

import module

```
In [1]: import numpy as np
import cv2
import matplotlib.pyplot as plt
import math
import hashlib
import sys
import random
import os
import pandas as pd
```

Haar Transform

Forward IWT

```
In [2]: def forwardIWT(matrix1):
m,n=matrix1.shape
matrix=matrix1.tolist() #matrix is a list

#row operation
row = list()
for i in range(m):
    a=[]
    d=[]
    for p in range(0,m):
        if m>2*p+1: #how many time we get the a & d value
            x=matrix[i] #get the rows
            t=math.floor((x[2*p]+x[2*p+1])/2)
            a.append(t)
            t=math.floor(x[2*p]-x[2*p+1])
            d.append(t)

    combine=a+d #combine the list
    row.append(combine)
    a.clear()
    d.clear()

#column Operation
column=list() #operation w.r.t 'row' matrix
row1=[[x[i] for x in row] for i in range(len(row[0]))] #transpose of row
for i in range(n):
    a=[]
    d=[]
    for p in range(0,n):
        if n>2*p+1: #how many time we get the a & d value
            x=row1[i] #get the rows
            t=math.floor((x[2*p]+x[2*p+1])/2)
            a.append(t)
            t=math.floor(x[2*p]-x[2*p+1])
            d.append(t)

    combine=a+d #combine the List
    column.append(combine)
    a.clear()
    d.clear()

result=[[x[i] for x in column] for i in range(len(column[0]))] #again transpose of column matrix
res=np.array(result)
final=np.reshape(res, (m,n))
return final
```

Inverse IWT

```
In [3]: def inverseIWT(matrix1):
matrix=list()
m,n=matrix1.shape
matrix=matrix1.tolist() #matrix is a list

#Column Operation
trans_matrix=[[x[i] for x in matrix] for i in range(len(matrix[0]))] #transpose of matrix
column_tr=list()
for i in range(n):
    flag=trans_matrix[i]
    a=flag[:n//2]
    d=flag[n//2:]
    temp=list()
    for p in range(0,n):
        if n>2*p+1: #how many time we get the value
            x1=a[p]+math.floor((d[p]+1)/2)
            temp.append(x1)
            x2=x1-d[p]
            temp.append(x2)
    column_tr.append(temp)

#row operation
column=[[x[i] for x in column_tr] for i in range(len(column_tr[0]))] #transpose of column matrix
row=list()
for i in range(m):
    flag=column[i]
    a=flag[:n//2]
    d=flag[n//2:]
    temp=list()
    for p in range(0,m):
        if m>2*p+1: #how many time we get the value
            x1=a[p]+math.floor((d[p]+1)/2)
            temp.append(x1)
            x2=x1-d[p]
            temp.append(x2)
    row.append(temp)

res=np.array(row)
result=np.reshape(res, (m,n))
return result
```

Histogram Shifting

Shifting

```
In [4]: def histogram_shifting(matrix, keyword):
row,col=matrix.shape
img=matrix.flatten() #image in 1D
list_img=list(set(img))
sort_pixel=sorted(list_img)

freq=[] #store freq of each element in sorted manner of element
for i in sort_pixel:
    freq.append(list(img).count(i))
key_size=len(keyword) #size of payload
```

```
# print("Maximum freq in the image",max(freq))

#find base point
flag=0
x=sorted(list_img)
for i in freq:
    if i>=key_size:
        temp=freq.index(i)
        if x[temp]>0 or x[temp]<=0:
            base_point=x[temp] #get base point
            flag +=1
if flag==0:
    print("No base Point")
    sys.exit()
    # return False, False

'''cahnge value of base point+1 and no. of basepoint>len(keyword),
i.e. more no of base point the the length of keyword,
then make those extra base point increase by 2'''
flag=len(keyword)
count=0
for j in range(0, len(img)):
    if img[j]==base_point+1:
        img[j] +=1

    if img[j]==base_point:
        count +=1
        if count>flag:
            img[j] = img[j]-1 #chnage the extra bp to pixel with the previous value
                                #eg. bp=152 then extra bps will be 151

#histogram shifting
index=0
key=str(keyword)
for item in range(0, len(img)):
    if img[item]==base_point and index<len(key):
        if key[index]=='0':
            img[item]= img[item] #string is 0 then value is 0
        elif key[index]=='1':
            img[item] += 1 #string is 0 then value increase by 1
        index += 1

result=np.resize(img, (row,col))
return result,base_point
```

Extraction

```
In [5]: def histogram_extract(matrix, base_point):
img=matrix.flatten()
payload=''
for item in img:
    if item==base_point:
        payload += '0'
    elif item==base_point+1:
        payload += '1'
return payload
```

SHA 256

```
In [6]: def sha256_code(img):
img_str = str(img)#convert to string

# Compute the SHA-256 hash of the string representation
sha256_hash = hashlib.sha256(img_str.encode('utf-8'))

# Get the hexadecimal representation of the hash
hex_digest = sha256_hash.hexdigest()

# Convert the hexadecimal digest to binary
binary_digest = bin(int(hex_digest, 16))[2:].zfill(256)
return hex_digest,binary_digest
```

Break & Make image into 4parts

```
In [7]: #extract 1/4th Components of image

def extract_quarter_components(image):
    height, width = image.shape

    # Determine the indices for slicing each quadrant
    half_height = height // 2
    half_width = width // 2

    # Extract each quadrant
    top_left = image[:half_height, :half_width]
    top_right = image[:half_height, half_width:]
    bottom_left = image[half_height:, :half_width]
    bottom_right = image[half_height:, half_width:]

    return top_left, top_right, bottom_left, bottom_right

#Make the image from 4parts
def reconstruct_image(top_left, top_right, bottom_left, bottom_right):
    height, width = top_left.shape

    # Create an empty array to hold the reconstructed image
    reconstructed_image = np.empty((height * 2, width * 2))

    # Place each component in its respective position
    reconstructed_image[:height, :width] = top_left
    reconstructed_image[:height, width:] = top_right
    reconstructed_image[height:, :width] = bottom_left
    reconstructed_image[height:, width:] = bottom_right

    return reconstructed_image
```

Convert Image to Binary

```
In [8]: def binaryImage(img):
x=list()
for row in img:
    for num in row:
        y=format(num, '08b')
        x.append(y) # '08b' ensures leading zeros for each byte
str_bin=''
for item in x:
    str_bin+=item
return x, str_bin
```

Arnold's Cat Map

```
In [9]: def arnold_cat_map(image,iteration):
        row,col=image.shape
        process_img = np.zeros_like(image)
        original_img=image.copy()

        # plt.title(f'Original Image')
        # plt.imshow(image,cmap='gray')
        # plt.axis('off')
        # plt.show()

        #calculating each pixel
        count=1
        while (count<=iteration):
            for x in range(row):
                for y in range(col):
                    nx=(2*x+y)%row
                    ny=(x+y)%col
                    process_img[nx,ny] = image[x,y]
                image=process_img.copy()
            count +=1
        return process_img
```

Convert Binary Image to 2D Image

```
In [10]: def decimalImage(binary_list,size):
        decimal_list = [] # Initialize an empty list to store decimal integers

        # Iterate over each binary string in the input List
        for binary_str in binary_list:
            decimal_int = int(binary_str, 2) # Convert the binary string to decimal integer
            decimal_list.append(decimal_int) # Add the decimal integer to the result list

        #convert to a 2D List equal to size of secert image
        result=list()
        result = [decimal_list[i*size : (i+1)*size] for i in range(size)]
        return result
```

Convert Text to binary & Binary to text & Binary to Text

```
In [11]: def text2binary(text):
        binary_text = ''.join(format(ord(char), '08b') for char in text)
        return binary_text

def binary2text(binary_text, errors='replace'):
    # Split the binary string into 8-bit substrings
    binary_list = [binary_text[i:i+8] for i in range(0, len(binary_text), 8)]

    # Convert binary to text
    text = ''.join(chr(int(binary, 2)) for binary in binary_list)

    # Encode and decode using utf-8 to ensure compatibility
    encoded_text = text.encode('utf-8', errors=errors)
    final_text = encoded_text.decode('utf-8', errors=errors)

    return final_text

def binary2number(binary_string):
    # Split the binary string into 8-bit segments
    binary_segments = [binary_string[i:i+8] for i in range(0, len(binary_string), 8)]
    # Convert each 8-bit binary segment to integer
    integer_list = [int(segment, 2) for segment in binary_segments]
    return np.array(integer_list)
```

Hamming Code

```
In [12]: #Hamming Code
def calcRedundantBits(m):
    for i in range(m):
        if(2**i >= m + i + 1):
            return i

def posRedundantBits(data, r):
    j = 0
    k = 1
    m = len(data)
    res = ''
    for i in range(1, m + r+1):
        if(i == 2**j):
            res = res + '0'
            j += 1
        else:
            res = res + data[-1 * k]
            k += 1
    return res[::-1]

def calcParityBits(arr, r):
    n = len(arr)
    for i in range(r):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])
        arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
    return arr

def detectError(arr, nr):
    n = len(arr)
    res = 0
    for i in range(nr):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])
        res = res + val*(10**i)
    return int(str(res), 2)

'''Hamming Code Insertion'''
def hammingcode(bindata): #give binary string as argument
    output=''
    for i in range(0,len(bindata),8):
        data=bindata[i:i+8]
        m=len(data)
        r=calcRedundantBits(m)
        arr=posRedundantBits(data,r)
        arr=calcParityBits(arr,r)
        output+=str(arr)
    return output,r #return binary string along with parity & return no. of redudent bits(r)
```

```
'''Hamming Code Error Correction'''
def removeerror(outdata,r): #takes argument the bit string along with parity & return no. of redundant bits(r)
    output=''
    for i in range(0,len(outdata),12):
        data=outdata[i:i+12]
        pos=detectError(data,r)
        if pos==0:
            temp=data[0:4]+data[5:8]+data[9]
        else:
            pos=len(data)-pos+1
            if data[pos-1]=='0':
                data=data[0:pos-1]+'1'+data[pos:]
            else:
                data=data[0:pos-1]+'0'+data[pos:]

        temp=data[0:4]+data[5:8]+data[9]
        output+=temp
    return output #return correct binary string
```

Image Hiding

```
In [13]: def imageHiding(image, secret, epr):
    #1. Forward IWT
    fIWT_img=forwardIWT(image) #apply 1st time to image
    LL,HL,LH,HH = extract_quarter_components(fIWT_img)

    #2.SECRET IMAGE TO SHA256 & CONVERT IT TO BINARY
    hash_image,hash_bin=sha256_code(secret)

    #3. Histogram Shifting of secret image to LL & SHA to HH
    secret_bin, secret_str=binaryImage(secret) #convert to binary

    #4. Histogram Shifting of secret image to HH & SHA to HH & EPR to LH
    LL_coded, bp1=histogram_shifting(LL, hash_bin) #SHA coded to LL
    HH_coded, bp2=histogram_shifting(HH, secret_str) #Secret Image coded to HH
    LH_coded, bp3=histogram_shifting(LH, epr) #EPR coded to LH
    # print(bp1,bp2,bp3)

    #5. Combine 4 Components
    forwardIWT_Stego=reconstruct_image(LL_coded, HL, LH_coded, HH_coded)
    Stego_Image=inverseIWT(forwardIWT_Stego) #again Inverse IWT to image

    #Return stego image & base points(bp1=LL, bp2=HH, bp3=LH)
    return Stego_Image, bp1,bp2,bp3
```

Image Extraction

```
In [14]: def imageExtraction(stego_img,bp1,bp2,bp3):
    #1. Forward IWT
    fIWT_img=forwardIWT(stego_img) #apply 1st time to image
    LL1,HL,LH1,HH1 = extract_quarter_components(fIWT_img)

    #2. Histogram Extraction from LL & HH
    extract_secret_bin=histogram_extract(HH1,bp2) #extract secret image in binary,bp2=159
    extract_sha_bin=histogram_extract(LL1, bp1) #extract SHA in binary, bp1=15
    extract_epr_bin=histogram_extract(LH1, bp3) #extract EPR binary from LH, bp3

    #3. Convert the binary str of image to Actual Secret Image
    extract_secret_lis=list()
    for i in range(0, len(extract_secret_bin) - 1, 8):
        pair = extract_secret_bin[i:i+8] # Extract a pair of consecutive digits
        extract_secret_lis.append(pair)

    #convert to 2d decimal array
    size=int(math.sqrt(len(extract_secret_lis)))#find size, square image so row & col value same
    extract_secret_decimal=decimalImage(extract_secret_lis,size)
    extract_secret_image=np.array(extract_secret_decimal)

    return extract_secret_image, extract_sha_bin, extract_epr_bin #return secret image & SHA in binary from stego image & epr
```

Calculate Erros

calculate normalized_cross_correlation (NCC)

```
In [15]: def NCC(img1, img2):
    # Ensure both images have the same depth and type
    gray1 = cv2.normalize(img1, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)
    gray2 = cv2.normalize(img2, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)

    # Perform template matching
    result = cv2.matchTemplate(gray1, gray2, cv2.TM_CCORR_NORMED)

    # Extract the maximum correlation coefficient
    max_corr_coeff = cv2.minMaxLoc(result)[1]

    return max_corr_coeff
```

ALL IMAGES

1) Image Sharnpening

code

```
In [16]: def sharpen_image(image):
    # Define the sharpening kernel
    kernel = np.array([[ -1, -1, -1],
        [-1, 9, -1],
        [-1, -1, -1]])
    # Apply the kernel to the image
    sharpened_image = cv2.filter2D(image, -1, kernel)
    return sharpened_image

In [17]: def sharp_attack():
    covering_list=os.listdir(r"C:\Study Meterial\Semester Project\Practical\WORK\Cover Images")
    secretimg=cv2.imread(r"C:\Study Meterial\Semester Project\Practical\WORK\splash.tiff",0)
    secret=cv2.resize(secretimg,(16,16), interpolation=cv2.INTER_AREA) #secert image of size 8x8

    #Read The EPR File
    file = open("C:\Study Meterial\Semester Project\Practical\WORK\Without Attack\sample epr - Copy.txt", 'r')
    EPR_txt = file.read()
    file.close()
    EPR_bin=text2binary(EPR_txt) #EPR text in binary
```



```
#apply arnold cat map using iteration=7
secret_cat=arnold_cat_map(secret,iteration=7) #total need 12 for 16x16
#WITHOUT ATTACK
psnr=list()
mse=list()
ncc=list()
sha_list=list()
epr_ncc=list() #for epr only

for img in covering_list:
    cover=cv2.imread(f"C:\Study Meterial\Semester Project\Practical\WORK\Cover Images\{img}",0)
    #hide
    stego_img, bp1,bp2,bp3 =imageHiding(cover,secret_cat, EPR_bin)
    cv2.imwrite(f"C:\Study Meterial\Semester Project\Practical\WORK\Attack\Image Sharnpening\Stego Images\Stego_{img[:-4]}(splash).png", stego_img)

    '''Apply ATTACKS Here'''
    attack_stego_img= sharpen_image(stego_img.astype(np.uint8)) #attack on image
    #save attack image
    cv2.imwrite(f"C:\Study Meterial\Semester Project\Practical\WORK\Attack\Image Sharnpening\Attack Images\Sharpen_Stego_{img[:-4]}(splash).png", attack_stego_img)

    #extraction & save files
    extract_secret_image, extract_sha_bin, extract_epr_bin=imageExtraction(attack_stego_img.astype(np.float64), bp1,bp2,bp3)
    #save EPR text from each iamge
    temp=binary2text(extract_epr_bin)
    path=f"C:\Study Meterial\Semester Project\Practical\WORK\Attack\Image Sharnpening\Extracted EPR\Extract EPR Stego_{img[:-4]}.txt"
    with open(path, 'w', encoding='utf-8', errors='replace') as file_epr:
        file_epr.write(temp)
    file_epr.close()

    #compare SHA
    hash_image,hash_bin=sha256_code(extract_secret_image)
    temp=(hash_bin==extract_sha_bin)
    sha_list.append(temp)
    #Arnold Cat Map Again
    secret_after_cat=arnold_cat_map(extract_secret_image,iteration=5)
    #save watermark image
    cv2.imwrite(f"C:\Study Meterial\Semester Project\Practical\WORK\Attack\Image Sharnpening\Extracted Watermark\Watermark Stego_{img[:-4]}.png", secret_after_cat)

    #calculate erros for secret image
    image1 = cover.astype(np.float64)
    image2 = stego_img.astype(np.float64)
    psnr.append(cv2.PSNR(image1,image2))
    mse.append(np.mean((image1 - image2) ** 2))
    ncc.append(NCC(secret,secret_after_cat))

    #calculate error for EPR text
    epr_inserted=binary2number(EPR_bin)
    epr_extracted=binary2number(extract_epr_bin)
    res=NCC(epr_inserted,epr_extracted)
    epr_ncc.append(res)

#for secert image
data_img={
    'Cover Image':covering_list,
    'Secret Image':['splash.tiff']*10,
    'PSNR':psnr,
    'MSE':mse,
    'NCC':ncc,
    'SHA':sha_list
}

#for EPR text
data_epr={
    'Cover Image':covering_list,
    'EPR':['Sample EPR']*10,
    'NCC':epr_ncc
}

df1=pd.DataFrame(data_img)
df2=pd.DataFrame(data_epr)
return df1,df2
```

In [19]: data1,data2=sharp_attack()
data1

Out[19]:

| | Cover Image | Secret Image | PSNR | MSE | NCC | SHA |
|---|------------------|--------------|-----------|----------|----------|-------|
| 0 | airplane.tiff | splash.tiff | 59.534156 | 0.072388 | 0.792096 | False |
| 1 | Barbara.tif | splash.tiff | 61.207613 | 0.049240 | 0.836295 | False |
| 2 | elaine.tiff | splash.tiff | 61.149793 | 0.049900 | 0.857912 | False |
| 3 | fishingboat.tiff | splash.tiff | 60.747986 | 0.054737 | 0.810501 | False |
| 4 | Goldhill.tif | splash.tiff | 61.512825 | 0.045898 | 0.858150 | False |
| 5 | house.tiff | splash.tiff | 60.280163 | 0.060963 | 0.811196 | False |
| 6 | lena.tiff | splash.tiff | 60.402531 | 0.059269 | 0.852687 | False |
| 7 | peeper.tiff | splash.tiff | 61.354405 | 0.047604 | 0.856223 | False |
| 8 | sailboat.tiff | splash.tiff | 61.061715 | 0.050922 | 0.838023 | False |
| 9 | tank.tiff | splash.tiff | 61.377085 | 0.047356 | 0.816531 | False |

In [21]: data1.to_csv(r'C:\Study Meterial\Semester Project\Practical\WORK\Attack\Image Sharnpening\Image Sharnpening.csv', index=False)
data2.to_csv(r'C:\Study Meterial\Semester Project\Practical\WORK\Attack\Image Sharnpening\EPR ncc.csv', index=False)