AMATH 482/582: HOMEWORK 5

SATHVIK CHINTA

Abstract. Using the discrete cosine transform and solving optimization problems, we are able to reconstruct corrupter (and mysterious) images to a very high degree of clarity.

1. Introduction and Overview

Imagine we are given an image that is corrupted. Can we reconstruct said image using what we have learned so far? In our effort to do so, we will go back to Fourier Transforms and apply them to the image. However, instead of regular Fourier transforms, we will instead be using the Discrete Cosine Transform.

We will first identify the compressibility of the image using the DCT. Then, we will attempt to recover the corrupted image using the inverse DCT. Finally, we will apply all methods to find out what the mystery image we are given is.

2. Theoretical Background

We will first take the discrete cosine transform of our image. The discrete cosine transform is a fast way to compute the Fourier transform of a discrete signal. The discrete cosine transform is very similar to the discrete Fourier transform, but we only use real numbers.

So, where DFT will use both sine and cosine waves, the discrete cosine transform only breaks the signal up into cosine waves. We can define the DCT of a real signal $f \in \mathbb{R}^K$ as

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

Let's take our SonOfMan image and then investigat the compressibility of its Discrete Cosine Transform. This just involves us passing the pixel values (flattened) into the equation above. We can then investigate the size of the coefficients. If there are not many large coefficients, then the image is very compressible.

Once we do that, we can reconstruct the image after keeping a certain amount of the coefficeints, and we can extrapolate what we learn to figure out if we can recover the original image or not.

From the original image, we can then attempt to solve the optimization problem

$$\text{minimize}_{x \in \mathbb{R}^N} ||x||_1$$
$$\text{Subject to} Ax = y$$

Where x is the DCT vector of our corrupted image, y is our vector of measurements, and A is constructed using a combination of our inverse DCT and random rows of the identity matrix (of dimension N where N is the number of pixels in our image).

Date: March 18, 2022.

3. Algorithm Implementation and Development

I used Python in conjunction with NumPy, Matplotlib, cvxpy, and scipy

I first used the scipy.spfft.dct() function to compute the DCT matrix, and then used the scipy.spfft.idct() to compute the inverse DCT matrix. Then I took the dot product between my image (flattened) and the DCT matrix to get the DCT coefficients (stored as DCT_F).

After looking at the DCT coefficients, I used the np.percentile() function to get the threshold value for each percentage within the array [5, 10, 20, 40]. Then I looped through my DCT_F value and set all values (absolute) below the threshold to 0. Once this was done, I applied the inverse DCT to the resulting array to get the image with only a certain percentage of the coefficients.

To reconstruct the corrupt image, I first used the np.random.percentile() function on the identity matrix (with dimensionality equivalent to my image size), and took the first M rows (percentage of my image size) to compute my M matrix. I then found the y value by multiplying my my flattened image, and my A matrix by multiplying the B matrix by the inverse DCT matrix. I then used the CVXPY library with the CVXOPT solver to solve the optimization problem of minimizing the one-norm of x subject to $Ax = y$. Then I took the dot product between this value (x^*) and multiplied it by the inverse DCT matrix to get my final image. I did this three times for each percentage within r = [0.2, 0.4, 0.6] to get a diverse subset of the random B matrix, and plotted the results.

Finally, I used a simlar form of solving to extract the mystery image. The B and y matrices were already provided, so I just had to plug them into the same CVXOPT solver to reconstruc the final image.

4. Computational Results

After calculating the DCT matrix and applying it to our image, I got the following result for the coefficeints.

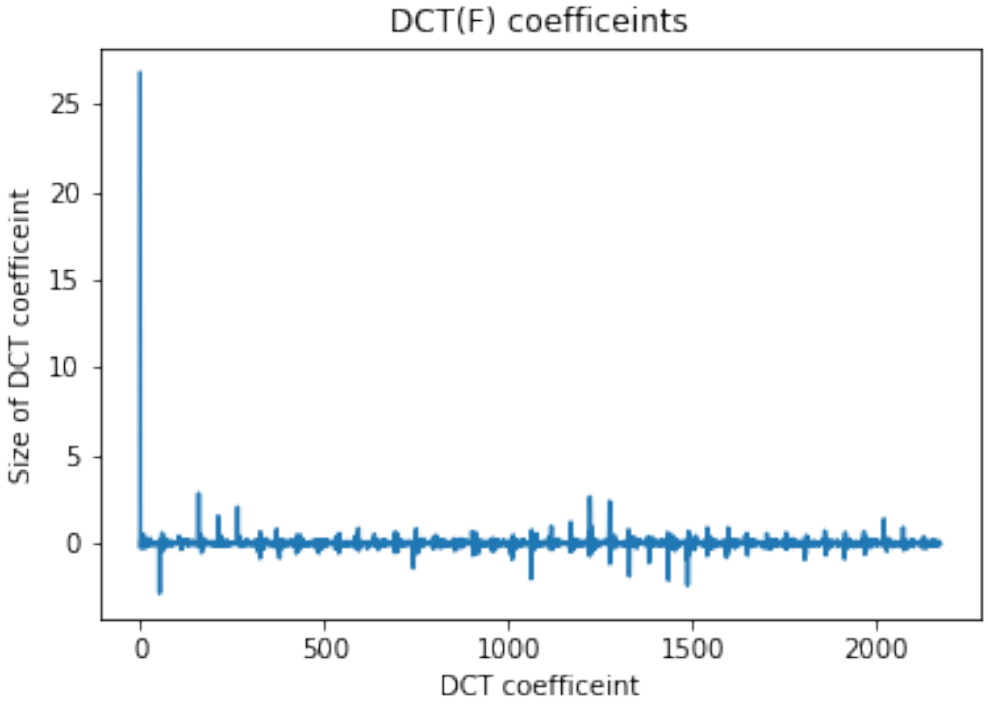

Figure 1. DCT Coefficients of SonOfMan image

We can see that the coefficients start out very large, then quickly become quite relatively small. After about 1500 coefficients, the absolute value of the coefficeints becomes even smaller. This indicates that the image is very compressible

Looking at the image with only a percentage of the coefficients we get the following result:

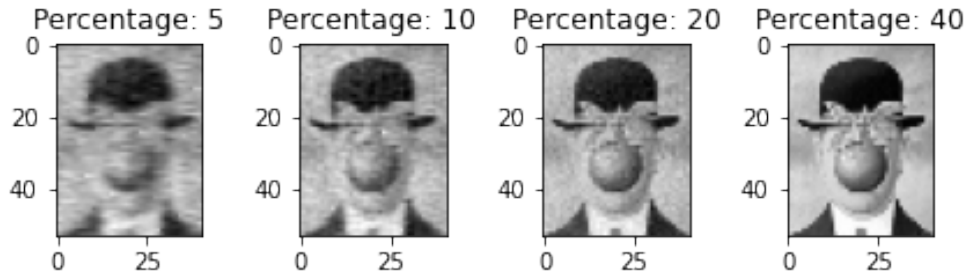

FIGURE 2. First DCT Coefficients of SonOfMan image broken up by percentages

We can see that keeping only 5% of the image does not result in a clear image, but at 40% we get a very clear image! In fact, we can see a surprising amount of detail at 20%, and a case could be made for using just 10% of the coefficients! This image is very compressible, just as we predicted.

Using our optimizer to recover the image, we get the following results:

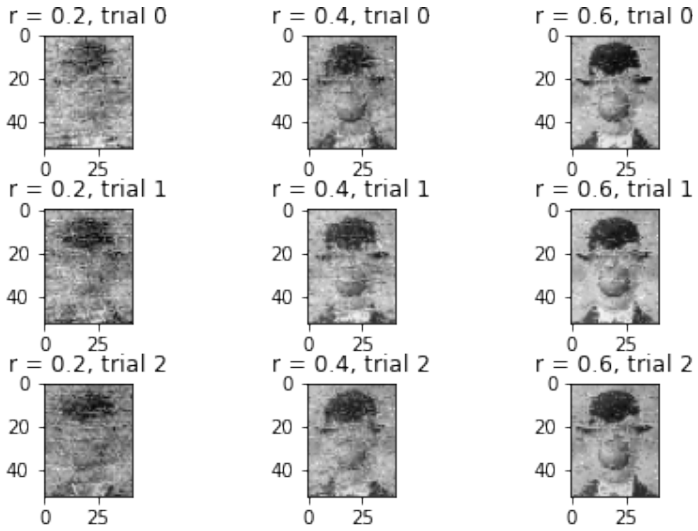

FIGURE 3. Reconstructing the SonOfMan image using the DCT with varying matrix sizes

We can see that we are able to recover the original image very well! When r = 0.6, we naturally have the most clarity. However, across multiple trials, we can also see that r = 0.4 provides good results as well. It makes sense that we need a larger percentage to reconstruct an image if the image was originally corrupted, but we can surprisingly see a lot more detail in the image than was present in the corrupted version!

Finally, let's see if we can apply the same methodology on an unknown mystery image. Performing the same optimization problem here, we get

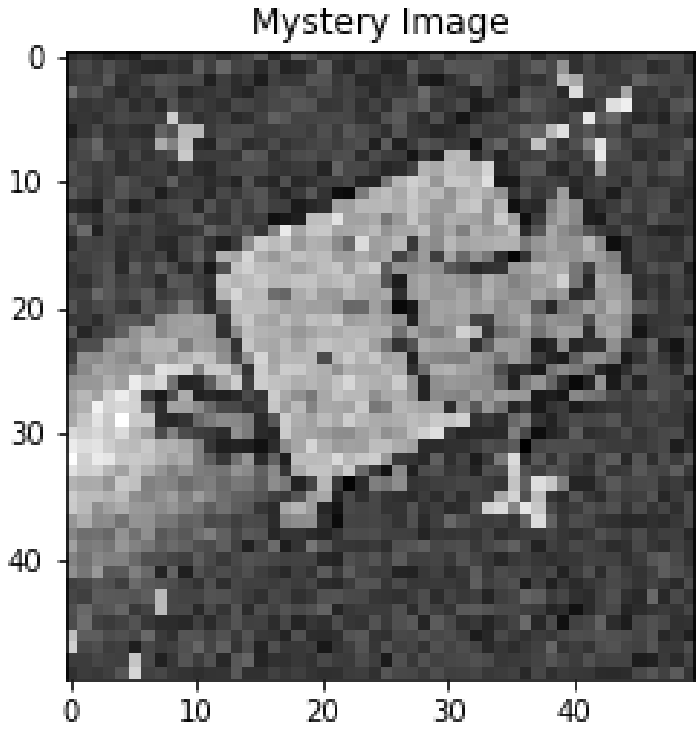

FIGURE 4. A Mysterious Image Reconstructed!

It works! We're able to reconstruct the mystery image to a remarkable degree of clarity. We can see that this image is none other than Nyan Cat!

5. Summary and Conclusions

We have seen that using our already pre-existing knowledge, we can re-construct images to a very high degree of clarity. We have also seen that we can use our knowledge to construct corrupted images even when we don't know the compressibility of it, as shown by Nyan Cat.

Acknowledgements

I am thankful to Professor Hosseini for introducing us to the concept of Fourier transforms, and by extension allowing us to learn about Discrete Fourier Transforms. Furthermore, I would also like to thank Professor Hosseini for

I am very thankful to my peers taking the class alongside me, they have helped me understand the material as well as provide a reference to compare my results against. I interacted with them both through Canvas discussion boards as well as Discord chat.

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] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] [2] [4] [1]