Hands-On Image Watermarking using Python

MBKM 2021
Data Security Research Group, Pusat Riset Informatika, BRIN

Introduction

Python-based watermarking:

- (+) Rapid development, Applicative purpose
- (-) Not for proposing a new method/ make improvement (use Matlab instead)

Topics:

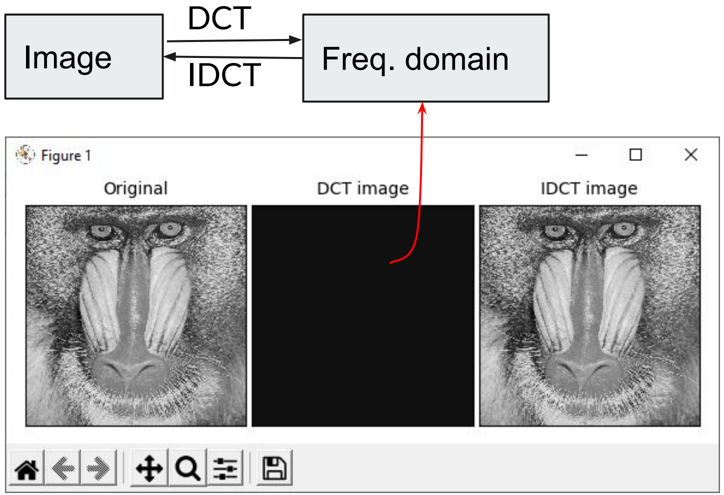
1. DCT (Discrete Cosine Transform) : Transform to frequency domain (Cosine)
2. DWT (Discrete Wavelet Transform) : Transform to Wavelet domain
3. SVD (singular value Decomposition) : Image decomposition
4. Exercises : Cascade computation (DWT, DCT, SVD)
5. SVD-based Image watermarking : Image watermarking
6. Image comparison : Compute the PSNR, SSIM, and NC

1. DCT (Discrete Cosine Transform)

DCT libraries:

- OpenCV: https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html#dct
- Scipy: https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.dct.html#scipy.fftpack.dct

For this tutorial we will use the scipy one

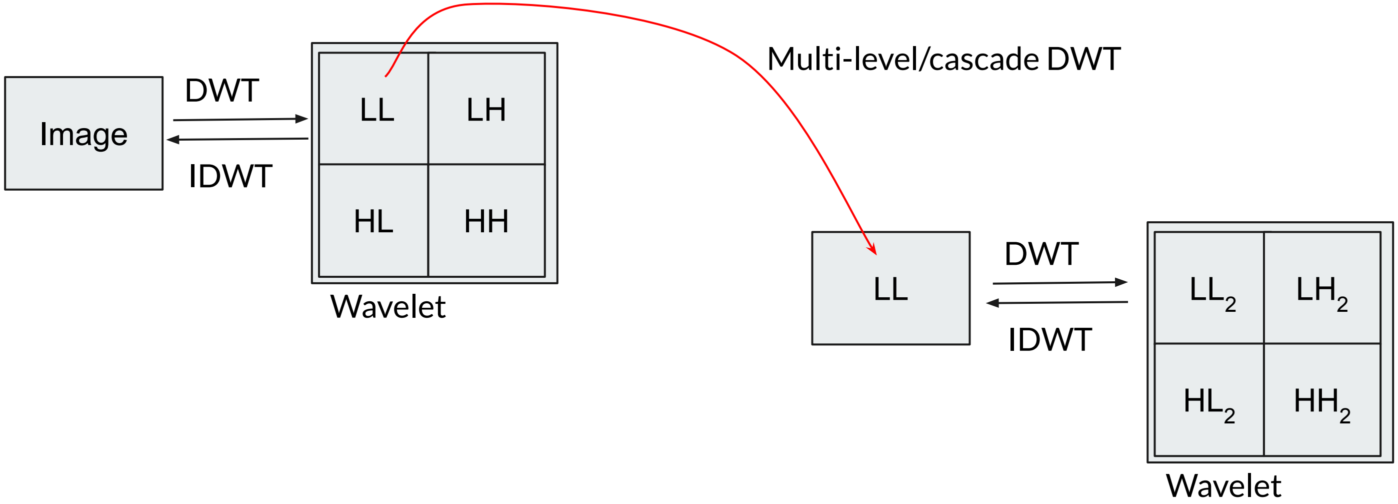


DCT.py

```python
from scipy.fftpack import dct, idct
from myutil import loadImage, plotImage

image = loadImage('babon.png')

def dct2(block):
    return dct(dct(block.T, norm = 'ortho').T, norm = 'ortho')

def idct2(block):
    return idct(idct(block.T, norm = 'ortho').T, norm = 'ortho')

dct_img = dct2(image)
idct_image = idct2(dct_img)
plotImage([image, dct_img, idct_image], ["Original", "DCT image", "IDCT image"])
```

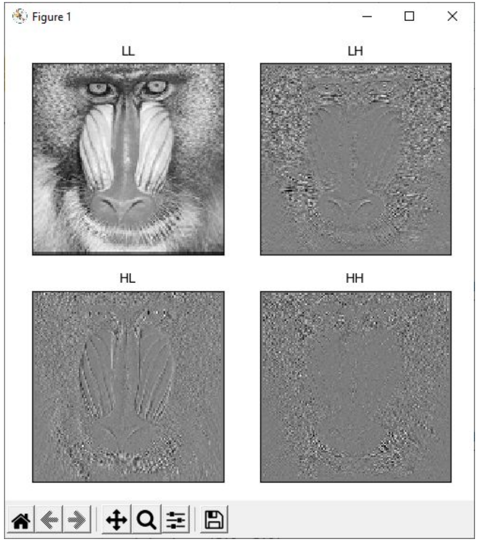
2. DWT (Discrete Wavelet Transform)

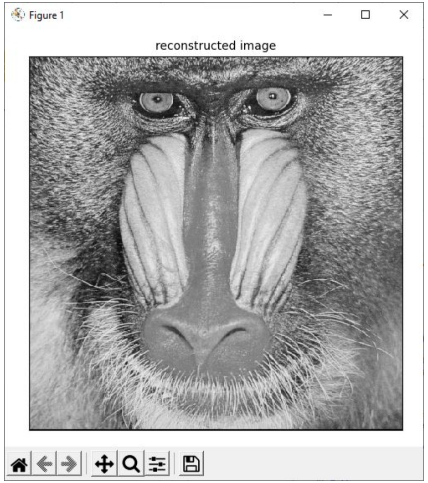
DWT libraries: PyWavelet, https://github.com/PyWavelets/pywt



DWT.py

```python
from scipy.fftpack import dct, idct
from myutil import loadImage, plotImage
import pywt
import numpy as np

image = loadImage('babon.png')
print("image size:",image.shape)

def multilevelDWT(image, N=1):
    ctr = 0
    result = {}
    HF = []
    LF = []
    data = image
    for i in range(N):
        result[ctr] = pywt.dwt2(data, 'bior1.3')
        data, (LH, HL, HH) = result[ctr]
        if i == 0:
            HF.append( np.array([LH, HL, HH]) )
            LF.append( data )
    plotImage([data, LH, HL, HH], ["LL","LH","HL","HH"])
    return LF, HF

LF, HF = multilevelDWT(image, 2)

def reconstructImg(LF, HF, showimg=False):
    temp = np.array([LF, HF])
    img=pywt.idwt2(temp,'bior1.3')
    if showimg:
        plotImage([img], ["reconstructed image"])
    return img

img = reconstructImg(LF[0], HF[0], True)
```

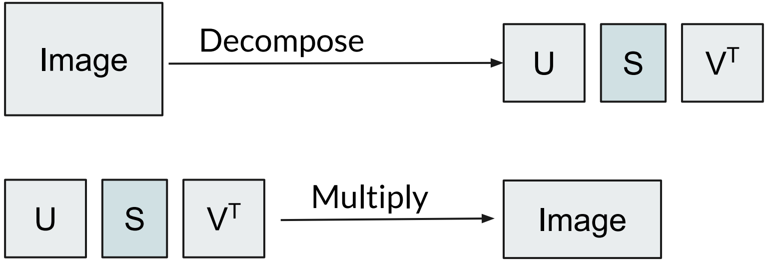


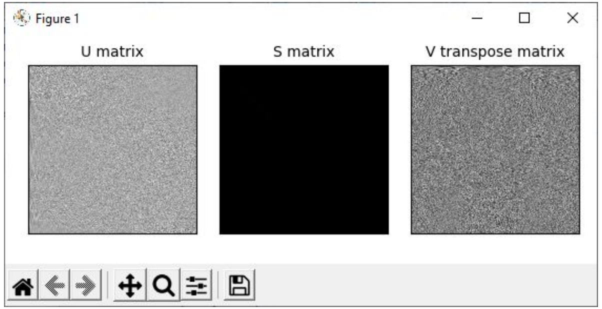


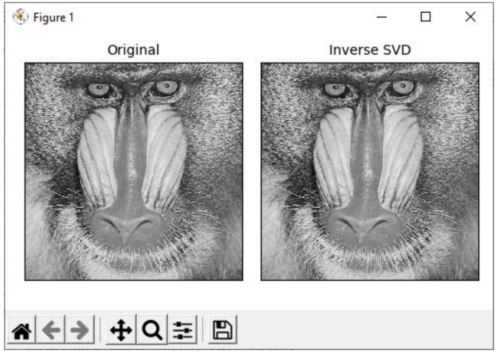
3. SVD (singular value Decomposition)

SVD: decompose a data into singular vectors (U & V) and singular value (S)

libraries:numpy linear algebra, https://numpy.org/doc/stable/reference/routines.linalg.html







SVD.py

```
from scipy.fftpack import dct, idct
from myutil import loadImage, plotImage
import pywt
import numpy as np

image = loadImage('babon.png')

def decomposeSVD(block):
    u, s, vh = np.linalg.svd(block)
    print( "u.shape", u.shape)
    print( "s.shape", s.shape)
    print( "vh.shape",vh.shape)
    return u, s, vh, block.shape

def reconstructSVD(u,s,vh, size, showimg=False):
    reconst_img = np.matrix(u[:, :size]) * np.diag(s[:size]) * np.matrix(vh[:size, :])
    if showimg:
        fig=plt.figure()
        ax=fig.add_subplot(1,1,1)
        plt.axis('off')
        plt.imshow(reconst_img, cmap=plt.get_cmap("gray"))
        plt.savefig('test.png', bbox_inches='tight', transparent=True, pad_inches=0)
        plt.show()
        return reconst_img
    else:
        return reconst_img

u, s, vh, imgshape = decomposeSVD(image)
matrix_s = np.diag(s)
plotImage([u, matrix_s, vh], ['U matrix', 'S matrix', 'V transpose matrix'])

newImage = reconstructSVD(u, s, vh, imgshape[0])
plotImage([image, newImage], ['Original', 'Inverse SVD'])
```

SVD can decompose any matrix, not limited to image but also DCT or wavelet data

4. Exercises

Implements:

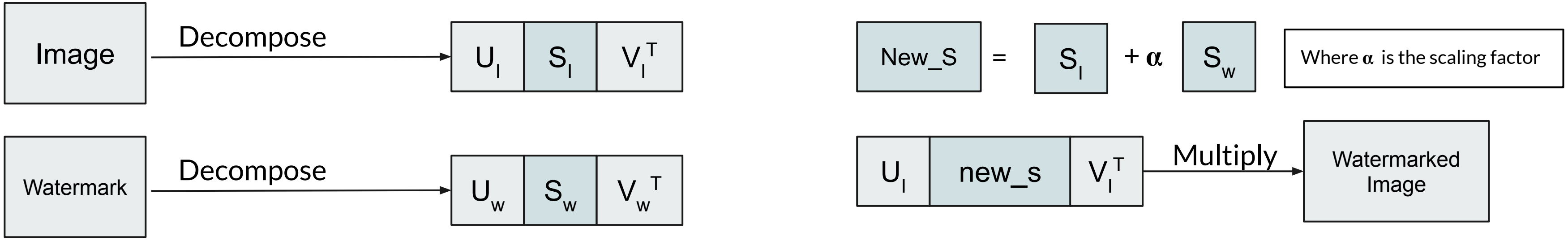
1. DWT-DCT transform
2. DCT-SVD decomposition and its inverse
3. DWT-SVD decomposition and its inverse
4. DWT-DCT-SVD decomposition and its inverse

Hints:

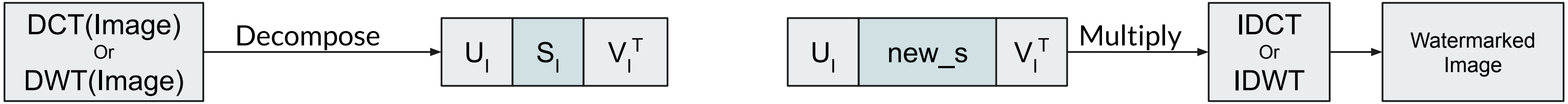
1. DWT-DCT.py
2. DCT-SVD.py
3. DWT-SVD.py
4. DWT-DCT-SVD.py

5. SVD-based Image Watermarking (embedding)

Adding watermark data into singular value (S) of the decompose data

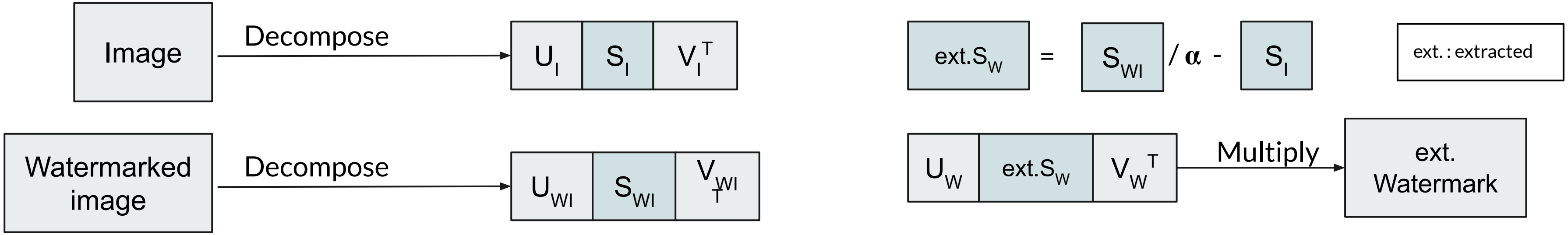


Watermarking usually done in transform domain (DCT/DWT), not in the spatial domain (image pixels)



5. SVD-based Image Watermarking (extraction)

Watermarking extraction (non-blind): compare the singular value from the watermarked image with the original image.

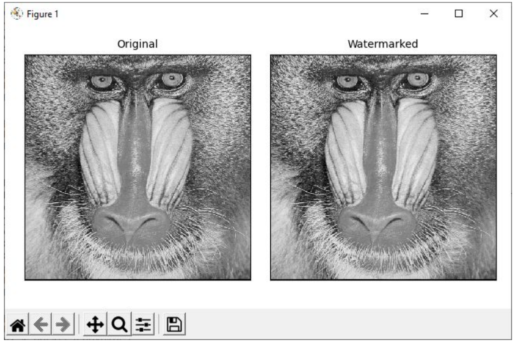


Watermark quality: compare the extracted watermark with the original watermark:

- PSNR : Peak to signal noise ratio
- SSIM: Structural Similarity Index
- NC: Normalized correlation

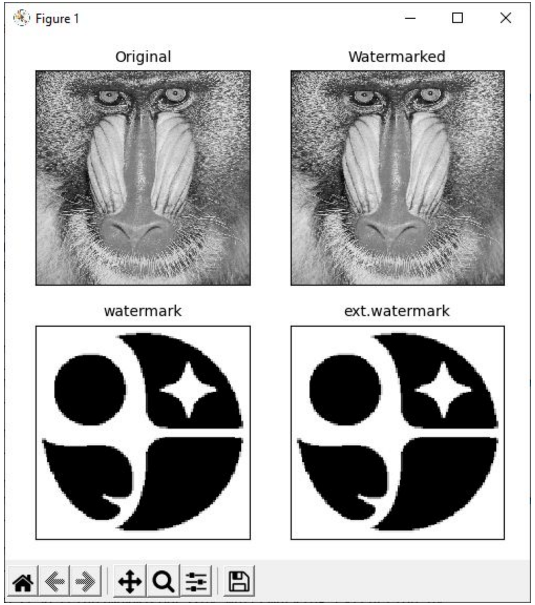
Watermarking embedding (dwt-dct-svd)

```python
def embedWatermark(image, watermark, alpha = 0.1):
    #1st level DWT
    LF, HF = multilevelDWT(image)

    #DCT
    dctImg = dct2(LF[0])

    #SVD image
    u, s, vh, imgshape = decomposeSVD(dctImg)
    matrix_s = np.diag(s)

    #SVD watermark
    def addWatermarkData(s_img, watermark_image, alpha_value=.1):
        #note: the watermark image must has smaller dimension than the quarter of the host image
        u_wi, s_wi, vh_wi, dimensi_wi = decomposeSVD(watermark_image)
        s_watermark = np.pad(s_wi, ((0,s_img.shape[0] - s_wi.shape[0] )), 'constant')

        tmp = s_watermark * alpha_value
        new_s = s_img + tmp
        return new_s

    new_s = addWatermarkData(s, watermark, alpha)

    #ISVD using new_s
    newSVD = reconstructSVD(u, new_s, vh, imgshape[0])

    #IDCT
    idctImg = idct2(newSVD)

    #inverse DWT
    newImage  = reconstructImg(idctImg, HF[0])
    return newImage

if __name__ == "__main__":
    image = loadImage('babon.png')
    wimage = loadImage('brinbw.png')

    newImage = embedWatermark(image, wimage)

    plotImage([image, newImage], ['Original', 'Watermarked'])
```



Watermarking extraction (dwt-dct-svd)

```
def extractWatermark(watermarkedImage, originalImage, alpha=0.1):
    def dwtdctsvd(image):
        #1st level DWT watermarkedImage
        LF, HF = multilevelDWT(image)

        #DCT
        dctImg = dct2(LF[0])

        #SVD image
        u, s, vh, imgshape = decomposeSVD(dctImg)
        return s

    s_wi = dwtdctsvd(watermarkedImage)
    s_oi = dwtdctsvd(originalImage)
    #1st level DWT originalImage

    s_w = s_wi - s_oi
    s_w = s_w / alpha
    return s_w

def constructExtractedWatermark(originalWatermark, ext_s):
    uw, sw, vhw, imgshape_w = decomposeSVD(originalWatermark)
    watermark = reconstructSVD(uw, ext_s, vhw, imgshape_w[0])
    return watermark


if __name__ == "__main__":
    image = loadImage('babon.png')
    wimage = loadImage('brinbw.png')

    newImage = embedWatermark(image, wimage)

    new_s = extractWatermark(newImage, image)

    newWimage = constructExtractedWatermark(wimage, new_s)

    plotImage([image, newImage, wimage, newWimage], ['Original', 'Watermarked', 'watermark', 'ext.watermark'])
```



6. Image comparison

Watermark quality: compare the extracted watermark with the original watermark:

- PSNR : Peak to signal noise ratio
- SSIM: Structural Similarity Index
- NC: Normalized correlation

$$PSNR = 10\log_{10}(\frac{255^2}{MSE})$$

where

$$MSE = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(I(i,j) - I_w(i,j))^2$$

$$SSIM\ (x,y) = [l(x,y)^{\alpha}][c(x,y)^{\beta}][s(x,y)^{\gamma}]$$

$$= \frac{(2\mu_x\mu_y + C_1)(2\sigma_x\sigma_x + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

$$MSSIM\ (x,y) = \frac{1}{M}\sum_{j=1}^{m} SSIM(x_j, y_j)$$

$$NC = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} W(i,j)W'(i,j)}{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} W(i,j)}\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} W'(i,j)}}$$

```python
def computeStats(oriImg, compImg,isMultichannel=False):
    allres = []
    img1 = oriImg
    img2 = compImg
    psnr = cv2.PSNR(img1, img2)
    print("PSNR:",psnr)
    allres.append(psnr)

    if isMultichannel:
        (score, diff) = structural_similarity(img1, img2, multichannel=True, full=True)
    else:
        (score, diff) = structural_similarity(img1, img2, full=True)
    diff = (diff * 255).astype("uint8")
    print("SSIM:",score)
    allres.append(score)

    result = cv2.matchTemplate(img1,img2, cv2.TM_CCOEFF_NORMED)
    print("NC:",result[0][0])
    allres.append(result[0][0])
    return allres
```

```python
if __name__ == "__main__":
    image = loadImage('babon.png')
    wimage = loadImage('brinbw.png')
    newImage = embedWatermark(image, wimage)
    new_s = extractWatermark(newImage, image)
    newWimage = constructExtractedWatermark(wimage, new_s)

    print(wimage.shape)
    print(newWimage.shape)

    print("Compare Image:")
    computeStats(image, newImage, True)

    print("Compare Watermark:")
    computeStats(wimage, newWimage)
```

```
Compare Image:
PSNR: 83.2805991605096
SSIM: 0.9991435223803447
NC: 0.99943674
Compare Watermark:
PSNR: 161.84188081966127
SSIM: 1.9961557680491062
NC: 1.0

D:\Documents\_Keltian\mbkm\codes>
```