My current work is to study how the JPEG quality factors can influence the Zernike Moments of images, and to find out which moments are most sensitive to the JPEG compression processing.

Firstly, I selected one hundred gray scale images from the image database to form an inspection set. For each image, I decide on a splitting block size of 16x16 to divide images non-overlappingly as a basic research block. The highest order of the Zernike Moment is set as 5, which results in 12 different Zernike Moments of the image. Then, I do the Zernike Moment calculation for each pixel of the image and use the one of the central pixels of every single block to represent the whole block's Zernike Moment attribute. That is, I divide images into 16x16 blocks and calculate twelve Zernike Moments for each block of the image.

I used the MATLAB to do the processing procedure above. In order to inspect the relation between the uncompressed feature values and the compressed values in different Zernike moments, I extract all feature values for all blocks from 100 images and compare uncompressed values with corresponding compressed ones. Firstly, I set up a 1x12 cell, named img_file_non, to store all uncompressed feature values in 12 moments. Within each slot of the cell, there are totally 76800 feature values, representing all uncompressed block features for 100 images. Similarly, I set up another 5x12 cell, named img_file to store all compressed feature values in 12 moments. That is in each row of the img_file cell, compressed feature values are stored into 12 slots for 12 moments, with each slot storing 76800 compressed feature values. And each row represents one compression situation, representing factors of 95, 90, 80, 70, 60 from up to down. The MATLAB coding for this procedure is shown asfollows.

```
%data stored in the img_file, to store the 100 images information after
%compression with an specific JPEG compression factor and for one single
%moment. In each slot of img_file, there are totally 76800 values to
%represent all block information for 100 images.

%And img_file_non for non-compressed images to store information of 100
%images. Similarly, after compression with an specific JPEG compression
%factor and for one single moment. In each slot of img_gile, there are
%totally 76800 values to represent all block information for 100 images.

%%%%   quality  = com_factor     factor list

img_file = cell(5,12);  %set a cell to store feature of 5 compressions, 12 moments

for factor_index = 1:5  %to pick out compression factors in order

    for num_moment = 1:Num_moments   %for each of 12 moments

        f_name = strcat('feat_JPEG_quality_', num2str(quality(factor_index)), '_', num2str(num_moment));
        f = [];

        for img_index = 1:100

            file_name = strcat('ZM_file_', num2str(img_index), '.mat');
            file = load(file_name); %obtain all information about the img_index-th image

            f = [f, reshape(file.(f_name)', 1, size(file.(f_name), 1) * size(file.(f_name), 2))];

        end

        img_file{factor_index, num_moment} = f;
```

```
    end
end


img_file_non = cell(1, 12);  %for data of non-compressed images

for num_moment = 1:Num_moments %for each of 12 moments

    f_name = strcat('feat_non', num2str(num_moment));
    f = [];

    for img_index = 1:100

        file_name = strcat('ZM_file_', num2str(img_index), '.mat');
        file = load(file_name);

        f = [f, reshape(file.(f_name)', 1, size(file.(f_name), 1) * size(file.(f_name), 2))];

    end

    img_file_non{num_moment} = f;

end
```

After the data extraction, the next step is the comparing procedure. Here I took two strategies for comparison.

First, I divided the absolute value of difference between the JPEG compressed and corresponding uncompressed feature values, by the original uncompressed feature value, and plot it out. The MATLAB coding for this part is shown as follows.

```
for num_moment = 1:Num_moments

    n = NM(num_moment, 1);
    m = NM(num_moment, 2);

    for factor_index = 1:5

        figure_index_str = strcat(num2str(quality(factor_index)), num2str(num_moment));
        figure_index = str2num(figure_index_str);

        figure(figure_index);

        abs_comparison{factor_index, num_moment} = abs(img_file{factor_index, num_moment} - img_file_non{num_moment}) ./ img_file_non{num_moment};
        %put the outcome of the method 1 into the corresponding slots in
        %the cell method_1
        subplot(2,1,1);
        semilogy(abs_comparison{factor_index, num_moment}, '.');
        title(strcat('moment(', num2str(n), ',', num2str(m), '),compression factor', num2str(quality(factor_index))));
```

For this strategy, if the plot outcome is close to 0, it means the compressed feature value is almost equal to the uncompressed one; on the other hand, if the plot outcome is close to 1, it means the feature value is almost reduced to 0 after compression.

Second, I directly plot the compressed feature values versus the corresponding uncompressed ones. It immediately shows the proportional relation between the compressed values and uncompressed ones. The MATLAB coding for this part is shown as follows.

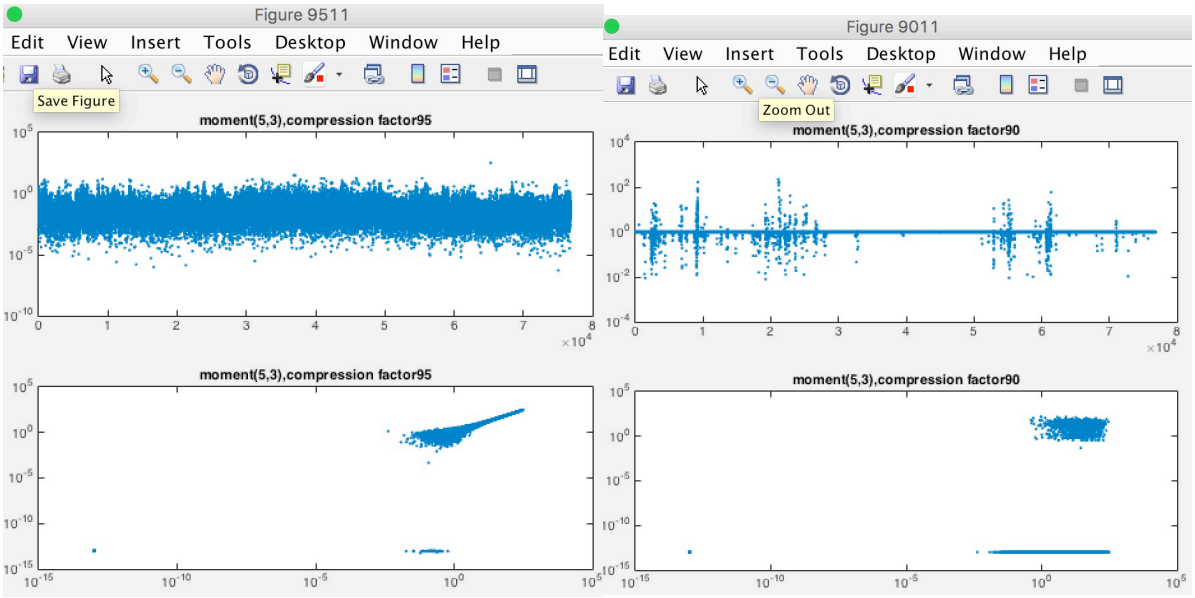
```
subplot(2,1,2);
loglog(img_file_non{num_moment}, img_file{factor_index, num_moment}, '.');
title(strcat('moment(', num2str(n), ',', num2str(m), '),compression factor', num2str(quality(factor_index))));
```

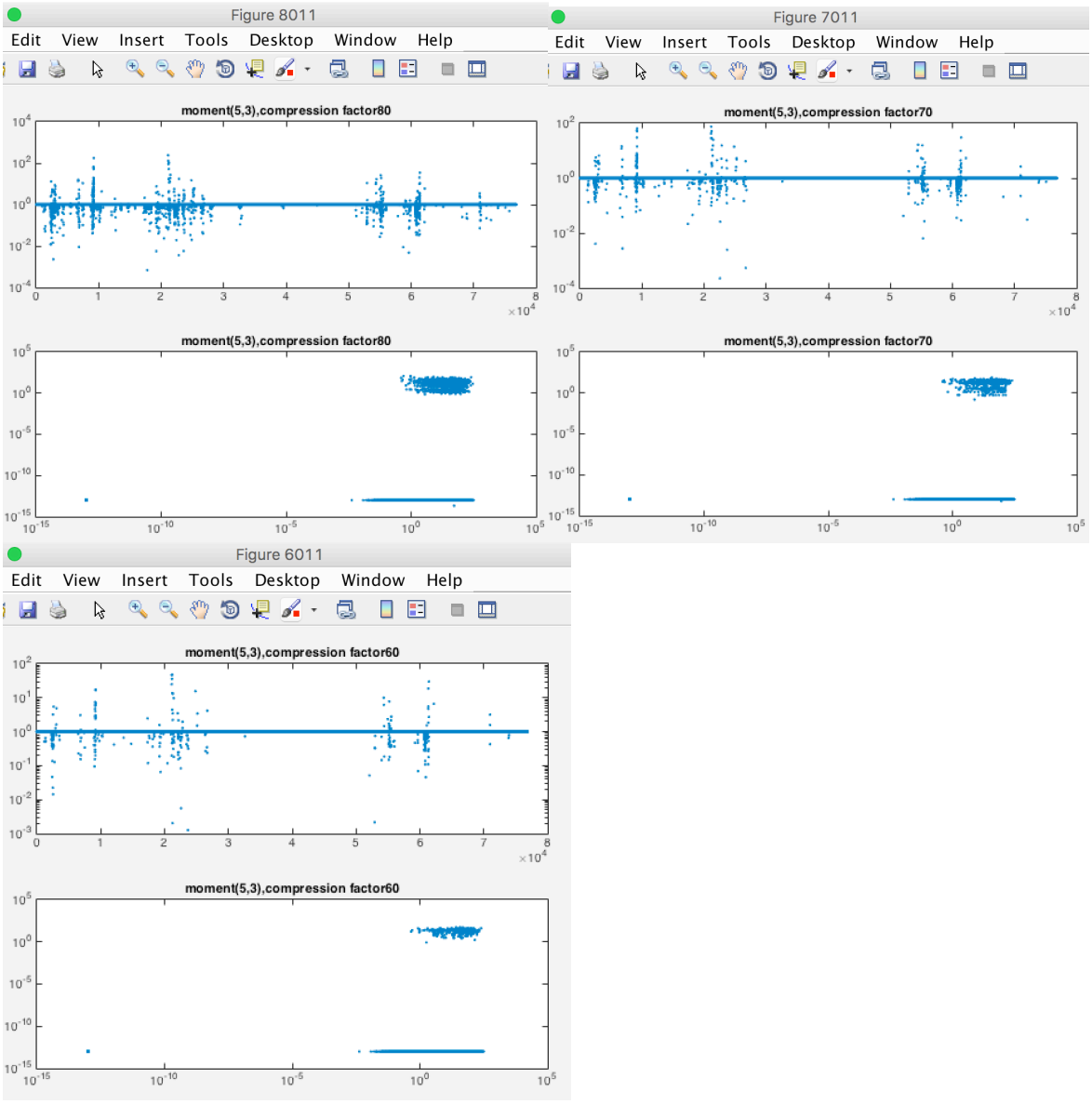
Also we can easily judge proportional factor from the plots. If plot spots tend to cluster to a line with slope of 1, then it shows that the compressed feature values are almost equal to the uncompressed ones; On the other hand, if the plot spots tend to cluster to the line with slope of 0, it means the feature values are almost reduced to 0.

As I said above, I extract all 76800 values for 100 images to make a basic data set of uncompressed or compressed feature values, and compare values in 12 Zernike moments with 5 different compression factors. So, here I totally obtained 60 plot figures for the comparison. In the following, I selected some represents to make further explanations.

In addition, in order to make it clear which compression factor and Zernike moment is engaged in a certain plot figure, I took a straightforward figure indexing method for plotting. In each figure, the first two digit indicates which compression factor is involved. And the last two or one digit indicates which Zernike moment is involved. For instance, if the figure index is 9511, it shows the comparison between compressed and corresponding uncompressed values with compression factor of 95, in the 11^{th} moment. Also, the specific (n, m) combination is shown is the plot title. For convenience, I combined plots of two strategies in the same figure, for a certain moment with a specific compression factor. The above plot is for the strategy, and the other plot is for the second strategy.

In general, compressed feature values have the same tendency for all moments. Let me take the moment 11 for instance. In the following, there are plots for the moment 11 with different compression factors.



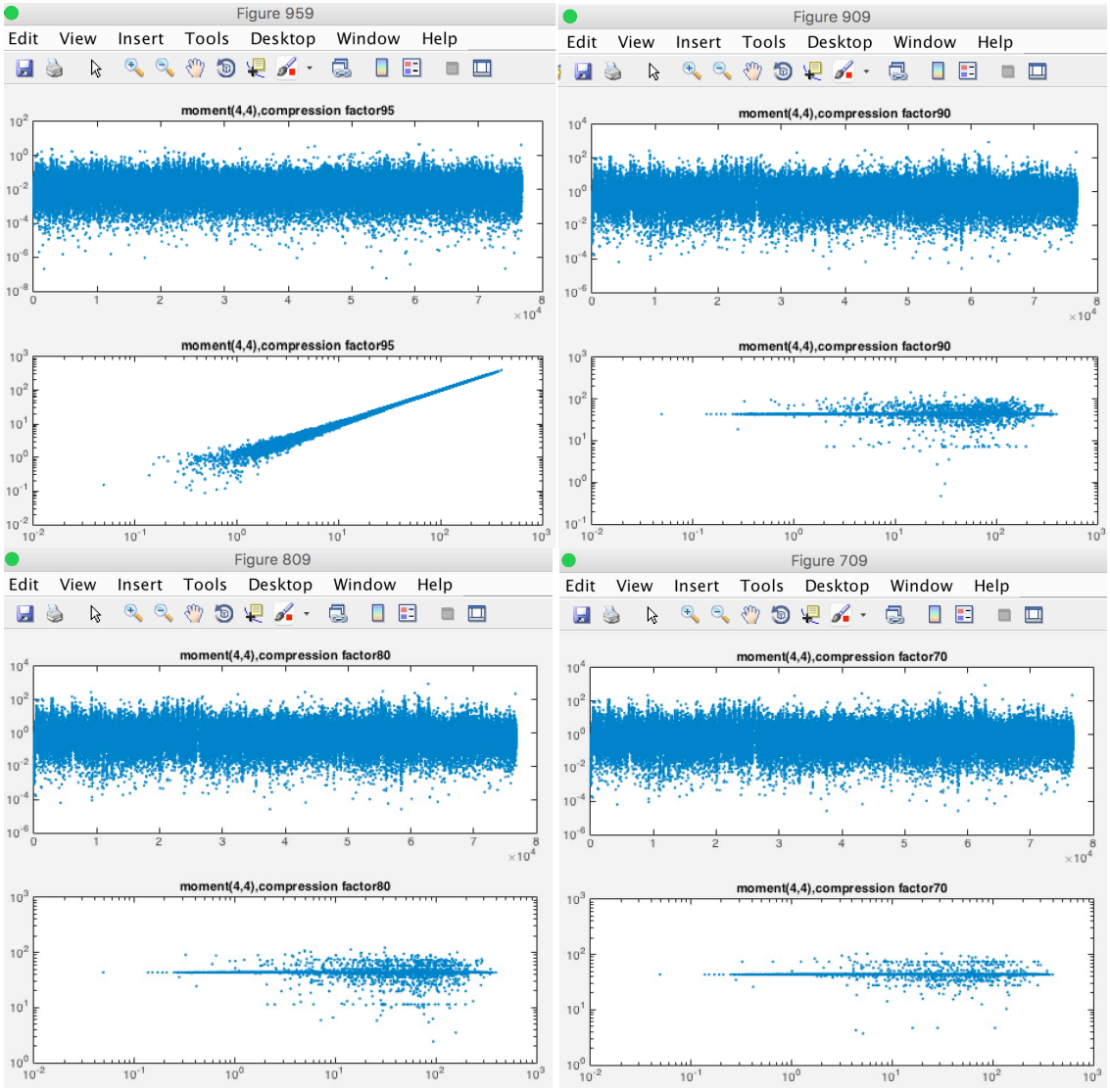


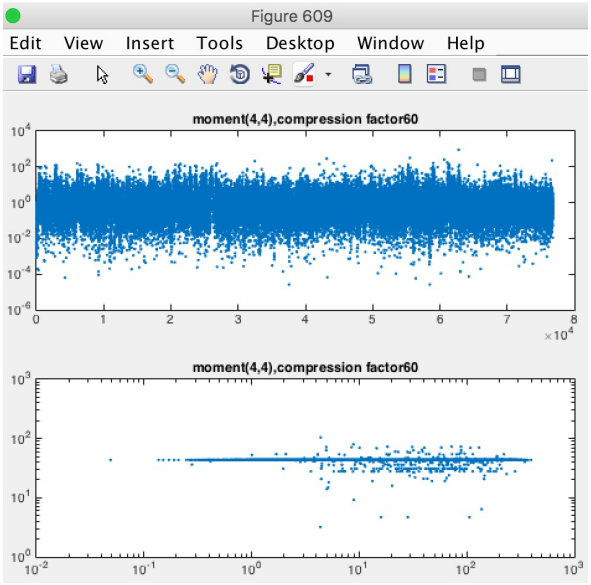
For the first strategy, we can see that outcome spots tend to cluster around 1 with compression factors from 90 to 60, which means the feature values are majorly reduced to 0 after compression, with minority of outcomes distributed at locations great than 1, indicating larger than the uncompressed values. With the factor goes down, we can see that more and more spots are located around 1, which means more feature values are reduced to 0 or around after compression with smaller factors. For the compression with the factor of 95, we can see that most outcome spots are located around values much close to 0. That is, the compressed values are almost the same to the uncompressed ones, with just a little change.

For the second strategy, we can see that outcome spots tend to cluster around 0, with a lot of them located higher than 0. However, most of these spots are below the line with slope of 1, which means these feature values are reduced after compression. Still, there are minor spots located above the line with slope of 1, indicating the feature values are enlarged after compression. With the factor goes down, we can see that more and more spots are located around 0, which also indicates more feature values are reduced to 0 or around after compression with smaller factors, just like what have been found in the first

strategy. For the compression with factor of 95, we can see that outcome spots are clustering quite tightly to the line with slope of 1, and this means that the compression with factor of 95 just make little change to the feature values.

However, there are some exceptions. For moments of 1, 3, 7 and 9, the outcome spots are located in a different way. Let me take the moment of 9 for instance. And the following shows the corresponding outcomes.





For the first strategy, we can see that outcom spots are located dispersively, with a lot of them loacted higher than 1. This means that a lot of feature values are enlarged after the compressions with factor from 90 to 60. In the second strategy, we see most spots are located higher than the line with slope of 1, which also indicates lager feature values after compression. And most of them are reduced to 43, not 0. Comparingly, in the above situation, with just a little part of feature values are reduced after compression. This situation is especially evident in the moment 1, with almost all feature values are enlarged after compression.

However, for the compression with factor of 95, it still keep the same outcome to the previous ones. The feature values are changed a little aftercompression.

So, we can conclude that for the factor of 95 condition, the compressed feature values are almost equal to the uncompressed feature values in all moments. And feature values are generally reduced to 0 with factor from 90 to 60 in most moments, with minor features are enlarged after compression, except for moments 1, 3, 7 and 9. In these moments, a lot of features are enlarged after compression, and the compression distribution is quite disperse.

All 60 figures are saved as PDF files for reference.

Actually, we can test this finding by seek out the mode values for all blocks under 5 different compression factors with 12 different moments, stored in the mode_cal cell. In the coding part, I searched for the mode values from the img_file, which as I said before, stores all compressed block feature values under different situations. The coding part is shown asfollows.

```
mode_cal = cell(5,12);  %set up a cell to calculate mode of compressed features with different compression factors, in different moments

for num_moment = 1:Num_moments

    n = NM(num_moment, 1);
    m = NM(num_moment, 2);

    for factor_index = 1:5

        mode_cal{factor_index, num_moment} = mode(img_file{factor_index, num_moment}); %store mode values in
                                             %slots with corresponding compression factors, in different moments
    end
end
```

And we can look into what really is in the mode_cal. Rows of mode_cal stand for 5 compression factors, with columns standing for 12 moments. As we can see, for the mode values, what really matters is the moment factor, not the compression one. In different moments, we have different mode values but keep the same values under different compression factor. And most moments have a mode value equaling to or almost close to 0, while moment 1, 3, 7, 9 have much great mode values, with 3.6777e+03, 206.7154, 249.1854, and 43.5901 respectively.

```
>> mode_cal

mode_cal =

  Columns 1 through 8

    [3.6777e+03]    [0]    [206.7154]    [5.1200e-15]    [3.1776e-14]    [2.2188e-13]    [249.1854]    [7.2416e-14]
    [3.6777e+03]    [0]    [206.7154]    [5.1200e-15]    [3.1776e-14]    [2.2188e-13]    [249.1854]    [7.2416e-14]
    [3.6777e+03]    [0]    [206.7154]    [5.1200e-15]    [3.1776e-14]    [2.2188e-13]    [249.1854]    [7.2416e-14]
    [3.6777e+03]    [0]    [206.7154]    [5.1200e-15]    [3.1776e-14]    [2.2188e-13]    [249.1854]    [7.2416e-14]
    [3.6777e+03]    [0]    [206.7154]    [5.1200e-15]    [3.1776e-14]    [2.2188e-13]    [249.1854]    [7.2416e-14]

  Columns 9 through 12

    [43.5901]    [2.7307e-14]    [1.0105e-13]    [9.7730e-14]
    [43.5901]    [2.7307e-14]    [1.0105e-13]    [9.7730e-14]
    [43.5901]    [2.7307e-14]    [1.0105e-13]    [9.7730e-14]
    [43.5901]    [2.7307e-14]    [1.0105e-13]    [9.7730e-14]
    [43.5901]    [2.7307e-14]    [1.0105e-13]    [9.7730e-14]
```

This is an interesting finding, and now let us trace back which blocks have mode values under a certain compression factor and moment, and inspect carefully what really is inside these blocks. In order to trace back where these blocks are, the first thing to do is to determine which image the block is in, with its row and column positions inside the corresponding image.

In the coding part, a 5x12 cell blk_pst is set up in order to store pick out blocks' indices, with mode values, under 5 different compression factors and 12 moments. As follows.

```
blk_pst = cell(5,12);  % a cell to store blocks' indices, with mode values,  in each moment with different compression factor

for num_moment = 1:Num_moments
    for factor_index = 1:5

        blk_pst{factor_index, num_moment} = find(img_file{factor_index, num_moment} == mode_cal{factor_index, num_moment});

    end
end
```

After all qualified blocks' indices determined, the we can determine the corresponding image indices, row and column positions. To store the corresponding image position for the all blocks with mode values, a 5x12 cell blk_info is set up, and the procedure is shown as follows.

```
blk_info = cell(5,12); %record position of mode values in each single image with different factors and moments.

%%%%%  img index search
for num_moment = 1:Num_moments
    for factor_index = 1:5

        blk_info{factor_index, num_moment} = zeros(1, length(blk_pst{factor_index, num_moment}));
        blk_temp = rem(blk_pst{factor_index, num_moment}, 768);  %rough blk position within an image
        img_temp = fix(blk_pst{factor_index, num_moment} / 768); %rough image position in 100 serial images

        v = find(blk_temp == 0);
        blk_info{factor_index, num_moment}(v) = img_temp(v);

        vv = find(blk_temp ~= 0);
        blk_info{factor_index, num_moment}(vv) = img_temp(vv) + 1;

    end
end

%%blk_pst: all blks' indices with mode values with different compression factors and
%different moment
%%blk_info: imgage indices of all blks with mode values in different
%%compression factor and different moment in.
```

After image indices determined, we still need indices found in blk_pst to search in the corresponding image, which is store in the blk_info, for the row and column positions. Because different images keep different sizes, and we first need to determine all image sizes. Just as follows.

```
%%%% image size track database construction
sizee = {};
for i = 1:100
    file_name = strcat('ZM_file_', num2str(i),'.mat');
    file = load(file_name);
    [r,c] = size(file.feat_non1); %get the i-th image size
    sizee = [sizee, [r,c]]; %store the image size data into the database
end
```

Then similarly, I set up two 5x12 cells of ROW and COL to store row and column indices for all blocks with mode values under different situations, within the corresponding images. The procedure is shown as follows

```
%%%%block position search
ROW = cell(5,12);
COL = cell(5,12);
for num_moment = 1:Num_moments
    for factor_index = 1:5

        [img_num, img_ind] = unique(blk_info{factor_index, num_moment});


        blk_temp = rem(blk_pst{factor_index, num_moment}, 768);


        img_temp = fix(blk_pst{factor_index, num_moment} / 768);

        Row = [];  %% to store row indices for all blks with mode values
        Col = [];  % to store column indices for all blks with mode values

        for i = 1:(length(img_num) - 1)  %% set aside the last image firstly

            img_index = img_num(i);   %read in the i-th image information
            file_name = strcat('ZM_file_', num2str(img_index),'.mat');
            file = load(file_name);
            [r,c] = size(file.feat_non1);  % obtain the i-th image's size

            blk_start = img_ind(i);
            blk_end = img_ind(i+1) - 1; %interval of blks that is in the i-th image

            row_temp_g = fix(blk_temp(blk_start:(blk_end-1)) /c);
            col_temp_g = rem(blk_temp(blk_start:(blk_end-1)), c); % firstly leave aside the lask blk,and estimate the rough row and col
            %indices for the rest blks by using the remainder and the quotient of division of
            %blk index divided by column number of the i-th image
```

```
col = zeros(1,length(col_temp_g));   %to store row indices of blks within the interval of the i-th image
row = zeros(1,length(col_temp_g));   %to store col indices of blks within the interval of the i-th image

c_0 = find(col_temp_g == 0);
c_n0 = find(col_temp_g ~= 0);

row(c_0) = row_temp_g(c_0); %if the remainder is 0, blk row index is the estimated row index, and the col index is the image col index
col(c_0) = c;

row(c_n0) = row_temp_g(c_n0) + 1; %if the remainder is not 0, blk row index is the estimated row index plus one, and the col index is
                                  %the estimated col index
col(c_n0) = col_temp_g(c_n0);


%now we calculate the last blk within the interval of the i-th
%image
if blk_temp(blk_end) == 0
    row = [row, r];
    col = [col, c];
else
   row_temp = fix(blk_temp(blk_end) / c);
   col_temp = rem(blk_temp(blk_end), c);
   if col_temp == 0
       row = [row, row_temp];
       col = [col, c];
   else
       row = [row, (row_temp + 1)];
       col = [col, col_temp];
   end
end
```

```
    Row = [Row, row]; % Row store all blk row indices
    Col = [Col, col]; % Col store all blk col indices
end

%%now think about the last image
i = length(img_num);
img_index = img_num(i);
file_name = strcat('ZM_file_', num2str(img_index),'.mat');
file = load(file_name);
[r,c] = size(file.feat_non1);

blk_start = img_ind(i);
blk_end = length(blk_info{factor_index, num_moment});

row_temp_g = fix(blk_temp(blk_start:(blk_end - 1)) /c );
col_temp_g = rem(blk_temp(blk_start:(blk_end - 1)),c);

col = zeros(1, length(col_temp_g));
row = zeros(1, length(col_temp_g));
c_0 = find(col_temp == 0);
c_n0 = find(col_temp_g ~= 0);
row(c_0) = row_temp_g(c_0);
col(c_0) = c;
row(c_n0) = row_temp_g(c_n0) + 1;
col(c_n0) = col_temp_g(c_n0);
```

```
       if blk_temp(blk_end) == 0
            row = [row, r];
            col = [col, c];
       else
            row_temp = fix(blk_temp(blk_end) / c);
            col_temp = rem(blk_temp(blk_end), c);
            if col_temp == 0
                row = [row, row_temp];
                col = [col, c];
            else
                row = [row, (row_temp + 1)];
                col = [col, col_temp];
            end
       end
       Row = [Row, row];
       Col = [Col, col];
       ROW{factor_index, num_moment} = Row;
       COL{factor_index, num_moment} = Col;
       %%row store all blk row indices, with mode values, within the i-th image
       %%col store all blk col indices, with mode values, within the i-th imag

   end
end
```

By now, we have have determine all block's row and column indices, with mode values, for all images. Let us take a random image for instance, to see how each pixel feature value changes in all qualified blocks under different situation.

```
%%%random image inspection

img = randi(100);  %randomly determine the image index
r = sizee{img}(1); %determine the size of image we are looking into
c = sizee{img}(2);
```

So firstly, a full pixel feature value diagram for the image under different situation should be obtained.

```
feat = ZM_JPEGFULL(pic_vector{1,img},param);
```

This procedure is operated by the sub-function ZM_JPEGFULL and store the outcome in the variant feat. The strategy is to set up a cell for all blocks within the image and take basic statistics of the median and standard deviation for each of them.

```
stat = cell(r,c);  % set up a cell to get the basic statistics of pixels within each 16x16 blk
```

Because we totally have 5 compression factors and 12 moments, in order to observe carefully each block under different situation, we also need to set up a cell of 5x12 to store statistics for each block under different situations.

```
for i = 1:1:r
    for j = 1:1:c

         stat{i,j} = cell(5,12);  %recording pixel features within one single blk
                                  %with 5 different
                                  %compression factors,
                                  %in 12 different
                                  %moments
    end
end
```

And then, store the median and standard deviation for each block under different situation in the corresponding cell.

```
for factor_index = 1:5
    for num_moment = 1: Num_moments

        i = find(blk_info{factor_index, num_moment} == img); %find out index involved with the image we are looking into

            if isempty(i) ~= 1 %% there exists image involved whose feature value equals to mode value under a certain
                               %% compression factor with a certain moment
               rr = ROW{factor_index, num_moment}(i);
               cc = COL{factor_index, num_moment}(i);  %find out all qualified blk's posistion within the image

               for ii = 1:1:length(i)

                    tmp = feat{factor_index}( ((rr(ii)-1)*16+1):1:((rr(ii)-1)*16+16), ((cc(ii)-1)*16+1):1:((cc(ii)-1)*16+16), num_moment );
                    ave = mean(tmp(:));
                    var = std(tmp(:));
                    stat{rr(ii), cc(ii)} {factor_index, num_moment}= [ave, var];

               end
            end
    end
end
```

Let us take a random block within the image for instance.

stat{11, 11}

1	2	3	4	5	6
[]	[]	[]	[]	[]	[]
[3.6777e+03,1.5947e-11]	[0,0]	[206.7154,7.6889e-13]	[5.1200e-15,1.9760e-29]	[3.1776e-14,2.5293e-29]	[2.2188e-13,5.0586e-28]
[3.6777e+03,1.5947e-11]	[0,0]	[206.7154,7.6889e-13]	[5.1200e-15,1.9760e-29]	[3.1776e-14,2.5293e-29]	[2.2188e-13,5.0586e-28]
[3.6777e+03,1.5947e-11]	[0,0]	[206.7154,7.6889e-13]	[5.1200e-15,1.9760e-29]	[3.1776e-14,2.5293e-29]	[2.2188e-13,5.0586e-28]
[3.6777e+03,1.5947e-11]	[0,0]	[206.7154,7.6889e-13]	[5.1200e-15,1.9760e-29]	[3.1776e-14,2.5293e-29]	[2.2188e-13,5.0586e-28]

stat | stat{11, 11}

stat{11, 11}

	7	8	9	10	11	12
	[]	[]	[]	[]	[]	[]
...	[249.1854,1.1391e-12]	[7.2416e-14,2.6558e-28]	[43.5901,1.4951e-13]	[2.7307e-14,1.4860e-28]	[1.0105e-13,5.0586e-28]	[9.7730e-14,3.9204e-28]
...	[249.1854,1.1391e-12]	[7.2416e-14,2.6558e-28]	[43.5901,1.4951e-13]	[2.7307e-14,1.4860e-28]	[1.0105e-13,5.0586e-28]	[9.7730e-14,3.9204e-28]
...	[249.1854,1.1391e-12]	[7.2416e-14,2.6558e-28]	[43.5901,1.4951e-13]	[2.7307e-14,1.4860e-28]	[1.0105e-13,5.0586e-28]	[9.7730e-14,3.9204e-28]
...	[249.1854,1.1391e-12]	[7.2416e-14,2.6558e-28]	[43.5901,1.4951e-13]	[2.7307e-14,1.4860e-28]	[1.0105e-13,5.0586e-28]	[9.7730e-14,3.9204e-28]

As we can see, in the first row, which represent the situation with factor of 95, since most compressed values are almost equal to the uncompressed ones, we cannot search any qualified pixel values according to the mode value, and of course, we cannot calculate the median and the standard deviation based on the mode values. With other factors, however, we can see that for each moment, the median value is equal to the mode value we have obtained before, and the deviation is quite small. This means that, for factor of 90 to 60, each pixel is compressed equivalently, resulting in almost the same feature values in on single block. More exactly, all pixel value almost equals to the mode values after the compression with factors of 90 to 60.

Think more, how should the block feature value changes in different moments with the same compression factor? We can firstly check if each block is compressed to the mode value under a certain compression factor and then to check if it is still compressed to the mode value in other 12 moments. If this is true, then we can say the mode value change for a qualified block in different moment is [3677, 0, 206, 0, 0, 0, 249, 0, 43.5, 0, 0, 0].

In order to simply the processing, I firstly calculate how many compressions does each block go through under different situation.

```
img = randi(100);  %randomly determine the image index
r = sizee{img}(1); %determine the size of image we are looking into
c = sizee{img}(2);
cal = cell(1,12);  %set up a cell to store distribution of blks with mode values in 12 different moments
for num_moment = 1:Num_moments   %for each moment, different mode values
    cal_temp = zeros(r,c); % single distribution of blks with mode value in one moment
    for factor_index = 1:5  %we look into different compression factor

        i = find(blk_info{factor_index, num_moment} == img); %find out index of the image we are looking into

        if isempty(i) == 1
            cal{1, num_moment} = cal_temp;
        else
            rr = ROW{factor_index, num_moment}(i);
            cc = COL{factor_index, num_moment}(i);

            for ii = 1:length(rr)
                cal_temp(rr(ii), cc(ii)) = cal_temp(rr(ii), cc(ii)) + 1;
            end

            cal{1, num_moment} = cal_temp;
        end
    end
end
%%%this part of code is to determine the statistics of how many
%%%compressions each block go through
%%%to see if blks are the same to the mode values under a certain compression factor
%%% and within different moments
%%%I compared blks one by one to see if there exists different blk values
%%%under the same compression factor but in different moments
%%% [3500,0,206.7,0,0,0,249,0,43.6,0,0,0] is the resulting feature value
%%% distribution that can be obtained.
```

The calculation outcome is quite large to show, and the following shows a snapshot of it. It calculate how many compressions totally does each block gothrough.

4	4	4	4	4	4
3	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	1	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	2	4
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4

Then, we can check if the calculations under the same compression factor are the same in different moment. I compared all other moment calculations with the that in moment 1. If it is true, then mark the corresponding slot with 1, and with 0 if the assumption is not founded. The coding part is shown as follows.

```
inspect = zeros(5,12);
for factor_index = 1:5
    inspect(factor_index, 1) = 1;
    for ind_moment = 2:Num_moments
        a = find(cal{1,1} ~= cal{1, ind_moment});
        if isempty(a) ~= 1
            inspect(factor_index, ind_moment) = 0;
        else
            inspect(factor_index, ind_moment) = 1;
        end
    end
end
```

And the outcome is as follows. So we conclude that the assumption above is founded.

inspect

5x12 double

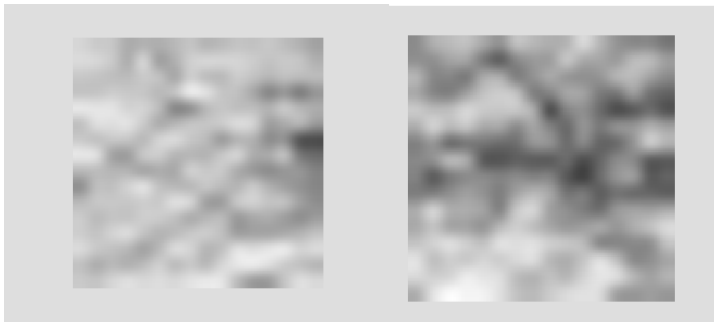
1	2	3	4	5	6	7	8	9	10	11	12	1
1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	
1	1	1	1	1	1	1	1	1	1	1	1	

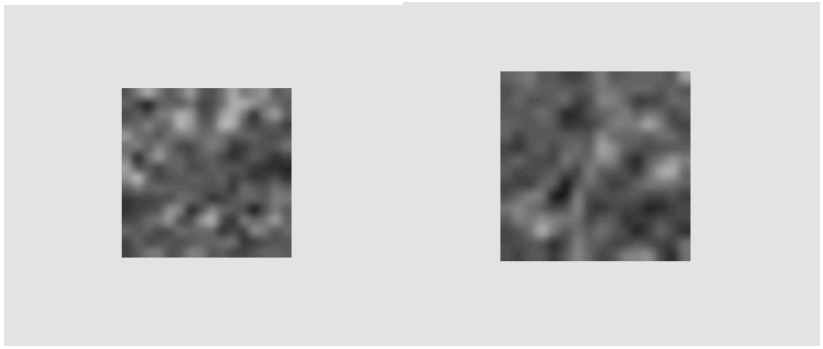
In order to have a closer inspection to details of qualified blocks with mode value, I set up a directory to store all block image snippets, with 60 different sub-directories to store the image snippets under corresponding situations.

```
%STORE ALL QUALIFIED BLKS PIXELS UNDER A CERTAIN SITUATION AND SHOW THEIR IMAGE DETAILS
cd /Users/caiyuhao/Desktop/newcode;
mkdir(file_name);  %create image directory to store blk pixels with different situations
cd(file_name); %change to the new directory
for factor_index = 1:5
    for num_moment = 1:Num_moments
        sub_file_index = [num2str(quality(factor_index)), num2str(num_moment,'%02d')];
        mkdir(sub_file_index); %create sub-directory to store blk pixels
        cd(sub_file_index); %change to the new sub-directory

        i = find(blk_info{factor_index, num_moment} == img); %find out involved with the image we are looking into

        if isempty(i) ~= 1 %there exisits image involved whose feature value equals to mode value under a certain situation

            rr = ROW{factor_index, num_moment}(i);
            cc = COL{factor_index, num_moment}(i); %find out all qualified blk's position within the image

            for ii = 1:length(i)

                tmp = feat( ((rr(ii)-1)*16+1):1:((rr(ii)-1)*16+16), ((cc(ii)-1)*16+1):1:((cc(ii)-1)*16+16) ); %blk pixel extraction
                blk_name = [num2str(rr(ii)) '|' num2str(cc(ii)) '.png']; %blk pixel image creation
                imwrite(tmp,blk_name);   %16x16 size blk image

                tmp2 = imresize(tmp, 10);      %resized blk image creation, in order to have a clearer inspection to blk pixels
                blk_name2 = ['**' num2str(rr(ii)) '|' num2str(cc(ii)) '**.png'];
                imwrite(tmp2,blk_name2); %enlarged blk image

            end
        end
        cd ..;
    end
end
```

The directory is created according to the random image inspected, named ‘ucid*****.png’. Within the directory, there are 60 sub-directories. In order to discriminate different situations, I named each sub-directory with its index, which combined with compression factor as the front two digits and moment as the later two digits. All qualified blocks are named with row index as the front part and column index as the later, separated by ‘|’.

Take a random sub-directory for instance, we can see that some blocks share the similar greyscale distribution, and distributions differ within several block groups.

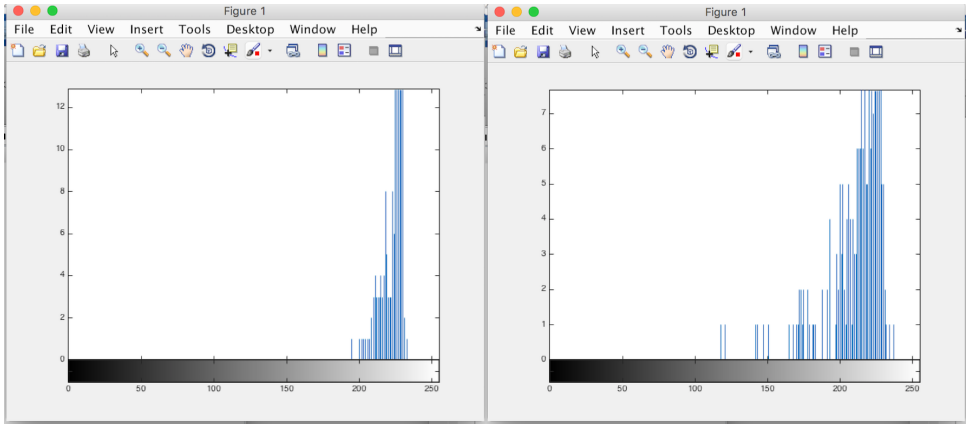


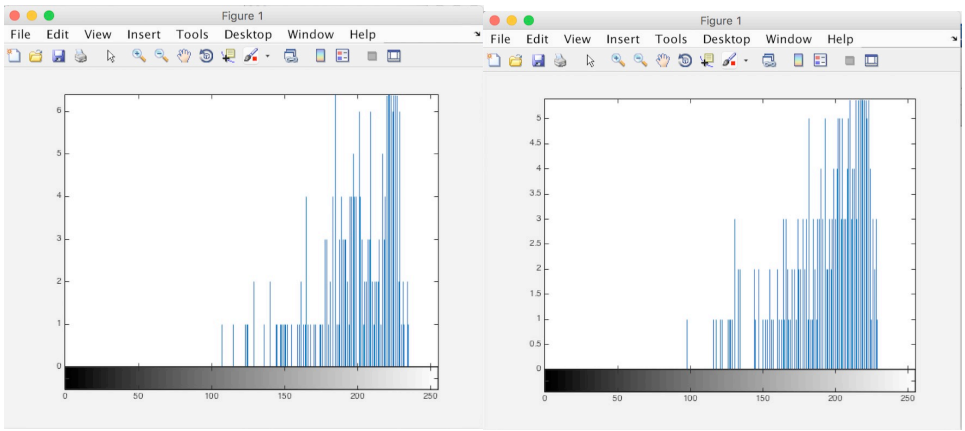


Let us turn to another respect, histograms of qualified block pixels. Because of huge number of blocks in each situation, I cannot go through all qualified blocks under all situations, here I take a random one for instance.

```
%Because of huge number of blks, we cannot go through all qualified blks
%under all situation, here let us take a random situation for instance

factor_random = quality(randi(5,1,1));
moment_random = randi(12,1,1);
cd([num2str(factor_random), num2str(moment_random,'%02d')]);

for first_name = 1:r

    %%% we get the first factor of block index%%%%%
    fi = dir([num2str(first_name),'|*.png']);

    for sec_name = 1:length(fi)

        blk_name = fi(sec_name).name;  % get the name of blk xx|xx.png
        img_data = imread(blk_name);
        imhist(img_data);
        figure(1);
        print(gcf,'-dpng',['*', blk_name]);

    end
end

cd ..
```

Figures of blocks are name with the corresponding block's name, with a star mark '*' as prefix. From the histograms, the similar finding is that some blocks share the similar histograms, with different histograms within several groups of blocks.





The coding part is attached to the mail for test.