

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



Disusun oleh:
Kelompok 37 - CupcakKe

1. Hugo Sabam Augusto 13522129
2. Nicholas Reymond Sihite 13522144
3. Samy Muhammad Haikal 13522151

BAB I

Deskripsi Masalah

Sebuah gambar (citra, *image*) adalah representasi visual dari sebuah data yang dapat direpresentasikan dalam bentuk digital dengan kombinasi biner. Kombinasi biner tersebut dinyatakan dalam bentuk matriks dengan panjang dan lebar tertentu, atau biasa disebut dengan dimensi gambar. Sebuah elemen dalam matriks tersebut disebut juga dengan istilah pixel, di mana 1 pixel dapat merepresentasikan kombinasi warna merah-hijau-biru (*RGB*), keabu-abuan (*grayscale*), ataupun hitam-putih (biner).

Komputer tidak memiliki mata layaknya manusia. Oleh sebab itu, dibutuhkan perhitungan pixel-per-pixel antara dua gambar agar komputer dapat mengetahui bahwa dua gambar sama atau berbeda. Pada Tugas Besar 2 IF2123 Aljabar Linear dan Geometri ini, mahasiswa diminta untuk membuat sebuah *website* yang berfungsi untuk melakukan komparasi antara 2 gambar dengan sistem temu balik gambar (*content-based image retrieval*) dengan parameter warna dan tekstur. Mahasiswa diminta untuk melakukan perhitungan dengan teknik-teknik yang dipelajari pada kuliah Aljabar Linear dan Geometri untuk menyelesaikan masalah yang diberikan.

BAB II

Landasan Teori

2.1 Dasar Teori

2.1.1 Sistem Temu Balik Gambar Berbasis Konten

Sistem temu balik gambar (*image retrieval system*) adalah sebuah sistem untuk mencari gambar-gambar dari sebuah kumpulan gambar (*image dataset*) yang relevan dengan sebuah gambar lain yang dijadikan referensi (*reference image*).

Salah satu pendekatan yang digunakan dalam sistem temu balik gambar adalah sistem temu balik gambar berbasis konten (*content-based image retrieval*) atau biasa disingkat CBIR. Cara kerja CBIR adalah dengan membandingkan fitur visual (warna, tekstur, dan lainnya) antara gambar pada *dataset* dengan gambar referensi.

2.1.2 CBIR dengan Parameter Warna

Pada CBIR kali ini akan dibandingkan input dari sebuah image dengan image yang dimiliki oleh dataset, hal ini dilakukan dengan cara mengubah image yang berbentuk *RGB* menjadi sebuah metode histogram warna yang lebih umum.

Pada perhitungan histogram, warna global HSV lebih dipilih karena warna tersebut dapat digunakan pada kertas (background berwarna putih) yang lebih umum untuk digunakan. Maka dari itu, perlu dilakukan konversi warna dari *RGB* ke HSV dengan prosedur sebagai berikut.

1. Nilai dari *RGB* harus dinormalisasi dengan mengubah nilai range [0, 255] menjadi [0, 1]

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

2. Mencari Cmax, Cmin, dan Δ

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

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

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

3. Selanjutnya gunakan hasil perhitungan di atas untuk mendapatkan nilai HSV

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

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

$$V = C_{max}$$

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.

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

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

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

Untuk melakukan pencarian histogram, image dibagi menjadi 3×3 blok. Setelah itu lakukanlah perbandingan histogram di tiap blok pada input dan query dengan menggunakan cosine similarity.

2.1.3 CBIR dengan Parameter Tekstur

Berbeda dengan sistem temu balik gambar berdasarkan parameter warna, sistem temu balik gambar berdasarkan parameter tekstur mengabaikan warna gambar sehingga perlu dilakukan konversi dari sistem warna *RGB* ke sistem warna *grayscale*. Selanjutnya, dilakukan pencarian fitur kontras, homogenitas, dan entropi gambar. Ketiga komponen itu kemudian dibentuk menjadi sebuah vektor berikut.

$$< \vec{c}, \vec{h}, \vec{e} >$$

Komponen c mewakili kontras, h mewakili homogenitas, dan e mewakili entropi.

Vektor yang dihasilkan 2 gambar (1 dari *dataset* dan 1 gambar referensi) kemudian akan dibandingkan dengan konsep *cosine similarity of vector*.

2.2 Pengembangan Website

Agar implementasi ini bisa dilihat langsung dengan baik secara kinerjanya, tentu pengembangan website memiliki peran yang penting juga. Pengembangan Website agar keberjalanan CBIR ini dibagi menjadi 2 bagian:

1. Frontend

Untuk Frontend, kami menggunakan HTML, CSS dan juga beberapa komponen Bootstrap. HTML, singkatan dari Hypertext Markup Language, adalah bahasa markup standar untuk membuat struktur dan menandai elemen-elemen di halaman web. CSS (Cascading Style Sheets) digunakan untuk mengatur tata letak dan gaya visual elemen-elemen HTML. Dengan CSS, Anda dapat mengontrol warna, font, margin, padding, dan elemen-elemen desain lainnya pada halaman

web. Bootstrap adalah kerangka kerja front-end yang mempermudah pengembangan web responsif dan menarik secara visual.

2. Backend

Untuk Backend, Kami menggunakan Flask sebagai pusat *logic* dari web app kami. Flask adalah kerangka kerja web Python yang ringan dan fleksibel. Dikembangkan oleh Armin Ronacher, Flask dirancang untuk membuat pengembangan aplikasi web menjadi sederhana dan mudah dipahami. Meskipun Flask termasuk dalam kategori kerangka kerja "micro" karena ukurannya yang kecil, tetapi memiliki kemampuan untuk membangun aplikasi web yang kuat dan efisien.

Routing dalam Flask mengacu pada cara aplikasi merespons permintaan dari klien. Dengan menggunakan routing, Anda dapat menentukan bagaimana URL tertentu harus ditangani oleh aplikasi. Dalam Flask, tampilan adalah fungsi yang menangani permintaan HTTP dan mengembalikan tanggapan.

BAB III

Analisis Pemecahan Masalah

3.1 Langkah Pemecahan Masalah

Dalam menyelesaikan masalah penentuan similaritas dua gambar, kami menggunakan dua metode, yaitu metode CBIR dengan parameter warna dan metode CBIR dengan parameter tekstur.

3.1.1 Pemecahan Masalah CBIR dengan Parameter Warna

Pada metode CBIR dengan parameter warna, proses dibagi menjadi 4 tahapan yaitu

1. Mengubah ruang warna dari *RGB* menjadi *HSV*, tahapan ini dilakukan karena warna *HSV* dapat digunakan pada kertas (background berwarna putih) yang lebih umum untuk digunakan.
2. Membagi gambar menjadi blok 3×3 untuk mempermudah penghitungan histogram
3. Membuat histogram dari tiap tiap blok pada query, nilai-nilai *HSV* perlu dikuantisasi terlebih dahulu untuk membuat bins pada histogram. Pada tahapan ini kami menggunakan 8 bins untuk nilai *H*, dan 3 bins untuk masing-masing nilai *S* dan *V*. Jadi histogram akan menjadi vektor berdimensi 14.
4. Bandingkan nilai histogram pada query dengan histogram pada *database* dengan menggunakan *cosine similarity*. Lalu, hitung rata-rata dari similarity tiap-tiap blok.

Untuk pembagian gambar sendiri ada kasus dimana dimensi gambar tidak habis dibagi 3. Setelah dilakukan sedikit *eksperimen* dengan menggunakan slice tool pada photoshop kami menemukan bahwa penanganan kasus seperti ini pada photoshop terbagi menjadi 3:

1. Dimensi gambar habis dibagi 3, maka photoshop akan membagi pixel secara sama rata.
2. Dimensi gambar dibagi 3 bersisa 1, maka photoshop akan membagi pixel untuk tiap blok sebagai berikut : blok kiri berukuran dimensi//3+1 pixel, blok tengah berukuran dimensi//3 pixel, dan blok kanan berukuran dimensi//3 pixel.
3. Dimensi gambar dibagi 3 bersisa 2, maka photoshop akan membagi pixel untuk tiap blok sebagai berikut : blok kiri berukuran dimensi//3+1 pixel, blok tengah berukuran dimensi//3 pixel, dan blok kanan berukuran dimensi//3+1 pixel.

3.1.2 Pemecahan Masalah CBIR dengan Parameter Tekstur

Pada metode CBIR dengan parameter tekstur, warna dalam skala *RGB* diabaikan sehingga dilakukan konversi dari sistem warna *RGB* ke sistem warna *grayscale* di mana setiap pixel *RGB* dilakukan perhitungan sebagai berikut.

$$gscale_{value} = \lfloor 0.29 \times r_{value} + 0.587 \times g_{value} + 0.114 \times b_{value} \rfloor$$

Perhatikan bahwa ada pembulatan ke bawah oleh fungsi *floor*. Hal ini disebabkan skala *grayscale* berada pada rentang bilangan bulat 0 - 255 sehingga jika hasil perhitungan tidak bulat, harus dikenakan fungsi *floor*.

Setelah mendapatkan nilai *grayscale*, dibentuklah sebuah *co-occurrence matrix* (lebih lanjut dibahas pada 3.2.2) yang isinya akan dijadikan penentu tiga buah fitur tekstur, yaitu kontras, homogenitas, dan entropi.

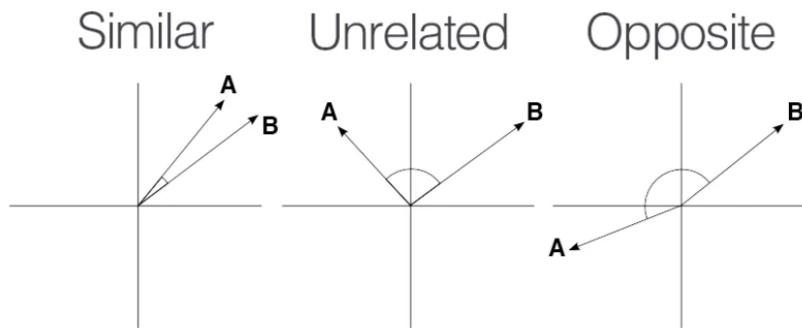
Ketiga buah fitur tersebut dapat dianggap sebagai komponen pembentuk vektor sehingga akan ada dua buah vektor dari dua gambar yang berbeda. Kemiripan dua gambar akan diwakili oleh kemiripan dua vektor perwakilan masing-masing gambar melalui pencarian *cosine similarity* kedua vektor tersebut. Semakin besar nilai *cosine similarity*-nya, semakin mirip pula tekstur kedua gambar tersebut.

3.2 Proses Pemetaan Masalah Menjadi Elemen Aljabar Geometri

Seperti yang disebutkan pada Bab I Deskripsi Masalah, sebuah gambar digital adalah representasi dari sebuah matriks yang setiap elemennya merupakan sebuah pixel yang berisi nilai tertentu yang kemudian dikonversi ke sistem warna yang berbeda-beda. Matriks ini yang akan menjadi dasar dalam perhitungan similaritas dua buah gambar.

3.2.1 Pemetaan Aljabar Geometri CBIR dengan Parameter Warna

Pada proses membandingkan histogram kita menggunakan sebuah konsep dari aljabar linear, yaitu cosine similarity. Secara singkat cosine similarity adalah algoritma yang akan menentukan kemiripan dua buah vektor berdasarkan nilai cosinus sudut di antara kedua vektor tersebut. Semakin kecil sudutnya maka nilai cos akan semakin mendekati 1. Semakin besar sudutnya maka nilai cosinus akan semakin mendekati 0. Ilustrasi pada ruang berdimensi 2 adalah sebagai berikut:



Kita dapat menerapkan konsep ini pada histogram sebuah gambar, karena pada dasarnya histogram yang dibuat adalah vektor pada ruang berdimensi 14. Sehingga kita dapat mengukur kemiripan dari dua buah gambar berdasarkan nilai cosine similarity dari histogramnya.

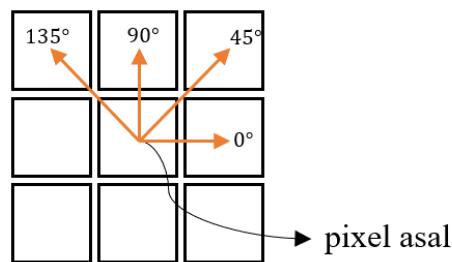
3.2.2 Pemetaan Aljabar Geometri CBIR dengan Parameter Tekstur

Pada bagian 3.1.2, disebut bahwa nilai *grayscale* akan digunakan untuk membentuk sebuah *co-occurrence matrix*. *Co-occurrence matrix* adalah sebuah matriks yang pada dasarnya memuat komponen-komponen tekstur sebuah gambar. Karena nilai *grayscale* sebuah gambar (misalkan matriks *grayscale* bernama G) berada pada rentang 0 - 255, *co-occurrence matrix* (misalkan C) yang

bersesuaian akan memiliki dimensi 256×256 . Nilai dari setiap elemen *co-occurrence matrix* pada awalnya akan di-set menjadi 0. Pembuatan *co-occurrence matrix* dari nilai *grayscale* sebuah gambar dapat dilakukan dengan cara sebagai berikut.

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

Nilai dari Δx dan Δy ditentukan berdasarkan sudut yang digunakan. Beberapa sudut yang umumnya digunakan adalah 0° , 45° , 90° , atau 135° . Berikut adalah visualisasi pencarian *co-occurrence matrix* dengan keempat sudut tersebut.



Sebagai contoh, jika pixel asal berada pada koordinat (i, j) pada matriks gambar *grayscale*: untuk sudut 0° , elemen *co-occurrence matrix* pada baris $G(i, j)$ dan kolom $G(i + 1, j)$ akan bertambah 1; untuk sudut 45° , elemen *co-occurrence matrix* pada baris $G(i, j)$ dan kolom $G(i + 1, j - 1)$ akan bertambah 1; untuk sudut 90° , elemen *co-occurrence matrix* pada baris $G(i, j)$ dan kolom $G(i, j - 1)$ akan bertambah 1; dan untuk sudut 135° , elemen *co-occurrence matrix* pada baris $G(i, j)$ dan kolom $G(i - 1, j - 1)$ akan bertambah 1. Pada penggerjaan Tugas Besar 2 ini, sudut yang kami gunakan adalah sudut 0° . Kami memilih sudut ini karena kami juga mengerjakan salah satu bonus, yaitu fitur kamera. Kamera laptop pada umumnya berorientasi *landscape* (horizontal) sehingga sudut yang paling tepat digunakan dalam konteks ini adalah sudut 0° .

Setelah mendapatkan *co-occurrence matrix*, akan dilakukan normalisasi dengan cara menjumlahkannya dengan *transpose*-nya, lalu membagi setiap elemen dengan jumlah seluruh elemennya. Perhitungan lengkapnya dapat dilihat di bawah ini.

$$sum = \sum_{i=1}^{255} \sum_{j=1}^{255} C(i,j) + C^T(i,j)$$

$$Norm(C) = \frac{1}{sum} \times (C + C^T)$$

Matriks hasil normalisasi ini lalu digunakan untuk mencari nilai-nilai komponen kontras, homogenitas, dan entropi gambar (*dataset* dan referensi). Perhitungan rincinya dapat dilihat sebagai berikut.

1. Kontras

$$\text{kontras} = \sum_i^{255} \sum_j^{255} Norm(C)_{i,j} \times (i - j)^2$$

2. Homogenitas

$$\text{homogenitas} = \sum_i^{255} \sum_j^{255} Norm(C)_{i,j} \times \frac{1}{1 + (i - j)^2}$$

3. Entropi

$$\text{entropi} = -1 \times \left[\sum_i^{255} \sum_j^{255} Norm(C)_{i,j} \times \log(Norm(C)_{i,j}) \right]$$

Khusus untuk bagian entropi, jika $Norm(C)_{i,j}$ bernilai 0, akan dilakukan perhitungan limit sebab nilai $\log(0)$ tidak ada. Berikut adalah nilai limitnya.

$$\lim_{x \rightarrow 0} x \times \log(x) = 0$$

Setelah mendapatkan nilai ketiga komponen tersebut, langkah selanjutnya adalah membentuk vektor 3D seperti disebutkan pada bagian 2.1.3.

$$< \vec{c}, \vec{h}, \vec{e} >$$

Jika awalnya terdapat sebuah gambar dari *dataset*, misal I_1 , dan sebuah gambar referensi, misal I_2 , sekarang akan terbentuk dua vektor yang mewakili tekstur I_1 dan I_2 , misal \vec{v}_1 dan \vec{v}_2 .

Langkah terakhir yang perlu dilakukan untuk menentukan similaritas dua gambar berdasarkan tekstur adalah mencari similaritas vektor yang berisi komponen tekturnya. Kemiripan dua buah vektor dapat ditentukan dengan menggunakan konsep *cosine similarity*. Perhitungan *cosine similarity* adalah sebagai berikut.

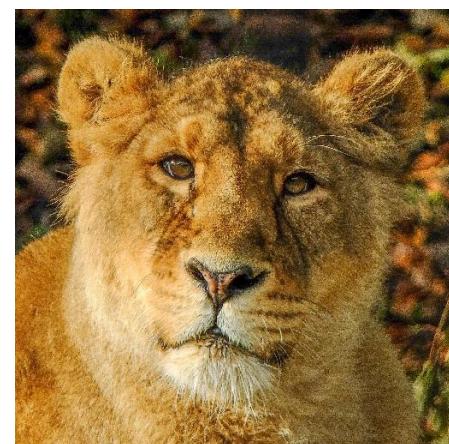
$$\cos(\theta) = \frac{\vec{v}_1 \cdot \vec{v}_2}{\|\vec{v}_1\| \times \|\vec{v}_2\|}$$

Nilai dari $\cos(\theta)$ adalah similaritas dua gambar berdasarkan tekstur. Untuk mengubahnya ke dalam format persen (%), $\cos(\theta)$ cukup dikali dengan 100%.

3.3 Contoh Ilustrasi Kasus dan Penyelesaian

3.3.1 Contoh Kasus dan Penyelesaian CBIR dengan Parameter Warna

Misalkan terdapat dua gambar *RGB* sebagai berikut



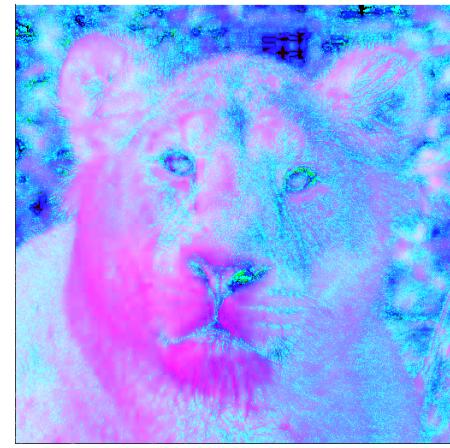
Gambar 1

Gambar 2

Kedua gambar ini akan diubah ke ruang warna HSV terlebih dahulu:

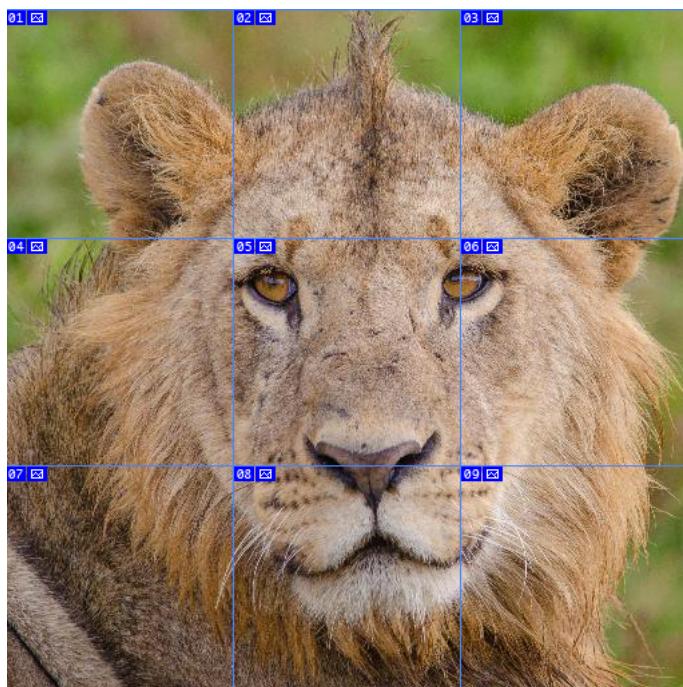


Gambar 1



Gambar 2

Lalu gambar dibagi menjadi blok 3×3 dan akan dicari histogram dan cosine similarity nya:



Didapatkan bahwa rata-rata cosine similarity histogram dari tiap blok pada gambar adalah 0.7886, untuk memudahkan perbandingan kita akan mengubah formatnya menjadi persentase, jadi persentase kemiripan kedua gambar berdasarkan warna adalah 78.86%

3.3.2 Contoh Kasus dan Penyelesaian CBIR dengan Parameter Tekstur

Misalkan terdapat dua gambar *RGB* (I_1 dan I_2) berdimensi 3×3 sebagai berikut:

Matriks *RGB* yang menggambarkan I_1 :

$$\begin{bmatrix} (21, 98, 47) & (100, 32, 66) & (9, 7, 8) \\ (220, 45, 33) & (10, 100, 64) & (23, 67, 80) \\ (4, 3, 2) & (210, 199, 200) & (27, 9, 150) \end{bmatrix}$$

Matriks *RGB* yang menggambarkan I_2 :

$$\begin{bmatrix} (1, 9, 247) & (10, 2, 66) & (98, 12, 7) \\ (20, 34, 71) & (10, 12, 18) & (100, 12, 1) \\ (33, 44, 55) & (10, 50, 9) & (5, 1, 84) \end{bmatrix}$$

Matriks gambar yang awalnya memiliki format warna *RGB* diubah menjadi matriks gambar yang memiliki format warna *grayscale*:

Matriks *grayscale* yang menggambarkan I_1 :

$$\begin{bmatrix} 68 & 55 & 7 \\ 93 & 68 & 55 \\ 3 & 200 & 30 \end{bmatrix}$$

Matriks *grayscale* yang menggambarkan I_2 :

$$\begin{bmatrix} 33 & 11 & 36 \\ 33 & 11 & 36 \\ 41 & 33 & 11 \end{bmatrix}$$

Maka, akan terbentuk *co-occurrence matrix* yang mewakili kedua gambar sebagai berikut (untuk efisiensi, hanya akan ditampilkan elemen *co-occurrence matrix* yang bernilai lebih dari 0):

Elemen *co-occurrence matrix* yang menggambarkan I_1 :

$$\begin{aligned} C_1(3,200) &= 1 \\ C_1(55,7) &= 1 \\ C_1(68,55) &= 2 \\ C_1(93,137) &= 1 \\ C_1(200,30) &= 1 \end{aligned}$$

Elemen *co-occurrence* yang menggambarkan I_2 :

$$\begin{aligned} C_2(11,36) &= 2 \\ C_2(33,11) &= 3 \\ C_2(41,33) &= 1 \end{aligned}$$

Selanjutnya, akan dihitung matriks normalisasi keduanya dengan hasil sebagai berikut:

Normalisasi *co-occurrence matrix* yang menggambarkan I_1 :

$$\begin{aligned} Norm(C_1)_{3,200} &= 0.08333 \\ Norm(C_1)_{7,55} &= 0.08333 \\ Norm(C_1)_{30,200} &= 0.08333 \\ Norm(C_1)_{55,7} &= 0.08333 \\ Norm(C_1)_{55,68} &= 0.16667 \\ Norm(C_1)_{68,55} &= 0.16667 \\ Norm(C_1)_{93,137} &= 0.08333 \\ Norm(C_1)_{137,93} &= 0.08333 \\ Norm(C_1)_{200,3} &= 0.08333 \\ Norm(C_1)_{200,30} &= 0.08333 \end{aligned}$$

Normalisasi *co-occurrence* yang menggambarkan I_2 :

$$\begin{aligned} Norm(C_2)_{11,33} &= 0.25 \\ Norm(C_2)_{11,36} &= 0.16667 \\ Norm(C_2)_{33,11} &= 0.25 \\ Norm(C_2)_{33,41} &= 0.08333 \\ Norm(C_2)_{36,11} &= 0.16667 \\ Norm(C_2)_{41,33} &= 0.08333 \end{aligned}$$

Setelah dinormalisasi, langkah selanjutnya adalah mencari nilai kontras, homogenitas, dan entropi I_1 dan I_2 lalu mengubah nilai-nilainya menjadi vektor:

Vektor kontras, homogenitas, dan entropi I_1 :

$$\vec{v}_1 = < 12047.3548, 0.002129228322, 0.97882300929 >$$

Vektor kontras, homogenitas, dan entropi I_2 :

$$\vec{v_2} = \langle 461.00374, 0.0041274198478, 0.74027527984 \rangle$$

Langkah terakhir, cari similaritas kedua vektor dengan menggunakan konsep *cosine similarity* sehingga didapat hasil:

$$\begin{aligned}\cos(\theta) &= \frac{\langle 12047.3548, 0.002129228322, 0.97882300929 \rangle \cdot \langle 461.00374, 0.0041274198478, 0.74027527984 \rangle}{\|\langle 12047.3548, 0.002129228322, 0.97882300929 \rangle\| \times \|\langle 461.00374, 0.0041274198478, 0.74027527984 \rangle\|}} \\ &= 0.9999988378493234 \\ &\approx 99.9998837\%\end{aligned}$$

BAB IV

Implementasi dan Uji Coba

4.1 Implementasi Program Utama

4.1.1 Implementasi CBIR dengan Parameter Tekstur

```
Program CBIRtexture
{Library untuk mencari similaritas dua gambar berdasarkan kemiripan tekstur (CBIR by
texture)}

Use PIL
Use math

KAMUS UMUM
function open (img_name: string) → matrix
{Fungsi dari library PIL untuk mengambil Image dari file Bernama img_name}
procedure close (input img: Image)
{Prosedur dari library PIL untuk menutup Image img}
function size (image: Image, x, y: integer) → (integer, integer)
{Fungsi dari library PIL untuk mengekstrak tupel dimensi gambar Image}
function getpixel (image: Image) → (integer, integer)
{Fungsi dari library PIL untuk mengekstrak tupel nilai RGB pada koordinat x, y}
function len (arr: array) → integer
{Fungsi bawaan python untuk mengekstrak Panjang sebuah array}
function pow (base, pow: float) → float
{Fungsi python untuk mengembalikan base pangkat pow}
function log10 (x: float) → float
{Fungsi dari library math untuk mengembalikan nilai logaritma basis 10 dari x}
function sqrt (x: float) → float
{Fungsi dari library math untuk mengembalikan nilai akar x}
function rgbToGrayscale (RGBImage: Image) → matrix
{Fungsi mengubah nilai RGB sebuah gambar menjadi nilai Grayscale}
function coOccMtx (gScaleMtx: matrix, Angle: integer) → matrix
{Fungsi membuat co-occurrence matrix dari gScaleMtx dengan sudut Angle}
function transpose (Mtx: matrix) → matrix
{Fungsi men-transpose matrix Mtx}
function normalize (Mtx: matrix) → matrix
{Fungsi normalisasi sebuah matrix}
function contrast (coOccMtx: matrix) → float
{Fungsi mengembalikan nilai kontras matrix coOccMtx}
function homogeneity (coOccMtx: matrix) → float
{Fungsi mengembalikan nilai homogenitas matrix coOccMtx}
function entropy (coOccMtx: matrix) → float
{Fungsi mengembalikan nilai entropi matrix coOccMtx}
function imgToVector (img_name: string) → vector
{Fungsi mengubah gambar Bernama img_name menjadi sebuah vektor}
function cosSimilarity(v1, v2: vector) → float
{Fungsi menghasilkan similaritas 2 vektor dengan metode cosine similarity}

REALISASI

function rgbToGrayscale (RGBImage: Image) → matrix

KAMUS LOKAL
width, height, i, j, r, g, b, gscalevalue: integer
matrix: matrix [0..height-1][0..width-1] of integer

ALGORITMA
```

```

(width, height) ← size(RGBImage)

i traversal [0..height-1]

    j traversal [0..width-1]

        (r, g, b) ← getpixel(RGBImage, j, i)
        gscalevalue ← 0.29 * g + 0.587 * g + 0.114 * b
        matrix[i][j] ← gscaledvalue

→ matrix

function coOccMtx (gScaleMtx: matrix, Angle: integer) → matrix

KAMUS LOKAL
coOcc: matrix [0..255][0..255] of integer
height, width, i, j: integer

ALGORITMA

i traversal [0..height-1]

    j traversal [0..width-1]

        coOcc[gScaleMtx[i][j]][gScaleMtx[i][j+1]] ← 0

height ← len(gScaleMtx)
width ← len(gScaleMtx[0])

if (Angle = 0) then

    i traversal [0..height-1]

        j traversal [0..width-1]

            coOcc[gScaleMtx[i][j]][gScaleMtx[i][j+1]] ←
coOcc[gScaleMtx[i][j]][gScaleMtx[i][j+1]] + 1

else if (Angle = 45) then

    i traversal [0..height-1]

        j traversal [0..width-1]

            coOcc[gScaleMtx[i][j]][gScaleMtx[i-1][j+1]] ←
coOcc[gScaleMtx[i][j]][gScaleMtx[i-1][j+1]] + 1

else if (Angle = 90) then

    i traversal [0..height-1]

        j traversal [0..width-1]

            coOcc[gScaleMtx[i][j]][gScaleMtx[i-1][j]] ←
coOcc[gScaleMtx[i][j]][gScaleMtx[i-1][j]] + 1

else if (Angle = 135) then

```

```

    i traversal [0..height-1]

    j traversal [0..width-1]

        coOcc[gScaleMtx[i][j]][gScaleMtx[i-1][j-1]] ←
        coOcc[gScaleMtx[i][j]][gScaleMtx[i-1][j-1]] + 1

```

→ coOcc

function transpose (Mtx: matrix) → matrix

KAMUS LOKAL

```

h, w, i, j: integer
MtxT: matrix [0..h-1][0..w-1] of integer

```

ALGORITMA

```

h ← len(Mtx)
w ← len(Mtx[0])

```

i traversal [0..h-1]

 j traversal [0..w-1]

 MtxT[i][j] ← Mtx[j][i]

→ MtxT

function normalize (Mtx: matrix) → matrix

KAMUS LOKAL

```

h, w, sum, i, j: integer
MtxT, MtxSym: matrix [0..h-1][0..w-1] of integer

```

ALGORITMA

```

MtxT ← transpose(Mtx)
h ← len(Mtx)
w ← len(Mtx[0])
sum ← 0

```

i traversal [0..h-1]

 j traversal [0..w-1]

 MtxSym[i][j] ← Mtx[i][j] + MtxT[i][j]
 sum ← sum + MtxSym[i][j]

i traversal [0..h-1]

 j traversal [0..w-1]

 MtxSym[i][j] ← Mtx[i][j] / sum

→ MtxSym

function contrast (coOccMtx: matrix) → float

KAMUS LOKAL

```
i, j: integer
con: float
```

ALGORITMA

```
con ← 0
```

```
i traversal [0..255]
```

```
    j traversal [0..255]
```

```
        con ← con + coOccMtx[i][j] * pow((i - j), 2)
```

```
→ con
```

function homogeneity (coOccMtx: matrix) → float

KAMUS LOKAL

```
i, j: integer
hom: float
```

ALGORITMA

```
hom ← 0
```

```
i traversal [0..255]
```

```
    j traversal [0..255]
```

```
        hom ← hom + coOccMtx[i][j] / (1 + pow((i - j), 2))
```

```
→ hom
```

function entropy (coOccMtx: matrix) → float

KAMUS LOKAL

```
i, j: integer
ent: float
```

ALGORITMA

```
ent ← 0
```

```
i traversal [0..255]
```

```
    j traversal [0..255]
```

```
        if (coOccMtx[i][j] == 0) then
```

```
            ent ← ent + coOccMtx[i][j] * log10(coOccMtx[i][j])
```

```
→ ent
```

function imgToVector (img_name: string) → vector

KAMUS LOKAL

```
img: Image
```

```
gscale_img: matrix [0..height-1][width-1] of integer
coOcc_img: matrix [0..255][0..255] of integer
cont, hom, ent: float
```

ALGORITMA

```
img ← open(img_name)
gscale_img ← rgbToGrayscale(img)
coOcc_img ← coOccMtx(gscale_img, 0)
coOcc_img ← normalize(coOcc_img)
cont ← contrast(coOcc_img)
hom ← homogeneity(coOcc_img)
ent ← entropy(coOcc_img)
close(img)

→ [cont, hom, ent]
```

function cosSimilarity(v1, v2: vector) → float

KAMUS LOKAL

```
numerator, denominator, denominatorv1, denominatorv2: float
i: integer
```

ALGORITMA

```
numerator ← 0

i traversal (len(v1))

    numerator ← numerator + v1[i] * v2[i]

denominatorv1 ← 0

i traversal (len(v1))

    denominatorv1 ← denominatorv1 + pow(v1[i], 2)

denominatorv1 ← sqrt(denominatorv1)

denominatorv2 ← 0

i traversal (len(v2))

    denominatorv2 ← denominatorv2 + pow(v2[i], 2)

denominatorv2 ← sqrt(denominatorv2)

denominator ← denominatorv1 * denominatorv2

→ (numerator / denominator)
```

4.1.2 Implementasi CBIR dengan Parameter Warna

```
Program color
{Library untuk mencari similaritas dua gambar berdasarkan kemiripan warna(CBIR by warna) }

Use cv2
Use numpy

KAMUS UMUM
procedure sort(*, key: None=..., reverse: bool=...)
{Prosedur bawaan python untuk mengurutkan sebuah array}
function dot (vector1, vector2 : array of integer) → integer
{fungsi dari library numpy untuk menghitung nilai dot product dari dua buah vektor}
function transpose (m: matrix) → matrix
{fungsi dari library numpy untuk mengembalikan transpose suatu matrix}
procedure append (input object: Object)
{Prosedur bawaan python untuk menambahkan suatu objek pada array}
function round (number: float, dec_count: integer) → float
{Fungsi bawaan dari python untuk membulatkan angka desimal ke 2 angka di depan koma}
function imread (image: string) → matrix
{Fungsi dari library cv2 untuk membaca file gambar menjadi suatu matrix}
function concatenate (a1,a2,a3... : array of integer) → array [...] of integer
{fungsi dari library numpy untuk menyambung serangkaian array menjadi satu array}
function len (arr: array) → integer
{Fungsi bawaan python untuk mengekstrak Panjang sebuah array}
function norm (vector: array of integer) → float
{Fungsi dari library numpy untuk menghitung nilai normalisasi dari sebuah vektor}
function mean (m: matrix) → float
{fungsi dari library numpy untuk mencari nilai rata-rata dari sebuah matrix}
function maximum(m:matrix, axis:integer) → matrix
{Fungsi dari library numpy untuk mengembalikan nilai maksimum dari matrix dengan axis tertentu} function minimum(m:matrix, axis:integer) → matrix
{Fungsi dari library numpy untuk mengembalikan nilai minimum dari matrix dengan axis tertentu}
function zeros ((a : int, b : int ), type : dtype)→ matrix
{Fungsi dari library numpy untuk membuat matrix axb berisi 0}
function count_nonzero(m: matrix) → integer
{Fungsi dari library numpy untuk menghitung elemen matrix m yang tidak nol}
function zeros-like (mat: matrix, type: dtype) → matrix
{Fungsi dari library numpy untuk membuat matrix salinan dari mat dengan nilai berisi 0}
function convert_rgb_to_hsv (RGBImage: numpy_array) → numpy_array
{Fungsi untuk mengubah nilai RGB menjadi HSV yang sudah dikuantisasi}
function quantify (h: numpy_array, s: numpy_array, v: numpy_array) → (numpy_array,
numpy_array, numpy_array)
{Fungsi untuk mengkuantisasi nilai-nilai hsv}
function cosine_similarity(vector1: numoy_array, vector2: numoy_array) → float
{Fungsi menghitung cosine similarity antara dua vektor}
function histogram (hsv: numpy_array) → numpy_array
{Fungsi menmbuat histogram dari suatu gambar}
function pencarian_blok (query: numpy_array, database: numpy_array) → float
{Fungsi mencari cosine similarity pada blok 3x3}
function color_based_image_retrieval (query_image: numpy_array, database_images: array of
numpy_array) → array_of_image
{Fungsi utama, mencari gambar yang mirip dengan query pada database, dan mengembalikan array of image yang memiliki similarity>60%}

REALISASI

function convert_rgb_to_hsv (RGBImage: numpy_array) → numpy_array

KAMUS LOKAL
image, r, g, b, h, s, v, cmax, cmin, delta, quantified_hsv : numpy_array
mask_r, mask_g, mask_b, nonzero_mask : boolean
```

ALGORITMA

```
{Ubah format dari BGR ke RGB karena cv2 otomatis membaca gambar dengan format BGR}

image ← image[:, :, ::-1]

{normalisasi nilai rgb}

image ← image/255

{pisah channel rgb}

r ← image[:, :, 0]

g ← image[:, :, 1]

b ← image[:, :, 2]

cmax ← maximum(image, axis=2)

cmin ← minimum(image, axis=2)

delta ← (cmax-cmin)

h ← zeros_like(r)

s ← zeros_like(r)

mask_r ← logical_and(delta!=0, cmax==r)

mask_g ← logical_and(delta!=0, cmax==g)

mask_b ← logical_and(delta!=0, cmax==b)

{hitung nilai h}

h[delta==0] ← 0

h[mask_r] ← 60*((g[mask_r]-b[mask_r])/(delta[mask_r]))%6

h[mask_g] ← 60*((b[mask_g]-r[mask_g])/(delta[mask_g]))+2

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

{hitung nilai S}

nonzero_mask ← cmax != 0

s[nonzero_mask] ← delta[nonzero_mask] / cmax[nonzero_mask]

{hitung nilai v}

v ← cmax

{kuantisasi hsv}

h, s, v = quantify(h, s, v)
```

```
{susun h,s,v menjadi array image}

quantified_hsv ← transpose([h, s, v], (1, 2, 0))

quantified_hsv ← quantified_hsv.astype(int)

→ quantified_hsv

function quantify (h: numpy_array, s: numpy_array, v: numpy_array) → (numpy_array,
numpy_array, numpy_array)
```

KAMUS LOKAL

ALGORITMA

```
h[logical_and(h>=0, h<=25)] ← 1

h[h>=316] ← 0

h[logical_and(h>25, h<=40)] ← 2

h[logical_and(h>40, h<=120)] ← 3

h[logical_and(h>120, h<=190)] ← 4

h[logical_and(h>190, h<=270)] ← 5

h[logical_and(h>270, h<=295)] ← 6

h[logical_and(h>295, h<=315)] ← 7

s[s>=0.7] ← 2

s[logical_and(s>=0.2, s<0.7)] ← 1

s[s<0.2] ← 0

v[v>=0.7] ← 2

v[logical_and(v>=0.2, v<0.7)] ← 1

v[v<0.2] ← 0

→ (h, s, v)
```

```
function cosine_similarity(vector1: numoy_array, vector2: numoy_array) → float
```

KAMUS LOKAL

```
dot_product, norm_vector1, norm_vector2, similarity: float
```

ALGORITMA

```
dot_product ← dot(vector1, vector2)

norm_vector1 ← norm(vector1)
```

```

norm_vector2 ← norm(vector2)

similarity ← dot_product / (norm_vector1 * norm_vector2)

→ similarity

function histogram (hsv: numpy_array) → numpy_array

KAMUS LOKAL
h_thresholds, s_thresholds, v_thresholds: array of integer
h_histogram, s_histogram, v_histogram, hist: numpy_array

h_mask, s_mask, v_mask : boolean

i : integer

ALGORITMA

h_thresholds ← [0, 1, 2, 3, 4, 5, 6, 7]

s_thresholds ← [0, 1, 2]

v_thresholds ← [0, 1, 2]

h_histogram ← zeros(len(h_thresholds), dtype=object)

s_histogram ← zeros(len(s_thresholds), dtype=object)

v_histogram ← zeros(len(v_thresholds), dtype=object)

i traversal [0..len(h_thresholds)-1]

    h_mask ← (hsv[:, :, 0] == h_thresholds[i])

    h_histogram[i] ← count_nonzero(h_mask)

i traversal [0..len(s_thresholds)-1]

    s_mask ← (hsv[:, :, 1] == s_thresholds[i])

    s_histogram[i] ← count_nonzero(s_mask)

    v_mask ← (hsv[:, :, 2] == v_thresholds[i])

    v_histogram[i] ← count_nonzero(v_mask)

hist ← concatenate((h_histogram, s_histogram, v_histogram))

→ hist

function pencarian_blok (query: numpy_array, database: numpy_array) → float

```

KAMUS LOKAL

```
i, j, p, q, r, s, height_q, width_q, height_db, width_db: integer
similarities : array [...] of float
block_query, block_database : array [...] of float
average_similarity, similarity_percentage, similarity : float
hist_query, hist_db : array [0..14] of integer
block_height_q, block_width_q, block_height_db, block_width_db: array[0..3] of integer
```

ALGORITMA

```
height_q, width_q, _ ← query.shape
height_db, width_db, _ ← database.shape

if (height_q%3==0) then
    block_height_q ← [int(height_q/3), int(height_q/3), int(height_q/3)]
else if (height_q%3==1) then
    block_height_q ← [int((height_q//3)+1), int(height_q//3), int(height_q//3)]
else
    block_height_q ← [int((height_q//3)+1), int(height_q//3), int((height_q//3)+1)]

if (height_db%3==0) then
    block_height_db ← [int(height_db/3), int(height_db/3), int(height_db/3)]
else if (height_db%3==1) then
    block_height_db ← [int((height_db//3)+1), int(height_db//3), int(height_db//3)]
else
    block_height_db ← [int((height_db//3)+1), int(height_db//3), int((height_db//3)+1)]

if (width_q%3==0) then
    block_width_q ← [int(width_q/3), int(width_q/3), int(width_q/3)]
else if (width_q%3==1) then
    block_width_q ← [int((width_q//3)+1), int(width_q//3), int(width_q//3)]
else
    block_width_q ← [int((width_q//3)+1), int(width_q//3), int((width_q//3)+1)]

if (width_db%3==0) then
    block_width_db ← [int(width_db/3), int(width_db/3), int(width_db/3)]
```

```

else if (width_db%3==1) then

    block_width_db ← [int((width_db//3)+1), int(width_db//3), int(width_db//3)]

else

    block_width_db ← [int((width_db//3)+1), int(width_db//3), int((width_db//3)+1)]


similarities ← []

p ← 0

r ← 0

i traversal [0..height_q, block_height_q[p]]

q ← 0

s ← 0

J traversal [0..width_q, block_width_q[q]]

block_query ← query[i:i+block_height_q[p], j:j+block_width_q[q]]

block_database ← database[r:r+block_height_db[p], s:s+block_width_db[q]]

hist_query ← histogram(block_query)

hist_db ← histogram(block_database)

similarity ← cosine_similarity(hist_query, hist_db)

similarities ← append(similarity)

s ← s + block_width_db[q]

q ← q + 1

r ← r + block_height_db[p]

p ← p + 1

similarities ← array(similarities)

average_similarity ← mean(similarities)

similarity_percentage ← (average_similarity)*100

→ similarity_percentage


function color_based_image_retrieval (query_image: numpy_array, database_images: array of numpy_array) → array of image

KAMUS LOKAL
query_image, db_image: integer
matches: array [...] of object

```

ALGORITMA

```
query_image ← convert_rgb_to_hsv(query_image)

matches ← []

I traversal [0..len(database_images)-1]

    db_image ← convert_rgb_to_hsv(database_images[i])

    similarity ← pencarian_blok(query_image, db_image)

    if (similarity>=60) then

        matches.append((image, round(similarity,2)))

matches ← sort(matches)

→ matches
```

4.1.3 Implementasi Image Scraping

Program Scrape

{Library untuk melakukan scraping image dari link tertentu}

Use PIL
Use io
Use requests
Use bs4
Use os

KAMUS UMUM

```
function get (url: string) → response
{Fungsi dari library requests untuk melakukan HTTP GET request dari url}
function BeautifulSoup (r.text, 'html.parser': string) → class_BeautifulSoup
{Fungsi dari library bs4 untuk melakukan parsing}
function find_all ('img': string) → Image
{Fungsi dari library bs4 untuk mengambil semua gambar dari file HTML}
function open (s: string) → matrix
{Fungsi dari library PIL untuk mengambil Image dari s dan mengubah ke matriks}
function path.exists (s: string) → boolean
{Fungsi dari os untuk mengecek keberadaan suatu direktori}
procedure makedirs (input s: string)
{Prosedur dari os untuk membuat sebuah direktori}
procedure write (input s: string)
{Prosedur bawaan Python untuk menuliskan string s ke dalam file}
procedure scrape_images (input url: string)
{Prosedur untuk mengambil gambar dari sebuah url}
```

REALISASI

procedure scrape_images (input url: string)

KAMUS LOKAL

r, im: response
soup: class_BeautifulSoup
images: Image
directory, link, image_path: string

```

existing_files: array [0..NEff] of files
file_count, i: integer
img: matrix

ALGORITMA

r ← get(url)
soup ← BeautifulSoup(r.text, 'html.parser')
images ← find_all('img')
directory ← 'static/datasetscrap'

if not (path.exists(directory)) then

    makedirs(directory)

existing_files ← [f for f in listdir(directory) if (path.isfile(path.join(directory, f)))]
file_count ← len(existing_files)

i traversal (image in enumerate(images, start=file_count + 1))

    link ← image['src']

    if (link.startswith('http')) then

        try

            im ← get(link)

            if (im.status_code == 200) then

                raise RequestException("Failed to download image ", link, ":
HTTP status code ", im.status_code)

            try

                img ← open(BytesIO(im.content))

            except

                output("Failed to open image", link)

            continue

            img.verify()

            image_path ← path.join(directory, "result_{", i, "}.jpg")

            with (open(image_path, 'wb')) as f

                f.write(im.content)

            except (RequestException) as e

                output(e)

                with (open("error_log.txt", 'a')) as error_file

                    error_file.write("Failed to download image ", link, ":", e)

                with open("error_image.txt", 'a') as error_img_file

```

```
error_image_file.write("Tidak bisa mengunduh gambar: ", link)
```

4.2 Penjelasan Struktur Program

Dalam pengembangan website kami, kami memilih Flask sebagai backend dan HTML sebagai frontend. Sebelum itu, disini akan dibahas terlebih dahulu struktur-struktur yang membangun program ini menjadi sedemikian rupa

- Template Rendering dengan 'render_template'

Untuk merender template halaman, kami mengandalkan fungsi 'render_template' Flask. Hal ini mempermudah pengembangan antarmuka pengguna yang dinamis dan responsif.

- Pengelolaan Sesi dan Penyimpanan Nilai dengan Library 'requests'

Dalam pengelolaan sesi, kami menggunakan library 'requests' untuk menyimpan nilai sesi, termasuk informasi seperti string nama file dan URL gambar. Ini memungkinkan kami memelihara data dengan efisien dan menyediakan pengalaman pengguna yang konsisten.

Kami merancang tiga halaman utama, yaitu 'About', 'Main', dan 'Use'. Halaman 'Main' bertindak sebagai landing page utama.

1. Halaman Utama ('Main')

Halaman 'Main' dirancang sebagai titik awal pengguna / *landing page* dengan tata letak yang menarik dan navigasi yang jelas. Di sini, pengguna dapat dengan mudah mengakses fitur utama, yaitu Content-Based Image Retrieval (CBIR), dengan mengklik tombol yang disediakan. Tak hanya itu, disini juga disediakan informasi sekilas mengenai cara penggunaan website ini.

2. Halaman Utama ('About')

Halaman 'About' digunakan sebagai laman informasi mengenai kami, para perancang program ini

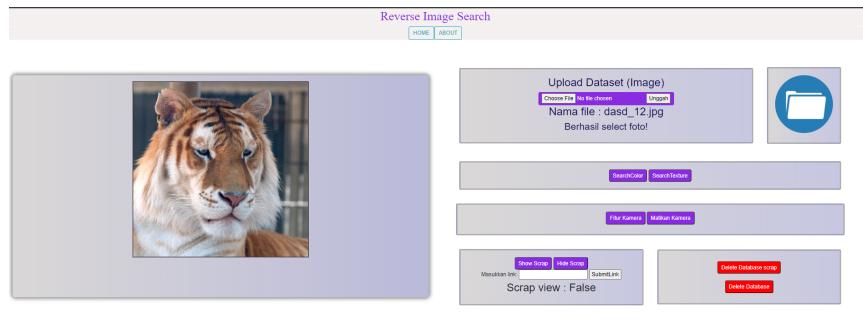
3. Implementasi Content-Based Image Retrieval (CBIR) di Halaman 'Use'

Halaman 'Use' didedikasikan sepenuhnya untuk proses CBIR serta segala fitur yang disediakan(real time camera, image scrap). Kami menggunakan Flask untuk menyediakan fungsionalitas ini dan memanfaatkan library 'requests' untuk menyimpan nilai sesi, seperti string nama file dan URL gambar. Pada langkah ini, kami juga menggunakan 'render_template' untuk memudahkan tampilan halaman dan library 'werkzeug.utils' sebagai utilitas untuk operasi transfer file di dataset.

Struktur untuk gambar-gambar hasil CBIR itu sendiri, disimpan dalam bentuk *array of image* yang nantinya akan disimpan didalam folder '*filter*'. Folder ini dirancang menjadi folder yang dinamis. Maksud dari dinamis ini adalah, isi folder filter akan berubah seiring digunakannya fitur CBIR. Perubahan ini dilakukan ketika user memasukan gambar referensi yang baru/*query* baru, yaitu dengan mengganti hasil foto olahan CBIR yang lama dengan yang baru. Dengan ini, ukuran folder akan selalu terminimalisir.

4.3 Cara Penggunaan Program

4.3.1 Segmen 1: Tampilan Gambar, Pilihan *Search*, dan *Upload*

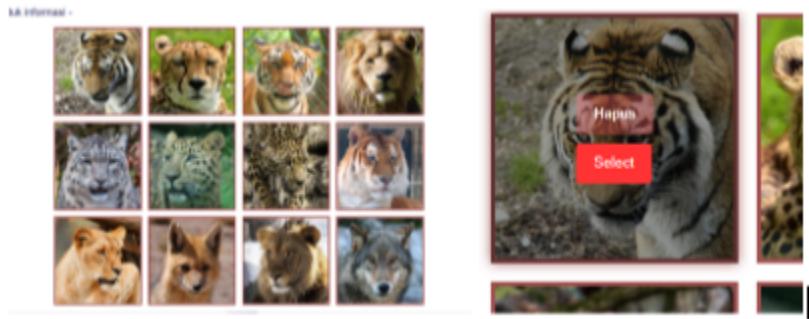


Pada segmen ini, sebelum pengguna ingin melakukan proses pencarian dengan 2 parameter, pengguna dapat meng *upload* gambar baik dalam bentuk file hingga folder yang berisi kumpulan gambar. Gambar gambar yang di-*upload* nantinya akan dijadikan sebagai *dataset*/referensi pencarian. Selanjutnya, pengguna akan melihat tampilan gambar yang diunggah dan siap untuk diproses dalam pencarian. Terdapat dua tombol utama untuk jenis pencarian CBIR, yaitu:

- A. *Search CBIR Secara Texture*: Tombol ini memungkinkan pengguna mencari gambar berdasarkan tekstur.
- B. *Search CBIR Secara Color*: Tombol ini memungkinkan pengguna mencari gambar berdasarkan warna.

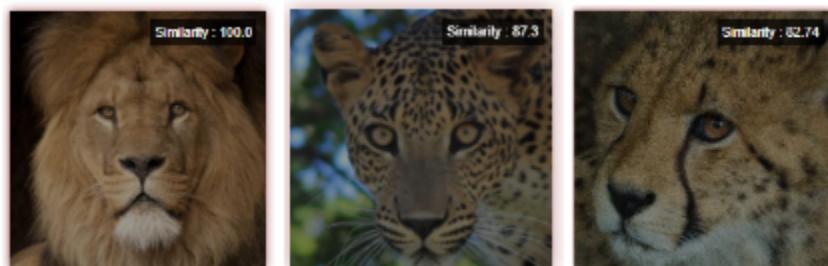
Fitur *Delete Database* - Untuk fitur ini dipisah jadi 2 buah *button*, yang satu berfungsi untuk menghapus seluruh *dataset*, dan yang satunya lagi menghapus *dataset* khusus *image* hasil *scraping*.

4.3.2 Segmen 2: Interaksi dengan *Image*



Pada segmen ini, pengguna dapat meng-*hover* kursor *mouse* ke gambar-gambar yang ada. Pengguna akan diberi 2 pilihan button yaitu ‘Select’ dan ‘Hapus’. Sesuai dengan namanya, *Select* akan menyeleksi gambar tersebut dan dijadikan sebagai *query* gambar untuk CBIR. *Button Hapus* berfungsi untuk menghapus gambar yang pengguna pilih dari *dataset*. Tiap *page*, gambar diberi maksimal sebanyak 12 buah, dan dapat berganti-ganti *page* dari *pagination* di bawah segmen ini.

Dengan menambahkan fungsionalitas *Select* dan *Hapus* pada gambar-gambar yang ditampilkan, pengguna dapat dengan mudah melakukan aksi yang diinginkan tanpa harus meninggalkan halaman. *Pagination* di bagian bawah juga memberikan kemudahan untuk melihat lebih banyak gambar dalam *dataset* tanpa mengalihkan pengguna ke halaman lain.



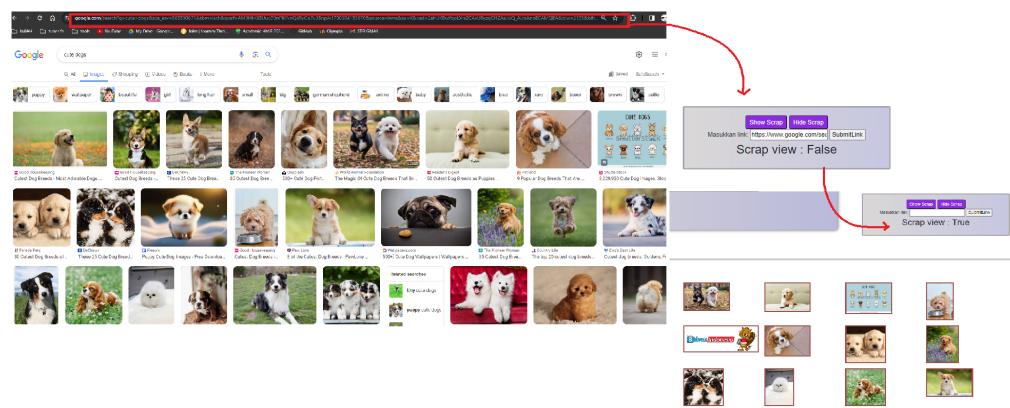
Berbeda tampilan untuk image yang sudah diolah baik menggunakan *search* berdasarkan *texture* maupun *color*. Ketika di-*hover*, maka urutan gambar akan diurut menurun/*descending* berdasarkan kemiripan/*similarity* terhadap gambar referensi.

4.3.3 Segmen 3: Kamera



Button Aktifkan Kamera: Mengaktifkan kamera dan akan secara otomatis menangkap foto setiap interval 5 detik. *Button Matikan Kamera:* Mematikan fitur kamera. Foto yang sudah di *capture* akan dialokasikan ke *database*, maksimal foto hasil *screenshot* hanya 1 saja dengan variabel nama tetap yaitu '*captured_image*'

4.3.4 Segmen 4: Segmen 4 : *Image Scraping* / Pengikisan Gambar



Fitur *Image Scraping*: - *Button View Scrap Database*: Menampilkan foto hasil dari *image scraping* ke dalam *database* untuk dicari dan diolah. - *Button Matikan View Scrap Database*: Mematikan tampilan *database* hasil *image*

scraping. - *Button Delete Database:* Menghapus data pada *database image scraping* (*database* awal dan hasil *scraping* dipisah)

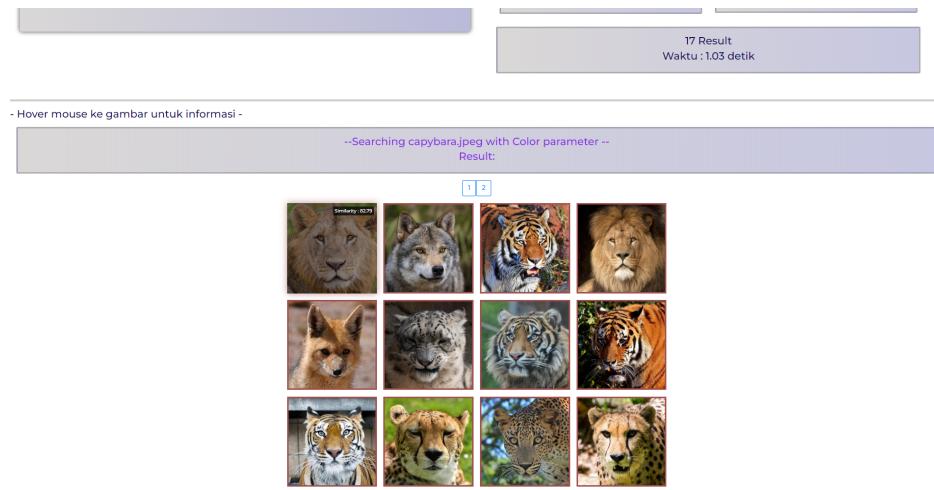
4.4 Hasil Pengujian

4.4.1 Pengujian CBIR dengan Parameter Warna

Gambar referensi:



Hasil pencarian dengan CBIR parameter warna:



The screenshot shows a search interface for a capybara image using color parameters. At the top right, it displays "17 Result" and "Waktu : 1.03 detik". Below the search bar, there is a message "- Hover mouse ke gambar untuk informasi -". The search results are presented in a grid of 17 images. The first row contains four images: a lion, a wolf, a tiger, and another lion. The second row contains four images: a fox, a snow leopard, a tiger, and another tiger. The third row contains three images: a tiger, a cheetah, and a leopard. Each image has a small red border around it, indicating they are the top matches. A navigation bar at the bottom of the grid shows "1 [2]". Above the search bar, there is a message "--Searching capybara.jpeg with Color parameter -- Result:".

4.4.2 Pengujian CBIR dengan Parameter Tekstur

Gambar referensi:



Hasil pencarian dengan CBIR parameter tekstur:

18 Result
Waktu : 4.37 detik

- Hover mouse ke gambar untuk informasi -

--Searching capybara.jpeg with Texture parameter --
Result:

[1 | 2]

4.4.3 Pengujian Fitur Kamera

Pengambilan gambar dari kamera perangkat:

Reverse Image Search

Upload Dataset (Image)

Search by URL

Nama file : None

Error! Anda mau search apa wkwkwk

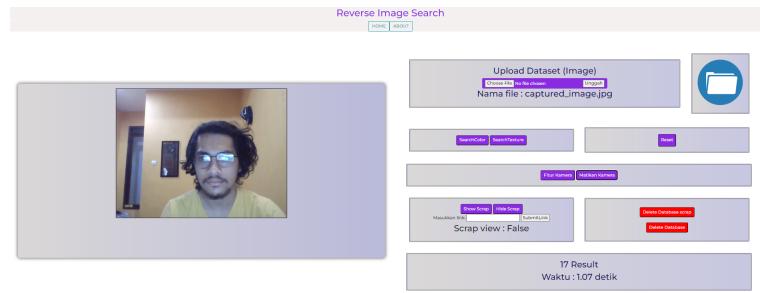
Fitur Deteksi

Gambar akan diambil dalam 3 detik.

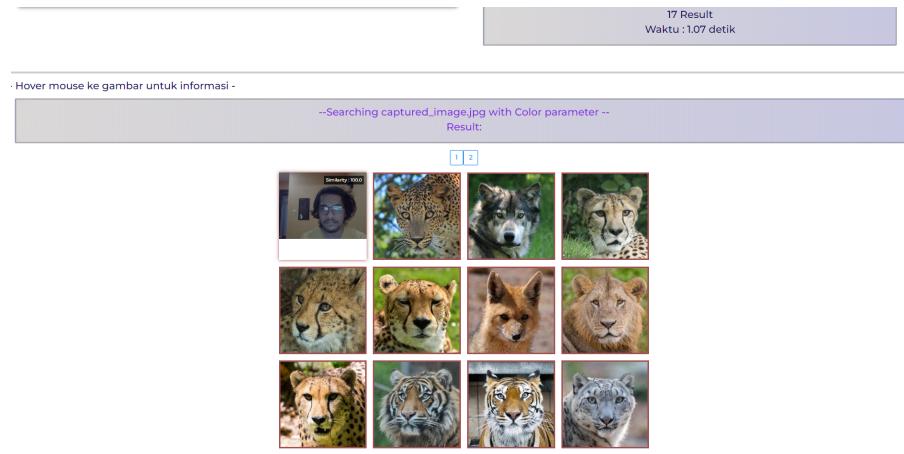
Scrap view : False

Deteksi Gambar

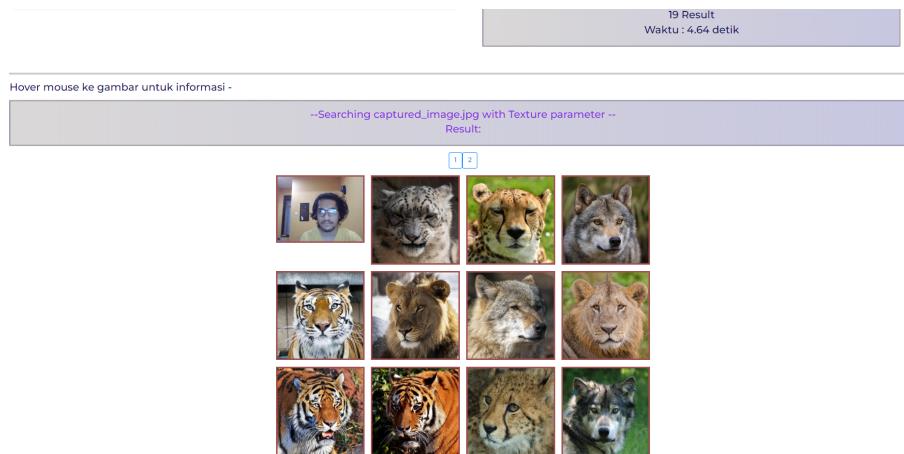
Gambar referensi:



Hasil pencarian dengan CBIR parameter warna:

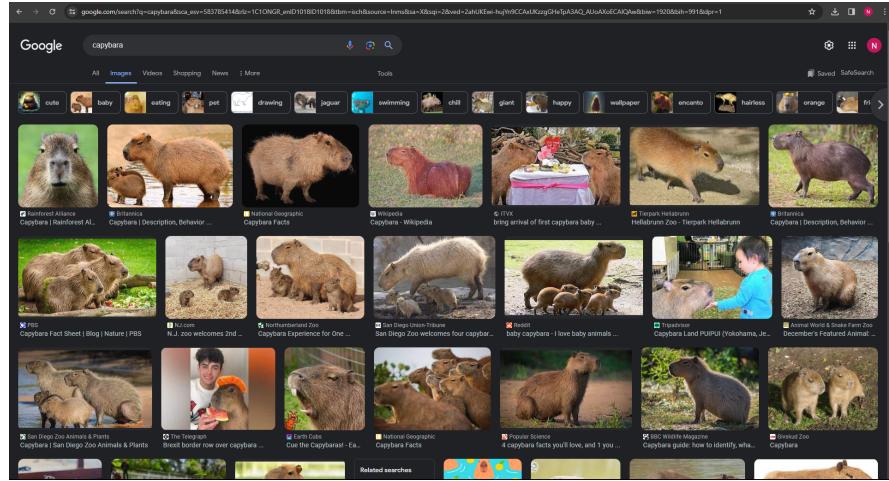


Hasil pencarian dengan CBIR parameter tekstur:

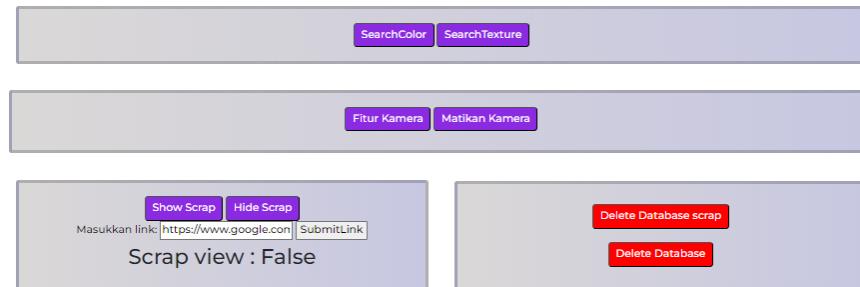


4.4.4 Pengujian Fitur *Image Scraping*

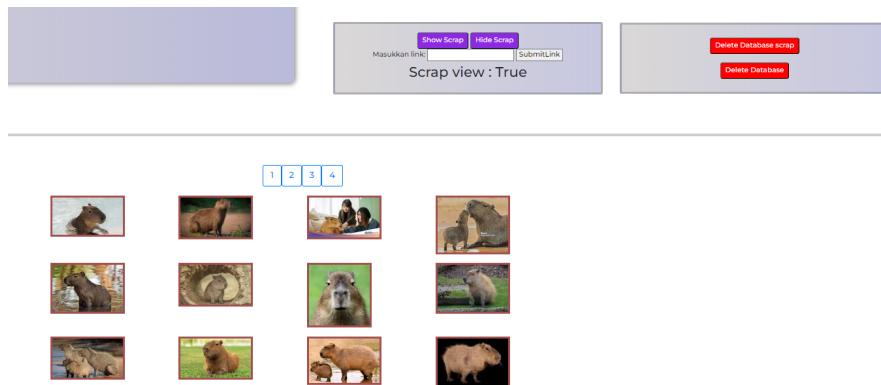
Web yang akan di-scrape:



Memasukkan *link url* ke website CBIR:



Hasil *scraping* yang dimasukkan ke *dataset*:



4.5 Analisis Desain Algoritma

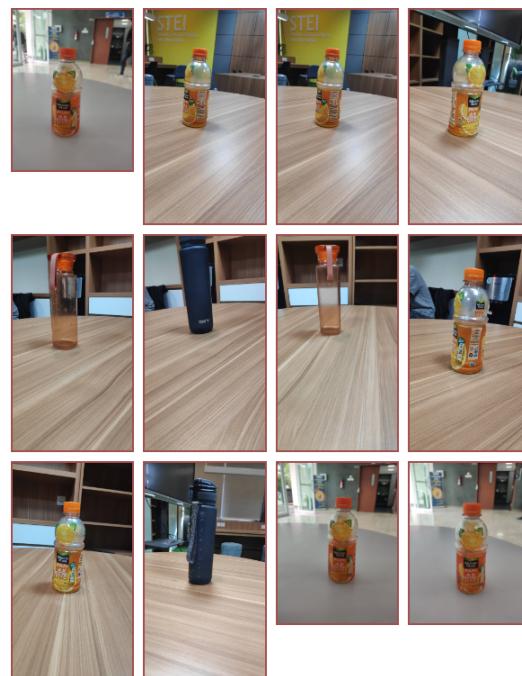
Penggunaan sistem temu balik gambar berbasis konten dengan parameter warna dan parameter tekstur memiliki kelebihan masing-masing. Terdapat kasus di mana CBIR

dengan parameter warna memberikan hasil yang lebih akurat dibanding dengan parameter tekstur serta sebaliknya.

Sebagai contoh, berikut merupakan gambar referensi yang akan digunakan dalam analisis kasus ini.



Berikut merupakan hasil pencarian gambar yang mirip dengan menggunakan parameter warna.



Berikut merupakan hasil pencarian gambar yang mirip dengan menggunakan parameter tekstur.



Pada penggunaan CBIR dengan parameter warna, hasil yang ditampilkan lebih akurat, 3 gambar paling mirip (posisi 1, 2, 3, 7, 8, 10, dan 11) memang merupakan gambar dengan isi yang serupa dengan gambar referensi (botol minuman berperisa jeruk), sementara pada CBIR dengan parameter tekstur, hasil pencarian menunjukkan bahwa botol minuman berperisa jeruk berada pada posisi 8, 9, dan 11. Penyebab hal ini adalah CBIR dengan parameter warna akan fokus pada warna yang dominan pada gambar (dalam kasus ini jingga) sehingga akan mengembalikan gambar dengan warna dominan jingga.

Namun, apabila kita melihat *background*-nya, CBIR dengan parameter tekstur akan menampilkan hasil yang lebih akurat, di mana gambar referensi berlatar di lantai dasar gedung KOICA dan 5 hasil pencarian juga berlatar di tempat yang sama. Sementara itu, hasil CBIR dengan parameter warna, hanya 2 gambar yang memiliki latar yang sama. Penyebab hal ini adalah CBIR dengan parameter tekstur mengabaikan warna (diubah ke *grayscale*) dan berfokus pada komponen kontras, homogenitas, dan entropi.

Gambar pulpy.jpg:



Gambar cws.jpg:



Gambar cola.jpg:



Berikut merupakan hasil konversi ketiga gambar ke *grayscale*.

Gambar pulpygscale.jpg:



Gambar cwscale.jpg:



Gambar colagscale.jpg:



Tanpa ‘intervensi’ warna, memang terlihat bahwa pulpygscale.jpg lebih mirip dengan colagscale.jpg dibanding dengan cwscale.jpg,

Bab V

Kesimpulan

5.1 Kesimpulan

Pemanfaatan sistem temu balik gambar berbasis konten dengan parameter warna dan tekstur memiliki kelebihan masing-masing. Dalam pencarian gambar yang mirip secara warna, jelas CBIR dengan parameter warna lebih baik digunakan. Namun, dalam pencarian gambar yang tidak relevan secara warna (misalkan gambar sebuah pemandangan dalam *RGB* dan dalam *grayscale*), CBIR dengan parameter tekstur lebih baik untuk digunakan.

5.2 Saran

Website sistem temu balik gambar berbasis konten yang telah dibuat disarankan untuk digunakan dalam masalah pencarian gambar yang mirip dan dikembangkan ke depannya agar menjadi lebih optimal.

5.3 Komentar atau Tanggapan

- *Website* yang telah dibuat sudah cukup baik untuk ukuran pemula yang belum memiliki pengalaman banyak dalam bidang *web development*.
- **Mungkin** terdapat sebuah *secret page* yang dapat diakses pengguna *website*.

5.4 Refleksi

Kelompok memiliki kerja sama yang sangat baik dalam pengembangan *website* dan pembuatan algoritma CBIR. Tugas besar ini dapat selesai dengan tepat waktu dan tanpa kendala yang berarti.

5.5 Ruang Perbaikan atau Pengembangan

- Waktu yang dibutuhkan untuk mendapatkan hasil pencarian gambar yang mirip, baik warna maupun tekstur, masih terlalu lama. Efisiensi algoritma yang digunakan mungkin dapat diperbaiki agar waktu pencarian menjadi lebih singkat.

- *Styling* (css) website walaupun sudah bagus, masih memiliki kekurangan seperti kotak-kotak yang tidak simetris dan teks yang terlihat seperti tidak pada tempat yang seharusnya.

Daftar Pustaka

- Alake, Richmond. 2023. “Understanding Cosine Similarity and Its Applications” di <https://builtin.com/machine-learning/cosine-similarity> (diakses 19 November 2023).
- Athanasiou, Lambros S., Dimitrios I. F., Lampros K. M. 2017. *4 - Plaque Characterization Methods Using Intravascular Ultrasound Imaging*. Jurnal ScienceDirect. Bab 4:71-94.
- Baeldung. 2022. “What Is Content-Based Image Retrieval” di [https://www.baeldung.com/cs/cbir-tbir#:~:text=Content%2DBased%20Image%20Retrieval%20\(CBIR\)%20is%20a%20way%20of,image%20to%20the%20database%20images](https://www.baeldung.com/cs/cbir-tbir#:~:text=Content%2DBased%20Image%20Retrieval%20(CBIR)%20is%20a%20way%20of,image%20to%20the%20database%20images) (diakses 19 November 2023)
- Flask. 2010. “Quickstart - Flask Documentation” di <https://flask.palletsprojects.com/en/2.1.x/quickstart/> (diakses 19 November 2023).
- HTML. 2017. “W3C Recommendation: HTML5.2 , HTML Introduction” https://www.w3schools.com/html/html_intro.asp (diakses 19 November 2023)
- MDN. 2023. “What is CSS?” di https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS (diakses 19 November 2023)
- Zola, Andrew. 2022. “image” di <https://www.techtarget.com/whatis/definition/image> (diakses 19 November 2023).

Link *repository* Github:

<https://github.com/miannetopokki/Algeo02-22129>