Серт Серкан, группа 8

Лабораторная работа №2

Вариант№1

**Метод Куттера-Джордана-Боссена**

относится к группе вероятностных методов стеганографического скрытия, реализующих встраивание битов сообщения в выбранные элементы пространственной области контейнеров изображений, представленных в цветовой модели RGB. Пространство сокрытия в данном методе формируется из значений синих цветовых компонент выбранного множества пикселей контейнера. Для встраивания данных выбирается синий цветовой канал, поскольку изменения в данном канале являются перцептивно наименее заметными.

Цель работы:

Реализовать метод Куттера-Джордана-Боссена. В качестве метрик для оценки искажений заполненных контейнеров . Построить зависимости вероятности ошибок при извлечении скрытых данных от энергии встраиваемого сигнала

Код программы:

from PIL import Image  
import numpy as np  
from dotenv import load\_dotenv  
import os  
import pandas as pd  
import matplotlib.pyplot as plt  
import matplotlib.style as style  
import seaborn as sns  
import warnings  
warnings.filterwarnings(action='once')  
encoding: str = 'utf-8'  
large = 22  
med = 16  
small = 12  
params = {'axes.titlesize': large,  
 'legend.fontsize': med,  
 'figure.figsize': (16, 10),  
 'axes.labelsize': med,  
 'axes.titlesize': med,  
 'xtick.labelsize': med,  
 'ytick.labelsize': med,  
 'figure.titlesize': large}  
plt.rcParams.update(params)  
plt.style.use('seaborn-v0\_8-white')  
  
class Generator:  
 def \_\_init\_\_(self, base: int, key: int):  
 self.base = base  
 self.\_first\_index = 40  
 self.\_second\_index = 0  
 self.\_buffer = [i for i in range(abs(key), abs(key) + 60)]  
 # Находим следующий {Xi}  
 def next(self) -> int:  
 value = (self.\_buffer[self.\_second\_index] + self.\_buffer[self.\_first\_index]) % self.base  
 del self.\_buffer[self.\_second\_index]  
 self.\_buffer.append(value)  
 return value  
  
 @property  
 def base(self) -> int:  
 return self.\_base  
  
 @base.setter  
 def base(self, value: int) -> None:  
 if isinstance(value, int):  
 if value <= 0:  
 raise ValueError('base > 0!')  
 self.\_base = value  
  
class KutterMethod:  
 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.\_\_lam: float = 1  
 self.\_\_sigma: int = 1  
 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, key\_generator: int):  
 img = Image.open(self.\_\_empty\_image\_path).convert('RGB')  
 image = np.asarray(img, dtype='uint8')  
 img.close()  
 height, width = image.shape[0], image.shape[1]  
 message\_bits = KutterMethod.str\_to\_bits(message)  
 if len(message\_bits) > height \* width:  
 raise ValueError('Размер сообщения превышает размер контейнера!')  
 # использованные пиксели  
 keys = []  
 generator = Generator(base=height \* width, key=key\_generator)  
 for bit in message\_bits:  
 coordinate = generator.next()  
 while coordinate in keys:  
 coordinate = generator.next()  
 keys.append(coordinate)  
 i, j = divmod(coordinate, width)  
 pixel = image[i, j]  
 lam = self.lam  
 L = 0.299 \* pixel[0] + 0.587 \* pixel[1] + 0.114 \* pixel[2]  
 if bit == 1:  
 pixel\_copy = pixel.copy()  
 pixel\_copy[2] = np.uint8(min(255, pixel[2] + lam \* L))  
 elif bit == 0:  
 pixel\_copy = pixel.copy()  
 pixel\_copy[2] = np.uint8(max(0, pixel[2] - lam \* L))  
 self.\_\_occupancy = len(message\_bits)  
 Image.fromarray(image).save(self.\_\_full\_image\_path, 'PNG')  
  
 def recover(self, key\_generator: int) -> str:  
 img = Image.open(self.\_\_full\_image\_path).convert('RGB')  
 image = np.asarray(img, dtype='uint8')  
 img.close()  
 height, width = image.shape[0], image.shape[1]  
 keys = []  
 generator = Generator(base=height \* width, key=key\_generator)  
 while len(keys) < self.occupancy:  
 coordinate = generator.next()  
 while coordinate in keys:  
 coordinate = generator.next()  
 keys.append(coordinate)  
 message\_bits = []  
 for coordinate in keys:  
 i, j = divmod(coordinate, width)  
 sigma = self.sigma  
 summary = 0  
 for n in range(1, sigma + 1):  
 if 0 <= i - n < height and 0 <= j < width:  
 summary += image[i - n, j, 2]  
 if 0 <= i + n < height and 0 <= j < width:  
 summary += image[i + n, j, 2]  
 if 0 <= i < height and 0 <= j - n < width:  
 summary += image[i, j - n, 2]  
 if 0 <= i < height and 0 <= j + n < width:  
 summary += image[i, j + n, 2]  
 if image[i, j, 2] > (summary / (4 \* sigma)):  
 message\_bits.append(1)  
 else:  
 message\_bits.append(0)  
 recovered\_message = KutterMethod.bits\_to\_str(message\_bits)  
 return recovered\_message  
  
 @property  
 def sigma(self) -> int:  
 return self.\_\_sigma  
  
 @sigma.setter  
 def sigma(self, value: int) -> None:  
 if isinstance(value, int):  
 if value <= 0:  
 raise ValueError('sigma > 0!')  
 self.\_\_sigma = value  
  
 @property  
 def lam(self) -> float:  
 return self.\_\_lam  
  
 @lam.setter  
 def lam(self, value: float) -> None:  
 if isinstance(value, float):  
 if abs(value) < 1E-14:  
 raise ValueError('lambda > 0!')  
 self.\_\_lam = value  
  
 @property  
 def occupancy(self) -> int:  
 return self.\_\_occupancy  
  
def error\_probability\_analysis(message: str, key ,old\_image: str, new\_image: str):  
 lam\_values = [] # Değişen lambda (lam) değerleri  
 sigma\_values = [] # Değişen sigma değerleri  
 error\_probabilities = [] # Hata olasılıkları  
 message\_bits = np.asarray(KutterMethod.str\_to\_bits(message))  
 for lam in (0.5, 1, 1.5, 2, 2.5, 3): # İstenilen lambda değerlerini belirleyin  
 kutter = KutterMethod(old\_image, new\_image)  
 kutter.lam = lam  
 kutter.embed(message, key)  
 for sigma in (1, 2, 3, 4, 5, 6, 7):   
 kutter.sigma = sigma # Sigma değerini ayarla  
 kutter.embed(message, key) # 'key\_range' değişkenini kullan  
 recovered\_message = kutter.recover(key)  
 error\_count = sum(1 for m1, m2 in zip(message, recovered\_message) if m1 != m2)  
 error\_probability = error\_count / len(message)  
 lam\_values.append(lam)  
 sigma\_values.append(sigma)  
 error\_probabilities.append(error\_probability)  
 return lam\_values, error\_probabilities, sigma\_values  
   
def dependence(key: int, old\_image: str, new\_image: str, message: str):  
 d = dict()  
 message\_bits = np.asarray(KutterMethod.str\_to\_bits(message))  
 for lam in (0.5, 1, 1.5, 2, 2.5, 3):  
 kutter = KutterMethod(old\_image, new\_image)  
 kutter.lam = lam  
 kutter.embed(message, key)  
 for sigma in (1, 2, 3, 4, 5, 6, 7):  
 kutter.sigma = sigma  
 recovered\_message = kutter.recover(key)  
 recovered\_message\_bits = np.asarray(KutterMethod.str\_to\_bits(recovered\_message))  
 d.setdefault('lambda', []).append(lam)  
 d.setdefault('sigma', []).append(sigma)  
 d.setdefault('e\_probability', []).append(  
 np.mean(np.abs(message\_bits - recovered\_message\_bits[:message\_bits.shape[0]])) \* 100)  
 df = np.round(pd.DataFrame(d), decimals=2)  
 df.to\_csv('log.csv', sep='\t', encoding=encoding)  
 print('Tablo:')  
 print(df)  
 print('Korelasyon:')  
 print(np.round(df.corr(), decimals=2))  
  
 df.groupby('lambda')['e\_probability'].mean().plot(kind='bar', grid=True, ylim=0)  
 plt.show()  
 df.groupby('sigma')['e\_probability'].mean().plot(kind='bar', grid=True, ylim=0)  
 plt.show()  
  
  
  
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)))  
 epsilon = 1e-10 # A small constant to avoid division by zero  
 SNR = np.sum(empty \* empty) / (np.sum((empty - full) \*\* 2) + epsilon)  
 H, W = empty.shape[0], empty.shape[1]  
 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('Максимальное абсолютное отклонение:{}'.format(max\_d))  
 print('Отношение сигнал-шум:{}'.format(SNR))  
 print('Среднее квадратичное отклонение:{}'.format(MSE))  
 print(f'Универсальный индекс качества (УИК):{UQI}\n')  
  
def main():  
 load\_dotenv('.env')  
 key= 1500  
 old\_image = 'input/s.png'  
 new\_image = 'output/S\_new.png'  
 with open('message.txt', mode='r', encoding=encoding) as file:  
 message = file.read()  
 lam\_values, sigma\_values, error\_probabilities = error\_probability\_analysis(message, key, old\_image, new\_image)  
 # Verileri bir veri çerçevesine ekleyin  
 data = {'Lambda': lam\_values, 'Sigma': sigma\_values, 'Error Probability': error\_probabilities}  
 df = pd.DataFrame(data)  
 # Verileri CSV dosyasına kaydedin  
 df.to\_csv('error\_analysis.csv', index=False)  
 plt.figure(figsize=(10, 6))  
 plt.plot(lam\_values, error\_probabilities, marker='o', linestyle='-')  
 plt.title('Hata Olasılığı vs. Lambda Değeri')  
 plt.xlabel('Lambda Değeri')  
 plt.ylabel('Hata Olasılığı')  
 plt.grid(True)  
 plt.show()  
 kutter = KutterMethod(old\_image, new\_image)  
 kutter.embed(message, key)  
 recovered\_message = kutter.recover(key)  
 print('Ваше сообщение:{}'.format(recovered\_message))  
 print(message)  
 metrics(old\_image, new\_image)  
 dependence(key, old\_image, 'output/imageS.png', message)  
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»):

