# AMATH 582: HOMEWORK 4

CASSIA CAI

*School of Oceanography, University of Washington, Seattle, WA*
*fmc2855@uw.edu*

Abstract. Compressed image recovery is important especially in recent decades. JPEG compression is used to save bandwidth and storage space, dividing the image into non-overlapped blocks and quantized in the DCT domain. Sometimes, images are corrupted. Our goal is to recover the original image from the corrupted version. First, we apply image compression, investigating the compressibility of the discrete cosine transform (DCT) of the image. We then reconstruct the image after thresholding the DCT to keep the top 5, 10, 20, and 40 percent of DCT coefficients. Our next goal is to recover the image F from limited random observations of its pixels using the prior knowledge that the DCT(F) is nearly sparse. We use an random permutations of an identity matrix to select pixels (20, 40, and 60 percent of the total number of pixels) and use the CVX package for convex optimization. We apply our method for compressed image recovery to a mysterious image. The image is of the Nyan cat, which is a cat flying through outer space with a Pop-Tart body.

## 1. Introduction

When an original clean image is recovered from the corrupted version, this is image reconstruction or image recovery. An image can be corrupted in many ways such as noise, low resolution, or motion blur. The goal of this study is to recover an image from limited observations of its pixels. We consider a portion of René Magritte's "The Son of Man" along with its corrupted variant. We want to recover the original image from the corrupted version. The original high resolution image 292x228 is first downscaled and gray-scaled to 53x41. We first apply image compression to investigate the compressibility of the discrete cosine transform (DCT) of the image. We then apply compressed image recovery to recover the image F from limited random observations of its pixels using the prior knowledge that the DCT(F) is nearly sparse. Finally, we apply our compressed image recovery steps to recover a mysterious image, which we find later to be of the Nyan cat.

This study is relevant because oftentimes, images are corrupted or altered via the compression process. Image corruption exists in different levels of severity. The restoration task exists in varying levels of difficulty. An extension is to attempt to recover a corrupted image where the corruption varies (in type and severity) as well as test different methods to recover images such as dictionary learning and shape-adaptive DCT thresholding among others.

## 2. Theoretical Background

**Discrete Cosine Transform:** The Discrete Fourier Transform (DFT) is the Fourier Transform (FT) applied to a discrete signal.

$$f(x) \approx \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} c_k \exp(-\frac{ikRx}{L}), c_k \in \mathbb{C} \tag{1}$$

*Date*: March 17 2022.

The DFT of f(x) is: $\hat{f} := c_{-\frac{N}{2}}, ..., c_{\frac{N}{2}-1}) \in \mathbb{C}^N$. However, there are some disadvantages to using the DFT because of its complexity and poor energy compaction, which is defined as the ability to pack the spatial sequence energy into as few frequency coefficients as possible.

DCT is more efficient at energy compaction, which is important for image compression. Given a discrete signal $\mathbf{f} \in \mathbb{R}^K$ where

$$DCT(\mathbf{f})_k = \sqrt{\frac{1}{K}}[f_0 \cos(\frac{\pi k}{2K}) + \sqrt{2}\sum_{j=1}^{K-1} f_j \cos(\frac{\pi k(2j+1)}{2K})] \tag{2}$$

The DCT is taking the real part of the Fast Fourier Transform (FFT) of f, i.e., writing the signal as a sum of cosines. The inverse DCT (iDCT) transform reconstructs the signal f from DCT(f). The 2D DCT is defined analogously to 2D FFT by successively applying the 1D DCT to the rows and columns of a 2D image.

The FT is is used to represent a signal as a sum of sine and cosine functions [1]. The FT is defined as:

$$F(f) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp(-ikx) f(x) dx \tag{3}$$

and its inverse:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \exp(ikx) F(k) dx \tag{4}$$

The FFT computes the DFT quickly, converting a signal from a time or space domain into the frequency domain [2]. The FFT computes $\hat{f}$ for a function f(x) given a uniform mesh. The FFT computes the DFT quickly because it costs O(NlogN) operations to compute if N is the number samples in signal by discretizing the range $x \in [L, L]$ into $2^n$ points.

For a function $f : [0, 2L] \to \mathbb{R}$ where $f_n := f(x_n)$ for n=0,...,N-1. FFT returns coefficients:

$$\tilde{c}_K = \sum_{n=0}^{N-1} f_n \exp(-2\pi i \frac{nk}{N}) \tag{5}$$

Essentially, the DCT takes the real part of the FFT of f. The iDCT transform reconstructs the signal f from DCT(f).

**SPARSE RECOVERY:** Consider a linear system $A\underline{\beta} = \underline{Y}, \quad \underline{\beta} \in \mathbb{R}^N, \underline{Y} \in \mathbb{R}^M, A \in \mathbb{R}^{M \times N}$ where $M < N$. The system is underdetermined and cannot be solved uniquely [3, 4, 5]. The vector $\underline{\beta}$ is sparse with s non-zero entries.

For example, a vector $\underline{\beta}^+ \in \mathbb{R}^{100}$ with $s = 10$ nonzero entries and a matrix $A \in \mathbb{R}^{40 \times 100}$ with random entries $A_{ij} \overset{iid}{\sim} N(0, 1)$. Then $\underline{Y} = A\underline{\beta}^+ \in \mathbb{R}^{40}$. We want to infer $\underline{\beta}^+$ from $\underline{b}$.

**Lasso:** We try Lasso, which solves:

$$\underline{\beta}_{\text{lasso}} = \underline{\beta} \in \mathbb{R}^{100} \frac{1}{2}\|A\underline{\beta} - \underline{Y}\|_2^2 + \lambda\|\underline{\beta}\|_1 \tag{6}$$

This is different from Ridge because a different regularization term is used. Recall Ridge:

$$\underline{\beta}_{\text{Ridge}} = \frac{1}{2}\|A\underline{\beta} - \underline{Y}\|_2^2 + \lambda\|\underline{\beta}\|_2^2 \tag{7}$$

Lasso is effective because the $\| \bullet \|_1$-norm prefers sparse solutions whereas the $\| \bullet \|_2$-norm does not. Given sparsity, Lasso/1-norm regularization is preferable.

To summarize, Lasso is able to find a sparse solution to regression problems.

$$_{\beta}\|A\beta - Y\|^2 + \lambda\|\beta\|_1 \tag{8}$$

$$\hat{f}(\underline{x}) = \sum_{j=0}^{J-1} \hat{\beta}_j \psi_j(\underline{x}) \approx y(x) \tag{9}$$

If $\hat{\beta}$ is $s$-sparse, then $\hat{f}$ depends only on $s$ features. Because the DCT of natural images have sparse coefficients, we choose to Lasso, which favors sparse solutions.

**Convex Optimization:** To find $\beta$, we use convex optimization: minimize $f_0(x)$ subject to $f_i(x) \leq b_i, i =, ...m$ where $f_0, ..., f_m : \mathbf{R}^n \to \mathbf{R}$ are convex or satisfies $f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$ for all $x, y \in \mathbf{R}^n$ and all $\alpha, \beta \in \mathbf{R}$ with $\alpha + \beta = 1, \alpha \geq 0, \beta \geq 0$ [6].

## 3. Algorithm Implementation and Development

I coded Python and used NumPy and CVXPY for mathematical calculations and Matplotlib for visualizations [7, 8, 9]. We used the CVX package for convex optimization. Additional packages that were used are SciPy and scikit-image [10, 11]. Our goal is to recover the original image from the corrupted version.

**Image Compression:** First, we downsampled the original image (292x228) to (53, 41). This means that the resolution changes. We also converted the image to gray scale. We then vectorized the image using numpy's flatten function, constructed the DCT and iDCT. To get DCT(F), we used numpy's matrix multiplication function using as inputs the DCT and the vectorized image. Plotting DCT(F), we can investigate its compressibility. We then thresholded the DCT to keep the top 5, 10, 20, and 40 percent of its DCT coefficients by using numpy's percentile function.

**Compressed Image Recovery:** For compressed image recovery, we want to recover the image F from limited random observations of its pixels using the prior knowledge that the DCT(F) is nearly sparse. We define a function where we first create an identity matrix, randomly select M rows of the identity matrix using numpy's random permutation function and calling this B. We then define a vector of measurements $\mathbf{y} \in \mathbb{R}^M$ by applying B to the vectorized image. A new matrix A is defined as $A = BD^{-1}$. We then use CVX to the solve the optimization problem, minimizing in the l-1 norm with the constraint $\mathbf{Ax} = \mathbf{y}$. We then take $M = rN$ where $r = 0.2, 0.4, 0.6$ and repeat the calculation for each $r$ three times.

**A Mysterious Image:** Given some unknown image, we can follow the Compressed Image Recovery steps to recover the image from limited and indirect measurements by solving problem.

## 4. Computational Results

We first gray-scaled and downscaled the image (Figure 1).

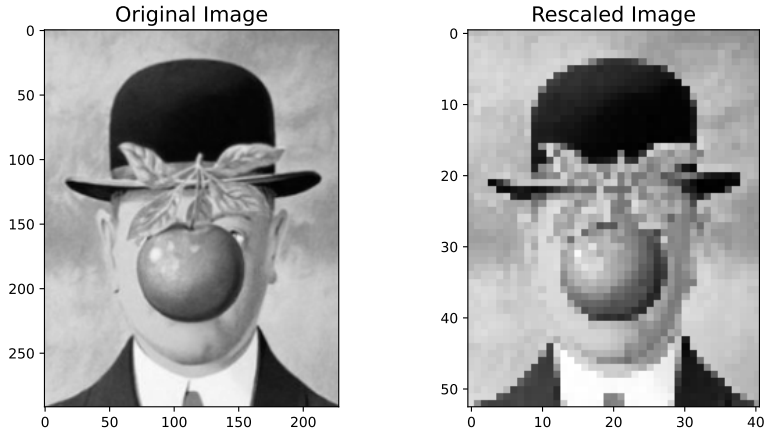


Figure 1. (Left) Original image 292x228. (Right) Rescaled image 53x41.

Then we plotted DCT(F) to investigate its compressibility. We see that there are some big coefficients but many small ones. This lets us know that we can use sparse image recovery (and use lasso, which favors sparse solutions) (Figure 2).

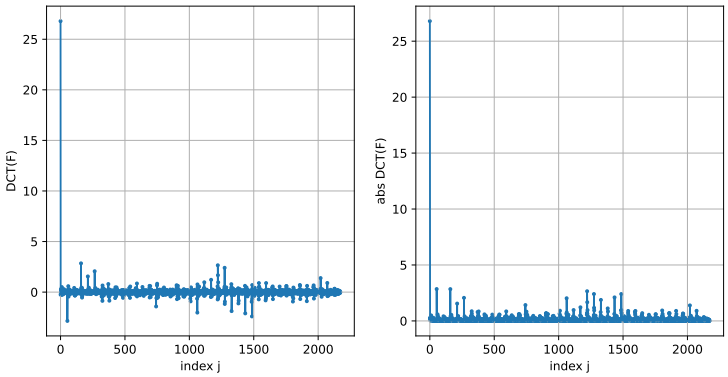


FIGURE 2. (Left) DCT(F) vs. index. (Right) Absolute value of DCT(F) vs. index.

We then plot the image after thresholding its DCT to keep the top 5, 10, 20, and 40 percent of DCT coefficients (Figure 3). Here, we see that the different percentages of DCT coefficients we keep results in varying levels of blurriness. As expected, when we keep more DCT coefficients, we are able to get a higher resolution image.

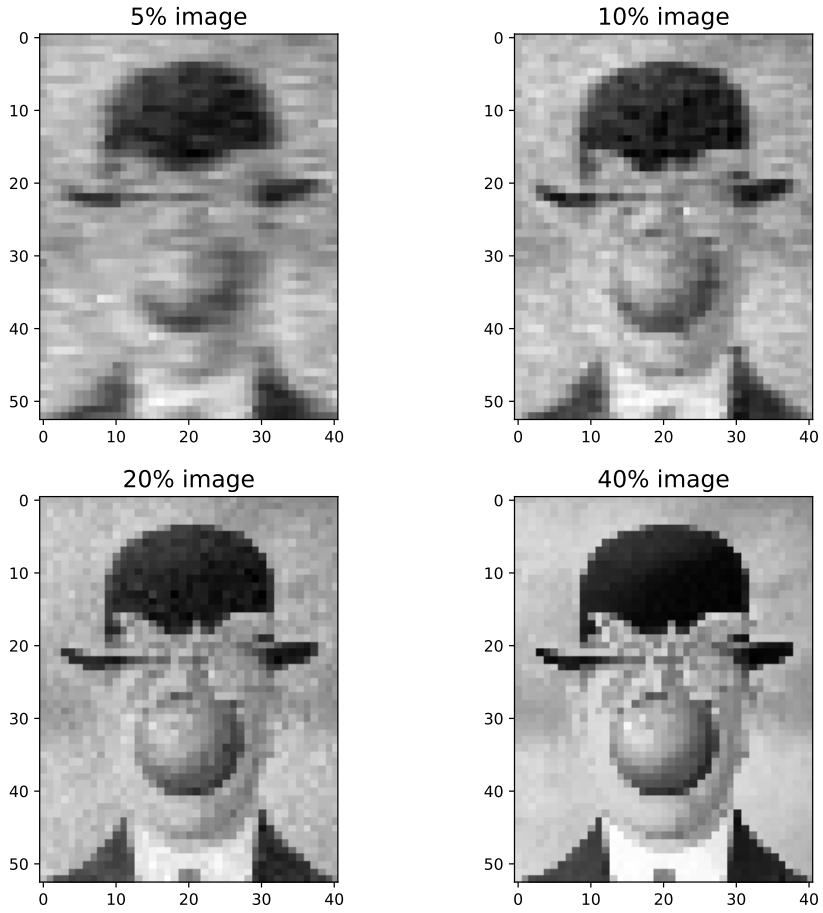


FIGURE 3. Image compression after thresholding its DCT to keep the top 5, 10, 20, and 40 percent of DCT coefficients.

We then apply compressed image recovery keeping varying percentages (M = 0.2, 0.4, and 0.6) of the total pixels to reconstruct the image. We use convex optimization to solve. Each M was iterated three times. For each iteration, we plot the reconstructed image. As expected, there are slight differences with each iteration because we randomly permuted the identity matrix to select pixels (Figure 4). If we had kept even more pixels, we could have generated a more high accurate image. This is useful in to show that if we have access to some pixels, we are still able to generate an image. If we only have a few pixels, the reconstruction task is very difficult.

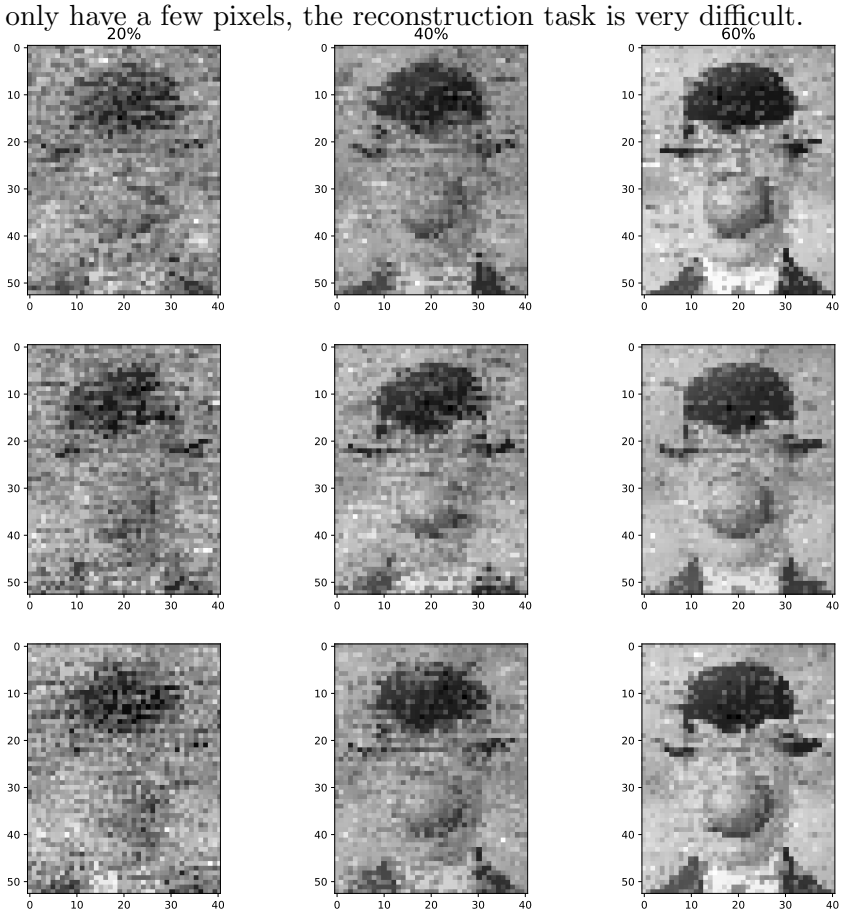


FIGURE 4. Compressed image recovery keeping different percentages of the number of pixels.

Finally, we apply our steps for compressed image recovery to recover a mysterious image. We see that the secret image is that of the Nyan cat, which became an internet meme after it was uploaded to Youtube in April 2011. It is a cartoon cat with a Pop-Tart body flying through outerspace with a rainbow trail (Figure 5).

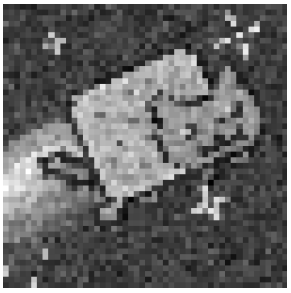

FIGURE 5. The secret image is of the Nyan cat.

## 5. Summary and Conclusions

We recovered the original "Son of Man" image from the corrupted version. Due to computation limits, we downscaled the original image. First, we applied image compression. We investigated the compressibility of the DCT of the image. We then reconstructed the image. To reconstruct the image, we thresholded the DCT to keep the top top 5, 10, 20, and 40 percent of DCT coefficients. We then recovered the image F from some limited random observations of its pixels assuming the the DCT of the image is nearly sparse. We used random permutations of an identity matrix to select pixels and then used convex optimization. We found that for each iteration of convex optimization, the reconstructed image varied slightly. This is not surprising because to pick pixels, we used random permutations of the identity matrix. This means that we are not picking the same pixels each time. We then applied this method for compressed image recovery to a mysterious image. The secret image is of the Nyan cat. An extension to this would be to test different image recovery methods.

## Acknowledgements and Code Availability Statement

The author is thankful for the algorithm-implementation suggestions from peers in AMATH 482-582. The code is publicly available on this Github repository after the submission date.

## References

[1] Steve Brunton and Nathan Kutz. *Data Driven Science  Engineering*. databook.uw.edu, 2017.
[2] Bamdad Hosseini. Signal processing with dft. University of Washington (LOW 216), Jan 2022. AMATH 482/582.
[3] Bamdad Hosseini. Introduction to sparse recovery. University of Washington (LOW 216), March 2022. AMATH 482/582.
[4] Bamdad Hosseini. Model selection with lasso. University of Washington (LOW 216), March 2022. AMATH 482/582.
[5] Bamdad Hosseini. Sparse signal image/recovery. University of Washington (LOW 216), March 2022. AMATH 482/582.
[6] Steven Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
[7] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
[8] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.
[9] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.
[10] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
[11] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.