AMATH 482/582: HOME WORK 5

YIXUAN LIU

University of Washington, Seattle, WA
yixuanl@uw.edu

Abstract. In this project, we resize the image of "The Son of Man". We compress the resized image with different values of a parameter by discrete cosine transform to test its compressibility. Based on the resized image, we randomly choose pixels and use `CVX` package to recover the corrupted image by the chosen pixels. Then we use the same method to recover an unknown image.

1. Introduction and Overview

We are given the original image of "The Son of Man", and rescale to lower resolution to make it easier to compute. With the rescaled image, we compressed it by discrete cosine transform with 4 different values of a parameter to reconstruct 4 compressed images. Then we use `CVX` to try to recover the image with only partial pixels chosen randomly from the resized image we did previously. With different percentages of pixels. We get different visualizations of the recovered image. With the same method, we then recover an unknown image with partial measurements.

2. Theoretical Background

Consider a linear system

$$A\underline{\beta} = \underline{y}, \underline{\beta} \in \mathbb{R}^N, \underline{y} \in \mathbb{R}^M, A \in \mathbb{R}^{M\times N}$$

where $M < N$, so the system is under determined so it cannot be solve uniquely. However, suppose that we have additional information that the vector $\underline{\beta}$ is sparse, more precisely it only has s-non-zero entries (we say $\underline{\beta}$ is s-sparse) [5].

To solve the system, we will use Lasso which solves

$$\underline{\beta}_{lasso} = \arg\min_{\underline{\beta}\in\mathbb{R}^N} \frac{1}{2}||A\underline{\beta} - \underline{y}||_2^2 + \lambda||\underline{\beta}||_1$$

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

If $\hat{\beta}$ is s-sparse, then only s entries in $\hat{\beta}$ are non-zero. so $\hat{f}$ is particularly simple and depends only on s features [6].

Date: March 17, 2022.

The Discrete cosine transform (DCT) is analogous to taking the real part of the Fast Fourier Transform (FFT) of a discrete signal, which is an algorithm that compute Fourier Transform for a function $f(x)$ given a set of equidistance points [7]: let $f_n := f(x_n)$ for $n = 0, ..., N-1$, the FFT returen coefficients

$$\hat{c_k} = \sum_{n=0}^{N-1} f_n e^{-2\pi i \frac{xk}{N}}$$

Given a discrete signal $\mathtt{f} \in \mathbb{R}^K$ we define its DCT as $\mathrm{DCT}(\mathtt{f}) \in \mathbb{R}^K$ [4] where

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

The `CVXPY` is a Python-embedded modeling language to solve convex optimization problems [2].

3. Algorithm Implementation and Development

We will use

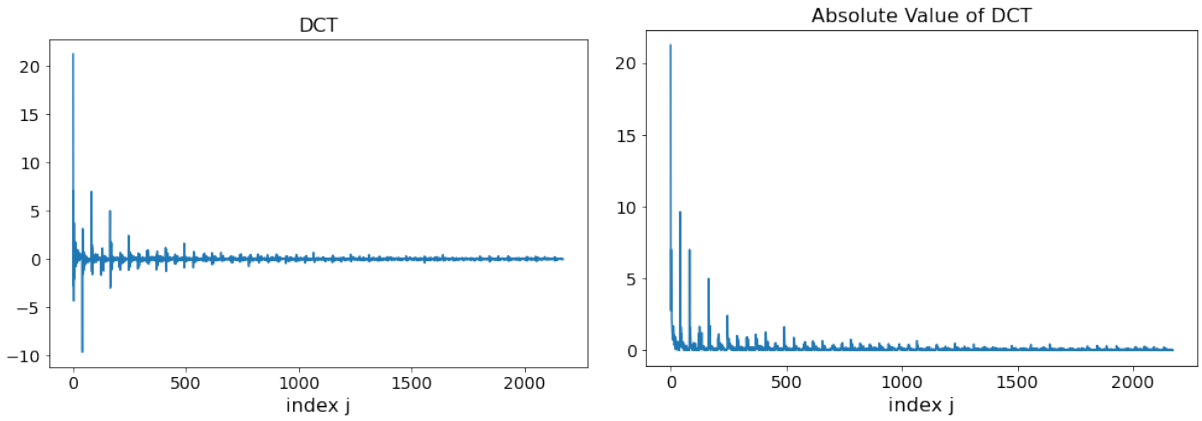
- `numpy` [3] for mathematical operations
- `matplotlib.pyplot` [8] to plot graph
- `cvxpy` [1] to solve convex optimization problems
- `skimage` [9] to process the original image
- `scipy.fftpack` [10] to implement DCT

4. Computational Results

Given with the original image of "The Son of Man", we rescale it from a 292×228 image to a 53×41 image as shown:

Figure 1. The original and rescaled image of "The Son of Man"

By flattening the rescaled image, we get a vec(F)$\in \mathbb{R}^{53\times 41}$ to do the discrete cosine transform. Apply the forward DCT matrice on vec(F), we can get vec(DCT(F)). Plot DCT(F) and abs(DCT(F)):



(A) (B)

FIGURE 2

We can see from the plot that the discrete cosine transform of the image is sparse since there are not many large coefficients, so we can use Lasso to recover the corrupted image. Now we take top 5%, 10%, 20% and 40% coefficients of abs(DCT(F)) separately and zeros all the others to compress the image. By applying inverse DCT on the reconstructed DCT(F), we get the following compressed images corresponding to the four percentages:

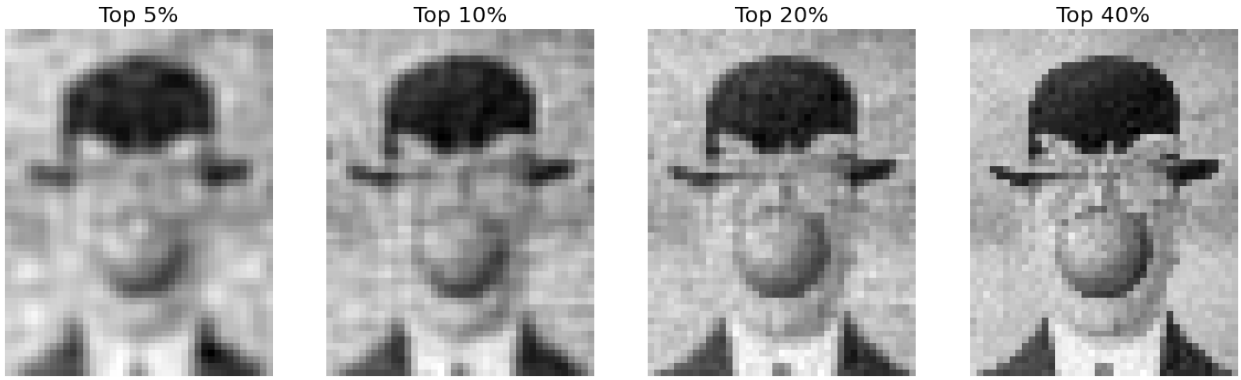


FIGURE 3. The compressed images with four different measurements

Now we want to randomly pick pixels from the image with the percentages of 20%, 40% and 60%. With DCT, we use `CVX` to solve for recovery of the corrupted images. For each percentage, we corrupt randomly three times and recover each of them to avoid extreme cases while using random numbers:

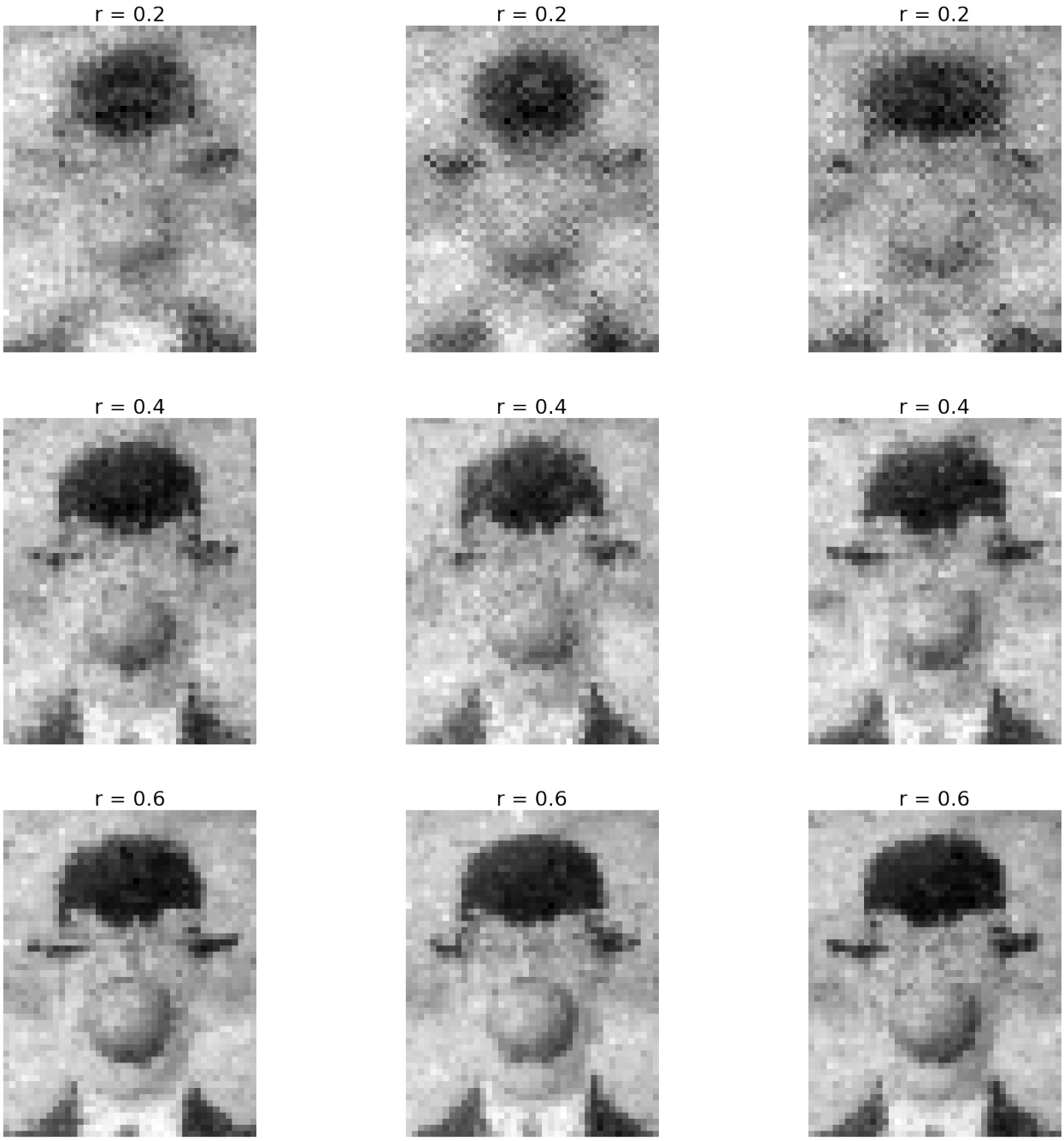


FIGURE 4. The recovered images with 9 different measurements

With percentage to be 20%, we can barely see that the recovery is similar to the rescaled image, with 60%, we can distinguish the recovery is a "noised" image of "The Son of Man", and with 40%, the recovery is still not clear enough to tell if it's "The Son of Man".
With the method to recover images, we can now recover the unknown image given:

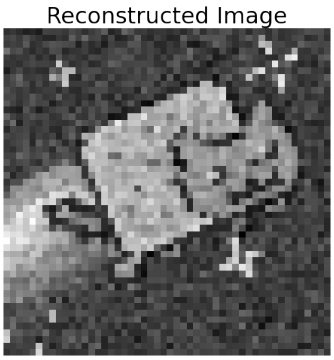


FIGURE 5. The recovered images of the unknown image

We can see it's the Nyan Cat.

5. Summary and Conclusions

By implementing DCT and CVX, we are able to compress images and recover images from limited observations of its pixels. From this project, we can see that the image is sparse after applying with DCT, so we can use large coefficients to represent the image.

Acknowledgements

The author is thankful to every thought in the AMATH 582/482 Discord server, and Professor Hosseini for the code in the helper notebook of rescaling the original image and DCT functions.

References

[1] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[2] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[3] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[4] B. Hosseini. Hw5: Compressed image recovery. University of Washington (LOW 216), Feb 2022.

[5] B. Hosseini. Introduction to sparse recovery. University of Washington (LOW 216), Feb 2022.

[6] B. Hosseini. Model selection with lasso. University of Washington (LOW 216), Mar 2022.

[7] B. Hosseini. Signal processing with dft. University of Washington (LOW 216), Jan 2022.

[8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[9] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[10] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.