COMPRESSED SENSING IMAGE RECOVERY

ADITYA BHAMIDIPATI

TABLE OF CONTENTS

INTRODUCTION

Image reconstruction is the process of recovering original images from corrupted images. These corrupted images suffer from the presence of noise which causes missing pixels. As a result, certain details within corrupted images appear grainy or obscured. Image noise is introduced by a multitude of sources including digital noise introduced by camera sensors. Therefore, a method capable of recovering corrupted images is an extremely valuable tool in the contexts of photography and computer vision.

The goal of this project is to develop an image reconstruction software tool that performs compressed sensing: reconstructing original images from corrupted images utilizing only a small number of uncorrupted pixels [1]. The concepts of utilizing two-dimensional discrete cosine transform that leverages lasso regression to estimate missing pixel values and applying median filtering to improve image quality and reduce image overall noise are explored to create an optimized and mathematically sound image reconstruction algorithm. The Image reconstruction algorithm is designed to first split the corrupted image into a set of KxK blocks where a certain number of pixels in each block are accurate while the remaining pixels are missing due to noise. The algorithm then recovers the missing pixels in each image block and the resulting processed blocks are concatenated and median filtered to create a recovered image. Therefore, another goal of this project was to analyze the impact of parameters such as block size and percentage of missing pixels per image block on the performance of the image reconstruction method which is done via simulation. In the simulation, the reconstruction algorithm first takes an uncorrupted image and splits the corrupted image into a set of KxK blocks. The algorithm then performs a sampling method on each block where a certain number of pixels in the block are sensed (sampled) while the remaining pixels are dropped to simulate the presence of noise. Then, the missing pixels in each image sampled block are recovered and the blocks are concatenated to create a recovered image that is compared against the original, uncorrupted image to evaluate algorithm performance.

One of the key results yielded from evaluating performance of the image reconstruction algorithm utilizing the simulation was that as the number of sampled pixels per image block increases, applying median filtering to the resulting recovered block diminishes image quality. Additionally, after comparing the impact of different image blocks sizes on algorithm performance, it is determined that larger image block size leads to better algorithmic performance.

MATHEMATICAL FORMULATION

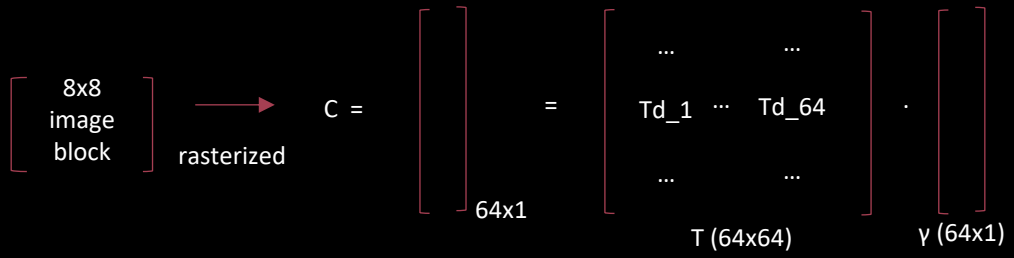
The core mathematical concepts of this image reconstruction algorithm are:

1) Two-dimensional discrete cosine transform (DCT) | Page 5 & 6
2) L1-norm lasso regression | Page 7
3) Random subset cross validation (CV) | Page 8 & 9
4) Median filtering | Page 10

The ordering of these concepts reflects the sequential theorization and development of this algorithm in Python. It is important to note that an input image is split into KxK blocks and each of these blocks are individually recovering utilizing a compressed sensing method developed using concepts (1-3). The recovered blocks are then combined, and median filtering (concept 4) is applied to create a fully recovered image. The resulting algorithm architecture is visualized on page 11. Additionally, an overview of the packages utilized when developing the algorithm in Python is provided on page 12.

TWO-DIMENSIONAL DCT

2D DCT is a transformation that converts a KxK image block (contains K*K pixels) into separate parts using spatial frequencies in both the horizontal and vertical directions [2]. More specifically, the image is converted into a row vector that is split into a basis (transformation) matrix and coefficient vector. Consider an 8x8 image block:



Each Td column in the matrix T contains values for all locations in the image [(x,y) pairs] for a fixed combination of horizontal and vertical spatial frequencies [(u,v) pair] while the γ vector contains all the DCT coefficients for each spatial frequency pair (u,v). The values that populate each Td column are calculated using the equation:

$$\sum_{u=1}^{P}\sum_{v=1}^{Q}\alpha_u \cdot \beta_v \cdot \cos\frac{\pi(2x-1)(u-1)}{2\cdot P}\cdot\cos\frac{\pi(2y-1)(v-1)}{2\cdot Q}\cdot G(u,v)$$ [6]

x,u ∈ {1,2, … , p}
y,v ∈ {1,2, … , Q}
* For an 8x8 block, P & = 8

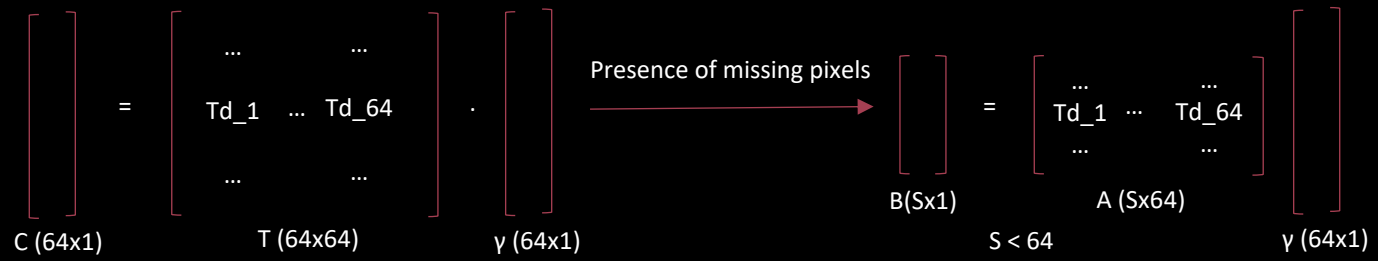
$$*\alpha_u = \sqrt{1/P}\ (u=1);\ \alpha_u = \sqrt{2/P}\ (2 \le u \le P)$$

$$*\beta_v = \sqrt{1/Q}\ (v=1);\ \beta_v = \sqrt{2/Q}\ (2 \le v \le Q)$$

When the equation above is utilized for an 8x8 block, each (u,v) pair yields an 8x8 matrix with is subsequently converted into a 64x1 vector using the same rasterization method used to convert the original 8x8 image block into vector C. The resulting vector is then used as a column of matrix T. Overall, this is the method created to derive the transformation matrix during the 2D DCT transformation concept.

TWO-DIMENSIONAL DCT

On the last page, the 2D DCT concept of using spatial frequencies to create a transformation matrix T for any image block size was outlined. The focus is now on deriving the values (DCT coefficients) in vector γ. If all the pixels in row vector C are known, then utilizing the derived matrix T, the DCT coefficients in vector γ can be derived by using $\gamma = T^{-1} * C$. However, in corrupted image blocks, not all pixel values are known. Hence, this problem turns into an under-determined linear system problem (shown using an 8x8 image block):



Therefore, a compressed sensing method to derive DCT coefficient values using only a limited number of pixels must be implemented to recover the image block. This is where the concept of regression is leveraged highlighting why a 2D DCT framework was chosen to represent the image block. More specifically, a regression model is formulated where the matrix B (contains the set of known pixel values of the corrupted image block) represents the target variable while each column of matrix A (contains rows of transformation matrix T corresponding to the known pixel values in matrix B) represents a predictor variable. Regression seeks to determine model parameters (weights) by minimizing the error (formulated by the residual sum of squares) between model predictions (predictor data * weights) and the actual target data. Hence, regression is utilized to estimate the weights corresponding to each predictor variable. These weights are the DCT coefficients of interest. This concept of regression to derive DCT coefficient of a corrupted image block will further be explored in the next section.

L1-NORM LASSO REGRESSION

As discussed in the previous section, regression was the underlying concept when theorizing a compressed sensing method to derive DCT coefficient values using only a limited number of pixels due to representation of a corrupted image block as an under-determined linear system. Since images tend to be sparse in the DCT domain (a small number of coefficients are non-zero), for this specific project, L1-norm lasso regression is the chosen form of regression. Lasso regression utilizes the L1 regularization method which implements a penalty equal to a regularization term (λ) multiplied by the absolute value of the sum of the magnitudes of the coefficients (L1-norm) into the objective error function [3]:

$$\min_{\gamma} \left\| A\gamma - B \right\|_2^2 + \lambda \left\| \gamma \right\|_1$$ [6]

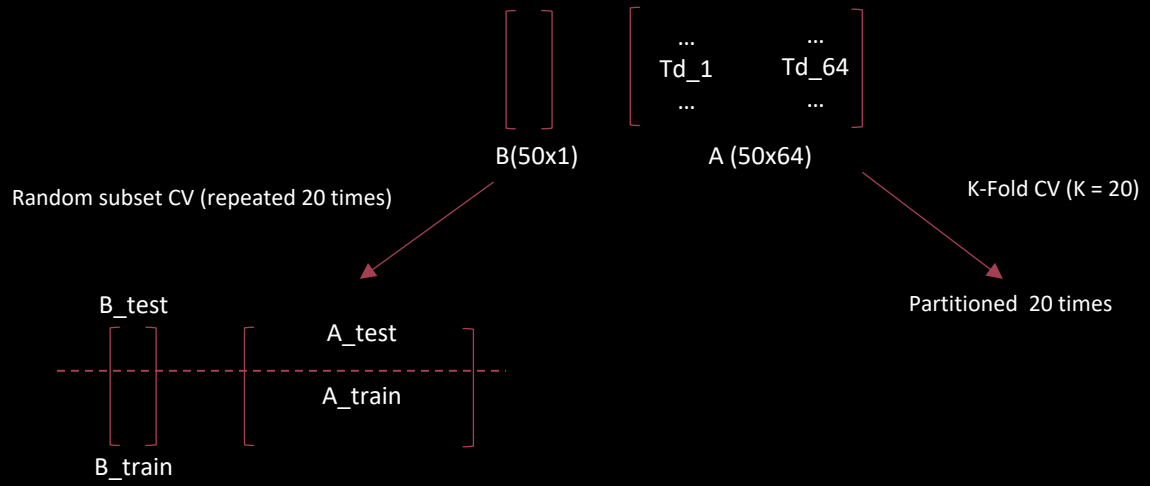
Regression Error Function

Penalty imposed by lasso regression

The goal of regularization in general is to penalize large coefficient values (weights) constraining them to be small to reduce the overall variance of the regression model. By reducing the variance, the bias of the model increases allowing the model produce more consistent results when predicting on different testing datasets avoiding overfitting. In the context of this project, when a lasso model is fitted (trained) using the A and B matrices and used to estimate the DCT coefficients (weights), the L1-norm regularization imposes a constraint on DCT coefficients leading to some coefficient values equaling zero creating the desired sparsity in the DCT domain. Therefore, by utilizing L1-norm lasso regression, the DCT coefficient vector γ estimated better represents the true values of the DCT coefficient. The optimal value for the regularization term λ to use when performing lasso regression is derived through cross validation and will further be explored in the next section.

RANDOM SUBSET CV

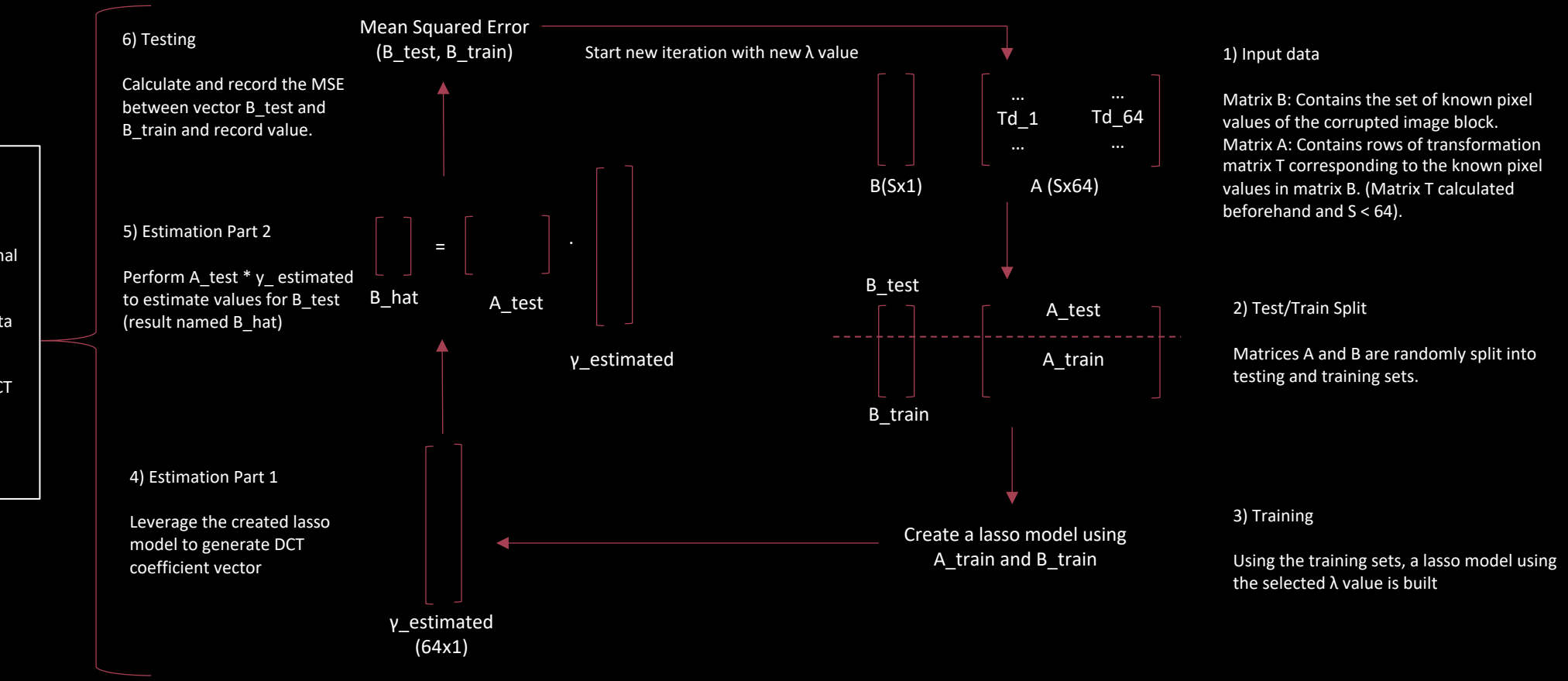
As stated in the previous section, L1-norm lasso implements a penalty equal to a regularization term (λ) multiplied by the absolute value of the sum of the magnitudes of the coefficients (L1-norm) into the objective error function. Therefore, when constructing a lasso model to estimate the DCT coefficient vector γ, finding an optimal value regularization term (λ) is of great importance. This is done by using the random subset cross validation (CV) method which independently tests multiple values of λ. In random subset CV, data is randomly portioned into training and testing subsets which are then used to create a model and evaluate the performance of the model [4]. In this project, random subset CV is performed iteratively 20 times to evaluate 20 lasso models that utilize different values of λ and has been chosen as the preferred method of CV over other methods such as K-Fold CV. In K-Fold CV, the data set is split into K number of subsets, called folds. Then, the model is iteratively fit K times, each time training the data on K-1 of the folds and evaluating on the Kth fold [4]. The reason random subset CV is utilized in this project is because of the lack of data present in corrupted blocks coupled with the goal of testing 20 different values of λ. Consider a corrupted 8x8 that contains 50 known pixels:



From the visualization above, iteratively performing random subset CV 20 times as opposed to portioning the data set 20 times allows for a larger testing data set to evaluate each of the 20 λ values of interest making it the optimal choice of CV. The random subset cross validation method is further detailed on the next page. **Programmatically:**

RANDOM SUBSET CV

Random subset cross validation using a corrupted 8x8 image block:



This iterative process is performed 20 times. Once all the iterations are complete:

1) The λ with the lowest associated MSE value is chosen as the optimal regularization value.
2) Using the derived λ, a new lasso model is created using all the data contained in matrices A and B.
3) The resulting lasso model is leveraged to estimate the the DCT coefficient vector γ.
4) Transformation matrix T * γ is performed to recover the corrupted block.

1) Input data

Matrix B: Contains the set of known pixel values of the corrupted image block.
Matrix A: Contains rows of transformation matrix T corresponding to the known pixel values in matrix B. (Matrix T calculated beforehand and S < 64).

2) Test/Train Split

Matrices A and B are randomly split into testing and training sets.

3) Training

Using the training sets, a lasso model using the selected λ value is built

MEDIAN FILTERING

After the concepts of 2D DCT, lasso regression, and random subset cross validation are utilized to recover (estimate) a corrupted image block, median filtering is applied to the recovered to improve image block quality via smoothing. Median filtering is the process of replacing each pixel in a given image block by the median of neighboring pixels [5]. Example:

100	120	110
90	X = 130	115
117	95	118

3x3 median filtering →

X = median(90, 95, 100, 110, 115, 117, 118, 120, 130) = 115

mean(90, 95, 100, 110, 115, 117, 118, 120, 130) = 115

100	120	110
90	115	115
117	95	118

The benefit of this filtering technique in comparison to other forms of filtering such as geometric mean filtering – another smoothing technique which replaces pixels in a given image block by the mean of neighboring pixels [5] – is that median filtering is adept at eliminating outlier neighboring pixel values when determining the value of a pixel. Consider the example:

100	400	110
90	X = 130	115
117	95	118

→

X = median(90, 95, 100, 110, 115, 117, 118, 130, 400) = 115

100	400	110
90	115	115
117	95	118

From the two examples, it can be observed that the outlier pixel value of 400 does not change the derived value of pixel X. However, if mean filtering was applied, the value for pixel X in the first example would be equal to 110.5 would be 141.7 in the in example 2 differing significantly. Hence, 3x3 median filtering is leveraged in this project to process recovered blocks.

ALGORITHM ARCHITECTURE

In the simulation, this algorithm architecture is adjusted where the input image is an uncorrupted image. Hence when the for each block, once the block has been rasterized into the C vector, a sampling method is applied (simulates noise in the image block by dropping a certain number of pixels) to sample the C vector and transformation matrix creating matrices B and A. This is done so that once a recovered image is constructed, it can be directly compared to the input image to gauge algorithm performance.

Uncorrupted Image used as input in simulation

Corrupted Image (Input Image)

Input Image split into KxK blocks + Transformation matrix T calculated (Same for all KxK blocks)

T =

K

K

Sampling method applied in simulation

For each

C (Rasterized image block)

T

Remove missing pixel rows

B

A

Random Subset CV

Optimal λ

Create a lasso model using A, B, and the optimal λ and estimate the DCT coefficients

γ

T * γ is performed to estimate the corrupted block.

Estimated C

recovered block

The missing pixels in C are replaces with value estimates in Estimated C to create recovered block

KxK blocks concatenated

Recovered Image

3x3 Median Filtering

Filtered Recovered Image

SOFTWARE PACKAGES LEVERAGED

The image processing algorithm outlined on the last page was developed in Python leveraging multiple predeveloped packages:

1) Two-dimensional discrete cosine transform (DCT)
   - Utilized the NumPy package's matrix utilities [11].
2) L1-norm lasso regression
   - Utilized the NumPy package's matrix utilities.
   - Utilized the Scikit-learn package's linear_model.Lasso method for regression [9].
3) Random subset cross validation (CV)
   - Utilized the NumPy package's matrix utilities.
   - Utilized the Random package's sample method for random subset data splitting [7].
   - Utilized the Scikit-learn package's linear_model.Lasso method for regression.
4) Median filtering
   - Utilized the SciPy package's signal.medfilt2d method for filtering [10]

All simple mathematical procedures in the algorithm are performed utilized the Math package [7]. The processes of converting the input image into a NxM matrix and converting the resulting NxM recovered matrix to an image were performed utilizing the Pillow package's Image method [8]. Additionally, the sampling method implemented in the simulation also leveraged the Random package's sample method.

EXPERIMENTAL RESULTS

Once the image reconstruction algorithm has been theorized and developed in Python, performance of the algorithm is evaluated via simulation. As stated previously, in each simulation, the image reconstruction algorithm first splits an original uncorrupted image is split into KxK blocks. The algorithm then samples each block to artificially create a corrupted image block, recovers the corrupted block, and assembles the resulting recovered blocks into a recovered image that is compared against the original image.

Simulations :

The goals of these simulations are to analyze the impacts of image block size, number of missing pixel per block, and median filtering on the performance of the image reconstruction algorithm.

FISHING BOAT IMAGE SIMULATION

In this simulation, an uncorrupted fishing boat image is utilized as the input image. The image block size is set to 8x8, and 5 independent runs of the simulation are performed for increasing pixel sample sizes (the number of pixels in each block that sensed). For each simulation run, the recovered image before and after median filtering is compared to the original input image via mean squared error (MSE).

Input Image

FISHING BOAT IMAGE SIMULATION

Sample Size = 10 Pixels

Before filtering MSE:
916.4079373554408

3x3 Median Filtering

After filtering MSE:
748.7113171254011

Sample Size = 20 Pixels

Before filtering MSE:
445.56384162925366

3x3 Median Filtering

After filtering MSE:
411.1653115140807

FISHING BOAT IMAGE SIMULATION

Sample Size = 30 Pixels

Before filtering MSE:
224.4894271944275

3x3 Median Filtering

After filtering MSE:
260.79496854050194

Sample Size = 40 Pixels

Before filtering MSE:
113.07215889662434

3x3 Median Filtering

After filtering MSE:
181.54635329199553

FISHING BOAT IMAGE SIMULATION

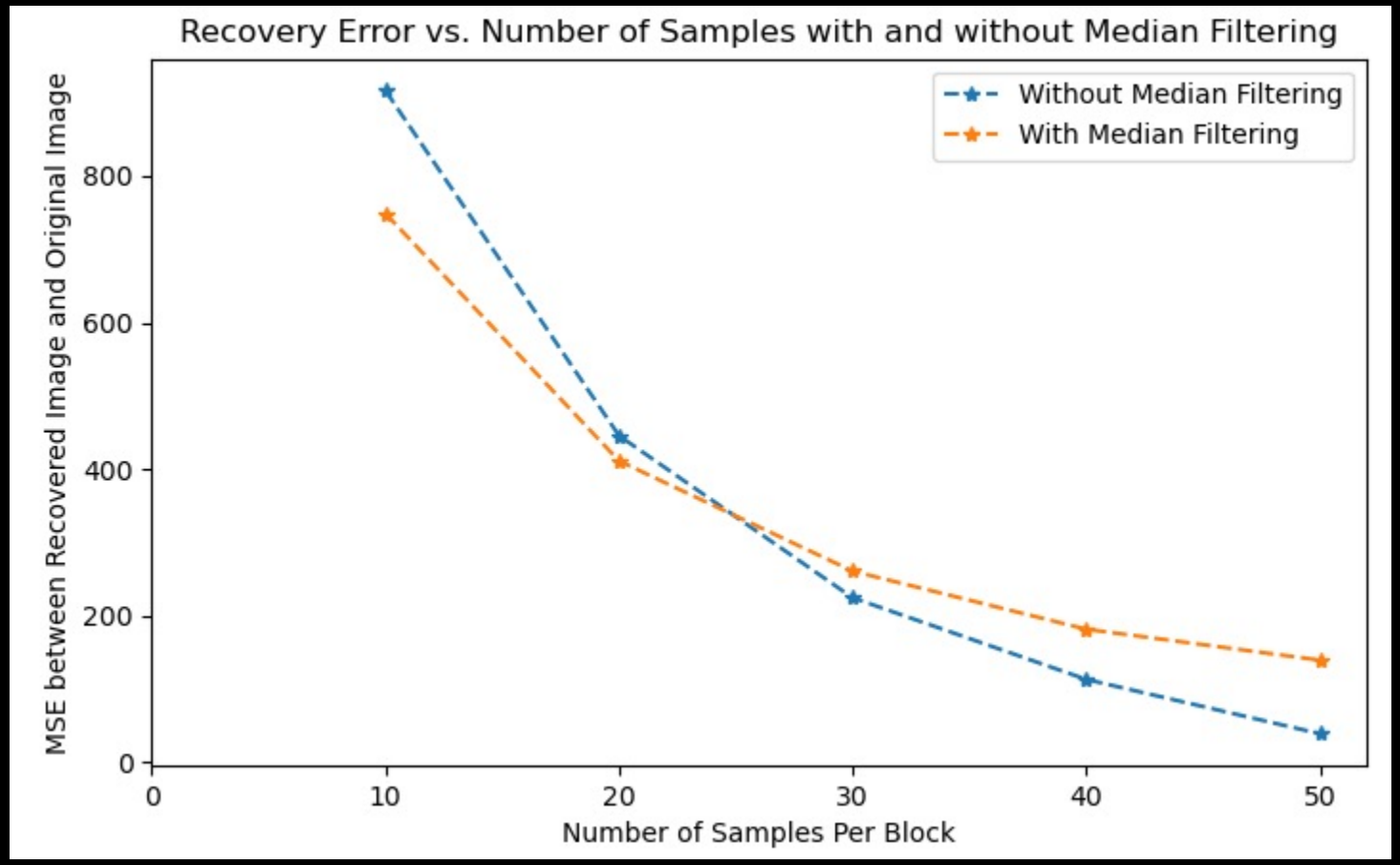
Sample Size = 50 Pixels

Before filtering MSE:
38.97056462540206

3x3 Median Filtering

After filtering MSE:
139.37475330643235

FISHING BOAT IMAGE SIMULATION

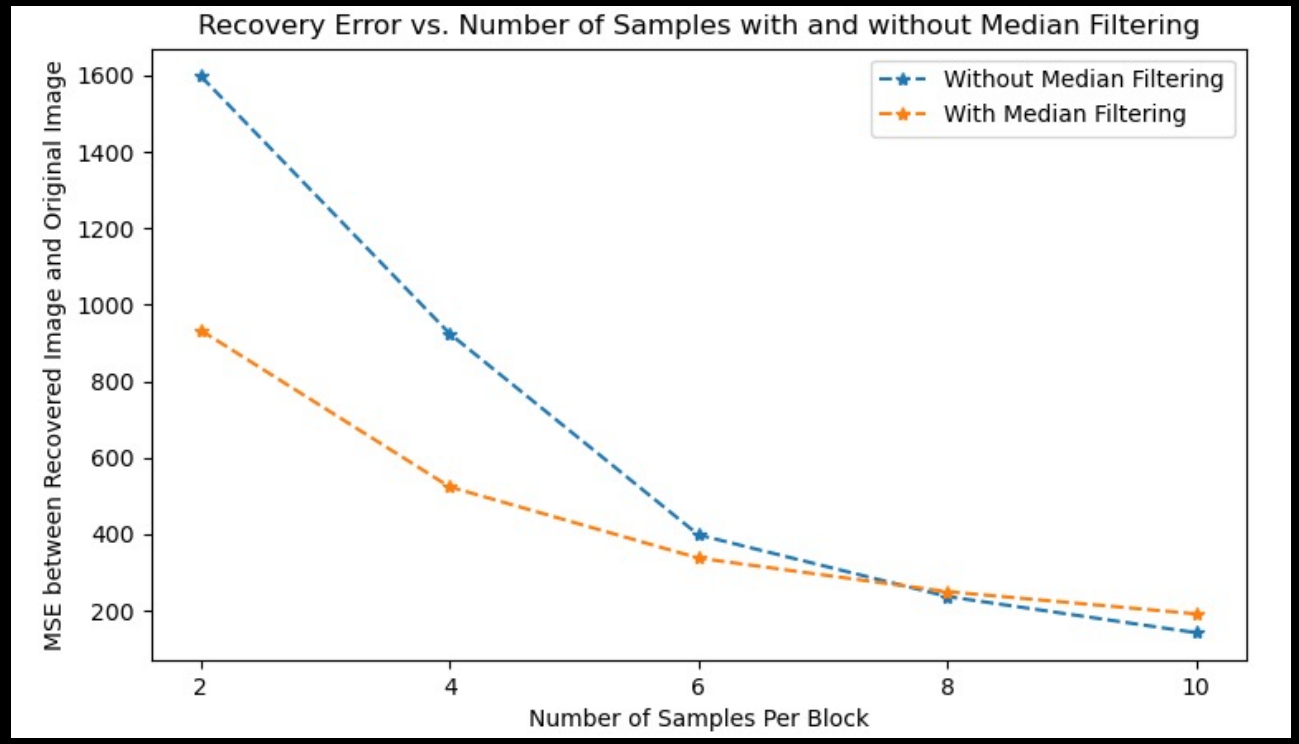


The key take aways from the 5 different runs of the fishing boat image simulation are:

- As the number of samples per block increases, the recovery error between the recovered image and original image decreases significantly.
- There is a tradeoff between number of samples per block and median filtering: For a lower number of samples per block, applying median filtering to the recovered image improves accuracy, however, for a larger number of samples per block median filtering diminishes accuracy.

FISHING BOAT IMAGE SIMULATION

To analyze the impact of image block size on algorithm performance, image block size is set to 4x4, and 5 independent runs of the simulation are performed for pixel sample sizes 2, 4, 6, 8, and 10:



After analyzing the simulation runs done for image block size 8x8 against the simulation runs done for image block size 4x4 it is determined that algorithmic performance is higher for the larger block size 8x8. This is done by comparing the simulation runs for the different block sizes that have the same percentage of missing pixels per block. More specifically, the simulation for image block size 8x8 with sample size equal to 40 and the simulation for image block size 4x4 with sample size equal to 10 both have the percentage of missing pixels per block equal to 62.5% (40/64 = 10/16 = .625). For the 8x8 block simulation (pixel sample size = 40), the recovery error is equal to 113.07215889662434 before median filtering and 181.54635329199553 after median filtering. For the 4x4 block simulation (pixel sample size = 10), the recovery error is equal to 142.66462884323565 before median filtering and 191.87063111452923 after median filtering which is higher than the 8x8 simulation in both scenarios. Therefore, the larger block enables better algorithmic performance.

NATURE IMAGE SIMULATION

In this simulation, an uncorrupted nature boat image is utilized as the input image. The image block size is set to 16x16, and 5 independent runs of the simulation are performed for increasing pixel sample sizes. For each simulation run, the recovered image before and after median filtering is compared to the original input image via mean squared error (MSE).

Input Image

NATURE IMAGE SIMULATION

Sample Size = 10 Pixels

Before filtering MSE:
887.3808371920459

3x3 Median Filtering

After filtering MSE:
774.9180608716736

Sample Size = 30 Pixels

Before filtering MSE:
561.955590245396

3x3 Median Filtering

After filtering MSE:
516.502846918607

NATURE IMAGE SIMULATION

Sample Size = 50 Pixels

Before filtering MSE:
443.37776805460163

3x3 Median Filtering

After filtering MSE:
436.7105080345377

Sample Size = 100 Pixels

Before filtering MSE:
264.768092421538

3x3 Median Filtering

After filtering MSE:
333.5478451821609

NATURE IMAGE SIMULATION

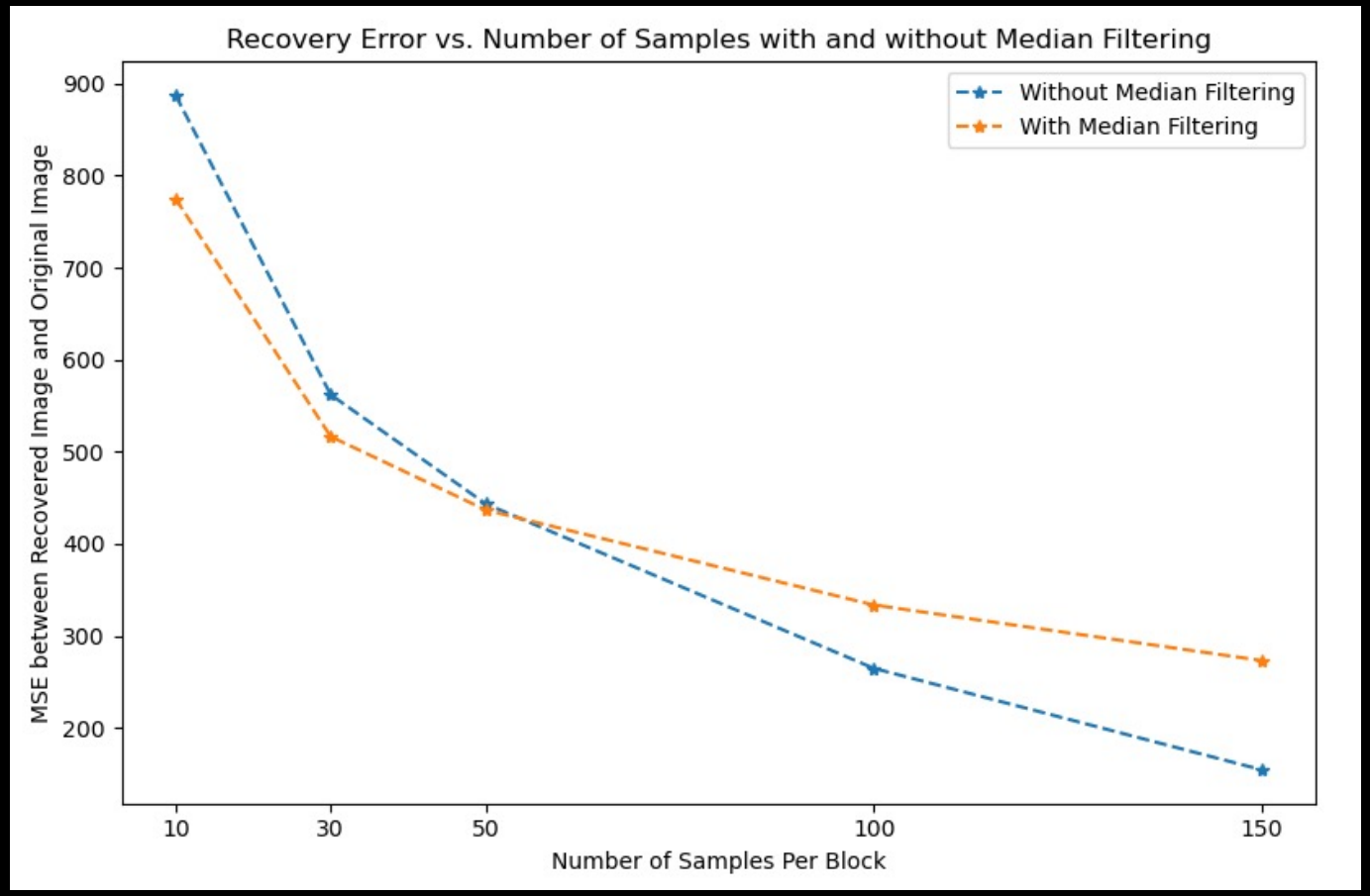
Sample Size = 150 Pixels

Before filtering MSE:
154.4752317836555

3x3 Median Filtering

After filtering MSE:
273.20743996965734

NATURE IMAGE SIMULATION



Evaluation of 5 different runs of the nature image simulation adds further support the idea that the recovery error between the recovered image and original image decreases significantly as the number of samples per block increases and the idea of the tradeoff between number of samples per block and median filtering. An important insight that can be observed when comparing the 8x8 fish boat image simulations and the nature image simulations is that for pixel sample sizes 10, 30 and 50, the nature image simulations consistently perform better (have lower recovery error) than the fish boat image simulations. This adds further support to conclusion drawn when comparing the fishing boat image simulation runs done for image block size 8x8 (sample sizes 10. 20, 30, 40, 50) against the fishing boat image simulation runs done for image block size 4x4 (sample sizes 2, 4, 6, 8, 10): larger block sizes allow for better algorithmic performance.

DISCUSSION / CONCLUSIONS

The experimental results yielded from the fishing boat image and the nature image simulations emphasize a direct correlation between pixel sample size and algorithmic performance. More specifically, As the number of samples per image block increases, the recovery error between the recovered image and the original image decreases. The reason for this is because regression is able to provide more precise estimates for model weights given more training data. In the context of this project, an increased number of samples (training data) allows the developed lasso regression models to provide more accurate DCT coefficient estimates for individual image blocks. The DCT coefficients are utilized to recover the individual blocks which are combined to create a recovered image. An additional trend observed from simulation analysis is that a larger image block size allows enables better algorithmic performance. More specifically, simulations that utilize larger block sizes have smaller recovery errors between the recovered image and the original image. This can be attributed to the fact that increasing the number of predictor variables enables a regression model to derive a better fit: variance in the target variable is better explained by the model allowing for better estimation of model parameter weights (quantified by the coefficient of determination). Previously, it is stated that each column of the transformation matrix T (that refers to a fixed pair of horizontal and vertical frequencies) represents a predictor variable, hence, by increasing the image block size, the transformation matrix has more columns. Since the Matrix T (predictor data) and corresponding pixels values (target data) for individual image blocks are utilized to develop lasso regression models that estimate DCT coefficients for the individual blocks, larger image blocks have more corresponding predictor data and in turn, have more accurate DCT coefficient estimates. The more accurate DCT coefficient estimation allows for larger image blocks to be recovered more accurately than smaller blocks.

An insight derived from analyzing the “Recovery Error vs Number of Samples with and without Median Filtering” visualizations for both the fishing boat image and the nature image simulations is that there exists a tradeoff between the number of samples per block and median filtering in terms of recovered image accuracy. In both types of simulations, it can be observed that as the number of samples per image block increases, the recovered image without median filtering is more accurate (has less recovery error) in comparison to the original input image than the recovered image with median filtering. Therefore, a design specification that enforces that median filtering should not be applied when the percentage of sensed/sampled pixels to the total number of pixels in a given image block is approximately 45%.

DISCUSSION / CONCLUSIONS

From an efficiency standpoint, a method that would greatly improve algorithmic efficiency is multithreading. The image recovery algorithm splits an input image into a set of KxK blocks, recovers each block, and concatenates the resulting blocks to produce the output recovered image (concatenated recovered image is first median filtered). Currently, the blocks are processed sequentially where a single block is recovered at a time. Since each KxK block is completely independent of the other blocks – recovering a block does not rely on any other blocks - multiple blocks can be processed concurrently improving the runtime of the algorithm. By implementing multithreading, a multitude of independent execution processes (threads) that perform block recovery concurrently. Additionally, a synchronization barrier can be implemented to ensure all image blocks have been processed before concatenation and filtering. I was initially interested in utilizing the multiprocessing Python package to implement multithreading into the algorithm but due to time restrictions was unable to. However, I believe that multithreading would immensely improve algorithmic performance in terms of efficiency.

In conclusion, a compressed sensing-based image reconstruction algorithm was successfully theorized and developed in python. Through simulation, trends that impact algorithmic performance have been analyzed, and further design specifications to not only improve algorithmic performance but also efficiency have been conceptualized.

REFERENCES

[1] D. L. Donoho, "Compressed sensing," in *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289-1306, April 2006, doi: 10.1109/TIT.2006.871582.

[2] C. S. Park, "2D Discrete Fourier Transform on Sliding Windows," in *IEEE Transactions on Image Processing*, vol. 24, no. 3, pp. 901-907, March 2015, doi: 10.1109/TIP.2015.2389627.

[3] 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.

[4] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, [Morgan Kaufmann Publishers Inc.], 1995, pp. 1137–1143.

[5] G. Devarajan, V. K. Aatre and C. S. Sridhar, "Analysis of median filter," in *Proceedings of the XVI Annual Convention and Exhibition of the IEEE In India*, 1990, pp. 274-276, doi: 10.1109/ACE.1990.762694.

[6] Tantum, Stacy. “Introduction to Machine Learning: Mini-Project 1 Compressed Sensing Image Recovery.” 2022. PDF file.

[7] Van Rossum, G. (2020). The Python Library Reference, release 3.8.2. Python Software Foundation.

[8] Clark, A. (2015). Pillow (PIL Fork) Documentation. readthedocs. Retrieved from https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

REFERENCES

[9] Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12(Oct), 2825–2830.

[10] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17, 261–272. https://doi.org/10.1038/s41592-019-0686-2

[11] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.