LAPORAN

TUGAS BESAR 2

IF2123 ALJABAR LINIER DAN GEOMETRI

Oleh:

Kelompok 19 “Mobilita”

Maharani Ayu Putri Irawan	/ 13520019
Suryanto	/ 13520059
Maria Khelli	/ 13520115

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Gambar adalah media yang sangat dibutuhkan pada dunia modern ini. Seringkali manusia berinteraksi dengan gambar baik untuk mendapatkan informasi maupun sebagai hiburan. Gambar digital banyak sekali dipertukarkan di dunia digital. Sayangnya, dalam proses transmisi dan penyimpanan gambar banyak ditemukan masalah karena ukuran file gambar digital yang cenderung besar.

Kompresi gambar merupakan suatu tipe kompresi data yang dapat 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 2. 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 mengecil. 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.

BAB II
TEORI SINGKAT

A. Perkalian Matriks

Perkalian matriks dibagi menjadi 2, yakni perkalian matriks dengan skalar dan perkalian matriks dengan matriks.

1. Perkalian Matriks dengan skalar

   Menurut Anton dan Rores (2013), jika A adalah sembarang matriks dan c adalah suatu skalar, hasil perkalian cA adalah matriks yang didapatkan dengan mengalikan setiap elemen matriks A dengan c. Matriks cA disebut sebagai kelipatan perkalian skalar matriks A. Dalam notasi matriks, jika $A = [a_{ij}]$, maka $(cA)_{ij} = c(A)_{ij} = ca_{ij}$.

2. Perkalian Matriks dengan Matriks

   Anton dan Rores (2013) menyatakan bahwa jika A adalah matriks $m \times r$ dan B adalah matriks $r \times n$, produk hasil perkalian AB adalah matriks $m \times n$ yang elemen-elemennya ditentukan berdasar aturan sebagai berikut. Untuk menemukan nilai elemen matriks AB pada baris ke-i dan kolom ke-j, ambil baris ke-i dari matriks A dan kolom ke-j dari matriks B. Kalikan elemen yang berkorespondensi dan jumlahkan hasil perkalian tersebut. Perkalian matriks dengan matriks mensyaratkan agar jumlah kolom matriks pertama sama dengan jumlah baris matriks kedua. Dalam notasi matriks, nilai elemen matriks $(AB)_{ij}$ dituliskan sebagai berikut.

$$(AB)_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ir}b_{rj}$$

B. Nilai Eigen dan Vektor Eigen

Jika A adalah matriks $n \times n$, vektor tak-nol x di R^n disebut sebagai vektor eigen dari A jika Ax (hasil perkalian matriks A dan x) adalah kelipatan skalar dari x, yakni $Ax = \lambda x$ untuk sebuah nilai skalar λ. Nilai skalar λ ini disebut nilai eigen dari A dan x disebut sebagai vektor eigen untuk sebuah nilai λ (Anton dan Rorres, 2013).

Untuk mencari nilai eigen dan vektor eigen, dengan memisalkan A sebagai matriks dan λ merupakan nilai eigen lalu menulis ulang persamaan $Ax = \lambda x$ menjadi $(\lambda I - A)x = 0$, pertama-tama perlu dihitung solusi sistem persamaan $(\lambda I - A) = 0$. Ekspansi terhadap persamaan ini membentuk polinom karakteristik berderajat n dengan n merupakan jumlah baris yang sama dengan jumlah kolom matriks A. Solusi tak nol dari polinom karakteristik ini merupakan nilai eigen.

Setelah nilai eigen didapatkan, basis ruang eigen dari nilai eigen yang bersangkutan didapat dengan mensubstitusikan masing-masing nilai eigen (λ) pada matriks persamaan $(\lambda I - A)x = 0$ lalu mencari nilai vektor x an menuliskannya dalam

representasi vektor, didapat basis ruang eigen untuk sebuah nilai eigen yang bersangkutan. Dengan menggunakan algoritma yang sama, didapatkan semua basis ruang eigen untuk setiap nilai eigen. Jika basis ruang eigen merupakan solusi nontrivial untuk persamaan $(\lambda I - A)x = 0$ dan juga bebas linier, maka basis ruang eigen tersebut juga merupakan vektor eigen.

C. Matriks Singular Value Decomposition (SVD)

Pemfaktoran matriks non-persegi dapat mengunakan metode SVD. Dua matriks persegi, misalnya A dan D dikatakan mirip secara ortogonal apabila dapat dicari sebuah matriks ortogonal P yang bisa dibalikkan sehingga memenuhi persamaan $P^T AP = D$. A dan D dikatakan matriks yang mirip secara ortogonal jika salah satu dari keduanya mirip secara ortogonal dengan yang lain. P merupakan matriks ortogonal dan D merupakan matriks diagonal. Dengan melakukan serangkaian operasi perkalian terhadap P di sebelah kiri ruas kiri dan P^T dari sebelah kanan ruas kiri, didapatkan persamaan $A = PDP^T$ dimana P adalah matriks ortogonal persegi dari vektor eigen A (kolom-kolomnya merupakan basis ruang eigen dari matriks A) dan D adalah matriks diagonal dengan elemen diagonal merupakan nilai eigen yang berkorespondensi terhadap vektor kolom P. Persamaan $A = PDP^T$ merupakan dekomposisi vektor eigen dari A.

Jika matriks persegi A tidak simetris, dapat dicari dekomposisi Hessenberg dengan persamaan yang sama seperti dekomposisi vektor eigen. Jika nilai eigen dari A merupakan bilangan real, maka matriks tersebut memiliki dekomposisi Schur yang persamaannya masih sama seperti dekomposisi vektor eigen. Bentuk dekomposisi ini dapat ditulis ulang menjadi $A = U\Sigma V^T$ dimana U dan V bersifat ortogonal namun tidak harus sama.

Untuk setiap matriks A berukuran $m \times n$, matriks $A^T A$ dapat didiagonalisasi secara ortogonal dan nilai eigen $A^T A$ bersifat tak negatif. Nilai singular dari matriks A merupakan akar kuadrat dari nilai eigen tak nol dari matriks $A^T A$. Nilai singular ini, jika disusun dalam diagonal utama matriks dengan dimensi yang sama dengan dimensi awal matriks A, disusun terurut mengecil membentuk matriks Σ. Matriks V dibentuk dengan menghitung vektor-vektor eigen dari nilai-nilai eigen matriks $A^T A$ yang telah dihitung sebelumnya, lalu dinormalisasi terhadap panjang vektor. Lalu, dapat dilakukan operasi transpose pada matriks V untuk membentuk matriks V^T. Matriks U dibentuk dengan pertama-tama mencari nilai eigen dari matriks AA^T lalu mencari vektor eigen yang berkorespondensi. Vektor eigen ini kemudian dinormalisasi terhadap panjang vektor lalu disusun menjadi matriks persegi. Dari dekomposisi ini, dapat dinyatakan bahwa $A = U\Sigma V^T$.

Dekomposisi dengan metode ini memiliki banyak kegunaan, antara lain pada kompresi, penyimpanan, dan transmisi informasi digital dan menjadi basis bagi banyak algorima komputasional untuk menyelesaikan sistem persamaan linier.

BAB III
IMPLEMENTASI PROGRAM

A. Kakas yang Digunakan

Kakas yang digunakan antara lain:

1. Bahasa pemrograman Python untuk kompresi gambar,
2. *Library* React.js dan Bootstrap untuk membangun tampilan *Front End*,
3. *Application Programming Interface (API)* Flask untuk *Back End*,
4. Visual Studio Code sebagai *Text Editor*, dan
5. *Library* NumPy, SciPy, Math, OpenCV, dan Matplotlib pada bahasa pemrograman Python.

B. Garis Besar Algoritma Kompresi

Pada tugas besar ini, kami menggunakan algoritma Implicit QL oleh Dubrulle, Martin, dan Wilkinson dalam buku yang dipublikasikan pada tahun 1971 berjudul *Handbook for Automatic Computation Linear Algebra Volume II*.

Algoritma QL (nonimplisit) dengan pergeseran (shift) pada matriks A didefinisikan sebagai

$$Q_s(A_s - k_s I) = L_s$$

$$A_{s+1} = L_s Q_s^T + k_s I$$

sehingga

$$A_{s+1} = Q_s A_s Q_s^T$$

dengan Q_s merupakan matriks ortogonal, L_s merupkan matriks segitiga bawah dan k_s adalah pergeseran yang ditentukan oleh upamatriks 2 x 2 dari A_s.

Pergeseran yang dilakukan untuk mempercepat konvergensi matriks sehingga menurunkan kompleksitas algoritma. Pemilihan pergeseran sebetulnya bisa saja diambil secara bebas karena pada akhirnya akan ditambahkan kembali kecuali jika pergeseran tersebut membuat komponen matriks menjadi nol (*cancellation*).

Algoritma ini cukup baik, tetapi algoritma nonimplisit dapat menyebabkan *loss* yang besar saat komputasi ketika nilai dalam matriks A berbeda jauh (membuat perhitungan tidak akurat), maka dipakailah algoritma QL implisit. Bedanya, algoritma implisit tidak melakukan pergeseran pada matriks A_s.

Menurut Vetterling, et al. (1986), kompleksitas algoritma ini pada matriks biasa adalah $O(n^3)$, pada matriks Hessenberg adalah $O(n^2)$, dan pada matriks tridiagonal adalah $O(n)$. Maka dari itu, sebelum melakukan algoritma ini, kami melakukan preproses pada matriks terlebih dahulu.

Untuk menghitung SVD, pada tahap pertama kami menghitung singular kanan V dan sigma Σ terlebih dahulu. Kedua nilai ini bisa didapatkan dari perkalian $A^T A$. Karakteristik matriks $A^T A$ adalah matriks ini berbentuk simetri.

Untuk mengubah $A^T A$ ke matriks tridiagonal dengan mempertahankan nilai eigen, kami menggunakan dekomposisi Hessenberg di mana $P^T(A^T A)P = T$ dengan T adalah matriks tridiagonal. Dekomposisi hessenberg ini dilakukan dengan bantuan transformasi Householder. Maka dari itu, didapatkanlah matriks T yang tridiagonal.

Dari matriks T tersebut bisa didapatkan nilai eigen, sedangkan vektor eigen bisa didapatkan dari akumulasi transformasi Householder Q yang telah dilakukan sebelumnya. Dalam tugas besar ini, kami menggunakan library SciPy sehingga langsung didapatkan T dan Q yang akan diterapkan pada QL Implisit.

QL implisit dilakukan dengan cara mencari elemen subdiagonal (diagonal di bawah diagonal utama) yang paling kecil. Menurut Wilkinson, et al., hal ini dilakukan untuk memecah matriks sehingga dapat membentuk matriks jumlah langsung (direct sum $\oplus$). Metode ini juga, menurutnya, dapat meningkatkan efisiensi komputasi.

Kemudian, pergeseran k_s dihitung dan diimplementasikan pada entri yang ingin dihilangkan. Terakhir, kita dapat menghitung nilai eigen dan mengakumulasikan vektor eigen dengan Givens *rotation* dan mengembalikan pergeseran yang dilakukan.

Setelah mendapatkan vektor eigen dan nilai eigen dari hasil aplikasi QL Implisit pada T dan Q, kami melakukan *sorting* terlebih dahulu baik pada nilai eigen maupun vektor eigen. Hal ini bertujuan untuk menyesuaikan dengan kriteria SVD, yaitu nilai eigen (atau akar eigen = sigma) diurut dari paling besar (kiri atas) ke paling kecil (kanan bawah). Nilai eigen / sigma berpasangan dengan vektor eigen, maka vektor eigen juga diurutkan.

Kemudian, vektor eigen digabung menjadi V dan nilai eigen diakar-kuadratkan sehingga menghasilkan sigma Σ. Dari V, kita dapat mencari u_i dengan rumus

$$\boldsymbol{u_i} = \frac{1}{\sigma_i} A\boldsymbol{v_i}$$

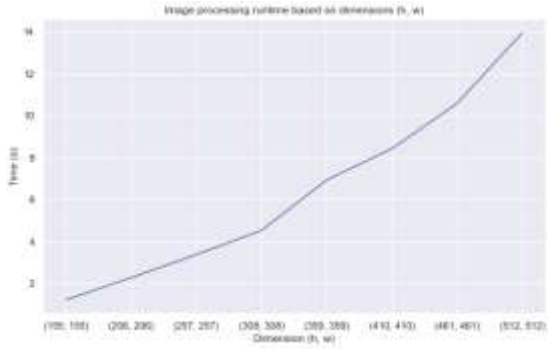
Sebagai akibatnya, matriks U hanya dapat memiliki dimensi maksimal (m, *rank*) dengan m adalah jumlah baris matriks A dan *rank* adalah jumlah nilai eigen tidak nol. Dengan membentuk

$$\boldsymbol{U} = [\boldsymbol{u_1}|\, \boldsymbol{u_2}|\, \ldots |\boldsymbol{u_n}]$$

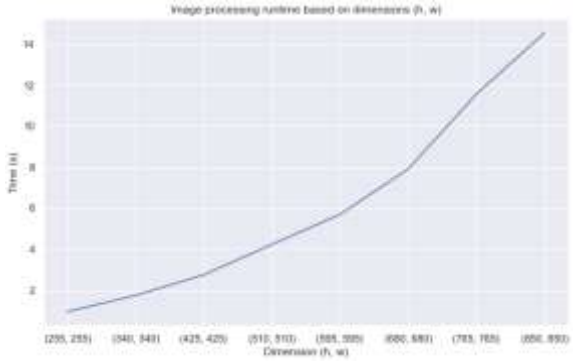
maka selesailah pencarian SVD dari A dalam bentuk U, Σ, dan V. Saat ingin melakukan komputasi, kita dapat tinggal melakukan transpose pada V.

Sebagai catatan tambahan, karena kami melakukan perhitungan U dari V, jumlah kolom V harus lebih banyak dari U. Maka dari itu, algoritma ini hanya bekerja jika baris ≤ kolom. Untuk mengatasinya, jika orientasinya landscape (baris ≥ kolom), kami melakukan transpose pada gambar awal dan pada tahap sebelum meng-export gambar (setelah diproses), kami melakukan transpose lagi pada gambar—sehingga menjadi seperti semula.

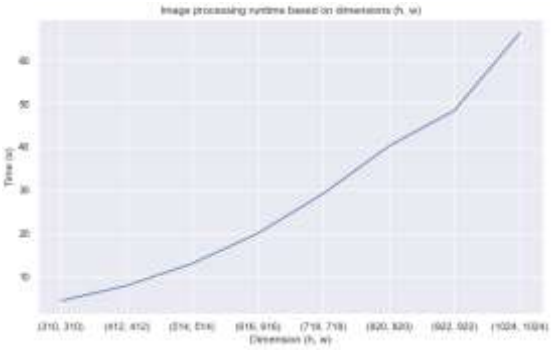
Jika dilihat, dapat diperkirakan bahwa algoritma ini berjalan dalam fungsi polinomial, meski secara teoretis SVD berjalan O(n). Hal ini mungkin disebabkan oleh proses lain yang dilakukan selain SVD.



Gambar 3.1 Plot tabel gambar tes “baboon.png” hitam putih berukuran 512 × 512



Gambar 3.2 Plot tabel gambar tes “lena.png” hitam putih berdimensi 850 × 850



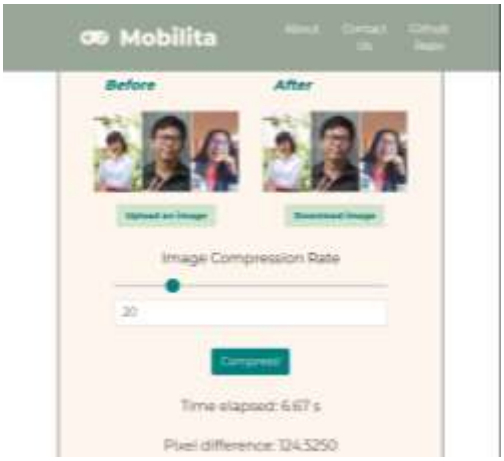
Gambar 3.3 Plot tabel gambar tes “logo_itb.png” warna berdimensi 1024 × 1024

BAB IV
EKSPERIMEN

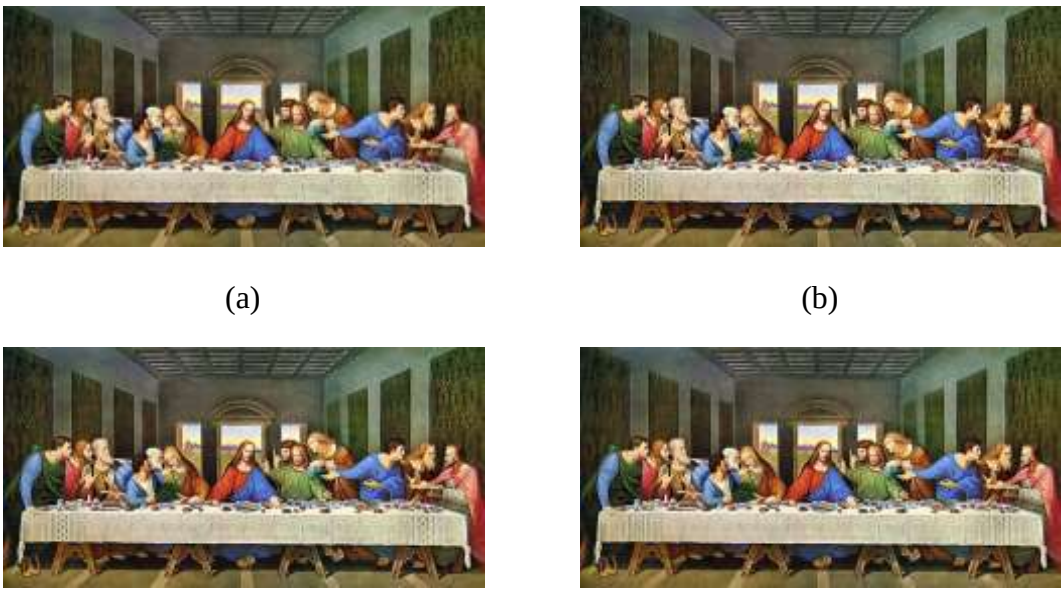
Gambar-gambar di bawah ini adalah tampilan dari website pengompresi gambar yang dibuat serta beberapa contoh pengunaannya secara nyata.

Gambar 4.1 Landing Page

Gambar 4.2 Tampilan upload gambar



Gambar 4.3 Tampilan hasil pemampatan citra



(a) (b)

(c) (d)

Gambar 4.4 (a) Citra Last Supper sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10%, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

Pada gambar 4.3 di atas, eksperimen dilakukan pada gambar dengan dimensi 800x400 pixels dengan format JPG. Gambar yang sebelumnya berukuran 96 KB dimampatkan dengan persentase 10%, 50%, dan 75%. Masing-masing proses memakan waktu yang relatif sama yaitu sekitar 5 detik dan menghasilkan citra baru dengan ukuran yang lebih kecil, yaitu secara berturut-turut 91 KB, 86 KB, dan 79 KB.

(a)

(b)

(c)

(d)

Gambar 4.5 (a) Citra Coco sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

Poster dari film Coco diatas memiliki dimensi 540x810 pixel dengan ukuran 204 KB pada format JPG. Ukuran dari citra berhasil dikecilkan dengan urutan pemampatan 10%, 50%, 75% secara berturut-turut menjadi 154 KB, 149 KB, dan 136 KB. Waktu yang dibutuhkan juga tidak memiliki perbedaan signifikan, yaitu 10 detik untuk pemampatan 10%, 9,5 untuk 50%, dan 8,7 detik untuk 75%.

(a)

(b)

(c)

(d)

Gambar 4.6 (a) Citra Boat sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10%, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

Citra dengan format JPG di atas memiliki dimensi 512x512 pixels. Sistem melakukan pemampatan dengan tingkat kompresi sebesar 10%, 50%, dan 75%, sehingga gambar yang semula berukuran 48 KB dimampatkan menjadi 43 KB, 42 KB, dan 38 KB.

(a)

(b)

(c) (d)

Gambar 4.7 (a) Citra Lena sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10%, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

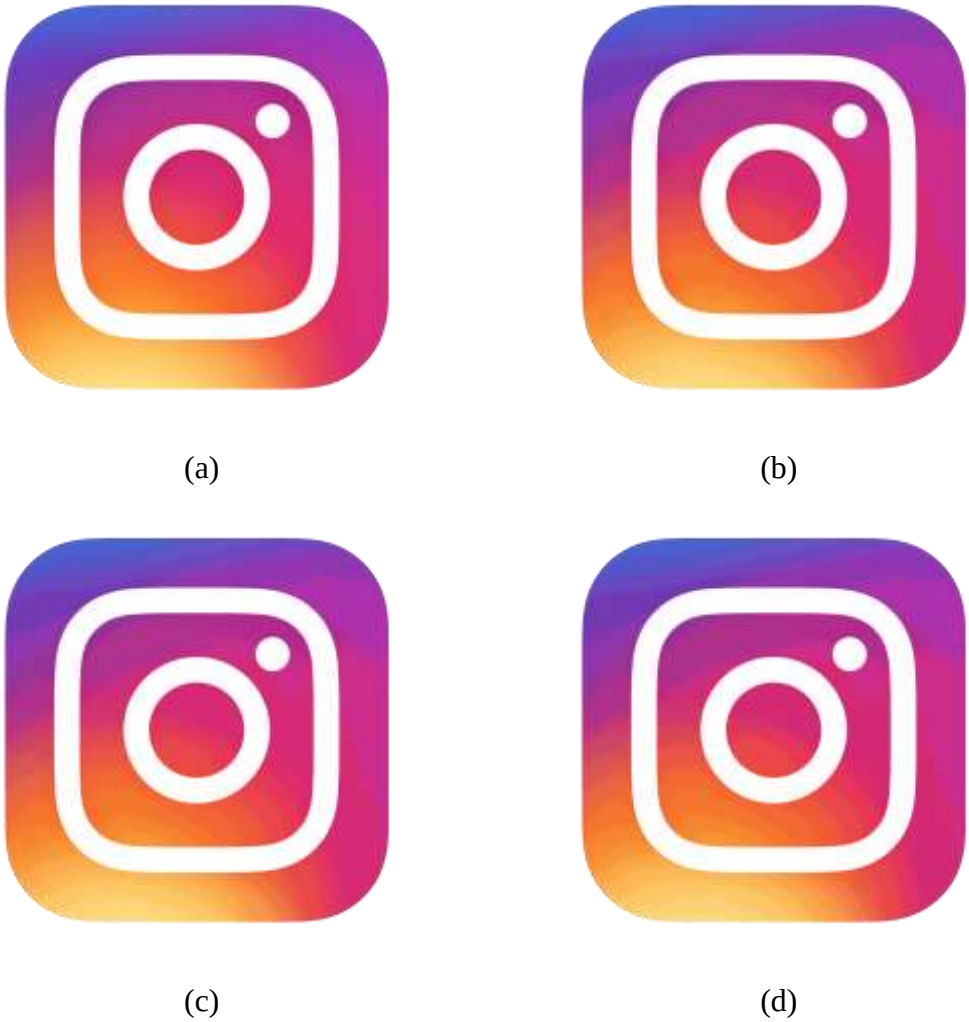
Gambar di atas hanya memiliki satu channel, yaitu channel L (hitam putih) tanpa channel *Alpha*. Gambar dengan dimensi 850x850 pixels ini berhasil dimampatkan oleh sistem dalam waktu rata-rata 9 detik. Citra diproses dengan ukuran awal 356 KB menjadi 295 KB, 285 KB, dan 265 KB untuk persen pemampatan secara berturut-turut 10%, 50%, dan 75%. Dapat dilihat bahwa citra yang dihasilkan dengan pemampatan 75% tidak mengalami penghilangan kualitas yang signifikan.

(a) (b)

(c) (d)

Gambar 4.8 (a) Citra Lena sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10%, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

Gambar Lena di atas memiliki 3 *channel* yaitu channel *Red, Green, dan Blue.* Pemampatan berhasil mengubah ukuran yang semulanya bernilai 72 KB menjadi lebih kecil, di antaranya 43 KB (pemampatan 10%), 45 KB (pemampatan 50%), dan 44 KB (pemampatan 75%).



(a) (b)

(c) (d)

Gambar 4.9 (a) Citra logo Instagram sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10%, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

Logo Instagram di atas merupakan citra dengan format PNG yang memiliki 4 channel, yaitu *Red, Green, Blue* dan *Alpha (*Transparansi). Waktu rasa-rata yang dibutuhkan program adalah sekitar 30 detik. Pada eksperimen kali ini, pemampatan sebesar 10% memiliki ukuran terkecil yaitu 95 KB, sedangkan untuk pemampatan 50% dan 75% masing-masing menghasilkan 109 Kb dan 107 KB.

(a) (b) (c) (d)

Gambar 4.10 (a) Citra Doge sebelum dimampatkan, (b) Citra yang dimampatkan sebesar 10%, (c) Citra yang dimampatkan sebesar 50%, (d) Citra yang dimampatkan sebesar 75%

Gambar Doge di atas juga memiliki channel *alpha* yang berfungsi untuk transparansi background. Pemampatan dilakukan pada tiga nilai uji, yaitu pemampatan sebesar 7%, 50% dan 75% dengan ukuran akhir secara berturut-turut adalah 49 KB, 54 KB, dan 55 KB. Pemampatan berhasil mengurangi ukuran dari gambar, tetapi berbanding lurus dengan persentasenya. Hal ini hanya ditemui pada kasus gambar dengan format PNG.

BAB V
KESIMPULAN

A. Kesimpulan

Sejauh ini, terdapat berbagai algoritma untuk mencari nilai eigen dan vektor eigen dari suatu matriks. Di antaranya yang pernah dicoba merupakan *householder transformation, power iteration, rayleigh quotient iteration, QR algorithm,* QL Implisit dan *Jacobi eigenvalue algorithm*.

Algoritma terbaik yang berhasil kami implementasikan hasil terbaik dalam waktu singkat adalah *QL Implisit*. Di sisi lain, algoritma yang membutuhkan waktu paling lama adalah *power iteration*. Oleh karena itu, pada tugas besar ini digunakan algoritma QL Implisit untuk mendapatkan nilai-nilai eigen dan vektor eigen. Meski demikian, bisa jadi kami salah mengimplementasikan sehingga tidak dapat memanfaatkan algoritma semaksimal mungkin.

Penggunaan *library* seperti NumPy, React, dan Bootstrap dapat mempermudah implementasi kode karena terdapat operasi-operasi pada matriks yang telah diimplementasikan sehingga mempersingkat waktu yang diperlukan untuk mengimplementasikan program.

B. Saran

Disarankan untuk matriks berukuran besar agar tidak menggunakan pencarian nilai eigen dan vektor eigen menggunakan metode *power iteration* (berbeda dengan *simultaneous power iteration*). Pengembangan lebih lanjut dari proyek ini dapat dilakukan dengan mencari berbagai algoritma lain yang dapat semakin mempercepat proses kompresi gambar. Pemampatan citra berformat PNG juga dapat dikembangkan agar dapat mencapai hasil yang diinginkan.

C. Refleksi

Diperlukan waktu eksplorasi yang cukup lama untuk menemukan algoritma SVD yang dapat diimplementasikan dan dijalankan dalam waktu yang relatif pendek. Pemampatan pada gambar berformat PNG juga belum dapat berjalan dengan sempurna. Hal ini bisa dilihat dari kasus pemampatan dengan persentase 10% yang menghasilkan ukuran lebih kecil dibandingkan pemampatan dengan persentase 75%.

DAFTAR PUSTAKA

Anton, Howard dan Rores, Chris. 2013. *Elementary Linear Algebra* (Edisi 11). Kanada: Wiley.

Lambers, Jim. "MAT 610 Summer Session 2009-10 Lecture 15 Notes". Diakses pada 19/11/2021 melalui: https://www.math.usm.edu/lambers/mat610/sum10/ex3.pdf.

Munir, Rinaldi. "Nilai Eigen dan Vektor Eigen" (Bagian 1). Diakses pada 27/10/2021 melalui: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-18-Nilai-Eigen-dan-Vektor-Eigen-Bagian1.pdf.

Munir, Rinaldi. "Nilai Eigen dan Vektor Eigen" (Bagian 2). Diakses pada 27/10/2021 melalui: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian2.pdf.

Munir, Rinaldi. "Singular Value Decomposition (SVD)". Diakses pada 27/10/2021 melalui: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-19b-Singular-value-decomposition.pdf

Vetterling, et al. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press.

Wilkinson, et al. 1971. *Handbook for Automatic Computation*. Berlin: Springer-Verlag.