POLITECNICO DI MILANO
Corso di Laurea MAGISTRALE in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Analysis of Different Footprints for JPEG Compression Detection

Supervisor: Prof.Paolo Bestagini
Associate Supervisor: Dr.Luca Bondi

Tesi di Laurea di:
Chen Ke, matricola 833240

Anno Accademico 2016-2017

Abstract

Due to the wide availability of image editing software, it becomes more easy for people to tamper with image content. This makes authentication of digital images a problem of paramount importance for forensic researchers. Therefore, a lot of methods have been proposed to detect tampering process, and many of these algorithms are based on the analysis of traces left by JPEG compression. As an important part of digital image forensics, JPEG compression detection can be used to make a first judgment of the authenticity of images.

In the light of this consideration, this thesis focuses on the analysis of JPEG compression detection, i.e., the ability of blindly understanding whether an image has been JPEG compressed or not from pixel analysis. More specifically, in this work we focus on the analysis of two specific state-of-the-art algorithms for JPEG compression detection: one based on pixel domain analysis; another one based on Fourier transform domain analysis. After proposing a modification that enhance the performance of one detector, we compare all the implemented methods showing pros and cons of these algorithms by means of a simulative campaign. Finally, in this work we also propose a graphical user interface to enable non-expert forensic investigators to use the implemented JPEG-based tools for image analysis.

Acknowledgment

I would first like to thank my thesis supervisor Prof. Paolo Bestagini of the Department of Electronics, Informatics and Bioengineering at Politecnico di Milano, and assistant supervisor Dr. Luca Bondi of the Image and Sound Processing Group (ISPL) at Politecnico di Milano. The doors to Prof. Bestagini and Dr. Bondi office were always open whenever I ran into a trouble spot or had a question about my research or writing. They consistently allowed this paper to be my own work, but steered me in the right the direction whenever they thought I needed it.

Finally, I must express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Contents

Abstract **1**

Acknowledgment **3**

1 Introduction **13**

1.1 Motivation Behind Image Forensics 13

1.2 Problem Statement and Contribution 15

1.3 Structure of the Thesis 17

2 Background **19**

2.1 JPEG Compression 19

2.1.1 JPEG Introduction 19

2.1.2 Luminance/Chrominance Space Transformation 20

2.1.3 Discrete Cosine Transform 21

2.1.4 Quantization 23

2.1.5 Coding 24

2.2 JPEG Forensics 28

2.2.1 JPEG Compression Detection 29

2.2.2 JPEG Double Compression Detection 30

2.2.3 JPEG Multiple Compression Detection 36

2.2.4 Tampering Localization 37

3 JPEG Compression Detection 41

3.1 Pixel Domain Compression Detection 41

3.2 Pixel Domain Cropped Image Detection 46

3.3 Pixel Domain Improved Cropped Image Detection 48

3.4 Transform Domain Compression Detection 48

4 Implementation and GUI 53

4.1 Experimental Software 53

4.2 Compressed Image Detector GUI Implementation 54

5 Experimental Results 57

5.1 Dataset Generation 57

5.2 Evaluation Metrics 59

5.3 JPEG Compression Detection With A Priori Grid Information 60

5.3.1 Pixel Domain Compression Detection 60

5.3.2 Transform Domain Compression Detection 61

5.4 JPEG Grid Position Estimation 64

5.5 JPEG compression detection in the wild 65

6 Conclusion and Future Works **67**

Bibliography **69**

List of Figures

1.1 Original and forged images of South Korea former leader Kim Jong-il's funeral. 14

2.1 Block Diagram of JPEG coding [1]. 20

2.2 Image 8×8 partition [2]. 22

2.3 DCT Spectrum [2]. 22

2.4 Zig-zag scanning process [2]. 26

2.5 A-DJPG and NA-DJPG compression [3]. 31

2.6 probability distribution of first digits of block-DCT coefficients [4]. 32

2.7 Effect of double-compression on histograms of absolute values of DCT coefficients [5]. 33

2.8 Contour Comparison [6]. 35

2.9 JPEG multiple compression steps [7]. 36

2.10 Example of blockwise tamper detection [8]. 38

2.11 Application to realistic forgeries [9]. 39

3.1 For each block the number $Z'[i,j] = |A - B - C + D|$ and $Z''[i,j] = |E - F - G + H|$ [10]. 42

3.2 Histogram of K with different quality factors 10 and 30. 43

3.3 Histogram of K with different quality factors 50 and 70. 44

3.4 Histogram of K with different quality factors 90 and 100. . . . 45

3.5 Histograms of region I (within block) and II (across blocks) and those difference. Top figure is related to QF 90, bottom figure is related to QF 10. 46

3.6 Vertical and horizontal masks of Sobel operator [11]. 50

3.7 Sobel operator example [11]. 50

3.8 Basic concept of harmonic analysis [12]. 51

3.9 Harmonic analysis process for estimating blockiness [12]. . . . 52

4.1 Compression detection GUI 56

5.1 UCID 58

5.2 Image datasets with different QF by using IrfanView 59

5.3 ROC of compress image datasets and compressed & cropped image datasets with pixel domain compression detection. . . . 61

5.4 Same image with different QF implemented by transform domain compression detection. 62

5.5 ROC curve for all proposed methods. 63

5.6 Histogram of result of cropped image detection algorithm. . . 64

5.7 Accuracy of old D21 and improved D21 (*alternative 1* is old D21, and *alternative 2* is improved D21). 66

List of Tables

2.1 Intermediate format of DC coefficients 25

2.2 Intermediate format of DC coefficients 28

4.1 Implementation software 54

5.1 Unranked retrieval 60

5.2 Method combinations of transform domain compression detection 62

Chapter 1

Introduction

1.1 Motivation Behind Image Forensics

With the progress of time, technology is changing rapidly and all linked costs are constantly decreasing. Therefore, digital cameras, printing and scanning equipment have started to spread. For this reason, digital images have become part of our everyday life. In the meanwhile, the possibilities opened by digital image processing and editing software suites (e.g., Photoshop, Lightroom, etc.) enable to perform very complex tampering operations. Therefore, it is becoming more difficult to distinguish tampered image by human eyes inspection. In particular, every picture we see instill so many doubts about its authenticity. Indeed, nowadays we often ask ourselves whether an image has been tampered with before, or if it is reliable and authentic. Even knowing that the image has been forged, which operations have been applied to the tampered image often remains a doubt [13].

In the present society, any users of personal computer and portable platform (mobile and tablet) can easily tamper the image due to the low-cost digital image intake devices and the popularity of powerful image editing software. Simultaneously, the powerful communication ability of Internet supplies abundant software and image resource for digital image tampering.

With a lot of images being used in the newspaper, image fraud and tampering still exist despite repeated bans. Just to mention an example, In 2011, a group of photos of Kim Jong-il's funeral reports the funeral scene of the former leader Kim Jong-il of South Korea. From the top image in Figure 1.1 reported by Japanese media, it is obvious that there is a group of people surrounding a camera on the bottom-left corner, but those people and camera have been removed from the photo in the bottom picture in Figure 1.1, which was taken by North Korea media.

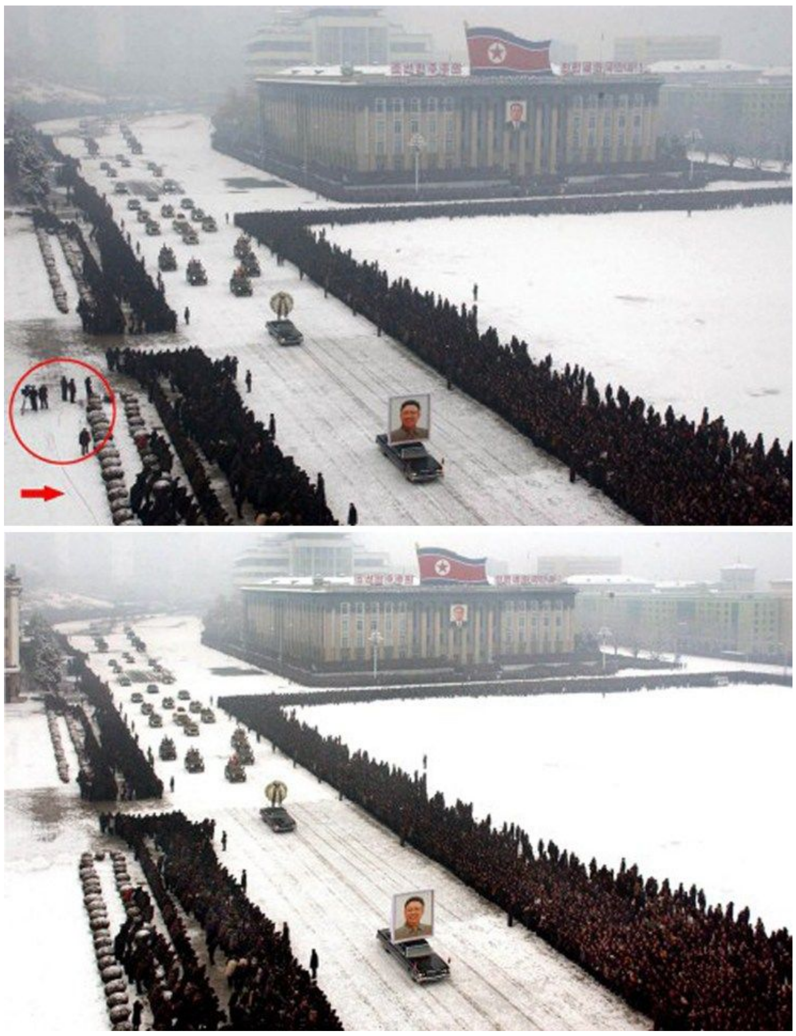


Figure 1.1: Original and forged images of South Korea former leader Kim Jong-il's funeral.

Let us imagine that such a fraud image appears in court of law as evidence. In this situation, it may lead to misjudgments and incorrect court results. Therefore, it is essential to find a way to analyze digital images with scientific way from a forensic perspective. To solve this issue, the forensic community has developed several algorithms to fight image tampering [3]. These algorithms work under the assumption that, every time an image is modified through editing operations, some characteristic footprints are left on the picture itself. Therefore, it is possible to analyze an image to assess the presence of these footprints, thus detecting tampering.

As one of the most commonly used non-invertible operations leaving footprints on images is JPEG compression [10, 14], in this thesis we focus on forensic methodologies able to detect the presence of JPEG compression on an image.

1.2 Problem Statement and Contribution

Digital image tampering forensics is an important branch of the field of digital forensics. It involves a variety of forensic techniques that are used for digital image authenticity analysis, detection and identification. JPEG-based forensics is an important part of digital image forensics, as JPEG compression is widely accepted as commonly used compression scheme. Therefore, traces left by JPEG compression can be often exploited to reconstruct the past history of an image and detect the use of possible tampering operations.

It is undeniable that continuously developing forensic technologies and algorithms have brought tremendous influence and enhancement on digital image forensics, especially in JPEG compression detection [3]. In particular, we can roughly split JPEG-based forensic methods into two categories: i) algorithms to detect the use of JPEG compression on an image [14, 15]; ii) algorithms to estimate JPEG parameters (e.g., quantization matrix) from an image which is known to be JPEG compressed [10].

In this thesis, we focus on the first class of algorithms. In particular, we analyze methods for JPEG compression detection. This means, given an image in the pixel domain, detect whether this image has been JPEG compressed, not using any information from the header.

Being able to solve this problem is useful in many application scenarios. As an example, in the context of professional photography, it is possible to detect whether an image is a real raw picture coming from a camera, or it is a lower quality compressed one. Moreover, it is possible to detect whether JPEG compression has been applied only to a portion of an image. This happens when a portion of JPEG image is copied and pasted onto a raw (e.g., bitmap, PNG, TIFF, etc.) image. Therefore, the ability of detecting JPEG compression paves the way to the development of image tampering detection methods.

Despite a set of methodologies to detect JPEG compression has been proposed in the literature [10, 14], some questions still remain unanswered. As a matter of fact, a common implementation of many algorithms is not readily available. Moreover, different algorithms have been tested on different datasets in different conditions. This makes difficult to choose which method to select in case of forensic analysis. For instance, after the implementation of one algorithm, it is necessary to analyze the result of the algorithm and make hypothesis that whether this algorithm can handle different types of JPEG image such as aligned compressed JPEG image and non-aligned compressed JPEG image.

Furthermore, we have mentioned in previous section that any user of personal computer and portable platform (mobile and tablet) can easily tamper with images due to the low-cost digital image intake devices and the popularity of powerful image editing software. Thus, it would be preferable that non-expert forensic investigators could perform their analysis with some easy-to-use tools. However, great part of the forensic algorithms proposed in the literature remain confined for expert people in the field.

From this premises, in this thesis we

- Implement of a set of state-of-the-art algorithms [10, 14, 6] for JPEG compression detection.
- Test and analyze these algorithm for JPEG compression detection, highlighting their pros and cons on a common dataset.
- Propose a modification that enhance the performance of [10, 14].

- Propose a methodology for merging results from different detectors.
- Build a GUI with those algorithms to enable non-expert forensic investigators to use these tools for JPEG image analysis.

1.3 Structure of the Thesis

The rest of the thesis is structured as it follows.

In Chapter 2, we focus our attention on the background and mechanisms of JPEG compression and JPEG forensics. In terms of JPEG compression, we will introduce the specific processes of JPEG compression such as luminance and chrominance transformation, DCT, Quantization, Zig-zag scanning and entropy coding. As far as JPEG forensics is concerned, we are going to give brief formulation and corresponding state-of-the-art algorithms and methods of JPEG compression detection, JPEG double compression detection, JPEG multiple compression detection and tampering localization.

In Chapter 3, a subset of chosen state-of-the-art algorithms and methods for JPEG compression detection will be discussed in detail. Three algorithms of JPEG compression detection will be proposed, two of them are committed to JPEG grid alignment, the last one is going to estimating the grid location. In this chapter we also provide details about our proposed modification to [10, 14].

In Chapter 4, we explain how to implement the above algorithms proposed in Chapter 3. In order to achieve the last objective, we will give the explanation and functional requirements of JPEG compression detection GUI.

In Chapter 5, we gather all simulations and experimental results in order to analyze the pros and cons of the implemented algorithms, together with our proposal.

In Chapter 6, we will present an overall view of conclusion about the achieved objectives and give possible future works.

Chapter 2

Background

In this chapter, we introduce the background and mechanism of JPEG compression and JPEG forensics. In terms of JPEG compression, we will discuss what are the mechanisms like luminance and chrominance transformation, DCT, Quantization, Zig-zag scanning and entropy coding and explain one example of JPEG compression as well. Concerning JPEG forensics, we are going to discuss JPEG compression detection, JPEG double compression detection, JPEG multiple compression detection and tampering localization. For each of these topics we briefly introduce how state-of-the-art methods work.

2.1 JPEG Compression

In this section, we introduce in detail how JPEG compression works and what processes have been used in each step. In the last, we give an example that how to compress an image to JPEG.

2.1.1 JPEG Introduction

JPEG stands for Joint Photographic Experts Group, and it is one of two sub-groups of ISO/IEC Joint Technical Committee, being responsible for the development of the well-known digital image compression standard also called JPEG. JPEG coding is the most widespread standard for representation of still images. JPEG is widely used today and a very flexible digital photograph compression standard. It could be lossy as well as lossless. The technique we are going to discuss hereinafter is the lossy one. In particular, JPEG compression process follows the steps depicted in Figure 2.1, and detailed in the following:

1. First an image is converted into YCbCr colorspace and split into blocks.
2. Each block is transformed through Discrete Cosine Transform (DCT).
3. DCT coefficients are quantized according to some quantization rules to be defined.
4. Encode and pack into bit-stream.

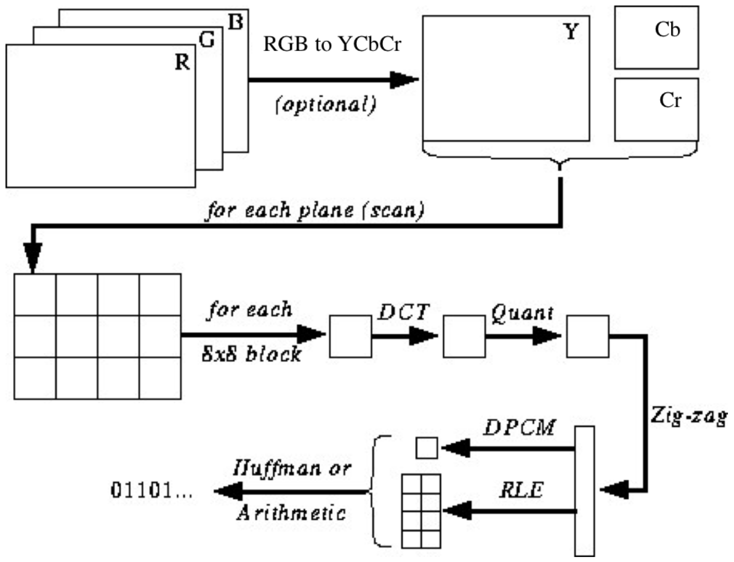


Figure 2.1: Block Diagram of JPEG coding [1].

2.1.2 Luminance/Chrominance Space Transformation

JPEG uses YCbCr luminance and chrominance space, but images are often available in RGB color space. Therefore, an image is first transformed from

RGB into luminance/chrominance space(YCbCr) before it is compressed. In YCbCr, Y stands for luminance, I and Q stand for chrominance. The two chrominance channels(Cb and Cr) are typically subsampled by a factor of two relative to the luminance channel(Y). And the transformation relationship between RGB and YCbCr are:

$$\begin{aligned} \mathrm{Y} &= 0.299R + 0587G + 0.114B \\ \mathrm{Cb} &= -0.1687R - 0.3313G - 0.5B + 128 \\ \mathrm{Cr} &= 0.5R - 0.418G - 0.0813B + 128 \end{aligned}$$

2.1.3 Discrete Cosine Transform

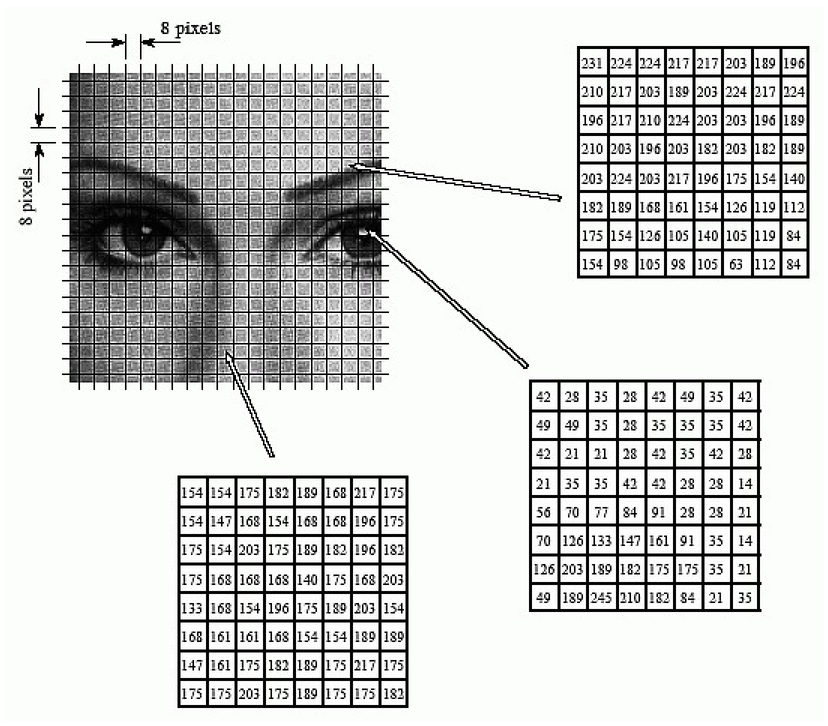
DCT (Discrete Cosine Transform), is a linear transformation commonly used in transform coding methods. Each channel is then partitioned into 8×8 pixel blocks. These values are converted from unsigned to signed integers (e.g. from [0, 255] to [-128, 127]) as shown in Figure 2.2. JPEG first partitioned an image into 8×8 non overlapping pixel blocks, then execute DCT operation for 8×8 pixel blocks one by one. Recall that the encoding of a JPEG image need DCT and decoding of a JPEG image need inverse DCT.

The DCT formula is

$$F[i,j] = C[i,j] \sum_{x=0}^{7} \sum_{y=0}^{7} f[x,y] \cos\left[\frac{(2x+1)i\pi}{16}\right] \cos\left[\frac{(2y+1)j\pi}{16}\right], \quad (2.1)$$

where $f[x,y]$ is a pixel in a 8×8 image block, i and j are DCT frequencies, and $C[i,j]$ is a normalization term.

The DCT basic spectrum consist of an 8×8 array as shown in Figure 2.3, with each element in the array being an amplitude of one of the 64 basis functions. Six of these functions are shown here, referenced where the corresponding amplitude resides



231	224	224	217	217	203	189	196
210	217	203	189	203	224	217	224
196	217	210	224	203	203	196	189
210	203	196	203	182	203	182	189
203	224	203	217	196	175	154	140
182	189	168	161	154	126	119	112
175	154	126	105	140	105	119	84
154	98	105	98	105	63	112	84

42	28	35	28	42	49	35	42
49	49	35	28	35	35	35	42
42	21	21	28	42	35	42	28
21	35	35	42	42	28	28	14
56	70	77	84	91	28	28	21
70	126	133	147	161	91	35	14
126	203	189	182	175	175	35	21
49	189	245	210	182	84	21	35

154	154	175	182	189	168	217	175
154	147	168	154	168	168	196	175
175	154	203	175	189	182	196	182
175	168	168	168	140	175	168	203
133	168	154	196	175	189	203	154
168	161	161	168	154	154	189	189
147	161	175	182	189	175	217	175
175	175	203	175	189	175	175	182

Figure 2.2: Image 8×8 partition [2].

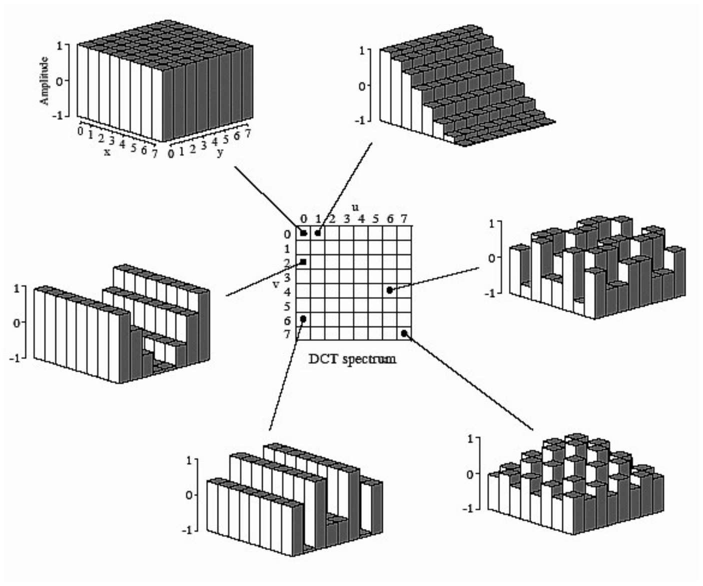


Figure 2.3: DCT Spectrum [2].

2.1.4 Quantization

Quantization is the process of converting a continuous range of infinitely many values into a finite discrete set of all possible values. The quantization process generally approximates the input set into preferably smaller set. The advantage of quantization is that it decreases the number of bits required for storing and transmitting the data.

Quantization is a lossy and non-reversible process since it involves rounding off and discarding negligible entities. The inverse quantization does not generate the same object which was fed to a quantizer. Whatever is lost is often modeled by an additive quantization noise.

Quantization matrices are used for defining the quantization process. Assuming $Q[i,j]$ is the quantizer matrix, every time a matrix of DCT coefficient, we call it $F[i,j]$, is encountered, it's divided by quantizer matrix $Q[i,j]$ and rounded to obtain quantized matrix $F_q[i,j]$. Formally, quantization equation can be given as $F_q[i,j] = round(F[i,j]/Q[i,j])$. The inverse quantization equation is $F'[i,j] = F_q[i,j] * Q[i,j]$.

An example, if we consider the matrix of DCT coefficients:

$$F[i,j] = \begin{bmatrix} -415 & -33 & -58 & 35 & 58 & -51 & -15 & -12 \\ 5 & -34 & 49 & 18 & 27 & 1 & -5 & 3 \\ -46 & 14 & 80 & -35 & -50 & 19 & 7 & -18 \\ -53 & 21 & 34 & -20 & 2 & 34 & 36 & 12 \\ 9 & -2 & 9 & -5 & -32 & -15 & 45 & 37 \\ -8 & 15 & -16 & 7 & -8 & 11 & 4 & 7 \\ 19 & -28 & -2 & -26 & -2 & 7 & -44 & -21 \\ 18 & 25 & -12 & -44 & 35 & 48 & -37 & -3 \end{bmatrix}$$

and the Quantizer Matrix:

$$Q[i,j] = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

we get the following quantized matrix:

$$F_q[i,j] = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -3 & 4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which will be further processed by the next JPEG standard steps.

Notice that, in order to control the trade-off between image quality and compression rate, quantization matrices can be scaled by a quality factor (QF). In doing so, DCT coefficients can be strongly (or lightly) quantized, thus generating a worse (or better) quality image. Higher QFs denote higher quality. Conversely, lower QFs denote lower quality.

2.1.5 Coding

In order to further compress quantized coefficients, JPEG standards makes use of different lossless coding techniques on direct current (DC) and alternating components (AC) coefficients. Specifically, Differential Pulse Code Modulation (DPCM) and Run Length Encoding (RLE) are used, respectively.

After DCT and quantization, the DC coefficients of 8×8 image blocks have two characteristics: the values of the coefficients are relatively large and the DC coefficient values of the adjacent 8×8 image blocks do not change too much. Therefore, DC coefficients are coded using DPCM (Difference Pulse Code Modulation) based on those two features, that's encoding of difference between each DC value of the same image component and the previous DC value, rather than storing DC actual values.

To save more space in JPEG, it groups 16 sets based on the values of data instead of saving the exact value of data and called it VLC (Variable Length Coding). VLC is the process of mapping the input symbols into codes of variable lengths. This enables us to compress the symbols without any error.

Intermediate format of DC coefficients. We mentioned in last paragraph that there is a difference between two DC values, we can check this value in Table 2.1. For example, if the difference between two DC values is 3, it is obvious that integer 3 located in set 2, thus its format can be written as (2)(3) , and this format is called intermediate format of DC coefficients.

Value	Set	Actual saved value
0	0	-
-1, 1	1	0, 1
-3, -2, 2, 3	2	00, 01, 10, 11
-7, -6, -5, -4, 4, 5, 6, 7	3	000, 001....111
-15, ..., -8, 8, ..., 15	4	0000, ..., 0111, 1000, ..., 1111
-31, ..., -16, 16, ..., 31	5	00000, ... ,11111
-63,....,-32, 32,..., 63	6	...
...	...	...
-32767,........	15	...

Table 2.1: Intermediate format of DC coefficients

Concerning AC coefficients, JPEG exploits the fact that many of them are quantized to zero and makes use of RLE (Run Length Encoding). And Zig-zag scanning is going to be implemented.

We know that the most AC values in a quantized matrix are zero [16] . Zig-zag is an approach that can be used to gather more numbers of zeros together. Zig-zag scanning groups low frequency coefficients before the high frequency coefficients. If the process stores 64 numbers column by column, thus there is no relationship between last node of current column and first node of next column. So, this 'Z' processes serialize the matrix into a string from left-top corner and this process as shown in Figure 2.4

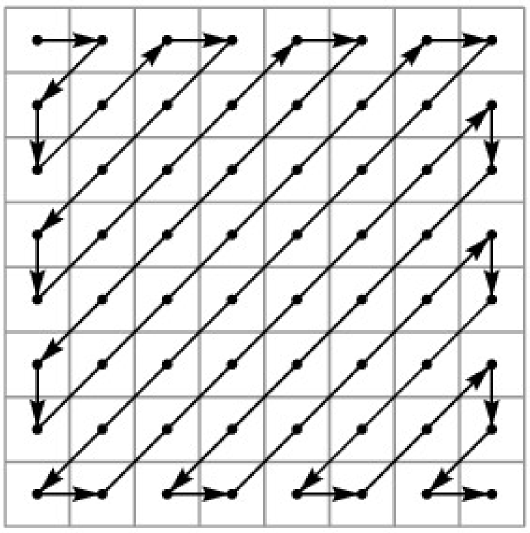


Figure 2.4: Zig-zag scanning process [2].

RLE (Run Length Encoding) is a lossless compression technique of encoding data where consecutively occurring entities are represented only once with a symbol along with frequency. The original sequence is transformed into a smaller run with data values and their count, thus enabling compression. RLE is an approach that is suitable for all kinds of information, text or binary.

An example of RLE is the following. Let us consider the input sequence of symbols: AAABBBBBBBCCCCCDDDDDEEFFFGGGGG

Using Run Length Coding, they become: 3A7B5C5D2E3F5G

Another example of RLE that only compresses consecutive zeros is as follows. Let us consider the input symbols: 0000010000000000001000101100000000000

Counting the number of zeros separated by 1's: 5 11 3 1 0 12 and in 4-bit code representation, the run length encoding is: 0101 1011 0011 0001 0000 1100.

Intermediate format of AC coefficients. Let us input a string: 57,45,0,0,0,0,23,0,-30,-8,0,0,1,000...... After the processing of RLE, we get the data with this format: (0,57) ; (0,45) ; (4,23) ; (1,-30) ; (0,-8) ; (2,1) ; (0,0). Then process the right-side data of the data in pair and then implement it with VLC table. For instance, we found that value 57 located in set 6, therefore, the format of this data can be written as (0,6), 57 and named it as intermediate format of AC coefficients.

To further compress the data, it's necessary to entropy the DPCM and RLE. JPEG standard specifies two entropy coding methods which are Huffman coding and arithmetic coding. Huffman coding defines that a binary code having a shorter character length is assigned to a character having a large probability of occurrence, and a binary code having a longer character length is assigned to a character having a small probability of occurrence, so that average encoding length of the character is shortest.

DC coefficients and AC coefficients are using different Huffman table when Huffman coding is being processed, and using different Huffman table based on luminance and chrominance components. Therefore, JPEG defines four Huffman coding tables to complete the work of entropy coding.

To better clarify this concept, let us now see an example. Let's assume that we already got matrix of quantized 8×8 image blocks:

$$M = \begin{bmatrix} 15 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

It's obvious that DC coefficient is 15, assume that previous quantized DC coefficient value of 8×8 image block is 12, then the difference between current DC coefficient and previous DC coefficient is 3, it's easy to find the intermediate format of DC coefficient is (2)(3) based on VLC coding table, it means the code length of number 3 is 2. After Zig-zag scanning, we found that the first non-zero AC coefficient value is -2, and zero is 1, then it can be presented to (1, -2) and belonging to set 2 in VLC table. Thus, the intermediate format of AC coefficient is (1, 2) -2, and repeat those steps for left values in matrix and obtained the intermediate format of 8*8 block entropy code are following: DC: (2)(3); AC (1, 2) (-2), (0, 1) (-1), (0, 1) (-1), (0, 1) (-1), (2, 1) (-1). The Table 2.2 shows the example of intermediate format.

Intermediate format	Luminance Huffman table	Code VLC
DC (2)(3)	011	11
AC (1, 2) (-2)	11011	01
AC (0, 1) (-1)	00	0
AC (2, 1) (1)	11100	0
AC (0, 0)	1010	

Table 2.2: Intermediate format of DC coefficients

Thus, the data flow of compressed 8*8 block luminance is 01111, 1101101, 000, 111000, 1010. Totally 31 bits and compression rate is 64*8/31 = 16.5.

2.2 JPEG Forensics

In this section, we introduce some background on JPEG forensics. First, we review some state-of-the-art JPEG compression detection methods. Then, we focus on JPEG double compression detection and JPEG multiple compression detection explaining what are the majority methods and algorithms used for this purpose.

2.2.1 JPEG Compression Detection

Let us consider the scenario in which a digital image is available in the pixel domain as BMP, without any knowledge about prior processing. JPEG compression detection is the problem of detecting whether that image is actually uncompressed, or it had been previously compressed and which were the compression parameters being used. This is useful to avoid fake-bitrate frauds, in which someone sells an image as high-quality uncompressed one, even if the image was previously compressed.

The idea of forensic methods coping with this problem is that block-based image coding like JPEG leaves characteristic artifacts on images. This artifact, as shall be explained in the rest of the section, can be exposed in different domains to reverse engineer the compression history of an image. To this purpose, we are going to discuss two different compression detection in pixel domain and transform domain.

In the pixel domain, there are several approaches in the literature that introduce different methods for estimating blockiness, i.e., a JPEG characteristic artifact. The authors of [10, 14] explain a method that estimate in conjunction with a very simple efficient way whether an image has been previously JPEG compressed, and the used quality factor (QF), which ranges from 1 to 100, 1 means highly compressed. The rationale behind the algorithm is that if an image has not been compressed, pixel differences across 8×8 block boundaries should be similar to those within blocks. It is then possible to compute a measure of this pixel differences, and compare it to a threshold. If it is higher than the threshold, the image has been compressed.

The authors of [17] explain a method that model a blocky image as sum of non-blocky image and a pure blocky signal. The method detects JPEG compression based on the estimation of the power of this blocky signal. There is a problem in evaluating the power of blocky signal without accessing the original image by using blind way of estimation of blockiness. So, the absolute value of row gradient and column gradient of image need to be computed separately.

There is a similar algorithm mentioned by authors of [18]. Compared with last case, horizontal gradient and vertical gradient of image are computed instead of row and column, using DFT to estimate their periodicity due to gradient peaks at block boundaries in frequency domain. After that, the gradient position can estimate block position, then they computed blockiness distortion evaluation, employing a weighting scheme based on the local gradient energy, thus, the block size and block location are identified.

The authors of [19] provide another method to estimate block size by using the periodicity of direction of gradient. In order to enhance the peaks, the authors subtract a median filtered version to the gradient, and set threshold based on the sum of the gradient which aims at avoiding spurious peaks caused by edges from objects in the image.

2.2.2 JPEG Double Compression Detection

The aforementioned solutions aim at detecting whether an image is uncompressed, or has been JPEG compressed once. However, images are often JPEG compressed once at photo inception directly by the acquisition device. When they are edited with whatever software suite (e.g., PhotoShop, GIMP, etc.) and saved, they often undergo a second compression. Being able to estimate whether an image has been JPEG compressed once or twice proves then paramount as forgery indicator. Specifically, we refer to this problem as double JPEG (DJPEG) compression detection.

DJPEG however depends on some parameters. As an example, first and second compression may use different quantization matrices Q^1_{ij} and Q^2_{ij}. Moreover, the 8×8 JPEG grid may be aligned or not between the two compression steps. Usually, DCT coefficient $F[i, j]$ are considered to be compressed twice when $Q^1_{ij} \neq Q^1_{ij}$.

Figure 2.5 reports an example of DJPEG compression. It is obvious that I0 is an uncompressed image, the second JPEG compression I_2 in case one that adopts a DCT grid aligned with one used in I_1, but it does not match in second case.

The main algorithms for detecting double JPEG compression are based on JPEG artifacts, there are two detection approaches according to whether the second JPEG compression adopts a DCT grid aligned with the one used

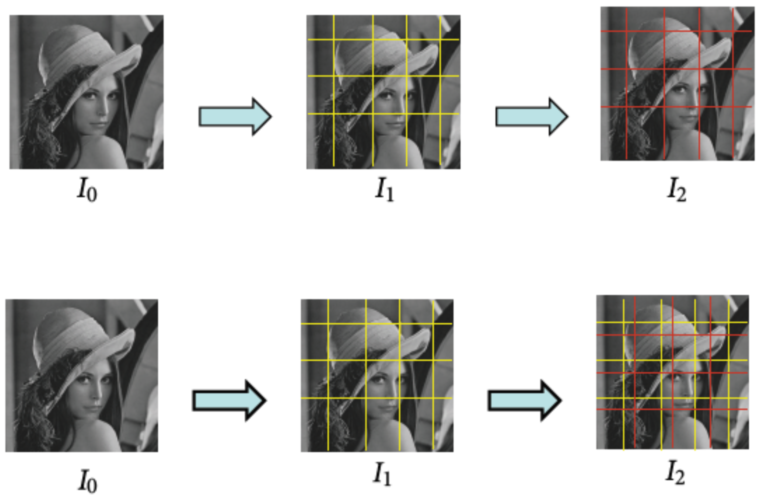


Figure 2.5: A-DJPG and NA-DJPG compression [3].

by first JPEG compression.

Detection of A-DJPG Compression. The authors of [4, 20] proposed the detection method that is based on the observation that in natural images the distribution of the first digit of DCT coefficients in single JPEG compressed images follows the generalized Benford's law [21]. The method uses the probabilities of the first digits of quantized DCT coefficients from individual AC modes to detect double compression JPEG images. The Figure 2.6 shows that the probability distribution of first digits of block-DCT coefficients follow the standard Benford law very well. The quality of the fitting can be present as:

$$X = \sum_{i-1}^{9} \frac{(\hat{P}_i - P_i)^2}{P_i},$$

where $\hat{P}_i$ is the actual first digit distribution and P_i is the probability predicted by Benford's law.

The experimental result shows that each compression step changes the statistics of the first digit distribution and with the number of the com-

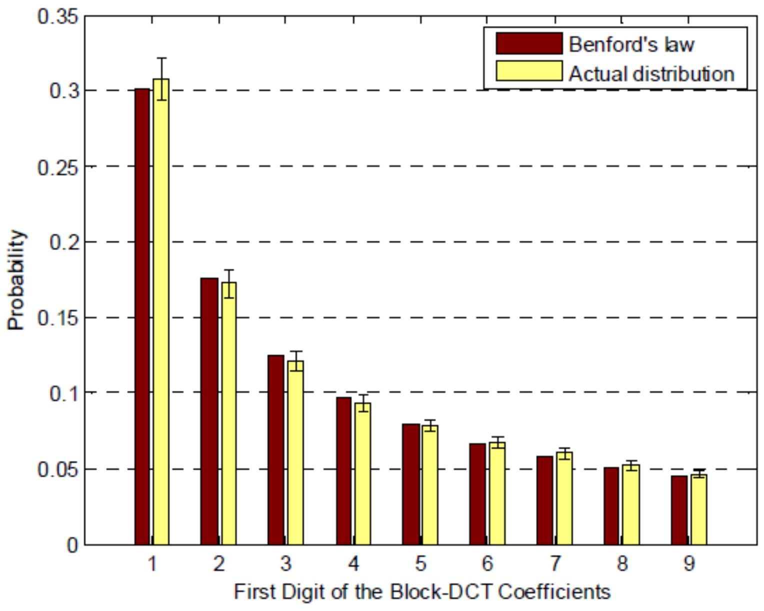


Figure 2.6: probability distribution of first digits of block-DCT coefficients [4].

pression, the accuracy the Benford's law has been decreased. Therefore, by thresholding X it is possible to infer information about single or double compression. The performance of this method does not seem adequate and the result can be improved by other methods.

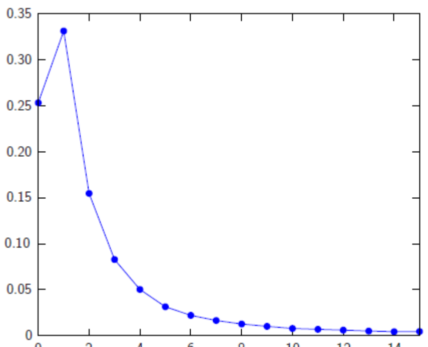
In [22], author introduce the observation that re-quantization includes periodic artifacts and discontinuities in the image histogram, a set of features is derived from the pixel histogram to train an SVM using for A-DJPEG compression detection. However, this method has been tested only for secondary quality factors 75 or 80.

A promising idea is introduced by authors in [5], they proposed the methods for detecting double-aligned JPEG compression and for estimation of primary quantization matrix, which is lost during recompression. The proposed methods are necessary for construction of accurate targeted and blind steganalysis methods for JPEG images and those methods based on SVM classifiers with features vectors by histograms of DCT coefficients.

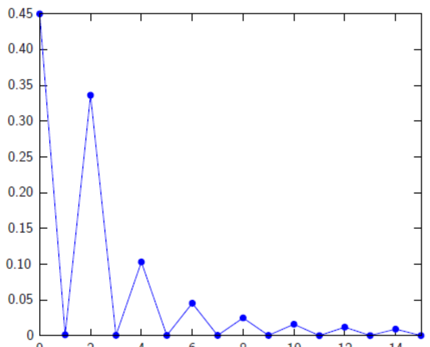
We have mentioned before that the quantization matrix Q^1_{ij} used during the first compression is called the primary quantization matrix. The quantization matrix Q^2_{ij} used in subsequent JPEG compression is called the secondary quantization matrix. They specific DCT coefficient $F[i,j]$ was double compressed if and only if $Q^1_{ij} \neq Q^2_{ij}$, the double compressed DCT coefficient $F[i,j]$ is

$$D_{ij} = \left\lfloor \lfloor \frac{F[i,j]}{Q^1_{ij}} \rfloor \cdot \frac{Q^1_{ij}}{Q^2_{ij}} \right\rfloor$$

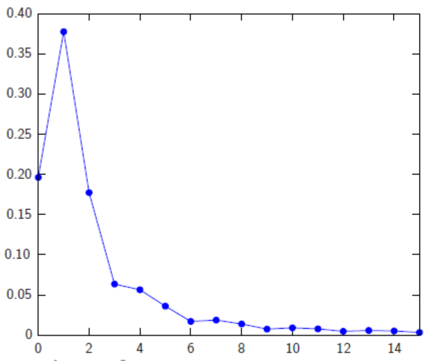
We can see that the values of double compressed DCT coefficient rely on the combination of quantization steps Q^1_{ij} and Q^2_{ij}. And Figure 2.7 shows the effect of double compression on histograms of absolute values of DCT coefficients. The secondary quantization coefficient is in all four cases the same $Q^2_{ij} = 4$, only the primary quantization coefficient Q^1_{ij} changes. These peaks in the histogram assume different configurations according to the relationship between the first quantization and secondary quantization.



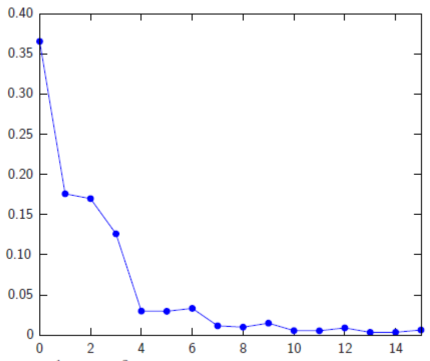
(a) $Q^1_{ij} = 4, Q^2_{ij} = 4$: histogram of a single-compressed DCT coefficient



(b) $Q^1_{ij} = 8, Q^2_{ij} = 4$: histogram with zeros at multiples $1, 3, 5, \ldots$



(c) $Q^1_{ij} = 3, Q^2_{ij} = 4$: histogram with double peaks at multiples $(1,2), (4,5), (7,8) \ldots$



(d) $Q^1_{ij} = 6, Q^2_{ij} = 4$: histogram with double peaks at multiples $(1,2), (4,5), (7,8), \ldots$

Figure 2.7: Effect of double-compression on histograms of absolute values of DCT coefficients [5].

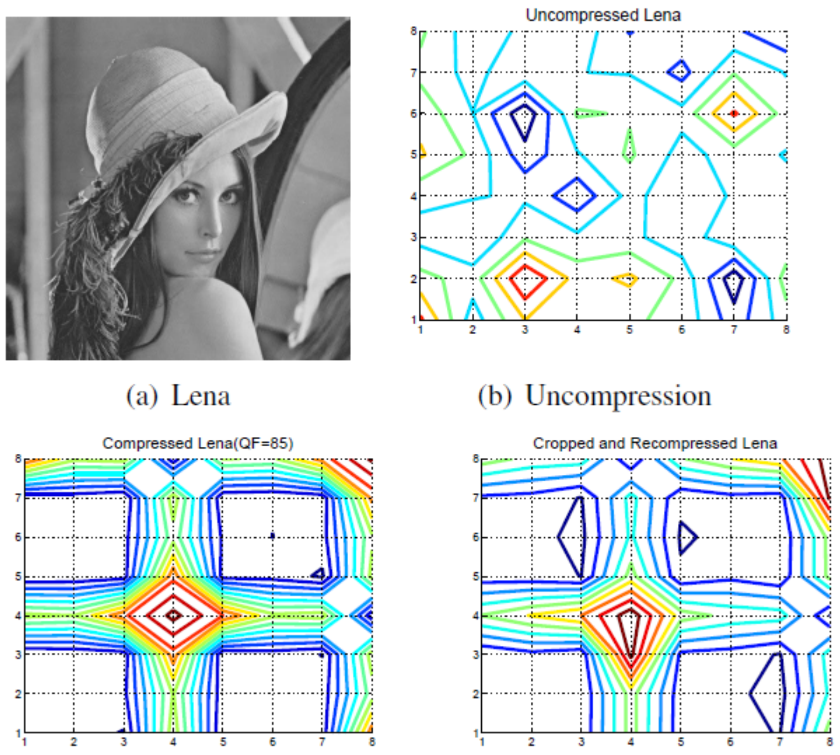
Then they implement the method as follow:

- Computing the histograms of absolute values of all analyzed DCT coefficients from the image under investigation I
- Using sets of quantization table to crop and compress the image.
- Re-compressing the cropped and compressed image by using Δ_2.
- Computing the histograms of absolute values of DCT coefficients from double compressed and cropped images. The estimator chooses the quantization table that the result histogram is as similar as possible to obtained from the image I.

Detection of NA-DJPG Compression. In order to analyze whether the reconstructed image has been compressed, the blocking artifacts analysis has been taken into account. These methods depend on the fact that, the original part of tampering image exhibits regular blocking artifacts, however the pasted one does not, because second compression was not aligned with the first one.

They start from an idea proposed in [10, 14] to detect blocking artifacts, and used in [6]. The authors in [6] proposed a method first analyses the process in JPEG compression like the methods mentioned in [10, 14]. They estimate a matrix $M[x, y]$, and derives the blocking artifacts characteristics matrix (BACM) to measure the symmetrical property of the blocking artifacts in a JPEG image. As Figure 2.8 shown, Figure 2.8d shows the contour of $M[x, y]$ in the cropped and recompressed image, the symmetry of the values of $M[x, y]$ descends comparing with Figure 2.8c, in other words, as asymmetric $M[x, y]$ will reveal the exist of Non-aligned JPEG compression. However, this method is available only when the tampered region is very large. (greater than 500×500 pixels)

There is another method [23] which improves usability and performance around 5% of the method we already have talked in [6]. The authors proposed the method which assumes the image signal is the result of the superposition of different components that are mixed together in the resulting image. The independent component analysis (ICA) algorithm is suitable for



(a) Lena (b) Uncompression

(c) $QF = 85$ (d) Cropped and recompressed

Figure 2.8: Contour Comparison [6].

this task which used to identify the different contribution and separate them into independent signal. Comparing with last method, this method can be implement with NA-DJPG image whose tampered regions are small.

The authors of [15] proposed a method which does not rely on SVM classifier but the threshold detector has been used. The proposed method evaluates a single feature based on the integer periodicity of DCT coefficient when the DCT has been computed according to the grid of previous compression. When NA-DJPG is detected, the parameters of the lattice give the primary quantization table.

2.2.3 JPEG Multiple Compression Detection

JPEG multiple compression detection, as the name implies, is the problem of detecting whether a JPEG image has been compressed more than twice. The authors of [7] proposed a method which is a statistical framework for the identification of multiple aligned compression in JPEG images and estimation of the applied quality factors. This method involves following steps and Figure 2.9 shows the example of this method.

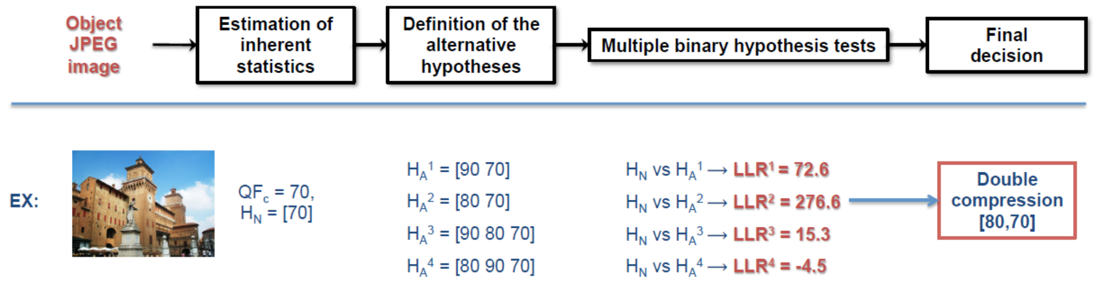

Object JPEG image
Estimation of inherent statistics
Definition of the alternative hypotheses
Multiple binary hypothesis tests
Final decision
EX:
QF_c = 70, H_N = [70]
H_A^1 = [90 70]
H_A^2 = [80 70]
H_A^3 = [90 80 70]
H_A^4 = [80 90 70]
H_N vs H_A^1 → LLR^1 = 72.6
H_N vs H_A^2 → LLR^2 = 276.6
H_N vs H_A^3 → LLR^3 = 15.3
H_N vs H_A^4 → LLR^4 = -4.5
Double compression [80,70]

Figure 2.9: JPEG multiple compression steps [7].

- ***Estimation of inherent statics***: estimate the unquantized DCT coefficients of the image, implement the noise on the DCT coefficient and modeled as Gaussian noise to extract its mean and variance from the given image.

- ***Definition of the alternative hypotheses***: create a collection of possible binary hypotheses and test the image on each of them, depending on the type of compression chains considered. The null hypothesis set as H_N , the collection of alternative hypotheses set as $\left\{H_A^k\right\}_k \in K$.

- ***Multiple binary hypothesis tests***: perform binary test H_N versus H_A^k and compute log-likelihood ratio (LLR).

- ***Final decision***: once all the alternative hypotheses have been tested, find LLR_max of LLR^k. If LLR_max is greater than a threshold, the null hypothesis is reject and the alternative hypothesis with LLR_max is considered as estimate of the image compression history.

2.2.4 Tampering Localization

Nowadays, the diffusion of tampered visual contents through the digital world is increasing because of the large availability of simple and effective image and video processing tools (Photoshop, Lightroom and so on). Due to this issue, the development of techniques for detection of image tampering operations changing the content of an image is getting more and more attention from a forensics point of view and many image forensic techniques have been proposed to detect the present of forgeries in digital image.

The authors of [8] mentioned that the detection of cut & paste operations whereby a portion of a source image is copied into a target image plays a crucial role, since this is the most common way of changing the semantic content of an image. Thus, authors proposed two algorithms for detection of image regions that have been transplanted from another image. The proposed methods work whenever the pasted region is extracted from a JPEG compressed image and inserted into a target image that is subsequently compressed with a quality factor (QF) larger than that used to compress the source image. In proposed methods, authors assume that the tampered image is generated by taking I region R from a source image S and pasting it into a target image T generating a fake image K. They assume that both images S and T are in JPEG format. Moreover, they assume that the tampered image K is JPEG compressed again and stored after the insertion of R within T. It is obvious that all the regions in K undergo a double JPEG compression, however in the part of T that have not been replaced by R, the two subsequent compressions used aligned 8×8 grids. The rationale behind the first algorithm Block-wise approach is that authors decided to consider a 128×128 region centered on the analyzed block. In practice, for each block a blocking artifact characteristics matrix (BACM) is built by considering the surrounding 128×128 area and the 14 features describing the symmetry of the BACM extracted. Then they are training neural network with 5000 JPEG images and obtained a tampering map with dark area corresponding to tampered regions. An example of the result produced by the block-wise detector is shown in Figure 2.10. The original and the tampered images are in the first row, respectively on the left and on the right. The tampering map is shown in the last row, and we can see that there are two pasted flowers are detected.

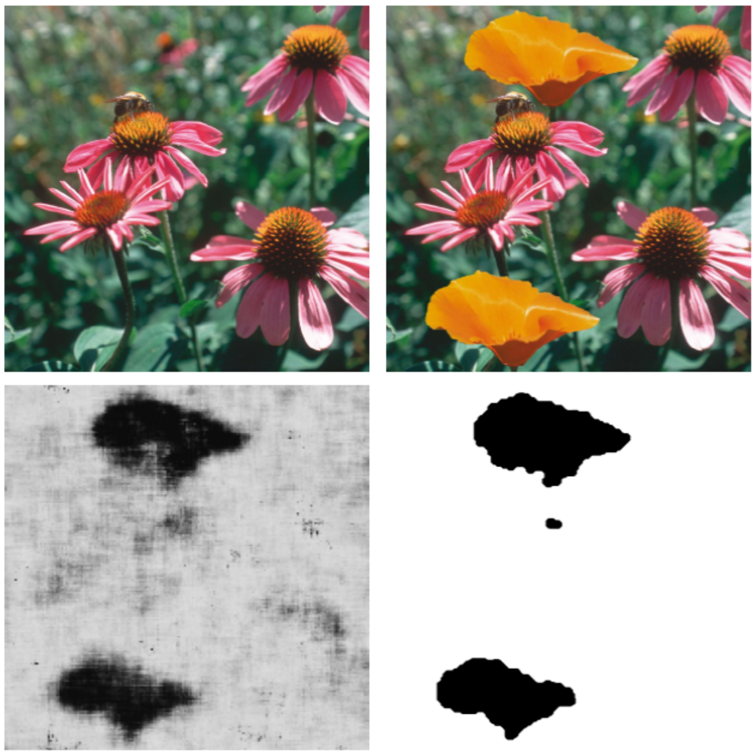


Figure 2.10: Example of blockwise tamper detection [8].

However, there are some drawbacks existing in *Block-wise approach*. Therefore, the authors proposed a *Region-wise approach* to improve the last approach, they first segment the image into homogeneous regions and then analyze each region separately. They build the BACM of each region by analyzing only the blocks belonging to it and used the features extracted from the BACM to classify the whole region at once. By doing in this approach, the computing time is reduced.

The authors of [9] proposed a forensic algorithm to discriminate between original and forged regions in JPEG images, under the hypothesis that the tampered image presents a double JPEG compression, either aligned (A-DJPG) or nonaligned (NA-DJPG). The proposed algorithm automatically computes a likelihood map indicating the probability for each 8×8 discrete cosine transform block of being doubly compressed. An example of this algorithm is reported in Figure 2.11, (a) images under analysis; (b) likelihood maps obtained using the A-DJPG simplified model; and (c) likelihood maps obtained using the NA-DJPG simplified model. Red/blue areas correspond to high/low probability of being doubly compressed. On the left side, the proposed algorithm shows that there is a high probability of the pyramid to be doubly compressed according to NA-DJPG model. On the right side, the proposed algorithm shows that the license plate has a high probability of being singly compressed, whereas the rest of the image has a high probability of being doubly compressed according to A-DJPG model. Quality settings are QF1=60, QF2=95 (left side), QF1=90, QF2=95 (right side).

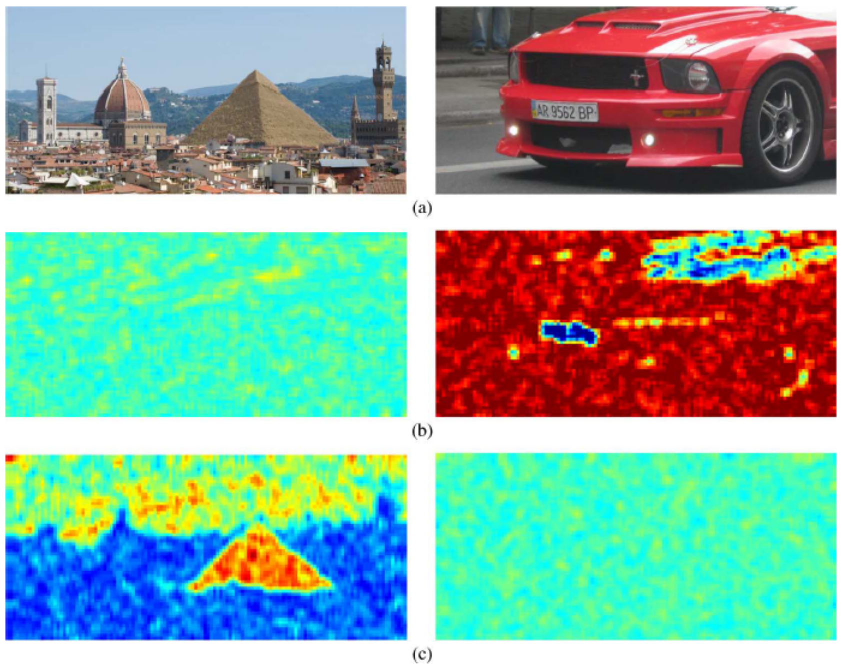


Figure 2.11: Application to realistic forgeries [9].

Chapter 3

JPEG Compression Detection

In the previous chapter, we briefly introduced some background on JPEG and some state-of-the-art detection methods for JPEG compression detection and how those algorithms work for different situations. In this chapter, we focus on the JPEG compression detection methods thoroughly analyzed in this thesis, highlighting improvements over the state-of-the-art. Specifically, we first tackle JPEG detection problem under the assumption that we know in advance a possible JPEG grid alignment. To this purpose we analyze two algorithms. Then, we remove the made assumption, and focus on a detector able to estimate the possible JPEG grid location.

3.1 Pixel Domain Compression Detection

The first method we analyze for JPEG compression detection is the one proposed in [10, 14]. This method is based on the analysis of JPEG artifacts directly in the pixel domain.

The rationale behind this method is that JPEG compression introduces blocking artifacts at the boundary of each 8×8 pixel block. Therefore, it is possible to detect the presence of these artifacts from a pixel-based analysis. Presence of these traces indicate JPEG compression.

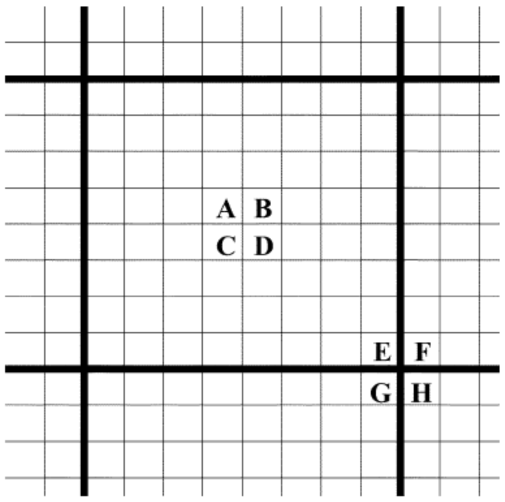


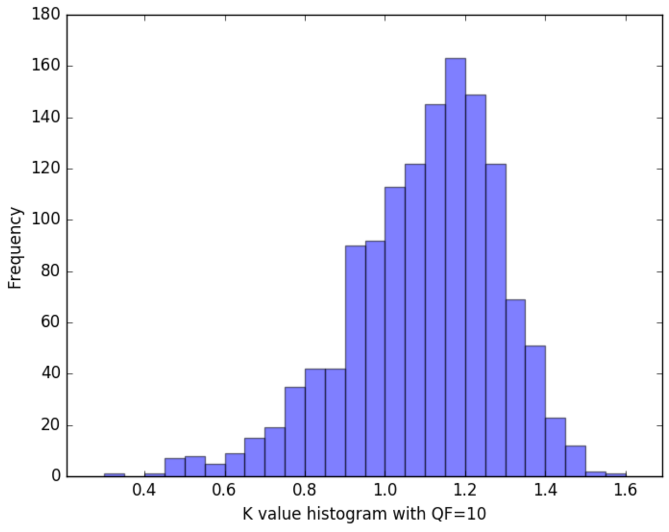
Figure 3.1: For each block the number $Z'[i,j] = |A - B - C + D|$ *and* $Z''[i,j] = |E - F - G + H|$ *[10].*

Formally, this algorithm (hereinafter denoted as detector one, or D1) works as it follows. Let us assume we are analyzing an image I. The first step of the method is image 8×8 block decomposition. Before this process, what we need to be careful is converting RGB to grayscale before we load the image. Then author explain that even less compressed image may leave small but consistent discontinuities across block boundaries, so the algorithm is that if there is no compression the pixel difference across blocks should be similar to those within blocks and spanning across a block boundary. We know that the block grid is 8×8, then we need to compute a sample of difference within a block and spanning across a block boundaries, as presented in Figure 3.1.

Figure 3.1 point out that for each decomposition 8×8 block we need to compute number $Z'[i,j] = |A - B - C + D|$ and $Z''[i,j] = |E - F - G + H|$ (A, B, C, D, E, F, G and H are the value of its corresponding pixels). For example, when we load an image with 384×512, we have to compute Z'' and Z' for 48×64 times and save those values into a 2D array. Then continue to compute the normalized histogram $H_1(n)$ and $H_2(n)$ of the $Z'[i,j]$ and $Z''[i,j]$, respectively. The blocking signature measure that we use is the energy of the difference between two histograms.

$$K = \sum_n H_1(n) - H_2(n)$$

We computed several K values based on different quality factors images and images with no compression and plot six histograms with related quality factors. The range of K value (X axis) from 0 to 1.6, the sum of frequency for compressed image is the number of input images and we simply call this image compression algorithm D1.



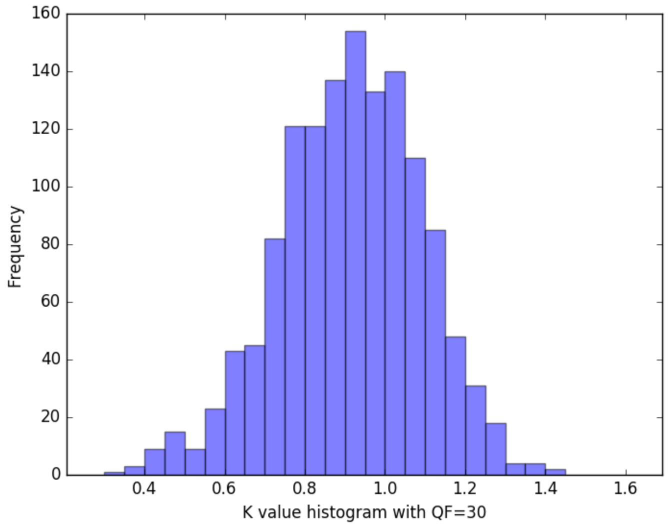
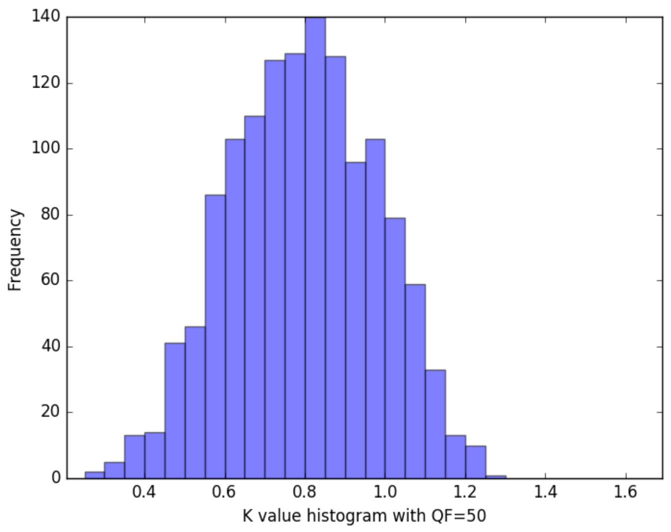


Figure 3.2: Histogram of K with different quality factors 10 and 30.

As Figure 3.2, Figure 3.3 and Figure 3.4. It is obvious that K value with no compression images mainly distribute from 0 – 0.1, with quality factor 90



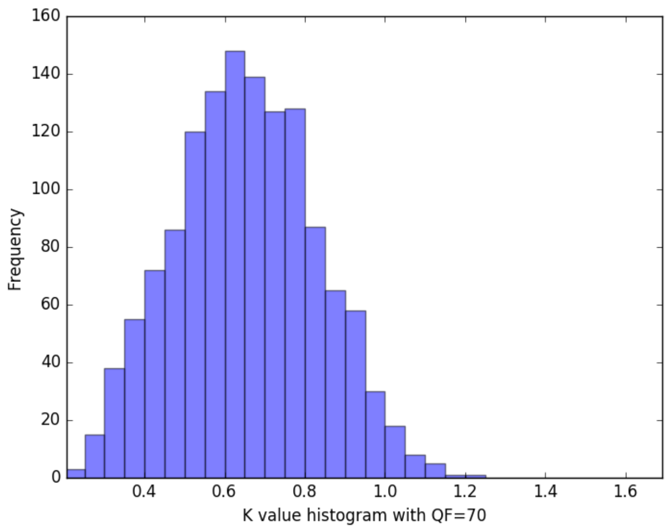
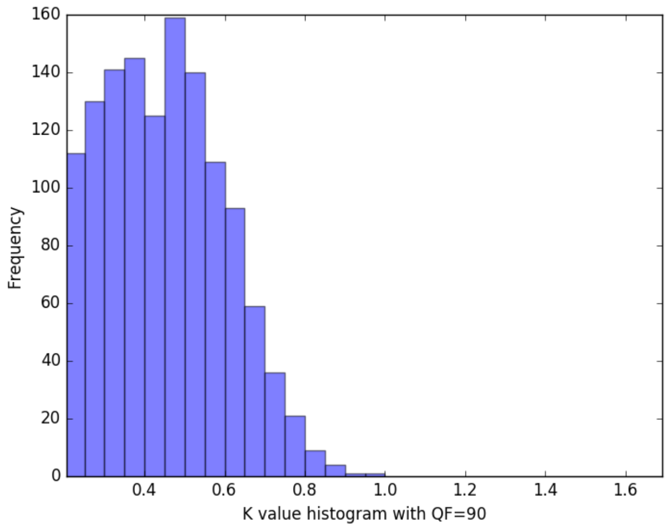


Figure 3.3: Histogram of K with different quality factors 50 and 70.

images mainly distribute from 0 – 0.6, with quality factor 70 images mainly distribute from 0.5 – 0.8, with quality factor 50 images mainly distribute from 0.6 – 1.0, with quality factor 30 images mainly distribute from 0.8 – 1.2, and with quality factor 10 images mainly distribute from 0.9 – 1.4. The quality factor is higher, the less is the K value (highly compressed image has higher K value). Also, K can be compared to a threshold or given as a confidence parameter.



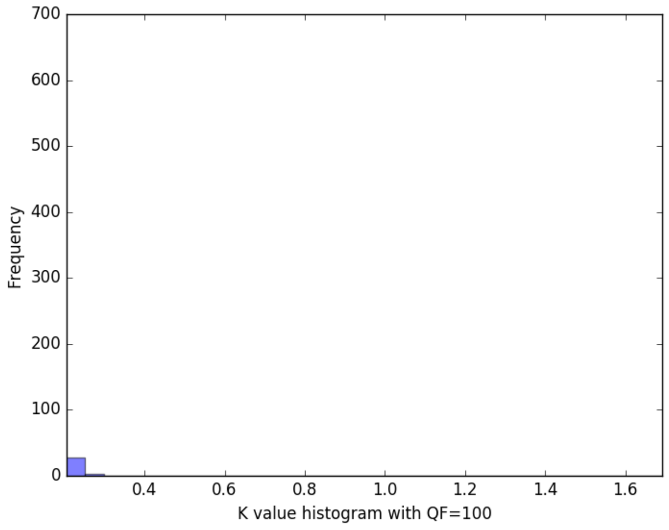


Figure 3.4: Histogram of K with different quality factors 90 and 100.

Figure 3.5 serves to illustrate the method. The top figure of Figure 3.5 shows histograms $H_1(n)$ and $H_2(n)$ for a typical image with QF 90, the absolute histogram differences K is 0.48969942587. The bottom figure of Figure 3.5 shows the same after the same image underwent compression with QF 10 and K is 1.12326916582.

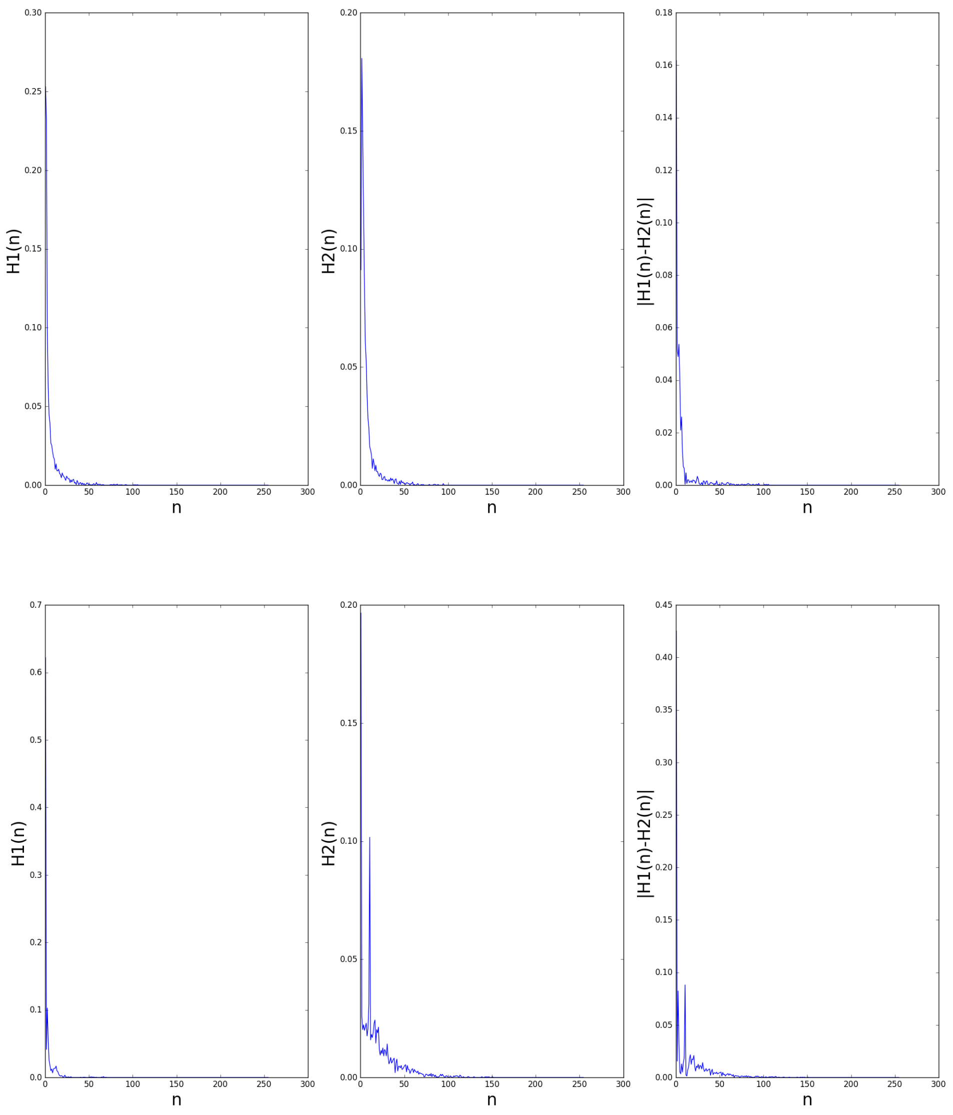


Figure 3.5: Histograms of region I (within block) and II (across blocks) and those difference. Top figure is related to QF 90, bottom figure is related to QF 10.

3.2 Pixel Domain Cropped Image Detection

The algorithm presented above assumes that the grid origin is the same as the image origin point, but the pre-condition is that there is no image cropping or pasting. In this section, we show how authors of [10, 14] propose to detect

JPEG grid alignment in case of cropping.

They let $f(m,n)$ be the image pixels, then the grid origin can be chosen as the pair $\{(p,q)|0 \leqslant p \leqslant 7, 0 \leqslant q \leqslant 7\}$ that maximizes E_{pq}, where

$$E_{pq} = \sum_i \sum_j |f(8i+p, 8j+q) - f(8i+p, 8j+q+1) - f(8i+p+1, 8j+q) + f(8i+p+1, 8j+q+1)|$$

We can observe that the grid should be aligned with the position where horizontal and vertical neighbor difference, in a periodic displacement, are at their maximum. If there is no compression, therefore, no blocking and all E_{pq} should be similar and the grid origin will be picked randomly.

Once we implement this method (D2) to detect the grid position of cropped image, we obtain a matrix which stores all E_{pq} for each block of an image. And the position of maximum value in matrix presents the p and q, respectively grid position of cropped image. An example shows below:

$$M = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & max & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

The p and q are [5,5], and cropped pixels of detected image are [2, 2].

In Chapter 3.1, we introduced the algorithm for pixel domain compression detection, what if we implement that algorithm to detect the cropped image. However, the problem of detecting cropped image by using first compression detector is that we don't know exactly what are the values of p and q, that means algorithm does not know what are the values of the pixels in the positions (from A through H). However, we can observe that p and q are fixed values (p=q=3) when we implement the first compression detector, but p and q will be flexible based on which grids have been cropped when we input cropped image with first detector.

Thus, for detecting cropped image, the first step is that implementing the algorithm of cropped image detection to compute the grid position E_{pq}. Then passing the new values p and q to the first compression detector to compute K value and the new cropped image detection algorithm has been generated and named it as D21.

3.3 Pixel Domain Improved Cropped Image Detection

The method just presented for JPEG grid alignment estimation can be further improved using some considerations. As a matter of fact, it is possible to notice experimentally that picking the maximum value of matrix E_{pq} may lead to wrong results. Instead, we propose to select p as the index of the row of E whose elements sum to the maximum value among all rows. In the same way, we select q as the index of the column of E_{pq} whose elements sum to the maximum value among all columns. Formally,

$$\begin{aligned} q &= \arg\max_q (\sum_p E_{pq}) \\ p &= \arg\max_p (\sum_q E_{pq}). \end{aligned} \tag{3.1}$$

As shall be reported in the experimental section, this solution allows to improve results obtained with detector D1. Indeed, it provides a more accurate estimation of p and q parameters.

3.4 Transform Domain Compression Detection

In Chapter 2, we mentioned a method for transform domain compression detection [6], which computes the horizontal gradient and vertical gradient of image, and use DFT to estimate their periodicity due to gradient peaks at block boundaries in frequency domain. Thus, we can use gradient information to estimate the block position and blockiness distortion evaluation can be computed. Moreover, this method is not affected by possible JPEG grid misalignment due to cropping as it works in the frequency domain.

Due to this possibility, the second algorithm we focused on is the one in [6], and hereinafter denoted as D3. This paper explains a novel method for estimating blockiness in MPEG-coded picture by measuring the harmonic generated by blocking artefacts in the frequency domain. However, we can also implement this algorithm on JPEG compression detection. In the following a detailed step-by-step explanation of how to do it.

Harmonic analysis: Blockiness is made by luminance discontinuities across the DCT block boundaries. We know that the DCT block size is fixed to 8×8 by JPEG standard and luminance discontinuity is thus periodic. So, we can use Sobel operators to detect the luminance variation in an JPEG-coded image.

Sobel operator is a derivate mask and is used for edge detection. This operator is also used to detect two kind s of edges in an image: vertical direction and horizontal direction.

The Figure 3.6 shows the vertical and horizontal mask of Sobel operator. When we apply the vertical mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in an edge region. As the center column is zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also the center values of both the first and third column is 2 and -2 respectively. It gives more weight age to pixel values around the edge region. This increase the edge intensity and it became enhanced comparatively to the original image.

The horizontal mask will find edges in horizontal direction and it is because that zeros column is in horizontal direction. When you will convolve this horizontal mask onto an image, it would prominent horizontal edges in the image. The only difference between it is that it has 2 and -2 as a center elements of first and third row.

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Sobel

Figure 3.6: Vertical and horizontal masks of Sobel operator [11].

Figure 3.7 reports that the original image and two images applied above two masks at one time. For the computation of horizontal and vertical gradient of an image, we need to use image gradient formula such that

$$G_x = f(x+1, y) - f(x, y)$$
$$G_y = f(x, y+1) - f(x, y)$$

Where G_x used to compute horizontal gradient and G_y used to compute and vertical gradient.

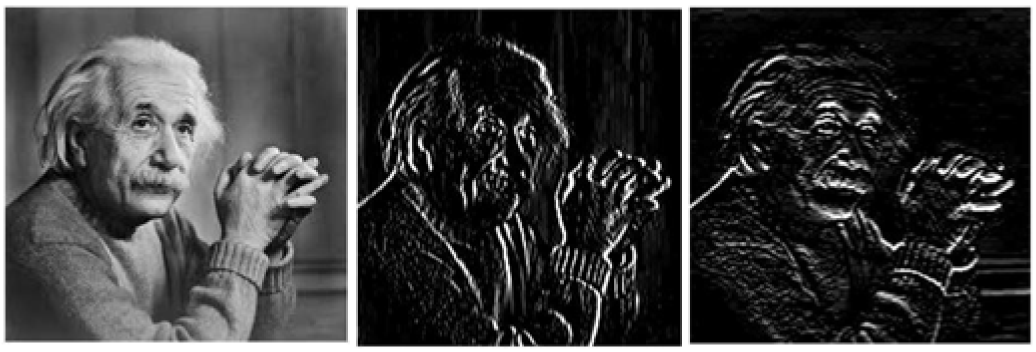


Figure 3.7: Sobel operator example [11].

The extracted luminance gradient information forms a gradient image. Figure 3.8a shows an ideal 32 × 32 gradient image, in Sobel operators, each luminance transition generates two pixels in the gradient image. The gradient image appears as a lattice pattern with a grid width of two pixels. Figure 3.8b shows the luminance level of pixels along line AB and Figure 3.8c presents actual lattice pattern image. The last figure 3.8d tells us that

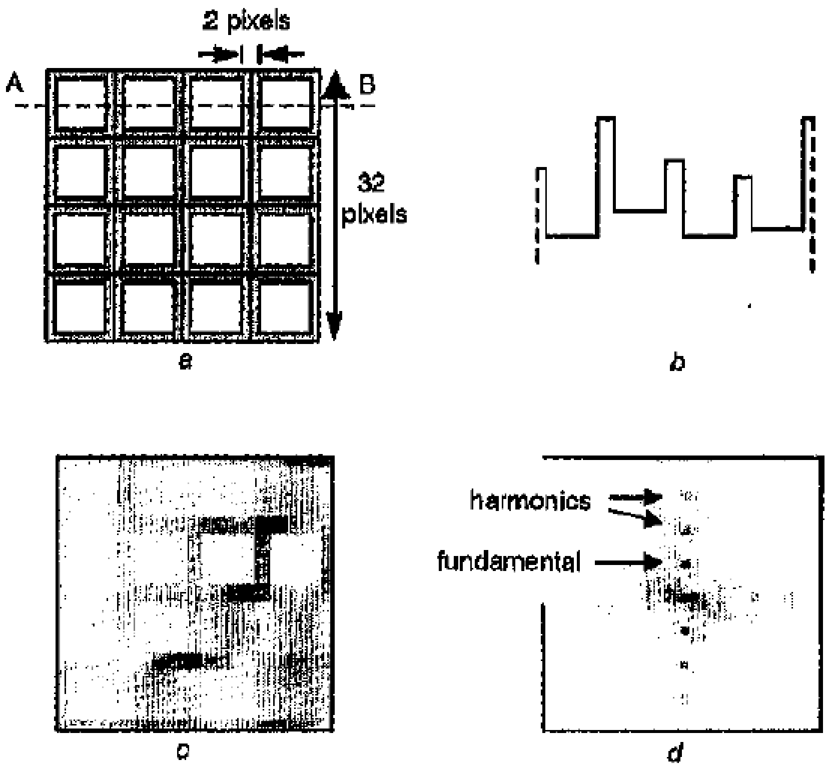


Figure 3.8: Basic concept of harmonic analysis [12].

fast Fourier transform has been used on this block, and it's obvious that we can see some outstanding frequency points along vertical axis. We found that the more obvious the lattice pattern is in the gradient image, the stronger the harmonics are. Therefore, we use this point to estimate the degree of blockiness in the codes picture, author quantifies the blockiness using the relative strength of harmonics compared to other frequency components, and the absolute strength of the harmonics. They define two ratios R_h and R_v where H_n and V_n are the magnitude of the nth frequency component on the horizontal and vertical axes.

$$R_h = \frac{H_4 + H_8 + H_{12}}{\sum_{15}^{n=1} H_n}$$

$$R_v = \frac{V_4 + V_8 + V_{12}}{\sum_{15}^{n=1} V_n}$$

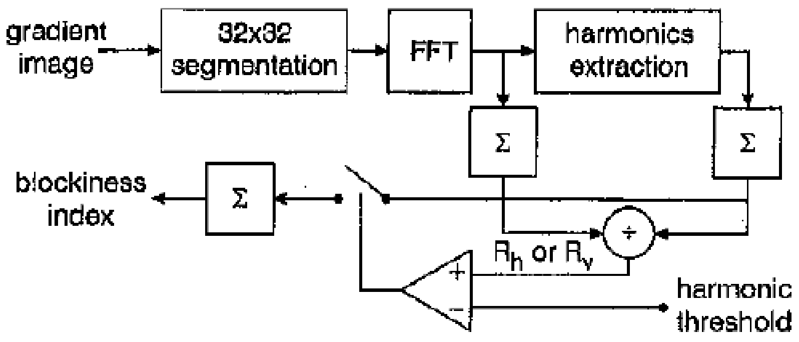


Figure 3.9: Harmonic analysis process for estimating blockiness [12].

Figure 3.9 illustrates the complete harmonic analysis process which includes the following steps: Given an image,

- 32×32 block segmentation
- Compute the horizontal gradient of the image.
- Compute the FFT of horizontal gradient.
- Compute the vertical gradient of the image.
- Compute the FFT of vertical gradient.
- Compute two ratios R_h and R_v

At this point we have two values (R_h and R_v) indicating blockiness likelihood for each 32×32 image block. We need therefore to devise a way to put these values together to take decision about possible JPEG compression at image level. Let us assume that the blockiness level as L. The first method we propose to merge all $R_h$and R_v results for each 32×32 image block consists in taking the maximum values of each R_h and R_v. Formally, L=max(R_h,R_v). This idea is driven by the fact that, even if just one block presents JPEG artifacts, we conclude that the image has been compressed. The second method we propose to merge all R_h and R_v results for each 32×32 image block consists in taking the minimum values of each R_h and R_h. Formally, L=min(R_h,R_v). The rationale behind this method is that when threshold is less that the L, then the image has been compressed. And we also consider other possible ways to merge R_h and R_v such as L=median(R_h,R_v).

Chapter 4

Implementation and GUI

In this chapter we report implementational details about the used software, and the development of the GUI embedding all previous mentioned algorithms.

4.1 Experimental Software

The software we used in our experimental includes one Python IDE PyCharm, a library of programming functions for computer vision OpenCV, Numpy which is the fundamental package for scientific computing with Python, Matplotlib is a Python 2D plotting library which produce publication quality figures in a variety of hardcopy and interactive environments across platforms. PIL is the Python imaging library and SciPy which is an open source Python library used for scientific computing and technical computing. And we list our software and its version in Table 4.1.

In order to develop image compression detector application, we need to install PyQt for designing GUI, PyQt is most popular Python bindings for the Qt cross-platform GUI/XML/SQL C++ framework. There are two available editions Qt 4 and Qt 5 which will build for Python 2 and Python 3 and we used Qt 4 because Python 2.7 is being used for implementation and development.

Library, framework and environment	Version
OpenCV	3.1.0
Numpy	1.11.2
Matplotlib	1.5.1
SciPy	0.16.1
PIL	1.1.6
PyQt	4.11.4
Pillow	3.2.0
PySide	1.2.4
Python	2.7.11
IrfanView	64

Table 4.1: Implementation software

There is one reason to install OpenCV fails on MacOS with Xcode 8. Once you updated your Xcode version to 8, the 'QTKit/QTKit.h' file will be not existed on your new Xcode. There are two solutions for people who want to use OpenCV on Mac, the first one is that using old version of Xcode, and second is that using HomeBrew to install Head OpenCV3 on your Xcode 8.

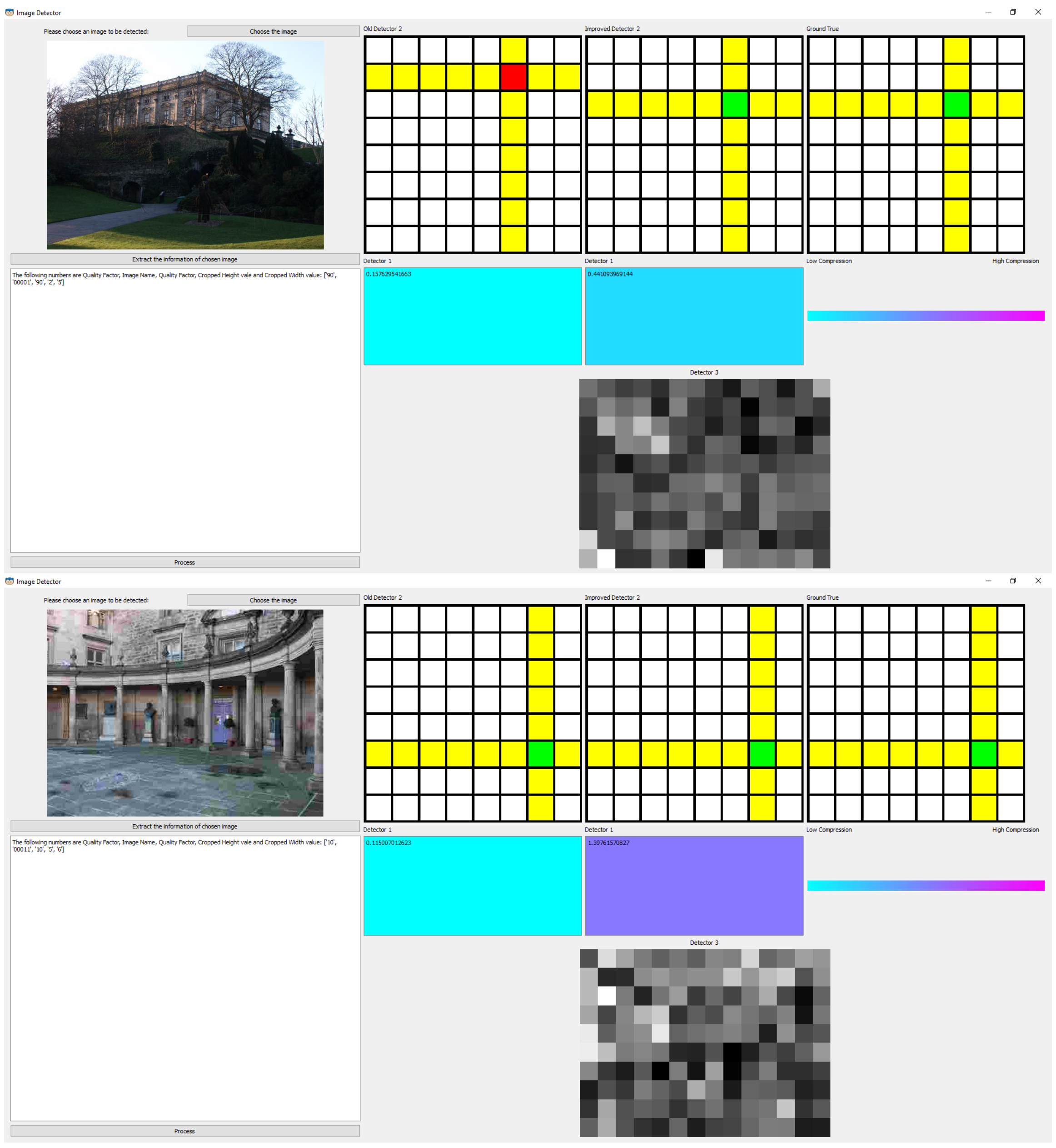
4.2 Compressed Image Detector GUI Implementation

In previous sections of chapter 3, we have introduced image compression detection algorithm D1 and D3, cropped image grid position detection algorithm D2 and cropped image detection algorithm D21 and improved version of D21. After implement those three detection algorithms, we intent to create a compressed image detector GUI using PyQt 4 and we named it Image Detector.

This Image Detector includes all the algorithms we discussed and improved in chapter 3, which has following functional requirements:

- Able to select cropped and compressed image from folder.
- Able to extract ground truth information when available (i.e., quality factor, cropping details).
- Able to implement the original D2 algorithm to obtain the grid position and display on grid image based on input image.
- Able to implement the improved D2 algorithm to obtain the grid position and display on grid image based on input image.
- Able to display ground truth crop data on grid image with green color based on input image.
- Able to use green color to display the position on grid image when the result of the old D2 and the improved D2 match with the ground true data, able to use red color to display the position on grid image when the result of the old D2 and the improved D2 do not match with the ground true data.
- Able to implement the D1 algorithm to compute K and display on GUI with color based on input image.
- Able to implement the improved D21 algorithm to compute K and display on GUI with color based on input image.
- Able to display a color bar to show the user how much the image has been compressed.
- Able to implement the D3 algorithm and display the matrix with image based on input image. User can see which block of image has been highly or less compressed.

Figure 4.1 shows the Image Detector GUI with input images and their results. The improved detector D21 always has better performance than old detector D21. As the results of those two input images, light block shows highly compression and dark block presents less compression.



The following numbers are Quality Factor, Image Name, Quality Factor, Cropped Height vale and Cropped Width value: ['90', '00001', '90', '2', '5']

0.157629541663

0.441093969144

The following numbers are Quality Factor, Image Name, Quality Factor, Cropped Height vale and Cropped Width value: ['10', '00011', '10', '5', '6']

0.115007012623

1.39761570827

Figure 4.1: Compression detection GUI

Chapter 5

Experimental Results

In this chapter we report the experimental validation of the analyzed techniques. First, we report details about the construction of the test dataset. Then we present the used evaluation metrics. Afterwards, we report results on JPEG detection based on pixel-domain and transform-domain detectors (D1 and D3), showing the behavior of our proposed solutions to merge results from different blockiness information. Additionally, we validate the grid detection algorithm, showing the performance increase due to our proposed modification. Finally, we validate the system as a whole, feeding results obtained from the grid detector to the pixel-domain-based JPEG detector (i.e., D21).

5.1 Dataset Generation

In order to test the mentioned algorithms, we need a certain number of uncompressed and compressed images. The uncompressed image dataset (UCID)[24] as shown in Figure 5.1, which includes 1338 images with TIF format comes from Loughborough University.

In the previous chapters, we mentioned that different quality factors can be used in JPEG compression. Thus, we used the application IrfanView to compressed dataset with six different quality factor QF 10, QF 30, QF 50, QF

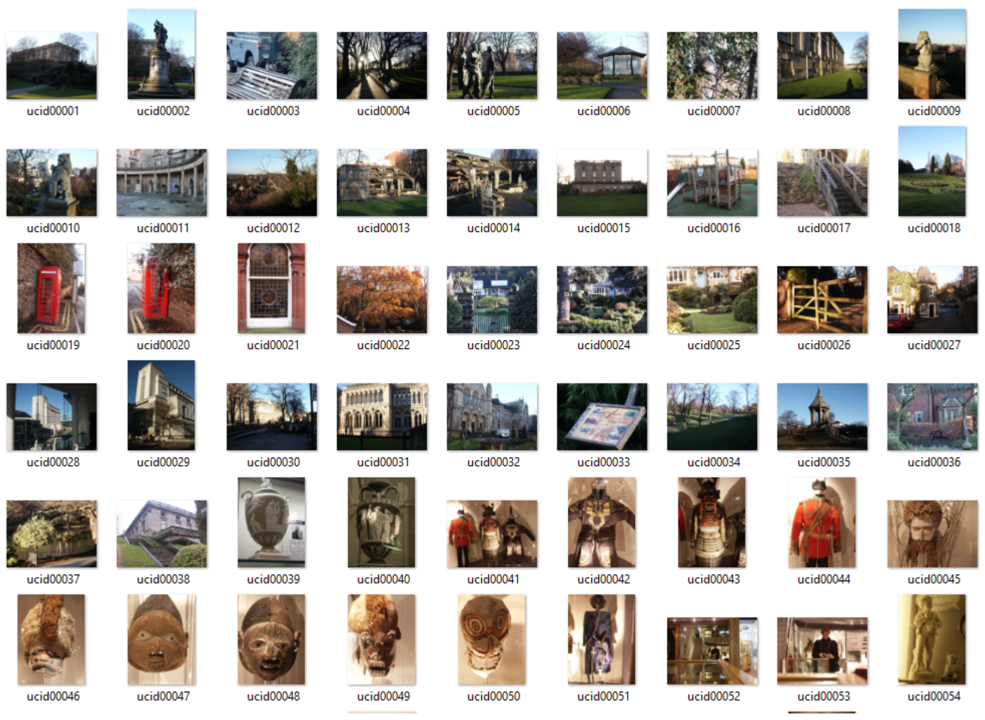
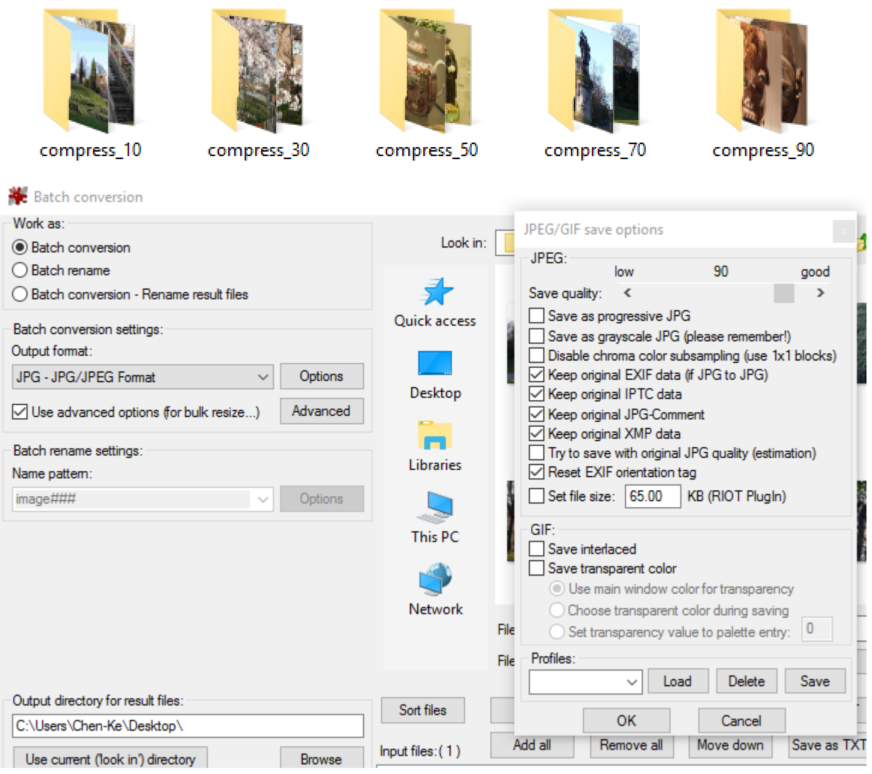


Figure 5.1: UCID

70 and QF 90 as Figure 5.2 shows. We ended up with five different datasets (one per quality factor) of 1338 images each. Notice that different JPEG implementations might use different quantization matrix. For this reason we specify the used implementation for the sake of reproducibility of results.

In order to test the algorithms on cropped images, we propose to randomly crop images for each dataset with different QF. We therefore ended up with five additional datasets for each QF.



Batch conversion

Work as:
- [✓ Batch conversion]
- Batch rename
- Batch conversion - Rename result files

Batch conversion settings:
Output format: JPG - JPG/JPEG Format — Options
[✓] Use advanced options (for bulk resize...) — Advanced

Batch rename settings:
Name pattern: image### — Options

Output directory for result files: C:\Users\Chen-Ke\Desktop\
Use current ('look in') directory — Browse

Look in:
Quick access
Desktop
Libraries
This PC
Network
Sort files
Input files: (1)
Add all — Remove all — Move down — Save as TXT

JPEG/GIF save options

JPEG:
Save quality: low 90 good
- [ ] Save as progressive JPG
- [ ] Save as grayscale JPG (please remember!)
- [ ] Disable chroma color subsampling (use 1x1 blocks)
- [✓] Keep original EXIF data (if JPG to JPG)
- [✓] Keep original IPTC data
- [✓] Keep original JPG-Comment
- [✓] Keep original XMP data
- [ ] Try to save with original JPG quality (estimation)
- [✓] Reset EXIF orientation tag
- [ ] Set file size: 65.00 KB (RIOT PlugIn)

GIF:
- [ ] Save interlaced
- [ ] Save transparent color
  - Use main window color for transparency
  - Choose transparent color during saving
  - Set transparency value to palette entry: 0

Profiles: Load — Delete — Save

OK — Cancel

Figure 5.2: Image datasets with different QF by using IrfanView

5.2 Evaluation Metrics

Algorithms D1 and D2 output a real number that should be compared to a threshold to assess presence of JPEG compression. To evaluate these algorithms, we rely on ROC (Receiver Operating Characteristic) curves, which are a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. Specifically, TPR is the percentage of JPEG compressed images correctly detected as such. FPR is the percentage of compressed images mistakenly detected as uncompressed. Formally,

True positive rate: $TPrate = Recall = \frac{TP}{TP+FN} = \frac{TP}{relevant}$.

False positive rate: $FPrate = \frac{FP}{FP+TN} = \frac{FP}{non-relevant}$.

Table 5.1 reports definitions of TP and FP in relation to relevant (or not) retrieved (or not) classes.

	Relevant	Non-relevant
Retrieved	true positive (TP)	false positive (FP)
Non-Retrieved	false negative (FP)	true negative (TN)

Table 5.1: Unranked retrieval

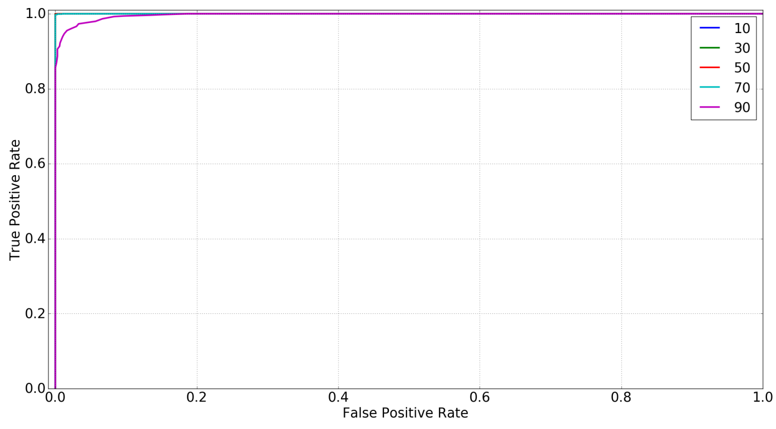
We can set that the threshold range from 0 to 0.3, and compute the prediction based on the data greater than threshold or not. If data is greater than threshold, then we store in prediction as 1 and it has been compressed, otherwise we store it as 0 and it has not been compressed. After that we can compute the TP, FP, TN, and FN of each dataset and plot ROC curve.

5.3 JPEG Compression Detection With A Priori Grid Information

In this section, we introduced the results of the previous algorithms mentioned in pixel domain compression detection and transform domain compression detection.

5.3.1 Pixel Domain Compression Detection

The first set of ROC curves in Figure 5.3 shows that the compressed image datasets with quality factor 10, 30, 50 and 70 are overlapped when we implement algorithm D1. In this case, JPEG compression detection is almost ideal. However, these results assume that JPEG grid alignment is known beforehand. Conversely, if this is not known, results change considerably. The second set of ROC curves in Figure 5.3 are related to cropped and compressed image dataset with quality factor 10, 30, 50, 70 and 90. It is possible to see that in this scenario the algorithm often fails to detect JPEG compression as expected.



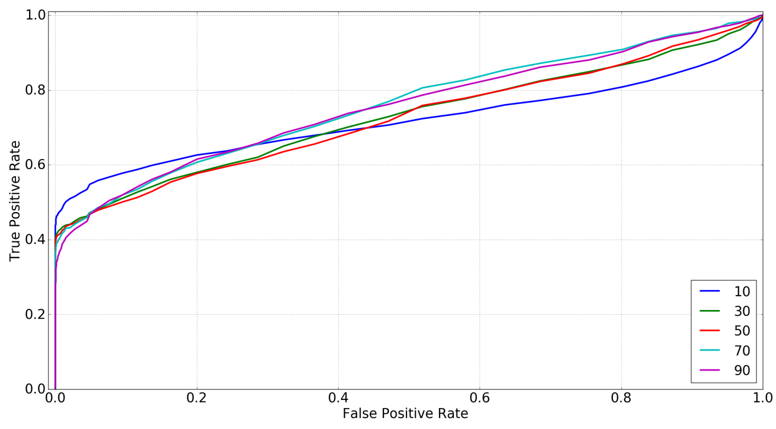


Figure 5.3: ROC of compress image datasets and compressed & cropped image datasets with pixel domain compression detection.

5.3.2 Transform Domain Compression Detection

In pixel domain compression detection, in order to observe the whether the image has been compressed, we input two images (same image with compression and without compression), result shown as Figure 5.4. The left one is the image without compression, the right plot is the image with highly compression (quality factor = 30). And it is obvious that more warm the color tone is in image, the stronger compression the image is.

We already explained in Chapter 4 that how to analyze the data with ROC curve and we need to implement ROC curve on this data analysis

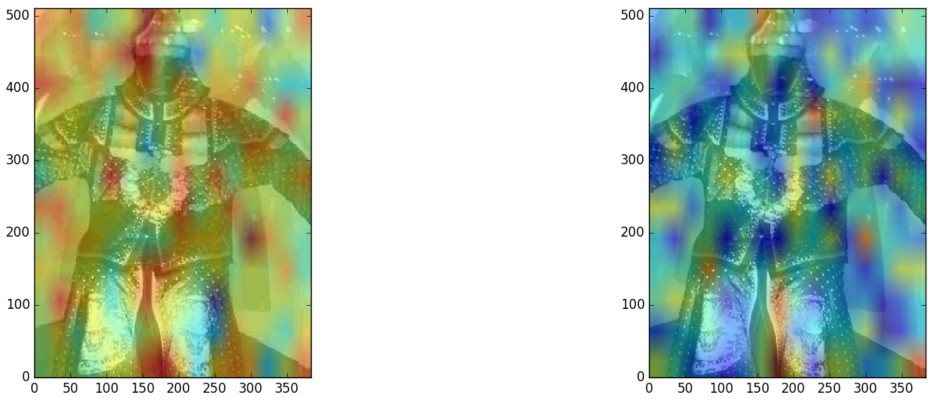


Figure 5.4: Same image with different QF implemented by transform domain compression detection.

as well. However, the problem different with last algorithm is that which number we are going to choose for comparison with the threshold. Indeed, D3 outputs two values (R_h and R_v) for each image block. We therefore have a collection of two matrices of size 16×12. In Chapter 3.2, we mentioned that how does the algorithm works and several methods to merge all R_h and R_v results for each 32×32 image block of an image. Therefore, we try several combinations of chosen value and operator between threshold and feature values. As shown in Table 5.2, we list all the combinations. As Table

Extract a value from matrix	Operator (threshold and feature values)	Result
Max	Greater	Sufficient
Min	Less	Sufficient
Median	Less	Insufficient

Table 5.2: Method combinations of transform domain compression detection

5.2 and Figure 5.5 show, only combinations (Max, Greater) and (Min Less) show promising results. Compared to D1, on average D3 performs better considering that JPEG grid position is not known. If the grid is known, D1 is a better choice.

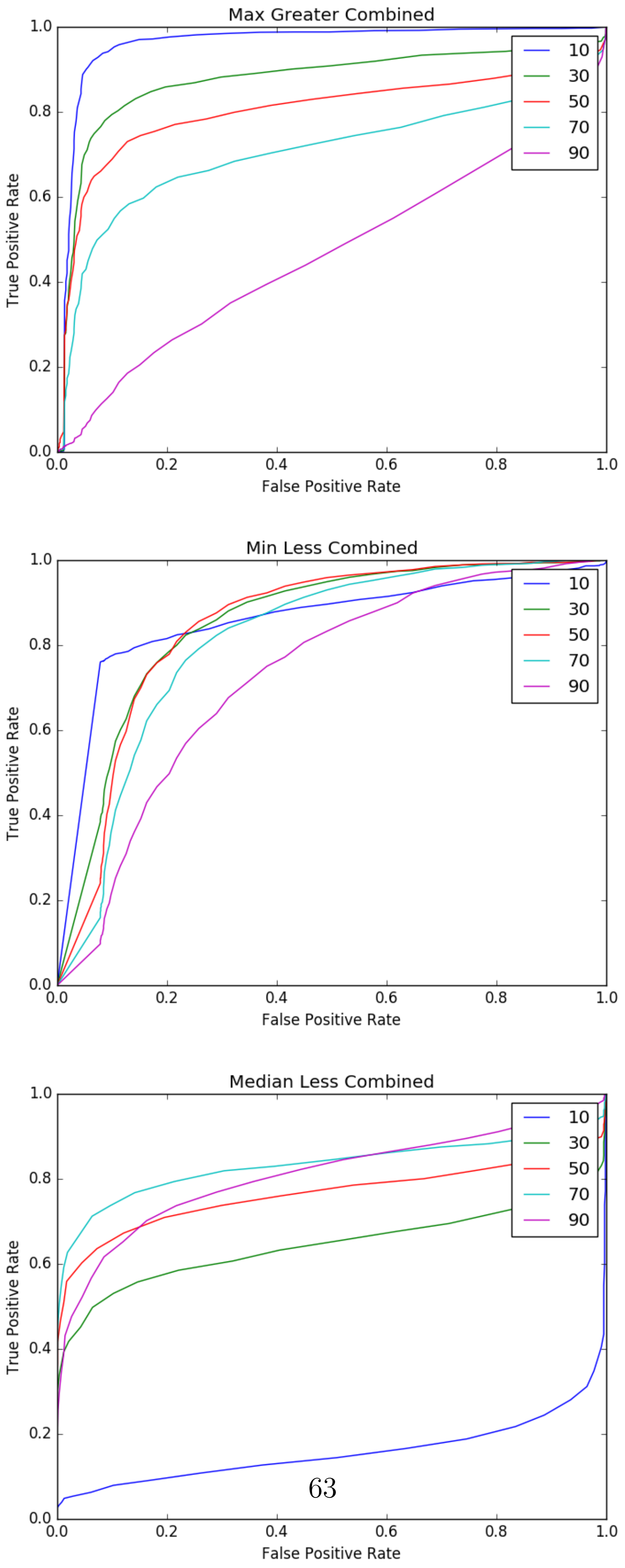


Figure 5.5: ROC curve for all proposed methods.

5.4 JPEG Grid Position Estimation

We have explained that detector (D2) is built to estimate the position of JPEG grid exploiting blocking artifacts. If there is no compression, no blocking artifacts are present, thus all E_{pq} values computed by D2 should be similar. The grid origin in this situation is picked randomly. Conversely, when an image is JPEG compressed, the origin of the JPEG grid should be correctly identified. Figure 5.6 shows the accuracy obtained in estimating JPEG grid for cropped and compressed images at different quality factors. It is possible to see that for high QF, accuracy is not achieved with our expectation.

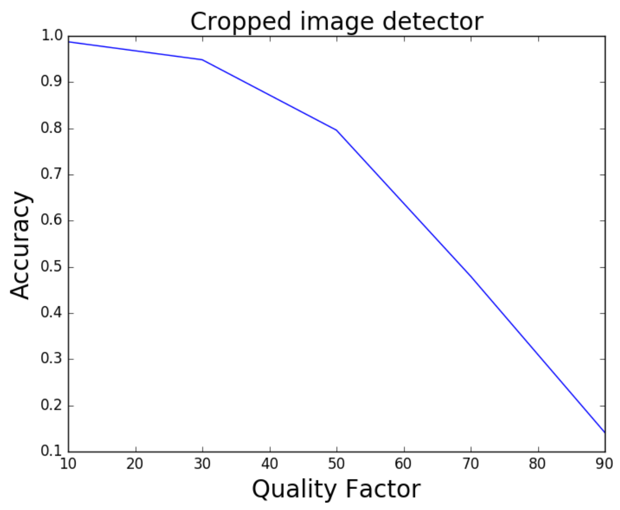


Figure 5.6: Histogram of result of cropped image detection algorithm.

In order to test the accuracy of this method, we tested several cropped images and obtained the results comparing with the clipping data of cropped images. We found the problem that the p and q values could not match the clipping data all the time for some cropped images. For example, we input a cropped image which cropped with factor [1, 0] (that means crop the first column of the image, did not crop the row of the image, pixels 512×384 changed to pixels 512×383) and we obtained the result as shown in Matrix M_{result}.

$$M_{result} = \begin{bmatrix} 24419 & 24529 & 24745 & 24500 & 24826 & 28140 & \mathit{35174} & 27137 \\ 24270 & 23910 & 24543 & 24605 & 24414 & 25856 & \mathit{30831} & 25754 \\ 24417 & 25561 & 25435 & 24951 & 24093 & 27107 & \mathit{32125} & 27500 \\ 24074 & 25642 & 24804 & 24957 & 24263 & 26516 & \mathit{32956} & 27209 \\ 24277 & 25607 & 25408 & 24992 & 25130 & 27410 & \mathit{32030} & 25959 \\ 24201 & 24409 & 24514 & 24419 & 24348 & 26181 & \mathit{30534} & 26811 \\ 25000 & 25080 & 25667 & 24733 & 24304 & 26313 & \mathit{35488} & 27724 \\ \mathit{40196} & \mathit{38817} & \mathit{38275} & \mathit{39905} & \mathit{38989} & \mathbf{42446} & \mathit{37137} & \mathit{42196} \end{bmatrix}$$

This matrix shows the maximize value of the matrix is 42446, the q1 and p1 are 5 and 7. However the clipping factor is [1, 0], and convert it to p and q are [7-1, 7-0], the result which are q2 = 6, p2 =7. We can see that $q1 \neq q2$, p1=p2, the way of finding p and q where p and q come from maximize value of matrix has problem. This is the motivation behind out improved solution presented in Chapter 3.

Matrix M_{result} shows the matrix E_{pq} and the p and q values of same image we input in last case. Obviously, we can see that the maximize average summing column and row are 5th column and 7th row. Simultaneously, the q1 and p1 are [5, 7], and it matched the clipping data [1, 0] where q2=7-1=6, p2=7-0=7. Thus, p1=p2,q1=q2 and the new D2 works well on other cropped images.

5.5 JPEG compression detection in the wild

After the improvement of D2 has been validated, we can implement it as additional step before running D1 and create the improved algorithm D21, but there is an issue we need to consider that values passing between D2 and D1. In D1 algorithm, we know that

$$Z'[i,j] = |A - B - C + D|$$
$$Z''[i,j] = |E - F - G + H|$$

$$E_{pq} = \sum\sum |y(8i+p, 8j+q) - y(8i+p, 8j+q+1) - y(8i+p+1, 8j+q) + y(8i+p+1, 8j+q+1)|,$$

where $\{(p,q)|0 \leqslant p \leqslant 7, 0 \leqslant q \leqslant 7\}$, so that, when values of p and q are greater and equal than 4, the p and q need to be recomputed as p=p-4,q=q-4 and the starting block will be the first block where i=0, otherwise p=p-4,q=q-4 and the starting block will jump to next one where i=1. The distance between $Z'[i,j]$ and $Z''[i,j]$ will keep the constant value = 4.

With the improvement of D21, we generate accuracy analysis as Figure 5.7 to see the performance of old algorithm D21 and new D21. The green line denoted as *alternative 2* stands for improved D21 and blue line denoted *alternative 1* stands for old D21 from the literature. Comparing those two different algorithms D21, the performance of improved D21 is much better than old D21 mentioned by paper. And there is less error occurred when detecting cropped and compressed image with improved D21.

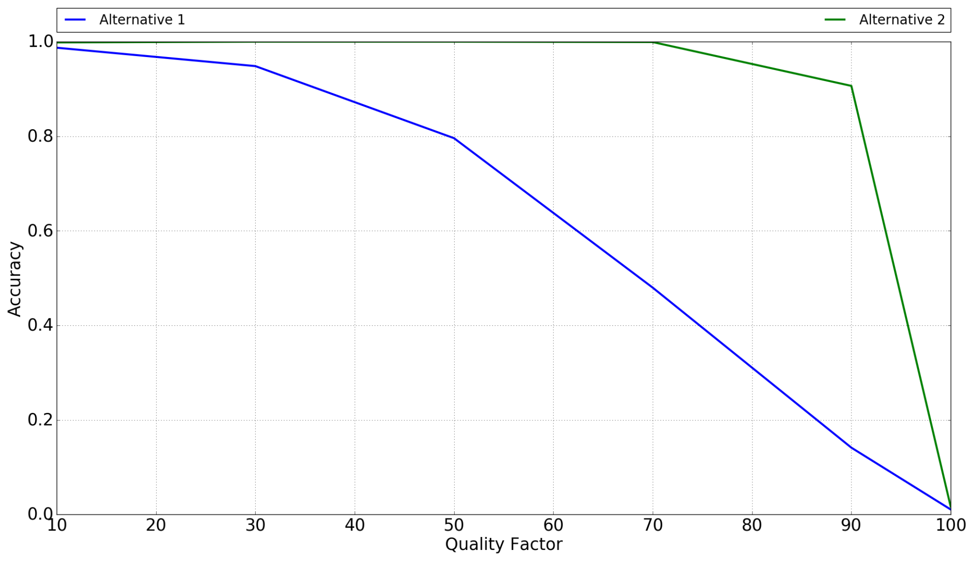


Figure 5.7: Accuracy of old D21 and improved D21 (alternative 1 is old D21, and alternative 2 is improved D21).

However, even if the performance of improved algorithm for detecting image compression has been enhanced, the improved algorithm (improved D21) still has some errors when detecting cropped and compressed images.

Chapter 6

Conclusion and Future Works

In this thesis, we faced the problem of JPEG compression detection, i.e., detecting whether an image has been JPEG compressed. We have provided details about the background and mechanism of JPEG compression and JPEG forensics. As far as JPEG forensics is concerned, first, we have reviewed some state-of-the-art JPEG compression detection methods. Then we have explained what is the main idea behind the many methods and algorithms for JPEG double compression detection, JPEG multiple compression detection and tampering localization.

After background study, we have analyzed a set of representative methods and algorithms for JPEG compression detection in detail [10, 14, 6]. First of all, we have analyzed an algorithm based on the assumption we know in advance a possible JPEG grid alignment. Then, we removed the made assumption and focus on a detector able to estimate the possible JPEG grid position. In this situation, we proposed a modified version of such detectors that outperforms the state of the art one. Finally, we focused on an algorithm that does not need to estimate grid position at all. Also in this situation, we proposed a few ways to deal with the multiple outputs produce by the method, and merge them into a single JPEG compression detection estimation.

After the theoretical study, we have implemented the proposed algorithms. Our results show that the method assuming knowledge about JPEG grid [10, 14], typically outperforms its counterpart [6] only if the grid has been correctly estimated. However the algorithm in [6] can detect not only whether the image has been compressed, but also show which part of image has been highly compressed or less compressed. Concerning the algorithm for estimating JPEG grid location, the version proposed in the literature does not provide sufficient accuracy, whereas the proposed modification greatly improves its results. Finally, we built a GUI to enable non-expert forensic investigators to use these methods and algorithms for image analysis.

As far as future work is concerned, we plan to follow two separate research liens. On one hand, we believe that the proposed analysis can be extended to the case of multiple JPEG compression detection, which is a much more challenging scenario. As a matter of fact, multiple compression may also involve the presence of multiple misaligned JPEG grids. On the other hand, we would like to further investigate the possibility offered by [6] to localize traces of JPEG compression to specific area of an image. This could be particularly useful to expose images obtained through copy and paste of many pictures.

Bibliography

[1] D. Marshall, "Jpeg compression," 2001. Available at `https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node234.html`.

[2] P. Steven W. Smith, "The scientist and engineer's guide to digital signal processing," 1997. Available at `http://www.dspguide.com/ch27/6.htm`.

[3] A. Piva, "An overview on image forensics," *ISRN Signal Processing*, vol. 2013, pp. 1–22, 2013.

[4] D. Fu, Y. Q. Shi, and W. Su, "A generalized benford's law for jpeg coefficients and its applications in image forensics," in *SPIE Conference on Security, Steganography, and Watermarking of Multimedia Contents* (E. J. Delp and P. W. Wong, eds.), vol. 6505, 2007.

[5] J. Lukas and J. Fridrich, "Estimation of primary quantization matrix in double compressed jpeg images," in *Digital Forensic Research Workshop*, Aug. 2003.

[6] W. Luo, Z. Qu, J. Huang, and G. Qiu, "A novel method for detecting cropped and recompressed image block," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 2, pp. II–217–II–220, April 2007.

[7] C. Pasquini, G. Boato, and F. Perez-Gonzalez, "Multiple jpeg compression detection by means of benford-fourier coefficients," in *2014 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 113–118, Dec 2014.

[8] M. Barni, A. Costanzo, and L. Sabatini, "Identification of cut and paste tampering by means of double-jpeg detection and image segmentation," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 1687–1690, May 2010.

[9] T. Bianchi and A. Piva, "Image forgery localization via block-grained analysis of jpeg artifacts," *IEEE Transactions on Information Forensics and Security*, vol. 7, pp. 1003–1017, June 2012.

[10] Z. Fan and R. de Queiroz, "Maximum likelihood estimation of jpeg quantization table in the identification of bitmap compression history," in *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, vol. 1, pp. 948–951 vol.1, 2000.

[11] T. Point, "Tutorials point-learn dip-sobel operator," 2017. Available at `https://www.tutorialspoint.com/dip/sobel_operator.htm`.

[12] K. T. Tan and M. Ghanbari, "Measuring blocking artefacts using harmonic analysis," *Electronics Letters*, vol. 35, pp. 1322–1323, Aug 1999.

[13] H. Farid, "Digital doctoring: can we trust photographs?," in *Deception: From Ancient Empires to Internet Dating*, Stanford University Press, 2009.

[14] Z. Fan and R. L. de Queiroz, "Identification of bitmap compression history: Jpeg detection and quantizer estimation," *IEEE Transactions on Image Processing*, vol. 12, pp. 230–235, Feb 2003.

[15] T. Bianchi and A. Piva, "Detection of nonaligned double jpeg compression based on integer periodicity maps," *IEEE Transactions on Information Forensics and Security*, vol. 7, pp. 842–848, April 2012.

[16] A. Bovik, "Introduction to image compression," in *Handbook of Image and Video Processing (Second Edition)*, Communications, Networking and Multimedia, pp. 641 –, Burlington: Academic Press, second edition ed., 2005.

[17] Z. Wang, A. C. Bovik, and B. L. Evan, "Blind measurement of blocking artifacts in images," in *Proceedings 2000 International Conference on Image Processing (Cat. No.00CH37101)*, vol. 3, pp. 981–984 vol.3, 2000.

[18] H. Liu and I. Heynderickx, "A no-reference perceptual blockiness metric," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 865–868, March 2008.

[19] S. Tjoa, W. S. Lin, H. V. Zhao, and K. J. R. Liu, "Block size forensic analysis in digital images," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, vol. 1, pp. I–633–I–636, April 2007.

[20] B. Li, Y. Q. Shi, and J. Huang, "Detecting doubly compressed jpeg images by using mode based first digit features," in *2008 IEEE 10th Workshop on Multimedia Signal Processing*, pp. 730–735, Oct 2008.

[21] J. M. Campanario and M. A. Coslado, "Benford's law and citations, articles and impact factors of scientific journals," *Scientometrics*, vol. 88, no. 2, pp. 421–432, 2011.

[22] X. Feng and G. Doërr, "Jpeg recompression detection," in *SPIE Conference on Media Forensics and Security*, 2010.

[23] Z. Qu, W. Luo, and J. Huang, "A convolutive mixing model for shifted double jpeg compression with application to passive image authentication," in *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1661–1664, March 2008.

[24] G. Schaefer and M. Stich, "Ucid - an uncompressed colour image database," 2003.