Assignment Report for JPEG Compression

Name: Paras Goyal

Entry Number: 2018CSB1111

Observations:

Input Image

Using the all the parameters as it is in the code as the above:

1. **No. of normalization parameters =1**

Output:

2. **No. of normalization parameters=2**

Output:

3. **No. of normalization parameters = 3**

Output:

4. **No. of normalization parameters=4**

Output:

5. **No. of normalization parameters =5**

Output:

6. **No. of normalization parameters =6**

Output:

7. **No. of normalization parameters =7**

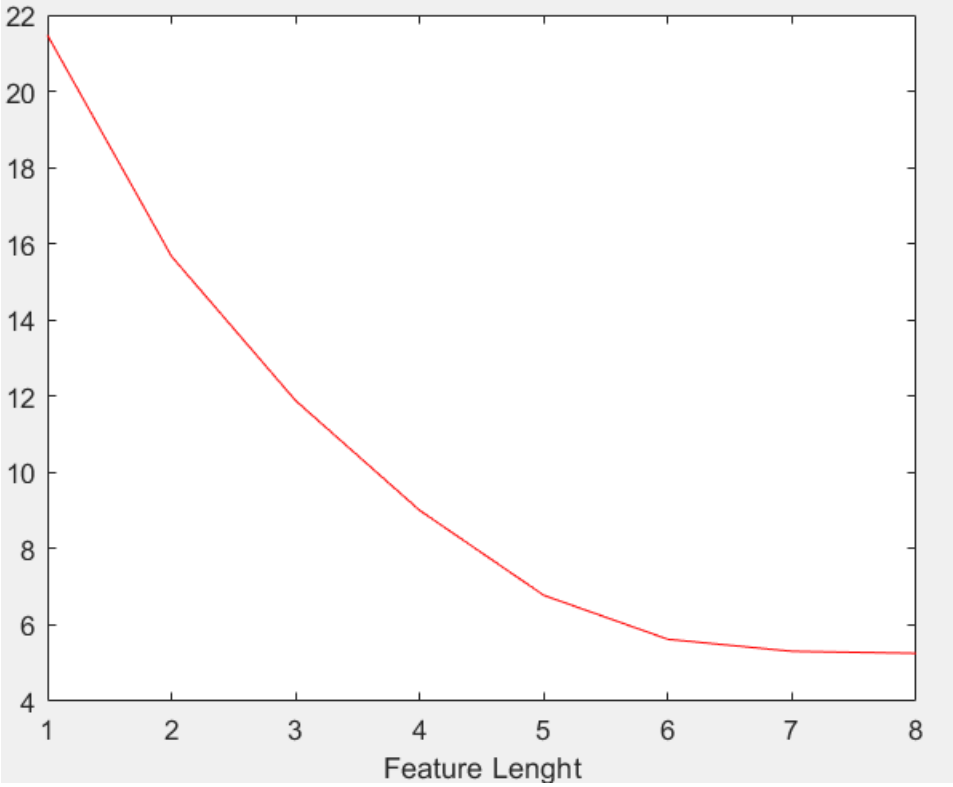
Output: 

8. **No. of normalization parameters =8**

Output: 

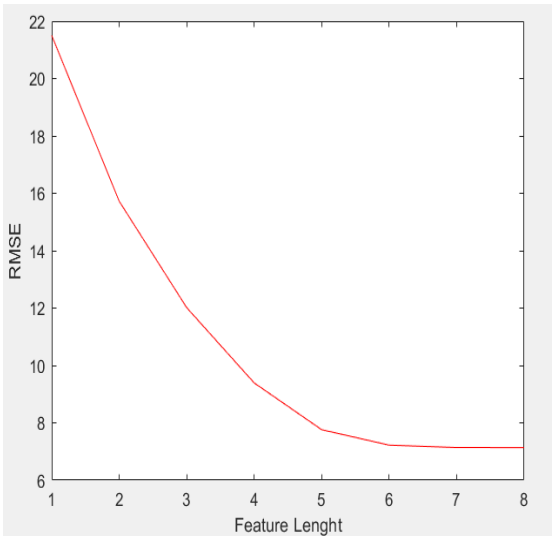
Clearly we can see that as we increase the number of parameters the quality is improved. Also, we cannot achieve an RMSE=0 with max_size of normalization matrix which proves that JPEG is indeed a lossy compression algorithm.

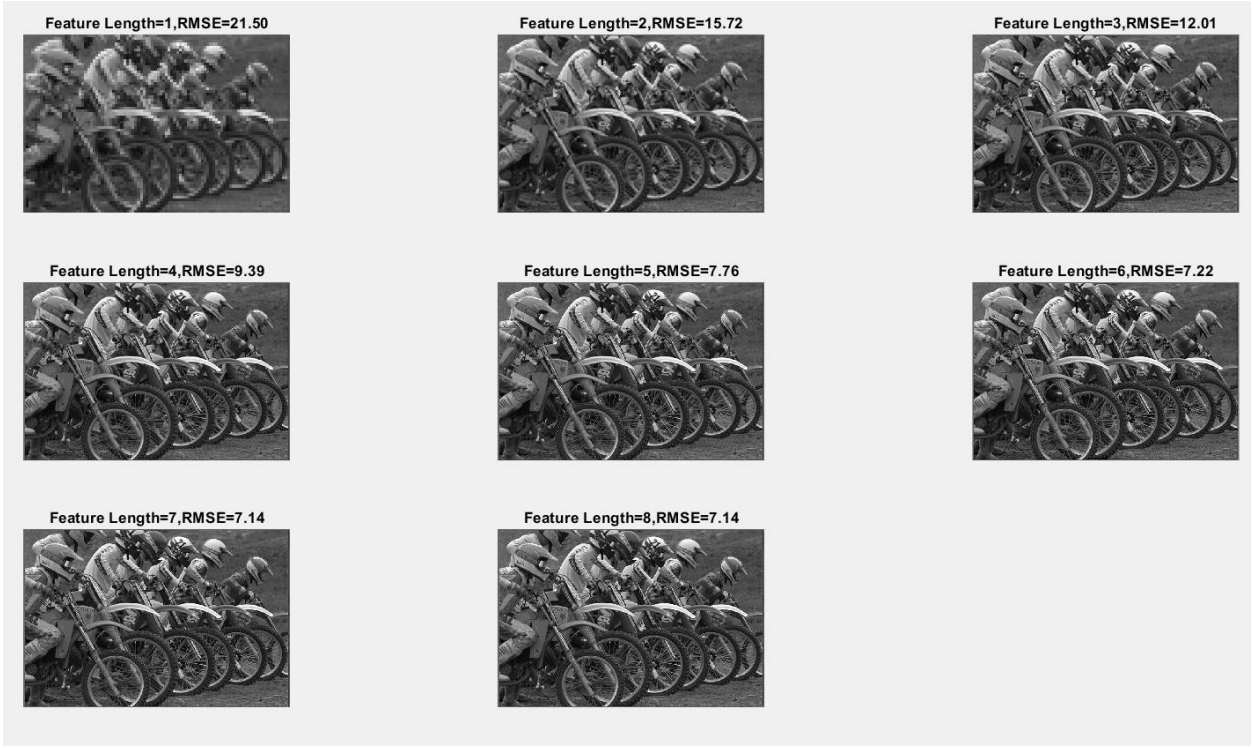
A gif can be created of the above all images using the generate_gif function which gives the effect of transmission of jpeg images on internet. I cannot insert due to gif incompatibility with pdf.



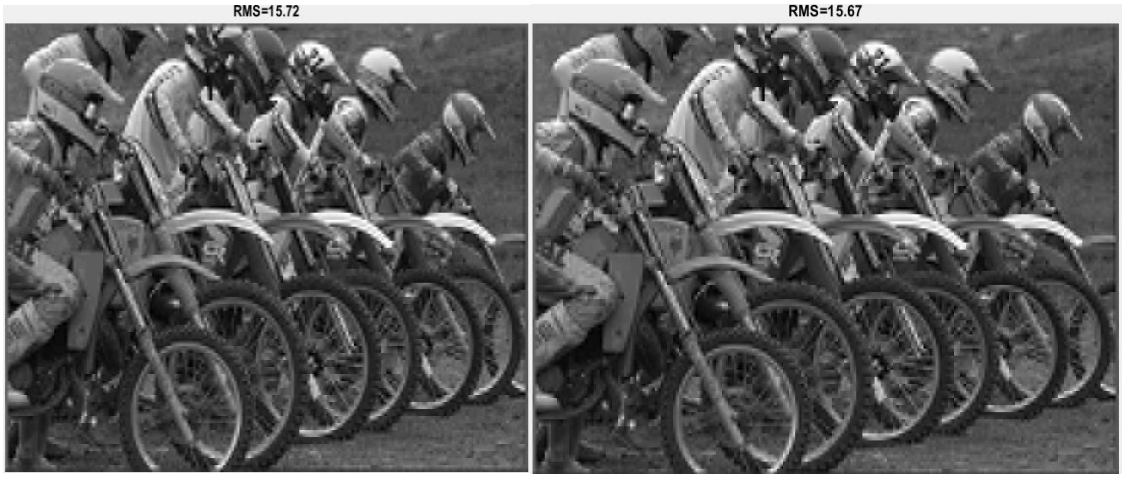
Plot of rmse vs no. of normalization parameters used.

Changing the matrix by setting the quality parameter = 25





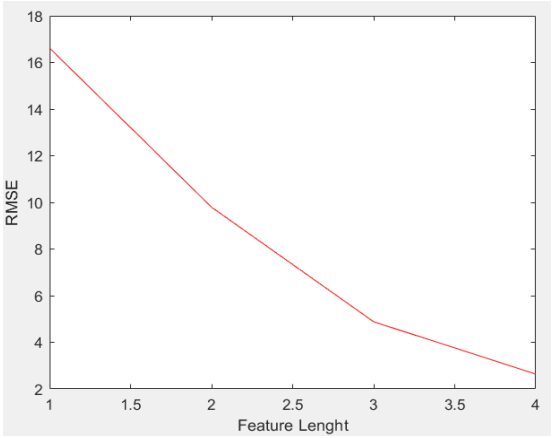
Clearly the RMSE values are higher as compared results.



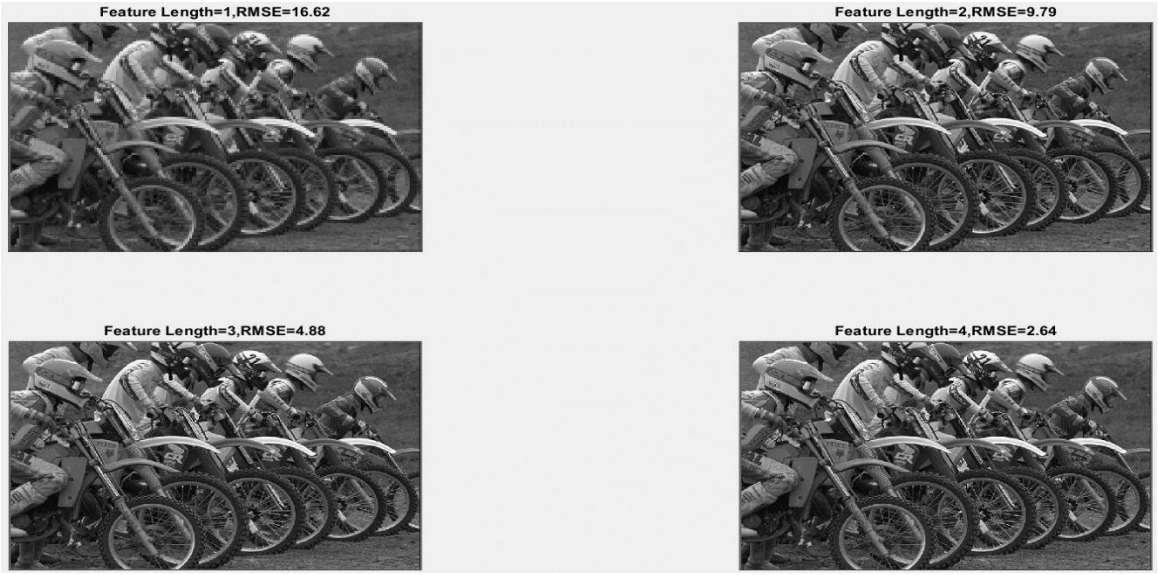
Quality =25 Quality=50

Normalization parameters =2 in both

Changing the Block Size to 4.

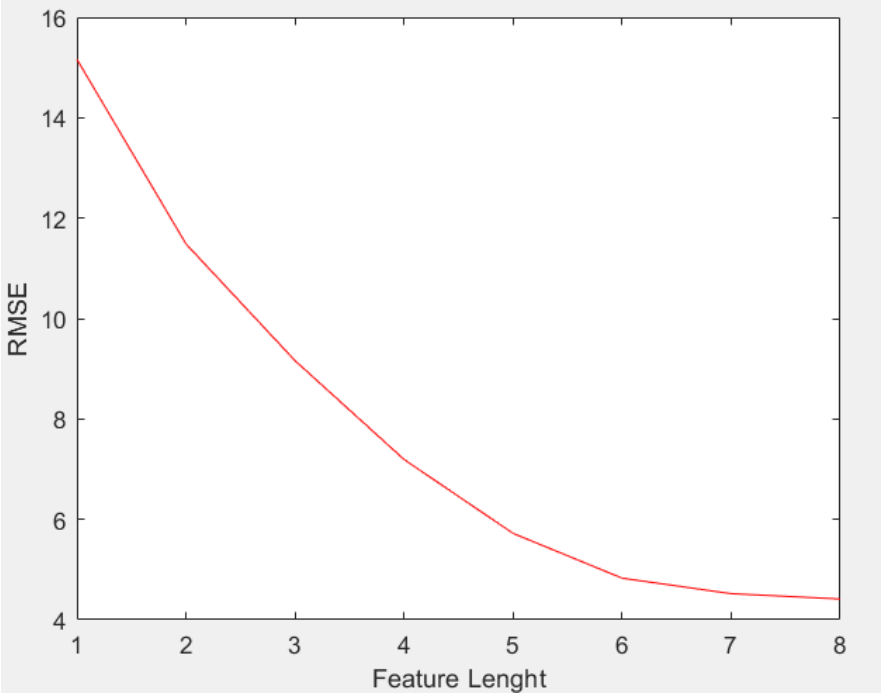


Plot of rmse vs normalization parameters

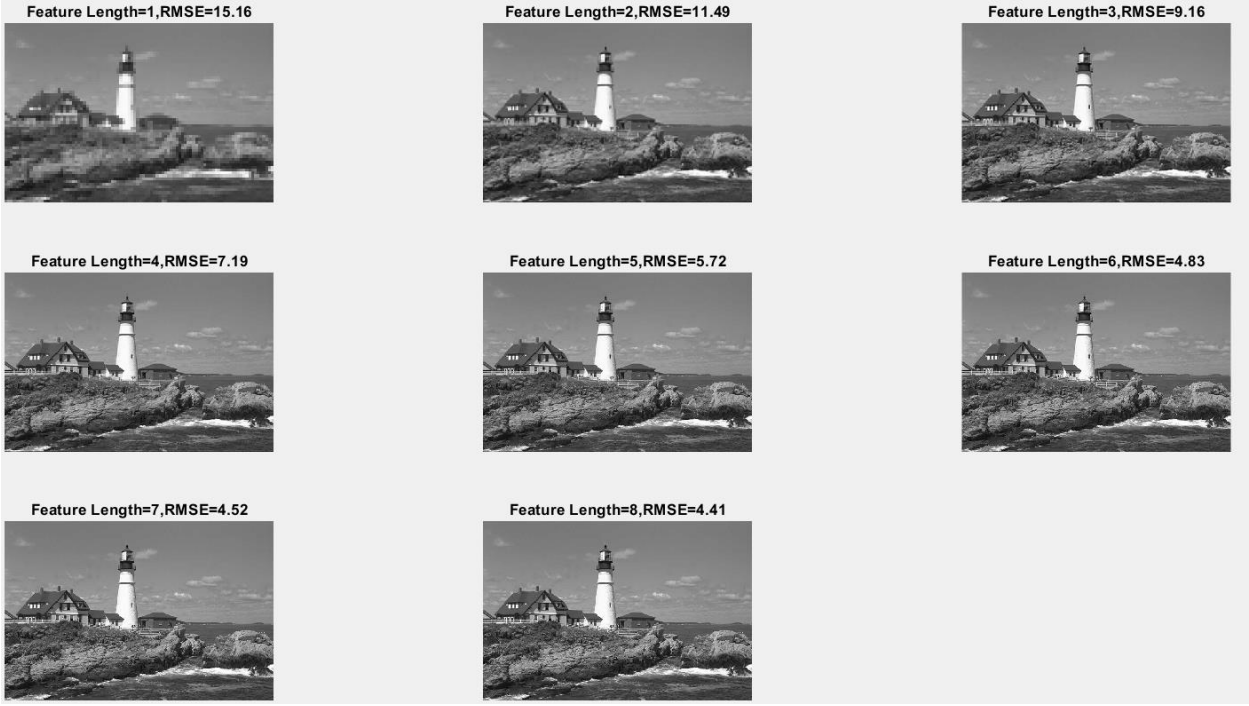


Images generated using different no. of normalization parameters the lower RMSE clearly shows the loss is less as compared to block size of 8 but the size of images in larger in this case.

Input Image 2:

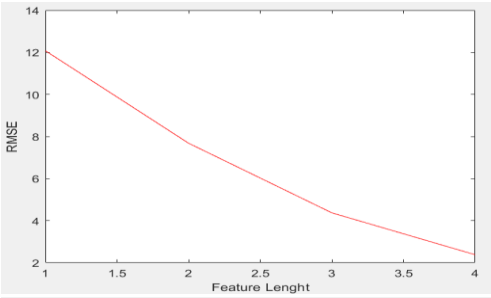


Plot of RMSE vs no. of normalization paramteres.

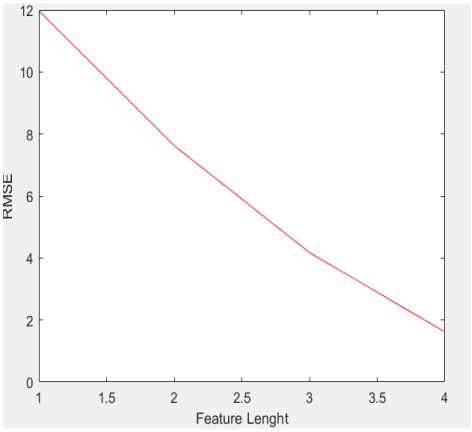


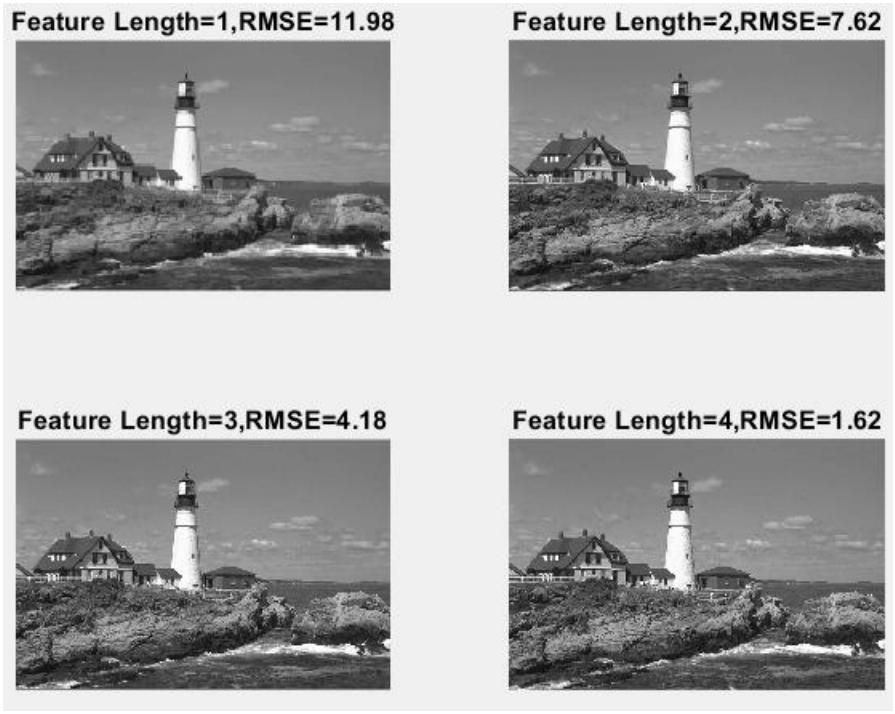
All the images with different number of normalizations parameters. The lower rmse from image 1 results is because this image does not that much variation.

Using the block_size=4



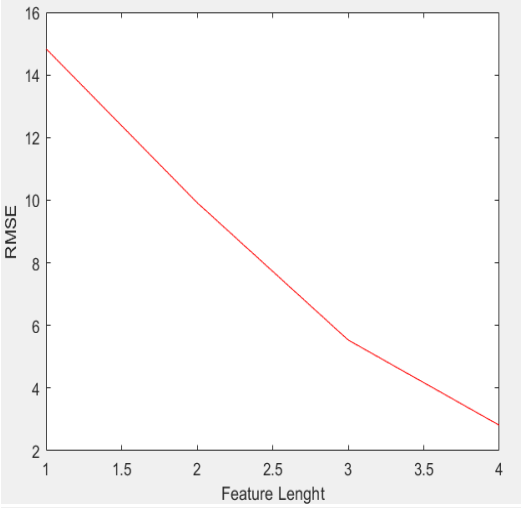
Using Quality =70





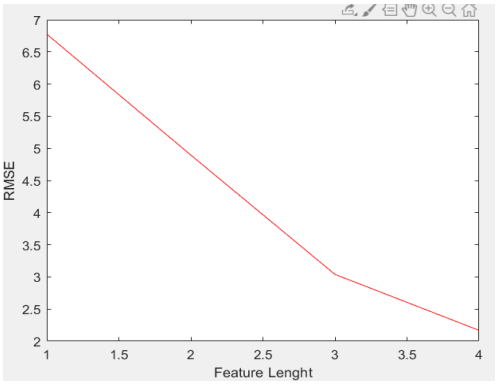
NOTE: when block_size 4 is used the corresponding first 4 rows and columns from QX are considered as the quantization matrix.

Input Image 3: 



Results with quality =50 , block_size=4 and varying normalization parameters.

Input Image 4:



Feature Length=1,RMSE=6.78

Feature Length=2,RMSE=4.89

Feature Length=3,RMSE=3.04

Feature Length=4,RMSE=2.17

Conclusion:

From the above it could be concluded that using lower number of normalization parameters for recovery from encoded image lowers the quality and blocking artifact is clearly visible. As we increase the parameters we get improved quality but never exactly equal to the input due to lossy nature of jpeg compression.

Also we can observe that for images with lower variation it works better as compared to images with high amount of variation.

As we reduce the blocking size we get improved results at the cost increased size of the encoded images.

As we change the quantization matrix using the quality parameter according to equations used in the code we get results accordingly. With lower quality (<50) the values of the matrix increases whereas with higher quality the values in matrix get reduced which permits the high frequency components as well and improved image quality but increased size since we no longer have a greater number of zeros in the encoded image. More the number of zeros in encoded image more will be the compression and reduced will be the quality since high frequency components will not be recovered.

Thus we conclude that it is a lossy compression method and compression ratio depend primarily on the quantization matrix and block size used (which is indeed the size of quantization matrix as well.) The quality of the recovered image depends on the number of parameters used in comparison to the block size which explains the gradual improvement of images transmitted over slow internet connections(JPEG images).

Code : (Written in Matlab)

```
function [O,O2]=jpeg(fname_inp,toshow,toplot)
    QX=[ 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=50;
    block_size=8;
    if quality > 50
        QX = round(QX.*(ones(8)*((100-quality)/50)));
    elseif quality < 50
        QX = round(QX.*(ones(8)*(50/quality)));
    end
    QX=QX(1:block_size,1:block_size);
    n=size(QX,1);
    input_image=rgb2gray(imread(fname_inp));
    dct_quantized=encode_jpeg(fname_inp,QX,n);
    O=cell(n,1);
    rms=zeros(n,1);
    for i=1:n
        O{i}=decode_jpeg(dct_quantized,QX,n,i);
        rms(i)=compute_imgs_rmse(input_image,O{i});
    end
    O2=cell(n,1);
    fig=figure;
    for i=1:n
        imshow(O{i});
        title(sprintf("RMS=%0.2f",rms(i)));
        drawnow
        frame=getframe(fig);
        O2{i}=frame2im(frame);
    end
    close;
    if toshow==1
        a=ceil(sqrt(n));
        figure('Name','Images');
        for i=1:n
            subplot(a,a,i);
            imshow(O{i});
            title(sprintf('Feature Length=%d,RMSE=%0.2f',i,rms(i)));
        end
    end
    if toplot==1
        figure('Name','Plot of rms vs feature length');
        plot(1:n,rms,'r-');
        xlabel('Feature Lenght');
        ylabel('RMSE');
    end
end
function O=encode_jpeg(fname_inp,QX,n)
    I=imread(fname_inp);
    I=double(rgb2gray(I));
    I=padarray(I,[ceil(size(I,1)/n)*n-size(I,1),ceil(size(I,2)/n)*n-
size(I,2)],'post');
    [row,col]=size(I);
    I=I-128;
    QX=double(QX);
    dct_quantized=zeros(row,col);
```

```
    for i1=1:n:row
     for i2=1:n:col
         zBLOCK=I(i1:i1+n-1,i2:i2+n-1);
         win1=dct2(zBLOCK);
         win2=round(win1./QX);
         dct_quantized(i1:i1+n-1,i2:i2+n-1)=win2;
     end
    end
    O=dct_quantized;
end
function O=decode_jpeg(dct_quantized,QX,n,features)
    feature=zeros(n);
    [row,col]=size(dct_quantized);
    feature(1:features,1:features)=ones(features);
    dct_restored=zeros(row,col);
    for i1=1:n:row
     for i2=1:n:col
         win2=dct_quantized(i1:i1+n-1,i2:i2+n-1);
         win2=win2.*feature;
         win3=win2.*QX;
         dct_restored(i1:i1+n-1,i2:i2+n-1)=idct2(win3)+128;
     end
    end
    O=uint8(dct_restored);
end
function xx = compute_imgs_rmse(A,B)
    t1=min(size(A,1), size(B,1));
    t2=min(size(A,2), size(B,2));
    err=double(A(1:t1,1:t2)-B(1:t1,1:t2));
    xx=sqrt( sum(sum(double(err.*err))) / (numel(err)) );
end


function generate_gif(I,output,frame_rate)
    ff=sprintf('%s.gif',output);
    delay=1/frame_rate;
    im=I;
    for idx=1:max(size(I,1),size(I,2))

        if(size(im{idx},3)==1)
             [A,map]=gray2ind(im{idx},256);
        else
            [A,map]=rgb2ind(im{idx},256);
         end
         if(isempty(A)),continue;end
        if idx==1
            imwrite(A,map,ff,'gif','LoopCount',Inf,'DelayTime',delay);
        else

imwrite(A,map,ff,'gif','WriteMode','append','DelayTime',delay);
        end
```

```
    end
    close;
end
```

Description of Function:

1. jpeg: This function is the driver function and decides other parameters to be used for encoding and decoding the jpeg. Quality parameter is used for changing the quantization matrix to allow different qualities. Block Size is the size taken into consideration of the image. This function outputs two cells variables O and O2. O contains all the images restored using the different number of the normalization parameters from 1 to block_size. O2 is also same but images have been titled by there RMSE from original image. to_plot paratmeter plot the rmse vs. no. of normalization parameters from 1 to block size. to_show plots all these images in a grid along with the RMSE Values.
2. encode_jpeg: This function takes the input image name, quantization matrix and block_size and returns the encoded image according to the jpeg algorithm.
3. decode_jpeg: This function takes the encoded matrix from 2nd function above. Also the quantization matrix and the number of normalization parameters to be used. Return the decoded image in uint8 format.
4. Compute_imgs_rmse: Compute the value of rmse between two images and returns it.
5. Generate_gif: takes the Cell which contains images from which gif is generated, output file name and delay (in s) between two consecutive image.