

Laporan Tugas Besar 2 IF2123

Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar

Ditujukan untuk memenuhi salah satu tugas besar mata kuliah IF2123 Aljabar Linier dan Geometri pada Semester I Tahun Akademik 2023/2024

Disusun oleh:

Yosef Rafael Joshua (K3)	13522133
Muhammad Roihan (K3)	13522152
Rayhan Ridhar Rahman (K3)	13522160



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

2023

DAFTAR ISI

DAFTAR ISI.....	ii
BAB I.....	1
BAB II.....	2
2.1. Dasar Teori Pengambilan Gambar.....	2
2.1.1. Aljabar Vektor.....	2
2.1.2. CBIR Warna.....	2
2.1.3. CBIR Tekstur.....	4
2.2. Website.....	5
2.2.1. Membangun Website.....	5
2.2.2. Implementasi Website Kami.....	6
BAB III.....	7
3.1. Langkah-Langkah Pemecahan Masalah.....	7
3.1.1. CBIR Warna.....	7
3.1.2. CBIR Tekstur.....	7
3.2. Pemetaan Masalah.....	7
3.2.1. CBIR Warna.....	7
3.2.2. CBIR Tekstur.....	8
3.3. Ilustrasi Kasus dan Penyelesaian.....	8
3.3.1. CBIR Warna.....	8
3.3.2. CBIR Tekstur.....	9
BAB IV.....	10
4.1. Implementasi Program Utama (pseudocode).....	10
4.1.1. CBIR Warna.....	10
4.1.2. CBIR Tekstur.....	13
4.2. Struktur Program.....	15
4.3. Tata Cara Penggunaan Program.....	16
4.4. Hasil Pengujian.....	16
4.5. Analisis Desain Solusi Algoritma.....	20
BAB V.....	21
5.1. Kesimpulan.....	21
5.2. Saran.....	21
5.3. Komentar dan Tanggapan.....	21
5.4. Refleksi.....	21
5.5. Ruang Perbaikan atau Pengembangan.....	22
DAFTAR PUSTAKA.....	23
LAMPIRAN.....	24

BAB I

DESKRIPSI MASALAH

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

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. Pada Tugas Besar kali ini, pera penulis diminta untuk mengimplementasikan 2 parameter CBIR yang paling populer, yaitu CBIR berdasarkan tekstur dan CBIR berdasarkan warna.

Di dalam Tugas Besar 2 ini, para penulis 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 II

LANDASAN TEORI

2.1. Dasar Teori Pengambilan Gambar

2.1.1. Aljabar Vektor

Ruang vektor atau ruang Euclidean adalah tempat vektor didefinisikan. Ruang vektor dinotasikan sebagai R^n dengan n adalah dimensi dari ruang vektor tersebut. Vektor-vektor di dalam ruang vektor R^n direpresentasikan sebagai berikut

Vektor di R^n :

$$\mathbf{v} = (v_1, v_2, \dots, v_n) \text{ atau } \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Vektor nol adalah vektor yang semua elemennya bernilai nol. Warna dengan representasi RGB merupakan salah satu ruang vektor. Vektor $c = (r, g, b)$ merepresentasikan sebuah warna yang ada pada model warna RGB. Sebuah gambar (*image*) dapat direpresentasikan sebagai matriks $m \times n$ yang setiap elemennya adalah vektor warna pada model warna RGB.

Penjumlahan 2 vektor didefinisikan sebagai berikut

$$\begin{aligned} &\text{Jika } \mathbf{v} = (v_1, v_2, \dots, v_n) \text{ dan } \mathbf{w} = (w_1, w_2, \dots, w_n) \\ &\text{maka } \mathbf{v} + \mathbf{w} = (v_1 + w_1, v_2 + w_2, \dots, v_n + w_n) \end{aligned}$$

Perkalian vektor dengan skalar didefinisikan sebagai berikut

$$\text{Jika } \mathbf{v} = (v_1, v_2, \dots, v_n) \text{ maka } k\mathbf{v} = (kv_1, kv_2, \dots, kv_n)$$

Perkalian titik disebut juga sebagai *Euclidean Inner Product* atau sebagai *Dot Product*. Jika ada dua vektor pada ruang vektor yang sama, maka kedua vektor tersebut dapat dikenakan *Dot Product*. Jika \mathbf{u} dan \mathbf{v} adalah vektor tidak nol pada ruang vektor yang sama, maka perkalian titik antara kedua vektor tersebut adalah

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\| \|\mathbf{v}\| \cos \theta$$

2.1.2. CBIR Warna

CBIR dengan perbandingan warna adalah yang paling sering digunakan untuk sistem pengambilan gambar. CBIR dengan perbandingan warna ini dilakukan dengan mengubah gambar yang berbentuk RGB menjadi sebuah histogram warna yang lebih umum.

Histogram warna adalah frekuensi dari berbagai warna yang ada pada ruang warna tertentu hal ini dilakukan untuk melakukan pendistribusian warna dari gambar. Terdapat dua jenis histogram gambar yaitu, histogram warna global dan blok.

Pada perhitungan histogram, warna global HSV (Hue, Saturation, Value) lebih dipilih karena warna tersebut dapat digunakan pada kertas (*background* berwarna putih) yang lebih umum untuk digunakan. Maka citra warna RGB perlu diubah dulu ke dalam bentuk HSV. Berikut adalah cara mengubah nilai RGB ke HSV:

1. Normalisasi komponen warna RGB

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

2. Mencari nilai maksimum dan nilai minimum dari ketiga nilai tersebut dan selisihnya

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

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

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

3. Gunakan rumus di bawah ini untuk mendapatkan H(0°-360°), S(0-100%), V(0-100%)

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

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

$$V = C_{max}$$

Gunakan rumus tersebut pada setiap piksel dari gambar yang berbentuk RGB, sehingga gambar yang dibentuk oleh komponen HSV.

Setelah gambar HSV terbentuk, untuk mendapatkan histogram warna blok, maka gambar tersebut akan dibagi menjadi 4×4 blok yang akan digunakan untuk pembuatan histogram. Kemudian setelah masing-masing blok histogram membentuk suatu vektor, menggunakan metode *cosine-similarity*, kemiripan satu gambar dengan yang lain dapat terhitung. Dinyatakan semakin mirip ketika nilai mendekati 1.

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

2.1.3. CBIR Tekstur

CBIR dengan perbandingan tekstur dilakukan dengan menggunakan *co-occurrence matrix*. Keunggulan dari *co-occurrence matrix* adalah proses komputasi yang lebih sedikit dan waktu ekstraksi fitur yang lebih cepat. Dengan *co-occurrence matrix*, vektor yang didapat akan memiliki dimensi yang lebih kecil sehingga mempercepat proses temu balik gambar. Co-occurrence matrix dirumuskan sebagai berikut:

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

dengan Δx dan Δy sebagai *offset* gambar, i dan j sebagai intensitas gambar, p dan q sebagai posisi gambar.

Setelah co-occurrence matrix berhasil dibuat, selanjutnya co-occurrence matrix akan dijumlahkan dengan transposenya. Kemudian akan dilakukan normalisasi dengan rumus:

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

Langkah pertama untuk CBIR Tekstur adalah mengubah gambar (*image*) dari bentuk RGB ke bentuk Grayscale dengan rumus berikut:

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

Selanjutnya akan dilakukan kuantifikasi Grayscale. Karena nilai Grayscale berkisar di antara 0-255, maka co-occurrence matrix yang terbentuk akan berukuran 256 x 256.

Dari co-occurrence matrix yang terbentuk dapat diekstrak tiga komponen, yaitu *Contrast*, *Homogeneity*, *Entropy*. Ketiga komponen tersebut dirumuskan sebagai berikut:

Contrast:

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

Homogeneity :

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

Entropy :

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

Setelah diekstrak, ketiga komponen di atas akan membentuk suatu vektor yang unik untuk gambar tersebut. Untuk membandingkan dengan gambar yang lain digunakan *Teorema Cosine Similarity*:

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

Semakin besar nilai Cosine Similarity, maka kedua gambar tersebut dikatakan semakin mirip.

2.2. Website

2.2.1. Membangun Website

Secara umum sebuah website terdiri dari dua komponen utama, yaitu komponen *front-end* dan komponen *back-end*.

Front-end adalah bagian yang secara langsung dilihat oleh pengguna. Bagian *front-end* adalah tempat pengguna berinteraksi secara langsung dengan website yang dibuat. Bahasa yang sering dipakai untuk *front-end* adalah HTML, CSS, dan JavaScript. Library dan framework yang umum digunakan untuk *front-end* adalah React, Bootstrap, Angular, Vue.js.

Back-end adalah bagian yang tidak dilihat oleh pengguna secara langsung. Bagian *back-end* meliputi server dari website, database website, dan fungsionalitas dari website tersebut. Bahasa yang sering dipakai untuk *back-end* adalah Python, Ruby, dan Java. MySQL, dan MongoDB dapat dipakai untuk membantu manajemen database website. Framework server yang mendukung *back-end* adalah Django(Python).

2.2.2. Implementasi Website Kami

Untuk menghemat waktu penggerjaan, pada tugas besar ini kami tidak membuat website secara murni dari nol melainkan menggunakan beberapa library dan framework untuk membantu dalam pengembangan website dan fungsionalitas website.

Kami menggunakan library react untuk JavaScript untuk membantu membuat bagian *front-end* dari website kami. Sedangkan untuk bagian *back-end* kami menggunakan framework flask untuk bahasa Python.

Untuk fungsionalitas website, kami menggunakan bahasa Python dengan dukungan beberapa library. Library yang kami pakai adalah openCV (pembacaan dan pemrosesan awal gambar), beautifulsoup (image scraping), numpy (membantu operasi matriks dan vektor), dan numba (optimasi program).

BAB III

ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah

3.1.1. CBIR Warna

Untuk CBIR dengan perbandingan warna, hal yang utama digunakan adalah histogram warna dari *image*. Tetapi menurut spesifikasi, pertama diperlukan untuk mengubah format gambar yang asal *channel*-nya *Red*, *Green*, and *Blue* (RGB) menjadi ke dalam *Hue*, *Saturation*, dan *Value* (HSV). Maka langkah pertama yang harus dilakukan adalah cara untuk melakukan konversinya. Setelah konversi, dilakukan pembagian area pada gambar menjadi 4×4 blok yang kemudian dihitung histogramnya, ini dinamakan histogram blok. Lalu nilai-nilai histogram blok disimpan sebagai vektor, yang kemudian dengan *cosine-similarity* dibandingkan dengan vektor blok gambar lain.

3.1.2. CBIR Tekstur

Untuk CBIR dengan perbandingan tekstur langkah pertama untuk membandingkan tekstur gambar adalah dengan membuat co-occurrence matrix dari gambar (*image*) yang akan dibandingkan. Setelah co-occurrence matrix berhasil dibuat, matrix tersebut akan dinormalisasi. Kemudian baru dapat dilakukan ekstraksi komponen *contrast*, *homogeneity*, dan *entropy*. Ketiga komponen tersebut dijadikan sebuah vektor yang unik terhadap gambar tersebut. Karena masing-masing gambar memiliki vektor yang unik, maka untuk membandingkannya dapat menggunakan cosine similarity.

3.2. Pemetaan Masalah

3.2.1. CBIR Warna

Masalah pertama yang dihadapi adalah return value dari perubahan rgb ke hsv. Pada awal range untuk *hue* masih 0 - 360. Selain itu *saturation* dan *value* berada pada range 0 - 1. Ini membuat library openCV tidak bisa mengolah data warna karena kelebihan dari range. Selain itu calcHist di awal tidak digunakan karena hal tersebut juga, dan menggunakan keringinan dengan rata-rata nilainya.

Selain itu, pada pembagian blok terjadi beberapa dilemma untuk menghiraukan 1 - 3 piksel yang dapat hilang. Karena itu dibuat lagi pengkondisionan yang membuat program memakan lebih banyak waktu dan memori lagi. Selain itu, solusinya mempengaruhi juga *readability* dari kode.

Kemudian yang dipermasalahkan lagi di awal adalah bagaimana mencari kemiripan dari kedua gambar? Karena *hue* adalah satuan derajat sedangkan *saturation* dan *value* adalah persentase. Ditakutkan *hue* akan sangat mempengaruhi nilai dari kemiripan. Selanjutnya, setelah semua fungsi dibuat, hasil perbandingan memakan waktu yang kurang memuaskan sehingga perlu dioptimasi lagi.

3.2.2. CBIR Tekstur

Masalah yang pertama dihadapi adalah co-occurrence matrix. Untuk membuat sebuah co-occurrence matrix diperlukan representasi gambar dalam matriks grayscale. Oleh karena itu program pertama-tama akan membaca gambar (*image*) sebagai matriks RGB. Dalam implementasi kami menggunakan library openCV untuk proses ini. Selanjutnya matriks RGB tersebut akan dikonversi menjadi matriks grayscale. Dalam implementasi, kami membuat fungsi sendiri untuk mengubah matriks RGB menjadi matriks grayscale. Setelah beberapa kali percobaan ternyata algoritma dengan *nested loop* untuk fungsi ini menghasilkan waktu yang sangat lama. Oleh karena itu kami menggunakan fungsi dot dari numpy. Pada intinya setiap elemen gambar adalah vektor warna yang terdiri dari komponen red, green, blue(Blue, Green, Red jika dibaca dengan openCV) sehingga jika masing-masing elemen dikenakan dot product dengan vektor [0.114, 0.587, 0.29], akan menghasilkan matriks grayscale. Lalu untuk konversi dari matriks grayscale ke co-occurrence matrix, kami membuat fungsi sendiri. Lagi-lagi setelah beberapa kali percobaan ternyata menggunakan nested loop menghasilkan waktu yang lama. Oleh karena itu kami hanya memakai satu loop, yang digunakan untuk sekaligus mengakses kolom matriks grayscale. Lalu kami menggunakan beberapa fungsi numpy seperti transpose untuk menghasilkan normalisasi dari co-occurrence matrix.

Masalah berikutnya adalah melakukan ekstraksi komponen contrast, homogeneity, dan entropy dari co-occurrence matrix. Dalam implementasinya kami membuat fungsi buatan. Fungsi ini akan menerima co-occurrence matrix lalu akan ada array kosong untuk menampung komponen yang akan diekstrak nanti. Kami menggunakan beberapa fungsi dari numpy untuk membantu proses ini. Beberapa fungsi yang kami pakai adalah sum(), arange(), reshape(), log10(), append(). Arange dan reshape digunakan untuk membentuk matriks yang setiap elemennya adalah $(i - j)^2$ dengan i dan j masing-masing adalah baris dan kolom matriks. Dengan begitu tidak perlu dilakukan nested loop untuk setiap elemen co-occurrence matrix, tapi co-occurrence matrix bisa langsung dioperasikan dengan matrix yang telah disebutkan di atas untuk mendapatkan *contrast*, *homogeneity*, dan *entropy*.

Kemudian setelah vektor dibuat, maka dikenakan cosine similarity. Untuk ini kami membuat fungsi sendiri. Kami memanfaatkan fungsi bawaan dari numpy untuk membantu proses ini agar lebih cepat. Nilai yang didapat merepresentasikan kemiripan antara dua gambar (*image*).

3.3. Ilustrasi Kasus dan Penyelesaian

3.3.1. CBIR Warna

Cara konversi dari RGB dan HSV telah dipaparkan pada **2.1.2**, yaitu normalisasi, penentuan nilai maksimum dan minimum kemudian dihitung pula rangenya (selisih). Lalu perlu digunakan rumus. Tetapi rumus yang digunakan tidak sama dengan yang

terdapat pada referensi. Ini karena penggunaan openCV. Perlu diingat, openCV membaca image dengan format BGR (terbalik dengan RGB).

Hal utama yang perlu diubah adalah *Hue*, openCV hanya bisa membaca sampai 8-bit (maksimum 255) sehingga range 0-360° terlalu besar. Maka dikali dengan 30° dibanding 60° sehingga terdapat pada range 0-180°. Selain itu untuk optimasi, warna tidak dinormalisasi pada range 0 - 1 di awal. Untuk *hue*, hasil pengurangan elemen lain dibagi delta akan sama pada range 0 - 1. Begitu pula dengan *saturation*, karena pada dasarnya sama seperti dikali nilai 255/255. Hanya *value* yang perlu dinormalisasi.

Kemudian dipertimbangkan untuk menggunakan histogram atau rata-rata nilai HSV, pada akhirnya digunakan histogram karena sudah ada fungsi tersebut pada library openCV yang tidak ada di *ban-list*. *calcHist* hanya bisa menerima elemen bernilai **uint8** sehingga hasil dari konversi harus disesuaikan pula. Range masing-masing nilai menjadi *hue*(0 - 180), *saturation*(0 - 100), dan *value*(1 - 100).

Lalu pada pembagian *image* menjadi beberapa blok, mungkin kehilangan 1 - 3 piksel di ujung kanan/bawah gambar, tetapi perbedaan tersebut dianggap insignifikan. Bins untuk histogram mungkin terlalu banyak, yaitu setengah dari range yang mungkin.

Setelah selesai, dipikirkan lagi untuk optimasi dan memutuskan untuk menggunakan library Numba yang dapat mempercepat proses looping. Numba akan melakukan *compile* fungsi menjadi lebih sederhana untuk mesin baca, sehingga ketika pertama kali dipanggil, menghabiskan waktu lebih lama.

3.3.2. CBIR Tekstur

Contoh kasus sederhana untuk CBIR Tekstur adalah jika kita ingin membandingkan suatu gambar dengan tiga gambar lainnya. Pertama-tama program kami akan memproses gambar acuan. Gambar acuan akan dibaca sebagai matriks RGB, kemudian diubah ke matriks grayscale. Dari sini akan dibuat co-occurrence matrix dan diekstrak *contrast*, *homogeneity*, dan *entropy*. Kemudian vektor yang terbentuk akan disimpan di sebuah variabel. Lalu program akan melakukan proses yang sama untuk ketiga gambar yang akan dibandingkan dengan gambar acuan. Setiap gambar akan diproses satu-satu. Pada program akan ada array yang digunakan untuk menyimpan vektor-vektor yang terbentuk. Setelah semua gambar telah diproses maka program akan membandingkan vektor acuan dengan vektor-vektor yang ada pada array dengan metode cosine similarity. Jika hasilnya lebih dari 60% maka gambar itu akan disimpan ke array lain yang isinya adalah semua gambar yang memiliki cosine similarity lebih dari 60% dengan gambar acuan. Lalu Program akan mengurutkan array tersebut dan menampilkan hasilnya pada website.

BAB IV

IMPLEMENTASI DAN UJI COBA

4.1. Implementasi Program Utama (*pseudocode*)

4.1.1. CBIR Warna

```
Program CBIRwarna
{Library untuk mencari kesamaan dua gambar berdasarkan warna}

KAMUS UMUM

type bgr : < b : int[0..255],
        g : int[0..255],
        r : int[0..255]>
type hsv : < h : int[0..180],
        s : int[0..100],
        v : int[0..100]>
type matrixbgr : matrix[...] of bgr
type matrixhsv : matrix[...] of hsv
# [...] memiliki arti semua index i dan j yang berlaku

function imread(image_file_name : string) -> matrixbgr
{fungsi dari openCV untuk membaca gambar menjadi matriks berelemen (b,g,r)}
function calcHist(image : list of matrix, channel : list of int, mask : list of bool, bins : list of int, range : list of int) -> histType
{fungsi dari openCV untuk menghitung frekuensi kemunculan angka, hanya bisa membaca sampai 8-bits. Mengharapkan input dalam list. histType adalah histogram yang telah direpresentasikan dalam list float}
function normalize
function astype(any, type : dtype) -> cast type
{fungsi dari python untuk melakukan casting terhadap suatu objek}
function append(l1 : array of integer, element : int) -> array of integer
{fungsi dari python untuk menambahkan elemen ke suatu array}
function shape(mat : matrix) -> int
{fungsi dari numpy yang mengembalikan dimensi dari numpy array}
function zeros((a : int, b : int), type : dtype(opsional)) -> matrix
{fungsi dari numpy untuk membuat matrix 0 dengan ukuran a x b bertipe data type}
function zeros_like(m : matrix, type : dtype(opsional)) -> matrix
{fungsi dari numpy untuk membuat matrix 0 dengan dimensi yang sama dengan m}
function dot(m1, m2 : array of integer) -> int
{fungsi dari numpy untuk operasi dot product antara dua array integer}
function maximum(a : int, b : int) -> int
{fungsi dari numpy untuk mengembalikan nilai maksimum dari dua elemen}
function minimum(a : int, b : int) -> int
{fungsi dari numpy untuk mengembalikan nilai maksimum dari dua elemen}
function round(a : float, b : int (opsional)) -> int or float
{fungsi dari numpy untuk membulatkan a ke bilangan genap terbesar (jika argumen b tidak diisi), atau b desimal di belakang '.'}
procedure flatten(input/output matrix : matrix)
{prosedur dari numpy untuk membuat elemen list di dalam list menjadi 1 baris}
function concatenate((array1, array2, ... : array of any) ) -> array of any
```

```

{fungsi dari numpy untuk menggabungkan elemen list}
function dot(m1, m2 : array of integer) -> int
{fungsi dari numpy untuk operasi dot product antara dua array integer}
function mean(m : array of integer) -> float
{fungsi dari numpy untuk menghitung rata-rata di dalam array}
function sqrt(n : float) -> float
{fungsi dari numpy yang mengembalikan akar 2, bisa untuk matriks juga}
function prange(start : int, stop : int) -> array of integer
{fungsi dari numba untuk membuat array of integer dari start sampai stop - 1
yang dapat diiterasi secara paralel}

# fungsi buatan
function bgr2HSV(image : matrixbgr) -> matrix
{fungsi buatan yang menerima gambar dalam bentuk matriks RGB dan mengubahnya ke
dalam format HSV}
function calcHis_HSV(image : matrixHSV) -> array[0..3] of histType
{fungsi buatan yang menerima matriks HSV kemudian mengembalikan histogramnya}
function block_histograms(image : matrix) -> matrix[0..16,0..3] of histType
{fungsi buatan yang menerima matriks HSV dan mengembalikan array}
function color_similarity(image1, image2 : array of histType) -> float
{fungsi buatan yang menerima 2 array histogram untuk diperlakukan cosine
similarity yang nanti dijadikan mean}

```

REALISASI

```
function bgr2HSV(image : matrixbgr) -> matrixHSV
```

KAMUS LOKAL

```
b,g,r,cmax,cmin,delta : array[...] of integer
hsv_image : matrixHSV
```

ALGORITMA

```

b[...] <- image[...].b
g[...] <- image[...].g
r[...] <- image[...].r

cmax[...] <- maximum(maximum(b[...],g[...]),r[...])
cmin[...] <- minimum(minimum(b[...],g[...]),r[...])

delta[...] <- cmax[...] - cmin[...]

hsv_image <- zeros(shape(image))

i traversal [0..shape(image).row - 1]
j traversal [0..shape(image).col - 1]
if delta[i,j] ≠ 0 then
    if cmax[i][j] = r[i][j] then
        hsv_image[i,j].h <- 30 * (((g[i,j] - b[i,j]) / delta[i,j]) mod 6)
    else if cmax[i][j] = r[i][j] then
        hsv_image[i,j].h <- 30 * (((g[i,j] - b[i,j]) / delta[i,j]) + 2)
    else
        hsv_image[i,j].h <- 30 * (((g[i,j] - b[i,j]) / delta[i,j]) + 4)
    hsv_img[i,j].s <- round(delta[i,j] / cmax[i,j])
    hsv_img[i,j].v <- round(cmax[i,j] * 100 / 255)

-> hsv_image

```

```

function calcHis_HSV(block : matrixhsv) -> array of histType
KAMUS LOKAL
    h_unnormalized, s_unnormalized, v_unnormalized : matrix[...]
    hist_h, hist_s, hist_v : histType
ALGORITMA
    h_unnormalized[...] <- block[...].h
    s_unnormalized[...] <- block[...].s
    v_unnormalized[...] <- block[...].v

    hist_h[...] <- flatten(calcHist([h_unnormalized],[0],None,[90],[0..180]))
    hist_s[...] <- flatten(calcHist([s_unnormalized],[0],None,[50],[0..100]))
    hist_v[...] <- flatten(calcHist([v_unnormalized],[0],None,[50],[0..100]))

    -> concatenate((hist_h[...],hist_s[...],hist_v[...]))
```



```

function block_histograms(image : matrixhsv) -> array of array of histType
KAMUS LOKAL
    height,width,block_height,block_width,x_start,x_end,y_start,y_end : int
    block : matrixhsv
    hist : array of histType
    block_histograms : array of array of histType
ALGORITMA
    height <- shape(image).row
    width <- shape(image).col

    block_height <- height div 4
    block_width <- width div 4

    Block_histograms <- []

    i traversal prange(4)
        j traversal prange(4)
            y_start <- i * block_height
            y_end <- (i + 1) * block_height
            x_start <- j * block_width
            x_end <- (j + 1) * block_width

            block[...] <- image[y_start..y_end-1,x_start..x_end-1]

            hist <- calcHis(block)
            flatten(normalize(hist,hist))
            append(block_histograms,block)
```



```

    -> block_histograms
```



```

function color_similarity(image1, image2 : array of histType) -> float
KAMUS LOKAL
    similarities : array[0..15] of float
    vector1,vector2 : array[0..3] of histType
    dotproduct,norm1,norm2,result : float
ALGORITMA
    similarities <- zeros_like(block_hist1)

    i traversal prange(16)
        vector1 <- block_hist1[i]
        vector2 <- block_hist2[i]
```

```

dotproduct <- dot(vector1, vector2)

norm1 <- sqrt(np.dot(vector1, vector1))
norm2 <- sqrt(np.dot(vector2, vector2))
result <- dotproduct / (norm1 * norm2)
similarities[i] <- result

-> mean(similarities)

```

4.1.2. CBIR Tekstur

```

Program CBIRtekstur
{Library untuk mencari kesamaan dua gambar yang berbeda berdasarkan tekstur}

KAMUS UMUM
# fungsi-fungsi bawaan dari library dan python
function imread(image_file : string) -> matrix
{fungsi dari openCV untuk membaca gambar sebagai matrix}
function dot(m1, m2 : array of integer) -> int
{fungsi dari numpy untuk operasi dot product antara dua array integer}
function astype(any, type : dtype) -> cast type
{fungsi dari python untuk melakukan casting terhadap suatu objek}
function zeros((a : int, b : int), type : dtype) -> matrix
{fungsi dari numpy untuk membuat matrix 0 dengan ukuran a x b bertipe data
type}
function transpose(mat : matrix) -> matrix
{fungsi dari numpy untuk mencari transpose dari matrix mat}
function shape(mat : matrix) -> int
{fungsi dari openCV yang mengembalikan salah satu dimensi dari matrix}
function sum(mat : matrix) -> int
{fungsi dari numpy untuk mencari jumlah dari seluruh elemen matrix mat}
function arange(start : int, stop : int) -> array of integer
{fungsi dari numpy untuk membuat array of integer dari start sampai stop
[start, ..., stop]}
function reshape(mat : matrix, (a : int, b : int)) -> matrix
{fungsi dari numpy untuk mengubah dimensi matrix tanpa mengubah
elemen-elemennya}
function log10(mat : matrix) -> matrix
{fungsi dari numpy yang mengembalikan matrix baru dengan elemen matrix lama
yang sudah dikenakan operasi log dengan basis 10}
function sqrt(mat : matrix) -> matrix
{fungsi dari numpy yang mengembalikan matrix baru dengan elemen matrix lama
yang sudah dikenakan operasi akar pangkat 2}
function append(l : array of integer, element : int) -> array of integer
{fungsi dari python untuk menambahkan elemen ke suatu array}

# fungsi buatan
function toGray(image : matrix) -> matrix
{fungsi buatan yang menerima gambar dalam bentuk matriks dan mengubahnya ke
grayscale}
function coMat(grayScale : matrix) -> matrix
{fungsi buatan yang menerima matrix grayscale dan membentuk co-occurrence}

```

```

matrix dari grayscale tersebut}
function toVec(coOccurrence : matrix)-> array of integer
{fungsi buatan yang menerima co-occurrence matrix dan membentuk vektor dari
komponen Contrast, Homogeneity, dan Entropy}
function cosineSimilarity(l1, l2 : array of integer)-> float
{fungsi buatan yang menerima 2 array of integer dan menghitung cosine
similarity}

REALISASI
function toGray(image : matrix)-> matrix

KAMUS LOKAL
gray : matrix
ALGORITMA
gray <- astype(dot(image,[0.114, 0.587, 0.29]), uint8)
-> gray

function coMat(grayScale : matrix)-> matrix
KAMUS LOKAL
coMat : matrix
ALGORITMA
coMat <- zeros((256,256), uint8)
j traversal [0..shape(grayScale) [1] - 1]
coMat[grayScale[:,j],grayScale[:,j + 1]] += 1

coMat <- coMat + transpose(coMat)
-> (coMat/sum(coMat))

function toVec(coOccurrence : matrix)-> array of integer
KAMUS LOKAL
contrast, homogeneity, entropy : int
Width : int
Vec, l1, l2 : array of integer
ALGORITMA
Width <- shape(coOccurrence) [1]
l1 <- arange(Width)
l2 <- reshape(arange(Width, (-1,1)))

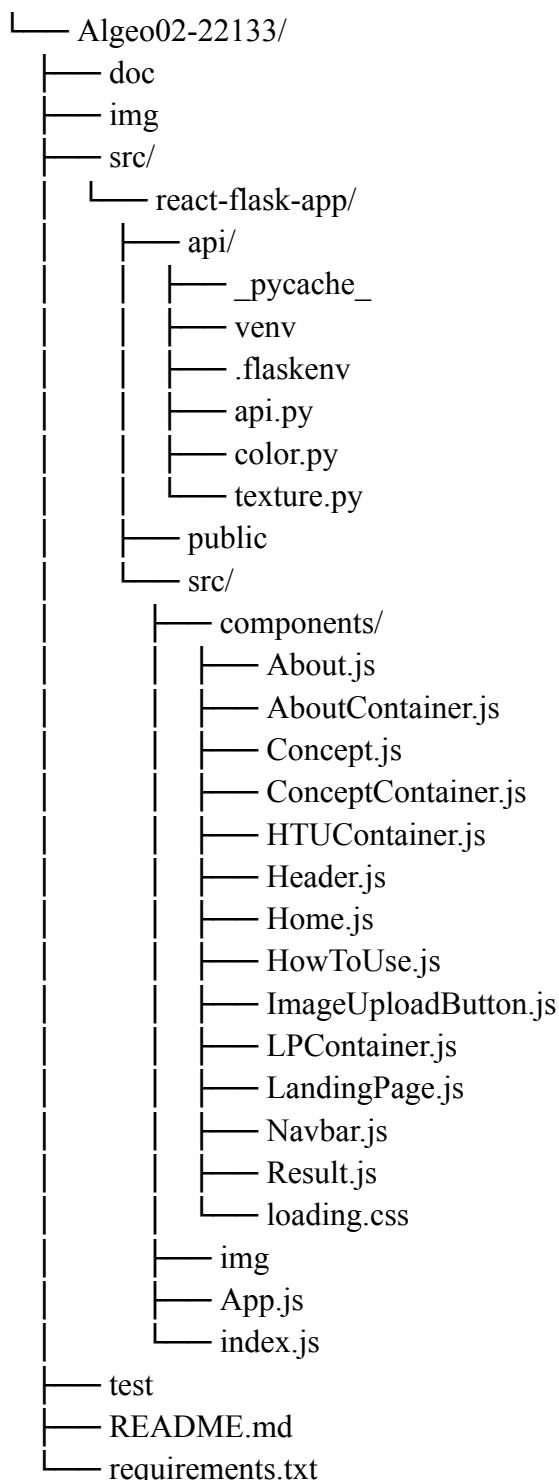
contrast <- sum(coOccurrence * (l1 - l2)**2)
homogeneity <- sum(coOccurrence / (1 + (l1 - l2)**2))
entropy <- -sum(log10(coOccurrence + (1e-15)) * coOccurrence)
append(Vec, contrast)
append(Vec, homogeneity)
append(Vec, entropy)
-> Vec

function cosineSimilarity(l1, l2 : array of integer)-> float
KAMUS LOKAL

ALGORITMA
-> (dot(l1, l2) / (sqrt(dot(l1, l1)) * sqrt(l2, l2)))

```

4.2. Struktur Program



4.3. Tata Cara Penggunaan Program

1. Upload Image Query

Image query berisi gambar yang akan digunakan dalam pencarian gambar. Untuk memasukkan *image query*, Anda dapat menggunakan tombol '*Insert Image*' atau menggunakan tombol '*Capture Image*'.

2. Upload Dataset

Kumpulan gambar (dataset), dilakukan dengan cara mengunggah multiple image dalam bentuk folder ke dalam web browser. Anda juga bisa mengambil dataset dari website lain menggunakan tombol image scraping.

3. Click Search Button

Setelah memasukkan *image query*, kumpulan gambar inilah yang akan diseleksi menjadi result.

4. Export

Hasil luaran program yang dapat diunduh ke *local storage* dalam format PDF.

4.4. Hasil Pengujian

1. Color dataset (43 image)

The screenshot shows a user interface for image search. At the top left are three buttons: 'Color' (blue), 'Insert Image' (blue), and 'Capture Image' (light blue). To the right is a large image of a tiger's face labeled '0.jpg'. Below this is a search bar with 'Search' and 'Total Result: 10' text. On the right is an 'Export' button. The search results are displayed in a grid of four images:

Image	Similarity
	Similarity: 100 %
	Similarity: 68.06 %
	Similarity: 64.73 %
	Similarity: 64.67 %

Time Taken : 3.401 s

2. Color dataset (4378 image)

Color

Insert Image

Capture Image



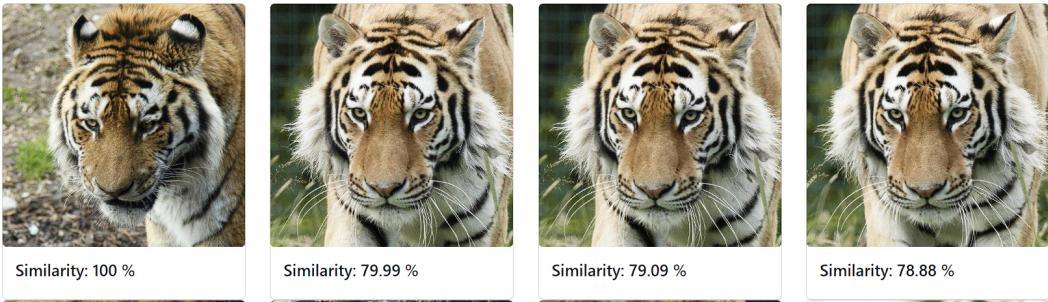
0.jpg

Search

Total Result: 1028

Export

Time Taken : 41.757199999928474 s



Similarity: 100 %

Similarity: 79.99 %

Similarity: 79.09 %

Similarity: 78.88 %

3. Texture dataset (43 image)

Texture

Insert Image

Capture Image



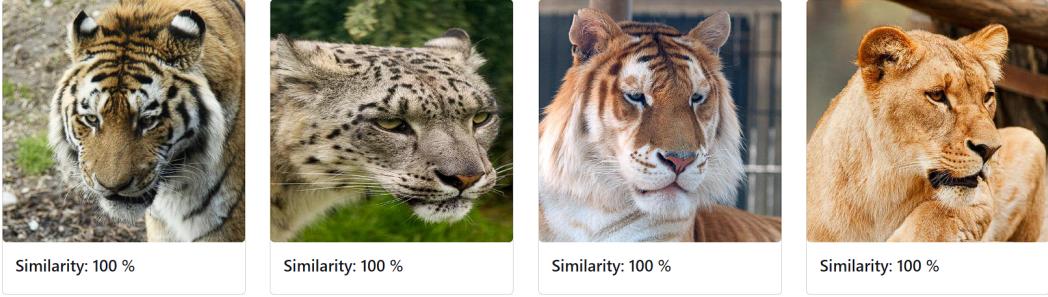
0.jpg

Search

Total Result: 44

Export

Time Taken : 1.5426000000238418 s



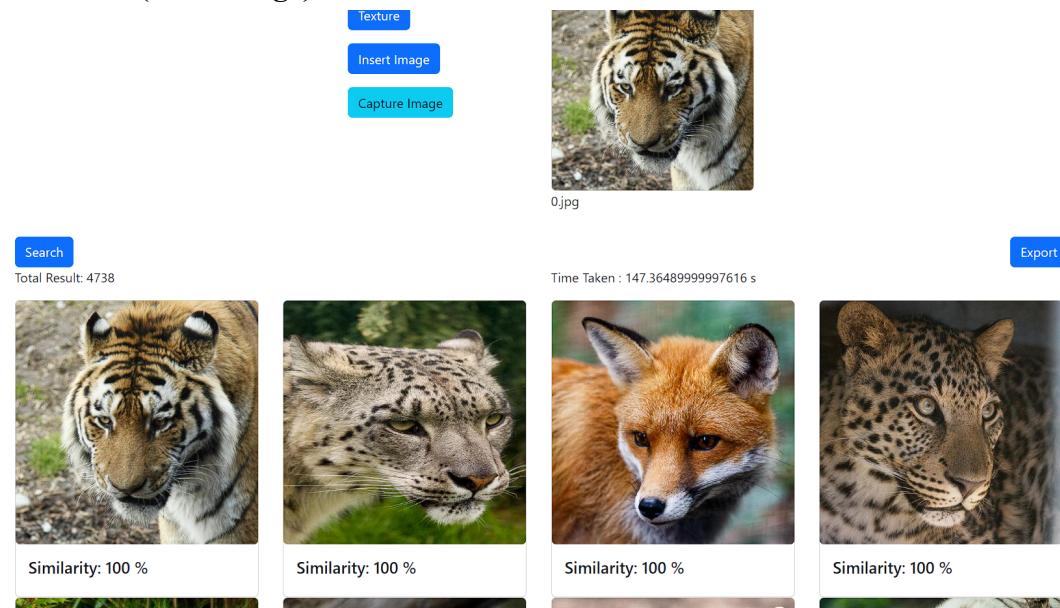
Similarity: 100 %

Similarity: 100 %

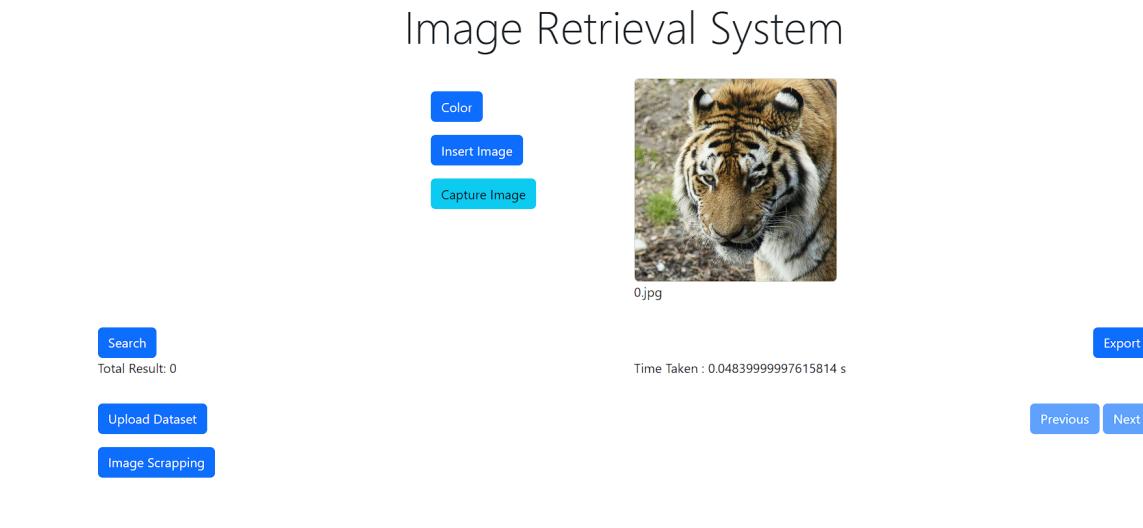
Similarity: 100 %

Similarity: 100 %

4. Texture dataset (4378 image)



5. Color image scraping (olympia.id)



6. Texture image scraping (olympia.id)

Image Retrieval System

Texture
Insert Image
Capture Image



0.jpg

Search Total Result: 9 Time Taken : 0.10939999997615814 s Export

Similarity: 99.96 %

Similarity: 99.96 %

Similarity: 99.96 %

Similarity: 99.96 %

7. Color capture image

Xandey Home Image Retrieval How To Use About Us Concept

Image Retrieval System

Color
Insert Image
Capture Image



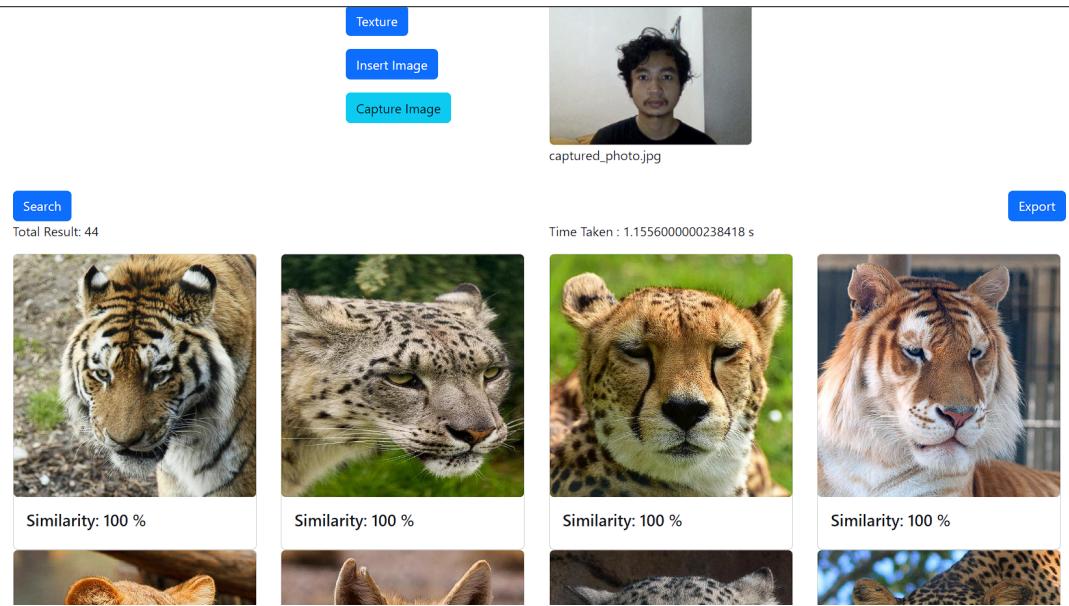
captured_photo.jpg

Search Total Result: 0 Time Taken : 0.788 s Export

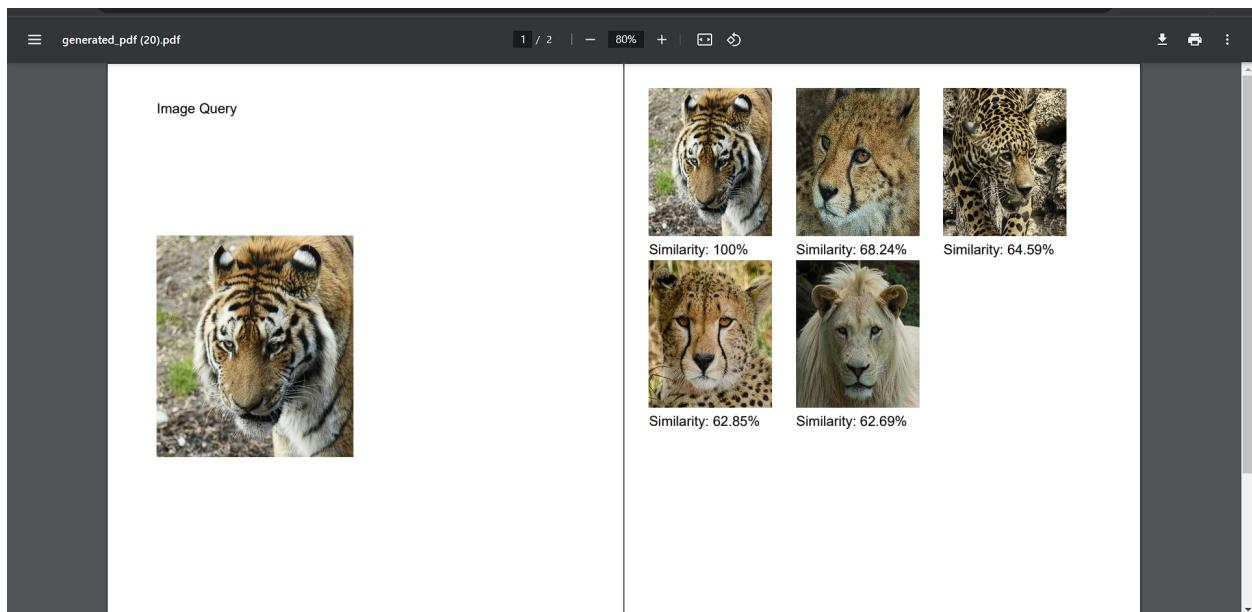
Upload Dataset Previous Next

Image Scrapping

8. Texture capture image



9. Export



4.5. Analisis Desain Solusi Algoritma

Berdasarkan hasil pengujian dapat terlihat hasil CBIR berdasarkan warna cenderung memiliki nilai similarity yang lebih beragam dibanding CBIR berdasarkan tekstur. Hal ini dapat terjadi karena warna dapat menyimpan informasi yang kaya pada suatu gambar. Keberagaman nilai similarity ini menunjukan bahwa CBIR berbasis warna mampu memberikan representasi yang lebih kompleks terhadap kesamaan antar gambar.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Kami telah berhasil membuat program temu balik gambar berbasis website pada tugas besar ini. Materi di mata kuliah IF 2123 Aljabar Linier dan Geometri yang kami terapkan pada tugas besar ini adalah Aljabar Vektor.

Dengan menerapkan materi Aljabar Vektor yang telah kami pelajari saat kuliah, kami berhasil membuat program sesuai Spesifikasi Tugas Besar 2 IF 2123 Aljabar Linier dan Geometri 2023/2024, yaitu program temu balik gambar dengan metode *Content-Based Image Retrieval* (CBIR) dengan parameter warna dan juga dengan parameter tekstur. Melalui tugas besar ini kami telah berhasil menerapkan Aljabar Vektor untuk mengoptimasi program temu balik gambar.

5.2. Saran

Pengerjaan tugas besar kami tidak lepas dari hambatan dan rintangan. Berikut beberapa saran dari kami bagi pihak yang ingin membuat program temu balik gambar seperti yang telah kami buat:

1. Kami menyarankan agar melakukan perancangan yang matang sebelum membuat program temu balik gambar. Hal ini guna mempermudah penggerjaan. Jika semua hal sudah direncanakan dengan matang, maka pihak yang terlibat bisa fokus pada pembuatan program.
2. Kami menyarankan untuk melakukan pembagian tugas yang merata, dengan bobot yang sesuai dengan kemampuan masing-masing anggota. Hal ini guna membuat penggerjaan lebih terarah bagi setiap anggota dan memastikan bahwa selalu ada yang dapat dikerjakan oleh masing-masing anggota.

5.3. Komentar dan Tanggapan

Dengan tugas besar ini kami belajar banyak tentang aplikasi materi Aljabar Vektor dalam sistem temu balik gambar. Setelah selesai membuat program, kami jadi memahami betapa pentingnya Aljabar Vektor dalam pemrosesan dan optimasi pemrosesan gambar (*image*). Tugas besar ini bukanlah hal yang mudah dilakukan ataupun hal yang cepat diselesaikan. Oleh karena itu telah tumbuh rasa apresiasi yang lebih dari kami kepada orang-orang yang telah membuat aplikasi temu balik gambar yang telah membantu banyak orang.

5.4. Refleksi

Tugas besar ini membawa kami ke dunia yang sangat relevan dan dinamis dalam era digital saat ini, yaitu pengembangan sistem temu balik gambar berbasis aljabar vektor. Melalui implementasi sistem ini dalam bentuk sebuah website, kami tidak hanya belajar

tentang konsep-konsep aljabar vektor, tetapi juga bagaimana menerapkannya secara praktis dalam pemrosesan data dan pencarian informasi.

Salah satu tantangan menarik dalam tugas ini adalah memastikan keakuratan dan kecepatan sistem temu balik gambar. Proses ini memerlukan optimasi algoritma dan penerapan konsep aljabar vektor dengan cermat. Pengaturan parameter dan evaluasi hasil menjadi langkah-langkah penting untuk meningkatkan performa sistem, sehingga memberikan pengalaman yang mendalam dalam menghadapi tantangan nyata dalam pengembangan aplikasi.

Kolaborasi dalam tim juga menjadi aspek penting dalam menyelesaikan tugas ini. Diskusi dan pertukaran ide dengan anggota tim membantu kami melihat sudut pandang yang berbeda dan menggabungkan keahlian masing-masing untuk mencapai tujuan bersama. Kemampuan untuk berkolaborasi dalam proyek-proyek seperti ini adalah keterampilan yang tidak hanya berharga dalam konteks akademis tetapi juga relevan dalam dunia kerja.

5.5. Ruang Perbaikan atau Pengembangan

Terdapat beberapa aspek yang perlu diperbaiki dalam sistem kami. Pertama, kami belum mengimplementasikan penyimpanan hasil perhitungan dataset menggunakan output file tertentu, seperti csv, fvecs, dan sebagainya. Langkah ini diharapkan dapat meningkatkan efisiensi dan efektivitas saat dataset digunakan secara berulang. Kedua, kami juga belum memanfaatkan detektor objek yang mampu melakukan pemotongan pada gambar input, sehingga hasil yang dihasilkan dapat lebih optimal.

DAFTAR PUSTAKA

“How to create a react flask project” by Miguel Grinberg

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

“RGB to HSV color conversion” from [rapidtables.com](https://www.rapidtables.com/convert/color/rgb-to-hsv.html)

<https://www.rapidtables.com/convert/color/rgb-to-hsv.html>

“Content-based image retrieval using color and texture fused features” by Jun Yue et al.

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

“Numba documentation”

<https://numba.readthedocs.io/en/stable/index.html>

“Numpy documentation”

<https://numpy.org/doc/>

“Feature Extraction : *Gray Level Co-occurrence Matrix (GLCM)*”

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

LAMPIRAN

Pranala Github:

<https://github.com/mroihn/Algeo02-22133>