Solution 1

(a) The observed noisy image described in lecture 2 is,

$$I = gI^0 + \epsilon \tag{1}$$
$$I = gI^0 + \left(g\sqrt(I^0)\epsilon_1 + g\sigma_{2a}\epsilon_{2a} + \sigma_{2b}\epsilon_{2b}\right) \tag{2}$$

where $\epsilon_{1,2a,2b}$ are normalized Gaussian Distributions,

$$\epsilon_{1,2a,2b} \sim \mathcal{N}(0,1) \tag{3}$$

Because gI^0 is a constant, and the mean of the noise, $\mu_\epsilon = 0$, we can describe the variance of the observed image,

$$\text{Var}(I) = \text{Var}(\epsilon) = \text{Var}\left(g\sqrt(I^0)\epsilon_1 + g\sigma_{2a}\epsilon_{2a} + \sigma_{2b}\epsilon_{2b}\right) \tag{4}$$

Additionally, since each distribution, $\epsilon_{1,2a,2b}$, is uncorrelated with each other, we can describe the variance using the following formula

$$\text{Var}\left(\sum_{i=1}^{N} \sigma_i\epsilon_i\right) = \sum_{i=1}^{N} \sigma_i^2\left(\text{Var}(\epsilon_i)\right) \tag{5}$$

giving us,

$$\text{Var}(I) = g^2I^0\text{Var}(\epsilon_1) + g^2\sigma_{2a}^2\text{Var}(\epsilon_{2a}) + \sigma_{2b}^2\text{Var}(\epsilon_{2b}) \tag{6}$$

and we have already described $\epsilon_{1,2a,2b}$ as a normalized Gaussian Distribution, therefore $\text{Var}(\epsilon_{1,2a,2b}) = 1$, and we get

$$\boxed{\text{Var}(I) = g^2I^0 + g^2\sigma_{2a}^2 + \sigma_{2b}^2} \tag{7}$$

(b) If we reduce the exposure time, T to $T_k = T/k$, the ideal image is also reduced by the same factor, and we scale g by that factor as well:

$$I_k^0 = I^0/k \tag{8}$$
$$g_k = gk \tag{9}$$

Nothing else has changed in our image, so we can just substitute $I_k^0,\ g_k$ into equation (7) for $I^0,\ g$ to get

$$\boxed{\text{Var}(I_k) = g^2kI^0 + g^2k^2\sigma_{2a}^2 + \sigma_{2b}^2} \tag{10}$$

(c) If we take k images $I_1, \ldots, I_k$, and average them to get I, we would get a variance of

$$\text{Var}(I) = \text{Var}\left(\frac{1}{k}\sum_{n=1}^{k} I_n\right) \tag{11}$$

where each image I_n has the variance described in equation (10). We can then use equation (5) again, since the noise of the individual images are not correlated to get,

$$\text{Var}(I) = \frac{1}{k^2}\sum_{n=1}^{k}\text{Var}(I_n) = \frac{1}{k^2}\left(g^2kI^0 + g^2k^2\sigma_{2a}^2 + \sigma_{2b}^2\right) \tag{12}$$

and we simplify to get

$$\boxed{\text{Var}(I) = \frac{g^2I^0}{k} + g^2\sigma_{2a}^2 + \frac{\sigma_{2b}^2}{k^2}} \tag{13}$$

(d) If we average a set of shorter exposure images over the same time frame, T, we can get a smaller variance in the observed image, I, which will result in a higher signal to noise ratio, up to a minimum noise of $g^2\sigma_{2a}^2$ if we let $k \to \infty$, where g is the optimal amplification factor for a single exposure of length T. **This means that taking a set of short exposure images is preferable to one long exposure image in the case of a static scene**. However, there may be hardware difficulties in taking a large number of short images, and software difficulties with processing them for large values of k.

Solution 2

The output of the histogram equalization algorithm I wrote from this problem is shown in Figure 1. I have also included the before and after histograms shown in Figure 2. The distribution is not perfect, but it approximately follows a discretized version of a linear function.

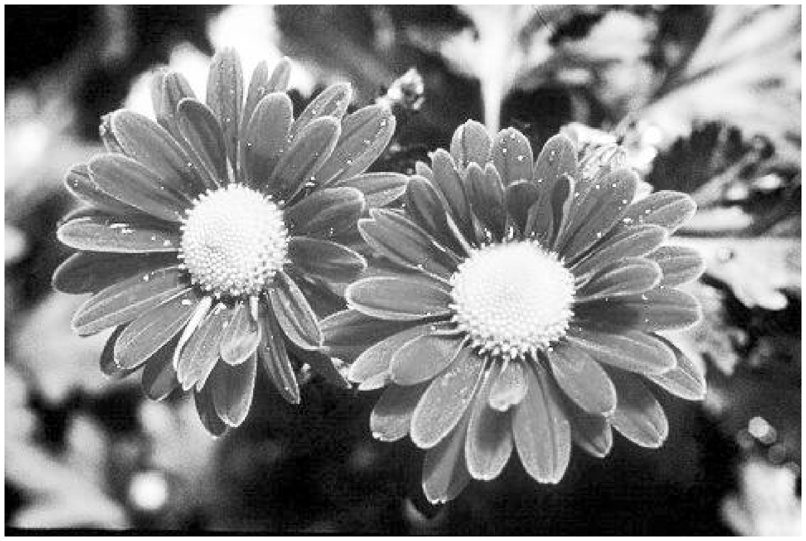

Figure 1: Histogram Equalized Image from Problem 2

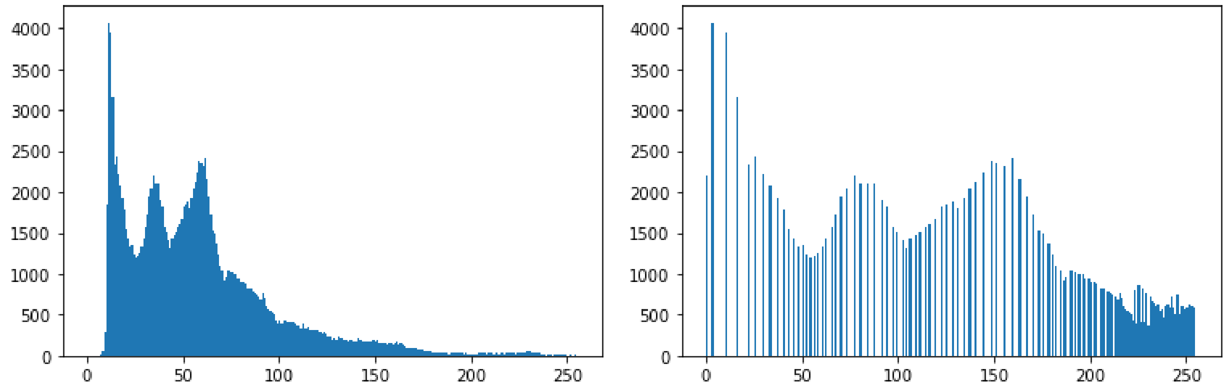

Figure 2: Original Image Histogram (left) and Post Equalization Histogram (right)

Solution 3

(a) I calculated the smoothed gradient of the image in each direction, I_x, I_y using the sobel kernel described in the problem. I used these to calculate the maximum magnitude and angle of the derivative, H, Θ using equations from the lecture,

$$H = \sqrt{I_x^2 + I_y^2} \tag{14}$$

$$\Theta = \arctan_2(I_y, I_x) \tag{15}$$

The resulting image is shown in Figure 3:

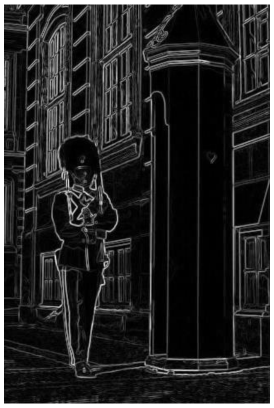

Figure 3: Magnitude H of the Sobel Derivative of the Image for Problem 3

(b) Next, the provided code processed the image shown in Figure 3 to be boolean, only containing pixels that have a magnitude above a threshold ϵ at values of $0.5, 1.0, 1.5$. The higher the threshold, the fewer pixels get marked as having edges. The result is shown in Figure 4:

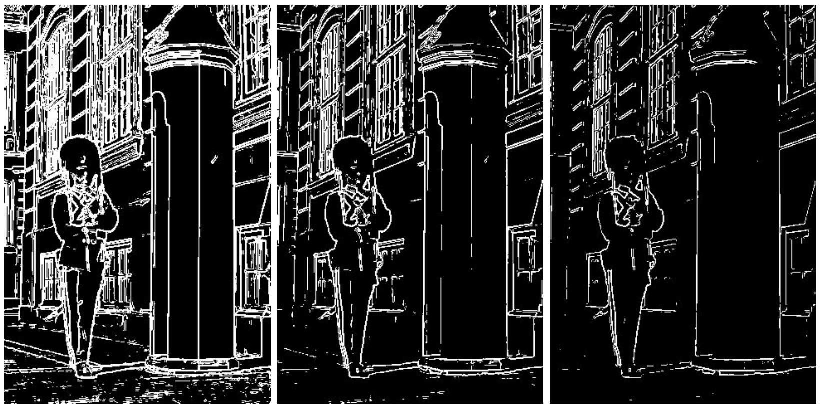

Figure 4: Different ϵ Threshold Values for Edge Detection. From left to right: $0.5, 1.0, 1.5$

I then implemented non-maxima suppression on these three images, to reduce the thickness of edges. Since the edge pixels don't have pixels on both sides, I assumed that they could not be edges and set their values to 0. Next, I rounded the angles for each pixel to the closest of four categories: vertical, $45\deg$ clockwise, horizontal, or $45\deg$ counter-clockwise. Since, a rotation by

$180\,\mathrm{deg}$ does not change these, I can assume that they are equivalent. I implemented this by multiplying by $4/\pi$ to make the desired angles integers, then rounding the result modulo 4,

$$\text{direction}[n] = \text{round}(4/\pi\ \Theta[n])\%4 \tag{16}$$

Then, for every pixel that was determined to be an edge by the Threshold was tested to see if its magnitude was greater than its neighbors along the direction of its derivative. If it was, it was kept, otherwise it was set to 0. The result for each Threshold value from 4 is shown in Figure 5

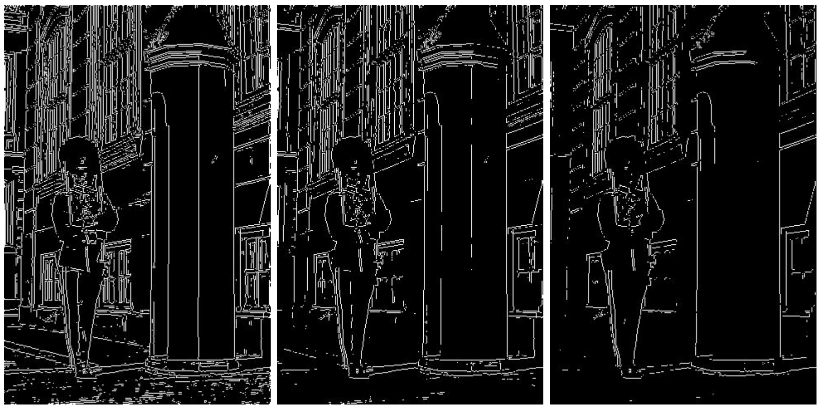

Figure 5: Result of NMS After Thresholding for $\epsilon = 0.5, 1.0, 1.5$ from left to right

Solution 4

In this problem I implemented a bilateral filter. I did this by first defining the Gaussian Kernel, since that is constant over the image, and can be easily multiplied by the pixel-dependent part of the filter. Next, I padded the image on each side with K zeros, where the size of the kernel is $2K + 1$, so I would never index outside the padded image when applying the filter. After this, I calculated the kernel B for every pixel from the padded image, and element-wise multiplied the kernel with the appropriate pixels in the padded image to get the output. The outputs for this on an image are shown in Figure 6, with varying values of σ_s and σ_i, the strength of the Gaussian portions of the filter and the pixel-dependent portion. The image on the right with the largest intensity-dependent portion retained most of its hard-edges, while the smooth parts of the image became more smoothed.

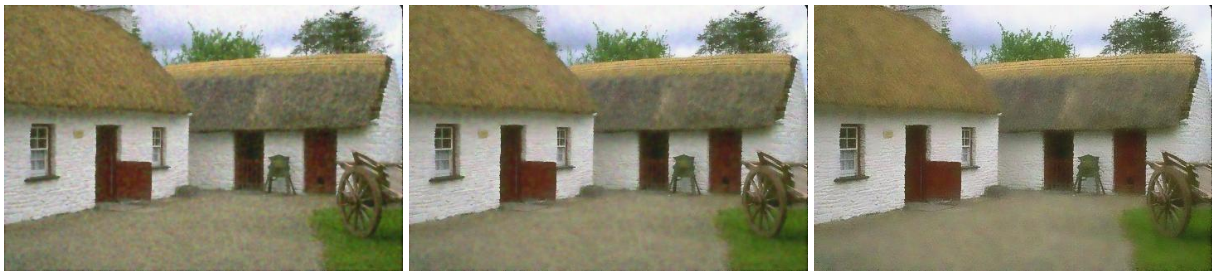

Figure 6: Bilateral Filter, from left to right, increasing strength of pixel-dependent portion and decreasing strength of Guassian

The images in the next Figure 7 were run through the bilateral filter multiple times, enhancing the effect.

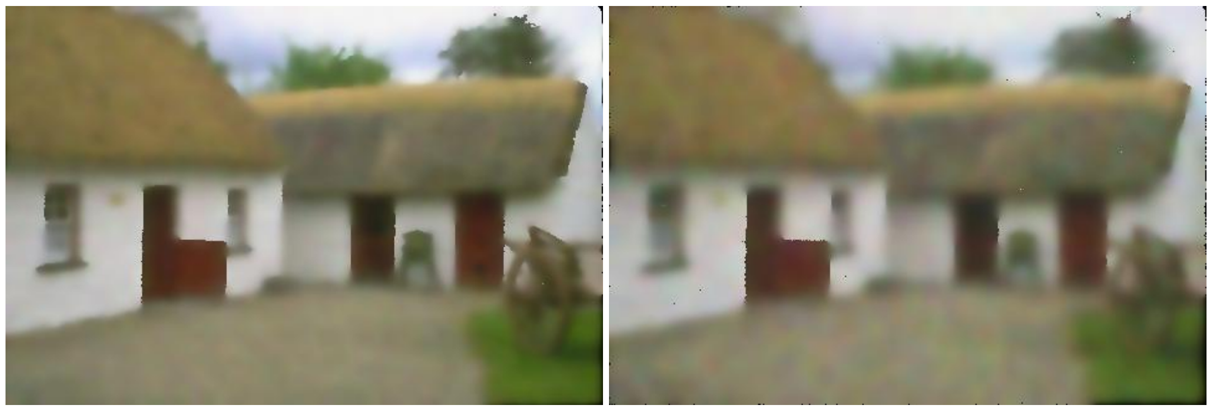

Figure 7: Bilateral Filter, Run Multiple Times

The algorithm I wrote is not computationally efficient, it took more than a minute to run a single implementation of the filter for $K = 9$.

Solution 5

(a) To prove that we only require $W_X H_X$ unique real coefficients to store a full 2D Fourier transform, I will start with property that

$$F[-u,-v] = F^*[u,v] \tag{17}$$

This property can be proven quickly using the definition of the Fourier transform

$$F[u,v] = \frac{1}{WH}\sum_{n_x=0}^{W-1}\sum_{n_y=0}^{H-1} X[n_x,n_y]\text{exp}\left(-j2\pi\left(\frac{un_x}{W}+\frac{un_y}{H}\right)\right) \tag{18}$$

replacing u, v with $-u, -v$ and factoring out the negative to the front of the complex exponential

$$F[-u,-v] = \frac{1}{WH}\sum_{n_x=0}^{W-1}\sum_{n_y=0}^{H-1} X[n_x,n_y]\text{exp}\left(-j2\pi\left(\frac{-un_x}{W}+\frac{-un_y}{H}\right)\right) \tag{19}$$

$$= \frac{1}{WH}\sum_{n_x=0}^{W-1}\sum_{n_y=0}^{H-1} X[n_x,n_y]\text{exp}\left(+j2\pi\left(\frac{un_x}{W}+\frac{un_y}{H}\right)\right) \tag{20}$$

$$= F^*[u,v] \tag{21}$$

Next, I will show that coefficients where $u = 0,\ W/2$ and $v = 0,\ H/2$, the imaginary component of $F[u,v]$ is zero, and therefore does not need to be stored. Again, using the definition of the transform, we let $u = kW/2$ and $v = lH/2$, where $k, l \in \mathcal{Z}$

$$F[kW/2, lH/2] = \frac{1}{WH}\sum_{n_x=0}^{W-1}\sum_{n_y=0}^{H-1} X[n_x,n_y]\text{exp}\left(-j2\pi\left(n_xk/2+n_yl/2\right)\right) \tag{22}$$

we can factor out a 2, and note that $\exp(jq) = \{-1, 1\}$ where q is an integer. Observe that our complex exponential now is only in terms of added and multiplied integers, and so is also an integer, and therefore the Fourier transform for $u = 0,\ W/2$ and $v = 0,\ H/2$ is completely real.
The last property I will use is

$$F[W/2,v] = F[-W/2,v] \tag{23}$$

$$F[u,H/2] = F[u,-H/2] \tag{24}$$

This is trivial using the circular property of the discrete Fourier transform,

$$F[u,v] = F[u+W,v] \tag{25}$$

$$F[u,v] = F[u,v+H] \tag{26}$$

and we let $u = -W/2$ in eq 23 and $v = -H/2$ in eq 24. This property is important because it allows us to find complex conjugate pairs along in the even cases along $u = -W/2$ and $v = -H/2$. Lastly, I will construct three Fourier transform images, one with W, H both even, one with W, H both odd, and one with W even and H odd (which is equivalent to H even and W odd), and count the unique terms that must be stored.

1. Case 1: We will start with H, W both even

$$F[u,v] = \left(\begin{array}{c|c|c|c} \overbrace{F[-W/2,-H/2]}^{(1\times 1)} & \overbrace{F[-u,-H/2]}^{(1\times W/2-1)} & \overbrace{F[0,-H/2]}^{(1\times 1)} & \overbrace{F[u,-H/2]}^{(1\times W/2-1)} \\ \hline \overbrace{F[-W/2,-v]}^{(H/2-1\times 1)} & \overbrace{F[-u,-v]}^{(H/2-1\times W/2-1)} & \overbrace{F[0,-v]}^{(H/2-1\times 1)} & \overbrace{F[u,-v]}^{(H/2-1\times W/2-1)} \\ \hline \overbrace{F[-W/2,0]}^{(1\times 1)} & \overbrace{F[-u,0]}^{(1\times W/2-1)} & \overbrace{F[0,0]}^{(1\times 1)} & \overbrace{F[u,0]}^{(1\times W/2-1)} \\ \hline \overbrace{F[-W/2,v]}^{(H/2-1\times 1)} & \overbrace{F[-u,v]}^{(H/2-1\times W/2-1)} & \overbrace{F[0,v]}^{(H/2-1\times 1)} & \overbrace{F[u,v]}^{(H/2-1\times W/2-1)} \end{array}\right) \tag{27}$$

We then apply the properties that we proved above in equations 19/21, 22 and 25/26. We start with the real parts, using $F[u, v] = F^*[-u, -v]$ and, $F[u, -v] = F[-u, v]$. $F[0, 0], F[-W/2, 0], F[0, -H/2]$ and $F[-W/2, -H/2]$ are all unique and real. Therefore every element either has a complex conjugate in the Fourier image or is real. Therefore the number of scalars required to store the Fourier transform is equal to the size of the image, WH.

2. Case 2: Both W, H are odd.

$$F[u,v] = \left(\begin{array}{c|c|c} \overbrace{F[-u,-v]}^{((H-1)/2\times(W-1)/2)} & \overbrace{F[0,-v]}^{((H-1)/2\times 1)} & \overbrace{F[u,-v]}^{((H-1)/2\times(W-1)/2)} \\ \hline \overbrace{F[-u,0]}^{(1\times(W-1)/2)} & \overbrace{F[0,0]}^{(1\times 1)} & \overbrace{F[u,0]}^{(1\times(W-1)/2)} \\ \hline \overbrace{F[-u,v]}^{((H-1)/2\times(W-1)/2)} & \overbrace{F[0,v]}^{((H-1)/2\times 1)} & \overbrace{F[u,v]}^{((H-1)/2\times(W-1)/2)} \end{array}\right) \tag{28}$$

We use the same properties as above, but this time there is only one unique element, $F[0, 0]$. Every other element has a complex conjugate in the image, and therefore require only one scalar each. Since $F[0, 0]$ is real, it only requires one scalar to store as well, therefore the total scalars needed is WH.

3. Case 3: H is odd, W is even (or vice versa)

$$F[u,v] = \left(\begin{array}{c|c|c|c} \overbrace{F[-W/2,-v]}^{((H-1)/2\times 1)} & \overbrace{F[-u,-v]}^{((H-1)/2\times W/2-1)} & \overbrace{F[0,-v]}^{((H-1)/2\times 1)} & \overbrace{F[u,-v]}^{((H-1)/2\times W/2-1)} \\ \hline \overbrace{F[-W/2,0]}^{(1\times 1)} & \overbrace{F[-u,0]}^{(1\times W/2-1)} & \overbrace{F[0,0]}^{(1\times 1)} & \overbrace{F[u,0]}^{(1\times W/2-1)} \\ \hline \overbrace{F[-W/2,v]}^{((H-1)/2\times 1)} & \overbrace{F[-u,v]}^{((H-1)/2\times W/2-1)} & \overbrace{F[0,v]}^{((H-1)/2\times 1)} & \overbrace{F[u,v]}^{((H-1)/2\times W/2-1)} \end{array}\right) \tag{29}$$

In this case, we also have every element with a unique complex conjugate, or is unique and real, making the total required scalars WH.

(b) Proof that circular rotation of a padded kernel multiplied by the FFT is equivalent to a same, wrap convolution is shown in Figure 8. I compared the two methods by subtracting their pixels values from each other. The biggest difference between two pixels was on the order of 10^{-6}, confirming that the methods are in fact equivalent.

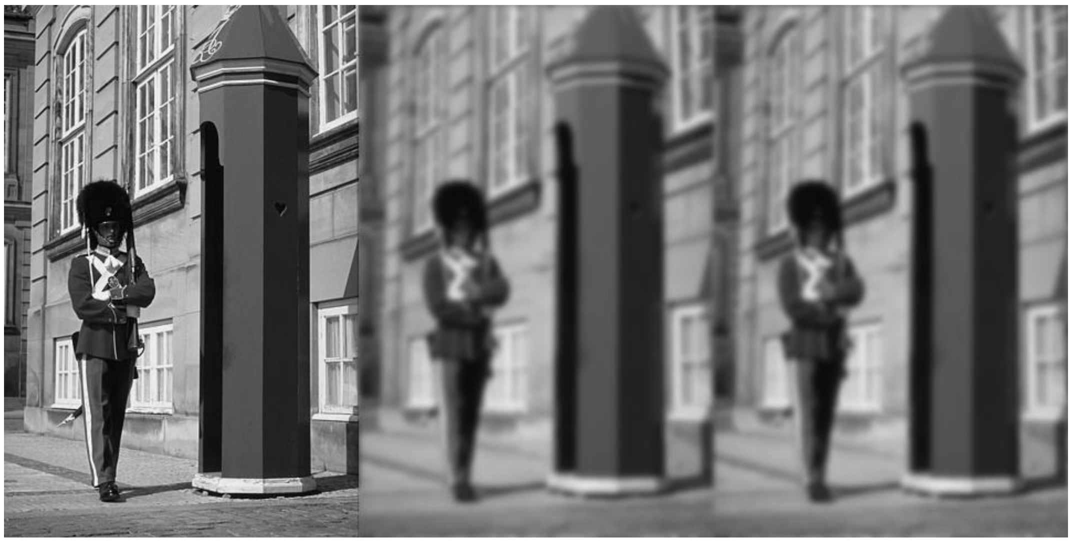

Figure 8: Original Image (left) and Equivalent Convolution and FFT Multiplication Images (center and right)

Solution 6

(a) In this problem I implemented the Harr wavelet decomposition using matrix multiplication and the transform matrix

$$W = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \tag{30}$$

For each iteration of the loop, I construct a, b, c, d from the previous iteration's L (or the original image in the case of the first iteration), by taking every other pixel from the image with different phases. I then reshape them into row vectors to multiply by W:

$$\begin{bmatrix} L \\ H_1 \\ H_2 \\ H_3 \end{bmatrix} = W \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \tag{31}$$

These new images can then be reshaped back into four images with half the dimensions, H_1, H_2, H_3 are appended as a list to a list tracking the decomposition, and L is reused for the next iteration. After the last iteration is complete, L itself is then appended. The result of the decomposition for 1, 2 and 3 levels is shown below in Figure 9.

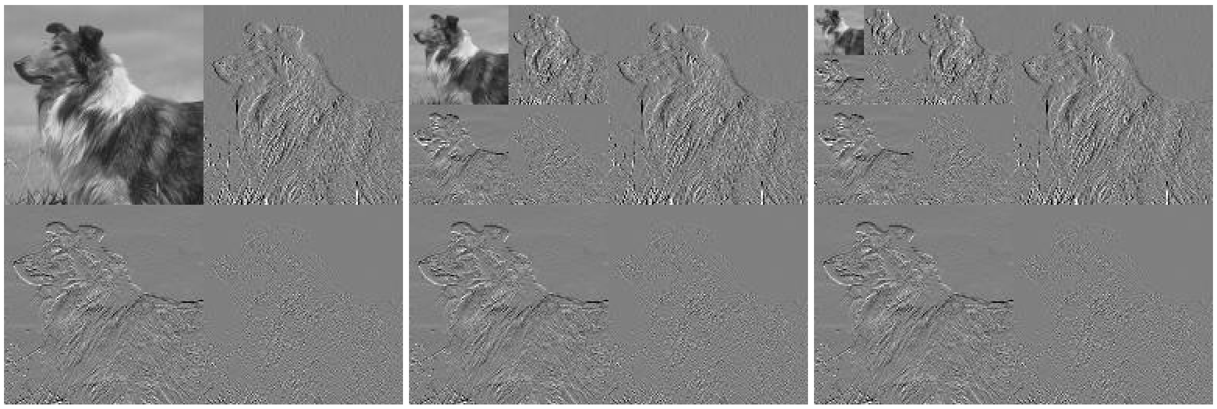

Figure 9: Harr Wavelet Decomposition at 1, 2 and 3 Levels (from left to right)

(b) Next, I implemented the inverse, the reconstruction of the image from its Wavelets. Because the decomposition is a linear transformation, we can take multiply the result by W^{-1} the same number of times to get the original. Since W is a unitary matrix, $W^{-1} = W^T$ so we can just use the transpose instead. We do the same steps as creating the decomposition in reverse. We start by popping off the final image of the decomposition from the list, L, and then for each iteration pop off H_1, H_2, H_3 from that iteration, and reshape them all into row vectors. To inverse transform, we apply the linear transform

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = W^T \begin{bmatrix} L \\ H_1 \\ H_2 \\ H_3 \end{bmatrix} \tag{32}$$

The next iteration uses L reconstructed from a, b, c, d using every second element of each, the same way it was deconstructed. The reconstruction doesn't require all the information to reconstruct an approximation of the original image. If you zero out the first few iterations of the wavelet, you lose the high frequency information of the image. In a similar manner, if you add zero padding to your image and apply an inverse wavelet, that will naively upscale the image. The results of a full reconstruction, and reconstruction removing the first 1, 2 and 3 iterations H lists is shown below in Figure 10

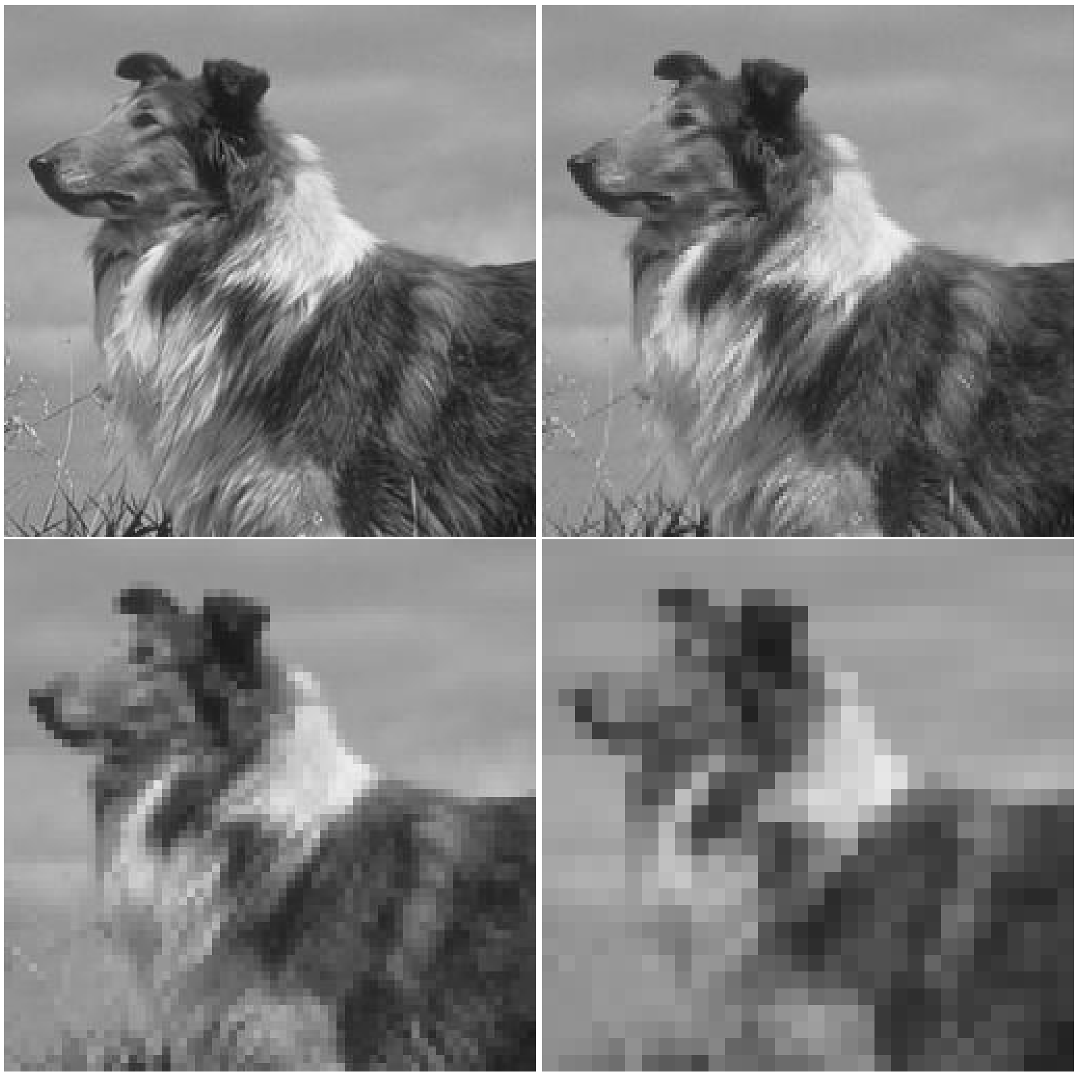

Figure 10: Harr Wavelet Reconstruction with full information, and missing 1, 2 and 3 iterations of H data (reading across)

Information

This problem set took approximately 20 hours of effort.

I discussed this problem set with no one.

I used from the following sources:

- Wikipedia article on Variance: `https://en.wikipedia.org/wiki/Variance`
- Wikipedia article on Bilateral Filter: `https://en.wikipedia.org/wiki/Bilateral_filter`
- Numpy shift arrays (5b): `https://numpy.org/doc/stable/reference/generated/numpy.roll.html`
- Recitation on convolutions