UNIVERSITY COLLEGE LONDON

Investigations on PRNU-Based Digital Camera Forensics

by

Benedikt Bjarni Bogason

This report is submitted as part requirement for the MSc Degree in Information Security at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Supervisor: Doctor Gwenaël Doërr

August 2009

Abstract

Digital image forensics is an emerging scientific field. Historically, investigators have been able to tie fingerprints to people, letters to typewriters, and bullets to firearms. They have achieved it via intrinsic investigations into grooves and patterns on fingertips, alignment and wear on typewritten letters, and scratches on bullets. Recently, there has been progress in establishing such ties between digital photographs and cameras. The most promising method involves examining the Photo Response Non-Uniformity Noise (PRNU), that is unique to each individual camera, due to imperfections in the camera manufacturing process.

This thesis covers the broad field of digital image forensics, including techniques of identifying, as well as warding off detection of various image manipulations. The main focus will be on the current state of PRNU estimation and detection methods. Each will be investigated in turn, with some new methods tried and compared to established ones. The practicality of PRNU-based identifications will be looked at, specifically whether current techniques are robust to dishonest parties trying to tinker with the identification process: trying to hinder or deceive the detection methods. Scalability issues are also of concern, since the number of digital cameras being sold each year is staggering. A simple experiment is devised to test the success rate of PRNU identification, when the number of candidate cameras exceeds 5,000.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Gwenaël Doërr, for his unyielding support throughout. His constant motivation and uplifting spirits have seen me through and enabled me to write this thesis. He has been actively involved in the whole process and tireless in suggesting improvements and new avenues of research.

Áslaug Högnadóttir and Patricia Thormar have graciously — and painstakingly — proof-read the manuscript and suggested countless refinements. To them I am most grateful. For any remaining errors, inaccuracies, or omissions I am alone culpable.

I would like to mention the support I received from the administrative staff throughout the year, both in Bloomsbury and especially at Adastral. Their service and helpfulness was exemplary.

Finally, I am indebted to my parents for enabling and encouraging me to further my studies. Thank you.

Contents

List of Figures

Chapter 1

Introduction

Photographic forgeries are nothing new. Arguably, they are as old as photographs themselves. Famously, Stalin doctored official photographs to remove traces of perceived political foes and rivals, such as Trotsky [4]. Even earlier, ca. 1860, an iconic portrait of Abraham Lincoln was concocted by splicing Lincoln's head onto the body of southern politician John Calhoun, who ironically was a strong supporter of slavery in the early part of his career (Figure 1.1). This was reportedly done in want of heroic portraits of Lincoln [16]. However, photographic manipulations were not solely exploited by political figures; the general public was prolific as well. In the early part of the twentieth century, sensations were caused by photographs "capturing" fairies, ghosts, and other paranormal phenomenons. The fairies invariably turned out to be painted cardboard cut-outs, and the ghosts were created by superimposing two pictures on top of each other [14].

Despite many famous incidents of well-crafted forgeries, people instinctively take photographs at face value, believing that they impartially document events that have taken place. For instance, photographs are routinely used as evidence in courts of law. The capacity of human memory for pictures far surpasses that of words, and images tend to get engraved in memory. A voter, asked days before the 2004 U.S. presidential elections why he would not vote for presidential candidate John Kerry, mentioned that he could not forget the image of Kerry at an anti-war rally with celebrity Jane Fonda. That picture had been revealed as a fake and, when reminded of the fact, the voter responded "I know, but I can't get the image out of my head" [16].

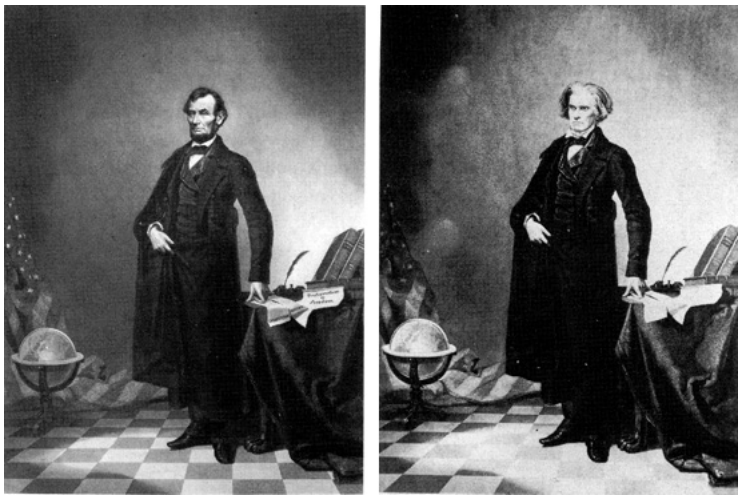


FIGURE 1.1: *Abraham Lincoln's doctored portrait, left, and the original portrait of John Calhoun, right. [18]*

Of course, with the dawn of the digital age, image manipulation has become incredibly easy. This has led to growing suspicion of forgeries and erosion of trust, especially between the public and news providers. In August 2006, Reuters published a photograph showing smoke emitting from a building in Beirut, Lebanon after being hit by an Israeli bomb (Figure 1.2). After it emerged that the photographer had doctored the image by adding extra smoke — apparently to exaggerate the impact of the attack — Reuters retracted the photograph and hundreds of others from the same source [6].

Perhaps even more serious than these examples of embellishing the truth, is the case of Professor Hwang Woo-Suk, who in 2004 published a scientific paper on groundbreaking advances in stem cell research. Investigation into accusation of fraud revealed that the images, purportedly showing the cell colonies Hwang claimed to have grown, were forged to a great extent [29].

In 2002, the U.S. Supreme Court ruled that portions of the Child Pornography Prevention Act that classified computer generated images as child pornography were overly restrictive and violated the First Amendment, which ensures free speech. The result is that virtual images of fictitious minors are constitutionally protected in the U.S. and the burden of proof has, in some cases, shifted from the defendants to the State as it must now prove that the images are real and not computer generated [14]. The problem is exacerbated by several court rulings that have found computer generated images to be so sophisticated that juries should not be asked to determine whether images are virtual or real [15].

FIGURE 1.2: *Reuters image, Lebanon. Strange repeating patterns in the smoke led to suspicion of forgery. [5]*

1.1 Forgery Detection

Even though digital image tampering can be extremely sophisticated, such manipulations may alter the underlying statistics of the image and thus enable detection [40]. A few of the most frequent alterations, and ways of detecting them, are explored in this section.

1.1.1 Cloning

One of the most common methods of manipulating images is to duplicate regions of the image, also known as cloning. This is usually done to remove or conceal objects in the image or to exaggerate the extent of some phenomenon, such as the smoke in Figure 1.2 or the number of soldiers in Figure 1.3. It is unlikely that a natural, untampered image has identical regions in it. As a result, their presence can be used as evidence of image manipulation [14]. Careful study can often spot such duplications, such as in the aforementioned pictures. However, cloned regions can be of any shape or size, and it is computationally unfeasible to search for all possible regions of cloning [17].

Robust matching has been proposed by Fridrich et al. [22] and a similar method by Popescu et al. [38]. Matching of replicas is done between small blocks of the image. The blocks are reordered so that identical or highly similar blocks are ordered side by side. A region-growing algorithm combines neighbouring blocks and the process is continued,

FIGURE 1.3: *George Bush's political ad from 2004, left and original picture, right. In the ad the president had been cut out and a region of soldiers duplicated and put in his place. [18]*

recursively [15]. The authors use Discrete Cosine Transform (DCT) and/or Principal Component Analysis (PCA) to decrease computational complexity and to be robust against lossy JPEG (Joint Photographic Experts Group [3]) compression [17, 31]. There is a problem with erroneously detecting regions of cloning in flat uniform areas and, therefore, the results should be inspected by humans to decide the correctness of the output [31].

1.1.2 Stitching

Stitching, when pieces from different photographs are combined into a new image, is a form of doctoring that is often difficult to detect. Figure 1.1 is an example of stitching. Detection, however, is not impossible. One ingenious observation is that objects originating from different photographs can display subtle lighting inconsistencies because the source of light is different between images [15]. Johnson and Farid have developed a method of inferring the direction of light sources on objects in photographs. This is achieved by examining points along surface contours, where the orientation of the surface is perpendicular to the contour and assuming the surface is Lambertian [14, 15, 27, 28]. An example of this method is depicted in Figure 1.4. A similar methodology is described by Farid [15] that examines the position of the eyes and how light reflects in them. It can be used to determine whether all persons in the photograph were captured under the same lighting conditions. Yet another detection method is based on noticing inconsistent Colour Filter Array patterns in certain regions of the image. Colour Filter Arrays are discussed in greater detail in Section 1.2.1.

FIGURE 1.4: *The ducks (3,4) were photographed under different lighting direction than the policemen's helmets (1,2). The light source is directly above the policemen (due north), whereas it is northwest of the ducks. [28]*

1.1.3 Double Quantisation

Most cameras store images in JPEG format. When images are manipulated in image processing programs, such as Photoshop®, the images are frequently reformatted in JPEG, leading to double compression. The second compression might be done with a different quality factor than the first. Such double compression, if coupled with quantisation as in the JPEG format, introduces special artifacts that can be detected. To aid discussion, it is assumed that an image is first compressed with quality factor q_1 that involves quantisation with step-size Δ_1. Secondly, the image is compressed again with another quality factor, q_2 which involves quantisation step-size Δ_2. If $\Delta_1 > \Delta_2$, some bins in the doubly compressed image histogram are empty, because the first quantisation step placed the samples in a few bins, that were re-distributed into a greater number of bins. Conversely, when $\Delta_1 < \Delta_2$, some bins of the histogram will contain more samples than their neighbouring bins. In either case, some special artifacts can be detected in the histogram that were not present in the singly compressed image (Figure 1.5) [17, 41].

FIGURE 1.5: *Left: histogram of once JPEG compressed image. Right top: twice compressed image where second quantisation step is smaller than the first. Right bottom: twice compressed image where the second step is greater. [17]*

1.1.4 Re-sampling

Another frequent operation on photographs is called re-sampling. It can involve one or more of the following; resizing, rotating, or stretching the image. Detection of re-sampling is based on the idea that up-sampling introduces correlation due to interpolation. Furthermore, re-sampling at arbitrary rates requires a combination of up- and down-sampling, and therefore also introduces correlated periodic coefficients [41]. Since investigators on average know neither the specific form of re-sampling the image underwent, nor which pixels are correlated, an exhaustive search must be carried out to examine whether these correlations exist. For that purpose, an iterative process based on the expectation/maximisation (EM) algorithm is used [17, 39, 41].

A few papers have set out to deceive the methods already discussed [24, 30]. Clearly there is an interest in the field. Scientists are actively investigating better forensic mechanisms, to catch up and possibly in the future to stay ahead of the manipulators of digital images.

1.2 Device Identification

The origin of photographs is important in many scenarios. Having reliable information about image paternity makes it possible to distinguish real photographs from computer generated images, and is essential for use in court cases. Digital cameras already prefix meta-data (such as camera model etc) to photographs in Exchangeable Image File Format (EXIF). However, such information is fragile as it can be destroyed by normal processes or worse still, can easily be replaced [32]. It has been proposed that watermarks be automatically inserted by digital cameras at the time of capture, carrying information about the digital camera, time-stamp, and even biometric of the person taking the image. However, such watermarks limit the usefulness to future use only, and every camera manufacturer has to implement such a mechanism [17, 33, 40]. Therefore, this idea has not taken off, although it could be useful in closed environments, such as in criminal scene investigations.

Defective pixels in digital cameras have been proposed as a method of distinguishing cameras even of the same make and model. Defective pixels can e.g. be “dead” or

"hot" (constantly black or white). These pixels create a unique consistent fingerprint, theoretically. However, studies have shown that the number of visible defect pixels varies with variables such as temperature and even image content. Furthermore, most cameras compensate and eliminate defective pixels on-board, thus rendering this method mostly mute [31, 33, 41].

As Farid has pointed out [17], most digital cameras use lossy JPEG format to compress and store images. Each camera manufacturer is free, to a certain degree, to tinker with the quantisation tables in an effort to balance quality and degree of compression. The quantisation tables can vary within a single model as a function of a quality setting, and conversely, cameras of different make can share the same tables. This method — although unsophisticated — does enable crude identification of camera models.

Two methods of device identification will be considered: Colour Filter Arrays (CFA) and Photo Response Non-Uniformity Noise (PRNU). The former is used to distinguish between different makes and models of cameras, while the latter can be used to differentiate between individual cameras, even of the same model.

1.2.1 CFA

A digital colour image consists of three colour channels, usually RGB (red, green, and blue). In digital cameras, Charged Coupled Device (CCD) or Complementary Metal-Oxide Semiconductor (CMOS) chips are used as sensors to capture the image's pixels. Sensors are insensitive to colour and only record the brightness of light, thus producing a monochromatic output. To produce a colour image, each pixel records the light intensity for a single colour, and then a Colour Filter Array (CFA) is used to interpolate (or demosaic) the missing values. Therefore, only a third of the samples in an image are captured by the camera; the rest are interpolated from those values (Figure 1.6) [31, 40, 41]. As mentioned before, there are different ways of interpolating and different camera manufacturers incorporate their preferred method, thus enabling analyst to distinguish between models [8, 42].

However the interpolation process is conducted, it will inevitably introduce a statistical correlation between the interpolated and original pixels [37]. Moreover, the possible absence of this correlation — that occurs by default in digital images — can be used

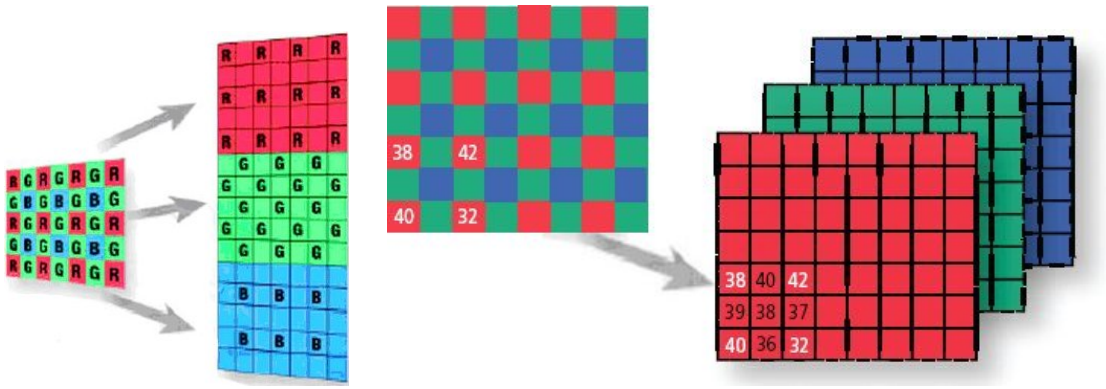


FIGURE 1.6: *Colour Filter Array. Each pixel captures only one principal colour. The missing values are interpolated from the captured values, separately for each colour. [2, 15]*

as an indication of image tampering, or that the image is computer generated [15, 43]. This method is hampered by the fact that it only works for original images. Lossy JPEG compression, resizing, or a scan of the photograph will distort these correlations [15].

Of course this identification method is vulnerable to counter-attacks. To ensure investigators cannot identify from which camera model an image came, the attacker simply has to corrupt the correlation, which is relatively easy. On the other hand, if the attacker wishes to make it seem that the image originated from a different camera model than it really came from, the attacker has to have knowledge of the camera's CFA pattern and interpolation algorithm. That requires the forger to gather significantly more information [40].

1.2.2 PRNU

One promising area of camera identification is based on *Photo Response Non-Uniformity Noise* (PRNU) analysis [11, 20, 25, 32, 33]. The *pattern noise* is defined as any noise component that survives frame averaging. One component of this noise is the *fixed pattern noise* which is a signal collected from the sensors when they are not exposed to light, also known as "dark currents". This fixed pattern noise can be used for camera identification, only if we have access to dark photographs taken with the camera, which may not always be the case [33]. Another more prominent component of the pattern noise (and indeed dominant in natural images) is the PRNU that stems from different sensitivity of pixels to light because of imperfections during manufacturing of the sensors. Light refraction on dust particles can also contribute to the noise. These imperfections

are not affected by ambient temperature or humidity and it is unlikely that different arrays of sensors exhibit correlated PRNU patterns [33].

The PRNU is consistently observed in all photographs from a particular camera, and it can be detected by conventional watermarking techniques. A reference pattern for the camera is obtained by averaging a large amount of photographs taken with the camera (ca. 50 images), thereby strengthening the PRNU and decreasing the strength of everything else (the image content). A photograph suspected of being taken by the camera is correlated with the reference pattern and, if above some detection threshold, it is deemed to have originated from the camera.

1.3 Structure of the Thesis

The rest of the thesis is organised as follows. Chapter 2 describes in details the current methods of PRNU estimation and detection, with special attention to the possible shortcomings and interesting avenues of research. In Chapter 3, enhancements to the PRNU estimation process are considered, while Chapter 4 is focused on comparing detection methods. Practical usability issues are briefly discussed in Chapter 5, such as scalability and handling attacks. Chapter 6 concludes the thesis. The bulk of the relevant source code is appended at the end.

Chapter 2

PRNU Based Camera Forensics

2.1 Overview

A simplistic explanation of the major steps in image capturing by digital cameras is illustrated in Figure 2.1. The camera captures light that rebounds from objects in a scene. The light passes through the lens, gets optically filtered and is recorded by an array of sensors. The vast majority of commercial cameras use a colour filter array instead of recording all three primary colours in each pixel. This allows lower production cost and averts the difficulties of aligning the three colour planes. The missing colour values are interpolated and the three colour channels go through post processing, such as colour correction, white balancing, gamma correction, to reduce disk space and improve picture quality [42].

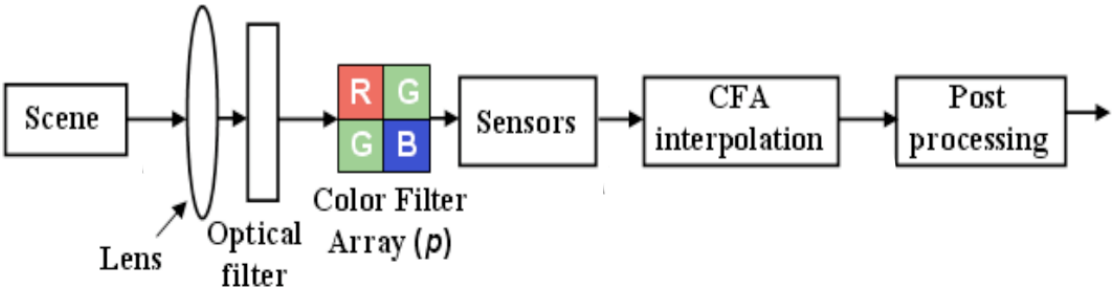


FIGURE 2.1: *A simplified model of the inner workings of a digital camera. [42]*

The many steps of image acquisition introduce various sources of noise. *Pattern noise* is comprised of any part of the noise that survives frame averaging, the dominant part of which is the photo-response non-uniformity noise (PRNU). Different pixels of the camera

are of slightly different dimensions and show variance in their sensitivity to light caused by the inhomogeneity of silicon wafers used in the camera sensors and other imperfections during the sensor manufacturing [33]. This difference in light sensitivity is constant, not affected by temperature, humidity, etc. Therefore, the PRNU noise is approximately the same when multiple photographs capture the same scene. In addition, the PRNU is robust to various kinds of image processing due to its spread spectrum nature, making this signal ideal for camera identification purposes [20].

The PRNU, $\mathbf{w}$, has been modelled as multiplicative noise that gets inserted onto the original image content, $\mathbf{i}_{\text{orig}}$, with some strength α as well as some random noise, $\mathbf{n}$, resulting in the camera image $\mathbf{i}$,

$$\mathbf{i} = \mathbf{i}_{\text{orig}} \times (\mathbf{1} + \alpha\mathbf{w}) + \mathbf{n}. \tag{2.1}$$

The multiplication is done element-wise.

2.2 PRNU Estimation

In order to estimate the PRNU, denoising techniques can be used to separate the noise from the actual image content, forming something called the *noise residual*. However, a large portion of the noise residual energy is due to some artifacts in the scene content, rather than being the PRNU that is to be estimated. For that reason, the noise residuals from multiple images from the same camera are combined, assuring that the PRNU part will be amplified since it is present in all images, but the random component from the subject matter gets attenuated [33]. To be able to distinguish noise from the image content, the variance in small neighbourhoods around each pixel is calculated. If a pixel is substantially different from its otherwise homogeneous neighbourhood, that pixel is deemed to be *noisy*. If the neighbourhood variance is large, i.e. inhomogeneous, that part of the image content is deemed semantically important, and should not be denoised.

2.2.1 Denoising

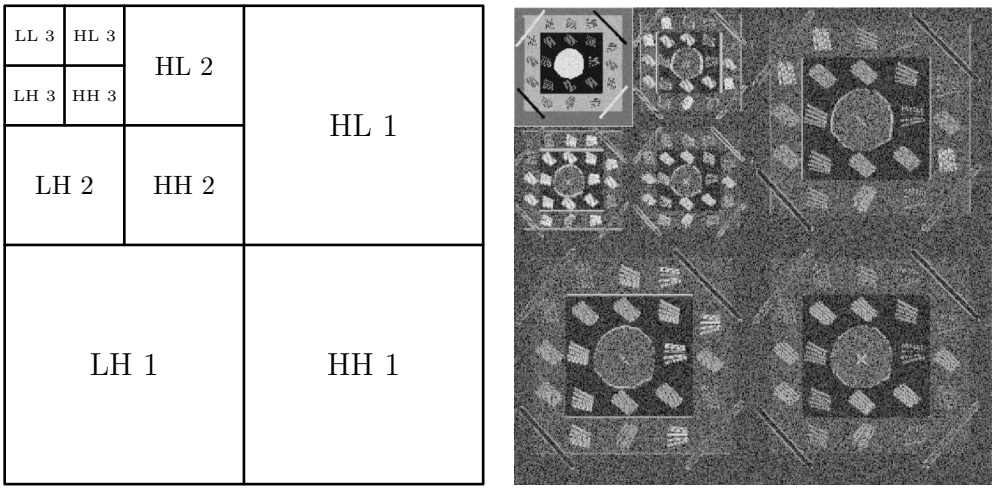
2.2.1.1 Wavelet Transform

The denoising filter originally proposed for PRNU estimation is the one by Mihçak et al. [34]. It performs Wiener filtering in the wavelet domain once the local variance in each band has been estimated.

The conventional 1-D wavelet transform separates the signal into two components: low and high frequency. This is done recursively on the low frequency part for each level of the transform. The result is that with each added level, the finest details are filtered out and only the coarser, basic shapes remain. With 2-D images, the two principal directions (horizontal and vertical) are treated separately, therefore giving 4 sub-bands for each level:

- HL which is a horizontal high-pass filter but a low-pass vertical filter.
- LH which is a horizontal low-pass filter but a high-pass vertical filter
- HH which is a high-pass filter in both horizontal and vertical directions.
- LL is a low-pass filter in both directions, i.e. it is a scaled down version of the coarser parts of the image, and the wavelet transform operates on this sub-band recursively.

A low-pass filter discards all high frequencies, whereas a high-pass filter discards all low frequencies. Figure 2.2(a) explains the position of each sub-band within the wavelet transform. Figure 2.2(b) depicts the first 2 levels in the wavelet domain for a synthetic image with multiplicative noise. For completeness, it is noted that the denoising filter uses four level decomposition with 8-tap Daubechies quadrature mirror filter. Due to the particular implementation of the Wabelab package [1] used in this thesis, the image is padded so that its dimensions are multiples of 16 (2^L, where L is the number of levels).



(a) (b)

FIGURE 2.2: *(a) Illustration of the wavelet sub-bands for the first tree levels. Further levels recursively replace the LL sub-band. (b) 2 level wavelet transform of a synthetic image with multiplicative noise. For illustration purposes, a logarithmic scale of magnitude is used for all sub-levels except LL 2.*

2.2.1.2 Local Variance Estimation

Each sub-band (HL, LH, and HH) of each level is processed separately. For each sub-band, the local variance of the original noise-free image is estimated. The neighbourhood of each pixel is determined as a $\mathcal{W} = W \times W$ square using four different values for $W \in \{3, 5, 7, 9\}$. The variance estimation is as follows [21, 33]

$$\hat{\boldsymbol{\sigma}}_W^2(i,j) = \max\left(0,\ \frac{1}{|\mathcal{W}|}\sum_{(i,j)\in\mathcal{W}} \mathbf{b}^2(i,j) - \sigma_0^2\right), \qquad (i,j) \in \mathcal{J} \tag{2.2}$$

where $\mathbf{b}$ is a sub-band of the wavelet transform, (i, j) are indices of that sub-band, which run through an index set $\mathcal{J}$ that depends on the decomposition level, and σ_0 is a parameter of the denoising filter. This results in four different estimations of the local variance, $\hat{\boldsymbol{\sigma}}_3^2, \ldots, \hat{\boldsymbol{\sigma}}_9^2$. The final estimate of the variance, $\hat{\boldsymbol{\sigma}}^2$, is obtained by taking the minimum of those as follows,

$$\hat{\boldsymbol{\sigma}}^2(i,j) = \min_W(\hat{\boldsymbol{\sigma}}_W^2(i,j)), \qquad W \in \{3,5,7,9\}. \tag{2.3}$$

2.2.1.3 Noise Residual Estimation

Having obtained the variance estimations, $\hat{\boldsymbol{\sigma}}^2$, the wavelet sub-band is denoised using a Wiener filtering technique:

$$\mathbf{b}_{\text{den}}(i,j) = \mathbf{b}(i,j)\frac{\hat{\boldsymbol{\sigma}}^2(i,j)}{\hat{\boldsymbol{\sigma}}^2(i,j) + \sigma_0^2} \tag{2.4}$$

The denoised sub-bands are reassembled (as in Figure 2.2(a)), inversely wavelet transformed, and finally cropped to the original image size, to create a denoised version of the image, $\mathbf{i}_{\text{den}}$. The denoised version is subtracted from the original image, thus isolating the noise residual $\mathbf{n_i}$,

$$\mathbf{n_i} = \mathbf{i} - \mathbf{i}_{\text{den}}. \tag{2.5}$$

2.2.2 Combining Individual Estimations

To estimate a specific camera's PRNU, several photographs from the source are needed. Chen et al. [10] suggested a method for estimating the PRNU. For each image, the noise residual is computed and the PRNU, $\mathbf{w}_1$, is obtained as:

$$\mathbf{w}_1 = \frac{\sum \mathbf{n_i} \times \mathbf{i}}{\sum \mathbf{i} \times \mathbf{i}}, \tag{2.6}$$

where $\mathbf{n_i} \times \mathbf{i}$ is the element-wise multiplication of $\mathbf{n_i}$ and $\mathbf{i}$. The sum is over multiple images and their noise residuals. The division is done element-wise as well.

Chen et al. [11] later pointed out that PRNUs from different cameras are slightly correlated because of similar processes that are the same or similar across different camera models, such as colour interpolation which was discussed earlier in Section 1.2.1.

They propose to eliminate this systematic correlation between pixels by subtracting the column average from each pixel in the column, and similarly for each row, thus resulting in a PRNU with zero mean in every row and column. This operation is denoted as ZM(.), and defines a second PRNU estimation,

$$\mathbf{w}_2 = \text{ZM}(\mathbf{w}_1). \tag{2.7}$$

Indeed, Figure 2.3 confirms the presence of a distinctive linear pattern in the $\mathbf{w}_1$ PRNU estimate that is absent in $\mathbf{w}_2$. Removing such a pattern should provide a better estimation of the PRNU, and avoid possible mistakes, as the linear pattern would contribute to a higher correlation score with PRNUs from other cameras that have a similar linear pattern, e.g. due to the same CFA interpolation technique.

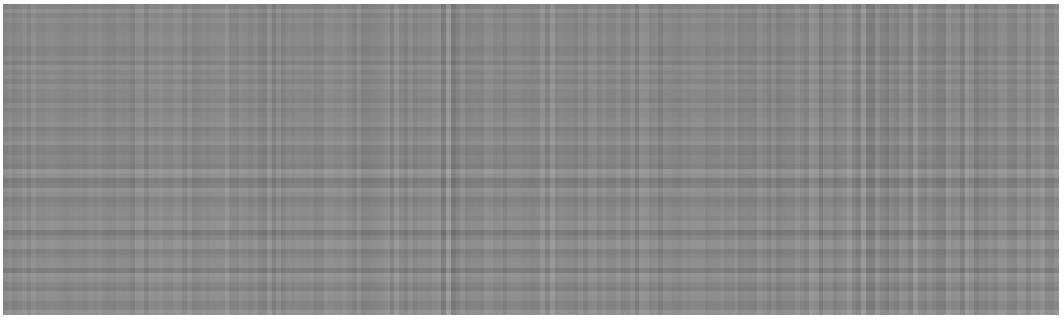


FIGURE 2.3: *A small subset of the linear pattern* $\mathbf{w}_1 - \mathbf{w}_2$ *that demonstrates the presence of correlation in rows and columns of natural images. The camera in question is Canon EOS D30.*

2.2.3 Impact of σ_0

The issue of how to select the parameter σ_0 has been ignored until now. Observations on the impact from the selection of σ_0 can be made by examining Equations (2.2) and (2.4). From Equation (2.2) it can be deduced that a large value of σ_0^2 will set the variance estimation ($\hat{\boldsymbol{\sigma}}^2$) to zero, thus having a large impact on the denoised version. The value of σ_0^2 therefore introduces a *threshold* in the variance estimation. If the variance in the neighbourhood of a pixel is quantitatively smaller than σ_0^2, the variance for that pixel is set to zero, i.e. it is determined to be homogeneous to its neighbourhood. Changes to that region are unlikely to make much difference to the perception of the content, so an observer will not notice much difference if that region is altered.

When $\hat{\boldsymbol{\sigma}}^2(i,j) \gg \sigma_0^2$ it follows from Equation (2.4) that $\mathbf{b}_{\text{den}} \approx \mathbf{b}$, meaning that little or no denoising is performed. In other words, all parts of the region should be preserved, no parts should be removed as noise. On the other extreme, if $\hat{\boldsymbol{\sigma}}^2(i,j) \ll \sigma_0^2$, the same equation dictates that $\mathbf{b}_{\text{den}} \approx \mathbf{0}$, i.e. that region of the sub-band is essentially noise, and should be denoised, thus resulting in a smoothed image.

From this discussion, it can be deduced that any noise with variance less than σ_0^2 is removed from the image, which is appropriate if the band has been corrupted by additive white Gaussian noise (AWGN) with variance σ_0^2.

2.3 PRNU Detection

Fridrich discussed achieving optimal PRNU detection in terms of processing speed [21]. The derivation is skipped, with only the results reproduced here. To detect whether a particular image, $\mathbf{i}$, was taken by a camera with an estimated PRNU, $\mathbf{w}$, the correlation between the PRNU value and the noise residual of the image (see Equation (2.5) for reference) is calculated using the *normalised correlation coefficient*,

$$\mathrm{cc}(\mathbf{n_i}, \mathbf{w}) = \frac{(\mathbf{n_i} - \overline{\mathbf{n_i}}) \cdot (\mathbf{w} - \overline{\mathbf{w}})}{|\mathbf{n_i} - \overline{\mathbf{n_i}}| \; |\mathbf{w} - \overline{\mathbf{w}}|}, \tag{2.8}$$

where $\mathbf{a} \cdot \mathbf{b}$ is the dot-product of $\mathbf{a}$ and $\mathbf{b}$ (extended naturally to 2-D signals), $|.|$ is the L_2 norm, and $\overline{\mathbf{w}}$ denotes the mean value of $\mathbf{w}$. If and only if the correlation is above some set threshold τ, is it asserted that the image was taken with the camera in question.

2.3.1 False-Positives and False-Negatives

When the detection method erroneously identifies photographs from other sources as taken with the scrutinised camera, it is considered a *false-positive*. On the other hand, a photograph truly taken with the camera that shows correlation value below some threshold value τ is named a *false-negative*.

Figures 2.4 and 2.5 show typical probability density functions (PDFs) for correlation scores from two cameras, each with its specific PRNU. The curve on the right shows the distribution of correlation scores of photographs captured with the camera from which the PRNU is estimated, while the curve on the left shows correlation score from some other camera. That curve is centred around zero, as is to be expected since it is the correlation between two statistically independent signals. Table 2.1 explains the definitions of the four possible states a detection method can associate with an image. The shaded areas in Figures 2.4 and 2.5 demonstrate the same thing, graphically.

TABLE 2.1: Confusion Matrix

	Actually YES	**Actually NO**
Hypothesised YES	True-Positive	False-Positive
Hypothesised NO	False-Negative	True-Negative

A detection threshold has been set as τ, and any photograph that has correlation value above τ is determined to be captured by the camera, any photograph that has correlation value below τ is deemed not to be from the camera. As shown in Figure 2.4, depending on the value of the threshold τ, we get some incorrect decisions. Images that are from the distribution on the left, but with correlation values above τ are false-positives, shown with a light shadow. Images from the distribution on the right with correlation values below τ are false-negatives, darkly shaded. As the image suggests the proportion of false-negatives or false-positives can be influenced by changing τ, but not in isolation. By decreasing the number of false-positives there is a simultaneous increase in the number of false-negatives, and vice versa.

2.4 Room for Improvement

The digital image forensic field is still in its infancy; although great breakthroughs have already been achieved. There are many opportunities for refining and innovating new methods for the whole process. A small subset of them are mentioned here.

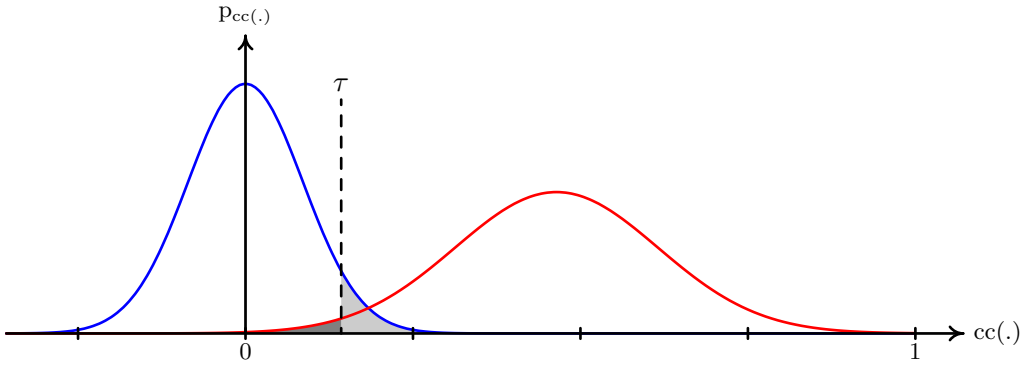


FIGURE 2.4: *Graphical demonstration of false-negatives (dark area) and false-positives (light area). Detection threshold is set as τ.*

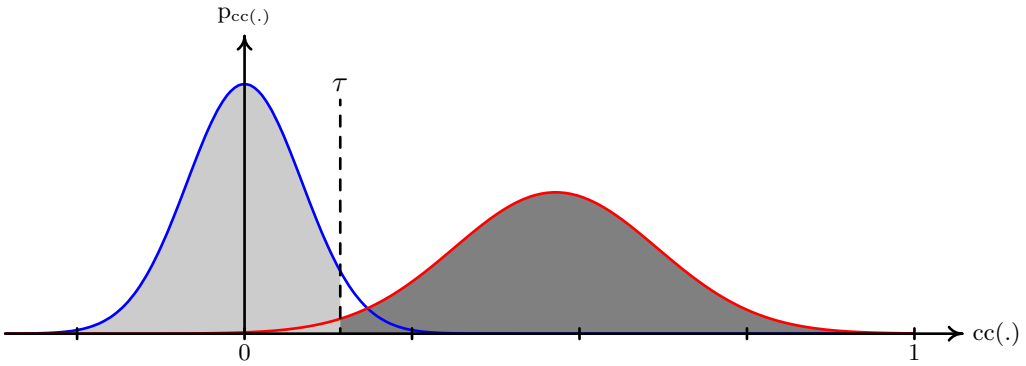


FIGURE 2.5: *Graphical demonstration of true-negatives (light area) and true-positives (dark area). Detection threshold is set as τ.*

2.4.1 Multiplicative Denoising Filters

The use of the Wiener filter for denoising in Equation (2.4) is based on the assumption that the noise to be extracted is additive. However, since the PRNU is multiplicative in nature, it would be logical to make some changes to the denoising filter to accommodate the fact that the noise variance is not constant throughout the image, but dependent on the image content. This idea is pursued and explained further in Section 3.1.1.

2.4.2 Detection Metric

The detection method of PRNUs — captured in Equation (2.8) and derived by Chen et al. [11] — is perhaps not the optimal detection method, regardless of the persuasive arguments in the paper. The object is to detect noise patterns inserted in images by digital cameras and is therefore similar in structure and scope as the well known and researched field of *watermarking* [12]. It should, therefore, be possible to build upon the extensive literature and tried and tested methods of signal detection in watermarked images. This idea will be explored in Chapter 4.

2.4.3 Reliability and Dependability

Defending against new kinds of attacks on digital images is likely to be a never-ending race, where adversaries continually try to find — and exploit — weak points of the system, while the defenders react and plug the holes. It could be argued that attacks on PRNU detection methods are not relevant yet, mainly because the actual methods are

primitive, and not currently in general use. Even so, possible attacks on the detection methods should be considered from the start alongside the method development, in hope that later attacks require more effort from the adversary. A plethora of attacks can be conceived; only one of them is considered in Section 5.1.1, adversaries dishonestly removing a PRNU from photographs.

The current method of PRNU detection seems to be well suited to determining the paternity of a particular image from a pool of say 2 or 3 possible cameras, with reasonable certainty. Experiments on the scalability of this approach are inconclusive. Whether this method will be able to distinguish the exact camera from a database of tens of thousands of candidates is still debatable. As a proof of concept, a simple experiment is proposed in Section 5.2.

Chapter 3

Evaluation of Different PRNU Estimation Techniques

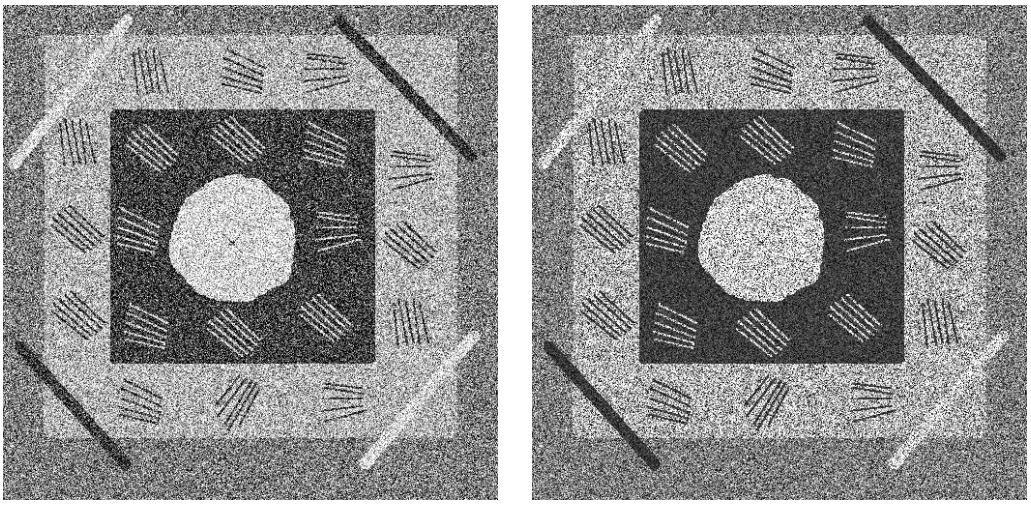
3.1 Potential Enhancements of Estimation Techniques

3.1.1 Pixel Dependent Variance Estimation

The PRNU estimation technique introduced in the previous Chapter has several potential issues that could be improved or researched further. One of the most obvious issues is that the method relies on a noise filter for *additive* noise, when it would be more appropriate to use a filter for multiplicative noise, since PRNU is believed to be of multiplicative nature. Additive noise is content-independent, meaning that the noise level is the same throughout the image, or put another way; the noise variance is constant. In contrast, multiplicative noise depends on the image content, the noise variance is much greater in lighter areas of the image as opposed to the darker regions. Figure 3.1 demonstrates the difference between additive and multiplicative noise.

In an effort to mitigate this shortcoming, a minimal change to the Wiener filter is proposed. Using a pixel dependent version of the σ_0^2 parameter — instead of the pixel independent version in Equations (2.2) and (2.4) — makes it possible to take the image content into account. The desired effect is achieved by substituting σ_0 with

$$\boldsymbol{\sigma}_0(i,j) = \sigma_0 \times \breve{\boldsymbol{i}}(i,j), \tag{3.1}$$



(a) (b)

FIGURE 3.1: *Image with additive noise, left, and same image with multiplicative noise, right. Additive noise demonstrates noise with the same variance in all parts of the picture, while the variance of the multiplicative noise is dependent on the image content; the variance is greater in lighter areas than in darker areas.*

where $\breve{\boldsymbol{i}}$ is a scaled down version of the image so its dimensions match those of the sub-band, $\mathbf{b}$. This means the variance of the noise to be eliminated in each pixel is a factor of the pixel value of the original image. If that pixel is dark, the noise to be eliminated has low energy, whereas in a brighter pixel a greater variance in the noise is anticipated.

As a side effect, Equation (2.4) becomes

$$\mathbf{b}_{\text{den}}(i,j) = \mathbf{b}(i,j)\frac{\hat{\boldsymbol{\sigma}}^2(i,j)}{\hat{\boldsymbol{\sigma}}^2(i,j) + \boldsymbol{\sigma}_0^2(i,j)}. \tag{3.2}$$

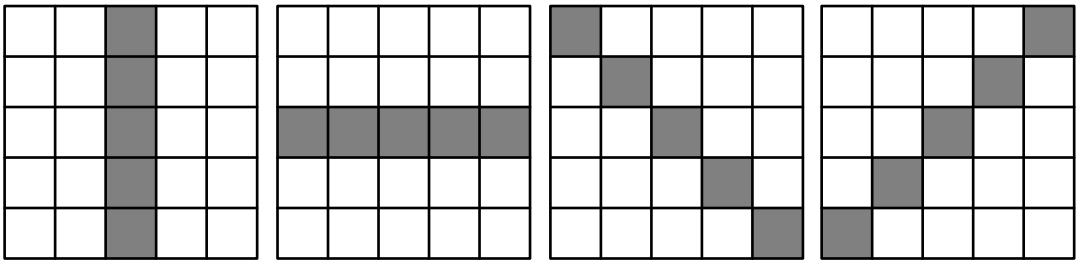
The PRNU estimation with pixel dependent variance estimation is named $\mathbf{w}_3$ to distinguish it from $\mathbf{w}_2$ in Equation (2.7).

3.1.2 Directional Filters

Hawwar et al. [26] proposed using 1-D vectors instead of the 2-D square windows, $\mathcal{W}$, for the neighbourhood variance estimation discussed in Section 2.2.1.2. The reason is that in the HL sub-bands (Figure 2.2(a)) mainly vertical features of the image are present, because the wavelet transform uses a high-pass horizontal filter and a low-pass vertical filter in those sub-bands. Therefore, a vertical vector should be used as a window to define the neighbourhood. This change should address one problem of the traditional

method; it has difficulty detecting edges and is therefore prone to blur the edges in the denoising process. With this proposed change, hard vertical, horizontal, and diagonal edges should be preserved in the denoising process.

For the same reason, a horizontal vector should be used as the window $\mathcal{W}$ in the LH sub-bands, where mainly horizontal features are present. Finally, the HH sub-bands contain mostly diagonal features, so a diagonal window should be used. The HH sub-band is different from the other two, since 2 windows have to be considered as there are two diagonals. Finally, the maximum of the two variance estimations (given by the two diagonal windows) for each pixel is used. In all other aspects the method proceeds as before, specifically Equation (2.2) is unchanged except that $\mathcal{W}$ has different dimensions than before. The windows are now 1-D vectors of length $W \in \{3, 5, 7, 9\}$, as depicted in Figure 3.2. The difference in variance estimation, with and without directional filters, is illustrated in Figure 3.3. We denote the presence of spatially adaptive filters such; $\tilde{\mathbf{w}}$.



(a) HL (b) LH (c) HH

FIGURE 3.2: *Spatially adaptive windows, $\mathcal{W}$, shown here for $W = 5$. From left to right: for HL-sub-bands, preserves vertical features (a). LH-sub-bands, preserves horizontal features (b). Two windows are used for HH-sub-bands, preserves diagonal features, in either direction (c).*

3.2 Experimental Setup

3.2.1 Image Dataset

For the experiments 500 computer generated images were downloaded from the website *myHDWallpaper* (`http://myhdwallpaper.com/`), that among other things offers free downloads of computer generated wallpapers. To aid comparison of different detection metrics, it was considered more appropriate to use synthetic images devoid of any pattern noise due to camera processing, rather than using actual camera images. Since the

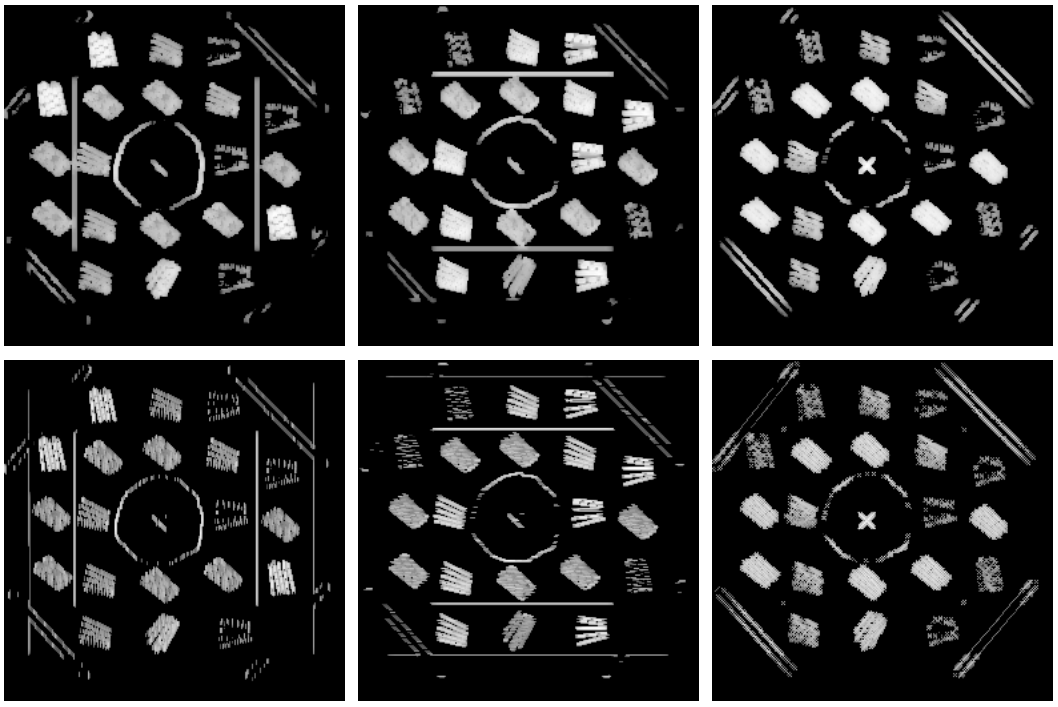


FIGURE 3.3: *Variance images ($\boldsymbol{\sigma}_0$). Top row: without spatially adaptive windows ($\mathbf{w}_2$). Bottom row: with spatially adaptive windows ($\tilde{\mathbf{w}}_2$). First column: HL 1, middle column: LH 1, last column: HH 1. Using spatially adaptive windows increases the quality of the variance estimation; more nuanced details can be observed in the bottom row than in the top row. For illustration purposes, a logarithmic scale of magnitude is used.*

PRNU will never be anything but an estimate — although hopefully getting stronger as more images contribute to the estimate — it would be unfeasible to make accurate assessments of the performance of different estimation methods, when the quantity being estimated is not fully known. For that reason a multiplicative noise was superimposed onto all images thus:

$$\mathbf{i}' = \mathbf{i} \times (\mathbf{1} + \alpha \mathbf{r}(K)), \tag{3.3}$$

where $\mathbf{i}$ is the original synthetic image, $\mathbf{r}(K)$ is a random noise pattern with seed K, used in lieu of an actual PRNU, α is the embedding strength, and $\mathbf{i}'$ is the resulting image.

The images were converted to grayscale for sake of convenience, the three colour channels (RGB, for red, green, and blue) could have been handled individually, estimating in practise three PRNUs. It should be noted that some of the synthetic images had in common text or other distinguishing features, such as the website URL. In order to minimise the effect of such recurring symbols the images were cropped to a fixed size, 512×512 pixels, but which sections of the image were cropped was randomised.

3.2.2 Performance Metrics

3.2.2.1 Quality of PRNU Estimations

As the PRNU is fully known in the experiments — it is the signal $\mathbf{r}(K)$ embedded with strength α — it is easy to assess qualitatively the estimated PRNU, $\mathbf{w}$, by computing the correlation coefficient,

$$\mathrm{cc}(\mathbf{r}(K), \mathbf{w}). \tag{3.4}$$

A value of 1 represents perfect correspondence, so values close to 1 are preferred over smaller values.

3.2.2.2 Speed of Convergence

Having a high correlation score between the estimated and real PRNU is desirable, but it cannot be the only goal. For instance, an estimation method that has a near perfect correlation with the real PRNU but only by using millions of images for the estimation, is of little use in practical situations. A method that achieves a lower — but acceptable in some sense — correlation score and needs only a handful of images to achieve it, should be preferred over the former method discussed. The ideal case would be to have an estimate that gains high correlation score by using as few images as possible.

To evaluate the connection between the correlation scores and the number of images required to achieve an acceptable score, the following experiment was conducted. For each image added to the PRNU estimation, the correlation between the estimated PRNU, $\mathbf{w}$, and the embedded signal, $\mathbf{r}(K)$, was recorded and a plot constructed of the correlation scores vs. number of images. This plot is called the *Speed of Convergence*. To limit the fluctuations in the plot, which were mainly due to statistical anomalies, the experiment was repeated and average values plotted. These plots can be compared for different PRNU estimation methods. For instance, some acceptable correlation score can be decided, say 0.8, and the PRNU estimator that reaches that score by using fewer images than the other ones is deemed preferable. For good estimators, the Speed of Convergence curve should rise sharply in the beginning for each image added to the estimate, and later level off, asymptotically approaching some correlation score, hopefully close to 1.

3.3 Experimental Results

Experiments were carried out for a large number of different embedding strengths, α, ranging from 0.001 to 0.1. Strengths above, say, $\alpha \geq 0.1$ leave noticeable artifacts, detectable to the naked eye. Embedding strengths below $\alpha \leq 0.001$ are not only undetectable by humans, but also to a great extent by our detection methods. For brevity, only the case when $\alpha = 0.05$ is introduced, chosen as a plausible compromise and a good substitute for a camera PRNU.

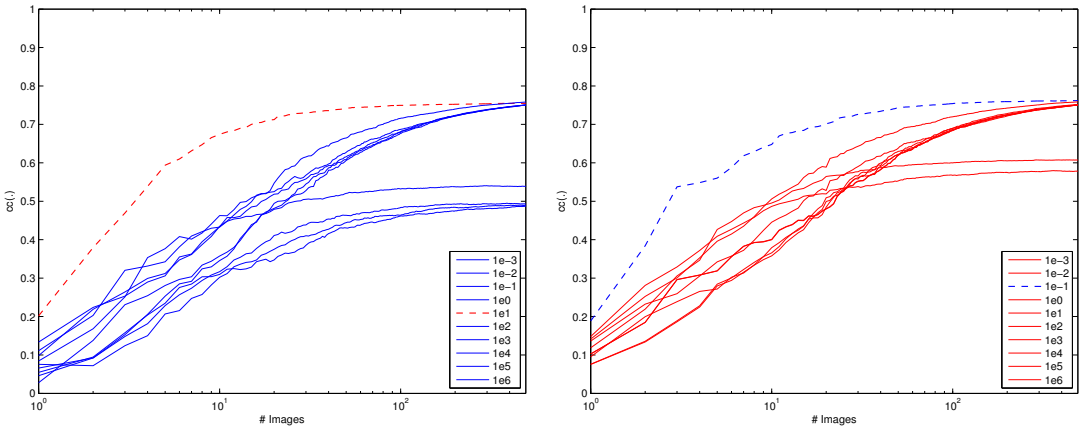
3.3.1 Determining σ_0

For different values of σ_0 the Speed of Convergence was plotted, and the optimal σ_0 selected and used throughout. This was done separately for $\mathbf{w}_2$ and $\mathbf{w}_3$. Each experiment was repeated 10 times to obtain smoother curves.

Figure 3.4, depicting the Speed of Convergence for $\mathbf{w}_2$ and $\mathbf{w}_3$, demonstrates an obvious optimal value for σ_0, both for achieving the highest correlation score, and also in the shortest time, i.e. using the fewest number of images. Based on the preliminary experiments, $\sigma_0 = 10$ for $\mathbf{w}_2$, and $\sigma_0 = 0.1$ for $\mathbf{w}_3$ were chosen. The optimal values for σ_0 are determined to the nearest degree of magnitude. A further search could be performed for values $\sigma_0 \in \langle 1; 100 \rangle$ for $\mathbf{w}_2$, and in the range $\sigma_0 \in \langle 0.01; 1 \rangle$ for $\mathbf{w}_3$. In the following study, using the rough optimal estimates was deemed satisfactory.

The experimental results, depicted in Figure 3.4, stress the importance of selecting σ_0 intelligently. Some values of σ_0 give very poor performance, even when using up to 500 images to estimate the PRNU. The correlation scores for $\sigma_0 \leq 0.1$ for $\mathbf{w}_2$ do not surpass 0.5, while the optimal σ_0 achieves a correlation score of 0.75, a performance boost of 50%. In the case of $\mathbf{w}_3$, the same tendency can be observed. Values of σ_0 lower than the optimal of $\sigma_0 = 0.1$ lead to suboptimal correlation values. Referring to Section 2.2.3, it can be concluded that if the optimal value for σ_0 is unknown, overestimating its value is more desirable than underestimating it. Therefore, images should be denoised liberally, rather than conservatively, if the optimal value cannot easily be found.

The plots suggest using up to 50 images for training purposes if the optimal value for σ_0 is known; using more than 50 images adds little to the quality of the PRNU estimation.



(a) (b)

FIGURE 3.4: *Plots of Speed of Convergence for* $\mathbf{w}_2$*, (a), and* $\mathbf{w}_3$*, (b), for different values of* σ_0*, ranging from* 10^{-3} *to* 10^6*. The optimal value is* $\sigma_0 = 10$ *and* $\sigma_0 = 0.1$ *for* $\mathbf{w}_2$ *and* $\mathbf{w}_3$*, respectively, shown as dashed curves in the plots. Embedding strength was selected as* $\alpha = 0.05$*. The horizontal axis is on a logarithmic scale.*

Experiments for $\tilde{\mathbf{w}}_2$ and $\tilde{\mathbf{w}}_3$ revealed the same optimal values for σ_0, as for $\mathbf{w}_2$ and $\mathbf{w}_3$, so those plots are not reproduced here.

3.3.2 Comparing Different PRNU Estimators

Figure 3.5 depicts the Speed of Convergence for $\mathbf{w}_2$, $\mathbf{w}_3$, $\tilde{\mathbf{w}}_2$, and $\tilde{\mathbf{w}}_3$. Although the difference is not substantial, inspection of the plot allows some classification of the different performances. It can be seen that $\mathbf{w}_3$ consistently gives better performance than $\mathbf{w}_2$, as it reaches higher correlation scores. This confirms the hypothesis that pixel dependent variance estimation surpasses the fixed variance estimation. At the start of the estimation process, $\mathbf{w}_3$ gives performance increase (measured in higher correlation score) of up to 50% compared to $\mathbf{w}_2$. However, the gap quickly narrows. At the 50 image mark, which is a rough guide to how many images should be used for a PRNU estimation, the performance gain has dropped to just over 3%.

Even with such moderate gains, the result does give reason for optimism that more drastic changes in the PRNU estimation method would achieve better results. Therefore, an interesting future research topic could be delving into other estimation methods, that take advantage of the multiplicative nature of PRNU, instead of using methods that are more suited for isolating additive noise. In fact, Amerini et al. [7] recently conducted a study comparing a Minimum Mean-squared Error Estimator (MMSEE) filter and the

denoising filter described in Section 2.2.1. That study, and hopefully others to come, demonstrates a promising avenue of research.

Shifting focus to the difference between $\mathbf{w}$ and $\tilde{\mathbf{w}}$, it is surprising that spatially adaptive filters seem to perform worse than the conventional crude method of having fixed filters for all sub-bands. By comparing the dashed lines, $\tilde{\mathbf{w}}$, to the solid lines, $\mathbf{w}$, in Figure 3.5 it can be seen that directional filters produce higher correlation scores when only a handful of images are used for the PRNU estimation. That quickly changes; as soon as the number of images exceeds 5–8 images, the conventional method achieves better results. Given that current estimation methods require dozens of images to gain a close to optimal PRNU, it is unlikely that directional filters will be used in practise in the near future. It could be speculated that although spatially adaptive windows allow better edge detection, as demonstrated in Figure 3.3, the overall variance estimation is lacking, i.e. the denoising in uniform areas is not as good. As natural images usually have more uniform areas than edges, this could explain the dip in performance.

In principal, directional filters should aid performance and further research should be carried out before this method is discarded. At present, spatially adaptive windows do not appear likely to contribute to better PRNU estimations. It cannot be ruled out, however, that enhancements to the possible deficiencies witnessed in this experiment will be forthcoming.

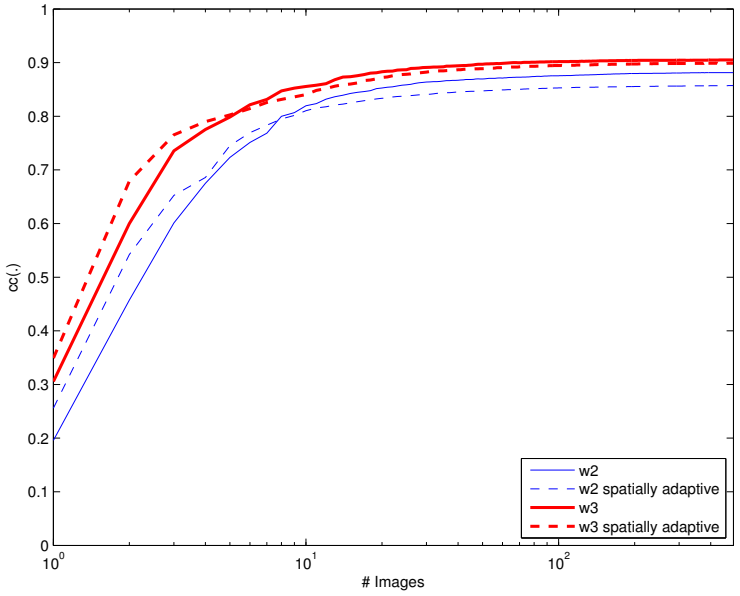


FIGURE 3.5: *Speed of Convergence for* $\mathbf{w}_2$ *(solid),* $\tilde{\mathbf{w}}_2$ *(dashed),* $\mathbf{w}_3$ *(solid bold), and* $\tilde{\mathbf{w}}_3$ *(dashed bold). The* σ_0 *values are those determined in Section 3.3.1. Embedding strength was selected as* $\alpha = 0.05$*. The horizontal axis is on a logarithmic scale.*

Chapter 4

Assessing Different Detection Metrics

4.1 Detecting Signals

The literature suggests several tools to aid detection of a signal embedded in another content, in this instance whether a PRNU, $\mathbf{w}$, is present in an image, $\mathbf{i}$. This includes the rudimentary method of linear correlation, lc(.) [12], and the correlation coefficient, cc(.), introduced in Section 2.3. A few more detection metrics are introduced and the best suited for PRNU detection is sought.

4.1.1 Pre-Whitening

Depovere et al. [13] pointed out that correlation-based detection is optimal when working on additive white Gaussian noise (AWGN). Natural images rarely have a white power spectrum; therefore the authors suggested applying a so-called whitening filter on the images before correlation. Instead of using the image itself in the correlation calculation with the PRNU, the image is pre-whitened, i.e. the denoised version of the image is subtracted from it. This reduces interference of the correlation due to image content, and transforms the non-white input into a signal with a constant power spectrum. That

should enable the correlation to function more effectively. This is in effect a generalisation of the work by Fridrich mentioned in Section 2.3 and can be used in other correlation methods than the correlation coefficient mentioned there.

4.1.2 Optimised Detectors for Multiplicative Noise

Furon [23] introduced yet another variant on correlation detection. The derivation is quite involved, but the highlights are mentioned here. The object, as before, is to detect the presence of PRNU, $\mathbf{w}$, in an image, $\mathbf{i}$, when $\mathbf{i} = \mathbf{i}_{\text{orig}} \times (\mathbf{1} + \alpha\mathbf{w})$. An optimal detector, according to Furon is

$$\text{fr}(\mathbf{i}, \mathbf{w}) = -\kappa \frac{\nabla p_{\mathbf{i}_{\text{orig}}}(\mathbf{i})}{p_{\mathbf{i}_{\text{orig}}}(\mathbf{i})} \times \mathbf{i} \times \mathbf{w}, \tag{4.1}$$

where $p_{\mathbf{i}_{\text{orig}}(.)}$ is the probability density function (PDF) of the original image, before the PRNU embedding. Assuming independent samples, and ignoring the constant factor $-\kappa$, this can be expressed as the differentiation

$$\text{fr}(\mathbf{i}, \mathbf{w}) = \sum \frac{p'_{\mathbf{i}_{\text{orig}}}(\mathbf{i})}{p_{\mathbf{i}_{\text{orig}}}(\mathbf{i})} \times \mathbf{i} \times \mathbf{w}, \tag{4.2}$$

where the sum runs over the entire pixel indices's set. With the (not entirely valid) assumption that the original image is a Gaussian signal, this simplifies to

$$\text{fr}(\mathbf{i}, \mathbf{w}) = \sum \mathbf{i} \times \mathbf{i} \times \mathbf{w} = (\mathbf{i} \times \mathbf{i}) \cdot \mathbf{w}. \tag{4.3}$$

As before, $\times$ is the element-wise multiplication, while $\cdot$ is the dot product.

4.1.3 Detection Metrics, Recap

So far, three correlation methods have been introduced, and with the pre-whitening technique, a set of 8 different combinations are devised, and listed below.

$$\text{lc}(\mathbf{i}, \mathbf{w}) = \mathbf{i} \cdot \mathbf{w} \tag{4.4}$$

$$\text{lcPw}(\mathbf{i}, \mathbf{w}) = \mathbf{n}_{\mathbf{i}} \cdot \mathbf{w} \tag{4.5}$$

$$\text{fr}(\mathbf{i}, \mathbf{w}) = (\mathbf{i} \times \mathbf{i}) \cdot \mathbf{w} \tag{4.6}$$

$$\text{frPw}(\mathbf{i}, \mathbf{w}) = (\mathbf{n}_{\mathbf{i}} \times \mathbf{i}) \cdot \mathbf{w} \tag{4.7}$$

$$\mathrm{cc}(\mathbf{i}, \mathbf{w}) = \frac{(\mathbf{i} - \bar{\mathbf{i}}) \cdot (\mathbf{w} - \overline{\mathbf{w}})}{|\mathbf{i} - \bar{\mathbf{i}}| \, |\mathbf{w} - \overline{\mathbf{w}}|} \tag{4.8}$$

$$\mathrm{ccPw}(\mathbf{i}, \mathbf{w}) = \mathrm{cc}(\mathbf{n_i}, \mathbf{w}) \tag{4.9}$$

$$\mathrm{ccFr}(\mathbf{i}, \mathbf{w}) = \mathrm{cc}((\mathbf{i} \times \mathbf{i}), \mathbf{w}) \tag{4.10}$$

$$\mathrm{ccFrPw}(\mathbf{i}, \mathbf{w}) = \mathrm{cc}((\mathbf{n_i} \times \mathbf{i})), \mathbf{w}) \tag{4.11}$$

Equation (4.9) has already been introduced in Section 2.3 as Equation (2.8).

4.2 Performance Metrics

4.2.1 ROC Curves

Plots of true-positive rate (TPR) vs. false-positive rate (FPR) for different threshold values, τ, are called Receiver Operating Characteristics (ROC) Curves [19]. The shape of an ROC curve gives an indication of the detector's success rate. A diagonal line, FPR(.) = TPR(.), represents a method that randomly guesses whether each image is from the camera or not. An unsuccessful method, that consistently fails to identify the paternity of images, would have an ROC curve in the lower right-hand side of the plot. A successful method's ROC curve would be in the upper left-hand side. An example that corresponds to a moderately successful method is illustrated in Figure 4.1.

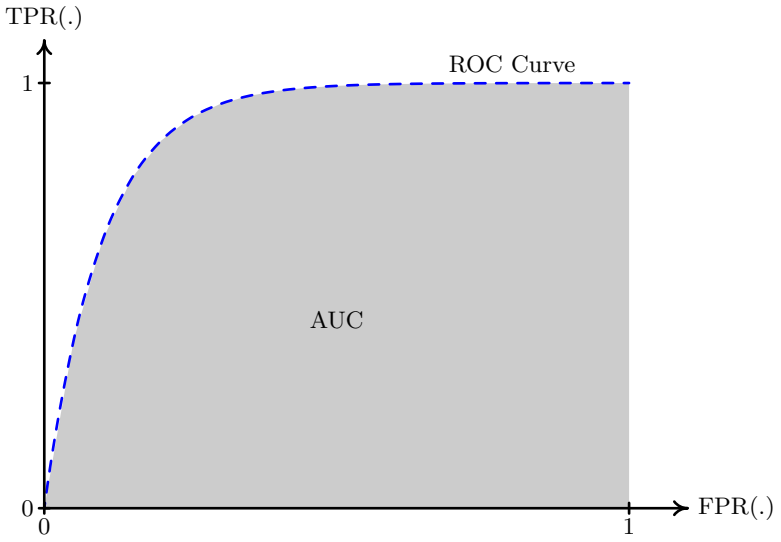


FIGURE 4.1: *Illustration of an ROC curve, dashed line, and AUC, the area under the ROC curve. TPR and FPR stand for true-positive rate, and false-positive rate, respectively.*

The upper left corner of the plot, $(0, 1)$, represents a perfect performance, as its true-positive rate is 100% with no false-positives. The lower right corner, $(1, 0)$, represents the exact opposite, all classifications are incorrect. The lower left corner, $(0, 0)$, represents a very pessimistic classifier that has no false-positives, but nor does it have any true-positives, i.e. it rejects all images. The upper right corner, $(1, 1)$, represents a classifier that rejects no image. By calculating the TPR and FPR for different values of τ, varying from 0 to 1, an ROC curve can be plotted. It will trace a curve from $(0, 0)$ to $(1, 1)$. The closer that curves comes to the $(0, 1)$ point, the better the estimator is.

4.2.2 Area Under ROC Curves, AUC

An easy way to estimate how close the ROC curve comes to the $(0, 1)$ point is to calculate the area under the ROC curve. This value is called AUC. The higher the AUC, the more accurate the detector is. A value of 1 would mean a perfect detector, whereas a value below 0.5 means that the detector performs worse than random guessing.

4.3 Experimental Results

Images in the dataset were embedded with a PRNU. Each time a different embedding strength was used, ranging from $\alpha = 0.002$ to $\alpha = 0.05$. For each of the detection metric, the existence of the embedded PRNU was investigated by calculating the correlation score between the estimated PRNU and the real PRNU that was embedded. ROC Curves were plotted and AUC values calculated. The AUC values for all eight detection metrics for various embedding strength, are depicted in Figure 4.2, for easy comparison.

Substantially better detection scores were achieved when using pre-whitening. There is a performance boost of up to 40% when using a detection metric with pre-whitening as compared to the same detection metric without pre-whitening. The single best detection method, for any PRNU embedding strength, appears to be ccFrPw(.), a method that combines the correlation coefficient, cc(.), the detector developed by Furon, fr(.), and pre-whitening. Second best are ccPw(.), and frPw(.), where the latter gives better results for low embedding strengths, whereas the former seems to have a small performance lead for higher embedding strengths. Of the methods explored in this Chapter, the linear correlation, lc(.), is least suited for PRNU detection. On the other hand, coupled

with the pre-whitening technique, linear correlation (i.e. lcPw(.)) is actually a viable contender, and for strong embedding it actually performs marginally better than frPw(.). This is potentially useful given the simplicity of the calculations involved, compared to some of the other techniques.

The dominance of PRNU detection metrics using pre-whitening can be intuitively understood. Removing the image content (as much as possible) before calculating the correlation with an entity that is essentially noise and image independent, is almost bound to give better results than the alternative.

It is noteworthy that ccFrPw(.) shows consistently better performance of PRNU detection than ccPw(.), which hitherto has been thought of as the most effective detection metric. The AUC value is increased by almost 10% for some embedding strenghts, suggesting current methods should be modified to reflect this new result. Concurrently, researchers should continue to pursue the most effective detection metric — be it correlation based or otherwise — that is tailored to multiplicative content, and specifically PRNUs.

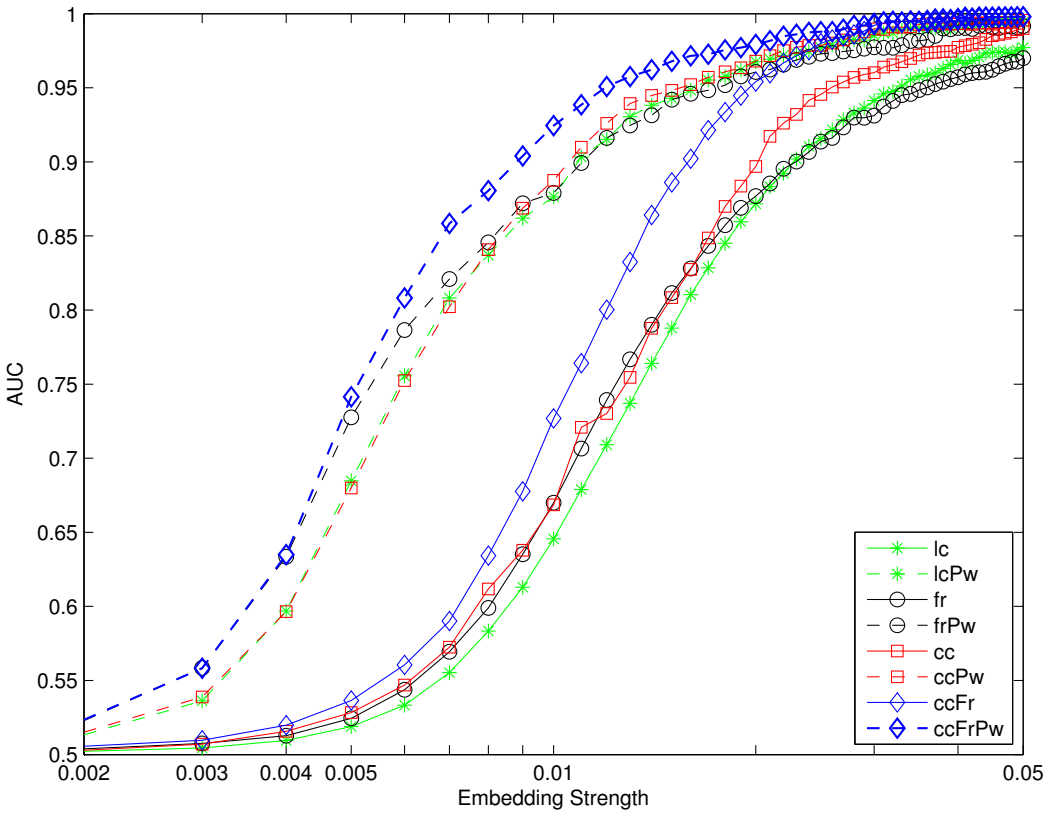


FIGURE 4.2: *The advantage of using pre-whitening (dashed curves) is self-evident from this plot. The apparent optimal detection metric is* ccFrPw(.)*, as it reaches AUC scores equal to or higher than those of other detection metrics.*

Chapter 5

PRNU Forensics in Real Life

5.1 Dealing With Adversaries

Like thieves that wear gloves or wipe their fingerprints to decrease the risk of getting caught, adversaries of digital image forensics want to remove any traces on photographs that can link them together. Child pornographers are anxious for authorities not to be able to prove they manufactured indecent material. Not only is the destruction of evidence a worrisome matter, fabrication of evidence is ethically a trickier issue. English jurist William Blackstone is quoted as saying: "Better that ten guilty persons escape than that one innocent suffer" [9]. If PRNU detection reaches such advanced stages that it will become admissible in court, false incrimination has to be duly considered. Recently it has been revealed that DNA evidence not only can be planted at crime scenes, but manufactured in a lab [35]. This is therefore a wider issue, not limited to PRNU forensics, and thus difficult to prevent always leaving more than a shred of doubt in all forensic evidences.

In the realm of PRNU forensics, two methods of malicious attacks are of interest: (i) intentionally removing the pattern noise to prevent identification, and (ii) extracting the PRNU and adding it to another image to "incriminate an innocent" camera [33]. The former can be achieved by an *informed* adversary — someone who has access to the camera or large amount of images from the camera — by establishing his own reference pattern and then subtracting it from the image. Even an *uninformed* adversary, i.e. someone without access to the camera, can prevent detection by desynchronising (e.g.

rotating slightly) the image and hence making correlation with the reference pattern more difficult [33]. PRNU insertion can be done given access to the camera, or enough images captured by that camera to have a good estimate of the PRNU. Uninformed adversaries will, however, have grave difficulties carrying out this attack. The adversary cannot insert a signal that he has no estimation of, and therefore he needs access to the camera or enough images captured by it. Overall, having access to the camera — or images from it — an adversary can conduct malicious attacks [32], and highly skillful adversaries can use signal processing methods to manipulate identification methods without any such access [33].

5.1.1 PRNU Removal

As before, images in the database were embedded with a PRNU, $\mathbf{r}(K)$, with embedding strength α. The object of the experiment was to remove traces of the PRNU from the images. Recalling Figure 2.5, the adversary's goal is to move the right PDF so it is indistinguishable from the left PDF, i.e. with mean 0, and as low variance as possible. The adversary does not have access to the real PRNU that was embedded, and therefore he has to estimate it in order to assess whether he has succeeded in removing enough traces of the PRNU to avert detection. For the purpose of this exercise, the estimation method chosen was $\mathbf{w}_3$, and the detection method was ccFrPw. Those have already been demonstrated in Section 3.3.2 and Section 4.3, respectively, to give the best results. The embedding strength was selected as $\alpha = 0.05$.

The following assumptions were made: the adversary has knowledge of the detection method, which in this case is ccFrPw, he has access to enough images to estimate the PRNU, and finally, he wishes to remove the PRNU from the images that he used to estimate the PRNU. It could be interesting to assume he uses only a part of the images to estimate the PRNU, and then removes it from the images that were not used for the estimation. This is a natural extension of the work presented here.

An image, $\mathbf{i}$, with an estimated PRNU, $\mathbf{w}$, has a correlation score $\text{ccFrPw}(\mathbf{i}, \mathbf{w}) = \rho$, i.e.

$$\frac{(\mathbf{n_i} \times \mathbf{i}) \cdot \mathbf{w}}{|\mathbf{n_i} \times \mathbf{i}| \ |\mathbf{w}|} = \rho. \tag{5.1}$$

The adversary wishes to remove that PRNU, that is to say create an image $\mathbf{j}$ where $\text{ccFrPw}(\mathbf{j}, \mathbf{w}) = 0$.

$$\mathbf{j} = \mathbf{i} \times (\mathbf{1} - \beta\mathbf{w}) \tag{5.2}$$

The selection of β, the estimated embedding strength of the PRNU, is crucial for successful removal. In an attempt to derive it, firstly, the correlation score $\text{ccFrPw}(\mathbf{j}, \mathbf{w})$ is calculated thus

$$\frac{(\mathbf{n_j} \times \mathbf{j}) \cdot \mathbf{w}}{|\mathbf{n_j} \times \mathbf{j}|\ |\mathbf{w}|}. \tag{5.3}$$

Using that $\mathbf{n_j} = \mathbf{j} - \mathbf{j}_{\text{den}}$, gives

$$\frac{((\mathbf{j} - \mathbf{j}_{\text{den}}) \times \mathbf{j}) \cdot \mathbf{w}}{|(\mathbf{j} - \mathbf{j}_{\text{den}}) \times \mathbf{j}|\ |\mathbf{w}|}, \tag{5.4}$$

and substituting Equation (5.2) gives

$$\frac{((\mathbf{i} - \mathbf{i}_{\text{den}} - \beta\mathbf{w} \times \mathbf{i}) \times \mathbf{i}) \cdot \mathbf{w}}{|(\mathbf{i} - \mathbf{i}_{\text{den}} - \beta\mathbf{w} \times \mathbf{i}) \times \mathbf{i}|\ |\mathbf{w}|}, \tag{5.5}$$

if the simplifying assumption is made that

$$\mathbf{j}_{\text{den}} \approx \mathbf{i}_{\text{den}}. \tag{5.6}$$

As the only difference between $\mathbf{i}_{\text{den}}$ and $\mathbf{j}_{\text{den}}$ is the multiplicative noise term $\beta\mathbf{w}$, that difference can be safely assumed to be mostly absent in the denoised versions. The following is a simple rearrangements of terms, and using ρ isolated from Equation (5.1):

$$\frac{(\mathbf{n_i} \times \mathbf{i}) \cdot \mathbf{w} - \beta(\mathbf{i} \times \mathbf{i} \times \mathbf{w}) \cdot \mathbf{w}}{|\mathbf{n_i} \times \mathbf{i} - \beta\mathbf{w} \times \mathbf{i} \times \mathbf{i}|\ |\mathbf{w}|}, \tag{5.7}$$

$$\frac{\rho|\mathbf{n_i} \times \mathbf{i}| \times |\mathbf{w}| - \beta(\mathbf{i} \times \mathbf{i} \times \mathbf{w}) \cdot \mathbf{w}}{|\mathbf{n_i} \times \mathbf{i} - \beta\mathbf{w} \times \mathbf{i} \times \mathbf{i}|\ |\mathbf{w}|}. \tag{5.8}$$

The adversary desires the above expression to be zero, so from the numerator, he can solve for β, thus

$$\beta = \frac{\rho|\mathbf{n_i} \times \mathbf{i}| \times |\mathbf{w}|}{(\mathbf{i} \times \mathbf{i} \times \mathbf{w}) \cdot \mathbf{w}}. \tag{5.9}$$

Disclaimer

At the time of writing, several flaws were noticed in the derivation. One of them being the hidden assumption $\mathbf{i} \approx \mathbf{j}$ used in the step between Equation (5.4) and Equation (5.5). This assertion is harder to justify than the assumption that the denoised versions are approximately equal. Without that assumption, however, the solutions of β are expressed by a 2. order equation, therefore giving two solutions, which are sometimes imaginary. The β value could be selected by examining which value left less visual artifacts of image tampering.

Another flaw in the reasoning is the omission in Equation (5.1) of the mean value terms. An observant reader would realise that the equation should read:

$$\frac{(\mathbf{n_i} \times \mathbf{i} - \overline{\mathbf{n_i} \times \mathbf{i}}) \cdot (\mathbf{w} - \overline{\mathbf{w}})}{|\mathbf{n_i} \times \mathbf{i} - \overline{\mathbf{n_i} \times \mathbf{i}}| \; |\mathbf{w} - \overline{\mathbf{w}}|} = \rho.$$

The mean terms can only be ignored if both $\mathbf{w}$ and $\mathbf{n_i} \times \mathbf{i}$ have zero mean. That can not generally be the case.

Although those revelations significantly complicate the derivations needed, nevertheless the results presented are the ones obtained with the simple, but erroneous method. It would be interesting to investigate whether the relatively positive results are a coincidence or whether a successful attack has accidentally been discovered. At any rate, this field is left for future refinements on the attack methods, that there sadly has not been time to investigate further at such a late stage in the writing.

5.1.2 Experimental Results

After estimating the PRNU, the embedding strength, β, and removing the PRNU from each image, the PDFs before and after attack are shown superimposed in the same plot (Figure 5.1). The PDF of the attacked images shows a near zero mean and a much smaller variance than before the attack. This demonstrates the effectiveness of the attack. There are some outliers, a few images who's correlation magnitude is great. This might suggest that the attack strategy is too brutal, and could be refined. None the less, the experiment is successful in shifting the PDF so that it has a much smaller mean and variance than before the attack. It is almost indistinguishable from PDFs from other cameras.

It is briefly noted that PRNU insertion works on the same principles as PRNU removal. It may prove necessary, though, to remove an actual PRNU before inserting the artificial one, and thus be more involved than the process mentioned here.

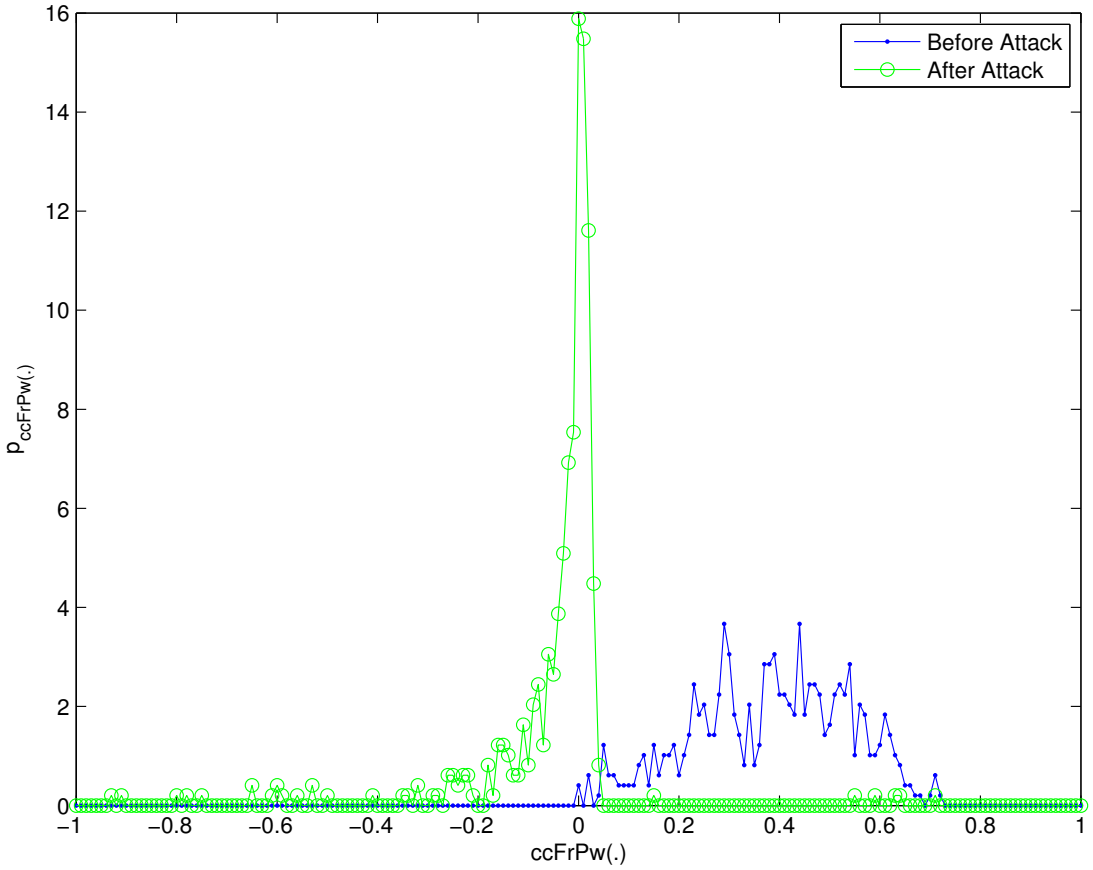


FIGURE 5.1: *The plot shows the PDF of images that are embedded with a PRNU with embedding strength $\alpha = 0.05$, dots, and after an adversary has removed an estimate of the PRNU from all images, circles.*

5.2 Multiple PRNUs

The PRNU detection method can only be considered reliable if it succeeds in identifying the paternity of images, even when presented with a database of multiple different cameras. Some large-scale tests have been conducted, but they have usually been restricted to a handful of different cameras. An experiment done by Goljan et al. [25] reached 150 different cameras. Although thorough, this number is insufficient. One could envisage a similar database of camera PRNUs as the FBI (Federal Bureau of Investigation) fingerprint database in the U.S., in which case the PRNU detection methods should scale to hundreds of thousands, or even millions of different PRNUs. Such enormous experiments are difficult to carry out, for want of enough images from such large amount of cameras. Instead, here the process is simulated by embedding a specific PRNU, $\mathbf{r}(K)$, and then comparing the images with tens of thousands of different PRNUs. Due to time

constraints, the experiment involved comparing the images with $8,000$ different PRNUs, $\mathbf{r}(K_i)$, where all K_i are different from each other and different from K. This work could be expanded beyond the $8,000$ comparisons, and using more varied embedding strengths. Three values were compared, $\alpha \in \{0.001, 0.005, 0.01\}$.

5.2.1 Experimental Results

The results are depicted in Figure 5.2. For embedding strength $\alpha = 0.01$, the AUC drops approximately 18% from 0.91 to 0.75 after being compared to $8,000$ different PRNUs. That may still be an acceptable value. However, for the other two embedding strengths, $\alpha = 0.005$, and $\alpha = 0.001$, the detection mechanism quickly becomes useless. An AUC value below 0.5 effectively means that the detector performs worse than randomly guessing. The drastic, almost exponential, drop in AUC values could very well be a factor of the image size. In the experiment, images of size 512×512 pixels were used. Having larger signals to compare could marginally improve the classification results.

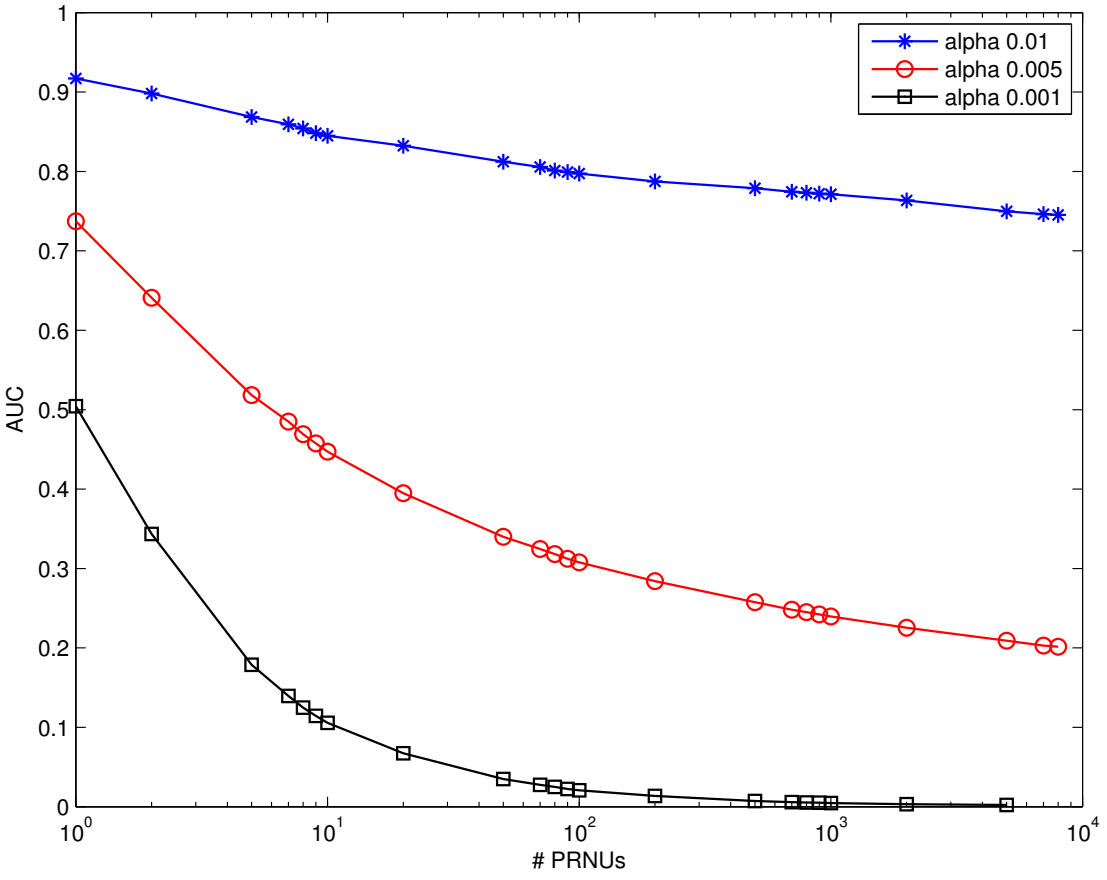


FIGURE 5.2: *AUC (Area under ROC curves) for three different embedding strengths.*

Chapter 6

Conclusions

Image forensics are becoming a hot topic. As this thesis was being finalised, a news story broke that the software company Microsoft had doctored an image on its Polish website. The image originally showed a white woman and two men, Asian and black, seated at a conference table. This version was on Microsoft's U.S. website, but the Polish version had the black man's head replaced with that of a white man. The perpetrator had not been very thorough, as the colour of the man's hands remained unchanged. Speculations abounded that Microsoft had made the change to appeal to Poland's predominantly white population. Microsoft's spokesperson quickly apologised and the picture was pulled from the website [36]. This is a tell-tale sign that photographic authenticity, integrity, and paternity will increasingly be doubted and scrutinised in the future, as manipulations get more sophisticated. PRNU-based forensics address one of these issues, as their role is to help identify which camera captured which photographs.

In Chapter 3, two new methods to aid PRNU estimation were introduced, both concerning variance estimation that is central to image denoising. Using directional filters for neighbourhood variance estimation in wavelet sub-bands, inspired by Hawwar et al. [26], did not prove successful. Further refinements are needed before the principle of using specialised filters suited to each sub-band can be brought to fruition. The other method discussed showed more promising results. This thesis supervisor, Dr. Doërr, had an idea of using pixel dependent variance estimation. It was premised on the current method's possible deficiency of using a fixed threshold for noise detection. The current method is more appropriate for additive noise, whereas a filter for multiplicative noise should

perform better. A simple modification of the Wiener filter to make it pixel variant was, therefore, devised and the results were very promising. Although the actual performance gain was moderate, the experiment showed that using denoising filters for multiplicative, rather than additive, noise can lead to more favourable results. This will undoubtedly spark new and interesting research.

After estimating PRNUs from multiple cameras, the problem of classifying which photographs originate from which cameras, is called PRNU detection. Chapter 4 was dedicated to comparing various metrics of detection, eight in total. The comparison showed a new method that outperformed the others. This method is based on the work of Furon [23] and incorporates pre-whitening techniques, generalised by Depovere et al. [13]. This method has not yet been utilised for PRNU detection, but has every potential of replacing the currently preferred method. An increase in AUC value of up to 10% was witnessed in the experiments, a sizable refinement that would provide more accurate classifications, especially for weak PRNU strengths.

Practical aspects, such as dangers of unauthorised PRNU removal or implanting, and the scalability of PRNU based systems were discussed in Chapter 5. A few large scale tests have been carried out to examine whether PRNU detection can be conducted when the number of possible cameras is large [25]. It is crucial for industrial adaptation that the detection process be able to efficiently handle up to millions of different cameras, even of the same model and make. Such experiments are hard to set up, tests have been done with some 150 cameras, which give some indication how the system will scale. A comparison with 8,000 different PRNUs was carried out in Chapter 5, albeit using randomly generated signals, instead of actual camera PRNUs, for practical reasons. There were obvious performance penalties when introducing so many PRNU candidates. Apart from time issues, which were not of particular concern, drops in AUC values were apparent and they were highly dependent on the PRNU embedding strength. Specifically, if embedding strength was lower than $\alpha < 0.005$, the detection system was rendered useless when dealing with more than 10 different PRNUs. That presents a serious concern for future development. Camera manufacturing could get refined enough that imperfections, which are currently used for PRNU estimation and detection, diminish such that PRNU strengths drop below usable values. The results are as of yet inconclusive and further tests, using actual camera data, should be carried out to investigate further.

Preliminary results were achieved with respect to removing PRNUs from images. They did show how PRNUs can successfully be removed from images and therefore hinder camera identification. However, there was an error found in the derivation and time not permitting it to be rectified satisfactorily. All assumptions about the validity of that attack are therefore deferred to another possible research in the future.

Sencar et al. [41] mention that most forensic techniques can be circumvented, even by novice manipulators, and because the factors utilised in the images are mostly imperceptible, they can easily be removed and hence prevent analysis. For that reason, camera identification and forgery detection mechanisms should be approached from multiple directions [33], combining different methods that perhaps are vulnerable to different attacks or standard processing methods. Each method will only work when specific assumptions are met, but as an attacker is hard pressed to circumvent all detection methods at once, without leaving visible artifacts, at least he can never be sure he has succeeded. And he cannot design an attack to circumvent unknown forensic tools [11, 30].

This will continue to be a cat-and-mouse game. Currently the forensics are merely catching up with the manipulators, but hopefully before too long the roles will have reversed, and forensic analysts will have the upper hand.

Bibliography

[1] *Wavelab 850* http://www-stat.stanford.edu/~wavelab/, (Accessed August 16, 2009).

[2] *Elements of sensor resolution* http://www.i-cubeinc.com/3D/3shotcm.shtml, (Accessed August 17, 2009).

[3] *The JPEG committee home page* http://www.jpeg.org/, (Accessed August 17, 2009).

[4] *Stalinist alteration of historical photographs* http://www.revleft.com/vb/stalinist-alteration-historical-t80231/index.html, (Accessed August 20, 2009).

[5] *Reuters doctoring Beirut photos.* http://mypetjawa.mu.nu/archives/184203.php, The Jawa Report, August 6, 2006. Accessed March 29, 2009.

[6] *Reuters toughens rules after altered photo affair.* http://www.reuters.com/article/latestCrisis/idUSL18678707, Reuters, January 18, 2007. Accessed August 18, 2009.

[7] Irene Amerini, Roberto Caldelli, Vito Cappellini, Francesco Picchioni, and Alessandro Piva, *Analysis of denoising filters for photo response non uniformity noise extraction in source camera identification*, 16th Int. Conference on Digital Signal Process, Santorini, Greece, 2009.

[8] Sevinc Bayram, Husrev T. Sencar, and Nasir Memon, *Classification of digital camera-models based on demosaicing artifacts*, Digital Investigation **5** (2008), no. 1-2, 49 – 59.

[9] William Blackstone, *Commentaries on the laws of England*, Printed at the Clarendon Press, Oxford, 1765.

[10] Mo Chen, Jessica Fridrich, and Miroslav Goljan, *Digital imaging sensor identification (further study)*, Proc. of SPIE Electronic Imaging, Photonics West (January 2007).

[11] Mo Chen, Jessica J. Fridrich, Miroslav Goljan, and Jan Lukáš, *Determining image origin and integrity using sensor noise*, IEEE Transactions on Information Forensics and Security **3** (2008), no. 1, 74–90.

[12] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker, *Digital watermarking and steganography*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[13] Geert Depovere, Ton Kalker, and Jean-Paul Linnartz, *Improved watermark detection reliability using filtering before correlation*, **1** (1998).

[14] Hany Farid, *Digital doctoring: How to tell the real from the fake*, Significance **3** (2006), no. 4, 162–166.

[15] ———, *Digital image forensics*, Scientific American **6** (2008), no. 298, 66–71.

[16] ———, *Deception: Methods, motives, contexts and consequences*, ch. Digital Doctoring: can we trust photographs?, pp. 95–108, Stanford University Press, 2009.

[17] ———, *A survey of image forgery detection*, IEEE Signal Processing Magazine, 2009 (in press).

[18] ———, *Photo tampering throughout history* `http://www.cs.dartmouth.edu/farid/research/digitaltampering/`, (Accessed March 27, 2009).

[19] Tom Fawcett, *ROC graphs: Notes and practical considerations for data mining researchers*, 2003.

[20] Tomáš Filler, Jessica Fridrich, and Miroslav Goljan, *Using sensor pattern noise for camera model identification*, ICIP, 2008, pp. 1296–1299.

[21] Jessica Fridrich, *Digital image forensic using sensor noise*, IEEE Signal Processing Magazine **6** (March 2009), no. 2, 26–37.

[22] Jessica Fridrich, D. Soukal, and J. Lukáš, *Detection of copy-move forgery in digital images*, (2003).

[23] Teddy Furon, *A constructive and unifying framework for zero-bit watermarking*, IEEE transactions on information forensics and security **2** (2007), no. 2.

[24] Thomas Gloe, Matthias Kirchner, Antje Winkler, and Rainer Böhme, *Can we trust digital image forensics?*, Multimedia '07: Proceedings of the 15th international conference on Multimedia (New York, NY, USA), ACM, 2007, pp. 78–86.

[25] Miroslav Goljan, Jessica Fridrich, and Tomáš Filler, *Large scale test of sensor fingerprint camera identification*, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 7254, feb 2009.

[26] Yousef Hawwar and Ali Reza, *Spatially adaptive multiplicative noise image denoising technique*, IEEE Transactions on Image Processing **11** (Dec. 2002), no. 12, 1397–1404.

[27] Micah K. Johnson and Hany Farid, *Exposing digital forgeries by detecting inconsistencies in lighting*, MM&Sec '05: Proceedings of the 7th workshop on Multimedia and security (New York, NY, USA), ACM, 2005, pp. 1–10.

[28] Micah K. Johnson and Hany Farid, *Exposing digital forgeries in complex lighting environments*, IEEE Transactions on Information Forensics and Security **3** (2007), no. 2, 450–461.

[29] Donald Kennedy, *Editorial retraction*, Science (2006), Science 211(5759), 335.

[30] Matthias Kirchner and Rainer Böhme, *Hiding traces of resampling in digital images*, Information Forensics and Security, IEEE Transactions on **3** (2008), no. 4, 582–592.

[31] Tran Van Lanh, Kai-Sen Chong, Sabu Emmanuel, and Mohan S. Kankanhalli, *A survey on digital camera image forensic methods*, ICME, IEEE, 2007, pp. 16–19.

[32] Jan Lukáš, Jessica J. Fridrich, and Miroslav Goljan, *Digital bullet scratches for images*, ICIP, 2005, pp. III: 65–68.

[33] ———, *Digital camera identification from sensor pattern noise*, IEEE Transactions on Information Forensics and Security **1** (2006), no. 2, 205–214.

[34] M. Kivan Mihçak, Igor Kozintsev, and Kannan Ramchandran, *Spatially adaptive statistical modeling of wavelet image coefficients and its application to denoising*, Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, Phoenix, AZ **6** (Mar. 1999), 3253–3256.

[35] The New York Times, *DNA evidence can be fabricated, scientists show.* `http://www.nytimes.com/2009/08/18/science/18dna.html?_r=2`, (Accessed August 30, 2009).

[36] BBC News, *Microsoft in web photo racism row.* `http://news.bbc.co.uk/1/hi/technology/8221896.stm`, (Accessed August 30, 2009).

[37] Tian-Tsong Ng, Shih-Fu Chang, Ching-Yung Lin, and Qibin Sun, *Passive-blind image forensics*, Multimedia Security Technologies for Digital Rights (W. Zeng, H. Yu, and and Ching-Yung Lin, eds.), Elsvier, 2006.

[38] Alin C. Popescu and Hany Farid, *Exposing digital forgeries by detecting duplicated image regions*, Tech. Report TR2004-515, Department of Computer Science, Dartmouth College, 2004.

[39] ———, *Exposing digital forgeries by detecting traces of re-sampling*, IEEE Transactions on Signal Processing **53** (2005), no. 2, 758–767.

[40] ———, *Exposing digital forgeries in color filter array interpolated images*, IEEE Transactions on Signal Processing **53** (2005), no. 10, 3948–3959.

[41] Husrev T. Sencar and Nasir Memon, *Overview of state-of-the-art in digital image forensics, part of indian statistical institute platinum jubilee monograph series titled 'statistical science and interdisciplinary research,'*, World Scientific Press, 2008.

[42] Ashwin Swaminathan, Min Wu, and K. J. Ray Liu, *Nonintrusive component forensics of visual sensors using output images*, Information Forensics and Security, IEEE Transactions on **2** (2007), no. 1, 91–106.

[43] ———, *Digital image forensics via intrinsic fingerprints.*, IEEE Transactions on Information Forensics and Security **3** (2008), no. 1, 101–117.

A Source Code

The core of the Matlab functions written for this project are listed below in alphabetical order. Many minor functions, such as for plotting, etc. are skipped.

comparePRNU.m

```
function [camera, corValues] = comparePRNU(foldername,sigma0,height,width, ...
     PRNU,trigger,spAd)
%function [camera, corValues] = comparePRNU(foldername,sigma0,height,width, ...
     PRNU,trigger,spAd)
% Calculates correlation value for each image in subdirectories of folder
% with the given PRNU.
% OUTPUT:
%   CORVALUES:      each entry is the correlation value between the noise
%                   residual of an image, and the given PRNU.
%   CAMERA:         Number of camera the corresponding corValue is from.
%
% INPUT:
%   FOLDERNAME:     String with the path to and name of folder that has
%                   camera-subdirectories.
%   PRNU:           The PRNU which we are comparing to.
%   HEIGHT, WIDTH:  Dimensions of images and PRNU. They should be the same.
%   SIGMA0:         ------
%   TRIGGER:        String, whether sigma0 should depend on image or not.
%                       w4: Yes, depends on image (W4),
%                       w3: No, doesn't depend on image(W3)
%                   stupid: Using a stupid method to denoise.
%   SPAD:           String, whether we used spatially adaptive
%                   multiplicative noise imageing
%                    false: No
%                     true: Yes
%                      log: Using log noise estimator

% Information printed to screen
disp(sprintf('\n.......ComparePrnu.......'));

% Initialize values
corValues = []; camera = [];
imgIndex = 1;
%cameraIndex = 1;

% Get list of images
cameralist = dir(foldername);

% Loop through all different cameras, count from 3 to skip folders '.' and '..'
for k=3:length(cameralist)

    % Make sure we only process directories
    camfolder = cameralist(k);
    if camfolder.isdir == 1

```

```
        % Update cameraIndex
        cameraIndex = str2num(camfolder.name(length(camfolder.name)-1:end));

        % Get content of camera folder with correct dimensions
        folder = strcat(foldername,'/',cameralist(k).name, ...
          sprintf('/%dx%d',height,width));
        pictures = dir(folder);

        % Loop through different images
        for j=1:length(pictures)

            % Process only images, not directories
            if pictures(j).isdir == 0

                % Read the image
                filename = strcat(folder,'/',pictures(j).name);
                img = readImage(filename);

                % Information printed to screen
                disp(sprintf('Compare: %d -- %s  (s:%.5f %s %s)', ...
               imgIndex,filename,sigma0,trigger,spAd));

                % Compute correlation score
                corValues(imgIndex) = detectPRNU(img,PRNU,sigma0,trigger,spAd);

                % Keep track of which camera this pic is from
                camera(imgIndex) = cameraIndex;

                % Update image counter
                imgIndex = imgIndex + 1;
            end
        end
    end
end
end
```

cropImage.m

```
function imgc = cropImage(img,row,col)
%function imgc = cropImage(img,row,col)
% Crop image to original size (row x col).

    % Size of padded image
    [M N] = size(img);

    % Get row indices
    nr = (M-row)/2;
    min_row = ceil(nr)+1; max_row = ceil(M-nr);

    % Get column indices
    nc = (N-col)/2;
    min_col = ceil(nc)+1; max_col = ceil(N-nc);

    % Crop image
    imgc = img(min_row:max_row,min_col:max_col);
end
```

denoise.m

```
function imgd = denoise(img,sigma0,trigger,spAd)
%function imgd = denoise(img,sigma0,trigger,spAd)
% Calculate the fourth-level wavelet decomposition of the image
% with 8-tap Daubechies QMF.
%
% OUTPUT:
%   IMGD:       Denoised version of img.
%
% INPUT:
%   IMG:        Image that shall be denoised (type: double, grayscale)
%   SIGMA0:     ---
```

```
%   TRIGGER:    String, whether sigma0 should depend on image or not.
%                       w4: Yes, depends on image (W4),
%                       w3: No, doesn't depend on image(W3)
%                   stupid: Using a stupid method to denoise.
%   SPAD:       String, whether we used spatially adaptive
%               multiplicative noise imageing
%                    false: No
%                     true: Yes
%                      log: Using log noise estimator

% Check that we can do the job
if size(img,3)~=1,
    error('Image not grayscale');
end

if ~(strcmpi(trigger,'w3') || strcmpi(trigger,'w4') || strcmpi(trigger,'stupid'))
    error('Trigger not correct: %s',trigger)
end

L = 4;

% Padding the image, dimensions must be multiple of 2^L
[img,prow,pcol] = padImage(img,L);

%% WAVELET TRANSFORM

% Define the wavelet filter
h = MakeONFilter('Daubechies',8);

% Compute the discrete wavelet transform
[wc,L] = mdwt(img,h,L);

%% DENOISE EACH WAVELET BAND

% Get image dimension
nRow = size(img,1);
nCol = size(img,2);

if strcmpi(trigger,'stupid')
    % == "STUPID" ==
    wc(1:nRow/2^L,1:nCol/2^L) = 0;
    disp('!!!!!! Being extremely stupid !!!!!!')
    trigger = 'w3';
end

% Loop through the wavelet bands
for b=1:L,

    % Define some indices
    r_low = nRow / 2^b;
    r_high = nRow / 2^(b-1);
    c_low = nCol / 2^b;
    c_high = nCol / 2^(b-1);

    % Extract the bands
    bandh = wc(1:r_low,c_low+1:c_high);
    bandv = wc(r_low+1:r_high,1:c_low);
    bandd = wc(r_low+1:r_high,c_low+1:c_high);

    % Check whether sigma0 should depend on the image or not.
    new_sigma = sigma0;
    if strcmpi(trigger,'w4')
        % Decrease the size of the original image so it fits the size of the band
        new_sigma = sigma0*imresize(img,size(img)*2^(-b),'nearest') + eps;
    end

    % Denoise the bands
    bandhd = denoiseBand(bandh,new_sigma,spAd,1); %1. quarter
    bandvd = denoiseBand(bandv,new_sigma,spAd,3); %3. quarter
    banddd = denoiseBand(bandd,new_sigma,spAd,4); %4. quarter
```

```
    % Push back the bands
    wc(1:r_low,c_low+1:c_high) = bandhd;
    wc(r_low+1:r_high,1:c_low) = bandvd;
    wc(r_low+1:r_high,c_low+1:c_high) = banddd;
end

%% INVERSE WAVELET TRANSFORM
[imgi,L] = midwt(wc,h,L);

% Crop to compensate padding
imgd = cropImage(imgi,prow,pcol);
end
```

denoiseBand.m

```
function h_den = denoiseBand(h,sigma0,spAd,noQuarter)
%function h_den = denoiseBand(h,sigma0,spAd,noQuarter)
% Takes as input subband of wavelet coefficients and calculates the
% denoised coefficients.
%
% OUTPUT:
%   h_den:      Denoised version of h.
%
% INPUT:
%   H:          Image that shall be denoised (type: double, grayscale)
%   SIGMA0:     ---
%   SPAD:       String, whether we used spatially adaptive
%               multiplicative noise imageing
%                    false: No
%                     true: Yes
%                      log: Using log noise estimator
%             simple: Use only 5x5 matrix
%   NOQUARTER: Which quarter? (Only relevant if SPAD is true.)
%                         1: HL (Vertical)
%                         3: LH (Horizontal)
%                         4: HH (Diagonal)

if ~(strcmpi(spAd,'false') ...
          || strcmpi(spAd,'true') ...
          || strcmpi(spAd,'log') ...
          || strcmpi(spAd,'simple') ...
          || strcmpi(spAd,'doerr'))
     error('spAd not correct: %s',spAd)
end

% Compute sigma0_sq
sigma0_sq = sigma0.^2;

% Compute the square of the wavelet band
h_sq = h.^2;

%Parameters
W = [3,5,7,9];

% Make it simpler
if strcmpi(spAd, 'simple')
     W = 5;
end

% Loop through the different window sizes
for i=1:length(W)
    m = W(i);

    if strcmpi(spAd,'true')
          % For 1. quarter: Vertical vector
        % For 3. quarter: Horizontal vector
        % For 4. quarter: Diagonal vectors
        if noQuarter == 1
          % [ 0 1 0
          %   0 1 0
          %   0 1 0 ]
```

```
            emptyspace = zeros(m,floor(m/2));
            [vec1, vec2] = deal([emptyspace, ones(m,1), emptyspace]);
        elseif noQuarter == 3
          % [ 0 0 0
          %   1 1 1
          %   0 0 0 ]
            emptyspace = zeros(floor(m/2),m);
            [vec1, vec2] = deal([emptyspace; ones(1,m); emptyspace]);
        elseif noQuarter == 4
          % [ 1 0 0          [ 0 0 1
          %   0 1 0     &      0 1 0
          %   0 0 1 ]          1 0 0 ]
            vec1 = eye(m);
            vec2 = flipdim(vec1,1);
        else
            error('noQuarter wrongly set %d',noQuarter);
        end

        sigma_W_1 = conv2(h_sq,vec1,'same') / m;
        sigma_W_2 = conv2(h_sq,vec2,'same') / m;
        sigma_W = max(sigma_W_1, sigma_W_2);
    else
        % Compute the local variance
        sigma_W = conv2(h_sq,ones(W(i)),'same') / m^2;

    end

    % Use their magic formula
    sigma_W_sq(:,:,i) = max(0, sigma_W - sigma0_sq);
end

%Compute the min along the depth
sigma_sq = min(sigma_W_sq,[],3);

% Perform denoising
h_den = h .* sigma_sq ./ (sigma_sq + sigma0_sq);
end
```

detectPRNU.m

```
function cc = detectPRNU(imag,PRNU,sigma0,trigger,spAd)
%function cc = detectPRNU(imag,PRNU,sigma0,trigger,spAd)
% Calculates correlation between noise-residual of imag and PRNU
%
% OUTPUT:
%   cc:       Correlation value.
%
% INPUT:
%   IMAG:       Image (type: double, grayscale)
%   SIGMA0:     ---
%   TRIGGER:    String, whether sigma0 should depend on image or not.
%                       w4: Yes, depends on image (W4),
%                       w3: No, doesn't depend on image(W3)
%                   stupid: Using a stupid method to denoise.
%   SPAD:       String, whether we used spatially adaptive
%               multiplicative noise imageing
%                    false: No
%                     true: Yes
%                      log: Using log noise estimator

% Compute noise residual of the image.
noiseRes = estimateNoiseResidual(imag,sigma0,trigger,spAd);

% Convert matrices to vectors and subtract mean vales.
n = noiseRes(:) - mean2(noiseRes);
p = PRNU(:) - mean2(PRNU);

% Perform the correlation calculation
cc = n' * p / (norm(n)*norm(p));
end
```

embedPRNU.m

```
function [outwstrength,PRNU] = embedPRNU(camfolder,camerano,strength, ...
                          height,width,writeBool,outputfolder)
%function PRNU = embedPRNU(camfolder,camerano,strength, ...
                          height,width,writeBool,outputfolder)
% camerano is used to initialize the state of random generator
% that is used to create the prnu.

outwstrength = sprintf('%s/%0.5g',outputfolder,strength);
outp = sprintf('%s/cam%02d/%dx%d',outwstrength,camerano,height,width);

%create output folder if not already present
if exist(outp) ~= 7
    mkdir(outp)
    system(sprintf('ln -s ~/matlab/picCGI-withoutPRNUALL/cam71/ %s/cam99', ...
                outwstrength));
end


campath = sprintf('%s/%dx%d/',camfolder,height,width);
d=dir(campath);

% initilize random generator so we get a specific PRNU
randn('state',camerano);
PRNU = randn(height,width);

% loop through all images
for i=3:length(d)
    % read the image
    im = double(rgb2gray(imread(strcat(campath,d(i).name))));

    %embed multiplicative noise
    im_p = uint8(im.*(1+strength*PRNU));

    % write to file
    if writeBool
        outfile = sprintf('%s/img%03d.jpg',outp,i-2);
        imwrite(im_p,outfile,'jpg')
     disp(sprintf('Writing file %s',outfile))
    end
end
end
```

estimateNoiseResidual.m

```
function W = estimateNoiseResidual(I,sigma0,trigger,spAd)
%function W = estimateNoiseResidual(I,sigma0,trigger,spAd)
%
% OUTPUT:
%   W:          Noise residual of I.
%
% INPUT:
%   I:          Image that shall be denoised (type: double, grayscale)
%   SIGMA0:     ---
%   TRIGGER:    String, whether sigma0 should depend on image or not.
%                       w4: Yes, depends on image (W4),
%                       w3: No, doesn't depend on image(W3)
%                   stupid: Using a stupid method to denoise.
%   SPAD:       String, whether we used spatially adaptive
%               multiplicative noise imageing
%                    false: No
%                     true: Yes
%                      log: Using log noise estimator

% For all color channels, subtract the denoised version from the actual image
% to obtain the noise residual

for channel=1:size(I,3),
    if strcmpi(spAd,'doerr')
        W(:,:,channel) = I(:,:,channel) - multiplicativeDenoise(I(:,:,channel));
```

```
        W = min(W,255);
    else
        W(:,:,channel) = I(:,:,channel) - ...
                denoise(I(:,:,channel),sigma0,trigger,spAd);
    end

end

% if we are doing the log estimator:
if strcmpi(spAd,'log')
    W = exp(W) - 1;
end
end
```

estimatePRNU.m

```
function [W1,W2,W3] = estimatePRNU(foldername,sigma0,height,width, ...
                trigger,spAd,noPics,nbChannels)
%function [W1,W2,W3] = estimatePRNU(foldername,sigma0,height,width, ...
                trigger,spAd,noPics,nbChannels)
% OUTPUT:
%   W1:             1/N * SUM W_k
%   W2:             (SUM W_k * I_k) / (SUM (I_k)^2)
%   W3:             Same as W3, except column and row means are subtracted
%
% INPUT:
%   FOLDERNAME:     String with the path to and name of camera subdirectory,
%             such as 'pics/1_Canon_EOS_D30'
%   SIGMA0:         ------
%   HEIGHT, WIDTH:  Dimensions of images and PRNU. They should be the same.
%   TRIGGER:        String, whether sigma0 should depend on image or not.
%                       w4: Yes, depends on image (W4),
%                       w3: No, doesn't depend on image(W3)
%                   stupid: Using a stupid method to denoise.
%   SPAD:           String, whether we used spatially adaptive
%                   multiplicative noise imageing
%                    false: No
%                      true: Yes
%                       log: Using log noise estimator
%   NOPICS:        [Optional] Number of pictures used to create PRNU,
%             the first noPics will be used. If not set it defaults to all images
%             in the directory.
%   NBCHANNELS:     The number of color-channels (rgb=3, grayscale=1)

% Get the list of images
subdir = sprintf('%s/%dx%d/',foldername,height,width);
folderlist = dir(subdir);

% Set default values
if nargin < 8, nbChannels = 1; end
if nargin < 7, noPics = length(folderlist)-2; end

% Information printed to screen
disp(sprintf('\n.......Estimating %d/%d from %s.......', ...
          noPics,length(folderlist)-2,foldername));
if strcmpi(trigger,'w4'),
     disp('.......ESTIMATING W4! (varible sigma).......');
else disp('.......ESTIMATING W1, W2 & W3 (fixed sigma).......');
end

%% INITIALIZE PRNUs:

% W1 = 1/N * SUM W_k
% W2 = (SUM W_k * I_k) / (SUM (I_k)^2)
% W3 = Same as W2, except column and row means are subtracted
[W1, W2_numerator, W2_denominator, W3] = ...
                deal(zeros(height,width,nbChannels));

% Image counter
nbImg = 0;

```

```
%loop through different images, the first noPics of them
for i=3:noPics+2

    % Get everything except directories
    if folderlist(i).isdir == false

        % Read the image
        filename = strcat(subdir,folderlist(i).name);
        img = readImage(filename,spAd);

        % check dimensions
        if (size(img)==size(W1))

        % Add the current noise residual to the PRNU estimation
        curr = estimateNoiseResidual(img,sigma0,trigger,spAd);
        W1 = W1 + curr;

        W2_numerator = W2_numerator + curr .* img;
        W2_denominator = W2_denominator + img.^2;

        % Increment counter
        nbImg = nbImg + 1;

        % Information printed to screen
        disp(sprintf('EstPRNU: %d/%d (%d) -- %s   (s:%.5f %s %s)', ...
             nbImg,noPics,length(folderlist)-2,filename,sigma0, ...
             trigger,spAd))
        end
    end
end

%% FINALIZE
W1 = W1 / nbImg;
W2 = W2_numerator ./ W2_denominator;
W3 = subMeanColRow(W2);
end
```

multiplicativeDenoise.m (Contributed by Dr. Doërr)

```
function imgDen = multiplicativeDenoise(img)
% WRITTEN BY GWENAEL DOERR
%
% Perform multiplicative denoising of the input image
% Companion paper:
%   Y. Hawwar and A. Reza
%   Spatially Adaptive Multiplicative Noise Image Denoising Technique
%   IEEE Transactions on Image Processing, 11(12):1397—1404, December 2002

    % Some parameters for the wavelet transform
    L = 3;                  % number of wavelet levels
    h = daubcqf(4,'min');   % Specify the mother wavelet

    % Some parameters for the denoising routine
    n1 = 5;      % Should be an odd number
    n2 = 7;      % Should be an odd number
    gamma2 = [0.98 0.92 0.84];
    gamma1 = [0.52 0.52 0.52];
%    gamma2 = [0.98 0.92 0.84];  %Exhaustive search for good values
%    gamma1 = [0.92 0.85 0.60];
    beta = [0 0 0];
    xi = [0 0 0];

    % A bit of display
    disp('. Computing the Discrete Wavelet Transform');

    % Compute the forward wavelet transform
    [wav,L] = mdwt(img,h,L);

    % Initialization of denoised wavelet transform
    wavDen = wav;
```

```
    % For each level
    for l=1:L,
        % A bit of display
        disp(sprintf('.. Processing level %d',l));

        % Define some index
        rowStart = size(img,1) / 2^l;
        rowEnd = size(img,1) /2^(l-1);
        colStart = size(img,2) / 2^l;
        colEnd = size(img,2) / 2^(l-1);

        % A bit of display
        disp('... Frequency band extraction')

        % Extract the frequency bands
        hl = wav(1:rowStart,colStart+1:colEnd);
        lh = wav(rowStart+1:rowEnd,1:colStart);
        hh = wav(rowStart+1:rowEnd,colStart+1:colEnd);

        % A bit of display
        disp('... Frequency band processing');

        % Process each frequency band
        alpha_hl = processHLBand(hl,n1,n2,gamma1(l),gamma2(l),beta(l),xi(l));
        alpha_lh = processLHBand(lh,n1,n2,gamma1(l),gamma2(l),beta(l),xi(l));
        [alpha_hh, alpha_hl, alpha_lh] = processHHBand(lh,n1,n2, ...
                    gamma1(l),gamma2(l),beta(l),xi(l),alpha_hl,alpha_lh);

        % Perform the denoising
        hlDen = alpha_hl.*hl;
        lhDen = alpha_lh.*lh;
        hhDen = alpha_hh.*hh;

        % A bit of display
        disp('... Frequency band denoising');

        % Push back the denoised bands
        wavDen(1:rowStart,colStart+1:colEnd)        = hlDen;
        wavDen(rowStart+1:rowEnd,1:colStart)        = lhDen;
        wavDen(rowStart+1:rowEnd,colStart+1:colEnd) = hhDen;
    end

    % A bit of display
    disp('. Computing the Inverse Discrete Wavelet Transform');

    % Compute the inverse wavelet transform
    [imgDen,L] = midwt(wavDen,h,L);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function alpha = processHLBand(band,n1,n2,gamma1,gamma2,beta,xi)
% Compute variance map and then perform the F-test

    %% VARIANCE MAP COMPUTATION WITH N1x1 VERTICAL WINDOW

    % Filter half length
    halfLength = (n1-1)/2;
    % Pad the band to avoid border effects with circshift below
    bPad = padBand(band,halfLength);
    % Stack the shifted version of the band
    bStack = zeros(size(bPad,1),size(bPad,2),n1);
    for i=1:n1,
        bStack(:,:,i) = circshift(bPad,[-halfLength+i-1 0]);
    end
    % Crop temporary stack
    bStack = bStack(halfLength+1:halfLength+size(band,1), ...
          halfLength+1:halfLength+size(band,2),:);
    % Sort along the depth
    bStack = sort(bStack,3);
```

```
    % Remove the outliers
105 bStack = bStack(:,:,2:n1-1);
    % Compute the variance (biased or unbiased)
    sigma2 = var(bStack,0,3);
    % Free memory
    clear bStack bPad i;
110
    %% Rmax AND Rmin COMPUTATION

    % Filter half length
    halfLength = (n2-1)/2;
115 % Pad a bit the band
    bPad = padBand(sigma2,halfLength);
    % Stack some shifted versions of the band
    bStack = zeros(size(bPad,1),size(bPad,2),n2);
    for i=1:n2,
120     bStack(:,:,i) = circshift(bPad,[0 -halfLength+i-1]);
    end
    % Crop temporary stack
    bStack = bStack(halfLength+1:halfLength+size(sigma2,1), ...
          halfLength+1:halfLength+size(sigma2,2),:);
125 % Rmax
    Rmax = max(bStack(:,:,halfLength),bStack(:,:,halfLength+2)) ./ sigma2;
    % Compute the two medians
    med1 = median(bStack(:,:,1:halfLength),3);
    med2 = median(bStack(:,:,halfLength+2:n2),3);
130 % Rmin
    Rmin = min(med1,med2) ./ sigma2;
    % Free memory
    clear bPad bStack i med1 med2

135 % F-TEST VALUES
    r1 = 1-gamma1;
    ftest1_low = finv(r1/2,n1-2,n1-2);
    ftest1_high = finv(1-r1/2,n1-2,n1-2);
    r2 = 1-gamma2;
140 ftest2_low = finv(r2/2,n1-2,n1-2);
    ftest2_high = finv(1-r2/2,n1-2,n1-2);

    %% COMPUTE THE ATTENUATION VALUE

145 % Initialize the attenuation array
    alpha = zeros(size(band));

    % Find the features i.e. locations for which neighbour pixels do not
    % share the same variance
150 index1a = find( (Rmax<=ftest2_low) | (Rmax>=ftest2_high) );
    % Leave the locations untouched
    alpha(index1a) = 1;

    % Isolate the untouched locations
155 indexLeft = setdiff([1:prod(size(alpha))]',index1a);
    % Find the uniform variance areas
    indexBeta = find( (Rmin(indexLeft)>=ftest1_low) & ...
                      (Rmin(indexLeft)<=ftest1_high) );
    % Set the attenuation value to beta
160 alpha(indexLeft(indexBeta)) = beta;

    % Isolate the untouched locations
    indexLeft = setdiff(indexLeft,indexLeft(indexBeta));
    % Find features
165 index1b = find( (Rmin(indexLeft)<=ftest2_low) | ...
                      (Rmin(indexLeft)>=ftest2_high) );
    % Leave the locations untouched
    alpha(indexLeft(index1b)) = 1;

170 % Isolate the untouched locations
    indexLeft = setdiff(indexLeft,indexLeft(index1b));
    % Perform adaptive filtering for the remaining pixels
    % WARNING: xi!=0 not supported at that point
    % Deal with locations where Rmin is between ftest2_low and ftest1_low
```

```
    linLeft = find ( (Rmin(indexLeft)>ftest2_low) & (Rmin(indexLeft)<ftest1_low) );
    % Perform a linear attenuation
    alpha(indexLeft(linLeft)) = 1 - (1-beta)* ...
              (Rmin(indexLeft(linLeft))-ftest2_low)/(ftest1_low-ftest2_low);
    % Find the remaining locations
    linRight = setdiff(indexLeft,indexLeft(linLeft));
    % Perform a linear attenuation
    alpha(linRight) = beta + (1-beta)* ...
              (Rmin(linRight)-ftest1_high)/(ftest2_high-ftest1_high);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function alpha = processLHBand(band,n1,n2,gamma1,gamma2,beta,xi)
% Compute variance map and then perform the F-test

    %% VARIANCE MAP COMPUTATION WITH N1x1 HORIZONTAL WINDOW

    % Filter half length
    halfLength = (n1-1)/2;
    % Pad the band to avoid border effects with circshift below
    bPad = padBand(band,halfLength);
    % Stack the shifted version of the band
    bStack = zeros(size(bPad,1),size(bPad,2),n1);
    for i=1:n1,
        bStack(:,:,i) = circshift(bPad,[0 -halfLength+i-1]);
    end
    % Crop temporary stack
    bStack = bStack(halfLength+1:halfLength+size(band,1), ...
              halfLength+1:halfLength+size(band,2),:);
    % Sort along the depth
    bStack = sort(bStack,3);
    % Remove the outliers
    bStack = bStack(:,:,2:n1-1);
    % Compute the variance (biased or unbiased)
    sigma2 = var(bStack,0,3);
    % Free memory
    clear bStack bPad i;

    %% Rmax AND Rmin COMPUTATION

    % Filter half length
    halfLength = (n2-1)/2;
    % Pad a bit the band
    bPad = padBand(sigma2,halfLength);
    % Stack some shifted versions of the band
    bStack = zeros(size(bPad,1),size(bPad,2),n2);
    for i=1:n2,
        bStack(:,:,i) = circshift(bPad,[-halfLength+i-1 0]);
    end
    % Crop temporary stack
    bStack = bStack(halfLength+1:halfLength+size(sigma2,1), ...
              halfLength+1:halfLength+size(sigma2,2),:);
    % Rmax
    Rmax = max(bStack(:,:,halfLength),bStack(:,:,halfLength+2)) ./ sigma2;
    % Compute the two medians
    med1 = median(bStack(:,:,1:halfLength),3);
    med2 = median(bStack(:,:,halfLength+2:n2),3);
    % Rmin
    Rmin = min(med1,med2) ./ sigma2;
    % Free memory
    clear bPad bStack i med1 med2

    % F-TEST VALUES
    r1 = 1-gamma1;
    ftest1_low = finv(r1/2,n1-2,n1-2);
    ftest1_high = finv(1-r1/2,n1-2,n1-2);
    r2 = 1-gamma2;
    ftest2_low = finv(r2/2,n1-2,n1-2);
    ftest2_high = finv(1-r2/2,n1-2,n1-2);
```

```
    %% COMPUTE THE ATTENUATION VALUE

    % Initialize the attenuation array
    alpha = zeros(size(band));

    % Find the features i.e. locations for which neighbour pixels do not
    % share the same variance
    index1a = find( (Rmax<=ftest2_low) | (Rmax>=ftest2_high) );
    % Leave the locations untouched
    alpha(index1a) = 1;

    % Isolate the untouched locations
    indexLeft = setdiff([1:prod(size(alpha))]',index1a);
    % Find the uniform variance areas
    indexBeta = find( (Rmin(indexLeft)>=ftest1_low) & ...
                (Rmin(indexLeft)<=ftest1_high) );
    % Set the attenuation value to beta
    alpha(indexLeft(indexBeta)) = beta;

    % Isolate the untouched locations
    indexLeft = setdiff(indexLeft,indexLeft(indexBeta));
    % Find features
    index1b = find( (Rmin(indexLeft)<=ftest2_low) | ...
                (Rmin(indexLeft)>=ftest2_high) );
    % Leave the locations untouched
    alpha(indexLeft(index1b)) = 1;

    % Isolate the untouched locations
    indexLeft = setdiff(indexLeft,indexLeft(index1b));
    % Perform adaptive filtering for the remaining pixels
    % WARNING: xi!=0 not supported at that point
    % Deal with locations where Rmin is between ftest2_low and ftest1_low
    linLeft = find ( (Rmin(indexLeft)>ftest2_low) & (Rmin(indexLeft)<ftest1_low) );
    % Perform a linear attenuation
    alpha(indexLeft(linLeft)) = 1 - (1-beta)* ...
                (Rmin(indexLeft(linLeft))-ftest2_low)/(ftest1_low-ftest2_low);
    % Find the remaining locations
    linRight = setdiff(indexLeft,indexLeft(linLeft));
    % Perform a linear attenuation
    alpha(linRight) = beta + (1-beta)* ...
                (Rmin(linRight)-ftest1_high)/(ftest2_high-ftest1_high);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [alpha, alpha_hl, alpha_lh] = ...
     processHHBand(band,n1,n2,gamma1,gamma2,beta,xi,alpha_hl,alpha_lh)
% Compute variance map and then perform the F-test

    %% VARIANCE MAP COMPUTATION WITH N1x1 DIAGONAL WINDOW

    % Filter half length
    halfLength = (n1-1)/2;
    % Pad the band to avoid border effects with circshift below
    bPad = padBand(band,halfLength);
    % Stack the shifted version of the band
    bStack_diag = zeros(size(bPad,1),size(bPad,2),n1);
    bStack_idiag = zeros(size(bPad,1),size(bPad,2),n1);
    for i=1:n1,
        bStack_diag(:,:,i) = circshift(bPad,[-halfLength+i-1 -halfLength+i-1]);
        bStack_idiag(:,:,i) = circshift(bPad,[halfLength-i+1 -halfLength+i-1]);
    end
    % Crop temporary stack
    bStack_diag = bStack_diag(halfLength+1:halfLength+size(band,1), ...
                  halfLength+1:halfLength+size(band,2),:);
    bStack_idiag = bStack_idiag(halfLength+1:halfLength+size(band,1), ...
                  halfLength+1:halfLength+size(band,2),:);
    % Sort along the depth
    bStack_diag = sort(bStack_diag,3);
```

```
    bStack_idiag = sort(bStack_idiag,3);
    % Remove the outliers
    bStack_diag = bStack_diag(:,:,2:n1-1);
    bStack_idiag = bStack_idiag(:,:,2:n1-1);
    % Compute the variance (biased or unbiased)
    sigma2_diag = var(bStack_diag,0,3);
    sigma2_idiag = var(bStack_idiag,0,3);
    % Free memory
    clear bStack_diag bStack_idiag bPad i;

    %% Rmax AND Rmin COMPUTATION

    % Filter half length
    halfLength = (n2-1)/2;
    % Pad a bit the band
    sigma2_padDiag = padBand(sigma2_diag,halfLength);
    sigma2_padIdiag = padBand(sigma2_idiag,halfLength);
    % Stack some shifted versions of the band
    bStack_diag = zeros(size(sigma2_padIdiag,1),size(sigma2_padIdiag,2),n2);
    bStack_idiag = zeros(size(sigma2_padDiag,1),size(sigma2_padDiag,2),n2);
    for i=1:n2,
        bStack_diag(:,:,i) = circshift(sigma2_padIdiag, ...
                    [-halfLength+i-1 -halfLength+i-1]);
        bStack_idiag(:,:,i) = circshift(sigma2_padDiag, ...
                    [halfLength-i+1 -halfLength+i-1]);
    end
    % Crop temporary stack
    bStack_diag = bStack_diag(halfLength+1:halfLength+size(sigma2_idiag,1), ...
                    halfLength+1:halfLength+size(sigma2_idiag,2),:);
    bStack_idiag = bStack_idiag(halfLength+1:halfLength+size(sigma2_diag,1), ...
                    halfLength+1:halfLength+size(sigma2_diag,2),:);
    % Rmax
    Rmax_diag = max(bStack_idiag(:,:,halfLength), ...
          bStack_idiag(:,:,halfLength+2)) ./ sigma2_diag;
    Rmax_idiag = max(bStack_diag(:,:,halfLength), ...
          bStack_diag(:,:,halfLength+2)) ./ sigma2_idiag;
    % Compute the two medians
    med1_diag = median(bStack_diag(:,:,1:halfLength),3);
    med2_diag = median(bStack_diag(:,:,halfLength+2:n2),3);
    med1_idiag = median(bStack_idiag(:,:,1:halfLength),3);
    med2_idiag = median(bStack_idiag(:,:,halfLength+2:n2),3);
    % Rmin
    Rmin_diag = min(med1_idiag,med2_idiag) ./ sigma2_diag;
    Rmin_idiag = min(med1_diag,med2_diag) ./ sigma2_idiag;
    % Free memory
    clear sigma2_padDiag sigma2_padIdiag bStack_diag bStack_idiag ...
     med1_diag med2_diag med1_idiag med2_idiag i

    % F-TEST VALUES
    r1 = 1-gamma1;
    ftest1_low = finv(r1/2,n1-2,n1-2);
    ftest1_high = finv(1-r1/2,n1-2,n1-2);
    r2 = 1-gamma2;
    ftest2_low = finv(r2/2,n1-2,n1-2);
    ftest2_high = finv(1-r2/2,n1-2,n1-2);

    %% COMPUTE THE ATTENUATION VALUE

    % Initialize the attenuation array
    alpha = zeros(size(band));

    % Find the features i.e. locations for which neighbour pixels do not
    % share the same variance
    index1a = find( (Rmax_diag<=ftest2_low) | (Rmax_diag>=ftest2_high) | ...
                (Rmax_idiag<=ftest2_low) | (Rmax_idiag>=ftest2_high) );
    % Leave the locations untouched
    alpha(index1a) = 1;
    % Modify the HL and LH bands
    alpha_hl(index1a) = 1;
    alpha_lh(index1a) = 1;
```

```
    % Isolate the untouched locations
    indexLeft = setdiff([1:prod(size(alpha))]',index1a);
    % Find the uniform variance areas
    indexBeta = find( ((Rmin_diag(indexLeft)>=ftest1_low) & ...
          (Rmin_diag(indexLeft)<=ftest1_high)) & ...
          ((Rmin_idiag(indexLeft)>=ftest1_low) & ...
          (Rmin_idiag(indexLeft)<=ftest1_high)) );
    % Set the attenuation value to beta
    alpha(indexLeft(indexBeta)) = beta;

    % Isolate the untouched locations
    indexLeft = setdiff(indexLeft,indexLeft(indexBeta));
    % Find features
    index1b = find( (Rmin_diag(indexLeft)<=ftest2_low) | ...
          (Rmin_diag(indexLeft)>=ftest2_high) | ...
          (Rmin_idiag(indexLeft)<=ftest2_low) | ...
          (Rmin_idiag(indexLeft)>=ftest2_high) );
    % Leave the locations untouched
    alpha(indexLeft(index1b)) = 1;
    % Modify the HL and LH bands
    alpha_hl(indexLeft(index1b)) = 1;
    alpha_lh(indexLeft(index1b)) = 1;

    % Isolate the untouched locations
    indexLeft = setdiff(indexLeft,indexLeft(index1b));
    % Compute some values
    alpha1 = (ftest1_low-ftest2_low) / (ftest1_high-ftest2_high);
    beta1 = (ftest1_high*ftest2_low-ftest2_high*ftest1_low) / ...
                          (ftest1_high-ftest2_high);
    alpha2 = (ftest1_high-ftest2_high) / (ftest1_low-ftest2_low);
    beta2 = (ftest1_low*ftest2_high-ftest2_low*ftest1_high) / ...
                          (ftest1_low-ftest2_low);
    % Perform adaptive filtering for the remaining pixels
    % WARNING: xi!=0 not supported at that point
    % Deal with locations where Rmin is between ftest2_low and ftest1_low
    linUpRight = find((Rmin_idiag(indexLeft)>alpha1*Rmin_diag(indexLeft)+beta1)&...
                (Rmin_idiag(indexLeft)>alpha2*Rmin_diag(indexLeft)+beta2) );
    % Perform a linear attenuation
    alpha(indexLeft(linUpRight)) = beta + ...
          (1-beta)*(max(Rmin_diag(indexLeft(linUpRight)), ...
          Rmin_idiag(indexLeft(linUpRight)))-ftest1_high)/ ...
          (ftest2_high-ftest1_high);
    % Find the remaining locations
    linDownLeft = setdiff(indexLeft,indexLeft(linUpRight));
    % Perform a linear attenuation
    alpha(linDownLeft) = 1 - (1-beta)*(min(Rmin_diag(linDownLeft), ...
          Rmin_idiag(linDownLeft))-ftest2_low)/(ftest1_low-ftest2_low);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pad = padBand(b,n1)
% Simple pading function with mirror effect

    pad = zeros(size(b)+2*n1);

    pad(1:n1,1:n1) = ...
          b(n1+1:-1:2,n1+1:-1:2);
    pad(1:n1,n1+1:n1+size(b,2)) = ...
          b(n1+1:-1:2,1:size(b,2));
    pad(1:n1,size(b,2)+n1+1:size(b,2)+2*n1) = ...
          b(n1+1:-1:2,size(b,2)-1:-1:size(b,2)-n1);
    pad(n1+1:n1+size(b,1),1:n1) = ...
          b(1:size(b,1),n1+1:-1:2);
    pad(n1+1:n1+size(b,1),n1+1:n1+size(b,2)) = ...
          b(1:size(b,1),1:size(b,2));
    pad(n1+1:n1+size(b,1),size(b,2)+n1+1:size(b,2)+2*n1) = ...
          b(1:size(b,1),size(b,2)-1:-1:size(b,2)-n1);
    pad(size(b,1)+n1+1:size(b,1)+2*n1,1:n1) = ...
          b(size(b,1)-1:-1:size(b,1)-n1,n1+1:-1:2);
```

```
    pad(size(b,1)+n1+1:size(b,1)+2*n1,n1+1:n1+size(b,2)) = ...
          b(size(b,1)-1:-1:size(b,1)-n1,1:size(b,2));
    pad(size(b,1)+n1+1:size(b,1)+2*n1,size(b,2)+n1+1:size(b,2)+2*n1) = ...
          b(size(b,1)-1:-1:size(b,1)-n1,size(b,2)-1:-1:size(b,2)-n1);
end
```

padImage.m

```
function [imp,M,N] = padImage(img,L)
%function [imp,M,N] = padImage(img,L)
% Make sure the image dimensions are multiple of 2^L.
%
% OUTPUT:
%   IMP:        Padded version of img, dimensions multiple of 2^L.
%   M, N:       Dimensions of original img.
%
% INPUT:
%   IMG:        Image to be padded
%   L:          The number of Wavelet decomposition levels.

% Minumum padding.
minpad = 0;

% Image dimensions should be multiples of 2^L.
m = 2^L;

% Size of original image
[M N] = size(img);

% How much to pad on each side, top, bottom.
nr = ceil((M+minpad)/m)*m;
nrow = ceil((nr-M)/2);

% How much to pad on each side, left, right.
nc = ceil((N+minpad)/m)*m;
ncol = ceil((nc-N)/2);

% Crop image
im = padarray(img,[nrow,ncol],'symmetric');

% Needed if original image not of even dimensions.
% In which case we pad ncol on left, and ncol-1 on right
% (and same with row indices).
imp = im(1:nr,1:nc);
end
```

readImage.m

```
function img = readImage(filename,spAd)
%function img = readImage(filename)
% Returns grayscale version of type double.

if nargin < 2,
    spAd = 'false';
    disp('old version of readImage in use?, spAd not specified');
end

img = imread(filename);

% handle both full color and grayscale images
if size(img,3) ~=1
    img = rgb2gray(img);
end

img = double(img);

% take log if using that estimator,
% exp(x)-1 is done in estimateNoiseResidual.m
if strcmpi(spAd,'log'), img = log(img + 1); end
end
```