

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



Oleh

Muhammad Zaki	13522136
Muhammad Dzaki Arta	13522149
Albert Ghazaly	13522150

Kelompok: masih bisa turu

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

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.

BAB 2

LANDASAN TEORI

2.1 Dasar Teori Secara Umum

Dasar dasar teori yang digunakan pada Aplikasi aljabar vektor dalam sistem temu balik gambar adalah sebagai berikut

2.1.1 CBIR dengan parameter warna

Pada bagian ini menyajikan metode untuk mengekstrak fitur warna suatu gambar dengan cepat untuk pengambilan gambar berbasis konten (CBIR). Pertama, ruang warna HSV diukur secara rasional. Fitur histogram warna dan tekstur berdasarkan matriks kejadian bersama diekstraksi untuk membentuk vektor fitur. Kemudian karakteristik histogram warna global, histogram warna lokal dan fitur tekstur dibandingkan dan dianalisis CBIR. Berdasarkan karya-karya tersebut, sistem CBIR dirancang menggunakan fitur-fitur yang menyatu dengan warna dan tekstur dengan membangun bobot vektor fitur. Eksperimen pengambilan fitur yang relevan menunjukkan bahwa pengambilan fitur gabungan memberikan kesan visual yang lebih baik dibandingkan pengambilan fitur tunggal, yang berarti hasil pengambilan lebih baik. Untuk menghitung CBIR dengan parameter warna, ada beberapa langkah yang dapat dilakukan diantaranya.

1. Pertama kita blok gambar menjadi 4x4 bagian.
2. Konversi gambar dari BGR menjadi HSV

Dalam mengkonversi gambar dari RGB menjadi HSV ada beberapa langkah yaitu

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

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

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

$$V = C_{max}$$

3. Selanjutnya kita melakukan proses kuantifikasi dengan menggunakan formula

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

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

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

4. Pada tahap ketiga kita menghitung banyak setiap nilai yang ada pada fitur
5. Pada tahap ini kita menghitung Histogram dari banyak nilai yang sudah kita hitung pada tahap ke 3

6. Pada tahap ini kita menghitung kesamaan dari 2 histogram representasi 2 gambar untuk setiap sub gambar dengan menggunakan rumus 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 Histogram dan n adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*.

7. Setelah didapat hasil dari Cosine Similarity dari setiap sub gambar. Kita hitung nilai rata rata dari setiap nilai Cosine Similarity yang kita dapat

2.1.2 CBIR dengan parameter tekstur

CBIR dengan perbandingan tekstur dilakukan menggunakan suatu matriks yang dinamakan *co-occurrence matrix*. Matriks ini digunakan karena dapat melakukan pemrosesan yang lebih mudah dan cepat. Vektor yang dihasilkan juga mempunyai ukuran yang lebih kecil. Misalkan terdapat suatu gambar I dengan $n \times m$ piksel dan suatu parameter offset ($\Delta x, \Delta y$), Maka dapat dirumuskan matriksnya sebagai berikut:
Dengan menggunakan nilai i dan j sebagai nilai intensitas dari gambar dan p serta q sebagai posisi dari gambar, maka *offset* Δx dan Δy bergantung pada arah θ dan jarak yang digunakan melalui persamaan di bawah ini.

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

Melalui persamaan tersebut, digunakan nilai θ adalah $0^\circ, 45^\circ, 90^\circ$, dan 135° .

Langkah - Langkah dalam menghitung CBIR dengan parameter tekstur

1. Pembuatan framework matrix
2. Konversi dari warna menjadi grayscale dengan menggunakan rumus:

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

3. Pembuatan co-occurrence matrix (mengisi framework matrix)

Dalam pembuatan co-occurrence matrix saya menggunakan 0° karena lebih mudah dalam pengaplikasiannya dan juga hasilnya tidak terlalu jauh jika menggunakan rata-rata dari beberapa parameter derajat lainnya.

4. Pembuatan symmetric matrix (penjumlahan co-occurrence matrix dengan transpose matrix)
5. Selanjutnya kita dapat membuat matrix yang dinormalisasi dengan menggunakan rumus

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

6. Selanjutnya kita dapat menghitung nilai contrast,homogeneity,entropy,ASM,dan juga energi dari matrix normalization tadi.

Untuk menghitung nilai contrast kita menggunakan rumus :

Contrast:

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

Contrast digunakan untuk mengukur variasi intensitas piksel yang ada dalam citra. Semakin tinggi nilai kontras, semakin besar perbedaan intensitas antara piksel yang berdekatan.

Untuk menghitung nilai homogeneity kita menggunakan rumus :

Homogeneity :

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

Homogeneity menunjukkan seberapa seragam atau "halus" tekstur dalam citra. Semakin tinggi homogenitas, semakin seragam distribusi intensitas pixelnya

Untuk menghitung nilai entropy kita menggunakan rumus :

Entropy :

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

Entropy mencerminkan tingkat ketidakpastian atau keacakan dalam distribusi intensitas piksel. Jika entropi tinggi, maka variasi intensitas pikselnya juga tinggi

Untuk menghitung ASM kita dapat menggunakan rumus:

$$\text{'ASM': } \sum_{i,j=0}^{levels-1} P_{i,j}^2$$

ASM digunakan untuk mengukur homogenitas tekstur dengan mempertimbangkan kekuatan dan kejelasan distribusi intensitas piksel dalam GLCM

Untuk menghitung Energy dapat menggunakan rumus:

$$\text{'energy': } \sqrt{ASM}$$

Energy sejalan dengan ASM, mengukur homogenitas tekstur dengan fokus pada distribusi intensitas piksel. Semakin tinggi energi, semakin homogen distribusi intensitasnya.

Untuk menghitung dissimilarity dapat menggunakan rumus:

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

Dissimilarity menunjukkan sejauh mana perbedaan antara piksel-piksel dalam citra. Semakin besar nilai dissimilarity, semakin besar perbedaan antar piksel.

7. Selanjutnya kita mendapatkan vektor yang terdiri dari nilai tersebut
8. Hitung similarity dari 2 image dengan menggunakan cosine-similarity

Untuk menghitung nilai dari cosine-similarity kita menggunakan rumus :

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

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

2.2 Penjelasan Singkat Mengenai Pengembangan Sebuah Website

Kami dalam membangun web ini menggunakan react sebagai front end dan juga python di dalam Flask API sebagai backend. Kami menggunakan react dikarenakan banyaknya fitur yang ditawarkan oleh react yang memudahkan untuk kostumisasi sebuah tampilan sebuah web itu sendiri. Sedangkan kami menggunakan Flask API dikarenakan kami lebih familiar menggunakan bahasa pemrograman python untuk mendesain algoritma CBIR kami, dan juga karena kemudahan dalam pengaturan endpoint-endpoint API yang diperlukan untuk menghubungkan algoritma Content-Based Image Retrieval (CBIR) yang telah dibuat dalam Python dengan bagian front end yang dibangun menggunakan React.

Dengan memanfaatkan keunggulan masing-masing teknologi ini, kami bertujuan untuk menghasilkan sebuah aplikasi web yang menggabungkan kemampuan visual dari React dengan

kecanggihan algoritma CBIR yang telah dirancang menggunakan Python di dalam Flask API. Hal ini memungkinkan kami untuk memberikan pengalaman pengguna yang unik dan interaktif dalam pencarian gambar berbasis konten (CBIR) sambil memanfaatkan kelebihan yang ditawarkan oleh kedua teknologi ini.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah Langkah Pemecahan Masalah

Dalam memecahkan masalah kami pertama tama mengidentifikasi terlebih dahulu bentuk atau representasi struktur data yang sesuai untuk memecahkan permasalahan tersebut. Selanjutnya kami melakukan analisis mendalam terhadap data yang ada, mengidentifikasi potensi ketidakteraturan atau kekacauan dalam representasi data, dan menentukan strategi untuk merapikan atau mengorganisir nya agar dapat diolah lebih efisien.

Setelah mengidentifikasi bentuk struktur data yang sesuai, langkah selanjutnya adalah mengembangkan pendekatan atau algoritma yang tepat untuk memanipulasi data tersebut sesuai dengan kebutuhan. Kami mempertimbangkan berbagai teknik atau metode yang dapat diterapkan, baik itu teknik klasik maupun pendekatan inovatif, untuk mencapai solusi yang optimal.

Selanjutnya, kami melakukan implementasi dari algoritma atau pendekatan yang telah dirancang ke dalam kode atau program komputer. Proses implementasi ini melibatkan penggunaan berbagai bahasa pemrograman atau library yang sesuai dengan kebutuhan, serta melakukan uji coba secara teratur untuk memastikan keberfungsiannya dengan baik.

3.2 Proses Pemetaan Masalah Menjadi Elemen-Elemen Pada Aljabar Geometri.

Setelah kami mengidentifikasi permasalahan yang terjadi kami memutuskan untuk menggunakan elemen elemen pada aljabar geometri seperti matrix untuk menyimpan berbagai data yang kami butuhkan seperti data image, matrix co occurrence,matrix GLCM, vektor yang digunakan untuk perhitungan cosine similarity.

Pertama, kami menggunakan matriks untuk menyimpan data gambar itu sendiri. Setiap elemen dalam matriks tersebut merepresentasikan nilai piksel dari gambar dalam sebuah susunan

yang terstruktur, memungkinkan kami untuk melakukan manipulasi dan analisis pada data gambar dengan efisien.

Kemudian, kami menggunakan matriks co-occurrence ataupun GLCM (Gray-Level Co-occurrence Matrix) pada CBIR Texture untuk merepresentasikan distribusi nilai piksel dan hubungan spasial antara piksel-piksel dalam gambar. Matriks ini memberikan informasi tentang frekuensi kemunculan pasangan nilai piksel tertentu, yang berguna untuk analisis tekstur dan pola dalam gambar. Sedangkan pada CBIR Warna kami menggunakan quantify untuk mengubah range pada matrix HSV menjadi lebih kecil dan membuat histogram dalam bentuk vektor dengan h,s,dan v sebagai bins dari histogram tersebut.

Selanjutnya, kami menggunakan vektor untuk menghitung similarity antara gambar menggunakan metode cosine similarity. Vektor ini memuat representasi fitur atau atribut penting dari gambar yang digunakan dalam perhitungan untuk menentukan seberapa mirip atau berbedanya dua gambar.

Dengan menggunakan elemen-elemen aljabar geometri ini, kami dapat memanfaatkan struktur matematis yang terdefinisi dengan baik untuk mengatur, menyimpan, dan mengolah data gambar serta informasi terkait

3.3 Ilustrasi Kasus dan Penyelesaiannya

3.3.1 Masalah nilai 0 yang digunakan untuk CBIR Texture

Kami menggunakan 0 derajat untuk membangun Gray-Level Co-occurrence matrix dikarenakan hasil yang didapat tidak terlalu berbeda jika kami menggunakan nilai rata rata dari sudut 0° , 45° , 90° , 135° , 180° , 225° , 270° , atau 315° . Selain itu menurut kami menggunakan 0 derajat memudahkan dalam membangun matrix glcm. Sudut 0 derajat mewakili arah horizontal, yang memudahkan untuk memahami orientasi dan posisi piksel dalam gambar. Ini membuat proses pemrosesan dan interpretasi GLCM menjadi lebih sederhana. Dalam segi komputasi menggunakan satu sudut saja (0 derajat) dapat mempercepat komputasi karena hanya memerlukan iterasi pada satu arah.

3.3.2 Masalah pembagian blok pada CBIR Warna

Dalam pencarian blok agar lebih efektif kami menggunakan 4x4 blok. Namun tidak semua gambar memiliki jumlah baris dan kolom yang habis dibagi 4 oleh karena itu untuk blok sisa blok gambar yang tidak habis dibagi oleh 4 kami letak pada blok terakhir, contoh ada sebuah gambar dengan resolusi 26x37 maka pada akan ada 9 blok gambar yang memiliki resolusi 6x9, 3 gambar yang memiliki resolusi 6x10, 3 gambar yang memiliki resolusi 8x9, dan 1 gambar yang memiliki resolusi 8x10.

3.3.2 Masalah waktu komputasi yang digunakan untuk memproses CBIR

Untuk mempersingkat waktu pemrosesan dari CBIR kami kami menggunakan library dari python yaitu NumPy dan juga multiprocessing. Untuk mempercepat waktu pemrosesan dalam Content-Based Image Retrieval (CBIR). NumPy memberikan efisiensi tinggi dalam manipulasi array numerik yang merepresentasikan data gambar, memungkinkan operasi cepat pada array besar serta vektorisasi yang mengurangi kebutuhan akan loop elemen. Sementara itu, modul multiprocessing memungkinkan kami untuk membagi tugas pemrosesan ke beberapa proses yang berjalan secara paralel. Dengan memanfaatkan kekuatan CPU secara maksimal, kami dapat melakukan ekstraksi fitur, manipulasi gambar, dan operasi pemrosesan lainnya secara efisien. Kombinasi keduanya memungkinkan kami untuk meningkatkan kinerja CBIR dengan mendistribusikan tugas-tugas pemrosesan ke berbagai proses secara simultan, mengurangi waktu eksekusi secara signifikan.

BAB 4

IMPLEMENTASI DAN UJI COBA

4.1 Implementasi Program Utama

4.1.1 Pseudocode CBIR Warna

```
Program color
{Program untuk mencari similarity berdasarkan warna dari gambar}

Use numpy
Use cv2
Use multiprocessing

KAMUS UMUM

function imread(image : Image)-> array
{fungsi dari library cv2 yang menerima image dan mengembalikan Matrix BGR}
function divide(matrix : array, num : float) -> numpy array
{fungsi dari library numpy yang menerima matrix dan membagi matrix tersebut dengan num}
function max(matrix : array, n: integer) -> numpy array
{fungsi yang menerima matrix dan mengembalikan matrix yang berisi nilai maksimum pada axis ke n}
function min(matrix : array, n: integer) -> numpy array
{fungsi yang menerima matrix dan mengembalikan matrix yang berisi nilai minimum pada axis ke n}
function subtract(matrix1,matrix2 : array) -> numpy array
{fungsi untuk mengurangi matrix1 dan matrix2}
function logical_and(Matrix1, Matrix2,... : array) -> numpy array
{fungsi yang menerima beberapa matrix yang berisi boolean dan melakukan operasi and pada kedua matrix tersebut}
function count_nonzero(matrix : Matrix) -> Integer
{sebuah fungsi yang menerima matrix dan mengirim banyak elemen yang bukan nol pada matrix tersebut}
function append(input object: Object)
{prosedur bawaan python untuk menambahkan object pada matrix}
function dot(Vektor1, Vektor2 : array) -> float
{fungsi bawaan numpy yang menerima 2 buah vektor dan melakukan operasi dot pada kedua vektor tersebut}
function norm(vektor : array) -> float
{fungsi bawaan numpy yang menerima array dan menghasilkan norma dari sebuah vektor atau matrix}
function len(Array : array) -> integer
{fungsi bawaan python yang menerima array dan mengembalikan panjang array tersebut}
function img_to_matrix_RGB(image: Image) -> numpy array
{fungsi yang menerima gambar dan mengembalikan matrix rgb}
function matrix_rgb_to_hist(image : numpy array) -> numpy array
{fungsi yang mengubah matrix rgb menjadi list histogram warna}
procedure quantify(input/output h,s,v : numpy array)
```

```

{prosedur yang mengubah nilai h,s,v dengan ketentuan quantify}
function create_submatrices(matrix :numpy_array) → array of numpy_array
{fungsi yang menghasilkan matrix bagian dalam array yang berisi 16 matrix}
function cosinesim(vektorA, VektorB : numpy_array) → float
{fungsi untuk menghitung cosine similarity dari kedua vektor}
function process_image(image_name : String, dataset_path : Image, vektor_reference :
numpy_array) → (image_name : String, similarity_score : float)
{fungsi Untuk menghitung setiap cosine similarity dalam proses multiprocessing}
function CBIR_Warna(img_path_1,img_path_2 : String) → float
{fungsi menerima 2 path image dan menghasilkan nilai similarity dari kedua gambar tersebut}

```

REALISASI

```

function img_to_matrix_RGB(image: Image) → numpy_array
{fungsi yang menerima gambar dan mengembalikan matrix rgb}

```

KAMUS LOCAL

img,img_rgb : Matrix

Algoritma

```

img ← cv2.imread(image)
img_rgb ← img.astype(np.float32)
img_rgb ← np.divide(img_rgb,255)
→ img_rgb

```

```

function matrix_rgb_to_hist(image :numpy_array) → numpy_array
{fungsi yang mengubah matrix rgb menjadi list histogram warna}

```

KAMUS LOCAL

r,g,b,cmax,xmin,delta,,h,s,mask_r,mask_g,mask_b : Matrix
hist : vektor

Algoritma

```

r ← image[:, :, 2]
g ← image[:, :, 1]
b ← image[:, :, 0]

cmax ← np.max(image, axis = 2)
cmin ← np.min(image, axis = 2)
delta ← np.subtract(cmax, cmin)

h ← np.zeros_like(r)
s ← np.zeros_like(r)

mask_r ← np.logical_and(delta ≠ 0, cmax = r)
mask_g ← np.logical_and(delta ≠ 0, cmax = g)
mask_b ← np.logical_and(delta ≠ 0, cmax = b)

h[delta = 0] ← 0
h[mask_r] ← 60*((g[mask_r] - b[mask_r])/(delta[mask_r]))%6
h[mask_g] ← 60*((b[mask_g] - r[mask_g])/(delta[mask_g]))+2

```

```

h[mask_b] ← 60*((r[mask_b] -g[mask_b])/(delta[mask_b]))+4)

s[cmax = 0] ← 0
s[cmax ≠ 0] ← (delta[cmax ≠ 0]) / (cmax[cmax ≠ 0])

quantify(h,s,cmax)

hist <- np.zeros(14)
hist[0] ← np.count_nonzero(h = 0)
hist[1] ← np.count_nonzero(h = 1)
hist[2] ← np.count_nonzero(h = 2)
hist[3] ← np.count_nonzero(h = 3)
hist[4] ← np.count_nonzero(h = 4)
hist[5] ← np.count_nonzero(h = 5)
hist[6] ← np.count_nonzero(h = 6)
hist[7] ← np.count_nonzero(h = 7)

hist[8] ← np.count_nonzero(s = 0)
hist[9] ← np.count_nonzero(s = 1)
hist[10] <-- np.count_nonzero(s = 2)

hist[11] ← np.count_nonzero(cmax = 0)
hist[12] ← np.count_nonzero(cmax = 1)
hist[13] ← np.count_nonzero(cmax = 2)
→hist

```

function quantify(*input/output h,s,v :numpy array*)
{prosedur yang mengubah nilai h,s,v dengan ketentuan quantify}
KAMUS LOCAL

Algoritma

```

h[h > 315] ← 0
h[np.logical_and(h > 0, h <= 25)] ← 1
h[np.logical_and(h > 25, h <= 40)] ← 2
h[np.logical_and(h > 40, h <= 120)] ← 3
h[np.logical_and(h > 120, h <= 190)] ← 4
h[np.logical_and(h > 190, h <= 270)] ← 5
h[np.logical_and(h > 270, h <= 295)] ← 6
h[np.logical_and(h > 295, h <= 315)] ← 7

s[s >= 0.7] ← 2
s[np.logical_and(s >= 0.2, s < 0.7)] ← 1
s[s < 0.2] = 0

v[v >= 0.7] ← 2
v[np.logical_and(v >= 0.2, v < 0.7)] ← 1
v[v < 0.2] ← 0

function create_submatrices(matrix : numpy array) -> numpy array
{fungsi yang menghasilkan matrix bagian dalam array yang berisi 16 matrix}

```

KAMUS LOCAL

Batas_baris,batas_kolom : integer
Submatrices : list

Algoritma

```
batas_baris ← matrix.shape[0] // 4
batas_kolom ← matrix.shape[1] // 4

submatrices ← []

i traversal (4):
    j traversal (4):
        if ( i = 3 and j = 3) then
            submatrices.append(matrix[i*batas_baris:,j*batas_kolom:])
        else if(i= 3) then
            submatrices.append(matrix[i*batas_baris:,j*batas_kolom:(j+1)*
            batas_kolom])
        else:
            submatrices.append(matrix[i*batas_baris:(i+1)*batas_baris,j*
            batas_kolom:(j+1)*batas_kolom])
→submatrices

function cosinesim(vektorA, vektorB : numpy_array) → float
{fungsi untuk menghitung cosine similarity dari kedua vektor}
```

KAMUS LOCAL

dot_product,normA,normB : float

ALGORITMA

```
dot_product ← np.dot(vektorA, vektorB)
norm_A ← np.linalg.norm(vektorA)
norm_B ← np.linalg.norm(vektorB)
→ dot_product / (norm_A * norm_B)
```

```
function process_image(image_name : String, dataset_path : Image, vektor_reference :
numpy_array) → (image_name : String, similarity_score : float)
{fungsi Untuk menghitung setiap cosine similarity dalam proses multiprocessing}
```

KAMUS LOCAL

image_path : String
matrix, n_matrix, Vektor : numpy_array
similarity : list
similarity_score : float

ALGORITMA

```
image_path ← os.path.join(dataset_path, image_name)
matrix ← img_to_matrix_RGB(image_path)
n_matrix ← create_submatrices(matrix)
Vektor ← [matrix_rgb_to_hist(matrix) for matrix in n_matrix]

similarity ← [cosinesim(vektor_reference[i], vektor[i]) for i in
```

```

range(len(vektor_reference))
similarity_score ← sum(similarity) / len(similarity) if len(similarity) ≠ 0 else 0
→ (image_name, similarity_score)

function CBIR_Warna(img_path_1,img_path_2 : String) → float
{fungsi menerima 2 path image dan menghasilkan nilai similarity dari kedua gambar tersebut}

KAMUS LOCAL
Matrix1, Matrix2, m_matriks, n_matriks, vektor1, vektor2, : numpy_array
similarity : list
result : float

ALGORITMA

Matrix1 ← img_to_matrix_RGB(img_path_1)
Matrix2 ← img_to_matrix_RGB(img_path_2)

m_matriks ← create_submatrices(Matrix1)
n_matriks ← create_submatrices(Matrix2)

vektor1 ← [matrix_rgb_to_hist(matrix) for matrix in m_matriks]
vektor2 ← [matrix_rgb_to_hist(matrix) for matrix in n_matriks]

similarity ← [cosinesim(vektor1[i], vektor2[i]) for i in range(len(vektor1))]

result ← sum(similarity) / len(similarity) if len(similarity) != 0 else 0
→ result

```

4.1.2 Pseudocode CBIR Tekstur

```

function compress_image(image : Matrix, compression_scale : float, quality : float)
-> Matrix
{Fungsi ini untuk Mengkompresi gambar dengan skala tertentu }

Algoritma
compressed_image <- cv2.resize(image,
(0,0),fx<-compression_scale,fy=compression_scale)_-, compressed_image_data <-
cv2.imencode('.jpg', compressed_image, [int(cv2.IMWRITE_JPEG_QUALITY), quality])
-> compressed_image_data

function cbirTexture(image : matrix)
{Fungsi ini untuk memproses image menjadi vector }

Algoritma
# Kompresi gambar sebelum diproses
compressed_image_data <- compress_image(image, compression_scale=0.7, quality=80)

# Mendekompressi gambar

```

```

decompressed_image <- cv2.imdecode(compressed_image_data, cv2.IMREAD_COLOR)
greyImage <- cv2.cvtColor(decompressed_image, cv2.COLOR_BGR2GRAY)
#buat matrix framework
coOccurrence <- np.zeros((256, 256))

#isi matrix coOccurrence
i traversal(0..greyImage.shape[0]):
    j traversal(0..greyImage.shape[1] - 1):
        coOccurrence[greyImage[i, j], greyImage[i, j + 1]] <- coOccurrence[greyImage[i, j], greyImage[i, j + 1]]+ 1

transposeCo <- np.transpose(coOccurrence)
symMatrix <- np.add(coOccurrence, transposeCo)

#cari matrix yang telah di normalisasi
if np.sum(symMatrix) > 0 then
    glcmNorm <- symMatrix / np.sum(symMatrix)

#hitung contrastnya
contrast <- calculateContrast(glcmNorm)

#hitung Homogeneity
homogeneity <- calculateHomogeneity(glcmNorm)

#hitung Entropy
entropy <- CalculateEntropy(glcmNorm)

#hitung dissimilarity
dissimilarity <- calculateDissimilarity(glcmNorm)

#hitung asm
asm <- calculateASM(glcmNorm)

#hitung energy
energy <- calculateEnergy(asm)
vector <- np.array([contrast, homogeneity, entropy, dissimilarity, asm, energy])

-> vector

function calculateHomogeneity(glcmNorm : Matrix) -> float
{Fungsi ini digunakan untuk menghitung homogeneity}

Algoritma
i, j <- np.indices(glcmNorm.shape)

# Hitung homogeneity
homogeneity <- np.sum(glcmNorm / (1. + (i - j) ** 2))

-> homogeneity

function CalculateEntropy(glcmNorm: Matrix) -> float
{fungsi ini digunakan untuk menghitung entropy}

```

```

Algoritma
positive_glcNorm <- glcmNorm[glcmNorm > 0]
# Hitung entropy
entropy <- -np.sum(positive_glcNorm * np.log2(positive_glcNorm))
-> entropy

Function calculateContrast(glcNorm) -> float
{fungsi ini digunakan untuk menghitung contrast}

Algoritma
i, j <- np.indices(glcNorm.shape)
# Hitung contrast
contrast <- np.sum(((i - j) ** 2) * glcNorm)
-> contrast

function cosineSim_texture(vector1, vector2)
{fungsi ini digunakan untuk menghitung cosine similarity dari 2 vektor}

Algoritma
dotProduct -<- np.dot(vector1, vector2)
A <- np.sqrt(np.sum(vector1 ** 2))
B <- np.sqrt(np.sum(vector2 ** 2))
-> dotProduct / (A * B)

function process_image_texture(args)
{fungsi ini digunakan untuk memproses CBIR texture}

Algoritma
image_name, dataset_path, reference_vector <- args
image_path <- os.path.join(dataset_path, image_name)
image <- cv2.imread(image_path)
image_vector <- cbirTexture(image)
similarity_score <- cosineSim_texture(reference_vector, image_vector)
-> (image_name, similarity_score)

function calculateDissimilarity(glcNorm) -> float
{fungsi ini digunakan untuk menghitung dissimilarity}

Algoritma
i, j <- np.indices(glcNorm.shape)
# Hitung dissimilarity
dissimilarity <- np.sum(np.abs(i - j) * glcNorm)
-> dissimilarity

function calculateASM(glcNorm) -> float
{fungsi ini digunakan untuk menghitung ASM}

Algoritma
# Hitung ASM
asm <- np.sum(glcNorm ** 2)
-> asm

```

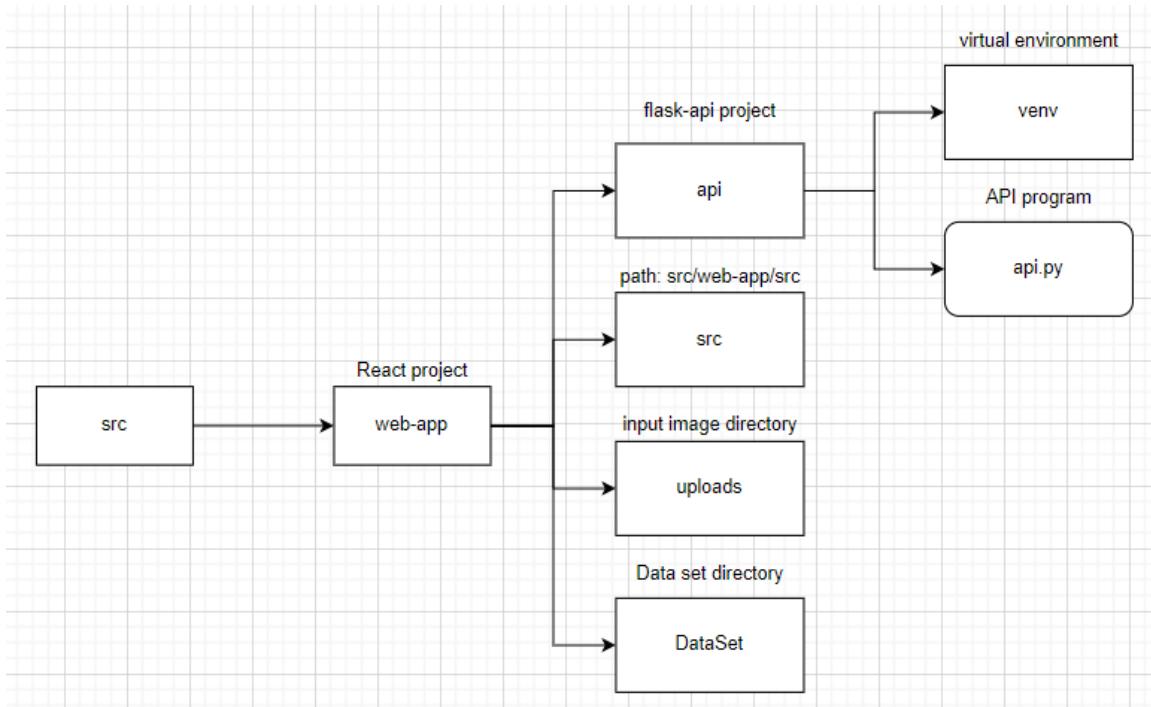
```
function calculateEnergy(asn)-> float
{fungsi ini digunakan untuk menghitung energy}
```

Algoritma

```
# Hitung energy
energy <- np.sqrt(asn)
-> energy
```

4.2 Penjelasan Struktur Program Berdasarkan Spesifikasi

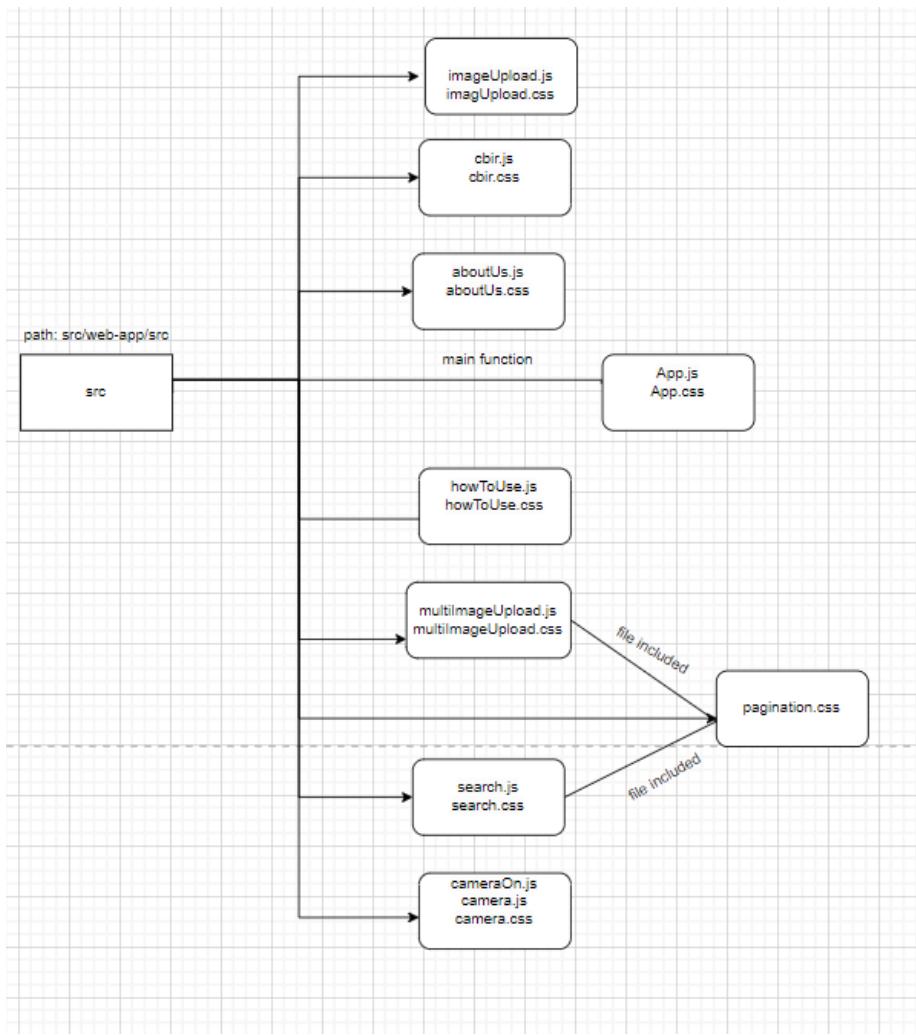
4.2.1 Struktur Program Secara Keseluruhan



Program dimulai pada folder “src”, yakni folder untuk menyimpan kode program pada Tugas Besar ini. Selanjutnya, terdapat folder “web-app” yang merupakan react.js project untuk membuat website. Lalu, di dalam folder “web-app”, terdapat beberapa folder dengan fungsionalitasnya masing-masing. Folder “uploads” dan “DataSet” berfungsi untuk menyimpan gambar yang akan digunakan dalam search engine. Folder “uploads” berisi input gambar yang akan dicari, sedangkan folder “DataSet” berisi data set gambar yang akan dibandingkan.

Selain itu, terdapat folder “api” yang merupakan folder backend berbasis python (flask) yang berfungsi untuk menjalankan operasi atau tugas sebagai api, seperti menyimpan gambar, mengirim gambar, dan mengolah gambar. Terakhir, terdapat folder “src” yang berisi file-file kebutuhan frontend yang akan dibahas lebih lanjut pada bab 4.2.2,

4.2.2 Struktur Website

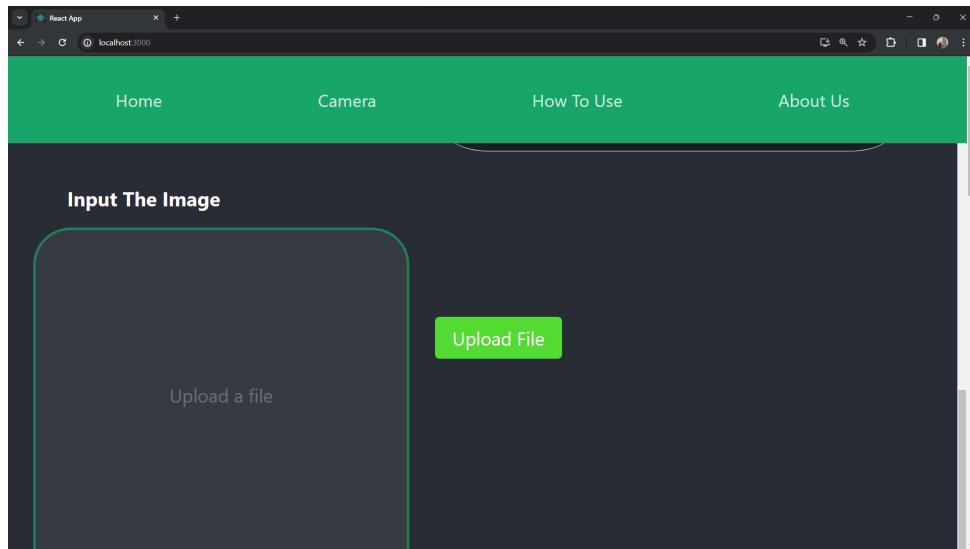


Struktur website dalam folder “src” berisi program javaScript dan css untuk mempercantik tampilannya. Program utama pada folder “src” adalah App.js dan App.css. Sisanya adalah file-file komponen dan css untuk styling nya. Komponen-komponen tersebut antara lain, “howToUse.js” dan “howToUse.css” untuk how to use webpage, “aboutUs.js” dan “aboutUs.css” untuk About Us webpage, “cbir.js” dan “cbir.css” untuk informasi mengenai search engine, “imageUpload.js” dan “imageUpload.css” untuk upload input image, “multilImageUpload.js” dan “multilImageUpload.css” untuk input data set, “search.js” dan “search.css” untuk search kemiripan gambar, “pagination.css” untuk styling pagination-nya, dan “cameraOn.js”, “camera.js”, dan “camera.css” untuk bonus camera.

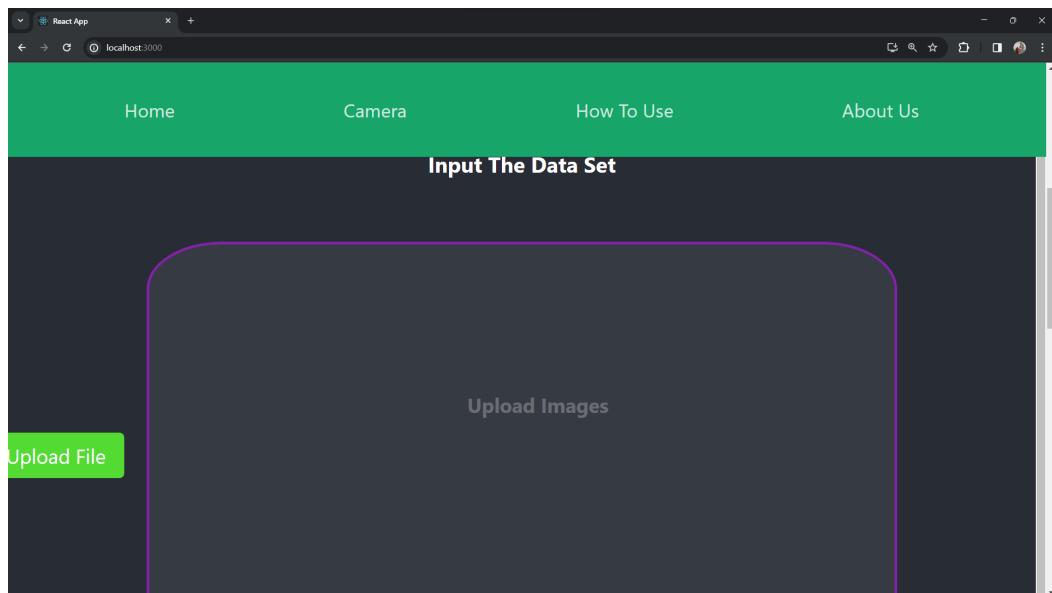
4.3 Penjelasan Tata Cara Penggunaan Program

4.3.1 Dengan menggunakan file sebagai input

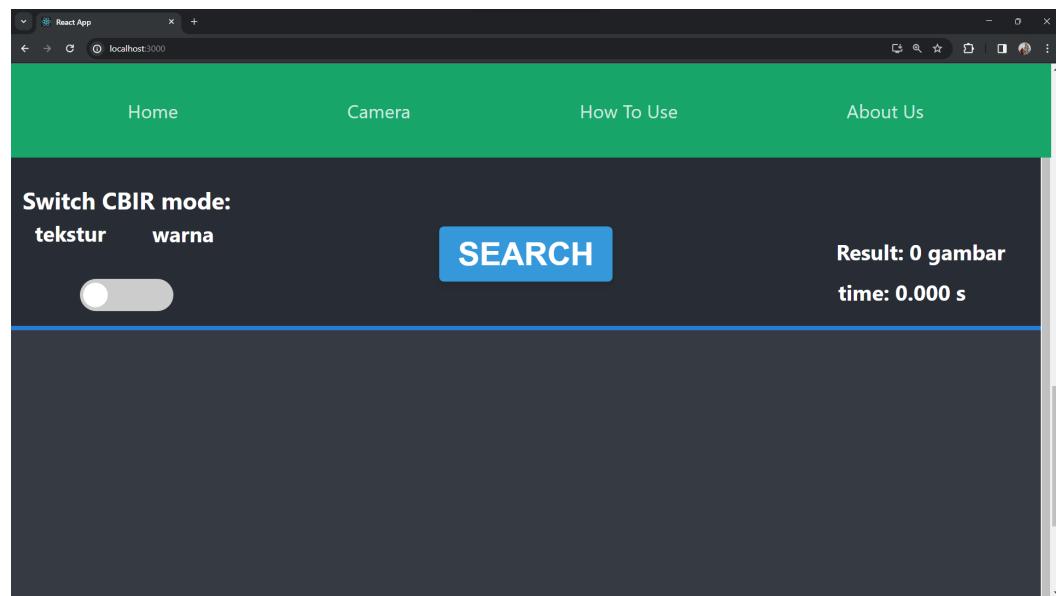
1. Untuk mulai menggunakan aplikasi ini anda bisa mengunggah gambar yang ingin anda cari pada box input image



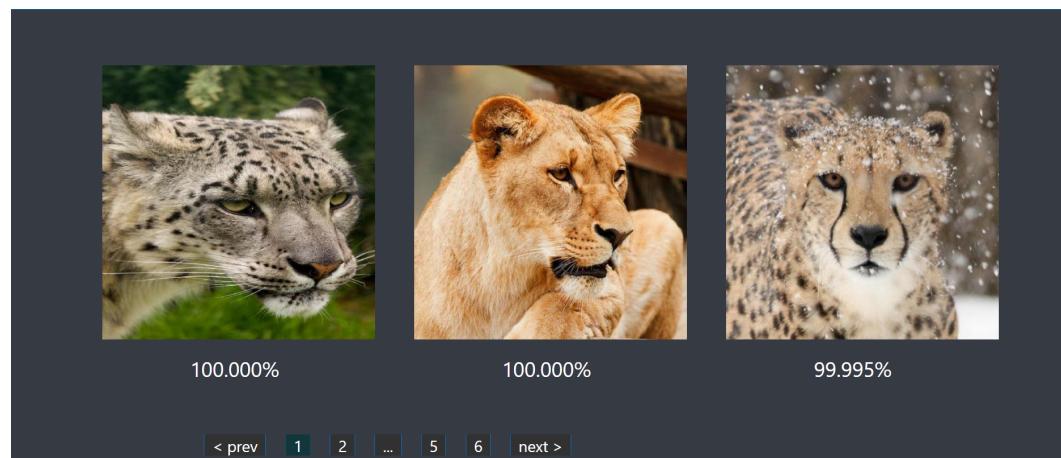
2. Setelah anda mengunggah gambar yang ingin anda cari, unggah juga dataset untuk mencari gambar mana pada dataset yang menyerupai dengan gambar yang tadi telah diinput



3. Jika anda sudah mengupload dataset anda bisa scroll kebawah lagi dan menemukan bagian yang seperti ini

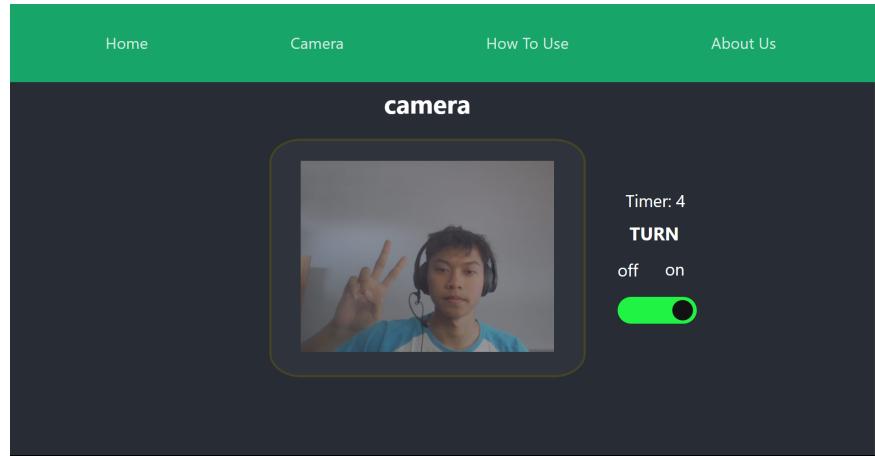


4. Disini anda bisa memilih untuk menggunakan metode CBIR dengan tekstur atau warna, setelah anda memilih anda bisa menekan tombol search untuk memulai pencarian, dan hasilnya akan tampil pada box yang dibawah, seperti ini

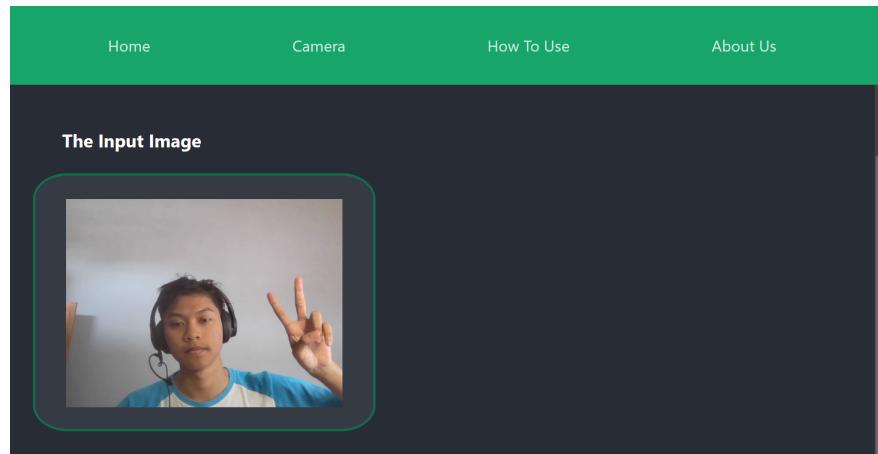


4.3.2 Dengan menggunakan tangkapan layar dari kamera sebagai input

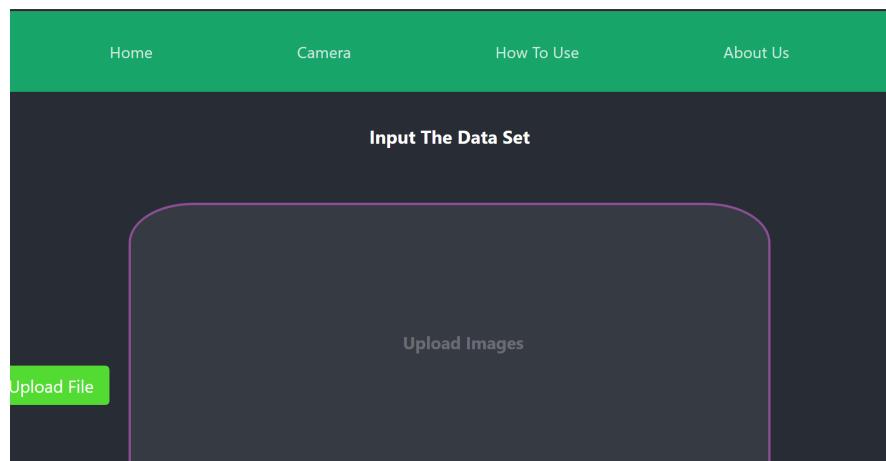
1. Pindah ke halaman camera dan aktifkan fitur kameranya



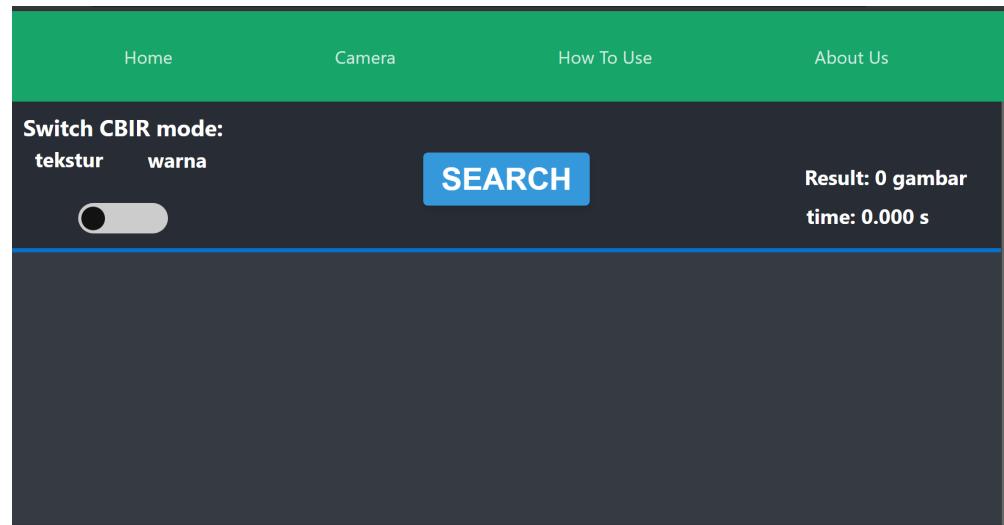
2. Selanjutnya periksa kembali apakah gambar yang anda ingin input sudah sesuai



3. Input data set



4. Lalu scroll lagi ke bawah dan pilih metode CBIR mana yang ingin anda pakai
5. Selanjutnya tekan search dan tunggu hasilnya



4.4 Hasil Pengujian

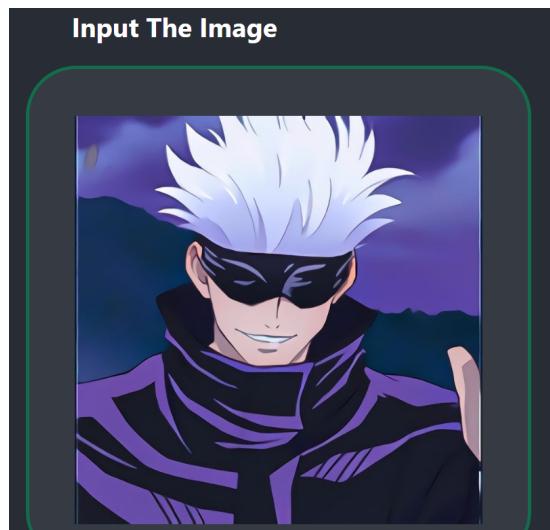
4.4.1 Pengujian dengan dataset hewan

1. Input gambar random

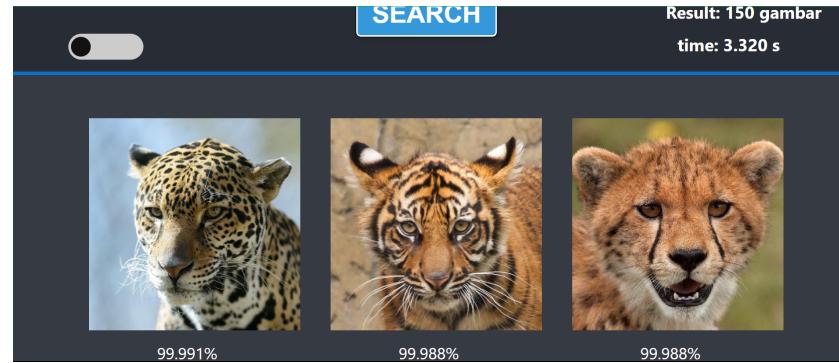
a. 200 Dataset

i. Tekstur

Input :

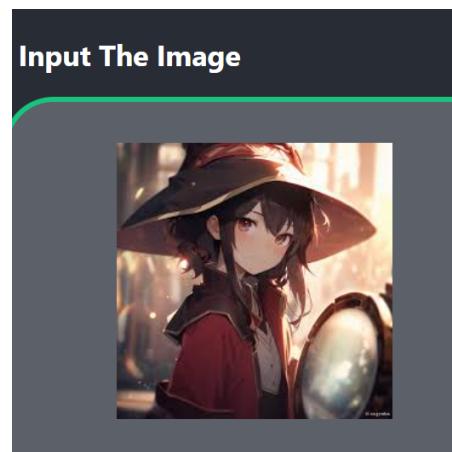


Hasil :

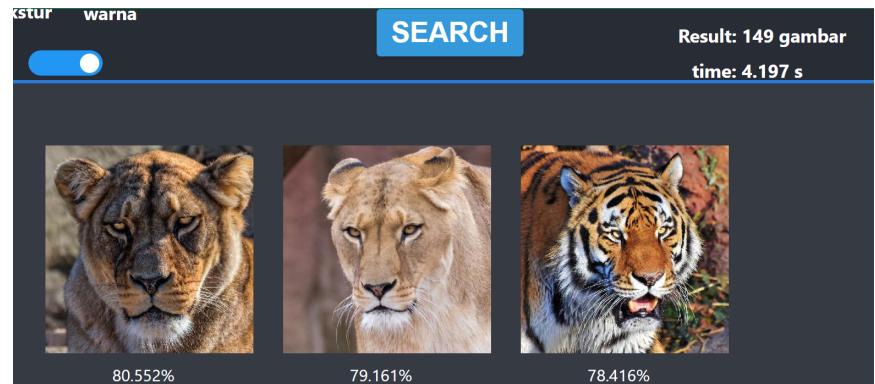


ii. Warna

Input:



Hasil:



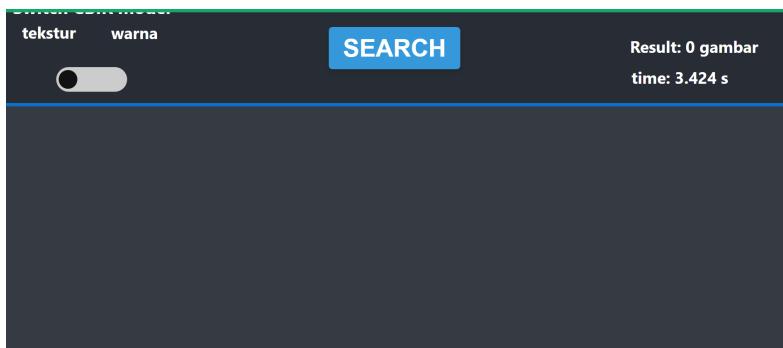
b. 150 Dataset

i. Tekstur

Input:

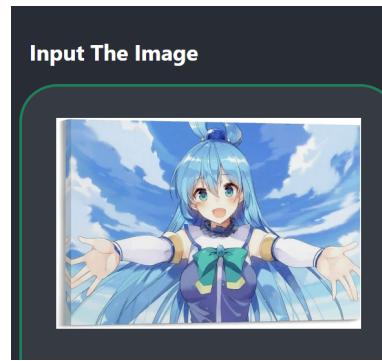


Hasil:

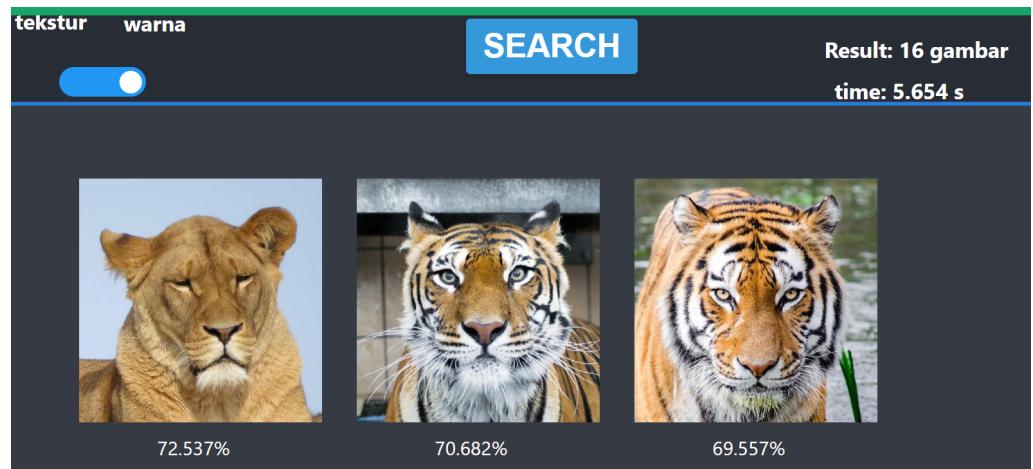


ii. Warna

Input:



Hasil:

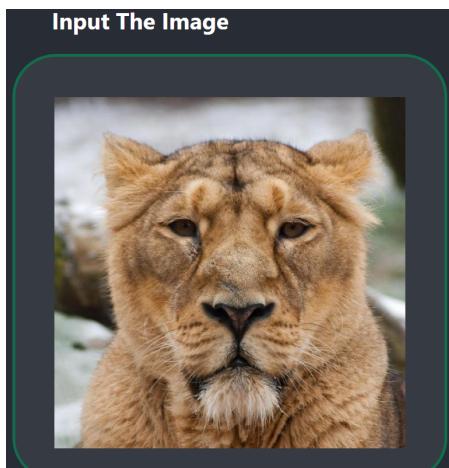


2. Input gambar hewan

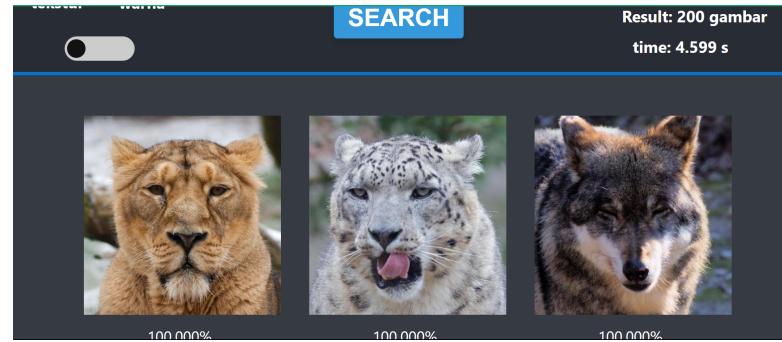
a. 200 Dataset

i. Tekstur

input :

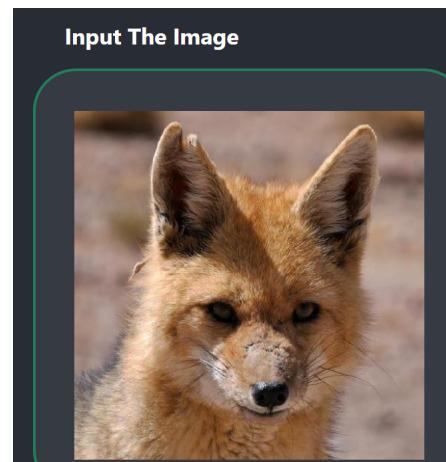


Hasil:

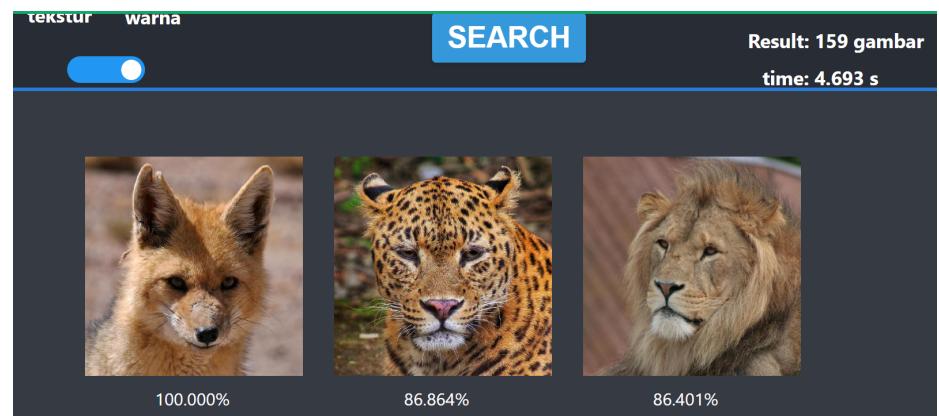


ii. Warna

Input:



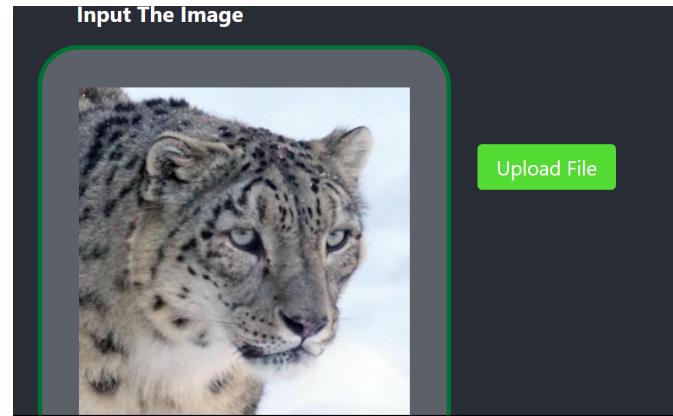
Hasil:



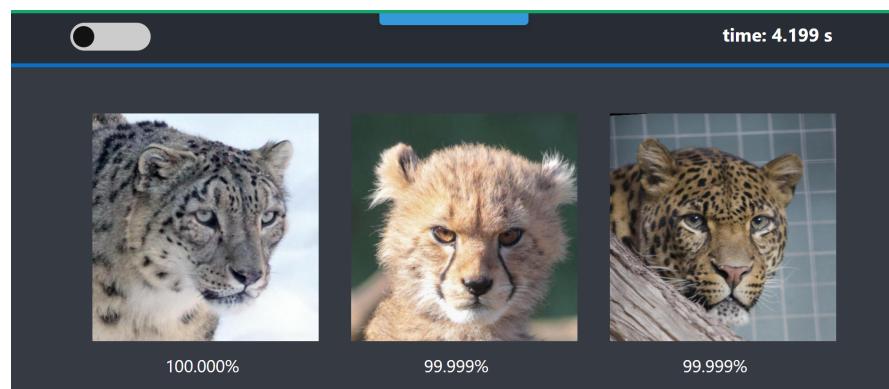
b. 150 Dataset

i. Tekstur

Input:

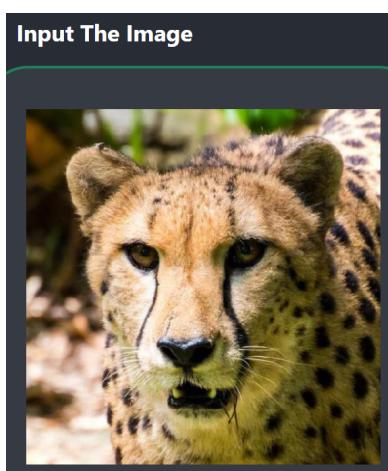


Hasil:

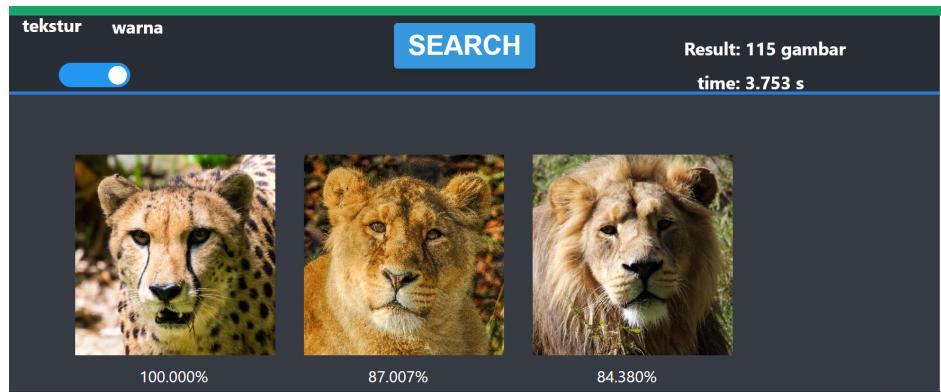


ii. Warna

Input:



Hasil:

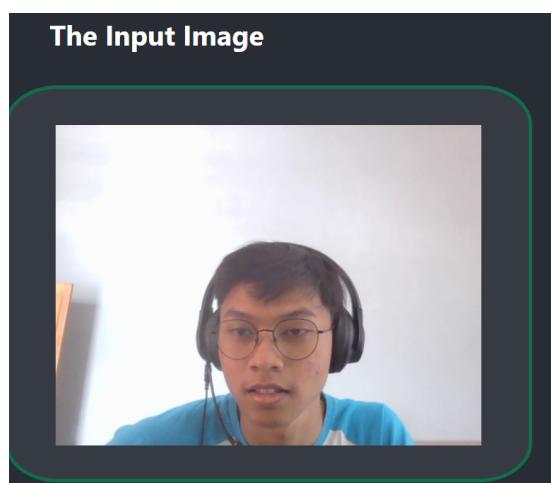


3. Input kamera

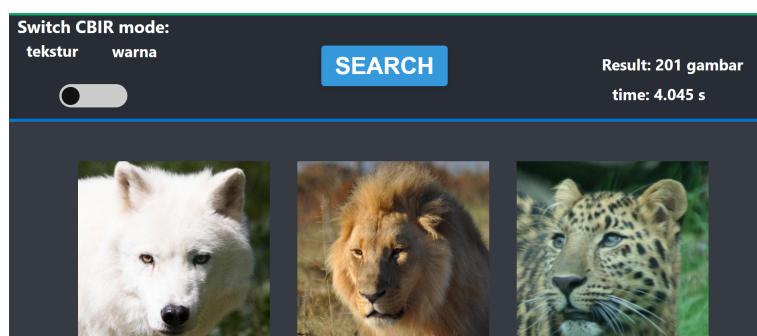
a. 201 Dataset

i. Tekstur

Input :

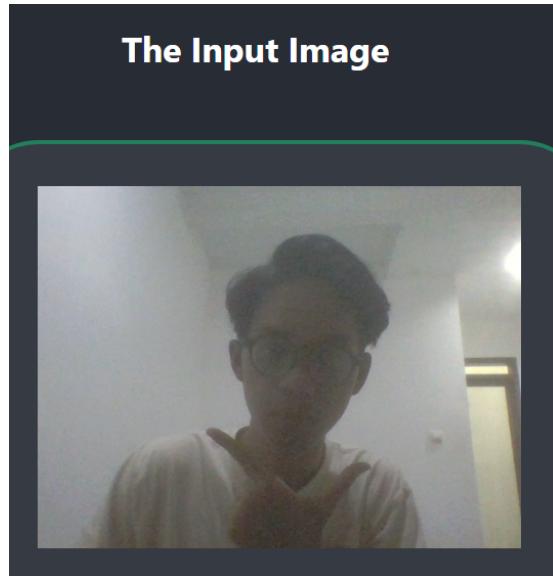


Hasil :

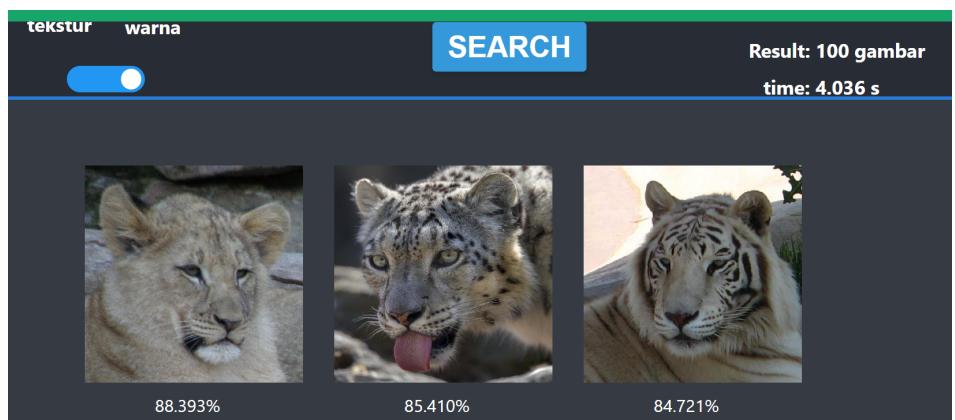


ii. Warna

Input:



Hasil:



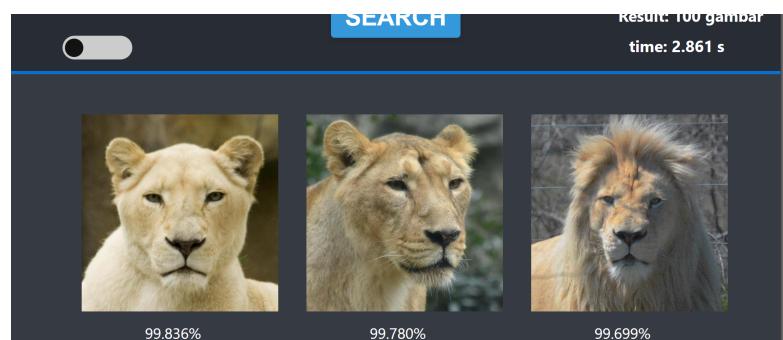
b. 100 Dataset

i. Tekstur

Input :

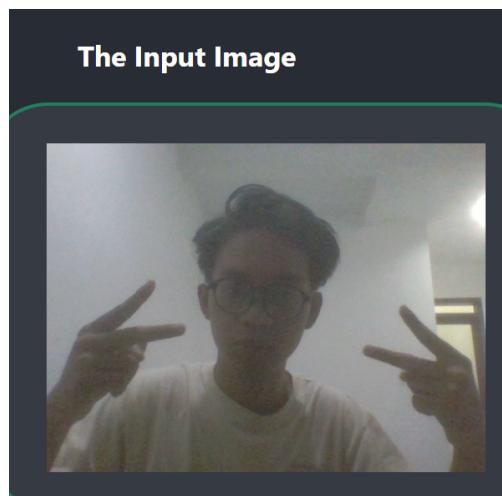


Hasil:

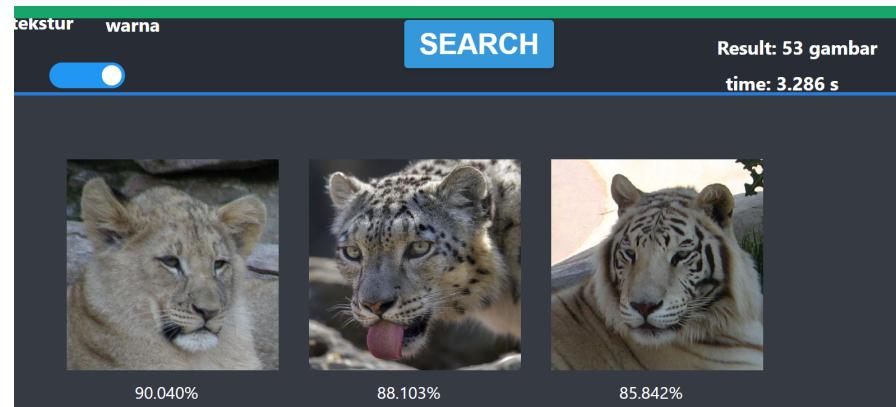


ii. Warna

Input:



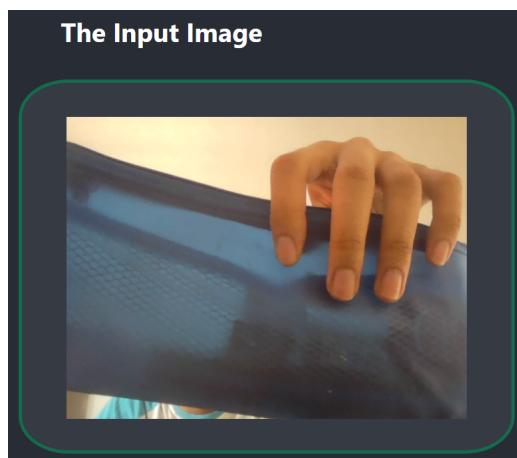
Hasil:



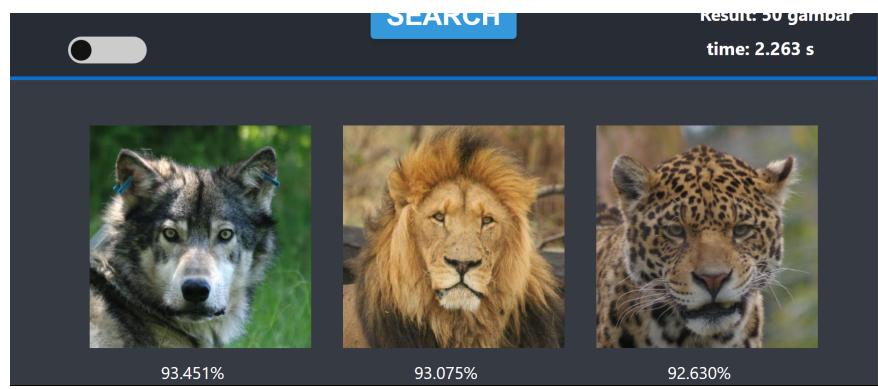
c. 50 Dataset

i. Tekstur

Input :

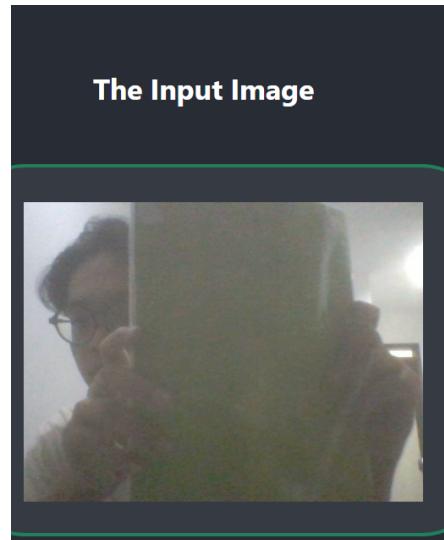


Hasil:

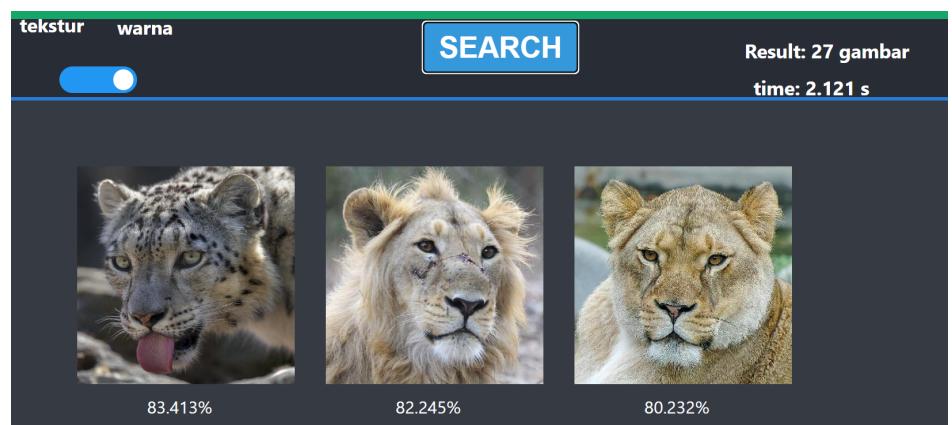


ii. Warna

Input:



Hasil:



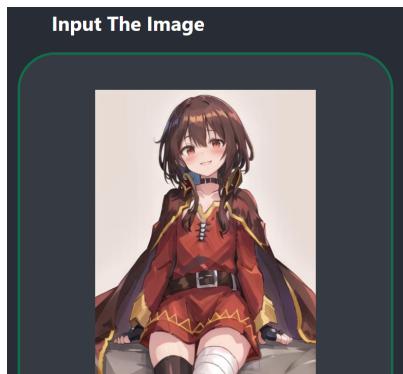
4.4.2 Pengujian dengan dataset kendaraan

1. Input gambar random

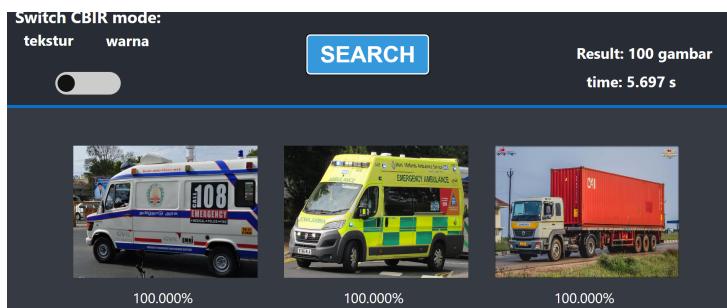
a. 100 Dataset

i. Tekstur

Input :

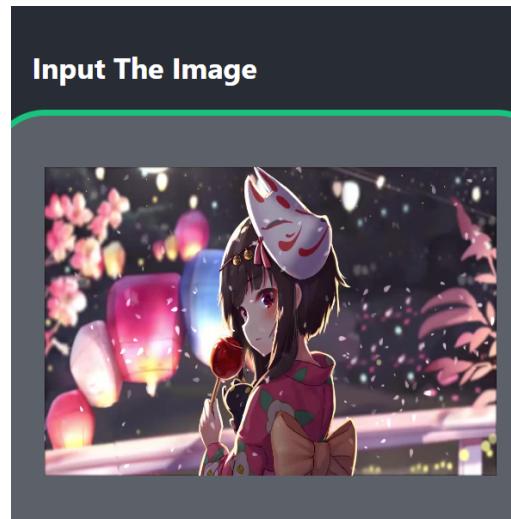


Hasil :

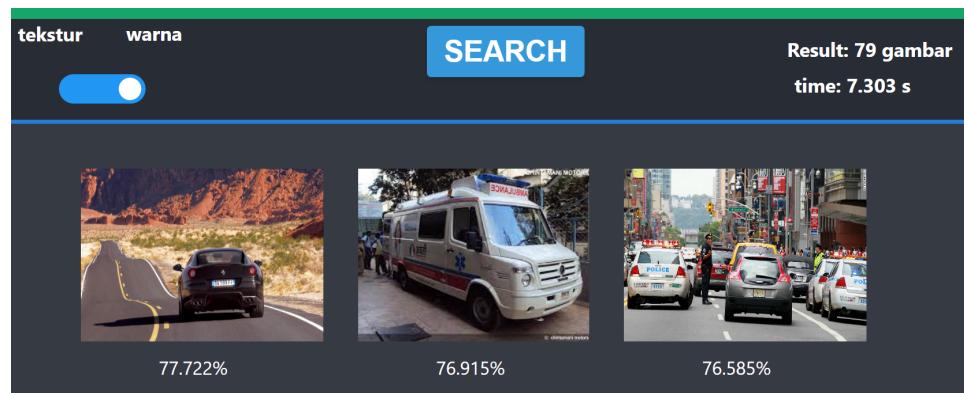


ii. Warna

Input:



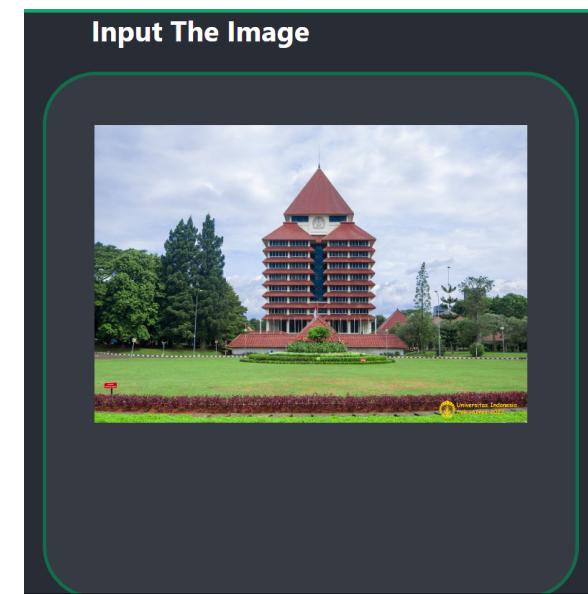
Hasil:



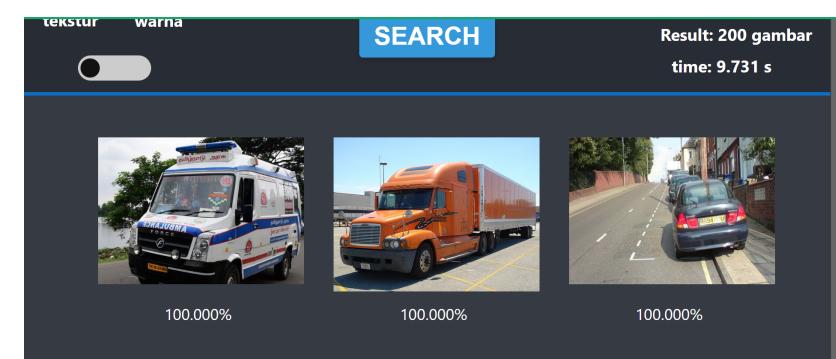
b. 200 Dataset

i. Tekstur

Input :

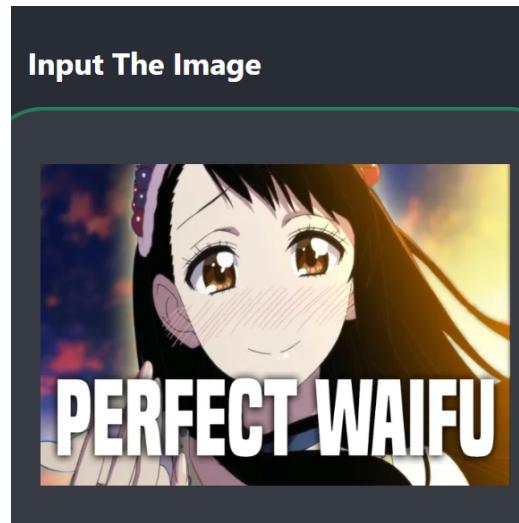


Hasil :

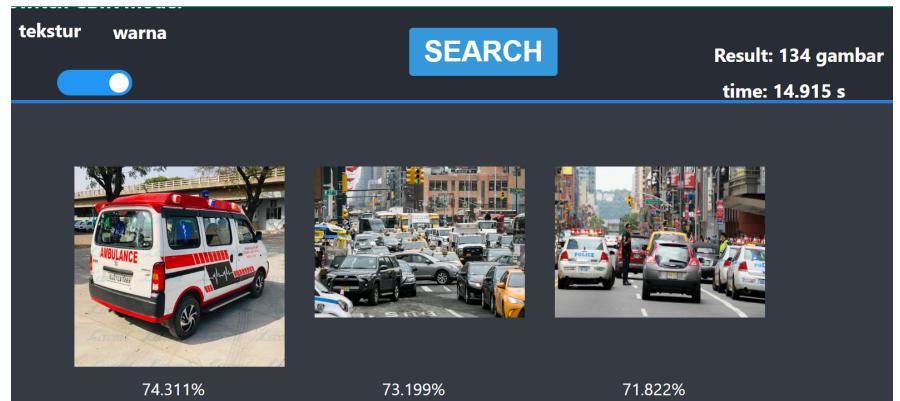


ii. Warna

Input:



Hasil

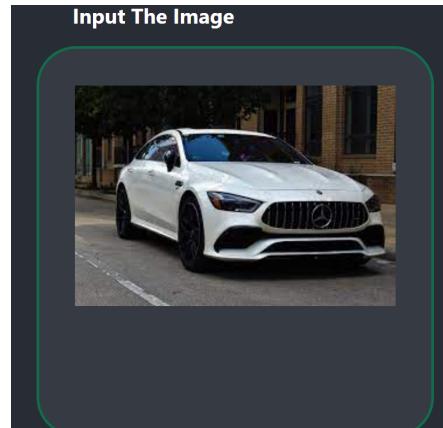


2. Input gambar kendaraan

a. 200 Dataset

i. Tekstur

Input :



Hasil :

Switch CBIR mode:
tekstur warna

SEARCH

Result: 200 gambar
time: 9.826 s



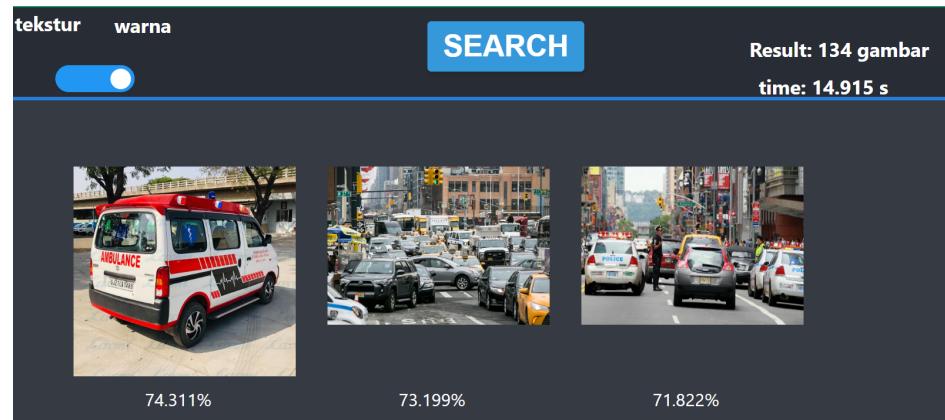
100.000% 100.000% 100.000%

ii. Warna

Input:



Hasil:



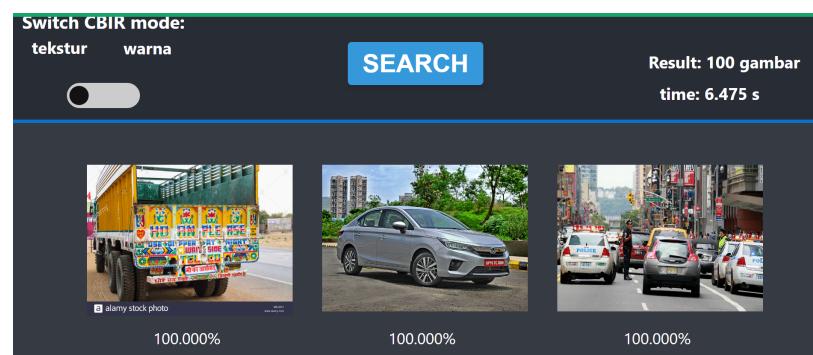
b. 100 Dataset

i. Tekstur

Input :

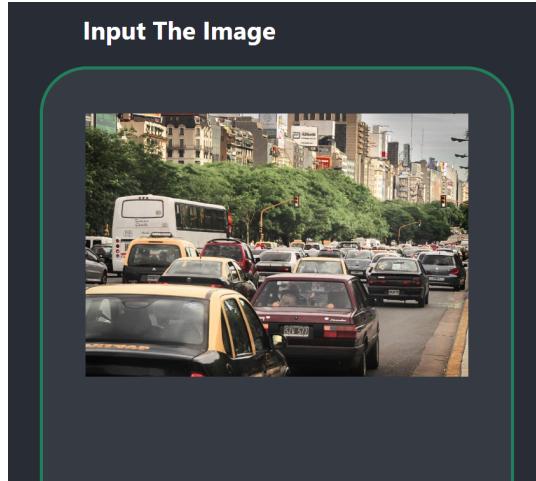


Hasil :

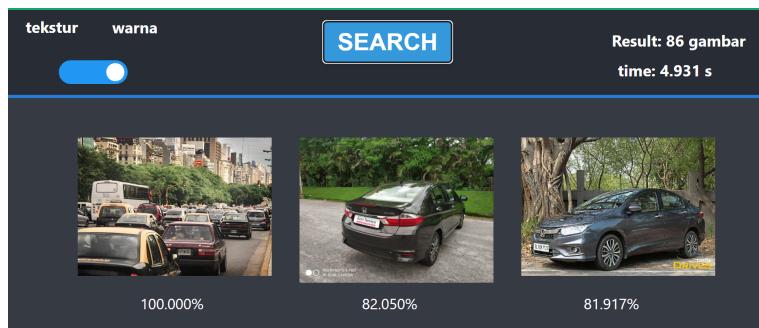


ii. Warna

Input :



Hasil :



4.3 Analisis

Content-Based Image Retrieval (CBIR) adalah teknik penelusuran gambar yang menggunakan fitur-fitur visual dari gambar untuk mencari gambar yang relevan. Ada banyak metode CBIR yang berbeda, tetapi dua metode yang paling umum digunakan adalah CBIR warna dan CBIR tekstur.

CBIR warna menggunakan informasi warna dari gambar untuk mencari gambar yang serupa. Metode ini bekerja dengan cara menghitung nilai warna dari setiap pixel dalam gambar. Nilai warna ini kemudian digunakan untuk membandingkan gambar yang dicari dengan gambar-gambar dalam database.

CBIR tekstur menggunakan informasi tekstur dari gambar untuk mencari gambar yang serupa. Metode ini bekerja dengan cara menghitung pola atau pola dalam gambar. Pola atau pola

ini kemudian digunakan untuk membandingkan gambar yang dicari dengan gambar-gambar dalam database.

4.3.3 Keunggulan CBIR Warna

CBIR warna lebih baik dari CBIR tekstur dalam kasus

- Saat gambar yang dicari memiliki warna yang sederhana. Misalnya, gambar yang hanya terdiri dari satu warna atau dua warna yang kontras.
- Saat gambar yang dicari memiliki warna yang khas. Misalnya, gambar yang menampilkan objek dengan warna yang unik, seperti bunga mawar merah atau langit biru.
- Saat gambar yang dicari memiliki warna yang dominan. Misalnya, gambar yang menampilkan objek dengan latar belakang yang berwarna solid.

Hal ini terjadi karena CBIR warna hanya menggunakan informasi warna untuk mencari gambar. Jika gambar yang dicari memiliki warna yang sederhana, khas, atau dominan, maka CBIR warna akan lebih mudah untuk membedakannya dengan gambar-gambar lain dalam database.

4.3.4 Keunggulan CBIR Tekstur

CBIR tekstur lebih baik dari warna pada kasus:

- Saat gambar yang dicari memiliki tekstur yang kompleks. Misalnya, gambar yang menampilkan objek dengan pola yang rumit, seperti kulit ular atau daun pohon.
- Saat gambar yang dicari memiliki tekstur yang khas. Misalnya, gambar yang menampilkan objek dengan tekstur yang unik, seperti kulit harimau atau pasir pantai.
- Saat gambar yang dicari memiliki tekstur yang dominan. Misalnya, gambar yang menampilkan objek dengan latar belakang yang memiliki tekstur yang kuat.

Hal ini terjadi karena CBIR tekstur menggunakan informasi tekstur untuk mencari gambar. Jika gambar yang dicari memiliki tekstur yang kompleks, khas, atau dominan, maka CBIR tekstur akan lebih mudah untuk membedakannya dengan gambar-gambar lain dalam database.

Karakteristik	CBIR Warna	CBIR Tekstur
Informasi yang digunakan	Warna	Tekstur
Akurasi	Kurang akurat untuk gambar yang memiliki warna kompleks	Lebih akurat untuk gambar yang memiliki warna kompleks

Kemampuan untuk mencari gambar berdasarkan warna	ya	tidak
Kemampuan untuk mencari gambar berdasarkan tekstur	tidak	ya

4.4.4 Pemilihan Metode CBIR

Pemilihan metode CBIR yang tepat tergantung pada beberapa faktor, termasuk:

- Kebutuhan pengguna. Jika pengguna hanya mencari gambar berdasarkan warna, maka CBIR warna akan lebih baik. Namun, jika pengguna perlu mencari gambar berdasarkan tekstur, maka CBIR tekstur akan lebih baik.
- Karakteristik gambar yang dicari. Jika gambar yang dicari memiliki warna yang sederhana atau khas, maka CBIR warna akan lebih baik. Namun, jika gambar yang dicari memiliki tekstur yang kompleks atau khas, maka CBIR tekstur akan lebih baik.
- Data set yang digunakan. Jika data set yang digunakan berisi gambar-gambar dengan warna yang beragam, maka CBIR warna akan lebih baik. Namun, jika data set yang digunakan berisi gambar-gambar dengan tekstur yang beragam, maka CBIR tekstur akan lebih baik.

Tidak ada metode CBIR yang sempurna yang cocok untuk semua situasi. Pemilihan metode CBIR yang tepat harus dilakukan secara hati-hati, dengan mempertimbangkan semua faktor yang disebutkan di atas.

BAB 5

KESIMPULAN, SARAN, KOMENTAR, DAN REFLEKSI

5.1 Kesimpulan

CBIR warna dan tekstur adalah dua metode CBIR yang memiliki kelebihan dan kekurangan masing-masing. Pemilihan metode CBIR yang tepat tergantung pada kebutuhan dan karakteristik gambar yang akan dicari.

Secara umum, CBIR warna lebih baik untuk mencari gambar yang memiliki warna yang sederhana atau khas. CBIR tekstur lebih baik untuk mencari gambar yang memiliki tekstur yang kompleks atau khas. Namun, pemilihan metode CBIR juga dapat dipengaruhi oleh dataset yang digunakan. Jika data set yang digunakan berisi gambar-gambar dengan warna yang beragam, maka CBIR warna akan lebih baik. Namun, jika data set yang digunakan berisi gambar-gambar dengan tekstur yang beragam, maka CBIR tekstur akan lebih baik.

Oleh karena itu, pemilihan metode CBIR yang tepat harus disesuaikan dengan kondisi data set dan input image.

5.2 Saran

Penting untuk membuat cache berupa csv dalam menghitung CBIR karena akan sangat berguna jika dataset yang digunakan adalah dataset yang sudah ada cache sebelumnya. Contohnya saja, pada CBIR warna, histogram dataset tidak perlu dihitung lagi karena sudah ada data cache mengenai histogram tersebut. Hal tersebut akan mempersingkat waktu eksekusi program.

Sebaiknya UI nya bisa lebih diperhatikan kembali agar responsif dan tampilannya lebih menarik, bisa juga dengan menggunakan bootstrap ataupun tailwind. Hal tersebut akan jauh lebih memudahkan dan mempercepat penggerjaan styling website.

5.3 Komentar

Kami menanggapi program yang kami buat dapat dimanfaatkan atau digunakan untuk hal-hal yang positif dan bermanfaat. Proses search image menggunakan CBIR dengan parameter warna maupun tekstur dapat membantu dalam pencarian gambar dalam sebuah dataset yang

banyak. Hal ini dapat dikembangkan lebih lanjut untuk mempermudah kehidupan sehari-hari dan membantu orang lain dalam menyelesaikan masalah di kehidupan nyata.

5.4 Refleksi

Tugas besar Aljabar Linear dan Geometri yang kedua ini berjalan sangat baik, tetapi terdapat hal-hal yang dapat diperbaiki, seperti menggali pengetahuan kepada bidang keilmuan website lebih lanjut, seperti library frontend maupun backend, mencari tahu pemrosesan numerik yang efisien, dan integrasi program CBIR dengan API yang terstruktur.

5.5 Ruang Perbaikan atau Pengembangan

Pengembangan dapat dimulai dari styling frontend website yang tidak hanya menggunakan pengetahuan CSS, tetapi juga memanfaatkan library seperti bootstrap atau tailwind agar website menjadi responsive terhadap resolusi layar pengguna. Selain itu, dalam hal pemrosesan searching engine, dapat dikembangkan pemanfaatan cache yang berisi informasi-informasi terkait pemrosesan gambar. Hal tersebut akan mempercepat pemrosesan gambar dan lebih efisien.

DAFTAR PUSTAKA

Referensi :

<http://telkomnika.uad.ac.id/index.php/TELKOMNIKA/article/download/4646/3026>

<https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-g1cm-10c45b6d46a1>

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

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

<https://www.kaggle.com/datasets/rohan300557/vehicle-detection>

<https://blog.miguelgrinberg.com/post/how-to-create-a-react--flask-project>

<https://www.npmjs.com/package/react-paginate>

<https://www.geeksforgeeks.org/flask-http-method/>

Link Release :

Link Video :

