

LAPORAN TUGAS BESAR 2

ALJABAR LINEAR DAN GEOMETRI

Dosen: Rinaldi Munir.



Kelompok DeleteLast
IF2123-22 Aljabar Linear dan Geometri

Daftar Anggota :
1. Mohammad Nugraha Eka Prawira
2. Ahmad Farid Mudrika
3. Zachary Samuel Tobing

SEKOLAH TEKNIK ELEKTRO
DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

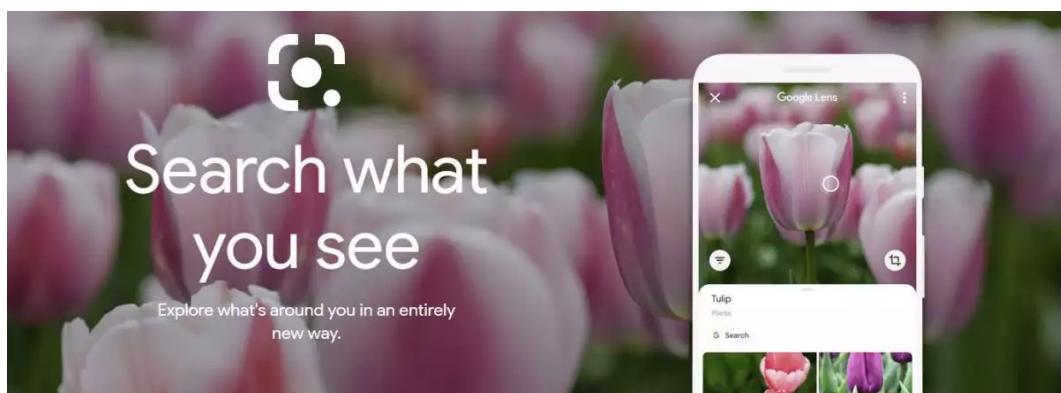
2023

BAB I

DESKRIPSI MASALAH

ABSTRAKSI

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.

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. Pada Tugas Besar kali ini, Anda diminta untuk mengimplementasikan 2 parameter CBIR yang paling populer, antara lain :

1. 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.

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. 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

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

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

$$V = C_{max}$$

Setelah mendapatkan nilai HSV lakukanlah perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity*

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

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

Untuk melakukan pencarian histogram, blok *image* dibagi menjadi $n \times n$ blok. Setiap blok akan menjadi tidak terlalu signifikan jika blok-blok tersebut terlalu besar dan akan meningkatkan waktu dalam memprosesnya jika ukuran dari blok terlalu kecil. Pada pencarian blok ini agar lebih efektif disarankan menggunakan 4×4 blok. Nyatakan nilai representatif dari sebuah blok dengan melakukan kalkulasi nilai rata-rata HSV dari blok terkait (sedikit berbeda dengan yang disampaikan pada jurnal referensi untuk menyederhanakan perhitungan. Akan tetapi, jika sudah telanjur mengimplementasikan metode pada referensi [rata-rata histogram HSV], diperbolehkan).

2. CBIR dengan parameter tekstur

CBIR dengan perbandingan tekstur dilakukan 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 $0^\circ, 45^\circ, 90^\circ$, dan 135° . Sebagai gambaran, berikut diberikan contoh cara pembuatan *co-occurrence matrix* dan [link cara pembuatannya](#) (Contoh ini dapat digunakan sebagai referensi, bukan acuan sebagai acuan utama).

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

Gambar 2. Cara Pembuatan Matrix *Occurrence*

Setelah didapat *co-occurrence matrix*, buatlah *symmetric matrix* dengan 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 . Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.
3. Dari *co-occurrence matrix* bisa diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*. Persamaan yang dapat digunakan untuk mendapatkan nilai 3 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 :

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

Keterangan : P merupakan matriks *co-occurrence*

Dari ketiga 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.

BAB II

TEORI SINGKAT

2.1 CBIR dengan parameter warna

Metode CBIR dengan parameter warna secara umum adalah metode pencocokan suatu gambar berdasarkan nilai cos dari vektor vektor histogram dari kedua gambar tersebut. Secara garis besar prosesnya adalah mengambil nilai vektor histogram dari sebuah foto dengan membagi gambar tersebut menjadi beberapa blok(Semakin banyak bloknya semakin akurat gambarnya dan semakin lama prosesnya, pada kali ini implementasi yang digunakan adalah blok 4x4). Sebelum mengambil nilai vektor histogram dari tiap blok gambar, gambar harus diolah dahulu dengan mengubah nilai RGBnya menjadi HSV dengan ketentuan pada Abstraksi. Nilai Vektor dari tiap block dihitung kemiripannya menggunakan rumus cos, lalu total nilai kemiripan itu dibagi dengan jumlah blok. Semakin nilainya mendekati 1 atau mungkin sama dengan 1, maka semakin mirip kedua gambar tersebut.

2.2 CBIR dengan parameter tekstur

CBIR dengan perbandingan tekstur dilakukan 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. Nilai RGB dari sebuah gambar diubah dulu menjadi Grayscale, lalu melakukan kuantifikasi dari nilai Grayscale tersebut. Hasil proses tersebut akan menghasilkan nilai contrast,entropy, dan homogeneity. Nilai tersebut kemudian digunakan untuk menghitung kemiripan dengan rumus cos.

2.3 Django Backend

Dalam tugas kali ini kita akan membuat sebuah website yang akan digunakan untuk melakukan proses CBIR tersebut. Oleh karena itu, dibutuhkan sebuah framework untuk mengurus backend dan frontend dari website tersebut. Terdapat banyak sekali framework yang tersedia, tetapi pada kesempatan kali ini kita memilih menggunakan Django.

2.4 Image Upload

Image upload adalah salah satu apps yang dibuat untuk menghandle upload foto dari client. Foto ataupun folder yang diupload oleh client akan diolah agar dapat digunakan dalam proses CBIR baik warna maupun texture.

2.5 React Frontend

Frontend yang digunakan adalah react. React sangat reaktif seperti namanya kita dapat membangun aplikasi kecil hingga sangat besar yang memperbarui DOM tanpa memuat ulang halaman, deklaratif dan mudah dipelajari.

2.6 Pengembangan Website

Secara umum dalam pembuatan website dibutuhkan sistem frontend dan backend. Backend adalah segala sesuatu logika yang mengurus sebuah data yang tidak terihat oleh pengguna. Data ini akan digunakan oleh frontend untuk disajikan kepada pengguna. Pengolahan data dapat berupa logika bisnis, pengolahan database, dan mengelola proses otentikasi (verifikasi identitas pengguna) dan otorisasi (pengendalian akses ke sumber daya) untuk menjaga keamanan aplikasi.

Bahasa Pemrograman: Beberapa bahasa pemrograman backend umum meliputi Python, Ruby, PHP, Java, dan JavaScript (Node.js).

Framework: Pengembang backend sering menggunakan framework seperti Django (Python), Ruby on Rails (Ruby), Laravel (PHP), dan Spring (Java). Frontend merujuk pada bagian dari aplikasi atau website yang terlihat oleh pengguna. Ini adalah antarmuka pengguna yang memungkinkan interaksi langsung dengan aplikasi atau website. Frontend bertanggung jawab dalam mengurus tampilan interface ke pengguna, interaksi pengguna, dan mengirim data ke backend.

Bahasa Pemrograman: HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), dan JavaScript adalah teknologi inti untuk pengembangan frontend.

Framework dan Library: Untuk mempermudah pengembangan, pengembang frontend sering menggunakan framework dan library seperti React, Angular, Vue.js, dan Bootstrap.

Backend dan frontend berinteraksi melalui API (Application Programming Interface). Backend menyediakan API untuk memungkinkan frontend mengakses dan mengirim data. Frontend menggunakan API ini untuk mendapatkan data dari backend dan menampilkan informasi tersebut kepada pengguna.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Langkah-langkah menggunakan website :

- 1) Pengguna mengupload foto sebagai query
- 2) Pengguna mengupload folder berisi foto sebagai dataset
- 3) Pengguna menentukan metode yang digunakan(Color atau Texture)
- 4) Hasil foto dengan kemiripan diatas 60% akan ditampilkan pada *interface* beserta waktu pengolahan.

a. Representasi Vektor:

Setiap gambar direpresentasikan sebagai vektor fitur, di mana setiap dimensi vektor mewakili atribut tertentu seperti warna atau tekstur. menciptakan ruang vektor di mana setiap gambar dapat ditempatkan.

b. Operasi Vektor:

Penerapan operasi vektor seperti penjumlahan atau pengurangan vektor untuk menggambarkan perbedaan antara dua gambar.

c. Aljabar Linier:

Konsep aljabar linier dapat diterapkan untuk manipulasi vektor dan mungkin membantu dalam pengolahan fitur vektor.

d. Pengukuran Similaritas:

Menggunakan operasi vektor seperti dot product atau jarak Euclidean untuk mengukur kesamaan atau perbedaan antara vektor fitur.

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

Temu-balik informasi (information retrieval): menemukan kembali (retrieval) informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis.

- 1) IR tidak sama dengan pencarian di dalam basis data (database)
- 2) IR umumnya digunakan pada pencarian informasi yang isinya tidak terstruktur
- 3) Informasi terstruktur: tabel-tabel di dalam basis data (database)

IR dengan Model Ruang Vektor :

- 1) Salah satu model IR adalah model ruang vektor

- 2) Model ini menggunakan teori di dalam aljabar vector
- 3) Misalkan terdapat n kata berbeda sebagai kamus kata (vocabulary) atau indeks kata (term index).
- 4) Kata-kata tersebut membentuk ruang vektor berdimensi n
- 5) Setiap dokumen maupun query dinyatakan sebagai vektor $w = (w_1, w_2, \dots, w_n)$ di dalam R^n . $\bullet w_i$ = bobot setiap kata i di dalam query atau dokumen \bullet Nilai w_i dapat menyatakan jumlah kemunculan kata tersebut dalam dokumen (term frequency)

- 1) Penentuan dokumen mana yang relevan dengan query dipandang sebagai pengukuran kesamaan (similarity measure) antara query dengan dokumen.
- 2) Semakin sama suatu vektor dokumen dengan vektor query, semakin relevan dokumen tersebut dengan query.
- 3) Kesamaan (sim) antara dua vektor $Q = (q_1, q_2, \dots, q_n)$ dan $D = (d_1, d_2, \dots, d_n)$ diukur dengan rumus *cosine similarity* yang merupakan bagian dari rumus perkalian titik (dot product) dua buah vektor: dengan $Q \cdot D$ adalah perkalian titik yang didefinisikan sebagai :
$$Q \cdot D = \|Q\| \|D\| \cos$$
- 4) Jika $\cos = 1$, berarti $= 0$, vektor Q dan D berimpit, yang berarti dokumen D sesuai dengan query Q. Jadi, nilai cosinus yang besar (mendekati 1) mengindikasikan bahwa dokumen cenderung sesuai dengan query.
- 5) Dalam hal CBIR maka nilai nilai yang akan dibandingkan adalah nilai nilai vektor yang telah diolah dalam sebuah gambar baik melalui metode color ataupun texture yang telah dibahas dalam bagian Teori Singkat.

3.3 Contoh ilustrasi kasus dan penyelesaiannya.

Misalkan terdapat sebuah gambar(P1), 1100 buah foto dataset.

Masing-masing dinyatakan sebagai vektor setelah diolah melalui salah satu metode baik color ataupun texture:

$$P1 = (2, 3, 5),$$

$$D = \{(3, 7, 1), (2, 3, 7), (3, 5, 0), \dots, (a_n, b_n, c_n)\},$$

Lalu hitung nilai cos dari tiap foto query dan tiap foto dalam dataset. Jika nilai cos nya mendekati 1, maka foto pada dataset tersebut mirip dengan query.

BAB IV

EKSPERIMEN

4.1 Pseudocode Program

1. Texture

```
IMPORT cv2
IMPORT numpy as np
DEFINE FUNCTION rgb_to_grayscale(rgb):
    RETURN np.round(0.29 * rgb[..., 0] + 0.587 * rgb[..., 1] + 0.114 *
rgb[..., 2])
DEFINE FUNCTION make_cooccurrence(path):
    SET img TO cv2.imread(path)
    SET img_rgb TO cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    SET height, width, _ TO img.shape
    SET matrix_size TO (256, 256)
    SET matrix TO np.zeros(matrix_size)
    SET grayscale_img TO rgb_to_grayscale(img_rgb)
    FOR i IN range(height):
        FOR j IN range(width - 1):
            SET x TO int(grayscale_img[i, j])
            SET y TO int(grayscale_img[i, j + 1])
            matrix[x, y] += 1
        matrix += matrix.T # Transpose in-place
        SET total TO np.sum(matrix)
        SET comatrix TO matrix / total
    RETURN comatrix
DEFINE FUNCTION contrast(comatrix):
    SET i, j TO np.ogrid[:256, :256]
    RETURN np.sum(comatrix * ((i - j) ** 2))
DEFINE FUNCTION homogeneity(comatrix):
    SET i, j TO np.ogrid[:256, :256]
    RETURN np.sum(comatrix / (1 + (i - j) ** 2))
DEFINE FUNCTION entropy(comatrix):
    SET matrix TO comatrix[comatrix!=0]
    RETURN -np.sum(matrix * np.log(matrix))
DEFINE FUNCTION cosine_similarity(vect1, vect2):
    SET dot_product TO np.dot(vect1, vect2)
    SET norm_vect1 TO np.linalg.norm(vect1)
    SET norm_vect2 TO np.linalg.norm(vect2)
```

```

    SET similarity TO dot_product / (norm_vect1 * norm_vect2)
    RETURN similarity
DEFINE FUNCTION makevector(path):
    SET comatrix TO make_cooccurrence(path)
    SET vector TO np.array([contrast(comatrix), homogeneity(comatrix),
entropy(comatrix)])
    RETURN vector
SET vect1 TO makevector(path)
SET vect2 TO makevector(path2)
OUTPUT(cosine_similarity(vect1, vect2))

```

2. Color

```

IMPORT cv2
IMPORT numpy as np
DEFINE FUNCTION calculate_histogram(image_matrix):
    SET # start TO time.time()
    SET (height, width, num_channels) TO image_matrix.shape
    # Inisialisasi vektor histogram
    SET histogram_vector TO np.zeros((16,48), dtype=float)
    # Mendefinisikan blok 4x4
    SET block_size TO 4
    SET num_blocks_height TO height // block_size
    SET num_blocks_width TO width // block_size
    # Iterasi melalui blok-blok
    x=0
    FOR i IN range(block_size):
        FOR j IN range(block_size):
            # Batas blok
            SET block_start_row TO i * num_blocks_height
            SET block_end_row TO (i + 1) * num_blocks_height
            SET block_start_col TO j * num_blocks_width
            SET block_end_col TO (j + 1) * num_blocks_width
            # Mendapatkan blok citra
            SET block TO image_matrix[block_start_row : block_end_row,
block_start_col:block_end_col]
            SET Hue TO block[:, :, 0].flatten()
            SET Sat TO block[:, :, 1].flatten()
            SET Val TO block[:, :, 2].flatten()
            SET block_histogram_hue, TO np.histogram(Hue, bins=16,
range=(0, 180))

```

```

        SET block_histogram_sat, TO np.histogram(Sat, bins=16, range=
(0, 1))
        SET block_histogram_val, TO np.histogram(Val, bins=16, range=
(0, 1))
        IF np.linalg.norm(block_histogram_hue) !=0:
            SET block_histogram_hue TO
block_histogram_hue/np.linalg.norm(block_histogram_hue)
        IF np.linalg.norm(block_histogram_sat) !=0:
            SET block_histogram_sat TO
block_histogram_sat/np.linalg.norm(block_histogram_sat)
        IF np.linalg.norm(block_histogram_val) !=0:
            SET block_histogram_val TO
block_histogram_val/np.linalg.norm(block_histogram_val)
        # OUTPUT(f'Hasil block histogram:
{np.array(block_histogram_hue)}')
        SET histogram_vector[x] TO
np.concatenate([block_histogram_hue.flatten(),
block_histogram_sat.flatten(), block_histogram_val.flatten()])
        # OUTPUT(f'Hasil block histogram: {histogram_vector}')
        x += 1
# end= time.time()
# OUTPUT(f'Lama HISTOGRAM: {end-start}')
RETURN histogram_vector
DEFINE FUNCTION calculate_cosine_similarity(hist1, hist2):
    # Flatten histograms
    result =0;
    FOR i IN range(16):
        SET dot_product TO np.dot(hist1[i],hist2[i])
        SET normal_a TO np.linalg.norm(hist1[i])
        SET normal_b TO np.linalg.norm(hist2[i])
        result += dot_product/(normal_a*normal_b)
    result/=16.0
    RETURN result
DEFINE FUNCTION rgb_to_hsv(rgbimage_path):
    # Baca gambar
    SET rgbimage TO cv2.imread(rgbimage_path)
    # Normalisasi nilai RGB
    SET normalized_image TO rgbimage / 255.0
    # Mendapatkan nilai-nilai RGB
    SET red TO normalized_image[:, :, 2]

```

```

SET green TO normalized_image[:, :, 1]
SET blue TO normalized_image[:, :, 0]
# Mendapatkan nilai maksimum dan minimum dari RGB
SET max_rgb TO np.maximum(red, np.maximum(green, blue))
SET min_rgb TO np.minimum(red, np.minimum(green, blue))#
Menghitung delta
SET delta TO max_rgb - min_rgb
# Mencari nilai hue
SET hue TO np.where(delta EQUALS 0, 0,
    60 * np.where(max_rgb EQUALS red, (green - blue) / (delta +
1e-9) % 6,
        np.where(max_rgb EQUALS green, (blue - red) / (delta +
+ 1e-9) + 2,
            (red - green) / (delta + 1e-9) + 4)))
# Menghitung nilai saturation
SET saturation TO np.where(max_rgb EQUALS 0, 0, delta / (max_rgb +
+ 1e-9))
# Menghitung nilai value
SET value TO max_rgb
# Menggabungkan nilai H, S, V ke dalam satu array
SET hsv_image TO np.stack((hue, saturation, value), axis=-1)
RETURN hsv_image
SET hist1 TO calculate_histogram(rgb_to_hsv(path))
SET hist2 TO calculate_histogram(rgb_to_hsv(path2))
SET hasil TO calculate_cosine_similarity(hist1, hist2)

```

4.2 Struktur Program

I. Backend

Dalam pengembangan backend website kami, kami menggunakan framework Django. Dalam implementasinya, kami membuat beberapa app untuk menghandle request user, yaitu:

1) colors

App colors dibuat untuk menghandle CBIR berbasis color

2) imageupload

App imageupload dibuat untuk menghandle upload image dan dataset oleh user. Image-image ini disimpan di folder media dan nanti diproses oleh colors dan texture.

3) texture

App texture dibuat untuk menghandle CBIR berbasis texture. Dalam implementasi matriks cooccurrence, kami memakai sudut teta 0° . Ini disebabkan beberapa hal, yaitu:

- a. Jumlah pixel yang tidak memiliki pasangan pada teta 0° adalah `height`(pixel vertikal), dan pada teta 90° adalah `width`(pixel horizontal). Kebanyakan gambar berada dalam posisi landscape, sehingga cooccurrence matrix akan lebih akurat (lebih sedikit pixel yang tidak memiliki pasangan) dari teta 90° .
- b. Dengan alasan yang sama, kami tidak menggunakan sudut teta diagonal (45° dan 135°) karena jumlah pixel yang tidak memiliki pasangan adalah `height+width-1`.

II. Frontend

Dalam pengembangan frontend website kami, kami menggunakan tools react. Dalam frontend, kami membuat pages home, howtouse, aboutus, dan briefconcept. Di home, kami mengimplementasikan program backend kami untuk menampilkan gambar-gambar hasil pencarian.

4.3 Tata Cara Penggunaan Program

Program utama dapat dijalankan dengan langkah berikut:

1. Pengguna memasukkan sebuah gambar yang ingin di-*search* dari dataset dengan menekan tombol *InsertImage* dan memilih fotonya di folder local.
2. Pengguna kemudian memasukkan dataset gambar dalam bentuk folder yang berisi kumpulan gambar dengan menekan tombol *UploadDataset*. Dataset gambar ini diperlukan sebelum proses searching agar ada perbandingan untuk gambar yang ingin dicari.
3. Pengguna kemudian dapat memilih untuk melakukan pencarian berdasarkan parameter warna atau tekstur dengan menekan *toggle* pada sisi yang diinginkan.
4. Pengguna kemudian memilih tombol *Search* untuk melakukan pencarian berdasarkan warna atau tekstur sesuai pilihan sebelumnya.
5. Program kemudian akan memproses, mencari gambar-gambar dari dataset yang memiliki kemiripan dengan gambar yang dimasukkan tadi.

6. Program akan menampilkan kumpulan gambar yang mirip, diurutkan dari yang memiliki kemiripan paling tinggi ke yang paling rendah. Setiap gambar yang muncul diberi persentase kemiripannya.
7. Pengguna dapat memilih *Previous* dan *Next* sesuai kumpulan gambar yang ada untuk melihat semua hasil pencarian.

Kemudian, terdapat pilihan *page* lain yang berisi informasi singkat tentang konsep yang digunakan, cara pemakaian singkat, dan sedikit informasi tentang kelompok kami.

4.4 Screenshot website



Gambar 4.1. Gambar homepage sebelum memulai search



Gambar 4.2. Gambar page Brief Concept

Home **Brief Concept** **How to Use** **About Us**

How to Use

1. Masukkan gambar yang mau diproses.
2. Pilih untuk membandingkan berdasarkan warna atau tekstur.
3. Upload dataset yang ingin dibandingkan dengan klik tombol di bawah.
4. Klik tombol "Search"
5. Gambar akan tersedia secara terurut dari yang paling mirip. Silahkan pilih pada angka-angka selanjutnya untuk melihat semua gambarnya.

Gambar 4.3. Gambar page How to Use

Home **Brief Concept** **How to Use** **About Us**

About Us

Website ini dibuat oleh kelompok DeleteLast yang beranggotakan Muhammad Nugraha Eka Prawira (13522001), Ahmad Farid Mudrika (13522008), dan Zachary Samuel Tobing (13522016) untuk penyelesaian Tugas Besar Aljabar Linear dan Geometri.



Gambar 4.4. Gambar page About Us

Home **Brief Concept** **How to Use** **About Us**



Percentase: 100.0%



Percentase: 99.99959155344284%



Percentase: 99.99954193083313%



Percentase: 99.99945506363466%



Percentase: 99.99934982292665%



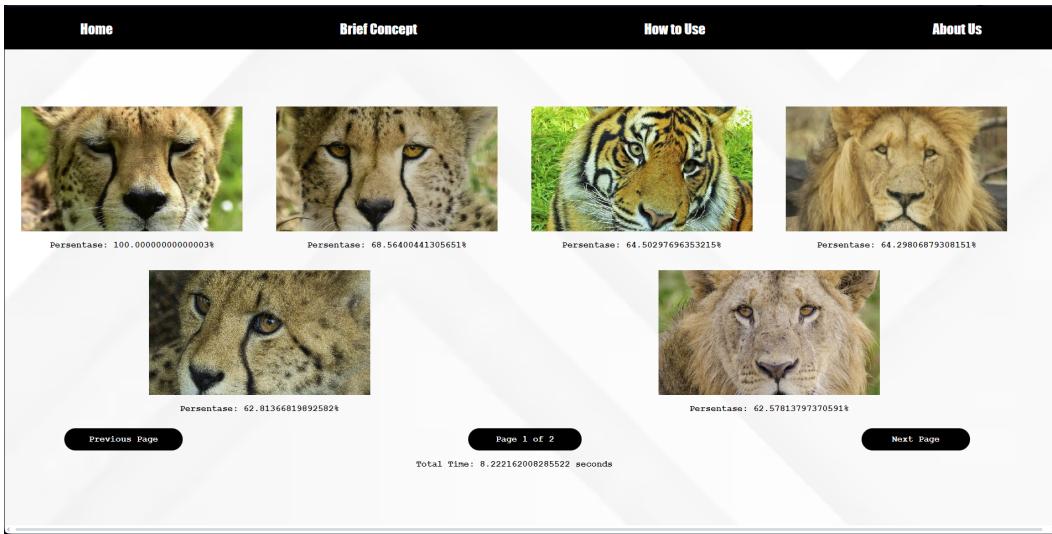
Percentase: 99.99878810097422%

Previous Page **Page 1 of 17** **Next Page**

Total Time: 14.64447832107544 seconds

Gambar 4.5. Contoh hasil search berdasarkan texture

Dapat dilihat gambar 3 memiliki persentase yang tinggi, hal ini karena gambar search(gambar 1) terlihat mirip dengan gambar 3 ketika diubah ke grayscale.



Gambar 4.5. Contoh hasil pencarian berdasarkan warna

Dapat dilihat bahwa gambar-gambar yang memiliki hasil pencarian tinggi adalah gambar dengan warna yang mirip dengan base image.

4.5 Analisa Hasil

Berdasarkan hasil yang didapat, dengan menggunakan dataset yang diberikan, CBIR dengan parameter warna memiliki hasil yang cukup bervariatif, sedangkan pada parameter tekstur, persentase kemiripan terlihat sangat tinggi. Bahkan, dari sekitar 4700 gambar dataset yang diberikan, tingkat kemiripan terendah dari CBIR parameter tekstur adalah 70%. Ini berarti untuk dataset yang diberikan, CBIR dengan parameter warna dapat lebih diandalkan.

BAB V

PENUTUP

5.1 Kesimpulan

Program yang kami buat dapat menghitung kemiripan dari 2 gambar berbeda, dengan menggunakan prinsip cosine similarity. Dari 2 gambar tersebut, dibentuk vektor yang berisi hue, saturation, dan value jika memilih pencarian berdasar warna, atau *contrast*, *homogeneity*, dan *entropy* jika memilih pencarian berdasar tekstur.

5.2 Saran

5.2.1 Menyertakan hasil yang diinginkan pada studi kasus tugas besar.

5.3 Komentar & Refleksi

Setelah mengerjakan tugas besar ini, pemahaman kami mengenai materi-materi yang dipelajari sepanjang kelas Aljabar Linier dan Geometri sangat meningkat. Tugas besar ini juga sangat membantu kami dalam memahami pengembangan website. Untuk kedepannya kami rasa manajemen waktu pengerjaan tugas besar dapat ditingkatkan lagi agar tidak mepet dengan deadline..

DAFTAR REFERENSI

<https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1>
<https://www.sciencedirect.com/science/article/pii/S0895717710005352#s000030>

LINK VIDEO DAN GITHUB REPOSITORY

<https://youtu.be/G0ifxfKrEvY?si=EbixYSjV1Xgk2NJt>
<https://github.com/EkaaPrawiraA/Algeo02-22001>