AMATH 482/582: HOME WORK 5

SHENGBO JIN

CFRM Program, University of Washington, Seattle, WA
Shengboj@uw.edu

Abstract. This report aims at researching image compression with the discrete cosine transform (DCT) and image recovery with convex optimization. Firstly, in image compression part, investigate the compressibility of DCT by thresholding DCT coefficients with different percentages. Then, by convex optimization for image recovery, compare the performance with different sampling ratios. At last, use the same method to recover a mysterious image.

1. Introduction and Overview

Our goal is to recover the original image from the corrupted version with convex optimization. The original image is actually a rescaled one with 53×41 pixels, which is from a portion of René Magritte's "The Son of Man". In addition to image recovery, the discrete cosine transform (DCT) is also used to investigate the compressibility.

In this report, under Python environment, Skimage[4] is used for reading image, and image compression and image recovery are realized by Numpy[2], Cvxpy[1], and Scipy[5]. Matplotlib[3] is used for visualization.

2. Theoretical Background

Discrete Cosine Transform (DCT)

Given a discrete signal $\mathbf{f} \in \mathbb{R}^K$ we define its DCT as DCT($\mathbf{f}$)$\in \mathbb{R}^K$ where

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

This transform is analogous to taking the real part of the FFT of $\mathbf{f}$, i.e., writing the signal as a sum of cosines (other definations of DCT exist in the literature). The inverse DCT (iDCT) transform reconstructs the signal $\mathbf{f}$ from DCT($\mathbf{f}$). The 2D DCT (resp. iDCT) is defined analogously to 2D FFT (resp. iFFT) by successively applying the 1D DCT (resp. iDCT) to the rows and columns of a 2D image.

Signal Recovery

Consider a signal(or image) $f^+ : [0, 1] \to \mathbb{R}$ along with some measurements of the signal $\mathbf{y} \in \mathbb{R}^M$. A simple example is

$$\mathbf{y}^+ = (y_0^+, .., y_{N-1}^+)$$

where $\mathbf{y}_j^+ = f^+(t_j)$ for some points $t_j \in [0, 1]$. Of course we can simply solve this problem using for ex, polynomial interpolation. But we want to consider a harder problem specifically one where the measurements $\mathbf{y}^+ \in \mathbb{R}^M$ are of the the form:

$$y_j^+ = \xi_j^T f^+$$

Date: March 11, 2022.

where $\xi_j \sim \mathcal{N}(0, I),\ f^+ = (f^+(t_0), f^+(t_1), .., f^+(t_{N-1}))$.

Goal: reconstruct f^+ given $\mathbf{y}$ & $M < N$ Our first step is to assume a model for $\hat{f}$

$$f(t) = \sum_{m,n} \beta_{m,n} \Psi_{m,n}(t)$$

where $\Psi_{m,n}$ are the Haar wavelet basis. Reparametraize the wavelet coefficients into a 1D array, $f(t) = \sum_{j=0}^{J-1} \beta_j \psi_j(t)$ such that $\mathbf{f} = W\beta$. Finally for such an $\mathbf{f}$ we have the measurements

$$\mathbf{y} = S\mathbf{f},\ S = \begin{bmatrix} \xi_0^T \\ \xi_1^T \\ .. \\ \xi_{M-1}^T \end{bmatrix} \in \mathbb{R}^{M \times N}$$

In other words,

$$\mathbf{y} = A\beta \ where\ A = SW$$

Lasso

Lasso regularization can be extended to other objective functions such as those for generalized linear models, generalized estimating equations, proportional hazards models, and M-estimators. Given the objective function

$\frac{1}{N} \sum_{i=1}^{N} f(x_i, y_i, \alpha, \beta)$

the lasso regularized version of the estimator "s" the solution to

$\min_{\alpha,\beta} \frac{1}{N} \sum_{i=1}^{N} f(x_i, y_i, \alpha, \beta)$ subject to $\|\beta\|_1 \leq t$

where only β is penalized while α is free to take any allowed value, just as β_0 was not penalized in the basic case.

3. Algorithm Implementation and Development

Step 1 Image compression

1. Use *ski.io.imread()* and *ski.color.rbg2gray()* to load the original image, and use *ski.transform.rescale* to create rescaled one. Then, visualize them by *plt.imshow()*.
2. Define *construct_DCT_Mat()* and *construct_iDCT_Mat()* with *spfft.dct()*, *spfft.idct()* and *np.kron()*. Then, use them to construct the forward and inverse DCT matrices D and D^{-1} for the image. Create Vec(F) by *flatten()* with rescaled image. Finally, obtain vec(DCF(F)) through multiplying D and vec(F) by *np.dot()*.
3. Take absolute value and plot DCT(F) with *np.abs()* and *plt.plot()*.
4. Separately keep the top 5, 10, 20, and 40 percent of DCT coefficients by *np.percentile()* and *np.where()*, then reconstruct image by multiplying D^{-1} and the thresholded DCT with *np.dot()*. Finally, visualize them by *plt.imshow()*.

Step 2 Compressed Image Recovery

1. Define *construct_Measurement_Mat()* with *np.identity()* and *np.random.permutation()*, and create a measurement matrix B with M and N (N= len(vec(F))). Then, Plot B with *plt.imshow()* to show which pixels are used.
2. With *np.dot()*, generate a vector of measurements y by multiplying B and vec(F), and a matrix A by multiplying B and D^{-1}. Then, use CVX to solve the optimization problem to find minimizer as x^*. Finally, recover image through multiplying D^{-1} and x^* with *np.dot()* and plot it with *plt.imshow()*.
3. Take M = r × N for r = 0.2, 0.4, 0.6, and for each choice of M perform **Step2.1-2.2** three times.

Step 3 A Mysterious Image

Use *np.load()* to load the measurement vector y and the measurement matrix B, then follow the same method in step2 to recover the image and visualize it.

4. Computational Results

Task 1

Figure 1. original image and rescaled image

The above figure shows the original image and rescaled image, and the latter one is the main research object.

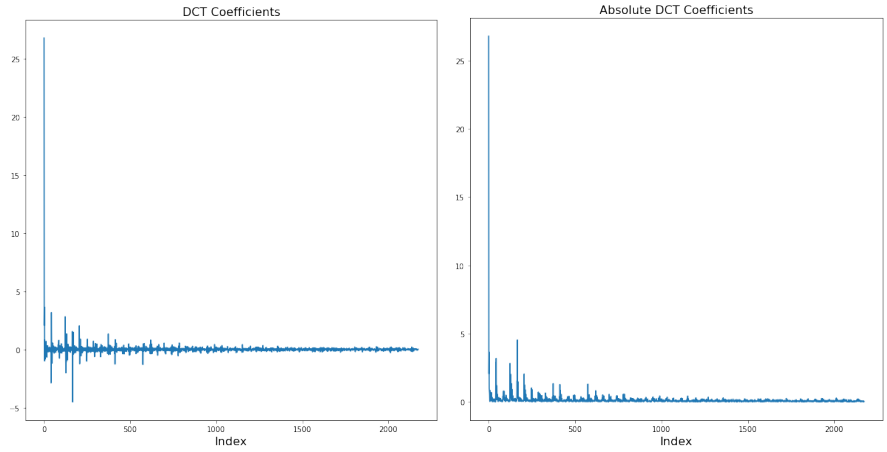

Figure 2. DCT coefficients and absolute ones of the image

From Figure2, at some certain low frequencies, there are a lot of DCT coefficients much large than others.

Keep the top 5, 10, 20, and 40 percent of DCT coefficients, then reconstruct the image by the inverse DCT transform. The reconstructed images:

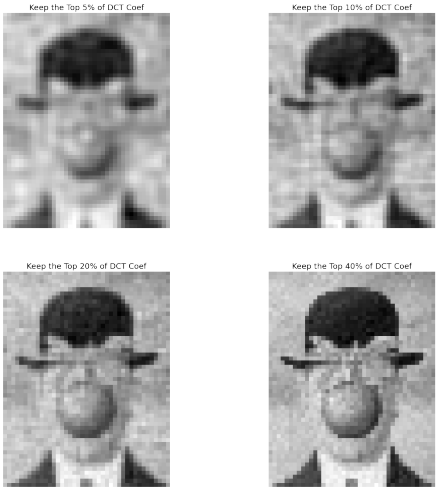

FIGURE 3. Compressed image

By keeping more DCT coefficients, the compressed image does become much clearer. The basic outline is well-captured in the compression, which shows a distinction of black and white, especially in the hat and the apple. However, there is a kind of white noise appearing in the compressed image, which makes the image much grey.

Task 2

For each figure in the followings, the above part shows which pixels are measured with white entries while black squares in place of the excluded pixels, and the below one shows the performance of image recovery.

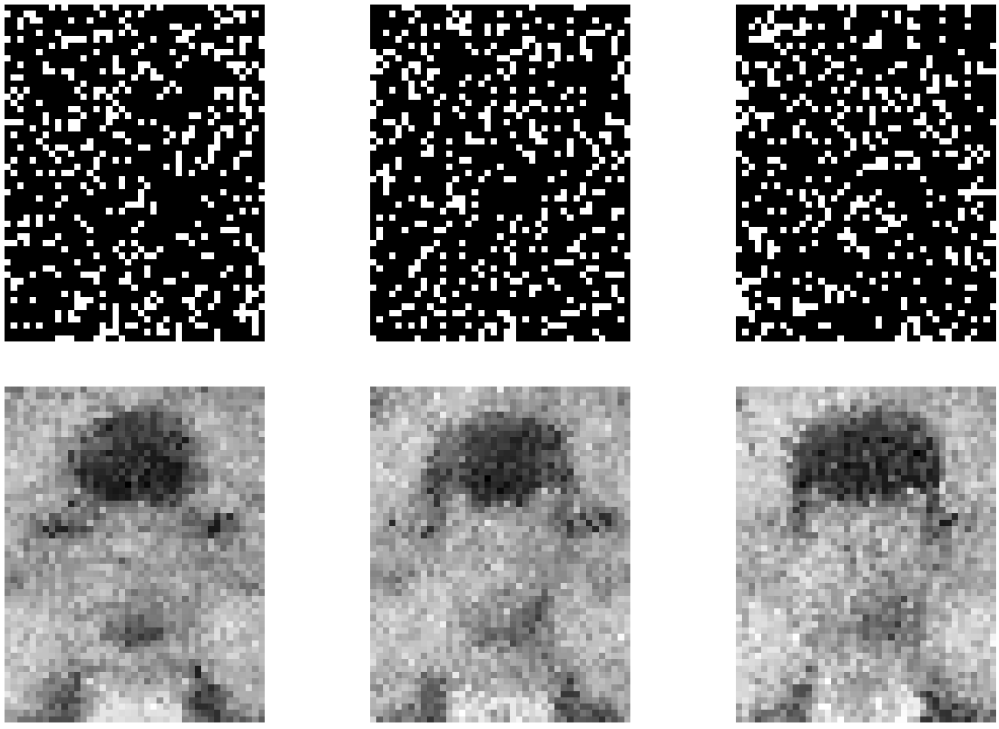

FIGURE 4. M = $0.2 \times N$

FIGURE 5. M = $0.4 \times N$

FIGURE 6. M = $0.6 \times N$

With the increase of the sampling ratio, the recovery image becomes much more approximate to the original image. In addition, the randomness of sampling has a great effect on the recovery. For the same sampling ratio, the recovery images can be very different, especially in the low ratio.

Task 3

It is Nyan Cat!

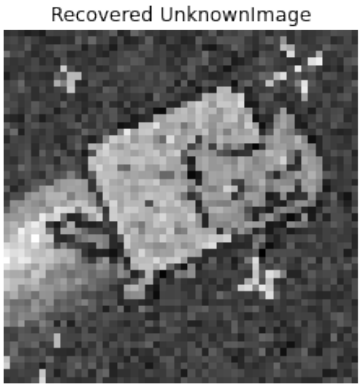

FIGURE 7. Nyan Cat

5. Summary and Conclusions

Firstly, in the image compression part, with the application of DCT, although the basic outline of compressed image is kept, a kind of white noise is likely to appear, which blurs the image. If further denoising can be realized with some kind of method, the performance of compression will be better. For image recovery, the randomness of sampling is likely to greatly influence the final recovery image, but this effect is milder with higher sampling ratio. When we do image recovery with seriously corrupted image, the character of final image may be very different from the true. Considering the expensive calculation, the above research is pretty simple, further detailed work needs to be done.

Acknowledgements

The author is thankful to Prof. Hosseini for lectures about image processing. I am also thankful to Dr. Owens and Dr. Levin for careful guidance about the report. Furthermore, my peer Xuqing Hu is helpful in the use of Latex.

References

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

[2] 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.

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

[4] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[5] 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.