

## **Tugas Besar 2**

**IF 2123 Aljabar Linear dan Geometri**

**Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar**

**Semester I Tahun 2023/2024**



**Kelompok Apick :**

- 1. Eduardus Alvito K.      13522004**
- 2. Edbert Eddyson G.      13522039**
- 3. Keanu Amadius G. W. 13522082**

**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
BAB III Analisa Pemecahan Masalah	5
BAB IV Implementasi dan Uji Coba	9
BAB V Kesimpulan	22
Referensi	24

## **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.

Di dalam Tugas Besar 2 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 II**

### **Landasan Teori**

#### **1. CONTENT-BASED INFORMATION RETRIEVAL (CBIR)**

Content Based Image Retrieval System (CBIR) merupakan suatu metode yang digunakan dalam pencarian citra dengan cara melakukan perbandingan antara citra yang ada didatabase dengan citra query berdasarkan informasi yang ada pada citra tersebut. Proses ini dimulai dengan ekstraksi fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk. Lalu mereka diwakili dalam bentuk vektor yang nantinya dapat dibandingkan dengan gambar lain. Kemudian, CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan ini digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling mirip dengan gambar yang dicari. Proses CBIR membantu pengguna dalam mengakses dan mengeksplorasi koleksi gambar dengan cara yang lebih efisien, karena tidak memerlukan pencarian berdasarkan teks atau kata kunci, melainkan berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut.

#### **CBIR dengan parameter warna**

CBIR dengan parameter warna akan mengubah image yang berbentuk RGB menjadi sebuah metode histogram warna yang lebih umum. Histogram warna merupakan suatu proses untuk meratakan histogram agar derajat keabuan dari yang paling rendah (0) sampai dengan yang paling tinggi (255) mempunyai kemunculan yang rata.. Namun, histogram warna tidak bisa mendeskripsikan posisi dari warna yang didistribusikan dan tidak bisa mendeteksi objek yang spesifik yang terdapat pada image.

Pembentukan ruang warna dilakukan untuk membagi nilai citra menjadi beberapa range yang lebih kecil dengan menganggap setiap interval tiap range sebagai bin. Histogram warna dapat dihitung dengan menghitung rata rata interval piksel yang menyatakan nilai warna pada setiap interval. Biasanya menggunakan warna global HSV karena warna tersebut dapat digunakan pada kertas (background berwarna putih) yang lebih umum untuk digunakan.

### **CBIR dengan parameter tekstur**

CBIR dengan perbandingan tekstur dapat dilakukan dengan co-occurrence matrix. Matriks ini dapat melakukan pemrosesan yang lebih mudah dan cepat dan vektor yang dihasilkan mempunyai ukuran yang lebih kecil.

Gray-Level Co-occurrence matrix (GLCM) merupakan teknik analisis tekstur pada citra. GLCM merepresentasikan hubungan antara 2 pixel yang bertetanggaan (neighboring pixels) yang memiliki intensitas keabuan (grayscale intensity), jarak dan sudut. Terdapat 8 sudut yang dapat digunakan pada GLCM, diantaranya sudut  $0^\circ$ ,  $45^\circ$ ,  $90^\circ$ ,  $135^\circ$ ,  $180^\circ$ ,  $225^\circ$ ,  $270^\circ$ , atau  $315^\circ$ .

Setelah didapat co-occurrence matrix, akan digunakan symmetric matrix dengan menjumlahkan co-occurrence matrix dengan hasil transpose-nya. Lalu mencari matrix normalization dengan persamaan.

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

### BAB III

#### Analisa Pemecahan Masalah

##### A. Langkah Langkah dalam penyelesaian masalah

###### 1. CBIR dengan parameter warna

Hal pertama yang harus dilakukan adalah membagi menjadi 4x4 blok. Lalu, mengubah RGB tiap blok menjadi HSV dengan normalisasi nilai RGB dari range [0 sampai 255] menjadi [0 sampai 1]. Setelah itu kita mencari nilai maksimal sekaligus minimal dari (R,G,B) sehingga didapatkan  $\Delta$ . Maka akan bisa dicari nilai HSV dengan rumus berikut :

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

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

$$V = C_{\max}$$

Selanjutnya, dengan nilai HSV kita bentuk histogramnya dengan pembagian (8 Hue, 3 Saturation, 3 Value) untuk tiap region/ tiap blok. Lalu kita dapat melakukan perbandingan antara image dari input dengan dataset dengan menggunakan cosine similarity dari vektor histogram untuk setiap blok/ region.

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

Dengan  $A$  dan  $B$  adalah vektor dan  $n$  adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*.

Semua hasil cosine similarity untuk tiap blok atau region dijumlahkan dan dibagi dengan 16 (jumlah total blok / region)

## 2. CBIR dengan parameter tekstur

Konversikan gambar menjadi grayscale yg berukuran 256 piksel. Hal ini dilakukan karena warna tidak penting dalam CBIR parameter tekstur. Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut. Lalu kita membuat co-occurrence matrix dari grayscale tersebut.

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

GLCM

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

Dari hasil co-occurrence matrix, kita bisa mencari contrast, entropy dan homogeneity dengan 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 :*

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

Dimana  $P$  merupakan matriks co-occurrence.

Selanjutnya kita melakukan perbandingan antara image dari input dengan dataset dengan menggunakan cosine similarity

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

Dengan  $A$  dan  $B$  adalah vektor dan  $n$  adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*.

### 3. Web

Pengguna diminta untuk upload data set gambar. Upload data set gambar dilakukan dengan select multiple image lalu upload ke tombol data set. Setelah itu pengguna juga diminta untuk upload image query yang akan dibandingkan kemiripannya.

Program akan mengkonversi tiap tiap gambar menjadi vektor dimana vektor tersebut akan dibandingkan dengan cosine similarity, Setelah didapatkan hasil cosine similarity, web akan menampilkan kembali gambar gambar dari data set yang memiliki kemiripan diatas 60% .

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

Aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (Content-Based Image Retrieval atau CBIR). CBIR inilah yang nantinya dapat digunakan untuk membantu pengguna dalam mengakses dan mengeksplorasi koleksi gambar dengan cara yang lebih efisien berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut.

Tiap tiap gambar akan didapatkan sebuah vektor melalui cara cara yang telah disebutkan sebelumnya. Setelah mendapatkan vektor vektor dari setiap gambar, vektor vektor tersebut akan dibandingkan dengan konsep Aljabar Geometri yaitu cosine similarity.



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

### C. Contoh ilustrasi kasus dan penyelesaiannya.

Ketika ingin mencari gambar yang mirip dengan yang ada di dataset. Kita dapat menggunakan web ini yang nantinya akan membentuk vektor untuk setiap gambar dan akan dibandingkan dengan cosine similarity.

Ketika ingin mencari gambar yang paling mirip dalam sebuah web, maka kita dapat menggunakan sistem temu balik gambar ini untuk mencarinya. Caranya dengan mengimplementasikan CBIR tadi untuk mencari kemiripan suatu gambar. Untuk mendapatkan dataset gambar dalam web, digunakan image scraping.

## BAB IV

### Implementasi dan Uji Coba

#### 1. Implementasi program utama

##### Algoritma COLOR.go

```
function rgbToHSV(rgb)
```

```
Deklarasi :  
var h, s, v
```

```
Algoritma:
```

```
r = rgb.R / 255.0
```

```
g = rgb.G / 255.0
```

```
b = rgb.B / 255.0
```

```
max = max(max(r, g), b)
```

```
min = min(min(r, g), b)
```

```
if max == min:
```

```
    h = 0
```

```
else if max == r:
```

```
    h = (60 * ((g - b) / (max - min)) + 360) % 360
```

```
else if max == g:
```

```
    h = (60 * ((b - r) / (max - min)) + 120) % 360
```

```
else if max == b:
```

```
    h = (60 * ((r - g) / (max - min)) + 240) % 360
```

```
if max == 0:
```

```
    s = 0
```

```
else:
```

```
    s = (max - min) / max
```

```
v = max
```

```
return h, s, v
```

```
function max(a, b)
```

```
Algoritma:
```

```
if a > b:
```

```
    return a
```

```
return b
```

```
function min(a, b)
```

Algoritma:

```
if a < b:  
    return a  
return b
```

```
function processPixel(rgba)
```

Algoritma:

```
h, s, v = rgbToHSV(rgba)  
return h, s, v
```

```
function checkH(h)
```

Algoritma:

```
if (316 <= h <= 360) or (h == 0):  
    return 0  
else if (1 <= h <= 25):  
    return 1  
else if (26 <= h <= 40):  
    return 2  
else if (41 <= h <= 120):  
    return 3  
else if (121 <= h <= 190):  
    return 4  
else if (191 <= h <= 270):  
    return 5  
else if (271 <= h <= 295):  
    return 6  
else if (296 <= h <= 315):  
    return 7  
return 0
```

```
function checkS(s)
```

Algoritma:

```
if (0 <= s < 0.2):  
    return 0  
else if (0.2 <= s < 0.7):  
    return 1
```

```
else if (0.7 <= s <= 1):  
    return 2  
return 0
```

```
function checkV(v)
```

Algoritma:

```
if (0 <= v < 0.2):  
    return 0  
else if (0.2 <= v < 0.7):  
    return 1  
else if (0.7 <= v <= 1):  
    return 2  
return 0
```

```
function processRegion(img, xStart, yStart, xEnd, yEnd)
```

Deklarasi :

```
index_h = integer  
index_s = integer  
index_v = integer  
array_h = [0, 0, 0, 0, 0, 0, 0, 0]  
array_s = [0, 0, 0]  
array_v = [0, 0, 0]
```

Algoritma:

```
for y from yStart to yEnd:  
    for x from xStart to xEnd:  
        colorAt = img.getPixel(x, y)  
        c = convertToRGBA(colorAt)  
        h, s, v = processPixel(c)  
        h = round(h)  
        index_h = checkH(h)  
        index_s = checkS(s)  
        index_v = checkV(v)  
        array_h[index_h] += 1  
        array_s[index_s] += 1  
        array_v[index_v] += 1  
  
newValues = [array_h[0], array_h[1], array_h[2],  
array_h[3], array_h[4], array_h[5], array_h[6],  
array_h[7], array_s[0], array_s[1], array_s[2],
```

```
array_v[0], array_v[1], array_v[2]]  
  
return mat.NewVecDense(14, newValues)
```

```
function processPicture(filename)
```

Algoritma:

```
arr_vector = createEmptyArray()  
  
file = openFile(filename)  
if file is null:  
    return arr_vector  
  
img = decodeImage(file)  
if img is null:  
    return arr_vector  
  
regionSizeX = img.width / 4  
regionSizeY = img.height / 4  
  
for i from 0 to 3:  
    for j from 0 to 3:  
        xStart = i * regionSizeX  
        yStart = j * regionSizeY  
        xEnd = (i + 1) * regionSizeX  
        yEnd = (j + 1) * regionSizeY  
        arr_vector[i][j] = processRegion(img, xStart,  
yStart, xEnd, yEnd)  
  
return arr_vector
```

```
function cosineSimilarity(vektor1, vektor2)
```

```
total = 0  
for i from 0 to 3:  
    for j from 0 to 3:  
        total += dotProduct(vektor1[i][j],  
vektor2[i][j]) / (magnitude(vektor1[i][j]) *  
        magnitude(vektor2[i][j]))  
  
return total / 16
```

### Algoritma TEXTURE.go

```
function greyScale2(matrix, img)
```

Algoritma:

```
bounds = img.Bounds()
for y from 0 to bounds.Max.Y - 1:
    for x from 0 to bounds.Max.X - 1:
        colorAt = img.At(x, y)
        c = convertToRGBA(colorAt)
        matrix[x][y] = uint8(c.R*0.299 + c.G*0.587 +
c.B*0.114)
```

```
function featureExtraction(fm):
```

Deskripsi:

```
colVec = mat.VecDense
var mSquare = mat.Dense
```

Algoritma:

```
rows, _ = fm.Dims()
matrixIndex = createVector(0, 1, 2, ..., rows - 1)
colVec = createColumnVector(matrixIndex)

mSquare = createMatrix(rows, rows)

for i from 0 to rows - 1:
    for j from 0 to rows - 1:
        mSquare[i][j] = (matrixIndex[i] - colVec[i]) *
(matrixIndex[j] - colVec[j])

for i from 0 to len(fm) - 1:
    contrast += mSquare[i] * fm[i]
    homogeneity += fm[i] / (1 + mSquare[i])
    if fm[i] == 0:
        entropy += 0
    else:
        entropy += fm[i] * log(fm[i])

entropy = -entropy

return createVector(contrast, homogeneity, entropy)
```

```
function createCoOccurence(matrix)
```

Algoritma:

```
occurence0 = mat.NewDense(256, 256, nil)

xlen = len(matrix)
ylen = len(matrix[0])

for y from 0 to ylen - 2:
    for x from 0 to xlen - 1:
        i = matrix[x][y]
        j = matrix[x][y + 1]
        occurence0[i][j] += 1

transpose0 = mat.DenseCopyOf(occurence0.T())
sum0 = occurence0 + transpose0

data0 = sum0.RawMatrix().Data
norm_val = floats.Sum(data0)

normalizedMatrix0 = mat.NewDense(256, 256, nil)
normalizedMatrix0.Scale(1/norm_val, sum0Matrix)

return normalizedMatrix0
```

```
function processPicture_texture(filename)
```

Algoritma:

```
file = openFile(filename)
if file is null:
    log.Fatal(err)

img = decodeImage(file)
if img is null:
    log.Fatal(err)

xLen = img.Bounds().Max.X
yLen = img.Bounds().Max.Y

matrix = createMatrix(xLen, yLen)
greyScale2(matrix, img)

occ_matrix = createCoOccurence(matrix)
```

```
vektor = featureExtraction(occ_matrix)

return vektor
```

```
function cosine_similarity_texture(vektor1, vektor2)

Algoritma:

vecDense1 = mat.NewVecDense(len(vektor1), vektor1)
vecDense2 = mat.NewVecDense(len(vektor2), vektor2)
dotProduct = dot(vecDense1, vecDense2)
return dotProduct / (norm(vecDense1) * norm(vecDense2))
```

## 2. Struktur program

Pada bagian front end, digunakan bahasa ReactJS dengan framework yang dipakai adalah tailwind. ReactJS merupakan library JavaScript yang populer dan biasanya digunakan dalam proses pengembangan aplikasi mobile dan web. React berisi kumpulan snippet kode JavaScript yang bisa digunakan berulang kali untuk mendesain antarmuka pengguna. Tailwind CSS merupakan utility-first framework CSS yang didesain agar dapat mempermudah dan mempercepat pembuatan aplikasi menggunakan desain custom. Framework ini mengutamakan utilitas untuk membuat desain khusus. Tailwind menawarkan opinionated building blocks yang dikenal sebagai kelas utilitas untuk membantu mengatur style komponen website.

Komponen pada front end ini yaitu : upload query image untuk upload gambar yg akan dicari kemiripannya, upload data set untuk mengupload data set yang akan dicari kemiripannya, tampil gambar untuk menampilkan gambar, pagination untuk membagi gambar yang akan ditampilkan agar tidak terlalu banyak.

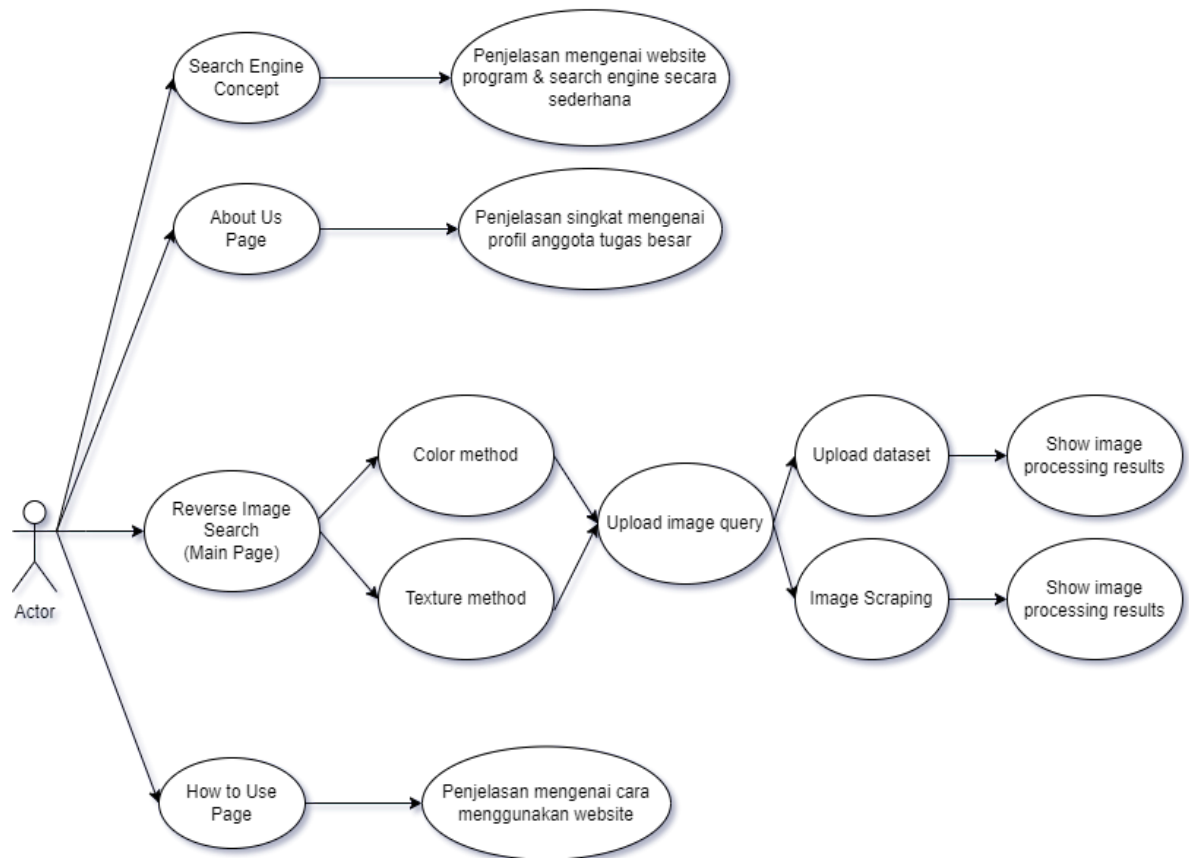
Pada bagian Back End menggunakan bahasa golang dengan framework yang dipakai adalah gin dan gorilla mux. Back End digunakan untuk api yang akan menghubungkan algoritma dan front end.

Pada bagian algoritma utama, digunakan bahasa golang. Terdapat 2 file yaitu file color dan texture. File color berisi pemrosesan CBIR dengan parameter warna. Didalamnya terdapat fungsi mengubah rgb to hsv, mencari histogram untuk tiap blok,



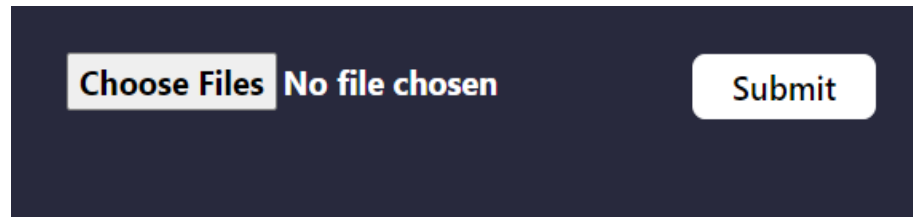
dan cosine similarity untuk mencari kemiripan suatu gambar dengan menjumlahkan hasil cosine similarity dari 16 blok lalu dibagi 16.

### 3. Tata cara penggunaan program

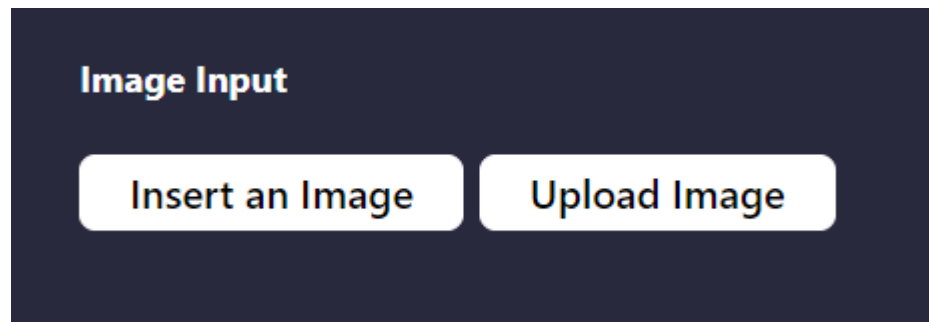


Pada website ini, terdapat 4 halaman, yaitu halaman *Search Engine Concept*, *About Us*, *Reverse Image Search*, dan *How to Use*. Saat program dijalankan, pengguna pertama kali akan masuk ke halaman utama yaitu halaman *Reverse Image Search*.

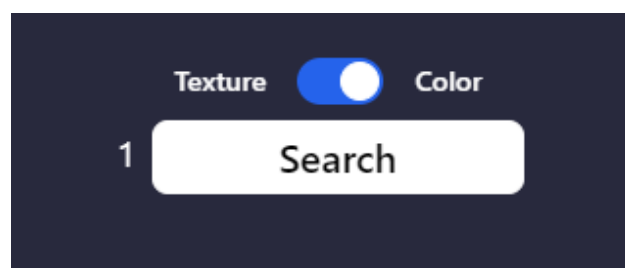
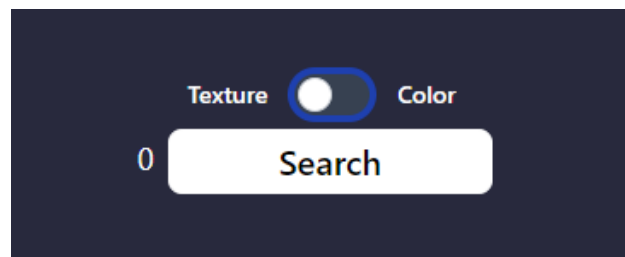
Kita dapat mencari kemiripan gambar query dengan gambar dari data set dengan cara mengupload data set dalam web dengan select multiple image. Pertama tekan tombol choose files dan pilih gambar dengan multiple image. Setelah dipilih, klik submit.



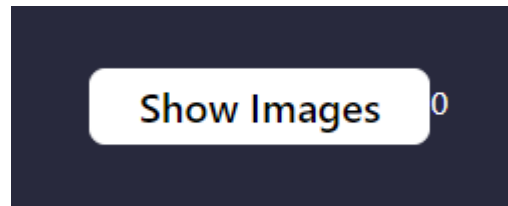
Setelah itu kita diminta untuk upload gambar yang akan dicari kemiripannya dengan cara insert gambar lalu tekan upload gambar.



Langkah terakhir, kita dapat memilih untuk mencari kemiripan gambar dengan metode perbandingan warna atau tekstur. Lalu tekan tombol search.



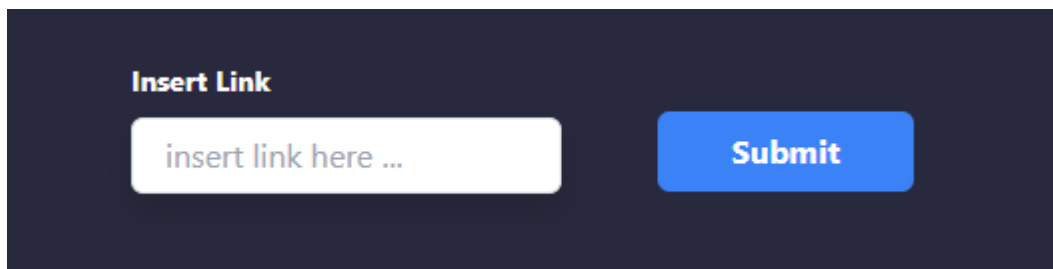
Hasil dapat dilihat dengan menekan tombol show images



Fitur tambahan :

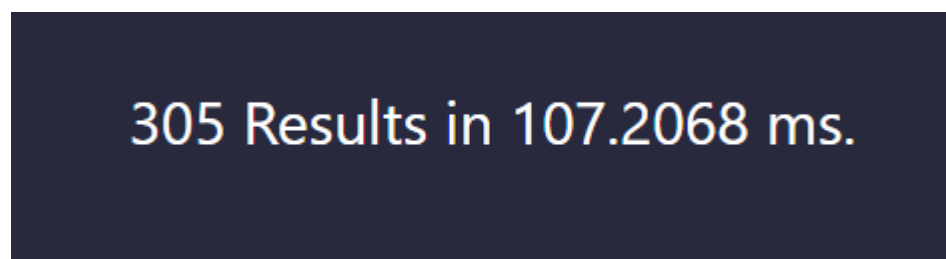
Image Scraping

Kita dapat membandingkan gambar melalui image scraping dengan memasukan link web pada tombol “image scraping”. Hasil dari *scraping* akan digunakan oleh program sebagai input dataset gambar.

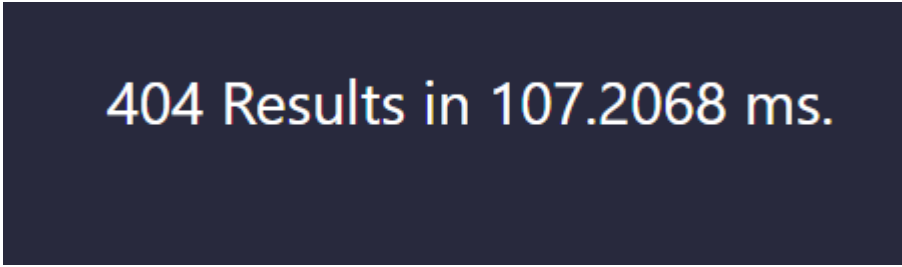


#### 4. Pengujian

Pengujian pertama dilakukan dengan upload dataset sebanyak 404 gambar singa & macan. Proses upload tergolong lama. Dari percobaan menggunakan 400 data set, diperlukan waktu 107.2 ms untuk menampilkan 305 gambar yang tingkat kemiripannya diatas 60 %.



Pengujian kedua dengan switch ke texture, disini program tidak perlu menjalankan proses image ke vektor lagi sehingga waktu yang didapatkan sangat cepat. Namun dari 404 dataset, semua gambar memiliki kemiripan.



404 Results in 107.2068 ms.

Pengujian ketiga dilakukan untuk menguji CBIR dengan parameter color. Dataset yang diberikan merupakan dataset warna. Pengujian dengan warna biru, merah, dll berjalan lancar. Namun pengujian dengan warna hitam, gambar warna putih juga ikut terdeteksi kemiripannya diatas 60%.

Pengujian keempat menggunakan dataset gambar karakter dalam animasi *spongebob*. Image query nya adalah patrick. Dengan CBIR warna, kita mendapatkan hasil yang diinginkan yaitu gambar gambar patrick lainnya. Jika di test dengan texture, tidak semua yang ada di dataset mirip.

Pengujian kelima yaitu memasukan dataset 2 gambar yang sama, namun salah satu gambar warnanya hitam putih. Image querynya yaitu gambar yang berwarna. Dengan texture, didapatkan result 99,99% terhadap 2 gambar di dataset. Namun dengan parameter warna, hanya didapatkan hasil 100% yaitu gambar berwarna dari dataset.

Pengujian terakhir yaitu menggunakan web scraping. Web yang digunakan adalah web XXI. Program berjalan dengan lancar baik dengan texture atau color.

Ada beberapa kasus pengujian dimana hasil persen yang didapat melebihi 100% namun itu sangat kecil yaitu 100,00000000000002.

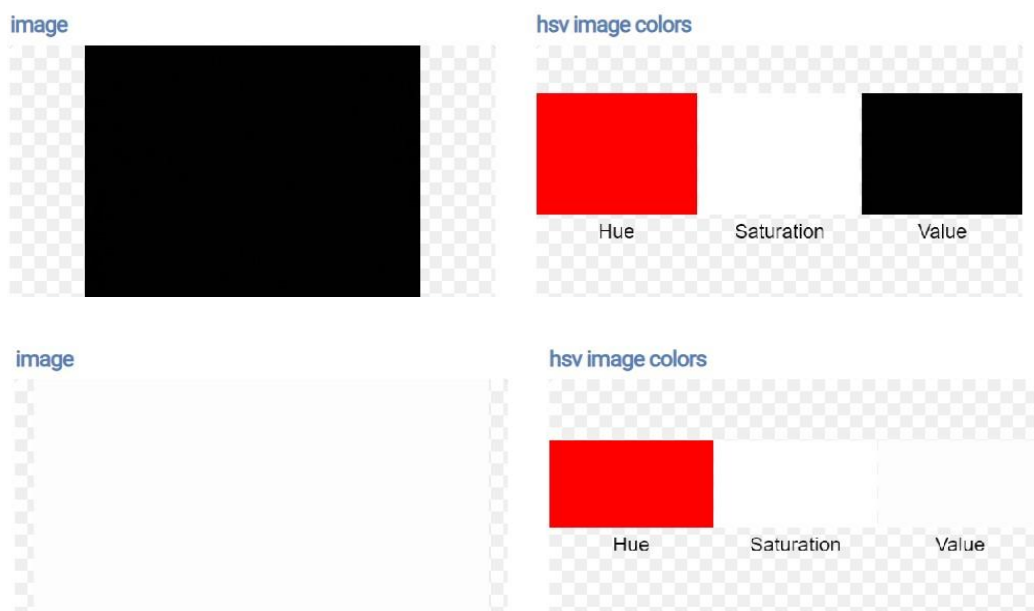
## 5. Analisis

Analisis pengujian pertama. Proses upload tergolong lama karena program sekaligus menjalankan proses picture menjadi vektor baik color maupun texture. Namun keuntungannya kita dapat switch untuk menampilkan hasil dari warna atau texture dengan cepat.

Analisis pengujian kedua. Didapatkan result 404 dari 404 dataset. hal ini disebabkan karena datasetnya merupakan potongan muka singa / macan & sejenisnya.

Sehingga jika dilakukan dengan parameter texture, ini akan terlihat mirip. Sebaliknya, jika digunakan parameter warna maka gambar yang memiliki perbedaan warna yang jauh tidak akan muncul / kemiripan dibawah 60%.

Analisis pengujian ketiga. CBIR Warna membandingkan 2 gambar dengan vektor 14 parameter (8 Hue, 3 Saturation, 3 Value). Namun, apabila gambar yang dibandingkan adalah full warna hitam dan full warna putih, maka akan menampilkan hasil 60% keatas. Hal ini terjadi karena Hue dan Saturation warna hitam dan putih sama



Karena yang menjadi pembeda hanya value saja, maka hasil cosine similarity akan semakin besar pula.

Analisis pengujian keempat. Program dapat menampilkan gambar patrick lainnya dengan parameter warna karena gambar yang memiliki warna pink hanyalah gambar patrick. Dengan texture, tidak semua yang ada di data set memiliki kemiripan seperti di pengujian pertama. Hal ini terjadi karena data set gambar bervariasi dan tidak mirip seperti pengujian pertama.

Analisis pengujian kelima. Dalam texture didapatkan hasil 99% untuk kedua gambar baik berwarna atau hitam putih karena texture tidak bergantung pada warna. Ia hanya membandingkan grayScale dari sebuah gambar dan yang menjadi parameter pembandingnya yaitu. Sehingga kedua gambar memiliki result 99%. Dengan parameter warna, hanya muncul gambar yang sama / gambar yg berwarna juga. Hal ini karena gambar hitam putih rgb nya sangat berbeda dengan gambar berwarna.

Analisis untuk angka 100,00000000000002 disebabkan oleh tipe datanya yaitu floating point.

## **BAB V**

### **Kesimpulan**

Kita dapat membandingkan kemiripan antara 2 gambar dengan menggunakan CBIR warna ataupun tekstur. Dengan CBIR warna, kita mengkonversikan RGB menjadi HSV (Hue, Saturatin, Value) dengan membagi blok 4x4 yang nantinya akan dimasukan kedalam histogram untuk tiap blok dengan keluaran vektor. Setelah itu vektor untuk setiap blok dari 2 gambar dibandingkan dengan cosine similarity yang nantinya akan dicari rata ratanya dengan memagi total blok (16). Dengan CBIR tekstur, kita membuat co-occurrence matrix untuk mencari contrast, entropy dan homogeneity lalu membandingkannya dengan cosine similarity.

CBIR warna sangat bergantung pada warnanya sehingga hasil yang dikeluarkan merupakan warna yang sama / mirip dengan image query. Namun, pada CBIR warna terdapat suatu error dimana saat kita membandingkan 2 gambar warna hitam full dan warna putih full akan menampilkan hasil 60% keatas. Hal ini disebabkan karena Hue dan Saturation warna hitam dan putih sama, yang membedakan hanya value nya saja.

Menurut kami, keakuratan untuk tiap metode lebih akurat CBIR dengan parameter warna. Hal ini terjadi karena parameter yang digunakan sebagai perbandingannya banyak (14 parameter terdiri dari 8 Hue, 3 Saturation, 3 Value) sedangkan CBIR Texture hanya menggunakan 3 parameter pembeding (contrast, entropy dan homogeneity). Ini menandakan bahwa, semakin besar vektor maka akan semakin akurat hasil yang dihasilkan. Namun menurut kami tetap lebih baik menggunakan machine learning karena hasil yang didapat akan lebih akurat daripada program ini.

Program ini dapat bermanfaat untuk membantu kita dalam mencari kemiripan gambar atau mencari gambar yang sama persis dengan gambar lain yang ada dalam dataset atau menggunakan image scraping. Program ini juga dapat digunakan untuk mengecek apakah kita sudah memiliki gambar dalam suatu data set atau belum, jika sudah maka akan ditampilkan gambar yang memiliki kemiripan 100%.

### **Saran**

Saran dari penulis untuk selanjutnya adalah deadline kurang panjang, masih banyak yang harus eksplor sendiri seperti pada parameter tekstur. Tetap lebih baik menggunakan machine learning karena program ini tetap kurang akurat

### **Komentar atau Tanggapan**

Sejauh ini dari penurunan Tugas Besar dan dari asistensi sudah sangat baik. Asisten sangat sabar dan membantu dalam menyelesaikan Tugas Besar ini.

### **Refleksi**

Tim penulis dapat menyelesaikan tugas besar ini dengan baik dalam kurun waktu yang telah ditentukan. Tim penulis juga menyelesaikan soal bonus yaitu image scraping dan pembuatan video. Tim penulis juga mendapatkan banyak hal baru baik dari kerja sama maupun materi materi baru dan juga memperdalam konsep Algeo beserta implementasinya.

### **Ruang Perbaikan atau Pengembangan**

Better performance dengan mempercepat eksekusi, alur program masih acak acakan. Masih menggunakan 2 frame work. Penerapan clean code dan modularity.



## Referensi

- Syarif, Hisyam, Pulung N. A. (2023). CONTENT BASED IMAGE RETRIEVAL BERBASIS COLOR HISTOGRAM UNTUK PENGKLASIFIKASIAN IKAN KOI JENIS KOHAKU. Jurnal Ilmiah Penelitian dan Pembelajaran Informatika Vol. 8, No. 2, Juni 2023.
- Vieira, Gabriel S., Afonso U. Fonseca, Fabrizzio Soares b (2023). CBIR-ANR: A content-based image retrieval with accuracy noise reduction. Elsevier. v1.0
- Yue, Jun Yue, Zhenbo Li, Lu Liub, Zetian Fub (2011). Content-based image retrieval using color and texture fused features. Elsevier.
- Atlam, Hany Fathy, Gamal Attiya, Nawal El-Fishawy (2013). Comparative Study on CBIR based on Color Feature. International Journal of Computer Applications. Vol. 78 No. 16
- Imran, Sajida & Lodhi, Bilal & Alzahrani, Ali. (2021). Unsupervised Method to Localize Masses in Mammograms. IEEE Access. PP. 1-1.  
10.1109/ACCESS.2021.3094768.