

**Laporan Tugas Besar 2
IF2123 Aljabar Liniear dan Geometri
Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar
Semester I Tahun 2023/2024**



Disusun Oleh:
Muhammad Althariq Fairuz 13522027
Muhammad Fairuz Alaudin Yahya 13522057
Randy Verdian 13522067

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

BAB 1

DESKRIPSI MASALAH

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Proses ini dimulai dengan ekstraksi fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk. Setelah fitur-fitur tersebut diekstraksi, mereka diwakili dalam bentuk vektor atau deskripsi numerik yang dapat dibandingkan dengan gambar lain. Kemudian, CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan ini digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling mirip dengan gambar yang dicari. Proses CBIR membantu pengguna dalam mengakses dan mengeksplorasi koleksi gambar dengan cara yang lebih efisien, karena tidak memerlukan pencarian berdasarkan teks atau kata kunci, melainkan berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut.

1.1. Tujuan

1. Menggambarkan proses dan teknik yang digunakan dalam Content-Based Image Retrieval (CBIR).
2. Menjelaskan bagaimana CBIR dapat digunakan untuk meningkatkan efisiensi dan akurasi dalam pengenalan wajah
3. Menyajikan studi kasus atau aplikasi nyata dari CBIR dalam berbagai bidang, seperti pengenalan wajah dan pencarian gambar online.

1.2. Spesifikasi

1. Program menerima input *folder dataset* dan sebuah citra gambar.
2. Program menampilkan gambar citra gambar yang dipilih oleh pengguna.
3. Program dapat memberikan kebebasan pada pengguna untuk memilih parameter pencarian yang hendak digunakan (warna atau tekstur) melalui *toggle*. *Default* parameter yang digunakan di awal dibebaskan kepada masing-masing kelompok.
4. Program mulai melakukan perhitungan nilai kecocokan antara *image* masukan dengan *dataset image* berdasarkan parameter yang telah dipilih (warna atau tekstur).
5. Program dapat menampilkan nilai kecocokan antara *image* masukan dengan setiap gambar dalam dataset.

6. Program menampilkan hasil luaran dengan melakukan *descending sorting* berdasarkan nilai kecocokan tiap gambar. Cukup tampilkan seluruh gambar yang **memiliki tingkat kemiripan > 60%** dengan gambar masukan.
7. Program mengimplementasikan *pagination* agar jumlah gambar dapat dibatasi dengan halaman-halaman tertentu. Jumlah gambar dalam dataset yang akan digunakan oleh asisten saat penilaian mungkin berbeda dengan jumlah gambar pada dataset yang kalian gunakan, sehingga pastikan bahwa *pagination* dapat berjalan dengan baik.
8. Program dapat menampilkan **Jumlah gambar** yang memenuhi kondisi tingkat kemiripan serta **waktu eksekusi**.

BAB 2

LANDASAN TEORI

2.1. Dasar Teori

Content-Based Image Retrieval (CBIR) adalah teknologi yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya, bukan metadata atau tag teks yang terkait dengan gambar tersebut. Ada dua aspek penting dalam CBIR, yaitu ekstraksi fitur tekstur dan warna:

- a. CBIR Tekstur merujuk pada pola visual atau permukaan gambar yang memiliki sifat konsisten. Dalam CBIR, fitur tekstur diekstraksi untuk merepresentasikan pola dan variasi intensitas dalam gambar. Salah satu metode yang umum digunakan adalah matriks *co-occurrence* level abu-abu (*Gray-Level Co-occurrence Matrix*, GLCM) yang mengukur sejauh mana perubahan intensitas tertentu cenderung terjadi dalam gambar. Misalkan terdapat suatu gambar I dengan $n \times m$ piksel dan suatu parameter *offset* ($\Delta x, \Delta y$). Dengan menggunakan nilai i dan j sebagai nilai intensitas dari gambar dan p serta q sebagai posisi dari gambar, maka *offset* Δx dan Δy bergantung pada arah θ dan jarak yang digunakan. Setelah *co-occurrence matrix* didapatkan, buatlah *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Lalu, normalisasi *symmetric matrix* tersebut dengan membagi *symmetric matrix* dengan jumlahnya dan diperoleh *normalize matrix*. Kemiripan antara gambar bisa diperoleh dengan membandingkan 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity* yang diperoleh dari *normalize matrix*.
- b. CBIR Warna: Warna adalah salah satu fitur visual paling intuitif dan dominan dalam gambar. Dalam CBIR, fitur warna diekstraksi untuk merepresentasikan distribusi warna dalam gambar. Salah satu pendekatan yang umum digunakan adalah histogram warna yang menghitung jumlah piksel dalam gambar untuk setiap bin warna. 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. Setelah mendapatkan nilai HSV lakukanlah perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity*.

2.2. Website Development

Pengembangan website atau web development adalah proses pembuatan, pembangunan, dan pemeliharaan situs web. Ini mencakup berbagai tugas mulai dari pengkodean, desain teknis, hingga kinerja situs web atau aplikasi yang berjalan di internet. Terdapat tiga bagian dalam web development, yaitu UI/UX, Front-end development, dan Back-end development. UI/UX yang berfokus pada desain dan interaksi pengguna dengan situs web. Sedangkan, Front-end development bertanggung jawab untuk membangun tampilan dan interaksi langsung yang dilihat pengguna pada situs web atau aplikasi, menggunakan HTML, CSS, JavaScript, dan TypeScript. Di sisi lain, Back-end development fokus pada manajemen server, logika bisnis, dan interaksi dengan basis data, menggunakan bahasa pemrograman seperti Python, Java, atau PHP. Back-end mengelola pemrosesan data, otentikasi pengguna, dan penanganan permintaan HTTP dari Front-end, memastikan fungsi dan keamanan yang efektif di balik layar.

BAB 3

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-langkah Pemecahan Masalah

3.1.1 CBIR Warna

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

1. Normalisasi nilai RGB 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 Cmax, Cmin, dan Δ

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

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

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

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

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

4. Lakukan compare antara image dari input dengan dataset dengan menggunakan *cosine similarity*

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

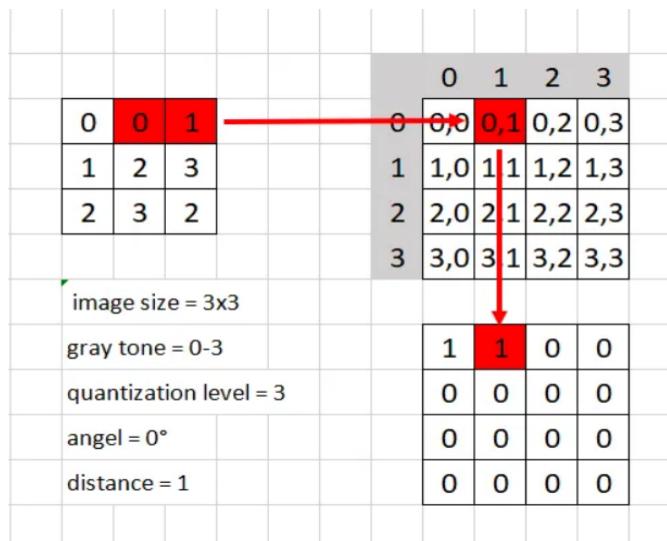
3.1.2 CBIR Tekstur

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

1. Konversi warna gambar menjadi *grayscale* karena warna tidaklah penting dalam penentuan tekstur dengan rumus:

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

2. Lakukan kuantifikasi dari nilai *grayscale* dengan ukuran 256x256 piksel.
3. Cari *Gray Level Co-Occurrence Matrix (GLCM)* dengan cara berikut:



Gambar 3.1.2.1 Cara memperoleh *co-occurrence matrix* (Sumber: [Feature Extraction : Gray Level Co-occurrence Matrix \(GLCM\) | by Muhammad Yunus | Medium](#))

4. Setelah didapat *GLCM* buatlah *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Lalu cari *matrix normalization* dengan persamaan:

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

Gambar 3.1.2.2 Cara memperoleh *matrix normalization* (Sumber: [Feature Extraction : Gray Level Co-occurrence Matrix \(GLCM\) | by Muhammad Yunus | Medium](#))

5. Dari *co-occurrence matrix* bisa diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*. Persamaan yang dapat digunakan untuk mendapatkan nilai 3 komponen tersebut antara lain :

Contrast:

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

Homogeneity :

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

Entropy :

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

6. Ukurlah kemiripan dari kedua gambar dengan menggunakan Teorema *Cosine Similarity*, yaitu:

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

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

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

3.2.1. CBIR Warna

1. Representasi Titik dalam Ruang Warna

Dalam kode, setiap warna dalam citra diwakili sebagai titik dalam ruang warna HSV. Masing-masing komponen warna (Hue, Saturation, Value) dapat dianggap sebagai koordinat dalam ruang vektor tiga dimensi.

2. Operasi aljabar pada Vektor Warna

Terdapat beberapa operasi aljabar seperti penjumlahan dan perhitungan norma vektor. Hal ini mencerminkan operasi-aljabar pada elemen-elemen vektor warna dalam ruang warna HSV.

3. Histogram sebagai Representasi Geometri

Ruang warna HSV dibagi menjadi beberapa bins untuk membuat histogram. Histogram warna global dihitung untuk setiap citra. Histogram ini dapat dianggap sebagai representasi geometri dari distribusi warna dalam citra.

4. Fungsi Similaritas (Cosine Similarity)

Metode pengukuran kesamaan antara vektor warna yang dapat dianggap sebagai operasi aljabar dalam analisis kesamaan.

3.2.2. CBIR Tekstur

1. Pengubahan citra dari RGB menjadi *Grayscale*

Setiap citra yang berwarna (RGB) akan diubah ke *grayscale* (hitam-putih) karena komponen warna tidak diperlukan pada CBIR Tekstur dengan cara mengalikan elemen setiap matriks citra dengan suatu konstanta. Hal ini dapat dianggap sebagai perkalian matriks dengan konstanta.

2. Kuantifikasi *Grayscale*

Karena citra *grayscale* berukuran 256 piksel (*range value* dari *grayscale* hanya 0-255), maka matriks yang berkoresponden akan berukuran 256×256 . Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.

3. Mencari GLCM

Setelah matrix *grayscale* dikuantifikasi, akan dicari GLCM-nya. Matriks ini digunakan karena dapat melakukan pemrosesan yang lebih mudah dan cepat. Vektor yang dihasilkan juga mempunyai ukuran yang lebih kecil.

4. Normalisasi GLCM

Lakukan normalisasi pada GLCM dengan membagi matriks GLCM asli yang sudah ditambah dengan *transpose*-nya dengan jumlahnya. Hampir semua operasi di CBIR tekstur melibatkan operasi matriks sehingga pemahaman tentang aljabar linier dan geometri sangat diperlukan.

5. Setelah *normalization matrix* diperoleh, nilai dari komponen ekstraksi tekstur bisa diperoleh. Tiga komponen tersebut adalah *contrast*, *entropy* dan *homogeneity*.

6. Fungsi Similaritas (Cosine Similarity)

Metode pengukuran kesamaan antara vektor komponen tekstur yang dapat dianggap sebagai operasi aljabar dalam analisis kesamaan.

3.3. Contoh Ilustrasi Kasus dan Penyelesaiannya

3.3.1. CBIR Warna

Ilustrasi Kasus

Ingin dibandingkan kesamaan antara Image 1 dan Image 2 dengan parameter warna.

Penyelesaian

- Setiap citra dari image diubah dari format warna RGB ke ruang warna HSV.
- Histogram warna global dihitung untuk masing-masing image.
- Histogram warna global setiap citra dapat dianggap sebagai vektor dalam ruang warna HSV. Setiap elemen vektor menggambarkan frekuensi kemunculan nilai warna dalam bins yang telah ditentukan.
- Kemiripan dari kedua image dihitung dengan *cosine similarity*.
- Hasil *cosine similarity* akan memberikan informasi tentang sejauh mana kedua citra mirip berdasarkan distribusi warna. Semakin tinggi nilai *cosine similarity*, semakin mirip kedua image tersebut.

3.3.2 CBIR Tekstur

Ilustrasi Kasus

Ingin mengambil gambar tekstur yang paling mirip dengan gambar tekstur input berdasarkan fitur tekstur.

Penyelesaian

- Setiap citra dari image diubah dari format warna RGB ke *grayscale*.
- Lakukan identifikasi pada citra yang telah diubah ke *grayscale*
- Cari GLCM dan *normalize matrix* dari citra *grayscale* yang telah dikuantifikasi.
- Setelah *normalize matrix* diperoleh, hitung nilai tiga komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*.
- Kemiripan dari kedua citra akan dihitung *cosine similarity* antara vektor komponen ekstraksi tekturnya.
- Hasil *cosine similarity* akan memberikan informasi tentang sejauh mana kedua citra mirip berdasarkan kemiripan tekturnya. Semakin tinggi nilai *cosine similarity*, semakin mirip kedua citra tersebut.

BAB 4

IMPLEMENTASI DAN UJI COBA

4.1. Implementasi program utama

4.1.1. CBIR Color

4.1.1.1. Pseudocode

Berikut adalah pseudocode dari implementasi program CBIR Color.

```
{ Convert RGB to HSV color space }
function rgb_to_hsv(r, g, b):
    r <- 255
    g <- 255
    b <- 255
    cmax <- max(r, g, b)
    cmin <- min(r, g, b)
    diff <- cmax - cmin + 0.00000001

    if cmax = cmin then
        h <- 0
    else if cmax = r then
        h <- (60 * ((g - b) / diff % 6))
    else if cmax = g then
        h <- (60 * ((b - r) / diff + 2))
    else
        h <- (60 * (((r - g) / diff) + 4))

    if cmax = 0 then
        s <- 0
    else
        s <- diff / (cmax + 0.00000001)

    v <- cmax

    -> h, s, v

{ compute histogram in HSV color space }
```

```

function
compute_global_color_histogram_hsv(image_matrix):
    image_np <- np.array(image)
    hsv      <- np.stack(rgb_to_hsv(image_np[...,     0],
image_np[..., 1], image_np[..., 2]), axis=-1)

    h_bins <- [0, 1, 25.5, 40.5, 120.5, 190.5, 270.5,
295.5, 316.5, 360]
    s_bins <- [0, 0.2, 0.7, 1]
    v_bins <- [0, 0.2, 0.7, 1]

    -> hist

{ compute cosine similarity between two vector }
function cosine_similarity(vec1, vec2):
    -> dot_product(vec1,   vec2) / (norm(vec1) *
norm(vec2))

{ compute and store histograms for all images in the
dataset }
void load_dataset_histograms():
    for each dataset_image_matrix in dataset_matrices
        dataset_histograms <-
compute_global_color_histogram_hsv(dataset_image_matrix)
        dataset_histograms.append(dataset_histogram)

{ compare with computing cosine similarity between input
histogram and each dataset histogram }
function compare_images(input_histogram):
    similarities <- []
    input_histogram <- input_histogram.flatten()
    for hist in dataset_histograms
        dataset_histograms <- hist.flatten()

    similarities.append(cosine_similarity(input_histogram,
hist)*100)

    -> similarities

```

4.1.1.2. Hasil

Berikut adalah contoh hasil perbandingan satu image query dengan 20 image dataset pada terminal.

```
Similarity with image 1: 100.00%
Similarity with image 2: 68.24%
Similarity with image 3: 49.20%
Similarity with image 4: 54.54%
Similarity with image 5: 27.67%
Similarity with image 6: 68.15%
Similarity with image 7: 60.62%
Similarity with image 8: 37.51%
Similarity with image 9: 15.28%
Similarity with image 10: 49.22%
Similarity with image 11: 62.19%
Similarity with image 12: 75.47%
Similarity with image 13: 18.07%
Similarity with image 14: 32.70%
Similarity with image 15: 32.93%
Similarity with image 16: 67.66%
Similarity with image 17: 63.19%
Similarity with image 18: 56.09%
Similarity with image 19: 80.88%
Similarity with image 20: 62.79%
```

4.1.2. CBIR Texture

4.1.2.1 Pesudocode

```
function convert_to_grayscale(img_array):
    R <- img_array[:, :, 0]
    G <- img_array[:, :, 1]
    B <- img_array[:, :, 2]
    grayscale <- 0.29 * R + 0.587 * G + 0.114 * B
    -> grayscale

function quantize_grayscale(img_array, levels=256):
    img <- convert_to_grayscale(img_array)
    img_quantized <- int((img * levels / 256))
    * 256 // levels
    -> img_quantized
```

```

function create_glc(m_img_array, distance=1):
    initialize empty matrix 256 x 256
    rows<- img_array.shape[0]
    cols <- img_array.shape[1]
    for row in range(distance, rows):
        for col in range(cols):
            i <- img_array[row, col]
            j <- img_array[row - distance, col]
            glcm[i, j] <- glcm + 1

function normalize(matrix):
    symmetric <- matrix + transpose of matrix
    -> symmetric / sum of symmetric

function calculate_contrast(matrix):
    contrast <- 0
    i traversal [0..256]
    j traversal [0..256]
        contrast      <-      contrast      +
        ((i-j)**2)*matrix[i][j]
    ->contrast

function calculate_entropy(matrix):
    entropy <- 0
    i traversal [0..256]
    j traversal [0..256]
        if ( matrix[i][j] > 0) then
            entropy      <-      entropy      +
            log(matrix[i][j])*matrix[i][j]
    -> - entropy

function calculate_homogeneity(matrix):
    homogeneity <- 0
    i traversal [0..256]
    j traversal [0..256]
        if ( matrix[i][j] > 0) then
            homogeneity <- homogeneity + matrix[i][j] / (1
                + (i - j)**2)
    -> homogeneity

```

```

function cosine_similarity(vec1, vec2):
    similarity      <- dot_product(vec1,
    vec2)/(norm(vec1) * norm(vec2))
    -> similarity

function process_image(img_array, distance=1,
levels=256):
    Convert image to grayscale and quantize using
    quantize_grayscale function
    Create GLCM using create_glcM function
    Normalize matrix using normalize function
    Calculate contrast, entropy, and homogeneity
    using calculate_contrast, calculate_entropy,
    and calculate_homogeneity function respectively
    -> contrast, entropy, homogeneity

function compare_with_dataset(query_img, dataset,
distance=1, levels=256):
    img quantized <- process_image(query_image)
    Initialize an empty list for similarities
    Loop through the dataset:
        Process each image in the dataset using
        process_image function
        Calculate the cosine similarity with the query
        image using cosine_similarity procedure
        Append the similarity to the list
    -> list of similarities

```

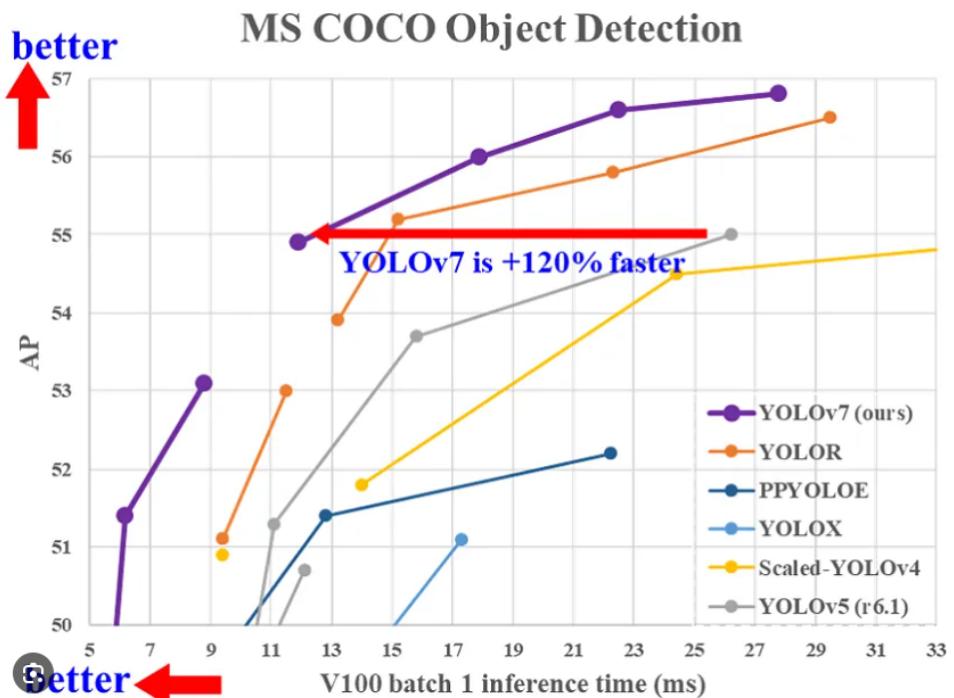
4.1.2.2 Hasil

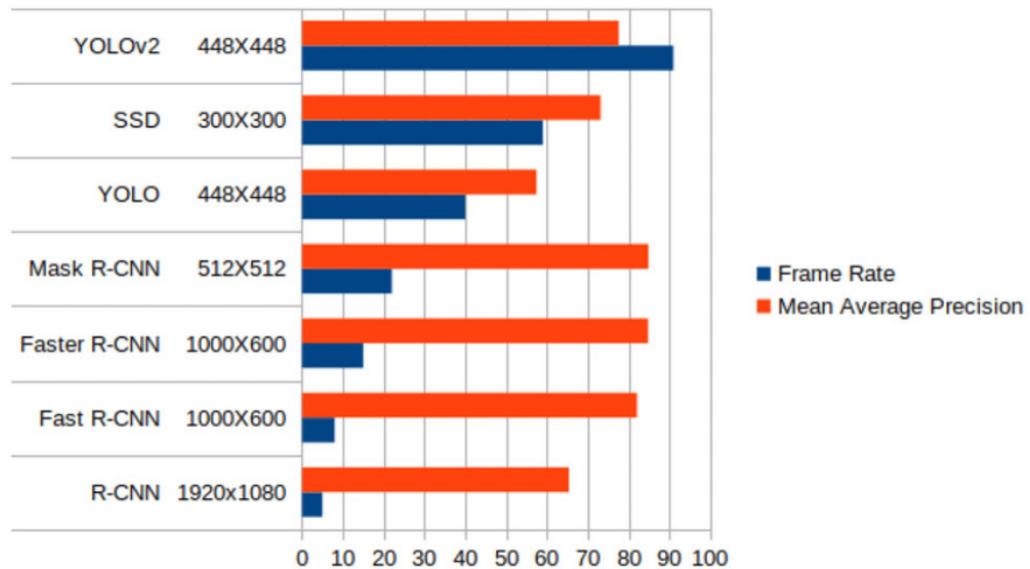
Berikut adalah hasil pengujian program :

```
Tingkat kemiripan dengan image ./image/0.jpg: 100.000
Tingkat kemiripan dengan image ./image/1.jpg: 99.929
Tingkat kemiripan dengan image ./image/10.jpg: 99.963
Tingkat kemiripan dengan image ./image/11(1).jpg: 99.967
Tingkat kemiripan dengan image ./image/2.jpg: 99.998
Tingkat kemiripan dengan image ./image/3.jpg: 99.438
Tingkat kemiripan dengan image ./image/4.jpg: 99.827
Tingkat kemiripan dengan image ./image/4558.jpg: 99.804
Tingkat kemiripan dengan image ./image/5.jpg: 99.941
Tingkat kemiripan dengan image ./image/6.jpg: 99.853
Tingkat kemiripan dengan image ./image/7.jpg: 99.867
Tingkat kemiripan dengan image ./image/8.jpg: 99.749
```

4.1.3. Object Detector

Pada pre-processing gambar, kami menambahkan crop detection dengan object detector. Kami menggunakan YOLO (You Only Look Once) sebagai object detectionnya.





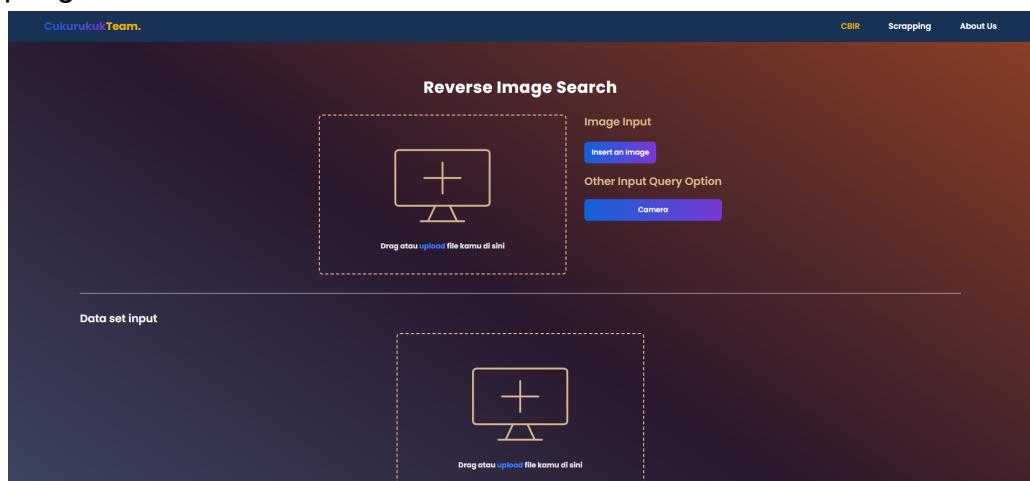
Pemilihan YOLO (You Only Look Once) sebagai metode deteksi objek dalam sistem Content-Based Image Retrieval (CBIR) didasarkan pada pertimbangan utama terkait kecepatan, efisiensi sumber daya, dan tingkat akurasi yang memadai. Versi Tiny dari YOLO dipilih karena kinerjanya yang tinggi dan bobot parameter yang lebih rendah, menjadikannya lebih ringan dan efisien dalam penggunaan sumber daya. Kecepatan eksekusi yang tinggi menjadi prioritas utama, terutama dalam konteks CBIR di mana respons cepat terhadap permintaan pencarian gambar sangat penting, terutama untuk pemrosesan gambar. Meskipun versi Tiny YOLO mengorbankan sebagian kecil akurasi, tingkat akurasi yang dihasilkan tetap cukup untuk kebanyakan aplikasi CBIR. Pilihan ini juga mendukung pemotongan otomatis gambar berdasarkan deteksi objek, meningkatkan akurasi pencarian dengan fokus pada bagian yang relevan dari gambar. Selain itu, dalam proses ini, kami mengimplementasikan threshold sebesar 0.2 untuk memilih objek dengan tingkat keyakinan (confidence) lebih tinggi dari nilai tersebut dan yang paling tinggi diantara semua objek dalam gambar. Hal ini dilakukan untuk memastikan bahwa hanya objek dengan kepercayaan yang tinggi yang dipertimbangkan dalam proses deteksi, meningkatkan keakuratan dan relevansi hasil CBIR. Oleh karena itu, YOLO versi Tiny dengan penerapan threshold menjadi pilihan optimal untuk memberikan keseimbangan antara kecepatan, efisiensi, dan akurasi, sekaligus memastikan selektivitas objek yang diambil untuk proses CBIR.

4.1.4. Website dan proses flow image processing

Kami menggunakan Next.js sebagai framework frontend yang berbasis ReactJS, Tailwind CSS untuk styling, dan Typescript untuk meningkatkan

skalabilitas kode dan melakukan komunikasi antar client dan server. Proses flow image processing pada website ini terfokus pada dua halaman utama: CBIR dan Scrapping. Pengguna dapat mengakses halaman "CBIR" untuk memproses dataset dan gambar, sementara halaman "Scrapping" digunakan untuk melakukan input URL dan melakukan web scraping.

Pada sisi server, kami menggunakan FastAPI, sebuah framework web yang cepat untuk membuat API dengan Python. Server ini memanfaatkan beberapa library penting seperti NumPy untuk operasi numerik, OpenCV untuk computer vision dan pemrosesan gambar, Requests untuk membuat permintaan HTTP, BeautifulSoup untuk web scraping, dan Unicorn sebagai server ASGI untuk menjalankan aplikasi FastAPI. Kami juga menggunakan library OS dan time untuk operasi terkait sistem dan pengelolaan waktu.



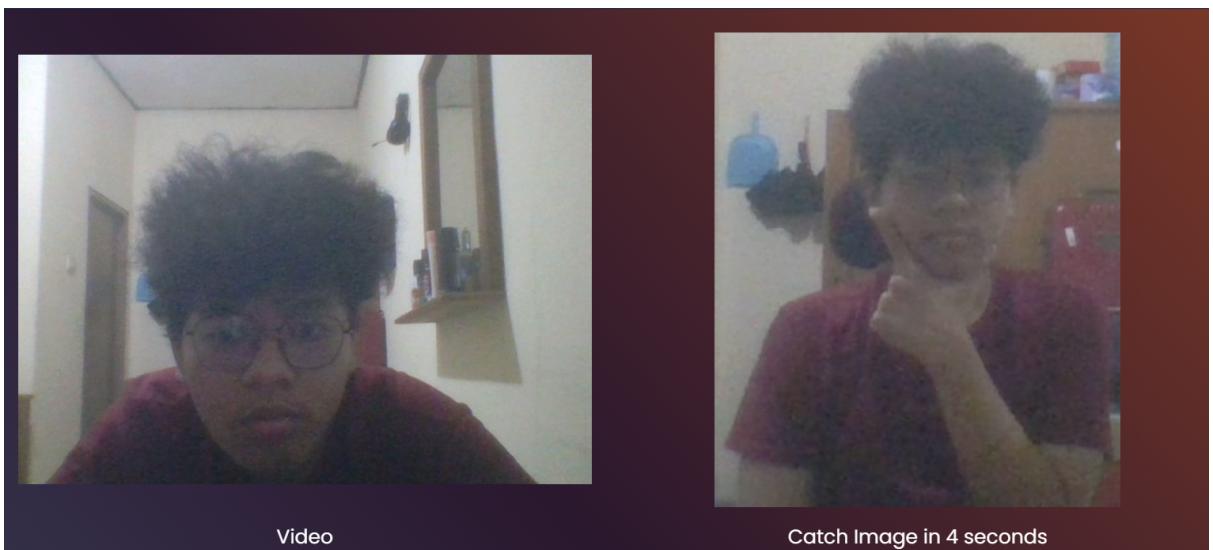
Interface pengguna yang kami rancang mencakup komponen penting untuk memudahkan interaksi pengguna. Tombol "Insert The Images" digunakan untuk mengunggah dataset melalui file input. Pengguna juga dapat memilih gambar yang ingin dicocokkan, entah dari kamera atau file input, menggunakan tombol yang sama di bagian "Insert an Image".

Pilihan pengolahan gambar berdasarkan CBIR color atau texture dapat diatur melalui toggle button. Setelah pengaturan ini dilakukan, pengguna dapat menekan tombol "Start Processing" untuk memulai proses CBIR calculation. Kami menampilkan label untuk memantau waktu eksekusi proses, memberikan informasi transparan tentang seberapa cepat hasil dapat diperoleh. Selain itu, label hasil akan menampilkan hasil analisis gambar, menyajikan persentase kemiripan dan informasi lain yang relevan.

4.1.5. Camera Input

Pengguna dapat mengambil gambar langsung dari perangkat klien menggunakan kamera. Proses pengambilan gambar ini diimplementasikan melalui tag HTML pada framework Next.js, yang memanfaatkan elemen video untuk menangkap streaming dari kamera. Data gambar yang tertangkap kemudian diproses menggunakan elemen canvas untuk diambil dalam format data base64 URL. Hasilnya, kami memperoleh data gambar yang siap untuk dikirimkan ke API untuk menerapkan Object Detector pada gambar yang diterima. Sebelumnya, kami memastikan bahwa pengguna mendapatkan pengalaman yang terencana dengan mengimplementasikan countdown setiap 10 detik pada saat pengambilan gambar. Proses deteksi objek dimulai setelah penghitungan mundur selesai, dan hasilnya, kami mendapatkan matriks objek yang mencakup informasi terperinci tentang objek yang terdeteksi.

Kami telah menempatkan suatu barier dalam proses ini. Pemrosesan CBIR hanya akan diaktifkan jika dataset yang diperlukan telah terisi. Hal ini memastikan bahwa hasil analisis dan pemilihan CBIR dilakukan secara optimal dan relevan. Seluruh proses ini, dari pengambilan gambar, deteksi objek, hingga analisis CBIR dilakukan secara otomatis. Pengguna hanya diperlukan mengganti pemrosesan CBIR dengan texture atau color saja.



4.1.6. Web Scraping

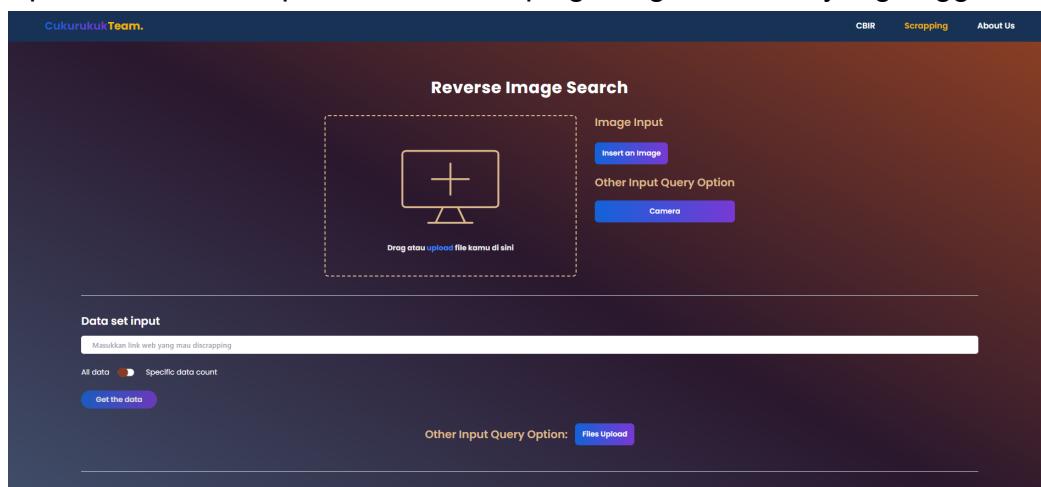
Dalam melakukan web scraping untuk mengumpulkan informasi gambar dari suatu URL, kami membuat kelas `ImageScraper` yang didukung oleh beberapa library, seperti Requests, BeautifulSoup, dan Selenium. Tujuan utama dari web scraping ini adalah untuk mengekstrak

URL, teks alternatif (alt text), dan matriks gambar dari elemen gambar pada halaman web yang dituju.

Dalam proses pengambilan halaman web, kami menggunakan Requests dan BeautifulSoup untuk melakukan HTTP GET request dan mem-parsing konten HTML halaman tersebut. Namun, ada situasi di mana elemen-elemen halaman web di-generate secara dinamis menggunakan JavaScript. Oleh karena itu, kami memanfaatkan Selenium WebDriver untuk mengatasi kasus-kasus di mana eksekusi JavaScript atau tindakan lebih lanjut diperlukan untuk memuat elemen-elemen yang diinginkan. Selenium memungkinkan kami untuk secara interaktif menjelajahi halaman web dan mengambil kontennya setelah proses eksekusi JavaScript selesai.

Setelah halaman web diakses, kami mengidentifikasi elemen gambar dengan tag "img" dan mengumpulkan informasi seperti URL absolut, teks alternatif, dan matriks gambar (diperoleh dari kelas `ImageProcessing`). Kami memeriksa ekstensi file dari URL gambar untuk memastikan bahwa hanya gambar dengan tipe file yang diizinkan (seperti 'jpeg', 'jpg', 'png', 'gif', 'bmp', 'tiff', dan 'webp') yang dapat diproses oleh library openCV.

Seluruh proses ini dilakukan untuk memberikan pengguna informasi gambar yang relevan dan sesuai dengan kebutuhan mereka. Pentingnya penggunaan Selenium WebDriver terletak pada kemampuannya untuk mengatasi kompleksitas halaman web yang menggunakan JavaScript, memastikan bahwa kami dapat mengakses semua elemen yang diperlukan untuk keperluan web scraping dengan akurasi yang tinggi.



Dalam konteks penggunaan Selenium WebDriver, perlu dicatat bahwa terdapat perbedaan antara situs web yang menggunakan JavaScript secara dinamis dan situs web yang bersifat statis. Contoh situs

web yang menunjukkan dinamis JavaScript yang tinggi adalah Airbnb (<https://www.airbnb.co.id/>). Pada situs seperti Airbnb, sebagian besar konten, termasuk elemen-elemen yang relevan untuk web scraping, mungkin di-generate secara dinamis menggunakan JavaScript saat halaman dimuat atau berinteraksi dengan pengguna.

Situs web dengan sifat statis, seperti On This Day (<https://www.onthisday.com/day/january/3>), umumnya memiliki halaman yang lebih sederhana dan elemen-elemen utama sudah terkandung dalam HTML saat halaman pertama kali dimuat. Oleh karena itu, dalam kasus seperti ini, penggunaan Selenium WebDriver mungkin tidak seefektif atau diperlukan sebanyak pada situs web yang dinamis. Kami telah menghandle kasus tersebut agar bisa dilakukan scrapping dengan lebih optimal.

4.1.7. PDF Downloader

Dalam pengembangan fitur PDF Downloader, kami memilih menggunakan FPDF sebagai library untuk menghasilkan dokumen PDF. Proses ini terjadi setelah mendapatkan data hasil pencarian dari query menggunakan metode body POST pada API FastAPI. Data ini mencakup informasi seperti gambar yang mirip, persentase kemiripan, dan detail lainnya yang relevan dengan CBIR Processing. Proses pemrosesan melibatkan pengorganisasian dan format ulang data hasil pencarian agar sesuai dengan struktur yang diinginkan dalam dokumen PDF. Selain itu, disesuaikan juga format presentasi informasi untuk memberikan hasil yang jelas dan mudah dipahami bagi pengguna.

Selanjutnya, hasil pemrosesan data tersebut diteruskan ke FPDF untuk menghasilkan dokumen PDF yang sesuai. FPDF memungkinkan pembuatan dokumen PDF secara dinamis dan dapat disesuaikan, sehingga informasi dapat disusun dengan rapi dalam dokumen tersebut.

Report Cukurukuk CBIR

Method: Texture

Image Query:



16 Results with similarities in 7.9904 seconds:



Similarity percentage: 100.0000%



4.2. Struktur program

Secara backend side, kami menggunakan fast api, dengan membuat class secara terpisah berdasarkan fungsionalitasnya, yaitu terdiri dari ImageComparator untuk CBIR color dan texture, image processing, object detector, pdf creator, dan scrapper yang semuanya dihubungkan pada index.py sebagai main fastapi. Berikut adalah struktur API yang kami gunakan

POST	/api/convert-image-to-base64	Convert Image To Base64
POST	/api/convert	Convert
POST	/api/convert-no-yolo	Convert
POST	/api/convert-multiple	Convert Multiple
POST	/api/convert-camera	Convert Camera
POST	/api/cbir-color	Compare Images
POST	/api/cbir-texture	Compare Images
GET	/api/scrape	Get Image Scrape
POST	/api/create-pdf-file	Create Pdf File

Fungsionalitas untuk api tersebut adalah:

- convert-image-to-base64: Untuk konversi hasil gambar scrapping dari url dinamis menjadi statis dalam bentuk encode base64.
- convert: Untuk pengolahan preprocessing image query file input.
- convert-no-yolo: Untuk pengolahan preprocessing image tanpa object detector pada query file input scrapping page.
- convert-multiple: Untuk pengolahan preprocessing image dataset file input.
- convert-camera: Untuk pengolahan preprocessing image query camera.
- cbir-color: Untuk pemrosesan image comparation dengan teknik CBIR berdasarkan warna.
- cbir-texture: Untuk pemrosesan image comparation dengan teknik CBIR berdasarkan tekstur.
- create-pdf-file: Untuk data processing ke dalam bentuk pdf sekaligus download file.

Dalam pengembangan frontend, kami menggunakan arsitektur Next.js dengan routing yang terorganisir dalam direktori "app". Pendekatan ini memudahkan pengembangan dan pemahaman struktur aplikasi. Penggunaan TypeScript di seluruh aplikasi memastikan keamanan dan konsistensi data antar komponen, serta antara server dan klien.

Komponen-komponen kami didesain dengan pendekatan berbasis komponen yang bersih dan reusable, dengan setiap komponen memiliki tanggung jawab spesifik. Penggunaan TypeScript memungkinkan definisi tipe data eksplisit, mengurangi kesalahan tipe, dan meningkatkan keterbacaan kode. Kami memaksimalkan reuse komponen dengan menciptakan komponen-komponen generik yang dapat digunakan di berbagai bagian aplikasi untuk memastikan bahwa perubahan pada suatu komponen tidak berdampak negatif pada fungsi-fungsi lain yang bergantung padanya.

Folder Structure:

```
+---app
|   |   favicon.ico
|   |   globals.css
|   |   layout.tsx
|   |   page.tsx
|
|   |
|   +---about-us
|       |       page.tsx
|
|   |
|   +---cbir
|       |       layout.tsx
|       |       page.tsx
```

```
|   |
|   \---scrapping
|       layout.tsx
|       page.tsx
|
+---components
|   |   button.tsx
|   |   camera.tsx
|   |   custom-link.tsx
|   |   group-pagination.tsx
|   |   iframe-yt.tsx
|   |   image-result.tsx
|   |   multiple-file-upload.tsx
|   |   navbar.tsx
|   |   pagination.tsx
|   |   scrape-pagination.tsx
|   |   scrapper.tsx
|   |   single-file-upload.tsx
|   |   switch.tsx
|   |   text-input.tsx
|
|   \
|   \---icons
|       chevron-icon.tsx
|       file-upload-empty-icon.tsx
|       menu-icon.tsx
|       x-icon.tsx
|
+---const
|   feature.ts
|   howtouse.ts
|
+---lib
|   helper.ts
|
+---public
|   Cloud-upload-alt.svg
|   white.jpg
|
\---types
    image-format.ts
```

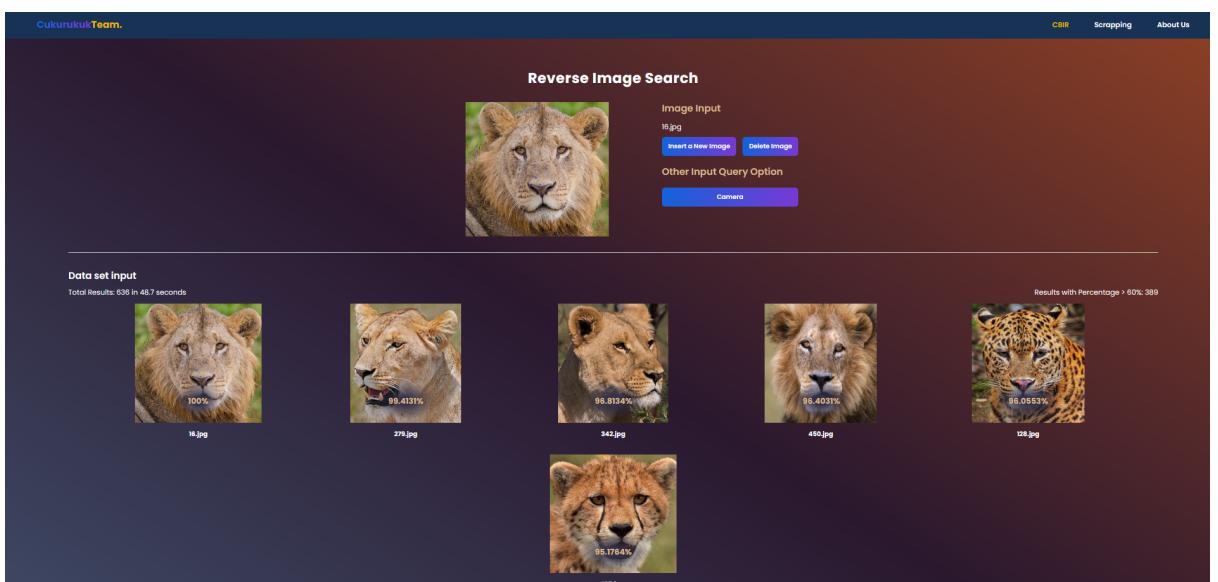
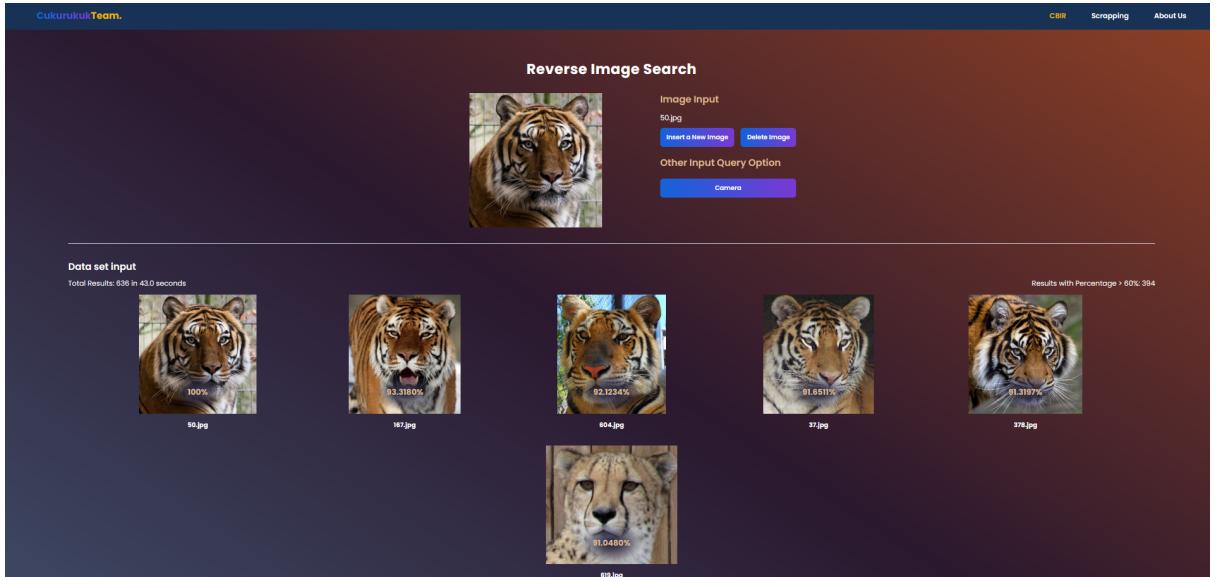
4.3. Penggunaan Program

1. Akses Halaman CBIR dan Scrapping
Buka aplikasi dan akses halaman "CBIR" untuk melakukan proses Content-Based Image Retrieval (CBIR) dengan dataset file input. Kunjungi halaman "Scrapping" untuk melakukan input URL dan melakukan web scrapping.
2. Pilihan Input Query
 - Kamera: Pilih opsi ini jika input menggunakan kamera perangkat untuk mengambil gambar sebagai query.
 - Upload File: Gunakan opsi ini untuk mengunggah file gambar dari penyimpanan perangkat sebagai query.
3. Pilih Jenis Pengolahan CBIR
Setelah memilih query, pilih jenis pengolahan CBIR yang diinginkan, apakah berdasarkan warna atau tekstur.
4. Otomatisasi Pengolahan dengan YOLO Detector
Gambar yang diunggah akan otomatis diolah menggunakan Object Detector berbasis YOLO untuk melakukan proses cropping secara otomatis.
5. Proses Pencarian dan Tampilan Hasil
Tekan tombol "Search" untuk memulai proses pencarian cosine similarities antara gambar query dan dataset. Program akan menampilkan kumpulan gambar yang mirip, diurutkan berdasarkan tingkat kemiripan. Setiap gambar akan disertai dengan persentase kemiripannya.
6. Informasi Tambahan
Informasi terkait jumlah gambar yang muncul, dan waktu eksekusi programnya akan ditampilkan.
7. Download Hasil dalam Format PDF
Setelah proses selesai, hasil analisis dapat diunduh dengan menekan tombol "Download PDF" ..

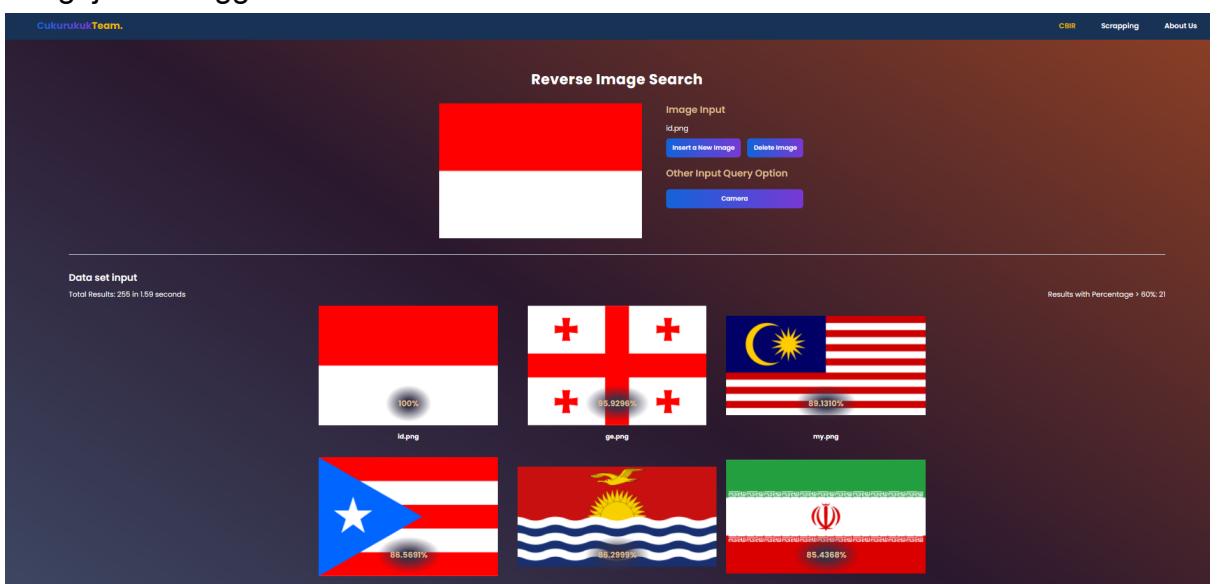
4.4. Hasil Pengujian

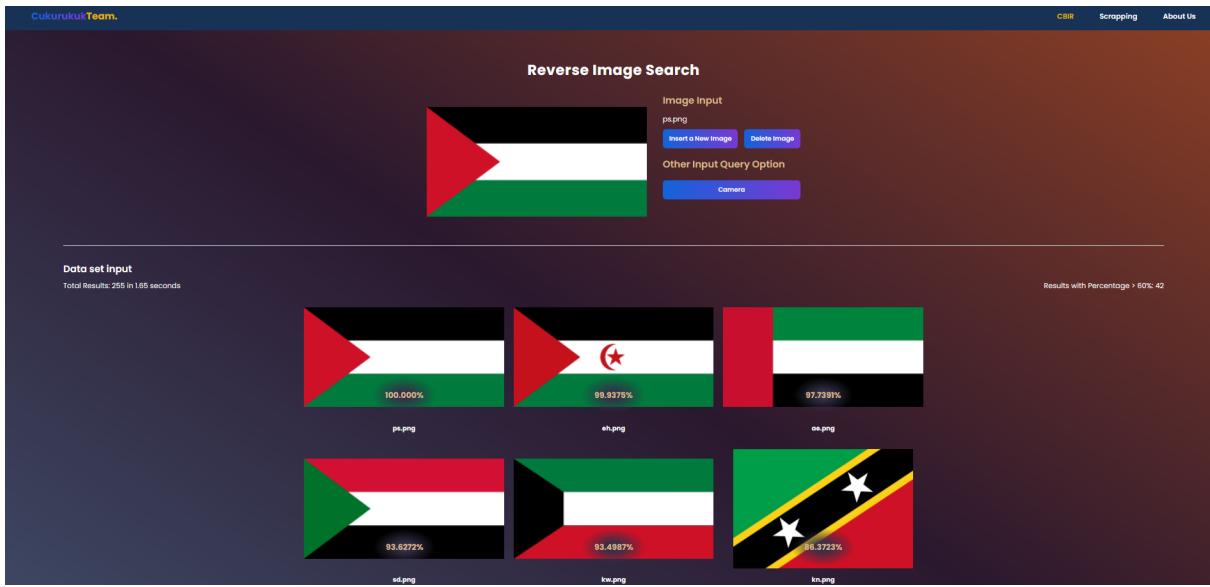
4.4.1. CBIR Color

- a. Pengujian menggunakan dataset singa

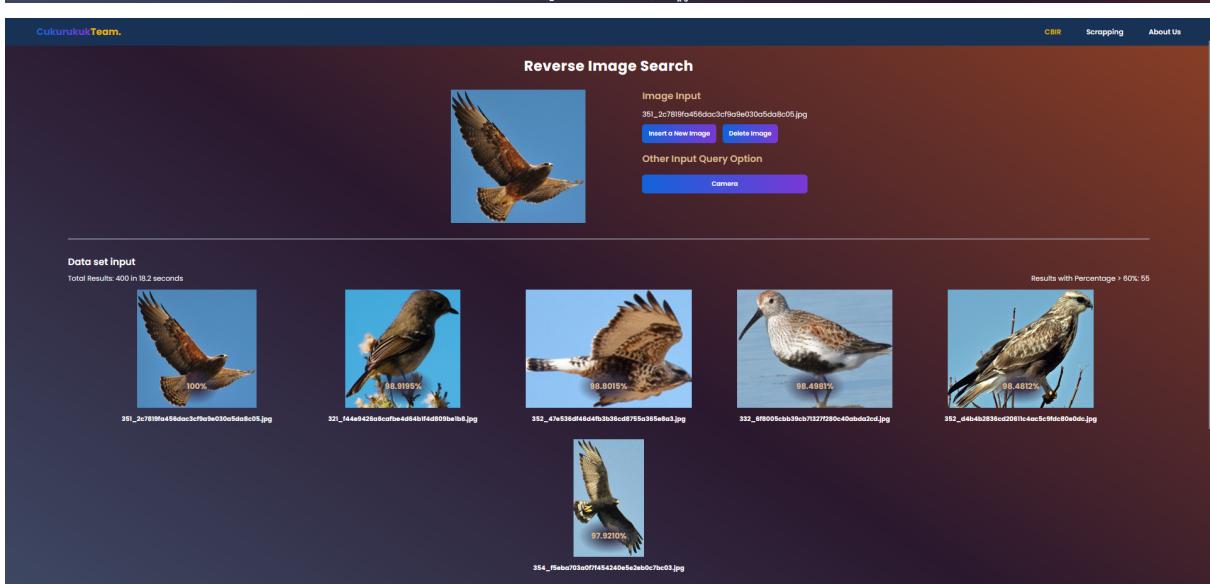
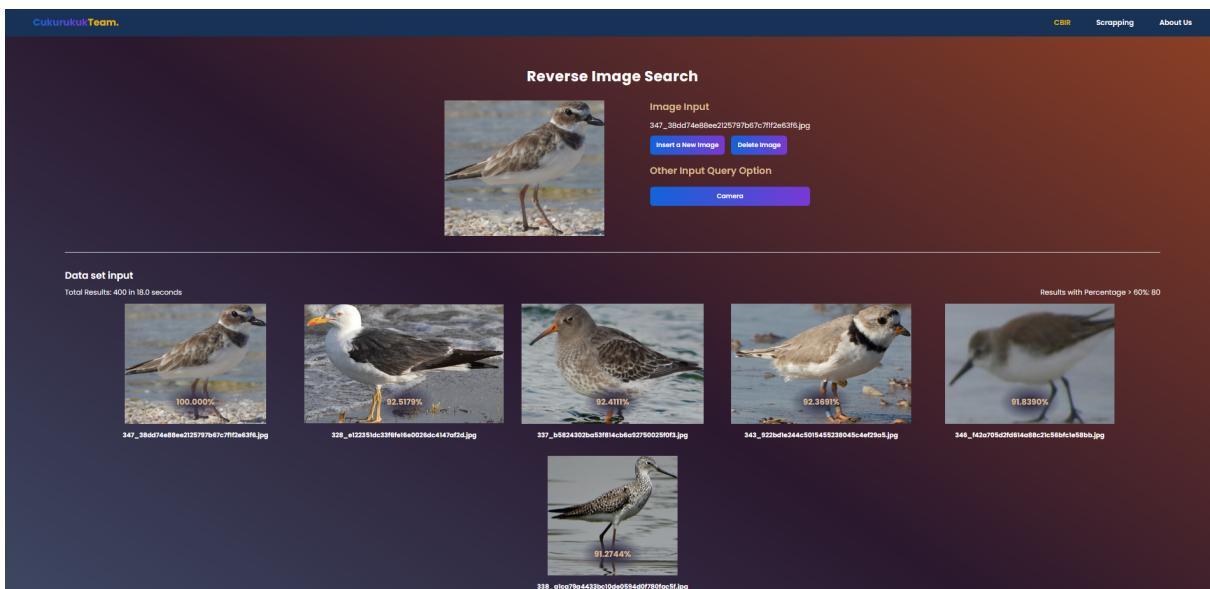


b. Pengujian menggunakan dataset bendera





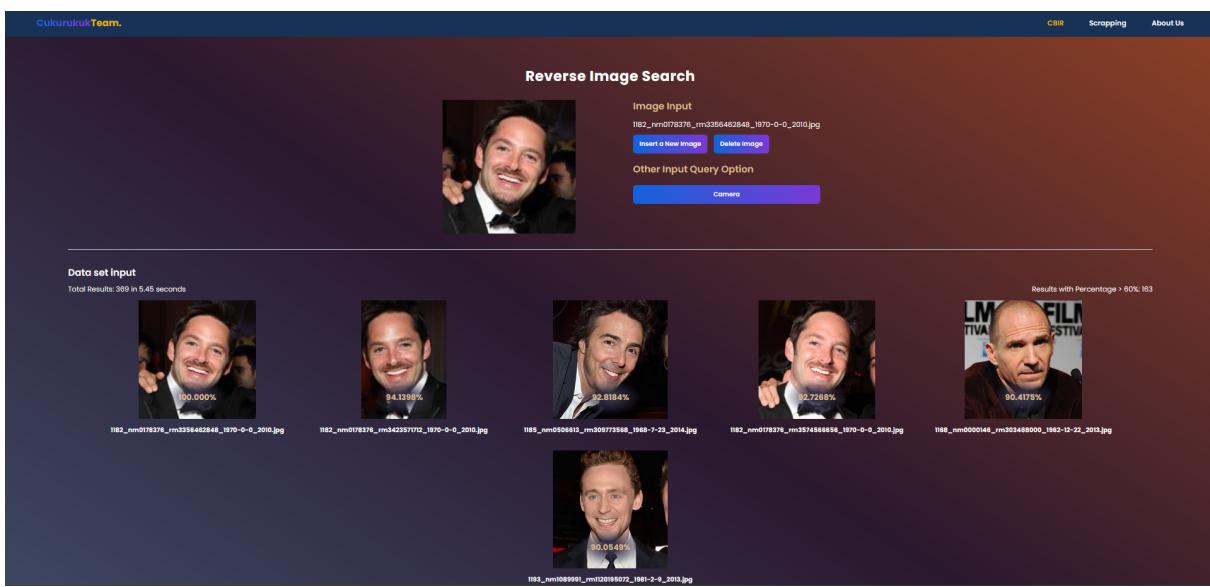
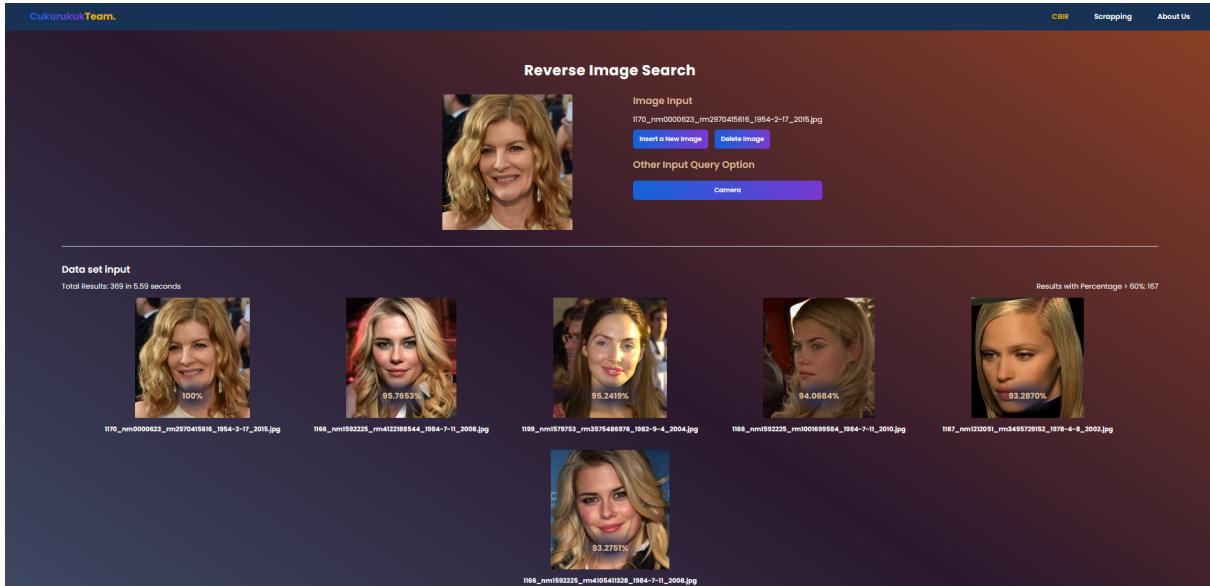
c. Pengujian menggunakan dataset burung



d. Pengujian menggunakan dataset gedung

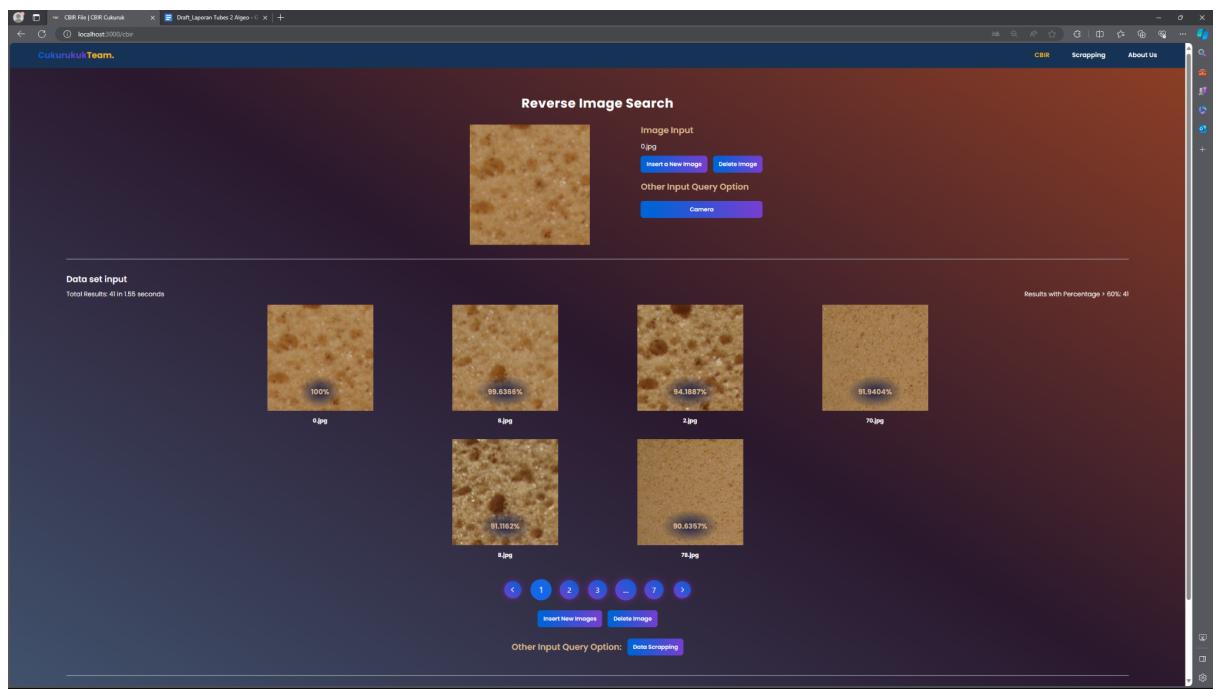
The screenshot shows the CukurukukTeam Reverse Image Search interface. At the top, there is a header with the logo "CukurukukTeam.", navigation links for "CBIR", "Scraping", and "About Us", and a "Reverse Image Search" button. Below the header, the main search area is titled "Reverse Image Search". It features a central image input field containing a photo of a two-story half-timbered house with a red roof. To the right of the input field are buttons for "Image Input" (containing the file path "149_619ca327e72839e9.jpg"), "Insert a New Image", "Delete Image", "Other Input Query Option", and "Camera". Below the input field, a section titled "Data set Input" displays the original image with a "100%" confidence rating. To the right, a section titled "Results with Percentage > 50%: 70" shows five similar building images with their respective confidence percentages: 97.0202%, 95.4929%, 94.7422%, 93.5458%, and 91.2879%. At the bottom, there is another "Reverse Image Search" section with its own image input field showing a castle-like building, followed by a "Data set Input" section with the original castle image and a "Results with Percentage > 50%: 29" section showing five castle images with confidence percentages: 100%, 98.7067%, 91.8285%, 90.5143%, and 88.3764%.

e. Pengujian menggunakan dataset orang

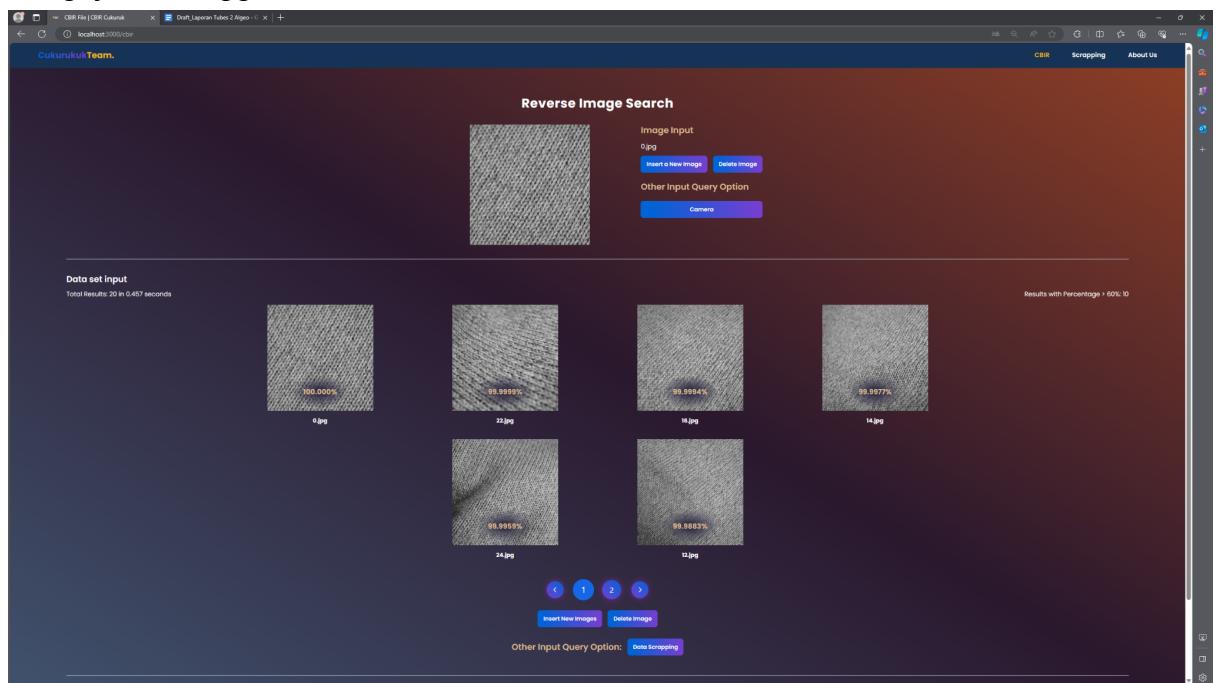


4.4.2. CBIR Texture

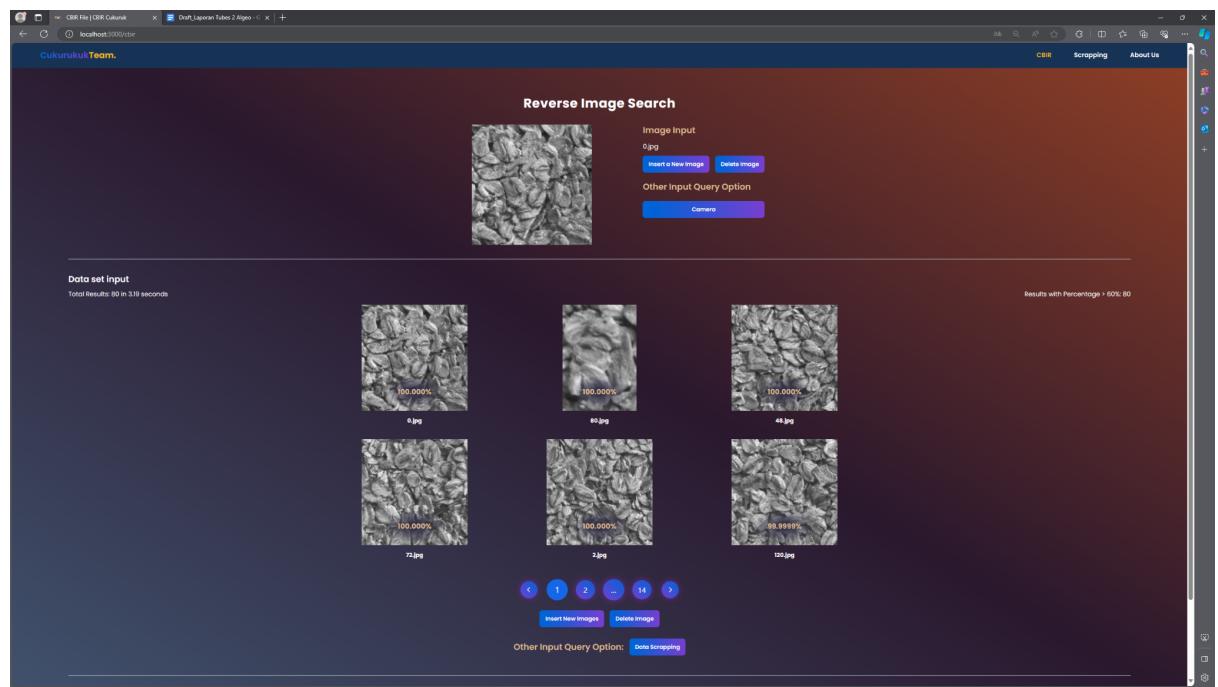
- a. Pengujian menggunakan dataset texture sponge



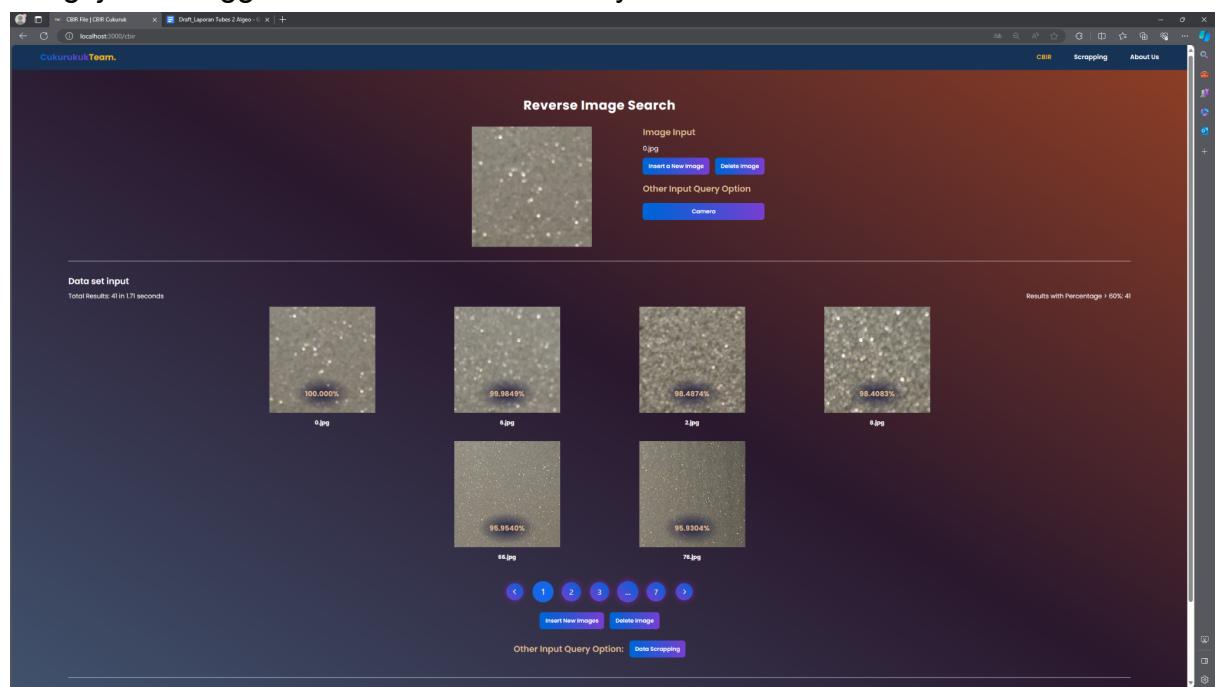
b. Pengujian menggunakan dataset texture knit



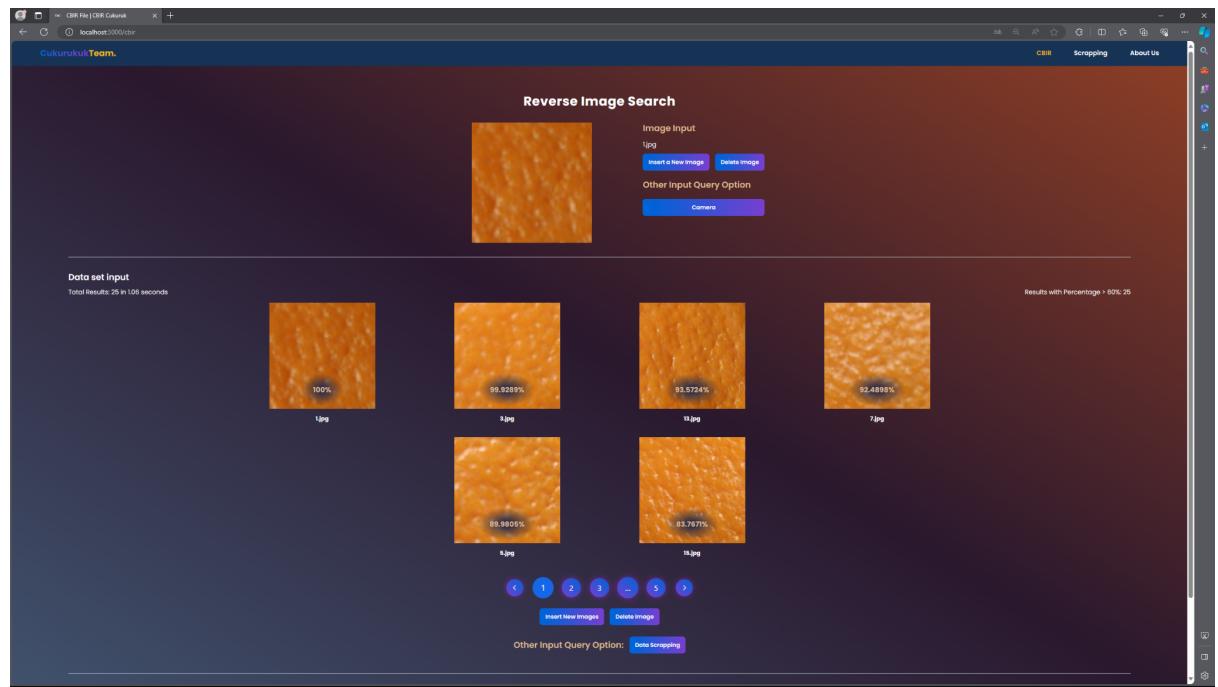
c. Pengujian menggunakan dataset texture oatmeal1



d. Pengujian menggunakan dataset texture styrofoam



e. Pengujian menggunakan dataset orange_peel1



4.5. Analisis

Pembangunan CBIR berdasarkan warna dapat memberikan hasil baik dalam kasus gambar yang menekankan perbedaan warna yang signifikan. Cocok untuk situasi dimana pengguna ingin mengenali atau mencari gambar berdasarkan warna objek yang spesifik. Namun, jika suatu objek tidak memiliki warna yang sangat khas, pencocokan warna mungkin tidak efektif dalam mengidentifikasi atau mencari objek tersebut. Sedangkan, CBIR berdasarkan tekstur lebih efektif dalam mengidentifikasi gambar berdasarkan struktur, pola, atau detail tekstur tertentu. Tidak terlalu dipengaruhi oleh perubahan pencahayaan karena fokus pada struktur gambar. Namun, kurang efektif untuk gambar dengan warna yang lebih monoton atau homogen. Sehingga metode yang lebih baik dari yang lain tidak dapat ditentukan, karena pilihan antara warna dan tekstur bergantung pada kebutuhan spesifik pengguna.

BAB 5

5.1. Kesimpulan

Dalam Tugas Besar 2 ini, tujuan utama adalah mengimplementasikan sistem temu balik gambar (CBIR) dengan dua parameter populer: warna dan tekstur. Pada parameter warna, pendekatan menggunakan histogram warna global HSV memberikan gambaran distribusi warna dengan lebih baik. Sementara itu, pada parameter tekstur, penerapan co-occurrence matrix digunakan untuk mengekstrak informasi tekstur dari gambar dalam bentuk matriks. Langkah-langkah ini melibatkan konversi gambar ke grayscale, kuantifikasi nilai grayscale, dan ekstraksi komponen tekstur seperti contrast, homogeneity, dan entropy. Pembentukan vektor dari kedua parameter tersebut menggunakan *cosine similarity* untuk mengukur tingkat kemiripan antara gambar.

Dengan mengintegrasikan kedua parameter ini, sistem temu balik gambar dapat memberikan solusi holistik untuk pencarian gambar berbasis konten. Pemanfaatan aljabar vektor dalam kedua parameter memungkinkan representasi yang efisien dan perbandingan yang akurat antara gambar-gambar dalam dataset. Sebagai hasilnya, pengguna dapat dengan mudah mencari dan mengelola koleksi gambar mereka tanpa perlu bergantung pada kata kunci atau teks, melainkan berdasarkan karakteristik visual yang lebih mendalam.

5.2. Saran

Terdapat beberapa saran diantaranya adalah diperlukan beberapa optimasi dalam algoritma pencocokan warna dan tekstur untuk meningkatkan efisiensi sistem secara keseluruhan. Kemudian, untuk pencocokan lebih akurat dapat digunakan kedua CBIR secara bersamaan. Dari sisi kelompok juga pembagian tugas sebaiknya dapat dilakukan lebih baik lagi.

5.3. Komentar atau Tanggapan

Tugas besar kedua ini sangat bermanfaat dalam melatih penerapan aljabar geometri vektor khususnya dalam konteks sistem temu balik gambar sehingga dapat dimanfaatkan dalam dunia nyata dalam pemrosesan citra. Kemudian dengan kolaborasi kelompok yang baik, tugas besar ini dapat diselesaikan.

5.4. Refleksi

Dari pengalaman Tugas Besar Algeo 2 ini, ada beberapa kekurangan pada kelompok kami, mungkin seperti lama waktu yang dibutuhkan dalam membandingkan image input dengan dataset, namun hal ini dapat diperbaiki kedepannya untuk meningkatkan kualitas proyek. Kami juga menyadari perlunya

perbaikan pada visualisasi hasil pencarian untuk memberikan pengalaman pengguna yang lebih baik. Selain itu, penggunaan dataset yang lebih beragam dapat meningkatkan validitas hasil dan membantu mengidentifikasi potensi kelemahan dalam algoritma. Refleksi ini menjadi landasan untuk perbaikan di masa mendatang, mengarahkan kami untuk terus meningkatkan kualitas solusi khususnya dalam pengembangan proyek yang berbasis aljabar geometri.

5.5. Ruang Perbaikan dan Pengembangan

Terdapat beberapa perbaikan dan pengembangan yang dapat dipertimbangkan diantaranya adalah optimasi algoritma, yaitu evaluasi ulang algoritma sehingga meningkatkan efisiensi dan kinerja sistem secara keseluruhan. Ekstensibilitas sistem agar dapat diintegrasikan dengan mudah seperti fitur tambahan sehingga dapat meningkatkan fungsionalitas dan visualisasi website juga dapat ditingkatkan.

DAFTAR PUSTAKA

<https://www.sciencedirect.com/science/article/pii/S0895717710005352>

[Content-Based Image Retrieval Using Color, Shape and Texture Descriptors and Features | Arabian Journal for Science and Engineering \(springer.com\)](#)

<https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcmb10c45b6d46a1>

<https://github.com/pjreddie/darknet/issues/2557>

https://www.researchgate.net/figure/Performance-comparison-of-various-object-detection-models_fig5_349960212

<https://jonathan-hui.medium.com/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359>

<https://medium.com/@robertsevan/scraping-dynamically-loaded-content-with-selenium-and-beautifulsoup-2cb7b067ea6c>
[Kylberg Texture Dataset \(kaggle.com\)](#)

Link *repository* github : <https://github.com/fairuzald/Algeo02-22027>

Link *website* : <https://algeo02-22027.vercel.app/>

Link *video* : https://www.youtube.com/watch?v=qAsYhie9_KI