LAPORAN TUGAS BESAR 2

IF2123 ALJABAR LINEAR DAN GEOMETRI

Disusun oleh:

Putri Nurhaliza	13520066
Jova Andres Riski Sirait	13520072
Adelline Kania Setiyawan	13520084

TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER 1 TAHUN 2021/2022

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Gambar adalah suatu hal yang sangat dibutuhkan pada dunia modern ini. Kita seringkali berinteraksi dengan gambar baik untuk mendapatkan informasi maupun sebagai hiburan. Gambar digital banyak sekali dipertukarkan di dunia digital melalui file-file yang mengandung gambar tersebut. Seringkali dalam transmisi dan penyimpanan gambar ditemukan masalah karena ukuran file gambar digital yang cenderung besar.

Kompresi gambar merupakan suatu tipe kompresi data yang dilakukan pada gambar digital. Dengan kompresi gambar, suatu file gambar digital dapat dikurangi ukuran filenya dengan baik tanpa mempengaruhi kualitas gambar secara signifikan. Terdapat berbagai metode dan algoritma yang digunakan untuk kompresi gambar pada zaman modern ini.

Three levels of JPG compression. The left-most image is the original. The middle image offers a medium compression, which may not be immediately obvious to the naked eye without closer inspection. The right-most image is maximally compressed.

Gambar 1. Contoh kompresi gambar dengan berbagai tingkatan

Sumber : Understanding Compression in Digital Photography (lifewire.com)

Salah satu algoritma yang dapat digunakan untuk kompresi gambar adalah algoritma SVD (Singular Value Decomposition). Algoritma SVD didasarkan pada teorema dalam aljabar linier yang menyatakan bahwa sebuah matriks dua dimensi dapat dipecah menjadi hasil perkalian dari 3 sub-matriks yaitu matriks ortogonal U, matriks diagonal S, dan transpose dari matriks ortogonal V. Dekomposisi matriks ini dapat dinyatakan sesuai persamaan berikut.

$$A_{m\times n} = U_{m\times m}\ S_{m\times n}\ V^T_{nxn}$$

Gambar 1. Algoritma SVD

Matriks U adalah matriks yang kolomnya terdiri dari vektor eigen ortonormal dari matriks AA^T. Matriks ini menyimpan informasi yang penting terkait baris-baris matriks awal, dengan informasi terpenting disimpan di dalam kolom pertama. Matriks S adalah matriks diagonal yang berisi akar dari nilai eigen matriks U atau V yang terurut menurun. Matriks V adalah matriks yang kolomnya terdiri dari vektor eigen ortonormal dari matriks A^TA. Matriks ini menyimpan informasi yang penting terkait kolom-kolom matriks awal, dengan informasi terpenting disimpan dalam baris pertama.

Gambar 2. Ilustrasi Algoritma SVD dengan rank k

Dapat dilihat di gambar di atas bahwa dapat direkonstruksi gambar dengan banyak *singular values* **k** dengan mengambil kolom dan baris sebanyak k dari U dan V serta *singular value* sebanyak **k** dari S atau Σ terurut dari yang terbesar. Kita dapat mengaproksimasi suatu gambar yang mirip dengan gambar aslinya dengan mengambil **k** yang jauh lebih kecil dari jumlah total *singular value* karena kebanyakan informasi disimpan di *singular values* awal karena singular values terurut mengecil. Nilai **k** juga berkaitan dengan rank matriks karena banyaknya *singular value* yang diambil dalam matriks S adalah *rank* dari matriks hasil, jadi dalam kata lain **k** juga merupakan rank dari matriks hasil. Maka itu matriks hasil rekonstruksi dari SVD akan berupa informasi dari gambar yang terkompresi dengan ukuran yang lebih kecil dibanding gambar awal.

Buatlah program kompresi gambar dengan memanfaatkan algoritma SVD dalam bentuk website lokal sederhana. Spesifikasi website adalah sebagai berikut:

1. Website mampu menerima file gambar beserta input tingkat kompresi gambar (dibebaskan formatnya).

2. Website mampu menampilkan gambar input, output, runtime algoritma, dan persentase hasil kompresi gambar (perubahan jumlah pixel gambar).
3. File output hasil kompresi dapat diunduh melalui website.
4. Kompresi gambar tetap mempertahankan warna dari gambar asli.
5. (Bonus) Kompresi gambar tetap mempertahankan transparansi dari gambar asli, misal untuk gambar png dengan background transparan.
6. Bahasa pemrograman yang boleh digunakan adalah Python, Javascript, dan Go.
7. Penggunaan framework untuk back end dan front end website dibebaskan. Contoh framework website yang bisa dipakai adalah Flask, Django, React, Vue, dan Svelte.
8. Kalian dapat menambahkan fitur fungsional lain yang menunjang program yang anda buat (unsur kreativitas diperbolehkan/dianjurkan).
9. Program harus modular dan mengandung komentar yang jelas.
10. Diperbolehkan menggunakan library pengolahan citra seperti OpenCV2, PIL, atau image dari Go.
11. Dilarang menggunakan library perhitungan SVD dan library pengolahan eigen yang sudah jadi.

BAB II
TEORI SINGKAT

A. Perkalian Matriks

Perkalian matriks merupakan salah satu operasi dua buah matriks yang menghasilkan sebuah matriks. Matriks A berukuran $p\ x\ q$ dapat dikalikan dengan matriks B berukuran $r\ x\ s$ jika $q = r$, dan akan menghasilkan matriks C dengan ukuran $p\ x\ s$. Misal $q = r = n$, maka

$$C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Perkalian matriks tidak bersifat komutatif. Operasi perkalian matriks sangat berperan dalam berbagai penerapan matriks untuk pemecahan masalah, diantaranya adalah mencari solusi persamaan linear (dengan berbagai macam metode matriks), transformasi linear, dan mendapatkan matriks SVD.

B. Nilai Eigen dan Vektor Eigen

Nilai eigen(λ) menyatakan nilai karakteristik dari sebuah matriks yang berukuran n x n. Salah satu metode untuk mendapatkan nilai eigen matriks A adalah dengan mencari akar-akar dari persamaan karakteirstik det(λI – A) = 0. Sementara itu, vektor eigen (x)adalah vektor yang berkoresponden dengan dengan λ dimana $Ax = \lambda x$. Cara mendapatkan vektor eigen adalah sebagai berikut.

1. Cari λ terlebih dahulu.
2. Tentukan basis untuk ruang eigen dari setiap λ (solusi $Ax = 0$)
3. Vektor eigen dibentuk dari basis tersebut

Vektor eigen diaplikasikan ke berbagai bidang. Dalam bidang Informatika khususnya, vektor eigen digunakan pada grafika computer.

C. QR Iteration

QR iteration merupakan salah satu metode lain untuk mendapatkan nilai eigen dan vektor eigen. Pada dasarnya, *QR decomposition* memfaktorkan matriks A menjadi matriks Q yang merupakan matriks orthogonal dan matriks R yang merupakan matriks Segitiga atas.

$$\mathbf{A} \qquad \mathbf{Q} \qquad \mathbf{R}$$

$$\begin{bmatrix} | & | & | \\ \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \\ | & | & | \end{bmatrix} = \begin{bmatrix} | & | & | \\ \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} \mathbf{e}_1^T \cdot \mathbf{a}_1 & \mathbf{e}_1^T \cdot \mathbf{a}_2 & \mathbf{e}_1^T \cdot \mathbf{a}_3 \\ 0 & \mathbf{e}_2^T \cdot \mathbf{a}_2 & \mathbf{e}_2^T \cdot \mathbf{a}_3 \\ 0 & 0 & \mathbf{e}_3^T \cdot \mathbf{a}_3 \end{bmatrix}$$

Misal, matriks awal yang digunakan adalah A_0 yang akan didekomposisi menjadi Q_0 dan R_0. Selanjutnya, dibentuk matriks $A_1 = R_0 Q_0$. Perhatikan bahwa A_1 dan A_0 memiliki nilai eigen dan vektor eigen yang sama sebab

$$Q_0 = A_0 {R_0}^{-1} \text{ maka } A_1 = R_0 A_0 {R_0}^{-1}$$

Langkah ini diiterasi beberapa kali hingga terbentuk A_n yang memiliki nilai sangat kecil (ε) pada elemen selain diagonal utama, dan nilai pada diagonal utamanya akan sangat dekat pada nilai Eigen yang kita cari.

D. Matriks SVD

Dekomposisi matriks adalah memfaktorkan sebuah matriks menjadi hasil kali dari sejumlah matriks lain. Terdapat beberapa metode untuk mendkomposisi matriks, salah satunya adalah Singular Value Decomposition (SVD). SVD dapat digunakan untuk memfaktorkan matriks non-bujursangkar berukuran m x n yang tidak memiliki nilai eigen. Misal A berukuran $m \; x \; n$, maka

$$A = U \Sigma V T$$

- U = Singular kiri berupa matriks ortogonal $m \; x \; m$, diperoleh dengan menentukan vektor eigen dari AA^T yang di normalisasi.
- $\sum$ = matriks berukuran m x n yang elemen-elemen diagonal utamanya adalah nilai-nilai singular (σ) dari matriks A dan elemen-elemen lainnya 0. Dimana $\sigma = \sqrt{\lambda}$, σ terurut dari yang paling besar.
- V = Singular kana berupa matriks ortogonal $n \; x \; n$, diperoleh dengan menentukan vektor eigen dari $A^T A$ yang di normalisasi.

SVD dapat diaplikasikan pada Kompresi gambar dan video, Image processing, machine Learning, dan sebagainya.

BAB III
IMPLEMENTASI

A. Implementasi Algoritma Kompresi Gambar

1. Vektor dan Nilai Eigen dengan QR Iteration

Untuk melakukan kompresi pada gambar, matriks-matriks yang menampung warna gambar perlu didekomposisi terlebih dahulu menjadi matriks SVD yang didapat dari vektor dan nilai eigen. Untuk mendapatkan nilai dan vektor eigen ini, kami menggunakan metode QR Iteration. Kami memanfaatkan fungsi qr yang terdapat pada Numpy Library Python. Fungsi qr tersebut akan mengembalikan dua buah matriks, yaitu matriks q dan r.

Setelah matriks didekomposisi menjadi matriks q dan r, matriks q akan dikalikan dengan matriks q sebelumnya ($q_t = q_1 * q_2 * ... * q_n$) sehingga setelah dilakukan iterasi sebanyak n kali akan menghasilkan matriks vektor eigen. Sedangkan untuk nilai eigen akan didapatkan dengan mengalikan matriks r dan q terakhir pada iterasi. Misal, matriks awal ($A = Q_0R_0$), matriks selanjutnya $A_1 = R_0Q_0 = Q_1R_1$, sehingga $A_n = R_{n-1}Q_{n-1}$. Implementasi dari vektor dan nilai eigen dengan QR Iteration ini terdapat pada fungsi getLambdaUV pada SVD.py dengan algoritma sebagai berikut:

```
def getLambdaUV(m):   # m sudah harus matriks simetris
    n = len(m)
    mVector = np.identity(n)
    for _ in range(0, 1000):
        q, r = qr(m)
        mVector = np.matmul(mVector, q)
        m = np.matmul(r, q)
    lamda = np.diag(m, k=0)
    return lamda, mVector
```

2. Singular Value Decomposition

Untuk mendapatkan matriks SVD dari suatu gambar dapat menggunakan nilai dan vektor eigen seperti yang sudah dijelaskan pada teori singkat. Matriks U dan V pada matriks SVD dapat langsung didapatkan dari mVector yang dikembalikan oleh fungsi getLambdaUV sebelumnya. Namun, untuk matriks $\sum$, nilai-nilai eigen (lamda) yang dikembalikan oleh fungsi getLambdaUV perlu

diakarkan. Karena pada QR Iteration tidak menjamin semua nilai eigennya bernilai positif, sehingga diperlukan pembatasan nilai, nilai yang dipilih adalah 10^{-10}. Algoritma untuk mendapatkan matriks ∑ adalah sebagai berikut

```
def getSigma(m, lamda):
    new_lamda = lamda.copy()
    np.absolute(new_lamda, out=new_lamda)
    sigma = np.sqrt(new_lamda[:min(m.shape)])
    return np.diag(sigma)
```

Setelah mendapatkan matriks ∑, matriks SVD dapat dilakukan seperti yang telah dijabarkan pada teori singkat dengan algoritma sebagai berikut

```
def getSVD(m: np.ndarray) :
    height, width = m.shape
    min_s = min(height, width)
    if width >= height:
        lamda1, U = getLambdaUV(m @ m.T)
    elif width < height:
        lamda1, VT = getLambdaUV(m.T @ m)

    S = getSigma(m, lamda1)
    if width >= height:
        VT = np.linalg.inv(S) @ (U.T)[:height, :] @ m
    elif width < height:
        U = m @ (VT.T)[:, :width] @ np.linalg.inv(S)

    return U[:, :min_s], S, VT[:min_s, :]
```

3. Kompresi Gambar

Sebelum melakukan dekomposisi matriks gambar menjadi matriks SVD dan kompresi gambar, gambar perlu di pisah terlebih dahulu menjadi 1 sampai 4 matriks sesuai dengan mode gambar, apakah L, P, LA, RGB, RGBA, atau CMYK. Untuk kompresi gambar, kami menggunakan metode pengurangan jumlah singular values dari matriks ∑ yang digunakan. Jumlah singular values yang digunakan adalah

$$\frac{100 - tingkat\ kompresi}{100} \times jumlah_{singular\ values\ awal}$$

Algoritma untuk kompresi matriks ∑ adalah sebagai berikut:

```
def compress_matrix(matrix_svd, diff):
    rank = len(matrix_svd[1][0])
    subtr = round((diff / 100) * rank)
    new_rank = rank - subtr
    u, e, vt = matrix_svd
    new_svd = [u[:, :new_rank], e[:new_rank, :new_rank], vt[:new_rank, :]]
    return new_svd
```

Setelah sudah mendapatkan kompresi matriks untuk gambar yang diinginkan, dilakukan perkalian matriks $U * \sum_{kompresi} * V^T$. Setelah itu, dilakukan kompresi matriks untuk semua matriks gambar yang telah dipisah menjadi 1 sampai 4 matriks. Matriks-matriks yang sudah dikompresi kemudian disatukan kembali dan dijadikan menjadi gambar lagi dengan keseluruhan algoritma sebagai berikut:

```python
def compress_image(b64Img: str, ratio: int):
    image = b64decode(b64Img)
    image = Image.open(BytesIO(image))
    original_mode = image.mode
    if original_mode == "P":
        image = image.convert("RGBA")
        mode = image.mode
    elif original_mode == "CMYK":
        image = image.convert("RGB")
        mode = image.mode
    else:
        mode = original_mode

    buffer = img_to_matrix(image, mode)
    new_buffer = []

    if mode == "RGB" or mode == "L":
        img_type = "JPEG"
        for color in buffer:
            u, s, vt = getSVD(color)
            new_color = compress_matrix([u, s, vt], ratio)
            new_color = svd_to_matrix(new_color)
            clip(new_color, 0, 255, out=new_color)
            new_buffer.append(new_color)
        img = matrix_to_img(new_buffer, original_mode)
```

```python
    elif mode == "RGBA" or mode == "LA":
        img_type = "PNG"
        for color in buffer[:-1]:
            u, s, vt = getSVD(color)
            new_color = compress_matrix([u, s, vt], ratio)
            new_color = svd_to_matrix(new_color)
            clip(new_color, 0, 255, out=new_color)
            new_buffer.append(new_color)
        new_buffer = mask_png(new_buffer, buffer[:-1])
        new_buffer.append(buffer[-1])
        img = matrix_to_img(new_buffer, original_mode)

    image_buffer = BytesIO()
    img.save(image_buffer, format=img_type)
    img_str = b64encode(image_buffer.getvalue())
    return img_str
```

B. Implementasi Pengembangan Website

1. Alur Kerja Web

- Image yang diupload di-encode ke base64 string (Frontend)
- Request api untuk compress dengan base64 string dan ratio sebagai body (Frontend)
- Decode base64 string ke image (Backend)

- Kompresi image dengan SVD (Backend)
- Image hasi kompresi di-encode ke base64 lagi sebagai respons ke frontend (Frontend)
- Antarmuka website menampilakan base64 string sebagai hasil gambar setelah dikompres (Frontend)

```
methods: {
  uploadImage: function (ev) {
    let input = ev.target;
    if (input.files && input.files[0]) {
      this.imgName = input.files[0].name;
      let reader = new FileReader();
      reader.onload = (e) => {
        this.imgData = e.target.result;
      }
      reader.readAsDataURL(input.files[0]);
    }
  },
  compress() {
    this.isLoading = true;
    axios.post("http://localhost:5000/api/compress", {image: this.imgData, ratio: this.rate})
        .then(res => {
          this.isLoading = false;
          if (res.data.success) {
            this.imgResult = res.data.image;
            this.time = res.data.time;
            this.diff =  res.data.diff;
          } else {
            this.showModal = true;
            this.message = res.data.message;
          }
        })
        .catch(err => {
          console.log("Error: " + err.message);
        });
```

methods untuk upload dan compress image

```
    def post(self):
        image_body = str(request.json['image']).split(",")
        compress_ratio = int(request.json['ratio'])
        image_mime = image_body[0]
        image_base64 = image_body[1]

        t1 = time()
        try:
            img_str, diff = compress_image(image_base64, compress_ratio)
        except:
            img_str, diff = "err", 0
        t2 = time()

        if img_str == "err":
            return {
                'success': False,
                'message': "An error occurred!"
            }
        else:
            return {
                'success': True,
                'image': image_mime + "," + img_str.decode("utf-8"),
                'diff': diff,
                'time': round(t2 - t1, 2)
            }
```

fungsi untuk me-return status hasil kompresi

```
methods: {
  downloadImage() {
    let a = document.createElement("a");
    let originalName = this.imgName.split(".");
    let extension = originalName.pop();
    let newName = originalName.join('.') + " (compressed)." + extension;
    a.href = this.imgResult;
    a.download = newName
    a.click();
  }
}
```

methods untuk mendowload gambar hasil kompresi

2. Backend (Flask Python)

Flask merupakan suatu framework Bahasa pemograman python yang diperuntukkan kepada *web/software developer*.

```
from flask import Flask, render_template
from api import api_blueprint
from flask_cors import CORS

app = Flask(__name__,
            static_folder='../frontend/dist/static',
            template_folder='../frontend/dist'
            )
app.register_blueprint(api_blueprint)
cors = CORS(app, resources={r"/api/*": {"origins": "*"}})

@app.route('/')
def index():
    return render_template('index.html')

app.run(port=5000)
```

file server.py

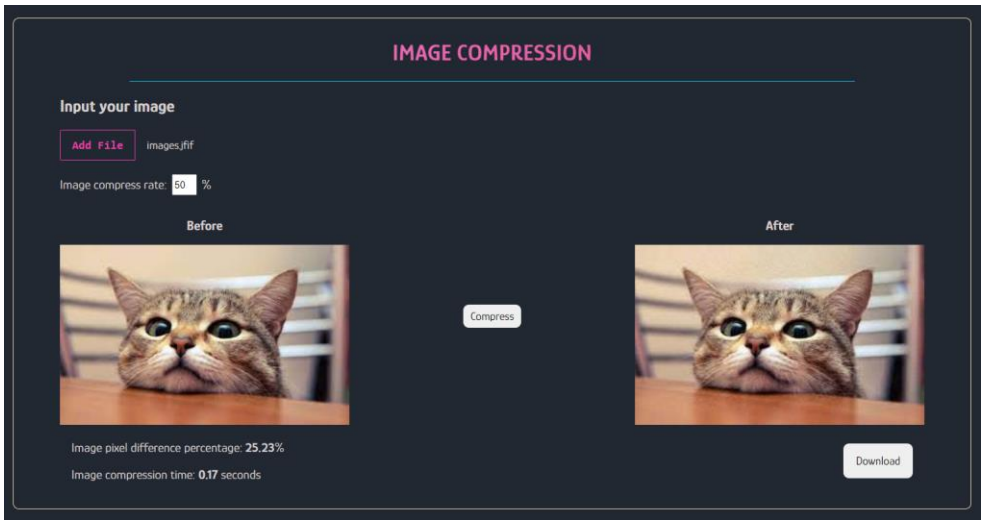
- route (/), menggunakan method=GET, berfungsi untuk menampilkan laman utama website dengan hasil build frontend
- route (/api/compress), menggunakan method=POST, dengan input data gambar awal dan output berupa gambar hasil kompres.

3. Frontend (Vue.js)

Vue.js merupakan salah satu framework JavaScript yang dikenal untuk membangun antarmuka dan *single-page application* (SPA). Vue.js dapat menerima data dari berbagai tools backend (misal PHP atau Python) dan kemudian membuat tampilan antarmuka.

- App.vue, view utama website yang memiliki beberapa komponen.

- Title.vue, berupa komponen header dari web
- Image.vue, berupa komponen untuk menampilkan gambar sebelum dan sesudah dikompres
- Output.vue, berupa komponen statistik meilputi persentase kompresi, waktu yang diperlukan, dan juga *button* untuk mengunduh gambar hasil kompresi

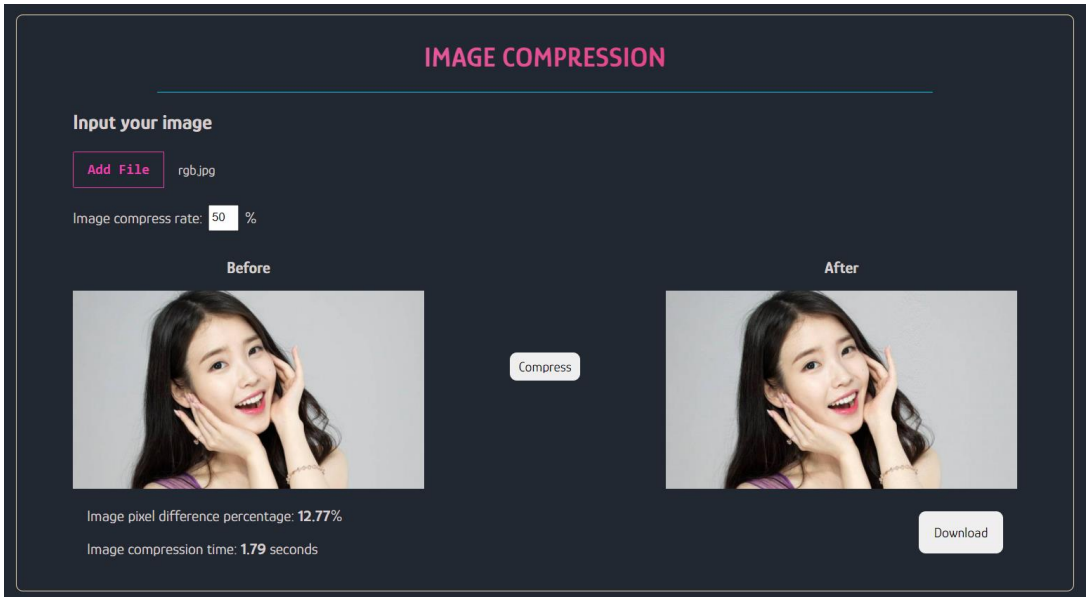


BAB IV

EKSPERIMEN

A. Gambar dengan Mode RGB

Eksperimen pertama merupakan percobaan kompresi gambar yang bermode RGB dengan dimensi 800 x 450 dalam satuan pixel dan dengan ukuran 76.9 KB. Proses kompresi gambar dilakukan dengan memisahkan *channel* R, G, dan B untuk dilakukan kompresi masing-masing, dan akhirnya akan disatukan kembali untuk membentuk gambar yang utuh. Pada percobaan ini, tingkat kompresi yang dicobakan adalah 50%, dengan hasil yang didapatkan seperti pada gambar di bawah ini.

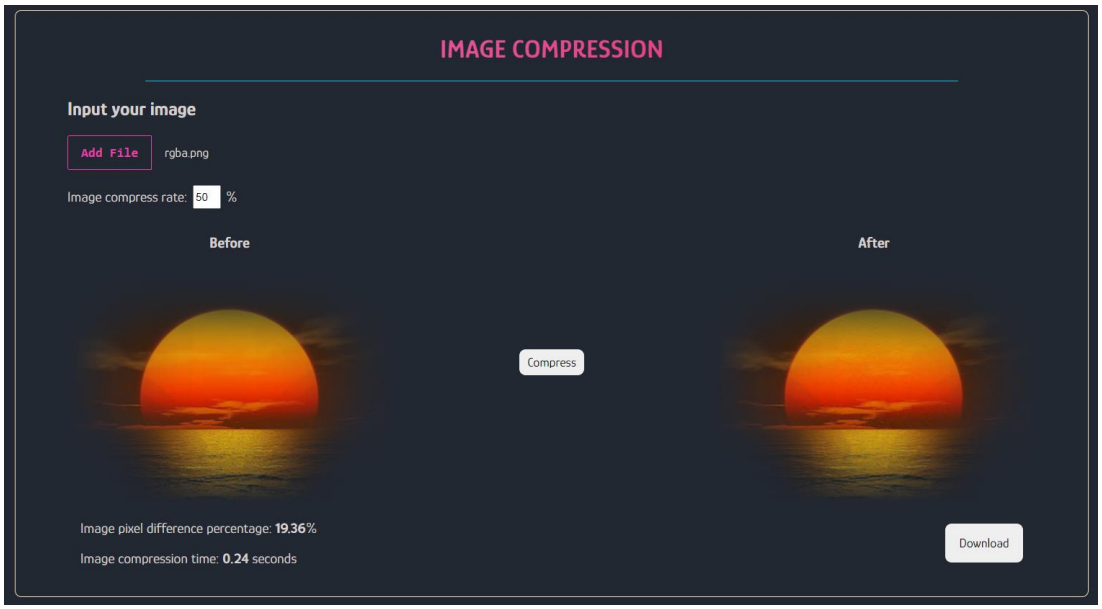


Pada gambar hasil kompresi, perbedaan dengan gambar awal tidak terlalu signifikan pada tingkat kompresi 50%. Dengan perhitungan perubahan *pixel*, didapatkan estimasi perbedaan 12.77%, yang secara kasat mata tidak berbeda dengan gambar awal. Proses ini memakan waktu sekitar 1.79 detik dengan ukuran akhir 34.3 KB, atau 44% dari ukuran awal. Berikut adalah hasil lengkapnya:

Tingkat kompresi (dalam persen)	Ukuran awal/Ukuran akhir (KB)	Perbedaan pixel (dalam persen)	Waktu kompresi (detik)
25	32.3/76.9	0.45	1.67
50	34.3/76.9	12.77	1.79
75	39.6/76.9	37.76	1.93
90	43.9/76.9	71.13	1.76

B. Gambar dengan Mode RGBA

Eksperimen kedua dilakukan pada gambar yang memiliki *channel alpha*, yaitu channel yang menentukan transparansi sebuah gambar. Gambar yang digunakan memiliki dimensi 300 x 221 *pixel* dengan ukuran 102KB. Proses kompresi gambar hanya dilakukan pada *channel* R, G, dan B, untuk mempertahankan transparansi dari gambar awal. Kemudian, *channel* R, G, dan B yang telah dikompresi akan di-*masking* dengan *channel alpha* agar bagian yang transparan pada gambar awal tidak rusak oleh *error* perhitungan SVD. Hasil dari kompresi dapat dilihat pada gambar di bawah ini.

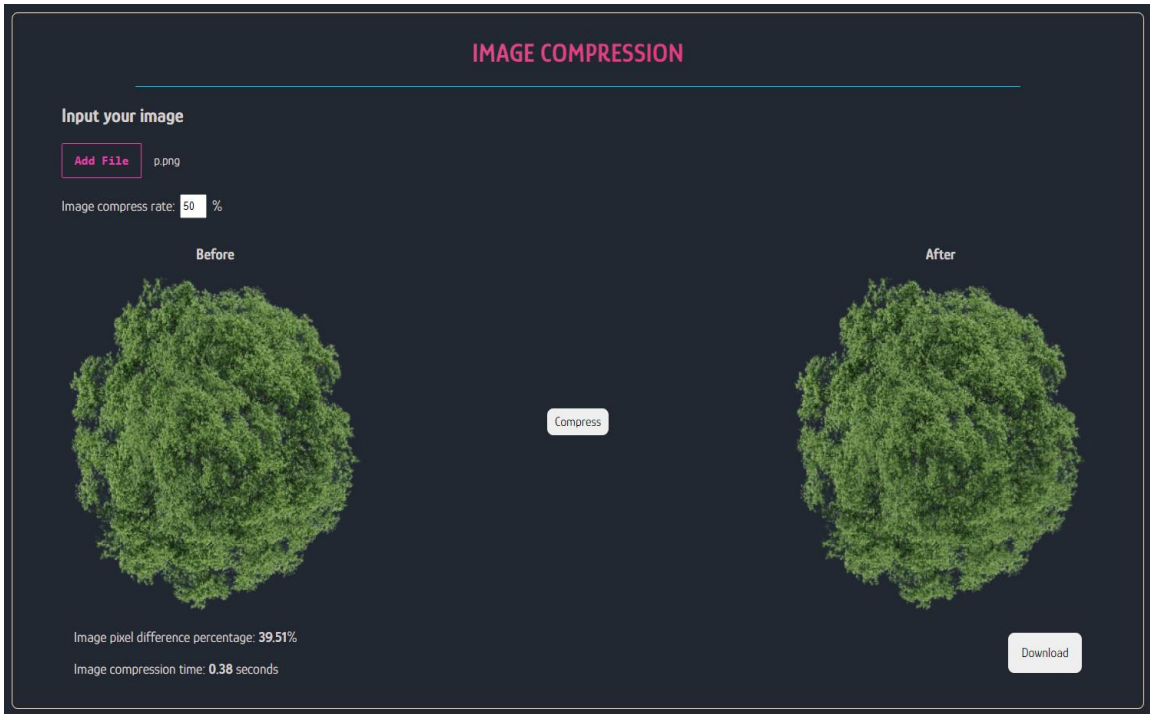


Pada gambar hasil kompresi, lagi-lagi tidak terdapat perbedaan yang signifikan pada tingkat kompresi gambar 50%. Perbedaan *pixel* yang didapat adalah 19.36% dengan waktu kompresi 0.24 detik. Ukuran gambar akhir adalah 110 KB, lebih besar 7.84% dari gambar awal, hal ini bisa terjadi karena gambar awal memiliki *pattern* warna yang teratur sehingga kompresi ke *storage* lebih efisien, sedangkan gambar hasil kompresi memiliki beberapa *error* yang menyebabkan *pattern* menjadi lebih tidak teratur, hal ini tergantung juga pada gambar yang diolah. Ketika tingkat kompresi pada gambar dinaikkan menjadi 90%, ukuran gambar turun menjadi 78KB, tetapi gambar menjadi lebih buram. Berikut adalah hasil lengkapnya:

Tingkat kompresi (dalam persen)	Ukuran awal/Ukuran akhir (KB)	Perbedaan pixel (dalam persen)	Waktu kompresi (detik)
25	110/102	2.61	0.29
50	110/102	19.36	0.24
75	101/102	38.82	0.32
90	77.4/102	54.73	0.31

C. Gambar dengan Mode P dan CMYK

Untuk gambar dengan mode ini, mode gambar akan dikonversi terlebih dahulu menjadi RGBA untuk mode P, dan RGB untuk mode CMYK. Setelah itu, kompresi akan dilakukan secara normal seperti yang telah dijelaskan pada bagian sebelumnya. Setelah proses kompresi selesai, gambar akan dikonversi kembali ke mode awalnya. Alasan dilakukannya konversi ini adalah untuk menghindari *error* yang tidak diinginkan, karena gambar dengan mode P adalah hasil palettised dari PNG. Hal ini juga terkait kompatibilitas library PIL yang digunakan yang tidak memiliki *full support* untuk kedua mode ini. Contoh yang akan ditampilkan pada gambar di bawah adalah gambar dengan mode P dengan dimensi 300 x 300 pixel.

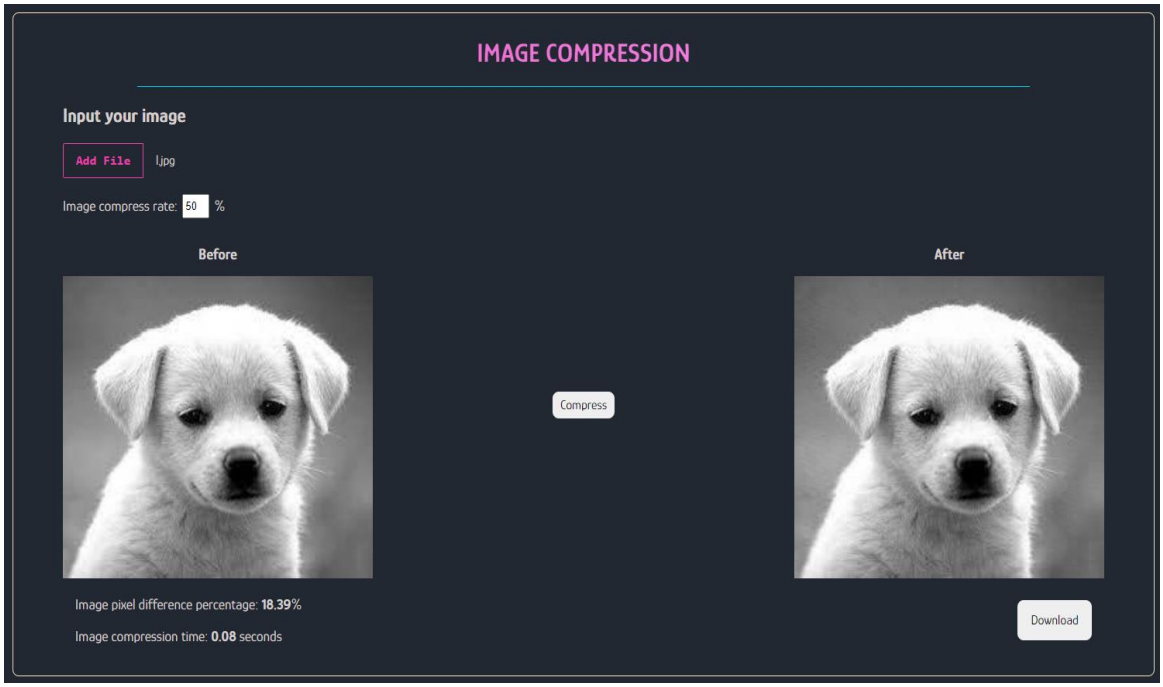


Hasil yang didapatkan seperti gambar di atas. Pada tingkat kompresi 50%, gambar yang didapat menjadi lebih buram, walaupun tidak terlalu jelas terlihat dengan perbedaan pixel sebesar 39.51% dengan waktu kompresi 0.38 detik. Pada tingkat kompresi ini, perbedaan ukuran gambar tidak terlalu signifikan dari 49.9 KB menjadi 49.7 KB atau berkurang sebesar 0.4%. Berikut adalah hasil lengkapnya:

Tingkat kompresi (dalam persen)	Ukuran awal/Ukuran akhir (KB)	Perbedaan pixel (dalam persen)	Waktu kompresi (detik)
25	49.4/49.9	6.27	0.34
50	49.7/49.9	39.51	0.38
75	48.9/49.9	56.11	0.47
90	44.2/49.9	61.13	0.36

D. Gambar dengan Mode L dan LA

Gambar yang memiliki mode L adalah gambar *grayscale* yang biasa dikenal dengan gambar hitam-putih. Gambar ini hanya memiliki satu *channel* yang menyimpan data luminance. Untuk gambar mode LA, sama seperti gambar *grayscale* dengan tambahan *channel alpha* yang mengatur transparansi gambar. Untuk gambar dengan mode L, hanya akan terjadi satu kali proses SVD sehingga waktu yang dibutuhkan relatif singkat. Untuk mode LA, prosesnya hampir sama dengan mode RGBA, dimana *channel* L akan melalui proses kompresi, dan pada akhirnya akan di-*masking* dengan *channel alpha*-nya. Berikut adalah percobaan pada gambar dengan mode L yang memilki dimensi 236 x 213 pixel dengan ukuran awal 6.60KB.



Pada tingkat kompresi 50%, tidak terdapat perbedaan yang signifikan pada gambar awal dan gambar akhir dengan perbedaan pixel sebesar 18.39% dan total waktu kompresi adalah 0.08 detik. Ukuran akhir gambar adalah 6.83 KB, atau lebih besar 3.48%. Berikut adalah hasil lengkapnya:

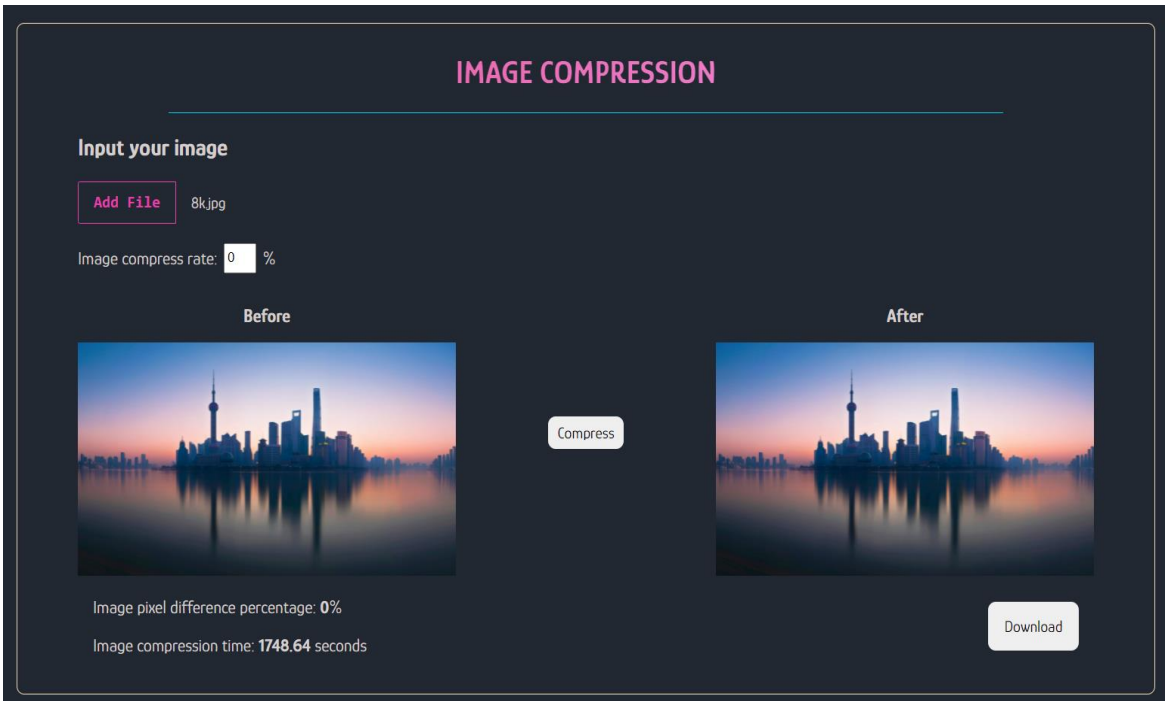
Tingkat kompresi (dalam persen)	Ukuran awal/Ukuran akhir (KB)	Perbedaan pixel (dalam persen)	Waktu kompresi (detik)
25	6.59/6.60	0.1	0.08
50	6.83/6/60	18.39	0.08
75	7.39/6.60	58.53	0.09
90	6.91/6.60	77.82	0.08

E. Gambar dengan Dimensi Besar

Eksperimen ini dilakukan untuk menguji algoritma yang kami buat, dengan input gambar yang berdimensi cukup besar. Tingkat kompresi disamakan sebesar 0% dan mode gambar RGB. Berikut adalah hasilnya pada beberapa dimensi:

Dimensi (pixel)	Ukuran awal/Ukuran akhir (MB)	Waktu kompresi (menit)
2K (1920 x 1280)	0.15/0.15	0.4
4K (3840 x 2160)	0.83/5.62	3
6K (5472 x 3648)	2.80/3.50	9
8K (8429 x 4832)	1.62/2.08	29

Pada hasil kompresi yang didapat, untuk keseluruhan percobaan, perbedaan pixel sebelum dan sesudah kompresi adalah 0%, dengan penurunan ukuran gambar. Ini berarti ukuran gambar semakin kecil tetapi tanpa mengurangi kualitas gambar, sehingga kompresi ini efektif, tergantung dengan gambar yang diolah. Data waktu kompresi di atas bisa berbeda pada setiap komputer, tergantung kecepatan *processing* masing-masing.



BAB V
KESIMPULAN, SARAN, DAN REFLEKSI

Dari pengerjaan Tugas Besar 2 Aljabar Linear dan Geometri ini, telah berhasil dibuat suatu program berbentuk website lokal sederhana yang dapat menerima file gambar, input tingkat kompresi gambar, dan melakukan kompresi terhadap gambar tersebut.

Akan tetapi, program belum bisa melakukan kompresi gambar secara maksimal dengan mempertahankan secara utuh warna dari gambar asli. Program masih mengubah warna gambar semakin jauh dari gambar aslinya ketika tingkat kompresi gambar ditingkatkan. Selain itu, program masih cukup lama dalam mengompresi gambar dengan dimensi yang cukup besar (di atas 1500 px) dan ukuran gambar yang cukup besar (di atas 3Mb) sehingga sangat diperlukan pengembangan lebih lanjut agar program ini bisa dapat digunakan dengan efektif, efisien, dan cepat dalam mengompresi gambar apapun. Saran algoritma yang dapat digunakan agar program menjadi lebih efektif dengan menggunakan metode *Divide and Conquer SVD*, yang jauh lebih cepat dan efisien, tetapi dengan implementasi dan perhitungan matematika yang cukup sulit.

REFERENSI

[1] *Chapter 4: The QR Algorithm*. https://people.inf.ethz.ch/arbenz/ewp/Lnotes/chapter4.pdf (diakses tanggal 1 November 2021)

[2] *Wikipedia: QR Algorithm*. https://en.wikipedia.org/wiki/QR_algorithm (diakses tanggal 1 November 2021)

[3] Munir, Rinaldi. *Singular Value Decomposition*. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-19b-Singular-value-decomposition.pdf (diakses tanggal 2 November 2021).