

# Indian Institute of Technology Bhubaneswar



## Security & Forensics Lab II

### Experiment No.3

Submitted by :  
Satendra  
21CS06012  
M.Tech(CSE)

## The aim of the experiment is :

1. Embed the watermark image in the original image by designing a new frequency domain algorithm (with the help of DCT coefficients only) that will ensure that the visual quality of the carrier image remains good in terms of PSNR after the watermarking.
2. Design as much attacks as possible to show that the watermarking scheme is robust to such attacks. The possible attacks are listed in the PPT shared with the assignment. You must implement at least two signal processing related attacks and two geometrical attacks and show how your watermarking algorithm is robust against such attacks in terms of NC index.

## **Digital Watermarking**

Digital Watermarking is a method by which a marker is inconspicuously hidden in a noise- tolerant signal such as image, audio, video. That is, it hides some data related to the signal in the signal itself. The hidden message imparts a digital signature to the digital content, and identifies the owner or the authorized distributor, whichever be the case.

This concept is similar to steganography, the difference being in their goals. In steganography, any random message is hidden, and the cover signal is simply there to hide the message. But in watermarking, the message is related to the actual content of the cover signal.

## **Blind and Non-Blind Watermarking**

**Blind** – Cover image is not required for extracting the watermark from the watermarked image.

**Non-blind** – Cover image is needed to separate the watermark from the watermarked image.

## **Embedding Methods**

- **Least Significant Bit Modification**

The least significant bit of each 8-bit pixel is written over by a bit from the watermark. It may be carried out in two ways: either the watermark information can be inserted over each and every pixel of cover image, or the busier areas of the image can be calculated and the watermark is embedded there to enhance imperceptibility.

The least significant bits are highly sensitive to noise, so that the watermark can easily be removed by image manipulations such as rotation and cropping. Thus, the LSB method provides high imperceptibility and less robustness.

- **Frequency Domain Technique**

frequency domain transformation algorithms which can be used individually or in combination with each other to create robust and imperceptible digital watermarks on images like dct,dwt,etc.

### **Discrete Cosine Transform (DCT):**

It is an orthogonal transformation that is used frequently in image compression and is widely accepted in multimedia standards.

It converts the signal into elementary frequency components. The image is represented as a sum of sinusoids of varying magnitudes and frequencies.

Inverse Discrete Cosine Transform gives us back the original image.

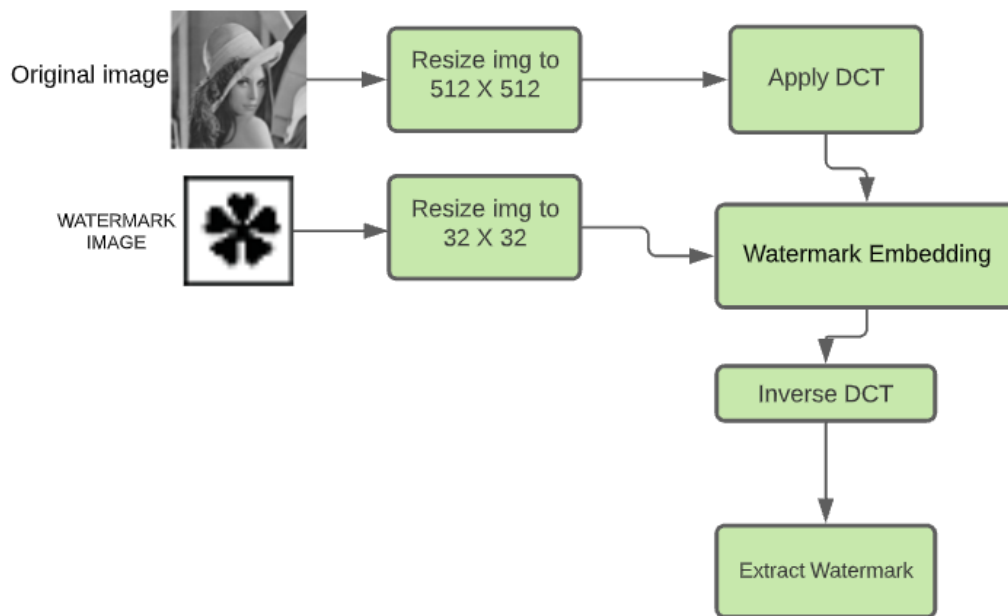
DCT breaks down the image into spectral sub-bands of differing importance. It is similar to Discrete Fourier Transform (DFT) in the way that DCT also converts the signal from spatial domain to frequency domain.

DCT may be global or block based.

### **Working**

- 1.Import all libraries
- 2.Take input images(wtermark\_image,original\_image)
- 3.convert to gray scale
- 4.Resize to 32x32 and 512x512 respectively
- 5.Apply dct
- 6.Embed the watermark
- 7.Inverse\_dct
- 8.Extract\_watermark
- 9.Compare psnr value
- 10.Implement attacks

## Working Flow chart



### Embedding algorithm:

step 1: Take dct and watermark\_image as input

step 2: Convert watermark\_array to flat array(1D)

step 3: Repeat the following step until all the watermark pixels are embedded into the original image,

- i. Take next 16 x 16 matrix from original\_image
- ii. Select the index
- iii. Put the watermark\_image pixels into the selected index

step 4: Return the result

```
def embed(dct, watermarkImage):
    sig=8
    water_mark=watermarkImage.ravel()
    flag=0
    for i in range (0, 512, 16):
        for j in range (0, 512, 16):
            if(flag<len(water_mark)):
                dct[i+11][j+11]=water_mark[flag]
                flag=flag+1
            if(flag>=len(water_mark)):
                break

    return dct
```

## Extraction Algorithm

Step 1:Take flat extract\_array=[]

Step2 :Apply\_dct

Step3:Append the extracted pixels

Step 4:Reshape to 32x32

Step5:The resultant image is the extracted watermark.

```
def extract_watermark(im_dct):
    extract_water_mark=[]
    sig=0
    i_dct=apply_dct(im_dct)
    for i in range (0, 512, 16):
        for j in range (0, 512, 16):
            if(sig<(32*32)):
                extract_water_mark.append(i_dct[i+11][j+11])
                sig=sig+1
            if(sig>=(32*32)):
                break

    extract_water_mark=np.array(extract_water_mark).reshape(32,32)
    return extract_water_mark
```

## Experimental Results and Analysis.

Performance of the proposed algorithm is analyzed on the basis of PSNR and NC values.

**NC** – It is a measure of similarity between the original watermark and the extracted watermark.

$$norm\_corr(x, y) = \frac{\sum_{n=0}^{n-1} x[n] * y[n]}{\sqrt{\sum_{n=0}^{n-1} x[n]^2 * \sum_{n=0}^{n-1} y[n]^2}}$$

**Cover Image:**lena.png    **Size:** 512x512

**Watermark:** logo.png    **Size:** 32x32

### Result:

PSNR value **26.811854465**

NC value of watermark and extracted watermark is **0.9999989**

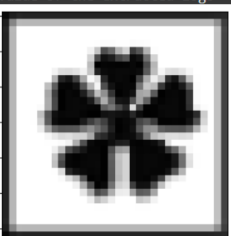
```
print(compute_psnr(carrierImage, originalImage))

26.81997012680888

plt.imshow(extracted_watermark, cmap='gray')
img1=watermarkImage
img2=extracted_watermark

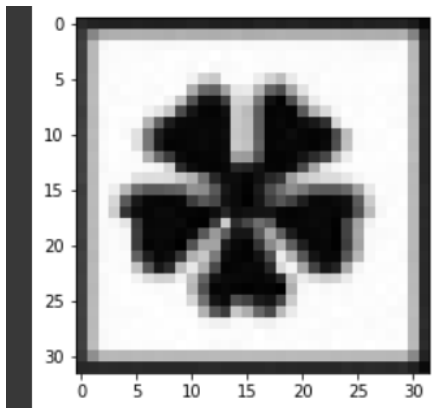
nc_res = cv2.matchTemplate(img1.astype(np.uint8), img2.astype(np.uint8), cv2.TM_CCORR_NORMED)
print('NC value of the extracted logo = {}'.format(nc_res))

NC value of the extracted logo = [[0.9999989]]
```

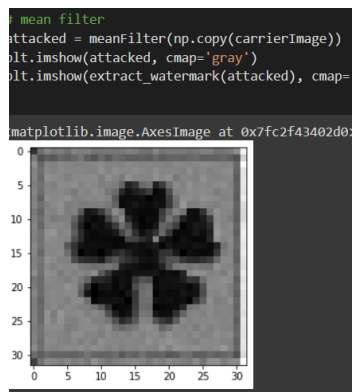


Attacks:

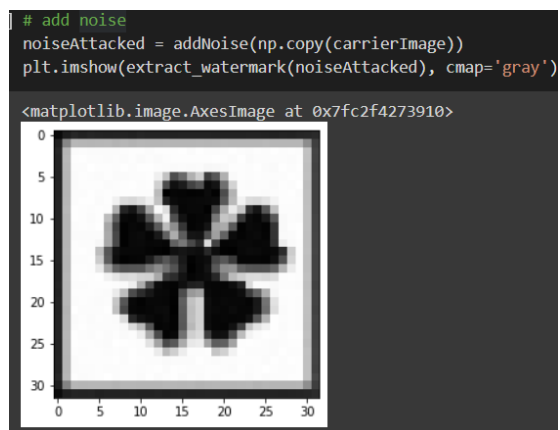
## 1.Rotate image



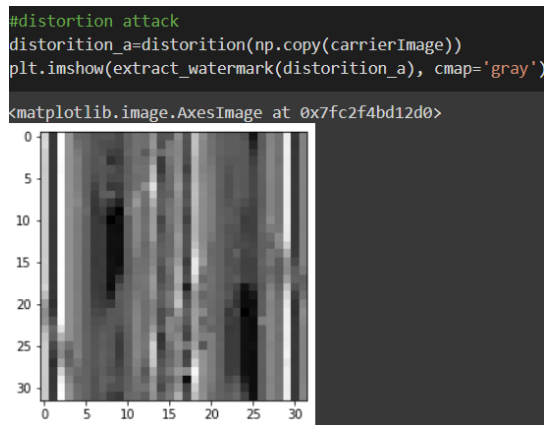
## 2.mean filter



## 3.add noise



## 4.Distortion



**Code link:**

<https://colab.research.google.com/drive/1SBv9EvRJhIJ55XHMZtH--3j9Q4KJ-b32?usp=sharing>