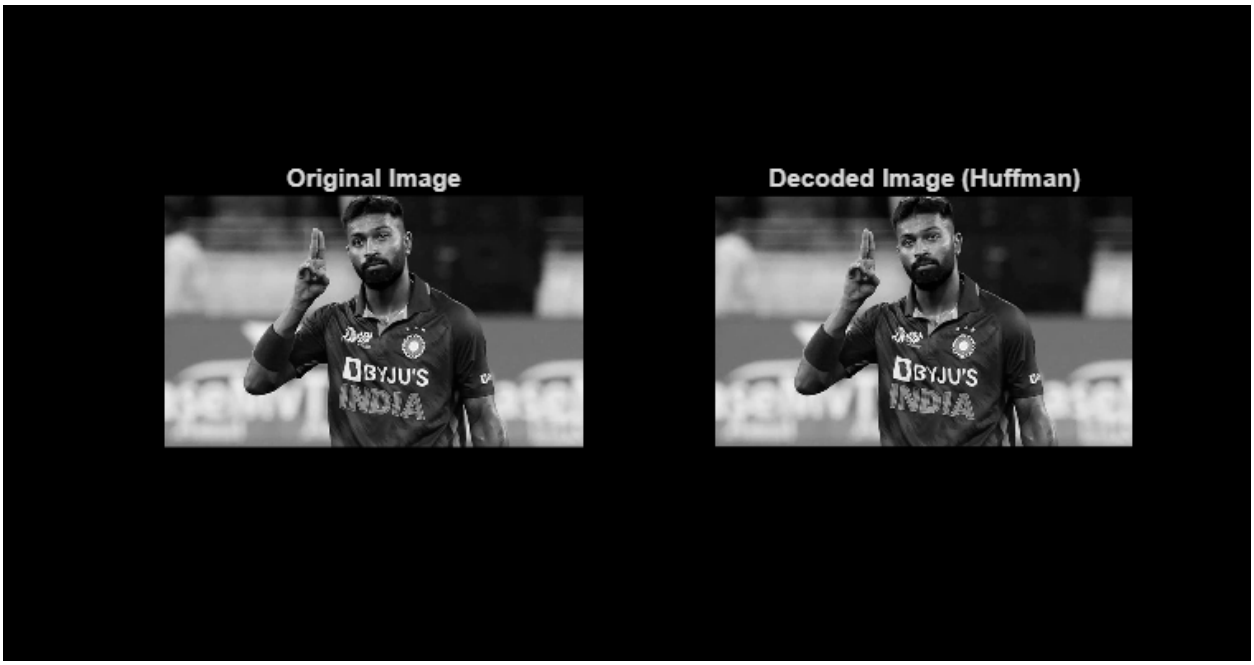
```
clc;
clear;
close all;

% Read grayscale image
img = imread('Hardik-Pandya.jpg');
if size(img,3) == 3
    img = rgb2gray(img);
end
img = uint8(img);

% Flatten image into 1D array
pixels = img(:);

% Calculate symbol frequencies
symbols = unique(pixels);
freq = zeros(length(symbols),1);

for i = 1:length(symbols)
    freq(i) = sum(pixels == symbols(i));
end

prob = freq / sum(freq);

% Build Huffman Tree
nodes = struct('symbol', {}, 'prob', {}, 'left', {}, 'right', {});

for i = 1:length(symbols)
    nodes(i).symbol = symbols(i);
    nodes(i).prob = prob(i);
    nodes(i).left = [];
    nodes(i).right = [];
end

while length(nodes) > 1
    [~, idx] = sort([nodes.prob]);
    nodes = nodes(idx);

    newNode.symbol = [];
    newNode.prob = nodes(1).prob + nodes(2).prob;
    newNode.left = nodes(1);
    newNode.right = nodes(2);

    nodes(1:2) = [];
    nodes(end+1) = newNode;
end

huffmanTree = nodes;

% Generate Huffman codes
codes = containers.Map('KeyType','double','ValueType','char');
codes = generateCodes(huffmanTree, '', codes);
```

```
% Huffman Encode
encoded_bits = '';
for i = 1:length(pixels)
    encoded_bits = [encoded_bits codes(double(pixels(i)))];
end

% Huffman Decode
decoded_pixels = zeros(size(pixels));
currentNode = huffmanTree;
idx = 1;

for i = 1:length(encoded_bits)
    if encoded_bits(i) == '0'
        currentNode = currentNode.left;
    else
        currentNode = currentNode.right;
    end

    if isempty(currentNode.left) && isempty(currentNode.right)
        decoded_pixels(idx) = currentNode.symbol;
        idx = idx + 1;
        currentNode = huffmanTree;
    end
end

decoded_img = reshape(uint8(decoded_pixels), size(img));

% Display images
figure;
subplot(1,2,1);
imshow(img);
title('Original Image');
axis off;

subplot(1,2,2);
imshow(decoded_img);
title('Decoded Image (Huffman)');
axis off;

% Compression statistics
original_size = numel(pixels) * 8;
compressed_size = length(encoded_bits);
compression_ratio = original_size / compressed_size;

fprintf('Original size (bits): %d\n', original_size);
fprintf('Compressed size (bits): %d\n', compressed_size);
fprintf('Compression ratio: %.2f\n', compression_ratio);

function codes = generateCodes(node, code, codes)
    if isempty(node.left) && isempty(node.right)
        if isempty(code)
            code = '0';
        end
```

```
        codes(node.symbol) = code;
        return;
    end
    codes = generateCodes(node.left,  [code '0'], codes);
    codes = generateCodes(node.right, [code '1'], codes);
end
```

```
Original size (bits): 1196000
Compressed size (bits): 1133845
Compression ratio: 1.05
```



Published with MATLAB® R2025b