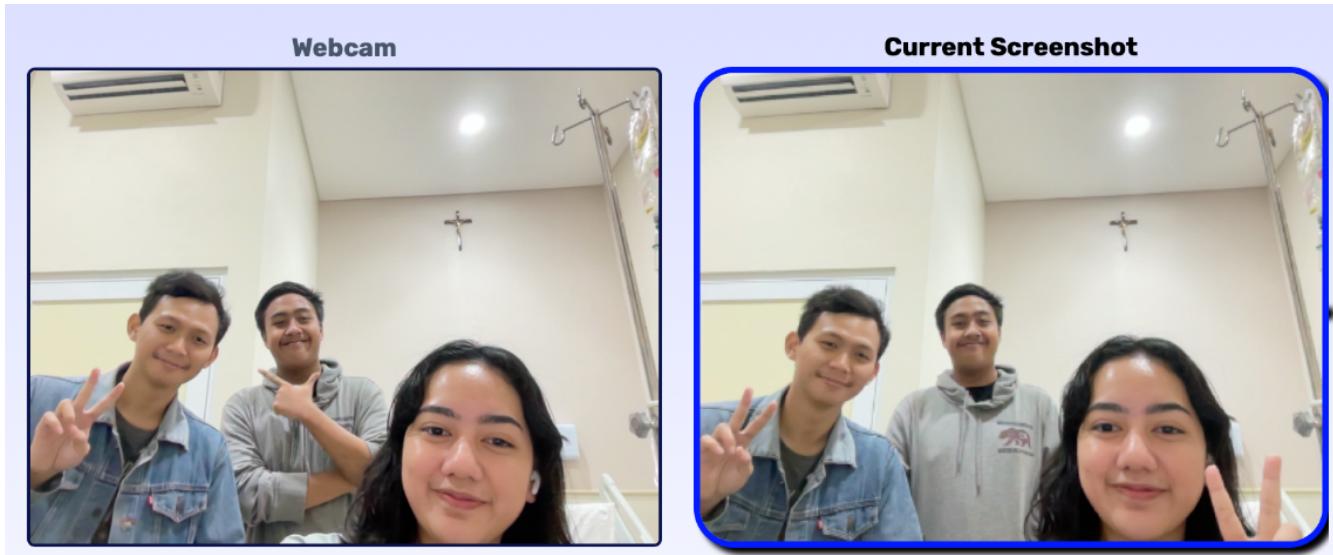


LAPORAN TUGAS BESAR 2
IF2123 - ALJABAR LINIER DAN GEOMETRI



Kelompok 23 “Bjir Anak Nopal”

Shazya Audrea Taufik 13522063

Bagas Sambega Rosyada 13522071

Raden Francisco Trianto B. 13522091

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

2023

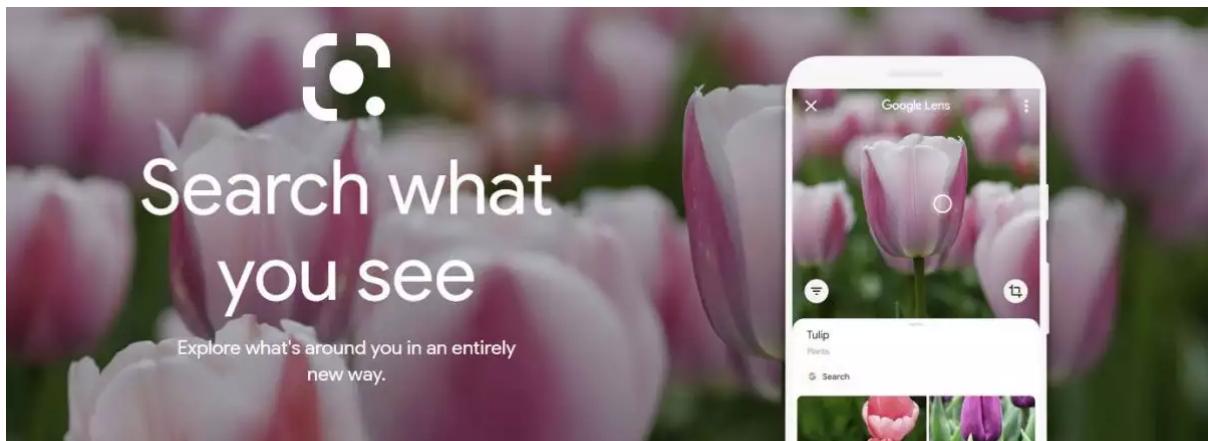
Daftar Isi

Daftar Isi.....	1
BAB I DESKRIPSI MASALAH.....	2
BAB II TEORI SINGKAT.....	3
2.1 Teori CBIR.....	3
2.1.1 Color.....	3
2.1.2 Texture.....	4
2.2 Pengembangan Website.....	6
BAB III ANALISIS PEMECAHAN MASALAH.....	7
3.1 Langkah-langkah Pemecahan Masalah.....	7
3.2 Pemetaan Masalah dalam bentuk Aljabar Geometri.	8
3.2.1 Variabel.....	8
3.2.2 Pemodelan Masalah.....	8
3.2.3 Penerapan Aljabar Geometri.....	9
3.3 Contoh Ilustrasi Kasus dan Penyelesaiannya.....	9
3.3.1 Kasus Warna.....	9
3.3.2 Kasus Tekstur.....	10
BAB IV IMPLEMENTASI DAN UJI COBA.....	14
4.1 Implementasi Program Utama.....	14
4.1.1 Color.....	14
4.1.2 Texture.....	16
4.2 Penjelasan Struktur Program Berdasarkan Spesifikasi.....	18
4.2.1 Website.....	19
4.2.2 Fungsi Pengolahan Warna.....	19
4.2.3 Fungsi Pengolahan Tekstur.....	20
4.3 Tata Cara Penggunaan Program.....	20
4.4 Hasil Pengujian.....	21
4.4.1 Uji Tekstur.....	21
4.4.1.1 Uji pencarian gambar yang terdapat pada dataset.....	21
4.4.1.2 Uji pencarian gambar yang tidak terdapat pada dataset.....	22
4.4.2 Uji Warna.....	23
4.4.2.1 Uji pencarian gambar yang terdapat pada dataset.....	23
4.4.2.2 Uji pencarian gambar yang tidak terdapat pada dataset.....	24
4.5 Analisis dari Desain Solusi Algoritma Pencarian.....	24
BAB V KESIMPULAN.....	26
5.1 Kesimpulan.....	26
5.2 Saran dan Refleksi.....	26
5.3 Komentar dan Ruang Perbaikan.....	27
DAFTAR PUSTAKA.....	28

BAB I

DESKRIPSI MASALAH

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



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

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

BAB II

TEORI SINGKAT

2.1 Teori CBIR

2.1.1 *Color*

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Salah satu kontennya adalah *color*. Pengambilan tekstur dimulai dengan melakukan *image processing* agar bisa mengakses bentuk matriks dari suatu gambar. Pada mulanya, kita perlu mengubah *image* yang berbentuk RGB menjadi sebuah metode histogram warna yang lebih umum.

Histogram warna digunakan untuk mendistribusikan warna dari gambar dengan menghitung frekuensi dari berbagai warna yang ada dalam ruang warna tertentu. Mereka tidak dapat mengidentifikasi objek tertentu pada gambar dan tidak dapat menjelaskan di mana warna didistribusikan. Pembagian nilai gambar menjadi interval yang lebih kecil harus dilakukan untuk membentuk ruang warna. Ini dilakukan untuk membuat sebuah histogram warna di mana setiap interval dianggap sebagai bin, dan piksel yang menunjukkan nilai warna untuk setiap interval dapat dihitung. Histogram global dan warna blok adalah fitur warna. Penggunaan histogram warna berbentuk blok nxn akan meningkatkan keakuratan pencarian gambar.

Penggunaan histogram warna akan memiliki kualitas yang lebih baik jika nilai-nilai pada tiap pixel-nya berbentuk HSV (*Hue, Saturation, Value*). Untuk mengubah RGB menjadi HSV dapat dilakukan dengan langkah-langkah berikut:

1. Normalisasi nilai RGB dengan membagi nilai R, G, dan B dengan 255.
2. Mencari C_{max} , C_{min} , dan Δ . Nilai C_{max} adalah nilai maksimum diantara R, G, dan B yang telah dinormalisasi. Nilai C_{min} adalah nilai minimum diantara R, G, dan B yang telah dinormalisasi. Nilai Δ adalah nilai perbedaannya.
3. Cari nilai HSV dengan rumus berikut

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

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

$$V = C_{\max}$$

Untuk mencari suatu kesamaan warna di antara dua gambar, kita dapat menjadikan komponen HSV sebagai vektor dan mencari nilai *cosine similarity* dengan rumus:

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

Semakin besar nilai cos nya (mendekati 1), maka semakin mirip warna gambarnya. Semakin kecil nilai cos nya (mendekati 0), maka semakin tidak mirip warna gambarnya.

2.1.2 Texture

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Salah satu kontennya adalah *texture*. Pengambilan tekstur dimulai dengan melakukan *image processing* agar bisa mengakses bentuk matriks dari suatu gambar. Langkah-langkah *image processing* adalah sebagai berikut:

1. Mengambil nilai RGB dari suatu gambar dalam bentuk matriks.
2. Mengubah nilai RGB menjadi nilai *grayscale* dengan rumus:

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

Setelah gambar diproses, untuk mencari matriks *Gray Level Co-occurrence Matrix* (GLCM), kita perlu melakukan langkah-langkah berikut ini:

1. Mencari matriks *co-occurrence* dengan cara mengecek nilai sebelah dari setiap elemen matriks dan menambahkan 1 dari kedua nilai tersebut sebagai baris dan kolom pada matriks. Matriks ini menghitung peluang terjadinya pasangan intensitas piksel pada jarak dan arah tertentu. Gambaran yang lebih lengkap dari pembuatan matriks *co-occurrence* bisa diperoleh dari gambar berikut ini.

GLCM

1	1	2	3	4	5	6	7	8
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 2. Contoh pembuatan *co-occurrence matrix*

2. Melakukan *transpose* terhadap matriks *co-occurrence* yang telah diperoleh.
3. Mencari *symmetric matrix* dengan menambahkan matriks *co-occurrence* dengan matriks hasil *transposenya*.
4. Mencari *normalized matrix* dari *symmetric matrix* dengan cara membagi setiap elemen dengan jumlah nilai semua elemen dalam matriks.

Dari nilai normalized matrix, kita bisa mencari beberapa properti dari matriks grayscale, antara lain *contrast*, *entropy*, *dissimilarity*, *energy*, *correlation*, dan *homogeneity*. Dalam tugas besar ini, kami menambahkan komponen *dissimilarity*, *energy*, dan *correlation*. *Dissimilarity* mengukur seberapa berbedanya dua gambar, *energy* mengukur konsentrasi nilai sinyal dalam suatu gambar, dan *correlation* mengukur hubungan statistik atau kesamaan antara nilai piksel dalam gambar. Ketiga kriteria ini memainkan peran penting dalam pemrosesan dan analisis gambar dengan memberikan ukuran kuantitatif untuk kesamaan gambar, penilaian kualitas, karakterisasi tekstur, ekstraksi fitur, pencocokan template, dan analisis gerakan. Tergantung pada aplikasi spesifiknya, kriteria yang berbeda mungkin ditekankan untuk mengekstrak informasi bermakna dari gambar dan membuat keputusan yang tepat. Properti-properti tersebut bisa dicari dengan rumus-rumus berikut:

1. *Contrast*

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

2. *Entropy*

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

3. *Dissimilarity*

$$\sum_{i,j=0}^{levels-1} P_{ij} \times |i - j|$$

4. Energy

$$\sqrt{\sum_{i,j=0}^{levels-1} P_{ij}^2}$$

5. Correlation

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

6. Homogeneity

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

Untuk mencari suatu kesamaan tekstur di antara dua gambar, kita dapat menjadikan semua 6 properti sebagai vektor 6 dimensi dan mencari nilai *cosine similarity* dengan rumus:

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

Semakin besar nilai cos nya (mendekati 1), maka semakin mirip tekstur gambarnya.

Semakin kecil nilai cos nya (mendekati 0), maka semakin tidak mirip tekstur gambarnya.

2.2 Pengembangan Website

Pengembangan Website secara dasarnya terbagi menjadi 2 bagian, yaitu *frontend* dan *backend*. *Frontend* adalah bagian program yang menghasilkan tampilan antarmuka website, sedangkan *backend* adalah bagian program yang mengatasi pemrosesan data seperti perhitungan.

Web Framework adalah struktur dasar program yang dapat meningkatkan kecepatan website, membawa fitur-fitur tambahan, serta mempercepat pembuatan website sehingga mempermudah pengembangan suatu website. Beberapa contoh framework adalah Nextjs, React, Expressjs, Django, Flask, yang semuanya memiliki kelebihan dan kekurangannya masing-masing.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

Dalam pengembangan website CBIR yang optimal dan efisien, terdapat beberapa parameter yang perlu diidentifikasi dan pertimbangkan. Pertama, kami perlu menentukan *framework* yang tepat dan bahasa yang tepat agar waktu eksekusinya tidak terlalu lama. Selain itu, pada tahap ini kami perlu menentukan fitur-fitur apa saja yang diperlukan sesuai dengan spesifikasi.

Selanjutnya, kami mengembangkan program kami sesuai dengan spesifikasi dan panduan yang ada. Pada setiap fitur, terdapat beberapa pilihan yang perlu kami buat. Program kami menggunakan dua metode pencarian gambar, yaitu fitur warna dan fitur tekstur. Pengguna dapat memilih untuk menggunakan pencarian berdasarkan warna atau berdasarkan tekstur.

Fitur warna menggunakan histogram warna dari sebuah gambar yang selanjutnya akan dipartisi menjadi blok-blok kecil $n \times n$. Program kami membagi sebuah gambar menjadi blok-blok kecil berukuran 4×4 , lalu menghitung histogram warna (dalam nilai HSV) dari blok-blok kecil tersebut, untuk selanjutnya akan dibandingkan dengan histogram warna dari gambar lain untuk mendapatkan nilai *cosine similarity* untuk setiap bloknya. Selanjutnya nilai *cosine similarity* akan dijumlahkan lalu dirata-ratakan untuk seluruh blok gambar pada dua gambar tersebut. Nilai rata-rata *cosine similarity* inilah yang akan menunjukkan tingkat kemiripan dari kedua gambar.

Fitur tekstur menggunakan matriks *grayscale*. Sehingga, pertama-tama kami perlu mendapatkan matriks tersebut dengan mengkonversi gambar ke dalam matriks RGB dan mengubahnya ke bentuk *grayscale*. Selanjutnya akan dicari matriks *co-occurrence*, hasil *transposenya*, *symmetric matrix*, dan *normalized matrix* nya. Setelah ditemukan *normalized matrix*, kami perlu mencari 6 komponen vektor, yaitu *contrast*, *homogeneity*, *entropy*, *energy*, *dissimilarity*, dan *correlation*. Hasil vektor tersebut akan dibandingkan menggunakan *cosine similarity*. Dalam fitur tekstur, kami perlu mempertimbangkan sudut dan jarak dari matriks *co-occurrence* agar bisa mendapatkan hasil yang paling optimal dan akurat. Dalam pengembangan program kali ini, kami menggunakan sudut 0° dan jarak 1. Pemilihan sudut tersebut didasarkan oleh perbandingan dengan sudut 45° , 90° , dan 135° . Dimana sudut 0° dan 90° memiliki jumlah pembanding yang lebih banyak dibandingkan sudut 45° dan 135° . Selain itu, sudut 0° memiliki kecepatan eksekusi yang relatif lebih cepat dibandingkan sudut 90° .

Setelah pengembangan fitur warna, fitur tekstur, dan bagian *frontend* selesai, kami lanjutkan dengan mengintegrasikan fitur-fitur tersebut ke websitenya (*backend*). Setelah bagian-bagian tersebut selesai, kami lanjutkan dengan tahapan menguji untuk memastikan kesesuaian program kami.

3.2 Pemetaan Masalah dalam bentuk Aljabar Geometri

3.2.1 Variabel

1. Fitur Warna

Variabel utama dari CBIR berbasis warna adalah vektor HSV yang tersusun atas histogram warna blok $n \times n$ dari sebuah gambar. Vektor HSV ini dibuat dari matriks *pixel* RGB yang diubah menjadi *pixel* HSV, lalu dihitung histogram untuk setiap blok $n \times n$.

2. Fitur Tekstur

Variabel utama dari CBIR berbasis tekstur adalah matriks *co-occurrence*. Matriks *co-occurrence* merepresentasikan hubungan spasial antar nilai *grayscale* tiap piksel.

3.2.2 Pemodelan Masalah

1. Matriks RGB dan HSV

Nilai RGB tiap pixel dari gambar akan diubah menjadi pixel bernilai HSV dengan menggunakan perhitungan RGB ke HSV.

2. Vektor Histogram HSV

Histogram HSV dibentuk dengan menghitung banyaknya nilai HSV dari setiap 4×4 blok pixel pada suatu gambar. Histogram HSV terdiri dari 14 bins dengan indeks 0 - 7 menunjukkan nilai Hue, indeks 8-11 menunjukkan nilai Saturation, dan indeks 12 - 14 adalah nilai Value.

3. Model Matriks *Co-occurrence*

Matriks *co-occurrence* dapat dinyatakan dalam vektor yang terdiri dari vektor. Dengan nilai maksimum *grayscale* adalah 255, maka matriks *co-occurrence* berukuran 256×256 (*quantization level* + 1).

4. Model Vektor Properti GLCM

Hasil 6 properti dijadikan suatu vektor dengan 6 dimensi. Vektor kedua gambar akan dibandingkan menggunakan *cosine similarity* agar bisa mendapatkan tingkat kemiripannya.

3.2.3 Penerapan Aljabar Geometri

1. Operasi Vektor dalam Mencari Kemiripan

Untuk mencari kemiripan di antara dua gambar, kami menggunakan prinsip *cosine similarity* yang diperoleh dari perkalian titik vektor dan penentuan panjang vektor.

2. Transformasi Geometri untuk Kasus Ukuran Gambar Berbeda

Untuk melakukan pemrosesan gambar dengan ukuran berbeda, dilakukan proses *resize* gambar menggunakan *library* agar perhitungan memungkinkan untuk dilakukan.

3. Ruang Warna Alternatif (HSV)

Pengubahan nilai RGB pada setiap pixel gambar menjadi nilai HSV, dan selanjutnya menghitung vektor histogram HSV dari setiap blok 4x4.

4. Penerapan Singular Value Decomposition (SVD) untuk memperoleh nilai RGB

SVD dapat melakukan dekomposisi terhadap tiga channel yang dimiliki matriks RGB.

3.3 Contoh Ilustrasi Kasus dan Penyelesaiannya

3.3.1 Kasus Warna

1. Nilai RGB pixel gambar diubah nilai-nilainya menjadi HSV.

$$\text{RGB } (12, 14, 200) = \text{HSV } (239, 0.94, 0.784)$$

2. Gambar dikuantifikasi menjadi blok berukuran 4 x 4.

Jika terdapat gambar berukuran 512×512 , maka akan terdapat $512 \times 512 / (4 \times 4) = 16.384$ blok.

3. Menghitung vektor histogram untuk tiap blok.

Misalkan tiap blok memiliki nilai HSV sebagai berikut:

$(0, 0, 0), (0, 0.98, 0), (0, 0, 0), (0, 0.63, 0),$

$(1, 0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 1)$

....

Maka akan menjadi $(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0)$, dengan 8 indeks pertama adalah nilai Hue dengan banyak kemunculannya, 3 indeks selanjutnya adalah nilai Saturation dengan banyak kemunculannya, dan 3 indeks terakhir adalah nilai Value dengan banyak nilai kemunculannya.

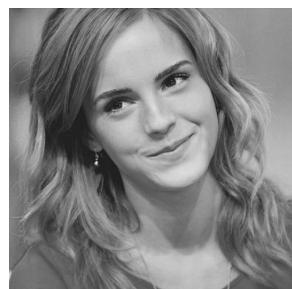
4. Perhitungan *cosine similarity*

Menghitung nilai *cosine similarity* antara dua vektor untuk setiap bloknya, lalu merata-ratakan total nilai *cosine similarity* tersebut dengan jumlah bloknya.

$$\cos\{(1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0), (1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0)\} = 1$$

3.3.2 Kasus Tekstur

1. Gambar diubah ke dalam bentuk grayscale.



2. Gambar grayscale dinyatakan dalam bentuk matriks

0	1	85
68	176	34
123	126	194
27	131	127

170	238	85
68	2	0
12	87	234

3. Buat matrix co-occurrence

	0	1	2	...	256
0	0	1	0	...	0
1	0	0	0	...	0
2	0	0	0	...	0
...
256	0	0	0	...	0

	0	1	2	...	256
0	0	0	0	...	0
1	0	0	0	...	0
2	1	0	0	...	0
...
256	0	0	0	...	0

4. Buat matriks transpose nya

	0	1	2	...	256
0	0	0	0	...	0
1	1	0	0	...	0
2	0	0	0	...	0
...
256	0	0	0	...	0

	0	1	2	...	256
0	0	0	1	...	0
1	0	0	0	...	0
2	0	0	0	...	0
...
256	0	0	0	...	0

5. Buat symmetric matrix nya

	0	1	2	...	256
0	0	1	0	...	0
1	1	0	0	...	0
2	0	0	0	...	0
...
256	0	0	0	...	0

	0	1	2	...	256
0	0	0	1	...	0
1	0	0	0	...	0
2	1	0	0	...	0

...
256	0	0	0	...	0

6. Buat normalized matrix nya

	0	1	2	...	256
0	0	0.125	0	...	0
1	0.125	0	0	...	0
2	0	0	0	...	0
...
256	0	0	0	...	0

	0	1	2	...	256
0	0	0	0.1666	...	0
1	0	0	0	...	0
2	0.1666	0	0	...	0
...
256	0	0	0	...	0

7. Cari nilai vektornya

Vektor Gambar 1: <contrast(matrix1), homogeneity(matrix1), entropy(matrix1), dissimilarity(matrix1), energy(matrix1), correlation(matrix1)>

Vektor Gambar 2: <contrast(matrix2), homogeneity(matrix2), entropy(matrix2), dissimilarity(matrix2), energy(matrix2), correlation(matrix2)>

8. *Cosine Similarity*

$$\cos = 0.9983425$$

Persen kemiripan = 99.83425%

BAB IV

IMPLEMENTASI DAN UJI COBA

4.1 Implementasi Program Utama

4.1.1 Color

Procedure/Function	Kegunaan
void convertHSV(vector<vector<HSV>> *hsv, const float* imgVector, int height, int width)	Fungsi digunakan untuk mengubah sebuah <i>pixel</i> RGB menjadi HSV.
vector<vector<HSV>> convertOriginalImagetohSV(const string &filename)	Fungsi digunakan untuk <i>parsing</i> seluruh <i>pixel</i> dari gambar yang dimasukkan pengguna (untuk dibandingkan dengan <i>dataset</i>) dan mengubah seluruh <i>pixel</i> -nya menjadi bernilai HSV.
vector<vector<HSV>> convertQueryImagetohSV(const string &filename, int expectedHeight, int expectedWidth)	Fungsi untuk <i>parsing</i> seluruh gambar dari <i>dataset</i> dan mengubah nilai <i>pixel</i> gambar menjadi HSV dan melakukan <i>resize</i> agar sama dengan gambar yang di-input pengguna.
string hashFunction(int histogram[14])	Fungsi untuk menyimpan histogram warna menjadi suatu string untuk meningkatkan efektivitas program.
vector<vector<int>> countHistogramOriginal(vector<vector<HSV>> hsv, int size)	Fungsi untuk menghitung nilai histogram warna dari gambar yang dimasukkan pengguna.
vector<string> countHistogram(vector<vector<HSV>> hsv, int size)	Fungsi untuk menghitung nilai histogram warna dari <i>dataset</i> dan melakukan <i>hashing</i> agar menjadi string
int getNumberofFiles(const string& path)	Menghitung banyak files dalam dataset.
void queryDatasetIntoJSON(const string& path, int heightImageInserted, int	Melakukan <i>parsing</i> dataset dan mengubahnya menjadi vektor histogram, untuk selanjutnya <i>di-hash</i> dan dimasukkan sebagai <i>cache</i> .

widthImageInserted)	
float cos(vector<int> A, vector<int> B)	Menghitung nilai <i>cosine similarity</i> dari dua blok gambar.
vector <int> unhash(const string& s)	<i>Unhash</i> string menjadi vektor histogram kembali.
float cosImage(vector <string> elem, vector <vector <int>> &histogramOri)	Menghitung <i>cosine similarity</i> dari dua buah gambar,
void calculateResult(const string &pathQuery, const string &pathJSON, vector<vector<int>> histogramOri, vector <Result> *resultVector)	Melakukan <i>parsing cache</i> dari JSON dan menghitung <i>cosine similarity</i> dari semua gambar di dataset dengan gambar yang di-input pengguna.
void saveResulttoJSON(vector <Result> &result, const string& filename)	Menyimpan hasil perbandingan gambar dalam sebuah JSON.
int binarySearch(const vector<Result>& a, float item, int low, int high), void sortResult(vector<Result>& a)	Mengurutkan hasil perbandingan gambar dari yang bernilai <i>cosine similarity</i> terbesar ke yang terkecil menggunakan <i>binary insertion sort</i> .

```

ALUR PROGRAM UTAMA
(main)
    input(pathImage) // Input gambar dari pengguna
    input(folder) // Input dataset
    histogramOriginal =
    countHistogramOriginal(convertOriginalImagetoHSV(pathImage))
    // Melakukan konversi gambar ke HSV sekaligus menghitung histogramnya

    if (not exist(fileJSON)):
        queryDataset(folder)
    // Cek apakah cache sudah ada, jika belum akan dibuat

    calculateResult(histogramOriginal, fileJSON)

```

```
// Kalkulasi cosine similarity dari vektor histogram
gambar utama dan setiap gambar dataset

saveResulttoJSON(outputJSON)
sortResult(outputJSON)
// Menyimpan hasil cosine similarity di JSON dan
diurutkan berdasarkan tingkat kemiripan
```

4.1.2 Texture

Procedure/Function	Kegunaan
vector<vector<double>> grayscale(const vector<vector<vector<int>>>& img)	Fungsi digunakan untuk mengkonversi nilai RGB menjadi nilai grayscale sesuai dengan rumus. Fungsi memiliki parameter gambar RGB dengan bentuk array of array of array.
vector<vector<double>> imageToGray(const string& path)	Fungsi digunakan untuk menerima input path dan mengecek keberadaan gambar tersebut. Apabila gambar ada, maka gambar RGB akan diubah ke dalam bentuk matriks grayscale dengan memanggil fungsi grayscale.
vector<vector<double>> cooc0(const vector<vector<double>>& grayscale)	Fungsi digunakan untuk mencari nilai matriks co-occurrence dengan sudut 0 dan jarak 1. Fungsi mengembalikan bentuk matriks.
vector<vector<double>> transpose(const vector<vector<double>>& matrix)	Fungsi digunakan untuk melakukan transpose terhadap matriks co-occurrence. Fungsi mengembalikan bentuk matriks.
vector<vector<double>> symmetric(const vector<vector<double>>& cooc, const vector<vector<double>>& transpose)	Fungsi digunakan untuk mencari symmetric matrix dengan menjumlahkan nilai matriks co-occurrence dan transpose nya. Fungsi mengembalikan bentuk matriks.
double sumMatrix(vector<vector<double>> matrix)	Fungsi digunakan untuk mencari nilai jumlah semua elemen pada matriks.

<code>vector<vector<double>> normalize(const vector<vector<double>>& matrix)</code>	Fungsi digunakan untuk mencari nilai normalized matrix. Fungsi mengembalikan bentuk matriks.
<code>double contrast(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti contrast dari gambar.
<code>double homogeneity(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti homogeneity dari gambar.
<code>double entropy(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti entropy dari gambar.
<code>int absolut(int x)</code>	Fungsi mencari nilai absolut dari suatu integer.
<code>double dissimilarity(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti dissimilarity dari gambar.
<code>double asmProp(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti ASM dari gambar.
<code>double energy(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti energy dari gambar.
<code>double correlation(const vector<vector<double>>& matrix)</code>	Fungsi mencari properti correlation dari gambar.
<code>double cosine(const vector<vector<double>>& matrix1, const vector<vector<double>>& matrix2)</code>	Fungsi mencari nilai cosine similarity dari kedua gambar yang sudah berbentuk matriks. Fungsi mengembalikan nilai cos dikalikan dengan 100 (persen).
<code>vector<vector<double>> glcm(const vector<vector<double>>& image)</code>	Fungsi mencari matriks glcm dengan memanggil fungsi cooc0, transpose, symmetric, dan normalize.
<code>void</code>	Fungsi melakukan proses glcm kepada semua

queryAllImage(vector<vector<double>> glcmimage1, const string& path, Result *hasil)	gambar pada folder.
int binarySearch(vector<double> a, float item, int low, int high)	Algoritma binary search.
void sortResult(Result *a)	Fungsi untuk mengurutkan hasil secara descending dan menyimpan nilai-nilai dengan persen lebih besar dari 60% di suatu array.

```

ALUR PROGRAM UTAMA
(main)
    input(pathImage)
    input(folder)

    gray = imagetoGray(pathImage)
    //sesuai rumus
    glcmImage = glcm(gray)
    //matrix co-occurrence, transpose, symmetric, normalize

    queryAllImage(glcmImage, folder, hasil)
    //menghasilkan nilai cos dari semua image pada folder dan
    menyimpan hasil dengan nilai cos lebih besar dari 60%

    sortResult(hasil)
    //mengurutkan hasil secara descending

```

4.2 Penjelasan Struktur Program Berdasarkan Spesifikasi

Program terbagi menjadi 2 yaitu Program Website dan Program Utama. Program Utama adalah Program yang melakukan pengolahan data gambar dan melakukan pencarian.

Struktur Program Website terbagi menjadi 2 bagian utama yaitu *frontend* dan *backend*. Untuk *frontend* kami menggunakan framework Nextjs sedangkan *backend* menggunakan framework Express atau Nodejs. Frontend dapat memanggil *backend* untuk melakukan pemrosesan terhadap data dan mengembalikan informasi dari hasil proses tersebut. *Backend* sendiri dapat mengeksekusi program utama yang melakukan pencarian gambar. *Backend* tersusun atas

fungsi-fungsi yang melakukan pemrosesan pengolahan data berupa gambar yang di-input oleh pengguna.

Hasil dari pencarian baik menggunakan warna atau tekstur akan disimpan pada file json bernama ‘query.json’. File ini dapat langsung dibaca oleh Nextjs untuk menampilkan hasil pencarian oleh program utama.

4.2.1 Website

Nextjs menggunakan sistem App Router untuk lakukan routing, yaitu menjalin hubungan antara client-side dan server-side yang efisien. *Rendering* adalah proses pengubahan program menjadi tampilan yang dapat dilihat oleh pengguna, setiap kali diinginkan perubahan tampilan maka perlu dilakukan pengulangan *render*. App Router Nextjs, mencegah *re-rendering* component-component serta status program yang tidak berubah, sehingga website lebih cepat.

Nextjs dapat menggunakan logika untuk melakukan *render* atau suatu panggilan ke *backend*. Logika pada Nextjs dapat melakukan pencegahan error, pencegahan aksi seperti button, dan pencegahan pemanggilan ke *backend*.

Express dapat menerima panggilan dan mengembalikan informasi dari panggilan tersebut menggunakan server *end-points*. Express juga dapat mengirimkan data gambar sehingga dapat ditampilkan oleh frontend. End-point Express dapat dipanggil oleh frontend, untuk melakukan suatu proses pada data server. Terdapat end-point untuk masing-masing keperluan antara lain, upload gambar dan dataset, melakukan pencarian berdasarkan warna, melakukan pencarian berdasarkan tekstur, menyimpan data nama-nama file isi dari dataset, menyimpan data nama gambar.

4.2.2 Fungsi Pengolahan Warna

Fungsi pengolahan warna merupakan program yang digunakan untuk mencari kesamaan antara gambar yang dimasukkan pengguna dengan dataset gambar. Fungsi pengolahan warna melakukan pengubahan tipe warna dari RGB ke HSV, untuk selanjutnya dilakukan kuantifikasi blok berukuran 4 x 4 dari gambar. Dari setiap blok-blok gambar akan dihitung vektor histogram HSV dan dilakukan rata-rata untuk keseluruhan vektor histogram pada dua gambar tersebut.

4.2.3 Fungsi Pengolahan Tekstur

Fitur tekstur menggunakan matriks *grayscale*. Sehingga, pertama-tama kami perlu mendapatkan matriks tersebut dengan mengkonversi gambar ke dalam matriks RGB dan mengubahnya ke bentuk *grayscale*. Selanjutnya akan dicari matriks *co-occurrence*, hasil transposenya, *symmetric matrix*, dan *normalized matrix* nya. Setelah ditemukan *normalized matrix*, kami perlu mencari 6 komponen vektor, yaitu *contrast*, *homogeneity*, *entropy*, *energy*, *dissimilarity*, dan *correlation*. Hasil vektor tersebut akan dibandingkan menggunakan *cosine similarity*. Pada implementasinya, kami membuat beberapa fungsi antara lain, fungsi untuk mengubah gambar menjadi *grayscale*, pembuatan matriks *co-occurrence*, transpose matriks, pembuatan *symmetric matrix*, pembuatan *normalized matrix*, penentuan properti *contrast*, penentuan properti *homogeneity*, penentuan properti *entropy*, penentuan properti *dissimilarity*, penentuan properti *energy*, penentuan properti *correlation*, pencarian nilai absolut, pencarian jumlah elemen matriks, *binary search*, *sorting*, serta *process all image*.

4.3 Tata Cara Penggunaan Program

- Mode manual
 - 1. Upload dataset
 - 2. Masukan gambar yang akan dicari
 - 3. Pilih mode warna atau tekstur
 - 4. Cari gambar dengan klik tombol search
- Mode web camera
 - 1. Upload dataset
 - 2. Pilih mode warna atau tekstur
 - 3. Pilih mode pencarian manual atau otomatis
 - 4. Start capture
 - 5. Jika manual, klik tombol Search

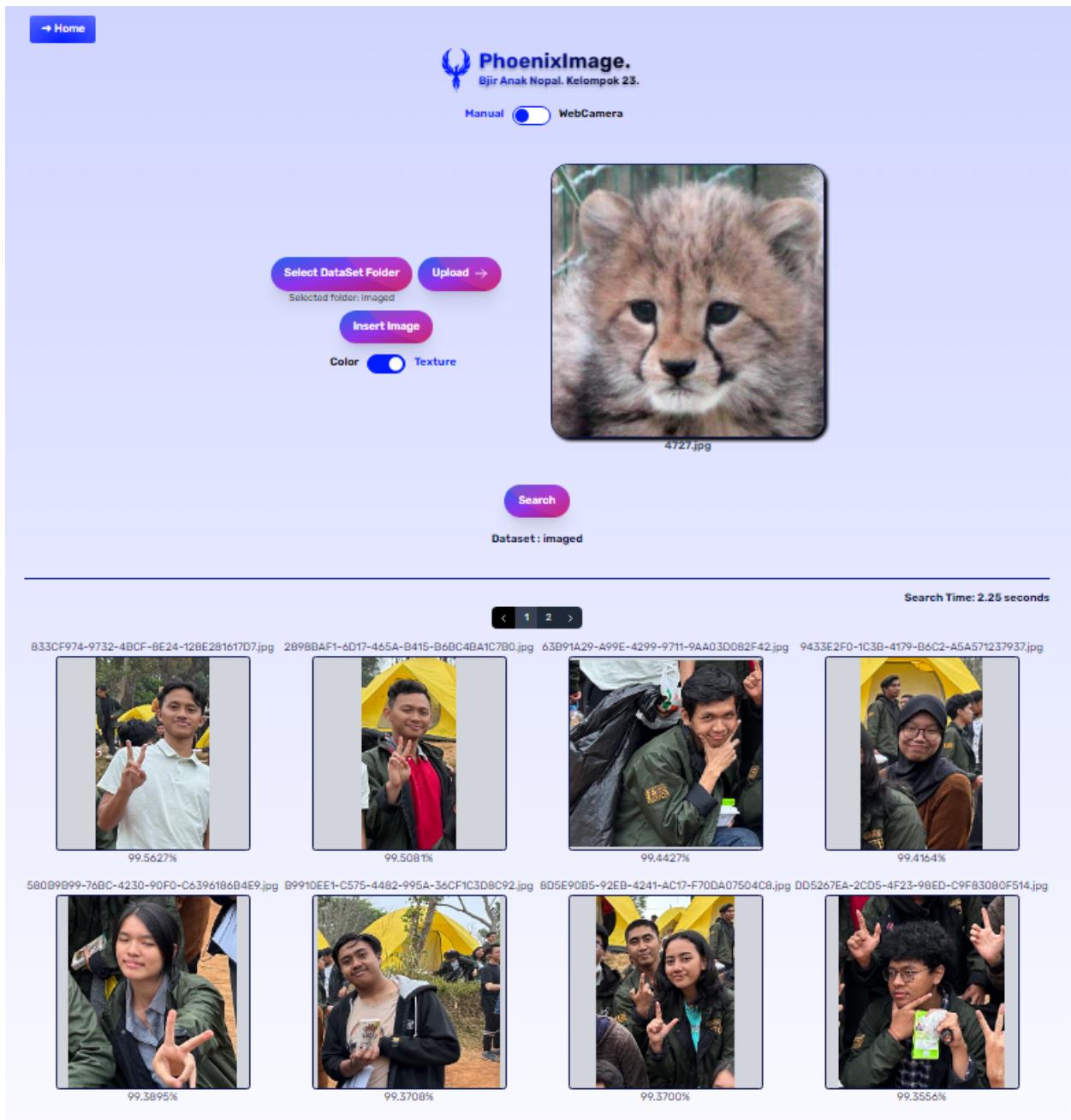
4.4 Hasil Pengujian

4.4.1 Uji Tekstur

4.4.1.1 Uji pencarian gambar yang terdapat pada dataset

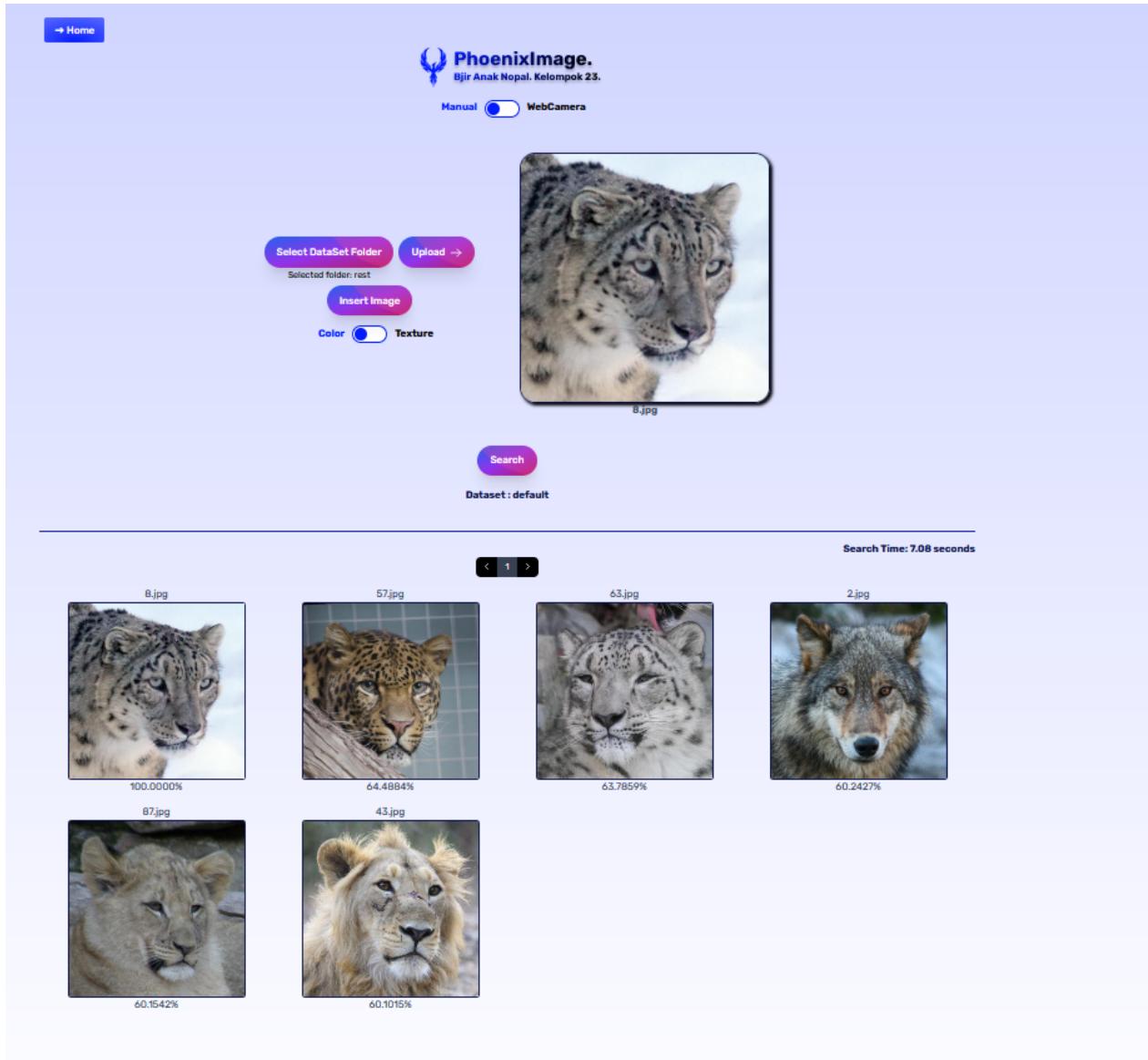


4.4.1.2 Uji pencarian gambar yang tidak terdapat pada dataset

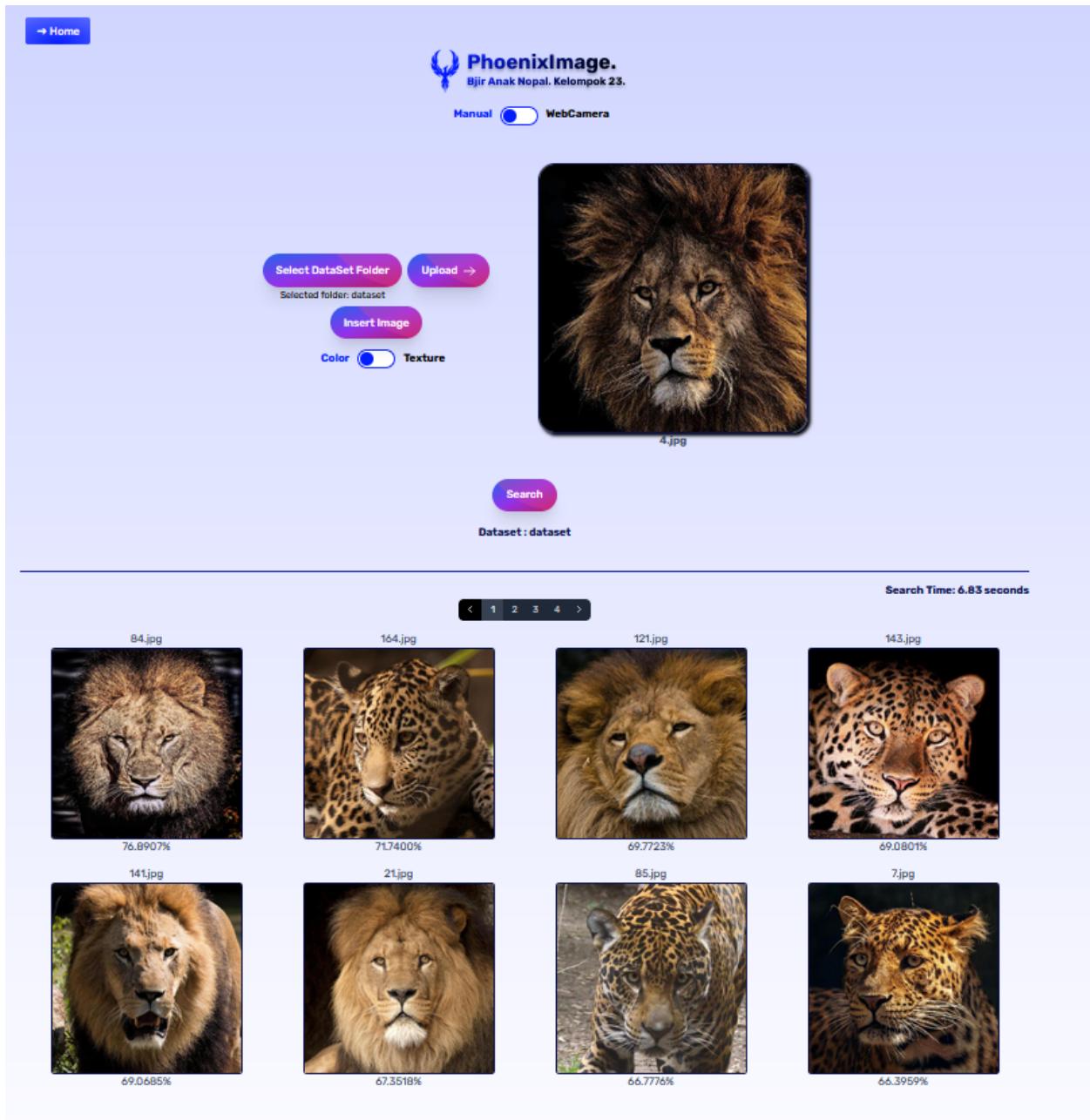


4.4.2 Uji Warna

4.4.2.1 Uji pencarian gambar yang terdapat pada dataset



4.4.2.2 Uji pencarian gambar yang tidak terdapat pada dataset



4.5 Analisis dari Desain Solusi Algoritma Pencarian

Berdasarkan hasil pengujian yang kami lakukan, kami rasa pencarian berdasarkan warna lebih efektif dalam membedakan gambar. Hal ini dikarenakan tekstur gambar yang cenderung seringkali memiliki tingkat kemiripan yang tinggi, sementara warna gambar belum tentu

memiliki tingkat kemiripan yang serupa. Maka dari itu, pencarian berdasarkan warna cenderung lebih baik dalam kebanyakan kasus.

Menurut hasil pengujian kami, tekstur gambar memiliki kemiripan hampir semua di atas 90% pada semua gambar dataset. Sementara, pencarian warna tidak seperti itu. Maka, apabila ingin melihat perbedaan pada dataset yang banyak, maka akan lebih baik jika menggunakan pencarian berdasarkan warna.

Sementara itu, proses mekanisme pencarian berdasarkan histogram warna memiliki akurasi yang bervariasi bergantung pada nilai histogram pada tiap blok-bloknya.

BAB V

KESIMPULAN

5.1 Kesimpulan

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Salah satu kontennya adalah *color* dan *texture*. Kedua komponen tersebut merupakan komponen pembentuk suatu gambar yang sangatlah penting. CBIR berdasarkan warna dapat diperoleh dengan konversi menjadi HSV. CBIR berdasarkan tekstur dapat diperoleh dengan memperoleh nilai *contrast*, *homogeneity*, *entropy*, *dissimilarity*, *energy*, dan *correlation*. Pengecekan kesamaan dari dua gambar dapat diperoleh dengan metode *cosine similarity*.

Metode-metode tersebut dapat diimplementasikan dalam bentuk *website*. Sehingga, pengguna dapat mengupload folder *dataset* dan gambar utama. Dari unggahan tersebut, *website* akan menampilkan gambar-gambar yang memiliki kesamaan diatas 60% dengan urutan *descending*.

Pada tugas besar ini, kita perlu membuat implementasi dari kebutuhan tersebut. Kami menggunakan nextjs, nodejs, express, tailwind css, framer-motion, react, cors, multer, dan bahasa C++ (serta library stb dan json) dalam implementasinya. Hasil kami menghasilkan website yang dapat menerima upload folder dan gambar kemudian menghasilkan persen kesamaan antara gambar utama dengan gambar dari dataset. Analisis kami juga menyatakan bahwa pencarian berdasarkan warna lebih efektif pada beberapa kasus tertentu karena bisa lebih menunjukkan perbedaan di antara gambar-gambar pada dataset.

5.2 Saran dan Refleksi

Dalam pengeraannya, untuk memenuhi run time yang ditetapkan pada spesifikasi, kami perlu menggunakan bahasa C++ padahal pada mulanya kami menggunakan bahasa Python. Perubahan ini menyebabkan kita untuk beradaptasi dengan bahasa C++ mengingat bahasa ini belum diajarkan selama perkuliahan kami. Tugas ini juga mewajibkan kami untuk membuat website dalam waktu yang relatif singkat mengingat kami belum berpengalaman dalam hal ini. Kemunculan tugas ini juga bersamaan dengan tugas besar lain sehingga menyebabkan fokus kami untuk terbagi.

Dalam pengeraan tugas besar ini, diperlukan ketelitian dalam pembacaan spek sehingga tidak salah dalam implementasinya. Selain itu, perlu dieksplor lebih banyak skema atau kondisi,

sehingga semua kondisi dapat ditemukan penyelesaian yang tepat (*error handling*). Dalam penggerjaan tugas besar ini, kurangnya komunikasi dapat menyebabkan miskomunikasi dalam progres masing-masing anggota, sehingga kedepannya hal tersebut perlu kami perbaiki. Melalui tugas besar ini, kami memahami pentingnya kemampuan kerja sama, koordinasi, dan komunikasi yang baik.

5.3 Komentar dan Ruang Perbaikan

- Mengingat banyaknya tugas besar yang sedang berjalan, alangkah baiknya apabila tenggat waktu dari penggerjaan tugas besar dapat diperlama.
- Ada banyak sekali ruang untuk perbaikan dari penggerjaan tugas besar kali ini. Koordinasi yang lebih baik dapat meningkatkan kinerja.

DAFTAR PUSTAKA

Informatika.stei.itb.ac.id. (2023). Review matriks. Diakses pada 4 November 2023, dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf>

Yunus, Muhammad. (2020). Feature Extraction : Gray Level Co-occurrence Matrix (GLCM). Diakses pada 4 November 2023, dari <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1>

Next.js. Next.js. Diakses pada 1 November 2023, dari <https://nextjs.org/>
tailwindcss. Tailwind CSS. Diakses 1 November 2023, dari <https://tailwindcss.com/>
Framer Motion. Documentation. Diakses 1 November 2023, dari <https://www.framer.com/motion/>
npm. react-webcam. Diakses 12 November 2023, dari <https://www.npmjs.com/package/react-webcam>

Express. Express. Diakses 1 November 2023, dari <https://expressjs.com/>
npm. cors. Diakses 12 November 2023, dari <https://www.npmjs.com/package/cors>
npm. multer. Diakses 12 November 2023, dari <https://www.npmjs.com/package/multer>

Link repository: <https://github.com/NoHaitch/Algeo02-22063>

Link video: https://youtu.be/8oIWbBt_jSQ