A Pixel-Based Technique to Detect Duplications in an Image

Min-Hua Tsai

Department of Computer Science
University at Albany - SUNY
1400 Washington Avenue, Albany, NY 12222, U.S.A.
mtsai@albany.edu

Abstract

With the growing of photo and video editing tools, it has become much easier to tamper with photos. One common way is to insert visually believable composites into target images. In this document, we implement a fully automatic method to detect the duplication parts in a digital image based on its pixel values. To detect this kind of forgery, we first load the image and make it numeralization, which means defining the values for each pixel. Then the method we proposed will automatically detect the forgery in the image, and then the result will present the duplication part also with its location. The method allows us to both understand if a copy–move attack has occurred, also indicates the locations where the attacks are. Besides, the method is able to deal with multiple duplications.

***Index Terms*—Image forensics, copy–move, pixel-based, cloning, duplications.**

I. Introduction

It is beyond any doubt that we are living in an age where digital information has grown exploded over the past few decades. There is an idiom says "to see is to believe", while this trust has begun erodent since there are more and more of tampered photos and videos flooding in our daily lives, such forgeries exist in televisions, magazines, and on the internet, which hide the truth. Besides, with powerful image and video editing tools, it is becoming much easier to tamper with images and videos. Therefore, evaluating the authentication of digital photos has turned out to be a crucial task nowadays.

Photos are often regarded as important evidences in many occasions, because people think they record the information objectively. However, by tampering the photos with some editing tools, this kind of proof would not be trustful. Those tools are not supposed to be used on cheating public, the original purpose of them is to beautify digital photographs and to create original digital artwork. To prevent the abuse of using those photo editing tools and maintain the reliability of photos, image forgery detection tools are needed.

We provide a method to detect copy-move manipulations in an image, a program was built to implement our thought. The program first loads the image and converts it into grayscale, and then we can obtain an array composed by numbers, which are pixel values for each pixel. Basically, the values of the pixels which were duplicated will remain the same as the original parts in the image. The program will automatically detect if there are two or more areas has the same pixel values, and then the result illustration where are the duplication parts and their locations in a new image.

II. RELATED WORKS

While we are processing the image in a digital domain, the technique we are applying is pixel-based. Pixel-based techniques analyze the correlations in a tampered image in pixel-level. There are other related works which detect the copy-move forgery.

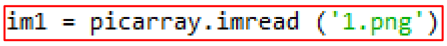
The authors in [1] proposed an efficient pixel-based algorithm to detect cloned. In [2], the authors quantified statistical correlations that result from specific forms of digital tampering, and then devised detection schemes to reveal these correlations. A method which could successfully detected the forged part even when the copied area is enhanced or retouched to merge it with the background and when the forged image is saved in JPEG format is described in [3]. In [4], the presence of duplicated regions in the image was regarded as a tell-tale sign for image forgery, a blind forensics approach based on DWT (discrete wavelet transform) and SVD (singular value decomposition) is proposed to detect the duplications. A technique was proposed in [5], which works by first applying a principal component analysis to small fixed-size image blocks to yield a reduced dimension representation.

III. Method

The method in this document is using Python with the following package: "matplotlib", "skimage", "numpy" and "scipy". The method can be roughly divided into four parts: Firstly loading the picture and store the pixel in an array; second, converting the original image from RGB to grayscale; third, detecting the duplications parts; last, producing and revealing the result.

A. *Loading the Tampered Image*

The program loads the image the stores its information by "image" module in "matplotlib" package.



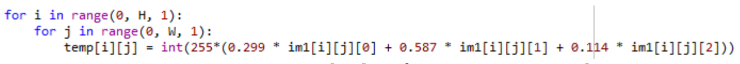
```
im1 = picarray.imread ('1.png')
```

Fig. 1 Loading the tampered image.

We first use Python to load the image and store its pixel information in a new created array "im1", shown as fig. 1. The pixel information represents the value of the pixel, also the color of the pixel. Assuming the size of the image is m*n, the size of the new created array would be m*n.

B. *Converting from RGB to Grayscale*

After the first step, we need to convert the values in the array from RGB to grayscale, it is because the in RGB color space, the value of each pixel is a tuple (x, y, z, 1) represents a colour that is made up of 'x' amount of red, 'y' amount of green, and 'z' amount of blue. It would be much complicated and time-consuming if compare each pixel value in RGB space, thus we convert the pixel value from RGB to grayscale, and then the pixel values are only one number, shown as fig. 2.



```
for i in range(0, H, 1):
    for j in range(0, W, 1):
        temp[i][j] = int(255*(0.299 * im1[i][j][0] + 0.587 * im1[i][j][1] + 0.114 * im1[i][j][2]))
```

Fig. 2 Converting pixel values from RGB to grayscale.

In figure 2, we convert each pixel value in im1 and store the result to a new array “temp”, and the elements in “temp” are both integers between 0 and 255. Thus the pixel values are much easier to compare to each other.

C. Analyze and Detect Duplications

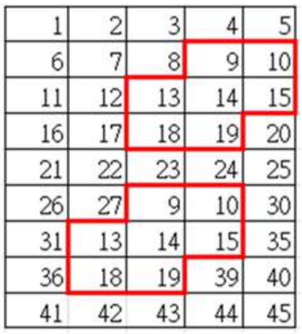
In this process, we divide the pixels into four groups according to their values. The first group is with the pixel value between 0 and 63; the second group is with the pixel value between 64 and 127; the third group is with the pixel value between 128 and 191; the forth group is with the pixel value between 192 and 255. Then we create four arrays to store the pixel information, the sizes of the arrays depend on the amount of pixels within the boundaries. The reason why dividing into four arrays is to avoid the large size image would cause the size of the array over the array size constraint in python. Fig. 3 illustrates the elements of the four arrays.

region1		
1	0	0
2	0	1
3	0	2
3	3	1
4	0	3
5	1	0

Fig. 3 Elements of the array.

The array in fig. 3 has three columns, the first column is the pixel value; the second column is the x coordinate of the pixel location, and the third column is the y coordinate of the pixel location. After storing each pixel value and location into the four arrays, we sort each array by pixel value. We can see that the pixels with the same value are adjacent, this helps making it easier to analyze.

Now we start to begin the detection process, the definition of the duplication parts is when there are more than a given number of pixels that are connected to each other and the distance between pixel pairs is the same, an example shown as fig. 4.



1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	9	10	30
31	13	14	15	35
36	18	19	39	40
41	42	43	44	45

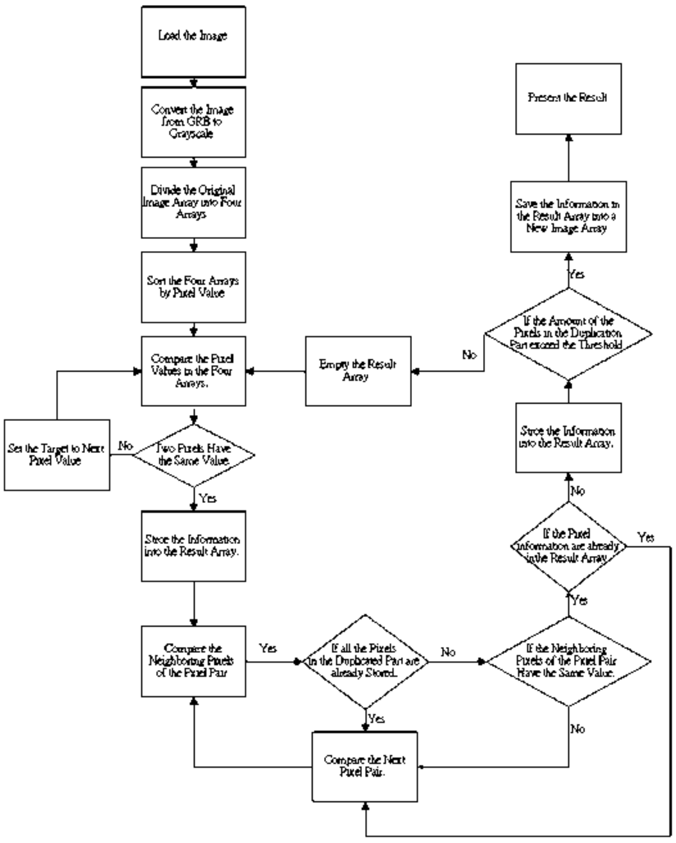
Fig. 4 Duplication parts in an image array.

To detect this kind of forgery, our method applies depth-first search (DFS). When we find that two pixels have the same value, we keep finding if there are other pixel pairs are duplicated recursively. We use an array to store the result which contains the pixel value and locations we determine to be forgery, shown as fig. 5.

result				
3	0	2	3	1

Fig. 6 Elements of the result array.

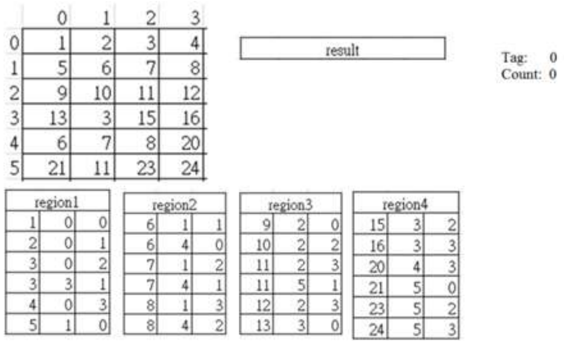
In fig. 6, there are five elements in the array, which stores one pixel value and one pair of pixel locations, each column represents one element. The first column is the pixel value; the second and third columns are the coordinates of the first pixel; the forth and fifth columns are the coordinates of the second pixel. The detailed procedure is shown as fig. 7.



Load the Image
Convert the Image from GRB to Grayscale
Divide the Original Image Array into Four Arrays
Sort the Four Arrays by Pixel Value
Compare the Pixel Values in the Four Arrays.
Two Pixels Have the Same Value
No
Set the Target to Next Pixel Value
Yes
Store the Information into the Result Array.
Compare the Neighboring Pixels of the Pixel Pair
Yes
If all the Pixels in the Duplicated Part are already Stored.
No
Yes
If the Neighboring Pixels of the Pixel Pair Have the Same Value.
No
Compare the Next Pixel Pair.
Yes
If the Pixel information are already in the Result Array.
Yes
No
Store the Information into the Result Array.
If the Amount of the Pixels in the Duplication Part exceed the Threshold
No
Empty the Result Array
Yes
Save the Information in the Result Array into a New Image Array
Present the Result

Fig.7 Program flow chart of the method.

To express how the duplicated parts are detected, fig. 8 to fig. x illustrate an example of the processes in detailed.



	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	3	15	16
4	6	7	8	20
5	21	11	23	24

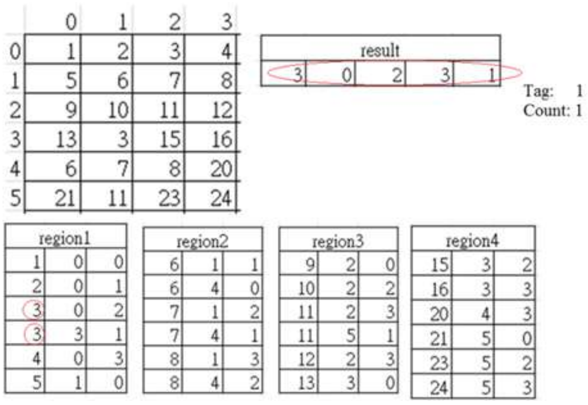
result

Tag: 0
Count: 0

region1			region2			region3			region4		
1	0	0	6	1	1	9	2	0	15	3	2
2	0	1	6	4	0	10	2	2	16	3	3
3	0	2	7	1	2	11	2	3	20	4	3
3	3	1	7	4	1	11	5	1	21	5	0
4	0	3	8	1	3	12	2	3	23	5	2
5	1	0	8	4	2	13	3	0	24	5	3

Fig. 8 The initial array information.

Fig. 8 shows the array information after loading the image and dividing into four arrays, and then sort the arrays by pixel value. There are two parameters needed, “Count” and “Tag”. “Count” is the number of pixel in the result array; “Tag” is the pixel location in result array which we are currently dealing with.



	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	3	15	16
4	6	7	8	20
5	21	11	23	24

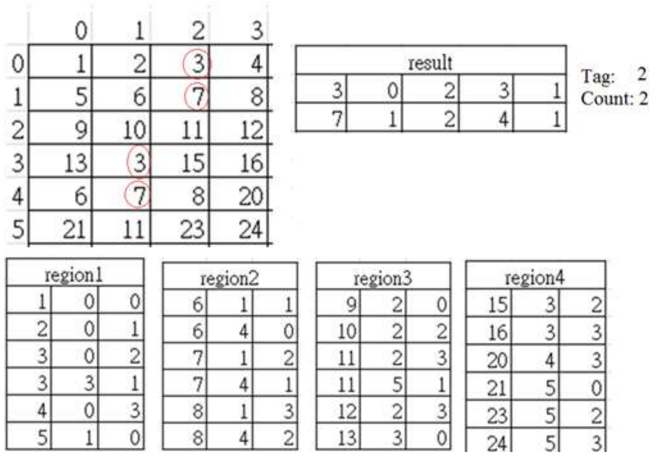
result				
3	0	2	3	1

Tag: 1
Count: 1

region1			region2			region3			region4		
1	0	0	6	1	1	9	2	0	15	3	2
2	0	1	6	4	0	10	2	2	16	3	3
3	0	2	7	1	2	11	2	3	20	4	3
3	3	1	7	4	1	11	5	1	21	5	0
4	0	3	8	1	3	12	2	3	23	5	2
5	1	0	8	4	2	13	3	0	24	5	3

Fig. 9 Storing the pixel information into the result array.

The program starts to detect if there are two pixels have the same value. In fig. 9, two pixels have the same value 3, and the information of them are store in the result array. Then we keep detect if there are neighboring pixels have the same value, and we find that the pixels under the current pixel pair are both 7, so the information of the new detected pixels is stored into the result array, shown as Fig. 10.



	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	3	15	16
4	6	7	8	20
5	21	11	23	24

result				
3	0	2	3	1
7	1	2	4	1

Tag: 2
Count: 2

region1		
1	0	0
2	0	1
3	0	2
3	3	1
4	0	3
5	1	0

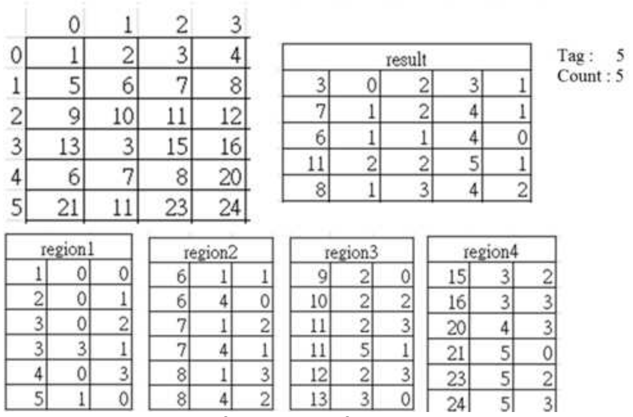
region2		
6	1	1
6	4	0
7	1	2
7	4	1
8	1	3
8	4	2

region3		
9	2	0
10	2	2
11	2	3
11	5	1
12	2	3
13	3	0

region4		
15	3	2
16	3	3
20	4	3
21	5	0
23	5	2
24	5	3

Fig. 10 Storing the new detected pixels information.

The program will keep detect and store the detected pixels information into the result array, also there is a function to check if the detect whether the detected pixel is already exist in the result array, this function is used to prevent the endless recursion. The detection will be terminated if the "Tag" equals "Count", which means all pixels in the duplication part are detected, the final detection result shown as Fig. 11. Then we define a threshold to determine whether the duplication parts are large enough to be called a forgery.



	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	3	15	16
4	6	7	8	20
5	21	11	23	24

result				
3	0	2	3	1
7	1	2	4	1
6	1	1	4	0
11	2	2	5	1
8	1	3	4	2

Tag : 5
Count : 5

region1		
1	0	0
2	0	1
3	0	2
3	3	1
4	0	3
5	1	0

region2		
6	1	1
6	4	0
7	1	2
7	4	1
8	1	3
8	4	2

region3		
9	2	0
10	2	2
11	2	3
11	5	1
12	2	3
13	3	0

region4		
15	3	2
16	3	3
20	4	3
21	5	0
23	5	2
24	5	3

Fig. 11 The final result of duplication detection.

The new image array will be created and stores the information in the result array. The result image is initialized all blank, then only store the pixel value to the corresponding coordinates in the result array, which means the result image will only contain the duplication parts of the original image.

IV. EXPERIMENTAL RESULTS

To validate our method, we tamper some images and use the program to process. The results will show if the method if effective and also the performance, there are three aspects we are going to test. Fig. 12 is the original and non-tampered image we use for the testing, the size of it is 416 * 336.

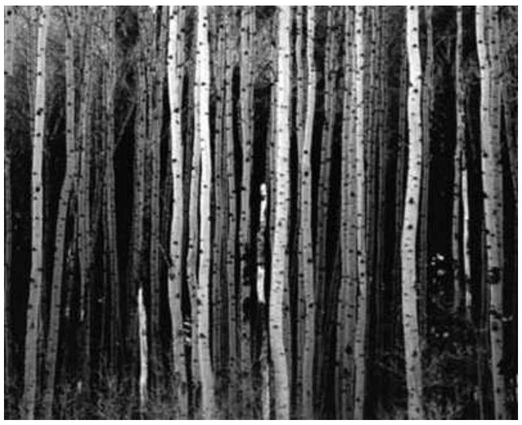


Fig. 12 The test sample used to validate our method.

A. Single Duplication in an Image

We tamper the original image shown as fig. 13 then execute the program.

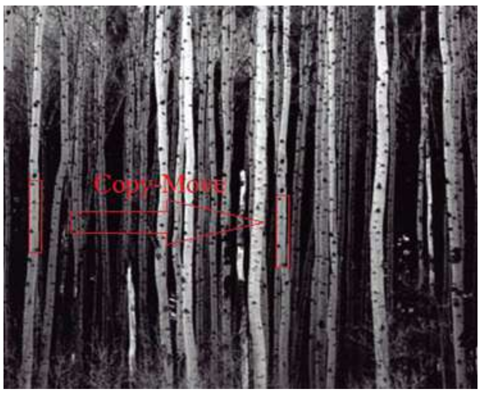


Fig. 13 Tamper with the image with one copy-move operation.

The execution time is 30 seconds, and the result is correct, shown as fig. 14.

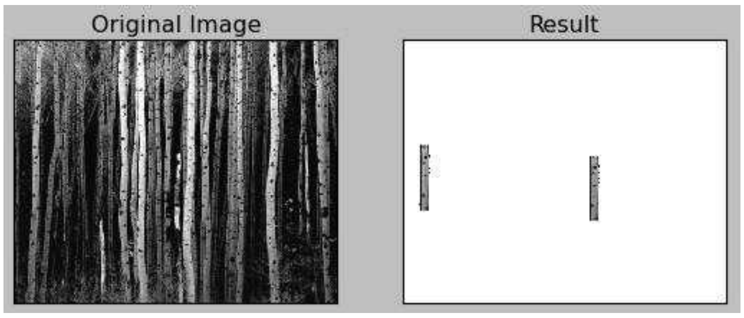


Fig. 14 The result of detecting single duplication in an image.

B. Multiple Duplications Copied from the Same Source

In this situation, we copy a part in an image, and then paste it in more than one location in the image, shown as fig. 15.

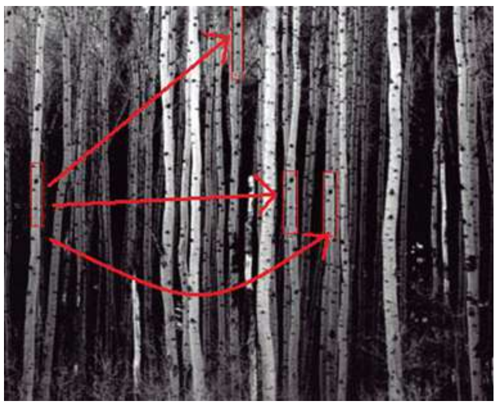


Fig. 15 Tamper with the image with two copy-move operations from the same source.

It takes 32 seconds to produce the result, and the result is also correct, shown as fig. 16.

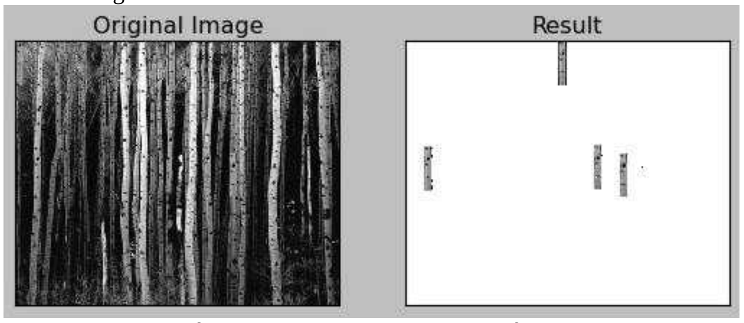


Fig. 16 The result of detecting multiple duplications from a source.

C. Multiple Duplications Copied from Different Sources

Finally, we validate the program with the most complicated situation; copying various parts and paste in different locations, the tampered image shown as fig. 17.

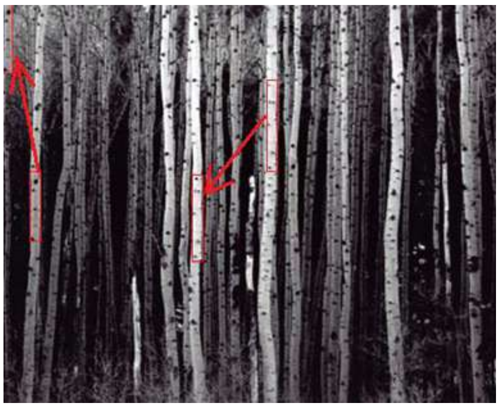


Fig. 17 Tamper with the image with three copy-move operations from the different sources.

The result is correct and takes 35 seconds, fig. 18 presents the result.

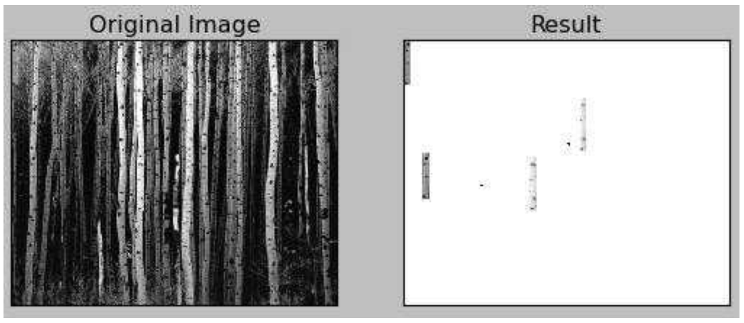


Fig. 18 The result of detecting multiple duplications from different sources.

After testing the three different situations, we find out the program is valid and able to detect different copy-move operations.

V. Discussion

We proposed a new method to detect image forgery in an image, our target is to detect the copy-move operations in an image, which is perhaps the most often seen forgery operation. A program was created base on the method, it automatically detect the duplication parts in an image, and presents the result.

The method works correctly on the situations tested in the part IV. However, if the forged image was manipulated with JPEG compression, then even the pixels in the duplication parts would not have the same values. Besides, the method is not robust to deal with some types of transformation, such as scaling and rotation. In the future, we will do our endeavors to enhance the algorithm to make it be able to detect the scaling and rotation transformations. Also, we will find out how the pixels change between JPEG compression and non-JPEG compression, and improve our algorithm to make it capable to detect the copy-move operations even the image has been manipulated with JPEG compression.

References

[1] B. Mahdian and S. Saic, “Detection of copy move forgery using a method based on blur movement invariants,” Forensic Sci. Int., vol. 171, pp. 180–189, 2007.

[2] A. C. Popescu and H. Farid, “Statistical tools for digital forensics,” in Proc. 6th Int. Workshop on Information Hiding, Toronto, Canada, 2004, pp. 128–147.

[3] J. Fridrich, D. Soukal, and J. Lukás, “Detection of copy move forgery in digital images,” in Proc. Digital Forensic Research Workshop, Aug. 2003.

[4] G. Li, Q. Wu, D. Tu, and S. Sun, “A sorted neighborhood approach for detecting duplicated regions in image forgeries based on DWT and SVD,” in IEEE Int. Conf. Multimedia and Expo, Beijing, China, 2007, pp. 1750–1753.

[5] A.C. Popescu and H. Farid, “Exposing digital forgeries by detecting duplicated image regions,” Dept. Comput. Sci., Dartmouth College, Tech. Rep. TR2004-515, 2004.