

Laporan Tugas Besar 2 IF 2123 Aljabar Linier dan Geometri  
Aplikasi Nilai Eigen dan Vektor Eigen dalam Kompresi Gambar



Kelompok 14 - CitraBodyLotion

Muhammad Garebaldhie Er Rahman - 13520029

Fikri Khoiron Fadhila - 13520056

Marchotridyo - 13520119

Semester I Tahun 2021/2022

## Bab 1

### Deskripsi Masalah

Spesifikasi Tugas Besar 2:

A. Buatlah program kompresi gambar dengan memanfaatkan algoritma SVD dalam bentuk website local sederhana, dengan spesifikasi:

- i. Mampu menerima *file* gambar beserta *input* tingkat kompresi gambar.
- ii. Website mampu menampilkan gambar *input*, *output*, *runtime* algoritma, dan presentase hasil kompresi gambar (perubahan jumlah *pixel* gambar).
- iii. *File output* hasil kompresi dapat diunduh melalui website.
- iv. Kompresi gambar tetap mempertahankan warna dari gambar asli.
- v. Bahasa pemrograman yang boleh digunakan adalah Python, Javascript, dan Go
- vi. Penggunaan framework untuk back end dan front end website dibebaskan. Contoh framework website yang bisa dipakai adalah Flask, Django, React, Vue, dan Svelte.
- vii. Diperbolehkan menggunakan library pengolahan citra seperti OpenCV2, PIL, atau image dari Go.

## Bab 2

### Teori Singkat

#### A. Perkalian Matriks

Dalam matematika, perkalian matriks adalah suatu operasi biner dari dua matriks yang menghasilkan sebuah matriks. Agar dua matriks dapat dikalikan, banyaknya kolom pada matriks pertama harus sama dengan banyaknya baris pada matriks kedua. Matriks hasil perkalian keduanya, akan memiliki baris sebanyak baris matriks pertama, dan kolom sebanyak kolom matriks kedua. Perkalian matriks A dan B dinyatakan sebagai AB.

Bentuk perkalian dua matriks:

- i. Jika matriks  $A = a_{ij}$ , matriks  $B = b_{ij}$  dan matriks  $C = c_{ij}$  maka:  $C = A \times B$  dengan  $c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$  untuk setiap  $I$  dan  $j$  (Leon, 2001).
- ii. Contoh perkalian dua matriks:

$$1. \begin{bmatrix} 3 & 1 \\ 4 & 2 \end{bmatrix} \times \begin{bmatrix} 4 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 13 & 11 \\ 18 & 18 \end{bmatrix}$$

Figure 1 Perkalian dua matriks

#### B. Nilai Eigen dan Vektor Eigen

Nilai Eigen  $\lambda$  adalah nilai karakteristik dari suatu berukuran  $n \times n$ , sementara vektor Eigen adalah vector kolom bukan nol yang bila dikalikan dengan suatu matriks berukuran  $n \times n$  akan menghasilkan vektor lain yang memiliki nilai kelipatan dari vektor Eigen itu sendir. Untuk setiap nilai Eigen ada pasangan vektor Eigen yang berbeda, namun tidak semua persamaan matriks memiliki nilai Eigen dan vektor Eigen.

a. Definisi formal:

- i. Jika  $T$  adalah transformasi linier dari ruang vector  $V$  di atas bidang  $F$  ke dalam dirinya sendiri dan  $v$  adalah vector tak nol

dalam  $V$ , maka  $v$  adalah vector eigen dari  $T$  jika  $T(v)$  adalah kelipatan scalar dari  $v$ . Ini dapat ditulis sebagai

$$T(v) = \lambda v$$

ii. Jika  $V$  berdimensi-hingga, persamaan di atas ekuivalen dengan

$$Au = \lambda u$$

Dimana  $A$  adalah representasi matriks dari  $T$ , dan  $u$  adalah vector koordinat dari  $V$ .

Persamaan karakteristik bisa diperoleh dengan cara:

$$\det(A - \lambda I) = 0$$

Untuk vector eigen bisa dicari dengan melakukan substitusi nilai Eigen ke dalam persamaan  $(A - \lambda I)x = 0$  dan didapatkan suatu persamaan baru, kemudian dilakukan operasi baris elementer sehingga akan diperoleh vector eigen untuk Nilai Eigen tersebut.

### C. Matriks *Singular Value Decomposition (SVD)*

*Singular Value Decomposition (SVD)* adalah faktorisasi dari matriks real atau kompleks. Secara khusus, dekomposisi *single value* dari  $m \times n$  matriks kompleks  $M$  adalah faktorisasi dari bentuk  $U\Sigma V^T$ , dimana  $U$  adalah  $m \times m$  matriks,  $\Sigma$  adalah  $m \times n$  demgam bilangan real non negatif pada diagonal, dan  $V$  adalah  $n \times n$  matriks. Jika  $M$  real,  $U$  dan  $V$  juga dapat dijamin matriks orthogonal real. Entri dari diagonal  $\sigma_i = \Sigma_{ii}$  dari  $\Sigma$  dikenal sebagai nilai-nilai singular dari  $M$ . Matriks  $U$  dan matriks  $V$  masing masing disebut dengan matriks singular kanan, dan matriks singular kiri.

Dengan nilai singular kiri merupakan normalisasi nilai vector eigen dari  $AA^T$  dan nilai singular kanan merupakan normalisasi nilai vector eigen dari  $A^TA$ .

$$M_{m \times n} = U_{m \times m} \Sigma_{m \times n} V^T_{n \times n}$$

$$U_{m \times m} \quad U^T_{m \times m} = I_m$$

$$V_{n \times n} \quad V^T_{n \times n} = I_n$$

Figure 2 visualisasi perkalian matriks dalam SVD

*Singular Value Decomposition (SVD)* dapat diaplikasikan di matematika untuk menghitung pseudoinverse, aproksimasi matriks. Selain itu SVD juga berguna di bidang pengolahan gambar, khusunya kompresi gambar.

## Bab 3

### Implementasi Program

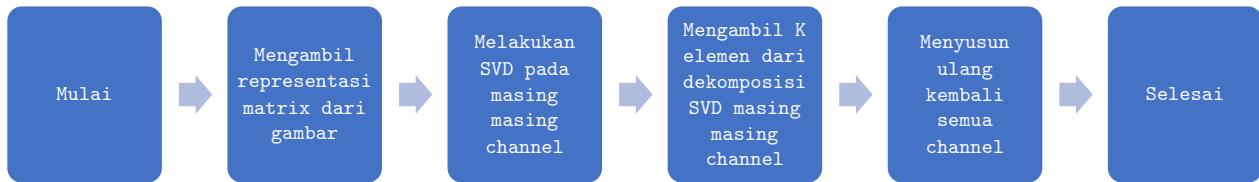
#### A. app.py

Nama	kembalian	Parameter	Deskripsi
Simultaneous_power_iteration	Array nilai eigen 1 Dimensi dan Matrix eigenvector	Matrix Awal, integer k	Mencari nilai eigen dan eigen vektor
Image.open	Matrix Image object	Path atau lokasi gambar	Membuka dan mengidentifikasi file gambar yang diberikan
SVD	Matrix dekomposisi	Matrix	Melakukan dekomposisi SVD dan mengembalikan
compress	File, runtime algoritma,	Namafile, ext, time	Melakukan perhitungan pixel difference percentage dan runtime kompresor

Pada pembuatan website *image-compressor* kami menggunakan modul FLASK untuk membuat API dan struktur backend dari website. Untuk frontend kami menggunakan HTML, CSS dan javascript. Selain itu kami juga menggunakan modul numpy untuk melakukan operasi matriks seperti perkalian, modul PIL untuk membuka dan

mengidentifikasi gambar, modul time untuk menganalisis runtime, modul os untuk membuka path.

Secara garis besar, alur dari program kami adalah



Ada beberapa algoritma menarik yang kami gunakan, salah satunya adalah simultaneous\_power\_iteration.



```
1 def simultaneous_power_iteration(A, k):
2     n, m = A.shape
3     Q = np.random.rand(n, k)
4     Q, _ = np.linalg.qr(Q)
5     Q_prev = Q
6
7     for i in range(200):
8         Z = A.dot(Q)
9         Q, R = np.linalg.qr(Z)
10
11    err = ((Q - Q_prev) ** 2).sum()
12    Q_prev = Q
13    if err < 1e-3:
14        break
15
16    return np.abs(np.diag(R)), Q
```

Figure 3 Algoritma simultaneous\_power\_iteration

Algortima diatas adalah algoritma yang kami gunakan untuk mencari nilai eigen value dan eigen vector dari matriks dengan memanfatkan dekomposisi QR kemudian kami menggunakan range 200 untuk mencari tingkat presisi dari nilai eigen, dan kami menggunakan presisi dengan tingkat 200 dengan mempertimbangkan runtime dari program agar tidak terlalu membutuhkan banyak waktu. Algoritma ini menerima parameter berupa matrix A dan nilai K yang akan diambil dengan nilai K terbesar.

Selain itu ada juga algoritma SVD (Singular value decomposition) yang kami gunakan untuk mendekomposisi matrix gambar.



```
1 # Algoritma SVD
2 def SVD(Matrix, k):
3     # 0 < compressionRate < 1
4     width, height = np.shape(Matrix)
5     # k = compressionRate * min(width, height)
6     Matrix_R = Matrix @ np.transpose(Matrix)
7     SIG, U = simultaneous_power_iteration(Matrix_R, int(k))
8     uRow, uWidth = U.shape
9
10    U_2 = np.pad(U, ((0,0), (0, height - uWidth)))
11    SIG = np.sqrt(SIG)
12    SIG = np.diag(SIG)
13    SIGRow, SIGWidth = SIG.shape
14    SIG_2 = np.pad(SIG, ((0, height - SIGRow), (0, width - SIGWidth)))
15
16    VT = np.linalg.inv(SIG) @ np.transpose(U) @ Matrix
17    VTHeight, VTwidht = VT.shape
18    VT_2 = np.pad(VT, ((0, width - VTHeight), (0,0)))
19    return np.clip(U_2 @ SIG_2 @ VT_2, 0, 255)
```

Figure 4 Algoritma SVD

Algoritma SVD diatas dimulai dari membuat matrix R yang merupakan perkalian dari input Matrix dikalikan dengan Matrix transpose kemudian melempar matrix R ke simultaneous\_power\_iteration untuk mendapat eigen value dan eigen vektornya yang kemudian dipakai sebagai matrix singular kiri. Kami juga menggunakan `np.pad` untuk melakukan padding pada matrix, agar ukuran matrix menjadi  $m \times m$ .

Dari matrix tersebut kami dapatkan matrix singular dengan `np.sqrt(SIG)` dan kami susun matrixnya menjadi matrix diagonal. Kemudian kami sesuaikan ukuran matrix menjadi  $m \times n$ , dan membatasi nilai dari matrix itu sendiri dengan rentang nilai element 0 sampai 255.

Selain itu kami juga mempunyai algoritma `compressImage` yang merupakan main program kami.

```

1 def compressImage(filename, img, compressionRate):
2     with Image.open(img, mode="r") as im:
3         hasAlpha = False # Cek jika memiliki alpha channel.
4         channels = Image.Image.split(im)
5         R = np.asarray(channels[0]).astype(float)
6         G = np.asarray(channels[1]).astype(float)
7         B = np.asarray(channels[2]).astype(float)
8         if (len(channels) > 3):
9             # Ada channel alpha; hasAlpha = true
10            hasAlpha = True
11            A = channels[3]
12
13    imFinal = np.dstack((R,G,B)).astype(np.uint8)
14    imFinal = Image.fromarray(imFinal)
15
16    svdR = SVD(R, compressionRate)
17    svdG = SVD(G, compressionRate)
18    svdB = SVD(B, compressionRate)
19
20    if (hasAlpha):
21        IM_matr = np.dstack((svdR, svdG, svdB, A)).astype(np.uint8)
22    else:
23        IM_matr = np.dstack((svdR, svdG, svdB)).astype(np.uint8)
24
25    IM_Final = Image.fromarray(np.uint8(IM_matr))
26    IM_Final.save(f'static/compressed/{filename}')
27 # ===== End of fungsionalitas compress image =====
28

```

Figure 5 Algoritma main atau compressImage

Algortima diatas merupakan algoritma main kami. Kami menggunakan modul PIL untuk membuka gambar dan merepresentasikannya menjadi matrix, yang kemudian kami ‘split’ dengan *image.split(im)* untuk mendapatkan channel gambar (R, G, B, A) atau (red, gree, blue, opacity) dari gambar. Setelah itu kami lempar masing-masing channel ke fungsi SVD, untuk mendekomposisi channel lalu mengambil nilai K atau compressionRate sesuai kehendak pengguna. Terakhir, kami menyatukan kembali channel dan menyusunnya menjadi gambar lalu menyimpan gambar tersebut.

```
1 def compress(filename, ext, ctime):
2     if request.method == 'POST':
3         k = request.form.get('k')
4         start_time = time.time()
5         compressImage(f'{filename}{ext}', f"static/uploads/{filename}{ext}", k)
6         duration = round(time.time() - start_time, 2)
7         return redirect(f'/compress/{filename}/{ext}/{duration}')
8     origPath = f'static/uploads/{html.unescape(filename)}{ext}'
9     compressedPath = f'static/compressed/{html.unescape(filename)}{ext}'
10
11     # Menghitung pixel difference percentage
12     i1 = Image.open(origPath)
13     if (i1.mode == 'P'):
14         i1 = i1.convert('RGBA')
15     elif (i1.mode == 'L'):
16         i1 = i1.convert('RGB')
17     elif (i1.mode == 'LA'):
18         i1 = i1.convert('RGBA')
19     i2 = Image.open(compressedPath)
20     if (i2.mode == 'P'):
21         i2 = i2.convert('RGBA')
22     elif (i2.mode == 'L'):
23         i2 = i2.convert('RGB')
24     elif (i2.mode == 'LA'):
25         i2 = i2.convert('RGBA')
26     pairs = zip(i1.getdata(), i2.getdata())
27     if len(i1.getbands()) == 1:
28         # for gray-scale jpegs
29         dif = sum(abs(p1-p2) for p1,p2 in pairs)
30     else:
31         dif = sum(abs(c1-c2) for p1,p2 in pairs for c1,c2 in zip(p1,p2))
32
33     origPath = url_for('static', filename=f'uploads/{filename}{ext}')
34     compressedPath = url_for('static', filename=f'compressed/{filename}{ext}')
35     ncomponents = i1.size[0] * i1.size[1] * 3
36     diffPercentage = round((dif / 255.0 * 100) / ncomponents, 2)
37     return render_template('compress.html', origPath=origPath, compressedPath=compressedPath, time=ctime, diffPercentage=diffPercentage)
```

Algoritma menarik yang terakhir adalah algoritma compress, disini akan terjadi perhitungan perbedaan pixel gambar asli dengan hasil kompresi dan lama waktu kompresi. Untuk menghitung perbedaan pixel gambar, secara umum kami menggunakan persamaan penjumlahan selisih nilai komponen matrix masing masing channel dari matrix awal dikurangi matrix channel hasil kompresi lalu membaginya dengan 255 dan mengalikannya dengan 100 dan dibagi oleh banyaknya data/komponen untuk mendapatkan persentase perbedaannya. Sedangkan untuk menghitung durasi kompresi, kami menggunakan modul time, dengan menghitung selisih waktu selesai dengan waktu mulai program.

## Bab 4

### Eksperimen

1. Gambar Lanskap dimensi 1200 x 800



*Figure 6 gambar lanskap original*



*Figure 7 gambar lanskap sudah dikompres  $k = 50$*



*Figure 8 gambar lanskap sudah dikompreksi  $k = 10$*



*Figure 9 gambar lanskap sudah dikompres  $k = 100$*

Tabel perbandingan dengan nilai k sebagai variable bebas:

Nilai K	Ukuran Awal	Ukuran Akhir	Runtime (s)	Pixel difference
10	625 KB	135 KB	0.93	2.31%
50	625 KB	198 KB	2.83	7.33%
100	625 KB	231 KB	9.77	5.57%

2. Gambar Potrait resolusi 4K

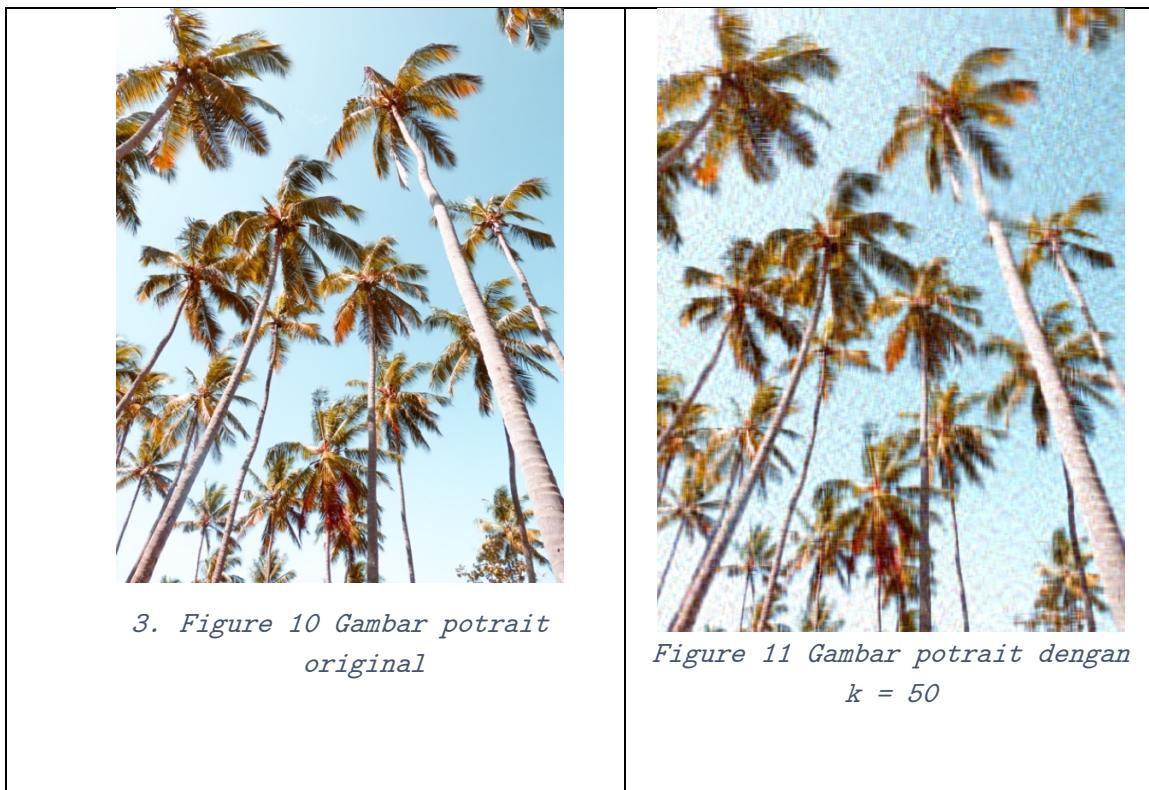




Figure 12 gambar potrait dengan  $k = 10$



Figure 13 Gambar potrait dengan  $k = 100$

Tabel perbandingan dengan nilai k sebagai variable bebas:

Nilai K	Ukuran Awal	Ukuran Akhir	Runtime (s)	Pixel difference
10	4.870 KB	1.035 KB	12.52	13.9%
50	4.870 KB	1.583 KB	44.92	8.54%
100	4.870 KB	1.993 KB	86.88	7.09%

3. Gambar Transparan dimensi 1000 x 1000 (persegi)



Tabel perbandingan dengan nilai k sebagai variable bebas:

Nilai K	Ukuran Awal	Ukuran Akhir	Runtime (s)	Pixel difference
10	170 KB	76 KB	2.43	2.41%
50	170 KB	85 KB	11.6	0.5%
200	170 KB	90 KB	6.36	1.49%

## Bab 5

### Kesimpulan, Saran, dan Refleksi

#### A. Kesimpulan

Hasil program kelompok kami berupa website yang dapat digunakan untuk melakukan kompresi gambar dengan tingkat kompresi sesuai kehendak pengguna melalui input berupa file gambar dan tingkat kompresi dan output berupa gambar awal, gambar yang sudah dikompresi, runtime algortima dan persentase hasil kompresi gambar. Gambar yang sudah dikompresi bisa diunduh oleh pengguna.

#### B. Saran

Dalam pengerjaan tugas besar ini, kami mengalami kesulitan dalam melakukan abstraksi dan menentukan algoritma yang efektif untuk mengatasi matriks dengan ukuran yang relatif besar seperti pixel pada gambar sehingga materi yang diajarkan di kelas tidak sepenuhnya *relate* dengan algoritma yang dipakai dalam menyelesaikan tugas besar ini. Materi yang diajarkan dikelas tidak bisa digunakan untuk memproses matriks dengan ukuran yang relative besar sehingga kami harus mencari algoritma yang lebih efisien dan tentunya lebih sulit untuk diimplementasikan daripada yang diajarkan di kelas. Oleh karena itu, kami menyarankan untuk sering melakukan eksplorasi dengan membaca paper dan dokumentasi. Selain itu kami menyarankan untuk berlatih menulis program dengan efisien dan kompleksitas yang lebih rendah.

#### C. Refleksi

Kami menemukan beberapa pelajaran berharga selama tugas besar ini. Komunikasi yang terjalin dengan baik sangat memudahkan kami dalam melakukan evaluasi setiap bagian dan memudahkan kami untuk melakukan penyelesaian masalah. Selain itu, kami juga belajar dalam mengelola alur kerja terutama saat

minggu-minggu yang padat dengan tugas. Kami juga belajar untuk menulis kode dengan kompleksitas yang efisien dan lebih cepat.

## Daftar Referensi

- [1] <http://eprints.umm.ac.id/37632/3/jiptumpp-gdl-nandarizky-50741-3-babii.pdf>
- [2] [https://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](https://en.wikipedia.org/wiki/Singular_value_decomposition)
- [3] [https://en.wikipedia.org/wiki/Eigenvalues\\_and\\_eigenvectors](https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors)
- [4] [http://mlwiki.org/index.php/Power\\_Iteration](http://mlwiki.org/index.php/Power_Iteration)
- [5] [https://www.wikiwand.com/en/Orthogonal\\_matrix](https://www.wikiwand.com/en/Orthogonal_matrix)
- [6] <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-19b-Singular-value-decomposition.pdf>
- [7] <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf>