An Analysis of Image Denoising Techniques

Benjamin A. Schifman
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85719
bschifman@email.arizona.edu

Abstract

Digital denoising techniques are used to filter out unwanted noise in a signal. In images, noisy signals are present in the form of non coherent Salt & Pepper noise and Gaussian noise to coherent noise introduced inherently from the imager or from signal processing algorithms. This paper examines some of the common methods for removing unwanted noise, along with implementing more adept filtering techniques in the form wavelet filtering.

1 Introduction

This paper explores noise filtering techniques implemented in Python and an available Python image processing library $OpenCV$. The filters are implemented on images with random Gaussian noise and Salt & Pepper noise, and their output Peak Signal to Noise Ratios are compared. The two $python$ files are detailed in section 5. The underlying input image $FIGUREX$ and it's corresponding noisy images $NoisyIMAGES$ are in this figure. The standard filters that were implemented using the $OpenCV$ library were the: *blur, gaussian blur, median* and *bilateral* filter and the filter windows were varied in order to generate the optimal resulting filter. The Haar Wavelet Transform was implemented by hand, and the Daubechies 4 wavelet was implemented using the available $PyWavelets$ library after an attempt by hand of the algorithm was unsuccessful.

2 Methods/Approach

2.1 Noise Generation

Two images were generated with different noise distributions for the purpose of analyzing the efficacy of the different applied filtering techniques. The first noisy image was generated with a normal Gaussian distribution that had been scaled by a factor of 10. The noise was scaled in order to be more visually evident in the image along with increasing the noise power in the image. Gaussian noise is generally a common form of noise that principally arises in images during acquisition and is caused by a number of factors, a few being poor illumination, high circuitry temperature, and electronic interference. The second noisy image was generated through adding a 0.4% Salt & Pepper (S&P) distribution. The S&P noise added was equally distributed "Salt" white pixels, and "Pepper" black pixels. S&P noise potentially occurs in images were intermittent and non-reliable image communication systems are present as they can elicit sharp and sudden disturbances in the image signal. The following Figure 1 depicts the noise induced images.

2.2 Peak Signal to Noise Ratio

In order to analyze the utility of the aforementioned filtering techniques the Peak Signal to Noise Ratios (PSNR) for each filtered image was calculated. The PSNR of an image is the maximum power

(a) cman w Gaussian Noise (b) cman w S&P Noise

Figure 1: Noisy cman Images

of an image and the power of the image noise. The following equation 1 details the calculations for PSNR.

$$PSNR = 10 * log_{10}\Big(\frac{MAX^2}{MSE}\Big) \tag{1}$$

Where MAX is the maximum grayscale pixel value for the image which in this case is an unsigned 8-bit image with a maximum value of 255, and the MSE is the Mean Squared Error between the filtered output image and the noisy image. By taking the maximum pixel

2.3 Haar Wavelet Transform

The Haar Wavelet Transform (HWT), proposed by the Hungarian mathematician Alfréd Haar is a computationally efficient method for analyzing the local aspects of an image. A key advantage to the use and implementation of the HWT is in it simplicity to compute, along with it being easier to understand than most other wavelet transforms. A great benefit to using the HWT is that it is effective in signal and image compression and the algorithm is a memory efficient in that all of the calculations can be done in place, however, the software depicted below uses temporary vectors for ease of following. The HWT can be expressed in terms of matrix operations such that the Forward HWT is:

$$y_n = W_n v_n \tag{2}$$

where v_n is the input vector to be transformed, W_n is the HWT matrix, and y_n is the transformed output vector. This calculation can be even more easily illustrated in the form of the following expanded graphic in Figure 2 that details the Haar transform of an input vector of length 8, its corresponding Haar Matrix W_8, and its output vector which is simply the mean or trend of two sequential pixel elements for the first half of the vector, and then the second half of the values are a running difference or fluctuation of two sequential pixel elements.

The algorithm for this vector-wise HWT has been implemented by hand in the software and can be found in the $OneD\_HWT$ function in the $DWT.py$ file. On a 2D image the HWT is first calculated on the rows of the image, and then the HWT is calculated on the columns of the image. This 2D implementation can be found in the $TwoD\_HWT$ in the $DWT.py$ file. This 2D function also accounts for multiple iterations of the HWT in that each following iteration will reduce the image into smaller sub-wavelets. The Inverse HWT can be calculated by the following equation with the same variables:

$$x_n = H^T y_n \tag{3}$$

$$\tilde{W}_8\mathbf{v} = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ -1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1/2 & 1/2 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \end{bmatrix} = \begin{bmatrix} (v_1+v_2)/2 \\ (v_3+v_4)/2 \\ (v_5+v_6)/2 \\ (v_7+v_8)/2 \\ (v_2-v_1)/2 \\ (v_4-v_3)/2 \\ (v_6-v_5)/2 \\ (v_8-v_7)/2 \end{bmatrix} = \mathbf{y}$$

Figure 2: Haar Transform

The HWT can be

2.4 Daubechies Wavelet Transform

2.5 Thresholding

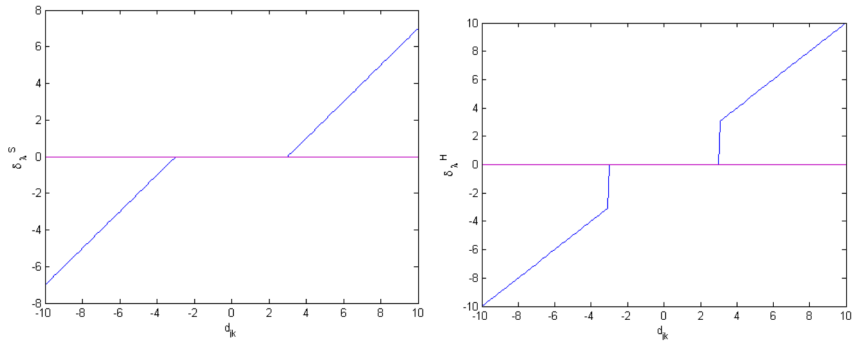
Pixel thresholding is often a simple but efficient non-linear denoising approach in the application of a wavelet transform. The thresholding is performed on the wavelet transformed image, and then the image is inverse wavelet transformed to yield a filtered output image. To determine the best threshold value to set, the $detThreshold$ function from $dataSetup.py$ was used. This function first iterates through thresholding values, then thresholds the Wavelet Transformed image, then inverse Wavelet Transforms the thresholded image, and finally calculates the Peak Signal to Noise ratio of the filtered image. The threshold is chosen based off of the one that results in the largest Peak Signal to Noise Ratio. The following equations (4) and (5) detail the Hard and Soft Thresholding calculations respectively, with Figure 3 depicting the thresholds.

Hard Thresholding

$$D^H(d|T) = \begin{cases} 0, & \text{if } |d| \leq T \\ d, & \text{if } |d| > T \end{cases} \tag{4}$$

Soft Thresholding

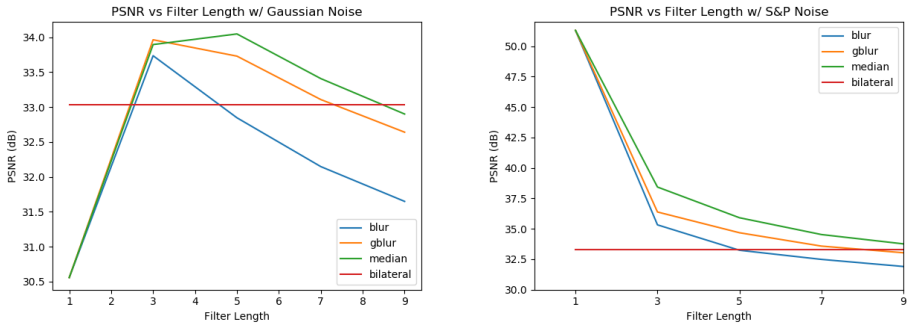
$$D^S(d|T) = \begin{cases} 0, & \text{if } |d| \leq T \\ d-T, & \text{if } d > T \\ d+T, & \text{if } d < -T \end{cases} \tag{5}$$



(a) Hard Threshold. (b) Soft Threshold.

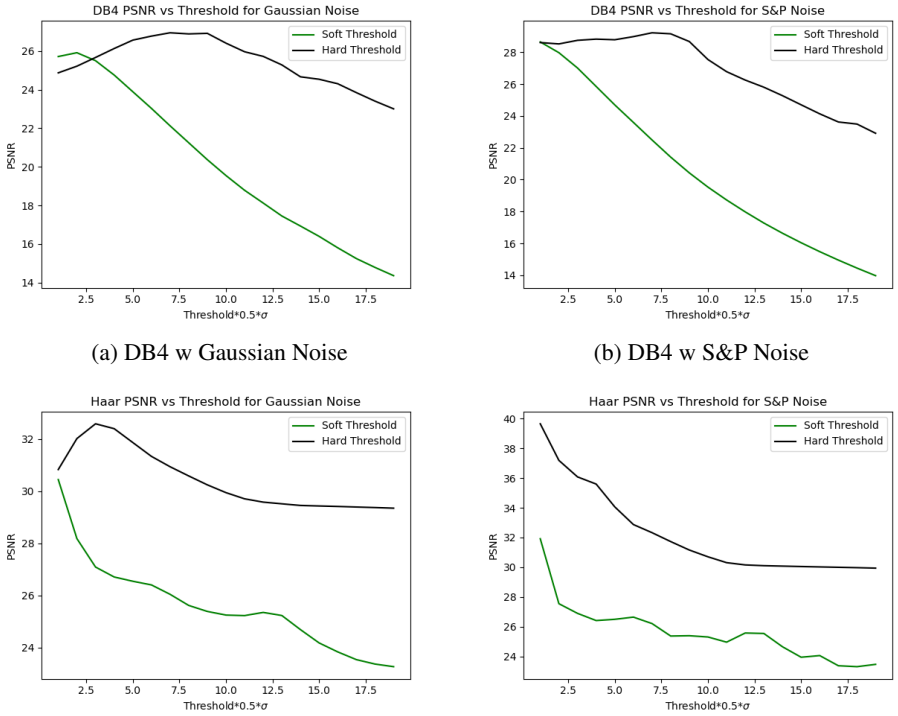
Figure 3: Thresholding Methods

The following Figure depicts the PSNR of the filtered wavelet transforms vs. a given threshold value, and the threshold value was chosen that resulted in the greatest PSNR value.



(a) Hard Threshold. (b) Soft Threshold.

Figure 4: PSNR vs. Filter Lengths



(a) DB4 w Gaussian Noise (b) DB4 w S&P Noise

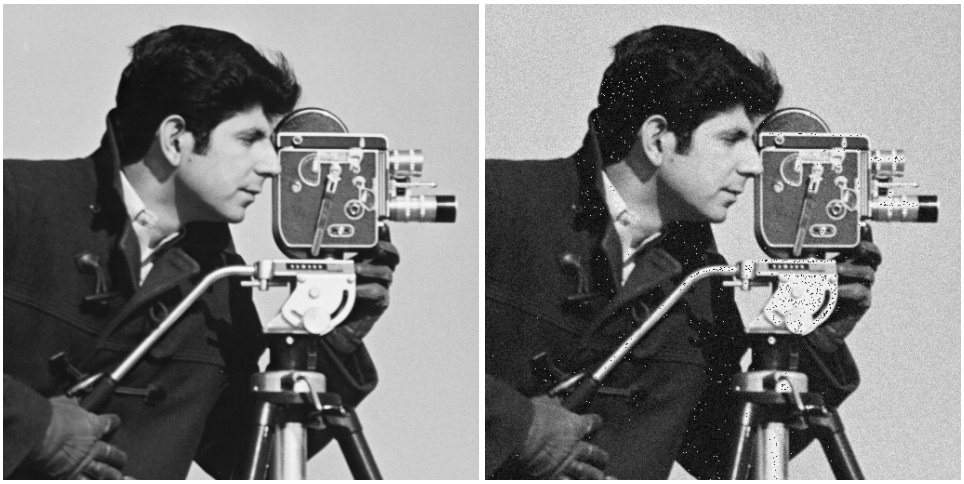
(c) HWT w Gaussian Noise (d) HWT w S&P Noise.

Figure 5: PSNR vs. Thresholding Value

3 Results

Filter	PSNR w/ Gaussian Noise	PSNR w/ SP Noise
Blur	33.652	51.418
Gaussian Blur	33.901	51.418
Median	33.991	51.418
Bilateral	33.02	33.297
HWT Soft Threshold	30.433	31.915
HWT Hard Threshold	32.633	39.664
DB4 Soft Threshold	26.979	29.358
DB4 Hard Threshold	26.005	28.338

4 Conclusion



(a) cman (b) cman w Gaussian Noise

(c) cman w S&P Noise

Figure 6: PSNR vs. Thresholding Value

References

References follow the acknowledgments. Use unnumbered first-level heading for the references. Any choice of citation style is acceptable as long as you are consistent. It is permissible to reduce the font size to `small` (9 point) when listing the references. **Remember that you can use a ninth page as long as it contains *only* cited references.**

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

5 Software Lisiting

5.1 dataSetup

5.2 DWT