UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Ott Adermann

Wavelet-based Image Denoising

Bachelor's Thesis (9 ECTS)

Supervisors: Irina Bocharova, PhD
Vitaly Skachek, PhD

Tartu 2019

Laineteisenduspõhine piltidelt müraeemaldus

Lühikokkuvõte:

Lühikokkuvõtte sisu.

Võtmesõnad:

Käesolev töö uurib laineteisenduste kasutust piltide kvaliteedi parandamise eesmärgil, neilt müra eemaldades. See annab ülevaate erinevates müra tüüpidest ning müraeemaldusmeetoditest. Edasi keskendub töö laineteisenduspõhistele müraeemaldusskeemidele. Samuti uurib töö laineteisenduspõhise müraeemaldusmeetodi ning kokkupakkimise kombineerimise kasulikkust ja pakub välja uue lävendamise tüübi ning muudatuse eksisteerivale BayesShrink meetodile. Pakutud meetod implementeeritakse C# keeles ning selle implementatsiooni tulemusi, jõudlust ning optimaalseid parameetreid analüüsitakse eksperimentaalsete tulemuste abil.

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

T111 Pilditehnika

Wavelet-based Image Denoising

Abstract:

This thesis studies the use of wavelets for the purpose of improving the quality of images by removing noise from them. It presents an overview of different types of noise and denoising methods. The thesis then focuses on the wavelet transform based denoising schemes. It explores the potential of combining wavelet-based denoising and compression, and presents a new thresholding type and a modification to the existing BayesShrink method. The proposed method is implemented in the C# language and its performance and optimal parameters are analyzed through experimental results.

Keywords:

Wavelets, image processing, image denoising, image compression

CERCS:

P170 Computer science, numerical analysis, systems, control

T111 Imaging, image processing

Table of Contents

1. Introduction....5
2. Overview of Denoising Methods....7
  2.1 Types of Noise and Denoising....7
  2.2 Linear Smoothing Filters....8
  2.3 Anisotropic Diffusion....8
  2.4 Rank Selection Filters....9
  2.5 Discrete Cosine Transform....9
  2.6 Discrete Wavelet Transform....10
3. Wavelet-based Denoising Procedure....12
  3.1 RGB to YC_bC_r Color Space Conversion....12
  3.2 Discrete Wavelet Transform....14
    3.2.1 Filtering....15
    3.2.2 Algorithm....16
    3.2.3 Extension to Images....18
  3.3 Thresholding....20
  3.4 Compression....24
4. Implementation Details....26
  4.1 Parameters....26
  4.2 Color Space Conversion....26
  4.3 Discrete Wavelet Transform....27
  4.4 Thresholding....29
  4.5 Compression....31
  4.6 Performance....32
5. Denoising and Compression Results....34
  5.1 Test Description....34
  5.2 Gaussian Noise....36
  5.3 Impulse Noise....38
  5.4 Speckle Noise....39
  5.5 Poisson Noise....41
  5.6 Compression....43
  5.7 Overall Results....45
6. Conclusion....47
7. References....48

Appendix..51
I. Source code and executables of the created programs..51
II. Full list of PSNR and SSIM results of the images denoised with the created program............51
III. Licence...52

1. Introduction

Digital images, including videos, are a prevalent and important part of today’s world. A lot of these images are acquired from the world through the use of sensors, and as such contain noise, which is inherent to the imperfections of sensors. These images can be medical or scientific images acquired through ultrasound, x-rays, and gamma rays, but are most often images acquired through a digital camera. Of the latter category are movies, advertisements, and other types of images, including a lot of personal pictures often made with poor quality cameras, such as those on mobile phones.

It is often desirable to remove noise from these images to improve their quality, either for practical purposes in medical imagery [1] and computer vision, or for aesthetic purposes [2]. However, removing noise manually is both very time consuming and very difficult, thus making it impractical. Therefore, an automated method for removing noise is desirable.

There exist different types of noise, depending on the capturing device and the type of image being captured. Different denoising methods are more effective on certain types of noise. However, transform-based denoising is known to be rather universal and is successfully used for denoising various kinds of images corrupted by different types of noise. The wavelet transform is a particular form of transform usable as part of the denoising of images.

Most stored or transmitted digital images are compressed using lossy compression techniques. These techniques usually include a digital transform as a step of the image encoding procedure. Because the wavelet transform can be used for both denoising and compression, it is natural to combine the two methods.

The goal of this thesis is the design of a method that improves image quality by removing noise from images using the wavelet transform. A distinguishing feature of this thesis is the focus on combining wavelet transform based denoising and compression. This approach is computationally less expensive than denoising and compressing images separately. Finally, to verify the effectiveness and usability of the method, a proof-of-concept computer program is created that denoises images and optionally estimates the possible amount of compression after the denoising process.

In Chapter 2, an overview is given of some of the existing denoising schemes and their effectiveness on various types of noise. Chapter 3 is a theoretical overview of the used methods and ideas, and Chapter 4 contains the implementation and performance details. The final chapter measures the denoising quality of the resulting method on different types of noise, and compares the results to some of the previously existing methods mentioned in Chapter 2.

The appendix contains links to the source code and executables of the finished program and associated tools, as well a table of all the denoising results.

2. Overview of Denoising Methods

There are a lot of different methods and approaches to improve the quality of an image by reducing the amount of noise in it. This chapter first covers the most common types of noise present in digital images, explains what denoising is, and gives an overview of some of the existing methods of image denoising, and how well they perform.

2.1 Types of Noise and Denoising

Image noise is the random undesirable differing of pixel color values from their expected values. Generally, noise is more noticeable and disturbing when the difference between the color values of pixels and their surrounding pixels is large. Most noise can be described by one or a combination of multiple of the four following noise models [3, 4], examples of which can be seen in Figure 1.

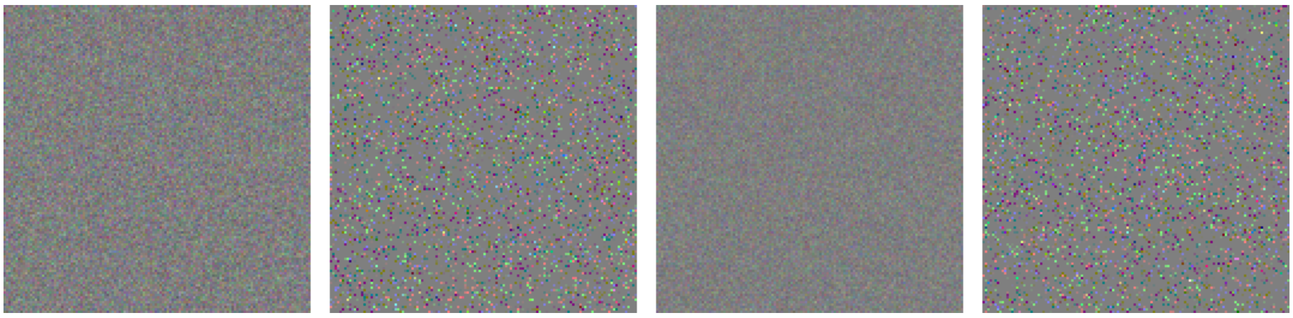


Figure 1. Different types of noise on a gray background. From left to right: Gaussian noise with a standard deviation of 15, impulse noise affecting 10% of the pixels in the image, Poisson noise with an average of 150 photons, speckle noise affecting 10% of the pixels in the image.

Gaussian noise is additive noise with a Gaussian distribution around zero. It is often the largest noise component when capturing an image due to imperfect image sensors, affecting all captured pixels. A lot of pixels corrupted by this type of noise are individually not noticeably noisy, due to the Gaussian distribution.

Impulse noise most often comes from errors in image transmission, analog-to-digital conversion, or broken pieces of the sensor. It replaces individual pixels of the image with a random color value. Salt and pepper noise is a subtype of impulse noise, but only has zero or full intensity values instead of random ones.

Poisson noise occurs in image sensors due to the amount of photons hitting the sensor statistically differing. This causes fluctuations in the color intensities of the image. Poisson noise is similar to Gaussian noise, but typically has a much lower intensity.

Speckle noise is found in radar and ultrasound systems. It is multiplicative noise, adding to the value of each pixel some positive or negative multiple of that value. Due to this, it affects brighter areas of the image more significantly.

Image denoising is the process of removing noise from an image. Ideally, denoising should first identify all the noisy pixels and then restore them to their original values, while leaving the color values of non-noisy pixels untouched. However, in practice, it is not always possible to correctly decide which pixels are noisy, which are not. Most commonly, fine details can be indistinguishable from noise, and are often lost after denoising. Secondly, the original color values of the pixels are not known either. This is worse when the noise covers all pixels in the image, such as for Gaussian and Poisson noise, because there are no neighboring non-noisy pixels which could be used to guess the original color values of the noisy pixels. Therefore, it is not possible to perfectly restore images, but different methods are still capable of eliminating some or multiple types of noise quite effectively without damaging the parts of the image not affected by noise nor the important details in the image.

2.2 Linear Smoothing Filters

Linear smoothing filters are perhaps the simplest way to reduce noise in an image. They work by taking an average or a weighted average of the pixel color values at and around each pixel in an image. The weights in the latter case are often in the form of a 2D Gaussian function, and applying such a filter is the equivalent of Gaussian blurring an image. Because it is characteristic of most forms of noise to generally be distinct from its neighbors, blurring can effectively reduce noise by forcing each pixel to be more similar to its neighbors [5]. This, however, has the undesired side effect of reducing the quality of the image by turning the image blurry because the filter indiscriminately averages all pixels, not just those with noise. Further, instead of just eliminating the noisy pixels, the smoothing filter instead smears them across the neighboring region.

2.3 Anisotropic Diffusion

Anisotropic diffusion builds upon linear smoothing by combining it with edge detection. More smoothing is applied in the direction of edges, while less smoothing is applied across edges. This means that uniform regions of an image are more heavily blurred, while regions with sharp transitions retain their quality [6]. While it solves the problem of indiscriminately blurring away important parts of an image – the edges, it still does not limit the blurring to just noise, and still smears the noise across the neighboring region.

2.4 Rank Selection Filters

Rank selection filters work by taking each pixel along with its neighboring region, sorting those pixels by their values, and then choosing one according to its ranking to replace the center pixel with. The simplest case of this is the median filter, which chooses the middle ranking value. Compared to a linear smoothing filter, the median has the advantage of generally not being affected by outliers, which is what noise usually is. Additionally, since the median value does not interpolate between any of the existing values, but is instead one of them, blurring does not occur, and edges are preserved well. This method is most effective when noise values deviate substantially from the surrounding values, such as in salt and pepper noise, but is sometimes even less effective than a linear smoothing filter for Gaussian noise [7, 8]. A quality-degrading side effect of the median not creating any new interpolated pixel values is that the number of colors an image has can only decrease, leading to an increasingly blocky image. Further, a regular median filter also does not discriminate between noisy and non-noisy pixels, and is thus applied to all pixels of an image.

An example of a median filter particularly effective against salt and pepper noise is the Iterative Trimmed Median Filter [9]. It identifies only zero or full intensity pixels as noise, and only corrects those by selecting a median from the non-noisy values. If the entire neighborhood is noisy, the value will be corrected in the following iteration. This method solves the issue of the image quality being degraded due to median filtering being applied to all pixels. However, only considering fully saturated values as noise limits the use of the filter to images that only have salt and pepper noise and are free of large areas of saturated color.

2.5 Discrete Cosine Transform

The discrete cosine transform (DCT) transforms an image into a collection of cosine functions with different frequencies and amplitudes. It is possible to represent the result of this transform as an image of equal size to the original, where similar frequencies are grouped together. An inverse transform can then be performed to retrieve the original image. Notable differences from the discrete Fourier transform (DFT) are that the DCT is real-valued, not complex-valued, and that the DCT has better energy compaction – the magnitudes of the frequencies are more concentrated into fewer frequencies. Both of these differences are an important part in compression, because the important parts of an image are packed into a smaller amount of data compared to the DFT [10, 11].

Noise is mostly localized in the high frequency components of images. Because the DCT separates an image into its various frequency components, it is possible to use DCT-based techniques to denoise images by removing high frequencies, which are most likely to correspond to noise, while leaving the rest of the image untouched. This is an improvement over the previously discussed methods, which generally failed to distinguish noise from the rest of the image. Examples of using DCT include Wiener filtering in DCT domain [12] and adaptive DCT-based filtering [13].

2.6 Discrete Wavelet Transform

In its simplest form, the discrete wavelet transform (DWT) decomposes an image into a quarter-size image representing the low frequency component of the image, and into three quarter-size images representing the directional high frequency components of the image. The process can be repeated on the low frequency component, further splitting it into low and high frequency components. This transform is invertible, meaning the components can be combined back into the original image [14, 15].

A downside of the DCT and the DFT is that they represent an image as a combination of frequency components which extend across the entire image, whilst it is more natural, and often also more useful, to represent the image as a combination of components that only have a value in a limited neighborhood around a point, therefore allowing the capture of information unique to the region around that point. The DWT has an advantage in this regard, as it captures both frequency and location information [16]. Dividing an image into smaller sections before applying a discrete Fourier or cosine transform can achieve a similar result that captures location information, but a wavelet transform achieves this in a more natural way. Moreover, wavelets can better represent sharp and non-recurring transitions in an image, such as edges, while sines and cosines are by definition non-local and extend to infinity [17].

For similar reasons as the DCT, the DWT is a good basis upon which to build methods for compressing and denoising images. The listed advantages of the DWT benefit denoising and compression as well. For denoising, an image is first transformed using a DWT. Notably, there exist different wavelet functions which can be used for this transform. The choice of the wavelet function can affect the quality of the results and is therefore important [15]. There also exist different methods to eliminate noise from the transformed image. Hard and soft thresholding are often used [18, 19], but statistical and other methods have been shown to be highly effective as well [19].

As part of this thesis, both Haar and orthogonal wavelets are tested for the DWT. Hard, soft, and a custom thresholding method are tested, and a modified, adjustable version of BayesShrink [18] is used for finding the thresholds.

3. Wavelet-based Denoising Procedure

The denoising procedure developed as part of this thesis consists of multiple separate steps. The image is first prepared for denoising through a color space conversion followed by a discrete wavelet transform. The transformed image is then denoised using thresholding and optionally compressed and saved. If the image is not compressed or when the compressed image is loaded for viewing, it is then transformed back. This chapter describes these steps and explains why they are a necessary part of the overall procedure.

3.1 RGB to YC_bC_r Color Space Conversion

In nearly all cases, monitors display their gamut of color using tiny red, green, and blue lights. Similarly, the individual photosites of digital cameras capture only red, green, or blue light each [20]. Because of this, digital images are also represented as a combination of red, green, and blue lights. An image can be thought of as being a combination of three separate images, each consisting of a varying intensity of only one color of light, as shown in Figure 2.

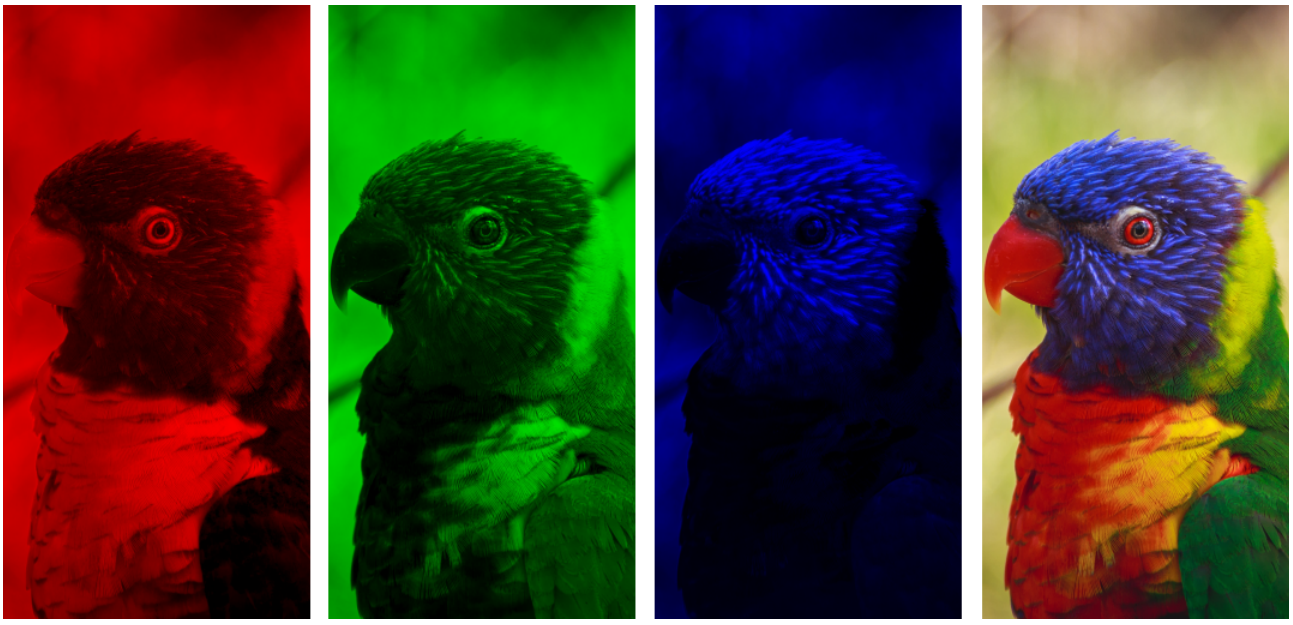


Figure 2. The red, green, and blue color components that make up an image of a parrot [21].

Our eyes are more sensitive to light intensity than the color of light. For images, this means that we care more about the brightness (luma) information quality of an image than its color (chroma) information quality. As such, it is beneficial that both the amount of denoising and the amount of compression be configurable separately for the luma and chroma components. For this reason, a color space conversion is performed from RGB to YC_bC_r.

In relation to the RGB color space, the YC_bC_r color space is defined such that Y (luma) contains the perceived brightness, which is also the grayscale of the image, while C_b (chroma blue) and C_r (chroma red) contain all the color information of the image. In mathematical terms, this relation is shown in the formulae in (1), which are an ITU-T standard [22], and a visual representation is given in Figure 3. It can be seen from Figure 2 that green is indeed perceived as the brightest of the images, while blue is the darkest, as is in the formula for luma.

$$\begin{aligned} Y &= 0.299\,R + 0.587\,G + 0.114\,B \\ C_b &= (B-Y)\,/\,1.772 + 128 \\ C_r &= (R-Y)\,/\,1.402 + 128 \end{aligned} \tag{1}$$

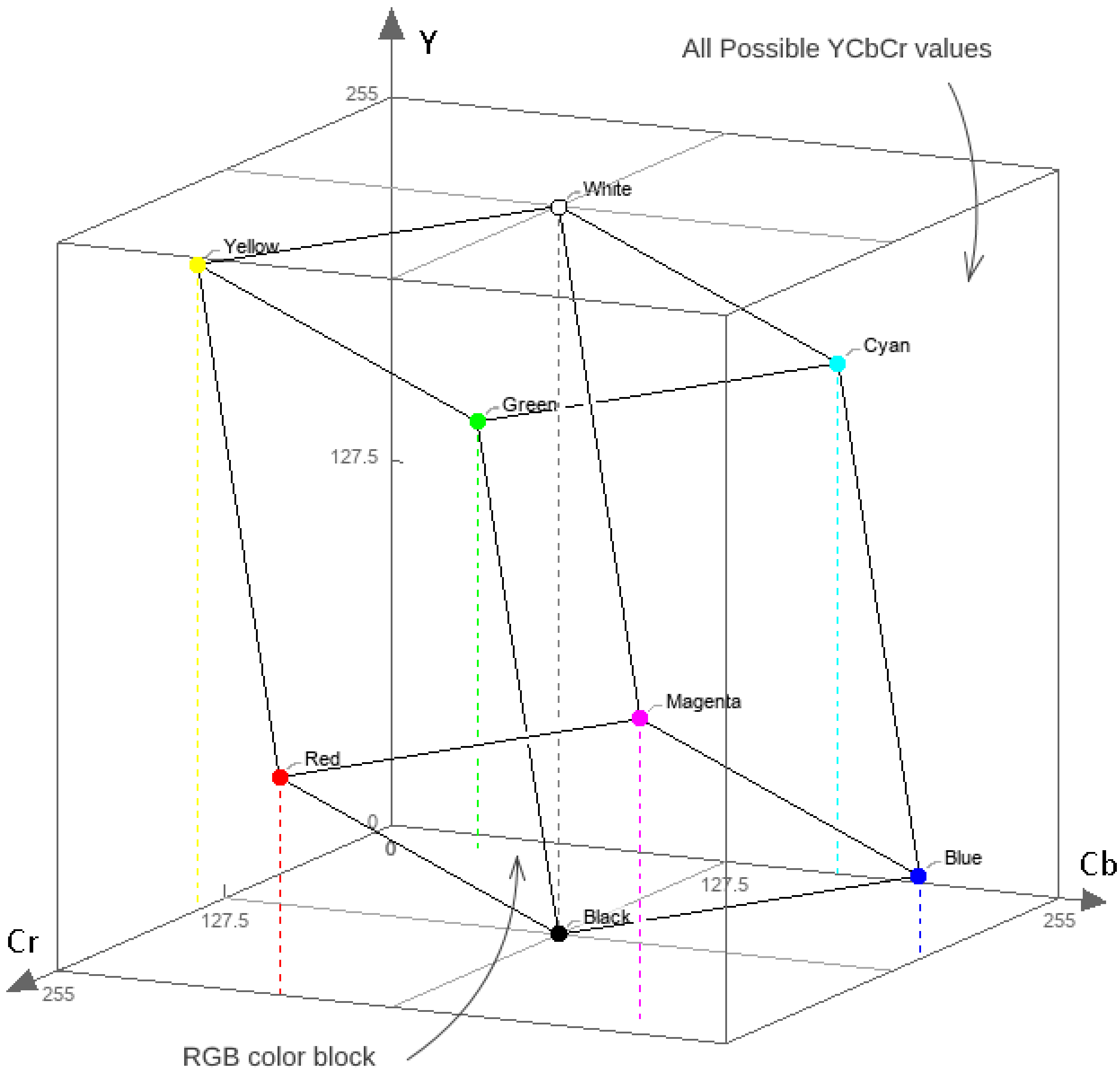


Figure 3: RGB color cube in the YC_bC_r color space.

The backwards color space conversion, YC_bC_r to RGB, can be derived from the formulae in (1), and is given by the formulae in (2), which are part of the same ITU-T standard [22]. It is important to note that, as seen in Figure 3, the space of all possible YC_bC_r values is larger than the space of all possible RGB values, so after the backwards conversion, the values have to be clamped to ensure they are valid.

$$\begin{aligned}
R &= \mathrm{Min}\left(\mathrm{Max}\left(Y + 1.402\left(C_r - 128\right), 0\right), 255\right) \\
G &= \mathrm{Min}\left(\mathrm{Max}\left(Y - 0.344\left(C_b - 128\right) - 0.714\left(C_r - 128\right), 0\right), 255\right) \\
B &= \mathrm{Min}\left(\mathrm{Max}\left(Y + 1.772\left(C_b - 128\right), 0\right), 255\right)
\end{aligned} \tag{2}$$

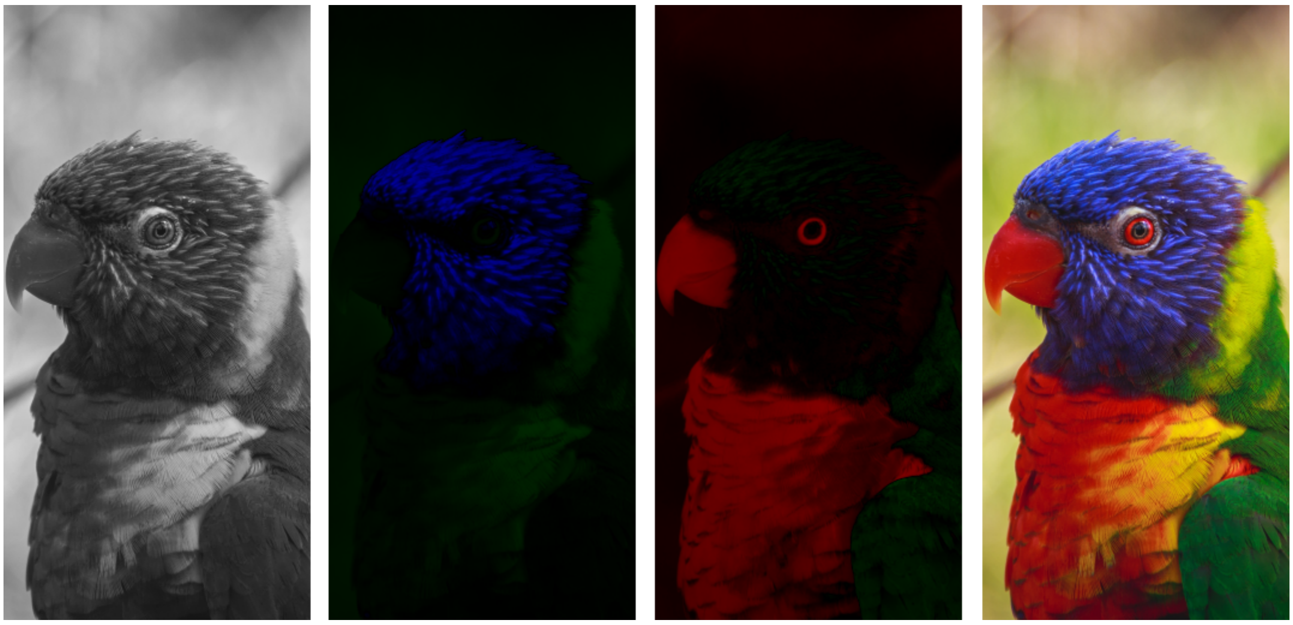


Figure 4. The luma and chroma components that make up an image of a parrot [21].

The result of the color space conversion can be seen in Figure 4, which also showcases that the luma component holds more visually useful information than either chroma component. This can be used for more efficient compression, allowing more information to be removed from the chroma components without significantly affecting the quality of the image.

3.2 Discrete Wavelet Transform

The most important part of the denoising process is the discrete wavelet transform (DWT). This transform, similarly to the DCT and the DFT, decomposes the original signal into a set of basis functions multiplied by the transform coefficients. As was shown by S. Mallat the wavelet transform can be implemented by filtering the original signal using a so-called wavelet filter bank, decomposing the signal into a multiresolution representation [23]. The coefficients of filters in the wavelet filter bank are related to the wavelet basis functions via dilation and wavelet equations [24].

A wavelet filter bank is also known as a wavelet family [25]. The so-called father wavelet and the son wavelets derived from it through scaling the father wavelet act as scaling functions – low-pass filters at the various resolutions which extract the low frequency information from the signal. On the other hand, the so-called mother wavelet and the daughter wavelets derived from it through scaling the mother wavelet act as the basis functions – the wavelets – for the wavelet transform at its multiple resolutions. They can also be considered the high-pass filters which extract the high frequency information from the signal. Intuitively, the low frequency component is like a smaller scale copy of the original signal, while the high frequency component contains all the information lost from the low frequency component due to the decrease in scale, such as contours and fine details.

3.2.1 Filtering

Filtering can be implemented as convolution, which is a mathematical operation that can be performed between two signals. For discrete signals *f* and *g*, the convolution $f*g$ is given by the formula in (3) [26].

$$(f*g)_i=\sum_{j=-\infty}^{\infty} f_j g_{i+j} \tag{3}$$

Usually, one of the signals is the longer input signal, and the other is a short signal used as the filter. It is not possible to do an infinite number of calculations for each sample of the convolution, but because wavelet filters have a finite impulse response, meaning they are of a finite length, and the values of the filter signal outside its defined range are 0, we can use the equivalent formula in (4) instead (assuming *g* is the input signal, and *f* is the filter with a finite length of *n*).

$$(f*g)_i=\sum_{j=0}^{n-1} f_j g_{i+j} \tag{4}$$

It can be seen that with a constant length filter, convolution works in linear time with respect to the length of the signal. For any sample of the input signal which is outside the defined range, we consider the signal periodic, and take the sample from the other end of the defined range. This is known as circular convolution, and is defined as

$$(f*g)_i=\sum_{j=0}^{n-1} f_j g_{(i+j)\bmod n}$$

Circular convolution is necessary for the DWT in order to have a convolution that is of the same length as the original signal and is invertible. For other purposes, these samples can also be clamped to each end of the signal or given zero values.

As an example, let $f = [0.5, 0.5]$, which is an averaging filter, and let our input signal $g = [1, 7, 7, 5, 4, 8, 7, 9]$. Then their circular convolution $f * g = [4, 7, 6, 4.5, 6, 7.5, 8, 5]$. It can be seen that the new signal is made of the pairwise averages of the original signal, and it has effectively been smoothed.

3.2.2 Algorithm

For a wavelet transform, two filters are required – a low-pass filter, which constructs the low frequency component, and a high-pass filter, which constructs the high-frequency component [25]. These filters are chosen such that the original signal can later be reconstructed from a combination of the low and high frequency components. This is known as an inverse transform and it requires two reconstruction filters – one which reconstructs the even-indexed samples, the other which reconstructs the odd-indexed samples.

Perhaps the simplest filter bank is the Haar filter bank. The non-normalized low-pass filter is [1, 1], and the non-normalized high-pass filter is [1, -1] [25]. It can be seen that these filters correspond to the sums and differences of the signal, respectively. To show a worked example, using the signal [1, 7, 7, 5, 4, 8, 7, 9], the circular convolution with the low-pass filter gives [8, 14, 12, 9, 12, 15, 16, 10], and the circular convolution with the high-pass filter gives [-6, 0, 2, 1, -4, 1, -2, 8]. As will be shown, all odd-indexed values can be discarded, leaving [8, 12, 12, 16] and [-6, 2, -4, -2]. Normalizing these filters by a factor of 0.5 gives the averages and half-differences instead – [4, 6, 6, 8] for the low frequency and [-3, 1, -2, -1] for the high frequency signals. It can now be seen that, for a pair of values, if the average and half-difference are saved, then the original pair can be recovered. For the first value of the pair, the average and the half-difference have to be summed, and for the second value, the half-difference has to be subtracted from the average. This gives us the reconstruction filters [1, 1] and [1, -1]. It is mostly a coincidence that they coincide with the low- and high-pass filters. For this simple example, it can be worked out by hand that using the described procedure indeed gives back the original signal. For longer filters, however, it would be simpler to define this operation as convolution. First, the low and high frequency signals must be interleaved, giving [4, -3, 6, 1, 6, -2, 8, -1]. This signal is convolved with each filter, and the odd-indexed values are again discarded, leaving [1, 7, 4, 7] and [7, 5, 8, 9]. Interleaving these gives back the original signal. The same algorithm can be used for longer filters as well.

Another benefit of the DWT is that the resulting low frequency signal resembles the original signal. This means the DWT can be applied to the low frequency component multiple times at different levels, each time obtaining the high frequency component of that level and an even lower frequency component. Figure 5 illustrates this decomposition and reconstruction process. $W_\varphi[J - n, k]$ are the low frequency wavelet coefficients which correspond to the n-th decomposition level. At each decomposition level, the low- and high-pass filters H_0 and H_1 produce higher level low and high frequency representation of the current level low frequency coefficients. Both signals are then downsampled, indicated by the down arrow, removing every other value. The high frequency coefficients of the n-th decomposition level are saved as $W\psi[J - n, k]$, while the low frequency component can be filtered further. The process can be inverted from the final low and high frequency wavelet coefficients by using the reconstruction filters G_0 and G_1. The highest level low and high frequency signals are upsampled, indicated by the up arrow, by adding zeroes for every other value, filtered with the corresponding reconstruction filters, then added together to get the one level lower low frequency signal. The process is repeated with the reconstructed signal and the next high frequency signal until the original image is reconstructed.

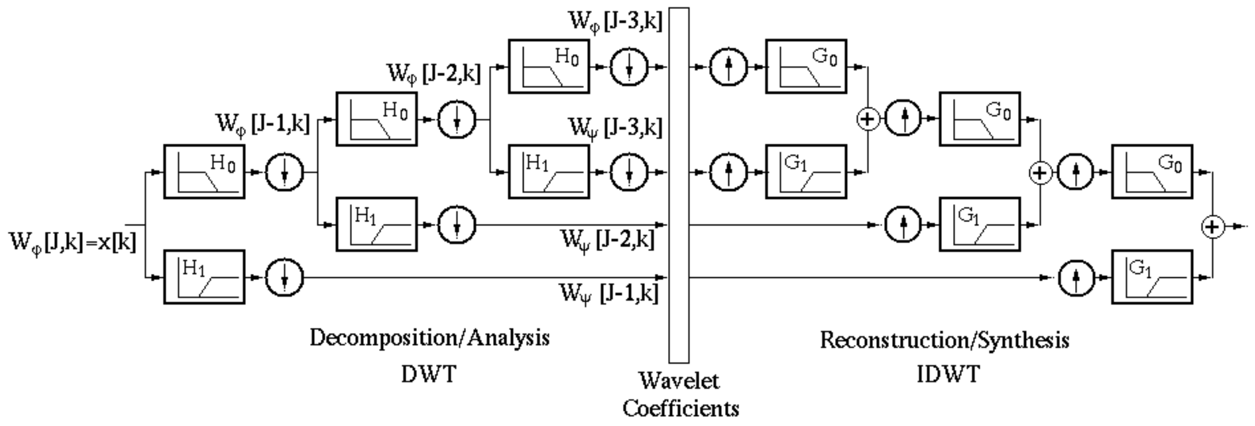


Figure 5: Signal decomposition and reconstruction with wavelets [26].

This multi-level decomposition is a useful step because it enables applying different amounts of denoising and compression to different levels. By its nature, noise occurs primarily in the high frequency component, so it makes sense to apply stronger denoising to that component. Further, as images are largely comprised of smooth gradients or flat areas of color with relatively few details such as edges, the low frequency components carry more useful information. For this reason, it is not as harmful to the quality of an image if information is lost from the high frequency components due to denoising or compression in comparison to losing information from the low frequency components.

3.2.3 Extension to Images

Images can be viewed as two-dimensional discrete signals. They have a value (color intensity) at every pixel for each color component (channel) they are made of. Because the wavelet transform filters are separable, no special considerations have to be made for using wavelet transforms on two-dimensional signals in comparison to one-dimensional signals. The transform is first applied on either the rows or columns, and then applied in the other dimension. As follows from the separability of the filters, this is equivalent to convolution with a two-dimensional filter that is the product of the corresponding one-dimensional filter and its transpose.

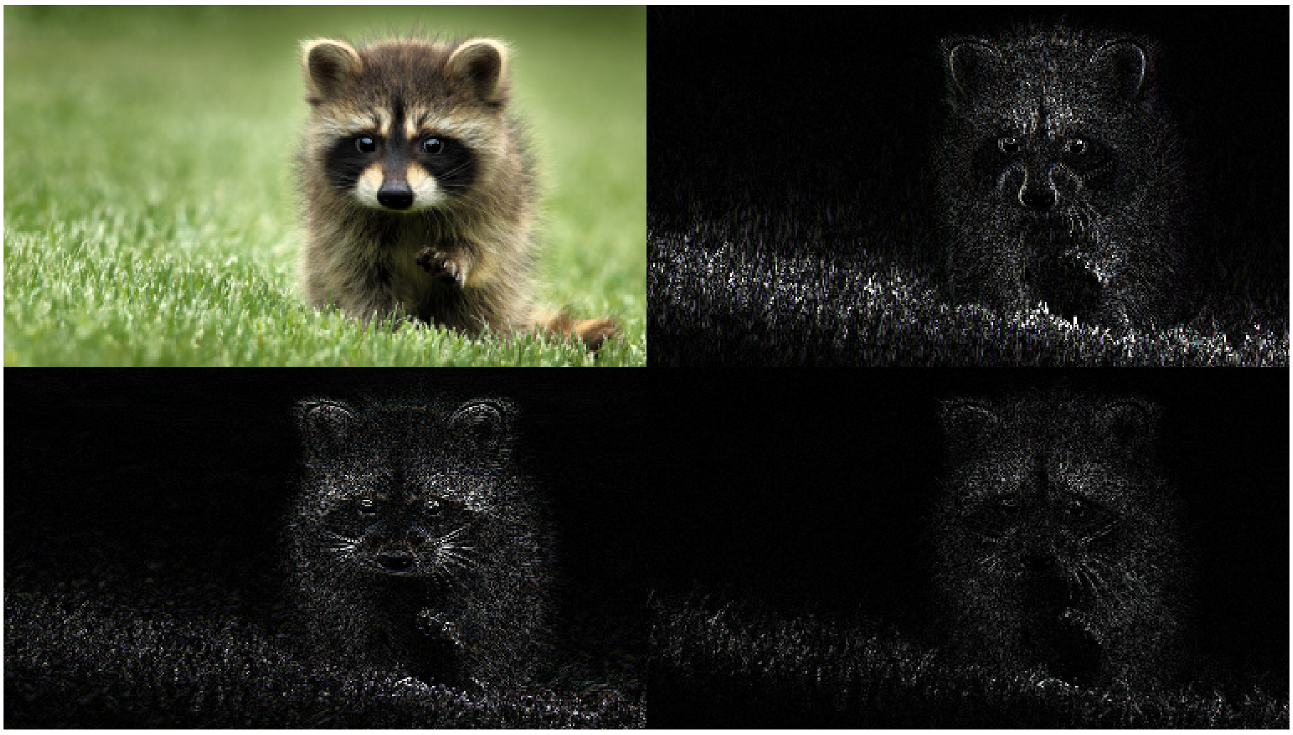


Figure 6: One level of wavelet transform on a picture of a raccoon [27].

Figure 6 shows a visual representation of the DWT of an image. The low frequency component is in the top left corner, and the high frequency components are on the right side and the bottom. This is not an entirely accurate representation of the underlying data. Most values of the high frequency components are either too low to be visible, or negative and could not be shown on an image at all. To overcome this, they are shown as absolute values, and their intensity has been increased for better visibility. Additionally, it is not necessary to fit all the components in one image, and the neighboring pixels on the edges of two components are not next to each other in a mathematical sense. However, this representation is convenient as it shows that after the transform, all the components take up as much space together as the original image, and it is intuitive which component is which, based on their location in the image. As is also illustrated in Figure 7, on the right side are the horizontal high frequency components, on the left are the horizontal low frequency components, while the bottom has vertical high frequency components, and the top has vertical low frequency components. Figure 7 also shows how the low frequency component can be further decomposed. The symbols show the name of the component, with the first letter showing the horizontal frequency, the second letter showing the vertical frequency, and the number showing which level of the transform they belong to. For example HL2 is the horizontal high frequency, vertical low frequency component of the level 2 transform.

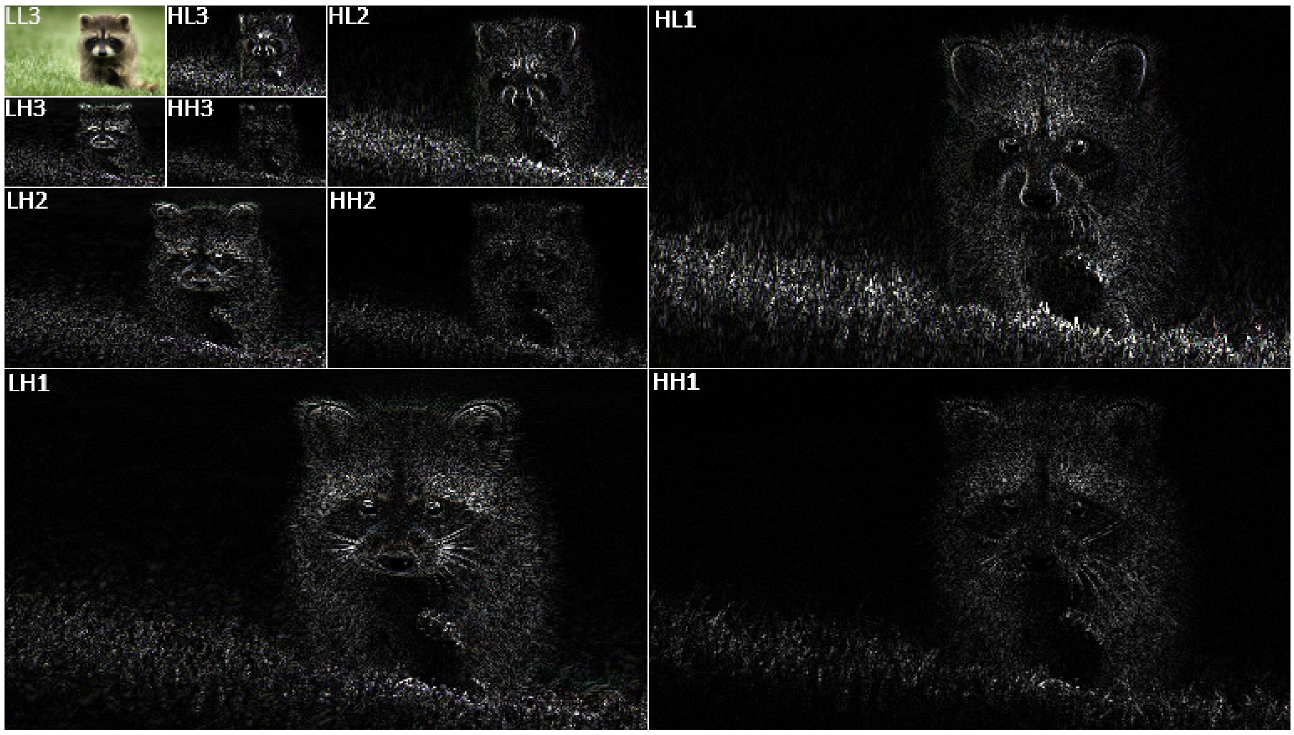


Figure 7: Three levels of wavelet transform on a picture of a raccoon [27].

It can be seen that high frequency components contain less information of the image. This includes the lower level high frequency components in comparison to the higher level high frequency components, because higher level components are derived from a lower frequency signal than the lower level components. As such, the HH1 component carries the least useful information about the image overall. These are important things to note for denoising with thresholding.

3.3 Thresholding

The idea behind thresholding is that for the high frequency components of a signal, most important features have a high intensity (large absolute value) in the samples that contain them, and that they are sparse, meaning that most samples do not contain important information [29]. This can be seen on Figures 6 and 7. Conversely, a low to moderate amount of noise has a lower intensity than the important features, and it is present in most samples. Removing the samples that have a low intensity while keeping ones with a high intensity should thus work as an effective method of denoising [29]. This process is known as thresholding, and the intensity below which samples should be removed is the threshold. Different forms of thresholding have been shown to be highly effective at removing additive Gaussian noise [18] and are expected to work on Poisson noise. However, thresholding is not expected to be an effective method for removing impulse nor speckle noise, as neither conforms to the properties of noise described – they can be both high intensity and sparse.

Thresholding defines that values below the chosen threshold are removed (set to zero). It does not define what is done with values above the chosen threshold. For this, two approaches are common: hard and soft thresholding [18]. Hard thresholding leaves values above the threshold as they were, while soft thresholding brings values above the threshold closer to zero by the threshold amount. Hard thresholding leaves discontinuities in the denoised images because all high frequency values between zero and the threshold have been removed. These can be large and visually unpleasant if the image is particularly noisy. On the other hand, soft thresholding may overly smooth an image, as all large values are brought closer to zero, lowering the intensity of details and edges. This thesis also tests a third type of thresholding thought up by the author, which has no discontinuities and which lowers the intensities of large values less. More precisely, it interpolates values above the threshold to be between zero and the maximum value. It is inspired by both hard and soft thresholding and will be referred to as moderate thresholding. The functions for hard, soft, and moderate thresholding are given in (5), (6), and (7), respectively, where T is the threshold parameter, and the visual representations of these functions can be seen in Figure 8.

$$h(x)=\begin{cases} 0, & \text{if } |x|<T \\ x, & \text{otherwise} \end{cases} \tag{5}$$

$$s(x)=\begin{cases} x-T, & \text{if } x>T \\ x+T, & \text{if } x<-T \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

$$m(x)=\begin{cases}\dfrac{x-T}{1-\dfrac{T}{255}}, & \text{if } x>T\\ \dfrac{x+T}{1-\dfrac{T}{255}}, & \text{if } x<-T\\ 0, & \text{otherwise}\end{cases} \tag{7}$$

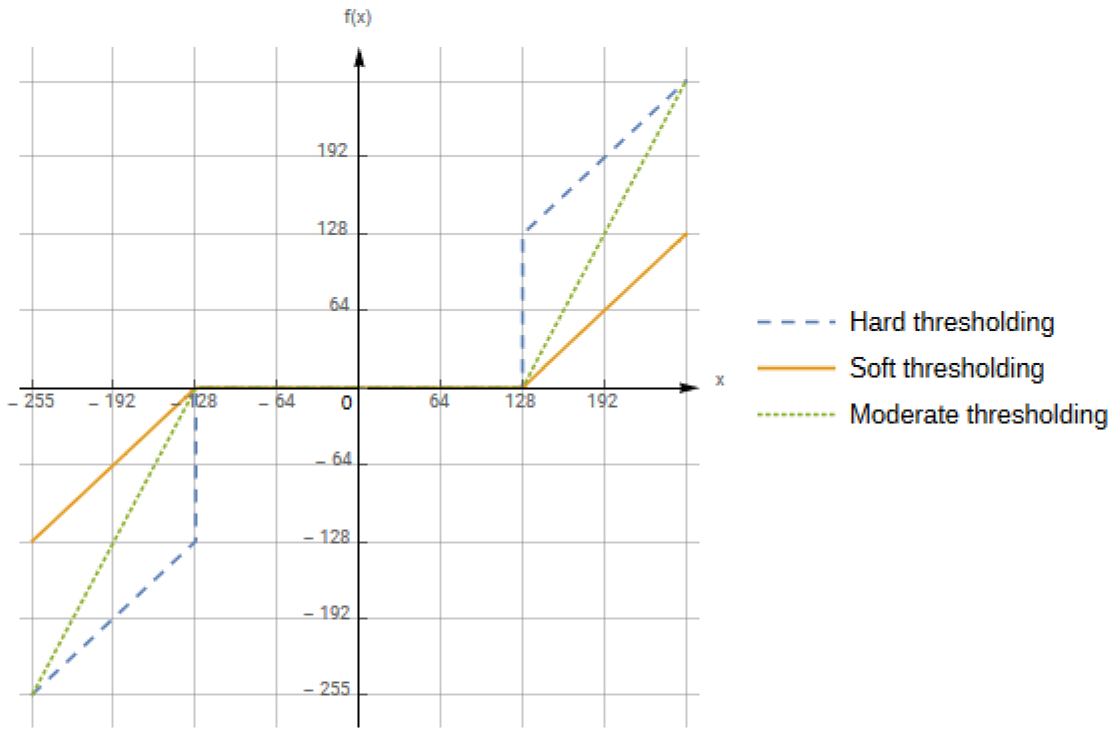


Figure 8: The effect of the hard, soft, and moderate thresholding functions on the value of a pixel, at a threshold of T=128.

Thresholding is a very simple yet effective technique when using optimally chosen thresholds, but it does not include an inherent way to find these optimal thresholds. Over time, more complicated and more successful methods have been developed. The simplest option of having a single prechosen value for the threshold does not perform adequately mainly because the amount of noise differs from image to image. A more complicated method, such as VisuShrink [30] picks the threshold by estimating the amount of noise on the image. However, its performance is still unsatisfactory, as this threshold is the same for each frequency component, while they may carry different amounts of noise. SureShrink [31] adapts for this by considering each frequency component separately. Finally, BayesShrink, the modified version of which is implemented in this thesis, has been shown to yield even better results and finding thresholding values that are close to the optimal [18].

The original formula for the threshold estimated by BayesShrink [18] is defined per high frequency component as the estimated noise variance over the signal standard deviation.

$$T_B = \frac{\sigma_{noise}^2}{\sigma_{signal}} \tag{8}$$

The noise variance in (8) is estimated from the median of all samples in the highest frequency component.

$$\sigma_{noise} = \frac{\text{median}(|Y_{ij}|)}{0.6745}, \quad Y_{ij} \in \text{component } \text{HH}_1 \tag{9}$$

The standard deviation of the signal in (8) is found for each high frequency component separately as

$$\sigma_{signal} = \sqrt{\max(\sigma_Y^2 - \sigma_{noise}^2, 0)} \tag{10}$$

where σ_Y^2 is the variance of all values Y_{ij} for that high frequency component *Y*. Because the filter coefficients of the high-pass filter producing the high frequency component have a mean of 0, each high frequency component *Y* also has a mean of 0. This allows the simplification of finding the variance of σ_Y^2 as

$$\sigma_Y^2 = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} Y_{ij}^2$$

As a more intuitive explanation, it can be seen from (10) that the original signal is roughly estimated to be the current signal, from which the noise is subtracted. This corresponds to an additive noise model, such as Gaussian noise, and the subtraction of noise is analogous to soft thresholding. Because of this, the method is expected to perform better removing Gaussian noise using soft thresholding than removing other types of noise or using other types of thresholding.

Experimental results in Chapter 5 show that the original BayesShrink formula is often too aggressive in removing noise. In addition to noise, important details are also removed. As such, a modification to the formula (9), which is used to estimate the standard deviation of noise, has been made by the author. As seen in formula (11), the median is multiplied by *k* instead, which acts as a parameter that controls the amount of noise removed. Lowering *k* removes less noise, but also keeps more details. In the case of a user-controlled application, the user could choose the value of *k* themselves, to get the result that subjectively looks best to them. The objective results of using different values of *k* are given in Chapter 5, and a default value for automatic use is suggested. A value of 1.5 is roughly equivalent to the original formula.

$$\sigma_{noise} = k \cdot \text{median}(|Y_{ij}|), \quad Y_{ij} \in \text{component HH}_1 \tag{11}$$

An example of what noise looks like in the wavelet domain, and how thresholding removes it can be seen in Figure 9.

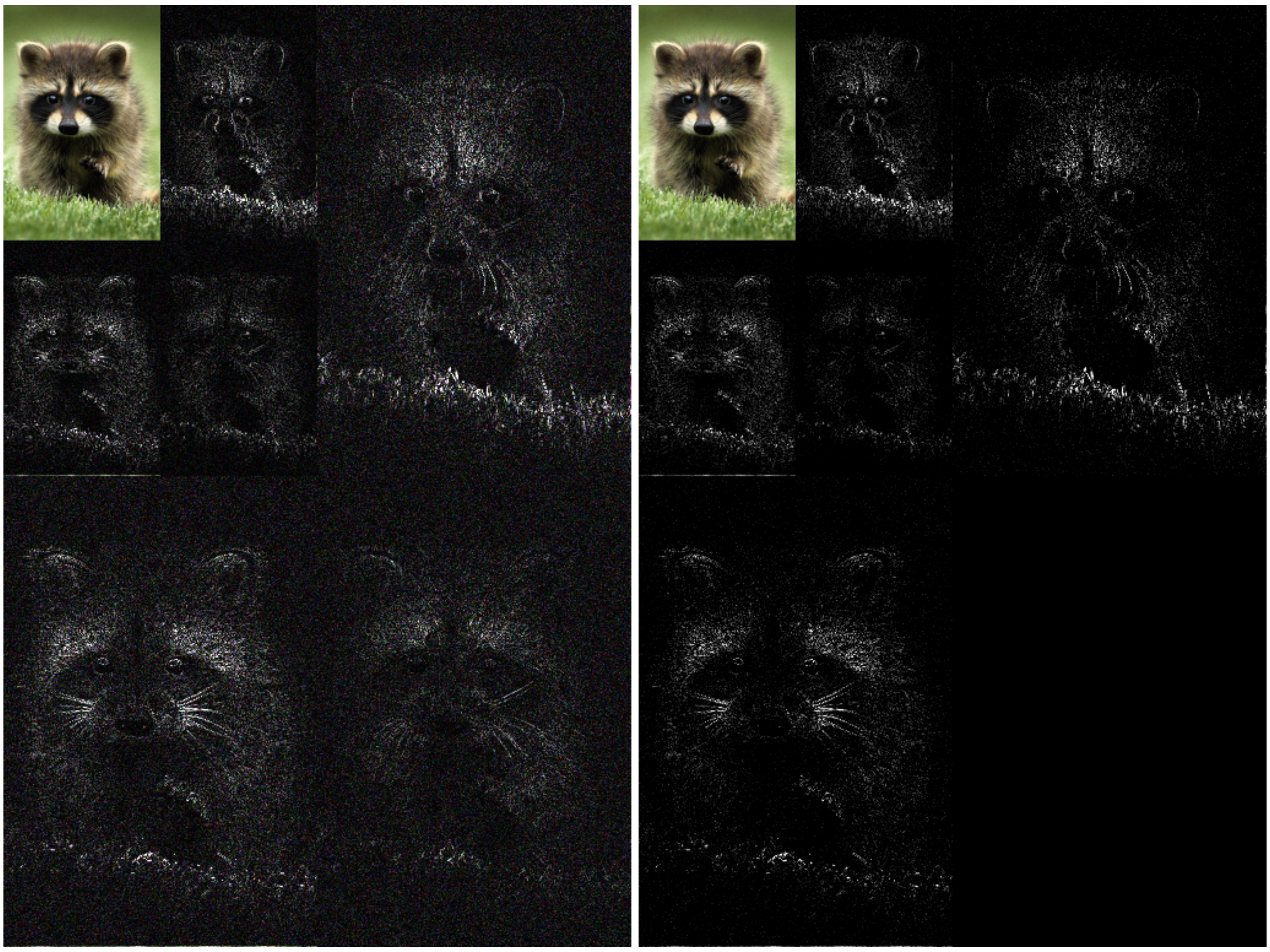


Figure 9: Example of noise and thresholding in the wavelet domain [27].

After thresholding, the inverse DWT and color space conversion are performed, giving back the original image with noise removed from it. Alternatively, the image can be quantized, then encoded in its current state and saved to a file, which acts as a form of lossy compression. The inverse DWT and color space conversion would then be performed each time the image is loaded from its compressed state for viewing.

3.4 Compression

A wavelet transform localizes the energy of an image into fewer areas. In other words, a lot of the pixels have a value that is close to zero, while the important information is kept in the smaller low frequency component and the sparsely occurring high value pixels in the high frequency components, as seen in Figures 6 and 7. This localization of energy is useful for compression. For example, the JPEG2000 standard uses a wavelet transform and quantization to reduce the entropy of images [32].

Quantization is the process of rounding values to a multiple of some quantization step. The larger the step size, the more the value deviates from its original value, and as such, the more quality is lost. However, a larger step size also means that there will be fewer distinct values, which reduces the entropy, as seen in formula (12). Therefore, the amount of quantization is the adjustable parameter which controls the balance of quality and compression. Some step sizes are not inherently better than others. Instead, the choice should depend on how much quality is valued in comparison to compression in a specific use case.

While implementing the encoding procedure necessary for proper compression is outside the scope of this thesis, it is possible to estimate the compression amount using first-order entropy [33], defined as

$$H(X) = -\sum_{i=1}^{n} P(x_i)\log_2 P(x_i) \tag{12}$$

where X is a discrete random variable with possible values of x_1, x_2, …, x_n and the probability of x_i occurring is $P(x_i)$. The variable X, and the entropy calculated from it, is found separately for each frequency and color component, and the possible values x_1 to x_n are the possible different values of individual pixels, with n being the amount of different values. The entropy $H(X)$ is the minimal average amount of bits that is required to represent each pixel in the corresponding component, and thus the minimal amount of bits required to represent an entire component is the amount of pixels in that component, multiplied by its entropy. The final compressed size of the image would then be sum of the bits needed for representation of all of its components. As a comparison, each pixel takes 8 bits per color channel uncompressed. The compression ratio can be estimated by dividing the uncompressed size (original image size multiplied by the bits per pixel) by the estimated compressed size.

It should, however, be noted that this model of entropy assumes no correlation between the values of individual pixels. For images, it is clear that there is generally a very high correlation between neighboring pixels. This correlation is reduced, but not removed, by the wavelet transform. Due to this correlation, encoding techniques used in practice can achieve a significantly lower entropy and file size than is the lowest bound estimated by first-order entropy.

4. Implementation Details

As an implementation of the described denoising procedure, a program was created in C#. This chapter describes the different parts of the program, provides pseudocode examples, and gives examples of running speed, complexity, and parallelizability. For the most part, the programmatic implementation is the same as the mathematical description in the theory chapter. The program serves as a proof-of-concept and is not designed to be consumer-friendly. For the full source code and compiled executables of the program and associated tools created as part of this thesis, see Appendix I.

4.1 Parameters

The program takes several parameters as arguments to control how denoising is performed. The first, mandatory parameter is the path to the image to be denoised. The program can open most common image formats, including JPEG, PNG, and BMP.

The second parameter is the amount of decomposition steps, or in other words, the wavelet transform level. It can be any positive integer, but values of 2-4 are recommended. Higher values do not necessarily yield better results.

The third parameter is the thresholding multiplier, or k, in formula (11). This can be any floating point value, but values in the range 0.5-2.0 are recommended. Higher values remove more noise, but also more detail. The amount of noise on an image does, however, not correlate with the ideal value for the multiplier. For best results, various values should be tested, but a value of 1.0 generally gives good results. Alternatively, a value of 0 skips the denoising step.

The fourth parameter is the quantization step size. It can be any positive floating point value, with higher values offering more compression, but also preserving less quality. Values around 1-64 are reasonable. Alternatively, a value of 0 skips the quantization and compression calculation steps.

4.2 Color Space Conversion

Color space conversion is the first, the last, and the simplest step. After loading the image into memory, each pixel of the input image is iterated through and converted from RGB to YC_bC_r according to (1). Similarly, after denoising, each pixel of the denoised image is iterated through and converted from YC_bC_r to RGB according to (2). These processes are separate for each pixel and take a constant amount of time per pixel. This makes the color space conversion easily parallelizable, and it runs in linear time.

4.3 Discrete Wavelet Transform

The DWT has the most differing implementation from the process described in the theory chapter. This is because a lot of efficiency would be wasted performing a full convolution on the image, then discarding half the samples in both dimensions. Instead, only every other output pixel is computed in both dimensions, thus increasing the speed by roughly a factor of 2. The following is simplified pseudocode which performs multiple levels of the DWT.

```
DWT(float[] image, int width, int height, int stride, float[] lowPassFilter,
    float[] highPassFilter)
  float[] intermediateImage = new float[image.length]
  //Filter the image horizontally
  for (int y from 0 to height)
    for (int x from 0 to width / 2)
      int targetIndex = y * stride + x
      float lowPass, highPass = 0, 0
      for (int i from 0 to lowPassFilter.length)
        float source = image[targetIndex + x + i]
        lowPass += source * lowPassFilter[i]
        highPass += source * highPassFilter[i]
      intermediateImage[targetIndex] = lowPass
      intermediateImage[targetIndex + width / 2] = highPass
  //Filter the image vertically
  for (int y from 0 to height / 2)
    for (int x from 0 to width)
      int targetIndex = y * stride + x
      float lowPass, highPass = 0, 0
      for (int i from 0 to lowPassFilter.length)
        float source = intermediateImage[targetIndex + (y + i) * stride]
        lowPass += source * lowPassFilter[i]
        highPass += source * highPassFilter[i]
      image[targetIndex] = lowPass
      image[targetIndex + height / 2 * stride] = highPass

int newWidth, newHeight = width, height //Image width and height
for (int i from 0 to level) //One loop for each level of decomposition
  for (float[] channel in image) //Process each color channel separately
```

```
    DWT(channel, newWidth, newHeight, width, [0.2, 0.6, 0.3, -0.1],
        [0.1, 0.3, -0.6, 0.2])
  newWidth /= 2  //Reduce the width and height of the area
  newHeight /= 2 //processed by the DWT by half for each level
```

The low- and high-pass filters passed to the DWT function, as well as the reconstruction filters used in the inverse DWT are the same as the ones used for the so-called orthogonal wavelets in Chapter 5. They have been normalized so that the low-pass filter coefficients sum up to 1. This is to keep the average intensity of the low frequency component, and by extension, the high frequency components derived from it, constant through all levels of decomposition. The inverse DWT function that is applied after denoising is similar to the DWT, except the indexes are different and reconstruction filters are used instead of the low- and high-pass filters.

```
IDWT(float[] image, int width, int height, int stride, float[] reconstruction1,
    float[] reconstruction2)
  float[] intermediateImage = new float[image.length]
  //Filter the image horizontally
  for (int y from 0 to height * 2)
    for (int x from 0 to width)
      int sourceIndex = y * stride + x
      float target1, target2 = 0, 0
      for (int i from 0 to reconstruction1.length)
        float source = image[sourceIndex + (i % 2 == 0 ? 0 : width) + i / 2 – 1]
        target1 += source * reconstruction1[i]
        target2 += source * reconstruction2[i]
      intermediateImage[sourceIndex + x] = target1
      intermediateImage[sourceIndex + x + 1] = target2
  //Filter the image vertically
  for (int y from 0 to height)
    for (int x from 0 to width * 2)
      int sourceIndex = y * stride + x
      float target1, target2 = 0, 0
      for (int i from 0 to reconstruction1.length)
        float source = image[sourceIndex + ((i % 2 == 0 ? 0 : height) + i / 2 –
                                          1) * stride]
        target1 += source * reconstruction1[i]
        target2 += source * reconstruction2[i]
      image[sourceIndex + y * stride] = target1
      image[sourceIndex + (y + 1) * stride] = target2
```

```
for (int i from 0 to level) //One loop for each level of decomposition
  for (float[] channel in image) //Process each color channel separately
    IDWT(channel, newWidth, newHeight, width, [0.6, -1.2, 0.4, 0.2],
          [-0.2, 0.4, 1.2, 0.6])
  newWidth *= 2  //Increase the width and height of the area
  newHeight *= 2 //processed by the IDWT two times for each level
```

As can be seen from the pseudocode, both the DWT and the IDWT have linear complexity relative to the amount of pixels in the image. For each output pixel, the amount of computations is proportional to the length of the filter, which is constant. The complexity stays linear when applying multiple levels of the transform, since the amount of pixels that need to be processed is reduced by a factor of 4 for every level, making it a converging geometric series. Further, since the value of every output pixel is not dependent on the value of any other output pixel, both of these functions are also easily parallelizable by processing each output pixel separately.

4.4 Thresholding

The main difficulty in thresholding is finding the threshold for each high frequency component. This consists of two steps. First, the noise variance is estimated from the HH1 component by computing its median value using QuickSelect, then the threshold is found for each high frequency component using that component’s variance. The following is the pseudocode for both of these processes.

```
float FindMedian(float[] values)
  int lastLength, medianIndex = 0, list.length / 2
  //Iterate until only 1 value is left or all the remaining values are equal
  while (values.length > 1 && lastLength != values.length)
    float pivot = values[0]
    List<float> smaller, larger = new List<float>(), new List<float>()
    lastLength = values.length
    for (float value in values)
      if (value <= pivot) smaller.Add(value) else larger.Add(value)
    if (medianIndex < smaller.length)
      values = smaller.array
    else
      medianIndex -= smaller.length
      values = larger.array
  return list[0]
```

Because the median is used to estimate the variance of the noise, the sign of the values is not important, only the amplitude is. Therefore, the list of values passed to the `FindMedian` function should be the absolute values of the HH1 component. The function works in linear time on average because the array of values it looks through decreases in size by a factor of 2 on average after each iteration, which is a converging geometric series. QuickSelect is not as easily parallelizable, and was not implemented as such in this thesis, but parallel implementations exist [34].

```
Threshold(float[] image, int x1, int y1, int x2, int y2, int stride,
          float noise, int level, Func threshFunc)
  //Compute the variance
  float variance = 0
  for (int y from y1 to y2)
    for (int x from x1 to x2)
      variance += (2^(level - 1) * image[y * stride + x])^2
  variance /= (x2 – x1) * (y2 – y1)
  //Compute the threshold from the variance
  float threshold = noise / Sqrt(Max(variance – noise, 0))
  //Threshold each pixel in the component
  for (int y from y1 to y2)
    for (int x from x1 to x2)
      int i = y * stride + x
      image[i] = threshFunc(image[i], threshold)

for (float[] channel in image) //Process each color channel separately
  float noise = (k * FindMedian(Abs(HH1)))^2
  //Get information about all high frequency components in a channel and process
  //each one separately according to their bounds and transform level
  for (Component comp in GetComponents(channel, width, height, level))
    Threshold(channel, comp.left, comp.top, comp.right, comp.bottom, width,
              noise, comp.level, (x, t) => (Abs(x) > t) ? (x – Sign(x) * t) : 0)
```

Finding the values of the HH1 component, as well as the component bounds and which decomposition level they are a part of has been heavily simplified in this example. The thresholding function thresholds the region specified by the bounds in-place and pixel-by-pixel. It can be passed another function, which takes into account the original value of the pixel and the threshold, to determine the new value. Because the filters have been normalized, effectively reducing the amplitude of the resulting wavelet coefficients by a factor of 2 for each decomposition level of the transform, the values of the pixels have to be multiplied by $2^{\text{level-1}}$ for the purposes of calculating their variance, in order to avoid denoising the higher levels too aggressively.

The variance calculation and the thresholding both have linear complexity, because each pixel is iterated over once – the components do not overlap and use no more pixels than the original image, regardless of how many of them there are. The thresholding is easily parallelizable because each pixel is processed separately. The parallelization of the variance calculation was not implemented in this thesis, but parallel implementations exist [35].

4.5 Compression

A typical algorithm for lossy image compression consists of two parts. First, some of the information is lost in favor of a simplified representation of the data through quantization. Secondly, the quantized data is encoded by a variable length lossless encoder into a representation that takes less bytes than the original data, which is the step that performs the actual compression. Upon requesting to view the compressed image, it is then decoded. The encoding and decoding procedures are quite complicated in the case of an effective solution, and are as such outside the scope of this thesis. However, due to the lossless nature of these steps, the effect of compression on the quality of the image can be shown by simply quantizing each component. Further, since quantization reduces the total amount of different values in the image, it also lowers its entropy, which is used to give some estimate of possible compression. The following is pseudocode for both quantization and entropy calculation.

```
Quantize(float[] image, int x1, int y1, int x2, int y2, int stride, float step)
  for (int y from y1 to y2)
    for (int x from x1 to x2)
      int i = y * stride + x
      image[i] = Round(image[i] / step) * step

for (float[] channel in image) //Process each color channel separately
  //Get information about all components in a channel and process
  //each one separately according to their bounds and transform level
  for (Component comp in GetComponents(channel, width, height, level))
    Quantize(channel, comp.left, comp.top, comp.right, comp.bottom, width,
             stepSize / 2^(comp.level - 1))
```

Similarly to thresholding, the process of getting the component bounds and decomposition levels is heavily simplified in these examples. The quantization step size is decreased by a factor of 2 each level to prioritize quality in the higher level components, which carry more important information. It is mostly a coincidence that this factor coincides with how much the amplitudes of the values have been decreased by the normalized filter coefficients. Some other quantization scheme can also be used.

If quantization would always be applied, it could be done in the same pass as thresholding, immediately after it, to save one iteration through the entire image. Because it follows a nearly identical process, it can be seen that it also has linear complexity and is easily parallelizable.

```
float CompressedSize(float[] image, int x1, int y1, int x2, int y2, int stride)
  occurances = new Dictionary<float, int>()
  for (int y from y1 to y2)
    for (int x from x1 to x2)
      occurances[image[y * stride + x]] += 1
  float size = 0
  for (int value in occurances)
    size -= value * Log2(value / (x2 - x1) / (y2 - y1))
  return size

float size = 0
for (float[] channel in image) //Process each color channel separately
  //Get information about all components in a channel and process
  //each one separately according to their bounds
  for (Component comp in GetComponents(channel, width, height, level))
    size += CompressedSize(channel, comp.left, comp.top, comp.right,
                           comp.bottom, width)
float compressionRatio = width * height * 24 / size //24 bits per pixel
```

This estimation of entropy gives both the estimated final file size and the compression ratio. The latter showing how many times the compressed file is smaller than its uncompressed counterpart. The speed and other performance-related qualities of estimating entropy are not important, as it is not a part of the denoising nor compression procedure.

4.6 Performance

One of the design goals of the program was an optimized running time. Every part of the program, and therefore the program as a whole runs in O(n), where n is the amount of pixels in the image being processed. Additionally, every part of the program that runs in linear time can be parallelized, reducing the running time by roughly a factor of the number of processing units available. Table 1 gives examples of running times of various parts of the program, as well as the running time of the program as a whole, excluding the quantization step. The tests were done on an i5-7600K CPU at 3.8GHz using a 2160 by 1440 pixel image and averaged over 10 runs.

Table 1: Average running times of various parts of the program in milliseconds.

RGB → YC_bC_r	DWT, length 2 filter	DWT, length 4 filter	Median	Thres-hold	Quantize	IDWT, length 2 filter	IDWT, length 4 filter	YC_bC_r → RGB	Total, short filter	Total, long filter
17.7ms	27.6ms	55.6ms	50.8ms	74.6ms	62.0ms	44.7ms	62.0ms	18.6ms	234ms	279ms

The results show that wavelet-based thresholding methods are fast and can be used to perform real-time image denoising. This can make the method usable in imaging devices for denoising the images right after they are taken, and for showing the user an already denoised image.

5. Denoising and Compression Results

For the purposes of testing, various images were corrupted with different types of noise and denoised using different parameters. This chapter highlights some of the more significant results, compares them with the results given by simpler methods, and gives a suggestion of which parameters are best to use. This chapter also reports the amount of compression possible using different quantization steps, and how much quality is lost using them.

5.1 Test Description

Eight different original images are used for testing, each corrupted with 4 different types of noise – Gaussian, impulse, speckle, and Poisson – of 4 different intensities for a total of 128 noisy images. Each noisy image is denoised with the program with 20 different noise intensity estimation parameter values, from 0.1 to 2.0, 2 different types of wavelets, and hard, moderate, and soft thresholding functions, for a total of 120 different results on a single image, and 15360 results overall. For a full list of these results, see Appendix II. The denoising performance of simple 3×3 mean and median filters is also shown. Two popular free software, Paint.NET [37] and FastStone Image Viewer [38], that have noise removal as part of their functionality were tested as well, but their objective performance did not exceed that of even the mean filter, so they have been excluded from the results.

The images used for testing range in size from 1920 by 1080 pixels to 2160 by 1440 pixels, and are downscaled from larger high quality images so that they have no visible noise left on them that might have a negative effect on the test results. A few of the images used can be seen in Figure 10. Each image is decomposed using four levels of wavelet transform, and has no quantization applied.

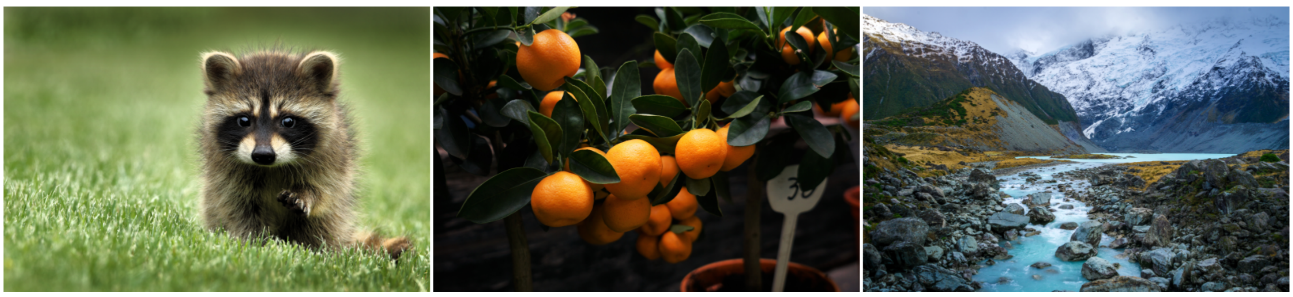


Figure 10: Original noise-free images used for testing. From left to right: Raccoon [27], Oranges [38], River [39].

Each result is compared to the corresponding original image using two objective quality metrics with the best results marked in bold. The first, located higher in each cell in the tables of results is the peak signal-to-noise ratio (PSNR). PSNR is a simple quality metric based on the mean square error on a logarithmic scale, defined as

$$10\log_{10}\left(\frac{255^2}{\mathrm{MSE}(a,b)}\right)$$

where 255 is the maximum pixel value (peak signal), and MSE is the mean square error (noise) of the pixel values between images *a* and *b*.

The second quality metric is the structural similarity (SSIM) index, designed to better correlate with human perception of image quality [36], because PSNR overvalues the quality of, for example, overly smoothed images. This can be seen in the relatively high PSNR results of the mean filter in the following comparisons. It is also the reason why SSIM results are favored in deciding which parameters produce good results.

The following subchapters show the results of denoising the Raccoon image corrupted with 4 different types of noise, at 4 increasing intensities. For each noisy image, the performance of the median and mean filters is shown, as well as the performance of the original BayesShrink, which uses soft thresholding. The choice of $k = 1$ and soft thresholding for the modified BayesShrink as the recommended default values is explained after the results. Finally, the best result from all the test data in terms of SSIM is also given. Manually adjusting k for optimal perceived quality when denoising would probably yield a result close to this near-optimal. An additional table featuring a different test image is provided for Gaussian and Poisson noise, as the results of the program on those types of noise is more interesting. For each type of noise, subsections of the images corresponding to the second row of one of the tables are also shown, for the purpose of visual comparison.

5.2 Gaussian Noise

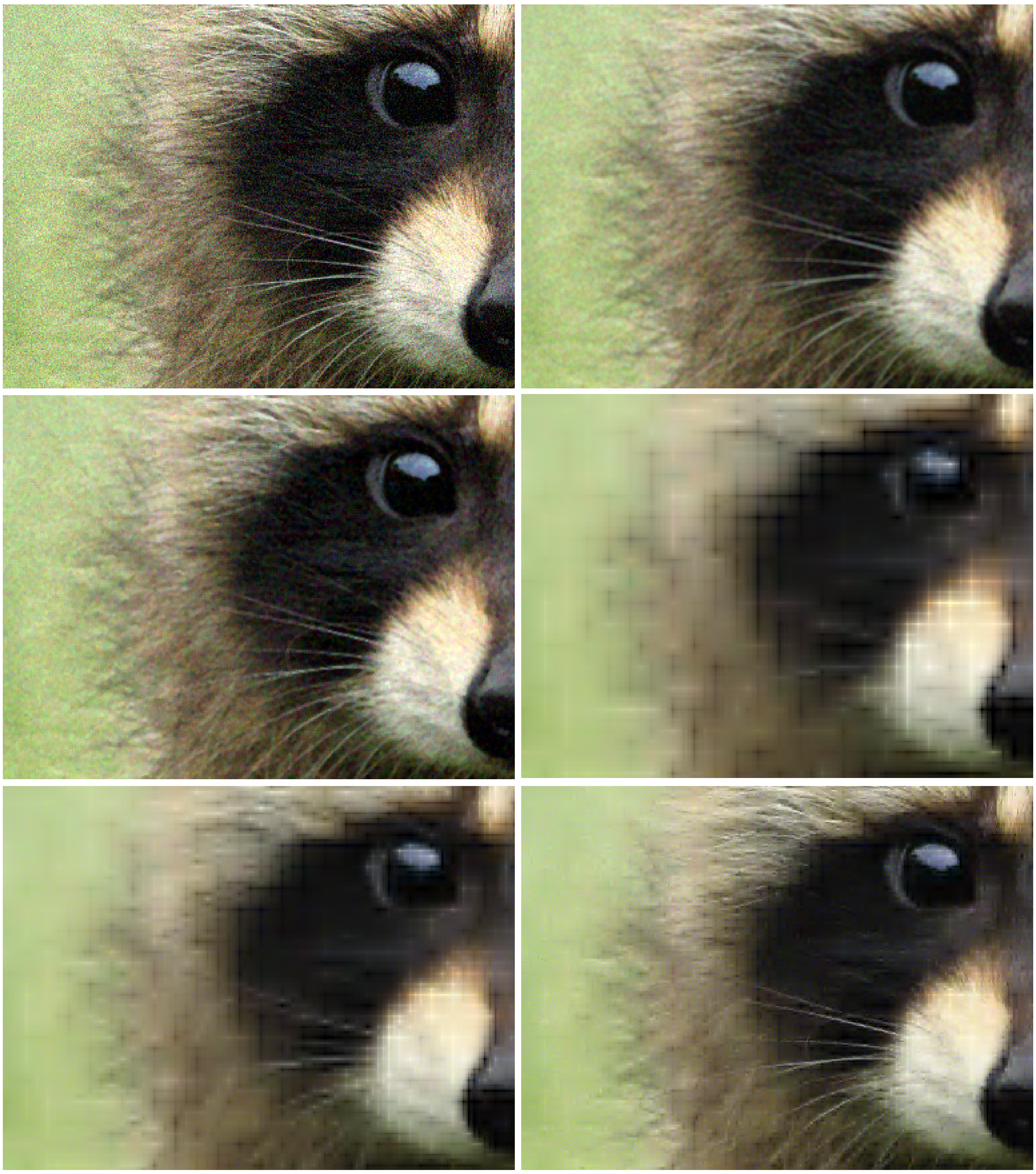


Figure 11: Quality comparison of various denoising methods on the Raccoon image. The images are row-by-row as follows: Original corrupted by Gaussian noise, σ=20. 3×3 mean filtered. 3×3 median filtered. Original BayesShrink, orthogonal wavelet, soft thresholding. BayesShrink, k=1.0, orthogonal wavelet, soft thresholding. BayesShrink, k=0.7, orthogonal wavelet, moderate thresholding.

Table 2: PSNR and SSIM values of various denoising results on the Raccoon image corrupted with Gaussian noise, σ = 10, 20, 30, 40.

σ	Noisy Original	Mean Filter	Median Filter	Original BayesShrink	BayesShrink k=1.0	BayesShrink Best SSIM
10	28.17 72.32%	32.31 89.29%	31.88 87.70%	33.12 91.25%	**33.67** 85.85%	33.62 **91.51%**
20	22.21 44.76%	**29.58** 79.31%	28.42 74.19%	26.58 73.47%	27.16 78.83%	26.41 **85.16%**
30	18.78 30.22%	27.12 68.06%	25.66 60.67%	25.32 69.07%	26.31 72.60%	**28.53** **79.44%**
40	16.42 22.01%	25.13 57.95%	23.48 49.40%	24.81 67.66%	**25.35** 69.37%	23.56 **75.74%**

Table 3: PSNR and SSIM values of various denoising results on the Oranges image corrupted with Gaussian noise, σ = 10, 20, 30, 40.

σ	Noisy Original	Mean Filter	Median Filter	Original BayesShrink	BayesShrink k=1.0	BayesShrink Best SSIM
10	29.03 67.88%	36.49 91.62%	35.53 90.32%	33.05 89.63%	35.36 92.26%	**36.91** **92.91%**
20	23.55 38.96%	31.14 78.76%	30.40 74.84%	29.52 80.45%	30.75 83.96%	**31.48** **85.67%**
30	20.36 24.29%	27.58 65.74%	27.28 60.11%	27.20 73.36%	**28.14** 76.94%	28.12 **78.52%**
40	18.10 16.46%	24.97 54.53%	25.06 48.35%	25.65 69.59%	26.04 66.78%	**26.35** **72.85%**

Table 2 and Table 3 show that for Gaussian noise for both the Raccoon and Oranges images, the default modified BayesShrink usually produces better results than both the original BayesShrink and the median and mean filters, sometimes significantly. The optimal choice further improves on this, always producing the best results, sometimes marginally, sometimes significantly. Figure 11 shows that median and mean filtering are unable to remove all the noise while the original BayesShrink removes the noise, but also far too much detail. The default modified BayesShrink also removes all the noise, keeping significantly more but still too little detail. Finally, the optimal choice keeps a good amount of detail while still succeeding in removing all the noise.

5.3 Impulse Noise

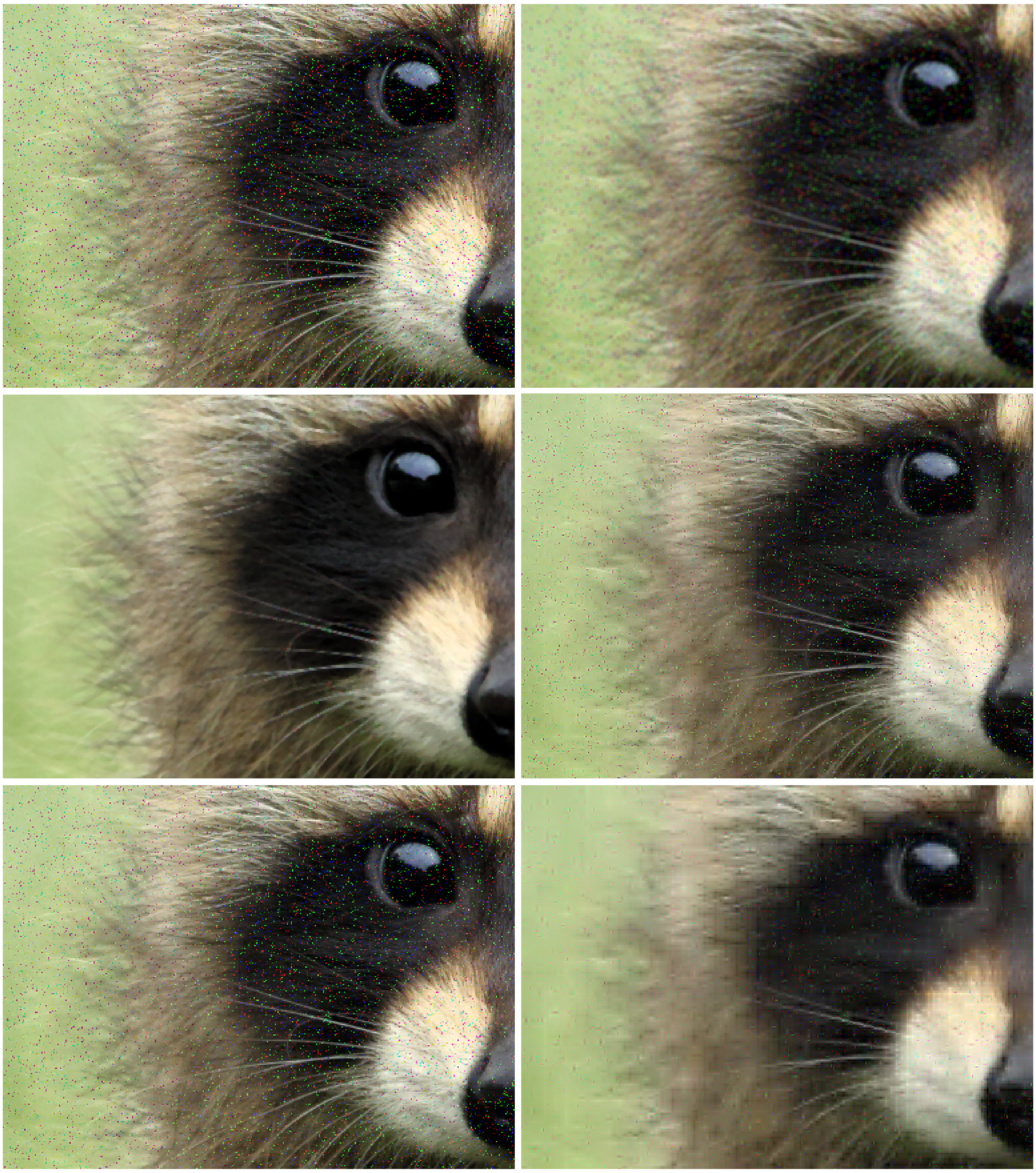


Figure 12: Quality comparison of various denoising methods on the Raccoon image. The images are row-by-row as follows: Original corrupted with Impulse noise affecting 5% of the pixels at random. 3×3 mean filtered. 3×3 median filtered. Original BayesShrink, orthogonal wavelet, soft

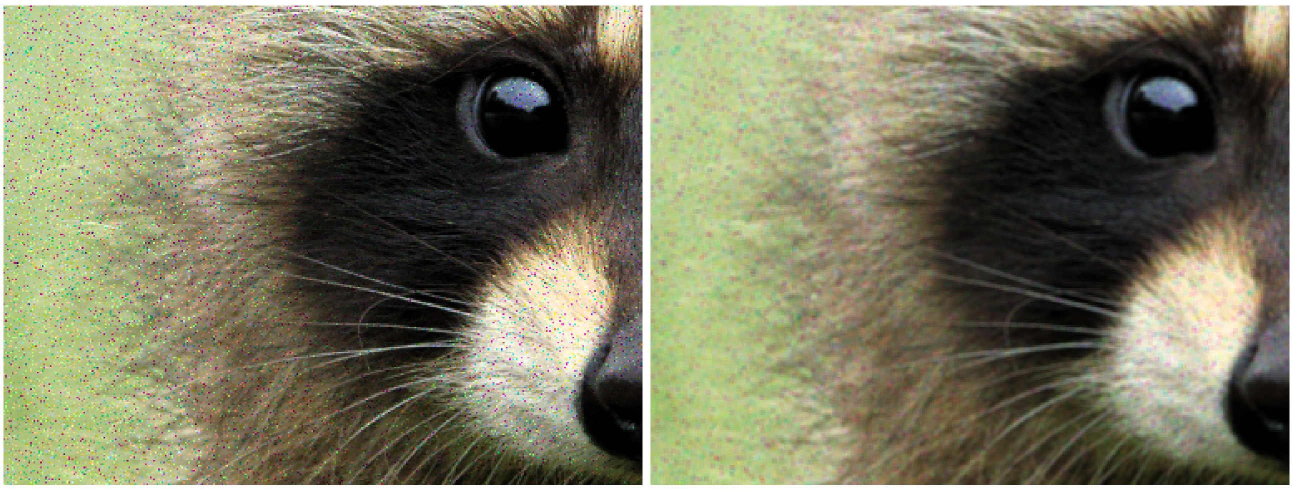
thresholding. BayesShrink, k=1.0, orthogonal wavelet, soft thresholding. BayesShrink, k=2.0, orthogonal wavelet, soft thresholding.

Table 4: PSNR and SSIM values of various denoising results on the Raccoon image corrupted with impulse noise affecting 2.5, 5, 10, or 20 percent of the pixels at random.

%	Noisy Original	Mean Filter	Median Filter	Original BayesShrink	BayesShrink k=1.0	BayesShrink Best SSIM
2.5	25.02 66.58%	30.97 86.23%	**34.35** **94.25%**	27.37 70.20%	26.05 68.32%	27.58 82.79%
5	21.97 49.81%	29.07 80.04%	**34.13** **94.15%**	24.52 65.78%	23.15 56.58%	26.01 82.07%
10	18.97 33.88%	26.49 70.37%	**33.61** **93.83%**	25.75 69.63%	22.62 45.17%	27.18 78.68%
20	15.96 21.06%	23.18 57.06%	**31.60** **91.23%**	23.72 70.53%	20.94 45.84%	22.97 74.01%

Table 4 shows that the median filter produces significantly better results on impulse noise than any other method, restoring even heavily noisy images to a good state. Figure 12 shows how all other methods are unable to remove the noise, even if heavily blurring the image, while the median filter removes the noise and keeps the image relatively sharp. This is expected, as impulse noise usually has a very high intensity in the wavelet domain, and wavelet-based thresholding is only effective at removing noise with a lower intensity than the image details.

5.4 Speckle Noise



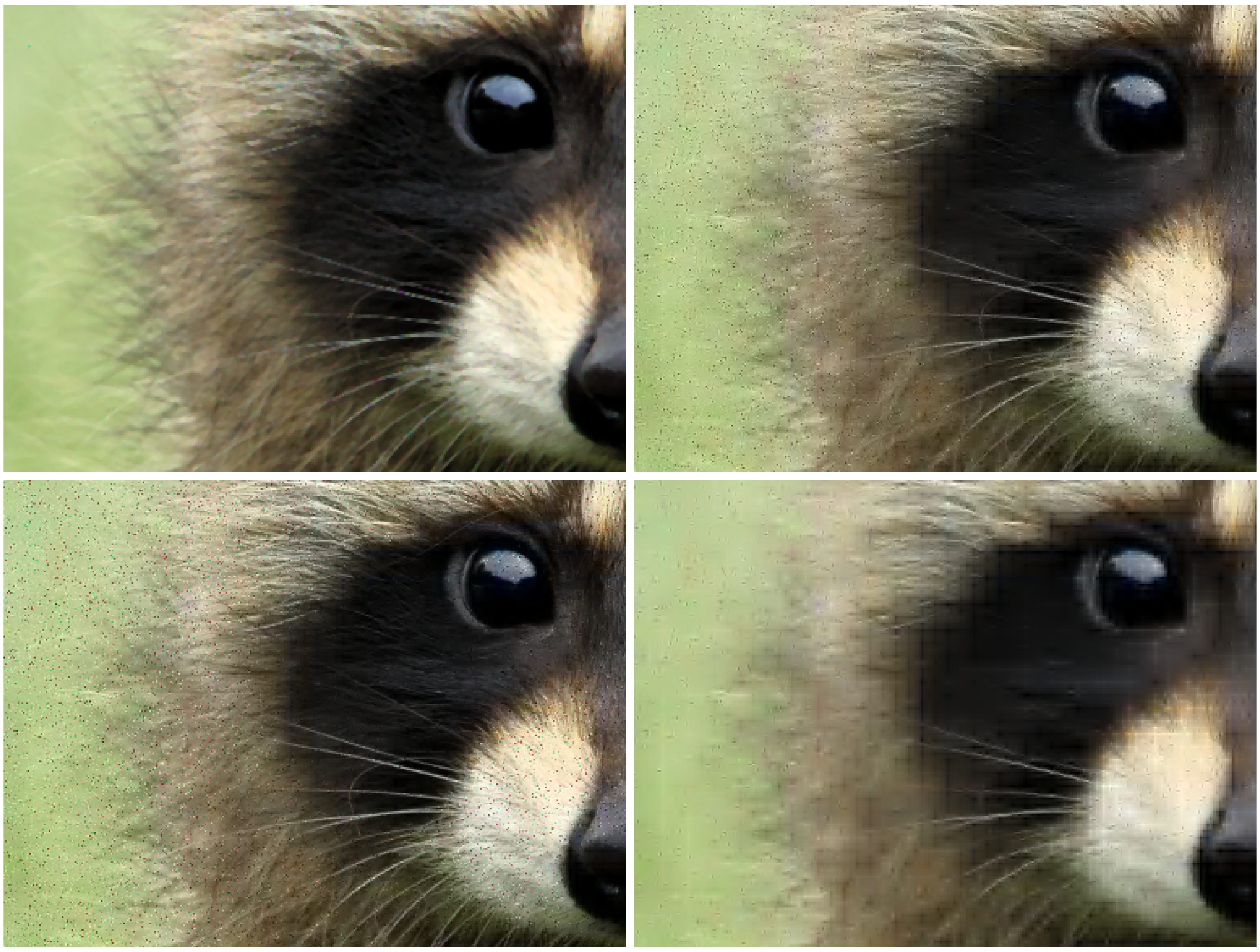


Figure 13: Quality comparison of various denoising methods on the Raccoon image. The images are row-by-row as follows: Original corrupted with speckle noise affecting 10% of the pixels at random. 3×3 mean filtered. 3×3 median filtered. Original BayesShrink, orthogonal wavelet, soft thresholding. BayesShrink, k=1.0, orthogonal wavelet, soft thresholding. BayesShrink, k=2.0, orthogonal wavelet, soft thresholding.

Table 5: PSNR and SSIM values of various denoising results on the Raccoon image corrupted with speckle noise affecting 2.5, 10, 20, or 30 percent of the pixels at random.

%	Noisy Original	Mean Filter	Median Filter	Original BayesShrink	BayesShrink k=1.0	BayesShrink Best SSIM
2.5	27.64 72.95%	32.15 88.27%	**34.40** **94.27%**	28.88 78.28%	28.50 75.51%	29.87 83.33%
10	21.65 44.02%	29.21 76.38%	**33.93** **94.02%**	27.39 59.74%	24.96 50.35%	29.25 79.90%

20	18.63	26.96	**33.08**	25.00	24.90	22.33
	31.62%	65.77%	**93.03%**	68.42%	70.57%	77.19%
30	16.87	25.44	**31.59**	24.77	21.16	17.92
	25.75%	58.44%	**89.59%**	68.41%	70.89%	74.62%

Very similarly to impulse noise, Table 5 shows that the median filter produces significantly better results on speckle noise than any other method, restoring even heavily noisy images to a good state. Figure 13 shows how most other methods are unable to remove the noise. Only the optimal BayesShrink manages to get rid of most of the noise, at the cost of heavily blurring the image. The median filter both completely removes the noise and keeps the image relatively sharp. A more specialized median filter would probably do an even better job, and is a more suitable method for removing high-intensity sparse noise than thresholding.

5.5 Poisson Noise

Table 6: PSNR and SSIM values of various denoising results on the Raccoon image corrupted with Poisson noise with an average amount of 225, 42, 18, or 8 photons per pixel.

P	Noisy Original	Mean Filter	Median Filter	Original BayesShrink	BayesShrink k=1.0	BayesShrink Best SSIM
225	29.54	32.67	32.44	31.19	33.06	**33.31**
	75.11%	89.78%	88.38%	88.89%	89.51%	**91.41%**
42	22.43	29.71	28.53	28.24	**29.86**	29.74
	45.38%	78.00%	72.50%	79.69%	75.58%	**83.53%**
18	19.01	27.25	25.66	27.71	25.10	**28.16**
	32.34%	66.68%	58.83%	77.96%	47.55%	**78.32%**
8	15.78	24.41	22.41	**24.63**	21.25	16.91
	22.33%	53.29%	43.72%	67.90%	69.55%	**73.28%**

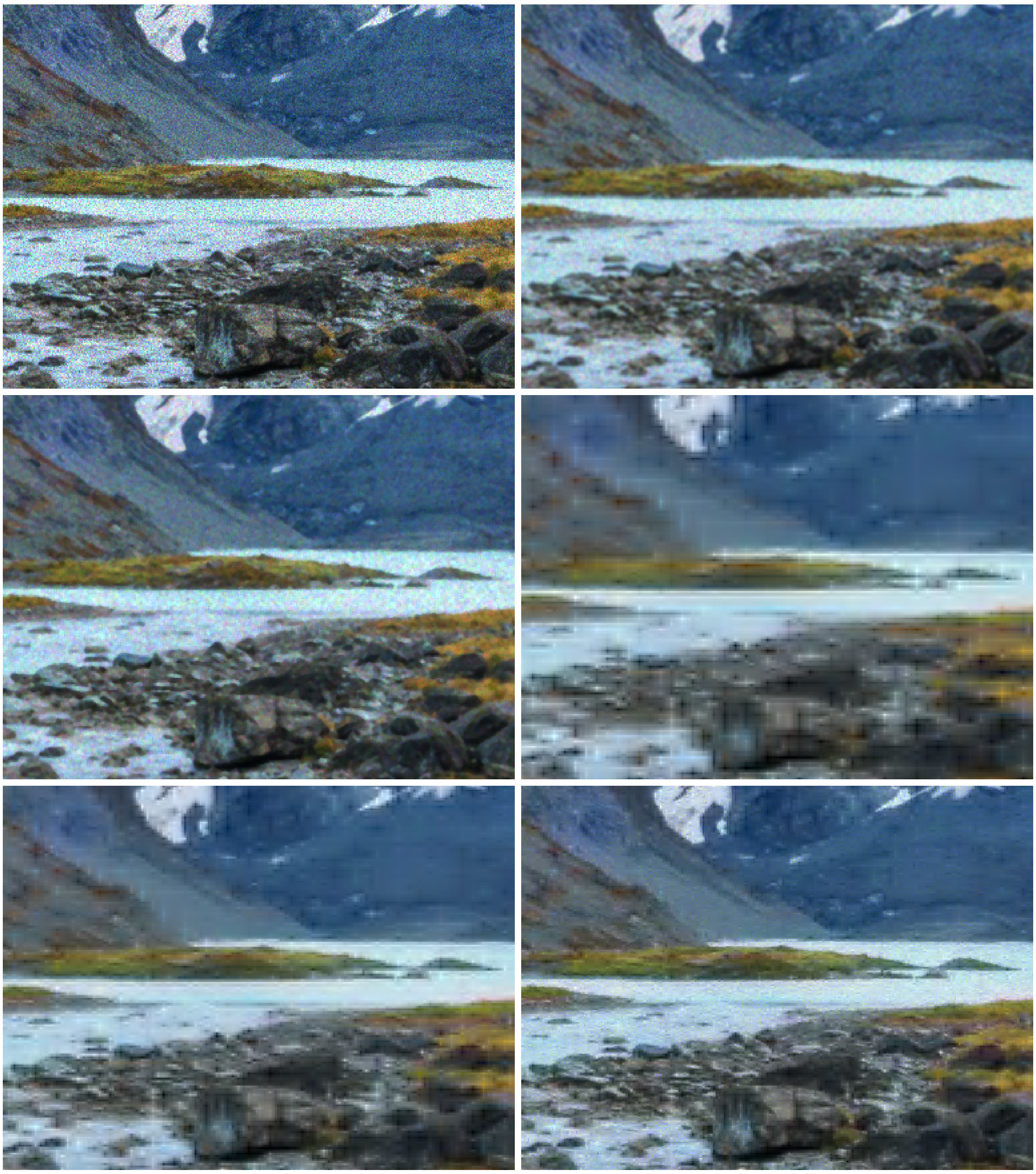


Figure 14: Quality comparison of various denoising methods on the River image. The images are row-by-row as follows: Original corrupted by Poisson noise, average of 42 photons per pixel. 3×3 mean filtered. 3×3 median filtered. Original BayesShrink, orthogonal wavelet, soft thresholding. BayesShrink, k=1.0, orthogonal wavelet, soft thresholding. BayesShrink, k=0.7, orthogonal wavelet, soft thresholding.

Table 7: PSNR and SSIM values of various denoising results on the River image corrupted with

Poisson noise with an average amount of 225, 42, 18, or 8 photons per pixel.

P	Noisy Original	Mean Filter	Median Filter	Original BayesShrink	BayesShrink k=1.0	BayesShrink Best SSIM
225	30.03 88.26%	32.24 **92.25%**	32.30 92.07%	**32.36** 92.13%	32.29 90.56%	32.20 92.16%
42	23.21 72.59%	**29.68** **86.75%**	28.61 84.09%	24.69 66.20%	26.81 77.13%	27.21 81.73%
18	19.93 61.78%	**27.34** **81.50%**	25.95 77.26%	22.55 58.49%	22.62 69.11%	22.76 74.34%
8	16.75 49.13%	**24.38** **74.45%**	22.91 68.61%	20.75 44.99%	21.31 48.31%	22.82 67.14%

Table 6 shows similar results for Poisson noise as for Gaussian noise. For the Raccoon image, the default modified BayesShrink usually produces better results than both the original BayesShrink and the median and mean filters, sometimes significantly. The optimal choice further improves on this, always producing the best results, sometimes marginally, sometimes significantly. However, as seen in Table 7, for the River image, the mean filter almost always produces the best results. This is not inherent to Poisson noise – the mean filter also produces better results for Gaussian noise on the same image. This is because the River image has very few flat-colored areas and lots of low intensity details. As can be seen in Figure 14, a high thresholding multiplier removes the noise, but also too much detail. A low thresholding multiplier removes most of the low intensity details first before removing most of the higher intensity noise. This produces a better result, but is still not as good as the mean or even the median filter, which do not remove as much noise, but make up for it with the lack of removed detail.

5.6 Compression

Compression is most efficient after denoising, as the denoising process has already eliminated a lot of information, which improves the compression ratio. The following Table 8 shows the effect of various levels of quantization on quality, as well as how many times smaller the resulting image would be. A single example of JPEG2000 is also included to show that these estimated results are reasonable realistically. The images corresponding to the second row of the table can be seen in Figure 15.

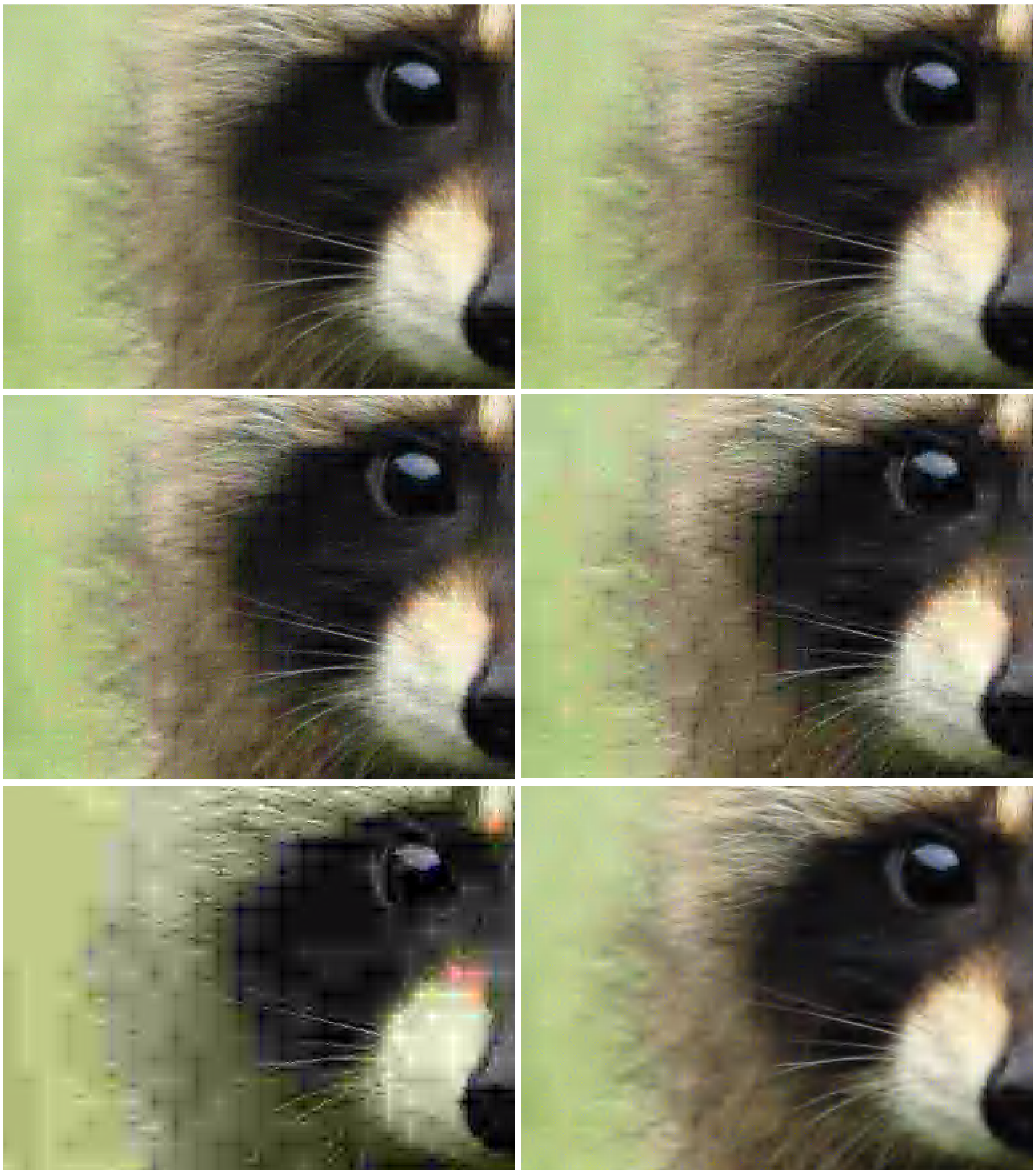


Figure 15: Quality comparison of various amounts of quantization on the denoised Raccoon image. The images are row-by-row as follows: Denoised with no quantization. Step size 8. Step size 16. Step size 32. Step Size 64. JPEG2000, quality 10.

Table 8: PSNR, SSIM values, and compression ratios of different step sizes on the optimally denoised Raccoon image corrupted with Gaussian noise, σ = 10, 20, 30, 40.

σ	No Quantization	Step Size 8	Step Size 16	Step Size 32	Step Size 64	JPEG2000, Quality 10
	33.62	32.76	31.46	27.96	23.05	32.50
10	91.51%	89.46%	87.42%	81.40%	63.84%	90.04%
	3.28	48.9	85.2	172	675	100
	26.41	25.86	25.45	27.50	25.12	29.02
20	85.16%	83.97%	82.28%	78.01%	70.33%	82.60%
	3.24	13.0	30.7	179	561	100
	28.53	28.52	27.24	27.18	24.19	28.74
30	79.44%	78.26%	74.17%	75.07%	68.81%	80.63%
	3.29	79.7	174	293	716	100
	23.56	23.57	22.99	22.86	24.20	25.80
40	75.74%	74.46%	72.18%	67.11%	66.79%	73.17%
	3.25	17.4	30.4	85.4	675	100

As can be seen from Table 8, the quality of images goes down the more quantization and compression is applied. However, a compression ratio that is around 10 times better than what can be achieved on an image with no quantization still produces a result that is almost not distinguishably worse. This can be seen from Figure 15, where there is little difference between the top left image with no quantization and the 10 times more compressed middle left image, with a quantization step size of 16. Increasing the step size to 64 allows the compression ratio to reach hundreds, but at the significant cost of details and color information.

5.7 Overall Results

Wavelet-based denoising is generally effective on Gaussian and Poisson noise, which both affect all pixels in an image. However, for cases where the image has many low intensity details and few areas which are roughly the same color, wavelets can fail to remove noise more effectively than a mean filter, because they remove too many of the details. For sparse noise, such as impulse and speckle noise, which only affects some pixels, median filtering is incredibly effective, while thresholding by nature is incredibly ineffective in removing these high intensity deviations from the rest of the image.

Focusing on Gaussian and Poisson noise, as these are the types of noise wavelet-based thresholding is effectively capable of removing, the larger tables in Appendix II give an idea what might be an optimal choice of wavelet, thresholding type, and thresholding multiplier. Haar wavelets almost always produce worse results than the length 4 orthogonal wavelets. Similarly, hard thresholding almost always produces worse results than moderate and soft thresholding. Moderate and soft thresholding have very similar performance, and the marginally better result is sometimes in favor of one thresholding type, sometimes in the favor of the other. Although, due to the relative simplicity of soft thresholding in comparison to moderate thresholding, it could be argued that soft thresholding is the better choice. Finally, the choice of the thresholding multiplier k is the most difficult. The optimal value for k varies based on the image and how much noise there is, as well as the wavelet and thresholding type, with no apparent pattern. Manually choosing this value would increase denoising quality, but a value of 1 generally produces an acceptable result.

6. Conclusion

This thesis studied methods for improving the quality of images by removing noise using wavelet transforms and thresholding. The efficiency of combining wavelet transform based denoising and compression as a computationally cheap process was explored. A C# implementation was developed as a proof-of-concept of both the effectiveness and the speed of the method.

The thesis surveyed different types of noise. A short description of various denoising schemes and their effectiveness at removing different types of noise was presented. A description of each important step of the denoising process and the motivation behind them was given in Chapter 3, with a focus on the wavelet transform process. A new thresholding type and a modification to the BayesShrink method that allows for configuration via a parameter were presented. The implementations and experimental performance of each of these steps were demonstrated.

Finally, the experimental results of using the developed method with different types of wavelets, thresholding, and different values of the configurable parameter were shown, and the optimal choices were highlighted. The effect of compression on the quality of images was analyzed. The implementation of different compression techniques in the context of denoising is a potential avenue for future research.

The developed method was shown to successfully remove both Gaussian and Poisson noise. It could be applicable to real-time image denoising. The proposed thresholding type was shown to be competitive with previously existing thresholding types, and the modification to BayesShrink produced better results than the original method.

7. References

[1] Pizurica A., Wink A.M., Vansteenkiste E., Philips W., Roerdink B.J. A review of wavelet denoising in MRI and ultrasound brain imaging. *Current medical imaging reviews*, 2006, vol. 2, no. 2, pp. 247-260.

[2] Jansen M. Noise reduction by wavelet thresholding. Springer Science & Business Media. 2012.

[3] Patidar P., Gupta M., Srivastava S., Nagawat A.K. Image de-noising by various filters for different noise. *International journal of computer applications*, 2010, vol. 9, no. 4, pp. 45-50.

[4] Verma R., Ali J. A comparative study of various types of image noise and efficient noise removal techniques. *International journal of advanced research in computer science and software engineering*, 2013, vol. 3, no. 10, pp. 617-622.

[5] Linear Filters. https://www8.cs.umu.se/kurser/TDBD09/VT02/cvbook/ch08linearfilters.pdf (21.01.2019)

[6] Perona P., Shiota T., Malik J. Anisotropic diffusion. *Geometry-driven diffusion in computer vision*. Dordrecht: Springer, 1994, pp. 73-92.

[7] Fisher R., Perkins S., Walker A., Wolfart E. Median filter. 2003. https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm (22.01.2019)

[8] Hardie R.C., Barner K.E. Rank conditioned rank selection filters for signal restoration. *IEEE transactions on image processing*, 1994, vol. 3, no. 2, pp. 192-206.

[9] Narayanan S.A., Arumugam G., Bijlani K. Trimmed median filters for salt and pepper noise removal. *International journal of emerging trends & technology in computer science*, 2013, vol. 2, no. 1, pp. 35-40.

[10] Rao K.R., Yip P. Discrete cosine transform: algorithms, advantages, applications. Academic press. 2014.

[11] Vasconcelos N. Discrete cosine transform. http://www.svcl.ucsd.edu/courses/ece161c/handouts/DCT.pdf (24.01.2019)

[12] Pogrebnyak O., Lukin V.V. Wiener discrete cosine transform-based image filtering. *Journal of electronic Imaging*, 2012, vol. 21, no. 4.

[13] Ponomarenko N.N., Lukin V.V., Zelensky A.A., Astola J.T., Egiazarian K.O. Adaptive DCT-based filtering of images corrupted by spatially correlated noise. *Image processing: algorithms and systems VI*, 2008, vol. 6812.

[14] Discrete wavelet transform. https://www.cs.toronto.edu/~mangas/teaching/320/slides/CSC320L11.pdf (24.01.2019)

[15] Mohideen S.K., Perumal S.A., Sathik M.M. Image de-noising using discrete wavelet transform. *International journal of computer science and network security*, 2008, vol. 8, no. 1, pp. 213-216.

[16] Heil C.E., Walnut D.F. Continuous and discrete wavelet transforms. *SIAM review*, 1989, vol. 31, no. 4, pp. 628-666.

[17] Graps A. An introduction to wavelets. *IEEE computational science and engineering*, 1995, vol. 2, no. 2, pp. 50-61.

[18] Chang S.G., Yu B., Vetterli M. Adaptive wavelet thresholding for image denoising and compression. *IEEE transactions on image processing*, 2000, vol. 9, no. 9, pp. 1532-1546.

[19] Kenterlis P., Salonikidis D. Evaluation of wavelet domain methods for image denoising. *1st international scientific conference eRA*, 2006.

[20] McHugh S. Understanding digital camera sensors. https://www.cambridgeincolour.com/tutorials/camera-sensors.htm (20.04.2019)

[21] Vasilyev M. Blue, red and green parrot. 2015. https://unsplash.com/photos/gGC63oug3iY (20.04.2019)

[22] T. 871: Information technology – digital compression and coding of continuous-tone still images: JPEG file interchange format (JFIF). *ITU-T*, 2012. https://www.itu.int/rec/T-REC-T.871 (21.04.2019)

[23] Mallat S.G. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis & machine intelligence*, 1989, vol. 2, no. 7, pp. 674-693.

[24] Bocharova I. Compression for multimedia. Cambridge University Press. 2010.

[25] Lin H.Y. An introduction to wavelets. 2013.

[26] Damelin S.B., Miller W. The mathematics of signal processing. Cambridge University Press. 2012.

[27] Wang R. 2008. http://fourier.eng.hmc.edu/e161/lectures/wavelets/node7.html (25.04.2019)

[28] Bendig G. Raccoon walking on lawn grass. 2017. https://unsplash.com/photos/6GMq7AGxNbE (25.04.2019)

[29] Fletcher A.K., Goyal V.K., Ramchandran K. Iterative projective wavelet methods for denoising. *Wavelets: applications in signal and image processing X,* 2003, vol. 5207, pp. 9-16.

[30] Donoho D.L., Johnstone I.M. Ideal spatial adaption by wavelet shrinkage. *Biometrika*, 1994, vol. 81, no. 3, pp. 425-455.

[31] Donoho D.L., Johnstone I.M. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the american statistical association*, 1995, vol. 90, no. 432, pp. 1200-1224.

[32] T. 800: Information technology – JPEG 2000 image coding system: Core coding system. *ITU-T*, 2015. https://www.itu.int/rec/T-REC-T.800 (05.05.2019)

[33] Borda M. Fundamentals in information theory and coding. Springer Science & Business Media. 2011.

[34] Zadeh R., Santucci A. Distributed algorithms and optimization. 2017. https://stanford.edu/~rezab/dao/notes/lecture04/cme323_lec4.pdf (06.05.2019)

[35] Chan T.F., Golub G.H., LeVeque R.J. Updating formulae and a pairwise algorithm for computing sample variances. *COMPSTAT 1982 5th symposium held at Toulouse*, 1982, pp. 30-41.

[36] Wang Z., Simoncelli E.P., Bovik A.C. Multi-scale structural similarity for image quality assessment. *The thrity-seventh asilomar conference on signals, systems & computers*, 2003, vol. 2, pp. 1398-1402.

[37] Paint.NET. https://www.getpaint.net/ (04.05.2019)

[38] FastStone Image Viewer. https://www.faststone.org/ (04.05.2019)

[39] Hesry E. Orange tree. 2016. https://unsplash.com/photos/_omuigahLco (07.05.2019)

[40] Postma D. River with gray rocks near mountain covered in snow. 2017. https://unsplash.com/photos/XqtJY5gTo5k (07.05.2019)

Appendix

I. Source code and executables of the created programs

https://github.com/TornOne/CS-Thesis-2019

II. Full list of PSNR and SSIM results of the images denoised with the created program

https://docs.google.com/spreadsheets/d/e/2PACX-1vRQjp5DeS34gn_HdATH-
Tl4nUA1WzAtfs_UzGNi4g7n_7ykYTwcXK10RWkHiBWy4Vx0GoAJaPpetUyC/pubhtml

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Ott Adermann,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Wavelet-based image denoising,

   supervised by Irina Bocharova and Vitaly Skachek.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Ott Adermann

10/05/2019