For “baboon.tif”, the PSNR between the 90, 70, 50, 30, and 10 quality factors and the original image are 37.3538, 31.0808, 28.9656, 27.3058, 24.3335 respectively. The file size for the quality factors of 90, 70, 50, 30, 10 are 104 KB, 56 KB, 41 KB, 29 KB, and 13 KB respectively. (*Figures 1 – 5*).

For “peppers.tif” the PSNR between the 90, 70, 50, 30, and 10 quality factors and the original image are 50.6244, 46.6185, 36.2785, 35.0866, 30.6462 respectively. The file size for the quality factors of 90, 70, 50, 30, 10 are 48 KB, 33 KB, 27 KB, 16 KB, and 9 KB respectively. (*Figures 6 – 10*).

As file size of the jpeg decreases, the quality of the image decreases. The images of quality factor 90 look the best, while the images with a quality factor of 10 is  When looking at the lower quality factor images, there are some distortions where certain areas of the picture do not have pixel values of what they should be, causing detail to be lost. These occur because JPEG is a lossy form of compression, where a very small file get’s rid of a lot of information in order to get to its small size. When the uncompressing the image, that lost information gets filled in with a best guess of what it should be. When looking at all of the images, only those with a quality factor of 10 show distortions that are unacceptable to look at.

The file size for the image when going through the jpeg compression is 23 KB. The original image size is 254 KB. This is a notable difference of 10x less space used when going under JPEG compression. The PSNR between the two images is 7.1185. A few different quantization tables where used to test which would be best for the image “peppers.tif”.  The decoded image can be seen in *Figure 11*. The final quantization table can be seen in *Table 1*. Some quantization tables gave a smaller file size, and some gave a larger PSNR. However, it is possible to achieve both a lower file size and a higher PSNR at the same time though one JPEG compression. This would require the optimal quantization table for this specific image to be used, in order to achieve both. A lower file size and a higher PSNR are not mutually exclusive properties.

While running this part of the experiment, there were a few issues encountered with the decoder. The information stored from the encoder works fine. All of the information was transferred properly, as noted by the text file output of the encoder. All of the information can be read by the decoder, and can be operated on in the inverse steps of what was done in the encoder. However, when the decoder was able to put the image together, the image did not look correct. The PSNR between the two images is lower than what it should be. Going off the first part of the assignment, the PSNR should be around 30.

It is possible that the method used to encode the picture has the information scrambled in the wrong place. This means the pixels values for each 8x8 correspond do a different pixel. It is also possible that the testing with the original quantization table lost too much data. When looking at the values for the image, a lot of the values became 0. This can translate to a lot of lost information. Additionally, it is possible, that the steps were not performed correctly or read into the decoder correctly. All of these issues combined or separately, caused the image to not be properly decoded.

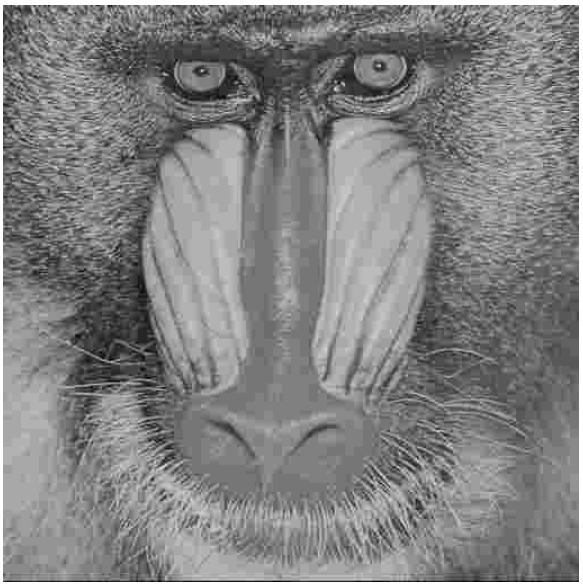

*Figure 1: Baboon_10*

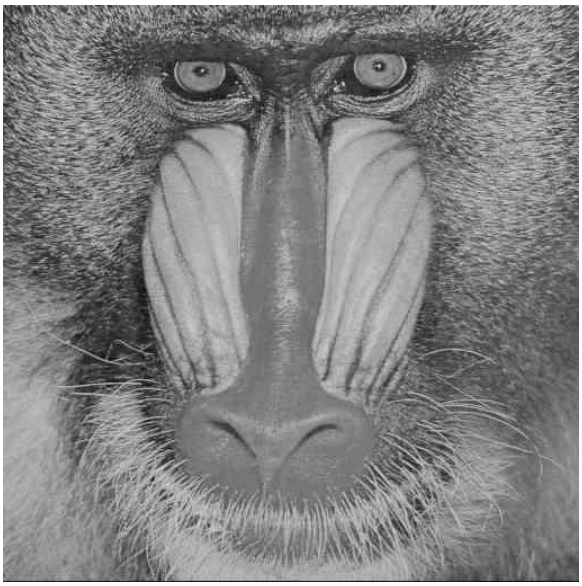

*Figure 2: Baboon_30*

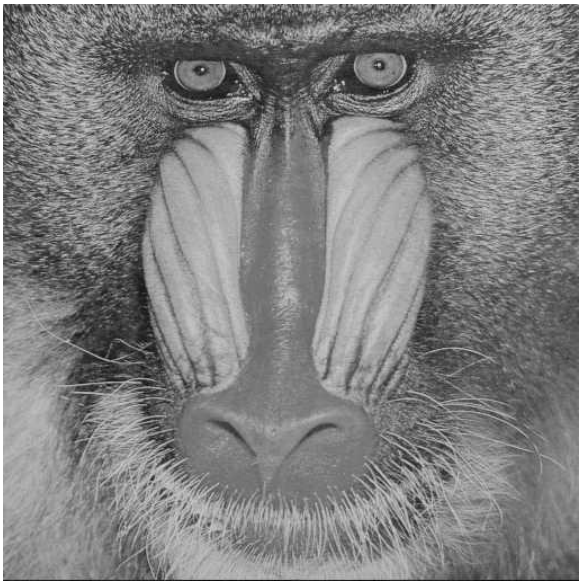

*Figure 3: Baboon_50*

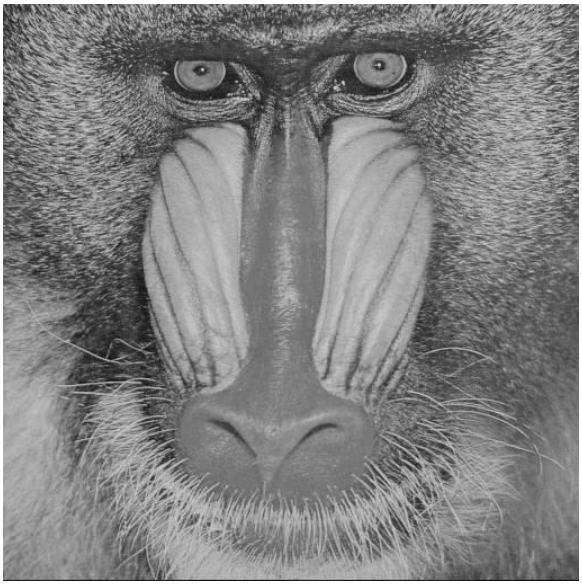

*Figure 4: Baboon_70*

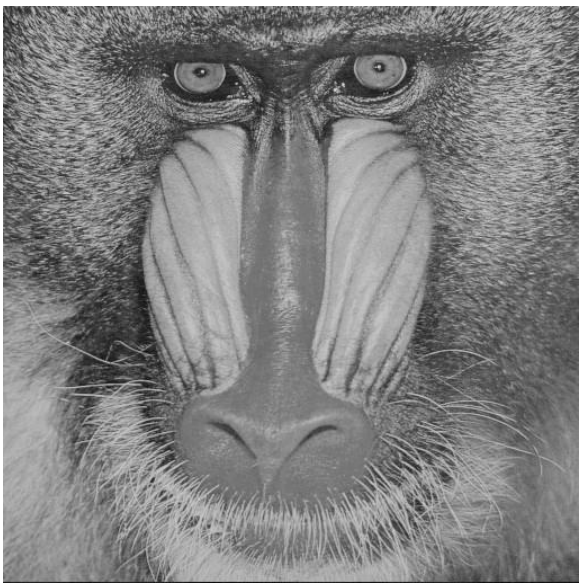

*Figure 5: Baboon_90*

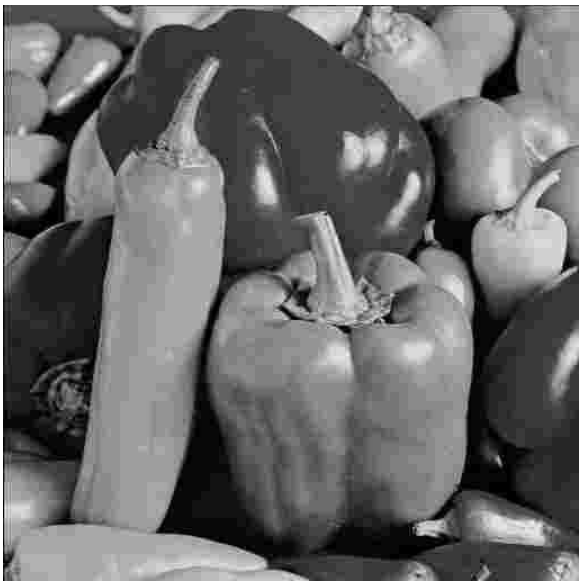

*Figure 6: Peppers_10*

*Figure 7: Peppers_30*

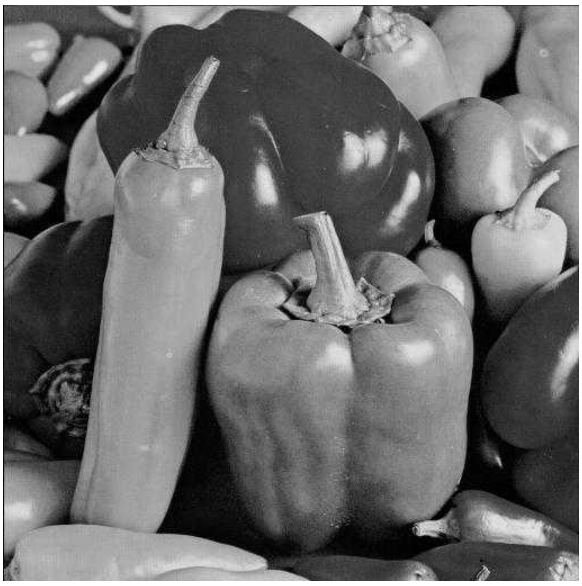

*Figure 8: Peppers_50*

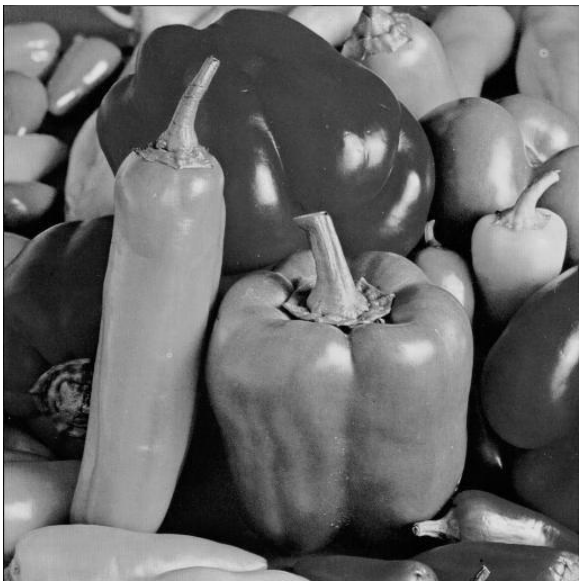

*Figure 9: Peppers_70*

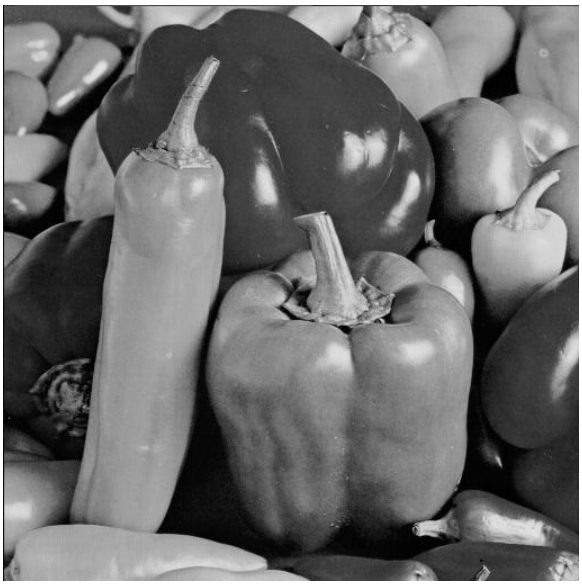

*Figure 10: Peppers_90*

Table 1: Alternate Quantization Table Used

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -76 | -73 | -67 | -62 | -58 | -67 | -64 | -55 |
| -65 | -69 | -73 | -38 | -19 | -43 | -59 | -56 |
| -66 | -69 | -60 | -15 | 16 | -24 | -62 | -55 |
| -65 | -70 | -57 | -6 | 26 | -22 | -58 | -59 |
| -61 | -67 | -60 | -24 | -2 | -40 | -60 | -58 |
| -49 | -63 | -68 | -58 | -51 | -60 | -70 | -53 |
| -43 | -57 | -64 | -69 | -73 | -67 | -63 | -45 |
| -41 | -49 | -59 | -60 | -63 | -52 | -50 | -34 |

Table 2: Default Quantization Table Used

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 | 65 |
| 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 |

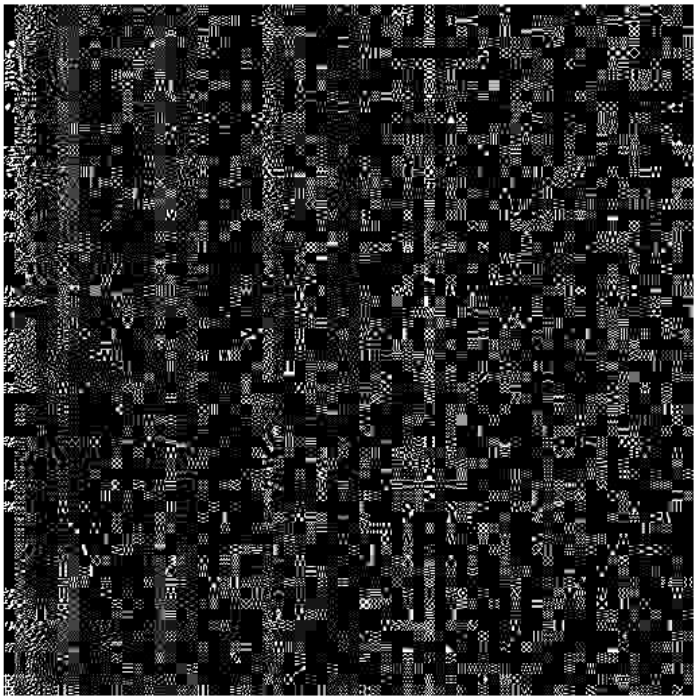

*Figure 11: Alternate Quantization Table Decoded Image*

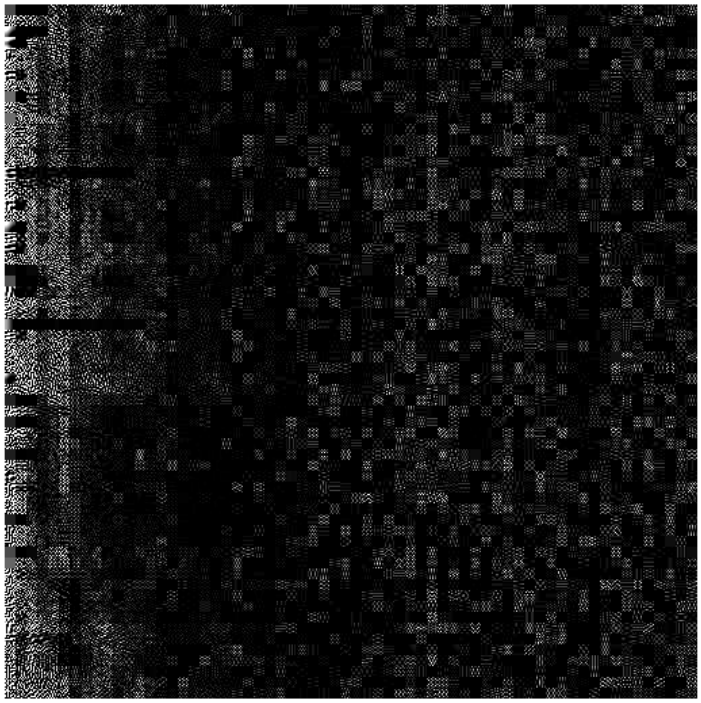

*Figure 12: Default Quantization Table Decoded Image*

.

Matlab Code:

```
% ECES 435
% Assignment 2
% Part 2
% Yoni Carver, Jonah Rubino

clear; clc; close all
%%

% load image
peppers = imread('peppers.tif');

% standard JPEG luminance quantization table
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];

% alternate quantization table
% Q = [-76 -73 -67 -62 -58 -67 -64 -55
%   -65 -69 -73 -38 -19 -43 -59 -56
%   -66 -69 -60 -15 16 -24 -62 -55
%   -65 -70 -57 -6 26 -22 -58 -59
%   -61 -67 -60 -24 -2 -40 -60 -58
%   -49 -63 -68 -58 -51 -60 -70 -53
%   -43 -57 -64 -69 -73 -67 -63 -45
%   -41 -49 -59 -60 -63 -52 -50 -34];

%% ENCODE
encoded_image = jpeg_encoder(peppers, Q);
disp(['Size of encoded image file: ', num2str(encoded_image), ' bytes'])% 23191 bytes

%% DECODE
decoded_image = jpeg_decoder();

%% Encode function
function encode = jpeg_encoder(image, Q)

    image = uint8(image);   % convert image to uint8

    % Using blkproc (distinct block processing for image)
    % For every 8x8 block of the image, take the discrete 2D cosine transform,
    % quantize each value using quantization table (Q), convert the matrix to a
    % zig-zagged vector, and transpose it

    % Note: y is the 8x8 block of the image
    fnc_encode = @(y) ZigzagMtx2Vector(dct2(y) ./ Q)';
```

```
    j = blkproc(image, [8 8], fnc_encode);  % process each 8x8 block with function
"fnc_encode"

    % [Len]=JPEG_entropy_encode(rowN, colN, dct_block_size, Q, ZZDCTQIm,
encoder_path, DisplayProcess_Flag)
    rowN = 512;             % number of rows in the original image
    colN = 512;             % number of columns in the original image
    dct_block_size = 8;     % size of DCT block (8x8 block)
    % Q = Q
    ZZDCTQIm = j;           % 4096 x 64
    encoder_path = './';    % path to current directory that contains all relevant
files
    DisplayProcess_Flag = 0;

    % run the given encode function
    encode = JPEG_entropy_encode(rowN, colN, dct_block_size, Q, ZZDCTQIm,
encoder_path, DisplayProcess_Flag); % encodes the sequence of quantized DCT
coeffcients

end

%% Decode function
function reconstructed_image = jpeg_decoder()

    % function
[rowN,colN,dct_block_size,iQ,iZZDCTQIm]=JPEG_entropy_decode(decoder_path)

    % rowN = 512;
    % colN = 512;
    % dct_block_size = 8;
    % iQ
    % iZZDCTQIm

    decoder_path = './';    % path to current directory that contains all relevant
files

    % decode the encoded image signal
    [rowN,colN,dct_block_size,iQ,iZZDCTQIm] = JPEG_entropy_decode(decoder_path);

    % Using blkproc (distinct block processing for image)
    % Each row of the returned "iZZDCTQIm" variable is a spread out 8x8 pixel
    % grid. We must un-zig-zag them, multiply them by the quantization table
    % (Q), and take the inverse 2D discrete cosine transform

    fnc_decode = @(y) idct2(Vector2ZigzagMtx(y) .* iQ)';
    jj = blkproc(iZZDCTQIm, [1 64], fnc_decode);

    % The output of jj gives us a giant matrix that contains all 8x8 pixel
    % blocks stacked on top of each other
    % Reconstruct the decoded image by taking each 8x8 pixel grid and place them
    % 64 blocks across by 64 blocks down (i.e. 512x512)
    % Take the transpose of each line since these represent each row (not column)
    % we're sorry and you're going to hate this...but:
```

```
a1 = jj(1:512,:)';
a2 = jj(512+1:512*2,:)';
a3 = jj(512*2+1:512*3,:)';
a4 = jj(512*3+1:512*4,:)';
a5 = jj(512*4+1:512*5,:)';
a6 = jj(512*5+1:512*6,:)';
a7 = jj(512*6+1:512*7,:)';
a8 = jj(512*7+1:512*8,:)';
a9 = jj(512*8+1:512*9,:)';
a10 = jj(512*9+1:512*10,:)';
a11 = jj(512*10+1:512*11,:)';
a12 = jj(512*11+1:512*12,:)';
a13 = jj(512*12+1:512*13,:)';
a14 = jj(512*13+1:512*14,:)';
a15 = jj(512*14+1:512*15,:)';
a16 = jj(512*15+1:512*16,:)';
a17 = jj(512*16+1:512*17,:)';
a18 = jj(512*17+1:512*18,:)';
a19 = jj(512*18+1:512*19,:)';
a20 = jj(512*19+1:512*20,:)';
a21 = jj(512*20+1:512*21,:)';
a22 = jj(512*21+1:512*22,:)';
a23 = jj(512*22+1:512*23,:)';
a24 = jj(512*23+1:512*24,:)';
a25 = jj(512*24+1:512*25,:)';
a26 = jj(512*25+1:512*26,:)';
a27 = jj(512*26+1:512*27,:)';
a28 = jj(512*27+1:512*28,:)';
a29 = jj(512*28+1:512*29,:)';
a30 = jj(512*29+1:512*30,:)';
a31 = jj(512*30+1:512*31,:)';
a32 = jj(512*31+1:512*32,:)';
a33 = jj(512*32+1:512*33,:)';
a34 = jj(512*33+1:512*34,:)';
a35 = jj(512*34+1:512*35,:)';
a36 = jj(512*35+1:512*36,:)';
a37 = jj(512*36+1:512*37,:)';
a38 = jj(512*37+1:512*38,:)';
a39 = jj(512*38+1:512*39,:)';
a40 = jj(512*39+1:512*40,:)';
a41 = jj(512*40+1:512*41,:)';
a42 = jj(512*41+1:512*42,:)';
a43 = jj(512*42+1:512*43,:)';
a44 = jj(512*43+1:512*44,:)';
a45 = jj(512*44+1:512*45,:)';
a46 = jj(512*45+1:512*46,:)';
a47 = jj(512*46+1:512*47,:)';
a48 = jj(512*47+1:512*48,:)';
a49 = jj(512*48+1:512*49,:)';
a50 = jj(512*49+1:512*50,:)';
a51 = jj(512*50+1:512*51,:)';
a52 = jj(512*51+1:512*52,:)';
a53 = jj(512*52+1:512*53,:)';
a54 = jj(512*53+1:512*54,:)';
a55 = jj(512*54+1:512*55,:)';
```

```
    a56 = jj(512*55+1:512*56,:)';
    a57 = jj(512*56+1:512*57,:)';
    a58 = jj(512*57+1:512*58,:)';
    a59 = jj(512*58+1:512*59,:)';
    a60 = jj(512*59+1:512*60,:)';
    a61 = jj(512*60+1:512*61,:)';
    a62 = jj(512*61+1:512*62,:)';
    a63 = jj(512*62+1:512*63,:)';
    a64 = jj(512*63+1:512*64,:)';

    % place all rows on top of each other & convert to uint8
    reconstructed_image = uint8([a1;a2;a3;a4;a5;a6;a7;a8;a9;
        a10;a11;a12;a13;a14;a15;a16;a17;a18;a19;
        a20;a21;a22;a23;a24;a25;a26;a27;a28;a29;
        a30;a31;a32;a33;a34;a35;a36;a37;a38;a39;
        a40;a41;a42;a43;a44;a45;a46;a47;a48;a49;
        a50;a51;a52;a53;a54;a55;a56;a57;a58;a59;
        a60;a61;a62;a63;a64]);
    imshow(reconstructed_image)
    % 64 blocks across = 512

end
```