Серт Серкан, группа 8

Лабораторная работа №3

**Алгоритм Брайндокса (Bruyndonckx)**

Вариант№2  
  
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 (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 . Для формирования последовательности  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.rgba 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].rgba = 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.rgba[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.rgba 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.rgba[0] + 0.587 \* pixel.rgba[1] + 0.114 \* pixel.rgba[2]))  
 g = self.\_\_func()  
 arr = np.asarray([pixel.rgba 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»):

