Table of Contents

```
%Pratap Luitel
%ENGS 111
%HW 7

%This script is based on prof Alex Hartov's jpgalg.m.
%Input
%imIn: input image
%quality: for quantization during compression
%Output
%originalSize: image size of orig image
%quantizedSize: image size after the G(u,v)/Q(u,v)*quality step
%RLEcompressed: image size after RLE compression


function [originalSize,quantizedSize,RLECompressed] = compressionHelp(imIn, qualit

R=double(imIn);
```

```
        Error using compressionHelp (line 18)
        Not enough input arguments.
```

Convert from offset binary (uint8) to straight binary (int8)

Note also that we operate on the RGB planes instead of the YCbCr, for demo purposes.

```
R=R-128;
```

Perfrom DCT on 8x8 blocks

We use the matlab dct2 function, combined with the blckproc() function. The DCT is a real valued function that is similar in nature to the fourier transform, except that the kernel is purely real and so is the output. It represents the spatial frequency content of an array or image. Note the mechanism to use the dct2 function on subimages of the R plane: First define a pointer to the function dct2, then call it with the array and block size.

```
imOut=double(zeros(size(imIn)));
```

```
fun=@dct2;      % Pointer to the function dct2
Rdct=blkproc(R,[8 8],fun);  % Note the use of the function on 8x8 blocks

imOut = Rdct;
imOut=(imOut-min(imOut(:)));
imOut=imOut/max(imOut(:));
```

Scale by the quantization matrix and round off

Quantization matrix, defined in JPEG format standard

```
Q=[16, 11, 10, 16,  24,  40,  51,  61
   12, 12, 14, 19,  26,  58,  60,  55
   14, 13, 16, 24,  40,  57,  69,  56
   14, 17, 22, 29,  51,  87,  80,  62
   18, 22, 37, 56,  68, 109, 103,  77
   24, 35, 55, 64,  81, 104, 113,  92
   49, 64, 78, 87, 103, 121, 120, 101
   72, 92, 95, 98, 112, 100, 103,  99];

% "Quality" factor.
q=quality;    % Try q=1 & q=15 for example

% Again, apply the operation of scaling through the blkproc() function
% call.  Here we define the function explicitely with the matrix Q and the
% quality factor.
fun=@(x) round(x./(Q*q));
Rdct=blkproc(Rdct,[8 8],fun);

imOut=Rdct;
imOut=(imOut-min(imOut(:)));
imOut=imOut/max(imOut(:));
```

Check how different the image is from the original

Compute the recovered image

```
fun=@(x) x.*Q*q;
R1=blkproc(Rdct,[8 8],fun);

fun=@idct2;
R2=blkproc(R1,[8 8],fun);

R3=R2+128;

imOut=uint8(zeros(size(imIn)));
imOut=uint8(R3);

% Compute the difference image between original pixel values and recovered
% image pixel values.
Rdiff=abs((R+128)-R3);
```

```
imOut=uint8(Rdiff);
```

Original Image

```
originalSize=prod(size(imIn));
```

Quantized Image

```
index=find(round(Rdct)~=0);
vals=Rdct(index);
quantizedSize=length(index);
```

Entropy coded

```
entropyCodedIm = entropyCode(imOut);
```

RLE

```
x = reshape(entropyCodedIm,1,numel(entropyCodedIm));
[RLE_Values,RLE_Frequency] = RLE(x);
RLECompressed = numel(RLE_Values);

end
```

Published with MATLAB® R2014a