

Contents

1. Introduction.....	4
1.1 Introduction	4
1.2 Background and Motivation.....	5
1.3 Problem in brief.....	5
1.4 Aim and Objectives	6
1.4.1 Aim	6
1.4.2 Objectives	6
1.5 Solution	6
1.6 Summary	7
2. Literature Review.....	7
2.1 Introduction.....	7
2.2 Similar approaches.....	7
2.2.1 Image Cloning.....	7
2.2.2 Image Resampling	11
2.2.3 Image Splicing	13
2.2.4 Image Retouching with Filters.....	17
2.3 Summary	21
3. Technology Adapted.....	21
3.1 Introduction.....	21
3.2 Technologies Adapted	21
3.2.1 Programming Languages	21
3.2.2 Development Tools.....	22
3.2.3 OpenCV	22
3.2.4 Libsvm	22
3.2.5 Version Controlling Systems	22
3.3 Summary	22

4. Our Approach.....	23
4.1 Introduction.....	23
4.2 Image Cloning Detection Module.....	23
4.2.1 Approach 1	23
4.2.2 Approach 2.....	24
4.3 Image Resampling Detection Module	30
4.4 Image Splicing Detection Module	31
4.5 Image Retouching Detection Module	31
4.6 Summary	31
5. Analysis and Design	32
5.1 Introduction.....	32
5.2 High Level Architecture and Design	33
5.2.1 Image Cloning Detection Module.....	33
5.2.2 Image Resampling Detection Module	35
5.2.3 Image Splicing Detection Module	42
5.2.4 Image Retouching using filters Detection Module	45
5.4 Summary	51
6. Implementation	52
6.1 Introduction.....	52
6.2 Implementation	52
6.2.1 Image Cloning Detection Module.....	52
6.2.3 Image Splicing Detection Module	59
6.2.4 Image Retouching using Filters Detection Module	64
7. Evaluation and Results.....	66
7.1 Introduction.....	66
7.2 Evaluation and Results.....	66
7.2.1 Image Cloning Detection Module.....	66

7.2.2 Image Resampling Detection Module	68
7.2.3 Image Splicing Detection Module	74
7.2.4 Image Retouching with Filters Detection Module.....	81
8. Conclusion and Further Work.....	82
8.1 Introduction.....	82
8.2 Conclusion and Further Work.....	82
References.....	84
Individual Contribution to the Project	87
Code Segments Used in the Project	91
➤ Fourier Transforming the Periodicity Map.....	92
.....	92
Image Retouching using Filters Detection Module	102

Chapter 1

1. Introduction

1.1 Introduction

The field of digital images has become a very trending and indeed a popular field among people no matter how age group they are in. Nowadays almost all the people from school kid to 70's elder have smart phones and smart devices with them which enable them to capture images any moment. Therefore the use of digital images have shown a rapid development as well as a wide spread in society. With the growing usage of digital images, the reliability and the confidentiality of images have been questioned at a severe rate.

With the high rise of the usage of Digital Images, forgeries and tampering of mages have taken place. Many people digitally edit these images and use them illegally without owner authentication. Therefore Digital Image Forgeries have become a severe threat in modern society.

This system is developed with the main intention of protecting people from the damages that could be done with the tampering of digital images. Image forgery detection is to be carried out under different approaches for four forgery types. They are Image Cloning, Image Resampling, Image Splicing and Image retouching with filters. Above mentioned are the widely used tampering approaches that are currently taken place. Hence we intend to identify images that are digitally tampered under those categories. For that four separate modules are being developed with the use of novel algorithms. According to the feasibility by the time of the completion of the project, it is planned to carry out an evaluation based on different algorithms that are used in the development process.

The organization of the report is as follows.

Chapter 2 will state a literature review based on the project and a review of other's work. Chapter 3 is about the technologies that are adapted in the implementation phase. Approach of the group towards the problem and its solution is briefly described in the fourth chapter. Chapter 5 will give an overview about the analysis and design of the

system with the aid of necessary charts and diagrams. Chapter 6 contains the discussion of the project which gives a brief summary on the project.

1.2 Background and Motivation

In the present world many people have the luxury of having digital cameras with them even on their routine works. Many people use smart phones equipped with digital cameras which lead to an era of digital images. Unfortunately this digital technology has begun to erode the trust and the confidence in the integrity of imagery. From the tabloid magazines to the fashion industry and in mainstream media outlets, scientific journals, political campaigns, courtrooms, and the photo hoaxes that land in e-mail inboxes, doctored photographs are appearing with a growing frequency and sophistication. Therefore the field of Digital Forensics has emerged in uplifting and restoring the trust towards digital images.

1.3 Problem in brief

In the present society, editing and tampering of a digital image has become much easier with the rapid development of digital image processing technologies and the popularity of the digital cameras. Even an inexperienced person can easily adopt to these technologies and perform image forgeries with the aid of user friendly software. Cloning, Splicing, Resampling, Image Splicing and use of filters to smoothen and sharpen an image have become the most common forgery types that could manipulate images in a way that is very hardly to perceive by naked human eye. Hence it is very important to detect these above mentioned image forgery types.

In addition digital images play a vital role in many different areas such as criminal and forensic investigation, military, journalism and etc. Therefore in order to avoid this critical situation, Image Forgery Detection system will be developed considering various aspects of Image Forgeries.

1.4 Aim and Objectives

1.4.1 Aim

The aim of this project is to develop a system that detects Editing and tampering of digital images under four categories such that Cloning, Re-Sampling, Splicing and use of Smoothing and Sharpening Filters.

1.4.2 Objectives

The objectives of this project are,

- Review and understand the literature that is published under Digital Image Forgery Detection.
- Develop fully functional modules to detect image forgery occurred with Image Cloning, Re-Sampling, Splicing and using Smoothing and Sharpening Filters.
- Finding the relevant datasets.
- Test Accuracy with the different approaches that are taken under same Image Forgery Type.

1.5 Solution

Solution for the above identified problem is a Digital Image Forgery system which covers four main areas of Image Forgeries such as Image Cloning, Image Resampling, Image Splicing and Use of Filters. The modules are implemented with two different technology streams in an independent manner; Image Cloning and Use of Filters modules with OpenCV and python where the Image Splicing and Image Resampling modules are implemented with Matlab. With the assistance of researches done on related works, the most suitable novel solutions have been introduced for each module. The implementation process has gone through various experiments in order to find the most suitable solution avoiding pitfalls and gaps that exist in recent approaches.

Development of all four modules follow the following steps:

- Research on related work
- Experimental Implementation
- Finding relevant datasets
- Implementation
- Evaluation

1.6 Summary

The chapter briefs about the research project with its aim and objectives. In addition it contains the background details of the project and what motivates us to select this area as the final year research project.

Chapter 2

2. Literature Review

2.1 Introduction

This chapter discusses about the similarities and differences of similar approaches that are taken to solve the exact same problem. In this chapter we mainly consider about comparing and contrasting the other's work with the suggested approach highlighting its importance and uniqueness. Each section contains the review of other's work related to separate modules as follows.

- Image Cloning
- Image Resampling
- Image Splicing
- Image Retouching with Filters

2.2 Similar approaches

2.2.1 Image Cloning

Basically the image cloning (copy move) forgery detection is done in two main approaches,

1. block based
2. key point based

The main issue with block based is the heavy computational intensity [1]. But on the other side of the scale we have the accuracy. Due to heavy searching through out the image the probability of getting the most correct place is high in block based approaches.

But the second approach of key point based copy move forgery detection is not that heavy computational as we only focus on the special blobs of the image. And bead on

the feature descriptor we can build a robust cloning detector. For instance if we use feature descriptors like Scale Invariant Transform (SIFT), Speeded Up robust features (SURF) and DAISY then the areas of detection will be robust against geometrical transformations.

In a typical key point based cloning forgery detection following three main steps are abundant.

1. Feature Extraction
2. Feature Matching
3. Post-processing

Here post processing is mostly done to remove the false-true matches of the detected areas through feature matching.

2.2.1.1 Segmentation Based Image Copy-Move Forgery Detection Scheme

[2] L. Jian et al. have developed image forgery detection scheme based on the following flow chart.

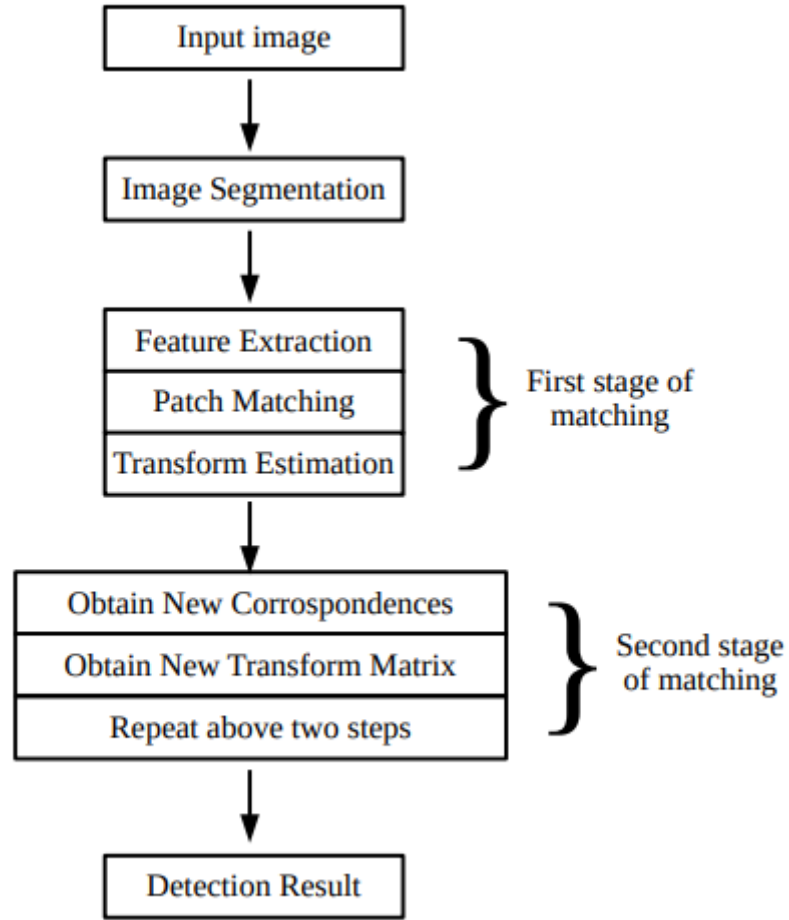


Figure 2.1: Flow Chart for the Image Forgery Detection Schema

Image segmentation → SLIC algorithm is used for image segmentation

Feature extraction → SIFT key points are extracted

Patch Matching → k nearest neighbors are searched for the features and measured similarity with a threshold of 0.04. And a patch is said to be matched with another only if both the patches have key points above some threshold.

Transform estimation → transformation matrix is calculated between two matched key points using RANSAC estimation method.

2.2.1.2 Image Copy-Move Forgery Detection using Hierarchical Feature Point Matching

[3] Yuanman et al. In their approach too follows the typical three steps of forgery detection algorithm as mentioned above.

Feature extraction → high number of features are extracted by lowering the contrast threshold of SIFT features and by scaling up the image by a factor of 2.

Feature Matching → two problems arrived in matching features as the number of extracted features were increased.

1. The matching was reduced to 1 to 5 due to having the identified key points were slightly different and though they were at the same place they are found at two different scales.
2. The computational intensity of matching key points increased by $O(n^2)$.

These two problems were solved using the following strategy.

1. Group the key points based on some overlapping quantized gray levels before matching the key points.

Post processing → local affine homography estimation was developed using RANSAC algorithm for each matched key points. This obtained homography estimation \emptyset_H was compared with the dominant orientation of SIFT key points \emptyset_k . This give the equation,

$$f(k, k', H) = |\emptyset_{k'} - \emptyset_k - \emptyset_H| \quad (2.1)$$

And it is declared as a forged image if there are 5 pairs of matched key points satisfy this function.

2.2.1.3 Adaptive Polar based Filtering method for Image Copy-Move Forgery Detection

In [4] Xiuli et al., T=the main focus of this paper in detecting forgery is to do the operation with minimum computational intensity and identification of small forged areas very accurately.

The main steps of the technique in detection are,

SIFT key point extraction

Feature matching (knn)

Pixel classification into groups Symmetrical Matched Pixels (SMP) and Unsymmetrical Matched Pixels (UMP) based on the fact that whether the feature pixel (P_i, P_j) fall into matched pixel pairs in both ways of (P_i, P_j) and (P_j, P_i)

Polar distribution is calculated using a navel equation for both the SMP and UMP separately.

Adaptive threshold is calculated separately for the SMP and UMP sets separately based on statistical manipulations.

Finally the forged area is confirmed and exact forged pixels are detected by using morphological operations.

2.2.2 Image Resampling

Image resampling is essential to give a consistent and natural appearance when a new image is created using splicing of other images. So every forged image have undergone geometric transformations such as scaling, rotation etc. which results in resampling of the images. Geometric transformations typically require a resampling. Due to this reason image resampling detection is an essential feature of image forgery detection. So many studies have been conducted on how to detect image resampling of images.

All the methods designed so far has a common underlying basis of detecting signal specific detectable statistical changes that occur due to the interpolation techniques. According to those studies resampling imposes periodic correlations between pixels do not exist in not resampled images. These correlations can estimated by a learning algorithm. In order to determine if the image has been resampled, the found correlations has to be passed to a classifier. Different studies have used different classifiers to detect the resampling of images [5].

2.2.2.1 Construction of a database of synthetic maps for different resampling ratios

The approach mentioned in [6] was developed with the idea that interpolated samples show a linear combination with their neighboring pixels. The initial step of this approach is identifying the correlated neighboring pixels. The Estimation Maximization Algorithm was used to find the correlated pixels using resampling coefficients to estimate the probability map which is the output from the EM Algorithm.

Then Fourier Transformation was applied to enhance the peaks available in the probability map to make it easier to detect periodic correlations available. The main

purpose of generating a synthetic map was to quantify the sensitivity and robustness of the EM Algorithm. Then the generated synthetic map was compared against the probability map and their difference is calculated. If this calculated difference exceeds the predefined threshold value that image was classified as a resampled image and if the difference is lesser than the threshold value that image was classified as a genuine image with no resampling artifacts.

Then a synthetic probability map which contains periodic patterns that are available in resampled images was created.

The main drawbacks in this approach is that it is only able to detect forgeries in uncompressed TIFF images, and JPEG and GIF images with minimal compression. So it can't detect forgeries in recompressed JPEG images which occurs more frequently.

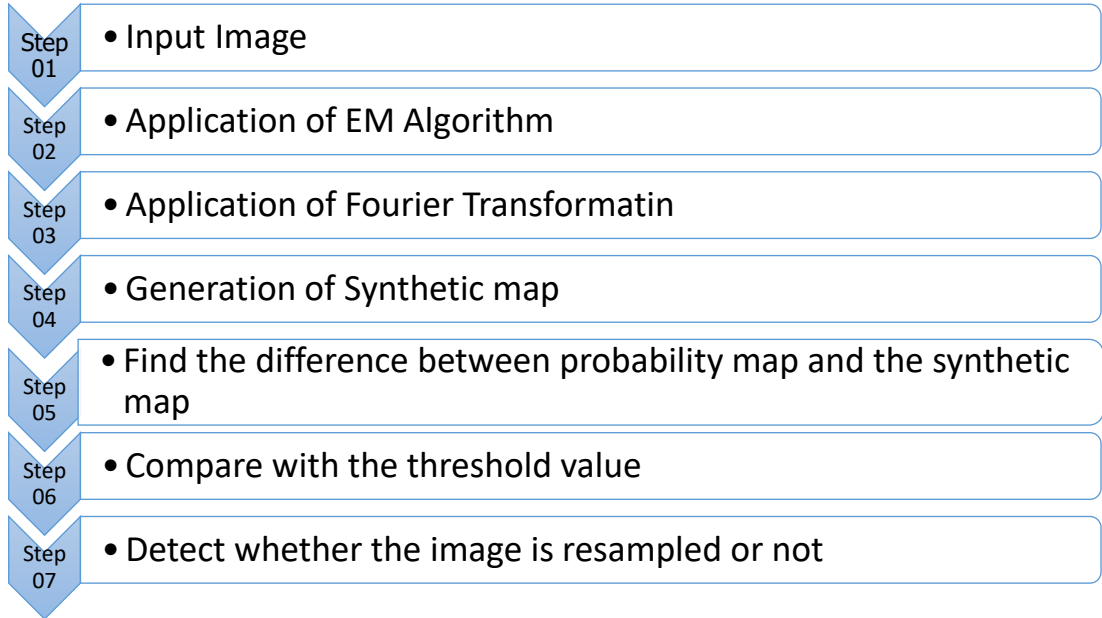


Figure 2.2: The Process Flow

2.2.2.2 Resampling Detection with Linear Predictor Residue

In this approach discussed in [5] also exploits the method in [6] and proposes a faster resampling detection method than [6]. The basis of this approach is weighted least squares (WLS) estimation of the initially unknown scalar weight incorporated into an iterative expectation maximization (EM) framework [5]. To calculate the scalar weights required for the generation of the p-map they have used a fast linear filtering method with preset coefficients. Here also initially they have created a probability map which is a measure of the strength of linear dependency derived from the prediction error e ,

and it is modelled as a zero mean Gaussian random variable. The probability map was calculated using the following equation where $\lambda, \sigma > 0$ and $r \geq 1$, which is faster than the methodology used in [6].

$$p = \lambda \exp\left(-\frac{|e|^r}{\sigma}\right)$$

Then Discrete Fourier Transformation is done to make the distinct peaks which are the characteristics that are used to detect the resampling visible. In this phase they have used an approach to automatically detect the peaks in a p-map's spectrum.

Then they have used a cumulative period gram, which is used for time series analysis to detect the presence of particular frequency components. This equation was changed to calculate a 2D version from the first quadrant of a p-map's DFT.

Then a new decision criteria was calculated using the maximum absolute gradient of the cumulative period gram. This value was compared with a Threshold value and if it exceeds that value it was classified as a resampled image.

But there are some ambiguities in identifying the characterized peaks generated by the probability map and it's unable to identify the geometric distortion attacks against resampling detection.

2.2.3 Image Splicing

Image splicing is commonly known as developing a composite picture by adding a separate objects or regions from some other image. Splicing is one of the forgery types that are commonly used in modern society. Therefore many researches have been initiated on different approaches in order to detect Image Splicing. Following are the Identified approaches of Splicing Detection.

- Splicing Detection using Inconsistencies in Motion Blur [7].
- Splicing Detection using Inconsistencies of Shadow Parameter [8].
- Splicing Detection using Discontinuities of Illumination Colour [9].

2.2.3.1 Splicing Detection using Inconsistencies in Motion Blur

This approach has been taken to detect Image Splicing using the discrepancies of motion blur estimations of the particular image [7]. Motion Blur refers to the relevant colour lines of a moving object that appears in a still image. Rather than that Motion Blur can occur due to the camera shakes.

In this approach, Motion Blur estimation has been used through image gradients in order to detect inconsistencies between the spliced region and the rest of the image. Complete absence of Motion blur is very helpful factor to detect Image Splicing. If the image does not have motion blur estimates though it was required, such incidents are said to be complete absence of Motion Blur, which lead to possible suspects of Image Forgery. Though the motion Blur was applied to a certain object in an image, there will be mismatches and inconsistencies. When compared to the original image.

When an image is given, its blur estimates are being calculated first. Then the image is being segmented using the previously calculated estimates. Different algorithms are used to calculate the Motion Blur Estimates. As it is mentioned in [7], a variant of the widely recognized cepstral method is used to estimate motion blur. Furthermore the study mentions that they have used spectral characteristics of the image gradients.



Figure 2.3: Process flow of Splicing Detection through Motion Blur

Though the approach is novel and high in accuracy, there exists some restrictions where the approach become unable in functioning. Since the detection is done through Motion Blur Estimates, this approach cannot be applied to images that do not contain motion blurs. Splicing in totally still images cannot be detected using this approach.

2.2.3.2 Splicing Detection using Inconsistencies of Shadow Parameter

This is another novel method of detecting Image Splicing. Forgeries are detected by applying photometric constraints on shadows. Inconsistencies of Shadow parameter that occur due to the spliced object with the rest of the image are used in this approach.

As it is mentioned in [8], first they formulate the colour characteristics of shadows measured by shadow matte value. Shadow matte value (C) is a scale factor which is evaluated by analyzing illumination intensity in shadows [8]. The discrepancies of matte values of different shadows implies a possibility of a forgery.

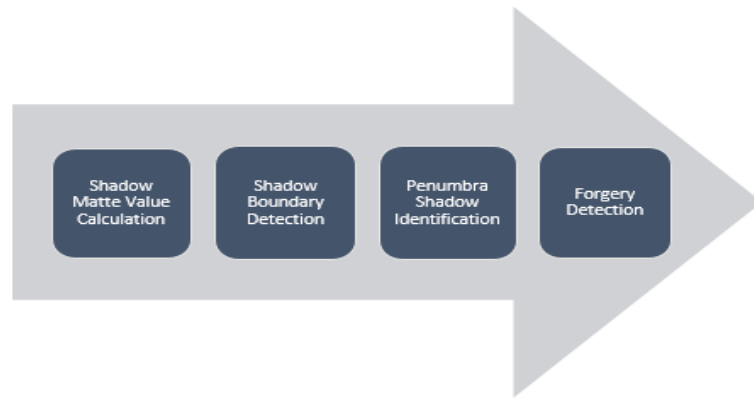


Figure 2.4: Process flow of Splicing Detection through Shadow Parameter

Even though this approach is used in Image Splicing detection, there are many restrictions and constraints that are needed to be concern when it comes to practical use.

Mainly this approach consider only about the photos that are taken outside considering that the only lighting source would be the sun. This method will not work properly in most of the cases where there are multiple lighting sources which cause multiple shadows for the same object. N such cases it would be very difficult to carry out the detection of Spliced Region. Next the researchers of [8] have made the assumption that the shadow receiving surface is always Lambertian. A Lambertian surface is a uniformly bright area from all directions of view where it reflects the same intense of light. In addition there is another restriction on the opacity of the object. The shadow generating object should be opaque and the distance between above object and the shadow receiving object should be non-trivial.

2.2.3.3 Splicing Detection using Discontinuities of Illumination Colour

This is another novel method which uses a machine learning approach to detect Image Splicing with respect to the Illumination estimators of colours. In this approach, the differences of the illumination estimators of colours regarding similar objects are used in order to recognize splicing objects.

According to the [9] the initial step is to segment the image into homogeneous regions. Information from physics and statistical based illuminant estimators on image regions or similar material will be incorporated [9]. An illuminant map will be created by applying the extracted colour to the each region. Once a human expert sets bounding boxes around each faces, those will be cropped and their texture based and gradient based Illumination Map values will be computed. According to the number of faces in the image, Joint Vectors will be constructed. As the final step which is being done for the classification purpose, a machine learning based approach has been used in this method. If at least one pair of faces in the image is classified as inconsistently illuminated, then the image is categorized as a spliced image [9].

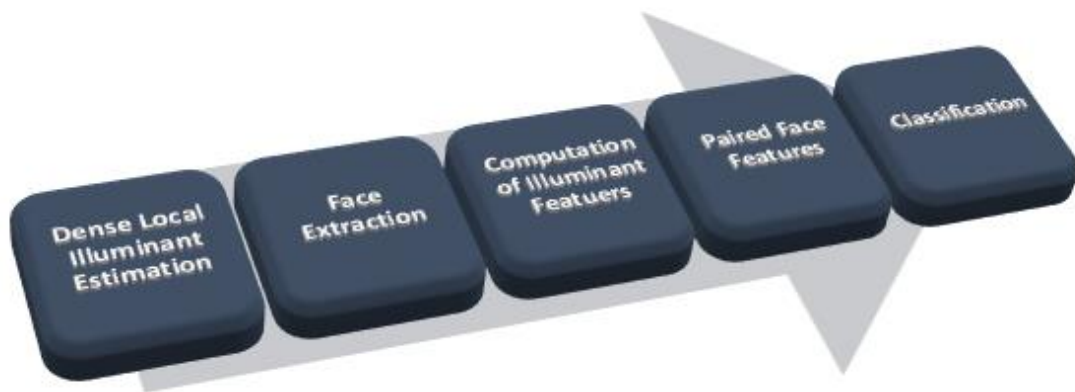


Figure 2.5: Process flow of Splicing Detection through Illumination colour

The main disadvantage of this method is that the approach is only restricted to images which have more than 2 or more faces. The approach mainly uses an automated skin comparison to check whether the image is spliced. Hence the method cannot be universally applied. In addition this approach requires human expertise to some extent

which can be often misleading. So relying on human visual assessment helps to drop the accuracy of the method.

2.2.4 Image Retouching with Filters

Filtering is a technique which is utilized to enhance a picture. For a case you can utilize a particular channel to accentuate certain highlights. Some of the time it is utilized to expel a few highlights. In the picture preparing there are numerous channels utilized for improving the nature of the picture. Additionally when it is thinking about the picture falsification recognition, it ought to be viewed as the way that particular picture had connected particular channel for improving the picture or some time utilizing a channel it can be conceal the imitation territory. Ex: When it is thinking about a duplicate move falsification there are substitution of particular territory of the first picture. In the wake of duplicating the picture it ought to be smoothed the edges of recently included square of the picture. Those errands are done with utilization of the picture Filters.

Picture channel is an area task, which the estimation of any pixel in the yield picture is controlled by applying some calculation to the estimations of the neighboring pixels of the comparing input picture. In picture preparing there are chiefly two kinds of channels.

- Linear Filtering - Gaussian, Laplacian, Average and unsharp
- Non-Linear Filtering - Median

In this project it is expected to implement the detection the forgeries using above mentioned linear and non-linear filters

As indicated by the circumstance it is chosen the sort of filter. More often than not filters are utilized as a post handling activity in picture improvement. In a few times filtering is connected to shroud alternate tasks done in the first picture. Utilizing a filter it can be shroud the other sort of forgeries happened to the first picture like copy move, re-sampling. In here it ought to be viewed as what sort of filter ought to be connected and what are the strategy ought to take after for identifying the kind of channel which is connected to the first picture. There are number of ways to deal with distinguish the filter compose. In any case, it ought to be considered what the kind of identification strategy ought to be connected to identify the comparing filter. Some of the time it

might be a linear filtering detection strategy. At times it will be a nonlinear filtering discovery strategy. There are a few ways to deal with identify the non-linear filtering. In any case, when it is thinking about the linear filters, there are few number of identifying techniques.

2.2.4.1 Methods for detect the median filter. (Non-linear filter)

2.2.4.1.1 Enhanced Statistical Approach for median filter detection using difference image

Unlike other approaches it is difficult to detect the non-linear filtering because of the non-linear characteristics of median-filtering. According to the [10] it is suggested a method for detecting the median filter using Local Expectation Features (LEF). In this approach utilize adjacent pixels of different image corresponding to input as well as median filtering (MF) version of input image for extracting proposed LEFs.

In this approach, it is considered the fact that no common analytical property between inputs and outputs because of the non-linearity nature of the median filter. Even though after applying the median filter, leaves some statistical characteristics. It is used these statistical characteristics to detect the median filter. In[11] authors used the probability of observing zeros in first difference image, as the identifying factor for separating median filter from other filtering operations. But the issue with this method is that give good accuracy only for UN-compressed median filtered images, the performance is deteriorated when using the JPEG compressed images. Here use a feature vector for detect the MF. It is consider the ratio h_0/h_1 . Here h_0 is the number of zeros and H_1 is the number of 1s in the first difference image. Here it is degraded the performance when using JPEG compressed images. There author [12] use a Subtractive Pixel Adjacency Matrix features (SPAM) as a distinguishing measure for the MF images. Authors in [13] considered the characteristics of first and second order difference domain images to Obtain Global Characteristics Features (GCF) and Local Characteristics Feature

(LCF), then joined these two vectors for get Global-Local Features (GLF) as a 56-D vector. This GLF has superior detection accuracy than others.

LEF – Local Extraction Features

This feature set is constructed from the expectation of probability maps obtained from image intensity difference values.

In blind detection of image forgery, the feature that output is a subset of input samples is used to detection of median filter. To overcome the issue of false alarm rates, only textured regions of an image are used for feature extraction. Binary mask is used to detect textured regions. For each pixel in the test image I with intensity value $I(i, j)$ variance in local neighborhood of a pixel is calculated according to below equation.

$$\sigma(i, j) = Var \{I(m, n)\}$$

$$m \in \{ \quad \quad \quad \} \quad (2.12)$$

$$n \in \{l - 3, l + 3\}$$

The binary mask $M(i, j)$ is then produced from $\sigma(i, j)$, based on variance threshold (σ_τ).

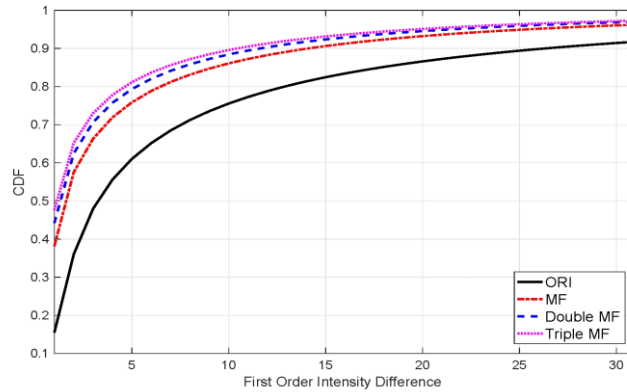


Figure 2.6: First Order Intensity Difference

Once the original image is median filtered, corresponding difference image will have more number of zeros, but increase in zeros or upward shift in cumulative histograms of consecutive MF images is not linear. This quality is used to detect the median filtered image.

In addition to this method there are many methods for median filtering detection. Convolution Neural Network, Local Ternary Pattern approach which fits to linear autoregressive model.

2.2.4.1.2 Linear Filter Detection

Detection of Gaussian Filter.

In the section of Gaussian Filter detection it is required to present three kinds of feature vector which are extracted from the edge ratios and parameters of Hough peaks .In the proposed algorithm[14], the formed a 10-dimensional feature vector and then trained using a support vector machine(SVM).

In here features are classified into two categories as spectral features and spatial features. J. Xu et al. [14] developed a method for detection of Gaussian low-pass filtering in the spectral domain. Authors used the frequency residual function that is used for detection of Gaussian low-pass filter in spectral domain. In here use the observation that frequency residual function of Gaussian filtered image presents a band-pass filter and original image presents a high-pass shape.

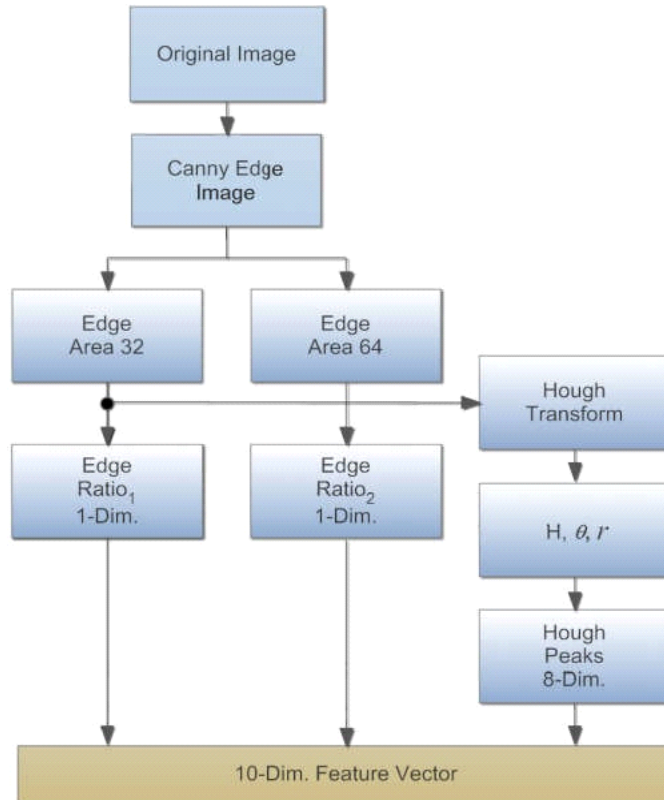


Figure 2.7: Gaussian Filtering Detection Diagram

2.3 Summary

Chapter two contained facts about similar approaches carried out by different people worldwide to solve the same kind of problem. It summarizes the other recent approaches that are taken to develop Digital Image Forgery Detection systems with respect to Image Cloning, Image Resampling, Image Splicing and Image Retouching with Filters. Chapter 3 will focus on the technologies used by the team to develop the system.

Chapter 3

3. Technology Adapted

3.1 Introduction

In this chapter the basic technologies we are going to use for the development process of different modules of our project are briefly discussed. We selected the technologies discussed below after analyzing the available technologies in terms of their appropriateness for our project. Accordingly we found that following are the most suitable technologies for our system. It also describes why we selected those technologies instead of other available technologies.

3.2 Technologies Adapted

3.2.1 Programming Languages

The programming language we are going to use for the development of our system is Python and Matlab. The main reasons for selecting Python is because it is a high level programming language which supports both object oriented programming, functional programming and there's a high level of library support. Scikit Library was used along with python for Image Cloning Detection Implementation.

Matlab is multi-paradigm numerical environment which is a programming language which allows mathematical functions such as matrix manipulations, function plotting etc. [15]. The main reasons why Matlab was selected is the availability of many libraries, ability to interface with programs written in other programming languages such as C++, Python etc.

3.2.2 Development Tools

PyCharm and Matlab IDE will be used as the primary IDEs for our project. Visual Studio will also be used upon the necessity for developing the system.

3.2.3 OpenCV

OpenCV which is a library of programming functions which supports computer vision is the main library we are going to use for this project. The main reason for selecting it is the availability of a large no. of library functions and simplicity in the use.

3.2.4 Libsvm

LIBSVM is a library for Support Vector Machines (SVMs). Here we use the libsvm classification for image forgery detection of filters. The goal of using libsvm is easily do the classification on extracted image features. LIBSVM has gained wide popularity in machine learning and many other areas. Here we use for image classification.

3.2.5 Version Controlling Systems

Git is used as the version controlling system and the code repository in the development of the system.

3.3 Summary

The stack of technologies we are going to use mainly in our project is described in this chapter. In addition to these we are going to utilize some more tools and libraries in the development process of our system.

Chapter 4

4. Our Approach

4.1 Introduction

This chapter summarizes the approach followed by us to solve the identified problem of image forgery detection. The approach is explained in abstract level in terms of input, output and process of each modules.

4.2 Image Cloning Detection Module

This is implemented as another possible approach to detect copy move and the forged area. This is done based on the assumption that the forged area is to give some semantic meaning to the image (this means that the image is forged so that it is taking more than 10x10 pixel area).

Basically the implementation is done in two main approaches as the initial approach that is based on Euclidean distance is making the computational intensity more in $O(n^2)$. This makes the calculation really slow when comes to matching key points of over 2000. So a second approach is done using the same theories but a different flow with the introduction of KNN (K nearest neighbor) algorithm that is reducing the matching complexity to a great extent.

4.2.1 Approach 1

This approach is the most economical and is giving fairly good results for the computational effort it is required. Here we initially segment the image and then extract

the SIFT key points. These extracted key points are then matched using KNN clustering (KNN was used instead of euclidean distance calculation to reduce the computational power requirement), where they were matched based on a threshold value. Now these matched key points are then localized based on their location and then based on the threshold of matched key points per cluster the patches are said to be forged or not.

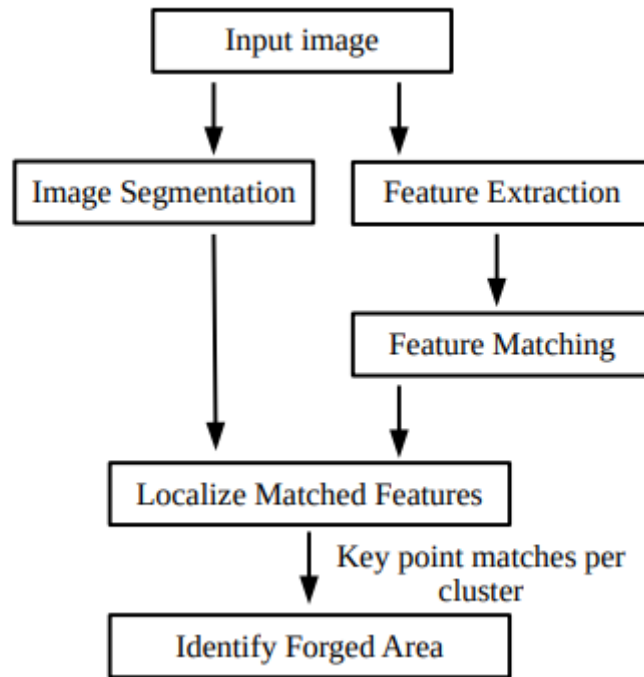


Figure 4.1: Flow Chart of Approach 1

4.2.2 Approach 2

Initially image is segmented using SLIC algorithm and SIFT features are identified. These identified SIFT key points are grouped according to the pre-identified segments and matching process is done between these pixels using KNN clustering. If key points match with less error than the threshold and matching pairs of key points in a particular patch is greater than the threshold of matching pairs per patch, then those two patches are detected as forged area. In the patches where there are no any key points detected then using a 8*8 Histogram of Oriented Gradients (HOG) kernel is used to extract the features. These patches with HOG features are matched with the other patches that have HOG descriptors instead of SIFT features.

The main issue arrived in this case was the matching between the features of SIFT that has high dimensionality. And the HOG features matching was not accurate as they matched the gradients with same appearance but not forged really.

The basic flow of the approach is as below.

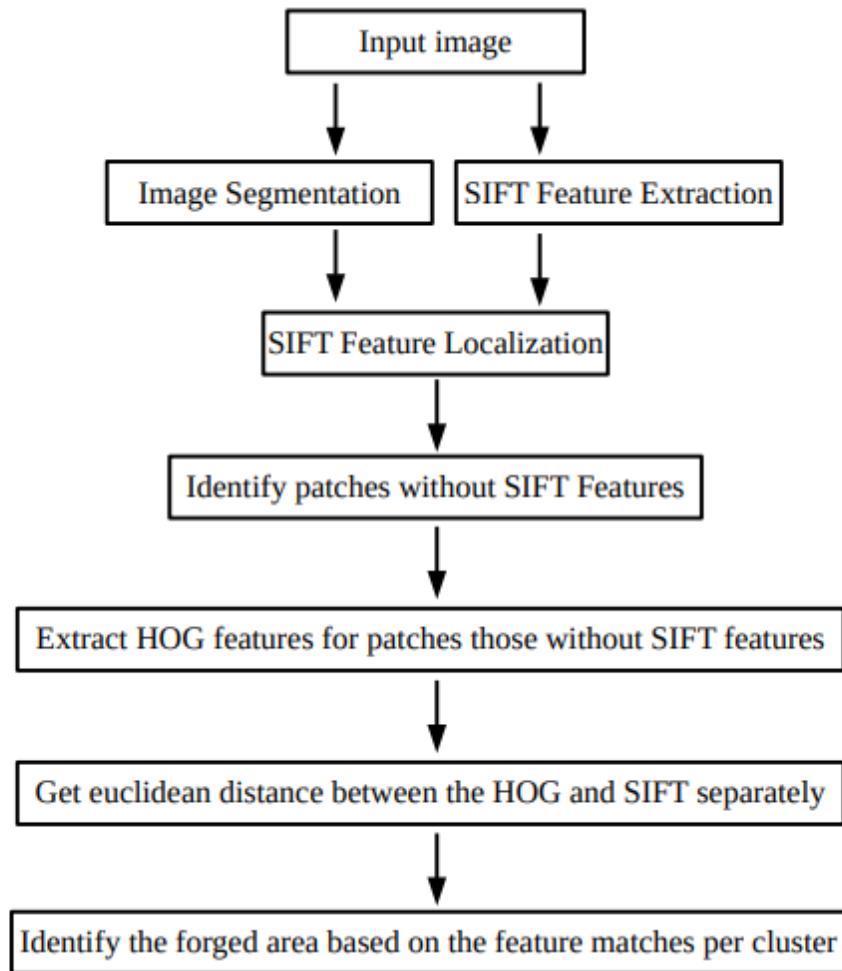


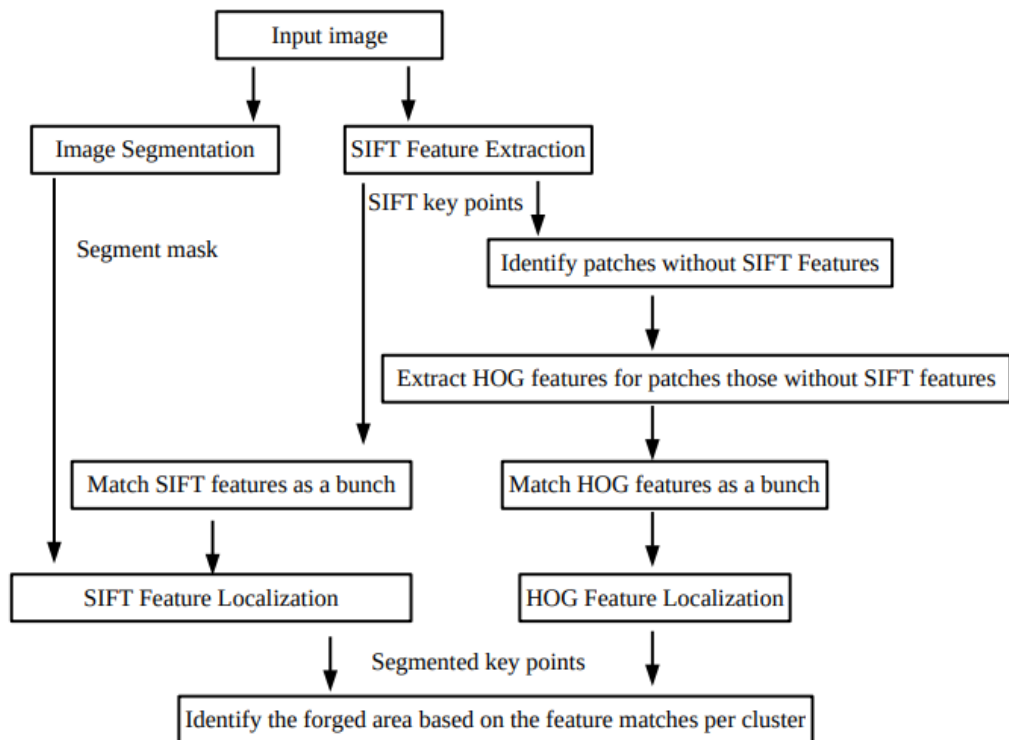
Figure 4.2: Flow Chart of Approach 2

4.2.3 Approach 3

Here this method was inspired by the heavy computational burden that was with the previous approach, which made almost impossible to do the calculation with a normal machine (i3 processor, 4GB ram).

In this approach initially we extract the features as in the previous approach and are grouped according to the SLIC mask produced. The descriptors in the group as a bunch

are taken to the KNN matching process that is relatively faster than the euclidean distance measure. And based on the matches we can decide for the forged area. Similar to the above method here we look for the HOG descriptor matching is there was no any key point match under our threshold.



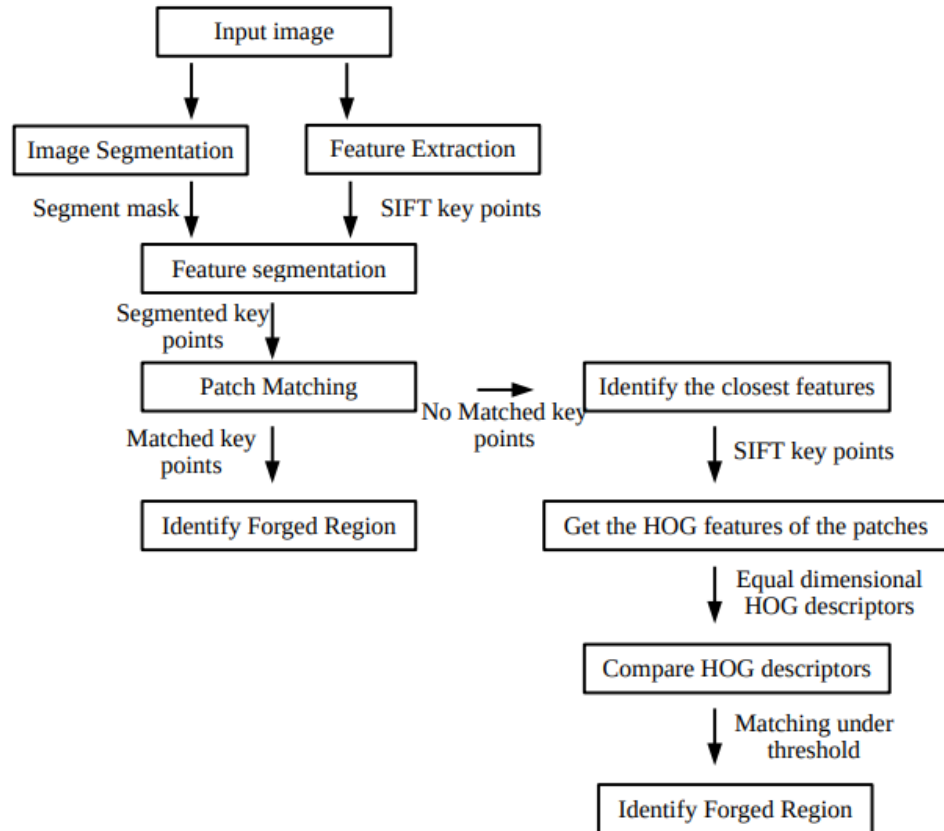


Figure 4.3: Flow Charts of Approach 3

3.2.4 Approach 4

This approach is mainly focusing on the possible miscalculations in detecting the proper cloned patch. That is there can be a possibility of falling a key point to a different segment though that particular patch contain most of the cloned area.

Imagine that the butterfly was cloned in the picture, and let the picture was segmented and key points were selected as below.

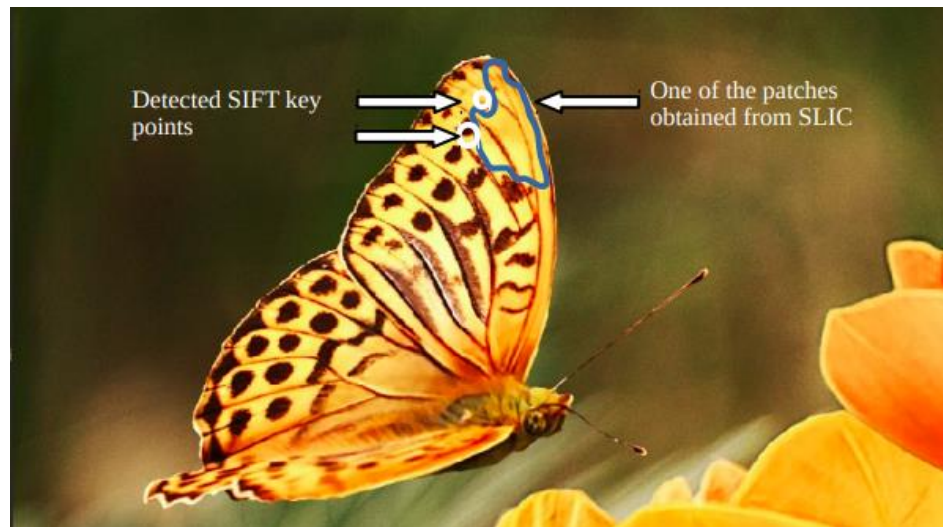


Figure 4.4: SIFT key points detected in an image

This is due to the reason that SLIC introduced by R. Achanta et al. in [10] is a super pixel based KNN clustering algorithm that clusters similar nearby pixels together and on the other hand SIFT is detecting the blobs of image, the proper cloned patch may not have enough SIFT key points to predict that it is a cloned area.

To overcome this issue the approach 4 is used where this segment is considered as the minimum area square contour as shown below.

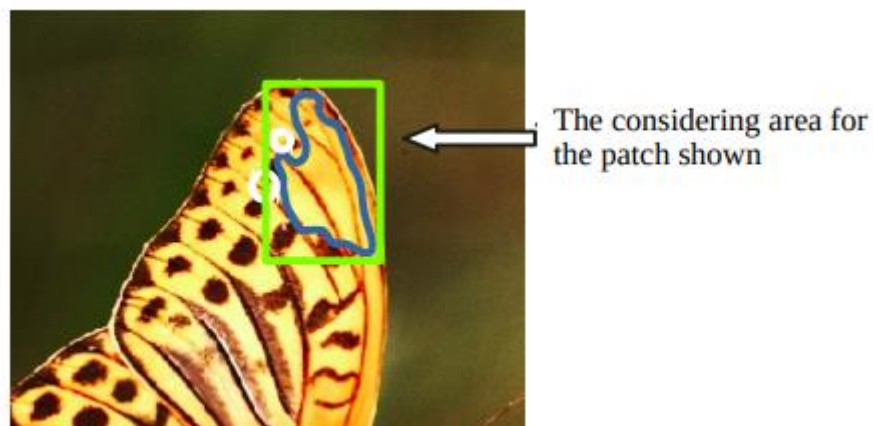


Figure 4.5: SIFT key points detected in an image

Now this will allow us to include the missing SIFT key points in the patch that are required to be included in the patch that is cloned.

Now in feature extraction this detected contour is removed from the image and SIFT keys are separately obtained from the contour and the contour removed image.

The resulting two segments will be as shown below.



Figure 4.6: Image with removed contour (a) and ROI Image (b)

Now for the whole image segments separately this contour removed image and the image of Region of Interest (ROI) is obtained. And their SIFT key points are obtained separately and matched. Therefore this method multiply key point matching burden by a factor of number of segment times.

Now if required number of key point matches are obtained for a particular ROI with the contour removed image then the matched SIFT keys in the contour removed image are localized and the patch is obtained. Now this identified patch and the patch used to take the ROI are considered to be cloned.

Now if we put the above steps into a flow chart it will be as follows.

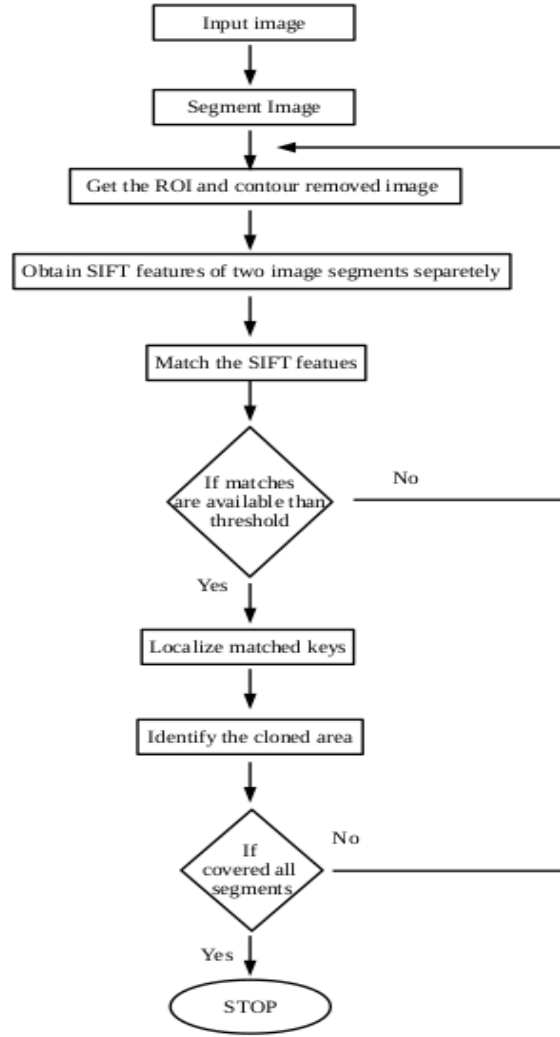


Figure 4.7: Flow Chart of Approach 4

4.3 Image Resampling Detection Module

The input of this module is an image. Then the correlations between pixels are identified using EM algorithm. The correlations exist in an untampered image and correlations introduced during resampling has to be distinctly identified. Then the probability map has to be created and then it is independently Fast Fourier Transformed and Discrete Fourier Transformed to convert the image data to frequency domain. In the next step these Fourier Transformed maps are high pass filtered to suppress low frequencies and the output is used to extract features to use as input to the SVM. The classification is done by the SVM and the localizing of the resampled region is done using the heat map generated using the probability map. The output of this module is the detection result

that whether an image is forged by using image resampling or it's a genuine image with no forgeries due to image resampling.

4.4 Image Splicing Detection Module

The input of this module is a tampered image. Then Image Splicing detection is done using Noise pattern Analysis and CFA (color filter array) interpolation in this module. Here we are going to use algorithms such as Wavelet Transformation, Noise Level Estimation etc. and some customized algorithms. This module produces a detection result which indicates whether the input image is forged by using image splicing or it's a genuine image with no forgeries due to image splicing.

4.5 Image Retouching Detection Module

This module also accepts an image as the input. In this module image forgeries due to sharpening and smoothing caused by both Linear Filters such as Gaussian, Unsharp, Average, Mean filters etc. and Non-linear Filters such as Median Filter are detected. The output of this module is the detection result that determines whether an image is forged due to sharpening or unsharpening or it's a genuine image with no forgeries due to image retouching with filters.

In here we build quantization noise model and image model which is specially build to detect filtering forgeries to extract compression noise then this extracted compression noise is modeled as first order spatial ergodic Markov chain which has been effective feature to do the classification. Then Libsvm/matlab multiclass classification is used to do the classification. Here we used NCID and UCID and Dresden Image Data Set (Raw Images).

4.6 Summary

This chapter describes the abstract level architecture of our proposed system in terms of their input, process and output. This summarizes how each module functions and assists in achieving its objectives to achieve the final goal of the system.

Chapter 5

5. Analysis and Design

5.1 Introduction

This chapter describes about the structure of the design of our approach which is followed to address the problem identified. It consists the high level architecture of each component with block diagrams. The flow of each algorithm is briefly mentioned under separate topics.

5.2 High Level Architecture and Design

The approach consist of four separate modules developed to detect Image Splicing, Image Resampling, Image Splicing and Image Retouching with Filters. The modules are not connected to each other and they are to function individually.

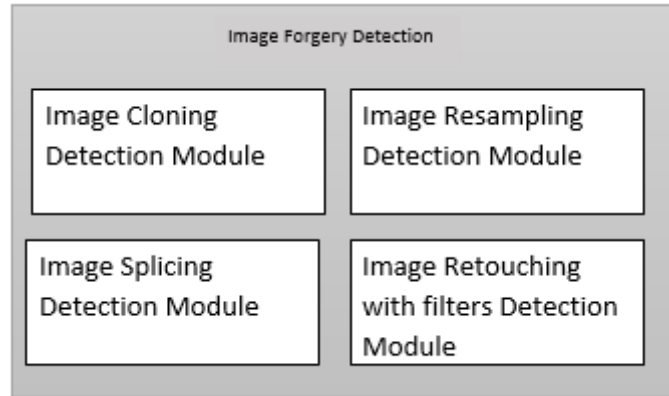


Fig 5.1: The Basic Structure of the Design

5.2.1 Image Cloning Detection Module

The basic structure of the implementation is as follows,

Image segmentation → Image need to be segmented to individual non interlacing patches in order to identify the forged region. That is if a forge is proved to be in a particular image then it will be one of these patches. SLIC algorithm is adopted to make this segmentation, that use a super pixel based image segmentation approach and k-nearest neighbor algorithm to expand the area. This will generate a segmentation mask of the image. Hence this algorithm will cluster the similar pixels together within a closed area.

The segmentation is done at an upper bound of 500 units and a particular segmented image would appear as shown below. For this “skimage”, image processing library is used.



Fig 5.2: An Image Processed with “skimage” library

Feature Extraction → At the same time we can do the image feature extraction, where we use SIFT key points as the features. For the features description SIFT was used as the key points are proved to be robust against geometrical changes. So that the interested areas of forgery that might have been undergone geometrical changes will also be detected.

Feature Segmentation → Above identified SIFT key points are grouped based on the mask obtained from SLIC algorithm. With this classification we can reduce the searching overhead to a great level.

Patch Matching → the classified key points are matched between the different patches that were obtained using a threshold of 50. then if a particular patch is comprising of a match of more than 2 key point values than both those patches are considered to be forged. This patch matching is done until the end of each iteration without stopping at a particular match, which enable to identify multiple copy pasted areas.

A sample result of the clone detected image is shown as below through patch matching. Here the cloning was done using the clone tool of the GIMP editing software.



Fig 5.2: An Original Image (a) and a Clone detected image (b)

5.2.2 Image Resampling Detection Module

The abstract design of the Image Resampling Detection Module can be shown as follows.

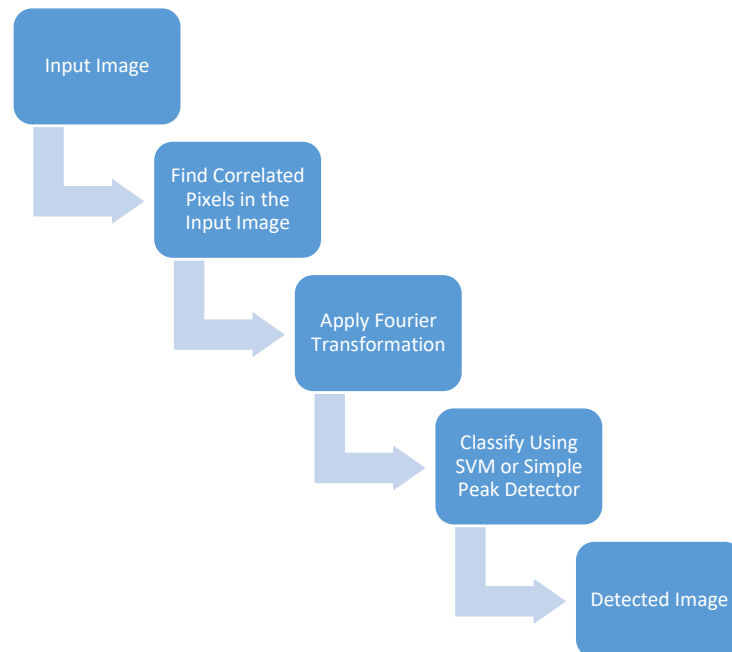


Figure 5.3 : The Abstract design of the Image Resampling Detection Module

5.2.2.1 Find Correlated Pixels in Image

Resampling introduces periodic correlations between neighboring pixels which are not available in an original image. So the initial step of the Resampling Detection Module is to detect whether any linear correlations are available in the input image. To identify these correlated pixels in the input image following steps have to be followed.

The input image is a discrete signal and any discrete signal can be resized by a factor of p/q to n samples using the following three steps. They are,

1. Up-sample : create a new signal $x_u[t]$ with pm samples, where $x_u[pt] = x[t]$, $t = 1, 2, \dots, m$, and $x_u[t] = 0$ otherwise, Fig. 1(b).
2. Interpolate : convolve $x_u[t]$ with a low-pass filter: $x_i[t] = x_u[t] * h[t]$, Fig. 1(c).
3. Down-sample: create a new signal $x_d[t]$ with n samples, where $x_d[t] = x_i[qt]$, $t = 1, 2, \dots, n$. Denote the re-sampled signal as $y[t] = x_d[t]$, Fig. 1(d).

For interpolation process different types of algorithms such as Nearest Neighbor, Bilinear, Bicubic, Bicubic Smoother, Bicubic Sharper can be used.

3

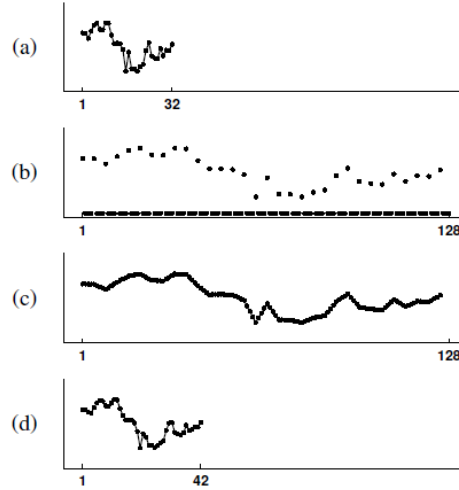


Figure.5.4. Re-sampling a signal by a factor of $2/3$; (a) the original signal; (b) the up-sampled signal; (c) the interpolated signal [6]

Since all three steps in the re-sampling of a signal are linear, this process can be described with a single linear equation. Depending on the re-sampling rate, the re-sampling process will introduce correlations of varying degrees between neighboring samples.

For each pixel, its neighbors are the all pixels within a window of length $2N + 1$. If a signal is resampled by an arbitrary factor of p/q it is necessary to determine α_k which is the set of weights across the neighborhood.

$$(5.1)$$

$$y_i = \sum_{k=-N}^N \alpha_k y_{i+k}$$

This can be written as α_k is the set of weights across the neighborhood. Let a_i be the i th row of the resampling matrix A p/q.

$$\vec{\alpha}_i = \sum_{k=-N}^N \alpha_k \vec{\alpha}_{i+k} \quad (5.2)$$

A resampled pixel y_i is correlated with its neighbors whenever its corresponding row of the resampling matrix α_i can be written as a linear combination of its neighboring rows [6]. This will result in periodic correlations between resampled pixels since the resampling matrix is periodic.

Then Expectation-Maximization Algorithm is used to determine the set of periodic samples that are correlated to their neighbors and the specific type of correlation between them.

At the initial stage it's assumed that each sample belongs to one of the two samples M1 or M2. M1 represents the samples that are correlated to their neighbors and M2 corresponds to the samples which are not correlated (outlier model).

The EM algorithm is a two-step iterative algorithm. In the Estimation step (E-Step) the probability that each sample belongs to each model is estimated and in the Maximization step (M-Step) the specific form of the correlations between samples is estimated.

In the E-Step Bayes' Rule is used to calculate the probability of each sample (y_i) belonging to model M1.

$$\Pr\{y_i \in M_1 | y_i\} = \frac{\Pr\{y_i | y_i \in M_1\}}{\sum_{k=1}^2 \Pr\{y_i | y_i \in M_k\} \Pr\{y_i | y_i \in M_k\}} \quad (5.3)$$

The output of the EM algorithm is a set of coefficients α and a probability map P .

Then the periodicity between the correlations are found using the Fourier Transform of the probability map. Here a peak detection algorithm has to be developed to enhance the peaks and suppress any correlations that occur due to the statistics of the original image.

5.2.2.2 Apply Fourier Transformation

The Fourier transformation of the probability map $P(\omega_x, \omega_y) = F(x, y) \cdot W(x, y)$ takes the form of,

$$f(r) = \begin{cases} 1 & 0 \leq r < \frac{3}{4} \\ \frac{1}{2} + \frac{1}{2} \cos\left(\frac{\pi\left(r - \frac{3}{4}\right)}{\sqrt{2} - \frac{3}{4}}\right) & \frac{3}{4} \leq r \leq \sqrt{2} \end{cases} \quad (5.4)$$

The probability map was Fast Fourier Transformed and Discrete Fourier Transformed independently. Then the resulted Fourier coefficients are high pass filtered to remove undesired low frequency noise. Following algorithm described in [2] was used for high pass filtering the periodicity map in the frequency domain.

- 1) Apply the following window to the Fourier transformed probability map. This gives the output $P(w_x, w_y) = F(p(x, y) \cdot W(x, y))$. Here the radius is normalized between 0 and $\sqrt{2}$.

$$f(r) = \begin{cases} 1 & 0 \leq r < 3/4 \\ \frac{1}{2} + \frac{1}{2} \cos\left(\frac{\pi\left(r - \frac{3}{4}\right)}{\sqrt{2} - \frac{3}{4}}\right) & \frac{3}{4} \leq r \leq \sqrt{2} \end{cases} \quad (5.5)$$

- 2) High pass filter the result to suppress low frequencies.

$$h(r) = \frac{1}{2} - \frac{1}{2} \cos\left(\frac{\pi r}{\sqrt{2}}\right), \quad 0 \leq r \leq \sqrt{2} \quad (5.6)$$

- 3) Normalize and Gamma correct the high passed signal to enhance the peaks to make them more distinct.

$$P_G = \left(\frac{P_H}{\max(|P_H|)}\right)^4 \times \max(|P_H|) \quad (5.7)$$

These values in the frequency domain are known as the periodicity map. The periodicity map is used to extract features for the SVM classifier.

Once the undesired low frequencies are removed, the high passed spectrum is normalized and gamma corrected in order to enhance frequency peaks then rescaled back to its original range.

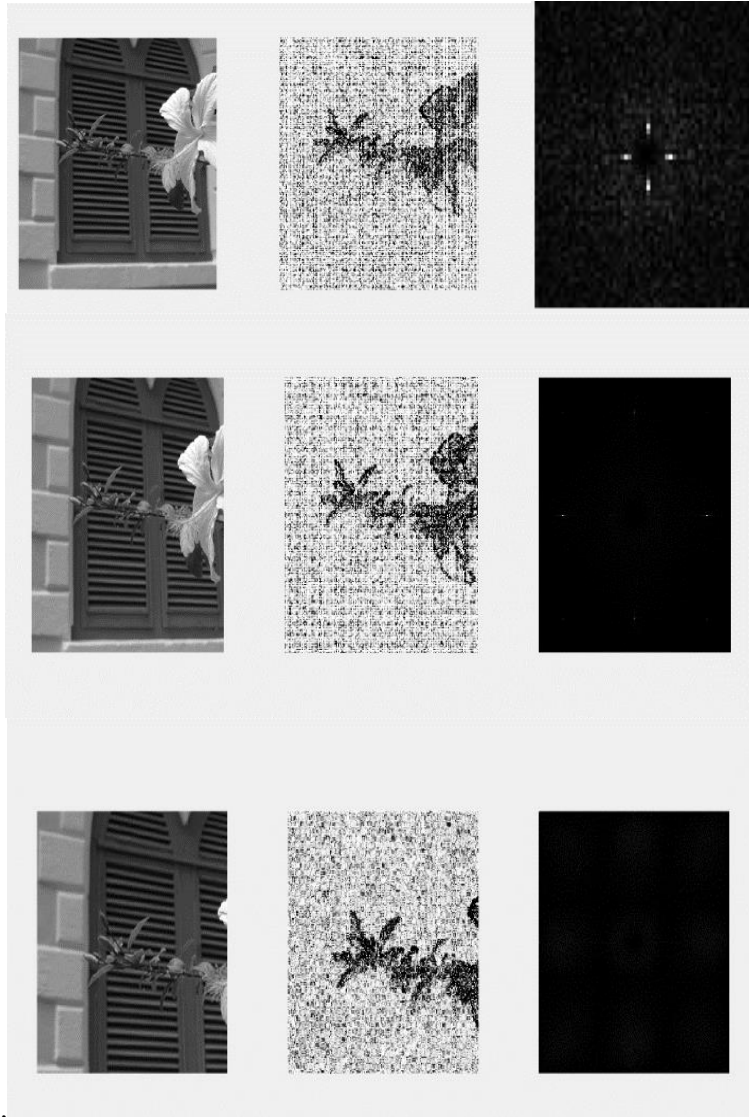


Fig 5.5: The Original Image, probability maps and & the periodicity maps of a resampled images for resampling rates of 10%, 20%, 30% and 60%)

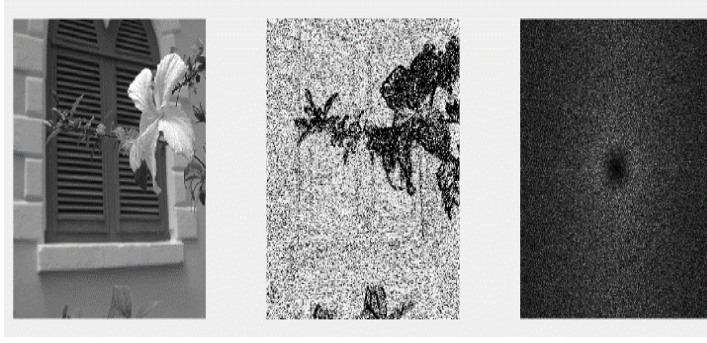


Fig 5.6: The Original Image, probability map and & the periodicity map of a non-resampled image

5.2.2.3 Classification of the Periodic Map Using SVM

We are going to use a SVM based classifier to classify the resampled images and not resampled images into two different classes based on the differentiation between natural peaks in an image and introduced peaks in the process of resampling. The periodic map generated in the previous map is used to extract the features required to differentiate between natural peaks in an image and introduced peaks in the process of resampling.

In the development of the SVM there are two steps as training the model and testing the module. For training the model we are going to use two groups of images which contains a balanced amount of resampled images and non-resampled images. These resampled images are resampled using various methods such as resizing, rotation etc.

These features were calculated twice for the Fast Fourier Transformation and Discrete Fourier Transform. The feature vector was calculated using the equations described in [6] and following additional features. So a feature vector of twelve features was used.

1) Here we are going to get the summation of the highest four coefficients of the respective periodicity map of an image because higher peaks indicate resampling.

$$f_1 = \sum_{i=1}^n \sqrt[m]{|c_i|} \quad (5.8)$$

2) The four largest coefficients generated by peak detection using Fast Fourier Transform

The summation of the four largest coefficients resulted from the applying of the Fast Fourier Transform was taken as a feature.

$$f_2 = \sum_{i=1}^n \sqrt[m]{|c_i|} \quad (5.9)$$

3) The four largest coefficient distance from center

As the natural peaks of an image have less frequencies, the four coefficients with highest frequencies are extracted and summed up.

$$f_3 = \sum_{i=1}^n \sqrt{u_i^2 + v_i^2} \quad (5.10)$$

4) The four largest coefficient to local energy ratio

This feature is extracted by considering a local rectangular window of width W and getting the respective ratio for that particular window and then getting the aggregation.

$$f_4 = \left(\sum_{i=1}^n 100 r_i \right)^m / 100 \quad (5.11)$$

5) Peak to total energy ratio

Here the ratio between the total energy of the image and peak is considered as a feature.

$$f_5 = (10^4 R)^m / 100 \quad (5.12)$$

6) The highest four coefficients after applying PCA to the periodicity map.

7) Four largest least squares after applying Partial Least Squares to the periodicity map.

Once these features are extracted, they are used as the inputs for the SVM Model. First the model is trained using the training set and then the testing set of images is used to verify the results. Then the trained model is capable of classifying a given image as resampled or not based on these features.

5.2.3 Image Splicing Detection Module

Passive splicing detection of an image is being addressed with the three following approaches.

- Noise pattern Analysis
- CFA (color filter array) interpolation Analysis

5.2.3.1 Noise pattern Analysis

In this approach the image is being segmented into similar like sections with accordance to the noise the image generates when it is being produced. The given image may include different level of noises in different regions. The task is to detect such regions and locate them in order to identify the spliced area of the image. To achieve this the following algorithm has been designed.

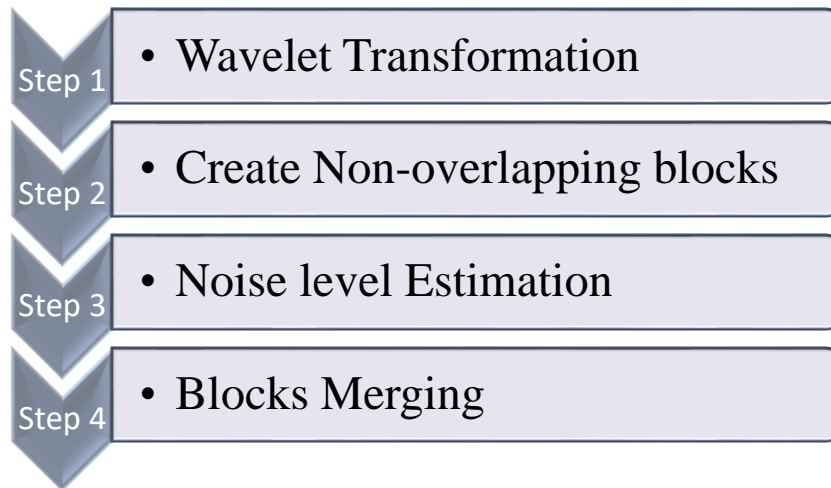


Figure 5.7: Steps of Noise Pattern Analysis Algorithm

As the initial step, the image noise level is defined as follows where $f_n(x,y)$ is the Image Noise level, $f(x,y)$ is the Uncorrupted Signal and $n(x,y)$ is the noise.

$$f_n(x,y) = f(x,y) + n(x,y) \quad (5.13)$$

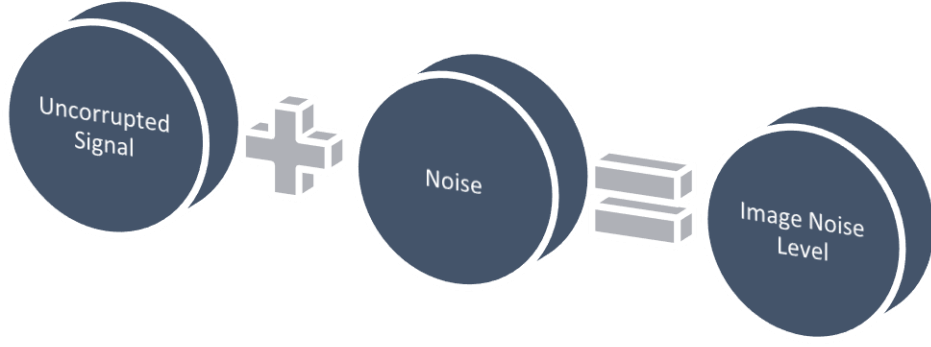


Figure 5.8: Image Noise level definition

Next, the image is converted into YCbCr model for further processing. For testing purposes the same implementation has been done with gray images. Once the image is converted into YCbCr model, its Y channel is being separated. Single level 2 dimensional wavelet transform is performed on the Y channel as the first step of the algorithm. For this Daubechies' external phase wavelet with 8 vanishing moments is used. As a result of the transformation, we receive a matrix with four coefficients. Hereafter we use Detail coefficient matrix diagonal for the further implementations. As the next step, the image is constructed in to non-overlapping blocks where these blocks are assumed to be smaller the size of the spliced sections where to be localized. For this segmenting process the HH1 sub-band which gives the diagonal details of the image with the highest resolution and the total number of blocks are determined by the following equation where the image is of $(M * N)$ pixels, R is the block size and B_t is the total number of non-overlapping blocks

$$B_t = \frac{M}{R} * \frac{N}{R} \quad (5.14)$$

As the proceeding step, the noise value of each segment will be estimated using wavelet-based noise estimation. Here we also calculate the standard deviation of noise. With that, homogeneous sub regions are classified based on the noise standard deviation condition. As the final step, region merging will be done comparing regions that have similar noise standard deviation values. As the final output, a map which includes regions with similar standard deviation of noise will be produced.

5.2.3.2 CFA interpolation Analysis

Splicing can disturb the CFA which is known as Colour Filter Array, interpolation in ways which the splices are rescaled filtered where it disrupts the inter-relations between the pixels. In general two cameras do not use the same CFA array or interpolation algorithms which makes the mixing of two separate images taken from different cameras can be detected and localized with the analysis of these parameters. Therefore the basic idea underneath this approach is detecting significant alterations of CFA demosaicing artifacts caused by the image splicing operation. The alterations could be either lack of artifacts or detection of weak artifacts.

Once a tampered image is given, its green channel is extracted first. The green channel is chosen out of Red and Blue channels, because it is always up sampled by a factor of two as described in Bayer pattern [31]. The Interpolated and Acquired samples are known as lattices. The implementation is based on the assumption that variance of prediction error in Acquired Lattice is always higher than the variance of prediction error in Interpolated Lattice. When there is no demosaicing in the image, it is assumed that there is no difference between variance values between Interpolated Lattice and the Acquired Lattice. Before this comparison, the prediction error for both lattices should be calculated.

Let's consider the observed image as $s(x, y)$ where x, y could be any Integer. The prediction error $e(x, y)$, could be defined as follows.

$$e(x, y) = s(x, y) - \sum_{\mu, v \neq 0}^{\infty} k_{\mu, v} s(x + \mu, y + v) \quad (5.15)$$

$k_{\mu, v}$ is a bi-dimensional prediction filter where it becomes similar to interpolation kernel of the demosaicing algorithm. Since the in-camera processing algorithms are varied from each other, we assume that the spatial pattern of the CFA is known. In this scenario, we take it as the Bayer CFA. As the next step, the weighted local variance is estimated.

When we get a $N \times N$ image, we consider it as a combination of $B \times B$ blocks. B is related to the period of Bayer's filter patterns. So the smallest value a B can obtain is 2. From the $B \times B$ block, the Feature $L(x, l)$ is determined by taking the logarithm of the ratio

between the geometric variance of $e(x, y)$ in acquired pixels and the same for the interpolated pixels. The theory behind this is shown by the below equation.

$$L(k, l) = \log \frac{GM_A(k, l)}{GM_I(k, l)} \quad (5.16)$$

Geometric mean of the variance of prediction errors at acquired pixel positions is calculated such a manner mentioned below.

$$GM_A = \left[\prod_{i,j \in B_{A_{k,l}}} \sigma_e^2(i, j) \right]^{\frac{1}{B_{A_{k,l}}}} \quad (5.17)$$

The same method will be followed for the geometric mean of the variance of prediction errors at interpolated pixel positions accordingly.

The calculated L value acts as the key part in determining the imbalance between two prediction errors in a tampered image. In generally, it is considered that L value always holds a positive amount. Therefore based on the above assumptions and statistical modes, we can decide that the tampered area of a photo gives a $L(k, l)$ value closer to zero where an untampered area gives a significant higher value. EM algorithm has been implemented to generate the forgery map. Forgery map is being generated using the likelihood ration and its maximizations. Bayes' Theorem is based for this likelihood map generation. In order to avoid noise, we cumulate the feature values and generate the map afterwards.

5.2.4 Image Retouching using filters Detection Module

The Filtering Detection Module is intended to distinguish the pictures which have modified by applying a linear or non-linear filter to improve the nature of the picture or shroud the produced territory of picture. It can be recognized the sort of the filter that applied to unique picture.

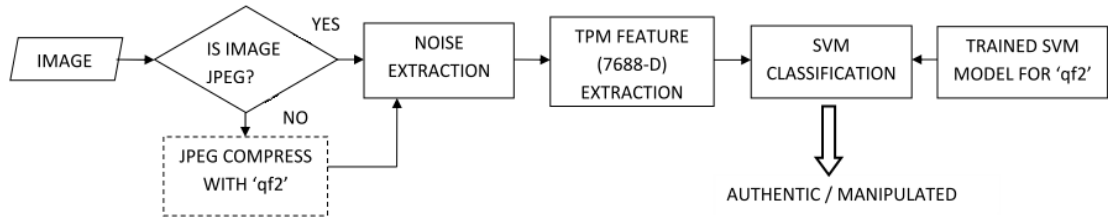


Figure 5.9: The Process Flow of Image Retouching using filters Detection Module

In here it ought to consider the way that identification technique can have the capacity to distinguish that filter is applied whether not considering the way that the kind of filter linear or nonlinear, compressed or uncompressed image. In above chart demonstrate the approach for recognizing the filter. In here utilize the method of when filtering and compressing is applied to a image, the spatial correlation of the pixels in the image gets disturbed. In [6] it is spoken to the compression noise of unfiltered image has low pass characteristics and when filter is applied, relationship of the image pixels is changed. In here it is utilized this conduct of compression noise to assemble a solitary model that can distinguish given picture has applied a filter or not. Here it is utilized the quantization compression noise demonstrate show by [6] and an adjustment to the prior image model to remove compression noise. And furthermore it is included some adjustments by incorporating the impacts of blocking artifacts due to compression. It is extricated compression noise and afterward demonstrated as a first order spatial ergodic Markov chain [7-10] which has been ended up being a powerful feature. These highlighted features are used to identify whether filter is applied or not. Here for the most part beneath strategies are relied upon to actualize on the general framework.

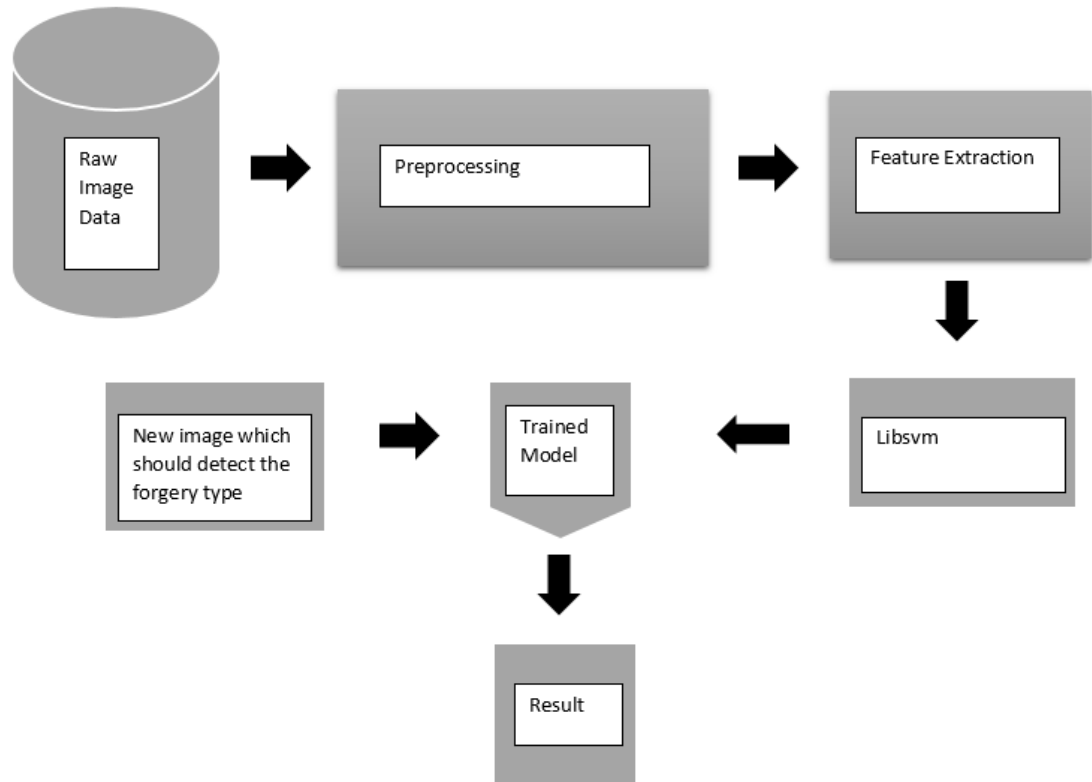


Figure 5.10: The complete design of the Image Retouching using filters Detection Module

Take the normal image at that point change over into vectorized form of 8×8 block. Then process each block separately. In here consider each block in dct domain. Then we need to take the quantization noise. Then this quantization noise is represented as a zero mean multivariate Gaussian. Here we utilize this representation for building a image model to easily get the compression noise. It additionally utilize the Markov Random Field Modeling and Huber function to fabricate the image model. Then we apply the Bayesian MAP estimation for extricating the compression noise from earlier and noise model. By utilizing this estimation we can figure the noise evacuated form of image. Using this compression noise removed image we can compute the noise from subtracting from the primary picture which has connected the compression and filter. Then the removed noise represented as first order ergodic spatial Markov chain. Here highlights that we use to describe this noise is progress likelihood matrix. Here we can separate 7688 features for each picture and this remaining parts unaltered the regardless of the span of the image. Then these features are utilized to prepare a SVM display and group a test image as filtered or unfiltered utilizing the prepared model. Here we utilize image dataset of 3000 NCID pictures and UNID pictures and crude picture of Dresden

picture database. Here we apply linear non-linear filters for image dataset and some are remains unchanged. Here we compress UCID picture with a specific quality factor. Here we likewise utilize the Libsvm for the classification likewise we need to consider the piece measure when we preprocess the pictures for preparing.

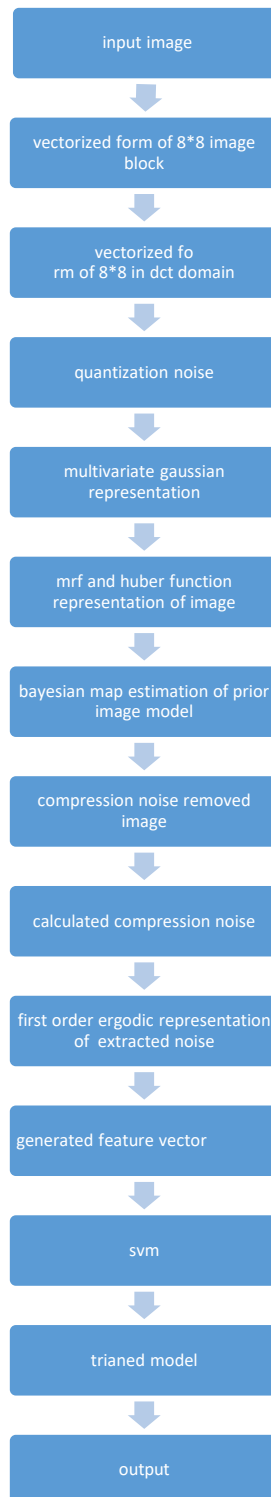


Figure 5.11: Flow chart for the Image Filtering Detection Module

In the first step it is divided the original image into 8*8 block as non-overlapping blocks. Here it denotes I is the input image. Then takes the vectorized 8*8 block image as the Z in the spatial domain. Then takes Y as the vectorized form of the corresponding block in DCT domain. Here all the processing are done as separate single blocks. Then calculate the compression noise for a non-overlapping block as

$$e_z = Z - Z_q \text{ where } Z_q \text{ is the quantized block} \quad (5.18)$$

D = 64 is the number of dimensions of the multivariate Gaussian.

Then use a natural language image model based on the Markov Random Field (MRF). It represent a local structure of image using conditional probability distribution where image pixels values depend only on its neighborhood.

Then take a joint distribution given by Gibbs measure as,

$$p(Z) = \frac{1}{\beta} \exp \left(-\lambda \sum_{c \in C} \rho T(d_c^t Z) \right) \quad (5.19)$$

β is a normalized constant.

c are local groups called cliques C is the set of all such cliques depending on the neighborhood structure of the Hubber Markove Random Field(HMRF).

$\rho T(\cdot)$ is the Hubber function.

d_c^t extract differences between a pixel and its neighbors such that the prior model generates to,

$$p(Z) = \frac{1}{\beta} \exp \left(-\lambda \sum_{m=0}^{M-1} \sum_{n \in Nm} \rho T(Z[n] - Z[m]) \right) \quad (5.20)$$

Where,

Nm is the index set of neighbors for the

m^{th} pixel, and M is the number of pixels in the block.

In the next step apply a denoising algorithm which uses a modified version of the above model. Then extracts the features and classify.

Then it is used a modified version of the above given equation (5.17) for quantization noise extraction. Then modified version of the Hubber function is derived as bellow.

$$\rho T(u) = \begin{cases} wu^2 & |u| \leq T \\ w(T^2 + 2T(|u| - T)), & |u| > T \end{cases} \quad (5.21)$$

$$w = \begin{cases} 1 \\ \gamma \end{cases} \quad \forall Z(u): u \in S, \text{ otherwise}$$

Here T the threshold set. Parameters

γ and T are empirically determined. S is the set of pixels which belong to the border pixels in each 8 * 8 block. Then it is used the Bayesian MAP estimation for extracting the compression noise from the prior and the noise model. The maximum a posteriori (MAP) criterion is,

$$\hat{Z} = \underset{Z}{argmax} p(Z|Z q) = \underset{Z}{argmax} p(Z)p(Z q|Z) \quad (5.22)$$

Here Z is the final estimate for the block after removing the compression noise. Though

Z q is a deterministic function of Z, the $p(Z q|Z)$ term in the above equation is considered as a Gaussian random variable with mean Z and auto covariance K_{eZ} . After substituting we can get,

$$\hat{Z} = \underset{Z}{argmax} \left\{ \frac{1}{\beta} \exp \left(-\lambda \sum_{m=0}^{M-1} \sum_{n \in N_m} \rho T(Z[n] - Z[m]) \right) \left(\frac{1}{(2\pi)^{D/2} |K_{eZ}|^{1/2}} \exp \left(\frac{-1}{2} e_Z^T K_{eZ} e_Z \right) \right) \right\} \quad (5.23)$$

Here the argument of exp (.) in eq(5.20) is minimized using the gradient descent algorithm. Then it is generate the noised image I by combining all the resulting denoised

non overlapping blocks. Here compression noise N_c for the image is then obtained as $N_c = I - \hat{I}$

Then it is calculate the transition probability matrix for the feature extraction. Noise extracted from a JPEG compressed image is modeled as a first order ergodic spatial Markov chain in below way.

$$p(X_{t+1} = x | X_1 = x_1, X_2 = x_2, \dots, X_t = x_t) \quad (5.24)$$

Where X_{t+1} is the present state and x_1, x_2, \dots, x_t are the previous states. Here we use the transition probability matrix (TPM) for characterize the noise.

Design should be considered with Dataset design.

5.4 Summary

The chapter 5 describes the analysis and the design of all four modules with illustrations as well as the necessary steps of algorithms. It gives a comprehensive understanding about how the project work is carried out under different modules.

6. Implementation

6.1 Introduction

This chapter describes the implementation of proposed solution under four modules that are Image Cloning Detection, Image Resampling Detection, Image Splicing Detection and Image Retouching with filters Detection. The details on how the previously mentioned technologies are used to implement the identified solution is included here. All the significant and important implementation steps are shown by pseudo codes.

6.2 Implementation

The implementation process of four modules are described below under sub chapters.

6.2.1 Image Cloning Detection Module

Segmentation → it is done using the scikit image library. This will generate a mask of segments with specific integer value for each segment. With the x,y values of a SIFT feature and this mask we can decide to which segment the key point is falling into.

Feature Localization → the obtained SIFT features are segmented into different patches obtained through SLIC algorithm, based on the position of the key point.

The clustered key points are held by a dictionary of patch integer value as the key and an array of “KeyDes” class as the value. That mean we have added all the key points found in a particular patch under the patch number in the dictionary. The definition of the class is as shown below,

```
class KeyDes:

    def __init__(self, keypoint=None, descriptor=None):

        """this act as the constructor"""

        self.keypoint = keypoint

        self.descriptor = descriptor
```

Then this obtained dictionary of classified key points is used to match between the key points and hence get the matched clusters.

Patch Matching → this is the most computationally intensive and longest process in the forgery detection. Because here we iterate the classified key point dictionary itself and on it again, without revisiting the checked key again. All together this will give $(n \times n - n!)$ number of iterations if the size of the dictionary is n .

Now this iteration means that we iterate through the patches that we have identified with key points. But we need to iterate through the key points of each patch to match with other. For this we iterate the added key points in the dictionary inside the iteration of dictionary key values that is already explained above. Now this will multiply the above iterations by a factor of $(\sum a_i x b_j)$ where i is the number of key points in the i th patch and j is that in the j th patch.

Now within each of this iteration euclidean distance is calculated between two patches. And in case the distance is not within the threshold then we keep track of the distance measure we obtained and the pair of key points participated as we need to do HOG calculation later in case we did not get any matches in the above mentioned process.

Now if we found the matches under the threshold then we stop the process from here and visualize the tampered region.

The different approaches are tested to get the best possible matching that we can have to get the most accurate result with least computational intensity.

The BFMatcher in OpenCV library provide KNN based matching. This reduces the computational requirement to a great extent. This is done as shown below.

```
# BFMatcher with default params  
  
bf = cv2.BFMatcher()  
  
matches = bf.knnMatch(description, description, k=2)
```

Suspect patch detection → In case we do not get any SIFT key points in the process up to now then based on the assumption that there can be some possible tampering between the patches where there are no or not the required amount of SIFT keys are detected, then they are checked for copy move forgery using HOG descriptor to extract the features.

Here what we do is taking HOG descriptor of rounded off area of one suspect patch and then same the same size area from the other patch. Now as we apply HOG algorithm we'll receive descriptor of same size and these are compared with each other to check whether they lie within our threshold defined. If they are matching then those two patches are selected as forged areas.

The method of key point extraction and matching of each of the approaches are discussed below.

6.2.1.1 Approach 1

The simplest way that works in most efficient manner. The SLIC algorithm in skimage library is used to get the segmentation mask and OpenCV library is used to get SIFT features extracted and to do the KNN match which are discussed above.

6.2.1.2 Approach 2:

Initially SIFT features are extracted and does a feature matching to check if we have any kind of possible forgery. Then a dictionary is created to get the patches without required amount of key points in it as shown below (this mainly aim to detect the possible forgery done at plane areas.),

`dictionary = {segmentValue:int, listOfSIFTDescriptors(2D list)}`

Now this dictionary is used to get the length of SIFT descriptors for each segment value and if there is no enough SIFT descriptors (it is one in this case) then that segment is considered to be eligible to get the HOG descriptors.

For getting the HOG descriptor OpenCV HOG descriptor is used with the following dimensions,

search window size $\rightarrow 32 \times 32$

block size $\rightarrow 16 \times 16$

cell size $\rightarrow 8 \times 8$

similar to the SIFT key point localization here too we use a dictionary to localize the HOG features as below,

```
dictionary = {segmentValue:int, listOfHOGDescriptors}
```

Now using KNN clustering of OpenCV BFMatcher we repeatedly match between the descriptors under different segment key values of the dictionary.

Though this is aimed to get the forgery detected in the areas that do not have SIFT features, this became a failure due to the reason that this implementation demand lot of computational power.

Hence the approach 3 was developed to get the task done in lesser computational power.

6.2.1.3 Approach 3:

Here we find the SIFT and HOG descriptors of the image as done in approach 2. the change done here was that the matching of SIFT and HOG features as individual bunch rather than matching localized features. This drastically saved a lot of time, yet the issue arrived here was that the HOG descriptors was not strong enough to describe textures in great detail, resulting HOG gave high false-positive detection.

Detection was carried out with varying HOG window sizes (8×8 - 32×32) and Block sizes (8×8 - 16×16) with fixed cell size of 8×8 . We can not use the default window size of HOG 128×64 that is implemented in OpenCV as a patch will not cover an area of 128×64 .

6.2.1.4 Approach 4:

This approach is a modification on the approach one, which experiments a successful way to detect the cloned area more correctly detecting most number of true positive cloned pixels.

Here as we have discussed earlier the SIFT keys are extracted using OpenCV and Segmentation mask is taken from SLIC algorithm. Then for each segment we identify the least area square possible to cover the segment and that area of image is considered as the region of interest – ROI. Now SIFT features of this segment and from the resulting image once this ROI is removed are stored in a dictionary.

Where dictionary key is the segment value and value is an object as follows,

```
class Features:

    def __init__(self, query_keys: list, query_des: numpy.ndarray, train_keys:
numpy.ndarray, train_des: numpy.ndarray):

        """this is the constuctor of the method"""

        self.query_keys = query_keys

        self.query_des = query_des

        self.train_keys = train_keys

        self.train_des = train_des
```

Now this object hold the query keys – SIFT features obtained from ROI and train keys – SIFT features obtained from the image that had removed ROI.

Then matches are done between query_des (descriptors of ROI) and train_des (descriptors of ROI removed image).

Then based on the matches we can predict for the forged area.

This matching introduced two additional thresholds than previous distance_threshold for SIFT descriptor matching, that is,

SIFT key points per segment → the required key points need to be matched with keys of ROI

number of segments per match → the number of segments that match with the ROI

6.2.2 Image Resampling Detection Module

The implementation of the Image Resampling Detection module was divide into two main parts as follows.

- Generation of the periodicity map
- Extracting features from the periodicity map and developing the Support Vector Machine for Classification

The implementation was carried out using the Matlab IDE.

6.2.2.1 Generation of the periodicity map

As mentioned above in the previous chapter, resampling introduces periodic correlations between pixels. These should be differentiated from the natural correlations exist between neighboring pixels to determine whether an image is resampled or not. For that Expectation Maximization (EM) Algorithm was used. Following pseudocode was used for the implementation of the EM algorithm.

```
/* Initialize values*/  
choose a random  $\vec{\alpha}$   
choose N and  $\sigma_0$   
set  $p0$  to the reciprocal of the range of the signal  $\vec{y}$   
set h to be a binomial low-pass filter of size  $(N_h) \times (N_h)$ .  
  
n = 0  
  
repeat  
  for each sample  $i$   
    Do the Expectation Step  
    Calculate the residual error  
  end
```

```

W = 0

repeat
  for each sample  $i$ 
    Do the Maximization Step
    Calculate new weights
end

```

128 x 128 window size of an image was used for simplicity and as tampering in small blocks can be identified better. So as the first step of the image resampling the image was cropped. Then we had to select values for $\vec{\alpha}$ for the first iteration of the E step and for σ of the Gaussian distribution. The initial values were assigned as follows. These values were determined by observing the results generated by the periodicity map.

```

% initialize parameters
alpha = rand(1,wlen^2-1)';
alpha = alpha./sum(alpha(:)); % normalize weights
sigma = 0.75;

```

Once these values were determined implementation of the Expectation Maximization algorithm was done using the code segments attached in the Appendix B.

The output from the periodicity map for a non-resampled image and a resampled image can be shown as follows.

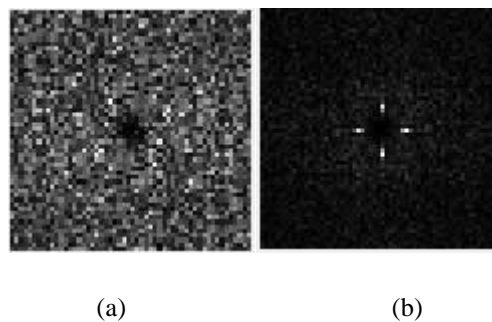


Figure 6.1: The periodicity map of a non-resampled image (a) and resampled image (b)

The periodic patterns that can be seen in the resampled image was generated due to correlations introduced during resampling. So it can be used to indicate resampling.

Then this periodicity map is Fast Fourier Transformed and Discrete Fourier Transformed and high pass filtered using the algorithm described in the previous chapter suppress natural peaks available in an image.

6.2.3 Image Splicing Detection Module

Image Splicing Detection was carried out under 3 different approaches. They are:

- Noise Pattern Analysis
- Colour Filter Array Interpolation Analysis

6.2.3.1 Noise Pattern Analysis

Noise Pattern Analysis Module was implemented with Matlab hence the matrix handling and other required processing were easier with it. High Resolution, Not skewed images that are not undergone with image resampling were needed for the testing purposes. Therefore a suitable dataset was developed by combining well-known datasets such as Dresden [21] and CASIA [22]. Different functions and scripts were developed for noise pattern estimation. The Pseudo code for the algorithm is shown at the end of this chapter with necessary code segments.

Once we select the image, it will be sent through “CleanUpImage” function to cover many extreme cases that appear in real world datasets. The input images could be with more than 3 channels and occasionally uint16 images. Therefore this function converts such images into 3 channel uint8 images.

Once the image is cleaned, the noise map should be generated. In order to do that image should be separated into non merging blocks. The BlockSize was determined as 32 after a series of experiments. The noise level analysis was done inside the GetNoiseMap function.

As the first step of analysis, the image is converted into YCbCr model. Then its Y component, as known as Luma component is separated for further processing. Discrete Wavelet Transformation is applied on top of the Y component in order to denoise the image. For that Daubechies wavelet with level 8 decomposition was selected after

comparing and contrasting with other wavelet families available. As a result of the wavelet transformation, we get four types of coefficient matrices such as:

- cA – Approximation Coefficient Matrix
- cH – Horizontal Detail Coefficient Matrix
- cV – Vertical Detail Coefficient Matrix
- cD – Diagonal Detail Coefficient Matrix

Out of the above, cD which returns as a double array, is selected for further processing hence it has image properties stored with itself. Separating the cD into blocks of 32 is the task lined up next. But the original cD array might not be a multiple of 32. Therefore cD has to be rounded off before merging it into blocks. After the rounding off process, a 3 dimensional matrix is created and filled with zeros to fill the blocks that are separated from the image.

For loop is used to store each and every block inside the 3 dimensional matrix. The for loop increment differs according to the selected BlockSize.

After the blocks are separated, the noise of each block will be estimated by further calculations as mentioned in [23]. The estimated noise levels are stored in a 2x2 matrix.

The noise map is plotted as the final output.

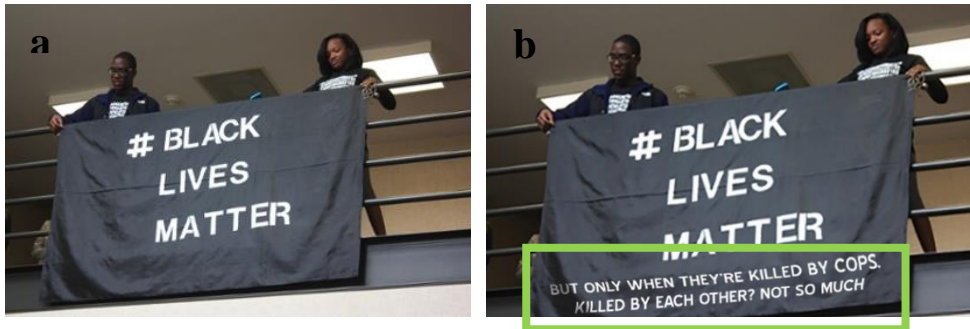


Figure 6.2: The Authentic Image (a) and the respective Tampered Image (b)

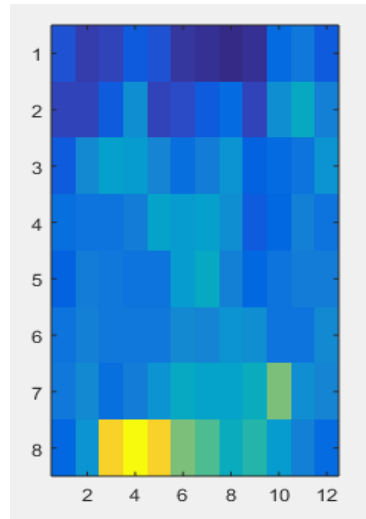


Figure 6.3: The respective Noise map with the localization of the tampered area.

Pseudo Code

Algorithm: Noise Pattern Analysis

Input: High resolution tampered image that hasn't under gone any resampling

Output: Noise map of the image with localization of tampered area

Process: The code segments used are attached in the Appendix B.

6.2.3.2 Colour Filter Array Interpolation Analysis

Colour Filter Array (CFA) Analysis Module was also implemented with Matlab. Finding suitable dataset for testing and evaluation purposes was one of the most difficult tasks found during the implementation. Images that are taken from different cameras were needed for the testing purposes. Therefore a suitable dataset was developed with the use of Benchmark Dataset [23]. The Pseudo code for the algorithm is shown at the end of this chapter with necessary code segments.

Image Interpolation can be taken as the process of estimating values of certain pixels with the aid of its neighboring pixel locations. The images come from digital cameras show demosaicing artifacts on each and every blocks with respect to CFA elements specially when there are no further processing. In this approach, the demosaicing artifacts are studied in local level which leads to image forgery detection.

The algorithm is mostly based on Bayes' Theorem and related statistical methods as described under the Analysis and Design chapter. Once an image is submitted its green colour component is chosen for the prediction error calculation. The prediction error for both interpolated and acquired scenarios are calculated and their differences are extracted. The Prediction error value is always higher in Acquired scenario whereas Interpolated scenario tends to give a value closer to zero. As the next step, the image is separated into smaller size blocks and calculate the feature L according to the equation (5.13). With the use of this likelihood value, the differences between two variances can be identified which finally leads to the forged area detection. Forgery map is developed with the use of the EM algorithm and the Bayes' theorem.

The figure 6.4 and 6.5 shows the output received from the algorithm when N_b value as known as feature dimension is set to 2. In addition to the above shown results, the probability map and the histogram map could be retrieved for different N_b values. For the testing purposes this algorithm outputs $N_b = 8$ and $N_b = 2$ histogram map and the probability map. After a series of experiments it was decided that the $N_b = 2$ is the ideal value.

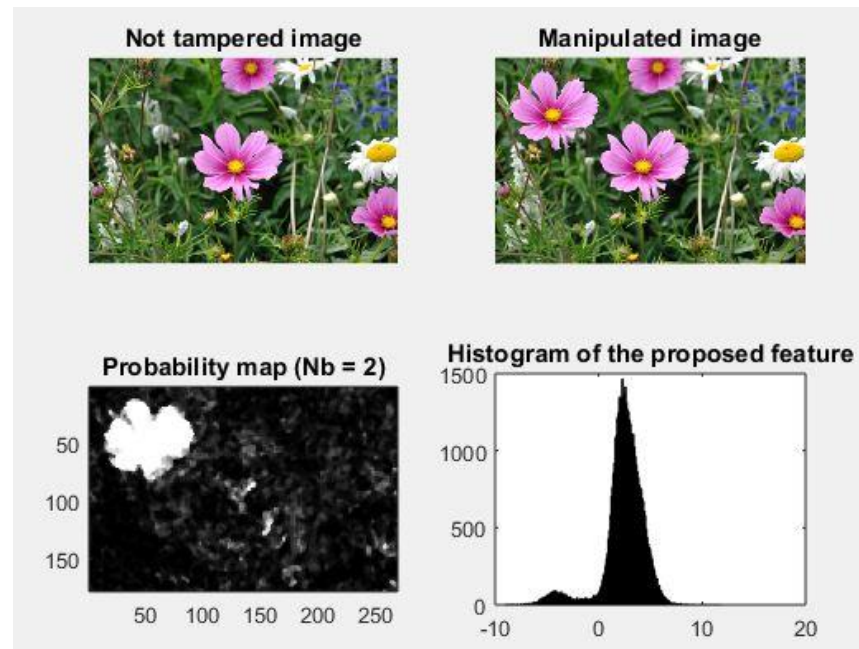


Figure 6.4: The Authentic Image, Tampered Image, Probability Map and the Histogram of the proposed feature.

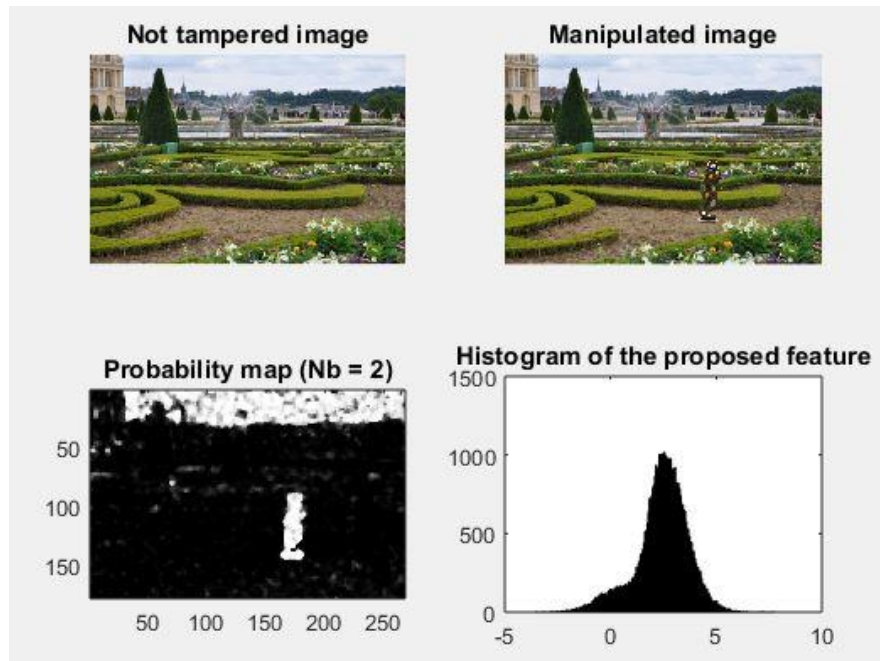


Figure 6.5: The Authentic Image, Tampered Image, Probability Map and the Histogram of the proposed feature.

Pseudo Code

Algorithm: CFA Interpolation Analysis

Input: High resolution tampered image – tampered area should be captured from a different camera other than the camera used to capture the authentic image.

Output: Probability map of the image with localization of tampered area and the Histogram Map for the feature calculated

Assumption : The 2x2 Bayer pattern for Green colour is known.

Process: Codes are attached in the Appendix B.

6.2.4 Image Retouching using Filters Detection Module

In the first step we get the denoised image. Then we generate the features using this extracted noise. Here also included the code for gradient calculation.

Here extracted features are sent to Libsvm and build a model to classify the upcoming images.

Here we preprocess the image dataset. Take the images from UCID images, NCID images and Dresden image database. NCID images are single compressed with a random factor $qf1 \in [2, 30; 80]$. Here we generate two classes of images for our experiments from these sets. Unfiltered and Filtered. In experiment a JPEG/TIFF image is forged, enhanced using filter randomly from median, Gaussian, average or unsharp filters and saved in JPEG format following a typical forgery pipeline. When given image is uncompressed TIFF format, it is JPEG compressed. Compression noise and TPM features are extracted from 5000(1000 unfiltered and 4000 filtered) images.

Pseudocode of the Image Filtering Detection is as follows.

```
for i=1 to end
    seperate into 8*8 block
    for block
        generate_compression_noise_removed_block
```

```
subtract_noised_remove_block_from first_block
        generate_compression_noise
    end
end
```



```
add_each_block_and_generate_complete_image  
add_each_block_compression_noise  
generate_feature_by_representing_first_order_ergodic_chain  
end  
Input feature vector to svm
```

7. Evaluation and Results

7.1 Introduction

This chapter showcases the results obtained for four modules through the implementations that are proposed. The summarized version of results received through experiments are attached herewith. The Authentic Images, Tampered Images and the relevant intermediate graphs and other maps are shown here. The results appear here lead to the conclusions of this research.

7.2 Evaluation and Results

7.2.1 Image Cloning Detection Module

The proposed solution is capable of finding multiple forged areas. But the drawback is the high computational intensity with the process.

For the calculation of following image the statistics was as follows with the first approach.

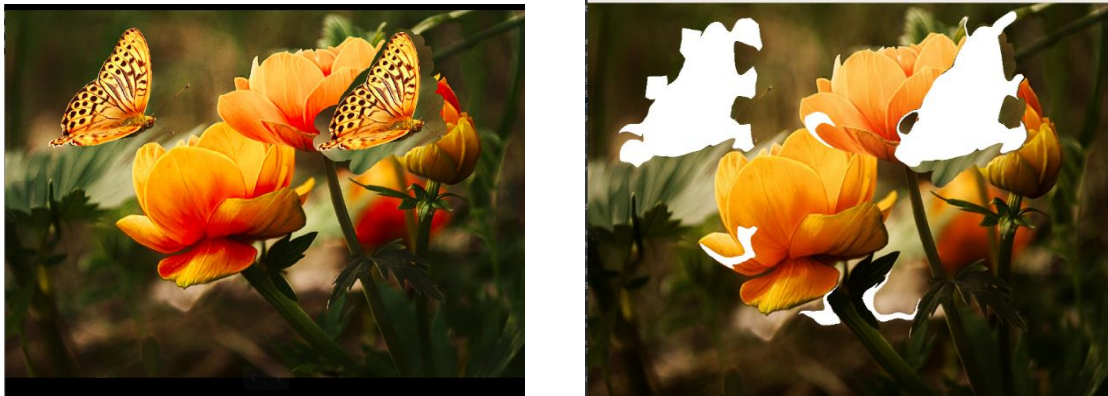


Figure 7.1: Results of Image Cloning Detection Module

Number of key points detected – 1741

time of execution - 62.843568325042725 s

shape of image : (712, 1072, 3) \rightarrow (height, width, channels)

number of matched patches – 2

upper bound of segments – 500 units

threshold for euclidean distance between two key points – 10

7.2.1.1 Approach 1:

The approach one is the fastest and is giving the most impressive results for lesser computation.

7.2.1.2 Approach 2, Approach 3:

There have been failures in detecting the correct forged area, where there are no any SIFT features are detected. The main reason for the failure is due to giving the HOG descriptor high amount of false positive matches as the descriptor is not robust in detecting the features on non blob areas.

7.2.1.3 Approach 4:

The most robust method out of the all 4 approaches and is capable if detecting the most number of cloned pixels.

This approach has been tested for an image set of 58 images and the results are as shown below.

		True Condition	
	Total Population	Condition Positive = 28	Condition Negative = 30
Predicted Condition	Predicted Condition Positive	True Positive=22 (Power)	False Positive = 1 (Type I Error)
	Predicted Condition Negative	False Negative = 6 (Type II Error)	True Negative = 29
		True Positive Rate (TPR), Sensitivity, Recall = TP/Condition Positive = 22/28 = 0.7857	False Positive rate (FPR), Fall Out = FP/Condition Negative = 1/30 = 0.0333
		False Negative Rate (FNR), Miss rate = FN/Condition Positive = 6/28 = 0.2142	True Negative Rate (TNR), Specificity = TN/Condition Negative = 29/30 = 0.96667

Table 7.1: Evaluation of Image Cloning Detection Module

Prevalence = Condition Positive/ Total Population =28/58 = 0.4827	Accuracy (ACC) = (TP+TN)/ Total Population =51/58= 0.8793	
Positive Predictive Value (PPV) =Precision = TP/Predicted Condition Positive = 22/23 = 0.9565	False Discovery Rate (FDR) =FP/ Predicted Condition Negative =1/35 = 0.0.285	
False Omission Rate (FOR) = FN/ Predicted Condition Negative =6/35 = 0.1714	Negative Predicted Value (NPV) = TN/ Predicted Condition Negative =29/35 = 0.8285	
Positive Likelihood Ratio (LR+) = TPR/FPR = 0.7857/0.0333 = 23.5945	Diagnostic Odds Ratio (DOR) = LR+/LR- =23.5945/0.2215 = 4.7946	F1 Score = 2 / {(1/Recall) + (1/ Precision)} =2/ {(1/0.7857) + (1/0.9565)} = 0.8627 = 0.9
Negative Likelihood Ratio (LR-) = FNR/TNR = 0.2142/0.96667 = 0.2215		

Table 7.2: Evaluation of Image Cloning Detection Module

7.2.2 Image Resampling Detection Module

The classification results generated by the SVM under different criteria such as resampling rate, image compression format used are analysed here. The classifier was tested for images resampled using Adobe Photoshop across a range of resampling ratios.

The results are summarized below.

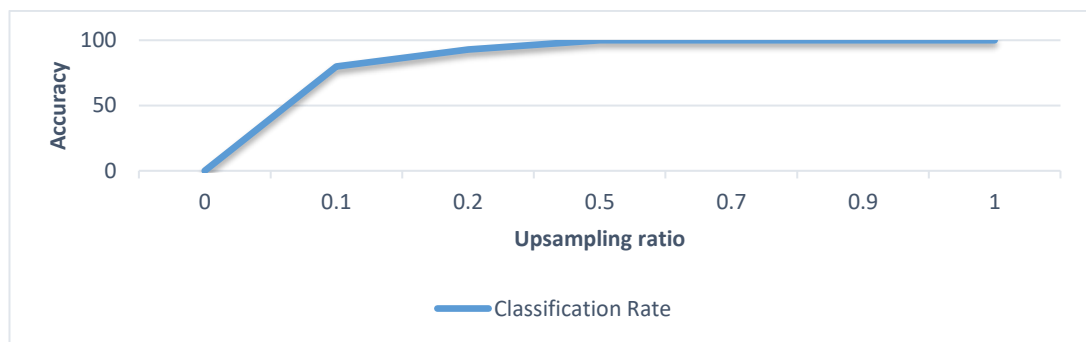


Figure 7.2. Classification rate for up-sampled images

It was observed that the system functions at optimum level for resampling rates over 20% and the performance is comparatively low at resampling rates less than 20%.

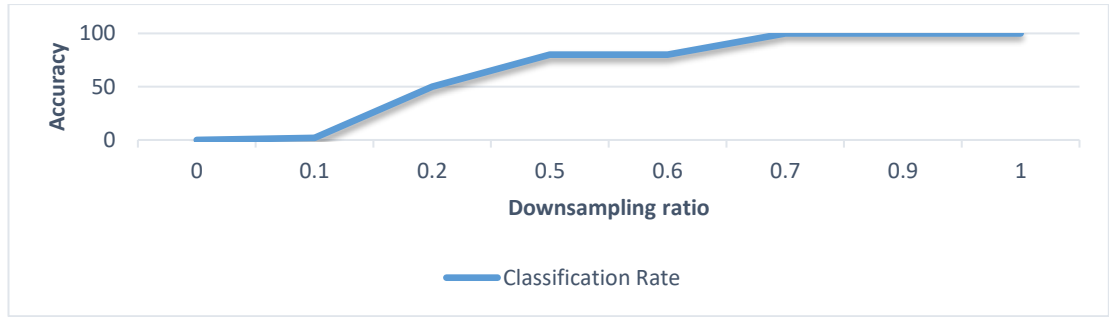


Figure 7.3. Classification rate for down-sampled images

When down-sampling of an image is considered it was observed that the prediction rate is comparatively low when the resampling rate is less than 40%. The reason is that a detectable linear correlation is not generated in those instances.

No significance difference was seen between the classification of the images of TIFF and PNG formats. The reason is both TIFF and PNG formats use lossless data compression algorithms.

But when compared with JPG compressed images although the classification rate was similar to PNG compressed format, the localization of the resampled area was relatively low due to the fact that the blocking artifacts in JPEG images introduced during JPEG compression and correlations introduced in image resampling overlapping with each other.



Figure 7.4. Resampled image

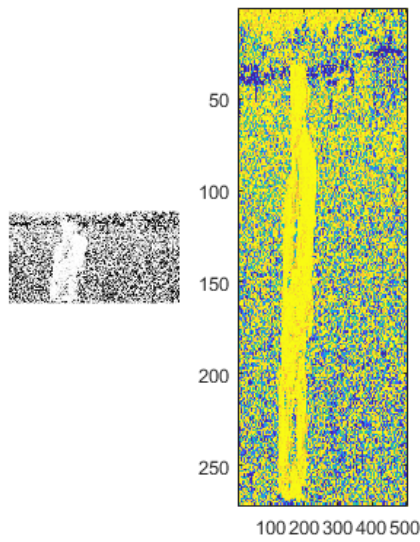


Figure 7.5. Localizing using TIFF format images

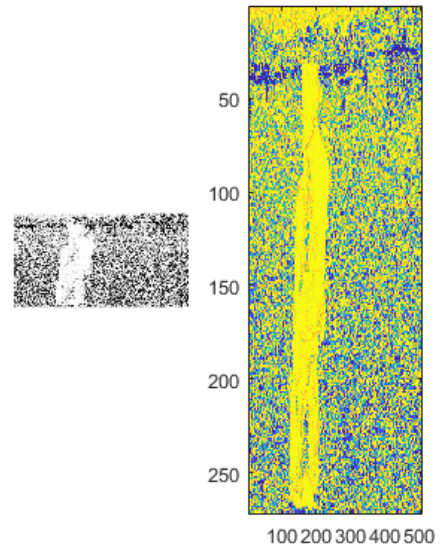


Figure 7.6. Localizing using PNG format images

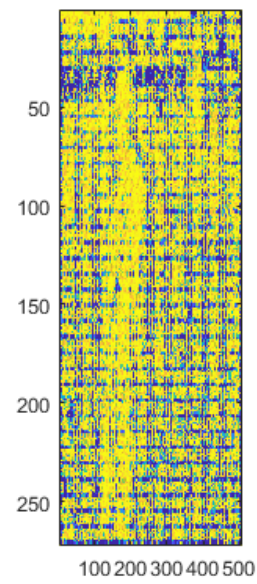


Figure 7.7. Localizing using JPEG format images

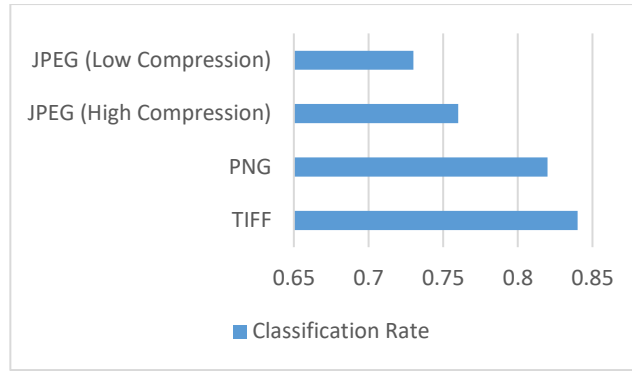


Figure 7.8. Comparison between classification of different image formats

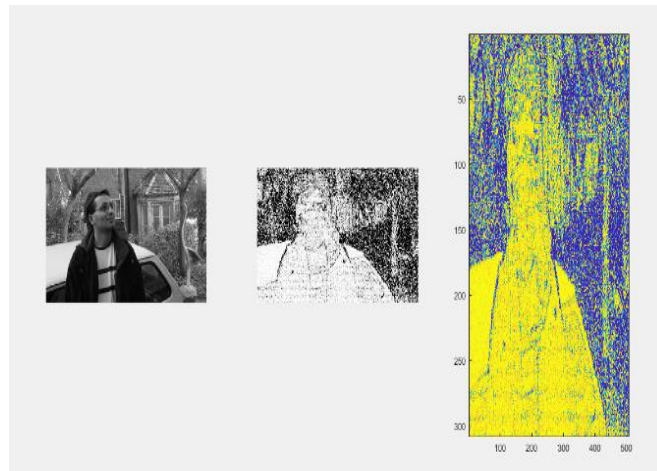


Figure 7.9. The Original Image, probability map and the resampled area localized image

The above image was constructed by merging the following two images from the UCID Image Dataset and then resampling accordingly. It clearly shows the resampled area in yellow colour. So the classifier is capable of detecting the resampled and non-resampled areas distinctly. The system is also capable of detecting iterative resampling with high accuracy.



Figure 7.10. The Original Images used to construct Figure 7.9

The system was evaluated using Confusion Matrix and ROC Curve. The system has an overall accuracy of 81% for the testing dataset of 100 images. While the precision rate is 0.84 which shows a significant improvement from the synthetic map based approaches.

	True Condition	
Total no. of observations = 100	Condition Positive	Condition Negative
Predicted Condition Positive	True Positive 42	False Positive 11
Predicted Condition Negative	False Negative 08	True Positive 39
	True Positive Rate (42/50) = 0.84	False Positive Rate (11/50) = 0.22

Table 7.3: Confusion Matrix for predicted data

Prevalence (50/100) = 0.5	Accuracy (42+39)/100 = 0.81
FOR (08/50) = 0.16	FDR (11/50) = 0.22
PLR (0.84/0.22) = 3.89	NPV (39/50)
NLR (0.16/0.78) = 0.25	DOR (3.89/0.25) = 15.56
F1 Score 2/(1.2+1.2) = 0.83	

Table 7.4: Sensitivity analysis for predicted data

	Non Resampled	Resampled
Recall (%)	89/100 0.89	84/100 0.84
Precision (%)	90/96 0.93	84/95 0.88

Table 7.5: Recall and Precision rate of the SVM classifier

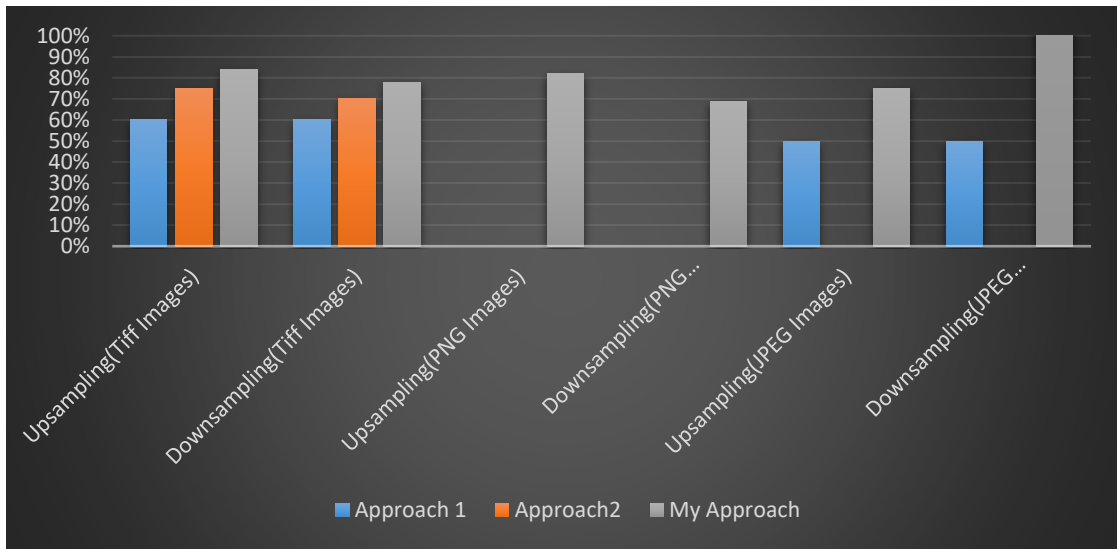


Figure 7.11. Comparison between Existing Approaches and Our Approach

7.2.3 Image Splicing Detection Module

7.2.3.1 Noise Pattern Analysis

The results of the evaluation is described here.

The noise pattern analysis algorithm was tested with hundreds of tampered images. The algorithm was able to successfully localize tampered areas in most of the times. Figures from 7.4 to 7.6 shows some of the successful scenarios.

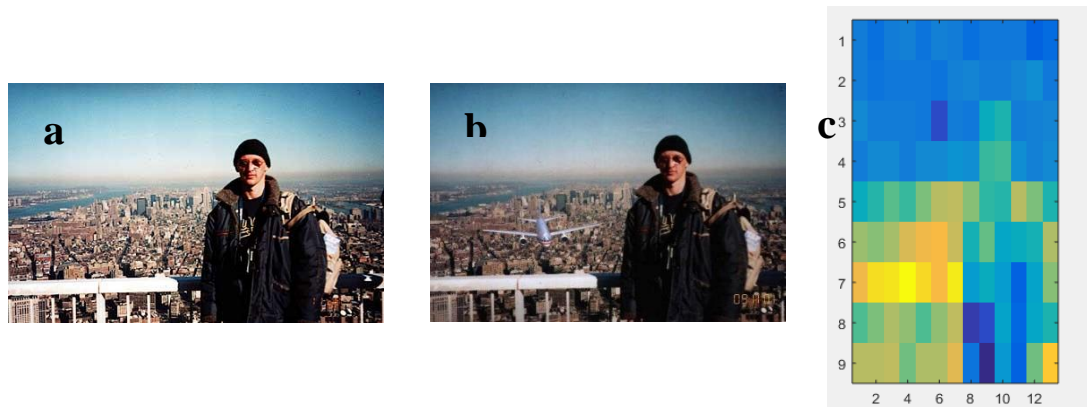


Figure 7.4: Authentic Image (a), Tampered Image (b) and the Noise Map (c)

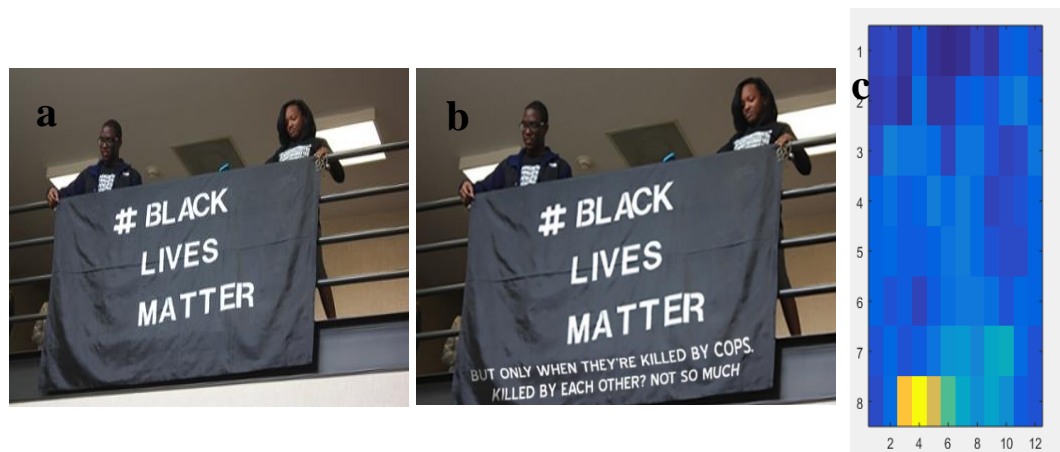


Figure 7.5: Authentic Image (a), Tampered Image (b) and the Noise Map (c)

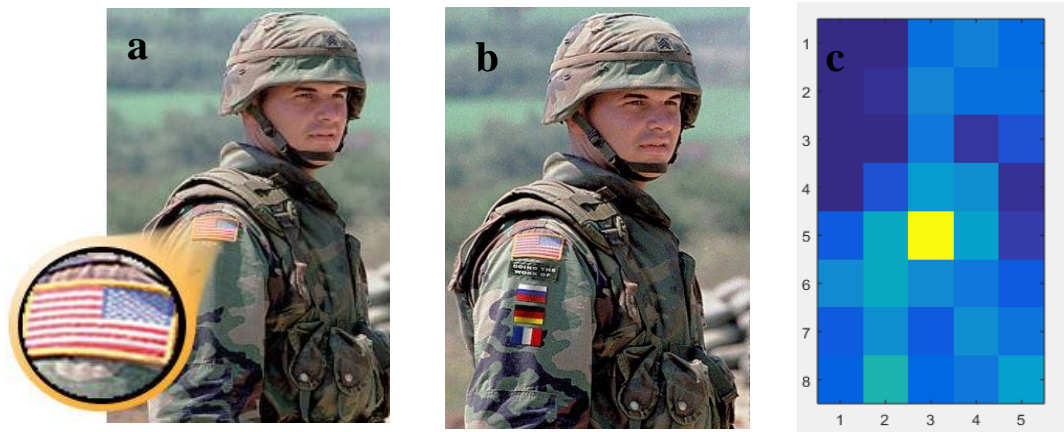


Figure 7.6: Authentic Image (a), Tampered Image (b) and the Noise Map (c)

Furthermore there were some scenarios where the algorithm was unable to detect the exact tampered area. There are many reasons behind those failures. When the images are very low in resolution, when the images are modified with different filters and other processing techniques, when the images are skewed, etc. the algorithm fails to give the correct localization. Some of the failed scenarios are shown under figure 7.7 and 7.8

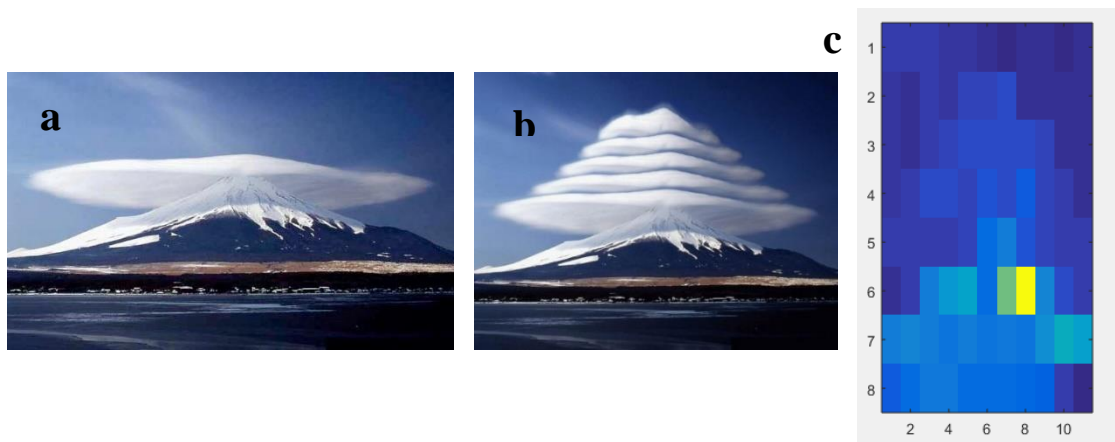


Figure 7.7: Authentic Image (a), Tampered Image (b) and the Noise Map (c) which fails to localize the area correctly.

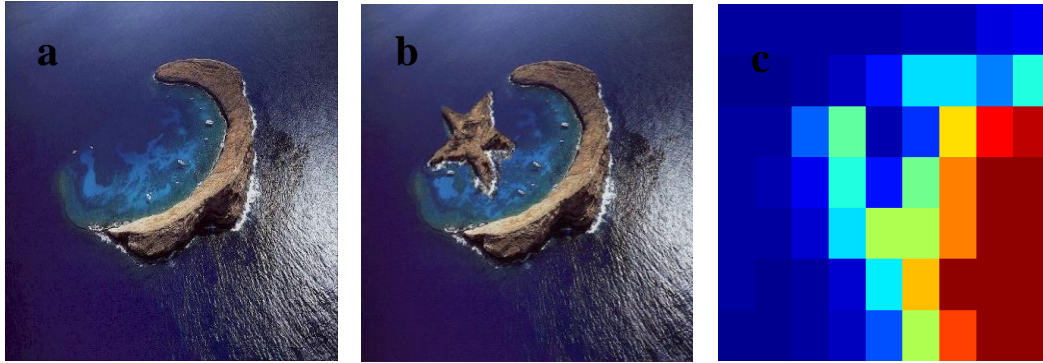


Figure 7.8: Authentic Image (a), Tampered Image (b) and the Noise Map (c) which fails to localize the area correctly.

In this algorithm when we define the tampered area and the authentic area, we consider the least percentage of an area of an image as the tampered one and the larger area as the authentic area as the authentic one after a comparison. But sometimes it is not the intended answer by human. Such scenario is shown under figure 7.9.

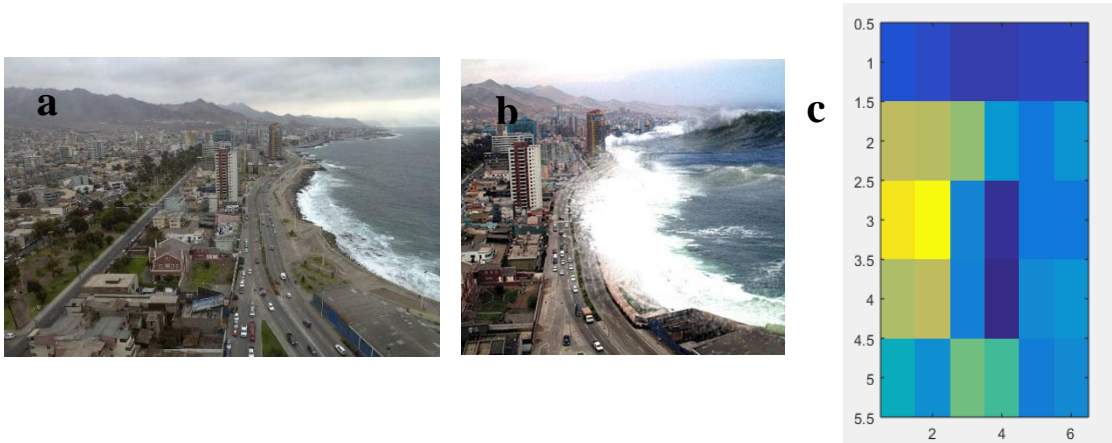


Figure 7.9: Authentic Image (a), Tampered Image (b) and the Noise Map (c) which shows the swapped version of the intended localization.

As the final step of the evaluation the algorithm was tested with a sample of 100 images comprising of 86 tampered images and 14 untampered images. The True Positive, True Negative, False Positive and False Negative values were calculated accordingly. With the recorded statistical values, the Accuracy, Sensitivity and other significant

measurements were calculated. The statistical Charts for all the calculations are shown below.

		True Condition			
	Total Population	Condition Positive = 67	Condition Negative = 33	Prevalence = Condition Positive/ Total Population = $67/100 = 0.67$	Accuracy (ACC) = $(TP+TN)/$ Total Population = $72/100 = 0.72$
Predicted Condition	Predicted Condition Positive	True Positive=56 (Power)	False Positive = 17 (Type I Error)	Positive Predictive Value (PPV), Precision = $TP/Predicted$ Condition Positive = $56/73 = 0.7671$	False Discovery Rate (FDR) = $FP/ Predicted$ Condition Negative = $17/73 = 0.2328$
	Predicted Condition Negative	False Negative = 11 (Type II Error)	True Negative = 16	False Omission Rate (FOR) = $FN/ Predicted$ Condition Negative = $11/27 = 0.4074$	Negative Predicted Value (NPV) = $TN/ Predicted$ Condition Negative = $16/27 = 0.5925$
		True Positive Rate (TPR), Sensitivity, Recall = $TP/Condition$ Positive = $56/67 = 0.8358$	False Positive rate (FPR), Fall Out = $FP/Condition$ Negative = $17/33 = 0.5151$	Positive Likelihood Ratio (LR+) = $TPR/FPR = 0.8358/0.5151 = 1.6225$	Diagnostic Odds Ratio (DOR) = $LR+/LR- = 1.6225/0.3384 = 4.7946$
		False Negative Rate (FNR), Miss rate = $FN/Condition$ Positive = $11/67 = 0.1641$	True Negative Rate (TNR), Specificity = $TN/Condition$ Negative = $16/33 = 0.4848$	Negative Likelihood Ratio (LR-) = $FNR/TNR = 0.1641/0.4848 = 0.3384$	

Table 7.6: Statistical results for Noise Pattern Analysis

Out of the 84 tampered images, the system was able to identify the tampered regions correctly in 56 images. Therefore the True Positive value holds 56. 16 Untampered images were correctly identified as authentic images without localizing any area as tampered. In 17 images non-tampered areas were identified as tampered regions which leads to the type 1 error giving False Positive value as 17. In 11 images, tampered areas were recognized as non-tampered areas. False Negative value holds 11 and the Error is known as the Type 2 error. The Sensitivity (Recall) for the experiment is 0.8358 and the Precision value is 0.7671. The F1 Score is rounded off to 0.8.

With all the tested images, I would like to state that the Noise Pattern Analysis algorithm has around 72% accuracy if and only if the input images has the required characteristics.

7.2.3.2 CFA Interpolation Analysis

The results of the algorithm are described below.

The CFA Interpolation Analysis algorithm was tested with different tampered images that are tampered with images taken from different cameras. For the testing purpose, a unique dataset was built using images that are available in Benchmark dataset [24].

But at the moment right now, we have very limited number of images that do not violate the requirements of the algorithm. Therefore the final accuracy level determination was difficult for some extent.

The below figures show some successful scenarios.

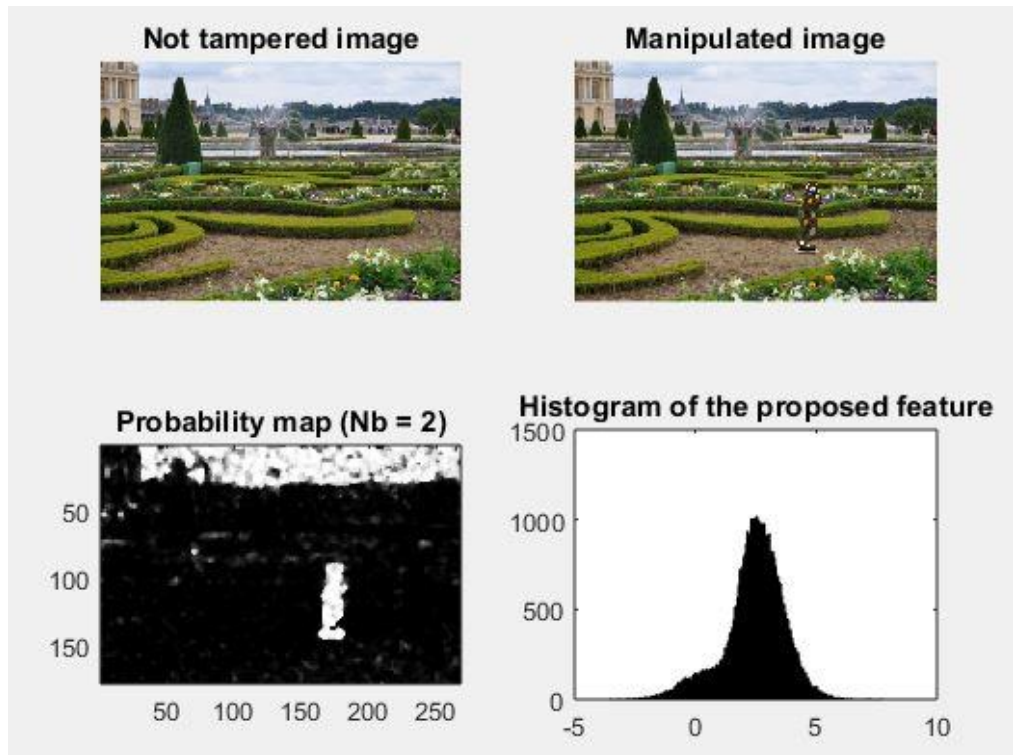


Figure 7.10: A Successful Scenario

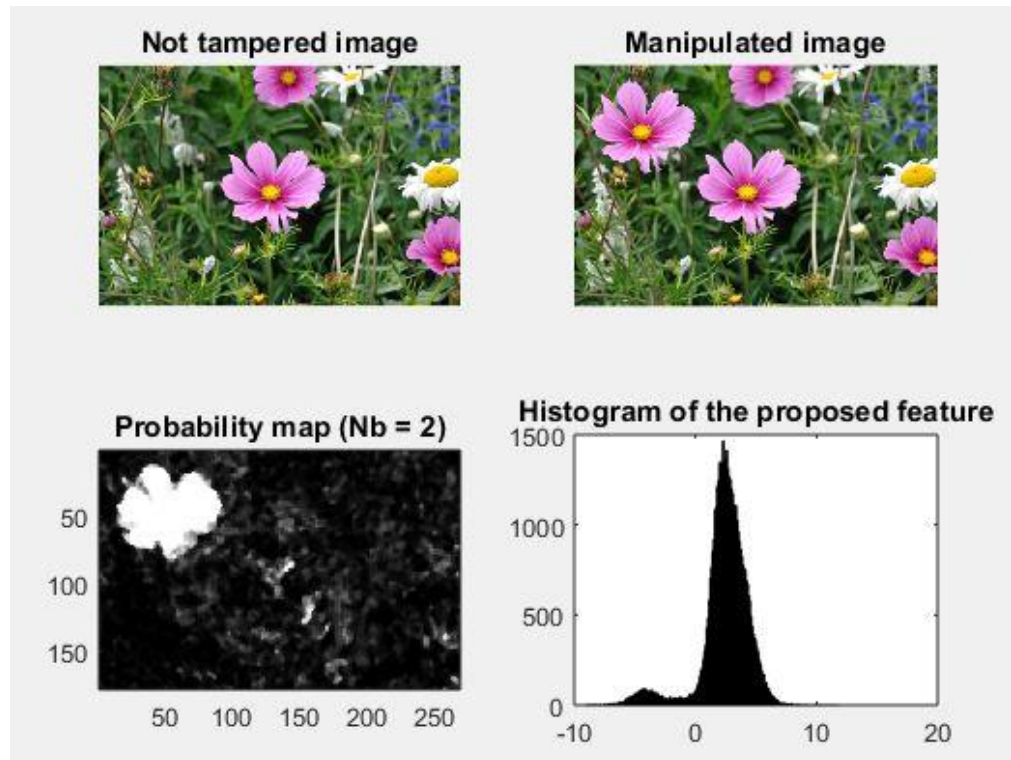


Figure 7.11: A Successful Scenario

The following figure show an unsuccessful scenario where the both tampered area and the authentic image are taken from the same camera.

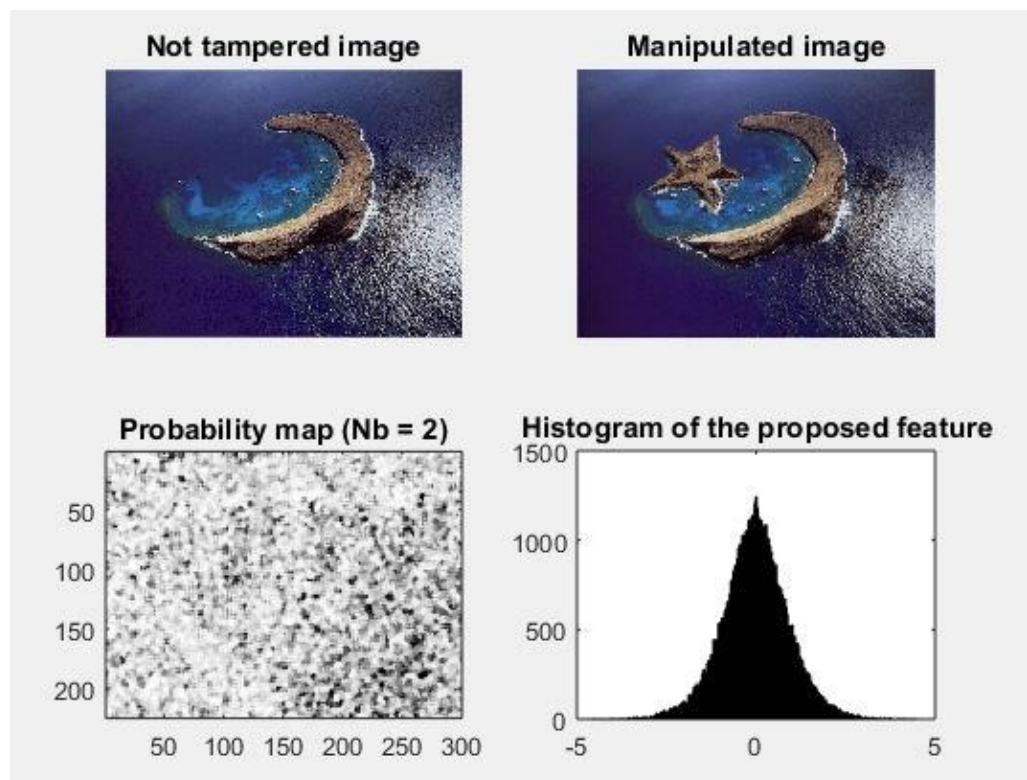


Figure 7.12: An Unsuccessful Scenario

As the final step of the evaluation the algorithm was tested with a sample of 100 images comprising of 84 tampered images and 16 untampered images. The True Positive, True Negative, False Positive and False Negative values were calculated accordingly. With the recorded statistical values, the Accuracy, Sensitivity and other significant measurements were calculated. The statistical Charts for all the calculations are shown below.

		True Condition			
	Total Population	Condition Positive = 75	Condition Negative = 25	Prevalence = Condition Positive/ Total Population = $75/100 = 0.75$	Accuracy (ACC) = $(TP+TN)/$ Total Population = $63/100 = 0.63$
Predicted Condition	Predicted Condition Positive	True Positive=49 (Power)	False Positive = 11 (Type I Error)	Positive Predictive Value (PPV), Precision = $TP/Predicted$ Condition Positive = $49/60 = 0.8167$	False Discovery Rate (FDR) = $FP/ Predicted$ Condition Negative = $11/60 = 0.1833$
	Predicted Condition Negative	False Negative = 26 (Type II Error)	True Negative = 14	False Omission Rate (FOR) = $FN/ Predicted$ Condition Negative = $26/40 = 0.65$	Negative Predicted Value (NPV) = $TN/ Predicted$ Condition Negative = $14/40 = 0.35$
		True Positive Rate (TPR), Sensitivity, Recall = $TP/Condition$ Positive = $49/75 = 0.6533$	False Positive rate (FPR), Fall Out = $FP/Condition$ Negative = $11/25 = 0.44$	Positive Likelihood Ratio (LR+) = $TPR/FPR = 0.65/0.44 = 1.4847$	Diagnostic Odds Ratio (DOR) = $LR+/LR-$ = $1.4847/0.2619 = 5.6689$
		False NegativeRate (FNR), Miss rate = $FN/Condition$ Positive = $26/75 = 0.1467$	True Negative Rate (TNR), Specificity = $TN/Condition$ Negative = $14/25 = 0.56$	Negative Likelihood Ratio (LR-) = $FNR/TNR = 0.1467/0.56 = 0.2619$	

Table 7.7: Statistical results for Noise Pattern Analysis

Out of the 86 tampered images, the system was able to identify the tampered regions correctly in 49 images. Therefore the True Positive value holds 49. 14 Untampered images were correctly identified as authentic images without localizing any area as tampered. In 11 images non-tampered areas were identified as tampered regions which leads to the type 1 error giving False Positive value as 11. In 26 images, tampered areas were recognized as non-tampered areas. False Negative value holds 26 and the Error is known as the Type 2 error. The Sensitivity (Recall) for the experiment is 0.6533 and the Precision value is 0.8167. The F1 Score is 0.7259.

With all the tested images, I would like to state that the Noise Pattern Analysis algorithm has around 63% accuracy if and only if the input images has the required characteristics.

7.2.4 Image Retouching with Filters Detection Module

Results of our experiment is given in below. By observing the results we can conclude that the classification accuracy between an unfiltered and a filtered image is on an average above 78% irrespective of the quality factor of the last JPEG compression. In below graph we can see the ROC. When we are taking the consideration of the graph we can see that TPR of 0.8 is achieved for considerable FPR of 0.16 taking into account multiple scenarios together. Here also average of 50 seconds takes for extracting compression noise from one image in a i7 core processor and 6GB RAM using MATLAB. Here compression noise extraction takes more time as each 8*8 block is denoised by gradient descent whose iterations and learning rate are empirically set. It is extracted 7688 features for one image. Here we use 5000 image data set for training. Separate 1000 images for one filter type and another 1000 images for non filter applied. First we used the libsvm for classification. But when predicting new images it was not generated a high accuracy. Therefore we had to considered the matlab multiclass classifier for predicting new image. When it checked the accuracy by inserting new images one by one generate a average of 78% of accuracy.

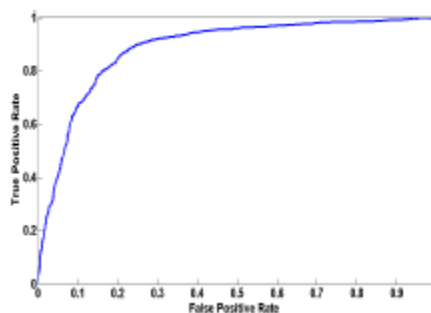


Figure 7.13: ROC graph

By studying previous result we can conclude that average of 80% accuracy can get using this approach without considering the type of filter.

8. Conclusion and Further Work

8.1 Introduction

This chapter shows the Conclusion and the further work of the research. It explains the progress of the research from the beginning itself.

8.2 Conclusion and Further Work

In this project we have implemented an image forgery detection system comprises of image cloning detection, image resampling detection, image splicing detection and image retouching detection. So far we have referred the available literature covering those areas. There we found the algorithms used, approaches followed so far which are applicable for our project.

Then the initial step carried out by us was analyzing the overall system and each module in the system in terms of input, process and output. The input of each module was identified as an image and the output is the detection result that defines whether an image is forged using a particular forging method or not. To design the process we identified the procedures needed to be followed and algorithms used in the approaches done so far. There we have to customize the available algorithms to suit our requirements and create new algorithms to improve the process in terms of efficiency and accuracy as most of the available methods require high computational cost.

Then we defined the objectives that need to be carried out for the successful completion of the system.

According to the proposed solution and its analysis with design, the implementation was carried out for all four modules. Due to the technical competencies, different technologies were

selected for different modules. The main challenges faced by us so far were understanding the available literature in terms of algorithms used, mathematical explanations, processes followed etc. This was the main challenge we faced even in the implementation phase.

After the implementation, we carried out evaluation and testing for separate modules. There we had to search for suitable images as input data. Sometimes we were unable to

find relevant datasets from other researchers where we had to implement our own datasets.

Each algorithm implemented for this research may have its own drawbacks and loopholes that are needed to be taken care of. As the future work, it is a must to improve the performance and the accuracy of these algorithms in order to cover a wide range of images. At the moment there are restrictions on input images from which we try to cover the pitfalls in these algorithms. But it would be really helpful if these algorithms are in a state where they are 100% capable to detect forgeries. Once the performance and the accuracy is increased, this system will be a trending one among people because of the rising necessities. Therefore we, Team Auxilium would like to invite anyone who is interested in this area to continue this research to a great height.

References

- [1] N. Kanagavalli, L. Latha, "A survey of copy-move image forgery detection techniques" in International Conference on Inventive Systems and Control (ICISC-2017).
- [2] L. Jian, L. Xiaolong, Y. Bin, S. Xingming, "Segmentation based image copy-move forgery detection scheme" in IEEE transactions on information forensics and security.
- [3] L. Yuanman, Z. Jiantao, "Image copy-move forgery detection using hierarchical feature point matching".
- [4] B. Xiuli, P. Chi-Man, Y. Xiao-Chen, "Adaptive Polar based filtering method for image copy-move forgery detection" in 2016 IEEE TrustCom-BigDataSe-ISPA.
- [5] M. Kirchner and T. Gloe, "On resampling detection in re-compressed images," 2009 First IEEE International Workshop on Information Forensics and Security (WIFS), 2009.
- [6] A. Popescu and H. Farid, "Exposing digital forgeries by detecting traces of resampling," IEEE Transactions on Signal Processing, vol. 53, no. 2, pp. 758–767, 2005.
- [7] P. Kakar, N. Sudha and W. Ser, "Exposing Digital Image Forgeries by Detecting Discrepancies in Motion Blur", IEEE Transactions on Multimedia, vol. 13, no. 3, pp. 443-452, 2011.
- [8] Qiguang Liu, Xiaochun Cao, Chao Deng and Xiaojie Guo, "Identifying Image Composites Through Shadow Matte Consistency", IEEE Transactions on Information Forensics and Security, vol. 6, no. 3, pp. 1111-1122, 2011.
- [9] T. de Carvalho, C. Riess, E. Angelopoulou, H. Pedrini and A. de Rezende Rocha, "Exposing Digital Image Forgeries by Illumination Color Classification", IEEE Transactions on Information Forensics and Security, vol. 8, no. 7, pp. 1182-1194, 2013.

- [10] Hardik Jain, Joydeep Das, Hemant Kumar Verma, and Nitin Khanna, "An Enhanced Statistical Approach for Median Filtering Detection using Difference Image"
- [11] G. Cao, Y. Zhao, R. Ni, L. Yu, and H. Tian. Forensic Detection of Median Filtering in Digital Images. In IEEE International Conference on Multimedia and Expo, pages 89–94, 2010.
- [12] M. Kirchner and J. Fridrich. On Detection of Median Filtering in Digital Images. In IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, 2010.
- [13] C. Chen, J. Ni, and J. Huang. Blind Detection of Median Filtering in Digital Images: A Difference Domain Based Approach. IEEE Transactions on Image Processing, 22(12):4699–4710, 2013.
- [14] Junyu Xu, Yun Ling and Xuan Zheng, "Forensic detection of Gaussian low-pass filtering in digital images," 2015 8th International Congress on Image and Signal Processing (CISP), pp. 819–823, 2015.
- [15] "MATLAB," Wikipedia, 20-Dec-2017. [Online]. Available: <https://en.wikipedia.org/wiki/MATLAB>. [Accessed:
- [16] M.A Robertson and R.L. Stevenson, "Dct quantization in compressed images," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, pp. 27–38, January 2005.
- [17] H. Ravi, A.V. Subramanyam, B. Avinash kumar, and G. Gupta, "Compression noise based video forgery detection," in Proc. IEEE International Conference on Image Processing, October 2014.
- [18] T. Pevny, P. Bas, and J. Fridrich, "Steganalysis by subtractive pixel adjacency matrix," IEEE Transactions on Information Forensic and Security, vol. 5, pp. 215–224, June 2010.
- [19] X. Zhao, S. Wang, S. Li, J. Li, and Q. Yuan, "Image splicing detection based on noncausal markov model," in Proc. IEEE Int. Conf. on Image Processing, September 2013, pp. 4462–4466.
- [20] D. Cozzolino, D. Gragnaniello, and L. Verdoliva, "Image forgery detection through residual-based local descriptors and block-matching," in Proc. I

- [21] T. Gloe and R. Böhme, "The `Dresden Image Database' for Benchmarking Digital Image Forensics, in Proceedings of the 25th Symposium On Applied Computing" (ACM SAC 2010), vol. 2, 1585-1591, 2010
- [22] Jing Dong, Wei Wang and Tieniu Tan, "CASIA Image Tampering Detection Evaluation Database" in 2013 IEEE China Summit and International Conference on Signal and Information Processing, 978-1-4799-1043-4
- [23] D. Donoho, I. Johnstone, Ideal spatial adaption by wavelet shrinkage, *Biometrika* 8 (1994) 425–455.
- [24] F. Perazzi, J. Pont-Tuset, B. McWilliams, "A Benchmark Dataset and Evaluation Methodology for Video Object Segmentation" in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1063-6919
- [31] Bayer pattern page

Appendix A

Individual Contribution to the Project

Name of the Student: Yagoda Y.G.J - 134200A

Identified a methodology to identify whether an image is applied with a linear or nonlinear filter at the same time. At the moment we have systems developed to individually identify whether an image is applied with linear or nonlinear filter but not with a single image.

To do this the existing algorithm flows are combined together in order to get a new flow of initially developing a model to train the SVM. The filter applied image is initially modified to remove the noises. Then the noise removed image is subtracted from the original image to get the noise only image. Now this image is used to develop the feature vector that is used for the SVM.

For this SVM training data sets were developed with linear filters like Gaussian, mean, and for nonlinear filter applied images like median.

Pre-processed the image set to get the SVM classification more accurate.

The OpenCV application development was heard in implementing the mathematical equations. Hence moved to Matlab implementation

Name of the Student: Waasala W.M.D.J.B - 134184B

Was able to implement and tested two new approaches to detect the copy move forgery of images. The main difference between the two is about the computational power required to resolve an image. That is due to the use of different algorithms in matching the key points, the Euclidean distance measure and KNN clustering based matching.

The flow of the approach was newly introduced to the knowledge gained so far. That is initially extracting the SIFT key points and extracting segments of the image. Then grouping the key points based on the segmentation. Next matching the patches based on the key points attached to the segment. Based on the matches we can determine whether the patch is cloned or not. In case there was no any matches then closest patches will be more examined with HOG descriptors to extract more features in the area and check for any possible forgery detection.

The approach was tested with the forgeries that are made with the opencv application like copy pasting using things like ROI, which gave perfect results of identification. But in coming to a cropping with a software like Gimp, and used a tool like cropping the key points are getting more distorted and the distance usually goes higher than the usual case that is explained above.

This situation was inspired to develop the next step in detecting the HOG descriptors for the most matched patch. Here the as the HOG cell get smaller and block get more bigger then the descriptor come more specific to the area, which is most of the time a texture than a whole object. But then the new issue arrived was the computation burden increased exponentially, due to exhaustive Euclidean distance calculation. This was drastically mitigated with the KNN clustering based best match that was introduced to the system. This flow of procedure can be further extended to detect what the geometrical change was applied in cloning. At the current situation SIFT can detect the copy move patch independent of the geometric transformation but cannot tell the exact geometric operation that has been applied. By reading about the possible methods to identify what the geometrical operation that was applied in cloning by using a transformation matrix which can be enabled through RANSAC algorithm (which is something like $x' = Hx$, where H says that the transformation of the matched patches.)

Name of the Student: Chandimali M.G.N - 134218K

I was responsible for the implementation of the Image Resampling Detection Module. The initial step was to read and understand the available literature available on Image Resampling Detection. There I had to go through a no. of research papers and understand the underlying architecture of the process. Then I had to design my module with the understanding gathered through them. While designing I had to determine the gaps exist in the available systems. Then the final design was developed to minimize those gaps such as accuracy, performance etc.

Then I had to learn how to work with Matlab and how to manipulate images, how to extract features from the images, how to design a SVM, how to optimize it etc. I also had to find the suitable datasets to train the classifier.

Next I started implementing the system. The first step was to generate the periodicity map. There I used the Expectation Maximization Algorithm to generate the probability map. Then to generate the periodicity map from it I had to learn about high pass filters available in Matlab as I had to high pass filter the probability map.

After the successful generation of the periodicity map I extracted features from it to train the SVM classifier. There I extracted many features to test the features that give the highest accuracy. Afterwards the extracted features were loaded to .csv files to generate the dataset. Then the dataset was divided into training set and testing test. I also had to learn about different kernel functions and test them with the extracted features to develop the model with highest accuracy. Once the model is developed with the training test it was tested and validated with the test data set and predictions were done.

Name of the Student: Mahawatta D.M.A - 134097K

Main responsibility that assigned to me was implementing the Image Splicing Detection Module and evaluate its accuracy. Before starting the implementation, I had to go through many different research papers in order to understand the context. In addition I had to get a complete knowledge on image processing concepts. With all the rapid developments in the field of Image Processing, The context has become huge. For the understanding purposes, I searched on existing solutions that have addressed the same problem that we focused. Once I identified the drawbacks of existing similar approaches, I had to research for novel methods of Splicing Detection. With the collected information on the relevant topic, I developed the design for my solution. With the guidance and support received by the supervisor, I was able to complete a novel design that handles most of the drawbacks of similar approaches.

Once the design was accepted, implementation process started. With the knowledge grabbed from other researchers' experiences, I decided the suitable technologies for the implementation. The main problem I had to encounter was understanding complex formulas and other statistical models which were used in the context. In addition I am responsible for developing two algorithms under two different approaches. Therefore the development was somewhat challenging and complex. With the knowledge I gathered from other various researches, I was able to complete two different approaches.

Evaluation and testing was the next challenge I faced due to the lack of data. I had to develop separate datasets according to the requirements. With that, the evaluation was carried out. Results were recorded for many test cases where successful and unsuccessful results were expected. Finally an overall accuracy level was calculated.

I personally had a very valuable experience with the implementation of this research. The knowledge gathering phase was very beneficial to improve my knowledge level on Image Processing. I was able to get to know novel methodologies and technologies that are used in the field.

Appendix B

Code Segments Used in the Project

Image Cloning Detection Module

- Code segment used for clustering the key points

```
def clusterKeypoints(keypoints: list, descriptors: numpy.ndarray, segments: numpy.ndarray) -> dict:
    dictionary = {}
    for keypoint, descriptor in zip(keypoints, descriptors): # index is starting from ZERO
        segmentVal = segments[int(round(keypoint.pt[1]))][int(round(keypoint.pt[0]))]
        # segments are in (row,col) order and keypoint.pt is in (x,y) order
        if segmentVal in dictionary:
            dictionary[segmentVal].append(KeyDes(keypoint=keypoint, descriptor=descriptor))
        else:
            dictionary[segmentVal] = [KeyDes(keypoint=keypoint, descriptor=descriptor)]
    return dictionary
```

- Code segment used for matching the clusters

```
def matchClusters(keypointCluster: dict, keyMatchPerCluster=1, threshold=40) -> dict:
    """this will match clusters of keypoints with other clusters"""
    matchedSegments = {}
    minimumThresh = 0
    minThreshKeyPair = None
    firstTime = True
    for index1, dictKeyVal1 in enumerate(keypointCluster): # here we get the segment value that was used as
the key
        for index2 in range(index + 1,
                               (len(keypointCluster))): # here we get the segment value that was used as the key
            for index2, dictKeyVal2 in enumerate(keypointCluster):
                if index2 > index1: # checking only if the index is pointing forward the current value
                    keypointMatchCount = 0
                    for keyDes1 in keypointCluster[dictKeyVal1]:
                        for keyDes2 in keypointCluster[dictKeyVal2]:
                            doubleDisVal = scipy.spatial.distance.euclidean(keyDes1.descriptor,
keyDes2.descriptor)
                            if doubleDisVal < threshold: # if under thresh then no problem we have
a forged area
                                keypointMatchCount += 1
                                break
                            else: # trying to get the minimum threshold value and identify the
possible forge area
                                if firstTime:
                                    minimumThresh = doubleDisVal
                                    firstTime = False
                                    # append the keypoint descriptor KeyDes object to x value
                                    minThreshKeyPair = Data(x1=keyDes1, y1=dictKeyVal1,
x2=keyDes2, y2=dictKeyVal2)
                                elif doubleDisVal < minimumThresh:
                                    minimumThresh = doubleDisVal
                                    # append the keypoint descriptor KeyDes object to x value
                                    minThreshKeyPair = Data(x1=keyDes1, y1=dictKeyVal1,
x2=keyDes2, y2=dictKeyVal2)
                                and segment mask value to y of two keypoints
```

```

        if keypointMatchCount >= keyMatchPerCluster:
            if dictKeyVal1 in matchedSegments:
                matchedSegments[dictKeyVal1].append(dictKeyVal2)
            else:
                matchedSegments[dictKeyVal1] = [dictKeyVal2]
    keyList = [*matchedSegments]
    segmentLength = len(matchedSegments)
    print('length of key list - ', len(keyList))
    print('length of segment - ', segmentLength)
    # for index3 in range(segmentLength):

    for num1 in range(0, len(keyList)):
        tempLength = len(matchedSegments)
        if tempLength >= num1:
            if keyList[num1] in matchedSegments:
                for value in matchedSegments[keyList[num1]]:
                    for num2 in range(num1 + 1, len(keyList)):
                        if value in matchedSegments:
                            del matchedSegments[value]

            else:
                break
    return matchedSegments

```

Image Resampling Detection Module

➤ Generation of the Probability Map and the Periodicity Map

```

% E-step
R = yt - Yt*alpha; % find residuals
P = 1/(sigma*sqrt(2*pi)) * exp(-R.^2/(2*sigma.^2)); % find
probabilities
w = P./(P+p0);

% M-step
sigma = sqrt( sum(w.*R.^2)/sum(w) );
alpha = (Yt'*diag(w)*Yt) \ Yt'*diag(w)*yt;

if(max(abs(prev-alpha))<EPSILON)
    break;
end

```

```

Pmap = 1/(sigma*sqrt(2*pi)) * exp(-(y-Y*alpha).^2/(2*sigma.^2));
Pmap = reshape(Pmap,[height-2*N width-2*N]);

```

➤ Fourier Transforming the Periodicity Map

```

function y = fft2c(x)
y = fftshift(fft2(fftshift(x)));

```

```
function y = fft2c(x)
y = fftshift(dft2(fftshift(x)));
```

- Extracting features from the Periodicity Map and training and testing the Support Vector Machine for classification

```
N = 4;
sum = 0 ;

[ b, ix ] = sort( pmapRow(:), 'descend' );

[ rr, cc ] = ind2sub( size( pmapRow), ix(1:N) );

for ii = 1 : N
    disp( pmapRow( rr(ii), cc(ii) ) )
    sum = sum + (pmapRow( rr(ii), cc(ii)));
end
f1 = sqrt(sum);
disp(sum);
```

```
N2 = 4;
sum2 = 0 ;

[ b, ix ] = sort(abs(fmapRow(:)), 'descend' );

[ rr, cc ] = ind2sub( size( fmapRow), ix(1:N) );

for ii = 1 : N2
    disp( fmapRow( rr(ii), cc(ii) ) )
    sum2 = sum2 + (fmapRow( rr(ii), cc(ii)));
end
f2 = sqrt(sum);
disp(f2);
```

```
for ii = 1 : N
    disp(R( rr(ii), cc(ii) ) )
    f3 = f3 + (R( rr(ii), cc(ii)));
end

f3 = f3/100;
disp(f3);
```

```

for ii = 1 : N
    disp(R( rr(ii), cc(ii) ) )
    f4 = 10000 (R( rr(ii), cc(ii)));
end
f4 = f4/100;
disp(f4);

```

```

load trainData2.csv
load featues.csv
load trainLabelsDataComplete.csv

grp2idx(trainLabelsDataComplete);

x = featues (1:42,:);
y = trainLabelsDataComplete(1:100,:);

rand_num = randperm(42);

X_train = x(rand_num(1:80),:);
y_train = y(rand_num(1:80),:);

X_test = x(rand_num(81:end),:);
y_test = y(rand_num(81:end),:);

c = cvpartition(y_train,'k',5);

opts = statset('display','iter');
fun = @(train_data,train_labels,test_data,test_labels)...
sum(predict(fitcsvm(train_data,train_labels,'KernelFunction','rbf'),test_data) ~= test_labels)
[fs,history] =
sequentialfs(fun,X_train,y_train,'cv',c,'options',opts,'nfeatures',2);

X_train_W_best_features = X_train(:,fs);

```

```

%Mdl = fitcsvm(X_train_W_best_features,y_train);
Mdl =
fitcsvm(X_train_W_best_features,y_train,'Standardize',true,'Kernel
Function','RBF',...
'KernelScale','auto');
CVSVMModel = crossval(Mdl);
classLoss = kfoldLoss(CVSVMModel);

sv = Mdl.SupportVectors;
figure
gscatter(X_train_W_best_features(:,1),X_train_W_best_features(:,2)
,y_train)
hold on
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
legend('1','0','Support Vector')
hold off

X_test_w_best_feature =X_test(:,fs);
accuracy =
sum(predict(Mdl,X_test_w_best_feature)==y_test)/length(y_test)*100
:

```

Image Splicing Detection Module

- **Noise Pattern Analysis**

- **CleanUp Image**

If the image is not in 3 channel uint8 format, convert it into 3 channel image.

```

if numel(size(im))>3
    im=im(:,:,:,1,1,1,1);
end

dots=strfind(filename, '.');
extension=filename(dots(end):end);

if strcmpi(extension, '.gif') && size(im,3)<3
    [im_gif,gif_map] =imread(filename);
    im_gif=im_gif(:,:,:,1);
    im=uint8(ind2rgb(im_gif,gif_map)*255);
end

if size(im,3)<3
    im(:,:,2)=im(:,:,1);
    im(:,:,3)=im(:,:,1);
end

```

- Convert the image into YCbCr model and extract the 'Y' component

```
YCbCr=double(rgb2ycbcr(im));  
  
Y=YCbCr(:,:,1);
```

- Apply Single level discrete 2-D wavelet transform

```
[cA1,cH,cV,cD] = dwt2(Y,'db8');
```

- Tiling sub-band HH1 with non-overlapping blocks

```
cD=cD(1:floor(size(cD,1)/BlockSize)*BlockSize,1:floor(size(cD,2)/B  
lockSize)*BlockSize);  
  
Block=zeros(floor(size(cD,1)/BlockSize),floor(size(cD,2)/BlockSize  
) ,BlockSize.^2);
```

- Fill the blocks with image tiles into a 3 dimensional matrix

```
for ii=1:BlockSize:size(cD,1)-1  
    for jj=1:BlockSize:size(cD,2)-1  
        blockElements=cD(ii:ii+BlockSize-1,jj:jj+BlockSize-1);  
        Block((ii-1)/BlockSize+1,(jj-  
1)/BlockSize+1,:)=reshape(blockElements,[1 1  
numel(blockElements)]);  
    end  
end
```

- Noise Variance Estimation

```
s=median(abs(Block),3)./0.6745;  
  
Map=s;
```


- **Colour Filter Array Interpolation**

- Stating Feature Dimension, Number of blocks to cumulate and the Bayer pattern for the Green colour. The system output results for two feature dimensions. The values have been selected after a series of experiments.

```
% dimensione of statistics
Nb = [2, 8];
% number of cumulated bloks
Ns = 1;
% Pattern of CFA on green channel
bayer = [0, 1;
         1, 0];
```

- **Input Images**

The system is capable to have four inputs at a time. Two pairs of Tampered Images and their respective Authentic Images.

```
for i = 1:2:3

    im_true = imread(filename{i});
    im = imread(filename{i+1});

    for j = 1:2
```

- **Green Channel Extracting**

```
im = image(:,:,2);

[h, w] = size(im);
dim = [h, w];
```

➤ Prediction Error Calculation

The prediction error I further filtered for the denoising purposes.

```
function [pred_error] = prediction(im)

% predictor with a bilinear kernel

Hpred = [ 0,    0.25,    0;
          0.25, -1,    0.25;
          0,    0.25,    0 ];

pred_error = imfilter(double(im),double(Hpred),'replicate');

return
```

➤ Local Variance for both Acquired and Interpolated pixels

```
var_map = getVarianceMap(pred_error, Bayer, dim);
```

```
function [var_map] = getVarianceMap(im,Bayer,dim)
% extend pattern over all image
pattern = kron(ones(dim(1)/2,dim(2)/2), Bayer);
% separate acquired and interpolate pixels for a 7x7 window
mask = [1, 0, 1, 0, 1, 0, 1;
        0, 1, 0, 1, 0, 1, 0;
        1, 0, 1, 0, 1, 0, 1;
        0, 1, 0, 1, 0, 1, 0;
        1, 0, 1, 0, 1, 0, 1;
        0, 1, 0, 1, 0, 1, 0;
        1, 0, 1, 0, 1, 0, 1];

% gaussian window for mean and variance
window = gaussian_window().*mask;
mc = sum(sum(window));
vc = 1 - (sum(sum((window.^2))));
window_mean = window./mc;
% local variance of acquired pixels
acquired = im.*(pattern);
mean_map_acquired =
imfilter(acquired,window_mean,'replicate').*pattern;
sqmean_map_acquired =
imfilter(acquired.^2,window_mean,'replicate').*pattern;
var_map_acquired = (sqmean_map_acquired -
(mean_map_acquired.^2))/vc;
% local variance of interpolated pixels
```

```

interpolated = im.*(1-pattern);
mean_map_interpolated =
imfilter(interpolated>window_mean,'replicate').*(1-pattern);
sqmean_map_interpolated =
imfilter(interpolated.^2>window_mean,'replicate').*(1-pattern);
var_map_interpolated = (sqmean_map_interpolated -
(mean_map_interpolated.^2))/vc;

var_map = var_map_acquired + var_map_interpolated;
return

```

➤ Proposed Feature Calculation

```
stat = getFeature(var_map, Bayer, Nb);
```

```

function [statistics] = getFeature(map,Bayer,Nb)

% Proposed feature to localize CFA artifacts

pattern = kron(ones(Nb/2,Nb/2),Bayer);

func = @(sigma)
(prod(sigma(logical(pattern)))/(prod(sigma(not(logical(pattern))))
));

statistics = blkproc(map,[Nb Nb],func);

statistics(isnan(statistics))=1;
statistics(isinf(statistics))=0;

return

```

➤ GMM Parameter Estimation

```
[mu, sigma] = MoGEstimationZM(stat);
```

```

function [mu,sigma,mix_perc] = MoGEstimationZM (statistics)

% Expectation Maximization Algorithm with Zero-Mean forced first
component
% E/M algorithm parameters initialization

tol = 1e-3;
max_iter = 500;

```

```

% NaN and Inf management
statistics(isnan(statistics)) = 1;
statistics(statistics<0)=0;
data = log(statistics(:));
data = data(not(isinf(data)|isnan(data)));

% E/M algorithm
[alpha, v1, mu2, v2] = EMGaussianZM(data, tol , max_iter);
xnew = z;

%error if value is Inf

if ~isfinite(xnew)

    display(['Number of iterations: ' num2str(niter),'alpha is:' num2str(alpha)])

    error('x is inf or NaN')

end

x = xnew;

[M,N] = size(xblock);

xblockinter = reshape(xnew,M-2,N-2);

xblock(2:end-1,2:end-1)=xblockinter;

xblock(1,2:end-1)=xblock(2,2:end-1);

xblock(end,2:end-1)=xblock(end,2:end-1);

xblock(:,1)=xblock(:,2);

return;

```

➤ Likelihood Map Generation

```

loglikelihood_map = loglikelihood(stat, mu, sigma);

```

```

function [L] = loglikelihood(statistics, mu, sigma)

% Loglikelihood map

% allowable values for logarithm
min = 1e-320;
max = 1e304;

statistics(isnan(statistics))=1;
statistics(isinf(statistics))=max;
statistics(statistics == 0) = min;
statistics(statistics<0)=0;

mu1=mu(2);
mu2=mu(1);

sigma1=sigma(2);
sigma2=sigma(1);

% log-likelihood
L = log(sigma1) - log(sigma2) ...
    -0.5.*(((log(statistics) - mu2).^2)/sigma2^2) - ...
    (((log(statistics) - mu1).^2)/sigma1^2));

return

```

➤ Filtered and Cumulated Likelihood map generation

```
map = getMap(loglikelihood_map, Ns, Nm);
```

```

function [map] = getMap(L, Ns, Nm)

% Cumulated and median filtered log-likelihood map

% cumulate blocks
func = @(x) sum(x(:));

log_L_cum = blkproc(L, [Ns Ns], func);

% median filtered log-likelihood
map = medfilt2(log_L_cum, [Nm Nm], 'symmetric');

return

```

➤ Plot the Results

```
% plot result
figure
subplot(2,2,1), imshow(im_true), title('Not tampered
image');
subplot(2,2,2), imshow(im), title('Manipulated
image');
subplot(2,2,3), imagesc(map), colormap('gray'),axis
equal, axis([1 w 1 h]), title(['Probability map (Nb =
',num2str(Nb(j)),')']);
subplot(2,2,4), hist(data, n_bins), title('Histogram
of the proposed feature');

display('Press any key to continue...')
```

Image Retouching using Filters Detection Module

➤ Calculating the denoised image.

```
function
[denoisedimage,noise,error]=noise_finalone(image,quality,weight,maxiter)

if ~exist('weight','var')

    weight = 1;

end

if ~exist('maxiter','var')

    maxiter = 50;

end

%patchsize (DCT block size)

step = 8;

%Get the image

readcompressimage=double(image);

%initialize gamma as in paper

gamma = 1;
```

```

%Block size as givenby patchsize

blockX = step;blockY = step;

%Quantization table

Q50=[ 16 11 10 16 24 40 51 61;

      12 12 14 19 26 58 60 55;

      14 13 16 24 40 57 69 56;

      14 17 22 29 51 87 80 62;

      18 22 37 56 68 109 103 77;

      24 35 55 64 81 104 113 92;

      49 64 78 87 103 121 120 101;

      72 92 95 98 112 100 103 99];

```

```

%Linear operation as done in JPEG compression

if quality > 50

    Q = round(Q50.*(ones(8)*((100-quality)/50)));

elseif quality < 50

    Q = round(Q50.*(ones(8)*(50/quality)));

elseif quality == 50

    Q = Q50;

end

Q=double(Q);

```

```

% if step==16

%   Q = repmat(Q,2,2);

% Create DCT Matrix

H = dct(eye(step));

% kronecker prod of DCT matrix

G =kron(H,H);

% getting noise covariance matrix

q=reshape(Q,[1 64]);

%determining variance as in the paper

varfreq = ((gamma.*q).^2)/12;

B = repmat(varfreq,[1 blockX*blockY/64]);

Key = double(diag(B));

%The noise covariance matrix

Kez = G'*Key*G;

kezinv=inv(Kez);

```

```

% Block by block denoising

for i=1:step:size(readcompressimage,1)

    for j=1:step:size(readcompressimage,2)

        y = readcompressimage(i:i+step-1,j:j+step-1);

    end

```



```

%      zq=reshape(y,[step*step 1]);

      xblock = padarray(y,[1 1],'replicate');

      denoisedimage(i:i+step-1,j:j+step-1)=
grad_finalone(y(:),xblock,kezinv,weight,maxiter);

%      k=k+1;

      end

%Extract noise from denoised image

noise      =      double(denoisedimage)      -
double(readcompressimage(1:size(denoisedimage,1),1:size(denoisedimage,2)));

noise = reshape(noise,[1 size(noise,1)*size(noise,2)]);

%truncating denoised image

denoisedimage(denoisedimage<0)=0;denoisedimage(denoisedimage>255)=255;

```

➤ Extracting the features

```
function output = extract_TPM(input,choice,range)

% Difference array calculation

% Just two or four directions for certain applications might be optimal

dir{1} = input(2:end,:) - input(1:end-1,:); % Vertical bottom to top direction

dir{2} = input(1:end-1,:) - input(2:end,:); % Vertical top to bottom

dir{3} = input(:,2:end) - input(:,1:end-1); % Horizontal right to left

dir{4} = input(:,1:end-1) - input(:,2:end); % Horizontal left to right

dir{5} = input(2:end,2:end) - input(1:end-1,1:end-1); %Major diagonal bottom to
top

dir{6} = input(1:end-1,2:end) - input(2:end,1:end-1); %Minor diagonal top to
bottom

dir{7} = input(2:end,1:end-1) - input(1:end-1,2:end); %Minor diagonal bottom to
top

dir{8} = input(1:end-1,1:end-1) - input(2:end,2:end); %Major diagonal top to
bottom
```

```

if strcmp(choice,'sign')

    val = 9;

elseif strcmp(choice,'truncate')

    val = (2*range+1)^2;

end

%TPM calculation

TPM = cell(1,8)

for i = 1 : 8

    D = dir{i};

    if strcmp(choice,'sign')

        D=sign(D);

        D=D+2;

    elseif strcmp(choice,'truncate')

        D=round(D);

        D(D>range)=range;D(D<-range)=-range;

        D=D+range+1;

    end

    %TPM calculates along the same direction that the gradient is

    %calculated (Can be modified if needed)

```

```

function TPM = transprobestimate(input,choice,range,direction)

%function output = transprobestimate(input,choice,range,direction)

if strcmp(choice,'sign')

    t=zeros(3,3);

elseif strcmp(choice,'truncate')

    t=zeros(range*2+1,range*2+1);

end

switch direction

    case 1

        % Vertical bottom to top direction

        t1=sparse(input(2:end,1:end),input(1:end-1,1:end),1);

    case 2

        % Vertical top to bottom direction

        t1=sparse(input(1:end-1,1:end),input(2:end,1:end),1);

    case 3

        %Horizontal right to left direction

        t1=sparse(input(1:end,2:end),input(1:end,1:end-1),1);

    case 4

        %Horizontal left to right direction

        t1=sparse(input(1:end,1:end-1),input(1:end,2:end),1);

    -

```

```

case 6

    %Minor diagonal top to bottom direction

    t1=sparse(input(1:end-1,2:end),input(2:end,1:end-1),1);

case 7

    %Minor diagonal bottom to top direction

    t1=sparse(input(2:end,1:end-1),input(1:end-1,2:end),1);

case 8

    %Major diagonal top to bottom direction

    t1=sparse(input(1:end-1,1:end-1),input(2:end,2:end),1);

otherwise

    error('Wrong Argument direction');

end

t1=full(t1);

t(1:size(t1,1),1:size(t1,2))=t1;

for i=1:size(t,1)

    e(i,:)= t(i,:) ./ sum(t(i,:));

end

for i=1:size(e,1)

    for j=1:size(e,2)

        if (isnan(e(i,j)))

            e(i,j)=0;

        end

    end

end

```

➤ Gradient calculation

```
% gradient calculation

function [xopt,err] = grad_finalone(zq,xblock,kezinv,weight,maxiter)

zq=double(zq);

z = zq;

% termination tolerance

tol = 1e-3;

% Initial step size for gradient descent

alpha = 0.05;

%Initial data vector

x = z;

%Initializing error vector

err=zeros(1,maxiter);

niter=1;

while (niter <= maxiter)

    % calculate gradient:

    g = grad(xblock,z,zq,kezinv,weight);

    temp = z;

% Gradient descent step

    z = z - alpha*g;

    %calculate error for tolerance measure

    err(1,niter) = mean(abs((abs(z) - abs(temp))));
```

```

xnew = z;

%error if value is Inf

if ~isfinite(xnew)

    display(['Number of iterations: ' num2str(niter),'alpha is:' num2str(alpha)])

    error('x is inf or NaN')

end

x = xnew;

%Update xblock with new values to calculate gradient again

[M,N] = size(xblock);

xblockinter = reshape(xnew,M-2,N-2);

xblock(2:end-1,2:end-1)=xblockinter;

xblock(1,2:end-1)=xblock(2,2:end-1);

xblock(end,2:end-1)=xblock(end,2:end-1);

xblock(:,1)=xblock(:,2);

xblock(:,end)=xblock(:,end-1);

%come out of the loop if tol level reached

if err(1,niter)<=tol

    break;

```

```

% disp(niter-1);

%Final optimized/denoised block

[M,N] = size(xblock);

xopt = reshape(x,M-2,N-2);

end

function g = grad(xblock,z,zq,kezin,weight)

% Output -

% g - gradient calculated based on HMRF model and Quantization noise model.

%Lambda governs the degree of smoothing

lambda = 0.08;

%Threshold to define huber function as set in the paper

T = 10;

%Initializations

[M,N] = size(xblock);

xblock = double(xblock);

pho=zeros(3,3);%r=0;

phoo = zeros(5,5);

r=xblock(M-2,N-2);

%HMRF calculation

```



```

for i = 1:3

    for j=1:3

        u = currentpx - xblock(l+i-2,k+j-2);

        if(abs(u)<=T)

            if(l-1==1 || k-1==1 || l-1==8 || k-1==8)

                pho(i,j) = weight*(2*u);

            else

                pho(i,j) = 2*u;

            end

        elseif((u)>T)

            if(l-1==1 || k-1==1 || l-1==8 || k-1==8)

                pho(i,j) = weight*2*T;

            else

                pho(i,j) = 2*T;

            end

        end

    end

end

```

```

elseif((u)< -T)

    if(l-1==1 || k-1==1 || l-1==8 || k-1==8)

        pho(i,j) = -weight*2*T;

    else

        pho(i,j) = -2*T;

    end

end

end

end

```

```

phoo(2:4,2:4) = pho;

for i = 2:size(phoo,1)-1

    for j=2:size(phoo,2)-1

        pho (i-1,j-1) = sum(sum(phoo(i,j) - phoo(i-1:i+1,j-1:j+1)));

    end

end

r(l-1,k-1) = sum(sum(pho));

end

end

% 'r' term from HMRF

rr=reshape(r,[size(r,1)*size(r,2) 1]);

```

```

diffZ = double(z - zq);

% 's' term from quantization noise model

s = kezinv*diffZ;

%gradient as computed in the paper

```

➤ Code snippet for SVM classifier

```

load('C:\Auxillium\Training and Testing Data\features.csv');%mention the training
dataset location
load('C:\Auxillium\Training and Testing Data\labels.csv');%mention the training
dataset labels location

X = features(1:end,:);
y = labels(1:end,:);

%dividing dataset into training and testing

X = features(1:end,:);
y = labels(1:end,:);

rand_num = randperm(6000);

%training Set
X_train = X(rand_num(1:5000),:);
y_train = y(rand_num(1:5000),:);

%testing Set
X_test = X(rand_num(5000:end),:);
y_test = y(rand_num(5000:end),:);

%preparing validation set out of training set
c = cvpartition(y_train,'k',10);

SVMModel = fitcecoc(X_train,y_train);

```

