Geometric Transformation Invariant Copy-MoveForgery Detection in Images

Badal Soni, Sai Akhil, Surya Pratap, Raktim Sharma, Preeti Kumari[1]

Abstract

In the present world, digital images can be included among the prevalent ways for information sharing. The term information ranges from casual to highly confidential. There is a high chance of forging the image with the existing available technology to tamper the information. Forging is done by replacing the important portion of the image with some other part of the same image. As the copied part resides in the same image, its important properties, such as noise, brightness, texture is compatible with rest of the image making its detection very difficult. The afore-mentioned type of forging is generally defined as **copy-move forgery**. Till date many algorithms have been proposed in this field to detect the copy-move forged regions of the image. The algorithms vary from one to one depending on the way it processes the image pixels (Block based or Key-point based). All the proposed algorithms suffer from the problem of time-consumption.

The purpose of this study is to present an efficient copy-move forgery detection algorithm which uses ORB (Oriented FAST Rotated BRIEF) for key-point detection and **Trie data structure** for the purpose of matching and to reduce the overall computation time of detecting duplicated regions of the image. The proposed algorithm is tested on the data-sets 'Micc-F220' and 'Micc-F2000'. Experimental results on the data sets show the accuracy and ability to reduce the computational time, though it consumes memory proportional to the number of key points.

1. Introduction

It is obvious to suggest that images play a vital role as a source of information. The main purpose of forgery analysis is to check whether there is any manipulation in the content of an image. There are two generic techniques of copy-move forgery detection, namely block-based and key point based. Key point based technique is preferred among the two techniques, owing to its various advantages.

We have implemented a copy-move forgery detection system that is invariant to geometric transformations like scaling and rotation, and also compared its performance to similar existing efficient systems.

*1.1. **Key Contribution of the paper***

Almost all algorithms based on ORB take similar computational time for key point detection. But the matching part of the algorithm differs from each other. So, there is a certain scope to optimize matching part of the algorithm. We used trie data structure to hold the bit strings of the detected key points and then matching is done on the data it holds. This usage of data structure optimizes the matching part of the algorithm.

2. Related Work

Following is the explanation of various papers that have been considered before designing the algorithm :

The first block-based method for detecting copy-move forgery was introduced by **Fridrich et al.**[1]. Discrete Cosine Transformation (DCT) based features have been used. The proposed algorithm had several limitations against operations like blurring, rotation and scaling. Image is initially is divided into overlapping blocks by the sliding window moving by one pixel each time from top left corner to bottom right corner of the image. A matrix is formed with each row holding the pixel values of a particular block. Similar blocks are found using Lexicographical sorting of matrix. This phenomenon is known as exact

match method. By calculating DCT coefficients for each block, a robust method had been used. A quality factor parameter named Q-factor is selected for appropriate quantization of DCT coefficients. A matrix with each row holding the DCT coefficients of a particular block is lexicographically sorted. Similar blocks are found with the help of highly occurred shift vectors. A desired threshold is fixed which controls the minimal size of image segment that can be detected.

Huang et al.[2], proposed a Scale-invariant feature transform (SIFT) based detection method to identify duplicate regions with scale and rotation and then used the Best Bin First search (BBF) method to find the nearest neighbors with high probability that return the matched key points (inliers) as a possible duplicate region. To increase the accuracy of detection methods, a Nearest Neighbor Distance Ratio (NNDR) is applied for matched key-points.

In the paper published by **Rublee**[3] on ORB (Oriented FAST Rotated BRIEF), an efficient alternative to SIFT and SURF has explained the advantages of ORB in detail. A very trivial point for considering ORB to SIFT and SURF is that ORB is open source and can be implemented or customized as per the application. Whereas, SIFT and SURF are licensed (not Open Source). The theory behind BRIEF and ORB is much easier to understand compared to SIFT and SURF. Computing Hamming distance is lightning fast compared to Euclidean distance. ORB is fast compared to both SIFT and SURF which uses the Oriented FAST algorithm for key-point detection and Rotated BRIEF for key-point description which makes the ORB invariant to Geometric transformations like Scaling and Rotation.

In the paper published by **Bodon**[4], many customized data structures have been explained. Among all such data structures, trie has the nearest quality as a data structure for holding the data that which will be well suited for the purpose of matching the long length strings similar to our purpose of matching the key-point descriptors (we used binary string of 64 length). Trie data structure reduces the computational time complexity of matching n key-points, each of l length binary string, from O(n*n*l) to O((n+$2^{threshold}$)*l). Here, the first mentioned computational time is taken by brute-force method of searching and

latter is the computational time taken by the trie data structure. So, this plays a very vital role in matching phase of the detection algorithms as the number of key-points increase. This is because the computational time for brute-force method increases in quadratic fashion, whereas, trie increases linearly. So, this was the most considered point for implementing the matching phase using trie data structure.

3. Proposed Methodology

3.1. ***Proposed Algorithm***

3.1.1. ***Noise Reduction***

During the time of capturing images, some noise will be added to the images. So, this added noise must be always removed before implementing the algorithm. This phase of the algorithm plays an important role in detecting the forged regions. Sometimes, it can be even said that none of the forged regions would be detected if the noise reduction is not performed on the image. There are various algorithms for image smoothing namely average masks, weighted average masks, Statistical masks (Median, Min, Max). In the proposed algorithm GaussianBlur (OpenCV function) is used for smoothing the image (Noise Reduction).

3.1.2. ***RGB to Gray Scale Conversion***

Generally in RGB images each pixel is made of 24 bits (Red - 8 bits, Blue - 8 bits and Green - 8 bits), whereas, a pixel of a Gray scale image is made of 8 bits (range of a pixel is from 0 to 255). Calculating the gradient of a pixel of Gray scale image is simple and understandable compared to RGB image. The designed algorithm is implemented on Gray scale images.

3.1.3. ***Key-Point Detection***

A point is said to be a key-point which is considered as a center of the feature. A feature is application dependent. Various examples of the features are Corner, Edges, BLOB (Region of Interest). In the proposed algorithm, Corners are considered as a feature. Here, in this context a key-point denotes

a pixel of a corner. In the proposed algorithm ORB (Oriented FAST Rotated BRIEF) algorithm has been used for Key-point detection. ORB is free from licensing restrictions of SIFT and SURF. The ORB algorithm uses 'Oriented FAST (Features from Accelerated Segment Test)' algorithm to detect the scale invariant corner key-points and Harris Corner filter to reject edges and provides a reasonable score. FAST is several times faster than other existing corner detectors. It is dependent on a appropriate threshold. It identifies a pixel 'p' in the image as an interest point or not, based on the threshold chosen. The detected key-points will be further used in the process of matching and localization.

3.1.4. ***Key-point Descriptor***

The selection of algorithm for key-point description also plays a very important role. ORB uses BRIEF (Binary Robust Independent Elementary Features), A key-point descriptor is basically a binary string of adjustable length (32 bits, 64 bits, 128 bits, etc). BRIEF also provides high recognition rate unless there is large in-plane rotation. The length of the binary string is equal to the number of neighboring pixels that we are considering for corner detection. This means each bit of the binary string is related to a unique neighbor pixel of a key-point. A bit which is set (1) in the binary string denotes that the intensity of the key-point is greater than the intensity of the neighboring pixels that corresponds to the bit. The bit will be unset (0) for the opposite case. So, to make the corner detection rotation invariant, the mean gradient with the neighbor pixels is calculated for 30 orientations ($360/12 = 30$) of the key-point and the angle with the maximum gradient value is selected and the particular binary description is given to the key-point.

3.1.5. ***ORB (Oriented FAST Rotated BRIEF)***

ORB is basically a fusion of FAST key-point detector and BRIEF descriptor with many modifications to enhance the performance in various situations. It is rotation invariant and resistant to noise, which is two orders of magnitudes

faster than SIFT key-point detector and descriptor. The process of ORB starts by detecting FAST points in the image.

1. FAST selects one parameter, an intensity threshold value '**t**'.

2. It selects a pixel **P** in the image which is to be identified as an interest point or not. Let its intensity be I_p.

3. It considers a circle of 16 pixels around the pixel under test(P). Pixel P will be considered as a corner if and only if there exists a set of **n** consecutive pixels in the circle which are all brighter than $I_p + t$, or all darker than $I_p - t$.

FAST does not produce a measure of cornerness. So, it makes use of Harris Corner measure to order the FAST key-points detected. It orders them according to the Harris measure and picks the top desired number (say N) of points. FAST does not produce sacle invariant features. To make the detection sacle invariant, ORB employs a scale pyramid of the image and produces FAST features at each level of the pyramid. FAST also doesn't compute the orientation. To make detection rotation invariance, it computes the intensity weighted centroid of the patch with located corner at centroid. The orientation is given by the direction of the vector from this corner point to centroid. Now for descriptors, ORB uses binary descriptor based on BRIEF. BRIEF provides a shortcut to find the binary strings directly without finding descriptors. The process of BRIEF goes as follows,

1. It takes smoothened image patch and selects a set of n_d (x,y) location pairs in a unique way.

2. Based on some pixel intensity comparisons are done on these location pairs, we get a n_d-dimensional bitstring.

3. It is done as, for example, let first location pairs be P and Q. If I(P) ¡ I(Q), then its result is 1, else it is 0.

The value of n_d can be 128, 256 or 512 depending on the requirement of the application. The process of matching is carried out on these binary strings of the detected key-points.

3.1.6. ***Matching the key-points***

The whole heart of the designed algorithm lies in this phase. This customized part of matching is used for reducing the computational time of the overall algorithm because in majority of the proposed algorithms matching consumes most of the computation time. So, minimizing this part of the algorithm will boost the overall performance of the algorithm.

The "Trie" data structure has been implemented to perform the matching process of detected key-points. The used data structure is generally a tree kind of structure which has to be pre-computed before the process starts. The following explains the further process of pre-computation and matching :

3.1.7. ***Pre-Computation and Matching in Trie***

The tree constructed after the process of pre-computation will be a binary tree. Every single node of a tree consists an array of pointer data type of size 2 to hold the address of nodes in the next level of the tree and a Point data type which holds NULL for all the non-leaf nodes, whereas, the leaf nodes hold the co-ordinate of a unique key-point. Here each pointer in a node is used to represent a 0 (for the bit which is unset in binary string) and to represent 1 (for the bit which is set in binary string) of a key-point bit string.

The computational process includes considering a binary string of length 5 and then the tree constructed will be of height 5. Let us suppose the binary string be "10011". The process starts from index 0 (first bit from left) and ends at index 4 (right most bit). Let us consider 2 variables ptr and bit-val to explain the algorithm. Initially ptr holds the address of the root of the tree and bit-val contains first bit '1' and checks if the bit-val index of the array of ptr node has been allocated memory (i.e, already defined). If that was not the case then memory would be allocated to the bit-val index of the ptr node and ptr is updated to the address that bit-val index of the array of ptr node holds. This process is recursively computed until bit-val reaches the end of binary string. In the final node, whose address is held by ptr, we will store the co-ordinates of the key-point to which the path (binary string) belongs. Now this way the

tree will be constructed using the binary string of the key-points prior to the initialization of matching process.

Similar to the tree construction process the matching process for every key-point is initialized from the top (root) of the tree as well as from the first index of the binary string of the key-point. There will also be an extra added variable mis-hit which will be initialized to zero and will be incremented for every mishit during the propagation of the tree from top to bottom. Now, the bit-val index of the array of ptr node will be checked if it is initialized. If it is initialized, then it is a hit and with out the increment of the mis-hit we will propagate to the next level. If there is a mishit then we will increment the value of the mis-hit by one and propagate to the next level of tree through other index and check if the mis-hit value is under the threshold (for maximum number of allowed miss hits). If the mis-hit value crosses the threshold at any stage, then the particular recursion is terminated. Every time ptr reaches a leaf node, and if the mis-hit value is less than the previously found, the point will be updated with the point in the leaf node. This is how, the key-point with maximum hamming distance is found if there is any. Below is the structure of the each node in the tree (Trie node).

```
struct trie{
    struct trie* bit[2];
    Point pt;
};
```

3.1.8. ***Displaying the Matched Key-points***

After the process of Matching, we will get a uniquely matched point for every key-point. For all such pairs, a line is drawn between the points to show the connection between the copied regions of the image.

3.2. ***Datasets Description***

"MICC-F220" and "MICC-F2000" are the data-sets used for testing the proposed copy-move forgery detection. The data-sets contain 130 and 160 im-

ages respectively. The size of the images used are of 500 x 800 pixels. Both the data-sets contains the images that had undergone geometrical transformations like Scaling and Rotation. Data-sets consists of mixture of Forged images (Geometrically transformed) and Original images.

3.3. ***Performance Measures***

Several parameters are used for the purpose of performance measure where we have considered 4 different parameters namely True positive rate (Recall/Sensitivity) (TPR), False positive rate (FPR), Precision, Accuracy. They explain various properties of the implemented algorithm in the following way :

3.3.1. ***True Positive (TP)***

Number of forged key points among all matched key points by the algorithm.

3.3.2. ***False Positive (FP)***

Number of unforged key points among all matched key points by the algorithm.

3.3.3. ***True Negative (TN)***

Number of unforged key points among all unmatched key points by the algorithm.

3.3.4. ***False Negative (FN)***

Number of forged key points among all unmatched key points by the algorithm.

3.3.5. ***True Positive Rate***

Proportion of correctly matched regions with respect to total forged regions.

$$TPR = \frac{TP}{TP + FN}$$

3.3.6. ***False positive rate***

Proportion of false matches with respect to total unforged regions.

$$FPR = \frac{FP}{FP + TN}$$

3.3.7. ***Precision***

Proportion of correct matches among the total matches.

$$Precision = \frac{TP}{TP + FP}$$

3.3.8. ***Accuracy***

Proportion of correct detection in the detection of image.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

4. Results and Discussion

Database (MICC-F220)	Image Size	Time for Brute Force match	Time for 'Trie' based match
Micc-f220-1.jpg	600*800	2.6115	0.5212
Micc-f220-2.jpg	532*800	2.7937	0.8622
Micc-f220-3.jpg	532*800	2.5523	0.9074
Micc-f220-4.jpg	599*800	2.5818	1.1580
Micc-f220-5.jpg	532*800	2.6421	0.8645

Table 1: Computational time for images of Dataset Micc-F220

Database (MICC-F2000)	Image Size	Time for Brute Force match	Time for 'Trie' based match
Micc-f2000-1.jpg	1536*2048	2.7060	1.0361
Micc-f2000-2.jpg	1536*2048	2.7936	1.0612
Micc-f2000-3.jpg	1536*2048	2.6550	1.0321
Micc-f2000-4.jpg	1536*2048	2.7561	1.0780
Micc-f2000-5.jpg	1536*2048	2.7658	1.0524

Table 2: Computational time for images of Dataset Micc-F2000

5. Conclusions and Future Scope

From the above experimental results on Datasets Micc-F220,

Average computational time for Brute force match = 2.60029

Average computational time for Trie based match = 0.92763

From the above experimental results on Datasets Micc-F2000,

Average computational time for Brute force match = 2.71996

Average computational time for Trie based match = 1.04758

The ratio of computational time between Brute force based match and Trie based match is,

$$Ratio = \frac{Avgtime for Brute force}{Avgtime for Trie based}$$

$$Ratio for Micc - F220 = \frac{2.60029}{0.92763} = 2.803154275$$

$$Ratio for Micc - F2000 = \frac{2.71996}{1.04758} = 2.59642223$$

It can be observed from the experimental results, that the Trie based matching method is faster compared to Brute-froce based method. The above results were given when 1000 key-points were considered. In case of large images, it is obvious that even number of key-points are also increased. But, the computational time for Brute-force method increases in quadratic fashion, whereas, Trie based match increases in linear fashion.

As for making the algorithm feasible for a 64-bit linux based operating system, we considered only 64-bit length binary string as key-point descriptor. As we have used only 64 bit descriptor, there will be some false matches along with the true matches. So, by considering 64-bit bit descriptor we will work on improving the accuracy by minimizing the false matches and increasing the true matches. We also consider the fact that threshold also plays a vital role in providing good accuracy. So simultaneously we will be working on making the threshold adjustable with respect to the size of the image, number of detected key-points, etc.

References

[1] Fridrich, Soukal, Lukas, Detection of copy-move forgery in digital images.

[2] Huang, Guo, Zhang, Detection of copy-move forgery in digital images using sift algorithm, Pacific-Asia Workshop on Computational Intelligence and Industrial Application.

[3] Rublee, Rabaud, Konolige, Bradski, Orb: An efficient alternative to sift or surf, Computer Vision (ICCV), 2011 IEEE International Conference.

[4] Bodon, Ronyai, Trie: An alternative data structure for data mining algorithms, Computer and Automation Institute, Hungarian Academy of Sciences Lgymnyosi, Budapest, Hungary.