

# LAPORAN TUGAS BESAR 2

## IF2123 ALJABAR LINIER DAN GEOMETRI



### Kelompok Chocolens

Disusun oleh:

Debrina Veisha Rashika W. 13522025  
Nabila Shikoofa Muida 13522069  
Novelya Putri Ramadhani 13522096

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2023

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>1</b>
<b>BAB 1 DESKRIPSI MASALAH</b>	<b>2</b>
<b>BAB 2 LANDASAN TEORI</b>	<b>3</b>
2.1 Content-Based Image Retrieval (CBIR)	3
2.2 CBIR dengan Parameter Warna	4
2.3 Gray-Level Co-occurrence matrix (GLCM)	5
2.4 CBIR dengan Parameter Tekstur	6
2.5 Pengembangan Website	8
<b>BAB 3 ANALISIS PEMECAHAN MASALAH</b>	<b>9</b>
3.1. Langkah-Langkah Pemecahan Masalah	9
3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen pada Aljabar Geometri	10
3.3. Contoh ilustrasi kasus dan penyelesaiannya	12
<b>BAB 4 IMPLEMENTASI DAN UJI COBA</b>	<b>13</b>
4.1. Implementasi Program Utama	13
4.2. Struktur Back-End Program	26
a. app.py	26
b. cbir_color.py	27
c. cbir_texture.py	28
4.3. Struktur Front-End Program	29
4.4. Cara Penggunaan Program	32
4.5. Hasil Pengujian Mode Warna	34
4.6 Hasil Pengujian Mode Tekstur	39
<b>BAB 5 PENUTUP</b>	<b>43</b>
5.1 Kesimpulan	43
5.2 Saran	43
5.3 Komentar atau Tanggapan	43
5.4 Refleksi terhadap tugas besar	43
5.5 Ruang Perbaikan atau Pengembangan	44
<b>DAFTAR PUSTAKA</b>	<b>45</b>
<b>LAMPIRAN</b>	<b>46</b>
● Link Repository	46
● Link Youtube	46

## **BAB 1**

### **DESKRIPSI MASALAH**

Di era digital yang semakin berkembang pesat, jumlah gambar yang tersebar dan tersimpan di dunia maya juga meningkat. Ledakan jumlah gambar yang dihasilkan oleh pengguna secara pribadi maupun dalam konteks profesional menciptakan kebutuhan mendesak untuk sistem yang dapat mengelola dan mengambil gambar secara efisien. Fenomena ini terjadi pada berbagai jenis gambar, mulai dari foto pribadi, gambar medis, ilustrasi ilmiah, hingga gambar komersial.

Peningkatan signifikan dalam keragaman sumber dan jenis gambar menimbulkan tantangan baru dalam pencarian, akses, dan manajemen koleksi gambar. Dalam menghadapi kompleksitas ini, sistem temu balik gambar menjadi sarana yang sangat relevan dan penting. Sistem ini memungkinkan pengguna untuk tidak hanya mencari dan mengakses gambar dengan lebih efisien, tetapi juga untuk mencari gambar yang memiliki kesamaan dan kemiripan visual yang tersebar di berbagai platform.

Salah satu hal yang dapat dilakukan untuk mengatasi tantangan ini adalah penerapan aljabar vektor dalam konteks algoritma Content-Based Image Retrieval (CBIR). Dengan pendekatan ini, gambar tidak hanya diidentifikasi berdasarkan meta datanya, tetapi juga oleh konten visualnya, seperti warna, tekstur, dan pola. Hal ini memudahkan pengguna untuk melakukan pencarian gambar dengan cara yang lebih efektif, sehingga memungkinkan pencarian gambar pribadi yang lebih akurat, analisis medis yang lebih mendalam, dan penemuan ilustrasi ilmiah yang lebih cepat.

Sebagai contoh nyata, Google Lens telah menjadi salah satu aplikasi yang memanfaatkan sistem temu balik gambar ini secara efektif. Melalui kemampuan analisis visualnya, Google Lens memungkinkan pengguna untuk mengeksplorasi dan memahami dunia di sekitar mereka dengan cara yang belum pernah terjadi sebelumnya. Di dalam Tugas Besar 2 ini, kami mengimplementasikan sistem temu balik gambar dengan memanfaatkan Aljabar Vektor dalam bentuk sebuah *website*. Dalam konteks ini, aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (*Content-Based Image Retrieval* atau CBIR), di mana sistem temu balik gambar bekerja dengan mengidentifikasi gambar berdasarkan konten visualnya, seperti warna dan tekstur.

## **BAB 2**

### **LANDASAN TEORI**

#### **2.1 *Content-Based Image Retrieval (CBIR)***

Content-Based Image Retrieval (CBIR) merupakan suatu pendekatan pencarian gambar yang didasarkan pada analisis fitur atau ciri-ciri dari gambar tersebut. Konsep dasar CBIR adalah membandingkan karakteristik visual dari gambar yang sedang dicari dengan gambar-gambar yang tersimpan dalam basis data.

Dalam membangun sistem CBIR, fitur-fitur dalam gambar akan digunakan untuk membedakan gambar satu dengan yang lain. Ramadjanti (2006) menunjukkan bahwa ciri-ciri ini mencakup aspek-aspek seperti warna, tekstur, dan bentuk. Warna digunakan untuk mengidentifikasi distribusi piksel dan kombinasi warna dalam gambar, sementara tekstur mencakup pola dan detail permukaan gambar, dan bentuk berfokus pada struktur geometris objek dalam gambar.

Proses CBIR dimulai dengan formulasi query berupa gambar atau representasi vektor fitur dari gambar yang dicari. Sistem kemudian membandingkan fitur-fitur ini dengan fitur-fitur gambar dalam basis data untuk menemukan kemiripan visual. Hidayat et al. (2017) menekankan bahwa keberhasilan CBIR sangat tergantung pada pemilihan fitur-fitur yang tepat dan metode pencocokan yang efisien. Penggunaan kombinasi warna, tekstur, dan bentuk dapat meningkatkan akurasi dan relevansi dalam mengambil gambar dari basis data yang besar dan beragam.

CBIR tidak hanya menyederhanakan proses pencarian gambar, tetapi juga meningkatkan pengalaman pengguna dengan memberikan solusi yang lebih visual dan intuitif. Kemampuan sistem ini untuk memahami dan membandingkan fitur-fitur kompleks dari gambar menciptakan solusi yang relevan dalam mengatasi kompleksitas koleksi gambar digital yang terus berkembang pesat. Seiring dengan perkembangan teknologi, CBIR terus menjadi alat yang digunakan untuk mencari dan mengelola kekayaan visual yang semakin melimpah.

## 2.2 CBIR dengan Parameter Warna

Pada CBIR 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  $\Delta$

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

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

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

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

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

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

$$V = C_{max}$$

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

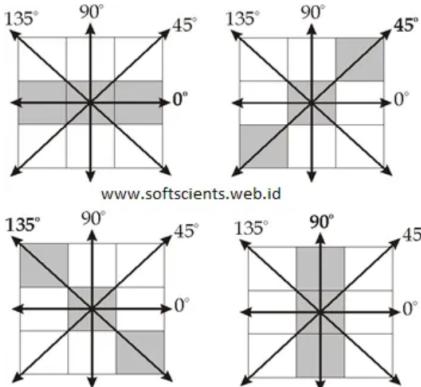
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

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.

### 2.3 Gray-Level Co-occurrence matrix (GLCM)

Secara visual tekstur gambar dapat diamati dari perulangan pola, distribusi spasial, dan susunan warna dan intensitas. Untuk melakukan analisis tekstur dapat digunakan Gray-Level Co-occurrence matrix (GLCM). GLCM merepresentasikan hubungan antara 2 pixel yang bertetanggaan (neighboring pixels) yang memiliki intensitas keabuan (grayscale intensity), jarak dan sudut. Terdapat 8 sudut yang dapat digunakan pada GLCM, diantaranya sudut  $0^\circ, 45^\circ, 90^\circ, 135^\circ, 180^\circ, 225^\circ, 270^\circ$ , atau  $315^\circ$ .



**Gambar 1.** Sudut pada GLCM

Matriks GLCM dapat dibuat dengan beberapa langkah yaitu, pembuatan framework matrix, pembuatan co-occurrence matrix (mengisi framework matrix), pembuatan matriks simetris (penjumlahan co-occurrence matriks dengan transpose matriks), dan membuat matriks *normalization* yang akan menghasilkan nilai matriks antara 0–1. Nilai matriks *normalization* didapat dari formula berikut:

$$glcmNorm = \frac{glcmValue}{\sum_i^N glcmValue}$$

**Gambar 2.** Rumus Matriks *Normalization*

## 2.4 CBIR dengan Parameter Tekstur

Untuk melakukan CBIR dengan perbandingan tekstur dapat dilakukan dengan bantuan *co-occurrence matrix*. Matriks ini didapat dari mencari *Gray-Level Co-occurrence matrix* dari sebuah gambar. Matriks ini digunakan karena dapat melakukan pemrosesan yang lebih mudah dan cepat. Vektor yang dihasilkan juga mempunyai ukuran yang lebih kecil.

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

**Gambar 3.** Matriks Co-occurrence

Langkah pertama yang dilakukan pada CBIR ini adalah mengkonversi gambar menjadi *grayscale* dengan rumus  $Y = 0.29 \times R + 0.587 \times G + 0.114 \times B$ . Setelah mengubah warna gambar, lakukan kuantifikasi dari nilai *grayscale* untuk membuat *co-occurrence matrix* yang memiliki ukuran  $256 \times 256$ .

Langkah kedua setelah mendapat *co-occurrence matrix*, dapat dicari 3 komponen ekstraksi tekstur, berupa *contrast*, *entropy*, dan, *homogeneity*. Ketiga komponen tersebut akan dibuat vektor untuk mengukur tingkat kemiripan suatu gambar. Ketiga komponen tersebut bisa dicari dengan rumus sebagai berikut.

*Contrast:*

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

*Homogeneity :*

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

*Entropy :*

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

Keterangan :  $P$  merupakan matriks *co-occurrence*

Setelah mendapatkan vektor yang berisi komponen ekstraksi tekstur. Kemiripan dua gambar dapat dicari dengan Teorema *Cosine Similarity*. Semakin besar hasil dari *Cosine Similarity* maka tingkat kemiripan kedua gambar yang dibandingkan akan semakin tinggi. Berikut rumus dari Teorema *Cosine Similarity*.

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

**Gambar 4.** Teorema Cosine Similarity.

## 2.5 Pengembangan Website

Pengembangan website adalah suatu proses kompleks yang melibatkan perancangan, pembangunan, dan pemeliharaan suatu situs web. Proses ini terbagi menjadi dua aspek utama: frontend dan backend. Frontend berkaitan dengan tampilan yang dapat dilihat pengguna, termasuk antarmuka pengguna, tata letak, dan elemen desain lainnya. Di sisi lain, backend merupakan bagian yang tidak terlihat oleh pengguna, tetapi mendukung keberlangsungan operasional website, termasuk pengelolaan database, logika aplikasi, dan server.

Pentingnya kerjasama antara frontend dan backend tidak dapat diabaikan. Frontend harus dirancang sedemikian rupa untuk menciptakan pengalaman pengguna yang menarik dan responsif, sementara backend harus dirancang untuk memastikan bahwa data diproses dengan efisien, keamanan terjaga, dan integrasi antara frontend dan backend berjalan lancar.

Langkah pertama dalam pengembangan website adalah perencanaan yang cermat, termasuk identifikasi tujuan, audiens target, dan kebutuhan fungsional. Setelah itu, tim pengembang merancang tata letak (layout) dan struktur informasi untuk memastikan *user experience* yang optimal. Pengembangan website melibatkan penerjemahan desain menjadi kode-kode yang dapat dijalankan oleh browser, menggunakan bahasa pemrograman seperti HTML, CSS, dan JavaScript untuk menciptakan antarmuka responsif dan dinamis.

Pemeliharaan website mencakup pembaruan rutin, perbaikan bug, dan peningkatan fitur untuk memastikan keberlanjutan dan keamanan. Pengoptimalan kinerja juga menjadi aspek penting untuk memastikan bahwa website dapat diakses dengan cepat dan efisien.

## **BAB 3**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1. Langkah-Langkah Pemecahan Masalah**

Langkah-langkah pemecahan masalah dalam mencari gambar yang memiliki kemiripan dari segi warna atau tekstur dengan CBIR sebagai berikut:

##### **1. Analisis Gambar dan Pemilihan Mode**

Melakukan analisis gambar yang akan dicari dan menentukan mode yang akan digunakan dalam metode Content-Based Image Retrieval (CBIR). Misalnya, pemilihan mode warna atau mode tekstur.

##### **2. Mode Warna**

Jika memilih mode warna, dari gambar akan diambil matrix R, G, dan B yang kemudian dikonversi menjadi matrix H, S, dan V. Selanjutnya, masing-masing matrix dibagi menjadi blok  $4 \times 4$  yang lalu dihitung nilai kemiripannya dengan *Cosine Similarity*.

##### **3. Mode Tekstur**

Jika memilih mode tekstur, gambar diubah menjadi *grayscale*. Selanjutnya, ekstraksi fitur tekstur dilakukan dengan mencari vektor yang mencakup komponen *contrast, entropy, dan homogeneity* dari gambar tersebut.

##### **4. Formulasi Query Pencarian**

Formulasikan query pencarian berdasarkan vektor fitur gambar yang ingin dicari. Query ini mencerminkan karakteristik visual yang diinginkan, baik berdasarkan warna atau tekstur.

##### **5. Pencocokan dengan Basis Data**

Lakukan pencocokan vektor fitur gambar referensi dengan vektor fitur gambar dalam basis data. Metode yang digunakan, dalam hal ini Teorema *Cosine Similarity*, membantu mengukur tingkat kesamaan antara vektor-vektor fitur.

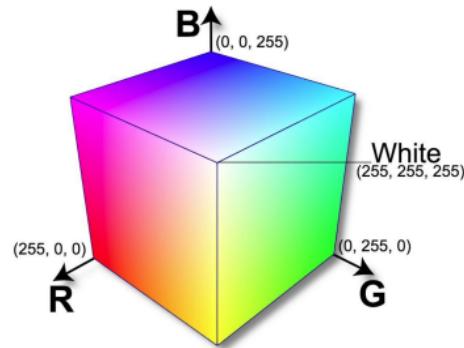
##### **6. Ranking Hasil Pencarian:**

Ranking hasil pencarian berdasarkan tingkat kesamaan. Gambar yang memiliki tingkat kesamaan di atas ambang batas tertentu, misalnya 60%, dapat dianggap sebagai hasil pencarian yang relevan. Tampilkan Hasil Pencarian

7. Tampilkan hasil pencarian yang telah dirangking kepada pengguna. Hasil disajikan dalam bentuk daftar gambar atau galeri yang memudahkan pengguna untuk melihat dan memilih gambar yang paling relevan.

### 3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen pada Aljabar Geometri

Dalam konteks Content-Based Image Retrieval (CBIR), representasi warna dalam sebuah gambar diwujudkan sebagai vektor dalam ruang warna RGB. Vektor ini menggambarkan komposisi piksel-piksel gambar dalam tiga saluran warna utama: merah (R), hijau (G), dan biru (B). Salah satunya mengkonversi warna dari RGB ke HSV yang digunakan pada fitur warna dan konversi dari RGB ke *grayscale* pada fitur tekstur.



**Gambar 5.** Vektor RGB

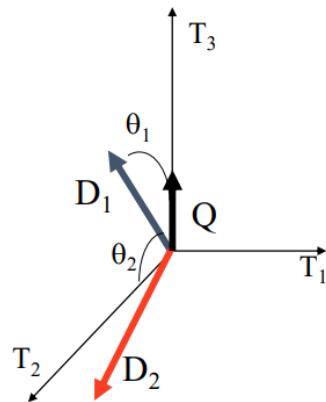
Setiap gambar pada *database* maupun query dinyatakan sebagai vektor  $w = (w_1, w_2, \dots, w_n)$  di dalam  $R^n$  dan  $w$  menyatakan komponen yang terdapat pada gambar misalnya untuk mencari CBIR tekstur diperlukan komponen *contrast*, *entropy*, dan *homogeneity*.

Penentuan gambar mana yang relevan dengan query dipandang sebagai pengukuran kesamaan (*similarity measure*) antara query dengan gambar pada *database*. Semakin tinggi tingkat kemiripan suatu vektor gambar dengan vektor query, semakin besar kemiripan gambar tersebut dengan query.

Kesamaan (sim) antara dua vektor  $Q = (q_1, q_2, \dots, q_n)$  dan  $D = (d_1, d_2, \dots, d_n)$  diukur dengan rumus *cosine similarity* yang merupakan bagian dari rumus perkalian titik (dot product) dua buah vektor:

$$sim(Q, D) = \cos \theta = \frac{Q \cdot D}{\|Q\| \|D\|}$$

Jika  $\cos$  bernilai 1, maka sudut antara kedua vektor tersebut adalah 0, yang berarti vektor  $Q$  dan  $D$  berimpit, sehingga Gambar  $D$  sesuai dengan query  $Q$ . Jadi, nilai cosinus yang besar (mendekati 1) mengindikasikan bahwa gambar cenderung sesuai dengan query.



**Gambar 6.** Vektor Q dan D

Selanjutnya hasil perhitungan di-ranking berdasarkan nilai cosinus dari besar ke kecil sebagai proses pemilihan gambar yang paling mirip dengan query. Pe-ranking-an tersebut menyatakan gambar yang paling mirip hingga yang kurang mirip dengan query. Nilai cosinus yang besar menyatakan gambar yang relevan, nilai cosinus yang kecil menyatakan gambar yang kurang relevan dengan query.

### 3.3. Contoh ilustrasi kasus dan penyelesaiannya

Misalkan kita ingin mencari kemiripan gambar pada database dengan gambar yang dimiliki, sebagai berikut:



**Gambar 7.** Gambar Query

Setelah memasukkan gambar dan folder *database* program akan mencari kemiripan berdasarkan mode yang dipilih, baik itu gambar atau tekstur. Jika memilih warna program akan fokus mencari gambar yang memiliki warna yang sama, sedangkan untuk tekstur program akan mencari gambar yang memiliki nilai *contrast, entropy, dan homogeneity* yang mirip.

Untuk mode warna, semakin banyak kemiripan warna antara dua gambar, maka nilai kemiripannya akan semakin tinggi. Contohnya, untuk gambar Query di atas, seharusnya gambar yang memiliki kemiripan tinggi adalah apel merah, tomat, dan lain sebagainya. Sedangkan, pisang, lemon, dan buah yang tidak memiliki warna merah akan memiliki nilai kemiripan yang rendah.

Untuk mode tekstur, gambar akan dilihat melalui tiga komponennya, yaitu *contrast, entropy, dan homogeneity*. Sehingga untuk gambar Query di atas akan memiliki nilai kemiripan yang tinggi dengan apel, pir, jeruk dan buah lain yang secara bentuk dan tekstur mirip. Sedangkan pisang memiliki nilai kemiripan yang lebih rendah karena secara bentuk dan tekstur berbeda dengan apel.

## BAB 4

### IMPLEMENTASI DAN UJI COBA

#### 4.1. Implementasi Program Utama

##### a. app.py

```
from flask IMPORT Flask, render_template, request,
send_from_directory, session, url_for

from werkzeug.utils IMPORT secure_filename

from reportlab.lib.pagesizes IMPORT letter

from reportlab.pdfgen IMPORT canvas

IMPORT os

from multiprocessing IMPORT Pool

from os.path IMPORT basename

IMPORT shutil

IMPORT cbir_color as col

IMPORT cbir_texture as tex

from urllib.parse IMPORT unquote

DEFINE FUNCTION create_pdf(top_images, destination_folder,time):

    os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

    pdf_path <- os.path.join(app.config['UPLOAD_FOLDER'],
'result.pdf')

    c <- canvas.Canvas(pdf_path, pagesize=letter)

    c.drawString(400, 720 - 10, f'<-tal Elapsed Time: {time:.5f} seconds')

    FOR i, (image_path, similarity_score) IN enumerate(top_images):

        c.drawImage(image_path, 20, 720 - (i%4 + 1) * 150, width=150,
height=150)

        c.drawString(200, 720 - (i%4 + 1) * 150 + 75, f'Similarity:
{similarity_score:.5f}')

        if(i%4==3):

            c.showPage()
```

```

c.save()

RETURN pdf_path


DEFINE FUNCTION file_exists(file_path):
    RETURN os.path.exists(file_path)


app <- Flask(__name__, static_url_path='/static')
app.secret_key <- b'_5#y2L"F4Q8z\n\xec]/'

app.config['UPLOAD_FOLDER'] <-
os.path.join(os.path.dirname(os.path.abspath(__file__)), 'test',
'datasave')

app.config['UPLOAD_IMAGES'] <-
os.path.join(os.path.dirname(os.path.abspath(__file__)), 'test',
'INPUT_images')

@app.route('/')

DEFINE FUNCTION home():

    RETURN render_template('home.html', file_exists=file_exists)

@app.route('/start')

DEFINE FUNCTION start():

    RETURN render_template('index.html')

@app.route('/how-to-use')

DEFINE FUNCTION how_to_use():

    RETURN render_template('home.html')

@app.route('/about-us')

DEFINE FUNCTION about_us():

    RETURN render_template('home.html')

@app.route('/uploaded-images/<path:filename>')

DEFINE FUNCTION uploaded_image(filename):

```

```

filename <- unquote(filename)

    RETURN send_from_directory(app.config['UPLOAD_IMAGES'],
secure_filename(filename))

@app.route('/images/<path:filename>')

DEFINE FUNCTION download_file(filename):

    OUTPUT("Uploaded Image Path:",
os.path.join(app.config['UPLOAD_FOLDER']))

    RETURN send_from_directory(app.config['UPLOAD_FOLDER'], filename,
as_attachment=True)

@app.route('/upload', methods=['POST'])

DEFINE FUNCTION upload():

    gambar1 <- request.files['gambar1']

    folder <- request.files.getlist('folder[]')

    destination_folder <-
os.path.join(os.path.dirname(os.path.abspath(__file__)), 'test',
'datasave')

    INPUT_images_folder <-
os.path.join(os.path.dirname(os.path.abspath(__file__)), 'test',
'INPUT_images')

    IF os.path.exists(destination_folder):

        shutil.rmtree(destination_folder)

    IF os.path.exists(INPUT_images_folder):

        shutil.rmtree(INPUT_images_folder)

    os.makedirs(destination_folder)

    os.makedirs(INPUT_images_folder)

    mode <- request.form.get('mode', 'color') # Mode default: color

    OUTPUT("Received mode:", mode)

    IF mode EQUALS 'on':

        mode <- 'texture'

```

```

IF folder:

    FOR uploaded_file IN folder:

        IF uploaded_file.filename != '':

            file_path <- os.path.join(destination_folder,
secure_filename(uploaded_file.filename))

            uploaded_file.save(file_path)

            OUTPUT(f'Saving file: {file_path}')

        ELSE:

            RETURN render_template('error.html', message='No folder
selected.')

        gambar1_path <- os.path.join(INPUT_images_folder,
secure_filename(gambar1.filename))

        gambar1.save(gambar1_path)

    IF mode EQUALS 'color':

        top_images, elapsed_time <-
col.main_process_color(gambar1_path, destination_folder)

    ELSEIF mode EQUALS 'texture':

        top_images, elapsed_time <-
tex.main_process_texture(gambar1_path, destination_folder)

    ELSE:

        RETURN render_template('error.html', message='Invalid mode
selected.')

        session['top_images'] <- top_images

        session['elapsed_time'] <- elapsed_time

        session['gambar'] <- gambar1.filename

        session['gambar_path'] <- gambar1_path

        session['page'] <- 1

        banyak_gambar <- len(top_images)

    RETURN render_template('index.html', top_images=top_images[:12],
elapsed_time=elapsed_time, banyak_gambar=banyak_gambar,
gambar=gambar1.filename,
gambar_path=gambar1_path, basename=basename, page=1,

```

```

total_pages=(banyak_gambar + 11) // 12)

@app.route('/pagination', methods=['GET', 'POST'])

DEFINE FUNCTION pagination():

    top_images <- session.get('top_images', [])

    elapsed_time <- session.get('elapsed_time', 0)

    gambar_path <- session.get('gambar_path', '')

    gambar=session.get('gambar', '')

    page <- int(request.args.get('page', 1))

    session['page'] <- page

    start_idx <- (page - 1) * 12

    end_idx <- start_idx + 12

    banyak_gambar <- len(top_images)

    total_pages <- (banyak_gambar + 11) // 12

    RETURN render_template('index.html',
    top_images=top_images[start_idx:end_idx], elapsed_time=elapsed_time,
                                         banyakgambar=banyak_gambar, gambar=gambar,
    gambar_path=gambar_path,
                                         basename=basename, page=page,
    total_pages=total_pages)

@app.route('/download-pdf')

DEFINE FUNCTION download_pdf():

    top_images <- session.get('top_images', [])

    elapsed_time <- session.get('elapsed_time', 0)

    pdf_path <- create_pdf(top_images, "test/datasave",elapsed_time)

    IF os.path.exists(pdf_path):

        RETURN send_from_directory(app.config['UPLOAD_FOLDER'],
        'result.pdf', as_attachment=True)

    ELSE:

        RETURN render_template('error.html', message='PDF file not
found.')

```

```
IF __name__ EQUALS '__main__':
    app.run(debug=True)
```

b. Cbir\_color.py

```
IMPORT numpy as np
IMPORT cv2
IMPORT os
IMPORT time
from multiprocessing IMPORT Pool
IMPORT multiprocessing
from tqdm IMPORT tqdm

DEFINE FUNCTION compress(image_matrix):
    { Mengompres matrix pixel gambar menjadi 256 x 256 }
    image <- np.array(image_matrix, dtype=np.uint8)
    resized_image <- cv2.resize(image, (256, 256))
    { Ubah kembali ke matrix pixel }
    compressed_matrix <- np.array(resized_image)
    RETURN compressed_matrix

DEFINE FUNCTION extract_rgb(image_matrix):
    { Mengambil nilai RGB dari matrix gambar }
    red_matrix <- image_matrix[:, :, 0]
    green_matrix <- image_matrix[:, :, 1]
    blue_matrix <- image_matrix[:, :, 2]
    RETURN red_matrix, green_matrix, blue_matrix
```

```

DEFINE FUNCTION rgb_<-_hsv(matrix_R, matrix_G, matrix_B):
    { Melakukan konversi warna dari RGB ke HSV }

    r, g, b <- matrix_R / 255.0, matrix_G / 255.0, matrix_B / 255.0

    Cmax <- np.maximum.reduce([r, g, b])

    Cmin <- np.minimum.reduce([r, g, b])

    delta <- Cmax - Cmin

    h <- np.where(delta != 0,
                  np.where(Cmax EQUALS r, (g - b) / (delta +
np.finfo(float).eps) % 6,
                  np.where(Cmax EQUALS g, (b - r) / (delta +
np.finfo(float).eps) + 2,
                  np.where(Cmax EQUALS b, (r - g) / (delta +
np.finfo(float).eps) + 4, 0))),
                  0)

    h <- (h * 60) % 360 { Konversi H ke range [0, 360] }

    s <- np.where(Cmax != 0, delta / Cmax, 0)

    v <- Cmax

    RETURN h, s, v

DEFINE FUNCTION cosine_similarity(vektor1, vektor2):
    { Hitung cosine similarity antara dua vektor fitur }

    dot_product <- sum(a * b FOR a, b IN zip(vektor1, vektor2))

    norm1 <- sum(a ** 2 FOR a IN vektor1) ** 0.5

    norm2 <- sum(b ** 2 FOR b IN vektor2) ** 0.5

    similarity <- dot_product / (norm1 * norm2 + 1e-10)

    RETURN similarity

DEFINE FUNCTION block_similarity(H1, S1, V1, H2, S2, V2):
    { Menghitung similarity antara 2 blok}

```

```

{ Meratakan matriks H, S, dan V menjadi vektor satu dimensi}

vector1_H <- H1.flatten().astype(float)

vector1_S <- S1.flatten().astype(float)

vector1_V <- V1.flatten().astype(float)

vector2_H <- H2.flatten().astype(float)

vector2_S <- S2.flatten().astype(float)

vector2_V <- V2.flatten().astype(float)

{ Menghitung cosine similarity untuk ap komponen (H, S, V) }

similarity_H <- cosine_similarity(vector1_H, vector2_H)

similarity_S <- cosine_similarity(vector1_S, vector2_S)

similarity_V <- cosine_similarity(vector1_V, vector2_V)

IF np.isnan(similarity_H) or np.isnan(similarity_S) or
np.isnan(similarity_V):

    RETURN 0.0

block_sim <- (similarity_H + similarity_S + similarity_V) / 3

RETURN block_sim


DEFINE FUNCTION process_image_color(image_path, gambar1_matrix,
results):

{ Membandingkan dua gambar dan mendapatkan similarity scorenya}

gambar2_matrix <- cv2.imread(image_path)

compressed_gambar1 <- compress(gambar1_matrix)

compressed_gambar2 <- compress(gambar2_matrix)

{ Mengambil matriks R, G, B dari matriks gambar1 dan gambar2 }

R1, G1, B1 <- extract_rgb(compressed_gambar1)

R2, G2, B2 <- extract_rgb(compressed_gambar2)

{ Konversi matriks R, G, B menjadi matriks H, S, V }

H1, S1, V1 <- rgb_->_hsv(R1, G1, B1)

```

```

H2, S2, V2 <- rgbs<- hsv(R2, G2, B2)

{ Inisialisasi similarity scores }

total_similarity <- 0.0

{ Iterasi untuk 16 blok 4 x 4 }

FOR y traversal [0, 256, 64]

    FOR x traversak [0, 256, 64]

        { Mendapatkan blok dari matriks H, S, V untuk gambar1 }

        block1_H <- H1[y:y + 64, x:x + 64]

        block1_S <- S1[y:y + 64, x:x + 64]

        block1_V <- V1[y:y + 64, x:x + 64]

        { Mendapatkan blok dari matriks H, S, V untuk gambar2 }

        block2_H <- H2[y:y + 64, x:x + 64]

        block2_S <- S2[y:y + 64, x:x + 64]

        block2_V <- V2[y:y + 64, x:x + 64]

        { Menghitung similarity score untuk blok saat ini }

        block_sim <- block_similarity(block1_H, block1_S,
block1_V, block2_H, block2_S, block2_V)

        { Menambahkan similarity score blok ke total similarity }

        IF not np.isnan(block_sim):

            total_similarity += block_sim

        { Menghitung rata-rata similarity untuk mendapatkan similarity
score akhir }

        similarity_score <- (total_similarity / 16) * 100

        similarity_score <- round(similarity_score, 8)

        { Menambahkan hasil ke dalam list results }

        results.append((image_path, similarity_score))

DEFINE FUNCTION main_process_color(INPUT_path, destination_folder):

    input_matrix <- cv2.imread(INPUT_path)

```

```

top_input_matrix <- compress(input_matrix)

files <- os.listdir(destination_folder)

total_files <- len(files)

with tqdm(total=total_files, desc="Processing Images") as progress:

    start_time <- time.time()

    { Jumlah proses paralel yang dijalankan }

    num_processes <- 4

    pool <- Pool(processes=num_processes)

    manager <- multiprocessing.Manager()

    { List hasil yang bisa dibagi antar proses }

    results <- manager.list()

    FOR file IN files:

        IF file != '':

            image_path <- os.path.join(destination_folder, file)

            pool.apply_async(process_image_color,
args=(image_path, com top_input_matrix, results))

    pool.close()

    pool.join()

    { Menyaring hasil dan menyimpan hanya yang memiliki
similarity_score > 60 }

    results <- [(image_path, similarity_score) FOR image_path,
similarity_score IN results IF similarity_score > 60]

    { Mengurutkan hasil berdasarkan similarity_score}

    sorted_results <- sorted(results, key=lambda x: x[1],
reverse=True)

    top_images <- sorted_results

    end_time <- time.time()

    elapsed_time <- end_time - start_time

RETURN <- top_images, elapsed_time

```

### c. cbir\_texture.py

```
IMPORT numpy as np

IMPORT cv2

IMPORT os

IMPORT time

from multiprocessing IMPORT Pool

IMPORT multiprocessing

from tqdm IMPORT tqdm

DEFINE FUNCTION compress(image_matrix):

    image <- np.array(image_matrix, dtype=np.uint8)

    resized_image <- cv2.resize(image, (256, 256))

    { Ubah kembali ke matrix pixel }

    compressed_matrix <- np.array(resized_image)

    -> compressed_matrix

DEFINE FUNCTION cosine_similarity(vektor1, vektor2):

    { Hitung cosine similarity antara dua vektor fitur}

    dot_product <- sum(a * b FOR a, b IN zip(vektor1, vektor2))

    norm1 <- sum(a ** 2 FOR a IN vektor1) ** 0.5

    norm2 <- sum(b ** 2 FOR b IN vektor2) ** 0.5

    similarity <- dot_product / (norm1 * norm2 + 1e-10)

    -> similarity

DEFINE FUNCTION ubahbw(matriksgambar):

    blue_channel <- matriksgambar[:, :, 0]

    green_channel <- matriksgambar[:, :, 1]

    red_channel <- matriksgambar[:, :, 2]
```

```

grayscale_values <- 0.114 * blue_channel + 0.587 * green_channel
+ 0.299 * red_channel

matriksgambar <- np.stack([grayscale_values, grayscale_values,
grayscale_values], axis=-1)

matriksgambar <- compress(matriksgambar)

-> matriksgambar

DEFINE PROCEDURE htgduaelmnt (mat,hasil):

FOR i traversal [0..len(mat)]:

    FOR j traversal [0..(len(mat)-1)]:

        x <- mat[i][j][0]

        y <- mat[i][j+1][0]

        hasil[x-1][y-1] += 1

DEFINE FUNCTION contrast(mat):

i, j <- np.indices(mat.shape)

-> np.sum(mat * (i - j)**2)

DEFINE FUNCTION homogeneity(mat):

i, j <- np.indices(mat.shape)

-> np.sum(mat / (1 + (i - j)**2))

DEFINE FUNCTION entropy(mat):

non_zero_indices <- np.where(mat != 0)

nonzero_elements <- mat[non_zero_indices]

nonzero_elements <- np.clip(nonzero_elements, 1e-10, None)

entropy <- np.sum(nonzero_elements * np.log(nonzero_elements))

-> entropy

DEFINE FUNCTION compare(gambar1,gambar2):

start_time <- time.time() { Waktu mulai proses }

```

```

{ occurence matrix }

occurmat <- np.zeros((256, 256), dtype=int)
occurmat2 <- np.zeros((256, 256), dtype=int)

htgduaelmnt(gambar1,occurmat)
htgduaelmnt(gambar2,occurmat2)

occurmat <- (occurmat + occurmat.T) / occurmat.sum()
occurmat2 <- (occurmat2 + occurmat2.T) / occurmat2.sum()

vek1 <-
[contrast(occurmat),homogeneity(occurmat),entropy(occurmat) ]

vek2 <-
[contrast(occurmat2),homogeneity(occurmat2),entropy(occurmat2) ]

sim <- cosine_similarity(vek1,vek2)*100

-> sim

DEFINE PROCEDURE process_image_texture(image_path, gambar1, results):
    gambar2 <- cv2.imread(image_path)
    gambar2 <- ubahbw(gambar2)

    similarity_score <- round(compare(gambar1, gambar2),8)
    results.append((image_path, similarity_score))

DEFINE FUNCTION main_process_texture(gambar1_path,
destination_folder):
    gambar1_matrix <- cv2.imread(gambar1_path)
    gambar1_matrix <- ubahbw(gambar1_matrix)
    files <- os.listdir(destination_folder)
    total_files <- len(files)

```

```

with tqdm(total=total_files, desc="Processing Images"):

    start_time <- time.time()

    num_processes <- 4

    pool <- Pool(processes=num_processes)

    manager <- multiprocessing.Manager()

    results <- manager.list()

    FOR file IN files:

        IF file != '':

            image_path <- os.path.join(destination_folder, file)

            pool.apply_async(process_image_texture,
args=(image_path, gambar1_matrix, results))

    pool.close()

    pool.join()

    results <- [(image_path, similarity_score) FOR image_path,
similarity_score IN results IF similarity_score > 60]

    sorted_results <- sorted(results, key=lambda x: x[1],
reverse=True)

    top_images <- sorted_results

    end_time <- time.time()

    elapsed_time <- end_time - start_time

-> top_images, elapsed_time

```

## 4.2. Struktur Back-End Program

### a. app.py

Fungsi / Prosedur	Deskripsi
create_pdf(top_images, destination_folder,time)	Menuliskan laporan pencarian gambar dalam bentuk pdf yang berisi top images

	dan waktu eksekusi program.
file_exists(file_path)	Memeriksa apakah file path sudah terdapat dalam program atau belum.
uploaded_image(filename)	Mengirimkan file gambar input dari direktori untuk ditampilkan.
download_file(filename)	Mengirimkan file gambar hasil dari direktori untuk ditampilkan.
upload()	Fungsi utama yang menerima gambar dan folder database input lalu mengolahnya dan mengembalikan list gambar hasil beserta nilai kemiripan, banyak gambar, dan waktu eksekusi.
download_pdf()	Fungsi untuk membuat pdf dan mengirimkan hasil pdf ke server.
pagination()	Fungsi untuk menampilkan hasil gambar dalam bentuk pagination (12 gambar per halaman).

b. cbir\_color.py

Fungsi / Prosedur	Deskripsi
compress(image_matrix)	Fungsi untuk mengkompress gambar menjadi ukuran 256x256.
cosine_similarity(vektor1, vektor2)	Fungsi untuk mencari nilai kemiripan

	dari kedua vektor dengan metode cosinus.
extract_rgb(image_matrix)	Fungsi untuk mengambil nilai RGB dari matriks gambar
rgb_to_hsv(matrix_R, matrix_G, matrix_B)	Fungsi untuk melakukan konversi warna dari RGB ke HSV
block_similarity(H1, S1, V1, H2, S2, V2)	Fungsi untuk menghitung similarity antara 2 blok.
process_image_color(image_path, gambar1_matrix, results)	Prosedur untuk memproses gambar dengan CBIR pada mode warna.
main_process_color(input_path, destination_folder)	Fungsi utama CBIR warna yang menerima masukan gambar input dan folder database dan mengembalikan list gambar hasil serta waktu eksekusi.

c. cbir\_texture.py

Fungsi / Prosedur	Deskripsi
compress(image_matrix)	Fungsi untuk mengkompress gambar menjadi ukuran 256x256.
cosine_similarity(vektor1, vektor2)	Fungsi untuk mencari nilai kemiripan dari kedua vektor dengan metode cosinus.
ubahbw(matriksgambar)	Fungsi untuk mengubah warna gambar dari RGB menjadi <i>grayscale</i> .
htgduaelmnt(mat, hasil)	Prosedur untuk membuat sebuah matriks

	co-occurrence. Matriks gambar (mat) akan dihitung per dua elemen dan variabel hasil adalah matriks co-occurrence.
contrast(mat)	Fungsi untuk mencari komponen nilai kontras dari sebuah matriks gambar.
homogeneity(mat)	Fungsi untuk mencari komponen nilai homogeneity dari sebuah matriks gambar.
entropy(mat)	Fungsi untuk mencari komponen nilai entropi dari sebuah matriks gambar.
compare(gambar1, gambar2)	Fungsi membandingkan dua gambar (gambar input dan gambar database) dan mengembalikan nilai kemiripan.
process_image_texture(image_path, gambar1, results)	Prosedur untuk memproses gambar dengan CBIR pada mode tekstur.
main_process_texture(gambar1_path, destination_folder)	Fungsi utama CBIR tekstur yang menerima masukan gambar input dan folder database dan mengembalikan list gambar hasil serta waktu eksekusi.

#### 4.3. Struktur Front-End Program

Website Content-Based Image Retrieval (CBIR) kami menggunakan HTML, CSS, dan Java Script. Terdapat 2 landing page yaitu, home.html dan index.html.

##### 1) Home.html

- Introduction : Bagian ini memberikan pengantar singkat tentang Image Processing App. Menggambarkan penggunaan Aplikasi Aljabar Vektor

dalam Sistem Temu Balik Gambar. Menyoroti konsep Content-Based Image Retrieval (CBIR) dan bagaimana aljabar vektor memperkaya pengalaman pencarian gambar.

- How to Use : Bagian ini berisi video youtube yang menampilkan cara singkat penggunaan website
- About Us: Bagian ini memberikan informasi tentang tim pengembang di balik Image Processing App.

## 2) Index.html

Halaman utama (index.html) bertindak sebagai inti dari Content-Based Image Retrieval (CBIR) pada aplikasi Pemrosesan Gambar. Tujuannya adalah memungkinkan pengguna untuk mencari gambar berdasarkan warna dan tekstur.

### Fitur Utama

1. Toggle Button Color dan Texture
  - 1.1. Tujuan: Memungkinkan pengguna memilih mode pencarian berdasarkan warna atau tekstur.
  - 1.2. Implementasi: Menggunakan toggle button dengan opsi "Color" dan "Texture". Pilihan ini mempengaruhi cara gambar diproses pada saat pencarian.
2. Upload Folder dan Image
  - 2.1. Tujuan: Pengguna dapat mengunggah folder berisi gambar atau mengunggah gambar tunggal.
  - 2.2. Implementasi: Menerapkan dua opsi pengungahan, yaitu mengunggah folder (**folderInput**) dan mengunggah gambar tunggal (**fileInput**).
3. Search Button
  - 3.1. Tujuan: Memulai proses pencarian berdasarkan konfigurasi yang dipilih oleh pengguna.
  - 3.2. Implementasi: Tombol "SEARCH" memicu pengiriman formulir ke server untuk memulai pencarian gambar.
4. Elapsed Time

- 4.1. Tujuan: Menampilkan waktu yang diperlukan untuk menyelesaikan proses pencarian gambar.
- 4.2. Implementasi: Setelah selesai melakukan pencarian, informasi waktu yang diperlukan ditampilkan di bagian hasil pencarian.
5. Total Images
  - 5.1. Tujuan: Menampilkan jumlah total gambar yang dianalisis dalam proses pencarian.
  - 5.2. Implementasi: Setelah selesai melakukan pencarian, informasi jumlah total gambar yang dianalisis ditampilkan di bagian hasil pencarian.
6. Tampilan Gambar dan Similarity
  - 6.1. Tujuan: Menampilkan hasil pencarian berupa gambar beserta tingkat kesamaannya.
  - 6.2. Implementasi: Setelah pengguna mengklik tombol "SEARCH", tampilan gambar yang mirip dengan gambar yang dimasukkan ditampilkan di bawahnya. Setiap gambar disertai dengan persentase kesamaan.
7. Download Hasil Pencarian sebagai PDF
  - 7.1. Tujuan: Memberikan pengguna opsi untuk mengunduh hasil pencarian dalam format PDF.
  - 7.2. Implementasi: Tombol "Download PDF" memicu pembuatan file PDF yang berisi gambar-gambar hasil pencarian dan persentase kesamaannya.

### 3) Java Script

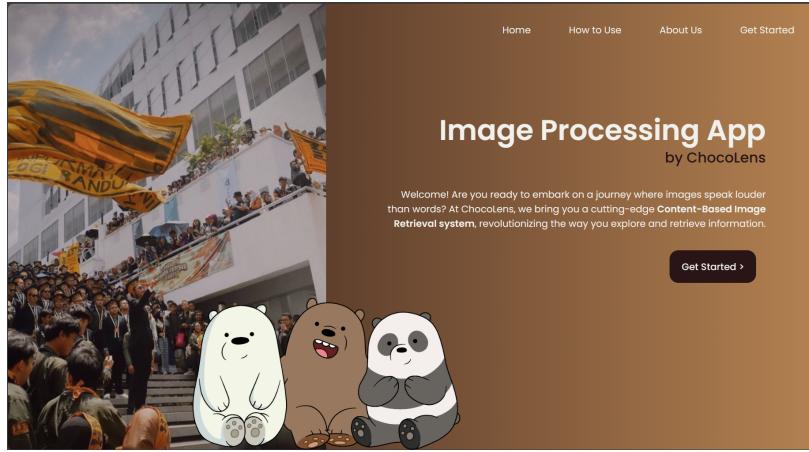
Fungsi / Prosedur	Deskripsi
<code>displayImage(input) :</code>	Fungsi ini ditujukan untuk menangani tampilan gambar setelah dipilih oleh pengguna. Ketika pengguna memilih gambar menggunakan elemen input file,

	fungsi ini membaca gambar tersebut dan menampilkannya di area gambar yang ditentukan. Ini melibatkan penggunaan <code>FileReader</code> untuk membaca konten gambar dan menetapkan gambar tersebut ke elemen HTML.
changeMode () :	Fungsi ini bertanggung jawab untuk mengubah tampilan gambar berdasarkan mode warna atau tekstur yang dipilih oleh pengguna. Saat pengguna mengganti mode menggunakan toggle button, fungsi ini menyesuaikan tampilan gambar sesuai dengan mode yang dipilih.

#### 4.4. Cara Penggunaan Program

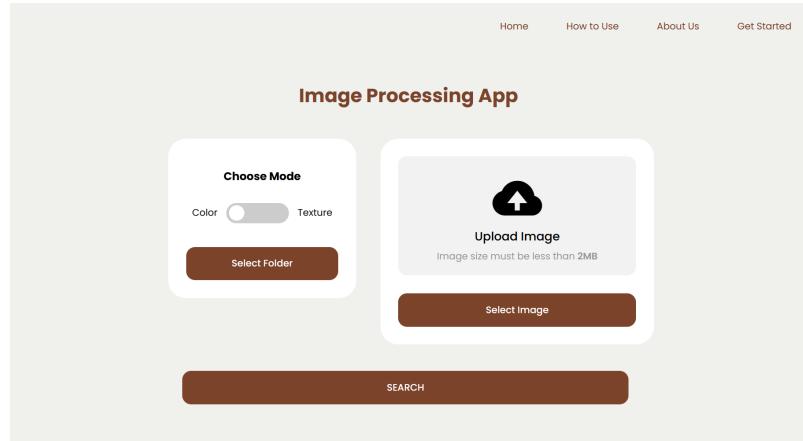
Berikut langkah-langkah yang harus dilakukan untuk memulai dan menggunakan program:

1. Clone repository github pada link yang terdapat pada lampiran, selanjutnya unduh semua *library* yang digunakan pada program. Daftar library yang digunakan terdapat pada *library.txt*. Untuk mengunduh semua *library* yang dibutuhkan dapat menjalankan command berikut pada terminal: `pip install -r requirements.txt`
2. Setelah mengunduh semua *library* program dapat dijalankan dengan perintah `python app.py`
3. Setelah memasuki tampilan web, pengguna akan masuk ke halaman *home*. Disini pengguna dapat melihat dan membaca *section-section* yang tersedia, seperti *How to Use*, dan *About us*.



**Gambar 8.** Tampilan Home

4. Terdapat juga *navigation-bar* yang dapat digunakan oleh pengguna untuk menuju *section* yang diinginkan dan membuka laman untuk mencari kemiripan gambar dengan menekan pilihan *Get Started*.
5. Setelah masuk ke laman *Image Processing*, maka pengguna dapat memasukkan folder *dataset*, gambar *query*, dan memilih mode *Image Processing*. Setelah melakukan memasukkan data, tekan tombol *search* untuk memulai pencarian.



**Gambar 9.** Tampilan Upload

6. Hasil pencarian akan muncul di bagian bawah, terdapat waktu eksekusi, jumlah gambar, dan nilai kemiripan antara gambar *query* dengan gambar yang ada di dataset.



**Gambar 10.** Tampilan Hasil

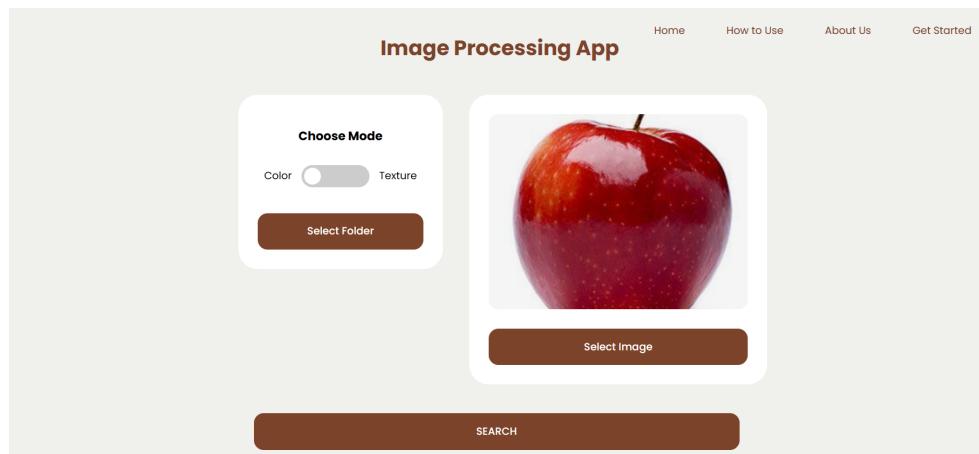
7. Jika pengguna ingin mengunduh hasil pencarian, pengguna dapat menekan tombol download pdf yang terdapat di bagian bawah. Maka otomatis maka pdf akan terunduh pada komputer.
8. Jika ingin mengunggah gambar lagi, pengguna dapat langsung mengunggah kembali gambar, dataset, serta memilih fitur pada bagian unggah.

#### 4.5. Hasil Pengujian Mode Warna

##### a. Folder Dataset yang Digunakan

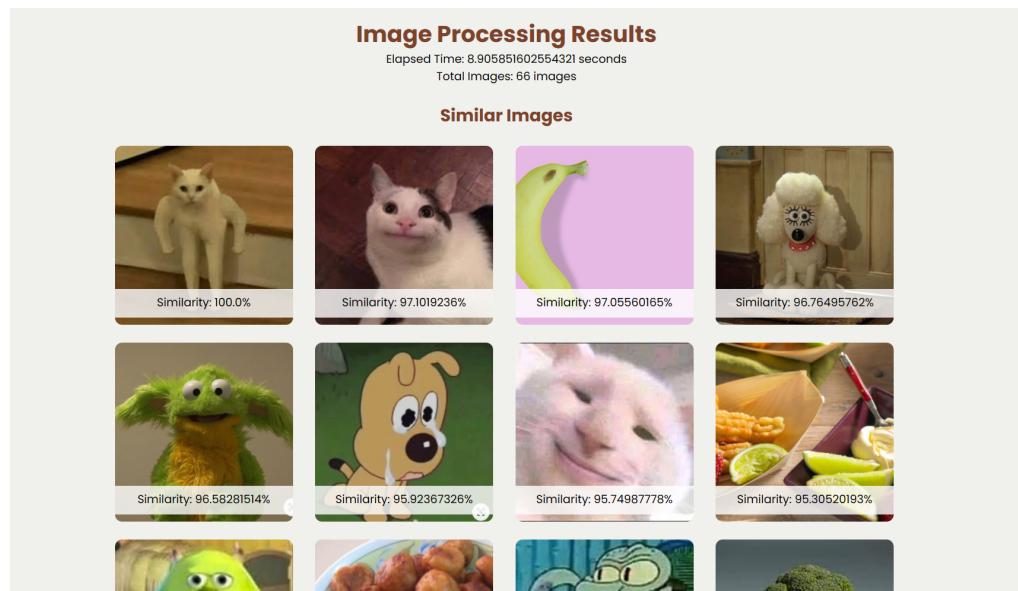
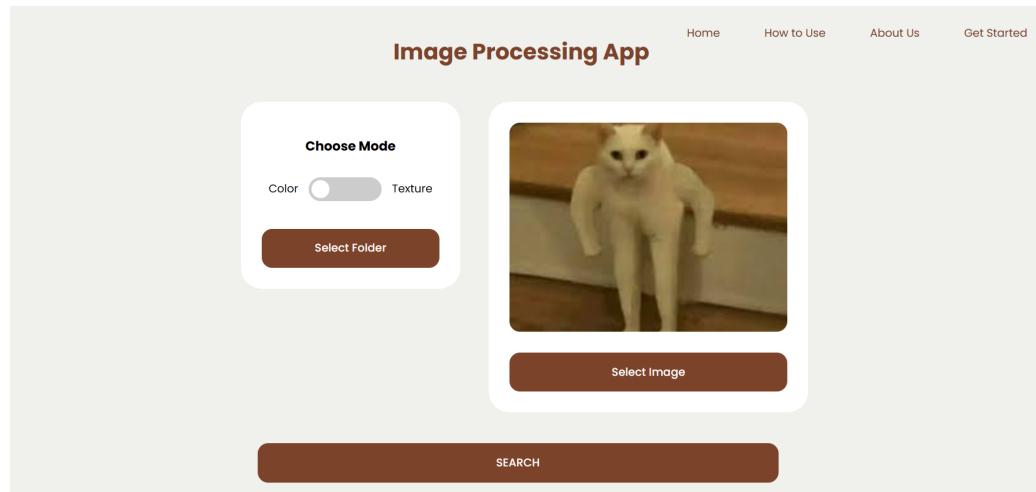
Dataset yang digunakan untuk eksperimen terletak pada folder data yang memiliki 70 gambar dataset yang terdapat pada folder *test* di github.

##### b. Hasil tes





Nama Test Gambar	apel.png
Jumlah Gambar	63 gambar
Waktu Eksekusi	8.937570333 seconds

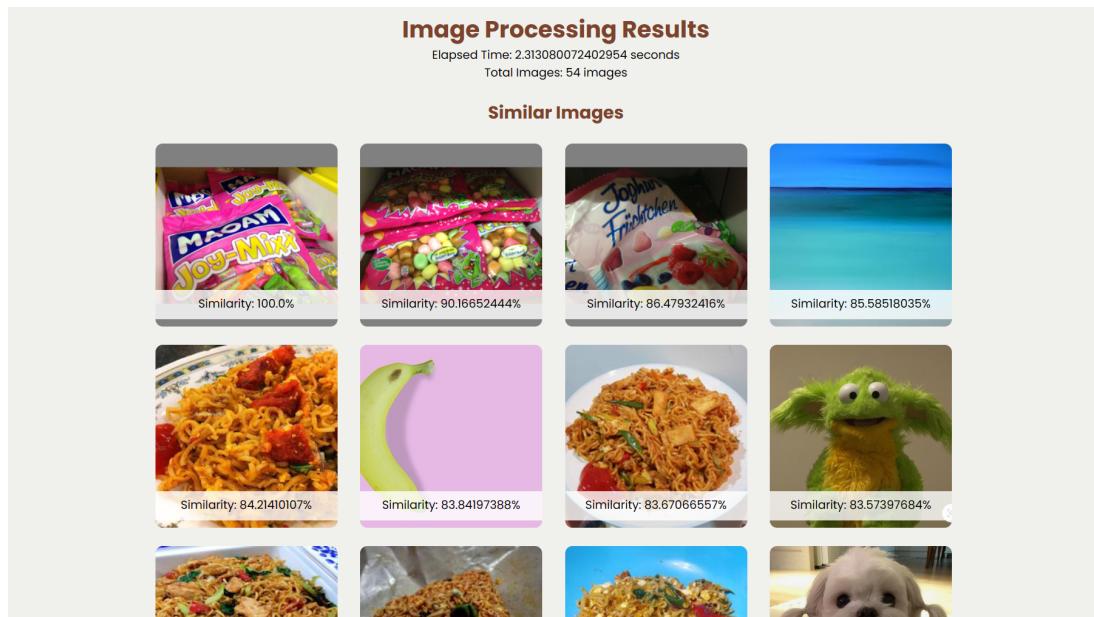
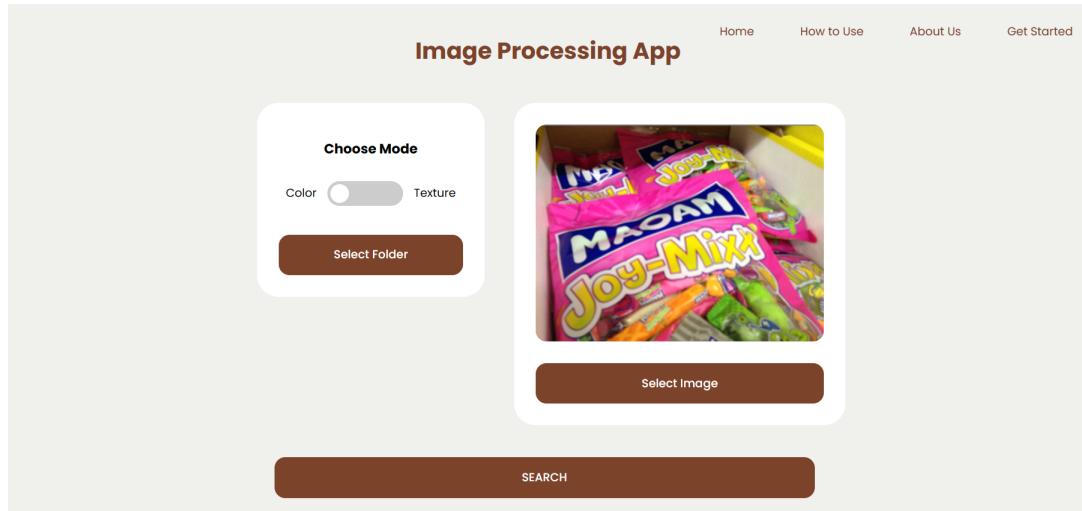


Nama Test Gambar	koceng2.jpeg
Jumlah Gambar	66 gambar
Waktu Eksekusi	8.905851602554321 seconds

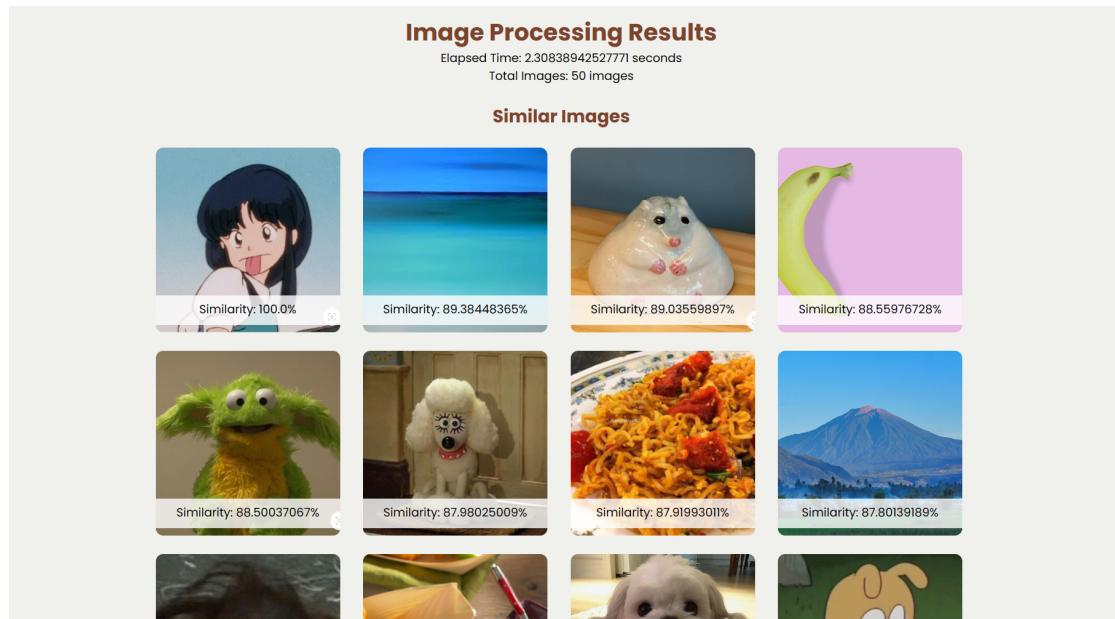
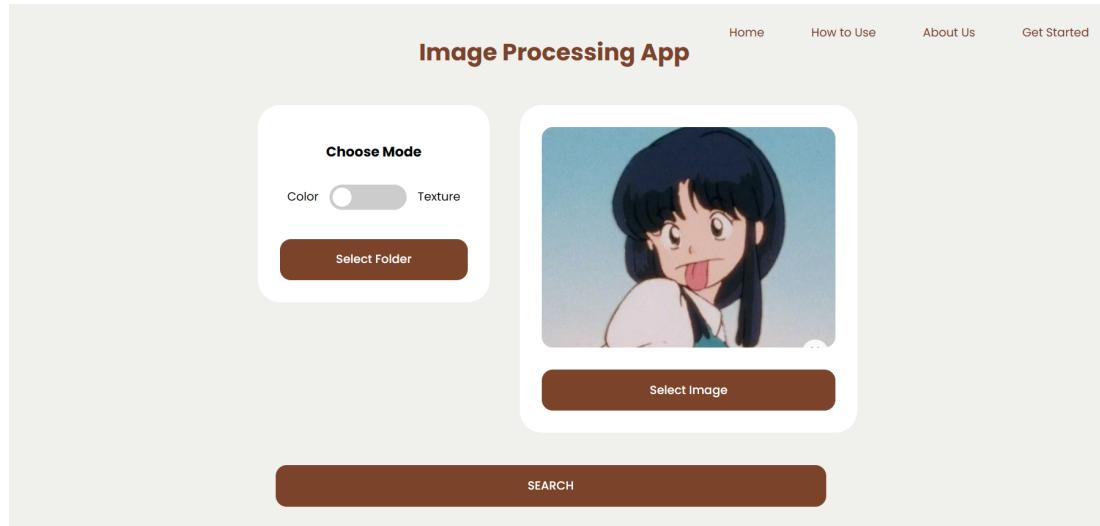
### c. Folder Dataset yang Digunakan

Dataset yang digunakan untuk eksperimen terletak pada folder database yang memiliki 55 gambar dataset yang terdapat pada folder *test* di github.

#### d. Hasil



Nama Test Gambar	CANDY0032.png
Jumlah Gambar	54 gambar
Waktu Eksekusi	2.4301867485046387 seconds



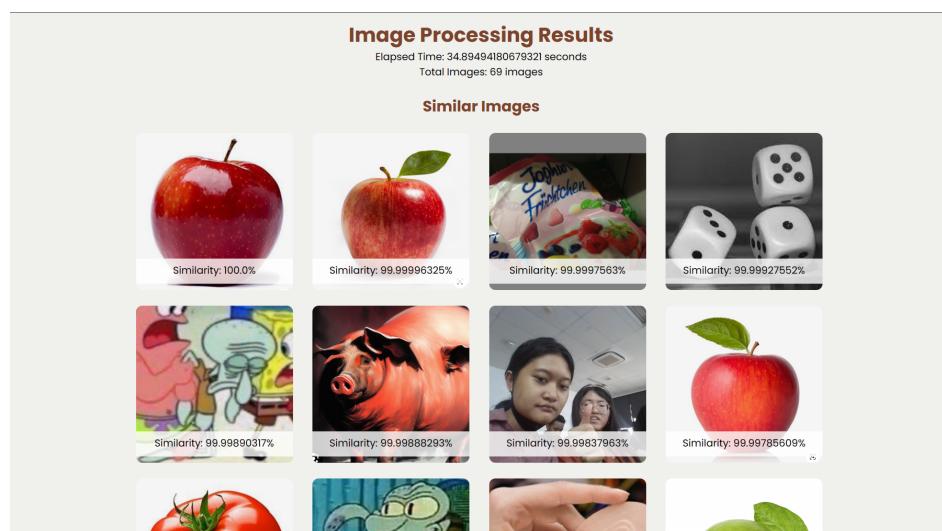
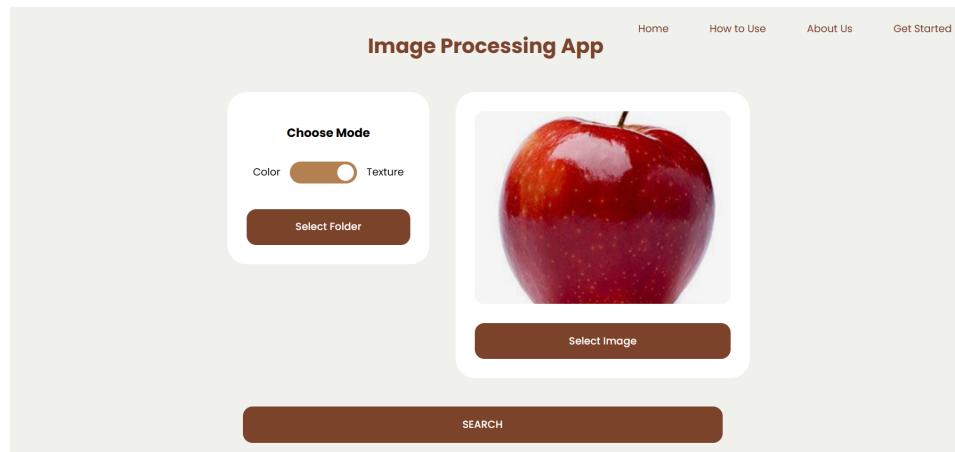
Nama Test Gambar	cewek.png
Jumlah Gambar	50 gambar
Waktu Eksekusi	2.30838942527771 seconds

## 4.6 Hasil Pengujian Mode Tekstur

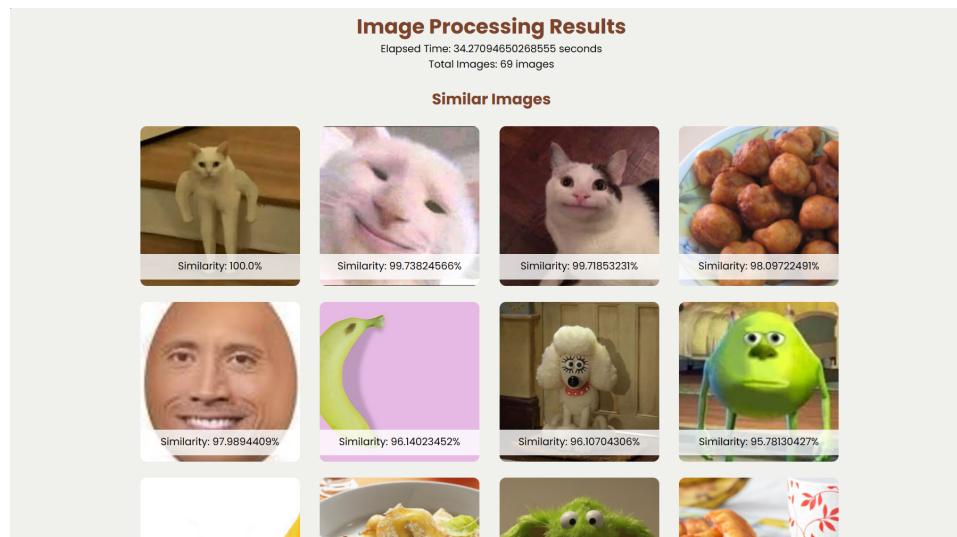
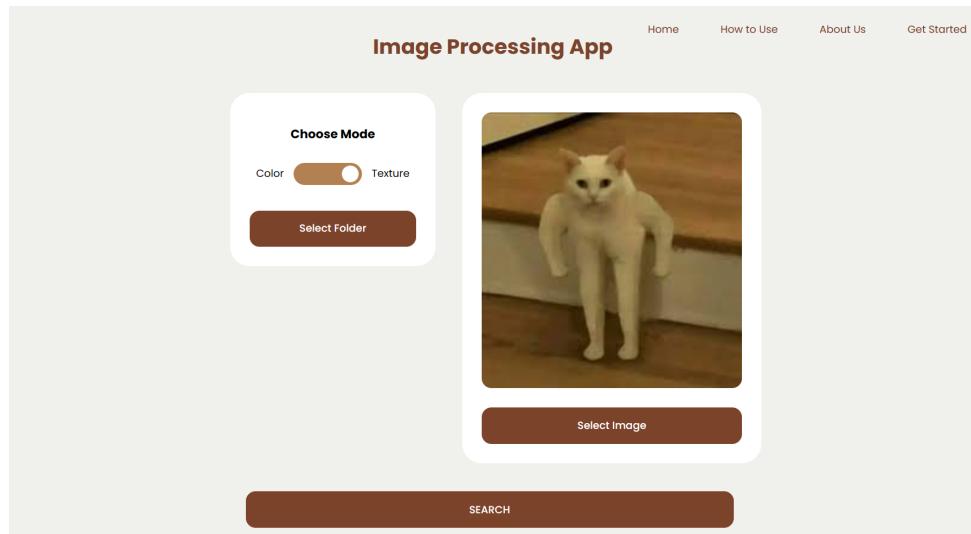
- Folder dataset yang digunakan

Dataset yang digunakan untuk eksperimen terletak pada folder data yang memiliki 70 gambar dataset yang terdapat pada folder *test* di github.

- Hasil tes



Nama Test Gambar	apel.png
Jumlah Gambar	69 gambar
Waktu Eksekusi	34.89494180679321 seconds



Nama Test Gambar	koceng2.jpeg
Jumlah Gambar	69 gambar
Waktu Eksekusi	34.27094650268555 seconds

### c. Folder Dataset yang Digunakan

Dataset yang digunakan untuk eksperimen terletak pada folder database yang memiliki 55 gambar dataset yang terdapat pada folder *test* di github.

#### d. Hasil

Image Processing App

Home How to Use About Us Get Started

Choose Mode

Color  Texture

Select Folder

Select Image

SEARCH

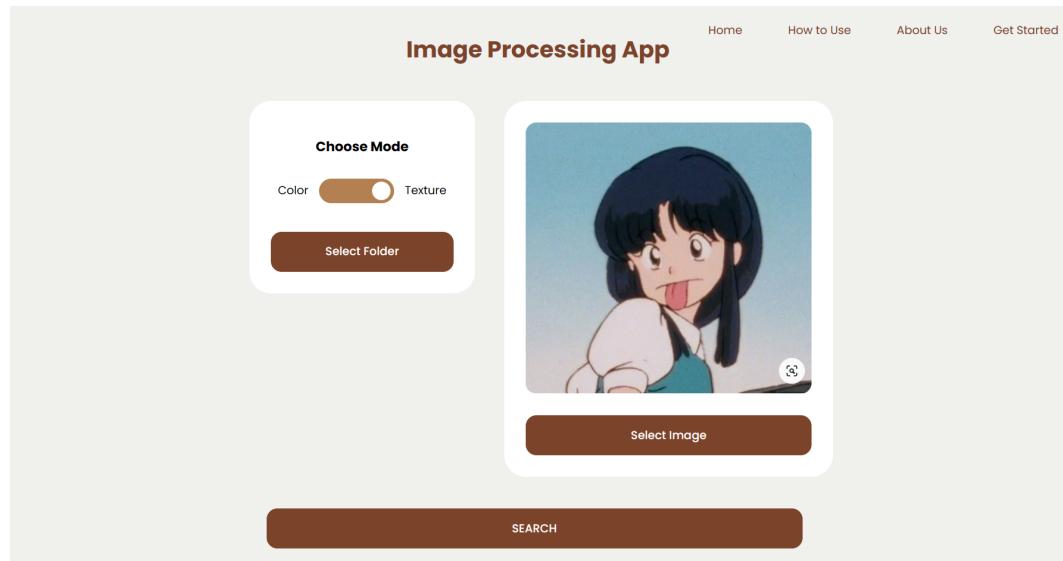
Image Processing Results

Elapsed Time: 8.264402389526367 seconds  
Total Images: 53 images

Similar Images

The screenshot shows the main interface of the 'Image Processing App'. At the top, there's a navigation bar with links to Home, How to Use, About Us, and Get Started. Below that is a 'Choose Mode' section with a toggle switch between 'Color' and 'Texture', and a 'Select Folder' button. To the right is a preview area showing a colorful bag of 'MAOAM Joy-Mixx' candies. Below the preview is a 'Select Image' button. At the bottom is a large 'SEARCH' button. The results section is titled 'Image Processing Results' and displays 53 images. It includes a timestamp of 8.264402389526367 seconds and a note that 53 images were processed. The results are organized in a grid under the heading 'Similar Images', showing various items like apples, ramen, and cartoon characters.

Nama Test Gambar	CANDY0032.png
Jumlah Gambar	53 gambar
Waktu Eksekusi	8.264402389526367 seconds



Nama Test Gambar	cewek.png
Jumlah Gambar	53 gambar
Waktu Eksekusi	9.623229026794434 seconds

## **BAB 5**

### **PENUTUP**

#### **5.1 Kesimpulan**

Dari materi kuliah Aljabar Linier dan Geometri IF2123 yang kami pelajari di kelas seperti matriks dan vektor, kami dapat mengimplementasikan kedua hal tersebut ke dalam sebuah program bahasa python untuk menyelesaikan sebuah kasus. Kami membuat program yang mencari kemiripan gambar dengan metode CBIR warna dan tekstur.

Melalui tugas besar ini, kami mengetahui algoritma dari Content-Based Image Retrieval (CBIR) dan cara untuk membangun sebuah *website*. Kami juga mempelajari algoritma agar pemrosesan gambar lebih efisien. Kami juga mempelajari dan memahami cara pemakaian flask, html, css, dan javascript sebagai bahasa untuk membangun sebuah web dari python. Selain itu, kami juga memahami algoritma dari opencv untuk mengambil gambar input dan mencari hasil kecocokan menggunakan komponen CBIR

#### **5.2 Saran**

Lebih baik sudah mengetahui bahasa-bahasa yang digunakan untuk membangun sebuah web agar proses pengerjaan lebih cepat tanpa harus mempelajari dan memahami bahasa tersebut. Pengerjaan juga lebih baik diselesaikan secepat mungkin agar tidak mengganggu proses belajar maupun tugas lainnya.

#### **5.3 Komentar atau Tanggapan**

Tugasnya sangat menarik dan menantang apalagi terdapat banyak tubes dalam satu waktu. Dan kami baru belajar tentang website, jadi harus belajar banyak hal untuk tubes ini. Deadline tubes ini juga paling cepat di antara lain, padahal banyak yang harus dipelajari. Sehingga waktu yang diberikan untuk tubes ini seharusnya lebih panjang.

#### **5.4 Refleksi terhadap tugas besar**

Melalui tugas besar ini mendapatkan berbagai macam pengalaman. Kami dapat mengasah kemampuan bekerjasama, berkomunikasi, cepat beradaptasi dan belajar hal baru. Karena tugas besar ini, mengharuskan kami untuk mempelajari banyak hal baru

dengan cepat, mengharuskan kami untuk dapat mengatur manajemen waktu dan prioritas dengan baik agar tugas-tugas, kuis, dan materi kuliah tidak tertinggal.

### 5.5 Ruang Perbaikan atau Pengembangan

Lebih baik hasil input gambar dan folder dataset disimpan dengan metode *Caching result* dari perhitungan sebuah dataset menggunakan *output file* tertentu, misalnya: csv, fvecs, dll. Hal ini akan meningkatkan efisiensi dan efektifitas ketika suatu dataset hendak digunakan berulang kali.

Selain itu perhitungan untuk mode tekstur ditambahkan komponen gambar yang lain agar nilai *similarity* lebih akurat. Bisa juga ditambahkan perhitungan matriks co-occurrence dengan sudut yang berbeda-beda untuk mendapat hasil yang paling akurat.

## DAFTAR PUSTAKA

Brain station. (n.d.). *What Is Web Development?* Retrieved from <https://brainstation.io/career-guides/what-is-web-development>

Howard Anton, Chris Rorres. (2013) Elementary Linear Algebra: Applications Version. 11th Edition, John Wiley & Sons Incorporated.

Muhammad, Y. (2020, June 16). *Feature Extraction : Gray Level Co-occurrence Matrix (GLCM)*. Retrieved from <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1>

Munir, R. (2023). Vektor di Ruang Euclidean (bagian 1). Diakses dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir>

Munir, R. (2023). Aplikasi Dot Product pada Sistem Temu-balik Informasi. Diakses dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir>

Program Studi Teknik Infomatika ITB. (n.d.). *Spesifikasi Tugas Besar 2 IF2123 Aljabar Linier dan Geometri*. Retrieved from <https://docs.google.com/document/d/1HVDyywnUdNz9hStgx5ZLqHypK89hWH8qfERJOiDw6KA/edit>

## LAMPIRAN

- **Link Repository**

Link repository kelompok kami untuk tugas besar 2 mata kuliah IF2123 Aljabar Linier dan Geometri adalah sebagai berikut.

**Link : <https://github.com/debrinashika/Algeo02-22025>**

- **Link Youtube**

Link video youtube kelompok kami untuk tugas besar 2 mata kuliah IF2123 Aljabar Linier dan Geometri adalah sebagai berikut:

**Link: <https://youtu.be/ZLtBSGCD3Ks>**