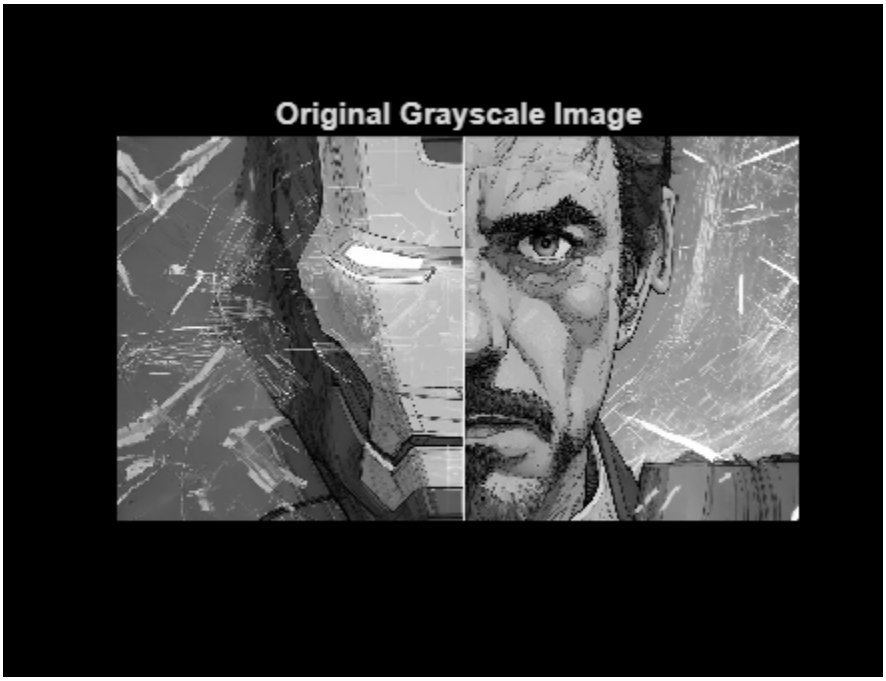
Table of Contents


```
% ============================================================
% Shannon and Huffman Coding with Encoding & Decoding
% MATLAB ONLINE COMPATIBLE | No Toolbox
% ============================================================

clc; clear; close all;
```

Read image

```
rgb_img = imread('input.jpg');
rgb_img = imrotate(rgb_img,90);

gray_img = rgb2gray(rgb_img);
[rows, cols] = size(gray_img);
img_data = gray_img(:);

figure; imshow(gray_img);
title('Original Grayscale Image');

orig_bits = numel(img_data) * 8;
```

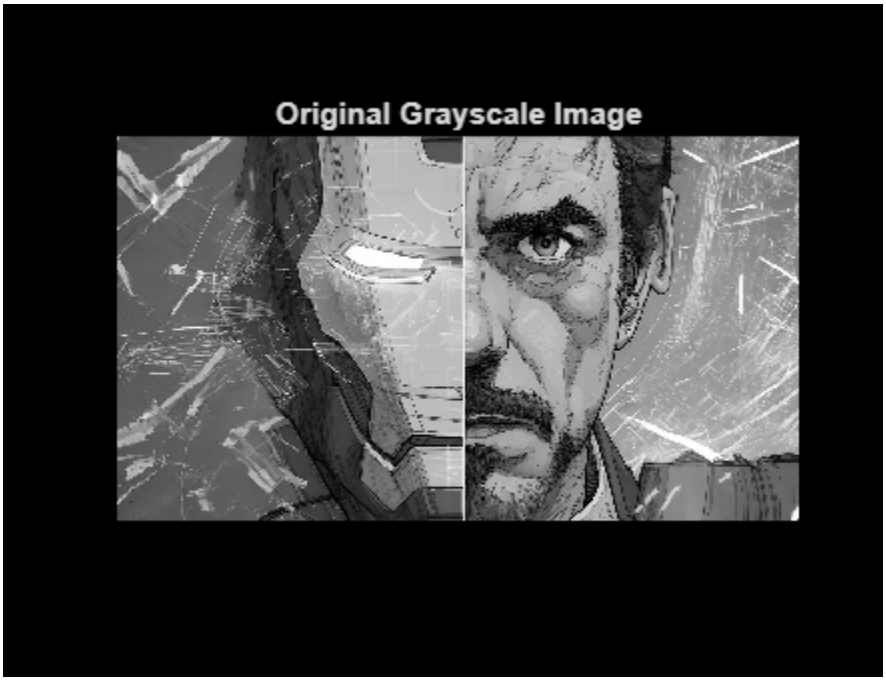


========== SHANNON CODING ==========

```
symbols = unique(img_data);
counts  = histcounts(img_data,[symbols; max(symbols)+1]);
prob    = counts / sum(counts);

[prob_s, idx] = sort(prob,'descend');
symbols_s = symbols(idx);

code_len = ceil(-log2(prob_s));

% Generate Shannon codes
shannon_codes = cell(length(symbols_s),1);
cum_prob = 0;
for i = 1:length(symbols_s)
    shannon_codes{i} = dec2bin( ...
        floor(cum_prob * 2^code_len(i)), code_len(i));
    cum_prob = cum_prob + prob_s(i);
end

% Encode
encoded_shannon = cell(numel(img_data),1);
for i = 1:numel(img_data)
    k = find(symbols_s == img_data(i),1);
    encoded_shannon{i} = shannon_codes{k};
end
shannon_stream = [encoded_shannon{:}];
shannon_bits = length(shannon_stream);
```

```
fprintf('\n--- Shannon Coding ---\n');
fprintf('Original bits   : %d\n', orig_bits);
fprintf('Compressed bits : %d\n', shannon_bits);
fprintf('Compression Ratio: %.3f\n', orig_bits/shannon_bits);
```

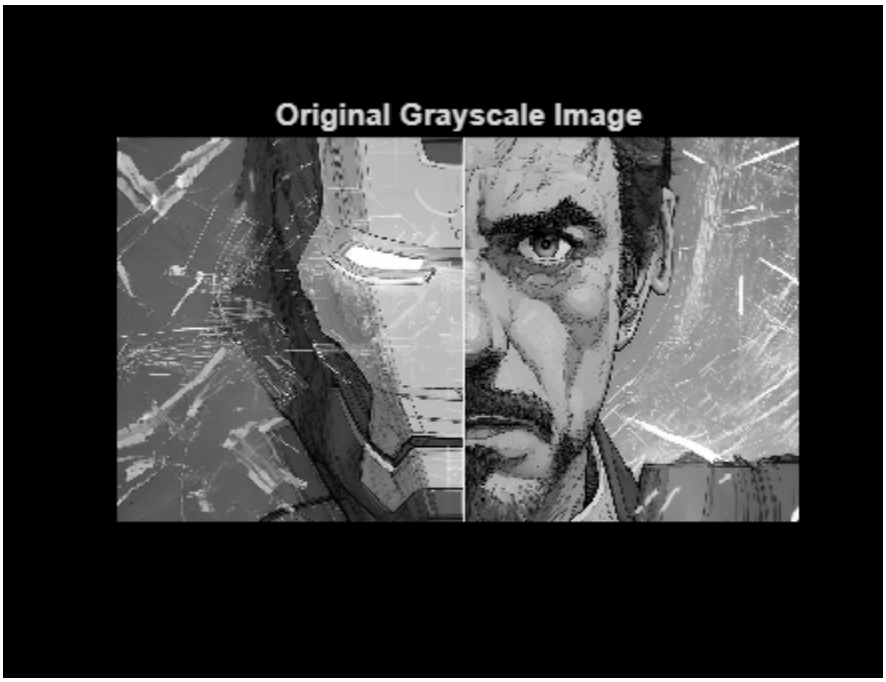
```
--- Shannon Coding ---
Original bits   : 7701504
Compressed bits : 7923467
Compression Ratio: 0.972
```



-------- SHANNON DECODING --------

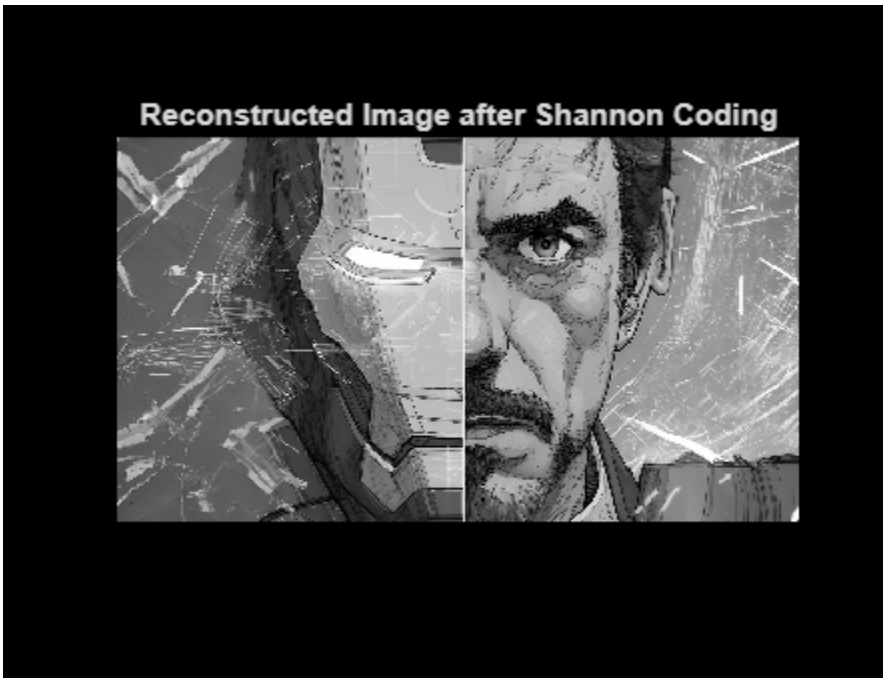
```
decoded_shannon = zeros(numel(img_data),1,'uint8');
bitIdx = 1;
outIdx = 1;

while bitIdx <= length(shannon_stream)
    for k = 1:length(shannon_codes)
        L = length(shannon_codes{k});
        if bitIdx+L-1 <= length(shannon_stream)
            if strcmp(shannon_stream(bitIdx:bitIdx+L-1), shannon_codes{k})
                decoded_shannon(outIdx) = symbols_s(k);
                outIdx = outIdx + 1;
                bitIdx = bitIdx + L;
                break;
            end
        end
    end
end
```

```
decoded_shannon_img = reshape(decoded_shannon, rows, cols);

figure; imshow(decoded_shannon_img);
title('Reconstructed Image after Shannon Coding');
```



================== HUFFMAN CODING ==========================

```
symbols = unique(img_data);
counts  = histcounts(img_data,[symbols; max(symbols)+1]);
prob    = counts / sum(counts);

nodeProb = prob(:);
nodeSym  = num2cell(double(symbols(:)));

% Build Huffman tree
while numel(nodeProb) > 1
    [nodeProb, idx] = sort(nodeProb);
    nodeSym = nodeSym(idx);

    newProb = nodeProb(1) + nodeProb(2);
    newSym  = {nodeSym{1}, nodeSym{2}};

    nodeProb = [newProb; nodeProb(3:end)];
    nodeSym  = [{newSym}; nodeSym(3:end)];
end

huffTree = nodeSym{1};
```



Generate Huffman codes

```
codes = containers.Map('KeyType','double','ValueType','char');
codes = buildCodes(huffTree,'',codes);
```

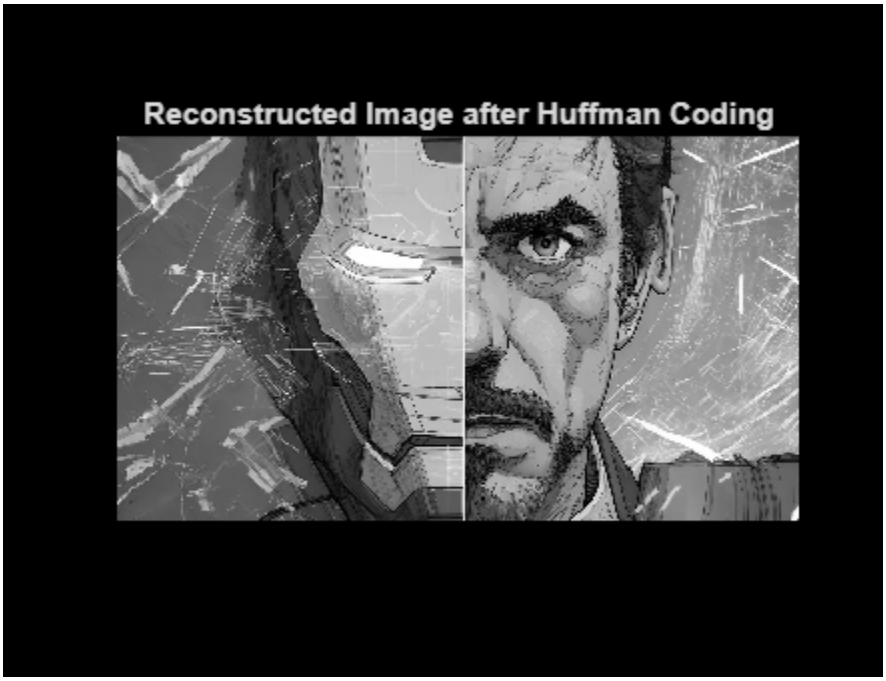
Encode Huffman

```
encoded_huff = cell(numel(img_data),1);
for i = 1:numel(img_data)
    encoded_huff{i} = codes(double(img_data(i)));
end
huff_stream = [encoded_huff{:}];
```

Decode Huffman

```
decoded = zeros(numel(img_data),1,'uint8');
node = huffTree;
bitIdx = 1;
outIdx = 1;

while bitIdx <= length(huff_stream)
    if isnumeric(node)
        decoded(outIdx) = uint8(node);
        outIdx = outIdx + 1;
        node = huffTree;
    else
        if huff_stream(bitIdx) == '0'
            node = node{1};
        else
            node = node{2};
        end
        bitIdx = bitIdx + 1;
    end
end

decoded_huff_img = reshape(decoded, rows, cols);

figure; imshow(decoded_huff_img);
title('Reconstructed Image after Huffman Coding');

huff_bits = length(huff_stream);

fprintf('\n--- Huffman Coding ---\n');
fprintf('Original bits   : %d\n', orig_bits);
fprintf('Compressed bits : %d\n', huff_bits);
fprintf('Compression Ratio: %.3f\n', orig_bits/huff_bits);
```

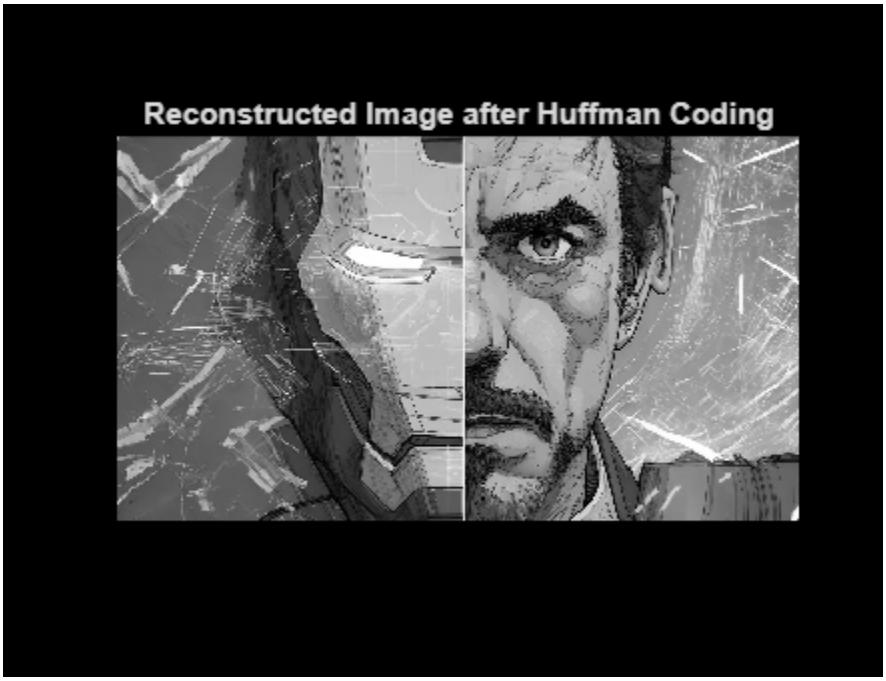
```
--- Huffman Coding ---
Original bits   : 7701504
Compressed bits : 7374582
Compression Ratio: 1.044
```



========== ENTROPY ==========

```
entropy_val = -sum(prob .* log2(prob));
fprintf('\nEntropy = %.4f bits/pixel\n', entropy_val);
fprintf('Theoretical minimum bits = %.0f\n', entropy_val*numel(img_data));
```

```
Entropy = 7.6324 bits/pixel
Theoretical minimum bits = 7347602
```



========== LOCAL FUNCTION ==========

```
function codes = buildCodes(tree, code, codes)
    if isnumeric(tree)
        codes(tree) = code;
    else
        codes = buildCodes(tree{1}, [code '0'], codes);
        codes = buildCodes(tree{2}, [code '1'], codes);
    end
end
```

Published with MATLAB® R2025b