Zoo

*Note: Sub-titles are not captured in Xplore and should not be used

1st Luca Araujo
Computer Science Department
Carleton College
Northfield, United States
email address or ORCID

2nd Joshua Lee
Computer Science Department
Carleton College
Northfield, United States
email address or ORCID

3rd Indy Lyness
Computer Science Department
Carleton College
Northfield, United States
email address or ORCID

4th Jiale Wan
Computer Science Department
Carleton College
Northfield, United States
email address or ORCID

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

As filesizes continue to increase, efficient compression algorithms become more important in order to keep up with the demand for transmitting large amounts of information. Images, a crucial datatype found all across the web, is a prime example of information that requires compression, especially as increasing display resolution pushes the need for higher (and more space-demanding) resolution images.

Data compression is the process of representing data in a form that occupies less space than the raw data. Compressed data is found everywhere today; almost all files that users interact with are stored in a compressed format. There are two types of data compression: lossless compression and lossy compression. Lossless compression involves representing the original data such that no information is changed or lost, with common examples being *.zip* and *.tar* files. Lossy compression, as the name suggests, permanently alters the information in order to achieve higher compression. In the case of images, lossy compression aims to discard the information that is least perceptible to humans so that compression can be more efficient while maintaining image quality. This way, the information that is lost via lossy compression is information that the human visual perception system does not need in order to recognize the image.

One of the most widely used image compression standards is the JPEG Compression Standard, which has been in use for over 30 years. This paper deals with the JPEG Compression Standard, exploring its use of different compression techniques and then tweaking its use of transform coding, a crucial part of the compression process implemented in JPEG. Originally utilizing the Discrete Cosine Transform (DCT), this project dives into other transforms by using different transforms in the JPEG pipeline to investigate their effects on various metrics used to determine the efficiency of an image compression algorithm.

II. Literature Review

One important metric used to calculate the bounds for how much a piece of data can be compressed is entropy. Not limited to just image compression, entropy describes how ”random” a string of data is, which directly correlates to the maximum level of compression it is possible to achieve. [*entropy source here*] G

III. Methods

A. pipeline

Our pipeline follows the standard structure of a JPEG-style system to test transform coding, shown in Fig. 1 (Image Compression Pipeline). Each stage prepares the image for the next, allowing us to isolate the effect of different transforms while keeping the rest of the system fixed.

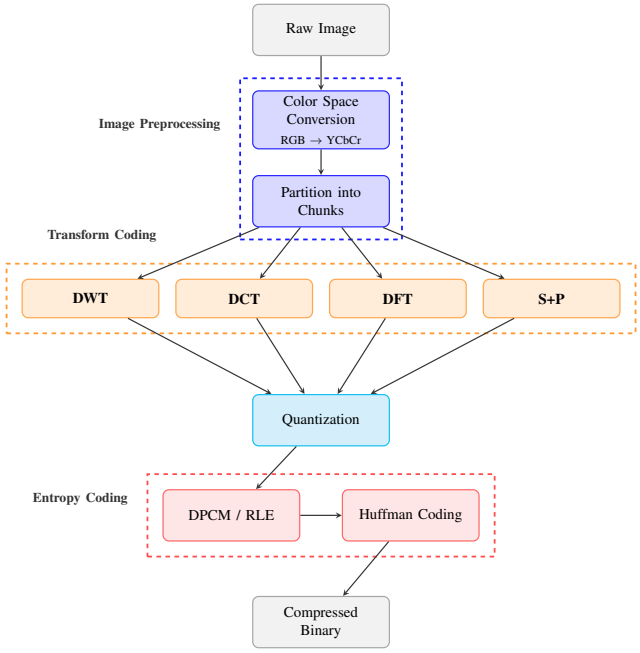


Fig. 1: Image compression pipeline.

We begin with a raw RGB image. Because RGB channels are highly correlated, we first convert the image into YCbCr, where luminance (Y) is separated from chrominance (Cb, Cr).

This step concentrates most visually important information into a single channel, making subsequent compression more effective.Next, the image is partitioned into blocks as per the specific transform(e.g., 8×8 for DCT/DFT, full-image tiles for Haar and S+P).

Then, the transform stage is the core of our investigation. Here we substitute one of four transforms (DCT, DFT, Haar, or S+P), while keeping all later steps unchanged, with the exception of routine subband-aware quantization for S+P. Each transform maps spatial pixel values into a domain where redundancy is more explicit: DCT and DFT concentrate energy into low-frequency coefficients; Haar and S+P build a multiresolution decomposition[fair to say this?]. All transforms produce a set of coefficients organized into subbands or frequency components.

These coefficients then undergo quantization, the primary source of loss in the pipeline. Quantization is a nonlinear operation that partitions the range of possible input values into disjoint intervals and assigns each interval a single representative value (the reconstruction level). All inputs falling within the same interval are encoded identically, which reduces precision and introduces quantization error. Because transforms ideally decorrelate the signal and push most meaningful information into a small number of coefficients, quantization can discard many small values with limited perceptual impact. This makes quantization the crucial stage where transform quality directly affects rate–distortion behavior.

The quantized coefficients are finally passed into an entropy coding module consisting of prediction (DPCM or linear predictive coding), run-length encoding of zeros, and Huffman coding. These operations are lossless and convert long runs of similar or repeated values, introduced by quantization, into compact binary representations.

The output of entropy coding forms the compressed bitstream. Decoding inverts each stage in reverse order. Keeping (virtually) all components except the transform fixed, this pipeline allows us to isolate how each transform influences entropy, compressibility, and reconstruction quality across datasets.

B. Background

Luca + Indy

Before diving into the individual transform methods, we first explain what transform coding is and why it helps with compression. The idea is to take the original representation of our image, a 2D array of pixel values, and transform it into a new space with lower entropy. The hope is that the result will be easily compressed by the techniques described in Section III-G.

We do this by considering some original signal, which we will denote as

$$x = \{x_1, x_2, \ldots, x_N\}. \tag{1}$$

The signal can be thought of as a vector in the vector space $\mathbb{R}^N$. Loosely speaking, a vector space V is any collection of elements where

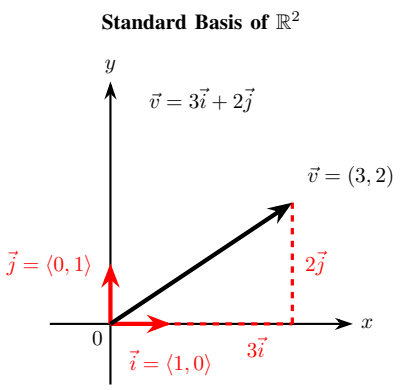


Fig. 2: A 2D basis of $\mathbb{R}^2$

1) For all $a, b \in \mathbb{R}$ and $u, v \in V$, $au + bv \in V$.
2) There is a unique element $0 \in V$ such that for all $v \in V$, $0v = 0$ and $v + 0 = v$.

In other words, it is a space that respects and is closed under the normal operation of addition and multiplication on the real numbers $\mathbb{R}$. Vector spaces have many rich theoretical properties, for which the interested reader can refer to [INSERT REFS HERE]. We will only cover the most essential properties here. A set of vectors $v_1, v_2, \ldots, v_k$ is called *linearly independent* if the only solution to the equation

$$c_1 v_1 + c_2 v_2 + \cdots + c_k v_k = 0 \tag{2}$$

is that $c_i = 0$ for $i = 1, 2, \ldots, k$. A set of linearly independent vectors $v_1, v_2, \ldots, v_N$ is called a *basis* if for any vector $u \in V$, there exists numbers $c_1, c_2, \ldots, c_N$ such that

$$u = \sum_{i=1}^{N} c_i v_i. \tag{3}$$

In other words, every member of the vector space can be represented as some weighted sum of vectors in a basis. These weights $c_1, c_2, \ldots, c_N$ are called the coefficients. Fig. 2 illustrates this with one of the simplest examples $\mathbb{R}^2$, where the vectors $\langle 0, 1 \rangle$ and $\langle 1, 0 \rangle$ form a basis. In this example, we get the equation $v = 3i + 2j$ where 3 and 2 are our coefficients. Notably, for any given vector space there are many possible bases. The transform algorithms described in this section are all effectively a *change of basis*, where the new, transformed signal is denoted

$$X = \{X_1, X_2, \ldots, X_N\}. \tag{4}$$

These new values can be thought of as the coordinates for the signal in the new transformed space. By cleverly choosing a basis, the hope is that more information of the original data is encoded in fewer of the new coefficients.

Applying this to images specifically, if we represent our image as a 2D array of pixels, the resulting transformed image can be acquired by applying the transform to each row and column, as demonstrated in Fig. 3.

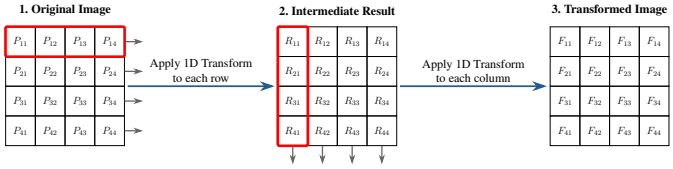


Fig. 3: Applying a transform to an image

C. Discrete Wavelet Transform

Luca

The Discrete Wavelet Transform (DWT) is a powerful transform coding technique that decomposes signals into a hierarchy of resolution levels, making it particularly well-suited for image compression. In this work we focus on the Haar wavelet, which is the simplest in the family of wavelets used for this transform. For now on we consider an image as a 1D signal, we later discuss how to convert it to 2D.

1) One Dimensional Haar Wavelet Transform in Practice: Consider a 1D signal with 2^k values:

$$\{x_1, x_2, \ldots, x_{2^k}\}$$

A naive approach to compression might be to store only the average of adjacent values, resulting in the compressed signal

$$A_{k-1} = \{(x_1 + x_2)/2, (x_3 + x_4)/2, \ldots, (x_{2^k-1} + x_{2^k})/2\}$$

Note that this approach clearly loses information. To enable perfect reconstruction of the original signal, we must also preserve the difference between adjacent values:

$$D_{k-1} = \{x_1 - (x_1 + x_2)/2, x_3 - (x_3 + x_4)/2, \ldots\}$$

Note that the difference of x_i the the mean is completely symmetric to the difference of x_{i+1} to the mean, so we only need to store one detail coefficient. This simple idea is the core of the Haar wavelet transform.

We can apply this operation once and we get a set of $2^k/2$ average coefficients and $2^k/2$ detail coefficients. We can keep applying the operation recursively to the average coefficients until we reach only one pixel of average coefficient. Table I illustrates this process for an 8-pixel signal.

Figure 4 visualizes this decomposition process, showing the signal (blue) and wavelets (red) at each resolution level. The blue curves represent the averaged signals at decreasing resolutions, while the red curves capture the detail coefficients (differences) that must be preserved for perfect reconstruction.

TABLE I: Haar decomposition example for an 8-pixel signal

Resolution	Averages	Details
8	[5, 8, 9, 7, 4, 2, 1, 3]	—
4	[6.5, 8, 3, 2]	[−1.5, 1, 1, −1]
2	[7.25, 2.5]	[−0.75, 0.5]
1	[4.875]	[2.375]

Note that for a given level we can perfectly reconstruct the previous level's average coefficients from the combination of the average coefficient and detail coefficients, so all we need to store in a compression is the set of all detail coefficients and the very last average.

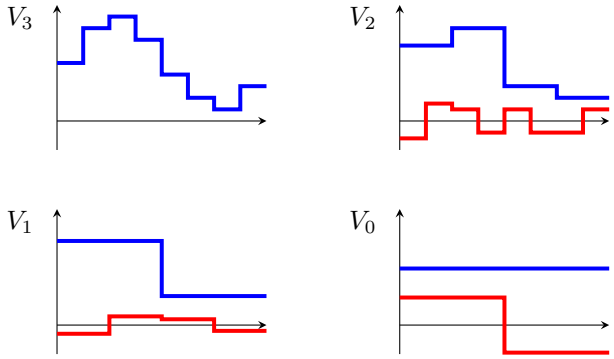


Fig. 4: Haar wavelet decomposition showing signal (blue) and wavelets (red) at each resolution level.

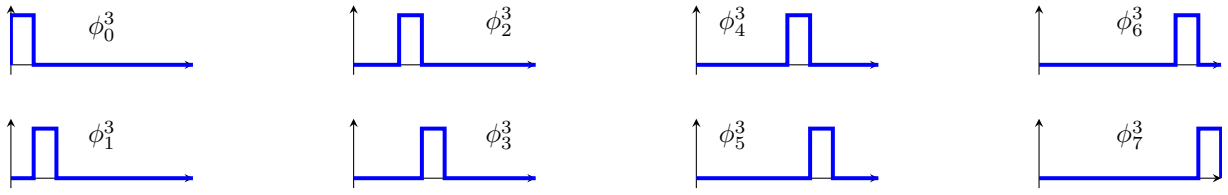
2) Theoretical Foundation for Haar Wavelet Transform: The Haar wavelet transform is founded on a set of nested vector spaces V^j representing piecewise-constant functions at different resolutions, where $V^0 \subset V^1 \subset V^2 \subset \ldots$. This nested structure enables a change of basis from the natural basis of scaling functions in V^{j+1} to a basis consisting of the scaling functions from the lower-resolution space V^j together with the wavelet basis functions from the orthogonal complement space W^j. The scaling functions provide the coarse approximation, while the wavelets capture the detail information needed for perfect reconstruction. Figure 5 visualizes the scaling functions and Haar wavelets for different resolution levels.

A detailed mathematical exposition of the vector space structure, scaling functions, inner products, orthogonality, and the Haar wavelet basis functions is provided in the appendix.

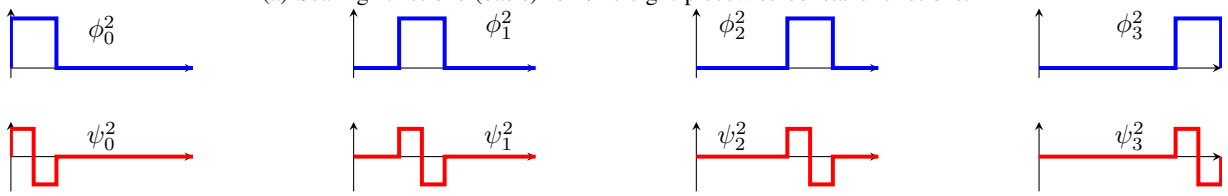
3) Two-Dimensional Haar Wavelet Transform: The 2D Haar wavelet transform generalizes the 1D "averaging and differencing" filter bank to 2D by alternating between operations on rows and columns at each level of the hierarchy.

a) The Decomposition Algorithm: The algorithm proceeds recursively as follows:

1) **Row Step:** First, perform one step of horizontal pairwise averaging and differencing on the pixel values in each row of the image. This splits the image horizontally into averages (left) and details (right).
2) **Column Step:** Next, apply vertical pairwise averaging and differencing to each column of the result from the Row Step.
   - This creates four distinct quadrants of coefficients:
     - **Top-Left:** Averages of averages (coarse approximation).
     - **Top-Right:** Averages of horizontal details.
     - **Bottom-Left:** Vertical details of horizontal averages.
     - **Bottom-Right:** Vertical details of horizontal details (diagonal details).



(a) Scaling functions (basis) for V^3: eight piecewise constant functions.



(b) Basis for V^2 (blue) and Haar wavelets basis for W^2 (red).

Fig. 5: Haar basis: scaling functions for V^3 and scaling functions and wavelets for V^2 and W^2.

3) **Recursion:** To complete the transformation, repeat this process recursively *only* on the quadrant containing the averages in both directions (the top-left quadrant).

Figure 6 visualizes this 2D Haar transform decomposition process, showing how an image is decomposed into four quadrants at each level.

Fig. 6: Visualization of a 2D Haar transform showing the decomposition of an image

4) Compression Properties: The Haar wavelet transform is particularly effective for image compression because it naturally concentrates energy in the low-resolution approximation coefficients. The detail coefficients (wavelets) typically have smaller magnitudes and can be more aggressively quantized or discarded in lossy compression scenarios. This energy compaction property, combined with the transform's computational efficiency makes the Haar wavelet an attractive choice for practical image compression systems.

D. Discrete Fourier and Cosine Transforms

Indy

Two of the most prevalent transforms in signal processing are the Discrete Fourier Transform (DFT) and its derivative, the Discrete Cosine Transform (DCT). We will first discuss the DFT and its implementation using the Fast Fourier Transform (FFT) algorithm, then show how this naturally gives rise to the DCT.

For the DFT, we consider our input as a 1-dimensional signal $x = \{x_1, \ldots, x_N\}$ as described in III-B. We define our basis to be the set of complex exponentials

$$E = \{E_0, E_1, \ldots, E_{N-1}\}, \tag{5}$$

where

$$E_k = \{e^{2\pi ik(0)/N}, e^{2\pi ik(1)/N}, \ldots, e^{2\pi ik(N-1)/N}\}. \tag{6}$$

As a reminder, complex numbers are of the form $a+bi$, where $a, b \in \mathbb{R}$ and $i = \sqrt{-1}$. Also, we have Euler's identity

$$e^{i\theta} = \cos\theta + i\sin\theta. \tag{7}$$

This means that the basis is effectively made of discrete rotations around the unit circle in the complex plane at different frequencies, as shown in

The normalized coefficients for these exponentials are then given by

$$X_k = \frac{1}{\sqrt{N}} \sum_{m=0}^{N-1} x_m e^{-2\pi ikm/N},$$

forming a new signal $X = \{X_1, X_2, \ldots, X_N\}$. We say that X lives in the **frequency domain**. In this sense, the DFT can effectively be thought of as a **change of basis** in $\mathbb{R}^N$.

E. The S+P Transform

1) Motivation and High-Level Overview: Transform coding for images typically follows a familiar pattern: a linear, approximately decorrelating transform (e.g., DCT, wavelets) , followed by quantization and entropy coding. For lossless or near-lossless compression, however, two constraints become particularly stringent: (i) the transform must be exactly invertible with integer arithmetic, and (ii) it should not inflate the

dynamic range unnecessarily. The S+P transform (Sequential transform + Prediction), originally proposed by Said and Pearlman [**?**], addresses both constraints via a simple yet powerful combination of a Haar-like multiresolution transform (the "S" stage) and an embedded predictive coding stage (the "P" stage).

In the broader context of our project, we treat S+P as a hybrid transform that lives between classical wavelet/subband decompositions and purely predictive schemes. Earlier sections introduce transform coding and Haar wavelets; here, we focus on S+P as a concrete case study that allows us to compare this hybrid design against more familiar transforms within a common pipeline.

At a high level, the S+P transform operates sequentially along one-dimensional signals (rows or columns of an image). The S stage computes lowpass (approximate local averages) and highpass (local differences) components using only integer additions, subtractions, and shifts. The P stage then applies a prediction filter to the highpass component, exploiting residual correlation that remains after the S transform. The entire process is recursively applied to the lowpass component, yielding a multiresolution pyramid analogous to a Haar wavelet decomposition, but with strictly integer-valued, in-place operations.

In our experimental compression pipeline, the S+P transform serves as one of the candidate spatial transforms applied to each image plane. It is followed by bandwise quantization and entropy coding. Our goal in this section is to expose both the theoretical structure of S+P and the concrete engineering choices that make our implementation robust and efficient enough to compare fairly with alternative transforms.

2) Sequential Transform (S) Recap: Before introducing prediction, we briefly recall the Sequential transform, which is essentially an integer variant of the Haar transform applied in a multiresolution fashion.

3) One-dimensional S transform: Let $c[n]$ be a sequence of integer samples of length N. The S transform groups samples in pairs and maps them to an approximate "sum" (lowpass) and an exact difference (highpass). One convenient integer formulation is

$$d_1[n] = c[2n+1] - c[2n], \tag{8}$$

$$s[n] = c[2n] + \left\lfloor \frac{d_1[n]}{2} \right\rfloor, \tag{9}$$

for $n = 0, \ldots, \lfloor N/2 \rfloor - 1$. If N is odd, the last sample can be carried forward as an extra lowpass sample. Here $s[n]$ plays the role of a local mean, while $d_1[n]$ stores the exact highpass detail.

This mapping is reversible with integer arithmetic. The inverse transform recovers the original pair via

$$c[2n] = s[n] - \left\lfloor \frac{d_1[n]}{2} \right\rfloor, \tag{10}$$

$$c[2n+1] = c[2n] + d_1[n]. \tag{11}$$

The key property is that no fractional bits are introduced: all operations are additions, subtractions, and integer floor-divisions by powers of two.

To make the mechanics more concrete, consider the short sequence

$$c = [100,\ 110,\ 90,\ 95].$$

We obtain

$$d_1[0] = 110 - 100 = 10, \quad s[0] = 100 + \left\lfloor \frac{10}{2} \right\rfloor = 105,$$

$$d_1[1] = 95 - 90 = 5, \quad s[1] = 90 + \left\lfloor \frac{5}{2} \right\rfloor = 92.$$

The transformed representation is thus $(s, d_1) = ([105,\ 92],\ [10,\ 5])$. Applying the inverse equations recovers the original c exactly.

Under mild correlation assumptions—in particular, when adjacent samples have correlation coefficient larger than $\frac{1}{3}$—the average variance of the low- and highpass components is strictly smaller than the variance of the original signal [**?**]. This reduction in variance is reflected in lower first-order entropy and thus improved compressibility.

4) Two-dimensional S transform and pyramid structure: To extend this transform to images, we apply the one-dimensional S transform first along rows and then along columns. After one full row–column pass, the image is partitioned into four subbands:

- LL: lowpass along rows and columns (coarse approximation),
- HL: highpass along rows, lowpass along columns,
- LH: lowpass along rows, highpass along columns,
- HH: highpass along rows and columns (fine detail).

Because HL corresponds to horizontal differences (applied along rows), it tends to emphasize *vertical* edges and structures. Conversely, LH corresponds to vertical differences (applied along columns) and tends to emphasize *horizontal* edges. The HH band behaves like a diagonal-detail channel.

The LL band is an image at half the resolution in each dimension, and its statistics closely resemble those of the original image. Consequently, we can recursively apply the same row–column S transform to LL to obtain a multilevel pyramid. At each level, we work strictly in-place: the LL, HL, LH, and HH coefficients are packed into quadrants of the same array, and no additional buffers are required beyond small temporary work arrays.

This S transform is already a valid integer multiresolution transform. However, because the highpass subbands retain nontrivial spatial correlation, significant entropy remains to be exploited. The P stage is designed precisely to target this residual correlation.

5) The S+P Transform: Prediction Embedded in the Transform: The S+P transform augments the S transform with a predictive step that operates directly on the highpass coefficients produced at each one-dimensional pass. Conceptually, we distinguish three sequences:

- $s[n]$: the lowpass component after the S stage,
- $d_1[n]$: the intermediate highpass component (exact differences),
- $d[n]$: the final highpass residual after prediction.

6) Predictor structure: Let us introduce a simple notation for local lowpass differences:

$$z_\ell[n] = s[n-1] - s[n], \tag{12}$$

where z_ℓ is intended to capture local gradients in the lowpass band. Following [**?**], we estimate $d_1[n]$ from neighboring lowpass samples and the next highpass value $d_1[n+1]$. A generic linear estimator has the form

$$\begin{aligned}\hat{d}_1[n] = \beta_{-1}(s[n-1] - s[n]) + \beta_0(s[n] - s[n+1]) \\ + \beta_{+1}(s[n+1] - s[n+2]) - \phi_1\, d_1[n+1].\end{aligned} \tag{13}$$

where $\beta_{-1}, \beta_0, \beta_{+1}, \phi_1$ are predictor coefficients. Intuitively, the first three terms exploit smoothness in the lowpass band, while the last term models correlation between adjacent highpass coefficients.

To preserve integer reversibility, we do not subtract $\hat{d}_1[n]$ directly. Instead, we perform a quantized correction using an integer denominator $K = 2^k$:

$$\mathrm{pred}[n] = \left\lfloor \frac{\hat{d}_1[n]}{K} \right\rfloor, \tag{14}$$

and define the residual

$$d[n] = d_1[n] - \mathrm{pred}[n]. \tag{15}$$

The final transformed highpass component is thus $d[n]$, and $d_1[n]$ never needs to be stored explicitly beyond the prediction step.

7) Inverse S+P and exact reconstruction: The inverse transform proceeds in the reverse order of the forward transform. During the inverse of a one-dimensional pass, we assume that $s[n]$ and $d[n]$ are available. We reconstruct $d_1[n]$ by reapplying the same predictor in reverse:

1) Visit indices in descending order $n = N/2 - 1, \ldots, 0$, ensuring that $d_1[n+1]$ is already reconstructed when needed.
2) For each n, compute the same predictor numerator using current $s[\cdot]$ and $d_1[n+1]$.
3) Recompute $\mathrm{pred}[n] = \left\lfloor \hat{d}_1[n]/K \right\rfloor$ and add it back:

$$d_1[n] = d[n] + \mathrm{pred}[n]. \tag{16}$$

Once $d_1[n]$ is recovered for all n, the inverse S transform recovers the original samples $c[2n]$ and $c[2n+1]$ exactly. The crucial point is that the predictor and its quantization are perfectly reproducible during inversion, so no information is lost.

In our implementation, we adopt a parameterization closely aligned with the "natural image" predictor family in [**?**], but we expose β_{-1}, β_0, β_{+1}, ϕ_1, and the shift k as tunable parameters. This makes it possible to experiment with both conservative predictors (small attenuation, robust on textured images) and more aggressive ones (strong low-frequency attenuation, favorable on smooth images).

8) Frequency-Domain Interpretation: Although the S+P transform is defined via local integer operations, its behavior is illuminating when viewed in the frequency domain. If we ignore truncation for the sake of analysis and treat the operations as linear, the mapping from the original sequence $c[n]$ to the highpass residual $d[n]$ can be represented as a linear time-invariant filter with a certain frequency response $F(\mathrm{e}^{j\omega})$.

The S stage alone already suppresses low frequencies in the highpass band, but it does so relatively mildly; the resulting highpass spectrum still exhibits considerable low-frequency energy due to the piecewise-constant nature of the Haar-like filter. By embedding prediction in the transform, S+P effectively implements an additional highpass filter: the predictor attempts to explain away low-frequency variations in $d_1[n]$ using neighboring lowpass differences and adjacent highpass values. When the predictor coefficients are chosen appropriately, $|F(\mathrm{e}^{j\omega})|$ exhibits significantly stronger attenuation at low frequencies than the basic S transform.

There is, of course, a trade-off. Stronger low-frequency attenuation generally comes at the cost of some amplification in higher frequencies. For smooth, low-noise images, this trade-off is advantageous: the additional whitening substantially reduces the variance and first-order entropy of $d[n]$, improving compressibility. For highly textured or noisy images, it may be beneficial to temper the predictor so as not to over-amplify high-frequency content. Said and Pearlman demonstrate that, in practice, a small set of carefully tuned predictors can work well across a broad range of natural and medical images [**?**].

In our experiments, we follow the same philosophy: we view S+P as a parametric family of integer filters in the highpass branch and use a "universal" predictor configuration as the default, with the option to specialize for particular image classes if time permits. Later sections provide empirical validation of the theoretical advantages sketched here.

9) Complexity and Runtime Considerations: From an algorithmic standpoint, the S+P transform has the same asymptotic complexity as the underlying S transform. For an image of size $W \times H$, a single S pass along rows and columns is $O(WH)$: every pixel participates in a constant number of operations. The P stage introduces only a small constant factor overhead: each highpass coefficient requires a few extra integer operations for the predictor numerator and a single integer shift.

More explicitly:

- S stage: for each pair $(c[2n], c[2n+1])$, we compute one difference and one sum with a shift. This costs $O(1)$ operations per pixel.
- P stage: for each highpass index n, we access a small fixed stencil of lowpass samples and one neighboring highpass value, perform a handful of integer multiplies/adds, and one right shift by a fixed number of bits. Again, this is $O(1)$ per highpass coefficient.

Because the transform is applied recursively only to the LL subband, which shrinks by a factor of four at each level, the total work over all levels of the pyramid remains $O(WH)$.

In comparison to more elaborate transforms such as a block DCT or longer-tap wavelets, the S+P transform is exceptionally light-weight. It can be implemented in-place with only a single scanline-sized buffer for processing columns. All coefficients remain integers, so there is no cost associated with floating-point operations or conversion. In our pipeline measurements, the S+P stage contributes only a modest fraction of total runtime; entropy coding and I/O dominate when using realistic image sizes and compression settings.

10) Implementation Details in Our Pipeline: We now describe how we integrated the S+P transform into our C++ image-compression pipeline and highlight several design decisions that distinguish our implementation from a baseline integer Haar transform.

11) Data model and in-place layout: Our pipeline operates on `Chunk` objects, each representing a rectangular block of a single image plane (Y, Cb, or Cr) with a contiguous data pointer and stride. The S+P core exposes a minimal interface: `forward2D(Plane plane)` and `inverse2D(Plane plane)`, both of which operate in-place on the provided buffer.

At each recursion level, we apply the one-dimensional forward S+P transform to all rows, then to all columns. The LL, HL, LH, and HH subbands are packed into quadrants of the same array, and we recurse only on the LL quadrant. This matches the conventional wavelet-style layout and allows us to treat the transformed plane as a collection of spatially localized subbands for quantization and entropy coding.

12) Integer arithmetic and border handling: A central goal of our implementation is to preserve exact integer reversibility while being robust to a wide range of image sizes. Two technical points are worth emphasizing:

- **Consistent floor-division for negatives.** We implement a helper `floor_divK(x, shift)` that emulates floor-division by 2^{shift} for both positive and negative integers. This eliminates subtle asymmetries that arise from naive right-shifts on negative values and is critical to making the prediction and its inverse bit-exact.
- **Configurable border policy.** The S+P core supports both clamping and mirror-padding at the boundaries. In our current configuration we default to clamping for simplicity, but the implementation exposes a `Border` enum so that we can experiment with symmetric extension if we wish to minimize edge artifacts in the transformed coefficients.

13) Stable inverse prediction at the boundaries: The inverse prediction step requires reconstructing $d_1[n]$ from residuals $d[n]$ and predicted values. In the interior of the signal, this is straightforward: $d_1[n+1]$ is available when computing $d_1[n]$. Near the right boundary, however, a naive closed-form expression that treats $d_1[n]$ and $d_1[n+1]$ symmetrically can fail for certain integer combinations, leading to small but catastrophic inconsistencies during reconstruction.

To avoid this, our implementation treats boundary positions explicitly. In the interior we reuse the analytic update rule proposed in [**?**]. At the last highpass index, we instead solve for $d_1[n]$ via a short fixed-point iteration: we initialize a candidate $d_1[n]$ and iteratively recompute the corresponding prediction until the quantized prediction stabilizes. In practice this converges in a handful of iterations and guarantees that the forward and inverse predictions are perfectly matched, even at the edges. This design choice trades a negligible amount of computation at the boundary for greatly improved robustness.

14) Comparison to a baseline Haar implementation: Relative to a simple integer Haar implementation that we used earlier in the project, the present S+P core differs in three main ways:

- It inserts the P stage on top of the basic S transform, with the explicit goal of reducing residual correlation in the highpass bands and thus lowering their entropy. The quantitative impact of this design choice is evaluated in later sections.
- It makes border handling explicit and configurable, whereas the baseline Haar version implicitly reused edge pixels in a fixed manner that was harder to reason about analytically.
- It enforces exact symmetry between forward and inverse prediction, including at the boundaries, via the fixed-point reconstruction strategy described above; the baseline Haar transform did not require such care because it lacked an embedded predictor.

These changes allow us to study S+P as a drop-in alternative to the Haar transform in our pipeline while keeping the external interface and overall complexity nearly identical.

15) Quantization for the S+P Transform: When used in a lossless mode, the S+P transform is followed directly by entropy coding, and all coefficients are preserved exactly. For lossy compression, we introduce bandwise scalar quantization on top of the S+P coefficients. Because S+P already segregates information into subbands of differing perceptual importance (LL vs. HL/LH/HH), it is natural to assign different step sizes to each band.

16) Bandwise and level-dependent parameterization: We expose a small set of quantization parameters grouped in a `QuantParams` structure:

- $q_{\text{LL}}, q_{\text{HL}}, q_{\text{LH}}, q_{\text{HH}}$: base step sizes for the four subbands,
- `deadzone`: a dimensionless multiplier for the central dead-zone around zero,
- `scale`: a global scale factor applied uniformly to all bands,
- `level_gamma`: a geometric multiplier applied per pyramid level.

From these, we compute per-level step sizes

$$q_{\text{band}}^{(L)} = \lfloor \texttt{scale} \cdot \text{level\_factor}(L) \cdot q_{\text{band}} \rfloor, \quad (17)$$

$$\text{level\_factor}(L) = \begin{cases} \texttt{level\_gamma}^{L} & \text{if } \texttt{level\_gamma} > 0, \\ 1 & \text{otherwise.} \end{cases} \quad (18)$$

for each band and pyramid level L. In practice, we often keep LL coarsely quantized or even unquantized at the coarsest lev-

els and allow more aggressive quantization in the HL/LH/HH bands, especially at finer scales where errors are less visually prominent. The choice of `deadzone` is made empirically and tuned based on observed rate–distortion performance rather than derived from a specific optimality criterion.

17) Dead-zone scalar quantization: Within each subband, we apply a standard dead-zone scalar quantizer. Given a coefficient x and step size Δ, we compute

$$q = \begin{cases} 0, & |x| < \texttt{deadzone} \cdot \Delta, \\ \operatorname{sign}(x) \left\lfloor \dfrac{|x|}{\Delta} \right\rfloor, & \text{otherwise}, \end{cases} \tag{19}$$

and reconstruct via $\hat{x} = q\Delta$ during decoding. The enlarged dead-zone around zero encourages small-magnitude coefficients—especially in highpass bands—to be quantized to zero, which in turn leads to long runs of zeros that are efficiently compressed by the entropy coder.

Because S+P already reduces the variance and entropy of the highpass residuals, we expect that competitive rate–distortion performance can be achieved with relatively simple scalar quantization. Moreover, the explicit control over band- and level-dependent step sizes gives us a clean experimental lever: we can vary $(q_{\mathrm{LL}}, q_{\mathrm{HL}}, q_{\mathrm{LH}}, q_{\mathrm{HH}})$ and `level_gamma` to probe how S+P behaves under different operating points without modifying the transform itself.

In the following sections, we present empirical results on compression ratio, entropy, PSNR, and decode time for a range of quantization settings, comparing S+P against our baseline transform choices within the same pipeline.

F. Quantization

Josh

Quantization is the lossy process of ”squashing” data from a larger range of values to a smaller range of values. By decreasing the range used to represent the data, entropy is lowered, as multiple values pre-quantization may be mapped to one value during quantization. With this mapping being many-to-one, this operation cannot be reversed, showing that quantization is necessarily lossy. With the goal of transforms being to decrease entropy while ensuring perfect reconstruction of an image, quantization is used to further improve compression ratio beyond entropy-bound compression used in lossless algorithms, sacrificing image fidelity in the process. Fig. 8 depicts the effect of increasing levels of quantization and the associated loss of image quality, along with compression ratio of the compressed file.

A central component to quantization is step size Δ, which points to the ranges of pre-quantized values that would be quantized to one resulting value. With R representing the range of data and M the number of steps, step size is given by:

$$\Delta = \frac{R}{M} \tag{20}$$

For example, given some 8-bit information, a step size of $\Delta = 2$ would result the quantized result being represented with 7 bits.

$$2 = \frac{256}{M} \tag{21}$$

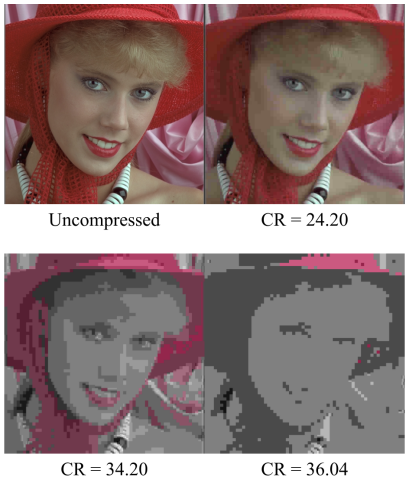


Fig. 7: Different levels of quantization on sample image with resulting compression ratio.

$$M = 128 = 2^7 \tag{22}$$

With 128 intervals post-quantization, the amount of bits required for this example would then be 7.

In this experiment, a relatively simple base quantization method was devised based on the JPEG Compression Standard. Quantization was applied on each chunk via division by a defined quantization matrix associated with each transform. DCT and DFT utilized a quantization matrix specified by the JPEG Compression Standard, whereas other transforms had quantization matrices calculated based on transformed data ranges.

Because the JPEG-specified quantization matrix is labeled as resulting in a 50% reduction in quality, the quantization methods for the Haar and S+P transforms aimed to reduce the theoretical bit-representation of the image from 8 bits to 7 bits. For the Haar transform, this was done by calculating the data ranges for each quadrant of each transformed chunk and choosing the appropriate step sizes based on $M = 128$ to achieve a basic quantization matrix. A unique quantization method was created for the S+P transform. Wan add info here

G. Entropy Coding

Josh Entropy coding is a class of compression algorithms to occur last in the pipeline. These algorithms are given their name because their effectiveness is bounded by the entropy of incoming data. In the JPEG Compression Standard, a combination of Run-Length Encoding (RLE) and Differential Pulse Code Modulation (DPCM) is used, whose results are then fed into a Huffman Coder. The areas where these algorithms are used can be observed in Fig. 9, with DPCM being used on the top left value of each chunk across all chunks in a channel (highlighted in green), and RLE being used on the remaining values in a single chunk, with the order in which values are encoded showed by the purple arrow.

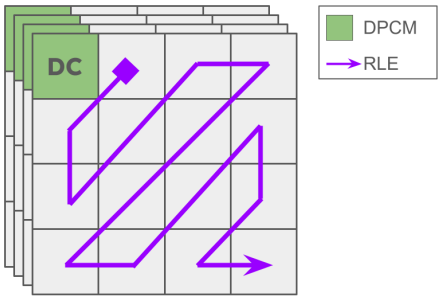


Fig. 8: Visual of different entropy encoding algorithms used for DCT as specified by JPEG Compression Standard.

Run-Length Encoding, as the name suggests, takes advantage of "runs" of data, storing the number of time a certain value or group of values repeats rather than storing every single repetition. In the case of the JPEG standard, RLE is used to group runs of zeroes in the AC components of DCT together, which appear in the data as a result of heavy quantization of DCT's high-frequency coefficients. So, each piece of data is encoded as a pair of integers, with the first being the number of preceding zeroes (up to 15) and the second being the following non-zero number (or a zero if there are 15 preceding zeroes). As an example, the string of values

$$0, 0, 0, 4, 0, 12, 0, 0, 8, 2.$$

would be encoded to

$$(3, 4); (1, 12); (2, 8); (0, 2).$$

While RLE is used for AC components, DPCM is used for DC components. At its core, DPCM is purely a differential coding algorithm, meaning it stores only the differences between values rather than the actual values themselves. This algorithm is most effective when adjacent values are correlated, which is the case for DC coefficients of adjacent image chunks (see "why images section"). By storing solely the differences between values, DPCM effectively stores values from the derivative function of the unencoded data, provided the unencoded data is correlated by some function itself. Thus, the efficiency of this algorithm depends much on the hope that the entropy of the derivative function is lower than that of the original function. Of course, to make data reconstruction possible, the first original value is stored, similarly to how integral calculation requires the constant C. To reconstruct the data, the differences are added to the first original value in sequence. For some sequence $[x]$ with difference sequence $[d]$:

$$\begin{aligned} d_0 &= x_0 \\ d_1 &= x_1 - x_0 \\ d_n &= x_n - x_{n-1} \end{aligned} \tag{23}$$

and for reconstruction:

$$\begin{aligned} x_0 &= d_0 \\ x_1 &= x_0 + d_1 \\ x_n &= x_{n-1} + d_n \end{aligned} \tag{24}$$

Related to DPCM is predictive coding, which uses previous values to predict future values and stores the difference between the actual value and the predicted value, known as prediction error. In cases when the data is highly correlated, predictions can be very accurate, causing the prediction error to be even smaller than differences calculated using DPCM, lowering entropy even further. Because of this, DPCM was substituted out for a linear predictive coder in this project pipeline. Similarly to DPCM, given some sequence $[x]$, we have predicted sequence $[p]$, with difference sequence $[d]$, where the difference sequence is stored in final data:

$$\begin{aligned} p_n &= f(x_{n-1}, x_{n-2}, ..., x_{n-k}) \\ d_n &= x_n - p_n \end{aligned} \tag{25}$$

where function $f()$ is a linear regression model using k previous values.

Luca

1) Huffman Coding: Huffman coding is an entropy encoding algorithm that serves a crucial role in the final stage of image compression pipelines. The purpose of Huffman coding is to assign variable-length binary codes to symbols based on their frequency of occurrence, with more frequent symbols receiving shorter codes and less frequent symbols receiving longer codes. This approach enables efficient representation of data by reducing the average number of bits required to encode each symbol.

In the context of image compression, Huffman coding is particularly useful because it can achieve near-optimal compression when the symbol frequencies are known. While the transform coding steps (such as DWT, DCT, and DFT) and quantization all aim at reducing the entropy of the image data by concentrating energy and removing redundancy, these steps alone do not actually compress the data if written out directly. Even after these transformations, the data still requires 8 bits per channel per pixel to represent each value. Huffman coding addresses this limitation by exploiting the non-uniform distribution of values in the transformed and quantized data, allowing frequently occurring values to be represented with fewer bits and less frequent values with more bits. This results in a reduction of the total number of bits needed to represent the data in a near-optimal way, at least when comparing to character representation with knowledge only of the character frequencies.

IV. Results

A. Experimental Setup

To compare each transform, we chose 4 unique datasets all of with different types of images.

indy: dataset wan: metrics

B. Collective Results

Josh

Firstly, as shown in fig. 10, all transforms' had their quality of compressed images compared against compression ratio. Quality of compressed images, measured by the similarity between compressed and original image, or PSNR (db), is much

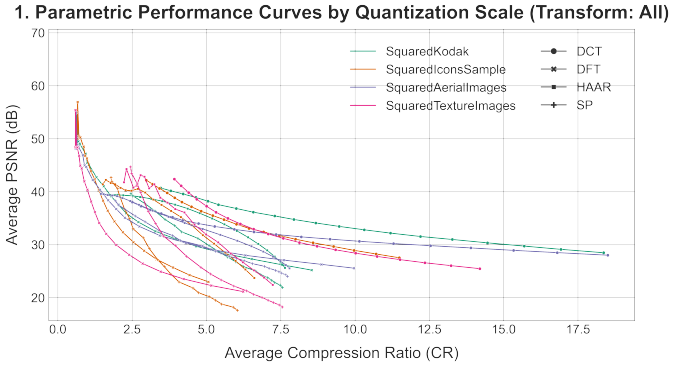


Fig. 9: Compressed image similarity based on compression ratio by transform and dataset.

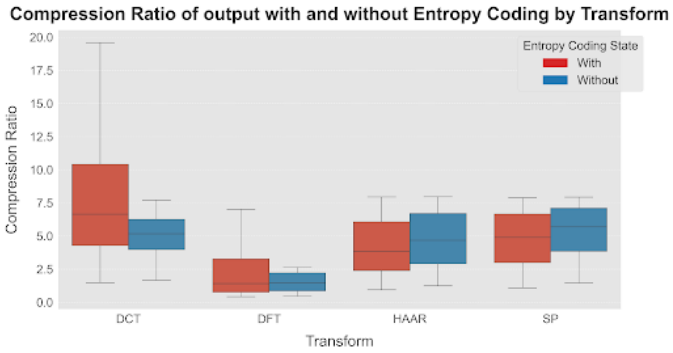


Fig. 10: Effect of entropy encoding on compression ratio for all transforms

higher for DCT, colored in blue, than all other transforms at all compression levels, with DCT additionally reaching higher compression levels than any other transform. Fig. 11 explores compression ratio by transform, with each box per transform representing compression ratio data with or without entropy coding used. For DCT and DFT, entropy coding generally increases compression ratio, with significant increases to the highest compression ratios achieved. On the other hand, Haar and SP see a flat decrease in compression ratio when entropy coding is used.

C. DWT Results

Luca

The results demonstrate that the Discrete Wavelet Transform (DWT) using the Haar wavelet basis exhibits varying performance across different image datasets. Notably, the method performed worse on the Texture Images and Icons Images datasets as quantization became more aggressive compared to other image types as can be see in Figure 11.

Figure 12 shows that the Haar wavelet transform performs slightly better than DCT on aerial images at smaller compression ratios.

In contrast, Figure 13 demonstrates that Haar performs worse than DCT on Kodak images across all compression ratio scenarios.

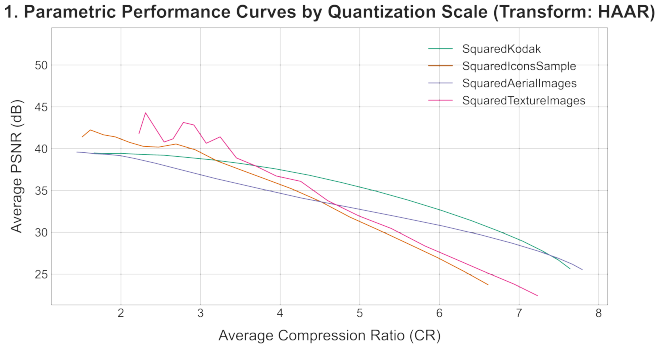


Fig. 11: Performance of the Haar wavelet transform across different datasets

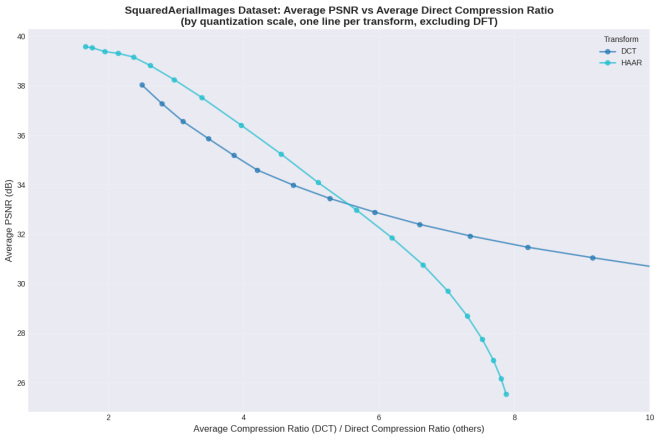


Fig. 12: Performance comparison of Haar and DCT on aerial images.

D. DCT Results

Indy

We claimed in III-D that DCT is superior to DFT for image compression. To see this clearly, Fig. 14 shows how DCT achieves much higher compression ratios at equivalent PSNR values across all datasets.

Notably, in IV-B it appears that the DCT outperforms all transform algorithms, not just the DFT. This comparison is slightly unfair since an additional step RLE was added to the entropy coding for DCT as described in III-G. Fig. 15 shows how the compression ratios change without RLE present for a more fair comparison.

This provides an excellent example of how compression algorithms can work together to achieve maximum compression for input data. Since the DCT is part of the well established JPEG pipeline, there are lots of extra tools to push the compression to the limit. Likely given more time, other techniques could be implemented to further improve algorithms like DWT and S+P as well.

E. DFT Results

Indy

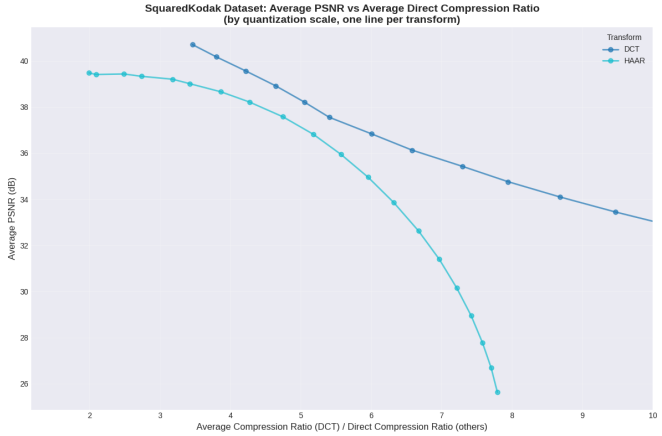


Fig. 13: Performance comparison of Haar and DCT on Kodak images.

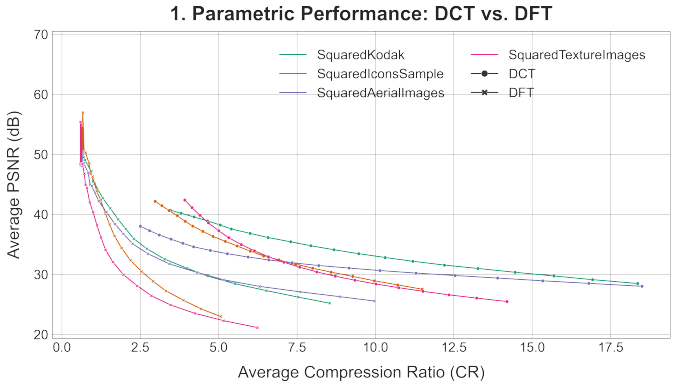


Fig. 14: Different levels of quantization on sample image with compression ratio.

This section will likely simply be merged with DCT results since there is nothing specific to just DFT to discuss.

F. SP Results

The SP results section content goes here. This file can be edited independently and will be automatically included in the main document when compiled.

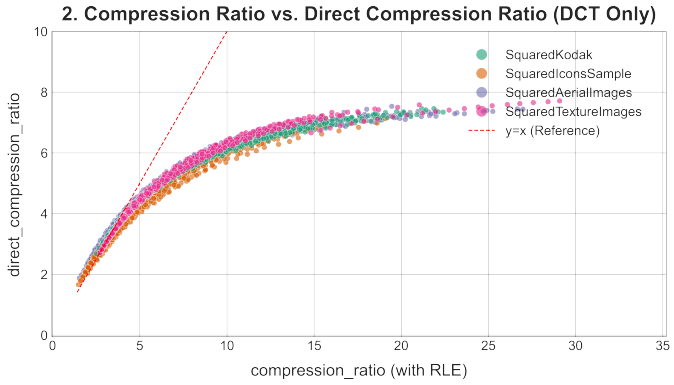


Fig. 15: Different levels of quantization on sample image with compression ratio.

V. Discussion

As shown in Figure 11 in the Results section, the Discrete Wavelet Transform (DWT) using the Haar wavelet basis exhibits degraded performance on the Texture Images and Icons Images datasets as quantization becomes more aggressive. This degradation in performance can be attributed to the finer details present in these datasets, which are more susceptible to being quantized away during the compression process. The quantization step in the compression pipeline tends to discard high-frequency components that represent fine textures and intricate icon details, leading to a more significant loss of visual information in these particular datasets compared to other image types.

The comparison between Haar and DCT on different image types, as shown in Figures 12 and 13, reveals important characteristics of the Haar wavelet transform. Haar performs better on aerial images because it operates on a global image scale, which is particularly effective for aerial images that typically contain large patches of very similar color, such as oceans or deserts. On the other hand, Haar performs worse on Kodak images since it is less optimized than DCT in the JPEG pipeline, as discussed previously. Additionally, Kodak images generally do not contain large global features of the same color as found in aerial images, which limits the effectiveness of Haar's global approach.

Appendix A
Theoretical Foundation for Haar Wavelet Transform

a) Vector Spaces and Nested Approximations: The foundation of wavelet transform can be understood through the definition of a set of nested vector spaces. We can represent any signal as piecewise-constant functions defined on the half-open interval $[0, 1)$. We utilize vector spaces, denoted as V^j, to represent these functions at different resolutions.

The space V^0 contains functions that are constant over the entire interval $[0, 1)$. A one-pixel image is an element of this space. The space V^1 contains functions with two constant pieces over the intervals $[0, 1/2)$ and $[1/2, 1)$. The space V^2 contains functions with four constant pieces over the intervals $[0, 1/4)$, $[1/4, 1/2)$, $[1/2, 3/4)$, and $[3/4, 1)$. Generalizing this, the space V^j includes all piecewise-constant functions defined on $[0, 1)$ with constant pieces over each of 2^j equal subintervals:

$$\left[0, \frac{1}{2^j}\right), \left[\frac{1}{2^j}, \frac{2}{2^j}\right), \cdots, \left[\frac{2^j - 1}{2^j}, 1\right)$$

A crucial property of these spaces is their nested structure. Because a function that is constant over a large interval is inherently constant over any subdivision of that interval, every vector in V^j is also contained in V^{j+1}. For example, a function with two intervals can be described as a function with four intervals where pairs of intervals have identical values. This creates a nested set of spaces:

$$V^0 \subset V^1 \subset V^2 \subset \dots$$

This nested structure is essential for the transform and motivates our representation of signals as this family of vector spaces.

b) Scaling Functions (The Basis for V^j): The basis functions for the spaces V^j are known as **scaling functions**, usually denoted by ϕ. A natural and simple basis for V^j is derived from the scaled and translated "box" function $\phi(x)$, defined as:

$$\phi(x) := \begin{cases} 1 & \text{for } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

For a specific resolution level j, the basis functions ϕ_i^j are defined by scaling and translating $\phi(x)$:

$$\phi_i^j(x) := \phi(2^j x - i), \quad i = 0, \ldots, 2^j - 1$$

A visualization for this basis in V^3 is shown in Figure 5(a).

c) Inner Products and Orthogonality: To define wavelets, we must first define an **inner product** to measure the relationship between functions. The standard inner product for two elements $f, g \in V^j$ is defined as:

$$\langle f|g\rangle := \int_0^1 f(x)g(x)dx$$

Using this inner product, we define a new vector space W^j as the **orthogonal complement** of V^j in V^{j+1}. In other words, we will let W^j be the space of all functions in V^{j+1} that are orthogonal to all functions in V^j under the chosen inner product. Informally, we can think of the wavelets in W^j as a means for representing the parts of a function in V^{j+1} that cannot be represented in V^j.

A collection of linearly independent functions $\psi_i^j(x)$ spanning W^j are called **wavelets**. These basis functions have two important properties:

1) The basis functions ψ_i^j of W^j, together with the basis functions ϕ_i^j of V^j, form a basis for V^{j+1}.
2) Every basis function ψ_i^j of W^j is orthogonal to every basis function ϕ_i^j of V^j under the chosen inner product.

Thus, the "detail coefficients" are really coefficients of the wavelet basis functions.

d) The Haar Wavelet Basis Functions: The wavelets corresponding to the box basis are called **Haar wavelets**. The "mother wavelet" $\psi(x)$ is defined as:

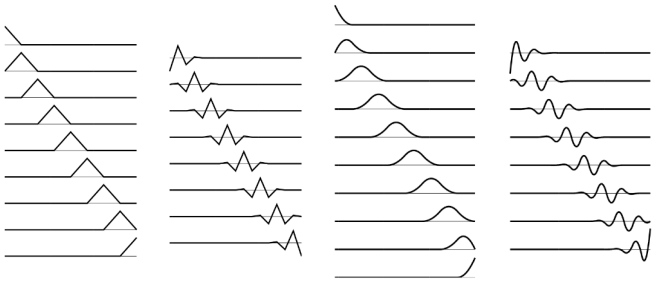
$$\psi(x) := \begin{cases} 1 & \text{for } 0 \leq x < 1/2 \\ -1 & \text{for } 1/2 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

Similar to the scaling functions, the Haar wavelets for a specific level j are generated by scaling and translating this function:

$$\psi_i^j(x) := \psi(2^j x - i), \quad i = 0, \ldots, 2^j - 1$$

Figure 5(b) illustrates the relationship between the scaling functions (blue) and the Haar wavelet basis functions (red) for W^2. The wavelets capture the detail information that cannot be represented in the lower-resolution space V^2.

e) Generalizing to Other Basis: This illustration of this specific vector space of piecewise constant functions is unique to the Haar Wavelets, however we can similarly define other vector spaces and inner products to work with different wavelets. The essential part illustrated by Haar is that the vector spaces must be nested and the definition of the wavelet space as the orthogonal complement of the scaling functions inside the next resolution level. This general idea of energy compactation into the lower resolution's basis and using the wavelets as the detail coefficient remain the same regardless of the basis. Figure 16 demonstrates alternative wavelet bases using linear and quadratic splines.



(a) Linear B-spline scaling and corresponding wavelet functions

(b) Quadratic B-spline scaling and corresponding wavelet functions

Fig. 16: Alternative wavelet bases demonstrating different spline functions: linear and quadratic splines, showing alternatives to the Haar wavelets. Figures adapted from [20]

It is somewhat standard to treat the space as some space of functions from the half open interval $[0, 1)$ when dealing with images because we can use the standard inner product previously defined very effectively but there is nothing that strictly requires such representation.

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

Please number citations consecutively within brackets [**?**]. The sentence punctuation follows the bracket [**?**]. Refer simply to the reference number, as in [**?**]—do not use "Ref. [**?**]" or "reference [**?**]" except at the beginning of a sentence: "Reference [**?**] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [**?**]. Papers that have been accepted for

publication should be cited as "in press" [**?**]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [**?**].

REFERENCES

[1] K. Sayood, *Introduction to Data Compression*, 4th ed. Elsevier, 2012.

[2] S. A. Khayam, "The discrete cosine transform (DCT): theory and application," *Michigan State University*, vol. 114, no. 1, p. 31, 2003.

[3] J. Han *et al.*, "A Technical Overview of AV1," *Proceedings of the IEEE*, vol. 109, no. 9, pp. 1435–1462, Sept. 2021.

[4] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *Journal of King Saud University - Computer and Information Sciences*, vol. 33, no. 2, pp. 119–140, Feb. 2021.

[5] M. K. Ibraheem, "A Comprehensive Literature Review on Image and Video Compression: Trends, Algorithms, and Techniques," *IIETA*. [Online]. Available: https://iieta.org/journals/isi/paper/10.18280/isi.290307.

[6] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.

[7] S. R. D. Sivakumar, "Development of a Modified Predictive Coding Algorithm for High-Resolution Image Compression in Real-Time Structural Health Monitoring of Metallic Component," *Metallurgical and Materials Engineering*, vol. 31, no. 4, pp. 76–83, Apr. 2025.

[8] R. M. Rad, A. Attar, and A. Shahbahrami, "A predictive algorithm for multimedia data compression," *Multimedia Systems*, vol. 19, no. 2, pp. 103–115, Mar. 2013.

[9] A. Said and W. A. Pearlman, "Reversible image compression via multiresolution representation and predictive coding," in *Visual Communications and Image Processing '93*, vol. 2094. SPIE, Oct. 1993, pp. 664–674.

[10] J. Lee, S. Cho, and M. Kim, "An End-to-End Joint Learning Scheme of Image Compression and Quality Enhancement with Improved Entropy Minimization," *arXiv preprint arXiv:1912.12817*, Mar. 2020.

[11] T. Chlubna and P. Zemčík, "Comparative survey of image compression methods across different pixel formats and bit depths," *Signal, Image and Video Processing*, vol. 19, no. 12, p. 981, Sep. 2025.

[12] G. K. Wallace, "The JPEG Still Picture Compression Standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, Feb. 1992.

[13] A. M. Raid, W. M. Khedr, M. A. El-dosuky, and W. Ahmed, "Jpeg Image Compression Using Discrete Cosine Transform - A Survey," *arXiv preprint arXiv:1405.6147*, May 2014.

[14] A. Mali, A. Ororbia, D. Kifer, and L. Giles, "Neural JPEG: End-to-End Image Compression Leveraging a Standard JPEG Encoder-Decoder," *arXiv preprint arXiv:2201.11795*, Jan. 2022.

[15] A. Skodras, *Discrete Wavelet Transform: An Introduction*. Jan. 2003.

[16] R. A. DeVore, B. Jawerth, and B. J. Lucier, "Image compression through wavelet transform coding," *IEEE Transactions on Information Theory*, vol. 38, no. 2, pp. 719–746, Mar. 1992.

[17] S. A. Broughton and K. Bryan, *Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing*. Hoboken, NJ: John Wiley & Sons, 2009.

[18] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. Academic Press, 2009.

[19] D. H. Salesin, T. D. DeRose and E. J. Stollnitz, *Wavelets for Computer Graphics: A Primer, Part 1* in IEEE Computer Graphics and Applications, vol. 15, no. 03, pp. 76-84, May 1995, doi: 10.1109/38.376616.

[20] D. H. Salesin, T. D. DeRose and E. J. Stollnitz, *Wavelets for Computer Graphics: A Primer, Part 2* in IEEE Computer Graphics and Applications, vol. 15, no. 04, pp. 75-85, July 1995, doi: 10.1109/38.391497.