

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



Disusun Oleh:

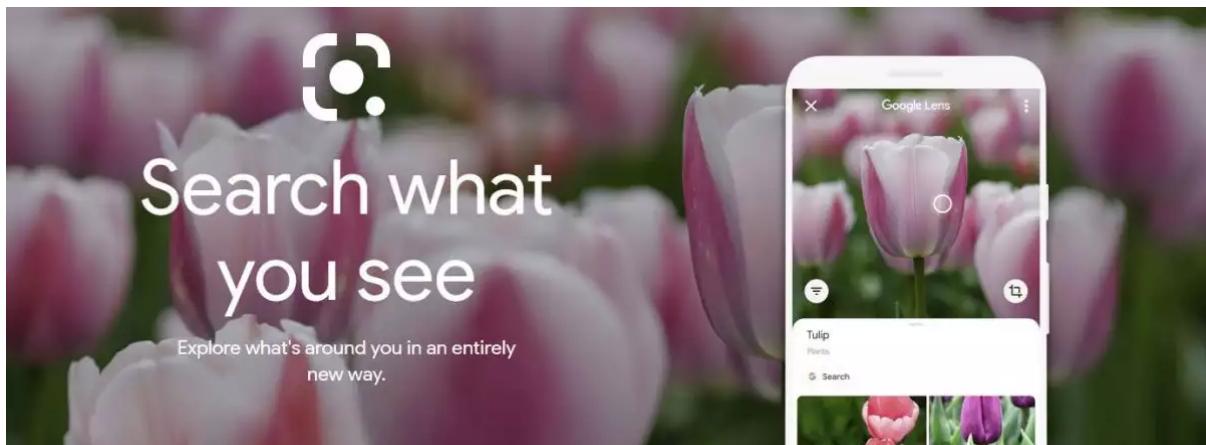
Ariel Herfrison	(13522002)
Bastian H. Suryapratama	(13522034)
Venantius Sean Ardi Nugroho	(13522078)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2023**

BAB 1

DESKRIPSI MASALAH

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



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

Di dalam Tugas Besar ini, kami 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 2

DASAR TEORI

A. Content-Based Information Retrieval

Content-Based Image Retrieval (CBIR) adalah proses pencarian dan pengambilan gambar berdasarkan kontennya. Konten suatu gambar yang dapat diekstrak terdiri dari warna, tekstur, maupun bentuk. Setelah fitur-fitur tersebut diekstrak, hasilnya direpresentasikan dalam vektor atau deskripsi numerik lainnya agar dapat dibandingkan dengan gambar lain. Setelah itu, suatu algoritma pencocokan diterapkan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan tersebut digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling mirip dengan gambar yang dicari. CBIR adalah salah satu cara yang lebih efisien dalam mengakses dan mengeksplorasi koleksi gambar. CBIR tidak menggunakan teks atau kata kunci sebagai basis pencarian, tetapi menggunakan kesamaan nilai citra visual antara gambar-gambar tersebut.

B. CBIR dengan parameter warna

CBIR dengan parameter warna dilakukan dengan mengekstrak fitur-fitur warna yang terdiri dari merah, hijau, dan biru (RGB). Fitur warna dalam format RGB kemudian dikonversi menjadi HSV.

Dalam melakukan konversi dari RGB ke HSV, nilai-nilai RGB perlu dinormalisasi untuk mengubah rangenya dari [0, 255] menjadi [0, 1] dengan persamaan berikut:

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

Setelah didapat nilai RGB yang sudah dinormalisasi, nilai Cmax, Cmin, dan Δ dapat dicari dengan persamaan berikut:

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

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

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

Hasil perhitungan Cmax, Cmin, dan Δ digunakan untuk menghitung nilai HSV dengan persamaan berikut:

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

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

$$V = C_{max}$$

Setelah didapat nilai HSV, kita dapat membandingkan dua buah gambar berdasarkan fitur warna dalam format HSV. Nilai perbandingan antara dua buah gambar dihitung dengan menggunakan *cosine similarity*. Persamaan *cosine similarity* adalah sebagai berikut:

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

A dan B adalah vektor dengan elemen berupa nilai *Hue*, *Saturation*, dan *Value*. Nilai n merupakan dimensi dari vektor. Tingkat kemiripan antara dua buah gambar ditentukan dari seberapa besar hasil dari *cosine similarity*. Semakin besar hasilnya, semakin mirip juga kedua gambar tersebut.

Untuk meningkatkan efisiensi, suatu gambar dibagi terlebih dahulu dalam blok 4 x 4. Nilai representatif dari sebuah blok dapat dicari dengan menghitung nilai rata-rata HSV dari blok terkait.

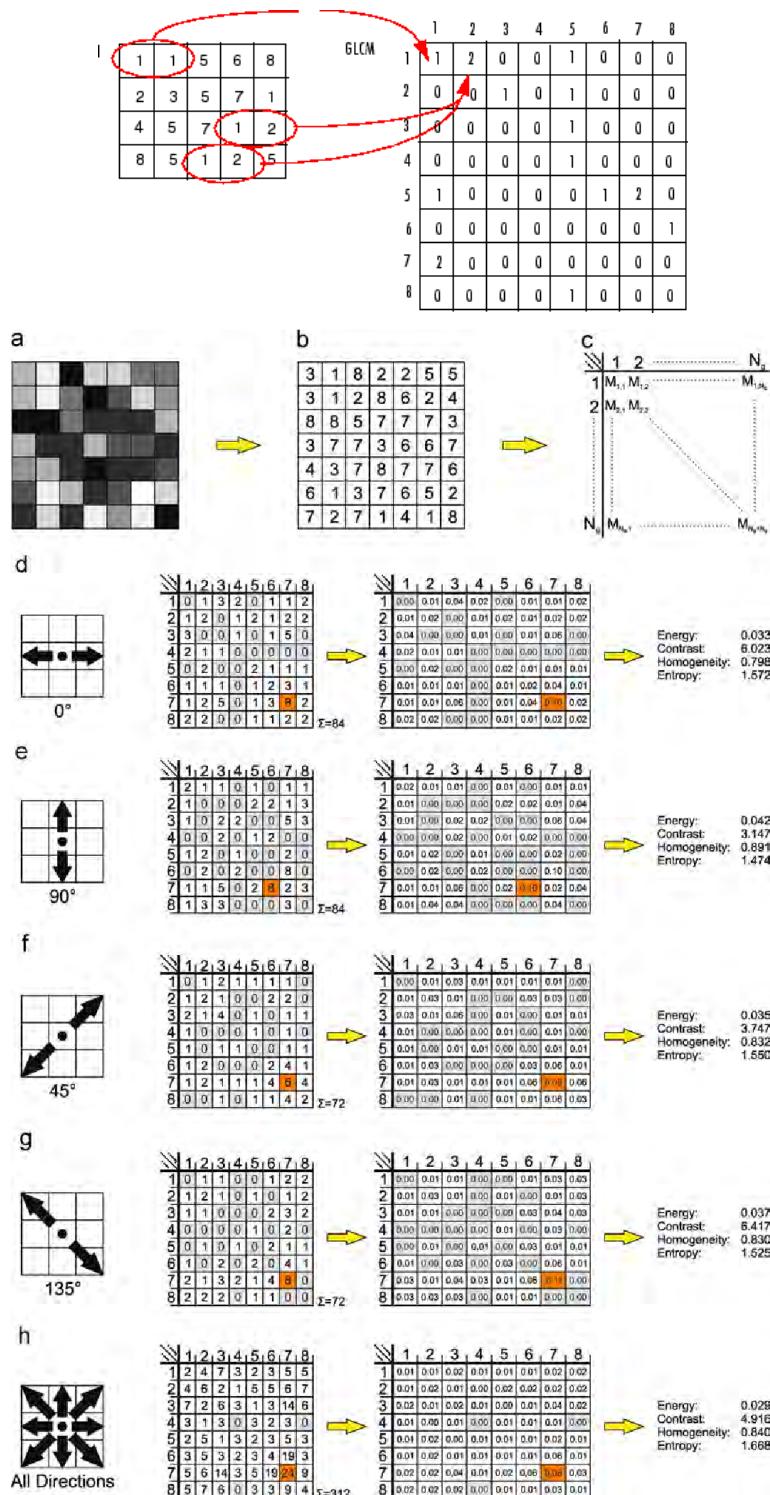
C. CBIR dengan parameter tekstur

CBIR dengan perbandingan tekstur dilakukan dengan ekstraksi komponen-komponen tekstur, seperti *contrast*, *entropy* dan *homogeneity*. Ekstraksi komponen tekstur dilakukan dengan bantuan *co-occurrence matrix*. Misalkan terdapat suatu gambar I dengan $n \times m$ pixel, maka dapat dibentuk *co-occurrence matrix* melalui persamaan berikut:

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

Co-occurrence matrix dibuat dengan meninjau intensitas dua pixel pada posisi yang berbeda. Nilai i adalah nilai intensitas pixel pertama, sedangkan nilai j adalah nilai intensitas pixel kedua. Nilai p serta q adalah posisi pixel pertama, sedangkan $p + \Delta x$ dan $q + \Delta y$ adalah posisi pixel kedua. *Offset* Δx dan Δy merupakan jarak antara dua pixel yang ditinjau dan θ adalah

arah/sudut peninjauan pixel. Dalam pembuatan *co-occurrence matrix*, dapat digunakan nilai θ sebanyak 0° , 45° , 90° , dan/atau 135° . Berikut adalah contoh cara pembuatan *co-occurrence matrix*.



Setelah didapat *co-occurrence matrix*, dibuat *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Lalu dicari *matrix normalization* dengan persamaan.

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

Lalu, komponen tekstur dapat diekstrak berdasarkan rumusnya masing-masing.

D. Website Development

Terdapat 3 komponen utama dalam pembangunan sebuah website, yaitu front-end, back-end, dan database. Back-end mengacu pada proses yang bersifat server-side, artinya tidak berinteraksi langsung dengan user. Penanganan data serta logika dari suatu website tercantum di sini. Bahasa yang bisa Anda gunakan untuk membangun back-end dari sebuah website tentu banyak sekali, contohnya Python, Ruby, Java dan masih banyak lagi.

Front-end adalah bagian dari website yang berinteraksi langsung dengan user. Contoh dari komponen - komponen front-end adalah UI (*User Interface*) dan juga UX (*User Experience*). Pembuatan front-end pada suatu website biasanya menggunakan HTML, CSS, dan JavaScript/TypeScript, tapi sering dibantu pembuatannya dengan framework - framework seperti React, Angular, dan Vue.js.

Database adalah koleksi data yang terstruktur, terorganisir , dan tersimpan sehingga mudah untuk diakses, dikelola, dan diubah. Dalam konteks *web development* database mengacu pada program yang digunakan untuk menyimpan dan mengambil data yang diperlukan oleh websitenya. Terdapat dua jenis database yaitu SQL (contohnya MySQL) dan NoSQL database (contohnya MongoDB), keduanya memiliki kelebihan dan kegunaannya masing - masing.

Antara front-end dan back-end tidak bisa semena - mena berinteraksi, untuk itu diperlukan API. Apa itu API ? API atau *Application Programming Interface* adalah himpunan peraturan - peraturan yang digunakan supaya aplikasi software bisa berinteraksi dengan aplikasi lainnya. dalam konteks web development, API digunakan supaya aplikasi front-end bisa berinteraksi dengan server back-end. Pada proyek ini kami menggunakan python sebagai back-end, oleh sebab itu, digunakan flask sebagai framework API-nya.

BAB 3

ANALISIS PEMECAHAN MASALAH

A. Langkah-langkah pemecahan masalah

1. Memperoleh parameter warna gambar
 1. Dari suatu gambar, dilakukan proses ekstraksi nilai-nilai RGB dari masing-masing pixel gambar tersebut. Dalam menghitung nilai perbandingan antara dua gambar, bukan nilai RGB yang dipakai, melainkan nilai HSV. Oleh karena itu, diperlukan konversi dari nilai RGB menjadi HSV.
 2. Langkah awal dalam konversi adalah menormalisasi nilai RGB agar rangenya berubah dari [0, 255] menjadi [0, 1]. Normalisasi dilakukan dengan persamaan berikut:

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

3. Dari nilai RGB yang dinormalisasi, dapat dicari nilai Cmax, Cmin, dan Δ dengan persamaan berikut:

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

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

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

4. Setelah diperoleh nilai Cmax, Cmin, dan Δ , nilai HSV dapat dihitung. Persamaan untuk mendapatkan nilai HSV adalah sebagai berikut:

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

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

$$V = C_{max}$$

5. Setelah nilai HSV diperoleh, perbandingan gambar dapat dilakukan dengan *cosine similarity*. Persamaan cosine similarity adalah sebagai berikut:

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

6. Untuk meningkatkan efisiensi, suatu gambar perlu dibagi menjadi blok 4×4 terlebih dahulu. Nilai representatif dari sebuah blok dicari dengan menghitung rata-rata nilai HSV dari semua pixel di dalam blok terkait.
7. Setelah rata-rata nilai HSV diperoleh dari masing-masing blok, kita dapat membandingkan gambar menggunakan *cosine similarity*. Setiap blok dalam suatu gambar akan dibandingkan kemiripannya dengan blok yang bersesuaian posisinya pada gambar yang lain. Karena suatu gambar dibagi menjadi 16 bagian, akan didapat 16 nilai *cosine similarity*.
8. Nilai akhir yang menyatakan kemiripan dua buah gambar dihitung dengan mencari rata-rata dari 16 nilai *cosine similarity* yang didapat sebelumnya. Setelah itu, nilai rata-rata yang diperoleh akan diubah dalam persentase. Persentase kemiripan dua buah gambar berada dalam rentang antara 0% sampai 100%.

2. Memperoleh parameter tekstur gambar:

1. *Extract* data warna-warna pixel gambar. Lalu, konversi warna gambar menjadi *grayscale*. Warna RGB dapat didapat nilai *grayscale*nya dengan rumus berikut:

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

2. Dibuat *co-occurrence matrix* berdasarkan intensitas *grayscale*. Karena rentang nilai *grayscale* 0-255, maka *co-occurrence matrix* yang dibuat akan berukuran 256×256 .
3. Pembuatan *co-occurrence matrix* menggunakan 1 variasi Δx , Δy , dan θ dengan $\Delta x = 1$, $\Delta y = 0$, dan $\theta = 0^\circ$. Walaupun ekstraksi komponen akan kurang akurat, hal ini meminimalisasi jumlah tahap yang perlu dilewati sehingga mempercepat prosedur ekstraksi.
4. Kemudian, *co-occurrence matrix* diproses untuk memperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*. Nilai 3 komponen tersebut didapatkan berdasarkan rumus berikut:

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 :

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

Dari ketiga komponen tersebut, dibuat vektor yang akan digunakan untuk mengukur tingkat kemiripannya dengan gambar lain.

5. Kemiripan dari dua gambar diukur dengan Teorema *Cosine Similarity*, yang memiliki rumus berikut:

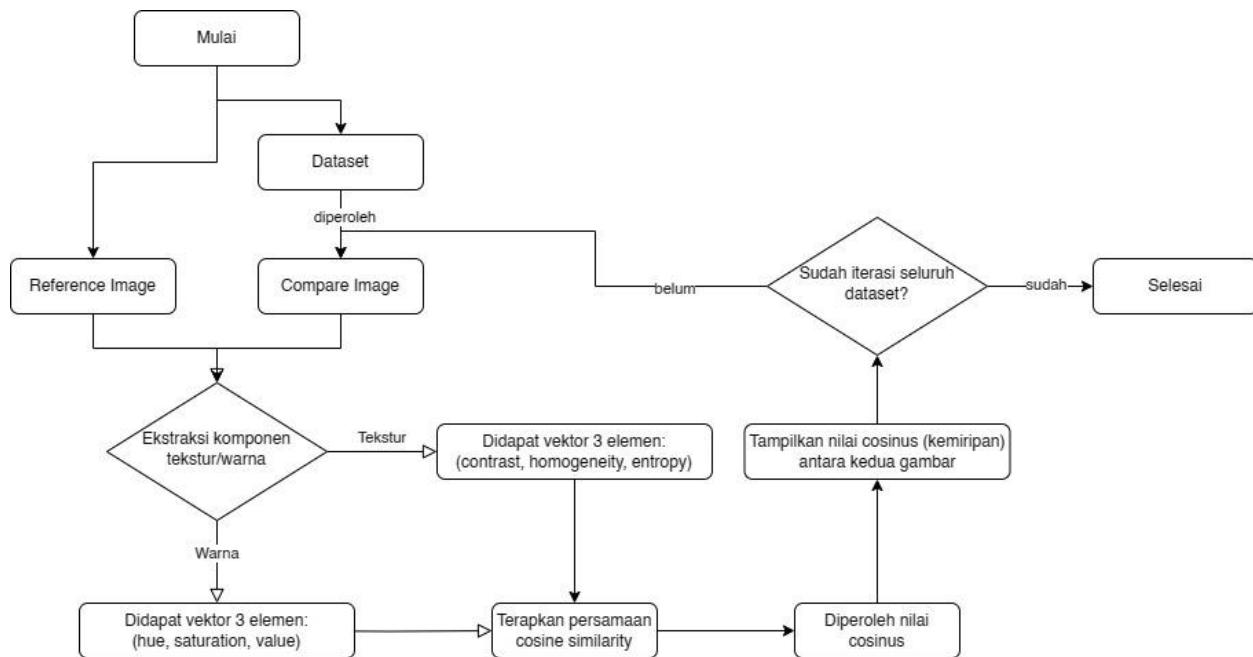
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Semakin besar hasil *Cosine Similarity* kedua vektor maka tingkat kemiripan kedua gambar semakin tinggi.

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

Sistem temu balik gambar dapat memanfaatkan Teorema Cosine Similarity, di mana apabila terdapat dua vektor, dapat diperoleh nilai cosinus sudut antara kedua vektor tersebut. Semakin dekat nilai cosinus tersebut dengan 1 – dalam kata lain, semakin kecil sudut antara kedua vektor tersebut – maka semakin besar tingkat kemiripan antara kedua vektor tersebut. Sebuah gambar dapat direpresentasikan dalam bentuk vektor dengan mengekstrak nilai komponen-komponennya. Dalam kasus ini, gambar direpresentasikan berdasarkan komponen tekstur atau warnanya. Setelah mendapat vektor berupa hasil ekstraksi komponen, vektor tersebut dapat dimasukkan ke dalam rumus Teorema Cosine Similarity untuk mendapat tingkat kemiripannya.

C. Ilustrasi kasus dan penyelesaiannya



BAB 4

IMPLEMENTASI DAN UJI COBA

A. Implementasi & Struktur Program

1. Algoritma CBIR dengan parameter warna

```
import base64
from io import BytesIO
from PIL import Image
import numpy as np
import math

function get_rgb_array_from_image(base64_image: String) -> numpy.ndarray
{menerima gambar dalam bentuk string base64}
{mengembalikan array 2D berisi RGB}
```

KAMUS LOKAL

```
Decoded_image: bytes
File_image: _io.BytesIO
image: PIL.JpegImagePlugin.JpegImageFile
resized_image: PIL.Image.Image
rgb_resized_image: PIL.Image.Image
```

ALGORITMA

```
{convert base64 to image}
decoded_image <- base64.b64decode(base64_image)
file_image <- BytesIO(decoded_image)
image <- Image.open(file_image)

{resize image (agar pembagian blocknya pas)}
resized_image <- image.resize((512, 512))

{mengambil hanya nilai RGB (karena format file mungkin bukan .jpg)}
rgb_resized_image <- resized_image.convert("RGB")

-> np.array(rgb_resized_image)
```

```
function get_c_max(normalized_rgb: numpy.ndarray) -> numpy.float64
{menerima array rgb yang sudah dinormalisasi}
{mengembalikan nilai Cmax}
```

KAMUS LOKAL

```
c_max: numpy.float64
```

ALGORITMA

```
c_max <- normalized_rgb[0]
if (normalized_rgb[1] > c_max) then
```

```
c_max <- normalized_rgb[1]

if (normalized_rgb[2] > c_max) then
    c_max <- normalized_rgb[2]

-> c_max
```

```
function get_c_min(normalized_rgb: numpy.ndarray) -> numpy.float64
{menerima array rgb yang sudah dinormalisasi}
{mengembalikan nilai Cmin}
```

KAMUS LOKAL

c_min: numpy.float64

ALGORITMA

```
c_min <- normalized_rgb[0]
if (normalized_rgb[1] < c_min) then
    c_min <- normalized_rgb[1]

if (normalized_rgb[2] < c_min) then
    c_min <- normalized_rgb[2]

-> c_min
```

```
function rgb_to_hsv(rgb: numpy.ndarray) -> numpy.ndarray
{menerima rgb, yaitu array dengan 3 elemen: red, green, dan blue}
{mengembalikan hsv, yaitu array dengan 3 elemen: hue, saturation, value}
```

KAMUS LOKAL

normalized_rgb: numpy.ndarray
c_max: numpy.float64
c_min: numpy.float64
delta: numpy.float64
hue: numpy.float64
saturation: numpy.float64
value: numpy.float64

ALGORITMA

```
{Normalize rgb}
normalized_rgb <- np.divide(rgb, 255)

{Calculate Cmax, Cmin, and delta}
c_max <- get_c_max(normalized_rgb)
c_min <- get_c_min(normalized_rgb)
delta <- c_max - c_min

{Menghitung nilai H (hue)}
depend_on (delta, c_max)
    delta = 0:
        hue <- 0
    c_max = normalized_rgb[0]:
```

```

hue <- 60 * (((normalized_rgb[1] - normalized_rgb[2]) / delta) mod 6)
c_max = normalized_rgb[1]:
    hue <- 60 * (((normalized_rgb[2] - normalized_rgb[0]) / delta) + 2)
else:
    hue <- 60 * (((normalized_rgb[0] - normalized_rgb[1]) / delta) + 4)

{Menghitung nilai S (saturation)}
if (c_max = 0) then
    saturation <- 0
else
    saturation <- (delta / c_max) * 100

{Menghitung nilai V (value)}
value <- c_max * 100

-> np.array([hue, saturation, value])

```

```

function create_blocks_array(hsv_array: numpy.ndarray) -> numpy.ndarray
{Membagi matriks 512x512 menjadi blok-blok berukuran 4x4,
lalu menghitung rata-rata warna dalam blok tersebut}

```

KAMUS LOKAL

```

blocks_array: numpy.ndarray
sum_hsv: numpy.ndarray
average_hsv: numpy.ndarray
i, j, k, l: integer

```

ALGORITMA

```

blocks_array <- np.empty((4, 4), dtype=np.ndarray)
i <- 0
while (i < hsv_array.shape[0]) do
    j <- 0
    while (j < hsv_array.shape[1]) do
        sum_hsv <- np.zeros(3)
        k traversal [i..(i + hsv_array.shape[0] div 4) - 1]
            l traversal [j..(j + hsv_array.shape[1] div 4) - 1]
                sum_hsv <- np.add(sum_hsv, hsv_array[k][l])

        average_hsv <- np.divide(sum_hsv, (hsv_array.shape[0] div 4) *
(hsv_array.shape[1] div 4))
        blocks_array[i div (hsv_array.shape[0] div 4)][j div
(hsv_array.shape[1] div 4)] <- average_hsv

        j <- j + hsv_array.shape[1] div 4

    i <- i + hsv_array.shape[0] div 4

-> blocks_array

```

```

function create_hsv_array_from_rgb_array(rgb_array: numpy.ndarray) ->
numpy.ndarray
{menerima array 2D berisi RGB}
{mengembalikan array 2D berisi HSV}

```

KAMUS LOKAL

```
hsv_array: numpy.ndarray
i, j: integer
```

ALGORITMA

```
i traversal [0..rgb_array.shape[0] - 1]
    j traversal [0..rgb_array.shape[1] - 1]
        hsv_array[i][j] <- rgb_to_hsv(rgb_array[i][j])
-> hsv_array
```

```
function cos_similarity(hsv1: numpy.ndarray, hsv2: numpy.ndarray) ->
numpy.float64
{menerima 2 buah HSV}
{mengembalikan nilai cosine similarity antara 2 buah HSV}
```

KAMUS LOKAL

```
dot_product: numpy.float64
norm_hsv1: real
norm_hsv2: real
```

ALGORITMA

```
dot_product <- hsv1[0] * hsv2[0] + hsv1[1] * hsv2[1] + hsv1[2] * hsv2[2]
norm_hsv1 <- math.sqrt(hsv1[0] ** 2 + hsv1[1] ** 2 + hsv1[2] ** 2)
norm_hsv2 <- math.sqrt(hsv2[0] ** 2 + hsv2[1] ** 2 + hsv2[2] ** 2)
-> dot_product / (norm_hsv1 * norm_hsv2)
```

```
function average_cos_similarity(blocks_hsv_array1: numpy.ndarray,
blocks_hsv_array2: numpy.ndarray) -> numpy.float64
{menerima 2 buah array 2D berisi rata-rata nilai HSV dari setiap blok gambar
{menjumlahkan nilai cosine similarity dari setiap elemen di dalam array 2D
tersebut,}
{kemudian mengembalikan nilai rata-rata cosine similarity}
```

KAMUS LOKAL

```
total_cos_similarity: numpy.float64
i, j: integer
```

ALGORITMA

```
total_cos_similarity <- 0
i traversal [0..blocks_hsv_array1.shape[0] - 1]
    j traversal [0..blocks_hsv_array1.shape[1] - 1]
        total_cos_similarity <- total_cos_similarity +
cos_similarity(blocks_hsv_array1[i][j], blocks_hsv_array2[i][j])
-> total_cos_similarity / (blocks_hsv_array1.shape[0] *
blocks_hsv_array1.shape[1])
```

2. Algoritma CBIR dengan parameter tekstur

```
import base64
from io import BytesIO
from PIL import Image
import numpy as np
import math

function greyscaled(elmt: tuple) -> integer
{menerima tuple 3 elemen berupa rgb values dan mengembalikan nilai
greyscalanya}

KAMUS LOKAL

r, g, b, value : integer

ALGORITMA

r <- elmt[0]
g <- elmt[1]
b <- elmt[2]
value <- 0.299*r+0.587*g+0.114*b

-> math.floor(value)

function cosSim(vec1, vec2: tuple) -> integer
{menerima dua vektor (tuple 3 elemen) dan mengembalikan nilai cosine similarity
antara keduanya}

KAMUS LOKAL

dot, len1, len2 : integer

ALGORITMA

dot <- (vec1[0]*vec2[0])+(vec1[1]*vec2[1])+(vec1[2]*vec2[2])
len1 <- math.sqrt((vec1[0]**2)+(vec1[1]**2)+(vec1[2]**2))
len2 <- math.sqrt((vec2[0]**2)+(vec2[1]**2)+(vec2[2]**2))

-> dot/(len1*len2)

function getTexture(loc) -> tuple
{menerima loc berupa kode base 64 dari suatu file gambar, lalu mengekstrak
komponen teksturnya dan mengembalikan tuple 3 elemen berisi contrast,
homogeneity, dan entropy}

KAMUS LOKAL

img_decoded : bytes
img_file : _io.BytesIO
img : Image
width, height, glcmSum, contrast, homogeneity, entropy, i, j, temp, p, d :
integer
Gray_arr : array of array of integer
coMatrix : array[0..256] of array[0..256] of integer
```

ALGORITMA

```
img_decoded <- base64.b64decode(loc)
img_file <- BytesIO(img_decoded)
img <- Image.open(img_file)

width <- img.size[0]
height <- img.size[1]

if (len(img.mode) < 3) then
    i traversal [0..height-1]
        j traversal [0..width-1]
            gray_arr[i][j] <- grayscaled(img.getpixel(j,i))
else
    i traversal [0..height-1]
        j traversal [0..width-1]
            gray_arr[i][j] <- img.getpixel(j,i)

i traversal [0..255]
    j traversal [0..255]
        coMatrix[i][j] <- 0

i traversal [0..height-1]
    j traversal [0..width-1]
        coMatrix[gray_arr[i][j]][gray_arr[i][j+1]] <-
        coMatrix[gray_arr[i][j]][gray_arr[i][j+1]] + 1

glcmSum <- 0
i traversal [0..255]
    j traversal [0..i]
        if (i==j) then
            coMatrix[i][j] <- coMatrix[i][j]*2
            glcmSum <- glcmSum + coMatrix[i][j]
        else
            temp <- coMatrix[i][j]
            coMatrix[i][j] <- coMatrix[i][j] + coMatrix[j][i]
            coMatrix[j][i] <- coMatrix[j][i] + temp
            glcmSum <- glcmSum + coMatrix[i][j]*2

contrast <- 0
homogeneity <- 0
entropy <- 0
i traversal [0..255]
    j traversal [0..i]
        p <- coMatrix[i][j]/glcmSum
        d <- i-j
        contrast <- contrast + p*(pow(d,2))
        homogeneity <- homogeneity + p/(1+(pow(d,2)))
        if (p /= 0) then
            entropy <- entropy - p*(log(p))

-> (contrast,homogeneity,entropy)

function compareImage(b64_1,b64_2) -> integer
{menerima path dua buah file gambar lalu mengembalikan nilai cosine similarity
antara keduanya}
```

KAMUS LOKAL

```
vector1, vector2 : tuple
similarity : integer
```

ALGORITMA

```
vector1 <- getTexture(b64_1)
vector2 <- getTexture(b64_2)

similarity <- cosSim(vector1, vector2)

-> similarity
```

3. Algoritma [Program Utama/Website]

```
procedure async sendPostDataset()

{digunakan untuk mengirim array base64 (dataset) ke API}
```

KAMUS LOKAL

```
response : JSON
```

ALGORITMA

```
try

    response <- await fetch("http://127.0.0.1:5000/api/receive")

        method: "POST"

        headers: "content-Type" : "application/json"

        body : JSON.stringify(data: dataSetImage)

    if (response.ok) then

        output("POST request successful")

    else

        output("Error")

catch(error)

    output("Error")
```

```

procedure sendPostRefImage()

{digunakan untuk mengirim reference image dan menerima response dalam bentuk
array base 64 yang sudah terurut serta value - value nya}

KAMUS LOKAL

response : JSON

startTime , endTime : real

ALGORITMA

try

    startTime <- performance.now()

    response <- await fetch("http://127.0.0.1:5000/api/RefImage")

        method: "POST"

        headers: "content-Type" : "application/json"

        body : JSON.stringify(Ref: refimage, Type: isChecked)

    if (response.ok) then

        endTime = performance.now()

        setProcessTime(endTime - startTime)

        responseBody <- await response.json()

        setResponseDAta(responseBody)

        output("POST request successful")

    else

        output("Error")

catch(error)

    output("Error")

```

4. Algoritma [app.py]

```

{utamanya digunakan untuk API}

function getColor(string: base64) → array

KAMUS LOKAL

rgb_array , blocks_rgb_array, block_hsv_array : array

```

ALGORITMA

```
rgb_array ← get_rgb_array_from_image(base64)
blocks_rgb_array ← create_blocks_array(rgb_array)
block_hsv_array ← create_hsv_array_from_rgb_array(blocks_rgb_array)
→ block_hsv_array
```

function pangkasBase64(string base64) → string

KAMUS LOKAL

split_string : string

ALGORITMA

```
split_string ← base64.split(',',1)
if len(split_string) > 1 then
    → split_string[1].strip()
else
    → base64
```

@app.route('/api/receive', methods=['POST'])

function receive_strings() → JSON

KAMUS

data : JSON

received_data, dataset: array of string

ALGORITMA

```
data ← request.get_json()
if "data" in data and isinstance(data["data"],list) then
    received_data ← data["data"]
    global dataset
    dataset ← received_data
    → jsonify({"message": "okay received", "dataset" : dataset})
```

```

else
    → jsonify({"error": "Invalid input format"}), 400

@app.route('/api/RefImage', methods=['POST'])

function receive_RefImage() → JSON

KAMUS LOKAL

data : JSON

received_data : string

type , ref : boolean

pairData : array of dictionary

ALGORITMA

data ← request.get_json()

if "Ref" and "Type" in data and isinstance(data["Ref"], str)then

    received_data = data["Ref"]

    type ← data["Type"]

    global ref

    ref ← pangkasBase64(received_data)

    pairData ← []

    if type = False then

        refTexture ← getTexture(ref)

        i traversal dataset

        textureI ← getTexture(pangkasBase64(i))

        pair ← ImageVal(pangkasBase64(i),cosSim(refTexture,textureI))

        if not math.isnan(pair.Val) then

            pair.Val ← round((pair.Val*100),2)

            pairData.append(pair.to_dict())

    else

        refColor ← getColor(ref)

```

```

    i traversal [0...len(dataset)]

    ColorI ← getColor(pangkasBase64(dataset[i]))

    pair ←
ImageVal(pangkasBase64(dataset[i]),average_cos_similarity(refColor,ColorI))

    if not math.isnan(pair.Val) then

        pair.Val ← round((pair.Val*100),2)

        pairData.append(pair.to_dict())

pairData = sorted(pairData, key=lambda d: d['Val'], reverse=True)

→ jsonify({"message": "Hasil", "pair_values" : pairData,"state": type})

```

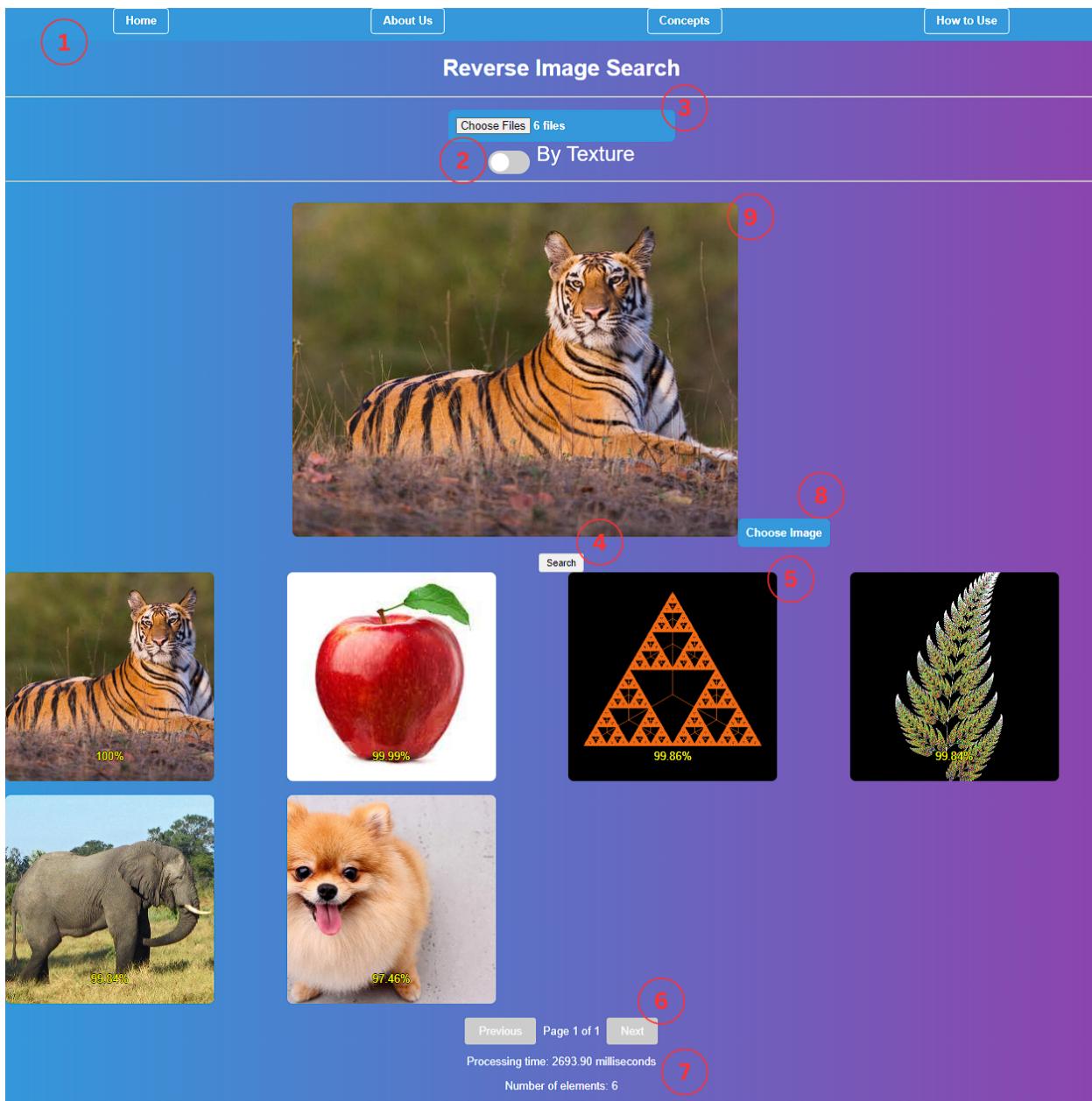
B. Penjelasan struktur program berdasarkan spesifikasi

Kami menggunakan framework React dengan bahasa TypeScript untuk membuat front-end nya. Di bagian front-end, akan diminta inputan dataset. Dataset tersebut akan di-convert ke bentuk base64 dan dimasukkan ke dalam array untuk dikirim ke API. Reference image juga dikirim dalam bentuk base64.

Back-end dari website kami menggunakan bahasa python dan berguna untuk mengubah sebuah gambar menjadi numerical value berdasarkan parameter yang dipilih. Numerical value tersebut akan dijadikan acuan untuk sortir serta menjadi value persen kemiripan.

Oleh karena back-end kami menggunakan bahasa Python, untuk membantu dalam pembuatan API, digunakan framework Flask. Terdapat dua API yang dibuat yaitu /api/receive dan /api/RefImage. /api/receive adalah API bermetode POST yang digunakan untuk menerima dataset dari front-end. /api/RefImage/ juga merupakan API bermetode POST, namun gunanya adalah untuk menerima reference image serta parameter apa yang akan digunakan untuk temu balik. Setelah API tersebut menerima kedua data tersebut, maka API akan memproses datanya dan mengembalikan hasil dataset yang telah disortir serta kemiripan - kemiripannya.

C Penjelasan tata cara penggunaan program



Komponen - komponen pada website:

1. Navbar: digunakan untuk navigasi ke page - page yang bersangkutan
2. Toggle : digunakan untuk mengeset parameter pencarian, by texture atau by color
3. dataset input : menerima folder yang berisi gambar - gambar yang akan diurutkan berdasarkan kemiripannya
4. Tombol search : diklik setelah sudah memilih gambar referensi (8) dan akan menampilkan hasil temu balik
5. Gambar hasil temu balik : awalnya akan kosong, setelah di klik search dan pemrosesan selesai, akan ditampilkan
6. Tombol previous dan Next : digunakan untuk navigasi pagination

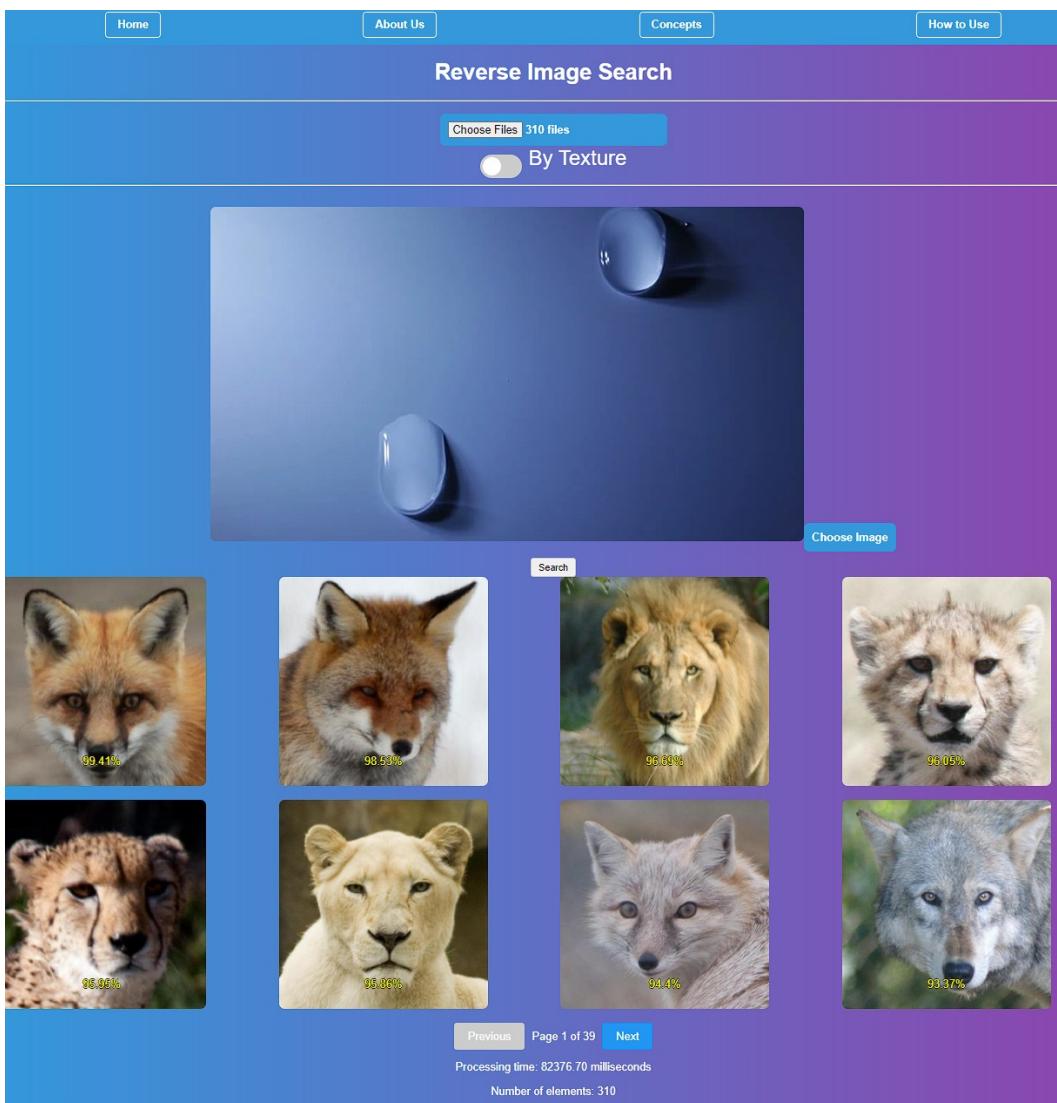
7. Display waktu pemrosesan dan banyak gambar
8. Tombol choose image : digunakan untuk memilih reference image yang akan dibandingkan dengan gambar - gambar dari dataset.
9. Display reference image : awalnya akan kosong, akan mendisplay reference image setelah gambar dipilih

Cara penggunaan program :

1. Masuk ke menu Home.
2. Klik choose file dan masukkan folder yang isinya dataset gambat- gambar yang ingin diuji.
3. Klik toggle bila anda ingin mengganti parameter pencarian, ada dua pilihan, dibandingkan terhadap warna atau dibandingkan terhadap tekstur.
4. Klik choose image untuk memilih gambar apa yang ingin di cari.
5. Klik search
6. Tunggu beberapa saat
7. Program akan menampilkan gambar - gambar dari dataset yang telah diurut dari kemiripannya

D. Hasil pengujian

1. CBIR dengan Parameter Tekstur



Home About Us Concepts How to Use

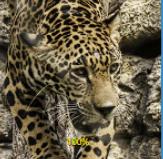
Reverse Image Search

Choose File 510 files

By Texture



Choose Image

Search

Previous Page 1 of 39 Next

Processing time: 84018.00 milliseconds

Number of elements: 310

Home About Us Concepts How to Use

Reverse Image Search

Choose Files 310 files

By Texture



Choose Image

Search



Previous Page 1 of 39 Next

Processing time: 81309.40 milliseconds

Number of elements: 310

[Home](#)[About Us](#)[Concepts](#)[How to Use](#)

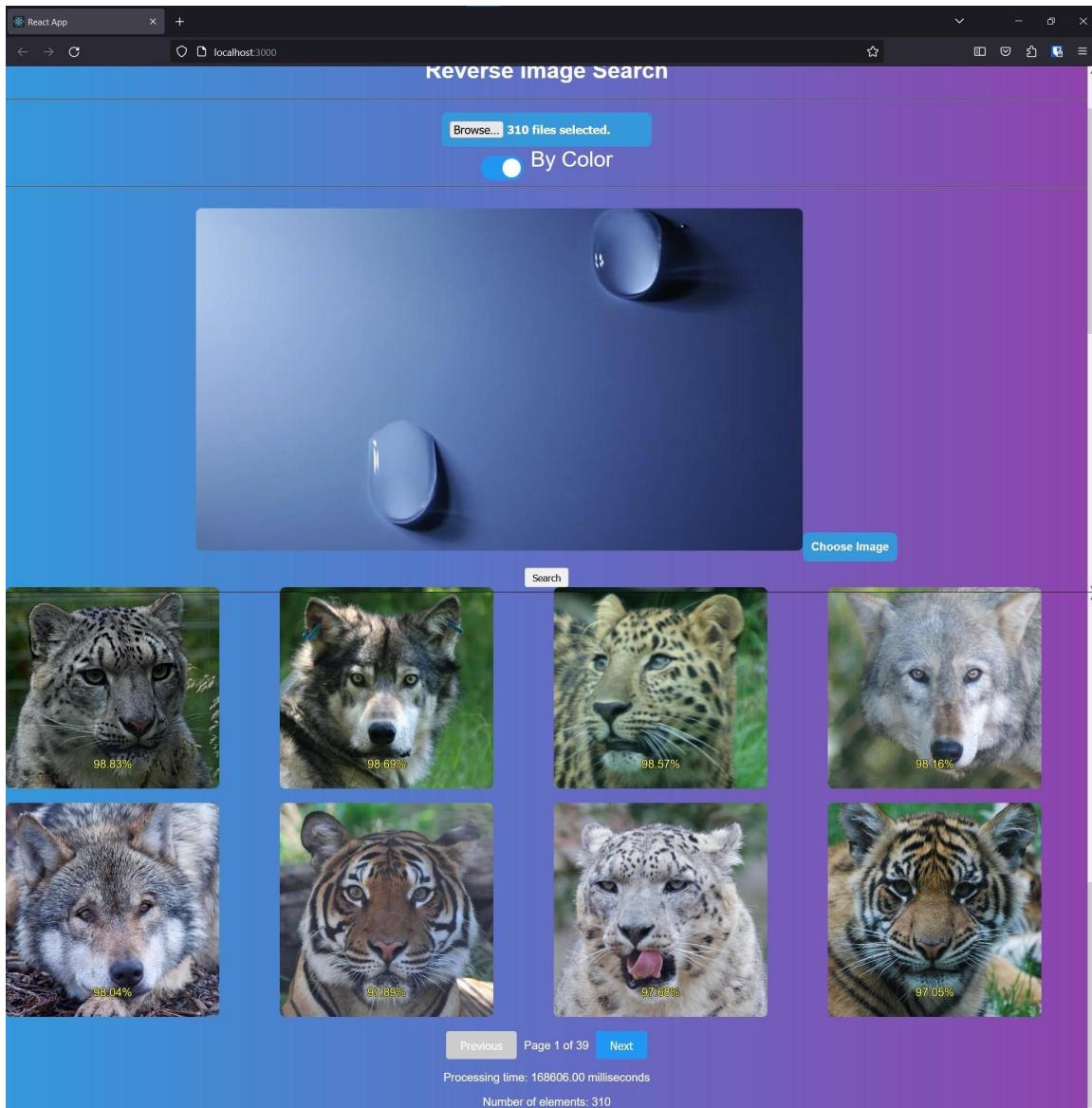
Reverse Image Search

[Choose Files 310 files](#) By Texture[Choose Image](#)[Search](#)[Previous](#) Page 1 of 39 [Next](#)

Processing time: 79753.80 milliseconds

Number of elements: 310

2. CBIR dengan Parameter Warna



React App +

localhost:3000

Reverse Image Search

Browse... 310 files selected.

By Color



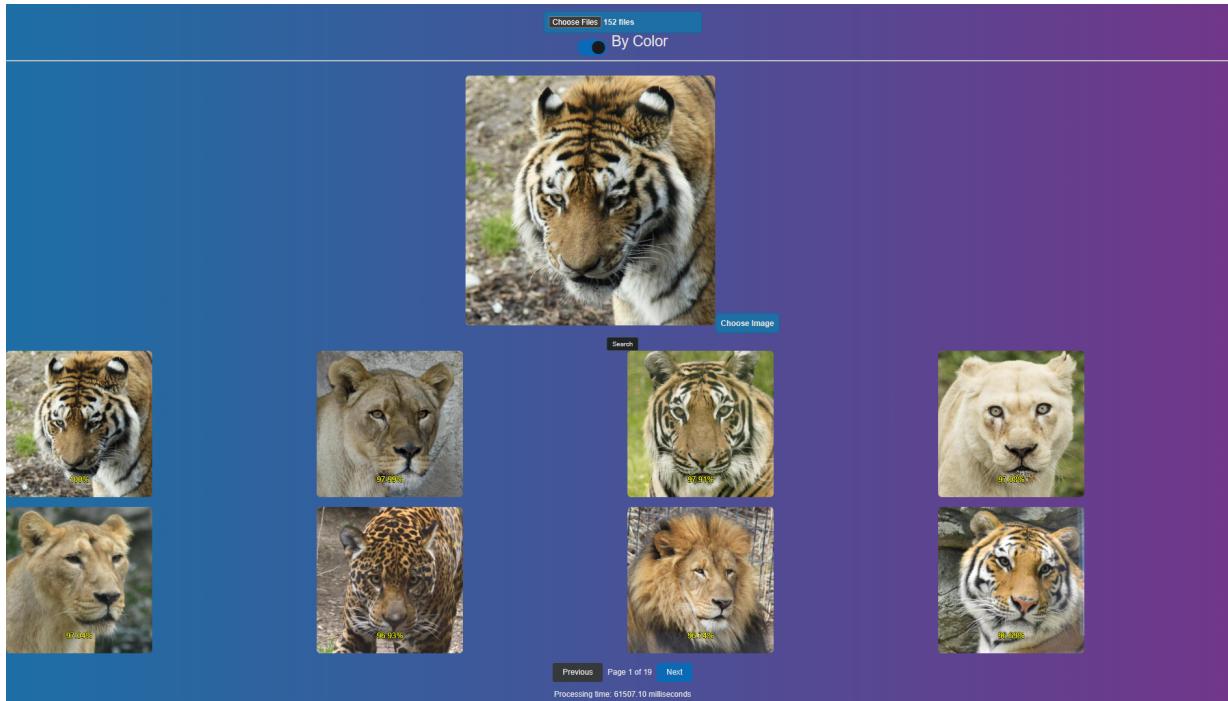
98.61% 98.57% 98.51% 98.49%

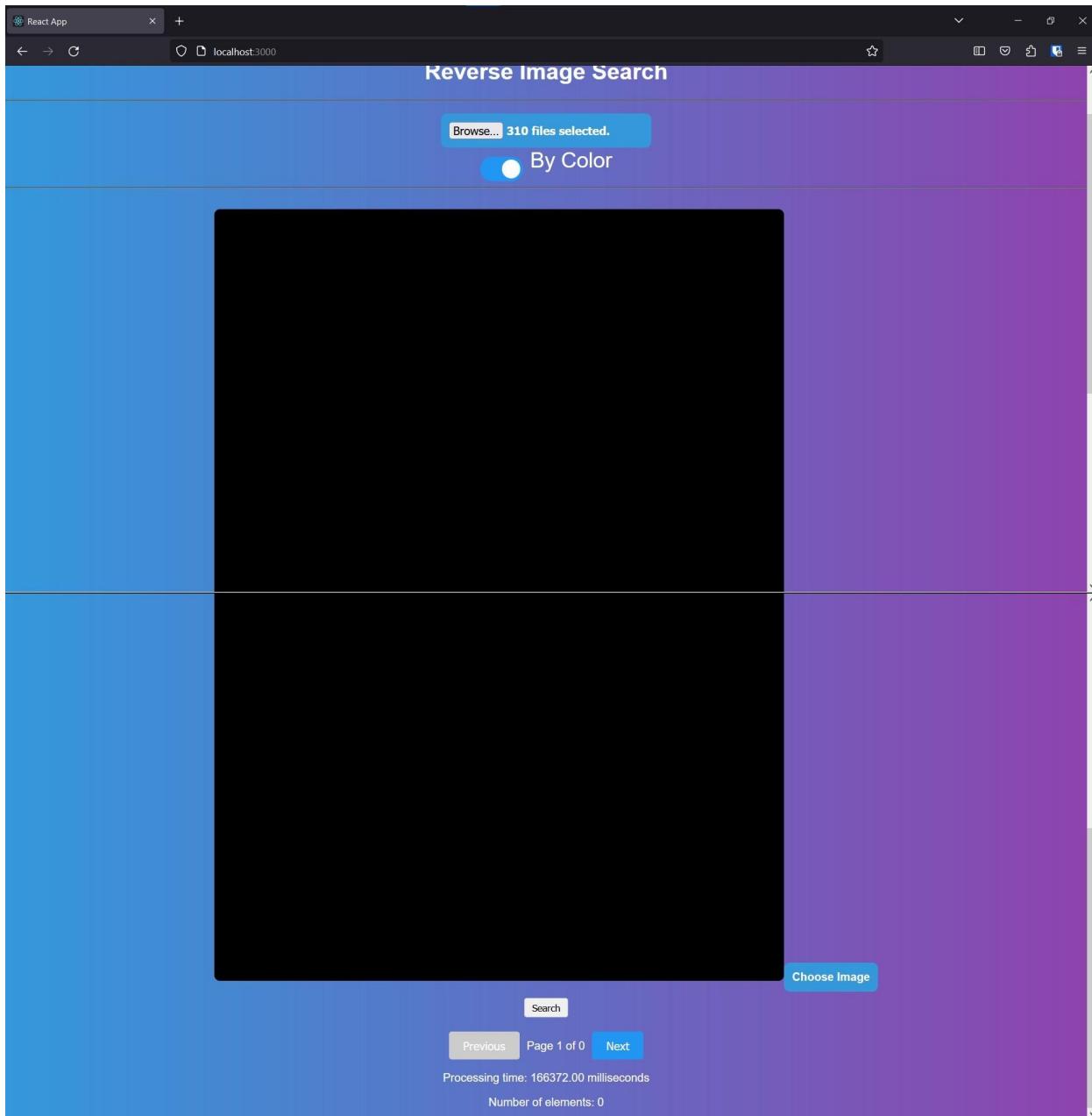
98.29% 98.23% 98.07% 97.98%

Previous Page 1 of 39 Next

Processing time: 164725.00 milliseconds

Number of elements: 310





E. Analisis dari desain algoritma pencarian

CBIR dengan parameter tekstur cenderung lebih sulit membedakan gambar dibandingkan CBIR dengan parameter warna. Nilai contrast yang jauh lebih tinggi daripada nilai homogeneity dan entropy cenderung membuat homogeneity dan entropy memiliki dampak sedikit terhadap nilai cosine similarity. Penggunaan banyak *co-occurrence matrix* dengan bermacam variasi *offset* dan *angle* (θ) dapat meningkatkan keakuratan komparasi gambar. Di sisi lain, CBIR dengan parameter tekstur lebih cepat daripada dengan parameter warna. CBIR dengan parameter warna perlu melewati lebih banyak tahap pemrosesan dibandingkan CBIR dengan parameter tekstur.

BAB 5

KESIMPULAN DAN SARAN

A. Kesimpulan

Pemrosesan dengan tekstur cenderung memberikan nilai kemiripan yang sangat tinggi. Dari sisi waktu eksekusi, pemrosesan dengan tekstur membutuhkan waktu sekitar 0,26 detik per gambar, sedangkan warna membutuhkan waktu sekitar 0,36 detik per gambar sehingga pemrosesan gambar dengan parameter tekstur lebih cepat dibanding warna. Hal tersebut terjadi karena pemrosesan dengan warna membutuhkan kalkulasi yang lebih kompleks dibandingkan pemrosesan dengan tekstur. Diperlukan kalkulasi yang melibatkan 3 komponen warna, yaitu *red*, *green*, dan *blue* (RGB) yang kemudian dikonversi menjadi *hue*, *saturation*, dan *value* (HSV). Pada pemrosesan dengan tekstur, warna tidak terlalu berpengaruh. Nilai yang diperlukan hanya grayscale.

[Dari sisi pembuatan website, pembuatan website dinamis dengan react lebih mudah dibandingkan dengan vanila karena lebih mudah dalam hal penambahan komponen. Integrasi typescript pada tiap komponen juga sangat membantu karena tidak harus menge-set *id* dari tiap komponen, tetapi terkadang menyulitkan karena diperlukannya spesifikasi data type.]

B. Saran

Pemrosesan tekstur dapat menggunakan beragam *offset* dan *angle*, lalu memanfaatkan standar deviasi untuk mendapat nilai akhir komponen-komponen tekstur. Dalam pemrosesan dengan warna, terdapat gambar yang tidak bisa dibandingkan karena tidak terdefinisi nilai cosine similarity-nya. Kasus yang ditemukan adalah kasus gambar berwarna hitam. Diperlukan penanganan khusus atau metode lain untuk mengatasi kasus tersebut.

C. Refleksi

Pemrosesan tekstur masih relatif mendasar karena masih hanya menggunakan 1 variasi *offset* dan *angle*. Pemrosesan warna masih kurang cocok karena hanya melakukan perbandingan *cosine similarity* dengan data rata-rata HSV. Penggunaan python untuk backend mengakibatkan keseluruhan prosesnya menjadi cukup lambat.

D. Komentar atau Tanggapan

Program sudah memenuhi batasan spesifikasi, tetapi masih banyak hal yang dapat dikembangkan atau dioptimalisasi. Pemrosesan gambar dapat dipercepat dengan pengimplementasian *parallel processing* dan sistem *cache*. Sistem temu balik dengan parameter tekstur juga dapat dibuat lebih akurat dengan pembuatan banyak *co-occurrence matrix* dengan variasi *offset* dan *angle* yang berbeda-beda. Website juga dapat dikembangkan lebih lanjut dari sisi penambahan fitur, seperti fitur kamera atau *web scraping*.

E. Ruang Perbaikan atau Pengembangan

Program dapat memanfaatkan parallel processing sehingga banyak foto yang dapat diproses sekaligus sehingga waktu pemrosesan lebih cepat. Program juga dapat dipercepat dengan pengimplementasian sistem cache sehingga program tidak perlu memproses ulang gambar yang telah diproses sebelumnya. Akurasi komparasi gambar dengan parameter tekstur juga dapat ditingkatkan dengan memanfaatkan beragam variasi *offset* dan *angle* dalam pembuatan *co-occurrence matrix*.

REFERENSI

Eichkitz, C.G., Amtmann, J.A., & Schreilechner, M.G. (2013). Calculation of grey level co-occurrence matrix-based seismic attributes in three dimensions. *Comput. Geosci.*, 60, 176-183.

Getting started. Create React App. (n.d.). <https://create-react-app.dev/docs/getting-started>

Jun Yue, Zhenbo Li, Lu Liu, Zetian Fu, Content-based image retrieval using color and texture fused features, *Mathematical and Computer Modelling*, Volume 54, Issues 3–4, 2011, Pages 1121-1127, ISSN 0895-7177, <https://doi.org/10.1016/j.mcm.2010.11.044>. (<https://www.sciencedirect.com/science/article/pii/S0895717710005352>)

Tech With Tim. (2023, May 23). *Create a python API in 12 minutes* [Video]. YouTube. <https://www.youtube.com/watch?v=zsYIw6RXjfM&t=433s>

Yunus, M. (2020, July 16). Feature Extraction: Gray Level Co-occurrence matrix (GLCM). Medium. <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1>

Link Repository: <https://github.com/Leaguemen/Algeo02-22002.git>