

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

Tugas Besar 2 IF 2123 Aljabar Linier dan Geometri
Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar
Semester I Tahun 2023/2024



Disusun oleh:
Kelompok AldyDPP Reborn

Renaldy Arief Susanto	(13522022)
Nyoman Ganadipa Narayana	(13522066)
Rayhan Fadhlan Azka	(13522095)

DAFTAR ISI

BAB I.....	3
DESKRIPSI MASALAH.....	3
BAB II.....	4
LANDASAN TEORI.....	4
A. Dasar Teori Secara Umum	4
B. Dasar Pengembangan Website	6
BAB III	8
ANALISIS PEMECAHAN MASALAH.....	8
A. Langkah-Langkah Penyelesaian Masalah	8
B. Pemetaan Masalah Menjadi Elemen-Element pada Aljabar Linier.....	8
C. Contoh Ilustrasi Kasus Uji dan Penyelesaiannya	9
BAB IV	12
IMPLEMENTASI DAN UJI COBA.....	12
A. Implementasi Program Utama	12
B. Struktur Program	18
C. Tata Cara Penggunaan Program	19
D. Hasil Pengujian	20
E. Analisis Desain Solusi Pencarian	24
BAB V	26
PENUTUP.....	26
A. KESIMPULAN	26
B. SARAN	26
C. KOMENTAR ATAU TANGGAPAN	27
D. REFLEKSI	27
E. RUANG PERBAIKAN DAN PEGEMBANGAN	27
BAB VI	28
PEMBAGIAN KERJA	28
DAFTAR PUSTAKA.....	29

BAB I

DESKRIPSI MASALAH

Dalam konteks era digital saat ini, kita menyaksikan pertumbuhan eksponensial dalam jumlah gambar yang dihasilkan dan disimpan di berbagai sektor. Fenomena ini merentang lintas berbagai domain, mulai dari foto pribadi yang diunggah ke media sosial, gambar medis yang digunakan dalam diagnostik kesehatan, ilustrasi ilmiah yang penting dalam penelitian, hingga gambar-gambar komersial yang digunakan dalam pemasaran dan penjualan. Peningkatan dramatis dalam volume dan keragaman gambar ini menimbulkan tantangan signifikan dalam hal penyimpanan, pengelolaan, dan pemulihan data visual. Dengan terus bertambahnya jumlah gambar, menjadi suatu kebutuhan mendesak untuk mengembangkan sistem yang mampu mengelola dan mengambil gambar ini secara efisien dan akurat.

Sebagai respons terhadap tantangan ini, sistem temu balik gambar (image retrieval system) muncul sebagai solusi kunci. Sistem ini tidak hanya memfasilitasi pencarian dan akses gambar dari koleksi besar, tetapi juga memberikan kemampuan untuk mengelola dan mengorganisasi data visual ini dengan cara yang intuitif dan terstruktur. Sistem temu balik gambar memungkinkan pengguna untuk menjelajahi dan mengakses informasi visual yang tersimpan di berbagai platform dengan mudah. Dari pencarian gambar pribadi untuk keperluan hiburan hingga analisis gambar medis untuk keperluan diagnostik, serta dari pencarian ilustrasi ilmiah yang mendukung inovasi penelitian hingga identifikasi produk dalam konteks komersial, sistem ini membuka jalan bagi akses yang lebih luas dan lebih mendalam terhadap sumber daya visual yang semakin berkembang. Dengan demikian, pengembangan dan peningkatan sistem temu balik gambar menjadi esensial dalam mengatasi kompleksitas yang dihadirkan oleh era digital ini.

BAB II

LANDASAN TEORI

A. Dasar Teori Secara Umum

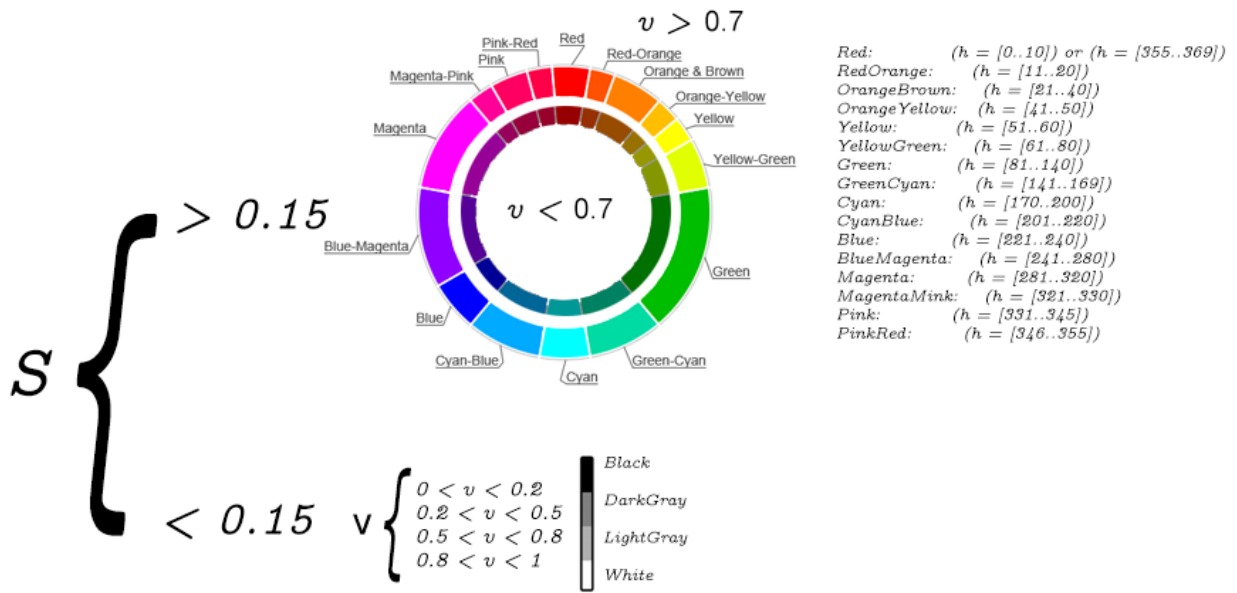
Dalam pembuatan sebuah sistem *Image Retrieval*, secara garis besar dibutuhkan dua hal: metode ekstraksi fitur/aspek dari sebuah citra (*Image*), serta metode perbandingan fitur/aspek tersebut. Dua hal ini bisa didapatkan dengan mendefinisikan fitur sebuah gambar sebagai sebuah vektor dalam suatu ruang sehingga dapat digunakan rumus *cosine similarity* sebagai metode perbandingan fitur. Rumus *cosine similarity* didefinisikan sebagai berikut.

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

Kemudian dipilih dua metode ekstraksi fitur/aspek untuk diimplementasikan (secara terpisah), yaitu metode warna dan metode tekstur. Masing-masing dirincikan sebagai berikut.

Metode warna sesuai namanya menarik kemiripan dua citra berdasarkan kecocokan warnanya. Metode ekstraksi warna dilakukan menggunakan pembangunan histogram warna dengan 36 *bins*, yang terdiri dari 16 warna gelap, 16 warna terang, serta 4 warna monokrom. Histogram warna ini dibangun berdasarkan warna setiap piksel dari citra yang diolah, dengan nilai RGB piksel sebagai data mentah yang akan diproses. Nilai RGB pertama diubah bentuknya menjadi dalam bentuk HSV *color space*, dan kuantisasi warna dilakukan berdasarkan nilai HSV yang didapatkan dengan ketentuan berikut.

1. Apabila nilai *S* atau saturasi lebih kecil dari 0.15, warna dikategorikan sebagai monokrom atau *grayscale*. Jika tidak, warna dikategorikan sebagai warna biasa.
2. Apabila warna adalah *grayscale*, warna digolongkan sebagai hitam, abu gelap, abu terang, atau putih berdasarkan nilai *V*-nya.
3. Apabila warna adalah warna biasa, warna dibedakan menjadi 16 warna biasa berdasarkan nilai *H*-nya, kemudian digolongkan lebih lanjut menjadi warna gelap atau terang berdasarkan nilai *V*-nya.



Metode ini adalah hasil simplifikasi gabungan metode dari tiga sumber [1], [2], dan [3] yang sudah dicantumkan pada daftar pustaka. Adapun metode ini dipilih karena setelah mencoba beberapa metode dari sumber lain, kami rasa hasilnya memberikan hasil yang cukup akurat untuk jumlah *bins* histogram yang cukup kecil. Gambar di atas adalah hasil pengeditan kami dan diadaptasi dari sumber [1].

Aspek tekstur dijelaskan oleh Haralick et al. dalam jurnalnya “Textural Features for Image Classification”. Tekstur didefinisikan sebagai sesuatu yang “mengandung informasi penting tentang susunan struktural permukaan dan hubungannya dengan lingkungan sekitarnya”. Intinya, tekstur adalah sesuatu yang mendeskripsikan ‘bentuk’ sebuah benda karena adanya pertimbangan. Untuk tugas besar ini, tiga fitur tekstur akan diekstrak dari citra yang akan diolah, yaitu *contrast*, *homogeneity*, dan *entropy*. Fitur-fitur ini dapat diperoleh dari sebuah gambar dengan cara membangun *GLCM* terlebih dahulu, yaitu *Gray Level Co-occurrence Matrix*. Adapun prosedur menentukan similaritas tekstur dari dua gambar adalah sebagai berikut.

1. Tiap-tiap piksel dari gambar dikuantifikasi menjadi *grayscale* dengan rumus:

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

2. Bangun *GLCM-matrix* dengan rumus:

$$C_{\Delta x, \Delta y}(i, j) = \sum_{p=1}^n \sum_{q=1}^m \begin{cases} 1, & \text{jika } I(p, q) = i \text{ dan } I(p + \Delta x, q + \Delta y) = j \\ 0, & \text{jika lainnya} \end{cases}$$

Keterangan: i atau $I(p, q)$ adalah *grayscale value* pada piksel pada koordinat (p, q) dan Δx dan Δy adalah *offset* nilai p dan q yang dipilih. Pada tugas besar ini, dipilih nilai Δx dan Δy secara berurutan adalah -1 dan -1 sehingga sudut *offset* adalah 135° .

3. Ekstrak tiga komponen (*contrast*, *homogeneity*, dan *entropy*) tekstur gambar dari *GLCM*. 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 :

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

4. Kemudian dibuat vektor dengan tiga komponen yang akan digunakan untuk membandingkan kemiripan teksturnya dengan rumus *cosine similarity*.

B. Dasar Pengembangan Website

Website pada umumnya terdiri dari dua bagian utama, yaitu front end dan back end. Front end atau client side adalah bagian dari website yang berinteraksi dengan pengguna. Front end mencakup estetika dan *design* dari sebuah website atau User Interface, dan pengalaman pengguna selama menggunakan dan berinteraksi dengan website atau User Experience. Front end pada umumnya dikembangkan dengan menggunakan Bahasa pemrograman seperti HTML, CSS, Javascript dan masih banyak yang lainnya.

Sementara itu, back end berada di belakang layer yang berguna dalam memproses logika program, manajemen database, dan banyak yang lainnya. Bahasa pemrograman yang

umum digunakan dalam pengembangan back end antara lain PHP, Ruby, Node.js dan banyak yang lainnya.

Kedua bagian ini, front end dan back end, dapat terkoneksi dan bekerja bersama dengan adanya API (Application Programming Interface) yang memungkinkan terjadi pertukaran antarkedua bagian. Dalam tugas besar kali ini, kami menggunakan Next.js sebagai front end dan Django sebagai back end dari website tersebut.

BAB III

ANALISIS PEMECAHAN MASALAH

A. Langkah-Langkah Penyelesaian Masalah

1. Pengumpulan data. Mengumpulkan kumpulan data gambar yang akan digunakan sebagai dataset untuk CBIR. Dataset ini harus mencakup berbagai jenis gambar dengan variasi warna dan tekstur yang representatif.
2. Pemodelan dan representasi data. Merepresentasikan data ekstraksi fitur ke dalam bentuk yang sesuai untuk proses pencarian.
3. Ekstraksi fitur. Melakukan ekstraksi fitur dari setiap gambar dalam dataset.
4. Pencocokan dan pengindeksan. Menerapkan algoritma pencocokan atau pengindeksan untuk mencocokkan permintaan pencarian (query) dengan gambar dalam dataset berdasarkan kesamaan fitur warna dan tekstur.
5. Validasi dan evaluasi. Melakukan evaluasi performa dari sistem CBIR yang dibangun. Ini bisa meliputi penggunaan dataset pengujian terpisah untuk mengevaluasi keakuratan dan efisiensi dari sistem dalam mengambil gambar yang sesuai dengan permintaan pencarian.
6. Peningkatan *performance* atau kecepatan pemrosesan program. Berdasarkan hasil evaluasi, melakukan penyesuaian dan perbaikan pada sistem CBIR.
7. Implementasi dan Penggunaan. Setelah pengujian dan peningkatan, mengimplementasikan sistem CBIR yang telah dikembangkan untuk penggunaan praktis.

B. Pemetaan Masalah Menjadi Elemen-Elemen pada Aljabar Linier

1. Pemodelan dan representasi data: konsep yang digunakan meliputi transformasi linier dan teori ruang vektor. Informasi fitur gambar akan direpresentasikan sebagai vektor. Representasi fitur telah dijelaskan secara detail secara bagian II.
2. Ekstraksi fitur: konsep yang digunakan meliputi operasi matriks dan transformasi linier. Operasi matriks yang digunakan dapat dilihat pada bagian II.
3. Pencocokan dan pengindeksan: Digunakan konsep operasi vektor untuk mencari kemiripan dua vektor dan alhasil kemiripan dua gambar.

C. Contoh Ilustrasi Kasus Uji dan Penyelesaiannya

Diberikan dua gambar sebagai berikut. Akan dihitung persentase kemiripannya menggunakan metode penyelesaian masalah yang sudah disebutkan di atas dan pada bab II. Metode yang digunakan pada kasus uji ini adalah metode berbasis warna.



Gambar 1: Arisa1.png

Gambar 2: Arisa2.png

Pertama, kedua gambar dibagi masing-masing menjadi blok 3x3 (tiap blok ukurannya tidak sama) dengan skema seperti berikut.



Kemudian, untuk kedua gambar dihitung vektor *color histogram* untuk masing-masing blok. Untuk contoh ilustrasi kasus uji ini, hanya akan diambil blok kiri atas sebagai contoh. Berikut hasil ekstraksi vektor warna untuk blok kiri atas dari kedua gambar.

Vektor telah dinormalisasi.

$$\begin{aligned} V_1 = & 0.0264x_1 + 0.0429x_2 + 0.0441x_3 + 0.0356x_4 + 0.0314x_5 + 0.0142x_6 + 0.0000x_7 + \\ & 0.0014x_8 + 0.0050x_9 + 0.0178x_{10} + 0.1935x_{11} + 0.4008x_{12} + 0.2848x_{13} + \\ & 0.0785x_{14} + 0.0241x_{15} + 0.0182x_{16} + 0.0166x_{17} + 0.0265x_{18} + 0.0243x_{19} + \\ & 0.0178x_{20} + 0.0157x_{21} + 0.0071x_{22} + 0.0000x_{23} + 0.0007x_{24} + 0.0025x_{25} + \\ & 0.0199x_{26} + 0.3183x_{27} + 0.6412x_{28} + 0.4042x_{29} + 0.0727x_{30} + 0.0143x_{31} + \\ & 0.0102x_{32} + 0.0000x_{33} + 0.0279x_{34} + 0.0558x_{35} + 0.1361x_{36} \end{aligned}$$

$$V_2 = 0.0264x_1 + 0.0429x_2 + 0.0441x_3 + 0.0356x_4 + 0.0314x_5 + 0.0142x_6 + 0.0000x_7 + 0.0014x_8 + 0.0050x_9 + 0.0178x_{10} + 0.1935x_{11} + 0.4008x_{12} + 0.2848x_{13} + 0.0785x_{14} + 0.0241x_{15} + 0.0182x_{16} + 0.0166x_{17} + 0.0265x_{18} + 0.0243x_{19} + 0.0178x_{20} + 0.0157x_{21} + 0.0071x_{22} + 0.0000x_{23} + 0.0007x_{24} + 0.0025x_{25} + 0.0199x_{26} + 0.3183x_{27} + 0.6412x_{28} + 0.4042x_{29} + 0.0727x_{30} + 0.0143x_{31} + 0.0102x_{32} + 0.0000x_{33} + 0.0279x_{34} + 0.0558x_{35} + 0.1361x_{36}$$

Setelah di dapat vektor warna kedua gambar ini, diambil hasil perkalian dotnya saja untuk mendapatkan *cosine similarity*-nya (karena vektor sudah dinormalisasi sehingga tidak perlu dibagi panjang vektor lagi). Hasil perkalian dot $V_1 \cdot V_2$ di atas adalah 0.9843 sehingga disimpulkan kemiripan blok kiri atas adalah 98.43%

Proses ini diulang untuk setiap blok, kemudian diambil *weighted average*-nya. Alasan digunakan *weighted average* atau rata-rata tertimbang adalah untuk memberikan bobot kemiripan yang lebih besar pada bagian tengah gambar. Alasan pemberian bobot kemiripan yang lebih besar pada bagian tengah gambar adalah karena objek-objek observasi dari gambar biasanya terletak pada bagian tengah gambar. Berikut skema pembobotan kemiripan blok:

1	2	1
2	9	2
1	2	1

Setelah seluruh blok kedua gambar diproses dan dibandingkan, didapatkan data kemiripan untuk masing-masing blok seperti berikut (urutan penomoran kiri ke kanan, atas ke bawah, blok 1 adalah kiri atas, blok 9 adalah kanan bawah):

Nomor blok	Kemiripan
1	0.9843436770034905
2	0.9328849009866156

3	0.22200817635196635
4	0.8754779803815108
5	0.847636417857349
6	0.7375784506556324
7	0.4599909557895019
8	0.7374963287805233
9	0.8728338345568272

Kemudian dihitung *weighted average*-nya adalah:

$$\frac{(\text{block 1} + \text{block 3} + \text{block 7} + \text{block 9}) * 1 + (\text{block 2} + \text{block 4} + \text{block 6} + \text{block 8}) * 2 + \text{block 5} * 9}{21}$$

$$= 0.7826 = \mathbf{78.26 \%}$$

Dalam kata lain, kesimpulannya adalah kemiripan arisa1.png dan arisa2.png adalah **78.26%**

BAB IV

IMPLEMENTASI DAN UJI COBA

A. Implementasi Program Utama

Tech Stack

Program system temu balik gambar ini diimplementasikan dengan menggunakan frontend Next js, dan backend dengan Python Django. Bagian frontend adalah bagian yang dapat dilihat pengguna, sedangkan bagian backend digunakan sebagai bagian dari pencocokan gambar query dengan dataset. Program menerima input image query dan dataset yang nantinya akan dicocokkan image query tersebut dan image pada dataset, lalu akan ditampilkan kembali image dengan kecocokan yang lebih dari 60%. Pada program ini library yang digunakan dalam pemrosesan gambar diantaranya adalah :

- os : menempatkan file dataset dan query pada folder yang telah ditentukan.
- PIL : library untuk membantu konversi image ke array.
- Time : digunakan untuk menghitung waktu yang dibutuhkan selama pemrosesan gambar

Algoritma CBIR

1. CBIR Color

Pada bagian CBIR color, ide utama algoritma ini adalah membagi gambar menjadi blok 3x3 yang nantinya di masing-masing blok akan diukur tingkat kesamaannya dengan image target.

Pada perhitungan masing-masing blok akan diekstrak 36 fitur warna yang elemen 1-16 berisi *light colors*, elemen ke 17-32 berisi *dark colors*, dan elemen 33 – 36 berisi *grayscale colors*.

```
def calculateBlockVector(image1: Image, start_x, start_y, end_x, end_y) -> list[float]:  
  
    pixels1 = image1.load()  
  
    # compression values; increase to increase spatial averaging -> higher performance  
    # with lower acc  
  
    # (value of 1 means images are not compressed at all)  
  
    compression_x = math.ceil(image1.width/125)  
  
    compression_y = math.ceil(image1.height/125)
```

```

# Calculate HSV Histogram / freq table for image 1

LightColors = [0 for i in range(16)]

DarkColors = [0 for i in range(16)]

GreyScaleColors = [0 for i in range(4)]

```

Pada bagian ini, kita melakukan load pada image dan menetapkan seberapa banyak kita melakukan *spatial averaging* (kompresi agar pemrosesan image lebih cepat) dalam hal ini, kita memilih angka 125, agar setiap gambar yang diproses, diproses secara efektif seperti gambar yang berukuran 125x125, angka ini digunakan agar pemrosesan gambar dapat dijalankan dengan lebih cepat, serta gambar yang berukuran 125 x 125 dan kami mengamati jika menggunakan angka ini, diperoleh kestimbangan antara kecepatan pemrosesan dan keakuratan.

```

for y in range(start_y, end_y - compression_y, compression_y):

    for x in range(start_x, end_x - compression_x, compression_x):

        # Access pixel color and allocate to {r,g,b}

        r, g, b = piksels1[x, y]

        h, s, v = RGBtoHSV(r, g, b)

        # Get color code for that particular RGB

        idx = HSVToIndex(int(h), s, v)

```

Pada bagian ini, kami akan mengekstrak fitur-fitur RGB pada piksel-piksel dan mengonversi RGB menjadi HSV. Lalu kami mengklasifikasikan warna dari bin histogram dengan nilai HSV yang telah diperoleh tadi.

```

if s < 0.15:                                     # Increment frequency of that color by +1

    GreyScaleColors[idx] += 1

    if idx == 1 or idx == 2:

        GreyScaleColors[idx-1] += 0.19628

        GreyScaleColors[idx+1] += 0.19628

    else:

        if v > 0.7:

            LightColors[idx] += 1

```

```

        LightColors[(idx - 1) % 16] += 0.499

        LightColors[(idx + 1) % 16] += 0.499

        DarkColors[idx] += 0.5

        DarkColors[(idx - 1) % 16] += 0.2499

        DarkColors[(idx + 1) % 16] += 0.2499

    else:

        DarkColors[idx] += 1

        DarkColors[(idx - 1) % 16] += 0.499

        DarkColors[(idx + 1) % 16] += 0.499

        LightColors[idx] += 0.5

        LightColors[(idx - 1) % 16] += 0.2499

        LightColors[(idx + 1) % 16] += 0.2499

```

Lalu kami melakukan *weighted incrementation* agar warna yang terlihat mirip/dekat oleh penglihatan manusia, dapat diproses dengan mirip menurut persepsi komputer, hal ini bersesuaian dengan kalimat pada buku Johannes Itten yang berjudul "The Elements of Color".

```

color_vector = []

for i in range(16):
    color_vector.append(LightColors[i])

for i in range(16):
    color_vector.append(DarkColors[i])

for i in range(4):
    color_vector.append(GreyScaleColors[i])

# Normalize vector

color_vector = normalize(color_vector)

```

```
return color_vector
```

Setelah itu, baru kita dapat memasukkan masing-masing fitur warna kedalam color vector.

Lalu, kita dapat mencari tingkat kemiripan dari dua buah *image* dengan cara,

```
for i in range(9):

    color_vector1 = color_vectors1[f"array_{i+1}"]

    color_vector2 = color_vectors2[f"array_{i+1}"]

    block_similarity = dotProduct(color_vector1, color_vector2)

    if (i+1) in [1, 3, 7, 9]:

        similarity += block_similarity

    elif (i+1) in [2, 4, 6, 8]:

        similarity += (2 * block_similarity)

    else:

        similarity += (9 * block_similarity)

similarity /= 21

return similarity
```

Pada bagian kode ini, kita akan mencari kesamaan pada blok di dua image dengan mencari dot product dari color vector pada blok image pertama dan color vector pada image kedua. Setelah itu kita menerapkan sistem *weight* pada similarity tersebut berupa blok yang berada di Tengah memiliki bobot lebih tinggi dari blok yang berada di pojok.

2. CBIR Texture

Pada algoritma CBIR Texture, kami mencocokkan dua gambar dengan mencari hasil dot product dari komponen Contrast, Homogeneity, dan Entropy pada kedua gambar. Sebelum mencari Contrast, Homogeneity, dan Entropy (CHE), terlebih dahulu mencari Grey Level Co-occurrence Matrix (GLCM), dengan cara,

```
def CalculateGLCMMatrix(img: Image) -> list[list[int]]:

    # Declare stuff

    glcmMatrix = [[0 for i in range(256)] for j in range(256)]

    glcmMatrixTranspose = [[0 for i in range(256)] for j in range(256)]

    # Load stuff

    img = img.convert('RGB')

    width, height = img.size

    piksels = img.load()

    compression_x = 3

    compression_y = 3

    normalizing_constant = (2 / 9) * (width) * (height)

    # Process stuff

    for y in range(1, height, compression_y):

        for x in range(1, width, compression_x):

            R, G, B = piksels[x, y]

            Y1 = GrayScaleValue(R, G, B)

            R, G, B = piksels[x - 1, y - 1]
```



```

        Y2 = GrayScaleValue(R, G, B)

        glcmMatrix[Y1][Y2] += 1

        glcmMatrixTranspose[Y2][Y1] += 1

# Add glcm with glcm^T
for i in range(256):
    for j in range(256):
        glcmMatrix[i][j] += glcmMatrixTranspose[i][j]

        glcmMatrix[i][j] /= normalizing_constant

return glcmMatrix

```

Pada bagian ini, kita mencari GLCM dengan terlebih dahulu mencari nilai grayscale pada masing-masing piksel, yang dimana grayscale tersebut didapat dari konversi dari RGB dengan,

```

def GrayScaleValue(R: int, G: int, B: int) -> float:

    return int((0.299 * R + 0.587 * G + 0.114 * B))

```

lalu nilai GLCM setiap elemen akan ditambah dengan nilai GLCM transpose.

Setelah didapat GLCM, kita dapat mencari nilai contrast, homogeneity, dan entropy dari masing masing gambar.

```

def writeAndCalculateFeatures(abspath_img: str, data) -> tuple:

    img = Image.open(abspath_img)

    GLCM = CalculateGLCMMatrix(img)

    contrast = 0

    homogeneity = 0

    entropy = 0

```

```

for i in range(256):

    for j in range(256):

        contrast += (GLCM[i][j] * (i - j) * (i - j))

        homogeneity += (GLCM[i][j] / (1 + ((i - j) * (i - j))))

        if GLCM[i][j] != 0:

            entropy += (GLCM[i][j] * log10(GLCM[i][j]))

CHEVector = [contrast, homogeneity, entropy]

```

Setelah itu, dapat dicari tingkat kesamaan kedua image dengan mencari hasil dot product antara kedua CHEVector

```
result = cosineSimilarity(CHEvector1, CHEvector2)
```

result ini menyatakan tingkat kemiripan dari kedua image dengan range 0-1.

B. Struktur Program

Program ini berada di dalam *directory* /src pada *repository*. secara keseluruhan, program memiliki folder untuk *frontend*, *backend*, dan *database*.

1. Frontend Web App

Beberapa folder di dalam *directory* /src digunakan untuk membuat tampilan pada website sehingga website yang ditampilkan kepada user lebih menarik. Berikut adalah rinciannya:

i. /app

Tampilan website yang berinteraksi secara langsung dengan user dengan menggunakan library seperti tailwind CSS dan next js.

ii. /components

Berisi komponen komponen seperti button, loading image, yang membantu *readability* kode meningkat.

iii. /constants

Berisi konstanta constant yang digunakan di dalam web agar kode menjadi lebih ringkas.

iv. /public

Berisi image, ataupun komponen image lainnya yang digunakan di dalam web ini. Folder ini juga merupakan pengintegrasian antara informasi image yang diberikan dari *backend*.

v. /styles

Berisi warna, font, dan lain sebagainya yang dibutuhkan oleh web.

2. Backend Web App

Selain yang disebutkan di atas, terdapat juga folder yang berguna untuk *backend*. Berikut rinciannya:

- a. */server*
Berisi bagaimana web dapat menerima saluran dari *frontend* terhadap informasi-informasi yang diperlukannya. Di sini juga terintegrasi terhadap database yang digunakan untuk caching dan juga algoritma CBIR.
3. Algoritma *CBIR*
Algoritma CBIR terletak pada folder *src/server* yang juga terintegrasi dengan *backend* serta dengan database.
4. Lainnya
Struktur lainnya seperti folder *src/types*, dan file-file lainnya digunakan dengan alasan untuk meningkatkan kecepatan *development* seperti *tailwind.config.ts* dan *next.config.js* yang keduanya merupakan framework untuk mendukung kecepatan produksi dari web ini. Selain itu terdapat *.gitignore* untuk memaksimalkan penggunaan github.

Lebih lanjut, aplikasi website ini memiliki cara kerja yang relative sederhana: *user* memberi informasi yang dibutuhkan, kemudian *frontend* menerima dan memberikannya kepada *backend*, lalu *backend* menerimanya dan memanggil algoritma CBIR bersamaan dengan melihat isi database (*cache results*) untuk memberi informasi *search result* kepada *user* melalui *frontend*.

C. Tata Cara Penggunaan Program

Program ini memuat 3 fitur *image search*:

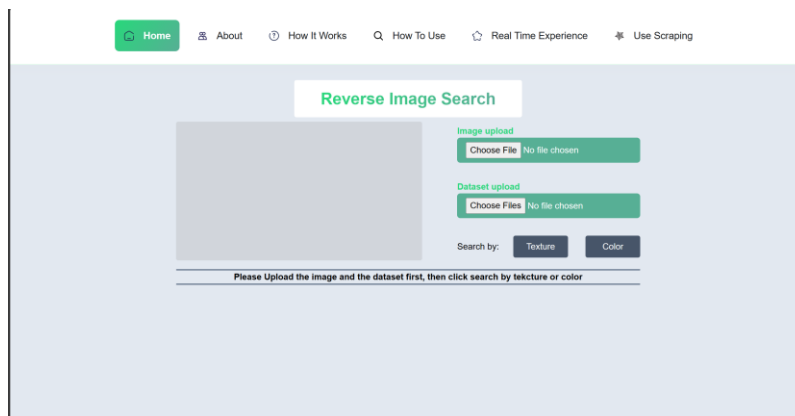
1. Fitur utama: kemiripan antara *image query* dan *dataset* yang diberikan
Fitur ini terletak pada *endpoint* */home*, untuk melakukan *searching* menggunakan fitur ini diperlukan 2 hal yang perlu diunggah oleh pengguna: *query image* dan *dataset* untuk dibandingkan. Perlu diperhatikan bahwa kecepatan pemrosesan *searching* adalah sekitar 40 gambar dalam 1 detik (sebelum *cache*), sehingga apabila mengharapkan hasil yang cepat gunakanlah *dataset* yang lebih sedikit.
2. Fitur bonus pertama: kemiripan antara apa yang dilihat kamera *device* Anda dengan *dataset* yang diberikan
Fitur ini terletak pada *endpoint* */realtime*, untuk menjalankan fitur ini, diperlukan *camera permission* dari pengguna agar aplikasi web ini dapat berjalan. Selain itu, diperlukan juga unggahan *dataset* dari pengguna sebagai acuan perbandingan antara apa yang dilihat kamera pengguna dengan *dataset*, serta tombol lock agar setelah pemrosesan selesai tidak dilakukan proses *searching* kembali.

Agar memaksimalkan waktu *realtime*, Fitur ini akan terus menerus mengulangi *search* apabila tidak di-lock, sehingga pengguna pun akan kesulitan untuk melihat hasil *similarity rate* dari hasilnya apabila belum lock.

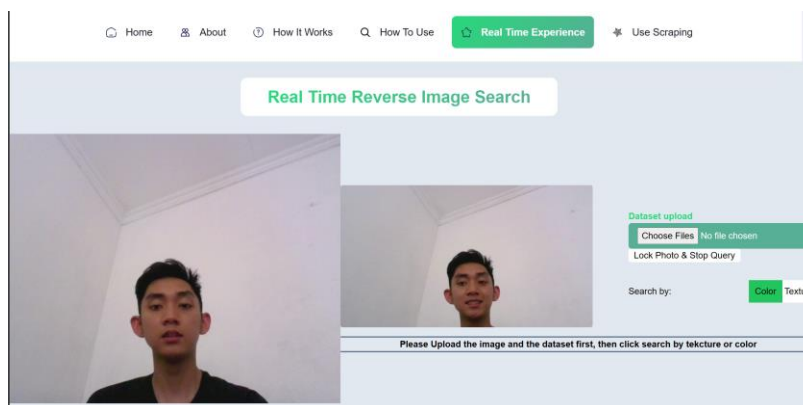
- Fitur bonus kedua: kemiripan antara *image query* yang diunggah dengan foto hasil *scraping* dari keyword yang diberikan.
Fitur ini terletak pada *endpoint* \scrape, untuk menjalankan fitur ini diperlukan satu foto sebagai foto perbandingan dan juga sebuah keyword agar dilakukan *scraping* menggunakan *search engine* Bing, hasil *scraping* Bing image kemudian dijadikan sebagai *dataset* yang kemudian dibandingkan dengan foto yang telah diunggah.

D. Hasil Pengujian

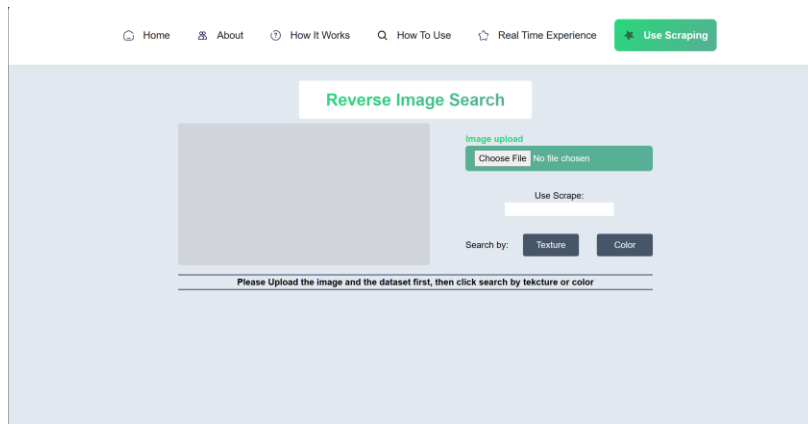
1. Tampilan antarmuka program



Gambar 4.1.1 Tampilan menu utama GUI



Gambar 4.1.2 Tampilan menu Real Time Experience

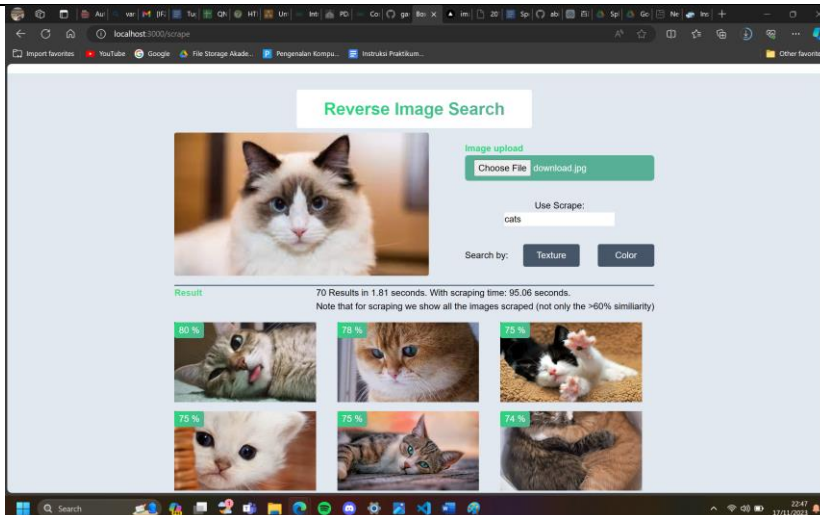


Gambar 4.1.3 Tampilan menu web scraping

2. Hasil Pengujian CBIR Color

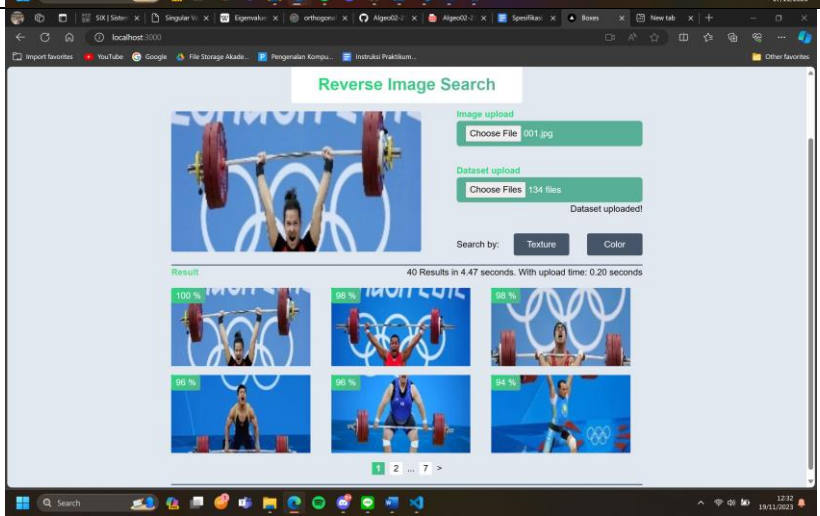
No	Hasil Pengujian	Deskripsi
1		<p>Digunakan dataset gambar pokemon sebanyak 800 gambar dengan resolusi 120 x 120 pixel, diperoleh waktu 21.81 detik untuk memproses gambar, dan 1.01 detik untuk upload ke website</p>
2		<p>Dataset pokemon yang sama seperti diatas, dan kali ini digunakan cache untuk menyimpan informasi dataset, diperoleh waktu pemrosesan yang jauh lebih rendah , yaitu 2.86 detik</p>

3



Dataset gambar kucing yang didapat dari fitur web scraping, dataset didapat dari website bing images, waktu scraping sebanyak 70 images adalah 95.06 detik dengan waktu pemrosesan 1.81 detik

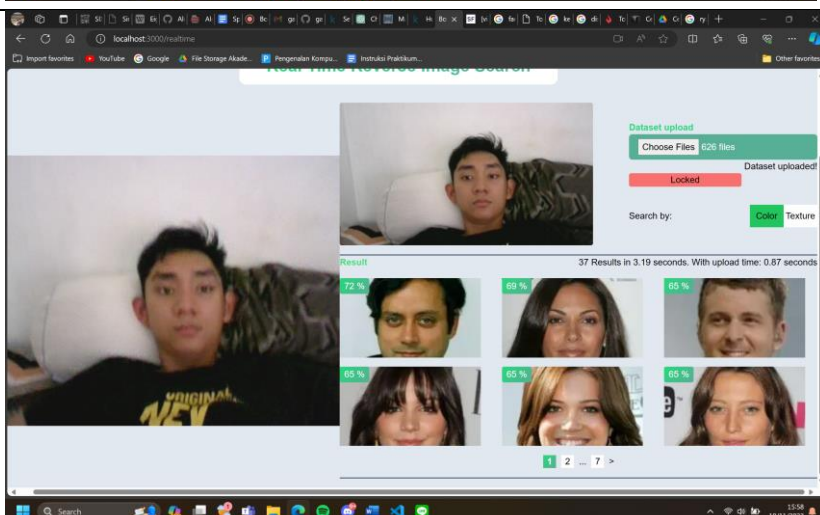
4



Ukuran gambar : 224 x 224 pixel
Banyak gambar : 134 gambar

Pada pencarian kali ini diperoleh waktu yang cepat dikarenakan ukuran gambar yang kecil dan jumlah dataset yang tidak terlalu banyak

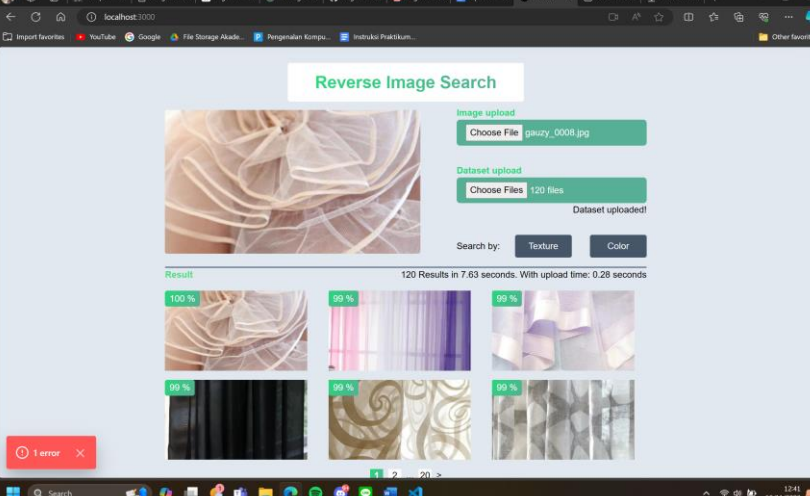
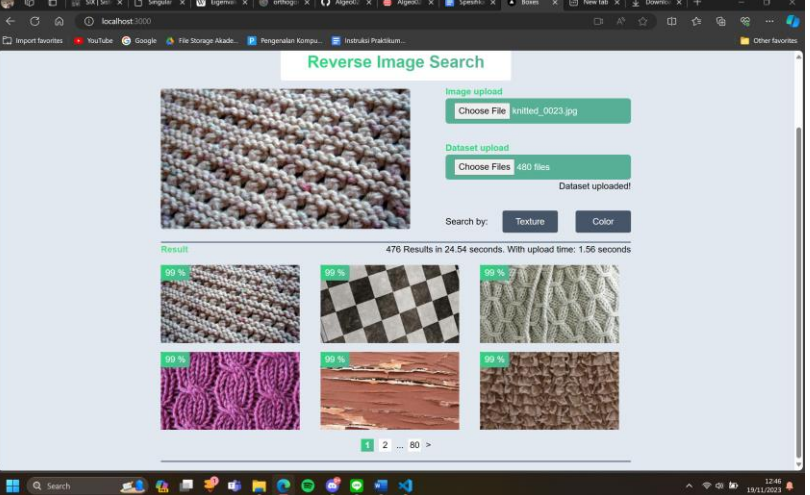
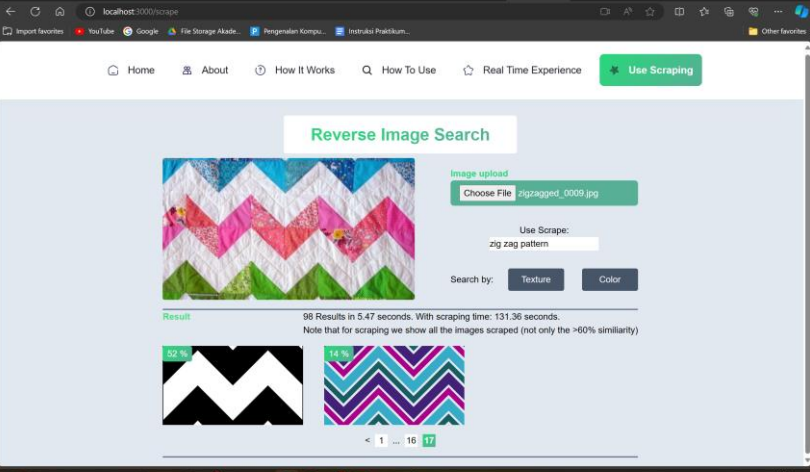
5

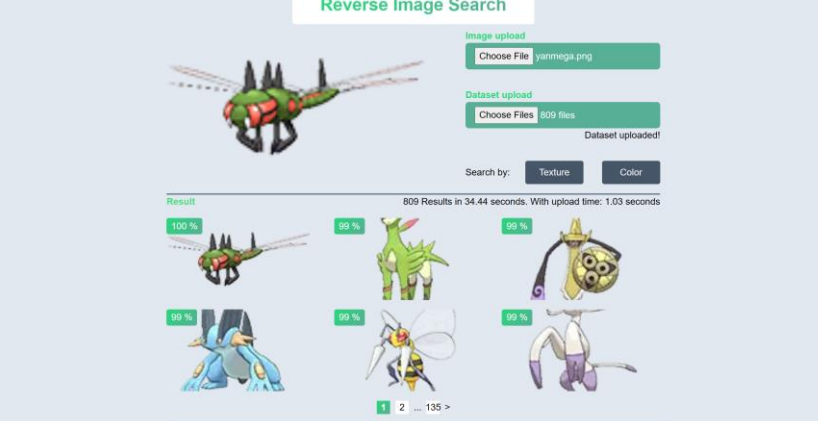
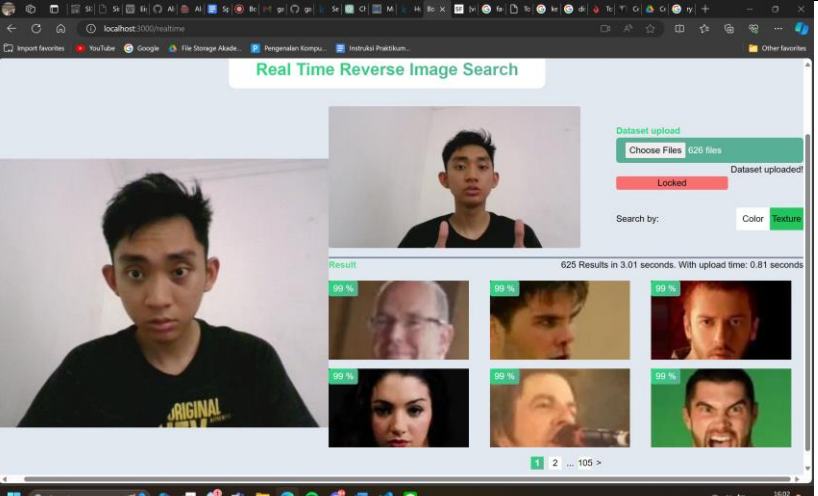


Ukuran gambar : 178 x 218 pixel
Banyak gambar : 626 gambar

Pencarian kali ini menggunakan kamera sebagai input query image, dengan caching diperoleh waktu pencarian 3.19 detik

3. Hasil pengujian CBIR Texture

No	Hasil Pengujian	Deskripsi
1		<p>Ukuran gambar : bervariasi dari 300x300 sampai 600x600</p> <p>Banyak gambar : 120 gambar</p> <p>Waktu yang dibutuhkan untuk memproses gambar adalah 7.63 detik</p>
2		<p>Ukuran gambar : bervariasi dari 300x300 sampai 600x600</p> <p>Banyak gambar : 120 gambar</p> <p>Waktu yang dibutuhkan 24.54 detik</p>
3		<p>Gambar dataset didapat dari hasil web scraping, dengan waktu scraping selama 131 detik dan waktu pemrosesan selama 5.47 detik.</p>

4		<p>Jumlah gambar : 809 gambar. Resolusi gambar : 120 x 120 Dengan dataset yang sama dengan CBIR color (1), dibutuhkan waktu yang lebih lama (34.4 detik)</p>
5		<p>Jumlah gambar : 626 gambar Resolusi gambar : 178 x 216 pixel Diperoleh waktu 3.01 detik untuk pemrosesan gambar dengan caching pada dataset</p>

E. Analisis Desain Solusi Pencarian

Saat kami menguji tingkat kemiripan beberapa gambar di dataset dengan gambar query, didapat bahwa algoritma CBIR Color dapat menampilkan tingkat kemiripan yang lebih akurat serta lebih cepat. Tingkat kecepatan ini dapat disimpulkan dari kasus 1 pada pengujian CBIR Color dan kasus 4 pada pengujian CBIR Texture, yang dimana saat digunakan dataset yang sama, tanpa caching, waktu yang dibutuhkan untuk memproses seluruh gambar pada dataset adalah 21.81 detik untuk CBIR Color, dan 34.4 detik untuk CBIR Texture.

Perbedaan kecepatan ini dapat terjadi karena pada algoritma CBIR Color, kami hanya melakukan traversal sebesar pixel image, namun pada algoritma CBIR Texture kami melakukan traversal sebesar pixel image dan traversal lagi untuk memproses GLCM Matriks.

Selain itu, terdapat faktor lain dimana saat kami menguji tingkat kemiripan menggunakan CBIR Texture, hasil yang didapat umumnya memiliki tingkat kemiripan >95% pada seluruh gambar yang ada di dataset. Menurut kami hal ini terjadi karena metode penentuan kemiripan dengan CBIR Texture tidak tepat menggunakan Cosine Similarity, berdasarkan studi pustaka yang telah kami pelajari, perlu adanya faktor-faktor lain yang dipertimbangkan, seperti simpangan baku, rata-rata, ataupun turunan berarah.

BAB V

PENUTUP

A. KESIMPULAN

Kami telah membuat program temu balik gambar berbasis web dengan ketentuan sesuai spesifikasi tugas besar ini. Ada pun penjelasan fitur yang telah diimplementasikan, yaitu sebagai berikut.

1. Dengan sebuah gambar *query*, program dapat mencari gambar yang mirip dalam suatu dataset yang diunggah oleh pengguna dengan metode kemiripan berbasis warna, kemudian menampilkan gambar mengurut dari kemiripan yang tertinggi.
2. Dengan sebuah gambar *query*, program dapat mencari gambar yang mirip dalam suatu dataset yang diunggah oleh pengguna dengan metode kemiripan berbasis tekstur, kemudian menampilkan gambar mengurut dari kemiripan yang tertinggi..
3. Program dapat menggunakan akses kamera pada gawai pengguna untuk menangkap gambar sebagai *query* dan melakukan pencarian gambar pada dataset dalam *real-time*.
4. Program dapat melakukan *caching* sehingga mempercepat pencarian gambar untuk pencarian-pencarian berikutnya pada dataset yang sama.
5. Program dapat menerima masukan string untuk melakukan *image scraping* dari hasil pencarian pada *Bing* dengan string tersebut.

B. SARAN

Saran yang mungkin dapat kami berikan untuk pengerjaan tugas besar ini adalah sebagai berikut. Untuk kedepannya, kami rasa ada baiknya jika metode algoritma yang diberikan pada spesifikasi, khususnya pada kasus ini penentuan kemiripan, perlu lebih dikaji terlebih dahulu agar mencegah adanya perubahan spek yang signifikan. Ada pun baiknya jika diberikan setidaknya satu contoh uji kasus kemiripan gambar untuk masing-masing metode. Tanpa adanya contoh tafsiran kemiripan gambar secara kasar, kami tidak dapat menentukan kebenaran dan/atau keakuratan secara detail dari algoritma yang kami bikin. Kemiripan pada dasarnya bersifat subjektif sehingga kami rasa perlu adanya semacam penetapan standar pada spesifikasi, walaupun kasar. Informalnya, perlu ada pernyataan yang bisa mengatakan “Oh, kedua gambar ini mirip, tetapi kedua gambar yang ini tidak.”

C. KOMENTAR ATAU TANGGAPAN

1. Aldy: Menambah wawasan banget sih, jadi tahu metode-metode image searching dan walaupun saya gak ikut bikin websitenya, jadi mengenal flow buulding website juga.
2. Rayhan: Lumayan enjoy mengerjakan tubesnya, dapat knowledge dan insight baru terutama dalam bidang website, karena saya sendiri belum pernah bikin website sebelumnya.
3. Gana: Menikmati prosesnya, enjoy begadang sampai malem bersama teman dekat. Pas ngerjain website juga seru sih bisa dapet insight banyak karena mau ngga mau harus keluar dari zona nyaman.

D. REFLEKSI

Selama keberjalanan tugas besar ini, ada banyak hal yang kami pelajari. Beberapa kali kami mengalami kesulitan perihal cara mempercepat performa algoritma, kemudian mempertimbangkan solusi yang tepat. Kami juga menjadi lebih memahami alur kerja sama dalam kelompok, seperti bagaimana kami membagi tugas dan saling membantu di area yang kami kurang memahami. Secara keseluruhan kami merasa tugas besar ini sudah memberikan manfaat yang sangat besar dalam berbagai aspek, dan kami cukup puas dengan hasil kerja kami. ありがとうございます。

E. RUANG PERBAIKAN DAN PEGEMBANGAN

Program dapat diperbaiki dengan metode CBIR yang lebih berbasis, khususnya CBIR tekstur yang masih sama sekali belum bisa menjadi parameter kemiripan gambar. CBIR warna pun dapat ditambahkan deteksi objek atau digunakan metode kuantisasi warna yang lebih baik agar hasil pencarian semakin akurat. Ada pun tampilan situs dapat dilengkapi berbagai fitur agar menjadi lebih *user-friendly*, contohnya penambahan animasi-animasi kecil.

BAB VI
PEMBAGIAN KERJA

No.	Deskripsi	Penanggung jawab
1	Desain Algoritma CBIR Warna	Aldy (13522022)
2	Desain Algoritma CBIR Tekstur	Aldy dan Rayhan (13522095)
3	Desasin dan Pengembangan <i>Front-End</i> Web	Gana (13522066)
4	Pengembangan <i>Back-End</i> Web	Gana dan Rayhan
5	Implementasi <i>Caching</i>	Gana dan Aldy
6	Implementasi <i>Image Scraping</i>	Rayhan
7	Implementasi Kamera <i>Real-Time</i>	Gana dan Rayhan
8	Pembuatan Laporan	Aldy, Gana, dan Rayhan
9	Pembuatan Video	Aldy, Gana, dan Rayhan

DAFTAR PUSTAKA

1. WorkWithColor.com., (2013) *The Color Wheel*. <http://www.workwithcolor.com/the-color-wheel-0666.htm>. Diakses pada November 10, 2023, pukul 20:00.
2. Sural, S., Qian, G., & Pramanik, S., (2002). *Segmentation and histogram generation using the HSV color space for image retrieval*. Proceedings of IEEE International Conference on Image Processing (pp. 589-592).
3. Qazanfari H., Hassanpour H., & Qazanfari K. (2019) *Content-Based Image Retrieval Using HSV Color Space Features*. International Journal of Computer and Information Engineering Vol:13, No:10.
4. R. M. Haralick, K. Shanmugam, & I. Dinstein. (1973). *Textural Features for Image Classification*. in IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-3, no. 6 (pp. 610-62q).
5. Yunus, M., (2020). *Feature Extraction : Gray Level Co-occurrence Matrix (GLCM)*. <https://yunusmuhammad007.medium.com>. Diakses pada November 12, 2023, pukul 21:00.

Tautan Repositroy Github: <https://github.com/ganadipa/algeo02-22022/>

Tautan Video: <https://youtu.be/pf0r96eciZ8>