IMAGE WATERMARKING

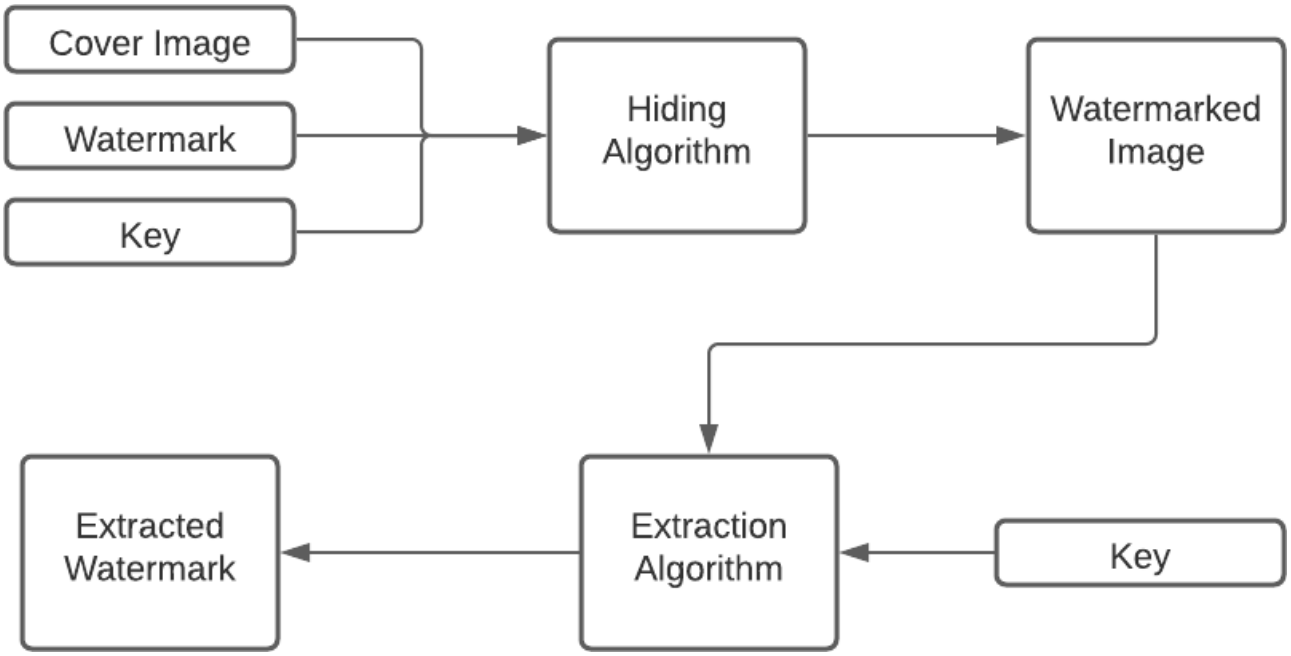
INDIAN INSTITUTE OF TECHNOLOGY, BHUBANESHWAR

INTRODUCTION :

With the rapid development of Internet and information technology, the problem of unauthorized acquisition, transmission, manipulation and distribution of digital content has become increasingly more severe in recent years. The research on information security has attracted considerable attention

Watermarking is a technique for labeling digital pictures by hiding secret information into the images. Sophisticated watermark embedding is a potential method to discourage unauthorized copying or attest the origin of the images.

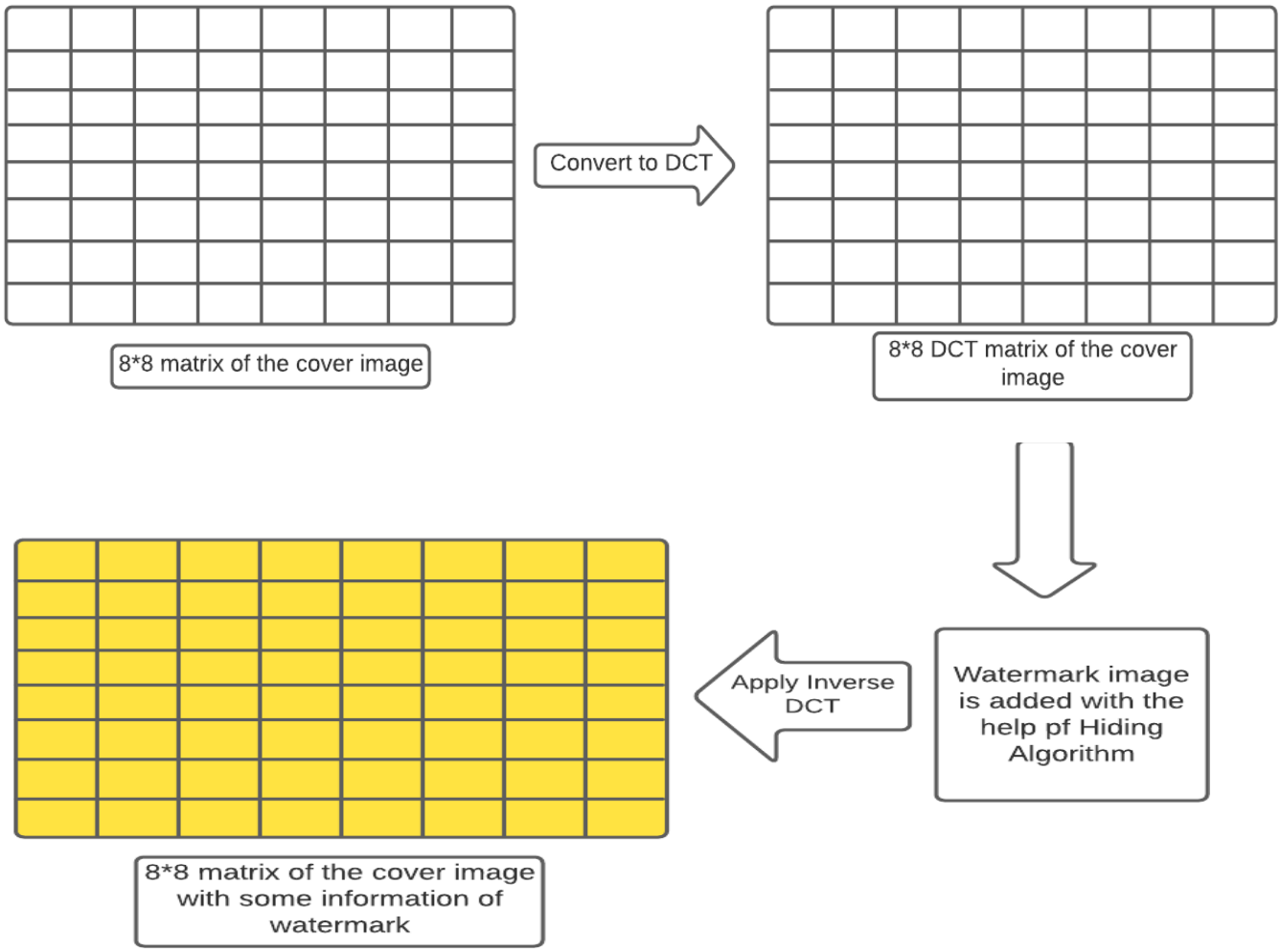
Basic Watermarking model:



DCT(Discrete Cosine Transform) based algorithm:

DCT stands for Discrete Cosine Transform. It is a type of fast computing Fourier transform which maps real signals to corresponding values in the frequency domain. DCT just works on the real part of the complex signal because most of the real-world signals are real signals with no complex components. We will discuss the implementation of DCT Algorithm on Image Data here and the potential uses of the same.

First we take 8*8 matrix of our cover image, then we convert it to DCT matrix, then embed some part of the watermark into it, again we convert the matrix to its inverse DCT and store it back to our image.



DCT algorithm implementation in Python:

In this program, I have mainly designed 2 functions: Hide_watermark() and extract_watermark().
The Hide_watermark() function takes 3 arguments -

1. The cover image file
2. The watermark that needed to be hide
3. The secret key

In output it gives us the watermarked image where the watermark is stored with the help of the secret key, details mentioned below in the Hide_watermark function.

The extract_watermark() function takes mainly 5 arguments -

1. The watermarked image that contain the watermark
2. The secret key
3. Dimensions of the cover image
4. Dimensions of the Watermark
5. Integer value that indicates the end of the watermark

In output it gives us the watermark that was stored in the watermarked image. If the key entered is the same as used in encoding, it will give the correct message, else it will generate an error message and tell you to enter the correct key.

At last we calculate the PSNR value and NC coefficients of the images and also try some of the Attack on the images.

Hide_watermark() Function:

Arguments passed:

1. Cover Image
2. Secret key
3. Watermark

We know that our cover image has 3 channels, first we take only one channel of the image and convert it into an numpy array. Then find its number of rows and columns.

Similarly convert the watermark into a numpy array and divide all the values by 255 and we find its number of rows and columns. Then we convert the 2D numpy array to 1D array by using the ravel() function.

Now we are taking 8*8 matrix part from the numpy array of the cover image and find its DCT by inbuilt DCT function. Now we have to store the value at each index of the watermark 1D array to the DCT matrix. To find the index where to store the value we use our secret KEY.

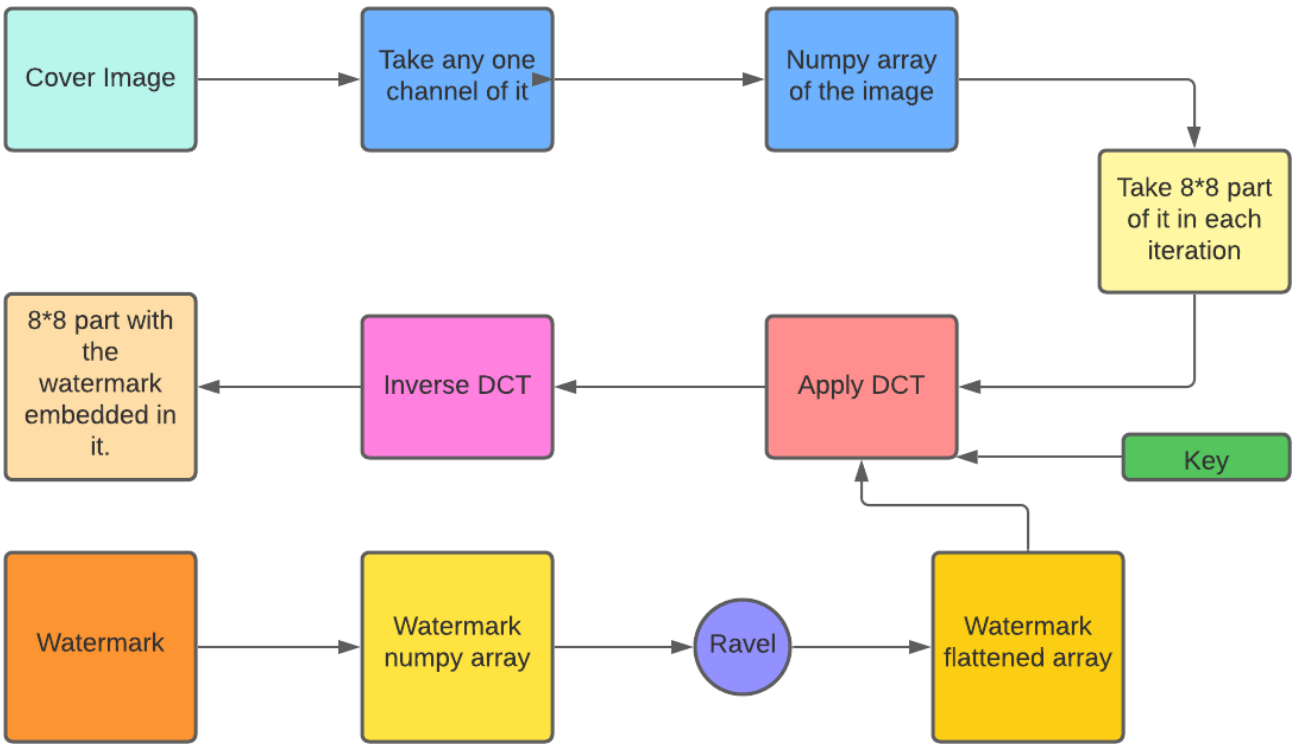
First we pass each character of the secret key into our generate_value() function that first converts it to its ASCII value then to its Binary, then it divides it into 2 parts 4 bits each and convert back them to 2 integers and find its mod 8. So that our value will be between 0 - 7. These values are added with the indexes of the DCT matrix to store the value of the watermark 1D array to the DCT matrix. Next we find its inverse DCT and store back the values to get our Watermarked image. The other 2 channels of the cover image remain as it is in the watermarked image.

```python
def hide_watermark(coverimage,watermark,key):
  wimage = watermark
  cimage=coverimage[:,:,1]
  cimage_array=np.array(cimage,dtype=float).reshape((cimage.shape[0],cimage.shape[1]))
  length=int(cimage_array.shape[0]/8) * 8
  width=int(cimage_array.shape[1]/8) * 8

  wimage_array=np.array(wimage,dtype=float).reshape((wimage.shape[0],wimage.shape[1],3))
  wimage_array=wimage_array/255
  wimage_flat=wimage_array.ravel()
  x=0
  y=0
  for i in range(0,length,8):
    for j in range(0,width,8):
      if(x<len(wimage_flat)):
        cimage_array[i:i+8,j:j+8]=dct(dct(cimage_array[i:i+8,j:j+8].T, norm="ortho").T, norm="ortho")
        s,t=generate_value(key[y%len(key)])
        y=y+1
        cimage_array[i+s][j+t]=wimage_flat[x]
        x=x+1
        cimage_array[i:i+8,j:j+8]=idct(idct(cimage_array[i:i+8,j:j+8].T, norm="ortho").T, norm="ortho")
      else:
        break

  embeded_img=np.zeros((cimage_array.shape[0],cimage_array.shape[1],3))
  embeded_img[:,:,1]=cimage_array
  embeded_img[:,:,0]=coverimage[:,:,0]
  embeded_img[:,:,2]=coverimage[:,:,2]
  watermarked_image=embeded_img
  print("<<---------------Watermark is successfully Hidden--------------->>\n")
  return watermarked_image, length,width,len(wimage_flat)
```

Flow chart



Extract_watermark() Function:

It has 2 primary arguments, the watermarked image and the secret key. First we select the same channel of the watermarked image, where we have stored our watermark.

Then we take an empty array to store the extracted watermark.

Then convert it to Numpy array, and take 8*8 matrix of it in each iteration and find its DCT.

Then as the previous logic of finding the indexes from the given secret key we find the indexes and add it to the array index, and store the value at that position in the empty array.

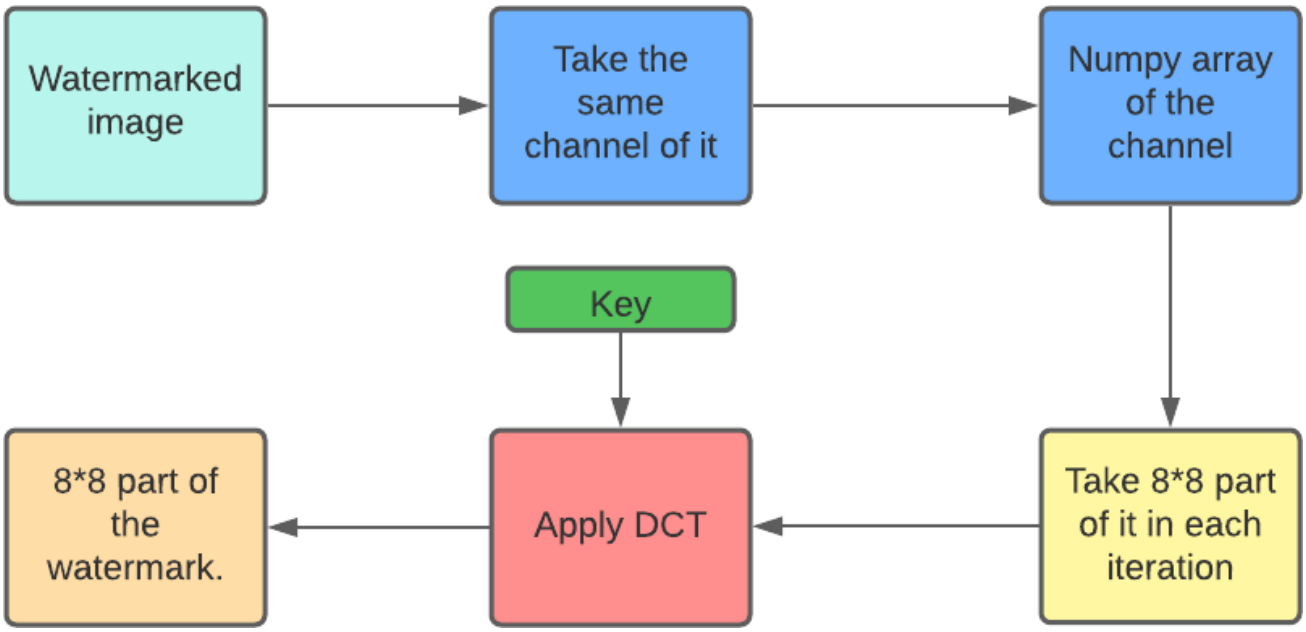
Then we multiply each of the values by 255 as while hiding we have divided it by 255

Then we reshape it to the dimensions of our main watermark and return it.

```
def extract_watermark(watermarked_image,key,length,width,len_w_flat,w_length,w_width):
  extracted_watermark=[]
  x=0
  y=0
  nwimage=watermarked_image[:,:,1]
  for i in range(0,length,8):
    for j in range(0,width,8):
      if(x<len_w_flat):
        nwimage[i:i+8,j:j+8]=dct(dct(nwimage[i:i+8,j:j+8].T, norm="ortho").T, norm="ortho")
        s,t=generate_value(key[y%len(key)])
        y=y+1
        val=nwimage[i+s][j+t]
        extracted_watermark.append(val)
        x=x+1
      else:
        break

  #extracted_watermark=np.array(extracted_watermark,dtype=float).reshape((wimage.shape[0],wimage.shape[1],3))
  extracted_watermark=np.array(extracted_watermark,dtype=float).reshape(w_length,w_width,3)
  extracted_watermark=extracted_watermark*255
  print("<<---------------Watermark is Successfully Extracted--------------->>\n")
  return extracted_watermark
```

Flow Chart



Attacks:

We have implemented 4 attacks on our Watermarked image to see how robust our algorithm is.

1. Gaussian attack
2. Salt-N-pepper attack
3. Speckle attack
4. Rotate-90 attack

For each attack we have extracted the watermark with the help of extract_watermark function and then we also calculated the PSNR values and the NC coefficients.
The details of all these are shown below.

1. After Implementing Gaussian attack:

   PSNR value: 68.13
   NC coefficient value: 0.700

```python
def attackGauss(image):
  row,col,ch= image.shape
  mean = 0
  var = 0.01
  sigma = var**0.5

  gauss = np.random.normal(mean,sigma,(row,col,ch))
  gauss = gauss.reshape(row,col,ch)
  gausianimage = image + gauss
  return gausianimage
```

```
After Gaussian attack the watermarked image is:
```

```
PSNR value after gaussian attack is:  68.13128011426375

The extracted watermark is
```

```
NC value of the extracted logo = [[0.7001881]]
```

2. After Implementing Salt-N-Pepper-Attack:

PSNR value: 35.54
NC coefficient value: 0.709

```
def attackSNP(image):
  row,col,ch = image.shape
  s_vs_p = 0.5
  amount = 0.01
  Snpimage = np.copy(image)

  # Salt mode
  num_salt = np.ceil(amount * image.size * s_vs_p)
  coords = [np.random.randint(0, i - 1, int(num_salt))
          for i in image.shape]
  Snpimage[coords] = 1
```

```
Salt-N-Pepper Attack Results :

After Salt-N-Pepper Attack the watermarked image is:
```

```
PSNR value after salt-N-pepper attack is:  34.54850010345182

The extracted watermark is:
```

```
NC value of the extracted logo = [[0.70991004]]
```

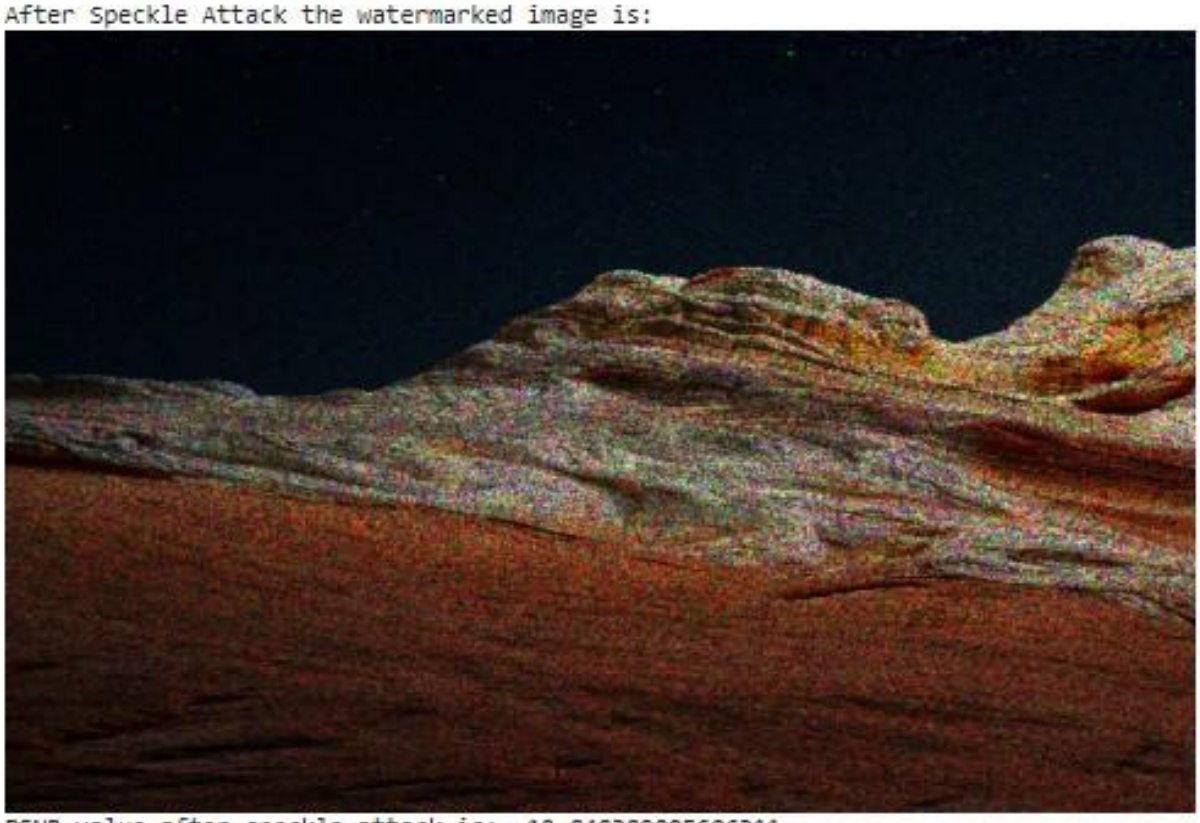
3. After Implementing Speckle Attack:

PSNR value: 12.84
NC coefficient value: 0.706

```
def attackSpeckle(image):
  row,col,ch = image.shape
  gauss = np.random.randn(row,col,ch)
  gauss = gauss.reshape(row,col,ch)
  Speckleimage = image + image * gauss
  return Speckleimage
```

```
Speckle Attack Results :

After Speckle Attack the watermarked image is:
```



```
PSNR value after speckle attack is:  12.848389005606311

The extracted watermark is
```

```
NC value of the extracted logo = [[0.7060344]]
```

4. After Implementing Rotate-90 Attack:

PSNR value: 13.48
NC coefficient value: 0.706

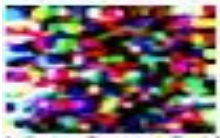
```
def attackRotate90(image):
  angle = 90
  scale = 1.0
  w = image.shape[1]
  h = image.shape[0]
  rangle = np.deg2rad(angle)  # angle in radians
  nw = (abs(np.sin(rangle) * h) + abs(np.cos(rangle) * w)) * scale
  nh = (abs(np.cos(rangle) * h) + abs(np.sin(rangle) * w)) * scale
  rot_mat = cv2.getRotationMatrix2D((nw * 0.5, nh * 0.5), angle, scale)
  rot_move = np.dot(rot_mat, np.array(
      [(nw - w) * 0.5, (nh - h) * 0.5, 0]))
  rot_mat[0, 2] += rot_move[0]
  rot_mat[1, 2] += rot_move[1]
  rotatedimage=cv2.warpAffine(image, rot_mat, (int(math.ceil(nw)), int(math.ceil(nh))), flags=cv2.INTER_LANCZOS4)
  return rotatedimage
```

```
Rotate-90 Attack Results :

After 90 degree rotation attack the watermarked image is:
```

```
PSNR value after rotate90 attack is:  13.484729494653141

The extracted watermark is
```



```
NC value of the extracted logo = [[0.7068904]]
```

Results:

The results before the attack are shown below.

PSNR Value: 68.131

Original Image:

Watermarked Image:

Original Watermark and Extracted watermark:

NC Coefficient: 0.999

The original watermark is

The extracted watermark is

NC value of the extracted logo = [[0.9999961]]

Google Colab Link:
https://colab.research.google.com/drive/1cC8bRpqIWAYGQLZtiBoEgsrKd_8Kl2N_?usp=sharing