

Laporan Tugas Besar 2 IF2123 Aljabar Linier dan Geometri

Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar

Semester I Tahun 2023/2024



oleh

1. Maximilian Sulistiyo 13522061
2. Evelyn Yosiana 13522083
3. Andhika Tantyo Anugrah 13522094

Kelompok

Algeo Lens

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

Daftar Isi.....	2
BAB 1	
Deskripsi Persoalan.....	3
BAB 2	
Landasan Teori.....	4
BAB 3	
Analisis Pemecahan Masalah.....	12
BAB 4	
Implementasi dan Uji Coba.....	21
BAB 5	
Kesimpulan, Saran, Komentar, dan Refleksi.....	41
Daftar Pustaka.....	42

BAB 1

Deskripsi Persoalan

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 1.1.1 Contoh penerapan *information retrieval system* (Google Lens)

Di dalam Tugas Besar 2 ini, kami diminta untuk mengimplementasikan sistem temu balik gambar yang sudah dijelaskan sebelumnya dengan memanfaatkan Aljabar Vektor dalam bentuk sebuah website, dimana hal ini merupakan pendekatan yang penting dalam dunia pemrosesan data dan pencarian informasi. Dalam konteks ini, aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (*Content-Based Image Retrieval* atau CBIR), di mana sistem temu balik gambar bekerja dengan mengidentifikasi gambar berdasarkan konten visualnya, seperti warna dan tekstur.

BAB 2

Landasan Teori

Content-Based Image Retrieval (CBIR) adalah proses pengambilan konten dari suatu gambar. Konten ini nantinya akan di-*query* dan dibandingkan dengan konten pada gambar lain. CBIR dimulai dengan ekstraksi fitur-fitur penting dari gambar, seperti warna dan tekstur. CBIR digunakan karena dataset gambar tradisional yang hanya bergantung pada anotasi yang dibuat secara manual dan tidak akurat oleh manusia. Untuk mengimplementasikan CBIR, sistem perlu mengerti dan bisa menginterpretasikan konten dari gambar yang diberikan.

CBIR mengacu pada konten gambar yang diambil langsung fitur-fiturnya. Lalu ditaruh ke database gambar. Ide dasar dari CBIR adalah membandingkan informasi gambar yang *low-level* dengan gambar yang ingin dicari kemiripannya. Setelah fitur-fitur tersebut diekstraksi, fitur-fitur itu dijadikan elemen sebuah vektor agar dapat dibandingkan dengan gambar lain. Kemudian, CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor yang berisi fitur-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 butuh pencarian berdasarkan teks atau kata kunci (anotasi yang sebelumnya telah ditulis secara manual oleh manusia lain), melainkan berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut. Pada tugas besar ini, kami diperintahkan untuk merealisasikan 2 jenis CBIR, yaitu CBIR dengan parameter warna dan CBIR dengan parameter tekstur.

1. CBIR dengan parameter warna

Pada CBIR warna, masukan dari sebuah gambar akan dibandingkan dengan gambar yang ada pada dataset. Proses pembandingan ini dilakukan dengan cara mengubah gambar yang bisa disebut sebuah matriks RGB menjadi sebuah histogram warna yang lebih umum.

Histogram warna adalah frekuensi dari beberapa warna yang ada pada ruang warna tertentu (pada kasus ini RGB). Histogram ini dibuat untuk melihat hasil distribusi warna dari gambar. Kelemahan dari metode CBIR warna ini adalah CBIR tidak mampu mendeteksi objek spesifik yang terdapat pada gambar dan tidak mampu mengetahui posisi dari warna yang sudah digeneralisasi.

Pembentukan ruang warna perlu dilakukan untuk menyebarluaskan 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 pixel 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. Oleh karena itu, perlu dilakukan konversi warna dari RGB ke HSV dengan prosedur sebagai berikut.

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

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

Gambar 2.1.1 Normalisasi RGB

- Mencari C_{max} , C_{min} , dan Δ

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

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

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

Gambar 2.1.2 Cmax dan Cmin

- Selanjutnya gunakan hasil perhitungan di atas untuk mendapatkan nilai HSV

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

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

$$V = C_{max}$$

Gambar 2.1.3 Cmax dan Cmin

Setelah mendapatkan nilai HSV dilakukan perbandingan antara gambar dari input dengan gambar pada dataset dengan menggunakan *cosine similarity*:

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

Gambar 2.1.4 Cosine Similarity

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 menggunakan 4×4 blok. Untuk menyatakan nilai representatif dari sebuah blok, dilakukan dengan melakukan kalkulasi nilai rata-rata HSV dari blok terkait.

2. CBIR dengan parameter tekstur

CBIR dengan perbandingan tekstur dilakukan menggunakan matriks yang bernama *Gray Level Co-occurrence Matrix (GLCM)*. Matriks ini digunakan karena dapat melakukan pemrosesan yang lebih mudah dan cepat, karena tidak seperti CBIR warna yang harus menyimpan 3 warna, GLCM hanya perlu menyimpan 1 warna saja, yaitu abu-abu. Tidak hanya itu, vektor fitur yang dihasilkan juga mempunyai ukuran yang lebih kecil.

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.

$$C_{\Delta x, \Delta y}(i, j) = \sum_{x=1}^n \sum_{y=1}^m \begin{cases} 1, & \text{if } I(x, y) = i \text{ and } I(x + \Delta x, y + \Delta y) = j \\ 0, & \text{otherwise} \end{cases}$$

Gambar 2.2.1 Offset

Melalui persamaan tersebut, digunakan nilai $\theta = 0^\circ$ untuk mempermudah perhitungan.

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

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

Gambar 2.2.2 Pembuatan Matriks Co-occurrence

Setelah didapat *Gray Level Co-occurrence Matrix (GLCM)*, dibuat *symmetric matrix* dengan menjumlahkan *GLCM* dengan hasil *transpose*-nya. Lalu dicari normal matriksnya dengan persamaan.

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

Gambar 2.2.3 Matriks Norm

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

- Pertama, warna gambar dikonversi menjadi *grayscale* karena warna tidak penting dalam penentuan tekstur. Oleh karena itu, warna RGB diubah menjadi suatu warna *grayscale* Y'_{601} dengan rumus:

$$Y'_{601} = 0.299R' + 0.587G' + 0.114B'$$

Gambar 2.2.4 Konversi Gambar Menjadi Grayscale

- Setelah itu dilakukan kuantifikasi dari nilai *grayscale* dan membuat *GLCM*-nya. Karena nilai minimum dan maksimum dari sebuah citra *grayscale* adalah 0 dan 255, maka GLCM akan berukuran 256×256 . Tingkat kemiripan dari gambar dinilai berdasarkan kekasaran (perbedaan tingkat keabu-abuan dari suatu pixel dengan tetangganya) tekstur dari gambar tersebut.

3. Dari *GLCM* yang sudah dinormalkan, dapat diperoleh 7 komponen ekstraksi tekstur, yaitu *contrast*, *energy*, *homogeneity*, *correlation*, *entropy*, *shade*, dan *prominence*. Persamaan yang kami gunakan untuk mendapatkan nilai 7 komponen tersebut adalah:

Contrast:

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

Gambar 2.2.5 Contrast

Homogeneity:

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

Gambar 2.2.6 Homogeneity

Entropy:

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

Gambar 2.2.7 Entropy

Energy:

$$\text{Energy} = \sum \sum q(i,j)^2$$

Gambar 2.2.8 Energy

Correlation (C):

$$\text{Correlation} = \sum_{i,j=0}^{N-1} P_{ij} \frac{(i - \mu)(j - \mu)}{\sigma^2}$$

Gambar 2.2.9 Correlation

Shade:

$$\text{Shade} = \text{sgn}(A)|A|^{1/3}$$

Gambar 2.2.10 Shade

Prominence:

$$\text{Prominence} = \text{sgn}(B)|B|^{1/4}$$

Gambar 2.2.11 Prominence

Mean (μ):

$$\mu = \sum_{i,j=0}^{N-1} i P_{ij}$$

Gambar 2.2.12 Mean

Variance:

$$\sigma^2 = \sum_{i,j=0}^{N-1} P_{ij} (i - \mu)^2$$

Gambar 2.2.13 Variance

A:

$$\sum_{i,j=0}^{N-1} \frac{(i+j-2\mu)^3 P_{ij}}{\sigma^3 (\sqrt{2(1+C)})^3}$$

Gambar 2.2.13 Variance A

B:

$$\sum_{i,j=0}^{N-1} \frac{(i+j-2\mu)^4 P_{ij}}{4\sigma^4 (1+C)^2}$$

Gambar 2.2.14 Variance B

Keterangan : P merupakan GLCM yang sudah dinormalisasi

Dari ketujuh komponen tersebut (*contrast, energy, homogeneity, correlation, entropy, shade, dan prominence*), dibuatlah sebuah vektor yang akan digunakan dalam proses pengukuran tingkat kemiripan suatu gambar dengan gambar-gambar lain.

4. Sama seperti pada perhitungan CBIR warna, kemiripan antara 2 gambar dihitung dengan menggunakan Teorema *Cosine Similarity*:

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

Gambar 2.2.15 Cosine Similarity

Mengembangkan Sebuah Laman Web

Pengembangan laman web biasanya mengacu pada pembuatan pilar utama dalam laman web, *markup* dan *coding* yang biasanya tidak ada proses desain. Kakas untuk membuat laman web mencakupi bahasa *markup*, *Cascading Style Sheets* (CSS), bahasa untuk *scripting*, aplikasi pemrosesan teks, aplikasi pemrosesan HTML, kakas pengembangan web, *template* web, dan *content management systems* (CMS).

Bahasa *markup* adalah sebuah sistem *coding* yang menggunakan *tags* untuk menyediakan instruksi mengenai penampakan, struktur, dan format dari sebuah dokumen. Bahasa *markup* yang biasanya digunakan untuk membuat laman web adalah HTML, XML, dan XHTML.

Cascading Style Sheet (CSS) adalah dokumen yang menggunakan peraturan untuk standarisasi penampakan dari konten laman *web* dengan mendefinisikan gaya untuk elemen seperti font, margin, posisi, warna latar, dan lain-lain.

Bahasa *scripting* adalah bahasa pemrograman yang digunakan untuk menulis program pendek yang biasa dipanggil *scripts*, berjalan dengan waktu nyata di *server* atau penjelajah *web* ketika laman *web* diunduh. *Scripts* membuat laman *web* lebih dinamis dan interaktif dengan menambahkan fitur seperti multimedia, animasi, dan formulir, atau dengan menghubungi *database*. Contoh dari bahasa *scripting* adalah JavaScript, PHP, dan CoffeScript.

Contoh *editor* teks yang bisa digunakan adalah Notepad, Sublime Text, VS Code, dan lain-lain. Jika ingin menggunakan kakas pengembangan *web* profesional, bisa menggunakan *Integrated Development Environment* (IDE), seperti Microsoft® Visual Studio Community, WebStorm, atau Eclipse, yang menyediakan desain laman *web* yang kompleks, *publishing*, dan kapabilitas manajemen.

BAB 3

Analisis Pemecahan Masalah

3.1. Algoritma CBIR

Masalah pertama yang harus diselesaikan adalah mengubah sebuah gambar menjadi matriks RGB. Hal ini dapat dengan mudah dilakukan di bahasa python dengan menggunakan pustaka *open-cv* yang akan mengubah gambar menjadi matriks BGR. Menggunakan pustaka yang sama, kami ubah matriks BGR menjadi matriks RGB. Matriks RGB ini bisa digunakan untuk diproses menjadi HSV, pemrosesan warna, dan diproses menjadi matriks *grayscale* untuk pemrosesan tekstur.

Dalam permasalahan CBIR warna, pertama kami harus merubah matriks RGB menjadi matriks HSV. Untuk pengubahan dari matriks RGB ke HSV, kami menggunakan manipulasi matriks menggunakan numpy karena numpy sendiri ditulis dalam bahasa c sehingga manipulasi matriks pun jauh lebih cepat dibandingkan looping di python. Kemudian untuk konversi ke HSV kami menggunakan operasi np.where dimana setiap elemen dalam matriks HSV akan diperiksa elemennya dan diubah sesuai dengan rumus konversi RGB ke HSV. Kemudian kami harus membuat histogram dari HSV matriks yang dimiliki. Untuk ini kita menggunakan library cv2 lagi untuk mengkalkulasikan histogram dari matrix of HSV. Kemudian untuk meningkatkan akurasi disini sebuah image akan diubah menjadi 4x4 sehingga terdapat 16 segment, setiap segment dari matriks itu pun dibuat histogramnya. Untuk mencari similarity maka kita membandingkan vektor histogram tiap segmen dengan segmen yang sama di gambar lain. Kemudian kita rata-rata kan hasil tersebut untuk mendapatkan hasil akhir.

Dalam permasalahan CBIR tekstur, pertama kami harus mengubah matriks RGB menjadi matriks *grayscale* dengan menggunakan rumus Y_{601} di atas. Kemudian membentuk *GLCM* yang memiliki kecepatan $O(N^2)$. Setelah itu, membuat normal dari *GLCM* yang bisa digunakan untuk menghitung semua parameter yang telah direncanakan sebelumnya. Dengan hasil dari rumus fitur-fitur yang ada pada landasan teori bagian CBIR tekstur, maka dapat dibentuk sebuah vektor dengan fitur-fitur tersebut sebagai elemennya dan dibandingkan dengan vektor dari gambar lain dengan *cosine similarity*.

Sekarang untuk permasalahan caching. Kita menganggap caching sangat penting sehingga setiap dataset yang diinput akan di cache dalam bentuk json file. Isi dari json file itu sendiri berbeda antara colour dan texture dimana jika dataset diproses menggunakan colour maka dalam json colour terdapat path tiap image dan juga semua histogram semua segment sedangkan dalam json texture terdapat path tiap image dan juga semua feature yang telah diambil dari GLCM.

3.2. Daftar Fungsi dan Library

Library yang kami gunakan antara lain:

- numpy : manipulasi matriks dan mempercepat operasi matriks;
- cv2 : membaca gambar dan mengubahnya menjadi matriks;
- werkzeug : menangani routing;
- os : manajemen file dan manipulasi path;
- shutil : menggandakan file dan menghapus directory;
- json : mentransfer data;
- time : menghitung runtime;
- fpdf : menghasilkan dan mengedit pdf;
- pillow : memanipulasi gambar;
- zipfile : memanipulasi file ZIP;
- tempfile : membuat dan menggunakan *directory* sementara;
- multiprocessing : mendukung pemrograman paralel dan konkuren;
- decimal : melakukan operasi aritmatika dengan presisi;
- warnings : menghilangkan warning di terminal.

Daftar function:

3.2.1. ImageToMatrix

- Create_histogram: mengubah matriks of HSV menjadi histogram.

```
def ImageToMatrix(path):
    img_bgr = cv2.imread(path)
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB).astype('float32')
    return img_rgb
```

Gambar 3.2.1 Fungsi Image To Matrix

- RGBtoHSV: Mengubah matriks RGB menjadi matriks HSV.

```

def RGBtoHSV(rgb_matrix):
    rgb_normalized = rgb_matrix / 255.0
    r, g, b = rgb_normalized[:, :, 0], rgb_normalized[:, :, 1], rgb_normalized[:, :, 2]

    Cmax = np.max(rgb_normalized, axis=-1)
    Cmin = np.min(rgb_normalized, axis=-1)
    Delta = Cmax - Cmin

    h = np.where(Delta == 0, 0,
                 np.where(Cmax == r, 60 * ((g - b) / Delta % 6), #Kasus Cmax = r
                           np.where(Cmax == g, 60 * ((b - r) / Delta + 2), #Kasus Cmax = g
                                   60 * ((r - g) / Delta + 4))) #Kasus Cmax = b

    s = np.where(Cmax == 0, 0, Delta / Cmax)
    v = Cmax

    hsv_matrix = np.dstack((h, s, v))

    return hsv_matrix

```

Gambar 3.2.2 Fungsi RGB To HSV

3.2.2. CBIR_Colour

- create_histogram: Menggunakan library cv2 kita mencari histogram dari image. Sebelum histogram diambil kita membagi image menjadi 4x4 yang berarti image akan menjadi 16 segment. Setelah itu kita akan mengkalkulasikan histogram tiap segmen yang kemudian akan di append menjadi satu list dan di return kan.

```

def create_histogram(img_path):
    hsv_image = imt.RGBtoHSV(imt.ImageToMatrix(img_path))

    height, width = hsv_image.shape[:2]
    segment_height = height // 4
    segment_width = width // 4

    hist_per_segments = []
    for i in range(4):
        for j in range(4):
            segment = hsv_image[i*segment_height:(i+1)*segment_height,
                                 j*segment_width:(j+1)*segment_width]
            hist_hue = cv2.calcHist([segment], [0], None, [8], [0, 360]).flatten()
            hist_saturation = cv2.calcHist([segment], [1], None, [3], [0, 1]).flatten()
            hist_value = cv2.calcHist([segment], [2], None, [3], [0, 1]).flatten()
            hist_combined = np.concatenate((hist_hue, hist_saturation, hist_value))
            hist_per_segments.append(hist_combined.tolist())

    return hist_per_segments

```

Gambar 3.2.3 Fungsi Create Histogram

- process_colour_image: Fungsi ini membantu konversi dari dataset menjadi histogram pada function DatasetToColourJSON.

```
def process_colour_image(filename, folder_path):
    img_path = os.path.join(folder_path, filename)
    hist_vector = create_histogram(img_path)
    return {'path': filename, 'histograms': hist_vector}
```

Gambar 3.2.4 Fungsi Process Color Image

- DatasetToColourJSON: Function ini akan merubah sebuah folder yang berisi gambar dan merubahnya menjadi JSON object yang terdiri dari path dan histogram dari tiap image. Function ini menggunakan multiprocessing.

```
def DatasetToColourJSON(folder_path, json_path):
    filenames = os.listdir(folder_path)
    data_list = []

    with Pool(processes=os.cpu_count()) as pool:
        results = pool.starmap(process_colour_image, [(filename, folder_path) for filename in filenames])

    data_list.extend(results)

    with open(json_path, 'w') as json_file:
        json.dump(data_list, json_file, indent=4)
```

Gambar 3.2.5 Fungsi Dataset To Color JSON

- SimilarityColour: Prosedur ini menerima path gambar berserta JSON dataset colour dan kemudian akan membandingkan histogram per segmen dan mengambil rata-rata dari 16 segmen. Kemudian prosedur ini akan menyaring hasil yang dibawah 60% dan kemudian menulis hasil ke dalam sebuah JSON similarity.

```
def SimilarityColour(img_path, colour_json_path, colour_similar_json_path):
    data = []
    with open(colour_json_path) as file:
        data = json.load(file)

    image_hists = create_histogram(img_path)

    similarities = []
    for img_dataset in data:
        segment_similarities = []
        for segment_hist_uploaded, segment_hist_dataset in zip(image_hists, img_dataset['histograms']):
            similarity = cs.cosine_similarity(segment_hist_uploaded, segment_hist_dataset)
            segment_similarities.append(similarity)

        avg = np.mean(segment_similarities)
        if avg >= 0.6:
            similarities.append({'path': img_dataset['path'], 'similarity': avg})

    similarities = sorted(similarities, key=lambda x: x['similarity'], reverse=True)

    with open(colour_similar_json_path, 'w') as file:
        json.dump(similarities, file, indent=4)
```

Gambar 3.2.6 Similarity Color

3.2.3. CBIR_Texture

- img2GRY: mengubah nilai RGB menjadi *grayscale* menggunakan koefisien Rec. 601 luma (Y_{60I}).

```
def img2GRY(path):
    img_rgb = imt.ImageToMatrix(path)
    gray = img_rgb[:, :, 0] * 0.299 + img_rgb[:, :, 1] * 0.587 + img_rgb[:, :, 2] * 0.114 # Y = 0.299 * R + 0.587 * G + 0.114 * B
    resize = cv2.resize(gray, None, fx = 0.4, fy = 0.4) # Resize gambar agar mudah diproses
    return resize
```

Gambar 3.2.7 Fungsi RGB to Grayscale

- glcm: membentuk sebuah matriks *Gray Level Co-occurrence Matrix*.

```
def glcm(path):
    img = img2GRY(path)
    quantized = np.round(img).astype(np.int32)
    max_val = quantized.max() + 1
    glcm_matrix = np.zeros((max_val, max_val), dtype=np.int32)
    for i in range(quantized.shape[0] - 1):
        for j in range(quantized.shape[1] - 1):
            glcm_matrix[quantized[i, j], quantized[i, j + 1]] += 1
            glcm_matrix[quantized[i, j + 1], quantized[i, j]] += 1
    return glcm_matrix
```

Gambar 3.2.8 Fungsi GLCM

- glcmNorm: mengembalikan normal dari GLCM.

```
def glcmNorm(glcm_matrix):
    return glcm_matrix / glcm_matrix.sum()
```

Gambar 3.2.9 Fungsi GLCM Norm

- glcmMean: mengembalikan parameter mean yang akan digunakan untuk menghitung fitur di fungsi yang lebih kompleks.

```
def glcmMean(glcm_norm):
    # https://support.echoview.com/WebHelp/Windows_And_Dialog_Boxes/Dialog_Boxes/Variable_Properties_Dialog_Box/Operator_Pages/GLCM_Texture_Features.htm
    i, j = np.indices(glcm_norm.shape)
    return np.sum(i * glcm_norm)
```

Gambar 3.2.10 Fungsi GLCM Mean

- glcmVariance: mengembalikan nilai varians yang akan digunakan untuk menghitung fitur di fungsi yang lebih kompleks.

```
def glcmVariance(glcm_norm):
    # https://support.echoview.com/WebHelp/Windows_And_Dialog_Boxes/Dialog_Boxes/Variable_Properties_Dialog_Box/
    # Operator_Pages/GLCM_Texture_Features.htm
    i, j = np.indices(glcm_norm.shape)
    mean = glcmMean(glcm_norm)
    return np.sum(glcm_norm * (i - mean) ** 2)
```

Gambar 3.2.11 Fungsi GLCM Variance

- contrast: mengembalikan komponen fitur yang akan digunakan nanti.

```
def contrast(glcm_norm):
    i, j = np.indices(glcm_norm.shape)
    return np.sum(glcm_norm * (i - j) ** 2)
```

Gambar 3.2.11 Fungsi Contrast

- energy: mengembalikan komponen fitur yang akan digunakan nanti.

```
def energy(glcm_norm):
    # https://iopscience.iop.org/article/10.1088/1742-6596/1591/1/012028/pdf
    return np.sum(glcm_norm ** 2)
```

Gambar 3.2.12 Fungsi Energy

- homogeneity: mengembalikan komponen fitur yang akan digunakan nanti.

```
def homogeneity(glcm_norm):
    i, j = np.indices(glcm_norm.shape)
    return np.sum(glcm_norm / (1 + (i - j) ** 2))
```

Gambar 3.2.13 Fungsi Homogeneity

- correlation: mengembalikan komponen fitur yang akan digunakan nanti.

```
def correlation(glcm_norm):
    i, j = np.indices(glcm_norm.shape)
    mean = glcmMean(glcm_norm)
    variance = glcmVariance(glcm_norm)
    return np.sum(glcm_norm * ((i - mean) * (j - mean)) / variance)
```

Gambar 3.2.14 Fungsi Correlation

- entropy: mengembalikan komponen fitur yang akan digunakan nanti.

```
def entropy(glcm_norm):  
    mask = glcm_norm > 0  
    return -np.sum(glcm_norm[mask] * np.log10(glcm_norm[mask]))
```

Gambar 3.2.15 Fungsi Entropy

- shade: mengembalikan komponen fitur yang akan digunakan nanti.

```
def shade(glcm_norm):  
    i, j = np.indices(glcm_norm.shape)  
    mean = glcmMean(glcm_norm)  
    variance = glcmVariance(glcm_norm)  
    c = correlation(glcm_norm)  
    a = np.sum(((i + j - (2 * mean)) ** 3) * glcm_norm)/(variance ** (3/2) * (np.sqrt(2 * (1 + c)) ** 3))  
    return np.sign(a) * (np.abs(a) ** (1/3))
```

Gambar 3.2.16 Fungsi Shade

- prominence: mengembalikan komponen fitur yang akan digunakan nanti.

```
def prominence(glcm_norm):  
    i, j = np.indices(glcm_norm.shape)  
    mean = glcmMean(glcm_norm)  
    variance = glcmVariance(glcm_norm)  
    c = correlation(glcm_norm)  
    b = np.sum(((i + j - (2 * mean)) ** 4) * glcm_norm)/(4 * (variance ** 2) * ((1 + c) ** 2))  
    return np.sign(b) * (np.abs(b) ** (1/4))
```

Gambar 3.2.17 Fungsi Prominence

- glcm_features: mengembalikan hasil penggabungan semua komponen fitur GLCM menjadi sebuah vektor.

```
def glcm_features(glcm_norm):  
    return np.array([contrast(glcm_norm), energy(glcm_norm), homogeneity(glcm_norm), correlation(glcm_norm), entropy(glcm_norm), shade(glcm_norm), prominence(glcm_norm)])
```

Gambar 3.2.18 Fungsi GLCM Features

- process_texture_image: otomatisasi proses perhitungan elemen-elemen dari vektor dan menulisnya pada file JSON.

```
def process_texture_image(filename, folder_path):
    img_path = os.path.join(folder_path, filename)
    glcmNorm_mat = glcmNorm(glcm(img_path))
    feat_vector = glcm_features(glcmNorm_mat)
    return {'path': filename, 'features': feat_vector.tolist()}
```

Gambar 3.2.19 Fungsi Process Texture Image

- DatasetToTextureJSON: menulis semua fitur-fitur gambar yang ada pada dataset.

```
def DatasetToTextureJSON(folder_path, json_path):
    filenames = os.listdir(folder_path)
    data_list = []

    with Pool() as pool:
        results = pool.starmap(process_texture_image, [(filename, folder_path) for filename in filenames])

    data_list.extend(results)

    with open(json_path, 'w') as json_file:
        json.dump(data_list, json_file, indent=4)
```

Gambar 3.2.20 Fungsi Dataset To Texture JSON

- SimilarityTexture: menghitung hasil *cosine similarity* dari gambar target dengan dataset dan mengurutkannya secara *descending* dan harus di atas 60%.

```
def SimilarityTexture(img_path, colour_json_path, colour_similiar_json_path):
    data = []
    with open(colour_json_path) as file:
        data = json.load(file)
    glcmNorm_mat_img = glcmNorm(glcm(img_path))
    feat_vector_img = glcm_features(glcmNorm_mat_img)
    similarities = []
    for img_dataset in data:
        vector_dataset = np.array(img_dataset['features'])
        similarity = cs.cosine_similarity(feat_vector_img, vector_dataset)
        if similarity >= 0.6:
            similarities.append({'path': img_dataset['path'], 'similarity': similarity})
    similarities = sorted(similarities, key=lambda x: x['similarity'], reverse=True)
    with open(colour_similiar_json_path, 'w') as file:
        json.dump(similarities, file, indent=4)
```

Gambar 3.2.21 Fungsi Similarity Texture

3.2.4. cosine_similarity

- cosine_similarity: sama seperti *cosine similarity* pada CBIR warna.

```
def cosine_similarity(v1,v2):
    dot_prod = np.dot(v1,v2)
    norm_v1 = np.linalg.norm(v1)
    norm_v2 = np.linalg.norm(v2)
    if norm_v1 == 0 or norm_v2 == 0:
        return 0
    else:
        return (dot_prod/(norm_v1*norm_v2))
```

Gambar 3.2.22 Fungsi Cosine Similarity

3.3. Ilustrasi Kasus

Dengan semua fungsionalitas diatas gambaran besar dari program kami

1. Pengguna mengupload dataset ke website
2. Pengguna mengupload image yang ingin dibandingkan ke website
3. Pengguna memilih antara colour atau texture
4. Program akan merubah dataset menjadi json sesuai dengan pilihan user
5. Jika pengguna memilih colour maka semua gambar dalam folder dataset akan diubah menjadi histogram menggunakan fungsi `create_histogram` dan kemudian dimasukkan ke json
6. Jika pengguna memilih texture maka semua gambar dalam folder dataset akan diubah menjadi GLCM dan diextract feature nya dan kemudian dimasukkan ke dalam file json
7. Program menampilkan hasil dari similarity
8. Pengguna dapat menggunakan searching lagi menggunakan dataset yang sama, jika program belum merubah dalam texture atau colour maka program melakukan langkah 5 atau 6. Jika sudah diubah maka program langsung membaca cache di json file dan menampilkan hasil similarity
9. Pengguna juga dapat mengupload dataset baru ke web dan backend akan merubah flag processed colour dan texture menjadi false

BAB 4

Implementasi dan Uji Coba

4.1. Implementasi Program Utama

Pada dasarnya program kita memiliki *frontend* dan *backend*. Di bagian *frontend*, kami menerima input image yang ingin dibandingkan dan juga *dataset* yang ingin dipakai. Sedangkan di bagian *backend*, terjadi processing untuk colour dan texture.

Fungsi `imageToMatrix` mengubah gambar menjadi matrix of RGB menggunakan *library* cv2.

```
ImageToMatrix(path)
    convert path to BGR matrix
    convert path to RGB matrix
    return matrix
```

Pertama akan dijelaskan terlebih dahulu untuk pemrosesan CBIR Colour. Terdapat tiga fungsi utama yaitu `create_histogram`, `DatasetToColourJSON` dan `SimilarityColour`. Untuk fungsi `create_histogram`, pertama-tama kita mengconvert image menjadi matrix of HSV dan kemudian membagi HSV matrix menjadi 4x4, kemudian tiap segmen akan dikalkulasi histogramnya dan kemudian di semua hasil histogram per segmen digabungkan menjadi satu array

```
create_histogram(img_path)
    convert img_path to matrix of HSV
    divide into 16 segments
    loop from 0 to 3
        loop from 0 to 3
            define current segment
            calculate histogram for H
            calculate histogram for S
            calculate histogram for V
            concat into 1 vector
            append to list of histogram
    return list of histogram
```

Untuk prosedur `DatasetToColourJSON`, fungsi ini menerima folder disimpannya image dan kemudian merubahnya menjadi JSON file yang telah ditentukan.

```
DatasetToColourJSON(folder_path,json_path)
    get image path and filename
    use multiprocessing to convert each image into histograms
    write all of the image path and histogram into json
```

Prosedur SimilarityColour menerima image yang ingin dibandingkan, kemudian dibandingkan dengan data pada JSON colour. Hasilnya akan dituliskan di JSON similarity

```
SimilarityColour(img_path,colour_json_path,similarity_json_path)
    get the data from colour_json_path
    compare each segments histogram with cosine similarity then average it
    if above 60%
        write to similarity_json_path
```

Fungsi img2GRY menerima input berupa *path* dari gambar yang ingin dikonversi, lalu memanggil fungsi ImageToMatrix() dan mengkonversinya menjadi matriks *grayscale*.

```
function img2GRY(path):
    img_rgb ← ImageToMatrix(path)
    gray ← img_rgb[:, :, 0] * 0.299 + img_rgb[:, :, 1] * 0.587 + img_rgb[:, :, 2] * 0.114
    → gray
```

Fungsi glcm menerima input berupa *path* dari gambar yang ingin dibuat GLCM-nya, lalu memanggil fungsi img2GRY() dan membuat GLCM-nya.

```
function glcm(path):
    img ← img2GRY(path)
    quantized ← round(img).astype(int)
    max_val ← quantized.max() + 1
    glcm_matrix ← zeros((max_val, max_val), dtype=int)
    for i in range(quantized.shape[0] - 1):
        for j in range(quantized.shape[1] - 1):
            glcm_matrix[quantized[i, j], quantized[i, j + 1]] += 1
            glcm_matrix[quantized[i, j + 1], quantized[i, j]] += 1
    → glcm_matrix
```

Fungsi glcmNorm menghitung normal dari GLCM.

```
function glcmNorm(glcm_matrix):
    → glcm_matrix / glcm_matrix.sum()
```

Fungsi glcmMean menghitung GLCM *mean* yang akan digunakan untuk perhitungan di fungsi lain.

```
function glcmMean(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    → sum(i * glcm_norm)
```

Fungsi glcmVariance untuk menghitung varians dari matriks normal GLCM.

```
function glcmVariance(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    mean ← glcmMean(glcm_norm)
    → sum(glcm_norm * (i - mean) ** 2)
```

Fungsi contrast untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function contrast(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    → sum(glcm_norm * (i - j) ** 2)
```

Fungsi energy untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function energy(glcm_norm):
    → sum(glcm_norm ** 2)
```

Fungsi homogeneity untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function homogeneity(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    → sum(glcm_norm / (1 + (i - j) ** 2))
```

Fungsi correlation untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function correlation(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    mean ← glcmMean(glcm_norm)
    variance ← glcmVariance(glcm_norm)
    → sum(glcm_norm * ((i - mean) * (j - mean))/variance)
```

Fungsi entropy untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function entropy(glcm_norm):
    mask ← glcm_norm > 0
    → -sum(glcm_norm[mask] * log10(glcm_norm[mask]))
```

Fungsi shade untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function shade(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    mean ← glcmMean(glcm_norm)
    variance ← glcmVariance(glcm_norm)
    c ← correlation(glcm_norm)
    a ← sum(((i + j - (2 * mean)) ** 3) * glcm_norm)/((variance ** (3/2)) * (sqrt(2 * (1 +
    c)) ** 3)))
    → sign(a) * (abs(a) ** (1/3))
```

Fungsi prominence untuk menghitung fitur yang merupakan elemen dari vektor nanti.

```
function prominence(glcm_norm):
    i, j ← indices(glcm_norm.shape)
    mean ← glcmMean(glcm_norm)
    variance ← glcmVariance(glcm_norm)
    c ← correlation(glcm_norm)
    b ← sum(((i + j - (2 * mean)) ** 4) * glcm_norm)/(4 * (variance ** 2) * ((1 + c) ** 2))
    → sign(b) * (abs(b) ** (1/4))
```

Fungsi glcm_features untuk membuat vektor umum untuk dibandingkan dengan vektor gambar lain di *cosine similarity*.

```
function glcm_features(glcm_norm):
    → array([contrast(glcm_norm), energy(glcm_norm), homogeneity(glcm_norm),
    correlation(glcm_norm), entropy(glcm_norm), shade(glcm_norm),
    prominence(glcm_norm)])
```

Fungsi process_texture_image untuk memanggil fungsi-fungsi diatas.

```
function process_texture_image(filename, folder_path):
    img_path ← path.join(folder_path, filename)
    glcmNorm_mat ← glcmNorm(glcm(img_path))
    feat_vector ← glcm_features(glcmNorm_mat)
    → {'path': filename, 'features': feat_vector.tolist()}
```

Prosedur DatasetToTextureJSON untuk memasukkan hasil perhitungan vektor dataset ke dalam file dengan *extension JSON*.

```
procedure DatasetToTextureJSON(folder_path, json_path):
    filenames ← listdir(folder_path)
    data_list ← []
    with Pool() as pool:
        results ← pool.starmap(process_texture_image, [(filename, folder_path) for filename
        in filenames])
        data_list.extend(results)
    with open(json_path, 'w') as json_file:
        dump(data_list, json_file, indent=4)
```

Prosedur SimilarityTexture untuk otomatisasi perhitungan vektor dataset.

```

procedure SimilarityTexture(img_path, colour_json_path, colour_similiar_json_path):
    data ← []
    with open(colour_json_path) as file:
        data = load(file)
    glcmNorm_mat_img ← glcmNorm(glcm(img_path))
    feat_vector_img ← glcm_features(glcmNorm_mat_img)
    similarities ← []
    for img_dataset in data:
        vector_dataset ← array(img_dataset['features'])
        similarity ← cosine_similarity(feat_vector_img, vector_dataset)
        if similarity >= 0.6:
            similarities.append({'path': img_dataset['path'], 'similarity': similarity})
    similarities ← sorted(similarities, key=lambda x: x['similarity'], reverse=True)
    with open(colour_similiar_json_path, 'w') as file:
        dump(similarities, file, indent=4)

```

Kemudian di frontend terdapat beberapa fungsi yang notable. Pertama ialah upload_dataset. Kita memilih untuk mengupload folder secara langsung, namun karena constraint dari flask sendiri kita memilih untuk meng-zip foldernya terlebih dahulu dan kemudian dikirim ke backend

```

upload_dataset()
    get zip file
    using multiprocessing unzip it into dataset folder
    change the state of processedColour to false
    change the state of processedTexture to false

```

Kedua ialah fungsi search yang melakukan comparison dari image yang di upload dengan dataset yang dimiliki

```

search()
    get images from javascript
    if image exist
        empty upload_image folder
        save to upload_image folder
        get the path image from the upload_image folder
    else
        get the path image from the upload_image folder
    start time
    find dataset path
    if toggle = colour
        if not processedColour
            DatasetToColourJSON()
            SimilarityColour()
    else
        if not processedTexture
            DatasetToTextureJSON()
            SimilarityTexture()
    stop time

```

4.2. Penjelasan Struktur Program

Berikut directory dari program kami:

```
src
| — data
|   | — colour.json
|   | — similarity.json
|   | — texture.json
|
| — function
|   | — CBIR_Colour.py
|   | — CBIR_Colour.py
|   | — cosine_similarity.py
|   | — ImageToMatrix.py
|
| — static
|   | — assets
|   | — dataset
|   | — image_upload
|   | — aboutus_style.css
|   | — howToUse_style.css
|   | — index_style.css
|   | — indexScripts.js
|   | — result_style.css
|   | — resultScripts.js
|   | — searchengineconcepts.css
|
| — templates
|   | — aboutus.html
|   | — howtouse.html
|   | — index.html
|   | — result.html
|   | — searchengineconcepts.html
|
| — app.py
```

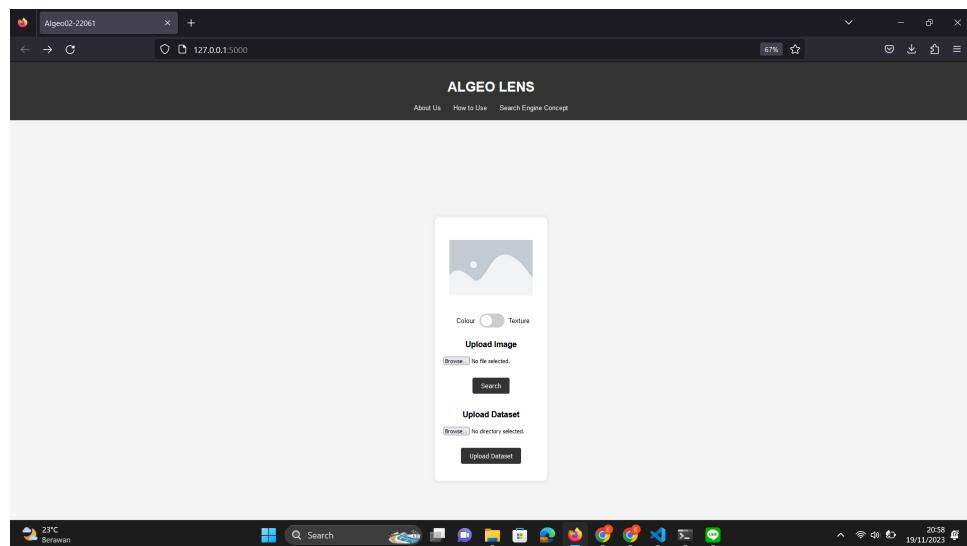
Semua kode disimpan dalam folder src yang berisi folder data, function, static, templates, dan app.py. Folder data berisi file-file json. Folder function berisi file python untuk memproses CBIR dengan metode warna maupun tekstur.

Folder static berisi folder assets, dataset, dan image_upload. Folder assets berisi gambar-gambar permanen yang digunakan dalam web. Folder dataset digunakan untuk menyimpan dataset. Sedangkan folder image_upload digunakan untuk menyimpan gambar yang akan dibandingkan. Selain itu, dalam folder static terdapat file-file css untuk mengatur style dan juga file-file javascript untuk backend.

Folder templates berisi file-file html yang terdiri atas index.html sebagai halaman utama, result.html, aboutus.html, howtouse.html, dan searchengineconcepts.html. File app.py berisi program untuk routing, menghapus dataset dan/atau gambar ketika akan diunggah ulang oleh pengguna, mengekstrak file, mengupload dataset, memasukkan data waktu, mengatur toggle, mengatur pagination, pengunduhan pdf.

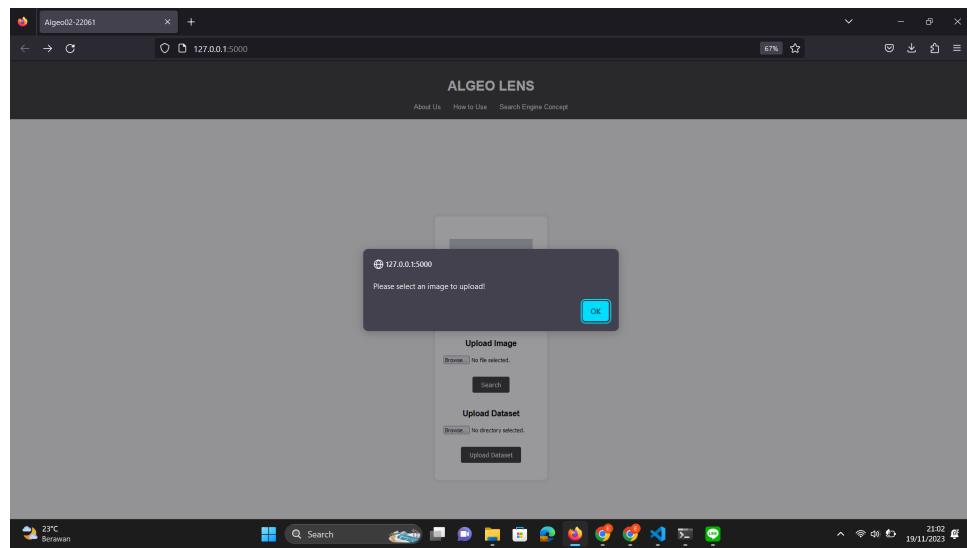
4.3. Penjelasan Tata Cara Penggunaan Program

Pada halaman utama, user dapat mengupload gambar yang ingin dibandingkan dan dataset yang akan diproses. User dapat memilih metode apa yang ingin digunakan (color atau texture) menggunakan toggle yang tersedia.



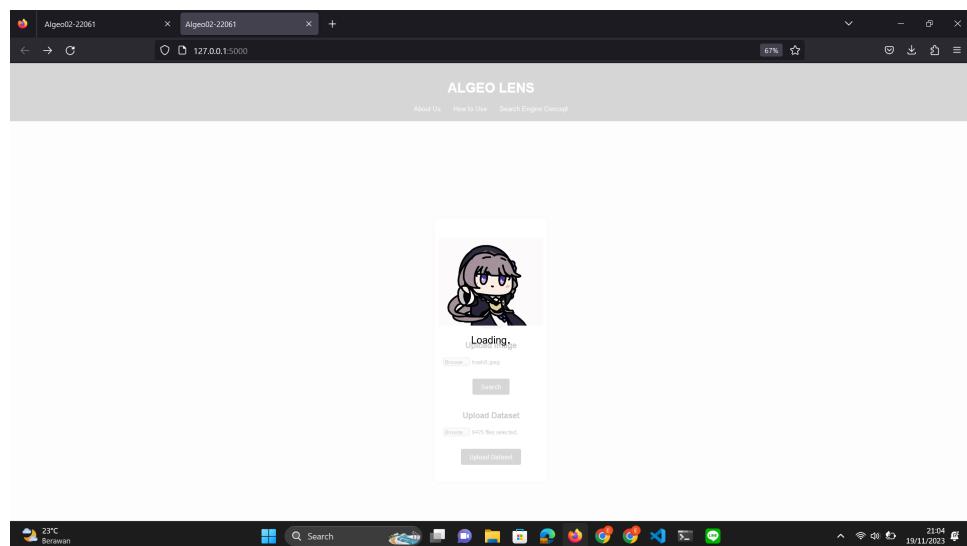
Gambar 4.3.1 Tampilan Utama Website

Jika belum ada gambar yang diupload, maka akan ditampilkan alert seperti berikut.



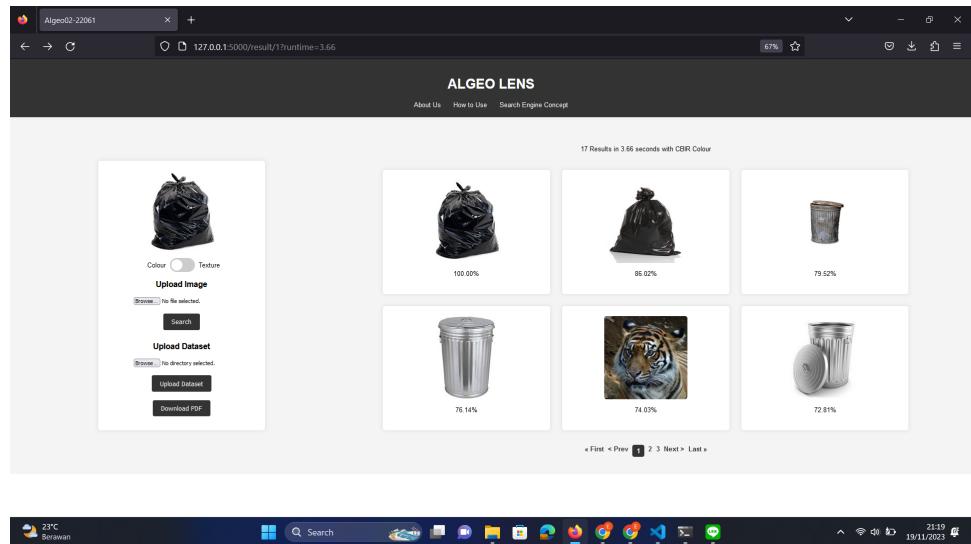
Gambar 4.3.2 Kondisi Ketika Gambar Belum Diunggah

Sesudah mengupload gambar dan dataset, user dapat mengklik tombol search untuk memulai pencarian. Selagi data diproses, akan terdapat loading page seperti berikut.



Gambar 4.3.3 Loading Page

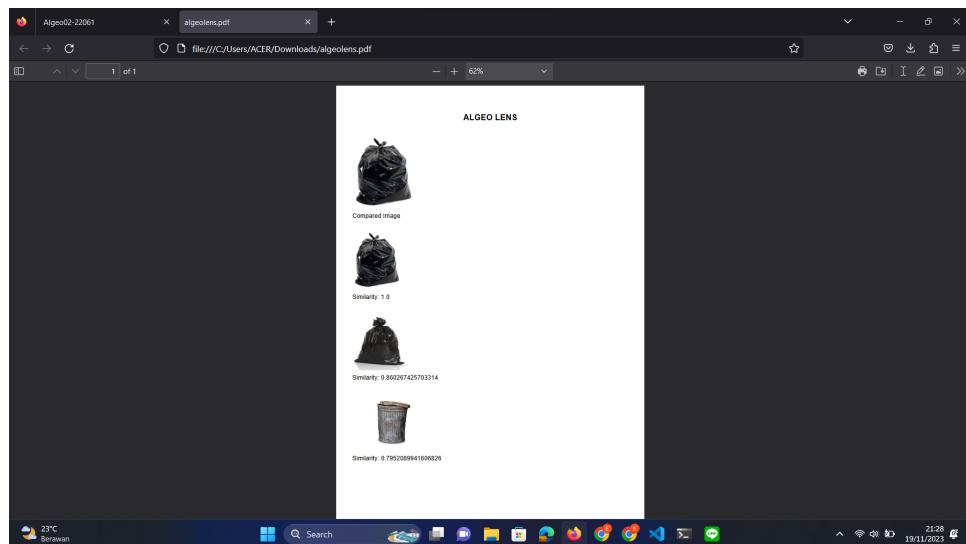
Hasil dari pemrosesan data yang telah diunggah ditampilkan sebagai berikut.



Gambar 4.3.4 Page Result

Terdapat beberapa fitur pada page ini antara lain:

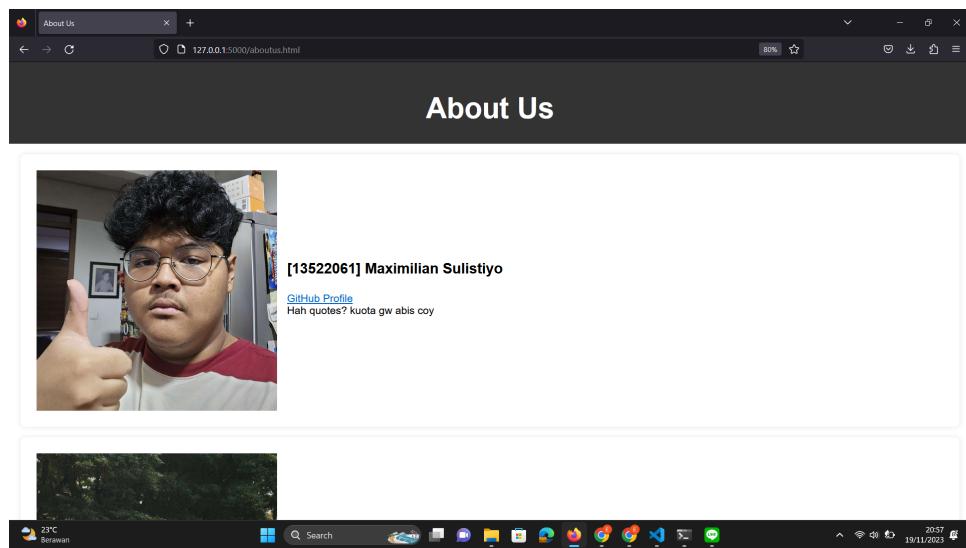
- reupload gambar: pengguna dapat mengunggah ulang gambar yang akan dibandingkan dengan dataset yang lama tanpa mengunggah ulang dataset. Hasil pemrosesan dataset yang lama akan tersimpan selama pengguna tidak mengunggah dataset baru.
- Toggle: pengguna dapat memilih ulang metode yang ingin digunakan.
- Reupload dataset: pengguna dapat mengunggah ulang dataset tanpa perlu mengunggah ulang gambar yang ingin dibandingkan.
- Download PDF: pengguna dapat mengunduh pdf yang berisi gambar yang dibandingkan dan tiga gambar paling mirip dengan gambar yang dicari beserta persentase kemiripannya. Tampilan pdf yang diunduh seperti berikut.



Gambar 4.3.5 Hasil PDF

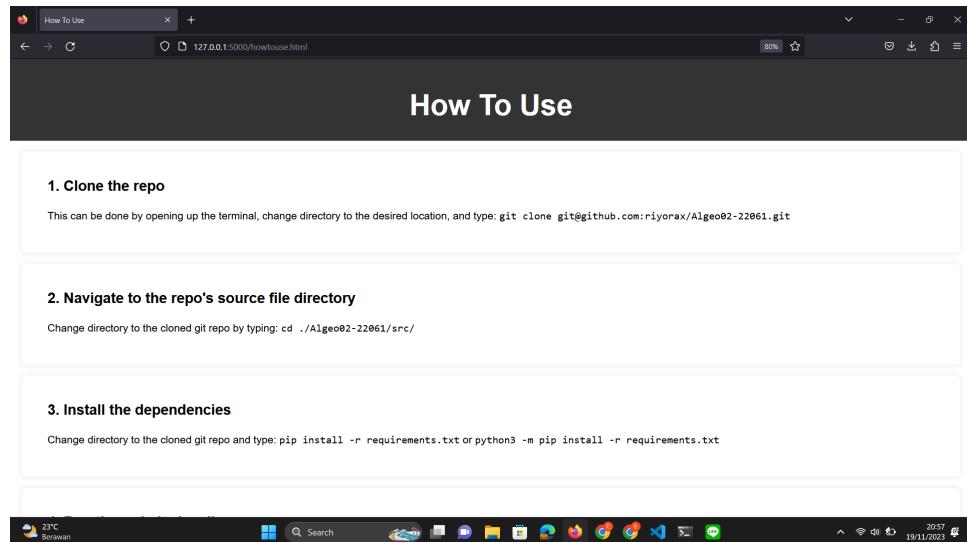
Selain itu, terdapat fitur about us, how to use, dan search engine concept.

Page about us berisi siapa saja yang berkontribusi dalam pembuatan web algeo lens ini.



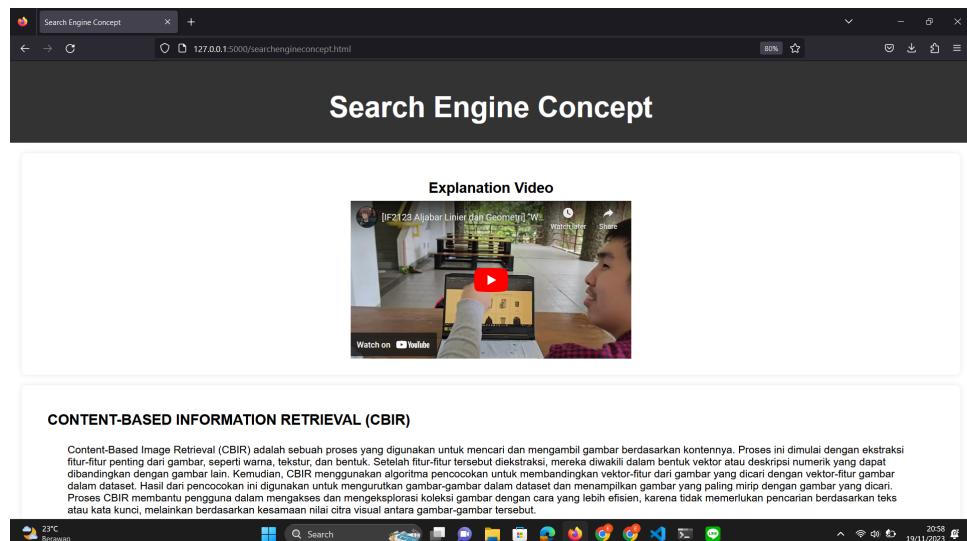
Gambar 4.3.5 Halaman About Us

Page how to use berisi langkah-langkah untuk menggunakan web ini, mulai dari clone repository sampai run.



Gambar 4.3.5 Halaman How To Use

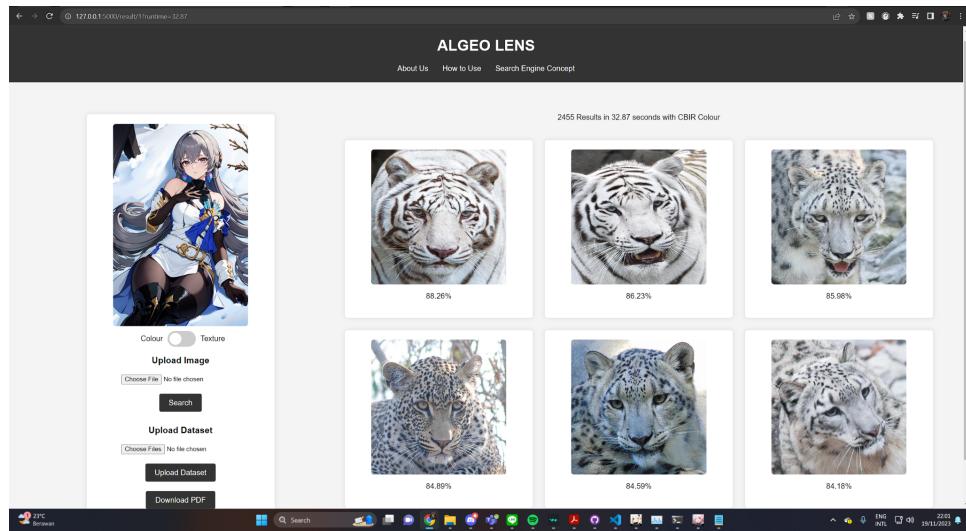
Page search engine concept berisi tentang penjelasan CBIR yang digunakan dalam pembuatan web ini, serta terdapat video singkat yang kami buat.



Gambar 4.3.5 Halaman Search Engine Concepts

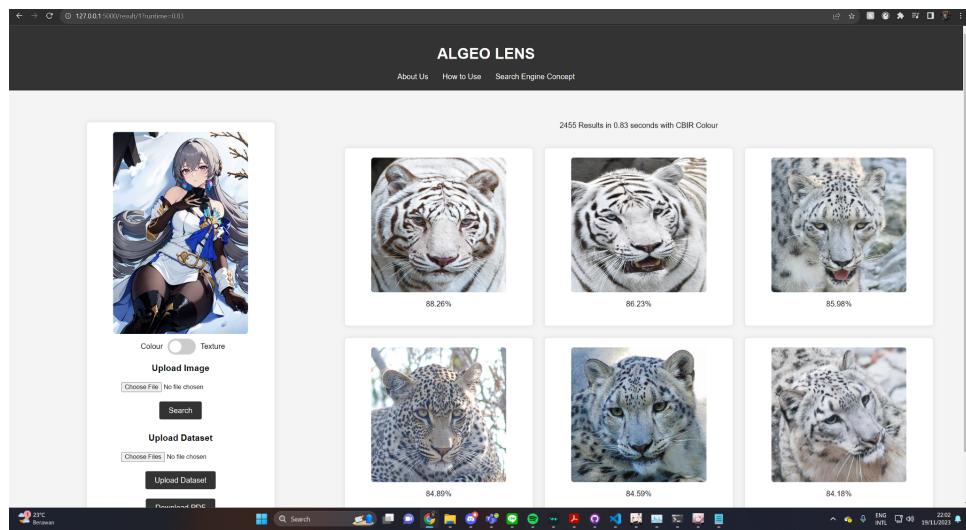
4.4. Hasil Pengujian

Melakukan searching menggunakan colour sebelum data di cache

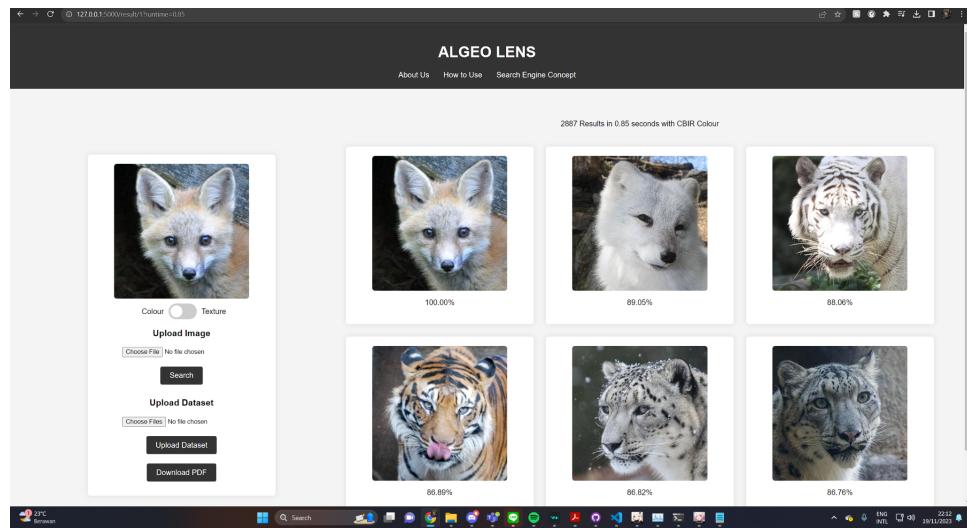


Gambar 4.3.1 Hasil sebelum menggunakan cache

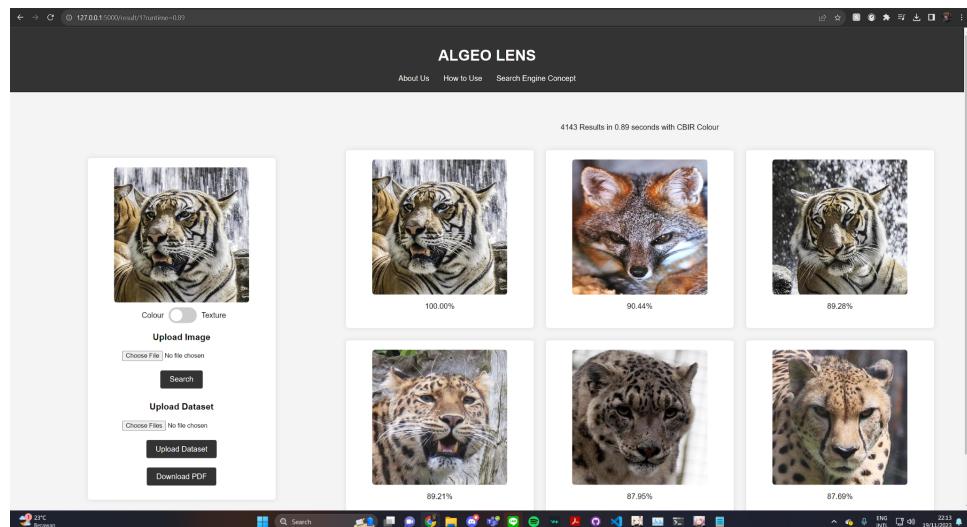
Melakukan searching menggunakan colour setelah data di cache



Gambar 4.3.2 Hasil setelah menggunakan cache

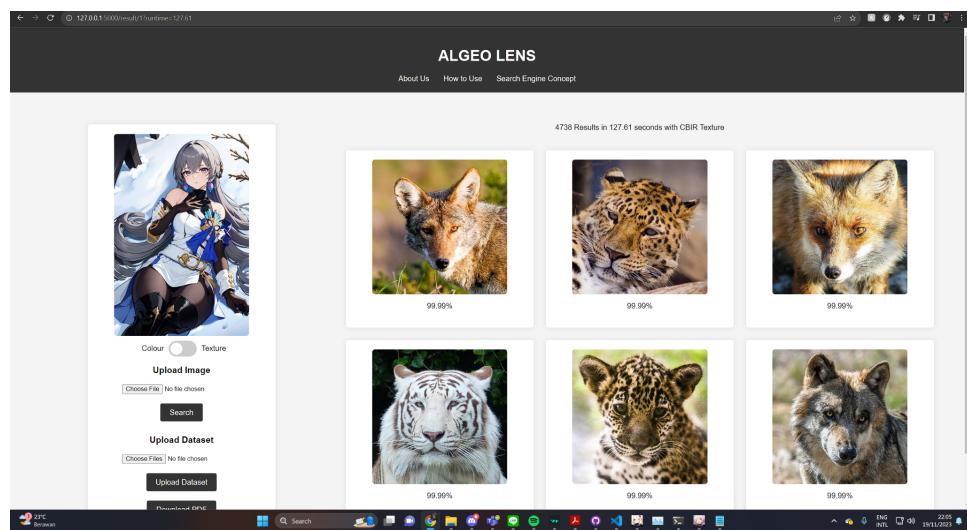


Gambar 4.3.3 Hasil setelah menggunakan cache



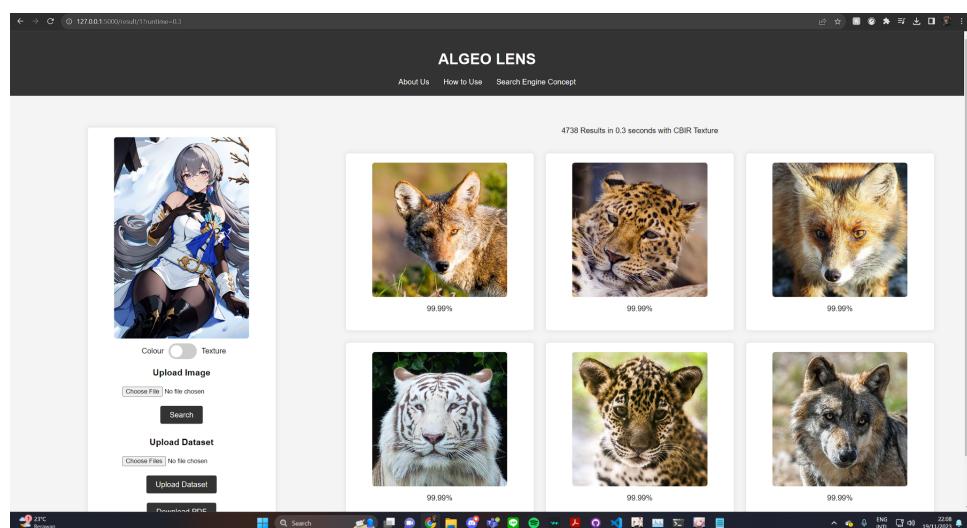
Gambar 4.3.4 Hasil setelah menggunakan cache

Melakukan searching menggunakan texture sebelum data di cache

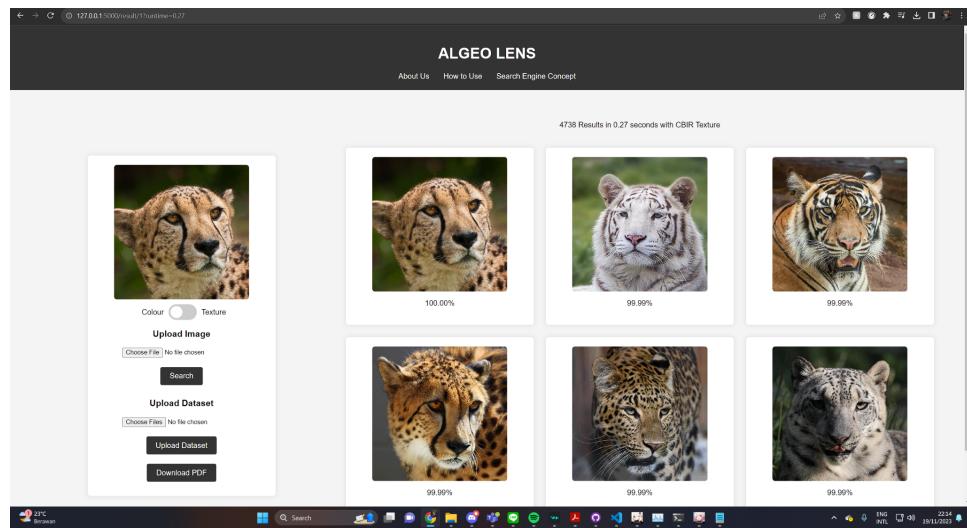


Gambar 4.3.5 Hasil sebelum menggunakan cache

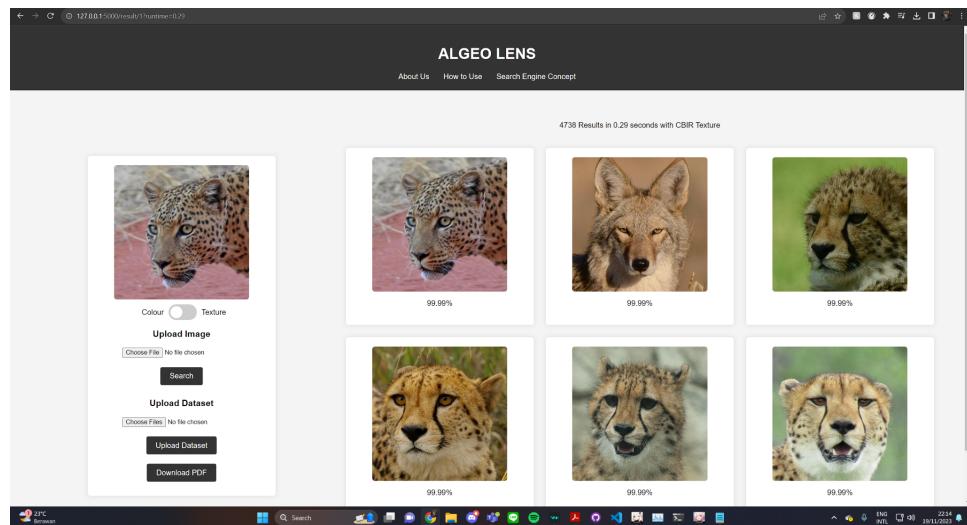
Melakukan searching menggunakan texture setelah data di cache



Gambar 4.3.6 Hasil sesudah menggunakan cache



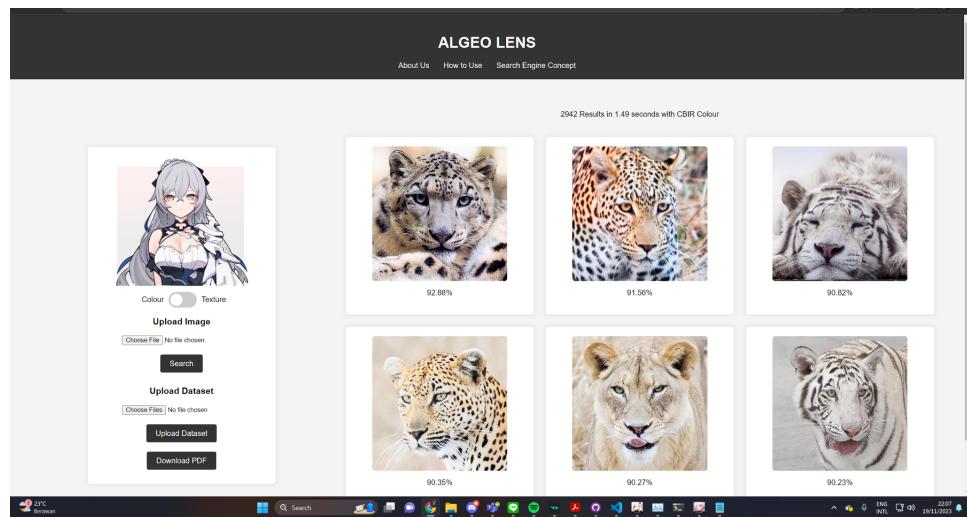
Gambar 4.3.7 Hasil sesudah menggunakan cache



Gambar 4.3.8 Hasil sesudah menggunakan cache

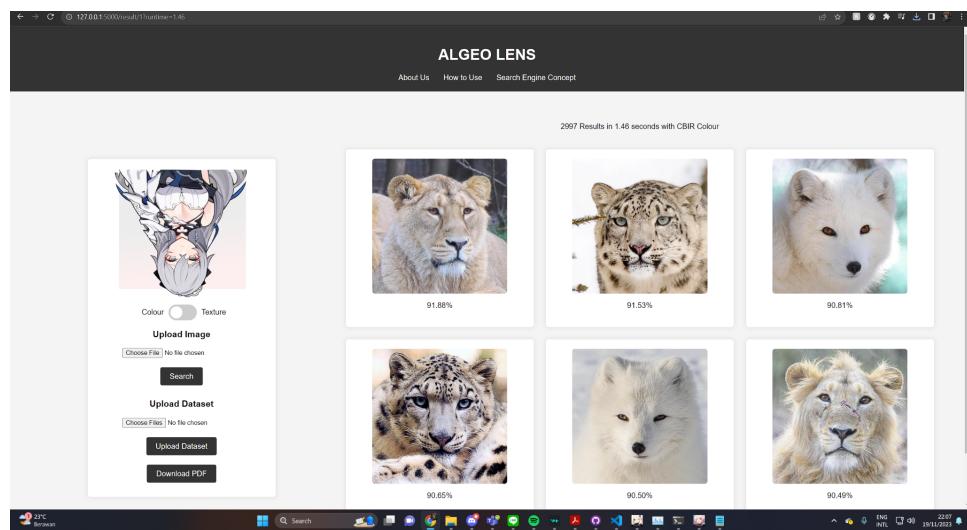
Melakukan searching menggunakan colour dengan gambar di normal dan di balik

Normal



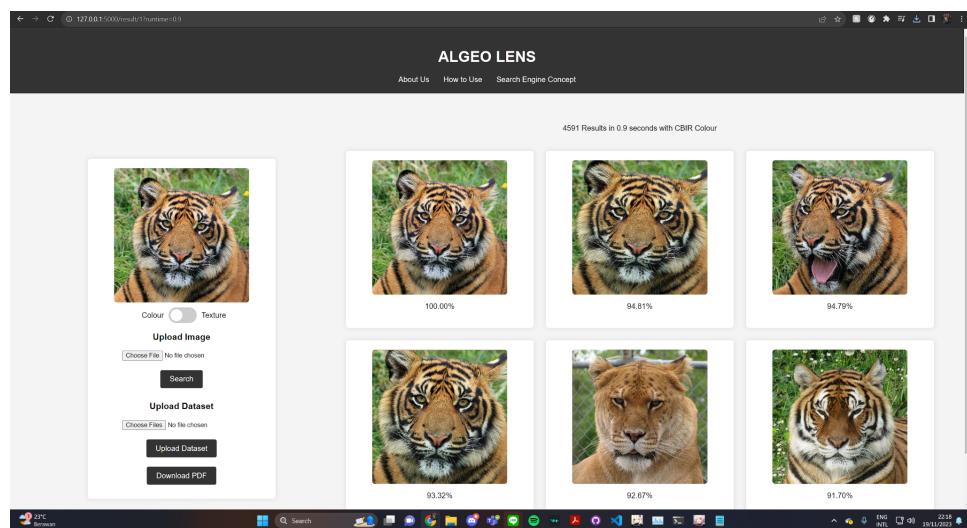
Gambar 4.3.9 Hasil gambar normal

Kemudian setelah gambar dibalik:



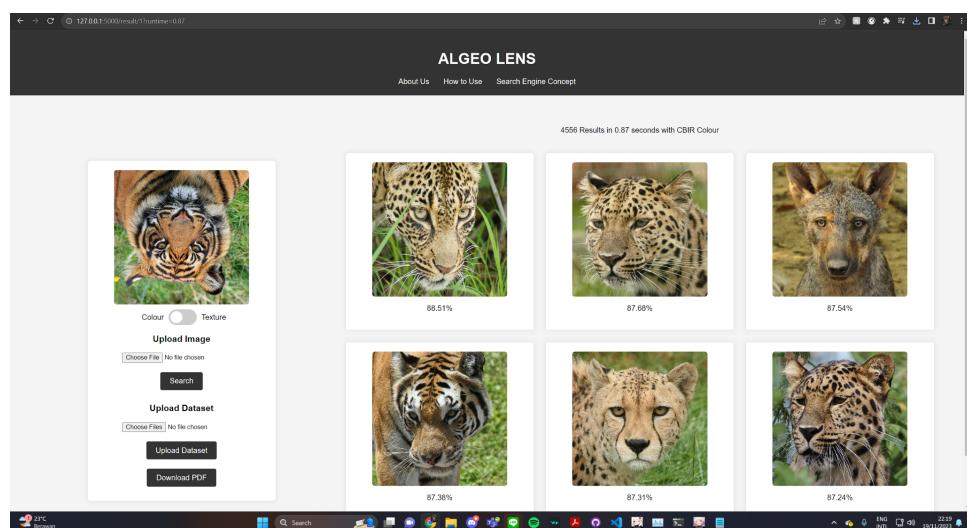
Gambar 4.3.10 Hasil gambar dibalik

Hasil gambar normal:



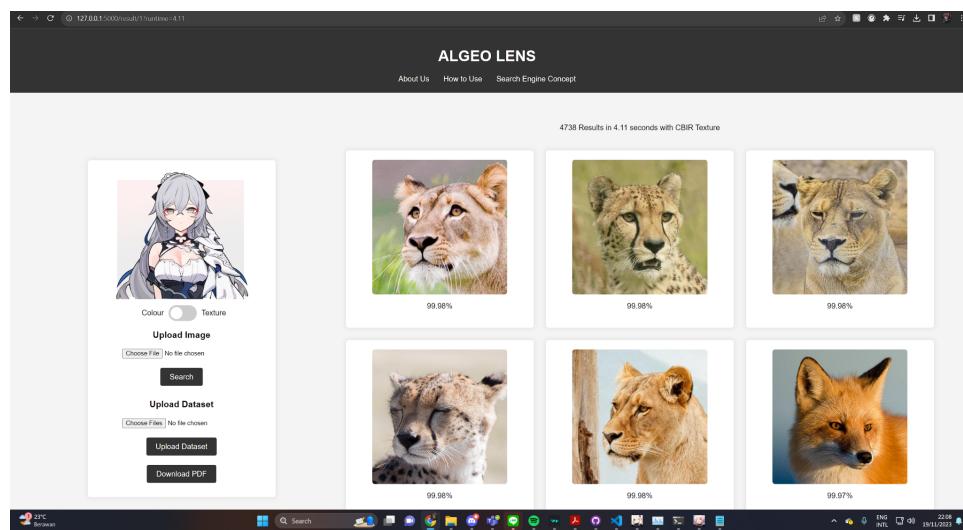
Gambar 4.3.11 Hasil gambar normal

Hasil gambar dibalik:

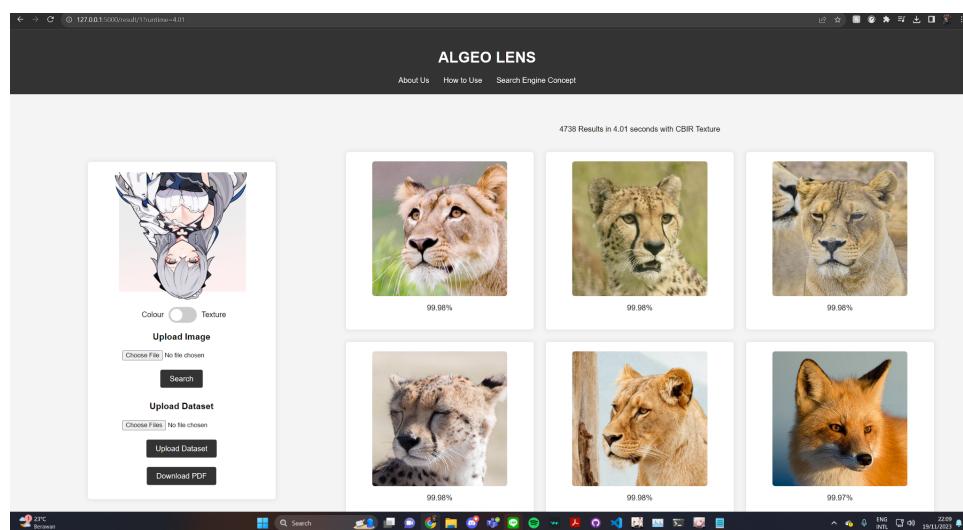


Gambar 4.3.12 Hasil gambar dibalik

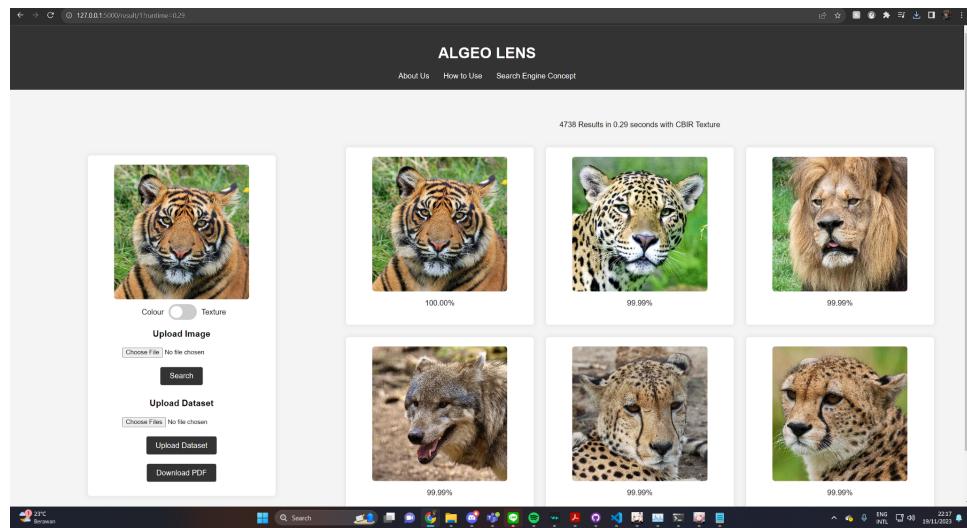
Melakukan searching menggunakan texture dengan gambar di normal dan di balik:



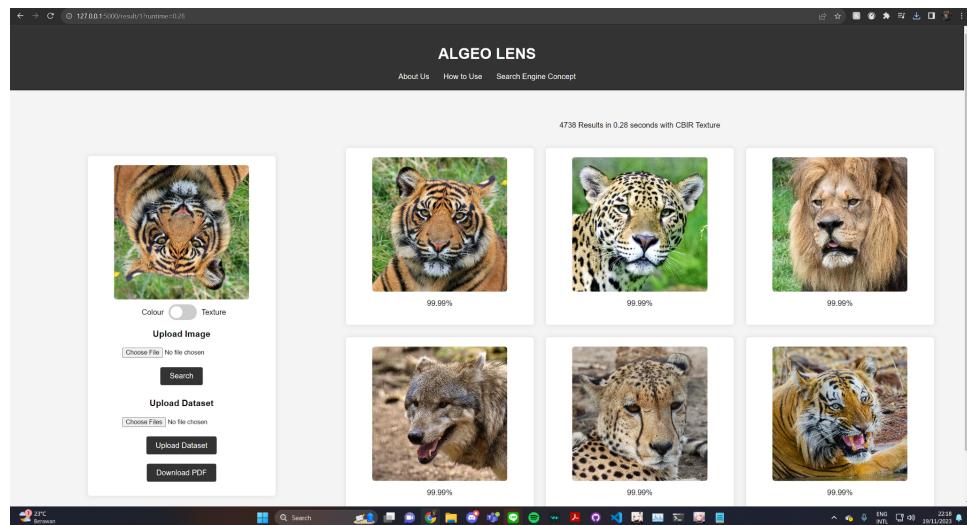
Gambar 4.3.13 Hasil gambar normal



Gambar 4.3.14 Hasil gambar dibalik



Gambar 4.3.15 Hasil gambar normal



Gambar 4.3.16 Hasil gambar dibalik

4.5. Analisis

Setelah membuat program ini kami mendapatkan beberapa hal yang dapat ditelusuri lebih mendalam. Pertama ialah ukuran bin pada pembuatan histogram mempengaruhi hasil akhir kemiripan. Pada penggunaan bin size sesuai dengan referensi di spek tugas besar kami mendapatkan hasil kurang lebih 4-5% lebih besar dibandingkan jika kita menggunakan bin size yang langsung dibagi rata yaitu membagi 8 H dan membagi 3 untuk S dan V.

Dengan menggunakan searching menggunakan texture kami melihat bahwa mayoritas hasil similarity menunjukkan angka 99%. Hal ini dikarenakan tingginya salah satu feature yaitu contrast. Untuk membantu akurasi dari program pun kami telah mencari 5 feature tambahan yaitu correlation, entropy, shade, dan prominence. Menambahkan feature tersebut terhadap vektor feature pun menambah akurasi dari program hingga dapat menghasilkan similarity yang cukup rendah dibanding sebelumnya yaitu 88%

Manipulasi matrix menggunakan python secara langsung merupakan cara yang sangat tidak efisien sehingga penggunaan library numpy pun merupakan sebuah breakthrough dalam mempercepat program kami secara keseluruhan

Penggunaan multiprocessing dalam mengubah dataset menjadi JSON merupakan salah satu breakthrough dalam mempercepat fitur searching. Terlihat juga bahwa walaupun kami menggunakan bahasa yang konotasi nya lambat namun kami berhasil mendapatkan hasil runtime yang cukup memuaskan.

Fitur caching sangat diperlukan melihat lamanya processing jika diberikan ribuan dataset. Namun perlu diperhatikan juga bahwa fitur ini dapat juga tidak mempengaruhi kecepatan jika CPU dari pengguna tidak kuat melakukan multiprocessing.

BAB 5

Kesimpulan, Saran, Komentar, dan Refleksi

5.1. Kesimpulan

Kesimpulan dari tugas besar ini adalah kami dapat mengaplikasikan ekstraksi informasi gambar dan memprosesnya menjadi fitur-fitur yang bisa dibandingkan dengan gambar lain untuk membandingkan kemiripannya. Kami berhasil membuat sebuah *website* yang memanfaatkan algoritma CBIR warna dan CBIR tekstur yang kami rasa cukup *performant* jika melihat bahasa python yang biasanya dipandang lambat dibandingkan bahasa lain. Kami juga berhasil melewati masalah yang dihadapi ketika mengunggah file foto yang banyak dan memiliki ukuran yang besar ke dataset dan berhasil mengimplementasikan fitur *export* hasil ke bentuk PDF.

5.2. Saran

Bagi para asisten mungkin bisa diperbanyak implementasi ilmu Aljabar Linier dan Geometrinya dalam pembuatan tugas besar ini karena kami merasakan bahwa tugas besar ini sangat diberatkan pada pembuatan website dibandingkan dengan ilmu Algeo itu sendiri. Kemudian jika tujuannya adalah untuk kita mencari cara membuat website maka kami sarankan lebih memperbanyak sumber belajar. Dari sisi implementasi searching nya sendiri kita tidak memiliki reference akan kebenaran dari program kita, mungkin dapat dibuat contoh pada spek akan kemiripan dua buah gambar agar kami memiliki benchmark dalam pembuatan function.

5.3. Komentar

Komentar kami untuk tugas besar ini hanyalah satu kata, “Wow”. Tidak dilupakan juga untuk kata-kata “Coba eksplor sendiri”.

5.4. Refleksi

Tugas besar ini mengajarkan kami untuk mengeksplor sangat jauh. Kami mencari dari berbagai sumber agar tugas besar ini dapat diselesaikan. Kami juga belajar untuk bekerja sama sebagai team untuk menyelesaikan tugas besar ini. Terakhir kita pun belajar time management dimasa tugas besar melimpah di prodi IF ini. Pengorbanan dan jerih payah kami pun yang mempersesembahkan tugas besar ini kepada para asisten.

Daftar Pustaka

1. [Spesifikasi Tugas Besar 2 IF2123 Algeo 2023/2024 - Google Drive](#)
2. [Content-based image retrieval using color and texture fused features - ScienceDirect](#)
3. [image processing - Converting RGB to grayscale/intensity - Stack Overflow](#)
4. [GLCMs — a Great Tool for Your ML Arsenal | by Martim Chaves | Towards Data Science](#)
5. [wp-Different-Color-Spaces-QC.pdf \(telestream.net\)](#)
6. [Co-occurrence matrix - Wikiwand](#)
7. Campbell, Jennifer (2017). Web Design: Introductory. Cengage Learning. p. 23-27.
8. [copy — Shallow and deep copy operations — Python 3.12.0 documentation](#)
9. [PyPI · The Python Package Index](#)
10. [OpenCV: Geometric Image Transformations](#)
11. [GLCM Texture Feature \(echoview.com\)](#)
12. [Texture Descriptor : Gray level Co-occurrence Matrix \(GLCM\) \(binus.ac.id\)](#)
13. [\(PDF\) Analysis and Comparison of Texture Features for Content Based Image Retrieval \(researchgate.net\)](#)
14. [Open Access proceedings Journal of Physics: Conference series \(iop.org\)](#)
15. [Почему невозможно корректно ограничить размер закачиваемого файла \(sysoev.ru\)](#)
16. <https://numpy.org/>
17. <https://www.w3schools.com/css/>
18. <https://stackoverflow.com/questions/40149743/return-pdf-generated-with-fpdf-in-flask>
19. Link Video <https://youtu.be/BNi19rGrGkU?si=Y0lZa1r6bNVwFoSU>
20. Link Repository <https://github.com/riyorax/Algeo02-22061>