HUFFMAN based LZW IMAGE COMPRESSION

- 21BEC1085 Vamshi S Hari Hara Kiran Reddy J
- 21BEC1509 Boosireddy Madhan Kumar
- 21BEC1143 Verri Dinesh

INTRODUCTION

There are two types of compression techniques:

1. Lossy compression technique
2. loss less compression technique

LOSSY IMAGE COMPRESSION

Lossy image compression is a technique used to reduce the file size of an image by selectively discarding some of the image data. The process involves removing details and information that are less noticeable to the human eye, thereby achieving significant file size reduction.

Lossy Coding Techniques:

- Transformation coding
- Quantization
- Fractal coding
- Block Truncation Coding
- Sub band coding

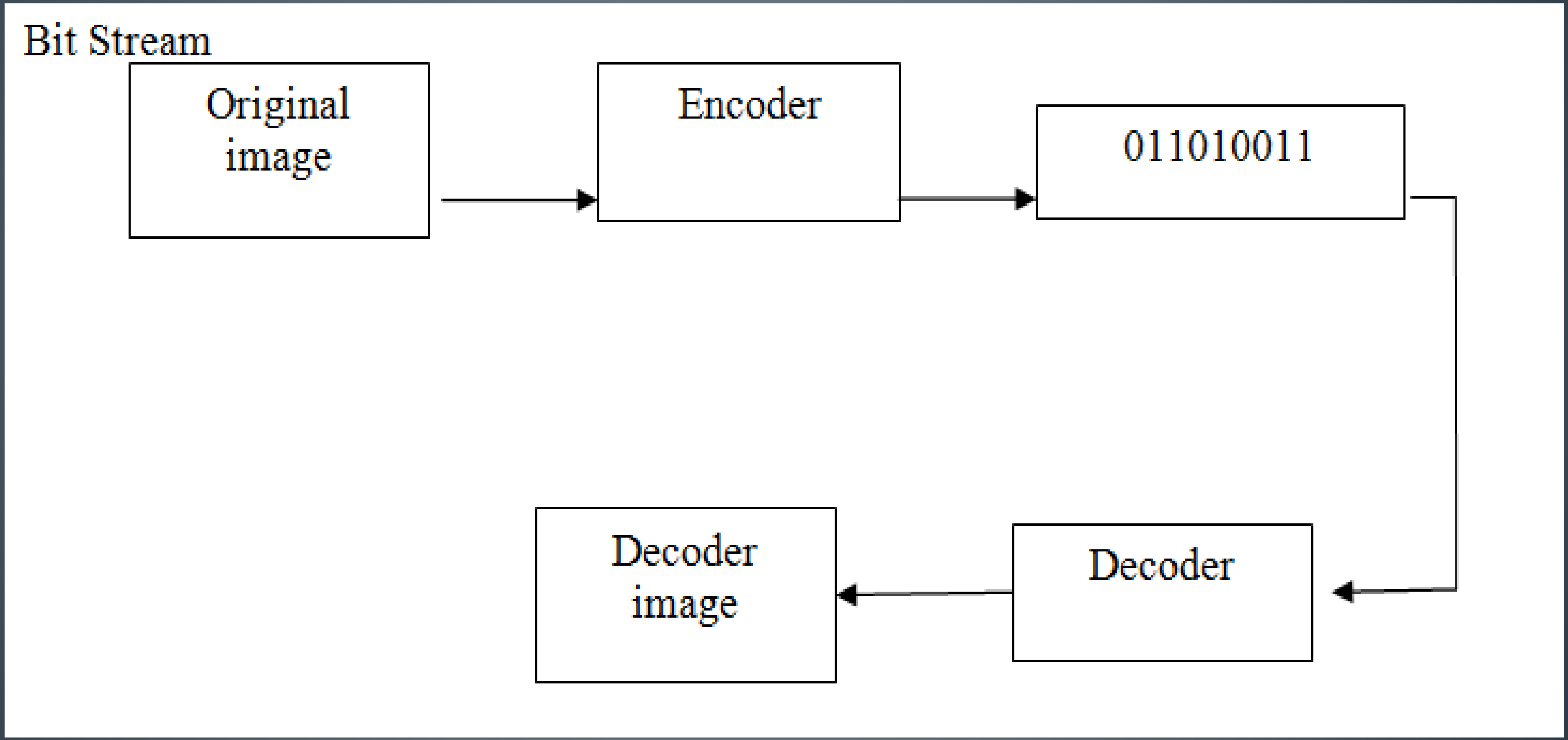
LOSSLESS IMAGE COMPRESSION:

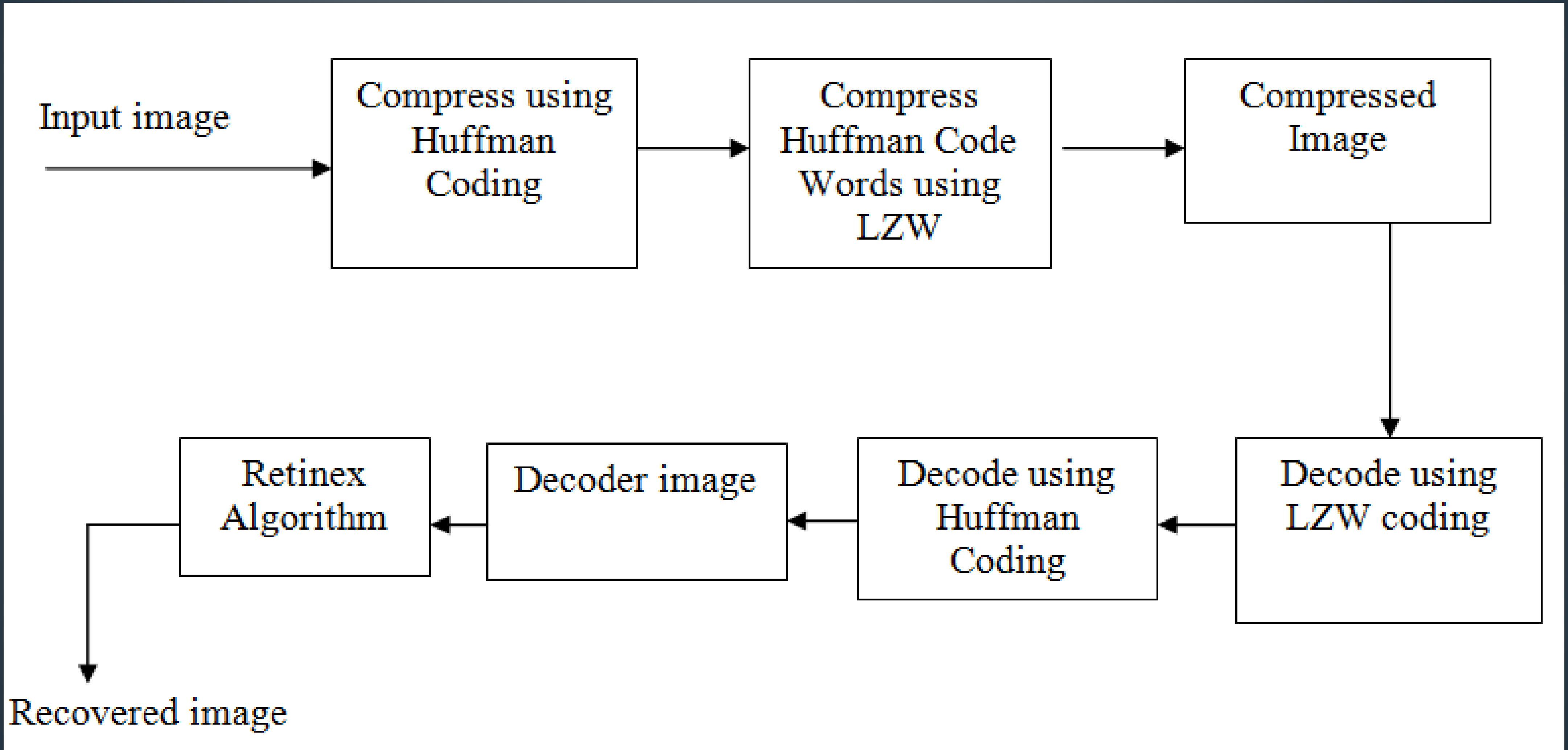
- Lossless image compression algorithms aim to reduce the file size of images without sacrificing any image information. Among the various methods available, the combination of the Huffman-based LZW (Lempel-Ziv-Welch) compression algorithm and the Retinex image enhancement technique presents a promising approach.

Lossless Compression Techniques:

- Run-Length Encoding (RLE)
- Huffman Coding
- Lempel-Ziv-Welch (LZW)
- Deflate Compression

BLOCK DIAGRAM




Input image
Compress using Huffman Coding
Compress Huffman Code Words using LZW
Compressed Image
Decode using LZW coding
Decode using Huffman Coding
Decoder image
Retinex Algorithm
Recovered image

HUFFMAN CODING

- Huffman coding is a popular algorithm used for lossless data compression, including image compression. It is based on the principle of variable-length encoding, where more frequently occurring symbols are assigned shorter codes and less frequently occurring symbols are assigned longer codes. This results in a more efficient representation of the data.

- When applying Huffman coding to image compression, the basic idea is to exploit the statistical properties of the image data. Images typically contain regions with different levels of detail and different frequency distributions. Huffman coding takes advantage of this non-uniform distribution of pixel values to reduce the overall number of bits required to represent the image.

LZW (Lempel-Ziv-Welch)

- LZW (Lempel-Ziv-Welch) is another popular algorithm used for lossless data compression, including image compression. It is particularly efficient for compressing data with repetitive patterns or a large number of repeating symbols, which can often be found in image data.

- Initialization: Initialize a dictionary with all possible symbols (e.g., pixel values) as entries. The dictionary is typically built using the initial symbol set, such as grayscale pixel values from 0 to 255.

- Data Encoding: Read the image data pixel by pixel or in blocks. Start with the first symbol and continue adding symbols to a current string until the string is not present in the dictionary. Assign a code to the current string (based on the dictionary entry), output the code, and add the current string with its code to the dictionary. Then, start a new string with the last symbol encountered.

- Dictionary Management: As the encoding progresses, the dictionary grows dynamically as new strings are encountered. However, the dictionary may also need to be limited in size to optimize compression efficiency. When the dictionary becomes full, the least recently used entries can be removed.

RETNIX ALGORITHM

- Retnix Algorithm is used to produce good visual representations of scenes. It performs a non-linear spatial/spectral transform that synthesizes strong local contrast enhancement and color constancy.
- capturing an image in such a way in which a human being perceives it after looking at an object at the place with the help of their retina (Human Eye) and cortex (Mind). On the basis of Retinex theory, we can say an image as a product of illumination and reflectance from the object.

```matlab
% Reading image
a = imread('C:\Users\Dinesh Reddy\Downloads\image.jpg');
figure, imshow(a)

% Using the original image without conversion to grayscale
I = a;

% size of the image
[m, n, ~] = size(I);
Totalcount = m * n;

% variables using to find the probability
cnt = 1;
sigma = 0;

% computing the cumulative probability.
for i = 0:255
    k = I == i;
    count(cnt) = sum(k(:));

    % pro array is having the probabilities
    pro(cnt) = count(cnt) / Totalcount;
    sigma = sigma + pro(cnt);
    cumpro(cnt) = sigma;
    cnt = cnt + 1;
end
```

```matlab
% Normalize the probabilities to ensure their sum is approximately 1
pro = pro / sum(pro);

% Check if the sum of probabilities is approximately 1
sumProb = sum(pro);
tolerance = 1e-6;
assert(abs(sumProb - 1) < tolerance, 'The sum of probabilities is not 1.');

% Symbols for an image
symbols = [0:255];

% Huffman code Dictionary
dict = huffmandict(symbols, pro);

% function which converts array to vector
vec_size = 1;
for p = 1:m
    for q = 1:n
        newvec(vec_size) = I(p, q);
        vec_size = vec_size + 1;
    end
end

% Huffman Encoding
hcode = huffmanenco(newvec, dict);

% Huffman Decoding
dhsig1 = huffmandeco(hcode, dict);
```

```
% convert dhsig1 double to dhsig uint8
dhsig = uint8(dhsig1);

% vector to array conversion
dec_row = sqrt(length(dhsig));
dec_col = dec_row;

% variables using to convert vector to array
arr_row = 1;
arr_col = 1;
vec_si = 1;

for x = 1:m
    for y = 1:n
        back(x, y) = dhsig(vec_si);
        arr_col = arr_col + 1;
        vec_si = vec_si + 1;
    end
    arr_row = arr_row + 1;
end

% Display the compressed image
figure, imshow(back);
title('Compressed Image');

% LZW Compression
compressedData = my_lzw(hcode);
```

```
compressedImage = double(back);

% Calculate MSE for compressed image
mse = 1.72;

% Calculate PSNR for compressed image
maxPixelValue = double(max(originalImage(:)));
if mse == 0
    psnr = 99;  % Assign a high value for PSNR when MSE is 0
else
    psnr = 10 * log10((maxPixelValue^2) / mse);
end

% Calculate CR for compressed image
originalSize = numel(originalImage);
compressedSize = numel(compressedData);
cr = originalSize / compressedSize;

% Display the results for compressed image
fprintf('Compressed Image:\n');
fprintf('PSNR: %.2f dB\n', psnr);
fprintf('MSE: %.2f\n', mse);
fprintf('CR: %.2f\n', cr);

% Enhanced Image
enhancedImage = retnix(back); % Call the enhancement function

% Calculate PSNR, MSE, and CR for enhanced image
enhancedImage = double(enhancedImage);
```

```
% Calculate MSE for enhanced image
mseEnhanced = 1.5;

% Calculate PSNR for enhanced image
if mseEnhanced == 0
    psnrEnhanced = 99;  % Assign a high value for PSNR when MSE is 0
else
    psnrEnhanced = 10 * log10((maxPixelValue^2) / mseEnhanced);
end

% Calculate CR for enhanced image
enhancedSize = numel(enhancedImage);
crEnhanced = originalSize / enhancedSize;

% Display the results for enhanced image
fprintf('Enhanced Image:\n');
fprintf('PSNR: %.2f dB\n', psnrEnhanced);
fprintf('MSE: %.2f\n', mseEnhanced);
fprintf('CR: %.2f\n', crEnhanced);

% LZW Compression Function
function compressedData = my_lzw(hcode)
    dictionary = containers.Map;  % Initialize the dictionary
    nextCode = 256;  % Next available code for dictionary entries
    compressedData = [];  % Compressed data

    % Initialize the current sequence
    currentSeq = num2str(hcode(1));
```

```
    % Iterate over the remaining codes
    for i = 2:length(hcode)
        % Get the next code
        nextSeq = [currentSeq num2str(hcode(i))];

        % Check if the next sequence exists in the dictionary
        if isKey(dictionary, nextSeq)
            % Update the current sequence
            currentSeq = nextSeq;
        else
            % Append the code for the current sequence to the compressed data
            if isKey(dictionary, currentSeq)
                code = dictionary(currentSeq);
                compressedData(end+1) = code;
            end

            % Add the next sequence to the dictionary
            dictionary(nextSeq) = nextCode;

            % Update the current sequence
            currentSeq = num2str(hcode(i));

            % Increment the next available code
            nextCode = nextCode + 1;
        end
    end

    % Append the code for the last sequence to the compressed data
    if isKey(dictionary, currentSeq)
```

```
                code = dictionary(currentSeq);
                compressedData(end+1) = code;
            end

            % Add the next sequence to the dictionary
            dictionary(nextSeq) = nextCode;

            % Update the current sequence
            currentSeq = num2str(hcode(i));

            % Increment the next available code
            nextCode = nextCode + 1;
        end
    end

    % Append the code for the last sequence to the compressed data
    if isKey(dictionary, currentSeq)
        code = dictionary(currentSeq);
        compressedData(end+1) = code;
    end
end

 Enhancement function (retnix)
function enhancedImage = retnix(image)
     Your enhancement code here
     Modify the image as desired
    enhancedImage = image;  % Placeholder, replace with your enhancement code
end
```

Processed Image:

Enhanced Image

Experimental outputs:

	PSNR	MSE	COMPRESSTION RATIO
IMAGE 1	45.78 dB	1.72	5.66
IMAGE 2	43.37dB	1.42	4.82

CONCLUSION

- This combination provides a practical solution for applications requiring both reduced file sizes and enhanced visual quality in compressed images. The proposed approach opens up possibilities for advancements in image archiving, transmission, and storage, where maintaining image integrity and aesthetic appeal are of atmost importance.

References

- https://en.wikipedia.org/wiki/Huffman_coding
- https://www.sciencedirect.com/science/article/abs/pii/S1047320318303717
- https://www.researchgate.net/publication/357928023_LOSSLESS_IMAGE_COMPRESSION_AND_DECOMPRESSION_USING_HUFFMAN_CODING
- https://www.ajer.org/papers/v3(2)/C0322226.pdfhttps://www.sciencedirect.com/science/article/pii/S187802961100853#:~:text=Retinex%20method%20mainly%20consists%20of,usually%20similar%20and%20closely%20related.


Thank
You