

Серт Серкан, группа 8

Лабораторная работа №3

## Алгоритм Брайндокса (Bruyndonckx)

Вариант №2

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

Цель работы:

Реализовать алгоритм Брайндокса. В качестве метрик для оценки искажений заполненных контейнеров использовать  $\mu_{SNR}$ ,  $\mu_{MSE}$ ,  $\mu_{LMSE}$ . Для формирования последовательности  $r_i \in \{-1, +1\}$  использовать генератор псевдо-случайных бит

Код программы:

```
from collections import deque
from PIL import Image
import numpy as np
from typing import List
#Вариант №2
encoding: str = 'utf-8'

class Pixel:
    pointer: int = 0

    def __init__(self, rgba: np.array):
        self.rgba: np.array = rgba
        self.order = Pixel.pointer
        Pixel.pointer += 1

class BruyndonckxMethod:
    def __init__(self, empty_image_path: str, filled_image_path: str):
        self.empty_image_path: str = empty_image_path
        self.filled_image_path: str = filled_image_path
        self.delta_l = 5
        self.occupancy: int = 0

    @staticmethod
    def str_to_bits(message: str):
        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 __func(self):
    g = {}
    for i in range(64):
        group = np.random.choice(['1A', '1B']) if i < 32 else
np.random.choice(['2A', '2B'])
        g.setdefault(group, []).append(i)
    return g

    def modification_brightness(self, sorted_block_pixels: List[Pixel],
bit:int):
        g = self.__func()
        arr = np.asarray([pixel.rgb for pixel in
sorted_block_pixels], dtype=np.uint8)
        delta_l = self.delta_l
        sign = 1 if bit else -1
        g1_arr = arr[(g['1A'] + g['1B'])]
        arr[g['1A'], 3] -= (np.mean(arr[g['1A'], 3]) - (np.mean(g1_arr[:,
3]) + (sign * arr[g['1B']].shape[0] * delta_l /
g1_arr.shape[0]))) .astype(np.uint8)
        arr[g['1B'], 3] -= (np.mean(arr[g['1B'], 3]) - (np.mean(g1_arr[:,
3]) - (sign * arr[g['1A']].shape[0] * delta_l /
g1_arr.shape[0]))) .astype(np.uint8)

        g2_arr = arr[(g['2A'] + g['2B'])]
        arr[g['2A'], 3] -= (np.mean(arr[g['2A'], 3]) - (np.mean(g2_arr[:,
3]) + (sign * arr[g['2B']].shape[0] * delta_l /
g2_arr.shape[0]))) .astype(np.uint8)
        arr[g['2B'], 3] -= (np.mean(arr[g['2B'], 3]) - (np.mean(g2_arr[:,
3]) - (sign * arr[g['2A']].shape[0] * delta_l /
g2_arr.shape[0]))) .astype(np.uint8)
        for i, pixel in enumerate(arr):
            sorted_block_pixels[i].rgb = pixel

    def embed(self, message: str, key_generator: int):
        np.random.seed(key_generator)
        with Image.open(self.empty_image_path).convert('RGBA') as img:
            picture = np.asarray(img, dtype=np.uint8).astype(np.uint8)
            picture[:, :, 3] = (0.299 * picture[:, :, 0] + 0.587 *
picture[:, :, 1] + 0.114 * picture[:, :, 2]).astype(int)
            height, width = picture.shape[0], picture.shape[1]
            message_bits = self.str_to_bits(message)
            message_bits_length = len(message_bits)
            if message_bits_length > (height // 8) * (width // 8):
                raise ValueError('Размер сообщения превышает размер
контейнера!')
            message_bits = deque(message_bits)
            for i in range(8, height - 7, 8):
                for j in range(8, width - 7, 8):
                    old_block = picture[i - 8: i, j - 8: j].copy()
                    old_size = old_block.shape
                    old_block = old_block.reshape(-1, 4)
                    new_block = sorted([Pixel(pixel) for pixel in old_block],
key=lambda obj: obj.rgb[3])
                    bit = message_bits.popleft()

```

```

        self.modification_brightness(new_block, bit)
        new_block = sorted(new_block, key=lambda obj: obj.order)
        new_block = (np.asarray([pixel.rgb for pixel in
new_block],
dtype=np.uint8)).reshape(old_size)
        picture[i - 8: i, j - 8: j] = new_block[:, :]
        self.occupancy += 1
        if self.occupancy == message_bits_length:
            Image.fromarray(picture,
'RGBA').save(self.filled_image_path, 'PNG')
            np.random.seed()
            return

def recover(self, key_generator: int):
    np.random.seed(key_generator)
    with Image.open(self.filled_image_path).convert('RGBA') as img:
        picture = np.asarray(img, dtype=np.uint8)
        height, width = picture.shape[0], picture.shape[1]
        message_bits = []
        for i in range(8, height - 7, 8):
            for j in range(8, width - 7, 8):
                modified_block = picture[i - 8: i, j - 8: j].copy()
                modified_block = modified_block.reshape(-1, 4)
                modified_block = sorted([Pixel(pixel) for pixel in
modified_block], key=lambda pixel: np.uint8(
                    0.299 * pixel.rgb[0] + 0.587 * pixel.rgb[1] + 0.114 *
pixel.rgb[2]))
                g = self.__func__()
                arr = np.asarray([pixel.rgb for pixel in modified_block],
dtype=np.uint8)
                if (np.mean(arr[g['1A'], 3]) - np.mean(arr[g['1B'], 3]) >
0) and \
                    (np.mean(arr[g['2A'], 3]) - np.mean(arr[g['2B'], 3]) >
0): message_bits.append(1)
                else:
                    message_bits.append(0)
            if len(message_bits) == self.occupancy:
                np.random.seed()
                message = self.bits_to_str(message_bits)
                return message

def metrics(empty_image_path: str, filled_image_path: str):
    with Image.open(empty_image_path).convert('RGBA') as img:
        empty = np.asarray(img, dtype=np.uint8).astype(np.uint8)
        empty[:, :, 3] = (0.299 * empty[:, :, 0] + 0.587 * empty[:, :, 1] +
0.114 * empty[:, :, 2]).astype(np.uint8)
    with Image.open(filled_image_path).convert('RGBA') as img:
        full = np.asarray(img, dtype=np.uint8)

    H, W = empty.shape[0], empty.shape[1]
    maxD = np.sum((empty - full) * (empty - full)) / np.sum((empty *
empty))
    Lp = 1 / maxD
    MSE = np.sum((empty - full) ** 2) / (W * H)

    print('Максимальное абсолютное отклонение: {}'.format(maxD))
    print('Норма Минковского = {}'.format(Lp))
    print('Среднее квадратичное отклонение: {}'.format(MSE))

```

```
empty_image_path = 'input/old_image.png'
filled_image_path = 'output/new_image.png'

with open('message.txt', mode='r', encoding=encoding) as file:
    message = file.read()
key = 21532
b = BruyndonckxMethod(empty_image_path, filled_image_path)
b.embed(message, key)
recovered_message = b.recover(key)
print('Зашированное сообщение:{}'.format(recovered_message))
metrics(empty_image_path, filled_image_path)
```

Результат работы программы:

Исходное изображение (пустой контейнер)



Изображение со встроенным сообщением (заполненный контейнер)



Результаты работы программы (введено сообщение « Secret Code»):

```
C:\Users\Serkan\AppData\Local\Programs\Python\Python312\py
Зашированное сообщение:Secret Code
Максимальное абсолютное отклонение:0.00015337316986503097
Норма Минковского = 6520.045200082936
Среднее квадратичное отклонение:0.07280219780219781

Process finished with exit code 0
```