山东大学

SHANDONG UNIVERSITY

Project 2: 数字水印

姓　　名: 谢星豪

学　　号: 202100161170

专　　业: 网络空间安全

班　　级: 网安一班

2025 年 8 月 6 日

目录

1 实验内容

Project 2: 基于数字水印的图片泄露检测

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 实验原理

基于离散小波变换（DWT）和离散余弦变换（DCT）的鲁棒水印算法是一种在变换域中嵌入信息的经典方法。该算法结合了 DWT 的多分辨率分析特性和 DCT 的能量集中特性，旨在实现水印的不可见性和对各种图像处理攻击的鲁棒性。

- **DWT (Discrete Wavelet Transform)**: DWT 将图像分解为不同分辨率和方向的子带。一次 DWT 分解会将图像分为四个子带：LL（低频近似分量），LH（水平细节分量），HL（垂直细节分量）和 HH（对角细节分量）。LL 子带包含了图像的大部分能量和主要视觉信息，而 LH、HL、HH 子带则包含了图像的边缘和纹理等细节信息。将水印嵌入到中频子带（如 LH 或 HL）可以在鲁棒性和不可见性之间取得很好的平衡。
- **DCT (Discrete Cosine Transform)**: DCT 能够将图像块的能量集中到左上角的少数几个低频系数中。这种能量集中的特性使得在 DCT 域中修改系数对图像视觉质量的影响更容易控制，同时也使得水印信息能够被嵌入到对信号失真（如 JPEG 压缩）不敏感的重要系数中。

算法的核心思想是：首先对原始图像进行 DWT 分解，然后选择一个或多个中频子带，将这些子带分块并进行 DCT 变换，最后根据水印信息修改选定的 DCT 系数。

2.1 水印嵌入过程

设原始载体图像为 I，水印信息为二值序列 $W = \{w_1, w_2, \ldots, w_N\}$，其中 $w_i \in \{0, 1\}$ 或 $\{-1, 1\}$。

1. **DWT 分解**: 对原始图像 I 进行多级 DWT 分解。例如，进行一级 DWT，得到四个子带 LL_1, LH_1, HL_1, HH_1。为了在鲁棒性和不可见性之间取得平衡，通常选择中频子带（如 LH_1 或 HL_1）作为嵌入区域。
2. **分块与 DCT 变换**: 将选定的子带（例如 HL_1）分割成多个不重叠的 $n \times n$ (例如 8×8) 的小块 B_k。对每个块 B_k 进行 DCT 变换，得到 DCT 系数矩阵 C_k。

$$C_k = \mathrm{DCT}(B_k)$$

3. **选择系数并嵌入水印**: 从每个 DCT 系数矩阵 C_k 中选择一个或少数几个中频系数用于嵌入水印信息。选择中频系数是因为它们对图像视觉质量影响较小，且相对于高频系数对压缩等攻击更具鲁棒性。

   假设从第 k 个块中选出的系数为 $c_k(i,j)$，嵌入规则可以采用加性嵌入法：

   $$c'_k(i,j) = c_k(i,j) + \alpha \cdot w_k$$

   或者乘性嵌入法（对 JPEG 压缩更鲁棒）：

   $$c'_k(i,j) = c_k(i,j) \cdot (1 + \alpha \cdot w_k)$$

   其中 $c'_k(i,j)$ 是修改后的 DCT 系数，w_k 是待嵌入的水印比特，α 是嵌入强度因子，它控制着水印的鲁棒性和不可见性之间的平衡。
4. **逆变换**: 对每个修改后的系数矩阵 C'_k 进行逆 DCT（IDCT）变换，得到修改后的图像块 B'_k。

   $$B'_k = \text{IDCT}(C'_k)$$
5. **合成图像**: 将所有修改后的块 B'_k 放回原来的位置，替换掉原始子带中的对应块，形成新的子带 HL'_1。
6. **DWT 逆变换**: 最后,用修改后的子带 HL'_1 与其他未经修改的子带 (LL_1, LH_1, HH_1) 一起进行逆 DWT（IDWT）变换，得到最终的含水印图像 I'。

   $$I' = \text{IDWT}(LL_1, LH_1, HL'_1, HH_1)$$

2.2 水印提取过程

水印提取过程是嵌入过程的逆操作。根据是否需要原始图像，可分为非盲提取和盲提取。以下介绍一种典型的盲提取过程。

1. **DWT 分解**: 对可能含水印的图像（也可能被攻击过）I^* 进行与嵌入时相同级别的 DWT 分解，得到 $LL_1^*, LH_1^*, HL_1^*, HH_1^*$。
2. **分块与 DCT 变换**: 选择与嵌入时相同的子带（如 HL_1^*），并将其分割成与嵌入时大小相同的 $n \times n$ 块 B_k^*。对每个块 B_k^* 进行 DCT 变换，得到系数矩阵 C_k^*。
3. **提取系数**: 从每个 C_k^* 中，定位到嵌入水印时所使用的相同位置 (i,j) 的 DCT 系数 $c_k^*(i,j)$。
4. **解码水印信息**: 提取出的水印比特 w'_k 取决于嵌入规则。

   - 如果采用的是上述的乘性嵌入法，并且假设原始水印 $w_k \in \{-1, 1\}$，提取规则通常基于相关性检测。这需要知道嵌入时使用的原始 DCT 系数的某些统

计特性，或者与其他未修改的系数进行比较。例如，可以与块内其他中频系数的某个均值 T_k 进行比较：

$$w'_k = \begin{cases} 1 & \text{if } c_k^*(i,j) > T_k \\ -1 & \text{if } c_k^*(i,j) \leq T_k \end{cases}$$

- 在更复杂的盲检测方案中，可能需要提取出一系列系数值，并将其与原始水印序列 W 进行相关性计算。设提取的系数序列为 $V^* = \{c_1^*, c_2^*, \ldots, c_N^*\}$，则相关性 S 可以计算为：

$$S = \frac{1}{N} \sum_{k=1}^{N} c_k^* \cdot w_k$$

如果 S 超过某个预设的阈值 T，则判定水印存在。

2.3 鲁棒性分析

该算法的鲁棒性主要来源于以下几个方面：

- **多分辨率特性**: DWT 将图像分解到不同的频带，将水印嵌入到中频子带（LH, HL），可以有效抵抗常见的低通滤波和高通滤波攻击。同时，由于大部分图像能量集中在 LL 子带，对中频子带的修改不易被察觉，保证了不可见性。
- **能量集中特性**: DCT 将每个图像块的能量集中在少数低频系数上。将水印嵌入到能量较大且相对稳定的中频 DCT 系数中，使得水印信息不容易在有损压缩（如 JPEG）等过程中被量化和丢弃。JPEG 压缩本身就是基于分块 DCT 的，因此对 DCT 域的修改具有天然的抗压缩性。
- **信息冗余**: 通常会将同一水印比特重复嵌入到多个不同的块中，或者使用纠错编码（如 BCH 码、RS 码）对水印信息进行编码后再嵌入。这样即使部分水印信息在攻击中被破坏，仍然可以通过冗余信息或纠错码恢复出原始水印。

因此，DWT-DCT 结合的策略利用了两种变换的优点，使得水印信息能够稳固地嵌入到图像的“感知重要”但又非最关键的区域，从而在遭受攻击后仍能被有效检测出来。

3 实验流程

3.1 环境准备

为简化开发流程并利用成熟的算法，本项目将选用 invisible-watermark 这个开源库。该库提供了多种不可见水印的嵌入和提取算法。同时，我们将使用 Pillow (PIL Fork) 库来进行图像的读取、处理和保存。

- **Pillow:** 它是 Python 图像处理事实上的标准库，提供了丰富的功能，用于读取、创建、修改和保存各种格式的图像。在本实验中，它不仅用于加载原始图片，更关键的是用于实现后续的各种图像攻击，如翻转、裁剪和调整对比度。
- **invisible-watermark:** 这是一个封装好的数字水印算法库。我们选择它的原因是其接口简洁，能够让我们专注于测试水印的鲁棒性，而非从零开始实现复杂的频域或空域算法。

为了保证实验的可复现性和一致性，我们使用外部图片 original.png。这样做可以排除原始图像内容（如纹理复杂度、色彩分布）对实验结果的干扰。

我们定义了一段具有特定含义的字符串——**"Confidential: Project 2 - 2025"** 作为水印。这段文本将在后续步骤中被转换为二进制数据嵌入到图片中。

3.2 水印嵌入

首先，我们编写一个脚本 embed_watermark.py，用于将文本水印嵌入到图片中。**完整代码见附录 A，**具体逻辑如下：

- **初始化水印处理器:** 我们实例化了 Watermark 类。构造函数中的 password_img 和 password_wm 参数非常重要，它们作为密钥，用于对水印信息和嵌入位置进行加密或伪随机化处理。这意味着，即使攻击者拥有相同的工具库，如果不知道正确的密钥，也无法提取或破坏水印。
- **加载素材:** 调用 wm.read_img() 方法加载 original.png，并调用 wm.read_wm() 方法将我们的文本字符串加载到处理器中，同时指定其模式为'str'。
- **执行嵌入:** wm.embed() 是核心方法。它执行了底层的算法，将水印字符串的二进制表示，根据密钥生成的伪随机序列，逐位修改宿主图片像素值的最低有效位（LSB, Least Significant Bit）或其他空域特征。修改的幅度非常微小，因此在视觉上是无法察觉的。最终，生成并保存了一张带有不可见水印的新图片 watermarked_image.png。

3.3 鲁棒性测试

现在，我们对已嵌入水印的图片 watermarked_image.png 进行一系列的图像处理攻击。创建一个名为 robustness_test.py 的脚本。我们模拟了四种在现实世界中图片可能遭遇的篡改：

- **水平翻转 (Flip):** 调用 Pillow 的 transpose(Image.FLIP_LEFT_RIGHT) 方法。这是一种全局性的几何变换，它改变了每个像素的坐标（x 变为 width - x），但保留了图像的全部像素信息。我们设计此攻击是为了测试水印是否依赖于像素的绝对空间位置。

- **平移 (Translate):** 此攻击通过裁剪和粘贴模拟。我们先将原图的大部分区域（除去右侧 50 像素）裁剪下来，然后粘贴到一个新的、同样大小的黑色背景画布上，并向右偏移 50 像素。这造成了两种破坏：一是左侧 50 像素宽的区域被黑色填充，丢失了原始信息；二是整个图像内容发生了空间位置的平移，这会挑战解码器的同步机制。
- **中心裁剪 (Crop):** 我们计算出图像中心区域的坐标（保留原图 25% 到 75% 的区域），然后使用 crop() 方法进行裁剪。这是一种严重的内容破坏性攻击，直接导致了图像 75% 的数据被永久丢弃。此攻击旨在测试水印信息是否具有冗余分布特性，以及在大量数据丢失后是否还能恢复。
- **对比度增强 (Contrast):** 使用 ImageEnhance.Contrast 模块，我们将图像对比度增强 1.8 倍。这是一种全局性的像素值变换，它会系统性地改变图像中每个像素的 RGB 值。此攻击用于测试水印信号是否能在像素值发生整体漂移或拉伸后依然保持可检测性。

最后，尝试从所有被攻击后的图片中提取水印信息，**完整代码见附录 B**。我们创建了一个包含所有待测图片（原图及四张攻击后的图片）的列表，并循环遍历，调用 wm.extract() 方法，尝试从每张图片中根据密钥定位并读出嵌入的二进制数据。通过与原始水印信息进行比对，来直观地判断该次攻击下水印的鲁棒性。

4 结果分析

运行以上脚本后，得到如下输出和文件。控制台输出显示每个阶段的操作结果以及最终的提取结果，如下图所示：

- watermarked_image.png: 含有不可见水印的图片。
- 一系列 attacked_*.png 文件，代表不同攻击后的图片。

```
PS D:\2. SDU\01.课程资料\6.大三下\实践\2\script> & S:/MACRO/anaconda/python.exe "d:/2. SDU/01.课程资料/6.大三下/实践/2/script/robustness_test1.py"

原始水印内容: Confidential: Project 2 - 2025
原始图片解码结果: Confidential: Project 2 - 2025

[翻转测试] 解码结果: Confidential: Project 2 - 2025
[平移测试] 解码结果: Confidg~v}ul: }rzwgco~fyvonuk
[截取测试] 解码结果: .v&fvRVN6LLL
[对比度测试] 解码结果: Confidential: Project 2 - 2025
```

图 1: 水印提取结果

表 1: 水印提取结果分析

攻击类型	测试图片文件名	提取结果	鲁棒性评估
无攻击（基准）	watermarked_image.png	信息完全一致	优秀
水平翻转	attacked_flipped.png	信息完全一致	优秀
增强对比度	attacked_contrasted.png	信息完全一致	优秀
平移	attacked_translated.png	信息部分一致	一般
中心裁剪（50%）	attacked_cropped.png	提取失败	极差

对于原图：能 100% 准确地提取出原始水印”Confidential: Project 2 - 2025”。

对于翻转和对比度调整后的图片：成功提取出所有水印。

- 对于翻转，虽然像素的坐标改变了，但像素本身及其邻域关系被完整保留。算法可能不依赖像素的绝对坐标，而是依赖于由密钥生成的伪随机序列来定位嵌入位置。只要图像的像素集合不变，解码器就能循着同样的序列找到水印位，因此翻转操作几乎不影响提取。
- 对于对比度调整，这是一种对像素值的整体线性或非线性变换。水印信息通常被嵌入在像素值的最低有效位 (LSB)。例如，一个像素值从 135 (10000111) 变为 150 (10010110)，其高位发生了显著变化，但最低位可能保持不变或有规律地变化。只要对比度调整的强度没有极端到导致 LSB 信息完全丢失或被噪声淹没，解码器依然可以成功恢复水印。

对于平移后的图片：数字水印的提取过程往往需要一个“起点”或“参考点”，这个过程称为同步。平移攻击严重破坏了这种同步机制。因此只提取出部分信息。

对于截取后的图片：由于大部分图片信息和嵌入的水印信息都已丢失，从这张小得多的图片中提取完整水印几乎是不可能的。提取出了无意义的乱码。

附录 A 水印嵌入代码

```
1 from imwatermark import WatermarkEncoder
2 from PIL import Image
3
4
5 def embed_watermark(original_image_path, watermarked_image_path,
      watermark_text):
6     # 初始化编码器
7     encoder = WatermarkEncoder()
```

```
    encoder.set_watermark("bytes", watermark_text.encode("utf-8"))  # 文本转字节

    # 用OpenCV读取图片（invisible-watermark依赖OpenCV）
    import cv2

    img = cv2.imread(original_image_path)
    img_encoded = encoder.encode(img, "dwtDct")  # 嵌入水印
    cv2.imwrite(watermarked_image_path, img_encoded)  # 保存


# 测试
original_image = "original.png"
watermarked_image = "watermarked_image.png"
watermark_info = "Confidential: Project 2 - 2025"


embed_watermark(original_image, watermarked_image, watermark_info)
```

附录 B 鲁棒性测试代码

```
from PIL import Image, ImageEnhance
from imwatermark import WatermarkDecoder
import cv2
import numpy as np


def decode_watermark(image_path, wm_length_bits=32):
    """尝试从图片中解码水印"""
    decoder = WatermarkDecoder("bytes", wm_length_bits)
    img = cv2.imread(image_path)
    try:
        watermark = decoder.decode(img, "dwtDct")
        return watermark.decode("utf-8", errors="ignore")
    except Exception as e:
        return f"解码失败: {str(e)}"


```

```python
def test_robustness(watermarked_image_path, original_watermark):
    """
    对带水印的图片进行鲁棒性测试并验证水印可读性

    参数:
    watermarked_image_path (str): 带水印的图片路径
    original_watermark (str): 原始水印文本（用于验证）
    """
    try:
        img = Image.open(watermarked_image_path)
        print(f"\n原始水印内容: '{original_watermark}'")
        print(f"原始图片解码结果: {decode_watermark(
            watermarked_image_path, len(original_watermark)*8)}")

        # 1. 翻转测试
        img_flipped = img.transpose(Image.FLIP_LEFT_RIGHT)
        img_flipped.save("attacked_flipped.png")
        print("\n[翻转测试] 解码结果:", decode_watermark("
            attacked_flipped.png", len(original_watermark) * 8))

        # 2. 平移测试
        width, height = img.size
        translated_img = Image.new(img.mode, img.size, color="black")
        region = img.crop((0, 0, width - 50, height))
        translated_img.paste(region, (50, 0))
        translated_img.save("attacked_translated.png")
        print("[平移测试] 解码结果:", decode_watermark("
            attacked_translated.png", len(original_watermark) * 8))

        # 3. 截取测试
        left, top = width * 0.25, height * 0.25
        right, bottom = width * 0.75, height * 0.75
        img_cropped = img.crop((left, top, right, bottom))
        img_cropped.save("attacked_cropped.png")
        print("[截取测试] 解码结果:", decode_watermark("
            attacked_cropped.png", len(original_watermark) * 8))

        # 4. 对比度测试
        enhancer = ImageEnhance.Contrast(img)
```

```
53          img_contrasted = enhancer.enhance(1.8)
54          img_contrasted.save("attacked_contrasted.png")
55          print("[对比度测试] 解码结果:", decode_watermark("
                attacked_contrasted.png", len(original_watermark) * 8))
56
57      except Exception as e:
58          print(f"鲁棒性测试出错: {e}")
59
60
61  if __name__ == "__main__":
62      # 注意: 这里的水印需要与嵌入时一致
63      watermarked_image = "watermarked_image.png"
64      original_wm = "Confidential: Project 2 - 2025"  # 必须与嵌入的水
            印相同
65      test_robustness(watermarked_image, original_wm)
```