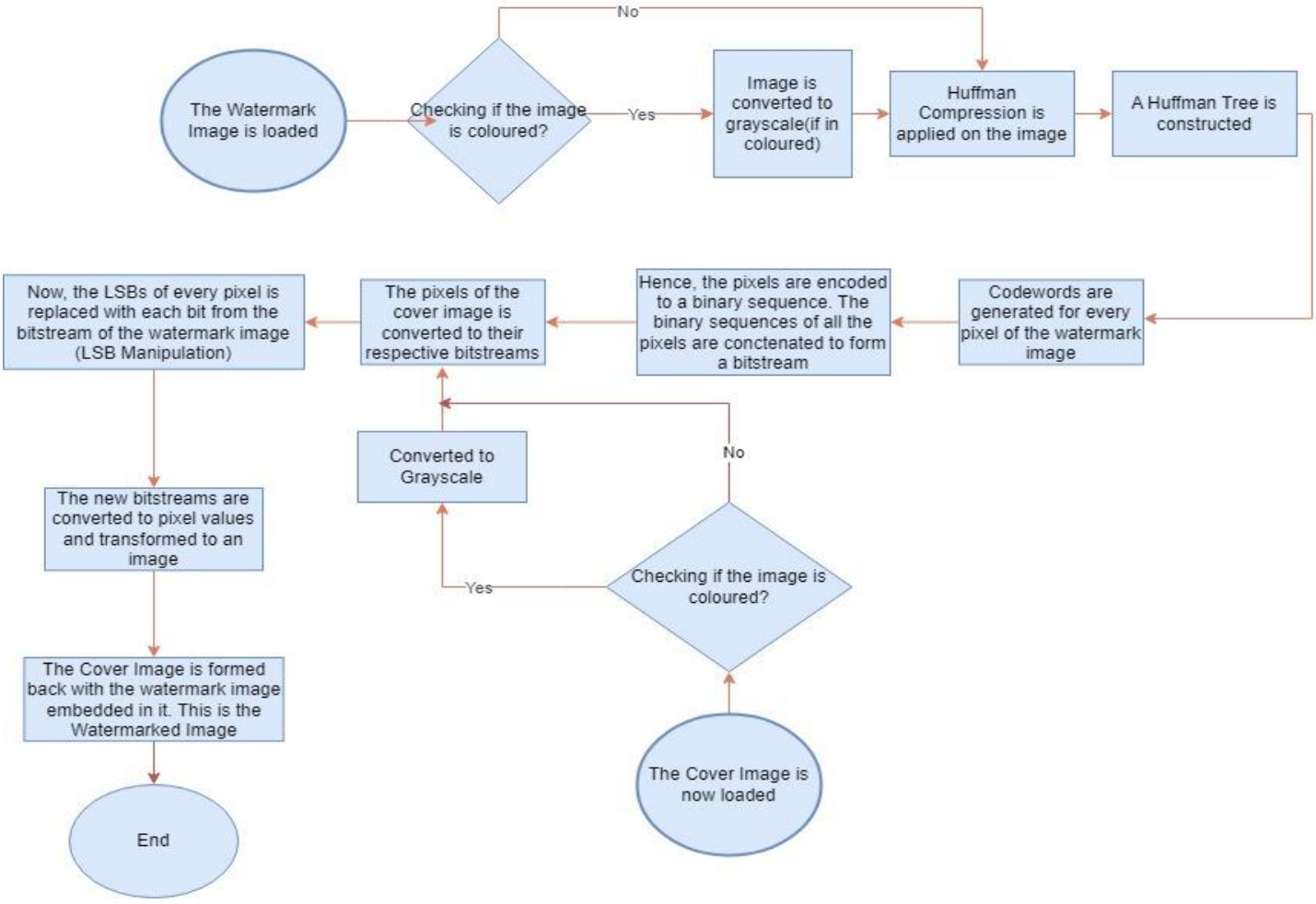
1. Proposed Methodology:

As discussed about the various domains of Digital Watermarking, the process followed in this project involves the manipulation of the Least Significant Bits of the Cover Image that happens to fall in the category of Spatial Domain Watermarking Technique. The code used for the sake, is developed and implemented in Python Programming Language of version 3.12.1.

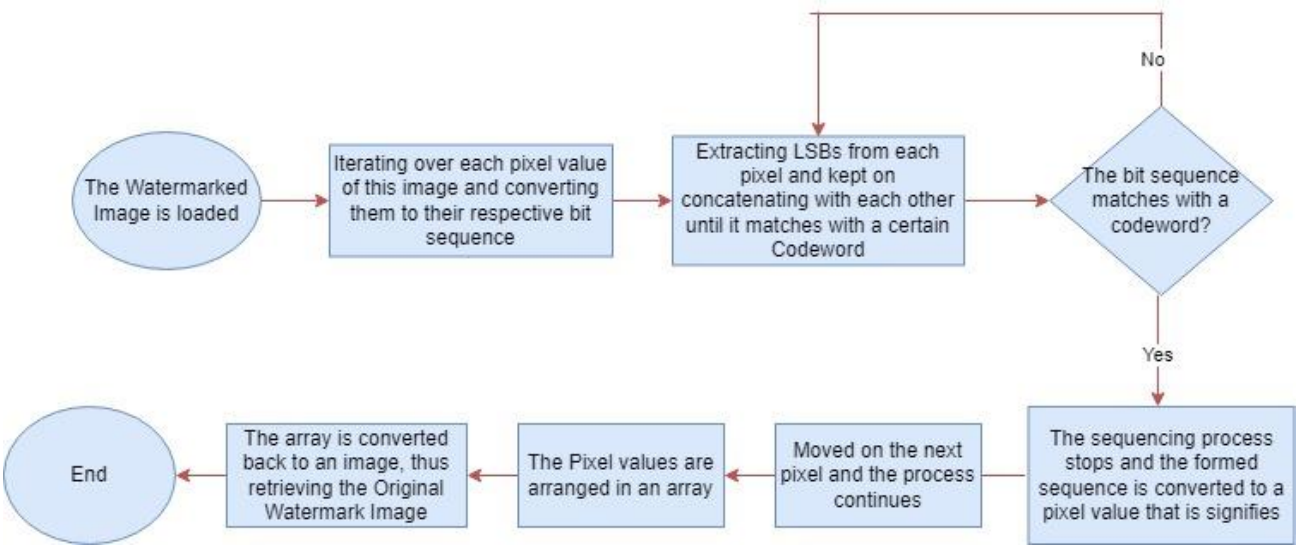
Here, the concept of Huffman Compression is vigorously used in compressing and encoding of the watermark image to ascertain that it retains its identity and original data while getting embedded into the cover image. On the other side, in the embedding process, the LSB operation ensures that the visuality of the cover image remains intact even after watermark embedding.

The proceedings are pictorially represented with the help of the following flowcharts for better understanding, and thereafter, discussed in detail.

Embedding of the Watermark



Extraction of the Watermark



A. Huffman Coding

Huffman Coding is primarily a lossless data compression technique. In the context of digital watermarking, it can be shrewdly implemented to ensure that the watermark is compressed to occupy minimal space within the cover image without losing any of its data. In this technique, the ultimate goal is to form shorter codes for more frequent symbols and longer codes for less frequent symbols of the watermark image. These codes are actually called the Codewords.

The following are the steps of Huffman Coding/Huffman Compression:

1. **Image Preparation**

   i. Reading the Image: The Watermark image is loaded into a matrix of pixel values.

   ii. Converting to Grayscale: If the image is in colour, it is converted to grayscale to simplify the process (each pixel will have a single intensity value instead of three RGB values).

2. **Calculating Pixel Frequency**

   i. The frequency of each pixel value (0-255) in the grayscale image is counted.

3. Huffman Tree Construction:

i. From the frequency, as calculated above, we evaluate the probability of occurrence of pixel by this simple formula –

$p_i = f_i/f$
$p_i \rightarrow$ Probability of occurrence of i^{th} element
$f_i \rightarrow$ Frequency of occurrence of i^{th} element
$f \rightarrow$ Total frequency of all the elements

ii. The symbols or the pixels are arranged vertically in decreasing order of their probabilities.

iii. The bottom two symbols are taken and joined together and their respective probabilities are added and written on the combined node.

iv. The two branches, thus formed, are assigned '0' & '1' with the former being for the lowest one the later for the upper one.

v. The sum of probabilities of the two symbols as obtained above is now treated as new probability for a new symbol. This process is repeated till we reach the top of the vertical column, i.e. the symbol with the maximum probability.

vi. The Huffman Tree gets constructed in this manner.

4. Generating Codewords

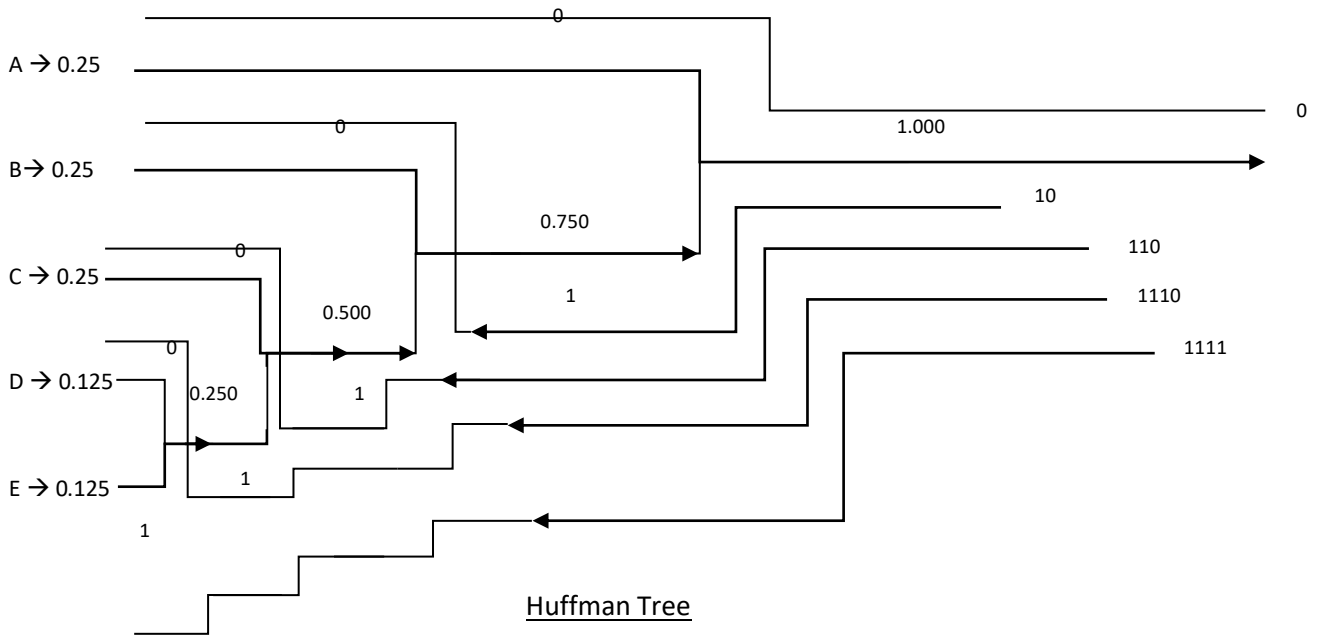
i. To get the codewords, the path is traced back from the final node to the symbol along the branches.

ii. The labels on the branches, along the way, are collected and put together to form a codeword.

Given is an example for a better understanding:

Symbols	Frequency	Probability
A	16	0.250
B	16	0.250
C	16	0.250
D	8	0.125
E	8	0.125

Symbols	CodeWords
A	0
B	10
C	110
D	1110
E	1111



Huffman Tree

5. Encoding the Image

i. Replacing Pixel Values with Huffman Codes:

- A mapping of pixel values is formed to their Huffman codes.
- Each pixel value in the image is replaced with its corresponding Huffman code.

6. Creating Bitstream

- All the Huffman codes are concatenated to form a continuous bitstream.

7. Store or Transmit the Encoded Data

- Storing the Huffman Tree: The Huffman tree or its equivalent mapping needs to be stored or transmitted along with the encoded bitstream to facilitate decoding.

- Storing Encoded Bitstream: The encoded bitstream, representing the image, is saved or transmitted.

B. Embedding Process:

i. Image Preparation

i. Reading the Cover Image: The Cover Image is loaded as input.

ii. Converting to Grayscale: For simplicity, the cover image is converted to grayscale image if it is in colour.

ii. LSB Replacement

i. Converting the Cover Image to an Array: The Cover Image is converted into a numpy array of its pixel values, using the Python Language.

ii. LSB Operation: The LSBs of every pixel of the cover image is now replaced with each bit from the bitstream obtained from Huffman Coding in the previous process.

iii. Forming the final watermarked image

i. The array is now converted back into an image to finally get back the Cover Image with the Watermark Image embedded inside it.

C. Extraction Process:

1. The Watermarked Image is loaded as input.

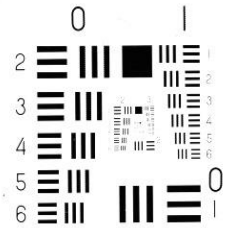
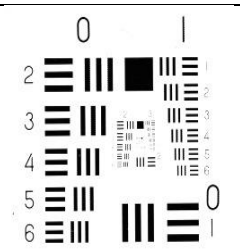
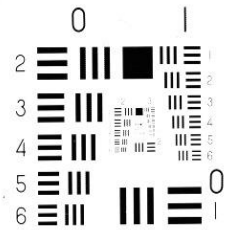
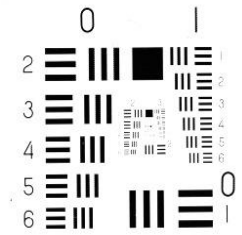
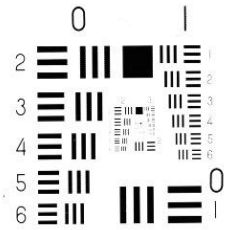
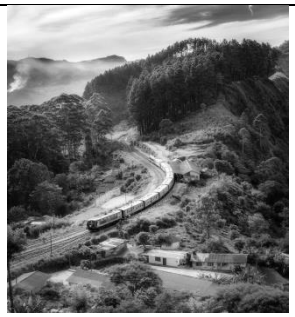
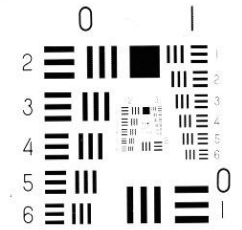
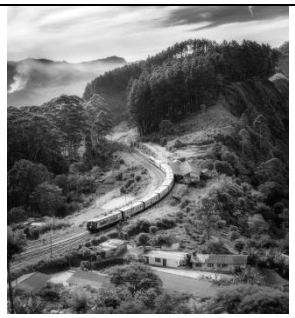
2. The LSBs of each pixel values of this image is extracted one by one and concatenated with each other to keep forming a bitstream.

3. This formation of a bitstream is continued until it matches with a particular codeword.

4. Once a match occurs, that bitstream is converted or decoded to the original pixel value which it represents. It is a simple Binary to Decimal operation.

5. The above process continues throughout the whole watermarked image. Ultimately, the decoded pixel values are arranged back into the watermark image.

Table of some results:

Sl. No.	Cover Image	Watermark Image	Watermarked Image	Retrieved Watermark Image
1.				
2.				
3.				
4.				
5.