

LAPORAN TUGAS BESAR

Tugas Besar 2 IF2123 Aljabar Linier dan Geometri Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar Semester I Tahun 2023/2024



KELOMPOK Oppeneigen

Ellijah Darrellshane Suryanegara (13522097)

Muhammad Dava Fathurrahman (13522114)

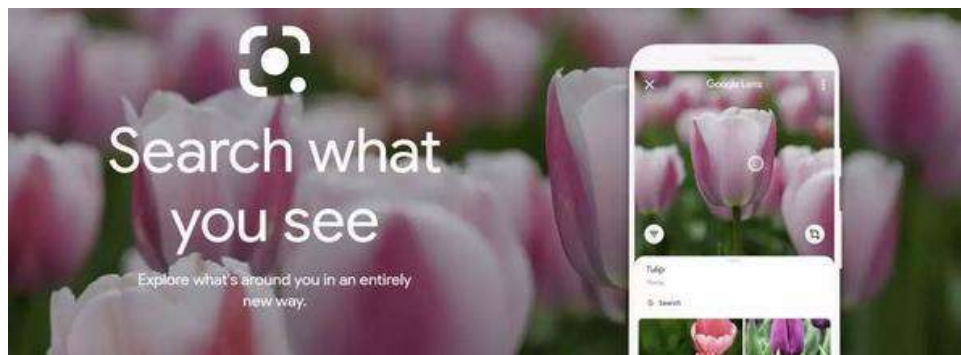
DAFTAR ISI

DAFTAR ISI.....	2
BAB I	
DESKRIPSI MASALAH.....	4
2.1 CONTENT-BASED INFORMATION RETRIEVAL (CBIR).....	5
2.1.1. CBIR dengan parameter warna.....	6
2.1.2. CBIR dengan parameter tekstur.....	8
2.2 Website Development.....	12
BAB III	
ANALISIS PEMECAHAN MASALAH.....	14
3.1 Langkah Penyelesaian Masalah.....	14
3.2. Elemen Aljabar Geometri.....	14
BAB IV	
IMPLEMENTASI DAN UJI COBA.....	16
4.1. Spesifikasi dan Tata Cara Penggunaan Program.....	16
4.2. Implementasi Program.....	18
4.2.1. Texture.....	18
a) preprocess_image.....	18
b) create_framework_matrix.....	18
c) contrast.....	19
d) homogeneity.....	19
e) entropy.....	20
f) dissimilarity.....	20
g) asm_val.....	21
h) correlation.....	21
i) inverse_contrast.....	22
j) distortion.....	23
k) save_csv.....	23
l) load_csv.....	24
m) cosine_similarity.....	25
n) search.....	26
4.2.2. Color.....	27
a) loadImages.....	27
b) rgb_to_hsv.....	27
c) hsv_quantify.....	28
d) split_image.....	29
e) build_vector.....	30

f) cosine_similarity_color.....	31
g) save_color_csv.....	31
h) load_color_csv.....	32
i) search_color.....	32
4.2.3. Website.....	33
4.3. Uji Coba.....	34
BAB V	
KESIMPULAN.....	41
REFLEKSI.....	42
DAFTAR PUSTAKA.....	43

BAB I DESKRIPSI MASALAH

Dalam era digital, jumlah gambar yang dihasilkan dan disimpan semakin meningkat dengan pesat, baik dalam konteks pribadi maupun profesional. Peningkatan ini mencakup berbagai jenis gambar, mulai dari foto pribadi, gambar medis, ilustrasi ilmiah, hingga gambar komersial. Terlepas dari keragaman sumber dan jenis gambar ini, sistem temu balik gambar (*image retrieval system*) menjadi sangat relevan dan penting dalam menghadapi tantangan ini. Dengan bantuan sistem temu balik gambar, pengguna dapat dengan mudah mencari, mengakses, dan mengelola koleksi gambar mereka. Sistem ini memungkinkan pengguna untuk menjelajahi informasi visual yang tersimpan di berbagai platform, baik itu dalam bentuk pencarian gambar pribadi, analisis gambar medis untuk diagnosis, pencarian ilustrasi ilmiah, hingga pencarian produk berdasarkan gambar komersial. Salah satu contoh penerapan sistem temu balik gambar yang mungkin kalian tahu adalah Google Lens.



Gambar 1. Contoh penerapan *information retrieval system* (Google Lens)

Di dalam Tugas Besar 2 ini, Anda diminta untuk mengimplementasikan sistem temu balik gambar yang sudah dijelaskan sebelumnya dengan memanfaatkan Aljabar Vektor dalam bentuk sebuah *website*, dimana hal ini merupakan pendekatan yang penting dalam dunia pemrosesan data dan pencarian informasi. Dalam konteks ini, aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (*Content-Based Image Retrieval* atau CBIR), di mana sistem temu balik gambar bekerja dengan mengidentifikasi gambar berdasarkan konten visualnya, seperti warna dan tekstur.

BAB II

TEORI SINGKAT

2.1 *CONTENT-BASED INFORMATION RETRIEVAL (CBIR)*

Content-Based Image Retrieval (CBIR) merujuk pada pencarian gambar secara langsung di mana gambar yang memiliki karakteristik khusus atau konten tertentu akan dicari dalam kumpulan data gambar. Konsep inti dari CBIR adalah menganalisis informasi dalam gambar berdasarkan ciri-ciri dasar dari gambar tersebut, seperti warna, tekstur, bentuk, hubungan objek dalam ruang, dan sebagainya, serta membuat vektor fitur dari gambar sebagai penandaannya. Pendekatan pengambilan fokus pada pencarian yang serupa dan didasarkan pada ciri-ciri multidimensi dalam sebuah gambar.

Ciri merupakan suatu tanda yang khas, yang membedakan antara satu gambar dengan gambar yang lain. Pada dasarnya, suatu gambar memiliki tiga ciri dasar, yakni:

a. **Warna**

Model warna RGB (red, green, blue) mendeskripsikan warna sebagai kombinasi positif dari 3 warna, yaitu merah, hijau, dan biru. Fitur warna merupakan salah satu dari ciri gambar yang paling mudah dipahami dan paling mencolok. Ini juga merupakan aspek yang krusial dalam persepsi visual. Jika dibandingkan dengan fitur gambar lainnya seperti tekstur dan bentuk, fitur warna cenderung lebih stabil dan memiliki kekuatan yang signifikan. Sifatnya tidak terlalu peka terhadap rotasi, translasi, atau perubahan skala. Salah satu metode yang paling umum digunakan untuk mengekstrak fitur warna adalah melalui histogram warna.

b. **Bentuk**

Ciri bentuk suatu gambar dapat ditentukan oleh tepi (sketsa), atau besaran momen dari suatu gambar. Pemakaian besaran momen pada ciri bentuk ini banyak digunakan dengan memanfaatkan

nilai-nilai transformasi fourier dari gambar. Proses yang dapat digunakan untuk menentukan ciri bentuk adalah deteksi tepi, threshold, segmentasi dan perhitungan momen seperti mean, median dan standar deviasi dari setiap lokal gambar

c. Tekstur

Tekstur merupakan karakteristik yang sangat penting untuk analisis permukaan berbagai jenis objek. Istilah tekstur secara umum mengacu kepada pengulangan elemen–elemen dasar tekstur yang disebut texel yang tersebar secara periodik, kuasi periodik atau secara acak. Dari berbagai penelitian tentang penglihatan manusia diperoleh kesimpulan bahwa analisa ruang frekuensi atau multiskala lebih

2.1.1. CBIR dengan parameter warna

Pada CBIR kali ini akan dibandingkan *input* dari sebuah *image* dengan *image* yang dimiliki oleh dataset, hal ini dilakukan dengan cara mengubah *image* yang berbentuk RGB menjadi sebuah metode histogram warna yang lebih umum.

Ruang warna RGB, meskipun merupakan hal mendasar dalam pencitraan digital, sering kali gagal memenuhi kebutuhan persepsi dalam pengambilan gambar. Salah satu alternatif umum adalah ruang warna HSV (Hue, Saturation, Value), yang menawarkan representasi warna lebih intuitif berdasarkan persepsi manusia. Dalam konteks pengambilan gambar, ruang HSV seringkali digunakan karena kemampuannya dalam memisahkan informasi warna dengan lebih baik. Pembagiannya menjadi komponen corak warna, saturasi, dan nilai memungkinkan interpretasi atribut warna yang lebih mudah. Peralihan ke HSV ini terbukti bermanfaat karena lebih selaras dengan cara manusia memandang dan mengkategorikan warna, sehingga meningkatkan efisiensi dan akurasi sistem pengambilan gambar.

Histogram warna adalah frekuensi dari berbagai warna yang ada pada ruang warna tertentu hal ini dilakukan untuk melakukan pendistribusian warna dari *image*. Histogram warna tidak bisa mendeteksi sebuah objek yang spesifik yang terdapat pada *image* dan tidak bisa mendeskripsikan posisi dari warna yang didistribusikan.

Pembentukan ruang warna perlu dilakukan dalam rangka pembagian nilai citra menjadi beberapa *range* yang lebih kecil. Hal itu dilakukan untuk membuat sebuah histogram warna yang setiap interval tiap *range* dianggap sebagai *bin*. Histogram warna dapat dihitung dengan menghitung piksel yang menyatakan nilai warna pada setiap interval. Fitur warna mencakup histogram warna global dan histogram warna blok.

Dengan demikian, perlu dilakukan konversi warna dari RGB ke HSV dengan prosedur berikut.

1. Menormalisasi nilai RGB dengan mengubah nilai *range* [0, 255] menjadi [0, 1]

$$R' = \frac{R}{255}$$

$$G' = \frac{G}{255}$$

$$B' = \frac{B}{255}$$

2. Mencari C_{max} , C_{min} , dan Δ

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

3. Menggunakan hasil perhitungan di atas untuk menentukan nilai HSV

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right) & , C' \max = R' \\ 60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right) & , C' \max = G' \\ 60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right) & , C' \max = B' \end{cases}$$

$$S = \begin{cases} 0 & , C_{\max} = 0 \\ \frac{\Delta}{C_{\max}} & , C_{\max} \neq 0 \end{cases}$$

$$V = C_{\max}$$

Setelah mendapatkan nilai HSV, perbandingan antara *image* dari input dengan dataset dapat ditentukan dengan menggunakan *cosine similarity*

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

A dan B adalah vektor dan n adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*.

Untuk melakukan pencarian histogram, blok *image* dibagi menjadi 4×4 blok untuk meminimalisasi waktu pencarian. Selain itu, blok yang berada di tengah memiliki bobot vektor lebih besar daripada blok-blok yang berada di pinggir gambar.

2.1.2. CBIR dengan parameter tekstur

Penelitian pengenalan pola pada gambar dengan menggunakan Content-Based Image Retrieval (CBIR) sering kali melibatkan analisis tekstur menggunakan suatu matriks yang disebut co-occurrence matrix.

Pemilihan matriks ini disebabkan oleh kemampuannya untuk melakukan pemrosesan dengan efisien dan cepat, menghasilkan vektor dengan ukuran yang lebih kecil. Misalkan kita memiliki gambar I dengan dimensi $n \times m$ piksel dan suatu parameter offset $(\Delta x, \Delta y)$. Matriks co-occurrence dapat dirumuskan sebagai berikut:

$$C_{\Delta x, \Delta y}(i, j) = \sum_{p=1}^n \sum_{q=1}^m \begin{cases} 1, & \text{jika } I(p, q) = i \text{ dan } I(p + \Delta x, q + \Delta y) = j \\ 0, & \text{jika lainnya} \end{cases}$$

Dalam rumus tersebut, i dan j merupakan nilai intensitas pada gambar, sedangkan p dan q adalah posisi pada gambar. Offset Δx dan Δy bergantung pada arah θ dan jarak yang diwakili oleh persamaan berikut

$$\Delta x = p \cos(\theta), \Delta y = p \sin(\theta)$$

Setelah didapat *co-occurrence matrix*, *symmetric matrix* dapat dibentuk dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Lalu cari *matrix normalization* dengan persamaan.

$$\text{MatrixNorm} = \frac{\text{MatrixOcc}}{\sum \text{MatrixOcc}}$$

Langkah-langkah dalam CBIR dengan parameter tekstur adalah sebagai berikut :

1. Konversikan warna gambar terlebih dahulu menjadi *grayscale*. Hal ini dilakukan karena warna tidaklah penting dalam penentuan tekstur. Adapun warna RGB dapat dikonversi menjadi suatu warna *grayscale* Y dengan rumus:

$$Y = 0.29 \times R + 0.587 \times G + 0.114 \times B$$

2. Lakukan kuantifikasi dari nilai *grayscale*. Karena citra *grayscale* berukuran 256 piksel, maka matriks yang berkoresponden akan berukuran 256×256 . Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.
3. Dari *co-occurrence matrix* bisa diperoleh banyak komponen ekstraksi tekstur, antara lain :

Contrast

$$\sum_{i,j=0}^{\text{dimensi} - 1} P_{i,j} (i - j)^2$$

Homogeneity

$$\sum_{i,j=0}^{\text{dimensi} - 1} \frac{P_{i,j}}{1 + (i - j)^2}$$

Entropy

$$-\left(\sum_{i,j=0}^{\text{dimensi}-1} P_{i,j} \times \log P_{i,j} \right)$$

Angular Second Moment (ASM)

$$\sum_{i,j=0}^{N-1} P_{i,j}^2$$

Energy

$$\sqrt{ASM}$$

Correlation

$$\sum_{i,j=0}^{levels-1} P_{i,j} \left[\frac{(i - \mu_i)(j - \mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}} \right]$$

Dissimilarity

$$\sum_{i,j=0}^{N-1} P_{i,j} |i - j|$$

Inverse Contrast

$$\sum_i \sum_j P(i, j) / (i - j)^2$$

Distortion

$$\sum_i \sum_j (i - j)^3 P(i, j)$$

Keterangan : P merupakan matriks *co-occurrence*

Dari sembilan komponen tersebut, dibuatlah sebuah vektor yang akan digunakan dalam proses pengukuran tingkat kemiripan.

4. Ukurlah kemiripan dari kedua gambar dengan menggunakan Teorema *Cosine Similarity*, yaitu:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Disini A dan B adalah dua vektor dari dua gambar. Semakin besar hasil *Cosine Similarity* kedua vektor maka tingkat kemiripannya semakin tinggi.

2.2 Website Development

Situs web dibangun menggunakan HyperText Markup Language (HTML), Cascading Style Sheets (CSS), serta Javascript (JS). Pengembangan aplikasi didukung oleh kerangka kerja Bootstrap untuk bagian depan (*frontend*) dan Flask untuk bagian belakang (*backend*). Sumber daya website disimpan dalam folder *src*, sementara masukan atau *input* disimpan dalam folder *test*. Sumber daya ini mencakup file *app.py* sebagai *file* utama yang menjalankan aplikasi. *File* utama tersebut mengimpor *file* *color.py* untuk melakukan Content-Based Image Retrieval (CBIR) berbasis warna, dan *file* *texture.py* untuk melakukan CBIR berbasis tekstur.

Kerangka kerja Flask memungkinkan pengembangan sistem yang terstruktur secara modular, memfasilitasi pembagian fungsi-fungsi utama ke dalam komponen-komponen terpisah. Kelebihan utamanya sebagai *backend* yang ringan memungkinkan penanganan permintaan dengan efisien. Penggunaan sistem *routing* dalam Flask memastikan pengelolaan perpindahan URL yang teratur dengan setiap rute memiliki fungsi khususnya sendiri.

Pada tahapan yang melibatkan proses yang memerlukan waktu yang cukup lama, pemanfaatan pustaka *concurrent.futures* memberikan kemampuan untuk menyelesaikan pekerjaan secara simultan dan paralel di latar belakang sistem. Hal ini berpotensi mengoptimalkan kinerja dan mengurangi waktu yang diperlukan untuk menyelesaikan tugas-tugas yang berat secara sekuensial. Pendekatan ini memungkinkan penanganan yang lebih efisien terhadap beban kerja yang besar, dengan penggunaan sumber daya yang lebih optimal. Kemampuan Flask dalam mengelola proses-proses ini secara efektif merupakan salah satu dari banyak alasan mengapa kerangka kerja ini menjadi pilihan yang populer dalam pengembangan backend.

Untuk menjalankan situs secara lokal, umumnya situs dapat diakses melalui alamat 127.0.0.1:5000 pada browser. Penggunaan opsi *debug=true* saat menjalankan Flask sangat berguna karena memungkinkan pemantauan dan pemecahan masalah secara efektif. Dengan pengaturan ini, seluruh proses eksekusi, termasuk informasi rinci tentang permintaan dan tanggapan server,

ditampilkan di terminal. Debug mode pada Flask memungkinkan pengembang untuk melihat pesan-pesan error secara langsung, membantu dalam identifikasi masalah dan memfasilitasi perbaikan kode secara *real-time*. Selain itu, informasi yang ditampilkan di terminal juga memberikan pandangan yang lebih mendalam tentang bagaimana aplikasi merespons terhadap permintaan yang diberikan, memudahkan penyesuaian dan peningkatan kinerja situs secara keseluruhan.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah Penyelesaian Masalah

Dalam melakukan analisis gambar melalui implementasi algoritma Content Based Information Retrieval (CBIR), kami terlebih dahulu memecah masalah implementasi algoritma CBIR menjadi masalah-masalah yang lebih sederhana. Kami menemukan bahwa implementasi algoritma CBIR bisa dipecah menjadi masalah yang lebih sederhana, yakni:

Color

- Konversi RGB menjadi HSV
- Mengubah matrix RGB dari gambar menjadi histogram warna
- Memecah gambar menjadi beberapa blok untuk efisiensi
- Membandingkan gambar dengan *Cosine Similarity*

Texture

- Konversi RGB menjadi *grayscale*
- Membentuk *co-occurrence matrix*
- Membentuk *similarity matrix* yang telah dinormalisasi
- Menghitung parameter-parameter *texture features*
- Membandingkan gambar dengan *Cosine Similarity*

3.2. Elemen Aljabar Geometri

Dalam konteks *Content Based Information Retrieval*, elemen-elemen aljabar geometri yang digunakan, antara lain:

a) Vektor

- Penggunaan vektor dapat mewakili fitur-fitur objek dalam ruang geometris. Misalnya, dalam konteks citra, setiap piksel atau kelompok piksel dapat direpresentasikan sebagai vektor fitur.
- Vektor fitur ini dapat mencakup informasi warna, tekstur, atau bentuk dari objek yang diwakili.

- b) Matriks Transformasi
 - Matriks transformasi dapat digunakan untuk memanipulasi dan mentransformasi representasi objek dalam ruang geometris.
 - Misalnya, matriks transformasi dapat digunakan untuk merotasi, mengubah skala, atau mentranslasikan objek.
- c) Ruang Euclidean
 - CBIR dapat menggunakan konsep ruang Euclidean untuk mengukur jarak antara vektor fitur objek. Ini memungkinkan untuk mengukur seberapa mirip atau berbeda dua objek dalam ruang fitur.
 - Matrik jarak seperti Euclidean distance atau Cosine similarity dapat digunakan untuk membandingkan vektor fitur.
- d) Dimensi
 - Konsep dimensi dapat digunakan untuk merepresentasikan berbagai fitur objek. Sebagai contoh, dalam citra, dimensi dapat mencakup warna merah, hijau, dan biru, serta tekstur dan intensitas.
- e) Perhitungan Geometri
 - Operasi aljabar geometri seperti perhitungan dot product, cross product, atau norm dapat digunakan untuk menggambarkan hubungan antara vektor-vektor fitur.

BAB IV

IMPLEMENTASI DAN UJI COBA

4.1. Spesifikasi dan Tata Cara Penggunaan Program

Langkah-langkah yang diberikan sebagai spesifikasi yang menjadi acuan dari alur program yang dirancang adalah sebagai berikut

1. Program menerima input *folder dataset* dan sebuah citra gambar.
2. Program menampilkan gambar citra gambar yang dipilih oleh pengguna.
3. Program dapat memberikan kebebasan pada pengguna untuk memilih parameter pencarian yang hendak digunakan (warna atau tekstur) melalui *toggle*.
4. Program mulai melakukan perhitungan nilai kecocokan antara *image* masukan dengan *dataset image* berdasarkan parameter yang telah dipilih (warna atau tekstur).
5. Program dapat menampilkan nilai kecocokan antara *image* masukan dengan setiap gambar dalam dataset.
6. Program menampilkan hasil luaran dengan melakukan *descending sorting* berdasarkan nilai kecocokan tiap gambar. Cukup tampilkan seluruh gambar yang **memiliki tingkat kemiripan > 60%** dengan gambar masukan.
7. Program mengimplementasikan *pagination* agar jumlah gambar dapat dibatasi dengan halaman-halaman tertentu. Jumlah gambar dalam dataset yang akan digunakan oleh asisten saat penilaian mungkin berbeda dengan jumlah gambar pada dataset yang kalian gunakan, sehingga pastikan bahwa *pagination* dapat berjalan dengan baik.
8. Program dapat menampilkan **jumlah gambar** yang memenuhi kondisi tingkat kemiripan serta **waktu eksekusi**

Dalam implementasinya, program kami dapat diimplementasikan dengan alur program sebagai berikut

1. Unggah dataset yang akan digunakan program. Dataset dapat berupa image folder atau bisa juga diperoleh menggunakan fitur scraping dari URL suatu situs.
2. Program akan mengolah dataset menjadi sebuah database yang dapat digunakan untuk perhitungan. Proses ini dapat memakan waktu sehingga pengguna harus menunggu hingga rendering database selesai
3. Setelah database berhasil dibuat, unggah query image yang akan dijadikan acuan dalam proses temu balik gambar.
4. Pilihlah mode fitur CBIR yang akan digunakan. Pengguna dapat memilih untuk menggunakan fitur *color* maupun *texture*. Kendati demikian, fitur *color* lebih direkomendasikan dibandingkan *texture* karena akurasinya yang lebih tinggi
5. Tekan tombol *search*. Program kemudian akan menampilkan gambar-gambar yang memiliki kemiripan dengan perbandingan nilai cosine similarity di atas 60%.
6. Pengguna dapat mengunduh hasil temu balik gambar dalam bentuk file pdf dengan menekan tombol *download*.

4.2. Implementasi Program

4.2.1. Texture

a) preprocess_image

```
function preprocess_image(image: Image) → Image:
{KAMUS LOKAL}
    dim, new_dim: integer
    Padded_image: Image
{ALGORITMA}
    image ← ImageOps.grayscale(image)

    dim ← max(image.size)

    new_dim ← 2 ^ ceil(log2(dim))

    padded_image ← ImageOps.pad(image,
    (new_dim, new_dim))

    → padded_image
```

Deskripsi

Mengubah image RGB menjadi grayscale untuk menyederhanakan perhitungan parameter tekstur. Hal ini bisa dilakukan karena warna tidaklah penting dalam membandingkan image dengan parameter-parameter tekstur

b) create_framework_matrix

```
function create_framework_matrix(a: np.array) → np.array
{KAMUS LOKAL}
    i, j, prev, current: integer
    framework_matrix: array of array of integer
{ALGORITMA}
    i traversal [0...255]
        j traversal [0...255]
            framework_matrix[i][j] ← 0

    i traversal [0...(a.rows - 1)]
        j traversal [0...(a.columns - 1)]
            if j = 0 then
                continue()
            else
                prev ← a[i][j - 1]
                current ← a[i][j]

                framework_matrix[prev][current] ←
                framework_matrix[prev][current] + 1

    → framework_matrix
```

Deskripsi
Membentuk framework matrix berdasarkan frekuensi dari kemunculan sebuah value pixel yang bersebelahan

c) contrast

```

function contrast(a: np.array) → float
{KAMUS LOKAL}
  i, j: integer
  c: float
{ALGORITMA}
  c ← 0

  i traversal [0...(a.rows - 1)]
    j traversal [0 to (a.columns - 1)]
      c ← c + (a[i][j] * (i - j) * (i - j))

  → c

```

Deskripsi
Menghitung <i>contrast</i> gambar berdasarkan rumus yang telah diberikan pada referensi

d) homogeneity

```

function homogeneity(a: np.array) → float
{KAMUS LOKAL}
  i, j: integer
  h: float
{ALGORITMA}
  h ← 0

  i traversal [0...(a.rows - 1)]
    j traversal [0 to (a.columns - 1)]
      h ← h + a[i][j] / (1 + (i - j) * (i - j))

  → h

```

Deskripsi
Menghitung <i>homogeneity</i> gambar berdasarkan rumus yang telah diberikan pada referensi

e) entropy

```

function entropy (a: np.array) → float
{KAMUS LOKAL}
    i, j: integer
    e: float
{ALGORITMA}
    e ← 0

    i traversal [0...(a.rows - 1)]
        j traversal [0 to (a.columns - 1)]
            if a[i][j] ≤ 0 then
                continue()
            else
                e ← e + a[i][j] * log(a[i][j])

    → (-1) * e

```

Deskripsi

Menghitung *entropy* gambar berdasarkan rumus yang telah diberikan pada referensi

f) dissimilarity

```

function dissimilarity(a: np.array) → float
{KAMUS LOKAL}
    i, j: integer
    d: float
{ALGORITMA}
    d ← 0

    i traversal [0...(a.rows - 1)]
        j traversal [0 to (a.columns - 1)]
            d ← d + (a[i][j] * |i - j|)

    → d

```

Deskripsi

Menghitung *dissimilarity* gambar berdasarkan rumus yang telah diberikan pada referensi

g) asm_val

```
function asm_val(a: np.array) → float
{KAMUS LOKAL}
  i, j: integer
  asm: float
{ALGORITMA}
  asm ← 0

  i traversal [0...(a.rows - 1)]
    j traversal [0 to (a.columns - 1)]
      asm ← asm + (a[i][j] * a[i][j])

  → asm
```

Deskripsi

Menghitung *angular second moment* gambar berdasarkan rumus yang telah diberikan pada referensi

h) correlation

```
function asm_val(a: np.array) → float
{KAMUS LOKAL}
  i, j: integer
  co, mu_x, mu_y, sigma_x, sigma_y: float
{ALGORITMA}
  co, mu_x, mu_y, sigma_x, sigma_y ← 0

  i traversal [0...(a.rows - 1)]
    j traversal [0...(a.columns - 1)]
      mu_x ← mu_x + i * a[i][j]
      mu_y ← mu_y + j * a[i][j]

  i traversal [0...(a.rows - 1)]
    j traversal [0...(a.columns - 1)]
      sigma_x ← sigma_x + (i - mu_x) * (i - mu_x) *
      a[i][j]
      sigma_y ← sigma_y + (j - mu_y) * (j - mu_y) *
      a[i][j]

  sigma_x ← sqrt(sigma_x)
  sigma_y ← sqrt(sigma_y)
```

```

i traversal [0...(a.rows - 1)]
  j traversal [0...(a.columns - 1)]
    if (sigma_x * sigma_y) ≤ 0 then
      continue()
    else
      co ← co + ((i - mu_x) * (j - mu_y) *
a[i][j]) / (sigma_x * sigma_y)
→ co

```

Deskripsi

Menghitung *correlation* gambar berdasarkan rumus yang telah diberikan pada referensi

i) inverse_contrast

```

function inverse_contrast(a: np.array) → float
{KAMUS LOKAL}
  i, j: integer
  ic: float
{ALGORITMA}
  ic ← 0

  i traversal [0...(a.rows - 1)]
    j traversal [0 to (a.columns - 1)]
      if (i - j) ≤ 0 then
        continue()
      else
        ic ← ic + a[i][j] / ((i - j) * (i - j))

→ ic

```

Deskripsi

Menghitung *inverse contrast* gambar berdasarkan rumus yang telah diberikan pada referensi

j) distortion

```
function distortion(a: np.array) → float
{KAMUS LOKAL}
    i, j: integer
    dt: float
{ALGORITMA}
    dt ← 0

    i traversal [0...(a.rows - 1)]
        j traversal [0 to (a.columns - 1)]
            dt ← a[i][j] * ((i - j) * (i - j) * (i - j))

    → dt
```

Deskripsi

Menghitung *distortion* gambar berdasarkan rumus yang telah diberikan pada referensi

k) save_csv

```
procedure save_csv(input dir: string)
{KAMUS LOKAL}
    i, j: integer
    co, mu_x, mu_y, sigma_x, sigma_y: float
    images: array of Image
{ALGORITMA}
    filenames ← get_filenames(directory)
    sorted_filenames ← sort_filenames(filenames)
    Images ← []

    for filename in sorted_filenames
        img = open_image(directory, filename)

        if (img = NOT None) then
            framework_matrix ←
                create_framework_matrix(preprocess_image(img))

            symmetric_matrix ←
                framework_matrix + transpose(framework_matrix)

            symmetric_matrix_normalized ←
                symmetric_matrix / sum(symmetric_matrix)
```

```

c, h, e, d, asm, en, co, ic, dt ←
(contrast(symmetric_matrix_normalized),
homogeneity(symmetric_matrix_normalized),
entropy(symmetric_matrix_normalized),
dissimilarity(symmetric_matrix_normalized),
asm_val(symmetric_matrix_normalized),
sqrt(asm_val(symmetric_matrix_normalized)),
correlation(symmetric_matrix_normalized),
inverse_contrast(symmetric_matrix_normalized),
distortion(symmetric_matrix_normalized)

append(images, ([c, h, e, d, asm, en, co, ic, dt]))

write_to_csv(images, "../test/db_texture.csv")

```

Deskripsi

Membuat file csv yang menyimpan *texture parameters* dari *dataset* yang ingin diolah

l) load_csv

```

function load_csv(filename: string) → array of array of
float
{KAMUS LOKAL}
i, j: integer
lines: string
arrays: array of array of float
values: array of float
{ALGORITMA}
lines = ""
with open(filename, 'r') as file:
    lines ← read_all_lines(file)

arrays = []
for line in lines:
    values ← list(map(float, line.strip().split(',')))
    append(arrays, values)

→ arrays

```


Deskripsi
Mengolah <i>input</i> file csv dengan mengonversi data di dalamnya menjadi sebuah array of array of float

m) cosine_similarity

```
function cosine_similarity(c1, h1, e1, d1, asm1, en1,
co1, ic1, dt1, c2, h2, e2, d2, asm2, en2, co2, ic2, dt2:
float) → float
{KAMUS LOKAL}
    cosine : float
{ALGORITMA}
    cosine ← ((c1 * c2) + (h1 * h2) + (e1 * e2) + (d1 *
d2) + (asm1 * asm2) + (en1 * en2) + (co1 * co2) +
(ic1 * ic2) + (dt1 * dt2)) / (sqrt(pow(c1, 2) +
pow(h1, 2) + pow(e1, 2) + pow(d1, 2) +
pow(asm1, 2) + pow(en1, 2) + pow(co1, 2) +
pow(ic1, 2) + pow(dt1, 2)) * sqrt(pow(c2, 2) +
pow(h2, 2) + pow(e2, 2) + pow(d2, 2) +
pow(asm2, 2) + pow(en2, 2) + pow(co2, 2) +
pow(ic2, 2) + pow(dt2, 2)))

    → cosine * 100
```

Deskripsi
Menghitung nilai <i>cosine similarity</i> berdasarkan <i>texture parameters</i> kedua gambar yang akan dibandingkan

n) search

```
function search(c1, h1, e1, d1, asm1, en1, co1, ic1, dt1:  
float) → array of float  
{KAMUS LOKAL}  
  i: integer  
  Res60, sorted_res: array of float  
  db: csv file of array of array of float  
  similarity: float  
{ALGORITMA}  
  i traversal [0...length(db) - 1]  
    c2, h2, e2, d2, asm2, en2, co2, ic2, dt2 ←  
    db[i][0], db[i][1], db[i][2], db[i][3], db[i][4],  
    db[i][5], db[i][6], db[i][7], db[i][8]  
  
    similarity ← cosine_similarity(c1, h1, e1, d1,  
    asm1, en1, co1, ic1, dt1, c2, h2, e2, d2, asm2,  
    en2, co2, ic2, dt2)  
    if similarity ≥ 60 then  
      res60[i] ← similarity  
  
  sorted_res = ← descending_sort(res60)  
  → sorted_res
```

Deskripsi
Mencari gambar dengan nilai <i>cosine similarity</i> yang lebih dari 60% untuk ditampilkan sebagai hasil pencarian

4.2.2. Color

a) loadImages

```
function loadImages(dir: string) → array of array of tuple  
of integer  
{KAMUS LOKAL}  
    filenames, sorted_filenames: string  
    images: array of array of tuple of integer  
{ALGORITMA}  
    filenames ← get_filenames(directory)  
    sorted_filenames ← sort_filenames(filenames)  
  
    images = []  
  
    for filename in sorted_filenames:  
        img ← read_image(directory, filename)  
        img ← convert_bgr_to_rgb(img)  
  
        if img is NOT None then  
            append(images, img)  
  
    → images
```

Deskripsi

Mencari gambar dari *dataset* dan menyimpannya dalam sebuah array of Image

b) rgb_to_hsv

```
function rgb_to_hsv(img: array of array of tuple of  
integer) → array of array of tuple of integer  
{KAMUS LOKAL}  
    cmax, cmin, delta, v: integer  
    r, g, b, hsv: array of array of tuple of integer  
    h, s: array of integer  
{ALGORITMA}  
    r, g, b ← img[:, :, 0], img[:, :, 1], img[:, :, 2]  
    cmax ← np.max(img, axis=2)  
    cmin ← np.min(img, axis=2)  
    delta ← cmax - cmin  
    h ← np.zeros_like(cmax)  
    s = np.zeros_like(cmax)  
    v ← cmax  
  
    with np.errstate(invalid='ignore'):  
        calculate_hue(h, delta, cmax, r, g, b)  
  
    s[cmax != 0] ← delta[cmax != 0] / cmax[cmax != 0]  
    hsv_image ← stack_arrays(h, s, v)  
    → hsv_image
```

Deskripsi
Melakukan konversi nilai RGB dari dataset yang telah disimpan dalam array of Image menjadi HSV

c) hsv_quantify

```

function map_h(h: integer) → integer
{KAMUS LOKAL}
-
{ALGORITMA}
  if h ≤ 25 then
    → 1
  else if h ≤ 40 then
    → 2
  else if h ≤ 120 then
    → 3
  else if h ≤ 190 then
    → 4
  else if h ≤ 270 then
    → 5
  else if h ≤ 295 then
    → 6
  else if h ≤ 315 then
    → 7
  else
    → 0

```

```

function map_sv(value: integer) → integer
{KAMUS LOKAL}
-
{ALGORITMA}
  if value < 0.2 then
    → 0
  else if value < 0.7 then
    → 1
  else
    → 2

```

```

function hsv_quantify(img: array of array of tuple of
integer) → array of array of tuple of integer
{KAMUS LOKAL}
  cmax, cmin, delta, v: integer
  r, g, b, hsv: array of array of tuple of integer
  h, s: array of integer
{ALGORITMA}
  h, s, v ← get_hsv_channels(img)

  mapped_h ← vectorize(map_h(h))
  mapped_s ← vectorize(map_sv(s))
  mapped_v ← vectorize(map_sv(v))
  mapped_image ← stack_arrays(mapped_h,
  mapped_s, mapped_v)

  → mapped_image

```

Deskripsi
Mengelompokkan image berdasarkan kategori yang ditentukan dari nilai HSV masing-masing gambar

d) split_image

```
function split_image(img: array of array of tuple of
integer) → array of array of tuple of integer
{KAMUS LOKAL}
  i, j, block_size_x, block_size_y: integer
  blocks, block: array of array of tuple of integer
{ALGORITMA}
  block_size_x ← img.rows // 4
  block_size_y ← img.columns // 4
  blocks = []

  i traversal [0...3]
    j traversal [0...3]
      if i = 3 then
        block_x ← img.rows - i * block_size_x
      else
        block_x ← block_size_x

      if j = 3 then
        block_y ← img.columns - j * block_size_y
      else
        block_y ← block_size_y

      block ← extract_block(img, i, j, block_size_x,
        block_size_y, block_x, block_y)

      append(blocks, block)

  → blocks
```

Deskripsi
Membagi masing-masing gambar dari dataset menjadi 4 x 4 blok untuk meningkatkan efisiensi program

e) build_vector

```
function build_vector(img: array of array of tuple of
integer) → array of array of tuple of integer
{KAMUS LOKAL}
    num_h_levels, num_sv_levels: integer
    blocks, block: array of array of tuple of integer
{ALGORITMA}
    num_h_levels ← 8
    num_sv_levels ← 3

    block_frequency_vectors = []

    for block in img:
        flattened_block ← flatten_block(block)

        block_frequency_vector ←
        initialize_block_frequency_vector(num_h_levels,
        num_sv_levels)
```

```
        for pixel in flattened_block:
            h_index ← get_hue_index(pixel)
            s_index ← get_saturation_index(pixel)
            v_index ← get_value_index(pixel)

            index ← calculate_index(h_index, s_index,
            v_index, num_sv_levels)

            block_frequency_vector[index] ←
            block_frequency_vector[index] + 1

        append(block_frequency_vectors,
        block_frequency_vector)

    block_frequency_vectors_array ←
    convert_to_numpy_array(block_frequency_vectors)
    → block_frequency_vectors_array
```

Deskripsi

Membentuk vektor warna dari gambar berdasarkan kategori HSV yang telah dikalkulasi

f) cosine_similarity_color

```

function cosine_similarity_color(A, B: np.array) → float
{KAMUS LOKAL}
    i, j: integer
    A, B: np.array of integer
    cosine, percent: float
{ALGORITMA}
    A ← convert_to_int64(A)
    B ← convert_to_int64(B)
    cosine ← 0

    i traversal [0 ... (A.rows - 1)]
        if (i = 5 OR i = 6 OR i = 9 OR i = 10) then
            cosine ← cosine + (1.25 *
                (dot_product(A[i], B[i]) / (norm(A[i]) *
                    norm(B[i]))))
        else
            cosine ← cosine + (dot_product(A[i], B[i]) /
                (norm(A[i]) * norm(B[i])))

    percent = (cosine / 20) * 100
    → percent

```

Deskripsi

Menghitung nilai *cosine similarity* berdasarkan vektor warna kedua gambar yang akan dibandingkan

g) save_color_csv

```

procedure save_color_csv()
{KAMUS LOKAL}
    i, j: integer
    images, hsv_quantized_image, feature_vector,
    flattened_vector: array of array of tuple of integer
    csv_writer: csv file
{ALGORITMA}
    if file_exists("../test/db_color.csv") then
        remove_file("../test/db_color.csv")

    images ←
    load_images_from_directory("../test/dataset")

    i traversal [0 ... (length(images) - 1)]
        hsv_quantized_image ←
        hsv_quantify(rgb_to_hsv(images[i]))

        feature_vector ←
        build_vector(split_image(hsv_quantized_image))

```

```

        flattened_vector ← flatten_array(feature_vector)

    for
        with open("../test/db_color.csv", "w", newline="")
        as file
            csv_writer ← csv.writer(file)
            csv_writer.writerow(flattened_vector)
    except Exception as e:
        print("Error: {e}")

```

Deskripsi

Membuat file csv yang menyimpan vektor warna dari *dataset* yang ingin diolah

h) load_color_csv

```
function load_color_csv(filename: string) → array of  
array of integer  
{KAMUS LOKAL}  
  i, j: integer  
  lines: string  
  arrays: array of array of integer  
  values: array of integer  
{ALGORITMA}  
  lines = ""  
  with open(filename, 'r') as file:  
    lines ← read_all_lines(file)  
  
  arrays = []  
  for line in lines:  
    values ← list(map(integer, line.strip().split(',')))  
    append(arrays, values)  
  
  → arrays
```

Deskripsi

Mengolah *input* file csv dengan mengonversi data di dalamnya menjadi sebuah array of array of integer

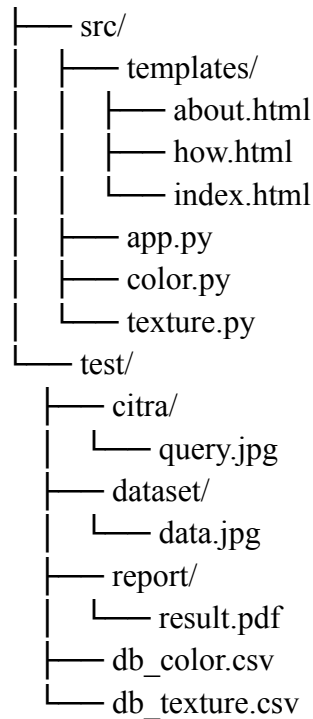
i) search_color

```
function search_color(img1: array of array of tuple of  
integer) → array of float  
{KAMUS LOKAL}  
  i: integer  
  res60, sorted_res: array of float  
  img1 ← array of array of tuple of integer  
  db: csv file of array of array of integer  
  similarity: float  
{ALGORITMA}  
  db ← load_color_csv("../test/db_color.csv")  
  res60 ← {}  
  
  i traversal [0...length(db) - 1]  
    similarity ← cosine_similarity_color(img1, db[i])  
    if similarity > 60 then  
      res60[i] ← similarity  
  
  sorted_res = ← descending_sort(res60)  
  → sorted_res
```


Deskripsi
Mencari gambar dengan nilai <i>cosine similarity</i> yang lebih dari 60% untuk ditampilkan sebagai hasil pencarian

4.2.3. Website

ALGEO02-22051/


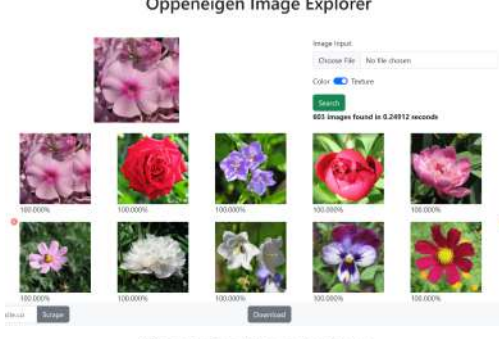

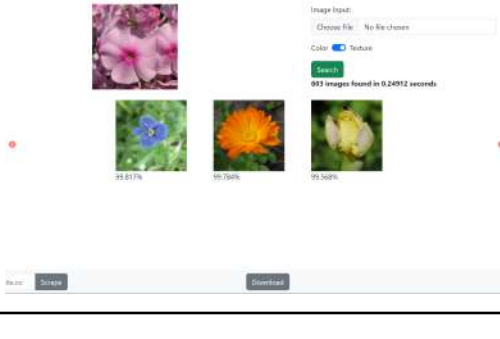


File utama website ketika dijalankan adalah file `app.py`. File utama ini akan mengimport `color.py` dan `texture.py` sebagai modul masing-masing fitur CBIR. Halaman utama website akan menampilkan `index.html`, seluruh proses CBIR dilakukan pada halaman ini.




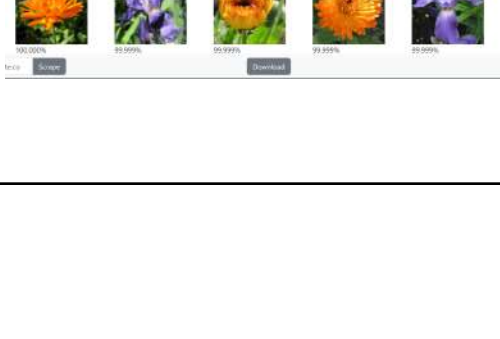
Ketika pengguna mengupload dataset atau melakukan scraping gambar, hasil gambar akan diletakkan pada folder `dataset` dengan penamaan dimulai dari nomor 0, 1, 2, ..., dst. Setelah masukan dataset selesai, program akan melakukan pelatihan dataset yang hasilnya disimpan dalam `db_color.csv` dan `db_texture.csv`. Jika pengguna mengupload query image, `citra` akan disimpan pada folder `citra` dengan penamaan dimulai dari nomor 0, 1, 2, ..., dst. Program akan otomatis membuat file laporan pencarian pada folder `report`, tetapi pengguna dapat memilih untuk mengunduh atau tidak. Selain itu, terdapat halaman tata cara penggunaan program pada file `how.html` dan halaman tentang pembuat program pada file `about.html`.

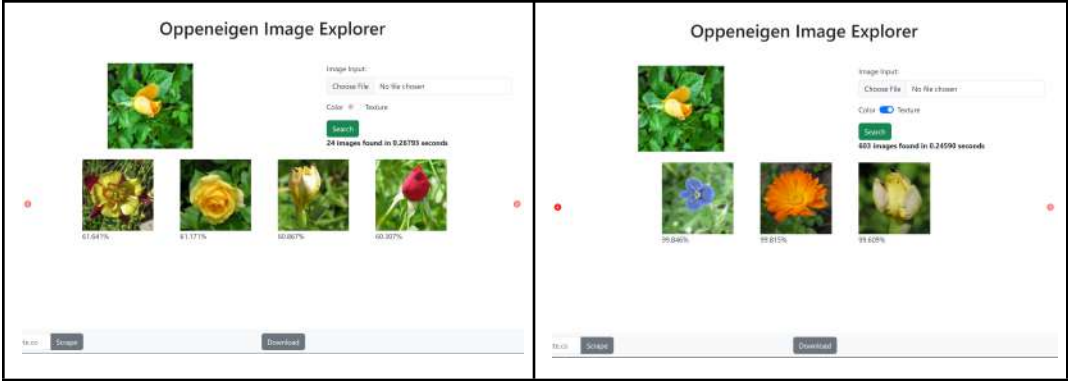
4.3. Uji Coba

Tabel 4.3.1 Eksperimen dengan Warna Masukan Mendekati Homogen dan Termasuk dalam Dataset

Color	Texture
	
	

Tabel 4.3.2 Eksperimen dengan Objek Utama Kecil dan Termasuk dalam Dataset

Color	Texture
	
	

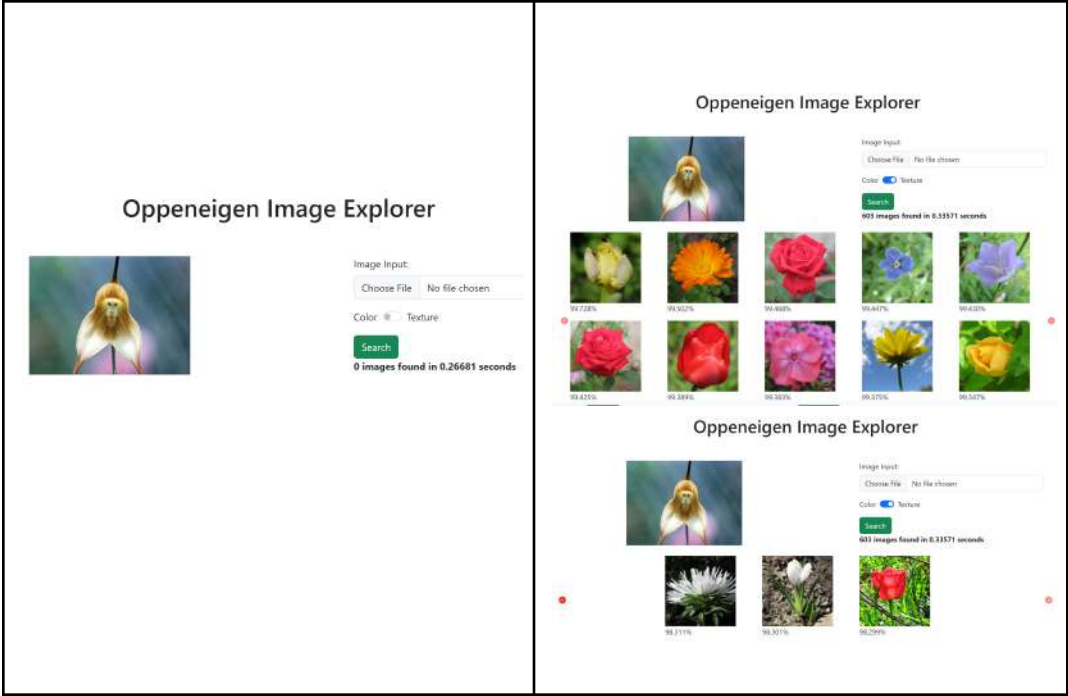


Tabel 4.3.3 Eksperimen dengan Warna Masukan Mendekati Homogen dan Masukan di luar Dataset

Color	Texture

Tabel 4.3.4 Eksperimen dengan Objek Utama Kecil dan Masukan di luar Dataset

Color	Texture
-------	---------

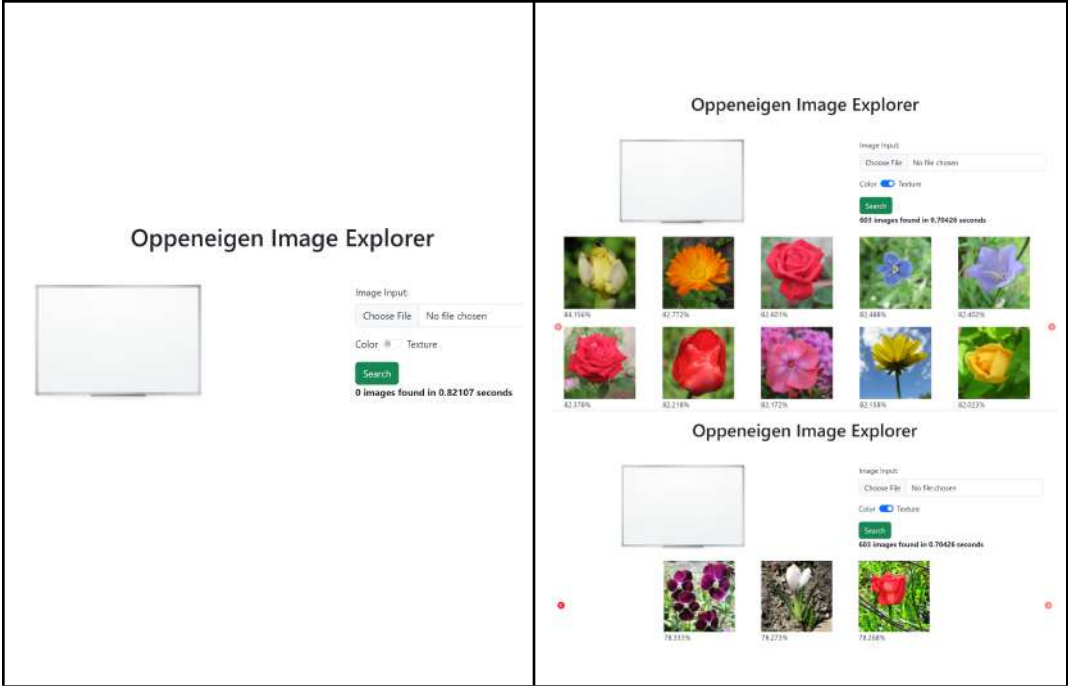


Tabel 4.3.5 Eksperimen dengan Jenis Objek yang Berbeda

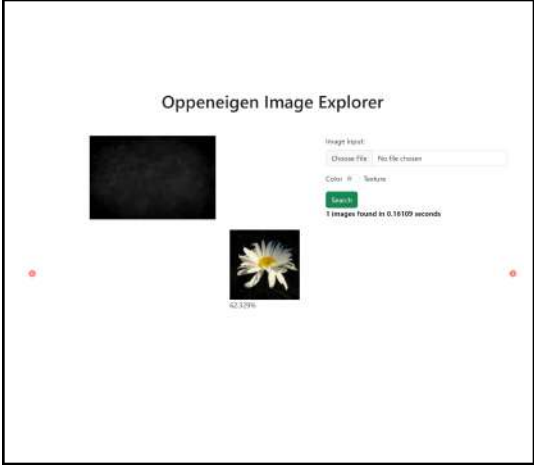
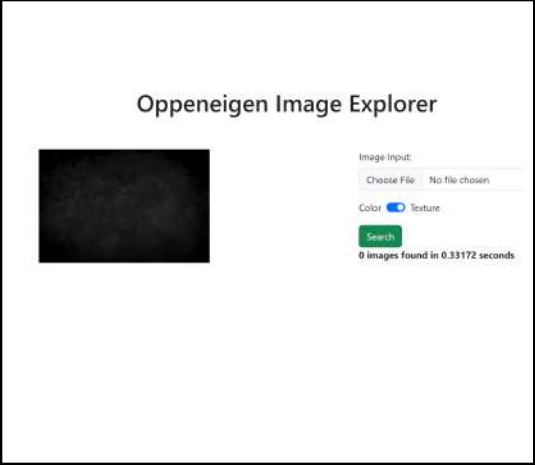
Color	Texture
<p>Oppeneigen Image Explorer</p> <p>The screenshot shows the Oppeneigen Image Explorer interface with a tiger image. The search results display a grid of similar images with their respective similarity scores.</p>	<p>Oppeneigen Image Explorer</p> <p>The screenshot shows the Oppeneigen Image Explorer interface with a tiger image. The search results display a grid of similar images with their respective similarity scores.</p>

Tabel 4.3.6 Eksperimen dengan Gambar Putih

Color	Texture
-------	---------

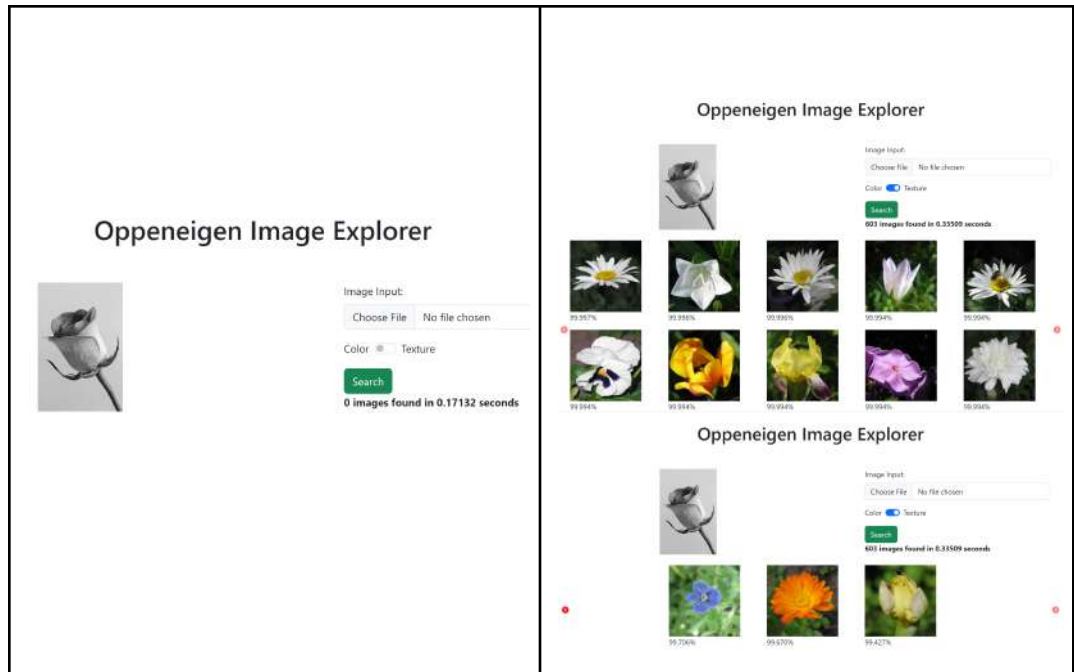


Tabel 4.3.7 Eksperimen dengan Gambar Hitam

Color	Texture
	

Tabel 4.3.8 Eksperimen dengan Gambar Abu-abu

Color	Texture
-------	---------



Menurut hasil uji coba, didapatkan bahwa algoritma CBIR dengan parameter warna jauh lebih akurat dibandingkan algoritma CBIR yang menggunakan parameter tekstur untuk mayoritas kasus. Algoritma CBIR dengan parameter tekstur unggul dari parameter warna hanya jika gambar yang menjadi *query* adalah gambar dalam format *grayscale*.

Ada beberapa faktor yang dapat menyebabkan hasil cosine similarity untuk parameter tekstur menjadi kurang optimal dalam Content-Based Image Retrieval (CBIR). Berikut adalah beberapa kemungkinan penyebab:

1. Ekstraksi Fitur yang Kurang Tepat

Jika ekstraksi fitur tekstur tidak memadai atau tidak memperhitungkan aspek tekstur yang relevan, hasil *cosine similarity* dapat menjadi kurang akurat. Pemilihan metode ekstraksi fitur tekstur yang tepat, seperti Gray Level Co-occurrence Matrix (GLCM) atau Local Binary Patterns (LBP), sangat penting.

2. Sensitivitas Terhadap Transformasi Citra

Cosine similarity dapat sensitif terhadap transformasi citra seperti rotasi, pergeseran, atau perubahan ukuran. Jika sistem tidak tahan terhadap variasi ini, hasil pencarian dapat menjadi kurang optimal.

3. Perubahan Kondisi Pencahayaan

Tekstur pada citra dapat dipengaruhi oleh kondisi pencahayaan. Jika algoritma tidak dapat menangani variasi pencahayaan, kemiripan kosinus dapat terpengaruh.

4. Kurangnya Representasi Fitur Lengkap

Jika fitur tekstur yang digunakan tidak mencakup semua aspek relevan dari tekstur, hasil pencarian dapat menjadi terbatas. Perlu mempertimbangkan representasi fitur yang lebih lengkap dan kompleks.

5. Pemilihan Threshold yang Tidak Tepat

Pengaturan threshold dalam metode cosine similarity dapat mempengaruhi hasilnya. Pemilihan threshold yang tidak tepat dapat menyebabkan hasil yang kurang optimal.

6. Kurangnya Data Pelatihan yang Representatif

Jika model atau algoritma dilatih dengan dataset yang tidak mencakup variasi yang cukup dari tekstur, kemampuannya untuk menangani berbagai kasus dapat terbatas.

7. Dimensi Fitur yang Tinggi

Jika dimensi fitur tekstur terlalu tinggi, *cosine similarity* dapat menjadi kurang efektif. Perlu dilakukan analisis dan reduksi dimensi jika diperlukan.

Untuk meningkatkan akurasi hasil cosine similarity pada CBIR, diperlukan pemilihan metode ekstraksi fitur yang cermat, penanganan variasi citra dengan baik, dan pengujian yang komprehensif terhadap berbagai kondisi citra

BAB V

KESIMPULAN

Aplikasi aljabar vektor dalam sistem temu balik gambar memberikan kontribusi penting dalam analisis dan pengelolaan konten visual. Aljabar vektor memungkinkan representasi matematis yang efisien dan dapat diolah dari citra, membuka peluang untuk pencarian dan analisis yang lebih canggih.

Dalam konteks sistem temu balik gambar, beberapa aplikasi utama aljabar vektor dapat disoroti. Pertama, representasi vektor dapat digunakan untuk menggambarkan fitur-fitur kunci dalam gambar. Melalui ekstraksi fitur, seperti warna, tekstur, dan bentuk, gambar dapat diubah menjadi vektor yang mewakili informasi esensial. Penerapan aljabar vektor dalam hal ini memungkinkan perbandingan dan pencarian gambar berdasarkan karakteristik visual tertentu. Kedua, teknik aljabar vektor dapat digunakan untuk menghitung kesamaan antara gambar. Melalui perhitungan kemiripan vektor, sistem temu balik gambar dapat mengidentifikasi gambar yang serupa atau memiliki karakteristik visual yang mirip. Hal ini mempermudah pengguna untuk menemukan gambar yang relevan dengan kebutuhan mereka tanpa harus bergantung pada pencarian teks.

Dengan menerapkan aljabar vektor, sistem temu balik gambar dapat menjadi lebih efisien, akurat, dan responsif terhadap kebutuhan pengguna. Kontribusi aljabar vektor dalam representasi dan analisis gambar menciptakan landasan matematis yang kuat untuk pengembangan sistem temu balik gambar yang lebih maju dan dapat diandalkan.

Akhir kata, besar harapan kami agar aplikasi aljabar vektor dalam sistem temu balik gambar terus dikembangkan dengan integrasi teknologi terkini. Selain itu, keberlanjutan penelitian dan pengembangan fitur-fitur inovatif diharapkan dapat memajukan kemampuan sistem, mendukung kebutuhan pengguna, dan meningkatkan daya saingnya di ranah temu balik gambar. Adapun saran untuk penelitian lebih lanjut adalah peningkatan akurasi pencarian dan efisiensi pengelolaan konten visual, khususnya untuk CBIR dengan menggunakan parameter tekstur.

REFLEKSI

Melalui tugas besar ini, kami bisa lebih memahami lebih mengenai materi aplikasi aljabar vektor yang diajarkan di kelas serta mengaplikasikannya dalam wujud sistem temu balik gambar. Kami kembali ditantang dengan “tugas yang melimpah dan lautan ujian” yang tentunya akan semakin berat ke depannya. Kelompok kami terkhususnya bahkan sempat mengalami kendala internal yang sempat mengancam selesainya pengerjaan Tugas Besar Aljabar Linier dan Geometri 2 ini. Namun, kami bangga karena telah menghadapi semua itu layaknya seorang kesatria yang memilih untuk mencoba dulu yang terbaik dibandingkan mengeluh dan menyerah. Kami menyadari bahwa kami tentu masih banyak kekurangan dan untuk itu kami ingin meminta maaf atas hal tersebut. Namun, layaknya sebuah proses belajar yang walau banyak jatuh bangunnya harus tetap memberikan esensi, kami merasa telah berhasil menyerap esensi dari Tugas Besar Aljabar Linier dan Geometri 2 dalam wujud peningkatan tingkat pemahaman kami mengenai materi aljabar vektor.

Untuk Tuhan, Bangsa, dan Almamater. Hidup Informatika!

DAFTAR PUSTAKA

Jun Yue, Zhenbo Li, Lu Liu, Zetian Fu, *Content-based image retrieval using color and texture fused features*, *Mathematical and Computer Modelling*, Volume 54, Issues 3–4, 2011. <https://www.sciencedirect.com/science/article/pii/S0895717710005352>.

Yunus, M. 2022. *Feature Extraction: Gray Level Co-occurrence Matrix (GLCM)*. <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1>. Diakses pada 05 November 2023

Felipe, Joaquim, et al. 2003. *Content of Medical Images Using Texture for Tissue Identification*. https://www.researchgate.net/figure/Texture-descriptors-proposed-by-Haralick_tbl1_221031099

Zhao, Qian, et al. 2014. *Role of the texture features of images in the diagnosis of solitary pulmonary nodules in different sizes*. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4153941/>

Hamami, Faqih. 2021. *Tutorial: Flask Web Framework in Python*. Ngoding Data. <https://ngodingdata.com/tutorial-flask-web-framework-python/>. Diakses pada 08 November 2023

Elijah, Ondiek, 2021. *Scraping the Web with Ease: Building a Python Web Scraper with Flask*. <https://dev.to/ondiek/building-a-python-web-scraper-in-flask-b87>. Diakses pada 18 November 2023

Github
<https://github.com/mdavaf17/Algeo02-22051>