Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An efficient expanding block algorithm for image copy-move forgery detection

Gavin Lynch [a], Frank Y. Shih [a,*], Hong-Yuan Mark Liao [b]

[a] *College of Computing Sciences, New Jersey Institute of Technology, Newark, NJ 07102, United States*
[b] *Institute of Information Science, Academia Sinica, Taiwan*

A R T I C L E I N F O

Article history:
Received 19 May 2012
Received in revised form 4 March 2013
Accepted 10 March 2013
Available online 21 March 2013

Keywords:
Copy-move forgery
Expanding block
Feature extraction
Copy detection

A B S T R A C T

Image forgery is becoming more prevalent in our daily lives due to advances in computers and image-editing software. As forgers develop more sophisticated forgeries, researchers must keep up to design more advanced ways of detecting these forgeries. Copy-move forgery is one type of image forgery where one region of an image is copied to another region in an attempt to cover a potentially important feature. This paper presents an efficient expanding block algorithm for detecting copy-move forgery and identifying the duplicated regions in an image. Experimental results show that the new method is effective in identifying size and shape of the duplicated region. Furthermore, it allows copy-move forgeries to be detected, where the copied region has been manufactured slightly lighter or darker, under JPEG compression, or with the effect of Gaussian blurring, in an attempt to throw-off detection algorithms.

1. Introduction

Image authentication techniques can be classified into two categories: active methods and passive methods. Active methods such as watermarking [10,21] or illegal image copy detection [8,13,17] depend on prior information about the image. However, in many situations, prior information regarding an image is not available and passive, or blind, methods should be used to authenticate the image. In this paper, we focus on one type of passive image forgery detection known as copy-move forgery detection. A copy-move forgery is used to hide a region of the image by covering it with a copy of another region of the image. Fig. 1 shows an example of a copy-move forgery and the identified forged and copied regions using the cameraman image. When detecting this type of forgery, not only are we interested in whether the image contains this forgery, but also the location and shape of the forged region. However, identifying the forged region can be complicated by the fact that the forged region may be similar, but not an exact copy of another region. This can happen through image transformations, such as scaling or JPEG compression, or because the forger put in extra effort into hiding the forgery, such as blurring or slightly rotating the forged region.

Several algorithms to detect copy-move forgery are based on the sliding block method presented by Fridrich et al. [6]. The main idea is that rather than trying to identify the entire forged region, the image is divided into small overlapping blocks. The blocks are compared against each other in order to see which blocks are matching. The regions of the image covered by the matching blocks are the copied and forged regions.

In general, copy-move forgery detection consists of feature extraction, comparison, and copy decision based on the similarity information [5]. In the feature extraction step, important features are selected from each block which are used to

* Corresponding author. Tel.: +1 973 596 5654; fax: +1 973 596 5777.
E-mail address: shih@njit.edu (F.Y. Shih).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.03.028

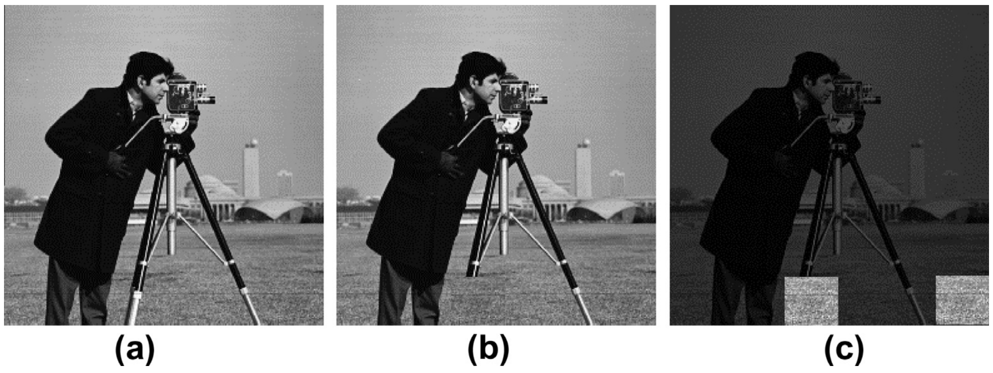


Fig. 1. (a) The original cameraman image, (b) the forged image, and (c) the forged image with the copied regions detected and highlighted.

indirectly compare the blocks. A good feature extraction algorithm should extract similar features for two blocks that are approximately the same. Since images can be altered in various ways (scaled, compressed, blurred, etc.), feature extraction methods should be able to extract important features while ignoring subtle noise. The features are placed into a matrix called the *feature matrix*.

Popescu and Farid [19] developed a method based on *principal component analysis* (PCA). PCA is an important image classification tool and is used in several algorithms such as facial recognition (eigenfaces) [23]. This method constructs a B^2 dimensional basis (for blocks with B rows and B columns) based on the covariance matrix of the block pixels. Each block can be represented as a vector in the space spanned by the constructed basis [24]. However, it is assumed that the significant features of a block are contained in the first few dimensions. The significant block features are extracted by projecting the block pixels onto the first few dimensions of the constructed basis. Noise and other insignificant features are assumed to lie in the last few dimensions. Li et al. [12] decomposed the image using *discrete wavelet transform* (DWT). They computed the singular value decomposition (which is related to PCA) of the overlapping blocks from the low-frequency component. A related method by Zimba and Xingming [25] uses a combination of DWT and PCA. Shih and Yuan [20] proposed a simple method of using two features, the mean and variance of the gray values within the block, and showed the comparable performance to more advanced methods.

Fridrich et al. [6] considered a method based on *discrete cosine transform* (DCT) (DCT has also been used in more recent methods see [3,9]). DCT is frequently used in compression of multimedia, such as images (JPEG) and music (MP3). With DCT, similar to PCA, the more significant features are assumed to be captured in a few coefficients [11]. Luo et al. [14] extracted features based on color images. Mahdian and Saic [15] extracted features based on blur invariants. This allows for identification of copy-move forgery under blur degradation, additive Gaussian noise, and contrast changes. Amerini et al. [1] developed a SIFT-based method for copy-move attack detection and transformation recovery. Pan and Lyu [18] estimated the transform between matched SIFT keypoints and searched all pixels within the duplicated regions after discounting the estimated transforms.

Once features have been extracted from each block, they must be compared with one another. The issue is how to quantify the similarity between blocks. The most obvious comparison is an exhaustive search where every block is compared against other blocks. However, the performance of an exhaustive search is slow and takes $(N_b)(N_b - 1)/2$ comparisons to complete, where N_b is the total number of blocks and may be quite large.

Fridrich et al. [6] considered blocks as a match if their features are matched exactly. Popescu and Farid [19] used a nearest neighbor approach. That is, the *feature matrix* is lexicographically sorted so that blocks close to each other in the sorted *feature matrix* have similar features. Specifically, blocks within N_n rows of another block are considered a match. For example, if $N_n = 1$, then only the blocks 1-row away in the sorted *feature matrix* are considered a match. Singh and Raman [22] used the Graphical Processing Unit (GPU) to perform radix sorting of the feature vector and showed this can dramatically improve performance. Bayram et al. [2] computed the hashes of the vectors in the *feature matrix* and concluded two features matched exactly if their hashes matched. This technique avoids the need to sort the *feature matrix*.

Unfortunately, coming to a decision about which blocks are duplicated is not as simple as it may seem. Often two blocks are viewed as a match because they are near each other in an image or simply due to coincidence. Popescu and Farid [19] used a shift vector to determine whether two blocks are duplicated. Let the upper-left coordinates of the block corresponding to row i of the sorted *feature matrix* be denoted by (x_i, y_i). For two matching blocks, they computed the offset (s_x, s_y) as:

$$(s_x, s_y) = \begin{cases} (x_i - x_j, y_i - y_j) & \text{if} \quad x_i < x_j \\ (x_j - x_i, y_i - y_j) & \text{if} \quad x_i > x_j \\ (0, |y_i - y_j|) & \text{if} \quad x_i = x_j \end{cases} \tag{1}$$

Let $C(s_x, s_y)$ be the number of times that two blocks have the same offset (s_x, s_y). If a region is duplicated, then every block in that region is likely to have the same offset. If $C(s_x, s_y)$ is larger than some threshold, say N_s, then it is assumed that this offset must be due to a duplicated region. Furthermore, let N_d be a user-defined parameter, such that two blocks will not be considered as a match if the distance between the *i*th and *j*th block is smaller than N_d.

2. The proposed expanding block algorithm

Unlike the aforementioned methods, our proposed method primarily uses direct block comparison rather than indirect comparisons based on block features. The proposed method first divides an image into N_b small overlapping blocks just like in the sliding block method. However, our approach to comparing the blocks is different. Many of the blocks are obviously different and do not need to be compared against each other. A dominant feature is computed for each block. Specifically, we use the average gray value computed from all the pixels in the block as a dominant feature. As long as the dominant feature differs vastly between blocks, there is no need for comparison. Blocks are grouped together according to their dominant features. The blocks are sorted by dominant feature and placed evenly (or as evenly as possible) into *G* groups, each of which contains the blocks with a similar dominant feature. The first and last blocks in a group will also have a similar dominant feature to the blocks in the previous and next groups, respectively. To remedy this problem, we create *G* buckets so that the *i*th bucket contains the blocks from group *i*, group *i* – 1, and group *i* + 1. Each block will be placed into 3 buckets (except the blocks that are in the first and last groups which will only be placed into 2 buckets). Fig. 2 illustrates an example of how blocks are sorted and placed into buckets.

Blocks are compared only against other blocks in the same bucket. The comparisons are conducted using a statistical hypothesis test. Suppose that there are two blocks, *X* and *Y*, to be compared. Assume that each block has *b* pixels. Let the gray values from block *X* be $[p_{x1}, p_{x2}, \ldots, p_{xb}] = \boldsymbol{p_x}$, and from block *Y* be $[p_{y1}, p_{y2}, \ldots, p_{yb}] = \boldsymbol{p_y}$ (where $\boldsymbol{p_x}$ and $\boldsymbol{p_y}$ are vectors). Assume that the value of every pixel in an image can be represented by a true mean μ and an error term ε, where the error term is due to noise such as blurring, lossy compression, or other sources. The pixels from blocks *X* and *Y* can be expressed as follows (where $\boldsymbol{\mu_x}$, $\boldsymbol{\varepsilon_x}$, $\boldsymbol{\mu_y}$, and $\boldsymbol{\varepsilon_y}$ are vectors):

$$\begin{cases} \boldsymbol{p_x} = \boldsymbol{\mu_x} + \boldsymbol{\varepsilon_x} = [\mu_{x1} + \varepsilon_{x1}, \mu_{x2} + \varepsilon_{x2}, \ldots, \mu_{xb} + \varepsilon_{xb}] \\ \boldsymbol{p_y} = \boldsymbol{\mu_y} + \boldsymbol{\varepsilon_y} = [\mu_{y1} + \varepsilon_{y1}, \mu_{y2} + \varepsilon_{y2}, \ldots, \mu_{yb} + \varepsilon_{yb}] \end{cases} \tag{2}$$

The null and alternative hypotheses for comparing blocks *X* and *Y* can be formulated as:

$$\begin{cases} \mathbf{H_0} : \boldsymbol{\mu_x} = \boldsymbol{\mu_y} \\ \mathbf{H_1} : \boldsymbol{\mu_x} \neq \boldsymbol{\mu_y} \end{cases} \tag{3}$$

That is, the null hypothesis ($\mathbf{H_0}$) states that the only difference between the blocks is due to the error term, whereas the alternative hypothesis ($\mathbf{H_1}$) describes that for at least one pixel there exists a real difference.

We assume that the error terms are independent Gaussians with mean zero and variance σ^2. Let the test statistic be

$$t = \frac{(\boldsymbol{p_x} - \boldsymbol{p_y})^T(\boldsymbol{p_x} - \boldsymbol{p_y})}{\sigma^2/b} \tag{4}$$

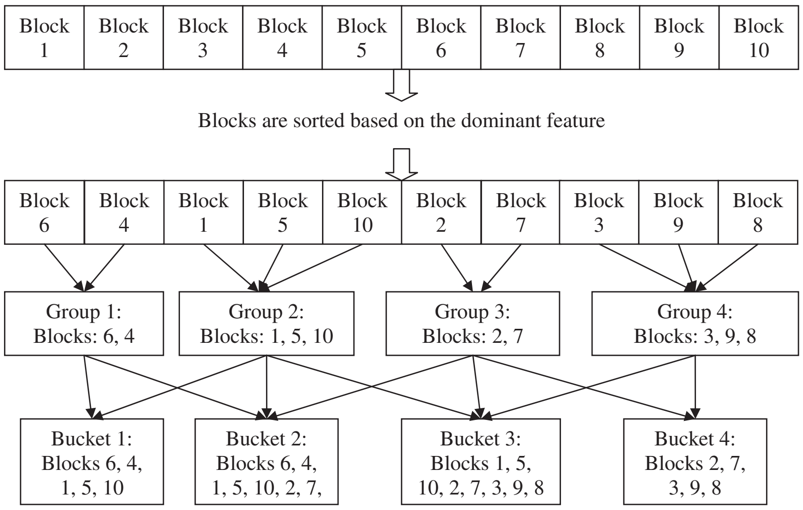

Block 1
Block 2
Block 3
Block 4
Block 5
Block 6
Block 7
Block 8
Block 9
Block 10
Blocks are sorted based on the dominant feature
Block 6
Block 4
Block 1
Block 5
Block 10
Block 2
Block 7
Block 3
Block 9
Block 8
Group 1: Blocks: 6, 4
Group 2: Blocks: 1, 5, 10
Group 3: Blocks: 2, 7
Group 4: Blocks: 3, 9, 8
Bucket 1: Blocks 6, 4, 1, 5, 10
Bucket 2: Blocks 6, 4, 1, 5, 10, 2, 7,
Bucket 3: Blocks 1, 5, 10, 2, 7, 3, 9, 8
Bucket 4: Blocks 2, 7, 3, 9, 8

Fig. 2. An example of how blocks are sorted and placed into buckets.

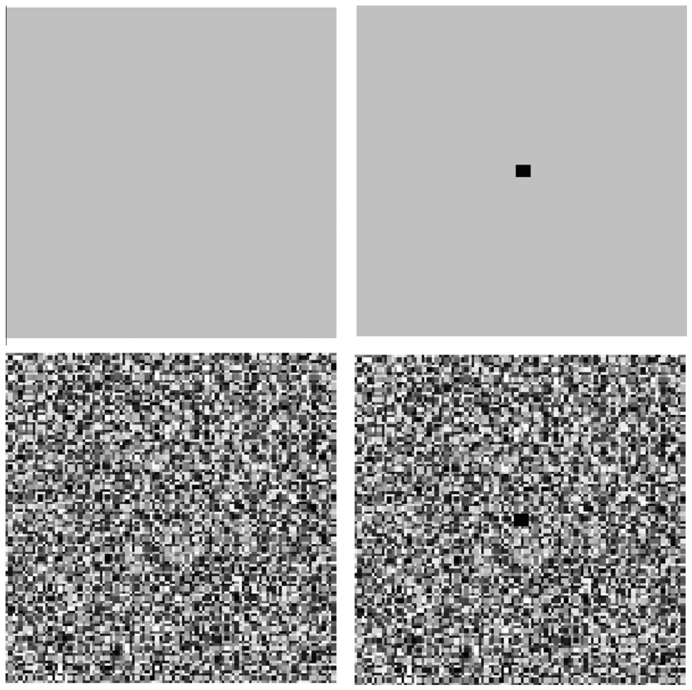


Fig. 3. The first set of images (top row) is two images where the difference is easily spotted because of lacking interesting features. The second set of images (bottom row) is much more interesting and the difference is much harder to spot.

Under the null hypothesis, the test statistic follows a chi-square distribution with b degrees of freedom [4,7]. If $Pr(\chi^2 > t)$ (where χ^2 follows a chi-square distribution with b degrees of freedom) is close to 0, then it can be concluded that $\mathbf{H_1}$ is true and the blocks are truly different. In other words, if the test statistic, t, is small, it provides evidence that the two blocks are the same; otherwise, they are different. σ^2 can be estimated based on the pooled variance of $\boldsymbol{p_x}$ and $\boldsymbol{p_y}$ as

$$\sigma^2 = \frac{\mathrm{var}(\boldsymbol{p_x}) + \mathrm{var}(\boldsymbol{p_y})}{2} = \frac{\sum_{i=1}^{b} p_{ix}^2 - \frac{1}{b}\left(\sum_{i=1}^{b} p_{ix}\right)^2 + \sum_{i=1}^{b} p_{iy}^2 - \frac{1}{b}\left(\sum_{i=1}^{b} p_{iy}\right)^2}{2b-2} \tag{5}$$

It would be beneficial to take a further look at the test statistic t. One component of the denominator is the variance. This can loosely be interpreted as the "interestingness" of the block. Fig. 3 presents two sets of similar images with the only difference being that one image has a black square in the middle. The difference in the first set of images can be seen quite easily because the first set of images is rather dull. On the other hand, the difference between the second set of images is much harder to spot because the images are much more interesting. This simple example is in agreement with what the formula for the test statistic is expressing. If two blocks are not interesting (i.e., they have low variance), then any small difference will be magnified and the test statistic will be large. If two blocks are very interesting (i.e., they have high variance), then small differences are acceptable and will not result in a large test statistic.

To perform the block comparisons, all blocks are compared with other blocks in the bucket. A block can be eliminated from the bucket as soon as there is enough information to conclude that it does not match any other block in the bucket. Comparing an entire block with another entire block is slow and unnecessary. Instead, a small region of each block is compared. If the small region of the block does not match the small region in any other block, then the block can be eliminated from the bucket. Once every block has been tested for matching on the small region, the region can be expanded and the comparisons are continued. When the comparison region is expanded, many of the blocks will have been eliminated and the size of the bucket will be significantly smaller.

In addition to block comparisons, it is assumed that the duplicated blocks will not overlap so that any blocks that are too close are considered as non-matching. Furthermore, it is assumed that the duplicated region is larger than some minimum size so that if the algorithm identifies a duplicated region smaller than this minimum size, it is disregarded. In this paper, we assume the duplicated region is at least 576 pixels corresponding to a 24×24 pixel square. Prior to presenting the proposed algorithm, the parameters are first introduced as follows.

blockSize: the width and height of a block, so that the total block size is *blockSize* $\times$ *blockSize*
numBuckets: the number of buckets used to compare blocks
pvalThreshold: a value between 0 and 1 used for the probability threshold for comparing blocks
minArea: a value denoting the minimum area of the duplicated region.

The proposed expanding block algorithm is presented below.

1. Divide an image into small overlapping blocks of size *blockSize* × *blockSize*.
2. For each block, compute the average gray value as the dominant feature.
3. Sort the blocks based on the dominant feature.
4. From the sorted blocks, place the blocks evenly into *numBuckets* groups.
5. Create *numBuckets* buckets. Place the blocks from groups $i-1$, i, and $i+1$ into bucket i.
6. Start with $S = 2$ where S represents the size of the current comparison region: $S \times S$ pixels. Process every bucket as follows:
    a. Suppose there are N blocks in the bucket. Construct an $N \times N$ matrix called the *connection matrix*. It denotes which blocks match each other. Initially, set the *connection matrix* to all ones so that all blocks match each other.
    b. If two blocks are less than *blockSize* pixels away, then the two blocks overlap. Set the *connection matrix* to 0 for these blocks.
    c. Compute the test statistic for every pairwise comparison for the upper-right $S \times S$ square of every block in the bucket. If for some block comparison, $Pr(\chi^2 > t)$ (where χ^2 follows a chi-square distribution with S^2 degrees of freedom) is less than *pvalThreshold*, then set the connection in the *connection matrix* to 0 for these blocks.
7. For each bucket, if the *connection matrix* has a row of zeros, then the block corresponding to this row is not connected to any other block in the bucket. Remove this block from the bucket.
8. If $S < blockSize$, then repeat step 6 with $S = \min(S^2, blockSize)$.
9. From the remaining blocks in the buckets, compute the total area. If the total area is less than *minArea*, then discard the remaining blocks; otherwise, the remaining blocks are assumed to be part of the duplicated region.

2.1. Enhanced expanding block algorithm

The expanding block algorithm can be enhanced to capture more sophisticated forgeries. Suppose that a forger modifies the duplicated region by adding or subtracting a small value to each pixel. The duplicated region will be lighter or darker than the original region, but if it is done carefully, then the difference would be invisible. By doing this, the forgery can evade detection by many forgery detection algorithms.

There are two parts of the algorithm that need to be modified. The first part is in selecting a dominant feature. Clearly, the average pixel value will not work since the forged region will have a higher or lower average pixel value and the forged blocks will not be placed in the same bucket as the original blocks. Instead, the variance is used as the dominant feature since it is not affected by constants. That is, $\mathrm{var}(\boldsymbol{p_x}) = \mathrm{var}(\boldsymbol{p_x} + c)$, where c is some constant.

The second part is changing the comparison procedure. Note that the null and alternative hypotheses need to be modified for some constant c as:

$$\begin{cases} \mathbf{H_0} : \boldsymbol{\mu_x} = \boldsymbol{\mu_y} + c \\ \mathbf{H_1} : \boldsymbol{\mu_x} \neq \boldsymbol{\mu_y} + c \end{cases} \tag{6}$$

Again, we assume that the error terms are independent Gaussians with mean zero and variance σ^2. Let the test statistic be:

$$t = \frac{(\boldsymbol{p_x} - \boldsymbol{p_y} - \mathbf{c})^T(\boldsymbol{p_x} - \boldsymbol{p_y} - \mathbf{c})}{\sigma^2/b} \tag{7}$$

Under the null hypothesis, the test statistic follows a chi-square distribution with b degrees of freedom. It is assumed that c is unknown, and therefore the most conservative (smallest) test statistic over all values of c is used:

$$t = \min_c \frac{(\boldsymbol{p_x} - \boldsymbol{p_y} - \mathbf{c})^T(\boldsymbol{p_x} - \boldsymbol{p_y} - \mathbf{c})}{\sigma^2/b} \tag{8}$$

It can be shown that the smallest value of the test statistic occurs when c is equal to the mean of block X minus the mean of block Y.

Fig. 4b shows an example of a forged image where the forged region has been made slightly lighter than the region it was copied from. Fig. 4c highlights the copied regions. Even though the two regions are slightly different, it is very difficult to tell that there is any difference between them.

2.2. Applying the expanding block algorithm on color images

Up until now, it has been assumed that the image has one color channel (gray). However, the algorithm can be easily extended to work with color images that have three color channels (red, green, and blue). To start, the blocks are sorted based on the dominant feature. A good choice for the dominant feature in color images is the average intensity ((RED + GREEN + BLUE)/3) of the pixels in the block. Next, block comparisons are done with each color channel sequentially

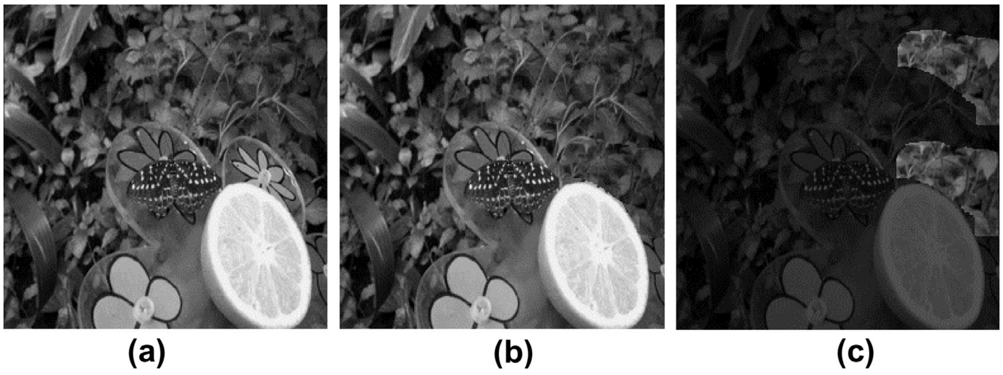


Fig. 4. (a) The original image, (b) the forged image where the copied region is slightly lighter than the region it was copied from, and (c) the highlighted copied region.

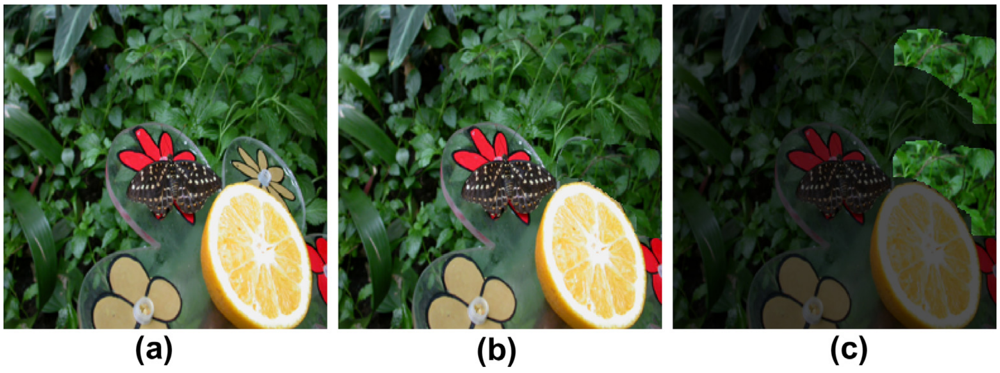


Fig. 5. (a) The original image, (b) the forged image, and (c) the highlighted copied region.

Table 1
The number of false and true positives for testing *numBuckets*.

numBuckets	Blur 0	Blur 3	Blur 5	Blur 7	Blur 10	Blur 15
False positives						
256	6	7	6	6	5	6
512	6	7	6	6	5	6
1024	6	7	6	6	5	6
2048	5	6	5	5	5	5
4096	4	4	4	4	5	4
True positives						
256	100 (100)	88.5 (97)	84.7 (97)	82.5 (95)	82.2 (95)	82.5 (95)
512	100 (100)	88.5 (97)	84.7 (97)	82.4 (95)	82.1 (95)	82.4 (95)
1024	100 (100)	88.4 (97)	84.4 (97)	81.4 (94)	81.0 (94)	81.4 (94)
2048	100 (100)	88.2 (97)	82.0 (95)	78.7 (92)	79.5 (93)	78.7 (92)
4096	100 (100)	88.1 (97)	79.9 (94)	77.6 (92)	77.8 (92)	77.6 (92)

Table 2
The average performance time in seconds.

numBuckets	256	512	1024	2048	4096
Seconds	13.81	7.53	7.03	6.89	9.33

Table 3
The number of false and true positives for testing *pvalThreshold*.

pvalThreshold	Blur 0	Blur 3	Blur 5	Blur 7	Blur 10	Blur 15
False positives						
0.5	5	5	6	6	5	6
0.75	4	5	6	6	5	6
0.9	4	5	6	6	5	6
0.95	4	5	5	6	5	5
0.99	4	4	4	4	5	4
True positives						
0.5	100 (100)	88.3 (97)	80.6 (94)	79.4 (93)	78.9 (92)	79.3 (93)
0.75	100 (100)	88.2 (97)	80.5 (94)	79.2 (93)	78.6 (92)	79.2 (93)
0.9	100 (100)	88.2 (97)	80.2 (94)	78.1 (92)	78.3 (92)	78.1 (92)
0.95	100 (100)	88.2 (97)	80.1 (94)	78.0 (92)	78.2 (92)	78.0 (92)
0.99	100 (100)	88.1 (97)	79.9 (94)	77.6 (92)	77.8 (92)	77.6 (92)

Table 4
The performance time for the different values of *pvalThreshold*.

pvalThreshold	0.5	0.75	0.9	.95	0.99
Seconds	9.00	8.58	8.09	7.82	7.43

Table 5
Comparison of the methods where the images have not been modified.

% of True Positives:	%
DCT	100
Statistical	100
PCA	100
EB	100
False Positives:	Number of Images
DCT	1
Statistical	2
PCA	0
EB	3
% of copied Region Correctly Identified	%
DCT	100
Statistical	100
PCA	100
EB	100
Pixels Incorrectly Identified as Copies:	Number of Pixels
DCT	633.21
Statistical	784.4
PCA	305.81
EB	85.12

on a small portion of the block. That is, the comparisons start with a small portion of the block on the red color channel and eliminate any non-matching blocks. After this is complete, the comparisons proceed to process the green color channel and then the blue color channel. Finally, the comparison region is expanded. Fig. 5 shows an example of a forged color image and the identified copied region.

3. Experimental results

This section is divided into four subsections. The first will investigate the effect of changing different parameters in the expanding block algorithm. The second and third subsection will compare the algorithm with other existing algorithms. The last subsection will evaluate the performance of the modified expanding block algorithm. All measurements are performed on a Lenovo laptop with a 2.1 GHz Intel Pentium processor and 8 GB of RAM.

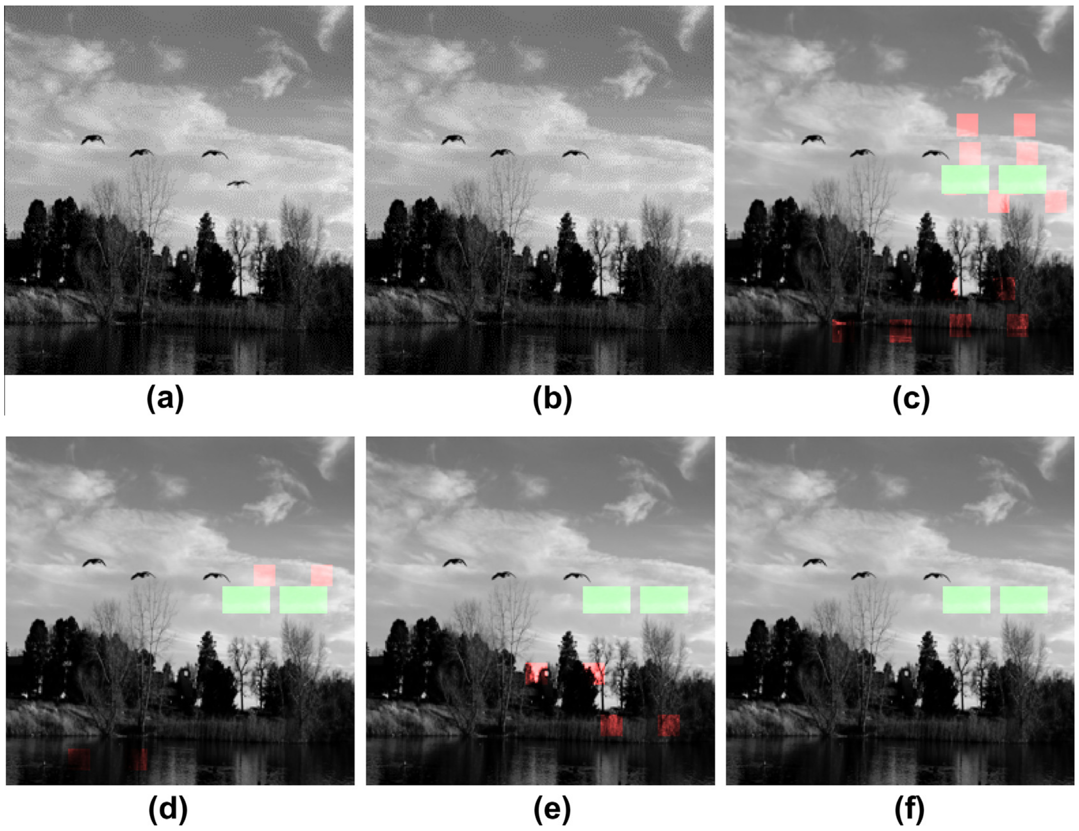


Fig. 6. (a) The original image, (b) the forged image, (c) DCT copy move detection, (d) statistical copy move detection, (e) PCA copy move detection, and (f) expanding block copy move detection.

Table 6
The performance time of different methods.

	DCT	PCA	Statistical	EB
Seconds	2.23	22.41	4.98	10.32

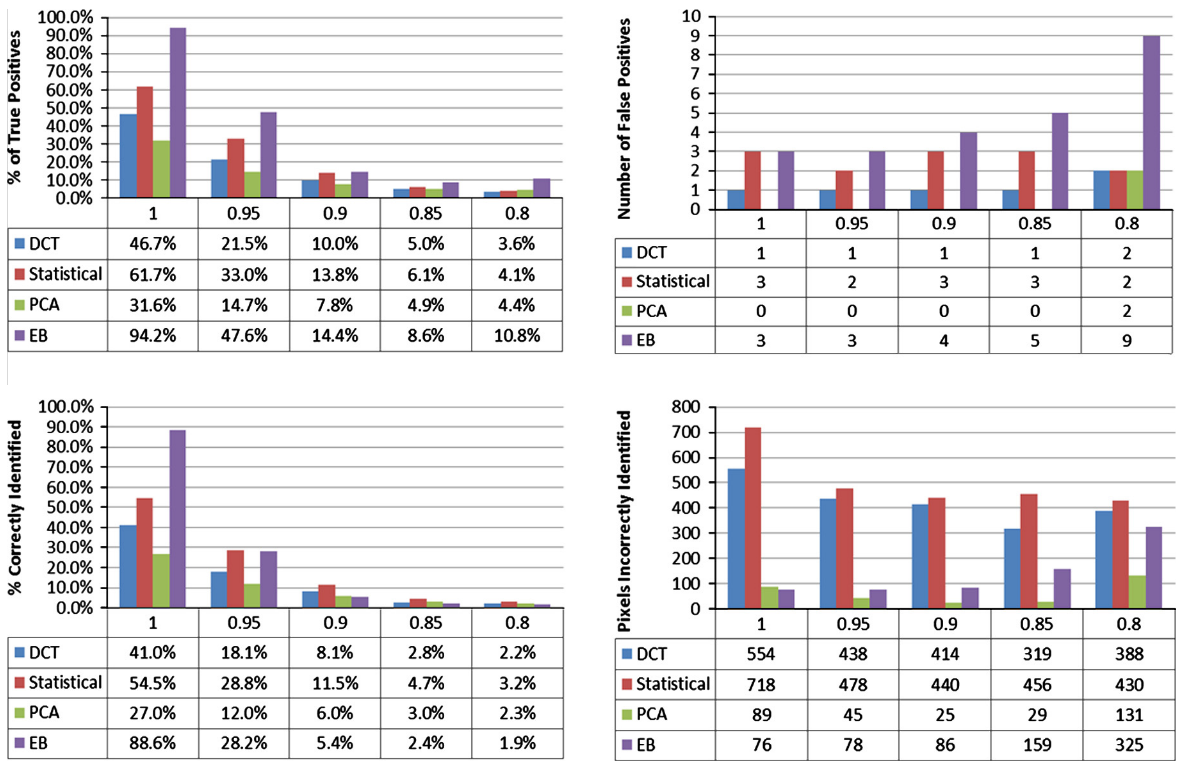
3.1. Adjusting the parameters

There are two parameters to be investigated: *numBuckets* and *pvalThreshold*. The parameters *blockSize* and *minArea* represent basic assumptions about the copied region and will not be investigated. Several values of each parameter will be tested to see the effect on the identification of the forged and non-forged images as well as their performance time. The test image set consists of 100 grayscale images of size 256×256 that did not contain a duplicated region. The experiments stepped through all 100 images under various degrees of Gaussian blur (0×0, labeled Blur 0; 3×3, labeled Blur 3; etc.). Forged images were generated by randomly copying a 64×64 pixel square into a non-overlapping region within the same image. In all tests, *blockSize* is set to 16 and *minArea* is set to 576 (i.e. 24×24).

3.1.1. Parameter numBuckets

The first test will investigate the value of *numBuckets*. The test will step through values of 256, 512, 1024, 2048, and 4096. The value of *pvalThreshold* will be fixed at 0.99. Table 1 displays the results when there is no duplicated region (false positive) and when there is a duplicated region (true positive). The number of false positives means the number of images where the algorithm identifies a forged region even though the image did not have one. The number of true positives (enclosed in parentheses) means the number of images where the algorithm correctly identifies a forged region. The percentage of the duplicated region that the algorithm identifies as being duplicated is also shown without parentheses.

The results are fairly promising. Although there are some false positives, the algorithm correctly identifies most images that contain a duplicated region. Moreover, it was able to identify a large percentage of the duplicated region. Table 2 shows the average performance time in seconds. When *numBuckets* is set to 256, the performance time is by far the longest. Arguably, the best overall results are when *numBuckets* = 4096 since it has the fewest false positives and a few less true positives.



	1	0.95	0.9	0.85	0.8
🟦 DCT	46.7%	21.5%	10.0%	5.0%	3.6%
🟥 Statistical	61.7%	33.0%	13.8%	6.1%	4.1%
🟩 PCA	31.6%	14.7%	7.8%	4.9%	4.4%
🟪 EB	94.2%	47.6%	14.4%	8.6%	10.8%

	1	0.95	0.9	0.85	0.8
🟦 DCT	1	1	1	1	2
🟥 Statistical	3	2	3	3	2
🟩 PCA	0	0	0	0	2
🟪 EB	3	3	4	5	9

	1	0.95	0.9	0.85	0.8
🟦 DCT	41.0%	18.1%	8.1%	2.8%	2.2%
🟥 Statistical	54.5%	28.8%	11.5%	4.7%	3.2%
🟩 PCA	27.0%	12.0%	6.0%	3.0%	2.3%
🟪 EB	88.6%	28.2%	5.4%	2.4%	1.9%

	1	0.95	0.9	0.85	0.8
🟦 DCT	554	438	414	319	388
🟥 Statistical	718	478	440	456	430
🟩 PCA	89	45	25	29	131
🟪 EB	76	78	86	159	325

Fig. 7. The performance of different algorithms under JPEG compression.

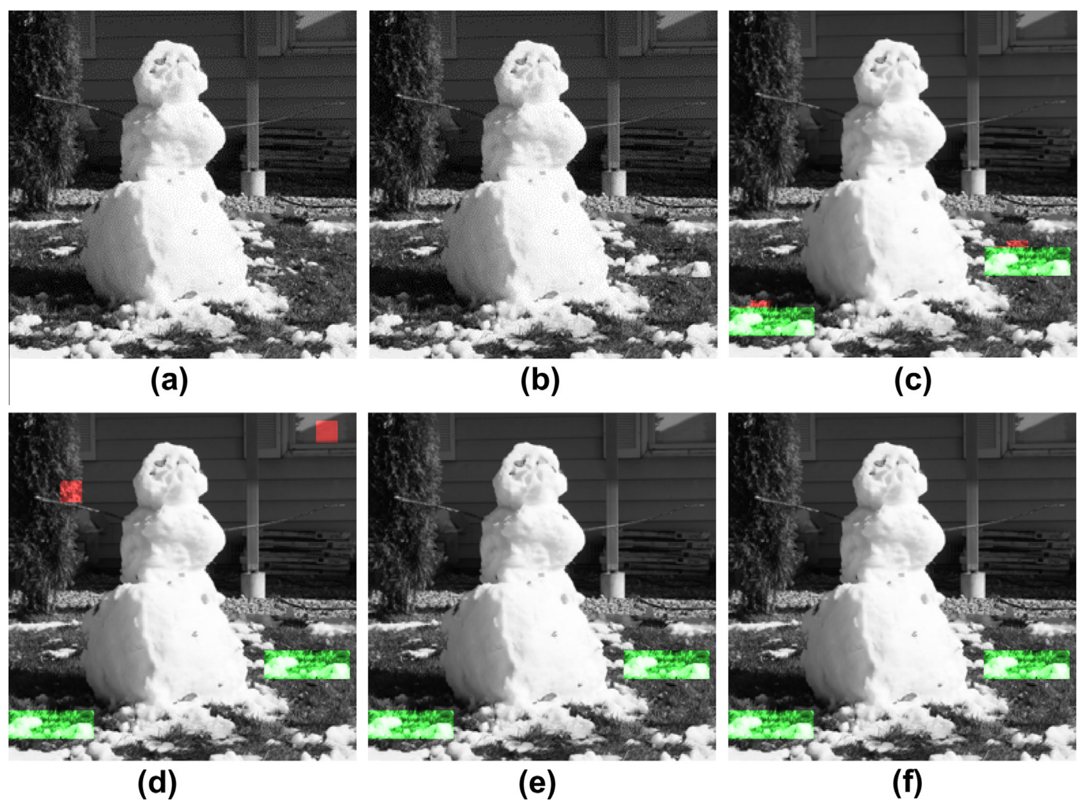
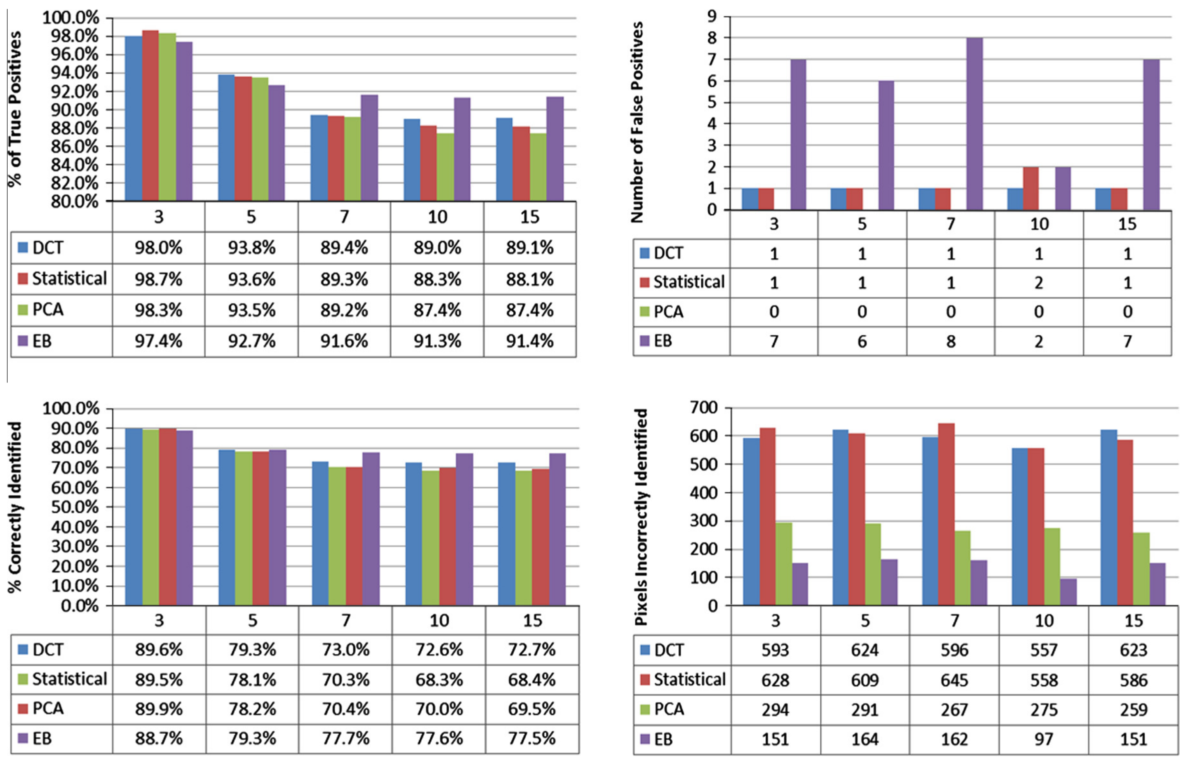


Fig. 8. (a) The original image, (b) the forged image with JPEG compression ratio 0.95, (c) DCT copy-move detection, (d) statistical copy-move detection, (e) PCA copy-move detection, and (f) expanding block copy-move detection.



	3	5	7	10	15
🟦 DCT	98.0%	93.8%	89.4%	89.0%	89.1%
🟥 Statistical	98.7%	93.6%	89.3%	88.3%	88.1%
🟩 PCA	98.3%	93.5%	89.2%	87.4%	87.4%
🟪 EB	97.4%	92.7%	91.6%	91.3%	91.4%

	3	5	7	10	15
🟦 DCT	1	1	1	1	1
🟥 Statistical	1	1	1	2	1
🟩 PCA	0	0	0	0	0
🟪 EB	7	6	8	2	7

	3	5	7	10	15
🟦 DCT	89.6%	79.3%	73.0%	72.6%	72.7%
🟩 Statistical	89.5%	78.1%	70.3%	68.3%	68.4%
🟥 PCA	89.9%	78.2%	70.4%	70.0%	69.5%
🟪 EB	88.7%	79.3%	77.7%	77.6%	77.5%

	3	5	7	10	15
🟦 DCT	593	624	596	557	623
🟥 Statistical	628	609	645	558	586
🟩 PCA	294	291	267	275	259
🟪 EB	151	164	162	97	151

Fig. 9. The performance of different algorithms under the effect of Gaussian blurring.

3.1.2. Parameter pvalThreshold

Next *pvalThreshold* will be investigated. The test will step through values of 0.5, 0.75, 0.9, 0.95, and 0.99. The value of *numBuckets* will be fixed at 4096. Similar to Table 1, Table 3 shows the number of false and true positives for 100 images.

When *pvalThreshold* is 0.5, the algorithm has the largest number of false negatives. The fewest false negatives are observed when *pvalThreshold* is 0.99. The performance for identifying the duplicated regions is fairly similar for all values of *pvalThreshold*. Table 4 shows the performance time for the different values of *pvalThreshold*. As expected, a higher *pvalThreshold* leads to faster performance time since connections will be set to 0 more easily.

3.2. Comparison with other algorithms

In this section, the performance of the expanding block algorithm (labeled EB) is compared with three other algorithms, which are based on the sliding block algorithm using nearest neighbor comparison: discrete-cosine transform (labeled DCT), principal component analysis (labeled PCA), and statistical extraction (labeled statistical). Every comparison will include the number of true positives, percentage of duplicated region identified, and false positives. In addition, we would like to know how accurately the algorithm can identify the location and shape of the duplicated region. To this end, the algorithms will be measured on the average number of pixels incorrectly identified as a copied region when the image contains a duplicated region.

In the first set of experiments, a separate set of 100 grayscale images of size 256×256 was used for the comparisons. The block size was set to 16, nearest neighbor to 1, and minimum distance (N_d) to 10. The copied region was assumed to be at least 24×24, so the shift threshold for the sliding block algorithm was set to 64. *numBuckets* was set to 4096 and *pvalThreshold* was set to 0.99. To test the algorithms with forgeries, a square of random size from 24×24 to 64×64 was copied and pasted into a non-overlapping random location within the same image. This was done 10 times for every image for a total of 1000 forged images (10 different forgeries for each of 100 images) and 100 non-forged images.

3.2.1. Varying the copied region size

Table 5 shows the first test of a basic comparison of the methods involving the original images and the forged images without any additional modifications. While the expanding block algorithm has the most false positives with 3, it is by far the best in correctly identifying the forged region at an average of only 85 pixels incorrectly. Fig. 6 shows an example of a copy-move forgery and the results from the four detection methods. The correctly identified portions of the image are shaded green and the incorrect portions are shaded red. Only the expanding block algorithm is able to correctly identify the forged areas with no mistakes.

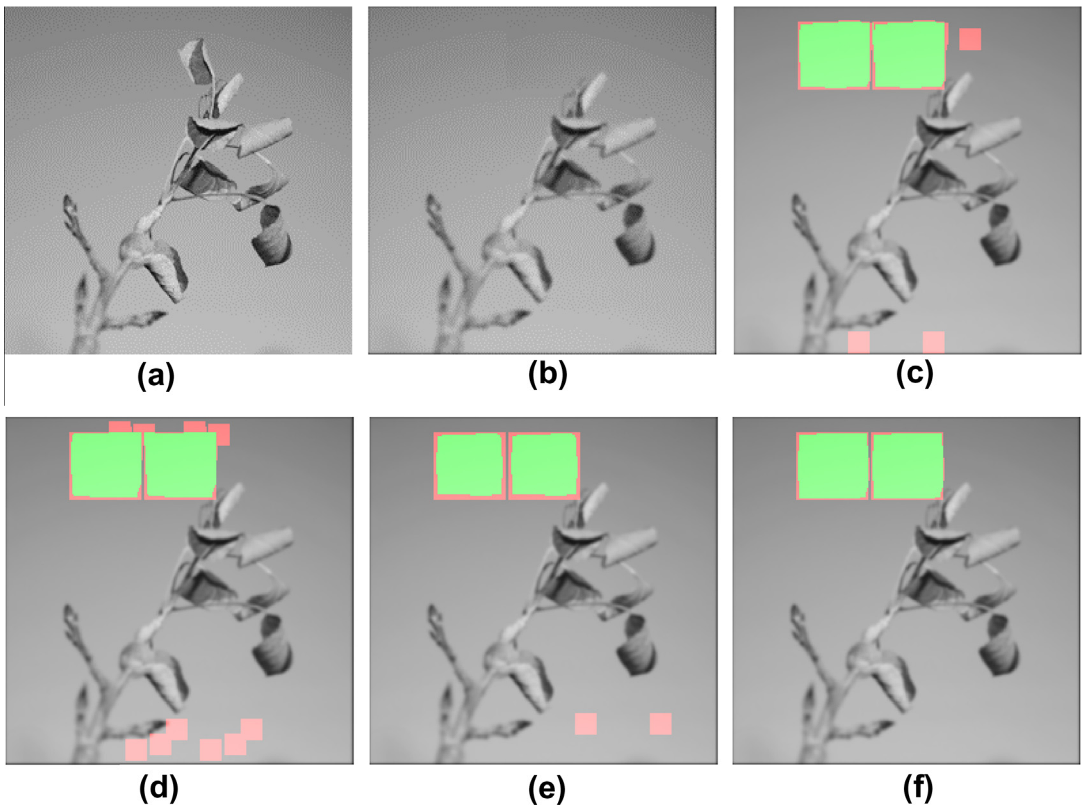
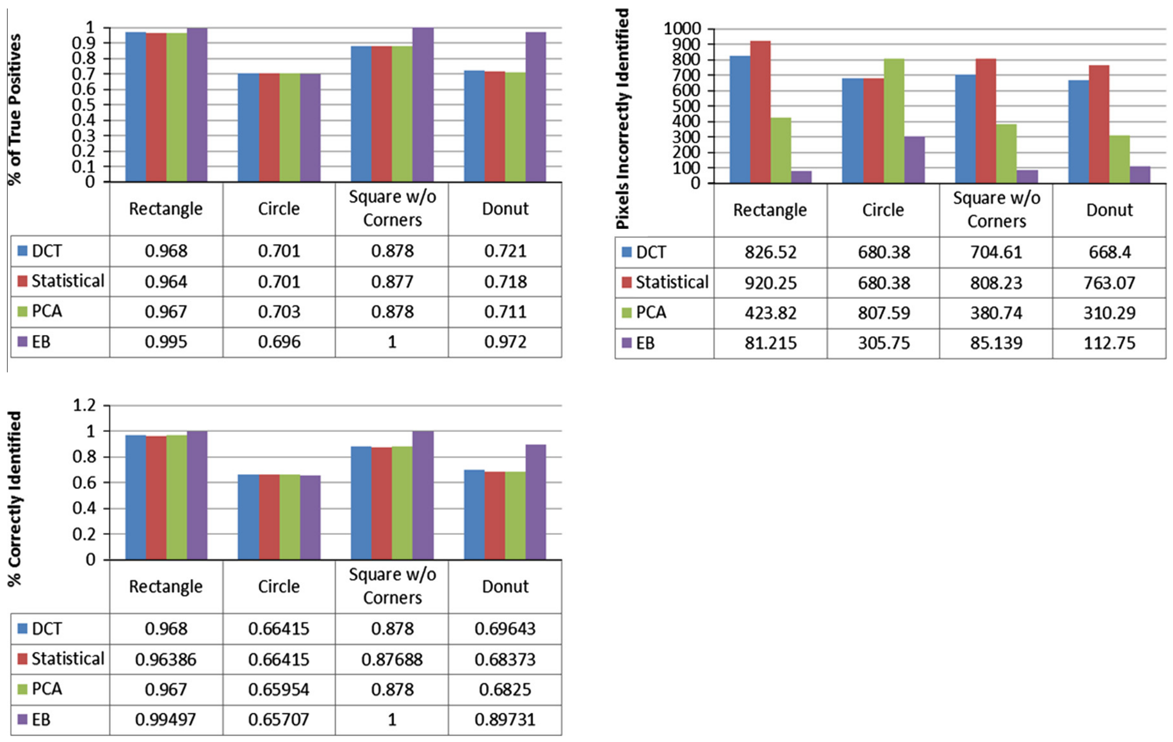


Fig. 10. (a) The original image, (b) the forged image with 10 × 10 Gaussian blurring, (c) DCT copy-move detection, (d) statistical copy-move detection, (e) PCA copy-move detection, and (f) expanding block copy-move detection.



	Rectangle	Circle	Square w/o Corners	Donut
🟦 DCT	0.968	0.701	0.878	0.721
🟥 Statistical	0.964	0.701	0.877	0.718
🟩 PCA	0.967	0.703	0.878	0.711
🟪 EB	0.995	0.696	1	0.972

	Rectangle	Circle	Square w/o Corners	Donut
🟦 DCT	826.52	680.38	704.61	668.4
🟥 Statistical	920.25	680.38	808.23	763.07
🟩 PCA	423.82	807.59	380.74	310.29
🟪 EB	81.215	305.75	85.139	112.75

	Rectangle	Circle	Square w/o Corners	Donut
🟦 DCT	0.968	0.66415	0.878	0.69643
🟥 Statistical	0.96386	0.66415	0.87688	0.68373
🟩 PCA	0.967	0.65954	0.878	0.6825
🟪 EB	0.99497	0.65707	1	0.89731

Fig. 11. The performance of different algorithms for irregular shaped regions.

Table 7
The testing results of the enhanced expanding block algorithm.

	−5	−2	−1	0	+1	+2	+5
% of True positives	100	100	100	100	100	100	100
% correctly identified	100	100	100	100	100	100	100
Pixels incorrectly identified	74.1	74.1	74.4	75.6	74.4	73.7	73.8

Table 6 shows the performance time of different methods. It is observed that the performance time of the expand block algorithm is slower than DCT and statistical, but much faster than PCA.

3.2.2. JPEG compression

This set of tests will investigate the performance of the expanding block algorithm under JPEG compression. In Fig. 7, the compression ratios used are 1, 0.95, 0.9, 0.85 and 0.8, where compression ratio is the compressed file size divided by the original file size. In all cases, the expanding block algorithm has the most true positives. The algorithm does very well when the JPEG compression ratio is on the larger side. However, when the JPEG compression ratio becomes smaller, the expanding block algorithm has more false positives.

Fig. 8 presents an example of a copy-move forgery and the results from the four detection methods under JPEG compression with compression ratio 0.95. All four methods are able to identify the forged region; however, only PCA detection and the expanding block algorithm are able to identify the forgery with no mistakes.

3.2.3. Varying the image blurring

Fig. 9 shows the test for the effect of Gaussian blurring. The blurring uses various sizes ranging from 3×3 to 15×15.

In terms of identifying the images that have duplicated regions, the expanding block algorithm does very well. It either does as good as other algorithms or much better. When the image has a forgery, the expanding block algorithm incorrectly identifies far fewer pixels as forgeries than the other algorithms. However, the expanding block algorithm tends to have more false positives than the other algorithms.

Fig. 10 shows an example of a copy-move forgery and the results from the four detection methods under 10×10 Gaussian blurring. None of the methods are able to completely identify the forged region; however, the expanding block algorithm identifies the largest portion of the forged region and has the least mistakes.

3.3. Comparisons with irregularly shaped regions

To evaluate the robustness of our proposed procedure, we investigated the performance of the expanding block algorithm with non-square duplicated regions. We chose four different shapes, a rectangle with a minimum length of 16 pixels, a square with a 20×20 pixel section of the upper-left and lower-right corner removed, labeled Square w/o Corners (the shape was also used in [16]), a circle, and a donut with an inner radius of 5 pixels. As in the previous section, 10 different forgeries for each of the 100 images were created for each shape using a random area of 576 (24×24) to 4096 (64×64). Fig. 11 shows the results. With the exception of the circle shape, the expanding block algorithm performed better than the other methods in terms of the percentage of forgeries identified, particularly for the square without corners and the donut shape. Remarkably, the expanding block algorithm had by far the least number of pixels incorrectly identified for every shape considered.

3.4. Results for the enhanced expanding block algorithm

Table 7 shows the testing results of the enhanced expanding block algorithm. The algorithm is tested on a set of 100 grayscale images of size 256×256, where each image contains 10 different forgeries. Like the above tests, a square of random size from 24×24 to 64×64 is copied. However, before pasting it back into the image, the copied region is made darker or lighter by a small amount, so that the original and forged regions differ by a small shade. Specifically, the forged region is made lighter by a value of 5, 2, and 1, darker by 5, 2, and 1, and no change. Other algorithms are not included in the test since the other algorithms are not designed to catch this specific type of forgery. For this test, *numBuckets* is set to 4096 and *pvalThreshold* is set to 0.99. The enhanced expanding block algorithm correctly identifies every image and is very accurate in identifying the shape and location of the forged region.

4. Conclusions

It has been shown that the expanding block algorithm is an effective method for identifying image copy-move forgery. It is particularly good at identifying the location and shape of the forged regions. It is also shown that the expanding block algorithm is able to catch specific types of forgeries, such as under JPEG compression, the effect of Gaussian blurring, or when the duplicated region is made lighter or darker. The advantage of the expanding block algorithm is that it can handle block comparison methods that require two blocks to be directly compared with each other (as opposed to nearest neighbor which only allows indirect block comparison based on the blocks' features). As shown in this paper, direct block comparison can be done without a large sacrifice in performance time.

References

[1] I. Amerini, L. Ballan, R. Caldelli, A. Del Bimbo, G. Serra, A SIFT-based forensic method for copy-move attack detection and transformation recovery, IEEE Trans. Inform. Forensics Secur. 6 (3) (2011) 1099–1110.
[2] S. Bayram, H.T. Sencar, N. Memon, A survey of copy-move forgery detection techniques, in: Proc. IEEE Western New York Image Processing Workshop, Rochester, NY, 2008.
[3] Y. Cao, T. Gao, L. Fan, Q. Yangl, A robust detection algorithm for copy-move forgery in digital images, Forensic Sci. Int. 214 (2012) 33–43.
[4] G. Casella, R.L. Berger, Statistical Inference, second ed., Duxbury Press, 2001.
[5] V. Christlein, C. Riess, J. Jordan, C. Riess, E. Angelopoulou, An evaluation of popular copy-move forgery detection approaches, IEEE Trans. Inform. Forensics Secur. 7 (6) (2012) 1841–1854.
[6] J. Fridrich, D. Soukal, J. Lukáš, Detection of copy-move forgery in digital images, in: Proc. Digital Forensic Research Workshop, Cleveland, OH, August, 2003.
[7] R.V. Hogg, A. Craig, J.W. McKean, Introduction to Mathematical Statistics, sixth ed., Pearson Prentice Hall, Upper Saddle River, NJ, 2004.
[8] J.-H. Hsiao, C.-S. Chen, L.-F. Chien, M.-S. Chen, A new approach to image copy detection based on extended feature sets, IEEE Trans. Image Process. 16 (8) (2007) 2069–2079.
[9] Y. Huang, W. Lu, W. Sun, D. Long, Improved DCT-based detection of copy-move forgery in images, Forensic Sci. Int. 206 (2011) 178–184.
[10] A. Khana, S.A. Malika, A. Alib, R. Chamlawia, M. Hussaina, M.T. Mahmoodc, I. Usmand, Intelligent reversible watermarking and authentication: hiding depth map information for 3D cameras, Inform. Sci. 216 (2012) 155–175.
[11] S.A. Khayam, The Discrete Cosine Transform (DCT): Theory and Application, Technical Report WAVES-TR-ECE802.602, Michigan State University, 2003.
[12] G. Li, Q. Wu, D. Tu, S. Sun, A sorted neighborhood approach for detecting duplicated regions in image forgeries based on DWT and SVD, in: Proc. IEEE International Conference on Multimedia and Expo, Beijing, China, 2004, pp. 1750–1753.
[13] H. Ling, H. Cheng, Q. Ma, F. Zou, W. Yan, Efficient image copy detection using multiscale fingerprints, IEEE Multimedia 19 (1) (2012) 60–69.
[14] W. Luo, J. Huang, G. Qui, Robust detection of region-duplication forgery in digital image, in: Proc. International Conference on Pattern Recognition, Washington, DC, 2006, pp. 746–749.
[15] B. Mahdian, S. Saic, Detection of copy-move forgery using a method based on blur moment invariants, Int. J. Forensic Sci. 171 (2007) 180–189.
[16] H.C. Nguyen, S. Katzenbeisser, Security of copy-move forgery detection techniques, in: International Conference on Acoustics, Speech and Signal Processing, 2011, pp. 1864–1867.
[17] S. Nikolopoulos, S. Zafeiriou, N. Nikolaidis, I. Pitas, Image replica detection system utilizing R-trees and linear discriminant analysis, Pattern Recogn. 43 (3) (2010) 636–649.
[18] X. Pan, S. Lyu, Region duplication detection using image feature matching, IEEE Trans. Inform. Forensics Secur. 5 (4) (2010) 857–867.
[19] A.C. Popescu, H. Farid, Exposing Digital Forgeries by Detecting Duplicated Image Regions, Technical Report, TR2004-515, Dartmouth College, 2004.
[20] F.Y. Shih, Y. Yuan, A comparison study on copy-cover image forgery detection, Open Artif. Intell. J. 4 (2010) 49–54.
[21] F.Y. Shih (Ed.), Multimedia Security: Watermarking, Steganograph, and Forensics, CRC Press, Boca Raton, FL, 2012.
[22] J. Singh, B. Raman, A high performance copy-move image forgery detection scheme on GPU, in: Proceedings of International Conference on Soft Computing for Problem Solving, 2011, pp. 225–233.
[23] L. Sirovich, M. Kirby, Low dimensional procedure for the characterization of human faces, J. Opt. Soc. Am. A (1987) 519–524.
[24] G. Strang, Linear Algebra and Its Applications, fourth ed., Thomson Brooks/Cole, Belmont, CA, 2005.
[25] M. Zimba, S. Xingming, DWT-PCA (EVD) based copy-move image forgery detection, Int. J. Digital Content Technol. Appl. 5 (1) (2011) 251–258.