The Art of Compression: JPEG and JPEG 2000 on Multi-spectral Satellite Imaging

Kritika Pandit
Department of Computer Science
Carleton College
Northfield, Minnesota
panditk@carleton.edu

Anika Rajbhandary
Department of Computer Science
Carleton College
Northfield, Minnesota
rajbhandarya@carleton.edu

Leon Liang
Department of Computer Science
Carleton College
Northfield, Minnesota
liangl@carleton.edu

Lucas Schattenmann
Department of Computer Science
Carleton College
Northfield, Minnesota
schattenmannl@carleton.edu

I. Abstract

In this project, we implemented both JPEG and JPEG 2000 in Python and tested them on 15 multispectral satellite images from the *Earth as Art* collection. For JPEG, we examined how quality level, chroma subsampling (4:4:4 vs. 4:2:0), and quantization tables affect compression and image quality. We found that area-based 4:2:0 subsampling produces nearly the same visual quality as 4:4:4 while using fewer bits, and that quantization is the main factor controlling the trade-off between compression and distortion. For JPEG 2000, we studied how changing the DWT level, block size, and entropy coding method impacts compression ratio, image quality, and runtime. We found that deeper wavelet decompositions and moderately sized code blocks provided the best rate–distortion trade-off for our satellite images. Finally, we compared our Python implementations to optimized library versions and observed a clear performance gap in both compression efficiency and speed. Our results highlight the key design choices that shape image compression performance and the importance of optimized implementations in real-world systems.

II. IEEE keywords

Image compression, JPEG, JPEG 2000, multispectral satellite imagery, chroma subsampling, quantization, discrete cosine transform (DCT), discrete wavelet transform (DWT), rate-distortion analysis, entropy coding, compression ratio

III. Introduction

Within the current digital world, massive amounts of image data are generated and shared, requiring efficient storage and transmission mechanisms. Image compression is a critical process of making image files smaller in order to take up less space while still preserving the quality of the image itself when reconstructed [1]. There are two main forms of image compression: "lossy" compression which intentionally loses information in order to achieve higher compression, and "lossless" compression which preserves the full quality of the original image but with greater restrictions in compression efficiency. In this project, we implement both JPEG and JPEG 2000 in Python, following the steps specified in the standards. In particular, the main difference between these two pipelines is the transformation step. JPEG employs Discrete Cosine Transform while JPEG 2000 employs Discrete Wavelet Transform. JPEG is a lossy image compression standard, while JPEG 2000 can be both lossy and lossless. The lossy versions of these image compression standards take advantage of fundamental aspects/limitations of our human visual system in order to achieve high compression ratios while still preserving the critical structure of the original image. In coding JPEG and JPEG 2000, we seek to understand the underlying implementation details of these standards and how altering certain parameters or methods can help produce optimal results that balance the trade off between compression efficiency, visual perception, and computational efficiency. While our implementations cannot reach the same performance as the highly optimized C JPEG and JPEG 2000 image codecs, we investigate the disparities in performance and critical limiting factors.

IV. Methods

A. JPEG

Our JPEG implementation can be broken down into the following steps as depicted in Figure 1. These steps are separated into five main sections: converting and separating the color space, downsampling the chrominance channels, transforming the spatial data into spatial frequency data, quantizing the values into discrete bins, and finally entropy encoding which performs lossless encoding of the transformed and quantized values. The inverse of these steps is then applied to the compressed image data to reconstruct the original image with quality loss.

1) Color Space Conversion (RGB to YCbCr): In the first step of the JPEG pipeline, the source image is converted from the RGB color space (Red, Green, Blue) into the YCbCr

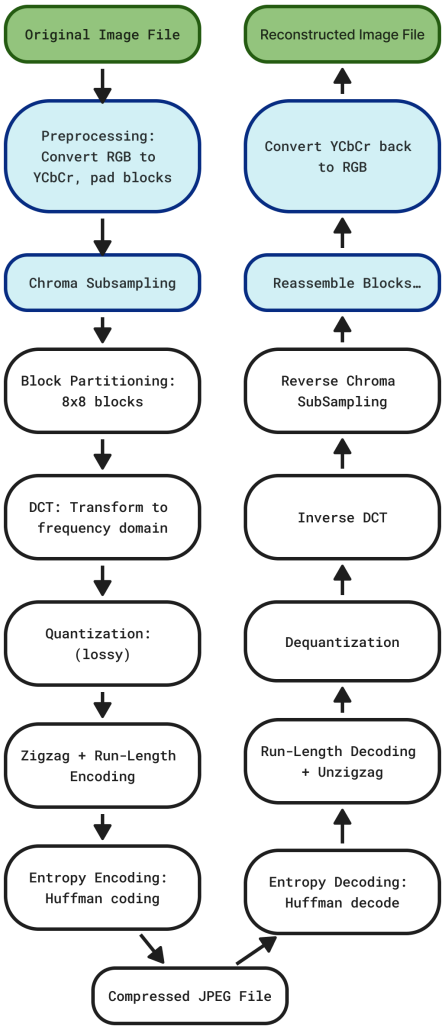

Fig. 1: JPEG Pipeline

color space, which separates brightness information from color information. The Y channel represents luminance (brightness), while the Cb and Cr channels represent chrominance (blue-difference and red-difference, respectively). This conversion is a standard step in lossy JPEG compression [2].

The conversion from RGB to YCbCr is performed using the following linear transformation:

$$Y = 0.299R + 0.587G + 0.114B \quad (1)$$

$$Cb = -0.1687R - 0.3313G + 0.5000B + 128 \quad (2)$$

$$Cr = 0.5000R - 0.4187G - 0.0813B + 128 \quad (3)$$

After compression and decompression, the three channels are recombined and converted back to RGB so that the final image can be displayed, regardless of the color space used during processing.

2) Chroma Downsampling: After conversion to YCbCr, JPEG further reduces file size by lowering the spatial resolution of the chrominance channels (Cb and Cr) while keeping the luminance channel (Y) at full resolution. This process, also known as chroma subsampling, is based on the fact that human vision is less sensitive to fine color details than to brightness details [2].

In many imaging systems, chrominance is sampled at a lower resolution than luminance. Common schemes include reducing chrominance resolution horizontally, vertically, or both. [2] In our experiments, we apply three standard chroma subsampling approaches, described below:

*a) **4:4:4 (No Subsampling)**:* All three channels (Y, Cb, and Cr) are kept at full resolution. This preserves the most color detail but produces the largest file size. We use this as our high-quality reference.

*b) **4:2:0 (Nearest Neighbor)**:* In this method, the chroma channels are reduced to half their height and width. Each channel is divided into 2×2 pixel blocks, and a single chroma value is selected to represent the entire block (typically the top-left or nearest neighbor value). This approach is computationally efficient, but can introduce blocky artifacts or uneven color transitions near edges.

*c) **4:2:0 (Average / Box Filter)**:* Here, the values in each 2×2 chroma block are averaged into a single value. This produces smoother color regions and usually looks better than nearest-neighbor downsampling while still reducing file size.

During decoding, the chroma channels are scaled back up to match the Y channel, which always stays at full resolution.

3) DCT: The discrete cosine transform (DCT) transforms spatially correlated pixel data into spatial frequency data. Frequency data represents the rate at which pixel intensities change across the block. This transformation concentrates the image data into fewer coefficients which are more easily manipulated during the quantization step. Each of the three separate color channels output from the previous step are partitioned into contiguous 8 x 8 blocks. DCT is applied independently to each block.

To prepare for the discrete cosine transform, the spatial pixel data are level-shifted by subtracting 128 in order to center the values around zero. The signal is then decomposed into a sum of 64 basis functions, shown in Figure 2. These basis functions are orthogonal to each other, ensuring that correlation with one signal will not interfere with the correlation with another signal, critical for reconstruction of the original pixel data from the grid of DCT coefficients during the inverse decoding step [3]. Low frequency basis functions are seen at the top left of the figure, and high frequency functions (rapid changes in pixel values) representing edges and fine details can be seen on the bottom right.

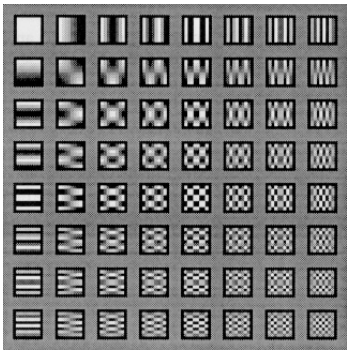

Fig. 2: 2D DCT basis functions [4]

The 1D DCT equation is defined as follows [4]:

$$F(u) = \frac{2}{N} C(u) \sum_{x=0}^{N-1} f(x) \text{Cos}\left(\frac{\pi(2x+1)u}{2N}\right),$$

$$C(u) = \frac{1}{\sqrt{2}} \quad \text{for } u = 0, \text{ otherwise } 1.$$

where DCT is applied to a sequence of N data points, f(x), to produce N coefficients, F(u), in the frequency domain. x represents the coordinate in the pixel domain and u represents a coordinate in the frequency domain.

The 2D DCT equation for the forward transformation is given below [4]:

$$F(u,v) = \frac{4}{N^2} C(u)C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \times \text{Cos}\left(\frac{\pi(2i+1)u}{2N}\right) \text{Cos}\left(\frac{\pi(2j+1)v}{2N}\right), \tag{4}$$

$$C(u) = \frac{1}{\sqrt{2}} \quad \text{for } u = 0, \text{ otherwise } 1$$
$$C(v) = \frac{1}{\sqrt{2}} \quad \text{for } v = 0, \text{ otherwise } 1. \tag{5}$$

where f(i,j) represents the coefficient value in the spatial domain, and F(u,v) represents the coefficient value in the transformed frequency domain.

Using the separability property, this 2D DCT transformation can be achieved by applying a 1D DCT transformation to the rows of the original matrix, transposing the matrix, then applying a 1D DCT to the columns. The equation for this operation is shown as follows [4].

$$F(u,v) = \frac{2}{N} C(u)\, C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) \cos\left(\frac{\pi(2i+1)u}{2N}\right) \cos\left(\frac{\pi(2j+1)v}{2N}\right) \tag{6}$$

This approach reduces the time complexity for an NxN block from $O(N^4)$ (required for a direct implementation of the 2D equation) to $O(N^3)$. For the inverse DCT, the basis function magnitudes are summed to calculated to reconstruct the original signal, applying the separability property using 1D IDCT to achieve the 2D IDCT transformation.

4) Quantization: Quantization is the critical "lossy" step which intentionally loses lower impact information in order to achieve higher compression ratios. This reduction in precision generates greater statistical redundancy that can be exploited in the subsequent entropy encoding steps. As seen in the equation below, uniform scalar quantization is achieved by dividing each DCT coefficient, $F(u,v)$ in the transformed grid with the corresponding quantization value, $Q(u,v)$ in the given quantization table. This value is then rounded into discrete integer bins, $F^Q(u,v)$ [3].

$$F^Q(u,v) = \text{Integer Round}\left(\frac{F(u,v)}{Q(u,v)}\right)$$

The quantization tables used in the JPEG standard are depicted below in Figures 3 and 4 [5]. These tables were empirically chosen using a series of psycho visual tests, optimized for the human visual system [6].

Luminance Quantization Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Fig. 3: Standard JPEG Luminance Quantization Table

Chrominance Quantization Table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Fig. 4: Standard JPEG Chroma Quantization Table

This quantization step takes advantage of two main features of the human visual system (HVS). The first feature is that the HVS is less perceptive to higher spatial frequency information [6]. Thus, the quantization tables use larger values, Q(u,v), in the bottom right coefficients. The second feature is that the HVS is less perceptive to chrominance information than

luminance information, as discussed in the color conversion section [6]. Thus, higher values can generally be seen in the chroma quantization table than in the luminance quantization table. Ultimately, this process divides both high frequency information and chroma information more heavily, obtaining greater runs of zero length coefficients for information less critical to the HVS. This significantly lowers the entropy of the input matrix, $F^Q(u, v)$ which is then efficiently compressed in the subsequent entropy encoding steps. This quality loss is depicted in the following heatmap, where several zero-length coefficients are seen concentrated towards the bottom right of the map after the quantization step.

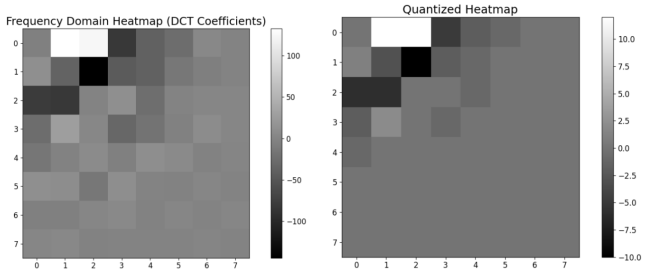

Fig. 5: Example quantization step of 8x8 DCT transformed block, from block 100 of "Deep-blue-cubism.tif"

Depending on the user input quality level, the following equation is used to scale the quantization table, where larger compression is applied to smaller quality levels. Given a quality level, Q, the scaling factor S is computed. The resulting quantization value for the scaled table, $T_s[i]$, is then calculated using the following formula, based on S and the base table quantization value, $T_b[i]$ [7].

$$S = (Q < 50)?\frac{5000}{Q} : 200 - 2Q$$

$$T_s[i] = \left\lfloor \frac{S \cdot T_b[i] + 50}{100} \right\rfloor$$

This quality level scaling allows users to adjust the extent of compression reached with a trade off in distortion of the reconstructed image. In the de-quantization step, each quantized frequency coefficient, $F^Q(u, v)$, is multiplied with the corresponding value in the scaled quantization table to reconstruct the approximate DCT coefficient values.

5) Entropy Coding: As described in the quantization step, many high-frequency and chrominance coefficients are heavily suppressed, leaving a large number of values rounded to zero in each 8×8 block. This significantly lowers the entropy of the quantized coefficient matrix, $F^Q(u, v)$, and creates strong statistical redundancy. Entropy coding exploits this redundancy and converts the quantized coefficients into a compact stream of symbols that can be stored or transmitted using far fewer bits. In our implementation, this stage consists of three main steps: zig-zag reordering, run-length encoding, and Huffman coding.

- **Zig-Zag Reordering:** Each 8×8 block of quantized coefficients is first flattened into a one-dimensional sequence using a fixed zig-zag scanning pattern, as shown in Figure 6. This scan begins at the top-left corner, which contains the DC coefficient, and then weaves through the block from low to high spatial frequencies, which are represented by the AC coefficients. The DC coefficient represents the average (low-frequency) intensity of the entire block, while the AC coefficients represent progressively higher-frequency variations such as edges and fine texture. Because most visually important energy is concentrated in the low-frequency components, the early part of the zig-zag sequence typically contains a few non-zero values, while the later part is dominated by long runs of zeros. By clustering these zeros toward the end of the sequence, zig-zag ordering prepares the data for more effective compression in the next step. [8]
- **Run-Length Encoding (RLE):** Run-length encoding compresses repeated patterns, especially the long sequences of zeros created by quantization and emphasized by zig-zag reordering (see Figure 6) [8]. In JPEG, the DC and AC coefficients are treated differently to further reduce redundancy:
  - *DC coefficient:* Instead of storing the absolute DC value of each block, we encode the difference between the current block's DC value and the previous block's DC value. Because neighboring blocks in natural images tend to have similar average intensities, this difference is usually small, making it more efficient to encode [8].
  - *AC coefficients:* For the remaining 63 AC coefficients in the zig-zag sequence, we encode the number of consecutive zeros that appear before each non-zero value. Each non-zero AC coefficient is stored as a $(run, value)$ pair, where *run* is the length of the zero sequence and *value* is the next non-zero coefficient. Very long zero sequences are represented using a special Zero-Run-Length (ZRL) symbol, and when the remainder of the block contains only zeros, an End-of-Block (EOB) symbol is used [8].

  Through these mechanisms, each block is transformed from a dense grid of 64 coefficients into a short sequence of highly compact symbols that capture only the essential information.
- **Huffman Coding:** In the final step, the symbols produced by RLE (Figure 6) are assigned binary codes using Huffman coding [9]. Symbols that appear more often are given shorter bit patterns, while less common symbols are given longer ones. This creates a compact bitstream that can be efficiently stored or transmitted. During decoding, the same Huffman table is used to reverse the process and recover the original RLE data.

B. JPEG 2000

Broadly speaking, the JPEG 2000 pipeline sports many similarities to the JPEG pipeline previously discussed, with a few key differences. Namely, after the color conversion and

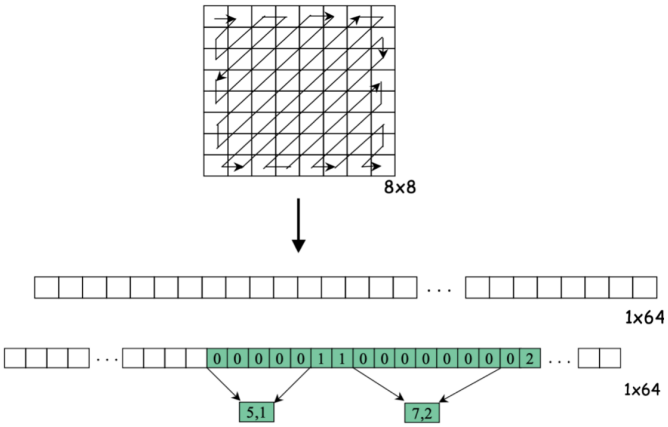

Fig. 6: Zig-zag scanning of an 8×8 block and the resulting 1D sequence, followed by run-length encoding of zeros [10].

separation step, JPEG 2000 performs the Discrete Wavelet Transform (DWT) rather than DCT. Also, after quantization, the image is partitioned into square chunks to prepare for entropy encoding. These steps can be performed in reverse to reconstruct the image with some quality loss. Fig. 7 provides a general overview of the JPEG 2000 pipeline.

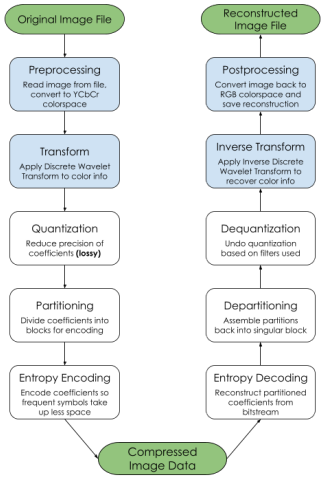

Fig. 7: JPEG 2000 Pipeline

1) Color Conversion: As mentioned above, the JPEG 2000 pipeline uses the same YCbCr color space conversion as JPEG. The color channels are then separated in preparation for the transform.

2) DWT: Once the image has been converted, the Discrete Wavelet Transform is applied to each separated color channel. In essence, the goal of DWT is to transform the way the image's information is represented in order to more effectively compress the results. Each pass of DWT will return four arrays, those being one array of approximant coefficients which represent a scaled-down version of the original image, and three arrays of detail coefficients, which store the difference between individual pixel values and the average value. These coefficients are known as the horizontal (H), vertical (V), and diagonal (D) coefficients based on the position of the pixel in each 2x2 chunk compared to the top-left pixel of that chunk.

This transform takes advantage of the fact that pixels in an image are highly correlated in terms of their color values, so pixels near each other will typically encode very similar information. DWT allows for image information to be encoded using these small differences, allowing for many of the detail coefficients to be very close to 0. This is crucial for more effective entropy encoding in the future. Finally, because an array of the approximant coefficients is also returned, the transform can be reapplied to these approximants many times over to continue generating additional detail coefficients in a process known as multi-level deconstruction. This can be seen a little more clearly in Fig. 8, which represents a 2D array after a 2-level wavelet deconstruction. The approximants, normally stored in the top-left corner after one level, have had DWT applied to them once more.

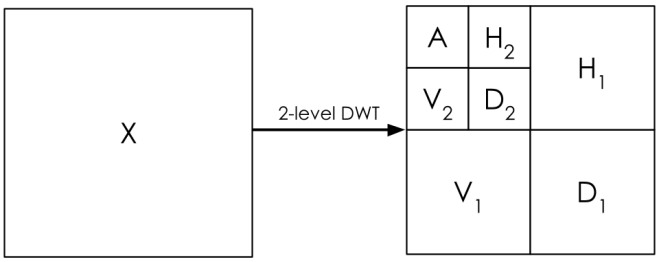

Fig. 8: 2-level DWT Example

3) Quantization: After applying DWT, the image is decomposed into one low–frequency component (LL) and several high frequency components (HL, LH, HH) at multiple decomposition levels. These sub-bands differ in perceptual importance: the LL band captures the coarse structure of the image, whereas the HL/LH/HH bands contain progressively finer directional details. [11]

We then apply scalar quantization with different step size depending on sub-band importance. The quantization step size Δ is the space between quantized coefficient values, and the dead-zone parameter δ defines a region around zero where coefficients are all mapped to zero. Smaller quantization steps are assigned to the LL band to preserve precision, while larger steps are used for the high-frequency bands, where visual sensitivity is lower. This converts float coefficients within a range into an integer, reduces the number of unique values, and provides storage savings with minimal perceptual degradation.

The quantization equation is:

$$Q(x, \Delta, \delta) = \begin{cases} \text{sgn}(x) \left\lfloor \dfrac{|x| - (\delta - 1)\Delta}{\Delta} \right\rfloor, & |x| \geq \delta\Delta \\ 0, & \text{otherwise} \end{cases}$$

where Δ denotes the step size and δ denotes the dead-zone parameter. This quantization stage is the only lossy step in our pipeline.

4) Partition: Following quantization, each sub-band is partitioned into blocks of user-defined size (64×64 by default). Partitioning provides two main benefits. First, within smaller spatial regions the coefficient tends to be more uniform, which improves the efficiency of subsequent entropy coding. Second, because the blocks are independent, each one can be encoded in parallel.

Conceptually, the quantized coefficient array is divided into 64×64 sub-arrays. Each block is treated as an independent coding unit for entropy coding. This strategy keeps the structure in JPEG 2000 and facilitates multi-threaded compression pipelines. [12]

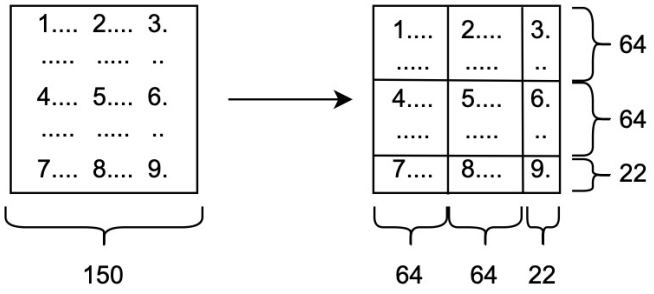

Fig. 9: Partitioning structure

5) Entropy Coding: Each partitioned block is entropy coded independently. The coder is selected adaptively based on the statistical properties of the block. Blocks that contain very few unique coefficient values or exhibit strong repetition patterns are encoded using DEFLATE (LZ77 combined with Huffman coding), as DEFLATE is highly effective at compressing repeated coefficients. This situation commonly occurs in high-frequency sub-bands where coarse quantization has driven many coefficients to zero.

Blocks with more diverse coefficient distributions are instead compressed using adaptive arithmetic coding, a form of entropy coding that represents an entire sequence of symbols as a number in [0,1), determined by symbol probabilities. Updating these probabilities adaptively during encoding improves efficiency for higher-entropy data and eliminates the need to store explicit entropy-coding tables, thereby reducing metadata overhead. Furthermore, all entropy coding operations run in parallel, which is particularly beneficial given that entropy coding is the most computationally demanding part of the pipeline. The result of this stage is a binary bitstream containing the encoded blocks along with metadata needed for decoding. [13]

V. Experiments and Results:

A. Metrics

In order to analyze the results of the experiments we conduct, we must introduce some standardized evaluation metrics, such as compression ratio (CR), bits-per-pixel (BPP), mean squared error (MSE), and peak signal-to-noise ratio (PSNR). These metrics allow us to quantitatively measure the efficiency and efficacy of our approaches.

When evaluating any compression algorithm, the amount by which it can compress a file is one of the most important factors to evaluate. The compression ratio of a file is defined by the size of the compressed file divided by the size of the original file. A lower compression ratio indicates a more highly compressed image.

$$\text{CR} = \frac{\text{Size of Compressed File}}{\text{Size of Original File}}$$

Another measure of size is the bits-per-pixel of an image, which calculates the average number of bits needed to encode the information of one pixel:

$$\text{BPP} = \frac{\text{Total \# of bits}}{\text{Total \# of pixels}}.$$

When comparing two images, a lower BPP indicates that less data is needed to encode the same information, so the storage of that information is more efficient.

Next, we introduce some metrics related to the quality of the reconstructed image. The mean squared error (MSE) of two images measures how large the differences between the two images are. In image compression applications, this is typically used to compare a reconstructed image to the original and measure how much information was lost during the compression process. The MSE between two images is defined as follows:

$$\text{MSE} = \frac{1}{M * N} \Sigma_{i=1}^{M} \Sigma_{i=1}^{N} (f(x, y) - g(x, y))^2$$

where $f(x, y)$ and $g(x, y)$ represent the values of each pixel at position x, y. A lower MSE indicates that there is lower error between the two images, meaning less information was lost in the compression process.

We can use the MSE to calculate the peak signal-to-noise ratio between the two images as well:

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{MSE}$$

B. Lab Machine Details

The machine ran Ubuntu 22.04.5 LTS on a 64-bit x86_64 architecture. It was equipped with a 13th Gen Intel® Core™ i7-13700 CPU (16 cores, 24 threads, up to 5.2 GHz) and 16 GB of RAM. All computations were performed on the CPU only. The software environment used Python 3.10.12 along with common scientific libraries, including NumPy, Matplotlib, and scikit-image. Package management was handled through Conda environment, supplemented with pip installations.

C. Dataset

We conducted our experiments using 15 satellite images downloaded from the Earth as Art gallery by Earth Resources Observation and Science (EROS) Center. The images were selected to cover a variety of landscape types and visual characteristics while keeping the experiments computationally feasible. The Earth as Art collection features satellite imagery captured by satellite sensors, relying on the interplay of visible and invisible light across the electromagnetic spectrum.
All images were downloaded directly from the EROS Center's Earth as Art online gallery. The original TIFF files were preserved to maintain the native variations in size and the authentic characteristics of the satellite imagery. No preprocessing was applied prior to running the pipelines.
This dataset was chosen due to its variety in scene types, including deserts, mountains, coastlines, ice formations, and river systems, which provides visual diversity for assessing the performance of our methods. Using realistic, unaltered scenes helps ensure that our experimental results better reflect real-world conditions. [14]

D. Overview of Experimental Findings

Our results are organized in three parts. First, we analyze how key parameters in our JPEG pipeline affect rate- distortion behavior. Second, we study how structural and algorithmic choices in our JPEG 2000 implementation affect DWT, partitioning, and entropy coding performance. Finally, we compare both custom implementations against widely used library to understand the performance gap between a pedagogical Python implementation and highly optimized production code. Throughout, we use a common set of satellite images and shared metrics (PSNR, SSIM, bits per pixel, compression ratio, and runtime) so that trends are directly comparable across standards.

E. JPEG Experiments and Results

For JPEG, since our goal is to understand how different parametric settings choices shape the trade-off between compression efficiency and image quality, first, we looked at the effect of altering quality levels on the compression efficiency and distortion. We then focused on how chroma subsampling impacts rate-distortion behavior across a range of quality levels. Lastly, we looked at how different quantization tables altered the performance results. Additional JPEG analyses (SSIM-based curves, entropy statistics such as coefficient occupancy and zero-run distributions, image property vs rate correlations) are included in the Appendix.

1) Quality Level Rate Distortion Curve: We tested the impact of quantization scaling by varying the input quality levels. The results are displayed in Figure 10, which depicts an inverse relationship between compression efficiency and reconstruction quality; while lower quality setting achieve better compression, it results in greater distortion of the reconstructed image. Critically, the section of the curve around quality levels 50-70 provide an effective balance between the rate and distortion. Points outside of this window provide diminishing returns: increasing the quality level above 70 results in marginal increases in quality result for disproportionately larger decreases in compression efficiency, while qualities below 50 results in disproportionate distortion for minimal compression rate gain. Differences in performance levels between images can be explained by intrinsic image complexity, specifically edge density (as further described in appendix, Figure 28).

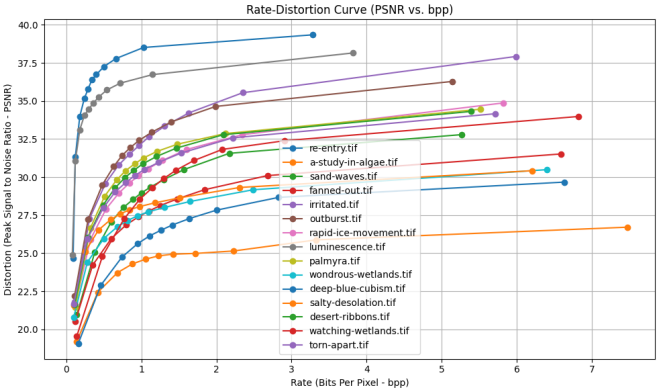

Fig. 10: PSNR vs BPP for Quality levels [0, 10, ... 90, 100]

2) Chroma Subsampling: Rate-Distortion Behavior: We next examine how different chroma subsampling strategies affect JPEG compression. Because JPEG allows the chroma channels (Cb and Cr) to be stored at lower resolution than the luminance channel (Y), we compare three modes: 4:4:4 (no subsampling), 4:2:0 with nearest-neighbor downsampling, and 4:2:0 with area-based (average) downsampling. For each image, we compress at multiple JPEG quality levels while keeping all other parameters fixed. To quantify this behavior, we use standard rate-distortion plots of PSNR vs. bits per pixel (BPP), where each point is averaged across all 15 satellite images.

Figure 11 shows that all three chroma modes follow the expected pattern: PSNR rises quickly as BPP increases, then levels off once the compression process has enough bits to preserve most visual detail. The 4:4:4 mode achieves the highest PSNR because it keeps full chroma resolution. However, 4:2:0 with area-based downsampling(average) performs very similarly to 4:4:4, while 4:2:0 with nearest-neighbor downsampling consistently has the lowest PSNR. Overall, these curves show that chroma subsampling can substantially reduce bitrate without significantly harming image quality, provided that a smooth downsampling filter (such as average) is used. Nearest-neighbor subsampling is computationally cheaper but produces noticeably worse quality at every bitrate.

3) Quantization Tables: In order to better understand the theoretical reasoning behind the empirically derived standard JPEG table, we tested three different quantization tables: the standard JPEG quantization table (referenced in the methods), a 'large flat' implementation with uniform values of 99, and a 'small flat' table with uniform values of 16. Unlike the standard quantization tables, the flat versions apply equal com-

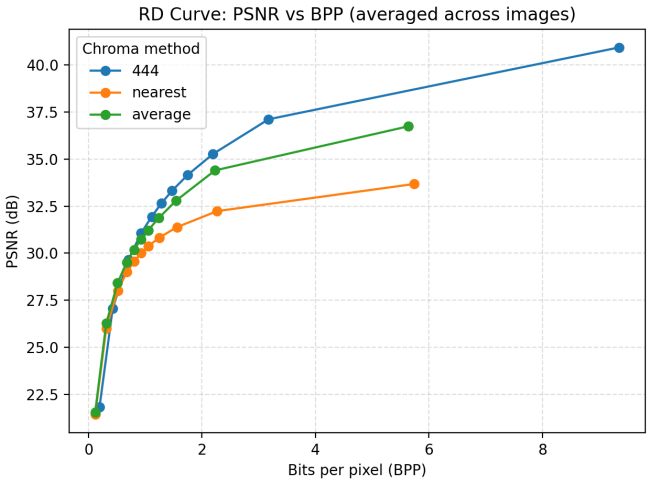

Fig. 11: PSNR vs. BPP for three chroma subsampling methods.

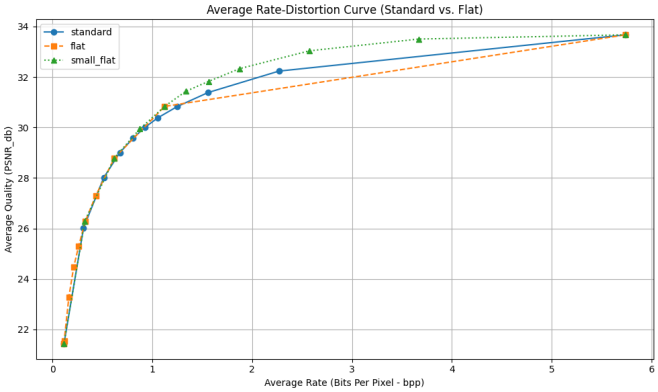

Fig. 12: PSNR vs BPP for Quality levels [0, 10, ... 90, 100]

pression across all frequency bands. Figure 12 illustrates the results of this test. As expected, the large flat implementation achieved better compression ratios given the same quality level at the expense of larger image distortion, generally below acceptable quality limits (30 db). Both flat tables also yielded results in suboptimal regions of the curve where the balance between rate and distortion is ineffective.

Given comparable rate and PSNR value for all three implementations, viewing the actual reconstructed image paints a clearer picture of the limitations of the flat table implementations. At this similar PSNR and compression ratio value, the visual quality of the 'large flat' implementation is noticeably worse, characterized by larger visible blocking artifacts (See images in appendix Figures 29, 30, and 31). This can be attributed to the fact that the flat implementation divides the visually important low frequency and less important high frequency data equally, resulting in discernible differences in visual quality despite similar PSNR metric values. Differences between the 'standard' and 'small flat' implementations are not discernible, though PSNR values are inflated by less mathematical distortions in high frequency bands. These results help expose a critical limitation of quality metrics like PSNR and MSE, which measure aggregate differences between the original and reconstructed image pixels, in being able to represent actual visual quality degradation. This highlights limitations in techniques that optimize for MSE and PSNR quality metrics, when seeking to optimize for the HVS.

In addition to the rate-distortion curves, we also looked inside the JPEG pipeline to see where information is being removed and how this affects entropy coding. Appendix VIII-A2 visualizes DCT coefficient occupancy as 8×8 heatmaps for different quality levels and chroma subsampling schemes, showing that at low quality almost all mid- and high-frequency coefficients are quantized to zero, while at high quality many more frequencies survive, especially in the luminance channel. Appendix VIII-A3 then analyzes the resulting zero-run distributions after zig-zag ordering and quantization, confirming that stronger quantization produces extremely long runs of zeros (especially in the chroma channels) and that increasing quality shortens these runs. Together, these visualizations support our main JPEG findings: quantization level is the dominant factor controlling sparsity, zero-run statistics, and hence the effectiveness of entropy coding, while chroma subsampling plays a smaller but still consistent role.

F. JPEG 2000 Experiments

Overview: We conducted three experiments using the same collection of 15 images described in the previous section. The compression pipeline includes several components: DWT, quantization, partitioning, and entropy coding, which are all user-defined parameters. In this study, we focused on three key adjustable parameters:

- **Deconstruction level** — the number of times DWT is performed on each color array before quantization.
- **Block size** — the partitioning size that controls how the image is divided into blocks.
- **Pivot value** — the threshold used to select between DEFLATE and adaptive arithmetic coding. (see Appendix VIII-B1 for details)

In all experiments, we varied one parameter at a time while holding the remaining parameters constant. This design isolates the functional effect of each parameter on the compression pipeline. The evaluation metrics considered include runtime, peak signal-to-noise ratio (PSNR), and compression ratio.

1) DWT Level Test: Our DWT experiments focused on the effects of varying the level of deconstruction performed. Each image was compressed using 0- to 11-level deconstructions, for a total of 12 runs per image. 0-level DWT is analogous with skipping the DWT portion entirely and quantizing the color components directly; on the other hand, 11-level DWT is the highest level of deconstruction possible on images of this size. All experiments were also done with maximum quality quantization and 64x64 partitioning.

Fig. 13 shows how the quality of each image is affected by the level of deconstruction used in DWT. As expected, the quality of the reconstructed image suffers with the addition of DWT; however, subsequent deconstructions have a low impact on the total image quality. This is to be expected; skipping DWT minimizes the amount of information lost, as no detail coefficients are produced which would be quantized more steeply later in the pipeline. All images have a PSNR of around 52 when skipping DWT. Once DWT is applied, the quality of each image takes a significant hit. The magnitude of the loss in quality is most likely correlated with how unpredictable the image is; images with big blocks of similar colors are far less susceptible to losing data to the quantization step, as detail coefficients will be very low across the board.

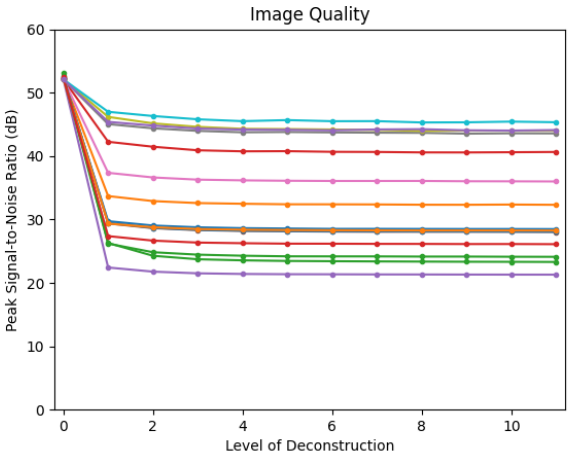

Fig. 13: Peak Signal-to-Noise Ratio vs DWT

As image quality decreases, however, so does compression ratio. Fig. 14 illustrates how the level of deconstruction affects the compression ratio of each image. Compression ratios are relatively high without DWT, as the entropy encoding portion of the pipeline is not nearly as effective without the detail coefficients which DWT produces. As more details are introduced compression ratios for each image improve, with most images ending up below 60% of the size of the original file with max-level deconstruction. This result showcases one of the primary strengths of including DWT in the JPEG 2000 pipeline.

Finally, similar to compression ratio, the runtime experiments looked generally opposite to those on image quality, as shown in Fig. 15. Without any detail coefficients which will typically end up very small and quantized to similar values, the entropy encoding portion of the algorithm suffers heavily. This leads to drastically inflated runtimes when skipping DWT entirely. Even one-level DWT significantly outperforms the runtime of the algorithm when skipping DWT on all images. One interesting phenomenon is that the runtime for all images increases between 1-level DWT and 3-level DWT, before dropping back down to expected levels. This could be related to the specific 64x64 partitioning block size used in all DWT-

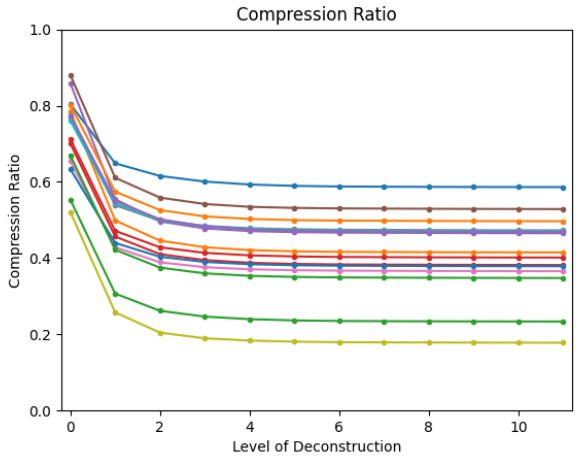

Fig. 14: Compression Ratio vs DWT

related experiments, but more testing is required to fully understand this finding.

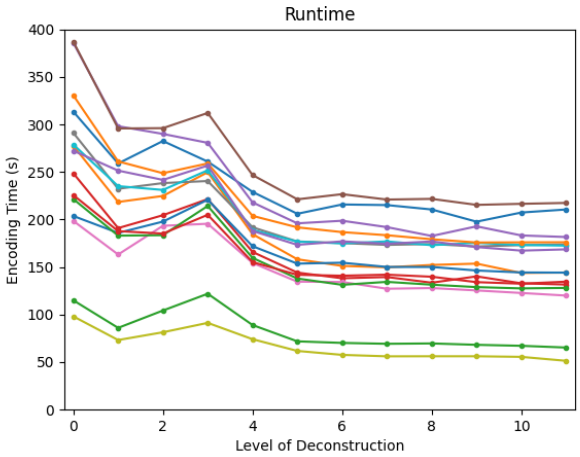

Fig. 15: Encoding Pipeline Runtime vs DWT

2) Block Size Test: This experiment examined the role of block size in the partitioning stage. Block size determines how the image is divided into non-overlapping blocks prior to entropy coding. Larger block sizes result in fewer total blocks, reducing the amount of side information that must be stored in the output bitstream. On the other hand, larger blocks contain more coefficients per block, which can affect the behavior of the entropy coder depending on coefficient diversity. Because block size does not modify the DWT or quantization stages, it does not change coefficient values themselves. Thus, the reconstructed image is not dependent on the block size we choose. For this test, block size was varied from 8 to 128, in increments of 16, while DWT level, quality level, and pivot were kept constant.

Figure 16 shows the resulting compression ratios across block sizes. Very small blocks (e.g., 8×8) exhibit poor

compression efficiency due to heavy metadata overhead. Each block requires its own header and entropy-coding state, and with many blocks the overhead dominates the bitstream. As block size increases, the number of blocks decreases and this metadata cost is reduced, leading to improvement in compression performance.

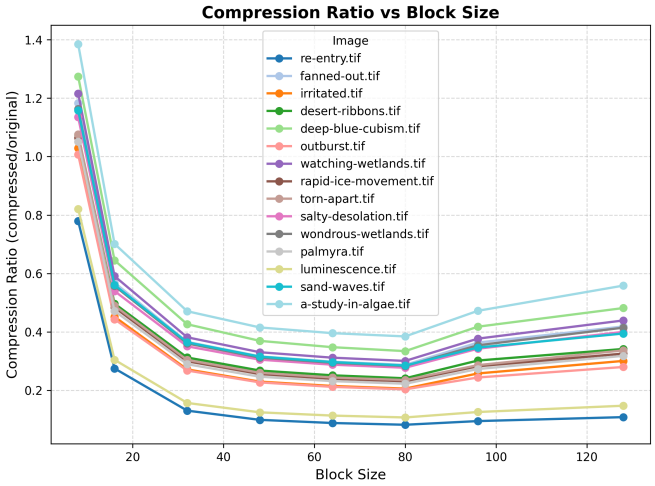

Fig. 16: Compression ratio vs. block size plot.

However, beyond block sizes of 80, the compression ratio begins to rise again. Larger blocks has greater coefficient diversity, making adaptive arithmetic coding less efficient. As a result, the coder produces longer representations for many symbols, offsetting the gains from reduced metadata. Fig. 16.

Runtime behavior is shown in Fig. 17. Overall runtime increases with block size because larger blocks require more computation per block because of adaptive probability updates. Runtime peaks around block sizes of 80. The increase before the peak is due to the increased cost of encoding each larger block. Beyond this point, runtime decreases because very large blocks substantially reduce the total number of blocks, reducing the overall number of entropy-coding operations and the overhead associated with initiating each block.

To understand the trade-off between compression ratio and runtime, Fig. 18 plots the Pareto frontier. Block sizes in the range of 32-64 consistently lie on or near this frontier, meaning that they provide the best balance of efficiency and computational cost. Based on these observations, our implementation adopts a default block size of 48, which performs well across all tested images.

G. Comparing our implementation with the libraries

Lastly, we compared the performance of our implementations against the industry standard Pillow JPEG codec and openJPEG Glymur JPEG 2000 codec (baseline implementations) across key compression efficiency and runtime/computational complexity metrics.

Fig. 19 illustrates that the baseline implementations had better compression results than our implementations. Given the same compression ratio, the built in implementations had

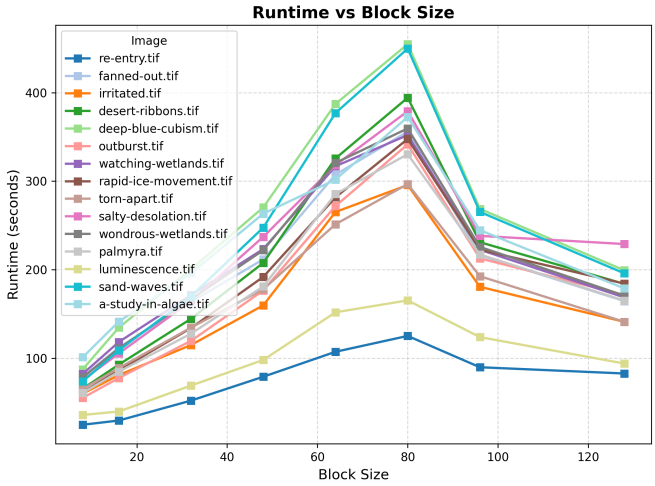

Fig. 17: Runtime vs. block size plot.

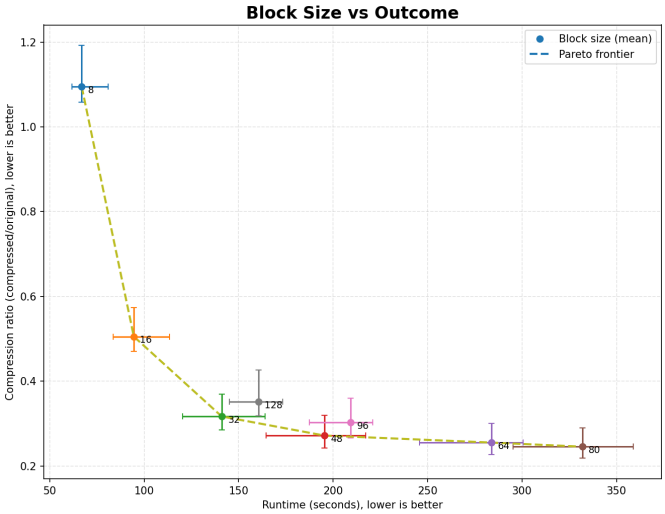

Fig. 18: Compression vs. runtime tradeoff plot.

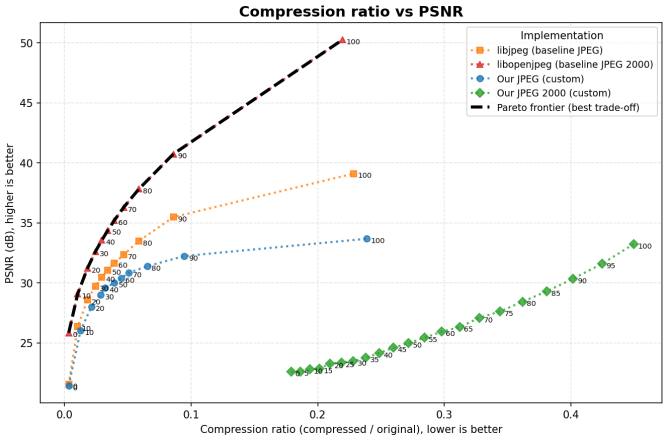

Fig. 19: Compression ratio vs. PSNR plot.

significantly higher PSNR values for both standards. We think that this disparity can be attributed to the following main factors. Our JPEG 2000 pipeline does not include many of the built-in tools and optimizations used in JPEG 2000 encoders (e.g., ROI coding, EBCOT, JP2 formatting, etc.). Pillow and OpenJPEG rely on highly optimized C/C++ code and hardware acceleration. Our Python version doesn't fully match this level of optimization. Our JPEG encoder is not highly tuned for performance. Specifically, our entropy coding stage uses a basic version of run-length and Huffman coding, while real-world encoders build and tune Huffman tables very carefully so that frequently occurring symbol patterns get as few bits as possible [8]. Second, our bitstream layout is relatively less compact: we treat blocks and channels more independently, include extra structural information, and do not take full advantage of the interleaving and table-reuse strategies that the standard allows to minimize header and marker overhead [15]. Finally, our choices for quantization-table scaling and chroma subsampling are conservative, which leads to coefficient distributions that are slightly harder to compress than those produced by the highly optimized C-based encoder in Pillow. Even though our pipeline correctly follows the Baseline JPEG steps, these cumulative differences in entropy coding, bitstream design, and parameter tuning add up to a noticeable gap in the performance of our code [4], [8].

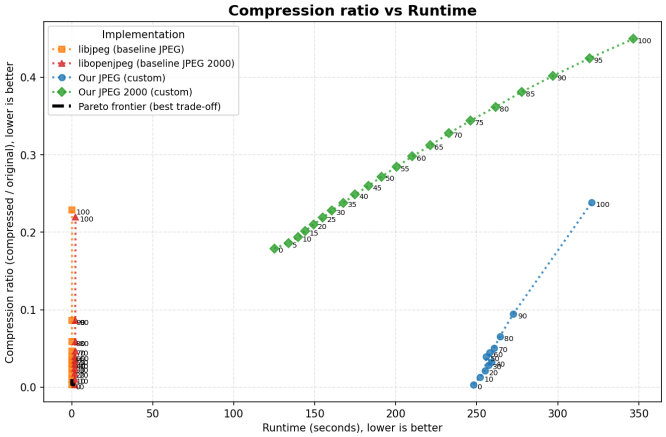

Fig. 20: Compression ratio vs. runtime plot.

As illustrated in 20, the runtime for the baseline implementations is significantly faster (up to 2500x) than our JPEG and JPEG 2000 implementations. For both JPEG and JPEG 2000 standards, the runtime was generally inhibited because the higher-level Python implementation lacks the specialized, hardware-optimized routines found in the official codecs. Specifically, we speculate that the slower runtime for our JPEG implementation can be ascribed to the fact that Pillow relies on highly optimized compiled C libraries like libjpeg-turbo to execute the complex, computationally-intensive steps, such as the DCT. Our Python implementation cannot match this speed due to the overhead of the Python interpreter and the lack of low-level optimizations like Single Instruction, Multiple Data (SIMD). Additionally, as DCT is performed on every block, sub optimal implementations (as opposed to optimized AAN DCT algorithm) aggregate across hundreds of thousands of blocks in the high resolution images, resulting in significantly slower runtimes. For JPEG 2000, the longer runtime is mainly due to the entropy coding stage, where our implementation uses adaptive arithmetic coding. This approach needs to update symbol probabilities sequentially, which adds overhead. These operations are especially slow in Python, since it lacks the low-level parallelism and bit-level optimizations used in well-tuned JPEG 2000 libraries. Our implementations shows a positive, linear relationship between compression ratio and runtime. A closer look at the builtin implementation's relationship between CR and runtime is shown in Figure 21, revealing that the baseline implementation also follows a similar trend. This linear relationship can be explained by higher quality levels preserving more details/information during the quantization steps, resulting in higher entropy tokens that are less efficiently compressed in the entropy encoding steps.

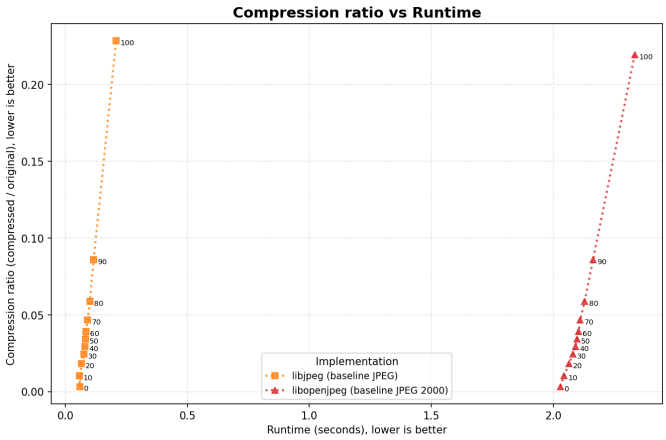

Fig. 21: Compression ratio vs. runtime plot.

VI. Conclusion

In this project, we implemented end-to-end JPEG and JPEG 2000 pipelines in Python on multi-spectral satellite images and used them to study how specific design choices shape the trade-off between compression efficiency, reconstruction quality, and runtime. For JPEG, our experiments show that "smart" chroma loss is very powerful: area-based 4:2:0 subsampling closely tracks full-resolution 4:4:4 in PSNR while achieving substantially lower bitrates, whereas nearest-neighbor 4:2:0 consistently underperforms at every bitrate. Together with the quantization and quality-level experiments, our results reinforce the view that quantization tables and chroma subsampling are the main parts that control both loss in perceived image quality and entropy-coding efficiency.

Testing the three different quantization table methods further validated previous findings of the standard method optimizing for psycho visual impact, while still achieving significant gains in compression efficiency. This analysis also further highlighted subtle conclusions about limitations of image

compression error metrics in accounting for more complex features of our visual system.

For JPEG 2000, we focused on configuration parameters in the wavelet-based pipeline. Varying the DWT deconstruction level and block size reveals clear trade-offs: choosing moderate DWT levels and block sizes improves compression ratio and reduces runtime by making coefficients more amenable to entropy coding, but very significant deconstruction or extremely large blocks can hurt either quality or efficiency. The pivot experiments further show how adaptively choosing between adaptive arithmetic coding and DEFLATE shifts the balance between compression performance and computational cost. Finally, comparing our implementations to well-optimized library code shows us the "correctness vs. optimization'" gap. Although our Python implementations reproduce the expected qualitative trends, highly tuned C-based libraries achieve much better compression and runtime at the same quality levels.

Overall, our study shows that even relatively simple implementations of JPEG and JPEG 2000 can show how color representation, transform structure, quantization strength, and entropy-coder design interact in practice, especially on a high resolution multi-spectral satellite imagery. At the same time, our findings reveal the need for not only algorithmic, but also systems-level optimization to move from prototype-level implementations to production-ready ones, since real-world performance depends strongly on efficient memory use and the optimized hardware designs.

VII. Future Work

First, we plan to implement the proposed optimizations in JPEG 2000 pipeline and re-run the experiments to evaluate their effect on performance more thoroughly. In particular, we will repeat the evaluation using a better experiment environment (e.g., CPUs with more cores) to better understand how the method scales with computational resources closer to those available in operational satellite systems. Another promising direction for improving the efficiency of the JPEG 2000 pipeline is to incorporate YCbCr-specific quantization step sizes and deadzone scaling, which may provide improved compression ratio.

Additionally, our current experiments treat every pixel as equally important, yet many real imaging tasks focus on specific regions, such as urban areas, coastlines, ice margins, or damaged infrastructure. A natural next step is to incorporate region-of-interest (ROI)-based compression, where critical regions are encoded at a higher quality while less important areas are more aggressively compressed. Both JPEG and JPEG 2000 support this conceptually, and JPEG 2000 in particular was explicitly designed with ROI mechanisms that allow different parts of an image to be reconstructed at different fidelities from a single bitstream [9].

Another critical application of satellite imagery is machine learning classification and segmentation analysis. For example, discerning whether a satellite image contains a landslide or differentiating between different land forms in an image. It would be interesting to look into the relationship and trend between quality degradation and classification accuracy, and the threshold of degradation that produces suboptimal classification results. Additionally, while our implementations are intended to be optimized for our visual system, it would be interesting to investigate how different steps could be altered to optimize instead for ML classification accuracy, and how this differs from the psycho visually motivated experiments.

Another possible direction is to perform multi-parameter sweeps (for example, jointly varying JPEG quality and chroma subsampling, or JPEG 2000 DWT level and block size) to produce more complete Pareto frontiers across compression ratio, PSNR/SSIM, and runtime. Another is to incorporate lightweight learned post-processing models that reduce compression artifacts or sharpen heavily compressed images. Finally, in addition to visual quality metrics, future evaluations could include task-oriented measures, such as the effect of compression on land-cover classification, object detection, or change-detection pipelines. These extensions would move our work toward systems that all together would be able to optimize compression efficiency, computational cost, and real-life application.

Lastly, it would be interesting to perform a multi-dimensional analysis involving the tweaking of two or more variables in the pipeline at once to better understand the interplay between each portion of the pipeline. In terms of the DWT experiments, the encoding runtime increasing at 2- and 3-level DWT is a surprising result and a further exploration of why it is happening would prove interesting.

VIII. Appendix

A. Other JPEG Experiments:

1) SSIM-BPP Curves for Chroma Subsampling: We also examined rate-distortion behavior using SSIM instead of PSNR as it is a new metric which. Figure 22 plots SSIM vs. BPP for the same three chroma modes (4:4:4, 4:2:0 nearest-neighbor, and 4:2:0 area-based), averaged across all satellite images in our dataset. SSIM (Structural Similarity Index) is a metric between 0 and 1 that focuses on structure, contrast, and texture; values closer to 1 indicate higher perceptual similarity.

The SSIM curves show the same qualitative pattern as the PSNR curves in the main text. All methods exceed SSIM 0.97 at moderate bitrates, and the difference between 4:4:4 and area-based 4:2:0 remains small. Nearest-neighbor 4:2:0 again underperforms due to harsher chroma loss and blockier artifacts. These results reinforce the conclusion that smooth chroma subsampling can provide substantial bitrate savings with minimal impact on perceived image quality.

2) DCT Coefficient Occupancy: What Survives Quantization: We wanted to see how often each DCT coefficient survives quantization, and how this depends on quality level and chroma subsampling. For every JPEG setting, we counted how many times each coefficient position (u, v) in each 8×8 block was non-zero and aggregated these counts into an 8×8 probability matrix for each channel (Y, Cb, Cr) under each

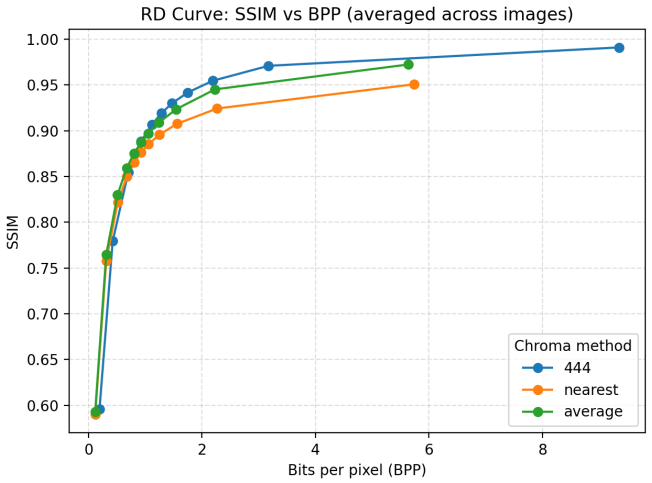

Fig. 22: SSIM vs. BPP for three chroma subsampling methods.

chroma method. These matrices are visualized as occupancy heatmaps.

Figures 23 and 24 show these heatmaps for all three channels and all chroma subsampling methods. Each heatmap is an 8×8 grid where brighter colors mean "this frequency is usually kept," and darker colors mean "this frequency is usually zero." At low quality ($Q = 10$), almost all mid- and high-frequency coefficients are removed: only the lowest frequencies in the upper-left corner remain non-zero. This happens because the quantization step is very strong at low quality, so most details are rounded to zero. The Y channel keeps slightly more low-frequency information than the chroma channels, which matches the fact that human vision is more sensitive to brightness than to color.

At high quality ($Q = 95$), many more coefficients survive quantization. Low and mid frequencies are non-zero most of the time, and even some higher-frequency values remain. The Y channel shows the highest occupancy overall, meaning it contains more texture and fine detail than Cb and Cr. Across both qualities, the differences between 4:4:4 and the two 4:2:0 methods are small. Subsampling mostly affects the chroma channels: reducing their resolution before the DCT makes their frequency content smoother, so they naturally have fewer non-zero high-frequency coefficients. However, this effect is minor compared to the much stronger influence of the quality setting.

These heatmaps show how JPEG decides where to keep or remove information in the frequency domain. Low-frequency components are kept most of the time, especially at low bitrates. High-frequency components only appear when the quality level is high. These patterns help explain the zero-run results later, since stronger quantization creates more zeros and longer zero runs for the entropy coding stage.

3) Zero-Run Distributions and Entropy Coding: We also analyzed the distribution of zero runs in the AC coefficient sequence after zig-zag ordering and quantization. A zero run is the number of consecutive zeros that appear before the next non-zero AC coefficient. For each configuration, we computed

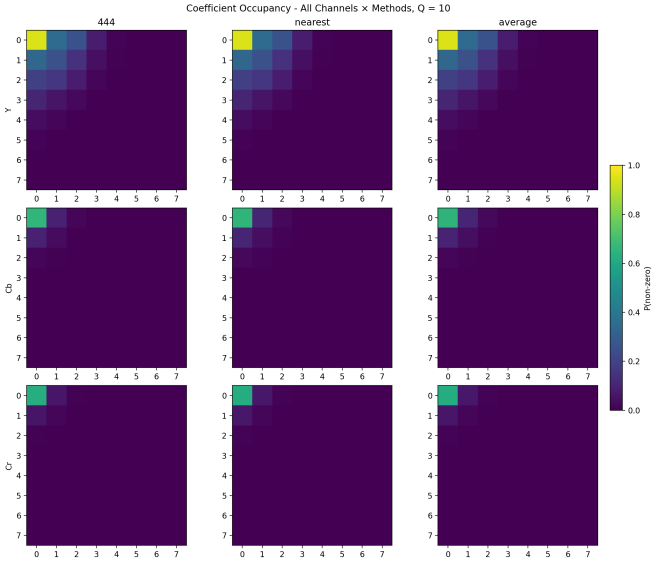

Fig. 23: Coefficient occupancy heatmaps for all channels and chroma methods at $Q = 10$. Brighter values indicate a higher probability that the DCT coefficient remains non-zero after quantization.

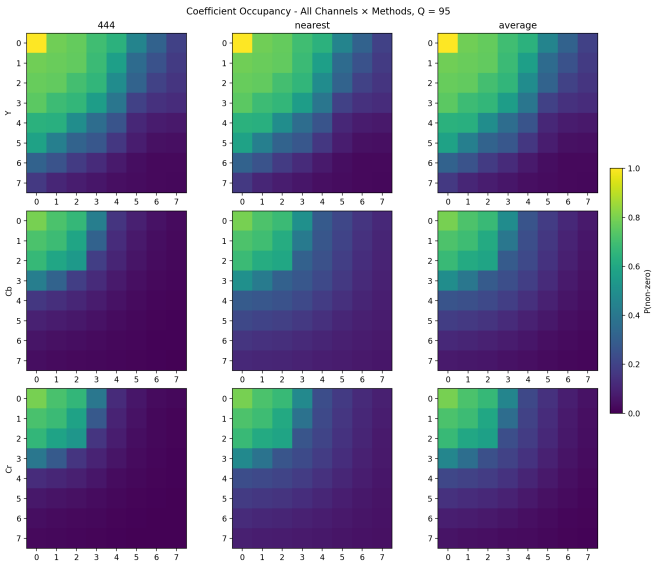

Fig. 24: Coefficient occupancy heatmaps for all channels and chroma methods at $Q = 95$. Brighter values indicate a higher probability that the DCT coefficient remains non-zero after quantization.

the histogram of zero-run lengths, the cumulative distribution function (CDF) of run lengths, and the mean zero-run length at each JPEG quality level. This shows how stronger quantization creates longer runs of zeros, how chroma subsampling affects the input to run-length and Huffman coding, and how sparsity differs between the luminance and chrominance channels.

Figures 25, 26, and 27 summarize how the lengths of zero runs change across JPEG quality levels, color channels, and chroma subsampling methods. Longer zero runs make the data easier to compress because they translate into more efficient run-length and Huffman coding.

Figure 25 shows the mean zero-run length as a function of JPEG quality for Y, Cb, and Cr. All three channels show the same trend: mean zero-run length decreases sharply as quality increases. At low quality (e.g., $Q = 10$), strong quantization removes most mid- and high-frequency details, producing long stretches of zeros. As quality increases, more coefficients survive quantization, breaking up these long zero sequences. The chroma channels (Cb and Cr) consistently exhibit longer runs than the luminance channel (Y), reflecting their sparser frequency content.

Figure 26 shows the full zero-run distributions at $Q = 10$. All channels and chroma methods have extremely long zero runs, often reaching the maximum possible length of 63. This shows that low-quality JPEG aggressively removes high-frequency information. The Cb and Cr channels are especially sparse, with nearly all their blocks having long zero runs. Differences between 4:4:4, nearest, and average subsampling are minimal here because quantization dominates the behavior at such low quality. At low quality, JPEG's quantization tables apply very large quantization steps, forcing almost all mid- and high-frequency coefficients to zero regardless of chroma subsampling or image content. Because quantization wipes out nearly all detail, it becomes the main factor controlling sparsity and zero-run behavior, and everything else has only a minor effect.

Figure 27 shows the distributions at $Q = 95$. Here the shape changes significantly: most zero runs are very short (often 0–3 coefficients), and long zero runs are rare. This shows that high-quality JPEG preserves much more frequency content, especially in the Y channel. The chroma channels still have more zeros overall, but the extreme sparsity seen at low quality disappears. Chroma subsampling again produces only small differences compared to the effect of the quality level.

Overall, these results show that JPEG's entropy coding behavior is largely driven by the quantization level: low quality produces extremely sparse coefficient sequences with long zero runs, while high quality preserves more detail and produces shorter runs. The chroma channels remain sparser than the luminance channel at all qualities, which explains why chroma information compresses more efficiently throughout the JPEG pipeline.

4) Edge Density vs Compression Efficiency: Different images have different compression rates given same quality level and resolution. As edge density of an image increases, the compression efficiency of an image decreases, explained by lower statistical redundancy in images with more edges/higher frequency structures.

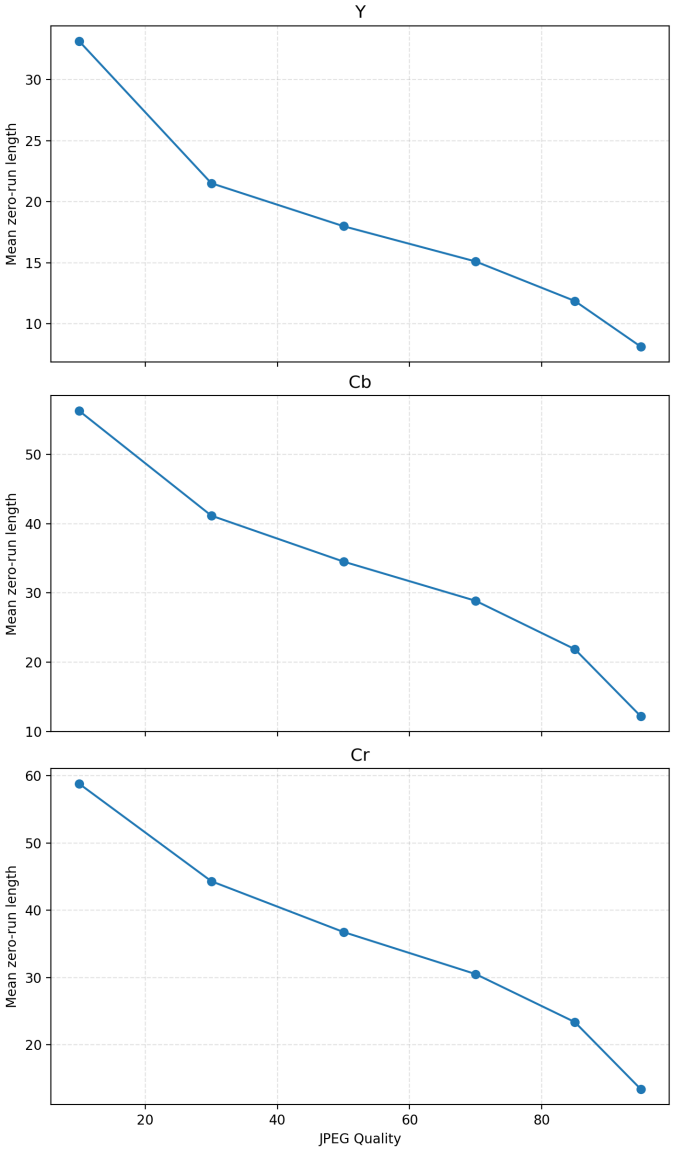

Fig. 25: Mean zero-run length vs. JPEG quality for Y, Cb, and Cr channels.

5) Comparison of Image Quality Between Quantization Methods:

B. Other JPEG 2000 experiments:

1) JPEG 2000 Pivot Test: The pivot parameter controls which entropy coding method is applied to each block in the partitioning stage. For every block, we compare the number of unique coefficient values to the pivot: blocks with coefficient diversity exceeding the pivot are encoded using adaptive arithmetic coding, while blocks with fewer unique coefficients are encoded using DEFLATE. This mechanism allows the entropy coder to adapt to varying block characteristics. In this experiment, we varied the pivot from 1 to 12 while keeping

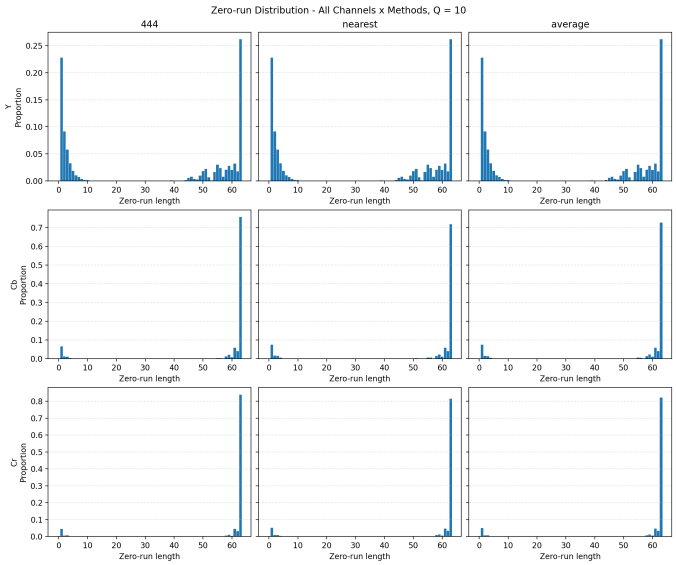

Fig. 26: Zero-run distributions for all channels and methods at $Q = 10$.

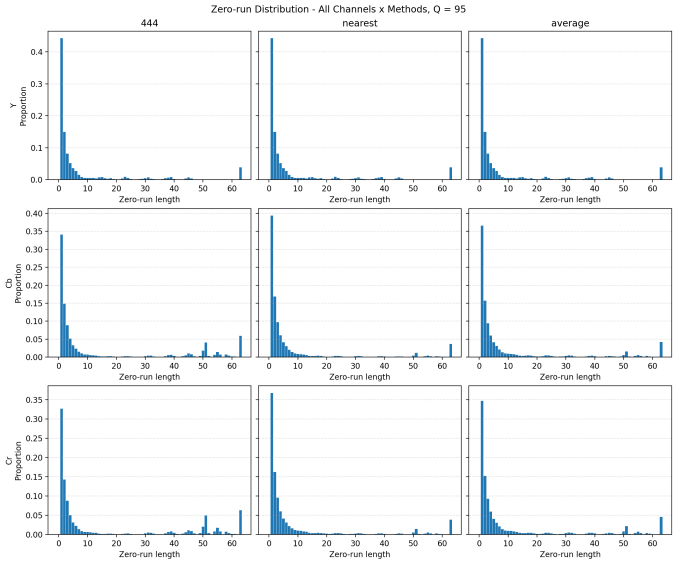

Fig. 27: Zero-run distributions for all channels and methods at $Q = 95$.

the remaining parameters, DWT level, block size, and quality level, fixed.

Figure 32 illustrates how the compression ratio changes with the pivot. We observe a steady increase in compression ratio as the pivot grows. A larger pivot classifies more blocks as low-diversity blocks. Therefore, encoding more blocks with DEFLATE rather than adaptive arithmetic coding. While DEFLATE performs extremely well on blocks with very few distinct symbols, it is less effective for the moderately diverse blocks found in most wavelet sub-bands. As more blocks switch from arithmetic coding to DEFLATE, the compressed bitstream becomes larger, leading to the growth in compression ratio.

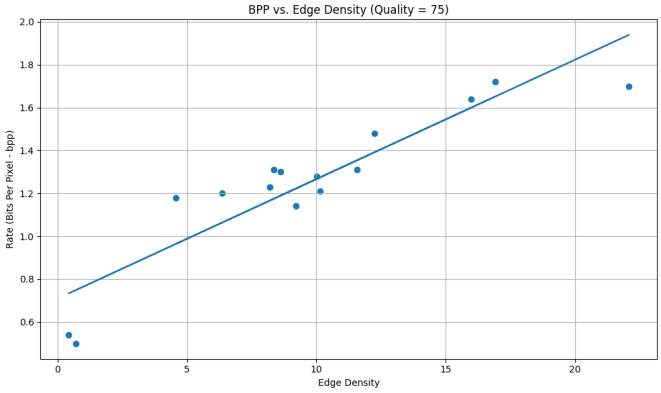

Fig. 28: Compression Efficiency based on Edge Density

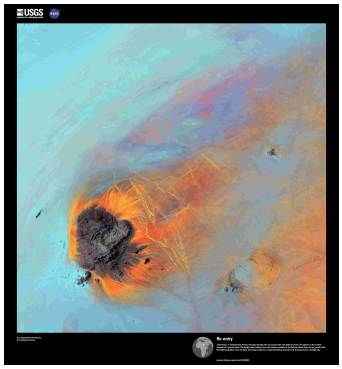

Fig. 29: Standard Quantization Table

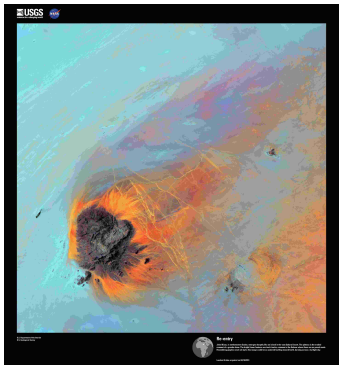

Fig. 30: 'Large flat' Quantization Table

Fig. 31: 'Small flat' Quantization Table

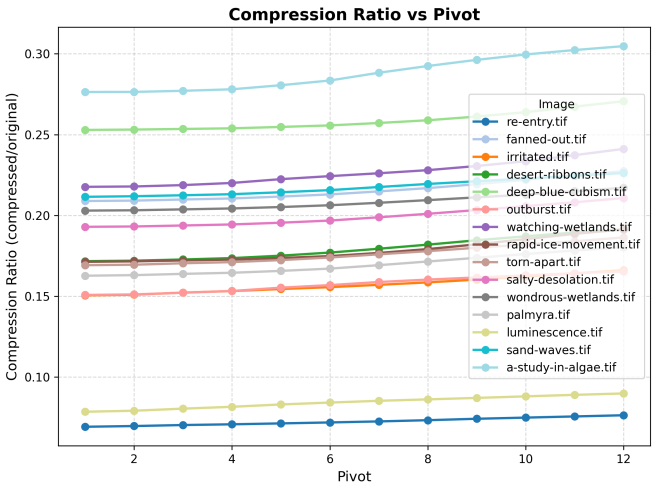

Fig. 32: Compression ratio vs. pivot plot.

Figure 33 shows the corresponding runtime behavior. In contrast to compression ratio, runtime consistently decreases as the pivot increases. Arithmetic coding is computationally expensive due to frequent probability updates, whereas DEFLATE operates primarily through fast LZ77 matching and Huffman coding. With a small pivot, many blocks are encoded with arithmetic coding, leading to higher runtime. As the pivot grows, more blocks are processed by DEFLATE, resulting in decline in encoding time.

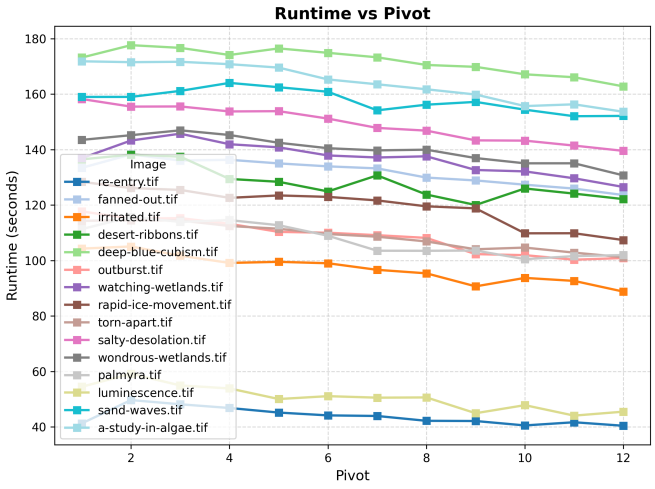

Fig. 33: Runtime vs. pivot plot.

To understand the trade-off between compression efficiency and runtime, Fig. 34 plots the pivot values on the Pareto frontier. Lower pivots yield better compression but at higher runtime, while higher pivots reduce runtime but compress less. The frontier reveals that pivots in the range of 6 to 9 offer a good balance.

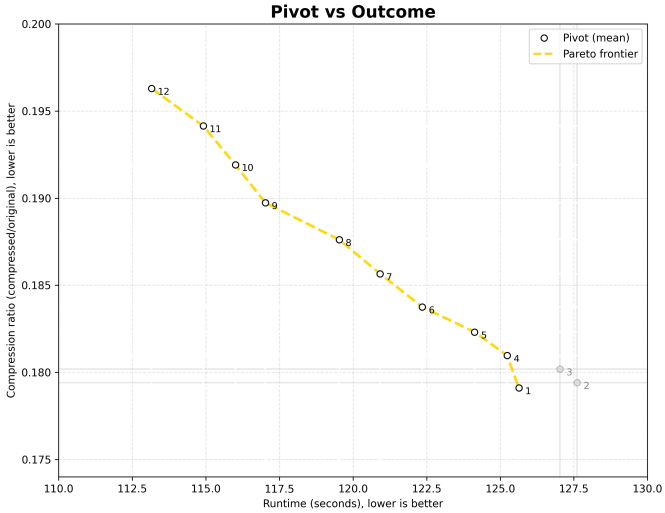

Fig. 34: Compression vs. runtime trade-off plot.

ACKNOWLEDGMENT

a) We would like to thank Professor Layla Oesper for her continuous guidance and critical feedback throughout this project. We are grateful to Mike Tie for arranging the computers for running the experiments. We want to thank each other for our collaboration, support and dedication.:

REFERENCES

[1] Cloudflare,"What is image compression?,"*Cloudflare*. [Online]. Available: https://www.cloudflare.com/learning/performance/glossary/what-is-image-compression/, Accessed: Nov. 24, 2025.

[2] R. Matsuoka, "Effect of Lossy JPEG Compression of an Image with Chromatic Aberrations on Target Measurement Accuracy," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-5, May 2014. [Online]. Available: https://doi.org/10.5194/isprsannals-II-5-235-2014

[3] G. K. Wallace, "The JPEG still picture compression standard," in IEEE Transactions on Consumer Electronics, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, doi: 10.1109/30.125072.

[4] A. Kassem, M. Hamad, and E. Haidamous, "Image compression on FPGA using DCT," in *2009 International Conference on Advances in Computational Tools for Engineering Applications*, July 2009.

[5] ScienceDirect, "Quantization Table," *ScienceDirect*. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/quantization-table, Accessed: Nov. 24, 2025.

[6] F. Ernawan and S. H. Nugraini, "THE OPTIMAL QUANTIZATION MATRICES FOR JPEG IMAGE COMPRESSION FROM PSYCHOVISUAL THRESHOLD," Journal of Theoretical and Applied Information Technology, vol. 70, no. 3, pp. 566–575, 2014.

[7] J. D. Kornblum, "Using JPEG quantization tables to identify imagery processed by software," Digital Investigation, vol.5, pp.S21–S25, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1742287608000285.

[8] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, Feb. 1992, doi: 10.1109/30.125072.

[9] D. S. Taubman and M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Springer US, 2002. doi: 10.1007/978-1-4615-0799-4.

[10] R. Eraballi, *Image Compression: JPEG (Multimedia Systems – Module 4, Lesson 1)*, lecture slides, University of Texas at Austin, 2025. [Online]. Available: https://users.ece.utexas.edu/~ryerraballi/MSB/pdfs/M4L1.pdf

[11] M. W. Marcellin, M. A. Lepley, A. Bilgin, T. J. Flohr, T. T. Chinen, and J. H. Kasner, "An overview of quantization in JPEG 2000," Signal Processing: Image Communication, vol. 17, no. 1, pp. 73–84, Jan. 2002, doi: 10.1016/S0923-5965(01)00027-3.

[12] "JPEG 2000: How does it work?" Accessed: Nov. 21, 2025. [Online]. Available: https://faculty.gvsu.edu/aboufade/web/wavelets/student_work/EF/how-works.html

[13] O. Hammami, R. Benmouhoub, and I. Aouadi, "Exploring JPEG-2000 entropy coder implementations on xilinx virtex-II pro platforms," in 2004 12th European Signal Processing Conference, Sept. 2004, pp. 2047–2050. Accessed: Oct. 17, 2025. [Online]. Available: https://ieeexplore.ieee.org/document/7080099

[14] "Earth As Art — EROS." Accessed: Oct. 03, 2025. [Online]. Available: https://eros.usgs.gov/media-gallery/earth-as-art

[15] International Telecommunication Union, *Recommendation T.81: Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines*, ITU, 1992. [Online]. Available: https://www.w3.org/Graphics/JPEG/itu-t81.pdf