COMPRESSED SENSING IMAGE RECOVERY

ECE 580 Mini Project #1

3/4/2022

Tyler Feldman

Table of Contents

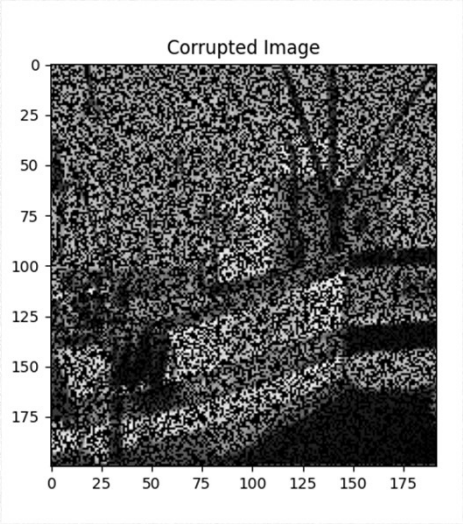
Introduction

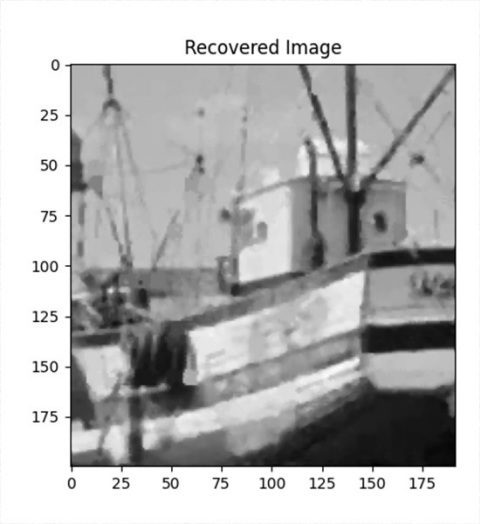
Compressed Sensing Image Recovery

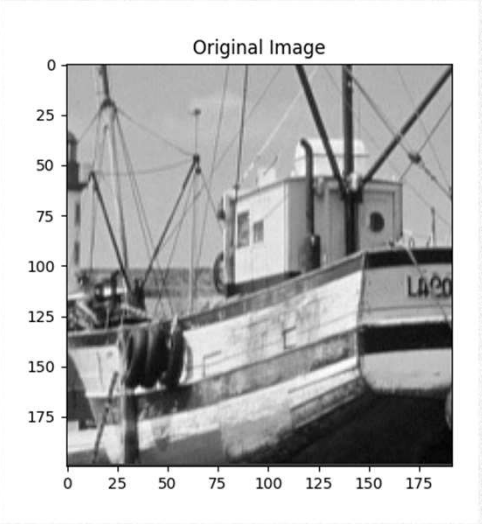
Our goal for this project is to use an algorithm to recover an image that has been corrupted. Given an image that has the value missing for certain pixels, we'd like to estimate the value of those pixels to recover the image to what it was before it was corrupted.

We'd like to develop a method that works effectively at this task and then explore how effective this method is at varying levels of image corruption. We'd also like to explore how certain steps and parameters in the algorithm affect the final output and how those steps interplay with one another so that we can develop a better understanding of this problem and how our chosen solution works.

Below you can see an illustration of our goal. Given the corrupted image on the left, we'd like to find a recovered image (middle) so that it is as close as possible to the original image seen on the right.

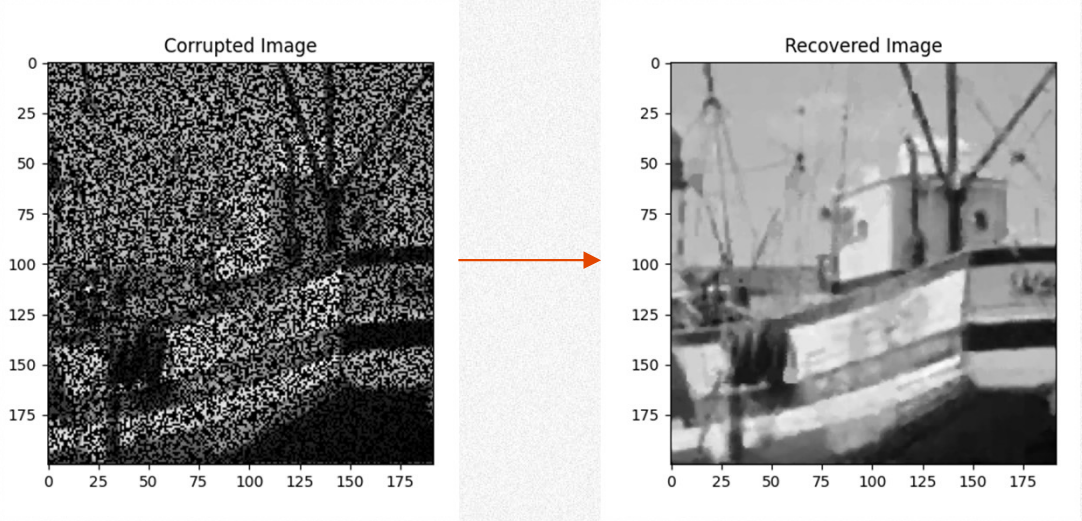






Summary of Key Results

Using a method involving discrete cosine transforms and lasso regression, I **was able to effectively recover the images**.



As the amount of corruption increased, this method performed increasingly worse. However, there were many parameters additional options that could mitigate the final error for high amounts of corruption. For example, I found that **using a filter after recovering the image significantly improved the quality of the recovered image when there was high amounts of corruption**.

I explored a variety of parameters and filters for this method to gain an intuitive understanding for how to certain parameters or steps affected the final output. After analyzing issues with this method, I was able to think of possible solutions to these problems that could significantly improve the performance of our image recovery method.

Mathematical Formulation

Mathematical Formulation

Below are the steps taken to take an image, corrupt it, and then use and evaluate our image recovery algorithm. Each step is explained in detail in the following slides.

Split up the image into square blocks of pixels and for each block:

1. Sample S pixels from this square block and remove the rest as testing pixels
2. Calculate the basis functions for the 2D discrete cosine transform of the block of pixels
3. Solve an under-determined linear system for the discrete cosine transform coefficients using LASSO, which is least squares minimization with L1-norm regularization. The regularization parameter for LASSO is found by exploring a range of possible values, evaluating each using cross validation, and then picking the best performing value
4. Use the discrete cosine transform coefficients to estimate the values for the missing pixels

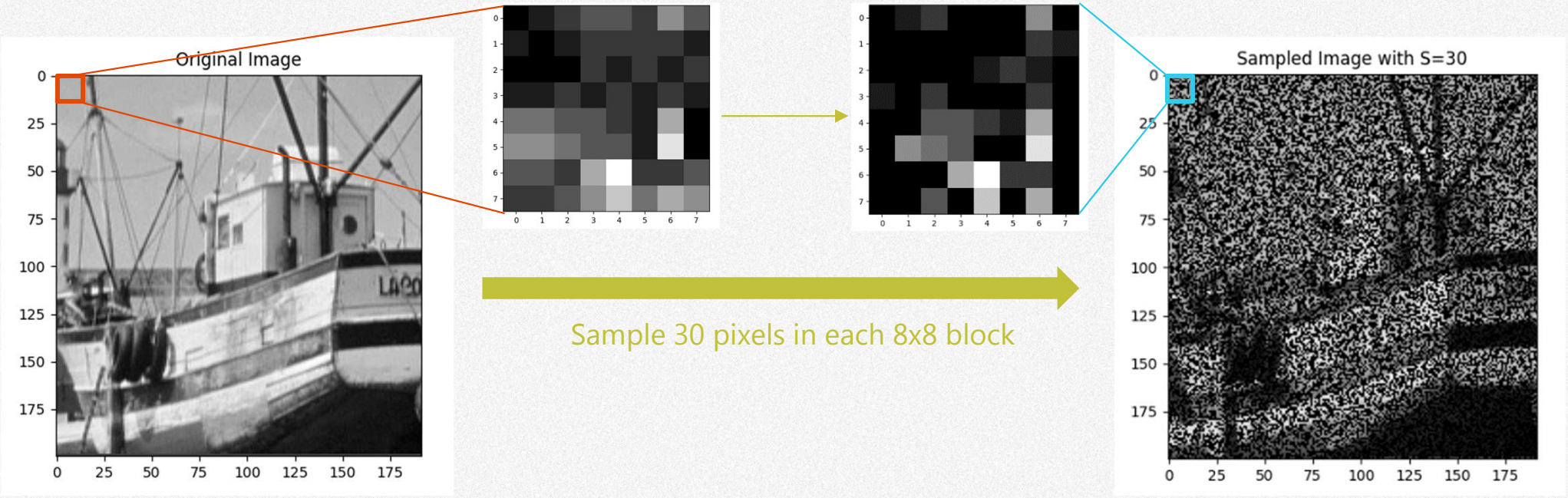
Stitch all the blocks together to get the recovered image
Perform median filtering on the recovered image
Calculate error between recovered image and original image using mean squared error

Corrupting the Image so That We Can Recover It

To simulate the act of recovering a corrupted image, we take a normal image, and randomly sample a number of pixels out of to keep and will discard the other pixels. These discarded pixels are what we want to estimate using the sampled pixels that remain.

In practice, we split the image into square blocks of pixels and sample a number of pixels (which we will call S) from that block and discard the rest. The reason for this will be explained on slide 11.

In the example below, the image was split into square blocks 8 pixels in each dimension, and S=30 pixels were sampled from each block. This means each of these blocks has 64 pixels, and 30 of them are kept, and the 34 remaining are discarded. This sampling process is done for each block to get the result on the right.



Discrete Cosine Transform (DCT)

The 2-dimensional discrete cosine transform (DCT) models the pixels in an image in terms of a sum of cosine functions oscillating at different frequencies. In other words, it transforms the image from the spatial domain to the frequency domain. This provides the spatial frequency content in both the vertical and horizontal directions[1].

Each pixel value is found by summing the basis function over all possible spatial frequency pairs, where each basis function is weighted by the discrete cosine transform coefficient for that spatial frequency pair. The equation below shows that the pixel value at x, y is equal to the sum of basis functions for each spatial frequency pair u, v, weighted by the DCT coefficient G(u, v)[2].

$$g(x,y)=\sum_{u=1}^{P}\sum_{v=1}^{Q}\alpha_u\cdot\beta_v\cdot\cos\frac{\pi(2x-1)(u-1)}{2\cdot P}\cdot\cos\frac{\pi(2y-1)(v-1)}{2\cdot Q}\cdot G(u,v)$$

Figure from [2]

This formula can be constructed as a matrix multiplication between the basis functions T and a column vector of DCT coefficients, gamma. This linear combination of basis functions then produces the pixel values in the image, here packaged as a flattened column vector C.

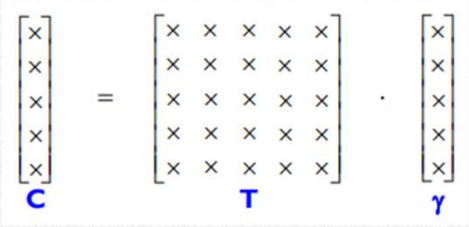


Figure from [2]

1. Marshall, Dave. "The Discrete Cosine Transform (DCT)." *Cardiff School of Computer Science & Informatics*, https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html.
2. Tantum, Stacy. "Introduction to Machine Learning: Mini-Project 1 Compressed Sensing Image Recovery" Introduction to Machine Learning. Spring 2022. Accessed 3 March 2022.

Under-Determined Linear System to Find DCT Coefficients

After performing the discrete cosine transform, we have the different basis functions that represent the various spatial frequency pairs. However, we still don't know how much each of these basis functions are weighted, or in other words, how much each spatial frequency pair is present in the image. These values are called the discrete cosine transform coefficients. These coefficients and the basis functions form a linear system[2]:

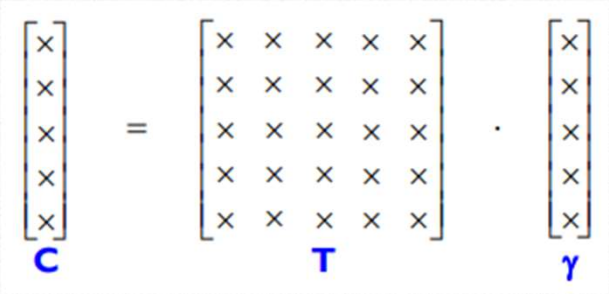
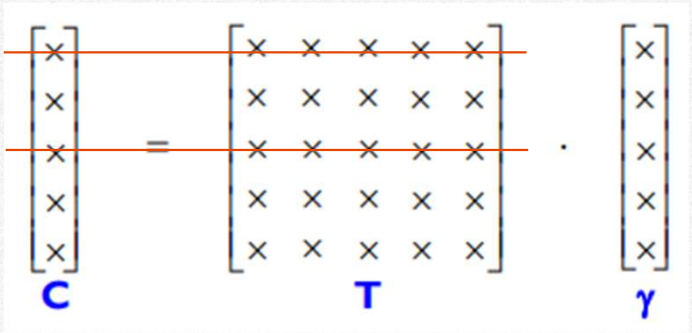


Figure from [2]

However, after removing pixels from the image, we are left with a linear system that has more unknowns than equations. This system has a nontrivial null space, so there will be an infinite number of solutions[3]. To solve this system, we must introduce additional constraints.



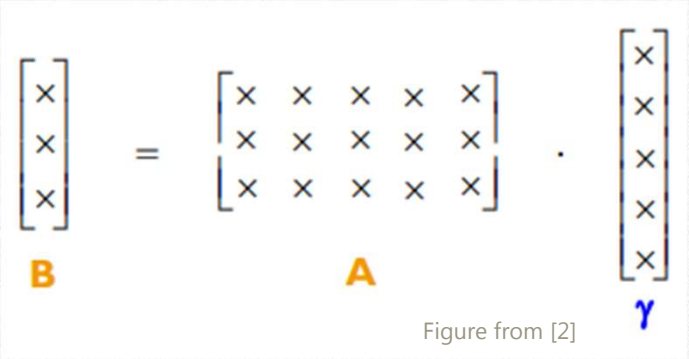


Figure from [2]

2. Tantum, Stacy. "Introduction to Machine Learning: Mini-Project 1 Compressed Sensing Image Recovery" Introduction to Machine Learning. Spring 2022. Accessed 3 March 2022.
3. "Chapter 8: Least Squares Solutions to Linear Systems." *Numerical Linear Algebra and Applications*, by Biswa Nath Datta, 2nd ed., Society for Industrial and Applied Mathematics, Philadelphia, 2010.

LASSO Regression

The constraint we chose is sparsity. This is because natural images tend to be sparse . In practice, however, these DCT coefficients are not sparse for a large image but are sparse locally. For this reason, we **split the image into square blocks and do the recovery for each block separately**.

To solve the under-determined linear system, we choose LASSO regression. This is because LASSO regression will impose a constraint of sparsity. LASSO regression minimizes the residual sum of squares with the constraint that the absolute value of the coefficients is less than a certain constant[4].

$$(\hat{\alpha}, \hat{\beta}) = \arg\min\left\{ \sum_{i=1}^{N} \left(y_i - \alpha - \sum_j \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \sum_j |\beta_j| \leqslant \lambda$$

Figure from [4]

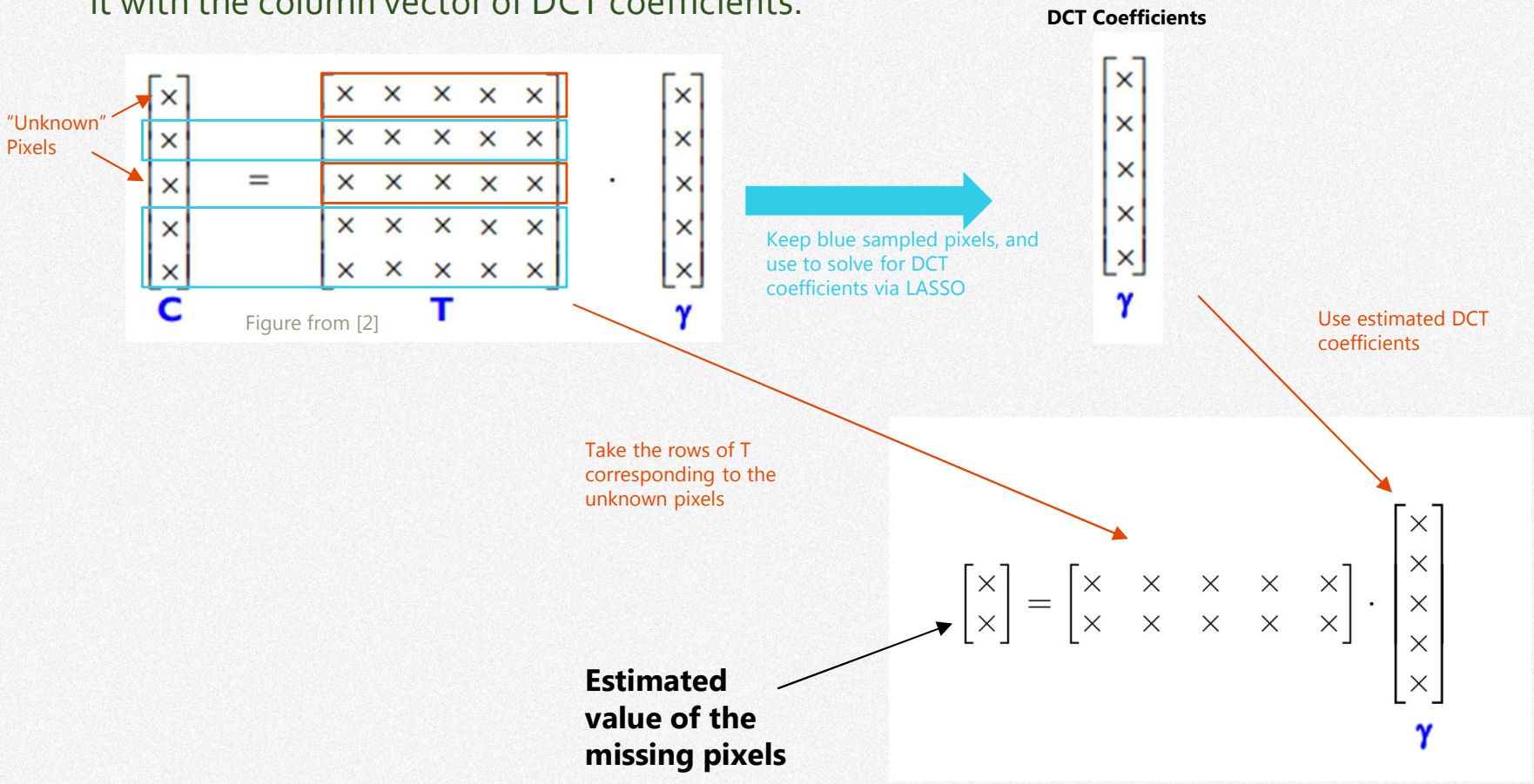
This constraint on the right will tend to produce some coefficients that are zero, which will result in sparsity in the estimated DCT coefficients. The larger lambda is, the stronger the constraint is. This value lambda is parameter that we can control, and for each block, we'll end up searching through a range of possible lambda values and select the best one. This mathematical formulation above is equivalent to the scikit learn implementation, which minimizes the following equation, where n = number of samples[5].

$$\frac{1}{2n} \|y - Xw\|_2^2 + \lambda \|w\|_1$$

4. Tibshirani, Robert. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, [Royal Statistical Society, Wiley], 1996, pp. 267–88, http://www.jstor.org/stable/2346178.
5. "Sklearn.linear_model.Lasso." *Scikit Learn*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html. Accessed 3 March 2022.

Estimating the Missing Pixels

After estimating the DCT coefficients using LASSO regression, we can use the linear system to recover the missing pixels. This involves taking the row of T that corresponds to the missing pixel and multiplying it with the column vector of DCT coefficients.



Evaluating The Estimated Pixels

After estimating the missing pixels, we can then **compare the estimated missing pixel to the actual value of the pixel from the original image**. We can do this using **mean squared error (MSE)**.

For a set of estimated pixels, y hat, and their true values, y, we can calculate the MSE by **summing the differences between each corresponding element in the two vectors, and then dividing by the number of elements in y**[6].

The equations below illustrate how the error epsilon is the element-wise difference between the true and predicted pixel values. The MSE is the sum of all these values squared, and then divided by how many values there are.

$$\epsilon = \hat{\mathbf{y}} - \mathbf{y} \qquad \epsilon_i = \hat{\mathbf{y}}_\mathbf{i} - \mathbf{y}_\mathbf{i} \qquad \mathrm{MSE}(\epsilon) = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i^2$$

6. Hodson, Timothy O., et al. "Mean Squared Error, Deconstructed." *Journal of Advances in Modeling Earth Systems*, vol. 13, no. 12, 23 Nov. 2021, doi:10.1029/2021ms002681.

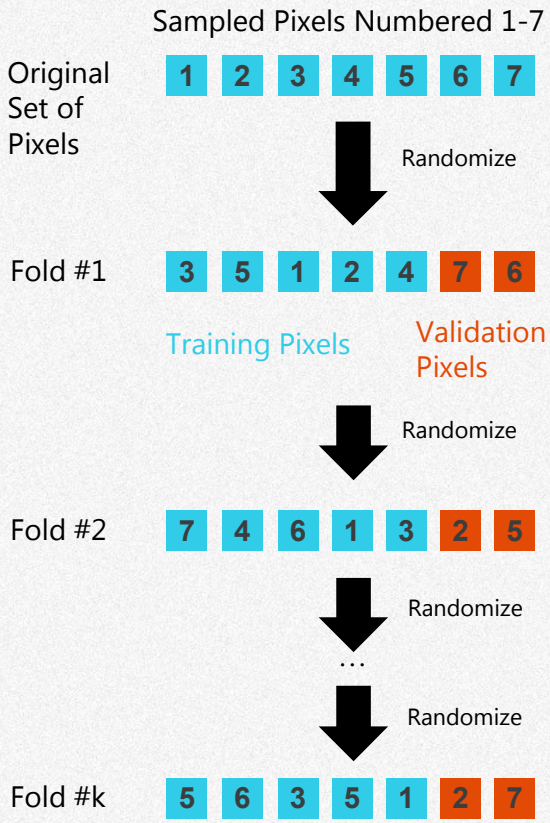
Cross Validation for Pixel Estimation

We have seen the process for estimating the missing pixels for a block of pixels. However, we'd like to **explore how the regularization parameter, lambda, in LASSO regression impacts the accuracy of the recovered blocks**. Since we're estimating the DCT coefficients for each block separately, we can have different values of lambda used to recover each block.

To determine the value of lambda for a given block, we can use cross validation. We first define a list of lambda values we'd like to try. The list I used was 30 different values spaced out evenly in log space from 10e-7 to 10e5 since that provided a large range of values to explore.

Cross-validation involves repeatedly selecting training and validation sets with the data points. The training set is used to estimate the data points in the validation set, and then we can find the MSE of these estimated validation pixels. Here, we use **k-fold random subsampling**, where we randomly select the training and validation k times.

Selecting multiple random validation sets allow us to observe the generalizability of each model[7], which is how well a particular model / value of lambda works over a variety of datasets.



7. Daniel Berrar, Cross-Validation, Editor(s): Shoba Ranganathan, Michael Gribskov, Kenta Nakai, Christian Schönbach, Encyclopedia of Bioinformatics and Computational Biology, Academic Press, 2019, Pages 542-545, ISBN 9780128114322, https://doi.org/10.1016/B978-0-12-809633-8.20349-X. (https://www.sciencedirect.com/science/article/pii/B978012809633820349X)

Cross Validation for Pixel Estimation

This process for each block of pixels is as follows[7]

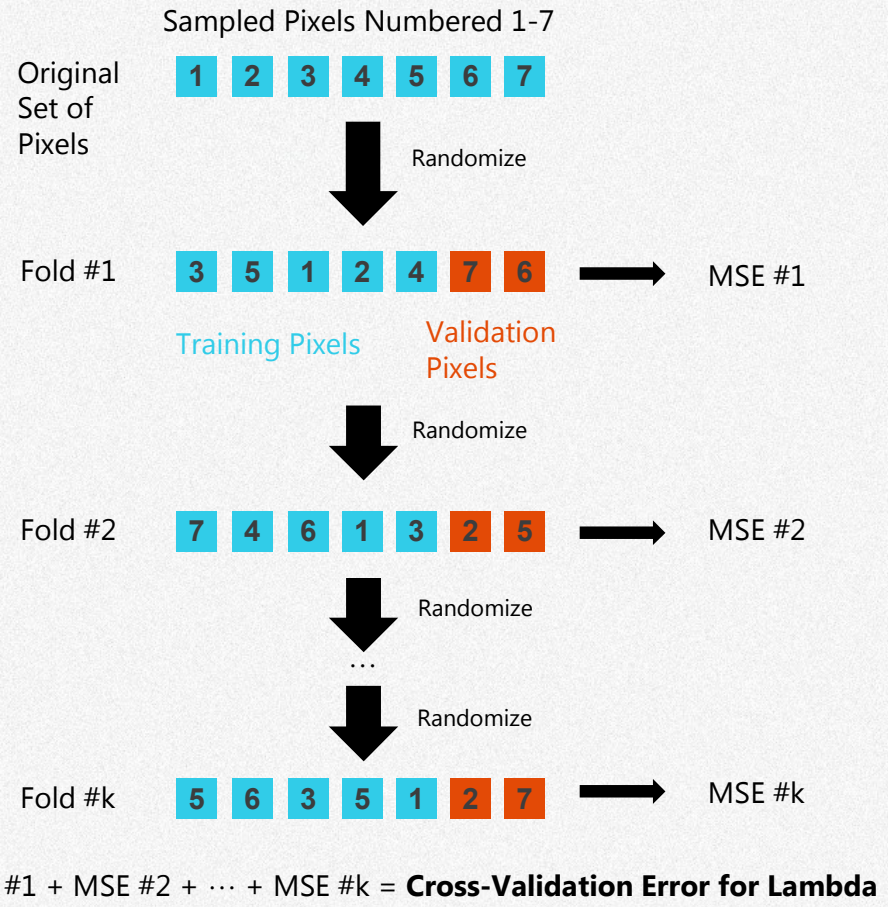
For each value of lambda:

Repeat k times:

1. Split the sampled pixels into training and validation set randomly
2. Estimate the DCT coefficients using the training pixels
3. Recover the validation pixels
4. Calculate the mean squared error (MSE)

Average the MSE over all k fold to get the cross-validation error for the current value of lambda

Finally, select the value of lambda with the lowest cross-validation error

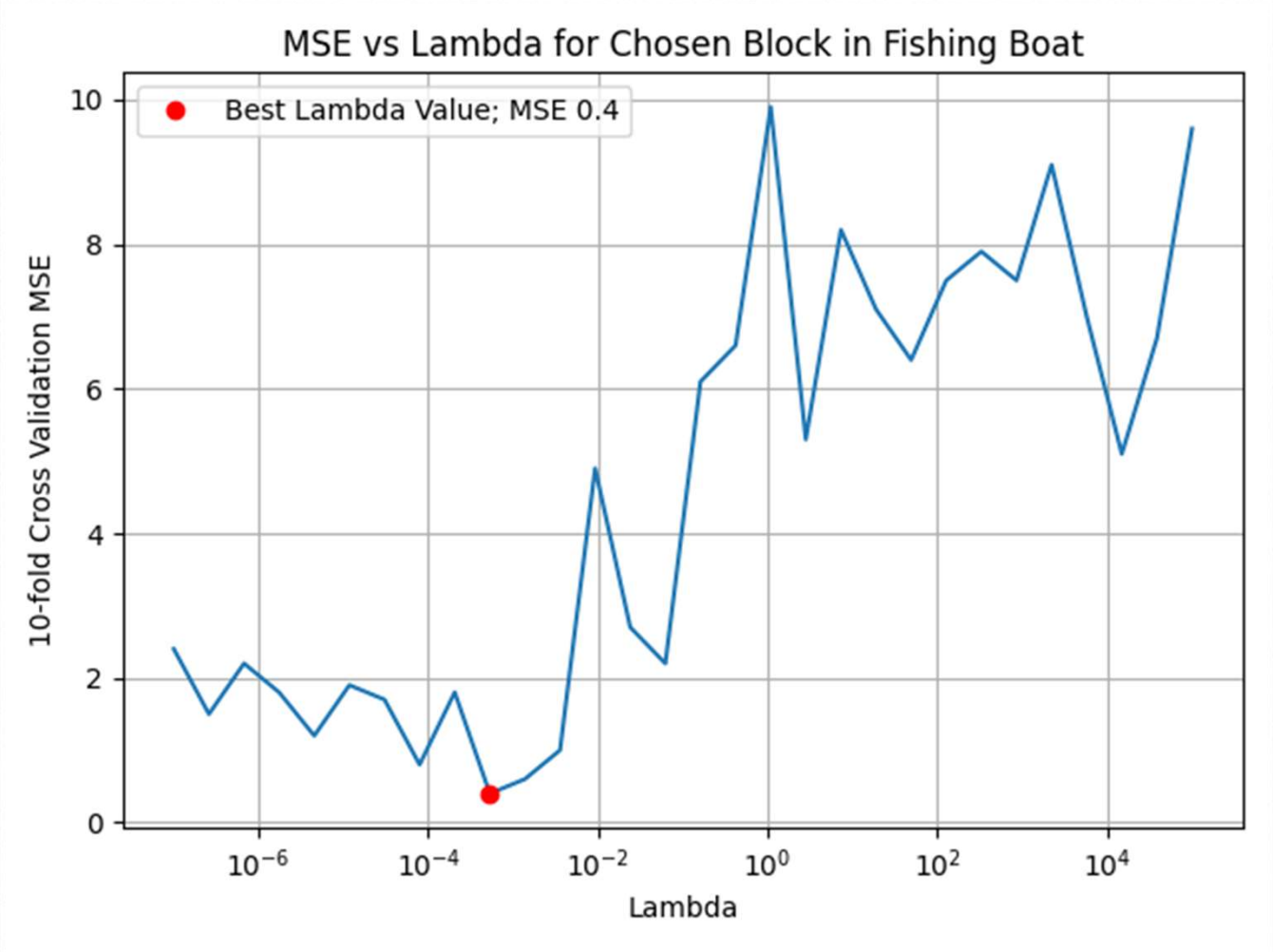


MSE #1 + MSE #2 + ⋯ + MSE #k = **Cross-Validation Error for Lambda**

7. Daniel Berrar, Cross-Validation, Editor(s): Shoba Ranganathan, Michael Gribskov, Kenta Nakai, Christian Schönbach, Encyclopedia of Bioinformatics and Computational Biology, Academic Press, 2019, Pages 542-545, ISBN 9780128114322, https://doi.org/10.1016/B978-0-12-809633-8.20349-X. (https://www.sciencedirect.com/science/article/pii/B978012809633820349X)

Cross Validation for Pixel Estimation

For each value of lambda, we perform cross-validation to find the error associated with that value of lambda. Plotting all these errors for each lambda produces the plot below. The red dot marks the location of the best performing value of lambda, which is the lambda value that would be selected for this block.



Median Filtering

Once we have determined an optimal lambda value for each block, we then use LASSO to find the DCT coefficients. We can use these DCT coefficients to then recover the missing pixels as seen before.

Because we split up the image into blocks before computing the DCT and coefficients, there might be errors with the estimated pixels at the boundaries of these blocks. Thus, it makes sense to apply a filter to the resulting image to smoothen out the inconsistencies between each block.

The median filter operates by taking a 2D window of pixel values, sorting them in increasing order, and then taking the mid value in this sorted sequence[8]. The median filter provides strong resistance to outlier noise / salt and paper noise[9], which will mitigate the impacts of the block-by-block approach we have taken.

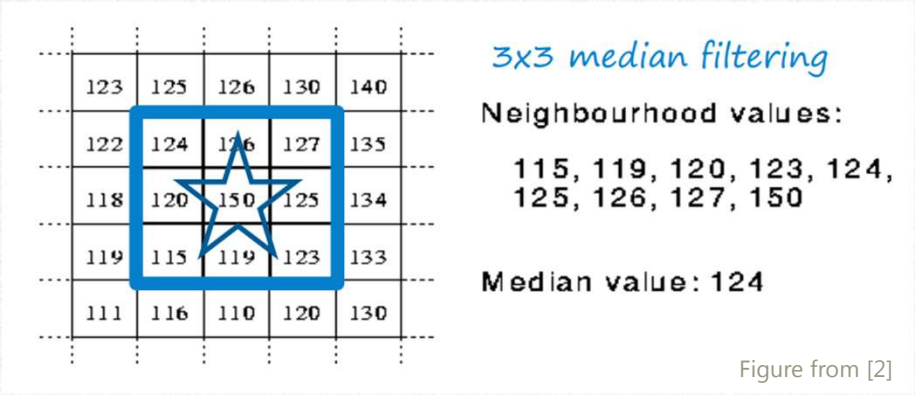


Figure from [2]

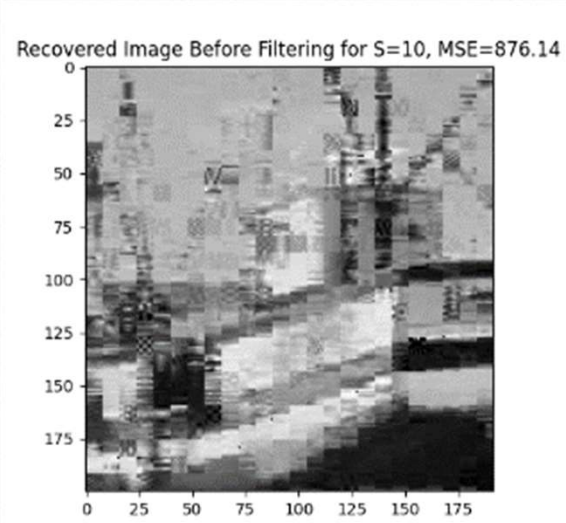
2. Tantum, Stacy. "Introduction to Machine Learning: Mini-Project 1 Compressed Sensing Image Recovery" Introduction to Machine Learning. Spring 2022. Accessed 3 March 2022.
8. G. George, R. M. Oommen, S. Shelly, S. S. Philipose and A. M. Varghese, "A Survey on Various Median Filtering Techniques For Removal of Impulse Noise From Digital Image," 2018 Conference on Emerging Devices and Smart Systems (ICEDSS), 2018, pp. 235-238, doi: 10.1109/ICEDSS.2018.8544273.
9. Villar, Sebastián A., et al. "Median Filtering: A New Insight." *Journal of Mathematical Imaging and Vision*, vol. 58, no. 1, 2016, pp. 130–146., doi:10.1007/s10851-016-0694-0.

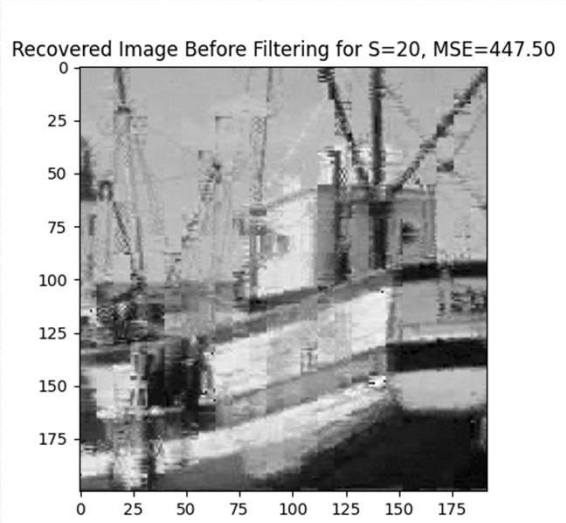
Experimental Results

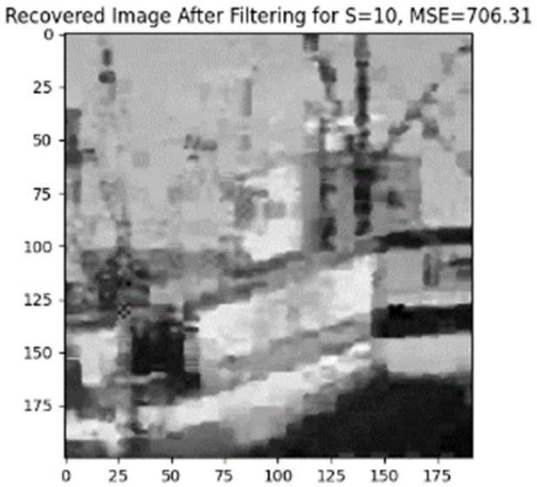
Fishing Boat Results

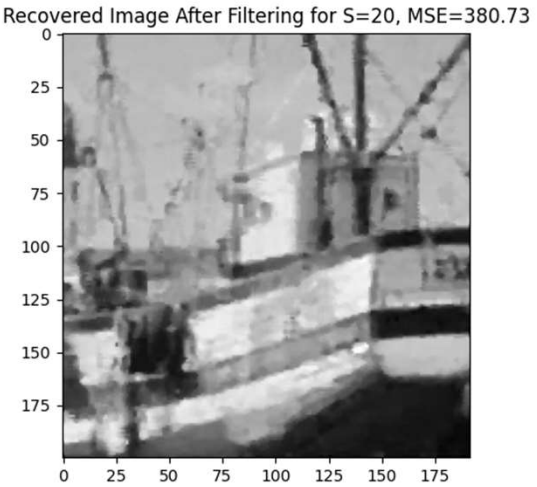
We can observe the results when sampling at different rates. For this image, I've picked values of S from 10 to 60 in steps of 5. The results for S=10 and 20 are shown below. We can see that as S increases, the recovered image better resembles the original image. This is reflected quantitatively in the mean squared error. The MSE value for S=20 is significantly lower than the MSE value for S=10, specifically by a factor of 1.95.

We can also observe that the median filtering improves the performance of the image recovery, since for both S=10 and S=20, the MSE after filtering is lower than the MSE before filtering. We can also see that the MSE improvement of median filtering for S=10 is much larger than that for S=20 .



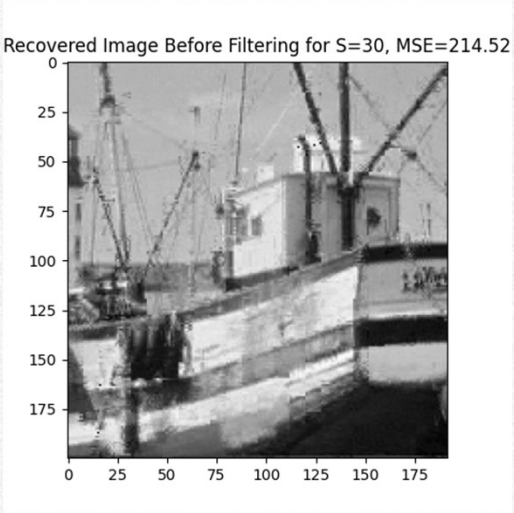


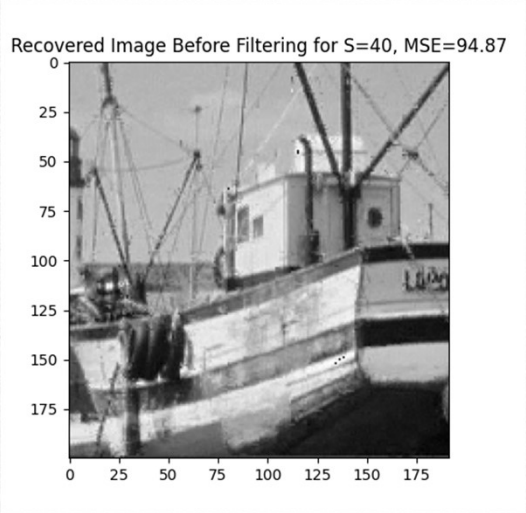


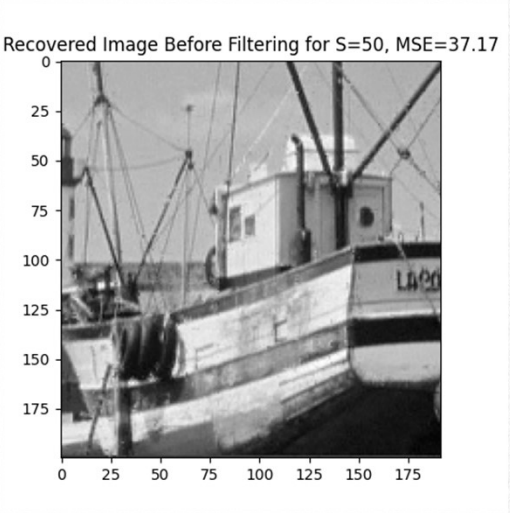


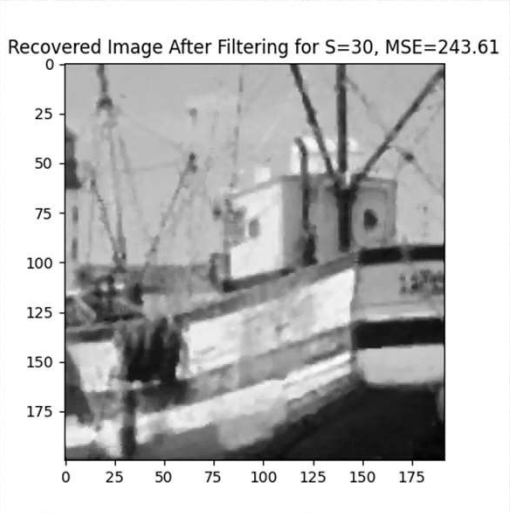
Fishing Boat Results for S = 30, 40, 50

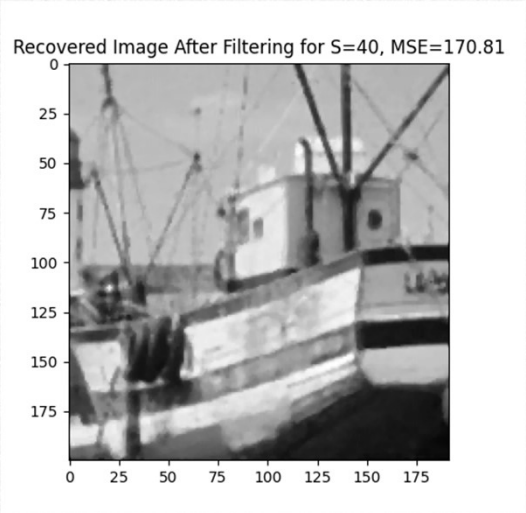
Shown below are the result for S=30, 40, and 50. We can see that as the number of sampled pixels increases, the resulting MSE decreases. We can also observe that now that S is higher than before, the MSE after applying a median filter is consistently larger than the recovered image before filtering.

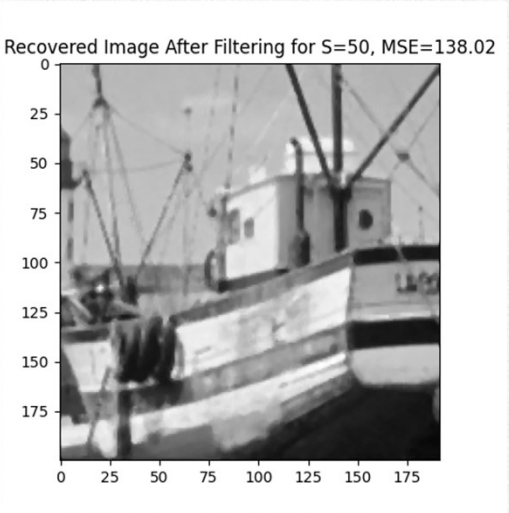






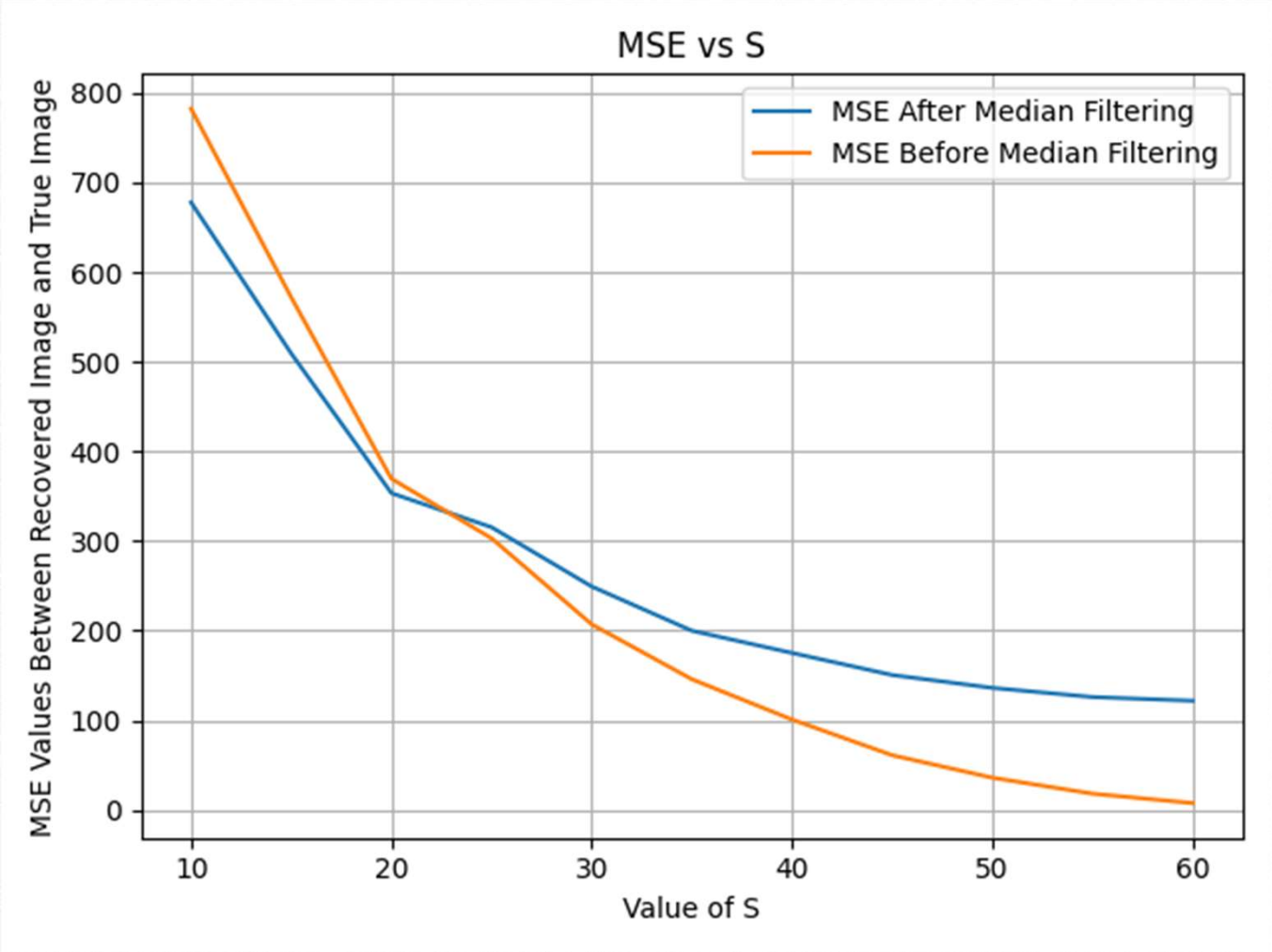






Fishing Boat Results Summary

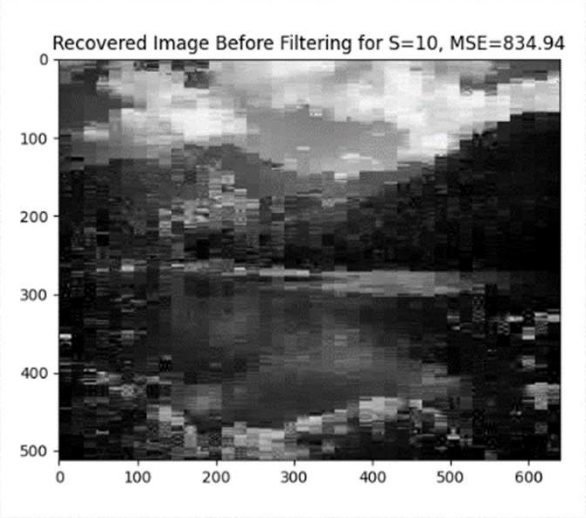
Below is the plot of MSE values before and after median filtering for a range of values of S between 10 and 60 and spaced out by a value of 5. The MSE values seem to decreases exponentially / quadratically as S increase. Once S is equal to or larger than 25, the MSE for the recovered image after filtering is larger than the error for the recovered image before filtering. As S increases past this point, the difference between the MSE before and after filtering increases. This shows us that with a less corrupted image, median filtering hurts performance, and with a more corrupted image, median filtering helps performance.

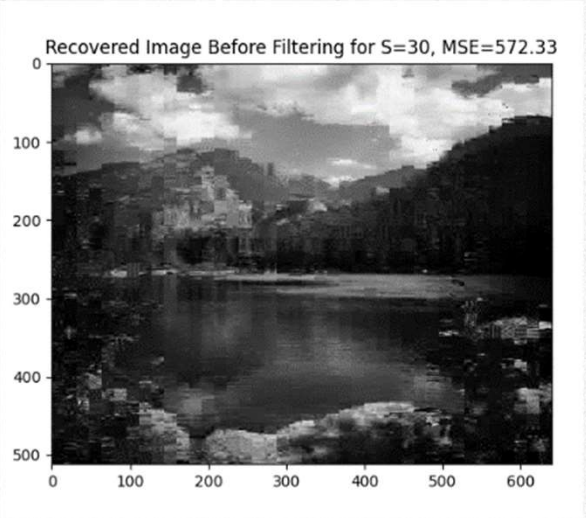


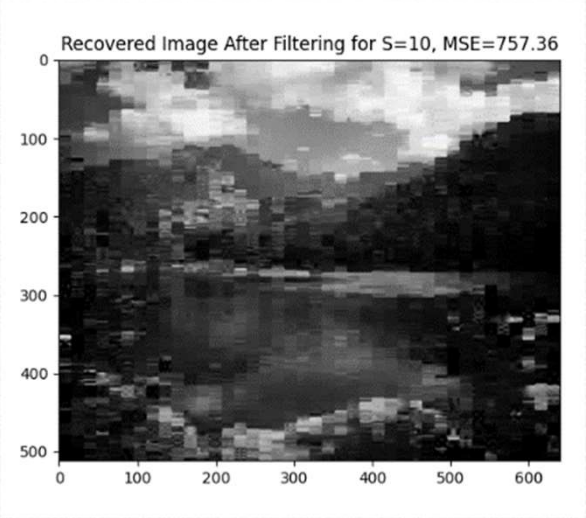
Nature Results

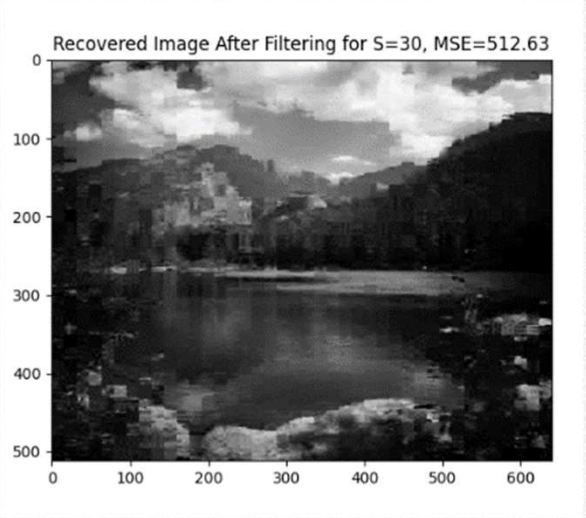
I repeated this experiment for a different image, titled nature. This image is much larger, and so the size of the blocks is increased to 16x16 as opposed to 8x8 for the fishing boat image. Since each block now contains 256 pixels, I chose a range of values of S from 10 to 230 that increased by 20 each time. The results for S=10 and S=30 are shown to the right.

The error after applying the median filtering is significantly lower than the error of the recovered image before applying the median filter. Like for fishing boat, at low values of S, the median filter improves performance.





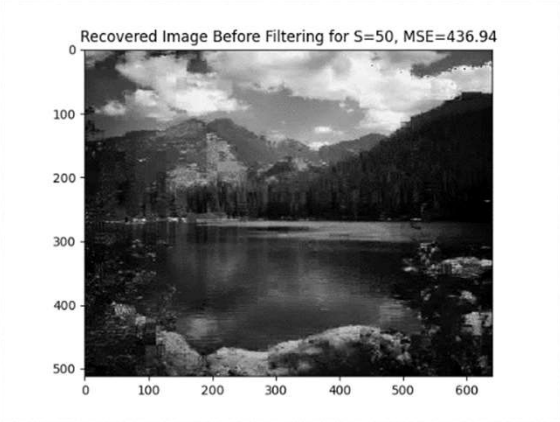


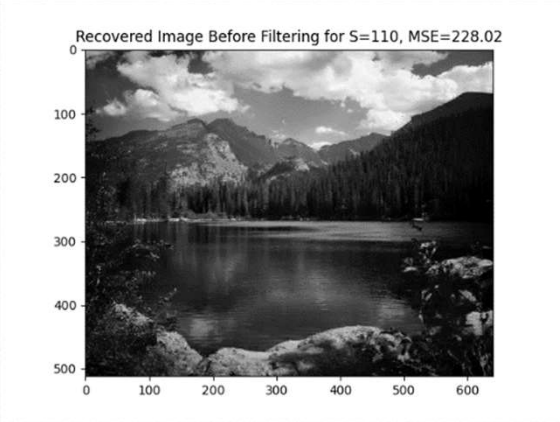


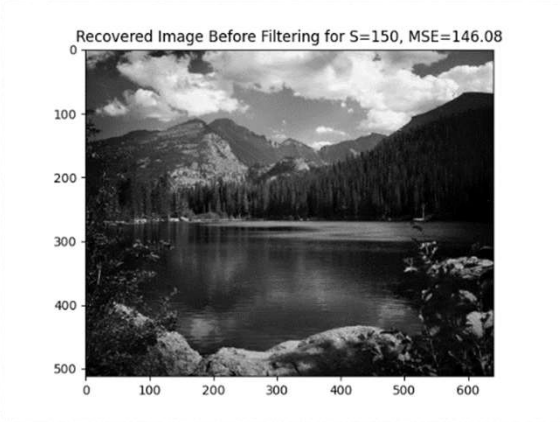
Nature Results

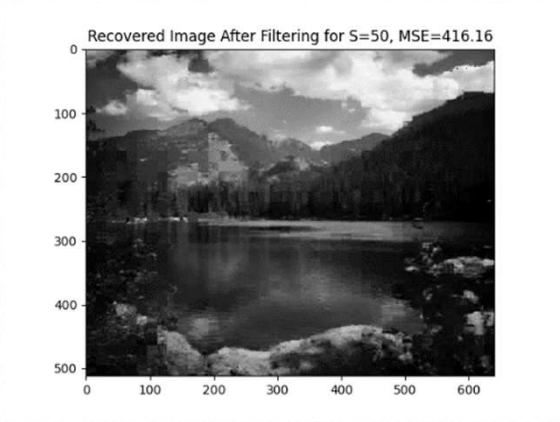
Below are the results for S = 50, 110, and 150. As S increases, the MSE decreases less. From S=50 to S=110, the MSE before filtering changes by 208, but from S=110 to S=150, the MSE only changes by 82.

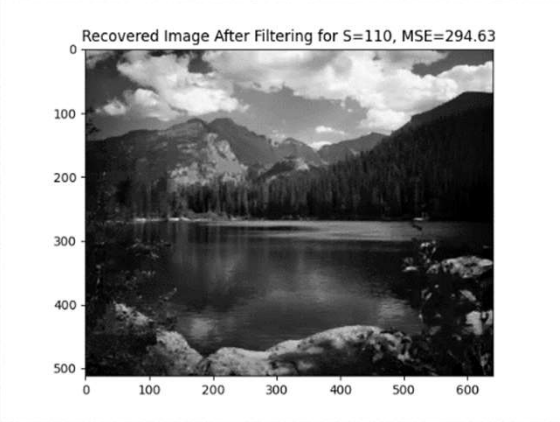
For S=50, the median filter still helps, but for S=110 and S=150, the median filter hurts performance. This shows a similar trend as with the fishing boat, where once the image has a certain proportion of sampled pixels, the filtering starts hurting the quality of the recovered image.

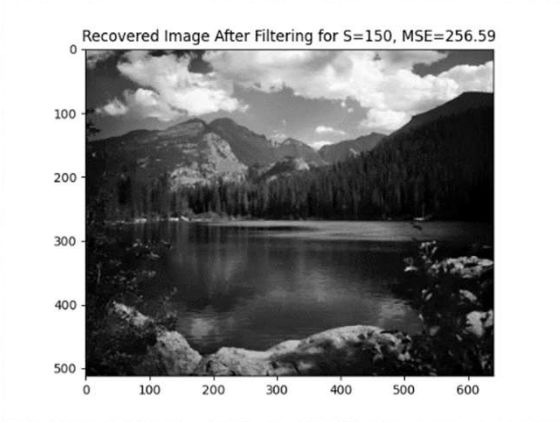






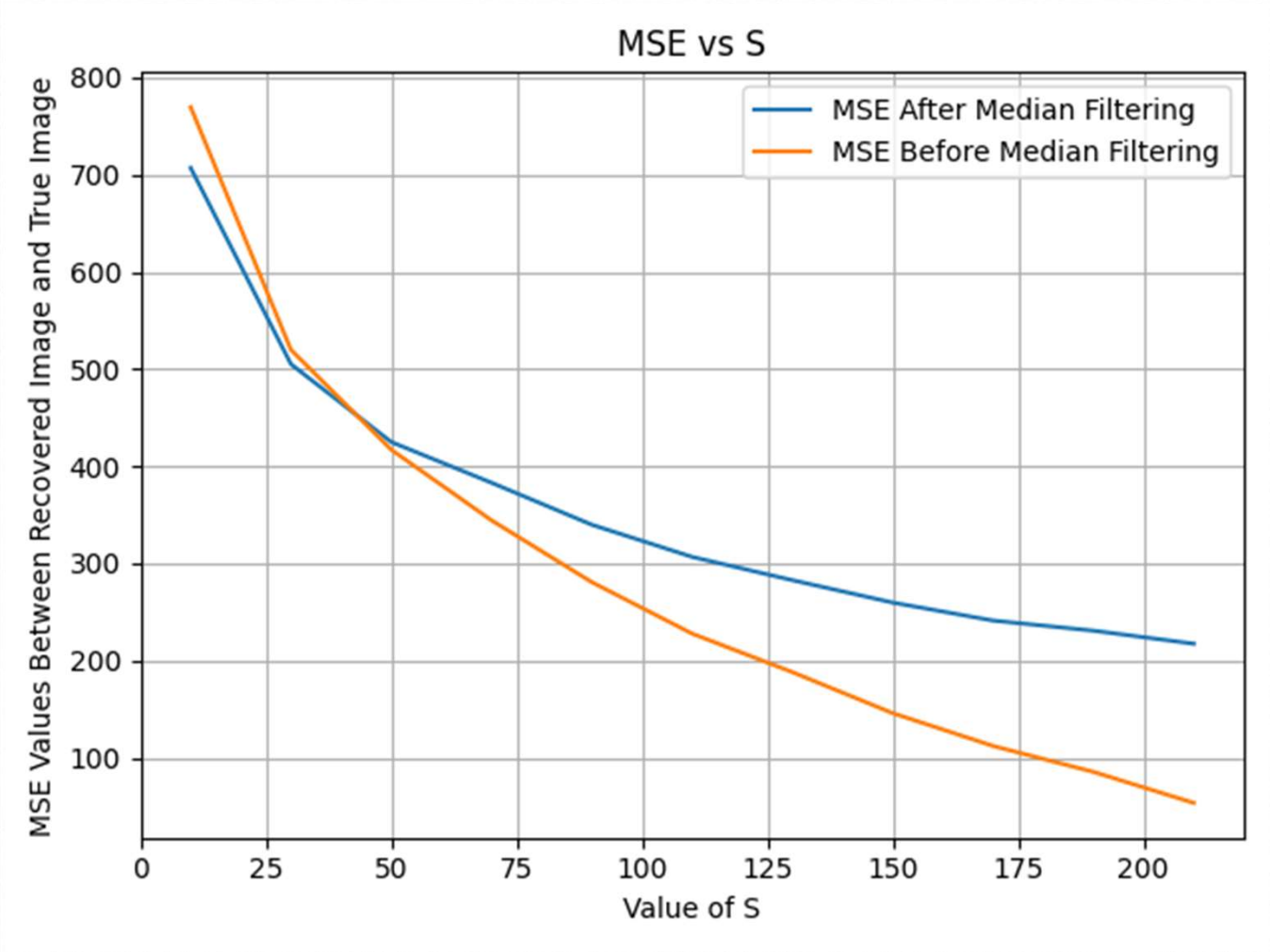






Nature Results Summary

When looking at the resulting MSE as a function of the number of pixels sampled, S, we see a similar trend for this image as for the fishing boat image. The MSE values approximately decrease exponentially / quadratically as S increases. Just like for the fishing boat image, once S reaches a certain value, the median filtering starts hurting performance.

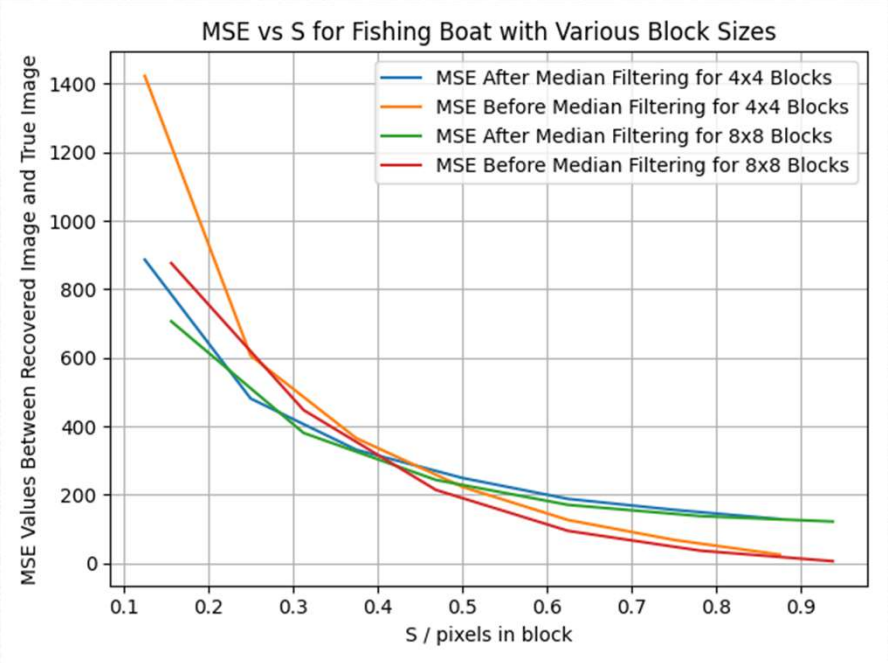


Exploring Block Size: Fishing Boat

To **explore how the block size impacts the quality of the recovered image,** I tried a different block size for each image. Here we'll focus on the fishing boat image. For this, I tried blocks of 4x4 and 8x8.

Since the value of S is the number of pixels sampled for each block, the range of S is different for the two different block sizes. However, the proportion of sampled pixels to corrupted pixels is approximately equal for both, since the 4x4 block has 4 times less pixels than the 8x8 block. **We can see that using 8x8 blocks is more effective,** especially for images with more corruption (on the left side of the graph).

Using a smaller block size will increase the number of pixels at the edge of a block. **If the blocks are smaller, then the irregularities at the edge of the pixel blocks will become worse** and increase the error of the recovered image.

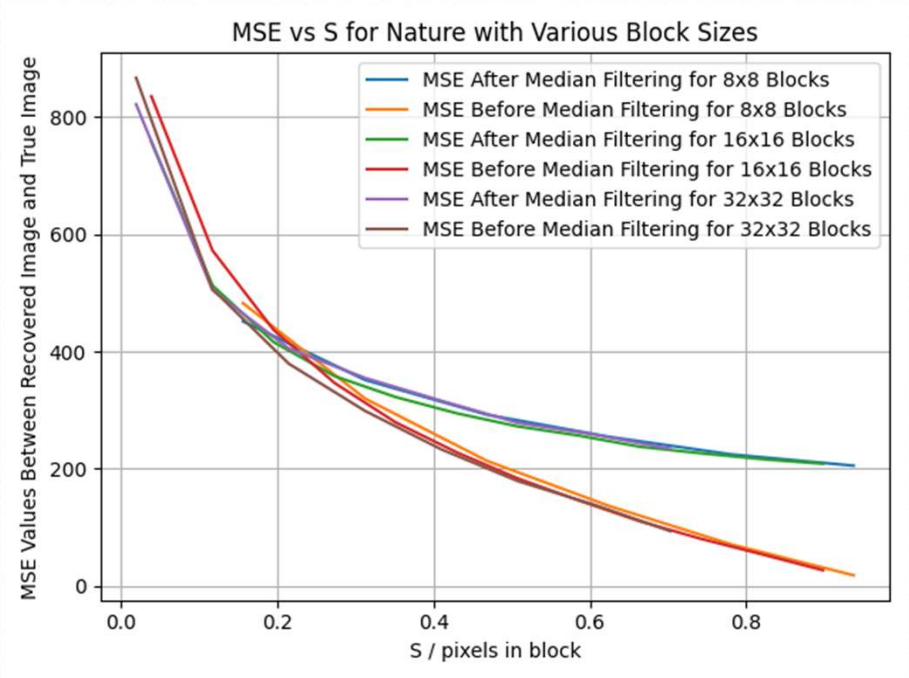


Exploring Block Size: Nature

I also tried this for the nature image. Here, I tried blocks of size 8x8, 16x16 and 32x32.

The issue with using larger block sizes is that our assumption of sparsity in spatial frequency becomes less accurate. With local regions in an image, we can assume that the spatial frequency content is sparse but as that local region increases in size, this assumption starts breaking down.

However, we can see below, that when looking at the ratio of sampled pixels to total pixels, **the block sizes all perform with similar quality**. This shows that with 32x32 blocks, it isn't large enough to cause issues with DCT sparsity. For all three block sizes, we can still see the trend where the median filtering starts hurting performance after a certain value of S.



Exploring Filtering Techniques

I chose to explore various filtering techniques. I chose two alternative filters to the 3x3 median filter being used in the previous examples: a 3x3 Gaussian kernel, and a 3x3 mean filter. This experiment was done for fishing boat and with 8x8 blocks.

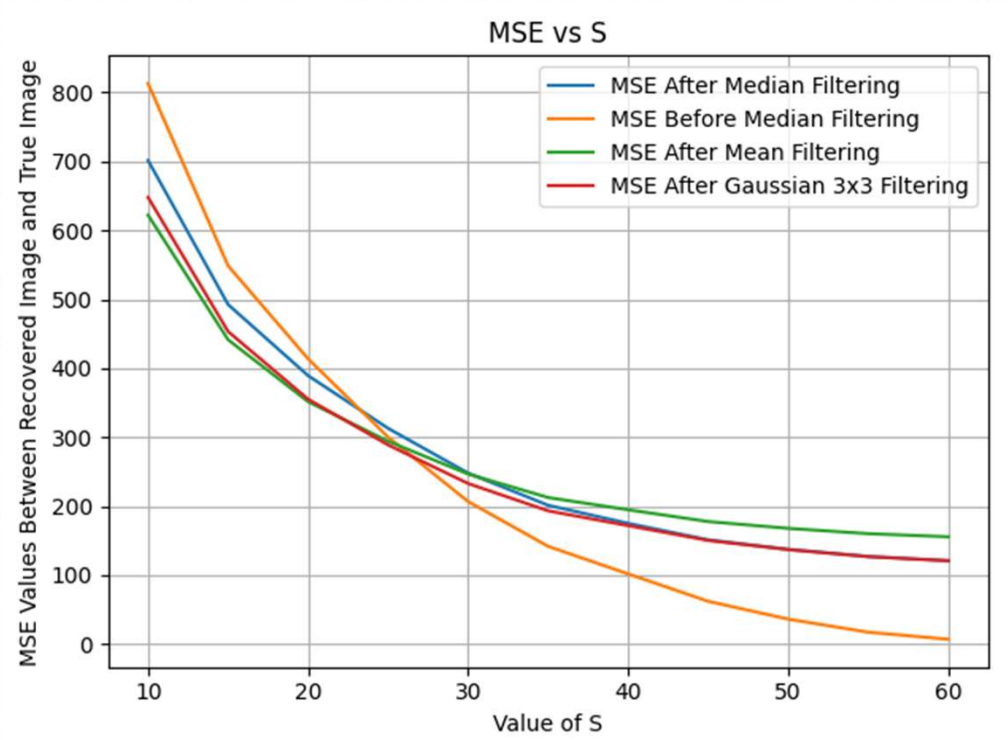
The Gaussian filter smooths the image based on a 2D gaussian. This is equivalent to convolution with the kernel G shown to the right.

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$M = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The mean filter simply took the mean the 3x3 area around each pixel to get the new value for each pixel. This is equivalent to convolution with the kernel M on the right.

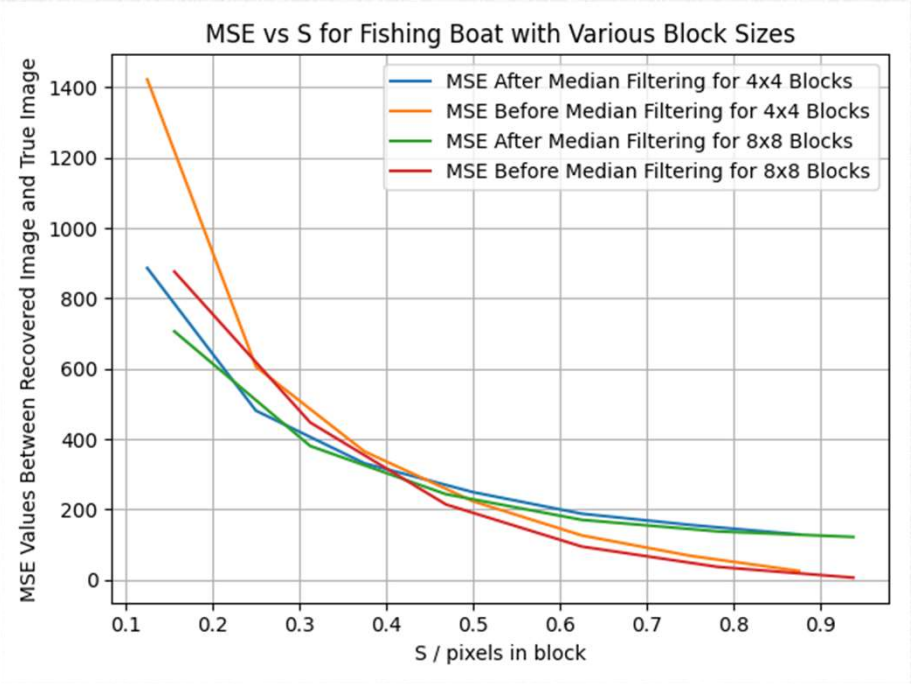
The resulting plot shows the filters all have similar effects and start hurting performance after a certain value of S. Before this point, the **mean filter performs best**. After this point, the median and gaussian filters work best, although still worse than the performance without filtering.

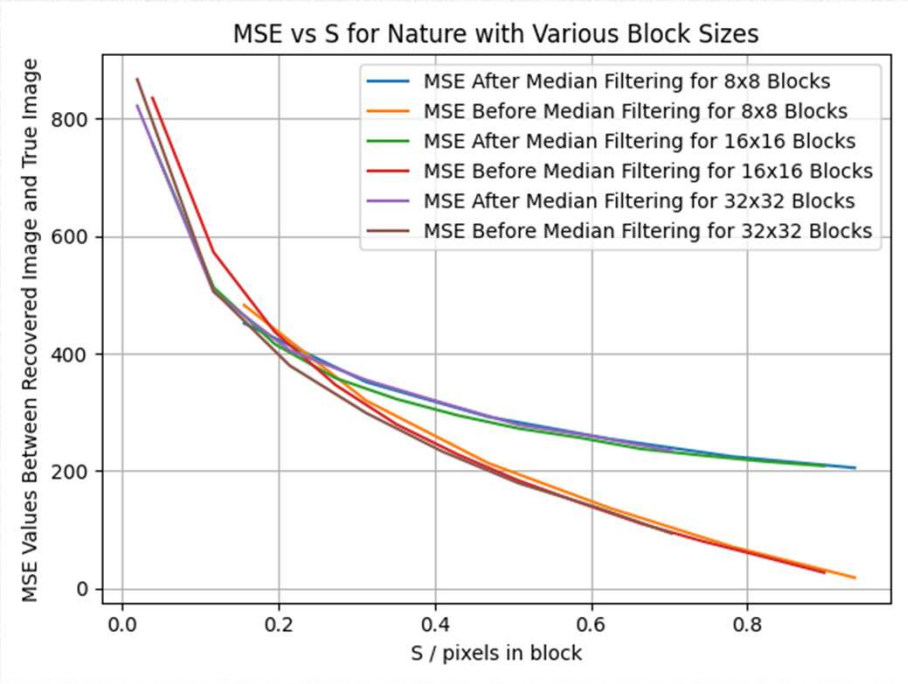


Discussion / Conclusion

Factors That Impact the Quality of the Recovered Image: Level of Corruption of the Image

The more corrupted the image, the worse the result. This was true for all tested block sizes and for both images. The graphs of MSE vs S show that as the image becomes more corrupted, the MSE grows at an increasing rate. This shows either that the problem itself becomes increasingly difficult with more corrupted images or our approach underperforms with more corrupted images. If there aren't many pixels available, this makes cross validation harder, so it becomes harder to estimate the right lambda value. With each pixel we're missing, the solution space for the linear system to find the DCT coefficients gains a new dimension, which will likely make our solution less optimal.





Factors That Impact the Quality of the Recovered Image: The Use of the Median Filter

The median filter was meant to resolve issues when estimating pixel values at the boundary of the pixel blocks. From the experimental results, we can see that after a certain ratio of sampled pixels to total pixels, the median filter worsens the recovered image. This ratio of sampled pixels to total pixels at which the median filter started hurting performance varied by block size. These results are shown below.

These results show that for a **smaller block size, the filtering is generally more valuable**. This makes sense since **with a smaller block size, there will be more pixels at the boundary of the blocks,** so the median filter can help more.

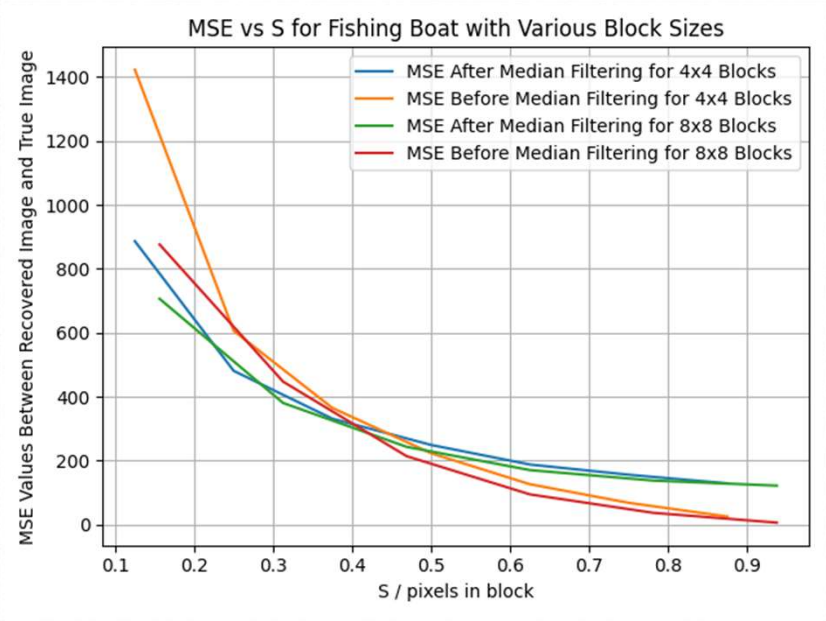
Image	Block Size	Ratio of Sampled Pixels to Total Pixels at which Median Filter Stops Helping
Fishing Boat	4x4	~0.44
Fishing Boat	8x8	~0.4
Nature	8x8	~0.23
Nature	16x16	~0.2
Nature	32x32	~0.12

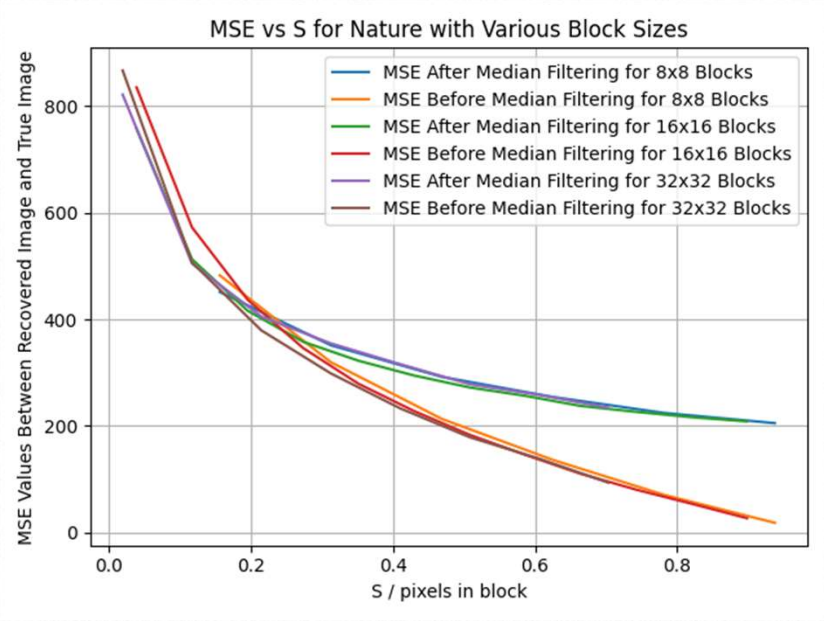
I also found that the type of filter affects the final output. When the filter is useful (at high levels of image corruption), the 3x3 mean filter performed best.

Factors That Impact the Quality of the Recovered Image: Size of Blocks

The larger the block size, our assumption about DCT coefficient sparsity starts breaking down. With smaller block size, there are more pixels located at the boundaries of pixels blocks. There are also less pixels available for cross validation and for estimation of the DCT coefficients.

This appeared to make a **larger difference when the image was more corrupted** (higher lower ratio S to total # of pixels in block), which you can clearly see on the left side of the left figure below. Here the error when using 4x4 blocks is significantly higher than that for 8x8 but for less corrupted images, the error for using 4x4 blocks is comparable to the error when using 8x8 blocks.



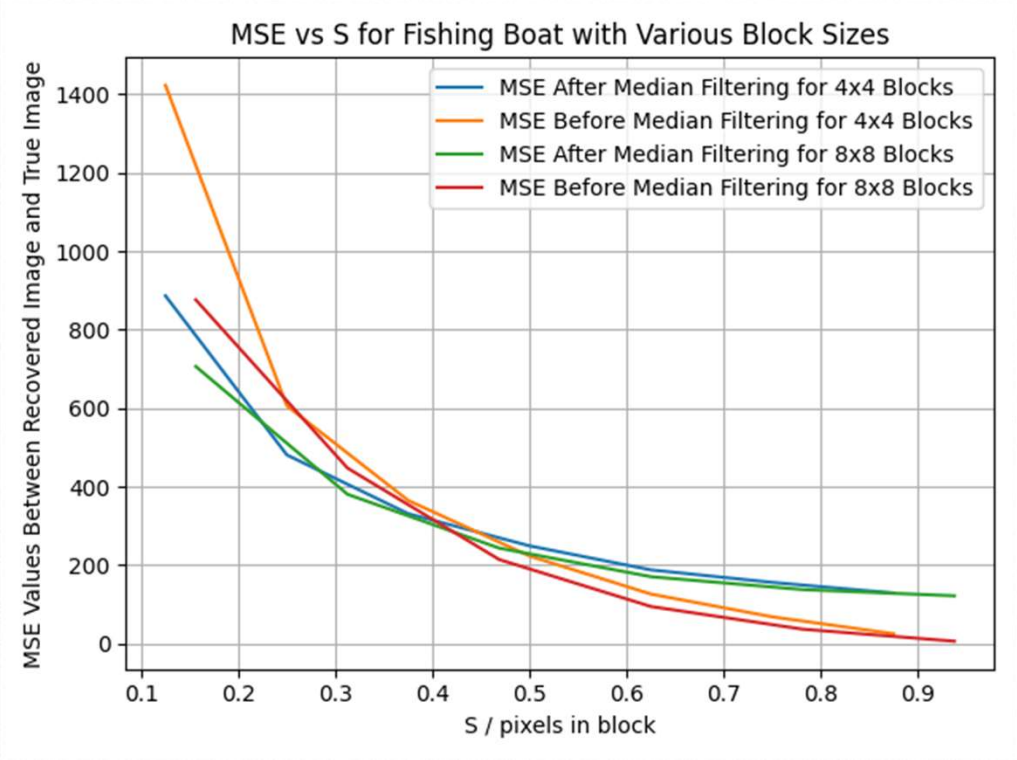


Limits and Problems with the Approach

Having to split up the image into blocks

Having to split up the image into blocks results in poor pixel estimation near the boundaries of the blocks. This is a necessary step in our approach since we are assuming sparsity in the DCT coefficients. The median filter is a way to minimize this, but it remains a problem.

We saw that when using 4x4 blocks instead of 8x8 blocks on fishing boat, the performance decreased, likely because there are more pixels located at the edge of a pixel block.



Possible Improvements That Can Be Made

Conditional Use of Filter

As we've seen, filtering the recovered image is only helpful at higher levels of corruption. To get an overall better output, we could only use the filter in the case that the image is corrupted more than a certain amount. This could be done by knowing beforehand how corrupted the image is or could be done by detecting the level of noise in the image before deciding whether to apply the filter. This includes switching filters, which incorporate the identification of corrupted and uncorrupted pixels and use that information to adapt the filtering process[10].

Overlapping Pixel Blocks

An issue we run into is the irregularities of the estimated pixels at the borders of the pixel blocks. A possible solution to this is to use pixel blocks that overlap with one another. For a pixel that is in multiple different pixel blocks, we could average the predicted pixel value to get the final pixel value. At the very least, this will mitigate the effects of the issues predicting pixel values on the edge of pixel blocks by averaging it with its value estimated when not on the edge of a pixel block.

Type of Filter

When the filtering was useful, the mean filter performed better than the median filter. When the image was not corrupted very much, neither filter helped, but the mean filter performed worst. If we can use the filter conditionally based on how much noise we detect, it seems that if the image is corrupted significantly, then using the mean filter instead of median will improve the quality of the recovered image.

10. Shah, Anwar, et al. "Comparative Analysis of Median Filter and Its Variants for Removal of Impulse Noise from Gray Scale Images." *Journal of King Saud University - Computer and Information Sciences*, 27 Mar. 2020, doi:10.1016/j.jksuci.2020.03.007.

References

References

Tibshirani, Robert. "Regression Shrinkage and Selection via the Lasso." Journal of the Royal Statistical Society. Series B (Methodological), vol. 58, no. 1, [Royal Statistical Society, Wiley], 1996, pp. 267–88, http://www.jstor.org/stable/2346178.

Tantum, Stacy. "Introduction to Machine Learning: Mini-Project 1 Compressed Sensing Image Recovery" Introduction to Machine Learning. Spring 2022.

Hodson, Timothy O., et al. "Mean Squared Error, Deconstructed." Journal of Advances in Modeling Earth Systems, vol. 13, no. 12, 23 Nov. 2021, doi:10.1029/2021ms002681.

Daniel Berrar, Cross-Validation, Editor(s): Shoba Ranganathan, Michael Gribskov, Kenta Nakai, Christian Schönbach, Encyclopedia of Bioinformatics and Computational Biology, Academic Press, 2019, Pages 542-545, ISBN 9780128114322, https://doi.org/10.1016/B978-0-12-809633-8.20349-X. (https://www.sciencedirect.com/science/article/pii/B978012809633820349X)

Marshall, Dave. "The Discrete Cosine Transform (DCT)." Cardiff School of Computer Science & Informatics, https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html.

References

Villar, Sebastián A., et al. "Median Filtering: A New Insight." Journal of Mathematical Imaging and Vision, vol. 58, no. 1, 2016, pp. 130–146., doi:10.1007/s10851-016-0694-0.

Marshall, Dave. "The Discrete Cosine Transform (DCT)." Cardiff School of Computer Science & Informatics, https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html.

Shah, Anwar, et al. "Comparative Analysis of Median Filter and Its Variants for Removal of Impulse Noise from Gray Scale Images." Journal of King Saud University - Computer and Information Sciences, 27 Mar. 2020, doi:10.1016/j.jksuci.2020.03.007.

G. George, R. M. Oommen, S. Shelly, S. S. Philipose and A. M. Varghese, "A Survey on Various Median Filtering Techniques For Removal of Impulse Noise From Digital Image," 2018 Conference on Emerging Devices and Smart Systems (ICEDSS), 2018, pp. 235-238, doi: 10.1109/ICEDSS.2018.8544273.

References: Python Packages

"Sklearn.linear_model.Lasso." Scikit Learn, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html. Accessed 3 March 2022.

"Scipy.signal.medfilt2d." Scipy.signal.medfilt2d - SciPy v1.8.0 Manual, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.medfilt2d.html. Accessed 3 March 2022.

NumPy, https://numpy.org/. Accessed 3 March 2022.

"Matplotlib.pyplot." Matplotlib.pyplot - Matplotlib 3.5.0 Documentation, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html. Accessed 3 March 2022.

"Scipy.ndimage.uniform_filter." Scipy.ndimage.uniform_filter - SciPy v1.8.0 Manual, docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.uniform_filter.html. Accessed 3 March 2022.

"Scipy.signal.convolve2d." Scipy.signal.convolve2d - SciPy v1.8.0 Manual, docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html. Accessed 3 March 2022.

Collaborations

Collaborations

I collaborated with the project group I was assigned to, which had Kevin Tu, Eric Qi, and Shaan Gondalia.

I shared and debated ideas with Kevin Tu, particularly about the process for calculating the T matrix since I was confused on its dimensions. Eric Qi, about specific extension ideas to try.

I compared results with Shaan Gondalia. I found through this that there were issues with resulting MSE values being too low because of the type of scalar that was being used in the calculation.

Who helped me overcome obstacles: Shaan Gondalia. One of these was the issues with my MSE values and the other was helping me understand what was needed in terms of calculating the intercept from LASSO without including it in the regularization.