

## **Laporan Tugas Besar 2**

### **IF2123 ALJABAR LINIER DAN GEOMETRI**

Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar



Disusun oleh :

- |                                  |          |
|----------------------------------|----------|
| 1. Rafiki Prawhira Harianto      | 13522065 |
| 2. Muhamad Rafli Rasyiidin       | 13522088 |
| 3. M. Hanief Fatkhani Nashrullah | 13522100 |

**Program Studi Teknik Informatika**  
**Sekolah Teknik Elektro dan Informatika**  
**Institut Teknologi Bandung**  
**2023**

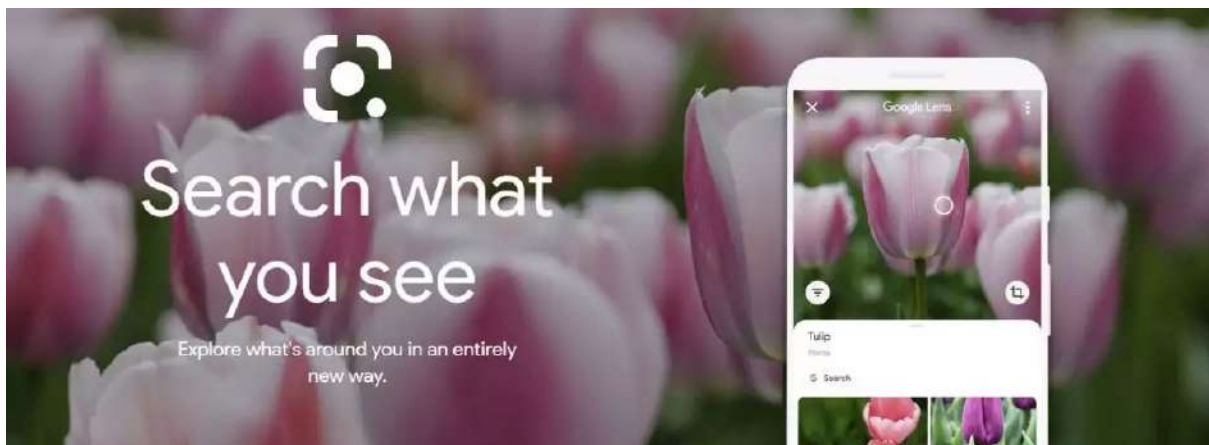
## Daftar Isi

<b>Daftar Isi</b>	<b>2</b>
<b>BAB 1</b>	
<b>Deskripsi Persoalan</b>	<b>3</b>
<b>BAB 2</b>	
<b>Landasan Teori</b>	<b>4</b>
<b>BAB 3</b>	
<b>Analisis dan Pemecahan Masalah</b>	<b>7</b>
<b>BAB 4</b>	
<b>Implementasi dan Uji Coba</b>	<b>10</b>
<b>BAB 5</b>	
<b>Kesimpulan, Saran, dan Komentar</b>	<b>27</b>
<b>Daftar Pustaka</b>	<b>28</b>

## BAB 1

### Deskripsi Persoalan

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



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

## BAB 2

### Landasan Teori

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

*Content-Based Image Retrieval* (CBIR) merupakan teknik untuk mencari gambar yang memiliki tingkat kemiripan cukup tinggi dari suatu *database*. Pencarian dan pengecekan kemiripan gambar tersebut memanfaatkan fitur-fitur yang ada pada gambar, seperti *color*, *contrast*, *entropy*, *homogeneity*, dan *dissimilarity*. Dalam tugas ini, digunakan 2 metode CBIR, yaitu CBIR berdasarkan warna dan CBIR berdasarkan tekstur.

##### 1. CBIR dengan parameter warna

CBIR dengan parameter warna merupakan metode CBIR yang paling penting dan sederhana dalam CBIR. Fitur warna merupakan ciri yang paling jelas dan paling intuitif dari sebuah gambar. Perhitungan fitur atau ciri warna dari sebuah gambar relatif sederhana. Histogram warna merupakan metode yang digunakan untuk ekstraksi fitur warna dari gambar dalam kasus ini.

Dalam ekstraksi fitur, diperlukan penentuan *color space* sebagai representasi warna dari gambar. Umumnya, gambar menggunakan *color space* RGB (*Red*, *Green*, *Blue*). Namun, dalam CBIR, HSV (*Hue*, *Saturation*, *Value*) lebih umum digunakan informasi yang bersifat *noise* sehingga informasi tidak mudah terpengaruh oleh pencahayaan.

Karena umumnya gambar menggunakan *color space* RGB, gambar perlu diubah dalam *color space* HSV dengan rumus :

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

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

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

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

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

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

$$V = C_{max}$$

Setelah didapatkan nilai HSV, kemudian dibentuk vektor yang berisi histogram HSV dari gambar yang diekstrak fiturnya. Vektor tersebut kemudian dibandingkan dengan vektor histogram HSV dari gambar lain untuk mencari tingkat kemiripan berdasarkan warnanya. *Cosine similarity* dari kedua vektor yang menjadi representasi dari kemiripan kedua gambar yang dibandingkan.

$$\cos(\theta) = \frac{V1 \cdot V2}{||V1|| ||V2||}$$

## 2. CBIR dengan parameter tekstur

Tekstur merupakan keadaan permukaan suatu benda atau kesan yang timbul dari apa yang terlihat pada permukaan benda (Ernawati et al, 2008: 204). CBIR dengan parameter tekstur adalah metode CBIR yang menggunakan informasi tekstur dari sebuah gambar untuk melakukan pencarian, perbandingan, dan pengambilan gambar yang serupa. Untuk melakukan hal tersebut, diperlukan fitur-fitur yang diekstrak dari gambar. Ada dua metode yang dapat digunakan untuk mengekstrak fitur dari gambar, yaitu metode Gabor dan metode Haralick. Metode Gabor menggunakan filter Gabor untuk menangkap informasi frekuensi dan orientasi dari tekstur dalam berbagai arah dan skala. Metode Haralick menggunakan *Gray Level Co-Occurrence Matrix* (GLCM) untuk mengukur kemungkinan *co-occurrence* antara pasangan intensitas piksel yang berbeda dalam suatu arah dan jarak tertentu. Untuk membuat GLCM matriks, diperlukan konversi gambar menjadi matriks *grayscale*. Konversi tersebut dapat menggunakan persamaan:

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

Dengan Y adalah nilai *grayscale/graytone*, R adalah komponen warna merah, G adalah komponen warna hijau, dan B adalah komponen warna biru.

GLCM matriks akan digunakan untuk mengekstraksi fitur-fitur gambar, seperti kontras, homogenitas, entropi, ASM, dan energi. Ekstraksi tersebut dapat menggunakan persamaan berikut.

- Kontras

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

- Homogenitas

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

- Entropi

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

- ASM

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

- Energi

$$\sqrt{ASM}$$

## 2.2. Website Development

Website development (atau sering dipanggil “Webdev”) merupakan upaya dalam membangun, mengembangkan, dan memelihara aplikasi yang bekerja melalui internet seperti website. Dalam konteks pelaksanaannya, Webdev dapat dibagi menjadi tiga aspek utama, yaitu Front-end, Back-end, dan Database. Front-end berfokus pada elemen yang terlihat oleh pengguna, Back-end menangani logika, pemrosesan data, dan interaksi dengan database, dan Database menyimpan informasi yang diperlukan oleh aplikasi, seperti gambar-gambar yang dianalisis untuk kemiripannya..

Selain itu, diperlukan suatu kumpulan solusi, infrastruktur teknologi, atau ekosistem data, yang dikenal sebagai Tech Stack. Tech Stack adalah daftar seluruh layanan teknologi yang digunakan dalam pembangunan dan pengembangan website. Pada konteks pengembangan website Tugas Besar ini, Tech Stack yang diterapkan melibatkan HTML, CSS, Javascript, Flask, dan Tailwind.

## BAB 3

### Analisis dan Pemecahan Masalah

Dalam tugas ini, terdapat 2 metode CBIR yang digunakan untuk mengecek tingkat kemiripan gambar, yaitu metode CBIR berdasarkan warna dan tekstur

#### 1. CBIR dengan parameter warna

Pada kasus ini, metode yang digunakan adalah membandingkan histogram dengan *color space* HSV dari satu gambar dengan histogram dari gambar lain. Berikut merupakan langkah-langkah utama dalam ekstraksi fitur warna dari gambar :

- a. Gambar diubah menjadi array terlebih dahulu. Umumnya, gambar menyimpan informasi berupa array dengan informasi intensitas RGB untuk setiap pixelnya.
- b. Setelah array RGB dari gambar didapatkan, array tersebut diubah dalam format HSV menggunakan rumus pada bagian [2.1.1](#) dan kemudian didapatkan array HSV dari gambar yang diekstrak.
- c. Kemudian, array tersebut dibagi menjadi nxn blok untuk meningkatkan akurasi agar setiap blok hanya akan dibandingkan dengan posisi blok yang sama pada gambar lain. Dalam kasus ini digunakan 4x4 blok agar proses ekstrak gambar tidak terlalu lama tetapi tetap akurat.
- d. Kemudian, setiap blok dapat diubah menjadi array 1 dimensi dengan setiap elemen berisi nilai HSV dari gambar yang diekstrak untuk mempermudah perhitungan dan mengurangi langkah dalam pengaksesan elemen.
- e. Setelah diubah menjadi array 1 dimensi, dibentuk vektor berisi histogram dari *hue*, *saturation*, dan *value* dari gambar yang diekstrak. Setiap elemen dikuantifikasi dengan bins sebagai berikut :

$$H = \begin{cases} 0 & h \in [1,25] \\ 1 & h \in [26,40] \\ 2 & h \in [41,120] \\ 3 & h \in [121,190] \\ 4 & h \in [191,270] \\ 5 & h \in [271,295] \\ 6 & h \in [296,315] \\ 7 & h \in [316,360] \end{cases}$$

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

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

- f. Setiap vektor histogram dari *hue*, *saturation*, dan *value* dikonkatenasi menjadi satu vektor HSV.

- g. Kemudian vektor tersebut dibandingkan dengan vektor dari gambar lain dengan menggunakan *cosine similarity*. Semakin tinggi nilai *cosine similarity* maka semakin tinggi kemiripan dari gambar yang dibandingkan.

## 2. CBIR dengan parameter tekstur

Pada tugas ini, digunakan metode Haralick dengan sudut  $0^\circ$  dan jarak 1 untuk melakukan pemrosesan gambar. Alasan digunakannya sudut  $0^\circ$  dan jarak 1 adalah untuk mempercepat proses ekstraksi gambar. Semakin besar sudut dan jarak yang digunakan akan membuat akurasi lebih tinggi, tetapi hal tersebut akan membuat waktu pemrosesan gambar menjadi lebih lama. Dalam perbandingan gambar, ada 3 fitur yang digunakan, yaitu kontras, homogenitas, dan entropi. Alasan penggunaan ketiga fitur tersebut adalah karena ketiga fitur tersebut cukup umum dibandingkan yang lain. Semakin banyak fitur yang digunakan, maka semakin akurat juga yang diberikan.

Berikut merupakan langkah-langkah untuk memproses gambar dengan metode Haralick:

- Lakukan konversi gambar menjadi sebuah matriks RGB. Jika ukuran gambar dirasa terlalu besar, lakukan kompresi terlebih dahulu.
- Konversi matriks RGB menjadi matriks *grayscale* dengan menggunakan persamaan pada bagian [2.1.2](#).
- Lakukan kuantifikasi dari nilai *grayscale*. Nilai Y pada matriks *grayscale* akan berada pada rentang 0-255 sehingga didapatkan level kuantisasi = 256. Buatlah sebuah matriks dengan ukuran  $256 \times 256$  yang semua selnya bernilai 0 (matriks ini akan disebut matriks GLCM). Misalkan X adalah elemen matriks *grayscale* pada baris i dan kolom j, serta Y adalah elemen matriks *grayscale* pada baris i dan kolom j+1, maka tambahkan 1 ke elemen matriks GLCM pada baris X dan kolom Y. Lakukan hal tersebut pada seluruh elemen matriks *grayscale*.

Matriks grayscale				Matriks GLCM			
0	1	1	3	0	1	2	3
1	3	2	0	1	0	0	0
1	2	0	0	2	0	0	0
3	2	1	1	3	0	0	0

Tambahkan 1 pada <i>framework matrix</i> elemen baris 0 dan kolom 1				Matriks GLCM			
				0	1	2	3
				0	0	1	0
				1	0	0	0
				2	0	0	0
				3	0	0	0

Gambar 3.2.1. Proses pembentukan matriks GLCM

Matriks grayscale				Matriks GLCM			
0	1	1	3	0	1	2	3
1	3	2	0	1	0	0	0
1	2	0	0	2	0	0	0
3	2	1	1	3	0	0	0

Tambahkan 1 pada <i>framework matrix</i> elemen baris 1 dan kolom 1				Matriks GLCM			
				0	1	2	3
				0	0	1	0
				1	0	1	0
				2	0	0	0
				3	0	0	0



Gambar 3.2.2. Proses pembentukan matriks GLCM

- d. Setelah terbentuk matriks GLCM, buatlah matriks matriks simetris dengan cara menjumlahkan matriks GLCM dengan matriks transpose-nya.
- e. Kemudian, buatlah matriks normalisasi dari matriks simetris dengan menggunakan persamaan:

$$\textit{Normalization Matrix} = \frac{\textit{Symmetric Matrix}}{\Sigma \textit{Symmetric Matrix}}$$

- f. Setelah matriks normalisasi terbentuk, cari kontras, homogenitas, dan entropi menggunakan persamaan pada bagian [2.1.2](#).
- g. Masukkan ketiga fitur tersebut ke dalam suatu vektor sehingga terbentuk [kontras, homogenitas, entropi]. Vektor tersebut akan digunakan dalam perbandingan tingkat kemiripan gambar.
- h. Lakukan perbandingan terhadap setiap gambar dengan menggunakan Teorema *Cosine Similarity*. Semakin besar nilai *cosine similarity*, maka tingkat kemiripannya akan semakin tinggi.

## BAB 4

### Implementasi dan Uji Coba

#### 1. Implementasi Program Utama

Terdapat 10 bagian dari program utama, yaitu

##### a. Cosine Similarity

```
function cosineSimilarity(vector1 : array of real, vector2 : array of real)->(real)
  USE math
  AB<-0
  lenA<-0
  lenB<-0
  i traversal [0..len(vector1)-1]
    AB<-AB+vector1[i]*vector2[i]
  i traversal [0..len(vector1)-1]
    lenA<-lenA+vector1[i]**2
  i traversal [0..len(vector2)-1]
    lenB<-lenB+vector2[i]**2
  lenA<-math.sqrt(lenA)
  lenB<-math.sqrt(lenB)
  -> AB/(lenA*lenB)
```

##### b. RGB to HSV

```
function rgb2hsv(r : integer, g : integer, b : integer)->(integer, real, real)
  r'<-r/255 ; g'<-g/255 ; b'<-b/255
  cmax<-max(r',g',b')
  cmin<-min(r',g',b')
  diff<-cmax-cmin
  hue<-0
  s<-0
  v<-cmax
  if diff = 0 then
    pass
  else
    depend on cmax
      cmax = r' : hue<-((g'-b')/diff)%6
      cmax = g' : hue<-((b'-r')/diff)+2
      cmax = b' : hue<-((r'-g')/diff)+4
  hue<-hue*60
  if cmax!=0 then
    s<-diff/cmax
  -> (hue,s,v)
```

##### c. getHistogram

```
function getHistogram(filename : string)->(array of integer)
  USE numpy
  USE PIL.Image
  OPEN(filename,image)
  sizeReso<-512
  blocking<-4
  hueBins<-[1,25,40,120,190,270,295,315,360]
  svBins<-[0,0.2,0.7,1]
  arr<-numpy.array(image)
  img<-numpy.split(arr,blocking)
  img_hsplitted<-[numpy.hsplitted(i,blocking) for i in img]
  img_ravel<-[]
  i traversal [0..blocking-1]
    j traversal [0..blocking-1]
      img_ravel.append(img_hsplitted[i][j].ravel())
  hsvarr<-[rgb2hsv_arr(i) for i in img_ravel]

  histogram<-[numpy.concatenate([numpy.histogram(i[0],bins=hueBins)[0],numpy.histogram(i
[1],bins=svBins)[0],numpy.histogram(i[2],bins=svBins)[0]]) for i in hsvarr]
```

```
CLOSE(filename)
-> histogram
```

d. RGB to *grayscale*

```
function rgbtograyscale(r,g,b : integer) -> real
-> (0.299*r + 0.587*g + 0.114*b)
```

e. RGB to *grayscale matrix*

```
function rgbtograyscale_array(m : matrix) -> matrix
mOutput <- []
i traversal [0..len(m)]
  j traversal [0..len(m[0])]
    mOutput.append(int(rgbtograyscale(m[i][j][0], m[i][j][1], m[i][j][2])))
-> mOutput
```

f. Create GLCM Matrix

```
function GCLMmat(m : matrix) -> matrix
framework <- numpy.zeros((256,256))
i traversal [0..len(m)]
  j traversal [1..len(m[0])]
    framework[m[i][j-1]][m[i][j]] <- framework[m[i][j-1]][m[i][j]] + 1
-> framework
```

g. Create Symmetric Matrix

```
function symmetricGCLM(m : matrix) -> matrix
framework <- []
transposed <- m.transpose()
i traversal [0..len(m)]
  j traversal [0..len(m[0])]
    framework.append <- m[i][j] + transposed[i][j]
-> framework
```

h. Create Normalization Matrix

```
function normalizeGCLM(m : matrix) -> matrix
sum <- m.sum()
i traversal [0..len(m)]
  j traversal [0..len(m[0])]
    m[i][j] <- m[i][j]/sum
-> m
```

i. Create Feature

```
function createThreeFeature(m : matrix) -> array of real
contrast <- 0
homogeneity <- 0
entropy <- 0
i traversal [0..len(m)]
  j traversal [0..len(m[0])]
    contrast <- contrast + m[i][j]*(i-j)*(i-j)
    homogeneity <- homogeneity + (m[i][j]/(1+(i-j)*(i-j)))
    if m[i][j] != 0 then
      entropy <- entropy + (m[i][j]*(math.log(m[i][j])))
entropy <- -1*entropy
-> [contrast, homogeneity, entropy]
```

## j. Normalize Feature and Cosine Similarity

```

function normalizedFeatureAndCosine(feature_array : matrix of array, main_feature :
array of real) -> array of tuple
    feat_arr <- []
    i traversal [0..len(feature_array)]
        feat_arr.append(feature_array[i][0])
    mean_feature_array <- numpy.sum(feat_arr, axis=0)*(1/len(feat_arr))
    std_feature_array <- np.std(feat_arr, axis=0)
    i traversal [0..len(feat_arr)]
        feat_arr[i][0] <- (feat_arr[i][0]-mean_feature_array[0])/std_feature_array[0]
        feat_arr[i][1] <- (feat_arr[i][1]-mean_feature_array[1])/std_feature_array[1]
        feat_arr[i][2] <- (feat_arr[i][2]-mean_feature_array[2])/std_feature_array[2]

    main_feature[0] <- (main_feature[0]-mean_feature_array[0])/std_feature_array[0]
    main_feature[1] <- (main_feature[1]-mean_feature_array[1])/std_feature_array[1]
    main_feature[2] <- (main_feature[2]-mean_feature_array[2])/std_feature_array[2]
    key <- []
    i traversal [0..len(feat_arr)]
        similarity <- cosineSimilarity(feat_arr[i], main_feature)
        if similarity >= 0.6 then
            if similarity > 1 then
                similarity <- 1
            key.append([similarity*100, feature_array[i][1]])
    key.sort(reverse=True)
    -> key

```

## 2. Struktur Program

Program ini merupakan program yang dibuat dengan *framework web application* Flask. Program ini bekerja dengan memuat beberapa *function* yang telah dibuat dan juga beberapa *library* terlebih dahulu ketika awal program mulai berjalan. *Library* yang cukup banyak digunakan dalam program ini adalah Numpy, Pillow, dan juga Numba. *Function* yang telah dibuat diantaranya adalah “extraction\_func.py”, “load\_features.py”, dan “feature\_extraction.py”. *Function* dan *library* yang telah dimuat akan digunakan untuk mengekstrak *features* dari gambar. Program akan meminta dataset terlebih dahulu, kemudian memasukkannya ke folder static/dataset/. Kemudian program akan langsung mengekstrak dan memuat *features* dari setiap dataset agar dapat digunakan berulang kali selama dataset yang digunakan masih sama. Lalu, untuk mencari gambar yang mirip dari input pengguna, *features* dari input gambar diekstrak kemudian dibandingkan satu per satu dengan *features* dataset yang telah dimuat sebelumnya.

Berikut adalah struktur direktori dari program

```

test/
├─ dataset1/
├─ dataset2/
├─ dataset3/
├─ imagetest/
doc/
├─ Algeo02-22065.pdf
img/
├─ elaina1.png
├─ elaina2.png
├─ elaina3.png
├─ elaina4.png
src/
├─ app.py
├─ requirements.txt
├─ extraction_func.py
├─ load_features.py
├─ tailwind.config.js
├─ package.json
├─ feature_extraction.py
├─ package-lock.json
├─ dataset_features/
│   └─ hsv/
│       └─ sample.jpg.npy
│   └─ texture/
│       └─ sample.jpg.npy
├─ static/
│   └─ css/
│       └─ main.css
│   └─ dataset/
│       └─ sample.jpg
│   └─ script/
│       └─ script.js
│       └─ pagination.js
│   └─ src/
│       └─ input.css
│   └─ uploads/
│       └─ image_sample.jpg
│   └─ Image/
│       └─ fotokelompok.jpg
│       └─ sample.jpg
├─ templates/
│   └─ index.html
│   └─ about_page.html
├─ .gitignore
└─ README.md

```

Berikut adalah penjelasan mengenai file ataupun folder dari program

a. app.py

Merupakan file utama konfigurasi dari *web application* Flask. File inilah yang berjalan ketika *web application* sedang berjalan. File ini bertanggung jawab untuk membuat route URL, menyimpan variable, render template HTML, dan juga untuk menjalankan *web application*.

b. extraction\_func.py

Merupakan file yang berisi fungsi untuk mengekstrak *features* dari dataset dan juga gambar yang akan dicari. Fungsi seperti

pengolahan matriks, operasi vektor, konversi *color space*, ataupun *cosine similarity* berada dalam file ini.

c. `load_features.py`

File ini merupakan file yang bertanggung jawab untuk memuat *features* yang telah dibuat sebelumnya. File ini juga memiliki prosedur untuk menghapus *features* dan juga gambar ataupun dataset yang tidak digunakan lagi.

d. `feature_extraction.py`

File ini merupakan file yang dijalankan ketika dataset telah diunggah. File ini akan mengekstrak *features* dari dataset, kemudian menyimpannya dalam file berekstensi “.npz” Ketika file ini selesai dijalankan, file akan mengembalikan nilai berapa banyak gambar yang telah diproses dan lama waktu pemrosesan seluruh gambar.

e. `dataset_features/`

Folder ini merupakan tempat untuk menyimpan *features* dari dataset. Setiap gambar memiliki dua file *features*, satu untuk parameter *color* dan satu untuk parameter *texture*.

f. `script/`

Folder ini merupakan folder penyimpanan script yang bertanggung jawab dalam perilaku tampilan website dan juga proses upload file. File seperti “script.js” mengatur proses submit file, tampilan animasi, dan juga menentukan posisi *toggle* dari *color/texture*. “Pagination.js” merupakan *script* yang bertanggung jawab untuk mengatur pagination dari hasil pencarian gambar.

g. `uploads/`

Folder ini berisi input gambar dari pengguna. Setiap kali pengguna mengunggah gambar, folder ini akan dikosongkan terlebih dahulu sehingga jumlah file dalam folder ini dijamin hanya akan berjumlah satu file.

h. `dataset/`

Folder ini berisi dataset yang diunggah oleh pengguna. Seluruh gambar pada dataset ini tetap disimpan karena akan digunakan untuk menampilkan hasil gambar yang dicari. Seluruh gambar dalam folder ini akan dihapus jika pengguna memasukkan dataset baru.

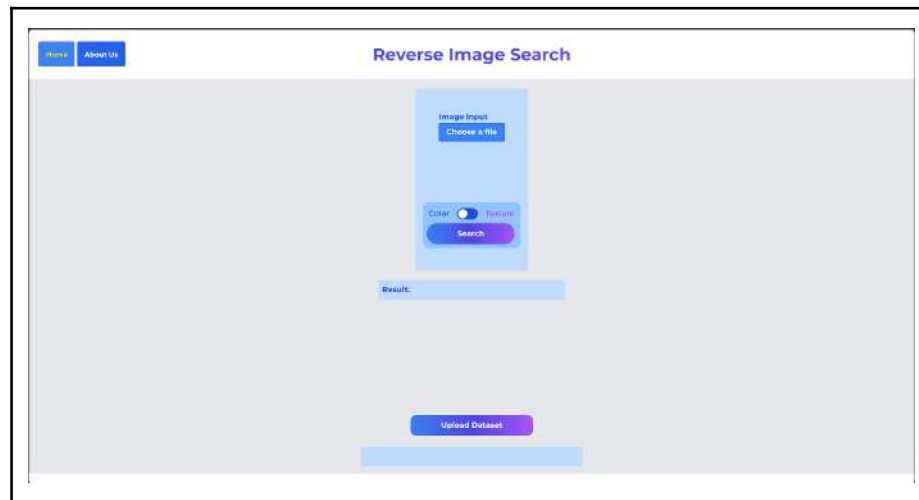
i. `templates/`

Folder ini berisi file html untuk ditampilkan ke pengguna. “index.html” merupakan halaman utama dan “about\_page.html” merupakan halaman yang menjelaskan konsep singkat dari cara

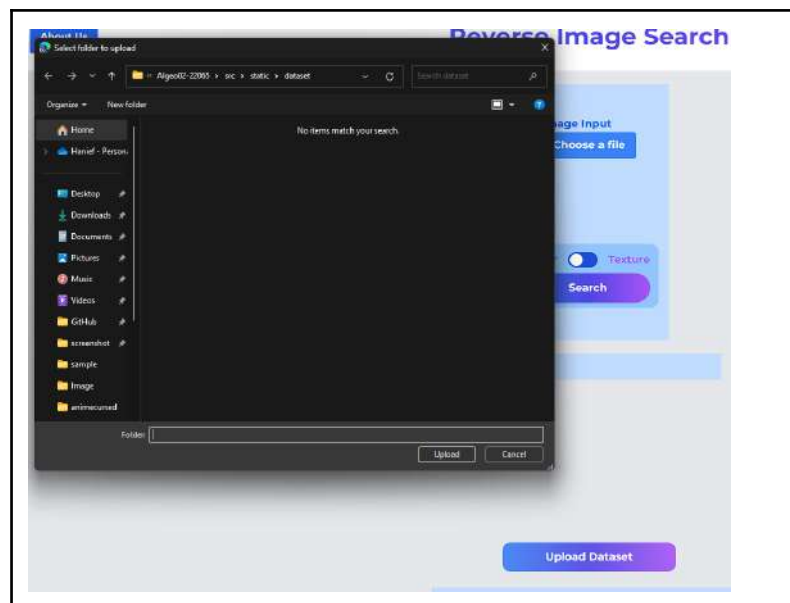
kerja pencarian gambar, cara menggunakan, dan tentang pembuat website.

### 3. Cara Penggunaan Program

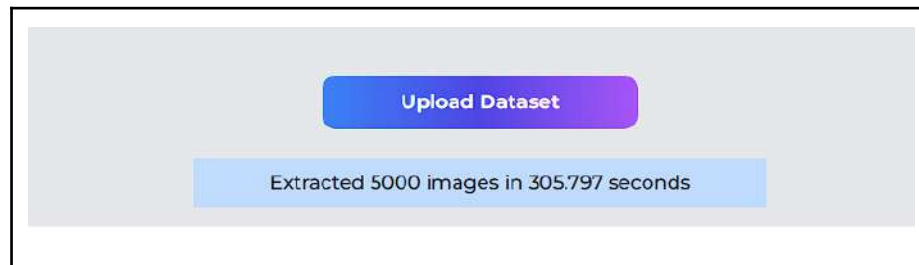
- a. Pertama, program akan menampilkan laman utama kepada pengguna



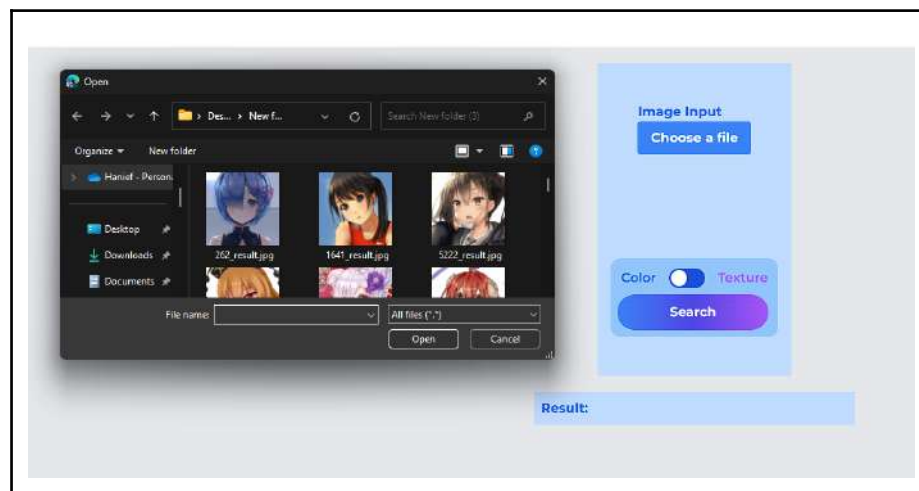
- b. Pengguna dapat langsung memilih gambar input untuk dicari jika sudah terdapat dataset yang termuat. Jika belum, upload dataset dengan memencet tombol “Upload Dataset”



Pengguna akan diminta untuk memilih folder yang berisi dataset yang akan digunakan. Tunggu setelah seluruh gambar dari dataset termuat dan terdapat tulisan “Extracted N images in X seconds” di bagian bawah halaman.

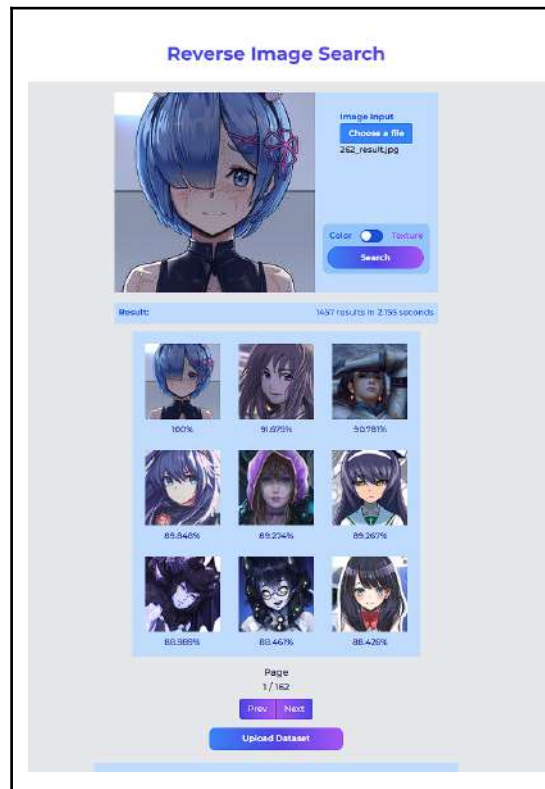


- c. Setelah seluruh gambar dari dataset diekstrak, pengguna dapat memilih gambar untuk mencari gambar lain yang mirip dengan gambar itu sendiri.

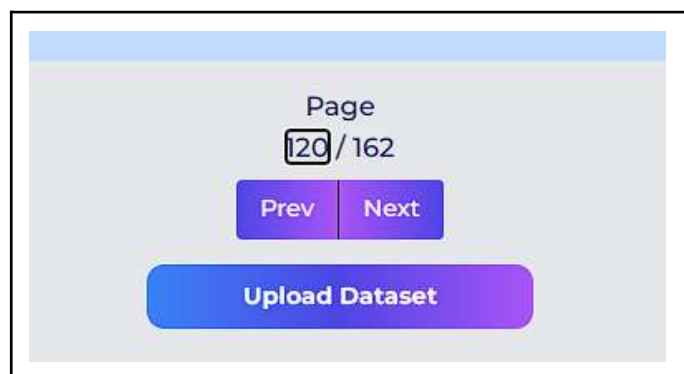




- d. Kemudian pengguna dapat memencet tombol search untuk memulai pencarian. Mode pencarian dapat diganti menjadi color atau Texture. Setelah tombol search dipencet, maka akan muncul hasil pencarian.



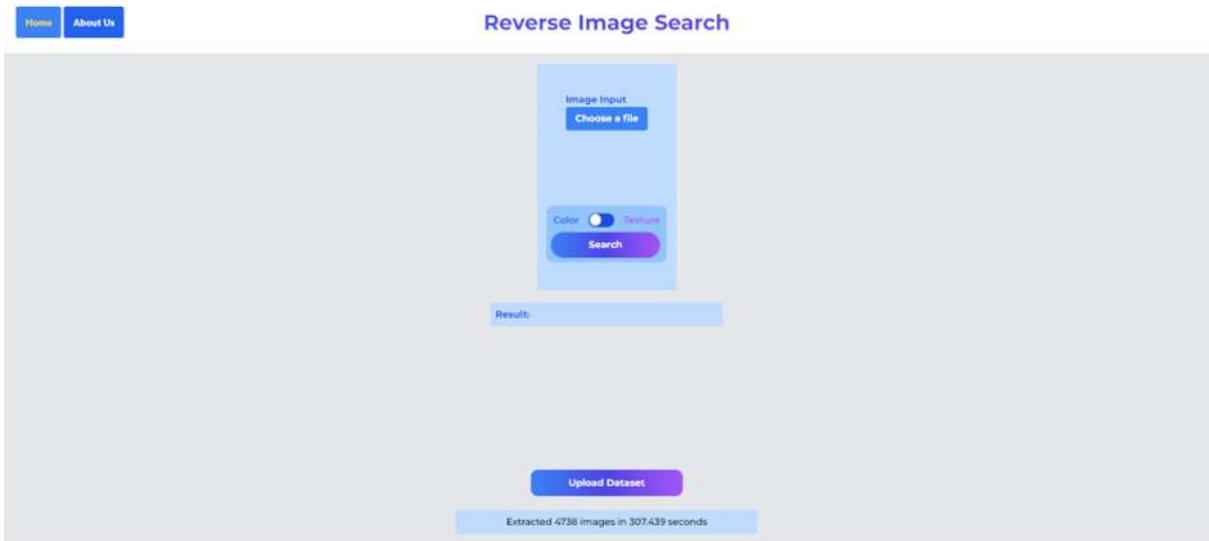
- e. Pengguna dapat mengubah nomor halaman untuk melewati beberapa halaman sekaligus



4. Hasil Pengujian

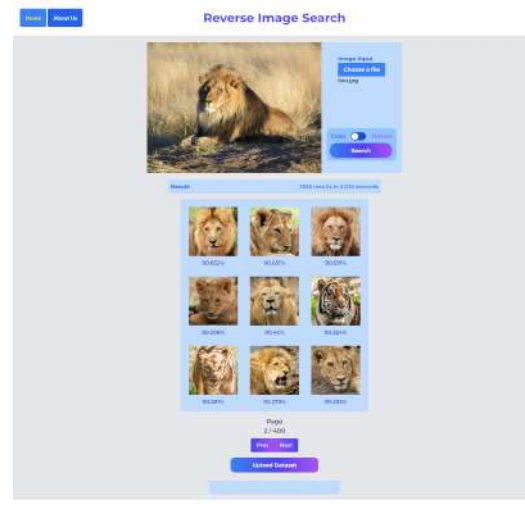
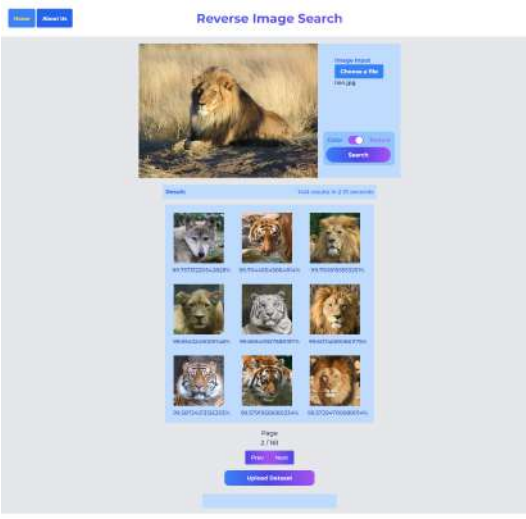
Dataset 1: 4738 Foto Hewan

Durasi Waktu Upload & Ekstrak Dataset: 307,439 detik

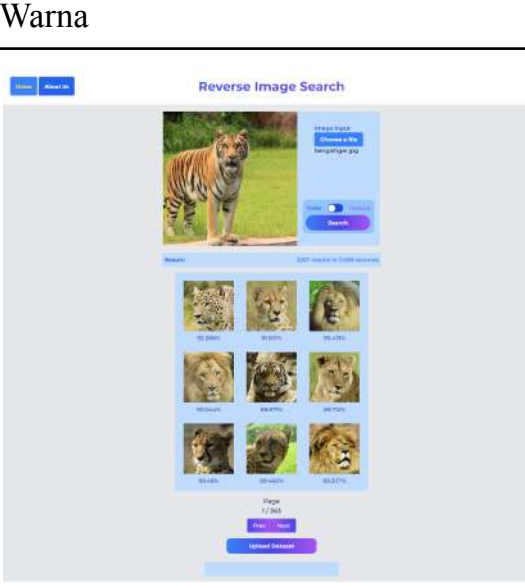
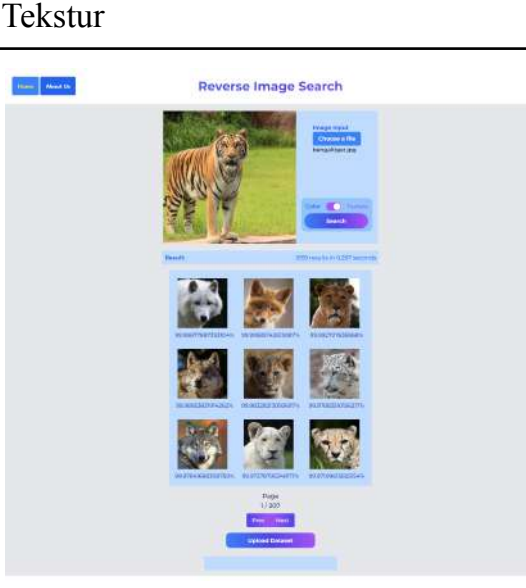


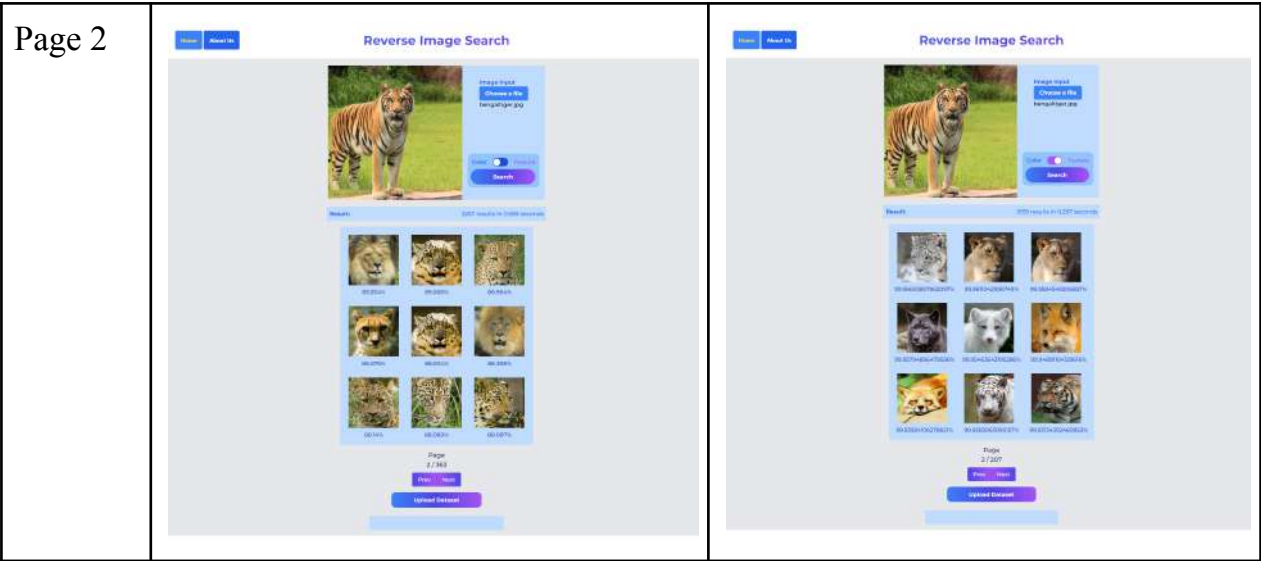
Test Gambar 1: Singa

Singa	Warna	Tekstur
Page 1		

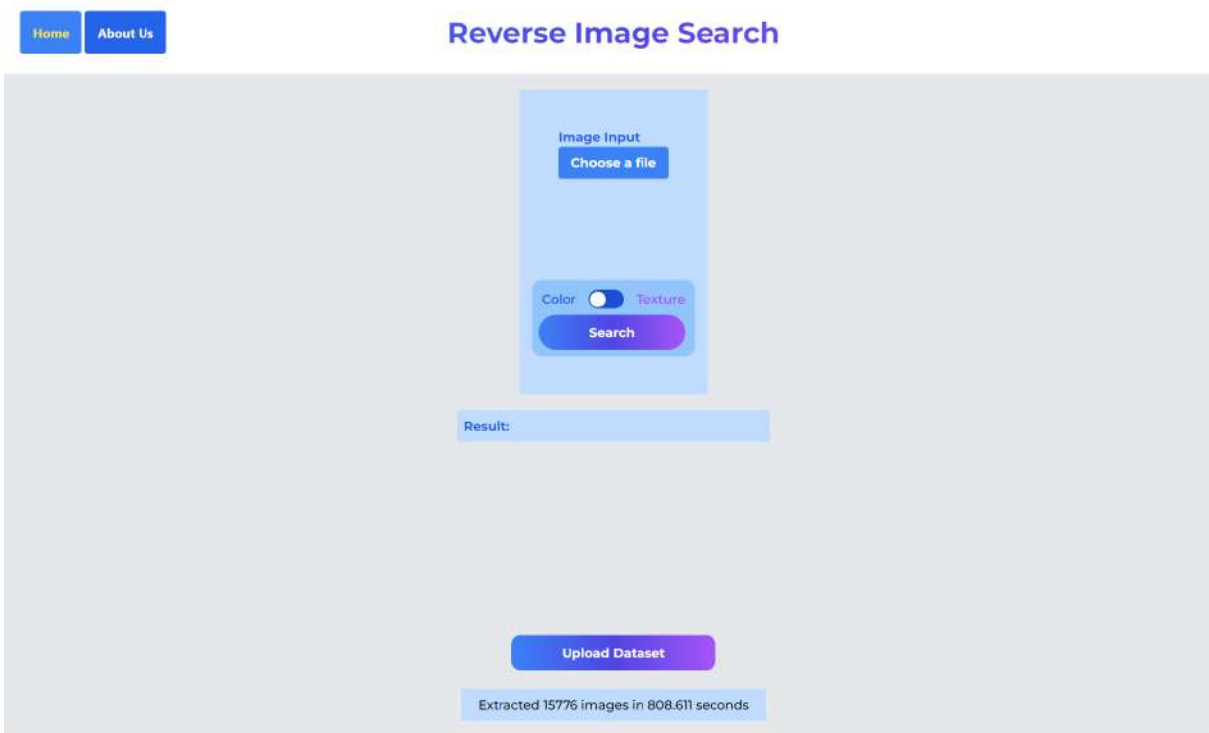
Page 2		
--------	---	--

Test Gambar 2: Harimau

Harimau	Warna	Tekstur
Page 1		

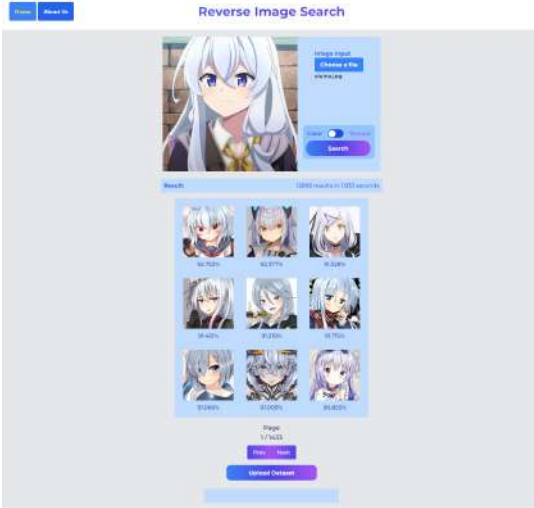
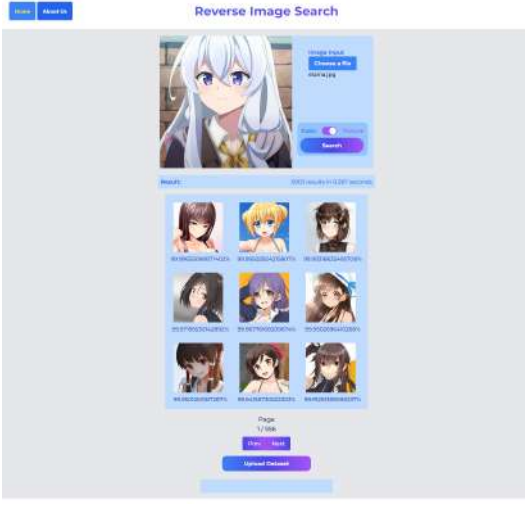
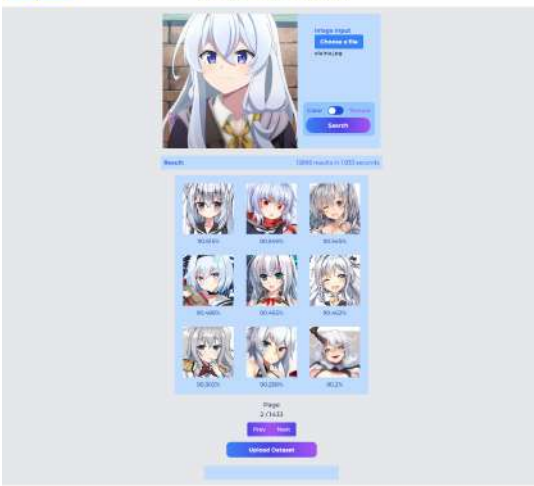
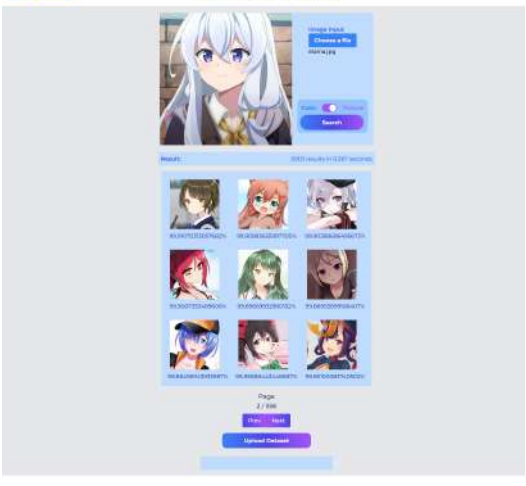


Dataset 2: 15,776 Foto Karakter Anime (dari 92,200)  
Durasi Waktu Upload & Ekstrak Dataset: 808,611 detik



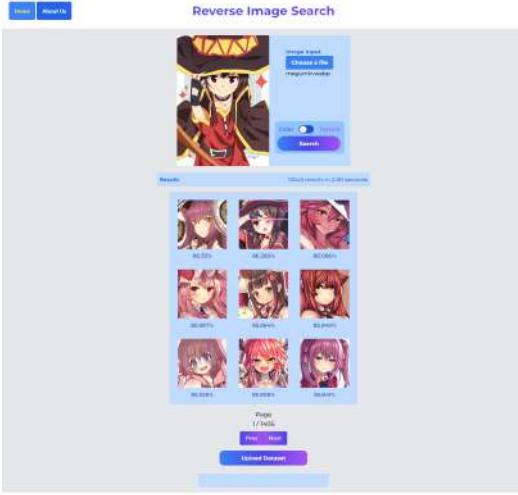
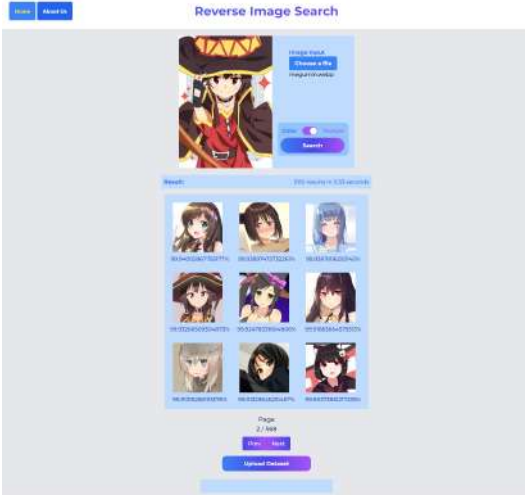
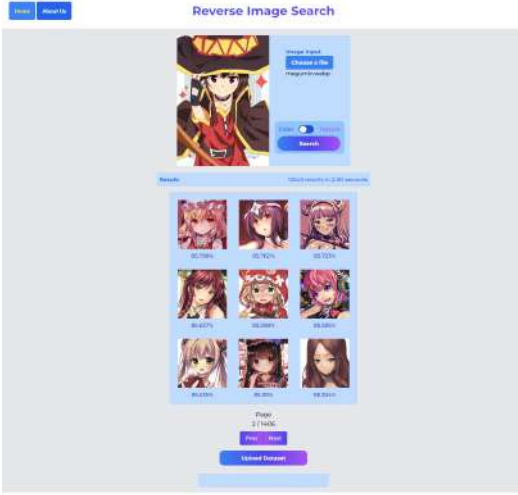
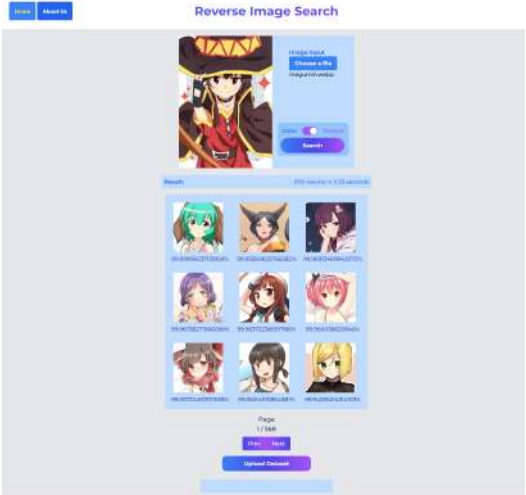
Test Gambar 1: Elaina

Elaina	Color	Texture
--------	-------	---------

Page 1		
Page 2		

Test Gambar 2: Megumin

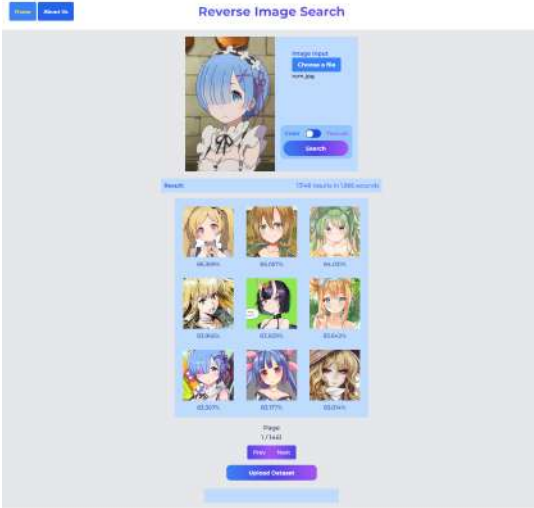
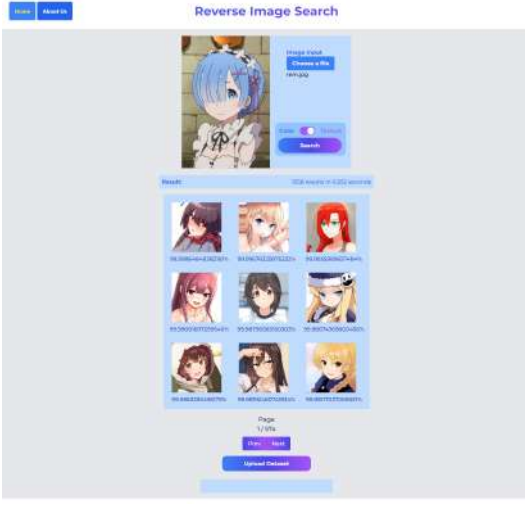
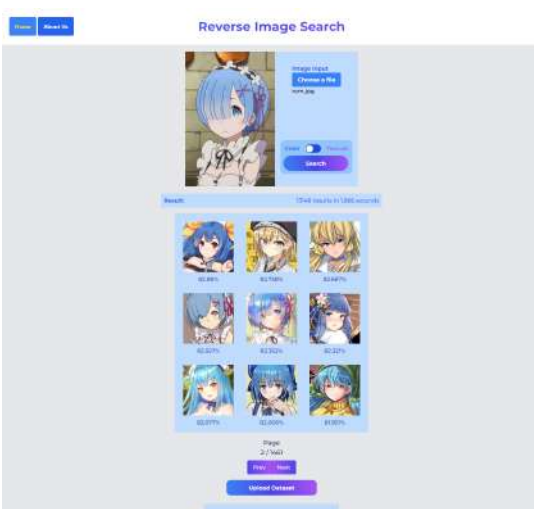
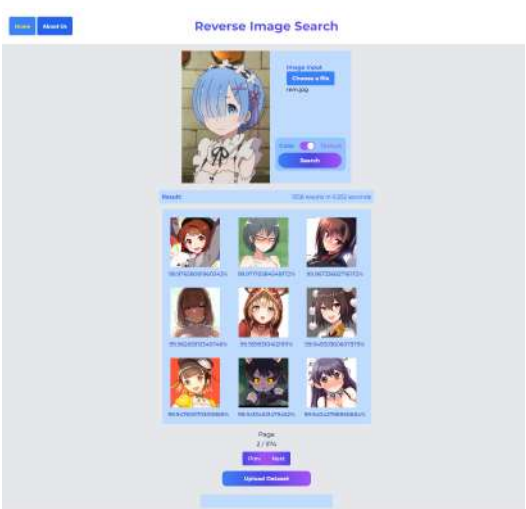
Megumin	Color	Texture
---------	-------	---------

Page 1		
Page 2		

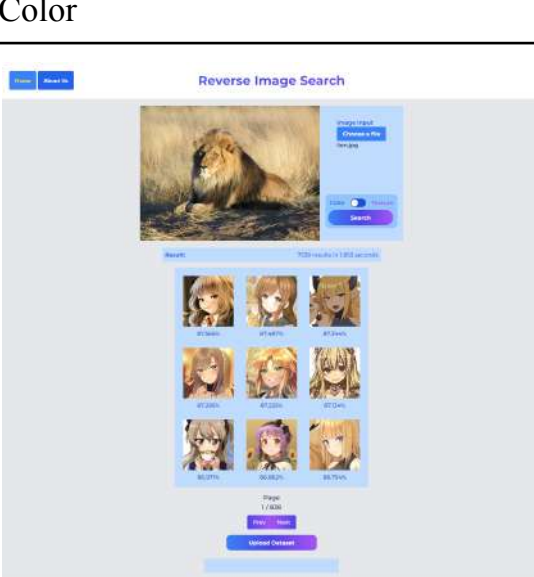
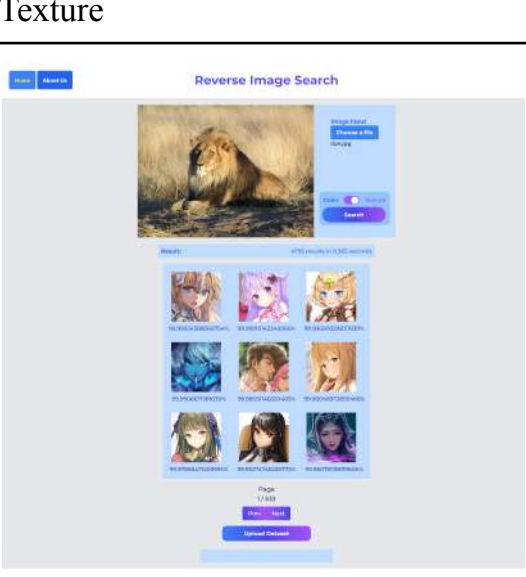
Test Gambar 3: Rem

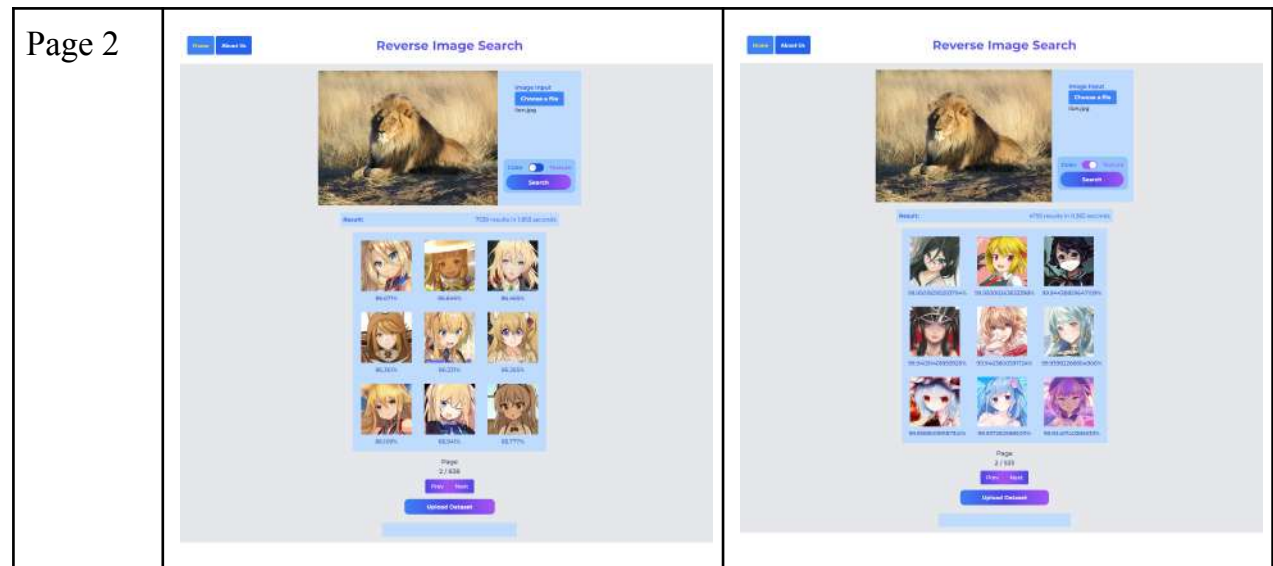
Rem	Color	Texture
-----	-------	---------



Page 1		
Page 2		

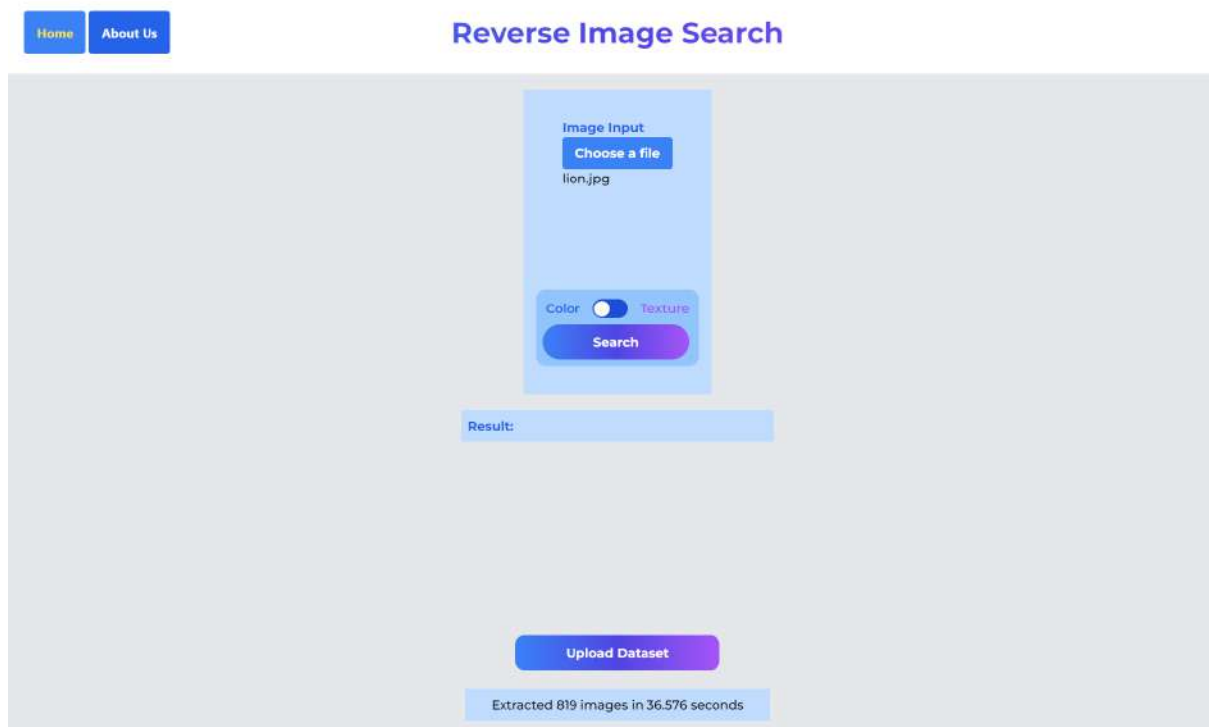
Test Gambar 4: Singa

Singa	Color	Texture
Page 1		



Dataset 3: 819 Foto Pokemon

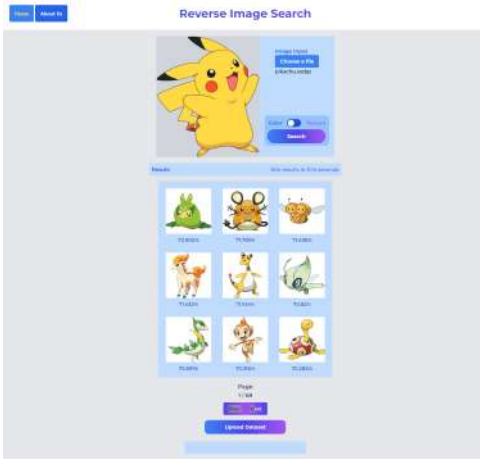
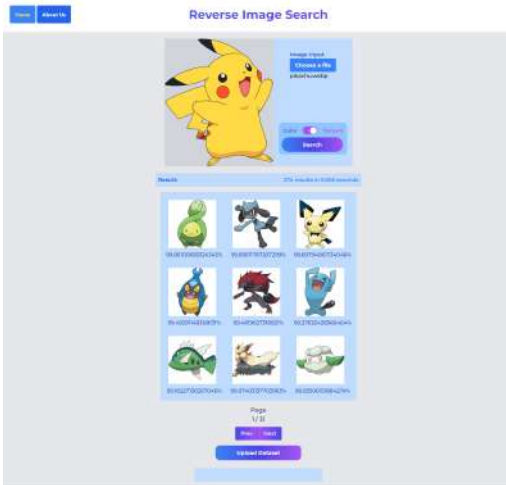
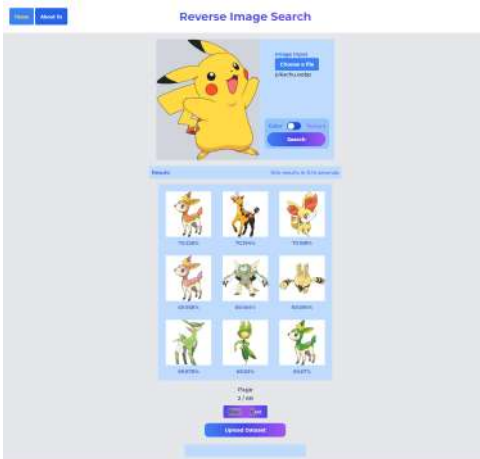
Durasi Waktu Upload & Ekstrak Dataset: 36,576 detik



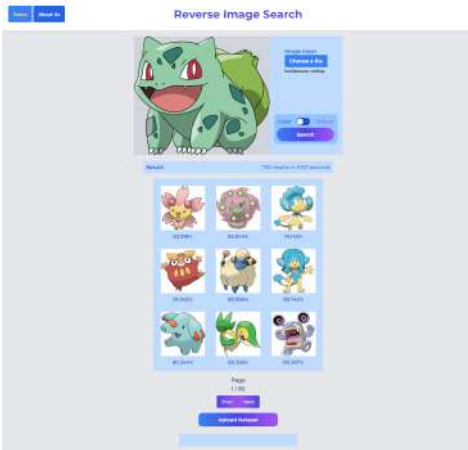
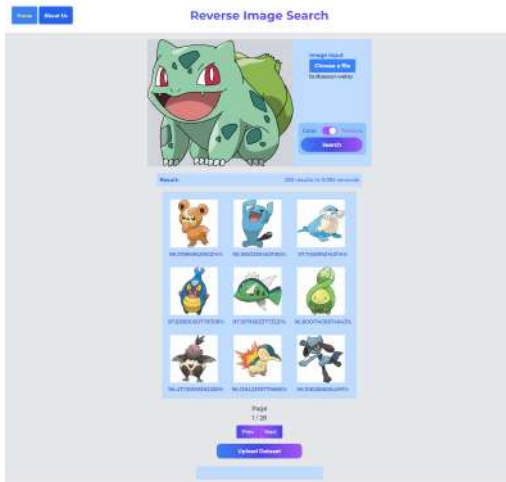
Test Gambar 1: Pikachu

Pikachu	Color	Texture
---------	-------	---------

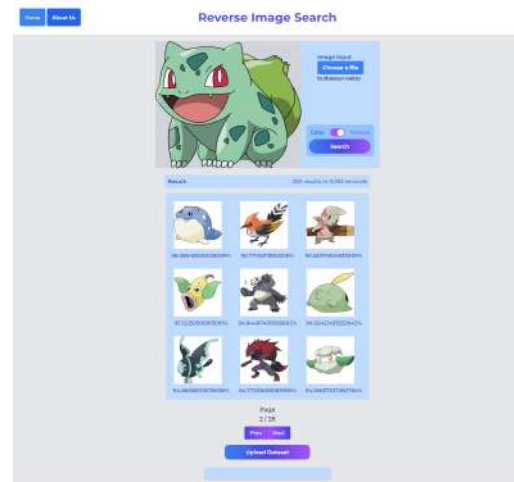
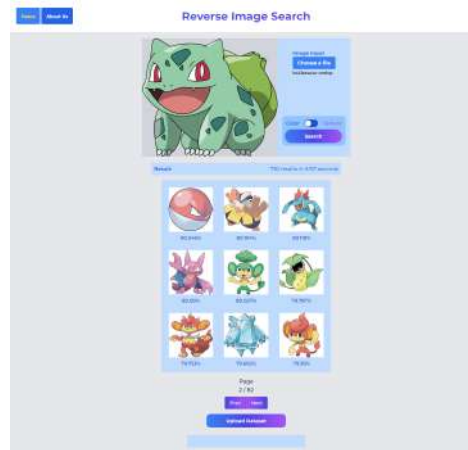


Page 1		
Page 2		

Test Gambar 2: Bulbasaur

Bulbasaur	Color	Texture
Page 1		

Page 2



## 5. Analisis Solusi Algoritma

Berdasarkan ujicoba dengan menggunakan tiga dataset, waktu yang digunakan untuk mengekstrak *feature* cukup normal. Untuk dataset 1, diperlukan 307.439 detik untuk memproses 4,738 gambar. Untuk dataset 2, diperlukan 808.611 detik untuk memproses 15,776 gambar. Untuk dataset 3, diperlukan 36.576 detik untuk memproses 819 gambar. Hal ini berarti pada dataset 1, dataset 2, dan dataset 3, masing masing memiliki kecepatan proses 15 gambar/detik, 19 gambar/detik, dan 22 gambar/detik. Sedangkan untuk pencarian, hasil dapat bervariasi dari 0.5 - 2 detik pencarian gambar. Variasi ini disebabkan karena pada pemrosesan pertama, program baru memuat *cache* dataset yang telah dibuat sehingga diperlukan waktu tambahan untuk memuat *cache* tersebut. Selain itu, *library* Numba juga digunakan sehingga ketika program baru pertama kali berjalan, Numba akan meng-*compile* fungsi fungsi terlebih dahulu sehingga menambah waktu pencarian.

Dari segi kemiripan warna, pada dataset 1, gambar yang dihasilkan menunjukkan kemiripan secara visual, meskipun tidak semua gambar menggambarkan satu jenis hewan. Sebagai contoh, terdapat kasus di mana muncul gambar macan saat pengujian untuk singa. Kemungkinan perbedaan ini dapat diatribusikan pada dominasi warna latar belakang pada gambar uji yang digunakan.

Pada dataset 2, hasilnya memuaskan. Uji 1 menghasilkan karakter yang menunjukkan kemiripan, uji 2 menunjukkan kemiripan dalam hal warna, uji 3 menghasilkan banyak karakter yang serupa, dan uji 4 menghasilkan karakter yang menyerupai singa. Keberhasilan ini mungkin disebabkan oleh jumlah dataset yang melebihi dan variasi karakter anime yang umumnya memiliki beragam warna rambut.

Terakhir, pada dataset 3, tidak ditemukan hasil yang sesuai dengan ekspektasi. Hal ini dapat dijelaskan dengan jumlah dataset yang terbatas, sehingga menghambat kemampuan model untuk menghasilkan hasil yang diinginkan.

Namun, pada kemiripan tekstur, tidak didapatkan hasil yang memuaskan. Hal tersebut disebabkan oleh algoritma pencarian tekstur yang kurang lengkap

## BAB 5

### Kesimpulan, Saran, dan Komentar

Berdasarkan hasil uji coba yang telah dilakukan, didapatkan kesimpulan bahwa CBIR dengan parameter warna lebih baik daripada CBIR dengan parameter tekstur sampai batas tertentu. Hal tersebut dikarenakan CBIR dengan parameter warna akan membandingkan gambar berdasarkan warnanya sehingga dua gambar yang memiliki komposisi warna yang serupa akan memiliki tingkat kemiripan yang tinggi. Berbeda dengan CBIR berparameter tekstur yang menggunakan fitur-fitur pada gambar. Dua gambar yang berisi objek dan warna yang sama dapat memiliki komposisi fitur yang berbeda sehingga ketika dilakukan perbandingan tekstur, tingkat kemiripannya sangat kecil.

Dalam pengerjaan ini, terdapat saran yang mungkin dapat menjadi masukan bagi kami sendiri ataupun bagi tim asisten dalam tugas besar ini. Dalam pengerjaan tugas besar ini, ditemukan kendala yang cukup berat yaitu terbatasnya pustaka yang dapat digunakan. Informasi mengenai pencarian dengan tekstur tidak terlalu banyak tersedia di internet dan sebagian besar jurnal yang ada merupakan konten berbayar. Selain itu, landasan teori dari spesifikasi dirasa masih belum lengkap dan ada bagian yang belum masuk dari referensi sehingga terdapat hasil yang tidak akurat.

Pada tugas besar kali ini, kami merasa bahwa hubungan antara materi di kelas dengan tugas ini sangat kecil, khususnya materi pasca-UTS. Karena materi dan tugas kali ini kurang berhubungan, kami harus mempelajari lagi mengenai pemrosesan gambar dan *website development*. Hal tersebut cukup menyulitkan karena banyaknya tugas lain dan juga pustaka yang sedikit. Oleh karena itu, kami berharap tugas-tugas besar selanjutnya dapat lebih berhubungan dengan materi di kelas.

Tugas besar kali ini memberikan kami pengetahuan baru terkait pengembangan website, pemrosesan gambar, serta pemanfaatan matriks dan aljabar linear dalam perkembangan teknologi. Selain itu, kami menjadi lebih tahu mengenai cara untuk membandingkan kemiripan dua buah gambar. Pada awalnya, kami berpikir bahwa perbandingan kemiripan gambar sangat sulit dilakukan. Namun, dengan tugas kali ini, kami sadar bahwa hal tersebut tidak terlalu sulit.

Setelah mengerjakan tugas ini, kami sadar bahwa program yang kami buat masih kurang cepat. Untuk meningkatkan hal tersebut, ada beberapa hal yang dapat dilakukan, seperti mengubah bahasa yang digunakan (tidak menggunakan python) dan mengganti algoritma yang digunakan. Selain itu, untuk mendapatkan hasil yang lebih maksimal, kami dapat memperbesar matriks-matriks yang digunakan dalam pemrosesan sehingga didapatkan hasil yang lebih akurat. Kemudian, dalam pengembangan website, seharusnya kami menggunakan *framework* lain yang lebih umum digunakan, seperti React.

## Daftar Pustaka

- Yue, J., Li, Z., Liu, L., & Fu, Z. (2010). "[Content-based Image Retrieval Using Color and Texture Fused Features.](#)"
- Yunus, M. (2023). "[Feature Extraction: Gray Level Co-Occurrence Matrix \(GLCM\).](#)"
- Bajaj, A. (2020). "[CBIR\\_Dataset.](#)" Kaggle.
- Scribbless. (2020). "[another anime face dataset.](#)" Kaggle.
- KVPRATAMA. (2020). "[Pokemon Images Dataset.](#)" Kaggle.
- Pal, S. (2022). "[Merging TailwindCSS into Flask apps.](#)" GeekPython.
- Twarog, A. "[Tailwind CSS Toggle.](#)" Flowbite.
- Pal, S. (2022). "[Upload and Display Images On the Frontend Using Python Flask.](#)" GeekPython.
- Coding Studio Team. (2021). "[Web Development: Definisi, Jenis, hingga Tahapan Kerja.](#)" CodingStudio.
- Azhar, N. (2018). "[Apa itu Tech Stack & kenapa Kita Membutuhkannya?](#)" IDS.

Link Repository GitHub :

<https://github.com/Intermaze/Algeo02-22065>