Indian Institute of Technology Bombay

CS 663: Fundamentals of Digital Image Processing

Autumn 2024

Image Compression

Using 2D DCT and Sparse Orthonormal Transforms

Asmith Reddy - 22B0663

Vishal Gautam - 22B0065

Ananya Nawale - 21D170004

Guide - Prof. Ajit Rajwade

Abstract

The aim of this project is to explore and implement image compression techniques, starting with the 2D DCT JPEG algorithm, and to enhance it using modern techniques such as sparse orthonormal transforms. The project involves a step-by-step implementation of JPEG compression, including the 2D Discrete Cosine Transform (DCT), quantization, Huffman encoding. Additionally, the algorithm's effectiveness is analyzed through the RMSE (Root Mean Squared Error) versus BPP (Bits Per Pixel) curve.

Enhancements to the JPEG algorithm are proposed using sparse orthonormal transforms, inspired by recent advancements in image processing. These enhancements are aimed at improving the compression quality while maintaining computational complexity.

Contents

1 Introduction

Image compression is a critical process in modern digital systems, enabling the efficient storage and transmission of image data without compromising visual quality to a significant extent. It is achieved by reducing redundancy in image representation while maintaining essential information, ensuring that compressed images occupy less storage space and require fewer resources for transmission. Compression methods can be categorized into two types: lossless compression, which preserves all original data perfectly, and lossy compression, which achieves higher compression ratios by discarding less noticeable image details.

The JPEG standard is one of the most widely used lossy image compression methods. It relies on a sequence of steps, including the Discrete Cosine Transform (DCT), quantization, and Huffman encoding, to achieve significant reductions in file size while maintaining acceptable visual quality.

Here, we aim to implement a basic JPEG compression engine, using 2D DCT, quantization, and Huffman encoding. Additionally, we want to explode enhancing this implementation by exploring advanced techniques inspired by the IEEE paper **"Approximation and Compression With Sparse Orthonormal Transforms" by Sezer, Guleryuz, and Altunbasak.**

Basically, this project includes:

- Developing a functional JPEG compression engine capable of handling grayscale images.
- Testing the algorithm across a diverse dataset of grayscale images.
- Performing a detailed analysis of results through metrics such as RMSE (Root Mean Square Error) versus BPP (Bits Per Pixel).
- Comparing the implemented JPEG compression to a PCA-based compression technique, focusing on specific image classes.
- Implementing sparse orthonormal transforms and evaluating their impact on compression performance.

2 JPEG Compression

JPEG (Joint Photographic Experts Group) is one of the most widely used standards for lossy image compression, particularly for photographic and natural images. It achieves significant compression ratios by exploiting human visual perception, which is less sensitive to high-frequency content. The JPEG algorithm combines mathematical transformations, quantization, and entropy coding to reduce the storage size of images.

2.1 Steps in JPEG Compression

1. **Image Division into Blocks:** The image is divided into non-overlapping 8×8 blocks. This step localizes the processing, allowing efficient compression by focusing on smaller regions.
2. **Discrete Cosine Transform (DCT):** Each 8×8 block undergoes a 2D Discrete Cosine Transform (DCT). This mathematical operation expresses the block as a sum of cosine functions oscillating at different frequencies. The DCT separates the block into spatial frequency components, concentrating most of the image's energy in a few low-frequency coefficients.
3. **Quantization:** Quantization is the step where compression becomes lossy. Each DCT coefficient is divided by a corresponding value in a quantization matrix and rounded to the nearest integer. Larger quantization values are used for high-frequency components, as these are less perceptible to the human eye.
4. **Entropy Coding:** After quantization, the coefficients are arranged in a zig-zag order to group zeros together. This grouping is followed by two forms of lossless compression:
   - *Run-Length Encoding (RLE):* Encodes sequences of zeros efficiently.
   - *Huffman Encoding:* Assigns shorter binary codes to more frequent coefficients, reducing the overall bit length of the encoded data.
5. **File Packaging:** The compressed data is stored along with metadata such as the quantization matrix, Huffman tables, and image dimensions.

2.2 What is Discrete Cosine Transform (DCT)?

The Discrete Cosine Transform (DCT) converts the spatial representation of an 8×8 image block into the frequency domain. Mathematically, the 2D DCT for an $M \times N$ block is defined as:

$$C(u,v) = \frac{1}{\sqrt{MN}}\alpha(u)\alpha(v)\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{\pi(2x+1)u}{2M}\right]\cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

where:

- $C(u, v)$ are the DCT coefficients.
- $f(x, y)$ is the pixel intensity at position (x, y).
- $\alpha(u) = \sqrt{1/M}$ for $u = 0$ and $\sqrt{2/M}$ otherwise.

The DCT coefficients represent the amplitudes of cosine waves at different frequencies:

- **DC Coefficient:** Corresponds to the average intensity of the block.
- **AC Coefficients:** Represent variations and details in the block.

2.2.1 Implementation of Discrete Cosine Transform (DCT) in JPEG Compression

The implementation of the JPEG compression algorithm utilizes the Discrete Cosine Transform (DCT) to convert image data from the spatial domain to the frequency domain. This section describes the major steps involved in the implementation, focusing on how the DCT and subsequent operations are carried out to achieve compression.

Preprocessing: The input image is first preprocessed to ensure compatibility with the JPEG compression requirements:

- The image is converted to grayscale if it is not already in that format.
- Pixel values are shifted by subtracting 128, centering them around zero within the range $[-128, 127]$.
- To meet JPEG's requirement for block processing, the image dimensions are adjusted to be multiples of 8 by cropping excess pixels.

Block Processing: The image is divided into non-overlapping 8×8 blocks using a function called `blockify`. Each block is processed independently, following these steps:

- A 2D Discrete Cosine Transform (DCT) is applied to each block. The DCT separates the block into low- and high-frequency components, concentrating most of the image's energy in the top-left corner of the frequency matrix.
- Quantization is performed on the DCT coefficients using a standard luminance quantization matrix scaled by the chosen quality factor (Q). Larger quantization values lead to higher compression at the cost of reduced image fidelity.

Entropy Coding: After quantization:

- The DC coefficient (top-left value of each block) is computed as the difference from the previous block's DC coefficient. This differential encoding reduces redundancy across blocks.
- The AC coefficients (remaining values) are traversed in a zigzag order to group zero values together, facilitating efficient encoding.
- Huffman encoding is applied to both the DC and AC coefficients to further compress the data. Run-length encoding (RLE) is used for sequences of zeros in the AC coefficients before Huffman coding.

Reconstruction: To decompress the image:

- The compressed data is decoded, reversing the Huffman and run-length encoding to recover the quantized DCT coefficients.
- The coefficients are dequantized using the quantization matrix, and the inverse DCT (IDCT) is applied to reconstruct the spatial representation of each block.
- The blocks are reassembled into the full image using the `unblockify` function. A postprocessing step shifts pixel values back to the range $[0, 255]$, ensuring the reconstructed image is visually comparable to the original.

3 JPEG Compression Results

3.1 Results of Implemented JPEG Compression

3.1.1 Compressed Image Results

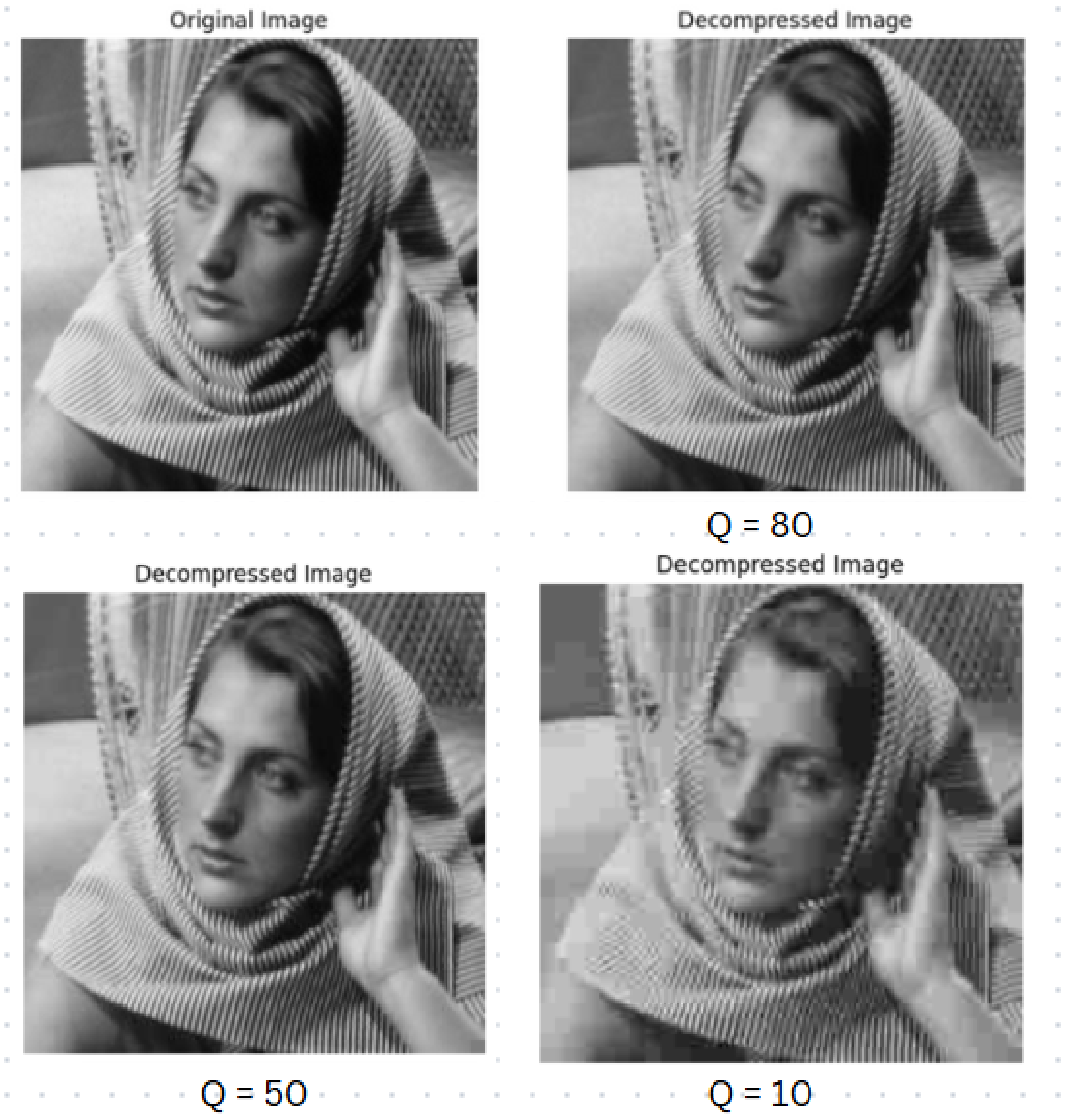

Figure 1: Compression Results for Q = 80,50,10

3.1.2 RMSE vs BPP Plots for 20 different images

Figures 2 to 6 demonstrate the relationship between Root Mean Square Error (RMSE) and Bits Per Pixel (BPP) for different compression settings.

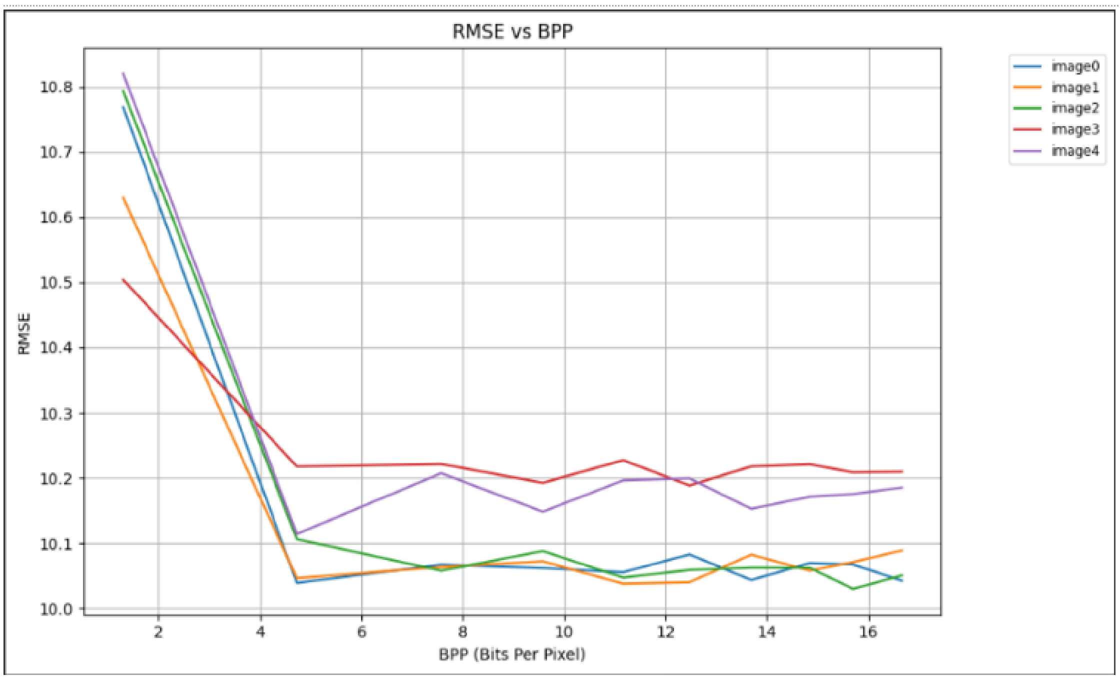

Figure 2: RMSE vs BPP Plot 1

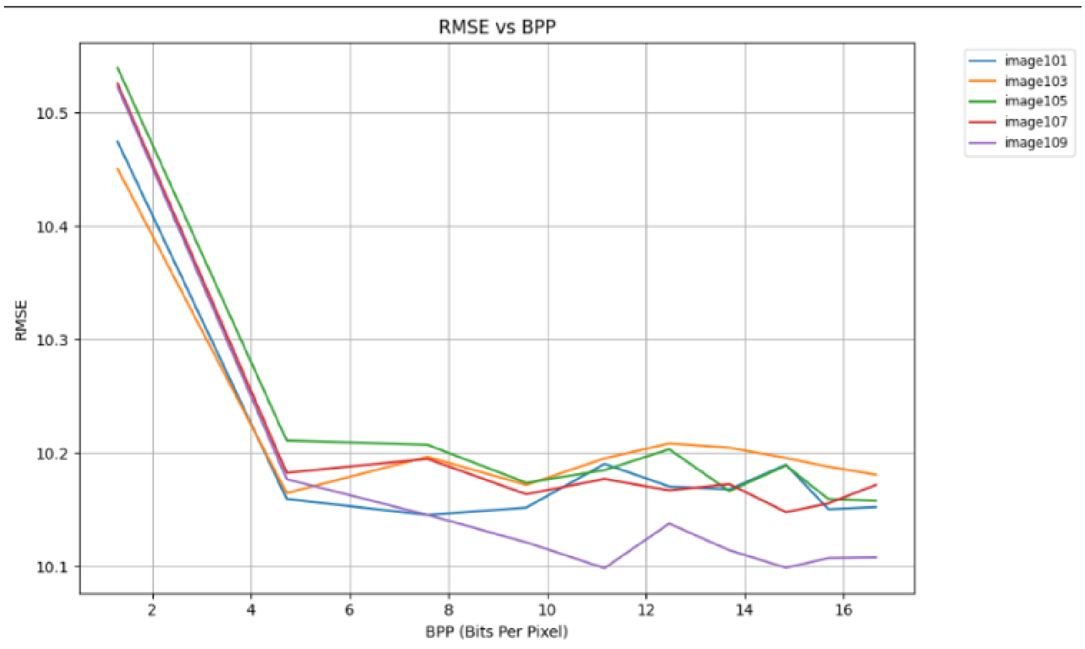

Figure 3: RMSE vs BPP Plot 2

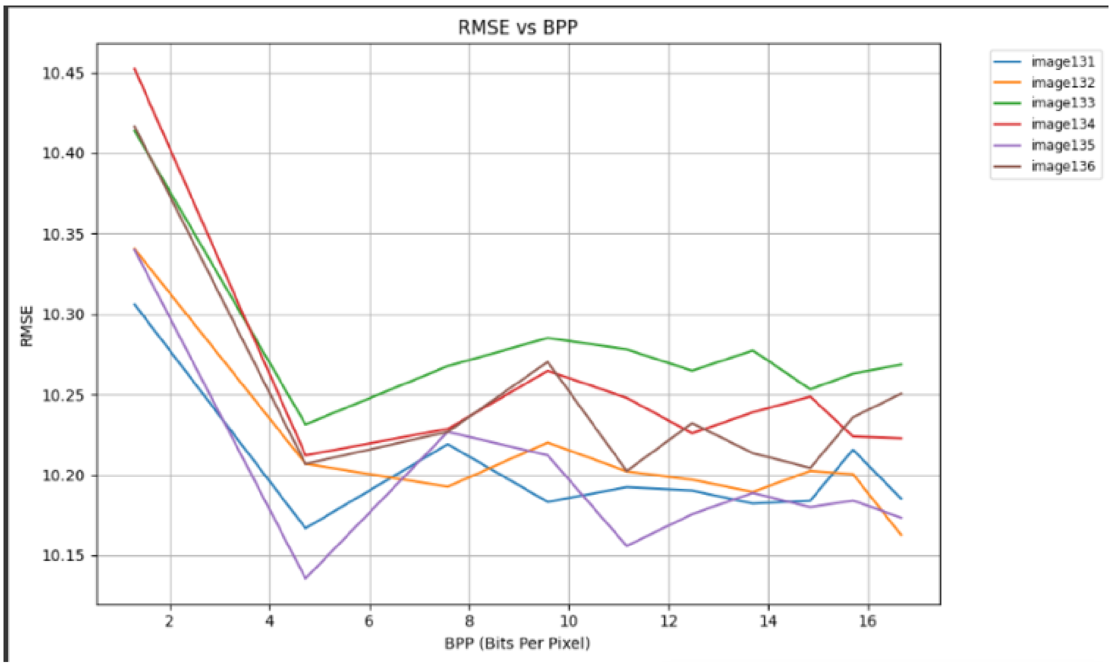

Figure 4: RMSE vs BPP Plot 3

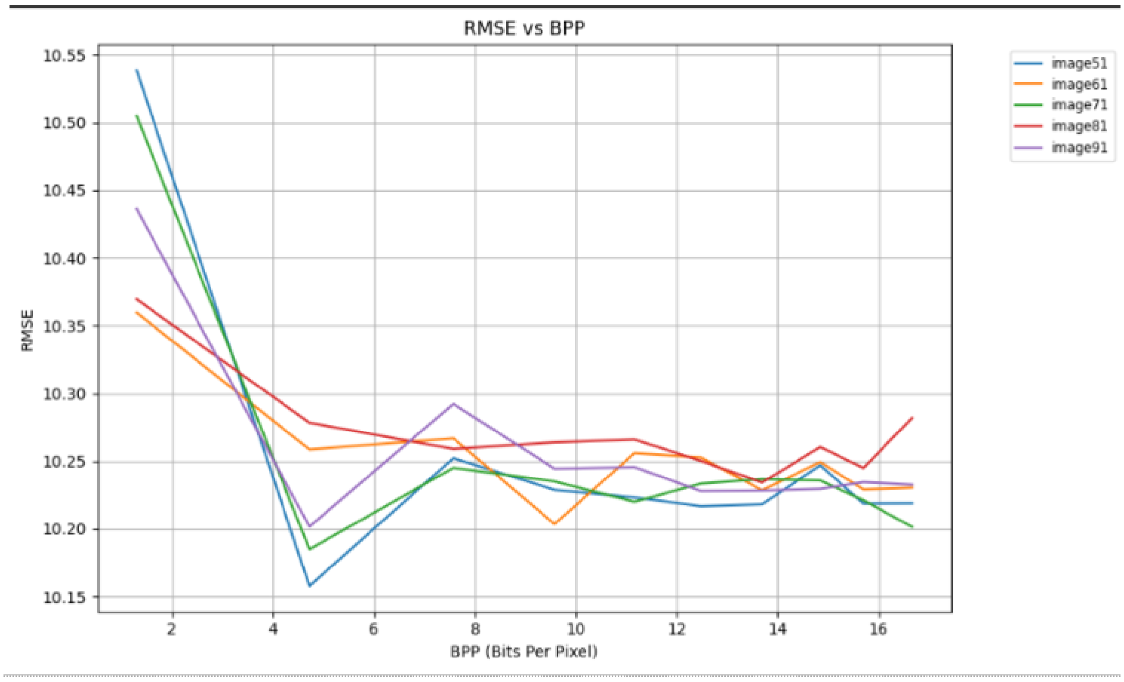

Figure 5: RMSE vs BPP Plot 4

3.1.3 Comparison with Existing JPEG Compression

Figure 6 presents a comparison of RMSE vs BPP between the implemented JPEG compression and existing JPEG compression methods.

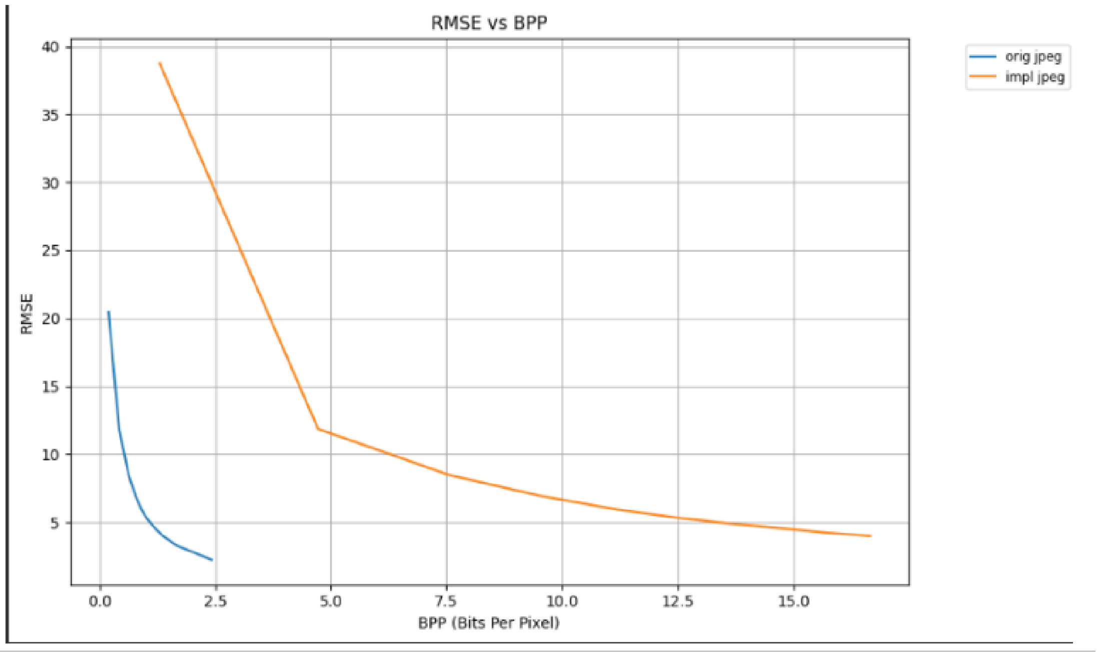

Figure 6: RMSE vs BPP: Implemented vs Existing JPEG Compression

4 PCA-Based Image Compression

4.1 Introduction to PCA-Based Image Compression

Principal Component Analysis (PCA) is a dimensionality reduction technique that can be applied to image compression. The core idea is to transform the original image into a new coordinate system where the data can be represented with fewer dimensions while retaining most of the original variance.

The PCA-based image compression process involves the following steps:

- Normalize the image data by subtracting the mean
- Compute the covariance matrix
- Calculate eigenvectors and eigenvalues
- Select top principal components
- Project the image onto the selected components
- Reconstruct the image using the reduced representation

4.2 Results of PCA-Based Image Compression

4.2.1 Compressed Image Results

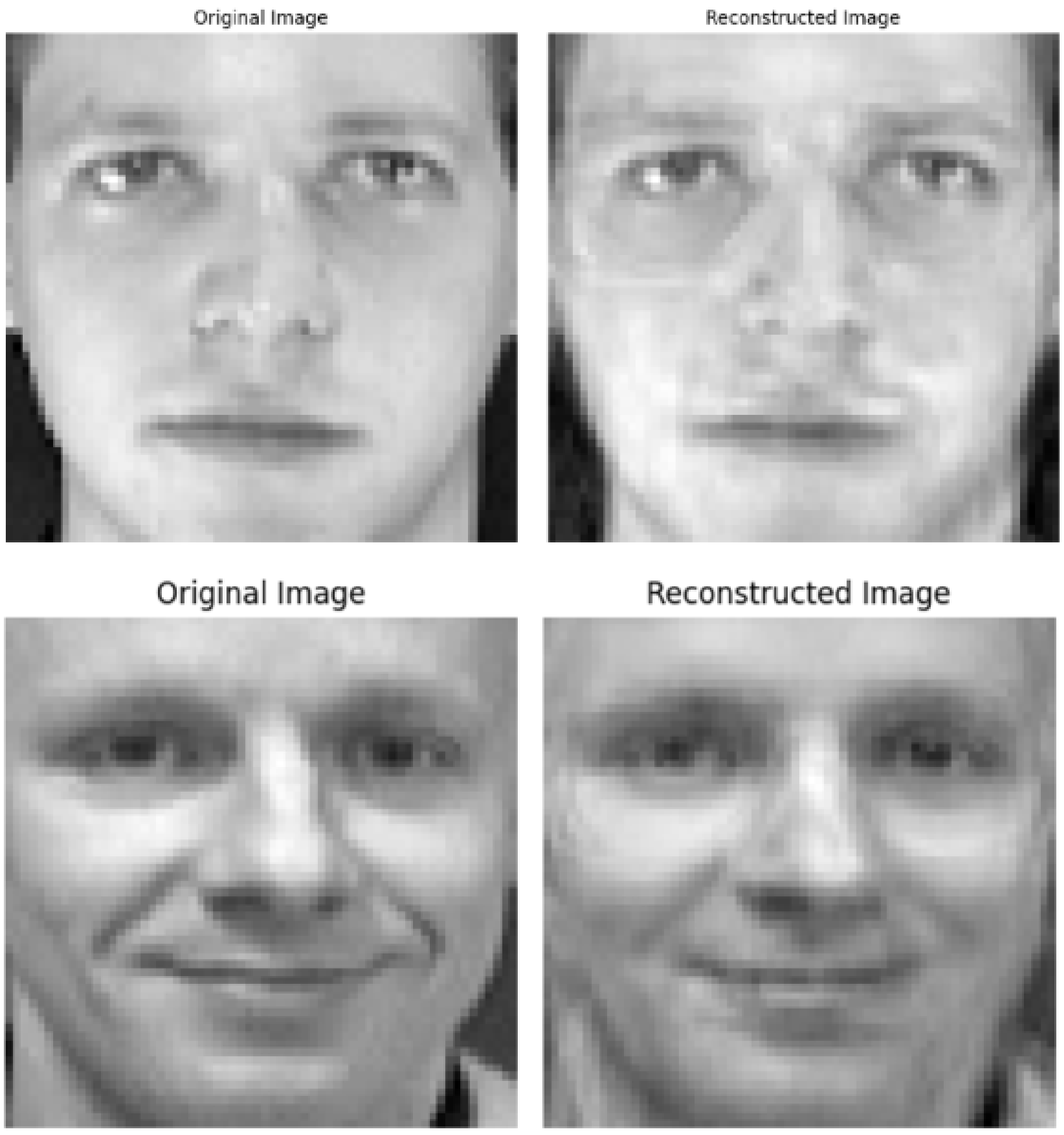

Figure 7: PCA Compression Result for 95% varaince retained

4.2.2 Compression Ratio vs Number of Components

Figure 8 illustrates the relationship between the number of principal components and the compression ratio.

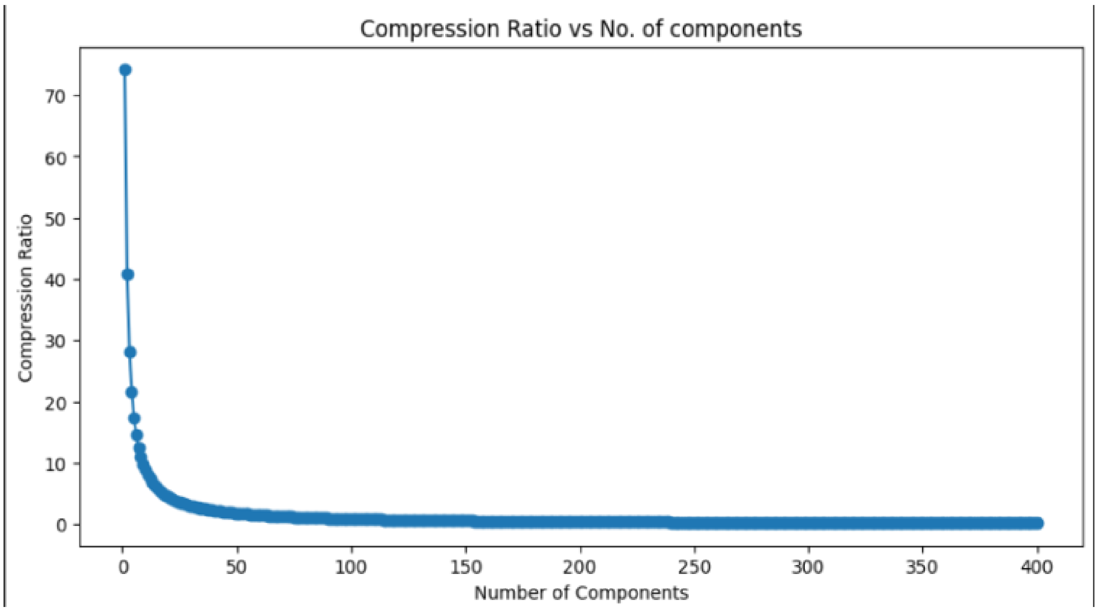

Figure 8: Compression Ratio vs Number of Principal Components

4.2.3 RMSE vs Variance Retained

Figure 9 shows the Root Mean Square Error (RMSE) as a function of the variance retained during PCA compression.

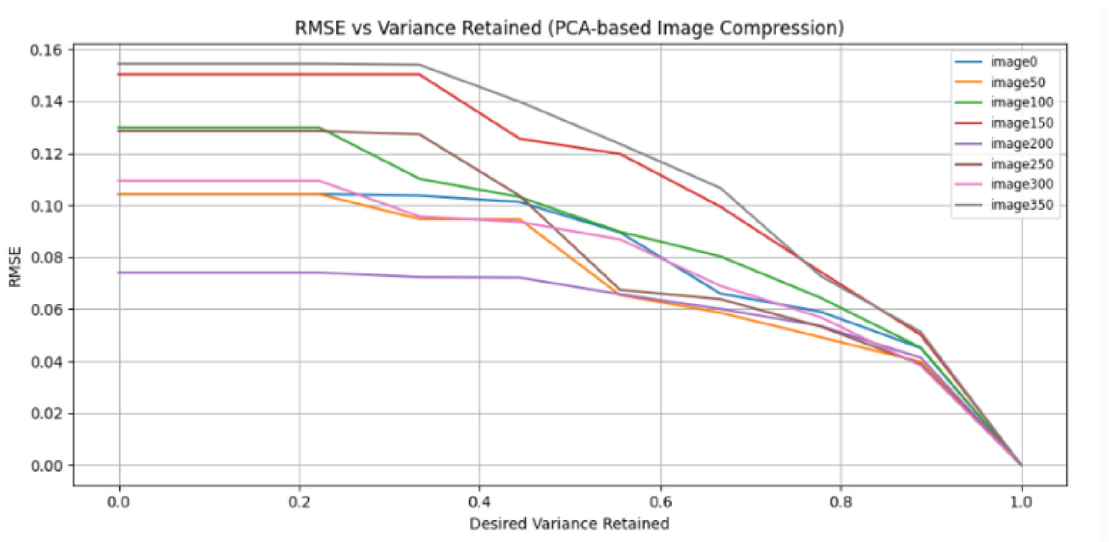

Figure 9: RMSE vs Variance Retained

5 Sparse Orthonormal Transforms (SOTs)

5.1 A Brief Introduction

Sparse Orthonormal Transforms (SOTs) are an advanced method for image compression and representation that aim to improve upon traditional transforms like the Discrete Cosine Transform (DCT) and Karhunen-Loeve Transform (KLT)! The basic idea behind SOTs is to minimize redundancy in image data by identifying and exploiting regularities locally. Unlike traditional approaches that rely on specific mathematical models (for example, something like a Fourier or wavelet based approximations), SOTs adopt a data-driven methodology, allowing them to handle diverse and non-stationary signal characteristics efficiently.

One of the most important aspects of SOTs is their adaptability. They are designed to automatically adjust to different types of signal regularity, making them well-suited for multimedia signals that often exhibit complex and non-Gaussian statistical behaviors.

5.1.1 Why SOTs in particular?

We are incorporating SOTs for several reasons:

- SOTs provide better approximation performance for a given number of coefficients, allowing for higher compression efficiency while preserving image quality.
- Unlike traditional methods, SOTs are model independent and are effective across a wide range of signal structures.
- They serve as a flexible extension to existing transforms like the DCT and can adapt to various data distributions without compromising performance.

5.2 What is Classification and Annealing?

One of the strengths of SOTs lies in their ability to efficiently represent a wide variety of signal structures using nonlinear approximation. Unlike linear approximation methods, which only extract the first few basis vectors of a transform, nonlinear methods can distribute evs across any subset of basis vectors, allowing for better handling of complex or irregular patterns. SOTs improve representation by classifying signals into distinct groups, each optimized with a specific orthonormal transform. By dividing signals into classes SOTs maintain orthonormal properties while capturing localized regularities effectively!

The annealing process iteratively optimizes them at decreasing levels of sparsity, controlled by the parameter λ. Starting with a high λ ensures broad changes in the transforms, stabilizing the larger structures in the data. As λ decreases, finer details are incorporated, leading to transforms that effectively capture high-energy clusters without overfitting due to noise.

5.3 Algorithm Description

The following describes the main algorithm for Sparse Orthonormal Transforms (SOTs), adapted from Sezer, Guleryuz, and Altunbasak (2015).

1. **Initialization:**
   (a) Partition the training set S into K sub-classes, S_k, where $k = 1, \dots, K$.
   (b) Set $H^k = H_0^k$, for $k = 1, \dots, K$.
2. **Transform Update:**
   (a) Set $\lambda_t = \lambda_{\max}$, $\forall k \in \{1, \dots, K\}$.
   (b) Set $G^k = H^k$.
   (c) Compute the optimal transform using S_k, $H_0 = G^k$, and λ_t, using Algorithm 1. Output G^k.
   (d) Perform the annealing step: $\lambda_t = \lambda_t - \delta\lambda$.
   (e) Repeat until cooled down: If $\lambda_t > \lambda$, return to step (b).
3. **Reclassification:**
   (a) Relabel data: For all $x \in S$, obtain $I(x)$ as:
   $$I(x) = \arg\min_k \left(\min_\alpha \|x - G^k\alpha\|_2^2 + \lambda\|\alpha\|_0 \right).$$
   (b) Update sub-classes: For all $k \in \{1, \dots, K\}$, $S_k = \{x \mid I(x) = k\}$.
4. **Overall Convergence Check:**
   (a) Compute the cost function:
   $$\hat{C}_N(G^1, \dots, G^K, \lambda) = \sum_{k=1}^{K} E\left[\min_\alpha \left(\|x - G^k\alpha\|_2^2 + \lambda\|\alpha\|_0\right) \mid x \in S_k\right].$$
   (b) Repeat until convergence criterion is met: If
   $$\Delta(\hat{C}_N(G^1, \dots, G^K, \lambda), \hat{C}_N(H^1, \dots, H^K, \lambda)) > \epsilon,$$
   set $H^k = G^k$, $k = 1, \dots, K$, and return to step 2(a).
   (c) Output G^k, $k = 1, \dots, K$.

5.4 Implementation

The implementation can be broken down as follows:

5.4.1 Preprocessing and Initialization

- The image is loaded, normalized, and divided into non-overlapping 8×8 blocks to ensure local compression, which captures fine-grained image structures.
- Initial transforms (H_k) are generated using KLT which calculates eigenvectors of the covariance matrix for each cluster of image blocks. These eigenvectors serve as the initial orthonormal transforms.
- Blocks are grouped into K clusters based on their gradient directions, estimated using horizontal and vertical gradients (G_x and G_y), and assigned to classes using a gradient-based heuristic.

5.4.2 Iterative Optimization with Annealing

- For each block, transform coefficients are computed using the assigned class's transform (H_k). Thresholding is applied to enforce sparsity, penalizing smaller coefficients based on the sparsity parameter λ.
- A cost function, combining reconstruction error and a sparsity penalty, is evaluated for each class. Blocks are reclassified to minimize this cost, allowing the algorithm to refine clustering iteratively.
- Each class's transform is updated using Singular Value Decomposition (SVD) on the covariance matrix of the blocks assigned to that class, ensuring the transforms remain orthonormal.

5.4.3 Annealing

- The sparsity parameter λ is gradually decreased, starting from λ_{max} and stepping down by $\delta\lambda$, in a process similar to simulated annealing. This encourages large changes initially and stabilizes the solution as λ reduces.
- The process repeats until the transforms converge, measured by the Frobenius norm of the difference between successive transforms.

5.4.4 Quantization and Encoding

- After optimization, transform coefficients are thresholded and quantized. Huffman encoding is applied to both the quantized coefficients and block labels to further reduce the storage size.
- The encoded data and transforms are saved, allowing the reconstruction of the compressed image.

5.5 Analysis of Results

5.5.1 Image Compression Results

The figure below demonstrates the results of applying the implemented compression algorithm to a sample image. On the left is the original image, while the right shows the reconstructed compressed image after processing with the Sparse Orthonormal Transforms (SOTs) algorithm.

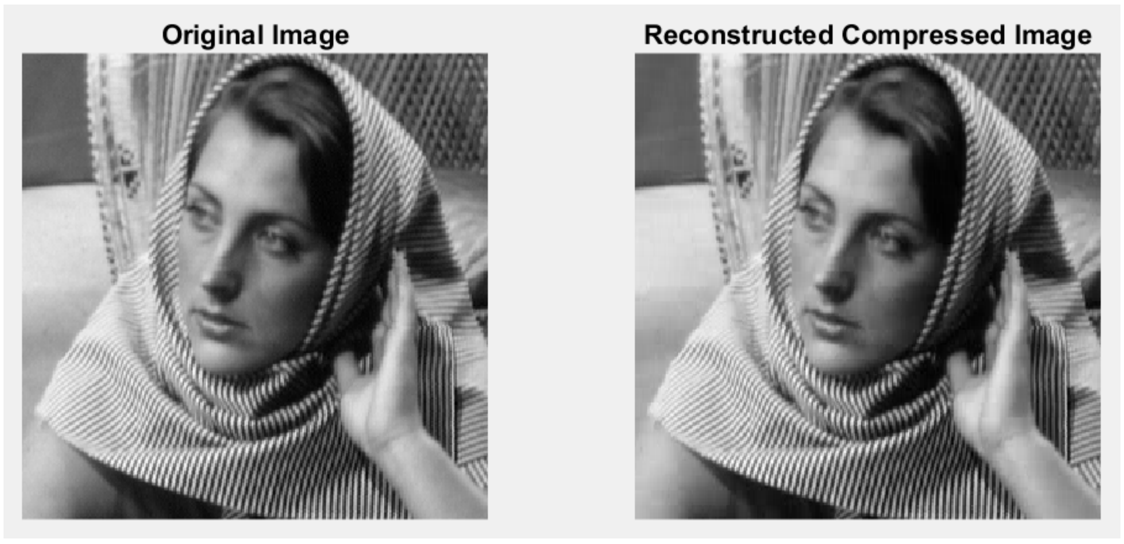

Figure 10: Comparison of Original and Reconstructed Compressed Images for Barbara PNG

- **Original Image Size:** 65,536 bytes
- **Compressed Image Size:** 20,924 bytes
- **Compression Ratio:** 3.13
- **Final RMSE:** 0.010011

The algorithm achieves a significant compression ratio of 3.13 while maintaining excellent visual quality indicated by the low RMSE value. The reconstructed image preserves the essential details of the original, with minimal perceptible loss in quality.

The plot on the next page showcases the relationship between Peak Signal-to-Noise Ratio (PSNR) and Rate (bits per pixel).

What Was Done? To generate this plot, the implemented Sparse Orthonormal Transforms (SOTs) algorithm was used to compress and reconstruct an image under varying conditions:

- **Sparsity Parameter (λ):** The sparsity of transform coefficients was controlled, simulating how aggressive or conservative the compression should be.
- **Quantization:** A constant quantization step size was applied to the transform coefficients, which directly affects the precision of the stored values.

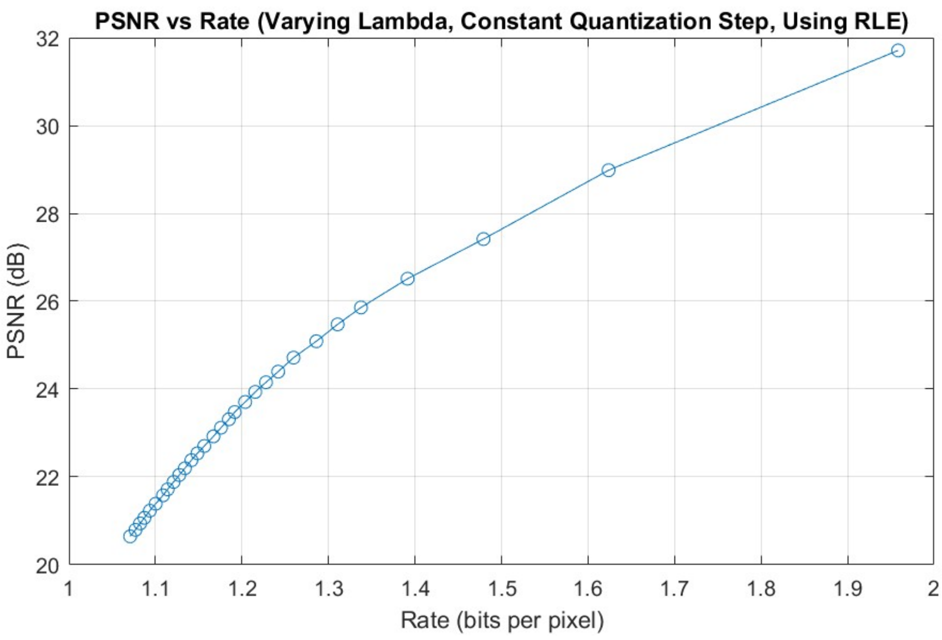

Figure 11: PSNR (db) vs Rate (bpp) for barbara256.jpg

- **Encoding:** Run-Length Encoding (RLE) was employed to compress the quantized coefficients further, reducing redundancy in the data.

For each setting, the compressed image was reconstructed, and its PSNR and compression rate were calculated:

- **PSNR (in dB):** A measure of the quality of the reconstructed image compared to the original. Higher PSNR indicates better reconstruction with less perceptual distortion.
- **Rate (bits per pixel):** The number of bits required to store each pixel of the compressed image. Lower rates indicate greater compression.

What Does the Graph Show?

- At lower rates (~ 1 bit/pixel), compression is more aggressive (PSNR $\sim$ 20–22 dB).
- As the rate increases (> 1.5 bits/pixel), the PSNR climbs steadily, reaching values around 30–32 dB.

Significance and Comparison This behavior closely mirrors trends observed in the original research paper! The graph validates that the implemented algorithm achieves similar efficiency and quality trade-offs as described in the Sparse Orthonormal Transforms framework. This visualization highlights the potential of the algorithm for applications requiring high quality compression with customizable trade-offs.

Let's look at the results for a different picture!

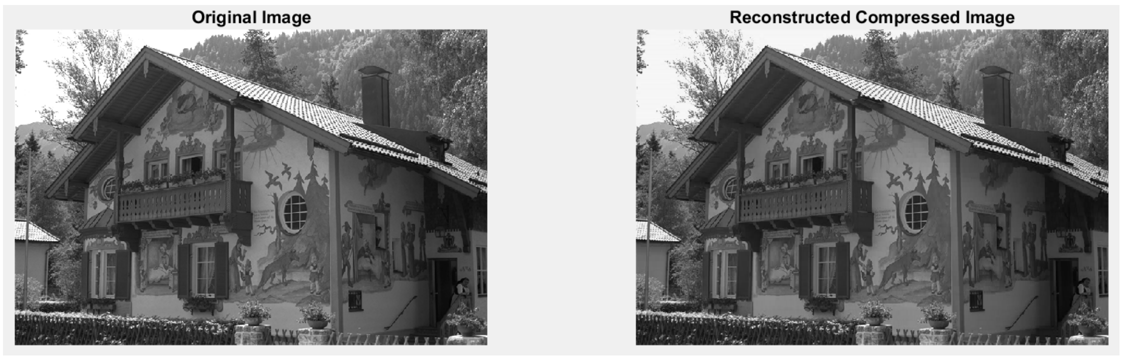

Figure 12: Comparison of Original and Reconstructed Compressed Images for kodak24.png

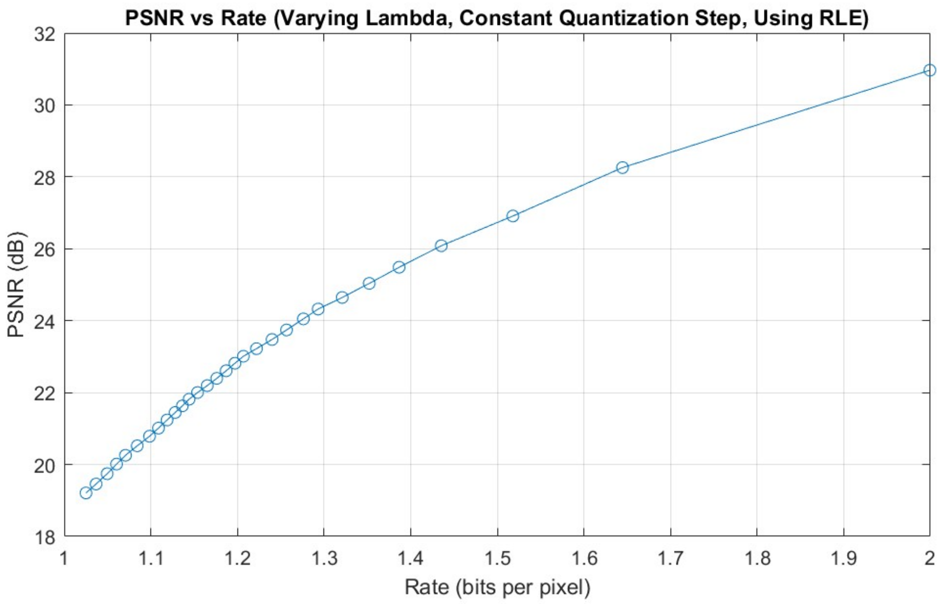

Figure 13: PSNR (db) vs Rate (bpp) for kodak24.jpg

For this image, which is larger in size compared to the previous example, the implemented compression algorithm achieves the following results:

- **Original Image Size:** 393,216 bytes
- **Compressed Image Size:** 129,777 bytes
- **Compression Ratio:** 3.03
- **Final RMSE:** 0.009562

Despite the larger size of the image, the algorithm maintains a high compression ratio of approximately 3 times while preserving excellent visual quality! The PSNR vs. Rate plot showcases a consistent pattern. Looking at the numbers and values, we can conclude that compression is model independent and consistent!

6 External References

O. G. Sezer, O. G. Guleryuz, and Y. Altunbasak, *"Approximation and Compression With Sparse Orthonormal Transforms,"* in *IEEE Transactions on Image Processing*, vol. 24, no. 8, pp. 2328–2343, Aug. 2015. doi: 10.1109/TIP.2015.2414879.