# LAPORAN TUGAS BESAR 02
# IF 2123 ALJABAR LINEAR DAN GEOMETRI

Kelompok JUN HOK 88


Disusun oleh:

| | |
|---|---|
| Elbert Chailes | 13522045 |
| Farel Winalda | 13522047 |
| Ivan Hendrawan Tan | 13522111 |

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA (STEI)**

**INSTITUT TEKNOLOGI BANDUNG**


**2023**

# DAFTAR ISI

# BAB I
# DESKRIPSI MASALAH

Dalam era digital, jumlah gambar yang dihasilkan dan disimpan semakin meningkat dengan pesat, baik dalam konteks pribadi maupun profesional. Peningkatan ini mencakup berbagai jenis gambar, mulai dari foto pribadi, gambar medis, ilustrasi ilmiah, hingga gambar komersial. Terlepas dari keragaman sumber dan jenis gambar ini, sistem temu balik gambar (*image retrieval system*) menjadi sangat relevan dan penting dalam menghadapi tantangan ini.

Dengan bantuan sistem temu balik gambar, pengguna dapat dengan mudah mencari, mengakses, dan mengelola koleksi gambar mereka. Sistem ini memungkinkan pengguna untuk menjelajahi informasi visual yang tersimpan di berbagai platform, baik itu dalam bentuk pencarian gambar pribadi, analisis gambar medis untuk diagnosis, pencarian ilustrasi ilmiah, hingga pencarian produk berdasarkan gambar komersial. Salah satu contoh penerapan sistem temu balik gambar adalah Google Lens.

Sistem temu balik gambar bekerja dengan mengidentifikasi gambar berdasarkan konten visualnya, seperti warna dan tekstur. Aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (*Content-Based Image Retrieval* atau CBIR).

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

Content-based image retrieval (CBIR) adalah teknik yang menggunakan fitur gambar dalam melakukan pencarian gambar dalam sebuah database gambar. Teknik ini mengekstrak karakteristik visual dari gambar seperti warna, tekstur, atau bentuk untuk mengidentifikasi suatu gambar. Ada 2 cara umum dalam content-based image retrieval yaitu dengan basis warna atau basis tekstur. Basis warna merupakan metode yang paling stabil dan kokoh karena tidak terpengaruh oleh rotasi, translasi, dan dilatasi. Proses menghitung dengan basis warna juga terbilang cukup simpel. Histogram warna digunakan untuk mengekstrak warna-warna dalam gambar.

Histogram warna adalah sebuah statistik frekuensi untuk warna yang berbeda-beda dalam ruang warna tertentu. Keuntungan dari histogram warna adalah histogram ini menjelaskan penyebaran warna pada gambar secara global. Akan tetapi, ada kelemahannya juga berupa histogram ini tidak mampu untuk menjelaskan distribusi lokal dari gambar dalam ruang warna dan posisi spasial dari setiap warna. Hal ini berarti histogram warna tidak bisa menjelaskan secara spesifik benda yang ada di dalam gambar.

Untuk membagi nilai citra menjadi range yang lebih kecil, maka diperlukan pembentukan suatu ruang warna. Histogram warna dapat dihitung dari setiap piksel yang menyatakan nilai warna setiap interval. Fitur warna mencakup histogram warna global dan histogram warna blok.

Pada perhitungan histogram, warna global HSV lebih dipilih karena warna tersebut dapat digunakan pada kertas (*background* berwarna putih) yang lebih umum untuk digunakan. Maka dari itu, perlu dilakukan konversi warna dari RGB ke HSV.

Setelah mendapatkan nilai HSV, lakukan perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity* untuk menghitung tingkat kemiripannya

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

*A* dan *B* adalah vektor dan *n* adalah jumlah dimensi dari vektor

Untuk melakukan pencarian histogram warna, blok image dibagi sebesar n x n blok. Blok menjadi kurang signifikan jika ukuran blok terlalu besar dan jika terlalu kecil, akan

meningkatkan waktu untuk memprosesnya. Ukuran yang umumnya lebih efektif berupa 3 x 3 blok.

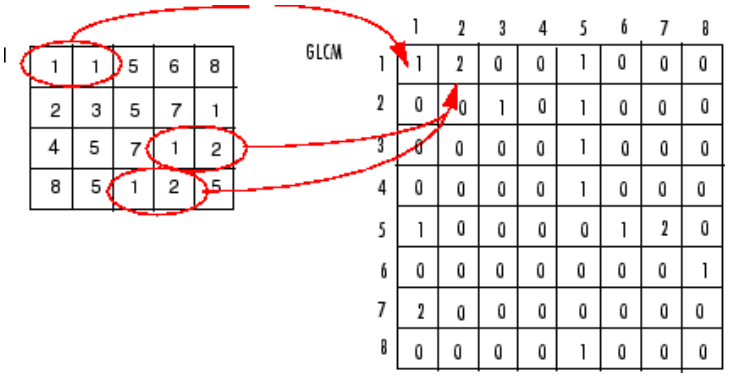
Pendekatan berbasis tekstur menggunakan suatu matriks yang bernama *co-occurrence matrix*. Matriks ini mempermudah dan mempercepat pemrosesan. Ukuran vektor yang dihasilkan juga lebih kecil.

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* $\Delta x$ dan $\Delta y$ bergantung pada arah $\theta$ 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 $\theta$ adalah 0°, 45°, 90°, dan 135°.




**Gambar 2.** Cara Pembuatan Matrix *Occurrence*

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 dalam CBIR dengan parameter tekstur adalah sebagai berikut :

1. Konversi warna gambar menjadi *grayscale* karena warna tidaklah penting dalam penentuan tekstur. 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 juga akan berukuran 256 × 256. Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut. Nilai *grayscale* pada gambar awal akan dikompres untuk mengurangi langkah perhitungan sebelum matriks *co-occurrence* dibentuk.
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 :

   *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 :*

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

   **Keterangan : *P* merupakan matriks *co-occurrence***

   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:

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

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

# BAB II LANDASAN TEORI

## 2.1 Dasar Teori

*Content Based Image retrieval* (CBIR) adalah teknik yang menggunakan fitur gambar (visual contents) dalam melakukan pencarían gambar dalam sebuah database gambar. Proses ini mengekstrak warna, tekstur, atau bentuk dari sebuah gambar. Hasil ekstrak ini lalu diubah menjadi bentuk vektor. Parameter yang kita gunakan kali ini adalah Parameter warna dan Parameter tekstur.

CBIR parameter warna memanfaatkan histogram warna untuk setiap piksel dari sebuah gambar. Histogram warna ini menghitung frekuensi kemunculan warna yang ada di sebuah ruang warna tertentu. Penghitungan histogram warna memiliki 2 cara yaitu secara global atau blok.

Histogram warna global memberikan gambaran mengenai distribusi warna pada gambar. Cara ini efektif untuk menangkap gambaran secara umum tetapi lemah dalam mengidentifikasi pola atau lokasi spesifik dari warna-warna tersebut. Dalam kata lain, gambar yang berbeda dapat dianggap mirip jika kedua gambar memiliki distribusi warna yang mirip satu sama lain. Namun, terdapat metode warna yang lebih akurat yaitu dengan melakukan pembagian matriks gambar menjadi 3x3 blok sehingga proses perbandingan warna tidak dilakukan secara global melainkan sesuai segmentasi gambar yang satu dengan gambar yang lainnya. Metode histogram warna 3x3 akan memenuhi kelemahan warna global yaitu dapat mengidentifikasi pola atau lokasi spesifik dengan lebih baik.

CBIR parameter tekstur biasanya menggunakan matriks *co-occurrence* untuk mengukur frekuensi pasangan piksel yang muncul bersamaan dalam sebuah gambar pada tingkat keabuan tertentu dan jarak serta arah yang spesifik. Dari data ini, kita dapat menghitung parameter-parameter tekstur seperti kontras, entropi, dan homogenitas. Parameter tersebut memberikan info mendalam mengenai karakteristik permukaan dan pola dari gambar.

Cara ini sangat berguna dalam mengidentifikasi tekstur yang ada di dalam gambar yang tidak bisa dicari dengan CBIR parameter warna. Cara ini juga biasanya dikombinasikan dengan CBIR parameter warna untuk membuahkan hasil yang lebih akurat sehingga hasil yang diberikan lebih dekat dan alami terhadap persepsi mata manusia.

## 2.2 Teori Pengembangan Website

Pengembangan sebuah website pada umumnya memerlukan banyak langkah, namun hal ini dapat bervariasi tergantung dari tipe website, sumber daya yang dimiliki, dan bahasa pemrograman yang digunakan. Pada umumnya, ada 5 tahap dalam mengembangkan sebuah website.

1. **Tahap perencanaan**

Tahap perencanaan sering juga disebut tahap permintaan sistem. Permintaan ini dapat berupa sistem yang baru atau pembaruan dari sistem yang lama. Tahap ini memerlukan identifikasi masalah dan strategi untuk rencana pengembangan kedepannya.

2. **Tahap analisis kebutuhan**

Tujuan utama dari tahap ini yaitu untuk memahami kebutuhan utama dari pembuatan website ini. Hasil analisis ini dapat berupa kebutuhan manajemen dan pengguna; rencana alternatif dan anggaran dalam mengembangkan website ini; dan solusi yang diberikan dari seorang konsultan sebagai ahli pengembangan web.

3. **Tahap desain**

Tahap desain akan mengidentifikasi keseluruhan aspek dari suatu website seperti input dan output, *interface*, dan proses pengerjaan pada backend website tersebut. Hal ini bertujuan agar sistem dapat diandalkan, akurat, mudah dirawat, dan aman. Beberapa faktor yang harus dipertimbangkan dalam mendesain situs web adalah pengaturan konten halaman web, pemilihan struktur situs web yang sesuai, dan menangani masalah aksesibilitas.

Situs web dapat menggunakan beberapa jenis struktur seperti linier, hirarki, dan jejaring. Struktur situs web dapat dinamakan struktur linier jika informasi pada halaman web terbaca sesuai urutan. Struktur linier digunakan dalam informasi pada halaman web pertama berguna untuk masuk ke halaman web kedua. Di sini, setiap halaman memiliki link dari satu halaman web ke halaman berikutnya

4. **Tahap pengembangan dan implementasi**

Tujuan dari tahap ini yaitu untuk membuat situs web yang diinginkan. Situs web dikembangkan dengan menyesuaikan metodologi yang dipilih sebelumnya.

Proses diawali dengan menulis program, menguji, dan setelah berhasil dikembangkan, dapat di host di server yang diinginkan. Hasil dari fase ini adalah:

- Sebuah situs web telah diproduksi harus siap digunakan
- Melakukan evaluasi sistem. memastikan bahwa situs web yang dikembangkan mampu beroperasi sesuai keinginan pengguna
- Memastikan bahwa biaya dan manfaat dari situs yang dikembangkan sesuai dengan perkiraan.

**5. Tahap pemeliharaan**

Pada tahap ini, pengembang website tersebut akan melakukan pemeliharaan atau *maintenance* secara berkala serta melakukan pengembangan fitur tambahan jika diperlukan. Pengembang harus memastikan bahwa website yang telah dikembangkan memiliki fungsionalitas yang baik ketika digunakan oleh penggunanya.

# BAB III
# ANALISIS PEMECAHAN MASALAH

## 3.1. Langkah-langkah pemecahan masalah

### 3.1.1. CBIR Parameter Color

Langkah-langkah yang dilakukan dalam pemecahan masalah dalam menghitung kemiripan adalah dua gambar, yaitu :

1. Melakukan normalisasi warna RGB pada gambar yang dari range [0,255] menjadi [0,1]
2. Mencari nilai *Cmax*, *Cmin*, dan Δ dari matriks RGB yang telah dinormalisasi yang disimpan pada setiap *pixel*
3. Melakukan konversi setiap pixel tersebut sesuai dengan perhitungan yang terdapat teori dasar yang kemudian disimpan dalam matriks HSV yang terdiri derajat Hue yang memiliki *range* [0,360], *Value* [0,1], dan *Saturation* [0,1]
4. Lakukan pembagian matriks HSV menjadi ukuran 3x3 blok sehingga akan dihasilkan sembilan matriks HSV yang berbeda sesuai dengan segmentasinya masing-masing
5. Lakukan alokasi bin sesuai dengan range yang telah ditentukan pada teori singkat bahwa *Hue* akan dibagi ke dalam 8 bin, *Saturation* akan dibagi ke dalam 3 bin, dan *Value* akan dibagi ke dalam 3 bin. Sehingga, akan terbentuk 72 kombinasi yang akan dialokasikan pada sebuah representasi vektor atau histogram. Karena 9 blok, maka akan dilakukan 9 kali alokasi, dengan hasil akhir 9 vektor dengan 72 elemen pada setiap vektornya.
6. Kemudian, perbandingan 9 vektor tersebut antara gambar yang satu dengan gambar yang lainnya menggunakan *cosine similarity*. *Cosine similarity* yang digunakan mengandung sifat pemberatan, yang memberikan nilai yang lebih tinggi pada objek yang berada di tengah gambar. Rasio pemberatan dapat disesuaikan dengan kebutuhan pengguna masing-masing dalam melakukan CBIR metode *color*.

### 3.1.2. CBIR Parameter Texture

Langkah-langkah yang dilakukan dalam pemecahan masalah dalam menghitung kemiripan adalah dua gambar, yaitu :

1. Mengubah nilai *RGB* tiap *pixel* menjadi *Grayscale*
2. Membuat / mengekstraksi gambar *grayscale* ke dalam *co-occurrence matrix* dengan menggunakan parameter $d = 1, \theta = 0°$
3. Membuat *symmetric matrix* dengan menjumlahkan nilai *co-occurrence matrix* dengan transposenya
4. Membuat *matrix normalization* dengan membagi *symmetric matrix* dengan total dari semua elemen dari *symmetric matrix*
5. Menghitung nilai dari *contrast, homogeneity,* dan *entrophy* dari *matrix normalization* dengan rumus:
6. Membuat vektor 3 dimensi yang berisikan nilai dari *contrast, homogeneity,* dan *entrophy*
7. Membandingkan kemiripan kedua gambar dengan menggunakan rumus cosine similarity dengan vektor *CHE* yang telah ditentukan

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

### 3.2.1. CBIR Parameter Color

Langkah penyelesaian masalah dimulai dengan mencari nilai RGB dari setiap pixel dalam sebuah gambar dan kemudian melakukan normalisasi terhadap nilai-nilai tersebut. Normalisasi tersebut dilakukan dengan menggunakan perhitungan berikut.

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

Lalu, hasil nilai normalisasi tersebut kemudian dilakukan pencarian nilai *Cmax*, *Cmin*, dan $\Delta$ dari matriks RGB yang telah dinormalisasi yang disimpan pada setiap *pixel*. Kemudian, nilai-nilai tersebut dapat diolah untuk diubah menjadi matriks HSV yang memiliki ukuran seperti gambar aslinya. Perhitungan dapat dilakukan dengan perhitungan berikut.

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

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

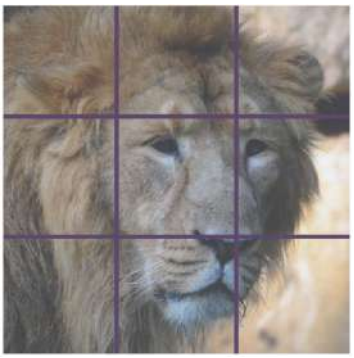
$$\Delta = C_{max} - C_{min}$$

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

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

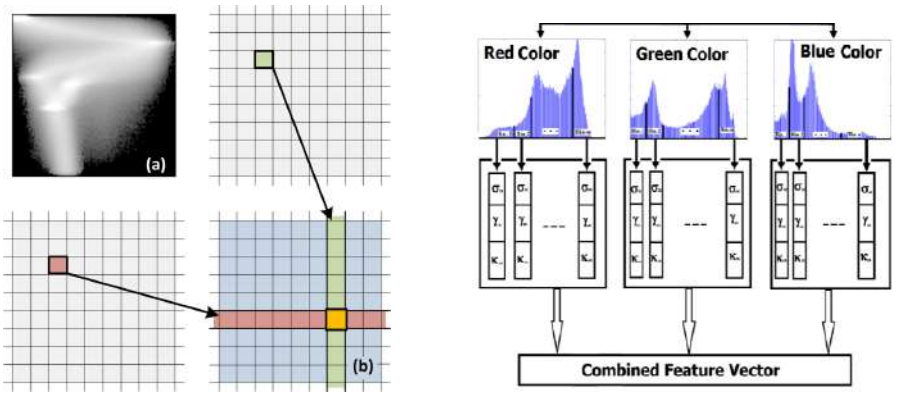
$$V = Cmaks$$

Kemudian, matriks HSV tersebut dibagi menjadi 3x3 blok sehingga terbentuk total sembilan matriks yang berbeda tergantung segmentasinya masing-masing seperti yang dapat terlihat pada gambar.



***Gambar 3.2.1*** *Gambar singa yang dilakukan pembagian blok 3x3*

Setiap blok gambar tersebut akan dilakukan alokasi dengan melakukan pembagian bin terhadap masing-masing *Hue, Saturation,* dan *Value* sehingga terbentuk vektor dengan 72 elemen untuk masing-masing blok tersebut.




***Gambar 3.2.2*** *Mekanisme alokasi bin kemudian direstorasi pada representasi vektor*

Setelah mendapatkan representasi vektor untuk masing-masing blok dengan total 9 vektor dengan 72 elemen masing-masing vektornya. Maka, vektor-vektor tersebut dapat dibandingkan dengan 9 vektor yang berasal dari gambar lainnya. Maka, dengan itu perbandingan akan menjadi lebih sebanding, yaitu bagian atas kiri gambar satu dengan gambar lainnya, serta berlaku untuk delapan segmentasi gambar lainnya. Untuk menghitung similaritas antara dua gambar, dapat dilakukan *cosine similarity* antara sembilan vektor tersebut kemudian pengolahan tergantung pada intensi penggunaan. Terdapat opsi untuk melakukan *averaging* dengan membagi seluruh jumlah hasil *cosine similarity* dengan banyak jumlah vektor atau dengan melakukan *weighting* terlebih dahulu untuk perbandingan segmentasi gambar tertentu. Sehingga, untuk masalah perhitungan similaritas harus disesuaikan dengan kebutuhan dan jenis gambar yang digunakan. Perhitungan *cosine similarity* dapat menggunakan perhitungan berikut.

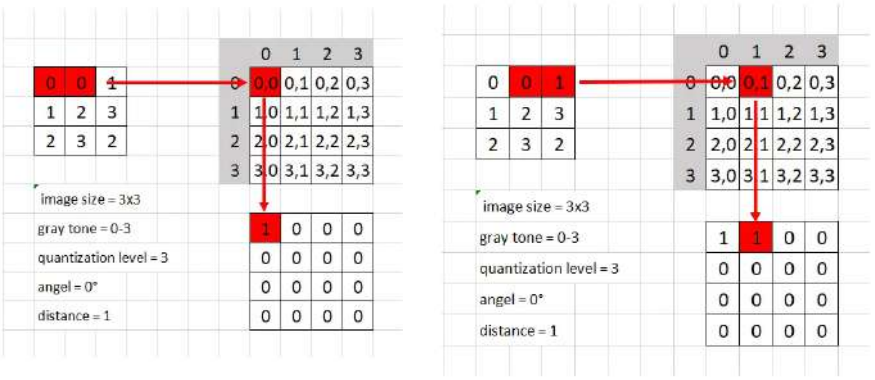
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

### 3.2.2. CBIR Parameter Texture

Langkah penyelesaian masalah dimulai dengan mencari nilai *RGB* dari setiap pixel dalam sebuah gambar, Misal gambar berukuran 400 x 600, maka akan diubah menjadi matriks *grayscale* yang berukuran sama dengan rumus, seperti berikut :

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

Lalu, setelah didapatkan nilai *grayscale* dari gambar, akan dipetakan dan diekstraksi ke dalam sebuah matriks yang berukuran 256 x 256 yang menandakan jumlah dimensi dari nilai *grayscale* , dengan cara berikut :




***Gambar 3.2.3*** *Pemetaan Gray Tone untuk membentuk co-occurrence matrix*

Hal tersebut dilakukan hingga semuanya terisi, misal ukuran matriks 300 x 400, maka akan dilakukan pemetaan sebanyak 300 x 399 kali untuk membentuk *co-occurrence matrix*.

Setelah dibentuk *co-occurrence matrix* akan dibuat *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan transpose diri nya. Hasil dari *symmetric matrix* dapat digunakan untuk menentukan nilai *matrix normalization*. Dapat digunakan rumus

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

Setelah didapatkan *matrix normalization*, dapat ditentukan nilai dari *contrast, homogeneity,* dan *entrophy* dengan menggunakan rumus, sebagai berikut :

*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 :*

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

Nilai dari ketiga komponen tersebut akan diubah menjadi sebuah vektor 3 dimensi yang akan digunakan untuk menentukan nilai kemiripan antara 2 gambar dengan menggunakan metode *cosine similarity* dengan rumus:

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

Lalu, nilai dari *cosine similarity* tersebut akan dikali dengan 100 yang menunjukkan berapa persen tingkat kemiripan antara 2 gambar tersebut.

## 3.3. Contoh Kasus dan Penyelesaiannya

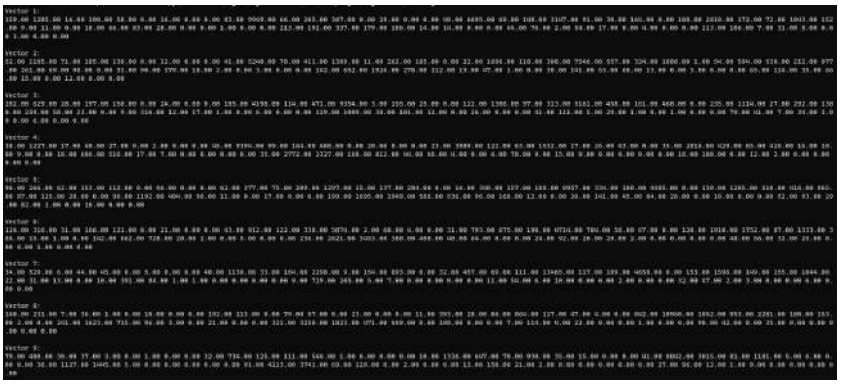
### 3.3.1. CBIR Parameter Color

Langkah-langkah yang dilakukan dalam pemecahan masalah dalam menghitung kemiripan adalah dua gambar, yaitu :

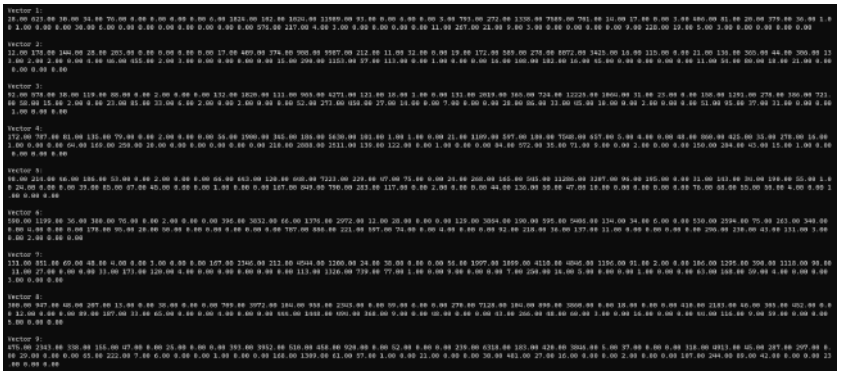
***Gambar 3.3.1*** *Sampel Harimau 1*

***Gambar 3.3.2*** *Sampel Harimau 2*

Hasil alokasi bin yang kemudian disimpan pada 9 vektor yang berisi 72 elemen dapat dilihat pada gambar yang terlampir.



***Gambar 3.3.3*** Hasil perhitungan 9 vektor untuk gambar sampel Harimau 1

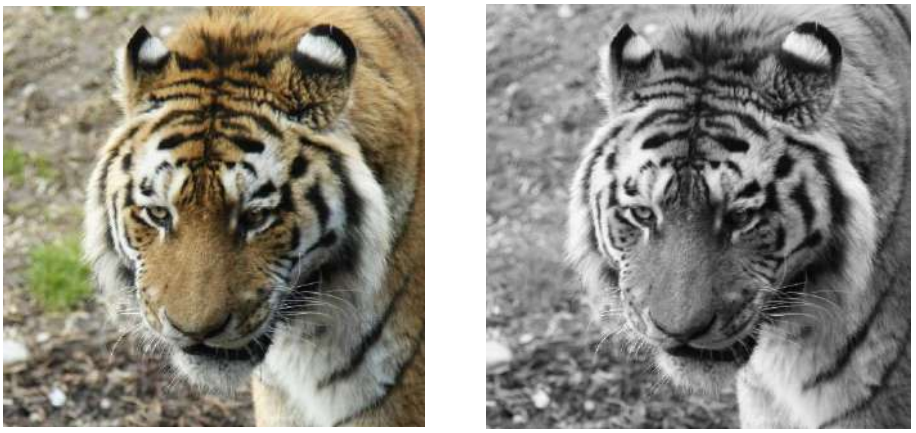


***Gambar 3.3.4*** Hasil perhitungan 9 vektor untuk gambar sampel Harimau 2

Setelah didapatkan nilai dari kedua vektor, maka dapat ditentukan pula *cosine similarity* antara vektor-vektor dari kedua gambar tersebut yang menghasilkan nilai sebesar 58.48%. Meski kedua gambar memiliki objek yang sama, yaitu harimau. Namun, warna yang dimiliki oleh kedua gambar cukup berbeda terutama di bagian lingkungan, dan segmen kiri bawah dari kedua gambar sampel.

### 3.3.2. Contoh Kasus CBIR Texture

Diberikan contoh gambar harimau yang akan dicari nilai vektor nya, dengan menggunakan parameter *Texture*, dengan mengubah terlebih dahulu menjadi *grayscale*



***Gambar 3.3.5*** *Harimau Oranye yang diubah menjadi grayscale*

Setelah dijadikan *grayscale* dan diekstraksi ke dalam *co-occurrence matrix* berukuran 256 x 256, yang selanjutnya akan dibentuk *normalization matrix*. Sehingga didapatkan hasil *vector* 3d dari *contrast,homogeneity,* dan *entropy*, yaitu [C : 136.91393, H : 0.16525963, E : 12.990985].

Lalu menentukan gambar pembanding yang akan dicari juga nilai vektor nya, contoh :

***Gambar 3.3.6*** *Harimau Oranye 2*

Gambar tersebut memiliki nilai vektor *contrast, homogeneity*, dan *entrophy* yaitu sebesar [C : 366.096, H : 0.2265042, E : 12.982778].

Setelah didapatkan nilai dari kedua vektor, maka dapat ditentukan pula *cosine similarity* yang memiliki nilai sebesar 99.83% tingkat kemiripan yang menandakan bahwa kedua gambar tersebut sangat mirip.

# BAB IV
# IMPLEMENTASI DAN UJI COBA

## 4.1. Implementasi Program Utama

### 4.1.1. CBIR Fitur Warna

```
Program colorCBIR

// Inisialisasi tipe-tipe yang akan dibutuhkan dalam program
  Define structure PixelRGB with fields R, G, B, cmax, cmin, delta, maxtype
  Define structure ImageRGB with field ValueRGB (2D array of PixelRGB), col, row
  Define structure PixelHSV with fields H, S, V
  Define structure ImageHSV with field ValueHSV (2D array of PixelHSV), col, row

  Function loadImage(input: filename string) return Image
  // Fungsi ini bertujuan untuk membaca image dari filename dan mengubahnya ke dalam bentuk decoded
     Open image file
     Decode image file
     return decoded image

  Function getNormalizedRGBValues(input: img Image) return ImageRGB
  // Fungsi ini bertujuan matriks hasil normalisasi RGB gambar yang dimasukkan
     Create 2D matrix of PixelRGB
     For each pixel in img
      Normalize and store RG values in the 2D matrix of PixelRGB
     return ImageRGB

  Function convertRGBToHSVValues(input: ImageRGB imgRGB, output: imgHSV ImageHSV)
```

```
  // Fungsi ini untuk melakukan konversi matriks RGB ke matriks
HSV
     Convert for every RGB pixel to HSV and store in imgHSV

  Function divideHSVMatrixTo9Vectors(input: hsvMatrix
ImageHSV, output: vector [9][72] array)
  // Fungsi ini bertujuan untuk mengubah matriks HSV ke dalam 9
vector dengan 72 elemen
     Divide hsvMatrix into 9 equal parts
     Convert each part into a vector representation with
convertHSVToVector function
     Append and store it one by one into the array of vector

  Function ArrayOfVectorCosineWeighting(input: vector1,
vector2 [9][72] array) return float
  // Fungsi ini bertujuan untuk melakukan cosine weighting
antara dua vektor sehingga mendapatkan hasil similaritas yang
telah dilakukan pemberatan
     Compute cosine similarity for each corresponding vector
in vector1 and vector2
     Calculate weighted average of these similarities //
Pemberatan dilakukan dengan pertimbangan gambar yang
dibandingkan
     Return the average similarity

   Function convertHSVToVector(input: imgHSV ImageHSV, output:
vector array[72])
     Replace every pixel of imgHSV based on the bin range and
store it to the vector based on the index // Pembagian range
untuk dimasukkan ke dalam bin berdasarkan index disesuaikan
dengan preferensi yang dimiliki

   Function cosineSimilarity(vecA, vecB: array): float
  // Operasi untuk melakukan perhitungan cosinus di antara dua
vektor
     Compute and return the cosine similarity between vecA and
vecB
   Function ColorProcessing(img: Image): array
```

```
  // Fungsi yang akan digunakan dalam main program sehingga
hanya mengeluarkan vektor yang kemudian akan diproses di
backend
     Convert image from RGB to HSV
     Divide HSV image into 9 vectors
     Return the vector representation

End Program
```

### 4.1.2. CBIR Fitur Tekstur

```
Program teksturCBIR

// Inisialisasi tipe-tipe yang akan dibutuhkan dalam program
  Define structure TexturePixel with fields Y
  Define structure ImageTexture with field TexturePixel (2D
array of PixelGrayscale), col, row
  Define structure VectorCHE with fields C, H, E

  Function getTextureValues(input: img Image) return
ImageTexture
  // Fungsi yang bertujuan mendapatkan nilai grayscale dari
sebuah image
     Create 2D matrix of pixelRGB
     For each pixel in img
       Store PixelGrayscale to the 2D matrix of ImageTexture
       // Grayscale = R*0.29 + G*0.587 + B*0.114
     return ImageTexture

  Function occurrence(input: pixel ImageTexture) return 2D
256*256 matrix of float
  // Fungsi ini bertujuan membuat matriks normalisasi dengan
membuat matriks co-occurrence
     Create 2D 256*256 matrix of float
     Create total int
     For each pixel in img
```

```
        Store Y value of current pixel in nrow
        Store Y value of next pixel same row in ncol
        Increment value matrix[nrow][ncol] and
matrix[ncol][nrow]
     // memetakan pixel ke matriks simetrik sekaligus membuat
matriks co-occurrence
        Increment total by 2
     For each row and col in matrix
        Divide value matrix[row][col] with total
     // membuat matriks normalisasi dengan membagi nilai matrix
co-occurrence dengan total semua nilai
     return 2D 256*256 matrix of float

  Function CHE(input: 2D 256*256 matrix of float) return
VectorCHE
  // Fungsi ini untuk mendapatkan nilai vektor berisi nilai
contrast, homogeneity, dan entrophy
     Create totalC float
     Create totalH float
     Create totalE float
     For each row and col in matrix
        Increment totalC with matrix[row][col]*(row-col)^2
        Increment totalC with matrix[row][col]/(1+(row-col)^2)
        Decrement totalC with
matrix[row][col]*log(matrix[row][col])

     return VectorCHE{totalC,totalH,totalE}

  Function TextureProcessing(input: img Image) return
VectorCHE
  // Fungsi ini untuk memproses image menjadi vektor
     Store getTextureValue(img)to imgTexture
     Store occurrence(imgTexture)to occ
     Store CHE(occ)to vectorCHE
      // menggunakan fungsi-fungsi yang telah dibuat sebelumnya
     Return VectorCHE
```

```
   Function TextureSimilarity(input: vector1 VectorCHE,
vector2 VectorCHE) return float
 // Fungsi ini untuk mencari similarity % antara 2 gambar yang
sudah diekstrasi dalam bentuk vektor contrast, homogeneity, dan
entrophy
     Store dotproduct with vector1.C*vector2.C +
vector1.H*vector2.H + vector1.E*vector2.E
     Store length1 with sqrt((vector1.C)^2 + (vector1.H)^2 +
(vector1.E)^2)
     Store length2 with sqrt((vector2.C)^2 + (vector2.H)^2 +
(vector2.E)^2)
     // mencari cosSimilarity
     Return dotproduct/(length1*length2)*100

End Program
```

## 4.2. Struktur Program


Penjelasan struktur program berdasarkan spesifikasi (dapat membahas terkait arsitektur kode secara keseluruhan, struktur pembangunan *website*, atau hal-hal lain yang berkaitan dengan struktur program).

Struktur program dalam pembangunan aplikasi berbasis website ini terdiri dari beberapa komponen utama. Di sisi frontend, aplikasi ini menggunakan framework React JS dan memanfaatkan Tailwind CSS untuk merancang antarmuka pengguna

(UI) website. Bahasa dasar yang digunakan untuk mengembangkan bagian frontendnya adalah JavaScript. Aplikasi ini juga menggunakan berbagai library yang disediakan oleh ReactJS, seperti ikon untuk memenuhi kebutuhan desain UI, serta library lainnya yang mendukung fungsi-fungsi tambahan, seperti pembuatan PDF.

Di sisi backend, aplikasi ini menggunakan bahasa pemrograman Go dan memanfaatkan framework Fiber. Ini membantu aplikasi dalam mengelola logika yang berjalan di balik website serta memungkinkan aplikasi untuk menyediakan layanan backend yang responsif dan efisien.

Berikut adalah struktur program yang lebih rinci terkait struktur yang terdapat dalam gambar terlampir.

1. ***package-lock.json*** dan ***package.json*** berisi package-package yang diinstall dengan menggunakan node package manager (npm) sehingga agar terinstal secara otomatis pada folder direktori proyek.
2. ***.gitignore*** berisi berkas konfigurasi yang digunakan untuk mengontrol versi git untuk menentukan file dan direktori mana yang hendak diabaikan.
3. **README.md** berisi informasi dan tata cara penggunaan yang hendak disampaikan oleh pengembang website ini, atau bisa dikatakan sebagai *guide* baik dalam penggunaan atau dalam melakukan penginstalan.
4. ***tailwind.config.js*** berisi berkas konfigurasi atas hasil instalasi atau perlakukan *setup* Tailwind CSS terhadap proyek ini. Dalam file ini, juga dapat dilakukan perubahan atau modifikasi pada konfigurasi untuk disesuaikan dengan keperluan.
5. Folder ***doc*** berisi berkas hasil pertanggungjawaban atas pembuatan website ini untuk pemenuhan tugas besar atau dengan kata lain pada folder *doc* akan berisi laporan yang dibuat ini.
6. Folder ***img*** berisi gambar-gambar yang digunakan dengan tujuan untuk melakukan testing secara lokal terkait efisiensi dan hasil dari *output* similaritas program yang telah dibuat. Folder ini berisi *dataset* serta sampel-sampel gambar yang dapat digunakan untuk melakukan uji coba *low-fidelity* terhadap logika website.

7. Folder ***src*** berisi kode-kode utama dari pembangunan website ini, yang terdiri dari bahasa pemrograman *Go* dan *Javascript*. Perincian dari folder tersebut terdiri dari.
    a. ***go.mod*** dan ***go.sum*** merupakan berkas atau file yang berisi konfigurasi utama dalam proyek ini dan berisi modul-modul yang digunakan dalam proyek ini. Ini merupakan bagian penting dalam manajemen dependensi.
    b. ***index.css*** dan ***index.js*** merupakan dasar dari pembangunan aplikasi React JS sehingga pemrograman atau penampilan website secara pusat dilakukan pada file ini. Hal yang sama juga dilakukan pada *index.css* dalam segi *styling*.
    c. Folder ***api*** merupakan folder yang berisi file dalam pembuatan *Application Programming Interface* (API) yang digunakan sebagai media komunikasi antara frontend dan backend. Folder ***api*** dirincikan sebagai berikut.
        i. File ***main.go*** berisi kode program yang bertujuan untuk membuat api dan memproses segala hal yang berhubungan dengan fitur backend. Maka dari itu untuk menjalankan backend dari website ini, harus melakukan kompilasi atas kode program tersebut yang kemudian akan dijalankan di port yang telah ditentukan.
    d. Folder ***components*** merupakan folder yang berisi komponen-komponen kecil yang digunakan untuk membangun frontend atau UI dari website.
    e. Folder ***pages*** merupakan folder yang berisi kode program untuk tampilan halaman website, yang terdiri dari semua halaman yang akan ditampilkan, serta tempat dimana dilakukan *routing* untuk menyambungkan semua website agar dapat berjalan ketika react app dijalankan.
    f. Folder ***program*** merupakan folder yang berisi kode-kode program dengan bahasa pemrograman *Go* terkait algoritma dari implementasi CBIR dengan metode color dan texture, serta algoritma untuk

melakukan *image scraping*. Ketiga ini dijadikan dalam satu package, yaitu *package program* yang kemudian akan digunakan dalam kode *main.go* ketika menjalankan backend dari website sebagai logika website yang berjalan.

8. Folder ***test*** merupakan folder yang berisi hasil uji coba dari aplikasi similaritas website yang telah digunakan. File-file yang terdapat dalam folder *test* ini akan berbentuk format PDF, yang masing-masingnya berisi laporan atas hasil uji coba yang telah digunakan.

### 4.3. Cara penggunaan program

Untuk menjalankan aplikasi MatchLens, dibagi menjadi 2 bagian yaitu untuk bagian frontend, yaitu menggunakan *React* dengan bahasa pemrograman *Javascript* dan *Tailwind CSS*, lalu untuk bagian backend, yaitu menggunakan bahasa pemrograman *GoLang*

Untuk bagian pertama, yaitu frontend atau tampilan website diperlukan untuk menginstal beberapa keperluan, seperti *HTML5*, *CSS3*, *Javascript (ES6+)* atau bisa juga menggunakan di Browser seperti *Google Chrome*, *Microsoft Edge*, *Firefox*, atau browser lainnya yang bisa support keperluan seperti yang telah disebutkan. Dalam menjalankan website, lalu dapat diikuti cara-cara berikut ini:

1. Menginstall package manager *npm*

   Dalam menggunakan beberapa fitur atau package diperlukan *npm* sebagai package manager yang dapat diinstal melalui website resmi npm.

2. Menginstall *Node.Js* untuk bisa menjalankan *React*

   Lalu untuk menggunakan react diperlukan instalasi *Node.Js* dengan versi terbaru yang dapat diinstall melalui website resmi *Node.Js*.

3. Menginstall semua package *React*

   Setelah instalasi npm dan *Node.Js*, perlu dilakukan instalasi semua package yang digunakan aplikasi MatchLens dengan menjalankan command berikut pada terminal di directory aplikasi MatchLens :

   ```
   npm install
   ```

4. Menjalankan web di server localhost

Setelah instalasi package, aplikasi dapat dijalankan dengan menggunakan terminal dengan perintah yang nantinya akan dijalankan di server localhost port 3000:

```
npm run start
```

Untuk bagian backend diperlukan instalasi bahasa pemrograman Go melalui website resmi *Golang*. Setelah menginstall keperluan, bisa dijalankan cara berikut untuk menjalankan backend:

1. Menginstall semua package *Go*

   Setelah instalasi bahasa pemrograman Go, dapat dijalankan aplikasi backend dengan menginstall semua package / framework yang diperlukan, yaitu dengan masuk ke directory src :

   ```
   cd src
   go mod tidy
   ```

2. Menjalankan server backend di localhost

   Setelah menginstal package yang dibutuhkan program dapat dijalankan server backend tempat pemrosesan input pengguna di server backend yang akan dijalankan di server localhost port 8080 dengan melakukan perintah:

   ```
   cd src/api
   go run ./main.go
   ```

dalam aplikasi MatchLens terdapat beberapa page / halaman yang bisa diakses oleh pengguna yang pertama dari *landingpage* yang berisi tampilan website dan penjelasan singkat fitur dari aplikasi MatchLens, lalu page Guides yang berisi cara pemakaian aplikasi, Technology yang berisi *tech stack* / *tech tools* yang digunakan, Developers yang berisi informasi-informasi terkait pembuat aplikasi MatchLens dan Aplikasi utama yang dapat digunakan untuk mendeteksi tingkat kemiripan suatu gambar dengan gambar lainnya, menjalankan fitur kamera, dan juga melakukan *web scraping.*

Pada halaman aplikasi utama, pengguna dapat mengunggah gambar yang ingin dibandingkan berupa format *jpg*, *jpeg*, dan *png*. Lalu, pengguna dapat mengunggah dataset sebagai pembanding untuk mencari gambar mana yang memiliki tingkat kemiripan paling tinggi dengan foto utama. Format dataset yaitu berupa *jpg*, *jpeg*, dan

*png* atau berupa *folder*. Setelah mengunggah dua kebutuhan tadi dapat menekan tombol *search* dan program akan mencari tingkat kemiripan dan akan ada tampilan gambar, banyak gambar yang diolah, dan waktu pemrosesan *file* dan *dataset*.

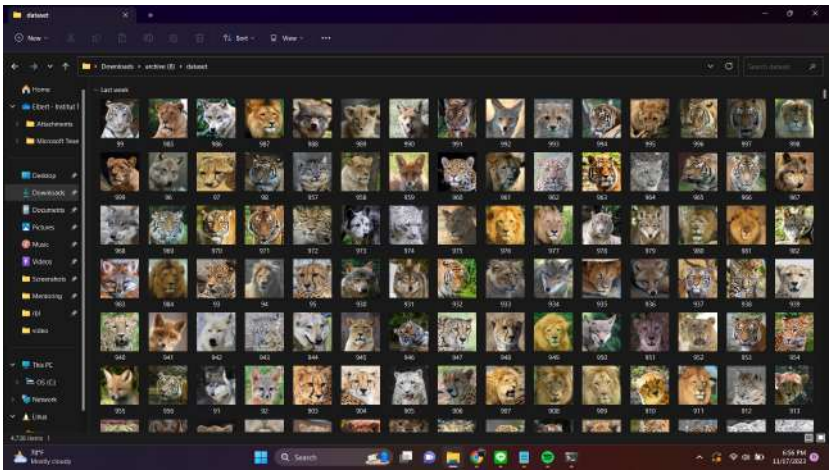
Pengguna juga bisa menggunakan fitur *web scraping*, yang digunakan sebagai pengganti dataset. Akan tetapi, tidak semua *website* mengizinkan untuk mengakses data *image* yang mereka punya. Lalu jika berhasil memilih *website*, aplikasi akan menampilkan tingkat kemiripan foto yang dibandingkan dengan gambar-gambar yang ada dalam web yang ingin di *scraping* tersebut.

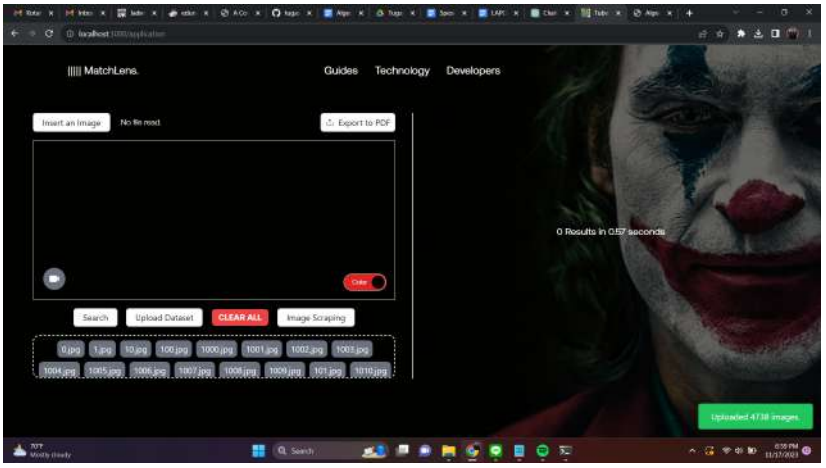
Lalu pengguna juga bisa melakukan pengambilan gambar melalui kamera masing-masing dengan menyalakan tombol menyalakan kamera dan kamera akan otomatis mengambil gambar setiap 10 detik saat kamera dihidupkan dan dapat dijadikan sebagai foto yang akan dibandingkan dengan dataset maupun data *scraping*.

## 4.4. Hasil pengujian

### 4.4.1. Dataset yang digunakan

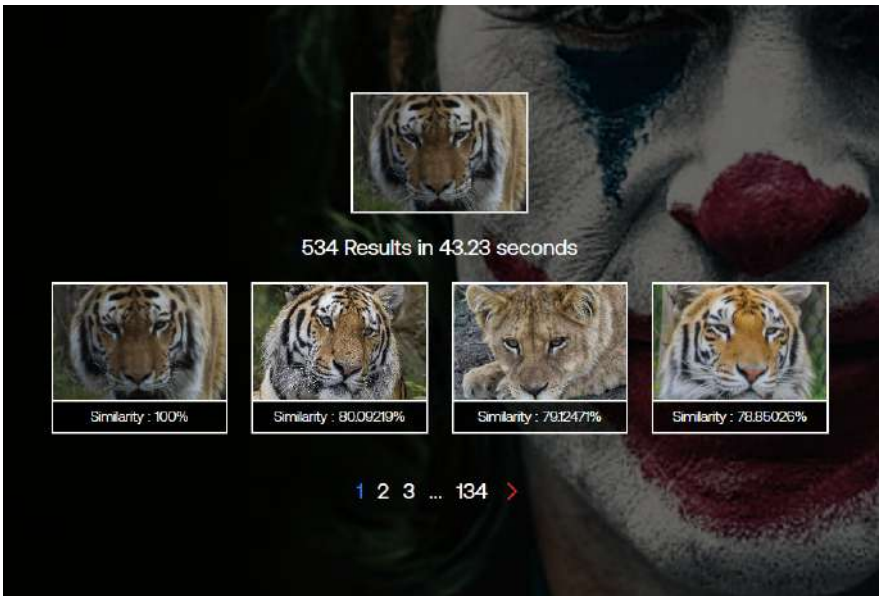
Folder dataset yang digunakan terdapat pada folder ***img*** yang terdapat pada repository github. Pada dataset ini, sampel-sampel yang digunakan merupakan hewan yang telah disediakan pada spesifikasi penugasan. Pada dataset ini terdapat secara spesifik 4.738 gambar dengan jenis hewan yang bervariasi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

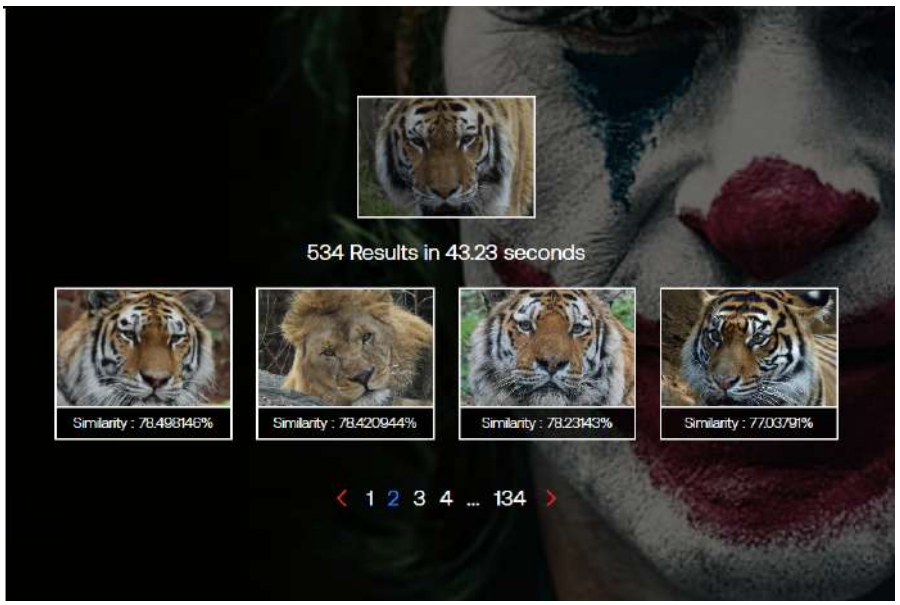



MatchLens.
Guides
Technology
Developers
Insert an image
No file read.
Export to PDF
Search
Upload Dataset
CLEAR ALL
Image Scraping
0 Results in 0.57 seconds
Uploaded 4738 images

### 4.4.2. Pengujian dengan Metode Color

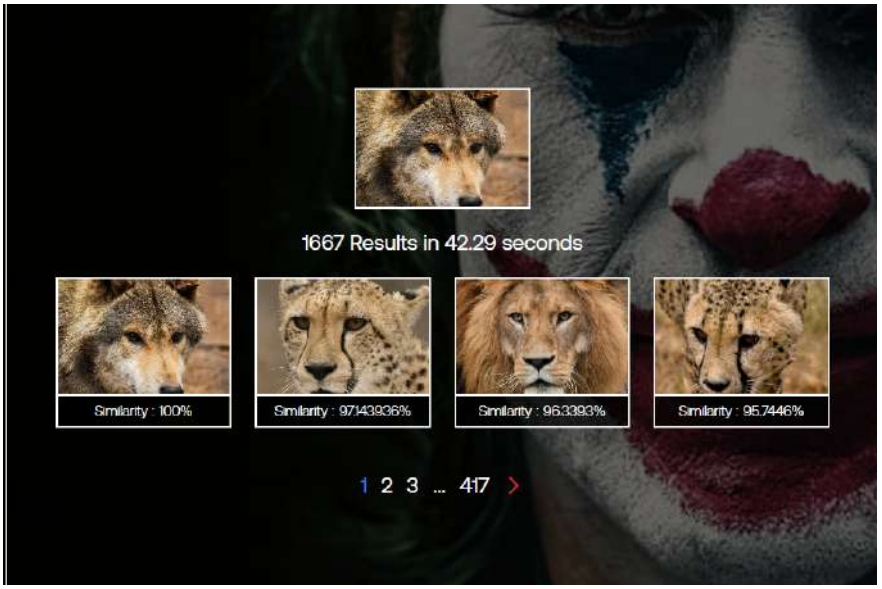
#### a. Hasil Pengujian I

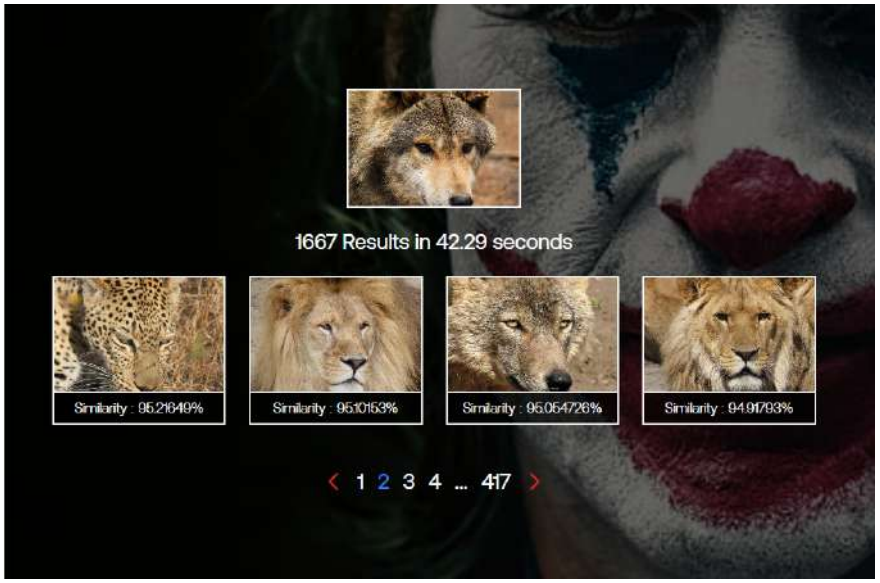



Dengan menggunakan metode *color*, harimau yang dijadikan sampel ketika dibandingkan dataset yang telah diinput pada gambar yang terlampir sebelumnya mengeluarkan hasil gambar yang kemiripannya diatas 60% sebanyak 534 gambar dalam waktu 43.23 detik waktu eksekusi. Dalam gambar yang terlampir juga, dapat dilihat delapan gambar dengan kemiripan tertinggi.

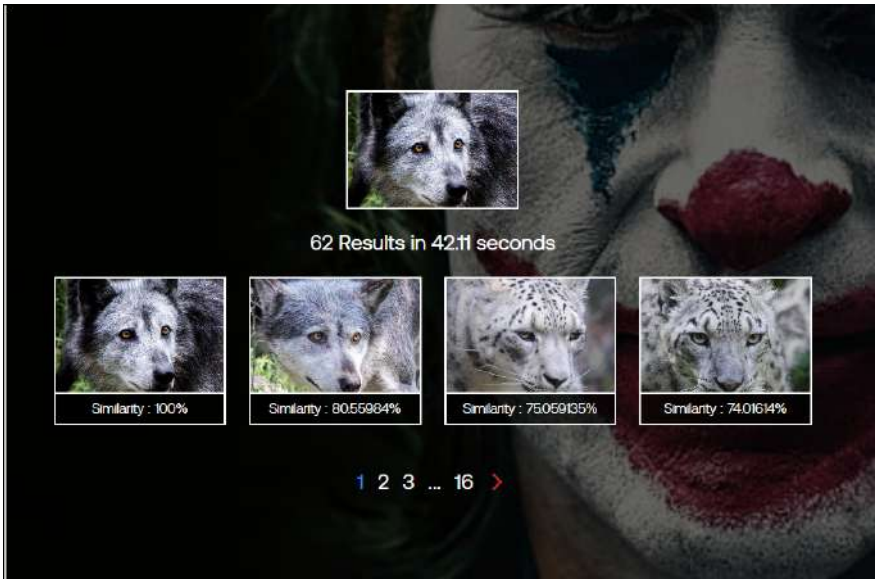
**b. Hasil Pengujian II**

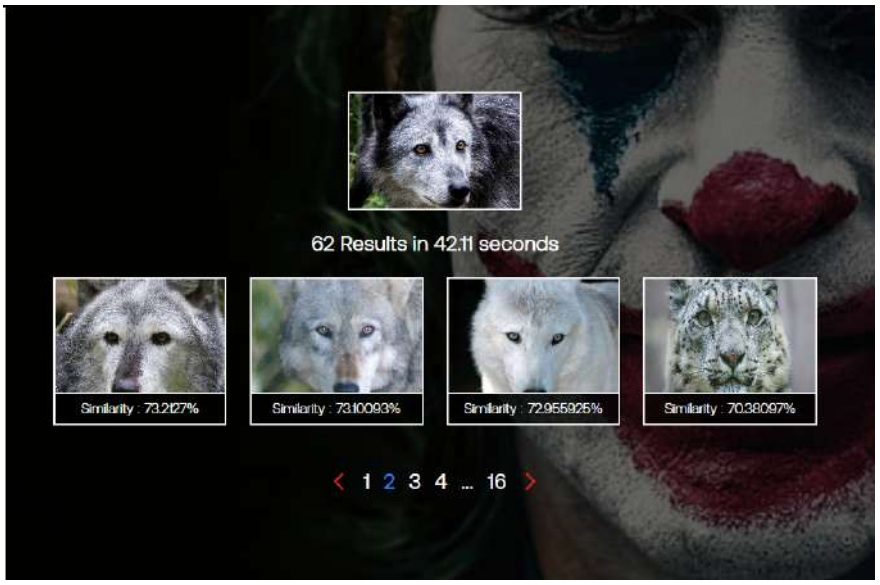



Dengan menggunakan metode *color*, serigala dengan lingkungan berwarna coklat yang dijadikan sampel ketika dibandingkan dataset yang telah diinput pada gambar yang terlampir sebelumnya mengeluarkan hasil gambar yang kemiripannya diatas 60% sebanyak 1667 gambar dalam waktu 42.29 detik waktu eksekusi. Dalam gambar yang terlampir juga, dapat dilihat delapan gambar dengan kemiripan tertinggi.

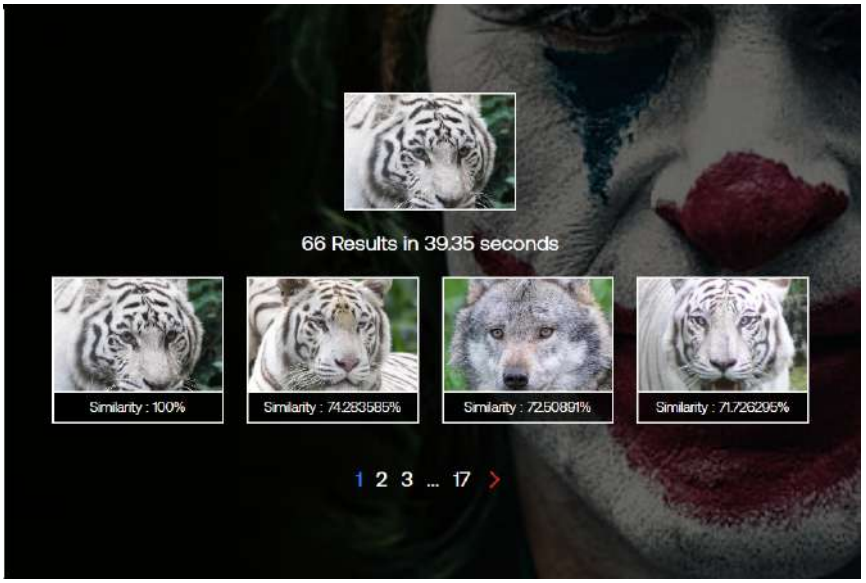
**c. Hasil Pengujian III**

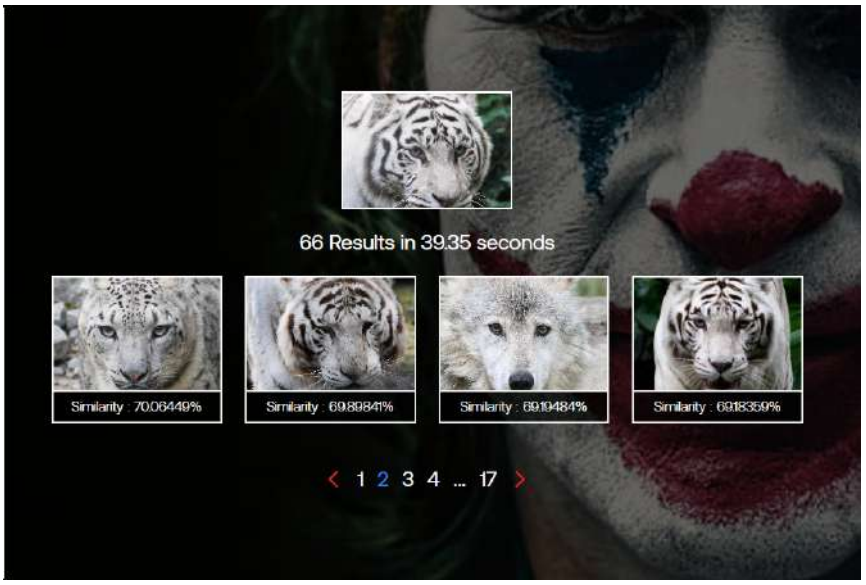



Dengan menggunakan metode *color*, harimau yang dijadikan sampel ketika dibandingkan dataset yang telah diinput pada gambar yang terlampir sebelumnya mengeluarkan hasil gambar yang kemiripannya diatas 60% sebanyak 62 gambar dalam waktu 42.11 detik waktu eksekusi. Dalam gambar yang terlampir juga, dapat dilihat delapan gambar dengan kemiripan tertinggi.

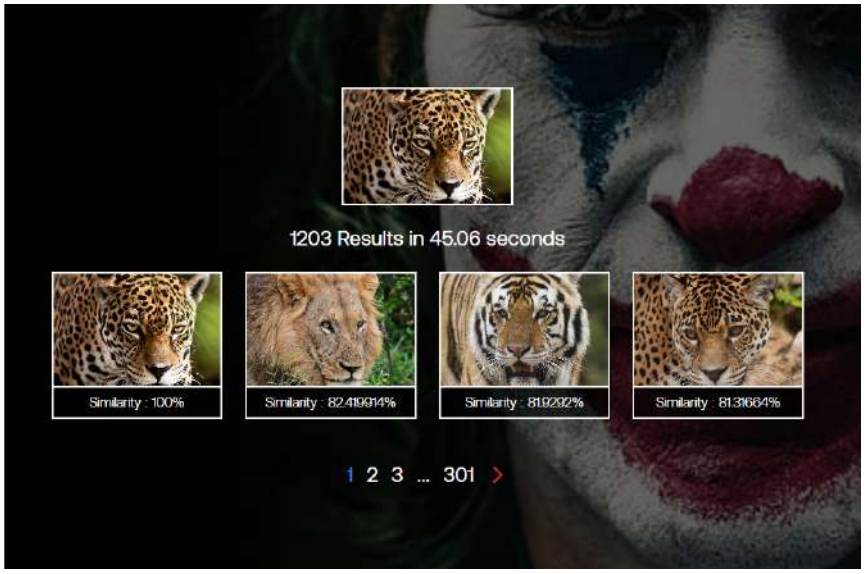
**d. Hasil Pengujian IV**

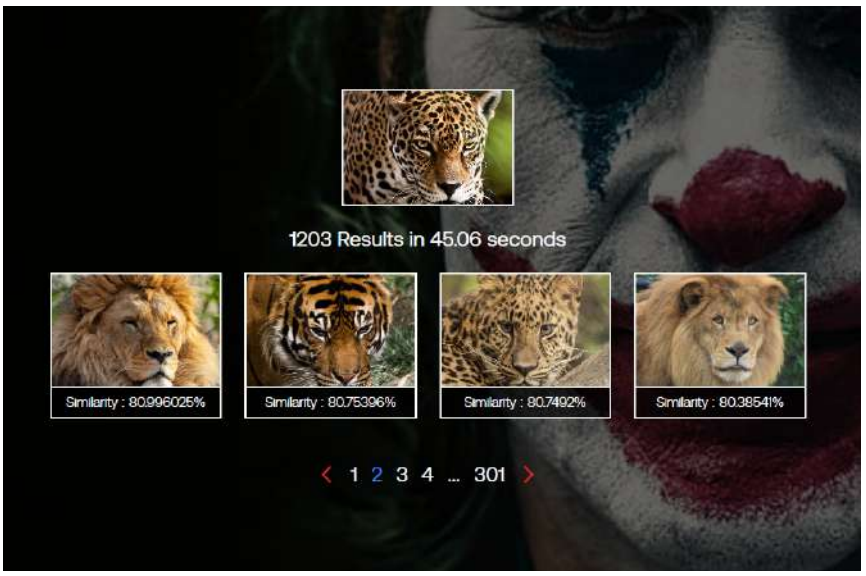



Dengan menggunakan metode *color*, harimau yang dijadikan sampel ketika dibandingkan dataset yang telah diinput pada gambar yang terlampir sebelumnya mengeluarkan hasil gambar yang kemiripannya diatas 60% sebanyak 66 gambar dalam waktu 39.35 detik waktu eksekusi. Dalam gambar yang terlampir juga, dapat dilihat delapan gambar dengan kemiripan tertinggi.

**e. Hasil Pengujian V**

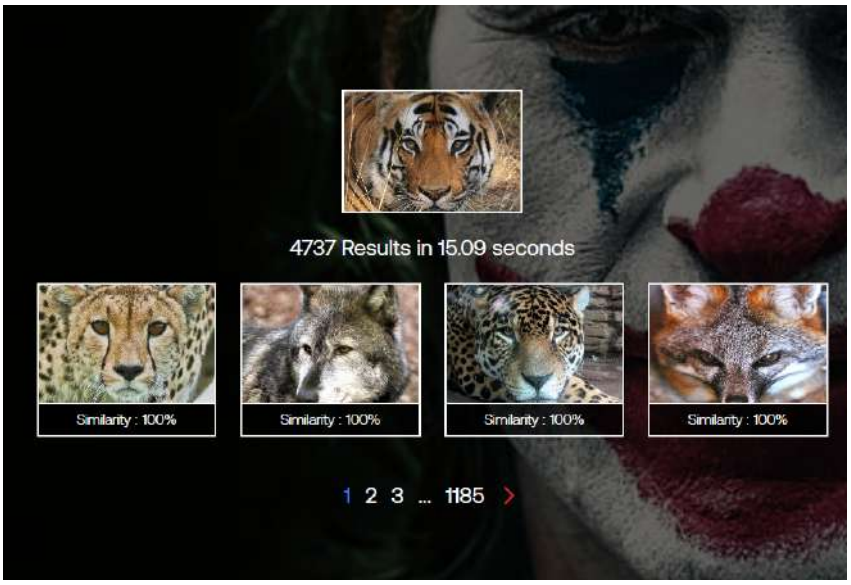


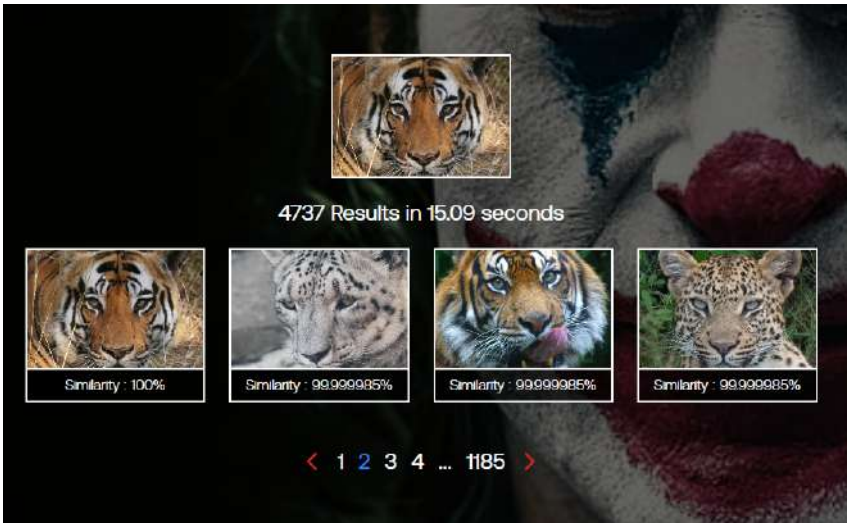




Dengan menggunakan metode *color*, harimau yang dijadikan sampel ketika dibandingkan dataset yang telah diinput pada gambar yang terlampir sebelumnya mengeluarkan hasil gambar yang kemiripannya diatas 60% sebanyak 1203 gambar dalam waktu 45.06 detik waktu eksekusi. Dalam gambar yang terlampir juga, dapat dilihat delapan gambar dengan kemiripan tertinggi.

### 4.4.3. Pengujian dengan Metode Texture

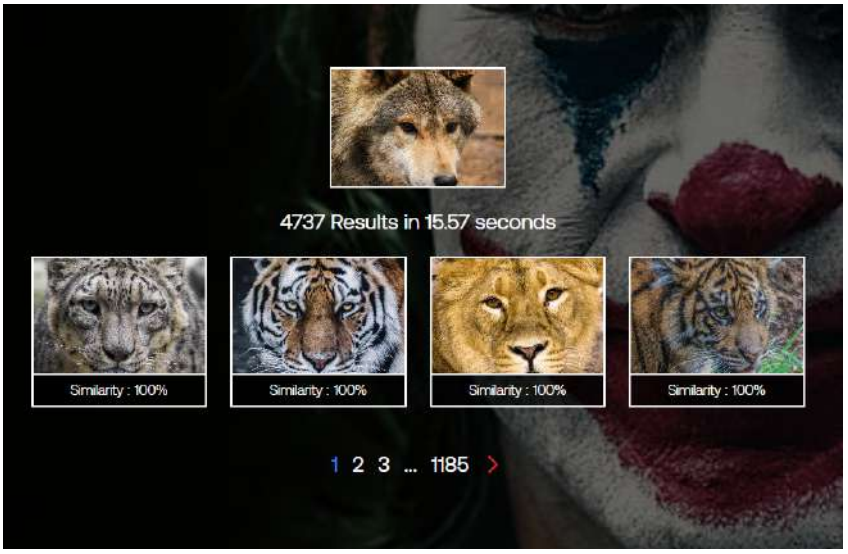
#### a. Hasil Pengujian I

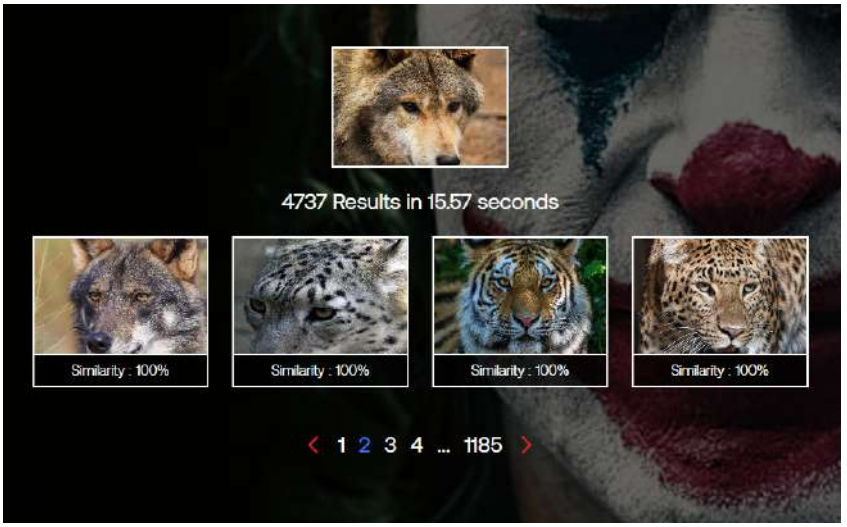



Dengan menggunakan metode *texture*, dengan menggunakan gambar **harimau** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 4737 gambar dengan waktu pemrosesan data selama 15.09 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

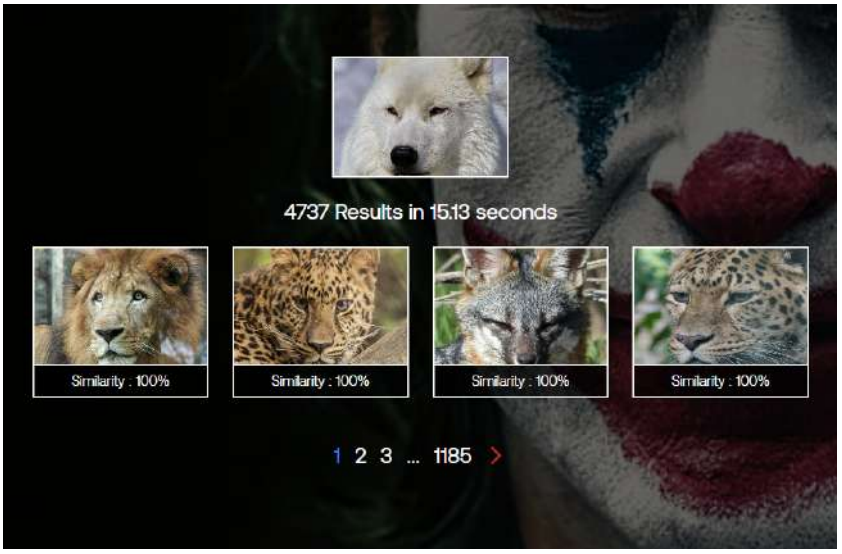
**b. Hasil Pengujian II**

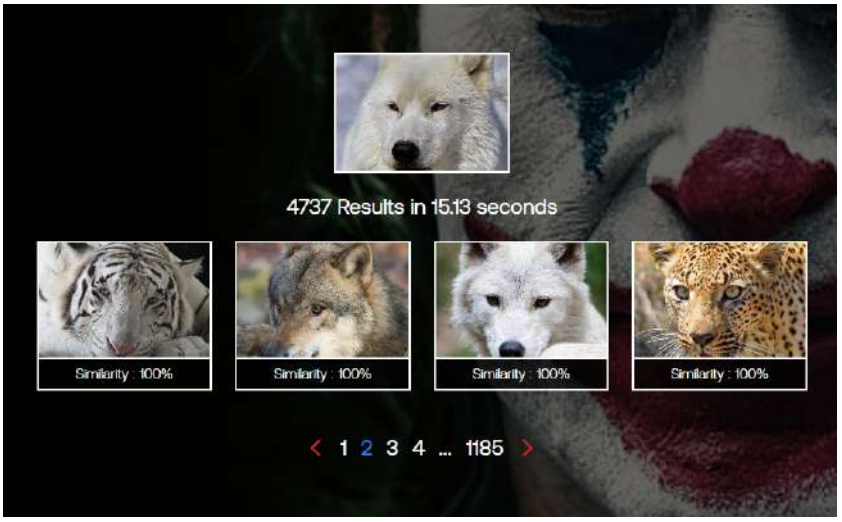



Dengan menggunakan metode *texture*, dengan menggunakan gambar **serigala hitam abu-abu** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 4737 gambar dengan waktu pemrosesan data selama 15.57 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

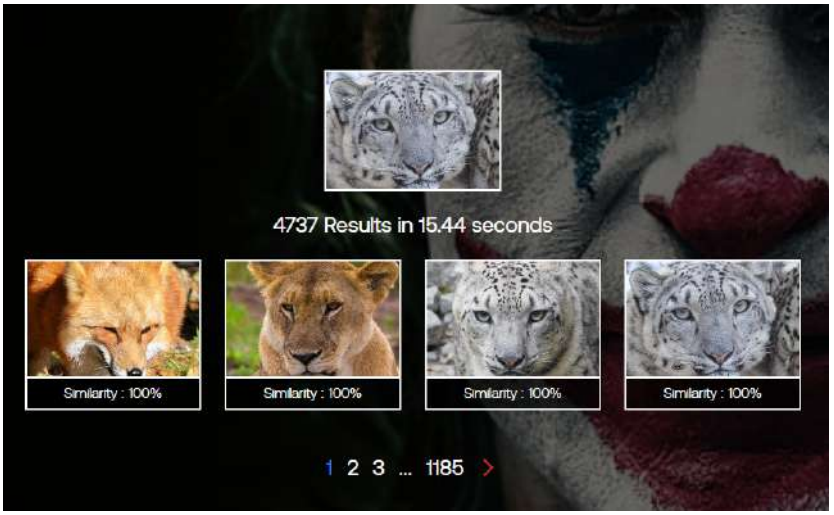
**c. Hasil Pengujian III**

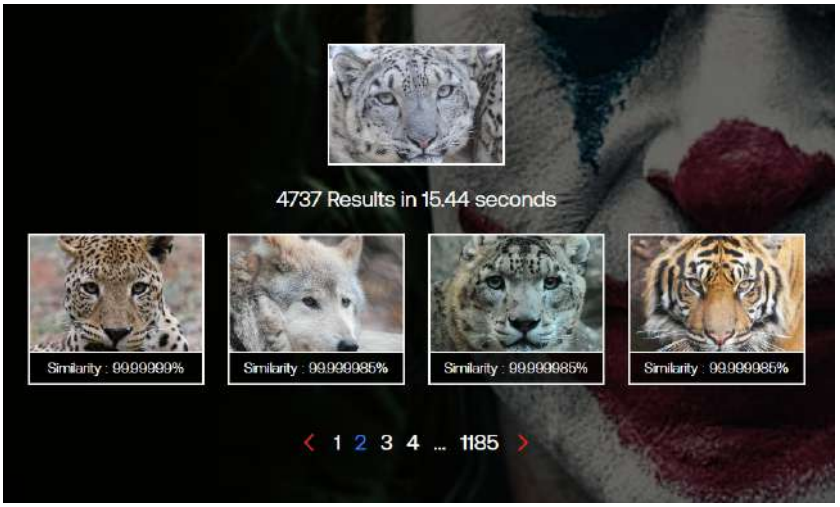



Dengan menggunakan metode *texture*, dengan menggunakan gambar **serigala putih** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 4737 gambar dengan waktu pemrosesan data selama 15.09 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

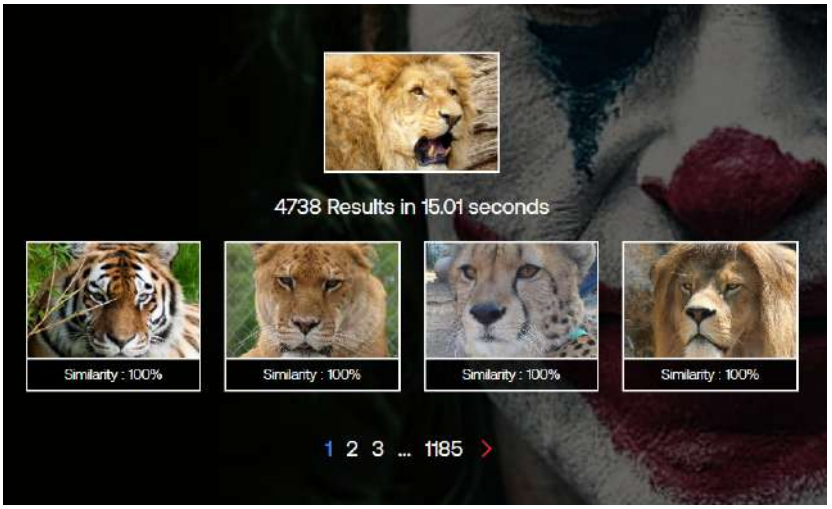
**d. Hasil Pengujian IV**

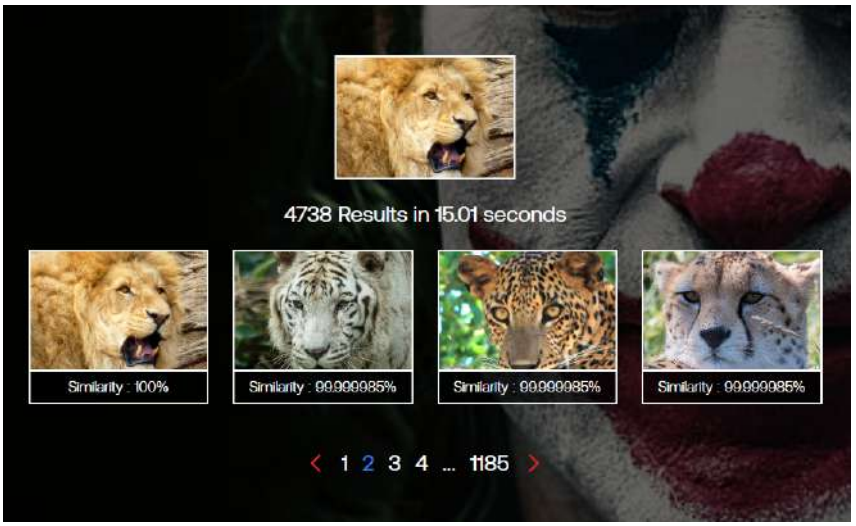



Dengan menggunakan metode *texture*, dengan menggunakan gambar **harimau kutub / putih** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 4737 gambar dengan waktu pemrosesan data selama 15.44 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

**e. Hasil Pengujian V**

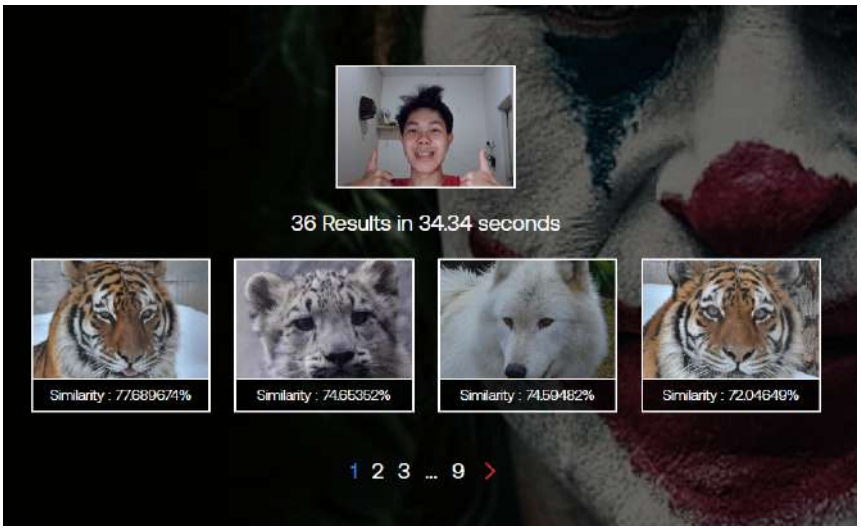


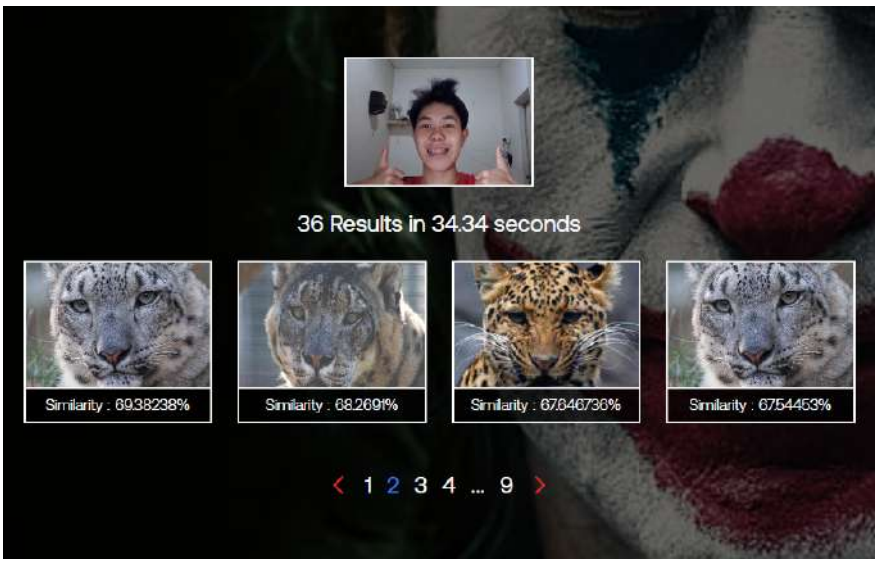




Dengan menggunakan metode *texture*, dengan menggunakan gambar **singa jantan** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 4738 gambar dengan waktu pemrosesan data selama 15.01 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

### 4.4.4. Pengujian dengan Webcam

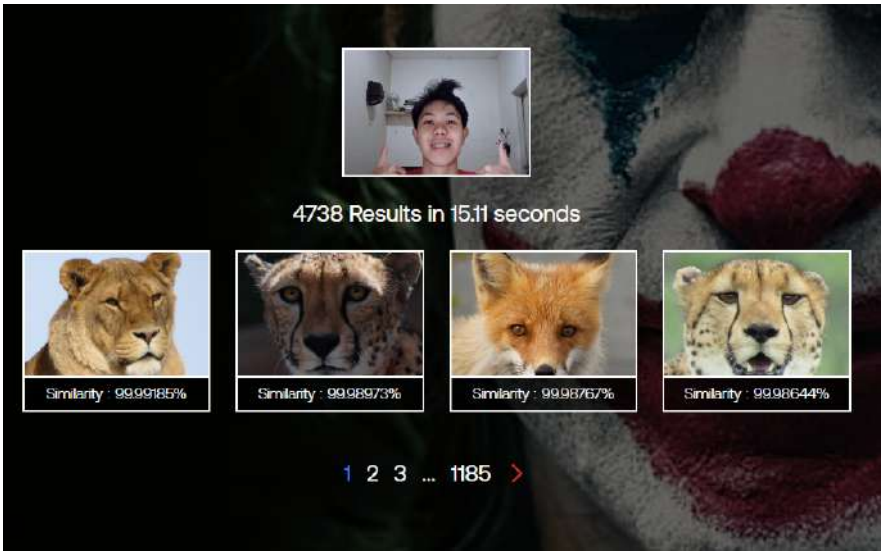
#### a. Hasil Pengujian dengan Color

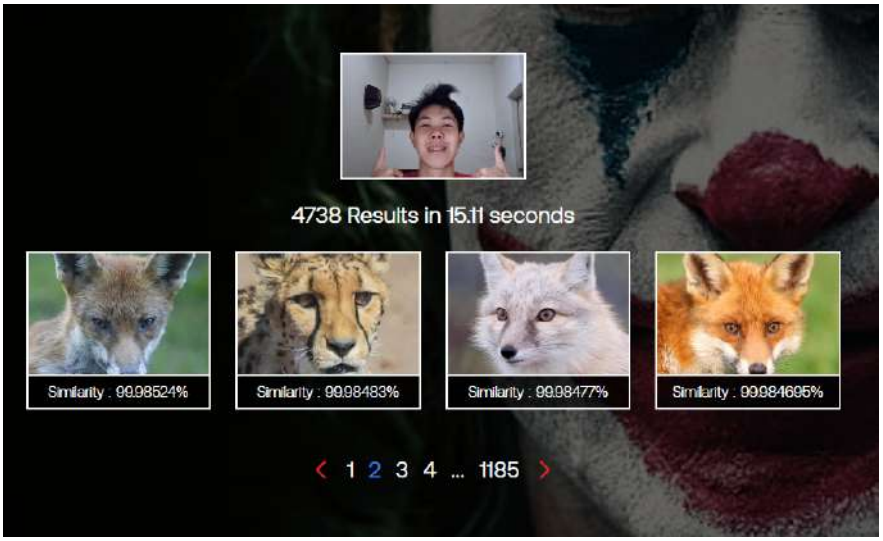



Dengan menggunakan metode *texture*, dengan menggunakan gambar **Farel Winalda** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 36 gambar dengan waktu pemrosesan data selama 34.34 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

**b. Hasil Pengujian dengan Texture**

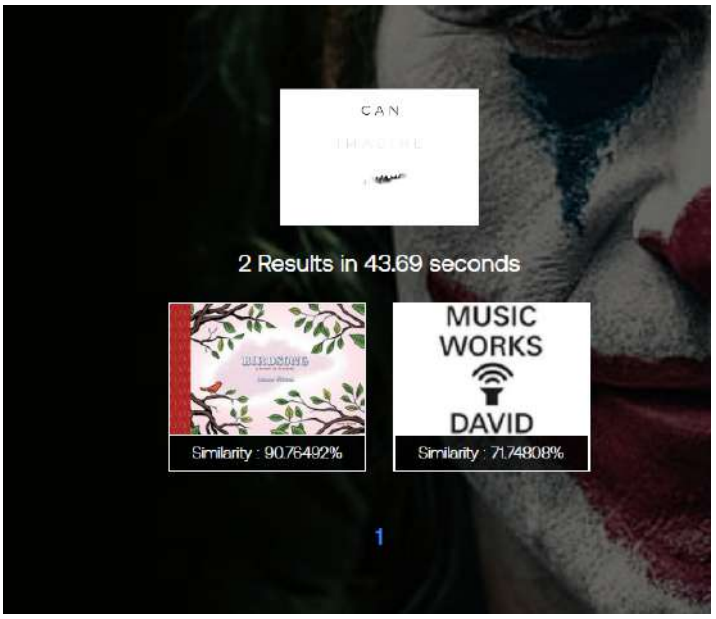



Dengan menggunakan metode *texture*, dengan menggunakan gambar **Farel Winalda** sebagai gambar yang dibandingkan dengan dataset didapatkan hasil gambar yang memiliki kemiripan di atas 60% sebanyak 4738 gambar dengan waktu pemrosesan data selama 15.11 detik. Dilampirkan data delapan gambar yang memiliki tingkat kemiripan tertinggi.

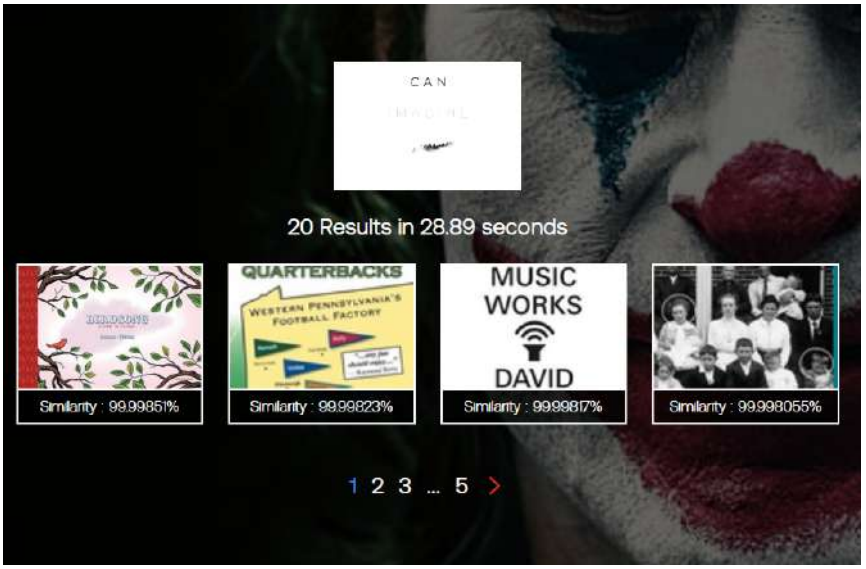
### 4.4.5. Pengujian Web Scraping

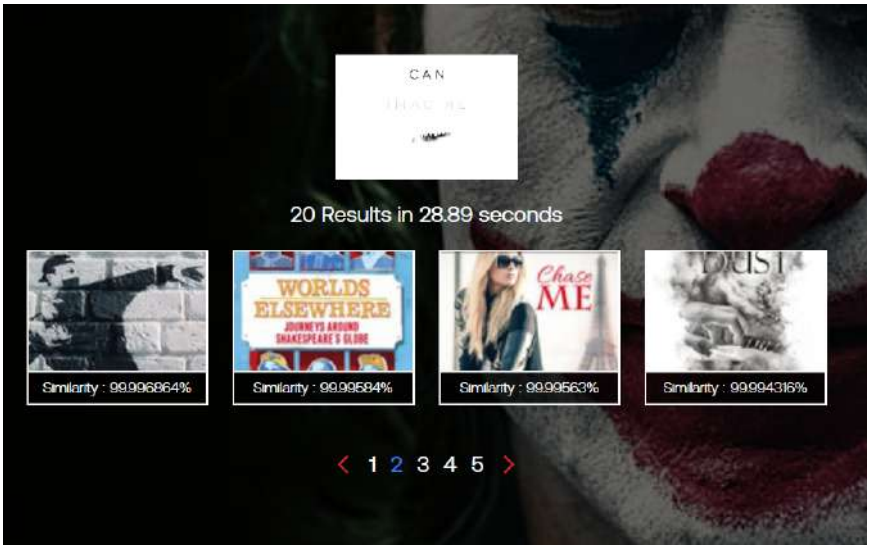
#### a. Hasil Pengujian dengan Color




Dengan menggunakan metode *color*, untuk mencari judul buku *Better Than You Can Imagine* karya Robert Ellis sebagai gambar yang dibandingkan dan dengan scraping gambar melalui website https://books.toscrape.com/catalogue/page-2.html didapatkan hasil gambar dengan kemiripan di atas 60% dengan waktu pemrosesan 43.69 detik. Dilampirkan 2 gambar dengan tingkat kemiripan tertinggi

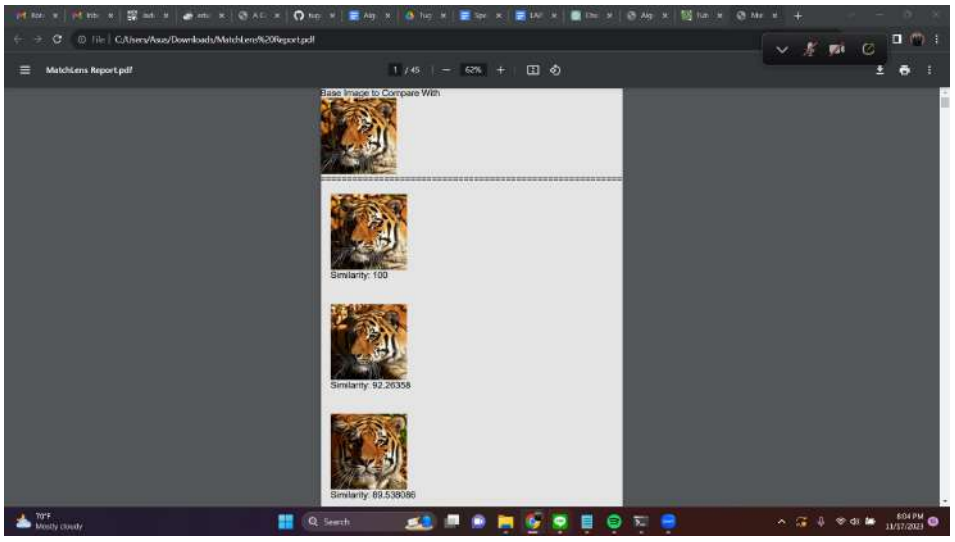
#### b. Hasil Pengujian dengan Texture






Dengan menggunakan metode *texture*, untuk mencari judul buku *Better Than You Can Imagine* karya Robert Ellis sebagai gambar yang dibandingkan dan dengan scraping gambar melalui website https://books.toscrape.com/catalogue/page-2.html didapatkan hasil gambar dengan kemiripan di atas 60% yang serupa dengan waktu pemrosesan 28.89 detik. Dilampirkan 8 gambar dengan tingkat kemiripan tertinggi

### 4.4.6. Pengujian Export PDF




Pada fitur ***Export PDF*** , fitur akan melakukan download ke local directory dari pengguna dalam format PDF yang berisi laporan terkait hasil perilaku *search* dengan menampilkan gambar yang sampel yang dibandingkan serta gambar-gambar lain yang memiliki similaritas di atas 60% beserta angka similaritasnya.

## 4.5. Analisis

Program CBIR (*Content-Based Image Retrieval*) berbasis fitur warna pada awalnya menggunakan konsep global dengan membandingkan seluruh pixel gambar yang satu dengan gambar yang lain. Namun, menurut hasil pengujian, terbukti bahwa dengan metode global akurasi CBIR menjadi rendah karena keacakan warna tidak dipertimbangkan karena yang dibandingkan hanyalah warna keseluruhan gambar. Maka dari itu, dilakukan metode CBIR dengan melakukan pembagian 3x3 blok agar akurasi meningkat dengan membandingkan spesifik blok dengan blok lain yang sama sehingga keacakan warna pada gambar lebih dapat dikurangi dibandingkan secara global. Dengan itu, metode color blok 3x3 akan menghasilkan similaritas yang lebih akurasi namun dengan waktu eksekusi yang lebih lambat karena algoritma yang harus membagi gambar menjadi sembilan blok untuk dibandingkan.

Analisis dari segi efisiensi algoritma menunjukkan bahwa waktu eksekusi untuk program CBIR berbasis fitur tekstur lebih efisien dibandingkan dengan yang berbasis fitur warna. Efisiensi ini berasal dari iterasi yang lebih sedikit dalam kode program CBIR yang menggunakan fitur tekstur, sehingga menghasilkan kompleksitas waktu yang lebih rendah. Sementara itu, algoritma fitur warna, yang membagi blok menjadi 3x3, membutuhkan iterasi yang lebih banyak. Dengan demikian, dari segi waktu eksekusi atau efisiensi algoritma, program CBIR berbasis fitur tekstur lebih unggul dibandingkan dengan yang berbasis fitur warna. Sebagai informasi tambahan, kedua algoritma ini ditulis dalam bahasa pemrograman Go, yang mempercepat waktu kompilasi dibandingkan dengan bahasa pemrograman lain.

Dalam analisis akurasi untuk pembangunan CBIR (*Content-Based Image Retrieval*), terungkap bahwa CBIR berbasis fitur warna menampilkan gambar yang lebih sesuai dengan sampel yang diberikan. Ini karena fokus utamanya pada karakteristik warna, dimana dataset dengan warna serupa seringkali merepresentasikan objek yang sama. Namun, perlu dicatat bahwa beberapa hasil dengan similaritas tinggi dalam CBIR berbasis warna bukanlah objek yang sama dengan sampel, disebabkan oleh kesamaan warna lingkungan dengan sampel yang menghasilkan similaritas tinggi. Dibandingkan dengan fitur tekstur, fitur warna menunjukkan akurasi yang lebih baik.

Sementara itu, analisis terhadap akurasi algoritma CBIR dengan fitur tekstur menunjukkan bahwa sebagian besar hasil dari dataset memiliki similaritas di atas 60%. Hal ini mengindikasikan bahwa algoritma tersebut kurang efektif dalam menentukan similaritas antar gambar. Oleh karena itu, dapat disimpulkan bahwa algoritma CBIR berbasis fitur warna lebih akurat dibandingkan dengan fitur tekstur.

Berdasarkan hasil analisis ini, pengembangan algoritma CBIR untuk mencari similaritas bisa dilakukan dengan mengkombinasikan fitur warna dan tekstur. Pendekatan ini berpotensi meningkatkan sensitivitas dalam mengidentifikasi objek yang sama, sehingga dapat meningkatkan kinerja aplikasi CBIR. Meskipun ini berarti peningkatan dalam akurasi, efisiensi algoritma bisa menjadi lebih kompleks.

# BAB V
# PENUTUP

## 5.1. Kesimpulan

Pada tugas besar ini, kami telah mengimplementasikan berbagai materi tentang vektor yang dipelajari dalam kursus Aljabar Linier dan Geometri IF2123. Implementasi ini kami realisasikan dalam bentuk website bernama MatchLens, yang dirancang untuk mendeteksi similaritas antara dua objek yang berbeda. Meski pembuatan website merupakan tantangan baru bagi kami, karena tidak dipelajari selama menjadi perkuliahan di mata kuliah yang kami ikuti, kami berhasil mengembangkan algoritma untuk mencari similaritas antara dua objek gambar yang cukup efisien dan cepat dengan menggunakan bahasa pemrograman *Go*. Sedangkan, untuk pengembangan website kami menggunakan *React JS* untuk membantu dalam segi *frontend*, *Fiber* untuk membantu dalam segi *backend*, dan *Tailwind CSS* untuk membantu melakukan realisasi desain website yang telah dibuat.

Pada aplikasi ini, terdapat dua fitur utama dalam menerapkan CBIR (*Content-Based Image Retrieval*) yaitu fitur color dan fitur texture yang dapat dimanfaatkan dalam website ini. Dengan website ini, pengguna akan merasa terbantu jika ingin mencari kemiripan antara satu objek dengan sebuah dataset yang memiliki jumlah yang masif dalam waktu yang relatif singkat. Dalam website ini, juga terdapat fitur-fitur tambahan lain yang dapat meningkatkan aksesibilitas dari pengguna seperti fitur *image scraping* yang dapat membantu pengguna untuk menambah dataset yang ia punya dengan mengambil gambar dari website-website tertentu, fitur *webcam* yang membuat penggunanya dapat memasukkan gambar sampel yang hendak dicari similaritasnya dengan tangkapan dari kamera *webcam* yang terdapat pada devicenya, dan fitur *export to PDF* yang memberi kesempatan bagi pengguna untuk mengunduh laporan hasil pencarian similaritas yang telah dilakukan sehingga pengguna dapat melakukan analisis lebih lanjut dengan mudah.

Melalui proyek ini, kami tidak hanya menerapkan pengetahuan teoritis, tetapi juga belajar tentang konsep algoritma untuk menghitung similaritas menggunakan vektor dan cosine similarity. Pengalaman ini memperluas pemahaman kami tentang

aplikasi praktis materi akademis dalam pengembangan solusi teknologi yang nyata. Kami berharap MatchLens dapat menjadi alat yang bermanfaat dalam berbagai aplikasi praktis, membuka jalan bagi inovasi dan penelitian lebih lanjut di bidang ini.

## 5.2. Saran

- Pembuatan algoritma seharusnya melibatkan semua anggota kelompok sehingga ketika semua anggota kelompok menjadi paham akan penggunaan algoritma di balik website yang telah dibuat.
- Pengerjaan bonus tugas seharusnya lebih awal agar tidak kewalahan jika menemukan *bug* pada fitur-fitur bonus yang telah dibuat.
- Struktur folder seharusnya telah dibuat dengan jelas sesaat dibuatnya github dan sebelum dilakukannya masa *development* website karena akan menimbulkan kebingungan ketika mengubah struktur folder yang telah mengandung jumlah file yang masif.
- Pembuatan laporan seharusnya dicicil sejak jauh hari karena formatnya yang cukup kompleks dan mendetail sehingga harus dikerjakan dengan lebih teliti dan lebih ekstra.
- Seharusnya, dilakukan pertemuan yang lebih sering agar *progress* tetap berjalan dan mengetahui jika ada anggota kelompok yang mengalami kewalahan.

## 5.3. Komentar

Elbert Chailes : Gokil bang ! Aku sudah kebal dengan tugas besar.
Farel Winalda : Ez bangetz
Ivan Hendrawan Tan : Kata farel EZ

## 5.4. Refleksi

Melalui tugas besar ini, kami mendapatkan ilmu-ilmu yang baru dan beragam, dimulai dengan penerapan aplikasi vektor dan perhitungan-perhitungannya, eksplorasi *package-package* untuk melakukan realisasi beberapa fitur bonus seperti *export to PDF*, *image scraping*, dan *webcam*, dan menambahkan skill kami dalam melakukan pengembangan website yang mungkin akan berguna di kemudian hari. Kami juga

belajar banyak dalam bekerja sama sebagai tim dalam pengembangan website dan fitur, sehingga setidaknya memiliki gambaran terkait pengembangan sebuah fitur dalam website ketika di dunia karir nantinya. *Git* juga menjadi teman kami selama melakukan realisasi tugas besar ini sehingga kami belajar banyak terkait penggunaan *git* yang baik dan melakukan koordinasi yang baik antara *contributors*-nya sehingga konflik yang terjadi menjadi jarang. Dengan tugas besar ini, kami juga menjadi sadar bahwa bahasa pemrograman dan perhitungan yang kompleks bukanlah sebuah tantangan yang besar jika dibandingkan dengan pengembangan program utama yang kita kerjakan, yaitu website dan algoritma aplikasinya sendiri.

## 5.5. Ruang perbaikan

Jika akan dilakukan pengembangan yang lebih detail terhadap aplikasi yang telah dibuat ini, maka dapat dilakukan perbaikan terhadap algoritma tekstur yang cenderung memiliki akurasi yang rendah. Kemudian, juga bisa dilakukan pengembangan dalam waktu eksekusi algoritma pencarian yang telah dibuat agar memiliki waktu eksekusi yang lebih singkat. Menambah fitur *caching* karena hal tersebut akan sangat membantu dan menghemat waktu pengguna karena tidak perlu melakukan muat dataset secara berulang.

# DAFTAR REFERENSI

Khrisne, Care dan M. David Yusanto. 2015. Content-Based Image Retrieval Menggunakan Metode Block Truncation Algorithm dan Grid Partitioning. Denpasar : Universitas Udayana.

https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/algeo23-24.htm

https://www.anakteknik.co.id/krysnayudhamaulana/articles/bagaimana-siklus-pengembangan-sebuah-web

https://books.toscrape.com/

https://www.sciencedirect.com/science/article/pii/S0895717710005352#s000010

https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1

https://react-pdf.org/

https://tailwindcss.com/docs/installation

https://legacy.reactjs.org/docs/create-a-new-react-app.html

https://create-react-app.dev/

https://www.zenrows.com/blog/web-scraping-golang

https://www.w3schools.com/go/

https://github.com/gocolly/colly

https://www.researchgate.net/figure/Calculation-of-a-joint-histogram-The-pixel-value-in-the-first-image-and-the_fig4_221051037

# LAMPIRAN

- Link Repository

  Link berikut merupakan link untuk menuju ke github repository dimana aplikasi ini dibuat dan dikembangkan.

  https://github.com/FarelW/Algeo02-22045

- Link Video

  Link berikut merupakan link untuk menuju video promosi produk MatchLens. yang diperani oleh pengembangnya sendiri yang di-*upload* pada media YouTube.

  https://youtu.be/jRxH3FCKLk4