HW5: Compressed Image Recovery

Jessica Jones
Mar. 18, 2022

Abstract
Our goal is to recover the original image from the corrupted version. The original image was resized and compressed via the Discrete Cosine Transform (DCT) method and recovered via the inverse of this method. We reconstruct the image with a limited number of coefficients and explore image quality.

1. Introduction and Overview

The problem we consider is that of recovering an image from limited observations of its pixels. For example, consider a portion of René Magritte's "The Son of Man" along with its corrupted variant **(Figure 1).** This image was originally in color, converted to gray scaling, and rescaled to create a corrupted version. Our goal is to recover the original image from the corrupted version using Discrete Cosine Transform (DCT). We explore how this technique, using a limited signal, minimizes memory computationally, which gives an advantage for everyday use cases.

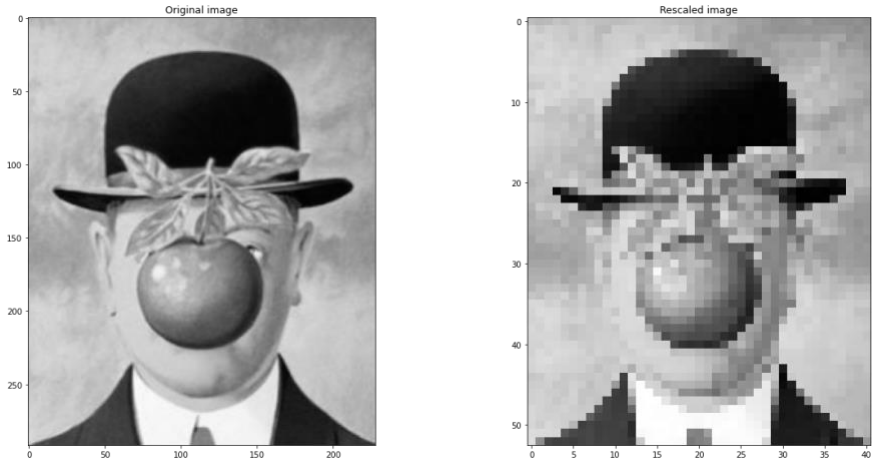

Figure 1: Original Uncompressed Image vs Rescaled,

2. Theoretical Background

2.1. Discrete Cosine Transform (DCT) and its inverse

DCT (Discrete Cosine Transform) is a cut-down version of the Fast Fourier Transform (FFT). Only the real part of FFT is used, and it is computationally simpler than FFT. This makes this method effective for media compression, which usually involves a more

extensive dataset, and MUCH more commonly used. For a 2-dimensional DCT, which is what we are attempting to do, the equation is as followed:

$$\mathrm{DCT}(k_1, k_2) = \frac{1}{\sqrt{2}} \sum_{n_1=0}^{N-1}, \sum_{n_2=0}^{N-1} x(n_1, r_2) \cos\left(\frac{\pi(2n_1+1)k_1}{2N}\right) \cos\left(\frac{\pi(2n_2+1)k_2}{2N}\right)$$

$$\text{where } 0 \le k_1 \le N-1 \text{ and } 0 \le k_2 \le N-1 \tag{1}$$

This can be simply written as:

$$DCT_{(\mathrm{m,n})} = DCT_{\mathrm{m}} \cdot DCT_{\mathrm{n}}^{\mathrm{T}} \tag{2}$$

the inverse, iDCT, is applied to reconstruct the original image from the transformed representation and is represented as

$$\mathrm{iDCT}(k_1, k_2) = \frac{2}{N}\left[\sum_{n_1=0}^{N-1}, \sum_{n_2=0}^{N-1} x(n_1, r_2) \cos\left(\frac{\pi(2n_1+1)k_1}{2N}\right) \cos\left(\frac{\pi(2n_2+1)k_2}{2N}\right)\right] \tag{3}$$

and as:

$$iDCT_{(\mathrm{m,n})} = \frac{2}{N}\left(DCT_{\mathrm{m}} \cdot DCT_{\mathrm{n}}^{\mathrm{T}}\right) \tag{4}$$

2.2. Optimization and Lasso Regression

Having an optimization problem involves finding a point that maximizes/minimizes the objective function through iterative computations (typically, iterative linear programming) involving convex functions. Our objective function to optimize is also subjected to convex constraints.

The Lasso Regression model is a linear model trained with L1 prior as a regularizer (aka the Lasso). For a Lasso problem:

$$\min_{\beta} \quad \frac{1}{2\sigma^2} \left|\left|(Y - A\beta)\right|\right|_2^2$$

$$\text{such that } |\beta|_1 \le t, \tag{5}$$

This is equivalent to finding the β that minimizes:

$$(Y - X\beta)'(Y - X\beta) + \gamma \sum_j |\beta_j| \,. \tag{6}$$

We want to do something similar. Let $M < N$ be an integer and construct a measurement matrix $B \in \mathrm{R}^{M \times N}$ by randomly selecting M rows of the identity matrix $I \in \mathrm{R}^{N \times N}$. We generate a vector of measurements $y \in \mathrm{R}^{M}$ by applying B to vec(F). We can then define the matrix

$$A = BD^{-1} \in \mathrm{R}^{M \times N}$$

(7)

and use CVX to solve the optimization problem:

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

(8)

We denote the minimizer as x*. This usually is not the minimizer that completely solves the problem, but it is the value where the solution reaches a certain tolerance; this optimization problem is unbounded and, thus, there is no optimal solution. For us, this vector x* is the DCT vector of an image F* that resembles the original image F.

3. Algorithm Implementation and Development

Load image and down sample to 53x41 pixels. We then convert this image to grayscale (via scikit image). For **Algorithm 1**, We took the DCT of the flattened image and filtered out certain frequencies to reconstruct (by taking the inverse DCT).

For **Algorithm 2,** we recovered the sparse image via **Section 2.2.** We used CVXPY, which is a domain-specific language for convex optimization embedded in Python. It allows the user to express convex optimization problems in a natural syntax that follows the math. The variable, objective, and constraints are each constructed separately and combined in the final problem. The optimal objective is returned by prob.solve(). Methods outlined in **Algorithm 2** were used to recover an Unknown Image **(Figure 4).**

3.1. Algorithm 1, *Image Compression*

Take DCT.
- Eqn. 1 & 2
- Ny, Nx = # rows, # columns of the image
- Construct D via ***construct_DCT_mat()***
- Vectorize image ***im_gray.flatten()***
- D · vectorized version of image
- Display via ***display_dct()***

Threshold to the first 5, 10, 20, 40% of the coefficients.
- absolute value(coefficients)
- Sort by size
- Identify the magnitude of the smallest coefficient that makes it into the top 5% or 10% or 20% etc.

Reconstruct Image using DCTs
- multiplying D^{-1} by the threshold DCT, where D^{-1} is constructed using ***construct_iDCT_mat()***
- Display via ***display_recons()***

3.2. Algorithm 2, *Optimization/Recovery*

Construct a “measurement matrix”, B
- B = M x N; M = # of points measured, N = # of flatten pixels in the image via ***B_()***
- $M < N$, integer
- Construct A via Eqn. 7

Convex optimization via ***optimize_F()***
- Generate measurements via B · flattened image
- Eqn. 8
- access x* by using ***x.value*** (x* is the DCT of your reconstructed image)
- reshape the product of D^{-1} times x* to the image dimensions to reconstruct via ***recov_image_processing()***
- Create matrix to plot via ***image_recov_mat()***

3.3. Implementation, *Unknown Image*

The measurements of the image are in y and the measurement matrix B tells us where the measurements were taken
- Create matrix A
- Define objective, constraints, and problem
- Solve for x*
- Plot

Do ***optimize_F()*** but on the unknown image and plots the result

4. Computational Results

Figure 2 plots the number of coefficients needed to construct our compressed Son of Man Image. We visualize frequency peaks DCT index # < 500. These coefficients were then thresh held and used to visualize our compressed image using the top 5, 10, 20, and 40% of these coefficients in **Figure 3.** Given the compressed image, these reconstructions are still easy to make out; In comparison to the top 5% of coefficients, which allows you to visualize the general outline of the apple in the man’s mouth, his hat, his coat, and his tie, thresholding the top 40% allows you to see finer details within the

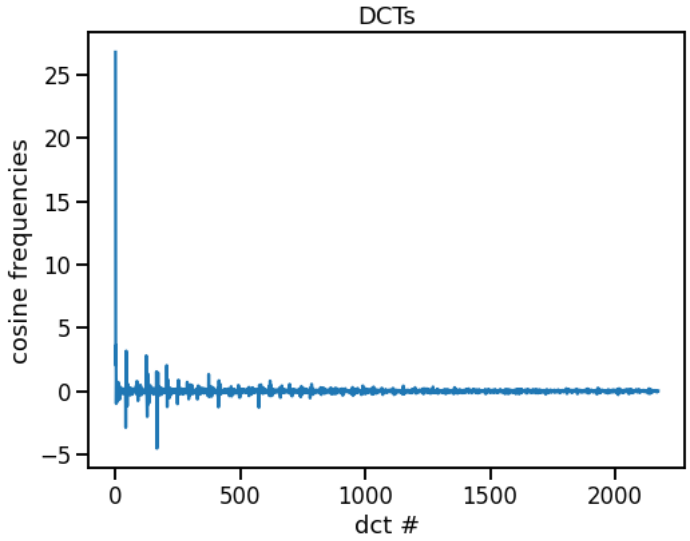

Figure 2: Plot of DCT. These are the coefficients needed to construct the image

painting. As comparison, I plotted the top 95% of coefficients that is comparable to the compressed image, and the top 40% was very similar. This is also seen in **Figure 2;** less noise is seen after ~869 DCTs, which is approximately 40% of the top coefficients. The image can therefore be compressed very well and reconstructed while preserving key details captured by the original, uncompressed painting from Rene.

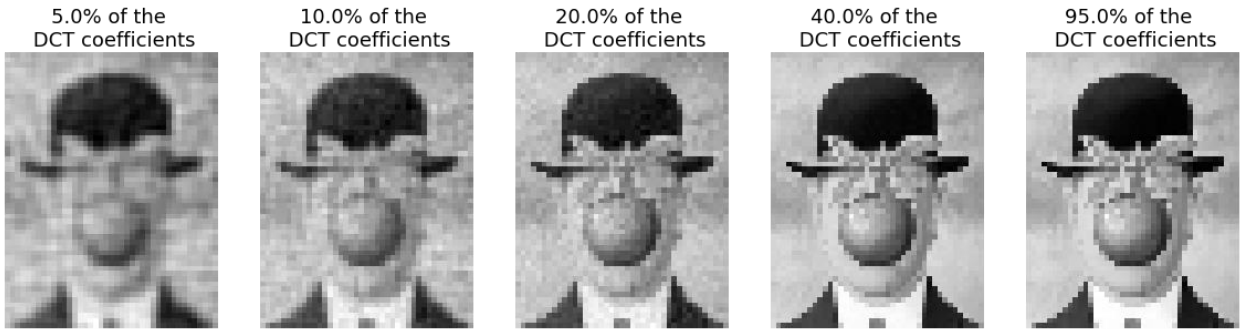

Figure 3: Reconstructed Image via top DCT coefficients. 95% of top DCT coefficients resembles original compressed image the most.

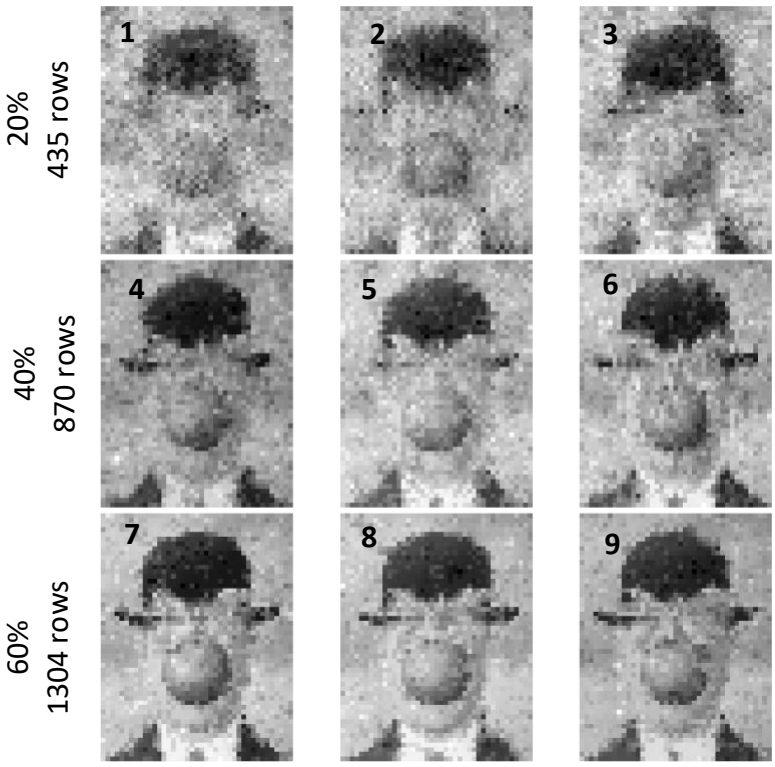

Figure 4: Reconstructed Compressed Images after Optimization using #M rows

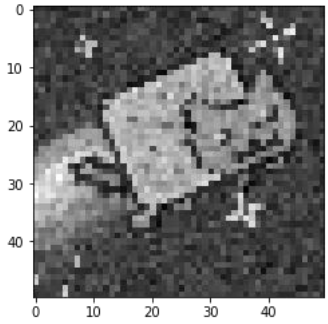

Figure 5: Reconstructed Unknown Image

Figure 4 displays the sparse reconstruction of the compressed image using 435 (20% pixels), 870 (40% pixels), and 1304 rows (60% pixels) of observations after implementing optimization in **Algorithm 2.** The image appears vaguer and more clouded just using the first 435 observation rows, but as we increase the number of observations, mainly from 870 to 1304 observations, the granularities start to minimize and features become more prominent and like the original compressed image; the man's hat, apple, clothing, and background become easily separable with increased iterations of the Lasso function outlined in **Equations 5-8.**

We implemented **Algorithm 2** to recover an unknown image in Figure 4, which is one of the most prolific cat memes the internet was ever so blessed with on YouTube, uploaded in April 2011. In recent news, this meme sold as a Non-Fungible Token (NFT) for 300 crypto ETH in February 2021 after an unexpected bidding war took place. That was equivalent to about $690,000 at the time.

5. Summary and Conclusions

Image compressed and recovery were explored via DCT, inverse DCT, and convex optimization. We can compress an image and reconstruct an image by thresholding the coefficients of the DCT, and we can recover the image, albeit sparsely, by optimizing the DCT observations in fifths. We found that, after optimizing after 20% of the observations, we can recover crucial components of the image.

6. Acknowledgements

I would like to give thanks to Professor Bamdad Hosseini for providing the necessary background to tackle this course, the AMATH Discord for coding suggestions and teaching me how to efficiently code, StackOverflow and Exchange, the friends I met along the way, and most importantly the AMATH582/482 Teaching Assistants.

References

Diamond, S., & Boyd, S. (2016). CVXPY: A Python-Embedded Modeling Language for Convex Optimization. Journal of machine learning research : JMLR, 17, 83.