Compressed Sensing Image Recovery

Elliot Ha
03.06.23

A Brief Overview

An Introduction to Image Recovery

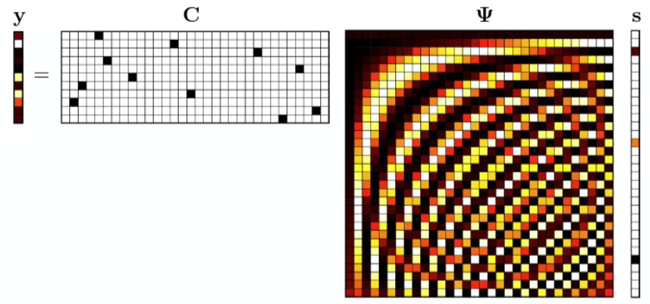
What is this **project**? The **objectives**? How are we going to do it?

Suppose we are given an image.

However, due to file corruption, or data loss, or mercury being in retrograde, our image is missing pixels. A **lot** of pixels. The way that we intend to “recover” these pixels is through a process known as **compressed sensing image recovery**. This is a signal processing technique which exploits the sparse representation of coefficients of the image vector in a generic transform basis in order to reconstruct the original uncorrupted image by solving an underdetermined system of equations for the sparsest solution via L1 regression[2].

A Brief Overview

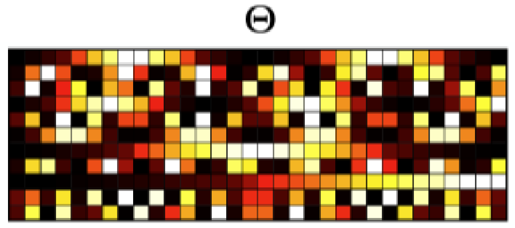
An Introduction to Image Recovery (Continued)



Figures (1) & (2):

Representation of the randomly sampled measurements, **y**, and its sparse representation in the transform basis, $\boldsymbol{\psi}$, with pixels selected by **C**.

$\boldsymbol{\Theta}$ is the matrix given by $\boldsymbol{C\psi}$, the rows of the identity matrix corresponding to the randomly measured pixels given by **C**.



That was a lot of big words, but what does it all mean? To understand the concepts behind **compressed sensing image recovery**, we must first understand the need for it. Suppose that we start with a complete image, where each pixel is represented by its intensity on a particular scale. For our purposes, we will be using a grayscale colormap with values linearly mapped from 0 to 255. Let's call our complete image the vector, **x**, where the 2D matrix of pixel intensities is rasterized into a single column vector.

Our image has a sparse representation, **s**, in **some transform basis different from our image's regression**[3], $\boldsymbol{\psi}$, where **sparsity means that many of the transform coefficients are close to or equal to zero in this basis**[4]. Mathematically, this relationship is represented by $\boldsymbol{x} = \boldsymbol{\psi s}$.

But what if we don't have a complete image? From a few randomly taken measurements, can we infer what the nonzero coefficients of **s** are such that we get the sparse representation in the transform basis, and from that get the original image? **The answer is yes**, and this is the whole idea behind compressed sensing.

Suppose we measure the column vector **y**, which is some measurement of our original image, **Cx**, where **C** is our choice of measurement matrix selecting random pixels from **x**. This relationship can be stated by $\boldsymbol{y} = \boldsymbol{Cx}$, and thus $\boldsymbol{y} = \boldsymbol{C\psi s}$. **These measurements are in our image's original basis**, which is different from the basis in which our signal is sparsely represented by **s**. Our goal, then, is to solve for **s** given that we have **y**, **C**, and $\boldsymbol{\psi}$. More specifically, we are **looking to solve this underdetermined system of equations by solving for the *sparsest* solution** of **s** in the transform basis, which allows us to arrive at a particular solution. The sparsest solution, **s**, **given by the L1 norm**[5], satisfies the optimization problem $\min_{s} \|\boldsymbol{C\psi s} - \boldsymbol{y}\|_2 + \lambda \|\boldsymbol{s}\|_1$

Which Optimization Problem are we Solving?

As stated previously, the L1 optimization problem we are trying to solve is given by $\min_{s} ||\boldsymbol{C\psi s} - \boldsymbol{y}||_2 + \lambda ||\boldsymbol{s}||_1$

Why L1 though? Why not L2?

L2 regularization pushes all parameters towards small values, but not necessarily zero[6], or the sparse solution we are looking for, whereas **the minimum L1 norm provides the sparsest solution**[5] to the underdetermined system that we need.

However, the problem that we are actually looking to solve is given by the L0 norm, the function that quite literally counts the number of non-zero components of a vector[7] .

Due to recent advancements, it has been proven that **the L1 norm actually converges to the sparse solution that is equivalent to the L0 norm**[5] , and the reason we choose the L1 solution is that it is computationally easier to solve by many orders of magnitude compared to the L0 solution[8].

An Introduction to Image Recovery (Continued)

It was mentioned previously that our image has a sparse representation, **s**, in some transform basis different from our image's basis. But what exactly is this transform basis?

“If a basis, ψ, is generic, such as the Fourier or wavelet basis, then only a few active (non-zero) terms in s are required to reconstruct the original signal, x. "[3]

This, again, is the idea behind compressed sensing image recovery, and the particular basis that we will be using for this project is the one given by the Discrete Cosine Transform (DCT). This transform is commonly used in signal processing due to needing fewer cosine functions than sine functions to approximate a typical signal[9].

As for the methodology, the DCT coefficients of a large image are often not sparse. Which prompts our approach of approach of breaking the overall image into smaller image blocks, each of which will have a few active terms/weights.

From here, we can reconstruct our original image by concatenating all the blocks together.

The two bases that we will be dealing with include the original image's coordinate location given by the horizontal and vertical values x and y, as well as the DCT basis given by the spatial frequency values u and v in both the horizontal and vertical directions again.

Compressed Sensing Image Recovery

Table of Contents

1

Sampling & Image Corruption

Chapter 1

Original Images & Methodology

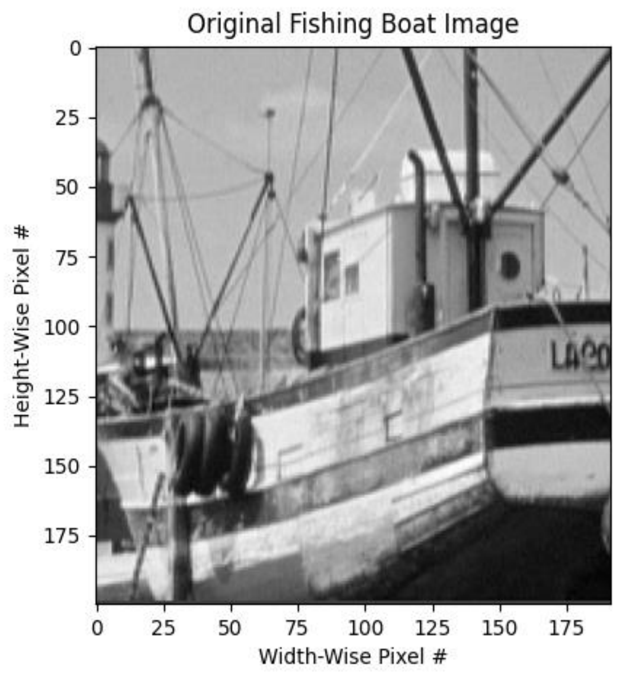


Figure (3):
Original Fishing Boat Image (K=8)

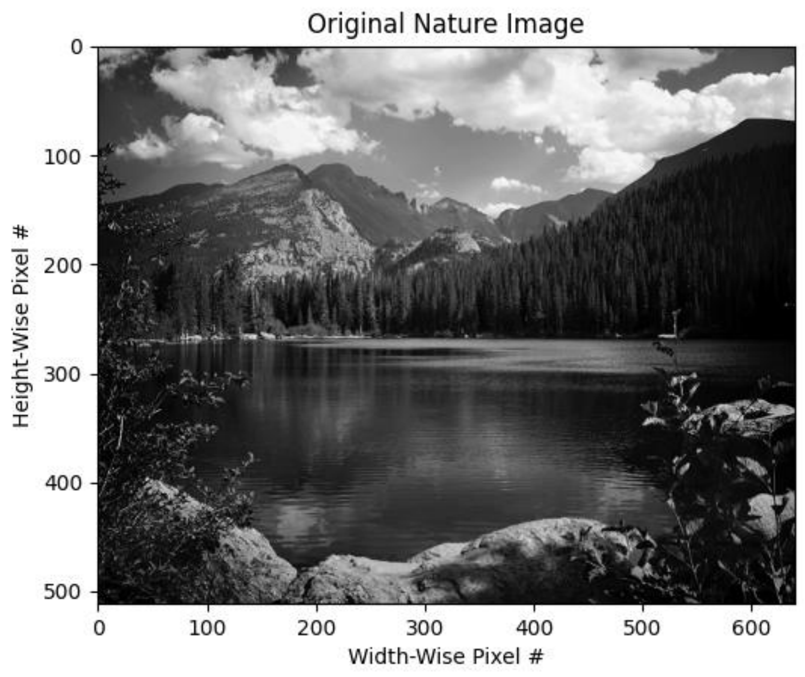


Figure (4):
Original Nature Image (K=16)

Our intended goal for this project is to be able to reconstruct, to the best of our ability, two images. The images in their unadulterated, original forms that we will be working with are shown in Figures (3) and (4).

We have chosen these two images in order to demonstrate our project's capabilities to different magnitudes of size.

For the Fishing Boat image, when we break the image into blocks of KxK pixels in order to identify the DCT coefficients of each block to concatenate later, we will be choosing to break the blocks into sizes of 8x8 pixels, or K = 8.

For the Nature image, to again demonstrate capability in higher magnitudes, we will be choosing to break the blocks into sizes of 16x16 pixels due to the larger size. K = 16.

Chapter 1

Image Corruption Process

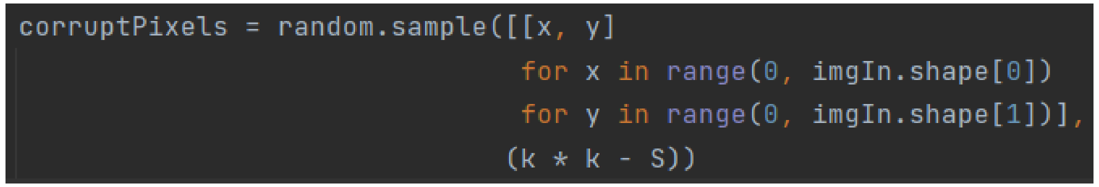
Our first task towards developing a working model is to be able to synthetically imitate a “corrupted image”. The way that we will do this is by taking one of the KxK blocks, and retaining a set number of pixels that will be our “sampled pixels”. The rest of the pixels in that KxK block will be set to NaN values, imitating being corrupted.

In order to corrupt the correct number of pixels, it is of paramount importance that we select our “sampled pixels” without replacement. If we were to replace pixels, we would run the probability of sampling one of the pixels multiple times, however unlikely.

We will repeat the corruption process of each image 5 times for different values of S, the number of sampled pixels.

I utilized Python’s inbuilt random library[10], of which includes a sampling function without replacement. The parameters that this function takes as arguments include a population sequence of which to sample, and an integer value of how many unique elements to sample.

It is also important to note that for reproducibility’s sake, the random number generator’s seed was set in my code to be explicitly the default parameters.



```
corruptPixels = random.sample([[x, y]
                                for x in range(0, imgIn.shape[0])
                                for y in range(0, imgIn.shape[1])],
                               (k * k - S))
```

Figure (5):
Code block of sampling method parameters without replacement. The number of corrupt pixels in each block is equal to the dimensions of each block squared minus the number of sampled pixels.

Chapter 1

Fishing Boat Image Corruption Results (K = 8)

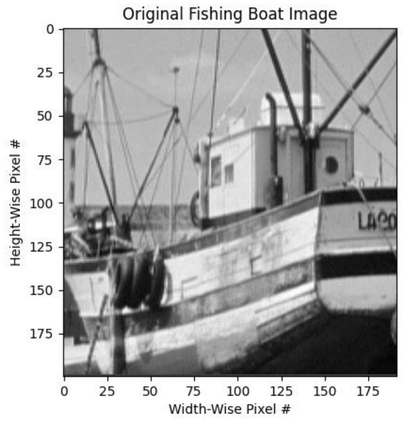


Figure (6):

Original Fishing Boat Image

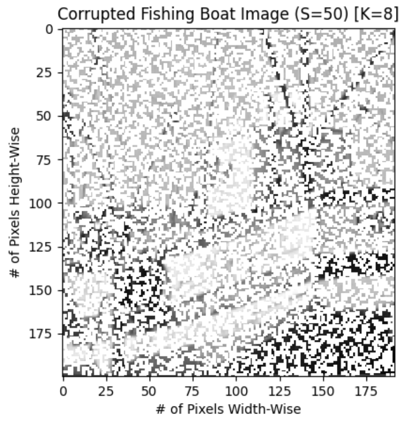


Figure (7):

Sample Corrupted Image for S = 50

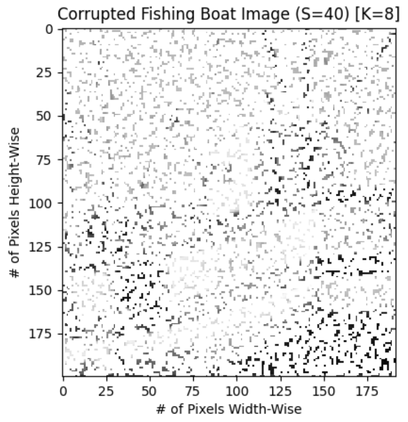


Figure (8):

Sample Corrupted Image for S = 40

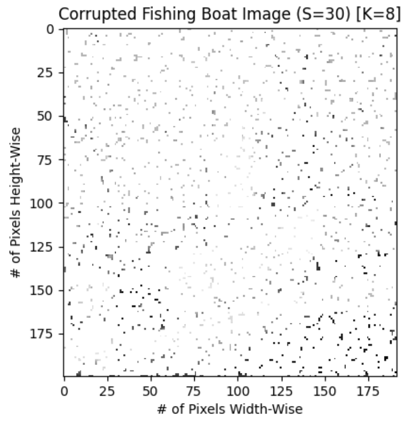


Figure (9):

Sample Corrupted Image for S = 30

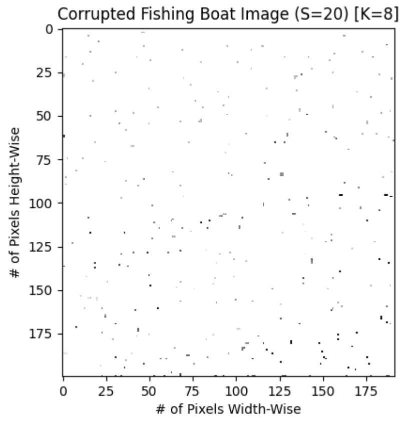


Figure (10):

Sample Corrupted Image for S = 20

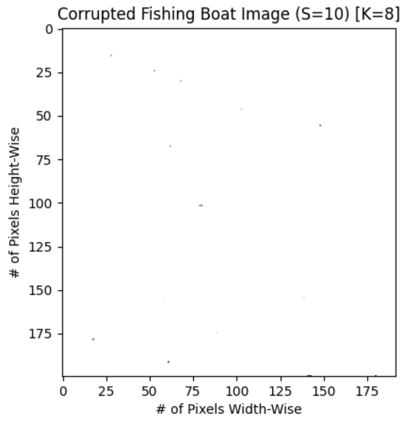


Figure (11):

Sample Corrupted Image for S = 10

Chapter 1

Nature Image Corruption Results (K = 16)

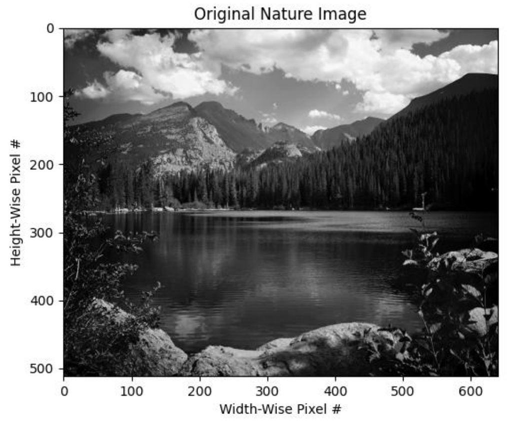


Figure (12):

Original Nature Image

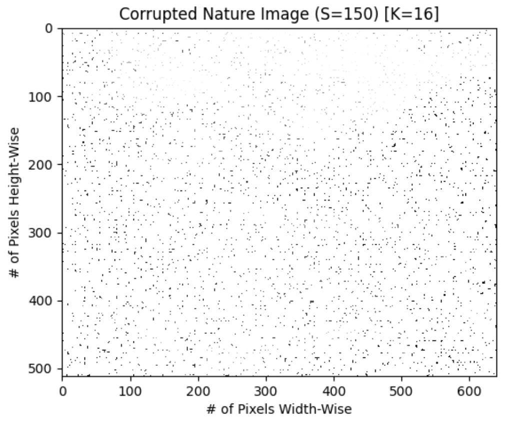


Figure (13):

Sample Corrupted Image for S = 150

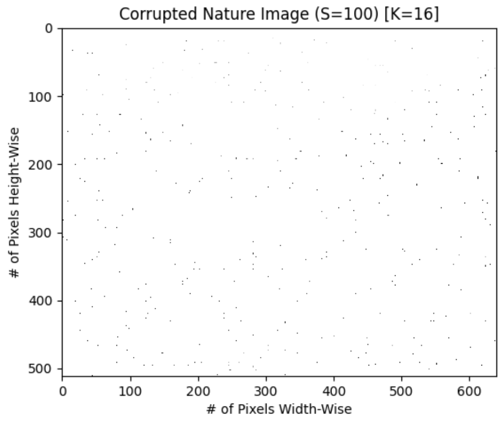


Figure (14):

Sample Corrupted Image for S = 100

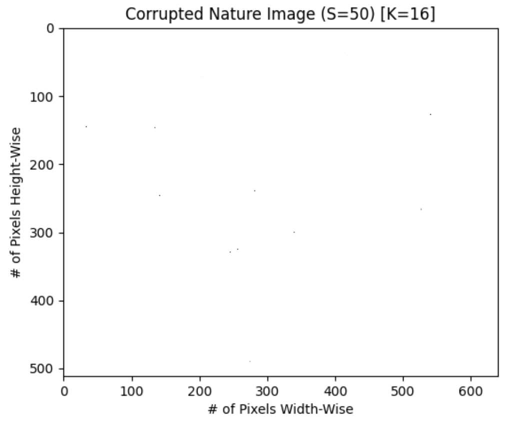


Figure (15):

Sample Corrupted Image for S = 50

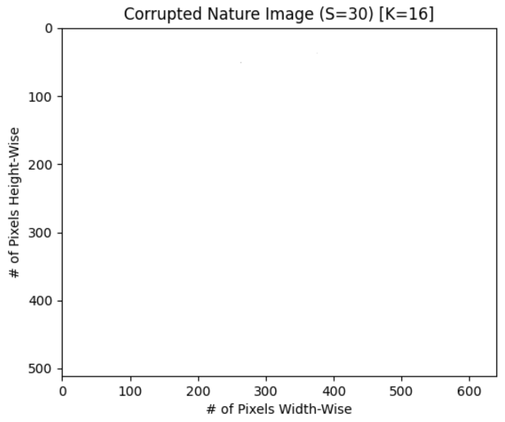


Figure (16):

Sample Corrupted Image for S = 30

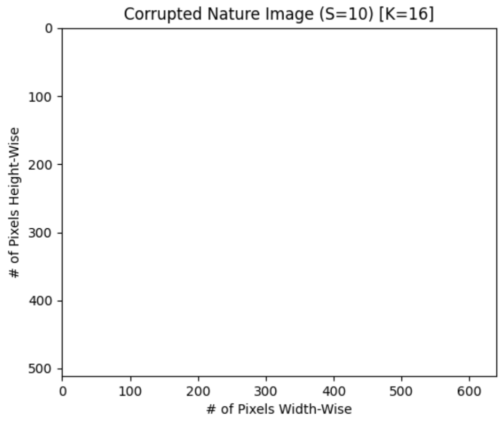


Figure (17):

Sample Corrupted Image for S = 10

2

Image Reconstruction (Block Level)

Chapter 2

Basis Vector Matrix Construction (K = 8)

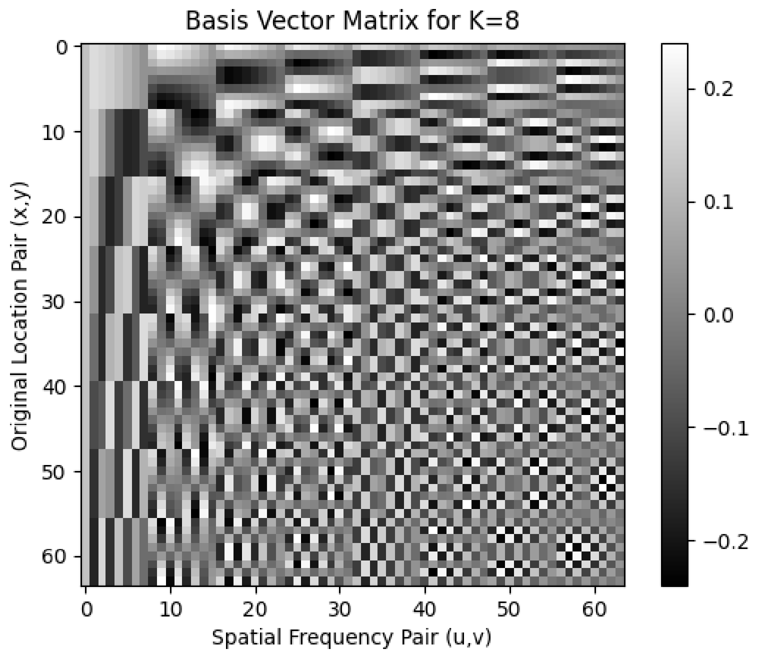


Figure (18):
Basis Vector Matrix Visualization for K=8

As stated previously, the basis chosen in which our image will have a sparse representation is given by the Discrete Cosine Transform (DCT). We can convert a pixel's spatial location given by its (x, y) coordinates into the basis given by the spatial frequency pair (u, v).

The methodology by which we generate these basis vector matrices is by calculating the basis vector via the parametric equation:

$$\alpha_u \cdot \beta_v \cdot \cos\left[\frac{\pi\cdot(2x-1)(u-1)}{2\cdot P}\right] \cdot \cos\left[\frac{\pi\cdot(2y-1)(v-1)}{2\cdot Q}\right]$$

Where (x, y) is the spatial location pair, (u, v) is the spatial frequency pair, with

$(x, u) \in \{1,2,\dots,P\}$ and $(y, v) \in \{1,2,\dots,Q\}$

Where P is the number of pixels in the horizontal direction and Q the number of pixels in the vertical direction.

$$\alpha_u = \begin{cases} \sqrt{\frac{1}{P}}, u = 1 \\ \sqrt{\frac{2}{P}}, 2 \leq u \leq P \end{cases} \text{ and } \beta_v = \begin{cases} \sqrt{\frac{1}{Q}}, v = 1 \\ \sqrt{\frac{2}{Q}}, 2 \leq v \leq Q \end{cases}$$

For the KxK block, each basis vector has K^2 elements and thus, the basis vector matrix necessarily has $K^2 * K^2$ elements.

Chapter 2

Basis Vector Matrix Construction (K = 16)

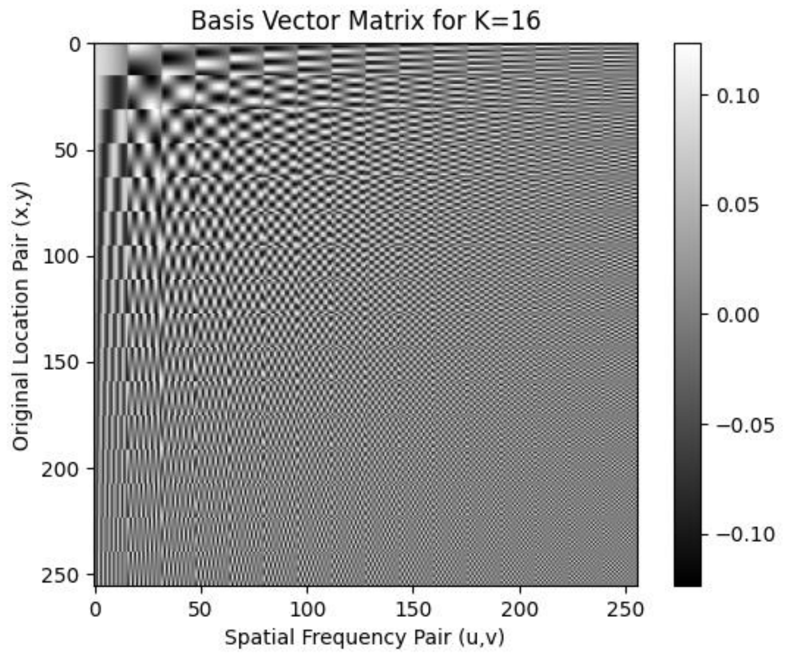


Figure (19):
Basis Vector Matrix Visualization for K=16

Given the mathematical background for generating the basis vector matrices on the previous slide, we can now utilize the parametric equation in our methodology.

In order to generate all $K^2 * K^2$ elements for our basis vector matrix, I constructed a series of four nested loopsK^2 basis vectors, each with K^2 elements, this necessitates a $K^2 * K^2$ basis vector matr. The easiest way to think about the reason why there are that number of elements in the matrix is that for all K^2 basis vectors, which are the K^2 number of (u, v) pairs, there are K^2 (x, y) pairs. Or the elements of each basis vector. If there are ix.

When the word "pair" is mentioned, the basis vector matrix contains *all* pairs, but what does that mean?

We have previously defined that both x and u are in the set of numbers of pixels in the horizontal direction from 1 to P. Similarly, y and v are in the set of numbers of pixels in the vertical direction from 1 to Q. Because we are breaking the image into $K * K$ pixel blocks, both P and Q are necessarily equal to K. Thus, the set of all (u, v) pairs will include {(u = 1, v = 1), (u = 1, v = 2), …. (u = 1, v = K), …. , (u = K, v = K)}, or K^2 pairs.

Similarly, the set of all the (x, y) pairs will have K^2 elements, and as stated previously, each of the K^2 (u, v) pairs has all the K^2 (x, y) pairs. By looping over u, v, x, and y, we can construct our basis vector matrix in painstaking $\Theta(K^4)$ fashion.

Choosing the Regularization Parameter

As stated previously, we are going to reconstruct our image by solving the minimum L1 norm optimization problem $\min_{s} \|\Theta s - y\|_2 + \lambda \|s\|_1$ for each block and then concatenating the reconstructed blocks together.

We can leverage sklearn's LASSO linear modeling library[11] in order to compute the sparse solution. However, one decision we must make is our choice of regularization parameter, λ. This parameter is the way in which we find sparse solution, as a value $\lambda = 0$ gives the ordinary least squares solution and $\lambda \to \infty$ gives $\mathbf{s} = 0$.

But how do we choose the regularization parameter?

Our answer to this is empirically through cross-validation with random subsets. For each KxK block, we have already established there are 'S' number of sampled, or sensed, pixels, with the rest being "corrupted". We choose a large range of λ to test experimentally, from $1e-6$ to $1e+6$, with a few values from each decade. I leveraged NumPy's logspace function[12], generating 40 values from the range in log base 10 including the endpoint.

For each of the 40 λ values, I ran cross-validation with random subsets 20 times.

Chapter 2

Cross-Validation with Random Subsets

For each of the 40 λ values, I ran cross-validation with random subsets 20 times.

First, I partitioned the 'S' sampled pixels in the given block into $m = \left\lfloor \frac{S}{6} \right\rfloor$ "**testing pixels**" and (S-m) "**training pixels**". These pixels were partitioned randomly.

Second, for only the (S-m) **training pixels**, I calculated the DCT coefficients for the given λ value by using sklearn's LASSO function and a basis vector matrix composed of the rows from the original basis vector matrix corresponding to the **training pixel locations**.

Third, with the **DCT coefficients calculated from the training pixels**, I recovered the block using LASSO's sparse solution.

Fourth, from this recovered block, I then took **the (x, y) coordinate locations of each of the pixel locations in the testing set**, and I calculated the mean squared error comparing the **testing pixels' locations** to the original image's corresponding pixel locations.

Fifth, I added up the MSE values from each of the 20 cross-validation runs and took the average MSE value over those 20 runs.

Sixth, I created an array of tuple pairs matching each of the λ values in my generated range to their calculated average MSE value.

Seventh, out of all the (λ , MSE) pairs, I chose the λ value with the minimum MSE value in the array.

With this particular λ value, I then used sklearn's LASSO function to again calculate the block's basis vector coefficients but using all 'S' sampled pixels as my data.

Finally, I recovered the particular block using the basis vector coefficients through use of the inverse DCT.

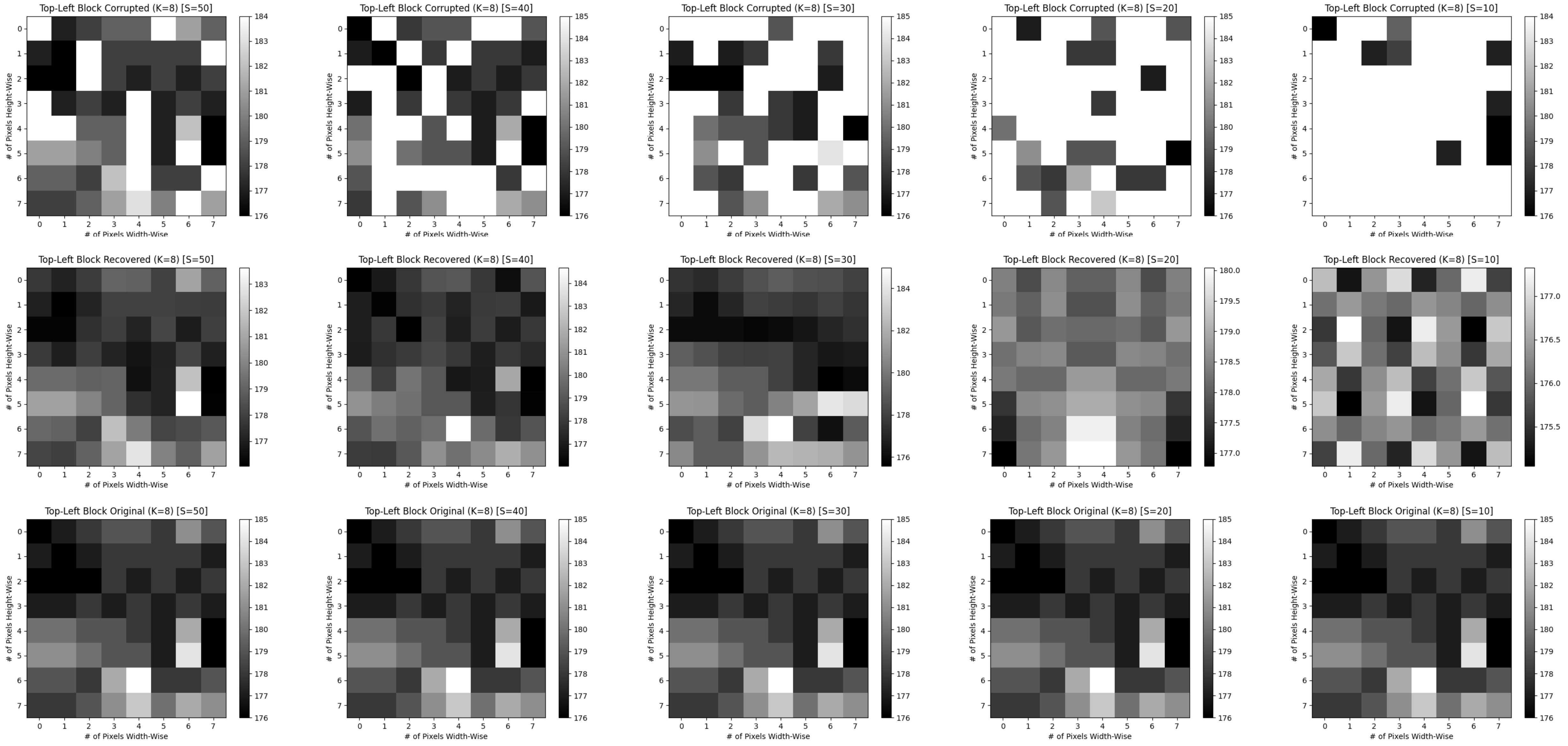
By repeating this process for each of the KxK blocks, I could in theory then concatenate all the recovered blocks together to obtain the final recovered image.

The way I concatenated the blocks together is by generating each recovered blocks from left to right and stitching each block together to form a row. I concatenated each of the rows together column-wise to obtain my final recovered image.

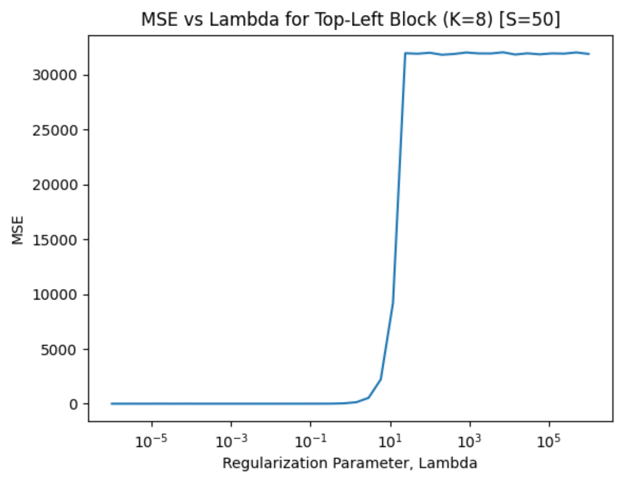
I also plotted MSE as a function of regularization parameter for each block.

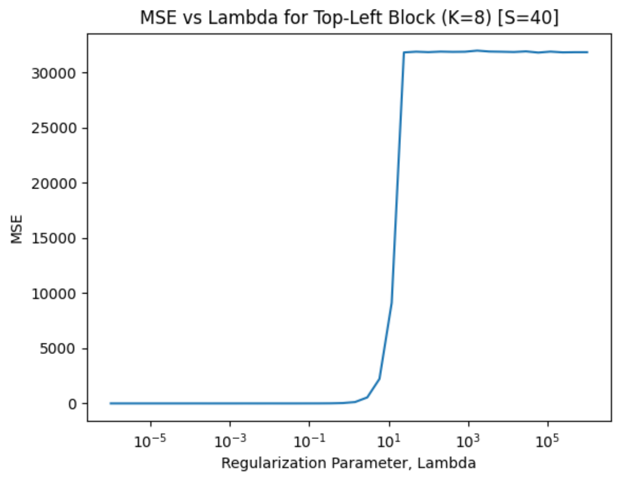
Fishing Boat Reconstruction

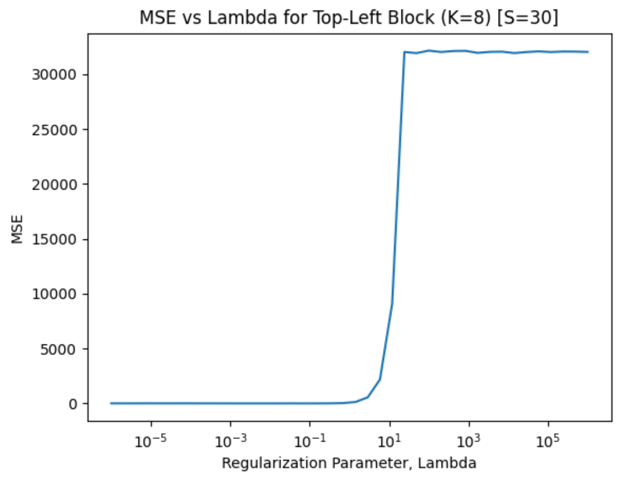
Results at the Individual Block Level

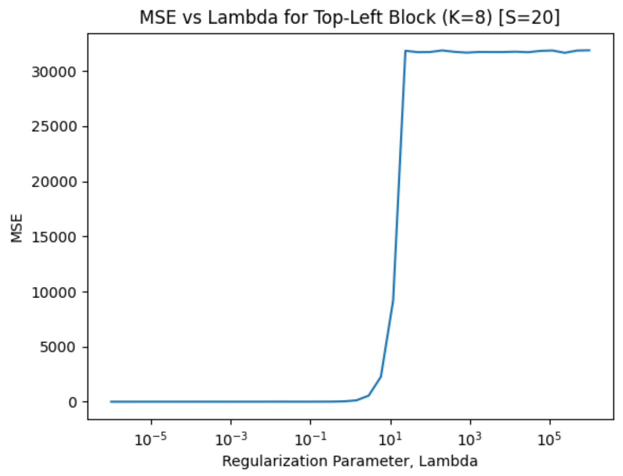


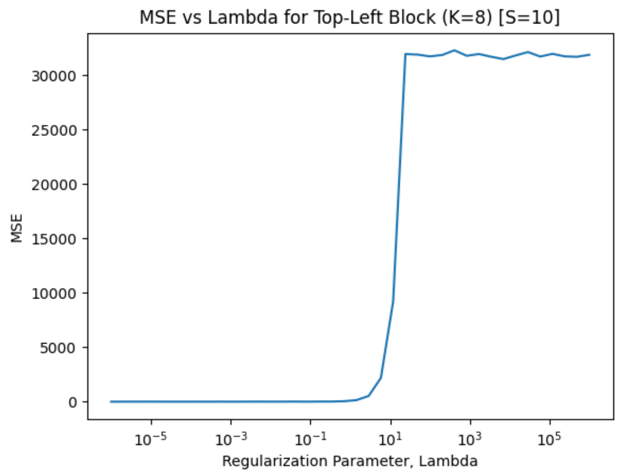
Figures (20 – 34): Fishing Boat Visualizations for Corrupted Blocks vs Recovered Blocks vs Original Blocks for K = 8 and various S values









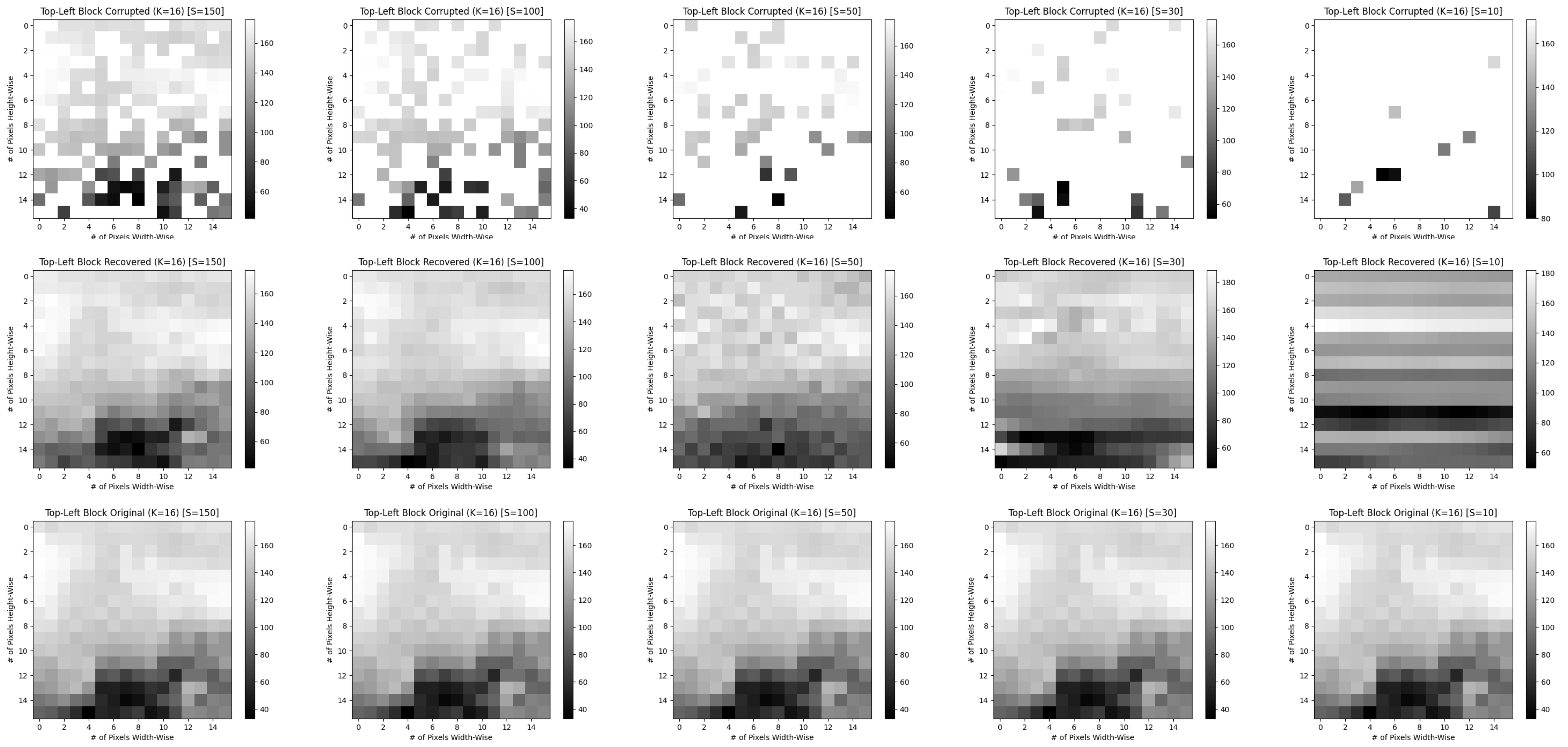


Figures (35 - 39):

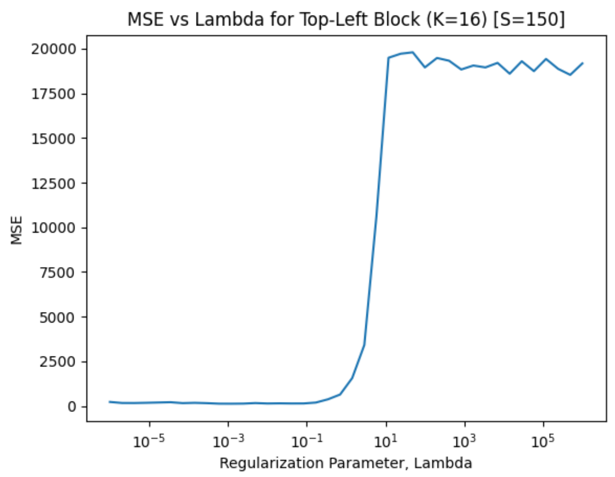
Total MSE values plotted against the chosen regularization parameter for the Top-Left Fishing Boat Block for K = 8 and various values of S.

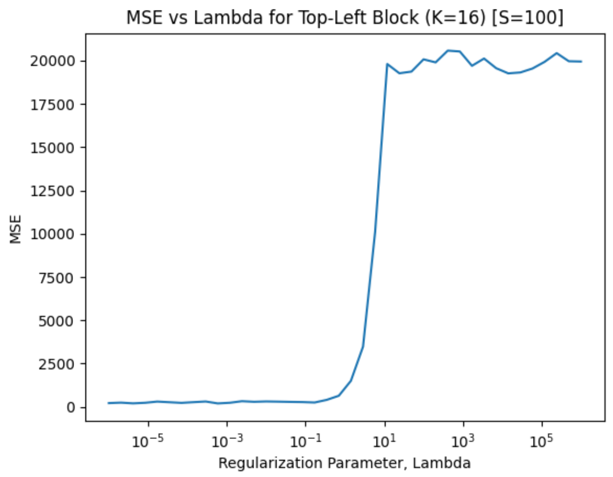
Nature Reconstruction

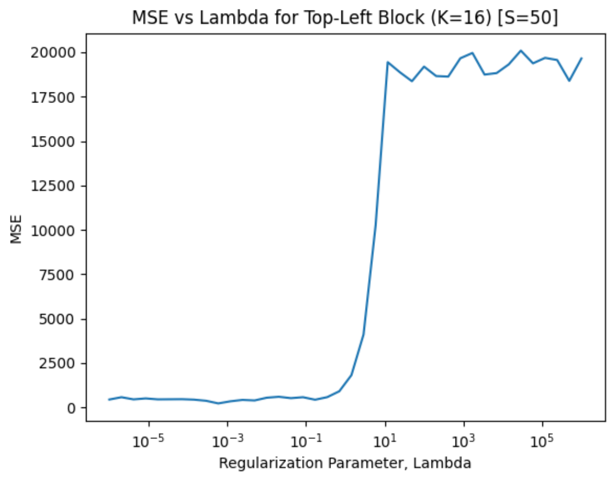
Results at the Individual Block Level



Figures (40 - 54): Nature Visualizations for Corrupted Blocks vs Recovered Blocks vs Original Blocks for K = 16 and various S values

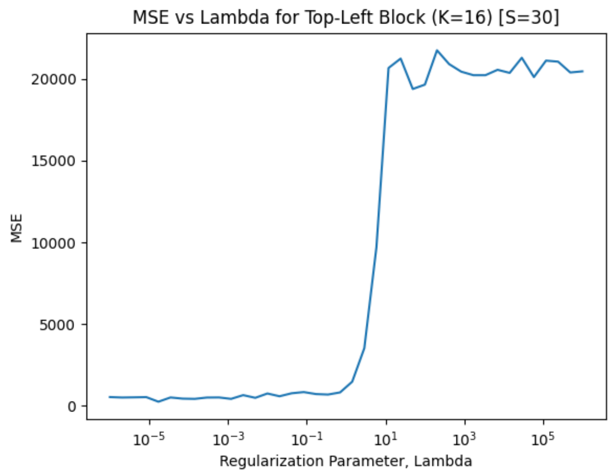






Figures (55 - 59):

Total MSE values plotted against the chosen regularization parameter for the Top-Left Nature Block for K = 16 and various values of S.



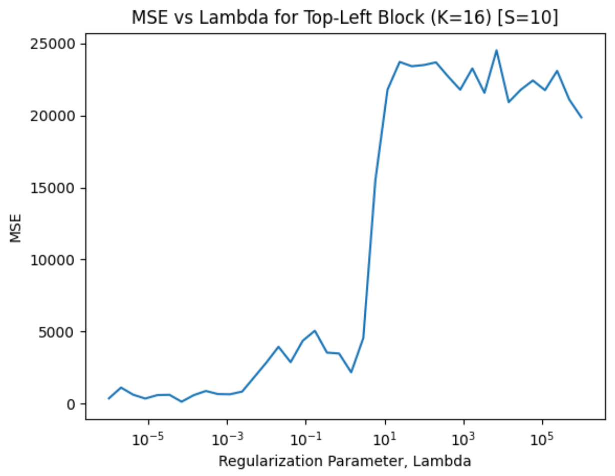


Image Reconstruction (Image Level)

3

Full Image Reconstruction

Methodology

Given that we know how to corrupt and reconstruct blocks, to obtain our full reconstructed image, it is only a matter of concatenating these blocks together. My method was to concatenate blocks in rows, and then concatenate the rows together column-wise, thus forming our original image.

We also implemented median filtering, of which we utilized sklearn's medfilt2 library[13], with a kernel size of 3.

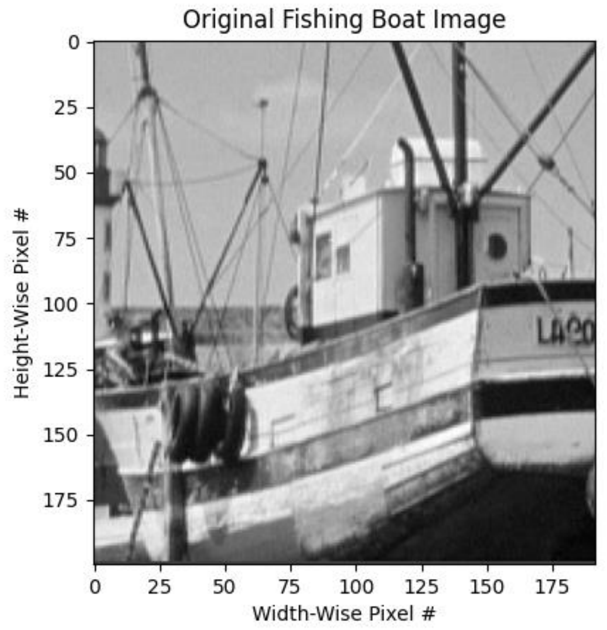
Finally, I visualized the log of each block's regularization parameter as a function of image block.

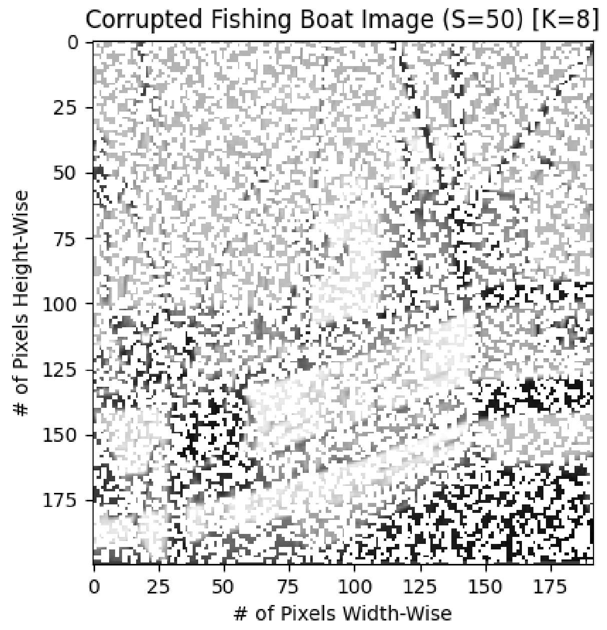
Fishing Boat Reconstruction

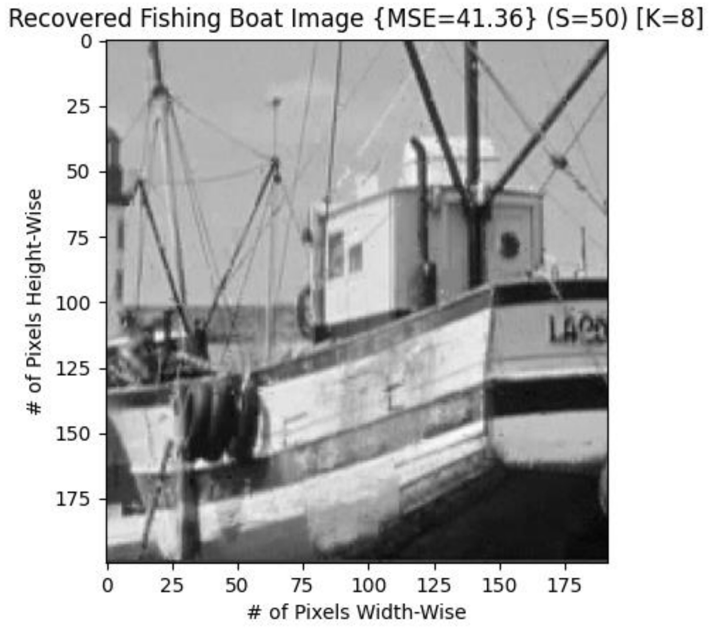
Results at the Image Level

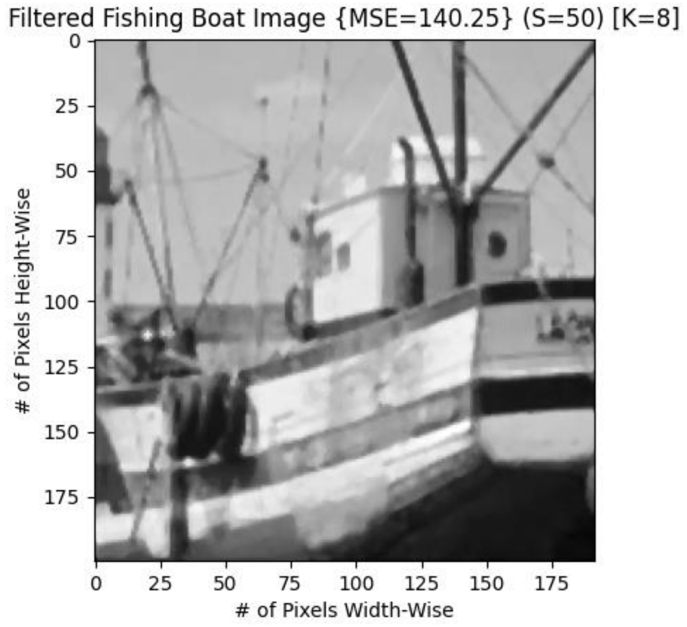
Chapter 3

Fishing Boat Reconstruction S = 50, K = 8







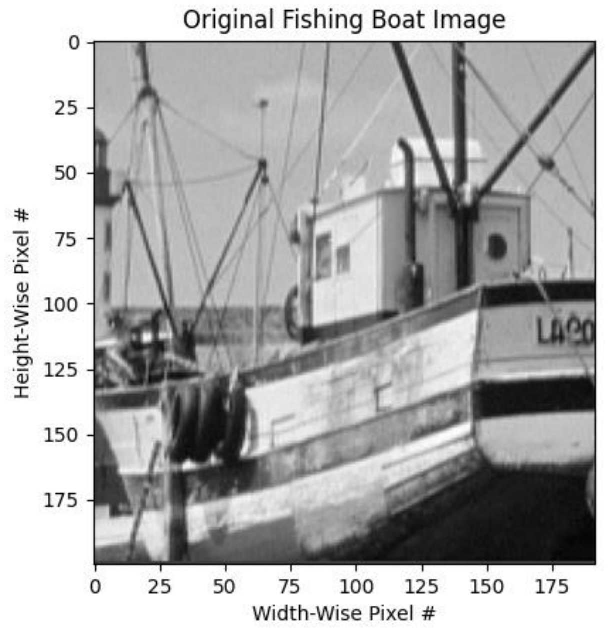


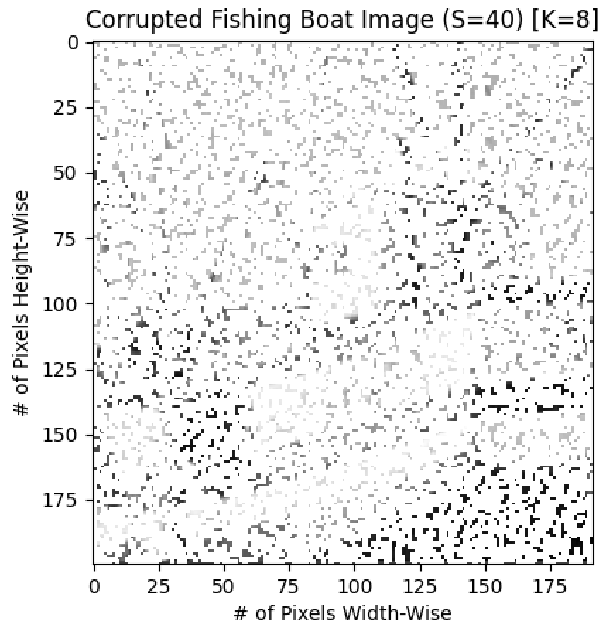
Figures (60 - 63):

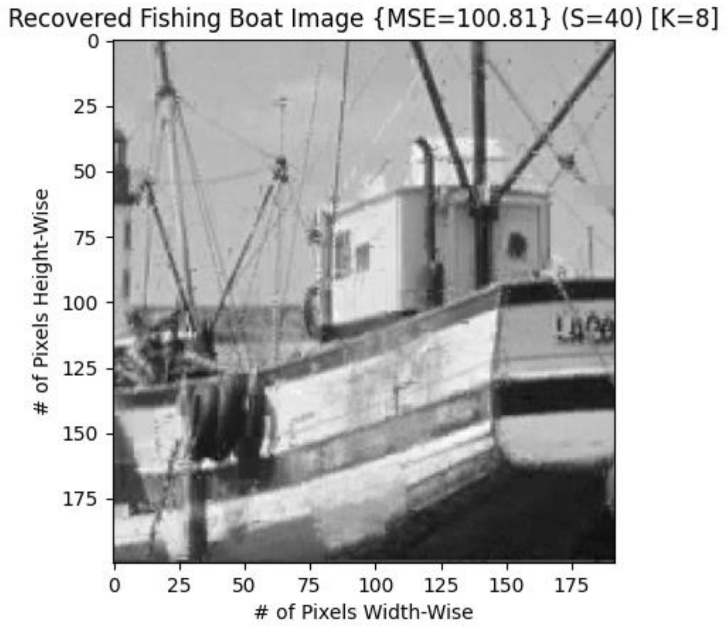
Fishing Boat full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 8 and S = 50.

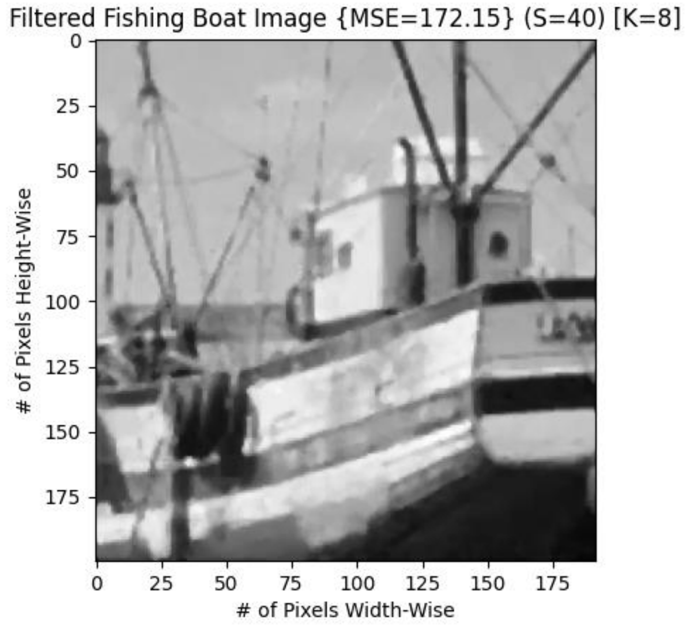
Chapter 3

Fishing Boat Reconstruction S = 40, K = 8







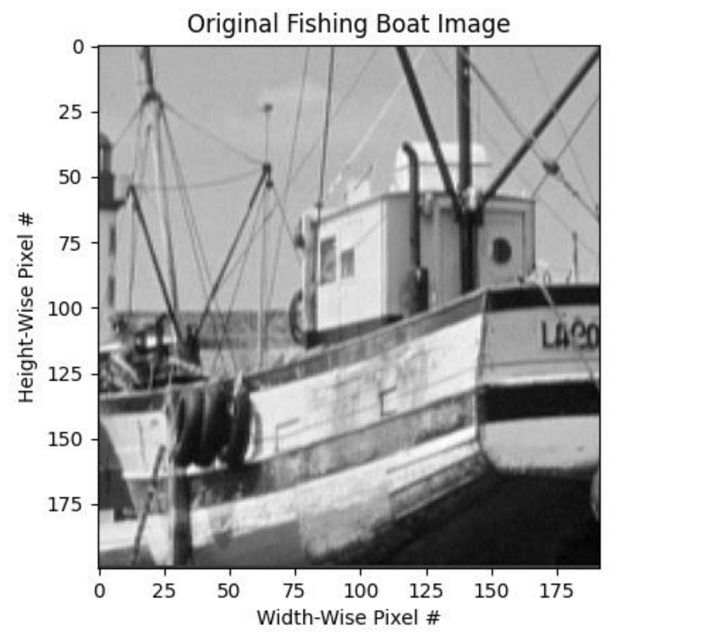


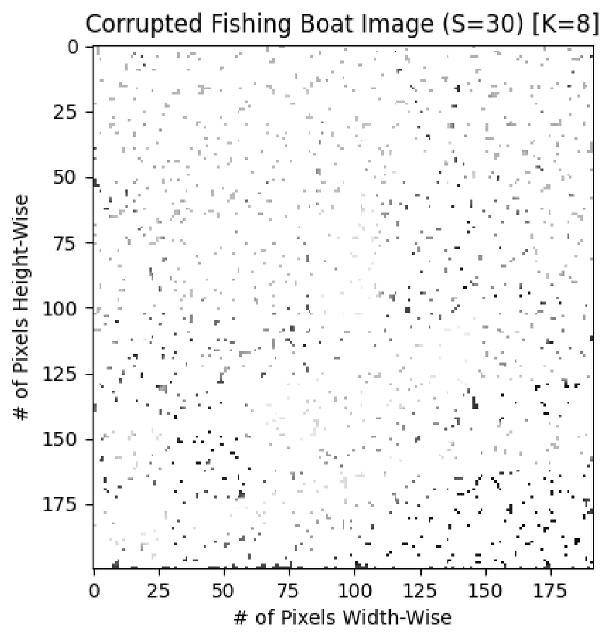
Figures (64 - 67):

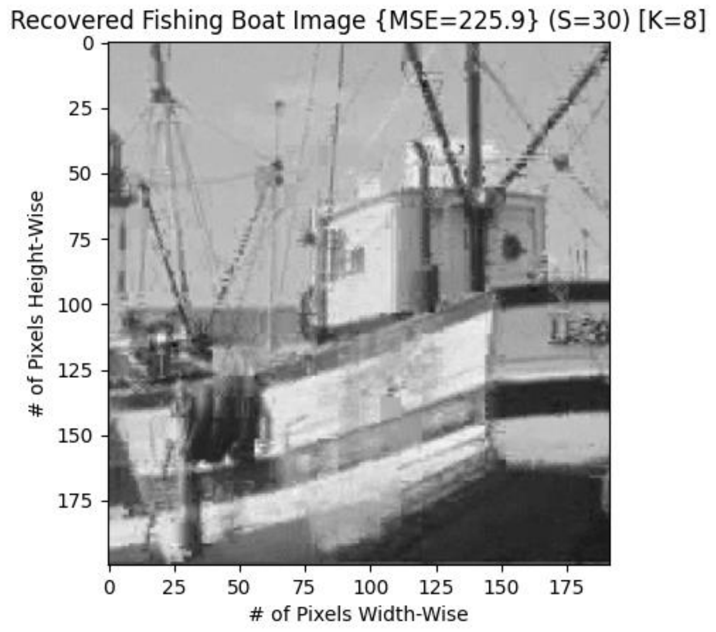
Fishing Boat full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 8 and S = 40.

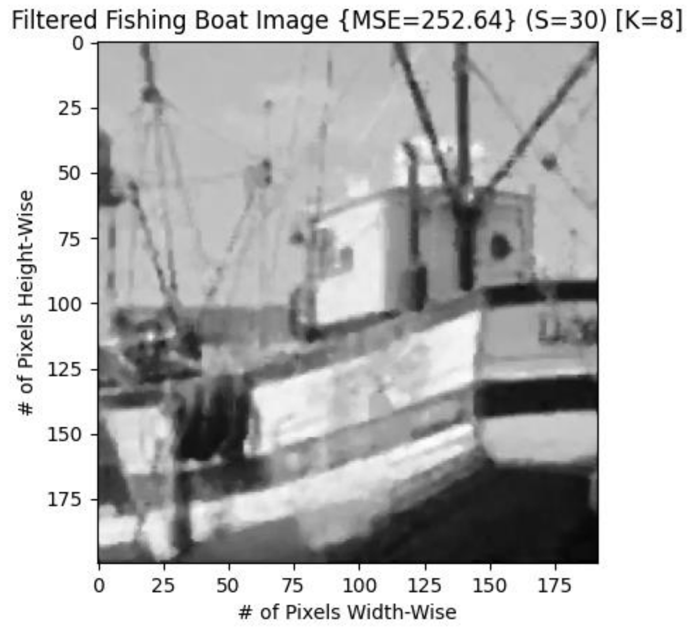
Chapter 3

Fishing Boat Reconstruction S = 30, K = 8







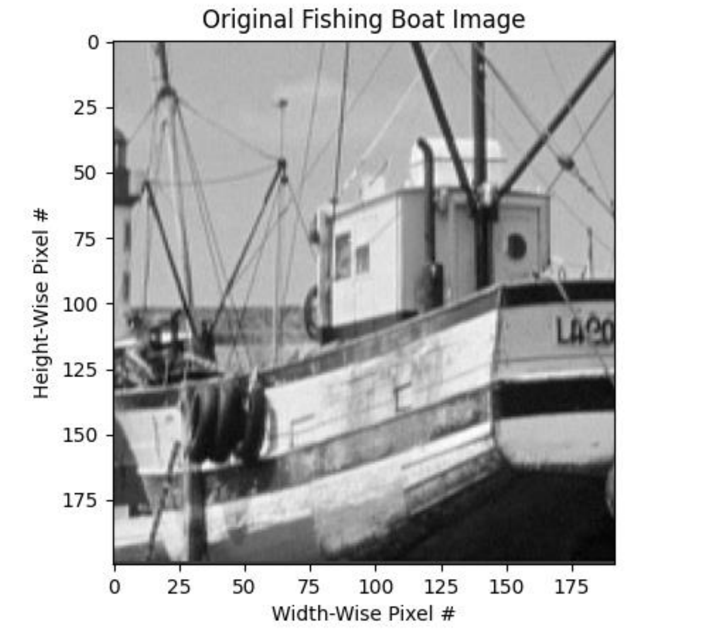


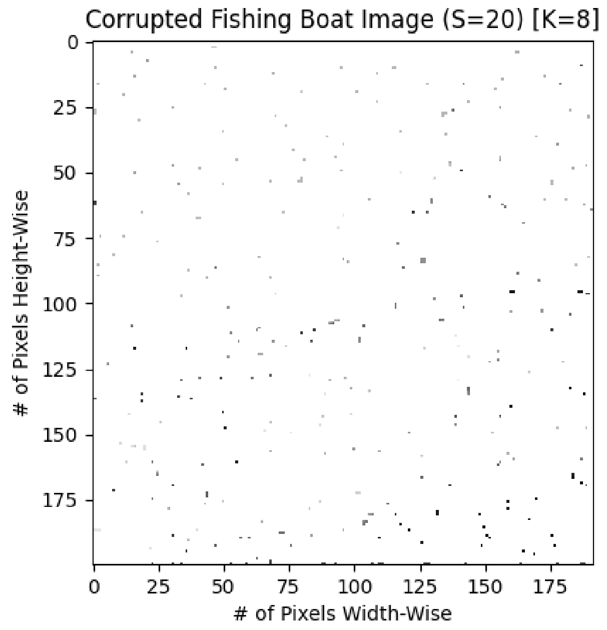
Figures (68 - 71):

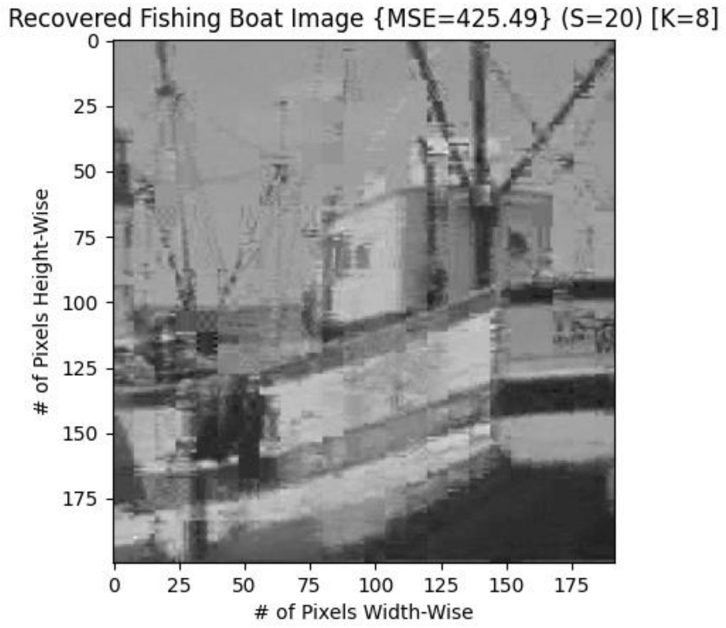
Fishing Boat full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 8 and S = 30.

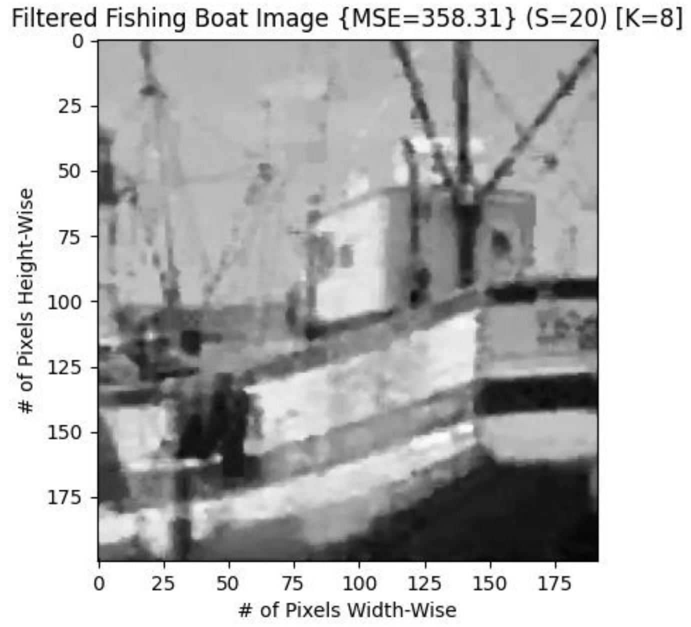
Chapter 3

Fishing Boat Reconstruction S = 20, K = 8





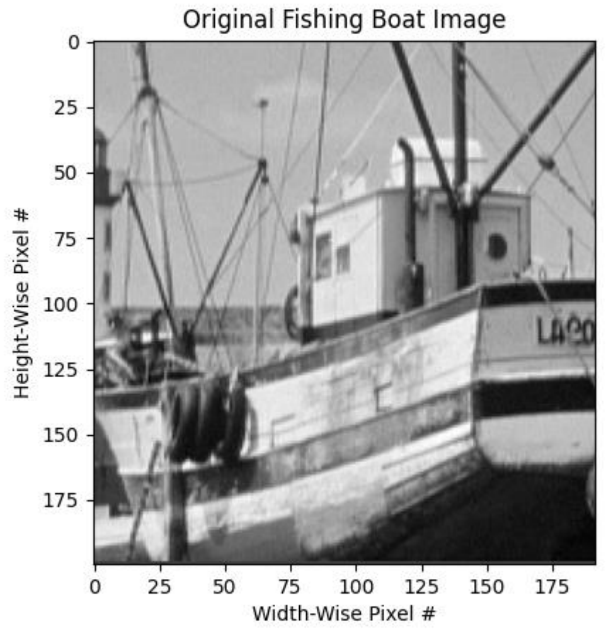


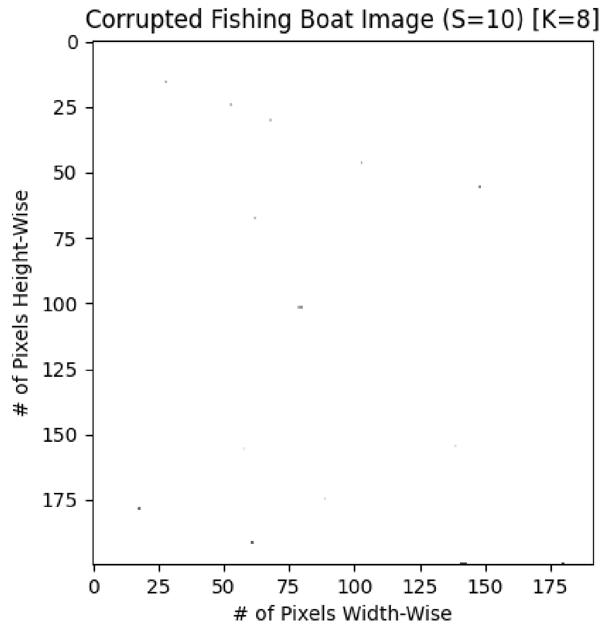


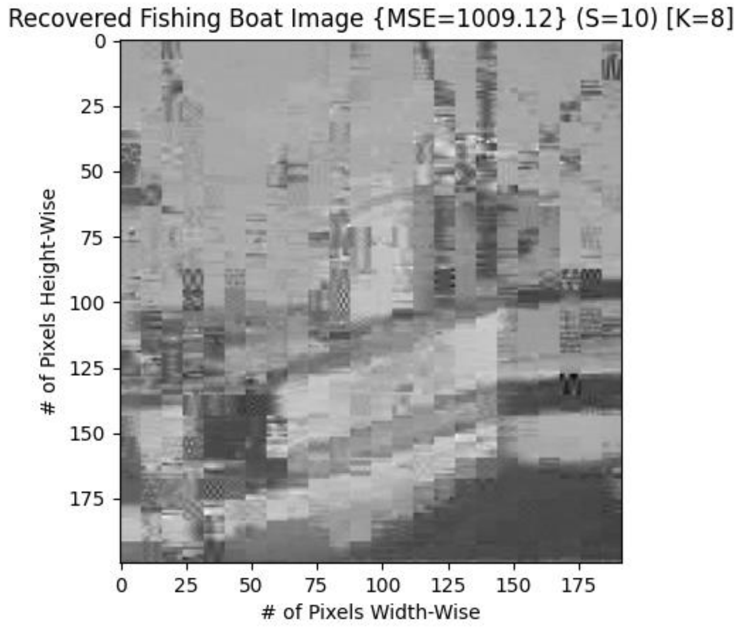
Figures (72 - 75):

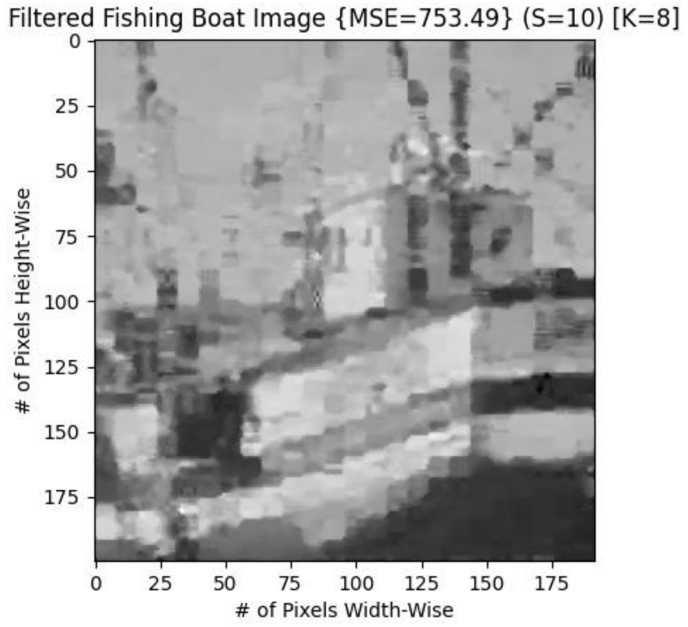
Fishing Boat full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 8 and S = 20.

Fishing Boat Reconstruction S = 10, K = 8



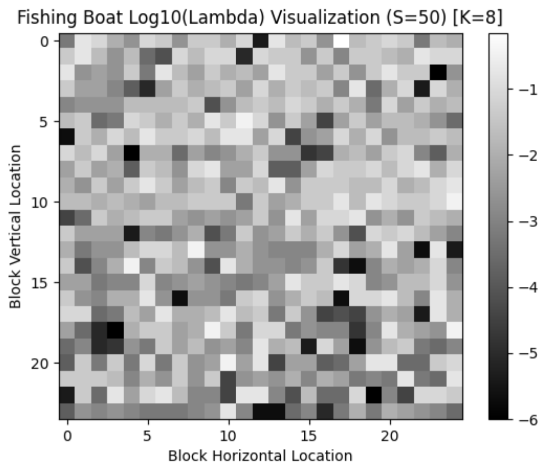


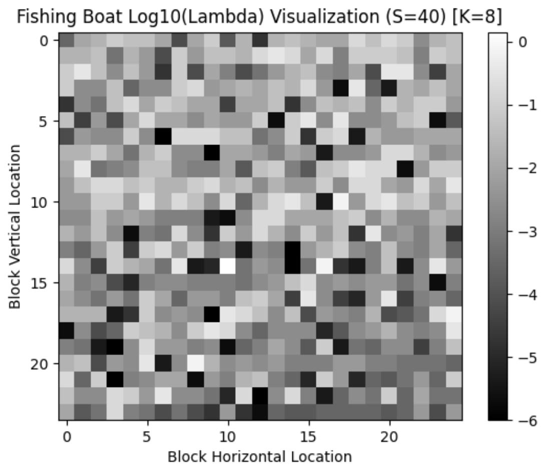


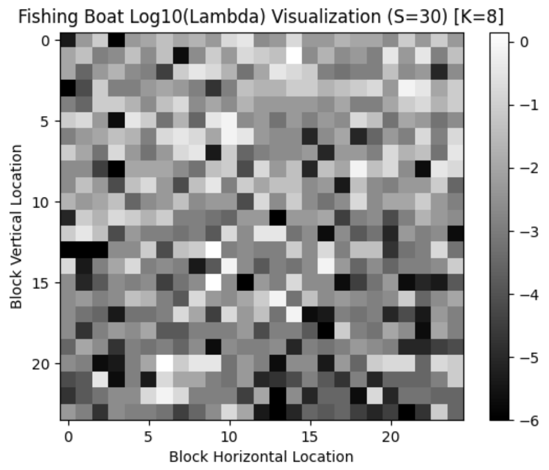


Figures (76 - 79):

Fishing Boat full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 8 and S = 10.

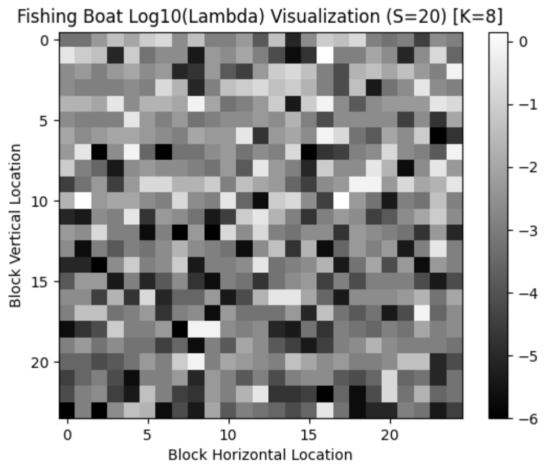


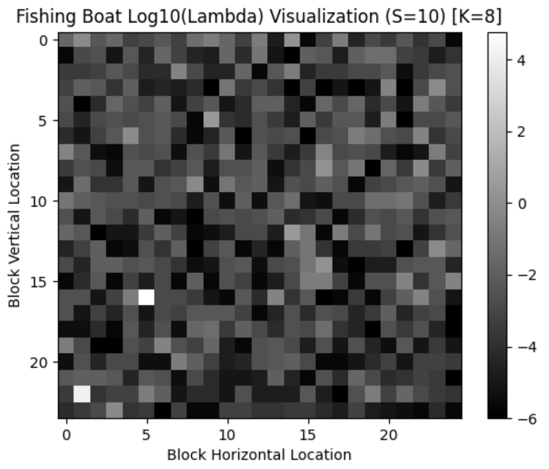




Figures (80 - 84):

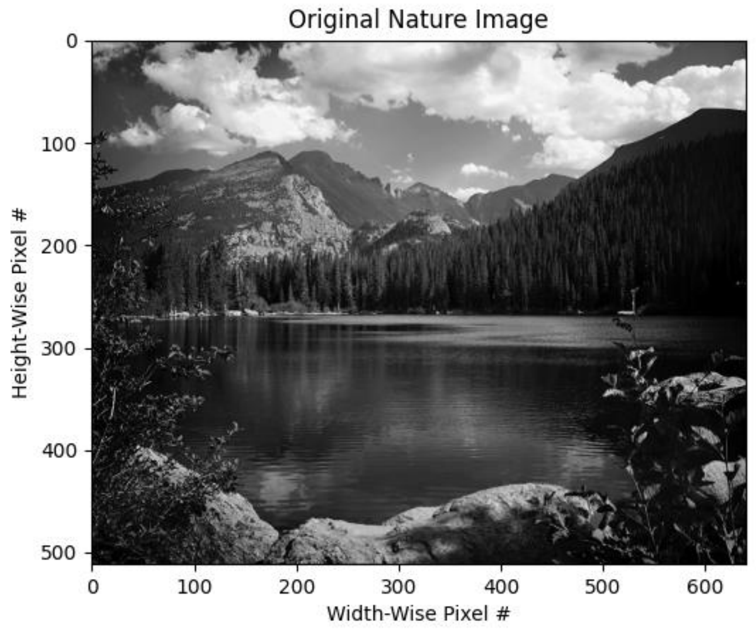
Each block's individual regularization parameter was linearly mapped to grayscale and plotted as a coordinate of its original top-left pixel's x-coordinate divided by k, and y-coordinate divided by k. We can track variations in the regularization parameter throughout the entire block this way.

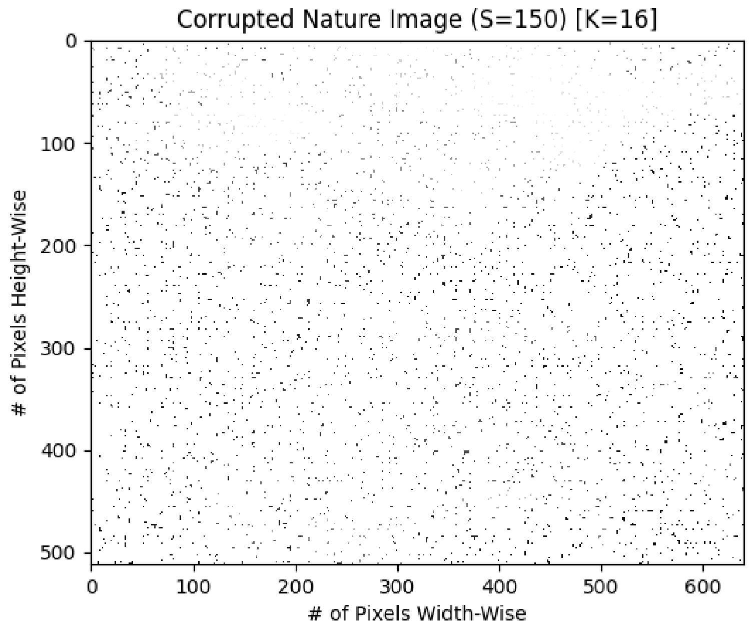




Nature Reconstruction

Results at the Image Level

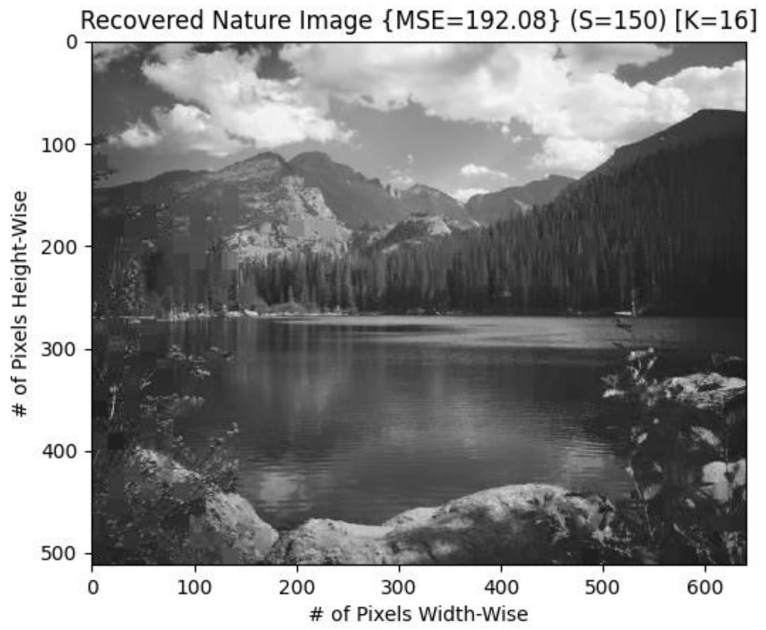


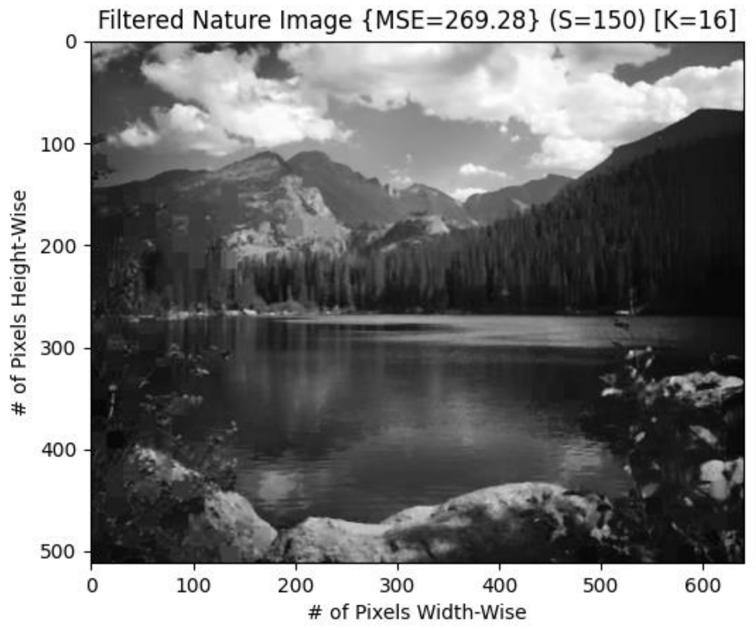


1 2 3 4 5 6

Chapter 3

Nature Reconstruction S = 150, K = 16



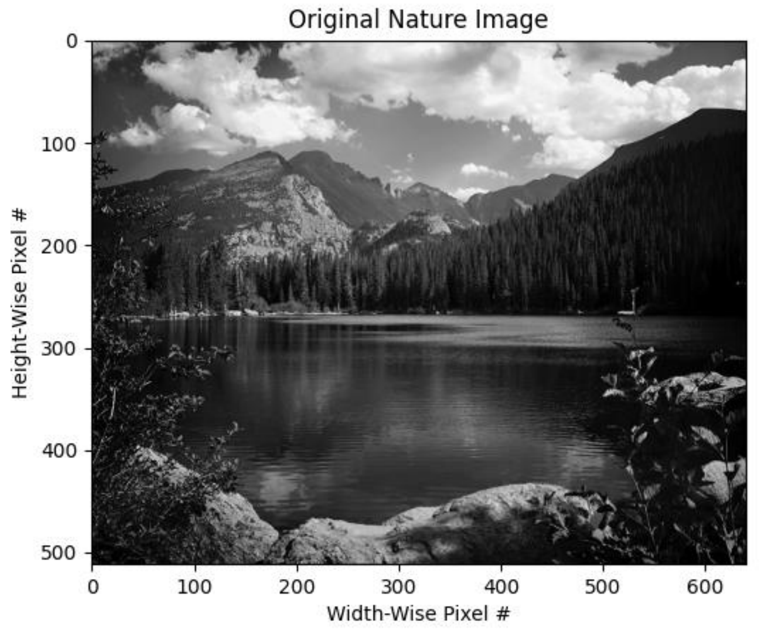


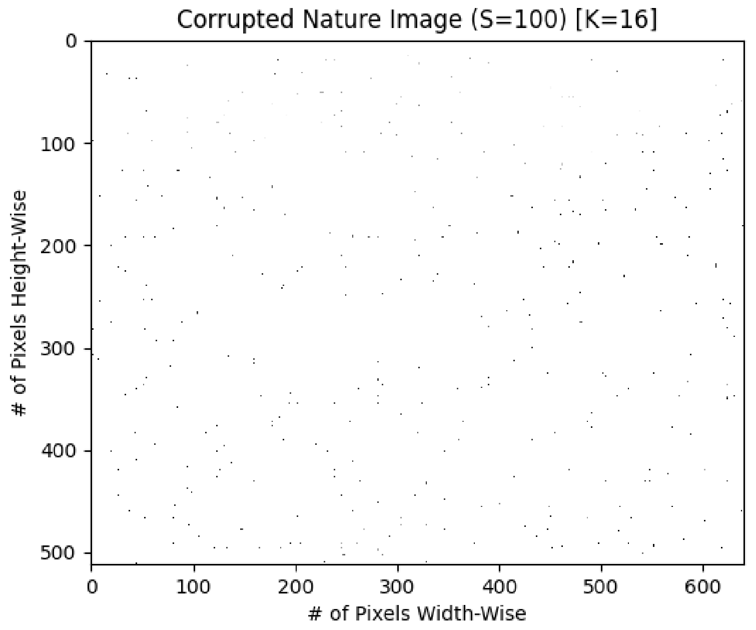
Figures (85 - 88):

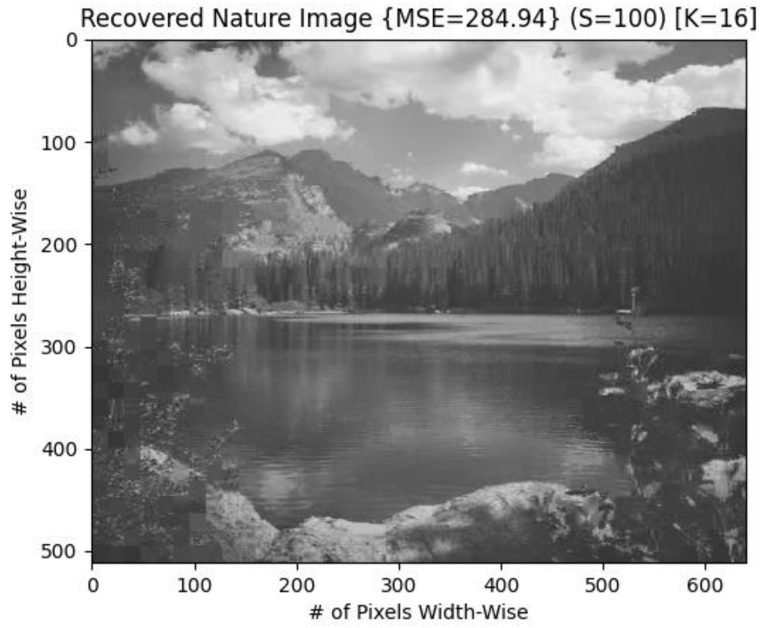
Nature full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 16 and S = 150.

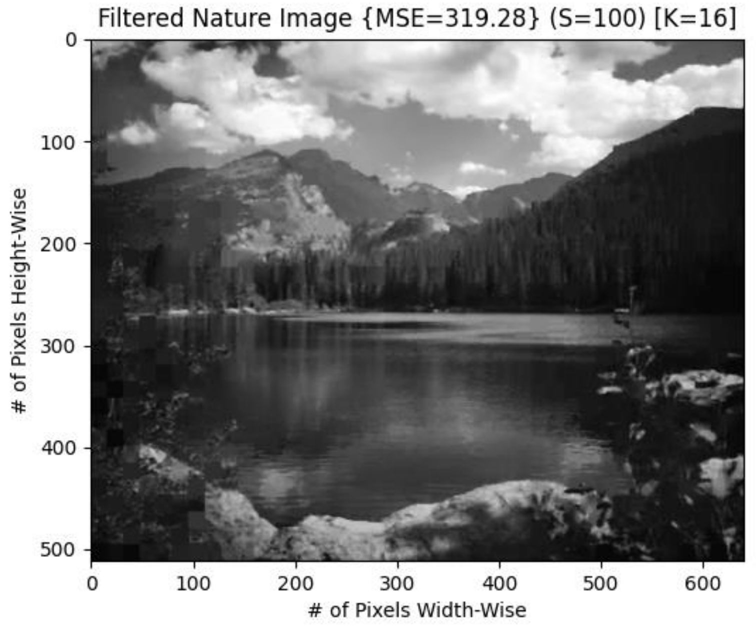
Chapter 3

Nature Reconstruction S = 100, K = 16







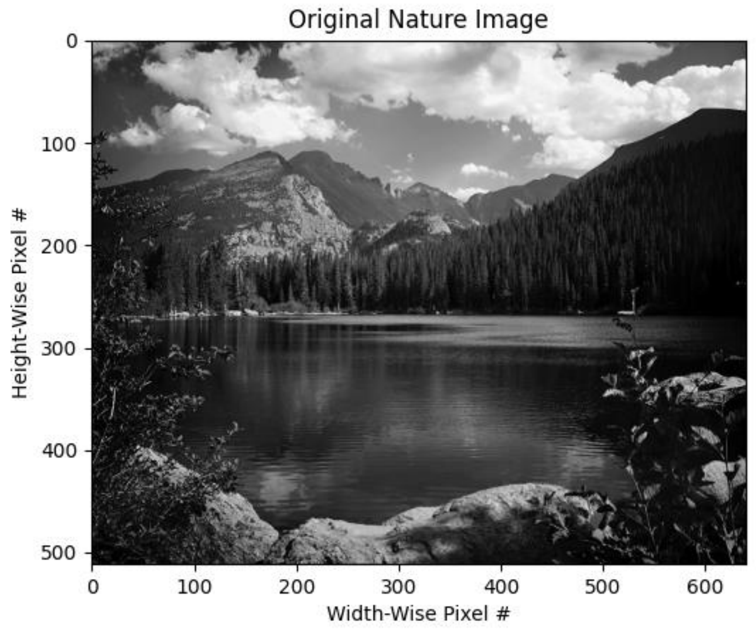


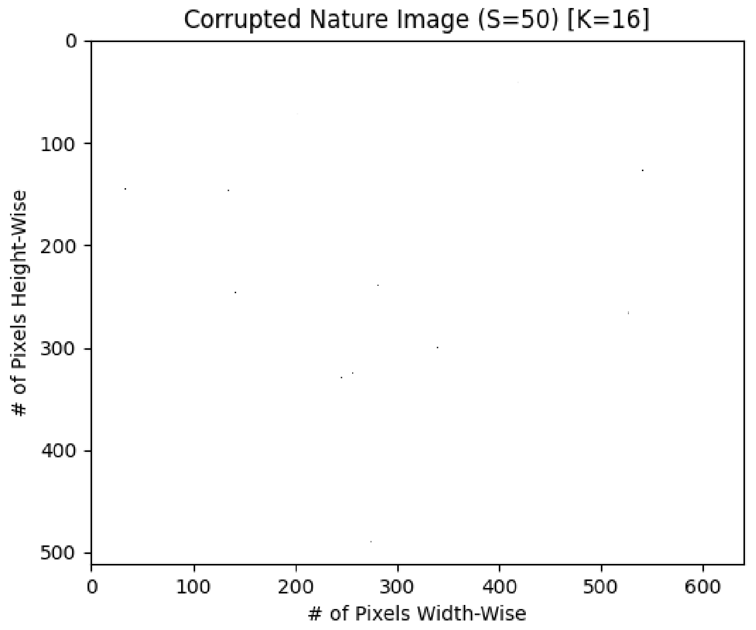
Figures (89 - 92):

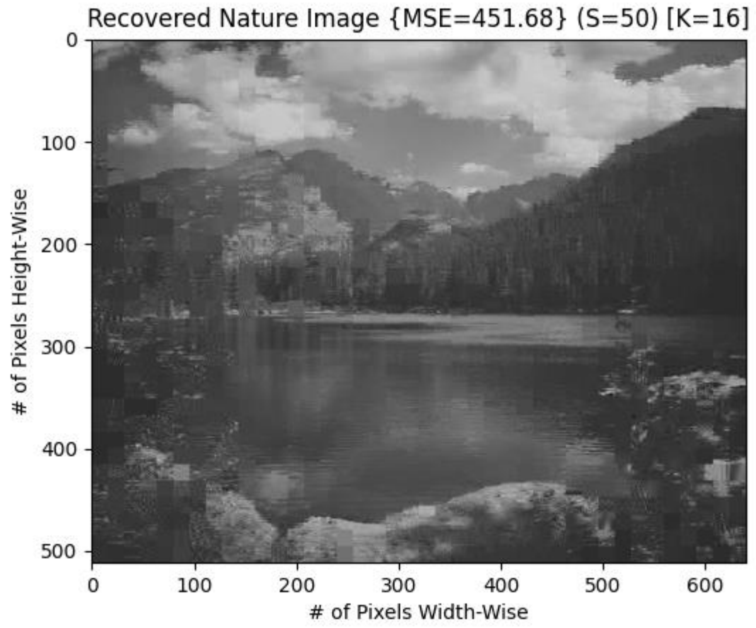
Nature full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 16 and S = 100.

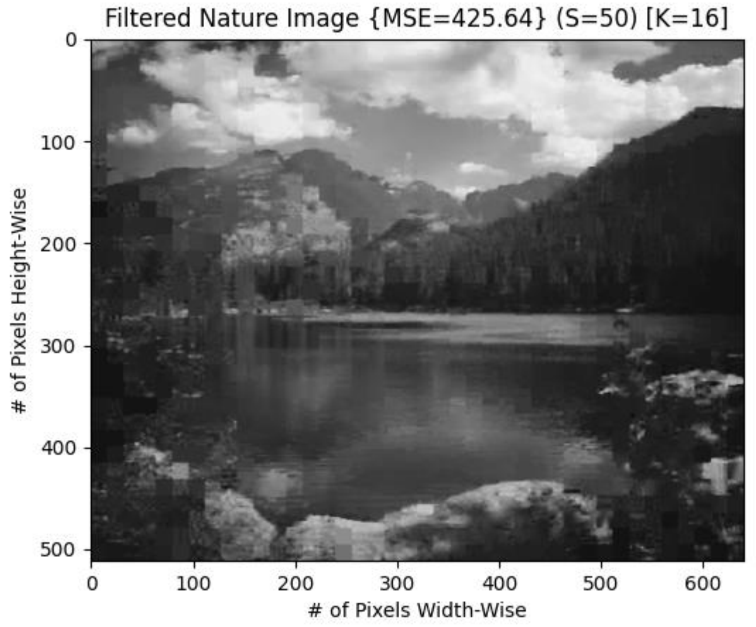
Chapter 3

Nature Reconstruction S = 50, K = 16







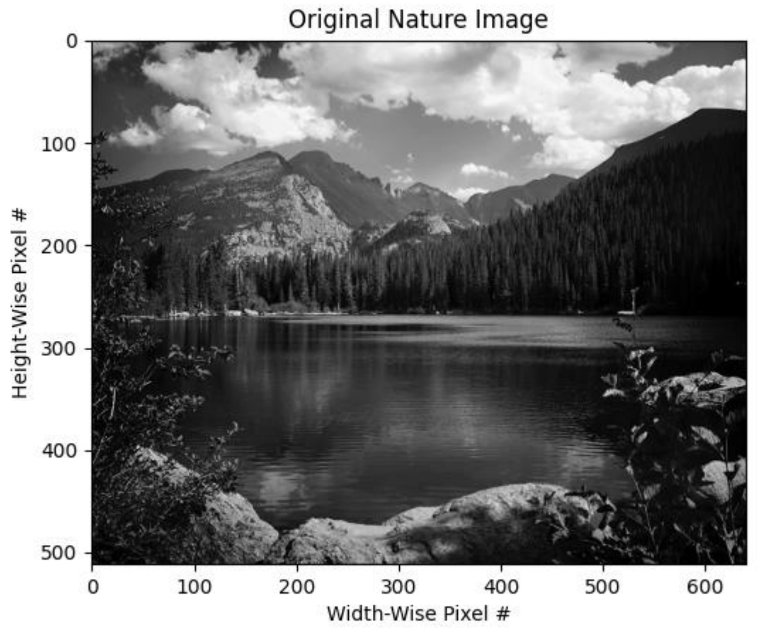


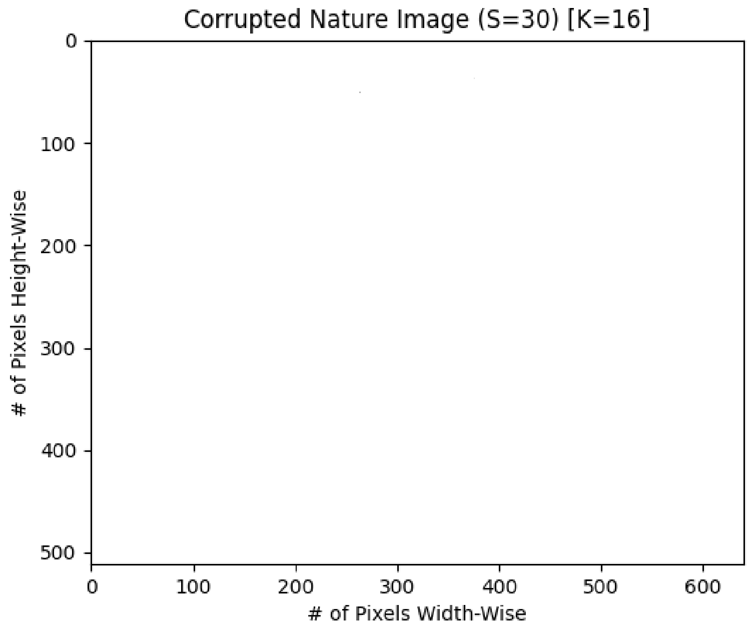
Figures (93 - 96):

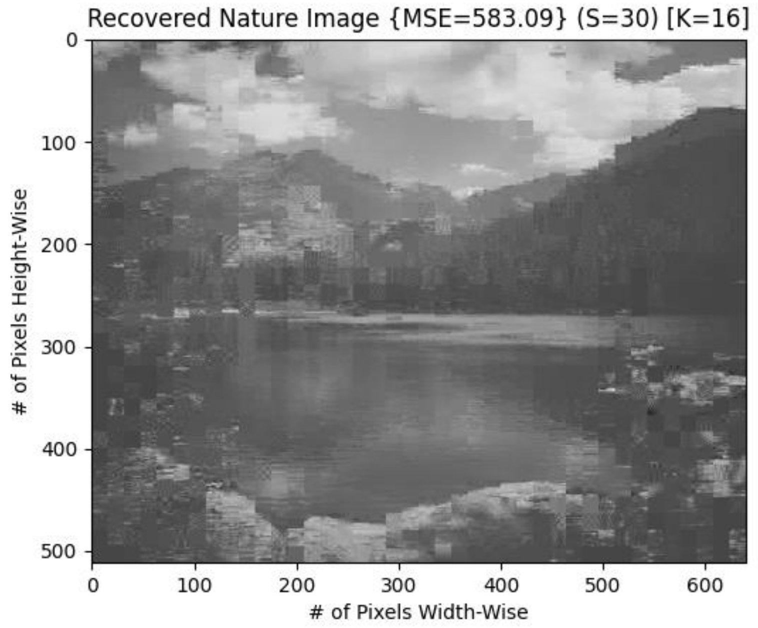
Nature full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 16 and S = 50.

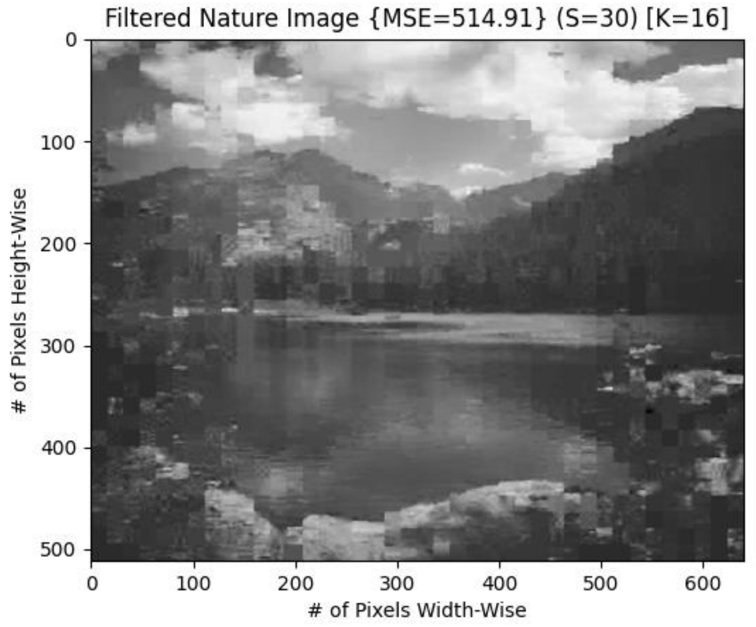
Chapter 3

Nature Reconstruction S = 30, K = 16







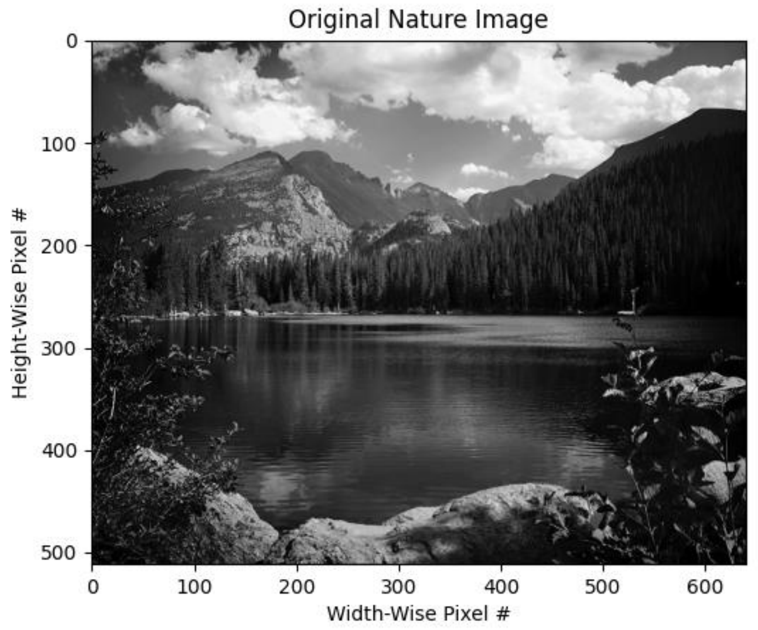


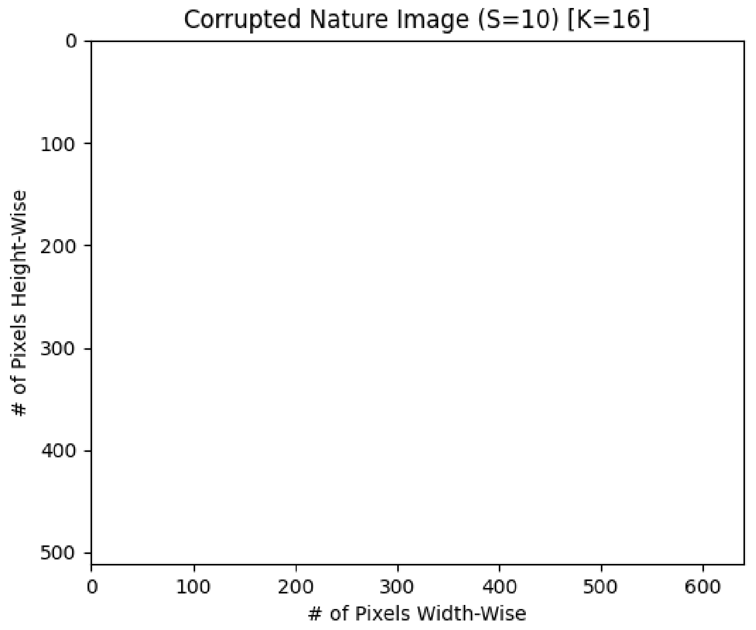
Figures (97 - 100):

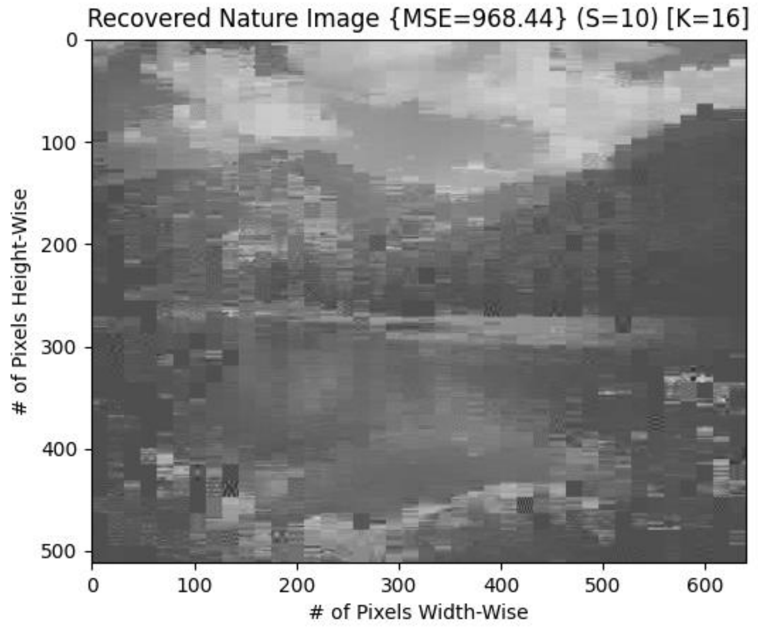
Nature full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 16 and S = 30.

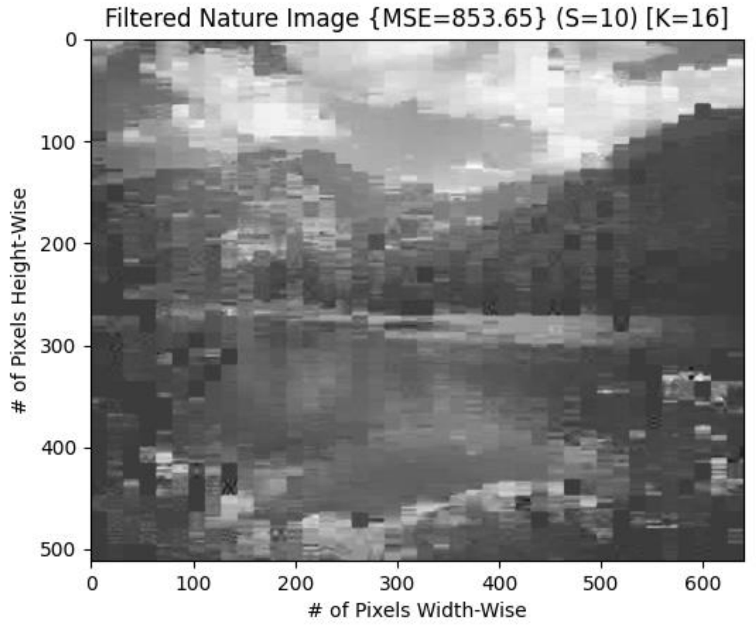
Chapter 3

Nature Reconstruction S = 10, K = 16



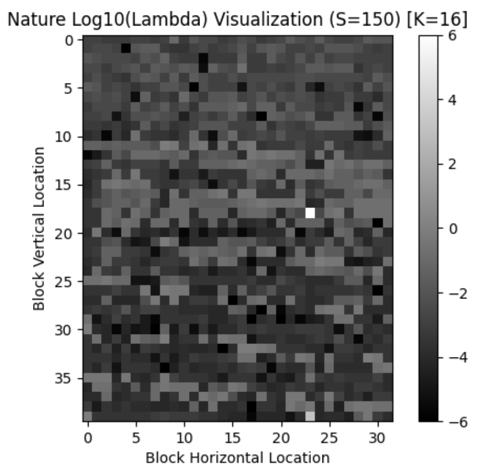


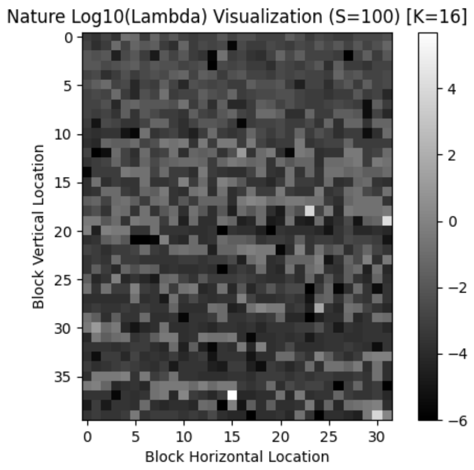


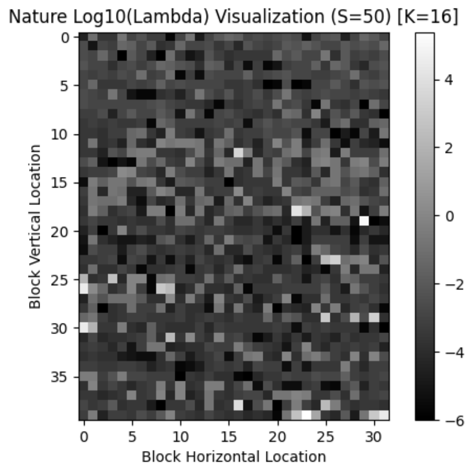


Figures (101 - 104):

Nature full image reconstruction visualizations for the Original image, the Corrupted image, Recovered image, and Median Filtered image for K = 16 and S = 10.

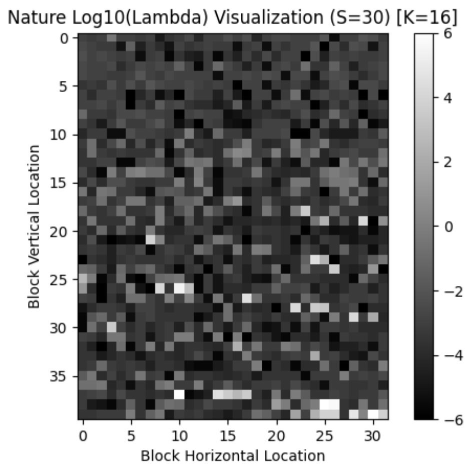


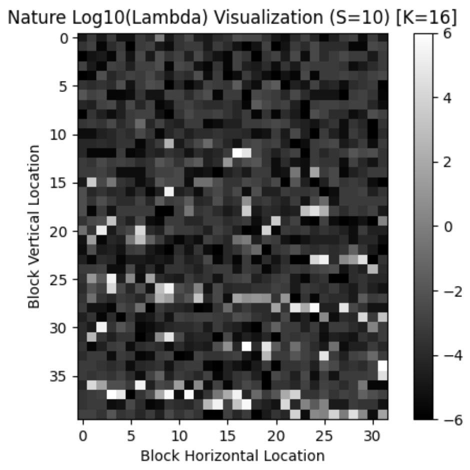




Figures (105 - 109):

Each block's individual regularization parameter was linearly mapped to grayscale and plotted as a coordinate of its original top-left pixel's x-coordinate divided by k, and y-coordinate divided by k.





4

Performance Comparisons

Chapter 4

Image Reconstruction Performance Comparison

As far as evaluating how “well” our image reconstruction model does, the primary quantitative measurement we will be taking is again the Mean Squared Error between our recovered image’s pixel intensities and our original image’s pixel intensities

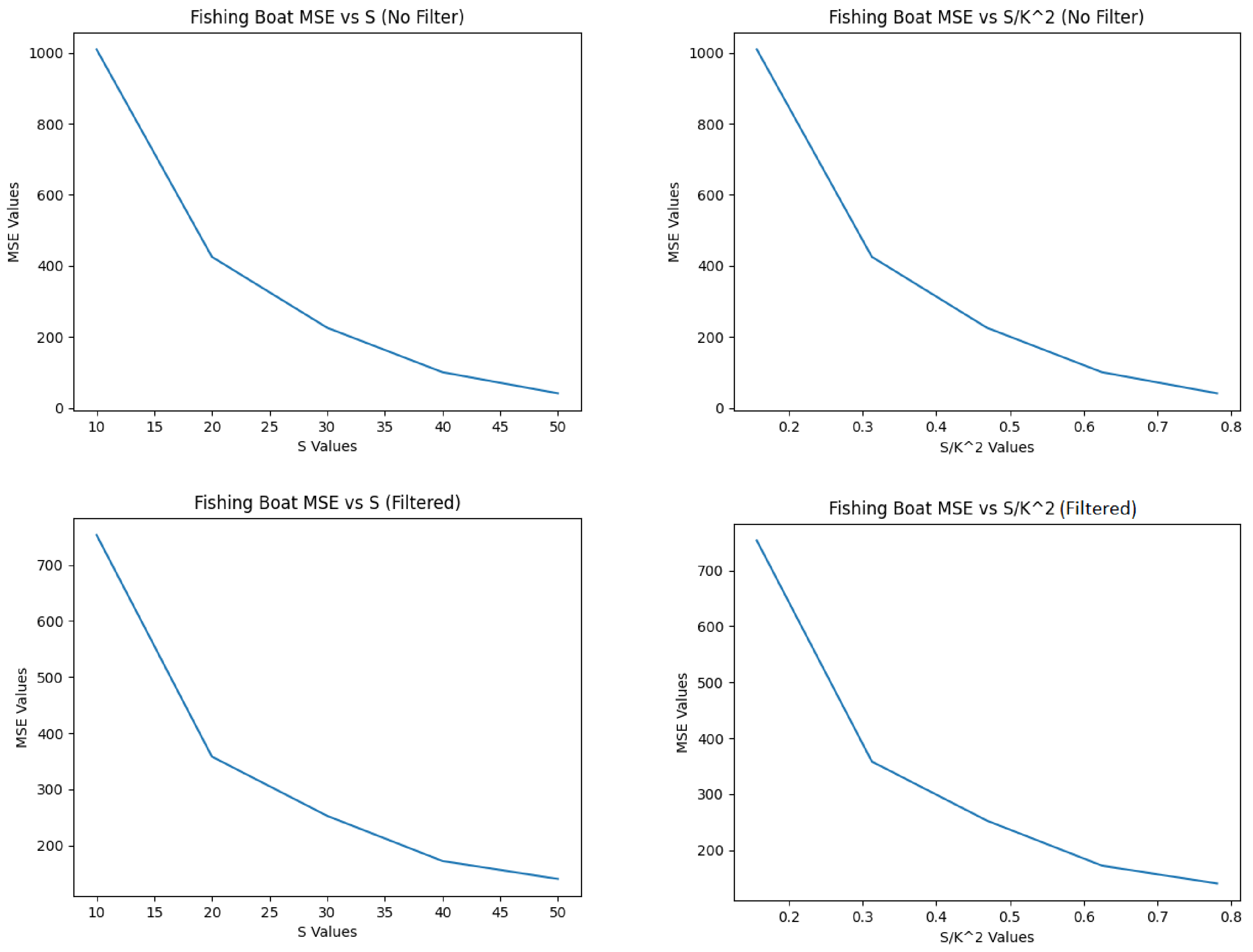
We will be comparing the MSE values for each of the sensed pixel quantities, in order to compare how well our model does when less and less samples are readily available to it. We also want to compare the MSE values against a function of the sensed pixels and block size.

Finally, we will be comparing the MSE values of our reconstructed image before and after we apply median filtering to it.

Fishing Boat Reconstruction

Performance Comparisons

Fishing Boat Reconstruction Performance Comparisons K = 8



Figures (110 - 113): Performance comparison graphs for the Fishing Boat reconstruction MSE values vs various S and S/K^2 values.

For the Fishing Boat image reconstruction for K = 8 across the various sensed pixel numbers (50, 40, 30, 20, 10), it is apparent and intuitive to see that the MSE is much higher the less sensed pixels there are.

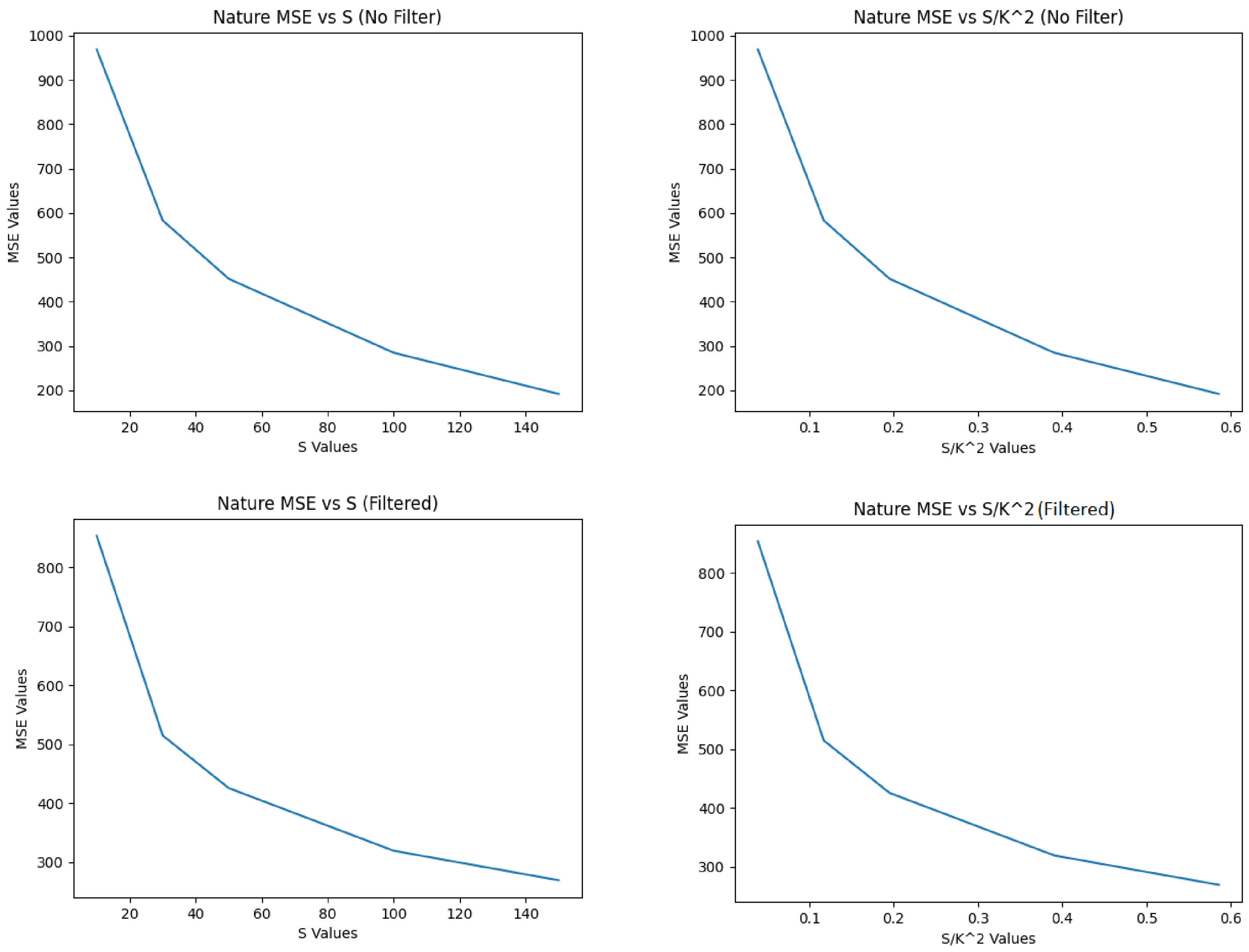
What is interesting to see, however, is that our median filtered image only begins to overtake our original reconstructed image in terms of MSE at a sensed pixel number of about 25 according to the graphs.

Similarly, when plotting the MSE value vs S/K^2, similar trends can be observed.

Nature Reconstruction

Performance Comparisons

Nature Reconstruction Performance Comparisons K = 16



Figures (114 - 117): Performance comparison graphs for the Nature reconstruction MSE values vs various S and S/K^2 values.

For the Nature image reconstruction for K = 16 across the various sensed pixel numbers (150, 100, 50, 30, 10), it is apparent and intuitive to see that the MSE is much higher the less sensed pixels there are.

Compared to the Fishing Boat image reconstruction, the curves of both the MSE vs S plots and the MSE vs S/K^2 plots are smoother at first glance, which suggests that the higher K and higher S values are responsible for both lower and more gradual values of MSE at smaller S values.

5

Interpretation & Design Approach

Project Approach

Explanations & Interpretations

Chapter 5

Image Recovery & Underdetermined Linear Systems

This image recovery problem is within the class of problems of solving underdetermined linear systems because of our constraint of sparsity. Natural signals such as images and audio are sparse in generic transform bases such as the Discrete Cosine Transform basis that we used in this project. Due to this sparse representation of the image, we can obtain our original image even when we only have a small number of sampled pixels due to being able to infer what the DCT coefficients are in the sparse domain. After calculating our DCT coefficients, we can use the basis vector matrix calculated from the parametric equation of the DCT in order to solve for our original image matrix.

Normally, there are an infinite number of solutions that would give us back a random image as is characteristic of the underdetermined linear system of equations. However, again due to our constraint of sparsity and looking for the sparsest solution, this is how we are able to obtain our original image.

Chapter 5

LASSO (L1-norm) Regularization Approach

The reason why LASSO regularization is applied to this project over other regularization approaches is because of our constraint of sparsity. **But why?** The L2 norm is most easily thought of as taking the Euclidean distance. And when we graph a plot of every vector that has the behavior of the L2 norm, we obtain a plot that looks like a geometric circle, which makes sense, as every point on the circle has an equal distance from the origin. **What about the L1 norm?** It is commonly known as the "Manhattan" or "Taxicab" norm as it represents the distance that a "taxicab" would take in the streets of Manhattan: lots and lots of right angle turns! Various geometric visualizations of norms are plotted in Figure (118). **So how does this apply to our original question of sparsity?** The infinite solution set to our underdetermined system of equations can be visualized as a line, with the intersection with the given p-norm being our returned solution from applying the respective regularization.

We want sparsity, meaning that one of our axes for the point of intersection will be zero, as shown by the L0-norm in Figure (119, a). This geometric visualization also shows how the L1-norm converges to the sparse L0-solutions, and why it is sparser than the L2-solution, and thus chosen.

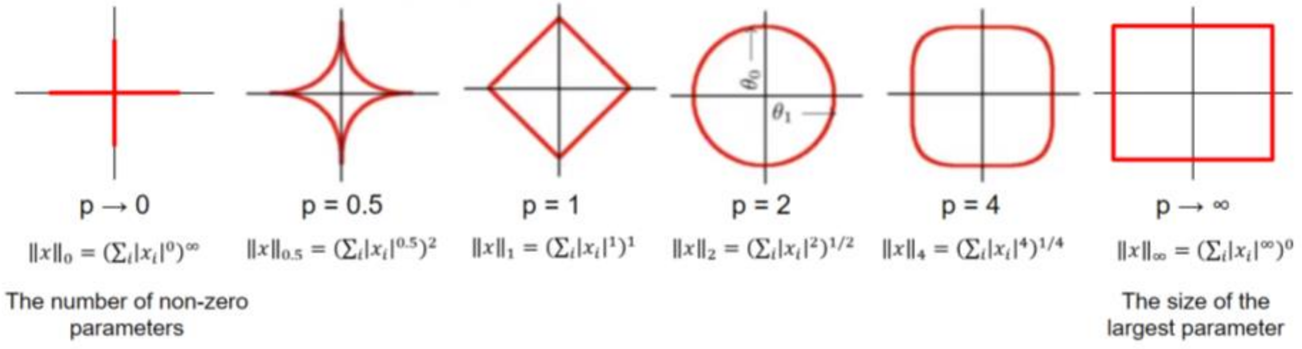

θ_0
θ_1
$p \to 0$
$\|x\|_0 = (\Sigma_i |x_i|^0)^\infty$
The number of non-zero parameters
$p = 0.5$
$\|x\|_{0.5} = (\Sigma_i |x_i|^{0.5})^2$
$p = 1$
$\|x\|_1 = (\Sigma_i |x_i|^1)^1$
$p = 2$
$\|x\|_2 = (\Sigma_i |x_i|^2)^{1/2}$
$p = 4$
$\|x\|_4 = (\Sigma_i |x_i|^4)^{1/4}$
$p \to \infty$
$\|x\|_\infty = (\Sigma_i |x_i|^\infty)^0$
The size of the largest parameter

Figure (118)[14] : Geometric visualizations for various p-norms

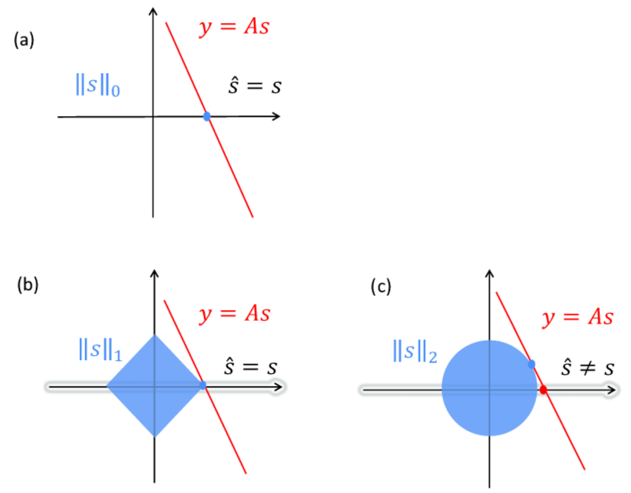


Figure (119)[15]: Solution intersections with the L0, L1, and L2 norms

Chapter 5

LASSO in Image Recovery & Constants in the Basis Vector

LASSO is applied to recover the image in a few steps. First, as previously described in the image recovery methodology, I selected a “best” regularization parameter for each block using the “training set” of pixels out of the total S sensed pixels. This regularization parameter was selected over 20 runs of cross-validation testing and training, and after a regularization parameter was calculated for a training set of pixels, I then used the testing set of pixels’ (x, y) coordinates in comparison to the same coordinates in the original image to calculate the Mean Squared Error for that particular regularization parameter. This process was repeated over 40 regularization parameter values between 1e-6 and 1e+6 logspace, with a couple values per decade. Finally, the regularization parameter with the minimum MSE value was ultimately chosen for the block.

The “constant” basis vector was able to be handled via sklearn LASSO’s parameter in the function. By setting the “fit_intercept” parameter to “False”, the intercept was not used in the calculation of the DCT coefficients.

Chapter 5

Median Filtering vs Other Filtering Approaches for Image Quality

The reason why filtering is applied to improve the image quality after applying LASSO is to reduce the amount of noise present in the image. What is meant by this is that outliers in the pixel intensities will be filtered out and thus the image quality will result in being a lot “smoother” after filtering.

The reason why we choose specifically median filtering is that outlier values will be replaced by values that are already present in the surrounding pixels, the median of the 3 by 3-pixel kernel. This is important because other filters, such as the Gaussian filter, will compute the replacement value with the weighted average, of which the outliers will tend to skew, thus making it less suited to reducing this specific type of noise than the median filter.[16]

Another reason why median filtering is chosen specifically is because it excels at preserving “edges” and other boundary lines in images. Again, using the Gaussian filter as an example, the weighted average of pixels necessarily means that each pixel will end up as an average of the ones around it. This will end up blurring edges, hence the name “Gaussian blur”.

Project Results

Explanations & Interpretations

Chapter 5

Variations in the Regularization Parameter

The regularization parameter varies over the entire block as shown in Figures (80 – 84) and (105 – 109) because each block has different DCT coefficients. When implementing LASSO for each block to determine the minimum MSE regularization parameter, a core parameter of LASSO is the basis vector matrix and the given sampled pixels. Each block naturally has different coordinates for its top-left origin pixel, and because of this, we select different rows from the basis vector matrix based off of these coordinates in the location domain.

With different basis vector matrices for each block, we will naturally have different DCT coefficients for each block, and thus different regularization parameters for each block.

Chapter 5

Variations in the Mean Squared Error vs Number of Sensed Pixels

It is intuitive to grasp that the more pixels that we have in our image, the better of a guess we can make as to what the neighboring pixels should be. As for our model, it is much the same.

In Figures (40 – 54), we can see that for the higher values of sensed pixels, our recovered block looks increasingly similar to the original block, whereas the lower end of sensed pixels recovers blocks that are very homogeneous horizontally. Due to being homogeneous with itself, the blocks that started with low values of sensed pixels may not be fully representative of the subtle changes in pixel intensities that we can see in both the original blocks and the blocks that started with higher values of sensed pixels.

And because of this, due to not being fully representative of the original block, there are more pixel to pixel differences in the individual intensities, thus leading to a higher Mean Squared Error corresponding to a decreasing number of sensed pixels.

To summarize, the Mean Squared Error has an inverse relationship with the number of sensed pixels.

Chapter 5

Impact of Median Filtering on the Mean Squared Error

Median filtering impacts the recovered image by going through each block in the image in steps of 3 by 3 kernels, and replacing the middle pixel with the median of the sorted values in the neighborhood, hence the name median filtering. Because of this, it necessarily changes the values of intensities of pixel values, which impacts the Mean Squared Error compared to the original image's pixel intensities.

But does median filtering always improve the results, i.e. result in a lower Mean Squared Error? For our particular model, we can see by the graphs in Figures (110 – 117) that in both the Fishing Boat and the Nature images, we see that median filtering only results in a lower Mean Squared Error for lower values of sensed pixels.

Why is it then, that median filtering doesn't objectively improve quality across the board? I believe that the reason for this is because the replacement of each pixel with the median of its neighborhood changes pixel intensities from the original sampled intensities, in other words, the correct intensities with a MSE value of zero when compared to the original image, to the said median. Thus, median filtering actually increases Mean Squared Error for higher values of sensed pixels due to the higher number of "correct pixels".

We concluded previously that median filtering starts overtaking our recovered image without filtering at a sensed pixel value of around 25 sensed pixels.

Chapter 6

References & Citations

1. "Design Photography Logo Camera PNG, Transparent PNG , Transparent PNG Image - Pngitem." *PNGitem.com*, https://www.pngitem.com/middle/TiJwRTo_design-photography-logo-camera-png-transparent-png/.

2. "Compressed Sensing." *Wikipedia*, Wikimedia Foundation, 7 Dec. 2022, https://en.wikipedia.org/wiki/Compressed_sensing.

3. Brunton, Steven Lee, and José Nathan Kutz. "3.1 Sparsity and Compression." *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems and Control*, Cambridge University Press, Cambridge, 2021, pp. 97–97.

4. "Sparse Approximation." *Wikipedia*, Wikimedia Foundation, 5 Mar. 2023, https://en.wikipedia.org/wiki/Sparse_approximation.

5. Brunton, Steven Lee, and José Nathan Kutz. "3.2 Compressed Sensing." *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems and Control*, Cambridge University Press, Cambridge, 2021, pp. 101–103.

6. Lindholm, Andreas. "5.3 Regularisation." *Machine Learning: A First Course for Engineers and Scientists*, Cambridge University Press, Cambridge, United Kingdom, 2022, pp. 109–111.

7. Mattmann, Chris A., and Scott Penberthy. *Machine Learning with Tensorflow.* Manning, 2020.

8. Chen, Yuansi, et al. "A Look at Robustness and Stability of $\ell_{1}$-versus $\ell_{0}$-Regularization: Discussion of Papers by Bertsimas Et Al. and Hastie Et Al.." *Statistical Science*, vol. 35, no. 4, 2020, https://doi.org/10.1214/20-sts809.

9. "Discrete Cosine Transform." *Wikipedia*, Wikimedia Foundation, 26 Jan. 2023, https://en.wikipedia.org/wiki/Discrete_cosine_transform.

10. "Random - Generate Pseudo-Random Numbers." *Python Documentation*, https://docs.python.org/3/library/random.html.

11. "Sklearn.linear_model.Lasso." *Scikit*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html.

12. "Numpy.logspace#." *Numpy.logspace - NumPy v1.24 Manual*, https://numpy.org/doc/stable/reference/generated/numpy.logspace.html.

13. "Scipy.signal.medfilt2d#." *Scipy.signal.medfilt2d - SciPy v1.10.1 Manual*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.medfilt2d.html.

Chapter 6

References & Citations

14. Binz, Kevin. “Intro to Regularization.” *Fewer Lacunae*, 10 June 2019, https://kevinbinz.com/2019/06/09/regularization/.

15. Meng, Fanyu. “4.: Geometrical Interpretation of the ℓ0-, ℓ1-, ℓ2-Norm Problems.” *Geometric Interpretation of the L0 L1 L2 Norm Problems*, Research Gate, https://www.researchgate.net/figure/Geometrical-interpretation-of-the-l0-l1-l2-norm-problems_fig5_326295400.

16. Flores, Tony. “Median Filtering with Python and Opencv.” *Median Filtering with Python and OpenCV*, Medium, 29 Mar. 2019, https://medium.com/@florestony5454/median-filtering-with-python-and-opencv-2bce390be0d1.

Collaborations

The main source of collaboration in my project was through the Peer Feedback sessions. Primarily, the MP#1 Feedback session where we had an open floor to discuss approaches and results were particularly helpful for me in elucidating how to go about dealing with the “constant” basis vector. Specifically, my group talked about sklearn’s fit_intercept parameter for the LASSO function and the pros vs cons of using it. Originally, I wrote code that manually separated the constant basis vector and used that new basis vector matrix in the calculations for the DCT coefficients. I found it a lot easier and concise to simply use sklearn’s fit_intercept parameter in the LASSO function due to not having to worry about and trace bugs in whether or not I was getting my axis directions correct, or if my NumPy reshapes were working correctly, etc. Overall, this collaboration saved me a lot of computation time.

Another source of collaboration I leaned **heavily** on was through the Ed Forum. It was extremely useful to see fellow students retracing their steps in the same errors and bugs I was dealing with, and Dr. Tantum’s elucidating comments on the rubric or instructions cleared up many of the instructions which I misinterpreted originally such as the Log 10 image visualization of the regularization parameter as a function of block number.

Outside of this, all ideas, code, and work in this project was done without help. I used a number of different resources as cited in my references, and did everything using those.

Compressed Sensing
Image Recovery

Elliot Ha // 03.06.23

Thank You!