
      # 8 bitlik ikili diziyi ayır ve karakterlere dönüştür
    for i in range(0, len(binary), 8):
        b = binary[i: i + 8]
        integer = int(b, 2)
        character = chr(integer)
        characters.append(character)
         # Karakterleri birleştirerek sonucu elde et
    return ''.join(characters)


def metrics(empty_image: str, full_image: str) -> None:
    with Image.open(empty_image).convert('L') as image:
        empty = np.asarray(image, dtype=np.uint8)

    with Image.open(full_image).convert('L') as image:
        full = np.asarray(image, dtype=np.uint8)
```

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

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

    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'Пиковое отношение сигнал-шум(PSNR):{PSNR}')
    print('Среднее квадратичное отклонение(MSE):{}'.format(MSE))
    print(f'Универсальный индекс качества(УИК):{UQI}')


def define_starts_of_blocks(height: int, width: int, n: int) ->
list[tuple]:
    two_d_list = [[tuple([i, j]) for j in range(0, width - n + 1, n)] for i
in range(0, height - n + 1, n)]
    one_d_list = [item for sublist in two_d_list for item in sublist]
    return one_d_list


class Cox:
    def __init__(self):
        self.__e: int = 2
        self.__occupancy = 0

    def embed(self, old_image: str, new_image: str, message: str, key: int)
-> bool:

        with Image.open(old_image) as image:

            pixels = np.asarray(image.convert('L'))

            quantization_table = image.quantization
        height, width = pixels.shape[0:2]
        #mesajın uzunluğunu kontrol et

        binary_seq = string_to_binary(message)
        #print(binary_seq)
        if len(binary_seq) > (height * width) // 64:
            raise ValueError('Message greater than capacity')

        start_points = define_starts_of_blocks(height, width, 8)
        np.random.seed(key)
        np.random.shuffle(start_points)
        np.random.seed()

        for i, bit in enumerate(binary_seq):
            start_point = start_points[i]
            block = pixels[start_point[0]: start_point[0] + 8,
start_point[1]: start_point[1] + 8].copy()

            dct_block = scipy.fftpack.dct(block)

            mid_freq_coeffs = np.asarray([dct_block[5, 4], dct_block[4,
```

```
4]])

            const = self.__e // 2
            if bit:
                while np.abs(mid_freq_coeffs[0]) -
np.abs(mid_freq_coeffs[1]) >= -self.__e:
                    mid_freq_coeffs[0] = mid_freq_coeffs[0] - const if
mid_freq_coeffs[0] > 0 \
                        else mid_freq_coeffs[0] + const
                    mid_freq_coeffs[1] = mid_freq_coeffs[1] + const if
mid_freq_coeffs[1] > 0 \
                        else mid_freq_coeffs[1] - const
                assert np.abs(mid_freq_coeffs[0]) <
np.abs(mid_freq_coeffs[1])
            else:
                while np.abs(mid_freq_coeffs[0]) -
np.abs(mid_freq_coeffs[1]) <= self.__e:
                    mid_freq_coeffs[0] = mid_freq_coeffs[0] + const if
mid_freq_coeffs[0] > 0 \
                        else mid_freq_coeffs[0] - const
                    mid_freq_coeffs[1] = mid_freq_coeffs[1] - const if
mid_freq_coeffs[1] > 0 \
                        else mid_freq_coeffs[1] + const
                assert np.abs(mid_freq_coeffs[0]) >
np.abs(mid_freq_coeffs[1])

            dct_block[5, 4] = mid_freq_coeffs[0]
            dct_block[4, 4] = mid_freq_coeffs[1]

            #print(dct_block)
            modified_block = np.round(dct_block /
np.array(quantization_table[0]).reshape(8, 8)).astype(np.uint8)
            pixels[start_point[0]: start_point[0] + 8, start_point[1]:
start_point[1] + 8]


        Image.fromarray(pixels).save(new_image, qtables=quantization_table)
        self.__occupancy = len(binary_seq)
        return True

    def recover(self, new_image: str, key: int) -> str:

        with Image.open(new_image) as image:
            pixels = np.asarray(image.convert('L'))
            quantization_table = image.quantization
        #print(quantization_table)
        height, width = pixels.shape[0: 2]

        start_points = define_starts_of_blocks(height, width, 8)
        np.random.seed(key)
        np.random.shuffle(start_points)
        np.random.seed()

        buffer_binary = io.StringIO()
        for start_point in start_points[:self.__occupancy]:
            block = pixels[start_point[0]: start_point[0] + 8,
start_point[1]: start_point[1] + 8].copy()
            dct_block = scipy.fftpack.dct(block) *
np.array(quantization_table[0]).reshape(8, 8)
            mid_freq_coeffs = np.array([dct_block[5, 4], dct_block[4, 4]])
```

```
            if np.abs(mid_freq_coeffs[0]) > np.abs(mid_freq_coeffs[1]):
                buffer_binary.write('0')
            elif np.abs(mid_freq_coeffs[0]) <= np.abs(mid_freq_coeffs[1]):
                buffer_binary.write('1')

        #print(buffer_binary.getvalue())
        return binary_to_string(buffer_binary.getvalue())


def main():
    key = 1234

    old_image = 'input/sert.jpg'
    new_image = 'output/sert_new_image.jpg'

    with open('message.txt', mode='r', encoding='utf-8') as file:
        message = file.read()

    cox = Cox()
    cox.embed(old_image, new_image, message, key)
    recovered_message = cox.recover(new_image, key)
    print('Messages:{}'.format(recovered_message))

    metrics(old_image, new_image)

    bin_original = np.array(list(map(int, string_to_binary(message))))
    bin_recovered = np.array(list(map(int,
string_to_binary(recovered_message))))
    print('двух бинарных массивов',np.mean(bin_original == bin_recovered))


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»):

```
C:\Users\Serkan\AppData\Local\Programs\Python\Python312\py
Messages:3óúóÞú:vÿÿó
Пиковое отношение сигнал-шум(PSNR):87062328.88888888
Среднее квадратичное отклонение(MSE):0.000706436420722135
Универсальный индекс качества(УИК):0.00024518367075305455
двух бинарных массивов 0.5454545454545454

Process finished with exit code 0
```