

IF2123 Aljabar Linear dan Geometri

APLIKASI ALJABAR VEKTOR PADA SISTEM TEMU BALIK GAMBAR

Laporan Tugas Besar 2

Disusun untuk memenuhi tugas mata kuliah Aljabar Linear dan Geometri
pada Semester 1 (satu) Tahun Akademik 2023/2024

Oleh

Auralea Alvinia Syaikha **13522148**

Muhammad Davis **13522157**
Adhipramana

Pradipta Rafa Mahesa **13522162**

Kelompok CBIR



PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023

DAFTAR ISI

BAB 1 DESKRIPSI MASALAH	4
BAB 2 LANDASAN TEORI	5
2.1 CBIR (Content-Based Image Retrieval)	5
2.1.1 CBIR dengan parameter warna	5
2.1.2 CBIR dengan parameter tekstur	6
2.2 Model RGB (Red, Green, Blue)	7
2.3 Model HSV (Hue, Saturation, Value)	7
2.4 Pengembangan website	8
BAB 3 ANALISIS PEMECAHAN MASALAH	9
3.1 Langkah-langkah Pemecahan Masalah	9
3.1.1 Color	9
3.1.1.1 Konversi warna dari RGB ke HSV	9
3.1.1.2 Mencari tingkat kemiripan dengan Cosine Similarity	10
3.1.2 Texture	11
3.1.2.1 Perumusan matriks co-occurrence, matriks symmetric, dan matriks normalization	11
3.1.2.2 Perumusan komponen Contrast, Homogeneity, Dissimilarity, Entropy, ASM, Energy	12
3.1.2.3 Mencari tingkat kemiripan dengan Cosine Similarity	13
3.2 Proses Pemetaan Masalah	13
3.2.1 Ekstraksi Fitur dari Gambar	13
3.3 Contoh Ilustrasi Kasus	14
3.3.1 Color	14
3.3.2. Texture	14
BAB 4 IMPLEMENTASI	16
4.1. Notasi Algoritmik	16
4.1.1 Color	16
4.1.2 Tekstur	28
4.2. Penjelasan Struktur Program dan Arsitektur Kode	37
4.3. Penjelasan Tata Cara Penggunaan	41
4.4 Hasil Pengujian dan screenshot UI Program	42
4.4.1 Hasil Pengujian	42
4.4.2 Screenshot UI	44
BAB 5 KESIMPULAN	48
5. 1. Kesimpulan	48
5.2 Saran	48
5.2 Refleksi	49
5.2 Ruang Perbaikan dan Pengembangan	49
DAFTAR PUSTAKA	50

BAB 1 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.

Di dalam Tugas Besar 2 ini, penulis membuat program yang mengimplementasikan sistem temu balik gambar yang sudah dijelaskan sebelumnya dengan memanfaatkan Aljabar Vektor dalam bentuk sebuah *website*. 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. Berikut spesifikasi program:

1. Program menerima input *folder dataset* dan sebuah citra gambar.
2. *Dataset* gambar dapat diunduh secara mandiri melalui pranala [berikut](#). Akan tetapi peserta diperbolehkan untuk menggunakan *dataset* lain yang telah dipersiapkan oleh kelompok masing-masing.
3. Program menampilkan gambar citra gambar yang dipilih oleh pengguna.
4. Program dapat memberikan kebebasan pada pengguna untuk memilih parameter pencarian yang hendak digunakan (warna atau tekstur) melalui *toggle*. *Default* parameter yang digunakan di awal dibebaskan kepada masing-masing kelompok.
5. Program mulai melakukan perhitungan nilai kecocokan antara *image* masukan dengan *dataset image* berdasarkan parameter yang telah dipilih (warna atau tekstur).
6. Program dapat menampilkan nilai kecocokan antara *image* masukan dengan setiap gambar dalam dataset.
7. 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.
8. 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.
9. Program dapat menampilkan jumlah gambar yang memenuhi kondisi tingkat kemiripan serta waktu eksekusi.

BAB 2 LANDASAN TEORI

2.1 CBIR (*Content-Based Image Retrieval*)

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Tahapan awal dari proses ini melibatkan ekstraksi fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk. Setelah fitur-fitur ini berhasil diekstraksi, mereka diwujudkan dalam bentuk vektor atau deskripsi numerik yang dapat dibandingkan dengan fitur gambar lainnya. Selanjutnya, CBIR memanfaatkan algoritma pencocokan untuk menilai kesamaan antara vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar yang ada dalam dataset. Hasil pencocokan ini kemudian digunakan untuk menyusun urutan gambar dalam dataset dan menampilkan gambar yang paling serupa dengan gambar yang sedang dicari. CBIR memiliki 2 parameter yang paling populer, yaitu:

2.1.1 CBIR dengan parameter warna

Dalam sistem Temu Balik Gambar Berbasis Konten (CBIR) ini, perbandingan antara gambar input dan gambar yang terdapat dalam dataset dilakukan dengan mengubah representasi warna dari gambar yang awalnya berformat RGB menjadi metode histogram warna yang lebih umum. Histogram warna mencerminkan frekuensi munculnya berbagai warna dalam suatu ruang warna tertentu dan bertujuan untuk mendistribusikan warna pada gambar. Namun, perlu dicatat bahwa histogram warna tidak memiliki kemampuan untuk mendeteksi objek spesifik dalam gambar atau memberikan informasi tentang posisi distribusi warna tersebut.

Proses pembentukan ruang warna menjadi langkah kritis dalam membagi nilai citra menjadi beberapa rentang nilai yang lebih kecil. Hal ini dilakukan untuk menciptakan histogram warna di mana setiap interval atau rentang nilai dianggap sebagai "bin." Perhitungan histogram warna melibatkan penghitungan piksel yang mencakup nilai warna pada setiap interval. Fitur-fitur warna yang dihasilkan mencakup histogram warna global dan histogram warna blok.

Dalam menghitung histogram, pemilihan model warna HSV (Hue, Saturation, Value) menjadi pilihan utama karena kemampuannya untuk menangani gambar yang memiliki latar belakang kertas putih yang umum digunakan. Pilihan ini membantu dalam menyusun representasi warna yang lebih sesuai dengan kondisi visual umum, memastikan keakuratan dan kegunaan sistem temu balik gambar.

2.1.2 CBIR dengan parameter tekstur

CBIR yang melibatkan perbandingan tekstur melibatkan penerapan co-occurrence matrix, sebuah matriks yang dipilih karena kemampuannya untuk melakukan pemrosesan secara efisien dan cepat. Keuntungan lainnya adalah ukuran vektor yang dihasilkan lebih kompak. Sebagai contoh, untuk suatu gambar I dengan dimensi $n \times m$ piksel dan parameter offset $(\Delta x, \Delta y)$, rumus matriksnya dapat dinyatakan sebagai berikut: Dengan nilai i dan j mewakili intensitas gambar, sementara p dan q merepresentasikan posisi dalam gambar. Offset Δx dan Δy ditentukan oleh arah θ dan jarak yang dapat dihitung menggunakan persamaan berikut.

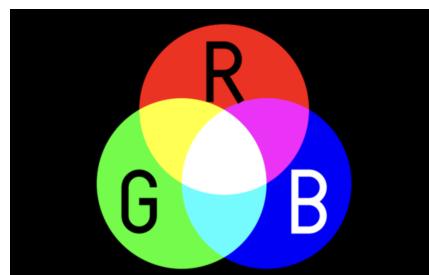
$$C_{\Delta x, \Delta y}(i, j) = \sum_{p=1}^n \sum_{q=1}^m \begin{cases} 1, & \text{jika } I(p, q) = i \text{ dan } I(p + \Delta x, q + \Delta y) = j \\ 0, & \text{jika lainnya} \end{cases}$$

Dari persamaan tersebut digunakan nilai θ sebesar $0^\circ, 45^\circ, 90^\circ$, dan 135° . Lalu, membuat *co-occurrence matrix* dan *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Berikut adalah contoh dari pembuatan *co-occurrence matrix*.

	1	2	3	4	5	6	7	8
1	1	1	2	0	0	1	0	0
2	0	0	1	0	1	0	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0
5	1	0	0	0	0	1	2	0
6	0	0	0	0	0	0	0	1
7	2	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0

2.2 Model RGB (*Red, Green, Blue*)

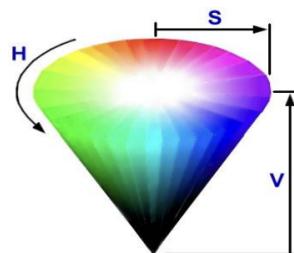
Model warna RGB beroperasi berdasarkan konsep penambahan intensitas cahaya primer, yaitu Merah (Red), Hijau (Green), dan Biru (Blue). Dalam suatu ruang tanpa cahaya, situasi tersebut dikategorikan sebagai kondisi gelap total, di mana mata atau sensor RGB tidak menyerap sinyal gelombang cahaya (0,0,0). Jika kita memperkenalkan cahaya merah ke dalam ruangan tersebut, warna ruangan akan berubah menjadi merah, contohnya dengan nilai RGB (255,0,0). Dengan demikian, semua objek dalam ruangan hanya akan terlihat dalam warna merah. Hal yang sama berlaku ketika cahaya yang diintroduksi adalah hijau atau biru. Struktur warna RGB dapat dilihat dalam gambar berikut.



Konsep ini didasarkan pada teori tri-stimulus vision, yang menyatakan bahwa manusia melihat warna dengan cara membandingkan cahaya yang tiba dengan sensor peka cahaya pada retina, yang memiliki sensor terdepan yang paling peka terhadap panjang gelombang cahaya 630 nm (merah), 530 nm (hijau), dan 450 nm (biru). Visualisasi model ini dapat diilustrasikan sebagai kubus dengan sumbu R, G, dan B.

2.3 Model HSV (*Hue, Saturation, Value*)

Model warna HSV mendefinisikan warna dalam terminologi Hue, Saturation dan Value. Hue menyatakan warna sebenarnya, seperti merah, violet, dan kuning. Hue digunakan untuk membedakan warna-warna dan menentukan kemerahan (redness), kehijauan (greeness), dan sebagainya, dari cahaya. Hue berasosiasi dengan panjang gelombang cahaya. Saturation menyatakan tingkat kemurnian suatu warna, yaitu mengindikasikan seberapa banyak warna putih diberikan pada warna. Value adalah atribut yang menyatakan banyaknya cahaya yang diterima oleh mata tanpa memperdulikan warna. Model warna HSV dapat dilihat pada gambar berikut.



2.4 Pengembangan website

Pengembangan website merupakan suatu proses kompleks yang melibatkan integrasi berbagai aspek teknologi dan desain untuk menciptakan pengalaman pengguna yang optimal. Bahasa pemrograman seperti HTML, CSS, dan JavaScript merupakan fondasi esensial dalam pembuatan halaman web. HTML digunakan untuk mendefinisikan struktur dasar konten, CSS bertanggung jawab atas tata letak dan penataan estetis, sementara JavaScript memberikan interaktivitas dan dinamisme pada halaman web.

Desain responsif juga menjadi landasan krusial dalam pengembangan website modern. Konsep ini memastikan bahwa website dapat merespons dengan baik terhadap berbagai ukuran layar, memungkinkan pengguna mengakses konten dengan kenyamanan dari berbagai perangkat, seperti komputer desktop, tablet, atau ponsel cerdas.

Aspek server-side scripting, seperti PHP, Python, atau Ruby, memainkan peran penting dalam mengelola logika bisnis, berkomunikasi dengan database, dan menyajikan konten dinamis kepada pengguna. Di sisi lain, client-side scripting, terutama melalui JavaScript, memberikan kemampuan interaktivitas di tingkat pengguna, memperkaya pengalaman browsing.

BAB 3 ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Sebelum mengekstraksi fitur-fitur yang relevan dari kumpulan data yang dimiliki, langkah awal yang harus diambil adalah melakukan pengumpulan dan penempatan dataset. Dataset ini akan berfungsi sebagai sumber data untuk menghasilkan output sesuai dengan permintaan pengguna melalui antarmuka web. Dataset disimpan dalam direktori src/BackEnd/dataset yang terdapat dalam struktur direktori proyek kami. Dataset diperoleh dari dua sumber, yaitu melalui unggahan dari pengguna dan melalui proses scraping dari internet yang dilakukan secara otomatis menggunakan Python. Selain itu, inputan gambar pengguna yang akan diproses akan disimpan di direktori src/BackEnd/input_images. Inputan gambar diperoleh dari dua sumber, yaitu melalui unggahan file gambar dari pengguna dan melalui *webcam*. Setelah memperoleh kedua data tersebut, program baru dapat mengekstraksi fitur-fitur dengan langkah sebagai berikut.

3.1.1 Color

3.1.1.1 Konversi warna dari RGB ke HSV

1. Memecah Gambar menjadi 4x4 Blok, perhitungan selanjutnya akan dilakukan per blok.
2. Mencari nilai RGB dari setiap pixel lalu dinormalisasi dengan mengubah nilai range [0,255] menjadi [0,1]

$$R' = \frac{R}{255} \quad G' = \frac{G}{255} \quad B' = \frac{B}{255}$$

3. Menghitung Cmax, Cmin, dan Δ

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

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

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

4. Dari hasil perhitungan diatas, HSV dapat dihitung dengan rumus berikut.

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

5. Dari hasil perhitungan diatas, kemudian kuantifikasi HSV berdasarkan rumus berikut.

$$H = \begin{cases} 0 & h \in [316, 360] \\ 1 & h \in [1, 25] \\ 2 & h \in [26, 40] \\ 3 & h \in [41, 120] \\ 4 & h \in [121, 190] \\ 5 & h \in [191, 270] \\ 6 & h \in [271, 295] \\ 7 & h \in [295, 315] \end{cases}$$

$$S = \begin{cases} 0 & s \in [0, 0.2) \\ 1 & s \in [0.2, 0.7) \\ 2 & s \in [0.7, 1] \end{cases}$$

$$V = \begin{cases} 0 & v \in [0, 0.2) \\ 1 & v \in [0.2, 0.7) \\ 2 & v \in [0.7, 1]. \end{cases}$$

6. Buatlah Histogram yang berisi frekuensi kemunculan dari range H,S,V diatas.

3.1.1.2 Mencari tingkat kemiripan dengan *Cosine Similarity*

1. Setelah dilakukan perhitungan Histogram HSV dari setiap blok dari gambar yang ingin dibandingkan dan gambar dari dataset.
2. Ukur kemiripan dari masing-masing unsur (H,S,V) dari masing-masing blok dengan menggunakan Teorema *Cosine Similarity* dengan persamaan berikut. A adalah vektor dari gambar , B adalah vektor dari dataset , n adalah banyak data dalam vektor (H : 8, S : 3, V : 3)

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

3. Tambahkan kemiripan H,S,dan V dari suatu blok lalu dibagi 3.
4. Tambahkan kemiripan HSV semua blok lalu dibagi banyak blok ($4 \times 4 = 16$).
5. Semakin besar hasil *Cosine Similarity* dari kedua vektor, maka tingkat kemiripannya semakin tinggi.

3.1.2 Texture

3.1.2.1 Perumusan matriks co-occurrence, matriks symmetric, dan matriks normalization

1. Mencari $\text{offset } \Delta x$ dan Δy yang bergantung pada arah θ dan jarak yang digunakan dengan persamaan berikut.

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

Dengan nilai i dan j sebagai nilai intensitas dari gambar, dan p serta q sebagai posisi dari gambar.

2. Dari persamaan tersebut, digunakan nilai θ adalah 0° , 45° , 90° , dan 135° .
3. Pada code yang kita gunakan, kita memilih 45 sebagai θ nya. Kita hanya menggunakan satu angle agar proses tidak berjalan terlalu lama. Kita memilih sudut 45 derajat karena dalam ekstraksi featurenya, lebih akurat daripada angle lainnya. Dapat dilihat pada tabel berikut untuk keakuratan angle pada ekstraksi fiturnya

Table 3 The results of the accuracy of the influence
of the Angle used on the accuracy

Feature	Angle $^\circ$	Accuracy (%)
Dissimilarity	0	71
	45	75
	90	53
	135	67
Correlation	0	56
	45	77
	90	78
	135	56
Homogeneity	0	57
	45	56
	90	51
	135	70
Contrast	0	58
	45	77
	90	67
	135	77
ASM	0	81
	45	88
	90	83
	135	79
Energy	0	46
	45	57
	90	41
	135	43

4. Untuk mencari co-occurrence matrix dengan $\theta, 0$, kita cukup memasukkan pada grayscale index saat itu dan index di kananya. Namun perlu juga untuk handle jika current pixelnya sudah berada di ujung maka perlu handler agar tidak melebihi width atau height dari grayscale matrix nya.
5. Setelah didapat *co-occurrence matrix*, hitung symmetric matrix dengan menjumlahkan co-occurrence matrix dengan hasil transpose-nya.
6. Hitung matrix normalization dengan persamaan berikut.

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

3.1.2.2 Perumusan komponen *Contrast, Homogeneity, Dissimilarity, Entropy, ASM, Energy*

1. Konversi warna gambar menjadi grayscale dengan mengubah warna RGB menjadi suatu warna grayscale Y dengan rumus berikut.

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

2. Lakukan kuantifikasi dari nilai *grayscale*. Karena citra *grayscale* berukuran 256 piksel, maka matriks yang berkoresponden akan berukuran 256×256 .
3. Dari *co-occurrence matrix* bisa diperoleh beberapa komponen untuk feature GLCM Matriksnya. Pada kali ini kita mengekstrak 6 komponen ekstraksi tekstur, yaitu *contrast, dissimilarity, homogeneity, asm, energy, entropy* dengan persamaan berikut

- ‘contrast’: $\sum_{i,j=0}^{levels-1} P_{i,j}(i - j)^2$
- ‘dissimilarity’: $\sum_{i,j=0}^{levels-1} P_{i,j}|i - j|$
- ‘homogeneity’: $\sum_{i,j=0}^{levels-1} \frac{P_{i,j}}{1+(i-j)^2}$
- ‘ASM’: $\sum_{i,j=0}^{levels-1} P_{i,j}^2$
- ‘energy’: \sqrt{ASM}
- ‘entropi’: $-\sum_{ij=0}^{levels-1} P_{ij} \log_2 P_{ij}$

3.1.2.3 Mencari tingkat kemiripan dengan Cosine Similarity

1. Setelah dilakukan perhitungan untuk GLCM matriks.
2. Diambil mengambil feature untuk GLCM matriks, seluruh feature itu kemudian dijadikan datu list. Dari ini didapatkan suatu vektor dari seluruh feature tersebut
3. Ukur kemiripan dengan menggunakan Teorema *Cosine Similarity* dengan persamaan berikut.n adalah 6 karena feature yang kita gunakan sebanyak 6

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

4. Semakin besar hasil *Cosine Similarity* dari kedua vektor, maka tingkat kemiripannya semakin tinggi.

3.2 Proses Pemetaan Masalah

Pertama-tama, perlu dipahami bagaimana gambar, yang semula merupakan representasi visual, dapat diubah menjadi bentuk vektor yang dapat diolah matematis. Pemetaan ini melibatkan konsep representasi vektor di mana setiap elemen vektor mewakili fitur-fitur tertentu dari gambar, seperti warna, tekstur, atau bentuk. Selanjutnya, pemetaan masalah juga mencakup pemahaman bagaimana aljabar linear dan teknik pemrosesan matematis dapat diterapkan pada vektor-vektor ini dalam konteks CBIR. Dengan memahami hubungan matematis di antara fitur-fitur gambar, sistem CBIR dapat secara efisien menemukan gambar yang serupa berdasarkan karakteristik tertentu yang diinginkan oleh pengguna.

3.2.1 Ekstraksi Fitur dari Gambar

- a) Ekstraksi Warna
 1. Ekstraksi warna dapat dilakukan dengan mengkonversi gambar menjadi potongan RGB
 2. Setiap piksel dapat direpresentasikan sebagai vektor warna dengan nilai intensitas untuk setiap saluran warna. Vektor warna inilah yang dimaksud sebagai RGB
 3. Ambil nilai intensitas Red (R), Green (G), dan Blue (B) dari setiap piksel.
 4. Normalisasi nilai R,G,dan B
 5. Hitung Hue, Saturation, dan Value
 6. Kuantifikasi H,S,dan V
 7. Buat histogram H,S,dan V
 8. Bandingkan histogram H,S,dan V gambar dengan H,S,dan V dari dataset dengan cosine similarity

b) Ekstraksi Tekstur

Pada kasus penyelesaian CBIR Tekstur, berdasarkan referensi yang ada di Spesifikasi, terdapat beberapa langkah yang harus diselesaikan untuk mendapatkan ekstraksi feature dari GLCM Matriks. Langkah tersebut adalah:

1. Mendapatkan Greyscale Matriks
2. Quantisasi nilai Grayscale matriks
3. Mencari GLCM Matriks
4. Buat symmetric matriks
5. Normalisasikan symmetric matriks tersebut
6. Terakhir, ekstrak fitur dari GLCM matriks gunakan cosine_similarity untuk mendapatkan data similarity nya dengan image yang lain.

3.3 Contoh Ilustrasi Kasus

3.3.1 Color

Pada langkah untuk mendapatkan ekstraksi R,G,B dari gambar kita hanya menggunakan melakukan slice pada data image untuk mendapatkan list red green dan blue nya. Pada langkah selanjutnya adalah mencari nilai untuk H,S,V nya. Untuk ini kita hanya perlu mengikuti rumus yang sudah diberikan. Setelah nilai H,S,V sudah didapatkan langkah berikutnya adalah mencari nilai histogram untuk masing masing H,S,V. Pada awalnya kita tidak memahami rumus dari mencari histogram ini dan membuat rumus lain yang tidak relevan. Hal ini baru disadari pada setelah testing karena kita mendapatkan nilai Histogram untuk Hue dengan nilai yang kebanyakan 0. Setelah melakukan beberapa research kami baru memahami bagaimana mencari nilai Histogram untuk HSV nya dari salah satu referensi yang ada di spek. Selanjutnya kami mencoba untuk memecahnya menjadi blok. Pada awalnya kami juga salah memahami maksud menjadikan 4x4 blok. Pada awalnya kami mengira untuk membagi image menjadi 4 yaitu matriks untuk pixel 0- $\frac{1}{4}$ image, $\frac{1}{4}$ image- $\frac{1}{2}$ image, $\frac{1}{2}$ image - $\frac{3}{4}$ image, $\frac{3}{4}$ image- 1 image. Misalkan 512x512 maka kita akan membaginya menjadi matriks yang akan memproses image pada pixel 0-128 dan 0-128. Namun ternyata kita harus mendapatkan seluruh kemungkinan pixel. Atau akan dihasilkan sejumlah 16 total matriks. Semua matriks ini akan diproses pada akhirnya. Jadi alur nya adalah kita memecah per image menjadi 16 blok lalu melakukan proses hsv hingga didapatkan seluruh histogramnya. Setelah semua itu didapatkan akan dicari cosine similarity untuk tiap blok pada 1 gambar dengan blok pada gambar lainnya. Setelah itu semuanya akan digabungkan menjadi 1 data similarity. Kesulitan pada kasus ini adalah memahami seluruh rumus yang ada. Untuk kecepatan pemrosesan pada color tidak terlalu lama yaitu dapat mendapat 100 image untuk 5 detik.

3.3.2. Texture

Pada langkah untuk mendapatkan grayscale matriks tidak ada yang spesial pada langkah ini cukup terapkan rumus untuk mendapatkan nilai grayscale dari RGB. pada langkah kedua, ada sedikit kebingungan mengenai value grayscale yang diambil ada sumber mengatakan untuk menggunakan nilai tertinggi dari grayscale atau nilai maximalnya yaitu 256. Pada pencarian GLCM Matriks, pada awalnya kami menggunakan seluruh angle yang ada yaitu 0,45,60,90 sampai 315. Untuk mendapatkan GLCM Matriks ini perlu sedikit memahami bagaimana implementasinya pada code.

Pada pencarian GLCM Matriks itu sendiri sebenarnya ada poin lain yang harus diperhatikan yaitu distance namun kami memilih distance 1 sebagai defaultnya untuk mendapatkan hasil yang lebih akurat. Selanjutnya untuk pencarian symmetric matriks, normalisasi, serta cosine Similarity nya cukup terapkan rumus yang sudah diberikan. Setelah seluruh code untuk CBIR Tekstur selesai dilakukan pengujian untuk beberapa image. Namun setelah menunggu lama tidak ada yang terjadi. Lalu kami mencoba menggunakan folder yang hanya berisi 1 image saja. Setelahnya kami mendapatkan hasil yaitu 99 % similarity dengan waktu yang cukup lama yaitu 50 detik. Oleh karena itu, kami mencoba untuk menggunakan library numpy untuk mempercepat kalkulasi matriksnya. Dimulai dengan menerapkan nya untuk greyscale hingga GLCM Matriks. Dengan numpy ini banyak sekali hal baru seperti penggunaan numpy.indices yang mengambil indeks i,j untuk ukuran tertentu dan penggunaan bitwise operation untuk masking data. Namun setelah dicoba lagi kami mendapatkan untuk 1 image membutuhkan waktu sekitar 5 detik. Setelah mencari solusi yang cukup lama dan bertanya kesana kemari, kami memutuskan untuk menggunakan cukup 1 angle yaitu 0 derajat. Pada awalnya kami menggunakan 0 tanpa alasan. Kami mendapatkan hasil yang cukup memuaskan yaitu 1 image per detik. Setelah itu, kami mencoba mencari cara yang lebih ekstrim lagi agar data lebih cepat. Pada awalnya kami hanya berencana untuk menggunakan caching agar data dapat diproses lebih cepat, tetapi kami menemukan suatu hal yang menarik dan ini disebut multi thread. Multi thread ini memaksakan komputer kita agar berjalan lebih kuat namun dapat mempercepat pemrosesan kode. Kami mencoba mengimplementasikan library numba. Penggunaan numba sangat simple namun membutuhkan beberapa hal harus dihapuskan seperti penggunaan np.indices hal ini tentu berpengaruh pada struktur code yang dibuat. Oleh karena itu kami hanya menggunakan numba di beberapa tempat yang banyak dilakukan kalkulasi yaitu pada pembuatan co-occurrence matriks. Setelah selesai implementasi numba, kami mendapatkan hasil yang sangat baik yaitu 1000 image dalam 12 detik. Setelah itu kami melakukan banyak bug testing dan kami juga menemukan bahwa angle 45 derajat lebih akurat dari angle lainnya sehingga kami memutuskan untuk menggunakan angle tersebut. Permasalahan yang terakhir adalah dimana hasil yang kami dapatkan adalah banyak sekali foto yang tidak mirip tapi memiliki kemiripan sebanyak 99 persen.

BAB 4 IMPLEMENTASI

4.1. Notasi Algoritmik

4.1.1 Color

```
function Color(input filepath : string, input folderpath : string)

KAMUS LOKAL

dataFile : array
dataSimilarity : array

function time.time() → float
{ Mendapatkan Waktu saat dieksekusi }

function cv2.imread(filepath : string) → array of bytes
{ Mengembalikan gambar dalam bytes }

function block_processing(input_image : array of bytes) → array
of float
{ Mengekstrak list dari nilai h,s, dan v dari suatu gambar pada
setiap blok}

procedure DataSetHSVBlockProcess(input folderpath : string,
input img_h : array of floats, input img_s : array of
floats, input img_v : array of floats,)
{ Mencari dan menampilkan nilai similarity dari setiap gambar
dalam folder path dengan gambar awal yang sudah diproses
menjadi list h,s,dan v}

function mergeSort(file,similarity) → array
{ Menghasilkan gabungan dari file dan similarity dan di sort }
```

ALGORITMA

```
start_time ← time.time()
input_image ← cv2.imread(filepath)

img_h, img_s, img_v ← block_processing(input_image)
DataSetHSVBlockProcess(folderpath,img_h,img_v,img_s)
```

```

end_time ← time.time()
elapsed_time ← end_time - start_time

end_time ← time.time()
elapsed_time ← end_time - start_time
dataFile,dataSimilarity ← mergeSort(dataFile,dataSimilarity)

insertLast(dataFile,"Time") {fungsi append}
insertLast(dataSimilarity,elapsed_time) {fungsi append}
sim_dict ← empty dictionary
for all key-value pairs (dataFile, dataSimilarity) in sim_dict
→ sim_dict

```

```

function cosine_similarity(hist1 : array of float , hist2 : array
of float) → float
{ Menghasilkan hasil cosine similarity dari 2 vektor (hist1 dan
hist2) }

```

KAMUS LOKAL

```

function length_vec(vec1 : array of float) → float
{Menghasilkan panjang dari vec1}

function dot(vec1 : array of float, vec2 : array of float) →
float
{ Menghasilkan hasil dot product dari vec1 dan vec2 }

similarity : float
length1 : int
length2 : int

```

ALGORITMA

```

length1 ← length_vec(hist1)
length2 ← length_vec(hist2)
if (length1 == 0 or length2 == 0) then
    → 0
else
    similarity ← dot(hist1,hist2)/(length1 * length2)
    → similarity

```

```
function dot(vec1 : array of float , vec2 : array of float) → float
{ Menghasilkan hasil product dari vec1 dan vec2 }
```

KAMUS LOKAL

```
dot_product : float
```

ALGORITMA

```
dot_product ← 0
i traversal [0...len(vec1)]
    dot_product ← dot_product + (vec1[i]*vec2[i])
→ dot_product
```

```
function length_vec(vec1 : array of float) → float
{Menghasilkan panjang dari vec1}
```

KAMUS LOKAL

```
length : float
```

ALGORITMA

```
length ← 0
i traversal [0...len(vec1)]
    length ← length + (vec1[i]^2)
→ length^(1/2)
```

```
function HSVBlockProcessor(image : , x_start : int , x_End : int ,
y_Start : int , y_End : int) → array of float
{ Menghasilkan Histogram dari H,S,dan V }
```

KAMUS LOKAL

```
function Hue(R : float,G : float,B : float) → float
{ Menghasilkan nilai Hue dari suatu pixel }

function Saturation(R : float,G : float,B : float) → float
{ Menghasilkan nilai Saturation dari suatu pixel }

function SVconv(sv : float) → int
{ Mengkuantifikasi nilai Saturation atau Value tergantung range tertentu}

function cmax(R : float,G : float,B : float) → float
{ Menghasilkan nilai terbesar antara R,G, dan B }

function Histogram_Calculation(HSV_Value,bins) → array of float
{ menghasilkan histogram dari suatu list nilai H,S, dan V }

function cv2.split(image) → array of binary
{ Menghasilkan nilai R,G,dan B dari suatu gambar }

R,G,B : array of binary
H,S,V : array of float
red,blue,green : float
```

ALGORITMA

```
R ← R/255.0
G ← G/255.0
B ← B/255.0
i traversal [y_Start...y_End]
    j traversal [x_Start...x_End]
        red ← R[i][j]
        green ← G[i][j]
        blue ← B[i][j]
        insertLast(H,Hue(red,green,blue)) {fungsi append}
        insertLast(S,Saturation(red,green,blue)) {fungsi append}
        insertLast(V,(SVconv(cmax(red, green, blue)))) {fungsi append}
Hist_H ← Histogram_Calculation(H,8)
Hist_S ← Histogram_Calculation(S,3)
Hist_V ← Histogram_Calculation(V,3)
→ Hist_H, Hist_S, Hist_V
```

```

function Histogram_Calculation(HSV_Value : array of float , bins :
           int) → array of float
{ menghasilkan histogram dari suatu list nilai H,S, dan V }

```

KAMUS LOKAL

```

histogram : array[0...bins] of float
sums_values : int

```

ALGORITMA

```

idx ← int(val)
if(idx >= bins) then
    idx = bins-1
if(0<=idx<bins) then
    histogram[idx] ← histogram[idx]+1
sums_values ← len(HSV_Value)
if(sums_values > 0) then
    histogram ← []
→ histogram

```

```

procedure DataSetHSVBlockProcess(input folder_path : string , input
           img_h : array of float , input img_s : array of float ,
           input img_v : array of float)
{I.S. dataFile dan dataSimilarity kosong, folderpath dan vec1
 terdefinisi}
{F.S. dataFile berisi data string pada folderpath dan
 dataSimilarity berisi hasil perhitungan Similiarities}

```

KAMUS LOKAL

```

function cv2.imread(filepath : string) → array of bytes
{ Mengembalikan gambar dalam bytes }

```

```

function cc.isinCache(filename : string, cachefile : string) →
    boolean
{ Mengecek apakah file bernama 'filename' terdapat pada file
  'cachefile', mengembalikan true jika benar dan false jika
  tidak }

function cc.readHSV(df[idx]) → array of float
{ membaca data yang ada di cache pada baris tertentu dan
  mengembalikan list dari h,s,dan v }

    procedure cc.write_to_file(input cachefile, input
      str(filename), input str(hist_h_list), input
      str(hist_s_list), input str(hist_v_list))
{ Menuliskan filename,hist_h_list,hist_s_list,dan hist_v_list
  ke dalam file cache 'cachefile' }

function cosine_similarity(hist1 : array of float, hist2 : array
  of float) → float
{ Menghasilkan hasil cosine similarity dari 2 vektor (hist1 dan
  hist2) }

cachefile : string
i, idx : int
img : array of binary

```

ALGORITMA

```

cachefile ← "Tubes_Algeo2_IF2110\Function\cacheHSV.csv"
i ← 0
f ← open(cachefile)
df ← read(f)
for sumthin sumthin
    idx ← -1
    img ← cv2.imread(os.path.join(folder_path, filename))
    idx ← idx = cc.isinCache(filename, cachefile)
    if (img != None) then
        if (idx != -1) then
            hist_h_list, hist_s_list, hist_v_list ←
            cc.readHSV(df[idx])
        else
            hist_h_list, hist_s_list, hist_v_list ←
            block_processing(img)

```

```

cc.write_to_file(cacheFile,str(filename),str(hist_h_list),
str(hist_s_list),str(hist_v_list))
    total_similarity ← 0
    i traversal [0...16]
        similarity_h ← cosine_similarity(img_h[i],
hist_h_list[i])
        similarity_s ← cosine_similarity(img_s[i],
hist_s_list[i])
        similarity_v ← cosine_similarity(img_v[i],
hist_v_list[i])
        block_similarity ← ((similarity_h + similarity_s +
similarity_v) / 3) * 100
        total_similarity ← total_similarity +
block_similarity

    similarity ← (total_similarity/4)
    insertLast(dataFile,filename) {fungsi append}
    insertLast(dataSimilarity,similarity) {fungsi append}

```

\

function cmax(R : float , G : float , B : float) → float

KAMUS LOKAL

maximum : float

ALGORITMA

maximum ← R <u>if</u> (R<G and B<G) <u>then</u> maximum ← G <u>else if</u> (G < B and R < B) <u>then</u> maximum ← B → maximum

```
function cmin(R : float , G : float , B : float) → float
```

KAMUS LOKAL

```
minimum : float
```

ALGORITMA

```
minimum ← R
if (R > G and B > G) then
    minimum ← G

else if (G > B and R > B) then
    minimum ← B

→ minimum
```

```
function Delta(R : float , G : float , B : float) → float
```

KAMUS LOKAL

```
function cmax(R : float,G : float,B : float) → float
{ Menghasilkan nilai terbesar antara R,G, dan B }
```

```
function cmin(R : float,G : float,B : float) → float
{ Menghasilkan nilai terkecil antara R,G, dan B }
```

```
Cmax,Cmin,delta : float
```

ALGORITMA

```
Cmax ← cmax(R,G,B)
Cmin ← cmin(R,G,B)
delta ← Cmax - Cmin
→ delta
```

```
function Saturation(R : float , G : float , B : float) → float
```

KAMUS LOKAL

```
function cmax(R : float,G : float,B : float) → float
{ Menghasilkan nilai terbesar antara R,G, dan B }
```

```
function Delta(R : float,G : float,B : float) → float
{ Menghasilkan selisih antara nilai terbesar dan terkecil
antara R,G, dan B }
```

```
function SVconv(sv : float) → int
{ Mengkuantifikasi nilai Saturation atau Value tergantung range
tertentu}
```

```
Cmax,delta : float
```

ALGORITMA

```
Cmax ← cmax(R,G,B)
delta ← Delta(R,G,B)
if (Cmax == 0) then
    s ← 0
else
    s ← delta/Cmax
    s ← SVconv(s)
→ s
```

```
function Hue(R : float , G : float , B : float) → float
```

KAMUS LOKAL

```
function Delta(R : float,G : float,B : float) → float
{ Menghasilkan selisih antara nilai terbesar dan terkecil
antara R,G, dan B }
```

```
function cmax(R : float,G : float,B : float) → float
{ Menghasilkan nilai terbesar antara R,G, dan B }
```

```
function Hconv(sv : float) → int
```

```
{ Menkuantifikasi nilai Hue tergantung range tertentu }
```

```
delta,max_val,result : float
```

ALGORITMA

```
delta ← Delta(R,G,B)
max_val ← cmax(R,G,B)
if (delta == 0) then
    → 0
else
    result ← 0
    if (max_val == R) then
        result ← 60*((G-B)/delta)%6
    else if (max_val == G) then
        result ← 60*((B-R)/delta)+2
    else
        result ← 60*((R-G)/delta)+4
    result ← Hconv(result)
    → result
```

```
function Hconv(h : float) → float
```

KAMUS LOKAL

ALGORITMA

```
depend on h
316<=h<=360 : h ← 0
1<=h<=25 : h ← 1
26<=h<=40 : h ← 2
41<=h<=120 : h ← 3
121<=h<=190 : h ← 4
191<=h<=270 : h ← 5
271<=h<=295 : h ← 6
296<=h<=315 : h ← 7
→ h
```

```
function SVconv(sv : float) → float
```

KAMUS LOKAL

ALGORITMA

```
if (0<=sv<0.2) then
    sv ← 0
else if (0.2<=sv<0.7) then
    sv ← 1
else
    sv ← 2
→ sv
```

```
function block_Calculation(image : array of binary) →
{ Menghasilkan list yang berisi index pixel-pixel dari setiap blok
}
```

KAMUS LOKAL

```
constant block = 4
px_Separation,py_Separation : array [0...block*block] of 0
height,width,k,py,px : int
```

ALGORITMA

```
px ← width/block
py ← height/block
k ← 0
i traversal [0...block]
    j traversal [0...block]
        px_Separation[k] ← px*(i+1)
        py_Separation[k] ← py * (j + 1)
        k ← k + 1
→ px_Separation, py_Separation
```

```
function block_processing(image : array of binary) →
```

KAMUS LOKAL

```
function block_calculation(image) → array of float
{ Menghasilkan list yang berisi index pixel-pixel dari setiap
  blok }

function HSVBlockProcessor(image, x_Start, x_End, y_Start,
  y_End) →
{ Menghasilkan Histogram dari H,S,dan V }

Hist_h_list,Hist_s_list,Hist_v_list : array of float
px_Separation,px_Separation : integer
```

ALGORITMA

```
px_Separation, py_Separation ← block_Calculation(image)
i traversal [0...4]
  j traversal [0...4]
    if (i == 0) then
      if (j ==0) then
        H,S,V← HSVBlockProcessor(image,0,px_Separation[i],0,py_Separation[j])
      else
        H,S,V ←
          HSVBlockProcessor(image,0,px_Separation[i],py_Separation[j-1],py_Sep
          aration[j])
      else
        if (j == 0) then
          H,S,V ←
            HSVBlockProcessor(image,px_Separation[i-1],px_Separation[i],0,py_Sep
            aration[j])
        else
          H,S,V←HSVBlockProcessor(image,px_Separation[i-1],px_Separation[i],py_S
          ession[j-1],py_Separation[j])
          insertLast(Hist_h_list,H) {fungsi append}
          insertLast(Hist_s_list,S) {fungsi append}
          insertLast(Hist_v_list,V) {fungsi append}
→ Hist_h_list,Hist_s_list,Hist_v_list
```

4.1.2 Tekstur

```
function Texture(filepath : string, folderpath : array of string) →  
array string of float
```

KAMUS LOKAL

```
function time.Time() → float  
{ Mendapatkan Waktu Saat dieksekusi }
```

```
function cv2.imread(filename : string) → array of bytes  
{ Mengembalikan gambar dalam bytes }
```

```
function calculate_texture_features(input_image : array of  
bytes) → array of float  
{ Mengekstrak feature GLCM Matrix dari suatu gambar}
```

```
procedure DataSetSimilarityProcess(input folderpath :  
string, input vec1 : float, output dataFile : array of string,  
output dataSimilarity : array of float)  
{ I.S. dataFile dan dataSimilarity kosong, folderpath dan vec1  
terdefinisi}  
{ F.S. dataFile berisi data string pada folderpath dan  
dataSimilarity berisi hasil perhitungan Similiarities}
```

```
function mergeSort(file : array of string, similarity : array  
of float) → array of string, array of float  
{ Mengembalikan hasil berupa array of string, integer yang  
merupakan data file yang di sort berdasarkan data similarities}
```

```
procedure(output )  
{ mengembalikan hasil }
```

ALGORITMA

```
start_time ← time.Time()  
input_image ← cv.imread(filepath)  
  
vec1 = calculate_texture_features(input_image)
```

```

DataSetSimilarityProcess(folderpath, vec1, dataFile,
dataSimilarity)

end_time ← time.time()
elapsed_time ← end_time - start_time
dataFile, dataSimilarity ← mergeSort(dataFile, dataSimilarity)

insertLast(dataFile, "Time") {fungsi append}
insertLast(dataSimilarity, elapsed_time) {fungsi append}
sim_dict ←
→ sim_dict

```

function greyscaleTransform(image : array of binary) → float

KAMUS LOKAL

ALGORITMA

```

height, width ← image.shape[:2]
bluePixels, greenPixels, redPixels = image[:, :, 0], image[:, :, 1], image[:, :, 2]

greyimage ← array[0...height] of array[0...width] of float(0)
greyimage ← 0.29*redPixels + 0.587*greenPixels +
0.114*bluePixels
→ greyimage

```

function calculate_texture_features(image : array of binary) → array of float

KAMUS LOKAL

function greyscaleTransform(image : array of byte) → float
{ }

```

function GLMCMATRIXPROCESSINGUNIT(mat : Matrix) → arrays of
float
    { Mengkonversi Matrix co occurence menjadi nilai contrast,
dissimilarity, homogeneity, asm, energy,dan entropy}

function CO_OCCURANCE_MAT(image : array of byte,angle : ) →
Matrix
    { Menghasilkan co occurence matrix yang telah dinormalisasikan
}

```

ALGORITMA

```

grey_image ← greyscaleTransform(image)

glcm_features ←
GLMCMATRIXPROCESSINGUNIT(co_occurrence_mat(grey_image,0))
glcm_features ← list of glcm_features
→ glcm_features

```

```

procedure DataSetSimilarityProcess(input folder_path : string
, input input_image_vec : array of float, input/output dataFile :
array of string, input/output dataSimilarity : array of string )
    {I.S. dataFile dan dataSimilarity kosong, folderpath dan vec1
terdefinisi}
    {F.S. dataFile berisi data string pada folderpath dan
dataSimilarity berisi hasil perhitungan Similiarities}

```

KAMUS LOKAL

```

function cv2.imread(filepath : string) → array of bytes
{ Mengembalikan gambar dalam bytes }

function cc.isInCache(filename : string,cachefile : string) →
boolean
{ Mengecek apakah file bernama 'filename' terdapat pada file
'cachefile', mengembalikan index nya, jika tidak ada index = -1 }

```

```

function cc.readHSV(df[idx]) → array of float
{ membaca data yang ada di cache pada baris tertentu dan
mengembalikan list dari h,s,dan v }

function calculate_texture_features(input_image : array of
bytes) → array of float
{ Mengekstrak feature GLCM Matrix dari suatu gambar}

procedure cc.write_list_to_file(input cachefile, input
str(filename), input str(hist_h_list), input str(hist_s_list),
input str(hist_v_list))
{ Menuliskan filename,hist_h_list,hist_s_list,dan hist_v_list
ke dalam file cache 'cachefile' }

function cosine_similarity(hist1 : array of float,hist2 : array
of float) → float
{ Menghasilkan hasil cosine similarity dari 2 vektor (hist1 dan
hist2) }

```

ALGORITMA

```

cache_file ← "cacheTextur.csv"
f ← open(cache_file)
df ← read(f)
read_folder
sumthin sumthin
sumthin
idx ← -1
img ← cv2.imread(os.path.join(folder_path, filename))
idx ← cc.isinCache(filename,cache_file)
if (img != None) then
    if (idx != -1) then
        vec2 ← cc.readHSV(df[idx])
    else
        vec2 ← calculate_texture_features(img)
    cc.write_list_to_file(cache_file,filename,vec2)
    similarity ← cosine_similarity(input_image_vec,vec2)
    if (similarity*100 > 60) then
        insertLast(dataFile,filename) {fungsi append}
        insertLast(dataSimilarity,similarity*100) {fungsi
append}

```

```
function merge(firstfile : array of float, lastfile : array of float , firstsim : array of float , lastsim : array of float) → array of float, array of float
```

KAMUS LOKAL

```
file,sim : array of float
```

ALGORITMA

```
i,j ← 0
file ← []
sim ← []
while (i < len(firstsim) and j < len(lastsim)) do
    if(firstsim[i] > lastsim[j]) then
        insertLast(sim,firstsim[i]) {fungsi append}
        insertLast(file,firstfile[i]) {fungsi append}
        i += 1
    else
        insertLast(sim,lastsim[j]) {fungsi append}
        insertLast(file,lastfile[j]) {fungsi append}
        j+=1
    while (i < len(firstsim)) do
        insertLast(sim,firstsim[i]) {fungsi append}
        insertLast(file,firstfile[i]) {fungsi append}
        i += 1
    while (j < len(lastsim)) do
        insertLast(sim,lastsim[j]) {fungsi append}
        insertLast(file,lastfile[j]) {fungsi append}
        j+=1
→ file,sim
```

```
function mergeSort(file : string , similarity : float) → array of float, array of float
```

KAMUS LOKAL

```
function merge(firstfile : array of float, lastfile : array of float , firstsim : array of float , lastsim : array of float) → array of float, array of float
```

```
{ Menggabungkan file dan similarity dan disort }
```

ALGORITMA

```
if(len(similarity) <= 1) then
    → file,similarity
middle ← len(similarity) div 2
firstfile,firstsim ←
mergesort(file[:middle],similarity[:middle])
    lastFile, lastSim ←
mergeSort(file[middle:],similarity[middle:])
    → merge(firstFile,lastFile,firstSim,lastSim)
```

```
function co_occurrence_mat(image : array of bytes, angle : float) →
Matrix
```

KAMUS LOKAL

```
constant pi = 3.14
```

```
function cos(sudut) → float
{Menghasilkan nilai cos dari sudut 'sudut'}
```

```
function sin(sudut) → float
{Menghasilkan nilai sin dari sudut 'sudut'}
```

```
function NormSymmetric(mat : Matrix) → Matrix
{Mengembalikan Matrix yang telah dinormalisasikan}
```

ALGORITMA

```
height,width ← image.shape[:2]
co_occurrence ← array [0...256] of array [0...256] of float(0)

sudut ← angle * (pi/180)
offset_x ← int(cos(sudut))
offset_y ← int(sin(sudut))

i traversal [0...height]
    j traversal [0...width]
        off_y ← i + offset_y
```

```

    off_x ← j + offset_x
    if (0 <= off_x < width and 0 <= off_y < height) then
        curr_val ← int(image[i,j])
        off_val ← int(image[off_y, off_x])
        co_occurrence[curr_val, off_val] += 1
    NormMatrix ← NormSymmetric(co_occurrence)
    → NormMatrix

```

function Tranpose(mat : Matrix) → Matrix

KAMUS LOKAL

ALGORITMA

```

matTrans ← array [0...len(mat[0])] of array [0...len(mat)] of
float(0)
    i traversal [0...len(mat)]
        j traversal [0...len(mat[0])]
            matTrans[i,j] = mat[j,i]
    → matTrans

```

function NormSymmetric(mat : Matrix) → Matrix

KAMUS LOKAL

function Tranpose(mat : Matrix) → Matrix
{Menghasilkan Matriks hasil transpose Matriks 'mat'}

ALGORITMA

```

mats ← Tranpose(mat)+mat
    i traversal len(mats)
        j traversal len(mats[0])
            sum ← sum + mats[i][j]
    → mats/sums

```

```
function DotAndLength(vec1 : array of float, vec2 : array of float)
→ float, float, float
{ Menghasilkan dot product 2 vektor serta panjang vektor untuk vec1
dan vec2 }
```

KAMUS LOKAL

```
dot_product, length_vec1, length_vec2 = 0 : float
```

ALGORITMA

```
i traversal[0..length(vec1)]
j traversal[0..length(vec2)]
dot_product ← dot_product + (vec1[i] * vec2[i])
length_vec1 ← length_vec1 + vec1[i]
length_vec2 ← length_vec2 + vec2[i]
length_vec1 = length_vec1 ^ (1/2)
length_vec2 = length_vec2 ^ (1/2)
→ dot_product, length_vec1, length_vec2
```

```
function cosine_similarity(vec1 : array of float, vec2 : array of
float) → float
{ Menghasilkan nilai data similarity berdasarkan kalkulasi 2 vector
}
```

KAMUS LOKAL

```
function DotAndLength(vec1 : array of float, vec2 : array of float)
→ float, float, float
{ Menghasilkan dot product 2 vektor serta panjang vektor untuk vec1
dan vec2 }

dot, l1, l2, similarity : float
```

ALGORITMA

```

dot,l1,l2 = DotAndLength(vec1,vec2)
similarity = dot/ (l1 * l2)
→ similarity

```

```

function GLCMMatrixProcessingUnit(mat : array[0..256] of
array[0..256] of float) → float, float, float, float, float
{ menghasilkan feature dari GLCM matrix seperti contrast,
dissimilarity, homogeneity, asm, energy dan entropy }

```

KAMUS LOKAL

```

function log(x : float) → float
{Menghasilkan nilai log dari x}

```

ALGORITMA

```

i traversal [0...len(mat)]
j traversal [0...len(mat[0])]
    px ← mat
    contrast ← contrast+(px * (i - j) ** 2)
    homogeneity ← homogeneity + (px / (1 + abs(i - j)))
    dissimilarity ← dissimilarity + (px * abs(i - j))
    asm ← asm + (px ** 2)
    if (px == 0) then
        entropy ← entropy - (px*log(1))
    else
        entropy ← entropy - (px*log(px))

energy ← asm ^ (1/2)
→ contrast, dissimilarity, homogeneity, asm, energy, entropy

```

4.2. Penjelasan Struktur Program dan Arsitektur Kode

No	Function	Penjelasan
TEXTURE		
1	Texture(filepath, folderpath)	Fungsi utama untuk CBIR Tekstur
2	greyscaleTransform(image)	Menghasilkan Matriks yang berisi nilai greyscale
3	calculate_texture_features(image):	Menghasilkan vektor yang berisi fitur dari GLCM matriks
4	DataSetSimilarityProcess(folder_path, input_image_vec, dataSimilarity)	Menghasilkan data similarity dan filename yang belum di sort pada pengujian input dengan dataset
5	merge(firstfile,lastfile,firstsim,lastsim)	Menghasilkan dataFilename dan data Similarity yang terurut dari gabungan pecahan 2 list
6	mergeSort(file, similarity)	Fungsi utama dari Merge menghasilkan dataFilename dan data similarity yang terurut berdasarkan value dari Similarity
7	co_occurrence_mat(image,angle)	Menghasilkan GLCM matriks dari matriks greyscale
8	Transpose(mat)	Menghasilkan Matriks Tranpose
9	GLCMMatrixProcessingUnit(mat)	Menghasilkan feature feature dari GLCM Matriks yang belum digabung
10	cosine_similarity(vec1,vec2)	Menghasilkan data Similarity dari 2 vektor
11	DotAndLength(vec1,vec2)	Menghasilkan dot product dari 2 vektor dan vektor satuan dari masing masing vektor
COLOR		
1	Color(filepath, folderpath)	Fungsi utama untuk CBIR Color
2	cosine_similarity(hist1,hist2)	Menghasilkan data similarity antara 2 histogram

No	Function	Penjelasan
TEXTURE		
1	Texture(filepath, folderpath)	Fungsi utama untuk CBIR Tekstur
2	greyscaleTransform(image)	Menghasilkan Matriks yang berisi nilai greyscale
3	dot(vec1,vec2)	Menghasilkan dot product dari 2 vektor
4	length_vec(vec1)	Menghasilkan panjang satuan matrix
5	HSVBlockProcessor(image, x_Start, x_End, y_Start, y_End)	Menghasilkan data Histogram dari HSV yang diextract dari suatu image
6	Histogram_Calculation(HSV_Value, bins)	Menghasilkan data Histogram dari suatu value antara H,S,V berdasarkan jumlah bins
7	DataSetHSVBlockProcess(folder_path, img_h, img_s, img_v, dataFile, dataSimilarity)	Melakukan kalkulasi dengan data Folder dataset dan menyimpan datanya di dataFile dan dataSimilarity
8	cmax(R,G,B)	Menghasilkan nilai tertinggi dari R,G,B pada suatu pixel
9	cmin(R,G,B)	Menghasilkan nilai terendah dari R,G,B pada suatu pixel
10	Delta(R,G,B)	Menghasilkan perbedaan atau selisih dari cmax dan cmin
11	Saturation(R,G,B)	Menghasilkan S dari HSV, atau saturation
12	Hue(R,G,B)	Menghasilkan H dari HSV atau Hue
13	Hconv(h)	Menghasilkan kuantifikasi dari Hue
14	SVconv(sv)	Menghasilkan kuantifikasi dari Saturation dan Value
15	block_Calculation(image)	Melakukan pembagian block berdasarkan pembagian pixel, untuk 4x4 maka setidaknya terdapat 16 block
16	block_processing(image)	Menghasilkan list histogram untuk seluruh block akan dihasilkan 4 list yang merupakan list buatan dari seluruh block yang ada
FRONTEND FILE		
1	AboutUs.jsx	Menghasilkan page untuk About us terletak pada page paling bawah

No	Function	Penjelasan
TEXTURE		
1	Texture(filepath, folderpath)	Fungsi utama untuk CBIR Tekstur
2	greyscaleTransform(image)	Menghasilkan Matriks yang berisi nilai greyscale
2	api.js	Database untuk Component API yang digunakan di beberapa file berbeda
3.	Camera.jsx	Camera ini tidak hanya berisikan komponen untuk Camera namun Image Scrapping juga
4.	Modak.jsx	Komponen yang memegang data atau design untuk Camera, Modal ini terletak pada Camera.jsx
5	ModalScrapping.jsx	Komponen yang memegang data atau design untuk Image Scrapping, Modal Scrapping ini terletak pula pada Camera.jsx
6	DatasetUpload	Komponen yang bertanggungjawab pada input dataset sekaligus sebagai holder untuk pagination
7	Hero.jsx	Komponen yang memegang design untuk home atau design dari UI ketika web pertama kali dibuka
8.	HowToUse.jsx	Komponen yang memegang design dan data mengenai cara pemakaian website
9	Information.jsx	Komponen yang memegang design dan data mengenai Informasi mengenai program seperti CBIR dan implementasiannya pada masalah terkait
10	Navbar.jsx	Komponen yang memegang Navbar pada bagian atas
11	Page.jsx	Page ini fokus pada dan fungsi dari Camera. Page ini tersimpan pada Modal
12	PageScrapping.jsx	PageScrapping ini fokus pada design serta implementasi untuk imagescrapper, PageScrapping ini tersimpan atau berada di dalam ModalScrapping
13	Pagination.jsx	Pagination ini berfungsi pada design untuk pagination nya itu sendiri termasuk implementasi dari buttonnya. Pagination ini akan melakukan mapping data dan menampilkan hasil foto dan similarity nya dengan input yang diberikan
14	Reverse	Komponen yang berisi banyak komponen lain. Komponen ini yang berfungsi dalam transmisi logic untuk seluruh data termasuk penggunaan input_image

No	Function	Penjelasan
TEXTURE		
1	Texture(filepath, folderpath)	Fungsi utama untuk CBIR Tekstur
2	greyscaleTransform(image)	Menghasilkan Matriks yang berisi nilai greyscale dengan database. Pada komponen ini terdapat DatasetUpload.jsx dan Camera.jsx
15	App.js	Komponen utama yang akan ditampilkan pada browser
16	index.css	Data css yang berisikan custom data yang dibuat
17	index.js	Sama halnya dengan data css yang berisikan custom data atau komponen yang dibuat
18	package.json	Digunakan untuk menyimpan dependencies yang akan digunakan lainnya
19	Sisanya komponen pendukung seperti tailwind.config.js untuk theme dan design, lalu tubes2.js seperti app.js.	Tailwind.config = pengesetan theme dan style custom
FRONTEND FOLDER		
1	node_modules	Modules node, karena menggunakan typescript dan react
2.	public	Tidak terlalu digunakan disini
3	src	Folder yang berisi seluruh component dan asset
4	src/assets	Folder yang berisi aset-aset yang digunakan pada Frontend (design)
5	src/components	Folder yang berisi komponen komponen yang digunakan pada web
6	src/Poppins	Folder yang berisi font poppins
BACKEND FILE		
1	cache.py	Melakukan caching untuk texture cbir dan color cbir
2	cacheHSV.csv	Container untuk caching hasil CBIR color
3	cacheTexture.csv	Container untuk caching hasil CBIR texture

No	Function	Penjelasan
TEXTURE		
1	Texture(filepath, folderpath)	Fungsi utama untuk CBIR Tekstur
2	greyscaleTransform(image)	Menghasilkan Matriks yang berisi nilai greyscale
4	main.py	Fungsi utama Backend yang berisi seluruh function untuk Fast API
5	requirement.txt	Berisi seluruh dependencies untuk backend, harus di-download jika ingin run program
6	Texture_CBIR.py	Berisi Seluruh function dan procedures untuk memproses gambar dengan dataset untuk mendapatkan data similaritynya berdasarkan texture
7	Color_CBIR.py	Berisi seluruh function dan procedures untuk memproses gambar dengan dataset untuk mendapatkan data similarity nya berdasarkan warna
BACKEND FOLDER		
1	datasets	Folder yang akan menjadi container sementara untuk file dataset yang diupload
2	input_images	Folder yang akan menjadi container sementara untuk file input_image yang diupload

4.3. Penjelasan Tata Cara Penggunaan

Set Up Back End

1. Download semua file dari github
2. Python yang digunakan harus versi > 8 dan < 12 untuk mengatasi install wheel
3. Buatlah virtual environment dalam python
4. Buka virtual environment pada back end
5. Download semua keperluan dengan command ‘pip install -r requirement.txt’
6. Run back end server dengan command ‘uvicorn main:app –reload’

Set Up Front End

1. Download npm untuk python (apabila belum)
2. Npm install untuk install seluruh dependencies
3. Jalankan ‘npm run start’ untuk memulai web
4. Pastikan backend juga sudah berjalan
5. Selamat mencoba

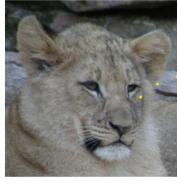
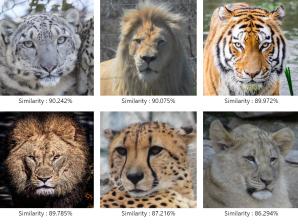
Penggunaan Website

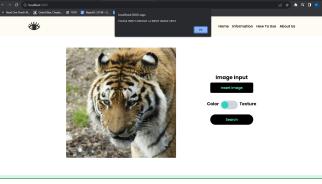
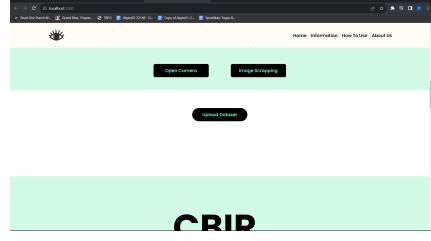
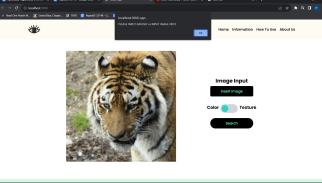
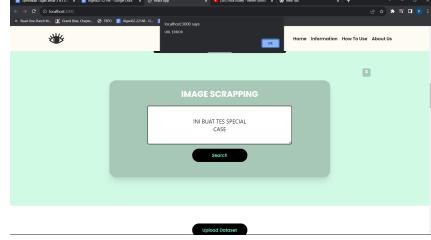
1. Input Image yang ingin dibandingkan bisa melalui file dalam komputer, dengan image scrapping atau camera. Tidak dapat menginput dataset sebelum menginput image awal.
2. Apabila menggunakan kamera, kamera akan secara otomatis menangkap gambar 5,menggantikan input sebelumnya dan langsung dihitung similaritynya. Apabila dataset belum diberikan, similarity tidak akan dihitung, dan tidak akan ada output. Namun kamera akan tetap menangkap gambar.
3. Untuk image scraping, tinggal masukkan url dari gambar yang ingin dibandingkan lalu tekan tombol search untuk mencari gambar tersebut dari internet sekaligus mulai menghitung similaritynya apabila dataset sudah diberikan. Apabila url mengarah ke file selain jpg,jpeg, dan png, maka input tidak akan diterima.
4. Dataset berupa folder yang berisi file jpg.jpeg atau png. File bentuk lain tidak akan terbaca sebagai input dataset. Dataset hanya dapat menerima 1000 gambar, semua gambar yang melebihi 1000 tidak akan terbaca dan tidak akan dimasukkan ke dalam perhitungan.
5. Apabila dataset diganti namun dibatalkan atau foldernya tidak sesuai dengan prasyarat, perhitungan akan dilanjutkan menggunakan dataset sebelumnya.
6. Perbandingan dapat dilakukan dengan CBIR color dan tekstur. Pilihan tersebut dapat diubah dengan mengklik tombol di tengah tulisan *colour* dan *texture*.
7. Tekan tombol search untuk memulai perbandingan. Apabila tombol search ditekan tanpa menginput dataset, tidak akan ada perhitungan dan output tidak akan keluar.
8. Output berupa gambar dari dataset yang memiliki kemiripan lebih dari 60% dengan gambar awal dan berurut dari yang paling mirip.

4.4 Hasil Pengujian dan screenshot UI Program

4.4.1 Hasil Pengujian

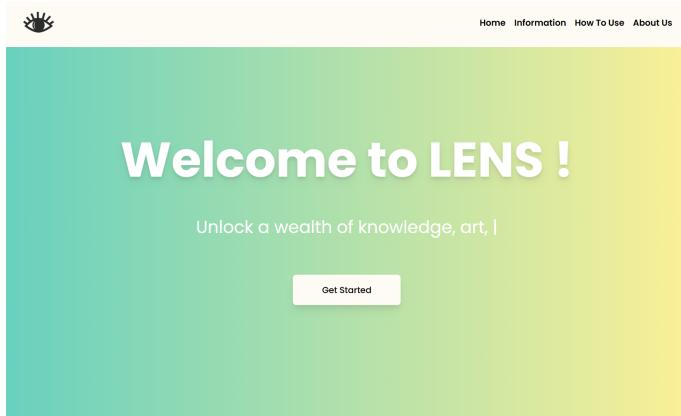
NORMAL CASES

1	COLOR CBIR	 <p>Image Input <input type="button" value="Insert image"/> <input checked="" type="radio"/> Color <input type="radio"/> Texture <input type="button" value="Search"/></p>	<p>Files upload: 13 Result : 13 images Time Taken: 0.202</p>  <table border="1"> <thead> <tr> <th>Similarity</th> </tr> </thead> <tbody> <tr><td>100%</td></tr> <tr><td>95.846%</td></tr> <tr><td>96.782%</td></tr> <tr><td>95.559%</td></tr> <tr><td>95.145%</td></tr> <tr><td>93.474%</td></tr> </tbody> </table>	Similarity	100%	95.846%	96.782%	95.559%	95.145%	93.474%
Similarity										
100%										
95.846%										
96.782%										
95.559%										
95.145%										
93.474%										
2	TEXTURE CBIR	 <p>Image Input <input type="button" value="Insert image"/> <input type="radio"/> Color <input checked="" type="radio"/> Texture <input type="button" value="Search"/></p>	<p>Files upload: 13 Result : 13 images Time Taken: 1.701</p>  <table border="1"> <thead> <tr> <th>Similarity</th> </tr> </thead> <tbody> <tr><td>99.999%</td></tr> <tr><td>99.999%</td></tr> <tr><td>99.999%</td></tr> <tr><td>99.992%</td></tr> <tr><td>99.932%</td></tr> <tr><td>99.895%</td></tr> </tbody> </table>	Similarity	99.999%	99.999%	99.999%	99.992%	99.932%	99.895%
Similarity										
99.999%										
99.999%										
99.999%										
99.992%										
99.932%										
99.895%										
3	CAMERA	<p>Camera</p>  <p>Result</p>	<p>Files upload: 13 Result : 13 images Time Taken: 0.080</p>  <table border="1"> <thead> <tr> <th>Similarity</th> </tr> </thead> <tbody> <tr><td>94.704%</td></tr> <tr><td>93.288%</td></tr> <tr><td>92.878%</td></tr> <tr><td>91.928%</td></tr> <tr><td>91.57%</td></tr> <tr><td>91.138%</td></tr> </tbody> </table>	Similarity	94.704%	93.288%	92.878%	91.928%	91.57%	91.138%
Similarity										
94.704%										
93.288%										
92.878%										
91.928%										
91.57%										
91.138%										
4	IMG SCRAPPER GAMBAR BEBEK	<p>IMAGE SCRAPPING</p> <pre>grin/purple-hand-drawn-cartoon-brown-line-duck-clipart-png-image_5529215.jpg</pre> <p><input type="button" value="Search"/></p>	<p>Files upload: 13 Result : 13 images Time Taken: 0.540</p>  <table border="1"> <thead> <tr> <th>Similarity</th> </tr> </thead> <tbody> <tr><td>90.242%</td></tr> <tr><td>90.079%</td></tr> <tr><td>89.972%</td></tr> <tr><td>89.769%</td></tr> <tr><td>87.216%</td></tr> <tr><td>86.294%</td></tr> </tbody> </table>	Similarity	90.242%	90.079%	89.972%	89.769%	87.216%	86.294%
Similarity										
90.242%										
90.079%										
89.972%										
89.769%										
87.216%										
86.294%										
SPECIAL CASES										

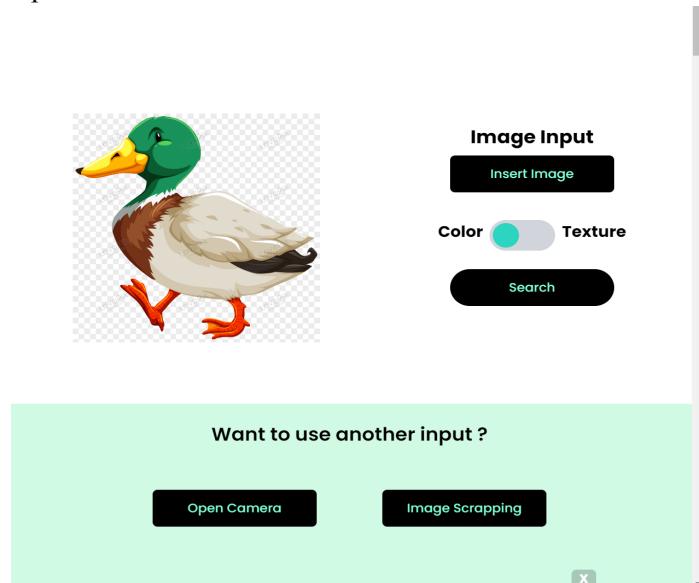
1	DATASET EMPTY FOLDER		
2	IMAGE SCRAPING NON URL		

4.4.2 Screenshot UI

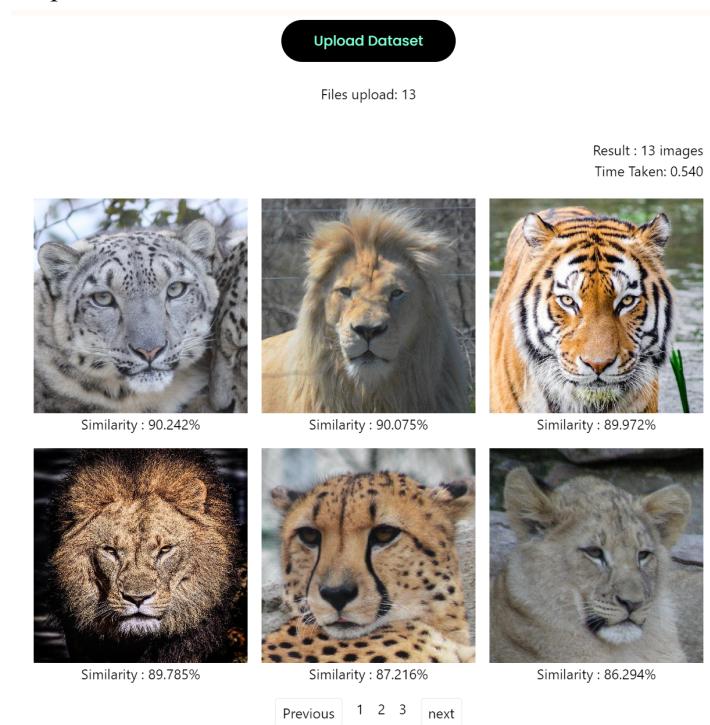
1. Entrance



2. Input



3. Output



4. Penjelasan Sistem

CBIR.

Our program implements an image retrieval system utilizing Vector Algebra. This approach is crucial in the realm of data processing and information retrieval. In this context, vector algebra is employed to represent and analyze data using a content-based classification approach, known as Content-Based Image Retrieval (CBIR).

Content-Based Image Retrieval (CBIR) is a process used to search for and retrieve images based on their content. The process begins with the extraction of important features from the images, such as color, texture, and shape. After extracting these features, they are represented in the form of vectors or numerical descriptions that can be compared with other images. Subsequently, CBIR employs matching algorithms to compare the feature vectors of the sought-after image with those of images in the dataset. The results of this matching are used to rank the images in the dataset and display the ones most similar to the queried image.

CBIR facilitates efficient access and exploration of image collections for users, as it doesn't rely on text or keyword searches but rather on the similarity of visual content values among the images. There are two popular parameters in CBIR:

1

CBIR with color parameter

In this CBIR approach, the input image is compared with images in the dataset by converting RGB images into a more common color histogram method. A color histogram represents the frequency of different colors in a specific color space, allowing for the distribution of colors in the image to be analyzed.

2

CBIR with texture parameter

CBIR with a texture comparison involves using a matrix known as a co-occurrence matrix. This matrix is chosen for its ease and speed of processing, producing smaller-sized vectors. For instance, considering an image I with $n \times m$ pixels and a parameter offset $(\Delta x, \Delta y)$, the co-occurrence matrix is employed for texture analysis.

5. How to use

HOW TO USE.

Step 1

Drop the image dataset in the form of a folder containing a collection of images. This image dataset is needed before the searching process so that there is a comparison for the image you want to search for.

Step 2

After the dataset is entered, insert an image that you want to search from the dataset.

Step 3

Select a search option, by color or texture. Then, press the search button. The program will then process, searching for images from the dataset that have similarities to the image entered earlier.

Step 4

The program will then process, searching for images from the dataset that have similarities to the image entered earlier.

Step 5

The program will display a collection of similar images, sorted from the highest to the lowest. The similarity percentage and the program's execution time will also appear.

6. About Us

ABOUT US.



Davis

A sophomore IT student at ITB with the id of 13522157. currently living in Bandung. I have a wide range of interests, including Web Development, Game Development, Mobile Applications, and Cybersecurity. Currently, the main focus is on mastering Game Development and refining skills in Web Development.



Chika

Drop the image dataset in the form of a folder containing a collection of images. This image dataset is needed before the searching process so that there is a comparison for the image you want to search for.



Rafa

Drop the image dataset in the form of a folder containing a collection of images. This image dataset is needed before the searching process so that there is a comparison for the image you want to search for.

BAB 5 KESIMPULAN

5. 1. Kesimpulan

Dalam proyek Tugas Besar 2 IF2123 Aljabar Linier dan Geometri ini, kami telah mengimplementasikan sistem temu balik gambar dengan memanfaatkan Aljabar Vektor. Implementasi ini berhasil diintegrasikan ke dalam sebuah aplikasi berbasis *website*. Pada tugas besar ini, pertama-tama digunakan sebuah folder dataset yang nantinya diubah menjadi sekumpulan matriks. Melalui vektor dari kedua gambar yang dibandingkan, dapat dicari perhitungan cosine similarity dari kedua gambar dengan menggunakan vektor dari kedua gambarnya. Gambar dengan nilai cosine similarity yang lebih tinggi memiliki tingkat kemiripan yang lebih tinggi. Dengan demikian, penulis menyimpulkan bahwa Tugas Besar 2 IF2123 Aljabar Linier dan Geometri ini dapat dibuat sebuah algoritma sistem temu balik gambar melalui metode Aljabar Vektor.

5.2 Saran

Pelaksanaan Tugas Besar 2 IF2123 Aljabar Linier dan Geometri di Semester I Tahun 2022/2023 merupakan pengalaman yang sangat berharga bagi penulis. Dari pengalaman ini, penulis ingin berbagi beberapa saran kepada pembaca yang mungkin akan menghadapi tugas serupa di masa depan:

1. Pemahaman dalam pengembangan *website*: Tugas ini mengharuskan penulis untuk mengimplementasikan program dalam sebuah *website*, yang mungkin belum familiar bagi sebagian mahasiswa. Penulis sangat menyarankan untuk meluangkan waktu yang cukup guna mengeksplor dan mempelajari proses pengembangan web, terutama dalam mempelajari ketiga bahasa penting dalam pengembangan web, yaitu HTML, CSS, dan Javascript.
2. Kerja Sama Tim: Efektivitas dalam kerja sama tim memiliki peran penting dalam menyelesaikan tugas ini. Kolaborasi melalui alat seperti VSCode LiveShare untuk pembuatan program dan kolaborasi langsung pada MS Word untuk pembuatan laporan bisa sangat membantu. Selain itu, dalam pengembangan source code, konflik antara anggota tim dapat terjadi. Oleh karena itu, menggunakan alat pengelola versi seperti Github sangat direkomendasikan agar mempermudah manajemen proyek secara asinkron.
3. Pengolahan Dataset: Memperhitungkan fakta bahwa program harus menangani sejumlah besar dan kompleks dataset, perlu dilakukan optimasi pada program dan penggunaan library telah tersedia. Library yang sudah tersedia tersebut telah dioptimalkan untuk algoritma yang diimplementasikan, sehingga dapat meningkatkan kecepatan pelaksanaan program secara signifikan.

Semoga saran-saran ini membantu pembaca dalam menyiapkan diri untuk menangani tugas serupa di masa depan.

5.2 Refleksi

Tugas Besar 2 IF2123 Aljabar Linier dan Geometri di Semester I Tahun 2023/2024 adalah salah satu tantangan awal yang kami hadapi pada semester ini. Proses penyelesaian tugas ini tidak datang tanpa hambatan. Melalui tugas ini, penulis semakin menghargai penggunaan vektor dan penerapannya dalam berbagai aspek kehidupan, terutama dalam konteks matematika dan teknologi. Sifat vektor dan atribut-atributnya memiliki peran penting dalam menyelesaikan masalah, salah satunya dalam sistem aplikasi temu balik gambar ini.

Dalam perjalanan penulis mengerjakan tugas ini, penulis juga merasakan kegembiraan dalam mempelajari banyak hal mengenai proses pengembangan web. Penulis menghadapi tantangan dalam proses pengintegrasian program ke dalam sebuah web selama beberapa minggu ini, tetapi berkat kerja sama tim yang erat dan kontribusi aktif setiap anggota kelompok, penulis berhasil mengatasi rintangan tersebut. Setiap anggota kelompok saling mendukung dan menyelesaikan tugas mereka dengan baik, menciptakan sinergi yang efektif dalam menyelesaikan Tugas Besar ini.

5.2 Ruang Perbaikan dan Pengembangan

Dalam pengembangan sistem aplikasi temu balik gambar ini, terdapat beberapa ruang perbaikan dan pengembangan yang dapat dieksplorasi. Pertama-tama, perlu dipertimbangkan untuk meningkatkan efisiensi algoritma pencarian gambar agar dapat merespons dengan lebih cepat. Selain itu, saat mempertimbangkan peningkatan pengalaman pengguna, dapat ditambahkan fitur-fitur interaktif pada *website* yang memperkaya penggunaan aplikasi. Peningkatan skalabilitas sistem juga menjadi fokus penting, memastikan aplikasi mampu menangani volume gambar yang semakin besar.

Integrasi teknologi kecerdasan buatan, seperti model deep learning atau machine learning, dapat menjadi langkah berikutnya untuk meningkatkan akurasi temu balik gambar. Selain itu, ruang perbaikan mencakup penanganan dataset terdistribusi dengan efisiensi, kustomisasi fitur dan pengaturan untuk memberikan fleksibilitas lebih kepada pengguna, serta implementasi pemantauan kinerja untuk mendapatkan wawasan yang lebih baik. Dengan merinci dan memprioritaskan ruang perbaikan ini, diharapkan sistem aplikasi temu balik gambar dapat berkembang menjadi solusi yang lebih responsif dan dapat memenuhi kebutuhan pengguna secara efektif.

DAFTAR PUSTAKA

Rinaldi Munir. "Aljabar dan Geometri untuk Informatika 2023/2024."

informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/algeo23-24.html

As'ad Shidqy Aziza,* , Firnanda Al Islama Achyunda Putra. "Effect of Features and Angle on GLCM Feature Extraction on Accuracy for Object Classification "

<https://ejurnal.unikama.ac.id/index.php/jst/article/download/7627/3698/23281>

Pengambilan 45 derajat sebagai angle untuk GLCM Matriks

"RGB to HSV color conversion"

<https://math.stackexchange.com/questions/556341/rgb-to-hsv-color-conversion-algorithm>

Proses konversi warna RGB ke HSV.

"Content-based image retrieval using color and texture fused features"

<https://www.sciencedirect.com/science/article/pii/S0895717710005352>

Penjelasan lebih jauh mengenai CBIR tekstur dan warna.

"Feature Extraction : *Gray Level Co-occurrence Matrix (GLCM)*"

<https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcma10c45b6d46a1>

Cara membuat GLCM

"Fastapi setup" : Setup untuk fastapi

<https://fastapi.tiangolo.com/tutorial/first-steps/>

"Tailwind CSS Gradient Generator" : Tools untuk membuat warna gradient dengan Tailwind CSS

[Tailwind CSS Gradient Generator \(tailwindcomponents.com\)](https://tailwindcomponents.com)