

LAPORAN TUGAS BESAR II

IF2123 ALJABAR LINEAR DAN GEOMETRI

Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar



Disusun oleh:

Dewantoro Triatmojo (13522011)

Ahmad Hasan Albana (13522041)

Haikal Assyauqi (13522052)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2023

DAFTAR ISI

BAB I	1
I. Tujuan	1
II. Spesifikasi Tugas	1
BAB II	2
I. Dasar Teori	2
1. CBIR (Content Based Image-Retrieval)	2
2. RGB	2
3. HSV	2
4. Grayscale	4
5. Histogram Warna	4
6. Matriks Co Occurrence	4
7. Cosine Similarity	6
II. Pengembangan Website	6
BAB III	8
I. Langkah Pemecahan Masalah	8
II. Proses Pemetaan Masalah Menjadi Elemen Aljabar Geometri	11
III. Contoh Ilustrasi Kasus dan Penyelesaiannya	12
BAB IV	13
I. Implementasi Program Utama	13
II. Struktur Program	24
III. Tata Cara Penggunaan Program	25
IV. Hasil Pengujian	27
V. Analisis	35
BAB V	36
I. Kesimpulan	36
II. Saran	36
III. Refleksi	36
DAFTAR PUSTAKA	37

BAB I

DESKRIPSI MASALAH

I. Tujuan

Tujuan tugas besar ini adalah sebagai berikut:

1. Membuat *website* yang dapat menerima gambar dan menampilkan hasil akhir beserta persentasenya.
2. Menerapkan CBIR baik berupa color maupun texture untuk mencari kemiripan menggunakan *cosine similarity*.
3. Mampu mengkonversi nilai RGB ke berbagai format warna seperti HSV dan RGB.

II. Spesifikasi Tugas

1. Program menerima input *folder dataset* dan sebuah citra gambar.
2. *Dataset* gambar dapat diunduh secara mandiri melalui pranala [berikut](#). Akan tetapi peserta diperbolehkan untuk menggunakan *dataset* lain yang telah dipersiapkan oleh kelompok masing-masing.
3. Program menampilkan gambar citra gambar yang dipilih oleh pengguna.
4. 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.
5. Program mulai melakukan perhitungan nilai kecocokan antara *image* masukan dengan *dataset image* berdasarkan parameter yang telah dipilih (warna atau tekstur).
6. Program dapat menampilkan nilai kecocokan antara *image* masukan dengan setiap gambar dalam dataset.
7. 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.
8. 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.
9. Program dapat menampilkan **Jumlah gambar** yang memenuhi kondisi tingkat kemiripan serta **waktu eksekusi**.

BAB II

LANDASAN TEORI

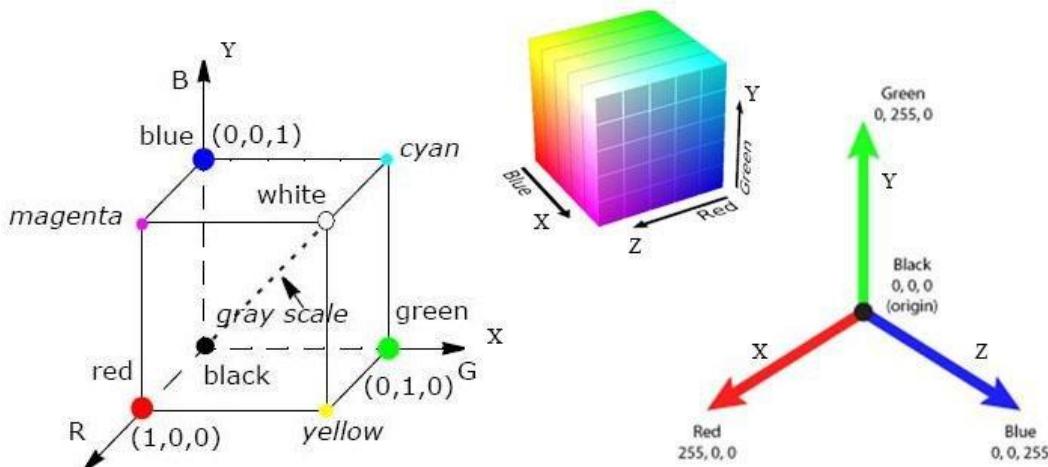
I. Dasar Teori

1. CBIR (Content Based Image-Retrieval)

CBIR atau Content-Based Image Retrieval merupakan sistem pengambilan informasi yang menggunakan karakteristik visual dari sebuah gambar untuk melakukan pencarian dan pengambilan gambar yang serupa dari basis data yang besar. Pada tugas kali ini, aspek dari CBIR yang digunakan antara lain, CBIR warna dan CBIR tekstur, CBIR seringkali digunakan untuk analisis statistik.

2. RGB

RGB merupakan model warna dasar dalam dunia digital, yakni merah (Red), hijau (Green), biru (Blue), warna dalam model RGB merupakan penggabungan 3 intensitas warna di atas, setiap warna memiliki intensitas 0 sampai 255. Kombinasi ini yang membentuk berbagai macam warna, dalam mencari kemiripan warna, setiap nilai RGB dalam bentuk vektor 3 dimensi, yang mana sumbu X,Y,Z, yang mewakili merah, hijau, dan biru.



Gambar 2.1 Vektor RGB

3. HSV

HSV merupakan singkatan dari Hue, Saturation, dan Value, Hue merupakan atribut yang menggambarkan jenis warna seperti merah, hijau, biru, Saturation merupakan atribut yang menggambarkan murni pekatnya suatu warna. Semakin tinggi

nilai saturasi, semakin jauh warna tersebut dari warna putih. Dan value, menunjukkan kecerahan atau kegelapan warna. Semakin tinggi nilai value, semakin terang warna tersebut.

Dalam CBIR warna, RGB diproses terlebih dahulu untuk mendapat nilai HSV, rumusnya yaitu:

1. Ubah nilai RGB dari [0,255] menjadi [0,1]

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

2. Cari C_{max} , C_{min} , dan Δ

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

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

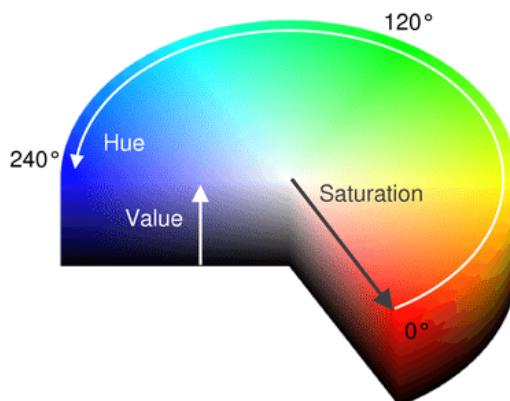
$$\Delta = C_{max} - C_{min}$$

3. Dapatkan nilai HSV

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

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

$$V = C_{max}$$



Gambar 2.2 Diagram HSV

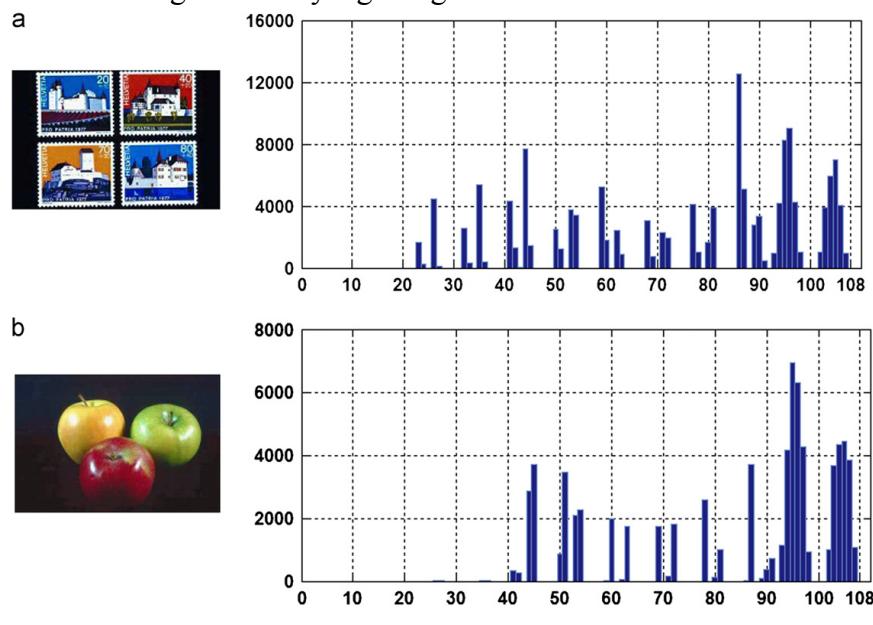
4. Grayscale

Grayscale merujuk pada representasi warna di mana setiap piksel dalam gambar direpresentasikan dalam skala keabuan. Artinya, setiap piksel memiliki nilai kecerahan tunggal yang berkisar dari hitam hingga putih, tanpa warna. Setiap nilai piksel mewakili tingkat keabuan yang berbeda, di mana 0 mewakili hitam, 255 mewakili putih, dan nilai-nilai di antaranya mewakili berbagai tingkat abu-abu di antara keduanya. Gambar grayscale tidak memiliki warna, hanya berisi informasi kecerahan. Teknik ini umumnya digunakan dalam fotografi, pengolahan citra, dan grafika komputer untuk berbagai tujuan, seperti mengurangi ukuran file, analisis, atau efek artistik. Rumus grayscale pada 1 pixel RGB yaitu:

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

5. Histogram Warna

Histogram warna adalah representasi grafis dari distribusi warna dalam sebuah gambar. Ini memperlihatkan seberapa banyak piksel dalam gambar memiliki intensitas warna tertentu. Histogram warna sangat berguna dalam pengolahan gambar, fotografi, dan analisis gambar. Mereka membantu pengguna untuk memahami distribusi tonal dalam gambar dan membuat penyesuaian warna yang diperlukan, memastikan gambar memiliki keseimbangan warna yang diinginkan.



Gambar 2.3 Histogram Warna

6. Matriks Co Occurrence

Matriks co-occurrence (CM) atau matriks kemunculan bersama adalah alat analisis statistik yang digunakan dalam pemrosesan gambar atau analisis citra. Matriks ini

menggambarkan seberapa sering pasangan piksel tertentu muncul bersama dalam suatu arah, baik secara horizontal, vertikal, maupun diagonal dalam citra. Matriks co-occurrence biasanya dibangun dari gambar dengan mengevaluasi hubungan spasial antara piksel. Untuk setiap pasangan piksel dalam gambar, matriks co-occurrence mencatat frekuensi kemunculan pasangan piksel tertentu dengan jarak dan arah tertentu. Misalnya, jika kita melihat pasangan piksel yang sama muncul secara horizontal satu sama lain dalam jarak satu piksel, matriks co-occurrence akan merekam informasi ini. Matriks co-occurrence sering digunakan dalam analisis tekstur citra untuk mengekstrak informasi tentang tekstur gambar, seperti kasar halusnya, arah garis, atau pola tertentu. Dengan menganalisis matriks co-occurrence, fitur-fitur tekstur seperti kontras, homogenitas, atau energi dari citra dapat diekstrak untuk digunakan dalam pengenalan pola atau analisis gambar. Untuk menemukan matriks GLCM

Karena hal inilah matriks Co occurrence digunakan untuk CBIR tekstur. Dari matriks co-occurrence kita akan mendapatkan 4 komponen penting menurut Newsam dan Kammath (2005) yakni *contrast, entropy, homogeneity, energy*:

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:

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

ASM:

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

Energy:

$$\sqrt{ASM}$$

7. Cosine Similarity

Cosine similarity ukuran yang digunakan untuk menentukan seberapa mirip dua vektor non-nol. Ini mengukur cosinus dari sudut antara kedua vektor, di mana nilai 1 menunjukkan kesamaan sempurna, 0 berarti tidak ada kesamaan..

Rumus cosine similarity:

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

II. Pengembangan Website

Website ini merupakan website yang dapat menampilkan gambar dataset yang paling mirip dengan gambar query dengan menggunakan CBIR warna dan CBIR tekstur. Program sistem temu balik gambar ini menggunakan website yang menggunakan bahasa pemrograman TypeScript, program ini memiliki bagian *frontend* dan *backend*, *frontend* tampilan yang dilihat oleh pengguna, pada web ini terdapat landing page, search menu, search input url, search input folder, about, dan how to use, sementara *backend* merupakan algoritma yang mengolah data inputan pengguna, inputan pengguna dapat berupa gambar dengan format file .jpg, .jpeg, .png, and .webp, jika input yang dimasukkan bukan gambar dengan 4 format di atas, akan diberikan pesan “Only these types are allowed .jpg, .jpeg, .png, .webp”.

Website ini merupakan website yang dapat menampilkan gambar dataset yang paling mirip dengan gambar query dengan menggunakan CBIR warna dan CBIR tekstur.

Beberapa tampilan frontend:

HBD

Search How To Use ☰

Choose compare methods!

You can choose between multiple methods to compare input image to a dataset!

 **Upload Dataset**
Compare dataset by uploading dataset folder

 **Scrape Dataset**
Compare dataset by scraping images from the web

Copyright © 2023 About

HBD Search How To Use ☰

Search Your Image Easily.

Here at HBD Lens, you can upload a dataset of images and a query image to find the most similar photo from your dataset with your query image!

[Try now →](#)



Copyright © 2023 About

HBD Search How To Use ☰

Reverse Image Search By Upload Data Set

Image Query

Choose File: No file chosen
OR CAMERA (10 S COUNTDOWN)


Calculation Method

Color Texture

 [Search](#)

Image Dataset

Choose File: No file chosen

Copyright © 2023 About

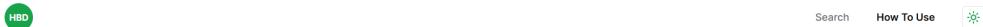


Reverse Image Search by Scrape Data Set

A screenshot of the "Image Query" section. It includes a large input field for an image file, a camera capture button, and a "Search" button. Below these are sections for "Choose File" (with a "No file chosen" message), "OR CAMERA (10 S COUNTDOWN)", and "Calculation Method" (with "Color" and "Texture" options).A screenshot of the "Image Data Set Link" section, which contains a single input field labeled "Enter image data set link to scrape".

Copyright © 2023

About

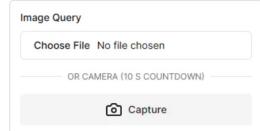


How To Use HBD Lens

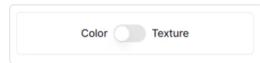
1. Choose option to compare query with uploading a dataset or scraping a dataset from a link.



2. Upload your input/query image to compare with dataset or you can capture an image using your webcam.



3. Choose to compare by color or by texture.



About HBD Lens

Welcome to HBD Lens, a cutting-edge project crafted by the ingenious minds of Haikal, Bana, and Dewo (Hence called HBD) from the Bandung Institute of Technology. This was made to fulfill the final task of linear algebra and geometry course. This project, a reverse image query utilizing Content-Based Image Retrieval (CBIR) technology. This project is fully open source and can be accessed by [this link](#).



BAB III

ANALISIS PEMECAHAN MASALAH

I. Langkah Pemecahan Masalah

1. CBIR Color

Langkah dalam mencari kemiripan dengan CBIR color yaitu:

- Nilai RGB dinormalisasi dengan range [0,255] menjadi [0,1]

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

- Cari CMax, Cmin, dan Δ

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

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

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

- Gunakan hasil perhitungan untuk mendapatkan nilai HSV

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

- Tiap nilai HSV akan dikuantifikasi dengan ketentuan berikut, sehingga terdapat 72 kemungkinan
- Blok gambar menjadi 4x4 dengan 4 blok di tengah memiliki bobot lebih tinggi, sehingga perhitungan lebih berfokus pada objek yang di tengah, dengan proses di bagian (d) maka akan terbentuk matriks matriks 4x4 yang berisi array dengan panjang 72 yang akan digunakan untuk mencari persamaan.
- Gunakan cosine similarity untuk menentukan kemiripan tiap blok

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

- g. Lalu tentukan rata-rata dari 16 blok dengan rumus pembobotan dimana 4 blok ditengah bobotnya dua kali bobot yang 12 blok yang diluar.

$$\bar{x} = \frac{\sum_{i=1}^n x_i \cdot w_i}{\sum_{i=1}^n w_i}$$

2. CBIR Tekstur

Langkah dalam menentukan CBIR tekstur yaitu:

- a. Konversi warna gambar menjadi *grayscale*, ini dilakukan karena warna tidaklah penting dalam penentuan tekstur. Oleh karena itu, warna RGB dapat diubah menjadi suatu warna *grayscale* Y dengan rumus:

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

- b. Dari RGB yang telah dikonversi menjadi grayscale, buatlah matriks GLCM yang berukuran 256 x 256. Untuk pemilihan sudut digunakan sudut 0 derajat, sementara untuk distance dipilih sebesar 1 untuk meningkatkan akurasi dan presisi.
- c. Tentukan transpose dari matriks GLCM dan jumlahkan transpose dengan matriks GLCM hingga menjadi matriks simetris

1	1	0	0
0	0	1	0
0	0	0	2
0	0	1	0

GLCM Matrix

1	0	0	0
1	0	0	0
0	1	0	1
0	0	2	0

Transpose GLCM Matrix

2	1	0	0
1	0	1	0
0	1	0	3
0	0	3	0

Matrix Symetric

- d. Cari nilai *normalization matrix* dari matriks simetris tadi dengan rumus berikut

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

- e. Dari *co-occurrence matrix* bisa diperoleh komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*, *energy*. Persamaan yang dapat digunakan untuk mendapatkan nilai 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 :

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

Energy:

$$\sqrt{ASM}$$

Yang mana rumus ASM :

$$\sum_{i,j=0}^{levels-1} P_{i,j}^2$$

- f. Dari nilai-nilai komponen tersebut, tentukan nilai dari Gaussian Normalizationnya untuk menormalisasi nilai.

$$h^{i,j'} = \frac{h^{i,j} - m_j}{\sigma_j}$$

• • • •

- g. Ukurlah kemiripan dengan menggunakan Cosine Similarity

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

II. Proses Pemetaan Masalah Menjadi Elemen Aljabar Geometri

1. CBIR Warna

- a. Titik dalam Ruang Warna

Setiap piksel dalam gambar memiliki baik nilai RGB maupun nilai HSV, masing-masing komponen baik red,green,blue maupun hue,saturation, value dianggap sebagai koordinat vektor 3 dimensi

- b. Operasi Aljabar pada Vektor Warna

Dalam konversi RGB menjadi HSV banyak operasi aljabar yang digunakan baik penjumlahan, perkalian, pengurangan dan lainnya

- c. Kuantifikasi Kemunculan HSV

Setiap piksel memiliki memiliki indeks berbeda pada kuantifikasi HSV dan blok yang berbeda sehingga dimasukkan dalam matriks 4 x 4 yang memiliki array dengan panjang 72.

- d. *Cosine Similarity*

Metode pengukuran untuk menentukan kemiripan

2. CBIR Tekstur

- a. Konversi RGB menjadi Grayscale
Perkalian vektor dengan suatu konstanta.
- b. Symmetric GLCM.
Usai mendapat matriks GLCM, matriks GLCM akan ditranspose.
- c. Normalisasi GLCM
Pada normalisasi GLCM banyak digunakan penjumlahan dan pembagian matriks.
- d. 4 Komponen Ekstraksi Tekstur
Terdiri dari *entropy*, *contrast*, *energy*, dan *homogeneity*.
- e. *Cosine Similarity*
Metode pengukuran untuk menentukan kemiripan.

III. Contoh Ilustrasi Kasus dan Penyelesaiannya

1. CBIR Warna

Kasus : Ingin membandingkan 2 gambar.

Penyelesaian

- Setiap citra gambar diubah dari RGB menjadi HSV.
- Histogram warna lokal dihitung untuk masing-masing image. Lokal histogram digunakan karena memiliki akurasi lebih baik dibanding global.
- Kemiripan kedua image dihitung dengan cosine similarity.
- Semakin tinggi nilai cosine similarity, semakin mirip.

2. CBIR Tekstur

Kasus: Mendapatkan gambar tekstur paling mirip.

Penyelesaian

- Mengubah format warna gambar dari RGB menjadi grayscale.
- Lakukan identifikasi pada citra yang diubah pada grayscale.
- Cari GLCM dan normalize matrix dari citra grayscale.
- Hitung komponen tekstur dari GLCM, yakni contrast, homogeneity, entropy, energy.
- Hitung query image dengan dataset menggunakan cosine similarity.
- Akan ditampilkan gambar dengan cosine similarity tertinggi.

BAB IV

IMPLEMENTASI

DAN UJI COBA

I. Implementasi Program Utama

Program sistem temu balik gambar ini menggunakan website yang menggunakan bahasa pemrograman TypeScript, program ini memiliki bagian *frontend* dan *backend*, *frontend* tampilan yang dilihat oleh pengguna, sementara *backend* merupakan algoritma yang mengolah data inputan pengguna. Website ini merupakan website yang dapat menampilkan gambar dataset yang paling mirip dengan gambar query dengan menggunakan CBIR warna dan CBIR tekstur.

Untuk mendapatkan nilai kemiripan dari CBIR warna dan CBIR tekstur, maka terdapat beberapa fungsi yang digunakan yang dapat dicek di fil *src/lib/*, di antaranya sebagai berikut:

1. Penyelesaian CBIR Warna

a. solveCBIRColor

```
export const solveCBIRColor = async (
  imageQuery: Buffer,
  imageDataSet: Buffer[]
): Promise<CBIRCalculationResult> => {
  // Convert input image (file) to hsv
  const inputImageData = await convertBufferToHSVMatrix(imageQuery);
  console.log(inputImageData);
  // Get feature vector
  const inputImage16FeatureVector = getColor16FeatureVector(inputImageData);
  console.log(inputImage16FeatureVector);
  // Get weighted block
  // Blok 6, 7, 10, 11 memiliki bobot 2
  // Blok lainnya memiliki bobot 1
  // prettier-ignore
  const weightedBlock = [
    1, 1, 1, 1,
    1, 2, 2, 1,
    1, 2, 2, 1,
    1, 1, 1, 1
  ];

  // Compare to dataset
  const compareResult = await Promise.all(
    imageDataSet.map(async (image, index) => {
      // Convert to hsv
      const dataSetImageData = await convertBufferToHSVMatrix(image);

      // Get feature vector
      const dataSetImage16FeatureVector =
        getColor16FeatureVector(dataSetImageData);

      // Get similarity each block
      const blockSimilarities: number[] = [];

      for (let i = 0; i < 16; i++) {
        const similarity = getSimiliarity(
          inputImage16FeatureVector[i],
          dataSetImage16FeatureVector[i]
        );

        blockSimilarities.push(similarity);
      }

      // Get weighted average
      // Blok 6, 7, 10, 11 memiliki bobot 2
      // Blok lainnya memiliki bobot 1
      const meanSimilarity = getWeightedMeanFromArr(
        blockSimilarities,
        weightedBlock
      );
      // To reduce load in data transmission, only return the index in the original dataset array.
      return {
        index,
        similarity: meanSimilarity,
      };
    })
  );

  // Filter out similarity
  const filteredCompareResult = compareResult.filter(
    (result) => result.similarity > 0.6
  );
}
```

b. convertRGBToHSV

Mengubah nilai RGB menjadi HSV

```
export const convertRGBToHSV = (rgb: RGB): HSV => {
  const [r, g, b] = rgb;

  const rPrime = r / 255;
  const gPrime = g / 255;
  const bPrime = b / 255;

  const cMax = Math.max(rPrime, gPrime, bPrime);
  const cMin = Math.min(rPrime, gPrime, bPrime);
  const delta = cMax - cMin;

  let h: number;
  if (delta === 0) {
    h = 0;
  } else if (cMax === rPrime) {
    h = 60 * (((gPrime - bPrime) / delta) % 6);
  } else if (cMax === gPrime) {
    h = 60 * ((bPrime - rPrime) / delta + 2);
  } else {
    h = 60 * ((rPrime - gPrime) / delta + 4);
  }

  let s: number;
  if (cMax === 0) {
    s = 0;
  } else {
    s = delta / cMax;
  }

  let v = cMax;

  return [h, s, v];
};
```

c. convertBufferToHSVMatrix

Mengubah matriks RGB dari sebuah gambar menjadi matriks HSV

```
export const convertBufferToHSVMatrix = async (
  buffer: Buffer
): Promise<ImageData<HSV>> => {
  // Convert file to raw image data
  // Must use failOn: "error" to handle "VipsJpeg: Invalid SOS parameters for
  const { data, info } = await sharp(buffer, { failOn: "error" })
  .raw()
  .toBuffer({ resolveWithObject: true });
  const pixelArray = new Uint8ClampedArray(data.buffer);
  const channels = info.channels;
  const width = info.width;
  const height = info.height;

  // Get HSV matrix (first from RGB, then convert each RGB to HSV)
  const hsvMatrix: ImageMatrix<HSV> = [];

  for (let i = 0; i < height; i++) {
    const row: HSV[] = [];
    for (let j = 0; j < width; j++) {
      const index = (i * width + j) * channels;

      const r = pixelArray[index];
      const g = pixelArray[index + 1];
      const b = pixelArray[index + 2];

      const pixel: HSV = convertRGBToHSV([r, g, b]);

      row.push(pixel);
    }
    hsvMatrix.push(row);
  }

  const imageData: ImageData<HSV> = {
    width,
    height,
    matrix: hsvMatrix,
  };

  return imageData;
};
```

d. getColor16FeatureVector

Matriks HSV dikonversi ke dalam sebuah matriks 4x4 dimana setiap elemennya adalah feature vektor tiap blok yaitu array sepanjang 72.

```
export const getColor16FeatureVector = (
  imageDataHSV: ImageData<HSV>
): number[][] => {
  // Bagi image menjadi 4x4 blok
  // Setiap blok memiliki 8x3x3 bin
  // Inisialisasi array untuk menyimpan 16 feature vector (1 untuk
  const featureVectors16 = initMatrix(16, 72);

  for (let i = 0; i < imageDataHSV.height; i++) {
    for (let j = 0; j < imageDataHSV.width; j++) {
      const [h, s, v] = imageDataHSV.matrix[i][j];

      // Get block index
      const blockI = Math.floor((4 * i) / imageDataHSV.height);
      const blockJ = Math.floor((4 * j) / imageDataHSV.width);
      const blockIndex = blockI * 4 + blockJ;

      // H
      // 0 => 316, 360
      // 1 => 1, 25
      // 2 => 26, 48
      // 3 => 41, 128
      // 4 => 121, 190
      // 5 => 191, 270
      // 6 => 271, 295
      // 7 => 295, 315
      // Hue 0 = Hue 360 (Circular)
      let hIndex;
      if ((h > 316.5 && h <= 360) || h == 0) {
        hIndex = 0;
      } else if (h > 0 && h <= 25.5) {
        hIndex = 1;
      } else if (h > 25.5 && h <= 40.5) {
        hIndex = 2;
      } else if (h > 40.5 && h <= 120.5) {
        hIndex = 3;
      } else if (h > 120.5 && h <= 190.5) {
        hIndex = 4;
      } else if (h > 190.5 && h <= 270.5) {
        hIndex = 5;
      } else if (h > 270.5 && h <= 295.5) {
        hIndex = 6;
      } else {
        hIndex = 7;
      }
    }
  }
}
```

e. Cosine Similarity

```
export const getSimiliarity = (A: number[], B: number[]): number => {
  // Precondition: A.length === B.length
  const n = A.length;

  let sumAB = 0;
  let sumA2 = 0;
  let sumB2 = 0;

  for (let i = 0; i < n; i++) {
    sumAB += A[i] * B[i];
    sumA2 += A[i] * A[i];
    sumB2 += B[i] * B[i];
  }

  const sim = sumAB / (Math.sqrt(sumA2) * Math.sqrt(sumB2));
  return sim;
};
```

2. Penyelesaian CBIR Tekstur

a. convertBufferGrayMatrix

Mengubah nilai matriks RGB dari sebuah gambar menjadi matriks grayscale

```
export const convertBufferGrayMatrix = async (
  buffer: Buffer
): Promise<ImageData<Gray>> => {
  // Convert file to raw image data
  // Must use failOn: "error" to handle "VipsJpeg: Invalid SOS parameters"
  const { data, info } = await sharp(buffer, { failOn: "error" })
    .raw()
    .toBuffer({ resolveWithObject: true });

  const pixelArray = new Uint8ClampedArray(data.buffer);

  const channels = info.channels;
  const width = info.width;
  const height = info.height;

  // Get Gray matrix
  const grayMatrix: ImageMatrix<Gray> = [];

  for (let i = 0; i < height; i++) {
    const row: Gray[] = [];
    for (let j = 0; j < width; j++) {
      const index = (i * width + j) * channels;

      const r = pixelArray[index];
      const g = pixelArray[index + 1];
      const b = pixelArray[index + 2];

      const rgbPixel: RGB = [r, g, b];
      const grayPixel: Gray = convertRGBtoGray(rgbPixel);

      row.push(grayPixel);
    }
    grayMatrix.push(row);
  }

  const imageData: ImageData<Gray> = {
    width,
    height,
    matrix: grayMatrix,
  };

  return imageData;
};
```

b. convertRGBToGray

Mengkonversi nilai RGB menjadi nilai grayscale

```
/* COLOR CONVERSION */
export const convertRGBtoGray = (rgb: RGB): Gray => {
  const [r, g, b] = rgb;

  return Math.round(0.29 * r + 0.587 * g + 0.114 * b);
};
```

c. getNormalizedGLCM

Membentuk matriks normalized

```
/* Function to get Normalized GLCM */
export const getNormalizedGLCM = (grayImageData: ImageData<Gray>): GLCM => {
  // Initialize symmetric glcm
  const symmetricGLCM = initMatrix(256, 256);

  // Get symmetric glcm
  for (let i = 0; i < grayImageData.height; i++) {
    for (let j = 0; j < grayImageData.width; j++) {
      if (j != 0) {
        const firstGrayElement = grayImageData.matrix[i][j - 1];
        const secondGrayElement = grayImageData.matrix[i][j];

        symmetricGLCM[firstGrayElement][secondGrayElement] += 1;
        symmetricGLCM[secondGrayElement][firstGrayElement] += 1;
      }
    }
  }

  // Normalize glcm
  const normalizedGLCM = getNormalizeMatrix(symmetricGLCM);

  return normalizedGLCM;
};
```

d. getImageTextureFeatureVector

Mendapat nilai contrast, homogeneity, dan entropy

```
export const getImageTextureFeatureVector = (
  normalizedGLCM: GLCM
): number[] => {
  let contrast = 0;
  let homogeneity = 0;
  let entropy = 0;
  let ASM = 0;

  // Get feature vector
  for (let i = 0; i < 256; i++) {
    for (let j = 0; j < 256; j++) {
      // Contrast
      contrast += normalizedGLCM[i][j] * (i - j) * (i - j);

      // Homogeneity
      homogeneity += normalizedGLCM[i][j] / (1 + (i - j) * (i - j));

      // ASM
      ASM += Math.pow(normalizedGLCM[i][j], 2);
    }
  }

  // energy
  const energy = Math.sqrt(ASM);

  // feature vector
  const featureVector: number[] = [contrast, homogeneity, entropy, energy];

  // Get gaussian normalization
  const mean = getMeanFromArr(featureVector);
  const standardDeviation = getStandardDeviationFromArr(featureVector);
  const gaussianNormalization = featureVector.map((value) => {
    return (value - mean) / standardDeviation;
  });

  return gaussianNormalization;
};
```

e. solveCBIRTexture

```
/* Function to Solve CBIR by Texture wise */
export const solveCBIRTexture = async (
  imageQuery: Buffer,
  imageDataSet: Buffer[]
): Promise<CBIRCalculationResult> => {
  // Convert to co-occurrence Matrix
  const inputImageGrayData = await convertBufferGrayMatrix(imageQuery);

  // Get normalized GLCM
  const inputImageNormalizedGLCM = getNormalizedGLCM(inputImageGrayData);

  // Get input image texture vector
  const inputImageFeatureVector = getImageTextureFeatureVector(
    inputImageNormalizedGLCM
  );

  // Compare to dataset
  const compareResult = await Promise.all(
    imageDataSet.map(async (image, index) => {
      // Convert to gray matrix
      const dataSetImageData = await convertBufferGrayMatrix(image);

      // Get normalized glcm
      const dataSetImageFeatureVector = getNormalizedGLCM(dataSetImageData);

      // Get feature vector
      const dataSetFeatureVector = getImageTextureFeatureVector(
        dataSetImageFeatureVector
      );

      // Check similarity
      const similarity = getSimiliarity(
        inputImageFeatureVector,
        dataSetFeatureVector
      );

      // To reduce load in data transmission, only return the index in the original dataset array.
      return {
        index,
        similarity,
      };
    })
  );

  // Filter out similarity
  const filteredCompareResult = compareResult.filter(
    (result) => result.similarity > 0.6
  );

  // Sort by similarity
  filteredCompareResult.sort((a, b) => b.similarity - a.similarity);

  return filteredCompareResult;
};
```

3. Program Lain

a. Program di statistics.ts

```
export const initMatrix = (row: number, col: number): number[][] => {
  const matrix: number[][] = [...Array(row)].map(_ => Array(col).fill(0));

  return matrix;
};

// Get sum element of matrix
export const getSumElementMatrix = (matrix: number[][]): number => {
  let count = 0;

  // Sum matrix
  matrix.forEach((row) => {
    row.forEach((col) => {
      count += col;
    });
  });

  return count;
};

// Normalize matrix
export const getNormalizeMatrix = (matrix: number[][]): number[][] => {
  // Get total element
  const total = getSumElementMatrix(matrix);

  // Normalize matrix
  const normalizeMatrix = matrix.map((row) => {
    return row.map((col) => {
      return col / total;
    });
  });

  return normalizeMatrix;
};
```

b. Program di matrix.ts

```
export const getMeanFromArr = (arr: number[]): number => {
  const sum = arr.reduce((acc, curr) => acc + curr, 0);

  return sum / arr.length;
};

// Get standard deviation of array
export const getStandardDeviationFromArr = (arr: number[]): number => {
  const mean = getMeanFromArr(arr);

  const variance = arr.reduce((acc, curr) => {
    return acc + Math.pow(curr - mean, 2);
  }, 0);

  return Math.sqrt(variance / arr.length);
};

// Get weighted mean of array
export const getWeightedMeanFromArr = (
  arr: number[],
  weights: number[]
): number => {
  const sumWiXi = arr.reduce((acc, curr, i) => acc + curr * weights[i], 0);
  const sumWi = weights.reduce((acc, curr) => acc + curr, 0);

  return sumWiXi / sumWi;
};
```

c. Program di api/search/scrape

```
import { type NextRequest, NextResponse } from "next/server";
import { SearchByScrapeFormSchema } from "@/lib/zod";
import { JSDOM } from "jsdom";
import { convertFileToBuffer, solveCBIR } from "@/lib/cbir";
import { SuccessSearchByScrapeResponse } from "@/types/api";
import { allowedImagesTypes } from "@/lib/constants";

export const POST = async (req: NextRequest) => {
    // Get data
    const formData = await req.formData();
    const rawImageInput = formData.get("imageInput") as File;
    const rawIsTexture = JSON.parse(
        formData.get("isTexture") as string
    ) as boolean;
    const rawScrapeUrl = formData.get("scrapeUrl") as string;

    const rawDataObject = {
        imageInput: rawImageInput,
        isTexture: rawIsTexture,
        scrapeUrl: rawScrapeUrl,
    };

    // Validate data
    const zodParseResult = SearchByScrapeFormSchema.safeParse(rawDataObject);
    if (!zodParseResult.success) {
        return NextResponse.json(
            {
                error: "Bad Request",
                message: "Invalid data format",
            },
            { status: 400 }
        );
    }

    // Get validated data
    const { imageInput, isTexture, scrapeUrl } = zodParseResult.data;

    // Get html from url
    const res = await fetch(scrapeUrl);
    const html = await res.text();
    const dom = new JSDOM(html);

    // Get images from html
    const document = dom.window.document;
    const imageElements = Array.from(document.querySelectorAll("img"));
    const imageUrls = imageElements.map((element) => {
        const rawUrl = element.getAttribute("src") + "";
        const url = new URL(rawUrl, scrapeUrl).href;
        return url;
    });

    // Convert image input to buffer
    const imageInputBuffer = await convertFileToBuffer(imageInput);
```

```

// Convert scraped urls to buffers paralelly (faster than sequentially)
const processedImages = await Promise.all(
  imageUrls.map(async (url) => {
    const res = await fetch(url);

    const arrBuff = await res.arrayBuffer();
    const imageBuffer = Buffer.from(arrBuff);
    const contentType = res.headers.get("content-type") as string;
    const imageBase64 = imageBuffer.toString("base64");
    const imageSrc = `data:${contentType};base64,${imageBase64}`;

    return {
      imageBuffer,
      imageSrc,
      contentType,
    };
  })
);

// Filter only valid image types to be processed
// Note that zod only validates input query image and dataset url source,
// but not the images type itself
const filteredProcessedImages = processedImages.filter((image) =>
  allowedImagesTypes.includes(image.contentType)
);

// Get array of image buffers
const imageDataSetBuffers = filteredProcessedImages.map(
  (image) => image.imageBuffer
);

// Get image src
const imageDataSetSrcs = filteredProcessedImages.map(
  (image) => image.imageSrc
);

const CBIRResult = await solveCBIR(
  imageInputBuffer,
  imageDataSetBuffers,
  isTexture
);

// Return response
const imageResults: SuccessSearchByScrapeResponse = CBIRResult.map(
  (result) => {
    const { index, similarity } = result;
    const imageSrc = imageDataSetSrcs[index];

    return {
      imageSrc,
      similarity,
    };
  }
);

return NextResponse.json(imageResults);
};

```

d. Program di search/api/upload

```
import { type NextRequest, NextResponse } from "next/server";
import { convertFileToBuffer, solveCBIR } from "@/lib/cbir";
import { SearchByUploadHttpSchema } from "@/lib/zod";

export const POST = async (req: NextRequest) => {
    // Get data from form data
    const formData = await req.formData();
    const rawImageInput = formData.get("imageInput");
    const rawIsTexture = JSON.parse(formData.get("isTexture") as string);
    const rawImageDataset = formData.getAll("imageDataSet");

    const rawDataObject = {
        imageInput: rawImageInput,
        isTexture: rawIsTexture,
        imageDataSet: rawImageDataset,
    } as unknown;

    // Validate data
    const zodParseResult = SearchByUploadHttpSchema.safeParse(rawDataObject);
    if (!zodParseResult.success) {
        return NextResponse.json(
            { error: "Bad Request", message: "Invalid data format" },
            { status: 400 }
        );
    }

    // Get data
    const { imageInput, isTexture, imageDataSet } = zodParseResult.data;

    // Convert image input to buffer
    const imageInputBuffer = await convertFileToBuffer(imageInput);

    // Convert files to buffers paralelly (faster than sequentially)
    const imageDataSetBuffer = await Promise.all(
        imageDataSet.map(async (file) => await convertFileToBuffer(file))
    );

    // Process data
    const CBIRColorResult = await solveCBIR(
        imageInputBuffer,
        imageDataSetBuffer,
        isTexture
    );

    // Return response
    return NextResponse.json(CBIRColorResult);
};
```

II. Struktur Program

Untuk pembuatan website, untuk *frontend* dan *backend* kami menggunakan bahasa TypeScript, untuk memudahkan pemrosesan terdapat beberapa library yang digunakan, di antaranya:

1. Next.js - v14.0.1 (Fullstack Framework)
2. Nodejs - v20.9.0 (JavaScript Runtime)
3. TailwindCSS -v3.3.0 (CSS Library)

4. shadcn/ui - vN/A (UI Library)
5. sharp - v0.32.6 (Image Processing)
6. zod - v3.22.4 (Schema Validation)
7. react-hook-form - v(7.47.0) (Form Hooks)
8. react-pdf/renderer - v3.1.14 (Render PDF)
9. jsdom - v22.1.0 (Convert String to DOM)
10. react-webcam - v7.2.0 (Webcam Hooks)

Untuk frontend, mayoritas struktur kode dapat dilihat di folder app, components, dan public, di folder app terdapat page about, how to use, search, sementara di components, terdapat footer, navbar, pagination, folder ui berisi komponen-komponen shadcn/ui, dan di public terdapat gambar yang berfungsi untuk mempercantik tampilan website, sementara untuk algoritma perhitungan dapat dicek di folder lib, dengan fungsi lebih lanjut dapat dicek pada bagian Implementasi Program Utama. Dan untuk backend api endpoint terdapat di folder app/api/search. Untuk api endpoint upload dataset berada di app/api/search/upload dan untuk api endpoint scrape dataset berada di app/api/search/scrape.

III. Tata Cara Penggunaan Program

1. Ketika masuk ke landing page, silahkan klik search bagi pengguna laptop/notebook dan untuk pengguna mobile phone, bisa klik garis tiga di ujung kanan lalu menekan menu search.

Search Your Image Easily.

Here at HBD Lens, you can upload a dataset of images and a query image to find the most similar photo from your dataset with your query image!

Try now! →

Copyright © 2023 About

2. Di menu search, anda dapat memilih cara input gambar baik melalui folder maupun link website



Choose compare methods!

You can choose between multiple methods to compare input image to a dataset!



Copyright © 2023

About

3. Usai memilih salah satu cara input, masukkan foto yang akan menjadi *query* atau bisa juga menggunakan kamera untuk dijadikan *query*, lalu lanjutkan dengan menambahkan folder berisi gambar jika memilih menu input folder atau link web jika memilih menu web scraping

Two side-by-side screenshots of the reverse image search interface. Both have a header with 'Search', 'How To Use', and a gear icon. The left screenshot is titled 'Reverse Image Search by Scrape Data Set' and shows fields for 'Image Query' (choose file or camera), 'Calculator Method' (Color or Texture), and a 'Search' button. Below is a text input field for 'Image Data Set Link'. The right screenshot is titled 'Reverse Image Search By Upload Data Set' and shows similar fields but with 'Image Dataset' instead of 'Image Data Set Link'. Both screenshots include copyright information at the bottom: 'Copyright © 2023' and 'About'.

Input link web

Input folder

4. Pilih metode apa yang akan digunakan antara color atau texture, adan dapat mengubah dengan menekan toggle yang berada di bawah bagian input.
5. Tunggu sekitar 10-20 detik untuk foto sebanyak 500 gambar dan 160-180 detik untuk 5000 foto.
6. Anda dapat melihat hasil kemiripan secara berurut dari yang paling mirip hingga gambar terakhir yang memiliki kemiripan di atas atau sama dengan 60%.
7. Anda dapat mengunduh hasil pencarian dalam bentuk PDF.
8. Jika memiliki kebingungan dalam mengoperasikan web, anda bisa melihat menu how to use.

Search How To Use *

How To Use HBD Lens

- Choose option to compare query with uploading a dataset or scraping a dataset from a link.

[Upload Dataset](#) [Scrape Dataset](#)

- Upload your input/query image to compare with dataset or you can capture an image using your webcam.

Image Query
Choose File No file chosen
OR CAMERA (10 S COUNTDOWN)

- Choose to compare by color or by texture.

Color Texture

IV. Hasil Pengujian

1. Input CBIR Warna

- a. Input Folder Gambar Berwarna (Lebih dari 1000 Gambar)

Reverse Image Search By Upload Data Set

Image Query
Choose File 98.jpg
OR CAMERA (10 S COUNTDOWN)

Calculation Method
 Color Texture

Results: 309 Results in 91.15 Seconds

Result 1	Result 2	Result 3
 100.00%	 72.13%	 71.55%
 71.22%	 71.03%	 70.39%

<< < 1 2 3 4 5 ... 52 > >>

Image Dataset
Choose File 4738 files

b. Input Folder Gambar Berwarna (Kurang dari 500 Gambar)

Image Query

Choose File 77.jpg
OR CAMERA (10 S COUNTDOWN)

 Capture

Calculation Method

Color Texture

Search

Results: 37 Results in 8.51 Seconds

 100.00%	 70.72%	 68.08%
 67.12%	 66.88%	 66.51%

<< < **1** 2 3 4 5 6 7 > >>

Convert to PDF

Image Dataset

Choose File 429 files

c. Input Folder Gambar Monokrom

Image Query

Choose File 97.jpg

OR CAMERA (10 S COUNTDOWN)

Calculation Method

Color Texture

Results:

632 Results in 1.82 Seconds

100.00%

93.44%

93.29%

92.54%

92.23%

91.83%

<< < **1** 2 3 4 5 ... 106 > >>

Convert to PDF

Image Dataset

Choose File 685 files

d. Input dari Link Web

Reverse Image Search by Scrape Data Set

Image Query

Choose File mutiara_azzahra.jpg

OR CAMERA (10 S COUNTDOWN)

Calculation Method

Color Texture

Results: 30 Results in 10.49 Seconds

Image Data Set Link

<https://jkt48.com/member/list?lang=id>

e. Input Kamera

Reverse Image Search By Upload Data Set

Image Query

Choose File No file chosen

OR CAMERA (10 S COUNTDOWN)

Capture

Calculation Method

Color Texture

Search

Results:

6 Results in 20.60 Seconds

86.88%

85.99%

71.53%

70.99%

70.59%

69.69%

« < 1 > »

f. Output PDF

Reverse Image Search Result

Image Input:



Method:

Color

Results:

30 results in 10.49 seconds

[Data Set Source](#)



2. Input CBIR Tekstur

a. Input Folder

Reverse Image Search By Upload Data Set

Image Query

Choose File 9.jpg

OR CAMERA (10 S COUNTDOWN)

Capture

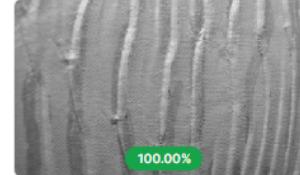
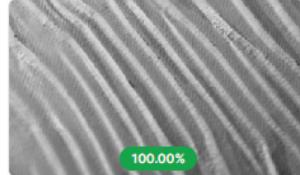
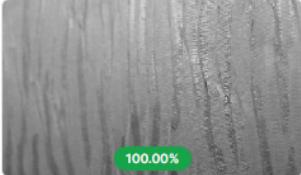
Calculation Method

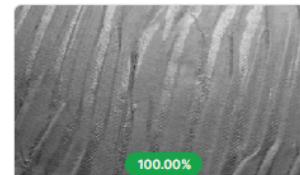
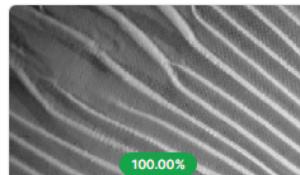
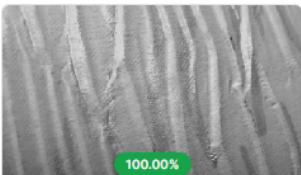
Color Texture

Search

Results:

20 Results in 0.56 Seconds



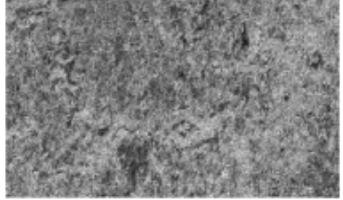


<< < 1 2 3 4 > >>

b. Output PDF

Reverse Image Search Result

Image Input:

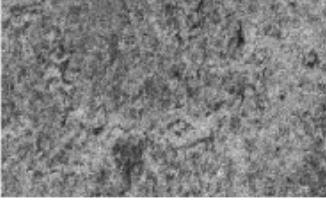
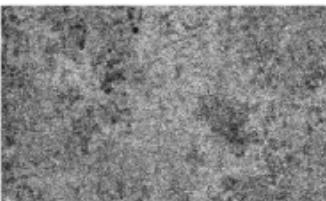
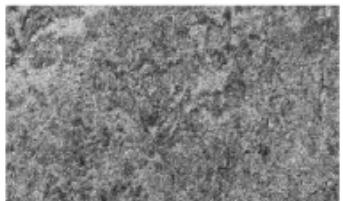
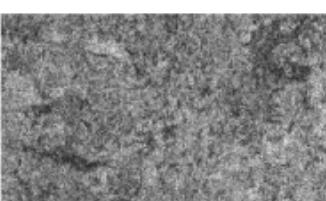
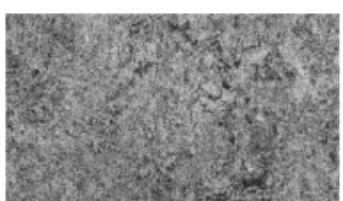


Method:

Texture

Results:

80 results in 0.58 seconds

Result Image	Similarity (%)
	100.00%
	100.00%
	100.00%
	100.00%
	100.00%
	100.00%
	100.00%
	100.00%
	100.00%

V. Analisis

Secara teoritis, CBIR warna akan bekerja sangat baik pada untuk menentukan kemiripan dengan konten berwarna jelas seperti, waktu (pag, siang, malam), warna kulit, dan lain-lain. Serta untuk CBIR tekstur berguna untuk konten yang memiliki pola serta bahan yang memiliki tekstur.

Dari hasil uji coba, pada CBIR warna, hasil yang didapat sangat akurat dikarenakan karena menggunakan histogram lokal yang akurasinya lebih baik, selain itu gambar dibagi menjadi 4×4 blok, dengan blok di tengah memiliki bobot lebih tinggi, sehingga objek memiliki faktor lebih tinggi dalam menentukan kemiripan dibanding latar, selain itu karena menggunakan bahasa pemrograman TypeScript proses ekstraksi gambar terbilang cukup cepat.

Namun, dari hasil uji coba CBIR tekstur, nilai kemiripan hanya berkisar di angka 95-100%, hal ini dikarenakan nilai dari contrast sangat mendominasi dibanding komponen lainnya karena contrast memiliki rumus yang bersifat kuadratik.

BAB V

SIMPULAN

I. Kesimpulan

1. Pembuatan website dapat dilihat pada bab 2 dan bab 3.
2. Penerapan CBIR diimplementasikan pada bab 3 dan bab 4.
3. Konversi RGB ke format warna lain terdapat pada bab 3 dan bab 4 .

II. Saran

1. Perbaikan pada algoritma CBIR tekstur karena hasilnya mirip-mirip.
2. Dapat menggunakan bahasa pemrograman dengan performa yang lebih bagus seperti go, cpp, maupun rust.

III. Refleksi

Dari tugas besar ini, kami belajar untuk mengembangkan sebuah web yang ternyata sangat sulit, dan kami harus belajar untuk manajemen waktu dengan baik karena pada saat yang bersamaan juga banyak diberikan tugas besar yang menyita banyak waktu.

DAFTAR PUSTAKA

<https://math.stackexchange.com/questions/556341/rgb-to-hsv-color-conversion-algorithm> (Diakses 10 November 2023)

<https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1> (Diakses 12 November 2023)

<https://www.sciencedirect.com/science/article/pii/S0895717710005352> (Diakses 11 November 2023)

Link Video: <https://youtu.be/-rm3GqrJkEQ>

Link Github: <https://github.com/dewodt/Algeo02-22011/releases/tag/v.1.0>