Final Project Report

ECES 680 - Media Forensics & Security

Shangqi Wu - 16/3/17

Topic Selection and Report Outline

I chose the paper of "A generalized Benford's law for JPEG coefficients and its applications in image forensics" for this project. So that this report will concentrate on statistics of JPEG quantized image coefficients.

Content of this report will cover theoretical part and experiment part. The first, theoretical part mainly provides a brief introduction and summary of the paper used. Then experiment part will introduce the environments, materials chosen for test and validation, and my Matlab code implementing the algorithm described in this paper will also be attached in one section. At last, some program output and data will be collected and analyzed, and hopefully we will reach an evaluation on the algorithms researched and proposed by the authors of this paper.

1. A Brief Introduction of Paper

This paper adapted a Benford's law, which originally comes from mathematics discipline, for an innovative statistics model for image forensics analysis. Surprisingly, JPEG picture coefficient distribution matches this statistics law to certain extent, and the innovative model is derived from this law and proposed in this paper.

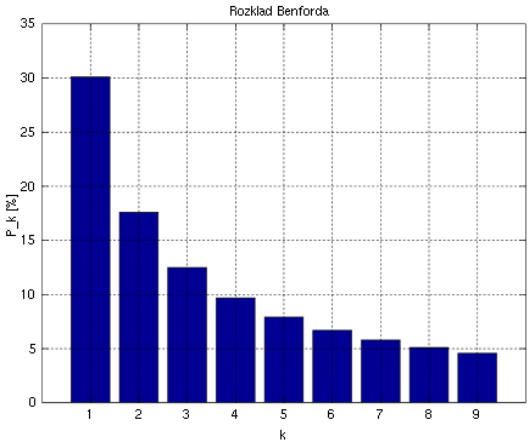
1.1. Brief Introduction to Benford's Law

Benford's law is often referred as first-digit law, which is a generalized observation of real-world natural number data. Frank Benford observed statistics reports and found that by only focus on first digit of numerical data, the lower first digit is, more frequent the digit appears. The distribution of first digit follows an logarithmic formula, and soon he proposed a mathematic model describe this phenomenon.

The formula of this model is:

$$p(x)=\log_{10}\left(1+\frac{1}{x}\right)$$

An example graph of the distribution is:



1.2. DCT Coefficient Distribution and Benford's Law

The authors believed image coefficients are also natural numerical data in real-world, so that DCT-block coefficients should also follow this law, though it may not follow Benford's law perfectly, there will be a match somehow.

The authors intended to verify Benford's law with block-DCT coefficients and JPEG coefficients respectively. In this section, the author verified the law with DCT coefficients. To clarify, the author explained JPEG compression procedure and stated difference between 2 coefficients - DCT coefficients are numbers before quantization while JPEG coefficients are those have already been quantized.

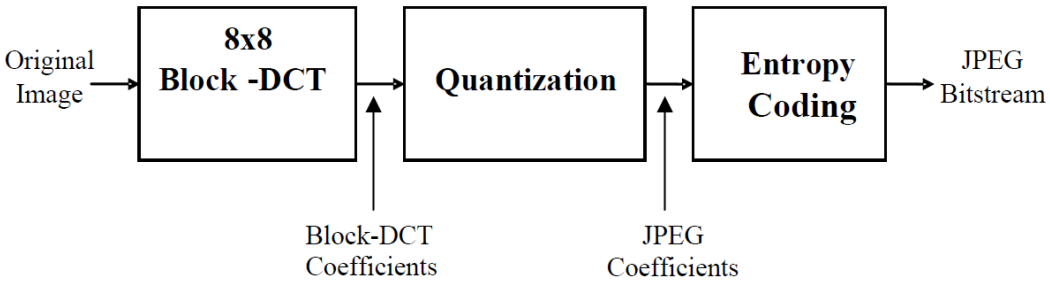


Figure 1.2.1 JPEG Compression Process

To eliminate the affect of image compression, which alters natural data, authors experimented with 1338 image from UCID database, which contains only uncompressed color digital images. Meanwhile for better comparison and easier implementation, authors converted color images into gray scale bit maps.

Since the DC components are too large and much less than number of AC components, this paper only took AC coefficients, i.e., every element in a 8×8 block other than the first element, into consideration. The statistics of DCT coefficients in 1338 uncompressed images, along with comparison with Benford's law is presented in the graph below.

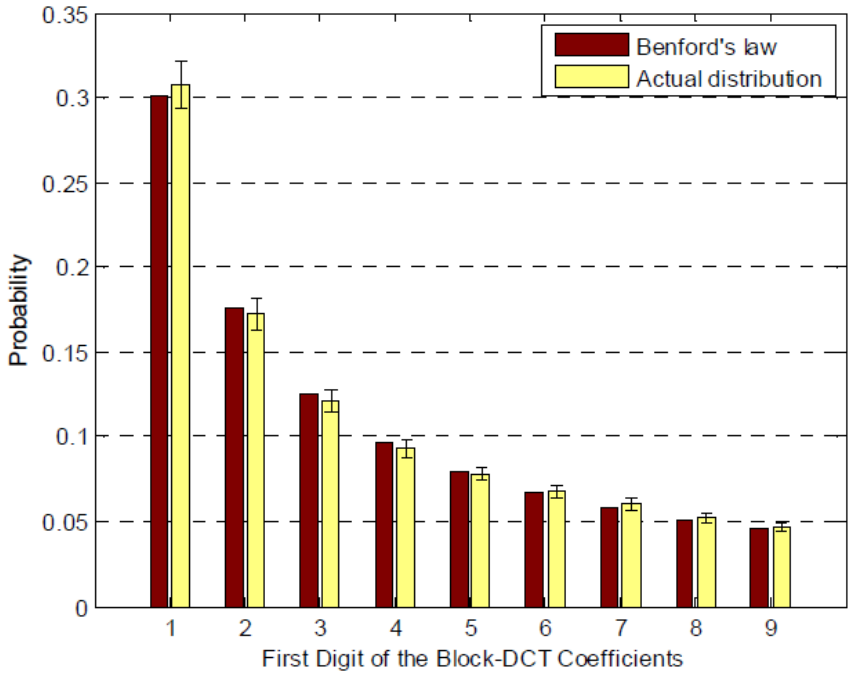


Figure 1.2.2 Statistics of DCT coefficients and Benford's law

The result is very close to Benford's law, and the error bar on actual distribution shown the distribution pattern is robust among all images.

1.3. JPEG Coefficients and Benford's Law

Then the authors moved to research on JPEG coefficients. Those numbers are different from previous section since they are artificially quantized, so that JPEG coefficients may represent a different distribution pattern.

So that in this section, authors performed quantization after DCT transform, and then extracted all quantized AC components. The results of the image compressed by quality factor of 50 along with comparison with Benford's law presented in this following figure.

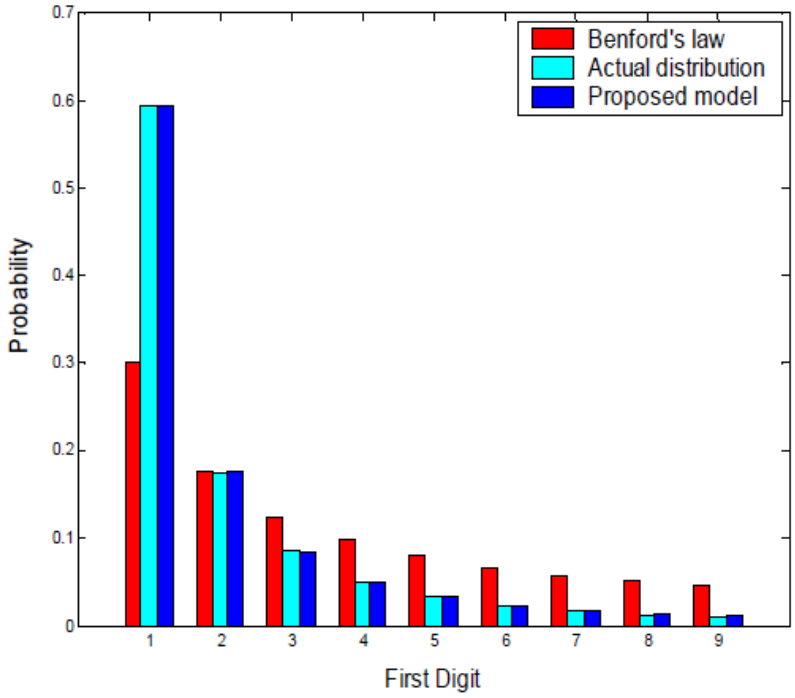


Figure 1.3.1 Statistics of JPEG coefficients and Benford's law

From the figure above, we can reach a conclusion that obviously JPEG coefficients do not follow the Benford's law. However authors still observed that the distribution pattern still somehow possibly matched Benford's law, with a few adjusted parameters. So that a modified model is proposed in the paper:

$$p(x)=N\log_{10}\left(1+\frac{1}{s+x^q}\right)$$

After applying the new first digit law, distributions of JPEG coefficients appeared to be linear in logarithmic view. The following graph showed distribution of images in UCID database.

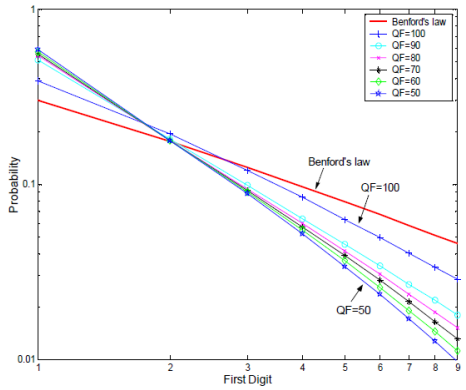


Figure 1.3.2 JPEG coefficient logarithmic view

1.4. Applications of this Model in Image Forensics

There are 3 applications proposed explicitly in this paper, detection of JPEG compression for bitmaps, estimation of quality factors of JPEG images and detection of JPEG double-compression trace.

The paper provided a procedure to detect JPEG compression. The first step is to take the bitmap for detection, then perform DCT transform and JPEG quantization. At last, first digit distribution of JPEG coefficients are extracted and presented in a logarithmic graph to show if it is linear.

The paper also provides several comparisons between uncompressed image bitmap and compressed bitmaps with different quality factors. One example of quality factor of 90 is presented in following graph.

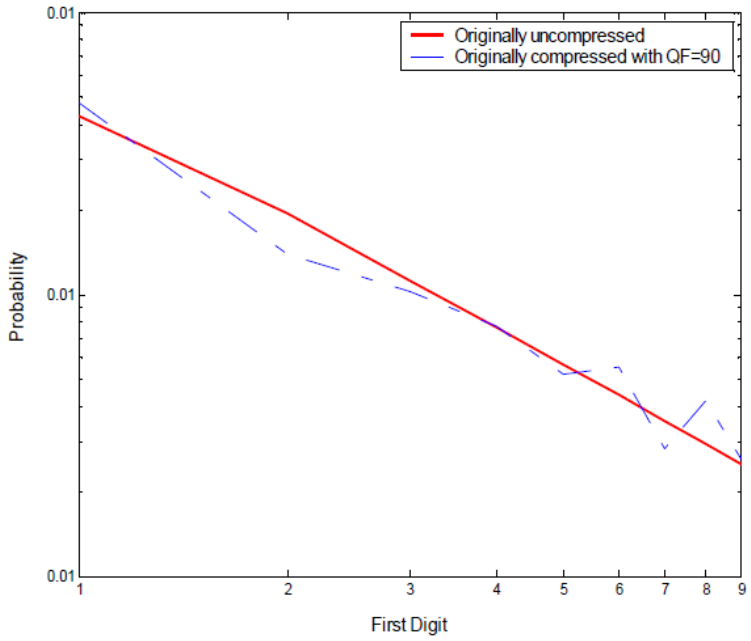


Figure 1.4.1 Distributions of uncompressed and compressed image

Quality factor estimation can be done by following steps: first, decompress JPEG image and get its JPEG coefficient first digit distribution, then generate decompressed bitmap with different quality factors, and pick the quality factor that generates the most fit distribution to the JPEG file.

It can be done in an iterative approach, by altering quality factor with certain step within a range, and retroactively update the range and its step.

The following graph provides and example of line-fitting result with different quality factors.

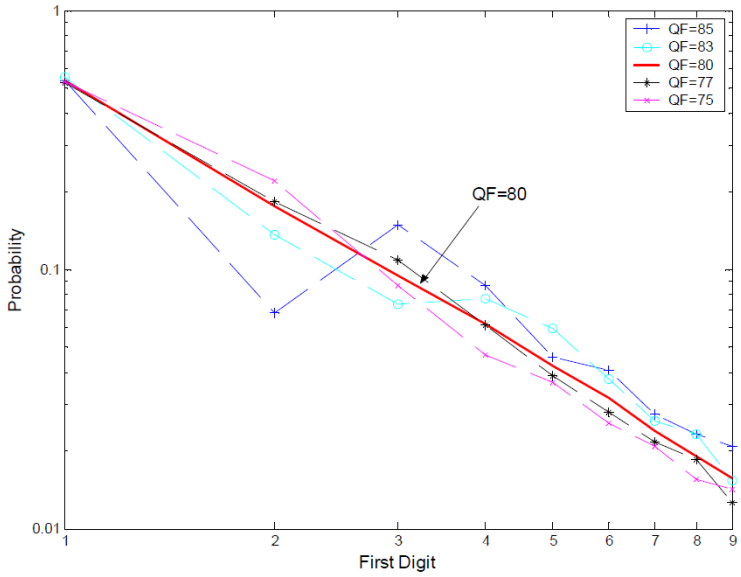


Figure 1.4.2 Lines of distributions with different quality factor

The third application mentioned in the paper is double JPEG compression. It can be done in a similar with as first application, and basically they are based on the same theory. The point is if the image is double compressed, the distribution will severely violates the law. The following figure shows how will double compressed image distributions be like, compared with uncompressed images.

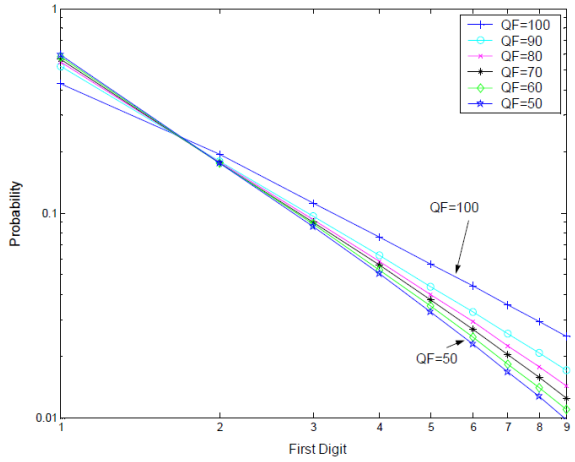


Figure 1.4.3 Distributions of single compressed files

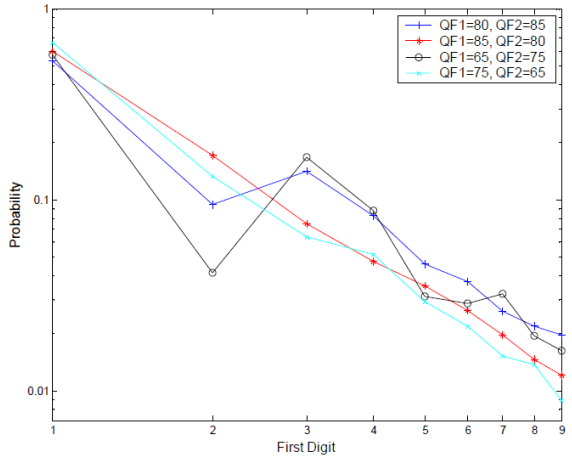


Figure 1.4.4 Distributions of double compressed files

1.5. Conclusions and Analysis

The approach proposed by this paper accurately reports image compression traces. It can be utilized widely in image forensics area.

However the model can still be improved since the formula introduced contains extra parameters, N, s and q, which still need to be trained and picked to fit an image fit most in order to verify the linear representation. In the future, it is worthwhile to do more research to generalize those parameter so that the model can fit different images automatically.

Also, there are only 3 applications proposed in this paper, it is possible to research and develop more applications based on this modified first digit law.

2. Algorithm Implementation

The algorithm introduced in the paper is straightforward, so a function code which takes image bitmap and quantization table and returns first digit distribution is attached in following section in this report.

Also, scripts used for verifying 3 applications will also be attached. Since this paper mainly discusses JPEG coefficients, so functions processing DCT coefficients are not implemented.

2.1. JPEG Coefficient Extracting Function

```
function digitStats = JPEGcoeffStats(image, quaMatrix)
% Input
%     image: input image bitmap in gray scale
%     quaMatrix: input JPEG quantization table
% Return
%     digitStats: first digit distributions
%                 of JPEG coefficients

    % Get size information.
    blockSize = 8;
    [numRow, numCol] = size(image);

    % Perform DCT transform and quantization.
    image = double(image) - 128;
    image = blkproc(image, [blockSize blockSize], ...
                                        'dct2(x)');
    image = blkproc(image, [blockSize blockSize], ...
                                        'round(x./P1)', quaMatrix);

    digitStats = zeros(1, 9);
    % Get distribution of JPEG coefficients.
    for i = 1 : numRow
        for j = 1 : numCol

            % Skip DC components.
```

```
            if (rem(i - 1, 8) ~= 0 || rem(j - 1, 8) ~= 0) && ...
                        image(i, j) ~= 0

                digit = getFirstDigit(image(i, j)) + 1;
                digitStats(1, digit) = digitStats(1, digit) + 1;

            end

        end
    end

    % Calculate average / probability.
    digitStats = digitStats ./ sum(digitStats);

end
```

2.2. First Digit Helper Function

```
function digit = getFirstDigit(number)
% Input
%     number: input integer
% Return
%     digit: most significant digit of the number

    % Get absolute value in case of negative input
    number = abs(number);

    % Remove less significant digit.
    while number >= 10
        number = floor(number / 10);
    end

    % Round to integer value.
    digit = floor(number);

end
```

3. Algorithm Verification and Experiments

In this section, I intend to utilize the Matlab function implemented in previous section, to verify the algorithms described in the paper. I will pass some images to the function and try to reconstruct some results or thoughts mentioned in the paper.

3.1. Experiment Environment

This experiment is conducted with Matlab 2015b version, and the codes are tested on both Windows and Mac OS X platforms. All codes are in Matlab language, and no additional tool box needed.

To get reliable result, I downloaded images from UCID version 2 at NJIT. The uncompressed image involved in this experiment is No.6. Compressed images are obtained by compressing with Matlab imwrite function by different quality factors. Corresponding quantization tables are obtained by software JPEGsnoop.

3.2. Script to Verify JPEG Compression for Bitmaps

```
clear;

% Read uncompressed image.
image = imread('image/original.tif');
image = rgb2gray(image);

% Retrieve and plot first digit distribution.
stats = JPEGcoeffStats(image, getQuanMatrix(100));
figure, loglog(0 : 9, stats, 'r-*');
hold on;

% Repeat the process for compressed images.
for i = 50 : 10 : 90

    image = imread(['image/compress', num2str(i), '.jpg']);
    stats = JPEGcoeffStats(image, getQuanMatrix(100));
    loglog(0 : 9, stats, '-*');
    hold on;

end

image = imread(['image/compress', num2str(95), '.jpg']);
stats = JPEGcoeffStats(image, getQuanMatrix(100));
loglog(0 : 9, stats, '-*');
hold on;

image = imread(['image/compress', num2str(99), '.jpg']);
stats = JPEGcoeffStats(image, getQuanMatrix(100));
loglog(0 : 9, stats, '-*');
hold on;

legend('uncompressed', 'Q = 50', 'Q = 60', 'Q = 70', 'Q = 80', ...
            'Q = 90', 'Q = 95', 'Q = 99');
hold off;
```

3.3. Output and Analysis of JPEG Compression Detection

The script above is to perform JPEG compression detection. It opens original version and compressed versions of a picture, then retrieves JPEG coefficient distributions by compression with quality factor of 100. According to this paper, distribution of uncompressed image is supposed to be linear under logarithmic scale. Following graph shows different distributions among different quality factors.

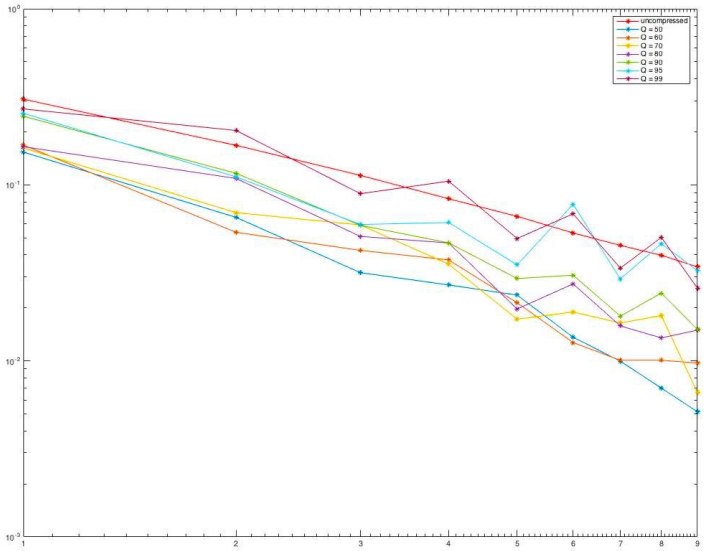


Figure 3.3.1 Different distributions under different quality factors

The red line most straight in the graph is the distribution of uncompressed image. While other distributions from compressed images seem to be segmented by several slopes, which show different patterns from uncompressed one. It is obvious to visually examine if an image is compressed, and most classifier algorithms would detect JPEG compression from this distribution easily and reliably.

3.4. Script to Estimate Quality Factor

```
clear;

for quality = 50 : 10 : 90
    % Pick a compressed image with current quality factor.
    image = imread(['image/compress', num2str(quality),'.jpg']);
```

```
    figure;
    for i = 50 : 10 : 100
        % Retrieve distribution under compression with
        % new quality factor.
        stats = JPEGcoeffStats(image, getQuanMatrix(i));
        loglog(0 : 9, stats);
        hold on;

    end

    legend('Q = 50', 'Q = 60', 'Q = 70', 'Q = 80', ...
                                        'Q = 90', 'Q = 100');
    hold off;

end
```

3.5. Output and Analysis of Quality Factor Estimation

The script above open an compressed image, with quality factor increasing from 50 to 90, and recompress bitmap with different quality factors from 50 to 100 and generate a graph of coefficient first digit distribution. According to this paper, distribution of recompressed image with the same quality factor is supposed to be linear under logarithmic scale, while other distributions are segmented lines with different slopes. Following graphs are selected outputs of distributions.

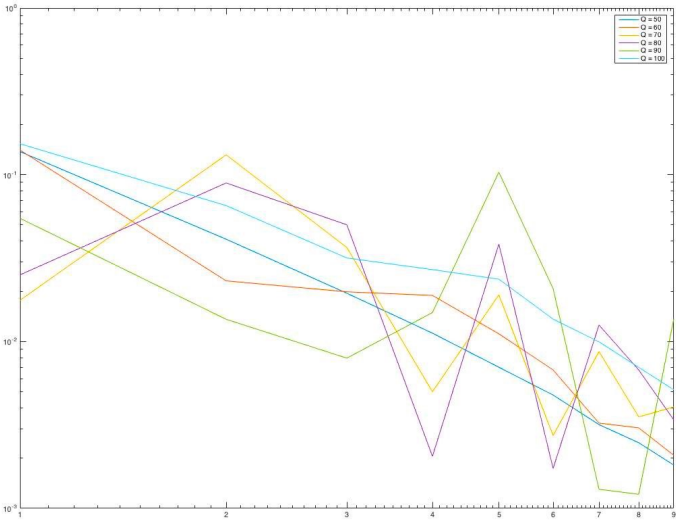


Figure 3.5.1 Quality factor of 50

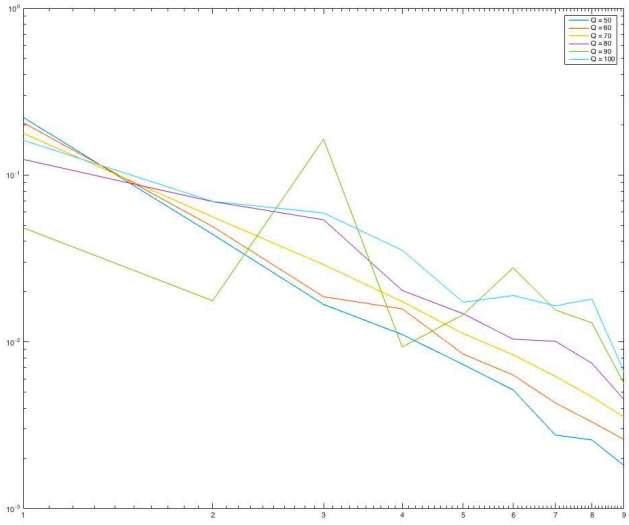


Figure 3.5.2 Quality factor of 70

Figure 3.5.1 demonstrated distributions under different quality factors after recompression. The opened image this time is of quality factor of 50, and the dark blue line in this graph stands for recompression quality factor of 50. The dark blue line is the only distribution appears to be linear under logarithmic view.

Figure 3.5.2 demonstrated similar distribution. At this time, the opened image is of quality factor of 70. The yellow line represents distribution of recompression with quality factor of 70. Still, this line is the only linear distribution under logarithmic view.

From 2 figures above, we can visually convinced that this statistics model is accurate enough to estimate quality factor of a JPEG compressed image.

However, this application still needs improvement since this model requires users to configure an appropriate set of parameters for the formula in the new model proposed. As research goes deeper, more accurate data will be required and visual examination may not be good enough. To find how well the distribution fitted to ideal value, user has to spend much time to config parameters, which always alter from different images. The authors did not mention how to config parameter effectively made it even worse.

There is another limitation of this application is that this experiment is conducted under the assumption of using standard JPEG quantization tables. As stated by authors, if original compressed image does not work with standard quantization table, the algorithm would not be able to estimate correct quality factor.

3.6. Script to Detect Double JPEG Compression

```
clear;

image = imread('image/original.tif');
image = rgb2gray(image);

% Single compression.
figure;
for quality = 50 : 10 : 100

    fileLen = myOwnJPGEncoder(image, getQuanMatrix(quality));
    stats = directJPEGcoeff();
    loglog(0 : 9, stats, '-*');
    hold on;

end

legend('Q = 50', 'Q = 60', 'Q = 70', 'Q = 80', 'Q = 90', 'Q = 100');
hold off;
```

```
% Double compression.
figure;
for quality = 60 : 10 : 100

    % Compress image with current quality factor.
    fileLen = myOwnJPGEncoder(image, getQuanMatrix(quality));
    tmpImage = myOwnJPGDecoder();
    % Recompress image with lower quality factor by 10.
    fileLen = myOwnJPGEncoder(tmpImage, ...
                          getQuanMatrix(quality - 10));
    stats = directJPEGcoeff();
    loglog(0 : 9, stats, '-*');
    hold on;

end

legend('Q1 = 60, Q2 = 50', 'Q1 = 70, Q2 = 60',...
               'Q1 = 80, Q2 = 70', 'Q1 = 90, Q2 = 80', ...
                    'Q1 = 100, Q2 = 90');
hold off;
```

3.7. Helper Function to Retrieve JPEG Coefficient Directly

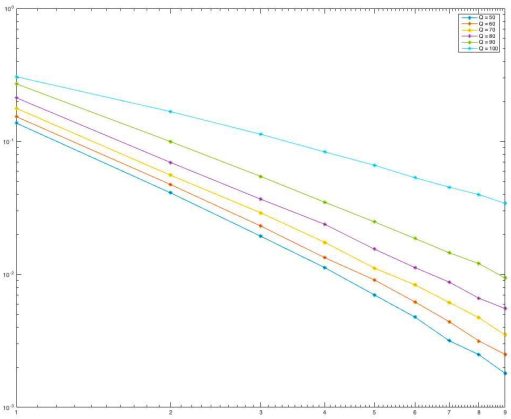
```
function digitStats = directJPEGcoeff()

    % Recover zigzag image from entropy decoder
    [numRow,numCol,blockSize,~,zImage] = JPEG_entropy_decode('');
    image = ones(numRow, numCol);
    rowIndex = 1;
    colIndex = 1;
    % Recover image from zigzaged image
    for i = 1 : size(zImage, 1)
        image(rowIndex : rowIndex+blockSize-1, ...
                        colIndex : colIndex+blockSize-1) = ...
                                  Vector2ZigzagMtx(zImage(i, :));
        colIndex = colIndex + blockSize;
        if colIndex > numCol
            colIndex = 1;
            rowIndex = rowIndex + blockSize;
        end
    end

    digitStats = zeros(1, 10);
    % Get distribution of JPEG coefficients.
    for i = 1 : numRow
```

```
        for j = 1 : numCol

            % Skip DC components.
            if (rem(i - 1, 8) ~= 0 || rem(j - 1, 8) ~= 0) && ...
                    image(i, j) ~= 0

                digit = getFirstDigit(image(i, j)) + 1;
                digitStats(1, digit) = digitStats(1, digit) + 1;

            end

        end
    end

    % Calculate average / probability.
    digitStats = digitStats ./ sum(digitStats);

end
```

3.8. Output and Analysis of JPEG Double Compression Detection

This section is similar to single compression detection, however the slight difference here required me to modify a lot of code from that section.

The theory of first section is basically force a second JPEG compression to make distribution violates logarithmic law. In this section, there is no need to force compression one more time, instead, we can examine the first digit distribution by looking into JPEG files directly. This goal can be achieved by customized JPEG decoder. After decoding the entropy code, instead of continuing inverse quantization and inverse DCT transform, we can simply stop and observe distribution.



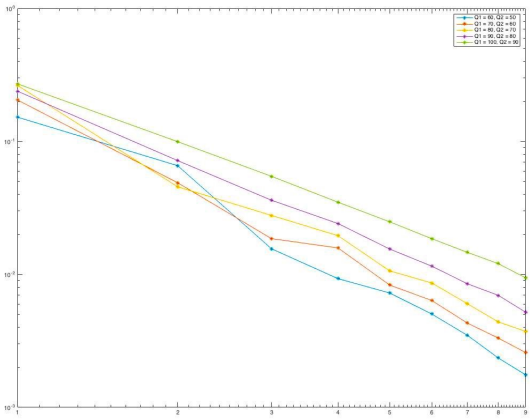


Figure 3.8.1 Single compressed image distributions

Figure 3.8.2 Double compressed image distributions

According to the paper and previous sections, single compressed files should observe first digit law, while double compressed files will violate this law. Above figures represents the distributions.

From figure 3.8.1, all distributions are linear in logarithmic view, which satisfies the expectation of first digit law. From 3.8.2, except the green line, all other distributions are not linear. Since the green line stands for the first quality factor of 100, i.e., the image is not compressed during first stage, in fact it represents the distribution of an image singly compressed with quality factor of 90.

In general, this algorithm works with exceptional outcomes. However, the same improvement needed - a better approach to find appropriate set of parameters for different images, since we need an ideal distribution to compare with.

4. Conclusion

Over the all experiments and analysis above, we can reach a conclusion that this model and algorithms proposed by the paper can be perfectly applied to image forensics area, namely JPEG compression detection and quality factor estimation.

Applications discussed in the paper can report image accurately and reliably, and the model and algorithm are straightforward enough for researchers to develop. The high reliability does not put requirements on the choice of classifiers, which makes it a very fast and convenient approach to forensics analysis.

As discussed in the report before, there are still mainly 2 aspects needs improvement. The first, the statistics model is not generalized enough so that each time researchers investigate an image further, they will have to configure a best set of parameters to this function. Moreover, the paper does not provide any suggestion on how to configure it easily and efficiently. Another limitation is that most discussions are proposed under the assumption that the JPEG images are compressed by standard quantization tables. If an image is compressed by certain customized table, this algorithm will not work or work with a probability of error output.

Addendum

To run my program provided, please use Matlab to execute files application1.m, application2.m and application3.m in order. They are implementations for JPEG single

compression detection, quality factor estimation and JPEG double compression detection respectively.

There are still some Matlab functions not shown in the report, and they will be included in the source code zip file. Since those functions are borrowed from assignment 1 of this course, they are omitted in this report, including JPEG_entropy_decode, JPEG_entropy_encode, myOwnJPGDecoder, myOwnJPGEncoder, Vector2ZigzagMtx, and ZigZagMtx2Vector. There are also JPEG.jpg and JPEG_DCTQ_ZZ.txt files, which are part of customized encoder and decoder, please put them under the same directory to ensure the whole program works.

Quantization tables under different quality factors are acquired through compressing using Matlab with different quality factors and extracted from JPEGsnoop. My function only preserves quantization tables under quality factors of 50, 60, 70, 80, 90, 95, 99 and 100. The quantization tables are implemented as a function, and the code is attached as following.

```
function matrix = getQuanMatrix(factor)

    if factor == 50
        matrix = [ 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 ];
    elseif factor == 60
        matrix = [ 13   9   8  13  19  32  41  49;
                   10  10  11  15  21  46  48  44;
                   11  10  13  19  32  46  55  45;
                   11  14  18  23  41  70  64  50;
                   14  18  30  45  54  87  82  62;
                   19  28  44  51  65  83  90  74;
                   39  51  62  70  82  97  96  81;
                   58  74  76  78  90  80  82  79 ];
    elseif factor == 70
        matrix = [ 10   7   6  10  14  24  31  37;
                    7   7   8  11  16  35  36  33;
                    8   8  10  14  24  34  41  34;
                    8  10  13  17  31  52  48  37;
                   11  13  22  34  41  65  62  46;
                   14  21  33  38  49  62  68  55;
                   29  38  47  52  62  73  72  61;
                   43  55  57  59  67  60  62  59 ];
    elseif factor == 80
```

```
        matrix = [  6   4   4   6  10  16  20  24;
                    5   5   6   8  10  23  24  22;
                    6   5   6  10  16  23  28  22;
                    6   7   9  12  20  35  32  25;
                    7   9  15  22  27  44  41  31;
                   10  14  22  26  32  42  45  37;
                   20  26  31  35  41  48  48  40;
                   29  37  38  39  45  40  41  40 ];
    elseif factor == 90
        matrix = [  3   2   2   3   5   8  10  12;
                    2   2   3   4   5  12  12  11;
                    3   3   3   5   8  11  14  11;
                    3   3   4   6  10  17  16  12;
                    4   4   7  11  14  22  21  15;
                    5   7  11  13  16  21  23  18;
                   10  13  16  17  21  24  24  20;
                   14  18  19  20  22  20  21  20 ];
    elseif factor == 95
        matrix = [ 2   1   1   2   2   4   5   6;
                   1   1   1   2   3   6   6   6;
                   1   1   2   2   4   6   7   6;
                   1   2   2   3   5   9   8   6;
                   2   2   4   6   7  11  10   8;
                   2   4   6   6   8  10  11   9;
                   5   6   8   9  10  12  12  10;
                   7   9  10  10  11  10  10  10 ];
    elseif factor == 99
        matrix = [ 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   2   2   1;
                   1   1   1   1   1   2   2   2;
                   1   1   1   1   2   2   2   2;
                   1   1   2   2   2   2   2   2;
                   1   2   2   2   2   2   2   2 ];
    else
        matrix = ones(8, 8);
    end

end
```