



Published in Image Processing On Line on 2021-09-00.
 Submitted on 2021-05-05, accepted on 2021-08-24.
 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.2021.355>

Image Forgeries Detection through Mosaic Analysis: the Intermediate Values Algorithm

Quentin Bammey, Rafael Grompone von Gioi, Jean-Michel Morel

Université Paris-Saclay, ENS Paris-Saclay, Centre Borelli, F-91190 Gif-sur-Yvette, France
 {quentin.bammey, rafael.grompone, jean-michel.morel}@ens-paris-saclay.fr

Abstract

Cameras sample each image pixel in one color channel only. The remaining channels are interpolated from ~~neighbouring~~ neighboring pixels during demosaicing. This operation leaves traces, that can be exploited to authentify images and detect forgeries. This paper describes the method introduced by Choi et al. that exploits the fact that interpolated pixels are more prone to be intermediate values to detect in which pattern an image has been sampled. We then use this information to find regions that are inconsistent with the global image. We attribute a confidence score to each detection, which can then be thresholded to provide a binary map of detected forgeries.

Source Code

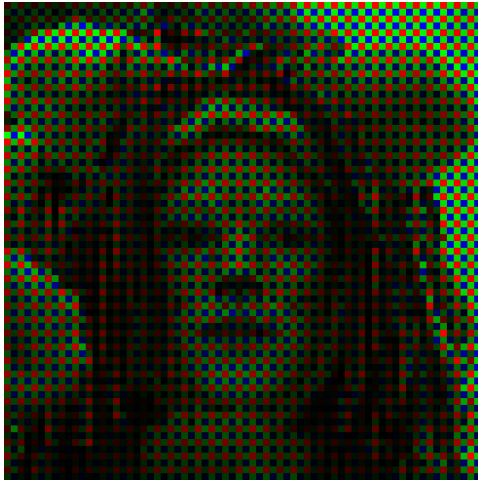
The reviewed source code and documentation for this algorithm are available from [the web page of this article](#)¹. Usage instruction are included in the `README.txt` file of the archive.

Keywords: image forensics; forgery detection; demosaicing; CFA

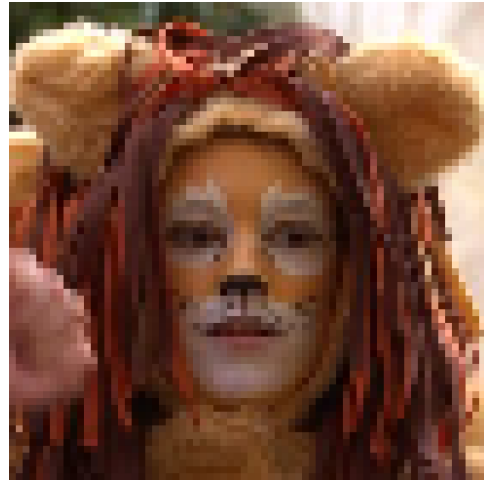
1 Introduction

Most cameras do not see full color images². They only sample one color per pixel using a color filter array (CFA), and must interpolate the missing values, as seen in Figure 1. The most common CFA is the Bayer matrix, shown in Figure 2a. It is made of a simple 2×2 pattern, which samples twice as many pixels in green as in red or blue. Because interpolation leaves traces, it is sometimes possible to identify in which pattern an image has been sampled, in other words, to know which of the four blocks seen in Figure 2b represents the first sampled pixels at the image origin. Other CFA than the Bayer matrix exist, however those are not commonly used; as a consequence this article focuses solely on the Bayer CFA.

If there is a forgery in an image, chances are that the pattern in the manipulated area will not coincide with the rest of the image. For instance, if part of an image is copied and pasted onto another (or potentially the same) image, there is a $\frac{3}{4}$ chance that the patterns will not be aligned, as

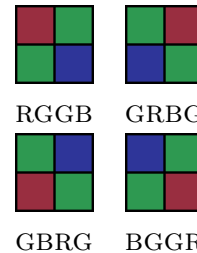
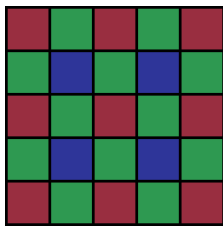


(a) In a raw image, each pixel is only sampled in one color channel.



(b) Demosaicing interpolates the missing colors.

Figure 1: Most cameras can only sample one color channel per pixel, and need to interpolate the missing data with one of many *demosaicing* algorithms. (Here, the mosaic has been artificially recreated on a low-resolution image so it can be seen to the naked eye.)



(a) The Bayer CFA, by far the most commonly used CFA. (b) The four potential patterns that arise from the Bayer CFA.

Figure 2: The Bayer color ~~Filter Array~~filter array (CFA), and the four potential patterns that arise from it. The pattern identifies the position of the top-left red-sampled pixel in an image, in other words the modulo 2 offset of the CFA. With a Bayer CFA, half the pixels are sampled in green, one quarter in red and the other quarter in blue. This is due to the human perception being able to detect finer details in the green values than in red or blue values, hence the need to better reconstruct the green.

can be seen in Figure 3. If this shift in the periodic pattern of sampled colors can be detected, then it will constitute an important evidence as to the presence of a forgery.

The method we present here, originally proposed by Choi et al. [5], uses the fact that demosaicing is basically an interpolation operation. As a consequence, interpolated pixels are more often intermediate values among their immediate neighboursneighbors, as seen in Figure 4. For instance, with the simple bilinear demosaicing, missing colors are directly averaged from the direct neighboursneighbors that were originally sampled in that color, and are thus always intermediate values.

Of course, this simple behaviourbehavior is no longer true with more complex algorithms, which interpolate pixels using more samples among all three channels. Nevertheless, with most algorithms, an interpolated pixel is still more likely to be an intermediate value than a sampled one.

In this paper, we describe, analyze and expand this method. The original article explains how to detect in which pattern an image, or part of it, has been sampled. Starting from there, we detect

¹<https://doi.org/10.5201/ipol.2021.355>

²Some cameras are able to sample full color image by using superposed sensors. Due to the high cost for a very small improvement on the final image quality, these are extremely rare.

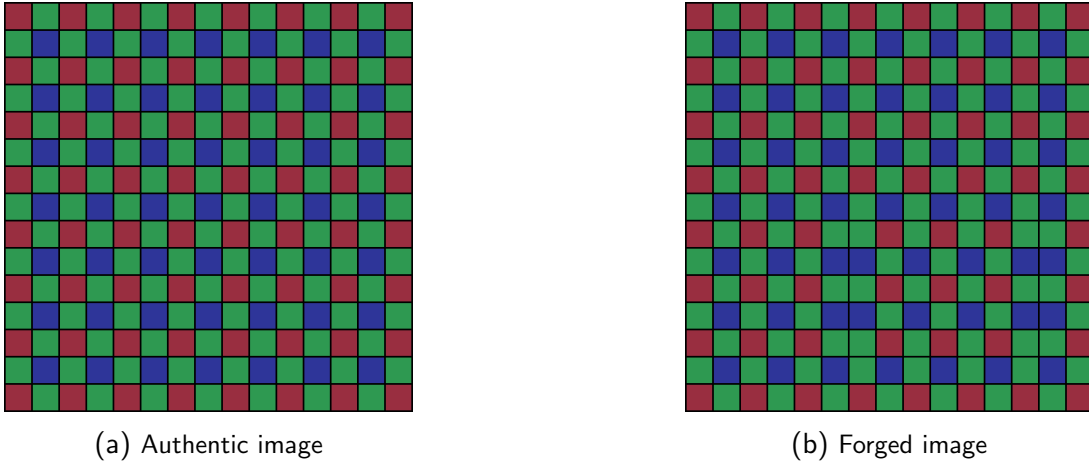


Figure 3: ~~colors~~-Colors in which pixels are sampled in an authentic and forged image. In the forged area of the second image, there is a $\frac{3}{4}$ probability that the patterns of the authentic and forged area are misaligned, causing a shift in the otherwise-periodic CFA.

18	56	94	85	76	96	116	104	139	240	154	16	94	56	72	20
49	56	63	52	41	64	88	87	92	131	168	76	72	94	24	43
80	56	32	19	6	33	60	70	85	24	100	48	102	224	130	72
59	62	66	49	32	59	87	88	60	107	160	68	64	122	200	153
38	69	100	79	58	86	114	106	92	184	125	0	50	0	133	108
40	51	63	74	85	75	66	63	52	155	156	76	136	117	224	127
42	34	26	69	112	65	18	21	146	228	111	12	110	108	107	44
38	47	57	75	94	72	50	35	56	114	48	90	184	141	52	90

Figure 4: Red and green channels of a toy image demosaiced with bilinear interpolation in the R_GB pattern. Red values correspond to positions where the value was interpolated. Highlighted cells correspond to pixels that take an intermediate value, i.e. that are not a local extremum among their direct neighbours-neighbors. While sampled pixels can have intermediate values, many more can be found among interpolated pixels in both the red and green channels. The blue channel, not shown here, behaves similarly to the red one.

which regions of an image are inconsistent with the main image, and attribute a confidence score to this detection. We also propose another way of computing intermediate values, which yields slightly better results.

2 Method

During demosaicing, missing colors on each pixel are interpolated from its neighbours-neighbors. As a consequence, pixels that are interpolated in a given channel are more likely to be an intermediate value, in other words, to be neither lower than all its direct neighbours-neighbors nor higher than all of them. This is especially true with the simplest demosaicing algorithm, the bilinear demosaicing which interpolates the three channels separately.

The detection method analysed-analyzed here counts the intermediate values corresponding to each of the four patterns. On the correct pattern, as most pixels are sampled, there should be fewer intermediate values than in the other patterns.

2.1 Intermediate Values Detection

Let I of shape (X, Y) be one color channel of an image. The pixel at location (x, y) is considered an intermediate value if $\min(I_{x-1,y}, I_{x+1,y}, I_{x,y-1}, I_{x,y+1}) \leq I_{x,y} \leq \max(I_{x-1,y}, I_{x+1,y}, I_{x,y-1}, I_{x,y+1})$. We define $\mathcal{M}(I)$ as the mask of intermediate values of I . Its value is 1 if (x, y) is an intermediate value of I , and 0 otherwise.

If (x, y) is at the border of the image, at least one of $x \pm 1$ and $y \pm 1$ is out of the image boundaries. To avoid border effects, we would thus have to mask out a 1-pixel border around the image. However, doing this would cause an imbalance in the number of pixels corresponding to different patterns, in other words there would be, in border windows, more pixels corresponding to one pattern than to another. To solve the imbalance, we mask out a 2-pixels-wide border instead. More formally, the mask of intermediate values is therefore defined by

$$\mathcal{M}(I)_{x,y} \triangleq \begin{cases} 0 & \text{if } x \in \{0, 1, X-2, X-1\} \text{ or } y \in \{0, 1, Y-2, Y-1\}, \\ 1 & \text{otherwise, if } \min(I_{x-1,y}, I_{x+1,y}, I_{x,y-1}, I_{x,y+1}) \leq I_{x,y} \leq \max(I_{x-1,y}, I_{x+1,y}, I_{x,y-1}, I_{x,y+1}), \\ 0 & \text{otherwise.} \end{cases}$$

The computation of this mask is described in Algorithm 1.

Algorithm 1: Mark intermediate values (original isotropic version)

```

1 function is_intermediate(arr)
   Input arr: Array of size  $(X, Y)$ , one channel of an image
   Output mask: Array of size  $(X-4, Y-4)$ , intermediate values mask
2   mask  $\Leftarrow \mathbf{0}_{(X-4, Y-4)}$ 
3   for x from 2 to  $X-2$  and y from 2 to  $Y-2$  do
4       mi  $\Leftarrow \min(\text{arr}_{x+1,y}, \text{arr}_{x,y-1}, \text{arr}_{x-1,y}, \text{arr}_{x,y+1})$ 
5       ma  $\Leftarrow \max(\text{arr}_{x+1,y}, \text{arr}_{x,y-1}, \text{arr}_{x-1,y}, \text{arr}_{x,y+1})$ 
6       if mi  $\leq \text{arr}_{x,y} \leq \text{ma}$  then
7           mask $x-2, y-2$   $\Leftarrow 1$ 
8   return mask

```

To limit demosaicing artifacts, many demosaicing algorithms tend to avoid interpolating against strong gradients, such as against an edge, and thus often only interpolate in one direction (in which the gradient is smaller). To take this into account, we propose to replace the original isotropic intermediate values mask with bidirectional filters, that separately consider horizontally and vertically intermediate values. We define the mask of horizontal intermediate values as

$$\mathcal{M}(I)_{x,y}^h \triangleq \begin{cases} 0 & \text{if } x \in \{0, 1, X-2, X-1\} \text{ or } y \in \{0, 1, Y-2, Y-1\}, \\ 1 & \text{otherwise, if } \min(I_{x-1,y}, I_{x+1,y}) \leq I_{x,y} \leq \max(I_{x-1,y}, I_{x+1,y}), \\ 0 & \text{otherwise.} \end{cases}$$

Vertical values are computed in a similar way as

$$\mathcal{M}(I)_{x,y}^v \triangleq \begin{cases} 0 & \text{if } x \in \{0, 1, X-2, X-1\} \text{ or } y \in \{0, 1, Y-2, Y-1\}, \\ 1 & \text{otherwise, if } \min(I_{x,y-1}, I_{x,y+1}) \leq I_{x,y} \leq \max(I_{x,y-1}, I_{x,y+1}), \\ 0 & \text{otherwise.} \end{cases}$$

With this definition, the bidirectional mask of intermediate values is then defined as the mean of the horizontal and vertical masks by

$$\mathcal{M}(I)_{x,y} \triangleq \frac{1}{2} (\mathcal{M}(I)_{x,y}^h + \mathcal{M}(I)_{x,y}^v).$$

The mask is therefore null at the border and where a pixel is not an intermediate value, equal to $\frac{1}{2}$ where the pixel is either horizontally or vertically an intermediate value, and equal to 1 when it is an intermediate value both horizontally and vertically. The computation of the bidirectional mask is detailed in Algorithm 2.

Algorithm 2: Mark intermediate values (bidirectional variant)

```

1 function is_intermediate(arr)
    Input arr: Array of size  $(X, Y)$ , one channel of an image
    Output mask: Array of size  $(X - 4, Y - 4)$ , intermediate values mask
2     mask  $\leftarrow \mathbf{0}_{(X-4, Y-4)}$ 
3     for x from 2 to  $X - 2$  and y from 2 to  $Y - 2$  do
4          $m_h \leftarrow \min(\text{arr}_{x-1,y}, \text{arr}_{x+1,y})$ 
5          $M_h \leftarrow \max(\text{arr}_{x-1,y}, \text{arr}_{x+1,y})$ 
6          $m_v \leftarrow \min(\text{arr}_{x,y-1}, \text{arr}_{x,y+1})$ 
7          $M_v \leftarrow \max(\text{arr}_{x,y-1}, \text{arr}_{x,y+1})$ 
8         if  $m_h \leq \text{arr}_{x,y} \leq M_h$  then
9              $\lfloor \text{mask}_{x-2,y-2} \rfloor \leftarrow \frac{1}{2}$ 
10        if  $m_v \leq \text{arr}_{x,y} \leq M_v$  then
11             $\lfloor \text{mask}_{x-2,y-2} \rfloor \leftarrow \frac{1}{2}$ 
12    return mask

```

The original isotropic mask and the bidirectional one will be compared in Section 3. For the rest of this section, we consider R , G and B the masks of intermediate values obtained on the respectively red, green and blue channels of the image. Which of the two methods was used to compute those masks is irrelevant to the rest of the algorithm.

2.2 Division into Windows

The strategy to find forgeries using inconsistencies in the CFA patterns is to first find in which pattern the full image has been demosaiced, then to find the pattern used in different windows of the image. If the pattern detected in a window is different from the one detected for the full image, then this window is inconsistent with the rest of the image and can be considered as forged.

To improve the precision of detection, we do not simply use adjacent windows, but rather sliding windows with overlap. The window size W and stride are set as parameters of the algorithm. The stride determines the number of pixels between the left (or top) border of two consecutive windows, so that a stride equal to the window size leads to adjacent windows without overlapping, a stride equal to half the window size leads to a new window starting at the middle of the previous one, etc.

Using a lower stride will not drastically improve the detection, but may help delineate a detected forgery more precisely, at the cost of a slower algorithm.

2.3 Finding the Pattern

The four Bayer patterns can be divided into two subgroups by their diagonal: RGGB and BGGR share the $\cdot GG \cdot$ diagonal, whereas GRBG and GBRG share the $G \cdot \cdot G$ diagonal (see Figure 5). Because the Bayer CFA samples twice as many pixels in green than in red or blue, it is easier to find information on the pattern in the green channel. This is amplified by the fact that many demosaicing algorithms first

interpolate the green channel by itself, but interpolate the red and blue channels using information from the green channel.

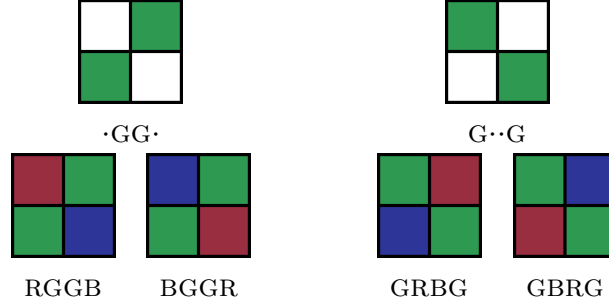


Figure 5: The four possible sampling patterns can be grouped by the diagonal on which the green channel was sampled: RGGB and BGGR share the $\cdot GG \cdot$ diagonal, whereas GRBG and GBRG share the $G \cdot \cdot G$ one.

As a consequence, the presented method first tries to detect the diagonal pattern using the green channel ($\cdot GG \cdot$ or $G \cdot \cdot G$), then uses the red and blue channels to compare the two potential patterns sharing that diagonal. We denote by R , G and B be the masks of intermediate values on the respectively red, green and blue channels. (They will not be confused with the R, G, B channels that we no longer use in the rest of this paper). These masks can represent either the full image or a window of it. To maintain the balance between patterns, the masks must be of even size. For this reason, the window size must be even, and the last row/column of the full image is removed if necessary to ensure the evenness of the shape. Here we denote the shape of these masks $(2X, 2Y)$ for easier notations of the different positions on the CFA. We start by looking at the green channel for the diagonal grids. The intermediate value count corresponding to the $\cdot GG \cdot$ pattern is

$$C_{\cdot GG \cdot} \triangleq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (G_{2x+1, 2y} + G_{2x, 2y+1}),$$

while the count corresponding to the $G \cdot \cdot G$ pattern is

$$C_{G \cdot \cdot G} \triangleq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (G_{2x, 2y} + G_{2x+1, 2y+1}).$$

The count difference of the diagonal is then defined as

$$\Delta_{\text{diag}} \triangleq \frac{1}{2X \cdot Y} (C_{\cdot GG \cdot} - C_{G \cdot \cdot G}).$$

This difference is positive if the detected diagonal is $G \cdot \cdot G$, and negative if it is $\cdot GG \cdot$:

$$D \triangleq \begin{cases} G \cdot \cdot G & \Delta_{\text{diag}} > 0, \\ \cdot GG \cdot & \Delta_{\text{diag}} < 0, \\ -1 & \Delta_{\text{diag}} = 0. \end{cases}$$

The normalization by $\frac{1}{2XY}$ means that the resulting difference belongs to $[-1, 1]$, and is equal to ± 1 if all pixels in one of the patterns are intermediate values, whereas the other pattern has no intermediate values (XY is the number of 2×2 blocks in a mask of shape $(2X, 2Y)$, and we sum two pixels in this block for each pattern). Note that the ± 1 limit is only theoretical: even with bilinear demosaicing, where all interpolated pixels are intermediate values, sampled pixels can be intermediate too, for instance where they belong to a slope. As a consequence, the difference will not reach those values in natural cases.

Once we know the main diagonal, we can compare the two patterns sharing that diagonal. The green channel does not provide any information on this, so we use the red and blue channels. The count of intermediate values corresponding to each pattern is

$$\begin{aligned} C_{\text{RGGB}} &\triangleq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (R_{2x,2y} + B_{2x+1,2y+1}), \\ C_{\text{BGGR}} &\triangleq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (R_{2x+1,2y+1} + B_{2x,2y}), \\ C_{\text{GRBG}} &\triangleq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (R_{2x+1,2y} + B_{2x,2y+1}), \\ C_{\text{GBRG}} &\triangleq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} (R_{2x,2y+1} + B_{2x+1,2y}). \end{aligned}$$

The count differences of the two pattern pairs are then defined as

$$\begin{aligned} \Delta_{\text{RGGB-BGGR}} &\triangleq \frac{1}{2XY} (C_{\text{RGGB}} - C_{\text{BGGR}}), \\ \Delta_{\text{GRBG-GBRG}} &\triangleq \frac{1}{2XY} (C_{\text{GRBG}} - C_{\text{GBRG}}), \end{aligned}$$

and are then combined into the main grid difference

$$\Delta_{\text{main}} \triangleq \begin{cases} \Delta_{\text{RGGB-BGGR}} & D = \cdot\text{GG}\cdot, \\ \Delta_{\text{GRBG-GBRG}} & D = \text{G}\cdot\text{G}. \end{cases}$$

Finally, the main detected grid can be obtained as

$$M \triangleq \begin{cases} \text{RGGB} & D = \cdot\text{GG}\cdot \text{ and } \Delta_{\text{main}} < 0, \\ \text{BGGR} & D = \cdot\text{GG}\cdot \text{ and } \Delta_{\text{main}} > 0, \\ \text{GRBG} & D = \text{G}\cdot\text{G} \text{ and } \Delta_{\text{main}} < 0, \\ \text{GBRG} & D = \text{G}\cdot\text{G} \text{ and } \Delta_{\text{main}} > 0, \\ -1 & \text{otherwise.} \end{cases}$$

Both for the diagonal and main grids, if there is strict equality in the two counts detected, no grid is considered detected. Naturally, if no decision is taken on the diagonal, no main grid is selected either. The grid detection is detailed in Algorithm 3.

While Δ_{main} is later used to make decisions on forgeries, the two intermediary comparisons $\Delta_{\text{RGGB-BGGR}}$ and $\Delta_{\text{GRBG-GBRG}}$ are easier to understand visually, and are thus kept for visualization.

Our implementation of the count difference computation is slightly different from the description of the original article. In the original article, the difference is not **normalised-normalized** by $\frac{1}{2XY}$. More importantly, the difference is computed separately in the red and blue channels, and the strongest of the two is kept, whereas we use their sum. The reason for this is that the original article only tries to classify in which pattern an image has been sampled, without considering how confident one can be in the detection, or how to use it to detect forgeries. When only considering classification of an image or window into the four patterns, both the original article and our implementation provide the same results. However, adding the normalization and summing the two channels makes it easier for us to also compute a confidence value for the detections, which will be described in the next subsection.

Finally, we note that even though this algorithm is presented for one window, the grid detection is obviously performed on all windows simultaneously.

2.4 Forgery Detection

Using the previously-described algorithms, we can compute the intermediate value masks in all channels, cut them into windows, and detect the diagonal and pattern of the global image and of each window. With this information, we could simply say that the windows which do not use the same pattern as the main grid correspond to forged regions. However, doing this creates many false positives, as the detection is not always correct. In first instance, if the grid of a window does not

Algorithm 3: Find the grid

```

1 function find_grid(R, G, B)
    Input R: Array of even size ( $2X, 2Y$ ), typically as returned by is_intermediate or a sub-window of it on the red channel
    Input G: Same as above for the green channel
    Input B: Same as above for the blue channel
    Output M: CFA pattern identified by the function (one of RGGB, GRBG, GBRG, BGGR)
    Output D: Diagonal pattern identified by the function (either  $\cdot GG \cdot$  or  $G \cdot G$ )
    Output  $\Delta_{\text{main}}$ : Difference of count of intermediate values between the two patterns sharing the same diagonal. Positive if the best pattern is RGGB or GRBG, negative if the best pattern is BGGR or GBRG.
    Output  $\Delta_{\text{diag}}$ : Difference of count of intermediate values between the two diagonal patterns. Negative for  $\cdot GG \cdot$ , Positive for  $G \cdot G$ .
    Output  $\Delta_{\text{RGGB-BGGR}}, \Delta_{\text{GRBG-GBRG}}$ : Difference of count of intermediate values between two grids sharing the same pattern.

    # First we select the best diagonal pattern using the green values
2   C
3    $\doteq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} G_{2x,2y+1} + G_{2x+1,2y}$ 
5 else
6    $D \doteq G \cdot G$ 

    # Compare patterns with the same diagonal.
7    $C_{\text{RGGB}} \doteq \sum_{x=0}^X \sum_{y=0}^Y R_{2x,2y} + B_{2x+1,2y+1}$   $C_{\text{RGGB}} \doteq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} R_{2x,2y} + B_{2x+1,2y+1}$ 
8    $C_{\text{BGGR}} \doteq \sum_{x=0}^X \sum_{y=0}^Y R_{2x+1,2y+1} + B_{2x,2y}$   $C_{\text{BGGR}} \doteq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} R_{2x+1,2y+1} + B_{2x,2y}$ 
9    $C_{\text{GRBG}} \doteq \sum_{x=0}^{\frac{X}{2}} R_{2x+1,2y} + B_{2x,2y+1}$   $C_{\text{GRBG}} \doteq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} R_{2x+1,2y} + B_{2x,2y+1}$ 
10   $C_{\text{GBRG}} \doteq \sum_{x=0}^{\frac{X}{2}} R_{2x,2y+1} + B_{2x+1,2y}$   $C_{\text{GBRG}} \doteq \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} R_{2x,2y+1} + B_{2x+1,2y}$ 
11   $\Delta_{\text{RGGB-BGGR}} = \frac{1}{2XY} (C_{\text{RGGB}} - C_{\text{BGGR}})$ 
12   $\Delta_{\text{GRBG-GBRG}} = \frac{1}{2XY} (C_{\text{GRBG}} - C_{\text{GBRG}})$ 
13  if  $D = \cdot GG \cdot$  then
14       $\Delta_{\text{main}} = \Delta_{\text{RGGB-BGGR}}$ 
15      if  $\Delta_{\text{main}} < 0$  then
16           $M \doteq \text{RGGB}$ 
17      else
18           $M \doteq \text{BGGR}$ 
19 else
20       $\Delta_{\text{main}} = \Delta_{\text{GRBG-GBRG}}$ 
21      if  $\Delta_{\text{main}} < 0$  then
22           $M \doteq \text{GRBG}$ 
23      else
24           $M \doteq \text{GBRG}$ 
25 return M, D,  $\Delta_{\text{main}}$ ,  $\Delta_{\text{diag}}$ ,  $\Delta_{\text{RGGB-BGGR}}$ ,  $\Delta_{\text{GRBG-GBRG}}$ 
    
```

match the global image’s grid, we can consider that window as forged with a confidence of $|\Delta_{\text{main}}|$ (or $|\Delta_{\text{diag}}|$ if looking at the diagonals). However, if the threshold is low, isolated detections of a given grid will be made by mistake. On the contrary, in a region with many windows sharing the same grid, only those above the threshold will be detected, so a high threshold will cause most of the detections to be missed. In both cases, using a fixed threshold will lead to mistakes that would be easy to avoid by looking at the map more globally.

We therefore propose to segment the windows into connected components by their grids. In other words, a connected component is a set of spatially connected windows whose detected pattern is the same. This segmentation is performed with `scikit-image` [10]. Components whose detected pattern is equal to the one of the global image are immediately discarded; they are not considered forged as they agree with the full image. For components whose detected pattern is different, we consider them as forged, with a confidence value which corresponds to the maximum absolute difference of count of all windows in that components (either $|\Delta_{\text{main}}|$ or $|\Delta_{\text{diag}}|$ depending on whether we are looking at the full pattern or the diagonal). In other words, the confidence of a component is the confidence of its most prominent window. The computation of the confidence by connected component is performed in Algorithm 4.

Algorithm 4: Connected confidence computation

```

1 function connected_confidence(G, global_G,  $\Delta$ )
    Input G: Grid/diagonal detected on each window, shape ( $X_W, Y_W$ )
    Input global_G: Grid/diagonal detected on the main image
    Input  $\Delta$ : Either  $\Delta_{\text{main}}$  or  $\Delta_{\text{diag}}$ 
    Output confidence: Confidence that each pixel is forged
2     labels = label_connected(G, global_G)
3     confidence =  $\mathbf{0}_{X_W, Y_W}$ 
4     for label from 0 to max(labels) do
        #  $\odot$  denotes Hadamard product
5         confidence += max((labels = label)  $\odot$   $|\Delta|$ )
6     return confidence
    
```

We apply this method separately to both the diagonal and the full pattern detection, ~~which; this~~ yields two confidence maps. ~~We merge those two confidence maps into one, which are then merged~~ by taking their pointwise maximum. Although the full pattern analysis can encompass the diagonal detection, in many cases the algorithm can only find the diagonal but hesitates on the full pattern. Hence, separating the detections enables the method to detect significant diagonal traces even when the full pattern cannot be detected.

These confidence maps are useful to visualize the detection. However, they do not constitute by themselves a decision on the detection. They cannot either be used as a heat map: the maximal absolute value of the difference, 1, is never reached in actual cases, and even the most confident detections will rarely reach a score of 0.3.

To make a final decision on the image, we thus threshold the obtained confidence map by a ~~given~~ fixed threshold γ . This is equivalent to performing hysteresis thresholding with a lower threshold 0 and a higher threshold γ on each map ($M = g$) \odot $|\Delta_{\text{main}}|$ for each pattern g except the full image’s pattern, and $(D \neq d_{\text{img}}) \odot |\Delta_{\text{diag}}|$, where d_{img} is the full image’s detected diagonal, \odot is the Hadamard product (pointwise multiplication), while the expression $(A = b)$ is an array of the same shape of A , equal to 1 where A takes the value b and 0 elsewhere.

Finally, all the outputs are resized to have one value per pixel, rather than per window. This is

done with nearest ~~neighbours~~ neighbors interpolation for binary outputs, and with linear interpolation for continuous outputs. The computation of the forgery map is detailed in Algorithm 5.

Algorithm 5: Global algorithm

```

1 function find_forgeries(img, W, stride, threshold)
    Input img: Input image, size  $(X, Y, 3)$ .  $X$  and  $Y$  must be even (the last row and/or
        column may be cut to ensure this).
    Param W: int, Window size
    Param stride: int, Distance between the left/top border of two consecutive windows.
        Must divide  $W$ .
    Param  $\gamma$ : float, higher hysteresis threshold to select relevant inconsistencies.
    Output forged_full: Final map of detected forgeries (pointwise maximum of forged_main
        and forged_diag)
    Output forged_{main, diag}: Detected forgeries after thresholding, respectively on the
        full pattern and on the diagonal
    Output confidence_{full, main, diag}: Confidence that each region is a forgery
    Output inconsistent_{full, main, diag}_raw: Binary mask of each region being
        inconsistent with the global image, regardless
        of significance.

2 intermediate  $\coloneqq$  is_intermediate(img)
3 windows  $\coloneqq$  create_sliding_windows(intermediate, W, stride)
4  $X_w, Y_w$   $\coloneqq$  number of windows per column/row
   # Pattern and diagonal on the global image
5 global_M, global_D,  $\rightarrow$ ,  $\rightarrow$ ,  $\rightarrow$  = find_grid(intermediate[:, :, 0], intermediate[:, :
    , 1], intermediate[:, :, 2])
   # Pattern and diagonal on each window
6 M, D,  $\Delta_{\text{main}}$ ,  $\Delta_{\text{diag}}$   $\coloneqq$   $\mathbf{0}_{X_w, Y_w}$ 
7 for  $x$  from 0 to  $X_w$  and  $y$  from 0 to  $Y_w$  do
8      $\lfloor$  main $x,y$ , diag $x,y$ ,  $\Delta_{\text{main}}$  $x,y$ ,  $\Delta_{\text{diag}}$  $x,y$   $\coloneqq$  find_grid(windows $x,y,0$ , windows $x,y,1$ , windows $x,y,2$ )
   # Inconsistent regions
9 bad_{main, diag}_raw  $\coloneqq$   $\{M, D\} \neq$  global_{main, diag}
10 bad_full_raw  $\coloneqq$   $\max(\text{bad\_diag\_raw}, \text{bad\_main\_raw})$ 
   # Connected confidence
11 confidence_main  $\coloneqq$  connected_confidence(M, global_M,  $\Delta_{\text{main}}$ )
12 confidence_diag  $\coloneqq$  connected_confidence(D, global_D,  $\Delta_{\text{diag}}$ )
13 confidence_full  $\coloneqq$   $\max(\text{confidence\_main}, \text{confidence\_diag})$ 
   # Threshold
14 forged_{full, main, diag}  $\coloneqq$  confidence_{full, main, diag}  $>$   $\gamma$ 
15 return forged_{full, main, diag}, confidence_{full, main, diag}, inconsistent_{full, main,
    diag}_raw

```

Overall, the full algorithm can achieve linear complexity in the input size. Indeed, each individual step is linear, including the connected confidence computation since each block of the image belongs to at most one component and is thus only processed once. In practice, the vectorized Python implementations processes all blocks for each component, thus leading to a quadratic complexity. Although an optimal computation in another language could offer the optimal worst-case linear complexity, this is largely irrelevant since the number of inconsistent connected components usually does not scale linearly with the image size.

3 Experiments

To evaluate the ability of this method to detect the CFA pattern correctly, we take 15 images from the Raise dataset [6], and demosaic them using the 7 algorithms available in LibRaw: Bilinear interpolation, AAHD, AHD, DCB, DHT, PPG and VNG. Eleven of these images are of size 4948×3280 , the other 4 are of size 4310×2868 . The selected images can be seen in Figure 6.

3.1 CFA Pattern Detection

We start by analyzing, at a global scale, whether the method is able to detect the correct pattern of the 15 images described above. Results can be seen in Table 1. One can see that the algorithm detects the correct grid in all 15 images when they are demosaiced with bilinear, AHD or DCB demosaicing. It also works well on the PPG and VNG algorithms, despite a few mistakes in the full pattern identification against PPG or VNG-demosaiced images. These mistakes are solved when using bidirectional filters. When the image is demosaiced with AAHD or DHT, however, the algorithm consistently fails to detect even the diagonal, and consequently also fails on the full pattern, in both versions of the algorithm.

Demosaicking <u>Demosaicking</u>	Diagonal	Full pattern	Demosaicking <u>Demosaicking</u>	Diagonal	Full pattern
AAHD	0/15	0/15	AAHD	0/15	0/15
AHD	15/15	15/15	AHD	15/15	15/15
DCB	15/15	15/15	DCB	15/15	15/15
DHT	3/15	3/15	DHT	2/15	2/15
Bilinear	15/15	15/15	Bilinear	15/15	15/15
PPG	15/15	13/15	PPG	15/15	15/15
VNG	15/15	14/15	VNG	15/15	15/15

(a) Original isotropic intermediate values

(b) Bidirectional filters for intermediate values

Table 1: Identification of the main diagonal and of the full pattern on the 15 images. For each demosaicing algorithm, we show how many of the 15 images had their diagonal/full pattern correctly detected by the method. In its original version, the algorithm works very well when the demosaicing is done with AHD, DCB or ~~Bilinear~~ bilinear demosaicing, with a few errors on the full pattern against PPG- or VNG-demosaiced images. It fails to detect even the diagonal on AAHD- and DHT-demosaiced images. Bidirectional filters for intermediate value computation yield perfect results on PPG and VNG, but still fail against AAHD- and DHT-demosaiced images.

Looking at the results on ~~image_r0a2ff882t~~ Image_r0a2ff882t in Figure 7, we can see again that the results depend on the demosaicing algorithm used by the method. All windows are detected correctly against DCB and ~~Bilinear~~ bilinear demosaicing, but the algorithm is confused on the diagonal pattern in the PPG-demosaiced image, though bidirectional filters for the intermediate value computation partly alleviate this problem. On the VNG-demosaiced image, there are also false detections on the diagonal itself. More importantly, the basket of the bike causes errors with AHD, PPG and VNG demosaicing. This was to be expected with a periodic structure that fools the detection. The result might easily be misinterpreted as a forgery. As can be seen on Figure 8, however, using bidirectional filters yields a very low relative confidence for the identification of the basket’s grid compared to the rest of the image. As a consequence, a reasonable thresholding level should still enable one to automatically discard this false detection.

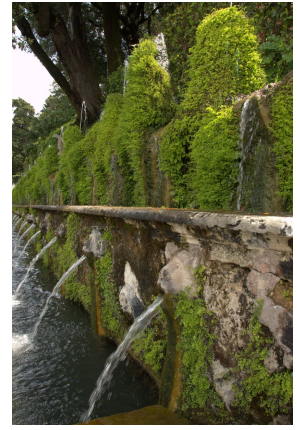
Most images that are found on the web are JPEG-compressed. It is thus vital to test the robustness of this algorithm to JPEG compression. JPEG compression quickly discards the highest frequencies, at which CFA artifacts are located. As a consequence, it would be illusory to expect



(a) r002fc3e2t



(b) r1ead3024t



(c) r1ceba29dt



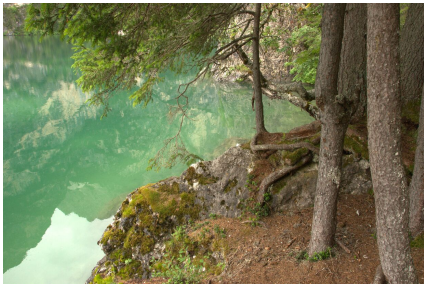
(d) r0a2ff882t



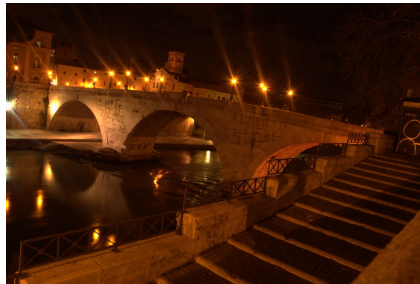
(e) r0a808003t



(f) r0a966704t



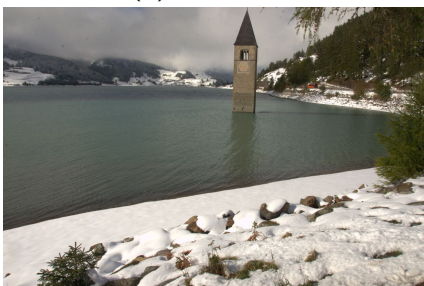
(g) r0e04cc91t



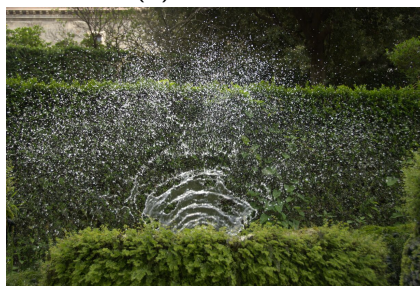
(h) r0ea0825ft



(i) r1a0f5585t



(j) r1c9fdcf4t



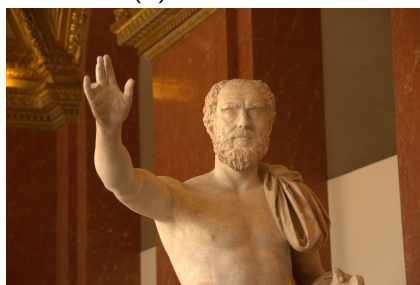
(k) r06aa7dabt



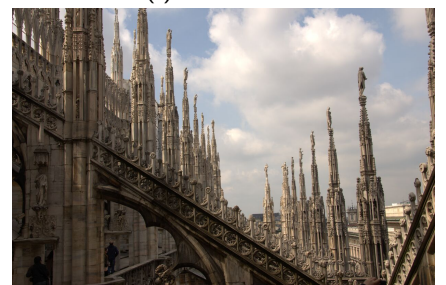
(l) r07cfb432t



(m) r07ffdc87t

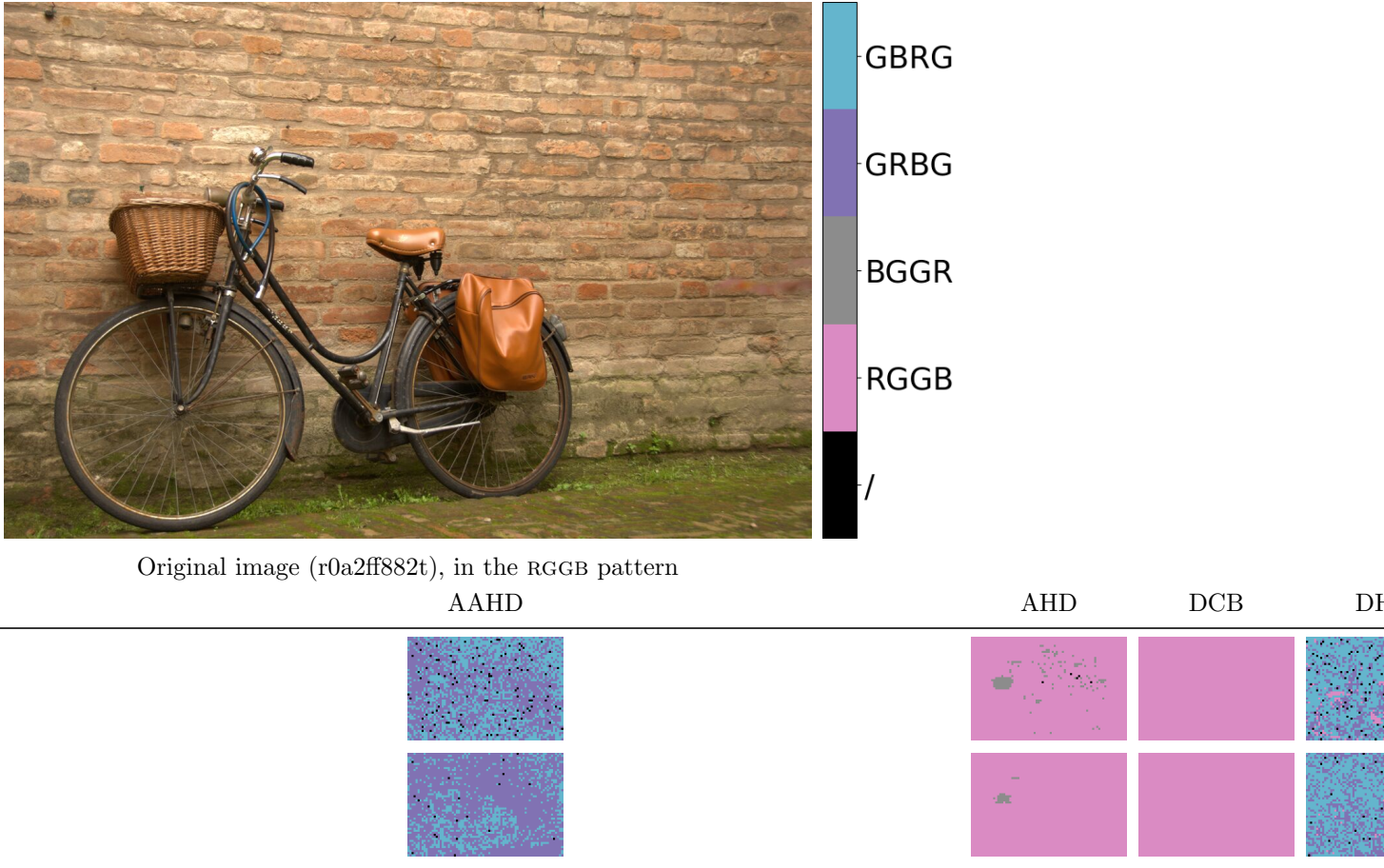


(n) r16da5576t



(o) r191f3cdet

Figure 6: These 15 images from the Raise dataset [6] were used in our experiments.



Original image (r0a2ff882t), in the RGGB pattern

Figure 7: Results of the method on 64×64 windows, both with the original isotropic intermediate value mask and the proposed bidirectional one, on one image with the 7 different demosaicing algorithms. Both methods work perfectly on the DCB- and ~~Bilinear-demosaiced~~-~~bilinear-demosaiced~~ images. With the AHD, PPG and VNG methods, both the original isotropic and the bidirectional filters have trouble discerning between the two patterns sharing the same diagonal, but the bidirectional detection makes fewer mistakes. Periodically textured regions like the basket can create a localized shift in the detected mosaic, which could be mistaken for a forgery. With the AAHD and DHT algorithm, the method consistently detects the wrong diagonal.

results on heavily-compressed images. However, being able to detect the CFA pattern on low-compression images extends the application range of a CFA grid detection method. We show results after JPEG compression on Figures 9 and 10. JPEG compression is done with the Pillow library³. On the two studied images, we can see that even the highest-quality compression of 100 causes many errors in the pattern detection, though the algorithm remains largely usable, especially when only looking at the diagonal. A quality factor of 100 does not specifically remove the high frequencies, however the discretization in the frequency domain already includes a loss of information. At JPEG quality 98, the algorithm no longer detects the correct pattern, except in the easier case of the bilinear demosaicing algorithm. However, it can still detect the diagonal of most windows, albeit with a few errors. Finally, at JPEG quality 95, the algorithm is unable to find anything.

All in all, JPEG compression remains the biggest limitation of this method, and of CFA detection in general.

In Figures 11, 12 and 13, we evaluate the robustness of the method to additive white Gaussian noise (AWGN). Because AWGN is not spatially correlated, it remains possible to detect the pattern in most cases with a noise of standard deviation $\sigma = 5$ (on $[0, 255]$ -ranged images). More localized

³Alex Clark, Pillow (PIL fork) documentation <https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf>

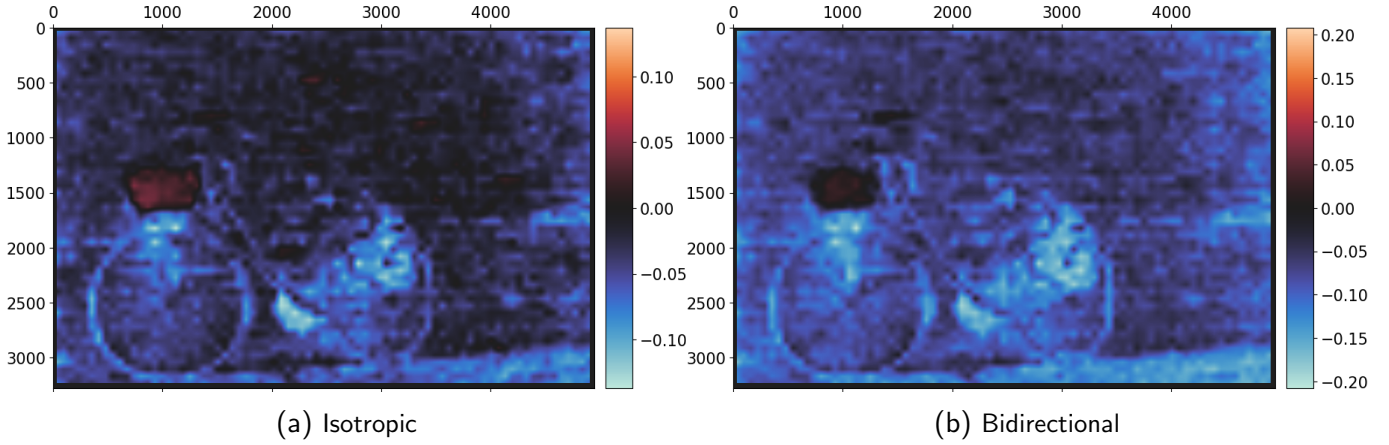


Figure 8: This figure shows, on the AHD-demosaiced bicycle image, the difference of counts of intermediate values corresponding to the RGG and BGG patterns, on the red and blue channels. This count is what is used by the algorithm to decide on a grid. A negative difference corresponds to the correct RGG pattern, a positive difference to the incorrect BGG pattern. The difference is normalized by dividing it by the size of the block (64×64). The texture in the basket area leads to a locally consistent shift in the position of the intermediate values. The error is slightly less prominent when a bidirectional mask is used, but is still consistently in ~~favour~~ favor of the wrong grid.

errors are made as the noise level increases, but thanks to the lack of spatial correlation of the noise (and consequently of the errors), the risk of mistakenly interpreting these as forgeries remains relatively low. Finally, we can see in Figure 13 that detecting the pattern over AWGN is made easier by using a larger window size, which averages the noise while keeping the artifacts. Of course, this comes at the price of potentially missing smaller forgeries.

Median filtering has often been proposed as a counter-forensics measure to hide forgeries. Although it can be easily detected [8], we evaluate the robustness of the presented method to median filtering in Figure 14, using a median filter of footprint $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$. We can see that the results of the method are completely inverted, because median filtering shifts the intermediate values. As a consequence, images on which the correct diagonal was found before filtering now yield wrong detection, whereas the method finds the correct pattern on AAHD- and DHT-demosaiced images, where it was failing without median filtering. Figure 15 explains this phenomenon with a toy example. Without further elaborating, we note that only the diagonal detection is affected. As a consequence, if median filtering has been detected, detections can be made correct again by simply reverting the detected diagonal.

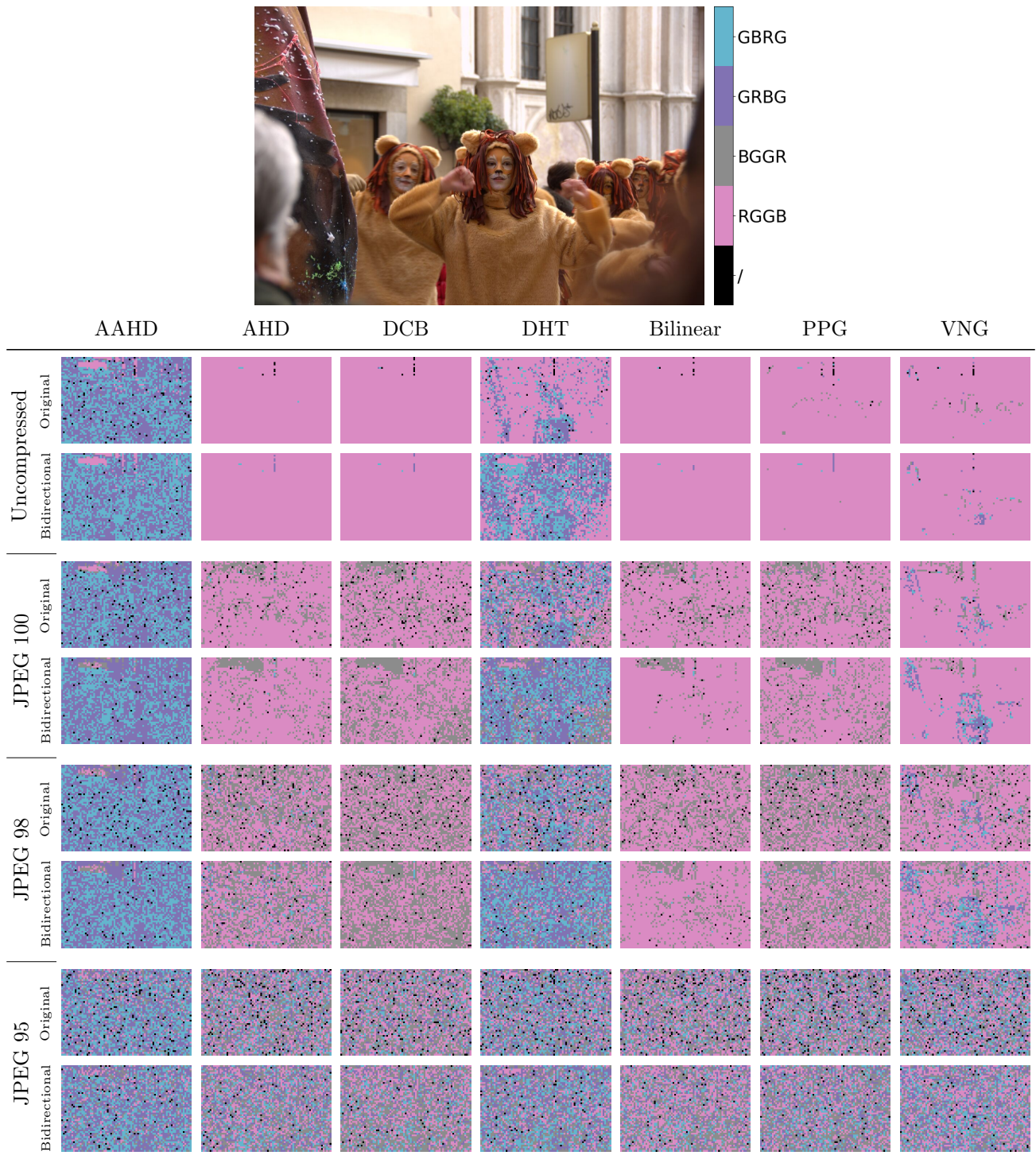


Figure 9: Detection of the method after JPEG compression. Results are shown on [image_r07ffdc87t](#) in [the](#) RGGG pattern, uncompressed and submitted to JPEG compression of quality 100, 98 and 95. At JPEG quality 100 (the highest possible), although the correct pattern is usually found in most blocks of the image, errors between the two dual patterns start to appear. At JPEG quality 98, the method remains globally able to detect the main diagonal, but cannot distinguish the dual patterns anymore. At JPEG quality 95, the algorithm is unable to do any detection, even against bilinear demosaicing. Bidirectional intermediates provide a small boost to JPEG robustness, though it is not enough to make the algorithm reliable to use on JPEG-compressed images.

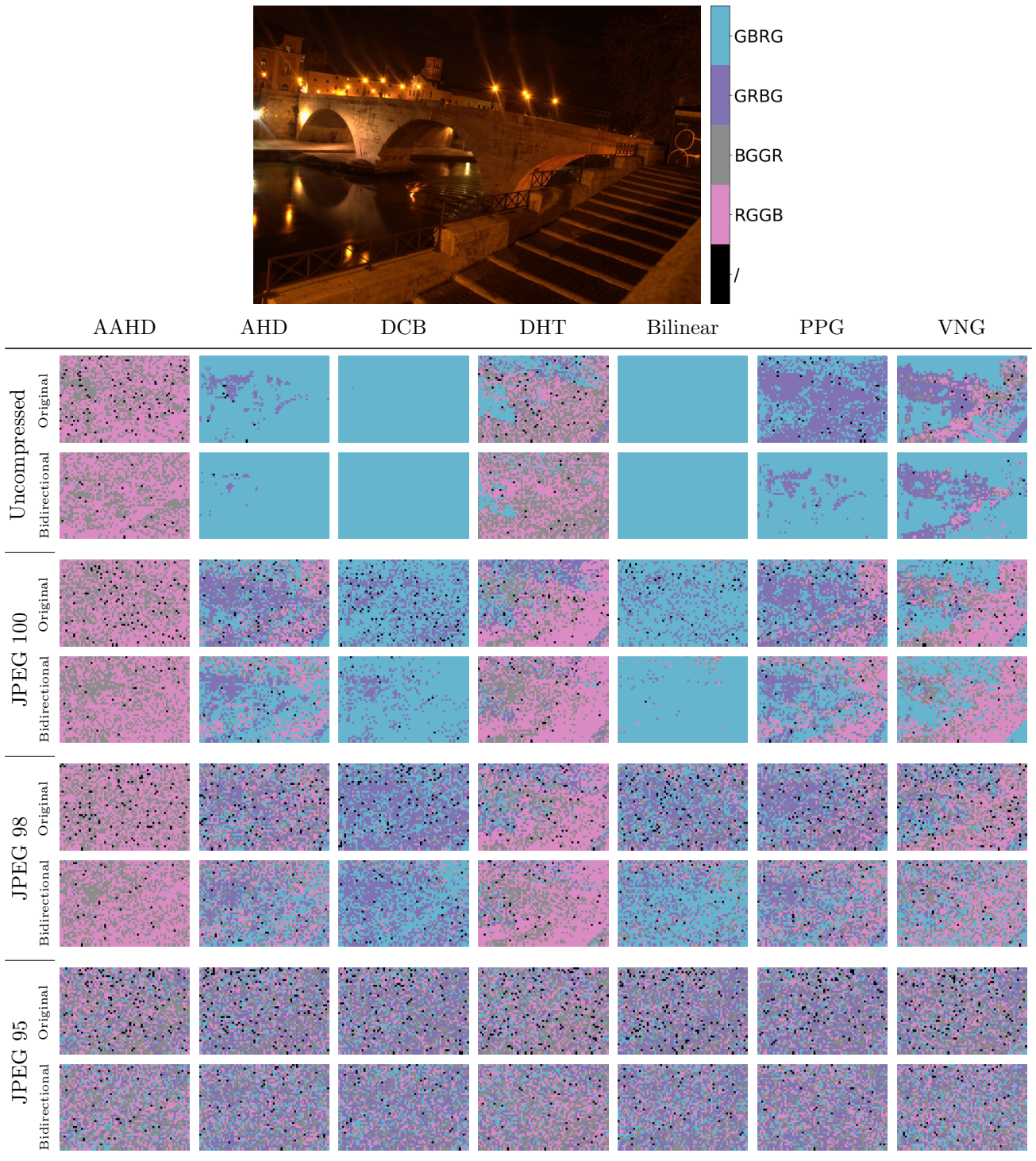


Figure 10: Detection of the method after JPEG compression. Results are shown on [image_r0ea0825ft](#) in the [GRBG](#) pattern, uncompressed and submitted to JPEG compression of quality 100, 98 and 95. On this image, which is more difficult to analyze than the one in Figure 9, errors are already present in the uncompressed image, the diagonal is also locally wrong on the stairs against VNG demosaicing, especially with the original isotropic intermediate values. These errors become more prominent against other demosaicing methods as well at JPEG quality 100 (highest possible), and detection becomes barely possible. At JPEG quality 98, contrarily to Figure 9, detection is mostly impossible, although the diagonal can still be found with local mistakes against bilinear and DCB demosaicing if bidirectional filters are used. Again, bidirectional intermediates provide a consistent, although small, boost to JPEG robustness.

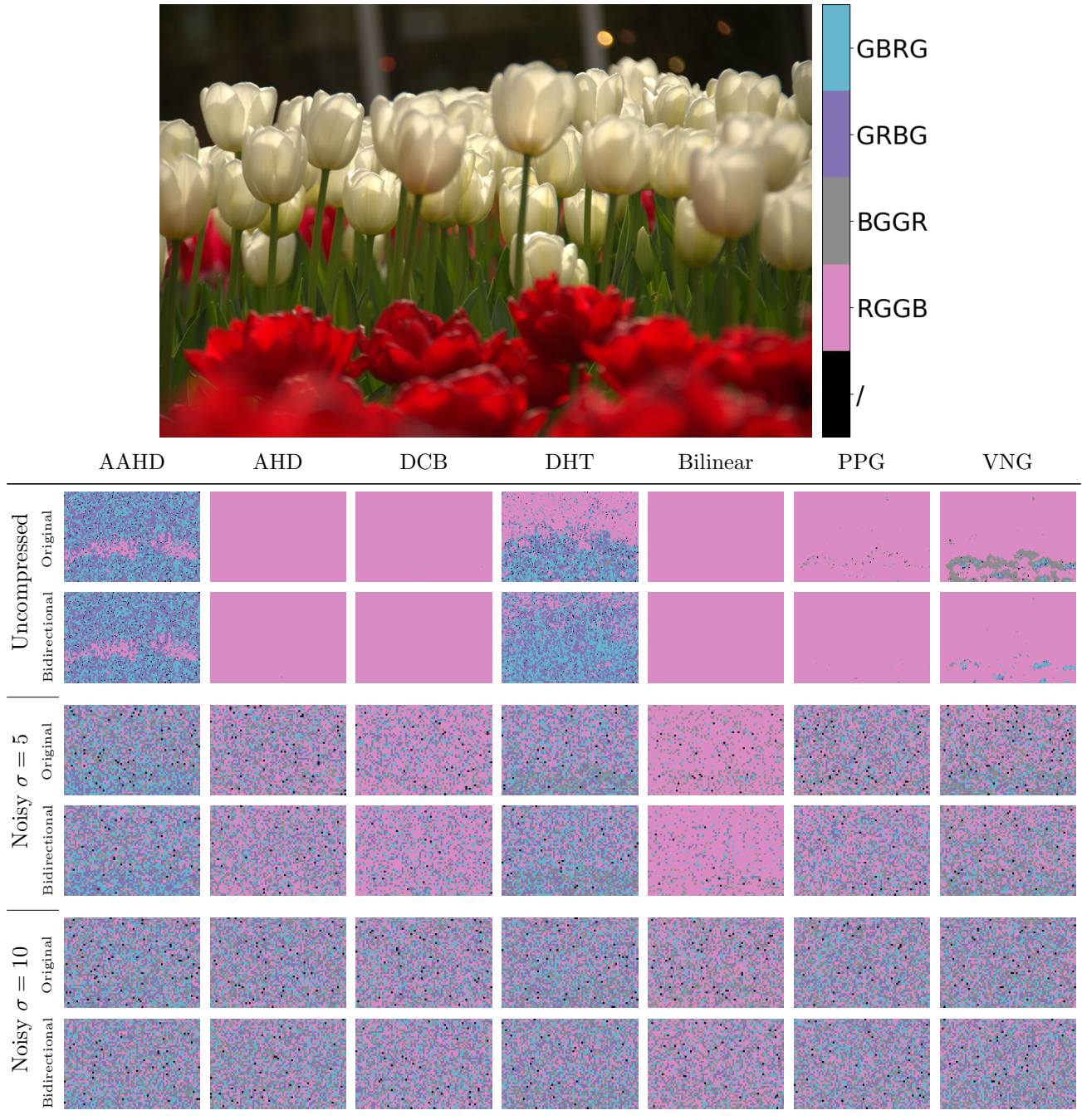


Figure 11: Robustness of the method to additive white Gaussian noise (AWGN), that can be added to images either for aesthetic reasons or to maliciously hide manipulations. Image r07cfb432t, in the RGGG pattern. We show results against noise of standard deviation from 0 (noiseless) to 10, window size 64×64 . Because the noise is independent of the image, it does not create locally coherent errors that can hardly be distinguished from forgeries. However, the probabilities of a sampled or interpolated pixel being an intermediate value go closer to one another as more noise is added, making the detection harder.

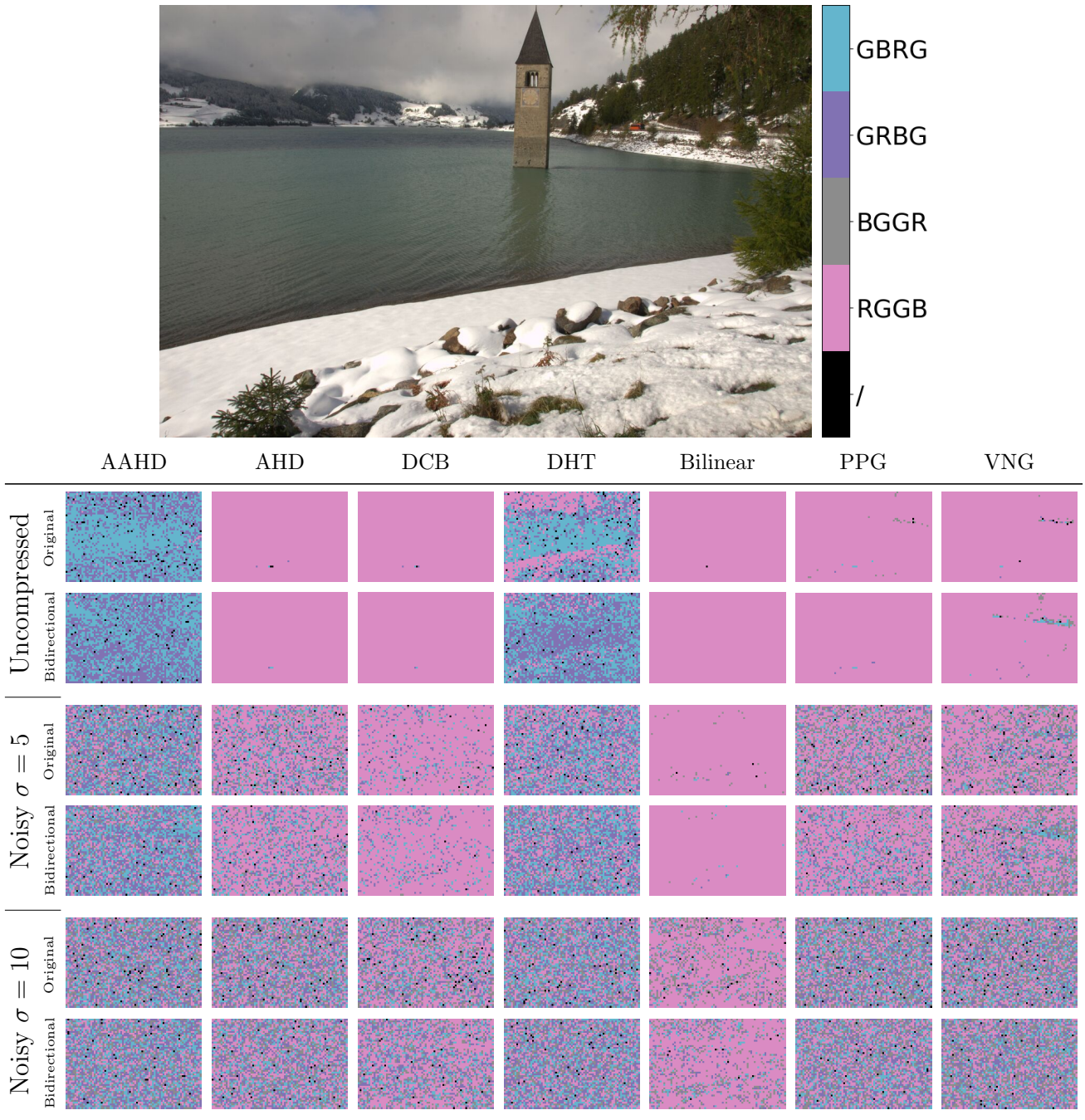


Figure 12: Robustness of the method to additive white Gaussian noise (AWGN), that can be added to images either for aesthetic reasons or to maliciously hide manipulations. Image r1c9fdcf4t, in the RGGG pattern. We show results against noise of standard deviation from 0 (noiseless) to 10, with window size 64×64 . Because the noise is independent to-of the image, it does not create locally coherent errors that can hardly be distinguished from forgeries. However, the probabilities of a sampled or interpolated pixel being an intermediate value go closer to one another as more noise is added, making the detection harder.

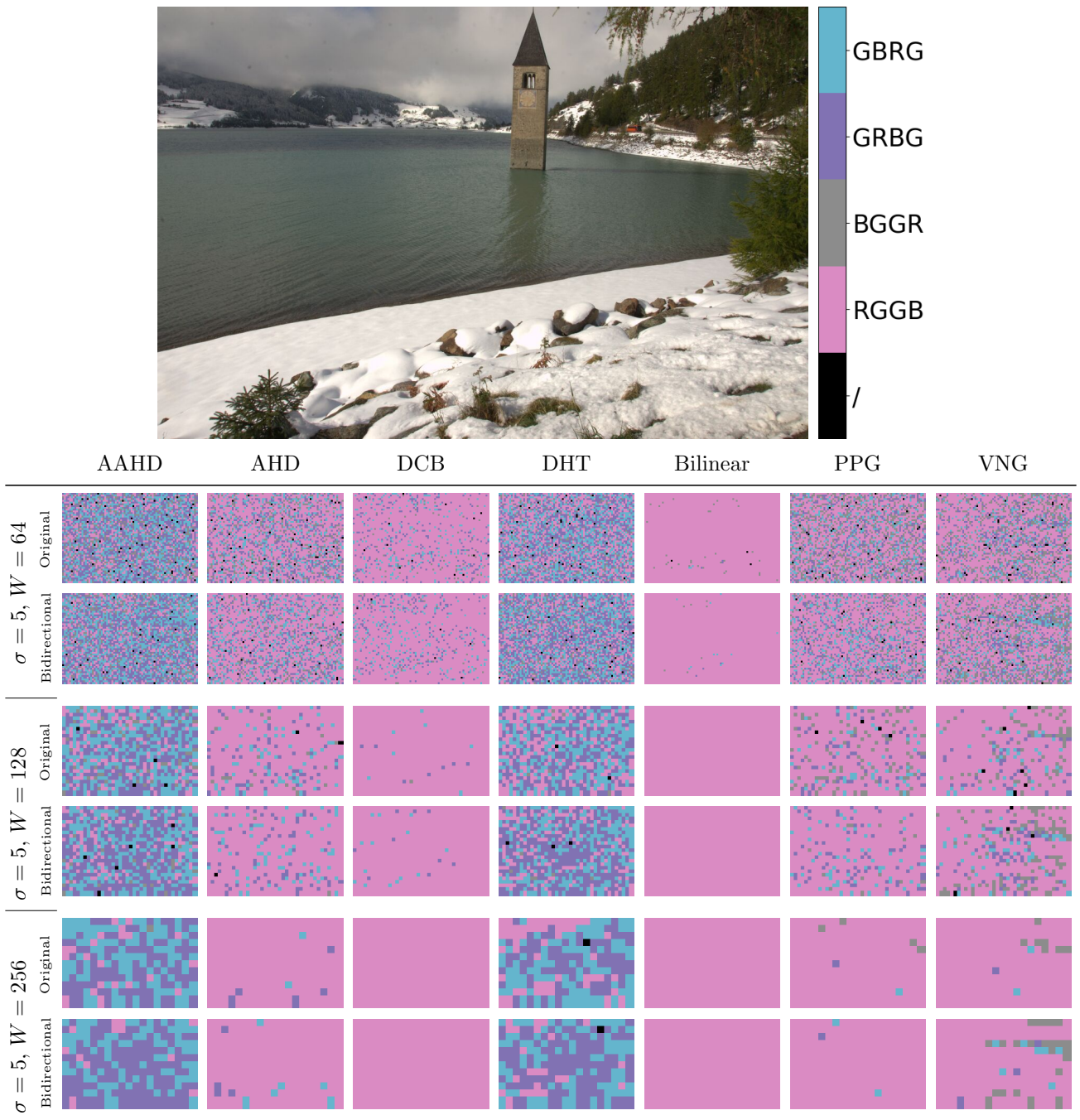
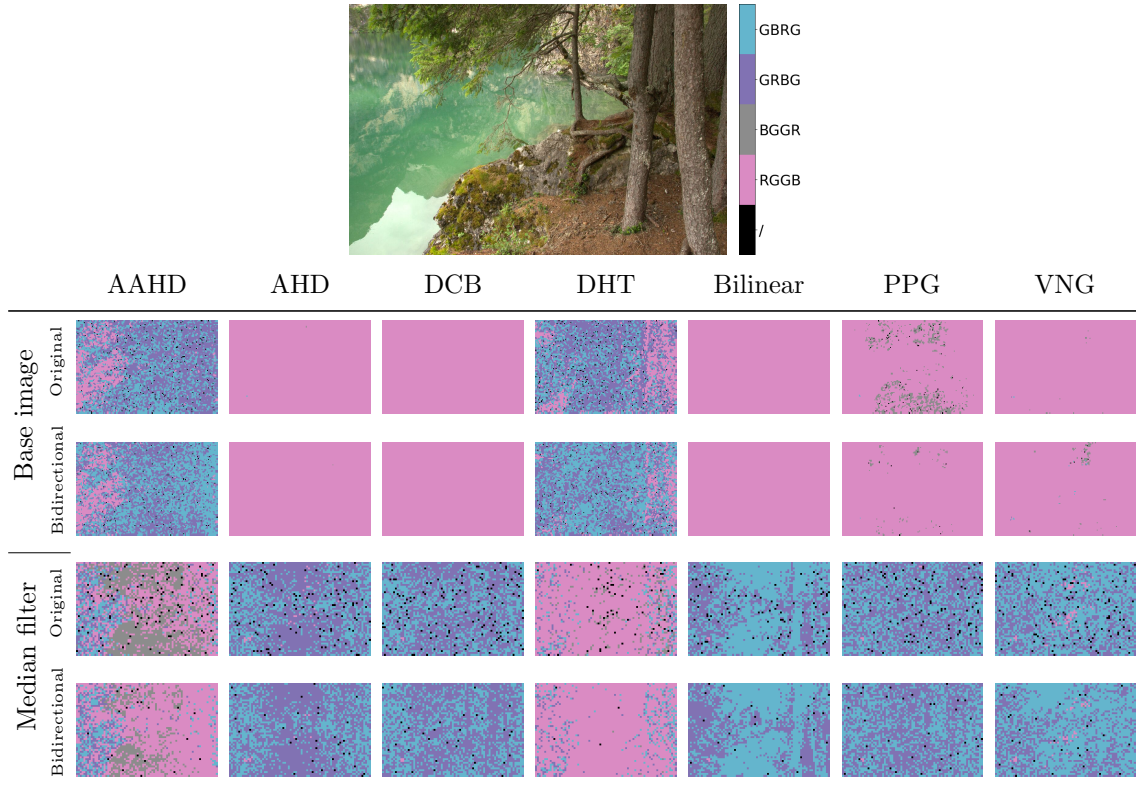
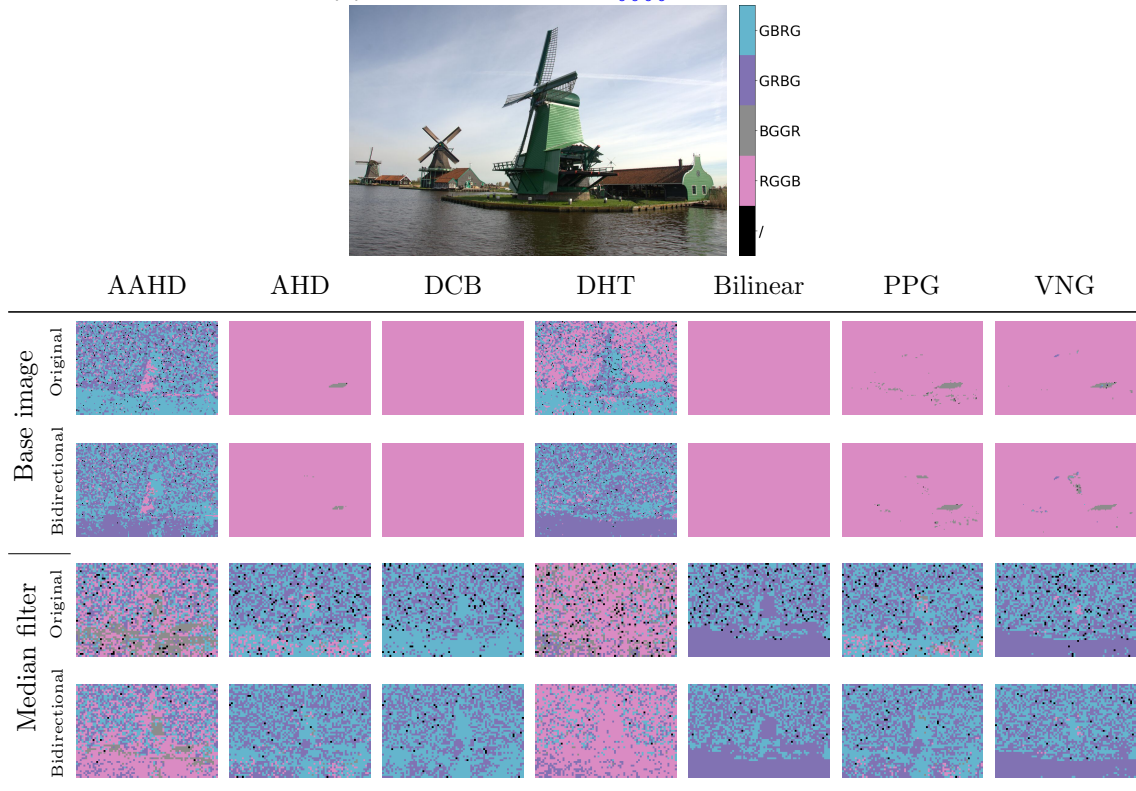


Figure 13: Robustness of the method to additive white Gaussian noise (AWGN), that can be added to images either for aesthetic reasons or to maliciously hide manipulations. Image r1c9fdcf4t, in [the](#) RGGB pattern. Noise standard deviation 5, with window sizes 64×64 , 128×128 and 256×256 . Because the noise is independent of the image and not spatially correlated, using bigger windows improves the robustness to it by providing more samples (at the cost of potentially missing smaller forgeries).



(a) Image r0e04cc91t, in the RGGB pattern.



(b) Image r1a0f5585t, in the RGGB pattern

Figure 14: Results of the method on 64×64 blocks on two images, unprocessed and median-filtered. The median filter is often used as a simple attack to hide forgeries by removing camera traces. It shifts the intermediate values on the green channel, thus confusing the algorithm on the diagonal pattern. Consequently, with the AAHD and DHT algorithms, which already shift the green channel intermediate values into the sampled pixels, the algorithms makes a better detection after median-filtering than on the unprocessed image.

139	240	154	16	94	56	72	20		139	139	168	94	72	94	56	43	
92	131	168	76	72	94	24	43		131	131	131	72	94	72	72	43	
85	24	100	48	102	224	130	72		85	100	100	76	72	122	130	123	
60	107	160	68	64	122	200	153		92	107	107	64	68	122	133	153	
92	184	125	0	50	0	133	108		72	125	156	68	50	117	133	133	
52	155	156	76	136	117	224	127		115	156	125	76	110	117	127	127	
146	228	111	12	110	108	107	44		146	146	111	90	110	110	107	100	
56	114	48	90	184	141	52	90		114	114	111	90	141	141	107	90	

(a) Original array
(b) After median filtering

Figure 15: Values on an array, before and after median filtering. The array corresponds to the green channel of an image in the ‘GG’ position demosaiced with bilinear interpolation. Red values correspond to positions where the value was interpolated. Highlighted cells correspond to pixels that take an intermediate value. Notice the shift of intermediate values from one diagonal to another: originally, almost all (18 out of the 20) intermediate values are found in the interpolated pixels, but after median filtering, most (16 out of 29) are located in the sampled (black) position.

3.2 Image Forgery Detection

The ultimate goal of the method is to find mosaic inconsistencies in an image. We use forgeries from the Trace database [2] to evaluate the method. The Trace database is constituted of 1000 images taken from the Raise dataset. Two forgery masks are made for each image: the endomask, obtained by taking a random object from the image’s automatic segmentation, and the exomask, which is simply the endomask of another image of the set and thus do not correlate to the contents of the image. The concept of the database is to process the image with two different pipelines, and merge them with one of the forgery masks. Of the six datasets that are proposed, two are of interest to us:

- in the CFA Grid dataset, the two pipelines are the same, but the pattern of demosaicing changes (the algorithm is the same). The forgery thus has a different CFA pattern than the rest of the image.
- in the CFA Algorithm dataset, the two pipelines are the same, but the algorithm of demosaicing changes. A new CFA pattern is also chosen at random for the forged region, with a $\frac{1}{4}$ chance of being the same than the original image’s.

For the quantitative experiments, we use the CFA grid with exomasks dataset. For the qualitative experiments, we use samples from both the CFA grid and CFA algorithm datasets. Unless otherwise specified, quantitative experiments are done with the Matthews Correlation Coefficient (MCC) [9]. This metric varies from -1 for a detection that is complementary to the ground truth, to 1 for a perfect detection. A score of 0 represents an uninformative result and is the expected performance of any random classifier. The MCC is more representative than the F1 and IoU scores [3, 4], partly because it is less dependent on the proportion of positives in the ground truth, which is especially important given the large variety of forgery mask sizes in the database. It is defined by

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

where TP, FP, TN and FN respectively represent the numbers of true positives (TP), false positives (FP), true negatives (TN) and false negative (FN). The score is computed for each image, and then averaged over each dataset. As most surveyed methods do not provide a binary output but a continuous heatmap, we weight the confusion matrix using the heatmap. For several results, we also provide the Intersection over Union (IoU), the F1 score and the Precision and Recall. Quantitative experimental results can be found in Table 2.

	MCC	IoU	F1
Isotropic, raw	0.518	0.490	0.573
Isotropic, $\gamma = 0.1$	0.592	0.567	0.622
Bidirectional, raw	0.543	0.515	0.595
Bidirectional, $\gamma = 0.1$	<u>0.610</u>	<u>0.584</u>	<u>0.642</u>
Bammey [1]	0.682	0.617	0.702

(a) Results with isotropic and bidirectional intermediate values, raw and with connected confidence both continuous and thresholded at $\gamma = 0.2$, compared with Bammey [1]. Both the presented method and Bammey are used on 32×32 windows.

	All images	Same diagonal	Different diagonal
Main grid	<u>0.476</u>	0.503	<u>0.461</u>
Diagonal	0.429	0.000	0.671
Combined	0.610	<u>0.501</u>	0.673

(b) Influence of using only main grid inconsistencies, diagonal inconsistencies and their combination (pointwise maximum of the two detection maps), on the full database, and when only looking at images whose authentic and forged parts share/do not share the same diagonal. The diagonal is shared in 364 out of the 1000 images of the dataset. By combining the two maps, the obtained results are almost as good as the diagonal map when the forgeries don't share their diagonal, almost as good as the full map when they do share the same diagonal (and thus cannot be detected by their diagonal), and overall much better than any of the two maps used alone.

Algorithm	All	AAHD	AHD	DCB	DHT	Bilinear	PPG	VNG
#Images	1000	126	138	133	155	154	147	147
Isotropic	0.592	0.372	0.696	0.786	0.305	0.742	0.590	0.657
Bidirectional	0.610	0.375	0.755	0.763	0.350	0.649	0.766	0.613

(c) Results of the presented method depending on how the image was demosaiced. The method is used with bidirectional filters, on 64×64 windows, with hysteresis thresholding and combining the main grid and diagonal inconsistencies. Even though the method finds the wrong diagonal with the AAHD and DHT algorithms, it is consistent in doing so, and can thus still detect some forgeries, though not as well as against other demosaicing algorithms.

	MCC	IoU	F1	Precision	Recall	Window size	MCC
$\gamma = 0.05$	<u>0.543</u>	<u>0.518</u>	<u>0.590</u>	0.570	0.733	16	<u>0.592</u>
$\gamma = 0.1$	0.610	0.584	0.642	0.670	<u>0.650</u>	32	0.610
$\gamma = 0.15$	0.531	0.513	0.558	<u>0.600</u>	0.535	64	0.412
$\gamma = 0.2$	0.382	0.371	0.400	0.433	0.382	128	0.163

(d) Results with different metrics, raw and with confidence thresholding. The method is used with bidirectional filters, on 64×64 windows and combining the main and diagonal inconsistencies. Even though thresholding slightly lowers the recall, its gain in precision is much larger, thus yielding better MCC, IoU and F1 scores.

(e) Results with different window sizes. The method is used with bidirectional filters, continuous normalisation at $\gamma = 0.2$, 32×32 windows.

Table 2: Quantitative experiments on the Trace database [2]. Where parameters are not specified, these are used: bidirectional filters, continuous normalization at $\gamma = 0.2$, 32×32 windows, combined results of the full pattern and diagonal maps.

In Table 2a, we can see that using bidirectional filters slightly improves the overall results. This corroborates the visual results of the previous subsection. Using thresholding not only improves the understandability of the method, it also provides significant improvements in the scores. Although this method does not work as well as Bammeay [1], we note that it is also much simpler and more interpretable.

Table 2b shows results from taking only the results of the diagonal detection, the full pattern or their combination by pointwise maximum. The strategy of merging the two maps by pointwise maximum is the good one: it performs almost as well as the diagonal map on forgeries which do not share their diagonal, almost as well as the full map on forgeries that do share their diagonal (and are thus invisible in the diagonal map), and thus performs much better on the overall database than any of the maps taken separately.

In Table 2c, we present the scores of the method depending on the demosaicing used to process the image. Unsurprisingly, the method does not work well on AAHD- and DHT-demosaiced images, as we saw in the previous subsection. However, because it is consistent in detecting the diagonal, it can still be used to see that two AAHD- or DHT-demosaiced regions use a different diagonal. This will be explored further below. Bidirectional filters work better than the original isotropic filters in most cases. The biggest gaps occur with PPG⁴ and AHD [7] demosaicing, which explicitly interpolate in the smoothest direction. On the other hand, isotropic filters work better with simpler demosaicing methods such as bilinear demosaicing, which does not try to find a better direction for interpolation.

We examine the influence of the threshold in Table 2d. As fewer windows get detected, a higher threshold systematically means that the recall is lower. However, a higher threshold does not necessarily improve the precision; the best precision (and best score overall) is achieved with a 0.1 threshold, and higher thresholds yield a lower precision. This can be explained by the fact that the most confident detections often correspond to textured areas, where intermediate values are created by the texture more than the demosaicing, and are thus a source of false positives.

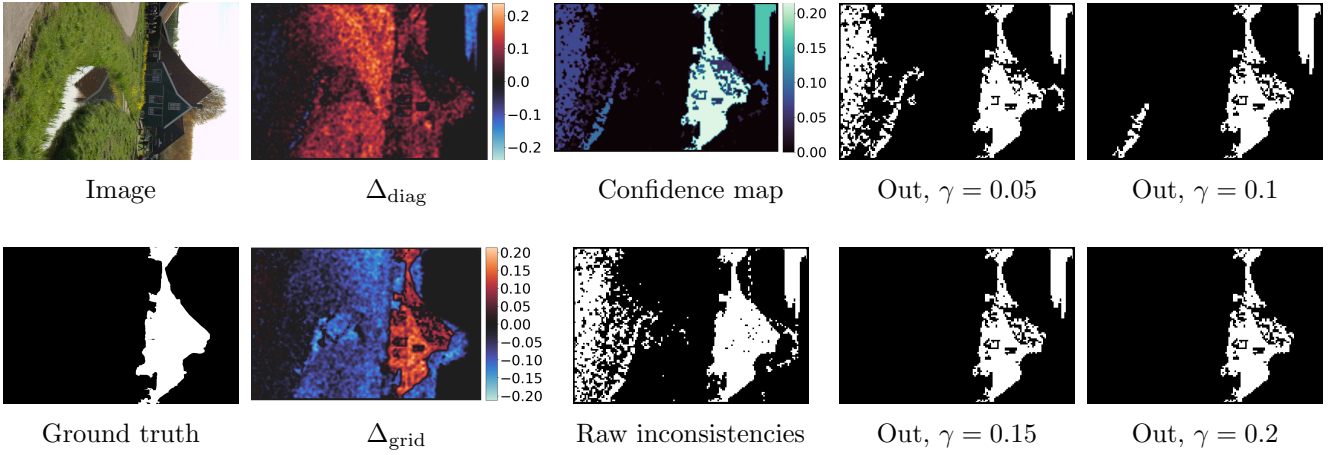
Finally, Table 2e shows the scores with different window sizes. While a window size of 32×32 yields the best results, this is inherently tied to the database in question. We saw earlier that increasing the window size would often lead to a better grid identification, but this also comes at the cost of missing small forgeries, and also failing to identify the borders of the bigger ones.

In Figure 16, we investigate the importance of the thresholding. Inconsistent false detections are usually not found in large connected regions of the same detected grid. As a consequence, even if some of those detections were to be significant, they would not cause a large number of false detections. On the other hand, regions detected because they are truly forged have a high chance of being actually forged. A confident result on one window of that region is thus enough to detect the whole region. Of course, this threshold is not fully automatic: it must still be set by the user, and will not filter out zones that are strongly detected for reasons other than a forgery, such as saturation or textured areas.

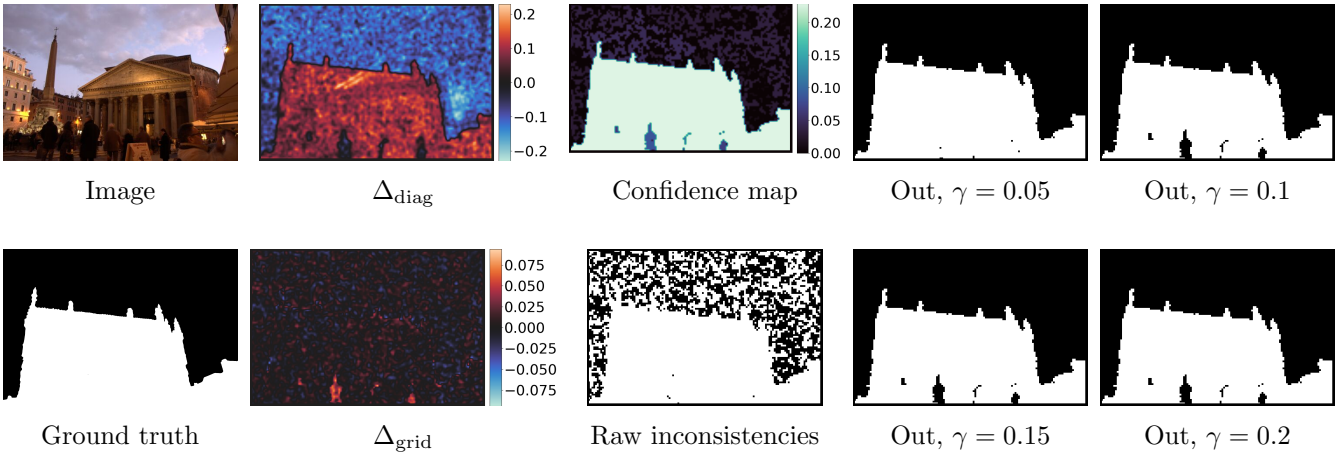
The image in Figure 16b is interesting: as it was demosaiced with AAHD, its diagonal is not detected correctly (and no confident detection is thus made in Δ_{grid}). However, because the two regions do not share their diagonal, the inconsistency is still detected. The region demosaiced in [the GRBG pattern](#) is detected as being in the $\cdot GG \cdot$ diagonal, whereas the region demosaiced in [the BGGR](#) is detected as $G \cdot G$: even though those predictions are wrong, they are still inconsistent with one another.

We further explore this phenomenon in Figure 17. As long as the two regions of an AAHD-demosaiced image do not share the same diagonal, they can still be detected. However, if the diagonal is wrong, then the algorithm will not look at the correct difference map to select the full

⁴A description of Patterned Pixel Grouping (PPG) by his author, Chuan-Kai Lin, can be found in <https://web.archive.org/web/20160923211135/https://sites.google.com/site/chklin/demosaic/>, archived from the original page on 23rd September 2016.



(a) Image r06888d38t of the CFA Grid dataset with endomask, demosaiced with the VNG algorithm. The authentic region was demosaiced in the GRBG pattern, the forged region in the GBRG pattern.

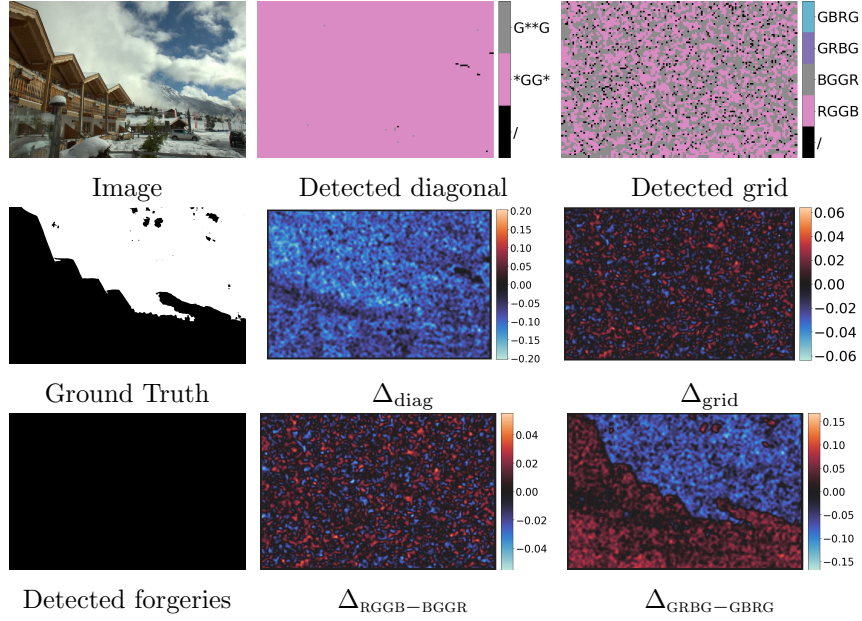


(b) Image r1594b4b3t of the CFA Grid dataset with exomask, demosaiced with the AAHD algorithm. The authentic region was demosaiced in the GRBG pattern, the forged region in the BGGR pattern.

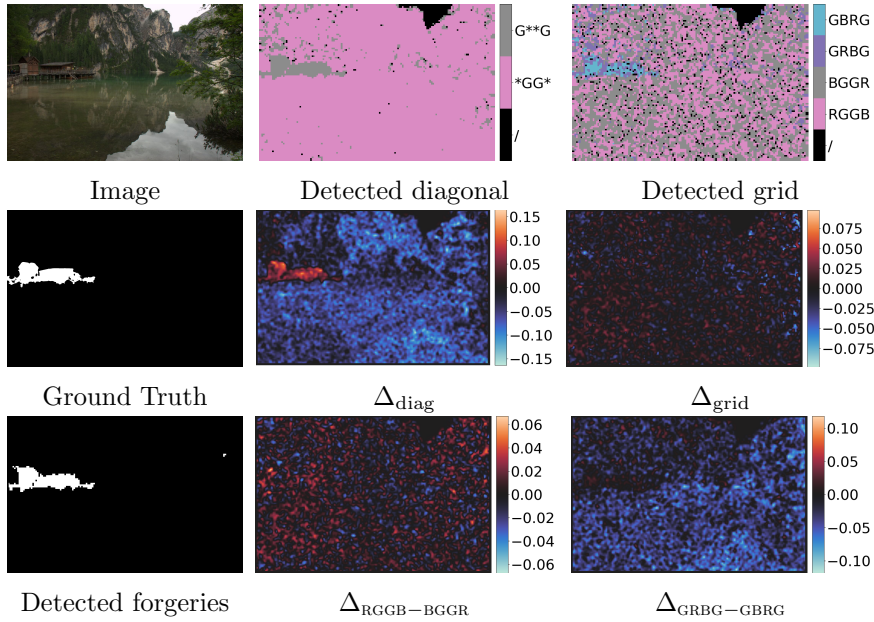
Figure 16: Influence of the threshold on the removal of inconsistent false positives. Less-confident regions of the correct detection are kept, as long as they are connected to a more confident window, whereas inconsistent false detections are not connected to a large region: even if a few of them are confidently detected, most will be filtered out. Raw inconsistencies highlight every block whose detected pattern is inconsistent with the main image, regardless of significance.

pattern. As a consequence, it will be unable to make a consistent decision, and will thus not detect two different patterns that share the same diagonal. Nevertheless, the unused Δ maps still shows clear traces of the forgery. If one is suspicious that the diagonal is reversed (for instance because it is detected confidently over the whole image, whereas the full pattern is inconsistent), inverting the results of the diagonal, or visually examining the unused Δ map, can thus still reveal the forgery.

In the case of an image whose forged regions come from different algorithms, one of which being AAHD or DHT, the diagonal inversion means that if the two regions share the same diagonal (or even the very same pattern), the forgery could be detected, albeit for the wrong reason. However, this also means that if the two regions do not share the same pattern – an inconsistency which is usually easier to find, as seen in Table 2b –, the forgery then becomes invisible. This problem can be seen in Figure 18.



(a) Image r1d53fccat of the CFA Grid dataset, with endomask. Both regions are demosaiced with the AAHD algorithm, the authentic region in the GBRG pattern and the forged region in the GRBG pattern. Because the method wrongly detects the ·GG· over the whole image, it only compares the two patterns sharing that diagonal, and $\Delta_{\text{grid}} = \Delta_{\text{RGGB-BGGR}}$ on almost all the image. As a consequence, no detection can be made on the grid level, and the forgery is not detected. Nevertheless, it appears clearly on $\Delta_{\text{GRBG-GBRG}}$, which is not used in the algorithm.



(b) Image r15919202t of the CFA Grid dataset, with endomask. Both regions are demosaiced with the AAHD algorithm, the authentic region in the GRBG pattern, the forged region in the BGGR pattern. Although the method finds the wrong diagonal in both regions, it still finds that the two regions use a different diagonal.

Figure 17: On those two ~~AAHD-demosaicked-AAHD-demosaicked~~ images, the method finds the wrong diagonal. On the second image, it can still detect that the two regions use a different diagonal, and detects the forgery. On the first image, however, the two regions share the same diagonal. Using the diagonal as a basis to detect the full pattern, the method is unable to make any consistent ~~detection-detections~~ on the full pattern when the diagonal is wrong, and thus does not find the forgery even though it is clearly visible when the correct diagonal is used. This shows the limits of using the detected diagonal as a strict basis for the full detection.

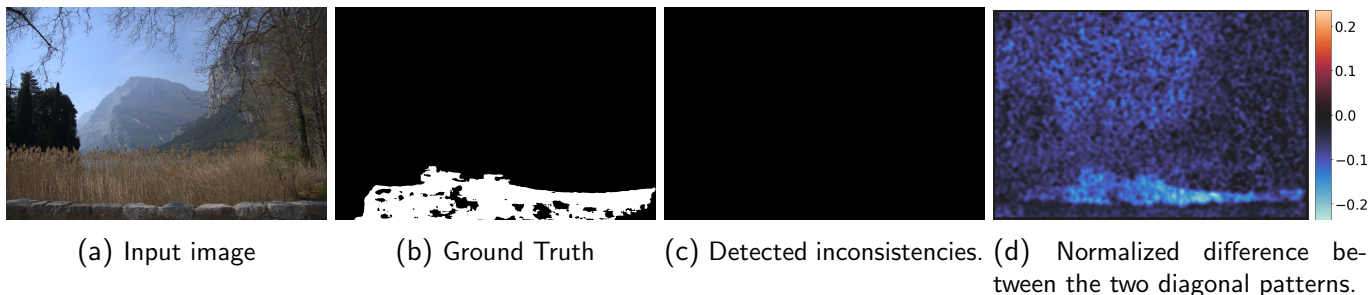


Figure 18: Image r040b3002t of the CFA Algorithm dataset, with exomask. The authentic region is demosaiced with the AAHD algorithm in the GRBG pattern, the forged region is demosaiced with the DCB algorithm in the BGGR pattern. Because the method consistently finds the wrong diagonal on AAHD-demosaiced images, but detects the correct diagonal on DCB-demosaiced images, it believes that the two regions share the same diagonal, even though they do not.

4 Conclusion

The presented method counts the number of locally intermediate values corresponding to each potential demosaicing pattern. The correct pattern, which contains all the originally-sampled pixels, is expected to have fewer intermediate values than the other patterns. The method starts by detecting which diagonal is used with the green channel, then uses the red and blue channels to distinguish between the two patterns sharing the detected diagonal. We proposed a different way of computing intermediate values, which yields slightly better results by exploiting the fact that demosaicing algorithms often interpolate on only one direction.

Doing this both on the full image and in local windows, we are able to detect locally inconsistent windows. The difference between the two compared counts of intermediate values in a window is normalized to serve as a confidence measure for the detection of this window. Those windows are then grouped into connected components of windows that share the same detected grid, and the confidence of a component is set as the maximal confidence of its windows.

This confidence map enables easy visualization of the detections and their significance, and thresholding it filters out most of the incoherent false detections, while keeping the consistent detections. However, the threshold must be set manually.

Of the seven tested demosaicing algorithms, two cause an inversion of the detected diagonal. In the green channel, AAHD and DHT actually cause more intermediate values to appear on the sampled pixels.

The main limitation of this method, and of CFA forgery detection in general, is its low robustness to JPEG compression. With this method, detections are already more difficult at a quality factor of 100, and become impossible at quality 95, effectively limiting its applicability range. We also saw that counter-forensic attacks based on the addition of white noise are quite effective. The classic median filter attack is instead relatively inefficient as it ends up inverting the interpolation masks, and the diagonal can thus still be detected.

Acknowledgment


Work partly funded by the French Ministère des Armées – Direction Générale de l’Armement, and by grant ANR-16-DEFA-0004 Signature d’Images – ANR/DGA DEFALS challenge. 

Image Credits



Raise dataset [6], processed with libraw⁵



Trace dataset [2]

References

- [1] Q. BAMMEY, R. GROMPONE VON GIOI, AND J-M. MOREL, *An adaptive neural network for unsupervised mosaic consistency analysis in image forensics*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020. <https://doi.org/10.1109/CVPR42600.2020.01420>.
- [2] Q. BAMMEY, T. NIKOUKHAH, M. GARDELLA, R. GROMPONE VON GIOI, M. COLOM, AND J-M. MOREL, *Non-Semantic Evaluation of Image Forensics Tools: Methodology and Database*, in Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, January 2022. <https://arxiv.org/abs/2105.02700>.
- [3] D. CHICCO, *Ten quick tips for machine learning in computational biology*, BioData Mining, 10 (2017), pp. 35–35. <https://doi.org/10.1186/s13040-017-0155-3>.
- [4] D. CHICCO AND G. JURMAN, *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation*, BMC genomics, 21 (2020), pp. 6–6. <https://doi.org/10.1186/s12864-019-6413-7>.
- [5] CHANG-HEE CHOI, JUNG-HO CHOI, AND HEUNG-KYU LEE, *Cfa pattern identification of digital cameras using intermediate value counting*, in Proceedings of the Thirteenth ACM Multimedia Workshop on Multimedia and Security, MM&Sec '11, New York, NY, USA, 2011, Association for Computing Machinery, p. 21–26. <https://doi.org/10.1145/2037252.2037258>.
- [6] D-T. DANG-NGUYEN, C. PASQUINI, V. CONOTTER, AND G. BOATO, *Raise: A raw images dataset for digital image forensics*, in Proceedings of the 6th ACM Multimedia Systems Conference, 2015, pp. 219–224. <https://dl.acm.org/doi/pdf/10.1145/2713168.2713194>.
- [7] K. HIRAKAWA AND T.W. PARKS, *Adaptive homogeneity-directed demosaicing algorithm*, IEEE Transactions on Image Processing, 14 (2005), pp. 360–369. <https://doi.org/10.1109/TIP.2004.838691>.
- [8] M. KIRCHNER AND J. FRIDRICH, *On detection of median filtering in digital images*, in Media Forensics and Security II, vol. 7541, International Society for Optics and Photonics, 2010, p. 754110. <https://doi.org/10.1117/12.839100>.
- [9] B.W. MATTHEWS, *Comparison of the predicted and observed secondary structure of T4 phage lysozyme*, Biochimica et Biophysica Acta (BBA) - Protein Structure, 405 (1975), pp. 442–451. [https://doi.org/10.1016/0005-2795\(75\)90109-9](https://doi.org/10.1016/0005-2795(75)90109-9).
- [10] S. VAN DER WALT, J.L. SCHÖNBERGER, J. NUNEZ-IGLESIAS, F. BOULOGNE, J.D. WARNER, N. YAGER, E. GOULLART, T. YU, AND THE SCIKIT-IMAGE CONTRIBUTORS, *scikit-image: image processing in Python*, PeerJ, 2 (2014), p. e453. <https://doi.org/10.7717/peerj.453>.

⁵LibRaw library, Copyright ©008-2019 LibRaw LLC, <https://www.libraw.org>