

Laporan Tugas Besar 2
IF2123 Aljabar Linier dan Geometri
Aplikasi Aljabar Vektor dengan Sistem Temu Balik
Gambar



Disusun oleh:

Aurelius Justin Philo Fanjaya (13522020)

Fedrianz Dharma (13522090)

Steven Tjhia (13522103)

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung**

2023

Daftar Isi

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung**

2023.....	1
Daftar Isi.....	2
Bab I	
Deskripsi Masalah.....	5
1.1 ABSTRAKSI.....	5
1.2 CONTENT-BASED INFORMATION RETRIEVAL (CBIR).....	5
1.2.1 CBIR dengan parameter warna.....	6
1.2.2 CBIR dengan parameter tekstur.....	7
1.3 PENGGUNAAN PROGRAM.....	10
Bab II	
Landasan Teori.....	12
2.1 Dasar Teori Secara Umum.....	12
2.1.1 CBIR dengan Parameter Warna.....	12
2.1.2 CBIR dengan Parameter Tekstur.....	12
2.2 Penjelasan Mengenai Pengembangan Sebuah Website.....	13
Bab III	
Analisis Pemecahan Masalah.....	14
3.1 Langkah-Langkah Pemecahan Masalah.....	14
3.1.1 CBIR dengan Parameter Warna.....	14
3.1.2 CBIR dengan Parameter Tekstur.....	15
3.2 Proses Pemetaan Masalah.....	16
3.2.1 CBIR dengan Parameter Warna.....	16
3.2.1.1 Membuka gambar dan mengubahnya menjadi matrix RGB.....	17
3.2.1.2 Mengubah nilai RGB pada matrix menjadi HSV.....	17
3.2.1.3 Mengkuantifikasi nilai HSV berdasarkan range tertentu.....	17
3.2.1.4 Menggabungkan matrix H, S, dan V menjadi satu matrix.....	18
3.2.1.5 Memecah matrix menjadi 4 x 4 blok.....	18
3.2.1.6 Menghitung frekuensi warna pada masing-masing blok.....	19
3.2.1.7 Menghitung cosine similarity dari 2 gambar.....	19
3.2.2 CBIR dengan Parameter Tekstur.....	19
3.2.2.1 Membuka gambar dan mengubahnya menjadi matrix RGB.....	20
3.2.2.2 Mengubah Matrix RGB menjadi Matrix Grayscale.....	20
3.2.2.3 Membentuk Matrix co-occurrence.....	20
3.2.2.4 Membentuk Matrix Symmetric.....	21
3.2.2.5 Membentuk Matrix GLCM (Normalisasi Matrix Symmetric).....	21
3.2.2.6 Hitung Contrast, Homogeneity, dan Entropy.....	21
3.2.2.7 Menghitung nilai cosine similarity.....	22
3.3 Ilustrasi Kasus dan Penyelesaiannya.....	22
3.3.1 CBIR dengan Parameter Warna.....	22
3.3.1.1 Membuka gambar dan mengubahnya menjadi matrix RGB dan	

dinormalisasi.....	22
3.3.1.2 Mengubah nilai RGB pada matrix menjadi HSV.....	23
3.3.1.5 Memecah matrix menjadi 4 x 4 blok.....	26
3.3.2 CBIR dengan Parameter Tekstur.....	27
3.3.2.1 Membuka gambar dan mengubahnya menjadi matrix RGB.....	27
3.3.2.2 Mengubah Matrix RGB menjadi Matrix Grayscale.....	27
3.3.2.3 Membentuk Matrix co-occurrence.....	27
3.3.2.4 Membentuk Matrix Symmetric.....	28
3.3.2.5 Membentuk Matrix GLCM (Normalisasi Matrix Symmetric).....	28
3.3.2.6 Hitung Contrast, Homogeneity, dan Entropy.....	28
3.3.2.7 Menghitung nilai cosine similarity.....	28
Bab IV	
Implementasi dan Uji Coba.....	29
4.1 Program Utama.....	29
4.1.1 CBIR dengan Parameter Warna.....	29
4.1.2 CBIR dengan Parameter Tekstur.....	30
4.2 Struktur Program.....	31
4.2.1 Struktur Folder.....	31
4.2.2 Struktur Program pada models.py.....	33
4.2.3 Struktur Program pada views.py.....	34
4.2.4 Struktur Program Pencarian pada Website.....	35
4.3. Tata Cara Penggunaan Program.....	37
4.3.1 Mengaktifkan Virtual Environment.....	37
4.3.2 Install Django.....	37
4.3.3 Install Numpy.....	37
4.3.4 Install Pillow.....	38
4.3.5 Install Widget Tweaks.....	38
4.3.6 Install Whitenoise.....	38
4.3.7 Menjalankan Server.....	38
4.3.8 Homepage.....	39
4.3.9 Upload Image.....	39
4.3.10 Upload Dataset.....	40
4.3.11 Pencarian.....	41
4.3.12 Change Image.....	42
4.3.13 Change Dataset.....	42
4.4 Hasil Pengujian.....	43
4.5 Analisis Solusi.....	53
4.5.1. Kasus Pencarian Gambar dengan Warna yang Mirip.....	53
4.5.2. Kasus Pencarian Gambar yang di-invert dan diberi Filter.....	53
4.5.3. Kasus Pencarian Gambar dengan Sudut Pandang yang Berbeda.....	54
4.5.4. Kasus Pencarian Gambar PNG Transparan.....	54
Bab V	
Kesimpulan.....	55
5.1 Kesimpulan.....	55
5.2 Saran.....	55

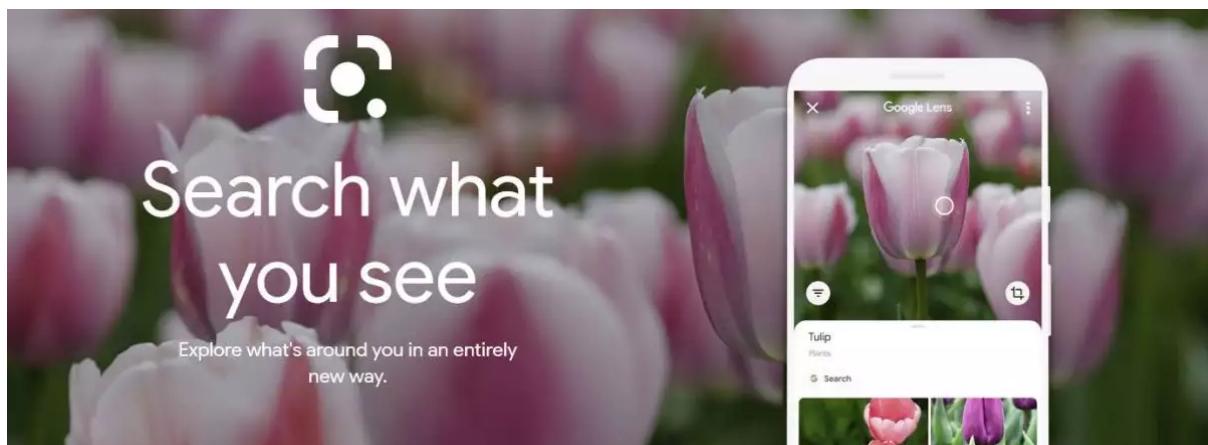
5.3 Komentar.....	55
5.4 Refleksi.....	56
5.5 Ruang Perbaikan.....	56
Daftar Pustaka.....	57
Link Repository dan Video.....	59

Bab I

Deskripsi Masalah

1.1 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.

1.2 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 :

1.2.1 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 C_{max} , C_{min} , 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 3×3 blok.

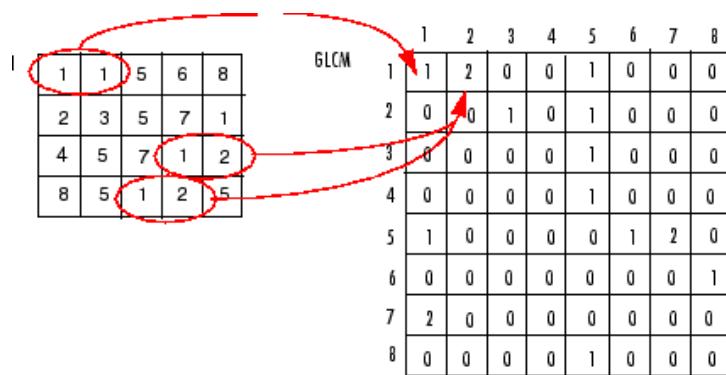
1.2.2 CBIR dengan parameter tekstur

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. 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).



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. *Grayscale* semula dari suatu gambar akan dikompresi untuk mengurangi operasi perhitungan sebelum dibentuknya *co-occurrence matrix*.

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:

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

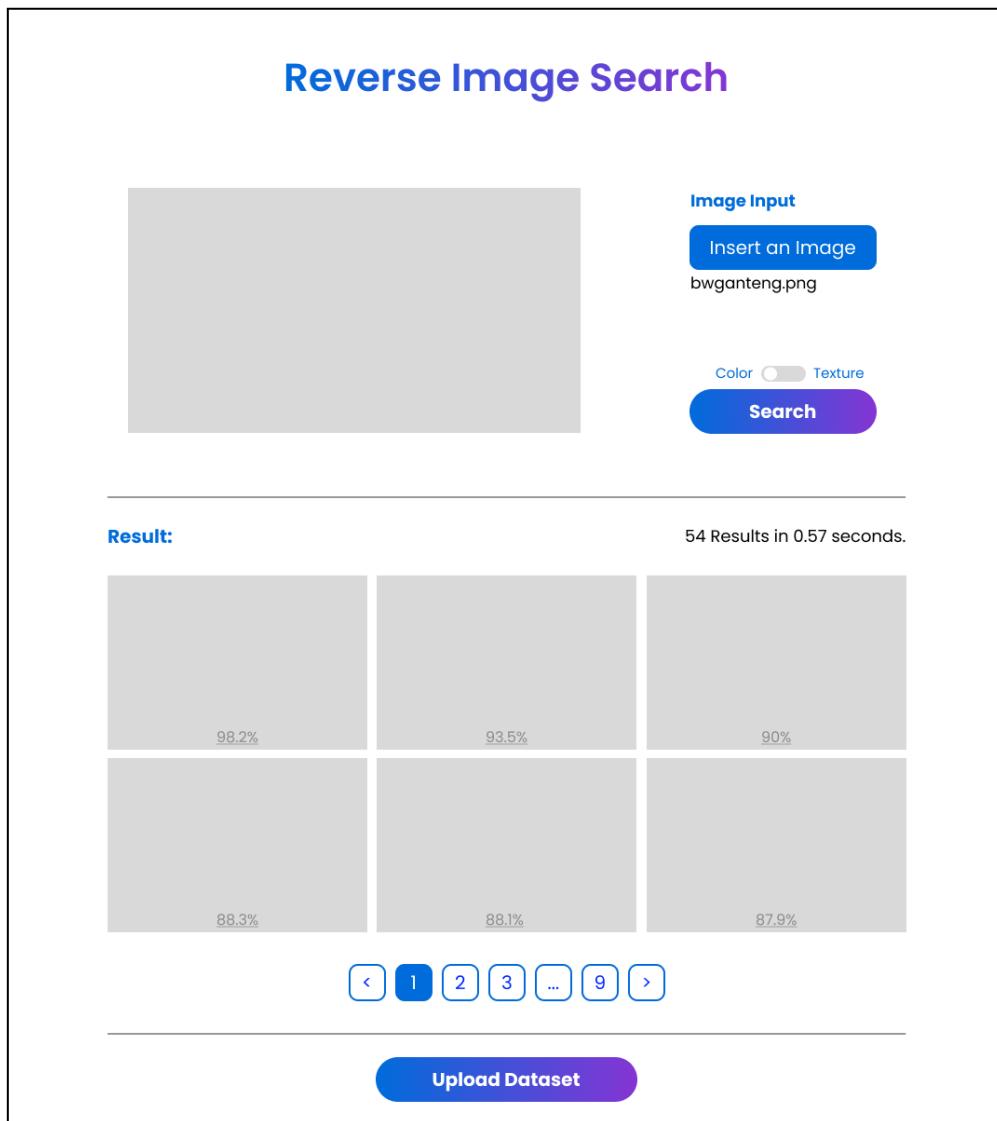
Disini A dan B adalah dua vektor dari dua gambar. Semakin besar hasil *Cosine Similarity* kedua vektor maka tingkat kemiripannya semakin tinggi.

1.3 PENGGUNAAN PROGRAM

Berikut ini adalah masukan yang akan diberikan oleh pengguna untuk eksekusi program.

1. *Image query*, berisi gambar yang akan digunakan untuk melakukan pencarian gambar.
2. Kumpulan gambar (dataset), dilakukan dengan cara mengunggah *multiple image* dalam bentuk *folder* ke dalam *web browser*. Setelah memasukkan *image query*, kumpulan gambar inilah yang akan diseleksi menjadi *result*.

Tampilan *layout* dari aplikasi *website* yang akan dibangun adalah sebagai berikut.



Gambar 3. Ilustrasi tampilan layout dari aplikasi *website*

Anda dapat menambahkan menu lainnya seperti gambar, logo, dan sebagainya. Tampilan Front-End dari website tidak harus sama persis dengan layout yang diberikan di atas, tetapi dibuat semenarik mungkin dan wajib mencakup komponen-komponen berikut:

- Judul Website
- Tombol Insert Gambar, beserta Display Gambar yang ingin dicari

- Toggle Button untuk memilih *searching* berdasarkan Warna atau Tekstur
- Tombol Search untuk memulai pencarian
- Kumpulan Gambar (result) yang didapat dari hasil pencarian
- Informasi mengenai Jumlah Result yang didapat dan Waktu Eksekusi
- Tingkat Kemiripan Setiap Gambar (result) dengan gambar yang ingin dicari, dalam persentase (%)
- Tombol Upload Dataset untuk mengunggah kumpulan gambar (dataset) dalam suatu folder
- Page-page tambahan: Konsep singkat search engine yang dibuat, How to Use, dan About us

Secara umum, berikut adalah cara umum penggunaan program:

1. Pengguna terlebih dahulu memasukkan dataset gambar dalam bentuk *folder* yang berisi kumpulan gambar. Dataset gambar ini diperlukan sebelum proses *searching* agar ada perbandingan untuk gambar yang ingin dicari.
2. Setelah dataset sudah dimasukkan, pengguna memasukkan sebuah gambar yang ingin *di-search* dari dataset.
3. Pilih opsi pencarian, ingin melakukan pencarian berdasarkan warna atau tekstur.
4. Tekan tombol *search*, program kemudian akan memproses, mencari gambar-gambar dari dataset yang memiliki kemiripan dengan gambar yang dimasukkan tadi.
5. 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.
6. Terdapat informasi terkait jumlah gambar yang muncul, dan waktu eksekusi programnya.

Bab II

Landasan Teori

2.1 Dasar Teori Secara Umum

2.1.1 CBIR dengan Parameter Warna

Content-Based image retrieval merupakan jenis CBIR yang paling sederhana dan penting. Warna merupakan bagian dari gambar yang paling jelas dan mudah untuk dilihat dan dibedakan. Jika dibandingkan dengan *feature* gambar lainnya, seperti *texture* dan *shape*, warna lebih stabil dan kuat. Warna tidak terpengaruh oleh rotasi, translasi, dan perubahan ukuran. Cara yang paling banyak digunakan untuk mengimplementasikan CBIR dengan parameter warna adalah dengan menggunakan histogram warna untuk mendapatkan *feature* warna dari gambar.

Histogram warna adalah frekuensi dari berbagai warna pada yang ada gambar untuk mengetahui pendistribusian warna pada gambar tersebut. Hal ini dapat dilakukan dengan membuat histogram warna dengan suatu nilai pada *range* tertentu dikuantifikasi lalu dijadikan sebagai *bin* pada histogram.

Dalam pembuatan histogram warna, ruang warna RGB akan diambil dari gambar dan diubah menjadi ruang warna HSV yang lebih umum digunakan. RGB sendiri adalah ruang warna yang terdiri dari tiga warna, yaitu merah, hijau, dan biru. Pallete warna ini biasanya digunakan oleh desainer dalam membuat produk digital. Sedangkan HSV adalah ruang warna yang mendefinisikan warna ke dalam tiga komponen, yaitu Hue, Saturation, dan Value. Keuntungan dari HSV adalah terdapat warna-warna yang sama dengan yang ditangkap oleh indra manusia. Selain itu juga, HSV dapat digunakan pada kertas (*background* warna putih). Oleh karena itu, dalam pembuatan histogram warna akan digunakan ruang warna HSV.

Terdapat dua jenis CBIR dengan parameter warna, yang pertama adalah dengan membuat histogram warna secara global dan yang kedua adalah dengan membuat histogram warna secara lokal. Keuntungan dari pembuatan histogram warna secara global adalah perhitungannya yang cepat dan tidak terpengaruh oleh rotasi dan translasi. Namun, histogram warna secara global juga memiliki beberapa kekurangan, seperti informasi lokasi dari distribusi warna tidak diperhitungkan. Dua gambar yang berbeda bisa memiliki histogram warna secara global yang sama.

Untuk membuat histogram warna secara lokal, gambar akan dibagi menjadi $n \times n$ blok. Setelah itu, histogram warna akan dibuat untuk masing-masing blok. Keuntungan dari membuat histogram warna secara lokal adalah informasi mengenai lokasi dari distribusi warna tetap tersimpan.

2.1.2 CBIR dengan Parameter Tekstur

Salah satu metode lain CBIR (*Content-Based Image Retrieval*) adalah dengan melakukan feature extraction terhadap fitur-fitur tekstur dari suatu gambar. Feature extraction dilakukan dengan membentuk sebuah matrix GLCM (Gray Level Co-occurrence Matrix) dan melakukan operasi-operasi terhadap matrix tersebut untuk mengekstraksi atau mendapatkan nilai fitur-fitur dari gambar. GLCM sendiri adalah Matrix yang menyatakan seberapa sering pasangan pixel yang memiliki hubungan

spasial tertentu (sudut dan jarak tertentu) dengan nilai Grayscale tertentu muncul. Matrix GLCM dibuat dengan membuat co-occurrence matrix dari suatu gambar dengan pixel grayscale, kemudian menjumlahkannya dengan transposenya sehingga mendapatkan *Symmetric Matrix*. *Symmetric Matrix* dinormalisasi untuk mendapatkan nilai matrix antara 0-1 dan menjadi GLCM.

Fitur-fitur tekstur yang dapat diekstraksi dari sebuah matrix GLCM antara lain: *contrast*, *dissimilarity*, *homogeneity*, *ASM*, *energy*, *correlation*, dan *entropy*. Dalam Tugas Besar ini, fitur tekstur yang akan dipakai adalah *contrast*, *homogeneity*, dan *entropy*. *Contrast* adalah fitur yang menggambarkan perbedaan nilai tertinggi dan terendah atau variasi lokal dari pixel-pixel yang berdekatan. *Homogeneity* adalah fitur yang menggambarkan kesamaan dari sebuah gambar. *Entropy* adalah fitur yang menggambarkan ketidakteraturan atau kompleksitas dari sebuah gambar.

2.2 Penjelasan Mengenai Pengembangan Sebuah Website

Website adalah kumpulan halaman yang digunakan untuk menampilkan informasi teks, gambar, animasi suara atau gabungan dari semuanya baik yang bersifat statis dan dinamis yang membentuk satu rangkaian saling terkait, yang masing-masing dihubungkan dengan jaringan terkait (Hidayat, 2010). Terdapat tiga teknologi dasar website: HTML (*Hyper Text Markup Language*), URL (*Uniform Resource Locator*), dan HTTP (*Hyper Text Transfer Protocol*). Berdasarkan sifat, jenis website dapat dibagi menjadi dua yaitu website dinamis dan website statis. Website Dinamis merupakan website yang memiliki konten yang selalu berubah setiap saat, seperti website *e-commerce*, website internet banking, dan juga website yang kita buat kali ini. Sedangkan Website Statis adalah website yang memiliki konten yang jarang berubah, seperti website landing page. Berdasarkan bahasa pemrograman, website dibagi menjadi dua yaitu *server side* dan *client side*. *Server side* merupakan website yang memerlukan server, jika tidak ada server, maka website yang dibangun menggunakan bahasa pemrograman tidak akan berjalan seperti yang seharusnya. Sedangkan *client side* adalah website yang tidak memerlukan server dalam menjalankannya.

Pada pengembangan sebuah website terdapat istilah *frontend* dan juga *backend*. *Frontend* adalah apa yang pengguna lihat pada tampilan sebuah website dan *backend* adalah sebuah sistem dibalik layar yang dapat mengolah database dan juga server. Bagian *frontend* sering juga disebut dengan *client-side* dan *backend* disebut dengan *server-side*. *Frontend* berperan pada komposisi tampilan pada sebuah website, mulai dari isi konten, warna font, jenis font, ukuran font, gambar, dan juga tombol-tombol. Sedangkan *backend* berperan pada sisi server, sistem, dan database. Beberapa bahasa pemrograman yang digunakan pada *frontend* adalah HTML (*Hyper Text Markup Language*), CSS (*Cascading Style Sheets*), dan Javascript. Pada bagian *backend*, beberapa bahasa pemrograman yang sering digunakan adalah PHP, Ruby, dan Python. Kemudian terdapat beberapa framework yang digunakan pada *frontend* adalah Angular.js, Tailwind, React.js, dll. Sedangkan framework yang sering digunakan untuk *backend* adalah Flask, GO, Express, Django, dll.

Bab III

Analisis Pemecahan Masalah

3.1 Langkah-Langkah Pemecahan Masalah

3.1.1 CBIR dengan Parameter Warna

Pada CBIR dengan parameter warna dimulai dengan mengekstrak nilai-nilai RGB dari gambar dan memasukkannya ke dalam matrix, kemudian mengubah matrix RGB tersebut menjadi matrix HSV. Setelah itu, bagi matrix HSV menjadi 4 x 4 blok dan lakukan perhitungan frekuensi warna dari masing-masing blok. Untuk mendapatkan tingkat kemiripan dari dua buah gambar, akan dilakukan *cosine similarity* dari masing-masing blok pada kedua gambar dengan posisi yang sama. Hasil *cosine similarity* dari masing-masing blok pada kedua gambar akan dirata-ratakan untuk mendapatkan tingkat kemiripan dari kedua gambar.

Pengambilan nilai RGB dari gambar menggunakan library PIL, dan diubah menjadi matrix RGB menggunakan library numpy melalui fungsi np.array. Kemudian, tiap elemen dari matrix RGB dinormalisasi dengan rumus:

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

lalu diubah menjadi HSV dengan rumus menggunakan fungsi np.where():

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

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

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

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

Setelah matrix RGB sudah diubah menjadi matrix HSV, elemen-elemen pada matrix HSV tersebut akan dikuantifikasi dengan menggunakan fungsi np.where() untuk memudahkan perhitungan dengan range sebagai berikut:

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

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

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

Kemudian bagi matrix tersebut menjadi 4×4 blok dengan menggunakan fungsi `np.array_split()`. Setelah matrix telah terbagi menjadi 4×4 blok, masing-masing blok akan dihitung frekuensi warnanya dengan menggunakan fungsi `np.histogram()`. Hasil dari histogram tiap blok dari 2 buah gambar akan dijadikan vektor untuk dilakukan *cosine similarity* dengan rumus:

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

Hasil *cosine similarity* akan dikali dengan 0.05 untuk 12 blok yang berada di luar dan 0.1 untuk blok yang berada di dalam. Kemudian akan dicari nilai rata-rata dari hasil *cosine similarity* tiap blok dan hasil rata-rata akan dikali 100% untuk mendapatkan tingkat kemiripan dari kedua gambar.

3.1.2 CBIR dengan Parameter Tekstur

Pada CBIR dengan parameter tekstur secara garis besar harus melakukan hal-hal berikut untuk mendapatkan hasil *cosine similarity* dari 2 buah gambar. Input gambar harus diubah menjadi Matrix RGB sehingga dapat diproses. Nilai-nilai pada tiap elemen matrix RGB diubah menjadi matrix grayscale yang dibulatkan menjadi *integer* untuk kemudian diproses menjadi matrix co-occurrence. Matrix co-occurrence diubah menjadi matrix symmetric, dan matrix symmetric diubah menjadi matrix GLCM. Dari matrix GLCM dapat dihitung nilai *contrast*, *homogeneity*, dan *entropy* dari gambar yang digunakan sebagai parameter *cosine similarity*.

Pengambilan nilai RGB dari gambar menggunakan library PIL, dan diubah menjadi matrix RGB menggunakan library numpy melalui fungsi np.array. Kemudian, tiap elemen dari matrix RGB harus diubah menjadi grayscale dengan rumus:

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

Hal ini dilakukan dengan melakukan dot product array [0.29, 0.587, 0.114] dengan tiap elemen pada matrix RGB menggunakan fungsi np.dot. Kemudian, setiap elemen pada matrix grayscale dibulatkan menjadi integer menggunakan astype(np.uint8).

Co-occurrence matrix didapatkan dengan melakukan looping pada setiap pasang elemen pada matrix grayscale yang memiliki hubungan berjarak 1 pada sudut 0°. Elemen Co-occurrence matrix akan di-increment jika kedua pasang pixel tersebut memiliki nilai grayscale sesuai index matrix co-occurrence.

Matrix *Symmetric* didapatkan dengan melakukan *transpose* pada matrix co-occurrence menggunakan np.transpose() dan menjumlahkannya dengan matrix co-occurrence awal. Setelah itu, dilakukan normalisasi pada matrix Symmetric sehingga mendapatkan matrix GLCM dengan cara membagi tiap elemen matrix symmetric dengan jumlah tiap elemen matrix Symmetric menggunakan (np.sum).

Setelah mendapatkan GLCM matrix, dapat dihitung nilai *contrast*, *homogeneity*, dan *entropy* dengan rumus masing sesuai dengan spesifikasi pada tiap elemen matrix, kemudian menjumlahkannya dengan np.sum(). Khusus untuk *entropy*, dibutuhkan fungsi np.where() agar bisa membagi kasus ketika log(Pi,j) tidak terdefinisi, yaitu ketika Pi,j = 0.

Setelah mendapatkan nilai *contrast*, *homogeneity*, dan *entropy*, ketiga fitur tersebut dimasukkan ke dalam suatu array/vektor untuk dilakukan *cosine similarity* dengan gambar lain. Rumus cosine similarity adalah dot product kedua vektor dibagi dengan hasil kali dari norma kedua vektor. Dot product dihitung dengan np.dot(), dan norma dihitung dengan np.norm().

3.2 Proses Pemetaan Masalah

3.2.1 CBIR dengan Parameter Warna

rgb_to_hsv(image_path)	Input: image path dari gambar yang ingin dihitung HSV-nya. Output: mengembalikan matrix HSV yang telah dibagi menjadi 4 x 4 blok
histogram(block)	Input: menerima input blok dari matrix Output: mengembalikan list yang berisi tuple vektor H, S, dan V dari hasil perhitungan frekuensi warna.
cosine_similarity_color(A, B)	Input: menerima 2 list tuple vektor H, S,

A, B = list tuple vektor H, S, dan V	dan V dari 2 gambar yang ingin dibandingkan kemiripannya. Output: mengembalikan tingkat kemiripan dari kedua gambar.
--------------------------------------	---

3.2.1.1 Membuka gambar dan mengubahnya menjadi matrix RGB

np.array(Image.open('filename')).convert('RGB')	<ul style="list-style-type: none"> • Image.open('filename'): membuka gambar dan memprosesnya. • convert('RGB'): mengirim salinan gambar menjadi dalam bentuk RGB. • np.array(): mengubah data RGB yang telah diekstrak dari gambar ke dalam bentuk array (matrix).
---	---

3.2.1.2 Mengubah nilai RGB pada matrix menjadi HSV

np.max(rgb, axis=2) dan np.min(rgb, axis=2) axis = 2 → artinya dimensi ketiga pada matrix	<ul style="list-style-type: none"> • np.max(): mengembalikan nilai maksimum dari setiap elemen pada matrix. • np.min(): mengembalikan nilai minimum dari setiap elemen pada matrix.
np.where(condition, x, result) x: array_like Contoh: <ul style="list-style-type: none"> • s = np.where(Cmax == 0, 0, np.where(Cmax != 0, delta/Cmax, 0)) 	<ul style="list-style-type: none"> • np.where(): berfungsi untuk mengembalikan index pada array berdasarkan kondisi yang diinginkan. Fungsi ini juga dapat digunakan untuk mengubah setiap elemen pada array berdasarkan kondisi yang diinginkan. • s = np.where(Cmax == 0, 0, np.where(Cmax != 0, delta/Cmax, 0)): mengubah elemen pada matrix s yang memenuhi kondisi Cmax=0 menjadi 0, dan memenuhi kondisi Cmax ≠ 0 menjadi $\frac{\text{delta}}{\text{Cmax}}$

3.2.1.3 Mengkuantifikasi nilai HSV berdasarkan range tertentu

np.logical_and(x, y)	<ul style="list-style-type: none"> • np.logical_and(): mengembalikan
----------------------	---

<pre>np.logical_or(x, y) x,y: array_like</pre> <p>Contoh:</p> <ul style="list-style-type: none"> • h0 = $\text{np.logical_or}(\text{np.logical_and}(316 \leq h, h \leq 360), h < 1)$ • s0 = np.logical_and(0 <= s, s < 0.2) • s1 = np.logical_and(0.2 <= s, s < 0.7) 	<p>nilai True atau False berdasarkan kondisi.</p> <ul style="list-style-type: none"> • np.logical_or(): mengembalikan nilai True atau False berdasarkan kondisi. • s0 akan menghasilkan matrix yang elemennya bernilai True untuk setiap elemen pada matrix s yang lebih besar sama dengan 0 dan lebih kecil dari 0.2
<pre>np.where(condition, x, result) x: array_like</pre> <p>Contoh:</p> <ul style="list-style-type: none"> • s = np.where(s0, 0, np.where(s1, 1, 2)) 	<ul style="list-style-type: none"> • s = np.where(s0, 0, np.where(s1, 1, 2)): berfungsi untuk mengubah elemen pada matrix s yang memenuhi kondisi s0 menjadi 0, elemen pada matrix s yang memenuhi kondisi s1 menjadi 1, dan sisanya menjadi 2.

3.2.1.4 Menggabungkan matrix H, S, dan V menjadi satu matrix

<pre>np.stack((h, s, v), axis = -1) axis = 0: first dimension axis = -1 last dimension</pre>	<ul style="list-style-type: none"> • np.stack(): digunakan untuk menyatukan matrix H, S, dan V menjadi satu. Axis = -1 menunjukkan matrix disatukan pada dimensi terakhir sebagai acuan.
--	---

3.2.1.5 Memecah matrix menjadi 4 x 4 blok

<pre>[np.array_split(row, 4, 1) for row in np.array_split(matrix, 4)]</pre>	<ul style="list-style-type: none"> • np.array_split(): berguna untuk membagi suatu array/matrix menjadi 2 atau lebih array/matrix yang berukuran hampir sama. Jika array/matrix tidak habis dibagi maka sisa hasil pembagian akan dimasukkan ke dalam bagian yang paling kiri. • [np.array_split(row, 4, 1) for row in np.array_split(matrix, 4)]: bagian ini berfungsi untuk membagi matrix menjadi 4 bagian pada baris dan 4 bagian pada kolom yang
---	---

	menghasilkan 16 blok yang berukuran hampir sama
--	---

3.2.1.6 Menghitung frekuensi warna pada masing-masing blok

```
histh = np.histogram(block[i][j][:,:,0],  
bins=[0,1,2,3,4,5,6,7,8])  
  
hists = np.histogram(block[i][j][:,:,1],  
bins=[0,1,2,3])  
  
histv = np.histogram(block[i][j][:,:,2],  
bins=[0,1,2,3])  
  
size = block[i][j][:,:,0].size  
histblock1 = (histh1/size, hists1/size,  
histv1/size)
```

- np.histogram(): digunakan untuk mendapatkan frekuensi warna dari masing-masing blok.
- bins=[] → digunakan untuk membagi nilai frekuensi histogram pada range tertentu
- block[i][j][:,:,0].size: digunakan untuk mendapatkan ukuran dari blok
- Hasil dari histogram tiap blok akan dinormalisasikan dengan membaginya dengan ukuran dari tiap blok

3.2.1.7 Menghitung cosine similarity dari 2 gambar

```
np.dot(A[j][i], B[j][i])  
np.sqrt(np.dot(A[j][i], A[j][i]))  
  
arr_weight =  
[0.05, 0.05, 0.05, 0.05,  
 0.05, 0.1, 0.1, 0.05,  
 0.05, 0.1, 0.1, 0.05,  
 0.05, 0.05, 0.05, 0.05]  
res = np.dot(res1, arr_weight)
```

- np.dot(): berfungsi untuk mendapat *dot product* dari 2 vektor.
- np.sqrt(np.dot(A[j][i], A[j][i])): berfungsi untuk mendapatkan nilai normal dari vektor.
- np.dot(res1, arr_weight): bagian ini berfungsi untuk mengalikan hasil *cosine similarity* dengan *weight coefficient*. *Coefficient* untuk 4 blok yang berada di tengah dua kali lebih besar dari 12 blok yang mengelilinginya.

3.2.2 CBIR dengan Parameter Tekstur

texture(image_path)

Input: image path dari gambar yang ingin dihitung Contrast, Homogeneity, dan Entropy-nya.
Output: vektor/array dengan bentuk [Contrast, Homogeneity, Entropy].

cosine_similarity(V1, V2)	Input: menerima vektor fitur texture. Output: nilai cosine_similarity dari vektor V1 dan V2.
---------------------------	---

3.2.2.1 Membuka gambar dan mengubahnya menjadi matrix RGB

np.array(Image.open('filename')).convert('RGB')	<ul style="list-style-type: none"> • <code>Image.open(filename)</code>: membuka gambar dengan nama 'filename' dan memprosesnya. • <code>convert('RGB')</code>: mengirim salinan gambar menjadi dalam bentuk RGB. • <code>np.array()</code>: mengubah data RGB yang telah diekstrak dari gambar ke dalam bentuk array (matrix).
---	---

3.2.2.2 Mengubah Matrix RGB menjadi Matrix Grayscale

np.array([0.29, 0.587, 0.114])	<ul style="list-style-type: none"> • Membuat array sebagai pengali tiap elemen RGB.
np.dot(rgb, P).astype(np.uint8)	<ul style="list-style-type: none"> • <code>np.dot(rgb, P)</code>: melakukan dot product dari Matrix RGB (rgb) dengan Matrix pengali (P) sehingga tiap elemen didapatkan nilai grayscalenya ($0.29*R + 0.587*G + 0.114*B$).

3.2.2.3 Membentuk Matrix co-occurrence

Y.shape	<ul style="list-style-type: none"> • <code>Y.shape</code>: Y adalah matrix grayscale dan akan di <i>return</i> dimensi/bentuk dari matrix grayscale (height, width).
MatrixOcc = np.zeros((256, 256), dtype='i')	<ul style="list-style-type: none"> • <code>Np.zeros((256,256), dtype='i')</code>: membentuk matrix dengan tiap elemennya bernilai 0 dan tipe datanya integer.
for i in range(h): for j in range(w-1): MatrixOcc[Y[i, j], Y[i, j+1]] += 1	<ul style="list-style-type: none"> • Melakukan looping untuk <i>increment</i> Matrix co-occurrence di index nilai grayscale pasangan pixel dengan sudut 0° dan jarak 1.

3.2.2.4 Membentuk Matrix Symmetric

Sym = MatrixOcc + MatrixOcc.transpose()	<ul style="list-style-type: none"> • MatrixOcc adalah Matrix co-occurrence. • Sym adalah Matrix Symmetric • transpose(): melakukan transpose terhadap matrix.
---	--

3.2.2.5 Membentuk Matrix GLCM (Normalisasi Matrix Symmetric)

np.divide(Sym, np.sum(Sym))	<ul style="list-style-type: none"> • np.sum: jumlah nilai tiap elemen dari matrix input. • np.divide: melakukan pembagian tiap elemen matrix di parameter pertama (Sym) dengan parameter kedua (np.sum(Sym)) sehingga mendapat matrix hasil normalisasi.
-----------------------------	--

3.2.2.6 Hitung Contrast, Homogeneity, dan Entropy

index = np.arange(Sym.shape[0])	<ul style="list-style-type: none"> • Membuat matrix berisi index untuk perhitungan Contrast, Homogeneity, dan Entropy.
Contrast = np.sum(np.square(index[:, None] - index) * Sym)	<ul style="list-style-type: none"> • np.sum: menjumlahkan tiap elemen matrix. • np.square: mengkuadratkan tiap elemen matrix. • Index[:, None] - index : menghitung i-j • Sym: matrix GLCM.
Homogeneity = np.sum(Sym / (1 + np.square(index[:, None] - index)))	<ul style="list-style-type: none"> • np.sum: menjumlahkan tiap elemen matrix. • np.square: mengkuadratkan tiap elemen matrix. • Index[:, None] - index : menghitung i-j
Entropy = -np.sum(np.where(Sym == 0, 0, Sym * np.log(Sym)))	<ul style="list-style-type: none"> • np.sum: menjumlahkan tiap elemen matrix. • np.log: menghitung nilai log dari elemen Matrix. • np.where: jika elemen matrix Sym == 0, maka 0, jika tidak maka nilai =

	Sym*np.log(Sym).
np.array([Contrast, Homogeneity, Entropy])	<ul style="list-style-type: none"> • Menggabungkan nilai Contrast, Homogeneity, dan Entropy menjadi satu array.

3.2.2.7 Menghitung nilai *cosine similarity*

np.dot(V1, V2)/(norm(V1)*norm(V2))	<ul style="list-style-type: none"> • np.dot: menghitung nilai dot dari V1 dan V2. • norm: menghitung panjang vektor/norma dari input. • Output yang dihasilkan adalah nilai cosine similarity dari V1 dan V2.
------------------------------------	--

3.3 Ilustrasi Kasus dan Penyelesaiannya

3.3.1 CBIR dengan Parameter Warna

3.3.1.1 Membuka gambar dan mengubahnya menjadi matrix RGB dan dinormalisasi

Matrix RGB:

```
[[[152 135 115]
  [152 135 115]
  [153 136 116]
  ...
  [161 144 116]
  [161 144 116]
  [161 144 116]]

  [[153 136 116]
  [153 136 116]
  [153 136 116]
  ...
  [161 144 116]
  [161 144 116]
  [160 143 115]]

  [[153 136 116]
  [153 136 116]
  [153 136 116]
  ...
  [161 144 116]
  [160 143 115]
  [160 143 115]]]
```

Matrix RGB yang telah dinormalisasi:

```
[[[0.59607843 0.52941176 0.45098039]
 [0.59607843 0.52941176 0.45098039]
 [0.6         0.53333333 0.45490196]
 ...
 [0.63137255 0.56470588 0.45490196]
 [0.63137255 0.56470588 0.45490196]
 [0.63137255 0.56470588 0.45490196]]
 [[0.6         0.53333333 0.45490196]
 [0.6         0.53333333 0.45490196]
 [0.6         0.53333333 0.45490196]
 ...
 [0.63137255 0.56470588 0.45490196]
 [0.63137255 0.56470588 0.45490196]
 [0.62745098 0.56078431 0.45098039]]
 [[0.6         0.53333333 0.45490196]
 [0.6         0.53333333 0.45490196]
 [0.6         0.53333333 0.45490196]
 ...
 [0.63137255 0.56470588 0.45490196]
 [0.62745098 0.56078431 0.45098039]
 [0.62745098 0.56078431 0.45098039]]
 ...
 ...
```

3.3.1.2 Mengubah nilai RGB pada matrix menjadi HSV

Matrix H:

```
[[32.43243243 32.43243243 32.43243243 ... 37.33333333 37.33333333
 37.33333333]
 [32.43243243 32.43243243 32.43243243 ... 37.33333333 37.33333333
 37.33333333]
 [32.43243243 32.43243243 32.43243243 ... 37.33333333 37.33333333
 37.33333333]
 ...
 [30.          30.          30.          ... 34.28571429 33.
 33.          ]
 [30.          30.          30.          ... 34.28571429 34.28571429
 34.28571429]
 [30.          30.          30.          ... 34.28571429 34.28571429
 34.28571429]]
```

Matrix S:

```
[[0.24342105 0.24342105 0.24183007 ... 0.27950311 0.27950311 0.27950311
 [0.24183007 0.24183007 0.24183007 ... 0.27950311 0.27950311 0.28125   ]
 [0.24183007 0.24183007 0.24183007 ... 0.27950311 0.28125     0.28125   ]
 ...
 [0.18934911 0.19047619 0.19161677 ... 0.26415094 0.25        0.2484472 ]
 [0.1871345  0.18823529 0.19047619 ... 0.26582278 0.26415094 0.26415094]
 [0.1871345  0.18823529 0.18934911 ... 0.26751592 0.26751592 0.26582278]]
```

Matrix V:

```
[[0.59607843 0.59607843 0.6 ... 0.63137255 0.63137255 0.63137255]
 [0.6 0.6 0.6 ... 0.63137255 0.63137255 0.62745098]
 [0.6 0.6 0.6 ... 0.63137255 0.62745098 0.62745098]
 ...
 [0.6627451 0.65882353 0.65490196 ... 0.62352941 0.62745098 0.63137255]
 [0.67058824 0.66666667 0.65882353 ... 0.61960784 0.62352941 0.62352941]
 [0.67058824 0.66666667 0.6627451 ... 0.61568627 0.61568627 0.61960784]]
```

3.3.1.3 Mengkuantifikasi nilai HSV berdasarkan range tertentu

Matrix H setelah dikuantifikasi:

```
[[2 2 2 ... 2 2 2]
 [2 2 2 ... 2 2 2]
 [2 2 2 ... 2 2 2]
 ...
 [2 2 2 ... 2 2 2]
 [2 2 2 ... 2 2 2]
 [2 2 2 ... 2 2 2]]
```

Matrix S setelah dikuantifikasi:

```
[[1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 ...
 [0 0 0 ... 1 1 1]
 [0 0 0 ... 1 1 1]
 [0 0 0 ... 1 1 1]]
```

Matrix V setelah dikuantifikasi:

```
[[1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 ...
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]
 [1 1 1 ... 1 1 1]]
```

3.3.1.4 Menggabungkan matrix H, S, dan V menjadi satu matrix

```
[ [ [2 1 1]
  [2 1 1]
  [2 1 1]
  ...
  [2 1 1]
  [2 1 1]
  [2 1 1] ] ]
```



```
[ [ [2 1 1]
  [2 1 1]
  [2 1 1]
  ...
  [2 1 1]
  [2 1 1]
  [2 1 1] ] ]
```



```
[ [ [2 1 1]
  [2 1 1]
  [2 1 1]
  ...
  [2 1 1]
  [2 1 1]
  [2 1 1] ] ]
```



```
... ...
```

3.3.1.5 Memecah matrix menjadi 4 x 4 blok

```
[[array([[2, 1, 1],
       [2, 1, 1],
       [2, 1, 1],
       ...,
       [2, 1, 1],
       [2, 1, 1],
       [2, 1, 1]],

      [[2, 1, 1],
       [2, 1, 1],
       [2, 1, 1],
       ...,
       [2, 1, 1],
       [2, 1, 1],
       [2, 1, 1]],

      [[2, 1, 1],
       [2, 1, 1],
       [2, 1, 1],
       ...,
       [2, 1, 1],
       [2, 1, 1],
       [2, 1, 1]],

      ...,

      [[2, 1, 1],
       [2, 1, 1],
       [2, 1, 1],
       ...,
       [2, 1, 1],
       [2, 1, 1],
       [2, 1, 1]]],
```

3.3.1.6 Menghitung frekuensi warna pada masing-masing blok

```
Histogram H dari block pertama: [ 0 178 15110 782 0 0 0 314]
Histogram S dari block pertama: [ 2152 14230 2]
Histogram V dari block pertama: [ 133 14918 1333]
```

3.3.1.7 Menghitung cosine similarity dari 2 gambar

Hasil cosine similarity dari kedua gambar: 18.87%

3.3.2 CBIR dengan Parameter Tekstur

3.3.2.1 Membuka gambar dan mengubahnya menjadi matrix RGB

```
Matrix RGB:  
[[[200 197 180]  
 [198 195 178]  
 [195 192 177]  
 ...  
 [185 168 140]  
 [181 163 139]  
 [175 159 134]]  
  
[[200 197 180]  
 [201 198 181]  
 [200 197 182]  
 ...  
 [198 178 151]  
 [193 176 150]  
 [187 171 146]]  
  
[[199 196 179]  
 [202 199 182]  
 [205 202 187]  
 ...  
 [196 177 147]  
 [193 173 148]  
 [185 168 142]]  
...
```

3.3.2.2 Mengubah Matrix RGB menjadi Matrix Grayscale

```
Vektor Pengali:  
[0.29 0.587 0.114]  
Matrix Grayscale:  
[[194 192 189 ... 168 164 159]  
 [194 195 194 ... 179 176 171]  
 [193 196 199 ... 177 174 168]  
 ...  
 [172 163 153 ... 67 67 68]  
 [173 164 159 ... 70 71 73]  
 [181 173 169 ... 75 78 82]]
```

3.3.2.3 Membentuk Matrix co-occurrence

```
Matrix co-occurrence:  
[[0 0 0 ... 0 0 0]  
 [1 1 0 ... 0 0 0]  
 [0 0 1 ... 0 0 0]  
 ...  
 [0 0 0 ... 0 0 0]  
 [0 0 0 ... 0 0 0]  
 [0 0 0 ... 0 0 0]]
```

3.3.2.4 Membentuk Matrix Symmetric

```
Matrix Symmetric:  
[[0 1 0 ... 0 0 0]  
 [1 2 0 ... 0 0 0]  
 [0 0 2 ... 0 0 0]  
 ...  
 [0 0 0 ... 0 0 0]  
 [0 0 0 ... 0 0 0]  
 [0 0 0 ... 0 0 0]]
```

3.3.2.5 Membentuk Matrix GLCM (Normalisasi Matrix Symmetric)

```
Matrix GLCM:  
[[0.0000000e+00 1.91108121e-06 0.0000000e+00 ... 0.0000000e+00  
 0.0000000e+00 0.0000000e+00]  
 [1.91108121e-06 3.82216243e-06 0.0000000e+00 ... 0.0000000e+00  
 0.0000000e+00 0.0000000e+00]  
 [0.0000000e+00 0.0000000e+00 3.82216243e-06 ... 0.0000000e+00  
 0.0000000e+00 0.0000000e+00]  
 ...  
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00  
 0.0000000e+00 0.0000000e+00]  
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00  
 0.0000000e+00 0.0000000e+00]  
 [0.0000000e+00 0.0000000e+00 0.0000000e+00 ... 0.0000000e+00  
 0.0000000e+00 0.0000000e+00]]
```

3.3.2.6 Hitung Contrast, Homogeneity, dan Entropy

```
Vektor Fitur Tekstur: [127.08134326 0.16798401 9.00894483]
```

```
Vektor Fitur Tekstur: [4.90986680e+02 1.84992556e-01 9.26063195e+00]
```

3.3.2.7 Menghitung nilai cosine similarity

```
Cosine Similarity: 0.9986523424
```

Bab IV

Implementasi dan Uji Coba

4.1 Program Utama

4.1.1 CBIR dengan Parameter Warna

1. Ambil nilai RGB dari gambar input menjadi sebuah matrix RGB.
2. Normalisasi nilai RGB dengan mengubah nilai range [0, 255] menjadi nilai range [0, 1].
 - a. $R' = \frac{R}{255}$ $G' = \frac{G}{255}$ $B' = \frac{B}{255}$
3. Cari nilai C_{max} , C_{min} , dan Δ . Kemudian nilai-nilai tersebut untuk mengubah nilai RGB menjadi nilai HSV dengan menggunakan rumus berikut:

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

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

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

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

4. Kuantifikasi masing-masing nilai pada HSV (Hue, Saturation, dan Value) pada suatu range tertentu.

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

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

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

5. Kemudian bagi matrix HSV tersebut menjadi 4×4 blok
6. Bentuk histogram H, S, dan V dari masing-masing blok dan gabungkan histogram tersebut menjadi suatu vektor.

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

7. Cari nilai *cosine similarity* dari vektor H, S, dan V tersebut dengan *weight coefficient* 0.05 pada bagian luar dan 0.1 pada bagian dalam.
8. Kemudian cari nilai rata-rata dari hasil *cosine similarity* dari vektor-vektor tersebut.

4.1.2 CBIR dengan Parameter Tekstur

1. Ambil nilai RGB dari gambar input menjadi sebuah matrix RGB.
2. Ubah matrix RGB menjadi matrix Grayscale menggunakan rumus $Y = 0.29*R + 0.587*G + 0.114*B$.
3. Bulatkan nilai grayscale menjadi integer pada setiap pixel.
4. Melakukan Looping pada tiap pasang pixel dengan hubungan spasial tertentu (dalam kasus ini sudut 0° dan jarak 1) dan menambahkan nilai 1 pada matrix co-occurrence berukuran 256×256 dengan index nilai grayscale dari pasangan pixel tersebut.
5. Membentuk *Symmetric Matrix* dengan menjumlahkan co-occurrence matrix pada step 4 dengan transposenya.

6. Membentuk GLCM dengan normalisasi *Symmetric Matrix* pada step 5, yaitu dengan membagi tiap elemen matrix dengan jumlah nilai tiap elemen matrix.
7. Melakukan ekstraksi nilai *contrast* dengan rumus:

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

8. Melakukan ekstraksi nilai *homogeneity* dengan rumus:

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

9. Melakukan ekstraksi nilai *entropy* dengan rumus

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

10. Bentuk sebuah vektor dengan komponen nilai *contrast*, *homogeneity*, dan *entropy*.

11. Ulangi step 1 - 10 untuk gambar lain.

12. Menghitung nilai *cosine similarity* dari 2 buah gambar dengan rumus:

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

13. Ulangi untuk tiap gambar pada dataset.

14. Menampilkan gambar *dataset* yang sudah di-*filter* dan di-*sorting* berdasarkan nilai *cosine similarity*.

4.2 Struktur Program

Framework yang digunakan untuk pengembangan website adalah Django. Alasan pemilihan framework tersebut adalah karena mudah dipahami serta komunitasnya cukup besar sehingga terdapat banyak sumber untuk belajar. Untuk keseluruhan program website berada pada folder **src**.

4.2.1 Struktur Folder

Struktur folder **src** kami adalah sebagai berikut:

```
algeo02
CBIR
    CBIR
        ...
    settings.py
```

```

urls.py
...
main
...
Static
    Color_Texture
    css
templates
...
forms.py
models.py
urls.py
views.py
media
    dataset
    inputimage
productionfiles
db.sqlite3
manage.py

```

Nama Folder / File	Penjelasan
algeo02	Folder untuk mengaktifkan <i>virtual environment</i> .
CBIR (yang berada dalam folder src)	Folder projek.
CBIR (yang berada dalam folder CBIR)	Folder yang berisi <i>settings</i> dalam projek ini, termasuk di dalamnya konfigurasi untuk root url, konfigurasi database, dll.
main	Folder app yang sebagian besarnya adalah program utama yang berjalan pada website.
static	Folder untuk menyimpan file css dan program utama CBIR dengan parameter warna dan tekstur.
templates	Folder yang berisi file html.
forms.py	Berisi program untuk menerima gambar yang diupload pengguna.
models.py	Berisi berbagai models yang digunakan pada program kami.

urls.py	Berisi aturan yang akan menyocokkan url yang diminta dengan tampilan website yang akan di- <i>render</i> pada views.py .
views.py	Menerima request dan mengembalikan response (termasuk mengolah data masukan ketika <i>method</i> POST) serta melakukan <i>render</i> pada tampilan website.
media	Folder yang berisi folder dataset dan inputimage.
dataset	Folder yang berisi kumpulan gambar dataset.
inputimage	Folder yang berisi gambar yang ingin dilakukan pencarian.
productionfiles	Sebenarnya folder ini digunakan untuk proses <i>deployment</i> tetapi kami tidak men- <i>deploy</i> website kami sehingga folder ini tidak begitu berpengaruh.
db.sqlite3	Database yang digunakan untuk menyimpan objek gambar masukan (gambar yang ingin dicari) dan gambar dataset sesuai dengan models yang telah dibuat pada models.py .
manage.py	Program untuk melakukan migrasi dan menjalankan server.

4.2.2 Struktur Program pada **models.py**

```

class ImageInput(models.Model):
    image = models.ImageField(upload_to='inputimage/', null=True)
    result = models.IntegerField(default=0)
    search_time = models.DecimalField(default=0, max_digits=9, decimal_places=3)

    def delete(self):
        self.image.delete()
        super().delete()

class ImageDataset(models.Model):
    image = models.ImageField(upload_to='dataset/', null=True)
    similarity = models.DecimalField(null=True, blank=True, max_digits=12, decimal_places=9)

```

Berdasarkan kode python di atas, terdapat dua buah model. Pertama adalah model untuk gambar masukan yaitu **ImageInput** yang memiliki atribut berupa:

- **image** → Nama file (gambar)
- **result** → Banyaknya hasil pencarian pada dataset yang memiliki kemiripan > 60% dengan gambar ini.

- **search_time** → Waktu yang diperlukan untuk melakukan proses pencarian pada dataset.

Sedangkan untuk model **ImageDataset** memiliki atribut sebagai berikut:

- **image** → Nama file (gambar)
- **similarity** → Tingkat kemiripan terhadap gambar masukan (skala 0 - 100).

4.2.3 Struktur Program pada views.py

Nama Fungsi	Penjelasan
index_view(request)	Melakukan render pada index.html
upload_form_img_input(request)	Menghapus seluruh objek pada database inputimg (jika ada) kemudian menyimpan gambar yang dimasukkan saat ini (gambar yang ingin dilakukan pencarian) dan melakukan render pada inputimg.html
upload_form_dataset(request)	Menghapus seluruh objek pada database dataset kemudian menyimpan kumpulan gambar dataset yang dimasukkan saat ini dan melakukan render pada inputdataset.html
display_result(request)	Melakukan proses perhitungan tingkat kemiripan gambar pada dataset dengan gambar masukan dan melakukan render pada displayresult.html
change_image(request)	Sama seperti upload_form_img_input , tetapi akan melakukan render pada changeimage.html yang kelak akan terhubung langsung dengan displayresult.html <i>*kalau yang inputimg.html akan terlebih dahulu menuju inputdataset.html</i>
change_dataset(request)	Sama seperti upload_form_dataset , tetapi akan melakukan render pada changedataset.html yang kelak akan terhubung langsung dengan displayresult.html <i>*kalau yang inputdataset.html dapat kembali pada inputimg.html dan kita tidak ingin hal tersebut terjadi karena tujuannya disini adalah mengganti dataset dan melakukan pencarian lagi dengan dataset</i>

	<i>yang telah diganti tersebut.</i>
how_to_use (request)	Melakukan render pada howtouse.html
learn_more (request)	Melakukan render pada learnmore.html
about_us (request)	Melakukan render pada aboutus.html

4.2.4 Struktur Program Pencarian pada Website

```

...
from main.static.Color_Texture.texture import cosine_similiarity, texture
from main.static.Color_Texture.color import cosine_similiarity_color, rgb_to_hsv,
histogram
...
def display_result(request):
    inputimage = ImageInput.objects.all()
    toggle = request.POST.get('toggle-checkbox')
    countSimilarity = 0

    if(toggle == 'tekstur'):
        start_search = time.time()

        isTekstur = True
        print("TEKSTUR")

        if(request.method == 'POST'):
            media_folder = os.path.abspath('media')

            image_folder = media_folder+'/inputimage/'
            # print(image_folder)
            for image in os.listdir(image_folder):
                image_path = image_folder+image
                # print(image_path)

                V1 = texture(image_path)

            dataset = ImageDataset.objects.all()

            for image in dataset:
                dataset_path = media_folder+'/'+str(image.image)
                # print(dataset_path)

                V2 = texture(dataset_path)
                similarity = cosine_similiarity(V1, V2)
                if(similarity>0.6):
                    countSimilarity+=1
                    # print(similarity)
                    # print(image.similarity)
                    image.similarity = similarity*100
                    image.save()
            end_search = time.time()
            search_time = end_search-start_search

```

```

        for img in inputimage:
            img.search_time = search_time
            img.result = countSimilarity
            img.save()
    else:
        start_search = time.time()
        isTekstur = False
        print("WARNA")

    if(request.method == 'POST'):

        media_folder = os.path.abspath('media')

        image_folder = media_folder+ '/inputimage/'
        # print(image_folder)
        for image in os.listdir(image_folder):
            image_path = image_folder+image
            # print(image_path)

            hsv1 = rgb_to_hsv(image_path)
            hist1 = histogram(hsv1)

        dataset = ImageDataset.objects.all()

        for image in dataset:
            dataset_path = media_folder+ '/' +str(image.image)
            # print(dataset_path)

            hsv2 = rgb_to_hsv(dataset_path)
            hist2 = histogram(hsv2)

            similarity = cosine_similiarity_color(hist1, hist2)
            if(similarity>0.6):
                countSimilarity+=1
                # print(similarity)
                # print(image.similarity)
                image.similarity = similarity*100
                image.save()

            end_search = time.time()
            search_time = end_search-start_search
            for img in inputimage:
                img.search_time = search_time
                img.result = countSimilarity
                img.save()

        dataset =
ImageDataset.objects.all().order_by('-similarity').filter(similarity__range=(60, 100))
        page = Paginator(dataset, 8)
        page_list = request.GET.get('page')
        page = page.get_page(page_list)
        for img in inputimage:
            countSimilarity = img.result
            search_time = img.search_time
            search_time = round(search_time,3)
            if(countSimilarity!=0):

```

```

if(countSimilarity%8 == 0):
    max_page = countSimilarity/8
else:
    max_page = int(countSimilarity/8)+1
else:
    max_page = 1
max_page = round(max_page)

print("Search time:", search_time)
context = {'dataset':dataset,
           'inputimage':inputimage,
           'isTekstur':isTekstur,
           'page':page,
           'search_time':search_time,
           'countSimilarity':countSimilarity,
           'max_page':max_page,}
return render(request, 'displayresult.html',context)

```

4.3. Tata Cara Penggunaan Program

4.3.1 Mengaktifkan *Virtual Environment*

Pertama-tama kita perlu mengaktifkan *Virtual Environment* terlebih dahulu. Masukkan command berikut pada *command prompt* di dalam folder src.

Windows:

```
algeo02\Scripts\activate.bat
```

Unix/MacOS:

```
source algeo02/Scripts/activate
```

4.3.2 Install Django

Lakukan instalasi Django pada *command prompt*.

```
pip install Django
```

4.3.3 Install Numpy

Lakukan instalasi library numpy pada *command prompt*.

```
pip install numpy
```

4.3.4 Install Pillow

Lakukan instalasi library pillow pada *command prompt*.

```
pip install Pillow
```

4.3.5 Install Widget Tweaks

Lakukan instalasi Widget Tweaks pada *command prompt* untuk mendukung keberjalanannya keseluruhan program.

```
pip install django-widget-tweaks
```

4.3.6 Install Whitenoise

Lakukan instalasi Whitenoise pada *command prompt* untuk mendukung keberjalanannya keseluruhan program.

```
pip install whitenoise
```

4.3.7 Menjalankan Server

Untuk menjalankan server, dari folder src **pindah terlebih dahulu ke folder CBIR**. Setelah itu jalankan command berikut pada *command prompt* untuk menjalankan server.

```
python manage.py runserver
```

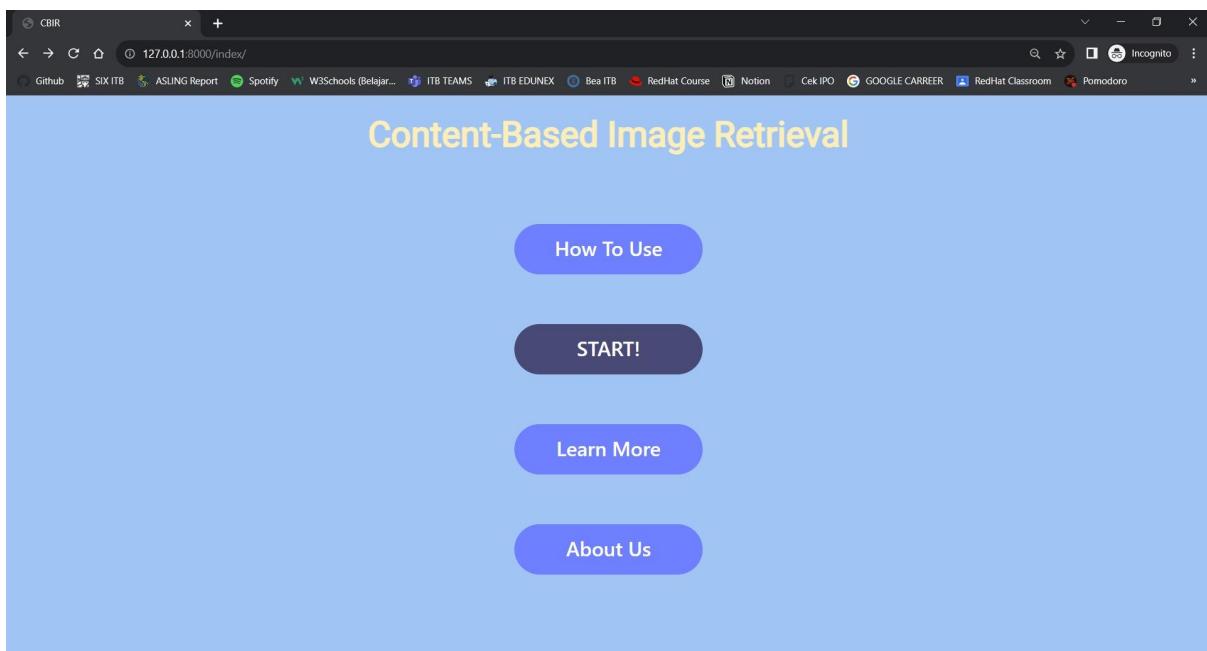
Setelah server berjalan selanjutnya akses website pada alamat development server yang tertera pada *command prompt*. Untuk gambar dibawah ini berarti harus mengakses website pada alamat <http://127.0.0.1:8000/>

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
November 19, 2023 - 13:51:32
Django version 4.2.7, using settings 'CBIR.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

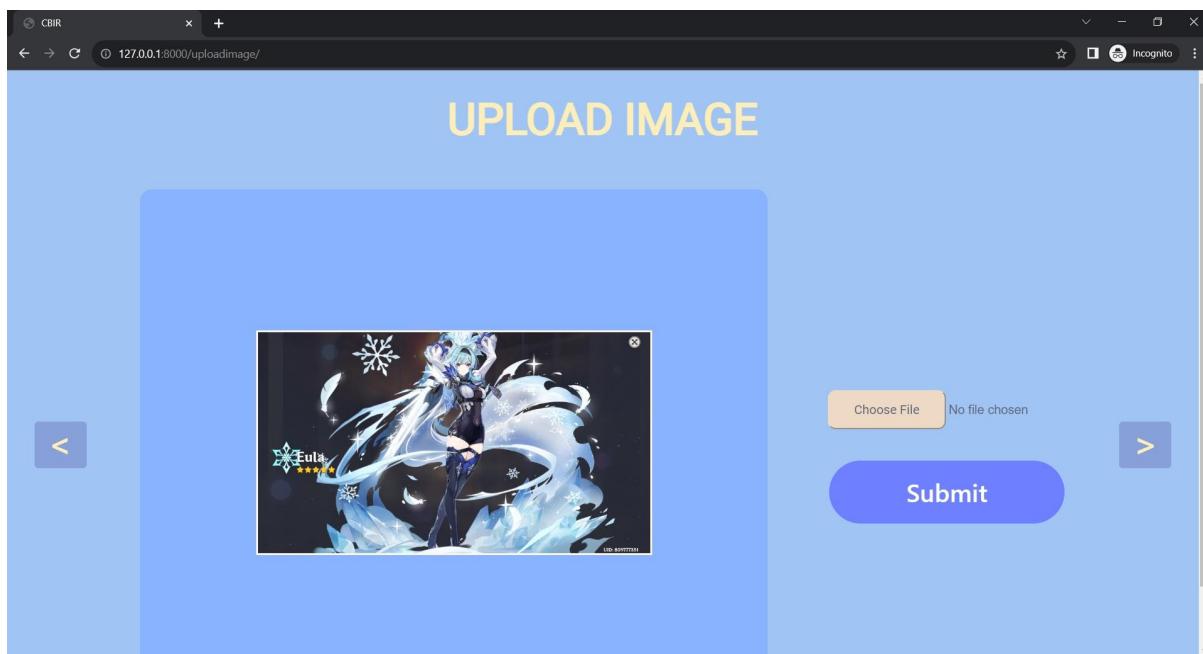
4.3.8 Homepage

Setelah mengakses alamat website, akan muncul tampilan homepage pada website. Untuk memulai melakukan proses pencarian gambar, tekan tombol **START!**.



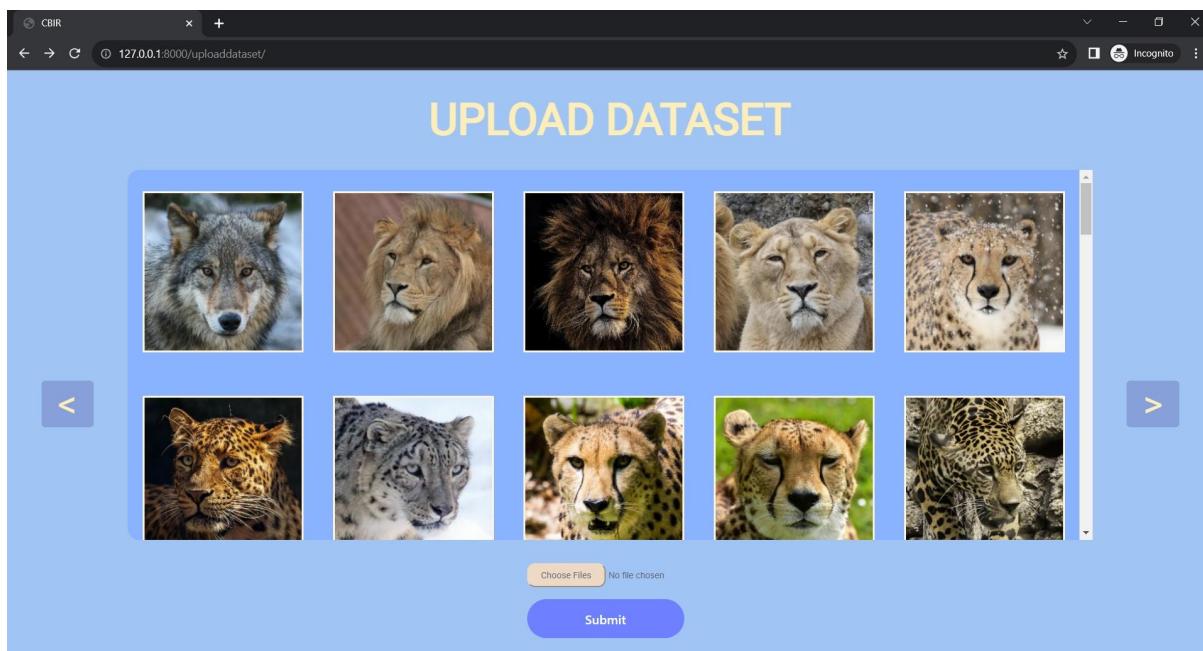
4.3.9 Upload Image

Pada bagian ini pengguna diminta untuk memasukkan gambar yang ingin dicari. Pengguna menekan tombol **choose file** terlebih dahulu kemudian pilih gambar yang akan digunakan. Setelah selesai, tekan tombol **Submit** dan tekan tombol tanda panah ke kanan untuk melanjutkan.



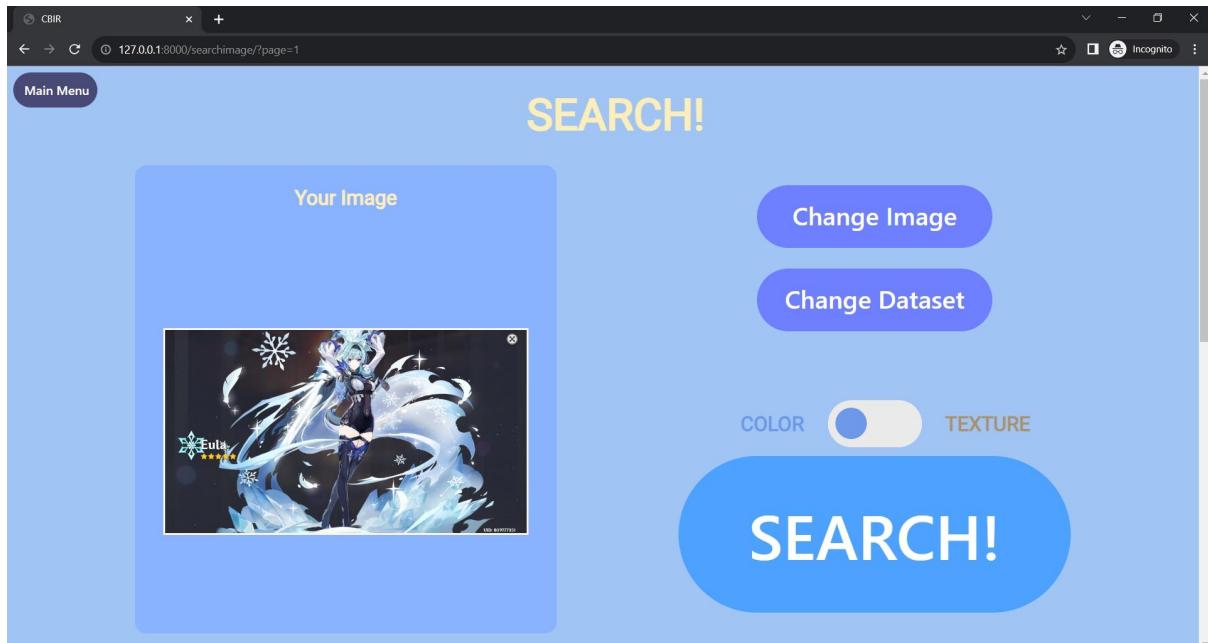
4.3.10 Upload Dataset

Upload kumpulan gambar yang akan dijadikan dataset dalam temu balik gambar masukan. Pengguna menekan tombol **choose file** terlebih dahulu kemudian pilih kumpulan gambar yang akan digunakan sebagai dataset. Setelah selesai, tekan tombol **Submit** dan tekan tombol tanda panah ke kanan untuk melanjutkan.



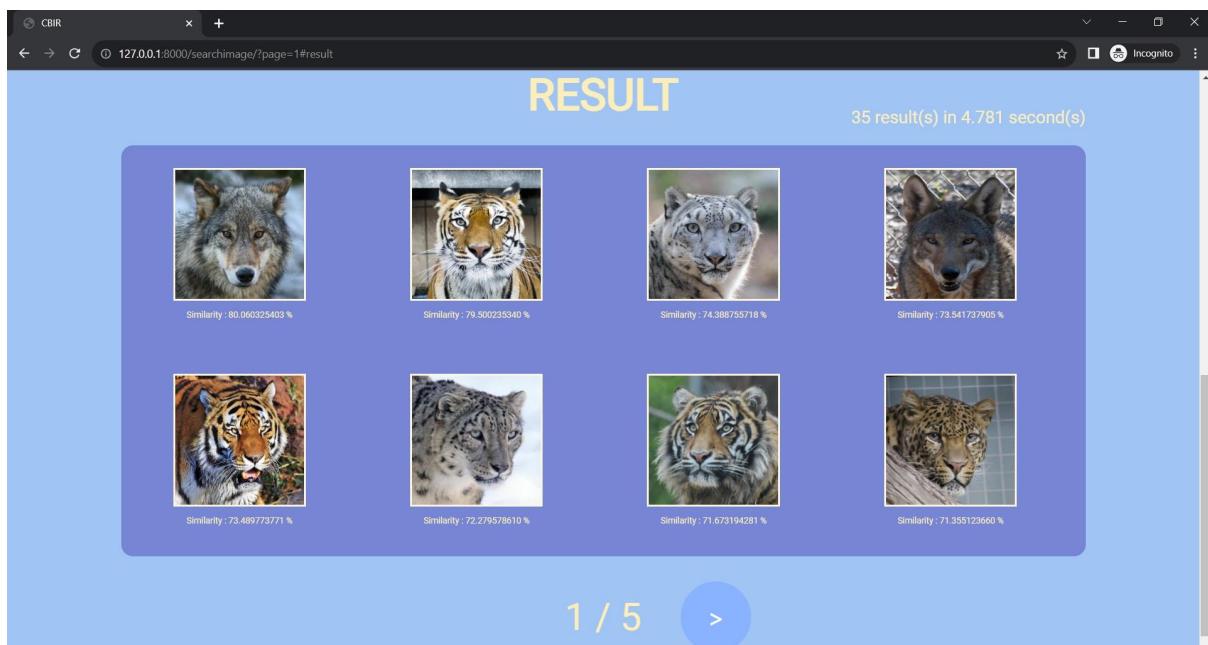
4.3.11 Pencarian

Pada halaman pencarian terdapat beberapa fitur seperti mengganti gambar masukan, mengganti dataset, dan fitur pencarian gambar masukan. Untuk fitur pencarian gambar masukan terdapat dua pilihan: pencarian berdasarkan kemiripan warna dan pencarian berdasarkan kemiripan tekstur pada gambar.



Gambar Halaman Pencarian Gambar

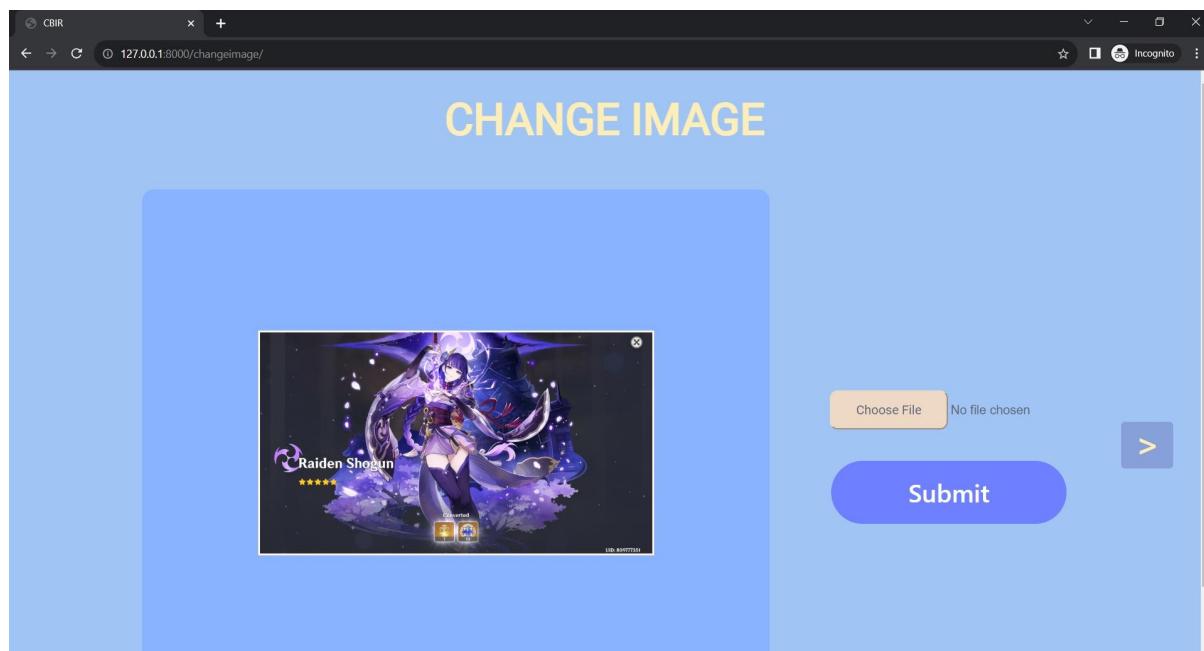
Setelah melakukan pencarian dan gambar pada dataset telah diproses, maka hasil berupa kumpulan gambar dari dataset yang memiliki tingkat kemiripan dengan gambar masukan lebih dari 60% akan ditampilkan beserta dengan persentase kemiripannya.



Gambar Hasil Pencarian Berdasarkan Warna

4.3.12 Change Image

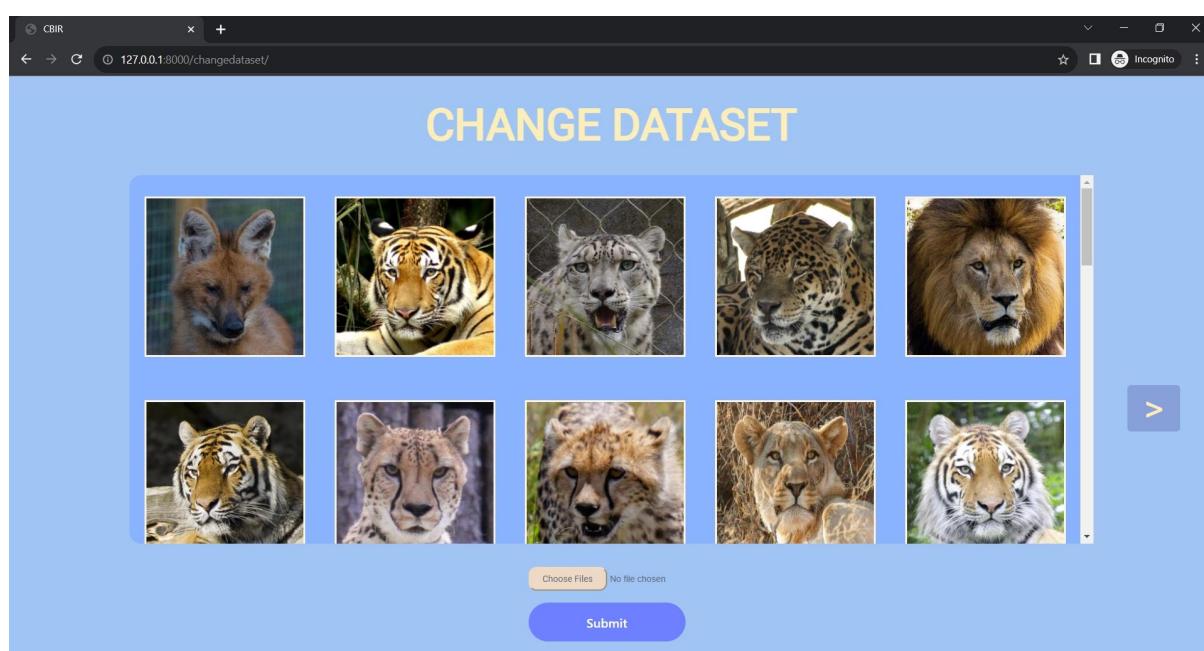
Pada bagian ini, pengguna dapat mengubah gambar masukan yang ingin dicari tanpa mengubah dataset yang sebelumnya sudah dimasukkan.



Gambar Halaman *Change Image*

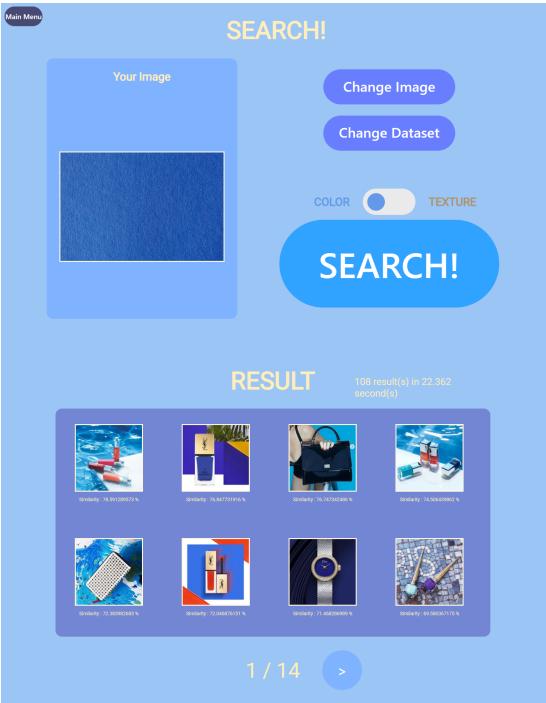
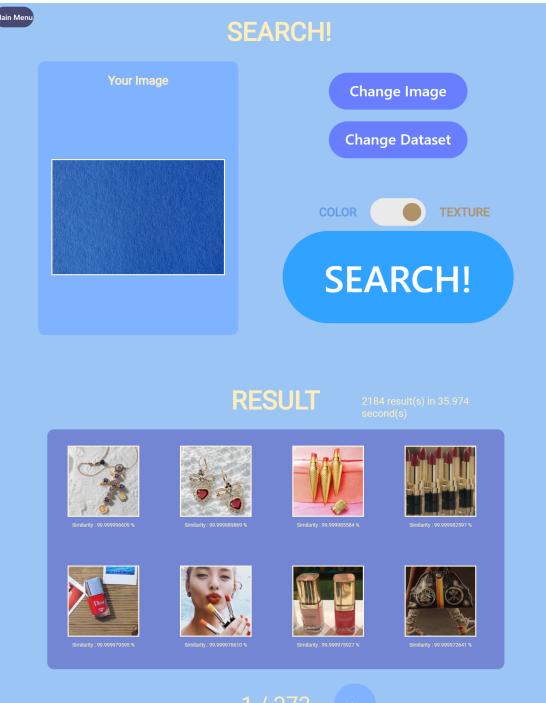
4.3.13 Change Dataset

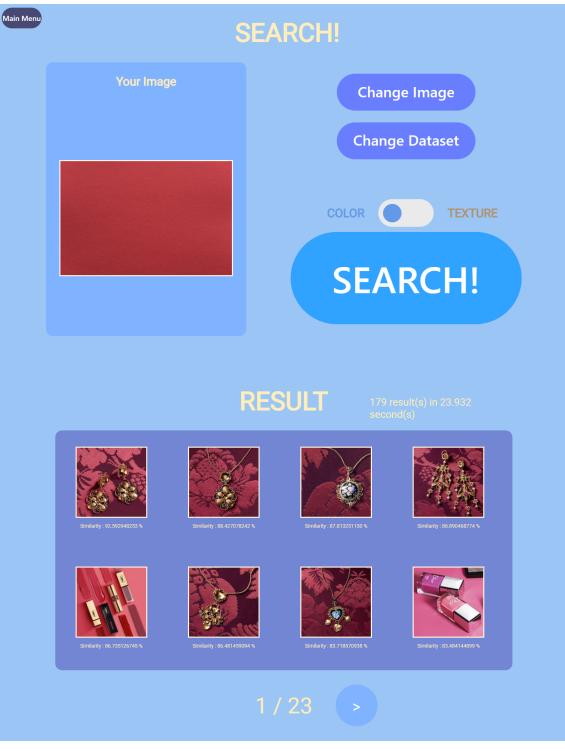
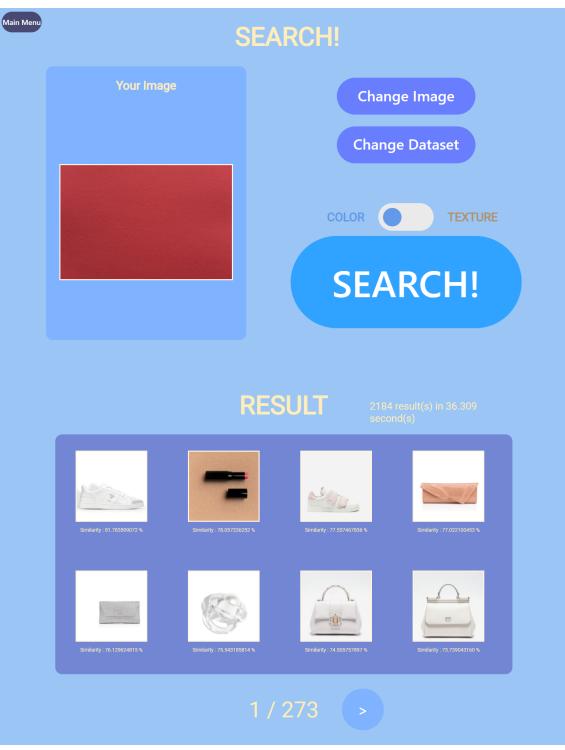
Pada bagian ini, pengguna dapat mengganti dataset yang ingin digunakan tanpa mengubah gambar masukan.



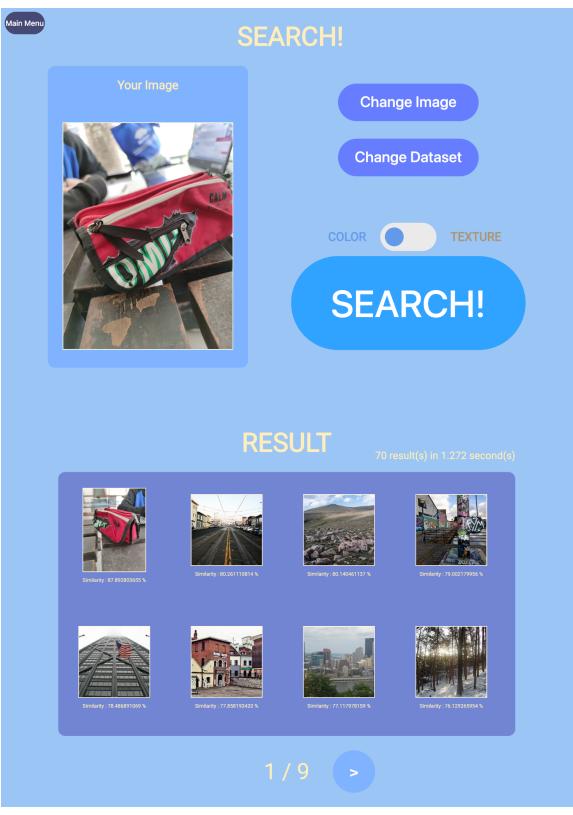
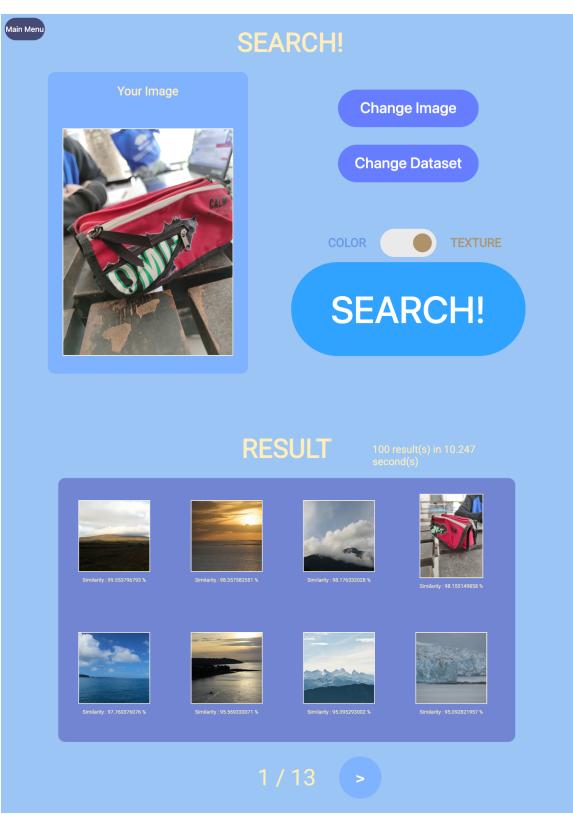
Gambar Halaman *Change Dataset*

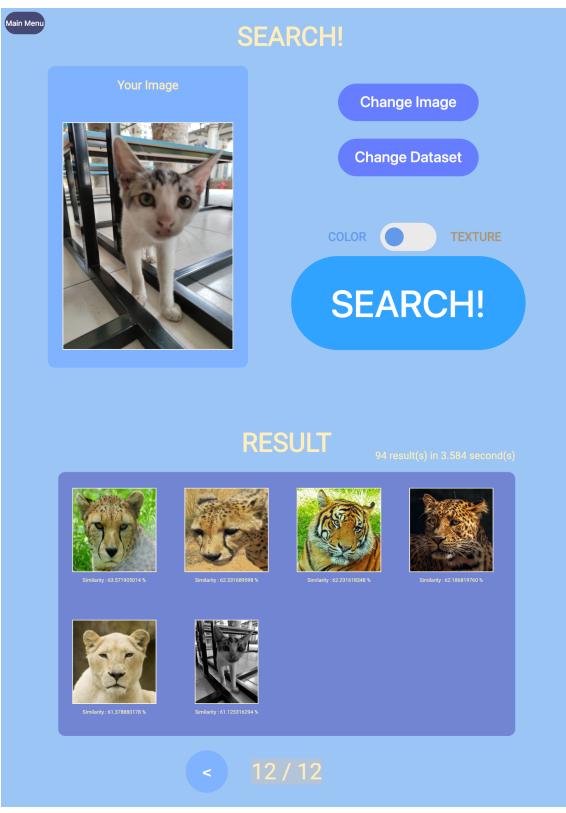
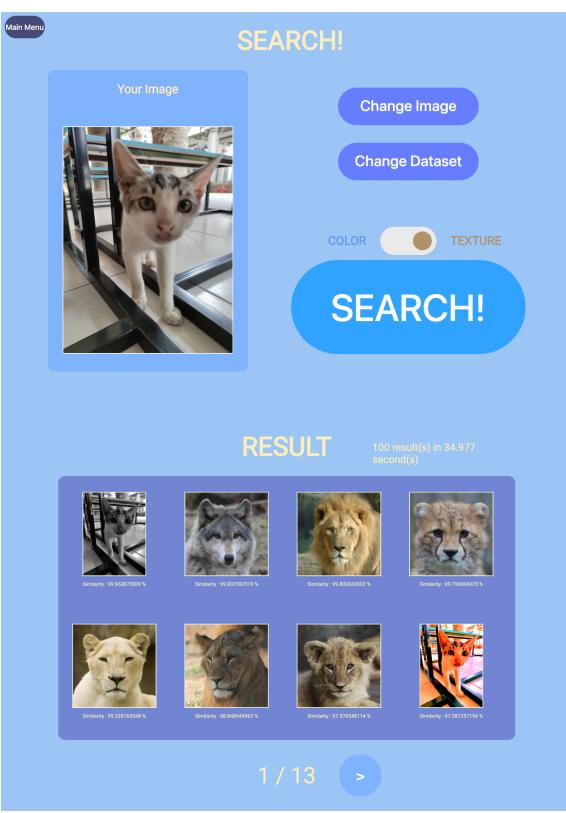
4.4 Hasil Pengujian

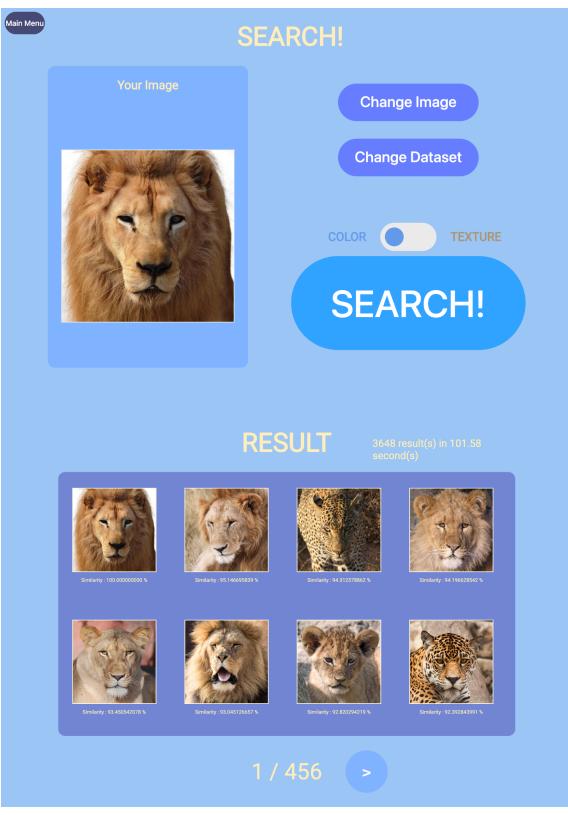
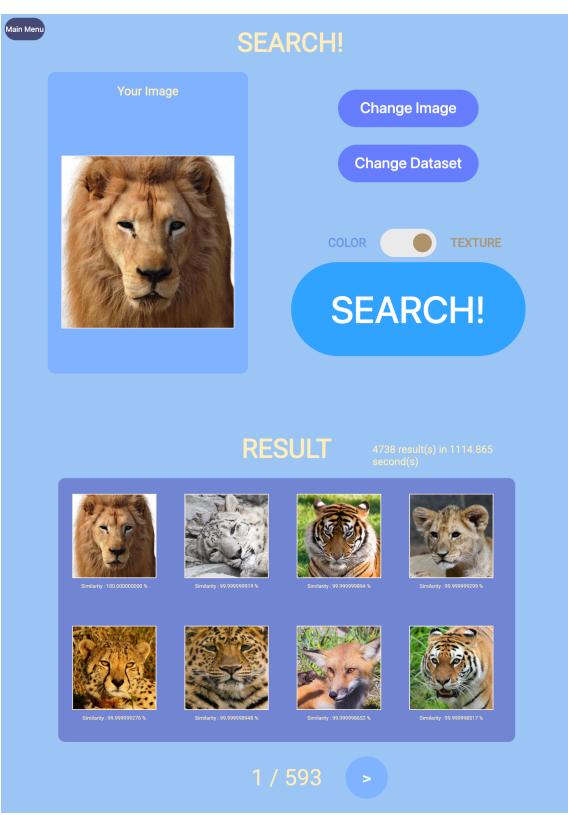
NO	HASIL PENGUJIAN	PENJELASAN
1		COLOR Pencarian gambar warna biru pada dataset 3 dengan metode color hasilnya menampilkan gambar-gambar lain yang berwarna biru juga.
2		TEXTURE Pencarian gambar warna biru bertekstur pada dataset 3 menampilkan gambar-gambar yang cukup random karena gambar awal tidak memiliki tekstur yang jelas.

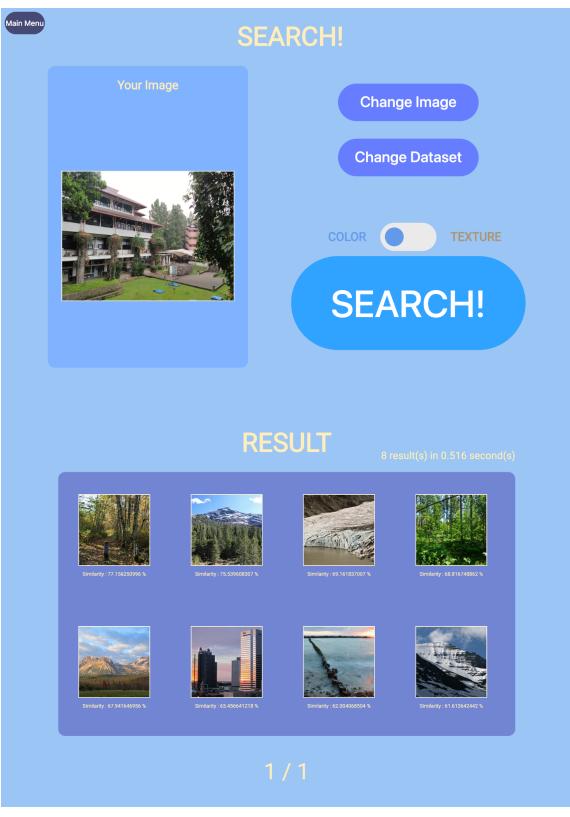
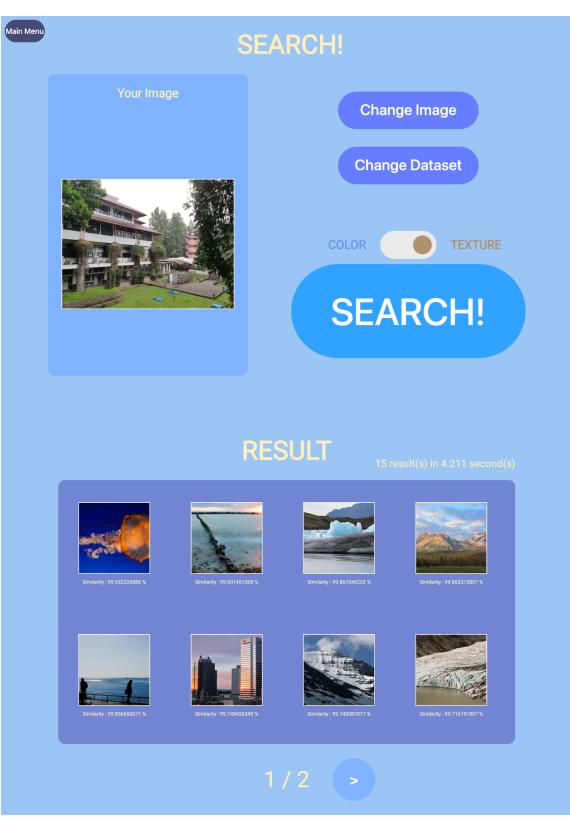
3	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image Change Dataset</p> <p>COLOR <input checked="" type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT 179 result(s) in 23.932 second(s)</p> <table border="1"> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 0.9323150%</td> <td>Similarity: 0.84270194%</td> <td>Similarity: 0.81332115%</td> <td>Similarity: 0.8004877%</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 0.8110039%</td> <td>Similarity: 0.8110039%</td> <td>Similarity: 0.7111000%</td> <td>Similarity: 0.6004469%</td> </tr> </tbody> </table> <p>1 / 23 ></p>					Similarity: 0.9323150%	Similarity: 0.84270194%	Similarity: 0.81332115%	Similarity: 0.8004877%					Similarity: 0.8110039%	Similarity: 0.8110039%	Similarity: 0.7111000%	Similarity: 0.6004469%	<h3>COLOR</h3> <p>Pencarian gambar warna merah pada dataset 3 dengan metode color hasilnya menampilkan gambar-gambar lain yang berwarna merah juga.</p>
																		
Similarity: 0.9323150%	Similarity: 0.84270194%	Similarity: 0.81332115%	Similarity: 0.8004877%															
																		
Similarity: 0.8110039%	Similarity: 0.8110039%	Similarity: 0.7111000%	Similarity: 0.6004469%															
4	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image Change Dataset</p> <p>COLOR <input type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT 2184 result(s) in 36.309 second(s)</p> <table border="1"> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 0.00094072%</td> <td>Similarity: 76.05720532%</td> <td>Similarity: 77.05161050%</td> <td>Similarity: 77.02215693%</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 76.12861401%</td> <td>Similarity: 75.54010814%</td> <td>Similarity: 74.00517981%</td> <td>Similarity: 75.75904316%</td> </tr> </tbody> </table> <p>1 / 273 ></p>					Similarity: 0.00094072%	Similarity: 76.05720532%	Similarity: 77.05161050%	Similarity: 77.02215693%					Similarity: 76.12861401%	Similarity: 75.54010814%	Similarity: 74.00517981%	Similarity: 75.75904316%	<h3>TEXTURE</h3> <p>Pencarian gambar warna merah pada dataset 3 menampilkan gambar-gambar yang mayoritas berwarna putih dan memiliki background yang polos.</p>
																		
Similarity: 0.00094072%	Similarity: 76.05720532%	Similarity: 77.05161050%	Similarity: 77.02215693%															
																		
Similarity: 76.12861401%	Similarity: 75.54010814%	Similarity: 74.00517981%	Similarity: 75.75904316%															

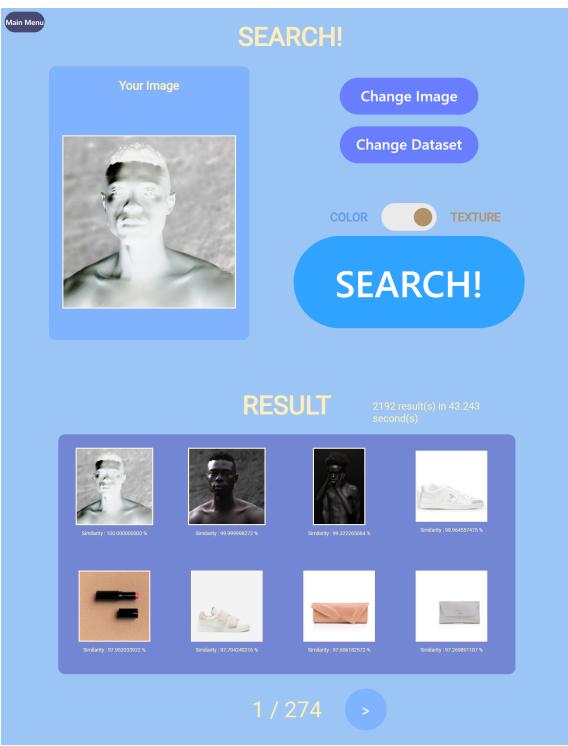
5		<h3>COLOR</h3> <p>Pencarian dengan input foto kucing pda pencarian dengan metode color berhasil menampilkan foto kucing yang sama dari sudut pandang yang berbeda dengan tingkat kemiripan yang paling besar.</p>
6		<h3>TEXTURE</h3> <p>Pencarian kucing yang difoto dari sudut pandang berbeda pada pencarian tekstur di dataset 1 didapatkan pada page 8 dari 13 (62 dari 100 gambar).</p>

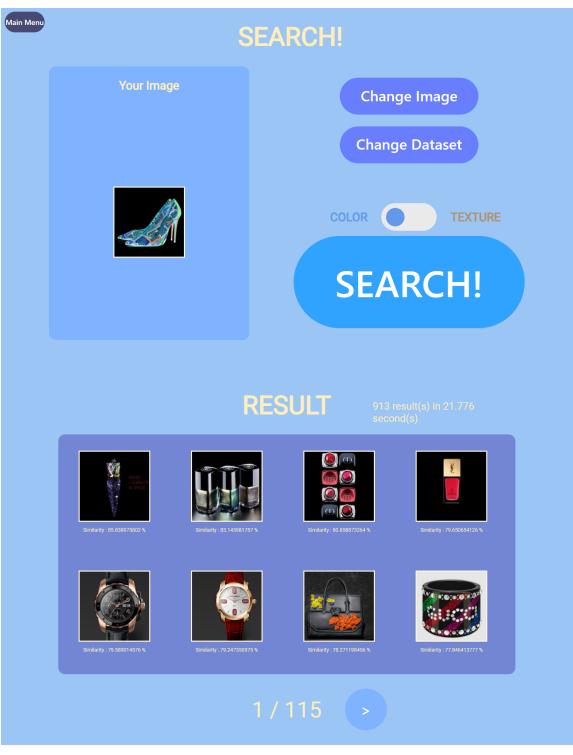
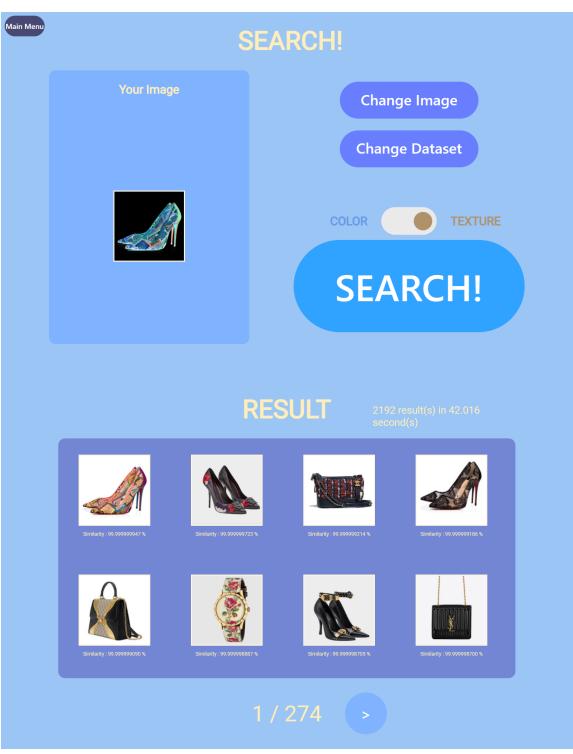
7	 <p>The screenshot shows the search interface with the 'COLOR' mode selected. The search term is a red pencil case with 'GOLF' written on it. The results page displays 70 images, each with a similarity score. The first few results are:</p> <ul style="list-style-type: none"> Similarity: 87.81031005 % Similarity: 85.201110814 % Similarity: 85.145461137 % Similarity: 79.023179956 % Similarity: 78.498471005 % Similarity: 77.89010432 % Similarity: 77.111920118 % Similarity: 76.132329954 % <p>Below the results is a navigation bar with '1 / 9' and a right arrow.</p>	<h3>COLOR</h3> <p>Pencarian dengan input foto tempat pensil dengan pencarian color berhasil menampilkan foto tempat pensil yang sama dari sudut pandang yang berbeda dengan tingkat kemiripan yang paling besar.</p>
8	 <p>The screenshot shows the search interface with the 'TEXTURE' mode selected. The search term is the same red pencil case. The results page displays 100 images, each with a similarity score. The first few results are:</p> <ul style="list-style-type: none"> Similarity: 99.353794792 % Similarity: 98.351782331 % Similarity: 95.176320208 % Similarity: 88.155149898 % Similarity: 97.783276776 % Similarity: 95.699323271 % Similarity: 95.693200002 % Similarity: 95.053221997 % <p>Below the results is a navigation bar with '1 / 13' and a right arrow.</p>	<h3>TEXTURE</h3> <p>Pencarian input foto tempat pensil berhasil menampilkan tempat pensil yang sama dari sudut pandang berbeda dengan pencarian tekstur pada posisi ke 4 dari 100 foto.</p>

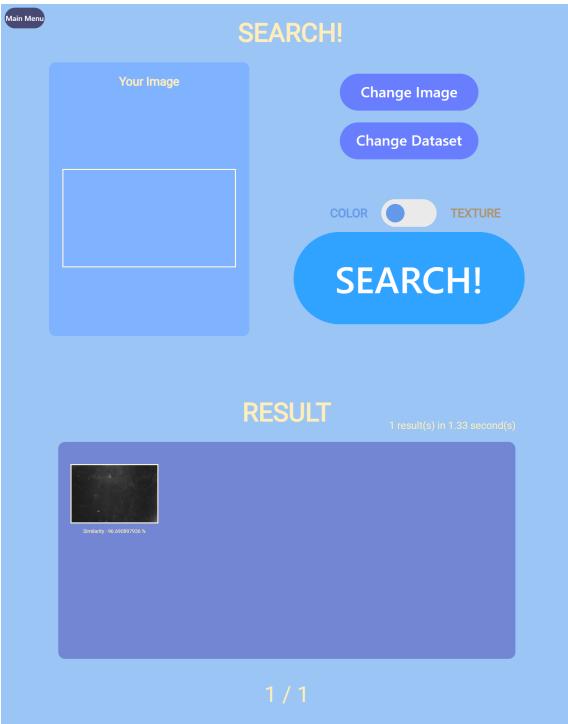
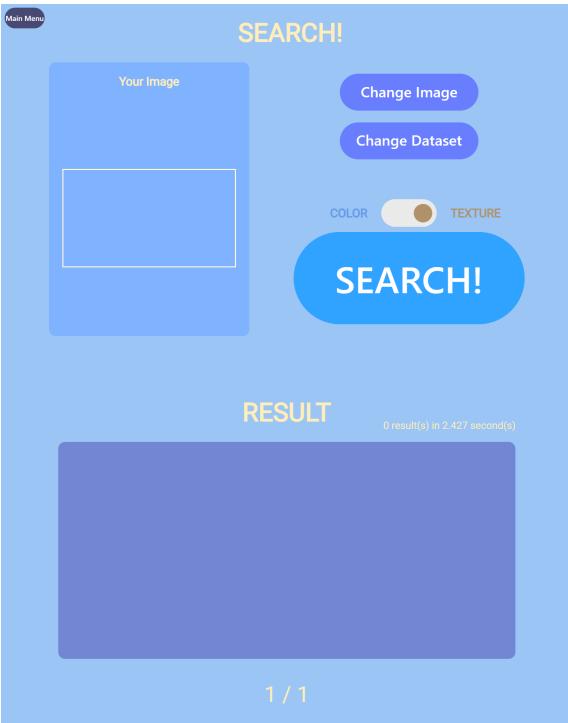
9	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image</p> <p>Change Dataset</p> <p>COLOR <input checked="" type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT</p> <p>94 result(s) in 3.584 second(s)</p> <p>Similarity: 41.57145014 %</p> <p>Similarity: 42.23169291 %</p> <p>Similarity: 42.72155048 %</p> <p>Similarity: 42.18847970 %</p> <p>Similarity: 41.21988317 %</p> <p>Similarity: 41.12031294 %</p> <p>12 / 12</p>	<h3>COLOR</h3> <p>Pencarian dengan input foto kucing dengan metode color berhasil menampilkan foto kucing yang sama yang telah dikenakan filter hitam putih dengan tingkat kemiripan 61% dan ada di halaman terakhir. Namun, foto kucing yang telah diberi filter contrast, saturation, vibrance, dan warmth tidak ditemukan</p>
10	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image</p> <p>Change Dataset</p> <p>COLOR <input type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT</p> <p>100 result(s) in 34.977 second(s)</p> <p>Similarity: 99.62887280 %</p> <p>Similarity: 99.62775151 %</p> <p>Similarity: 99.80344002 %</p> <p>Similarity: 99.79362482 %</p> <p>Similarity: 99.52919340 %</p> <p>Similarity: 98.84849982 %</p> <p>Similarity: 97.37046014 %</p> <p>Similarity: 97.28123779 %</p> <p>1 / 13</p>	<h3>TEXTURE</h3> <p>Pencarian dengan input foto kucing yang diberi filter black and white dapat ditemukan oleh pencarian tekstur di posisi pertama dari 100 foto.</p> <p>Pencarian foto kucing yang diberi <i>filter contrast, saturation, vibrance, dan warmth</i> juga masih dapat ditemukan di page 1 (posisi 8 dari 100 foto).</p>

11	 <p>The screenshot shows a search interface with a blue header. It features a "Main Menu" button at the top left. In the center, there's a "SEARCH!" button above a "Your Image" input field containing a lion's face. To the right of the input field are three buttons: "Change Image", "Change Dataset", and a toggle switch between "COLOR" and "TEXTURE". Below these is another "SEARCH!" button. The word "RESULT" is displayed in yellow at the top of the results section. It shows 3648 results in 101.58 seconds. The results are a grid of eight images, each with its similarity score below it. The first row includes a lion, a leopard, and two other lions. The second row includes a lion, a tiger cub, and two leopards. The page number "1 / 456" is shown at the bottom left.</p>	<h3>COLOR</h3> <p>Pencarian gambar singa yang sama persis pada dataset 1 berhasil ditemukan pada posisi pertama dan kemiripan 100% pada pencarian color. Pada halaman pertama juga menampilkan gambar-gambar singa lain yang berbeda.</p>
12	 <p>The screenshot shows a search interface with a blue header. It features a "Main Menu" button at the top left. In the center, there's a "SEARCH!" button above a "Your Image" input field containing a lion's face. To the right of the input field are three buttons: "Change Image", "Change Dataset", and a toggle switch between "COLOR" and "TEXTURE". Below these is another "SEARCH!" button. The word "RESULT" is displayed in yellow at the top of the results section. It shows 4738 results in 1114.865 seconds. The results are a grid of eight images, each with its similarity score below it. The first row includes a lion, a snow leopard, a tiger, and a lion cub. The second row includes a cheetah, a leopard, a fox, and a tiger. The page number "1 / 593" is shown at the bottom left.</p>	<h3>TEXTURE</h3> <p>Pencarian gambar singa yang sama persis pada dataset 1 berhasil ditemukan pada posisi pertama dan kemiripan 100% pada pencarian tekstur, walaupun pada halaman pertama tidak ditemukan gambar singa lainnya.</p>

13	 <p>The screenshot shows a search interface with a blue header. On the left, there's a "Main Menu" button. In the center, the word "SEARCH!" is displayed above a large input field labeled "Your Image" which contains a photo of a multi-story building. To the right of the input field are three buttons: "Change Image", "Change Dataset", and a toggle switch between "COLOR" and "TEXTURE". Below these is a large blue "SEARCH!" button. Underneath the search area, the word "RESULT" is shown in yellow, followed by "8 result(s) in 0.516 second(s)". A grid of eight images is displayed, each with its similarity percentage below it. The images include various landscapes and buildings. At the bottom, a page number "1 / 1" is shown.</p>	<h3>COLOR</h3> <p>Pencarian foto ITB pada 15 gambar dari dataset 2 dengan pencarian color tidak menghasilkan gambar yang cukup dekat.</p>
14	 <p>The screenshot shows a search interface with a blue header. On the left, there's a "Main Menu" button. In the center, the word "SEARCH!" is displayed above a large input field labeled "Your Image" which contains a photo of a multi-story building. To the right of the input field are three buttons: "Change Image", "Change Dataset", and a toggle switch between "COLOR" and "TEXTURE", which is currently set to "TEXTURE". Below these is a large blue "SEARCH!" button. Underneath the search area, the word "RESULT" is shown in yellow, followed by "15 result(s) in 4.211 second(s)". A grid of eight images is displayed, each with its similarity percentage below it. The images are mostly abstract textures or landscapes. At the bottom, a page number "1 / 2" is shown with a right arrow.</p>	<h3>TEXTURE</h3> <p>Pencarian foto ITB pada 15 gambar dari dataset 2 dengan pencarian tekstur tidak menghasilkan gambar yang cukup dekat.</p>

15	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image</p> <p>Change Dataset</p> <p>COLOR <input checked="" type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT 1872 result(s) in 22.766 second(s)</p> <table border="1"> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 100.00000000 %</td> <td>Similarity: 95.99999930 %</td> <td>Similarity: 86.03330366 %</td> <td>Similarity: 85.02619988 %</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 70.00000000 %</td> <td>Similarity: 75.49116441 %</td> <td>Similarity: 85.34910222 %</td> <td>Similarity: 75.07041670 %</td> </tr> </tbody> </table> <p>1 / 234 ></p>					Similarity: 100.00000000 %	Similarity: 95.99999930 %	Similarity: 86.03330366 %	Similarity: 85.02619988 %					Similarity: 70.00000000 %	Similarity: 75.49116441 %	Similarity: 85.34910222 %	Similarity: 75.07041670 %	<h3>COLOR</h3> <p>Pencarian gambar orang yang di-<i>invert</i> warnanya pada pencarian color tidak berhasil menemukan gambar aslinya.</p>
Similarity: 100.00000000 %	Similarity: 95.99999930 %	Similarity: 86.03330366 %	Similarity: 85.02619988 %															
Similarity: 70.00000000 %	Similarity: 75.49116441 %	Similarity: 85.34910222 %	Similarity: 75.07041670 %															
16	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image</p> <p>Change Dataset</p> <p>COLOR <input type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT 2192 result(s) in 43.243 second(s)</p> <table border="1"> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 100.00000000 %</td> <td>Similarity: 98.00000000 %</td> <td>Similarity: 98.00000000 %</td> <td>Similarity: 98.00000000 %</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 97.00000000 %</td> <td>Similarity: 97.00000000 %</td> <td>Similarity: 97.00000000 %</td> <td>Similarity: 97.00000000 %</td> </tr> </tbody> </table> <p>1 / 274 ></p>					Similarity: 100.00000000 %	Similarity: 98.00000000 %	Similarity: 98.00000000 %	Similarity: 98.00000000 %					Similarity: 97.00000000 %	Similarity: 97.00000000 %	Similarity: 97.00000000 %	Similarity: 97.00000000 %	<h3>TEXTURE</h3> <p>Pencarian gambar orang yang di-<i>invert</i> warnanya pada pencarian tekstur dapat ditemukan di posisi setelah gambar aslinya.</p>
Similarity: 100.00000000 %	Similarity: 98.00000000 %	Similarity: 98.00000000 %	Similarity: 98.00000000 %															
Similarity: 97.00000000 %	Similarity: 97.00000000 %	Similarity: 97.00000000 %	Similarity: 97.00000000 %															

17	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image Change Dataset</p> <p>COLOR <input checked="" type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT 913 result(s) in 21.776 second(s)</p> <table border="1"> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 25.038877802 %</td> <td>Similarity: 21.140567737 %</td> <td>Similarity: 20.038877034 %</td> <td>Similarity: 20.038877035 %</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 20.038877034 %</td> <td>Similarity: 20.038877035 %</td> <td>Similarity: 19.111104045 %</td> <td>Similarity: 17.000041277 %</td> </tr> </tbody> </table> <p>1 / 115 ></p>					Similarity: 25.038877802 %	Similarity: 21.140567737 %	Similarity: 20.038877034 %	Similarity: 20.038877035 %					Similarity: 20.038877034 %	Similarity: 20.038877035 %	Similarity: 19.111104045 %	Similarity: 17.000041277 %	<p>COLOR</p> <p>Pencarian gambar sepatu yang di-<i>invert</i> warnanya pada pencarian color tidak dapat menemukan gambar aslinya.</p>
Similarity: 25.038877802 %	Similarity: 21.140567737 %	Similarity: 20.038877034 %	Similarity: 20.038877035 %															
Similarity: 20.038877034 %	Similarity: 20.038877035 %	Similarity: 19.111104045 %	Similarity: 17.000041277 %															
18	 <p>SEARCH!</p> <p>Your Image</p> <p>Change Image Change Dataset</p> <p>COLOR <input type="checkbox"/> TEXTURE</p> <p>SEARCH!</p> <p>RESULT 2192 result(s) in 42.016 second(s)</p> <table border="1"> <tbody> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 96.000000001 %</td> <td>Similarity: 99.000000722 %</td> <td>Similarity: 99.000000214 %</td> <td>Similarity: 99.000000106 %</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>Similarity: 96.000000001 %</td> <td>Similarity: 99.000000887 %</td> <td>Similarity: 99.000000701 %</td> <td>Similarity: 99.000000870 %</td> </tr> </tbody> </table> <p>1 / 274 ></p>					Similarity: 96.000000001 %	Similarity: 99.000000722 %	Similarity: 99.000000214 %	Similarity: 99.000000106 %					Similarity: 96.000000001 %	Similarity: 99.000000887 %	Similarity: 99.000000701 %	Similarity: 99.000000870 %	<p>TEXTURE</p> <p>Pencarian gambar sepatu yang di-<i>invert</i> warnanya pada pencarian tekstur dapat ditemukan di posisi pertama.</p>
Similarity: 96.000000001 %	Similarity: 99.000000722 %	Similarity: 99.000000214 %	Similarity: 99.000000106 %															
Similarity: 96.000000001 %	Similarity: 99.000000887 %	Similarity: 99.000000701 %	Similarity: 99.000000870 %															

19	 <p>The screenshot shows the search interface with the following elements:</p> <ul style="list-style-type: none"> Main Menu button at the top left. SEARCH! button at the top center. Your Image placeholder box containing a white rectangle. Change Image and Change Dataset buttons on the right. COLOR toggle switch (blue) and TEXTURE toggle switch (orange). SEARCH! button at the bottom center. RESULT section below the search button, displaying a small thumbnail of a black and white image with the text "Similarity: 96.00001000 %". 1 / 1 indicator at the bottom. 	<p>COLOR</p> <p>Pencarian gambar png transparent pada dataset 4 hanya menghasilkan gambar berwarna hitam dengan tingkat kemiripan 96.6%.</p>
20	 <p>The screenshot shows the search interface with the following elements:</p> <ul style="list-style-type: none"> Main Menu button at the top left. SEARCH! button at the top center. Your Image placeholder box containing a white rectangle. Change Image and Change Dataset buttons on the right. COLOR toggle switch (blue) and TEXTURE toggle switch (brown). SEARCH! button at the bottom center. RESULT section below the search button, displaying a large blue placeholder box with the text "0 result(s) in 2.427 second(s)". 1 / 1 indicator at the bottom. 	<p>TEXTURE</p> <p>Pencarian gambar png transparent pada dataset 4 tidak menghasilkan foto apa-apa.</p>

4.5 Analisis Solusi

4.5.1. Kasus Pencarian Gambar dengan Warna yang Mirip

Pada kasus-kasus pencarian gambar dengan warna yang mirip, CBIR dengan parameter warna menghasilkan hasil yang jauh lebih baik daripada CBIR dengan parameter tekstur. Contohnya, pada hasil pengujian 1 ketika query berupa gambar warna biru, dihasilkan barang-barang berwarna biru pada dataset 3. Sama halnya pada hasil pengujian 1, pada hasil pengujian 3 ketika query berupa gambar warna merah solid, dihasilkan gambar-gambar berwarna merah juga pada dataset 3.

Pada pencarian CBIR dengan parameter tekstur, mencari gambar dengan warna yang sama tidak akurat. Contohnya pada hasil pengujian 2 dan 4, hasil yang ditampilkan tidak memiliki warna yang sama dengan *query*-nya. Hal ini disebabkan karena pencarian CBIR dengan parameter tekstur menerima *feature* tekstur dari gambar dan menghitung tingkat kemiripannya kemudian menampilkan gambar-gambar dengan tekstur yang mirip dengan *query*-nya. Karena gambar dengan *feature* tekstur yang mirip belum tentu memiliki warna yang sama, maka gambar yang ditampilkan juga belum tentu memiliki warna yang sama.

Pada pencarian CBIR dengan parameter warna, mencari gambar dengan warna yang sama jauh lebih akurat dan bekerja dengan baik jika dibandingkan dengan pencarian CBIR dengan parameter tekstur. Hal ini disebabkan karena CBIR dengan parameter warna menerima warna-warna yang ada pada gambar dan membuat hasil distribusi warna dan dihitung tingkat kemiripannya. Oleh karena itu, gambar dengan distribusi warna yang paling mirip yang akan ditampilkan.

4.5.2. Kasus Pencarian Gambar yang di-*invert* dan diberi *Filter*

Pada kasus-kasus pencarian gambar yang di-*invert* warnanya, CBIR dengan parameter tekstur menghasilkan hasil yang jauh lebih baik daripada CBIR dengan parameter warna. Buktinya, pada hasil pengujian 10, 16, dan 18, CBIR dengan parameter tekstur menghasilkan hasil pencarian yang lebih baik daripada CBIR dengan parameter warna (hasil pengujian 9, 15, 17).

Pada pengujian 9 & 10, dilakukan query foto kucing pada dataset 1 yang ditambah gambar kucing query yang telah diberi *filter black & white* dan *filter contrast, saturation, vibrance*, dan *warmth*. Pada pengujian 9 (CBIR dengan parameter warna), foto kucing *black & white* ditemukan di halaman terakhir (kemiripan 61%) sedangkan foto kucing yang diberi *filter contrast, saturation, vibrance*, dan *warmth* tidak ditampilkan (kemiripan < 60%). Pada pengujian 10 (CBIR dengan parameter tekstur), foto kucing *black & white* didapatkan di posisi pertama, sedangkan foto kucing yang diberi *filter* berada di posisi 8 dari 100 gambar.

Pada pengujian 15 & 16 serta 17 & 18 juga terlihat bahwa CBIR dengan parameter tekstur berhasil menemukan gambar yang di-*invert* warnanya, sedangkan CBIR dengan parameter color tidak dapat menemukan gambarnya.

Hal ini disebabkan ketika dilakukan pemberian *filter* pada gambar atau gambar di-*invert*, warna gambar akan berubah sedangkan tekstur gambar tidak akan berubah banyak. Oleh sebab itu, CBIR dengan parameter warna akan menghasilkan tingkat kemiripan yang jauh berbeda dengan gambar aslinya. Berkebalikan dengan itu, CBIR dengan parameter tekstur akan menghasilkan tingkat kemiripan yang tinggi pada gambar yang di-*filter* dan di-*invert* karena tekstur gambar tidak berubah jauh.

4.5.3. Kasus Pencarian Gambar dengan Sudut Pandang yang Berbeda

Pada pencarian gambar atau foto dengan sudut pandang yang berbeda, kedua metode CBIR menghasilkan hasil pencarian yang cukup baik. Namun, CBIR dengan parameter warna menghasilkan hasil yang dapat dibilang jauh lebih baik daripada CBIR dengan parameter tekstur.

Pada hasil pengujian 5 & 6, dilakukan query foto kucing pada dataset 1 yang ditambah foto kucing yang sama dengan sudut pandang yang berbeda. Pada hasil pengujian 5 (CBIR dengan parameter warna), foto kucing yang sama dengan sudut pandang berbeda berhasil ditampilkan sebagai hasil pertama. Pada hasil pengujian 6 (CBIR dengan parameter tekstur), foto kucing dengan sudut pandang berbeda hanya dapat ditampilkan pada posisi 62 dari 100 gambar.

Pada hasil pengujian 7 & 8, dilakukan query foto tempat pensil pada dataset 2 yang ditambah foto tempat pensil yang sama dari sudut pandang yang berbeda. Hasil pengujian CBIR dengan parameter warna menghasilkan foto tempat pensil dari sudut pandang berbeda di posisi pertama, sedangkan CBIR dengan parameter tekstur menghasilkan foto tersebut di posisi keempat dari 100 foto.

Hal ini dapat terjadi karena ketika kita mengambil gambar dari suatu objek pada 2 sudut pandang yang berbeda (hasil pengujian 5 & 6 dan 7 & 8), komposisi warna dari objek-objek pada foto tidak jauh berbeda sehingga CBIR dengan parameter warna dapat menghasilkan hasil yang baik. Di sisi lain, suatu objek yang difoto dari sudut pandang yang berbeda akan menghasilkan tekstur gambar yang cukup berbeda. Oleh sebab itu, CBIR dengan parameter tekstur tidak dapat melakukan pencarian/temu balik sebaik CBIR dengan parameter warna.

4.5.4. Kasus Pencarian Gambar PNG Transparan

Pada pencarian gambar PNG transparan, metode CBIR dengan parameter warna menghasilkan 1 gambar, sedangkan CBIR dengan parameter tekstur tidak memberikan hasil.

Pada pengujian 19, dilakukan query dengan gambar PNG transparan pada dataset 4 yang berisi macam-macam warna. Hasil pengujian CBIR dengan parameter warna hanya memberikan 1 output, yaitu warna hitam. Hal ini disebabkan karena gambar PNG transparan mempunyai nilai HSV 0, sama seperti warna hitam.

Pada pengujian 20, dilakukan query dengan gambar PNG transparan pada dataset 4 yang berisi macam-macam warna. Hasil pengujian CBIR dengan parameter tekstur tidak memberikan output apa pun karena tidak ada gambar pada data set yang tekturnya mirip dengan query. Hal ini disebabkan karena gambar PNG transparan memiliki *contrast* 0, *homogeneity* 1, dan *entropy* 0 sedangkan gambar lain pada dataset bertekstur.

Bab V

Kesimpulan

5.1 Kesimpulan

CBIR dengan parameter warna dapat dilakukan dengan mengekstrak nilai-nilai RGB dari gambar dan mengubahnya menjadi HSV, kemudian gambar akan dibagi menjadi 4×4 blok agar informasi mengenai lokasi distribusi warna tidak hilang. Masing-masing blok juga akan dicari frekuensi warnanya yang kemudian akan digunakan untuk mencari *cosine similarity* dengan blok-blok pada gambar lain. Hasil dari *cosine similarity* tersebut menunjukkan tingkat kemiripan kedua gambar berdasarkan distribusi warnanya.

CBIR dengan parameter tekstur menerima input gambar yang akan diubah menjadi Matrix RGB sehingga dapat diproses. Nilai-nilai pada tiap elemen matrix RGB diubah menjadi matrix grayscale yang dibulatkan menjadi *integer* untuk kemudian diproses menjadi matrix co-occurrence. Matrix co-occurrence diubah menjadi matrix symmetric dengan menjumlahkan tiap elemennya dengan elemen di matrix *transpose*-nya. Kemudian, matrix symmetric diubah menjadi matrix GLCM dengan dinormalisasi dan dari matrix GLCM dapat dihitung nilai *contrast*, *homogeneity*, dan *entropy* dari gambar tersebut yang akhirnya digunakan sebagai parameter *cosine similarity*.

Berdasarkan hasil pengujian, CBIR dengan parameter warna maupun tekstur masing-masing memiliki kasus yang menonjolkan kelebihan dan kekurangannya. CBIR dengan parameter warna lebih baik dalam melakukan pencarian gambar yang memiliki warna mirip dengannya. CBIR dengan parameter tekstur lebih baik dalam mencari gambar dengan objek-objek/tekstur yang sama, walaupun berbeda warnanya. Pada kasus pencarian foto objek dari sudut pandang yang berbeda, CBIR dengan parameter warna dapat melakukan temu balik yang lebih baik dari CBIR dengan parameter tekstur. Jadi, penggunaan masing-masing metode dapat disesuaikan dengan kasus yang akan dicari.

5.2 Saran

Dalam program yang dibuat dalam tugas besar ini, hanya dapat dilakukan pencarian atau temu balik menggunakan CBIR dengan parameter warna dan CBIR dengan parameter tekstur, tidak bisa keduanya. Karena kedua metode CBIR tersebut memiliki kelebihan dan kekurangannya masing-masing, maka akan lebih baik jika dapat dilakukan pencarian yang menggabungkan hasil kedua parameter tersebut. Hasil *cosine similarity* dapat dirata-ratakan atau diberi weight pada hasil kedua metode CBIR. Dengan demikian, temu balik untuk kasus-kasus umum dapat menghasilkan hasil yang lebih baik.

5.3 Komentar

Tugas Besar 2 IF2123 Aljabar Linier dan Geometri sudah sangat menarik karena mengeksplorasi banyak bidang keinformatikaan dan disiplin ilmu dalam mata kuliah Algeo sendiri. Pada bidang keinformatikaan, dibutuhkan skill Web Development hingga mempelajari library image processing dan array processing seperti PIL dan Numpy. Dalam bidang Algeo, kita belajar untuk mengaplikasikan ilmu aljabar vektor dan matrix dalam mengimplementasikan CBIR (*Content-Based Image Retrieval*) dengan parameter *Color* (warna) dan *Texture* (tekstur).

5.4 Refleksi

Refleksi yang kami dapatkan setelah selesai mengerjakan Tugas Besar 2 IF2123 Aljabar Linier dan Geometri adalah membagi beban tugas pembuatan website ke beberapa orang sehingga untuk menambahkan fitur-fitur yang berhubungan dengan website dapat dilakukan oleh lebih dari satu orang. Selain itu, mungkin pengerajan Tugas Besar 2 IF2123 Aljabar Linier dan Geometri ini bisa dikerjakan lebih awal lagi sehingga bisa mengerjakan lebih bonus dan pengerajaannya tidak terlalu dekat dengan deadline.

5.5 Ruang Perbaikan

Setelah selesai mengerjakan Tugas Besar 2 IF2123 Aljabar Linier dan Geometri, kami merasa masih banyak hal yang dapat diperbaiki dan dikembangkan lebih jauh lagi. Misalnya, menambahkan loading screen atau loading icon saat mengupload image atau dataset, maupun saat searching dengan metode color atau tekstur sehingga dapat diketahui dengan lebih jelas apakah proses tersebut sudah selesai atau belum.

Selain itu, pada CBIR dengan parameter tekstur, dapat dilakukan *compression* pada matrix grayscale sehingga program dapat dijalankan dengan lebih cepat karena untuk saat ini processing CBIR dengan parameter tekstur masih tidak secepat color.

Daftar Pustaka

[1]

“5.159.200+ Colored Paper Foto Stok, Potret, & Gambar Bebas Royalti - iStock,” www.istockphoto.com. <https://www.istockphoto.com/id/foto-foto/colored-paper> (accessed Nov. 19, 2023).

[2]

A. Fauzan, “Ruang Warna Hue Saturation Value (HSV) serta Proses Konversinya,” www.kitainformatika.com, Jan. 03, 2015. <http://www.kitainformatika.com/2015/01/ruang-warna-hue-saturation-value-hsv.html> (accessed Nov. 18, 2023).

[3]

adminlp2m, “Front End dan Back End, Mengenal Definisi dan Perbedaannya,” *Lembaga Penelitian dan Pengabdian Masyarakat*, Jun. 29, 2022. <https://lp2m.uma.ac.id/2022/06/29/front-end-dan-back-end-mengenal-definisi-dan-perbedaan-nya/> (accessed Nov. 18, 2023).

[4]

“Apa Bedanya RGB dengan CMYK? - INSTIKI,” *Instiki*, Jun. 12, 2022. <https://instiki.ac.id/2022/06/12/apa-bedanya-rgb-dengan-cmyk/> (accessed Nov. 18, 2023).

[5]

BugBytes, “Django Media Files - Handling User Uploads in Django Forms & Models,” www.youtube.com, Mar. 10, 2023. https://www.youtube.com/watch?v=lKyH_ZGtvwM&t=1002s (accessed Nov. 06, 2023).

[6]

Django, “Django,” *Django Project*. <https://docs.djangoproject.com/en/4.2/> (accessed Nov. 05, 2023).

[7]

“Django Tutorial,” www.w3schools.com. <https://www.w3schools.com/django/index.php> (accessed Nov. 05, 2023).

[8]

J. Bhathena, “Weather Image Recognition,” www.kaggle.com, 2021. <https://www.kaggle.com/datasets/jehanbhathena/weather-dataset?rvi=1> (accessed Nov. 10, 2023).

[9]

J. Yue, Z. Li, L. Liu, and Z. Fu, “Content-based image retrieval using color and texture fused features,” *Mathematical and Computer Modelling*, vol. 54, no. 3, pp. 1121–1127, Aug. 2011, doi: <https://doi.org/10.1016/j.mcm.2010.11.044>.

[10]

“NumPy Documentation,” numpy.org. <https://numpy.org/doc/> (accessed Nov. 05, 2023).

[11]

P. Bansal, “Intel Image Classification,” [www.kaggle.com](http://www.kaggle.com/puneet6060/intel-image-classification), 2018. <https://www.kaggle.com/datasets/puneet6060/intel-image-classification> (accessed Nov. 10, 2023).

[12]

T. Redaksi, “7 Pengertian Website Menurut Ahli, Lengkap Jenis & Fungsinya,” *CNBC Indonesia*, Jun. 18, 2022. <https://www.cnbcindonesia.com/tech/20220618152119-37-348229/7-pengertian-website-menurut-ahli-lengkap-jenis-fungsinya> (accessed Nov. 18, 2023).

[13]

“Texture Analysis Using the Gray-Level Co-Occurrence Matrix (GLCM) - MATLAB & Simulink,” [www.mathworks.com](https://www.mathworks.com/help/images/texture-analysis-using-the-gray-level-co-occurrence-matrix-glcm.html). <https://www.mathworks.com/help/images/texture-analysis-using-the-gray-level-co-occurrence-matrix-glcm.html> (accessed Nov. 17, 2023).

[14]

“Texture Feature.” Accessed: Nov. 18, 2023. [Online]. Available: https://www.davuniversity.org/images/files/study-material/Texture%20feature_2-end.pdf

[15]

Very Academy, “Django | Upload Multiple Images,” [www.youtube.com](https://www.youtube.com/watch?v=_JfJ0QmqZL8&t=2116s), Apr. 12, 2022. https://www.youtube.com/watch?v=_JfJ0QmqZL8&t=2116s (accessed Nov. 06, 2023).

[16]

Y. Muhammad, “Feature Extraction: Gray Level Co-occurrence Matrix (GLCM) ,” *Medium*, Jul. 16, 2020. <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1> (accessed Nov. 06, 2023).

Link Repository dan Video

Link Repository:

<https://github.com/Zechtro/Algeo02-22020>

Link Video:

https://youtu.be/tpejp_cRtLg