Report on the paper

FORENSIC DETECTION OF MEDIAN FILTERING IN DIGITAL IMAGES

Anirban Haldar

[anirbanh@iitbhilai.ac.in](mailto:anirbanh@iitbhilai.ac.in)

Atharva Inamdar

[atharvai@iitbhilai.ac.in](mailto:atharvai@iitbhilai.ac.in)

Karande Jaysing Vitthalrao

[karandej@iitbhilai.ac.in](mailto:karandej@iitbhilai.ac.in)

December 3, 2021

## Summary of the paper

In today’s world, due to advancement in technology and cheap hardware cost, clicking & editing photo is very common for anyone with a smartphone. Now, we have started taking photogra- phy as a proof of some incident or fact which sometimes lead to malpractices. There are many free and paid image manipulation software available to make content-preserving & content changing-manipulations. Content preserving manipulations are generally applied to conceal the trail of previous manipulations. So detection of such manipulations is highly important. Here we propose a blind forensic algorithm to detect one such content preserving manipulation which is known as median filtering (MF). Many existing anti forensic techniques depend on the assumption that there is a linear correlation between neighboring pixels which is actually destroyed by the median filter since it is a non-linear operator. So blind detection of MF be- comes highly significant in the present times. The probability of zero values on the first order difference map in texture regions is used to in detection of median filtering. Experiments have been performed to prove the high efficiency of our algorithm in blind detection of Median Filtering.

## Introduction

In today’s day and age authenticity of a digital image has to be taken very seriously due to the easy availability of powerful media editing software. So Image forensic has become an art which has to be mastered in order to avoid any disastrous consequences caused by the image tampering.

Image tampering or image manipulation can be classified into two types:

1. **Content preserving**: Only change the perceptual quality of an image
2. **Content changing**: Changes the semantic content of the entire image

Although the content-preserving manipulations merely change image perceptual quality but not semantic content, such manipulations are usually applied to conceal visual trail of tampering operations to create realistic forgeries. These manipulations destroy the forensically significant fingerprints, which are left by previous tampering operations. So the existing forensic algo- rithms are bound to fail in such situations. Highly skilled forgery makers use MF to retouch the and hide the crude tampering cases. One such example is usage of MF to hide the interpo- lation traces in a new resampling algorithm which actually fools the state-of-the-art Popescue and Farid’s resampling detector. Such examples state the importance of MF detection in the image forensics field.

There are some methods which involve usage of streaking artifacts and subtractive pixel adjacency matrix (SPAM) features to detect MF in bitmap and JPEG post-compressed images, respectively. Our algorithm uses a different metric altogether which will be explained in further sections.

## Problem Definition

Many of the MF detection algorithms present actually depend on an assumption that there is some linear correlation between the neighboring pixels. But median filter is a non-linear operator which actually destroys any such linear correlation which is present in an image. Below is a simple image showing how median filter destroys the linear correlation.

|  |  |  |
| --- | --- | --- |
| Sequences | Mean | Median |
| X*n* = 1 1 3 | 5/3 | 1 |
| Y*n* = 1 2 0 | 1 | 1 |
| X*n* + Y*n* = 2 3 3 | 8/3 | 3 |

Table 1: Non-Linearity of Median Filter

So if we consider X and Y to be neighboring signals then as we can see that after applying mean filter the linear correlation between them which is simple X+Y is preserved since mean is a linear operator. But in case of median this correlation is simply destroyed (1 + 1 != 3) This is the reason why the existing algorithms such as resampling and CFA interpolation detection which rely on the assumption of linear correlation between neighboring pixels are bound to fail. So there is a need to devise a MF detection forensic algorithm which doesn’t involve this linearity assumption.

## Explanation of Math Behind

In this section, we will discuss basic mathematics of Median Filter. Also, we will carry out an analytic investigation of MF fingerprint properties from the signal statistic standpoint.

### Median

Median is the value separating the higher half from the lower half of a data sample. Median is used to show robustness in various domains. Median depicts smoothness, robustness and lesser deviation in the data.

To find the median value in a list with an odd amount of numbers, one would find the number that is in the middle with an equal amount of numbers on either side of the median. To

find the median, first arrange the numbers in order, usually from lowest to highest. For example, in a data set of {3, 13, 2, 34, 11, 26, 47}, the sorted order becomes {2, 3, 11, 13, 26, 34, 47 }. The

median is the number in the middle {2, 3, 11, 13, 26, 34, 47}, which in this instance is 13 since there are three numbers on either side.

To find the median value in a list with an even amount of numbers, one must determine the middle pair, add them, and divide by two. Again, arrange the numbers in order from lowest to highest. For example, in a data set of {3, 13, 2, 34, 11, 17, 27, 47}, the sorted order becomes {2, 3,

11, 13, 17, 27, 34, 47}. The median is the average of the two numbers in the middle {2, 3, 11, 13, 17,

26, 34, 47}, which in this case is fifteen {(13 + 17) ÷ 2 = 15}.

### Median Filter

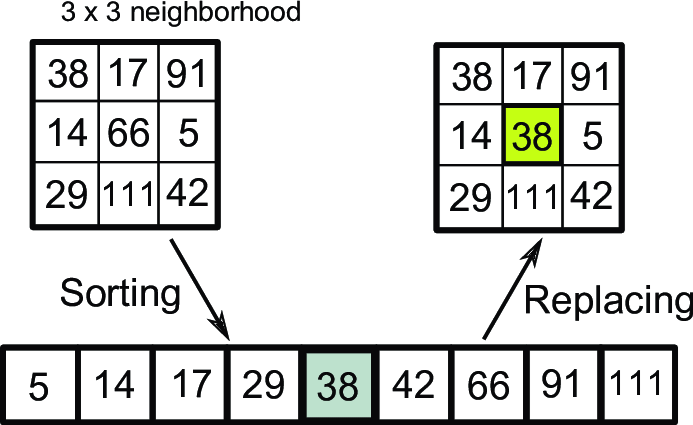
The median filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. Such noise detection is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image). Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise, also having applications in signal processing. With the help of median filter one can easily modify image. According to the median filter, the center pixel of M×M neighborhood is replaced by the median value of the corresponding window. For a median filter window dxd, sort all pixel values in a dxd window around (i, j)th pixel and pick the median element (Figure 1). This will be the new pixel value at (i, j)th position.

*I’(i, j) = median[I(x, y)]*

where, x ∈ [(i – d//2), (i + d//2)]

y ∈ [(j – d//2), (j + d//2)] d = window size

‘//’ is integer division operator



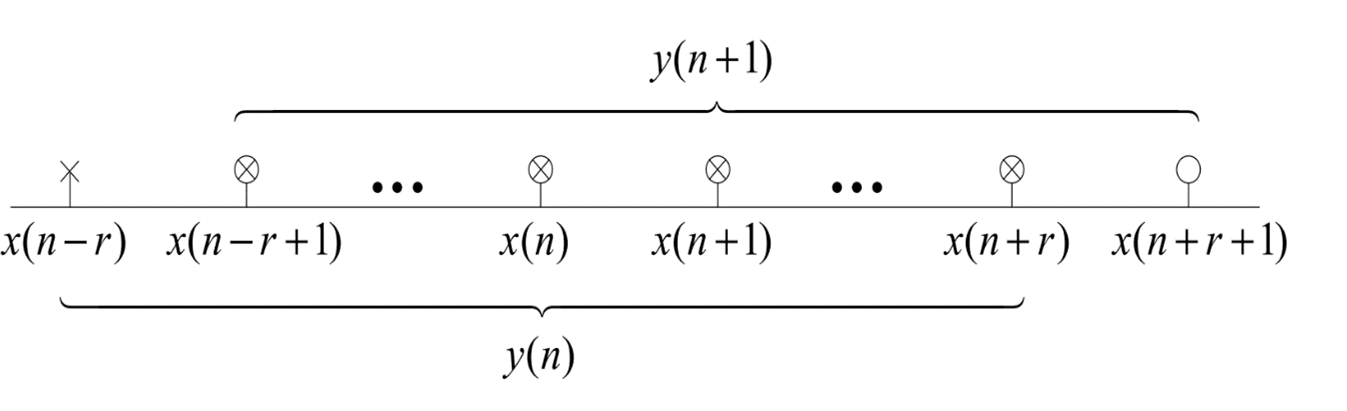
*Figure 1: median filtering in image processing*

Similarly, we will raster scan the whole image and finally we will get the median filtered image.

### Analysis of Median-Filtered Signal

In this section, we carry out an analytic investigation of MF fingerprint properties from the signal statistic standpoint. Such fingerprint metric will become the identity card to distinguish MF from un-MF and other operations. It is easy to understand that probability will be high for the intensity values of two adjacent pixels of a median filtered image to be same or nearby with respect to that of original image. For the analysis we will consider only

textured regions in the image for analysis. The reason for excluding untextured pixels is that the definite statistical discrepancy can’t be assured in smooth regions, especially those among different images.



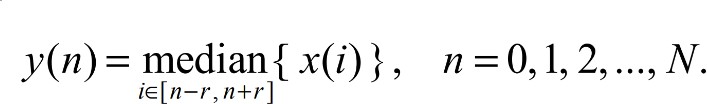
*Figure 2: The original signal x (n) is median-filtered to be y (n)*

Let’s discuss median filtering on 1-D digital signal as illustrated in figure 2. Here, x is the original signal,

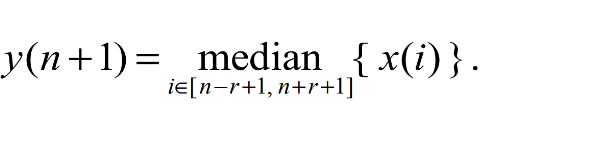


where the integer x (n) ∈ [0, 255]

Suppose the median-filtered signal is y, which can be written as,



Here, width of the filter window is w = 2r + 1, r = 1, 2... Boundary is processed implicitly and the output is supposed to have the same size as the input array. Correspondingly, we have



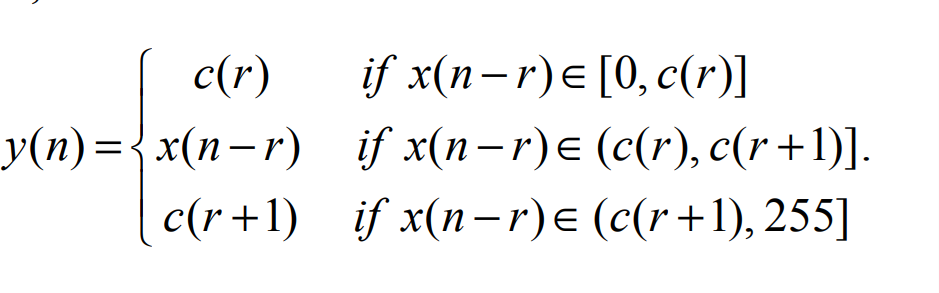
From the formula (2) and formula (3), we can see that there exist common elements for computing the median values y(n) and y(n+1). They are



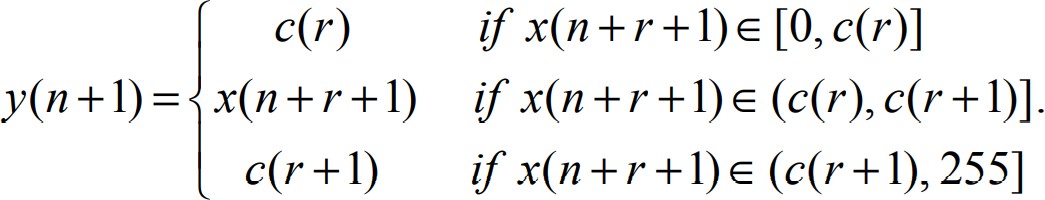
It can be concluded that a certain distinct order must exist among the elements of the set C. The sorted sequence can be rewritten as,



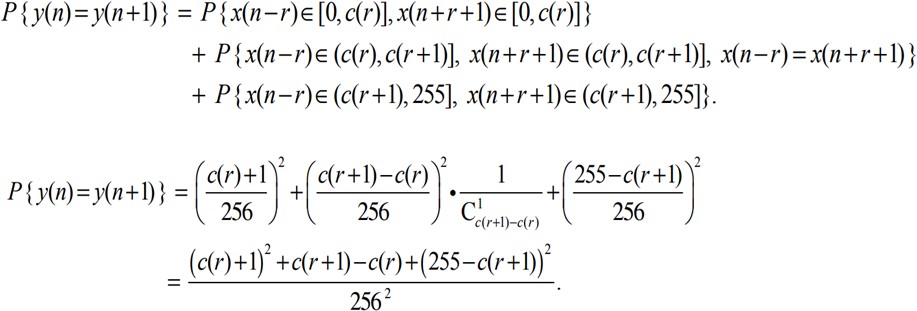
Then the filtered value y(n) can be determined in terms of the relationship between x(n-r) and the elements within C0. That is,



Similarly, y(n+1) is related with x(n+r+1) as follows,

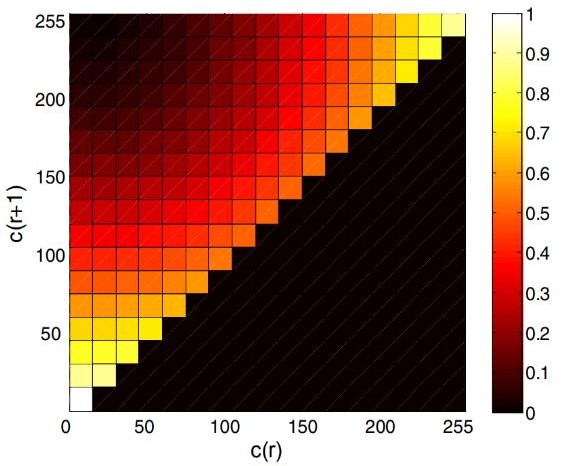


Because the space between x(n-r) and x(n+r+1) is as long as 2r+1, the correlation between such two variables is so weak that they can be assumed to be independent. Then probability of y(n) equaling to y(n+1) can be deduced as the formula



On the assumption that both x(n-r) and x(n+r+1) confirm the uniform distribution U [0, 255], the probability formula (8) can be simplified as that in formula (9). From this formula, we can see that the probability P{y(n)=y(n+1)} is just a function of the two median elements, namely c(r) and c(r+1).

### Probability Between Common Elements using Heat-Map.



*Fig 3: Probability map for P{y(n)=y{n+1)}*

Heat map is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by intensity, giving obvious visual

cues to the reader about how the phenomenon is clustered or varies over space. Many different color schemes can be used to illustrate the heat map, with perceptual advantages and disadvantages for each. Rainbow color maps are often used, as human can perceive more shades of color than they can of gray, and this would purportedly increase the amount of detail perceivable in the image.

Visualized probability map is indicated in Fig.3. In the above heat map, we are placing c(r) on horizontal and c(r+1) on the vertical axis. The intensity of the color show probability distribution. To produce the heat map, we are mapping the color coded probability distribution over the common elements. From the map, it can be observed that higher probability occurs if c(r) and c(r+1) get closer. It can be verified by looking at equation (9). For the above heat map, part below the diagonal will not be possible to draw. Because we have condition c(r+1) greater than c(r) that is common elements are sorted in ascending order. Hence by default smaller one will get name as ‘c(r)’ and larger one as ‘c(r+1)’.

## Explanation of the main Detection Algorithm

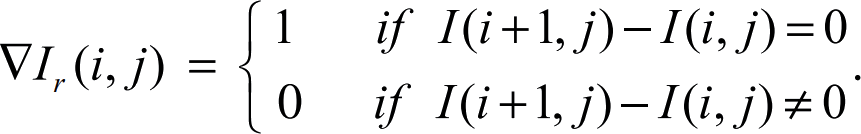
The main algorithm is based on one point i.e. in median filtered image, there will be similarity in neighboring pixels in textured region. Or we can say In texture regions, the fluctuation of pixel gray-levels is more severe than that in smooth areas. The correlation between neighboring pixels is also comparatively weak if it is original image.

Here our algorithm is based on grayscale images. So before applying this algorithm on any image, we first have to convert it in to grayscale version.

There are only few steps by which we can calculate the f value. These are

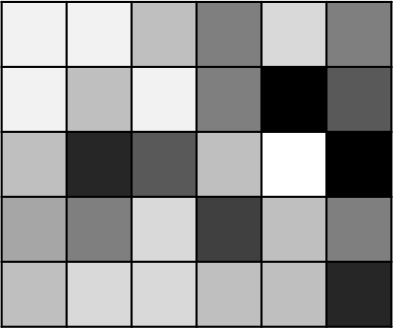
**Step 1 :**

Detect relation between neighboring pixels as



In this way, we are basically finding whether two adjacent pixel values are same or not.

# 0



1

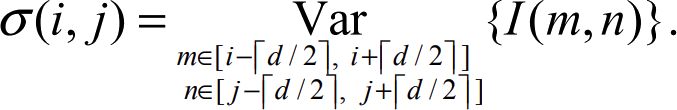
= Adjacent pixels are **same**

= Adjacent pixels are **different**

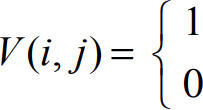
So ultimately, we are getting a binary matrix for row difference. Similarly, we also have to calculate the same thing to get the column binary matrix.

**Step 2 :**

Detect whether a region is textured or not. To do this, we have taken a 7x7 window, and we are calculating the variance in that window. If the variance is higher than a threshold value, then we can consider that region as a textured region. So, we can do here as.

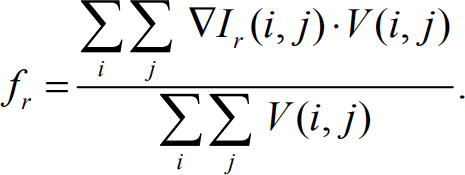


Now to generate the binarized version of it, we have to compare each term with a value tau ( = 100) as

**Step 3:**

# If σ >= τ = Textured Region If σ < τ = Smooth Region

Finally, we will calculate two f values, one for row using the row difference map and variance map, and another with the column difference map and variance map. As



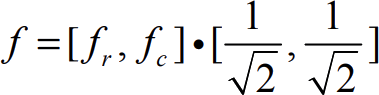
Here, what we are doing is, we are considering only those pixels having Same intensity, but it falls under textured region.

As

|  |  |  |
| --- | --- | --- |
| ∇ ***Ir(i, j)*** | ***V(i, j)*** | ∇ ***Ir(i, j) \* V(i, j)*** |
| 0 (Diff. intensity) | 0 (Smooth) | 0 (Ignore) |
| 0 (Diff. intensity) | 1 (Textured) | 0 (Ignore) |
| 1 (Same intensity) | 0 (Smooth) | 0 (Ignore) |
| 1 (Same intensity) | 1 (Textured) | 1 (Consider) |

**tep 4:**

At the end, we are merging two values i.e. fc and fr ro get the final f value as



**Step 5:** End

1. **Analysis of the Results & Performance**

### Dataset Explained

To evaluate the performance of the MF detection algorithm, the popular image dataset UCID is introduced for test. The UCID dataset consists of 1338 uncompressed TIFF images on a

variety of topics including natural scenes and man-made objects, both indoors and outdoors. Note that all images are color images and we convert them to grayscale in the standard manner.

### 6.3 Distribution of Feature Matric f.

For getting the f matric the parameters used for determining textured pixels are set:d = 7 , τ

= 100.

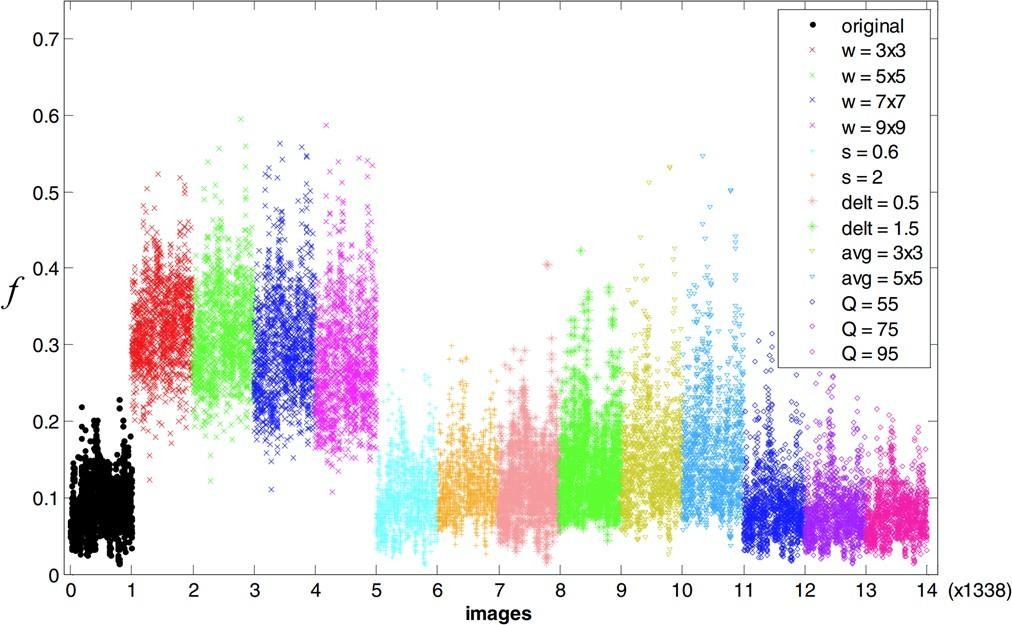


Figure 1: Distribution of the feature metric.

Distribution of the feature metric f extracted from different types of sample images: origi- nal, median filtered (window size w=3x3, 5x5, 7x7, 9x9), bilinear scaled (scaling factor s=0.6, 2), Gaussian filtered (delt=0.5, 1.5), average filtered (window size avg=3x3, 5x5), JPEG com- pressed (Q=55, 75, 95) is shown above.

From the above figure we can intuitively say that the values of feature metric f for median filtered images are higher than the original images and original images with manipulations other than median filtering.

### Performance of Algorithm

#### Baseline performance

The basic capability of MF detection can be divided into two cases:

#### Without the disturbance of other operations

In this case, untouched original images are taken as negative (N) and their median- filtered versions are taken as positive samples (P).

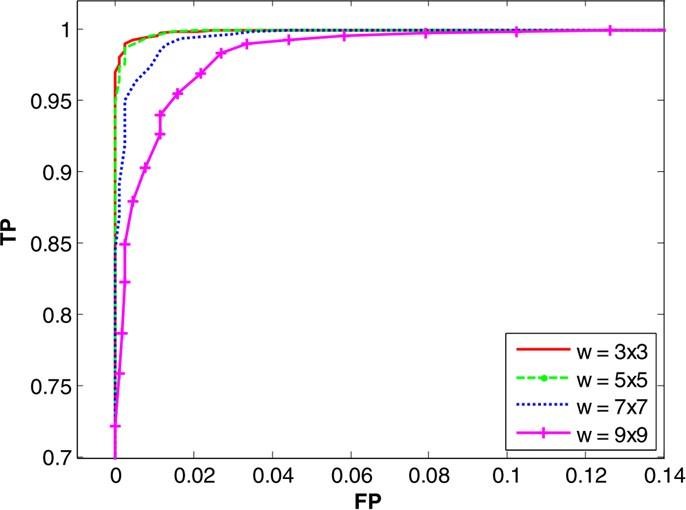


Figure 2: ROC curve for the classification between orig- inal images and their median-filtered versions.

As we can see from the ROC curve above perfect performance has been achieved, especially in the case of small window sizes, i.e. w=3x3 where probability of getting true positive can be extended up to 0.98 while keeping the false positive at 0.

#### With other operations occurred before MF

In this case, classification is performed between previously manipulated images (N) and their 5x5 median filtered versions (P).

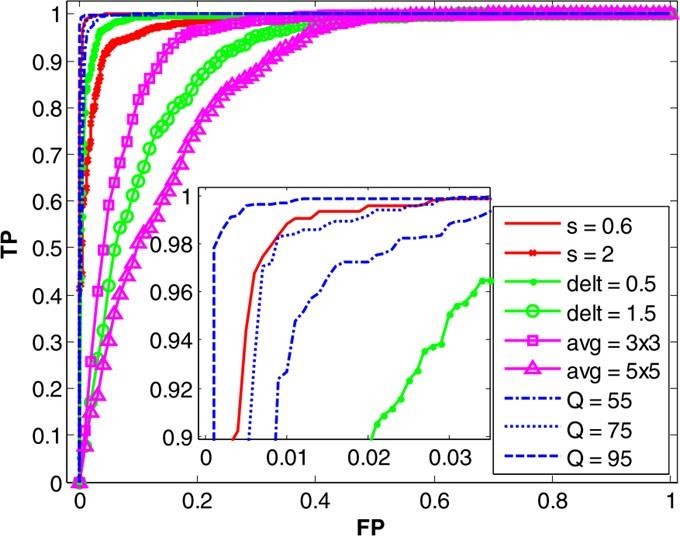


Figure 3: ROC curve for the classification between median-filtered (5x5) images and the images processed by other manipulations

Here we basically want to see if our algorithm can detect MF on the previously ma- nipulated images. We can see from the ROC curve that MF is harder to detect on the heavily Gaussian filtered (delt = 1.5) and average filtered( avg = 5x5) images than that on other manipulated images. However the results are not too bad as we can obtain a true positive probability TP = 0.85 with the false probability FP = 0.2 in case of Gaussian blurred (delt = 1.5).

It can also be seen that there is no problem in detecting the median filtering on the resampled images irrespective of the sampling rate s. This proves that our forensic

technique is successful against the anti-forensic resampling algorithm that uses MF on interpolated images.

#### Differentiating performance:

This type of test is carried out to check whether our algorithm is able to differentiate between the MF and other manipulations. In this test, manipulated images are taken as negative samples (N) and 5x5 median-filtered images are taken as positive samples (P).

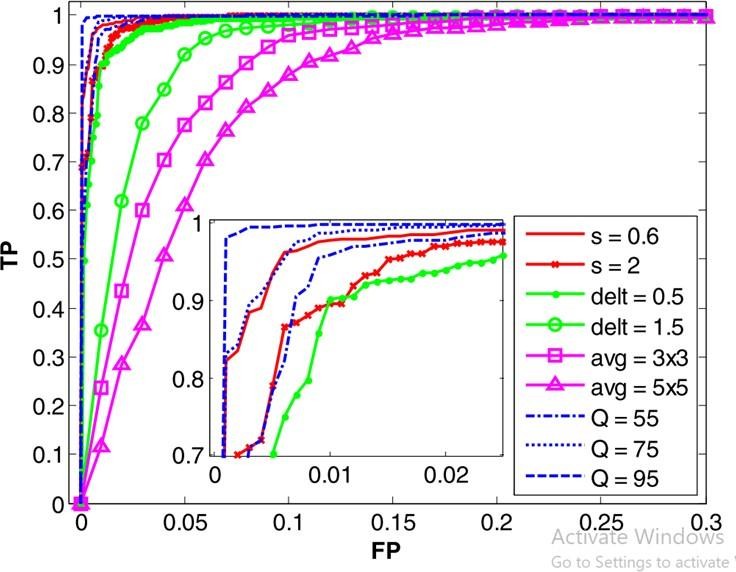


Figure 4: ROC curve for the classification between the manipulated images and their median-filtered (5x5) ver- sions.

From the curve it can be seen that distinguishing MF from Gaussian and average filtering are more difficult than that from other manipulations. But the results are still satisfactory as we can see that even for the worst case, average filtering (avg= 5x5), TP value can arrive at 0.87 while keeping the false positive probability at FP = 0.1. In the other cases, TP always keeps higher than 0.95.

#### Robustness:

This type of test is performed to check if our algorithm can detect that if any post pro- cessing has occurred after the median filtering operation. In this test the median-filtered (5x5) sample images’ different post-manipulated versions are taken as positive samples

(P) and the manipulated original images are taken as negative samples (N).

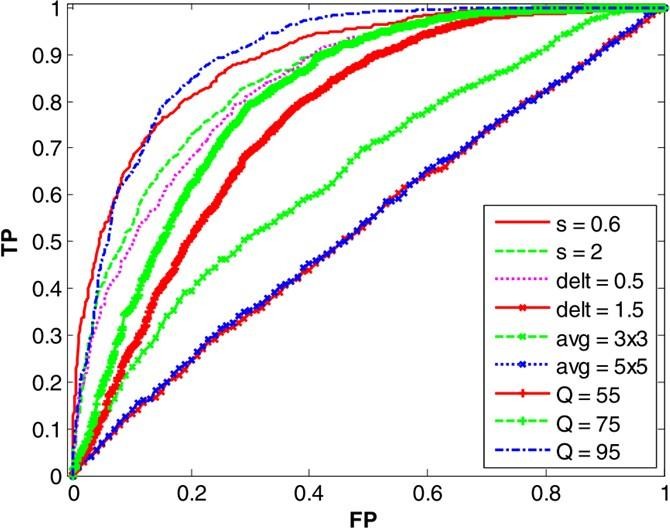


Figure 5: ROC curve for the classification between the manipulated images and the post-manipulated median- filtered (5x5) images.

Test results show that the MF detection scheme can resist the post operations to some extent, except the heavily Gaussian filtering and average filtering operations. **The reason for poor performance can be the interference in local pixel configuration due to these post processing operations.** It can be said that weaker the post-operation strength is better I the detection accuracy. It can be admitted that this forensic algorithm is fragile when it comes to the detection of post processing on the median filtered images but should also be noted that such post-processing is rarely applied after median filtering in practical applications.

## Conclusion

1. An effective forensic algorithm is presented for blind detection of Median Filtering.
2. Statistical characteristics of the median-filtered signal is analyzed and measured by the probability of zero value on the difference map of textured pixels.
3. The median-filtered image is identified by performing thresholding adjudication on the fingerprint metric.
4. Differentiation capability and robustness of the algorithm is also discussed.

## Implementation of Code

We get a MATLAB code along with this paper, which is actually the same code written by the author himself. But as all of us are not familier wih the matlab environment and all the functions used in the code, we decided to write the code from the scratech using the understanding we get by reading the paper.

We have implemented the code in python with the help of following libraries (other than the default provided by the Python itself).

* Numpy (np)
* Pullow (PIL).
* Matplotlib

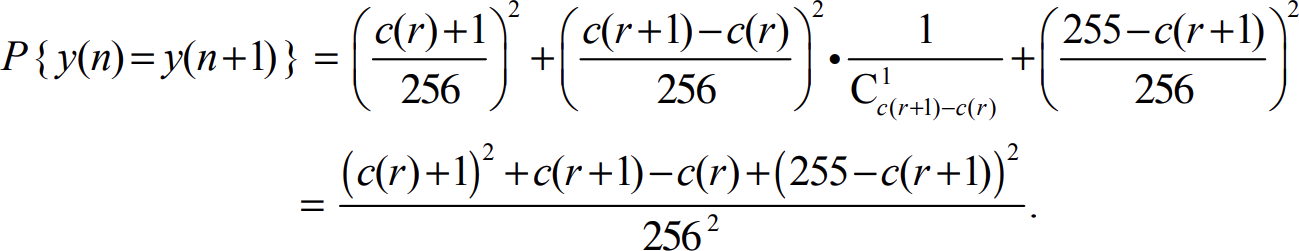
Our version of the code is present in the same folder of this report.

## Recreation of Figures

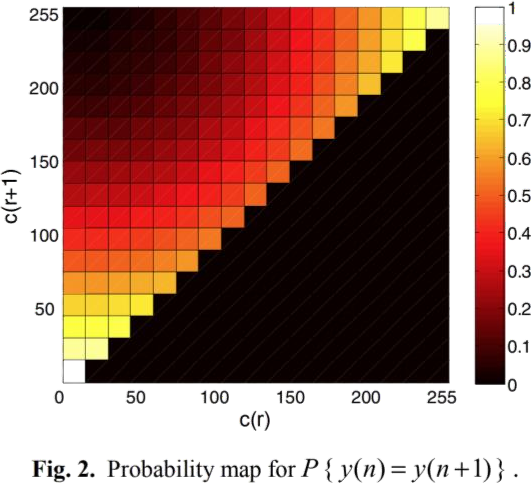
As we have mentioned in the presentation, we are going to recreate 4 diagrams from the paper, these are as follows

### Heatmap:

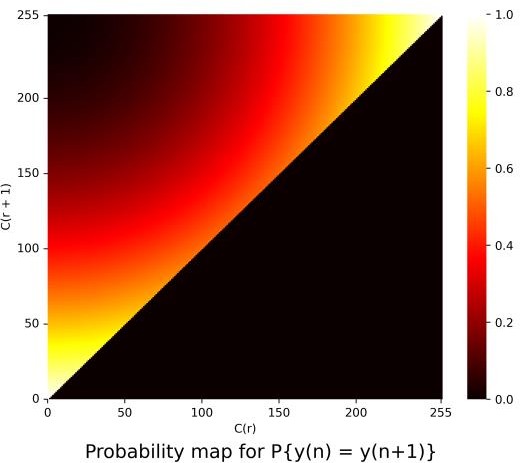
Here, our target is to recreate the heatmap produced using the probability function



Now the actual author’s version of the diagram is,

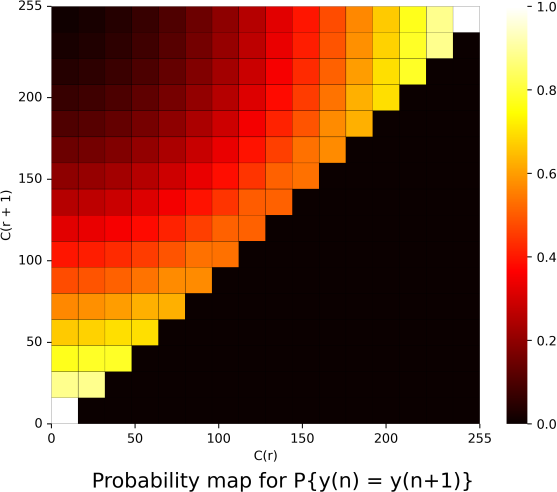


To recreate this, we have solved the following equation and our result is



We can see our output is very smooth, but it is not exact copy of the author’s work, so we

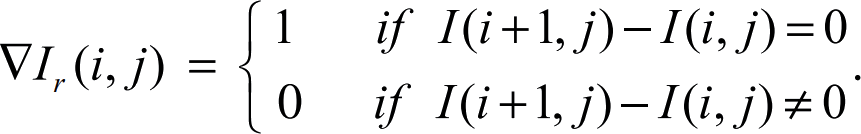
have done MaxPooling on 16x16 -> 1 and the next output is



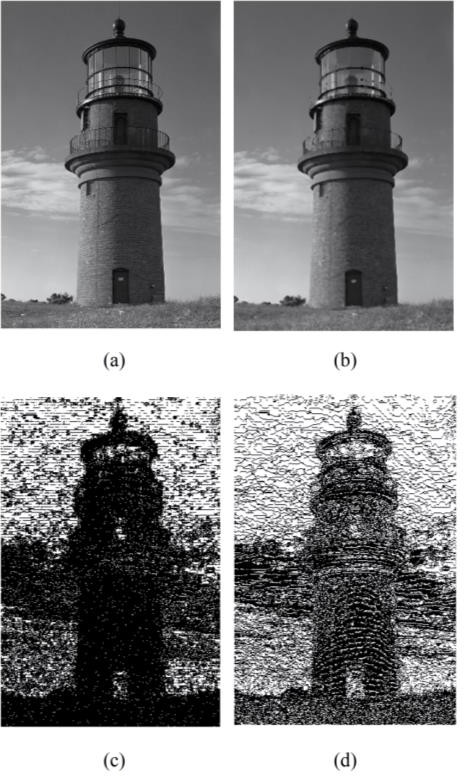
Now, we can say our result is almost exact as the author’s work.

### Light-House

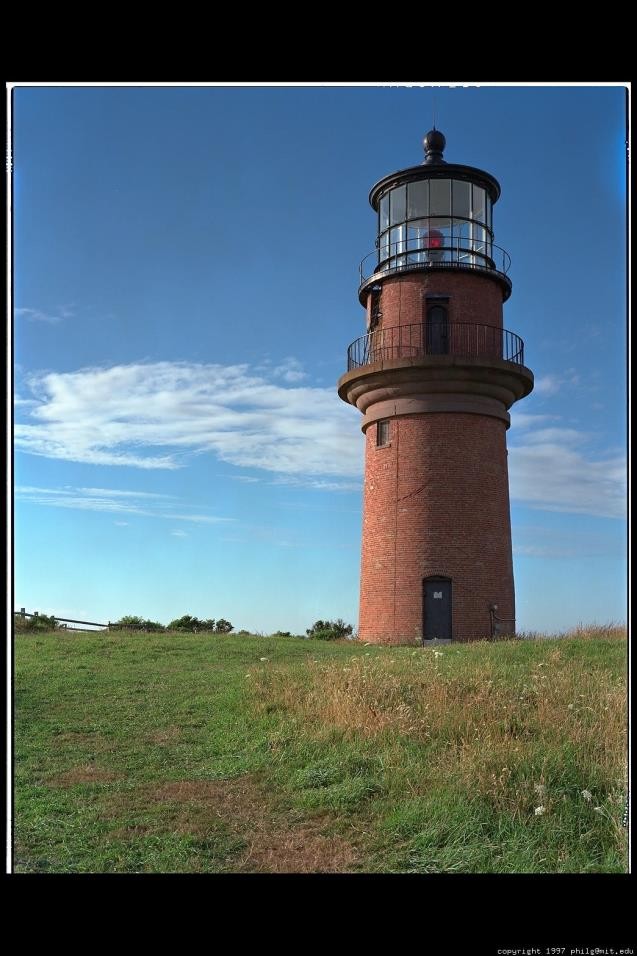
Here, our target is to recreate the Lighthouse image produced using the function



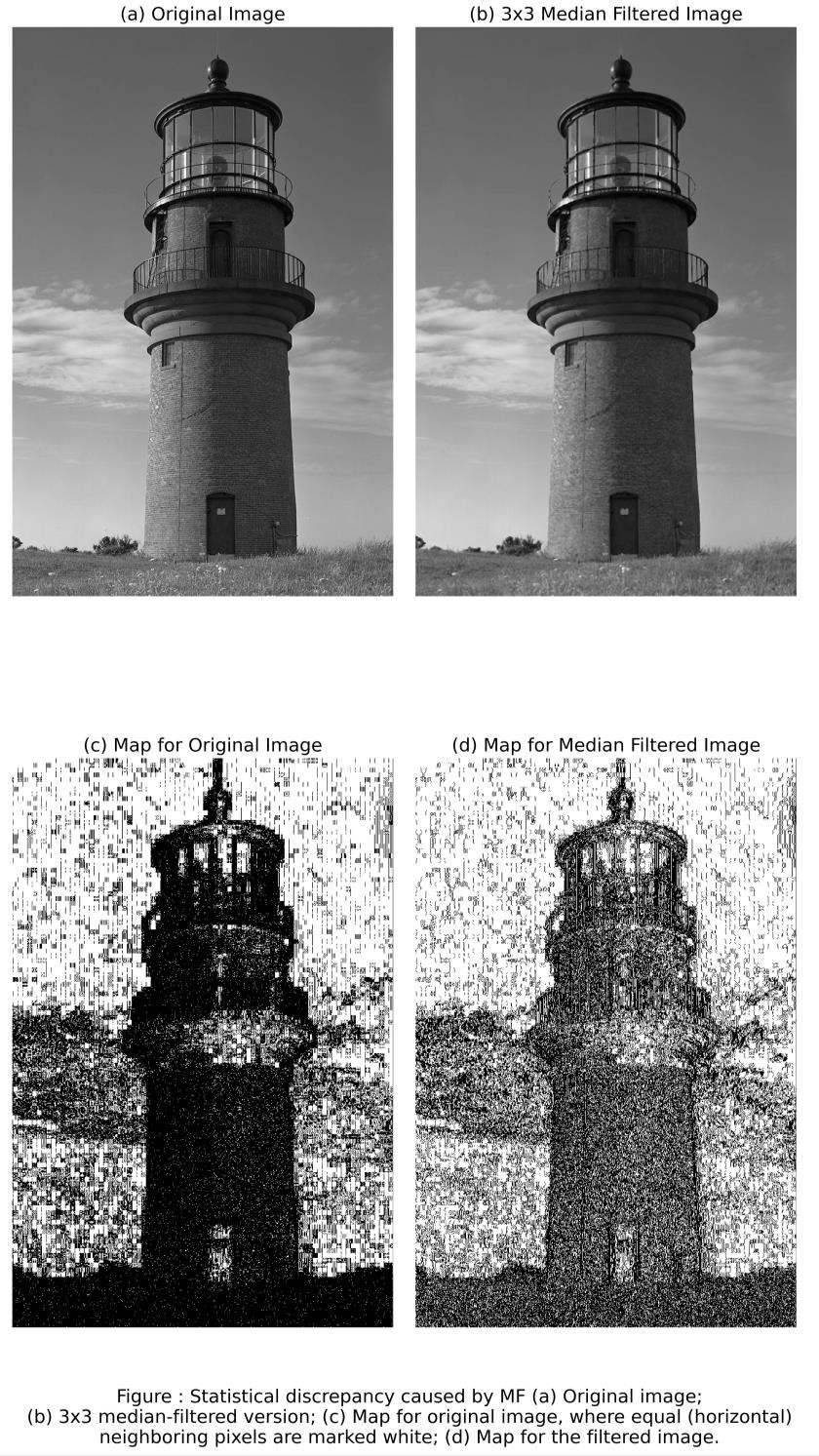
Now the actual author’s version of the image is



To recreate this, our first challenge is to find the same image. So we have done google image search and find this image at [http://philip.greenspun.com/images/pcd2331/gay-](http://philip.greenspun.com/images/pcd2331/gay-head-lighthouse-1.tcl) [head-lighthouse-1.tcl](http://philip.greenspun.com/images/pcd2331/gay-head-lighthouse-1.tcl)



This is a JPEG compressed image and larger in size, so we have cropped it at (366, 160, 1007, 1117) locations and applied the equation. Our result is



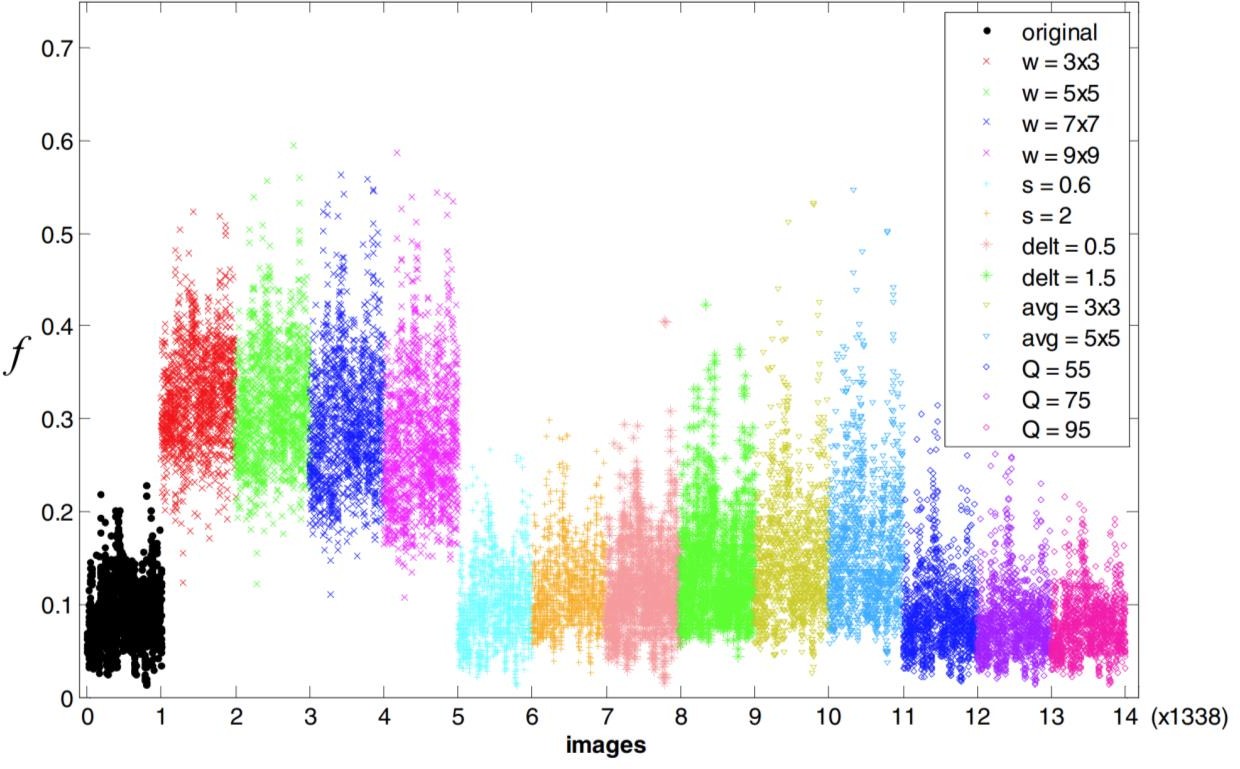
Now, we can say our result is almost exact as the author’s work, but there is a white patch at the middle of image (c). Our intuition is that, the white patch is generated due to underexposed region and further JPEG compression, which lead to similar pixel values.

But overall, the output is almost same as original work.

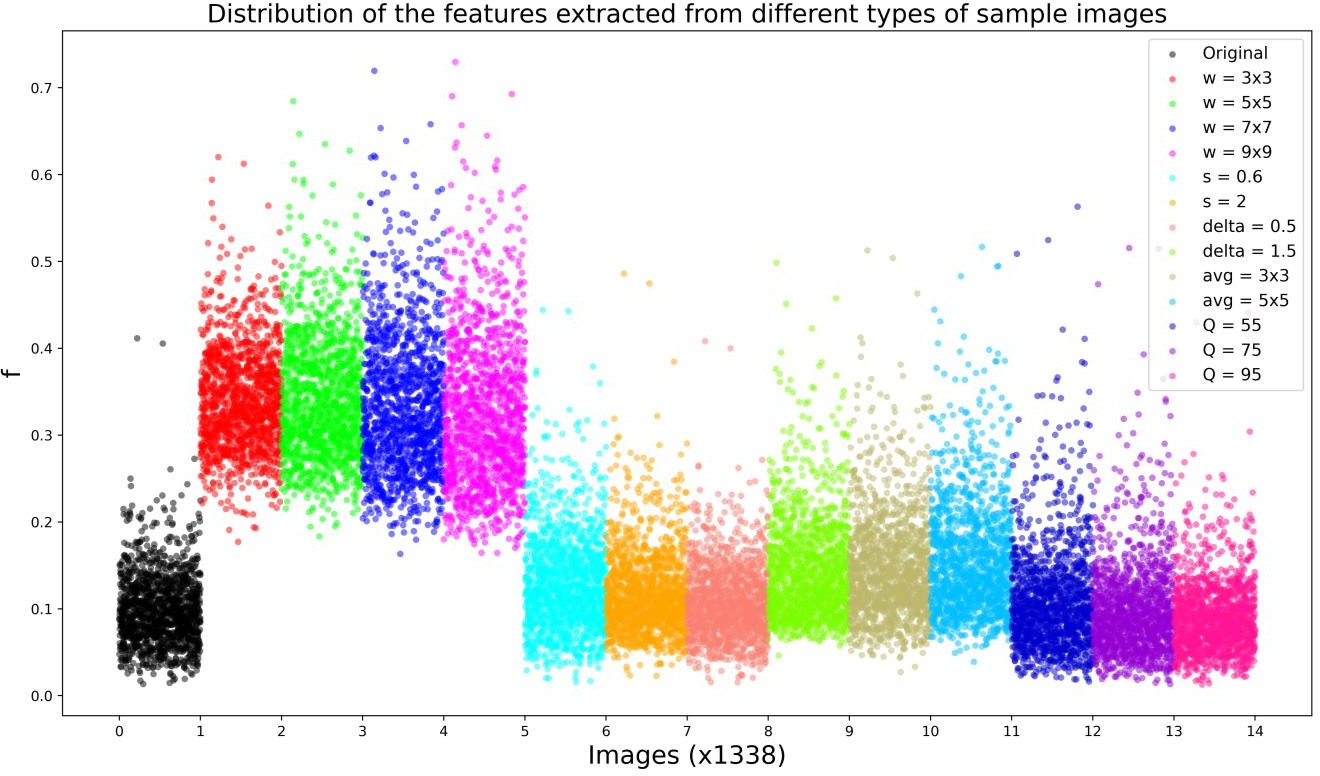
#### Main Distribution of f values

To get this diagram, we have written a almost 500 line code which mainly contains several operation on the images of UCID dataset and passing them through the algorithm to get the f value.

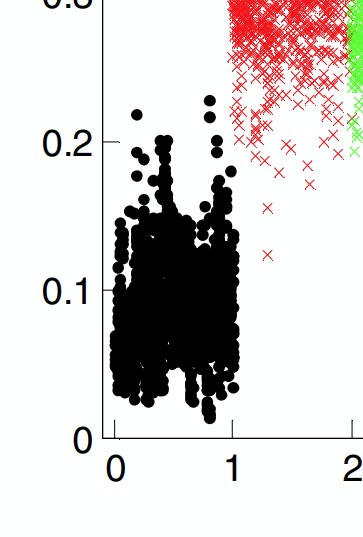
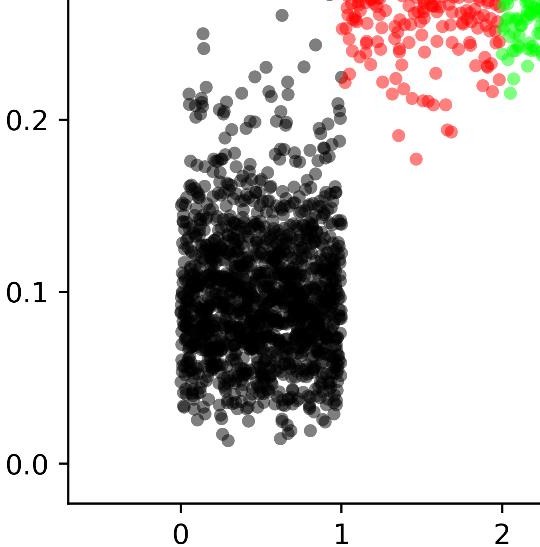
The diagram in the paper is as follows



Now, our version of the same is



Here also we can hardly find any significant difference between two plots. If we see very close, we can find,



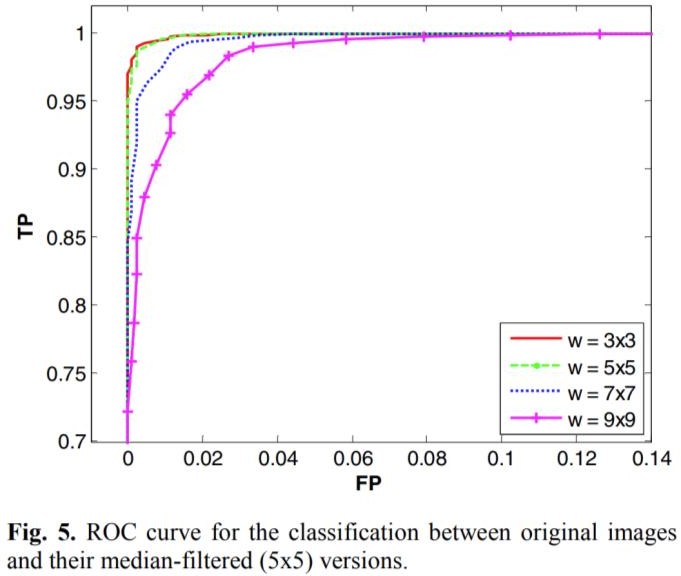
There are more points over the value f = 0.2 in our version as compared to the original one. And we can justify this by two reasons as,

* 1. Their UCID dataset may contain different images than ours
  2. We have done grayscale operation. It is possible, that they have used some other formula to convert RGB to grayscale.

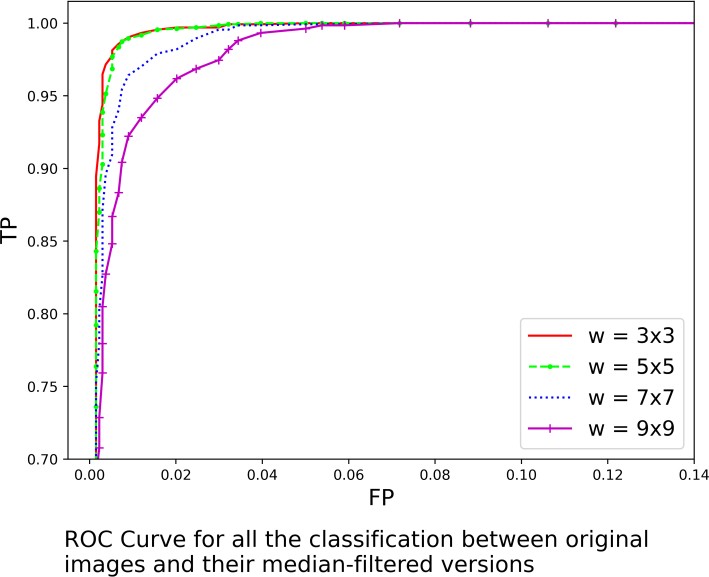
But at the end, the overall diagrams are pretty close. And one more point, this took almost 6 – 7h to generate the scatterplot.

1. ROC Curve:

Here, we are implementing only one out of four ROC curved, which is the baseline performance curve. The author has compared the classes original with all median filtered classes (3x3, 5x5, 7x7, 9x9) to get this ROC curve. The author’s version of the curve is



Now, we have plotted the same using our own data, which we get while calculating the Feature Distribution Values. I,e,



Similar to previous figures, here also we can see we are very close to the original work , but if we see very minutely, we will find very little difference which is due to the same reason discussed earlier.

So at the end we want to conclude that, all four diagrams we have recreated are very close to the original work.

## Improvements

Due to short of time and parallel projects we didn’t get enough chance to make significant improvement in our project, but we have done something.

As we have mentioned earlier, in a step of main algorithm, we need to calculate variance of a 7x7 matrix, and by analyzing the value of the variance, we can determine, whether the window is a textured region or not.

Now, initially, we have implemented the same using variance function in numpy (np) as,

for i in range(row-7):

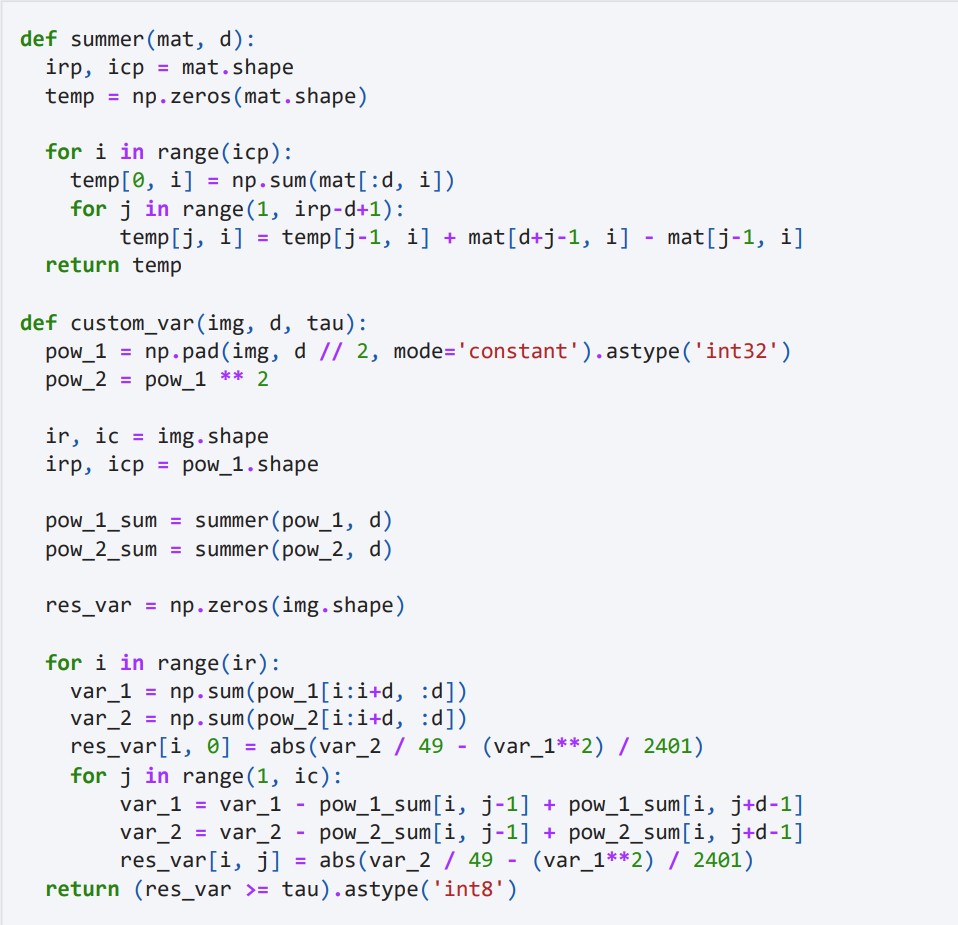
for j in range(col-7):

V[I, j] = np.var(img[i:i+7, j:j+7])

V = (V >= tau).astype(‘int8’)

But this implementation is taking average 2 hours to calculate f value for a batch of UCID images (1338 images). Now, we need to plot 14 such batches, so it will take days to calculate all values.

Now, what we have done is, we have improved the variance calculation technique and made it very efficient, so, now it is taking average 25 mins to calculate f values for a batch of images. The code is



## Contribution of Each Group Members

The three members in our group are:

* Anirban Haldar
* Atharva Inamdar
* Karande Jaysing Vitthalrao

Group Work:

In terms of group work, we have arranged google meet sessions almost every alternate day to make some progress about the project. The topics we discussed during the group meeting are

* Understanding the Paper & Discussion
* Making Presentation
* Making Report

So, we have spent almost 16h+ doing these works.

Individual Work:

And, the implementation of code and Figure reproduction is done by Anirban Haldar only. He took almost 14h+ to write all codes from scratch and recreate figures. To be frank, It wouldn’t take that long but he have tried to make the figures as close as possible to the figures present in the paper. At the end, it took 7 – 8h to run the code on UCID Dataset to generate our main figure containing 14 patches of f values for different types of images.