Project2 图片水印嵌入和提取

学院 网络空间安全

专业 网络空间安全

学号 202200460149

班级姓名 网安 22.1 班张弛

2025 年 7 月 12 日

目录

1　实验任务

编程实现图片水印嵌入和提取（可依托开源项目二次开发），并进行鲁棒性测试，包括不限于翻转、平移、截取、调对比度等

2 图片水印嵌入和提取

在图片水印嵌入和提取这一块，我依托开源项目进行二次开发，开源项目链接如下所示：

https://github.com/guofei9987/blind_watermark

其具体技术如下所示：

分类	**技术方法**
水印类型	盲水印（Blind Watermark）
嵌入域	小波-离散余弦混合变换域（DWT+DCT）
嵌入方式	奇异值分解（SVD）调整嵌入
安全机制	DCT 系数打乱（加密）+ 水印 bit 打乱（加密）
多通道嵌入	对图像的 YUV 色彩空间的 Y, U, V 三个通道分别嵌入
提取方式	无需原图（盲提取），使用 k-means 聚类还原 bit

表 1 整体采用的图片水印技术

2.1 图片水印和提取

该图片水印模型已经较为完善了，因此没有修改相关技术的逻辑，只是在此基础上做了微调。

主要的调整是在提取图片水印时调整了处理方式和格式，详细代码如下所示：

```
#blind_waterark.py
#!/usr/bin/env python3
# coding=utf-8
# @Time    : 2020/8/13
# @Author  : github.com/guofei9987
import warnings

import numpy as np
import cv2

from bwm_core import WaterMarkCore
#from version import bw_notes


class WaterMark:
    def __init__(self, password_wm=1, password_img=1, block_shape=(4, 4), mode='common', processes=None):
        #bw_notes.print_notes()

        self.bwm_core = WaterMarkCore(password_img=password_img, mode=mode, processes=processes)

        self.password_wm = password_wm

        self.wm_bit = None
        self.wm_size = 0

    def read_img(self, filename=None, img=None):
        if img is None:
            # 从文件读入图片
            img = cv2.imread(filename, flags=cv2.IMREAD_UNCHANGED)
            assert img is not None, "image file '{filename}' not read".format(filename=filename)

        self.bwm_core.read_img_arr(img=img)
        return img

    def read_wm(self, wm_content, mode='img'):
        assert mode in ('img', 'str', 'bit'), "mode in ('img','str','bit')"
        if mode == 'img':
            wm = cv2.imread(filename=wm_content, flags=cv2.IMREAD_GRAYSCALE)
            assert wm is not None, 'file "{filename}" not read'.format(filename=wm_content)

            # 读入图片格式的水印，并转为一维 bit 格式，抛弃灰度级别
            self.wm_bit = wm.flatten() > 128

        elif mode == 'str':
            byte = bin(int(wm_content.encode('utf-8').hex(), base=16))[2:]
            self.wm_bit = (np.array(list(byte)) == '1')
        else:
            self.wm_bit = np.array(wm_content)

        self.wm_size = self.wm_bit.size

        # 水印加密：
        np.random.RandomState(self.password_wm).shuffle(self.wm_bit)

        self.bwm_core.read_wm(self.wm_bit)

    def embed(self, filename=None, compression_ratio=None):
        '''
        :param filename: string
            Save the image file as filename
        :param compression_ratio: int or None
            If compression_ratio = None, do not compression,
            If compression_ratio is integer between 0 and 100, the smaller, the output file is smaller.
        :return:
        '''
        embed_img = self.bwm_core.embed()
        if filename is not None:
            if compression_ratio is None:
                cv2.imwrite(filename=filename, img=embed_img)
            elif filename.endswith('.jpg'):
                cv2.imwrite(filename=filename, img=embed_img, params=[cv2.IMWRITE_JPEG_QUALITY, compression_ratio])
            elif filename.endswith('.png'):
                cv2.imwrite(filename=filename, img=embed_img, params=[cv2.IMWRITE_PNG_COMPRESSION, compression_ratio])
            else:
                cv2.imwrite(filename=filename, img=embed_img)
        return embed_img

    def extract_decrypt(self, wm_avg):
        wm_index = np.arange(self.wm_size)
        np.random.RandomState(self.password_wm).shuffle(wm_index)
        wm_avg[wm_index] = wm_avg.copy()
        return wm_avg

    def extract(self, filename=None, embed_img=None, wm_shape=None, out_wm_name=None, mode='img'):
        assert wm_shape is not None, 'wm_shape needed'

        if filename is not None:
            embed_img = cv2.imread(filename, flags=cv2.IMREAD_COLOR)
            assert embed_img is not None, "{filename} not read".format(filename=filename)

        self.wm_size = np.array(wm_shape).prod()

        if mode in ('str', 'bit'):
            wm_avg = self.bwm_core.extract_with_kmeans(img=embed_img,
                wm_shape=wm_shape)
        else:
            wm_avg = self.bwm_core.extract(img=embed_img, wm_shape=wm_shape)

        # 解密：
        wm = self.extract_decrypt(wm_avg=wm_avg)

    # 转化为指定格式（修改了此处处理逻辑）：
        if mode == 'img':
            wm = 255 * wm.reshape(wm_shape[0], wm_shape[1])
            cv2.imwrite(out_wm_name, wm)
        elif mode == 'str':
            bits = [int(i >= 0.5) for i in wm]
            usable_len = (len(bits) // 8) * 8
            bits = bits[:usable_len]
            bytes_list = [int(''.join(map(str, bits[i:i + 8])), 2) for i in range(0, usable_len, 8)]
            wm = bytes(bytes_list).decode('utf-8', errors='replace')

        return wm

#blind_core.py
#!/usr/bin/env python3
# coding=utf-8
# @Time    : 2021/12/17
# @Author  : github.com/guofei9987
import numpy as np
from numpy.linalg import svd
import copy
import cv2
from cv2 import dct, idct
from pywt import dwt2, idwt2
from pool import AutoPool


class WaterMarkCore:
    def __init__(self, password_img=1, mode='common', processes=None):
        self.block_shape = np.array([4, 4])
        self.password_img = password_img
        self.d1, self.d2 = 36, 20  # d1/d2 越大鲁棒性越强,但输出图片的失真越大

        # init data
        self.img, self.img_YUV = None, None  # self.img 是原图，self.img_YUV 对像素做了加白偶数化
        self.ca, self.hvd, = [np.array([])] * 3, [np.array([])] * 3  # 每个通道 dct 的结果
        self.ca_block = [np.array([])] * 3  # 每个 channel 存一个四维 array，代表四维分块后的结果
        self.ca_part = [np.array([])] * 3  # 四维分块后，有时因不整除而少一部分，self.ca_part 是少这一部分的 self.ca

        self.wm_size, self.block_num = 0, 0  # 水印的长度，原图片可插入信息的个数
        self.pool = AutoPool(mode=mode, processes=processes)

        self.fast_mode = False
        self.alpha = None  # 用于处理透明图

    def init_block_index(self):
        self.block_num = self.ca_block_shape[0] * self.ca_block_shape[1]
        assert self.wm_size < self.block_num, IndexError(
            '最多可嵌入{}kb信息，多于水印的{}kb信息，溢出'.format(self.block_num / 1000, self.wm_size / 1000))
        # self.part_shape 是取整后的ca二维大小,用于嵌入时忽略右边和下面对不齐的细条部分。
        self.part_shape = self.ca_block_shape[:2] * self.block_shape
        self.block_index = [(i, j) for i in range(self.ca_block_shape[0]) for j in range(self.ca_block_shape[1])]

    def read_img_arr(self, img):
        # 处理透明图
        self.alpha = None
        if img.shape[2] == 4:
            if img[:, :, 3].min() < 255:
                self.alpha = img[:, :, 3]
                img = img[:, :, :3]

        # 读入图片->YUV化->加白边使像素变偶数->四维分块
        self.img = img.astype(np.float32)
        self.img_shape = self.img.shape[:2]

        # 如果不是偶数，那么补上白边，Y（明亮度）UV（颜色）
        self.img_YUV = cv2.copyMakeBorder(cv2.cvtColor(self.img, cv2.COLOR_BGR2YUV),
                                          0, self.img.shape[0] % 2, 0, self.
```

```
                                              img.shape[1] % 2,
                                          cv2.BORDER_CONSTANT, value=(0, 0, 0))

        self.ca_shape = [(i + 1) // 2 for i in self.img_shape]

        self.ca_block_shape = (self.ca_shape[0] // self.block_shape[0], self.ca_shape[1] // self.block_shape[1],
                               self.block_shape[0], self.block_shape[1])
        strides = 4 * np.array([self.ca_shape[1] * self.block_shape[0], self.block_shape[1], self.ca_shape[1], 1])

        for channel in range(3):
            self.ca[channel], self.hvd[channel] = dwt2(self.img_YUV[:, :, channel], 'haar')
            # 转为4维度
            self.ca_block[channel] = np.lib.stride_tricks.as_strided(self.ca[channel].astype(np.float32),
                                                                     self.ca_block_shape, strides)

    def read_wm(self, wm_bit):
        self.wm_bit = wm_bit
        self.wm_size = wm_bit.size

    def block_add_wm(self, arg):
        if self.fast_mode:
            return self.block_add_wm_fast(arg)
        else:
            return self.block_add_wm_slow(arg)

    def block_add_wm_slow(self, arg):
        block, shuffler, i = arg
        # dct->(flatten->加密->逆flatten)->svd->打水印->逆svd->(flatten->解密->逆flatten)->逆dct
        wm_1 = self.wm_bit[i % self.wm_size]
        block_dct = dct(block)

        # 加密（打乱顺序）
        block_dct_shuffled = block_dct.flatten()[shuffler].reshape(self.
            block_shape)
        u, s, v = svd(block_dct_shuffled)
        s[0] = (s[0] // self.d1 + 1 / 4 + 1 / 2 * wm_1) * self.d1
        if self.d2:
            s[1] = (s[1] // self.d2 + 1 / 4 + 1 / 2 * wm_1) * self.d2

        block_dct_flatten = np.dot(u, np.dot(np.diag(s), v)).flatten()
        block_dct_flatten[shuffler] = block_dct_flatten.copy()
        return idct(block_dct_flatten.reshape(self.block_shape))

    def block_add_wm_fast(self, arg):
        # dct->svd->打水印->逆svd->逆dct
        block, shuffler, i = arg
        wm_1 = self.wm_bit[i % self.wm_size]

        u, s, v = svd(dct(block))
        s[0] = (s[0] // self.d1 + 1 / 4 + 1 / 2 * wm_1) * self.d1

        return idct(np.dot(u, np.dot(np.diag(s), v)))

    def embed(self):
        self.init_block_index()

        embed_ca = copy.deepcopy(self.ca)
        embed_YUV = [np.array([])] * 3

        self.idx_shuffle = random_strategy1(self.password_img, self.block_num,
                                            self.block_shape[0] * self.block_shape[1])
        for channel in range(3):
            tmp = self.pool.map(self.block_add_wm,
                                [(self.ca_block[channel][self.block_index[i]], self.idx_shuffle[i], i)
                                 for i in range(self.block_num)])

            for i in range(self.block_num):
                self.ca_block[channel][self.block_index[i]] = tmp[i]

            # 4维分块变回2维
            self.ca_part[channel] = np.concatenate(np.concatenate(self.ca_block[channel], 1), 1)
            # 4维分块时右边和下边不能整除的长条保留，其余是主体部分，换成
```

```
                embed 之后的频域的数据
            embed_ca[channel][:self.part_shape[0], :self.part_shape[1]] = self.ca_part[channel]
            # 逆变换回去
            embed_YUV[channel] = idwt2((embed_ca[channel], self.hvd[channel]), "haar")

        # 合并3通道
        embed_img_YUV = np.stack(embed_YUV, axis=2)
        # 之前如果不是2的整数，增加了白边，这里去除掉
        embed_img_YUV = embed_img_YUV[:self.img_shape[0], :self.img_shape[1]]
        embed_img = cv2.cvtColor(embed_img_YUV, cv2.COLOR_YUV2BGR)
        embed_img = np.clip(embed_img, a_min=0, a_max=255)

        if self.alpha is not None:
            embed_img = cv2.merge([embed_img.astype(np.uint8), self.alpha])
        return embed_img

    def block_get_wm(self, args):
        if self.fast_mode:
            return self.block_get_wm_fast(args)
        else:
            return self.block_get_wm_slow(args)

    def block_get_wm_slow(self, args):
        block, shuffler = args
        # dct->flatten->加密->逆flatten->svd->解水印
        block_dct_shuffled = dct(block).flatten()[shuffler].reshape(self.block_shape)

        u, s, v = svd(block_dct_shuffled)
        wm = (s[0] % self.d1 > self.d1 / 2) * 1
        if self.d2:
            tmp = (s[1] % self.d2 > self.d2 / 2) * 1
            wm = (wm * 3 + tmp * 1) / 4
        return wm

    def block_get_wm_fast(self, args):
        block, shuffler = args
        # dct->svd->解水印
        u, s, v = svd(dct(block))
        wm = (s[0] % self.d1 > self.d1 / 2) * 1

        return wm

    def extract_raw(self, img):
        # 每个分块提取 1 bit 信息
        self.read_img_arr(img=img)
        self.init_block_index()

        wm_block_bit = np.zeros(shape=(3, self.block_num))  # 3个channel, length 个分块提取的水印，全都记录下来

        self.idx_shuffle = random_strategy1(seed=self.password_img,
                                            size=self.block_num,
                                            block_shape=self.block_shape[0] * self.block_shape[1],  # 16
                                            )
        for channel in range(3):
            wm_block_bit[channel, :] = self.pool.map(self.block_get_wm,
                                                     [(self.ca_block[channel][self.block_index[i]], self.idx_shuffle[i])
                                                      for i in range(self.block_num)])
        return wm_block_bit

    def extract_avg(self, wm_block_bit):
        # 对循环嵌入+3个 channel 求平均
        wm_avg = np.zeros(shape=self.wm_size)
        for i in range(self.wm_size):
            wm_avg[i] = wm_block_bit[:, i::self.wm_size].mean()
        return wm_avg

    def extract(self, img, wm_shape):
        self.wm_size = np.array(wm_shape).prod()

        # 提取每个分块埋入的 bit：
        wm_block_bit = self.extract_raw(img=img)
        # 做平均：
        wm_avg = self.extract_avg(wm_block_bit)
        return wm_avg

    def extract_with_kmeans(self, img, wm_shape):
        wm_avg = self.extract(img=img, wm_shape=wm_shape)

        return one_dim_kmeans(wm_avg)


def one_dim_kmeans(inputs):
    threshold = 0
    e_tol = 10 ** (-6)
    center = [inputs.min(), inputs.max()]  # 1. 初始化中心点
    for i in range(300):
        threshold = (center[0] + center[1]) / 2
        is_class01 = inputs > threshold  # 2. 检查所有点与这k个点之间的距离，每个点归类到最近的中心
        center = [inputs[~is_class01].mean(), inputs[is_class01].mean()]  # 3. 重新找中心点
        if np.abs((center[0] + center[1]) / 2 - threshold) < e_tol:  # 4. 停止条件
            threshold = (center[0] + center[1]) / 2
            break

    is_class01 = inputs > threshold
    return is_class01


def random_strategy1(seed, size, block_shape):
    return np.random.RandomState(seed) \
        .random(size=(size, block_shape)) \
        .argsort(axis=1)


def random_strategy2(seed, size, block_shape):
    one_line = np.random.RandomState(seed) \
        .random(size=(1, block_shape)) \
        .argsort(axis=1)

    return np.repeat(one_line, repeats=size, axis=0)


#pool.py
import sys
import multiprocessing
import warnings

```

```
if sys.platform != 'win32':
    multiprocessing.set_start_method('fork')


class CommonPool(object):
    def map(self, func, args):
        return list(map(func, args))


class AutoPool(object):
    def __init__(self, mode, processes):

        if mode == 'multiprocessing' and sys.platform == 'win32':
            warnings.warn('multiprocessing not support in windows, turning to multithreading')
            mode = 'multithreading'

        self.mode = mode
        self.processes = processes

        if mode == 'vectorization':
            pass
        elif mode == 'cached':
            pass
        elif mode == 'multithreading':
            from multiprocessing.dummy import Pool as ThreadPool
            self.pool = ThreadPool(processes=processes)
        elif mode == 'multiprocessing':
            from multiprocessing import Pool
            self.pool = Pool(processes=processes)
        else:  # common
            self.pool = CommonPool()

    def map(self, func, args):
        return self.pool.map(func, args)

#main.py
import os
import cv2
from blind_watermark import WaterMark

# ========== 配置 ==========
# 原始图像路径（用于嵌入水印）
```

```
img_path = 'test.jpg'

# 水印内容，可以是字符串，也可以是图片路径
watermark = '这是一段测试水印'  # 或 'logo.png'
#watermark = 'watermark.png'  # 或 'logo.png'
# 输出路径
output_embed_path = 'test_embed.jpg'
output_extracted_path = 'test_extracted.png'

# 密码
password_wm = 1
password_img = 1

#判断水印类型
if isinstance(watermark, str) and os.path.isfile(watermark):
    wm_mode = 'img'
elif isinstance(watermark, str):
    wm_mode = 'str'
else:
    raise ValueError('无法识别的水印类型')

#嵌入
bwm = WaterMark(password_wm=password_wm, password_img=password_img)
bwm.read_img(img_path)

if wm_mode == 'str':
    bwm.read_wm(watermark, mode='str')
    wm_shape = (1, len(bwm.wm_bit))  #一维字符串的比特长度
elif wm_mode == 'img':
    wm_img = cv2.imread(watermark, cv2.IMREAD_GRAYSCALE)
    assert wm_img is not None, f"无法读取水印图像: {watermark}"
    bwm.read_wm(watermark, mode='img')
    wm_shape = wm_img.shape  # 获取图像尺寸

bwm.embed(output_embed_path)
print(f"成功嵌入水印，保存为: {output_embed_path}")

#提取
bwm_extract = WaterMark(password_wm=password_wm, password_img=password_img)
wm_result = bwm_extract.extract(
    filename=output_embed_path,
    wm_shape=wm_shape,
    out_wm_name=output_extracted_path if wm_mode == 'img' else None,
    mode=wm_mode
)

if wm_mode == 'str':
    print(f"成功提取字符串水印：{wm_result}")
elif wm_mode == 'img':
    print(f"成功提取图片水印，保存为：{output_extracted_path}")
```

上述代码中前三个都是开源库中的代码，我对其进行了一些图片处理上的调整，具体而言，blind_watermark.py 定义了一个 WaterMark 类，用于：读取原图和水印；加入水印和提取水印；并且支持水印为图片、字符串或 bit 序列。blind_core.py 是系统的核心算法层，定义了 WaterMarkCore 类，封装了嵌入和提取的核心过程，嵌入水印时，大致步骤如下所示：

原图 → 转为 YUV 色彩空间 → 对 Y 通道（亮度）进行小波变换（pywt.dwt2）。

对低频部分（CA）做分块（4x4）→ 每块 DCT 变换 → SVD 分解。

利用特征值（奇异值）嵌入水印 bit（s[0]、s[1]）→ 逆过程生成嵌入后的图像。

最后恢复 YUV → RGB。

提取水印时的大致步骤如下所示：读入嵌入图像 → 转 YUV → 小波变换 → 与嵌入时一样分块。

每块 → DCT → SVD → 从特征值中反推出水印 bit（通过奇异值范围判断）。

最终的水印是三个通道的平均值（或 KMeans 聚类）结果。

pool.py 自动根据操作系统和模式选择 “串行”、“多线程”、“多进程” 或 “向量化” 等不同方式，来加速处理任务列表，用于并行执行如图像分块水印嵌入等操作。mian.py 则是我用来进行测试的代码，能够自动识别输入水印的类型是图片还是字符串，并且使用变量记录下来水印的 shape 方便后续提取。

2.2 图片水印和提取结果

我们测试了字符和图片类型两种水印，得到的结果肉眼都无法区分，具体添加盲水印后的结果如图所示，左边是原图，右边是添加后的图片：

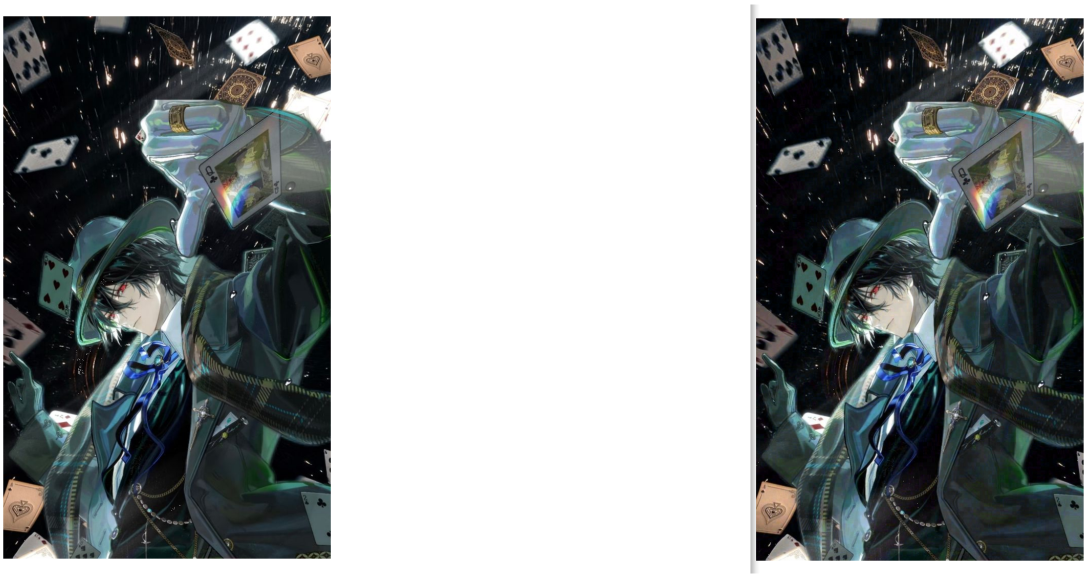
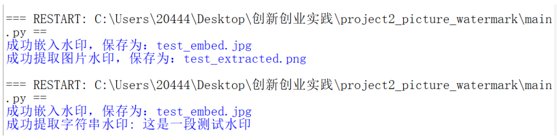


图 1 添加水印

我们可以看到，由于是盲水印，所以在右边的图片中没能看到我们添加的水印，无论水印是图片类型还是字符串类型的。



```
=== RESTART: C:\Users\20444\Desktop\创新创业实践\project2_picture_watermark\main
.py ==
成功嵌入水印，保存为：test_embed.jpg
成功提取图片水印，保存为：test_extracted.png

=== RESTART: C:\Users\20444\Desktop\创新创业实践\project2_picture_watermark\main
.py ==
成功嵌入水印，保存为：test_embed.jpg
成功提取字符串水印：这是一段测试水印
```

图 2 结果

上图是我们测试了字符和图片的运行结果，字符串类型水印的提取结果已经显示在上图中，我们看一下图片型的输出：

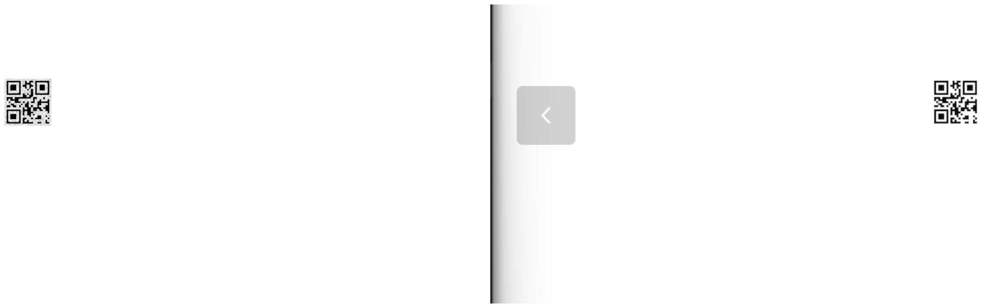


图 3 提取图片型水印

我们可以从上图中看到，能够成功提取出来该图形中的水印。

3　鲁棒性测试

对图片水印系统进行鲁棒性测试，就是要验证水印在各种图像攻击（如翻转、平移、裁剪、压缩、亮度调整等）下是否还能成功提取出来。我们对已经打好水印的图片进行测试，

3.1　攻击方法

该开源项目中 att.py 也已经有了一些攻击方法，具体代码见文章末尾，于是我们也使用 att.py 中的一些攻击方法，具体如下表所示：

序号	名称	具体操作
1	flip	水平翻转攻击，使用 cv2.flip(...), 1，模拟图像左右对调后对水印的影响。
2	shift	图像平移攻击，使用 np.roll(...), axis=1 实现水平滚动（像素级偏移）。
3	crop	裁剪攻击，剪掉图像的部分区域（使用坐标比例），再提取剩余部分。
4	contrast	调节亮度攻击，提升对比度（乘以 1.5）模拟高曝光或调亮对图像的影响。
5	blur	模糊攻击，使用高斯模糊 cv2.GaussianBlur() 模拟图像失焦。
6	jpeg	JPEG 压缩攻击，降低图像质量（压缩质量设为 25）以测试水印在有损压缩下的鲁棒性。
7	resize	缩放攻击，将图像缩小为 300×300 再提取，模拟分辨率改变的场景。
8	shelter	遮挡攻击，在图像中随机添加白色遮挡块（模拟物理遮挡或故障遮挡）。
9	salt_pepper	椒盐噪声攻击，在图像中随机像素点添加噪声（类似拍摄环境下的干扰噪声）。
10	rotate	旋转攻击，将图像绕中心点旋转 30 度，模拟图像旋转后水印是否仍然可提取。

表 2 图像攻击类型及描述

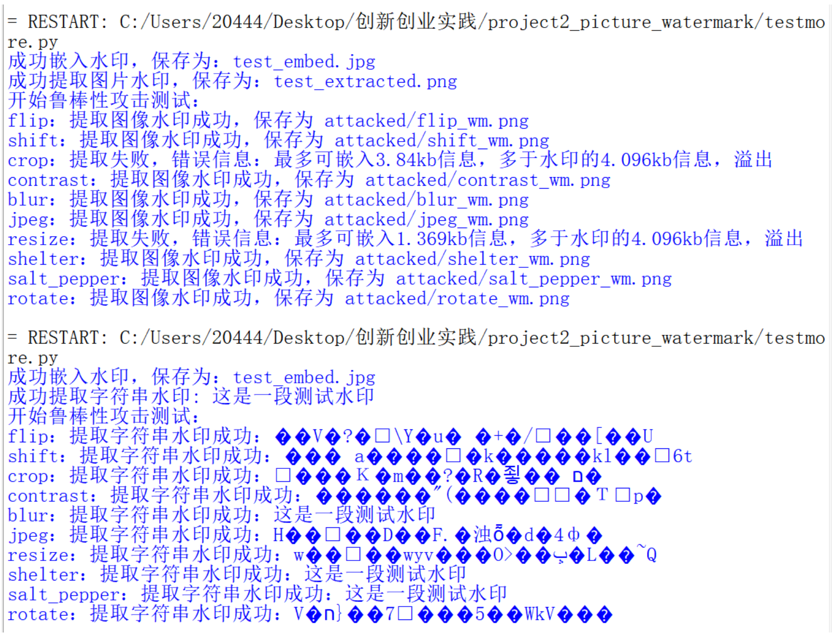
我们编程实现的共计代码如下所示：

```
#main.py
import os
import cv2
from blind_watermark import WaterMark
import att
import numpy as np

# ========== 配 置 ==========
# 原始图像路径（用于嵌入水印）
img_path = 'test.jpg'

# 水印内容，可以是字符串，也可以是图片路径
watermark = '这是一段测试水印' # 或 'logo.png'
#watermark = 'watermark.png' # 或 'logo.png'
# 输出路径
output_embed_path = 'test_embed.jpg'
output_extracted_path = 'test_extracted.png'

# 密码
password_wm = 1
password_img = 1

#判断水印类型
if isinstance(watermark, str) and os.path.isfile(watermark):
    wm_mode = 'img'
elif isinstance(watermark, str):
    wm_mode = 'str'
else:
    raise ValueError('无法识别的水印类型')

#嵌入
bwm = WaterMark(password_wm=password_wm, password_img=password_img)
bwm.read_img(img_path)

if wm_mode == 'str':
    bwm.read_wm(watermark, mode='str')
    wm_shape = (1, len(bwm.wm_bit)) #一维字符串的比特长度
elif wm_mode == 'img':
    wm_img = cv2.imread(watermark, cv2.IMREAD_GRAYSCALE)
    assert wm_img is not None, f"无法读取水印图像：{watermark}"
    bwm.read_wm(watermark, mode='img')
    wm_shape = wm_img.shape # 获取图像尺寸

bwm.embed(output_embed_path)
print(f"成功嵌入水印，保存为：{output_embed_path}")

#提取
bwm_extract = WaterMark(password_wm=password_wm, password_img=password_img)
wm_result = bwm_extract.extract(
    filename=output_embed_path,
    wm_shape=wm_shape,
```

```python
    out_wm_name=output_extracted_path if wm_mode == 'img' else None,
    mode=wm_mode
)

if wm_mode == 'str':
    print(f"成功提取字符串水印：{wm_result}")
elif wm_mode == 'img':
    print(f"成功提取图片水印，保存为：{output_extracted_path}")

# 创建一个攻击输出目录
os.makedirs('attacked', exist_ok=True)

# 攻击测试列表（函数名、参数、输出名）
attack_list = [
    ('flip', lambda: cv2.flip(cv2.imread(output_embed_path), 1), 'attacked/flip.jpg'),
    ('shift', lambda: np.roll(cv2.imread(output_embed_path), 10, axis=1), 'attacked/shift.jpg'),
    ('crop', lambda: att.cut_att3(input_filename=output_embed_path, loc_r=((0.2, 0.2), (0.8, 0.8))), 'attacked/crop.jpg'),
    ('contrast', lambda: att.bright_att(input_filename=output_embed_path, ratio=1.5), 'attacked/contrast.jpg'),
    ('blur', lambda: cv2.GaussianBlur(cv2.imread(output_embed_path), (5, 5), 0), 'attacked/blur.jpg'),
    ('jpeg', lambda: cv2.imdecode(cv2.imencode('.jpg', cv2.imread(output_embed_path), [int(cv2.IMWRITE_JPEG_QUALITY), 25])[1], 1), 'attacked/jpeg.jpg'),
    ('resize', lambda: att.resize_att(input_filename=output_embed_path, out_shape=(300, 300)), 'attacked/resize.jpg'),
    ('shelter', lambda: att.shelter_att(input_filename=output_embed_path, ratio=0.2, n=2), 'attacked/shelter.jpg'),
    ('salt_pepper', lambda: att.salt_pepper_att(input_filename=output_embed_path, ratio=0.01), 'attacked/salt.jpg'),
    ('rotate', lambda: att.rot_att(input_filename=output_embed_path, angle=30), 'attacked/rotate.jpg'),
]

print("开始鲁棒性攻击测试：")
for name, attack_func, attacked_path in attack_list:
    try:
        attacked_img = attack_func()
        cv2.imwrite(attacked_path, attacked_img)

        # 提取攻击后的水印
        bwm_attacked = WaterMark(password_wm=password_wm, password_img=
            password_img)
        wm_att = bwm_attacked.extract(
            filename=attacked_path,
            wm_shape=wm_shape,
            out_wm_name=None if wm_mode == 'str' else f'attacked/{name}_wm.
                png',
            mode=wm_mode
        )

        if wm_mode == 'str':
            print(f"{name}: 提取字符串水印成功: {wm_att}")
        else:
            print(f"{name}: 提取图像水印成功, 保存为 attacked/{name}_wm.png
                ")

    except Exception as e:
        print(f"{name}: 提取失败, 错误信息: {e}")
```

3.2 测试结果

运行结果如下所示：



```
= RESTART: C:/Users/20444/Desktop/创新创业实践/project2_picture_watermark/testmo
re.py
成功嵌入水印，保存为：test_embed.jpg
成功提取图片水印，保存为：test_extracted.png
开始鲁棒性攻击测试：
flip：提取图像水印成功，保存为 attacked/flip_wm.png
shift：提取图像水印成功，保存为 attacked/shift_wm.png
crop：提取失败，错误信息：最多可嵌入3.84kb信息，多于水印的4.096kb信息，溢出
contrast：提取图像水印成功，保存为 attacked/contrast_wm.png
blur：提取图像水印成功，保存为 attacked/blur_wm.png
jpeg：提取图像水印成功，保存为 attacked/jpeg_wm.png
resize：提取失败，错误信息：最多可嵌入1.369kb信息，多于水印的4.096kb信息，溢出
shelter：提取图像水印成功，保存为 attacked/shelter_wm.png
salt_pepper：提取图像水印成功，保存为 attacked/salt_pepper_wm.png
rotate：提取图像水印成功，保存为 attacked/rotate_wm.png

= RESTART: C:/Users/20444/Desktop/创新创业实践/project2_picture_watermark/testmo
re.py
成功嵌入水印，保存为：test_embed.jpg
成功提取字符串水印：这是一段测试水印
开始鲁棒性攻击测试：
flip：提取字符串水印成功：��V�?�□\Y�u� �+�/□��[��U
shift：提取字符串水印成功：��� a����□�k�����kl��□6t
crop：提取字符串水印成功：□���K�m��?�R�좋�� ם�
contrast：提取字符串水印成功：������"(����□□�T□p�
blur：提取字符串水印成功：这是一段测试水印
jpeg：提取字符串水印成功：H��□��D��F.�浊ỗ�d�4ф�
resize：提取字符串水印成功：w��□��wyv���0>��ڀ�L��˜Q
shelter：提取字符串水印成功：这是一段测试水印
salt_pepper：提取字符串水印成功：这是一段测试水印
rotate：提取字符串水印成功：V�n}��7□���5��WkV���
```

图 4 鲁棒性测试

可以看到对字符串，能够抵抗 blur，shelter，rotate 的攻击，图片我们看一下

提取出来的水印图片：

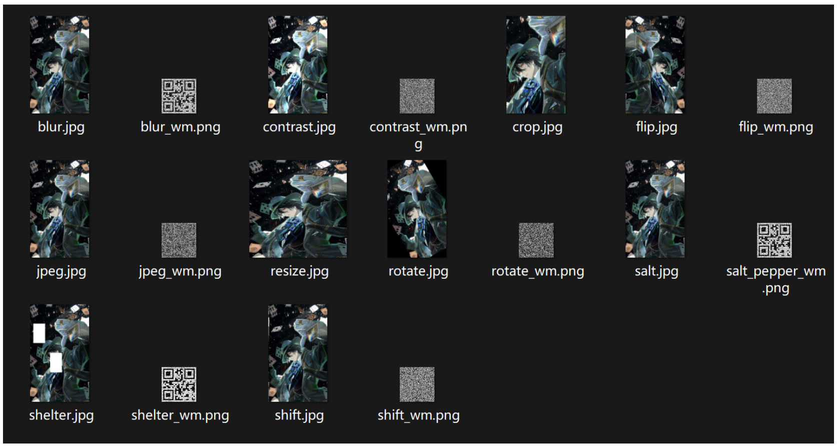


图 5 鲁棒性测试

可以看出能够抵抗 blur，salt，shelter 攻击。总体而言，图片水印和字符串水印的鲁棒性并不完全相同，不过综合来说抵抗 blur 和 shelter 攻击的能力较强。

3.3 攻击方式代码

```
# coding=utf-8

# attack on the watermark
import cv2
import numpy as np
import warnings


def cut_att3(input_filename=None, input_img=None, output_file_name=None, loc_r=None, loc=None, scale=None):
    # 剪切攻击 + 缩放攻击
    if input_filename:
        input_img = cv2.imread(input_filename)

    if loc is None:
        h, w, _ = input_img.shape
        x1, y1, x2, y2 = int(w * loc_r[0][0]), int(h * loc_r[0][1]), int(w * loc_r[1][0]), int(h * loc_r[1][1])
    else:
        x1, y1, x2, y2 = loc

    # 剪切攻击
    output_img = input_img[y1:y2, x1:x2].copy()

    # 如果缩放攻击
    if scale and scale != 1:
        h, w, _ = output_img.shape
        output_img = cv2.resize(output_img, dsize=(round(w * scale), round(h * scale)))
    else:
        output_img = output_img

    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


cut_att2 = cut_att3


def resize_att(input_filename=None, input_img=None, output_file_name=None, out_shape=(500, 500)):
    # 缩放攻击：因为攻击和还原都是缩放，所以攻击和还原都调用这个函数
    if input_filename:
        input_img = cv2.imread(input_filename)
    output_img = cv2.resize(input_img, dsize=out_shape)
    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


def bright_att(input_filename=None, input_img=None, output_file_name=None, ratio=0.8):
    # 亮度调整攻击，ratio应当多于0
    # ratio>1是调得更亮，ratio<1是亮度更暗
    if input_filename:
        input_img = cv2.imread(input_filename)
    output_img = input_img * ratio
    output_img[output_img > 255] = 255
    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


def shelter_att(input_filename=None, input_img=None, output_file_name=None, ratio=0.1, n=3):
    # 遮挡攻击：遮挡图像中的一部分
    # n个遮挡块
    # 每个遮挡块所占比例为ratio
    if input_filename:
        output_img = cv2.imread(input_filename)
    else:
        output_img = input_img.copy()
    input_img_shape = output_img.shape

    for i in range(n):
        tmp = np.random.rand() * (1 - ratio)  # 随机选择一个地方，1-ratio是为了防止溢出
        start_height, end_height = int(tmp * input_img_shape[0]), int((tmp + ratio) * input_img_shape[0])
        tmp = np.random.rand() * (1 - ratio)
        start_width, end_width = int(tmp * input_img_shape[1]), int((tmp + ratio) * input_img_shape[1])

        output_img[start_height:end_height, start_width:end_width, :] = 255

    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


def salt_pepper_att(input_filename=None, input_img=None, output_file_name=None, ratio=0.01):
    # 椒盐攻击
    if input_filename:
        input_img = cv2.imread(input_filename)
    input_img_shape = input_img.shape
    output_img = input_img.copy()
    for i in range(input_img_shape[0]):
        for j in range(input_img_shape[1]):
            if np.random.rand() < ratio:
                output_img[i, j, :] = 255
    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


def rot_att(input_filename=None, input_img=None, output_file_name=None, angle=45):
    # 旋转攻击
    if input_filename:
        input_img = cv2.imread(input_filename)
    rows, cols, _ = input_img.shape
    M = cv2.getRotationMatrix2D(center=(cols / 2, rows / 2), angle=angle, scale=1)
    output_img = cv2.warpAffine(input_img, M, (cols, rows))
    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


def cut_att_height(input_filename=None, input_img=None, output_file_name=None, ratio=0.8):
    warnings.warn('will be deprecated in the future, use att.cut_att2 instead')
    # 纵向剪切攻击
    if input_filename:
        input_img = cv2.imread(input_filename)
    input_img_shape = input_img.shape
    height = int(input_img_shape[0] * ratio)

    output_img = input_img[:height, :, :]
    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img


def cut_att_width(input_filename=None, input_img=None, output_file_name=None, ratio=0.8):
    warnings.warn('will be deprecated in the future, use att.cut_att2 instead')
    # 横向裁剪攻击
    if input_filename:
        input_img = cv2.imread(input_filename)
    input_img_shape = input_img.shape
    width = int(input_img_shape[1] * ratio)

    output_img = input_img[:, :width, :]
    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img

```

```

def cut_att(input_filename=None, output_file_name=None, input_img=None, loc=((0.3, 0.1), (0.7, 0.9)), resize=0.6):
    warnings.warn('will be deprecated in the future, use att.cut_att2 instead')
    # 截屏攻击 = 裁剪攻击 + 缩放攻击 + 知道攻击参数 (按照参数还原)
    # 裁剪攻击：其它部分都补0
    if input_filename:
        input_img = cv2.imread(input_filename)

    output_img = input_img.copy()
    shape = output_img.shape
    x1, y1, x2, y2 = shape[0] * loc[0][0], shape[1] * loc[0][1], shape[0] * loc[1][0], shape[1] * loc[1][1]
    output_img[:int(x1), :] = 255
    output_img[int(x2):, :] = 255
    output_img[:, :int(y1)] = 255
    output_img[:, int(y2):] = 255

    if resize is not None:
        # 缩放一次，然后还原
        output_img = cv2.resize(output_img,
                                dsize=(int(shape[1] * resize), int(shape[0] * resize))
                                )

        output_img = cv2.resize(output_img, dsize=(int(shape[1]), int(shape[0])))

    if output_file_name is not None:
        cv2.imwrite(output_file_name, output_img)
    return output_img


# def cut_att2(input_filename=None, input_img=None, output_file_name=None, loc_r=((0.3, 0.1), (0.9, 0.9)), scale=1.1):
#     # 截屏攻击 = 剪切攻击 + 缩放攻击 + 不知道攻击参数
#     if input_filename:
#         input_img = cv2.imread(input_filename)
#     h, w, _ = input_img.shape
#     x1, y1, x2, y2 = int(w * loc_r[0][0]), int(h * loc_r[0][1]), int(w * loc_r[1][0]), int(h * loc_r[1][1])
#
```

```
#     output_img = cut_att3(input_img=input_img, output_file_name=output_file_name,
#                           loc=(x1, y1, x2, y2), scale=scale)
#     return output_img, (x1, y1, x2, y2)

def anti_cut_att_old(input_filename, output_file_name, origin_shape):
    warnings.warn('will be deprecated in the future')
    # 反裁剪攻击：复制一块范围，然后补全
    # origin_shape 分辨率与约定理解的是颠倒的，约定的是列数*行数
    input_img = cv2.imread(input_filename)
    output_img = input_img.copy()
    output_img_shape = output_img.shape
    if output_img_shape[0] > origin_shape[0] or output_img_shape[1] > origin_shape[1]:
        print('裁剪打击后的图片，不可能比原始图片大，检查一下')
        return

    # 还原纵向打击
    while output_img_shape[0] < origin_shape[0]:
        output_img = np.concatenate([output_img, output_img[:origin_shape[0] - output_img_shape[0], :, :]], axis=0)
        output_img_shape = output_img.shape
    while output_img_shape[1] < origin_shape[1]:
        output_img = np.concatenate([output_img, output_img[:, :origin_shape[1] - output_img_shape[1], :]], axis=1)
        output_img_shape = output_img.shape

    cv2.imwrite(output_file_name, output_img)


def anti_cut_att(input_filename=None, input_img=None, output_file_name=None, origin_shape=None):
    warnings.warn('will be deprecated in the future, use att.cut_att2 instead')
    # 反裁剪攻击：补0
    # origin_shape 分辨率与约定理解的是颠倒的，约定的是列数*行数
    if input_filename:
        input_img = cv2.imread(input_filename)
    output_img = input_img.copy()
    output_img_shape = output_img.shape
    if output_img_shape[0] > origin_shape[0] or output_img_shape[1] > origin_shape[1]:
        print('裁剪打击后的图片，不可能比原始图片大，检查一下')
        return

    # 还原纵向打击
    if output_img_shape[0] < origin_shape[0]:
        output_img = np.concatenate(
            [output_img, 255 * np.ones((origin_shape[0] - output_img_shape[0], output_img_shape[1], 3))]
            , axis=0)
        output_img_shape = output_img.shape

    if output_img_shape[1] < origin_shape[1]:
        output_img = np.concatenate(
            [output_img, 255 * np.ones((output_img_shape[0], origin_shape[1] - output_img_shape[1], 3))]
            , axis=1)

    if output_file_name:
        cv2.imwrite(output_file_name, output_img)
    return output_img
```