

LAPORAN TUGAS BESAR 02

IF 2123 ALJABAR LINEAR DAN GEOMETRI

Kelompok Taman Bermain



Disusun oleh :

Filbert	13522021
Juan Alfred Wijaya	13522073
Azmi Mahmud Bahzeid	13522109

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA (STEI)
INSTITUT TEKNOLOGI BANDUNG
2023

DAFTAR ISI

DAFTAR ISI.....	2
BAB I DESKRIPSI MASALAH.....	4
BAB II LANDASAN TEORI.....	5
2.1 Dasar Teori yang Digunakan Secara Umum.....	5
2.1.1 CBIR dengan Parameter Warna.....	5
2.1.2 CBIR dengan Parameter Tekstur.....	6
2. 2 Pengembangan Website.....	9
BAB III ANALISIS PEMECAHAN MASALAH.....	11
3.1 Langkah - Langkah Pemecahan Masalah.....	11
3.1.1 Pemecahan Masalah Warna.....	11
3.1.2 Pemecahan Masalah Texture.....	12
3.2 Proses Pemetaan Masalah Menjadi Elemen pada Aljabar Geometri.....	13
3.3 Contoh Ilustrasi Kasus dan Penyelesaiannya.....	13
3.3.1 Kasus Warna.....	13
3.3.2 Kasus Tekstur.....	14
BAB IV IMPLEMENTASI DAN UJI COBA.....	18
4.1 Implementasi Program Utama.....	18
4.1.1 Color - Based CBIR.....	18
4.1.2 Texture - Based CBIR.....	19
4.2 Struktur Program.....	21
4.3 Tata Cara Penggunaan Program.....	23
4.4 Hasil Pengujian.....	24
4.4.1 Dataset yang Digunakan.....	24
4.4.2 Pengujian dengan Metode Color.....	24
4.4.3 Pengujian dengan Metode Texture.....	28
4.4.4 Pengujian dengan Webcam.....	32
4.4.5 Pengujian fitur Web Scraping.....	33
4.4.6 Pengujian Export PDF.....	36
4.5 Analisis.....	37
BAB V SARAN DAN KESIMPULAN.....	38
5.1 Kesimpulan.....	38
5.2 Saran.....	38
5.3 Komentar.....	38
5.4 Refleksi.....	39
5.5 Ruang Perbaikan.....	39
DAFTAR PUSTAKA.....	40
LAMPIRAN.....	41

BAB I

DESKRIPSI MASALAH

Di era digital saat ini, di mana jumlah gambar digital meningkat secara eksponensial, kebutuhan untuk mengelola dan mengakses kumpulan gambar yang besar dan beragam dengan efisiensi tinggi menjadi semakin penting. Dengan berkembangnya teknologi dan peningkatan produksi konten visual, tradisional metode pencarian gambar yang berdasarkan teks atau tag menjadi kurang efektif. Inilah sebabnya sistem Temu Balik Gambar Berbasis Konten (Content-Based Image Retrieval, CBIR) memainkan peran krusial dalam industri pengolahan data dan pencarian informasi.

CBIR merupakan teknologi yang memungkinkan pengguna untuk mencari dan mengakses gambar berdasarkan kesamaan nilai citra visual, bukan melalui teks atau kata kunci. Ini berarti sistem CBIR menganalisis konten visual dari gambar - seperti warna, bentuk, tekstur, atau kombinasi dari semua unsur tersebut - untuk memungkinkan pencarian yang lebih intuitif dan relevan. Pendekatan ini sangat bermanfaat dalam berbagai aplikasi, mulai dari pengelolaan arsip digital, pencarian medis dan ilmiah, hingga aplikasi dalam e-commerce dan media sosial.

Salah satu aspek penting dalam implementasi CBIR adalah penggunaan aljabar vektor. Aljabar vektor digunakan untuk menggambarkan dan menganalisis data gambar, memungkinkan klasifikasi berbasis konten yang lebih akurat. Dengan mengubah elemen visual gambar menjadi vektor dalam ruang multidimensi, sistem CBIR dapat mengukur kesamaan atau perbedaan antara gambar-gambar tersebut berdasarkan jarak atau kesamaan vektor. Misalnya, dalam konteks warna, sistem dapat mengkonversi histogram warna dari setiap gambar menjadi vektor dan kemudian membandingkan vektor-vektor ini untuk menentukan kesamaan.

Pengembangan CBIR yang diintegrasikan dalam bentuk situs web membuka peluang baru untuk aksesibilitas dan interaksi pengguna. Dengan menggunakan teknologi web, sistem CBIR dapat diakses oleh pengguna dari berbagai perangkat dan lokasi, meningkatkan keterjangkauan dan kemudahan penggunaan. Integrasi dengan teknologi web juga memungkinkan pemanfaatan algoritma pencarian gambar yang lebih canggih dan interaktif, memberikan pengalaman pengguna yang lebih intuitif dan responsif.

BAB II

LANDASAN TEORI

2.1 Dasar Teori yang Digunakan Secara Umum

Dalam Content-Based Image Retrieval (CBIR), pendekatan berbasis warna dan tekstur adalah dua metode utama yang digunakan untuk memfasilitasi pencarian dan pengambilan gambar. Metode-metode ini mengandalkan pada karakteristik visual intrinsik dari gambar untuk mengidentifikasi dan mengklasifikasikan konten gambar.

2.1.1 CBIR dengan Parameter Warna

Dalam Content-Based Image Retrieval (CBIR) yang berfokus pada warna, metode yang sering digunakan adalah histogram warna. Histogram ini menghitung frekuensi kemunculan warna-warna dalam sebuah ruang warna yang ditentukan. Ada dua metode utama dalam penghitungan histogram warna ini, global dan blok.

Histogram warna global memberikan gambaran keseluruhan distribusi warna pada gambar tanpa memperhatikan informasi spasial. Ini efektif untuk menangkap gambaran umum tetapi tidak mampu mengidentifikasi pola atau lokasi spesifik dari warna-warna tersebut. Akibatnya, gambar yang berbeda namun memiliki distribusi warna serupa dapat dianggap mirip oleh metode ini.

Sebaliknya, histogram warna blok menyediakan wawasan yang lebih mendetail mengenai distribusi lokal warna dengan membagi gambar menjadi grid 3x3. Setiap blok dianalisis secara terpisah, dan kemudian informasi warna dari semua blok digabungkan, menghasilkan representasi yang lebih kaya. Pendekatan ini lebih alami bagi persepsi visual manusia karena mempertimbangkan konteks lokal warna, meskipun lebih kompleks dari segi perhitungan.

Untuk implementasi CBIR berdasarkan warna dalam tugas ini, citra input akan dibandingkan dengan citra dalam basis data menggunakan metode histogram warna. Proses ini dimulai dengan mengonversi citra dari format RGB ke ruang warna yang lebih umum, seperti HSV, yang lebih sesuai dengan persepsi manusia terhadap warna. HSV dianggap lebih intuitif untuk penggunaan dalam kertas kerja karena menyerupai cara manusia memahami warna pada umumnya.

Berikut adalah langkah - langkah konversi warna dari RGB ke HSV,

- a. Nilai dari RGB harus dinormalisasi dengan mengubah nilai *range* [0, 255] menjadi [0, 1]

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

- b. Mencari C_{max} , C_{min} , dan Δ

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

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

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

- c. Selanjutnya gunakan hasil perhitungan di atas untuk mendapatkan nilai HSV.

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

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

$$V = C_{maks}$$

Pembentukan histogram dimulai dengan normalisasi nilai HSV ke rentang yang lebih kecil dan mengategorikan nilai-nilai tersebut ke dalam 'bin' yang sesuai. Dengan menghitung jumlah piksel yang jatuh dalam setiap 'bin', kita mendapatkan histogram warna yang bisa digunakan untuk mengidentifikasi dan membandingkan fitur warna baik secara global maupun blok. Ini memungkinkan sistem CBIR untuk mengevaluasi dan membandingkan citra berdasarkan karakteristik warna yang diekstrak, baik secara keseluruhan maupun dalam segmen-segmen lokal.

Setelah mendapatkan nilai histogram dalam bentuk vektor lakukanlah perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity*.

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

Dengan A dan B adalah vektor dan n adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*.

2.1.2 CBIR dengan Parameter Tekstur

Dalam sistem Content-Based Image Retrieval (CBIR), pendekatan berbasis tekstur memainkan peran penting dalam analisis dan pengambilan gambar. Tekstur dalam gambar, yang mencakup pola dan struktur visual, biasanya diekstraksi menggunakan matriks co-occurrence. Matriks ini efektif dalam mengukur frekuensi pasangan piksel dengan tingkat keabuan tertentu yang muncul bersamaan dalam sebuah gambar, dengan jarak dan arah yang

spesifik. Dari informasi ini, kita dapat menghitung berbagai parameter tekstur seperti kontras, entropi, dan homogenitas, yang memberikan wawasan mendalam tentang karakteristik permukaan dan pola gambar.

Metode ini, meskipun mungkin kurang langsung dibandingkan dengan pendekatan berbasis warna, sangat berguna dalam mengidentifikasi dan membedakan tekstur, yang seringkali tidak bisa dilihat hanya dengan pengamatan warna. Parameter tekstur ini dapat sangat membantu dalam situasi dimana warna tidak memberikan informasi yang cukup untuk pengambilan atau klasifikasi gambar.

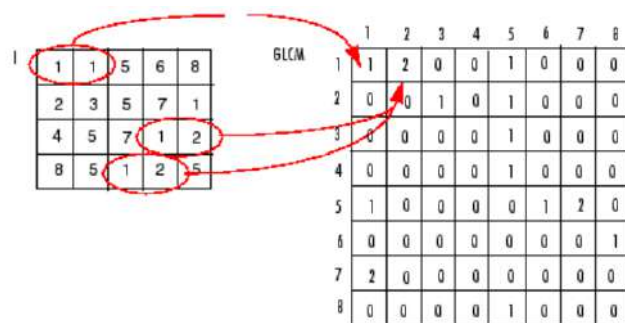
Pendekatan berbasis warna dan tekstur sering kali digunakan secara bersamaan dalam sistem CBIR untuk menghasilkan hasil yang lebih akurat dan efisien. Integrasi ini memungkinkan sistem untuk memanfaatkan kekuatan dari kedua pendekatan tersebut, sehingga mencapai pencocokan gambar yang lebih alami dan lebih dekat dengan cara manusia mempersepsi gambar.

CBIR dengan perbandingan tekstur dilakukan menggunakan suatu matriks yang dinamakan *co-occurrence matrix*. Matriks ini digunakan karena dapat melakukan pemrosesan yang lebih mudah dan cepat. Vektor yang dihasilkan juga mempunyai ukuran yang lebih kecil. Misalkan terdapat suatu gambar I dengan $n \times m$ piksel dan suatu parameter offset $(\Delta x, \Delta y)$, Maka dapat dirumuskan matriksnya sebagai berikut:

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.

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

Melalui persamaan tersebut, digunakan nilai θ adalah 0° , 45° , 90° , dan 135° .



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

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

Langkah - langkah yang dilakukan dalam pemrosesan CBIR berbasis tekstur adalah sebagai berikut,

1. 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:

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

2. Lakukan kuantifikasi dari nilai *grayscale*. Karena citra *grayscale* berukuran 256 piksel, maka matriks yang berkoresponden akan berukuran 256×256 . Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.
3. 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 :

- a. Contrast

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

P : matriks *co-occurrence*

Contrast mengukur tingkat kontras lokal pada citra. Dalam matriks *co-occurrence*, contrast tinggi mengindikasikan adanya perbedaan intensitas yang besar antara piksel-piksel yang berdekatan. Ini biasanya berarti adanya pola atau tekstur yang jelas dan tegas pada citra. Matematis, contrast dihitung sebagai varians dari perbedaan intensitas piksel, dengan memberikan bobot lebih pada pasangan piksel yang memiliki perbedaan intensitas yang lebih besar.

- b. Homogeneity

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

P : matriks *co-occurrence*

Homogeneity (kadang disebut juga sebagai uniformity atau maksimum probability) mengukur kehomogenan atau keseragaman dalam distribusi tekstur citra. Sebuah citra yang memiliki tekstur atau pola yang seragam akan memiliki nilai homogeneity yang tinggi. Dalam matriks *co-occurrence*,

homogeneity dihitung dengan menjumlahkan setiap elemen matriks dengan mempertimbangkan kedekatan mereka ke diagonal utama matriks. Jadi, pasangan piksel dengan tingkat keabuan yang hampir sama (yakni, pasangan yang serupa atau homogen) akan memberikan kontribusi lebih besar kepada nilai homogeneity.

c. Entropy

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

P : matriks co-occurrence

Entropy mengukur keragaman informasi atau kompleksitas dalam distribusi tekstur citra. Dalam konteks matriks co-occurrence, entropy tinggi menandakan distribusi piksel yang sangat bervariasi dan kompleks, dengan banyak informasi tekstur yang terdistribusi secara acak. Dengan kata lain, sebuah citra dengan banyak tekstur yang berbeda dan tidak teratur akan memiliki nilai entropy yang tinggi. Entropy dihitung dengan menjumlahkan produk dari setiap elemen matriks dengan logaritma natural dari elemen tersebut, di mana elemen dengan nilai nol diabaikan karena kontribusinya terhadap entropy adalah nol.

Dari ketiga komponen tersebut, dibuatlah sebuah vektor yang akan digunakan dalam proses pengukuran tingkat kemiripan.

4. 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.

2.2 Pengembangan Website

Pengembangan sebuah website melibatkan beberapa tahap penting, yang masing-masing memiliki peranan krusial dalam menciptakan sebuah situs web yang efektif dan menarik. Berikut adalah uraian dari setiap tahap tersebut:

1. Perencanaan dan Analisis Kebutuhan: Tahap awal ini sangat penting karena di sinilah tujuan, target audiens, dan kebutuhan fungsional website ditentukan.

Analisis ini termasuk menentukan jenis konten, fitur, dan teknologi yang akan digunakan.

2. Desain dan Tata Letak: Setelah kebutuhan ditentukan, tahap selanjutnya adalah mendesain tampilan visual website. Ini termasuk pemilihan skema warna, font, dan layout. Desain harus tidak hanya menarik secara visual tetapi juga user-friendly dan mudah dinavigasi.
3. Pengembangan: Pada tahap ini, desain yang telah dibuat diubah menjadi kode yang sebenarnya. Ini termasuk pengembangan front-end (HTML, CSS, JavaScript) dan back-end (server, database, aplikasi). Pengembang juga perlu memastikan website responsif dan kompatibel dengan berbagai perangkat dan browser.
4. Pengujian: Sebelum website diluncurkan, penting untuk melakukan pengujian menyeluruh. Ini termasuk pengujian fungsionalitas, kompatibilitas browser, performa, dan keamanan. Tujuan utama dari tahap ini adalah untuk mengidentifikasi dan memperbaiki bug atau masalah apa pun.
5. Peluncuran: Setelah website diuji dan dianggap siap, website tersebut dapat diluncurkan. Ini bisa melibatkan migrasi dari server pengembangan ke server produksi.
6. Pemeliharaan dan Pembaruan: Pengembangan website tidak berhenti setelah peluncuran. Website perlu dipelihara secara teratur untuk memastikan keamanan, memperbarui konten, dan mengimplementasikan perbaikan bug atau peningkatan fitur.
7. SEO dan Pemasaran Digital: Penting juga untuk mempertimbangkan optimasi mesin pencari (SEO) dan strategi pemasaran digital untuk meningkatkan visibilitas dan lalu lintas ke website.

Pembangunan sebuah website melibatkan sejumlah tahapan yang memerlukan keterampilan dan pengetahuan yang berbeda. Tahapan ini seringkali melibatkan tim multidisiplin yang terdiri dari desainer, pengembang, pengujian, dan pemasaran. Keberhasilan sebuah website sangat bergantung pada seberapa baik setiap tahap ini dijalankan.

Dalam pembangunan website ini, fokus hanya mencapai tahap pengujian. Tahap peluncuran dan semua langkah yang terjadi setelahnya tidak dilakukan dalam konteks tugas ini. Oleh karena itu, tahap pengujian dan pengembangan dilakukan berulang kali untuk memastikan bahwa website memiliki fungsionalitas terbaik dan efisiensi yang optimal. Tahap peluncuran tidak termasuk dalam ruang lingkup tugas ini, karena bukan menjadi tujuan utama dari pengerjaan tugas ini. Sebaliknya, fokus utama adalah memastikan bahwa website ini melewati tahap pengujian dengan baik untuk mencapai kualitas yang diinginkan.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah - Langkah Pemecahan Masalah

3.1.1 Pemecahan Masalah Warna

Pertama kita bagi HSV color space ($0 \leq H \leq 180$, $0 \leq S$, $V \leq 255$) menjadi beberapa partisi yang mirip warnanya:

Hue

[0, 12.5], [13.0, 20.0], [20.5, 60.0], [60.5, 95.0], [95.5, 135.0],
[135.5, 147.5], [147.5, 157.5], [158.0, 180.0]]

Saturation

[[0, 50.5], [51.0, 179.0], [179.5, 255]]

Value

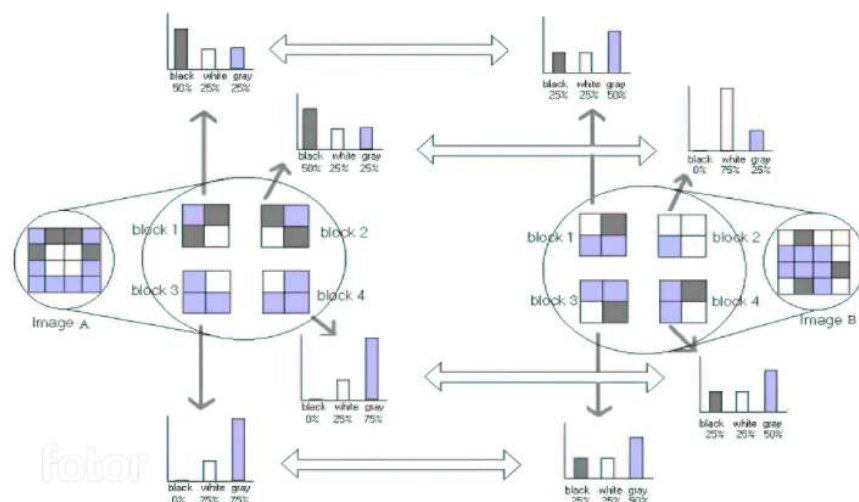
[[0, 50.5], [51.0, 179.0], [179.5, 255]]

Sehingga banyaknya partisi adalah $8 \times 3 \times 3 = 72$ partisi.

a. global

1. Buatlah sebuah histogram (untuk setiap gambar) dengan 72 partisi yang sudah ditentukan sebagai *bins*-nya.
2. Isilah histogram dengan melakukan iterasi setiap pixel, dan meng-*increment* bin yang sesuai dengan pixel yang bersangkutan.
3. Buatlah suatu vektor (untuk setiap gambar) dengan 72 komponen dari 72 bins pada histogram yang bersangkutan.
4. Nilai similaritas antara kedua gambar dapat dihitung dengan menghitung cosine similarity antara dua vektor yang bersangkutan.

b. block



(gambar dari https://cis.temple.edu/~lakaemper/courses/cis595_2004/papers/wang2001.pdf halaman 12)

1. Resize kedua gambar sehingga berukuran sama.
2. Untuk setiap block 3 x 3 (gambar di atas menggunakan block 2 x 2, namun konsepnya tetap sama) yang bersesuaian pada kedua gambar, hitunglah kedua vektor dengan cara yang sama dengan global.
3. Hitung nilai similaritas antara kedua block yang bersesuaian dengan menghitung cosine similarity antara kedua vektor yang bersesuaian.
4. Nilai similaritas antara kedua gambar dapat dihitung dengan mencari nilai rata-rata dari semua nilai similaritas antar-block.

3.1.2 Pemecahan Masalah Texture

Berikut ialah pemecahan masalah tekstur :

1. Crop kedua gambar sehingga berukuran sama.
2. Membuat *co-occurrence matrix* dari masing-masing gambar menggunakan parameter $d = 1, \theta = 0^\circ$
3. Membuat *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan transposenya.
4. Membuat matriks normalisasi dari *symmetric matrix* dengan membagi *symmetric matrix* dengan total semua elemen di *symmetric matrix*.
5. Menghitung *contrast*, *homogeneity*, dan *entropy* dengan rumus

$$\text{Contrast} = \sum_{i=0}^{255} \sum_{j=0}^{255} P_{i,j} (i - j)^2$$

$$\text{Homogeneity} = \sum_{i=0}^{255} \sum_{j=0}^{255} \frac{P_{i,j}}{1 + (i - j)^2}$$

$$\text{Entropy} = - \sum_{i=0}^{255} \sum_{j=0}^{255} P_{i,j} \ln P_{i,j}$$

6. Mengalikan *contrast*, *homogeneity*, dan *entropy* dengan *weight* yang sesuai agar hasil tidak terlalu bias.
7. Membuat vektor dengan 3 komponen yang merupakan hasil perhitungan *contrast*, *homogeneity*, dan *entropy*.

8. Menghitung similaritas antara kedua gambar dengan menghitung cosine similarity antara kedua vektor.

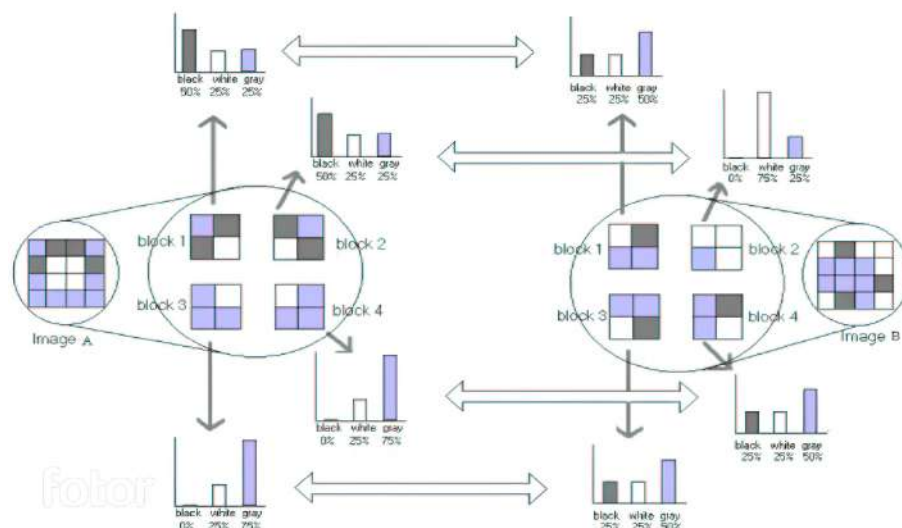
3.2 Proses Pemetaan Masalah Menjadi Elemen pada Aljabar Geometri

Berikut ialah pemetaan masalah menjadi elemen aljabar geometri:

1. Pemodelan dan representasi data dan persoalan: Gambar direpresentasikan dalam matriks. Histogram warna direpresentasikan dalam vektor yang komponen-komponennya merupakan *bins* dari histogramnya. Perhitungan tekstur menggunakan matriks *co-occurence*. Informasi fitur warna dan fitur tekstur dan direpresentasikan dalam vektor.
2. Perhitungan fitur warna dan fitur tekstur: Operasi matriks dan vektor digunakan dalam banyak perhitungan. Perhitungan tekstur memanfaatkan konsep matriks transpose, perkalian matriks dengan skalar, transformasi linier (perkalian komponen vektor tekstur dengan *weight* yang sesuai), dan yang lain-lainnya.
3. Pencarian berdasarkan kemiripan: Kemiripan dua fitur yang telah dinyatakan dengan vektor dapat dihitung menggunakan operasi vektor yaitu perkalian titik dan magnitudo vektor untuk menentukan *cosine similarity* antara dua vektor. *Cosine similarity* menyatakan nilai cosinus sudut antara kedua vektor sehingga semakin mirip vektornya, semakin kecil sudutnya dan nilai cosinus sudut akan mendekati 1 atau 100%.

3.3 Contoh Ilustrasi Kasus dan Penyelesaiannya

3.3.1 Kasus Warna



Untuk memudahkan perhitungan, kita akan membagi color space menjadi 3 dan bukan seperti pada perhitungan aslinya yang menggunakan 72. Kita namakan tiga partisi tersebut black, white, gray. Kita juga akan menggunakan block berukuran 2 x 2 dan bukan seperti pada perhitungan aslinya yang menggunakan 3 x 3. Sebelum menghitung, gambar harus sudah

berukuran sama sehingga setiap block di gambar pertama memiliki block yang bersesuaian di gambar yang kedua (kita lakukan *resizing* jika belum). Pada contoh di atas, telah didapatkan histogram-histogram warna yang mewakili setiap block. Sebagai contoh, pada gambar A di block ujung kiri atas, histogram atau vektor menyatakan distribusi warna adalah $(2, 1, 1)$ (atau dapat menggunakan persentase namun tidak mengubah jawaban akhir). Kita dapat menghitung nilai similaritas gambar A dan gambar B dari histogram-histogram yang telah dihitung dengan menghitung rata-rata *cosine similarity* dari setiap pasangan block yang bersesuaian:

$$\frac{(2,1,1) \cdot (1,1,2)}{\|(2,1,1)\| \cdot \|(1,1,2)\|} + \frac{(2,1,1) \cdot (0,3,1)}{\|(2,1,1)\| \cdot \|(0,3,1)\|} + \frac{(0,1,3) \cdot (1,1,2)}{\|(0,1,3)\| \cdot \|(1,1,2)\|} + \frac{(0,1,3) \cdot (1,1,2)}{\|(0,1,3)\| \cdot \|(1,1,2)\|} = 0.78$$

Untuk perhitungan yang global, kita tidak perlu membagi gambar menjadi banyak block-block. Perhitungan global mirip dengan perhitungan lokal dengan block-block seperti yang di atas, namun ukuran blocknya adalah ukuran gambarnya.

3.3.2 Kasus Tekstur

Contoh perhitungan untuk tekstur adalah berikut yang telah diilustrasikan oleh Yunus Muhammad:

Misalkan ada matriks 3 x 3 di mana setiap cell mewakili gray tone dari 0 sampai 3.

0	0	1
1	2	3
2	3	2

Quantization level didefinisikan sebagai banyaknya kemungkinan gray tone. Pada kasus ini, *quantization level*-nya adalah 4 karena ada 4 bilangan dari 0 sampai 3. Maka, dibuat matriks 4 x 4 terlebih dahulu yang awalnya adalah 0 semua:

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Setiap cell mewakili pasangan 2 warna gray tone, (a, b) di mana $0 \leq a, b \leq 3$.

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

Selanjutnya, kita isi matriks 4 x 4 dengan mengiterasikan setiap row setiap 2 pixel bersebelahan seperti pada gambar-gambar berikut:

image size = 3x3
 gray tone = 0-3
 quantization level = 3
 angel = 0°
 distance = 1

Kemudian, kita normalisasi *symmetric matrix*-nya dengan membagi dengan total elemen ($2+1+1+1+1+3+3 = 12$):

0.16	0.08	0	0
0.08	0	0.08	0
0	0.08	0	0.25
0	0	0.25	0

Normalized Matrix

Kontrasnya dapat dihitung sebagai berikut:

$$\begin{aligned}
 &0.1666 \cdot (0 - 0)^2 + 0.0833 \cdot (0 - 1)^2 + 0 \cdot (0 - 2)^2 + 0 \cdot (0 - 3)^2 + \\
 &0.0833 \cdot (1 - 0)^2 + 0 \cdot (1 - 1)^2 + 0.0833 \cdot (1 - 2)^2 + 0 \cdot (1 - 3)^2 + \\
 &0 \cdot (2 - 0)^2 + 0.0833 \cdot (2 - 1)^2 + 0 \cdot (2 - 2)^2 + 0.25 \cdot (2 - 3)^2 + \\
 &0 \cdot (3 - 0)^2 + 0 \cdot (3 - 1)^2 + 0.25 \cdot (3 - 2)^2 + 0 \cdot (3 - 3)^2 = 0.833
 \end{aligned}$$

Dengan cara yang sama, kontras, homogeneitas, dan entropi dapat dihitung sehingga didapatkan:

$$\text{Kontras} = 0.833$$

$$\text{Homogeneitas} = 0.583$$

$$\text{Entropi} = 1.82$$

Maka, vektor tekstur direpresentasikan sebagai vektor dengan 3 komponen yang memiliki nilai di atas. Kemudian, kita akan mengalikan komponen dari vektor dengan *weight* yang didapatkan dari banyak eksperimen. Kita lakukan ulang prosesnya pada gambar kedua (yang telah dicrop sehingga memiliki ukuran yang sama dengan gambar pertama) sehingga didapatkan vektor tekstur yang lain. Maka, nilai similaritas antara dua gambar dapat dihitung dengan mencari *cosine similarity* antara kedua vektor tersebut.

BAB IV

IMPLEMENTASI DAN UJI COBA

4.1 Implementasi Program Utama

4.1.1 Color - Based CBIR

GLOBAL COLOR BASED CBIR
<pre> constant h_ranges: array of array of integer = [[0, 12.5], [13.0, 20.0], [20.5, 60.0], [60.5, 95.0], [95.5, 135.0], [135.5, 147.5], [147.5, 157.5], [158.0, 180.0]] constant s_ranges: array of array of integer = [[0, 50.5], [51.0, 179.0], [179.5, 255]] constant v_ranges: array of array of integer = [[0, 50.5], [51.0, 179.0], [179.5, 255]] function get_vector(image) Initialize vec to zero # vec adalah array of 72 integers (zero-indexed). # Awalnya semua komponennya adalah 0. for every h_interval in h_ranges for every s_interval in s_ranges for v_interval in v_ranges hi <- index_of_h_interval si <- index_of_s_interval vi <- index_of_v_interval increment vec[9 * h_i + 3 * s_i + v_i] -> vector # Prekondisi: Banyaknya komponen first_vector sama dengan banyaknya komponen # second_vector function cosine_similarity(first_vector, second_vector) dot_product <- 0 first_norm <- 0 second_norm <- 0 components_count <- banyaknya komponen di first_vector atau second_vector for 0 <= i < components_count dot_product <- dot_product + first_vector[i] * second_vector[i] first_norm <- first_norm + first_vector[i] * first_vector[i] second_norm <- second_norm + second_vector[i] * second_vector[i] first_norm <- square_root(first_norm) second_norm <- square_root(second_norm) return dot_product / (first_norm * second_norm) function get_color_similarity_globally(first_image, second_image) first_vector <- get_vector(first_image) second_vector <- get_vector(second_image) -> cosine_similarity(first_vector, second_vector) </pre>

BLOCK 3x3 COLOR BASED CBIR

```
constant RESIZED_DIMENSION: integer = 150
    # Image sebelum didapatkan vektornya,
    # diresize terlebih dahulu menjadi persegi dengan
    # ukuran RESIZED_DIMENSION x RESIZED_DIMENSION.

constant BLOCK_DIMENSION: integer = 3      # Ukuran block yang digunakan untuk
    # menghitung vektornya yaitu
    # BLOCK_DIMENSION x BLOCK_DIMENSION

function get_color_similarity_locally(first_image, second_image)
    Resize first_image to RESIZED_DIMENSION x RESIZED_DIMENSION
    Resize second_image to RESIZED_DIMENSION x RESIZED_DIMENSION
    first_blocks <- Partitions of first_image into blocks of BLOCK_DIMENSION
x BLOCK_DIMENSION
    second_blocks <- Partitions of second_image into blocks of
BLOCK_DIMENSION X BLOCK_DIMENSION
    total_blocks <- Number of blocks in first_blocks or second_blocks

    similarity <- 0
    for 0 <= i < total_blocks
        similarity <- similarity +
cosine_similarity(get_vector(first_blocks[i]),
get_vector(second_blocks[i]))
    -> similarity / total_blocks
```

4.1.2 Texture - Based CBIR

```
constant CROPPED_DIMENSION : integer = 300
    # Image sebelum dapat diproses,
    # dicrop terlebih dahulu menjadi persegi dengan
    # ukuran CROPPED_DIMENSION x CROPPED_DIMENSION.

constant CONTRAST_WEIGHT: float = 1 / 50
constant HOMOGENEITY_WEIGHT: integer = 1
constant ENTROPY_WEIGHT: integer = 1

function get_co_occurrence_matrix(image) # d = 1, 0 = 0
    Initialize GLCM to 0
    # GLCM adalah matriks 256 x 256 yang awalnya 0 semua
    for every row in image
        for every adjacent_pixels in row
            Increment GLCM[adjacent_pixels.first,
adjacent_pixels.second]

    GLCM <- transpose(GLCM)
    sum <- sum of every element in GLCM
    GLCM <- GLC / sum
    -> GLCM

function get_contrast(occurence_matrix)
    result <- 0
```

```

    for 0 <= i < 256
        for 0 <= j < 256
            result <- result + (i - j)^2 * occurrence_matrix[i, j]
    -> result

function get_homogeneity(occurrence_matrix)
    result <- 0
    for 0 <= i < 256
        for 0 <= j < 256
            result <- result + occurrence_matrix[i, j] / (1 + (i -
j)^2)
    -> result

function get_entropy(occurrence_matrix)
    result <- 0
    for 0 <= i < 256
        for 0 <= j < 256
            logged <- ln(occurrence_matrix[i, j])
            if logged is not 0 then
                result <- result + occurrence_matrix[i, j] *
logged
    -> result

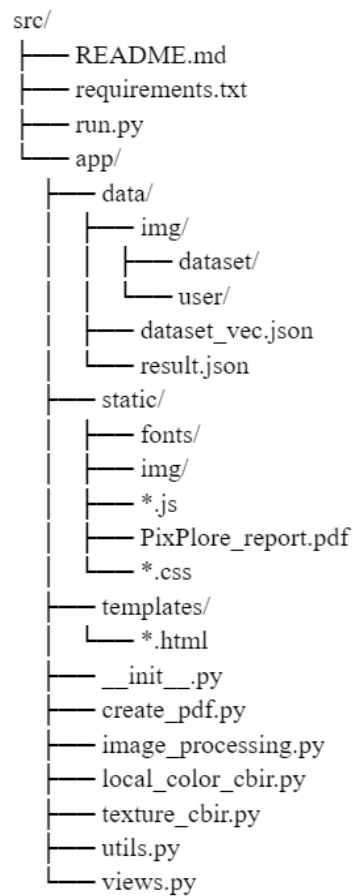
function get_vector(occurrence_matrix)
    contrast = get_contrast(occurrence_matrix) * CONTRAST_WEIGHT
    homogeneity = get_homogeneity(occurrence_matrix) * HOMOGENEITY_WEIGHT
    entropy = get_entropy(occurrence_matrix) * ENTROPY_WEIGHT
    -> [contrast, homogeneity, entropy]

function get_texture_similarity(first_image, second_image)
    Crop first_image to CROPPED_DIMENSION x CROPPED_DIMENSION
    Crop second_image to CROPPED_DIMENSION x CROPPED_DIMENSION

    first_occurrence_matrix = get_co_occurrence_matrix(first_image)
    second_occurrence_matrix = get_co_occurrence_matrix(second_image)
    first_vector = get_vector(first_occurrence_matrix)
    second_vector = get_vector(second_occurrence_matrix)
    similarity <- 0
    -> cosine_similarity(first_vector, second_vector)

```

4.2 Struktur Program



Struktur program dari aplikasi web ini terdiri dari komponen backend yang menggunakan framework Flask dan bahasa pemrograman Python, serta komponen frontend yang mengandalkan HTML, CSS, JS, dan HTMX, sebuah library JavaScript yang memungkinkan program untuk memuat dan memperbarui bagian-bagian tertentu dari halaman web tanpa harus melakukan penjajaran ulang seluruh halaman, memberikan pengalaman pengguna yang lebih cepat dan interaktif. Berikut adalah struktur dari program yang lebih terperinci,

1. **src/** : Direktori utama.
 - **README.md** : File ini berisi dokumentasi dan penjelasan singkat mengenai program, cara penggunaannya, dan panduan instalasi.
 - **requirements.txt** : File ini berisi daftar dependencies dan versi yang diperlukan untuk menjalankan program ini. Biasanya digunakan dengan pip untuk menginstal semua dependencies yang diperlukan.
 - **run.py** : Ini adalah file utama yang akan dijalankan untuk menjalankan aplikasi Flask.
2. **app/** : Direktori program di dalam direktori /src
 - **data/** : Direktori ini digunakan untuk menyimpan data yang dibutuhkan oleh aplikasi.
 - **img/** : Direktori ini berisi gambar-gambar yang digunakan dalam aplikasi.

- **dataset/** : Direktori ini digunakan untuk menyimpan gambar-gambar dataset yang akan digunakan untuk pencarian.
 - **user/** : Direktori ini digunakan untuk sementara menyimpan gambar yang diunggah oleh pengguna. Direktori ini hanya akan menampung satu gambar.
 - **dataset_vec.json** : Ini adalah file JSON yang berisi data tentang gambar-gambar dalam dataset, termasuk nama file, path file, vektor warna, dan vektor tekstur.
 - **result.json** : File ini digunakan untuk menyimpan hasil pencarian, termasuk nama file, nilai kesamaan, tipe pencarian, dan URL gambar yang cocok.
- **static/** : Direktori ini digunakan untuk menyimpan file statis seperti JavaScript, CSS, font, dan dokumen PDF.
 - **fonts/** : Direktori ini berisi file-font yang digunakan dalam tampilan situs web.
 - **img/** : Direktori ini berisi gambar-gambar statis yang digunakan dalam tampilan situs web.
 - ***.js** : File - file JavaScript yang digunakan untuk interaksi klien atau tampilan web.
 - **PixPlore_report.pdf** : Dokumen PDF yang berisi laporan dari hasil pencarian dari aplikasi.
 - ***.css** : File - file CSS yang digunakan untuk mengatur tampilan dan gaya tampilan web.
- **templates/** : Direktori ini digunakan untuk menyimpan berkas-berkas HTML yang digunakan oleh aplikasi Flask untuk menghasilkan tampilan halaman web.
 - ***.html** : File - file HTML yang mungkin mencakup template dasar dan halaman-halaman yang dihasilkan oleh aplikasi.
- **__init__.py** : File ini berisi kode yang digunakan untuk inisialisasi dan konfigurasi aplikasi Flask. Biasanya digunakan untuk mengatur konfigurasi seperti pengaturan database atau koneksi lainnya.
- **create_pdf.py** : Ini adalah modul yang berisi kode untuk membuat file PDF yang mungkin berisi laporan hasil pencarian gambar.
- **image_processing.py** : Modul ini berisi fungsi-fungsi yang berkaitan dengan pemrosesan gambar. Mungkin digunakan untuk mengolah gambar sebelum melakukan pencarian.
- **local_color_cbir.py** : Modul ini berisi fungsi-fungsi yang berkaitan dengan pemrosesan pencarian berbasis warna (Color-based Image Retrieval).
- **texture_cbir.py** : Modul ini berisi fungsi-fungsi yang berkaitan dengan pemrosesan pencarian berbasis tekstur (Texture-based Image Retrieval).

- **utils.py** : Modul ini berisi funTgsi-fungsi utilitas yang mungkin digunakan dalam berbagai bagian aplikasi.
- **views.py** : File ini berisi definisi routing untuk aplikasi Flask, yaitu bagaimana aplikasi akan menangani permintaan HTTP dan memberikan tanggapan kepada pengguna melalui API atau tampilan halaman web.

4.3 Tata Cara Penggunaan Program

Untuk memulai menggunakan website kami, Anda dapat menjalankannya dengan mengikuti langkah-langkah berikut :

1. Membuat virtual environment.

Pertama-tama, buatlah virtual environment bernama 'env' menggunakan modul 'venv' yang tersedia di Python. Anda dapat melakukannya dengan perintah `python -m venv env`.

2. Mengaktifkan Virtual Environment:

- Untuk pengguna Windows, aktifkan dengan menjalankan `env\Scripts\activate`.
- Pengguna macOS/Linux dapat menggunakan source `env/bin/activate`.

3. Instalasi Dependensi:

Pastikan virtual environment aktif (akan terlihat dari adanya '(env)' pada prompt terminal Anda). Kemudian, instal dependensi proyek dengan perintah `pip install -r requirements.txt`.

4. Menjalankan Website:

Setelah itu, jalankan website dengan `flask run --debug`. Ini akan memulai server dan biasanya tersedia di `http://127.0.0.1:5000/`.

5. Menonaktifkan Virtual Environment:

Jika selesai, nonaktifkan virtual environment dengan perintah `deactivate`.

Dalam web ini, terdapat beberapa page yang terdiri dari page untuk Main Program, Home page, Developers dan guides. Jika kita ingin mencoba fitur-fitur yang ada, Anda dapat memulai dengan mengklik tombol 'launch App' di halaman utama atau, jika Anda memerlukan panduan, klik 'get started' yang akan mengarahkan Anda ke halaman panduan.

Pada halaman program, User dapat mengunggah dataset terlebih dahulu dalam bentuk zip atau folder. Setelah itu, unggah gambar yang diinginkan dan pilih pencarian berdasarkan warna atau tekstur melalui toggle yang tersedia. Klik tombol 'search' dan hasilnya akan segera ditampilkan. Kami juga menyediakan fitur untuk mengunduh semua hasil dalam satu file PDF, memudahkan User untuk melihat keseluruhan hasil.

Jika User memerlukan dataset tambahan, fitur web scrape kami dapat membantu. Cukup berikan link website dari mana Anda ingin mengambil foto. Jika diizinkan oleh website tersebut, foto-foto akan secara otomatis tersimpan di dataset dan bisa langsung digunakan di website.

Selain hanya dengan menginsert image, User juga dapat memanfaatkan fitur kamera yang akan langsung mengambil foto dan mencari kemiripannya dengan foto-foto yang ada di dataset kami.

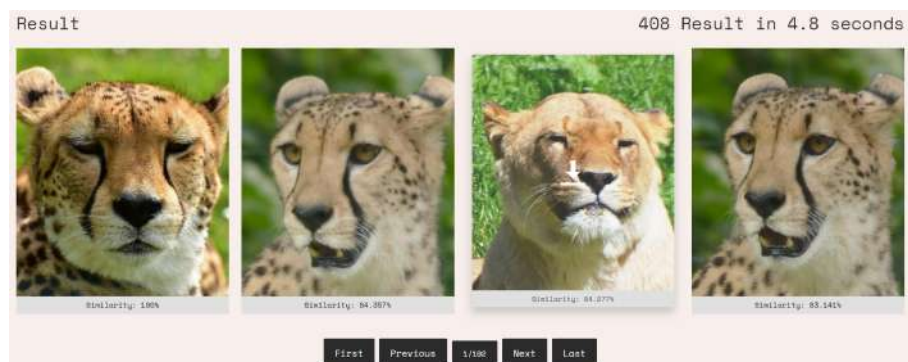
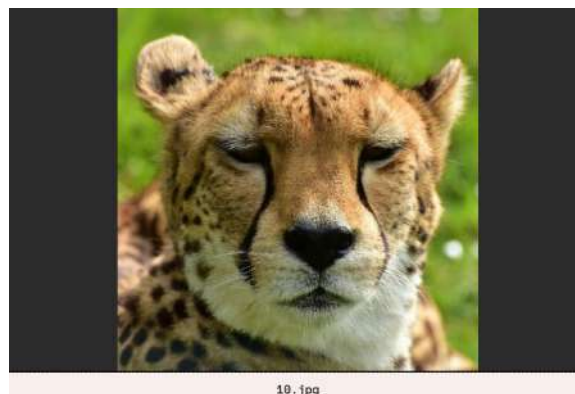
4.4 Hasil Pengujian

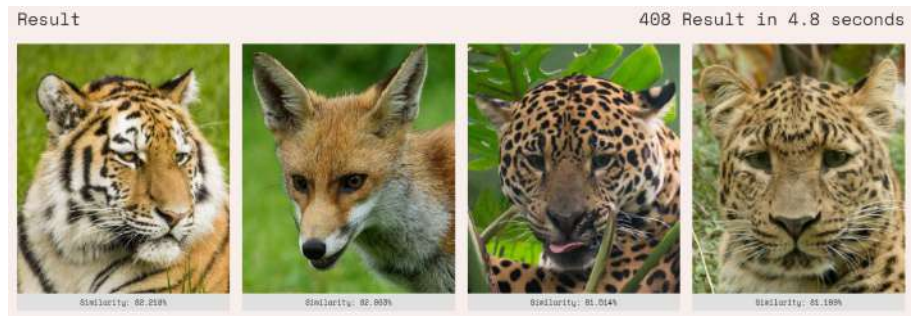
4.4.1 Dataset yang Digunakan

Folder dataset yang digunakan terdapat pada folder *img* yang terdapat pada repository github. Pada dataset ini, sampel-sampel yang digunakan merupakan hewan yang telah disediakan pada spesifikasi penugasan. Pada dataset ini terdapat secara spesifik 4.738 gambar dengan jenis hewan yang bervariasi sehingga hasil-hasil yang didapatkan lebih dapat terlihat similaritasnya.

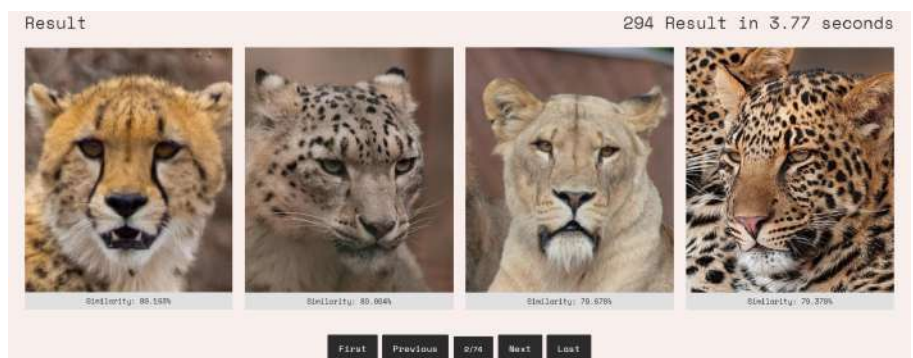
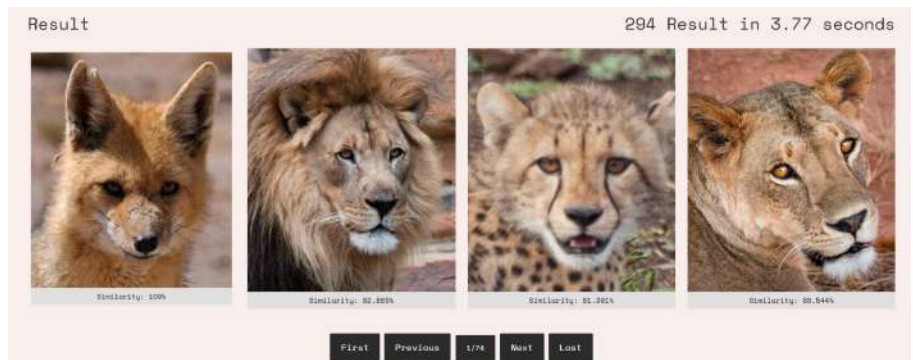
4.4.2 Pengujian dengan Metode Color

a. Hasil Pengujian I

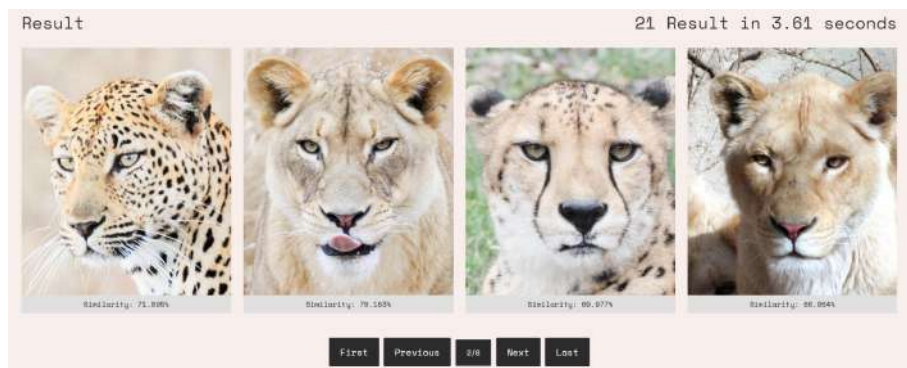
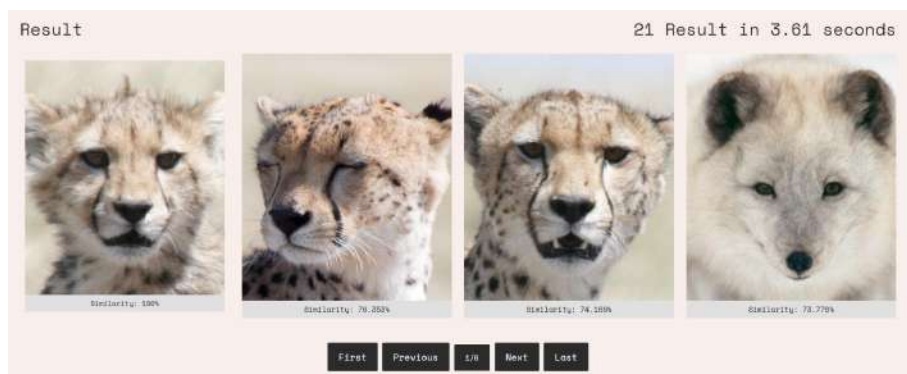




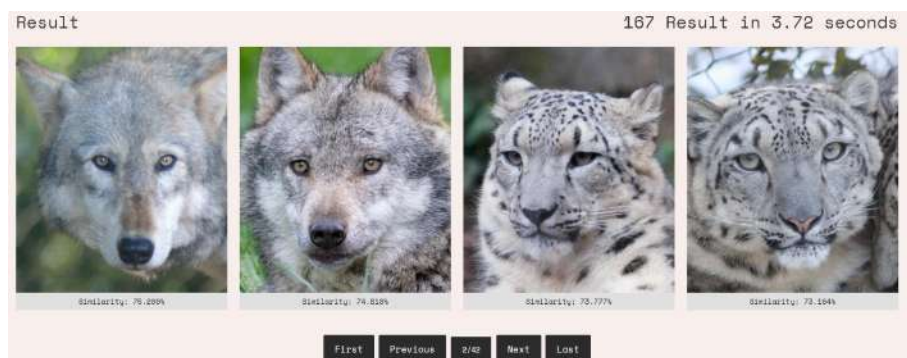
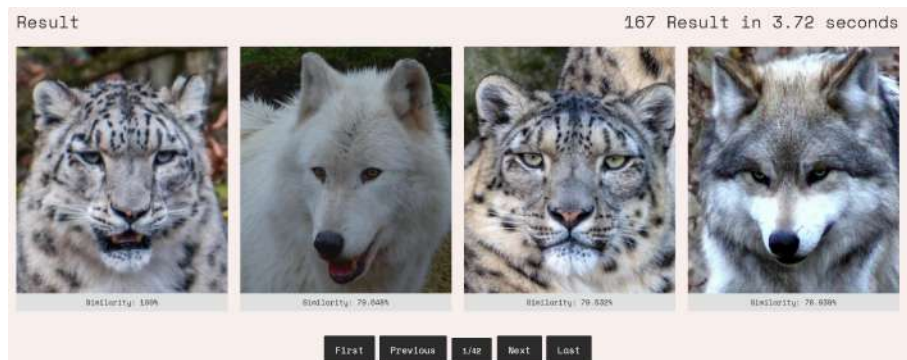
b. Hasil Pengujian II



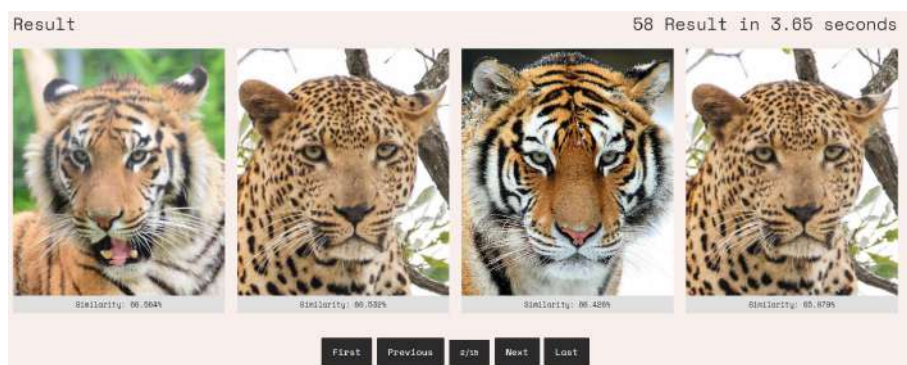
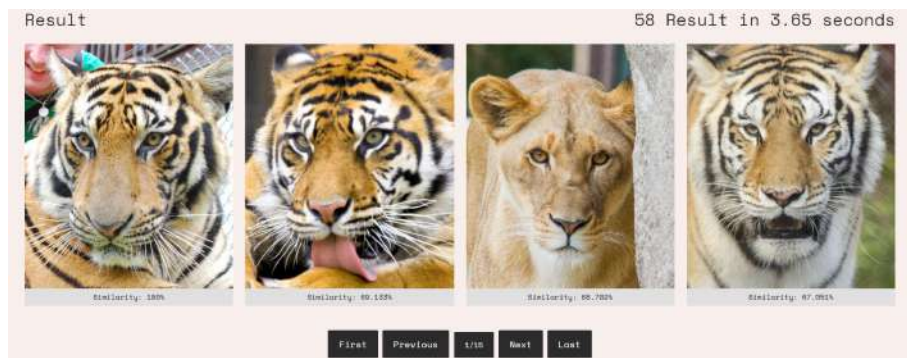
c. Hasil Pengujian III



d. Hasil Pengujian IV

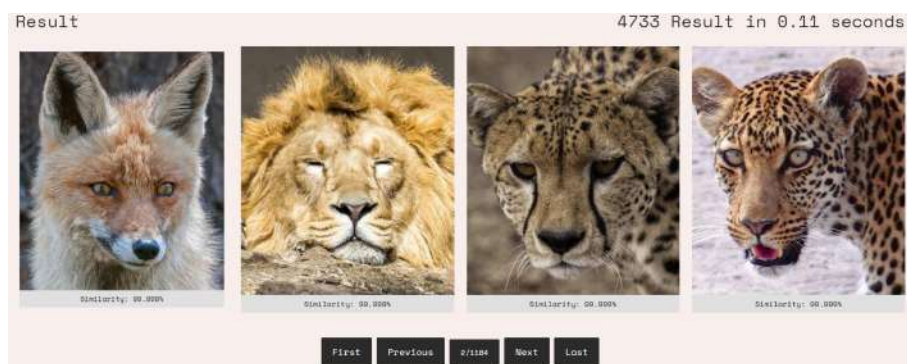
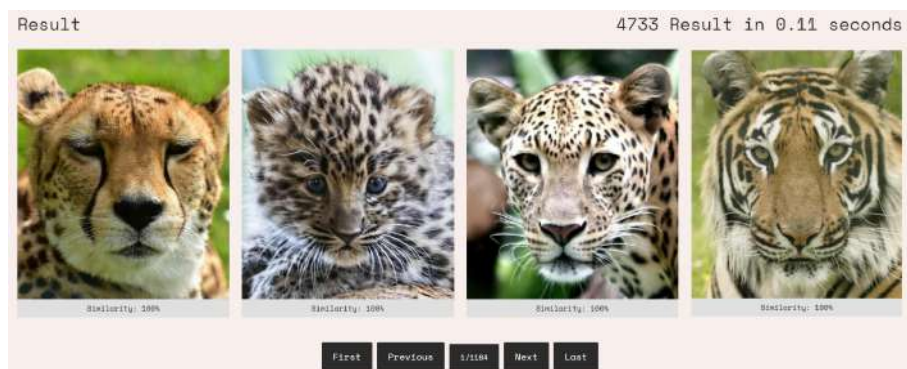
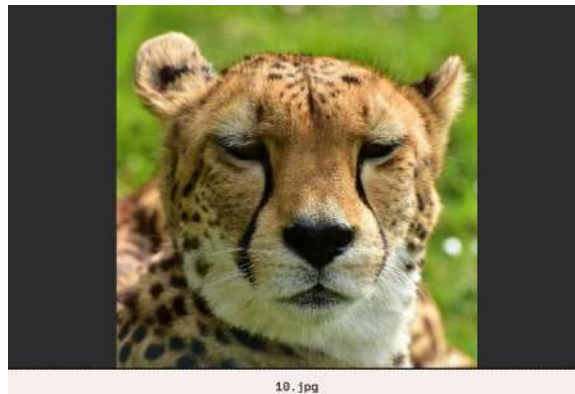


e. Hasil Pengujian V

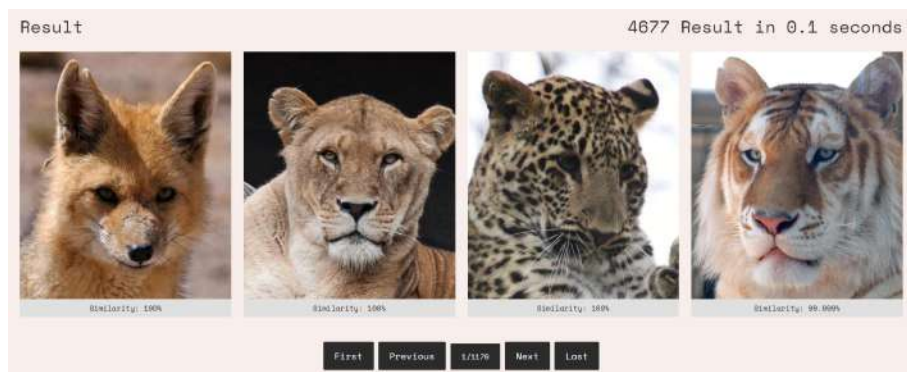


4.4.3 Pengujian dengan Metode Texture

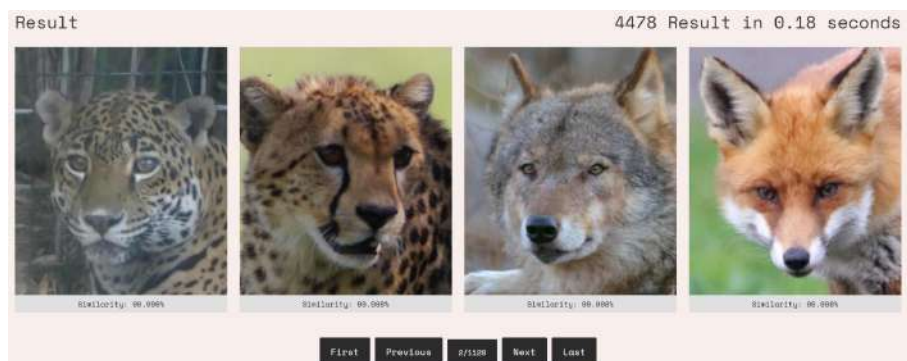
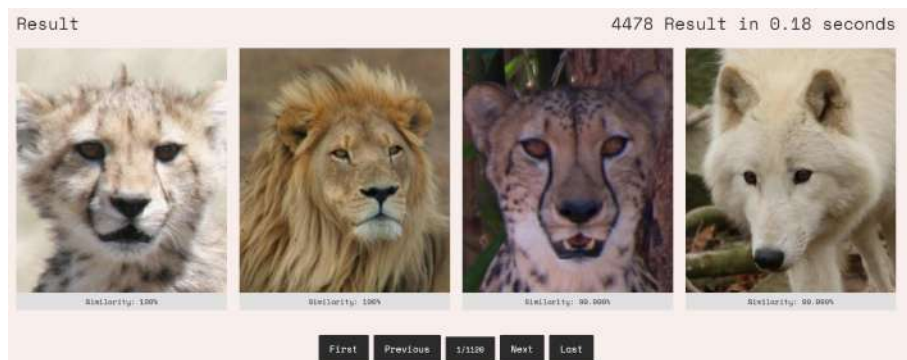
a. Hasil Pengujian I



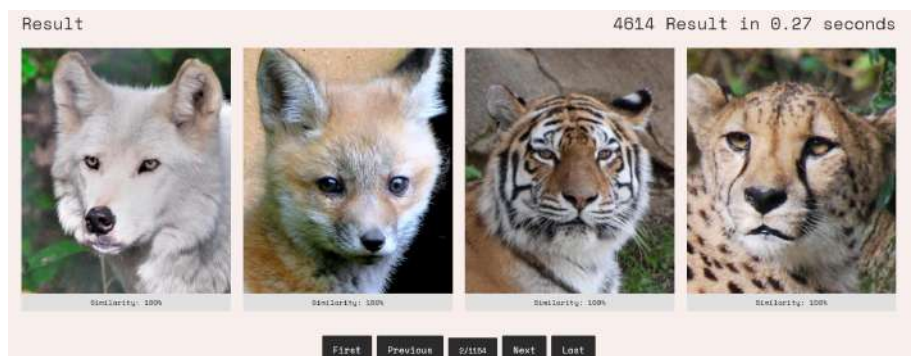
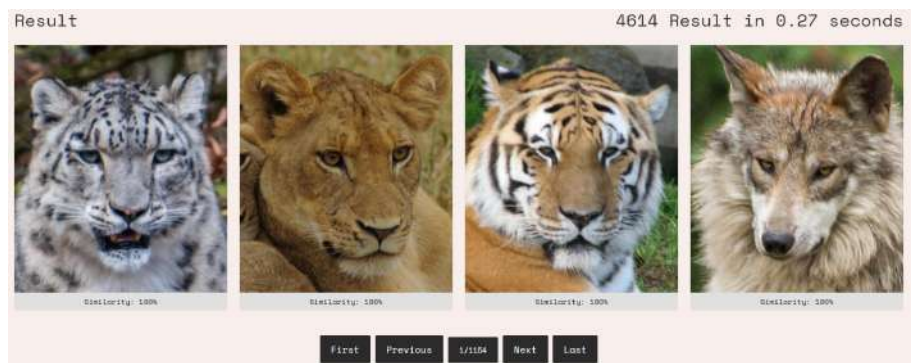
b. Hasil Pengujian II



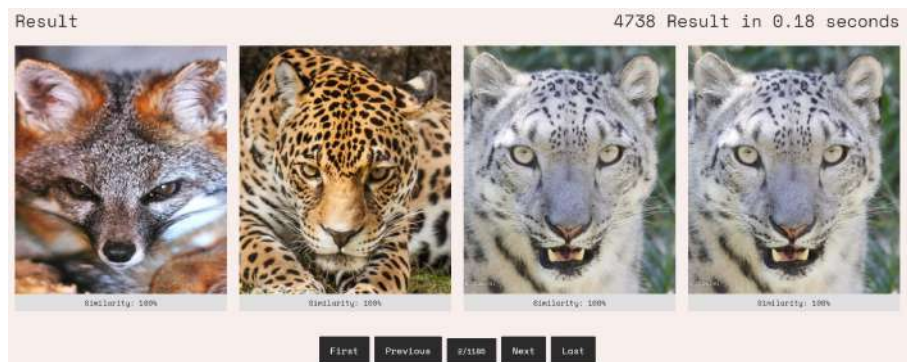
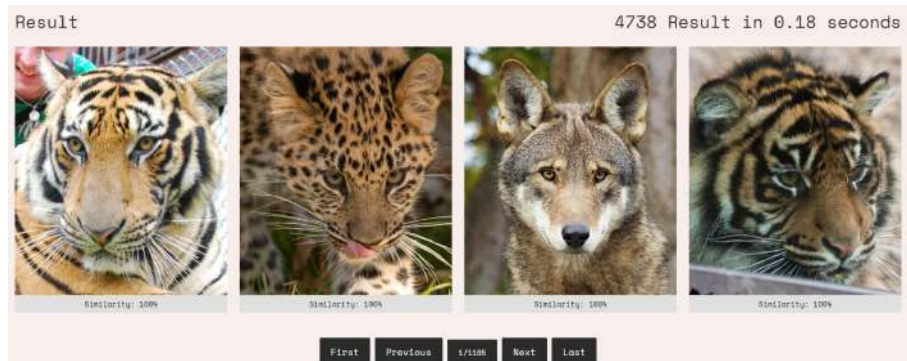
c. Hasil Pengujian III



d. Hasil Pengujian IV

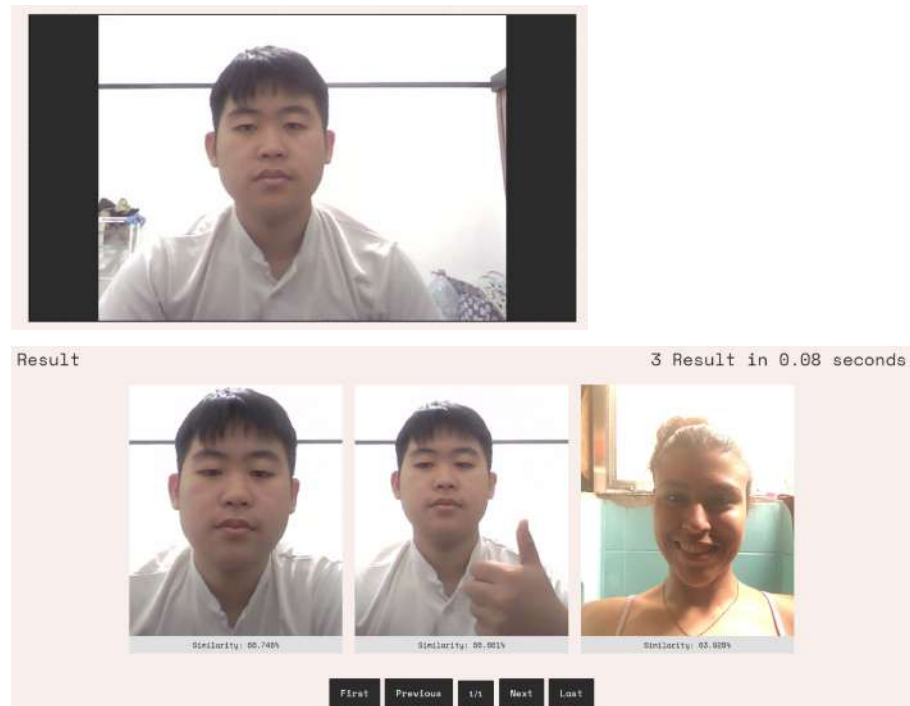


e. Hasil Pengujian V

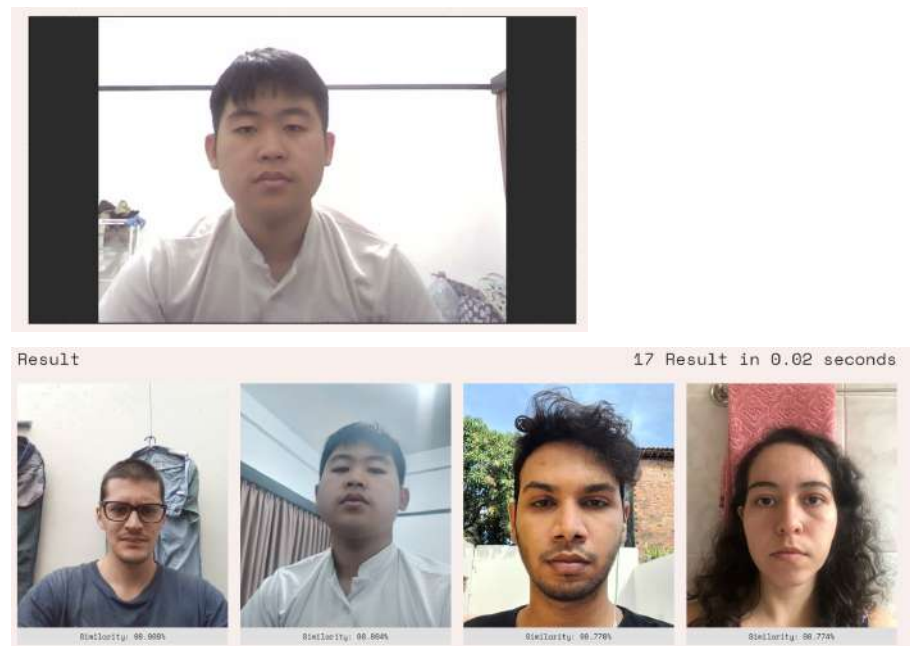


4.4.4 Pengujian dengan Webcam

a. Hasil Pengujian dengan Color




b. Hasil Pengujian dengan Texture




Result


17 Result in 0.02 seconds




Similarity: 98.78%



Similarity: 98.85%



Similarity: 98.88%



Similarity: 98.97%

First

Previous

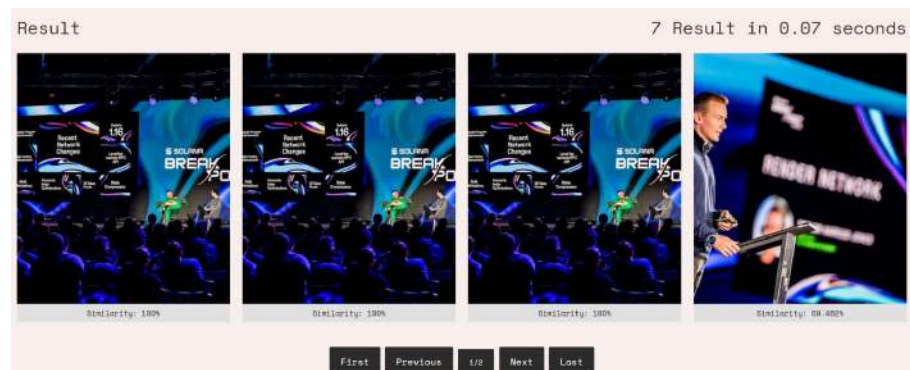
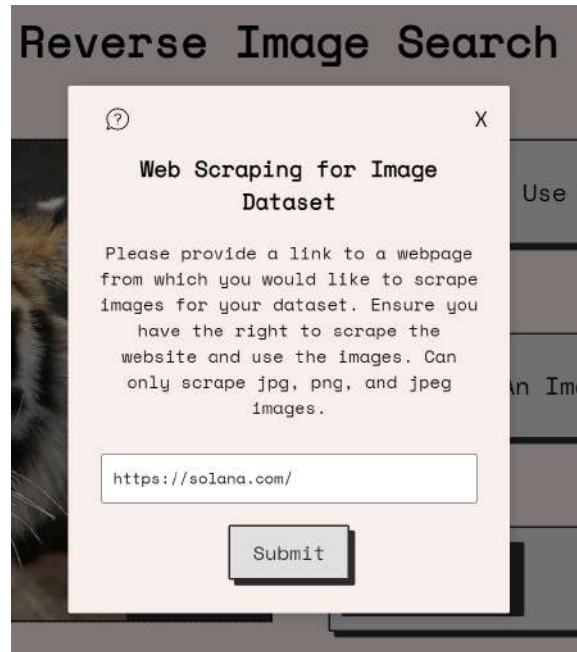
2/5

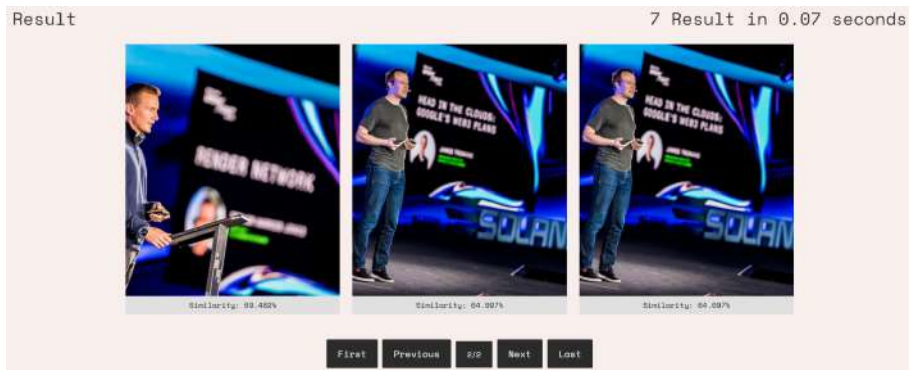
Next

Last

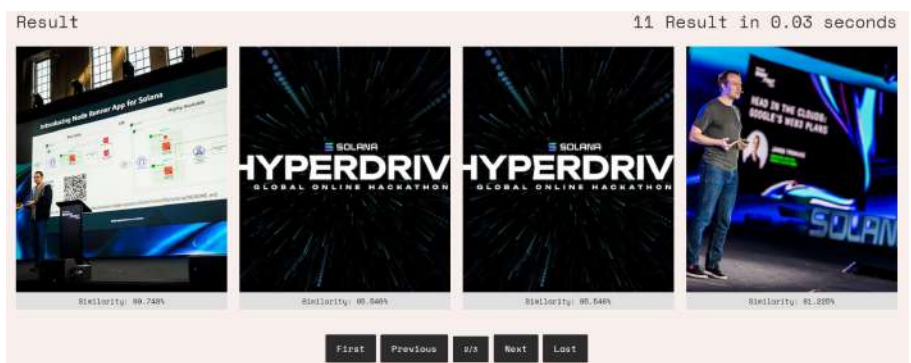
4.4.5 Pengujian fitur Web Scraping

a. Hasil Pengujian dengan Color





b. Hasil Pengujian dengan Texture



4.4.6 Pengujian Export PDF

SIMILARITY REPORT by PixPlore.

SEARCH TYPE: COLOR

INPUT IMAGE



RESULT

Result 1/237



Similarity: 100%

Result 2/237



Similarity: 78.290%

Result 3/237



Similarity: 75.780%

Result 4/237



4.5 Analisis

Dalam menganalisis desain solusi algoritma Content-Based Image Retrieval (CBIR), kita menemukan bahwa efektivitas metode ini sangat bergantung pada konteks dan karakteristik gambar. CBIR berbasis warna sangat efektif untuk gambar dimana warna merupakan fitur dominan, seperti karya seni atau fotografi lanskap, karena mampu mengidentifikasi gambar dengan kesamaan warna yang signifikan. Namun, metode ini memiliki keterbatasan dalam situasi dimana warna tidak memberikan informasi kontekstual yang cukup, atau dalam gambar dengan variasi warna yang minimal. Hal ini bisa menghasilkan hasil yang tidak relevan, terutama untuk gambar dengan tekstur halus atau tidak jelas.

Di sisi lain, CBIR berbasis tekstur unggul dalam mengenali gambar dengan pola atau tekstur khas. Tekstur memberikan kemampuan untuk mengenali objek berdasarkan detail yang mungkin tidak tercakup oleh analisis warna. Namun, metode ini mungkin kurang efektif untuk gambar dengan tekstur serupa tetapi konteks yang berbeda. Dalam konteks ini, kombinasi kedua metode dapat memberikan hasil yang lebih akurat, memanfaatkan kelebihan masing-masing pendekatan.

Untuk CBIR warna yang menggunakan metode blok, mempercepat pencarian pada dataset yang telah diproses dapat dicapai dengan *me-resize* semua gambar agar berukuran sama. Hal ini dimotivasi oleh kebutuhan untuk menghindari penghitungan ulang vektor dari semua gambar dalam dataset. Dengan *me-resize* gambar input agar sesuai ukuran yang ditetapkan, hanya vektor dari gambar input saja yang perlu dihitung, memungkinkan iterasi linear dalam pencarian gambar dengan kesamaan tertinggi. Keputusan menggunakan blok 3x3 dalam CBIR warna didasarkan pada eksperimen, di mana ukuran blok yang terlalu kecil atau terlalu besar menghasilkan sensitivitas yang tidak ideal dalam pencarian yang menyebabkan hasil pencarian yang tidak terlalu benar.

Untuk CBIR tekstur, agar mempercepat pencarian pada dataset yang sudah diproses (didapatkan vektornya), maka gambar harus dicrop sehingga berukuran sama. Kami memilih untuk *me-crop* dan tidak *me-resize* berdasarkan eksperimen, karena *me-crop* lebih mempreservasi sifat keteksturan. Alasan lain *me-crop* adalah karena agar semua gambar berukuran sama sehingga tidak ada bias ketika menghitung *cosine similarity* antara kedua vektor yang sangat bergantung kepada dimensi gambar. Kemudian, kita kalikan komponen dari vektor tekstur dengan *weight* - *weight* yang telah ditentukan dari eksperimen karena formula kontras, homogeneitas, dan entropi sangat bergantung ukuran gambar dan kontras cenderung lebih besar dari yang lainnya sebab mengandung pengkuadratan.

Secara keseluruhan, efektivitas CBIR sangat dipengaruhi oleh kualitas dan jenis gambar, seperti resolusi dan pencahayaan. Oleh karena itu, sangat penting untuk mempertimbangkan karakteristik spesifik dari dataset dan kebutuhan pengguna dalam pengembangan dan peningkatan algoritma CBIR, agar dapat menyesuaikan dan mengoptimalkan metode yang digunakan.

BAB V

SARAN DAN KESIMPULAN

5.1 Kesimpulan

Pada Tugas besar ini, kami telah menerapkan berbagai konsep vektor yang kami pelajari dalam mata kuliah Aljabar Linier dan Geometri IF2123. Kami mewujudkan konsep ini dalam sebuah situs web bernama PixPlore yang dirancang untuk mengidentifikasi kemiripan antara dua objek berbeda. Meski waktu yang kami miliki terbatas, terutama bagi pemula dalam pengembangan web, kami berhasil menciptakan algoritma yang efisien dan akurat untuk menemukan kemiripan antara dua gambar menggunakan bahasa pemrograman Python. Untuk pengembangan situs web, kami memanfaatkan HTML dan CSS untuk aspek frontend, serta Flask untuk backend.

Aplikasi ini memiliki dua fitur utama dalam CBIR (Content-Based Image Retrieval), yaitu fitur warna dan tekstur yang dapat diakses di situs web. Dengan situs ini, pengguna dapat dengan cepat menemukan kesamaan antara satu objek dan dataset besar lainnya. Selain itu, situs ini juga menyediakan fitur tambahan yang meningkatkan kemudahan penggunaan, seperti image scraping untuk menambah dataset dengan gambar dari situs tertentu, fitur webcam untuk memasukkan gambar langsung dari kamera perangkat, serta fitur ekspor ke PDF yang memungkinkan pengguna mengunduh laporan hasil pencarian untuk analisis lebih lanjut. Kami juga mengimplementasikan caching result yang disimpan di dalam JSON untuk meningkatkan efektifitas dan efisiensi ketika akan dilakukan pencarian berulang dari sebuah dataset tersebut.

Melalui proyek ini, kami memperoleh banyak pengetahuan baru, mulai dari pengembangan situs web hingga pengembangan algoritma untuk menghitung similaritas dengan vektor dan cosine similarity. Kami berharap Pixplore dapat menjadi alat yang berguna dalam berbagai aplikasi praktis dan mendorong inovasi serta penelitian lebih lanjut di bidang ini.

5.2 Saran

- Pembuatan laporan seharusnya dicicil sejak jauh hari dan dibuat secepatnya karena format laporan yang cukup kompleks dan mendetail sehingga membutuhkan effort dan waktu yang lebih besar
- Seharusnya, Pembuatan algoritma dilakukan oleh semua anggota kelompok

5.3 Komentar

- Dari Filbert : Lanjut TBFO.
- Dari Juan : Mantap.
- Dari Azmi : Lanjut Tubes lain.

5.4 Refleksi

Selama mengerjakan proyek besar ini, kami mendapatkan berbagai pengetahuan baru, termasuk cara menerapkan dan menghitung vektor, serta mengeksplorasi berbagai package dan libraries untuk mengimplementasikan fitur-fitur ekstra seperti ekspor ke PDF, image scraping, dan penggunaan webcam. Proyek ini juga meningkatkan kemampuan kami dalam pengembangan website, sebuah keterampilan yang pasti akan berguna di masa depan. Kami juga mengembangkan kemampuan kerja tim kami dalam pembuatan website dan fitur-fiturnya, memberi kami pengalaman berharga yang akan bermanfaat dalam karier profesional kami. Penggunaan Git menjadi aspek kunci dalam proyek ini, di mana kami belajar tentang penggunaannya yang efisien dan koordinasi yang baik antara anggota tim untuk mengurangi konflik. Proyek ini juga mengajarkan kami bahwa tantangan pemrograman dan perhitungan yang kompleks ternyata tidak sebesar tantangan mengembangkan website dan algoritma aplikasinya itu sendiri.

5.5 Ruang Perbaikan

Jika di masa depan terdapat rencana untuk melakukan pengembangan lebih lanjut pada aplikasi ini, salah satu area utama yang bisa ditingkatkan adalah waktu eksekusi algoritma untuk semua perhitungan. Fokus pada optimasi ini akan berkontribusi pada pengurangan waktu yang dibutuhkan untuk mengupload dan mencari gambar, sehingga meningkatkan efisiensi secara keseluruhan.

Untuk mencapai peningkatan tersebut, salah satu strategi yang dapat diadopsi adalah penggunaan bahasa pemrograman yang lebih cepat, seperti C/C++ atau Rust. Kedua bahasa ini dikenal karena efisiensi mereka dalam hal pengelolaan memori dan kecepatan eksekusi, yang dapat memberikan peningkatan signifikan dalam performa algoritma. Dengan mengimplementasikan komponen inti aplikasi dalam salah satu dari bahasa tersebut, waktu respons aplikasi dapat dikurangi secara signifikan, memberikan pengalaman pengguna yang lebih responsif dan efisien.

DAFTAR PUSTAKA

- Munir, R. (2023). Aljabar Geometri. Institut Teknologi Bandung. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/algeo23-24.html>.
- Yue, J., Li, Z., Liu, L., & Fu, Z. (2010, November 4). Content-based image retrieval using color and texture fused features. ScienceDirect. Diakses dari <https://www.sciencedirect.com/science/article/pii/S0895717710005352>
- Wang, S. (2001, October 13). A Robust CBIR Approach Using Local Color Histograms. Diakses dari https://cis.temple.edu/~lakaemper/courses/cis595_2004/papers/wang2001.pdf
- Muhammad, Y. (2020, Juli 16). Feature Extraction: Gray Level Co-Occurrence Matrix (GLCM). Diakses dari <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1>.

LAMPIRAN

- Link Repository

Link Repository kelompok kami untuk tugas besar 2 mata kuliah IF2123 Aljabar Linier dan Geometri adalah sebagai berikut:

<https://github.com/Filbert88/Algeo02-22021>

- Link Video

Link Video berikut merupakan link persembahan dari PixPlore yang diperankan oleh semua developernya. Berikut merupakan link dari video tersebut :

<https://www.youtube.com/watch?v=crFtRzulTjg>