IPOL
IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2021–12–16.
Submitted on 2021–11–29, accepted on 2021–12–10.
ISSN 2105–1232 © 2021 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2021.390

ZERO: a Local JPEG Grid Origin Detector Based on the Number of DCT Zeros and its Applications in Image Forensics

Tina Nikoukhah, Jérémy Anger, Miguel Colom, Jean-Michel Morel, Rafael Grompone von Gioi

Universit Paris-Saclay, ENS Paris-Saclay, CNRS, Centre Borelli, F-91190 Gif-sur-Yvette, France
{tina.nikoukhah, jeremy.anger, colom-barco, morel, grompone}@ens-paris-saclay.fr

Communicated by Jose-Luis Lisani *Demo edited by* Jérémy Anger and Tina Nikoukhah

Abstract

This work describes a method for detecting JPEG compression as well as its grid origin. The JPEG algorithm performs a quantization of the DCT coefficients of non-overlapping 8×8 blocks of images, setting many of those coefficients to zero. The method described here exploits these facts and identifies the presence of a JPEG grid when a significant number of DCT zeros is observed for a given grid origin. This method can be applied globally to identify a JPEG compression, and also locally to identify image forgeries when misaligned or missing JPEG grids are found. The algorithm includes a statistical validation step according to Desolneux, Moisan and Morel's a contrario theory, which associates a number of false alarms (NFA) with each tampering detection. Detections are obtained by a threshold of the NFA, which renders the method fully automatic and endows it with a false alarm control mechanism.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the `README.txt` file of the archive.

Keywords: JPEG compression; DCT coefficients analysis; a contrario method; forgery detection

[1] https://doi.org/10.5201/ipol.2021.390

1 Introduction

Along the image formation pipeline of a camera, the raw data from the sensor undergoes a series of operations: denoising, demosaicing, white balance, gamma correction, compression, to mention a few [7]. These operations create artifacts in the final image, often imperceptible to the naked eye but nevertheless statistically significant and therefore detectable. The detection and interpretation of these traces make it possible to reconstruct, to some extent, the history of the image; in other words, to know the operations that took place during the creation of the image, as well as their order and parameters. Anomalies in these traces may indicate the presence of image forgeries.

This work describes a method called ZERO, which aims at detecting whether an image has undergone a JPEG compression during its history. A statistical test, based on Desolneux, Moisan and Morel's *a contrario* theory [8], is used to decide when a significant JPEG grid is found while controlling the number of false detections. The method described here derives from the one in [16] but includes some improvements such as the fact that it can detect more types of forgeries.

When the image to be analyzed is encoded in a JPEG file, there is no need to detect the JPEG grid as all the relevant information is contained in the header of the JPEG file itself. Nevertheless, even in that case, it may be interesting to detect traces of a previous JPEG compression. More importantly, in forensic applications one wants to analyze an image in any format and study whether one or more JPEG compressions were applied. For these reasons, the method described here takes as input only the pixel values of the image and uses no metadata information that may be contained in the image file.

The method described here can be applied globally to the whole image to detect JPEG compression. The grid detected globally is considered the main grid. The method can also be applied locally to parts of the image. When a JPEG grid is detected in a part of the image and its grid origin is different from the main JPEG grid, it is considered as an anomaly and thus as a forgery. Another kind of anomaly is when the JPEG traces are missing in a part of an image where they should be present. The proposed algorithm provides two binary forgery masks: one showing foreign grid areas and the other showing missing grid areas. The statistical test leads to some theoretical guarantees and provides secure results for tampering detection. The aim is to obtain very few false detections.

The rest of this work is organized as follows. Section 2 provides a brief summary of the JPEG algorithm. Then, the JPEG grid detection method is described in Section 3; the method is composed of two steps: the voting process and the statistical validation. Section 4 explains how to apply the basic algorithm in three different ways. First, on the whole image to determine the global grids; having multiple grids may mean that the image has been manipulated. Second, on small patches of the image, chosen by a region growing heuristic; the aim is to detect foreign grids and therefore forgeries. Finally, the method can also be used after a specific pre-processing that allows to detect missing grid areas. A detailed description of the full algorithm is given in Section 5 while its computational complexity is studied in Section 6. Several experiments are then analyzed in Section 7 and the limitations are studied in Section 8. Finally, Section 9 concludes the paper.

2 JPEG Compression

The JPEG algorithm (ISO/IEC 10918 — ITU-T Recommendation T.81 [1]) is currently the most common method for compression of digital photography. The encoding process, shown in Figure 1, is detailed in the following. The first step is to perform a color space transformation from RGB to YC_BC_R, where Y is the luminance component and C_B and C_R are the chrominance components of the blue and red difference. The chroma channels are optionally downsampled in the JPEG algorithm; the most common options are using full chroma resolution (noted 4:4:4), downsampling by a factor

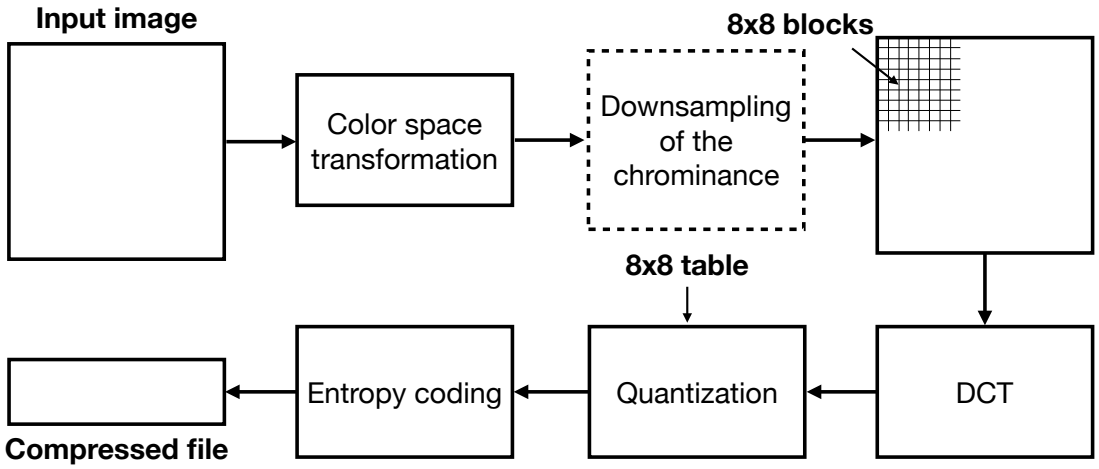


Figure 1: The JPEG compression pipeline.

of 2 only in the horizontal direction (noted 4:2:2), or reduction by a factor of 2 in both the horizontal and vertical directions (noted 4:2:0).

The value 127 is subtracted from each pixel to obtain values with a distribution that is roughly centered around zero[2]. Then, each channel is divided into non-overlapping 8×8 blocks and each block is processed independently. The type II 2D Discrete Cosine Transform (DCT) is applied to each block and the coefficients are quantized according to a given table (Figure 2 shows an example). This quantization table provides a factor for each DCT component and determines the compression level; the larger the factors, the lower the resulting file size, but also the lower the image quality. The quantization table is associated with a compression quality QF, an integer value from 1 to 100; the worst quality corresponds to $QF = 1$ and the best quality corresponds to $QF = 100$. DCT coefficients are put to zero when their value is smaller than the quantization factor. Hence, each 8×8 block gets a number of zeroed DCT coefficients, that depends both on the compression quality and on the image content.

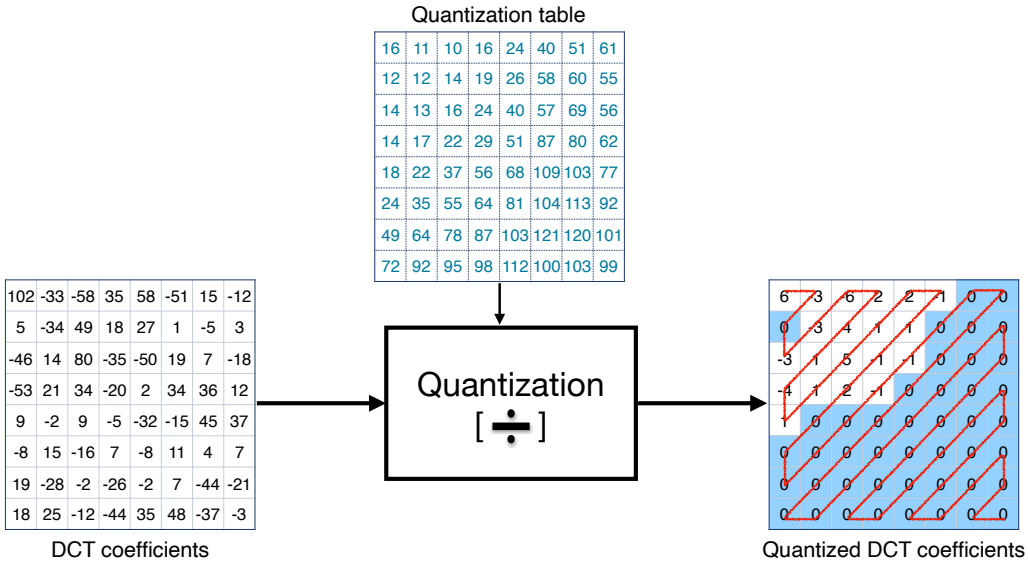


Figure 2: The impact of quantization on an example DCT block. Each DCT coefficient is quantized by the associated value in the quantization matrix (here associated to $QF = 50$). Rounding leads to setting to zero many of the high frequency coefficients. Each block is scanned in a zig-zag pattern to be encoded as a vector ending with a sequence of zeros.

Finally, the quantized DCT coefficients are losslessly compressed by exploiting, among other

[2]As we will see later, the method ZERO only looks at the AC coefficients on which this subtraction has no impact.

things, the presence of zero values. Indeed, each 8×8 block is scanned in a zig-zag pattern and the coefficients are arranged in the form of a vector in which the first components represent the low frequencies and the last ones the high frequencies. An example is shown in Figure 2. A lossless compression by Run-Length Encoding (RLE) exploits the long series of zeros at the end of each vector. A final lossless compressed file is obtained by Huffman encoding, to which a header is finally added to form the final JPEG file.

The quantization step, which has the most impact on the compression factor and the image quality, leaves traces at the boundaries of each 8×8 block, as shown in Figure 3. These traces, characteristic of JPEG compression, illustrate what we call the grid, depicted in red in the figure. Since the blocks are of size 8×8, there are 64 possible grid origins. In the following, a grid will be characterized by its origin's coordinates g_x and g_y. If a JPEG image has not been further processed after decompression, the grid's origin should be $(0, 0)$.

As described above, the JPEG algorithm sets to zero some of the DCT coefficients of 8×8 blocks, and the more zeros in the quantized DCT, the smaller the JPEG file size. Based on this fact, the core of the method described here is to count the total number of zeros of each hypothesized DCT block position. In the presence of JPEG compression, this number should be maximum when the 8×8 block is aligned with the JPEG grid. Indeed, non-aligned blocks include additional discontinuities due to blocking artifacts, leading to larger DCT coefficients compared to an aligned block. The JPEG block artifacts can be clearly seen when the image has been strongly compressed, i.e. with large values in the quantization table, and are almost imperceptible when the compression quality is high. However, a grid is always present in lossy compression ($QF \leq 99$) and the method ZERO should be capable to detect it.

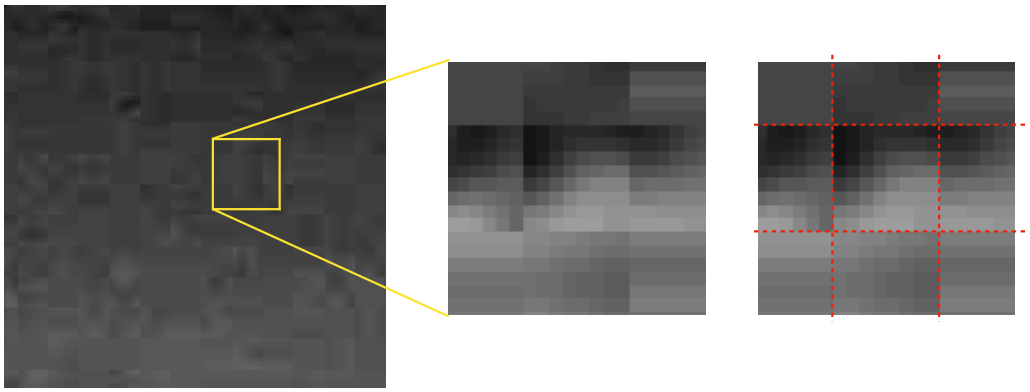


Figure 3: JPEG block artifacts. The red dotted lines highlight the boundaries of the 8×8 blocks used in the compression.

3 The JPEG Grid Detection Method

This section describes the main component of ZERO, namely the algorithm to identify a JPEG grid and its position based on the number of zeros in the DCT of blocks.

3.1 Luminance Computation

The algorithm ZERO takes an RGB image and focuses on the luminance channel computed according to the JPEG standard

$$Y = \text{round}\Big(0.299\ R + 0.587\ G + 0.114\ B\Big),$$

where R, G and B are the red, green and blue channels. The grid detection method could be adapted to use the chroma components Cb and Cr. This requires knowing the chroma downsampling factors. Alternatively, all the possible downsampling factors could be tested and the right one validated by a statistical test similar to the one described in this work. But this would imply considering a number of different cases, making the algorithm more complex. For the sake of simplicity, and because the algorithm is already reasonably sensitive, the chroma components are not used by the method described here. Nevertheless, exploiting the information of the chroma components should extend a little the detectability of forgeries; this will be the focus of future work.

3.2 Identifying DCT Zeros

A crucial point is how to determine which DCT coefficient were zeros, which is far from being a trivial task. Indeed, DCT coefficients, which were set to zero during compression, usually don't keep an exact zero value after decompression. During JPEG decompression, an inverse DCT transform is performed on each block, transforming the integer DCT coefficients into pixel values that are *real numbers*. Then, those real numbers are rounded to produce an integer image. Different mathematical operators (e.g. *floor*, *ceil* or *round*) can be used to convert the pixel values from floating-point to integer values [2]. This uncompressed, integer image is the input to the present method (in the case that the image was indeed JPEG compressed).

This rounding step, which results in an integer image, also modifies the corresponding DCT values. It can be shown that this step is statistically equivalent to the addition of a Gaussian noise (with standard deviation equal to $\frac{1}{12}$) to the initial integer DCT values [21]. Thus, a DCT coefficient that was put to zero during compression, does not keep an exact zero value after decompression. Yet, it remains close to zero. We propose to count the number of coefficients with absolute values smaller than 0.5. This allows one to discriminate zeros even when the DCT coefficient quantization is at its finest rate, with a quantization factor of one.

3.3 Grid Origin Vote Map

Each pixel may belong to 64 different overlapping 8×8 blocks, as illustrated in Figure 4. To compute the JPEG grid origin vote map, those 64 blocks are evaluated for each pixel. The 64 DCTs of those blocks are computed as well as the corresponding number of zeros. Then, each pixel votes for the grid origin of the block with most zeros. In the case of a tie, the pixel does not vote. There is relevant information when two or more blocks have the same number of zeros. However, exploiting this information would make the algorithm more complex. Again, given that the method is already reasonably sensitive, we preferred to keep a simple formulation. Exploiting such information is possible and would be the focus of future work.

Performing the count as described, which means evaluating the 64 blocks for each pixel, requires computing 64 DCTs per pixel; but this is wasteful as every block is shared with 64 other pixels and this can be exploited to avoid recomputing the DCT. A more efficient procedure will be described in Section 5.1.

In order to make a fair vote, a pixel must actually belong to 64 blocks included inside the image domain. This is not true for the pixels within a 7 pixel wide region around the image border, or for the pixels on the lower and right border of the image, where JPEG blocks are incomplete if the image size is not a multiple of 8. In those cases, the pixels could vote for a grid origin different from the actual one, just because there is no 8×8 block in the image aligned with the grid and containing them. When the image has been manipulated, the same problem can appear on any border. Thus, the positions within 7 pixels from the border are prevented from voting.

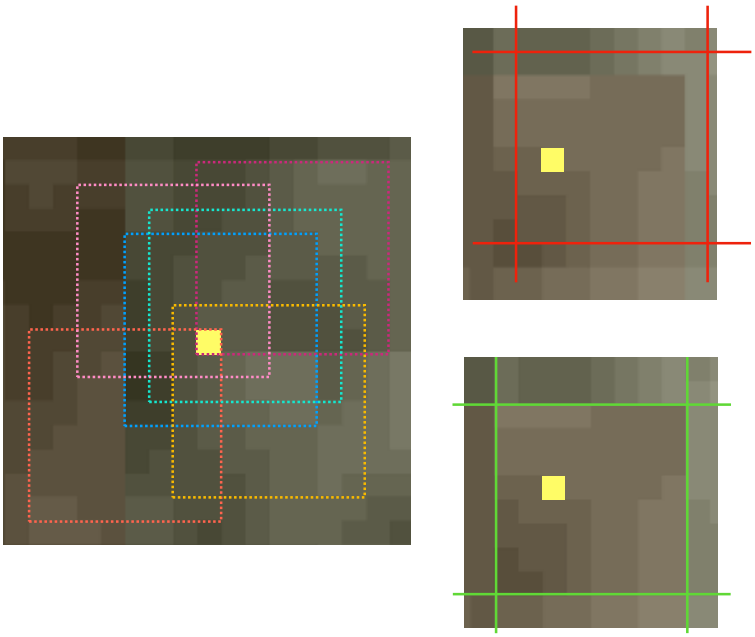


Figure 4: Each pixel (yellow) belongs to 64 different 8×8 blocks of the image. Six of them were drawn in different colors on the left. Top right shows (in red) the position of a patch not aligned with the grid. Bottom right shows (in green) the position of the patch containing the pixel matching the JPEG grid.

Figure 5: Left: Uncompressed image and its vote map. Right: JPEG compressed image and its vote map. In both cases, the pixels which return a NON VALID vote (a tie, border or a third reason that is explained in Section 3.5) are shown in black.

Figure 5 shows two vote map examples. Each color is assigned to one of the 64 possible grid origin votes and black to non-valid votes. The vote map on the left corresponds to an uncompressed image;

we observe a random vote map. The vote map on the right corresponds to a JPEG compressed image. The black areas correspond to the pixels which did not vote because of a tie, because they are on the border, or for another reason explained in Section 3.5. Blocks in flat image regions can have the same maximal number of zeros, resulting in ties; this happens (e.g.) in the saturated parts of the image.

3.4 Statistical Validation

When analyzing a JPEG image, the most voted grid probably corresponds to the right one. But the most voted grid origin does not necessary correspond to a JPEG grid actually present. Indeed, even in uncompressed images, one of the grids will get more votes than the others, usually by a small margin. A statistical criterion is therefore needed to decide whether this prominence is caused by JPEG compression or not.

The validation procedure proposed here is based on the *a contrario* theory [8], which relies on the *non-accidentalness principle* [14, 22]. Informally, this principle states that there should be no detection in noise. In the words of D. Lowe, "*we need to determine the probability that each relation in the image could have arisen by accident, $P(a)$. Naturally, the smaller that this value is, the more likely the relation is to have a causal interpretation*" [14, p. 39]. This principle has shown its practical use for detection purposes such as line segment detection [10], vanishing points detection [13], anomaly detection [6], or forgery detection [3, 18].

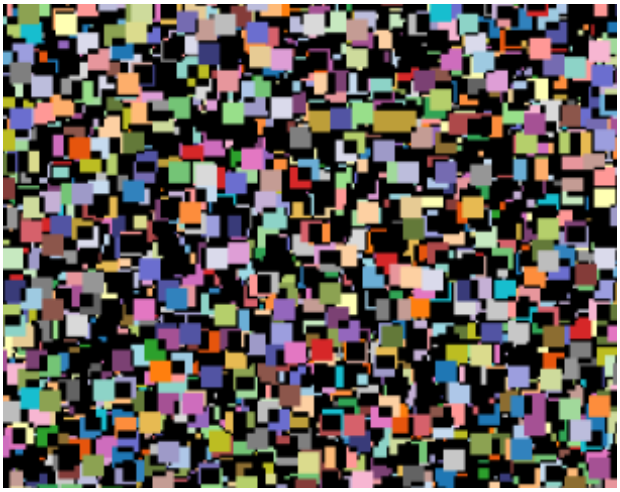


Figure 6: Zoom on a vote map for an image of Gaussian noise. Each color represents a vote for a given grid origin. The black color corresponds to NON-VALID votes. One can observe entire blocks of 8×8 pixels voting for the same origin; this is the case when a block has a local maximum of number of zeros.

In our context, we need to assess the probability that a given grid origin gets a large number of votes purely by chance. To that aim, a stochastic null model H_0 for the votes is required. It is here easily given by Laplace's principle of indifference: in absence of JPEG compression, each of the 64 blocks containing a given pixel would have the same chance of being the one with the largest number of zeros; that would depend on the image content and there is no reason to suppose that it is synchronized with a particular 8×8 grid origin. However, the votes of neighbor pixels are not independent, even in noise images. Indeed, there are always blocks that are local maxima of the number of zeros, and those blocks get the votes of every pixel belonging to it. Figure 6 shows a vote map obtained in an image of noise where one can observe entire blocks of 8×8 pixels voting for the same origin, the result of local maxima. This implies that votes are correlated within a distance of 8 pixels; on the other hand, pixels at distance larger than eight are largely uncorrelated. Thus,

we define a stochastic null model H_0 for votes at distance eight in which votes are independent and uniformly distributed among all the 64 grid origins.

Let us suppose that we are observing a patch of an image where the number of votes for a given valid grid origin is counted at a distance of eight pixels. Let us say that k votes are counted for that valid grid among a total of n votes. Under the null hypothesis H_0, votes for the given grid origin become Bernoulli random variables with probability $\frac{1}{64}$. So under H_0, the number of votes becomes a random variable K and, given the independence of votes (at distance larger than eight), it follows a binomial distribution of parameter $p = \frac{1}{64}$. Thus,

$$\mathbb{P}(K \geq k) = \mathcal{B}(n, k, p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j},$$

where $\mathcal{B}(n, k, p)$ is the tail of the binomial distribution. Given an observed number of votes k, $\mathbb{P}(K \geq k)$ is the probability of obtaining at least k votes under H_0. When this probability is small enough, there exists evidence to reject the null hypothesis and declare that a meaningful grid origin was found.

However, the multiplicity of tests needs to be taken into account when considering that the probability is small enough. To use an analogy, even if the odds of each individual lottery ticket is 1/1000, the chances of winning are very high when buying 1000 tickets. Similarly, if 1000 tests were performed, it would not be surprising to observe an event that appears with probability 1/1000 under random conditions. The number of tests N_T needs to be included as a correction factor, as it is standard in statistical multiple hypothesis testing [9]. The null hypothesis H_0 is rejected when $\mathbb{P}(K \geq k) < \frac{\varepsilon}{N_T}$ for a predefined value ε. This is called the Bonferroni correction.

Following the *a contrario* methodology, we define the Number of False Alarms (NFA) of a candidate grid g on a given window w as

$$\mathrm{NFA}(g, w) = N_T \mathbb{P}(K \geq k). \tag{1}$$

This is equivalent to what was just stated, the null hypothesis H_0 is rejected when $\mathrm{NFA}(g, w) < \varepsilon$. It can be shown [8] that under the null hypothesis H_0 the expected number of false alarms with $\mathrm{NFA}(g, w) < \varepsilon$, is bounded by ε

$$\mathbb{E}_{H_0}\left[\sum_{(g,w)\in\mathcal{N}_T} \mathbb{1}_{\mathrm{NFA}(g,w)<\varepsilon}\right] < \varepsilon, \tag{2}$$

where $\mathcal{N}_T$ is the set of N_T tests. As a result, ε is a (tight) upper bound to the mean number of false detections per image under H_0. In most practical applications, the value $\varepsilon = 1$ is suitable; we will set it once and for all in our application as well. With this choice, the expected number of false grid detections per image under H_0 is guaranteed to be upper-bounded by 1. (Notice that this is the expected number of false *JPEG grid* detections, not the expected number of false *forgery detections*, for which one false detection per image would not be an acceptable rate.)

The validation procedure will be used to evaluate the JPEG grid on the whole image as well as on local windows, thus enabling local forgery detection. To that aim, every window of a $X \times Y$ pixels image is included in the family of tests. Also, the 64 grid origins are tested on each window. Finally, as was mentioned, the vote map is sampled on a grid with 8×8 cells. A difficulty is that there are 64 possible such grids, all of which should be evaluated. It follows that the number of tests is approximately

$$N_T = XY \cdot XY \cdot 64 \cdot 64, \tag{3}$$

because there are XY possible positions for the upper-left corner and XY possible positions for the lower-right corner of the window. The two 64 terms correspond to all possible JPEG grid origins and to all possible grids to subsample the vote map. All in all, given a window to be analyzed, the grid origin with the maximum of votes is selected and its number of votes at distance eight pixels is counted. Then, the NFA is given by

$$\mathrm{NFA}(g, w) = 64^2 \cdot (XY)^2 \cdot \mathcal{B}\left(n, k, \frac{1}{64}\right), \tag{4}$$

where k is the number of votes in the window w (at distance 8) for the grid g among a total of n points inside w (again at distance 8). A JPEG grid is detected when $\mathrm{NFA} < 1$.

In addition to be used to decide whether a JPEG grid is present or not, the NFA also allows to compare two possible grid interpretations. In some cases, two or more grid origins could be meaningful ($\mathrm{NFA} < 1$) on the same window. The lower the NFA, the more surprising the observation is under the stochastic null model H_0. Thus, the lower the NFA, the more meaningful the detection. When two or more grids are detected for the same region, the one with the lower NFA will be selected as the main one.

In principle, votes must be counted at distance eight in both directions. This corresponds to the votes with coordinates $(x_0 + 8i, y_0 + 8j)$ for integers i and j. For a given window, this test must be performed for all other 64 grids with x_0 and y_0 in $\{0, 1, \ldots, 7\}$. A simpler way is to evaluate a lower bound for the number of votes in the most voted grid; this is performed as follows. Instead of counting votes at distance of eight pixels for those offsets, we can count every vote and divide the number by 64. Indeed, let v be the total of votes in the window for the given grid. If those votes were equally distributed on the eight-distance subsamplings, one would have $k = \frac{v}{64}$ for each of the subsamplings. If not, necessarily one of the subsamplings will have more votes. Hence we can deduce that there is at least one of those subsamplings with k votes satisfying $k \geq \frac{v}{64}$. So by counting every vote and dividing the count by 64 we are considering the worst case and we are sure that a detected grid is meaningful. Naturally, the count of votes for every pixel in the window is also divided by 64. The NFA is evaluated then by

$$\mathrm{NFA}(g, w) \approx 64^2 \cdot (XY)^2 \cdot \mathcal{B}\left(\frac{|w|}{64}, \frac{v}{64}, \frac{1}{64}\right), \tag{5}$$

where $|w|$ is the total number of pixels in the window w and v is the total number of votes for g in w. Algorithm 3 describes the ensuing JPEG grid detection method.

3.5 Blocks that are Constant along the Vertical or Horizontal Direction

The picture in Figure 7 was taken with the portrait mode of a smartphone. The background of the scene is made blurrier to mimic the *bokeh* effect. When compressed heavily, the blurry area is quantized, and a large number of the 8×8 blocks that were almost flat, *become* flat. Indeed, most DCT coefficients have small values in almost flat blocks. A strong quantization will put them all to zero, resulting in flat blocks. Because the DCT is separable, a similar effect is observed in blocks presenting a soft gradient in a direction roughly vertical or horizontal: in this case, the coefficients in one direction have almost constant values, and the quantization may make them equal. As a result, blocks that are almost constant in the vertical or horizontal direction, will become exactly constant in the vertical or horizontal direction. This phenomenon can be seen in Figure 8. In other words, a strong JPEG compression results in a large number of vertically or horizontally constant blocks.

Regions of images that are constant along the vertical or horizontal direction are problematic for the proposed statistical validation. Indeed, for a given pixel, instead of having 64 blocks to

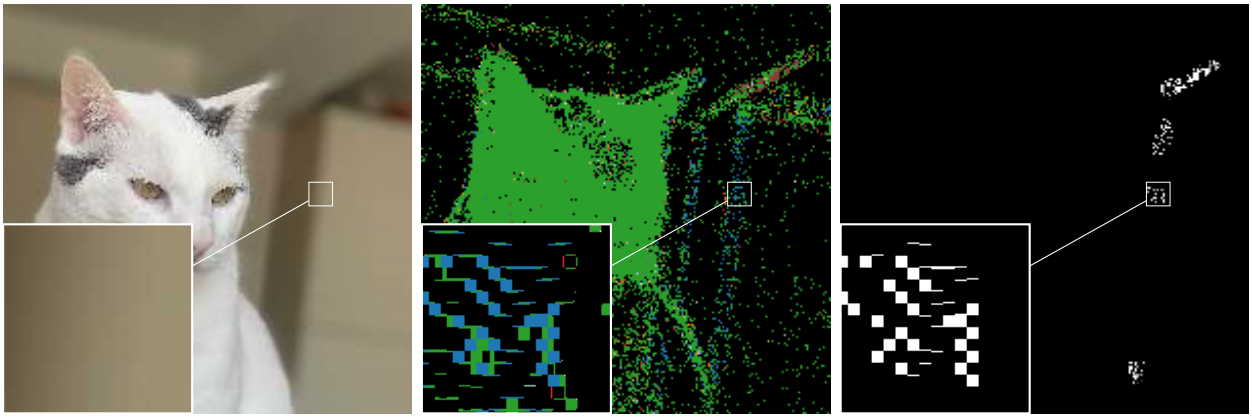


Figure 7: Left: Image compressed with quality 50. The image was taken with the portrait mode which creates this *bokeh* effect. Middle: vote map without the constant fix, i.e., when counting blocks that are constant along the vertical or horizontal direction. Right: forgeries detected when not using the constant fix; these are false detections.

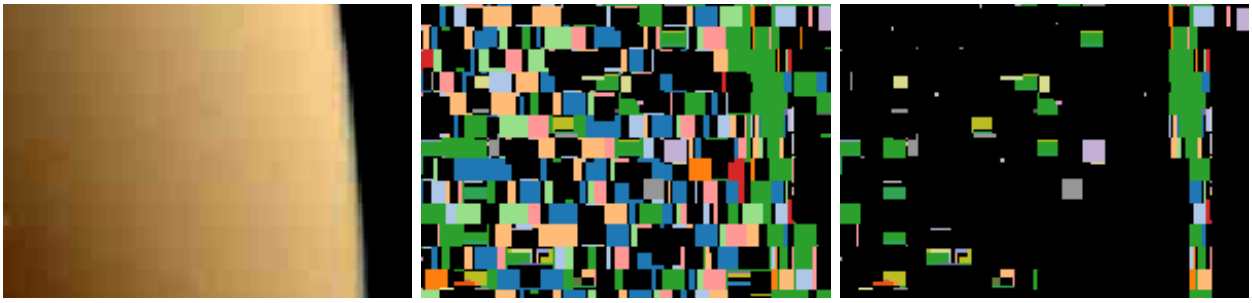


Figure 8: Zoom on a constant region along the vertical direction. Middle: vote map when taking into account all votes. Right: vote map when discarding the votes of blocks constant along the vertical or horizontal direction.

compare the votes, only 8 of them are different. Thus, the probability of voting for a particular grid by chance is actually $\frac{1}{8}$ and not $\frac{1}{64}$. The statistical test could be misled to consider as meaningful configurations that are not. Under random conditions, this would be an extremely rare accident; however, the tendency of JPEG to create blocks that are constant along the vertical or horizontal direction requires handling this situation.

In many cases, regions that are constant along the vertical or horizontal direction extend over several blocks in both directions. Then, there would be a tie in the voting process and corresponding pixels will not vote. Nevertheless, there are cases in which there are several blocks in a row that are constant along the vertical direction, but only of one block height. Then, the correct vertical origin will get the best vote. But in the horizontal direction, pixels could vote for any grid with probability $\frac{1}{8}$ instead of $\frac{1}{64}$, undermining the statistical model. An analogous case appears interchanging vertical and horizontal directions.

A more sophisticated statistical model could handle this case. But a simpler solution is to prevent blocks that are constant along the vertical or horizontal direction from voting. In such cases the pixel gives a non valid vote, as in the case of ties. The result can be seen in Figure 8. This simple solution is nevertheless discarding useful information; a refined solution might be the focus of future work.

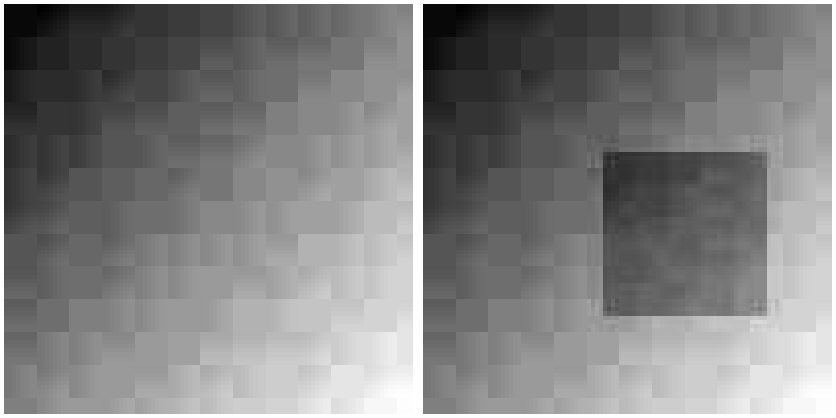


Figure 9: Copy-move impact on a strongly compressed JPEG image. The left image is authentic and the right image is forged by copy-move. The JPEG grids are apparent. A local shift of a square region is easily spotted on the right image.

4 Application to Image Forensics

4.1 Detection of Global JPEG Grids

A first application of the method is to tell whether an image has undergone JPEG compression or not. JPEG grid detection is the first step of many forgery detection algorithms, and the image compression history an important cue. It can be used to detect a grid origin different from $(0, 0)$, which indicates that the image has been cropped. In image restoration, grid detection is also used to remove grid artifacts by a deblocking procedure [4].

When JPEG compression is lossless, which is obtained by setting the quality factor QF equal to 100, then no DCT coefficient is forced to zero; in such a case the actual grid cannot be discriminated by ZERO and this is a clear limitation of the method. Another one is when the image is too compressed, for instance $QF = 1$; in that case most blocks are constant (vertically and horizontally) as all AC coefficients are put to zero. The compression is not detected by ZERO, because of the decision made in the Section 3.5. Yet, as explained before, a refined version of the method could handle this situation. Such high-compressed images are anyway easily detectable by other methods [17].

Finding the main grid origin yields a reference to which local grid detections can be compared to detect forgeries. The image may actually have several meaningful global grids. This may be caused by several reasons, all suspicious (double compression, resampling, etc.). In that situation, the grid having the most votes, therefore being the most meaningful globally, is selected as the main grid.

4.2 Detection of Local Foreign JPEG Grids

The proposed JPEG grid detection method can be performed globally but also locally on every image window. Any region with a meaningful grid that is different from the main one hints at a forgery. Indeed, when part of a JPEG image is copied and pasted, it retains its grid traces. In 63 out of 64 times (assuming that the forger did not explicitly align the grid), the grid origin will not correspond to the main one, thus allowing its detection. This is true whether it is a case of copy-move from the same image (see Figure 9) or when the copied part is taken from a different JPEG image. It can also happen that the method fails to detect a global grid, but still finds local areas with meaningful JPEG grids; this generally indicates the presence of a forgery.

The same algorithm as described in Section 3 can be applied directly on every window of the image. But this would be computationally expensive. Instead, we propose a heuristic using a greedy algorithm to accelerate the search for forged regions; the final validation still uses the same statistical test used for the global grid. What follows is a quick overview. The detailed description of the full

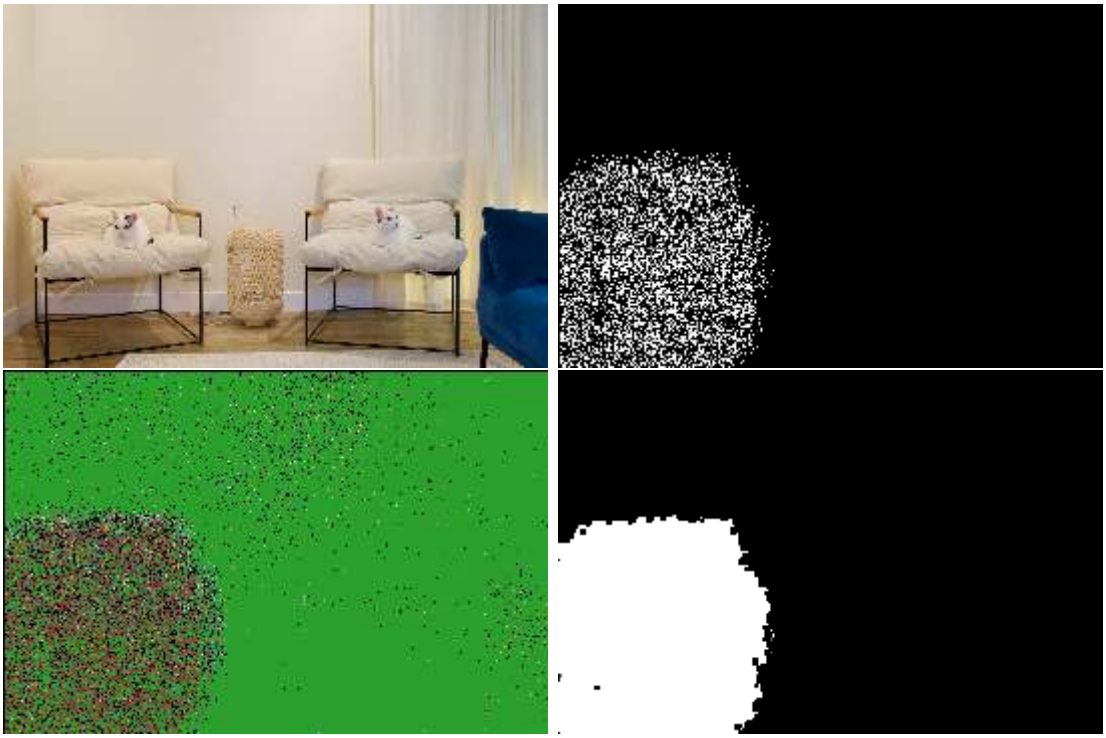


Figure 10: Up-left: a tampered image. Down-left grid origin vote map. Up-right: raw forgery mask. Down-right: final forgery mask after morphological closing.

heuristic procedure will be given in Section 5.3.

The proposed heuristic uses a region growing procedure to partition the vote map into connected regions sharing the same grid vote. Starting from a seed pixel (x, y), the neighboring pixels voting for the same grid are incrementally aggregated. As Figure 10 shows, votes for the same grid can be disconnected, so a relaxed notion of neighborhood is needed. We observe that a window showing a meaningful grid origin necessarily has a vote density for this grid origin larger than $\frac{1}{64}$. Thus, votes for the right grid should not stand farther away than eight pixels, on average. To allow for some local variation in the distribution, we set this neighborhood size a little larger and use $W = 9$. This value was chosen experimentally, the goal being to avoid false positives.

For each connected region endowed with a valid grid origin different from the main one, a bounding box is then computed and the NFA statistical test is performed. If the statistical test confirms that a foreign grid is indeed present, the pixels in the connected region (which all voted for the same grid) are marked in a forgery mask. Figure 10 shows an example. The forged image was obtained by splicing two successive snapshots taken from the same camera at the same position and with the same JPEG QF. Only the grid is incoherent. The region growing algorithm is naturally greedy: Once a region has been evaluated, its pixels are marked to accelerate the algorithm by preventing them from being used again in other regions.

Due to variations in the number of votes, the raw forgery mask contains holes, see Figure 10 up-right. To give a more useful forgery map, these holes are filled by a mathematical morphology closing operator [20] with a square structuring element of size W (the same as the neighborhood used in the region growing step). The down-right image of Figure 10 shows an example of the final forgery mask.

Forgeries are detected when an area has a meaningful grid origin that is different from the main one. However, there are no direct meaningful results with this approach when the forged area has no JPEG traces. For instance, the forged area may come from another image which was not JPEG compressed, or the JPEG traces may have being lost due to several post-processes such as resampling

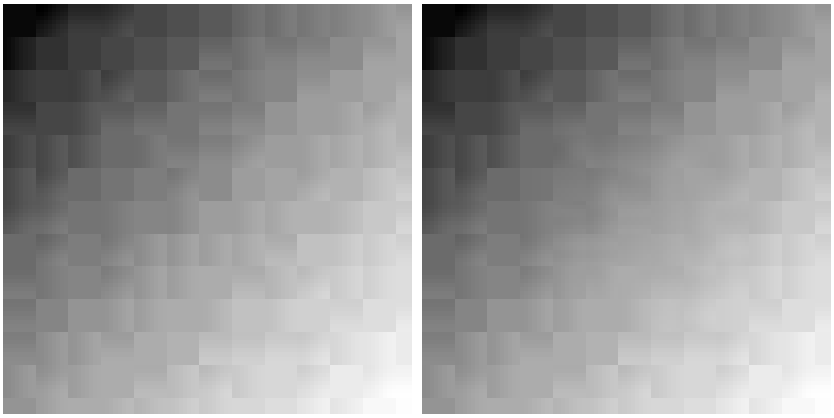


Figure 11: Erasing impact on a strongly compressed JPEG image. The left image is authentic and the right image is forged by erasing an area. The JPEG grids are easy to see and so are the local missing grids on the right image.

or blurring. The next section describes a way to detect this type of forgery.

4.3 Detection of Local Missing JPEG Grids

When an image has a global grid, erasing an area as it is done in Figure 11 – or adding a part from an uncompressed image, or processing the forged area to make it fit – may erase traces of any JPEG grid. If the forged area has no JPEG traces left, the method described in Section 3 cannot detect it directly as anomalous. Indeed, the proposed statistical test detects the *presence* of a JPEG grid, but not its *absence*. Nevertheless, an anomaly is sometimes visible in the vote map, as shown in the middle-left image of Figure 12. We describe here an easy adaptation of the method to cope with this situation.

The idea is to perform a second forgery detection step where the image to be analyzed is JPEG compressed with the best possible quality that is still detectable ($QF = 99$); in that way, a detectable grid of origin $(0, 0)$ is induced in parts where none was present. A second vote map can be computed after this procedure, which is illustrated in the middle-right image of the Figure 12. But only the pixels which did not vote significantly for the main grid in the first forgery detection step (as indicated in the initial vote map) are allowed to vote. The pixels that initially voted for the main grid are instead put to a non valid vote. In the figure, the forged region has now a consistent grid of origin $(0, 0)$ (green) as imposed, while the rest shows NON VALID votes (black). Every region with a meaningful $(0, 0)$ grid on the second vote map corresponds to a region where the JPEG traces were indeed originally missing; they are therefore marked on the second forgery detection map, as shown on the bottom-right of Figure 12.

Why is a second JPEG compression useful? Its goal is to verify whether the missing grid traces are due to the absence of JPEG compression, or just to some peculiarity of the image contents. For example, the initial vote map on Figure 12 shows many pixels not voting for $(0, 0)$; most of them, which appear in black, correspond to saturated areas; others are just too small to lead to a meaningful detection. Figure 13 right shows another example where most pixels of an authentic JPEG image do not vote for $(0, 0)$, due simply to the high-quality JPEG compression ($QF = 99$). When performing the second compression, such regions generally remain undetected, confirming that it was the contents of the image that led to the missing votes. On the other hand, a forged region is confirmed by detecting JPEG traces after the imposed compression, showing that those traces would have been present if the region had been even mildly JPEG compressed. So the absence of traces is indeed an anomaly, probably the result of an image forgery.

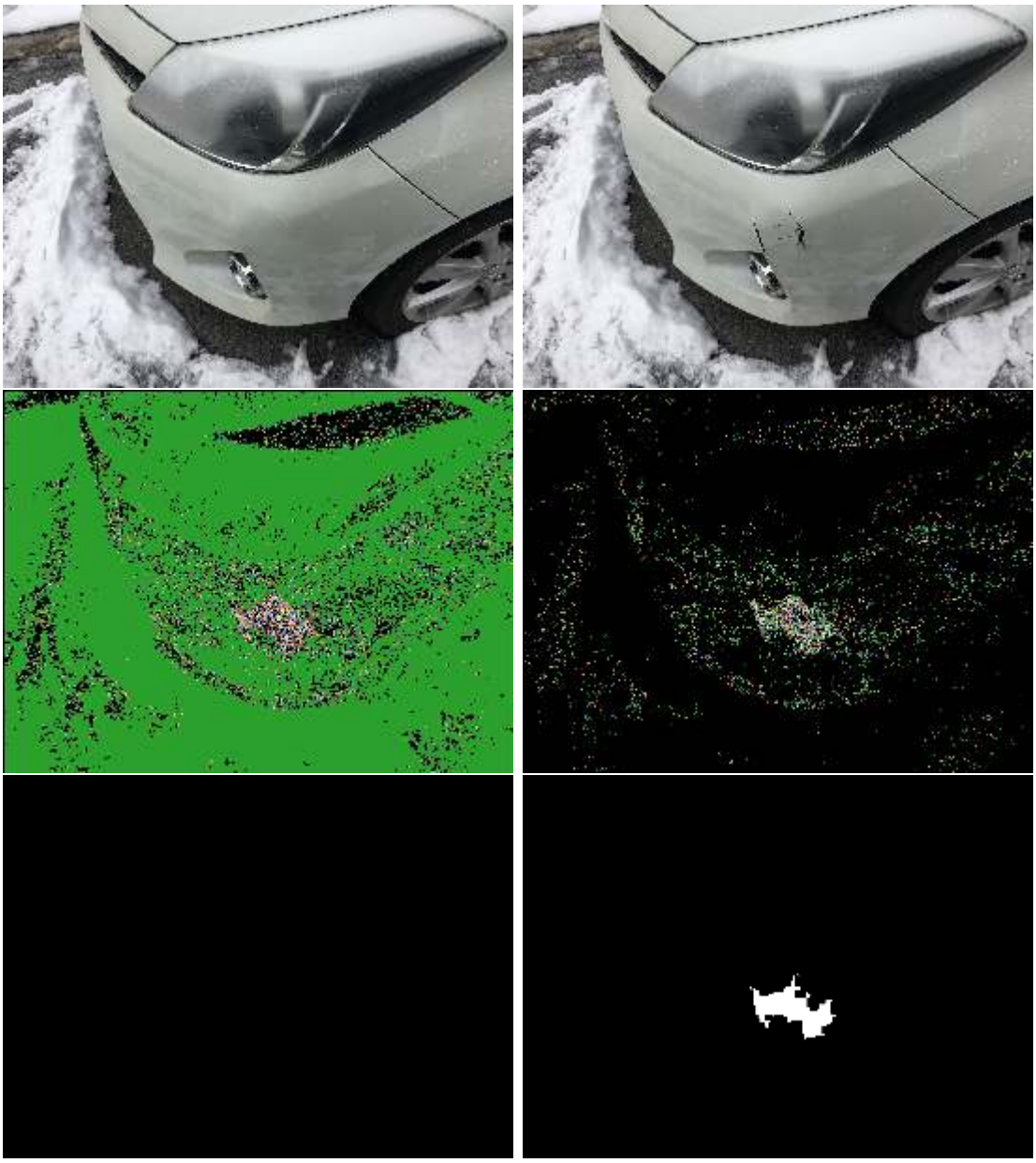


Figure 12: Forged image (top-left) and original image (top-right). The forgery has erased the JPEG traces. The middle-left image is the vote map of the forged image and the middle-right image is the second vote map. The final row shows the initial forgery detection mask (bottom-left) and the forgery detection mask after re-compression (bottom-right).

5 Detailed Description of the Method

Algorithm 1 provides a pseudo-code of the method applied to an image. Each step of this algorithm is described in the following subsections.

Algorithm 1: ZERO

input : Image $I = (R, G, B)$ defined in Ω
output: Main grid Γ
output: Forgery map of foreign grids F
output: Forgery map of missing grids M

1 $Y \leftarrow \text{Round}(0.299\,R + 0.587\,G + 0.114\,B)$ — *compute luminance image*
2 votes $\leftarrow$ ComputeVotes(Y) — *algorithm 2*
3 $\Gamma \leftarrow$ DetectGlobalGrids(votes) — *algorithm 3*
4 $F \leftarrow$ DetectForgeries(votes, Γ) — *algorithm 4 excluding main grid* Γ
5 **if** Γ *is a valid grid* **then** — *look for local areas without a grid*
6 $\quad I' = (R', G', B') \leftarrow \text{JPEGCompression}(I, 99)$ — *compress input image at quality* $QF = 99$
7 $\quad Y' \leftarrow \text{Round}(0.299\,R' + 0.587\,G' + 0.114\,B')$ — *compute luminance image*
8 $\quad$ votes$' \leftarrow$ ComputeVotes(Y') — *algorithm 2*
9 $\quad$ **for** $(x, y) \in \Omega$ **and** votes$(x, y) = \Gamma$ **do**
10 $\quad\quad$ votes$'(x, y) \leftarrow$ NON VALID — *remove pixels that initially voted for the main grid*
11 $\quad M \leftarrow$ DetectForgeries(votes$', [1:63]$) — *algo. 4 excluding grids 1 to 63, use only grid 0* $(0, 0)$
12 **return** Γ, F, M

For the sake of simplify, the reference code does not include an implementation of the JPEG compression algorithm, as would be required to perform step 6 of Algorithm 1; instead, the code takes as a second input the JPEG compressed version of the input image with $QF = 99$. In terms of the pseudo-code, the image I' is expected as a second input, to be computed externally using standard tools (such as the ImageMagick package `https://imagemagick.org`). In the case where this second input is not provided, the analysis of missing grids is not performed and only the F mask is produced.

5.1 Voting Process

Performing the count as described in Section 3.3 requires computing 64 DCTs per pixel. But every block is shared with 64 other pixels and this can be exploited to avoid recomputing the DCT. Algorithm 2 describes the procedure. A table is used to keep track of the largest number of zeros found for each pixel, initially set to zero everywhere. The DCT of every 8×8 block in the image and its number of zeros are computed[3]. Every pixel included in the block is checked and the table of zeros is updated when the current block has more zeros than previously found for that pixel. The table of votes is also updated to the grid origin corresponding to the block with more zeros (GridAlignedWith(b)), or to NON VALID in case of a tie, a constant block or a pixel on the border. In that way, the DCT of each 8×8 block in the image is computed only once. Finally, the votes of all the pixels within a distance 7 from the image border are set to non-valid.

[3]The DCT computation uses the usual float IEEE 754 number representation. Different platforms and even compilers use slightly different numerical approximations which may result in slightly different DCT values. As a result, the comparison of the absolute value to 0.5 may lead in some rare cases to different results depending on the computational setting.

Algorithm 2: ComputeVotes

input : luminance channel Y
output: votes

votes ← NON VALID — *initialize votes*
zeros ← 0 — *initialize number of zeros*
for $b \in$ Blocks8x8 **do** — *loop on all 8×8 blocks*
  $d \leftarrow \mathrm{DCT}(Y(b))$ — *DCT of the block*
  $z \leftarrow \sum_{\substack{d_{i,j} \in d \\ i,j \neq 0,0}} \mathbb{1}_{|d_{i,j}|<0.5}$ — *number of zeros in the block, excluding the DC coefficient*
  for $(x, y) \in b$ **do**
    if $z = \mathrm{zeros}(x, y)$ **then** — *tie, do not vote*
      $\mathrm{votes}(x, y) \leftarrow$ NON VALID
    else if $z > \mathrm{zeros}(x, y)$ **then**
      $\mathrm{zeros}(x, y) \leftarrow z$
      if IsConstantAlongH(b) **or** IsConstantAlongV(b) **then**
        $\mathrm{votes}(x, y) \leftarrow$ NON VALID
      else
        $\mathrm{votes}(x, y) \leftarrow \mathrm{GridAlignedWith}(b)$
votes(7-pixel wide border) ← NON VALID — *prevent border pixels from voting*
return votes

The vote maps illustrated in the examples of this article and in the demo are colored with a Python script. The NON VALID votes are in black, the other votes are colored. For example, the vote for the grid $(0, 0)$ is green.

5.2 Main Grid Detection

After computing the vote map, the NFA for each of the 64 possible grids is computed on the whole image. Algorithm 3 describes the procedure. The most meaningful one, which is the one with most votes, will be called the main grid. If none of the grids is meaningful, which means that the NFA is larger than 1, then no main grid is detected. A warning is produced when more than one global grid is found, as this may indicate that the image was manipulated.

Concerning the numerical implementation, two comments are relevant. First, in our implementation, the computation of the binomial tail is performed using the following relation to the Gamma function

$$\binom{n}{k} = \frac{\Gamma(n+1)}{\Gamma(k+1) \cdot \Gamma(n-k+1)},$$

for which there are effective implementations readily available, for example on http://www.rskey.org/gamma.htm. To speed up the computations, the sum of the binomial tail is truncated when the error can be bounded to be less than 10%.

Secondly, the NFA may reach very small values, which may underflow the usual IEEE 754 number representation. Our implementation in the C programming language, which uses IEEE 754 number representation, computes $\log_{10}(\mathrm{NFA})$ instead of NFA, allowing for a larger numeric range. Any logarithm base is equally useful for this purpose; the 10 base makes it slightly easier to read the order of magnitude of the NFA values. Of course, the test must now compare $\log_{10}(\mathrm{NFA})$ to $\log_{10}(\varepsilon)$, which for $\varepsilon = 1$ is zero.

Algorithm 3: DetectGlobalGrids

input : votes
output: main grid Γ

GridFound $\leftarrow 0$
for $g \in$ *AllGrids* **do**
  $v \leftarrow \sum_{x,y} \mathbb{1}_{\text{votes}(x,y)=g}$
  $\text{NFA}_g \leftarrow 64^2 \cdot (XY)^2 \cdot \mathcal{B}\left(\frac{XY}{64}, \frac{v}{64}, \frac{1}{64}\right)$
  if $\text{NFA}(g) < 1$ **then**
    GridFound $\leftarrow$ GridFound $+ 1$
if GridFound ≥ 2 **then**
  Warning("More than one meaningful grid globally")
if GridFound ≥ 1 **then**
  $\Gamma \leftarrow \arg\min_g \text{NFA}(g)$
  return Γ *JPEG grid found*
else
  return $\varnothing$ *JPEG grid not found*

5.3 Forgery Detection

After determining the main grid (one of the 64 different possibilities or none at all), a forgery may be a local area having a meaningful grid different from the main one or a missing grid when there is a main grid. To detect each type of forgery, the following method described in Algorithm 4 is called with different parameters. The vote map (which may be the initial vote map or the secondary one after the JPEG compression) is analyzed by a region growing method (Algorithm 5) to detect regions that voted for a particular grid. For the foreign JPEG grid detection, the method aims at detecting regions that voted for a grid that is different from the main grid. For the missing JPEG grid detection, the method aims at detecting regions that voted for the grid $(0, 0)$[4].

A NFA computation is done in a bounding box around the area and a forgery is detected if the result is meaningful. This step results in a binary forgery mask.

In the reference code, to accelerate the method, the NFA is only computed if the region is larger or equal to a minimal size of $\left\lceil 64 \frac{\log N_T}{\log 64} \right\rceil$, which corresponds to the minimal size of a region which can lead to meaningful detection in the most favorable case in which all pixels vote for the same grid. Indeed, for any smaller region, the NFA will never be smaller than 1.

6 Computational Complexity

ZERO, as described in Algorithm 1, essentially computes a vote map and then goes through all the votes to apply the statistical validation. To create the vote map, the equivalent of 64 JPEG compressions are needed. Indeed, the main cost of Algorithm 2 is computing the DCT of every 8×8 block in the image, while the JPEG algorithm computes the DCT of non-overlapping 8×8 blocks,

[4]In the reference C code, the grids are numbered from 0 to 63, 0 corresponding to $(0, 0)$ and 63 to $(7, 7)$. The function `detect_forgeries` takes as input a grid number to be excluded, plus the largest grid number to be included. Therefore, for the foreign grid detection: the main grid is selected as the excluded grid parameter and the maximum grid value is set to 63; for the missing grid detection: the NON VALID grid is set as grid to be excluded and the maximum grid number is set to 0, thus only grid 0, i.e. $(0, 0)$, is used.

Algorithm 4: DetectForgeries

input : votes defined on Ω of size $X \times Y$
input : main grid Γ
input : set of grids to exclude χ
input : neighborhood size $W = 9$
output: forgery mask

mask$(\Omega) \leftarrow$ FALSE — *initialize forgery mask*
usedpixels$(\Omega) \leftarrow$ FALSE — *initialize used pixels' mask*
for $(x, y) \in \Omega$ **do**
  if votes(x, y) *is* VALID **and** votes$(x, y) \notin \chi$ **and** usedpixels$(i, j) =$ FALSE **then**
    $R \leftarrow$ GrowRegion(votes, x, y, W, usedpixels) — *algorithm 5*
    $B \leftarrow$ BoundingBox(R)
    NFA $\leftarrow 64^2 \cdot (XY)^2 \cdot \mathcal{B}\left(\frac{B_x B_y}{64}, \frac{|R|}{64}, \frac{1}{64}\right)$
    if NFA < 1 **then** — *forgery found*
      mask$(R) \leftarrow$ TRUE — *mark tampered region*
    usedpixels$(R) \leftarrow$ TRUE — *do not test again in R*
mask $\leftarrow$ Closing(mask, W) — *fill holes in mask*
return mask

Algorithm 5: GrowRegion

input : votes
input : seed pixel (x, y)
input : distance to look for neighbors W
input : usedpixels
output: a region of pixels R

$\gamma \leftarrow$ votes(x, y)
$R \leftarrow (x, y)$
repeat
  for $(x', y') \in R$ **do**
    for $i \in \{x' - W, \ldots, x' + W\}$ **do**
      for $j \in \{y' - W, \ldots, y' + W\}$ **do**
        if votes$(x, y) = \gamma$ **and** usedpixels$(i, j) =$ FALSE **then**
          $R \leftarrow R \cup (i, j)$
until R *does not change*
return R

which implies keeping only one out 64 blocks. The DCT computations are independent, making it possible for us to implement them in parallel (OpenMP is used in the reference code). Afterwards, the validation and the region growing both have linear complexity relative to the number of pixels. The whole process is roughly repeated, after a JPEG compression, looking for missing grids. As a result, the computational cost is roughly the same as performing 129 times the JPEG compression of the same image.

More precisely, there are XY blocks to be processed (neglecting incomplete blocks on the border), and the DCT transform of 8×8 block requires a constant number of operations. Thus, computing

the vote map has a computational complexity linear with the number of pixels in the image. The validation and the region growing steps are also linear with the number of pixels. Performing a JPEG compression is also linear. All in all, the computational complexity of ZERO is linear with the number of pixels

$$\text{ComputationalComplexity} = O(XY),$$

where $X \times Y$ is the size of the input image.

7 Experiments

This section will illustrate the different cases of use of the proposed method through multiple experiments. The input image as well as the output of ZERO will be shown. The latter includes one or two colored vote maps, one or two forgery masks and a colored merged forgery mask. An additional text output, provided by the reference code, gives an interpretation and the relevant information.

7.1 JPEG Detection

If the image is large enough, ZERO can detect JPEG traces even for high quality lossy JPEG compression (up to $QF = 99$). Figure 13 shows an example where the image (of size 512×512) on the right is the compressed version with quality factor 99 of the image on the left.

Figure 13: Up: uncompressed image and compressed image at quality 99. Down: vote maps.

The vote maps on Figure 13 illustrate the slight difference between a vote map of an uncompressed image and a high-quality JPEG compressed image. The amount of green on the vote map on the right is enough to detect that the image has gone through a JPEG compression and to name the

correct main grid. The forgery masks (not shown) are all black, since in these cases no forgeries were detected.

The following text output explains that the main grid of the image is $(0, 0)$. Therefore, the image has gone through a JPEG compression.

```
main grid: #0 [0 0] log(nfa) = -296.33

No suspicious traces found in the image with the performed analysis.
```

Note that the NFA, expressed by its logarithm, is actually $10^{-296.33}$, an extremely small number, indicating how meaningful the detection is.

7.2 Crop Detection

Figure 14: Up: compressed image and cropped version of the same image. Down: vote maps.

In Figure 14, we took an original JPEG image and cropped a square out of it. The origin of the global grid being different from $(0, 0)$, the (anticipated) conclusion is that the image has been cropped. The vote maps have different colors: the first one is for the $(0, 0)$ grid and the second one for the $(4, 4)$ grid.

7.3 Unaligned Double Compression Detection

Double compression can be detected by the presence of two or more global grids. Nevertheless, the detection by ZERO is possible only when the two grids are not aligned. Figure 15 shows three experiments, in which the same image have been respectively and successively, compressed, cropped and compressed again. The conclusion on the first is that it was JPEG compressed. The result on the second image shows a JPEG grid origin different from $(0, 0)$, indicating that the image has been cropped. Finally, in the last vote map of Figure 15, the two colors are present. The most significant

is the one from the last compression, even if the quality is higher than the previous one. If the second compression were much stronger (with a lower quality), then it would probably have masked the previous one. Then only one grid would have been detected.

The text output for the first image is:

```
main grid found: #0 (0,0) log(nfa) = -6060.7

No suspicious traces found in the image with the performed analysis.
```

For the second image:

```
main grid found: #36 (4,4) log(nfa) = -5959.07

The most meaningful JPEG grid origin is not (0,0).
This may indicate that the image has been cropped.
```

And for the last image:

```
main grid found: #0 (0,0) log(nfa) = -3110.45

meaningful global grid found: #36 (4,4) log(nfa) = -169.145

There is more than one meaningful grid. This is suspicious.
```

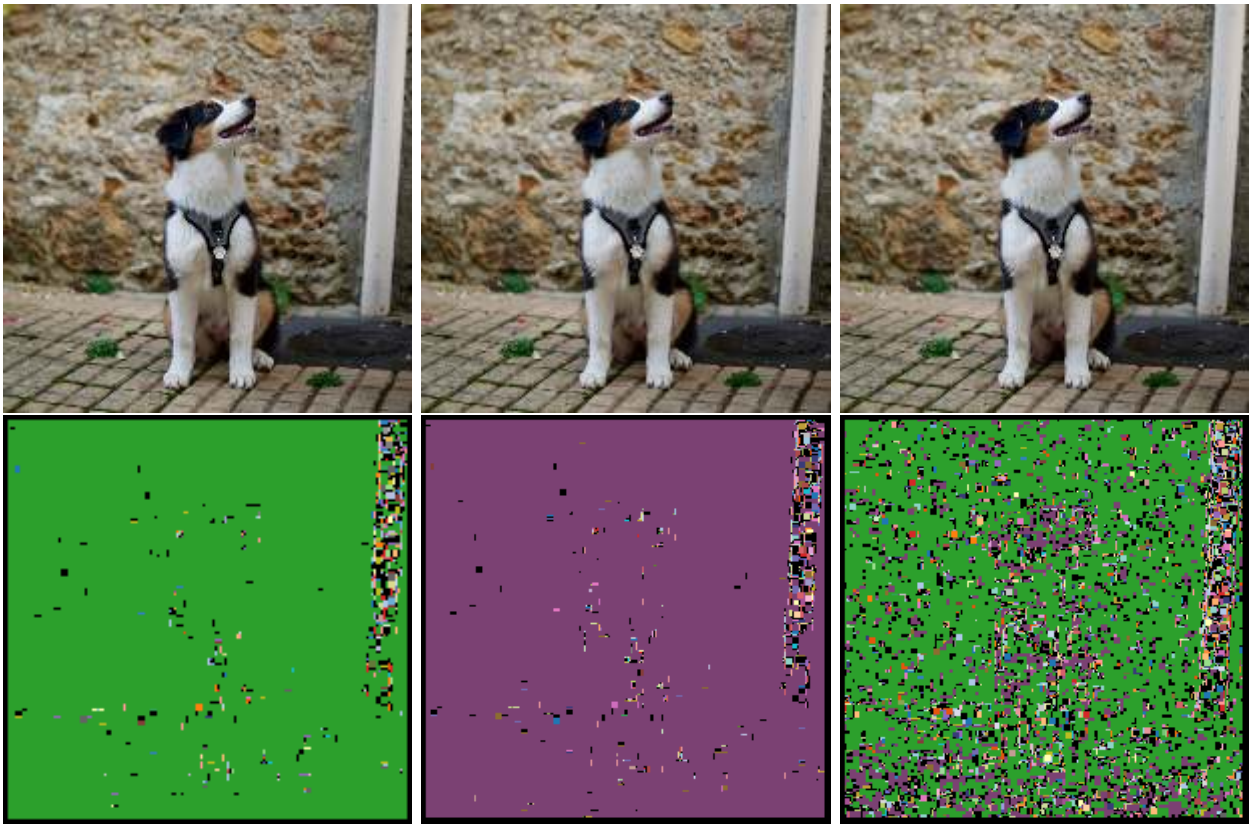


Figure 15: Left: compressed image at quality 95 and its vote map. Middle: previous image cropped and its vote map. Right: previous image compressed at quality 98 and its vote map.

7.4 Copy-move Forgery

The forged image in Figure 16 was made by internal copy-move operations. A brick has been copy-moved four times. This is clearly visible in the vote map as regions with a different vote than the background grid $(0, 0)$. The resulting forgery mask shows a perfect detection in this case. The forgery mask F shows the area were foreign grids (different from the main one $(0, 0)$) were detected.

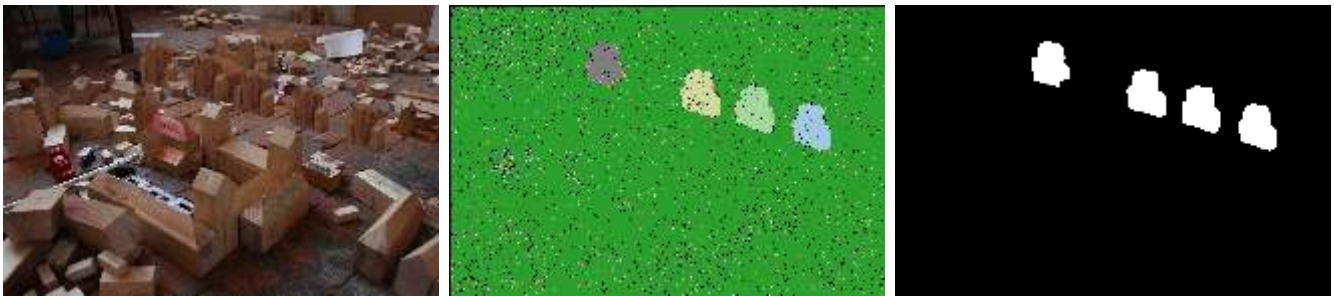


Figure 16: Left: tampered image by copy-move. Middle: vote map. Right: forgery mask F.

The text output of the method, when the first part of the method is executed, is:

```
main grid found: #0 (0,0) log(nfa) = -224056

A meaningful grid different from the main one was found here:
bounding box: 1206 305 to 1554 728 [349x424] grid: #14 (6,1) log(nfa) = -1976.32

A meaningful grid different from the main one was found here:
bounding box: 2042 563 to 2399 994 [358x432] grid: #27 (3,3) log(nfa) = -2175.22

A meaningful grid different from the main one was found here:
bounding box: 2538 702 to 2881 1125 [344x424] grid: #50 (2,6) log(nfa) = -2204.62

A meaningful grid different from the main one was found here:
bounding box: 3042 863 to 3391 1294 [350x432] grid: #42 (2,5) log(nfa) = -2085.66

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

7.5 Splicing Forgery

The image in Figure 17 is quite interesting because the main grid is not $(0, 0)$ but the forged area is. Indeed, the algorithm detects several global grids in the image. This is a cue that the image has gone through multiple JPEG compressions with different grid origins. Indeed, the image on the bottom-left (the original image) has probably been compressed, then cropped and forged before being compressed again. ZERO detects as most meaningful the first compression, which indeed has the lowest quality.[5] The forgery is therefore the area having a different grid than the rest. In this case, the pristine area has gone through two compressions and ZERO detects the one with the $(6, 6)$ grid, while the forged area only shows traces of the last compression with the $(0, 0)$ grid. If the inserted image region had any trace of a previous JPEG compression, it was probably erased by the re-sampling needed to make it fit in the desired position. The text output of ZERO is:

[5]This can be verified by JPEG quantization table estimation methods.

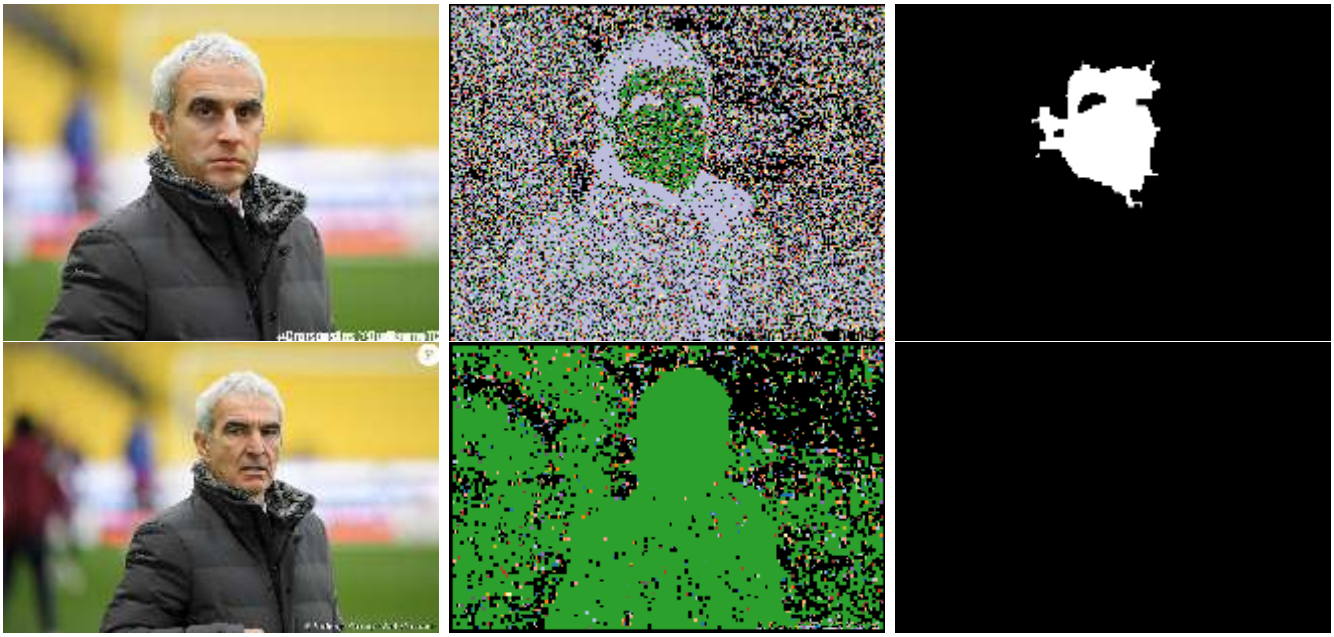


Figure 17: Up: forged image from the well-known Twitter account GuillaumeTC, its vote map and forgery mask F. Down: Original image found online, its vote map and forgery mask F.

```
main grid found: #54 (6,6) log(nfa) = -14127.2
meaningful global grid found: #0 (0,0) log(nfa) = -494.782
meaningful global grid found: #6 (6,0) log(nfa) = -13.2262

A meaningful grid different from the main one was found here:
bounding box: 622 247 to 1215 917 [594x671] grid: #0 (0,0) log(nfa) = -736.034

The most meaningful JPEG grid origin is not (0,0).
This may indicate that the image has been cropped.

There is more than one meaningful grid. This is suspicious.

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

7.6 Compressed Forgery in an Uncompressed Image

Figure 18: Left: tampered image. Middle: vote map. Right: forgery mask F.

The image in Figure 18 has not been JPEG compressed, however the spliced area comes from

an image having JPEG traces. The result is that the global image has no JPEG grid but a local detection is made. In this case, the method stops with the computation of the forgery map F of foreign grids and does not try to detect areas with missing JPEG grid since this is the main estimation in the image.

The text output of ZERO is:

```
No overall JPEG grid found.

A meaningful grid was found here:
bounding box: 1461 642 to 1804 883 [344x242] grid: #13 (5,1) log(nfa) = -86.9189

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

7.7 Missing JPEG Grid Forgery

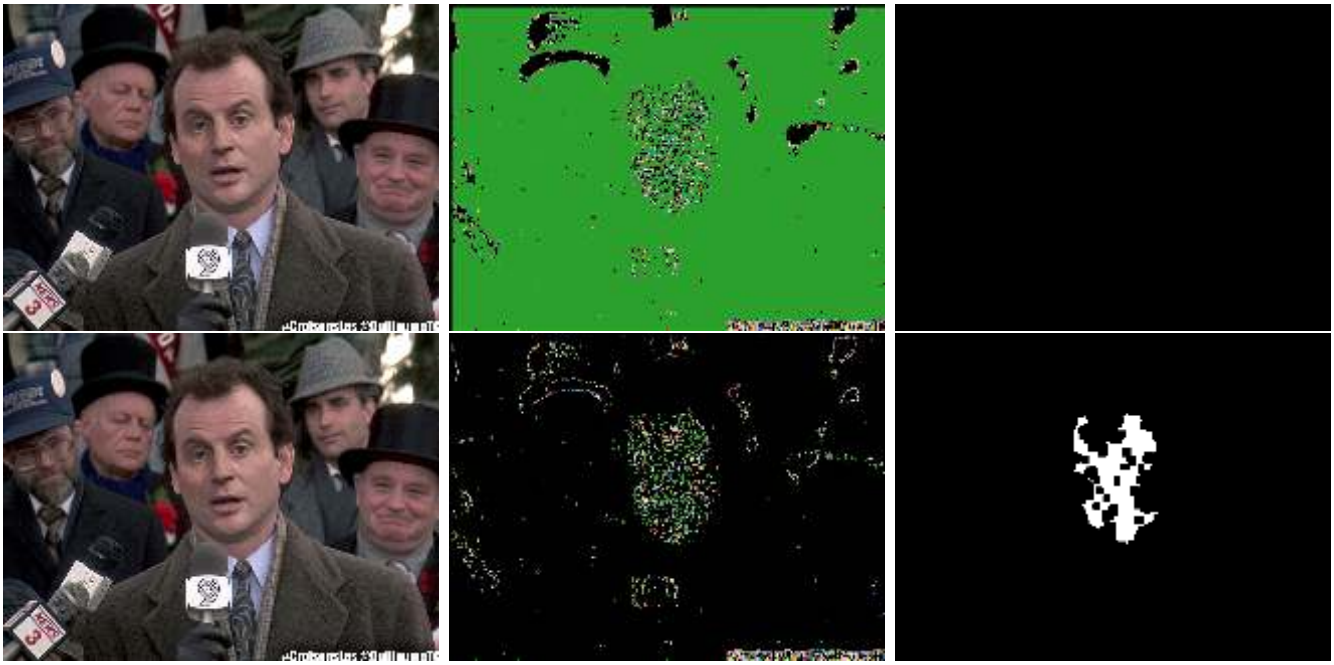


Figure 19: Up: forged image, its vote map and forgery mask F. Down: compressed version at $QF = 99$, its vote map for the second round and forgery mask M. The forged image comes from GuillaumeTC's twitter account.

The forged area in the image in Figure 19 has no detectable JPEG traces. This can be seen in the colored vote map in the middle-top image of Figure 19. This image is to be compared with the middle-bottom vote map obtained after imposing a $QF = 99$ JPEG compression to detect the missing grids. In the middle-top vote map, the central face shows a different behaviour than its background: In it there are fewer and less meaningful votes for the main grid, and no other grid is meaningful either. Indeed, in this area the JPEG traces (present in the rest of the image) are missing. Applying the compression (bottom images) reveals JPEG traces with grid origin $(0, 0)$ in this area; the missing traces are not due to the image contents, it is indeed a forged area without JPEG traces, and it is detected.

The text output of ZERO is:

```
main grid found: #0 (0,0) log(nfa) = -33624

A region with missing JPEG grid was found here:
bounding box: 568 256 to 839 670 [272x415] grid: #0 (0,0) log(nfa) = -49.259

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

7.8 Removed Area with Healing

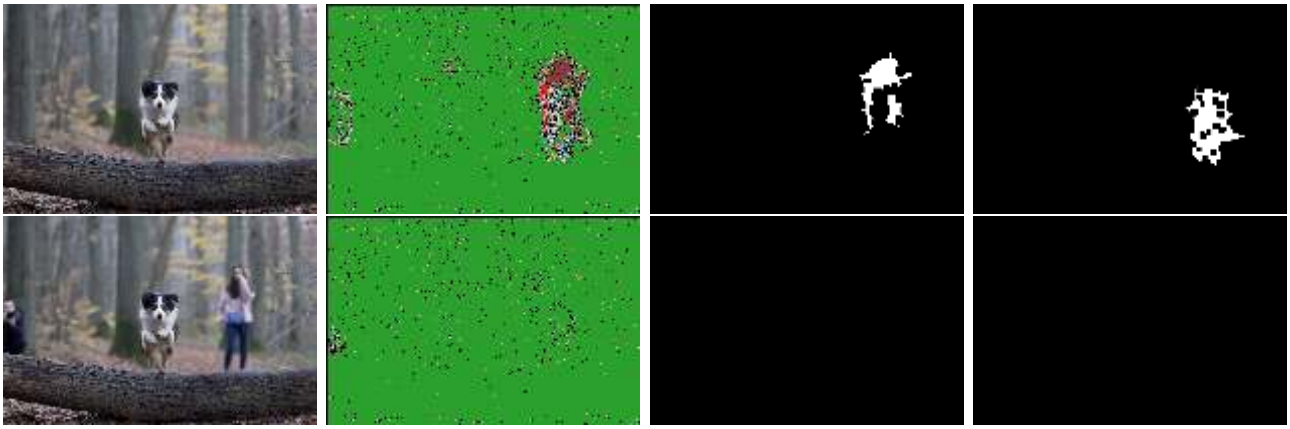


Figure 20: Up: forged image by healing, vote map, forgery mask F and forgery mask M. Down: original image, vote map, forgery mask F and forgery mask M.

In Figure 20, the forgery was made with the "healing" functionality of the software Affinity. The inpainting removed two persons in the scene background. For the person on the left, the edges can be seen in the vote map. For the inpainting region on the right, several different grids were detected. Most likely, the inpainting algorithm copied patches from the rest of the image to fill in the area, therefore also copying the JPEG traces in wrong positions. The first forgery mask in Figure 20 shows the areas with a foreign grid and the second forgery shows areas where JPEG traces are missing.

As can be noticed in the forgery maps of Figure 20, areas with a different JPEG grid overlap with areas with a missing grid. This overlap is mainly caused by the morphological mask closing and by the fact that bounding boxes are evaluated rather than the pixels themselves.

7.9 Merged Masks

A convenient way to visualise the results is by merging both forgery masks. Red is chosen for the areas that have a foreign JPEG grid whereas blue is for the areas having no traces of the main grid. If a block is both blue and red, it is marked in red, as this is the more valuable detection. See some examples in Figure 21.

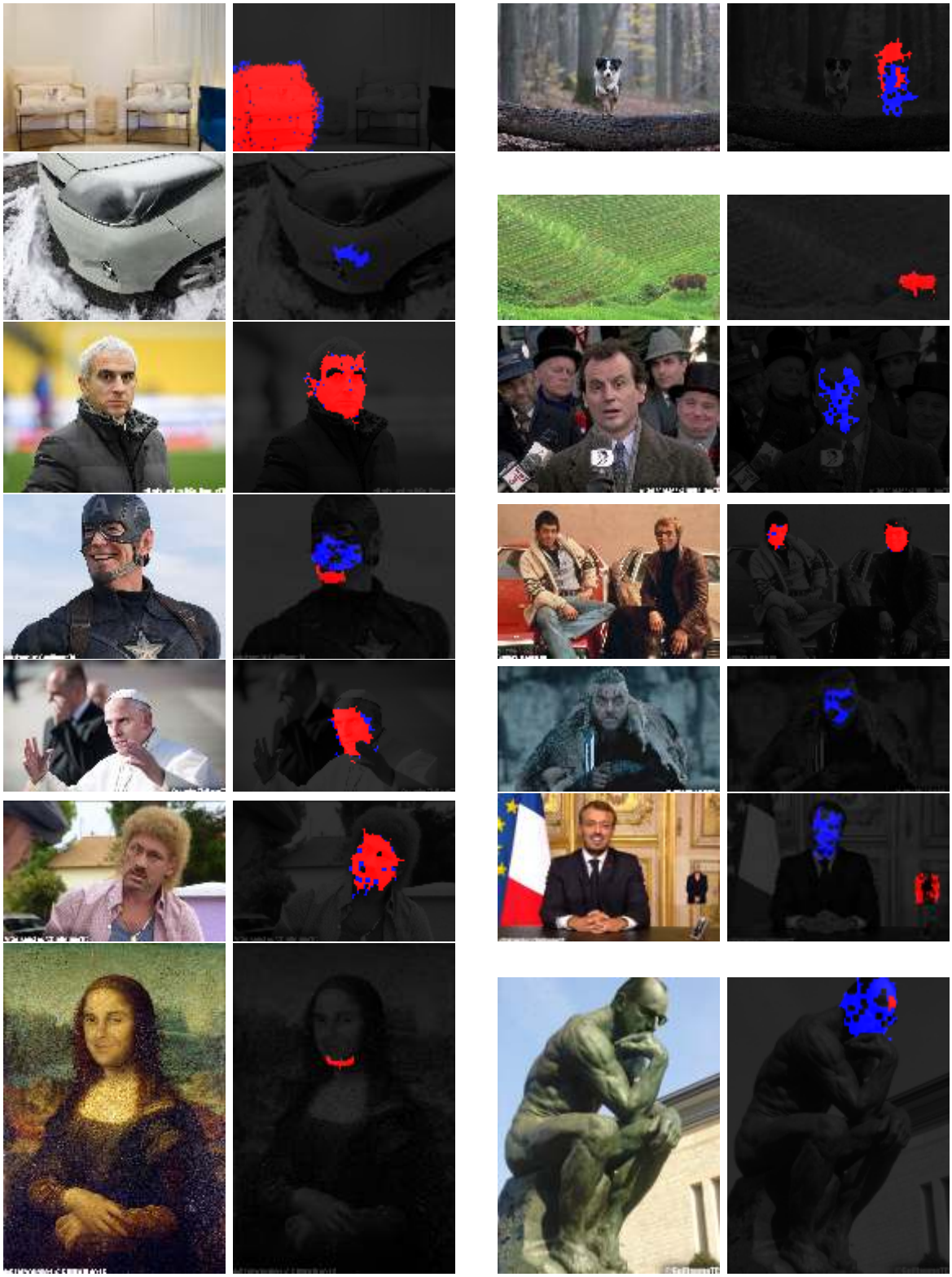


Figure 21: Images to analyze and the final results of both forgery masks merged.

7.10 Variability Due to Floating Point Arithmetic

Figure 22: Left: forged image. Middle: result on MacOS, where a detection was made, in red. Right: result on Linux, where no detection was made.

The reference code of Algorithm 2 dealing with floating point arithmetic, is implemented in the C programming language. Depending on the system hardware, the compiler used, and the compiler optimization options, different approximations may be performed. This may lead occasionally to slightly varying DCT values which, when compared to ± 0.5 for the rounding, may result in an additional or missing zero. Small differences may occur in the vote maps and therefore in the NFA computations. Figure 22 is an example where this difference in the voting process implies differences in the forgery maps.

The text output of the method on the two platforms were as follows. On a MacOS system:

```
main grid found: #0 (0,0) log(nfa) = -96814.3

A meaningful grid different from the main one was found here:
bounding box: 848 2493 to 912 2599 [65x107] grid: #57 (1,7) log(nfa) = -0.187477

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

and a Linux system:

```
main grid found: #0 (0,0) log(nfa) = -96839.7

No suspicious traces found in the image with the performed analysis.
```

As can be observed, the numerical values are very similar, but nevertheless different, and these small differences lead in rare occasions, as the one in Figure 22, to different detection results.

8 Limitations

8.1 Quality Factor 1 and Quality Factor 100

The result for an uncompressed image, compressed at quality 100, or for an image compressed at quality 1, give the same result:

```
No overall JPEG grid found.
```

An image compressed at $QF = 100$ has no zero DCT coefficients as their values are not quantized with this quality factor. At $QF = 1$ instead, non-detection is caused by the fact that most of the pixels do not vote as they belong to constant blocks. Since no vote is performed, no grid is found, see Figure 23.

The strong compression cases can be handled easily by other methods [17]. Also, as explained before, refining the voting approach should allow us to handle these cases using the same main ideas, at the price of a more complex algorithm.

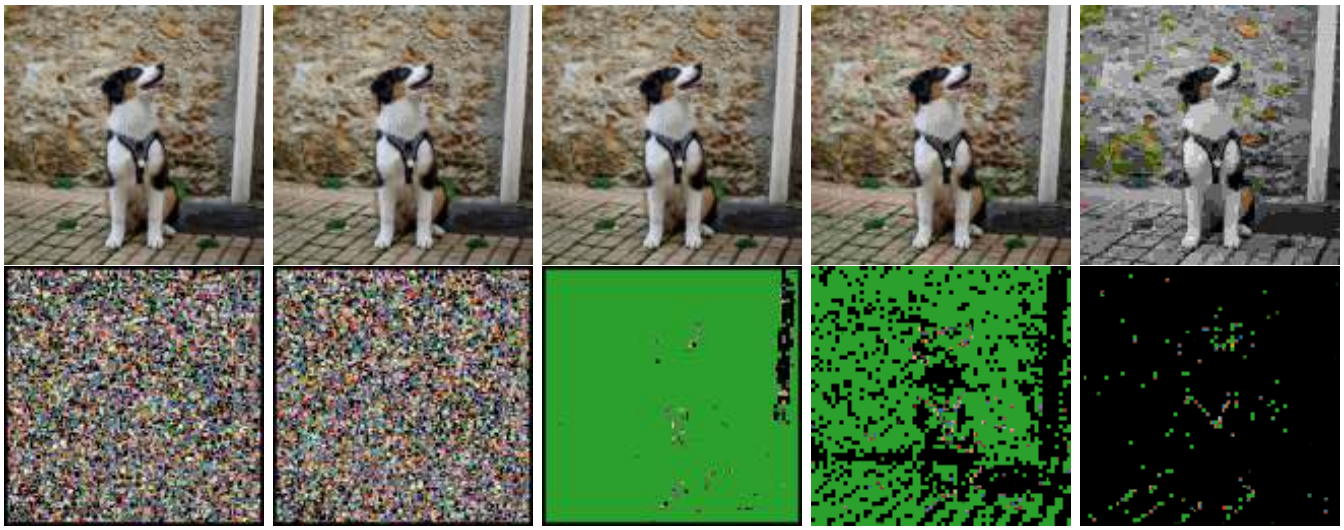


Figure 23: Images and vote maps for the uncompressed image (left) and for JPEG compression with $QF = 100, 80, 10, 1$ (middle-left, middle, middle-right, and right, respectively).

8.2 Missed Detection with a Chance of 1/64

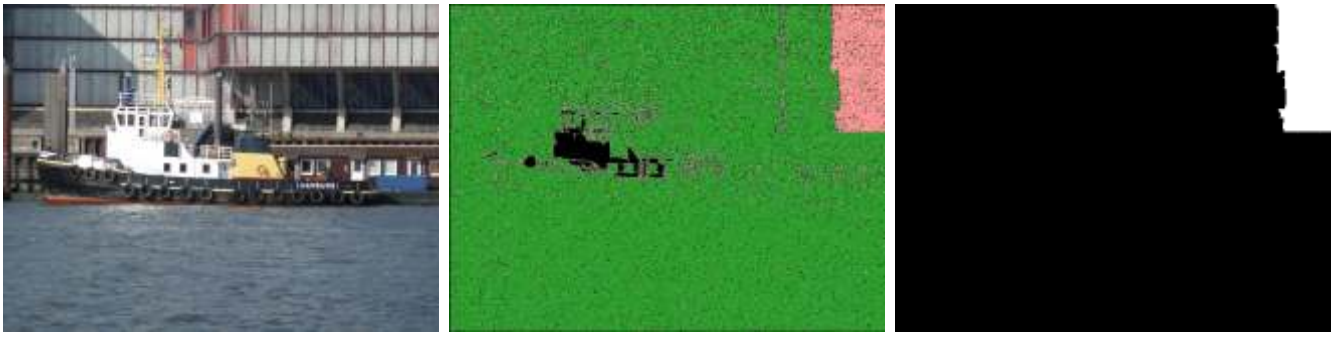


Figure 24: Left: tampered image by copy-move (twice). Middle: vote map. Right: forgery mask F. An example of a missed detection: one of the two forged regions was not detected because its local grid was correctly aligned with the global grid.

Figure 24 illustrates a limitation of the proposed method, and all the methods based on the JPEG grid origin. Forgeries are detected as regions in which the local grid origin does not agree with the one of the global grid. This means that when the grid of the forged regions aligns perfectly with the global grid, the proposed method will fail to detect the forgery. Nevertheless, this happens only once for every 64 positions. In Figure 24, the same rectangle area was copied twice in the forged image, but only one of the copies was detected. The other one has the correct grid origin. Since the copy-move was, in this case, done only horizontally, this happened with a chance of 1/8.

8.3 Saturated Area

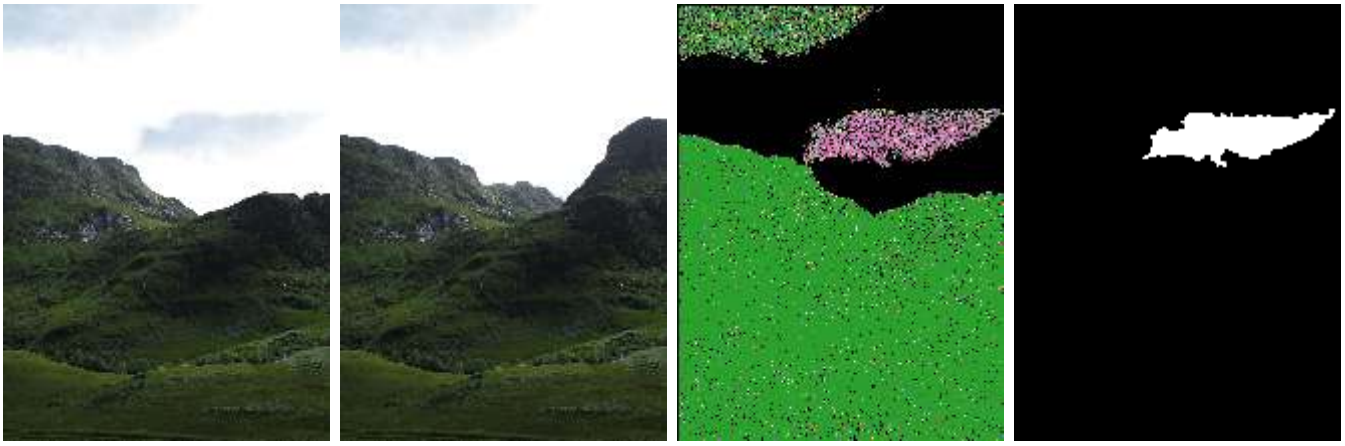


Figure 25: Left: tampered image in the sky. Middle-left: original image. Middle-right: vote map. Right: forgery mask F. An incomplete detection caused by saturation in the image.

In a saturated region, the DCT coefficients of the blocks are all equal to zero, except for the DC coefficient. The number of zeros are tied and the votes are all NON-VALID. Thus it is impossible for the method ZERO to distinguish the JPEG grid in saturated regions. Since no valid JPEG grid can be found, it will never disagree with the global grid and therefore saturated parts of a forgery cannot be found. However, as soon as a part of the forgery is not saturated it can be detected as it is shown in Figure 25.

8.4 Small Forgeries

Figure 26: Left: tampered image by copy-move of three people. Middle: vote map. Right: forgery mask F. An example of a missed detection: one of the three forged regions was not detected because it is too small.

Another limitation is when the forged region is too small. In Figure 26, three people have been copy-moved in the image but only two of them have been detected by the method ZERO. Since the statistical test must be satisfied to detect a forgery, there is a minimal detectable region size that depends on the image size, the JPEG compression quality and the image contents.

8.5 Resampling Detections

Resampling traces may disrupt JPEG blocking traces. Indeed, resampling an image creates a regular pattern [19] which can, when aligned horizontally or/and vertically, interfere with the JPEG 8×8 grid. For example, a JPEG image loses (naturally) its JPEG blocking artifact when stretched, as the period of the artifacts is no longer eight pixels. However, sometimes, it creates a new periodic

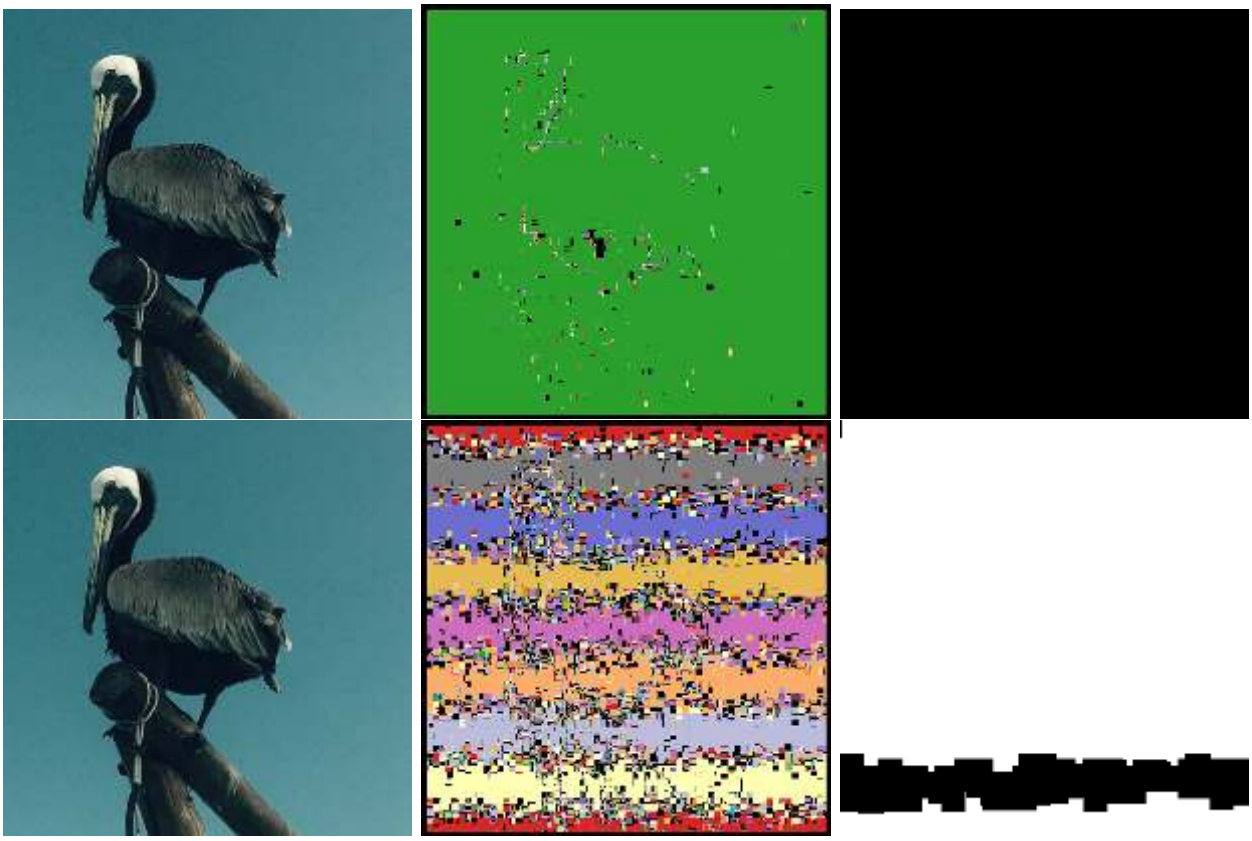


Figure 27: Up: Compressed image and its vote map and forgery mask F. Down: Compressed and stretched version of the image (8 pixels in height) and its vote map and forgery mask F.

pattern as it can be detected in the image of Figure 27. The image was JPEG compressed and of size 512×512, after being stretched vertically, became of size 512×520. The horizontal JPEG traces are still present. The vertical JPEG traces are modified, but the interference between the original sampled grid, and the new re-sampling grid leads to local artifacts with a vertical periodicity of near eight pixels. The periodic pattern is detected locally, therefore it results in partial detections covering almost all the image, indicated as forgeries. One of the local detections is selected as the main grid just because it is slightly more meaningful than the others.

The text output of the method shows the numerical values for these detections:

```
main grid found: #62 (6,7) log(nfa) = -148.748
meaningful global grid found: #6 (6,0) log(nfa) = -81.0943
meaningful global grid found: #14 (6,1) log(nfa) = -143.941
meaningful global grid found: #22 (6,2) log(nfa) = -143.941
meaningful global grid found: #30 (6,3) log(nfa) = -122.885
meaningful global grid found: #38 (6,4) log(nfa) = -107.188
meaningful global grid found: #46 (6,5) log(nfa) = -116.833
meaningful global grid found: #54 (6,6) log(nfa) = -141.555

A meaningful grid different from the main one was found here:
bounding box: 7 7 to 504 55 [498x49] grid: #6 (6,0) log(nfa) = -106.731

A meaningful grid different from the main one was found here:
bounding box: 7 8 to 504 121 [498x114] grid: #14 (6,1) log(nfa) = -337.412

A meaningful grid different from the main one was found here:
```

```
bounding box: 7 35 to 504 186 [498x152] grid: #22 (6,2) log(nfa) = -279.446

A meaningful grid different from the main one was found here:
bounding box: 7 122 to 504 251 [498x130] grid: #30 (6,3) log(nfa) = -265.387

A meaningful grid different from the main one was found here:
bounding box: 7 203 to 504 316 [498x114] grid: #38 (6,4) log(nfa) = -255.273

A meaningful grid different from the main one was found here:
bounding box: 7 275 to 504 405 [498x131] grid: #46 (6,5) log(nfa) = -255.758

A meaningful grid different from the main one was found here:
bounding box: 7 325 to 504 438 [498x114] grid: #54 (6,6) log(nfa) = -317.069

A meaningful grid different from the main one was found here:
bounding box: 7 463 to 504 512 [498x50] grid: #6 (6,0) log(nfa) = -89.6458

The most meaningful JPEG grid origin is not (0,0).
This may indicate that the image has been cropped.

There is more than one meaningful grid. This is suspicious.

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

8.6 Double Compression

The fact that the algorithm may detect several global grids may come from the fact that the image was compressed multiple times with a crop operation in the middle. For example, the image in Figure 28 was compressed, then cropped and then compressed again in a lighter way than the first compression. The output of the method is:

```
main grid found: #0 (0,0) log(nfa) = -7462.48
meaningful global grid found: #7 (7,0) log(nfa) = -53.2931
meaningful global grid found: #39 (7,4) log(nfa) = -2782.57

A meaningful grid different from the main one was found here:
bounding box: 7 7 to 1377 1374 [1371x1368] grid: #39 (7,4) log(nfa) = -2753.02

A region with missing JPEG grid was found here:
bounding box: 7 7 to 1377 1347 [1371x1341] grid: #0 (0,0) log(nfa) = -677.296

A region with missing JPEG grid was found here:
bounding box: 7 1115 to 471 1369 [465x255] grid: #0 (0,0) log(nfa) = -23.3211

There is more than one meaningful grid. This is suspicious.

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
```

to make your own opinion about a potential forgery.

Figure 28: Not aligned double compressed image, its vote map and result. This image was first compressed with quality 90, then cropped so that the grid origin became $(7, 4)$ and compressed again with quality 98.

The method detected the first grid $(7, 4)$ and the second one $(0, 0)$ as the main one, but also detected a grid with the horizontal coordinate of the first compression and the vertical coordinate of the second one. It could have detected also the grid $(0, 7)$. This is both an asset of the method (detecting double compression is a first cue to forgery detection) but also a drawback since the local detection is afterwards disrupted.

8.7 Big JPEG Compressed Forgery in an Uncompressed Image

Figure 29: JPEG forgery in an uncompressed image, its vote map and result. An example of wrongly attributed detection: a JPEG compressed image has been pasted into an uncompressed image. The only JPEG traces are the ones in the forged part, which is large enough to result in a global detection. Indeed, this part is larger than $1/64th$ of the image. The method correctly detects the missing JPEG grids in the background. The result is that the background is marked as forged, while it is more natural to say that the small region is the forged one.

In the example in Figure 29, the forged area is JPEG compressed and the rest of the image is not. This could be detected as it was in the example of Figure 18. Here, however, the forged area is larger than $1/64th$ of the image. Therefore, its grid is detected as the global grid. The forged mask F is all black, detecting no local grid. The algorithm detected, correctly, the presence of anomalies in the image. However, the interpretation produced is not correct, or at least not the most natural one. Indeed, it would be more natural to say that the forgery is the small region in a larger background, although this is a subjective evaluation.

The text output for this image is:

```
main grid found: #2 (2,0) log(nfa) = -80.0532

A region with missing JPEG grid was found here:
```

```
bounding box: 7 7 to 1912 1072 [1906x1066] grid: #0 (0,0) log(nfa) = -1788.81

A region with missing JPEG grid was found here:
bounding box: 1698 7 to 1895 199 [198x193] grid: #0 (0,0) log(nfa) = -2.81145

The most meaningful JPEG grid origin is not (0,0).
This may indicate that the image has been cropped.

Suspicious traces found in the image.
This may be caused by image manipulations such as resampling,
copy-paste, splicing.  Please examine the deviant meaningful region
to make your own opinion about a potential forgery.
```

8.8 Missed Missing Grid Detection

The image in Figure 30 is a JPEG image where the forged area comes from an uncompressed image. The area may be too small in this example and is therefore not detected in the forgery mask M.

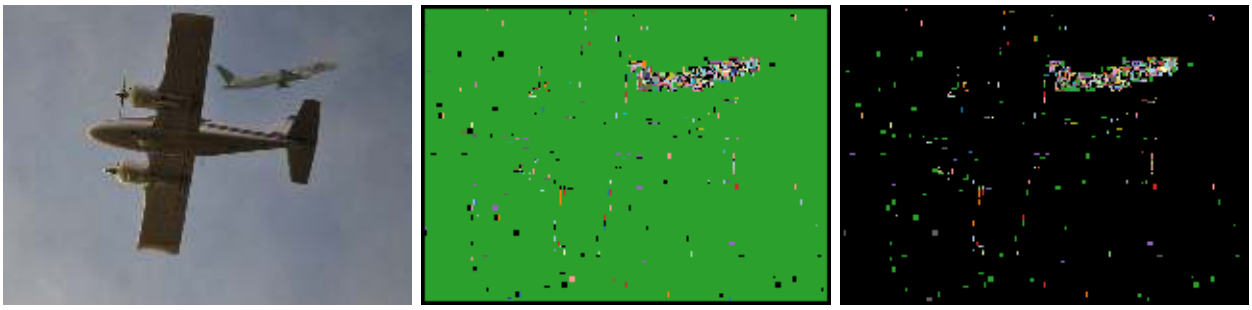


Figure 30: Tampered image and its vote maps F and M. An example of a missed detection: the forgery and the image are too small. The forged area is uncompressed.

The image in Figure 31 has probably gone through several JPEG compressions, as it was downloaded from Twitter. The face in the image has fewer JPEG traces than the rest of the image, like the example of Figure 19, but detection fails.

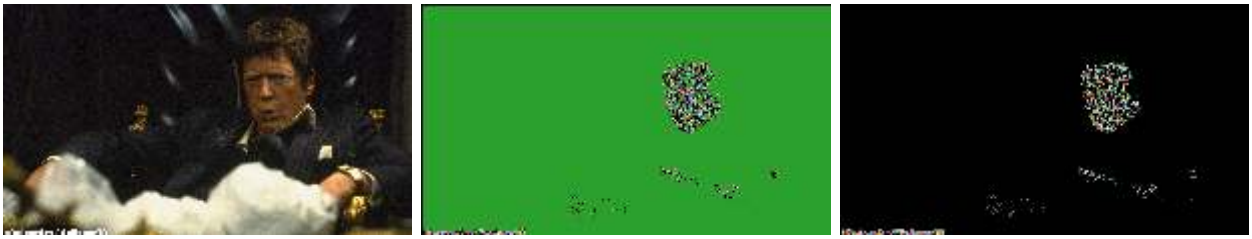


Figure 31: Tampered image from GuillaumeTC's Twitter account and its vote maps. An example of a missed detection.

Both the examples of Figure 30 and Figure 31 yield vote maps where visual detection is possible. However, after re-compressing with $QF = 99$ and removing the main grid votes of the previous map, the forgery was not detected, the traces were not meaningful enough. This rises the question of whether it would be better to use stronger JPEG compression in the missing grid detection step. And indeed, in the examples of Figure 30 and Figure 31, the forged regions are detected when re-compressing the image at quality 98 instead of 99. Figure 32 shows the colored votes and the final results of the detection.

Nevertheless, a stronger JPEG compression can lead to false detections too. Figure 33 shows an example of forgery detection when the re-compression step is performed with the quality factors 99,

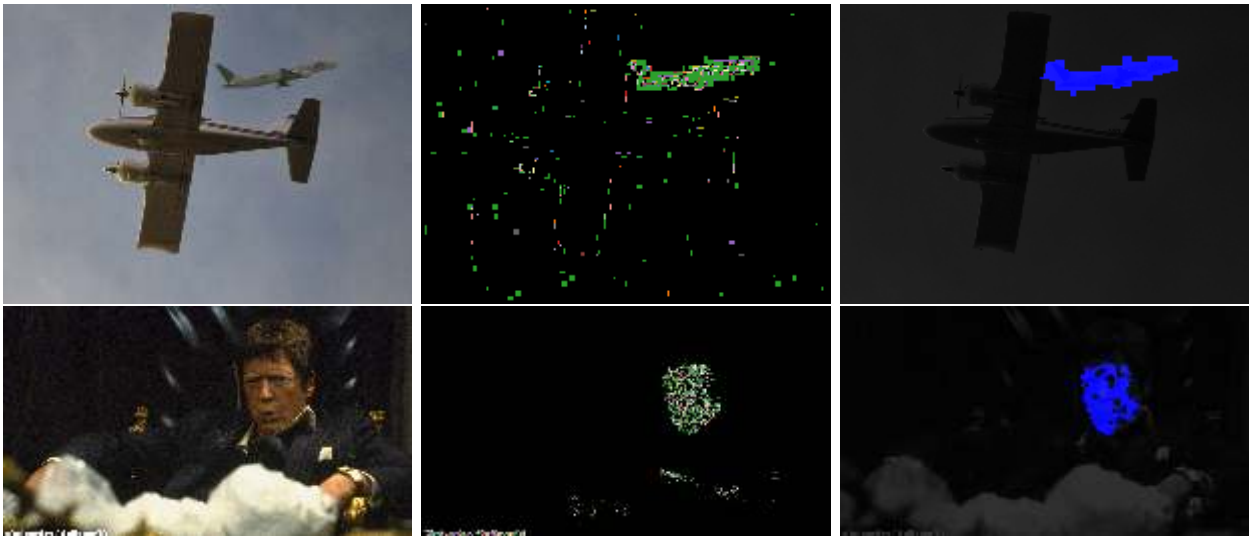


Figure 32: Tampered images, their vote maps after compression at quality 98 and final detection.

98 and 95. In this example again, better forgery detection is obtained with quality factor 98. Indeed, three men have been erased and the car and the second man from the left have been moved a bit and forged. Unfortunately, many false detections are also made with a lower quality factor (95 here). We decided to keep the quality factor to the maximum value 99 to keep the false detection rate to a minimum, even though some detections (visible in the vote map) are missed. Future work will focus on improving the detectability of missing grids while still controlling false detections.

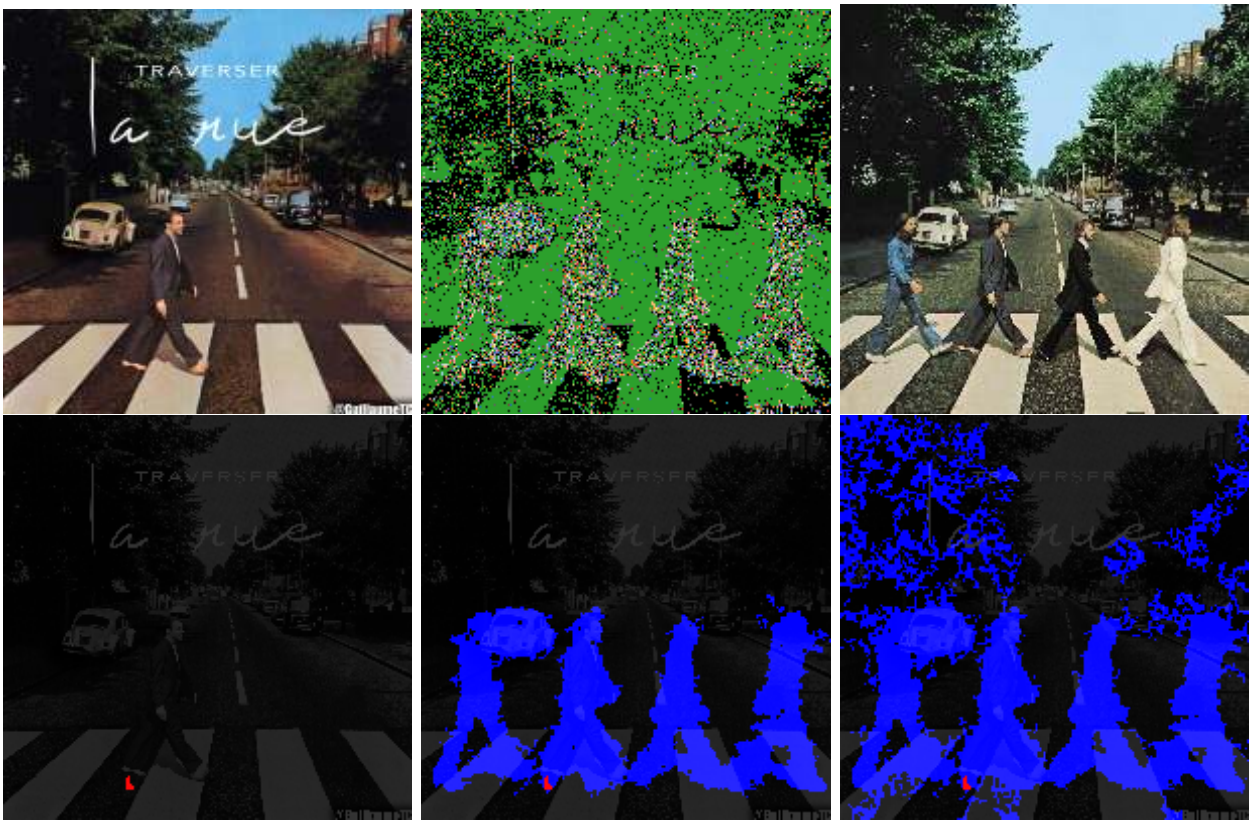


Figure 33: Up: Forged image, colored vote map and possible original image. Down: ZERO applied to the image of Figure 22 with compression levels 99, 98 and 95. All the forged area is detected in the second version but also some false detection at the top. Indeed, when compressing too much, the number of false detections increases as can be seen on the third image.

8.9 Merged Masks

Figure 34 shows examples of the different limitations described before by the merged masks visualisation, that combines the two forgery masks F and M.

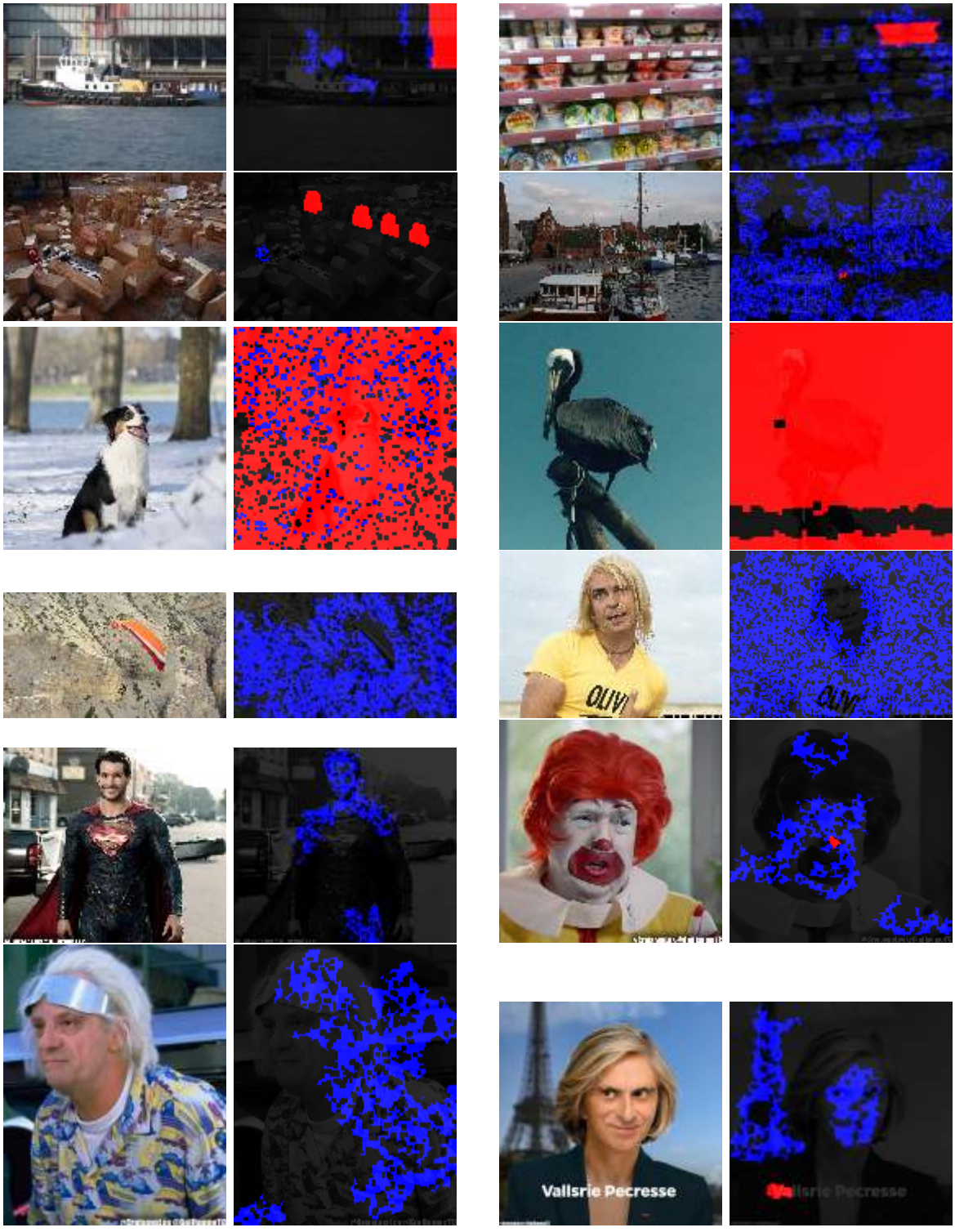


Figure 34: Examples of limitations of ZERO. Images to analyze and the final results of both forgery masks merged.

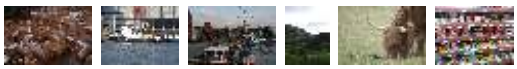
9 Conclusion

This paper describes a JPEG grid detection and tampering localization method based on the number of zeros in the DCT blocks. It has a high accuracy detecting JPEG compression up to quality factor of 99. It performs reliable reverse engineering and detects forgeries by giving an automatic, localized, and reliable result without requiring any human interpretation. The proposed algorithm is efficient; the bottleneck is the computation of the vote maps, which requires about the same number of operations as performing 129 JPEG compressions of the same image. The perspective for future work includes handling the color information, and improving the detectability of missing grids.

Acknowledgment

Work partially funded by the ANR-DGA challenge DEFALS (ANR-16-DEFA-0004) and IFCN for the project Envisu4.

Image Credits

 images from FAU dataset [5].

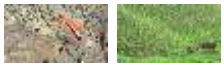
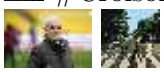
 images from Korus dataset [11, 12].

 image from DEFACTO dataset [15].


 #CroisonsLes images from @GuillaumeTC's twitter account[6].

 original images found on the Internet.

 images from the authors.

References

[1] *ITU-T recommendation T.81: Information technology digital compression and coding of continuous-tone still images requirements and guidelines*, tech. report, International Telecommunication Union, 1992.

[2] S. Agarwal and H. Farid, *Photo forensics from JPEG dimples*, in IEEE Workshop on Information Forensics and Security (WIFS), 2017, pp. 1–6. https://doi.org/10.1109/WIFS.2017.8267641.

[3] Q. Bammey, R. Grompone von Gioi, and J-M. Morel, *Automatic detection of demosaicing image artifacts and its use in tampering detection*, in IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), IEEE, 2018, pp. 424–429. https://doi.org/10.1109/MIPR.2018.00091.

[6] https://twitter.com/GuillaumeTC

[4] J. Chou, M. Crouse, and K. Ramchandran, *A simple algorithm for removing blocking artifacts in block-transform coded images*, IEEE Signal Processing Letters, 5 (1998), pp. 33–35. https://doi.org/10.1109/97.659544.

[5] V. Christlein, C. Riess, J. Jordan, C. Riess, and E. Angelopoulou, *An evaluation of popular copy-move forgery detection approaches*, IEEE Transactions on Information Forensics and Security, 7 (2012), pp. 1841–1854. https://doi.org/10.1109/TIFS.2012.2218597.

[6] A. Davy, T. Ehret, J-M. Morel, and M. Delbracio, *Reducing anomaly detection in images to detection in noise*, in IEEE International Conference on Image Processing (ICIP), IEEE, 2018, pp. 1058–1062. https://doi.org/10.1109/ICIP.2018.8451059.

[7] M. Delbracio, D. Kelly, M.S. Brown, and P. Milanfar, *Mobile computational photography: A tour*, Annual Review of Vision Science, 7 (2021), pp. 571–604. https://doi.org/10.1146/annurev-vision-093019-115521.

[8] A. Desolneux, L. Moisan, and J.-M. Morel, *From Gestalt Theory to Image Analysis*, Springer, 2008. ISBN: 978-0-387-74378-3.

[9] A. Gordon, G. Glazko, X. Qiu, and A. Yakovlev, *Control of the mean number of false discoveries, Bonferroni and stability of multiple testing*, Annals of Applied Statistics, 1 (2007), pp. 179–190. https://doi.org/10.1214/07-AOAS102.

[10] R. Grompone von Gioi, J. Jakubowicz, J-M. Morel, and G. Randall, *LSD: A fast line segment detector with a false detection control*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 32 (2010), pp. 722–732. https://doi.org/10.1109/TPAMI.2008.300.

[11] P. Korus and J. Huang, *Evaluation of random field models in multi-modal unsupervised tampering localization*, in IEEE International Workshop on Information Forensics and Security, 2016. https://doi.org/10.1109/WIFS.2016.7823898.

[12] ——, *Multi-scale analysis strategies in PRNU-based tampering localization*, IEEE Transactions on Information Forensics & Security, (2017). https://doi.org/10.1109/TIFS.2016.2636089.

[13] J. Lezama, R. Grompone von Gioi, G. Randall, and J-M. Morel, *Finding vanishing points via point alignments in image primal and dual domains*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 509–515. https://doi.org/10.1109/CVPR.2014.72.

[14] D.G Lowe, *Visual recognition from spatial correspondence and perceptual organization*, in International Joint Conference on Artificial Intelligence (IJCAI), Citeseer, 1985, pp. 953–959.

[15] G. Mahfoudi, B. Tajini, F. Retraint, F. Morain-Nicolier, J-L. Dugelay, and M. Pic, *DEFACTO: Image and face manipulation dataset*, in European Signal Processing Conference (EUSIPCO), 2019. https://doi.org/10.23919/EUSIPCO.2019.8903181.

[16] T. Nikoukhah, J. Anger, T. Ehret, M. Colom, J.M. Morel, and R. Grompone von Gioi, *JPEG grid detection based on the number of DCT zeros and its application to automatic and localized forgery detection*, in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2019.

[17] T. Nikoukhah, M. Colom, J-M. Morel, and R. Grompone von Gioi, *Local JPEG grid detector via blocking artifacts, a forgery detection tool*, Image Processing On Line, 10 (2020), pp. 24–42. https://doi.org/10.5201/ipol.2020.283.

[18] T. Nikoukhah, R. Grompone von Gioi, M. Colom, and J-M. Morel, *Automatic JPEG grid detection with controlled false alarms, and its image forensic applications*, in IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 2018, pp. 378–382. http://dx.doi.org/10.1109/MIPR.2018.00083.

[19] A. C. Popescu and H. Farid, *Exposing digital forgeries by detecting traces of resampling*, IEEE Transactions on Signal Processing, 53 (2005), pp. 758–767. https://doi.org/10.1109/TSP.2004.839932.

[20] Jean Serra, *Image Analysis and Mathematical Morphology*, Academic Press, 1982.

[21] T.H. Thai, R. Cogranne, F. Retraint, and T-N-C. Doan, *JPEG quantization step estimation and its applications to digital image forensics*, IEEE Transactions on Information Forensics and Security, 12 (2016), pp. 123–133. https://doi.org/10.1109/TIFS.2016.2604208.

[22] A.P. Witkin and J.M. Tenenbaum, *On the role of structure in vision*, in Human and Machine Vision, Elsevier, 1983, pp. 481–543.