Implementasi Algoritma Block-Based Singular Value Decomposition (SVD) pada Kompresi Citra Digital untuk Mempertahankan Detail Lokal

Indraswara Galih Jayanegara - 13522119[1,2,3]

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

[1]13522119@std.stei.itb.ac.id, [2]hi@indraswara.me [3]indrswara@gmail.com

***Abstract*—Pertumbuhan pesat penggunaan citra digital dalam berbagai aplikasi teknologi informasi menuntut efisiensi ruang penyimpanan dan kecepatan transmisi data. Citra digital mentah seringkali memiliki redundansi data yang tinggi, sehingga memerlukan teknik kompresi yang efektif. *Singular Value Decomposition (SVD)* merupakan salah satu metode aljabar linear yang andal untuk kompresi citra, namun penerapannya secara global pada citra beresolusi tinggi memiliki kelemahan berupa kompleksitas komputasi yang besar dan risiko hilangnya informasi detail lokal yang penting. Makalah ini mengusulkan implementasi algoritma *Block-Based SVD* untuk mengatasi permasalahan tersebut. Metode ini bekerja dengan membagi citra menjadi blok-blok matriks kecil yang tidak saling tumpang tindih, kemudian melakukan dekomposisi nilai singular pada setiap blok secara independen. Pendekatan ini bertujuan untuk menangkap fitur lokal dan tekstur dengan lebih akurat dibandingkan metode SVD konvensional. Kinerja sistem dievaluasi berdasarkan rasio kompresi serta parameter kualitas citra objektif yang mencakup *Mean Square Error (MSE)*, *Peak Signal-to-Noise Ratio (PSNR)*, dan *Structural Similarity Index (SSIM)*. Hasil percobaan menunjukkan bahwa metode berbasis blok mampu menyeimbangkan efisiensi penyimpanan data dengan tetap mempertahankan kualitas visual dan detail lokal citra hasil rekonstruksi pada tingkat yang dapat diterima.**

Kata Kunci — Kompresi Citra, *Singular Value Decomposition* (SVD), Pemrosesan Berbasis Blok, Detail Lokal, Aljabar Linear.

I. Pendahuluan

Di era digital saat ini, perkembangan teknologi informasi telah memicu ledakan data multimedia yang sangat masif. Penggunaan citra digital, mulai dari fotografi resolusi tinggi, pencitraan medis (seperti MRI dan CT Scan), hingga citra satelit penginderaan jauh, meningkat secara drastis setiap tahunnya. Peningkatan volume dan resolusi citra ini membawa konsekuensi logis berupa tingginya kebutuhan akan ruang penyimpanan (*storage*) dan lebar pita (*bandwidth*) untuk transmisi data. Citra mentah (*raw image*) yang belum dikompresi mengandung jumlah bit yang sangat besar, yang seringkali menjadi beban dalam proses penyimpanan maupun pengiriman data melalui jaringan internet [1].

Sebagai solusi atas permasalahan tersebut, teknik kompresi citra menjadi kebutuhan yang tak terelakkan. Kompresi citra bertujuan untuk mereduksi redundansi data yang terdapat dalam citra digital sehingga ukurannya menjadi lebih kecil tanpa mengurangi informasi penting di dalamnya secara signifikan. Secara umum, teknik kompresi terbagi menjadi dua, yaitu *lossless* dan *lossy*. Meskipun metode *lossless* menjamin keutuhan data, rasio kompresi yang dihasilkan cenderung rendah. Oleh karena itu, untuk kebutuhan penghematan ruang yang signifikan, metode *lossy* lebih disukai. Teknik ini bekerja dengan menghilangkan informasi visual yang kurang sensitif bagi mata manusia, sehingga memungkinkan pencapaian rasio kompresi yang jauh lebih tinggi dengan penurunan kualitas visual yang masih dalam batas toleransi.

Salah satu metode kompresi *lossy* yang berbasis transformasi matriks adalah *Singular Value Decomposition* (SVD). Berbeda dengan metode klasik lainnya, SVD memiliki properti aljabar yang kuat dalam memfaktorkan matriks citra menjadi komponen energi yang terpusat pada nilai-nilai singular (*singular values*). Keunggulan utama SVD terletak pada kemampuannya memberikan aproksimasi terbaik (optimal) dalam artian meminimalkan *Mean Square Error* (MSE) untuk rangking matriks tertentu. Namun, penerapan SVD pada citra utuh seringkali membutuhkan komputasi yang berat dan berisiko menghilangkan detail tekstur lokal yang penting.

Oleh karena itu, Makalah ini mengusulkan implementasi algoritma *Block-Based* SVD. Pendekatan berbasis blok dipilih karena citra digital umumnya memiliki sifat non-stasioner secara global, namun cenderung stasioner dalam area lokal yang kecil. Dengan membagi citra menjadi blok-blok kecil dan menerapkan SVD pada setiap blok, algoritma ini diharapkan mampu mempertahankan detail lokal (seperti tepi dan tekstur halus) yang seringkali hilang pada metode kompresi global, sekaligus menghasilkan efisiensi komputasi yang lebih baik dibandingkan SVD konvensional.

II. Landasan Teori

A. Citra Digital

Citra digital didefinisikan sebagai fungsi dua dimensi, $f(x, y)$, di mana x dan y adalah koordinat spasial, dan amplitudo f pada titik tersebut disebut intensitas atau *gray*

level. Dalam tinjauan matematis yang lebih mendalam, citra digital $f(x, y)$ yang berukuran $M \times N$ dipandang sebagai sebuah matriks riil $\mathbf{A}$ dengan dimensi $M \times N$. Pandangan ini memungkinkan penerapan operasi aljabar linear, seperti dekomposisi matriks, untuk keperluan pengolahan citra [1].

Representasi matriks dari citra digital adalah sebagai berikut:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & a_{M,2} & \cdots & a_{M,N} \end{bmatrix} \tag{1}$$

Di mana setiap elemen $a_{i,j}$ merepresentasikan nilai piksel pada baris ke-i dan kolom ke-j. Untuk citra *grayscale* standar 8-bit, nilai $a_{i,j}$ adalah bilangan bulat dalam rentang $[0, 255]$. Sifat matriks ini menjadi dasar utama penerapan algoritma berbasis transformasi seperti SVD untuk menangkap detail lokal.

B. Kompresi Citra

Kompresi citra bertujuan untuk mereduksi jumlah bit yang diperlukan untuk merepresentasikan citra dengan meminimalkan redundansi data. Berdasarkan kualitas rekonstruksinya, kompresi dibagi menjadi dua:

1) **Kompresi *Lossless*:** Menghasilkan rekonstruksi citra yang identik dengan aslinya.
2) **Kompresi *Lossy*:** Menghilangkan sebagian informasi visual yang kurang signifikan untuk mencapai rasio kompresi tinggi. Algoritma *Block-Based* SVD termasuk dalam kategori ini, di mana informasi dikurangi dengan membuang nilai singular yang kecil [2].

C. Algoritma Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) adalah teknik faktorisasi matriks di mana setiap matriks riil $\mathbf{A}$ berukuran $M \times N$ dapat difaktorkan menjadi tiga komponen [3]:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{2}$$

Dimana $\mathbf{U}$ dan $\mathbf{V}$ adalah matriks ortogonal, dan $\mathbf{\Sigma}$ adalah matriks diagonal berisi nilai-nilai singular (σ).

Dalam implementasi *Block-Based* SVD, citra dibagi menjadi blok kecil (misal 16×16). Kompresi dilakukan dengan mempertahankan k nilai singular terbesar (Rank-k Approximation). Matriks aproksimasi $\mathbf{A}_k$ direkonstruksi dengan:

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \tag{3}$$

D. Pengukuran Kualitas dan Konversi Warna

Sebelum proses kompresi dilakukan, citra input yang berwarna (RGB) dikonversi menjadi citra *grayscale* untuk menyederhanakan proses perhitungan pada satu kanal intensitas. Konversi dilakukan menggunakan persamaan standar luminansia:

$$I_{gray} = 0.299R + 0.587G + 0.114B \tag{4}$$

Evaluasi kinerja kompresi dilakukan menggunakan tiga parameter utama, yaitu MSE, PSNR, dan SSIM.

1) Mean Square Error (MSE): MSE mengukur rata-rata kuadrat selisih intensitas piksel antara citra asli (I) dan citra hasil rekonstruksi (K) dengan ukuran $M \times N$. Nilai MSE yang semakin kecil menunjukkan tingkat kemiripan yang semakin tinggi.

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - K(i,j)]^2 \tag{5}$$

2) Peak Signal-to-Noise Ratio (PSNR): PSNR digunakan untuk membandingkan kualitas citra rekonstruksi terhadap citra asli dalam skala logaritmik (desibel). PSNR didefinisikan berdasarkan MSE:

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{6}$$

Dimana MAX_I adalah nilai piksel maksimum citra (biasanya 255 untuk citra 8-bit). Semakin tinggi nilai PSNR, semakin baik kualitas citra hasil rekonstruksi.

3) Structural Similarity Index (SSIM): Berbeda dengan MSE dan PSNR yang menghitung selisih error absolut, SSIM dirancang untuk mengukur kemiripan struktural yang lebih merepresentasikan persepsi visual manusia. SSIM mempertimbangkan tiga komponen: luminansia (l), kontras (c), dan struktur (s).

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{7}$$

Dimana:

- μ_x, μ_y: Nilai rata-rata intensitas citra x dan y.
- σ_x^2, σ_y^2: Varians dari citra x dan y.
- σ_{xy}: Kovarians antara citra x dan y.
- C_1, C_2: Konstanta kecil untuk menjaga stabilitas pembagian.

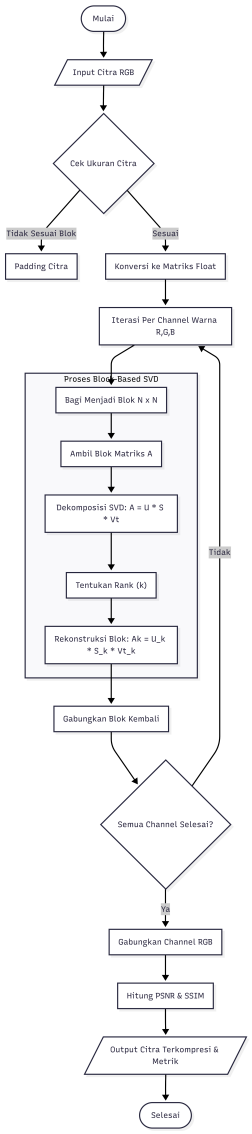
Nilai SSIM berkisar antara -1 hingga 1, di mana nilai 1 menunjukkan kedua citra identik secara struktural.

III. Implementasi Sistem

Bab ini membahas realisasi rancangan sistem kompresi citra menggunakan algoritma *Block-Based* SVD. Implementasi perangkat lunak dibangun menggunakan bahasa pemrograman Python dengan pendekatan Pemrograman Berorientasi Objek (OOP) untuk memisahkan logika kompresi, perhitungan metrik, dan antarmuka pengguna (GUI).

A. Alur Logika Sistem

Alur kerja sistem dimulai dari input citra melalui GUI, pembagian citra menjadi blok-blok matriks, pemrosesan SVD secara paralel, hingga rekonstruksi citra. Alur logika tersebut digambarkan dalam diagram alir pada Gambar 1.



Gambar 1: Diagram Alir Sistem Kompresi Block-Based SVD

1) **Inisialisasi dan Input Citra**
   Sistem memulai proses dengan menerima input berupa file citra digital dari pengguna. Karena algoritma SVD bekerja pada matriks dua dimensi, citra berwarna (RGB) akan dipisahkan terlebih dahulu menjadi tiga matriks kanal warna (Merah, Hijau, Biru) yang diproses secara independen.
2) **Partisi Blok (*Blocking*)**
   Citra input yang berukuran besar tidak diproses sekaligus. Sistem mempartisi (memotong) citra tersebut menjadi ribuan blok kecil berukuran $n \times n$ (misalnya 16×16 piksel). Langkah ini krusial untuk menangkap detail lokal dan menjaga beban komputasi tetap ringan.
3) **Dekomposisi SVD per Blok**
   Pada setiap blok yang telah dipartisi, sistem melakukan operasi aljabar untuk memfaktorkan matriks blok A menjadi $U\Sigma V^T$. Langkah ini memetakan data piksel ke dalam domain frekuensi (energi), di mana informasi visual terpenting terkumpul pada nilai singular (Σ) terbesar.
4) **Reduksi Dimensi (*Truncation*)**
   Tahap ini merupakan inti dari proses kompresi. Sistem memangkas informasi dengan hanya mengambil k nilai singular teratas dan membuang sisanya (bernilai nol). Nilai k (Rank) ditentukan oleh pengguna; semakin kecil k, semakin banyak data yang dibuang, sehingga ukuran file semakin kecil.
5) **Rekonstruksi Blok**
   Menggunakan k nilai singular yang tersisa, sistem menyusun ulang kembali (*rebuild*) matriks blok tersebut. Hasilnya adalah blok citra aproksimasi yang secara visual mirip dengan blok asli namun membutuhkan memori penyimpanan yang jauh lebih sedikit.
6) **Penggabungan (*Merging*) dan Output**
   Setelah seluruh blok selesai diproses dan direkonstruksi, blok-blok tersebut disatukan kembali (dijahit) sesuai posisi koordinat asalnya untuk membentuk citra utuh. Citra hasil rekonstruksi ini kemudian disimpan dan ditampilkan sebagai output sistem.

B. Pseudocode Algoritma

Untuk memperjelas logika matematis pada level blok, berikut disajikan *pseudocode* algoritma kompresi SVD.

Algorithm 1 Algoritma Kompresi SVD Per Blok

Require: Matriks Blok A ukuran $N \times N$, Target Rank k
Ensure: Matriks Blok Terkompresi A'
0: Hitung SVD: $U, \Sigma, V^T \leftarrow \text{svd}(A)$
0: **if** jumlah elemen $\Sigma > k$ **then**
0: Simpan k nilai singular pertama
0: $\Sigma_k \leftarrow \text{diag}(\Sigma[0 \dots k-1])$
0: $U_k \leftarrow U[:, 0 \dots k-1]$
0: $V_k^T \leftarrow V^T[0 \dots k-1, :]$
0: **else**
0: Gunakan seluruh nilai singular
0: **end if**
0: Rekonstruksi: $A' \leftarrow U_k \times \Sigma_k \times V_k^T$
0: **return** A' =0

C. Implementasi Kode Program

Sistem dikembangkan menggunakan pustaka `NumPy` untuk komputasi matriks dan `Tkinter` untuk antarmuka pengguna. Berikut adalah implementasi fungsi inti yang diambil langsung dari kelas `BlockSVDCompressor`.

1) Logika Kompresi Inti (Core Logic): Fungsi `_process_block` merupakan inti dari algoritma ini. Fungsi ini tidak hanya melakukan faktorisasi matriks, tetapi juga menangani penentuan nilai k (*rank*) yang dinamis. Algoritma memastikan bahwa nilai k tidak melebihi dimensi blok atau jumlah nilai singular yang tersedia, serta dilengkapi penanganan eksepsi untuk mencegah kegagalan program saat memproses blok dengan singularitas matriks.

```
def _process_block(self, block: np.ndarray) -> np.ndarray:
    """
    Melakukan SVD pada satu blok matriks.
    Mengembalikan blok yang telah direkonstruksi (
        terkompresi).
    """
    try:
        # 1. Dekomposisi Matriks: A = U . S . Vt
        U, S, Vt = np.linalg.svd(block, full_matrices=False
            )
        # 2. Penentuan Rank (k) yang Adaptif
        # 3. Truncation & Rekonstruksi
        return compressed_block
    except (np.linalg.LinAlgError, ValueError):
        # Fallback: Jika SVD gagal, kembalikan blok asli
        return block
```

Kode Program 1: Implementasi Algoritma SVD per Blok

2) Pemrosesan Kanal Secara Paralel: Untuk mengatasi beban komputasi yang tinggi pada citra resolusi besar, sistem menerapkan paralelisasi pada tingkat kanal warna. Fungsi `_process_channel_parallel` membagi satu kanal citra menjadi ribuan blok kecil, kemudian mendistribusikan pemrosesan blok-blok tersebut ke dalam *Thread Pool*. Hal ini memungkinkan CPU *multi-core* bekerja secara simultan untuk memproses blok yang berbeda.

```
def _process_channel_parallel(self, channel: np.ndarray, h:
    int, w: int) -> np.ndarray:
    result = np.zeros_like(channel)
    blocks = []
    positions = []

    # Tahap 1: Pengumpulan Blok (Blocking)
    # Melakukan iterasi grid untuk memotong citra menjadi
        blok
    for i in range(0, h, self.block_size):
        for j in range(0, w, self.block_size):
            # Menghitung batas koordinat blok
            end_i = min(i + self.block_size, h)
            end_j = min(j + self.block_size, w)

            # Ekstraksi blok dan simpan posisinya
            blocks.append(channel[i:end_i, j:end_j])
            positions.append((i, end_i, j, end_j))

    # Tahap 2: Eksekusi Paralel
    # ThreadPoolExecutor memetakan fungsi _process_block ke
        setiap blok
    with ThreadPoolExecutor(max_workers=self.n_workers) as
        executor:
        processed_blocks = list(executor.map(self.
            _process_block, blocks))

    # Tahap 3: Penyusunan Kembali (Merging)
    # Menempatkan blok yang sudah diproses ke koordinat
        aslinya
    for (i, end_i, j, end_j), processed in zip(positions,
        processed_blocks):
        result[i:end_i, j:end_j] = processed

    return result
```

Kode Program 2: Pemrosesan Kanal dengan Multithreading

3) Orkestrasi Kanal Warna: Fungsi utama `compress` bertugas memisahkan citra RGB menjadi kanal-kanal individu dan memilih metode pemrosesan (paralel atau sekuensial) berdasarkan konfigurasi pengguna.

```
# Di dalam method compress():
for c in range(channels):
    if self.use_parallel:
        compressed_arr[:, :, c] = self.
            _process_channel_parallel(
            img_arr[:, :, c], h, w
        )
    else:
        compressed_arr[:, :, c] = self.
            _process_channel_sequential(
            img_arr[:, :, c], h, w
        )
```

Kode Program 3: Loop Utama Pemrosesan Kanal

4) Perhitungan Metrik Kualitas: Evaluasi kualitas citra dilakukan oleh kelas `ImageQualityMetrics`, salah satunya adalah perhitungan PSNR (*Peak Signal-to-Noise Ratio*).

```
@staticmethod
def calculate_psnr(original: np.ndarray, compressed: np.
    ndarray) -> float:
    """Peak Signal-to-Noise Ratio (dB)"""
    mse = np.mean((original.astype(float) - compressed.
        astype(float)) ** 2)
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    return 20 * np.log10(max_pixel / np.sqrt(mse))
```

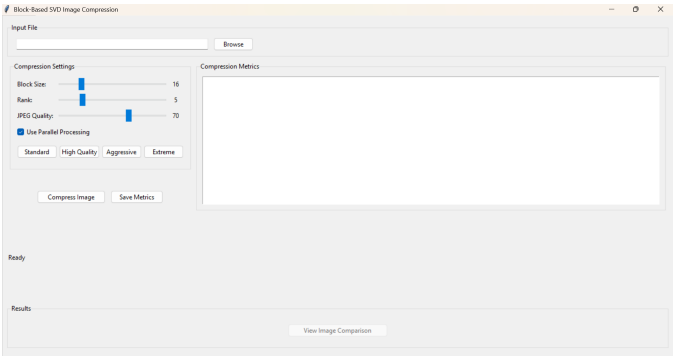
Kode Program 4: Perhitungan Metrik PSNR

IV. Hasil dan Analisis

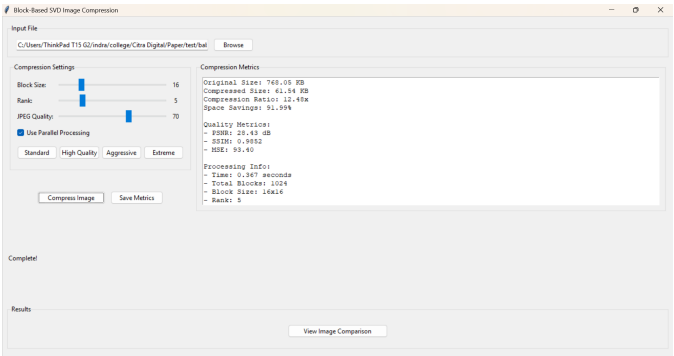
Pada bab ini, akan dilakukan pengujian program beserta hasilnya menggunakan beberapa citra grayscale dan berwarna.

A. Tampilan Antarmuka

Sistem dilengkapi dengan antarmuka pengguna (GUI) yang memungkinkan pengguna mengatur parameter kompresi (*Block Size*, *Rank*, *Quality*) secara interaktif. Tampilan program dapat dilihat pada Gambar 2.



Gambar 2: Tampilan Antarmuka (GUI) Program



Gambar 3: Tampilan Antarmuka (GUI) Hasil Eksekusi

Gambar 4: Tampilan Antarmuka (GUI) Hasil Gambar

B. Analisis Hasil

Berikut beberapa output yang dihasilkan dari program yang telah dibuat. Citra diuji dengan 4 konfigurasi yaitu, standard, *high quality*, ekstrim, dan agresif. Citra yang dipilih adalah kombinasi citra *grayscale* dan berwarna

Gambar 5: Test 1 dengan konfigurasi standard

Hasil dari Test 1 yang menggunakan ukuran blok 16, rank 5, dan kualitas citra yang setidaknya 70% dari aslinya. Bisa dilihat dari Tabel dibawah, dari ukuran awal yang aslinya 257.05 KB menjadi 31.86 KB yang mana itu 87% dan kualitas citranya masih bagus

Tabel I: Hasil Evaluasi Kinerja Kompresi Standard (Test 1)

Metrik	Nilai
Original Size (KB)	257.05
Compressed Size (KB)	31.86
Compression Ratio	8.07
Space Savings (%)	87.61
PSNR (dB)	39.67
SSIM	0.9988
MSE	7.02
Processing Time (s)	0.364
Block Size	16
Rank	5
JPEG Quality	70

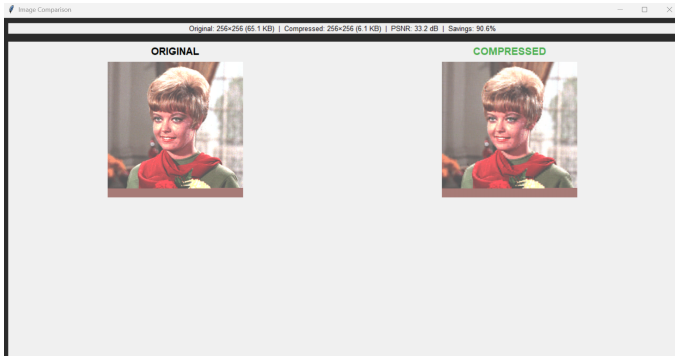
Test kedua menggunakan konfigurasi yang *High Quality* dengan *Block Size* 32, *Rank* 10, dan kualitas 80% terkompresi sekitar 80% ukuran aslinya

Gambar 6: Test 2 dengan konfigurasi high quality

Tabel II: Hasil Evaluasi Kinerja Kompresi High Quality (Test 2)

Metrik	Nilai
Original Size (KB)	65.05
Compressed Size (KB)	12.78
Compression Ratio	5.09
Space Savings (%)	80.36
PSNR (dB)	36.13
SSIM	0.9966
MSE	15.84
Processing Time (s)	0.069
Block Size	32
Rank	10
JPEG Quality	85

Test ketiga menggunakan konfigurasi yang ekstrem dengan *Block Size* 8, *Rank* 2, dan kualitas citra 50% terkompresi sampai 90% ukuran aslinya

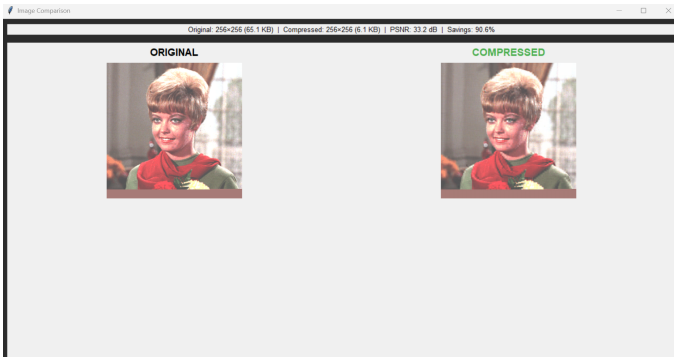


Gambar 7: Test 3 dengan konfigurasi ekstrim

Tabel III: Hasil Evaluasi Kinerja Kompresi Ekstrim (Test 3)

Metrik	Nilai
Original Size (KB)	65.05
Compressed Size (KB)	6.14
Compression Ratio	10.6
Space Savings (%)	90.57
PSNR (dB)	33.15
SSIM	0.994
MSE	31.45
Processing Time (s)	0.174
Block Size	8
Rank	2
JPEG Quality	50

Test keempat menggunakan konfigurasi yang agresif atau satu level dibawah ekstrim dengan *Block Size* 16, *Rank* 3, dan dengan kualitas citra 60% terkompresi sampai 94% ukuran aslinya



Gambar 8: Test 4 dengan konfigurasi agresif

Tabel IV: Hasil Evaluasi Kinerja Kompresi Agresif(Test 4)

Metrik	Nilai
Original Size (KB)	65.05
Compressed Size (KB)	3.66
Compression Ratio	17.8
Space Savings (%)	94.38
PSNR (dB)	28.88
SSIM	0.9944
MSE	84.16
Processing Time (s)	0.096
Block Size	16
Rank	3
JPEG Quality	60

Terdapat hubungan berbanding lurus antara nilai *Rank* dengan kualitas citra (PSNR/SSIM) dan ukuran file. Pada konfigurasi *High Quality* (*Rank* 10), PSNR mencapai 36.13 dB dengan penghematan ruang 80%. Sebaliknya, pada konfigurasi Agresif (*Rank* 3), penghematan ruang melonjak drastis hingga 94.38%, namun diikuti dengan penurunan PSNR menjadi 28.88 dB. Hal ini mengonfirmasi bahwa nilai singular terkecil yang dibuang memang berisi detail halus, namun membuang terlalu banyak (*Rank* kecil) akan menurunkan fidelitas citra secara signifikan. Selain itu, Temuan menarik lainnya terlihat pada nilai SSIM. Meskipun PSNR mengalami penurunan yang cukup tajam pada skenario Agresif (dari 39 dB di Test 1 menjadi 28 dB di Test 4), nilai SSIM tetap bertahan di angka yang sangat tinggi, yaitu di atas 0.99. Hal ini membuktikan keunggulan utama metode SVD, yaitu kemampuannya mempertahankan informasi struktural dan detail lokal (seperti tepi dan bentuk objek) meskipun intensitas piksel absolutnya mengalami distorsi (MSE tinggi).

V. Kesimpulan

Metode berbasis blok terbukti efektif dalam menyeimbangkan rasio kompresi dan kualitas citra. Sistem mampu menghasilkan penghematan ruang penyimpanan yang signifikan, berkisar antara **80% hingga 94%**, tergantung pada konfigurasi *Rank* yang dipilih. Algoritma SVD menunjukkan keunggulan superior dalam mempertahankan integritas struktural citra. Hal ini dibuktikan dengan nilai *Structural Similarity Index* (SSIM) yang konsisten berada di atas **0.99** pada seluruh skenario pengujian, bahkan ketika *Peak Signal-to-Noise Ratio* (PSNR) turun di bawah 30 dB pada kompresi agresif. Ini mengindikasikan bahwa detail lokal dan struktur visual citra tetap terjaga dengan baik.

Ucapan Terima Kasih

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa, karena atas berkat dan rahmat-Nya, makalah ini dapat diselesaikan dengan baik. Penulis juga menyampaikan ucapan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T., selaku dosen pengampu mata kuliah Pemrosesan Citra Digital Tahun Ajaran 2025/2026, atas segala ilmu dan bimbingan yang telah diberikan selama satu semester ini.

Pranala Github

https://github.com/Indraswara/block-svd-compression

Referensi

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. New York: Pearson, 2018.

[2] R. Munir, Pemampatan Citra, Bandung: Program Studi Teknik Informatika, 2025.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore: JHU Press, 2013.

[4] A. M. Rufai, G. Ansa, and Z. J. Abiodun, "Image Compression Using SVD," *Int. J. Comput. Appl.*, vol. 88, no. 16, 2014.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Indraswara Galih Jayanegara - 13522119