Серт Серкан, группа 8

Лабораторная работа №4

**Алгоритм Коха**

Вариант№3

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 Cox. В качестве метрик для оценки искажениймаркированных 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»):

