

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



Disusun Oleh:

Farhan Nafis Rayhan 13522037

Emery Fathan Zwageri 13522079

Muhamad Rifki Vrziadeili Harisman 13522120

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

DAFTAR ISI

BAB I	3
I. Tujuan	3
II. Spesifikasi	3
BAB II	4
A. CBIR Warna	4
B. CBIR Tekstur	4
II. Pengembangan Website	6
A. Front End	6
B. Back End	6
BAB III	7
I. Langkah Penyelesaian Masalah	7
B. CBIR Tekstur	8
II. Pemetaan Masalah	9
B. CBIR Tekstur	9
III. Contoh Ilustrasi Kasus	10
BAB IV	11
I. Implementasi Program Utama	11
II. Struktur Program Utama	16
III. Tata Cara Penggunaan Program	16
IV. Hasil Pengujian	17
V. Analisis Desain Solusi Algoritma	19
BAB V	21
I. Kesimpulan	21
II. Saran	21
III. Komentar/Tanggapan	21
IV. Refleksi	21
V. Perbaikan dan Pengembangan	22
REFERENSI	23

BAB I

DESKRIPSI MASALAH

I. Tujuan

Tugas besar ini dibuat dalam rangka memenuhi tugas besar II mata kuliah IF2123 Aljabar Linear dan Geometri. Tujuan dari tugas besar ini adalah memanfaatkan Aljabar Vektor dalam mengimplementasikan sistem temu balik gambar berbentuk *website*. Aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (*Content-Based Image Retrieval* atau yang sering disingkat CBIR), dimana sistem ini menggunakan metode identifikasi gambar berdasarkan konten visualnya. Terdapat 2 parameter konten visual yang diimplementasikan, antara lain parameter warna & tekstur.

II. Spesifikasi

Program sistem temu balik gambar (*image retrieval system*) yang dibuat memenuhi beberapa spesifikasi sebagai berikut:

1. Program menerima input *folder dataset* dan sebuah citra gambar, utamanya menggunakan *dataset* pada pranala [berikut](#).
2. Program menampilkan gambar citra gambar yang dipilih oleh pengguna.
3. Program dapat memberikan pilihan bagi pengguna untuk memilih parameter pencarian yang hendak digunakan (yaitu parameter warna atau tekstur) melalui *toggle*.
4. Program mulai melakukan perhitungan nilai kecocokan antara *image* masukan dengan *dataset image* berdasarkan parameter yang telah dipilih (warna atau tekstur).
5. Program akan menampilkan nilai kecocokan antara *image* masukan dengan seluruh gambar dalam dataset.
6. Hasil luaran juga menampilkan setiap gambar dengan tingkat kemiripan $> 60\%$ yang terurut secara *descending*.
7. Program mengimplementasikan *pagination* agar jumlah gambar dapat dibatasi dengan halaman-halaman tertentu.
8. Program dapat menampilkan banyaknya gambar pada *dataset* yang memenuhi kondisi tingkat kemiripan serta waktu eksekusi pencarian.

BAB II

LANDASAN TEORI

I. **CONTENT-BASED INFORMATION RETRIEVAL (CBIR)**

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Proses ini dimulai dengan ekstraksi fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk. Setelah fitur-fitur tersebut diekstraksi, mereka diwakili dalam bentuk vektor atau deskripsi numerik yang dapat dibandingkan dengan gambar lain. Kemudian, CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan ini digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling mirip dengan gambar yang dicari. Proses CBIR membantu pengguna dalam mengakses dan mengeksplorasi koleksi gambar dengan cara yang lebih efisien, karena tidak memerlukan pencarian berdasarkan teks atau kata kunci, melainkan berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut. Pada Tugas Besar kali ini, Anda diminta untuk mengimplementasikan 2 parameter CBIR yang paling populer, antara lain :

A. **CBIR Warna**

Histogram warna mencerminkan sebaran frekuensi berbagai warna dalam suatu ruang warna, yang bertujuan untuk menggambarkan distribusi warna pada suatu gambar. Tetapi, histogram warna tidak memiliki kemampuan untuk mengidentifikasi suatu objek secara spesifik dalam gambar dan tidak dapat memberikan deskripsi mengenai lokasi warna yang tersebar. CBIR warna akan membandingkan masukan dari sebuah gambar dengan suatu gambar yang dimiliki oleh *dataset*. Metode ini mengubah gambar yang berbentuk awalnya RGB menjadi sebuah metode histogram warna yang lebih umum.

Pembentukan ruang warna diperlukan untuk membagi nilai-nilai citra ke dalam beberapa rentang nilai yang lebih kecil. Langkah ini bertujuan untuk membentuk histogram warna, di mana setiap interval rentang dianggap sebagai *bin*. Perhitungan histogram warna melibatkan penghitungan piksel yang mewakili nilai warna dalam setiap interval. Pada umumnya, warna global HSV lebih dipilih dalam perhitungan histogram karena ia merupakan ruang warna yang lebih umum untuk digunakan.

B. **CBIR Tekstur**

CBIR dengan perbandingan tekstur umumnya menggunakan suatu matriks yang dinamakan *co-occurrence matrix*. Matriks ini lebih banyak dipilih karena dapat

melakukan pemrosesan yang lebih mudah namun juga cepat, terlebih ukuran vektor yang dihasilkan juga mempunyai ukuran yang lebih kecil. Misalkan terdapat suatu gambar I dengan $n \times m$ piksel dan suatu parameter offset $(\Delta x, \Delta y)$. Dengan menggunakan nilai i dan j sebagai nilai intensitas dari gambar dan p serta q sebagai posisi dari gambar, maka *offset* Δx dan Δy bergantung pada arah θ dan jarak yang digunakan melalui persamaan di bawah ini. Besar sudut θ adalah 0° , 45° , 90° , dan 135° . Maka, dapat dirumuskan matriksnya sebagai

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

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

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

Setelahnya bisa diperoleh *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Lalu didapat *matrix normalization* dengan persamaan.

$$\text{MatrixNorm} = \frac{\text{MatrixOcc}}{\sum \text{MatrixOcc}}$$

C. Cosine Similarity

Tingkat kesamaan suatu gambar dikalkulasi melalui besarnya hasil cosinus pada persamaan dibawah ini. A dan B merupakan dua vektor dari kedua gambar, yang diperoleh dengan mengekstraksi nilai *energy*, *contrast*, *homogeneity*, *entropy*, *dissimilarity*, dan *ASM*.

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

II. Pengembangan Website

Pada pengerjaan tugas besar kali ini, kelompok kami mengintegrasikan *react.js* sebagai *frontend* dan *express-node.js* sebagai *backend*.

A. *Front End*

Front End merujuk pada bagian dari pengembangan perangkat lunak yang berfokus pada pengembangan antarmuka pengguna (UI) dan interaksi pengguna di sisi klien (*client side*) dari aplikasi atau situs web. *Front End* mencakup elemen-elemen yang dapat dilihat dan dirasakan oleh pengguna akhir, seperti halaman web, tata letak, warna, *font*, formulir, dan elemen interaktif lainnya.

Front End dirancang agar responsif terhadap berbagai ukuran layar dan perangkat, memberikan pengalaman pengguna yang baik dari desktop hingga perangkat mobile. Pengembang *Front End* perlu memperhatikan aksesibilitas web untuk memastikan bahwa halaman web dapat diakses oleh semua pengguna, termasuk mereka yang memiliki tantangan aksesibilitas.

B. *Back End*

Secara teori, *Back End* merujuk pada bagian dari pengembangan perangkat lunak yang berfokus pada pengelolaan dan pengolahan data di sisi server (*server side*) suatu aplikasi. *Back End* menangani logika, pengelolaan database, dan pemrosesan server yang tidak terlihat oleh pengguna akhir. Ini bekerja di balik layar untuk mendukung fungsionalitas dan data yang diperlukan oleh *Front End*.

Back End bertanggung jawab untuk mengelola data, termasuk penyimpanan, pengambilan, pembaruan, dan penghapusan data dari *database*. *Back End* harus menjaga keamanan data dan informasi sensitif. Ini mencakup pengaturan hak akses, enkripsi data, dan langkah-langkah keamanan lainnya untuk melindungi aplikasi dari serangan dan pelanggaran keamanan. Tetapi, *Back End* juga perlu menangani pertumbuhan dan beban kerja yang mungkin berubah seiring waktu.

Komunikasi antar keduanya diatur oleh *API (Application Programming Interface)*. Antarmuka ini memungkinkan aplikasi untuk mengirim dan menerima data serta meminta operasi tertentu dari server.

BAB III

ANALISIS PENYELESAIAN MASALAH

I. Langkah Penyelesaian Masalah

A. CBIR Warna

1. Gambar diubah ukurannya menjadi 300x300 pixel, lalu Matrix warna R,G, dan B, diekstrak.
2. Nilai dari RGB harus dinormalisasi dari *range* [0, 255] menjadi range [0, 1]

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

3. Nilai C_{max} , C_{min} , dan Δ diperoleh dengan

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

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

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

4. Hasil perhitungan di atas untuk mendapatkan nilai HSV melalui persamaan

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

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

$$V = C_{maks}$$

5. Gambar dibagi menjadi blok 4x4 dimana nilai HSV untuk setiap blok dirata-ratakan.
6. Perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity*. Hal ini dilakukan satu-persatu untuk masing-masing blok untuk mengoptimasi perhitungan.

7. Hasil perhitungan setiap blok dirata-ratakan, lalu semua *image* pada *dataset* diurutkan berdasarkan nilai kecocokan tertinggi.

B. CBIR Tekstur

1. Gambar diubah ukurannya menjadi 300x300 pixel, lalu Matrix warna R,G, dan B, diekstrak.
2. Karena warna tidak penting dalam analisis tekstur, gambar diubah terlebih dahulu menjadi grayscale melalui persamaan.

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

3. *Co-occurrence matrix* yang berukuran 256×256 dikalkulasi dengan persamaan yang tertera pada bab sebelumnya.
4. Dari *co-occurrence matrix* bisa diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity* dengan persamaan:

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 :

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

5. *Dissimilarity*, *energy*, dan *ASM* juga dikalkulasi, lalu kelima parameter diatas dijadikan vektor yang melambangkan suatu gambar.
6. Ukurlah kemiripan dari kedua gambar dengan menggunakan *Teorema Cosine Similarity*, kemudian gambar pada *dataset* diurutkan berdasarkan tingkat kecocokan.

II. Pemetaan Masalah

A. CBIR Warna

1. Konversi RGB ke HSV:
RGB dan HSV keduanya merupakan suatu titik pada ruang warna, yang bisa dinyatakan sebagai vektor 3 dimensi. RGB terdiri atas 3 komponen yaitu *channel Red*, *Green*, dan *Blue*. Sedangkan HSV terdiri atas komponen *Hue*, *Saturation*, dan *Value*.
2. Pembagian menjadi blok:
Gambar dibagi menjadi 5625 buah blok 4x4 dimana nilai HSV setiap pixel dalam blok dirata-ratakan.
3. Cosine Similarity:
Untuk 2 gambar yang ingin dibandingkan, nilai kemiripan vektor HSV masing masing blok dikalkulasi lalu dirata-ratakan.

B. CBIR Tekstur

1. Konversi RGB ke Grayscale:
Mengalikan setiap *channel* dari matrix HSV dengan suatu konstanta
2. Pembentukan *Co-occurrence matrix & Normalized GLCM*:
Operasi pada setiap *pixel* pada *grayscale* agar diperoleh penggambaran tekstur.
3. Ekstraksi 6 parameter:
Meliputi *energy*, *contrast*, *homogeneity*, *entropy*, *dissimilarity*, dan *ASM*. Terbentuk sebuah vektor 6 dimensi.
4. Cosine Similarity:
Untuk 2 gambar yang ingin dibandingkan, Dibandingkan vektornya agar diperoleh persentase kemiripan keduanya.

III. Contoh Ilustrasi Kasus

A. CBIR Warna

Kasus : Mendapatkan gambar yang paling mirip dari parameter warna.

1. Format warna gambar diubah dari RGB ke HSV
2. Nilai HSV setiap blok dirata-ratakan
3. Dilakukan Cosine Similarity untuk masing-masing blok agar diperoleh hasil rata-rata.
4. Gambar dengan hasil cosine similarity tertinggi adalah yang paling mirip.

B. CBIR Tekstur

Kasus : Membandingkan tekstur 2 gambar

1. Setiap citra gambar diubah menjadi grayscale
2. Cari *GLCM matrix*, kemudian tentukan *Normalized Matrix*-nya
3. Hitung fitur dari *GLCM Matrix* sebagai komponen vektor.
4. Lakukan *Cosine Similarity* antara kedua gambar yang ingin dibandingkan
5. Semakin besar hasilnya, semakin besar kemiripan tekstur kedua gambar.

BAB IV

IMPLEMENTASI DAN UJI COBA

I. Implementasi Program Utama

A. Modul CBIR.py

```
import sys
import CBIRTekstur as texture
import CBIRWarna as color
import numpy as np
from PIL import Image
import os
import threading
import time
import cProfile
import concurrent.futures
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES <- True
import json

selected_option <- sys.argv[1]

{ --- Similarity ---}
function cosine_similarity(a : matrix of vectors, b : matrix of vectors) -> array of real
    dots <- np.einsum('ij, ij->i', a,b)
    norms <- np.linalg.norm(a, axis = 1)*np.linalg.norm(b, axis = 1)
    -> np.where(norms=0, 0, dots/norm)

procedure search_similarities(input: database_features, input: database_path, input:
query_vector, output: result_dict)
    try:
        dataset_vector <- np.array(database_features[database_path])
        res_value <- texture.cosine_similarity(query_vector,dataset_vector)
    except Exception as e:
        output(f"Error processing {database_path}: {e}")
        res_value <- None

    result_dict[database_path] <- res_value
    result_dict[database_path] <- "{:.4f}".format(result_dict[database_path])

function perform_similarity_analysis(database_features : matrix, query_feature : matrix) ->
dictionary
    result_dict <- {}
    query_vector<= np.array(list(query_feature.values())[0])

    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures <- {executor.submit(search_similarities, database_features, path,
query_vector, result_dict): path for path in database_features}

        concurrent.futures.wait(futures)

    -> result_dict

{--- Color ---}
function perform_color_analysis_query(image_paths : path, base_path : path) -> matrix
```

```

try:
    for image_filename in image_paths
        image_path <- os.path.join(base_path,image_filename)
        rgb_img <- Image.open(image_path)
        rgb_img <- rgb_img.resize((300,300))
        if rgb_img.mode not 'RGB' then
            raise ValueError("Image is not in RGB mode.")
        img <- np.array(rgb_img)
        hsv <- color.RGBtoHSV(img)
        blocks <- hsv[:75 * 4, :75 * 4].reshape(75, 4, 75, 4, 3)
        repVal <- np.sum(blocks, axis=(1,3))
        repVal = repVal / 16
        repVal = repVal.resize(5625,3)
except Exception as e:
    output(f"Error processing {image_path}: {e}")
-> repVal

function perform_color_analysis_database(image_paths : path, base_path : path) -> dictionary
try:
    resultDict <- {}
    for image_filename in image_paths:
        image_path <- os.path.join(base_path,image_filename)
        rgb_img <- Image.open(image_path)
        rgb_img <- rgb_img.resize((300, 300))
        if rgb_img.mode not 'RGB' then
            raise ValueError("Image is not in RGB mode.")

        img <- np.array(rgb_img)
        hsv <- color.RGBtoHSV(img)
        blocks <- hsv[:75 * 4, :75 * 4].reshape(75, 4, 75, 4, 3)
        repVal <- np.sum(blocks, axis=(1,3))
        repVal <- repVal / 16
        repVal <- repVal.resize(5625,3)
        result <- np.where(repVal.any(axis = 1), cosine_similarity(repVal, query), 0)
        similarity <- np.sum(result)
        resultDict[image_path] <- similarity/5625
        resultDict[image_path] <- "{:.2f}".format(resultDict[image_path])
except Exception as e:
    output(f"Error processing {image_path}: {e}")
-> resultDict

{ --- Texture ---}

procedure extract_texture_features(input : image_path, output : result_dict, input : lock)
features <- None
try:
    rgb_img <- Image.open(image_path)
    rgb_img <- rgb_img.resize((300, 300))
    if rgb_img.mode not 'RGB' then
        raise ValueError("Image is not in RGB mode.")
    img_grayscale <- texture.convertToGrayscale(rgb_img)
    coocurrence_matrix <- texture.cooccurrenceMatrix(img_grayscale,0)
    glcm_matrix <- texture.symmetricMatrix(coocurrence_matrix)
    glcm_matrix <- texture.normMatrix(glcm_matrix)

    energy <- texture.energy(glcm_matrix)
    entropy <- texture.entropy(glcm_matrix)
    contrast <- texture.contrast(glcm_matrix)
    homogeneity <- texture.homogeneity(glcm_matrix)

```

```

        asm <- texture.ASM(glcm_matrix)
        dissimilarity <- texture.dissimilarity(glcm_matrix)
        features <- [energy,entropy,contrast,homogeneity,asm,dissimilarity]

except Exception as e:
    output(f"Error processing {image_path}: {e}")

with lock :
    result_dict[image_path] <- features

function perform_texture_analysis(image_paths : path, base_path : path) -> dictionary
try:
    result_dict <- {}
    lock <- threading.Lock()
    threads <- []

    for image_filename in image_paths:
        image_path <- os.path.join(base_path,image_filename)
        thread <- threading.Thread(target=extract_texture_features, args=(image_path,
result_dict,lock))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

except Exception as e:
    output(f"Error processing {image_path}: {e}")
-> result_dict

{--- Main ---}

if __name__ == "__main__" then

    if selected_option == "texture" then
        script_path_relative <- os.path.dirname(os.path.abspath(__file__))
        base_path_query_list <-
os.listdir("E:\\Tubes-Algeo2\\Algeo02-22037\\uploads\\client_image")

        Query_features <-
perform_texture_analysis(base_path_query_list,"E:\\Tubes-Algeo2\\Algeo02-22037\\uploads\\cli
ent_image")

        database_path <- os.listdir("E:\\Tubes-Algeo2\\Algeo02-22037\\uploads\\dataset")
        database_features <-
perform_texture_analysis(database_path,"E:\\Tubes-Algeo2\\Algeo02-22037\\uploads\\dataset")

        result_similarity <- perform_similarity_analysis(database_features,query_features)
        sorted_result <- dict(sorted(result_similarity.items(), key=lambda item:
item[1],reverse=True))
        result_list <- list(sorted_result.items())
        result_list <- [i for i in result_list if float(i[1])>=0.60]
        json_list <- json.dumps(result_list)
        output(json_list)

    else if selected_option == "color" then
        script_path_relative <- os.path.dirname(os.path.abspath(__file__))

```

```

base_path_query <- os.path.join(script_path_relative,'..','uploads','client_image')
base_path_query_list <- os.listdir(base_path_query)

query_representative_blocks <-
perform_color_analysis_query(base_path_query_list,base_path_query)
base_path_database <- os.path.join(script_path_relative,'..','uploads','dataset')
database_path <- os.listdir(base_path_database)
database_result <- perform_color_analysis_database(database_path,base_path_database,
query_representative_blocks)
sorted_result <- dict(sorted(database_result.items(), key=lambda item: item[1],
reverse=True))
result_list <- list(sorted_result.items())
result_list <- [i for i in result_list if float(i[1])>=0.60]

json_list <- json.dumps(result_list)
output(json_list)

```

B. Modul CBIRTekstur.py

```

from math import *
import numpy as np
from PIL import Image
from numpy.linalg import norm
function convertToGrayscale(rgb_image : image) -> matrix 300x300 of real

    rgb_array <- np.array(rgb_image)

    brightness <- np.dot(rgb_array[:, :, :3], [0.29, 0.587, 0.114])

    grayscale_image <- brightness.astype(np.uint8)

    -> grayscale_image

function cooccurrenceMatrix(matrix : matrix 300x300 of real, angle : real) -> matrix
256x256 of real
    res <- matrix 256x256 of real
    x <- cos(angle)
    y <- sin(angle)

    if (x < 0.5) then
        x <- 0
    else
        x <- 1

    if (y < 0.5) then
        y <- 0
    else
        y <- 1

    size <- matrix.shape
    i traversal[0..size[0]-y]
    j traversal[0..size[0]-x]
        row <- matrix[i,j]
        col <- matrix[i+y,j+x]
        res[row, col] <- res[row, col] + 1
    -> res

function symmetricMatrix(matrixCo : matrix 256x256 of real) -> matrix 256x256 of real

```

```

t <- matrixCo.transpose()
res <- t + matrixCo
-> res

function normMatrix(matrixSym : matrix 256x256 of real) -> matrix 256x256 of real
sum <- matrixSym.sum()
-> (matrixSym/sum)

function contrast(matrixNorm : matrix 256x256 of real) -> real
sum <- 0
i traversal[0..len(matrixNorm)]
  j traversal[0..len(matrixNorm)]
    sum <- sum + matrixNorm[i][j]*(i-j)^2
-> sum

function homogeneity(matrixNorm : matrix 256x256 of real) -> real
sum =0
i traversal[0..len(matrixNorm)]
  j traversal[0..len(matrixNorm)]
    Sum <- sum + matrixNorm[i][j]/(1+(i-j)^2)
-> sum

function entropy(matrixNorm : matrix 256x256 of real) -> real
sum =0
i traversal[0..len(matrixNorm)]
  j traversal[0..len(matrixNorm)]
    if matrixNorm[i][j]>0 then
      sum <- sum + matrixNorm[i][j]*log2(matrixNorm[i][j])
-> sum*(-1)

function dissimilarity(matrixNorm : matrix 256x256 of real) -> real
sum = 0
i traversal[0..len(matrixNorm)]
  j traversal[0..len(matrixNorm)]
    sum <- sum + matrixNorm[i][j]*abs(i-j)
-> sum

function ASM(matrixNorm : matrix 256x256 of real) -> real
sum =0
i traversal[0..len(matrixNorm)]
  j traversal[0..len(matrixNorm)]
    sum <= sum matrixNorm[i][j]^2
-> sum

function energy(matrixNorm : matrix 256x256 of real) -> real
-> sqrt(ASM(matrixNorm))

```

C. Modul CBIRWarna.py

```

import numpy as np
from PIL import Image
from math import *

function RGBtoHSV(arr : matrix 300x300 of RGB channel) -> matrix 300x300 of tuple [0..2]
of real
arr <- arr / 225

```

```

Val <- np.max(arr, axis=-1)
Cmin <- np.min(arr, axis=-1)
Delta <- Val - Cmin
Sat <- np.where(Val = 0, 0, Delta/Val)
R <- Red channel of arr
G <- Green channel of arr
B <- Green channel of arr
Hue <- np.where(Delta == 0, 0,
  np.select([Val = R, Val = G, Val = B], [((pi/3) * (G-B) / Delta)%6,
                                           ((pi/3) * (B-R) / Delta)+2,
                                           ((pi/3) * (R-G) / Delta)+4])
-> np.stack([Hue, Sat, Val], axis = -1)

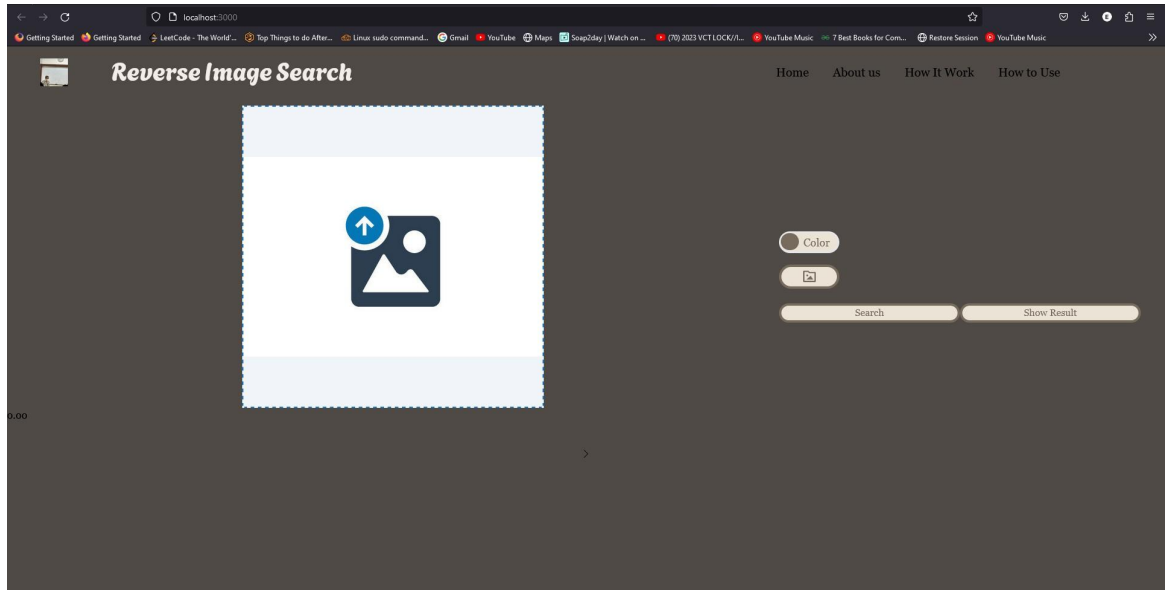
```

II. Struktur Program Utama

Program utama adalah program cbir yang ditulis dengan python dengan nama CBIR.py. Kami menjalankan program ini dengan modul child process yang mengeksekusi terminal “python3 CBIR/CBIR.py argument” . argument disini diterima dari toggle value dari client. Bisa texture ataupun color. Program CBIR kami mengakses file di uploads folder yang dijadikan list. Untuk argumentnya texture kami melakukan multithreading untuk extract feature agar lebih cepat. Pada percobaan kami , sebelum multithreading, 4838 foto 27 menit sedangkan setelah multithreading menjadi 18 menit. Didapat bahwa multi threading dapat menjalankan program 33% lebih cepat. Setelah program mengekstrak feature maka dilakukan perkalian vektor cosine similarity dari feature image query dan dataset. Untuk color program merubah setiap foto menjadi blok blok dulu dengan fungsi yang telah dibuat, lalu dibandingkan dengan cosine similarity.

III. Tata Cara Penggunaan Program

Berikut adalah halaman awal dari *interface web* yang kami buat



Pada bagian atas halaman, terdapat navigation bar yang memiliki beberapa fitur sebagai berikut:

- Home: Kembali ke halaman utama
- About us: Informasi tentang pengembang program
- How it works: Informasi mengenai bagaimana program bekerja
- How to use: Bagaimana cara menggunakan program

Terdapat container besar untuk memasukkan gambar yang ingin dicari pada bagian kiri program. Sedangkan pada bagian kanan terdapat beberapa tombol yaitu:


- Toggle bar: memilih parameter yang akan digunakan (warna/gambar)
- Upload dataset: memilih directory dataset yang akan digunakan
- Search: perintah memulai pencarian
- Show Result: menampilkan hasil pencarian foto di dataset yang paling menyerupai

IV. Hasil Pengujian

A. CBIR Warna

localhost:3000


Getting StartedGetting StartedLeetCode - The World...Top Things to do After...Linux sudo command...GmailYouTubeMapsSoap2day | Watch on ...70 2023 VCT LOCK//L...YouTube Music7 Best Books for Com...Restore SessionYouTube Music




Color

Selected Folders: datatest


SearchShow Result




Similarity: 90%




Similarity: 82%



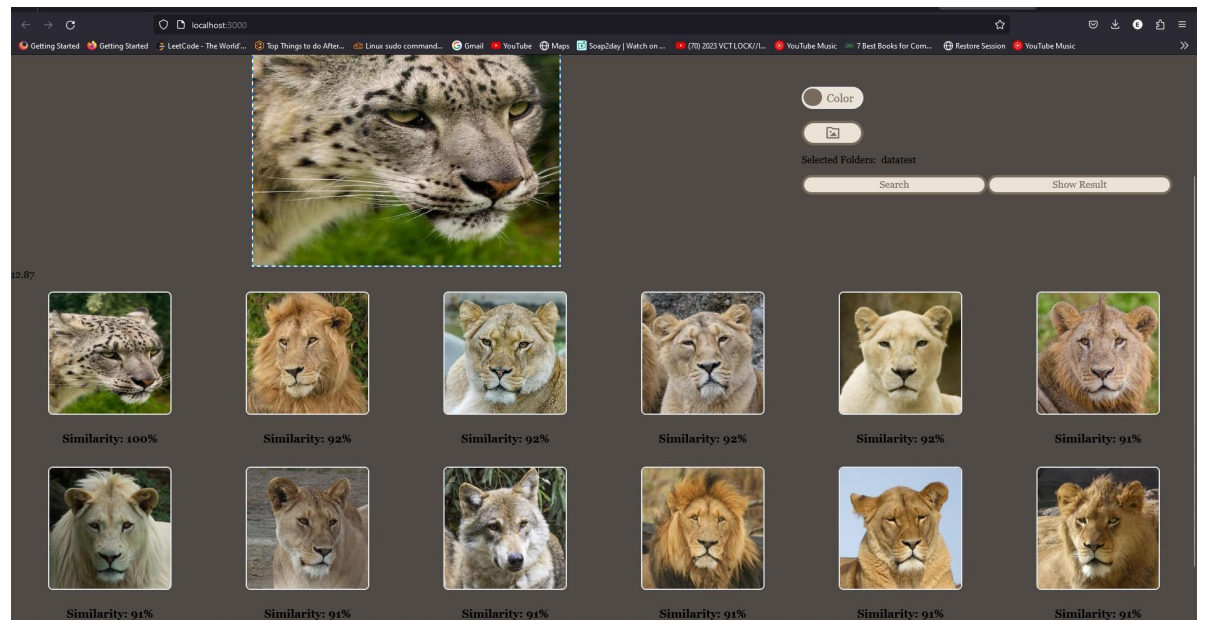
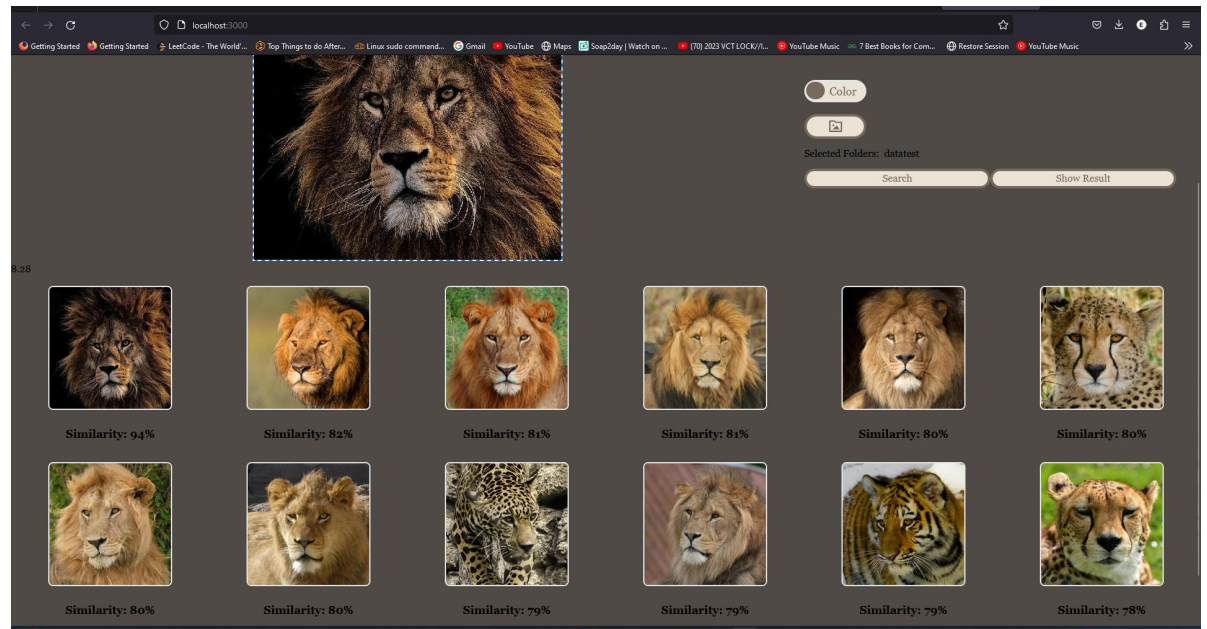
Similarity: 74%



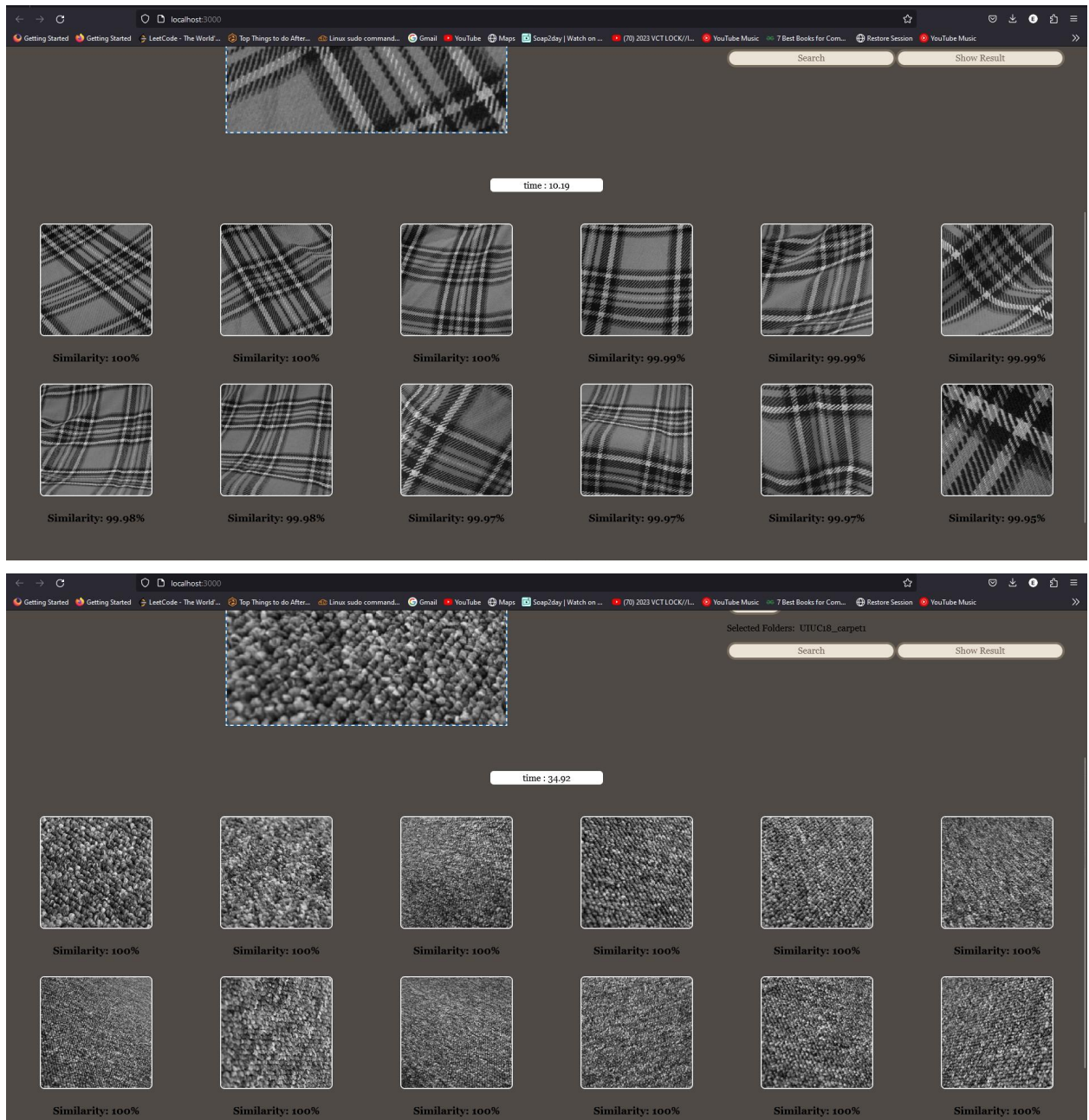
Similarity: 74%



Similarity: 65%



B. CBIR Tekstur



V. Analisis Desain Solusi Algoritma

Berdasarkan hasil pengujian, tercatat bahwa rata-rata waktu pencarian menggunakan parameter warna sebesar 10.58, lebih cepat dibandingkan waktu pencarian dengan tekstur yaitu . Tetapi, bisa diamati bahwa hasil pencarian menggunakan tekstur lebih akurat dibandingkan warna.

Hal ini diakibatkan oleh adanya 6 buah komponen pada vektor yang dibandingkan dalam metode tekstur, meliputi *energy*, *contrast*, *homogeneity*, *entropy*, *dissimilarity*, dan *ASM*. Sedangkan pada implementasi metode warna, hanya terdapat 3 komponen untuk dibandingkan, yaitu nilai *Hue*, *Saturation*, dan *Value*. Terlebih dengan adanya pembagian gambar pada metode warna menjadi blok 4x4 turut menurunkan akurasi karena signifikansi setiap *pixel* berkurang setelah nilai HSV nya dirata-ratakan.

Namun, blok-blok tersebutlah yang justru memberikan optimasi pada metode warna. Hal ini diakibatkan karena proses perbandingan gambar dengan teorema *Cosine Similarity* berkurang sebesar 16 kalinya. Tidak hanya itu, pada implementasi metode warna kami turut menggunakan *library python* yaitu *numpy*, yang ditulis menggunakan bahasa C. Karena bahasa C sendiri memiliki efektifitas kecepatan yang tinggi, kalkulasi menggunakan *array numpy* pun turut memiliki *runtime* yang sangat cepat.

BAB V

I. Kesimpulan

Implementasi dari *website* yang dibuat dapat dimanfaatkan untuk menemukan suatu gambar yang relevan dari suatu dataset. Dari implementasi solusi algoritma, diperoleh bahwa *Content Based Image Retrieval (CBIR)* menggunakan parameter warna dapat melakukan pencarian lebih cepat, namun tidak seakurat pencarian dengan parameter tekstur.

II. Saran

Terdapat beberapa hal yang kami sadari dapat memudahkan kami dalam pembuatan *website* ini, diantaranya adalah:

- Struktur dari web dapat dibuat lebih rapi dan sistematis, sehingga memudahkan pengembang dalam pembuatan program maupun saat proses awakutu.
- Sebaiknya memperdalam pemahaman mengenai suatu bahasa atau *framework* yang baru dipelajari terlebih dahulu sehingga tidak mengakibatkan kesalahan logika pada program nantinya.
- Untuk mendukung kelancaran pengembangan web, dan menghindari permasalahan pada *github* diperlukan *workflow* antar pengembang yang terstruktur.

III. Komentar/Tanggapan

“Sudah menyiapkan prom*g sesuai spesifikasi”

“Siap menyusul kuis”

“Waduh conflict mulu”

IV. Refleksi

Kami menyadari bahwa komunikasi dan kerjasama selama pengerjaan tugas besar ini masih belum maksimal. Adanya jadwal yang padat juga *workflow* yang kurang mendukung mengakibatkan berbagai hambatan selama pengerjaan. Selain itu, kurangnya pengalaman dalam pengembangan web juga menjadi masalah yang besar bagi kami, sehingga kami perlu usaha yang lebih keras lagi dalam menyelesaikan tugas besar ini. Namun, kami bersyukur bahwa tugas ini dapat menjadi suatu pengalaman baru dan

wadah untuk bereksplorasi lebih. Semoga tugas besar ini dapat memberikan manfaat sebanyak-banyaknya

V. Perbaikan dan Pengembangan

Untuk web ini, masih terdapat ruang yang bisa dikembangkan di lain hari, antara lain:

- *User Interface Website* dapat dibuat lebih menarik dan intuitif bagi pengguna.
- Program masih dapat dioptimasi dengan berbagai metode seperti *caching* dan mengurangi permintaan *server*.
- Menambahkan fitur pencarian lanjutan untuk meningkatkan kemudahan pengguna dalam menemukan konten.
- Menambahkan fitur yang tertulis pada spesifikasi bonus seperti *web scraping* dan input gambar melalui *webcam*.
- Menambahkan elemen-elemen interaktif seperti animasi atau fitur *drag-and-drop* untuk meningkatkan keterlibatan pengguna.

REFERENSI

1. <https://prism.ucalgary.ca/server/api/core/bitstreams/8f9de234-cc94-401d-b701-f08ceee6cfd/c/content>
2. Spesifikasi tubes algeo 2
3. <https://stackoverflow.com/>
4. <https://github.com/Farhannr28/Algeo02-22037>