AMATH 482/582: HOMEWORK 5

JONATHAN BEAUBIEN

Applied Mathematics Department, University of Washington, Seattle, WA
beaubj@uw.edu

Abstract. We have been tasked with compressing and recovering corrupted versions of René Magritte's "The Son of Man". We found success using the Discrete Cosine Transform (DCT), and inverse DCT, to compress the image by 90%. Our next technique used a form of lasso regression on corrupted versions of the image to recover the original image having success when the corrupted image was missing $\approx 60\%$ of its original data. Lastly, we were able to recover an unknown image using our image recovery code developed in the previous section.

1. Introduction and Overview

The first task will be to compress René Magritte's "The Son of Man" and trying to reconstructing it from versions of the image that are corrupted. The original image is in color and with a resolution of (292, 228). Because many of the computations being done are intensive, we will downscale the original image to a resolution of (53, 41). Here is what that downscaling looks like:

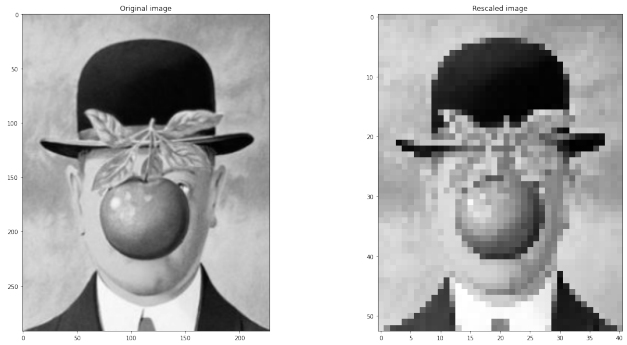


Figure 1. The Son of Man downsized from (292, 228) to (53, 41) to save computation time.

Our first technique will be to use the Discrete Cosine Transform (DCT) to help compress our image. We can represent the image as a vector and then take the DCT of the resulting vector by treating it as a signal. After investigating the distribution of coefficients in the DCT of the image, a variable threshold can be used to keep only a certain percentage of the DCT's coefficients. Then we can take the inverse DCT to return to a compressed image and see our results.

Next, we will investigate the recoverability of a corrupted version of our image. By removing a certain proportion of random pixels from the picture, a version of lasso regression can be used on the inverse DCT to find a recovered image that should hopefully resemble the original.

Lastly, we will recover a mysterious, unknown image with our previously generated image recovery code given to us as a fun extra task.

Date: March 18, 2022.

2. Theoretical Background

Our first order of business is defining the Discrete Cosine Transform. If we are given a discrete signal $f \in \mathbb{R}^K$, we can define its DCT as $\text{DCT}(f) \in \mathbb{R}^K$ where ([3])

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

Although this is the 1D DCT, the 1D DCT can be applied to the rows and columns of our 2D image to construct a 2D DCT. This is the same process that will be used for finding the 2D inverse DCT.

For our purposes, the image can be reshaped into a 1D vector denoted by $F \in \mathbb{R}^{N_x \times N_y}$ where N_x, N_y are the row and column lengths (in pixels) of the image. Thus, the number of pixels in the image is $N = N_x \times N_y$. We denote the DCT and iDCT as the matrices $D, D^{-1} \in \mathbb{R}^N$ such that $D\text{vec}(F) = \text{vec}(\text{DCT}(F))$ and $D^{-1}\text{vec}(\text{DCT}(F))$ where $\text{vec}(F) \in \mathbb{R}^N$ [3].

We expect the DCT of our image to be sparse as it is for most real images. That means there are a small number of coefficients in our DCT that have a large magnitude while most are not as important. This is important because our image is only compressible if we can threshold the DCT by using only the top 5, 10, 20, and 40% largest coefficients. [4]

For our image recovery task, the corrupted image will be defined in relation to the original where we let $M < N$ be an integer where matrix $B \in \mathbb{R}^{M \times N}$ by selecting M random rows of the identity matrix $I \in \mathbb{R}^{N \times N}$ [3].

Next, we define our lasso regression problem where our measurements vector $y \in \mathbb{R}^M$ is denoted by $y = B\text{vec}(F)$. We also define the matrix $A = BD^{-1} \in \mathbb{R}^{M \times N}$. Our regression is defined by [3]

$$\begin{aligned} &\underset{x \in \mathbb{R}^N}{\text{minimize}} \, ||x||_1 \\ &\text{where } Ax = y \end{aligned} \tag{2}$$

The vector resulting in optimizing these conditions with be denoted x^* and will be the DCT vector of an image F^* that will be our recovered image. To test varying levels of corruption in our image, we can set M to be a proportion of pixels to remove defined by $M = r \times N$ for $r \in \{0.2, 0.4, 0.6\}$.

3. Algorithm Implementation and Development

The code for constructing the DCT and iDCT matrixes (D, D^{-1}) were provided by Prof. Bamdad and use SciPy's *fftpack.dct()* and *fftpack.idct()* functions [6].

Processing the image data and array/matrix manipulation was done in conjunction with scikit-image [5] and NumPy [2].

Lastly, we used CVXPY and CVXOPT [1] that helps with convex optimization problems. We used this for the lasso regression in reconstructing the corrupted image.

4. Computational Results

The first computational result we will share is the confirmation of the sparsity of the DCT of our image. Here is a plot of the magnitudes (absolute value) of our DCT coefficients 2.

Thankfully, sparsity is our friend and lets us compress the image with a relatively loss of interpretability. As discussed in our Theory section (2) we will vary the threshold for what proportion of DCT coefficients will be saved (ordered by magnitude) denoted by p. For example, if $p = 5$ then we will save the DCT coefficients that are in the top 5% with regard to magnitude. After that, we will apply the iDCT matrix to return to image space with a compressed version of our image. Here is a plot of those images with varying levels of p (image 3).

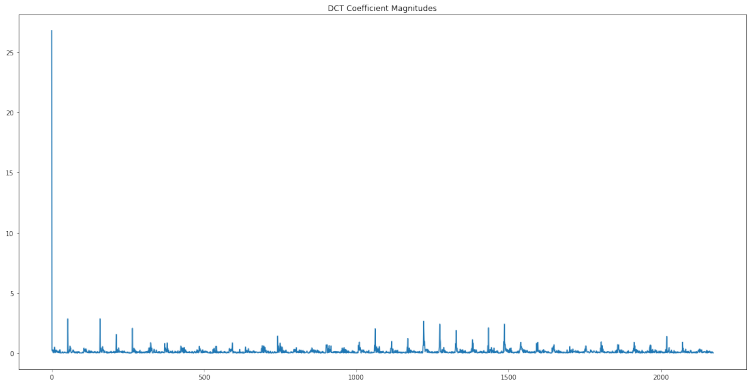


FIGURE 2. Our DCT is very sparse. There are a few coefficients with large magnitudes while most are small.

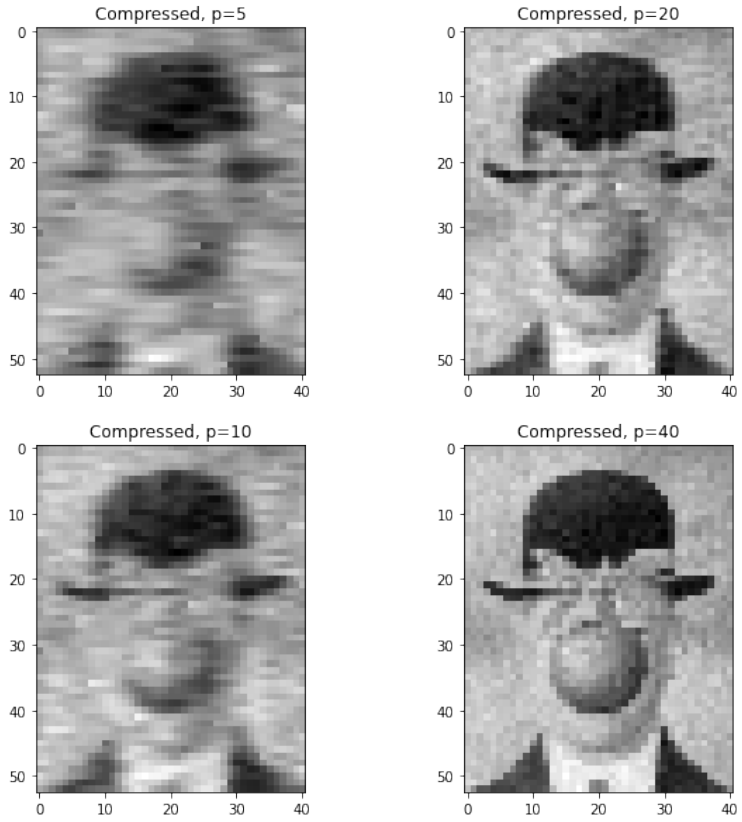


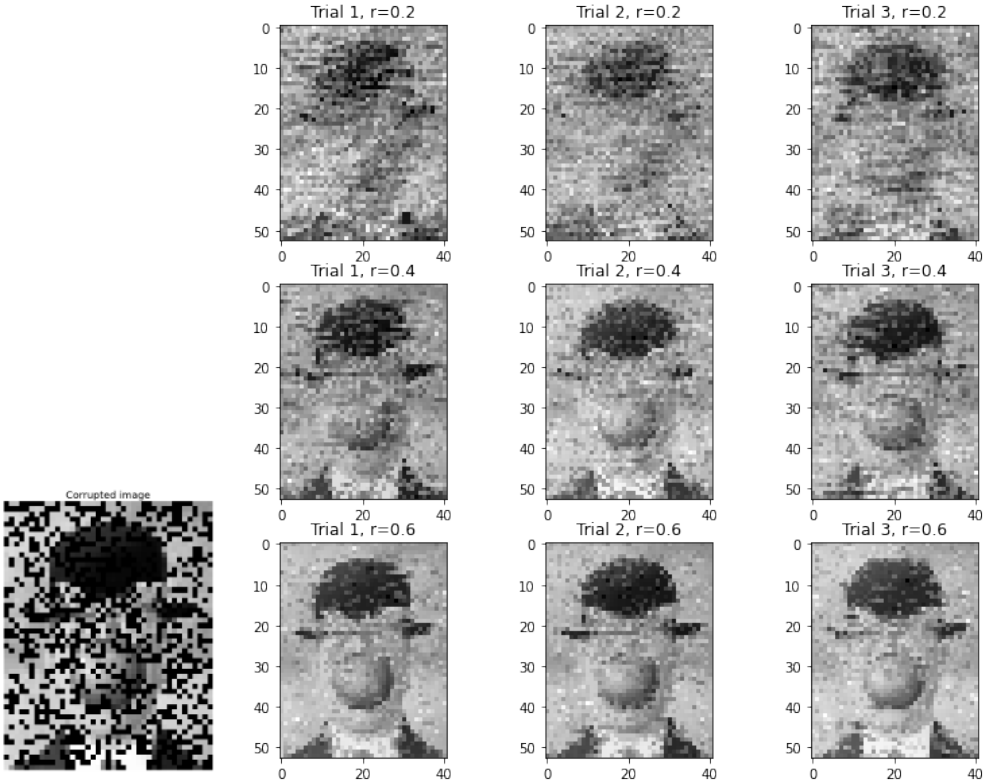
FIGURE 3. We tried various values of p including 5, 10, 20, and 40. As we can see, the more coefficients we use (the larger p is) the smaller the difference between our compressed image and the original.

We can see that even with only the top 5% of DCT coefficients, we can still make out a man with a hat and something in front of his face. This proves that we can obtain a lossy but accurate compression using DCT.

Next, we will investigate recovering a corrupted version of the Son of Man image. Here is an example of what a corrupted version would look like (image 4a).

The proportion of black pixels (i.e. corrupted pixels) in our image will be our value for r where $M = r \times N$ as described in section 2. We will try values for $r \in \{0.2, 0.4, 0.6\}$. Because our matrix B is a random selection of the rows of the identity matrix, there will be 3 trials for each value of r to make sure we aren't getting lucky or unlucky with the pixels that are corrupted (image 4b).

CVXPY took around 45 seconds to find an optimal x^* vector that satisfied the conditions in 2.



(A) An example corrupted image.

(B) An array of reconstructed images from various levels of corruption with 3 randomized trials for each level.

Now that we have code that can reconstruct an image from limited information we are given an unknown image to try to reconstruct. We are given a measurement vector y and the measurement matrix B as defined in section 3. Here is the image that we recovered (image 5).

5. Summary and Conclusions

We were able to compress the Son of Man by applying a Discrete Cosine Transformation, picking the largest DCT coefficients, and applying the inverse DCT to return to a compressed image. We found high success in compressing the image even with only the top 5 and 10% DCT coefficients. Next, we solved a lasso regression problem on corrupted versions of the image (with varying levels of corruption) to retrieve our best attempt at reconstructing the original image. We saw interpretability fall when we reached $\approx 60\%$ corruption. Lastly, we were able to recover a mysterious corrupted image that turned out to be Nyan cat.

Acknowledgements

The author is thankful to Professor Bamdad for his helpful Python notebooks on using CVXPY and constructing DCT and iDCT matrices. The author is also thankful to the AMATH 482/582 Discord channel for help regarding various NumPy operations.

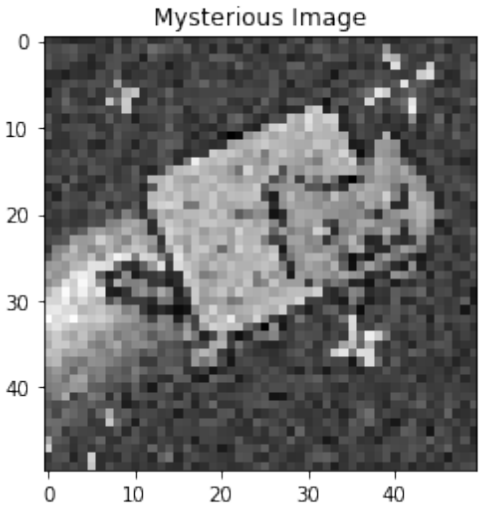


FIGURE 5. It's Nyan Cat!!

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] B. Hosseini. Homework 5: Compressed image recovery. University of Washington (LOW 216), Feb 2022.

[4] B. Hosseini. Sparse signal image/recovery. University of Washington (LOW 216), Feb 2022.

[5] 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.

[6] 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.