Project 1: Image Recovery

Hongrui Guo
steven.guo@duke.edu

Abstract

This project uses inverse discrete cosine transform and orthogonal matching pursuit to recover a “damaged” image. The image is divided into numbers of K×K small blocks to speed up the computation, and data are sparse sampled from them. Then the blocks are recovered, stitched together, and then be applied with median filter to improve the quality. This report would also cover how the algorithm is improved and some other findings.

1. Overview

In this project, a MATLAB program was developed for recover a grey scale image using compressed sensing. Usually reconstruct parts of the signal (2D image in our case) that were not sampled. However, from some prior knowledge and reasonable assumptions (e.g. the sparsity) reconstruction of the signal becomes possible.

Before divide the image into multiple K×K small blocks, the inverse discrete cosine transform (IDCT) matrix was firstly computed since it is only dependent on the block size. Then each block will be sparse sampled.

After that, orthogonal matching pursuit (OMP) algorithm would iterate from 1 to sample size from choosing the optimal sparsity of discrete cosine transform (DCT) coefficients. This step is repeated for multiple time on different training and testing size. This Monte Carlo cross-validation process is for fit the model (sparsity vs. error) as well as possible.

Then the sparsity that produce the lowest average error in the testing set would be chosen. Then the estimated DCT coefficients would be computed from all the samples. Then the block can be recovered by simply multiplying IDCT matrix and the estimated coefficients.

Finally, the recovered block would be stitched together and be applied with a median filter to reduce mean square error (MSE) by removing some of the noise.

Section 2 will explain the how to formulate this image recovery problem as a linear equation system, and how the OMP algorithm works and how it was been improved for better efficiency and accuracy. Section 3 will present the recovered images and some other collected data. Section 4 will discuss the results and other improvements on the algorithm and the code implementation.

2. Mathematical Formulation

a. Formulate Image Recovery as a Linear System

The DCT transformation for each block formulated as the following:

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

$$x,u \in \{1,2,\cdots,P\}, \qquad y,v \in \{1,2,\cdots,Q\}$$

$$\alpha_u = \begin{cases} \sqrt{1/P} & (u=1) \\ \sqrt{2/P} & (2 \le u \le P) \end{cases} \qquad \beta_v = \begin{cases} \sqrt{1/Q} & (u=1) \\ \sqrt{2/Q} & (2 \le u \le Q) \end{cases}$$

P, Q are the width and height of the block; x, y are the pixels’ coordinates.

B would be a vector of all pixels: $g(x,y)$ (flattened block); α would be a vector of DCT coefficients: $G(u,v)$; then the remaining nested summation would form matrix A, also known as the transformation matrix T.

Then we can try to find the optimal α by solving the following:

$$\min_{\alpha} \|A\cdot\alpha - B\|_2^2$$

Or by solving the following linear system:

$$B = A\cdot\alpha$$

For a complete block of the image, the linear equation system would be determinant and have shape as the following:

$$\underbrace{\begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix}}_{B_{complete}} = \underbrace{\begin{bmatrix}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\end{bmatrix}}_{A_{complete}} \cdot \underbrace{\begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix}}_{\alpha}$$

The picture is sparse sampled with size $S \le PQ$ and split into testing set S_{test} and training set S_{train}. The testing set size S_{test} is computed by $S_{test} = \left\lfloor \frac{S}{6} \right\rfloor$ and the training set size $S_{train} = S - S_{test}$. The length of B and the number of rows of A would be equal to training set size S_{train}, which is smaller than PQ. The new system needs to be solved for recovering image would shape as the following:

$$\underbrace{\begin{bmatrix}\times\\\times\\\times\end{bmatrix}}_{B} = \underbrace{\begin{bmatrix}\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\\\times&\times&\times&\times&\times\end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix}\times\\\times\\\times\\\times\\\times\end{bmatrix}}_{\alpha}$$

That make the system under-determined, cannot be simply solved as a linear system $A\alpha = B$.

Since the DCT coefficients, α, are sparse, a unique and deterministic solution can be found. By solving the following:

$$\min_{\alpha} \|A\alpha - B\|_2^2$$
$$\text{S.T.} \|\alpha\|_0 \le \lambda$$
$$\lambda = 1,2,\dots,S$$

To solve this optimization problem, orthogonal matching pursuit would be used, which would be explained in detail in the next subsection.

b. Orthogonal Matching Pursuit Algorithm

Orthogonal matching pursuit (OMP) algorithm is a greedy algorithm for computing a sparse and approximate solution for a linear system.

The OMP algorithm used in the experiment are as the following:

```
Algorithm: Orthogonal Matching Pursuit
Input    $A, B, \lambda, P, Q$
Output   $\alpha$
Step 1   $F \leftarrow B, \Omega \leftarrow \emptyset, p \leftarrow 1, \theta \leftarrow 0_{PQ,1}, \alpha \leftarrow 0_{PQ,\lambda}$
Step 2       $\theta \leftarrow \left\{\theta_i = \langle \frac{A_i}{\|A_i\|}, F \rangle \middle| \forall A_i \in A \right\}$
Step 3       $s \leftarrow \{s \mid |\theta_s| \ge |\theta_i|, \forall \theta_i \in \theta\}$
Step 4       $\Omega \leftarrow \Omega \cup \{s\}$
Step 5       $\alpha_{i\in\Omega,p} \leftarrow \min_{\alpha_i, i\in\Omega} \left\| \sum_{i\in\Omega} \alpha_i \cdot A_i - B \right\|_2^2$
Step 6       $F \leftarrow B - \sum_{i\in\Omega} \alpha_i \cdot A_i$
Step 7       if $p \le \lambda$
                 $p \leftarrow p + 1$
                 goto Step 2
             else
                 return $\alpha$
             Note: $\alpha_i \in [1, PQ], \forall i \in \Omega;\ \alpha_i = 0, \forall i \notin \Omega$
```

In *Step 5*, α_i is computed by solving the following:

$$B = A_{i,i\in\Omega} \cdot \alpha_{i,i\in\Omega}$$

By using the mldivide (backslash, \) function in MATLAB, the optimal α_i can be obtained.

Two modifications were made on top of the original algorithm presented in the slides to improve the performance and accuracy. In *Step 2*, each vector A_i is normalized since B is projected to each A_i. Without normalization, the angles between the A_i and B are no longer the only factor for choosing the optimal A_i; the magnitude of each A_i would also affect the result of inner product and produce undesired solution.

The second improvement is to record each $\alpha_{i,p}$ in each iteration instead of recording the last α_i for each λ. This improved the computation complexity from $O(n^2)$ to $O(n)$.

After getting the α, a vector of pixels B' or $\hat{g}(x,y)$ can be recovered by simply compute $B' = A \cdot \alpha$. Then the mean square error for the testing set is calculated as the following:

$$\varepsilon = \frac{1}{S_{test}} \sum_{S_{test}} [\hat{g}(x,y) - g(x,y)]^2$$

Monte Carlo cross-validation, the process of fitting randomly split data for multiple times, is applied for 20 time for each λ. Then the average error for a λ can be calculated as the following:

$$\varepsilon_\lambda = \frac{1}{20} \sum_{i=1}^{20} \varepsilon_{\lambda,i}$$

The λ with the lowest error will then be chosen to recover the block form the sample (both training and testing set).

3. Experimental Results

For the small test image boat.bmp, the block size was set to 8×8 and 5 sample size $S = 10,20,30,40,50$ was chosen. The MSE, computation time, and the recovered images are as the following:

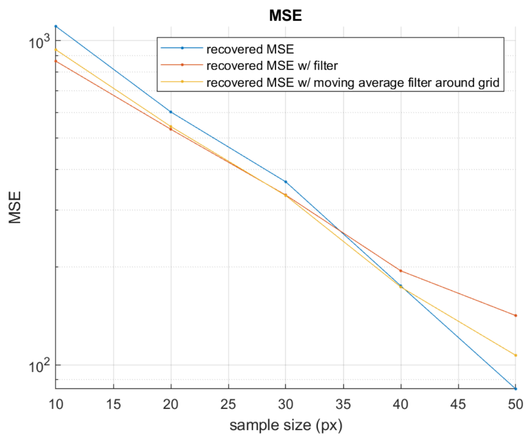

Fig 1. sample size vs. MSE

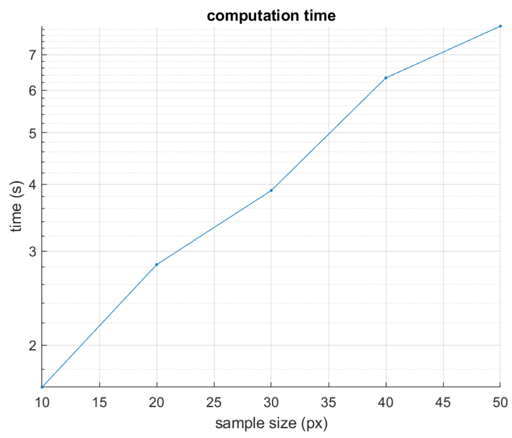

Fig 2. sample size vs. computation time

Sample size (px)	10	20	30	40	50
MSE of Recovered image	1.28E3	6.02E2	3.66E2	1.91E2	7.86E1
MSE of recovered image w/ filter	9.29E2	5.35E2	3.33E2	2.02E2	1.45E2
MSE of recovered image w/ moving average filter around grid	1.03E3	5.48E2	3.34E2	1.86E2	1.05E2
Computation time (s)	1.87	2.84	4.14	6.32	8.10

Table 1. sample size, MSE, and computation time for small test image

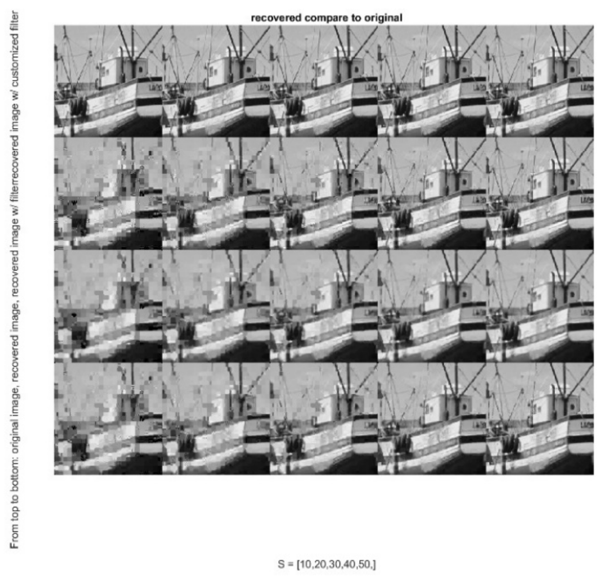
recovered compare to original
From top to bottom: original image, recovered image, recovered image w/ filterrecovered image w/ customized filter
S = [10,20,30,40,50,]

Fig 3. Recovered image vs. original image (w/ and w/o filter)

For the large test image nature.bmp, the block size was set to 16×16 and 5 sample size $S = 10, 30, 50, 100, 150$ was chosen. The MSE, computation time, and the recovered images are as the following:

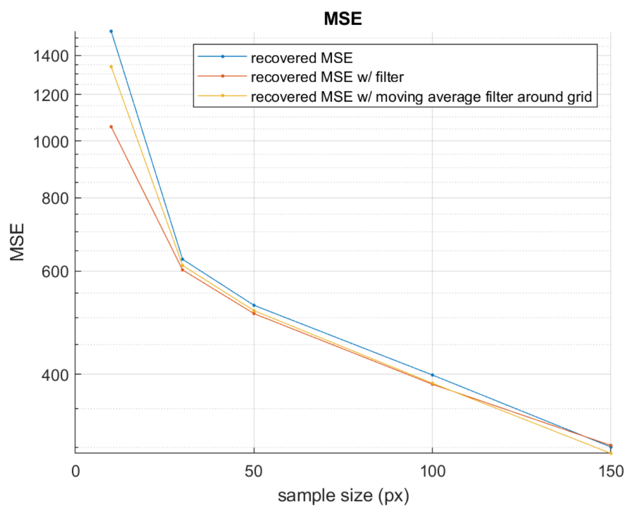

Fig 4. sample size vs. MSE

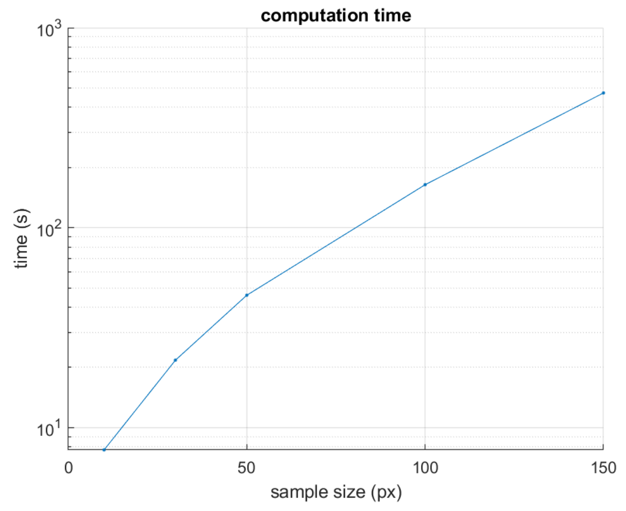

Fig 5. sample size vs. computation time

Sample size (px)	10	30	50	100	150
MSE of Recovered image	1.55E3	6.24E2	5.28E2	4.06E2	3.00E2
MSE of recovered image w/ filter	1.06E3	5.93E2	5.10E2	3.93E2	3.03E2
MSE of recovered image w/ moving average filter around grid	1.34E3	6.07E2	5.16E2	3.93E2	2.93E2
Computation time (s)	7.10	2.32E1	4.39E1	1.70E2	4.38E2

Table 2. sample size, MSE, and computation time for large test image

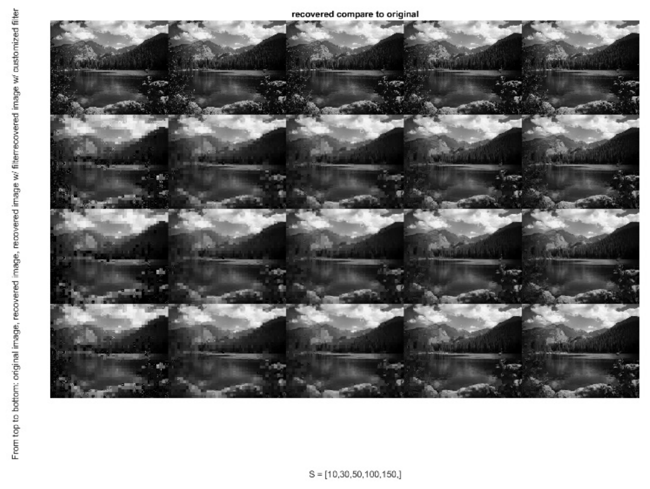
recovered compare to original
From top to bottom: original image, recovered image, recovered image w/ filterrecovered image w/ customized filter
S = [10,30,50,100,150,]

Fig 6. Recovered image vs. original image (w/ and w/o filter)

The entropy and MSE of each block are also computed and plotted for analysis as the following:

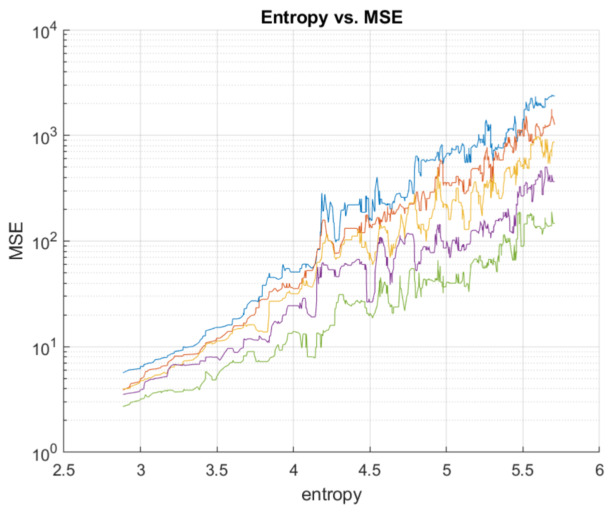

Fig 7. Entropy vs. moving mean of MSE with length of 20 for each block of small test image, color blue, red, yellow, purple, and green correspond to S_1, S_2, S_3, S_4, S_5

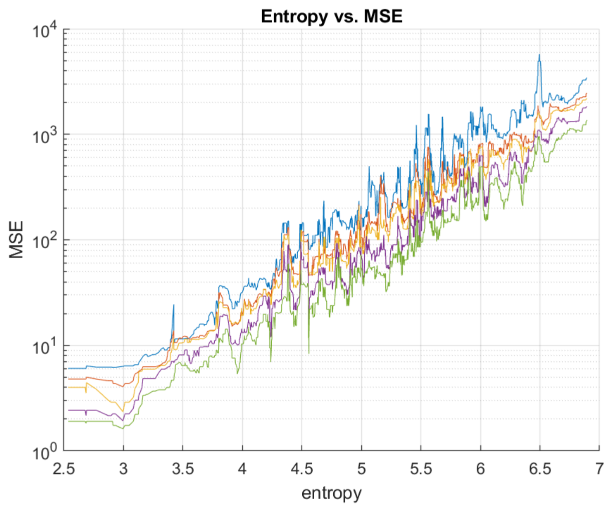

Fig 8. Entropy vs. moving mean of MSE with length of 20 for each block of large test image, color blue, red, yellow, purple, and green correspond to S_1, S_2, S_3, S_4, S_5

4. Discussion

It is easy to see from Fig 1, 2, 4, 5 that as the sample size increase, the computation cost would increase exponentially, and the MAE would decrease exponentially in general. Also, from Fig 7, 8, we can see that as the entropy of each block increase, the MSE increases exponentially and the trial with lower sample size would always have higher MSE in general. This also mean the more complex images (signals) are harder to recover from sparse samples.

Another observation is that the median filter worked well when the sample size is small since the badly recovered block often resembles something similar to the salt and pepper noise. But it starts to worsen as the sample size increase, since it would make the junction more obvious and blur the recovered the details.

Two of the improvements on the OMP algorithm were already mentioned in the previous section. One of the other improvements made throughout the code is to use matrix operation to substitute for loops whenever is possible. For example, the following for loop can be easily replaced by a matrix operation:

```
for i = 1:N
   c = c + a(i)*b(i)   ⟹   c = dot(a, b)
end
```

Another one is to use block processing to simplify the code and recover multiple blocks in parallel. This method in MATLAB would automatically divide the image to multiple K×K blocks, perform the same operation to them (compressed sensing in our case), and then be stitched together to form a complete image.

The last improvement is to create a mask on where the blocks are connected to each other, then apply a moving mean filter to it to smooth the junctions since they are more like steps functions rather than salt and pepper noise. This filter is useful when the sample size is large since it can smooth block junctions and largely preserve the recovered details.

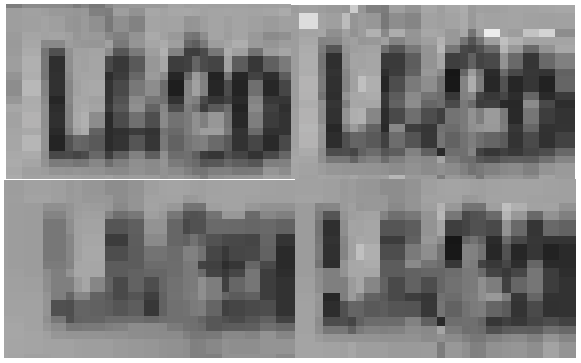

Fig 9. From top left to bottom right, examples of original vs. recovered vs. median filtered recovered vs. moving mean filtered recovered w/ mask image

References

[1] ECE 580 lecture slides by Dr. Xin, Li