

Lab 2: JPEG Compression

Brian Hosler & Sarah Peachey

February 1, 2018

Abstract

Contents

1	JPEG Quality Factor	2
2	Writing JPEG compression in matlab	7
3	Evaluating Quantization Tables	8

1 JPEG Quality Factor

Peak Signal to Noise Ratio (PSNR) is a common method for measuring the quality of a image that has gone through compression. PSNR is a way to quantize the quality of a decoded image as perceived by the human eye. (1) is how PSNR is calculated, where σ_e^2 is the variance of noise/error and A is the bit depth of the image. The value calculated can be interpreted as unnoticable noise if it is greater than 40dB, as low quality if it is less than 30dB, and as good quality otherwise.

$$PSNR = 10 \log_{10} \left(\frac{A^2}{\sigma_e^2} \right) \quad (1)$$

```
pep=imread('peppers.tif');
bab=imread('baboon.tif');

imwrite(pep, 'pep90.jpg', 'Quality',90)
imwrite(pep, 'pep70.jpg', 'Quality',70)
imwrite(pep, 'pep50.jpg', 'Quality',50)
imwrite(pep, 'pep30.jpg', 'Quality',30)
imwrite(pep, 'pep10.jpg', 'Quality',10)

imwrite(bab, 'bab90.jpg', 'Quality',90)
imwrite(bab, 'bab70.jpg', 'Quality',70)
imwrite(bab, 'bab50.jpg', 'Quality',50)
imwrite(bab, 'bab30.jpg', 'Quality',30)
imwrite(bab, 'bab10.jpg', 'Quality',10)

pep_psnr=zeros(1,6);
pep_size=zeros(1,6);

temp=imfinfo('peppers.tif');
pep_size(1)=temp.FileSize;
pep_psnr(1)= psnr(pep, pep);

temp=imfinfo('pep90.jpg');
pep_size(2)=temp.FileSize;
pep_psnr(2)= psnr(imread('pep90.jpg'), pep);

temp=imfinfo('pep70.jpg');
pep_size(3)=temp.FileSize;
pep_psnr(3)= psnr(imread('pep70.jpg'), pep);

temp=imfinfo('pep50.jpg');
pep_size(4)=temp.FileSize;
pep_psnr(4)= psnr(imread('pep50.jpg'), pep);

temp=imfinfo('pep30.jpg');
pep_size(5)=temp.FileSize;
pep_psnr(5)= psnr(imread('pep30.jpg'), pep);

temp=imfinfo('pep10.jpg');
pep_size(6)=temp.FileSize;
pep_psnr(6)= psnr(imread('pep10.jpg'), pep);

figure
subplot(2,1,1)
plot(100*[1 .9 .7 .5 .3 .1], pep_size, '—o')
hold on
title('peppers--file_size_vvs_quality')
```

```

subplot(2,1,2)
plot(100*[1 .9 .7 .5 .3 .1], pep_psnr, '—o')
hold on
title('peppers—psnr—vs—quality')

bab_psnr=zeros(1,6);
bab_size=zeros(1,6);

temp=imfinfo('baboon.tif');
bab_size(1)=temp.FileSize;
bab_psnr(1)= psnr(bab, bab);

temp=imfinfo('bab90.jpg');
bab_size(2)=temp.FileSize;
bab_psnr(2)= psnr(imread('bab90.jpg'), bab);

temp=imfinfo('bab70.jpg');
bab_size(3)=temp.FileSize;
bab_psnr(3)= psnr(imread('bab70.jpg'), bab);

temp=imfinfo('bab50.jpg');
bab_size(4)=temp.FileSize;
bab_psnr(4)= psnr(imread('bab50.jpg'), bab);

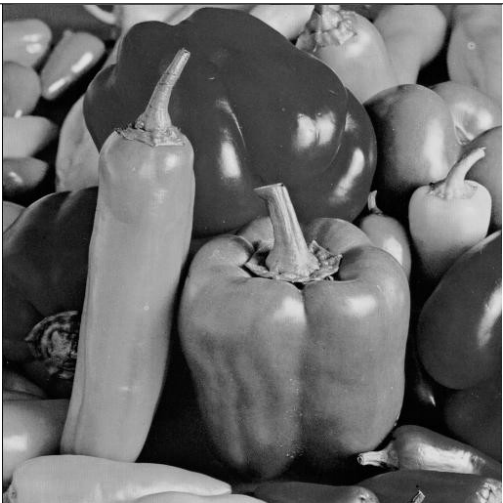
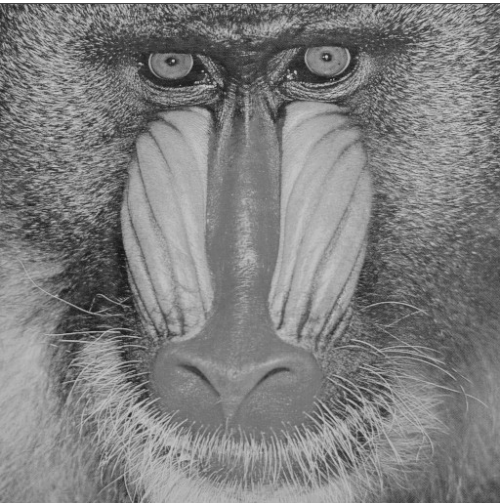

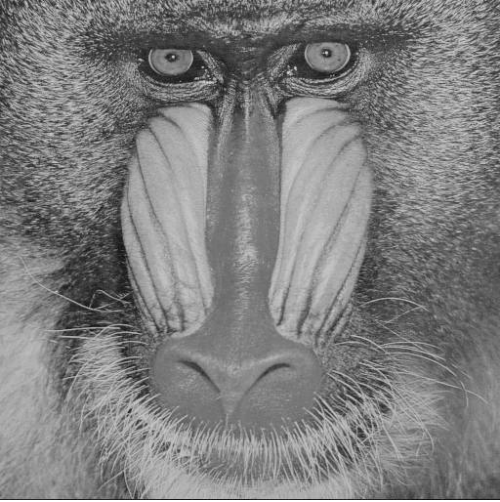
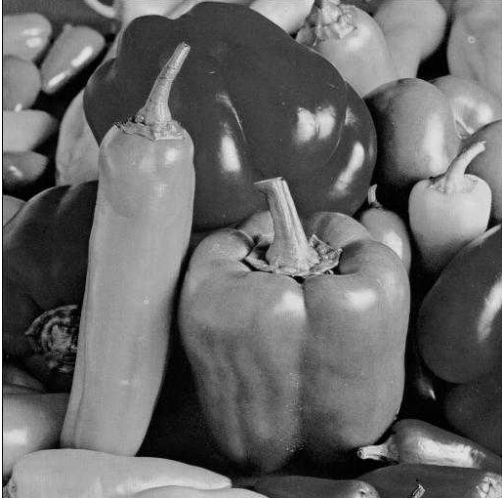
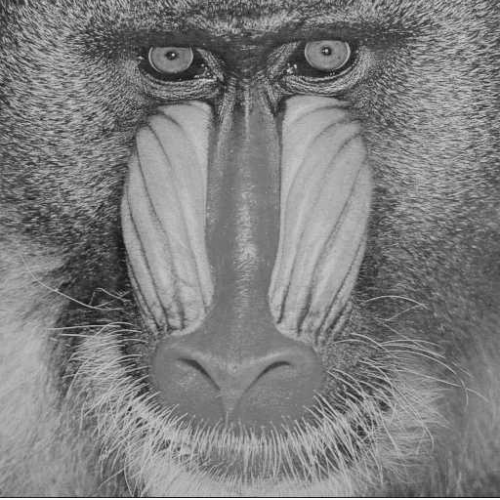
temp=imfinfo('bab30.jpg');
bab_size(5)=temp.FileSize;
bab_psnr(5)= psnr(imread('bab30.jpg'), bab);

temp=imfinfo('bab10.jpg');
bab_size(6)=temp.FileSize;
bab_psnr(6)= psnr(imread('bab10.jpg'), bab);

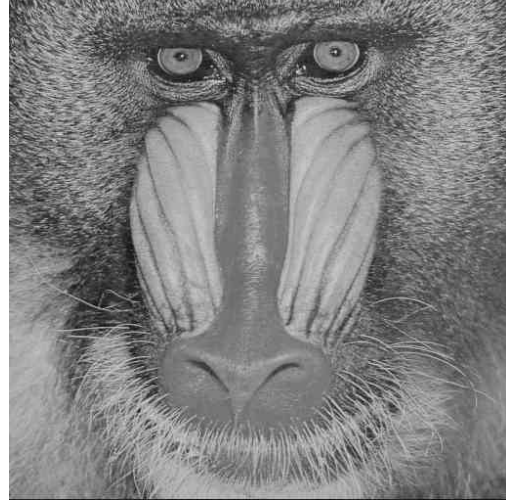
figure
subplot(2,1,1)
plot(100*[1 .9 .7 .5 .3 .1], bab_size, '—o')
hold on
title('baboon—file—size—vs—quality')
subplot(2,1,2)
plot(100*[1 .9 .7 .5 .3 .1], bab_psnr, '—o')
hold on
title('baboon—psnr—vs—quality')

```

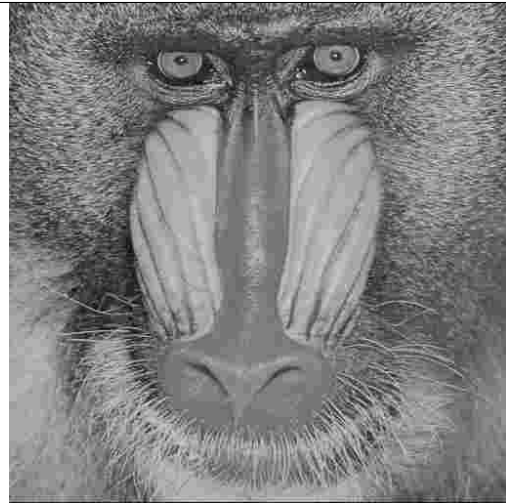
In the following table the peppers.tif and baboon.tif were compressed with image quality factors of 90, 70, 50, 30, & 10.

Quality Factor	Peppers	Baboon
90		
70		
50		

30



10



As seen in figures 1, and 2 as the image file size gets smaller the PSNR of the image drops lower. Also, as seen in the images as the quality gets lower the image becomes more blocky and noise is added in the high frequency sections. Which is normally fine because the human eye is less perceptive to changes in high frequency regions, but edge also count as high frequency and noise can be perceived on edged of smooth regions. The blockiness occurs because jpeg divides the image into 8×8 block and then runs the algorithm on each block, therefore, some block may throw away or keep more information than the one next to it. Also the jpeg algorithm takes into account the humans do not perceive change in high frequency regions as well, so it will discard this information.

There is a clear drop in image quality around a quality factor of 30 or 10. The image becomes far more blocky because of the loss in information. The PSNR graphs also reflect this, because the peppers at quality 10 have a PSNR of about 30, and the baboon at quality of 30 drops below 30dB. Since less than 30dB is the qualification for low quality it make sense that these images have visually perceived error.

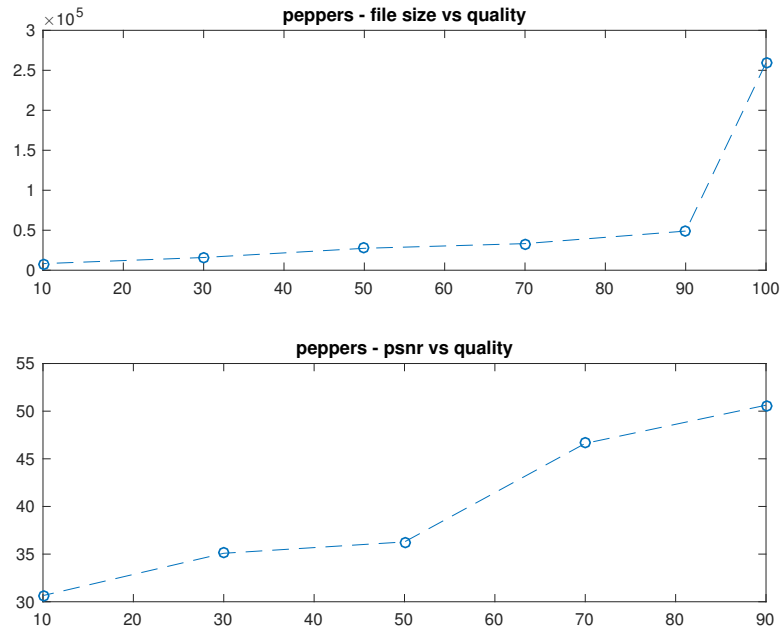


Figure 1: Peppers file size and PSNR as the quality changes

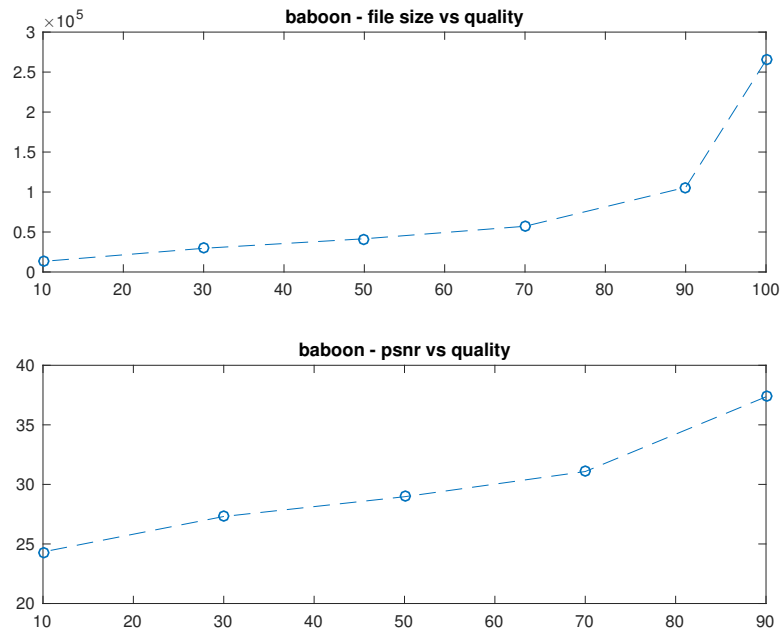


Figure 2: Baboon file size and PSNR as the quality changes

2 Writing JPEG compression in matlab

The following sections are the code that implements the jpeg encoding algorithm by segmenting the image into 8×8 pixel blocks, computing the discrete cosine transform, and quantizing each block. Then using the provided functions to further format and entropy encode the jpeg. That process is then reversed for decoding the image.

myJpgEncode.m

```
function [result] = myJpgEncode( pep,Q ) 1
%myJpgEncode implement my own jpeg algorithm 2
% using the notes 3
A=zeros(size(pep)); 4
stor=[]; 5
for i=1:512/8 6
    for j=1:512/8 7
        tempA=dct2(pep(8*(i-1)+1:i*8,8*(j-1)+1:j*8)); 8
        pep_quan=round(tempA./Q).*Q; 9
        stor=[stor; ZigzagMtx2Vector(pep_quan)]; 10
    end 11
end 12
result=JPEG_entropy_encode(512,512,8,Q,stor, './',1); 13
end 14
```

myJpgDecode.m

```
function [pep] = myJpgDecode() 1
%myJpgEncode implement my own jpeg algorithm 2
% using the notes 3
[rowN,colN,dct_block_size,iQ,iZZDCTQIm]=JPEG_entropy_decode(' 4
    './');
pep=zeros(512); 5
ndx=1; 6
for i=1:512/8 7
    for j=1:512/8 8
        pep(8*(i-1)+1:i*8,8*(j-1)+1:j*8)=idct2( 9
            Vector2ZigzagMtx(iZZDCTQIm(ndx,:)));
        ndx=ndx+1; 10
    end 11
end 12
end 13
```

3 Evaluating Quantization Tables

Our jpeg encoding and decoding algorithms are then tested by using the standard jpeg luminance quantization table. Then a custom quantization table is created based off of the values from the DCT of the peppers.tif image. This custom table should better compress the image, without losing too much information as to distort the image.

```

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];

tempQ=zeros(8);
for i=1:512/8
    for j=1:512/8
        tempQ=tempQ+abs(dct2(pep(8*(i-1)+1:i*8,8*(j-1)+1:j*8)))
    end
end
DCTs=tempQ/4096;
nrm1=max(max(DCTs))./DCTs;
Q2=double(uint8(nrm1.^65));

stor=[];
for i=1:512/8
    for j=1:512/8
        tempA=dct2(pep(8*(i-1)+1:i*8,8*(j-1)+1:j*8));
        stor=[stor;ZigzagMtx2Vector(tempA)];
    end
end
vrnce=Vector2ZigzagMtx(var(stor));
nrm2=max(max(vrnce))./vrnce;
Q3=double(uint8(log(nrm2).^2))+1;

A=myJpgEncode(pep,Q);
jpg1=uint8(myJpgDecode());
psnr(pep,jpg1)

B=myJpgEncode(pep,Q2);
jpg2=uint8(myJpgDecode());
psnr(pep,jpg2)

C=myJpgEncode(pep,Q3);
jpg3=uint8(myJpgDecode());
psnr(pep,jpg3)

figure
imshow([pep,jpg1;jpg2,jpg3])

```

**** NOTE:** Put some words about the actual Quantization tables used and the results of each one, in terms of PSNR and visual perception. ******

Q-Table	Compressed Size(kB)	PSNR(dB)
Standard	44.7	36.27
Mean	33.1	36.33
Variance	23.7	34.40

Table 1: Comparison of Different JPEG Quantization Tables Used



Figure 3: Resultant image after no JPEG compression(top-left), compression using the JPEG standard quantization table(top-right), compression using a quantization table based on the mean magnitude of the DCT coefficients(bottom-left), and compression using a quantization table derived from the DCT coefficients' variances(bottom-right).