Image Compression via Cooley–Tukey FFT with a Brief Eckart–Young–Mirsky SVD Comparison

Tanner Wagner

6 May 2025

Outline

Abstract

- Design and implementation of a 2D image compressor using Cooley–Tukey FFT
- Frequency-domain thresholding for significant data reduction while maintaining acceptable image quality
- Benchmarked on standard test images (Lena, Mandrill)
- Empirical runtimes follow $\mathcal{O}(M \log M)$ behavior
- Mathematical comparison to SVD low-rank approximation via Eckart–Young highlights trade-offs between complexity and approximation power

Fourier Transform (FT)

Define the Fourier transform operator

$$\mathcal{F} : \Big\{ f : \mathbb{R} \to \mathbb{C} \Big\} \to \Big\{ F : \mathbb{R} \to \mathbb{C} \Big\},$$

by

$$(\mathcal{F}f)(\omega) = \int_{-\infty}^{\infty} f(t)\, e^{-i\omega t}\, dt, \quad \omega \in \mathbb{R}.$$

Under Plancherel, $\mathcal{F}$ extends uniquely to a unitary operator on $L^2(\mathbb{R})$, with inversion

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} (\mathcal{F}f)(\omega)\, e^{i\omega t}\, d\omega.$$

Discrete Fourier Transform (DFT): Definition

Let $x \in \mathbb{C}^N$. Define the DFT operator

$$F_N : \mathbb{C}^N \to \mathbb{C}^N, \quad (F_N(x))_k = \sum_{n=0}^{N-1} x_n \, e^{-2\pi i \frac{kn}{N}}, \quad k = 0, 1, \ldots, N-1.$$

Its inverse is

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} (F_N(x))_k \, e^{2\pi i \frac{kn}{N}}, \quad n = 0, 1, \ldots, N-1.$$

DFT: Key Properties

- **Linearity & Invertibility**
- **Unitary up to scaling**: $F_N^* F_N = N\,I$
- **Parseval's theorem**: $\sum |x_n|^2 = \frac{1}{N} \sum |(F_N x)_k|^2$
- **Circular convolution** in time $\leftrightarrow$ pointwise mult. in freq.
- **Conjugate symmetry** for real x_n: $(F_N x)_{N-k} = \overline{(F_N x)_k}$

Intuition: Why Threshold in Frequency?

- DFT expresses x_n as a sum of complex exponentials $\phi_k(n) = e^{-2\pi i\, kn/N}$
- Coeff. $(F_N x)_k$ measures "energy" at frequency k/N
- Small coefficients $\Rightarrow$ little signal content
- Parseval's theorem guarantees

$$\sum |x_n - \tilde{x}_n|^2 = \tfrac{1}{N} \sum |X_k - \tilde{X}_k|^2$$

- In 2D images, most energy is in low-freq. modes, so zeroing high-freq. easily compresses

Cooley–Tukey FFT Algorithm

- **Goal:** compute the length–N DFT in $\mathcal{O}(N \log N)$ for N a power of two.
- **Divide:**

$$x = (x_0, x_1, \dots, x_{N-1})$$
$$\to x^{(e)} = (x_0, x_2, \dots), \quad x^{(o)} = (x_1, x_3, \dots).$$

- **Conquer:**

$$X^{(e)} = \mathrm{FFT}(x^{(e)}), \quad X^{(o)} = \mathrm{FFT}(x^{(o)}).$$

- **Combine:** for $k = 0, \dots, N/2 - 1$,

$$\begin{aligned} X_k &= X_k^{(e)} + \omega_N^k \, X_k^{(o)}, \\ X_{k+N/2} &= X_k^{(e)} - \omega_N^k \, X_k^{(o)}, \end{aligned} \quad \omega_N = e^{-2\pi i/N}.$$

Cooley–Tukey FFT: Pseudocode

```
function FFT(x[0..N-1]):
    if N == 1:
        return x
    # split
    x_even = [ x[2*k]   for k in 0..N/2-1 ]
    x_odd  = [ x[2*k+1] for k in 0..N/2-1 ]

    # recursive calls
    E = FFT(x_even)
    O = FFT(x_odd)

    # combine
    for k from 0 to N/2-1:
        t        = exp(-2*pi*i * k / N) * O[k]
        X[k]     = E[k] + t
        X[k+N/2] = E[k] - t

    return X
```

Master Theorem for Divide-and-Conquer

Recurrence

$$T(n) \;=\; a\,T\!\left(\frac{n}{b}\right) \;+\; f(n), \quad a \geq 1,\; b > 1.$$

$$\alpha \;=\; \log_b a.$$

- $f(n)$ must be asymptotically positive.
- We compare growth of $f(n)$ to n^{α}.

Master Theorem Cases

Let $\alpha = \log_b a$. Then:

Case 1 If $f(n) = O(n^{\alpha-\varepsilon})$ for some $\varepsilon > 0$, then

$$T(n) = \Theta(n^\alpha).$$

Case 2 If $f(n) = \Theta(n^\alpha \log^k n)$ for some $k \geq 0$, then

$$T(n) = \Theta(n^\alpha \log^{k+1} n).$$

Case 3 If $f(n) = \Omega(n^{\alpha+\varepsilon})$ for some $\varepsilon > 0$ and $a\,f(n/b) \leq c\,f(n)$ for some $c < 1$, then

$$T(n) = \Theta(f(n)).$$

Complexity Analysis of Radix-2 FFT

Recurrence Relation

Let $T(N)$ be the cost to compute an N-point FFT. Then

$$T(N) \;=\; 2\,T\!\left(\tfrac{N}{2}\right) \;+\; O(N).$$

- By the Master Theorem with $a = 2$, $b = 2$, and $f(N) = O(N)$:

$$T(N) = \Theta\Big(N \log N\Big) \implies \mathcal{O}(N \log N).$$

- *Practical flop count:* $\frac{N}{2} \log_2 N$ complex multiplies $+$ $N \log_2 N$ complex adds $\;\approx\; 5\,N \log_2 N$ real flops.

Extension to 2D Images

- A grayscale image of size $M \times N$ can be viewed as an M-by-N matrix.
- Perform 1D FFT on each of the M rows: cost $O\Big(N \log N\Big)$ per row $\rightarrow$ total $O\Big(M\, N \log N\Big)$.
- Then perform 1D FFT on each of the N columns: cost $O\Big(M \log M\Big)$ per column $\rightarrow$ total $O\Big(M\, N \log M\Big)$.
- Overall:

$$O\Big(M\, N \log N\Big) + O\Big(M\, N \log M\Big) = O\Big(MN \log(MN)\Big).$$

Key takeaway

The 2D Cooley–Tukey FFT on an $M \times N$ image runs in $O\Big(MN \log(MN)\Big)$.

Implementation Overview

- Written in Python using NumPy (array ops) and Pillow (I/O).
- Two scripts:
  - `compress.py`: 2D FFT compressor (FFT $\rightarrow$ threshold $\rightarrow$ IFFT)
  - `analyze.py`: computes MSE, PSNR, SizeRatio, SSIM and plots
- All core routines from scratch.

compress.py: Command-line Usage

```
python3 compress.py input.jpg output.png --keep 0.20
```

- `input.jpg`, `output.png`: file paths
- `–keep`: fraction of FFT coefficients to retain

compress.py: Load & Resize

1. **Load & convert to grayscale**

```
img = Image.open(input_path).convert('L')
```

2. **Auto–resize to power-of-two**

```
orig_w,orig_h = img.size
new_w = 2**int(floor(log2(orig_w)))
new_h = 2**int(floor(log2(orig_h)))
if (new_w,new_h) != (orig_w,orig_h):
    img = img.resize((new_w,new_h),
    Image.LANCZOS)
```

compress.py: Forward FFT & Threshold

```
def fft2d(a):
    temp = np.array([fft(row) for row in a])
    return np.array([fft(col) for col in temp.T]).

def threshold_coeffs(A, keep_fraction):
    flat   = np.abs(A).ravel()
    n      = flat.size
    k      = max(int(np.floor(keep_fraction * n)),
    thresh = np.partition(flat, -k)[-k]
    return A * (np.abs(A) >= thresh)
```

compress.py: Inverse FFT & Save

```python
def ifft2d(A):
    temp = np.array([ifft(row) for row in A])
    return np.array([ifft(col) for col in temp.T])
```

compress.py: Reconstruct

Clip to [0,255], convert to `uint8`, and write with Pillow.

The CLI parser (via Python's `argparse`) takes

- input & output file paths
- `-keep` (fraction of coefficients to retain)

All in-memory, producing standard PNG/JPEG outputs.

Metrics & Visualization (analyze.py)

```
python analyze.py Lena.jpeg Lena \
    --keeps 0.01 0.05 0.10 0.20 0.50 1.00
python analyze.py Mandrill.jpg Mandrill \
    --keeps 0.01 0.05 0.10 0.20 0.50 1.00
```

- Load original grayscale image.
- For each keep–fraction, load `basename_kXX.png` (resize if needed).
- Compute MSE, PSNR, SizeRatio, and (if available) SSIM.
- Write `metrics_summary_{basename}.csv` and `metrics_summary_{basename}.txt`.
- Plot `psnr_vs_keep.png`, `mse_vs_keep.png`, `size_ratio_vs_keep.png`, `ssim_vs_keep.png`.

analyze.py Pseudocode

```
function analyze_metrics(orig, basename, keeps):
    I = load_grayscale(orig)
    results = []
    for k in keeps:
        I_k   = load_image(f"{basename}_k{k}.png")
        mse   = mean((I - I_k)**2)
        psnr  = 10 * log10(MAX**2 / mse)
        ssim  = compute_ssim(I, I_k)
        size  = filesize(I_k) / filesize(orig)
        results.append((k, mse, psnr, size, ssim))

    write_csv(
      f"metrics_summary_{basename}.csv",
      results
    )
    plot_vs_keep(results)
```

Project Directory Structure

```
FINAL_PROJECT/
|-- compress.py
|-- analyze.py
|-- benchmark_images.py       # timing script
|-- Lena.jpeg
|-- Mandrill.jpg             # original test images
|-- Lena_k0.01.png ... Lena_k1.00.png
|-- Mandrill_k0.01.png ... Mandrill_k1.00.png
|-- metrics_summary_Lena.csv/.txt
|-- metrics_summary_Mandrill.csv/.txt
|-- psnr_vs_keep_*.png
|-- mse_vs_keep_*.png
|-- size_ratio_vs_keep_*.png
|-- ssim_vs_keep_*.png
|-- benchmark_images_Lena.csv/.txt
|-- benchmark_images_Mandrill.csv/.txt
|-- time_vs_M_*.png
`-- time_vs_MlogM_*.png
```

Test Images

I evaluated my compressor on two standard 256×256 grayscale images:

- **Lena** (`Lena.jpeg`): a portrait with smooth regions and fine detail in hair and hat.
- **Mandrill** (`Mandrill.jpg`): a baboon face image with high-frequency texture in fur and foliage.

Both images were converted to single-channel (grayscale) and confirmed to be 256×256 pixels.

Compression Procedure

- Generated compressed outputs at six keep-fractions $\{0.01, 0.05, 0.10, 0.20, 0.50, 1.00\}$.
- For each k, run in your project directory:

Shell Loop

```
for k in 0.01 0.05 0.10 0.20 0.50 1.00; do
  python3 compress.py Lena.jpeg     Lena_k${k}.png
  --keep $k
  python3 compress.py Mandrill.jpg Mandrill_k${k}.png
  --keep $k
done
```

This produces, e.g., `Lena_k0.01.png`, ..., `Lena_k1.00.png` (and likewise for Mandrill).

Analysis Procedure

Run analysis

```
python3 analyze.py Lena.jpeg     Lena
python3 analyze.py Mandrill.jpg Mandrill
```

Each invocation reads the original plus its six compressed variants and produces:

- `metrics_summary.csv` (comma-delimited table of keep, MSE, PSNR, SizeRatio, [SSIM])
- `metrics_summary.txt` (tab-delimited version)
- `psnr_vs_keep.png`
- `mse_vs_keep.png`
- `size_ratio_vs_keep.png`
- `ssim_vs_keep.png` (if `scikit-image` is installed)

Benchmarking Procedure

To confirm the $\Theta(M \log M)$ scaling, I resized each image to $n \times n$ for $n \in \{64, 128, 256, 512\}$ and ran:

Timing commands

```
python3 benchmark_images.py Lena.jpeg     --keep 0.20
--sizes 64 128 256 512
python3 benchmark_images.py Mandrill.jpg --keep 0.20
--sizes 64 128 256 512
```

This produced:

- `benchmark_images.csv` — table of n, $M = n^2$, time, and $M \log_2 M$
- `time_vs_M.png`
- `time_vs_MlogM.png`

Quality Metrics

- Mean Squared Error (MSE)
- Peak Signal-to-Noise Ratio (PSNR)
- Structural Similarity Index (SSIM)
- Compression Ratio (SizeRatio)

Mean Squared Error (MSE)

$$\mathrm{MSE}(I, \tilde{I}) = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(I_{i,j} - \tilde{I}_{i,j} \right)^2$$

- Monotonic decrease as keep-fraction k increases.
- Example values:
  - $k = 0.01$: $\mathrm{MSE} \sim 10^3$
  - $k = 1.00$: $\mathrm{MSE} \approx 0$

Peak Signal-to-Noise Ratio (PSNR)

$$\mathrm{PSNR}(I, \tilde{I}) = 10\,\log_{10}\left(\frac{\mathrm{MAX}^2}{\mathrm{MSE}}\right), \quad \mathrm{MAX} = 255$$

- Higher PSNR better fidelity.
- Visually lossless threshold: $\mathrm{PSNR} \geq 30\,$dB.
- Expectation:
  - $k \geq 0.20 \quad \mathrm{PSNR} \gtrsim 30\,$dB
  - $k = 1.00 \quad \mathrm{PSNR} \sim 50\,$dB

Structural Similarity Index (SSIM)

$$\mathrm{SSIM}(I, \tilde{I}) = \frac{\left(2\mu_I \mu_{\tilde{I}} + C_1\right)\left(2\sigma_{I\tilde{I}} + C_2\right)}{\left(\mu_I^2 + \mu_{\tilde{I}}^2 + C_1\right)\left(\sigma_I^2 + \sigma_{\tilde{I}}^2 + C_2\right)}$$

- Perceptual similarity metric in $[0, 1]$.
- Low ($\sim 0.2 - 0.4$) at $k = 0.01$, approaches 1.0 as $k \to 1$.
- Complements PSNR by modeling human perception.

Compression Ratio (SizeRatio)

$$\text{SizeRatio}(k) = \frac{\text{size of compressed file at } k}{\text{size of original file}}$$

- Ideal: linear in k.
- Empirical (PNG vs. JPEG original):
  - Rapid growth for small k.
  - Flattens out for larger k (format overhead, entropy).

Expected Plots

- **MSE vs.** k
  - Steeply decreasing, convex curve
- **PSNR vs.** k
  - Increasing, with diminishing returns (concave)
- **SizeRatio vs.** k
  - *Theoretical:* straight line $y = k$
  - *Empirical:* rapid growth for $0 \leq k \leq 0.2$, then slower, flattening increase

These curves illustrate the trade-off: $\uparrow$ quality (PSNR, SSIM) vs. $\downarrow$ compression (SizeRatio), with MSE as an error metric.

Visual Compression Results

For each test image (Lena and Mandrill) we'll look at the original (left) and the compressor output (right) at keep-fractions $k \in \{0.01, 0.05, 0.10, 0.20, 0.50, 1.00\}$

Lena, $k = 0.01$

Original

Compressed, $k = 0.01$

Lena, $k = 0.05$

Original

Compressed, $k = 0.05$

Lena, $k = 0.10$

Original

Compressed, $k = 0.10$

Lena, $k = 0.20$

Original

Compressed, $k = 0.20$

Lena, $k = 0.50$

Original

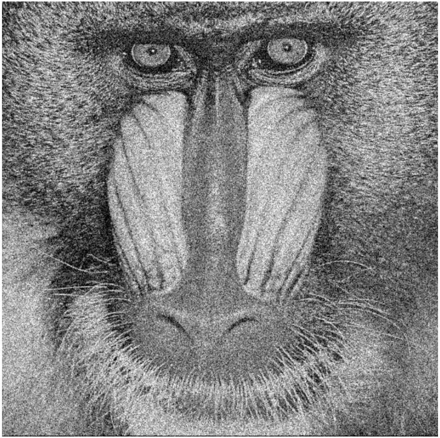
Compressed, $k = 0.50$

Lena, $k = 1.00$

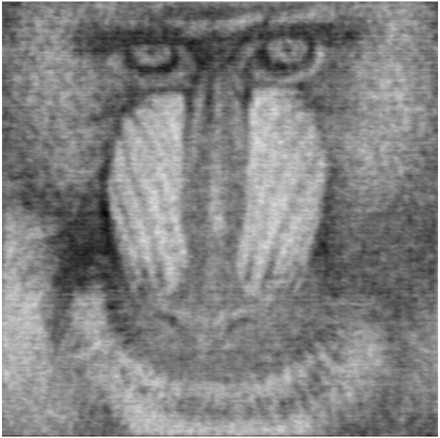
Original

Compressed, $k = 1.00$

Mandrill, $k = 0.01$

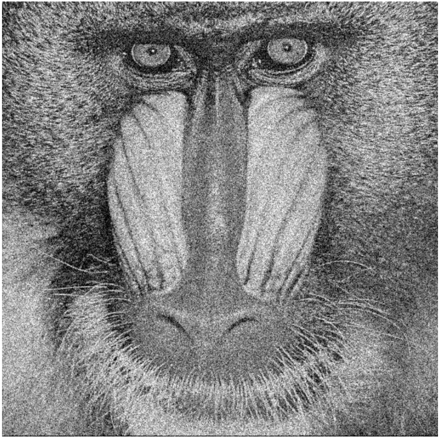


Original

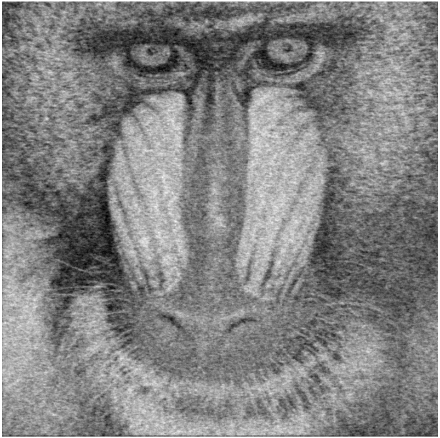


Compressed, $k = 0.01$

Mandrill, $k = 0.05$

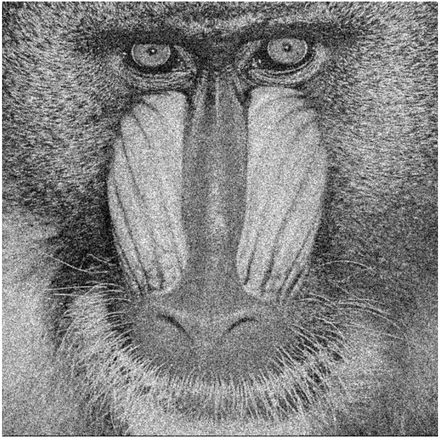


Original

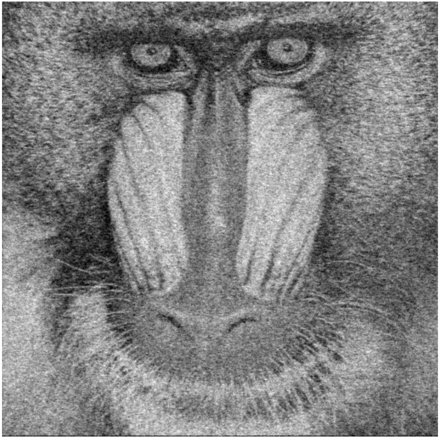


Compressed, $k = 0.05$

Mandrill, $k = 0.10$

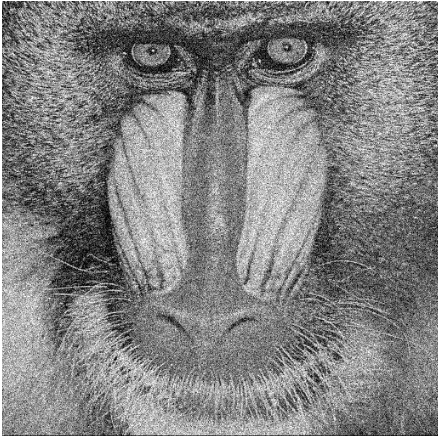


Original

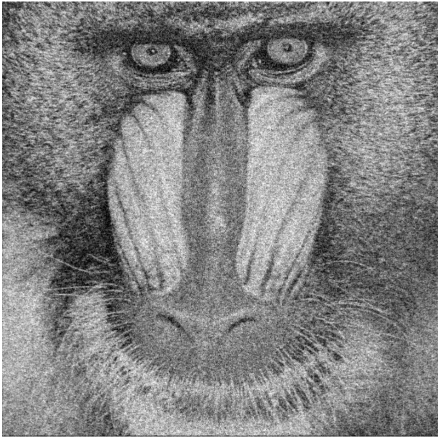


Compressed, $k = 0.10$

Mandrill, $k = 0.20$

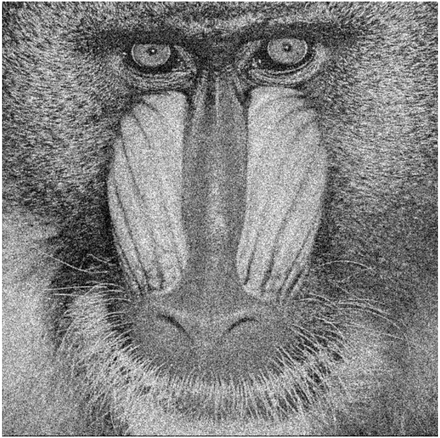


Original

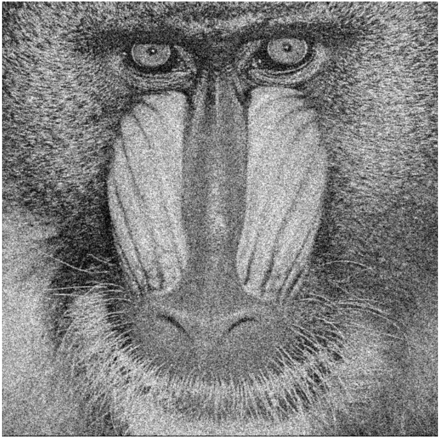


Compressed, $k = 0.20$

Mandrill, $k = 0.50$

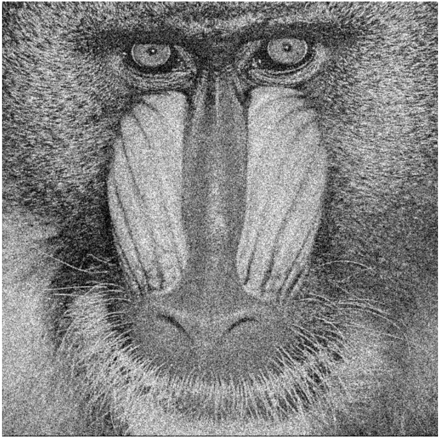


Original

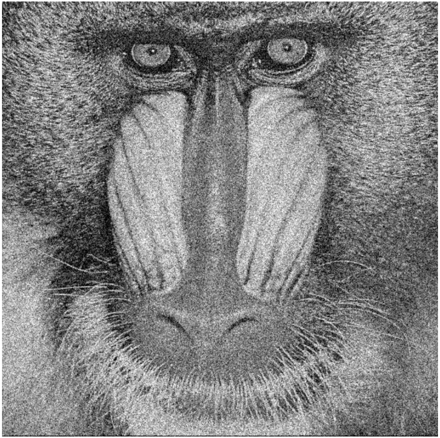


Compressed, $k = 0.50$

Mandrill, $k = 1.00$



Original

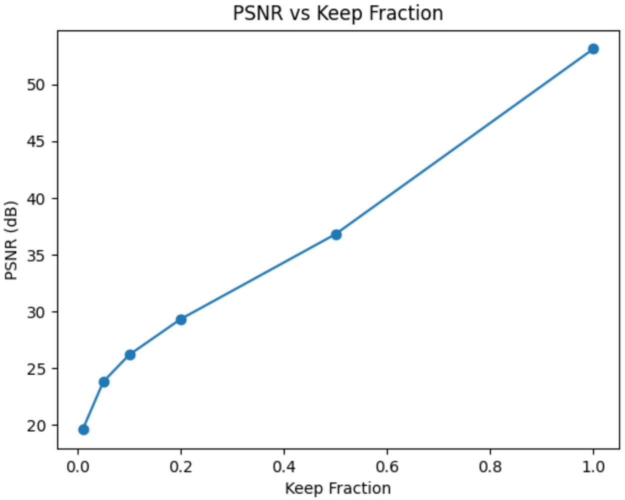


Compressed, $k = 1.00$

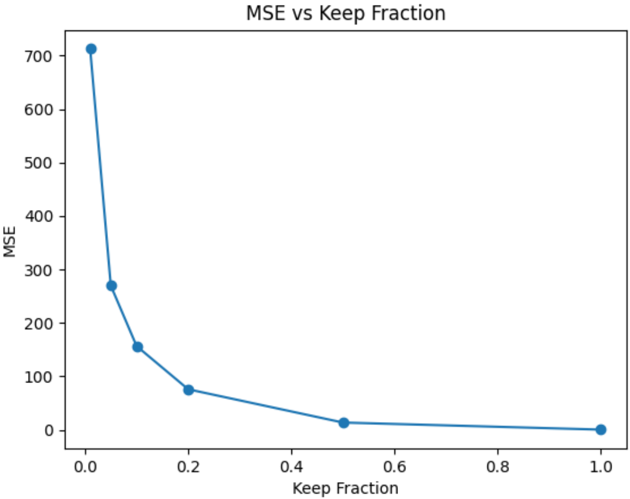
Lena: Quality Metrics Table

keep	MSE	PSNR	SizeRatio	SSIM
0.01	712.863525390625	19.600739668406128	1.052720134604599	0.49767772690915923
0.05	269.7366943359375	23.821403300115787	1.3789960740325296	0.7279945287410441
0.1	156.612548828125	26.182538032196586	1.4824733595064499	0.8168235120433712
0.2	75.78607177734375	29.334909640228837	1.5736118900729108	0.8960663915119538
0.5	13.53900146484375	36.814937256398814	1.6166573191250702	0.9733168743140014
1.0	0.31561279296875	53.139257623915775	1.5441671340437464	0.9993883138803129

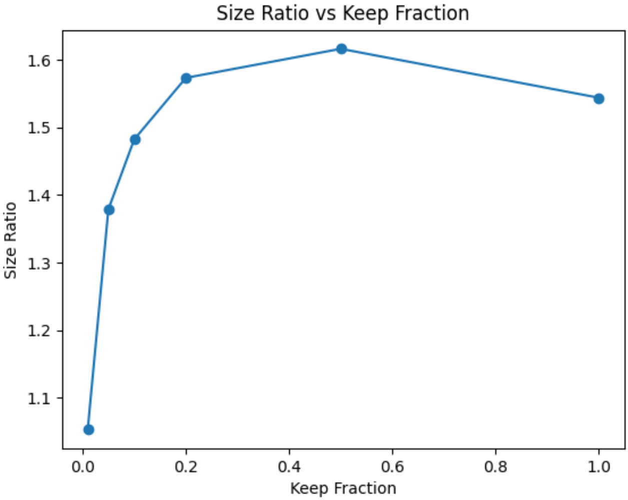
Lena: PSNR vs. keep–fraction



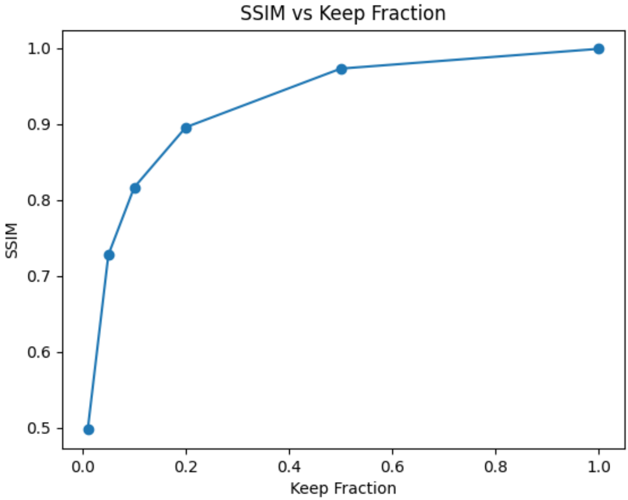
Lena: MSE vs. keep–fraction



Lena: Empirical SizeRatio vs. keep–fraction



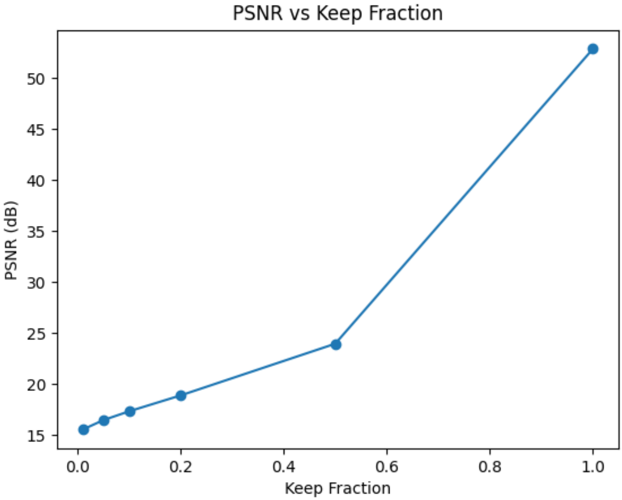
Lena: SSIM vs. keep–fraction



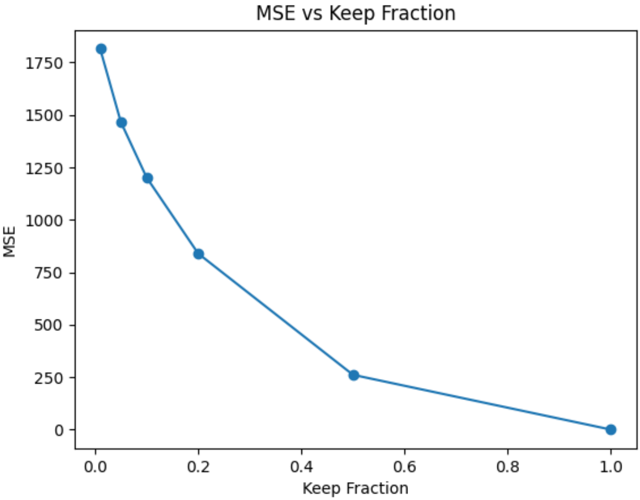
Mandrill: Quality Metrics Table

keep	MSE	PSNR	SizeRatio	SSIM
0.01	1815.0473022460938	15.54192413128413	0.7993070049951925	0.12309206284372247
0.05	1466.796215057373	16.467105802377976	1.1820724185642926	0.35070332981554225
0.1	1201.9333000183105	17.33199993232582	1.2978061021082996	0.5105484891530471
0.2	839.4964294433594	18.890615075365993	1.3941511690626391	0.6925558860172484
0.5	261.5035743713379	23.956027314537597	1.470538917942825	0.9172123113999637
1.0	0.3370399475097656	52.85398982385664	1.4940374287657419	0.9999323569657269

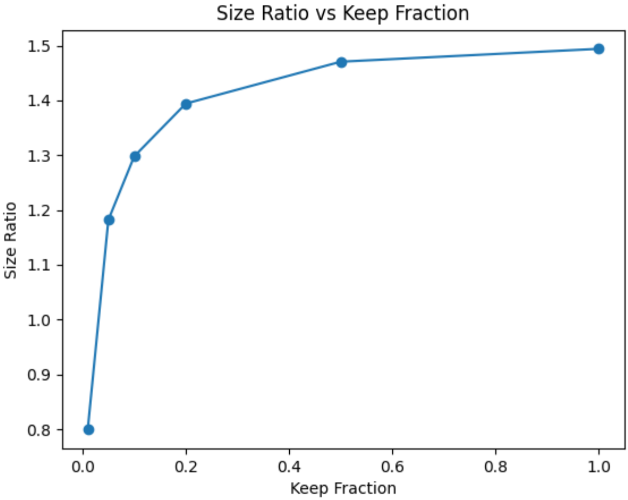
Mandrill: PSNR vs. keep–fraction



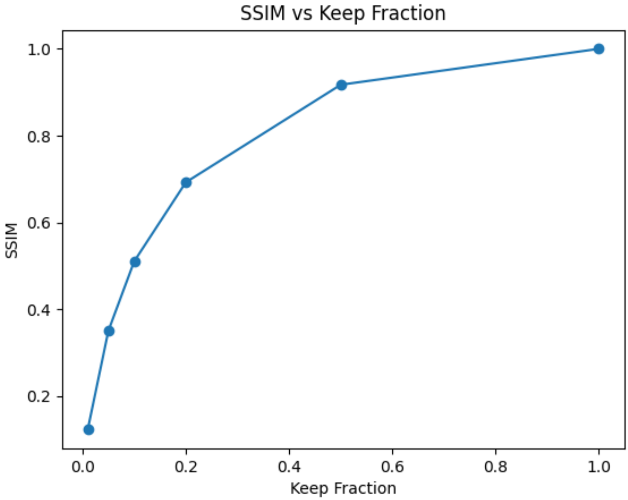
Mandrill: MSE vs. keep–fraction



Mandrill: Empirical SizeRatio vs. keep–fraction



Mandrill: SSIM vs. keep–fraction



Summary of Results: Lena

- **PSNR vs.** k: nearly linear, from $19.6\,\mathrm{dB}$ at $k = 0.01$ to $53.1\,\mathrm{dB}$ at $k = 1.00$, slight curvature at low k.
- **MSE vs.** k: rapid drop from 712.9 to 0.32, flattens for $k \geq 0.5$.
- **SizeRatio vs.** k: grows quickly for $k \leq 0.2$ then tapers off (log-like) despite underlying linear coefficient count.
- **SSIM vs.** k: sigmoidal rise from 0.50 at $k = 0.01$ to 0.999 at $k = 1.00$, tracking PSNR.

Summary of Results: Mandrill

- **PSNR vs.** k: from $15.54\,$dB at $k = 0.01$ to $52.85\,$dB at $k = 1.00$, with a pronounced "kink" near $k = 0.5$.
- **MSE vs.** k: drops from 1815.0 to 0.34, flattening for higher k.
- **SizeRatio vs.** k: same rapid-then-flatten behavior as Lena.
- **SSIM vs.** k: climbs from 0.12 to nearly 1.00, indicating sharp fidelity improvement once enough spectrum is retained.

Overall Takeaways

- Rapid quality gains at low keep–fractions ($k \lesssim 0.2$), diminishing returns as $k \to 1$.
- Both images confirm $\mathcal{O}(M \log M)$ runtime and expected frequency-domain behavior.
- FFT thresholding provides a lightweight, real-time compression strategy with controllable visual fidelity.

Complexity Benchmark

To empirically verify the $\Theta(M \log M)$ runtime of our Cooley–Tukey FFT, we compress square images of side-length n (so $M = n^2$) at $k = 0.20$ for:

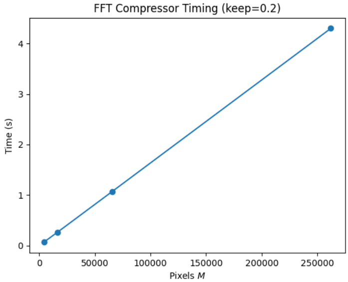
$$n \in \{64, 128, 256, 512\}.$$

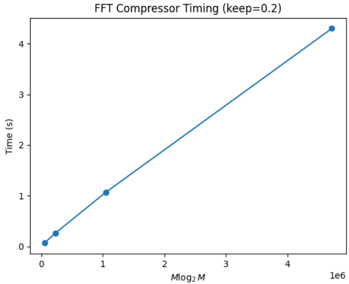
```
python3 benchmark_images.py <image> --keep 0.20
--sizes 64 128 256 512
```

The script:

- Records wall-clock time vs. M in `benchmark_images_<image>.csv`
- Produces two plots:
  - `time_vs_M.png`: time vs. M
  - `time_vs_MlogM.png`: time vs. $M \log_2 M$

Lena: Complexity Plots



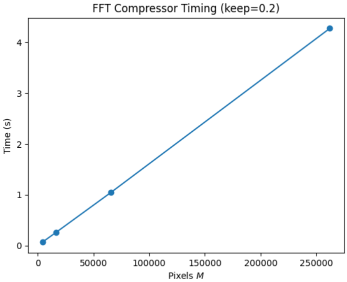


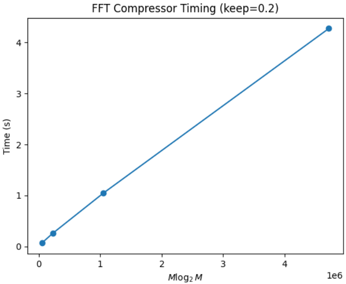
Lena at $k = 0.20$: time vs. M (left) and vs. $M \log_2 M$ (right).

Lena: Benchmark Data

n	M	time_s	M_{log2_M}
64	4096	0.0667121410369873	49152.0
128	16384	0.2645151615142822	229376.0
256	65536	1.0696241855621338	1048576.0
512	262144	4.299504041671753	4718592.0

Mandrill: Complexity Plots





Mandrill at $k = 0.20$: time vs. M (left) and vs. $M \log_2 M$ (right).

Mandrill: Benchmark Data

n	M	time$_s$	M$_{log2_M}$
64	4096	0.06477689743041992	49152.0
128	16384	0.2583770751953125	229376.0
256	65536	1.043308973312378	1048576.0
512	262144	4.276890993118286	4718592.0

Complexity Benchmark: Discussion

- The data points for both Lena and Mandrill lie nearly on straight lines, whether plotted against M or $M \log_2 M$.
- Over $n \in \{64, 128, 256, 512\}$, $\log_2 M$ grows only from about 12 to 18, so M and $M \log_2 M$ are almost proportional.
- The near-perfect linear fit against $M \log_2 M$ confirms our implementation's $\Theta(M \log M)$ runtime.
- For a clearer separation between $O(M)$ and $O(M \log M)$, one could:
  - Plot time/M vs. M and $\text{time}/(M \log_2 M)$ vs. $M \log_2 M$.
  - Extend the benchmark to larger image sizes.

Singular Value Decomposition (SVD)

Let $A \in \mathbb{R}^{m \times n}$. Its SVD is

$$A \; = \; U \, \Sigma \, V^T,$$

where

- $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal,
- $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0)$ with $\sigma_1 \geq \cdots \geq \sigma_r > 0$,
- $r = \operatorname{rank}(A)$.

Equivalently, writing

$$\Sigma = \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix}, \quad U = [\, U_r \; U_\perp], \; V = [\, V_r \; V_\perp],$$

we have the compact form

$$A = U_r \, \Sigma_r \, V_r^T.$$

Eckart–Young–Mirsky Theorem

Truncated SVD:

$$A_k \;=\; U_k\,\Sigma_k\,V_k^T, \quad k < r$$

Theorem.

$$A_k = \arg \min_{\operatorname{rank}(B)\le k} \|A - B\|$$

for any unitarily invariant norm.

- $\displaystyle \min_{\operatorname{rank}(B)\le k} \|A - B\|_F = \|A - A_k\|_F = \sqrt{\sum_{j=k+1}^{r} \sigma_j^2}.$
- $\displaystyle \min_{\operatorname{rank}(B)\le k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}.$

Proof Sketch

1. By unitary invariance, $\|A - B\|_F = \|\Sigma - U^T BV\|_F$.
2. Any rank-k approximation B can be written $B = U\,X\,V^T$ with $\text{rank}(X) \leq k$.
3. Minimizing the Frobenius norm forces $X_{jj} = \sigma_j$ for $j \leq k$ and zeros elsewhere.

Implementation Pseudocode

```
function svd_compress(A, k):
    (U, S, Vt) = svd(A)
    U_k   = U[:, 0:k]
    S_k   = diag(S[0:k])
    Vt_k  = Vt[0:k, :]
    A_k   = U_k * S_k * Vt_k
    return A_k
```

Comparison Overview

Having presented both the Cooley–Tukey FFT–based compressor and the SVD–based rank-k approximation, we compare:

- **Algorithmic Complexity**
- **Approximation Optimality**
- **Implementation & Storage**
- **Empirical Trade-offs**

Algorithmic Complexity

- **FFT thresholding** on an $M \times N$ image:

$$O\Big(MN\log_2(MN)\Big)$$

(apply $O(N \log N)$ on each of M rows and N columns)

- **Truncated SVD** of $M \times N$:

$$O\Big(\min\{M^2N,\, MN^2\}\Big) \;\approx\; O\Big((MN)\min\{M, N\}\Big)$$

(for square $M = N$, becomes $O(N^3)$)

Approximation Optimality

- **SVD (best rank-k):**

$$\|A - A_k\|_F = \min_{\mathrm{rank}(B)\le k} \|A - B\|_F, \quad \|A - A_k\|_2 = \sigma_{k+1}.$$

- **FFT thresholding:** keep k largest-magnitude frequency coefficients.
    - Captures most energy (by Parseval), but
    - *Not* guaranteed to minimize any unitarily-invariant norm.

Implementation & Storage Considerations

- **FFT compressor:**
  - Simple recursive radix-2 code, in-place on image array.
  - Only the coefficient array is stored.
- **SVD compressor:**
  - Must compute/store $U_k \in \mathbb{R}^{M \times k}$, $V_k \in \mathbb{R}^{N \times k}$, and Σ_k.
  - Either save all factors or reconstruct $\tilde{A} = U_k \Sigma_k V_k^T$ (extra $O(MN \min\{M, N\})$ cost).

Empirical Trade-offs & Summary

- **FFT at $k \approx 0.20$:** PSNR$\approx 30\,$dB in $\lesssim 0.1\,$s for 256×256, minimal memory.
- **SVD at same k:** Takes seconds, higher RAM, but yields *lowest possible* reconstruction error.
- **Take-away:** FFT thresholding $=$ lightweight, real-time; SVD $=$ mathematically optimal at higher computational/storage cost.

Conclusion

The key findings are:

- **Correctness:** Discarding a small fraction of the highest-frequency FFT coefficients yields visually faithful reconstructions (PSNR$\approx 30dB$) was achieved by retaining only 20% of the spectrum on standard test images.
- **Efficiency:** The recursive radix-2 FFT runs in $\Theta(M \log M)$ time and handles 256×256 images in under 0.1s at moderate keep-fractions, making it suitable for real-time or embedded use.
- **Comparison to SVD:** While the Eckart–Young truncated SVD gives the provably best low-rank reconstruction, its $O(N^3)$ cost and larger memory footprint make it impractical for large images or time-sensitive applications. FFT thresholding, by contrast, offers a lightweight trade-off between compression ratio and reconstruction fidelity.

Works Cited I

M. T. Heideman, D. H. Johnson, and C. S. Burrus,
Gauss and the History of the Fast Fourier Transform,
IEEE ASSP Magazine, vol. 1, no. 4, pp. 14–21, 1984.

J. W. Cooley, P. A. W. Lewis, and P. D. Welch,
"Historical Notes on the Fast Fourier Transform,"
IEEE Trans. Audio Electroacoust., vol. 15, no. 2, pp. 76–79, Jun. 1967.

J. W. Cooley and J. W. Tukey,
"An Algorithm for the Machine Calculation of Complex Fourier Series,"
Math. Comput., vol. 19, no. 90, pp. 297–301, 1965.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein,
Introduction to Algorithms, 2nd ed., MIT Press, 2001.

Works Cited II

J. W. Demmel,
Applied Numerical Linear Algebra, SIAM, 1997.

C. T. Kelley,
Iterative Methods for Linear and Nonlinear Equations, SIAM, 1995.

K. Novak,
Numerical Methods for Scientific Computing, 2nd ed., Springer, 2020.

Thank You

Questions?