

LAPORAN TUGAS BESAR II
IF2123 ALJABAR LINEAR DAN GEOMETRI
Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar



Disusun oleh :
Christian Justin Hendrawan (13522135)
Muhammad Fatihul Irhab (13522143)
M. Zaidan sa'dun R. (13522146)

**Program Studi Teknik Informatika
Institut Teknologi Bandung
2023**

Daftar Isi

Daftar Isi.....	2
Bab I.....	3
Bab II.....	4
2.1. Citra.....	4
2.2. Citra Digital.....	5
2.3. Warna.....	6
2.4. Tekstur.....	9
2.5. Content Based Image Retrieval (CBIR).....	10
2.5.1 CBIR dengan parameter warna.....	11
2.5.2 CBIR dengan parameter tekstur.....	13
2.6. Website.....	16
Bab III.....	18
3.1. Langkah-Langkah Pemecahan Masalah.....	18
3.2. Proses pemetaan masalah menjadi elemen-elemen pada aljabar geometri.....	21
3.3. Contoh ilustrasi kasus dan penyelesaiannya.....	23
Bab IV.....	27
4.1. Implementasi program utama.....	27
4.2. Penjelasan struktur program.....	47
4.3. Tata cara penggunaan program.....	50
4.4. Hasil Pengujian.....	51
4.5. Analisis dari desain solusi.....	57
Bab V.....	59
I. Kesimpulan.....	59
II. Saran.....	59
III. Komentar atau Tanggapan.....	59
IV. Refleksi.....	60
V. Ruang Perbaikan atau Pengembangan.....	60
Bab VI.....	61

Bab I

Deskripsi Masalah

Dalam era digital, jumlah gambar yang diproduksi dan disimpan mengalami lonjakan pesat, termasuk dalam ranah pribadi dan profesional. Keanekaragaman ini mencakup segala jenis gambar, mulai dari foto pribadi, gambar medis, ilustrasi ilmiah, hingga gambar komersial. Meskipun gambar-gambar ini berasal dari berbagai sumber dan memiliki ragam jenis, sistem temu kembali gambar menjadi semakin relevan dan penting dalam menghadapi tantangan ini. Dengan bantuan sistem ini, pengguna dapat dengan mudah mencari, mengakses, dan mengelola koleksi gambar mereka. Sistem ini memungkinkan mereka untuk menjelajahi informasi visual yang tersimpan di berbagai platform, mulai dari pencarian gambar pribadi hingga analisis gambar medis untuk diagnosis, atau mencari ilustrasi ilmiah dan produk berdasarkan gambar komersial. Salah satu contoh penerapan sistem temu balik gambar yang mungkin familiar bagi banyak orang adalah Google Lens.

Content-Based Image Retrieval (CBIR) merupakan proses pencarian dan pengambilan gambar berdasarkan konten visualnya. Tahapan awalnya melibatkan ekstraksi fitur-fitur utama dari gambar seperti warna, tekstur, dan bentuk. Fitur-fitur ini kemudian diubah menjadi vektor atau deskripsi numerik yang bisa dibandingkan dengan gambar lain. CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan ini digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling mirip dengan gambar yang dicari. Proses CBIR membantu pengguna dalam mengakses dan mengeksplorasi koleksi gambar secara lebih efisien karena tidak memerlukan pencarian berdasarkan teks atau kata kunci, melainkan berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut. Dalam Tugas Besar ini, kami harus mengimplementasikan dua parameter CBIR yang paling populer, yaitu CBIR berdasarkan warna dan CBIR berdasarkan tekstur.

Bab II

Landasan Teori

2.1. Citra

Secara sederhana, citra adalah representasi visual dari objek dalam bidang dua dimensi. Dalam konteks matematika, citra dapat dianggap sebagai fungsi yang terus menerus menggambarkan intensitas cahaya pada bidang dua dimensi. Proses terjadi saat sumber cahaya memancar dan menerangi objek, lalu objek memantulkan sebagian cahaya tersebut. Alat optik seperti mata manusia, kamera, atau pemindai menangkap pantulan cahaya ini untuk merekam citra objek. Citra didefinisikan sebagai fungsi intensitas cahaya dua dimensi yang bergantung pada koordinat spasial x dan y . Nilai dari fungsi pada titik tertentu mencerminkan tingkat kecerahan citra pada titik tersebut.

$$f(x, y)$$

(x, y) : koordinat pada bidang dwimatra

$f(x, y)$: intensitas cahaya (*brightness*) pada titik (x, y)

Gambar 1. Fungsi dwimatra yang menyatakan intensitas cahaya pada bidang dwimatra.

Gambarnya yang dihasilkan oleh sistem perekaman sinyal bisa berwujud:

1. Optik, contohnya adalah foto.
2. Analog, seperti visual pada layar televisi.
3. Digital, bisa disimpan langsung di disk atau pita magnetik.

Citra dapat dibagi menjadi dua jenis utama: citra diam (still image) dan citra bergerak (moving image). Citra diam adalah gambar yang tetap, sedangkan citra bergerak adalah serangkaian gambar yang ditampilkan secara berurutan sehingga menciptakan kesan gerakan bagi mata. Setiap gambar dalam rangkaian ini disebut frame, dan film atau tayangan televisi sebenarnya terdiri dari banyak frame, bisa mencapai ratusan hingga ribuan.



Gambar 2. Contoh citra diam

Selain itu, citra juga dapat dibagi menjadi dua kategori: citra yang terlihat seperti foto, gambar, atau tampilan layar monitor/televisi, hologram, dan sejenisnya. Sedangkan, citra yang tidak terlihat adalah data gambar/foto yang tersimpan dalam file atau representasi citra dalam bentuk fungsi matematis.

2.2. Citra Digital

Citra digital mengacu pada pemrosesan gambar dua dimensi dengan menggunakan komputer. Definisi yang lebih luas tentang pemrosesan citra digital juga mencakup semua data dua dimensi. Citra digital terdiri dari kumpulan bilangan nyata atau kompleks yang direpresentasikan oleh bit-bit individual. Unit terkecil dari data digital adalah bit, yang terdiri dari angka biner 0 atau 1. Sebuah kumpulan data sebanyak 8 bit membentuk sebuah unit data yang disebut byte, dengan rentang nilai dari 0 hingga 255. Dalam konteks citra digital, nilai intensitas energi direpresentasikan dalam satuan byte. Sebuah kumpulan byte dengan struktur tertentu dapat dibaca oleh perangkat lunak dan disebut sebagai citra digital 8 bit.

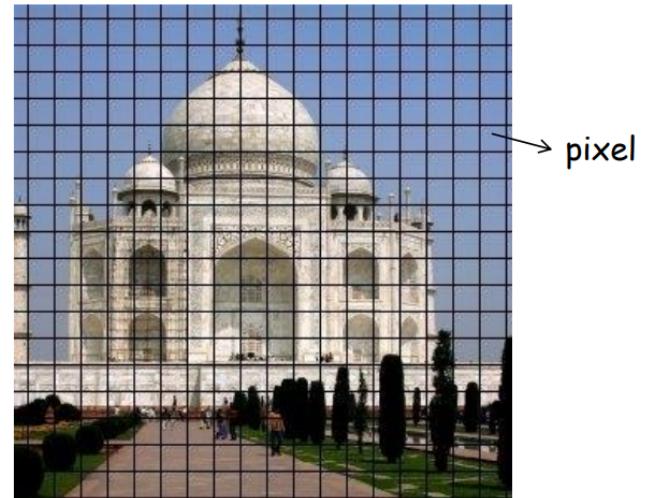
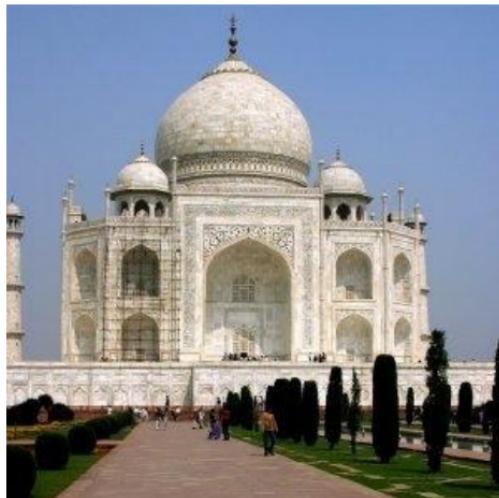
Citra digital dapat didefinisikan sebagai fungsi dua variabel $f(x,y)$, di mana x dan y menunjukkan koordinat spasial, dan nilai $f(x,y)$ merepresentasikan intensitas citra pada koordinat tersebut, seperti yang diilustrasikan pada gambar 3.

Teknologi dasar untuk menciptakan dan menampilkan warna dalam citra digital didasarkan pada penelitian bahwa warna merupakan gabungan dari tiga warna dasar, yaitu merah, hijau, dan biru (Red, Green, Blue - RGB).

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0, N-1) \\ f(1,0) & f(1,1) & \dots & f(1, N-1) \\ \dots & \dots & \dots & \dots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1, N-1) \end{bmatrix}$$

Gambar 3. Citra digital direpresentasikan sebagai matriks berukuran M x N

Citra dengan resolusi 1200 x 1500 berarti memiliki 1200×1500 pixel = 1.800.000 pixel



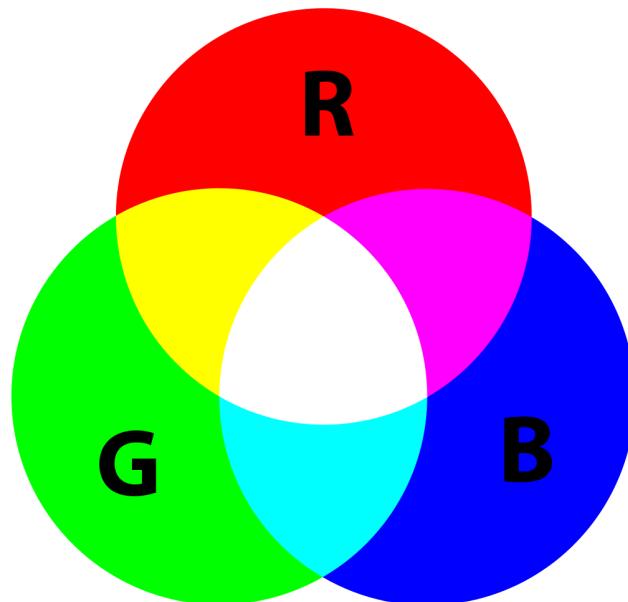
Gambar 4. Citra digital direpresentasikan dengan pixel

2.3. Warna

Isinya sebuah gambar digital terdiri dari piksel atau blok warna. Manusia mampu melihat panjang gelombang 400 hingga 700 nanometer dari radiasi elektromagnetik sebagai berbagai warna. Hewan juga memiliki kemampuan untuk melihat berbagai warna dari spektrum elektromagnetik yang berbeda dari apa yang dapat dilihat oleh manusia. Pengalaman alami dalam pengenalan warna melibatkan kerja sama antara mata dan otak.

Mata berperan sebagai alat penerima cahaya sementara otak mengolah informasi visual dari mata menjadi pengertian akan warna. Penglihatan manusia didasarkan pada tiga penerima: satu untuk warna merah, satu untuk hijau, dan satu lagi untuk biru. Ada banyak representasi warna dengan cakupan yang berbeda, dimana model umumnya memiliki tiga hingga empat saluran atau kanal yang merepresentasikan warna-warna tersebut.

Dalam gambar digital, setiap piksel mewakili warna dengan kombinasi tiga warna primer: merah, hijau, dan biru. Jumlah nilai yang mewakili masing-masing warna bergantung pada jumlah bit yang diperlukan untuk merepresentasikan warna tersebut. Semakin tinggi jumlah bit yang digunakan, semakin besar rentang nilai yang dapat digunakan untuk mewakili warna merah, hijau, dan biru.



Gambar 5. Model warna RGB

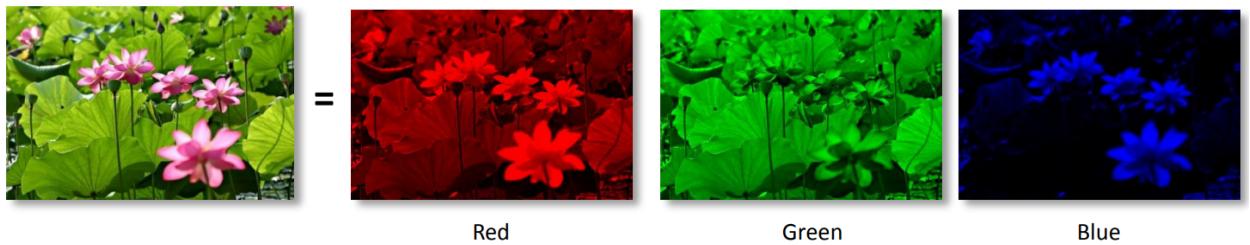
Sebuah model warna adalah model matematis abstrak yang menggambarkan cara warna dapat diwakili sebagai tupel angka, umumnya sebagai tiga atau empat nilai atau komponen warna. Terdapat 3 model warna yang akan dibahas pada bab ini, yakni model warna RGB, HSV, dan Grayscale.

2.3.1. Model warna RGB

Model warna RGB sering disebut sebagai sistem warna yang bersifat aditif karena menggunakan cahaya untuk menghasilkan warna. Banyak perangkat yang mengadopsi model warna RGB ini, seperti mata manusia, proyektor, televisi, kamera video, kamera digital, serta perangkat lain yang mengeluarkan cahaya. Warna

dihasilkan dengan mencampur ketiga warna dasar tersebut. Setiap warna memiliki skala intensitasnya yang berkisar dari 0 hingga 255.

Ketika nilai intensitas merah, hijau, dan biru masing-masing adalah 255, maka akan terbentuk warna putih. Sebaliknya, jika ketiga warna tersebut memiliki intensitas 0, akan menghasilkan warna hitam, serupa dengan suasana ruangan gelap tanpa cahaya yang hanya tampak hitam. Fenomena ini dapat terlihat ketika menonton film di bioskop lama yang menggunakan proyektor dengan tiga warna terpisah; ketika adegan menampilkan ruangan gelap, cahaya yang dipancarkan dari ketiga lubang proyektor tersebut berkurang.



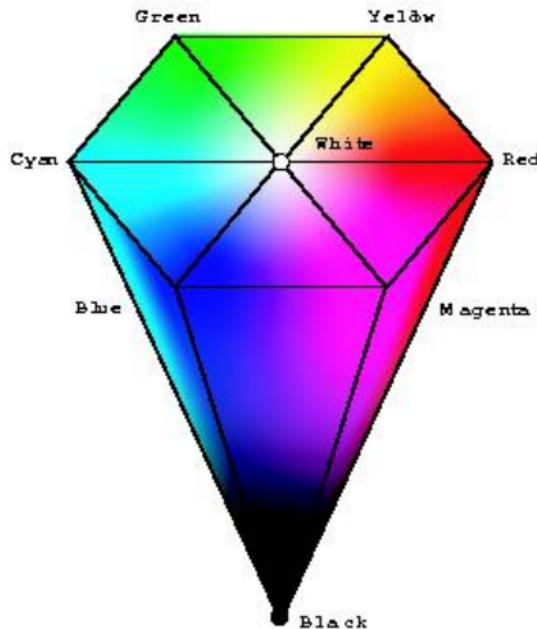
Gambar 6. Ekstraksi warna merah, hijau, dan biru dari suatu gambar

2.3.2. Model warna HSV

Model warna HSV (Hue, Saturation, Value) adalah model yang menggambarkan cara kita memahami dan mengklasifikasikan warna dalam ruang warna. Hue, yang merupakan atribut yang mencerminkan warna sebenarnya seperti merah, violet, dan kuning, digunakan untuk membedakan warna-warna serta menentukan karakteristik kemerahahan, kehijauan, dan lainnya dari cahaya. Nilai Hue dikuantisasi dari 0 sampai 2π , dengan 0 mewakili merah dan perputaran nilai-nilai spektrum tersebut kembali ke 0 untuk menyatakan merah lagi.

Saturation mengacu pada tingkat kemurnian warna cahaya, menunjukkan seberapa jauh warna itu dari warna putih. Sebagai contoh, warna merah adalah 100% warna jenuh (saturated color), sedangkan warna pink adalah warna merah dengan tingkat kejemuhan yang rendah karena memiliki warna putih di dalamnya. Jadi, Saturation mencerminkan kedalaman atau seberapa murni suatu warna.

Value, di sisi lain, adalah tingkat kecerahan sebuah warna. Jika Saturation adalah seberapa jauh dari warna putih, Value menunjukkan seberapa terang atau gelap warna itu. Ketika Saturation adalah 0, warna tidak memiliki Hue dan terdiri dari warna putih saja. Namun, jika Saturation adalah 1, warna tersebut tidak memiliki warna putih tambahan (warna dasar).



Gambar 7. Model warna HSV berbentuk cone

2.3.3. Model warna Grayscale

Citra grayscale merupakan gambar yang hanya menampilkan skala warna dari hitam hingga putih. Nilai kecerahan dari setiap piksel dalam gambar bergantung pada tingkat keabuan, yang mengindikasikan sejauh mana tingkat kecerahan itu dapat diwakili. Dalam konteks gambar digital, tingkat keabuan sangat terkait dengan jumlah bit yang digunakan untuk merepresentasikan kecerahan piksel dalam gambar.

Suatu gambar berwarna dapat diubah menjadi citra grayscale dengan mengubah nilai piksel dari saluran RGB ke saluran grayscale. Terdapat rumus yang digunakan untuk melakukan konversi piksel warna ke piksel grayscale. Berikut adalah rumus untuk mengkonversi piksel berwarna menjadi grayscale :

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

Gambar 8. Rumus konversi RGB ke Grayscale

2.4. Tekstur

Secara keseluruhan, tekstur merujuk pada pengulangan elemen-elemen dasar yang sering disebut sebagai primitif atau texel (texture element). Texel sendiri terdiri dari beberapa piksel yang diatur dalam pola posisi yang bisa bersifat periodik, semi-periodik, atau bahkan acak. Sebuah aspek yang penting dalam menganalisis tekstur adalah penerapan matriks pasangan intensitas, yang juga dikenal sebagai Gray

Level Co-occurrence Matrix (GLCM), yang merupakan representasi matriks dua dimensi.

Matriks pasangan intensitas adalah representasi matriks yang menunjukkan seberapa sering dua piksel dengan intensitas tertentu muncul dalam suatu jarak dan arah yang spesifik di dalam citra. GLCM dapat dihitung dengan menggunakan beberapa orientasi spasial seperti 0 derajat, 45 derajat, 90 derajat, dan 135 derajat. Dari hasil matriks GLCM tersebut, fitur-fitur khusus dapat diukur untuk menggambarkan karakteristik tekstur dari citra yang sedang diteliti dalam penelitian ini.

1. Entropy

Entropy, digunakan untuk mengukur keteracakannya distribusi intensitas dengan menggunakan rumus persamaan berikut.

$$\text{entropy} = - \sum_{i=1}^K \sum_{j=2}^K p(i_1, i_2) \log p(i_1, i_2)$$

2. Correlation

Correlation, digunakan untuk mengukur korelasi antara sebuah piksel terhadap piksel tetangga dari keseluruhan citra dengan menggunakan rumus persamaan berikut.

$$\text{correlation} = \sum_{i=1}^K \sum_{j=1}^K \frac{(i - m_r)(j - m_c)p_{ij}}{\sigma_r \sigma_c}$$

3. Homogeneity

Homogeneity, digunakan untuk mengukur kehomogenan variasi intensitas dalam citra dengan menggunakan rumus persamaan berikut.

$$\text{homogeneity} = \sum_{i_1} \sum_{i_2} \frac{p(i_1, i_2)}{1 + |i_1 - i_2|}$$

2.5. Content Based Image Retrieval (CBIR)

Content-Based Image Retrieval (CBIR) atau temukan kenali citra adalah metode yang digunakan untuk mencari citra digital dalam suatu basis data citra. Proses ini bergantung pada konten aktual yang terdapat dalam sebuah citra seperti warna, bentuk, tekstur, dan informasi lain yang dapat diambil dari citra tersebut. Objek-objek ini dianalisis dalam proses pencarian berbasis konten.

CBIR merupakan aplikasi dari computer vision yang menggunakan teknik pencarian gambar dari basis data yang menyediakan gambar sebagai gambar uji. Proses query gambar melibatkan ekstraksi fitur-fitur dari citra yang mencakup

berbagai aspek seperti histogram, nilai warna, tekstur, dan deteksi tepi. Ini memungkinkan pencarian citra dengan menggunakan atribut-atribut visual yang terkandung dalam citra-citra yang ada dalam basis data. Pada kali ini, kita akan membahas

2.5.1 CBIR dengan parameter warna

Warna merupakan salah satu komponen gambar yang sering dipakai dalam sebagian besar sistem CBIR. Model warna adalah cara untuk menggambarkan warna yang dilihat manusia dalam komputasi. Saat ini, model warna dapat dibagi menjadi dua kategori: yang berorientasi pada perangkat keras dan yang berorientasi pada pengguna. Model warna berorientasi perangkat keras umumnya dipakai untuk warna pada perangkat tertentu.

Sebagai contoh, model warna RGB (merah, hijau, biru) digunakan pada monitor dan kamera. Model warna CMY (sian, magenta, kuning) dipakai pada printer, sementara warna YIQ digunakan dalam siaran televisi berwarna. Sedangkan model warna yang berorientasi pada pengguna meliputi HLS, HCV, HSV, MTM, dan CIE-LUV, yang didasarkan pada tiga aspek persepsi manusia terhadap warna: hue (variasi warna), saturation (kepekatan), dan brightness (kecerahan).

Sistem CBIR sering menggunakan model warna RGB secara luas. Dalam model ini, warna diungkapkan melalui tiga warna primer: merah, hijau, dan biru, dengan nilai masing-masing warna primer berkisar antara 0 hingga 255. Di sisi lain, HSV (hue, saturation, value) adalah turunan dari RGB. HSV sebenarnya menunjukkan kinerja yang lebih baik dalam membedakan warna dibandingkan dengan RGB.

Berikut ini rumus mengkonversi nilai-nilai RGB menjadi HSV serta langkah-langkah CBIR dengan *Global Color Histogram* :

1. Nilai dari RGB harus dinormalisasi dengan mengubah nilai *range* [0, 255] menjadi [0, 1]

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

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. Selanjutnya gunakan hasil perhitungan di atas untuk mendapatkan nilai HSV

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

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

$$V = C_{max}$$

4. Setelah mendapatkan nilai HSV lakukanlah perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity*

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

Dengan A dan B adalah vektor dan n adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*. Histogram warna global merupakan cara sederhana untuk mengekstrak fitur dari gambar. Keunggulan utamanya adalah perhitungan dan pencocokan yang efektif tinggi.

Fitur ini tidak berubah saat rotasi maupun translasi. Namun, kelemahannya adalah histogram warna global hanya menghitung frekuensi warna. Distribusi spasial

informasi warna hilang. Dua gambar yang benar-benar berbeda bisa memiliki histogram warna global yang sama, yang akan menyebabkan kesalahan dalam pengambilan data.

Selain itu, ada CBIR dengan melakukan pencarian histogram dengan block, image dibagi menjadi blok-blok berukuran $n \times n$. Setiap blok akan kehilangan signifikansinya jika ukurannya terlalu besar, sementara ukuran blok yang terlalu kecil akan meningkatkan waktu pemrosesan. Untuk pencarian blok yang lebih efektif, disarankan menggunakan blok berukuran 4×4 . Representasi nilai yang mewakili sebuah blok dapat dihasilkan dengan menghitung nilai rata-rata dari model warna HSV dari blok yang bersangkutan.

2.5.2 CBIR dengan parameter tekstur

Tekstur merupakan sifat penting yang menggambarkan permukaan berbagai jenis objek. Secara umum, istilah "tekstur" mengacu pada pengulangan elemen-elemen dasar yang disebut texel yang terdistribusi secara periodik, kuasi-periodik, atau acak. Berdasarkan penelitian tentang persepsi visual manusia, disimpulkan bahwa analisis dalam domain frekuensi atau skala yang berbeda lebih tepat digunakan untuk mengidentifikasi tekstur dan struktur suatu objek. Pendekatan ini memiliki tingkat sensitivitas yang lebih tinggi dalam memahami dan menganalisis tekstur secara lebih detail.

Co-occurrence matrix digunakan dalam CBIR untuk membandingkan tekstur antara piksel-piksel dalam suatu gambar. Matriks ini memungkinkan pemrosesan yang lebih efisien dan cepat, serta menghasilkan vektor dengan ukuran yang lebih kecil. Dalam konteks ini, misalkan terdapat gambar I dengan $n \times m$ piksel dan parameter offset $(\Delta x, \Delta y)$. Matriksnya dapat dirumuskan sebagai berikut:

Misalkan i dan j adalah nilai intensitas dari gambar, sementara p dan q mewakili posisi dalam gambar. Offset Δx dan Δy berkaitan dengan arah θ dan jarak yang digunakan dalam persamaan berikut:

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

Melalui persamaan tersebut, digunakan nilai θ adalah $0^\circ, 45^\circ, 90^\circ$, dan 135° . Sebagai gambaran, berikut diberikan contoh cara pembuatan *co-occurrence matrix*

The diagram illustrates the process of creating a Gray Level Co-occurrence Matrix (GLCM). On the left, a 4x5 input image is shown with values: Row 1: 1, 1, 5, 6, 8; Row 2: 2, 3, 5, 7, 1; Row 3: 4, 5, 7, 1, 2; Row 4: 8, 5, 1, 2, 5. Red circles highlight the first three columns of the first two rows. A red arrow labeled "GLCM" points to the resulting 8x8 co-occurrence matrix on the right. The matrix has columns labeled 1 through 8 and rows labeled 1 through 8. The matrix entries are: Row 1: 1, 1, 2, 0, 0, 1, 0, 0, 0; Row 2: 0, 0, 1, 0, 1, 0, 0, 0, 0; Row 3: 0, 0, 0, 0, 1, 0, 0, 0, 0; Row 4: 0, 0, 0, 0, 1, 0, 0, 0, 0; Row 5: 1, 0, 0, 0, 0, 1, 2, 0; Row 6: 0, 0, 0, 0, 0, 0, 0, 1; Row 7: 2, 0, 0, 0, 0, 0, 0, 0; Row 8: 0, 0, 0, 0, 1, 0, 0, 0.

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

Gambar 9. Cara Pembuatan Matrix *Occurrence*

Setelah didapat *co-occurrence matrix*, buatlah *symmetric matrix* 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 untuk menerapkan CBIR dengan parameter tekstur adalah sebagai berikut:

1. Gambar dikonversi dari warna ke grayscale karena warna tidak relevan dalam menentukan tekstur. Proses ini mengubah warna RGB menjadi warna grayscale Y menggunakan rumus :

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

2. Kuantifikasi nilai grayscale dilakukan dengan mengelompokkan nilai-nilai piksel ke dalam berbagai kategori atau rentang. Dalam citra grayscale berukuran 256 piksel, matriks yang berkoresponden akan menjadi 256×256 untuk menggambarkan kombinasi nilai piksel yang mungkin.

Penilaian kemiripan gambar berdasarkan kekasaran tekstur dapat dilakukan dengan mempertimbangkan distribusi nilai-nilai piksel dalam matriks ini. Misalnya, mengidentifikasi pola nilai-nilai piksel yang sering muncul atau menghitung perbedaan antara nilai-nilai piksel tetangga untuk menentukan tingkat kekasaran tekstur pada gambar.

Pendekatan ini akan membantu dalam membandingkan gambar dan menilai kemiripannya berdasarkan tingkat kemiripan distribusi nilai-nilai piksel yang menggambarkan tekstur dalam gambar tersebut.

3. Dari *co-occurrence matrix* bisa diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*. Persamaan yang dapat digunakan untuk mendapatkan nilai 3 komponen tersebut 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:

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

Keterangan : P merupakan matriks *co-occurrence*

Dari ketiga 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:

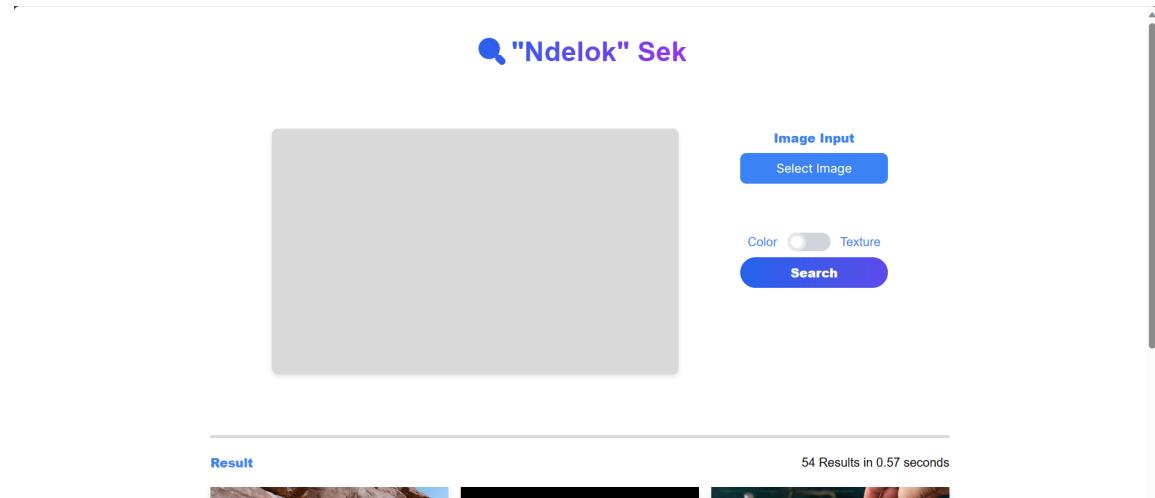
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

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

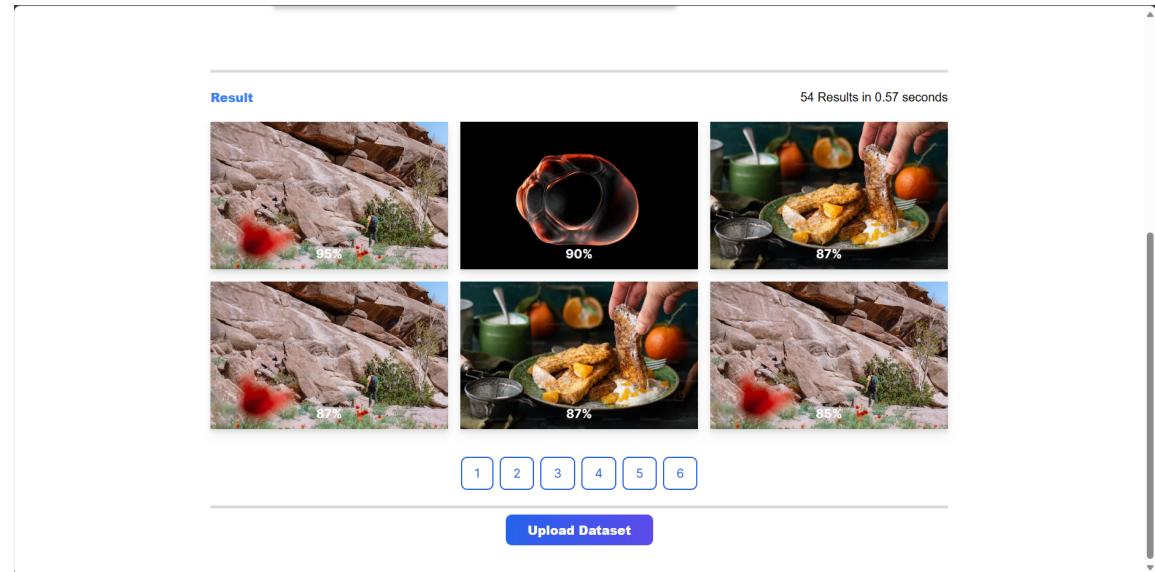
2.6. Website

Untuk keberjalanannya tugas kami, kami memerlukan sebuah website untuk menjalankan algoritma yang telah kami buat. Pada dasarnya, program website kami menggunakan framework berupa next.js yang terintegrasi dengan bahasa pemrograman typescript. Next.js bekerja dengan memanfaatkan beberapa fitur dan teknologi modern dalam pengembangan web, seperti React, Node.js, dan Webpack. Untuk bagian frontend, kami menggunakan library javascript yaitu React. Lalu disisi algoritma backend kami menggunakan python untuk memudahkan proses pengerjaan CBIR. Selain itu, API yang kami gunakan yaitu Fast API karena dinilai lebih cepat dibandingkan dengan platform API yang lain. Kami menyimpan cache data menggunakan database mongoDB. Terdapat alur pengembangan web yang kami buat, yaitu;

1. Membuat rancangan untuk Frontend
2. Melakukan styling untuk Frontend
3. Melakukan finalisasi Frontend
4. Membuat rancangan untuk Backend
5. Melakukan pembuatan database dengan menggunakan mongoDb
6. Mengintegrasikan database dengan Backend
7. Proses penggabungan antara Frontend dan Backend



Gambar 10. Tampilan website CBIR “Ndelok” Sek



Gambar 11. Tampilan website CBIR “Ndelok” Sek

Bab III

Analisis Pemecahan Masalah

3.1. Langkah-Langkah Pemecahan Masalah

Bab ini akan membahas dan menganalisis langkah-langkah pemecahan masalah dalam program CBIR berbasis web yang menggunakan parameter warna dan tekstur. Berikut merupakan langkah-langkahnya.

1. Menerima input data dari pengguna di website

Pada tahap ini, program dapat menerima dua input yaitu berupa dataset sebagai pembanding dan gambar yang ingin dibandingkan. Ketika pengguna memberi input dataset maka dataset akan diproses dalam CBIR lalu disimpan ke dalam database. Ketika pengguna memberi input gambar yang ingin dibandingkan, maka gambar tersebut akan diproses dalam CBIR lalu dibandingkan dengan yang ada di database.

2. Memproses gambar input dengan gambar pada dataset dengan menggunakan *Content Based Information Retrieval* (CBIR)

Tahap CBIR akan dibagi menjadi 2 parameter, yakni CBIR dengan parameter warna dan CBIR dengan parameter tekstur. Berikut merupakan tahap pemecahan masalah pada CBIR warna dan CBIR tekstur.

2.1 Tahap pemrosesan gambar dengan CBIR Warna

A. Ekstraksi fitur dari gambar

Proses dimulai dengan membaca gambar menggunakan fungsi cv2.imread(Path Gambar) dari library OpenCV, dilanjut dengan gambar diubah ke dalam format floating point dan kemudian dinormalkan dengan membaginya dengan 255. Normalisasi dilakukan untuk memastikan rentang nilai piksel berada antara 0 hingga 1.

B. Konversi RGB ke HSV

Setelah itu gambar yang dinormalisasi dikonversi ke format HSV. Ini dilakukan dengan membuat array kosong yang serupa dengan gambar dalam tipe data float32 untuk menyimpan nilai HSV dari gambar. Perhitungan nilai-nilai Hue, Saturation, dan Value dilakukan berdasarkan nilai piksel pada gambar yang telah dikonversi ke format HSV. Hue diisi dengan nilai Hue (warna) yang dihitung berdasarkan formula yang kompleks. Ini tergantung pada perbandingan nilai-nilai RGB dari piksel dan dihitung dalam rentang 0-360 . Saturation diisi

dengan nilai Saturation (kejenuhan) yang dihitung berdasarkan perbandingan nilai-nilai RGB. Value diisi dengan nilai Value (kecerahan) yang diambil dari nilai maksimum di antara komponen RGB.

C. Pembuatan histogram distribusi warna HSV

Setelah mendapatkan nilai-nilai Hue, Saturation, dan Value untuk setiap piksel gambar, histogram warna dibuat menggunakan fungsi cv2.calcHist. Histogram dibuat dengan memanfaatkan OpenCV dengan 8 bins untuk setiap komponen Hue, Saturation, dan Value, sehingga menghasilkan histogram 3D dengan 512 entri (8x8x8).

Pada tahap ini sebenarnya ada cara lain dengan melakukan blocking 4x4 pada gambar yang sudah diubah ke bentuk HSV. Setelah dilakukan blocking, menggunakan OpenCV untuk membuat histogram per blok lalu dicari rata-ratanya. Namun langkah ini menambah kompleksitas program dan menambah waktu pemrosesan gambar secara signifikan.

D. Perbandingan gambar menggunakan teorema *cosine similarity*

Setelah mendapatkan histogram dari gambar 1 (gambar input) dan gambar 2 (gambar pada dataset). Dilakukan pengukuran kemiripan dengan menggunakan teorema cosine similarity. Rumus cosine similarity adalah hasil dot product vektor histogram 1 dengan vektor histogram 2 dibagi dengan perkalian panjang vektor histogram 1 dan panjang vektor histogram 2. Untuk dot product, dapat dikalkulasi dengan memanfaatkan np.dot pada library NumPy. Serta, memanfaatkan np.linalg.norm untuk mencari panjang vektor dari histogram.

2.2 Tahap pemrosesan gambar dengan CBIR Tekstur

A. Ekstraksi fitur dari gambar dan mengkonversi RGB ke Grayscale
Proses dimulai dengan membaca gambar menggunakan fungsi cv2.imread(Path Gambar) dari library OpenCV. Setelah itu gambar RGB dikonversi menjadi citra abu-abu dengan memberikan bobot tertentu pada setiap saluran warna (merah, hijau, biru). Bobot ini digunakan untuk menghitung nilai piksel abu-abu.

B. Mengekstraksi contrast,homogeneity, dan entropy dari GLCM

Untuk membuat GLCM atau *Gray-Level Co-Occurrence Matrix* adalah dengan cara membuat matriks kosong dengan ukuran 256 x 256 yang akan merepresentasikan matriks kemunculan tingkat abu-abu (GLCM). Setelah itu, dilakukan pengisian GLCM dengan menghitung kemunculan pasangan nilai tingkat abu-abu yang berdekatan dalam gambar. Setelah GLCM terbentuk, berbagai fitur tekstur diekstrak dari matriks ini menggunakan persamaan contrast, homogeneity, dan entropy pada bagian landasan teori.

C. Perbandingan gambar menggunakan teorema *cosine similarity*

Setelah mendapatkan nilai contrast, entropy, dan juga homogeneity, kita dapat membuat suatu vektor yang berdimensi 3 hal tersebut. Dilakukan pengukuran kemiripan dengan menggunakan teorema cosine similarity. Rumus cosine similarity adalah hasil dot product vektor 1 dengan vektor 2 dibagi dengan perkalian panjang vektor 1 dan panjang vektor 2. Untuk dot product, dapat dikalkulasi dengan memanfaatkan np.dot pada library NumPy. Serta, memanfaatkan np.linalg.norm untuk mencari panjang vektor dari histogram.

3. Mengirimkan hasil CBIR dari Back-End ke Front-End

Setelah proses CBIR selesai dieksekusi di bagian back-end, hasilnya perlu dikirimkan kembali ke bagian front-end agar dapat ditampilkan kepada pengguna. Hal ini dapat dilakukan dengan beberapa cara:

Menggunakan Protokol Komunikasi :

- RESTful API: Back-end dapat menyediakan endpoint API yang dapat dipanggil oleh front-end untuk meminta hasil CBIR.
- HTTP Requests: Penggunaan HTTP requests (GET atau POST) dari front-end ke back-end untuk mengambil hasil yang diperlukan.

Format Data yang Dikirimkan :

- JSON: Hasil CBIR dapat dikirimkan dalam format JSON agar mudah diproses dan ditampilkan di front-end.
- Binary Data: hasilnya adalah gambar, bisa dikirimkan langsung sebagai data gambar (binary)

3.2. Proses pemetaan masalah menjadi elemen-elemen pada aljabar geometri.

Bab ini akan membahas pemetaan masalah menjadi elemen-elemen pada aljabar geometri dalam program CBIR berbasis web yang menggunakan parameter warna dan tekstur. Berikut merupakan proses pemetaannya:

3.2.1 Pemrosesan gambar dengan CBIR Warna

Masalah pada pemrosesan gambar dengan CBIR warna adalah mencari gambar dalam sebuah dataset yang mendekati gambar yang ingin dicari dengan berdasarkan karakteristik warna. Berikut adalah proses pemetaan masalahnya:

- 1. Ekstraksi fitur dari gambar**

Pada proses ini terdapat penggunaan materi aljabar geometri yaitu operasi matriks antara kernel (filter) dan bagian gambar. Setiap operasi konvolusi pada piksel-piksel gambar dapat diwakili sebagai produk titik antara matriks kernel dan area yang sesuai dari gambar.

- 2. Konversi RGB ke HSV**

Pada proses ini terdapat penggunaan materi aljabar geometri yaitu setiap piksel dalam gambar berwarna dapat direpresentasikan sebagai vektor dalam ruang warna. Misalnya, dalam ruang warna RGB, setiap piksel dapat direpresentasikan sebagai vektor dengan tiga dimensi, yaitu Red (R), Green (G), dan Blue (B). Konversi dari RGB ke HSV melibatkan transformasi ini, di mana vektor-vektor ini digunakan untuk menghitung nilai Hue, Saturation, dan Value.

- 3. Pembuatan histogram distribusi warna HSV**

Pada proses ini terdapat penggunaan materi aljabar geometri yaitu saat melakukan "blocking" pada gambar (membagi gambar menjadi blok-blok kecil), blok-blok ini dapat direpresentasikan sebagai matriks atau vektor dalam aljabar geometri. Setiap blok dapat dipandang sebagai entitas vektor atau matriks yang memiliki representasi spasial dalam ruang piksel. Pendekatan untuk membuat histogram blok dapat dipandang sebagai upaya untuk memetakan distribusi warna dalam ruang piksel. Blok-blok kecil pada gambar dapat dipandang sebagai sub bagian dari ruang piksel yang mencerminkan distribusi warna dalam area kecil.

- 4. Pada proses ini terdapat penggunaan materi aljabar geometri yaitu perbandingan gambar menggunakan teorema cosine similarity**

Menggunakan rumus cosine similarity untuk mengukur kesamaan antara dua vektor histogram. Rumus cosine similarity memanfaatkan

dot product dan panjang vektor untuk menghitung sudut kosinus antara dua vektor dalam ruang fitur. Konsep ini menerapkan prinsip geometri tentang hubungan sudut antara vektor-vektor dalam ruang.

3.2.2 Pemrosesan gambar dengan CBIR Tekstur

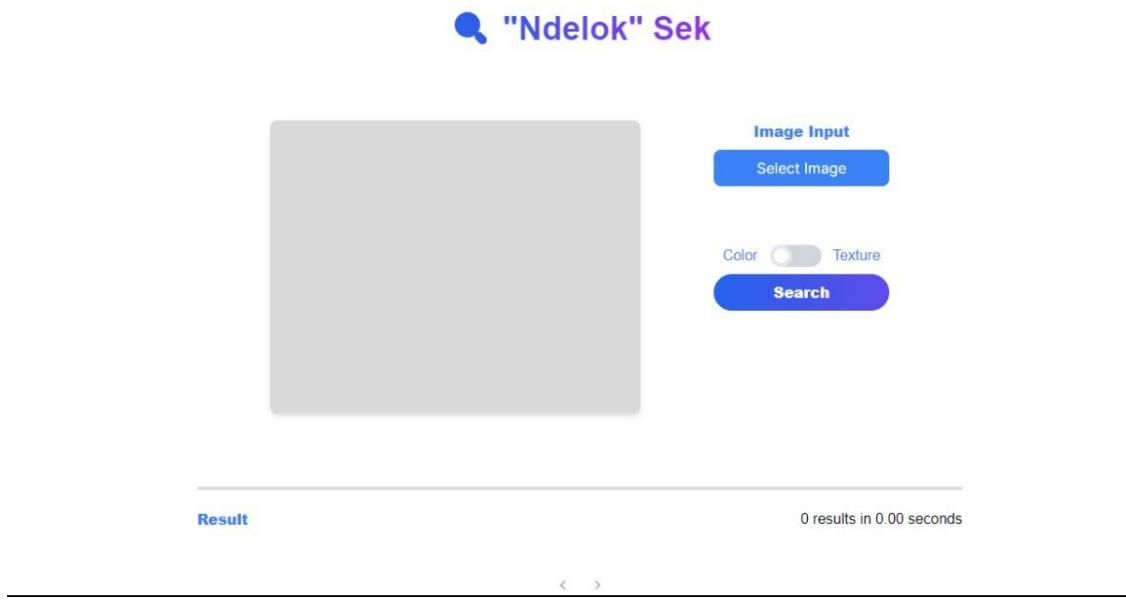
Masalah pada pemrosesan gambar dengan CBIR tekstur adalah mencari gambar dalam sebuah dataset yang mendekati gambar yang ingin dicari dengan berdasarkan karakteristik warna. Berikut adalah proses pemetaan masalahnya:

1. Ekstraksi fitur dari gambar dan mengonversi RGB ke Grayscale, pada proses ini terdapat penggunaan materi aljabar geometri yaitu citra dalam ruang warna RGB direpresentasikan sebagai matriks berdimensi tinggi x lebar x 3, di mana setiap elemen matriks mewakili nilai intensitas untuk setiap saluran warna (R, G, B) pada setiap piksel. Konsep ini mirip dengan representasi matriks dalam aljabar linear, di mana setiap piksel diinterpretasikan sebagai vektor dalam ruang piksel. Konversi ke citra abu-abu melalui pemberian bobot pada saluran warna merupakan pengubahan representasi warna dari ruang RGB yang kompleks menjadi representasi yang lebih sederhana dalam citra abu-abu. Proses ini dapat diinterpretasikan sebagai transformasi geometri yang mengubah representasi warna dalam ruang warna yang lebih kompleks menjadi ruang warna yang lebih sederhana dalam konteks analisis.
2. Mengekstraksi contrast,homogeneity, dan entropy dari GLCM, pada proses ini terdapat penggunaan materi aljabar geometri yaitu GLCM direpresentasikan sebagai matriks 2D dengan ukuran 256x256 (atau sesuai dengan jumlah level abu-abu dalam gambar). Matriks ini merepresentasikan frekuensi kemunculan pasangan nilai tingkat abu-abu yang berdekatan dalam gambar. Dalam konteks aljabar geometri, matriks ini dapat dipandang sebagai representasi transformasi linier dari distribusi nilai piksel dalam gambar ke dalam ruang yang lebih tinggi. Hasil dari fitur-fitur yang diekstrak dari GLCM, seperti contrast, homogeneity, dan entropy, dapat dianggap sebagai metrik yang mencerminkan sifat-sifat tekstur dalam gambar. Konsep ini sejalan dengan beberapa ide dalam aljabar geometri di mana metrik atau pengukuran digunakan untuk menganalisis sifat-sifat dalam ruang atau struktur yang dipelajari.
3. Perbandingan gambar menggunakan teorema cosine similarity,

pada proses ini terdapat penggunaan materi aljabar geometri yaitu, nilai-nilai contrast, entropy, dan homogeneity digunakan untuk membentuk vektor berdimensi tiga. Setiap dimensi vektor mewakili nilai dari satu fitur, sehingga vektor tersebut merepresentasikan suatu titik dalam ruang tiga dimensi. Konsep ini serupa dengan representasi vektor dalam aljabar geometri, di mana setiap dimensi vektor mewakili sumbu dalam ruang. Cosine similarity adalah metrik yang mengukur kemiripan antara dua vektor dalam ruang berdimensi banyak. Rumus cosine similarity memanfaatkan dot product dan panjang vektor untuk menghitung sudut kosinus antara dua vektor dalam ruang fitur. Konsep ini menerapkan prinsip geometri tentang hubungan sudut antara vektor-vektor dalam ruang.

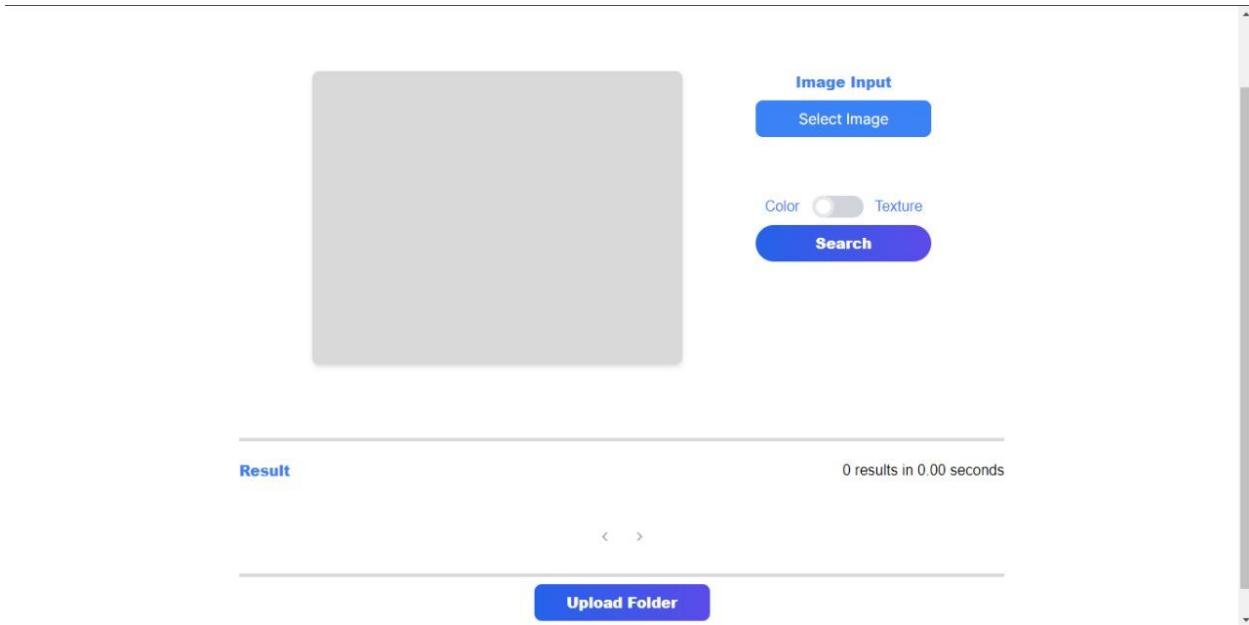
3.3. Contoh ilustrasi kasus dan penyelesaiannya.

1. User melakukan input gambar dan dataset



Gambar 12. Tampilan website CBIR saat menerima input gambar

Pada tahap ini, pengguna akan diminta untuk melakukan Input gambar. gambar tersebut akan berperan sebagai gambar yang ingin dibandingkan dengan dataset. Gambar ini diproses menggunakan CBIR dan dibandingkan dengan gambar yang ada di database.

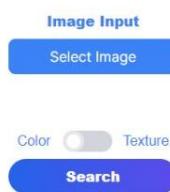


Gambar 13. Tampilan website CBIR saat menerima input dataset

Setelah melakukan input image, pengguna akan diminta untuk melakukan Input Dataset. Dataset tersebut akan berperan sebagai pembanding. Pada tahap ini juga Program memproses dataset ini menggunakan Content-Based Image Retrieval (CBIR) dan menyimpannya ke dalam database.

2. User memilih ingin melakukan CBIR dengan warna atau dengan tekstur

Ketika pengguna memilih opsi CBIR dengan berbasis warna, maka proses yang terjadi adalah Front End mengirimkan permintaan ke backend dengan parameter CBIR berbasis warna. Ketika pengguna memilih opsi CBIR dengan berbasis tekstur, maka proses yang terjadi adalah Front End mengirimkan permintaan ke backend dengan parameter CBIR berbasis tekstur.



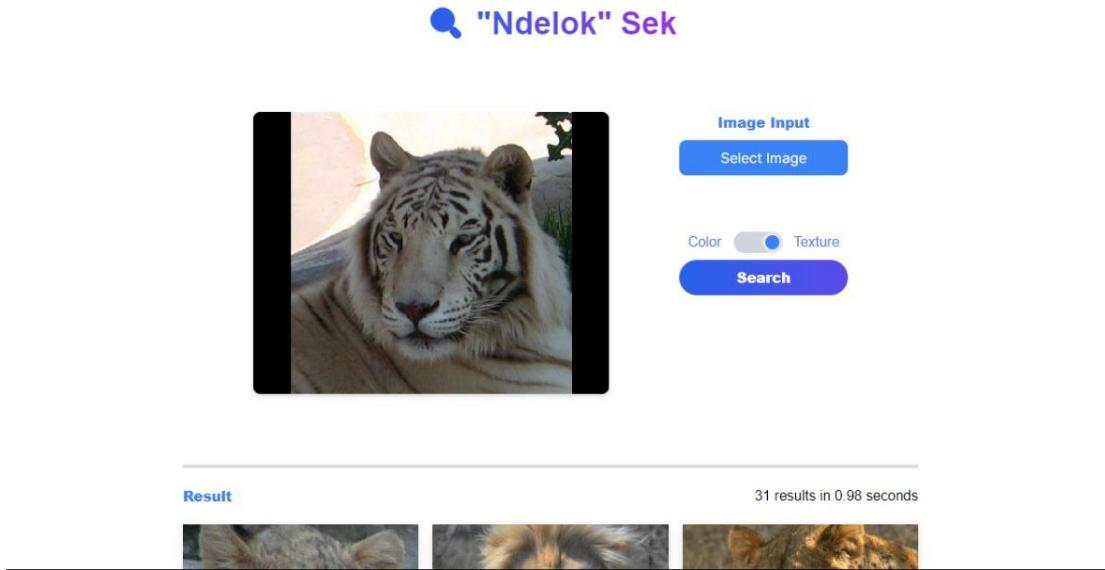
Gambar 14. Toggle untuk memilih parameter CBIR

3. User memencet search untuk memulai proses CBIR

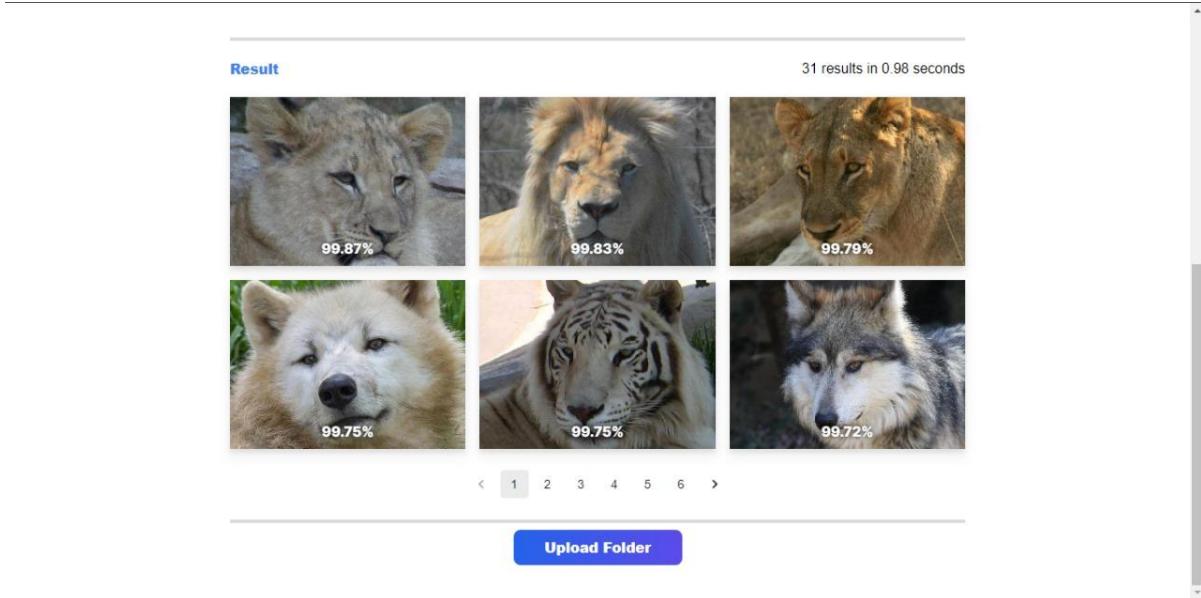
Setelah user mengklik search maka pada tahap ini, proses CBIR dijalankan. Tergantung apa parameter yang CBIR yang dipilih oleh user. Ketika user memilih CBIR warna, proses yang terjadi adalah Backend mulai memproses gambar dengan langkah-langkah CBIR berbasis warna seperti melakukan konversi RGB ke HSV, kalkulasi histogram HSV, menghitung kemiripan dengan cosine similarity. Ketika user memilih CBIR tekstur, proses yang terjadi adalah Backend memulai pemrosesan gambar menggunakan langkah-langkah CBIR berbasis tekstur seperti konversi RGB ke Grayscale, ekstraksi contrast,entropy, dan homogeneity. Setelah proses CBIR selesai, maka proses yang akan dijalani program adalah pengembalian hasil ke user interface.

4. User melihat hasil gambar yang paling mirip dengan gambar yang diinput

Proses yang terjadi pada program adalah hasil dari CBIR berbasis warna atau tekstur dikirim kembali ke frontend. Pada front end, hasil akan ditampilkan kepada pengguna. Pada akhirnya, Pengguna dapat melihat gambar-gambar yang mirip atau informasi yang relevan berdasarkan pilihan yang telah mereka input



Gambar 15. Hasil CBIR yang ditampilkan pada website CBIR



Gambar 16. Hasil CBIR yang ditampilkan pada website CBIR

Bab IV

Implementasi dan Uji Coba

4.1. Implementasi program utama

4.1.0. Type

```
{Definisi Type Matrix dengan element float}

constant ROW_CAP : integer = image.shape[0]

constant COL_CAP : integer = image.shape[1]

type Matrix: array [0..ROW_CAP -1] of array [0..COL_CAP
- 1] of float

{Definisi Type histogram dengan bins 8}

type histogram: array [0..8*8*8 - 1] of float

{Definisi Type Pixel}

type Pixel : array [0..2] of float {Menyimpan r,g,b}

{Definisi Type filename}

type filename : String {Menyimpan nama file}

{Definisi Type path}

type path : String {Menyimpan nama path directory}
```

4.1.1. CBIR_Colour

```
USE cv2
```

```

USE numpy as np

USE math

USE os

USE time

USE csv

USE pandas as pd

function calculate_histogram(img : Matrix) -> histogram

    {Initial State : Image terdefinisi dan sudah dibaca
     oleh cv2.imread(path_image) }
    {Final State : Terbentuk histogram yang sudah
     terdefinisi dengan jumlah bins 8 untuk setiap
     channel Hue, Saturation, dan Value}

KAMUS LOKAL

Cmax,Cmin,delta : float
Image, hsv_image : Matrix
hist : histogram

ALGORITMA

{Melakukan normalisasi image dengan membaginya
dengan 255}

image <- image.astype(np.float32) / 255.0

hsv_image = np.zeros_like(image, dtype=np.float32)

{Mencari Cmax,Cmin, dan delta}

Cmax <- np.max(image, axis = 2)

Cmin <- np.min (image, axis = 2

delta <- Cmax - Cmin

{Mengubah matriks RGB menjadi matriks HSV}

```

```

Pixel traversal [0..len(image)-1]

    if (delta = 0) then

        hsv_image[..., 0] ← 0

    else if (Cmax = image[..., 2]) then

        hsv_image[..., 0] ← 60 * (((image[..., 1]
- image[..., 0]) / (delta + 1e-5)) mod 6)

    else if (Cmax = image[..., 1]) then

        hsv_image[..., 0] ← 60 * (((image[..., 0]
- image[..., 2]) / (delta + 1e-5)) + 2)

    else if (Cmax = image[..., 0]) then

        hsv_image[..., 0] ← 60 * (((image[..., 2]
- image[..., 1]) / (delta + 1e-5)) + 4)

if (Cmax = 0) then

    hsv_image[..., 1] ← 0

else if (Cmax ≠ 0) then

    hsv_image[..., 1] ← delta / Cmax

    hsv_image[..., 2] ← Cmax

    hsv_image[..., 0] ← hsv_image[..., 0] * 0.5

{Mengkonversi ke bentuk uint8}

hsv_image ← (hsv_image * 255).astype(np.uint8)

{Membuat Histogram dengan bins 8 per channel H,S,V}

hist ← cv2.calcHist([hsv_image], [0, 1, 2], None,
[8, 8, 8], [0, 256, 0, 256, 0, 256])

hist ← cv2.normalize(hist, hist).flatten()

-> hist

```

```

function cosine_similarity(histogram1 : histogram,
histogram2 : histogram) -> float

    {Initial State : Parameter histogram telah
terdefinisi }
    {Final State : Mengembalikan angka kemiripan
berdasarkan rumus cosine similarity}

KAMUS LOKAL

similarity, norm1, norm2, dot : float

ALGORITMA

{Melakukan perhitungan kemiripan dengan menggunakan
teorema cosine similarity}

dot <- np.dot(histogram1, histogram2)

norm1 <- np.linalg.norm(histogram1)

norm2 <- np.linalg.norm(histogram2)

if (norm1 * norm2 ≠ 0) then

    similarity <- dot / (norm1 * norm2)

else

    similarity <- 0

-> similarity

function compareimage(input_image : Matrix,
data_directory : path) -> array [0..len(list_of_files)]
of float , array [0..len(list_of_files)] of integer,
array [0..len(list_of_files)] of filename

    {Initial State : Parameter image sudah terdefinisi,
path data_directory sudah terdefinisi}

```

```
{Final State : Mengembalikan hasil perbandingan  
kemiripan, nama file gambar yang sudah disortir  
dalam array }
```

KAMUS LOKAL

```
histogram_input : histogram  
  
List_of_files : array [0..number_of_files_in_folder  
- 1] of filename  
  
sim, sorted_similarities : array  
[0..len(list_of_files)] of float  
  
filenames, sorted_filenames : array  
[0..len(list_of_files)] of filename  
  
sorted_indices : array [0..len(list_of_files)]of  
integer
```

ALGORITMA

```
histogram_input ← calculate_histogram(input_image)  
  
{Menyimpan hasil similarity dalam array}  
  
sim ← []  
  
{Menyimpan nama file dalam array}  
  
filenames ← []  
  
Menampung file dalam folder dataset  
  
list_of_files ← os.listdir(data_directory)  
  
{Melakukan pengecekan terhadap semua file dalam  
list_of_files}  
  
filename traversal [list_of_files] :  
  
dataset_image ←
```

```
cv2.imread(os.path.join(data_directory, filename))  
  
    dataset_hist <  
calculate histogram(dataset_image)
```

```
        similarity <  
cosine similarity(histogram_input, dataset_hist)
```

```
if (similarity > 0.6 )then  
    sim.append(similarity * 100)  
    filenames.append(filename)
```

```
{Melakukan sort terhadap nama file dan nilai  
similarity}
```

```
sorted_indices < np.argsort(sim) [::-1]
```

```
sorted_similarities < np.sort(sim) [::-1]
```

```
sorted_filenames < []
```

```
i traversal [0..length(filenames)-1]
```

```
    index < sorted_indices[i]
```

```
    filename < filenames[index]
```

```
    sorted_filenames.append (filename)
```

```
-> sorted_indices, sorted_similarities,  
sorted_filenames
```

```
function compareimagehsv(input_image : Matrix, csv_file :  
path) -> array [0..len(df['histogram'])] of float, array  
[0..len(df['histogram'])] of integer, array  
[0..len(df['filepath'])] of filename
```

```
{Initial State : Parameter image sudah terdefinisi,  
path csv sudah terdefinisi, csv memiliki element  
histogram dan filepath}  
{Final State : Mengembalikan hasil perbandingan  
kemiripan, nama file gambar yang sudah disortir  
dalam array }
```

KAMUS LOKAL

```
histogram_input : histogram
```

```
sim,sorted_similarities : array  
[0..len(df['histogram'])] of float
```

```
filenames,sorted_filenames : array  
[0..len(df['filepath'])] of filename
```

```
sorted_indices : array [0..len(df['histogram'])] of  
integer
```

ALGORITMA

```
{Menghitung histogram dari gambar yang diinput}
```

```
histogram_input <- calculate_histogram(input_image)
```

```
{Membaca CSV file}
```

```
df <- pd.read_csv(csv_file)
```

```
{Mengkonversi histogram dengan elemen string  
menjadi list angka}
```

```
df['histogram'] <- df['histogram'].apply(lambda x:  
[float(i) for i in x.strip('[]').replace('\'r\n', '  
').split(' ') if i])
```

```
{Menghitung similarity dari histogram gambar input  
dengan semua histogram dataset dan menampungnya  
pada array sim}
```

```
sim <- [cosine_similarity(histogram_input,hist)*100
```

```

for hist in df['histogram']]

{Melakukan sort terhadap nama file dan nilai
similarity}

sorted_indices ← np.argsort(sim) [::-1]

sorted_similarities ← np.sort(sim) [::-1]

sorted_filenames ← [df['filepath'].iloc[i] for i in
sorted_indices]

-> sorted_indices, sorted_similarities,
sorted_filenames

```

4.1.2. CBIR_Tekstur

```

USE numpy as np

USE matplotlib.pyplot as plt

USE cv2

USE csv

USE math

USE os

USE pandas as pd

from PIL USE Image

from pathlib USE Path

USE multiprocessing as mp

function rgbtogray1(img : Matrix) -> Matrix

{Initial State : Image terdefinisi dan sudah dibaca
oleh cv2.imread(path_image)}
{Final State : Terbentuk histogram yang sudah
terdefinisi dengan jumlah bins 8 untuk setiap
channel Hue, Saturation, dan Value}

```

KAMUS LOKAL

```
r,g,b : array [0..len(img)-1] of integer
gray : Matrix
```

ALGORITMA

```
{Mengekstrak r,g,b pada img}
```

```
r, g, b < cv2.split(img)
```

```
{Mengubah warna RGB menjadi warna Grayscale}
```

```
gray < 0.299*r + 0.587*g + 0.114*b
```

```
gray < gray.astype(int)
```

```
-> gray
```

```
function compare(img1 : image, img2 : image) -> float
```

```
{Initial State : Image 1 dan Image 2 terdefinisi  
dan sudah dibaca oleh cv2.imread(path_image)}
```

```
{Final State : Terbentuk nilai similaritas image 1  
dengan image 2 dalam bentuk float}
```

KAMUS LOKAL

```
hasilDot,norm1,norm2 : float
```

ALGORITMA

```
{Menghitung kemiripan dengan menggunakan teorema  
cosine similarity}
```

```
hasilDot < np.dot(img1,img2)
```

```
norm1 < np.linalg.norm(img1)
```

```
norm2 < np.linalg.norm(img2)
```

```
-> hasilDot/(norm1*norm2)
```

```

function tesktur(img : Matrix) -> array [0..2] of float

    {Initial State : Image terdefinisi dan sudah dibaca
     oleh cv2.imread(path_image)}
    {Final State : Terbentuk array dengan elemen
     contrast,homogeneity,entropy}

KAMUS LOKAL

matrixFrameWork,imgGray : Matrix
imgGray : boolean
perintah : string

ALGORITMA

imgGray <- rgbtogray1(imgGray4)

matrixFrameWork <- np.zeros((256, 256),
dtype=np.int32)

i traversal[0..len(imgGray)-1] :

    j traversal[0..len(imgGray[0])-2] :

        matrixFrameWork [imgGray[i,j],imgGray[i,j+1]] <-
        matrixFrameWork [imgGray[i,j],imgGray[i,j+1]] + 1

        contrast <- np.sum(np.square(np.arange(256)[:,,
        np.newaxis] - np.arange(256)[np.newaxis, :]) *
        matrixFrameWork)

        homogeneity <- -np.sum(matrixFrameWork *
        np.log(matrixFrameWork + np.finfo(float).eps))

        entropy <- np.sum(matrixFrameWork / (1 +
        np.abs(np.arange(256)[:, np.newaxis] -
        np.arange(256)[np.newaxis, :])))

        array <- [contrast,homogeneity,entropy]

-> array

```

4.1.3. Back-End (main.py)

```
USE base64

from dataclasses USE Field

from typing USE Optional

from bson USE ObjectId

from fastapi USE FastAPI, File, UploadFile

from matplotlib USE image

from pydantic USE BaseModel, BeforeValidator,
ConfigDict, parse_obj_as

from enum USE Enum

from typing USE Annotated

USE pymongo

USE driver

USE os

USE cv2

USE numpy as np

USE CBIR_colour

USE csv

from PIL USE Image

from fastapi USE FastAPI, File, UploadFile,
HTTPException

from fastapi.middleware.cors USE CORSMiddleware

from fastapi.responses USE JSONResponse

from typing USE List
```

```
USE shutil
```

```
USE time
```

```
Program Main
```

```
{ Program yang menjalankan Back-End website }
```

```
ALGORITMA
```

```
app <- FastAPI()
```

```
arrrFILE <- []
```

```
app.add_middleware(
```

```
CORSMiddleware,
```

```
allow_origins <- ["*"] ,
```

```
allow_credentials <- True,
```

```
allow_methods <- ["*"] ,
```

```
allow_headers <- ["*"] ,
```

```
)
```

```
class dataTekstur(BaseModel) :
```

```
    contrast:float
```

```
    homogeneity:float
```

```
    entropy:float
```

```
    filepath:str
```

```
    client <-
```

```
pymongo.MongoClient("mongodb+srv://predatorfocus17:muhhu  
l@clusterdata.adbmei2.mongodb.net/?retryWrites=true&w=ma  
jority")
```

```

db <- client.get_database("dataSetImg")

transaction <- db.get_collection("transaction")

dataBaseColour <- db.get_collection("dataColour")

dataBaseTekstur <- db.get_collection("dataTekstur")

dataBaseBaru <- db.get_collection("tesss")



sim <- []

dataset <- []

global_waktu <- 0


UPLOAD_FOLDER <- "upload_images"

folder_path <- os.path.join(UPLOAD_FOLDER)





procedure delete_files_in_folder(input folder_pat : path, output folder : folder_data) :

    filename traversal [0..os.listdir(folder_pat)- 1]

        file_path <- os.path.join(folder_pat, filename)

        if (os.path.isfile(file_path) or
os.path.islink(file_path)) then

            os.unlink(file_path)

        else if (os.path.isdir(file_path)) then

            shutil.rmtree(file_path)






@app.post('/tekstur')

procedure insert_tekstur(input data_directory :

```

```

path,input namaColl : string, output databaseBaru :
database) :

    list_of_files < os.listdir(data_directory)

    output(data_directory)

    i < 0

    DataBaseBaru < db.get_collection(namaColl)

    DataBaseBaru.delete_many({ })

    output(len(list_of_files))

    filename traversal [0..list_of_files - 1] :

        dataset_image <
cv2.imread(os.path.join(data_directory, filename))

        dataset_image <
cv2.resize(dataset_image, (0,0), fx=0.5, fy=0.5)

        dataset_tekstur < driver.tekstur(dataset_image)

        data <
dataTekstur(contrast<dataset_tekstur[0], homogeneity<dataset_tekstur[1], entropy<dataset_tekstur[2], filepath<filename)

        DataBaseBaru.insert_one(data.dict())



@app.get('/colour')

procedure get_colour(input data_directory : path, output
arrHasil : array [0..len(sorted_similarities]-1) of
float)

    list_of_files < os.listdir(data_directory)

    filename < list_of_files[0]

    input_image <
cv2.imread(os.path.join(data_directory, filename))

```

```

sorted_indices, sorted_similarities,sorted_filenames
← CBIR_colour.compareimage(input_image, data_directory)

arrHasil ← []

i traversal [0..len(sorted_similarities)-1]

sim ← []
sim.append(sorted_similarities[i])
sim.append(sorted_filenames[i])
arrHasil.append(sim)

@app.post('colour')

procedure insert_colour(input data_director : path,
output dataCSV : CSV file)

list_of_files ← os.listdir(data_director)

array ← []

pathCSV ←
"D:/Hul/ITB/Akademik/S3/Algeo/Tubes/Tubes2/algeo02-22135
/src/backend/dataCSV/data.csv"

filename traversal [list_of_files]

dataset_image ←
cv2.imread(os.path.join(data_director, filename))

histogram ←
CBIR_colour.calculate_histogram(dataset_image)

array.append({
    "histogram":histogram,
    "filepath":filename
})

```

```

with open(pathCSV, 'w', newline='') as csv_file:
    fieldnames = ['histogram', 'filepath']

    writer = csv.DictWriter(csv_file,
fieldnames=fieldnames)

    writer.writeheader()

    row traversal [array]

    writer.writerow(row)

@app.post("/uploadfile/")

function create_upload_file(file: UploadFile =
File(...)) -> array[0..len(sorted_indices)-1] of
string,float

    image_content = await file.read()

    nparr = np.frombuffer(image_content, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

    vektor1 = driver.tekstur(img)

    result_filter = dataBaseBaru.find({})

    result_filter = list(result_filter)

    arrHasil = []

    file = []

    sim = []

    i traversal[0..len(result_filter)-1]

    vektor2 =
[result_filter[i].get('contrast'), result_filter[i].get('

```

```

homogeneity') , result_filter[i].get('entropy'))]

hasil <- driver.compare(vektor1, vektor2)*100

sim.append(hasil)

file.append(result_filter[i].get('filepath'))

sorted_indices <- np.argsort(sim) [::-1]

sorted_similarities <- np.sort(sim) [::-1]

sorted_filenames <- [file[i] for i in sorted_indices]

hasil <- []

i traversal[0..len(sorted_indices)-1]

image <- cv2.imread(os.path.join(folder_path,
sorted_filenames[i]))

_, buffer <- cv2.imencode('.JPG', image)

image_encode <-
base64.b64encode(buffer.tobytes()).decode("utf-8")

hasil.append({

    "persentase":sorted_similarities[i] ,
    "images":image_encode
})

-> hasil

@app.post("/uploadfile2/")

function create_upload_file(file: UploadFile =
File(...)) -> array [0..49] of string, float

start_time <- time.time()

image_content <- await file.read()

```

```

arrHasil < []

nparr < np.frombuffer(image_content, np.uint8)

img < cv2.imdecode(nparr, cv2.IMREAD_COLOR)

pathCSV <
"D:/Hul/ITB/Akademik/S3/Algeo/Tubes/Tubes2/algeo02-22135
/src/backend/dataCSV/data.csv"

sorted_indices, sorted_similarities,sorted_filenames
<- CBIR_colour.compareimageHSV(img, pathCSV)

hasil < []

i traversal[0..49]

    image < cv2.imread(os.path.join(folder_path,
sorted_filenames[i]))

    _,buffer < cv2.imencode(' .JPG', image)

    image_encode <
base64.b64encode(buffer.tobytes()) .decode("utf-8")

    hasil.append({ 

        "persentase":sorted_similarities[i] , 

        "images":image_encode

    })

output("sukses")

end_time < time.time()

global global_waktu

global_waktu < end_time - start_time

-> hasil

@app.post("/upload")

```

```

procedure upload_files(input/output file: UploadFile =
File(...))

    output ("tess")

    os.makedirs(folder_path, exist_ok=True)

    file_path <- os.path.join(folder_path,
file.filename.replace('/', '_'))

    arrrFILE.append(file.filename)

    with open(file_path, "wb") as f:

        f.write(await file.read())

@app.post("/uploadtodatabase")

procedure upload_files_toDB():

    insert_colour(folder_path)

    output ("sukses")

@app.post("/hapusdataset")

procedure upload_files_toDB():

    arrrFILE <=[]

    delete_files_in_folder(folder_path)

@app.get("/hasil/{namafile}")

function get_image(input namafile:str) -> image

    list_of_files <- os.listdir(folder_path)

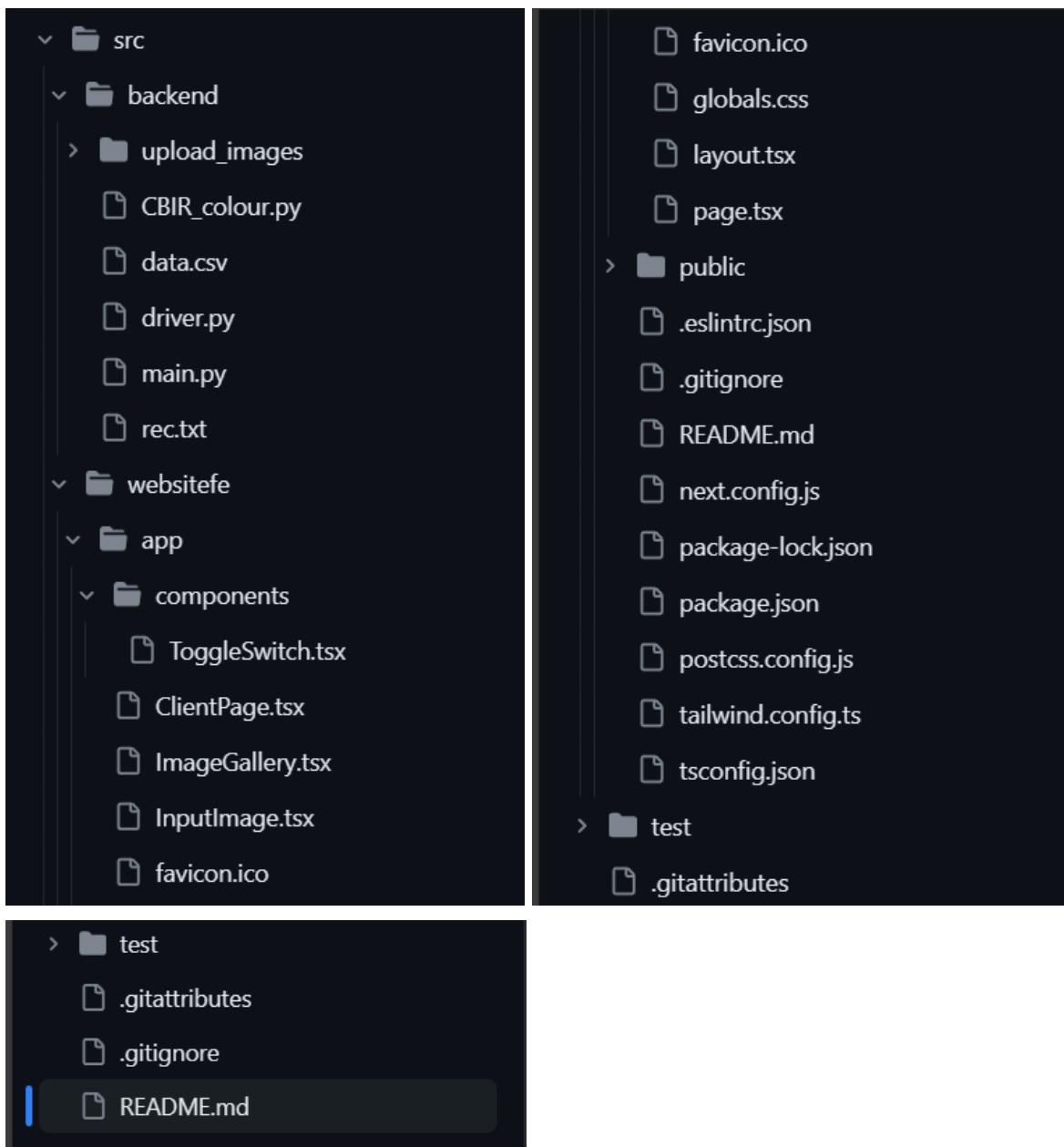
    img traversal [list_of_files]

```

```
if (img = namafile) then
    -> (cv2.imread(os.path.join(folder_path,
img)) )

@app.post ("/time")
function timee() -> time
    -> global_waktu
```

4.2. Penjelasan struktur program



Frontend (Klien):

- Bahasa Pemrograman: TypeScript
- Framework: Next js
- Markup: HTML, CSS, Tailwind
- Alat-alat Pengembangan: Webpack

Backend (Server):

- Bahasa Pemrograman: Python
- Framework: FastAPI

- Database: MongoDB

Database:

- MongoDB

Tools & lainnya:

- Git Hub (Version Control)
- RESTful APIs
- Code Editors/IDEs (Visual Studio Code)

4.2.1. Struktur program CBIR Colour

No	Function	Penjelasan function
1	calculate_histogram	Menerima gambar sebagai parameter lalu mengembalikan color histogram dari gambar tersebut
2	cosine_similarity	Menerima parameter 2 histogram yang ingin dibandingkan, lalu mengukur kemiripan menggunakan teorema cosine similarity, mengembalikan nilai kemiripan kedua histogram tersebut
3	compareimage	Menerima parameter berupa image yang ingin dibandingkan beserta path dataset pembanding. Menggabungkan fungsi calculate_histogram dan cosine_similarity untuk mencari gambar pada dataset yang memiliki kemiripan lebih dari 60%. Mengembalikan list berisi nama file yang memiliki kemiripan lebih dari 60 % serta list berisi nilai similarity yang lebih dari 60 %

4	compareimageHSV	Menerima parameter berupa image yang ingin dibandingkan beserta path data CSV pembanding. Menggabungkan fungsi calculate_histogram dan cosine_similarity untuk mencari histogram pada dataset yang memiliki kemiripan lebih dari 60%. Mengembalikan list berisi nama file yang memiliki kemiripan lebih dari 60 % serta list berisi nilai similarity yang lebih dari 60 %
---	-----------------	---

4.2.2. Struktur program CBIR Teksture

No	Function	Penjelasan function
1	rgbtogray1	Mengubah image input dengan model warna RGB menjadi model warna Grayscale
2	tekstur	Melakukan ekstraksi contrast,homogeneity,entropy suatu gambar dan mengembalikkan vektor 1D dengan element contrast,homogeneity,entropy
3	compare	Menerima parameter 2 histogram yang ingin dibandingkan, lalu mengukur kemiripan menggunakan teorema cosine similarity, mengembalikan nilai kemiripan kedua vektor tersebut

4.2.3. Back-End

No	File	Penjelasan file
1	main.py	Mengoperasikan sistem back-end yakni menerima file dari front-end, mengoperasikan CBIR warna dan tekstur, menaruh hasil CBIR pada database, mengembalikan hasil

		kepada front-end
--	--	------------------

4.2.4. Front-End

Semuanya terdapat di folder app.

No	Filename	Penjelasan function
1	Page.tsx	Menampilkan tampilan web dengan memanggil file ClientPage.tsx
2	ClientPage.tsx	Tempat dimana menyatukan antara input user dan menampilkan ke halaman. Di dalamnya terdapat end point dr backend.
3	layout.tsx	Digunakan untuk styling.
4	InputImage.tsx	Di dalamnya terdiri dari input image, toggle dan juga proses search button.
5	ImageGallery	Di dalamnya terdapat untuk mengupload file dan juga pagination untuk menampilkan hasil dari back end
6	ToogleSwitch.tsx	Komponen pembuatan toogle

4.3. Tata cara penggunaan program

1. Clone github, buka terminal
2. Ganti directory ke website
3. Install semua dependencies yang diperlukan seperti, `npm i`, install node js
4. Lalu jalankan frontend dengan cara `npm run dev`
5. Buka terminal baru
6. Ganti directory ke backend
7. Install semua dependencies yang diperlukan seperti, `pip install -r rec.txt`, pastikan anda sudah menginstall python, jika belum install terlebih dahulu
8. Lalu jalankan backend dengan cara `python -m unicorn main:app`

9. Buka link web pada frontend, web siap digunakan
10. Note: mungkin akan terdapat kesalahan saat menjalankan backend, hal ini bisa terjadi karena ip address belum ditambahkan di database, mohon untuk contact salah satu dari kami, terimakasih.

4.4. Hasil Pengujian

Pengujian color 1

Image Input

Color Texture

Result 0 results in 0.00 seconds

< >

Hasil pengujian color 1

Result 21 results in 1.74 seconds

< 1 2 3 4 >

Pengujian color 2

🔍 "Ndelok" Sek



Image Input

Select Image

Color Texture

Search

Result

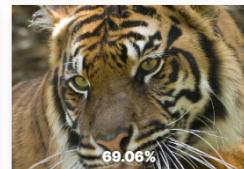
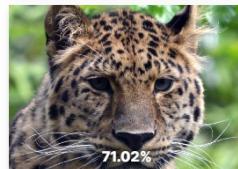
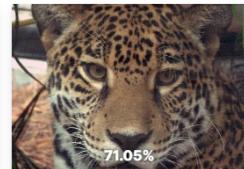
21 results in 1.74 seconds



Hasil pengujian color 2

Result

21 results in 1.74 seconds



< 1 2 3 4 >

Upload Folder

Pengujian texture 1

🔍 "Ndelok" Sek



Image Input

Select Image

Color Texture

Search

Result

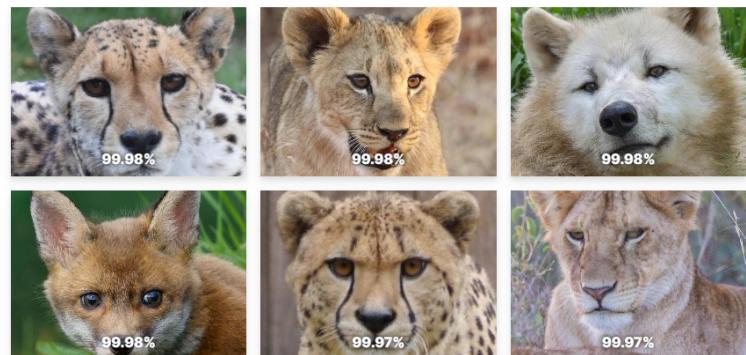
21 results in 1.74 seconds



Hasil pengujian texture 1

Result

106 results in 1.53 seconds



< 1 ... 14 15 16 17 18 >

Upload Folder

Pengujian texture 2

🔍 "Ndelok" Sek



Image Input

Select Image

Color Texture

Search

Result

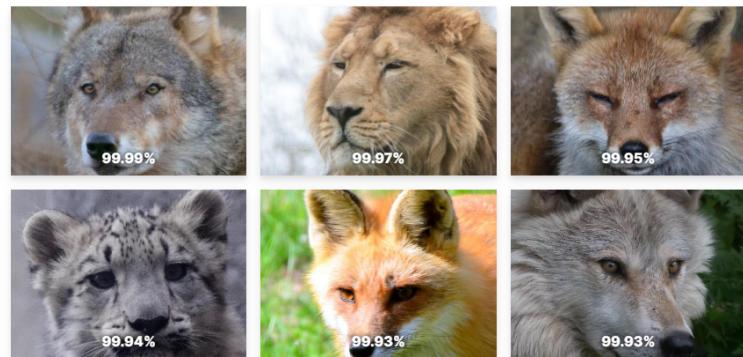
5 results in 1.11 seconds



Hasil pengujian texture 2

Result

106 results in 1.50 seconds



< 1 2 3 4 5 ... 18 >

Upload Folder

Pengujian upload dataset

Upload 106 files to this site?
This will upload all files from "tes". Only do this if you trust the site.

Result

06 results in 1.50 seconds

Upload Cancel

99.99% 99.97% 99.95%

99.94% 99.93% 99.93%

< 1 2 3 4 ... 18 >

Upload Folder

Hasil image di folder

upload_images	
 tes_1009.jpg	U
 tes_1010.jpg	U
 tes_1011.jpg	U
 tes_1012.jpg	U
 tes_1013.jpg	U
 tes_1014.jpg	U
 tes_1015.jpg	U
 tes_1016.jpg	U
 tes_1017.jpg	U
 tes_1018.jpg	U
 tes_1019.jpg	U
 tes_1020.jpg	U
 tes_1021.jpg	U
 tes_1022.jpg	U
 tes_1023.jpg	U
 tes_1024.jpg	U
 tes_1025.jpg	U
 tes_1026.jpg	U
 tes_1027.jpg	U
 tes_1028.jpg	U
 tes_1029.jpg	U
 tes_1030.jpg	U
 tes_1031.jpg	U
 tes_1032.jpg	U
 tes_1033.jpg	U
 tes_1034.jpg	U

Hasil chace untuk color

```

1 histogram,filepath
2 "[3.2079875e-02 1.56171797e-02 1.45229828e-02 7.31122773e-03
3 6.66465657e-03 4.37678909e-03 5.35161942e-02 1.35730287e-01
4 3.97889933e-04 2.93443819e-03 2.56141648e-02 4.59562875e-02
5 4.14382908e-02 1.02506392e-01 1.84322506e-01 7.41069987e-02
6 9.94724804e-04 3.08364784e-02 6.54528961e-02 6.96307346e-02
7 7.18191341e-02 7.25651756e-02 6.93820566e-02 9.00225993e-03
8 3.49153699e-04 4.53097150e-02 6.70941919e-02 4.33700010e-02
9 2.87475474e-02 5.09796478e-02 1.04943467e-02 4.97362402e-04
10 3.92916286e-03 1.88500360e-02 2.45697033e-02 1.51695535e-02
11 4.22260687e-02 5.37151424e-03 1.24340699e-03 0.00000000e+00
12 1.591155973e-03 9.30067711e-03 5.66993142e-03 1.34785213e-02
13 7.75885349e-03 3.97889933e-04 0.00000000e+00 0.00000000e+00
14 9.44988569e-04 3.82969063e-03 3.18311946e-03 3.82969063e-03
15 1.04446104e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
16 1.98944961e-03 9.94724804e-04 5.96834929e-04 7.95779866e-04
17 1.98944967e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
18 0.00000000e+00 0.00000000e+00 1.97452884e-02 1.17874891e-02
19 5.023366252e-03 1.17377527e-02 2.22321804e-02 8.78839418e-02
20 0.00000000e+00 3.36216986e-02 3.28259207e-02 4.68515381e-02
21 2.49675941e-02 7.09238797e-02 1.00815363e-01 7.95779899e-02
22 7.06254644e-03 2.85983384e-02 5.17754294e-02 5.66495866e-02
23 6.19216226e-02 8.619129059e-02 1.19317241e-01 4.40165736e-02
24 6.41597528e-03 2.76836132e-02 2.67588971e-02 4.49113279e-02
25 4.92388792e-02 6.73926109e-02 1.59653332e-02 6.96307397e-04
26 3.28259193e-03 9.59999426e-03 1.88500360e-02 2.37241872e-02
27 5.50082847e-02 1.43737737e-02 1.74076844e-03 0.00000000e+00
28 2.88470206e-03 5.57845018e-03 7.90806208e-03 1.34024088e-02
29 1.45727191e-02 1.09419727e-03 0.00000000e+00 0.00000000e+00
30 1.29314233e-03 1.93971337e-03 3.73021816e-03 7.36096362e-03
31 1.34287856e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00
32 7.46043632e-04 6.96307397e-04 8.95252335e-04 7.46043632e-04
33 1.49288732e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00
34 3.97889921e-03 1.31303677e-02 4.70007472e-02 2.00437047e-02

```

Hasil chace database untuk texture

4.5. Analisis dari desain solusi

Membandingkan keakuratan antara Color-Based Image Retrieval (CBIR) berbasis warna dan tekstur tidak memiliki jawaban definitif karena keduanya memiliki kelebihan dan kelemahan yang mempengaruhi performanya. CBIR berbasis warna menonjolkan fitur yang mudah terlihat dan dikenali manusia. Keunggulan lainnya terletak pada kemampuannya dalam mencari objek yang sangat bergantung pada warna, seperti dalam bidang fashion.

Namun, kelemahannya terletak pada kerentanannya terhadap pengaruh lingkungan sekitar, membuatnya kurang efektif dalam situasi di mana objek memiliki struktur serupa tetapi warnanya berbeda atau saat informasi semantik dalam gambar perlu ditangkap.

Sementara itu, CBIR berbasis tekstur cenderung lebih stabil dalam menghadapi perubahan pencahayaan karena tidak bergantung pada perubahan warna. Tekstur juga efektif untuk gambar dengan pola serupa meskipun memiliki variasi warna yang signifikan, serta dapat lebih baik dalam menggambarkan informasi semantik dalam beberapa situasi. Namun, pengolahan tekstur dapat lebih kompleks daripada pemrosesan warna, dan persepsi terhadap tekstur bersifat subjektif, berbeda antara individu.

Oleh karena itu, untuk menentukan keunggulan antara CBIR berbasis warna dan tekstur, diperlukan evaluasi yang kontekstual dan spesifik terhadap kasus penggunaan. Dalam beberapa situasi, tekstur mungkin lebih unggul, sementara pada kasus lain, CBIR berbasis warna dapat memberikan hasil yang lebih baik. Tidak ada kesimpulan umum yang dapat menetapkan mana yang lebih baik karena faktor-faktor seperti jenis gambar, tujuan aplikasi, dan karakteristik data dapat mempengaruhi performa keduanya secara berbeda.

Bab V

Kesimpulan, Saran, dan Refleksi

I. Kesimpulan

Dalam sistem temu balik gambar atau biasa yang dikenal dengan CBIR, dapat digunakan berbagai macam metode. Metode tersebut dibagi menjadi 3 yakni, warna, tekstur, dan bentuk. Namun pada tugas besar ini, kami berfokus pada implementasi 2 metode yaitu CBIR dengan parameter warna dan tekstur.

Pada tugas besar ini, kami membuat sebuah aplikasi CBIR yang didasarkan pada aplikasi aljabar vektor. Proses CBIR dilakukan dengan mengubah matriks gambar menjadi vektor, lalu memanfaatkan teorema *Cosine Similarity* untuk mengukur kemiripannya. Kami mengimplementasikan ini dalam bentuk website lokal, yang dibuat menggunakan framework *NextJS* dan backend *FastAPI* sebagai tech stack dari website ini. Website yang kami buat berhasil melakukan CBIR sebuah gambar dengan dataset yang diberikan.

II. Saran

Saran pengembangan untuk tugas besar ini diantaranya:

1. Memanfaatkan library seperti *Multi-Processing* untuk memproses gambar dengan waktu yang lebih efisien.
2. Memahami / melakukan research lebih dalam mengenai CBIR untuk mendapatkan keakuratan yang maksimal.
3. Pembuatan UI / UX website dapat dibuat lebih indah dan elegan serta *user-friendly*

III. Komentar atau Tanggapan

Komentar atau tanggapan mengenai tugas besar ini dari kelompok kami adalah tugas besar ini cukup menyenangkan tapi juga cukup melelahkan untuk dikerjakan. Tugas besar ini menantang kami untuk berpikir *outside the box* dalam memecahkan berbagai rintangan yang kami hadapi. Tugas besar ini juga memaksa kami untuk belajar sesuatu yang belum pernah kami pelajari sebelumnya secara mendalam yakni *web development* yang mana menurut kami, itu adalah sesuatu yang sangat menantang dikarenakan tidak ada dari kami yang pernah membuat website yang mengimplementasikan *Back-End* sebelumnya.

IV. Refleksi

Dari tugas besar ini, kami merefleksikan bahwa pembelajaran tidak hanya terbatas pada pemahaman tentang materi aljabar vektor, tetapi juga melibatkan pengalaman dalam mengelola sebuah website. Tentunya, itu menjadi tantangan hebat karena kami bertiga belum menguasai *web development*. Sebagai sebuah kelompok, kami berhasil belajar bagaimana bekerja sama secara efektif dan saling mendorong satu sama lain. Meskipun demikian, kami menyadari adanya kekurangan dalam penyelesaian tugas besar ini, yang sebagian besar disebabkan oleh keterbatasan waktu dan kurangnya pemahaman mendalam dalam beberapa aspek. Namun, di tengah kekurangan tersebut, kami mendapatkan segudang pelajaran berharga dari proses ini. Tentunya, pengetahuan dari penggerjaan tugas besar ini akan kami manfaatkan untuk lebih siap menghadapi tugas-tugas besar berikutnya.

V. Ruang Perbaikan atau Pengembangan

Ruang perbaikan pada tugas besar ini diantaranya :

1. *Time management* yang lebih teratur dan jelas dalam penggerjaan tugas besar ini agar dapat menangani error sebelum waktu *deadline*.
2. Menyelesaikan program wajib jauh sebelum deadline agar dapat mengerjakan program bonus.
3. Mencari berbagai cara / alternatif untuk meningkatkan kecepatan waktu dan keakuratan.
4. Pembagian tugas dilakukan dengan lebih jelas dan baik lagi agar dapat meminimalisir pemberian tugas yang terlalu berlebihan pada satu individu.

Bab VI

Daftar Pustaka

Berikut adalah daftar referensi yang dipakai dalam penggerjaan tugas besar ini.

1. [Skripsi Rika Amalia \(Perpus\).pdf \(uinsu.ac.id\)](#) diakses pada 14 November 2023
2. [bab 2.pdf \(umg.ac.id\)](#) diakses pada 14 November 2023
3. [Materi IlmuKomputer.Com](#) diakses pada 14 November 2023
4. [Pengantar Pengolahan Citra \(itb.ac.id\)](#) diakses pada 16 November 2023
5. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2019-2020/Makalah2019/13516_133.pdf](#) diakses pada 16 November 2023
6. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2019-2020/16-Warna.pdf](#) diakses pada 15 November 2023
7. [https://elib.unikom.ac.id/files/disk1/697/jbptunikompp-gdl-riekalfahm-34835-7-uniko_m_r-a.pdf](#) diakses pada 16 November 2023
8. [https://media.neliti.com/media/publications/68886-ID-none.pdf](#) diakses pada 15 November 2023

Link Repository GitHub : <https://github.com/ChrisCS50X/Algeo02-22135.git> /
[ChrisCS50X/Algeo02-22135 \(github.com\)](https://github.com/ChrisCS50X/Algeo02-22135)

Link Youtube : <https://youtu.be/4xGEtr984sA?si=t4HYkChpMQwviq6U>