

Tugas Besar 2 Teknik Informatika Aljabar Linier dan Geometri

Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar

EL GATO



13522028 - Panji Sri Kuncara Wisma

13522030 - Imam Hanif Mulyarahman

13522033 - Bryan Cornelius Lauwrence

Sekolah Teknik Elektro dan Informatika

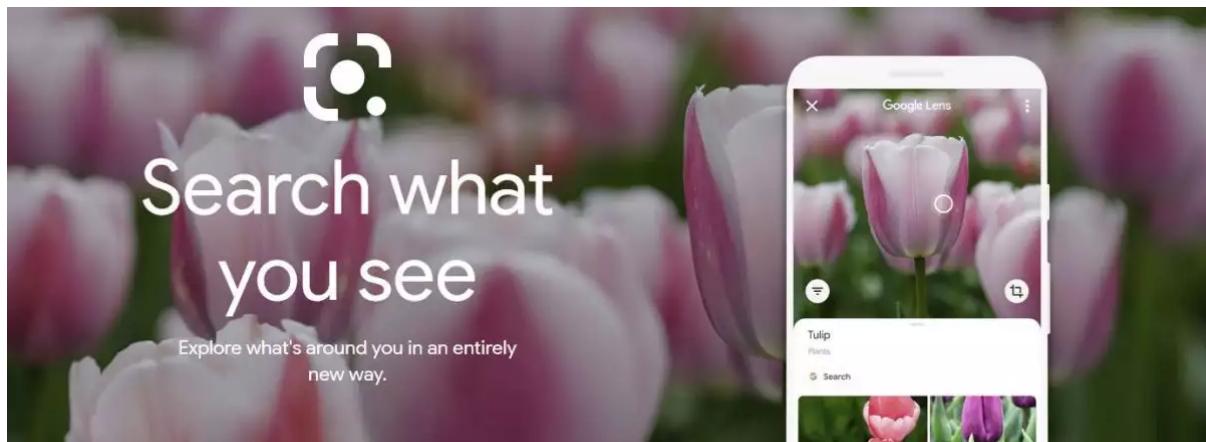
Institut Teknologi Bandung

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 yang mungkin kalian tahu adalah Google Lens.



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

Di dalam Tugas Besar 2 ini, Anda diminta untuk 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:

TEORI SINGKAT

2.1 Content-Based Information Retrieval (CBIR)

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Proses ini dimulai dengan ekstraksi fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk. Setelah fitur-fitur tersebut diekstraksi, mereka diwakili dalam bentuk vektor atau deskripsi numerik yang 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.

2.2 Content-Based Information Retrieval dengan Parameter Warna

CBIR dengan parameter warna berfokus pada membandingkan RGB yang sudah diubah menjadi histogram warna dari satu gambar dengan gambar yang lainnya. Histogram warna adalah frekuensi dari berbagai warna yang ada pada ruang warna tertentu hal ini dilakukan untuk melakukan pendistribusian warna dari *image*. Histogram warna tidak bisa mendeteksi sebuah objek yang spesifik yang terdapat pada *image* dan tidak bisa mendeskripsikan posisi dari warna yang didistribusikan.

Pembentukan ruang warna perlu dilakukan dalam rangka pembagian nilai citra menjadi beberapa *range* yang lebih kecil. Hal itu dilakukan untuk membuat sebuah histogram warna yang setiap interval tiap *range* dianggap sebagai *bin*. Histogram warna dapat dihitung dengan menghitung piksel yang menyatakan nilai warna pada setiap interval. Fitur warna mencakup histogram warna global dan histogram warna blok.

Pada perhitungan histogram, warna global HSV lebih dipilih karena warna tersebut dapat digunakan pada kertas (*background* berwarna putih) yang lebih umum untuk digunakan. Tahapan yang dilakukan dalam proses CBIR ini adalah sebagai berikut:

1. Menormalisasi nilai RGB dengan mengubah nilai *range* [0, 255] menjadi [0, 1]

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

2. Menentukan nilai *Cmax*, *Cmin*, dan Δ

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

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

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

3. Menggunakan hasil perhitungan di atas untuk mendapatkan nilai HSV

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

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

$$V = C_{max}$$

Setelah mendapatkan nilai HSV, kemiripan dari suatu gambar dengan gambar yang lain dapat dicari nilainya dengan rumus *cosine similarity* berikut:

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

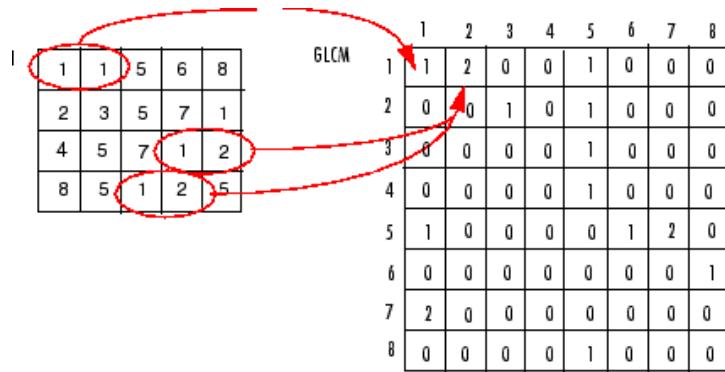
2.3 Content-Based Information Retrieval dengan Parameter Tekstur

CBIR dengan perbandingan tekstur dilakukan dengan 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° dan jarak yang digunakan adalah 1 piksel.



Gambar 2. Cara Pembuatan Matrix *Occurrence*

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

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

Langkah-langkah dalam CBIR dengan parameter tekstur adalah sebagai berikut :

1. Konversi warna gambar menjadi *grayscale*, ini dilakukan karena warna tidaklah penting dalam penentuan tekstur. Oleh karena itu, warna RGB dapat diubah menjadi suatu warna *grayscale* Y dengan rumus:

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

2. Lakukan kuantifikasi dari nilai *grayscale*. Karena citra *grayscale* berukuran 256 piksel, maka matriks yang berkoresponden akan berukuran 256×256 .
3. Dari *co-occurrence matrix* bisa diperoleh 3 komponen utama ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*. Pada tugas besar ini diberikan 3 komponen tambahan ekstraksi tekstur untuk meningkatkan keakuratan program. Komponen tambahan tersebut adalah *dissimilarity*, *ASM*, dan *energy*. Persamaan yang dapat digunakan untuk mendapatkan nilai 6 komponen tersebut antara lain :

Contrast:

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

Homogeneity :

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

Entropy :

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

Dissimilarity :

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

ASM :

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

Energy :

$$\sqrt{ASM}$$

Keterangan : P merupakan matriks *co-occurrence*

Dari keenam komponen tersebut, dibuatlah sebuah vektor yang akan digunakan dalam proses pengukuran tingkat kemiripan.

4. Ukurlah kemiripan dari kedua gambar dengan menggunakan Teorema *Cosine Similarity*, yaitu:

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

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

2.4 Pengembangan Website

Website adalah sekumpulan halaman web yang saling terhubung. Halaman web adalah halaman tunggal yang menampilkan informasi, baik berupa teks, gambar, maupun media lainnya. Web-web tersebut akan hanya bisa dijalankan pada aplikasi browser. Tampilan website dibangun dari layout dasar HTML dan bisa dihias dengan CSS maupun JavaScript. Aplikasi browser akan

menerima informasi dari *template* yang disediakan lalu membacanya dan menampilkannya sesuai dengan informasi yang ada. *Template* tersebut hanya mengurus tampilan dari website (*Front-End*), sedangkan supaya website yang dibuat bisa bekerja sesuai keinginan, diperlukan sebuah *Back-End* sebagai program utama yang mengolah informasi dan API untuk menghubungkan *Front-End* dengan *Back-End*.

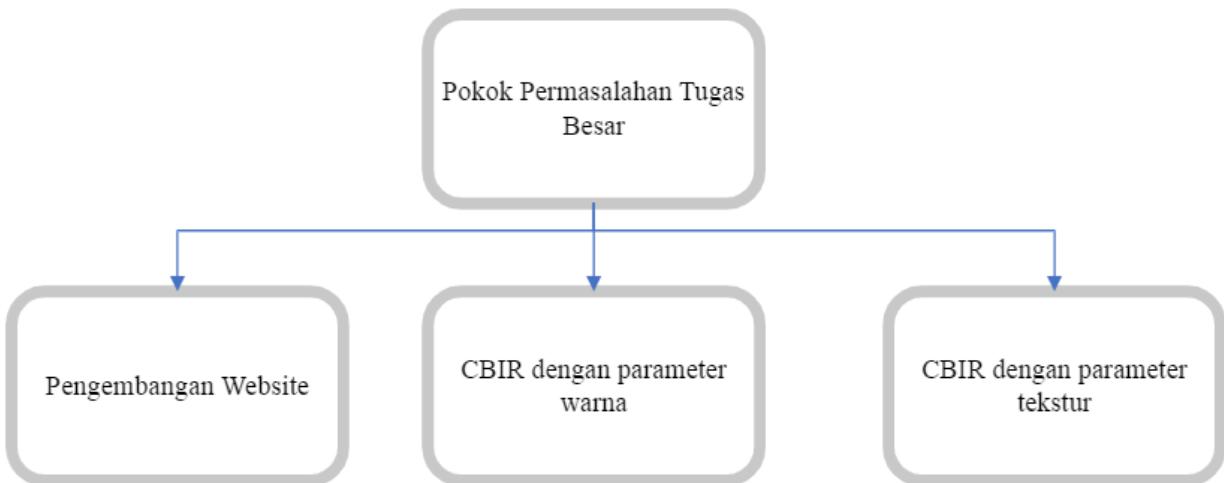
Back-End dari website yang dibuat adalah pengolahan gambar seperti yang disebutkan di atas. Di sisi lain, API mengurus bagaimana data akan dikirim dari *Front-End* ke *Back-End* dan sebaliknya. Dalam API, dikenal istilah RESTful API, yaitu arsitektur untuk merancang aplikasi berbasis jaringan yang memanfaatkan protokol HTTP (seperti GET, POST, PUT, DELETE) untuk melakukan operasi CRUD (Create, Read, Update, Delete).

BAB 3:

Analisis Pemecahan Masalah

3.1 Dekomposisi Masalah

Awalnya, tugas besar ini terlihat rumit karena spesifikasinya tidak tertulis dengan detail dan terkesan banyak yang harus dilakukan. Akan tetapi, setelah mempelajari secara lebih dalam, sebenarnya ada tiga pokok spesifikasi utama yang harus dikerjakan yaitu pengembangan website, CBIR dengan parameter warna, dan CBIR dengan parameter tekstur. Kami membagi tiga hal tersebut di dalam kelompok sedemikian rupa sehingga satu orang memegang sah pokok spesifikasi.



Gambar 3. Dekomposisi masalah

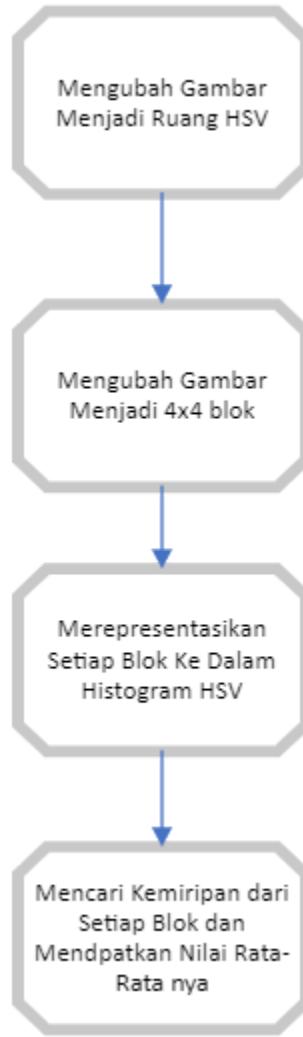
3.2 CBIR dengan Parameter Warna

Tantangan utama dari CBIR dengan parameter warna adalah memecah gambar menjadi blok - blok kecil dan merepresentasikan setiap blok menjadi histogram dalam ruang hsv. Histogram tersebut akan dianggap sebagai vektor dan digunakan untuk mencari persentase kemiripan dari blok-blok gambar yang bersesuaian.

Hal yang pertama dilakukan adalah mengubah gambar dari RGB ke dalam ruang HSV. Kami menggunakan pustaka OpenCV dan NumPy untuk proses tersebut. Proses konversi dilakukan secara manual menyesuaikan rumus yang disediakan oleh spesifikasi. Untuk langkah berikutnya, kami membagi gambar menjadi 4×4 blok. Ukuran tersebut dipilih karena kebanyakan gambar memiliki ukuran kelipatan empat.

Untuk setiap blok dari suatu gambar , kami merepresentasikan informasi hsv dari blok tersebut dalam bentuk histogram. Kami melakukan Kuantisasi pada informasi hsv untuk setiap blok dan menganggap histogram yang terbentuk sebagai vektor. Nilai kemiripan untuk setiap

blok yang bersesuaian dari dua gambar dicari menggunakan rumus *Cosine Similarity* dan di rata-rata. Nilai rata-rata tersebut adalah nilai kemiripan dari dua gambar secara umum.



Gambar 4. Tahapan CBIR dengan parameter warna

3.3 CBIR dengan Parameter Tekstur

Dalam tugas besar ini kami menggunakan python sebagai bahasa utama. Kami menggunakan berbagai pustaka untuk mendukung tugas besar ini. Pustaka yang digunakan adalah openCV untuk membaca gambar, Numpy untuk mempercepat dalam mengolah data, dan math untuk menggunakan rumus matematika.

Hal pertama yang dilakukan dalam CBIR parameter tekstur adalah mengubah gambar dalam ruang vektor BGR menjadi ruang vektor Grayscale. Selanjutnya kami membuat matriks

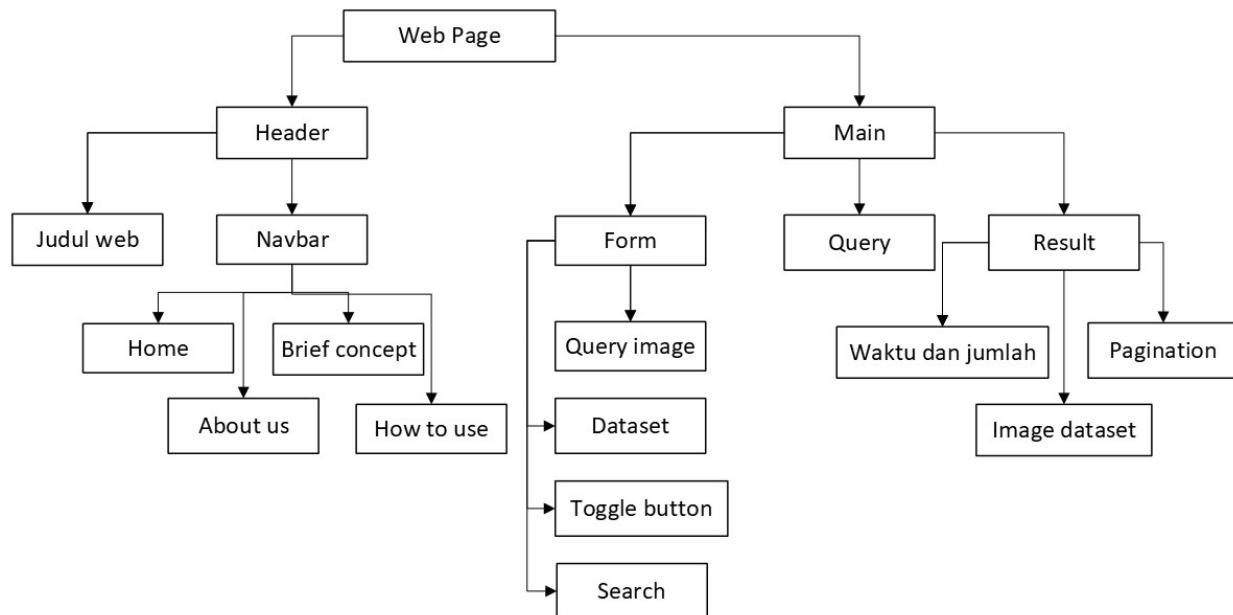
cooccurrence berukuran 256×256 berdasarkan matriks gambar tersebut. Lalu kami mengekstrak komponen-komponen tekstur seperti *contrast*, *entropy*, *homogeneity*, *dissimilarity*, *ASM*, dan *energy*. komponen - komponen tersebut kemudian digabung menjadi satu vektor.

Untuk setiap gambar dibuat satu matriks cooccurrence yang berkoresponden dengan gambar tersebut. Dari matriks tersebut didapatkan vektor yang berisi komponen-komponen tekstur. Vektor tersebut kemudian dibandingkan dengan vektor komponen tekstur dari gambar lain menggunakan cosine similarity. Hasil ini menunjukkan seberapa mirip kedua gambar tersebut. Hal ini dilakukan untuk setiap gambar dalam folder dataset.

3.4 Pengembangan Website

Website akan menampilkan judul website, tombol insert gambar dan dataset, *toggle button* untuk memilih pencarian berdasarkan tekstur atau warna, tombol *Search*, dan halaman tambahan (Konsep singkat search engine yang dibuat, How to Use, dan About us). Setelah menekan tombol *Search*, akan ditampilkan gambar *query* dan kumpulan gambar dari dataset. Gambar dataset ditampilkan dengan paginasi disertai persentase kemiripan untuk setiap gambar dataset dengan gambar *query*.

Untuk mempermudah *layout* pada website, halaman web dibagi menjadi beberapa bagian. Berikut merupakan gambaran *layout* lengkap dari website:



Gambar 5. *Layout* utama website

BAB 4: **Implementasi dan Uji Coba**

4.1 Pseudocode

4.1.a Pseudocode untuk CBIR dengan parameter warna

USE cv2
USE numpy as np
USE math
USE os
USE time

function getCmaxCminDelta (pixel : array[0..2] of integer) —> (float,float,float)
{I.S. : 1 pixel atau 1 array of RGB dari suatu gambar tedefinisi}
{F.S. : Mengembalikan nilai cmax, cmin, delta, untuk perhitungan mencari HSV}

KAMUS LOKAL

ALGORITMA

```
cmax ← np.max(pixel)
cmin ← np.min(pixel)
delta ← cmax - cmin
→ (cmax, cmin, delta)
```

function divide_image_into_blocks(image : array of array of array[0..2] of integer, block_size : integer) → array of array of array[0..2] of integer
{I.S. : matrix RGB dari images terdefinisi }
{F.S. : mengembalikan array of blocks atau array untuk matrix RGB dari setiap block , untuk images yang dibagi menjadi 4 x 4 block}

KAMUS LOKAL

ALGORITMA

```
height, width, _ ← image.shape
delta_height ← height // 4
delta_width ← width // 4
```

blocks ← [image[y:y+delta_height, x:x+delta_width] for y in range(0, height, delta_height) for x in range(0, width, delta_width)]

→ blocks

```
function imageToVector(arrayNormal : array of array of array[0..2] of integer ) -> array of integer
{I.S. : arrayNormal terdefinisi, yaitu array 3D gambar yang sudah dinormalisasi }
{F.S. : Mengembalikan histogram dalam bentuk vektor}
```

KAMUS LOKAL

ALGORITMA

```
array ← [0,0,0,0,0,0,0,0,0,0,0,0]
i traversal[0..arrayNormal.shape[0]]:
    j traversal[0..(arrayNormal.shape[1])]:
        h, s, v ← rgb_to_hsv(arrayNormal[i, j, :])
        if h >= 316 and h <= 360:
            array[0] ← array[0] + 1
        elif h >= 1 and h <= 25:
            array[1] ← array[1] + 1
        elif h >= 26 and h <= 40:
            array[2] ← array[2] + 1
        elif h >= 41 and h <= 120:
            array[3] ← array[3] + 1
        elif h >= 121 and h <= 190:
            array[4] ← array[4] + 1
        elif h >= 191 and h <= 270:
            array[5] ← array[5] + 1
        elif h >= 271 and h <= 295:
            array[6] ← array[6] + 1
        elif h >= 296 and h <= 315:
            array[7] ← array[7] + 1

        if s >= 0 and s < 0.2:
            array[8] ← array[8] + 1
        elif s >= 0.2 and s < 0.7:
            array[9] ← array[9] + 1
        elif s >= 0.7 and s <= 1:
            array[10] ← array[10] + 1

        if v >= 0 and v < 0.2:
            array[11] ← array[11] + 1
        elif v >= 0.2 and v < 0.7:
            array[12] ← array[12] + 1
        elif v >= 0.7 and v <= 1:
            array[13] ← array[13] + 1
end ← time.time()
→ array
```

```
function rgb_to_hsv(pixel : array[0..2] of integer) → (float,float,float)
{I. S. pixel dari gambar terdefinisi}
{F.S. Mengbalikan nilai HSV sesuai dengan aturan yang ada}
```

KAMUS LOKAL

ALGORITMA

```
r ← pixel[0]
g ← pixel[1]
b ← pixel[2]
```

```
cmax, cmin, delta ← getCmaxCminDelta(pixel)
```

```
if delta = 0:
    h ← 0
else:
    if cmax = r:
        h ← 60 * (((g-b)/delta)%6)
    elif cmax = g:
        h ← 60 * (((b-r)/delta)+2)
    elif cmax = b:
        h ← 60 * (((r-g)/delta) +4)
if cmax = 0:
    s ← 0
else:
    s ← delta/cmax
```

```
v ← cmax
```

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

```
function cosineSimilarity(vektor1,vektor2) -> float
{I.S vektor1 dan vektor2 terdefinisi}
{F.S. kemiripan dari dua vektor dapat dihasilkan}
```

KAMUS LOKAL

ALGORITMA

```
function dot_product(vec1, vec2) → float
```

{Menghasilkan pembilang dari rumus}

KAMUS LOKAL

ALGORITMA

->sum(x * y for x, y in zip(vec1, vec2))

function magnitude(vec) → float

{Menghasilkan penyebut dari rumus}

KAMUS LOKAL

ALGORITMA

->math.sqrt(sum(x**2 for x in vec))

dot_prod ← dot_product(vektor1, vektor2)

mag_vec1 ← magnitude(vektor1)

mag_vec2 ← magnitude(vektor2)

if mag_vec1 = 0 or mag_vec2 = 0:

 → 0

similarity ← dot_prod / (mag_vec1 * mag_vec2)

return similarity

function cosineSimilarityForArrayVector(arrayVector1,arrayVector2) → float

{I.S. arrayVector adalah array yang berisi dari kumpulan vector }

{F.S meghasilkan kemiripan rata-rata untuk 16 blok}

KAMUS LOKAL

sum : float

ALGORITMA

for i in range(16):

 sum += cosineSimilarity(arrayVector1[i], arrayVector2[i])

-> sum / 16

function gambar(query, folder) → array of float

{I.S. query adalah path menuju gambar dari query, folder adalah dataset}

{F.S. mengembalikan array atau kumpulan nilai dari kemiripan dari setiap gambar di dataset dengan query}

KAMUS LOKAL

ALGORITMA

```

vectorQuery ← storeQueryVector(query)
arrayOfArrayVectorDataset ← []
for file in folder:
    file ← 'src/static/upload/dir' + file
    vectorForEveryImage ← storeQueryVector(file)
    arrayOfArrayVectorDataset.append(vectorForEveryImage)

arrayCosineSimilarity ← []
for i in arrayOfArrayVectorDataset:
    x ← cosineSimilarityForArrayVector(vectorQuery, i)
    arrayCosineSimilarity.append(x)

→ arrayCosineSimilarity

```

```

function urutGambarWarna (ar_cos : array of float ,ar_file : array of string) -> array of
(float,string)

{I.S. array cosinus dan array nama file terdefinisi}

{F.S. array cosinus dan array nama file yang telah diurut dan difilter}

a <- [(ar_cos[i],ar_file[i]) for i in range(len(ar_cos))]

gambarUrut <- list(filter(lambda x: x[0] >= 60, a))

gambarUrut.sort(reverse = True)

-> gambarUrut

```

4.1.b Pseudocode untuk CBIR dengan parameter tekstur

USE numpy as np

USE math as m

USE cv2

```

function nContrast(mCo : array of array of float) -> float

{I.S. Matriks coocurrence terdefinisi}

{F.S. Mengembalikan nilai contrast}

i, j <- np.indices(mCo.shape)

```

```
contrast <- np.sum(mCo * (i - j)**2)
```

```
-> contrast
```

```
function nHomogeneity(mCo : array of array of float) -> float
```

```
{I.S. Matriks coocurrence terdefinisi}
```

```
{F.S. Mengembalikkan nilai homogeneity}
```

```
i, j <- np.indices(mCo.shape)
```

```
homogeneity <- np.sum(mCo / (1 + (i - j)**2))
```

```
-> homogeneity
```

```
function nEntropy(mCo : array of array of float) -> float
```

```
{I.S. Matriks coocurrence terdefinisi}
```

```
{F.S. Mengembalikkan nilai entropy}
```

```
non_zero_elements <- mCo[mCo != 0]
```

```
entropy <- -np.sum(non_zero_elements * np.log10(non_zero_elements))
```

```
-> entropy
```

```
function nDissimilarity(mCo : array of array of float) -> float
```

```
{I.S. Matriks coocurrence terdefinisi}
```

```
{F.S. Mengembalikkan nilai dissimilarity}
```

```
i, j <- np.indices(mCo.shape)
```

```
dissimilarity <- np.sum(mCo * np.abs(i - j))
```

```
-> dissimilarity
```

```
function nASM(mCo : array of array of float) -> float
{I.S. Matriks coocurrence terdefinisi}
{F.S. Mengembalikan nilai ASM}
    asm <- np.sum(mCo**2)
    -> asm
```

```
function nEnergy(mCo : array of array of float) -> float
{I.S. Matriks coocurrence terdefinisi}
{F.S. Mengembalikan nilai energy}
    energy <- m.sqrt(nASM(mCo))
    -> energy
```

```
function matriksCoOccurance(mGrayImage : array of array of int) -> array of array of float
{I.S. Matriks gambar gray terdefinisi}
{F.S. Mengembalikan matriks coocurrence}
    mCo <- np.zeros((256, 256), dtype<-float)
    for i in range(len(mGrayImage)):
        p <- mGrayImage[i, :-1]
        q <- mGrayImage[i, 1:]
        np.add.at(mCo, (p, q), 1)
    matriksCO <- mCo + mCo.transpose()
    sigma <- 2 * len(mGrayImage) * (len(mGrayImage[0]) - 1)
    matriksCO <- matriksCO / sigma
    -> matriksCO
```

```

function RGBtoGrayscale (gambar : array of array of int) -> array of array of int

{I.S. Matriks gambar terdefinisi}

{F.S. Mengembalikan matriks gambar gray}

    red_channel <- gambar[:, :, 2]

    green_channel <- gambar[:, :, 1]

    blue_channel <- gambar[:, :, 0]

    mGray_float <- 0.299*red_channel + 0.587*green_channel + 0.114*blue_channel +0.5

    mGray <- mGray_float.astype(np.uint8)

-> mGray

```

```

function cosSimilarity (vektorA : array of integer, vektorB : array of integer) -> float

{I.S. Matriks coocurrence terdefinisi}

{F.S. Mengembalikan nilai kontras}

    dotProduct <- 0

    i traversal [0..len(vektorA)]

    dotProduct <- dotProduct + vektorA[i]*vektorB[i]

    jarakA <- 0

    jarakB <- 0

    i traversal [0..len(vektorA)]

    jarakA <- jarakA + m.pow(vektorA[i],2)

    jarakB <- jarakB + m.pow(vektorB[i],2)

    jarakA <- jarakA*jarakB

    jarakA <- dotProduct/m.sqrt(jarakA)

```

```

-> round(jarakA*100 , 2)

function texture (queryImg : string , folder : array of string) -> array of float

{I.S. path queryImg dan path isi folder terdefinisi}

{F.S. Mengembalikan array of cosinus}

array_vektor<- []

for file in folder :

    file <- 'src/static/upload/dir/' + file

    image <- cv2.imread(file)

    image <- RGBtoGrayscale(image)

    image <- matriksCoOccurance(image)

    vektor      <-      [nContrast(image),      nHomogeneity(image),      nEntropy(image),
nDissimilarity(image), nASM(image), nEnergy(image)]

    array_vektor.append(vektor)

array_cos <- []

image <- cv2.imread(queryImg)

image <- RGBtoGrayscale(image)

image <- matriksCoOccurance(image)

vektorq      <-      [nContrast(image),      nHomogeneity(image),      nEntropy(image),
nDissimilarity(image), nASM(image), nEnergy(image)]

for i in array_vektor :

    x <- cosSimilarity(vektorq,i)

array_cos.append(x)

-> array_cos

```

```

function urutGambar (ar_cos : array of float ,ar_file : array of string) -> array of (float,string)

{I.S. array cosinus dan array nama file terdefinisi}

{F.S. array cosinus dan array nama file yang telah diurut dan difilter}

a <- [(ar_cos[i],ar_file[i]) for i in range(len(ar_cos))]

gambarUrut <- list(filter(lambda x: x[0] >= 60, a))

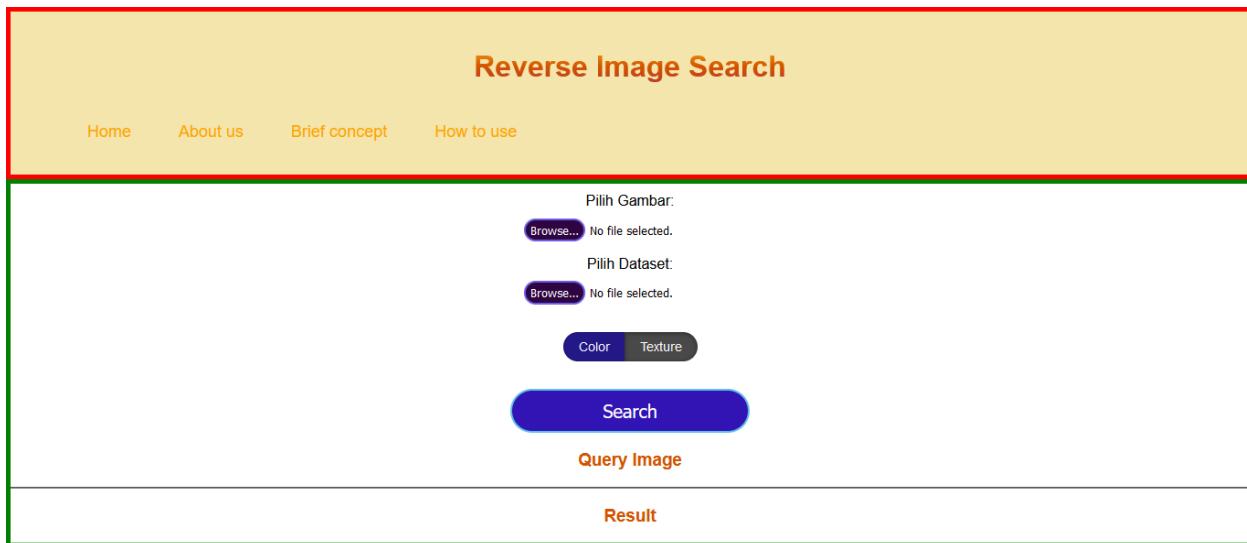
gambarUrut.sort(reverse = True)

-> gambarUrut

```

4.2 Struktur Website

Dari pembagian dasar yang sudah dibuat pada subbab 3.4, berikut merupakan hasil dan desain dari website beserta penjelasan untuk setiap bagiannya.



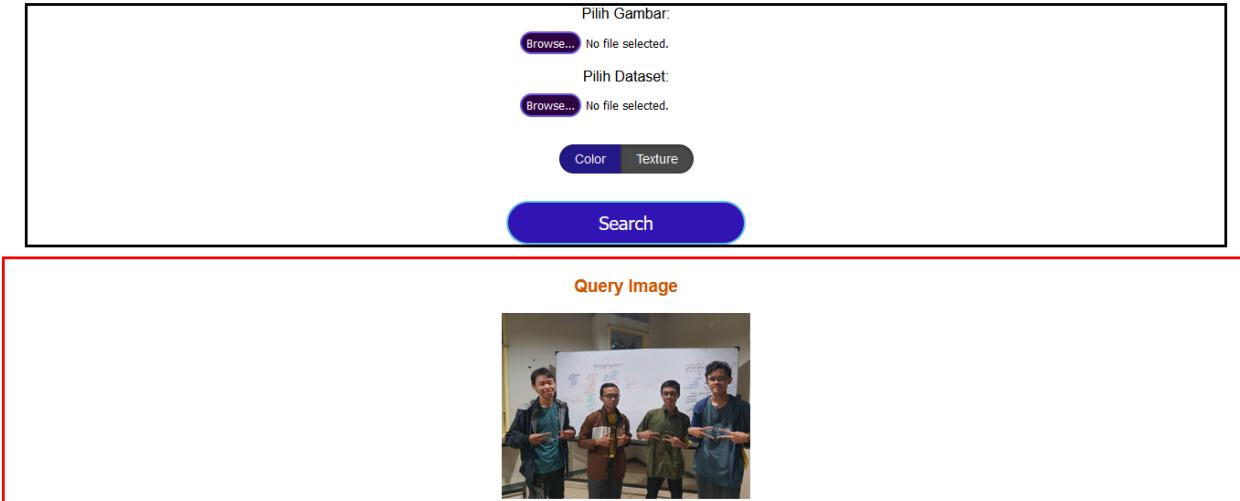
Gambar 6. Layout utama website

Pada *layout* tersebut, di dalam kotak merah adalah elemen *header* dan di dalam kotak hijau adalah elemen *main*. Selanjutnya elemen-elemen website dibagi lagi supaya mempermudah penataan. Berikut adalah pembagian untuk elemen *header*:



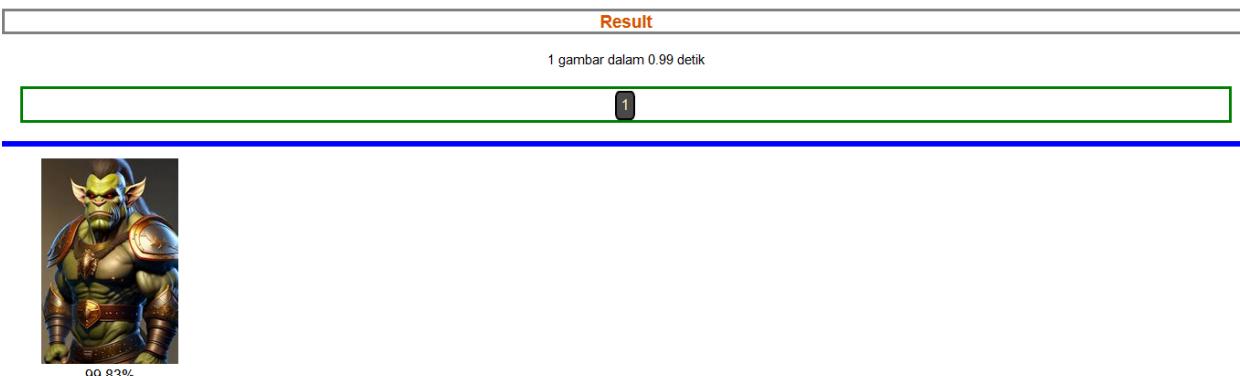
Gambar 7. Layout *header* website

Di dalam kotak hitam adalah judul website dan di dalam kotak abu-abu adalah *navigation bar* untuk halaman tambahan.



Gambar 8. Layout main website bagian *form* dan *query*

Pada gambar tersebut, merupakan bagian awal dari elemen *main* setelah dilakukan pencarian. Di dalam kotak hitam merupakan bagian *form*, yaitu tempat untuk memilih gambar, memilih set data, memilih pencarian warna atau tekstur dan tombol *Search*. Di dalam kotak merak merupakan tampilan dari *query* yang dipilih user. Selanjutnya adalah bagian hasil gambar:



Gambar 9. Layout main website bagian *pagination*, waktu pencarian dan hasil, dan gambar dari *dataset* beserta persentase kemiripan

Pada gambar 9, terlihat elemen waktu eksekusi dan jumlah gambar yang ditampilkan. Serta bagian untuk paginasi pada kotak hijau. Sedangkan dalam kotak biru, diperlihatkan gambar dan persentase kemiripannya.

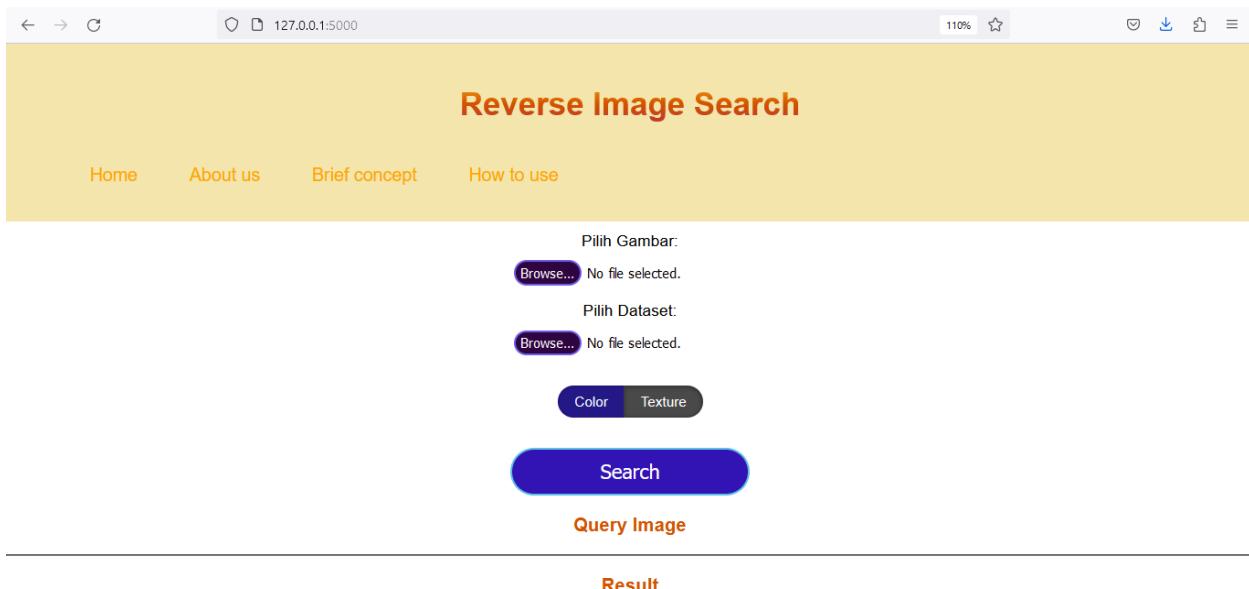
4.3 Tata Cara Penggunaan Program

Untuk menjalankan program, langkah awal yang dilakukan adalah membuka command prompt pada folder yang hasil clone github. Setelah itu, ketikkan command ‘python src/app.py’, maka akan muncul PORT seperti ditunjukan pada gambar di bawah ini.

```
D:\ITB\1.KULIAH\Semester3\Algeo\Tubes_2\Algeo02-22028>python src/app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

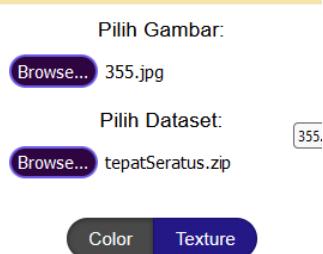
Gambar 10. Tampilan command prompt setelah ‘python src/app.py’

Setelah itu, salin link http dan jalankan pada web browser sehingga website akan terlihat dan bisa digunakan sebagaimana mestinya.



Gambar 11. Tampilan halaman utama website

Website memiliki tampilan awal seperti gambar di atas. Pada gambar tersebut, terdapat *navigation bar* yang telah dijelaskan pada subbab 4.2. Terdapat pilihan menu Home, About us, Brief concept, dan How to use. Di bawahnya, terdapat tombol untuk mengupload *query image* dan set data. *Query image* adalah file gambar dengan ekstensi apa pun, sedangkan file dataset adalah file zip yang hanya berisi gambar-gambar. Selanjutnya adalah *toggle button*, tombol ini berfungsi untuk memilih perbandingan yang dilakukan berdasarkan tekstur atau warna. Terakhir terdapat tombol *Search* untuk memulai proses.



Gambar 12. Contoh setelah upload gambar dan dataset

Pada gambar 12, kita akan mencoba membandingkan sebuah gambar dengan seratus gambar lainnya berdasarkan parameter warna. Setelah proses selesai, *query image* akan dimunculkan di bawah tombol *Search*. Selain itu, akan muncul hasil gambar beserta persentase kemiripannya dan muncul halaman paginasi. Setiap halaman paginasi menampilkan 18 gambar dan jumlah halaman disesuaikan dengan jumlah gambar.

Query Image

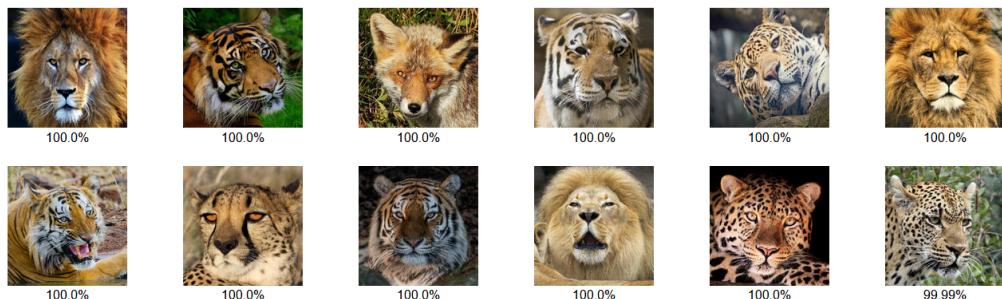


Gambar 13. Tampilan *query image*

Result

100 gambar dalam 5.74 detik

1 2 3 4 5 6

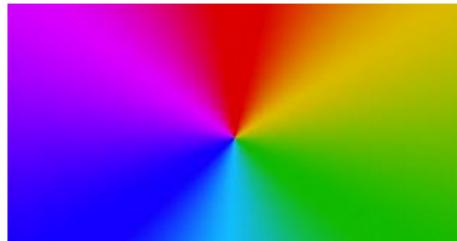


Gambar 14. Tampilan hasil

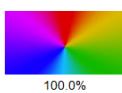
4.4 Hasil Pengujian

1. Perbandingan *query* warna pelangi dengan 50 gambar dataset yang tekstur dan warnanya beragam
 - Perbandingan tekstur
- Gambar *query*:

Query Image



Hasil:



100.0%



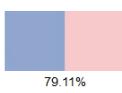
85.44%



83.73%



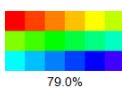
80.61%



79.11%



76.11%



79.0%



75.74%



73.74%



73.72%



73.71%



74.22%



74.18%



74.15%



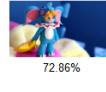
73.26%



73.12%



73.05%



72.86%



72.84%



72.84%



72.38%



72.35%



72.32%



72.3%



72.27%



72.23%



72.08%



72.07%



72.06%



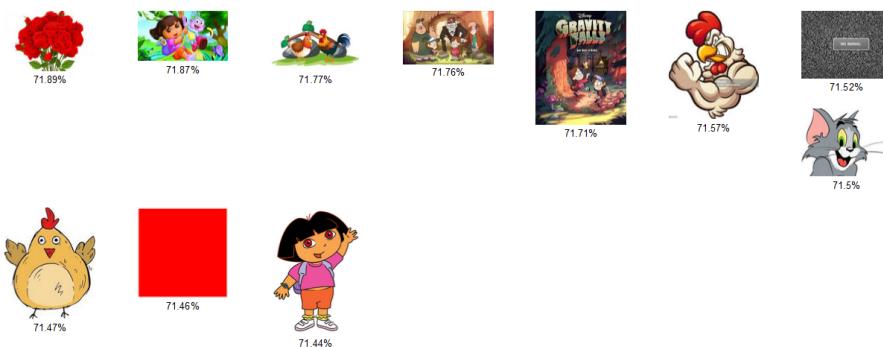
71.97%



71.93%



71.92%

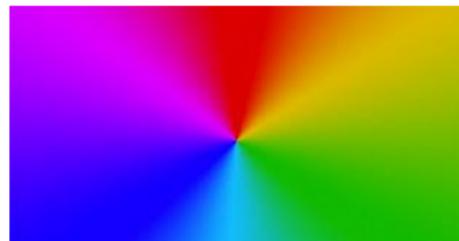


Terdapat 47 gambar dalam 2 detik.

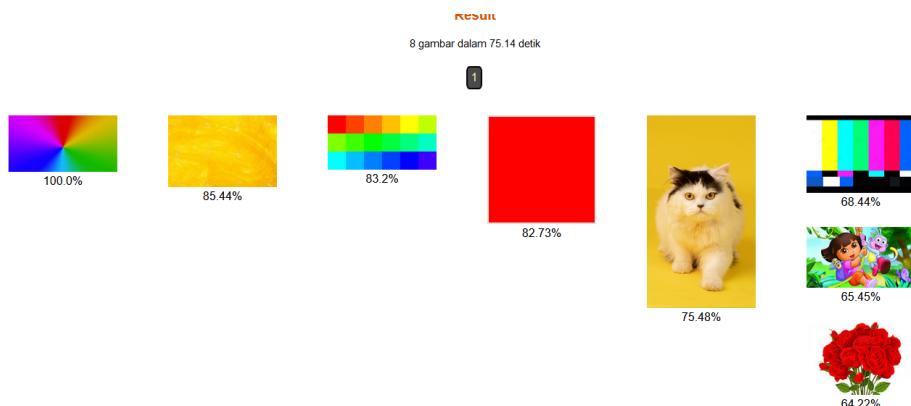
- Perbandingan warna

Gambar *query*:

Query Image



Hasil:



2. Perbandingan warna polos dengan 50 gambar dataset yang tekstur dan warnanya beragam

- Perbandingan warna

Gambar *query*:

Query Image

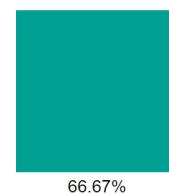
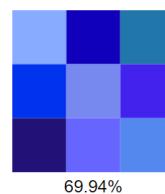
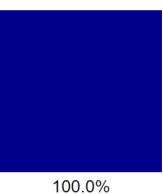


Hasil:

Result

4 gambar dalam 132.82 detik

1



- Perbandingan tekstur

Gambar *query*:

Query Image



Hasil:

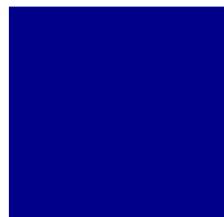
Result

3 gambar dalam 2.63 detik

1



100.0%



100.0%

3. Perbandingan gambar kucing dengan 100 singa

- Perbandingan tekstur

Gambar *query*:

Query Image



Hasil:

100 gambar dalam 4.77 detik

1 2 3 4 5 6



100.0%



100.0%



100.0%



100.0%



100.0%



100.0%



100.0%



100.0%



100.0%



100.0%



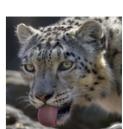
100.0%



100.0%



100.0%



100.0%



100.0%



100.0%



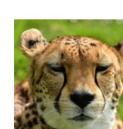
100.0%



100.0%



100.0%



100.0%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.99%



99.98%



99.98%



99.98%



99.98%



99.98%



99.98%



99.98%



99.98%



99.98%



99.98%



99.98%



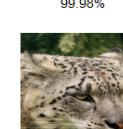
99.98%



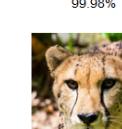
99.98%



99.98%



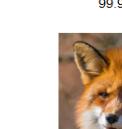
99.98%



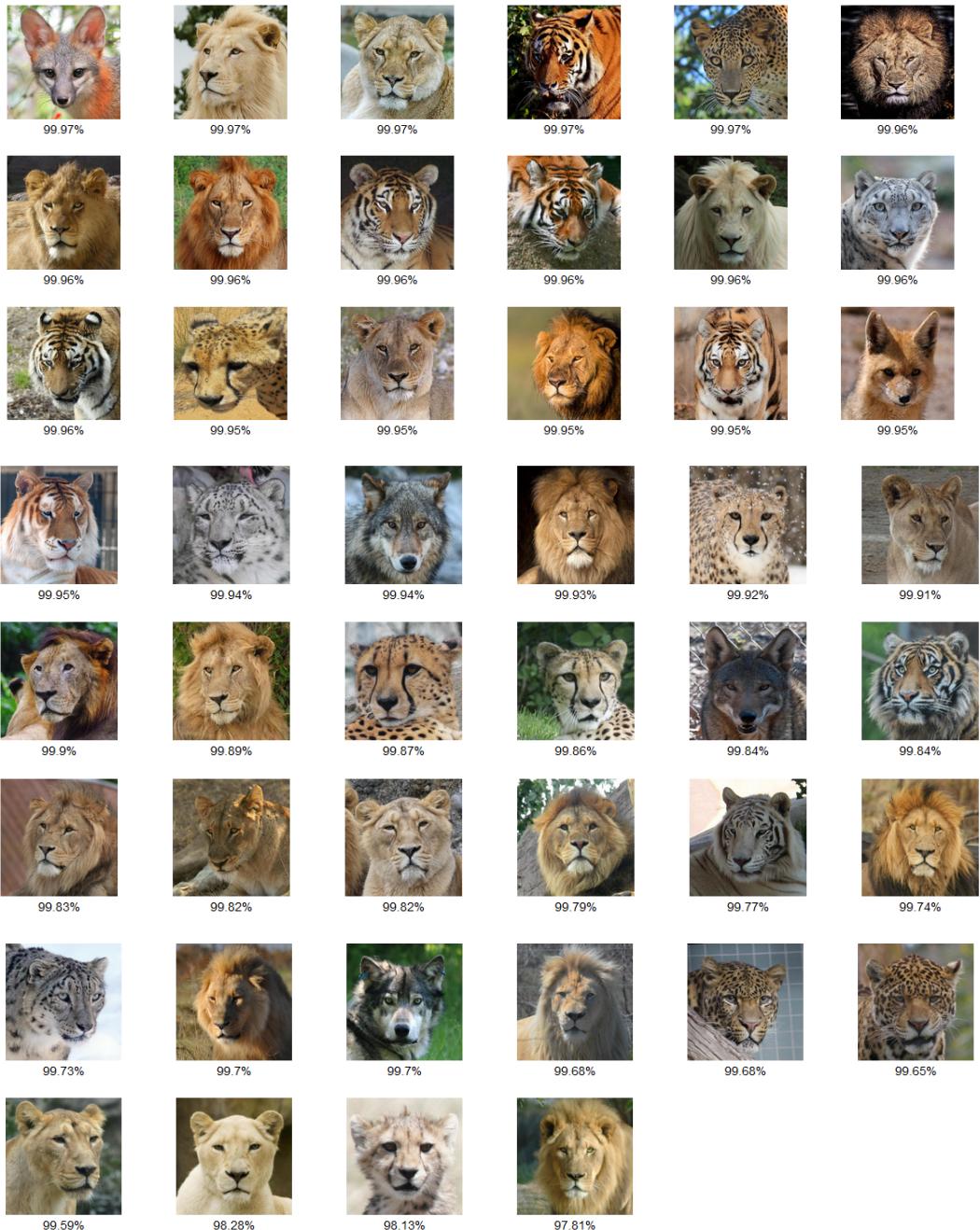
99.97%



99.97%



99.97%



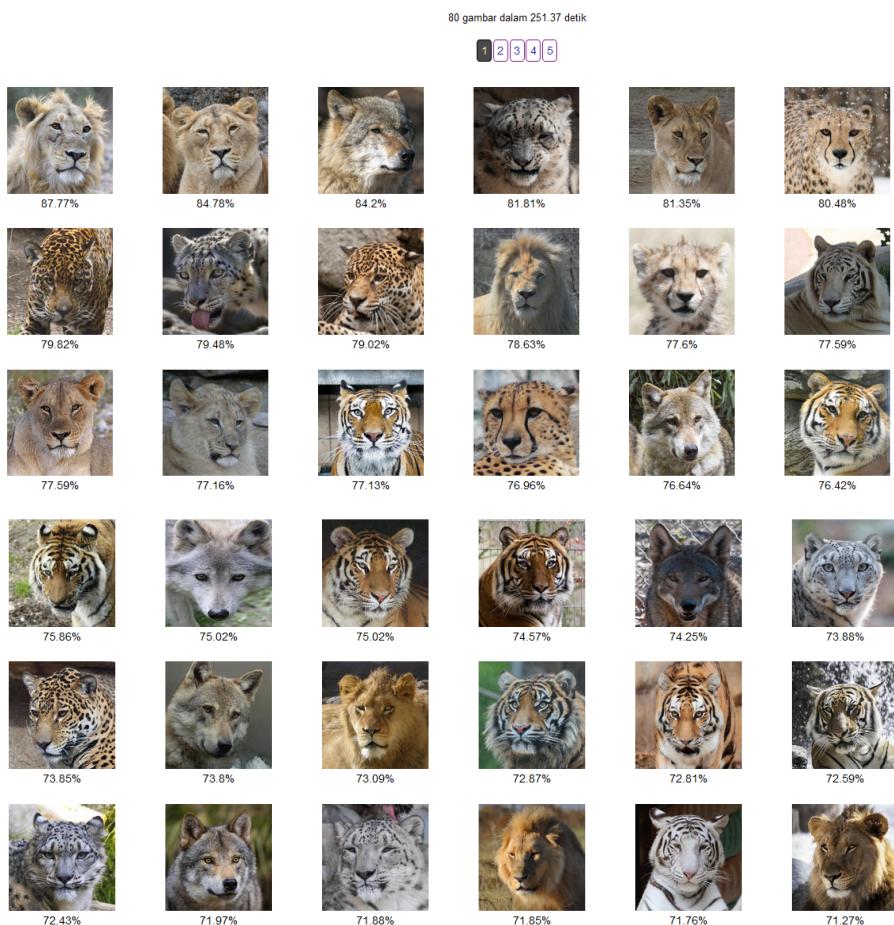
- Perbandingan warna

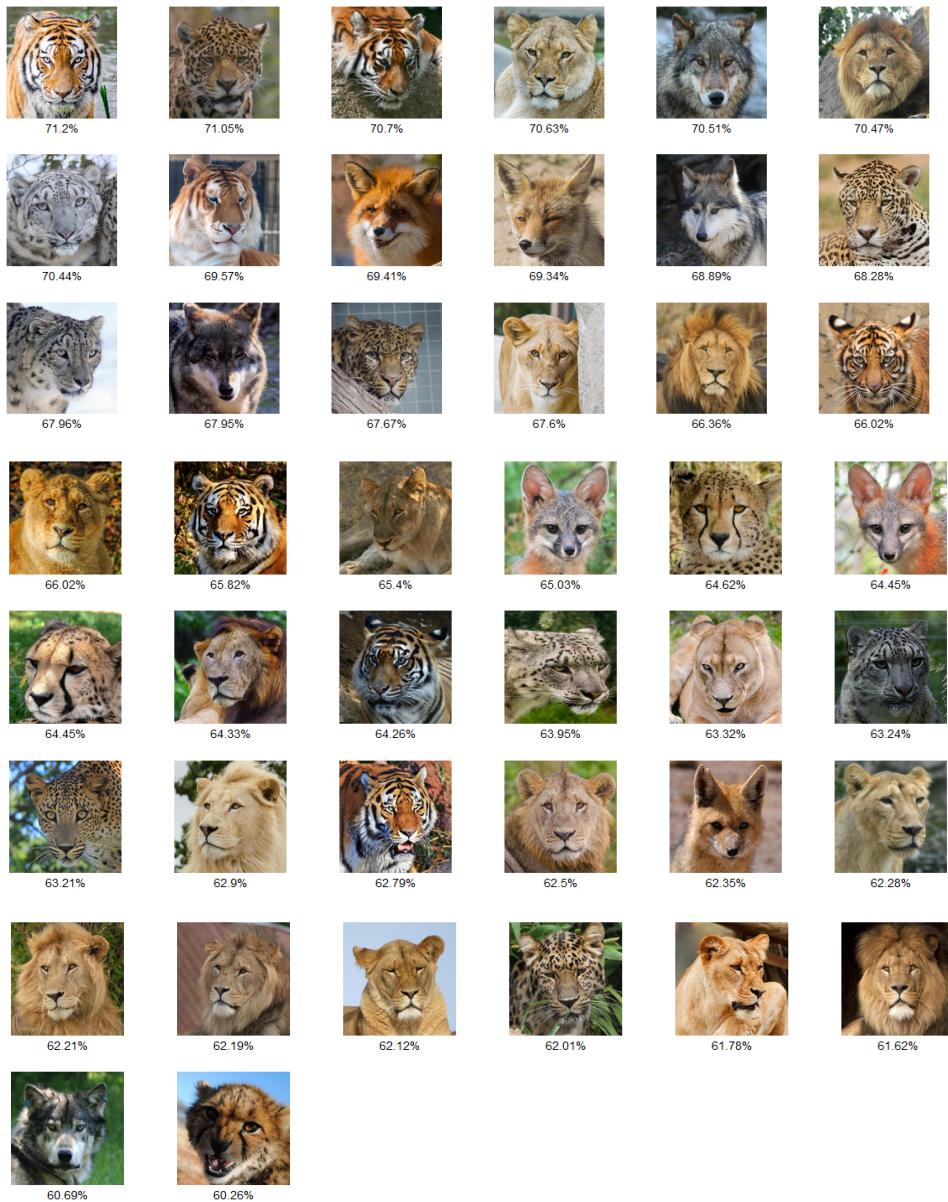
Gambar *query*:

Query Image



Hasil:





4.5 Analisis hasil Pengujian

Dari beberapa pengujian yang dilakukan dapat disimpulkan bahwa pengujian dengan warna lebih baik dibandingkan dengan tekstur. Hal ini disebabkan karena pada pengujian tekstur didapatkan hasil yang seringkali berada di atas 90% untuk gambar. Hasil ini didapatkan karena tekstur dari gambar yang “ramai” memiliki tingkat kecocokan yang tinggi dengan gambar “ramai” lainnya. Ini ditunjukkan pada percobaan 3 yang mendapatkan hasil kecocokan diatas 90% pada seluruh dataset. Namun untuk tekstur gambar yang “rata” didapatkan tingkat kecocokan di bawah 10%. Di sisi lain, pengujian dengan warna lebih membandingkan perbedaan warna di tiap pixelnya sehingga gambar yang “ramai” bisa jadi tidak memiliki kecocokan yang tinggi jika pada tiap pikselnya memiliki warna yang berbeda.

BAB 5: **KESIMPULAN**

1. Kesimpulan

Konsep materi aljabar linear dan geometri sangat berperan besar dalam pengembangan teknologi khususnya *search engine*. Informasi-informasi pada suatu gambar dapat diubah menjadi suatu vektor dan kita melakukan *Image Retrieval* dengan membandingkan vektor *query* dengan vektor dataset. Proses tersebut bisa dilakukan dari dua basis, yaitu basis warna dan basis tekstur, lalu mengintegrasikannya dengan website. Jadi, CBIR dengan parameter warna dan CBIR parameter warna adalah proyek tugas besar yang sangat menggambarkan penerapan teori aljabar linear dan geometri di dunia nyata.

2. Saran

- Panji Sri Kuncara Wisma

Untuk tubes berikutnya mungkin menggunakan GUI saja karena kebanyakan orang-orang belum belajar website sehingga akan kesulitan. Tapi secara umum ini tubes yang menantang dan asik.

- Imam Hanif Mulyarahman

Waktunya lumayan singkat untuk membuat website bagi kami para pemula. Ada baiknya untuk dipertimbangkan waktu penggerjaannya agar lebih banyak yang bisa dieksplorasi

- Bryan Cornelius Lauwrence

Tubes Algeo yang kedua cukup sudah cukup baik, hanya saja waktunya kurang karena tidak sampai 3 minggu.

3. Tanggapan

- Panji Sri Kuncara Wisma

Saya setuju dengan teman saya, ini tubes yang keren dan menarik tapi melelahkan. Tubes ini juga menjadi pengalaman baru bagi saya khususnya memanfaatkan vektor di dunia nyata.

- Imam Hanif Mulyarahman

Tubesnya menarik, tetapi lumayan melelahkan. Dengan adanya tubes ini saya lebih mengenal penggunaan vektor dalam kehidupan nyata.

- Bryan Cornelius Lauwrence

Tubesnya menyenangkan meskipun cukup sulit. Yang penting tubes ini dapat membuat saya berkembang dan belajar banyak hal baru.

4. Refleksi

- Panji Sri Kuncara Wisma

Sepertinya saya harus lebih mengoptimalkan waktu dan tenaga saya untuk mengerjakan tubes tapi disaat yang sama juga harus menjaga kesehatan.

- Imam Hanif Mulyarahman

Dalam pelaksanaan tubes ini, saya masih kerepotan dalam memanfaatkan waktu yang saya punya, sehingga kurang dapat memaksimalkan performa saya. Semoga kedepannya saya bisa lebih disiplin kepada diri saya.

- Bryan Cornelius Lauwrence

Dalam pelaksanaan tubes ini, saya masih kurang mengoptimalkan waktu yang saya miliki apalagi dengan adanya tubes lainnya. Ke depannya, saya harus lebih memanfaatkan waktu yang saya miliki dengan baik.

5. Ruang Perbaikan

- Panji Sri Kuncara Wisma

Untuk efisiensi code yang warna mungkin harusnya bisa saya optimalkan lagi, tapi saya masih kurang pemahaman mengenai kompleksitas dan sebagainya.

- Imam Hanif Mulyarahman

Saya harus belajar memanajemen waktu agar dapat mengerjakan tugas dengan lebih optimal.

- Bryan Cornelius Lauwrence

Saya harus mengoptimalkan penggunaan waktu saya dalam mengerjakan tugas sehingga bisa mencari sumber referensi yang lebih banyak dan lebih baik.

REFERENSI

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

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

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

<https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask>

LINK REPOSITORY GITHUB

<https://github.com/BryanLauw/Algeo02-22028.git>

LINK VIDEO

<https://youtu.be/4HjCBSIhAtQ>