

SKRIPSI

PENERAPAN DATA MINING PADA MASALAH PENGENALAN POINT OF INTEREST



Gian Martin Dwibudi

NPM: 6181801015

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2022

UNDERGRADUATE THESIS

**APPLICATION OF DATA MINING ON THE PROBLEM OF
RECOGNIZING POINT OF INTEREST**



Gian Martin Dwibudi

NPM: 6181801015

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2022**

ABSTRAK

Point of Interest (POI) merupakan sebuah lokasi geografis yang memiliki fungsi tertentu dan dikenali orang. Beberapa POI memiliki bagian penting yang unik terhadap POI tersebut. Bagian penting tersebut biasanya berupa logo atau bagian lainnya yang mudah terlihat dan berbeda dari bagian-bagian dari POI lain. Bagian penting dari POI tersebut dapat digunakan untuk mengenali sebuah POI dengan mudah, seseorang dapat mengenali sebuah POI hanya dengan melihat bagian tersebut saja.

Komputer dapat mengenali POI dari sebuah gambar dengan menggunakan fitur lokal dalam gambar. Salah satu teknik yang dapat digunakan untuk mengenali gambar adalah OIR. Teknik OIR mengenali gambar dengan memasangkan fitur lokal dari gambar masukkan dan gambar-gambar di *dataset*. Sebuah gambar dapat mengandung sangat banyak fitur lokal, sehingga proses pemasangan dapat memakan waktu yang sangat lama. Proses dapat dipercepat dengan menggunakan hanya fitur lokal dari bagian POI yang penting saja.

Penelitian ini bertujuan untuk menyaring fitur lokal dan mengambil yang hanya berasal dari bagian penting POI saja. Penyaringan fitur lokal berdasarkan pada anggapan bahwa bagian penting dari POI adalah bagian yang sifatnya konsisten dan unik terhadap POI tersebut. Bagian penting tersebut bersifat konsisten dan unik maka akan menghasilkan fitur lokal yang sifatnya konsisten dan unik juga. Fitur lokal yang konsisten dan unik dapat ditentukan dengan menggunakan metode *clustering*.

Metode *clustering* digunakan untuk membagi fitur-fitur lokal menjadi kelompok-kelompok. Kelompok-kelompok tersebut akan digunakan untuk menentukan tingkat kekonsistennan dan keunikan fitur-fitur lokal anggotanya.

Kata-kata kunci: *Point of interest*, fitur lokal, OIR, *clustering*, konsistensi, keunikan

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	xiii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	2
1.3 Tujuan	3
1.4 Batasan Masalah	3
1.5 Metodologi	3
1.6 Sistematika Pembahasan	3
2 LANDASAN TEORI	5
2.1 Point of Interest	5
2.2 Object Instance Recognition	6
2.3 SIFT (Scale Invariant Feature Transform)	10
2.3.1 Pencarian Extrema	10
2.3.2 Penentuan Skala	13
2.3.3 Penentuan Orientasi	13
2.3.4 Pembuatan Deskriptor	15
2.3.5 SIFT di OpenCV	15
2.4 ORB (Oriented FAST and Rotated BRIEF)	16
2.4.1 Pencarian Keypoint	16
2.4.2 Penentuan Skala	17
2.4.3 Penentuan Orientasi	18
2.4.4 Pembuatan Deskriptor	19
2.4.5 ORB di OpenCV	21
2.5 BSIS (Best Score Increasing Subsequence)	22
2.5.1 Pairing	22
2.5.2 Verification	23
2.5.3 Scoring	25
2.6 KD-Tree	25
2.7 Locality-Sensitive Hashing	26
2.8 Agglomerative Clustering	27
2.9 Dataset Stanford Mobile Visual Search	28
3 ANALISIS & EKSPERIMEN	33
3.1 Analisis Pengenalan POI	33
3.1.1 Ide Dasar Analisis	33
3.1.2 Tahapan Analisis	33
3.1.3 Implementasi	35
3.1.4 Hasil Analisis	36
3.2 Analisis Fitur Lokal dari Logo dan Bagian yang Konsisten	36

3.3	Dataset yang Digunakan	38
3.3.1	Dataset SMVS	38
3.3.2	Dataset GSV	39
3.4	Analisis Sifat Konsisten pada Fitur Lokal dari Gambar POI	43
3.4.1	Ide Dasar dan Tahapan Analisis	43
3.4.2	Implementasi	45
3.4.3	Hasil	45
3.5	Analisis Sifat Unik pada Fitur Lokal dari Gambar POI	47
3.5.1	Ide Dasar dan Tahapan Analisis	47
3.5.2	Implementasi	49
3.5.3	Hasil	49
3.6	Analisis Tingkat Keunikan dan Kekonsistennan Fitur Lokal pada Gambar POI	51
3.6.1	Ide Dasar Analisis	51
3.6.2	Tahapan Analisis	52
3.6.3	Implementasi	54
3.6.4	Hasil Analisis	55
3.6.5	Visualisasi Nilai Konsistensi dan Keunikan Fitur Lokal	57
3.7	Analisis Penentuan Nilai Threshold untuk Konsistensi dan Keunikan	61
3.7.1	Ide Dasar dan Tahapan Analisis	63
3.7.2	Metode Scoring	63
3.7.3	Hasil Analisis	65
3.8	Analisis Penggunaan Nilai Konsistensi dan Nilai Keunikan untuk OIR dengan BSIS	68
3.8.1	Data Train dan Test	68
3.8.2	Metode BSIS	70
3.8.3	Tahapan Analisis	71
3.8.4	Tahapan Implementasi	71
3.8.5	Hasil Analisis	72
3.9	Analisis Metode Ekstraksi Fitur Lokal ORB	78
3.9.1	Hasil Pengujian	79
4	PERANCANGAN	81
4.1	Rancangan Alur Program	81
4.2	Rancangan Implementasi Metode Clustering Pembuatan Model	82
4.2.1	Rancangan Struktur Folder	82
4.2.2	Rancangan Proses Clustering	83
4.3	Rancangan Kelas ClusterModel	86
4.4	Rancangan Kelas Util	87
4.5	Rancangan Kelas-kelas Implementasi BSIS	88
4.6	Rancangan Perangkat Lunak BSIS	90
5	PENGUJIAN	93
5.1	Pengujian Perangkat Lunak BSIS	93
5.2	Pengujian Metode Clustering untuk Identifikasi dengan BSIS	95
5.2.1	Ide Analisis	95
5.2.2	Penentuan Threshold dan Hasil Analisis Metode SIFT	96
5.2.3	Penentuan Threshold dan Hasil Analisis Metode ORB	101
5.2.4	Analisis Hasil Pengujian	106
6	KESIMPULAN & SARAN	109
6.1	Kesimpulan	109
6.2	Saran	110

DAFTAR REFERENSI	111
A KODE PROGRAM	113
B HASIL EKSPERIMEN	139

DAFTAR GAMBAR

1.1	Contoh POI yang memiliki logo unik. POI seperti ini dapat dikenali dengan melakukkan identifikasi pada logo tersebut.	1
1.2	Empat gambar di atas menunjukkan permasalahan yang dihadapi pada penelitian ini. Gambar (a) dan (b) merupakan gambar dari dua POI yang berbeda tetapi memiliki logo dengan bagian sudut yang mirip, sudut yang mirip tersebut kemungkinan akan menghasilkan fitur lokal yang mirip juga. Sedangkan pada gambar (c) dan (d) merupakan dua gambar dari POI yang sama tetapi banyak objek latar yang berbeda sehingga akan memunculkan fitur lokal yang tidak konsisten.	2
2.1	Salah satu contoh POI.	5
2.2	Contoh POI dengan logo unik.	6
2.3	Contoh variasi pada gambar <i>cover</i> buku yang dapat menyebabkan masalah pada OIR	7
2.4	Ilustrasi pasangan fitur lokal yang konsisten dan tidak konsisten.	9
2.5	Kurva Gaussian dan bentuk representasi <i>matrix</i> -nya.	10
2.6	Kurva dan <i>matrix</i> Gaussian pada nilai σ yang berbeda.	11
2.7	Efek nilai σ pada hasil gambar konvolusi.	11
2.8	Operasi DoG pada gambar	12
2.9	Penggunaan DoG pada SIFT	12
2.10	Oktaf pada proses konvolusi SIFT	13
2.11	Ilustrasi pembobotan pada <i>Gaussian Weighting</i> . Titik tengah merupakan <i>keypoint</i> yang diperiksa sedangkan setiap kotak merupakan <i>pixel-pixel</i> di sekitar <i>keypoint</i> . Tanda panah pada tiap kotak menunjukkan <i>magnitude</i> dan orientasi <i>pixel</i> tersebut, panjang panah merupakan nilai <i>magnitude</i> dan arahnya merupakan orientasi	14
2.12	Histogram untuk menentukan orientasi dari <i>keypoint</i> . Bin dengan nilai tertinggi (tanda panah biru) akan digunakan sebagai orientasi dari <i>keypoint</i> . Untuk bin lain yang jumlahnya berada dalam rentang 80% dari bin tertinggi (tanda panah hijau) digunakan untuk membuat <i>keypoint baru</i>	15
2.13	Ilustrasi penentuan <i>keypoint</i> pada ORB. Setiap kotak menunjukkan sebuah <i>pixel</i> pada gambar dan angka di dalamnya merupakan nilai intensitasnya. <i>Pixel</i> di tengah gambar (<i>pixel</i> bernilai 4) merupakan <i>pixel</i> yang akan diperiksa apakah merupakan <i>keypoint</i> . <i>Pixel-pixel</i> yang ditandai dengan kotak putih merupakan 16 <i>pixel</i> yang digunakan untuk memeriksa apakah <i>pixel</i> di tengah merupakan <i>keypoint</i>	17
2.14	<i>Image Pyramid</i> pada ORB	18
2.15	Ilustrasi penentuan orientasi pada ORB.	19
2.16	Ilustrasi penghitungan <i>Integral Image</i> . <i>Matrix I</i> merupakan <i>matrix</i> awal dan <i>Matrix I_{Sigma}</i> merupakan <i>Integral Image</i> dari <i>I</i>	20
2.17	Penghitungan nilai total sebuah daerah dengan menggunakan <i>Integral Image</i>	21
2.18	Ilustrasi penghitungan jarak Hamming antara dua vektor.	21
2.19	Tahapan verifikasi pada BSIS.	24
2.20	Contoh pohon struktur KD-Tree. Angka di sebelah kanan menunjukkan elemen vektor yang diperiksa pada level tersebut.	25
2.21	Ilustrasi proses <i>banding</i> dan penggunaan <i>bucket</i> sebagai indeks dalam LSH.	27

3.1	<i>Flowchart</i> tahapan analisis pengenalan POI	34
3.2	Dua gambar yang digunakan untuk melakukan analisis pengenalan POI dengan logo. Gambar yang di sebelah kiri adalah Gambar <i>Q</i> dan yang di sebelah kanan adalah Gambar <i>T</i>	34
3.3	Pasangan <i>keypoint</i> dari Gambar <i>Q</i> dan Gambar <i>T</i> yang kuat.	36
3.4	Beberapa sudut pengambilan dan waktu pengambilan berbeda dari suatu POI yang sama.	37
3.5	POI yang memiliki logo dengan sudut yang mirip.	38
3.6	Beberapa contoh gambar yang digunakan pada analisis untuk melihat sifat konsisten pada fitur lokal.	44
3.7	Histogram sebaran jumlah gambar unik tiap <i>cluster</i>	46
3.8	Contoh <i>keypoint</i> yang konsisten menurut analisis ini.	47
3.9	Beberapa contoh gambar yang digunakan pada analisis untuk melihat sifat unik pada gambar POI.	48
3.10	Histogram sebaran jumlah kelas gambar yang unik tiap <i>cluster</i>	50
3.11	Contoh <i>keypoint</i> yang unik menurut analisis ini.	51
3.12	<i>Flowchart</i> tahapan analisis <i>clustering</i> untuk mencari fitur lokal yang konsisten dan unik.	52
3.13	Contoh beberapa gambar yang digunakan pada analisis ini.	53
3.14	Histogram sebaran nilai keunikan.	57
3.15	Histogram sebaran nilai keunikan.	57
3.16	Beberapa contoh gambar POI beserta <i>keypoint</i> yang fitur lokalnya sudah diberi warna sesuai dengan nilai konsistensi dan keunikannya.	59
3.17	Contoh objek yang sifatnya konsisten dan unik dalam POI.	61
3.18	Contoh objek yang sifatnya konsisten dan unik dalam POI.	62
3.19	Contoh tampilan aplikasi LabelImg.	64
3.20	Skor ketepatan untuk tiap <i>threshold</i> pada nilai konsistensi.	66
3.21	Skor ketepatan untuk tiap <i>threshold</i> pada nilai keunikan.	67
3.22	Contoh gambar pada <i>dataset book_covers</i> . Keempat gambar tersebut berada dalam satu kelas yang sama.	68
3.23	Contoh gambar referensi dari <i>dataset book_covers</i>	69
3.24	Contoh gambar referensi dari <i>dataset book_covers</i> yang telah ditransformasi.	70
3.25	Modifikasi pada metode BSIS yang dilakukan pada analisis ini.	70
3.26	Tahapan yang dilakukan dalam analisis untuk menguji penggunaan <i>clustering</i> pada BSIS.	71
3.27	Sebaran nilai konsistensi (<i>consistency</i>) untuk Book Covers 400.	73
3.28	Sebaran nilai keunikan (<i>uniqueness</i>) untuk Book Covers 400.	73
3.29	Perbandingan sebaran waktu Book Covers 400.	74
3.30	Sebaran nilai keunikan (<i>uniqueness</i>) untuk Book Covers 600.	75
3.31	Sebaran nilai konsistensi (<i>consistency</i>) untuk Book Covers 600.	76
3.32	Perbandingan sebaran waktu Book Covers 600.	77
4.1	Rancangan alur program pada penelitian ini.	81
4.2	Rancangan struktur <i>folder</i> untuk pembuatan model.	82
4.3	Tahapan proses <i>clustering</i> hingga didapat nilai keunikan dan konsistensi.	83
4.4	Diagram kelas <i>ClusterModel</i>	86
4.5	Diagram kelas <i>Util</i>	87
4.6	Diagram <i>Package</i> <i>bsis</i>	88
4.7	Rancangan tampilan perangkat lunak untuk BSIS.	91
5.1	Tampilan awal aplikasi perangkat lunak BSIS.	93
5.2	Tampilan aplikasi perangkat lunak BSIS setelah memilih gambar <i>input</i>	94

5.3	Tampilan aplikasi perangkat lunak BSIS melakukan identifikasi pada gambar <i>input</i> .	95
5.4	Histogram sebaran nilai keunikan pada <i>dataset GSV 400</i> .	96
5.5	Histogram sebaran nilai keunikan <i>dataset GSV 400</i> .	97
5.6	Sebaran waktu pengujian metode SIFT <i>dataset GSV 400</i> .	98
5.7	Histogram sebaran nilai keunikan pada <i>dataset GSV 600</i> .	99
5.8	Histogram sebaran nilai keunikan <i>dataset GSV 600</i> .	99
5.9	Sebaran waktu pengujian metode SIFT <i>dataset GSV 600</i> .	100
5.10	Histogram sebaran nilai keunikan pada <i>dataset GSV 400</i> .	102
5.11	Histogram sebaran nilai keunikan <i>dataset GSV 400</i> .	102
5.12	Sebaran waktu pengujian metode ORB <i>dataset GSV 400</i> .	103
5.13	Histogram sebaran nilai keunikan pada <i>dataset GSV 600</i> .	104
5.14	Histogram sebaran nilai keunikan <i>dataset GSV 600</i> .	105
5.15	Sebaran waktu pengujian metode ORB <i>dataset GSV 600</i> .	106

1

BAB 1

2

PENDAHULUAN

3 1.1 Latar Belakang

4 Sebuah lokasi atau titik geografis yang memiliki kegunaan tertentu biasa disebut sebagai *Point of*
5 *interest* (POI). POI ini dapat berupa tempat apa saja yang memiliki ciri khas tertentu dan dapat
6 dikenali dari ciri khas tersebut. POI-POI tertentu dapat dikenali dari logo tempat tersebut atau
7 objek-objek lainnya yang terlihat secara langsung. Seperti ditunjukkan pada Gambar 1.1, kedua
8 POI tersebut memiliki logo unik yang terlihat dengan jelas.

9 Pada skripsi ini akan dibuat sebuah sistem untuk mengidentifikasi POI dari masukan yang berisi
10 gambar POI tersebut. Proses identifikasi POI dilakukan dengan mendeteksi logo khusus atau objek
11 unik yang ada pada POI tersebut.



Gambar 1.1: Contoh POI yang memiliki logo unik. POI seperti ini dapat dikenali dengan melakukan identifikasi pada logo tersebut.

12 Proses identifikasi POI akan dilakukan menggunakan teknik *Object Instance Recognition* (OIR).
13 Teknik OIR merupakan teknik pengenalan objek spesifik. Sebuah algoritma OIR harus dapat
14 mengatasi masalah seperti pencahayaan, sudut pengambilan, objek *background*. Objek harus tetap
15 dapat diidentifikasi walaupun gambar memiliki gangguan-gangguan tersebut. OIR dapat dilakukan
16 dengan memanfaatkan fitur lokal.

17 Fitur lokal merupakan fitur yang mendeskripsikan sebuah daerah penting (*keypoint*) pada gambar.
18 Salah satu cara mendapatkan *Keypoint* adalah dengan mencari sudut-sudut atau perpotongan
19 garis (*corner*) yang terdapat pada gambar. *Keypoint-keypoint* yang terdeteksi pada gambar akan
20 memiliki sebuah vektor untuk mendeskripsikan daerah di sekitar *keypoint* tersebut yang disebut
21 sebagai vektor deskriptor. Proses pencarian fitur lokal pada penelitian ini akan dilakukan dengan
22 menggunakan metode *Scale Invariant Feature Transform* (SIFT) dan *Oriented FAST and Rotated*

¹ **BRIEF** (ORB). Metode SIFT dan ORB akan menghasilkan vektor deskriptor untuk tiap fitur lokal
² yang terdeteksi, vektor deskriptor ini dapat digunakan untuk mengidentifikasi fitur lokal.

³ Salah satu masalah yang ada pada pengenalan POI adalah pada sebuah gambar POI tidak
⁴ semua fitur lokal yang dideteksi bersifat unik terhadap POI tersebut. Ada fitur lokal yang juga
⁵ dimiliki oleh POI lain atau fitur lokal yang berasal dari objek di latar belakang yang sifatnya tidak
⁶ konsisten. Masalah ini akan mempersulit pada proses OIR untuk mengidentifikasi POI yang tepat.
⁷ Gambar 1.2 menunjukkan masalah-masalah ini, gambar 1.2a dan 1.2b menunjukkan fitur lokal yang
⁸ mirip dari dua POI yang berbeda, sedangkan gambar 1.2c dan 1.2d menunjukkan objek-objek latar
⁹ belakang yang tidak konsisten pada POI. Penelitian ini akan melakukan analisis untuk menemukan
¹⁰ dan memisahkan fitur-fitur lokal tersebut agar tidak ikut diproses dalam pembuatan model POI.



(a)



(b)



(c)



(d)

Gambar 1.2: Empat gambar di atas menunjukkan permasalahan yang dihadapi pada penelitian ini. Gambar (a) dan (b) merupakan gambar dari dua POI yang berbeda tetapi memiliki logo dengan bagian sudut yang mirip, sudut yang mirip tersebut kemungkinan akan menghasilkan fitur lokal yang mirip juga. Sedangkan pada gambar (c) dan (d) merupakan dua gambar dari POI yang sama tetapi banyak objek latar yang berbeda sehingga akan memunculkan fitur lokal yang tidak konsisten.

¹¹ Fitur-fitur lokal yang tidak unik dan tidak konsisten tersebut akan dipisahkan dengan meng-
¹² gunakan metode *clustering*. Metode *clustering* merupakan teknik pemrosesan data yang akan
¹³ mengelompokkan data-data dengan sifat yang mirip ke dalam satu kelompok. Metode *clustering*
¹⁴ pada penelitian ini akan menggunakan metode *Agglomerative*. Penelitian ini mengasumsikan fitur
¹⁵ lokal yang merepresentasikan suatu POI adalah fitur lokal yang muncul secara konsisten di gambar
¹⁶ POI tersebut dan relatif unik terhadap POI tersebut.

¹⁷ 1.2 Rumusan Masalah

¹⁸ Skripsi ini memiliki rumusan masalah sebagai berikut:

- 1 • Bagaimana membuat model pengenalan POI berdasarkan fitur lokalnya menggunakan teknik
2 *data mining*?
- 3 • Bagaimana mengidentifikasi POI dalam sebuah gambar berisi POI dengan memanfaatkan
4 model pengenalan POI yang telah dibuat?

5 **1.3 Tujuan**

6 Skripsi ini memiliki tujuan sebagai berikut:

- 7 • Membuat perangkat lunak yang akan menghasilkan model pengenalan POI berdasarkan dari
8 *dataset* yang diberikan
- 9 • Membuat perangkat lunak yang dapat melakukan identifikasi POI dari sebuah gambar POI
10 dengan menggunakan model yang dihasilkan.

11 **1.4 Batasan Masalah**

12 Berikut batasan-batasan masalah dari skripsi ini:

- 13 • Pengambilan fitur lokal untuk analisis dilakukan menggunakan metode SIFT dan ORB dengan
14 implementasi OpenCV pada Python.

15 **1.5 Metodologi**

16 Skripsi ini akan memiliki metodologi sebagai berikut:

- 17 1. Melakukan studi literatur tentang metode OIR, teknik ekstraksi fitur lokal SIFT dan ORB,
18 serta teknik-teknik *data mining* yang digunakan pada skripsi ini. Studi literatur dilakukan
19 dengan mencari dan membaca *paper* atau buku yang berkaitan dengan topik tersebut.
- 20 2. Mengumpulkan *dataset* gambar POI yang diperlukan untuk penelitian dan pembuatan model
21 identifikasi.
- 22 3. Melakukan analisis pada latar belakang masalah pengenalan POI, dengan melihat sifat-sifat
23 fitur lokal pada gambar POI.
- 24 4. Menyusun rancangan perangkat lunak.
- 25 5. Melakukan implementasi perangkat lunak.
- 26 6. Menguji kinerja perangkat lunak.
- 27 7. Menulis buku skripsi.

28 **1.6 Sistematika Pembahasan**

29 Sistematika pembahasan yang digunakan pada penelitian ini adalah sebagai berikut:

- 30 1. Bab 1 Pendahuluan

31 Bab ini berisi tentang hal-hal yang menggambarkan skripsi ini secara garis besar. Hal yang
32 dibahas merupakan latar belakang masalah, rumusan masalah, tujuan penelitian, batasan
33 masalah, dan metodologi penelitian.

- 34 2. Bab 2 Landasan Teori

35 Bab ini berisi tentang dasar-dasar teori dari teknik atau metode yang digunakan dalam skripsi

ini, yaitu POI, OIR, metode SIFT, metode ORB, *Best Score Increasing Subsequence* (BSIS), KD-Tree, teknik *clustering Agglomerative*, serta metode SIFT dan ORB di *library* OpenCV.

3. Bab 3 Analisis

Bab ini berisi analisis pada masalah yang dibahas pada skripsi beserta solusi yang digunakan untuk menyelesaikan masalah tersebut.

4. Bab 4 Perancangan

Bab ini berisi tentang perancangan baik dari metode *clustering* pada pemilihan fitur dan perancangan metode identifikasi POI dengan OIR. Bab juga akan berisi rancangan struktur *file* dan *folder* pada hasil akhir perangkat lunak.

5. Bab 5 Implementasi dan Pengujian

Bab ini berisi implementasi perangkat lunak pada metode pemilihan fitur dan identifikasi POI serta pengujian terhadap kinerja kedua metode tersebut.

6. Bab 6 Kesimpulan dan Saran

Bab ini berisi kesimpulan yang didapatkan dari hasil analisis serta keseluruhan implementasi dan pengujian yang dilakukan pada penelitian ini.

1

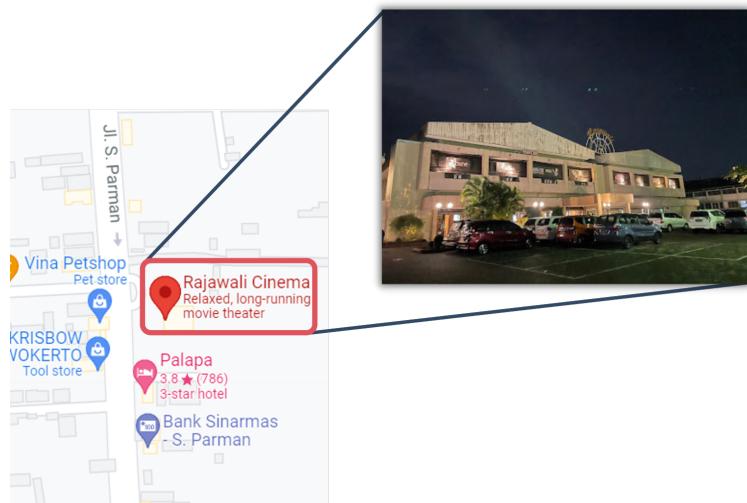
BAB 2

2

LANDASAN TEORI

3 2.1 Point of Interest

4 *Point of Interest* (POI) adalah sebuah lokasi geografis yang memiliki kegunaan tertentu. POI
5 biasanya dikenali oleh banyak orang dan memiliki keunikan tertentu pada tampilannya. Salah satu
6 contoh POI dapat dilihat pada Gambar 2.1. POI juga dapat dimanfaatkan untuk menjadi penanda
7 lokasi seseorang. Seseorang dapat mengerti lokasinya dengan melihat POI yang ada di sekitarnya.



Gambar 2.1: Salah satu contoh POI.

8 Beberapa POI tertentu dapat memiliki logo yang sifatnya unik, contohnya seperti pada POI-POI
9 di Gambar 2.2. POI dengan logo unik tersebut dapat dikenali dengan hanya melihat logonya saja.
10 POI dengan logo unik ini dapat dikenali oleh komputer salah satunya dengan menggunakan teknik
11 *Object Instance Recognition* (OIR).



Gambar 2.2: Contoh POI dengan logo unik.

¹ 2.2 Object Instance Recognition

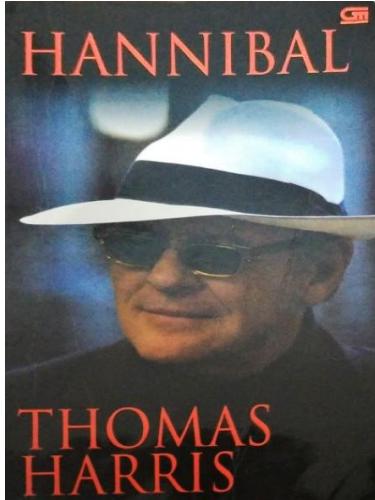
² *Object Instance Recognition* (OIR) adalah teknik pengenalan objek spesifik. Pada teknik OIR
³ deteksi tidak memberikan kelas dari gambar tetapi label yang khusus, seperti contohnya jika objek
⁴ merupakan sebuah mobil, OIR tidak hanya akan memberikan keluaran bahwa objek merupakan
⁵ mobil melainkan hal yang spesifik seperti merk dan jenis mobil tersebut. OIR bekerja dengan
⁶ menerima gambar masukkan dan mencari gambar pada *dataset* yang memiliki paling banyak
⁷ kemiripan.

⁸ Penyelesaian OIR akan mudah bila gambar masukkan merupakan gambar yang sudah bersih.
⁹ Pada praktiknya sebuah algoritma OIR seharusnya tetap dapat mengenali objek walaupun gam-
¹⁰ bar bervariasi. Beberapa perubahan pada gambar berikut merupakan faktor-faktor yang dapat
¹¹ mempersulit proses OIR:

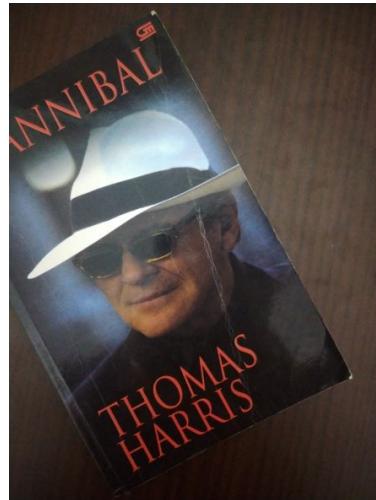
- ¹² • Cahaya
¹³ Perubahan pada tingkat pencahayaan pada gambar yang mengakibatkan perubahan nilai
¹⁴ *pixel-pixel* pada gambar.
- ¹⁵ • Skala
¹⁶ Jarak diambilnya gambar yang berisi objek. Perbedaan ukuran objek pada gambar akan
¹⁷ menyebabkan sudut-sudut pada objek menjadi berbeda.
- ¹⁸ • Rotasi
¹⁹ Orientasi atau arah pengambilan gambar yang berbeda akan mengakibatkan objek pada
²⁰ gambar jadi terlihat berbeda. Sudut-sudut akan menjadi berbeda karena arah hadapnya
²¹ berbeda.
- ²² • Latar Belakang
²³ Objek-objek lain di sekitar objek yang ingin diidentifikasi akan berpotensi mempersulit
²⁴ pemrosesan. Objek-objek tersebut dapat menghasilkan fitur-fitur lokal yang tidak relevan
²⁵ terhadap objek yang ingin diidentifikasi.
- ²⁶ • Bagian Objek Tertutup
²⁷ Adanya objek lain yang menutupi sebagian dari objek yang ingin diidentifikasi akan berpotensi
²⁸ menyebabkan beberapa fitur lokal dari objek tidak terdeteksi.
- ²⁹ • Sudut Pandang
³⁰ Sudut pengambilan gambar yang berbeda akan memengaruhi pemrosesan. Fitur-fitur lokal
³¹ dari objek yang ingin diidentifikasi akan menjadi berbeda.

1 • Translasi

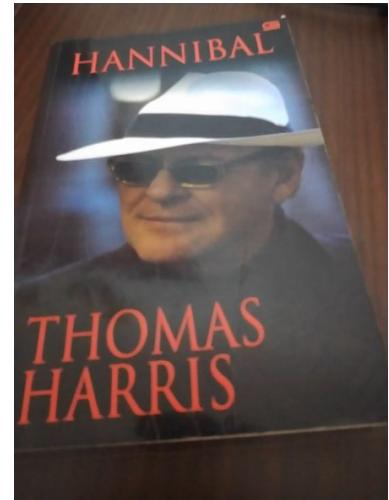
2 Posisi objek yang ingin diidentifikasi dalam gambar akan memengaruhi pemrosesan. Pencarian
3 pasangan fitur lokal tidak dapat dengan hanya menggunakan posisi di mana fitur lokal tersebut
4 ditemukan pada gambar.



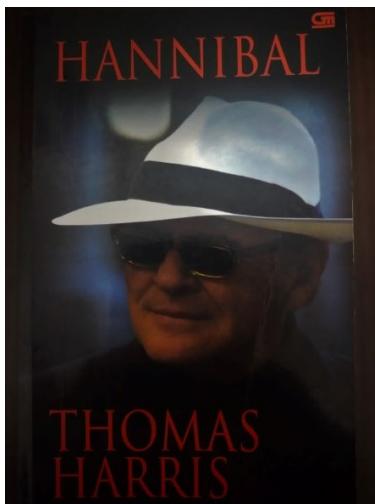
(a) Gambar objek yang ideal, dari sudut tegak lurus



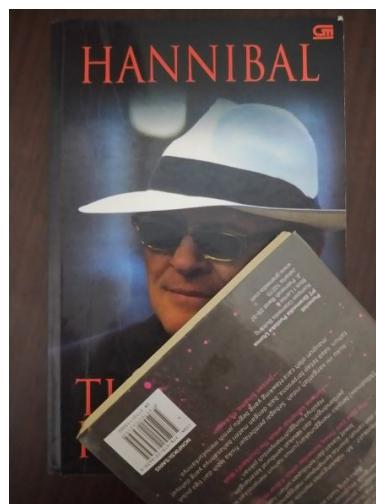
(b) Objek pada gambar mengalami translasi dan rotasi



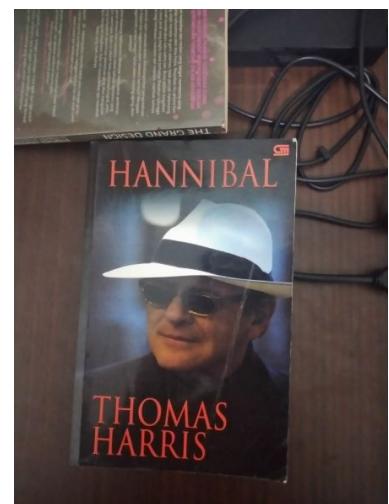
(c) Gambar diambil dari sudut yang berbeda



(d) Gambar dengan pencahayaan yang berbeda



(e) Objek pada gambar terhalang oleh objek lain



(f) Terdapat objek lain di latar belakang dari objek utama

Gambar 2.3: Contoh variasi pada gambar *cover* buku yang dapat menyebabkan masalah pada OIR

5 OIR dapat dilakukan dengan menggunakan fitur lokal. Fitur lokal sendiri merupakan fitur
6 dalam gambar yang mendeskripsikan sebuah daerah tertentu pada gambar tersebut. Fitur lokal
7 dapat berupa perpotongan garis atau yang biasa disebut *keypoint*. Sebuah *keypoint* akan dapat
8 diidentifikasi dengan menggunakan daerah di sekitar *keypoint* tersebut. Deskripsi *keypoint* ini
9 dibuat dalam sebuah vektor yang dinamakan vektor deskriptor.

10 Teknik OIR bekerja dengan melakukan deteksi fitur lokal pada gambar masukkan dan pada
11 gambar-gambar di *dataset*. Fitur-fitur lokal dari gambar masukkan tersebut kemudian dipasangkan
12 dengan fitur lokal dari gambar *dataset*. Gambar yang memiliki pasangan terbanyak akan merupakan
13 hasil deteksi gambar masukkan tersebut.

1 Pengambilan fitur lokal dari gambar dapat dilakukan dengan beberapa metode, seperti SIFT
2 (lihat 2.3) dan ORB (lihat 2.4). Kedua metode tersebut—SIFT dan ORB—sudah menangani
3 masalah perubahan translasi, skala dan rotasi karena fitur lokal yang dihasilkan oleh SIFT dan
4 ORB bersifat invarian terhadap translasi, skala dan rotasi.

5 Proses pencarian pasangan fitur lokal dari gambar masukkan dan *dataset* dilakukan dengan
6 menggunakan vektor deskriptor dari fitur lokal. Perubahan-perubahan pada gambar walaupun
7 sedikit akan menyebabkan perubahan nilai pada vektor deskriptor. Vektor deskriptor dari fitur
8 lokal pada gambar masukkan hampir tidak akan persis sama dengan vektor deskriptor dari gambar
9 yang ada di *dataset*. Oleh karena itu pencarian pasangan fitur lokal dilakukan dengan mencari
10 pasangan fitur lokal yang memiliki nilai kemiripan paling tinggi.

11 Secara garis besar tahapan proses OIR pada penelitian ini adalah sebagai berikut:

12 1. Ekstraksi Fitur

13 Pertama akan dilakukan pengambilan fitur-fitur lokal dari gambar masukkan. Dengan
14 menggunakan metode SIFT atau ORB pengambilan fitur lokal akan menghasilkan *tuple*
15 yang berisi *keypoint*. Setiap *keypoint* yang dihasilkan akan memiliki sebuah vektor yang
16 akan digunakan untuk mengidentifikasi *keypoint* tersebut. Vektor ini disebut sebagai vektor
17 deskriptor.

18 2. Pairing

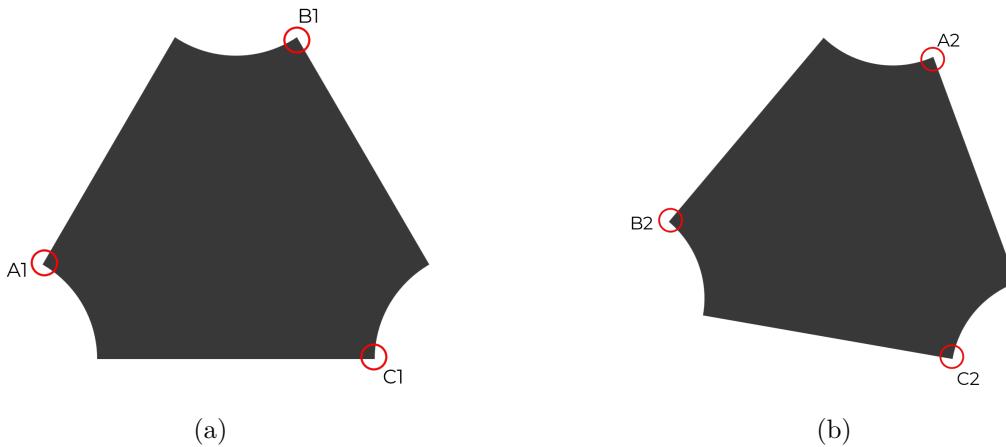
19 Untuk setiap fitur lokal yang terdeteksi pada gambar masukkan akan dicari beberapa fitur
20 lokal dari gambar pada *dataset* yang paling mirip. Kemiripan fitur lokal ditentukan dengan
21 menghitung jarak (sesuai dengan metrik yang digunakan) antara vektor deskriptor dari kedua
22 fitur lokal tersebut.

23 3. Verification

24 Pasangan fitur lokal yang didapat pada tahap sebelumnya merupakan pasangan yang hanya
25 memiliki ciri yang mirip tetapi belum tentu konsisten secara geometris. Pasangan fitur lokal
26 dikatakan konsisten secara geometris jika keduanya memiliki posisi yang sama relatif terhadap
27 pasangan lain di sekitarnya. Penjelasan mengenai pasangan yang konsisten akan dijelaskan
28 pada paragraf di bawah. Hanya pasangan-pasangan fitur lokal yang konsisten secara geometris
29 yang akan diproses lebih lanjut.

30 4. Scoring

31 Setelah didapatkan semua pasangan fitur lokal yang paling mirip dan konsisten secara geometris
32 maka dapat ditentukan pasangan gambar yang menjadi label gambar masukkan. Kemudian
33 perlu dihitung nilai kemiripan dari label yang dihasilkan. Kemiripan dapat dihitung dengan
34 menghitung $\frac{1}{d}$ untuk semua pasangan fitur lokal, di mana d merupakan jarak pasangan
35 tersebut. Nilai-nilai $\frac{1}{d}$ tersebut kemudian dihitung totalnya untuk menjadi nilai kemiripan
36 label hasil.



Gambar 2.4: Ilustrasi pasangan fitur lokal yang konsisten dan tidak konsisten.

- 1 Ilustrasi untuk menunjukkan pasangan yang tidak konsisten dapat dilihat pada Gambar 2.4. Pada
2 kedua gambar tersebut, fitur lokal A1, B1, dan C1 pada Gambar 2.4a secara berurutan dipasangkan
3 dengan fitur lokal A2, B2, dan C2 pada Gambar 2.4b.

4 Pada orientasi seperti di gambar, pasangan A1 dengan A2 dan B1 dengan B2 merupakan
5 pasangan yang tidak konsisten. Hal tersebut dikarenakan posisi A1 yang berada di sebelah kiri
6 B1 tetapi pasangan dari A1, yaitu A2 berada di sebelah kiri pasangan dari B2, pasangan dari B1.
7 Sedangkan pasangan C1 dengan C2 merupakan pasangan yang konsisten, karena baik C1 maupun
8 C2 memiliki posisi relatif yang sama terhadap pasangan fitur lokal lain.

9 Gambar 2.4b dapat dirotasi sebanyak 180 derajat sehingga B2 berada di sebelah kanan A1.
10 Dengan menggunakan orientasi gambar yang seperti ini, pasangan A1 dengan A2 dan pasangan B1
11 dengan B2 menjadi konsisten posisinya secara geometris. Tetapi dengan orientasi yang seperti ini
12 pasangan C1 dengan C2 menjadi tidak konsisten karena posisi C2 pada Gambar 2.4b akan berada
13 di sebelah kiri A2 dan B2.

14 Salah satu metode untuk melakukan verifikasi geometris adalah BSIS (lihat 2.5). BSIS dapat
15 mencari pasangan-pasangan fitur lokal yang posisi relatif konsisten terhadap pasangan lain. Verifikasi
16 yang dilakukan BSIS dilakukan dengan menggunakan beberapa orientasi gambar untuk mendapatkan
17 orientasi yang menghasilkan nilai kemiripan paling tinggi.

18 Cara lain yang dapat dilakukan pada tahap verifikasi adalah dengan melakukan *Lowe's Ratio
19 Test*. Cara ini tidak mencari pasangan yang konsisten secara geometris melainkan hanya menyaring
20 pasangan yang kuat. *Lowe's Ratio Test* mencari pasangan yang kuat dengan membandingkan
21 kemiripan dua pasangan paling mirip untuk tiap fitur lokal di gambar masukkan.

Untuk sebuah fitur lokal Q dari gambar masukkan dihitung nilai kemiripannya dengan setiap fitur lokal gambar *dataset*, didapatkan fitur lokal T1 merupakan fitur lokal yang nilai kemiripannya paling tinggi dan T2 merupakan fitur lokal yang nilai kemiripannya kedua tertinggi. Pasangan fitur lokal Q dan T1 akan dikatakan pasangan yang jika nilai kemiripannya memenuhi persamaan berikut:

$$similarity(Q, T1) > similarity(Q, T2) \times n \quad (2.1)$$

- 22 Variabel n pada persamaan di atas merupakan sebuah nilai konstan yang nilainya lebih dari
23 1. Persamaan tersebut bertujuan untuk mencari pasangan yang benar-benar mirip dan nilai

- 1 kemiripannya berbeda jauh dengan pasangan lainnya. Nilai n dapat diatur untuk menyatakan
 2 seberapa perlu seberapa jauh jarak pasangan termirip dengan kedua termirip.

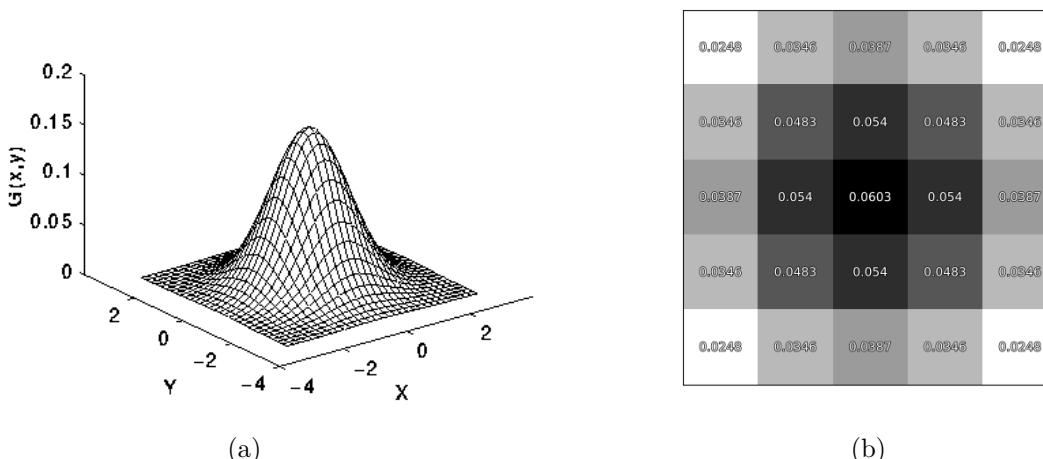
3 2.3 SIFT (Scale Invariant Feature Transform)

4 SIFT adalah salah satu metode pencarian fitur lokal yang dicetuskan pada [1]. Fitur lokal yang
 5 dihasilkan SIFT bersifat invarian terhadap rotasi, perubahan skala, dan translasi pada gambar. Sifat
 6 invarian ini berarti fitur lokal yang sama pada gambar yang telah dirotasi, diubah skalanya, atau
 7 ditranslasi akan tetap memiliki ciri yang mirip. Setiap fitur lokal akan memiliki sebuah vektor yang
 8 mendeskripsikan daerah area fitur lokal tersebut, vektor ini biasa disebut sebagai deskriptor. Vektor
 9 deksriptor SIFT berbentuk vektor bilangan bulat yang memiliki 128 elemen. Tahap pencarian fitur
 10 lokal pada SIFT dapat dibagi menjadi 4 langkah yang akan dijabarkan pada subbab-subbab berikut.

11 2.3.1 Pencarian Extrema

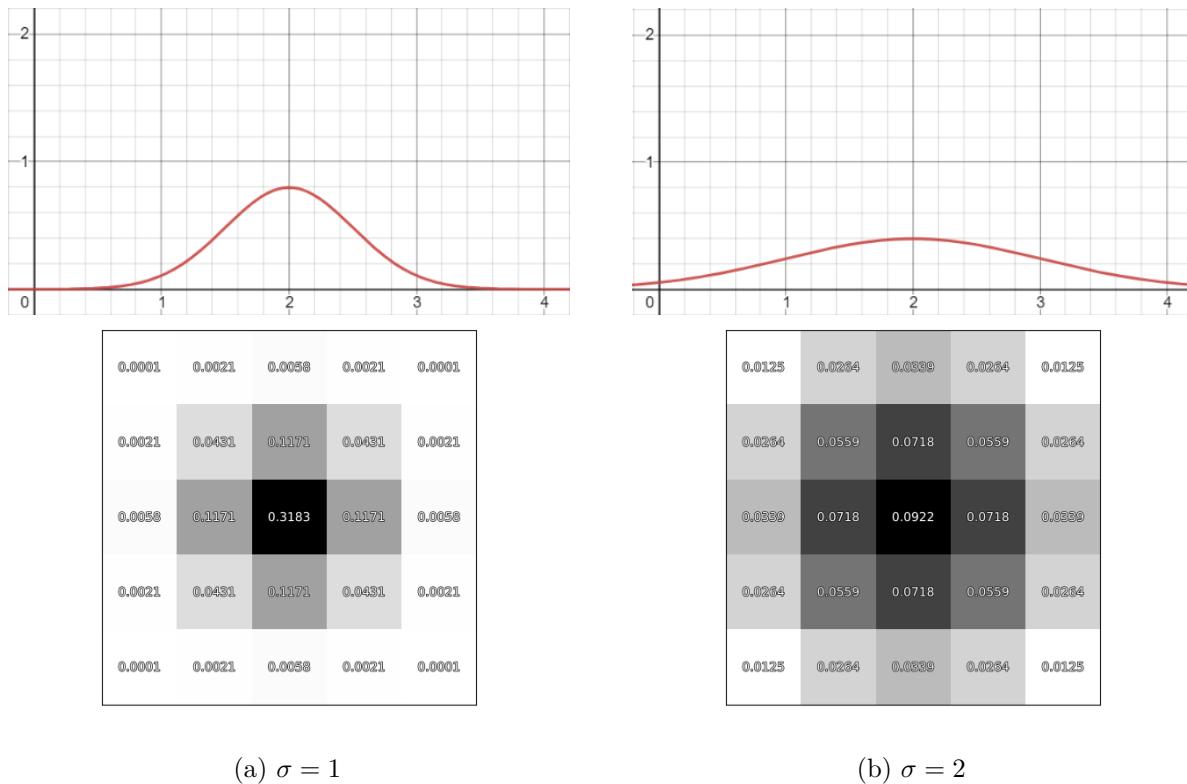
12 Pada tahap ini akan dicari *pixel-pixel* pada gambar yang merupakan *corner* atau biasa disebut
 13 *keypoint*. *Keypoint* pada SIFT dicari dengan memeriksa *pixel-pixel* pada gambar hasil turunan kedua
 14 Gaussian. Turunan kedua Gaussian akan dihitung dengan menggunakan *Difference of Gaussian*
 15 (DoG).

16 Penghitungan DoG ini dilakukan dengan memanfaatkan sifat dari *Matrix Konvolusi Gaussian*.
 17 *Matrix Konvolusi Gaussian* merupakan *matrix* yang memiliki sifat distribusi Gaussian, di mana titik
 18 tengah *matrix* memiliki nilai yang tinggi dan nilai-nilai di sekitarnya semakin mengecil semakin
 19 mendekati tepi *matrix*. Seperti ditunjukkan pada Gambar 2.5.



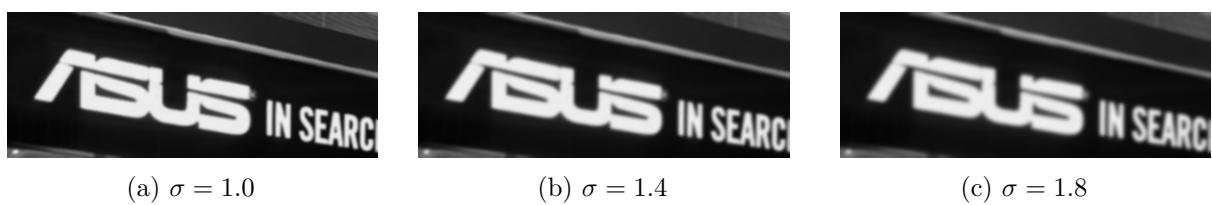
Gambar 2.5: Kurva Gaussian dan bentuk representasi *matrix*-nya.

20 Pada fungsi Gaussian tingkat penyebaran data dapat diatur dengan mengubah parameter σ yang
 21 mengatur nilai deviasi standar. Gambar 2.5a menunjukkan bagaimana nilai σ mengatur bagaimana
 22 data tersebar dari *mean*, di mana semakin tinggi nilai σ maka data akan semakin menyebar. Nilai
 23 yang semakin menyebar menyebabkan perbedaan nilai antar titik semakin kecil. Efeknya pada
 24 *matrix* dapat dilihat pada Gambar 2.6. Nilai σ yang tinggi menyebabkan kurva semakin lebar
 25 dan pada *matrix* selisih nilai antar titik menjadi semakin kecil.



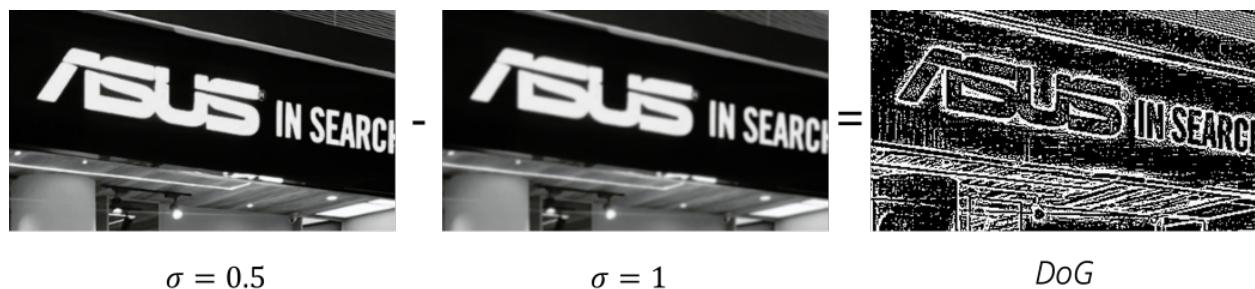
Gambar 2.6: Kurva dan *matrix Gaussian* pada nilai σ yang berbeda.

Matrix Konvolusi Gaussian ketika diaplikasikan pada gambar akan menyebabkan perubahan nilai tiap *pixel* pada gambar. Nilai dari setiap *pixel* akan menjadi mirip dengan *pixel* tetangga di dekatnya. Perubahan nilai *pixel* akan paling berpengaruh pada daerah dengan perubahan nilai *pixel* yang tinggi. Tingkat perubahan nilai dipengaruhi oleh nilai σ yang digunakan, nilai σ yang tinggi akan menyebabkan nilai *pixel* yang berdekatan semakin mirip—perubahan nilai *pixel* pada daerah tersebut semakin mengecil. Jika dilihat pada gambar, maka gambar hasil konvolusi akan terlihat kabur (*blur*). Nilai σ menentukan tingkat *blur* gambar.



Gambar 2.7: Efek nilai σ pada hasil gambar konvolusi.

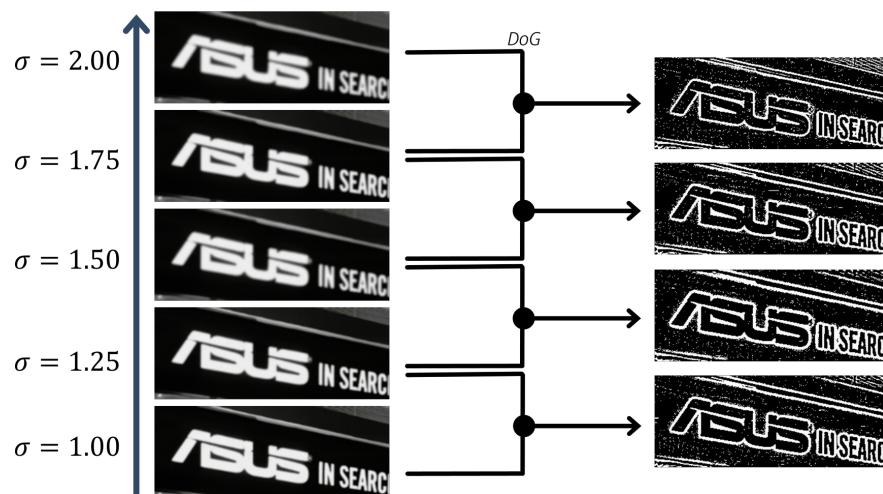
Perubahan nilai σ pada *matrix* konvolusi serta efeknya pada gambar akan dimanfaatkan untuk menghitung *Difference of Gaussian* (DoG). DoG merupakan hasil turunan kedua Gaussian pada gambar. Gambar DoG dapat diperoleh dengan menghitung perbedaan nilai tiap *pixel* dari dua gambar yang telah dikonvolusi oleh *matrix* Gaussian dengan nilai σ yang berbeda. Perbedaan nilai untuk DoG dihitung dengan mengurangi setiap *pixel* pada gambar konvolusi yang memiliki nilai σ yang lebih kecil, dengan setiap *pixel* pada posisi yang sama pada gambar konvolusi yang memiliki nilai σ yang lebih besar. Ilustrasi dapat dilihat pada Gambar 2.8.



Gambar 2.8: Operasi DoG pada gambar

Metode SIFT mencari *keypoint* dengan memanfaatkan konsep DoG. Sebuah gambar akan dikonvolusi dengan *matrix* Gaussian beberapa kali dengan nilai σ yang berbeda. Setelah didapatkan beberapa gambar maka akan dihitung DoG untuk setiap gambar yang nilai σ -nya bersebelahan (Gambar 2.9). Pasangan gambar konvolusi yang berbeda akan menghasilkan gambar DoG yang berbeda juga.

Untuk setiap gambar DoG akan ditentukan *pixel* mana saja yang merupakan *keypoint* dengan mencari *pixel* yang merupakan *extrema*. Sebuah *pixel* merupakan *extrema* jika nilai pixel tersebut lebih besar dari seluruh 26 *pixel* di sekitarnya atau lebih kecil dari seluruhnya. Ke-26 *pixel* tersebut merupakan 8 *pixel* yang mengelilingi, 9 *pixel* pada posisi yang sama dari gambar di atasnya, dan juga 9 *pixel* pada posisi yang sama dari gambar di bawahnya.



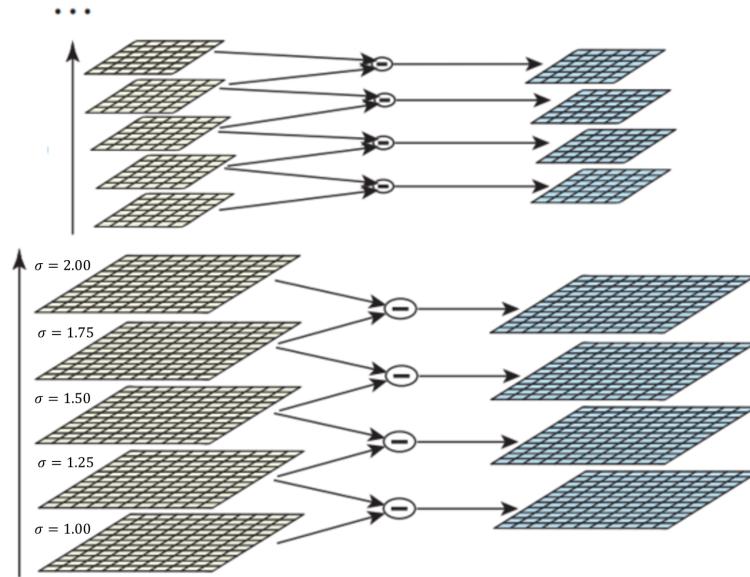
Gambar 2.9: Penggunaan DoG pada SIFT

2.3.2 Penentuan Skala

Pada tahap sebelumnya sudah didapatkan *keypoint-keypoint* dalam gambar. Agar *keypoint* dapat invariant terhadap skala, *keypoint* perlu untuk dapat tetap terdeteksi walaupun ukuran gambar berubah. Untuk setiap *keypoint* perlu untuk dicari skala terkecil di mana *keypoint* tersebut dapat terdeteksi. Untuk mencapai ini SIFT menggunakan lanjutan dari metode pada Gambar 2.9 dengan langkah sebagai berikut (ilustrasi pada Gambar 2.10):

1. Lakukan konvolusi sampai nilai σ sudah mencapai 2 kali nilai awal
2. Perkecil ukuran gambar (*downsample*) menjadi setengah resolusinya
3. Kembalikan nilai σ ke nilai awal
4. Ulang tahap dari langkah 1 hingga gambar sudah terlalu kecil.

Pada langkah di atas setiap siklus ukuran gambar disebut sebagai oktaf, dimulai dari oktaf pertama, lalu kedua, dan seterusnya. Dengan setiap oktaf ukuran gambar akan semakin kecil. Pencarian *keypoint* dilakukan pada tiap oktaf, dan untuk tiap *keypoint* tersebut ditulis nilai oktaf tertinggi (ukuran gambar terkecil) di mana *keypoint* tersebut dapat terdeteksi.



Gambar 2.10: Oktaf pada proses konvolusi SIFT

2.3.3 Penentuan Orientasi

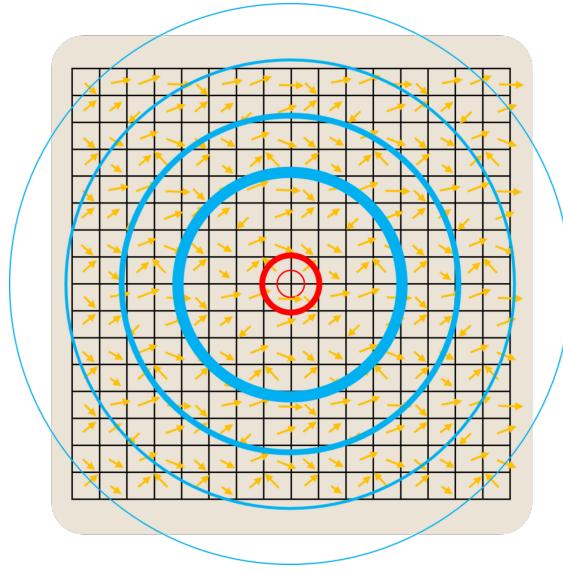
Untuk dapat invariant terhadap rotasi gambar, setiap *keypoint* perlu memiliki orientasi yang konsisten. Untuk mendapatkan orientasi yang sama pada setiap rotasi gambar, orientasi perlu ditentukan dari atribut yang akan selalu sama bagaimanapun gambar dirotasi. Untuk itu orientasi *keypoint* ditentukan dengan menggunakan orientasi yang dominan dari *pixel-pixel* di sekitar *keypoint*. Luas daerah yang digunakan untuk mendapat orientasi ditentukan oleh skala dari *keypoint*.

Penentuan orientasi yang dominan dihitung dengan menggunakan *magnitude* ($m(x, y)$) dan orientasi ($\theta(x, y)$) dari *pixel-pixel* dengan menggunakan rumus pada Persamaan 2.2 dan Persamaan 2.3. $L(x, y)$ pada kedua persamaan tersebut merupakan gambar hasil konvolusi.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (2.2)$$

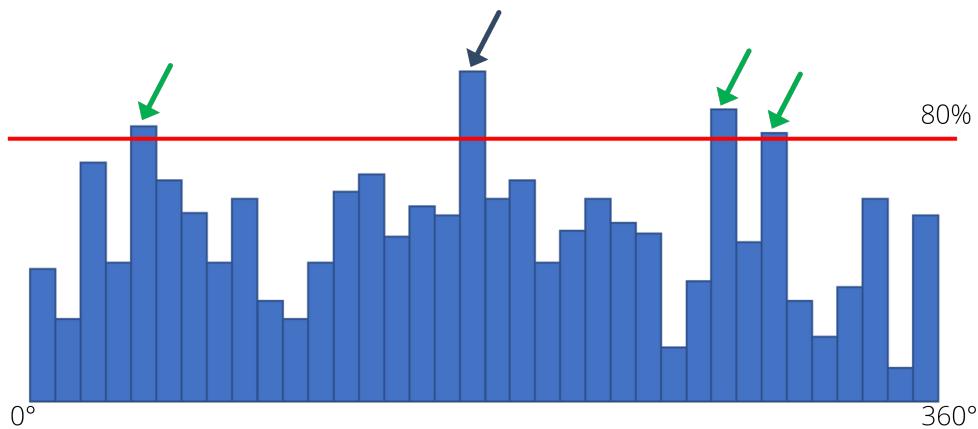
$$\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y))) \quad (2.3)$$

- 1 Setiap *pixel* akan dihitung orientasi dan *magnitude*-nya. *Magnitude* akan digunakan sebagai bobot
 2 dari *pixel* tersebut. Selain *magnitude*, bobot sebuah *pixel* juga dipengaruhi oleh *Gaussian Weighting*,
 3 yang membuat *pixel* yang posisinya dekat dengan titik pusat (titik *keypoint*) akan memiliki bobot
 4 yang lebih tinggi dibanding yang lokasinya jauh dari titik pusat. Ilustrasi pada Gambar 2.11
 5 menunjukkan bagaimana pembobotan dihitung, *pixel-pixel* yang berada dekat dengan *keypoint* (titik
 6 tengah) akan diberi bobot yang lebih besar—ditandai dengan lingkaran yang tebal. Sedangkan
 7 bobot akan semakin berkurang untuk *pixel* yang jauh dari *keypoint*.



Gambar 2.11: Ilustrasi pembobotan pada *Gaussian Weighting*. Titik tengah merupakan *keypoint* yang diperiksa sedangkan setiap kotak merupakan *pixel-pixel* di sekitar *keypoint*. Tanda panah pada tiap kotak menunjukkan *magnitude* dan orientasi *pixel* tersebut, panjang panah merupakan nilai *magnitude* dan arahnya merupakan orientasi

- 8 Setelah setiap *pixel* sudah dihitung orientasi dan bobotnya menggunakan *magnitude* dan *Gaussian*
 9 *Weighting*, nilai bobot tersebut akan dimasukkan ke dalam histogram berdasarkan orientasinya.
 10 Histogram yang digunakan memiliki 36 bin yang masing-masing mewakili 10 derajat orientasi.
 11 Ilustrasi dapat dilihat pada Gambar 2.12.



Gambar 2.12: Histogram untuk menentukan orientasi dari *keypoint*. Bin dengan nilai tertinggi (tanda panah biru) akan digunakan sebagai orientasi dari *keypoint*. Untuk bin lain yang jumlahnya berada dalam rentang 80% dari bin tertinggi (tanda panah hijau) digunakan untuk membuat *keypoint baru*.

- ¹ Dari histogram tersebut puncak nilai bin tertinggi akan digunakan sebagai orientasi dari *keypoint*.
- ² Untuk puncak-puncak lain yang berada dalam rentang 80% dari puncak tertinggi akan digunakan
- ³ untuk membuat *keypoint* baru pada lokasi yang sama dengan orientasi yang berbeda sesuai dengan
- ⁴ nilai orientasi pada bin tersebut.

⁵ 2.3.4 Pembuatan Deskriptor

- ⁶ Setelah didapatkan *keypoint* beserta skala dan orientasinya, perlu untuk diberikan sebuah identitas
- ⁷ pada setiap *keypoint*. Pemberian identitas ini berguna untuk mengidentifikasi *keypoint* yang satu
- ⁸ dengan yang lainnya, agar dapat ditemukan *keypoint-keypoint* dengan ciri yang mirip. Identifikasi
- ⁹ *keypoint* dapat dilakukan dengan membuat sebuah vektor deskriptor. Vektor deskriptor merupakan
- ¹⁰ vektor yang mendeskripsikan daerah di sekitar *keypoint*. Vektor deskriptor pada SIFT berbentuk
- ¹¹ vektor berjumlah 128 elemen bilangan bulat.

¹² Pembuatan vektor dilakukan dengan mengambil daerah di sekitar *keypoint* dari gambar yang
¹³ terlebih dahulu dirotasi sesuai dengan orientasi *keypoint*. Ukuran daerah tersebut ditentukan
¹⁴ berdasarkan dari skala *keypoint*. Daerah tersebut kemudian dibagi menjadi 4×4 subdaerah. Untuk
¹⁵ setiap subdaerah dihitung nilai *magnitude* dan orientasi setiap *pixel*-nya dengan diberi bobot
¹⁶ menggunakan *Gaussian Weighting* lalu hasilnya dimasukkan ke dalam histogram dengan 8 bin.
¹⁷ Setiap bin dalam histogram mewakili 45 derajat orientasi. Nilai dari setiap bin pada histogram
¹⁸ akan menjadi satu elemen pada deskriptor. Terdapat total 16 subdaerah dengan setiap daerah
¹⁹ menghasilkan 8 elemen, sehingga didapat total sebanyak $16 \times 8 = 128$ elemen untuk vektor deskriptor.

²⁰ 2.3.5 SIFT di OpenCV

- ²¹ *Library* OpenCV di Python memiliki modul implementasi SIFT. Modul dapat digunakan untuk
- ²² melakukan ekstraksi fitur lokal dari gambar dan menghasilkan vektor deskriptor untuk tiap fitur
- ²³ lokal. Untuk melakukan deteksi fitur lokal, pertama perlu untuk dibuat objek SIFT dengan
- ²⁴ menggunakan fungsi `SIFT_create`. Beberapa parameter dari fungsi `SIFT_create` yang relevan
- ²⁵ terhadap skripsi ini adalah sebagai berikut:

- **nfeatures**: jumlah fitur yang ingin dihasilkan. Dipilih berdasarkan skor yang didapat dari nilai kontras pada daerah sekitar *keypoint*.

- **nOctaveLayers**: jumlah lapisan untuk tiap oktaf. Ditentukan secara otomatis dari resolusi gambar.

- **sigma**: nilai σ Gaussian yang digunakan pada oktaf pertama.

Objek SIFT tersebut lalu digunakan untuk mendeteksi fitur lokal dengan menggunakan fungsi `detectAndCompute`. Fungsi `detectAndCompute` menerima beberapa parameter berikut:

- **image**: gambar masukkan yang ingin dideteksi *keypoint*-nya. Berbentuk *array* 2 dimensi.

- **mask**: *array* 2 dimensi dengan ukuran yang sama dengan gambar, menentukan *pixel* mana yang digunakan dan yang tidak.

Fungsi `detectAndCompute` tersebut akan menghasilkan sebuah *tuple* yang berisi objek *keypoint* dan *array* berjumlah 128 elemen sebanyak *keypoint* yang terdeteksi. Objek *keypoint* yang dihasilkan memiliki atribut sebagai berikut:

- **pt**: atribut ini memiliki dua buah atribut **x** dan **y** yang menunjukkan koordinat dari *keypoint*.

- **size**: diameter daerah di sekitar *keypoint* yang diperiksa.

- **angle**: orientasi dari *keypoint*.

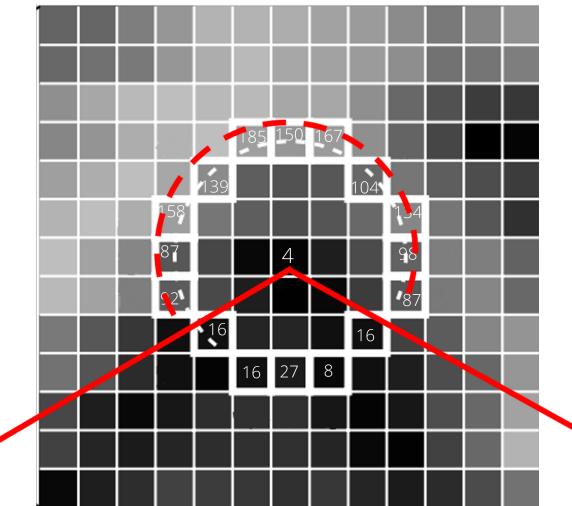
- **octave**: oktaf di mana *keypoint* tersebut didapat.

2.4 ORB (Oriented FAST and Rotated BRIEF)

ORB adalah metode pencarian fitur lokal yang dijelaskan pada [2]. ORB dapat menemukan fitur lokal dengan lebih cepat jika dibandingkan dengan SIFT, walaupun fitur lokal yang dihasilkan tidak seakurat yang dihasilkan SIFT. ORB mencari fitur lokal dengan mencari *pixel* yang merupakan sudut perpotongan garis (*Keypoint*). *Keypoint* dalam ORB dicari dengan ide bahwa sebuah *pixel* yang merupakan sudut akan memiliki garis kontinu membentuk lingkaran di sekitarnya dengan nilai intensitas yang lebih kecil atau lebih besar dari nilai intensitas *pixel* tersebut. Fitur lokal yang dihasilkan ORB bersifat invariant terhadap rotasi, translasi dan invariant sebagian terhadap skala. Fitur lokal ORB juga memiliki sebuah vektor deskriptor yang berbentuk vektor sebanyak 256 elemen bilangan biner. Tahap pencarian fitur lokal pada ORB dibagi menjadi 4 langkah yang dijelaskan pada subbab-subbab berikut.

2.4.1 Pencarian Keypoint

Untuk menentukan apakah sebuah *pixel* dalam gambar merupakan *keypoint*, ORB mengambil nilai *pixel* tersebut dan 16 *pixel* di sekitarnya yang membentuk lingkaran. Dari ke 16 *pixel* tersebut dibandingkan nilai intensitasnya dengan nilai intensitas *pixel* yang berada di tengah (*p*), yaitu *pixel* yang ingin diperiksa apakah merupakan *keypoint*. Sebuah *pixel* merupakan *keypoint* jika dari 16 *pixel* di sekitarnya terdapat setidaknya *n* *pixel* kontinu yang nilai intensitasnya lebih besar dari nilai *p* + *t* atau lebih kecil dari *p* - *t*. Proses ini diilustrasikan pada Gambar 2.13



Gambar 2.13: Ilustrasi penentuan *keypoint* pada ORB. Setiap kotak menunjukkan sebuah *pixel* pada gambar dan angka di dalamnya merupakan nilai intensitasnya. *Pixel* di tengah gambar (*pixel* bernilai 4) merupakan *pixel* yang akan diperiksa apakah merupakan *keypoint*. *Pixel-pixel* yang ditandai dengan kotak putih merupakan 16 *pixel* yang digunakan untuk memeriksa apakah *pixel* di tengah merupakan *keypoint*.

1 Pada ilustrasi di Gambar 2.13 *pixel* bernilai 4 yang berada di tengah gambar dapat menjadi
 2 *keypoint* tergantung dari parameter n dan t yang digunakan. Di sekitar *pixel* tersebut terdapat 11
 3 *pixel* kontinu yang nilai intensitasnya lebih besar setidaknya 83 satuan dari nilai intensitas *pixel*
 4 tersebut.

5 *Keypoint-keypoint* yang dihasilkan pada tahap ini belum tentu merupakan sebuah *corner*. Metode
 6 yang digunakan ORB akan mendeteksi sebuah titik terang yang dikelilingi lingkaran dengan titik
 7 gelap atau sebaliknya menjadi sebuah *keypoint*. Untuk itu perlu untuk disaring untuk mendapatkan
 8 hanya *keypoint* yang merupakan *corner*.

9 Penentuan apakah *keypoint* merupakan *corner* pada ORB dilakukan dengan menggunakan
 10 *Harris Corner Measure*. *Harris Corner Measure* akan memberikan sebuah skor untuk sebuah daerah
 11 di sekitar *pixel* untuk menunjukkan seberapa daerah tersebut merupakan *corner*.

12 *Harris Corner Measure* menentukan apakah sebuah daerah merupakan *corner* dengan meng-
 13 gunakan sebuah *window* yang terpusat pada titik *keypoint*. *Window* tersebut lalu digerakkan ke
 14 beberapa arah dan dihitung intensitasnya untuk tiap pergerakan. Daerah tersebut merupakan
 15 *corner* jika terdapat perubahan nilai intensitas ke arah manapun *window* tersebut digerakkan.

16 Teknik *Harris Corner Measure* akan menghasilkan sebuah skor M yang menunjukkan nilai
 17 seberapa daerah yang diperiksa merupakan *corner*. ORB akan mengambil N *keypoint* dengan nilai
 18 M tertinggi untuk ditentukan sebagai *keypoint*.

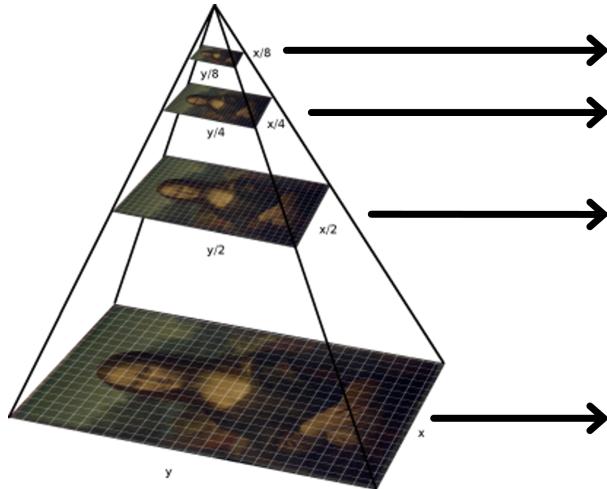
19 2.4.2 Penentuan Skala

20 *Keypoint* yang telah dideteksi pada tahap awal perlu untuk dapat terdeteksi juga walaupun ukuran
 21 gambar berubah agar sifatnya invariant terhadap skala. *Keypoint* yang invariant terhadap skala
 22 adalah *keypoint* dengan ciri yang tetap walaupun dideteksi pada ukuran gambar yang berbeda.

23 Untuk mencapai sifat ini ORB menggunakan metode *Image Pyramid*. ORB menggunakan *Image*
 24 *Pyramid* dengan cara memperkecil ukuran gambar beberapa kali dan untuk setiap ukuran gambar

1 dilakukan deteksi untuk *keypoint*. Dengan menggunakan cara ini *keypoint* tidak akan benar-benar
 2 invariant terhadap perubahan skala, *keypoint* hanya invariant terhadap beberapa skala gambar yang
 3 digunakan pada *pyramid*.

4 Ilustrasi dapat dilihat pada Gambar 2.14. Pada ilustrasi tersebut gambar awal diperkecil
 5 beberapa kali dengan membagi panjang dan lebarnya menjadi setengahnya. Untuk setiap ukuran
 6 gambar dicari *keypoint-keypoint*-nya.



Gambar 2.14: *Image Pyramid* pada ORB

7 Tahap ini akan menghasilkan *keypoint-keypoint* dengan skala yang berbeda sesuai dengan ukuran
 8 gambar di mana *keypoint* tersebut ditemukan. *Keypoint-keypoint* yang dihasilkan tersebut juga
 9 perlu untuk disaring dengan menggunakan *Harris Corner Measure* seperti yang dijelaskan pada
 10 subbab sebelumnya.

11 2.4.3 Penentuan Orientasi

Orientasi *keypoint* pada ORB ditentukan oleh sudut antara sumbu *x* dan vektor dari *keypoint* menuju titik *Intensity Centroid* dari daerah sekitarnya. *Intensity Centroid* pada sebuah daerah gambar merupakan titik di mana terjadi perubahan nilai intensitas terbesar. Titik *Intensity Centroid* (*C*) didefinisikan pada Persamaan 2.4.

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \quad (2.4)$$

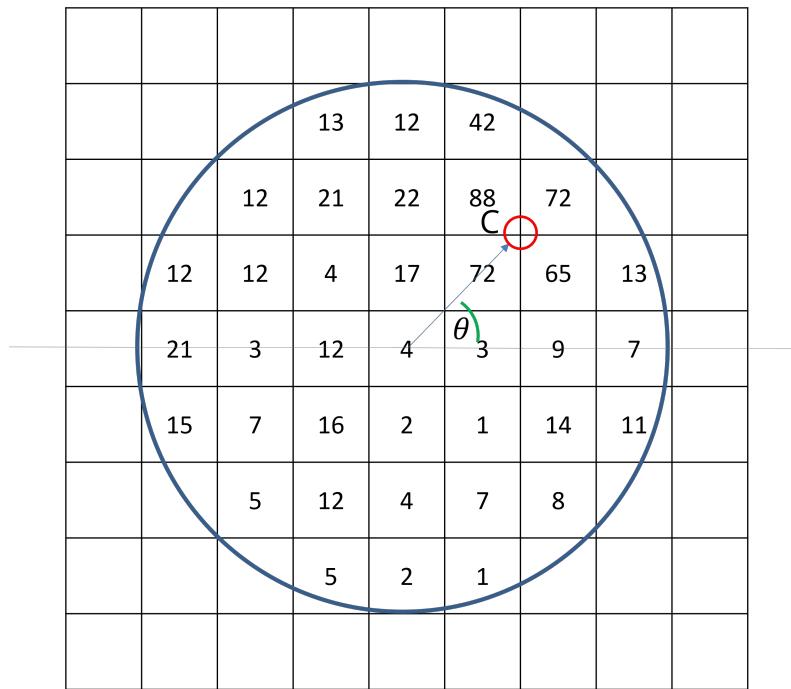
Centroid ditentukan dengan menghitung *moment* pada gambar yang didefinisikan pada Persamaan 2.5.

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \quad (2.5)$$

12 Pada Persamaan 2.4 m_{10} merupakan *moments* gambar dari arah sumbu *x*. *Moments* dari arah
 13 sumbu *x* merupakan nilai perubahan intensitas yang dihitung dari sumbu tersebut, dihitung dari
 14 total seluruh nilai *pixel* dikalikan dengan posisinya pada sumbu *x*. Sedangkan m_{01} merupakan
 15 *moments* dari arah sumbu *y* yang dihitung dengan cara yang sama. Kedua nilai tersebut— m_{10} dan
 16 m_{01} —masing-masing dibagi oleh m_{00} yang didapat dengan menghitung jumlah dari nilai semua
 17 *pixel* pada daerah yang diperiksa.

1 Titik *centroid* dihitung pada daerah yang dikelilingi 16 *pixel* yang digunakan pada tahap
 2 Penentuan *Keypoint*. Dari daerah tersebut didapat titik yang merupakan *centroid*. Lalu dibuat garis
 3 vektor yang berasal dari titik *keypoint* (titik tengah) menuju titik *centroid*. Orientasi ditentukan
 4 dari sudut antara garis lurus sumbu x dengan garis vektor.

5 Proses penentuan orientasi ini diilustrasikan pada Gambar 2.15. Pada gambar tersebut titik
 6 dengan lingkaran hijau merupakan titik ditemukannya *keypoint*. Penentuan orientasi dilakukan
 7 dengan mencari titik *centroid* pada daerah yang dikelilingi lingkaran biru. Setelah didapat titik
 8 *centroid* (lingkaran merah) maka orientasi merupakan garis lurus dan vektor dari titik *keypoint* ke
 9 *centroid*, ditunjukkan oleh θ .



Gambar 2.15: Ilustrasi penentuan orientasi pada ORB.

10 2.4.4 Pembuatan Deskriptor

11 Untuk setiap *keypoint* yang dihasilkan perlu untuk dibuat sebuah vektor yang mendeskripsikan
 12 daerah di sekitar *keypoint* tersebut. Vektor deskriptor berguna untuk melakukan identifikasi pada
 13 *keypoint* tersebut. Pada ORB, vektor deskriptor berbentuk vektor biner berjumlah 256 elemen.
 14 Vektor didapat dengan melihat perbandingan nilai intensitas *pixel* pada daerah di sekitar *keypoint*.

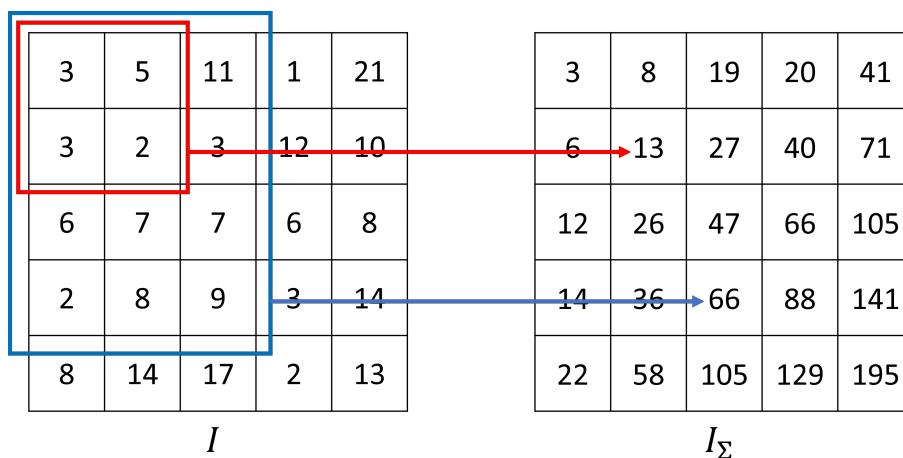
Agar deskriptor bersifat invarian terhadap rotasi gambar terlebih dahulu dirotasi sesuai dengan orientasi yang sudah didapat sebelumnya. Dari gambar yang telah dirotasi diambil daerah berukuran 31×31 untuk dilakukan *binary test*. *Binary test* adalah fungsi yang membandingkan nilai intensitas antar dua titik pada gambar. Fungsi *binary test* τ didefinisikan sebagai berikut:

$$\tau(p; x, y) = \begin{cases} 1 : p(x) < p(y), \\ 0 : p(x) \geq p(y) \end{cases} \quad (2.6)$$

15 Fungsi tersebut akan menghasilkan nilai 1 atau 0 bergantung pada nilai intensitas dari dua titik
 16 yang dibandingkan.

Hasil dari *binary test* yang dilakukan dengan membandingkan nilai dari pasangan-pasangan *pixel* pada gambar akan sangat terpengaruh oleh *noise* yang ada pada gambar. Jika pada saat melakukan *binary test* digunakan *pixel* yang merupakan *noise* maka hasilnya akan tidak representatif terhadap sifat daerah sebenarnya. Untuk menangani masalah ini perlu terlebih dahulu dilakukan *smoothing* pada gambar tersebut. ORB melakukan *smoothing* dengan menggunakan *integral image*.

Integral image adalah sebuah *matrix* 2 dimensi yang dapat digunakan untuk menghitung hasil penjumlahan semua nilai *pixel* di sebuah daerah pada gambar. *Integral Image* dapat mempercepat proses penghitungan total nilai *pixel* pada sebuah daerah dari gambar dengan ukuran berapapun. Nilai sebuah elemen pada koordinat (x, y) di *Integral Image* adalah total penjumlahan semua nilai yang posisi koordinatnya lebih kecil atau sama dengan (x, y) , seperti diilustrasikan pada Gambar 2.16.



Gambar 2.16: Ilustrasi penghitungan *Integral Image*. *Matrix I* merupakan *matrix* awal dan *Matrix I_Σ* merupakan *Integral Image* dari *I*.

Gambar 2.16 menunjukkan cara menghitung nilai elemen pada sebuah koordinat. Kotak pada *Matrix I* merupakan daerah yang dihitung total nilainya untuk mendapatkan nilai pada *Matrix I_Σ* yang ditunjuk oleh tanda panah.

Matrix Integral Image yang sudah dihasilkan dapat digunakan untuk menghitung nilai total dari sebuah daerah berukuran berapapun dengan melakukan operasi penjumlahan dan pengurangan pada 4 angka. Sebuah daerah pada Gambar I yang sudah dibuat *Integral Image*-nya (*I_Σ*) yang berada pada persegi dengan titik pojok kiri atas (T_x, T_y) dan pojok kanan bawah (B_x, B_y) dapat dihitung nilai total elemennya dengan persamaan berikut:

$$Su(T_x, T_y, B_x, B_y) = I_{\Sigma}(B_x, B_y) - I_{\Sigma}(B_x, T_y - 1) - I_{\Sigma}(T_x - 1, B_y) + I_{\Sigma}(T_x - 1, T_y - 1) \quad (2.7)$$

Proses penghitungan nilai total sebuah daerah menggunakan *Integral Image* diilustrasikan pada Gambar 2.17. Total nilai elemen dari daerah yang di dalam kotak biru pada Gambar I dapat dihitung dengan menjumlahkan elemen pada Gambar *I_Σ* yang diberi kotak hijau dan menguranginya dengan elemen yang diberi kotak merah. Daerah di dalam kotak biru tersebut memiliki nilai total $141 - 41 - 36 + 8 = 72$.

3	5	11	1	21		3	8	19	20	41
3	2	3	12	10		6	13	27	40	71
6	7	7	6	8		12	26	47	66	105
2	8	9	3	14		14	36	66	88	141
8	14	17	2	13		22	58	105	129	195

I I_{Σ}

Gambar 2.17: Penghitungan nilai total sebuah daerah dengan menggunakan *Integral Image*

¹ *Smoothing* pada ORB dilakukan dengan mengubah *binary test* yang dilakukan. *Binary test*
² pada ORB dilakukan dengan membandingkan nilai total *pixel* dari dua daerah berukuran 5×5
³ dalam daerah 31×31 yang digunakan untuk menghitung deskriptor. Penghitungan nilai total pada
⁴ daerah 5×5 dilakukan dengan menggunakan *Integral Image* untuk mempercepat proses.

⁵ *Binary test* pada ORB dilakukan sebanyak 256 kali untuk menghasilkan 256 elemen untuk
⁶ vektor. ORB menggunakan 256 pasangan *binary test* yang sudah ditentukan sebelumnya. Pasangan-
⁷ pasangan tersebut didapat dari eksperimen yang dilakukan pada [2].

⁸ ORB menghasilkan vektor bilangan biner sebagai *descriptor* untuk tiap fitur lokal. Karena vektor
⁹ *descriptor* ORB berbentuk vektor biner maka penghitungan jarak antar vektor harus dilakukan
¹⁰ dengan menggunakan metrik jarak Hamming. Metrik jarak Hamming merupakan menghitung jarak
¹¹ antar dua vektor dengan menghitung berapa banyak elemen pada lokasi sama yang nilainya berbeda.
¹² Contoh penghitungan jarak Hamming antar dua vektor biner dapat dilihat pada Gambar 2.18.

A [10011011]
 B [10110111]

Gambar 2.18: Ilustrasi penghitungan jarak Hamming antara dua vektor.

¹³ Gambar 2.18 menunjukkan ilustrasi penghitungan jarak Hamming antar dua vektor biner A
¹⁴ dan B. Vektor A dan B memiliki jarak Hamming 3 karena terdapat 3 elemen di posisi sama yang
¹⁵ nilainya berbeda. Elemen-elemen berbeda tersebut ditunjukkan dengan warna merah pada gambar.

¹⁶ 2.4.5 ORB di OpenCV

¹⁷ *Library* OpenCV di Python memiliki implementasi untuk metode ORB. Implementasi ORB ini
¹⁸ memiliki kegunaan dan cara penggunaan yang mirip dengan implementasi SIFT. Untuk melakukan
¹⁹ deteksi fitur lokal perlu untuk terlebih dahulu dibuat objek ORB dengan fungsi `ORB_create`.
²⁰ Beberapa parameter fungsi `ORB_create` yang relevan terhadap skripsi ini adalah:
²¹ • `nfeatures`: jumlah maksimum fitur lokal yang dihasilkan.

- 1 • `scaleFactor`: rasio pengurangan resolusi pada setiap level *pyramid*.
 - 2 • `nlevels`: jumlah tingkatan (*level*) pada *pyramid*.
 - 3 • `firstLevel`: tingkat dalam *pyramid* sebagai tempat gambar asli.
 - 4 • `patchSize`: ukuran daerah yang digunakan untuk membuat deskriptor.
- 5 Objek ORB tersebut lalu dapat digunakan untuk mendeteksi fitur lokal dalam gambar dengan
6 menggunakan fungsi `detectAndCompute`. Fungsi `detectAndCompute` menerima beberapa parameter
7 berikut:
- 8 • `image`: gambar masukkan yang ingin dideteksi *keypoint*-nya. Berbentuk *array* 2 dimensi.
 - 9 • `mask`: *array* 2 dimensi dengan ukuran yang sama dengan gambar, menentukan *pixel* mana
10 yang digunakan dan yang tidak.
- 11 Fungsi tersebut menghasilkan *tuple* berisi objek *keypoint* dan array vektor deskriptor untuk tiap
12 objek *keypoint*. Setiap vektor merupakan vektor berisi bilangan biner berjumlah 256 elemen. Objek
13 *keypoint* memiliki atribut sebagai berikut:
- 14 • `pt`: atribut ini memiliki dua buah atribut `x` dan `y` yang menunjukkan koordinat dari *keypoint*.
 - 15 • `size`: diameter daerah di sekitar *keypoint* yang diperiksa.
 - 16 • `angle`: orientasi dari *keypoint*.
 - 17 • `octave`: lapisan *pyramid* di mana *keypoint* tersebut didapatkan.

18 2.5 BSIS (Best Score Increasing Subsequence)

19 Pada tahapan OIR (lihat 2.2) sebuah pasangan fitur lokal perlu untuk memiliki sifat yang mirip
20 (dilihat dari vektor deskriptornya) dan juga konsisten secara geometris. Pasangan yang konsisten
21 secara geometris adalah pasangan yang memiliki posisi spasial yang konsisten terhadap pasangan
22 lain di sekitarnya (contoh dapat dilihat pada Gambar 2.4 di 2.2). BSIS [3] adalah salah satu metode
23 yang dapat digunakan untuk menentukan apakah pasangan fitur lokal bersifat konsisten secara
24 geometris.

25 BSIS merupakan modifikasi metode WLIS [4] yang merupakan implementasi dari OIR. Modifikasi
26 terdapat pada tahap *Pairing*, *Verification*, dan *Scoring*.

27 Tahapan dari BSIS dilakukan setelah fitur lokal telah diekstrak dari gambar *query* dan gambar
28 *training* (gambar yang ada di *dataset*). Tahapan BSIS ini dilakukan untuk setiap pemasangan
29 gambar *query* dan gambar *training*. Pasangan gambar dengan skor tertinggi ditentukan sebagai
30 pasangan yang benar dari gambar *query*. Rincian tahapan dijabarkan pada subbab-subbab di bawah
31 ini.

32 2.5.1 Pairing

33 Pada tahap awal setiap fitur lokal pada gambar *query* akan dicari N fitur lokal dari gambar
34 *training* yang nilai kemiripan vektor deskriptornya paling tinggi. Pencarian N pasangan paling
35 mirip dilakukan dengan menggunakan KD-Tree (lihat 2.6) untuk mempercepat waktu komputasi.
36 Pasangan-pasangan ini akan memiliki sebuah skor kemiripan yang dihitung dari vektor deskriptor
37 kedua fitur lokal. Tidak semua dari N pasangan paling mirip akan digunakan dalam proses BSIS.

38 Untuk setiap fitur lokal, BSIS hanya akan mengambil beberapa pasangan yang benar-benar
39 mirip atau paling berpeluang menjadi pasangan yang benar. Pasangan yang benar-benar mirip

1 pada BSIS adalah pasangan yang nilai kemiripannya tinggi dan memiliki perbedaan yang cukup
 2 jauh dengan nilai kemiripan pasangan lain.

3 BSIS menganggap pasangan-pasangan yang sangat mirip adalah pasangan yang nilai kemiri-
 4 pannya tinggi dan merupakan *outlier*. Penentuan *outlier* dilakukan dengan menggunakan *mean*
 5 dan deviasi standar. Penggunaan *mean* dan deviasi standar untuk menentukan *outlier* karena
 6 BSIS mengasumsikan bahwa nilai kemiripan pasangan-pasangan dari sebuah fitur lokal terdistribusi
 7 secara normal.

8 BSIS menentukan *outlier* seperti pada Persamaan 2.8. Untuk sebuah fitur lokal gambar *training*
 9 P_Q yang dipasangkan dengan gambar *training* P_T , pasangan tersebut hanya akan digunakan jika
 10 nilai kemiripannya lebih dari *mean* (m) ditambah konstanta K dikali deviasi standar (σ). *Mean*
 11 merupakan rata-rata dari nilai kemiripan dari N pasangan P_Q dan σ merupakan nilai deviasi
 12 standarnya sedangkan untuk konstanta digunakan nilai $K = 4$.

$$\text{similarity}(P_Q, P_T) > m + (K \times \sigma) \quad (2.8)$$

13 Pasangan-pasangan yang telah dipilih (merupakan *outlier*) tersebut lalu diberi bobot berdasarkan
 14 nilai kemiripannya. Bobot sebuah pasangan didapatkan dengan menghitung seberapa jauh nilai
 15 kemiripan pasangan tersebut terhadap rata-rata dengan memperhatikan penyebaran nilainya. Bobot
 16 (P_w) untuk sebuah pasangan dari fitur lokal P_Q dan P_T didefinisikan pada Persamaan 2.9.

$$P_w(P_Q, P_T) = \left(\frac{\text{similarity}(P_Q, P_T) - m}{\sigma} \right)^2 \quad (2.9)$$

17 P_Q merupakan fitur lokal gambar *query* dan P_T merupakan fitur lokal dari gambar *training*.
 18 Nilai P_w yang semakin tinggi menunjukkan bahwa pasangan tersebut semakin berpotensi untuk
 19 menjadi pasangan yang tepat.

20 2.5.2 Verification

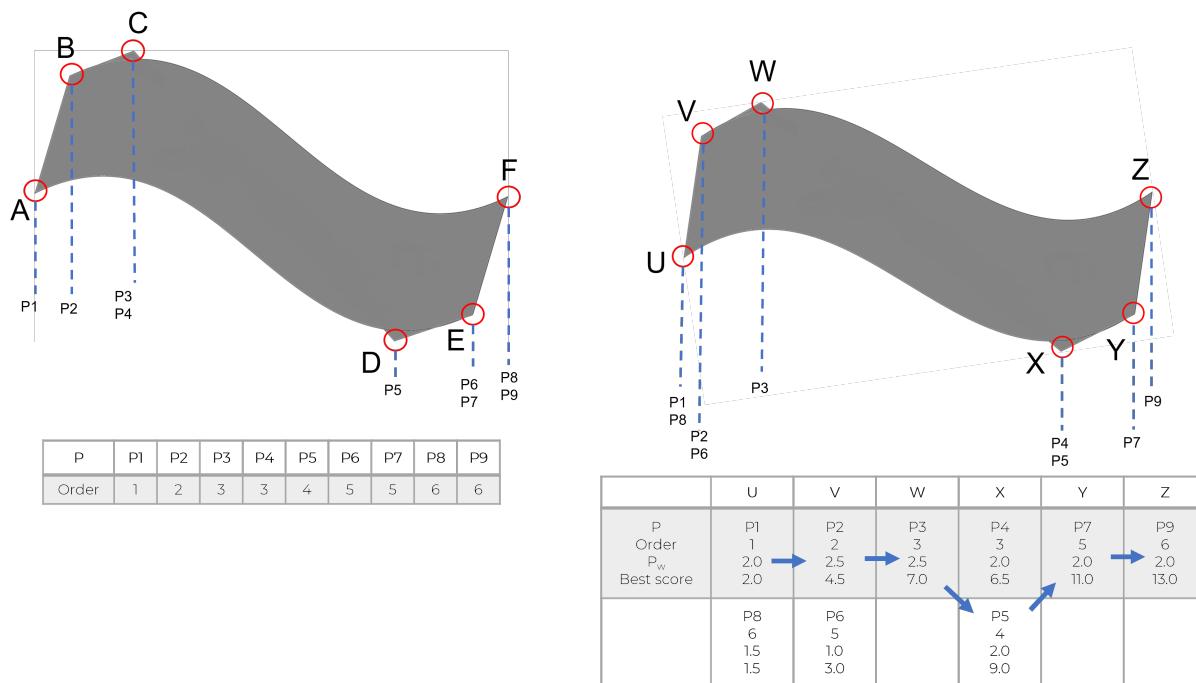
21 Pada tahap *Pairing* telah didapat beberapa pasangan antara fitur lokal gambar *query* dan gambar
 22 *training* yang paling berpotensi merupakan pasangan yang benar. Dari tahap sebelumnya setiap
 23 fitur lokal pada gambar *query* dapat dipasangkan dengan lebih dari 1 fitur lokal pada gambar
 24 *training*. Pasangan-pasangan tersebut akan diperiksa pada tahap ini untuk mencari pasangan mana
 25 saja yang konsisten secara geometris. Setiap fitur lokal baik dari gambar *query* maupun *training*
 26 hanya akan berada dalam satu pasangan setelah dilakukan verifikasi.

27 Pasangan yang konsisten secara geometris adalah pasangan-pasangan yang memiliki posisi spasial
 28 yang konsisten terhadap pasangan fitur lokal lain di sekitarnya. Sifat konsisten ini ditunjukkan oleh
 29 posisi fitur-fitur lokal pada gambar *query* dan gambar *training*. Sebagai contoh, dua buah fitur
 30 lokal dari gambar *query* PQ_1 dan PQ_2 dipasangkan dengan fitur lokal PT_1 dan PT_2 dari gambar
 31 *training*. Jika PQ_1 berada di sebelah kiri PQ_2 (PQ_1 muncul lebih dulu dalam proyeksi sumbu x)
 32 maka PT_1 juga harus berada di sebelah kiri PT_2 untuk agar kedua pasangan tersebut dikatakan
 33 konsisten secara geometris.

34 Tahap *Verification* pada BSIS bertujuan untuk menemukan pasangan-pasangan fitur lokal yang
 35 konsisten secara geometris sesuai dengan penjelasan di atas. BSIS melakukan verifikasi dengan
 36 memproyeksikan seluruh fitur lokal dari gambar *query* dan *training* pada sumbu x dan mencari
 37 pasangan yang konsisten dan memiliki skor yang paling tinggi. Langkah-langkah verifikasi pada

1 BSIS secara rinci adalah sebagai berikut:

- 2 1. Ambil semua fitur lokal dari gambar *query* dan proyeksikan pada sumbu *x*. Setiap fitur lokal akan diberi sebuah *order* sesuai dengan urutan kemunculannya.
- 3 2. Ambil semua fitur lokal dari gambar *training* dan proyeksikan pada sumbu *x*. Urutkan fitur lokal tersebut sesuai dengan urutan kemunculannya.
- 4 3. Buat tabel untuk fitur lokal pada gambar *training* dengan setiap kolom merupakan fitur lokal yang diurutkan berdasarkan kemunculannya pada sumbu *x*;
- 5 4. Isi setiap kolom dengan pasangan-pasangan fitur lokal yang melibatkan fitur lokal kolom tersebut. Untuk tiap isi kolom, catat nomor *order* dari fitur lokal gambar *query* pasangan tersebut dan bobot (P_w) pasangan tersebut. Kolom pada tabel merupakan fitur lokal pada gambar *training* dan tiap barisnya merupakan fitur lokal pada gambar *query*.



Gambar 2.19: Tahapan verifikasi pada BSIS.

- 12 5. BSIS hanya mengambil satu pasangan dari setiap fitur lokal di gambar *query* dengan bobot pasangan yang tertinggi. Untuk itu buat sebuah *subsequence* dengan mengambil satu baris dari tiap kolom berurutan mulai dari kolom paling kiri. *Order* menunjukkan urutan kemunculan pasangan tersebut pada gambar *query*, untuk itu *order* dari *subsequence* perlu untuk terus bertambah tiap elemen. *Subsequence* yang dicari adalah yang memiliki total nilai bobot pasangan paling banyak.
- 13 6. Verifikasi juga perlu dilakukan untuk sumbu *y*. Fitur lokal yang telah dipilih pada langkah sebelumnya (fitur lokal yang masuk dalam *subsequence*) akan dilakukan verifikasi ulang menggunakan urutannya pada sumbu *y*.
- 14 21 Tahapan di atas dilakukan beberapa kali pada untuk orientasi fitur lokal gambar *training* yang berbeda untuk mencari orientasi gambar yang memberikan total skor pasangan paling tinggi.

1 2.5.3 Scoring

- 2 Setelah dipilih dan didapat pasangan fitur lokal yang konsisten secara geometris akan dihitung nilai
 3 kemiripan gambar masukkan dengan gambar *training* yang terpilih. Tingkat kemiripan ditentukan
 4 dengan menghitung total P_w dari pasangan yang terpilih pada tahap *Verification*. Nilai kemiripan
 5 ini dapat digunakan untuk menentukan apakah pasangan gambar cukup mirip untuk menjadi
 6 pasangan yang benar.

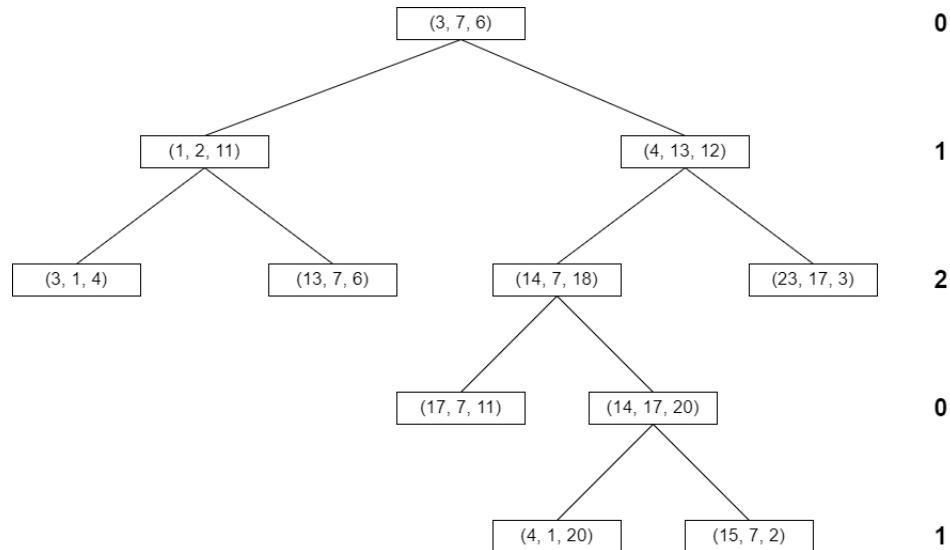
7 2.6 KD-Tree

- 8 KD-Tree [5] merupakan sebuah struktur data yang dapat digunakan untuk mencari vektor yang
 9 paling mirip dengan waktu proses yang relatif lebih singkat. Struktur KD-Tree digunakan untuk
 10 mencari sejumlah pasangan fitur lokal yang sifatnya paling mirip pada tahapan *Pairing* di BSIS 2.5.

Secara umum KD-Tree merupakan sebuah pohon pencarian biner di mana setiap *node*-nya merupakan sebuah vektor berjumlah k -elemen. Setiap *level* pada pohon membandingkan elemen vektor yang berbeda. *Level* pertama pohon akan menggunakan elemen pertama dari vektor sebagai pembanding, elemen kedua pada *level* kedua dan seterusnya. Saat *level* dari pohon sudah mencapai $k - 1$, maka akan kembali ke elemen pertama. Elemen yang diperiksa pada *level* l di KD-Tree dengan jumlah k elemen didefinisikan pada Persamaan 2.10.

$$\text{target}(l, k) = l \bmod k \quad (2.10)$$

- 11 Gambar 2.20 menunjukkan contoh pohon KD-Tree dengan jumlah 3 elemen.



Gambar 2.20: Contoh pohon struktur KD-Tree. Angka di sebelah kanan menunjukkan elemen vektor yang diperiksa pada level tersebut.

- 12 KD-Tree membuat pohon dari sekumpulan vektor di *dataset*. Vektor pertama yang masuk akan
 13 menjadi *root* dari pohon. Vektor selanjutnya akan dibandingkan nilai elemen pertamanya dengan
 14 elemen pertama dari *root*, jika nilainya lebih kecil maka vektor tersebut akan menjadi *child* sebelah
 15 kiri dan menjadi *child* sebelah kanan jika sebaliknya. Proses berlanjut untuk vektor selanjutnya,

¹ elemen di vektor yang masuk akan dibandingkan dengan vektor pada *node* pohon sesuai dengan
² *level* pohon saat itu.

³ Setelah terbentuk pohon dari semua vektor pada *dataset*, pohon tersebut dapat digunakan untuk
⁴ pencarian tetangga terdekat (*nearest neighbor search*) dari sebuah vektor masukkan baru. Tahapan
⁵ untuk mencari tetangga terdekat dari vektor *Q* pada sebuah pohon KD-Tree *K* adalah sebagai
⁶ berikut:

- ⁷ 1. Jelajahi pohon *K* seperti biasa, dimulai dari *node root*. Bandingkan elemen pada *node* tersebut
⁸ dengan elemen pada *Q*.
- ⁹ 2. Untuk semua *node* yang diperiksa hitung jaraknya ke *Q*. Simpan jarak tersebut ke dalam
¹⁰ variabel *closest*. Jika pada *node* selanjutnya didapat jarak yang lebih kecil maka ganti *closest*
¹¹ dengan jarak tersebut.
- ¹² 3. Jelajahi sampai sudah mencapai *node* yang merupakan *leaf*. Pada titik ini masih mungkin ada
¹³ *node* pada bagian pohon yang tidak dijelajahi yang jaraknya lebih kecil dari *closest*.
- ¹⁴ 4. Untuk semua *node* bukan *leaf* yang dilewati hitung jarak elemen ke-*n* dari *Q* ke elemen ke-*n*
¹⁵ *node* tersebut. Jika jaraknya lebih kecil dari *closest* maka jelajahi *child* dari *node* tersebut
¹⁶ yang sebelumnya tidak dipilih.

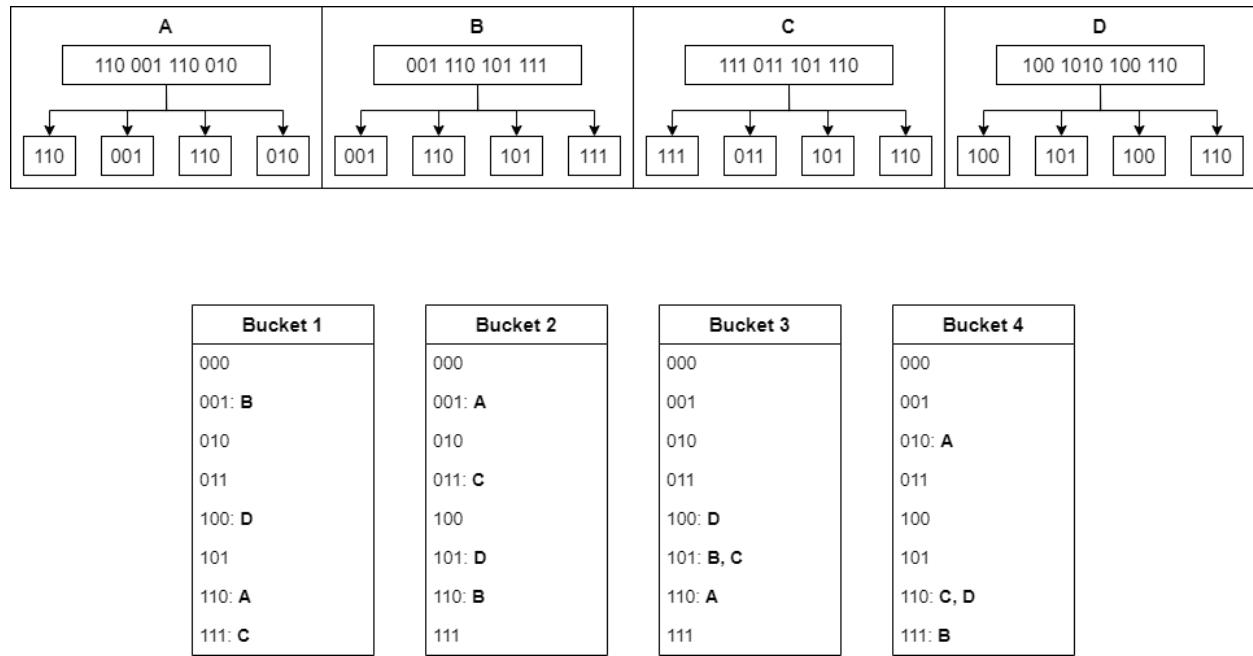
¹⁷ 2.7 Locality-Sensitive Hashing

¹⁸ *Locality Sensitive Hashing* (LSH) adalah sebuah teknik yang dapat digunakan untuk mencari tetangga
¹⁹ terdekat dari sebuah vektor biner. LSH dapat mencari tetangga terdekat dengan menggunakan
²⁰ teknik *hashing*. Teknik *Hashing* sendiri merupakan teknik untuk mengubah sebuah nilai awal
²¹ menjadi sebuah nilai *hash* yang memiliki format tertentu dengan menggunakan sebuah fungsi. Pada
²² teknik *Hashing* nilai awal yang berbeda akan menghasilkan nilai hasil *hash* yang berbeda juga.
²³ Teknik *Hashing* ini biasa digunakan untuk memberi indeks dari kumpulan data, sehingga dapat
²⁴ mempermudah pencarian suatu data tertentu.

²⁵ Teknik *Hashing* pada umumnya hanya mementingkan bahwa nilai awal yang berbeda meng-
²⁶ hasilkan nilai *hash* yang berbeda. Nilai yang awalnya mirip atau berdekatan belum tentu akan
²⁷ menghasilkan nilai *hash* yang mirip juga. Teknik LSH melakukan *hashing* dengan cara yang berbeda
²⁸ dengan teknik *hashing* pada umumnya. Pada LSH nilai awal yang mirip akan menghasilkan nilai
²⁹ hasil *hash* yang mirip juga. Sifat ini membuat nilai hasil *hash* yang memiliki dimensi lebih kecil
³⁰ dapat digunakan untuk mencari pasangan yang mirip dengan lebih efektif.

³¹ LSH dapat digunakan pada data berbentuk vektor biner (*bitstring*) yang menggunakan metrik
³² jarak Hamming dengan teknik *banding*. Teknik *banding* akan membagi *bitstring* menjadi beberapa
³³ *band* dengan panjang yang relatif sama. *Band-band* dari *bitstring* tersebut akan dibandingkan
³⁴ dengan *band-band* dari *bitstring* lain. Pada saat pencarian, *band* yang dibandingkan adalah *band*
³⁵ pada lokasi yang sama saja.

³⁶ Dalam pencarian tetangga terdekat LSH akan terlebih dahulu membuat *n* buah *bucket*, di
³⁷ mana *n* adalah jumlah *band*. Setiap *bucket* akan berfungsi sebagai indeks untuk *band* tertentu dari
³⁸ vektor-vektor di *dataset*. Sebagai contoh *bucket* pertama akan mencatat vektor mana saja yang
³⁹ *band* pertamanya merupakan sebuah *bitstring* tertentu. Ilustrasi *banding* dan *bucket* dalam LSH
⁴⁰ dapat dilihat pada Gambar 2.21.



Gambar 2.21: Ilustrasi proses *banding* dan penggunaan *bucket* sebagai indeks dalam LSH.

1 Gambar 2.21 menunjukkan 4 buah vektor *A*, *B*, *C*, dan *D* yang semuanya berada dalam sebuah
 2 *dataset*. Keempat vektor tersebut memiliki panjang 12 bit dan dilakukan proses *banding* dengan
 3 panjang *band* 3. Proses *banding* menghasilkan 4 *band* berukuran 3 bit untuk tiap vektor. Empat
 4 buah *bucket* dibuat sesuai dengan jumlah *band* dalam setiap vektor. Setiap *bucket* menyimpan set
 5 untuk setiap kombinasi *bitstring* yang sesuai dengan panjang *band*, pada kasus ini maka diperlukan
 6 sebanyak 2^3 buah set.

7 Sebuah vektor *q* yang ingin dicari tetangga terdekatnya akan terlebih dahulu dibagi menjadi
 8 *band* dengan panjang yang sama dengan vektor-vektor di *dataset*. Untuk setiap *band* akan dicari
 9 dari *bucket* yang bersesuaian vektor mana saja yang memiliki jarak $\leq j$. Vektor-vektor dengan
 10 *band* yang sama atau mirip dengan *band* vektor *q* akan ditandai sebagai kandidat tetangga terdekat.
 11 Semakin banyak *band* dari sebuah vektor yang bersesuaian dengan *band* dari vektor *q* maka semakin
 12 besar kemungkinan vektor tersebut untuk menjadi tetangga yang terdekat. Dengan menggunakan
 13 cara ini LSH dapat menyaring banyak kandidat yang memiliki kemungkinan kecil untuk menjadi
 14 tetangga terdekat, sehingga mempercepat proses pencarian.

15 2.8 Agglomerative Clustering

16 *Clustering* adalah salah satu teknik pengolahan data dalam *machine learning*. Pada dasarnya
 17 *clustering* akan membagi objek-objek pada *dataset* menjadi beberapa kelompok (*cluster*) berdasarkan
 18 sifatnya. Objek-objek yang memiliki sifat mirip akan masuk kedalam satu kelompok. Sebuah
 19 pembagian *cluster* yang baik adalah di mana setiap objek dalam *cluster* memiliki sifat yang mirip
 20 dan antar *cluster* memiliki sifat yang sangat berbeda. Salah satu contoh metode *clustering* yang
 21 ada adalah *Agglomerative Clustering*.

22 Teknik *Agglomerative Clustering* adalah salah satu teknik *clustering* yang berbasis hierarki
 23 (*hierarchical*). Teknik ini membentuk *cluster* dengan menyusun hierarki atau tingkatan antar objek

berdasarkan kemiripannya. Pasangan objek dengan jarak yang paling kecil akan berada di tingkatan terendah, jarak terkecil selanjutnya akan berada di tingkat atasnya, dan seterusnya hingga semua objek telah tercakup.

Agglomerative Clustering adalah teknik *clustering* berbasis hierarki yang menggunakan metode *bottom-up*. Metode *bottom-up* memulai tahapan dengan membentuk *cluster-cluster* kecil dan kemudian menggabungkan *cluster* kecil tersebut menjadi *cluster* yang lebih besar.

Tahapan *Agglomerative Clustering* dimulai dengan terlebih dahulu menghitung jarak antar tiap objek. Setelah itu ambil pasangan dengan jarak paling kecil dan gabungkan menjadi satu *cluster* dan lakukan juga untuk jarak paling kecil berikutnya. Jika jarak terkecil yang ada adalah antara objek yang sudah berada di dalam *cluster*, maka gabungkan kedua *cluster* tersebut menjadi *cluster* yang lebih besar.

Langkah-langkah pada tahapan tersebut dilakukan hingga semua objek sudah berada di dalam satu *cluster* yang sama. *Cluster* besar tersebut lalu dapat dibagi berdasarkan dari jaraknya dengan menggunakan *threshold* yang dapat ditentukan secara manual.

Algoritma *clustering Agglomerative* dapat dibagi menjadi beberapa tipe dibedakan dari cara menghitung jarak antar *cluster-cluster*-nya. Teknik *single-linkage* menghitung jarak antar *cluster* dengan jarak objek terdekat antar kedua *cluster* tersebut. Sedangkan teknik *complete-linkage* menghitung jarak dengan jarak objek terjauh antar dua *cluster*.

Proses penggabungan objek menjadi *cluster* pada *Agglomerative Clustering* ditunjukkan pada *Pseudocode 1*.

Pseudocode 1: Agglomerative Clustering

Input:

- D : dataset berisi n buah objek
- t : *threshold* untuk menghentikan penggabungan

Output: set berisi *cluster-cluster*

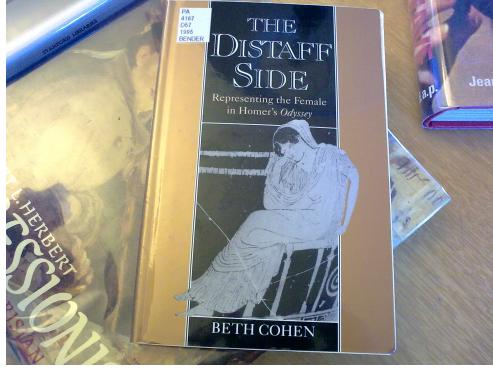
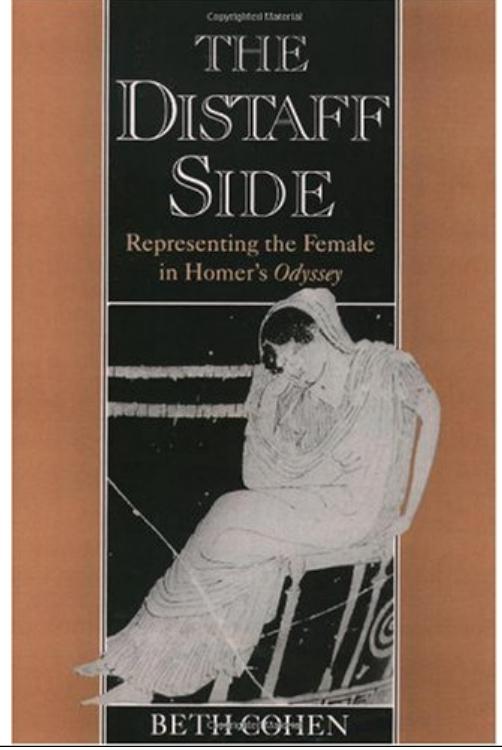
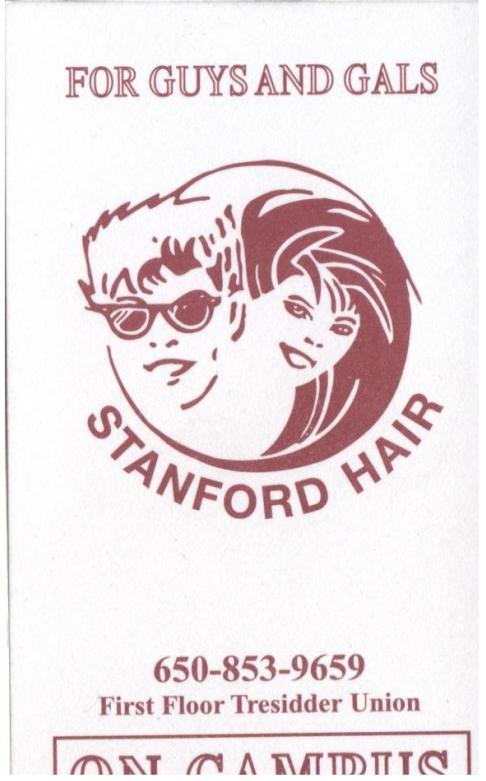
- 1: Buat semua objek dalam D menjadi *cluster* dengan 1 anggota
 - 2: Hitung jarak untuk tiap *cluster*
 - 3: **while** jumlah *cluster* dalam $D > 1$ **do**
 - 4: Ambil jarak dari *cluster* r dan s , di mana r dan s adalah dua cluster dalam D dengan jarak terkecil.
 - 5: Gabungkan r dan s menjadi satu *cluster* t .
 - 6: Hitung jarak t ke semua *cluster* lain, sesuai dengan metode *linkage* yang digunakan.
 - 7: **end while**
-

2.9 Dataset Stanford Mobile Visual Search

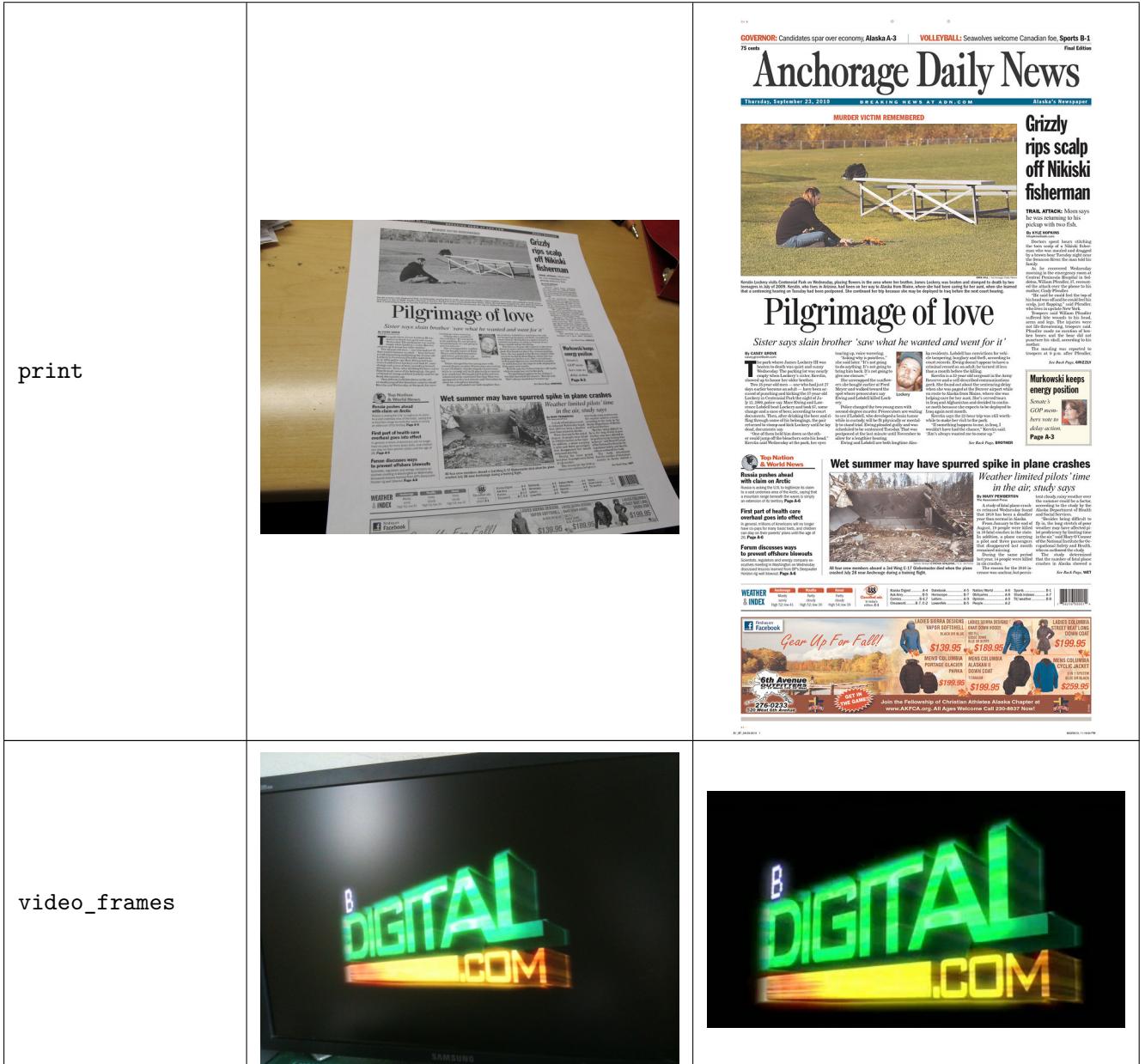
Dataset Stanford Mobile Visual Search (SMVS) merupakan *dataset* yang dapat diakses melalui website Stanford Libraries (<https://exhibits.stanford.edu/data/catalog/rb470rw0983>). *Dataset* ini terdiri dari beberapa kelompok gambar, setiap kelompok berisi gambar-gambar dari sebuah objek dengan jenis yang sama. Sebuah kelompok gambar dalam *dataset* SMVS memiliki beberapa gambar objek yang berbeda, dan untuk setiap objek terdapat sebuah gambar yang menjadi referensi. Gambar referensi berupa gambar sebuah objek yang diambil dari sudut paling jelas dan tidak ada

- 1 benda lain di latar belakang. Keterangan untuk tiap kelompok gambar dapat dilihat pada Tabel 2.1.

Tabel 2.1: Daftar kelompok gambar dalam *Dataset SMVS* beserta contoh gambar dan contoh referensinya.

Kelompok Gambar	Contoh Gambar	Contoh Referensi
book_covers		
business_cards		

cd_covers		
dvd_covers		
landmarks		
museum_paintings		



¹

BAB 3

²

ANALISIS & EKSPERIMENT

³ 3.1 Analisis Pengenalan POI

⁴ Seperti telah dijelaskan sebelumnya terdapat beberapa POI yang memiliki logo dan bagian yang
⁵ konsisten terhadap POI tersebut. POI dengan logo dan bagian konsisten tersebut dapat dikenali
⁶ dengan menggunakan fitur-fitur yang hanya didapat dari bagian tersebut. Pada tahap ini akan
⁷ dilakukan analisis untuk menguji apakah logo dan bagian yang konsisten pada sebuah POI dapat
⁸ memberikan fitur unik yang cukup kuat jika digunakan untuk melakukan pengenalan POI tersebut.

⁹ Analisis pada tahap ini merupakan analisis tahap awal yang dilakukan untuk menguji fitur-fitur
¹⁰ dari POI. Saat ini belum dilakukan *clustering* untuk memisahkan fitur lokal yang unik dan konsisten
¹¹ dengan yang tidak. Analisis juga tidak menggunakan metode OIR BSIS.

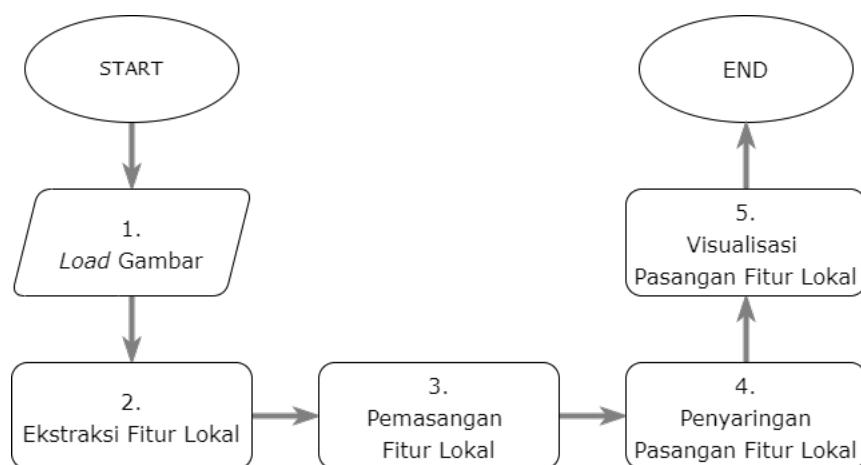
¹² 3.1.1 Ide Dasar Analisis

¹³ Analisis ini bertujuan untuk melihat bagian mana dari sebuah POI yang dapat memberikan fitur
¹⁴ yang cukup kuat jika digunakan untuk melakukan pengenalan. Untuk menguji hal tersebut akan
¹⁵ dilakukan pemasangan fitur lokal dari dua gambar POI. Kedua gambar tersebut merupakan gambar
¹⁶ dari POI yang sama diambil dari sudut pengambilan yang berbeda dan pada waktu yang berbeda.

¹⁷ Hasil pemasangan fitur lokal tersebut lalu disaring berdasarkan tingkat kekuatan pasangannya.
¹⁸ Sebuah pasangan fitur lokal q dan t , di mana t merupakan fitur lokal paling mirip dengan q
¹⁹ dikatakan kuat jika nilai kemiripan q dan t jauh lebih tinggi dibanding dengan nilai kemiripan
²⁰ q dengan fitur lokal r , di mana r merupakan fitur lokal kedua termirip dengan q . Pencarian
²¹ pasangan fitur lokal yang kuat tersebut diimplementasikan dengan melakukan *Lowe's Ratio Test*,
²² seperti yang telah dijelaskan pada [2.2](#). Setelah didapatkan pasangan-pasangan fitur lokal yang kuat,
²³ pasangan-pasangan tersebut akan ditampilkan dalam gambar untuk melihat dari objek mana dalam
²⁴ POI pasangan-pasangan tersebut berasal.

²⁵ 3.1.2 Tahapan Analisis

²⁶ Untuk melakukan implementasi dari ide di atas dilakukan langkah-langkah analisis seperti yang
²⁷ dapat dilihat pada *flowchart* di Gambar [3.1](#).



Gambar 3.1: *Flowchart* tahapan analisis pengenalan POI.

Langkah-langkah pada Gambar 3.1 secara rinci adalah sebagai berikut:

1. *Load Gambar*

Pada langkah ini digunakan dua buah gambar untuk analisis yang masing-masing diberi nama Gambar *Q* dan Gambar *T*. Gambar *Q* dan *T* merupakan gambar dari sebuah POI sama yang diambil dari sudut dan waktu pengambilan berbeda. Waktu dan sudut pengambilan berbeda tersebut menyebabkan ada objek dalam gambar yang terlihat di kedua gambar dan ada yang hanya terlihat di satu gambar saja. Kedua gambar yang digunakan tersebut dapat dilihat pada Gambar 3.2.



Gambar 3.2: Dua gambar yang digunakan untuk melakukan analisis pengenalan POI dengan logo. Gambar yang di sebelah kiri adalah Gambar *Q* dan yang di sebelah kanan adalah Gambar *T*.

2. *Ekstraksi Fitur Lokal*

Untuk masing-masing Gambar *Q* dan *T* dilakukan ekstraksi fitur lokalnya.

3. *Pemasangan Fitur Lokal*

Untuk masing-masing fitur lokal di Gambar *Q* dipasangkan dengan setiap fitur lokal Gambar *T* dan dihitung nilai kemiripannya. Dari pasangan-pasangan yang terbentuk ambil dua pasangan yang nilai kemiripannya paling tinggi.

4. *Penyaringan Pasangan Fitur Lokal*

Dari tahap sebelumnya setiap fitur lokal di Gambar *Q* memiliki dua pasangan terhadap fitur lokal Gambar *T*. Dengan menggunakan dua pasangan tersebut pilih pasangan yang cukup kuat dengan menerapkan *Lowe's Ratio Test*. Pasangan paling mirip dipilih jika nilai

1 kemiripannya lebih besar dari n kali nilai kemiripan pasangan kedua termirip.

2 5. Visualisasi Pasangan Fitur Lokal

3 Setelah didapat pasangan fitur lokal yang cukup kuat selanjutnya pasangan-pasangan tersebut
4 divisualisasikan pada gambar aslinya untuk melihat objek yang menjadi asalnya.

5 **3.1.3 Implementasi**

6 Langkah-langkah tahapan analisis yang dijelaskan pada subbab sebelumnya dilakukan dengan
7 membuat implementasi di Python. Pada penelitian ini digunakan Python versi 3.7.5 dari distribusi
8 Anaconda dan untuk pemrosesan gambar dan ekstraksi fitur serta pemasangan fitur lokal menggu-
9 nakan *library* OpenCV versi 4.5.5.64 untuk Python. Ekstraksi fitur lokal dilakukan dengan metode
10 SIFT menggunakan implementasi yang tersedia di OpenCV. Langkah-langkah implementasi yang
11 dilakukan adalah sebagai berikut:

12 1. Melakukan *import library* OpenCV (`cv2`).

13 2. Memuat kedua gambar dengan menggunakan fungsi `cv2.imread`. Masing-masing gambar akan
14 disimpan dalam bentuk *array* 2 dimensi.

15 3. Membuat objek SIFT dengan menggunakan fungsi `cv2.SIFT_create` masukkan ke sebuah
16 variabel bernama `sift`.

17 4. Menggunakan objek `sift` untuk melakukan ekstraksi fitur lokal dengan menggunakan fungsi
18 `sift.detectAndCompute` dan memasukkan *array* gambar sebagai parameter. Fungsi akan
19 mengembalikan sebuah *tuple* yang berisi objek `keypoint` dan sebuah *array* dua dimensi
20 berukuran $n \times 128$ dengan n merupakan jumlah `keypoint`. Lakukan ini untuk kedua gambar.

21 5. Membuat objek *descriptor matcher* berbasis FLANN dengan menggunakan fungsi
22 `cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED)` masukkan ke da-
23 lam objek bernama `matcher`. Objek ini berguna untuk membuat pasangan `keypoint` dengan
24 menggunakan vektor `descriptor`.

25 6. Membuat pasangan `keypoint` dengan menggunakan fungsi `matcher.knnMatch(descriptors1,`
26 `descriptors2, 2)`. Parameter `descriptors1` merupakan *descriptor* untuk Gambar Q dan
27 `descriptors2` merupakan *descriptor* untuk Gambar T sedangkan angka 2 menunjukkan
28 bahwa dicari dua pasangan yang paling mirip. Fungsi mengembalikan sebuah *list* dengan
29 setiap elemennya berisi 2 pasangan, *list* ini disimpan dengan nama `knn_matches`. Setiap
30 pasangan memiliki atribut `distance` yang menunjukkan jarak *descriptor* dari kedua `keypoint`
31 di pasangan tersebut.

32 7. Menyaring pasangan `keypoint` yang cukup kuat dengan melakukan iterasi pada *list* `knn_matches`
33 dan mengambil pasangan yang jarak pasangan terdekatnya kurang dari 0.3 kali jarak pasagan
34 kedua terdekat. Masukkan pasangan yang terpilih ke dalam *list* bernama `good_matches`.

35 8. Melakukan visualisasi dari pasangan-pasangan `keypoint` di `good_matches` dengan terlebih
36 dahulu menggambarkan pasangan `keypoint` pada gambar asal (Gambar Q dan Gambar T)
37 menggunakan fungsi `cv2.drawMatches`. Gambar asal yang sudah digambarkan pasangan
38 `keypoint`-nya tersebut lalu ditampilkan dengan menggunakan fungsi `cv2.imshow`.

1 3.1.4 Hasil Analisis

- 2 Setelah dilakukan semua tahapan implementasi pada 3.1.3 maka didapatkan hasil sebuah gambar
 3 yang menunjukkan pasangan *keypoint* yang kuat. Gambar hasil tersebut dapat dilihat pada
 4 Gambar 3.3. Gambar tersebut menunjukkan Gambar Q dengan Gambar T dengan lingkaran-
 5 lingkaran kecil serta garis merupakan pasangan *keypoint* dari kedua gambar yang telah tersaring
 6 berdasarkan kekuatan pasangannya.



Gambar 3.3: Pasangan *keypoint* dari Gambar Q dan Gambar T yang kuat.

7 Dapat dilihat dari Gambar 3.3 bahwa pasangan-pasangan *keypoint* yang kuat semua berasal
 8 dari bagian POI yang merupakan logo dan bagian konsisten. Baik logo yang berada di bagian atas
 9 POI (ditunjukkan dengan lingkaran biru) maupun logo yang berada di bagian tembok belakang
 10 (lingkaran merah) dan logo yang berada di salah satu meja (lingkaran hijau). Berdasarkan analisis
 11 yang telah dilakukan untuk saat ini dapat diambil kesimpulan bahwa bagian logo dari sebuah POI
 12 dapat memberikan fitur yang kuat untuk digunakan dalam melakukan pengenalan POI.

13 Berdasarkan pada ide bahwa logo dan bagian konsisten dari POI memberikan fitur yang kuat,
 14 pengenalan POI yang dilakukan dengan algoritma OIR seharusnya dapat ditingkatkan efektifitasnya
 15 dengan melakukan OIR hanya menggunakan fitur-fitur yang kuat tersebut. OIR dengan hanya
 16 menggunakan fitur-fitur yang kuat tersebut juga dapat diproses dengan waktu yang lebih cepat
 17 karena fitur lokal yang diproses lebih sedikit.

18 3.2 Analisis Fitur Lokal dari Logo dan Bagian yang Konsisten

19 Berdasarkan dari analisis pada 3.1 diperkirakan bahwa pengenalan POI dengan OIR dapat dipercepat
 20 dengan terlebih dahulu memilih fitur lokal yang kuat saja. Fitur-fitur lokal yang kuat tersebut
 21 biasanya dapat berupa fitur lokal yang berasal dari logo atau bagian yang konsisten dari POI. Untuk
 22 dapat memisahkan fitur lokal yang berasal dari logo dan bagian konsisten dari yang bukan perlu
 23 dilakukan analisis lebih lanjut.



Gambar 3.4: Beberapa sudut pengambilan dan waktu pengambilan berbeda dari suatu POI yang sama.

1 Dari gambar-gambar pada Gambar 3.4 terlihat bahwa sebuah POI yang diambil gambarnya dari
 2 berbagai sudut dan waktu pengambilan berbeda akan memiliki beberapa objek dalam POI yang
 3 sifatnya konsisten selalu muncul dan beberapa yang tidak. Pada contoh gambar di atas bagian yang
 4 konsisten adalah seperti logo dari POI (objek-objek dalam kotak hijau). Bagian yang tidak konsisten
 5 ditunjukkan oleh objek seperti orang-orang atau kendaraan yang lewat (objek-objek dalam kotak
 6 merah). Objek-objek yang konsisten tersebut juga seharusnya dapat memberikan fitur lokal dengan
 7 ciri yang mirip. Untuk itu perlu dicari ciri fitur lokal yang konsisten muncul di gambar-gambar
 8 POI untuk mendapatkan fitur lokal yang berasal dari logo atau bagian yang konsisten.

9 Bagian POI yang merupakan logo dan objek yang konsisten tersebut tidak selalu dapat membe-
 10 rikan fitur lokal yang baik untuk dilakukan pengenalan. Terdapat beberapa bagian yang walaupun
 11 konsisten tetapi sifatnya tidak unik terhadap POI tersebut. Seperti dapat dilihat pada Gambar 3.5,
 12 bagian yang diberi lingkaran merah merupakan bagian yang berasal dari logo kedua POI tersebut.

- 1 Walaupun berasal dari logo POI yang berbeda tetapi bentuk sudut yang ditunjukkan lingkaran
- 2 merah tersebut mirip. Sudut yang mirip tersebut akan menghasilkan fitur lokal yang mirip juga.
- 3 Fitur lokal yang berasal dari POI berbeda tetapi cirinya mirip dapat mempersulit proses pengenalan
- 4 POI. Untuk itu selain dicari fitur lokal yang konsisten perlu juga dicari fitur lokal yang sifatnya
- 5 unik terhadap sebuah POI.



Gambar 3.5: POI yang memiliki logo dengan sudut yang mirip.

- 6 Fitur lokal yang memiliki sifat konsisten dan unik tersebut dapat dicari dengan menggunakan
- 7 teknik *clustering*. Fitur lokal yang didapat dari gambar-gambar yang berbeda tersebut dibagi
- 8 menjadi beberapa kelompok yang disebut *cluster*. Dari *cluster* yang terbentuk tersebut dicari
- 9 kelompok mana saja yang sifatnya konsisten dan unik terhadap sebuah POI.

10 **3.3 Dataset yang Digunakan**

- 11 Pada analisis ini ada dua *dataset* berbeda yang digunakan, yaitu *dataset GSV* dan *dataset SMVS*.
- 12 Masing-masing *dataset* tersebut akan dijelaskan pada subbab-subbab di bawah ini.

13 **3.3.1 Dataset SMVS**

- 14 *Dataset SMVS* [2.9] memiliki beberapa kumpulan data yang masing-masing merupakan kumpulan
- 15 gambar dari benda-benda dengan jenis yang sama. Daftar kelompok data yang ada pada SMVS
- 16 adalah sebagai berikut:

- 17 • **book_covers**: berisi gambar-gambar *cover* buku.
- 18 • **bussines_cards**: berisi gambar-gambar kartu nama.
- 19 • **cd_covers**: berisi gambar-gambar sampul kaset CD.
- 20 • **dvd_covers**: berisi gambar-gambar sampul kaset DVD.
- 21 • **landmarks**: berisi gambar-gambar dari bangunan-bangunan seperti gedung, rumah, dan
- 22 lain-lain.
- 23 • **museum_paintings**: berisi gambar-gambar lukisan di museum.
- 24 • **print**: berisi gambar-gambar hasil foto dari dokumen hasil *print*.
- 25 • **video_frames**: berisi gambar-gambar yang merupakan foto dari layar yang sedang memutar
- 26 video.

- 27 Pada penelitian ini *dataset* dari SMVS yang digunakan adalah *dataset book_covers*. *Dataset*
- 28 tersebut digunakan karena gambar-gambar pada *dataset* ini cenderung mudah untuk diproses.

- 1 Data dari kelompok `book_covers` juga terbagi dengan sifat sesuai dengan yang dibutuhkan pada
 - 2 penelitian, di mana terdapat beberapa kelas gambar dan untuk setiap kelas terdapat beberapa foto
 - 3 dari benda yang sama diambil dari sudut pengambilan yang berbeda.
- 4 Pada *dataset* SMVS ini terdapat kelompok gambar `landmark` yang berisi gambar bangunan-
- 5 bangunan di sebuah lokasi tertentu yang menyerupai POI. Objek yang digunakan pada kelompok
- 6 data tersebut sesuai dengan objek yang diteliti pada penelitian ini. Kelompok gambar `landmark`
- 7 tersebut walaupun berisi objek yang sesuai, tetapi sifat data tidak sesuai dengan yang diperlukan
- 8 oleh penelitian ini. Gambar-gambar dalam kelompok gambar `landmark` tidak terbagi menjadi
- 9 kelas-kelas sesuai lokasi atau POI pada gambarnya.

10 3.3.2 Dataset GSV

11 *Dataset* GSV ini merupakan dataset berisi gambar-gambar POI. *Dataset* ini dikumpulkan dengan

12 tujuan mengumpulkan data yang sesuai dengan topik penelitian. Pada penelitian ini objek yang

13 diteliti adalah gambar dari sebuah POI, sedangkan pada *dataset* SMVS tidak terdapat data berisi

14 POI yang sesuai dengan kebutuhan penelitian.

15 Dalam *dataset* GSV terdapat beberapa kelas, di mana setiap kelas merupakan sebuah POI

16 tertentu. Setiap kelas terdiri dari gambar-gambar sebuah POI yang diambil dari sudut waktu

17 pengambilan yang berbeda. Pengambilan dari sudut dan waktu yang berbeda ditujukan agar dalam

18 gambar-gambar di sebuah kelas ada bagian dari POI yang selalu muncul dan ada yang hanya muncul

19 di satu atau beberapa gambar saja.

20 Data-data gambar dalam *dataset* GSV ini diperoleh dari dua sumber yang berbeda. Sumber

21 yang pertama merupakan Google Street View (<https://www.google.com/streetview/>). Google

22 Street View merupakan sebuah *website* yang dapat digunakan untuk melihat tampilan 360 deg

23 dari titik-titik di jalan raya. Dengan menggunakan Google Street View diambil gambar-gambar

24 POI yang dapat dilihat dari titik di jalan raya sebagai data. Sumber yang kedua adalah dengan

25 mengambil gambar secara langsung dengan menggunakan kamera. Gambar-gambar yang dihasilkan

26 baik dari Google Street View dan yang diambil menggunakan kamera memiliki ciri yang sama

27 sehingga keduanya dapat digabungkan ke dalam satu *dataset*.

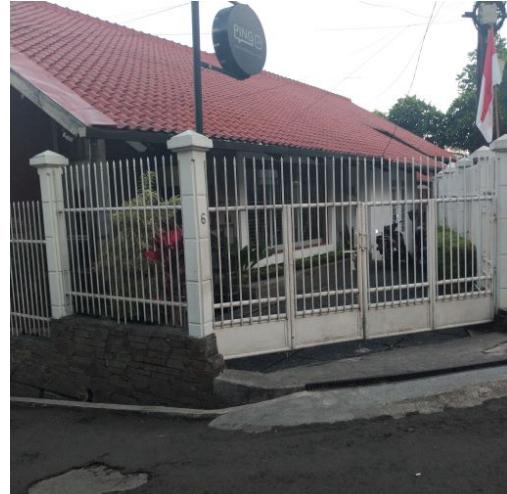
28 Keseluruhan *dataset* GSV terdiri dari sebanyak 150 gambar. Gambar-gambar tersebut terbagi

29 menjadi 10 kelas, di mana setiap kelas memiliki 15 gambar berbeda. Contoh gambar untuk tiap

30 kelas dalam *dataset* ini dapat dilihat pada Tabel 3.1.

Tabel 3.1: Daftar kelas gambar dalam *dataset GSV* beserta lokasi dan contoh gambarnya.

Nama Kelas	Lokasi POI (Koordinat)	Contoh Gambar
alfamart	Jl. Ciumbuleuit No. 42A, Hegarmanah, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.88121, 107.60377)	
binus	Jl. Ciumbuleuit No. 163, Ciumbuleuit, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87392, 107.60456)	
gembul	Jl. Ciumbuleuit No. 106, Hegarmanah, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87375, 107.60486)	

harris	Jl. Ciumbuleuit No. 50-58, Hegarmanah, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.88054, 107.60475)	 A photograph showing the exterior of a modern hotel building. The sign above the entrance reads "HARRIS HOTEL & CONVENTION". The building has a red and white facade with large glass windows. In front of the entrance, there is a small information kiosk with a red sign that also says "HARRIS". A few cars are parked on the street in front.
oxy	Jl. Ciumbuleuit No. 163, Ciumbuleuit, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87389, 107.60448)	 A photograph of a laundry shop. The building has a blue facade with a large blue double door. Above the door, there is a sign that says "OXY LAUNDRY" and "HIGIENIS". To the left of the entrance, there is a small blue wall with some graffiti. A pink trash bag is visible near the entrance.
ping	Jl. Bukit Indah I No. 6, Ciumbuleuit, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87406, 107.60375)	 A photograph of a single-story house with a red-tiled roof. The house is surrounded by a white metal fence. A satellite dish is mounted on a pole in front of the house. The house has a dark-colored entrance door and windows.

porcafe	Jl. Ciumbuleuit No. 86, Hegarmanah, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87738, 107.60426)	
starbucks	Jl. Ciumbuleuit No. 115, Hegarmanah, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87317, 107.60557)	
toyota	Jl. Ir. H. Juanda No. 131, Lb. Siliwangi, Coblong, Kota Bandung, Jawa Barat 40132 (-6.89078, 107.61305)	

yogya	Jl. Ciumbuleuit No. 147, Hegarmanah, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87573, 107.60439)	
-------	--	--

1 3.4 Analisis Sifat Konsisten pada Fitur Lokal dari Gambar POI

2 Fitur lokal yang didapat dari hasil ekstraksi ada yang memiliki sifat konsisten. Pada penelitian ini
3 fitur lokal yang konsisten terhadap sebuah POI merupakan fitur lokal dengan sifat yang muncul
4 pada mayoritas gambar berbeda dari POI tersebut. Sebuah fitur lokal yang dihasilkan dengan
5 menggunakan metode SIFT —seperti yang digunakan pada penelitian ini—akan memiliki sebuah
6 sifat yang dideskripsikan dengan sebuah vektor yang berisi nilai-nilai sudut dari *pixel-pixel* di
7 sekitarnya.

8 Pencarian fitur lokal yang konsisten dapat dilakukan dengan mengelompokkan fitur lokal
9 berdasarkan *descriptor*-nya. Pengelompokan fitur lokal ini dapat dilakukan dengan menggunakan
10 metode *Clustering*. Metode *Clustering* pada analisis ini digunakan untuk membagi fitur-fitur lokal
11 dari berbagai gambar menjadi beberapa kelompok. Setiap kelompok tersebut berisi beberapa
12 fitur lokal yang memiliki sifat yang mirip, di mana kemiripan tersebut ditunjukkan dengan jarak
13 Euclidean yang kecil antar *descriptor*-nya.

14 3.4.1 Ide Dasar dan Tahapan Analisis

15 Analisis pada tahap ini dilakukan untuk melihat apakah ada fitur lokal yang sifatnya konsisten
16 pada sekumpulan gambar POI. Analisis dilakukan dengan menggunakan 10 gambar berbeda dari
17 sebuah POI yang sama. Pada analisis ini digunakan kelas *ping* dari *dataset* GSV. Beberapa contoh
18 gambar kelas *ping* yang digunakan dapat dilihat pada Gambar 3.6.



Gambar 3.6: Beberapa contoh gambar yang digunakan pada analisis untuk melihat sifat konsisten pada fitur lokal.

1 Dari contoh-contoh gambar pada Gambar 3.6 terlihat bahwa ada bagian objek dari POI yang
 2 selalu muncul dan ada bagian yang hanya muncul di satu gambar saja. Objek-objek yang konsisten
 3 muncul tersebut akan menghasilkan fitur lokal dengan sifat yang saling mirip. Jika objek-objek
 4 dengan sifat yang mirip tersebut ditemukan dalam mayoritas gambar yang berbeda maka dapat
 5 dikatakan bahwa sifat tersebut merupakan sifat yang konsisten.

6 Pencarian fitur lokal dengan sifat konsisten pada analisis ini dilakukan dengan tahapan sebagai
 7 berikut:

- 8 1. Memuat seluruh *file* gambar yang digunakan.
- 9 2. Melakukan ekstraksi fitur pada semua gambar.
- 10 3. Menggabungkan vektor-vektor *descriptor* dari semua gambar ke dalam sebuah *dataframe*.
 Setiap vektor *descriptor* akan dicatat gambar asalnya.
- 11 4. Melakukan *clustering* pada seluruh vektor *descriptor* dari *dataframe*.
- 12 5. Menghitung jumlah gambar yang berbeda untuk tiap *cluster*.

13 Pada penelitian ini tahapan *Clustering* yang disebutkan pada poin 4 dilakukan dengan menggunakan
 14 metode Agglomerative Clustering. Metode Agglomerative Clustering dilakukan dengan menggunakan
 15 parameter *distance_threshold*. Parameter *distance_threshold* menerima sebuah nilai yang
 16 merupakan batas, di mana dua buah *cluster* tidak akan digabung jika jarak antar *cluster*-nya lebih
 17 besar dari nilai tersebut. Nilai *distance_threshold* yang digunakan didapat dari menghitung
 18 jarak rata-rata antar data dari sebanyak 100 sampel data yang diambil secara acak.

19 Langkah-langkah pada tahapan di atas akan menghasilkan kelompok-kelompok fitur lokal dengan
 20 jumlah gambar berbeda untuk tiap kelompok. Banyaknya gambar berbeda tersebut menentukan
 21 apakah fitur lokal tersebut memiliki sifat yang konsisten atau tidak.

1 3.4.2 Implementasi

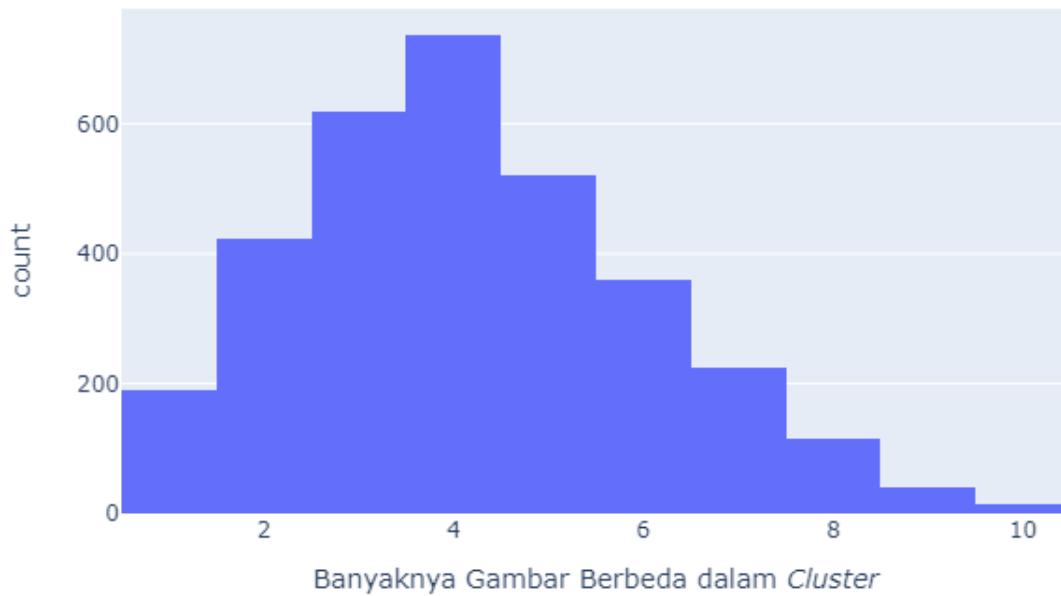
2 Analisis ini dilakukan dengan menggunakan Python versi 3.7.5 dari distribusi Anaconda. Seluruh
3 tahap pemrosesan gambar dan ekstraksi fitur lokal dilakukan dengan menggunakan fungsi-fungsi
4 dari *library* OpenCV versi 4.5.5.64. Metode *clustering* yang digunakan adalah Metode *Agglomerative*
5 *Clustering* dengan menggunakan implementasi dari *library* Scikit-learn versi 1.0.2. Tahap pemrosesan
6 dan tabulasi data dilakukan dengan menggunakan *library* Pandas versi 1.3.5, serta visualisasi data
7 dihasilkan dengan menggunakan *library* Plotly. Tahapan implementasi yang dilakukan adalah
8 sebagai berikut:

- 9 1. Melakukan *Import* semua *library* yang diperlukan: OpenCV (`cv2`), Scikit-learn (`sklearn`),
10 Pandas (`pandas`), dan Plotly (`px`)
- 11 2. Memuat semua gambar yang akan digunakan dengan fungsi `cv2.imread`. Setiap gambar
12 diperkecil ukurannya sehingga tidak ada sisi yang panjangnya lebih dari 600 *pixel*. Format
13 warna gambar juga diubah menjadi *grayscale*.
- 14 3. Membuat objek SIFT yang akan digunakan untuk melakukan ekstraksi fitur lokal. Pembuatan
15 objek SIFT dengan menggunakan fungsi `cv2.SIFT_create`, objek SIFT tersebut disimpan
16 dalam variabel dengan nama `sift`.
- 17 4. Menggunakan objek `sift` untuk melakukan ekstraksi fitur lokal dari masing-masing gambar
18 yang telah dimuat sebelumnya. Ekstraksi fitur lokal dilakukan dengan menggunakan fungsi
19 `sift.detectAndCompute`. Fungsi `sift.detectAndCompute` menghasilkan sebuah *array* yang
20 berisi objek *keypoint* dan sebuah *array* yang berisi vektor *descriptor* untuk tiap *keypoint*.
- 21 5. Menggabungkan *array descriptor* dari masing-masing gambar ke dalam satu *dataframe*.
- 22 6. Menambahkan kolom `img` ke dalam *dataframe descriptor*. Kolom `img` tersebut berisi nama
23 gambar asal dari sebuah *descriptor*.
- 24 7. Melakukan *clustering* pada vektor *descriptor* dari *dataframe*. *Clustering* dilakukan dengan
25 membuat objek *AgglomerativeClustering* dari *sklearn*.
- 26 8. Setelah didapat label *cluster* untuk tiap *descriptor*, masukkan label tersebut ke dalam *dataframe*
27 sebagai kolom baru `cluster_label`.
- 28 9. Mengelompokkan data pada *dataframe* berdasarkan kolom `cluster_label` dengan menggu-
29 nakan fungsi `groupby` yang tersedia dari Pandas.
- 30 10. Menghitung jumlah gambar (kolom `img`) yang unik dari setiap kelompok `cluster_label`
31 dengan menggunakan fungsi `nunique`.
- 32 11. Menampilkan sebaran jumlah gambar yang unik pada sebuah histogram. Histogram dibuat
33 dengan menggunakan memanggil fungsi `px.histogram` dan memasukkan data yang sesuai.

34 3.4.3 Hasil

- 35 Dari implementasi program yang telah dilakukan didapat hasil berupa nomor *cluster* beserta jumlah
36 gambar berbeda dari *cluster* tersebut. Sebaran untuk jumlah gambar unik untuk setiap *cluster*
37 dapat dilihat pada histogram di Gambar 3.7.

Sebaran Jumlah Gambar Unik



Gambar 3.7: Histogram sebaran jumlah gambar unik tiap *cluster*.

1 Dari histogram pada Gambar 3.7 terlihat bahwa ada *cluster* yang berisi fitur-fitur lokal yang
2 sifatnya konsisten ditandai dengan nilai jumlah gambar unik yang tinggi dalam *cluster* tersebut.
3 Sebagai contoh sebuah *cluster* yang memiliki sebanyak 8 gambar unik di dalamnya berarti *cluster*
4 tersebut berisi fitur-fitur lokal dengan sifat yang muncul di 8 gambar berbeda.

5 Sebaran pada histogram menunjukkan bahwa mayoritas fitur lokal dari gambar merupakan fitur
6 lokal yang sifatnya tidak konsisten. Terlihat dari sebagian besar data berada pada rentang nilai
7 gambar unik ≤ 5 yang terbilang cukup rendah. Hanya sebagian kecil fitur lokal dari gambar-gambar
8 di kelas ping yang memiliki fitur lokal dengan sifat yang konsisten terhadap POI.

9 Beberapa contoh *keypoint* yang fitur lokalnya memiliki sifat konsisten berdasarkan analisis ini
10 dapat dilihat pada Gambar 3.8. *Keypoint* yang ditampilkan adalah yang fitur lokalnya berasal dari
11 *cluster* di mana dalam *cluster* tersebut ada 9 atau lebih gambar berbeda.



Gambar 3.8: Contoh *keypoint* yang konsisten menurut analisis ini.

3.5 Analisis Sifat Unik pada Fitur Lokal dari Gambar POI

Fitur lokal yang didapat dari hasil ekstraksi pada gambar POI juga memiliki sifat yang unik terhadap sebuah POI. Pada penelitian ini sebuah fitur lokal dikatakan unik terhadap sebuah POI jika fitur lokal tersebut memiliki sifat yang hanya muncul di POI tersebut saja. Sebuah sifat fitur lokal tidak perlu benar-benar hanya muncul pada satu POI saja untuk dikatakan unik, jika sebuah sifat muncul pada beberapa gambar berbeda dan mayoritas dari gambar berbeda tersebut merupakan gambar dari POI yang sama maka sifat fitur lokal tersebut juga bisa dikatakan unik. Analisis kali ini mirip dengan analisis sebelumnya yaitu dilakukan dengan melakukan *clustering* pada vektor *descriptor* dari fitur lokal yang dihasilkan dengan metode SIFT.

3.5.1 Ide Dasar dan Tahapan Analisis

Analisis dilakukan untuk melihat apakah ada fitur lokal yang sifatnya unik terhadap sebuah POI atau tidak. Analisis akan dilakukan dengan menggunakan gambar-gambar dari beberapa POI yang berbeda, di mana setiap POI akan memiliki lebih dari satu gambar. Pada analisis ini digunakan sebanyak 5 kelas dari *dataset* GSV, yaitu **alfamart**, **binus**, **gembul**, **harris**, dan **oxy**. Dari setiap kelas gambar hanya diambil sebanyak 4 gambar saja. Contoh gambar-gambar yang digunakan pada analisis ini dapat dilihat pada Gambar 3.9.



Gambar 3.9: Beberapa contoh gambar yang digunakan pada analisis untuk melihat sifat unik pada gambar POI.

1 Gambar-gambar tersebut diekstrak fitur lokalnya dan vektor-vektor *descriptor* dari gambar-
2 gambar berbeda akan dikumpulkan menjadi satu. Kumpulan vektor *descriptor* tersebut lalu dibagi
3 menjadi beberapa kelompok dengan menggunakan metode *clustering*. Hasil kelompok-kelompok
4 dari *clustering* tersebut lalu akan dianalisis untuk melihat apakah fitur-fitur lokal dalam kelompok
5 tersebut memiliki sifat yang unik atau tidak.

6 Sebuah *cluster* dengan anggota beberapa fitur lokal merupakan sebuah kelompok fitur lokal
7 yang memiliki sebuah sifat yang sama. Sifat fitur-fitur lokal tersebut dapat dikatakan unik terhadap
8 sebuah POI jika mayoritas fitur lokal dalam *cluster* tersebut berasal dari gambar sebuah POI yang
9 sama. Sebaliknya jika ada sebuah *cluster* yang memiliki anggota fitur lokal yang berasal dari banyak
10 gambar berbeda maka fitur lokal dalam *cluster* tersebut memiliki sifat yang tidak unik.

11 Pencarian fitur lokal dengan sifat unik pada analisis ini dilakukan dengan tahapan sebagai
12 berikut:

- 13 1. Memuat file gambar yang digunakan
 - 14 2. Melakukan ekstraksi fitur pada semua gambar
 - 15 3. Menggabungkan vektor-vektor *descriptor* dari semua gambar ke dalam sebuah *dataframe*.
Setiap vektor *descriptor* akan dicatat gambar asalnya serta kelas gambarnya (nama POI)
 - 16 4. Melakukan *clustering* pada seluruh vektor *descriptor* dari *dataframe*. *Clustering* dilakukan
menggunakan cara yang sama dengan *clustering* pada
 - 17 5. Menghitung banyak kelas gambar berbeda yang muncul dalam setiap *cluster*. **3.4**.
- 20 Setelah melakukan semua tahapan di atas akan didapat *cluster-cluster* fitur lokal beserta dengan
21 jumlah kelas gambar yang berbeda untuk tiap *cluster*-nya. Jumlah kelas gambar berbeda tersebut
22 akan menentukan apakah fitur lokal dalam *cluster* memiliki sifat yang unik atau tidak.

¹ 3.5.2 Implementasi

² Analisis ini dilakukan dengan menggunakan Python versi 3.7.5 dari distribusi Anaconda. Seluruh
³ tahap pemrosesan gambar dan ekstraksi fitur lokal dilakukan dengan menggunakan fungsi-fungsi
⁴ dari *library* OpenCV versi 4.5.5.64. Metode *clustering* yang digunakan adalah Metode *Agglomerative*
⁵ *Clustering* dengan menggunakan implementasi dari *library* Scikit-learn versi 1.0.2. Tahap pemrosesan
⁶ dan tabulasi data dilakukan dengan menggunakan *library* Pandas versi 1.3.5, serta untuk visualisasi
⁷ data dihasilkan dengan menggunakan *library* Plotly. Tahapan implementasi yang dilakukan adalah
⁸ sebagai berikut:

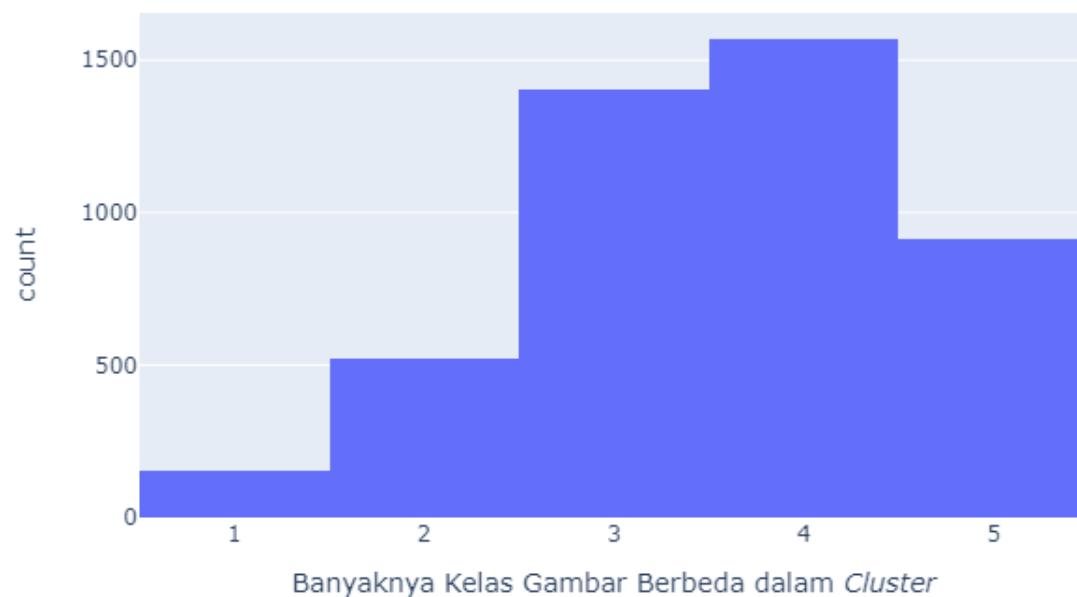
- ⁹ 1. Melakukan *Import* semua *library* yang diperlukan: OpenCV (`cv2`), Scikit-learn (`sklearn`),
¹⁰ Pandas (`pandas`), dan Plotly (`px`)
- ¹¹ 2. Memuat semua gambar yang digunakan. Gambar-gambar dimasukkan ke dalam *dictionary*
¹² dengan nama kelas gambar (nama POI) sebagai *key* dan *value*-nya berupa *list* yang menyimpan
¹³ nama gambar dan *file* gambar.
- ¹⁴ 3. Membuat objek SIFT yang akan digunakan untuk melakukan ekstraksi fitur lokal. Pembuatan
¹⁵ objek SIFT dengan menggunakan fungsi `cv2.SIFT_create`, objek SIFT tersebut disimpan
¹⁶ dalam variabel dengan nama `sift`.
- ¹⁷ 4. Menggunakan objek `sift` untuk melakukan ekstraksi fitur lokal dari masing-masing gambar
¹⁸ yang telah dimuat sebelumnya. Ekstraksi fitur lokal dilakukan dengan menggunakan fungsi
¹⁹ `sift.detectAndCompute`, fungsi tersebut akan menghasilkan sebuah *array* yang berisi objek
²⁰ *keypoint* dan sebuah *array* yang berisi vektor *descriptor* untuk tiap *keypoint*.
- ²¹ 5. Menggabungkan setiap *array descriptor* dari masing-masing gambar ke dalam satu *dataframe*.
- ²² 6. Menambahkan kolom `img` ke dalam *dataframe descriptor*. Kolom `img` tersebut berisi nama
²³ gambar asal dari sebuah *descriptor*.
- ²⁴ 7. Menambahkan kolom `img_class` ke dalam *dataframe descriptor*. Kolom `img_class` tersebut
²⁵ berisi kelas gambar asal dari sebuah *descriptor*.
- ²⁶ 8. Melakukan *clustering* pada vektor *descriptor* dari *dataframe*. *Clustering* dilakukan dengan
²⁷ membuat objek *AgglomerativeClustering* dari `sklearn`.
- ²⁸ 9. Setelah didapat label *cluster* untuk tiap *descriptor*, masukkan label tersebut ke dalam *dataframe*
²⁹ sebagai kolom baru `cluster_label`.
- ³⁰ 10. Mengelompokkan data pada *dataframe* berdasarkan kolom `cluster_label` dengan menggu-
³¹ nakan fungsi `groupby` yang tersedia dari Pandas.
- ³² 11. Menghitung jumlah kelas gambar (kolom `img_class`) yang unik untuk tiap `cluster_label`
³³ dengan menggunakan fungsi `nunique`.
- ³⁴ 12. Menampilkan sebaran jumlah kelas gambar yang unik pada sebuah histogram. Histogram
³⁵ dibuat dengan menggunakan memanggil fungsi `px.histogram` dan memasukkan data yang
³⁶ sesuai.

³⁷ 3.5.3 Hasil

- ³⁸ Setelah selesai dilakukan semua tahapan implementasi akan didapat kelompok-kelompok fitur lokal
³⁹ serta jumlah kelas gambar yang unik untuk setiap kelompok. Banyaknya kelas gambar yang unik
⁴⁰ tersebut menentukan apakah sifat fitur-fitur lokal dalam kelompok tersebut termasuk unik atau
⁴¹ tidak. Jumlah kelas gambar unik yang tinggi menunjukkan bahwa sebuah kelompok memiliki sifat

- 1 fitur lokal yang tidak unik, karena jumlah kelas gambar unik yang tinggi berarti sifat fitur lokal
- 2 pada gambar tersebut muncul di banyak jenis POI yang berbeda dan tidak unik terhadap satu POI
- 3 saja. Sebaran nilai jumlah kelas gambar yang unik dapat dilihat pada histogram di Gambar 3.10.

Sebaran Jumlah Kelas Gambar yang Berbeda



Gambar 3.10: Histogram sebaran jumlah kelas gambar yang unik tiap *cluster*.

- 4 Terlihat dari histogram pada Gambar 3.10 bahwa ada *cluster* dengan jumlah kelas gambar unik
- 5 yang sedikit seperti 1 dan 2, di mana merupakan *cluster* tersebut merupakan *cluster* dengan sifat
- 6 yang unik. Jumlah *cluster* dengan sifat yang unik termasuk sedikit dibandingkan dengan jumlah
- 7 *cluster* yang sifatnya tidak unik. Sedikitnya jumlah *cluster* dengan sifat yang unik menunjukkan
- 8 bahwa dalam sebuah gambar POI sebagian besar fitur lokal merupakan fitur lokal yang sifatnya
- 9 tidak unik.
- 10 Beberapa contoh *keypoint* yang fitur lokalnya memiliki sifat unik berdasarkan analisis ini dapat
- 11 dilihat pada Gambar 3.11. *Keypoint* yang ditampilkan adalah yang berasal dari *cluster* di mana
- 12 dalam *cluster* tersebut hanya ada maksimum sebanyak 2 kelas gambar yang berbeda.



Gambar 3.11: Contoh *keypoint* yang unik menurut analisis ini.

3.6 Analisis Tingkat Keunikan dan Kekonsistensi Fitur Lokal pada Gambar POI

Berdasarkan hasil yang didapat dari analisis di 3.4 dan 3.5 didapatkan bahwa ada fitur lokal yang sifatnya konsisten, unik, dan ada yang tidak. Tetapi dari kedua analisis tersebut belum didapat didapat nilai yang dapat digunakan untuk mengukur tingkat kekonsistensi dan nilai yang dapat digunakan untuk mengukur tingkat keunikan fitur lokal. Analisis pada bagian ini bertujuan untuk membuat sebuah metode yang dapat memberikan nilai konsistensi dan nilai keunikan pada suatu fitur lokal. Nilai konsistensi dan nilai keunikan ini nantinya akan berguna untuk menyaring fitur lokal, sehingga dapat dengan mudah diambil fitur lokal yang sifatnya unik dan sifatnya konsisten saja.

Analisis yang dilakukan merupakan penggabungan dari analisis di 3.4 dan 3.5, dengan beberapa modifikasi untuk mengatasi masalah yang ditemui. Nilai konsistensi dan nilai keunikan akan dihitung dari hasil *clustering* yang sama. Berbeda dengan sebelumnya di mana proses *clustering* untuk melihat konsistensi dan keunikan dilakukan dengan format *dataset* yang berbeda.

3.6.1 Ide Dasar Analisis

Analisis dilakukan dengan menggunakan gambar-gambar dari *Dataset GSV* (3.3.2). Gambar-gambar dalam *dataset* tersebut diekstrak fitur lokalnya, dan setiap fitur lokal dicatat asal gambarnya serta kelas dari gambar asalnya. Fitur-fitur lokal dari gambar tersebut lalu dibagi menjadi kelompok-kelompok dengan menggunakan teknik *clustering*.

Terdapat masalah jika ada pola tertentu pada POI yang berulang, pola berulang tersebut akan menghasilkan banyak fitur lokal dengan ciri yang mirip. Banyaknya fitur lokal dengan ciri yang sama tersebut diatasi dengan terlebih dahulu melakukan *clustering* pada masing-masing gambar. Tahap *clustering* yang dilakukan akan dibagi menjadi 2 bagian, tahap pertama merupakan *clustering* per

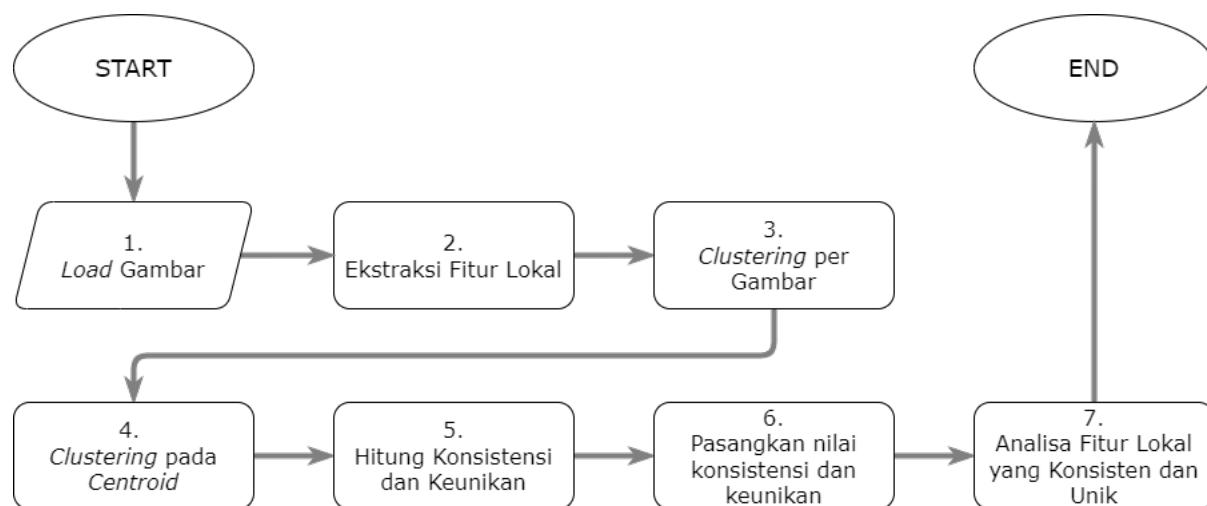
1 gambar dan tahap kedua merupakan *clustering* keseluruhan gambar. *Clustering* yang dilakukan pada
 2 masing-masing gambar akan menghasilkan *centroid* dari masing-masing *cluster*. *Centroid-centroid*
 3 ini nantinya digabungkan dengan *centroid-centroid* dari gambar lain untuk dilakukan *clustering*
 4 tahap kedua. *Cluster-cluster* yang terbentuk dari *clustering* tahap kedua ini digunakan untuk
 5 menentukan nilai konsistensi dan keunikan fitur lokal.

6 Seperti yang telah dijelaskan sebelumnya pada 3.4 fitur lokal bersifat konsisten terhadap sebuah
 7 POI jika fitur lokal tersebut muncul di mayoritas gambar-gambar POI tersebut. Sebuah fitur
 8 lokal akan bersifat semakin konsisten dengan semakin banyaknya gambar di mana ia muncul.
 9 Kekonsistennan ini dihitung dengan melihat fitur-fitur lokal dalam sebuah *cluster* jika dalam *cluster*
 10 tersebut terdapat fitur lokal dari kelas A dan terdapat beberapa fitur lokal lain dari kelas yang sama
 11 tetapi dari gambar yang berbeda maka fitur lokal dengan kelas A di *cluster* tersebut merupakan
 12 fitur lokal yang konsisten. Tingkat kekonsistennan sebuah fitur lokal akan ditunjukkan oleh sebuah
 13 nilai yang didapat dari menghitung jumlah gambar yang berbeda dari kelas POI yang sama dalam
 14 sebuah *cluster* dan membaginya dengan jumlah gambar POI tersebut dalam *dataset*. Semakin tinggi
 15 nilai tersebut maka semakin konsisten sifat fitur lokalnya.

16 Seperti telah dijelaskan pada 3.5 fitur lokal bersifat unik jika fitur lokal tersebut hanya muncul
 17 di satu atau sedikit POI saja. Semakin sedikit POI di mana fitur lokal dengan sifat tersebut
 18 muncul maka semakin unik sifat fitur lokalnya. Tingkat keunikan sebuah fitur lokal dihitung dengan
 19 melihat ada berapa kelas gambar berbeda dalam *cluster* yang sama dengan fitur lokal tersebut.
 20 Jika sebagian besar fitur lokal berasal dari kelas gambar A maka fitur lokal dengan kelas gambar A
 21 di *cluster* tersebut merupakan fitur lokal yang unik. Nilai keunikan didapat dari terlebih dahulu
 22 menghapus fitur lokal yang gambarnya memiliki duplikat dan lalu untuk tiap kelas gambar dihitung
 23 jumlahnya dan dibagi dengan jumlah anggota *cluster* tersebut (setelah dihapus yang duplikat).
 24 Semakin tinggi nilai keunikan ini maka semakin unik fitur lokal tersebut.

25 3.6.2 Tahapan Analisis

26 Langkah-langkah analisis yang dilakukan untuk mencari fitur lokal dengan sifat konsisten dan unik
 27 dapat dilihat pada *flowchart* di Gambar 3.12



Gambar 3.12: *Flowchart* tahapan analisis *clustering* untuk mencari fitur lokal yang konsisten dan unik.

Langkah-langkah pada *flowchart* di Gambar 3.12 secara rinci adalah sebagai berikut:

1. Load Gambar

Pada analisis ini digunakan sebanyak total 100 gambar yang terbagi menjadi 10 kelas POI. Setiap kelas POI memiliki 10 gambar yang merupakan gambar POI tersebut dengan sudut pengambilan dan waktu pengambilan yang berbeda. Beberapa contoh gambar yang digunakan dapat dilihat pada Gambar 3.13.



Gambar 3.13: Contoh beberapa gambar yang digunakan pada analisis ini.

2. Ekstraksi Fitur Lokal

Lakukan ekstraksi fitur lokal dari semua gambar yang digunakan. Fitur lokal yang diekstraksi dikelompokkan berdasarkan gambar asalnya.

3. Clustering per Gambar

Setiap kelompok fitur lokal yang telah didapat dari tahap sebelumnya digunakan untuk melakukan *clustering*. Setelah didapat *cluster-cluster* lalu dihitung *centroid* untuk setiap *cluster*. *Centroid* yang telah didapat ini lalu dimasukkan kedalam satu kelompok yang sama untuk semua gambar.

4. Clustering pada *Centroid*

Setelah didapat *centroid-centroid* dari hasil *clustering* tiap gambar, dilakukan *clustering*

1 pada *centroid-centroid* tersebut. Dari *cluster-cluster* yang terbentuk dianalisis pada tahapan
2 berikutnya.

3 5. Hitung Konsistensi dan Keunikan

4 Untuk setiap *cluster* dari hasil *clustering* pada *centroid* dihitung nilai konsistensi dan keuni-
5 kannya dengan menggunakan ide yang telah dijelaskan pada 3.6.1. Konsistensi dan keunikan
6 akan ditunjukkan dengan sebuah angka pada setiap *centroid* dalam *cluster*.

7 6. Pasangkan Nilai Konsistensi dan Keunikan

8 Pada langkah sebelumnya didapatkan nilai konsistensi dan keunikan untuk setiap *centroid*
9 hasil *clustering* fitur lokal per gambar. Nilai konsistensi dan keunikan ini lalu dipasangkan ke
10 fitur lokal asalnya sesuai dengan nomor *cluster* dari *clustering* tahap pertama.

11 7. Analisa Fitur Lokal yang Konsisten dan Unik

12 Setelah didapatkan nilai konsistensi dan keunikan untuk tiap fitur lokal maka akan dilakukan
13 analisis dengan cara melakukan visualisasi fitur lokal yang memiliki nilai konsistensi dan
14 keunikan yang tinggi untuk melihat asal fitur lokal tersebut.

15 **3.6.3 Implementasi**

16 Langkah-langkah yang telah dijelaskan pada 3.6.2 dijalankan dengan membuat implementasi
17 menggunakan Python. Versi Python yang digunakan adalah Python versi 3.7.5 dari distribusi
18 Anaconda. Pemrosesan gambar serta ekstraksi fitur lokal dilakukan dengan menggunakan *library*
19 OpenCV versi 4.5.5.64. Untuk *clustering* digunakan metode *Agglomerative Clustering* dari *library*
20 Scikit-learn versi 1.0.2. Serta digunakan juga *library* Pandas versi 1.3.5 untuk pemrosesan data.
21 Langkah-langkah implementasi yang dilakukan adalah sebagai berikut:

- 22 1. Melakukan *import library* OpenCV (`cv2`), Scikit-learn (`sklearn`), dan Pandas (`pandas`).
- 23 2. Memuat semua gambar yang digunakan dengan fungsi `cv2.imread`. Gambar yang dimuat
terlebih dahulu diubah ukurannya agar menjadi tidak ada sisi yang panjangnya lebih dari 600
pixel. *Array* gambar disimpan dalam sebuah *dictionary* dengan nama gambar sebagai *key*.
- 24 3. Membuat objek SIFT dengan fungsi `cv2.SIFT_create` masukkan ke dalam objek dengan
nama `sift`.
- 25 4. Menggunakan objek `sift` untuk melakukan ekstraksi fitur lokal dari masing-masing gambar.
Ekstraksi dilakukan dengan fungsi `sift.detectAndCompute`. *Keypoint* yang dihasilkan dima-
sukkan ke dalam sebuah *list* bernama `keypoints` untuk semua gambar. Untuk *descriptor*
dimasukkan ke dalam sebuah *dataframe* dengan nama `descriptors`.
- 26 5. Pada *dataframe* `descriptor` ditambahkan kolom `img` yang menyimpan nama gambar asal,
`img_class` untuk kelas dari gambar asal, dan `kp_idx` yang menunjukkan posisi *keypoint* di
list `keypoints`.
- 27 6. Mengelompokkan *dataframe* `descriptors` berdasarkan kolom `img` dengan menggunakan fungsi
`groupby`.
- 28 7. Melakukan iterasi untuk setiap *group*. Dalam setiap iterasi lakukan *clustering* pada *descriptor*
dengan mengambil 128 kolom pertama dari *dataframe* `descriptors`. *Clustering* dilakukan
dengan menggunakan cara yang sama seperti pada 3.4.
- 29 8. Menghitung *centroid* dari tiap *cluster*. Menyimpan hasil *centroid* ke dalam sebuah *dataframe*
bernama `df_centroid`. Juga disimpan nomor *cluster* (`cluster_label`), nama gambar (`img`),

- 1 dan kelas gambar (`img_class`).
- 2 9. Melakukan *clustering* pada *dataframe* `df_centroid`. *Clustering* dilakukan dengan cara yang
3 sama seperti sebelumnya. Nomor *cluster* dimasukkan ke dalam `df_centroid` sebagai kolom
4 dengan nama `cluster2_label`.
- 5 10. Membuat sebuah *dictionary* bernama `cluster2_class_count` yang nantinya digunakan untuk
6 menyimpan nilai keunikan.
- 7 11. Mengelompokkan data pada `df_centroid` berdasarkan kolom `cluster2_label`.
Untuk setiap kelompok pertama hanya ambil satu baris dari tiap gambar
9 (`drop_duplicates(subset='img')`). Setelah itu hitung persentase banyaknya tiap kelas
10 gambar dengan fungsi `value_counts(normalize=True)` pada kolom `img_class`. Masukkan
11 hasilnya ke dalam `cluster2_class_count`. *Dictionary* `cluster2_class_count` akan memiliki
12 nomor dari `cluster2_label` sebagai *key* dan *value*-nya akan berisi sebuah *series* dengan *key*
13 merupakan kelas gambar dan isinya persentase kemunculan kelas gambar tersebut.
- 14 12. Membuat sebuah *list* dengan nama `uniqueness`. Isi *list* tersebut dengan melakukan iterasi pada
15 kolom `cluster2_label` dan `img_class` dan ambil nilai persentase keunikan yang bersesuaian
16 dari `cluster2_class_count`.
- 17 13. Memasukkan *list* `uniqueness` sebagai kolom pada `df_centroid`.
- 18 14. Mengelompokkan `df_centroid` berdasarkan `cluster2_label` dan `img_class` dengan menggunakan
19 fungsi `groupby`. Untuk setiap kelompok dihitung jumlah gambar yang berbeda. Hasil
20 pengelompokan dimasukkan ke dalam variabel bernama `cluster2_img_class_group`.
- 21 15. Membuat sebuah *list* dengan nama `img_count`. *List* ini akan berisi nilai konsistensi untuk
22 tiap *centroid* di `df_centroid`.
- 23 16. Melakukan iterasi pada kolom `cluster2_label` dan `img_class` dari `df_centroid`. Untuk
24 setiap iterasi ambil nilai yang bersesuaian dari `cluster2_img_class_group`. Nilai tersebut
25 dibagi dengan 10 (jumlah gambar tiap kelas) dan dimasukkan ke dalam *list* `img_count`.
- 26 17. Masukkan `img_count` ke sebagai kolom di `df_centroid` dengan nama `consistency`.
- 27 18. Mengambil kolom `cluster_label`, `cluster2_label`, `uniqueness`, dan `consistency` dari
28 `df_centroid`. Lakukan `merge` pada kolom-kolom tersebut dengan *dataframe* `descriptors`
29 dengan bertumpu pada kolom `cluster_label`.

30 **3.6.4 Hasil Analisis**

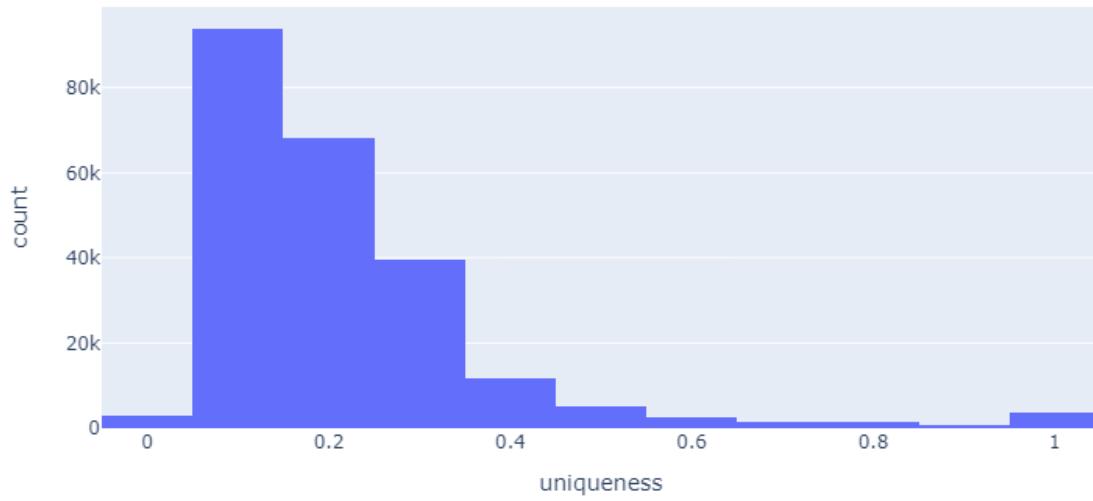
- 31 Setelah dilakukan implementasi sesuai dengan tahapan pada 3.6.3 didapatkan tabel berisi *descriptor*
32 untuk *keypoint* beserta dengan nilai keunikan (`uniqueness`) dan konsistensinya (`consistency`).
33 Beberapa contoh potongan dari tabel hasil dapat dilihat pada Tabel 3.2

img	img_class	cluster_label	cluster2_label	uniqueness	consistency
...
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_7.jpg	yogya	33	3639	0.4	0.4
yogya_9.jpg	yogya	238	3639	0.4	0.4
yogya_9.jpg	yogya	238	3639	0.4	0.4
yogya_9.jpg	yogya	238	3639	0.4	0.4
yogya_9.jpg	yogya	238	3639	0.4	0.4
gembul_6.jpg	gembul	330	3640	0.666667	0.2
gembul_6.jpg	gembul	330	3640	0.666667	0.2
gembul_6.jpg	gembul	330	3640	0.666667	0.2
gembul_8.jpg	gembul	313	3640	0.666667	0.2
gembul_8.jpg	gembul	313	3640	0.666667	0.2
oxy_10.jpg	oxy	29	3640	0.333333	0.1
oxy_10.jpg	oxy	29	3640	0.333333	0.1
oxy_10.jpg	oxy	29	3640	0.333333	0.1
oxy_10.jpg	oxy	29	3640	0.333333	0.1

Tabel 3.2: Potongan beberapa contoh dari tabel *descriptor*.

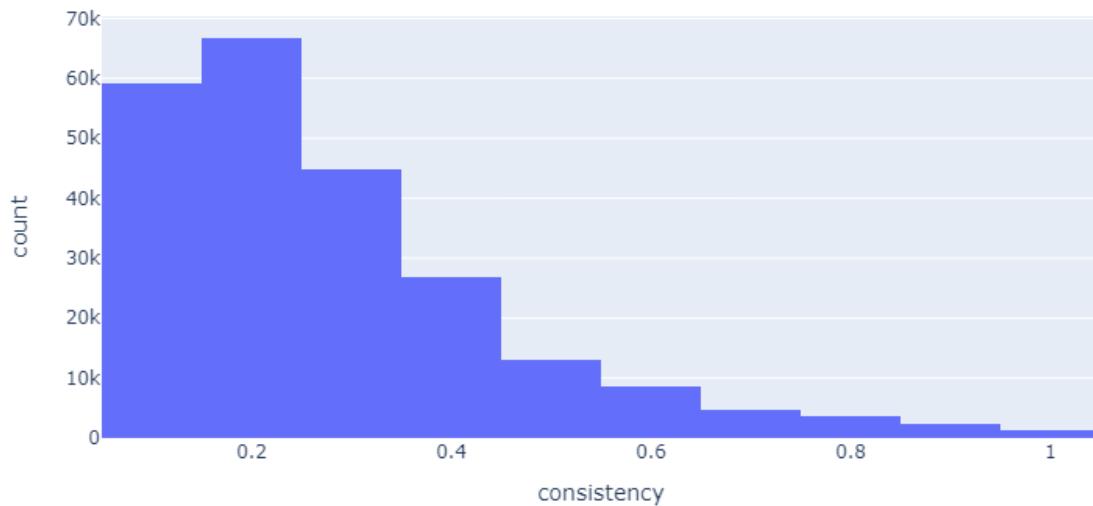
- 1 Nilai-nilai pada kolom **uniqueness** dan **consistency** didapat dari melakukan penghitungan
 2 sebaran anggota tiap *cluster* seperti yang telah dijelaskan di subbab-subbab sebelumnya.
 3 Data pada Tabel 3.2 menunjukkan fitur lokal dari *keypoint* beserta nilai konsistensi (**consistency**)
 4 dan keunikannya (**uniqueness**). Dari tabel tersebut terlihat bahwa nilai keduanya cukup beragam.
 5 Sebaran untuk nilai keunikan dan konsistensi dapat dilihat pada Gambar 3.14 dan Gambar 3.15.

Histogram Sebaran Nilai ***uniqueness***



Gambar 3.14: Histogram sebaran nilai keunikan.

Histogram Sebaran Nilai ***consistency***



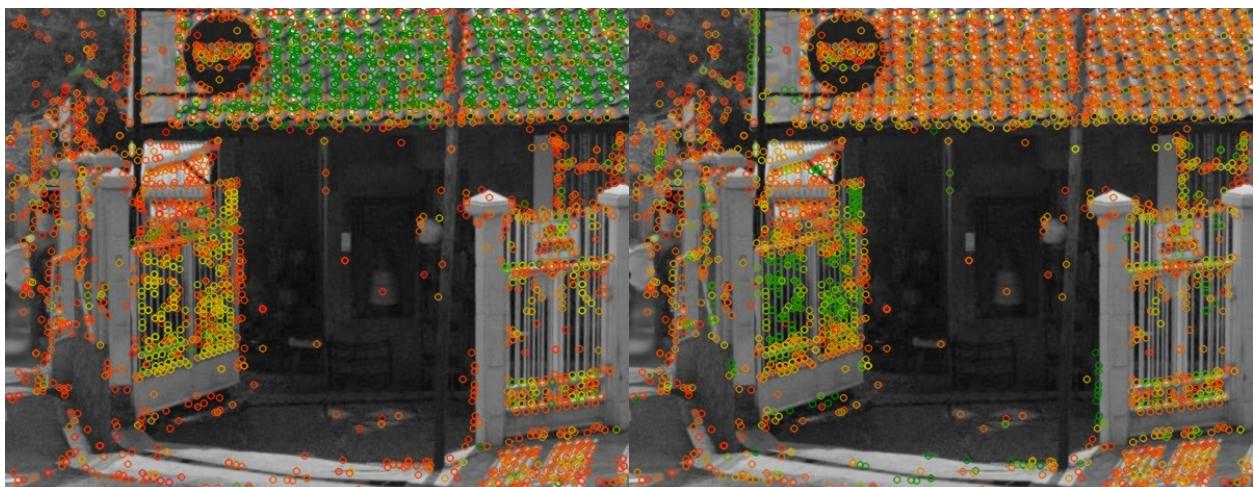
Gambar 3.15: Histogram sebaran nilai keunikan.

¹ 3.6.5 Visualisasi Nilai Konsistensi dan Keunikan Fitur Lokal

- 2 Dari tahap sebelumnya telah didapat nilai konsistensi dan keunikan untuk setiap fitur lokal. Tetapi
- 3 nilai-nilai tersebut belum tentu dengan benar menunjukkan bagian POI yang konsisten dan unik.
- 4 Perlu dilakukan sebuah tes untuk memeriksa apakah nilai-nilai tersebut benar menunjukkan bagian
- 5 atau objek dari POI yang sifatnya konsisten dan unik. Selain itu juga akan dilihat bagian POI

1 seperti apa yang akan cenderung memiliki nilai konsistensi dan keunikan tinggi berdasarkan dengan
2 penghitungan nilai konsistensi dan keunikan pada penelitian ini.

3 Untuk melihat nilai konsistensi dan keunikan dari fitur lokal pada bagian-bagian POI maka perlu
4 untuk menampilkan *keypoint* dari fitur lokal pada gambar asalnya. *Keypoint* yang ditampilkan pada
5 gambar asal tersebut perlu untuk memiliki sebuah indikasi untuk menunjukkan nilai konsistensi dan
6 keunikan dari fitur lokalnya. Pada bagian ini *keypoint* yang ditampilkan pada gambar asal akan
7 diberi warna sesuai dengan nilai konsistensi dan keunikannya. *Keypoint* ditampilkan di gambar
8 asalnya sebanyak dua kali dengan pemberian warna masing-masing untuk nilai konsistensi dan
9 keunikan. Beberapa contoh hasil visualisasi dapat dilihat pada Gambar 3.16. Gambar di sebelah
10 kiri merupakan pemberian warna berdasarkan nilai keunikan dan gambar sebelah kiri berdasarkan
11 nilai konsistensi.



(a) Keunikan - Konsistensi



(b) Keunikan - Konsistensi



(c) Keunikan - Konsistensi

Gambar 3.16: Beberapa contoh gambar POI beserta *keypoint* yang fitur lokalnya sudah diberi warna sesuai dengan nilai konsistensinya dan keunikannya.

1 Berdasarkan hasil berupa gambar-gambar yang ada pada Gambar 3.16 didapatkan beberapa
2 poin berikut. Poin-poin dibagi menjadi 3 kelompok untuk masing-masing gambar POI.

3 **Gambar 3.16a**

- 4 • Gambar ini merupakan contoh kelas yang hasil penghitungan nilai konsistensi dan keunikan
5 memberikan hasil yang baik.
- 6 • Nilai konsistensi/keunikan yang tinggi terpusat pada suatu bagian dari POI.
- 7 • Bagian dari POI yang dinilai konsisten berbeda dengan bagian yang dinilai unik.
- 8 • Bagian atap dari POI merupakan bagian yang dinilai paling unik, dapat dilihat dari mayoritas
9 *keypoint* pada bagian itu berwarna hijau di gambar nilai keunikan.
- 10 • Bagian pagar dari POI merupakan bagian yang dinilai paling konsisten, dapat dilihat dari
11 mayoritas *keypoint* pada bagian tersebut berwarna hijau di gambar nilai konsistensi.
- 12 • Jika dilihat dari kedua gambar, bagian atap merupakan bagian yang unik tetapi tidak konsisten.
13 Hal tersebut menunjukkan bahwa pola yang terdapat pada bagian atap POI tersebut tidak
14 ditemui di POI lainnya, tetapi nilai konsistensinya rendah karena bagian atap tersebut tidak
15 muncul dalam seluruh gambar dari POI tersebut yang digunakan di *dataset*.

16 **Gambar 3.16b**

- 17 • Gambar ini merupakan contoh kelas yang hasil penghitungan nilai konsistensi dan keunikan
18 memberikan hasil yang cukup baik.
- 19 • Baik nilai konsistensi dan keunikan keduanya memiliki nilai tinggi yang terpusat pada bagian
20 yang sama, yaitu pada bagian logo (tulisan WARMINDO).
- 21 • Nilai keunikan memberikan hasil yang sedikit lebih baik, dapat dilihat bahwa hanya bagian
22 logo yang memiliki *keypoint* warna hijau. Sedangkan untuk bagian-bagian lainnya semuanya
23 memiliki *keypoint* dengan warna merah.
- 24 • Nilai konsistensi walau memberikan hasil yang cukup baik, tetapi masih terlihat dengan
25 beberapa *keypoint* dengan nilai yang tidak rendah (warna oranye-kuning) di bagian-bagian
26 seharusnya tidak unik atau konsisten.

27 **Gambar 3.16c**

- 28 • Gambar ini merupakan contoh kelas yang hasil penghitungan nilai konsistensi dan keunikan
29 memberikan hasil yang kurang baik.
- 30 • Baik dari nilai konsistensi maupun keunikan, sangat sedikit *keypoint* yang memiliki nilai tinggi
31 (warna hijau).
- 32 • Terlihat beberapa *keypoint* berwarna hijau pada gambar konsistensi, tetapi letaknya tersebar
33 di seluruh bagian gambar.

3.7 Analisis Penentuan Nilai Threshold untuk Konsistensi dan Keunikan

Pada analisis-analisis sebelumnya telah didapat dua buah nilai untuk masing-masing fitur lokal. Kedua buah nilai tersebut masing-masing menyatakan tingkat konsistensi dan keunikan sebuah fitur lokal terhadap POI-nya. Kedua nilai tersebut memiliki rentang dari 0 sampai 1, di mana semakin tinggi nilainya (semakin mendekati 1) maka semakin konsisten/unik fitur lokal tersebut.

Nilai-nilai konsistensi dan keunikan pada setiap fitur lokal dapat digunakan untuk menyaring fitur lokal. Dengan memanfaatkan nilai-nilai konsistensi dan keunikan tersebut dapat dipilih hanya fitur lokal yang sifatnya konsisten/unik saja. Permasalahannya adalah sulit untuk menentukan sebuah batas (*threshold*) yang ideal sehingga dapat diperoleh fitur-fitur lokal yang asalnya dari objek dalam POI yang sifatnya memang konsisten dan unik. Objek dalam POI yang sifatnya konsisten dan unik dalam penelitian ini diartikan sebagai objek yang selalu terlihat dalam gambar-gambar POI dan memiliki bentuk atau pola yang tidak terlihat di POI lain. Contoh yang paling mudah dari objek-objek dengan sifat yang konsisten dan unik ini adalah logo dari POI. Gambar 3.17 merupakan beberapa contoh gambar POI, kotak hijau pada gambar POI tersebut menandakan contoh objek dalam POI yang sifatnya konsisten atau unik.



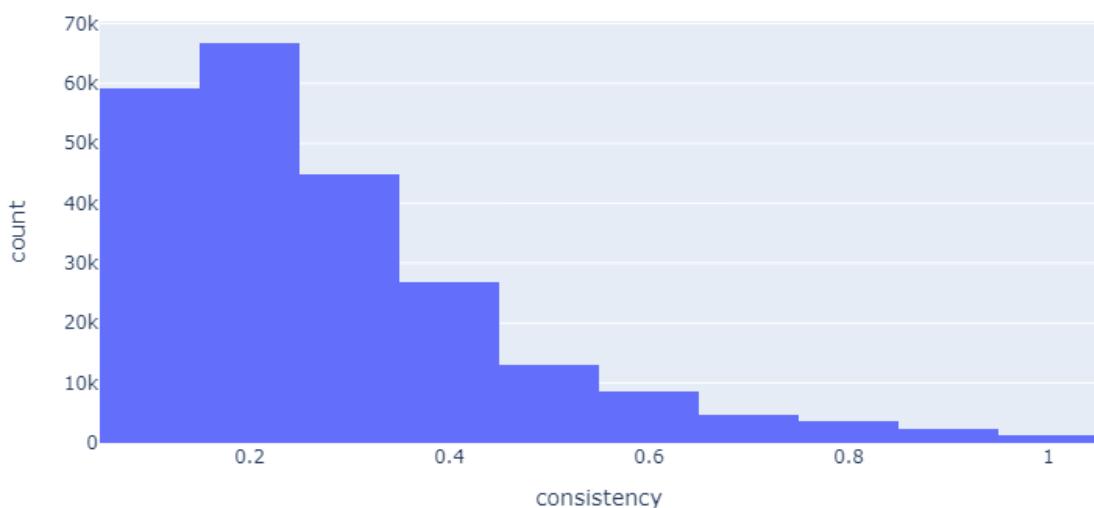
Gambar 3.17: Contoh objek yang sifatnya konsisten dan unik dalam POI.

Penentuan *threshold* untuk nilai konsistensi dan keunikan akan dilakukan dengan mencoba beberapa nilai *threshold*. Untuk masing-masing nilai *threshold* akan diambil *keypoint* yang nilai konsisten/keunikan fitur lokalnya lebih tinggi dari nilai tersebut. *Keypoint* yang nilainya lebih dari *threshold* tersebut akan ditampilkan pada gambar dan dilihat apakah *keypoint-keypoint* tersebut berasal dari objek dalam POI yang memang bersifat konsisten/unik. Proses ini akan dilakukan masing-masing untuk nilai konsistensi dan keunikan.

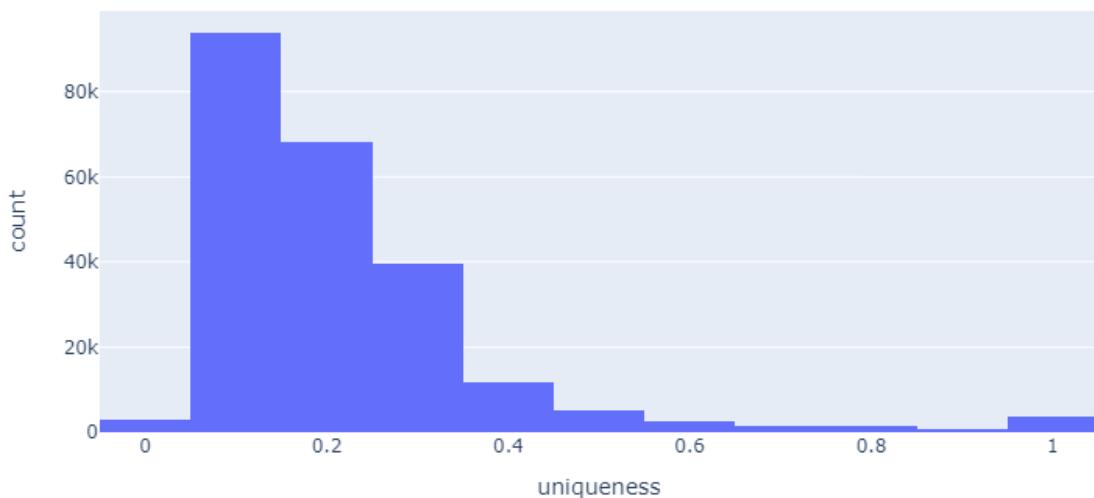
Nilai konsistensi dan keunikan untuk fitur lokal pada *dataset* tersebut seperti ditunjukkan oleh dua histogram pada Gambar 3.18. Jika dilihat dari kedua histogram pada Gambar 3.18 tersebut

- 1 baik nilai konsistensi dan keunikan tersebar dengan sebaran yang mirip. Mayoritas dari fitur lokal pada gambar POI memiliki nilai konsistensi dan keunikan yang rendah.

Histogram Sebaran Nilai ***consistency***



Histogram Sebaran Nilai ***uniqueness***



Gambar 3.18: Contoh objek yang sifatnya konsisten dan unik dalam POI.

- 3 Jika dilihat dari contoh pada Gambar 3.17 bagian yang konsisten dan unik dari dari sebuah
- 4 gambar POI hanya merupakan bagian kecil jika dibandingkan dengan keseluruhan gambar. Karena
- 5 ingin diambil fitur lokal dari bagian POI yang merupakan bagian kecil dari gambar maka hanya
- 6 perlu diambil sebagian kecil dari keseluruhan fitur lokal. Oleh karena itu—dengan melihat sebaran
- 7 dari kedua histogram pada Gambar 3.18—fitur lokal dengan nilai konsistensi dan keunikan < 0.6
- 8 akan dibuang terlebih dahulu. Pembuangan fitur lokal ini karena fitur lokal dengan nilai konsistensi

- 1 dan keunikan tersebut diasumsikan berasal dari bagian POI dengan sifat tidak konsisten dan tidak
- 2 unik yang merupakan sebagian besar dari sebuah gambar POI.

3 3.7.1 Ide Dasar dan Tahapan Analisis

4 Analisis ini dilakukan untuk menentukan sebuah nilai batas bawah (*threshold*) yang paling ideal
5 untuk digunakan dalam memilih fitur lokal yang sifatnya konsisten dan unik. Analisis akan dilakukan
6 dengan menyaring fitur lokal menggunakan nilai *threshold* untuk masing-masing nilai konsistensi
7 dan keunikan. Akan digunakan lima *threshold* yang berbeda: 0.6, 0.7, 0.8, 0.9, dan 1.0. Untuk
8 setiap *threshold* yang digunakan maka diambil fitur lokal yang nilai konsistensi/keunikannya lebih
9 dari atau sama dengan nilai *threshold* tersebut. Fitur lokal yang didapat dari hasil penyaringan
10 tersebut lalu ditampilkan *keypoint*-nya pada gambar. Dari gambar dengan *keypoint* tersebut lalu
11 dilihat apakah *keypoint-keypoint* yang ditampilkan tepat berada pada bagian objek yang sifatnya
12 memang konsisten dan unik.

13 Nilai konsistensi dan keunikan pada analisis ini didapat dari hasil proses *clustering* seperti
14 analisis pada 3.6. Dataset yang digunakan untuk proses *clustering* berasal dari 10 POI yang berbeda
15 dengan masing-masing POI terdiri dari 10 gambar berbeda dari POI tersebut. Sebanyak total
16 100 gambar tersebut lalu diproses untuk mendapatkan nilai konsistensi dan keunikan untuk tiap
17 fitur lokalnya. Dari sebanyak 10 kelas gambar (jenis POI) akan diambil sebanyak 3 kelas untuk
18 digunakan sebagai sampel dalam menguji nilai *threshold*.

19 Tahapan yang dilakukan pada analisis ini adalah seperti dijelaskan pada tahapan di bawah ini.
20 Langkah-langkah tahapan ini dilakukan setelah didapat fitur lokal beserta nilai konsistensi dan
21 keunikannya melalui proses *clustering*:

- 22 1. Memuat *dataset* berupa fitur lokal beserta keterangan untuk tiap fitur lokal
- 23 2. Memilih sebuah POI (kelas gambar) sebagai sampel pengujian
- 24 3. Mengambil hanya fitur lokal dari kelas gambar yang dipilih
- 25 4. Melakukan iterasi untuk tiap nilai *threshold*
- 26 5. Untuk tiap iterasi mengambil fitur lokal yang nilai konsistensi/keunikannya \geq *threshold* dan
menampilkan *keypoint*-nya pada gambar.

27 Langkah 4 dan 5 pada tahapan di atas dilakukan untuk masing-masing nilai konsistensi dan
28 keunikan. Tahapan tersebut akan menghasilkan gambar-gambar yang digunakan pada *dataset*
29 disertai dengan *keypoint*-nya yang telah tersaring berdasarkan nilai *threshold*. Hasil gambar tersebut
30 akan diproses untuk dihitung skor ketepatannya. Skor ketepatan akan menunjukkan seberapa
31 bagus hasil penyaringan fitur lokal dengan menggunakan sebuah nilai *threshold*. Metode untuk
32 menghitung skor ketepatan dijelaskan pada subbab di bawah ini.

34 3.7.2 Metode Scoring

35 Pada analisis ini pemberian nilai untuk setiap *threshold* yang digunakan dengan menghitung
36 persentase *keypoint* yang berada di daerah *target*. Daerah *target* adalah daerah yang dari sebuah
37 gambar POI yang sifatnya konsisten dan unik terhadap POI tersebut, jika dinilai secara manusia.
38 Daerah *target* akan berupa logo dari POI atau objek lain dari POI yang konsisten dan unik. Contoh
39 daerah *target* adalah seperti pada Gambar 3.17.

1 Implementasi daerah *target* tersebut dilakukan dengan menggunakan aplikasi LabelImg (<https://github.com/heartexlabs/labelImg>). Aplikasi LabelImg sendiri adalah aplikasi yang dapat
2 digunakan untuk memberi anotasi pada gambar. Tampilan layar LabelImg dapat dilihat pada
3 Gambar 3.19.



Gambar 3.19: Contoh tampilan aplikasi Labelimg

Dengan menggunakan aplikasi Labelimg kita dapat menandai sebuah daerah pada gambar dan memberikan label pada daerah tersebut. Aplikasi nanti akan menghasilkan sebuah *file* xml yang berisi informasi titik-titik koordinat daerah yang ditandai dan label daerah tersebut. Format sebuah anotasi dalam *file* xml dapat dilihat pada potongan kode di bawah ini.

```
9
101 <annotation>
112     <folder>annotated_alfamart</folder>
123     <filename>alfamart_1.jpg</filename>
134     <path>D:\ Skripsi - Program3\datasets\annotated_poi\alfamart\alfamart_1.
14         jpg</path>
155     <source>
166         <database>Unknown</database>
177     </source>
188     <size>
199         <width>600</width>
200         <height>474</height>
211         <depth>1</depth>
222     </size>
233     <segmented>0</segmented>
244     <object>
255         <name>target</name>
266         <pose>Unspecified</pose>
277         <truncated>0</truncated>
288         <difficult>0</difficult>
```

```

19 <bndbox>
20     <xmin>84</xmin>
21     <ymin>40</ymin>
22     <xmax>497</xmax>
23     <ymax>89</ymax>
24 </bndbox>
25 </object>
26 </annotation>

```

- 10 Potongan kode tersebut menunjukkan sebuah *file xml* yang dihasilkan dari Labelimg. Pada *file*
 11 tersebut terlihat bahwa ada sebuah anotasi dengan nama **target** yang ditunjukkan oleh *tag object*.
 12 Area yang ditandai sebagai target dapat dilihat pada *tag bndbox* yang berada di dalam *tag object*.
 13 Fungsi penandaan daerah tersebut akan digunakan pada setiap gambar untuk membuat *file xml*
 14 yang menunjukkan anotasi pada gambar tersebut. Sebuah gambar dapat memiliki lebih dari satu
 15 anotasi, di mana beberapa anotasi berbeda itu akan disimpan pada sebuah *file xml* yang sama.
 16 Pada analisis ini setiap gambar yang digunakan pada *dataset* akan terlebih dahulu diberi anotasi
 17 dengan label **target**. Setiap gambar dapat memiliki lebih dari satu anotasi (ada beberapa objek
 18 konsisten dan unik pada POI yang saling terpisah). Setiap **keypoint** yang didapat setelah tersaring
 19 dengan *threshold* akan ditentukan apakah **keypoint** tersebut berada di dalam atau di luar daerah
 20 **target**. Nilai *score* akan ditentukan dari persentase **keypoint** yang berada di dalam daerah **target**.

21 3.7.3 Hasil Analisis

- 22 Setelah dilakukan tahapan dan juga pemberian nilai sesuai seperti yang telah dijelaskan pada
 23 subbab-subbab sebelumnya didapat hasil seperti yang dijabarkan di bawah ini.

24 Nilai Konsistensi

- 25 Untuk tiap *threshold* yang digunakan pada nilai konsistensi didapat rata-rata nilai skor untuk tiap
 26 kelas gambar seperti yang ditampilkan pada Tabel 3.3.

threshold	0.6	0.7	0.8	0.9	1.0
img_class					
alfamart	0.149268	0.144771	0.158488	0.089263	0.112664
binus	0.507728	0.476035	0.294314	0.010000	0.000000
gembul	0.287119	0.291524	0.311475	0.264435	0.186858
harris	0.186551	0.182550	0.185563	0.233336	0.000000
oxy	0.083837	0.071717	0.061667	0.061667	0.000000
ping	0.715547	0.673606	0.660579	0.569977	0.177269
porcafe	0.272689	0.300066	0.318216	0.000000	0.000000
starbuck	0.235460	0.221705	0.270528	0.413136	0.294983
toyota	0.119899	0.078839	0.081004	0.061839	0.000000
yogya	0.218260	0.224289	0.330336	0.299550	0.299550

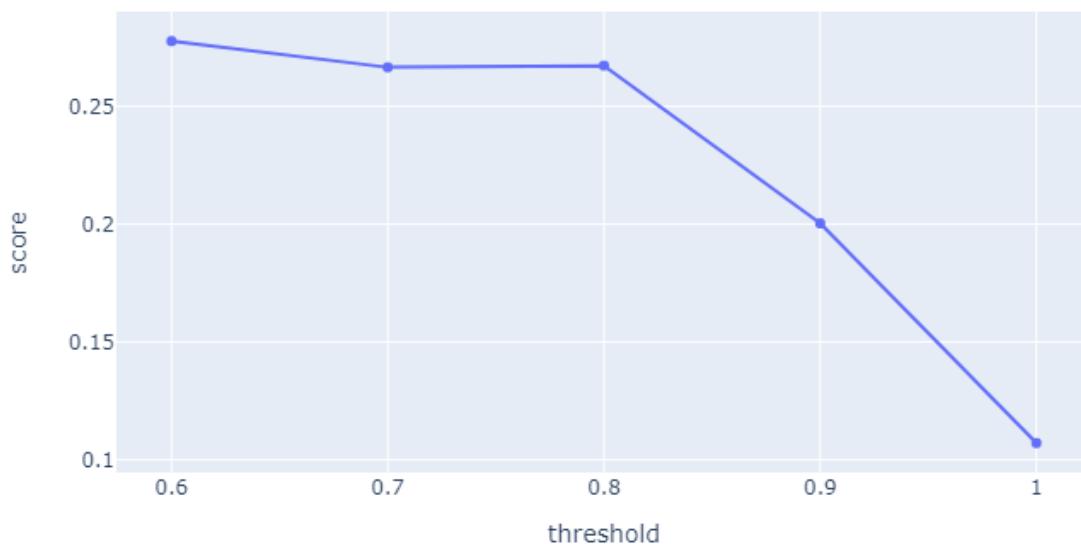
Tabel 3.3: Rata-rata skor ketepatan tiap kelas untuk tiap *threshold* pada nilai konsistensi.

- 27 Jika dilihat dari Tabel 3.3 skor ketepatan cenderung berada pada angka yang kecil. Skor
 28 ketepatan juga nilainya sangat berbeda tiap kelas gambar. Contohnya seperti kelas *ping* nilainya

1 bisa mencapai $\tilde{0.7}$ pada *threshold* 0.6, dibandingkan dengan skor untuk kelas *oxy* yang nilai
 2 tertingginya hanya pada $\tilde{0.08}$ saja. Nilai skor yang sangat berbeda tersebut menunjukkan
 3 bahwa ada kelas-kelas POI yang cenderung mudah dikenali dan ada yang lebih sulit untuk dikenali.
 4 POI yang mudah dikenali mungkin adalah POI yang memiliki objek dengan sebuah pola yang kuat
 5 dan mudah terdeteksi oleh algoritma pencarian fitur lokal.

6 Untuk memilih *threshold* mana yang paling baik digunakan untuk pemotongan pada nilai
 7 konsistensi maka akan dihitung rata-rata nilai skor seluruh kelas untuk tiap *threshold*. Hasilnya
 8 dapat dilihat pada grafik di Gambar 3.20.

Skor Ketepatan untuk tiap Threshold pada Nilai Konsistensi



Gambar 3.20: Skor ketepatan untuk tiap *threshold* pada nilai konsistensi.

9 Dari grafik pada Gambar 3.20 terlihat bahwa nilai skor ketepatan tertinggi ada pada *threshold*
 10 0.6. Nilai skor tidak berubah banyak sampai *threshold* 0.8, dan setelah itu ada penurunan yang
 11 cukup tajam pada *threshold* 0.9 dan 1.0. Dari hasil ini maka akan digunakan 0.6 sebagai *threshold*
 12 yang digunakan untuk memotong nilai konsistensi. Walaupun jika dilihat dari sebaran tiap kelasnya
 13 terlihat sebaran yang tidak merata sehingga nilai *threshold* yang ditentukan mungkin tidak akan
 14 baik untuk semua kelas.

15 Nilai Keunikan

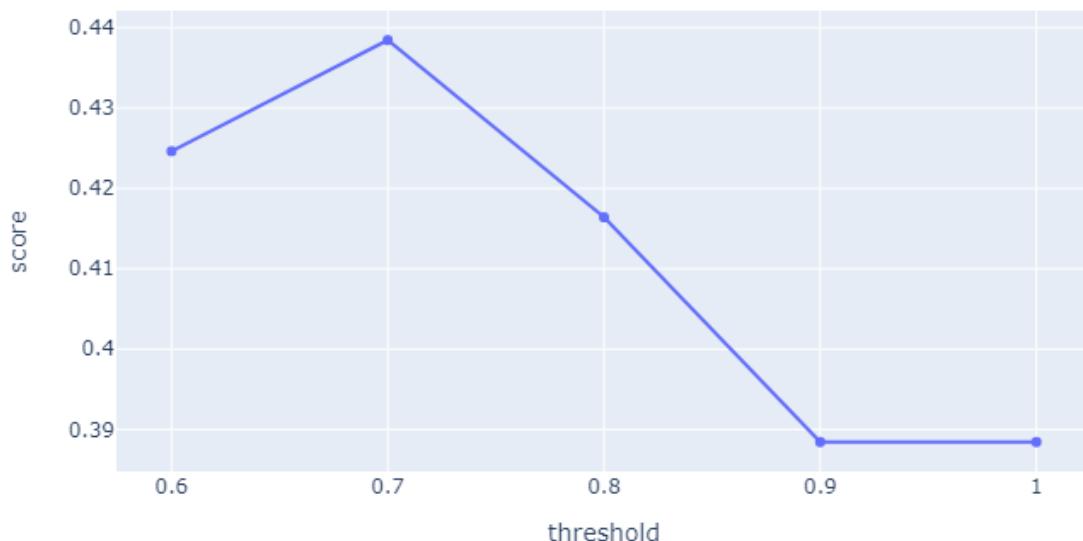
16 Untuk tiap *threshold* yang digunakan pada nilai keunikan didapat rata-rata nilai skor untuk tiap
 17 kelas gambar seperti yang ditampilkan pada Tabel 3.4.

threshold	0.6	0.7	0.8	0.9	1.0
img_class					
alfamart	0.515710	0.366667	0.383333	0.100000	0.100000
binus	0.677219	0.680215	0.676765	0.591431	0.591431
gembul	0.635185	0.660688	0.679726	0.680655	0.680655
harris	0.208597	0.158760	0.059408	0.154762	0.154762
oxy	0.161507	0.053333	0.055098	0.000000	0.000000
ping	0.952552	0.972462	0.959630	0.985047	0.985047
porcafe	0.166364	0.250000	0.250000	0.200000	0.200000
starbucks	0.375029	0.439195	0.516766	0.589057	0.589057
toyota	0.191150	0.406667	0.250000	0.250000	0.250000
yogya	0.363011	0.396667	0.333333	0.333333	0.333333

Tabel 3.4: Rata-rata skor ketepatan tiap kelas untuk tiap *threshold* pada nilai keunikan.

Nilai-nilai pada Tabel 3.4 menunjukkan nilai dengan kisaran yang mirip dengan saat digunakan nilai konsistensi (Tabel 3.3). Pada penggunaan nilai keunikan ini juga kelas *ping* memiliki nilai rata-rata skor yang lebih tinggi jika dibandingkan dengan kelas-kelas lainnya. Kelas *oxy* juga memiliki nilai skor yang kisarannya paling rendah diantara semua kelas lainnya. Sedangkan untuk rata-rata skor seluruh kelas untuk tiap *threshold* pada nilai konsistensi dapat dilihat pada Gambar 3.21.

Skor Ketepatan untuk tiap Threshold pada Nilai Keunikan

Gambar 3.21: Skor ketepatan untuk tiap *threshold* pada nilai keunikan.

Grafik pada Gambar 3.21 menunjukkan bahwa skor tertinggi ada pada saat digunakan nilai *threshold* 0.7. Dari nilai pada grafik tersebut maka akan digunakan nilai *threshold* 0.7 sebagai batas untuk memotong nilai keunikan. Sama seperti pada nilai konsistensi, sebaran skor antar kelas pada nilai keunikan tidak merata sehingga nilai *threshold* yang ditentukan mungkin tidak akan baik untuk semua kelas.

3.8 Analisis Penggunaan Nilai Konsistensi dan Nilai Keunikan untuk OIR dengan BSIS

Pada analisis sebelumnya telah diterapkan metode *clustering* untuk mencari fitur lokal yang unik dan konsisten. Fitur lokal yang unik dan konsisten tersebut diharapkan dapat menjadi fitur yang kuat untuk melakukan identifikasi sebuah gambar.

Penelitian ini tidak dilakukan dengan menggunakan *dataset* GSV (3.3.2). Pada tahap ini penelitian dilakukan dengan menggunakan *dataset book_covers* dari SMVS (3.3.1).

Analisis dilakukan dengan melakukan identifikasi sejumlah gambar tes terhadap *dataset* yang menggunakan semua fitur lokal dan *dataset* yang menggunakan fitur lokal yang telah tersaring berdasarkan nilai konsistensi dan keunikannya. Penyaringan fitur lokal pada analisis ini tidak dilakukan dengan menggunakan pengujian *threshold* seperti pada 3.7. Penentuan batas untuk menyaring fitur lokal ditentukan dengan melihat sebaran nilai konsistensi dan nilai fitur lokalnya.

Analisis bertujuan untuk membandingkan hasil tes *dataset* secara keseluruhan dan *dataset* yang telah disaring. Ingin dilihat bagaimana perbedaan tingkat akurasi dan waktu pemrosesan yang diperlukan.

3.8.1 Data Train dan Test

Data Train

Data *train* yang digunakan berjumlah total sebanyak 200 gambar yang terbagi menjadi 50 kelas. Tiap kelas berjumlah 4 gambar buku yang sama diambil dari sudut pengambilan yang berbeda dan terdapat objek di latar belakang yang berbeda. Salah satu contoh kelas dari *dataset* dapat dilihat pada Gambar 3.22.



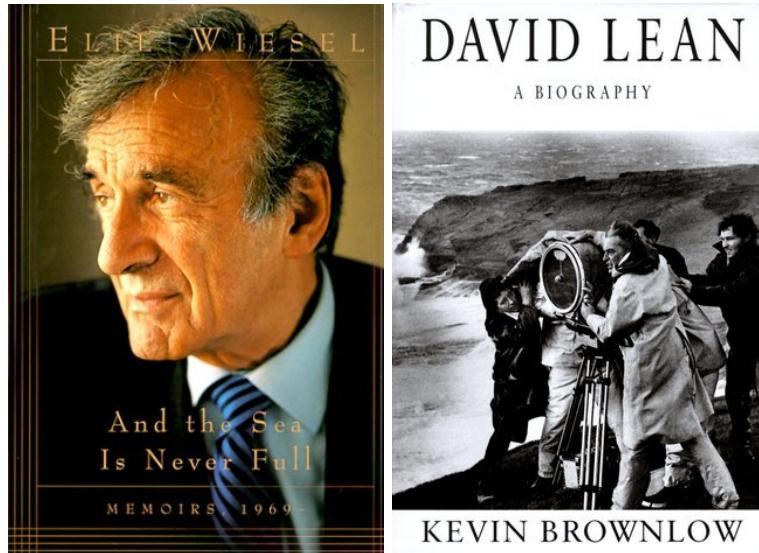
Gambar 3.22: Contoh gambar pada *dataset book_covers*. Keempat gambar tersebut berada dalam satu kelas yang sama.

Data *train* yang sama digunakan dua kali dengan ukuran yang berbeda. Ukuran gambar diperkecil sehingga sisi paling panjangnya tidak lebih dari 400 *pixel* untuk set pertama dan tidak lebih dari 600 *pixel* untuk set kedua. Ukuran gambar yang berbeda akan menghasilkan jumlah fitur lokal yang berbeda juga, semakin besar ukurannya maka semakin banyak fitur lokal yang dihasilkan. Penggunaan ukuran yang berbeda ini bertujuan untuk melihat apakah ada perbedaan akurasi jika ukuran gambar pada *dataset* berubah.

- 1 Gambar-gambar tersebut diekstrak fitur lokalnya dan untuk setiap fitur lokal dihitung nilai
 2 konsistensi dan keunikan menggunakan metode seperti pada [3.6](#). Fitur-fitur lokal dari gambar ini
 3 lah yang akan digunakan untuk melakukan identifikasi gambar tersebut.

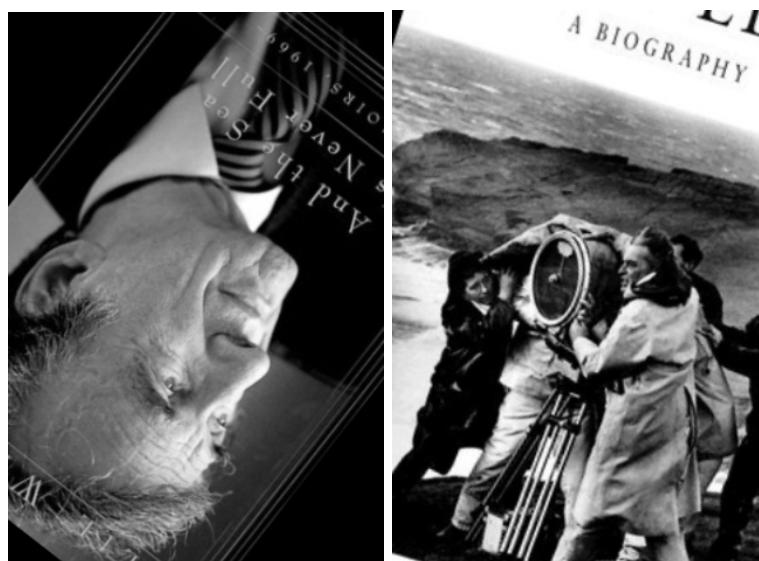
4 Data Test

- 5 Data *test* yang digunakan diambil dari gambar referensi *dataset book_covers*. Gambar referensi
 6 merupakan gambar buku yang menjadi objek utama dalam setiap kelas dengan kondisi yang ideal.
 7 Contoh gambar referensi dapat dilihat pada Gambar [3.23](#).



Gambar 3.23: Contoh gambar referensi dari *dataset book_covers*.

- 8 Gambar-gambar referensi tersebut berjumlah total 50 gambar masing-masing untuk tiap kelas.
 9 Dari 50 gambar tersebut lalu ditransformasi dengan mengubah tingkat perbesaran dan rotasi
 10 secara acak. Transformasi tersebut dilakukan sebanyak dua kali untuk tiap gambar. Contoh
 11 gambar referensi yang telah ditransformasi dapat dilihat pada Gambar [3.24](#). Masing-masing hasil
 12 transformasi disimpan sebagai satu gambar sehingga terdapat total sebanyak 100 gambar untuk
 13 data *test*.
 14 Data *test* yang berjumlah 100 tersebut memiliki ukuran lebar yang berkisar antara 133 – 441
 15 pixel, sedangkan untuk tingginya berkisar antara 198 – 502 pixel.



Gambar 3.24: Contoh gambar referensi dari *dataset book_covers* yang telah ditransformasi.

3.8.2 Metode BSIS

Analisis ini akan melakukan identifikasi terhadap gambar yang dilakukan menggunakan metode BSIS seperti yang telah dijelaskan pada 2.5. Metode BSIS sendiri secara garis besar terdiri dari 3 langkah sebagai berikut:

1. *Pairing*

Untuk setiap fitur lokal pada gambar *test* dibuat pasangan dengan memasangkannya pada fitur lokal dari *dataset train*.

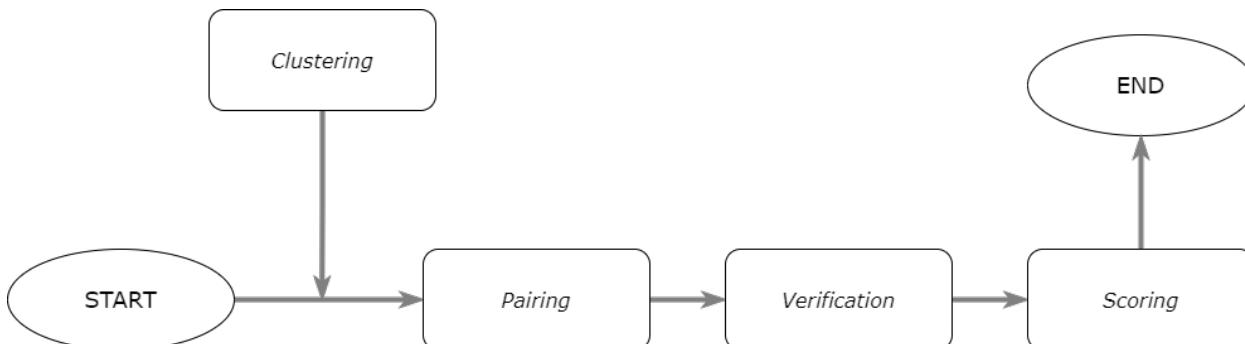
2. *Verification*

Dari pasangan-pasangan yang terbentuk dari tahap sebelumnya dilakukan verifikasi untuk mencari pasangan yang konsisten secara geometris.

3. *Scoring*

Setelah didapat pasangan-pasangan yang memiliki sifat konsisten secara geometris dihitung total bobot dari pasangan-pasangan tersebut sebagai nilai yang menunjukkan kemiripan dua buah gambar tersebut.

Pada analisis ini dilakukan sedikit modifikasi pada metode BSIS, modifikasi dilakukan seperti yang ditunjukkan pada Gambar 3.25

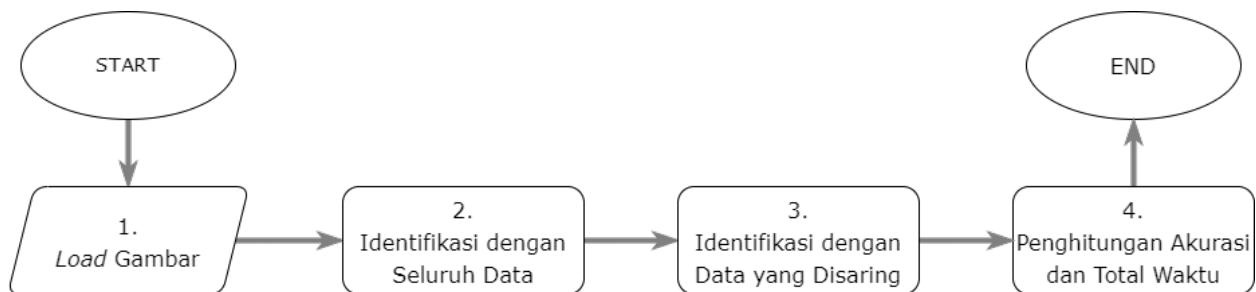


Gambar 3.25: Modifikasi pada metode BSIS yang dilakukan pada analisis ini.

- 1 Terdapat tahap *clustering* yang dilakukan terhadap *dataset train* sebelum melakukan *pairing*, sehingga *pairing* dilakukan terhadap *dataset train* yang telah tersaring berdasarkan hasil *clustering*.
- 2 Tahap penyaringan akan menyebabkan fitur lokal *dataset train* yang perlu diperiksa menjadi lebih sedikit. Hal ini akan mempercepat waktu yang diperlukan untuk melakukan *pairing*.

5 3.8.3 Tahapan Analisis

- 6 Analisis dilakukan dengan tahapan pada dua *dataset* yang berbeda dengan tahapan yang sama.
- 7 *Dataset* pertama akan disebut Book Covers 400 yang menggunakan gambar dengan ukuran sisi terbesarnya tidak lebih dari 400 *pixel*. *Dataset* kedua disebut Book Covers 600 yang ukuran sisi terbesarnya tidak lebih dari 600 *pixel*. Tahapan analisis yang dilakukan dapat dilihat pada *flowchart* di Gambar 3.26.



Gambar 3.26: Tahapan yang dilakukan dalam analisis untuk menguji penggunaan *clustering* pada BSIS.

- 11 Secara rinci tahapan yang dilakukan adalah sebagai berikut:

- 12 1. *Load* Gambar

13 *Load* semua gambar *test* yang digunakan.

- 14 2. Identifikasi dengan BSIS pada seluruh Dataset

15 Untuk masing-masing gambar *test* lakukan ekstraksi fitur lokal dan gunakan fitur lokal tersebut untuk melakukan identifikasi BSIS dengan menggunakan seluruh fitur lokal pada *dataset*.
16 Untuk setiap gambar hitung waktu yang digunakan untuk melakukan BSIS hingga didapat hasilnya. Total waktu tidak termasuk waktu yang digunakan untuk ekstraksi fitur.

- 17 3. Identifikasi dengan BSIS pada Dataset yang Telah tersaring

18 Lakukan kembali identifikasi seluruh gambar *test* tetapi dengan menggunakan hanya sebagian *dataset* yang telah tersaring berdasarkan nilai keunikan dan konsistensinya.

- 19 4. Penghitungan Akurasi dan Total Waktu

20 Hitung total waktu yang digunakan untuk menyelesaikan seluruh gambar dan berapa gambar yang mendapat hasil benar.

- 21 Tahapan tersebut dilakukan sebanyak 2 kali untuk masing-masing Book Covers 400 dan Book Covers 600. Setelah didapat hasil akurasi dan total waktu untuk Book Covers 400 dan Book Covers 600 bandingkan keduanya dan lakukan analisis.

22 3.8.4 Tahapan Implementasi

- 23 Implementasi dilakukan dengan membuat program Python. Digunakan Python versi 3.7.5 dengan library OpenCV versi 4.5.5.64 untuk pemrosesan gambar dan Pandas versi 1.3.5 untuk memroses

- 1 data. Langkah-langkah implementasi yang dilakukan adalah sebagai berikut:
- 2 1. Melakukan *import* untuk *library* yang digunakan: OpenCV (*cv2*) dan Pandas (*pandas*).
- 3 2. Memuat semua gambar *test* masukan ke dalam variabel *list* bernama *test_dataset*. Setiap
- 4 elemen dalam *test_dataset* berupa *tuple* dengan dua elemen. Elemen pertama merupakan
- 5 nama gambar dan elemen kedua merupakan *array* gambar tersebut.
- 6 3. Memuat *dataset* yang digunakan untuk data *train*.
- 7 4. Menentukan parameter batas nilai *uniqueness* dan *consistency* untuk menyarang *dataset*.
- 8 5. Membuat objek SIFT dengan fungsi *cv2.SIFT_create*.
- 9 6. Melakukan identifikasi gambar dengan BSIS. Tahap *pairing* dalam BSIS dilakukan dengan
- 10 menggunakan metode KD-Tree.
- 11 7. Menyimpan 10 nilai dari hasil BSIS masing-masing dalam sebuah *list*. Nilai-nilai yang disimpan
- 12 adalah sebagai berikut:
- 13
 - *q_name*: nama dari gambar *test*.
 - *q_class*: kelas gambar dari gambar *test*.
 - *most_similar*: nama gambar hasil BSIS yang paling mirip dengan data *test* (nilai total bobotnya paling tinggi).
 - *most_similar_class*: kelas gambar paling mirip dari BSIS.
 - *total_weight*: total bobot gambar paling mirip dari hasil BSIS.
 - *same_class_idx*: posisi gambar dengan kelas yang sama yang bobotnya paling tinggi. Jika hasil identifikasi benar maka nilainya akan 0.
 - *same_class_weight*: total bobot gambar dengan kelas yang sama yang bobotnya paling tinggi. Jika hasil identifikasi benar maka nilainya akan sama dengan nilai pada *total_weight*.
 - *extract_time*: waktu yang diperlukan untuk ekstraksi fitur lokal gambar *test*.
 - *pairing_time*: waktu yang diperlukan untuk menyelesaikan tahap *pairing* dalam BSIS.
 - *total_bsisc_time*: waktu total yang diperlukan untuk menyelesaikan seluruh tahapan BSIS hingga didapat hasil.
- 28 8. Menggabungkan 10 *list* tersebut ke dalam sebuah *dataframe*. Masing-masing *list* menjadi satu
- 29 kolom dalam *dataframe*
- 30 9. Untuk setiap baris di *dataframe* memeriksa apakah hasilnya benar. Hasil adalah benar jika
- 31 nilai pada *q_class* sama dengan *most_similar_class*.
- 32 10. Melakukan lagi langkah 6-8 dengan *dataset* yang telah tersaring berdasarkan nilai batas
- 33 *consistency* dan *uniqueness* yang telah ditentukan sebelumnya.
- 34 Langkah-langkah tersebut dilakukan untuk Book Covers 400 dan Book Covers 600. Hasil dari kedua
- 35 *dataset* tersebut lalu dibandingkan.

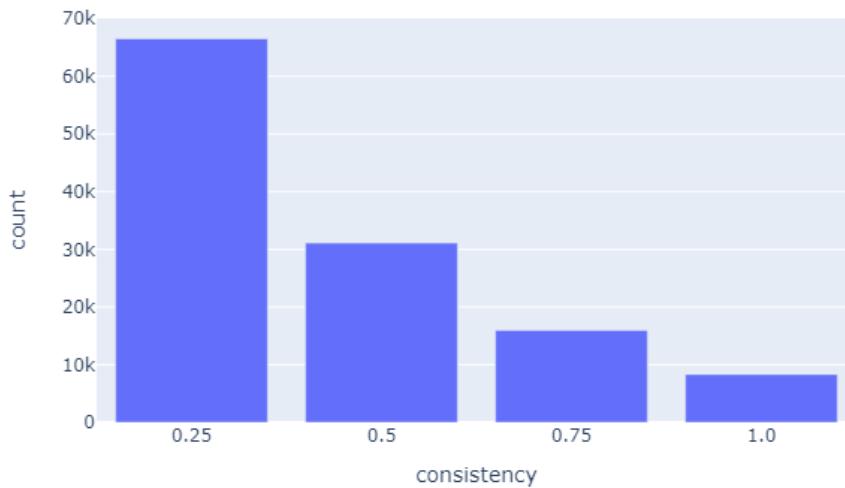
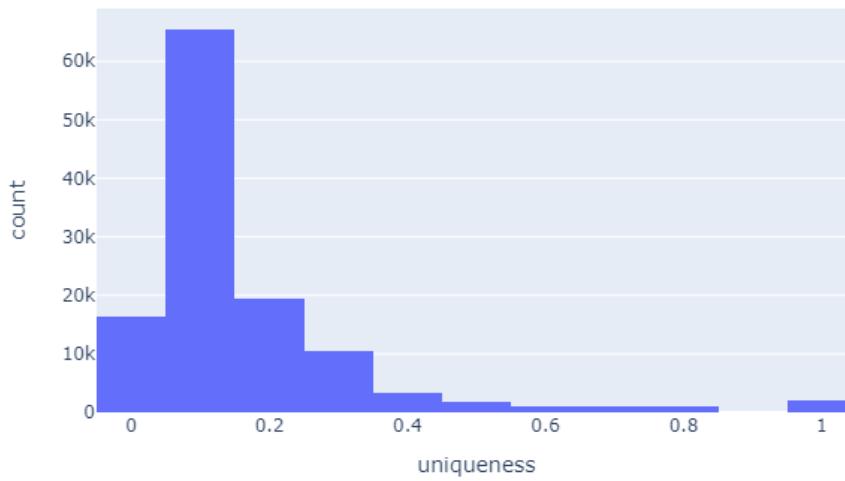
36 **3.8.5 Hasil Analisis**

37 **Book Covers 400**

38 Book Covers 400 memiliki total fitur lokal yang dihasilkan dari seluruh gambar sebanyak 121909

39 fitur lokal. Setelah dilakukan *clustering* menghitung nilai keunikan dan konsistensi didapat hasil

40 sebaran nilai keunikan dan konsistensi seperti pada Gambar 3.27 dan Gambar 3.28.

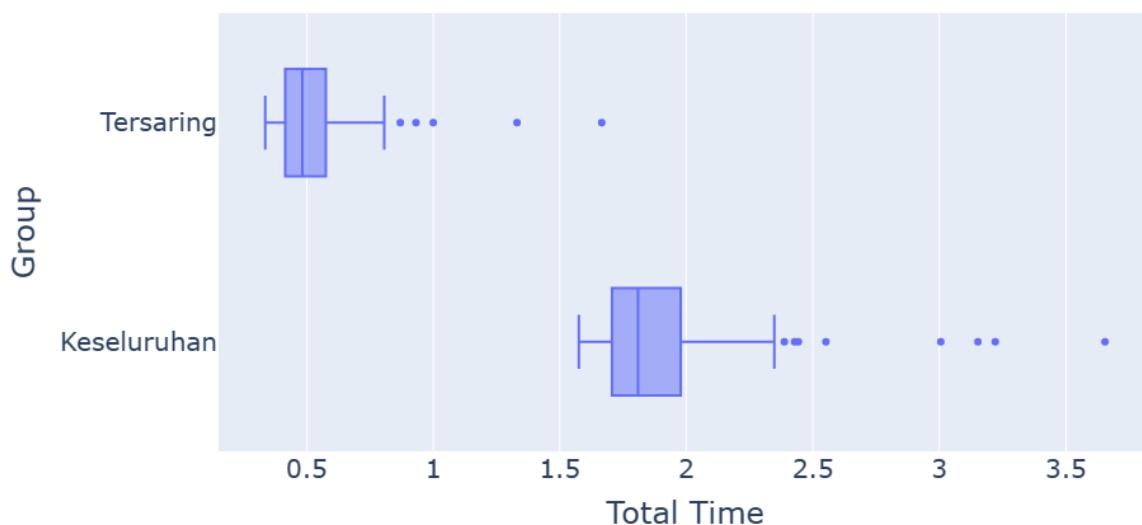
Histogram Sebaran Nilai ***consistency***Gambar 3.27: Sebaran nilai konsistensi (***consistency***) untuk Book Covers 400.Histogram Sebaran Nilai ***uniqueness***Gambar 3.28: Sebaran nilai keunikan (***uniqueness***) untuk Book Covers 400.

- 1 Berdasarkan sebaran nilai konsistensi dan keunikan pada Gambar 3.27 dan Gambar 3.28 ditentukan bahwa fitur lokal yang digunakan untuk data yang telah tersaring adalah fitur lokal dengan ciri sebagai berikut:
- 2 • Memiliki nilai ***consistency*** ≥ 0.5
- 3 • Memiliki nilai ***uniqueness*** ≥ 0.2
- 4 Nilai yang digunakan sebagai batas pengambilan fitur lokal tersebut pada saat penelitian ini tidak didapat dari hasil penghitungan. Nilai tersebut digunakan karena dengan menggunakan batas
- 5
- 6
- 7

tersebut dapat menyaring sebagian besar fitur-fitur lokal yang nilai **consistency** dan **uniqueness**nya rendah. Dengan menggunakan nilai-nilai batas tersebut jumlah fitur lokal pada *dataset* menurun dari yang sebelumnya sebanyak 121909 data, menjadi sebanyak 24040 data.

Keseluruhan *dataset* dan *dataset* yang telah tersaring digunakan untuk melakukan identifikasi dengan BSIS dan dihitung akurasi serta total waktu yang digunakan. Kedua variasi *dataset* tersebut memiliki sebaran waktu proses yang cukup berbeda. Perbandingan waktu keduanya dapat dilihat pada *boxplot* di Gambar 3.29.

Perbandingan Sebaran Waktu tiap Kelompok Dataset



Gambar 3.29: Perbandingan sebaran waktu Book Covers 400.

Detail untuk kedua *boxplot* pada Gambar 3.29 tersebut dapat dilihat pada Tabel 3.5 berikut.

	Tersaring	Keseluruhan
<i>min</i>	0.3360	1.5756
<i>lower fence</i>	0.3360	1.5756
q1	0.4154	1.7064
q2 (median)	0.4824	1.8094
q3	0.5760	1.9782
<i>upper fence</i>	0.8069	2.3480
<i>max</i>	1.6659	3.6549

Tabel 3.5: Nilai-nilai perbandingan waktu Book Covers 400.

Dapat dilihat dari Gambar 3.29 serta dengan melihat nilainya pada Tabel 3.5 bahwa ada perbedaan sebaran nilai yang cukup besar antara *dataset* yang telah tersaring dan *dataset* keseluruhan. *Dataset* yang telah tersaring memiliki total waktu yang tersebar pada nilai-nilai yang lebih kecil dibandingkan dengan *dataset* keseluruhan.

Selain memengaruhi waktu yang diperlukan untuk mengidentifikasi gambar, penyaringan *dataset* dengan nilai konsistensi dan keunikan juga memengaruhi tingkat akurasi dari proses identifikasi.

- 1 Perbandingan nilai akurasi serta total waktu BSIS antara data yang tersaring dan data keseluruhan
 2 dapat dilihat pada Tabel 3.6.

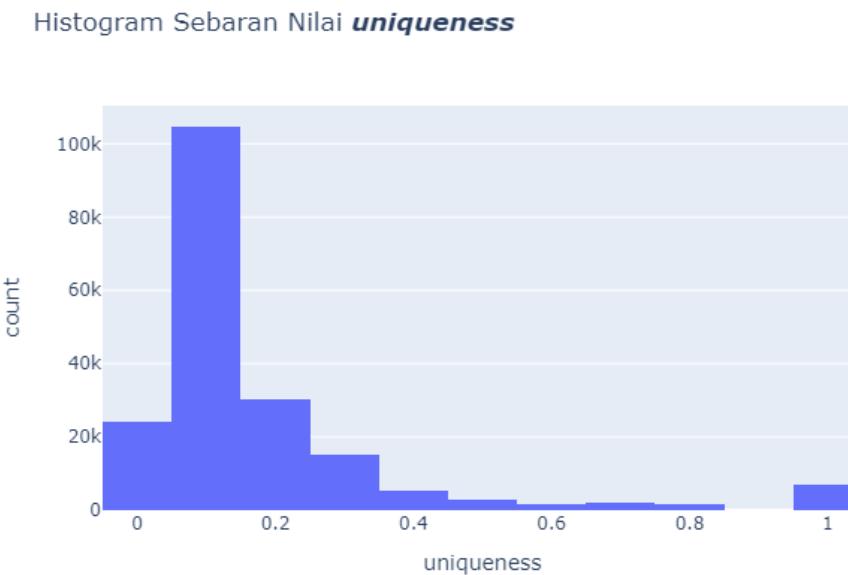
	Jumlah Fitur Lokal	Total Waktu (s)	Akurasi (%)
Keseluruhan	121909	190.85	99
Tersaring	24040	52.66	92

Tabel 3.6: Perbandingan total waktu dengan akurasi BSIS pada Book Covers 400.

- 3 Dari data pada Tabel 3.6 didapat beberapa poin berikut:
 4 • Ada penurunan sebanyak 72.40% pada total waktu dari *dataset* keseluruhan dan tersaring.
 5 • Akurasi menurun sebanyak 7% dari *dataset* keseluruhan dan tersaring.
 6 Terlihat bahwa pada pengujian dengan *dataset* ini penyaringan fitur lokal dengan menggunakan
 7 nilai konsistensi dan keunikan memberikan hasil yang cukup baik. Dengan menggunakan hanya
 8 sebagian dari *dataset* dapat meningkatkan waktu proses secara signifikan dengan tetap mendapat
 9 hasil akurasi yang tinggi.

10 Book Covers 600

- 11 Book Covers 600 memiliki total fitur lokal yang dihasilkan dari seluruh gambar sebanyak 195558
 12 fitur lokal. Setelah dilakukan *clustering* menghitung nilai konsistensi dan keunikan didapat hasil
 13 sebaran nilai konsistensi dan keunikan seperti pada Gambar 3.30 dan Gambar 3.31.



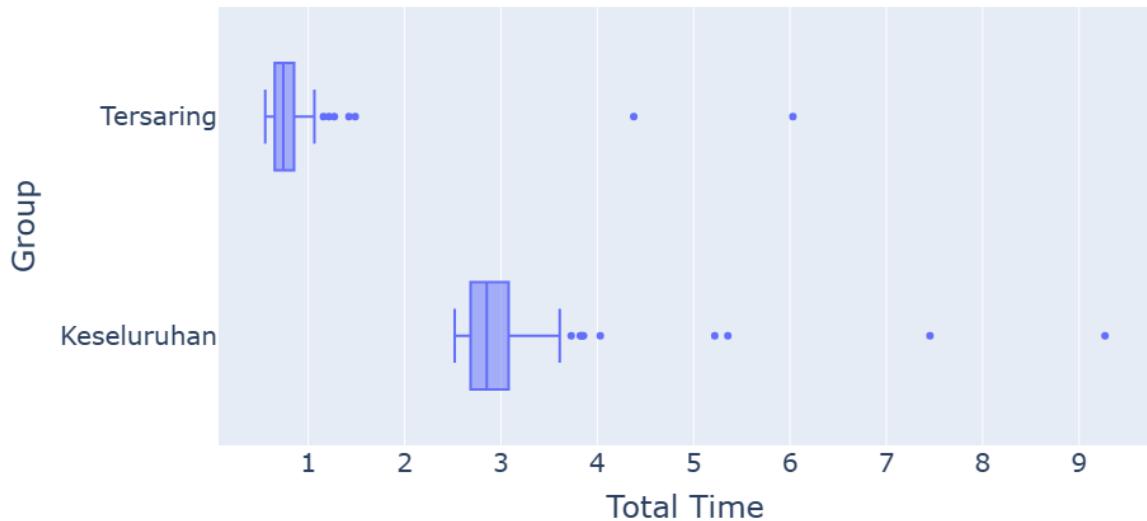
Gambar 3.30: Sebaran nilai keunikan (*uniqueness*) untuk Book Covers 600.



Gambar 3.31: Sebaran nilai konsistensi (*consistency*) untuk Book Covers 600.

1 Sebaran nilai konsistensi dan keunikan untuk Book Covers 600 memiliki ciri yang sama dengan
2 sebaran untuk Book Covers 400. Pada Book Covers 600 ini akan digunakan nilai batas yang sama
3 dengan Book Covers 400 untuk menyaring nilai konsistensi dan keunikan. Nilai-nilai batas yang
4 digunakan adalah 0.5 untuk konsistensi dan 0.2 untuk keunikan. Setelah dilakukan penyaringan
5 dengan menggunakan batas-batas tersebut didapat sebanyak 43056 data. *Dataset* keseluruhan dan
6 yang sudah tersaring tadi digunakan untuk melakukan identifikasi gambar-gambar di *dataset test*.
7 Perbedaan sebaran waktu yang digunakan untuk tiap gambar pada saat digunakan *dataset*
8 secara keseluruhan dan yang telah tersaring dapat dilihat pada Gambar 3.32.

Perbandingan Sebaran Waktu tiap Kelompok Dataset



Gambar 3.32: Perbandingan sebaran waktu Book Covers 600.

¹ Detail untuk kedua *boxplot* pada Gambar 3.32 tersebut dapat dilihat pada Tabel 3.7 berikut.

	Tersaring	Keseluruhan
<i>min</i>	0.5527	2.5186
<i>lower fence</i>	0.5527	2.5186
q1	0.6511	2.6836
q2 (median)	0.7413	2.8522
q3	0.8514	3.0799
<i>upper fence</i>	1.0632	3.6103
<i>max</i>	6.0301	9.2698

Tabel 3.7: Nilai-nilai perbandingan waktu Book Covers 600.

² Dapat dilihat dari Gambar 3.32 serta nilainya pada Tabel 3.7 bahwa ada perbedaan sebaran ³ nilai yang cukup besar antara *dataset* keseluruhan dan yang telah tersaring. Perbedaan ini sama ⁴ dengan yang ada pada saat digunakan Book Covers 400. Untuk pengaruh penyaringan nilai pada ⁵ akurasi BSIS dapat dilihat pada Tabel 3.8

	Jumlah Fitur Lokal	Total Waktu (s)	Akurasi (%)
Keseluruhan	195558	307.43	99
Tersaring	43056	87.37	93

Tabel 3.8: Perbandingan total waktu dengan akurasi BSIS pada Book Covers 600.

⁶ Dari data pada Tabel 3.8 didapat beberapa poin berikut:
⁷ • Ada penurunan sebanyak 71.58% pada total waktu dari *dataset* keseluruhan dan tersaring.
⁸ • Akurasi menurun sebanyak 6% dari *dataset* keseluruhan dan tersaring.
⁹ Sama seperti pengujian pada Book Covers 400, pada pengujian ini penyaringan *dataset* meningkatkan
¹⁰ kecepatan waktu proses dengan cukup signifikan. Sedikit berbeda dengan Book Covers 400, pada

- 1 penggunaan Book Covers 600 ini penurunan nilai akurasi lebih banyak. Nilai akurasi menurun
 2 sebanyak 6% dibandingkan pada Book Covers 400 yang hanya menurun 1%.

3 Perbandingan Book Covers 400 dan Book Covers 600

- 4 Percobaan yang dilakukan dengan menggunakan Book Covers 400 dan Book Covers 600 memiliki
 5 perbedaan yang cukup terlihat dari sisi akurasi yang dihasilkan dan waktu proses yang diperlukan.
 6 Perbandingan tersebut dapat dilihat pada Tabel 3.9.

	Keseluruhan			Tersaring		
	Jumlah Fitur Lokal	Total Waktu (s)	Akurasi (%)	Jumlah Fitur Lokal	Total Waktu (s)	Akurasi (%)
Book Covers 400	121909	190.85	99	24040	52.66	92
Book Covers 600	195558	307.43	99	43056	87.37	93

Tabel 3.9: Perbandingan Book Covers 400 dan Book Covers 600 pada analisis ini.

7 Tabel di atas menunjukkan bahwa penggunaan Book Covers 400 memerlukan waktu proses
 8 yang lebih cepat dibanding dengan Book Covers 600. Perbedaan waktu proses yang diperlukan ini
 9 disebabkan karena Book Covers 600 memiliki jumlah total data (fitur lokal) yang lebih banyak
 10 dibanding Book Covers 400.

11 Dari hasil akurasi tidak ada perbedaan yang cukup signifikan antara Book Covers 400 dan
 12 Book Covers 600. Pada penggunaan data keseluruhan Book Covers 400 dan Book Covers 600
 13 memiliki nilai akurasi yang sama. Pada penggunaan data yang telah tersaring nilai akurasi Book
 14 Covers 400 hanya lebih kecil sebanyak 1% dibandingkan dengan nilai akurasi Book Covers 600. Jika
 15 dibandingkan maka Book Covers 400 lebih baik karena memiliki waktu proses yang lebih cepat
 16 dengan tingkat akurasi yang hampir sama.

17 3.9 Analisis Metode Ekstraksi Fitur Lokal ORB

18 Salah satu metode ekstraksi fitur lokal selain SIFT adalah ORB. Seperti yang telah dijelaskan
 19 sebelumnya pada 2.4, metode ORB mencari *keypoint* dan fitur lokalnya dengan cara yang lebih
 20 sederhana dibandingkan dengan SIFT. Cara yang lebih sederhana tersebut membuat ORB dapat
 21 mencari *keypoint* dan melakukan ekstraksi fitur lokal dari gambar dengan lebih cepat. Walaupun
 22 fitur lokal yang dihasilkan ORB lebih sederhana, seharusnya fitur-fitur lokal tersebut tetap dapat
 23 digunakan untuk identifikasi gambar dengan tingkat akurasi yang tidak berbeda jauh dengan SIFT.

24 Pada analisis ini akan diuji bagaimana performa metode ORB jika dibandingkan dengan SIFT
 25 untuk melakukan identifikasi pada gambar. Analisis akan dilakukan untuk melihat perbedaan ORB
 26 dan SIFT dalam waktu yang diperlukan untuk melakukan ekstraksi fitur lokal dan akurasinya pada
 27 *dataset test* yang sama. Tahapan yang dilakukan sama dengan tahapan pada 3.8.4 dengan beberapa
 28 perbedaan sebagai berikut:

- 29 • Tidak membuat objek SIFT melainkan membuat objek ORB dengan fungsi (`cv2.ORB_create()`).
- 30 • Tahap *pairing* pada BSIS dilakukan dengan menggunakan struktur data LSH.

- 1 • Pengujian dilakukan hanya pada *dataset* keseluruhan, tidak pada *dataset* yang telah tersaring
2 seperti pada 3.8.
- 3 *Dataset train* yang digunakan pada analisis ini juga sama dengan analisis di 3.8. Ada dua variasi
4 *dataset* train yang digunakan Book Covers 400 dan Book Covers 600. *Dataset test* yang digunakan
5 juga sama seperti pada analisis sebelumnya.

6 3.9.1 Hasil Pengujian

7 Setelah dilakukan identifikasi dengan BSIS menggunakan fitur lokal ORB dihitung total waktu
8 ekstraksi fitur dari seluruh gambar *test* dan akurasinya. Nilai total waktu dan akurasi tersebut
9 kemudian dibandingkan dengan pada saat digunakan fitur lokal SIFT, yang merupakan hasil dari
10 analisis pada 3.8. Hasilnya untuk masing-masing Book Covers 400 dan Book Covers 600 dapat
11 dilihat pada Tabel 3.10 dan Tabel 3.11.

	Jumlah Fitur Lokal	Total Waktu Ekstrak (s)	Total Waktu BSIS (s)	Akurasi (%)
ORB	94281	0.24	135.63	86
SIFT	121909	1.78	190.85	99

Tabel 3.10: Perbandingan Waktu Proses dengan Akurasi Metode ORB dan SIFT pada Book Covers 400

	Jumlah Fitur Lokal	Total Waktu Ekstrak (s)	Total Waktu BSIS (s)	Akurasi (%)
ORB	99658	0.30	144.16	88
SIFT	195558	2.44	307.43	99

Tabel 3.11: Perbandingan Waktu Proses dengan Akurasi Metode ORB dan SIFT pada Book Covers 600

12 Terlihat dari kedua tabel di atas bahwa metode ORB memiliki total waktu ekstrak dan total
13 waktu BSIS secara keseluruhan yang lebih kecil. Walaupun begitu akurasi hasil identifikasi juga
14 menurun. Dari nilai-nilai pada kedua tabel didapat beberapa poin berikut.

15 Book Covers 400

- 16 • Dari penggunaan metode SIFT ke ORB terdapat penurunan total waktu ekstrak sebesar
17 86.51%, dan penurunan total waktu BSIS sebesar 28.93%.
- 18 • Metode ORB memiliki nilai akurasi yang lebih rendah sebesar 13% dari metode SIFT.

19 Book Covers 600

- 20 • Dari penggunaan metode SIFT ke ORB terdapat penurunan total waktu ekstrak sebesar
21 87.77%, dan penurunan total waktu BSIS sebesar 53.10%.
- 22 • Metode ORB memiliki nilai akurasi yang lebih rendah sebesar 11% dari metode SIFT.

23 Setelah didapat fitur lokal waktu proses yang diperlukan untuk BSIS dengan metode ORB
24 lebih cepat dari SIFT tetapi perbedaannya tidak sebanyak perbedaan waktu ekstraksi. Perbedaan
25 waktu ekstraksi pada Book Covers 400 dan Book Covers 600 lebih disebabkan oleh Perbedaan
26 yang ada pada waktu BSIS lebih disebabkan oleh perbedaan jumlah fitur lokal yang diproses pada

- 1 metode SIFT dan ORB. Metode SIFT menghasilkan lebih banyak fitur lokal dibanding dengan
- 2 ORB sehingga waktu BSIS metode SIFT lebih lambat.

1

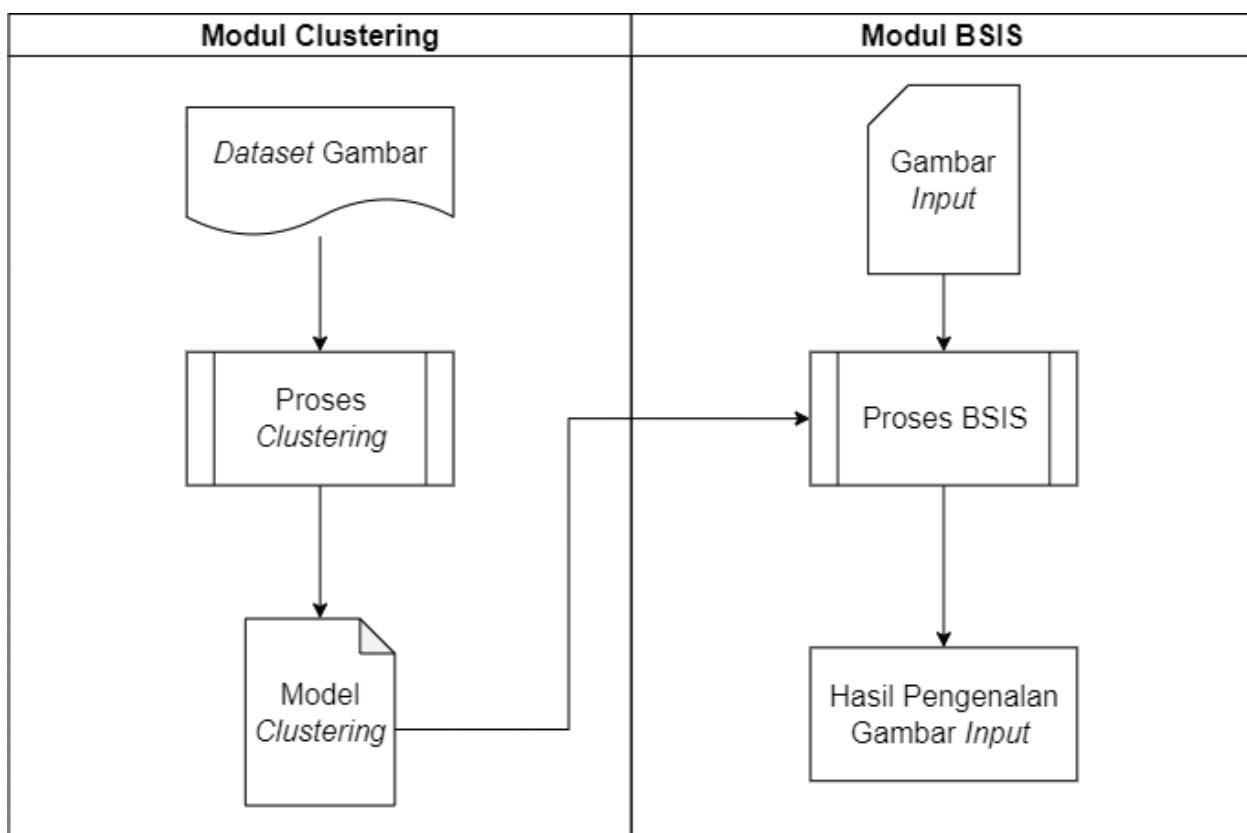
BAB 4

2

PERANCANGAN

3 4.1 Rancangan Alur Program

- 4 Secara garis besar program terbagi menjadi dua proses yang saling terhubung. Proses pertama
5 merupakan proses *clustering* untuk membuat model. Proses *clustering* menerima *dataset* dalam
6 bentuk *file* gambar, melakukan ekstraksi fitur lokal dari gambar-gambar tersebut dan membuat
7 model. Proses yang kedua adalah proses OIR yang dilakukan dengan menggunakan metode BSIS.
8 Proses kedua ini menerima *file* model yang dihasilkan oleh proses *clustering* dan menggunakan
9 model tersebut untuk dapat melakukan pengenalan terhadap gambar masukan. Rancangan alur
10 program ini dapat dilihat pada diagram di Gambar 4.1.



Gambar 4.1: Rancangan alur program pada penelitian ini.

- 11 Berdasarkan dari diagram pada Gambar 4.1 tersebut program akan terbagi menjadi 2 modul.
12 Modul pertama merupakan modul *clustering* yang implementasinya dijelaskan pada 4.2. Modul

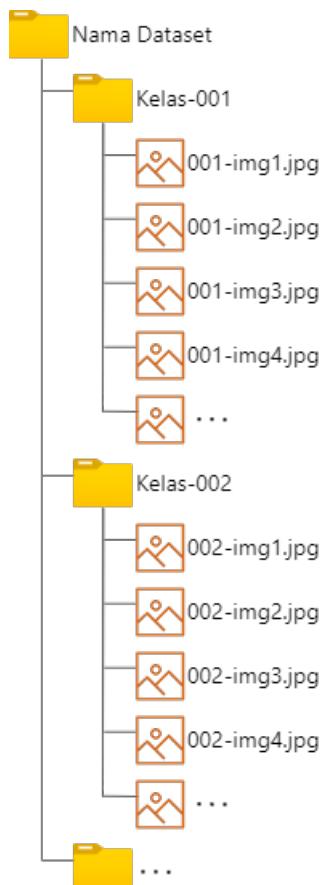
- 1 yang kedua adalah modul BSIS yang implementasinya dijelaskan pada [4.5](#). Selain itu untuk model
 2 yang dihasilkan oleh proses *clustering* dan digunakan untuk proses BSIS akan dijelaskan pada [4.3](#).

3 4.2 Rancangan Implementasi Metode Clustering Pembuatan Model

- 4 Metode *clustering* diimplementasi dalam sebuah modul Python yang menerima *file* dalam bentuk
 5 gambar dan menghasilkan sebuah *dataframe* yang berisi detail-detail fitur lokal dari gambar.
 6 Gambar-gambar masukan yang digunakan perlu untuk memiliki sebuah struktur *folder* tertentu.
 7 Keterangan struktur *folder* masukan serta fungsi-fungsi dalam modul hingga struktur *file* keluaran
 8 akan dijelaskan pada subbab-subbab berikut ini.

9 4.2.1 Rancangan Struktur Folder

- 10 Rancangan struktur *folder* yang diperlukan untuk dapat digunakan sebagai masukan pada modul
 11 *clustering* adalah sebagai berikut, dapat dilihat pada Gambar [4.2](#).

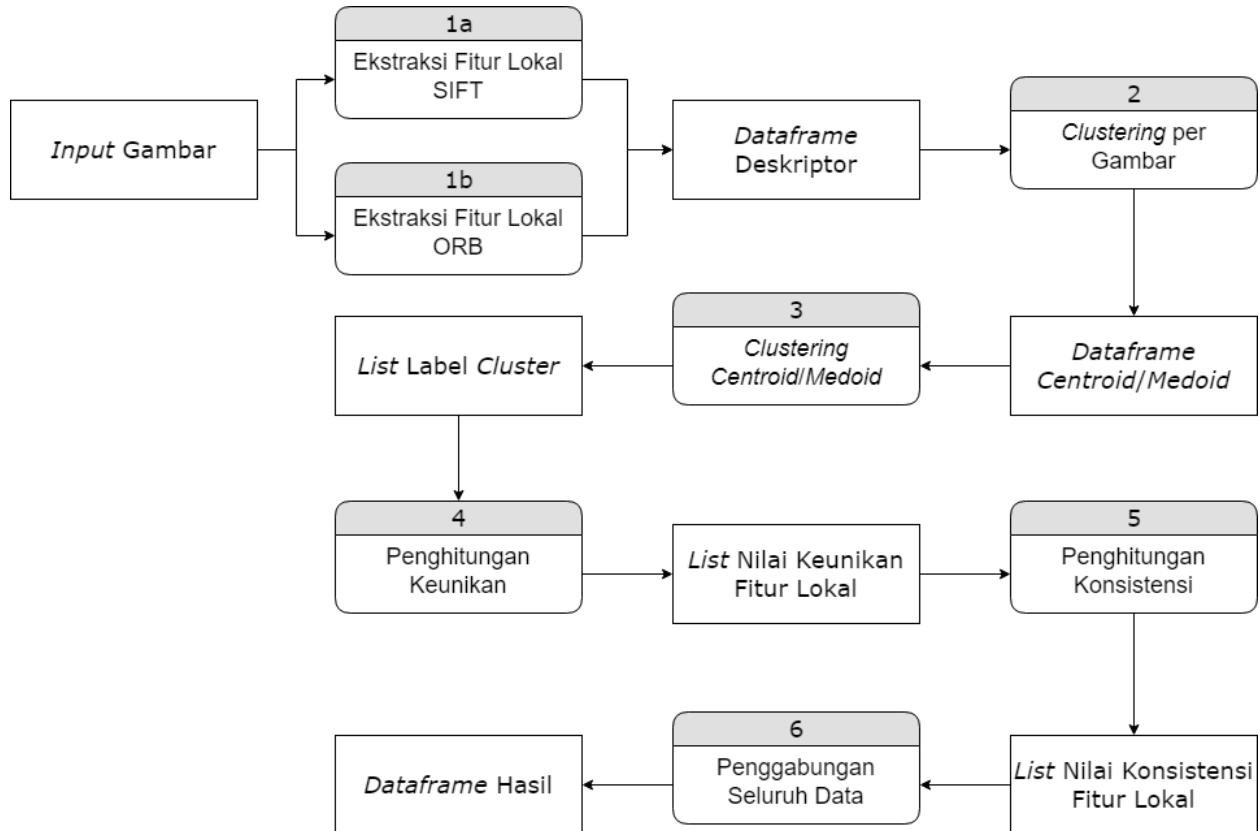


Gambar 4.2: Rancangan struktur *folder* untuk pembuatan model.

- 12 Gambar tersebut menunjukkan struktur *folder* masukan yang diperlukan pada modul ini. *Folder*
 13 yang paling atas merupakan *folder* utama yang juga menjadi nama *dataset*. *Folder* utama akan
 14 berisi beberapa *subfolder* yang menunjukkan kelas dari gambar. Setiap *subfolder* ini berisi beberapa
 15 *file* gambar yang merupakan bagian dari kelas tersebut. Gambar-gambar *dataset* yang sudah dalam
 16 struktur seperti ini dapat digunakan dalam proses *clustering*

1 4.2.2 Rancangan Proses Clustering

2 Proses *clustering* menerima *dataset* gambar sesuai dengan struktur *folder* yang telah dijelaskan
 3 sebelumnya pada 4.2.1. Dari *dataset* tersebut diproses dan dihasilkan *dataframe* yang berisi deskriptor dari setiap fitur lokal serta nilai keunikan dan konsistensi. Langkah-langkah yang dilakukan
 5 proses ini serta hasil keluaran dari setiap langkah dapat dilihat pada diagram di Gambar 4.3.



Gambar 4.3: Tahapan proses *clustering* hingga didapat nilai keunikan dan konsistensi.

6 Proses dalam Modul Clustering ini dapat dilakukan dengan menggunakan dua tipe metode
 7 ekstraksi fitur lokal yaitu SIFT dan ORB. SIFT dan ORB memiliki bentuk dan isi deskriptor
 8 fitur lokal yang berbeda, di mana deskriptor SIFT merupakan vektor bilangan bulat sedangkan
 9 deskriptor ORB adalah vektor bilangan biner. Perbedaan bentuk dan isi vektor deskriptor tersebut
 10 menyebabkan perlunya dibuat implementasi yang berbeda untuk SIFT dan ORB. Beberapa
 11 perbedaan implementasi untuk SIFT dan ORB adalah sebagai berikut:

- 12 • Di langkah 2 dan 3 *clustering* untuk SIFT dilakukan dengan menggunakan metrik jarak
 13 Euclidean, sedangkan untuk ORB metrik jarak yang digunakan adalah Hamming.
 - 14 • Di langkah 2 saat menggunakan SIFT dihitung *centroid* untuk setiap hasil *clustering* tiap
 15 gambar, sedangkan pada ORB yang dihitung adalah *medoid*.
- 16 Selain dari perbedaan-perbedaan tersebut langkah implementasi yang dilakukan sama, baik untuk
 17 metode ekstraksi fitur lokal SIFT maupun ORB. Proses penghitungan nilai keunikan dapat dilihat
 18 pada Pseudocode 2.

Pseudocode 2: UNIQUENESS

Input:

- D : *dataframe* yang memiliki kolom berisi nama gambar (`img`), kelompok gambar (`img_class`), dan label *cluster* (`cluster_label`).

Output: list berisi nilai keunikan

```

1: buat dictionary class_count
2: kelompokan  $D$  berdasarkan kolom cluster_label.
3: for semua cluster_label  $c$  kelompok cluster_label do
4:   buat variabel  $o$  yang berisi semua elemen  $D$  yang memiliki cluster_label  $c$ 
5:   hapus baris dalam  $o$  yang nilai kolom img-nya duplikat sehingga untuk setiap nilai img yang
   berbeda hanya muncul satu kali.
6:   hitung jumlah setiap img_class yang ada dalam  $o$  dan bagi nilainya dengan jumlah elemen
   dalam  $o$ 
7:   masukkan nilai penghitungan jumlah tiap img_class ke dalam sebuah dictionary dua
   tingkat dengan key pertama adalah  $c$  dan key kedua adalah isi img_class.
8: end for
9: buat list uniqueness
10: for baris  $r$  dalam  $D$  do
11:   ambil nilai keunikan dari class_count dengan menggunakan cluster_label dan
      img_class sebagai key
12:   masukkan nilai keunikan tersebut ke uniqueness.
13: end for
14: return uniqueness

```

² Sedangkan untuk penghitungan nilai konsistensi mengikuti proses pada Pseudocode 3.

Pseudocode 3: CONSISTENCY

Input:

- D : *dataframe* yang memiliki kolom berisi nama gambar (`img`), kelompok gambar (`img_class`), dan label *cluster* (`cluster_label`).

Output: *list* berisi nilai konsistensi

```

1: kelompokan  $D$  berdasarkan cluster_label dan img_class.
2: hitung jumlah gambar yang unik untuk tiap kelompok. Masukkan hasilnya ke dalam dictionary
   dua tingkat  $d$  yang memiliki key pertama adalah isi dari cluster_label dan key keduanya
   adalah isi dari img_class.
3: buat list consistency
4: for baris  $r$  dalam  $D$  do
5:   buat variabel  $v$  dengan mengambil nilai  $d$  menggunakan cluster_label dan img_class
      dari  $r$  sebagai key.
6:   bagi nilai  $v$  dengan jumlah gambar pada dataset yang kelasnya sama dengan img_class dari
       $r$ .
7:   masukan  $v$  ke consistency
8: end for
9: return consistency

```

- 1 Proses yang dilakukan pada Gambar 4.3 akan menghasilkan sebuah *dataframe* dengan rincian
 2 kolom seperti pada Tabel 4.1 untuk metode SIFT, untuk metode ORB ada perbedaan pada kelompok
 3 kolom *Descriptor*.

Nama Kolom	Kelompok Kolom	Deskripsi
0	<i>Descriptor</i>	Berjumlah 128 kolom yang masing-masing merupakan satu elemen dalam vektor deskriptor SIFT.
1		
...		
126		
127	Informasi Gambar	Nama gambar asal fitur lokal Kelas dari gambar asal fitur lokal
img		
img_class		
cluster_label		
cluster2_label	Label Cluster	Label <i>cluster</i> dari hasil <i>clustering</i> per gambar Label <i>cluster</i> dari hasil <i>clustering centroid</i>
uniqueness		
consistency		
point_x		
point_y	Nilai Hasil Penghitungan <i>Atribut keypoint</i>	Koordinat x dari <i>keypoint</i> Koordinat y dari <i>keypoint</i> Diameter daerah di sekitar <i>keypoint</i> yang diperiksa Orientasi dari <i>keypoint</i> Tingkat respon yang menyatakan seberapa kuat <i>keypoint</i> Oktaf di mana <i>keypoint</i> tersebut didapat Kelas objek yang menyatakan kelompok dari <i>keypoint</i> jika dikelompokkan
size		
angle		
response		
octave		
class_id		

Tabel 4.1: Rincian kolom dari *dataframe* yang dihasilkan modul proses *clustering* SIFT.

4 Rancangan proses *clustering* ini dibuat ke dalam dua buah *script* Python dengan nama
 5 *clustering_sift.py* dan *clustering_orb.py*. Kedua *script* tersebut berguna untuk melakukan
 6 proses *clustering* sesuai dengan metode ekstraksi fitur lokalnya SIFT atau ORB. Baik *clustering_sift.py*
 7 dan *clustering_orb.py* dapat dijalankan dengan menerima dua argumen berikut:

- 8 • **dir**: lokasi tempat *dataset* gambar yang akan dibuat modelnya. Direktori harus memiliki
 9 struktur *folder* yang sesuai dengan 4.2.1.
 10 • **maxsize**: ukuran sisi maksimum dari gambar *dataset*. Gambar akan diperkecil jika ukuran
 11 panjang atau lebarnya melebihi nilai ini.

12 Contoh *command* untuk menjalankan *script* proses *clustering* adalah sebagai berikut:

13 python clustering_sift.py --dir datasets/gsv --maxsize 600

14 *Script* tersebut akan menghasilkan sebuah *file* yang dapat dibaca oleh kelas *ClusterModel*.

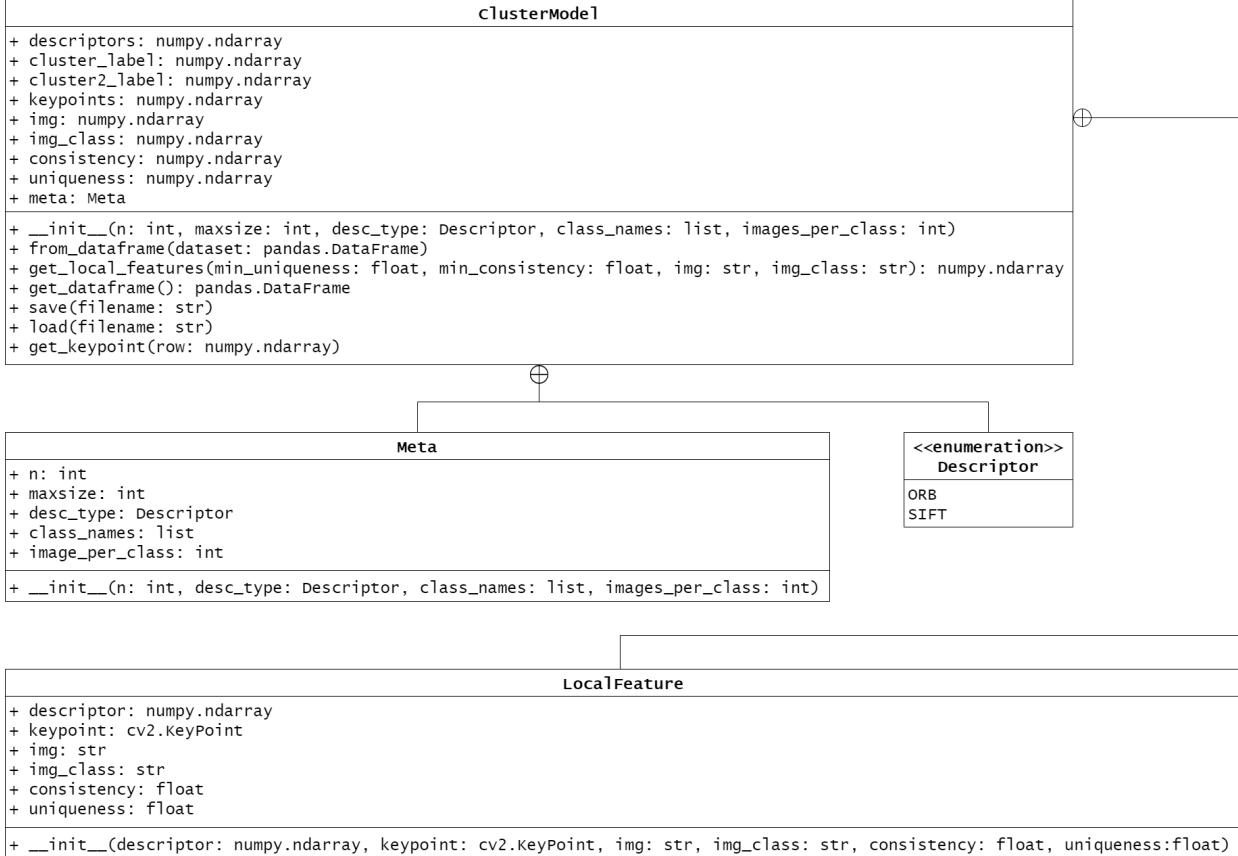
15 Kelas *ClusterModel* sendiri akan dijelaskan pada subbab berikut ini.

4.3 Rancangan Kelas ClusterModel

Kelas `ClusterModel` adalah kelas yang berguna untuk menyimpan dan memproses model yang dihasilkan oleh proses *clustering* untuk digunakan pada BSIS. Kelas ini menyimpan data-data fitur lokal serta *keypoint* dan nilai-nilai yang didapat dari hasil *clustering*. Kelas memiliki *method* untuk menyimpan model kedalam sebuah *file* dan juga *method* untuk membaca model dari *file*.

Kelas ini juga berguna sebagai perantara antara proses *clustering* dan BSIS. Kelas memiliki *imethod* untuk menghasilkan fitur-fitur lokal yang sudah disaring berdasarkan nilai konsistensi dan keunikan. Fitur-fitur lokal yang dihasilkan ini dalam format yang dapat dibaca oleh kelas BSIS.

Diagram kelas `ClusterModel` beserta *inner-class*-nya dapat dilihat pada Gambar 4.4.



Gambar 4.4: Diagram kelas `ClusterModel`.

Kelas ini terdiri dari kelas `ClusterModel` dan tiga *inner-class* yang masing-masing memiliki fungsi sebagai berikut:

- **Meta**: menyimpan informasi dasar tentang model yang dihasilkan. Informasi yang disimpan meliputi jumlah data, ukuran maksimum gambar, tipe deskriptor, nama-nama kelas gambar, dan jumlah gambar tiap kelas.
- **Descriptor**: merupakan enumerasi yang menyimpan dua tipe deskriptor yang dapat digunakan. Tipe deskriptor ini akan berpengaruh pada format data masukan saat membuat objek `ClusterModel`.
- **LocalFeature**: menyimpan keluaran fungsi `get_local_features` dari `ClusterModel` dengan format tertentu. Format ini merupakan format yang digunakan sebagai data masukan pada

- 1 kelas BSIS.
- 2 Kelas `ClusterModel` sendiri memiliki beberapa fungsi penting sebagai berikut:
- 3 • `from_dataframe`: fungsi yang digunakan untuk membuat objek `ClusterModel` dari `dataframe`
4 yang berisi kolom-kolom seperti pada Tabel 4.1.
- 5 • `get_local_features`: fungsi untuk mengembalikan sebuah *list* berisi objek `LocalFeature`.
6 Fungsi ini digunakan untuk menghasilkan masukkan yang dapat digunakan oleh kelas BSIS.
- 7 • `get_dataframe`: fungsi untuk mengembalikan data yang disimpan dalam objek `ClusterModel`
8 dalam bentuk `dataframe`. Fungsi ini berguna untuk melakukan analisis pada hasil proses
9 *clustering* yang telah tersimpan sebagai objek `ClusterModel`.

10 4.4 Rancangan Kelas Util

- 11 Kelas *Util* berisi fungsi-fungsi pembantu yang digunakan pada kelas-kelas lainnya. Fungsi-fungsi
12 ini berguna untuk mempermudah proses komputasi saat menggunakan kelas lainnya dan saat
13 melakukan analisis. Fungsi-fungsi dalam kelas *Util* dapat dilihat pada diagram di Gambar 4.5.

Util
<pre>+ get_image(filename: str, bw: bool, maxwidth: int, maxheight: int): numpy.array + get_all_image(directory: str, bw: bool, maxwidth: int, maxheight: int): list + get_dataset(directory: str, bw: bool, maxwidth: int, maxheight: int): dict + agglo_cluster(dataset: pandas.DataFrame, n_clusters: int, distance_threshold: float, affinity: str): numpy.array + dbscan(dataset: pandas.DataFrame, eps: float, min_pts: int, metric: str): numpy.array + euclidean_dist(point1: numpy.array, point2: numpy.array): float + hamming_bin(point1: numpy.array, point2: numpy.array): int + hamming_bin_affinity(X: numpy.ndarray): numpy.ndarray + combination_distance(dataset: pandas.DataFrame, metric: function, sample: int): float + int_to_binary(desc: numpy.ndarray): numpy.ndarray + binary_to_int(desc: numpy.ndarray): numpy.ndarray + medoid(arr: numpy.array, metric: function, get_index: bool) + rotate(p: tuple, origin: tuple, degrees: int): tuple + show_keypoints(image: numpy.array, keypoints: tuple, color: tuple, flags: int, name: str) + show_matches(img1: numpy.array, kp1: tuple, img2: numpy.array, kp2: tuple, name: str) + show_polygon(img: numpy.array, kp: tuple, name: str) + rotate_image(image: numpy.array, angle: int): numpy.array + zoom_center(image: numpy.array, zoom_factor: float): numpy.array</pre>

Gambar 4.5: Diagram kelas Util

- 14 Beberapa fungsi penting dari kelas `Util` adalah sebagai berikut:

- 15 1. `get_image`
16 Fungsi untuk memuat gambar dari *file*. Parameter `bw` menyatakan apakah gambar yang
17 dimuat akan diubah menjadi format *grayscale*. Sedangkan untuk `maxwidth` dan `maxheight`
18 digunakan untuk mengatur ukuran dari gambar.
- 19 2. `get_all_image`
20 Fungsi untuk memuat semua gambar dalam suatu *folder*. Fungsi ini memanggil fungsi
21 `get_image` untuk semua *file* gambar dalam *folder* di *directory*.
- 22 3. `get_dataset`
23 Fungsi untuk memuat semua *file* gambar dengan fungsi `get_image` dari *file* yang tersusun sesuai
24 struktur *folder* seperti yang dijelaskan pada 4.2.1. Fungsi ini digunakan untuk mendapatkan
25 gambar masukkan pada 4.2.2.
- 26 4. `agglo_cluster`
27 Fungsi untuk melakukan *Agglomerative Clustering* pada *dataset*. Fungsi akan mengembalikan
28 sebuah *array* yang berisi label *cluster*. Fungsi ini digunakan untuk melakukan *clustering*

per gambar dan *clustering centroid* pada analisis yang dilakukan di [3.6](#) dengan mengikuti rancangan pada [4.2.2](#).

5. combination_distance

Fungsi untuk menghitung jarak rata-rata antar baris baris dalam sebuah *array*. Fungsi ini digunakan untuk mendapatkan nilai *threshold* pada saat dilakukan *clustering* di [4.2](#).

6. show_keypoints

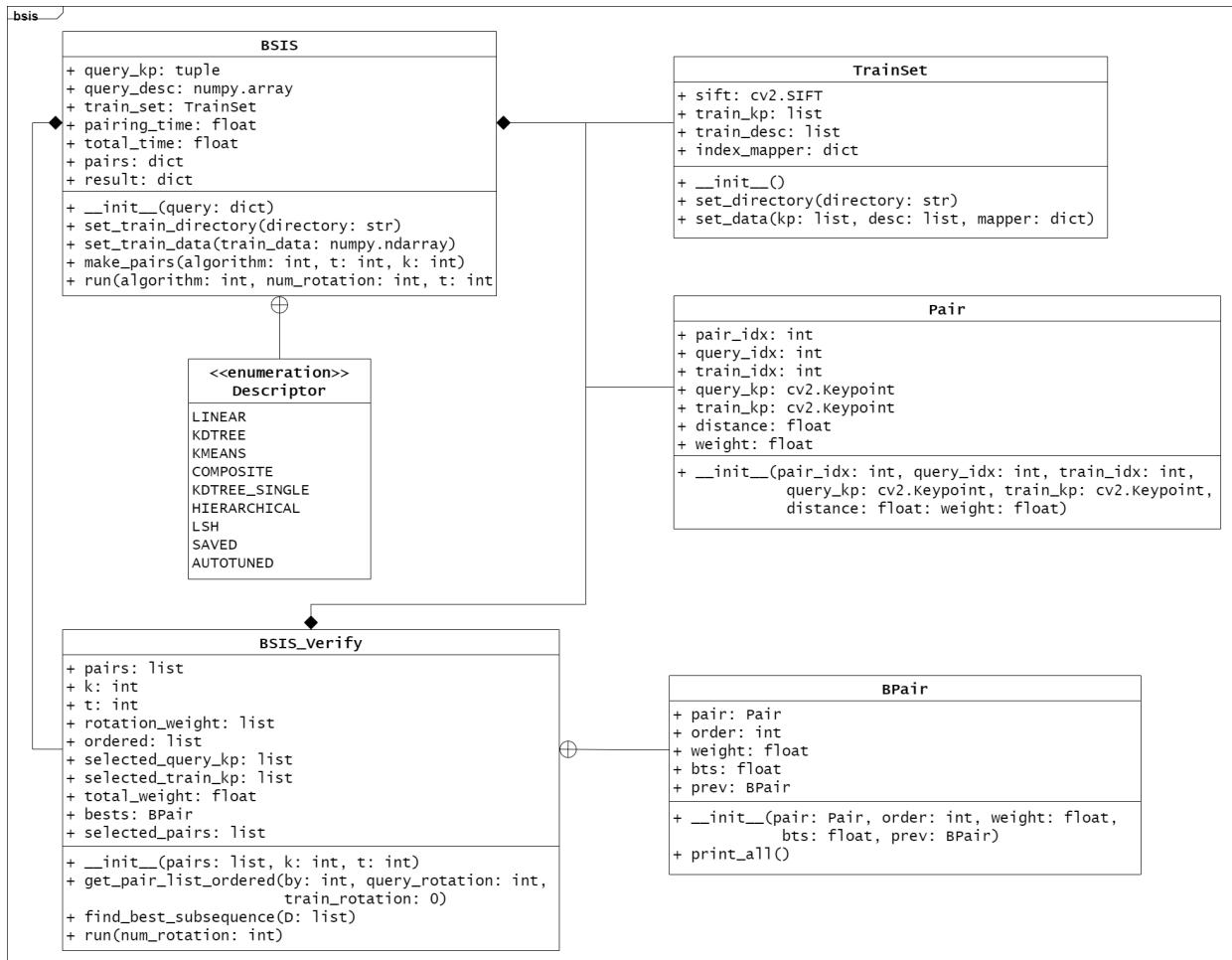
Fungsi untuk menampilkan *keypoint* pada gambar. Fungsi digunakan untuk melakukan visualisasi *keypoint* di gambar yang dilakukan pada [3.6](#).

7. show_matches

Fungsi untuk menampilkan pasangan *keypoint* dari dua gambar. Fungsi ini digunakan untuk menampilkan pasangan *keypoint* yang kuat pada analisis di [3.1](#).

4.5 Rancangan Kelas-kelas Implementasi BSIS

Implementasi BSIS dibuat dalam sebuah *package* yang didalamnya berisi beberapa kelas yang saling berhubungan. Kelas-kelas beserta fungsi dalam kelas yang ada di *package* BSIS dapat dilihat pada Gambar [4.6](#). Kelas-kelas pada *package* ini digunakan untuk melakukan BSIS pada [3.8](#).



Gambar 4.6: Diagram *Package* `bsis`.

Kelas BSIS merupakan *entry point* dari *package* ini. Untuk melakukan identifikasi gambar

1 dengan BSIS perlu dibuat sebuah objek BSIS. Objek BSIS akan memanggil kelas-kelas lainnya saat
2 dijalankan. Fungsi dari tiap-tiap kelas dalam *package* BSIS adalah sebagai berikut:

3 1. **TrainSet**

4 Kelas **TrainSet** berguna untuk menyimpan data *train* sebuah objek BSIS. Kelas ini me-
5 nyimpan *list keypoint* dan *list descriptor* dalam atribut **train_kp** dan **train_desc**. Atribut
6 **index_mapper** merupakan sebuah *dictionary* yang menyimpan nama gambar asal untuk tiap
7 *keypoint*.

8 2. **Pair**

9 Kelas **Pair** berfungsi untuk menyimpan objek pasangan *keypoint* gambar *query* dan *keypoint*
10 gambar *train* di BSIS. Kelas ini menyimpan nomor indeks (**query_idx** dan **train_idx**) serta
11 objek *keypoint* dari *keypoint* gambar *query* (**query_kp**) dan *train* (**train_kp**). Kelas juga
12 menyimpan nilai jarak dari vektor *descriptor* kedua *keypoint* (**distance**) dan nilai bobot
13 pasangan (**weight**).

14 3. **BSIS_Verify**

15 Kelas **BSIS_Verify** berguna untuk melakukan tahap verifikasi geometris pada BSIS. Kelas me-
16 nerima parameter berupa *list* berisi objek **Pair**. Kelas **BSIS_Verify** ini akan membuat sebuah
17 *jagged array* berisi objek **BPair** yang diimplementasikan sebagai *list* di dalam *list* melalui fung-
18 si **get_pair_list_ordered**. Dari *jagged array* yang dihasilkan tersebut akan dicari sebuah
19 *sequence* yang konsisten secara geometris dan total bobotnya paling tinggi menggunakan fung-
20 si **find_best_subsequence**. Proses yang dilakukan dalam fungsi **find_best_subsequence**
21 dapat dilihat pada Pseudocode 4.

22 4. **BPair**

23 Kelas dengan struktur *Linked List* yang berguna untuk menyimpan sebuah *sequence* objek
24 *Pair*.

25 Langkah-langkah untuk melakukan identifikasi gambar menggunakan BSIS setelah dibuat model
26 *ClusterModel* dan dilakukan ekstraksi fitur lokal pada gambar *query* adalah sebagai berikut:

27 1. Buat objek **BSIS** dengan memasukkan *dictionary* berisi *list keypoint* dan *list descriptor* dari
28 gambar *query* sebagai parameter.

29 2. Panggil fungsi **set_train_data** pada objek **BSIS** untuk mengisi data *train*. Isi fungsi tersebut
30 dengan *list* yang diperoleh dari fungsi **get_local_features** milik *ClusterModel*.

31 3. Panggil fungsi **run** pada objek **BSIS** dengan memasukkan parameter berikut:

- 32 • **num_rotation**: jumlah rotasi gambar *query*.
- 33 • **algorithm**: algoritma yang digunakan untuk mencari pasangan-pasangan terdekat.
- 34 • **k**: jumlah maksimum pasangan terdekat yang ingin dicari untuk tiap *keypoint* di gambar
35 *query*.
- 36 • **t**: menentukan kelipatan standar deviasi yang digunakan untuk menghitung *threshold*
37 dalam tahap *pairing* BSIS.

38 Fungsi **run** pada objek **BSIS** akan mengembalikan nama gambar yang memiliki nilai total bobot
39 paling tinggi setelah melalui semua tahapan BSIS. Selain hasil berupa nama gambar, objek **BSIS**
40 yang telah dipanggil fungsi **run**-nya akan memiliki sebuah atribut bernama **result**. Atribut **result**
41 merupakan sebuah *dictionary* yang memiliki nama gambar sebagai *key*, dan *value*-nya merupakan
42 sebuah *dictionary* dengan *key* serta *value*-nya sebagai berikut:

-
- 1 • *query_kp*: *list* berisi *keypoint* pada gambar *query*.
 2 • *train_kp*: *list* berisi *keypoint* pada gambar *train*.
 3 • *total_weight*: nilai total bobot pasangan gambar.

Pseudocode 4: FIND_BEST_SEQUENCE

Input:

- *D*: sebuah *list* dengan setiap elemennya merupakan *list* berisi objek *BPair*, dapat memiliki panjang yang berbeda-beda.

Output: *sequence BPair* yang merupakan pasangan yang konsisten secara geometris dan nilai total bobotnya paling tinggi

```

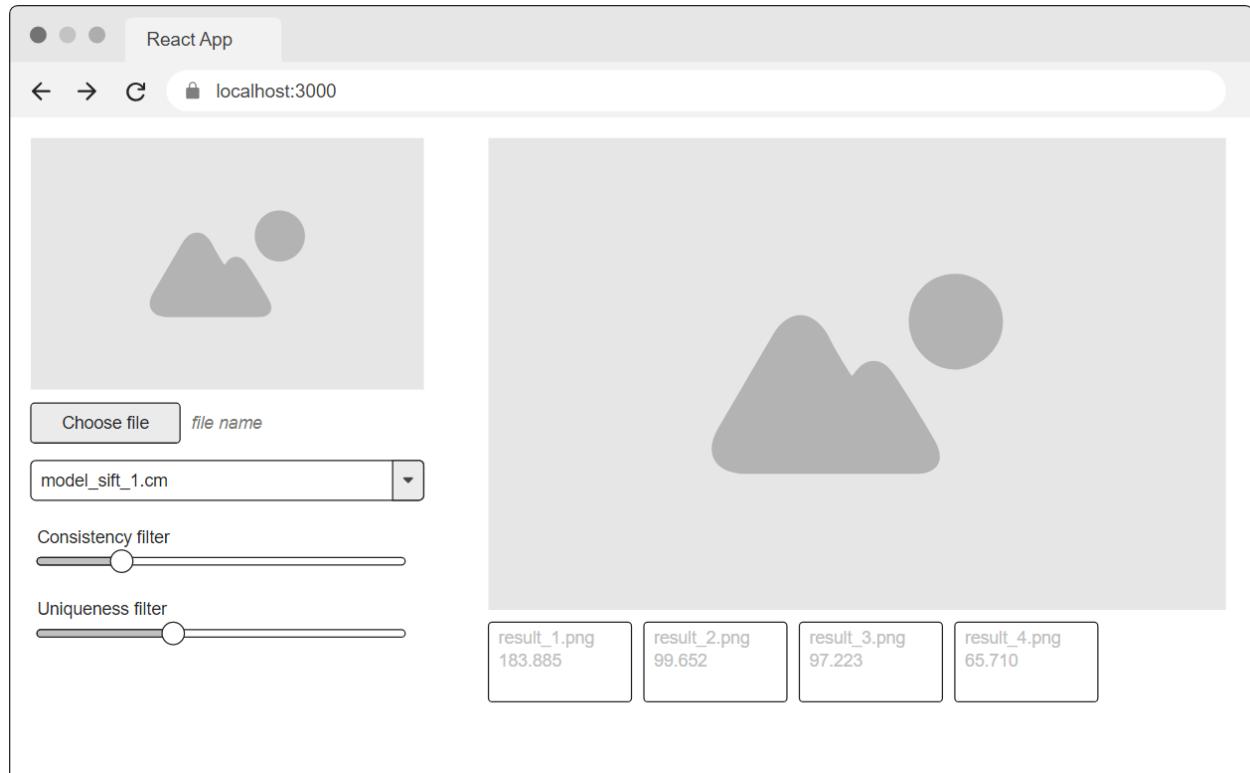
1: buat objek BPair best_subsequence
2: for i=1 hingga LENGTH(D) do
3:   for j=1 hingga LENGTH(D[i]) do
4:     buat objek BPair dBestPrev
5:     for k=1 hingga i do
6:       for l=1 hingga LENGTH(D[k]) do
7:         if ORDER(D[k][l]) < ORDER(D[i][j]) dan BTS(D[k][l]) > BTS(dBestPrev) then
8:           isi dBestPrev dengan
9:             D[k][l]
10:          end if
11:        end for
12:      end for
13:      isi BTS(D[i][j]) dengan WEIGHT(D[i][j]) + BTS(dBestPrev)
14:      isi PREV(D[i][j]) dengan dBestPrev
15:      if BTS(D[i][j]) + WEIGHT(D[i][j]) > BTS(best_subsequence) +
WEIGHT(best_subsequence) then
16:        isi best_subsequence dengan D[i][j]
17:      end if
18:    end for
19:  end for
20: return best_subsequence

```

5 4.6 Rancangan Perangkat Lunak BSIS

- 6 Untuk lebih mudah melakukan BSIS pada sebuah gambar masukkan, dibuat sebuah perangkat
 7 lunak dengan GUI yang dapat melakukan identifikasi gambar masukkan dengan BSIS. Perangkat
 8 lunak yang dibuat memiliki fitur sebagai berikut:
 9 • Memilih model yang ingin digunakan untuk identifikasi gambar masukkan.
 10 • Memilih gambar masukkan yang ingin diidentifikasi.
 11 • Menentukan nilai *threshold* untuk konsistensi dan keunikan.
 12 • Melakukan identifikasi pada gambar yang telah dipilih, dengan menggunakan model serta
threshold yang telah ditentukan.
 13 Saat dilakukan identifikasi pada gambar, perangkat lunak akan melakukan BSIS pada gambar

- 1 masukkan dengan menggunakan *dataset* berdasarkan model dan nilai *threshold* yang telah dipilih.
- 2 Perangkat lunak lalu akan menampilkan hasil beberapa pasangan yang didapat diurutkan berdasarkan total bobotnya. Hasil yang dikembalikan berupa nama gambar, total bobot, dan gambar yang menunjukkan pasangan *keypoint*-nya. Untuk memenuhi fungsionalitas tersebut dibikin perangkat lunak dengan rancangan tampilan seperti yang dapat dilihat pada Gambar 4.7



Gambar 4.7: Rancangan tampilan perangkat lunak untuk BSIS.

1

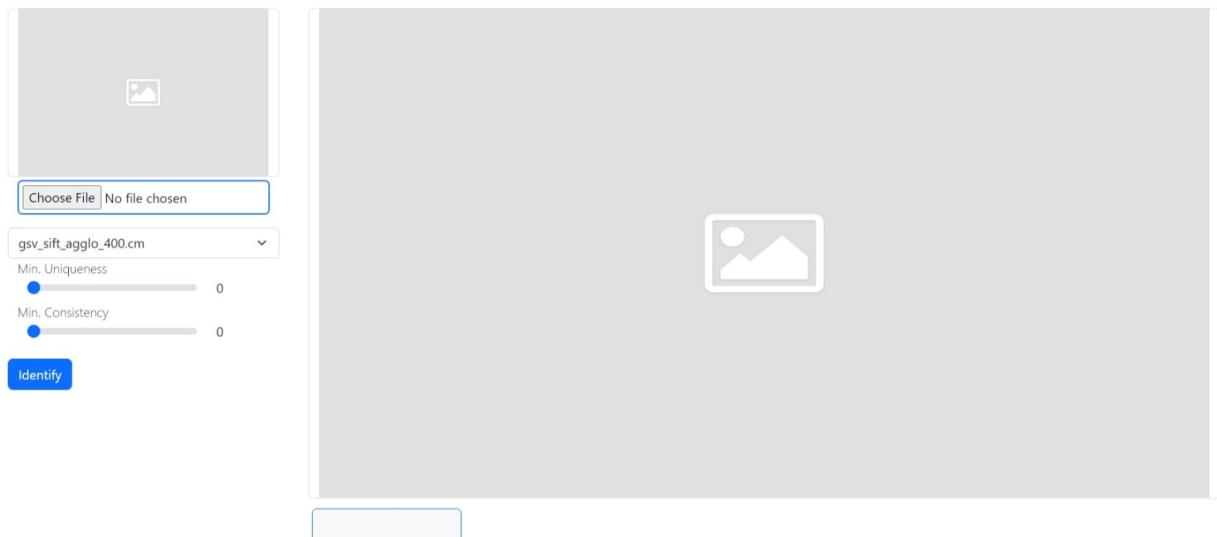
BAB 5

2

PENGUJIAN

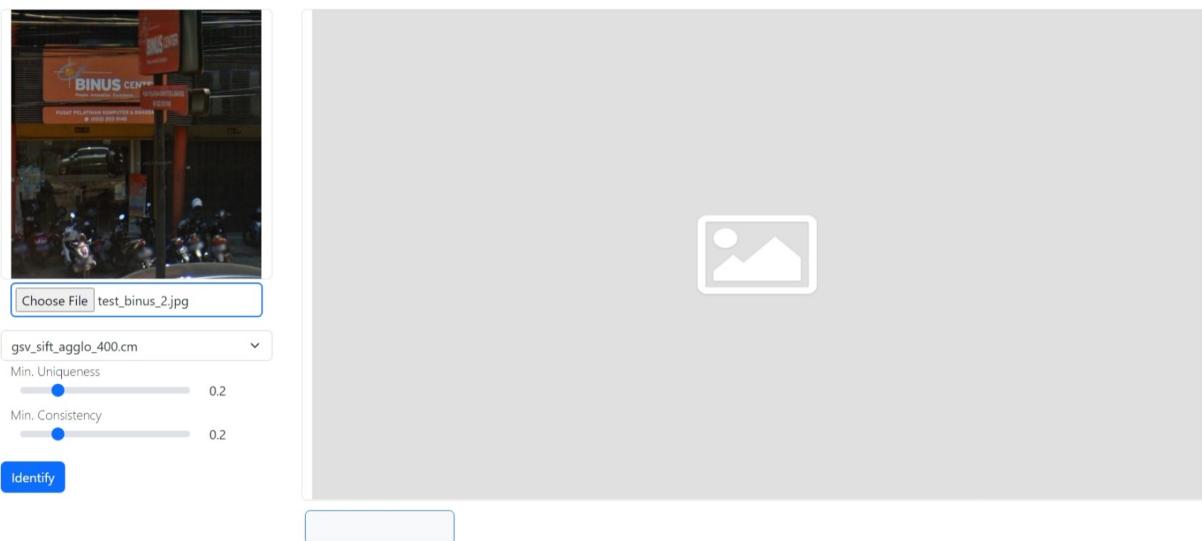
3 5.1 Pengujian Perangkat Lunak BSIS

- 4 Berdasarkan rancangan pada 4.6 telah dibuat sebuah perangkat lunak yang dapat melakukan
5 identifikasi dari suatu gambar masukkan dengan menggunakan teknik BSIS. Perangkat lunak yang
6 dibuat berbentuk aplikasi berbasis *web* yang terbagi ke *server* dan *client*. Bagian *server* dibuat
7 dengan menggunakan *framework* Flask di Python. *Server* akan menerima *request* HTTP dari *client*
8 dan melakukan identifikasi pada gambar masukkan dengan menggunakan parameter-parameter
9 yang diberikan. Bagian *client* dibuat dengan bentuk aplikasi *web* berbasis React. Tampilan halaman
10 *client* saat aplikasi pertama dijalankan dapat dilihat pada Gambar 5.1.



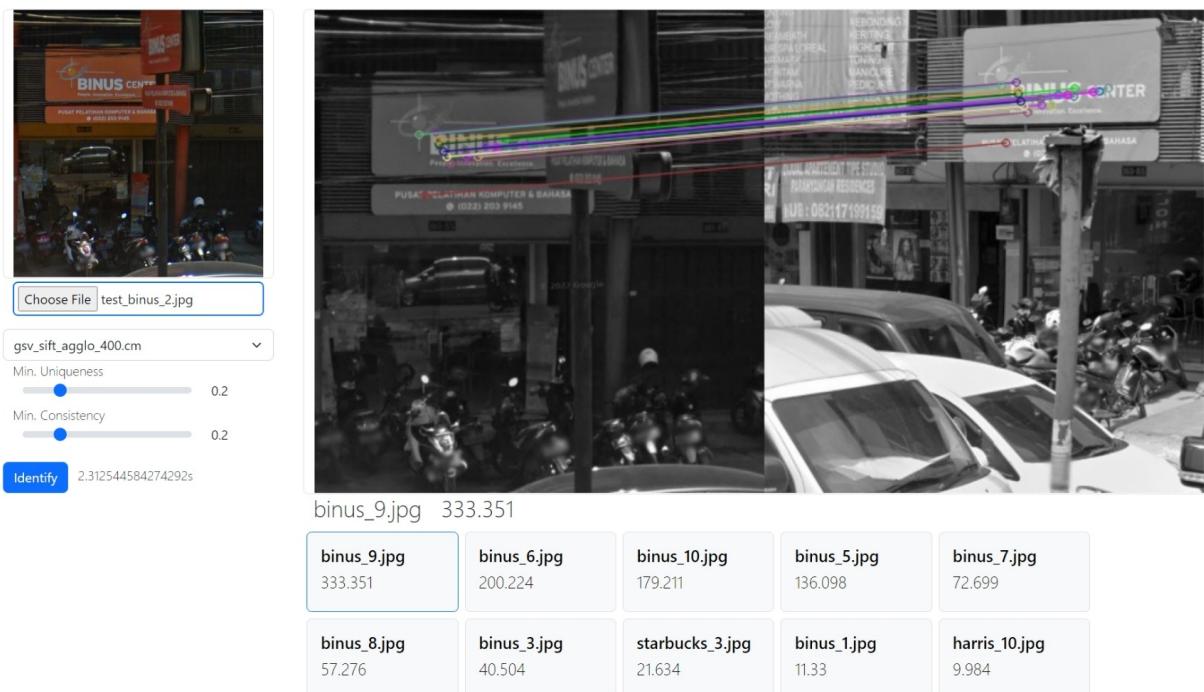
Gambar 5.1: Tampilan awal aplikasi perangkat lunak BSIS.

- 11 Pada bagian kiri halaman terdapat panel untuk *input*. Bagian paling atas merupakan tempat
12 untuk memilih gambar *input*, saat tombol “*Choose file*” ditekan akan membuka sebuah *window*
13 baru untuk memilih *file* dari *directory* lokal. Setelah dipilih gambar akan ditampilkan pada bagian
14 kotak gambar. Tampilan aplikasi saat telah memilih gambar dapat dilihat pada Gambar 5.2.



Gambar 5.2: Tampilan aplikasi perangkat lunak BSIS setelah memilih gambar *input*.

- 1 Bagian di bawah tombol “Choose file” merupakan sebuah *dropdown* yang berisi daftar model
- 2 *ClusterModel* (lihat 4.3) yang dapat digunakan. Di bawah *dropdown* pemilihan model terdapat
- 3 dua *slider* yang berguna untuk menentukan *threshold* untuk nilai konsistensi dan nilai keunikan.
- 4 Setelah dipilih gambar *input* dan model yang akan digunakan serta telah ditentukan *threshold*
- 5 untuk nilai konsistensi dan keunikan maka dapat dilakukan identifikasi dengan menekan tombol
- 6 “*Identify*” di bawah *slider* nilai keunikan. Setelah ditekan tombol “*Identify*” maka di bagian kanan
- 7 akan ditampilkan hasil pemasangan *keypoint* dari gambar *input* dengan gambar di *dataset* yang
- 8 memiliki nilai total bobot dari BSIS paling tinggi. Tampilan aplikasi setelah dilakukan identifikasi
- 9 pada gambar *input* dapat dilihat pada Gambar 5.3.



Gambar 5.3: Tampilan aplikasi perangkat lunak BSIS melakukan identifikasi pada gambar *input*.

- 1 Pada bagian kanan halaman aplikasi—seperti yang dapat dilihat pada Gambar 5.3—terlihat
- 2 gambar yang menunjukkan pemasangan *keypoint* dari gambar *input* dengan sebuah gambar *dataset*.
- 3 Pada bagian bawah gambar pemasangan *keypoint* terdapat teks yang menunjukkan nama gambar
- 4 dan disebelahnya merupakan total bobot pasangan gambar tersebut. Di bawah teks terdapat sebuah
- 5 *list* yang menunjukkan 10 gambar dari *dataset* yang memiliki nilai total bobot BSIS tertinggi. Setiap
- 6 *item* dalam *list* dapat ditekan untuk menampilkan pasangan *keypoint* dengan gambar tersebut.

7 5.2 Pengujian Metode Clustering untuk Identifikasi dengan BSIS

- 8 Pada bab ini akan dilakukan pengujian untuk melihat efek penerapan metode *clustering* untuk
- 9 menyaring fitur lokal terhadap ketepatan dan waktu proses BSIS. Pengujian yang dilakukan ini
- 10 akan menggunakan cara yang sama dengan yang ada pada 3.8 tetapi dengan menggunakan *dataset*
- 11 dan parameter *threshold* yang berbeda. Pada pengujian ini akan digunakan *dataset* GSV (3.3.2).

12 5.2.1 Ide Analisis

- 13 Seperti yang telah dijelaskan sebelumnya, pengujian yang dilakukan pada bagian ini akan mengikuti
- 14 alur yang telah dilakukan sebelumnya pada 3.8. Pengujian ini bertujuan untuk menguji bagaimana
- 15 penyaringan fitur lokal berdasarkan nilai konsistensi dan keunikannya dapat berpengaruh
- 16 terhadap hasil saat dilakukan identifikasi terhadap gambar. Pengaruh hasil identifikasi yang diamati merupakan bagaimana tingkat akurasi dan waktu proses yang diperlukan untuk melakukan
- 17 identifikasi.

19 Pada pengujian ini penyaringan fitur lokal akan dilakukan menggunakan dua cara. Penyaringan

20 pertama dilakukan dengan menggunakan nilai *threshold* yang didapat dari hasil analisis di 3.7.

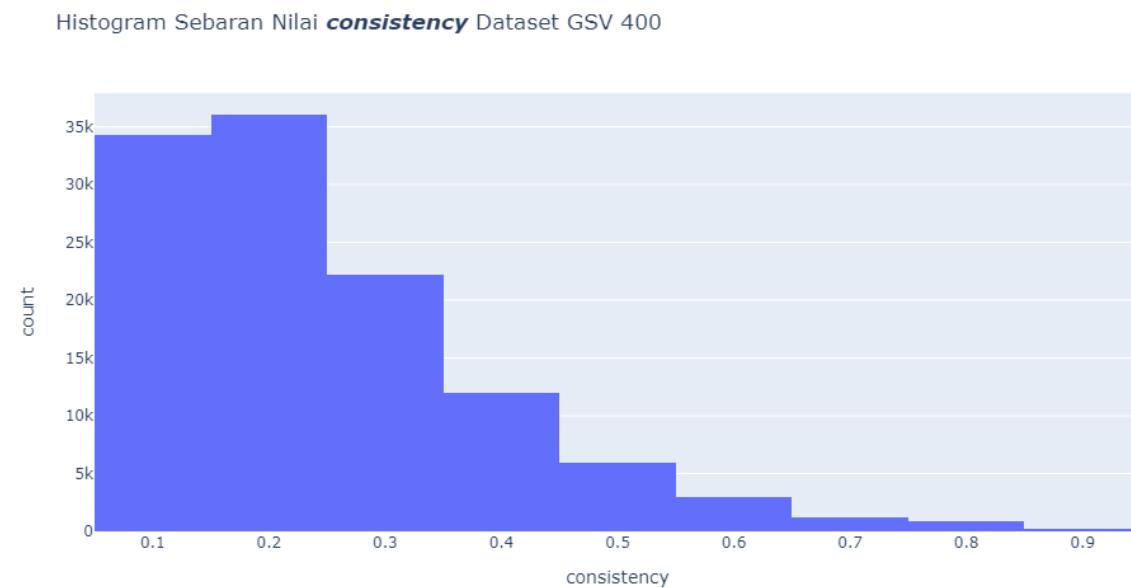
1 Berdasarkan pada hasil analisis di [3.7](#), penyaringan pertama akan dilakukan dengan menggunakan
2 nilai *threshold* 0.6 untuk konsistensi dan 0.7 untuk keunikan. Penyaringan kedua ditujukan untuk
3 menggunakan nilai *threshold* yang lebih rendah sehingga dapat digunakan sebagai perbandingan.
4 Untuk penyaringan kedua akan ditentukan *threshold* dengan cara yang sama dengan penentuan
5 *threshold* pada analisis di [3.7](#).

6 Pengujian ini akan dilakukan menggunakan *dataset* GSV dengan dua ukuran gambar yang
7 berbeda sama seperti pada analisis di [3.8](#). Dua ukuran gambar yang digunakan akan dinamakan
8 GSV 400 dan GSV 600, sesuai ukuran sisi terpanjang pada gambar-gambar yang digunakan. Berbeda
9 dengan analisis pada [3.8](#), pada pengujian ini ukuran dari gambar di data *test* akan disesuaikan
10 dengan ukuran gambar yang digunakan pada *dataset*. Pengujian juga akan dilakukan dengan
11 menggunakan dua metode ekstraksi fitur lokal yaitu SIFT dan ORB.

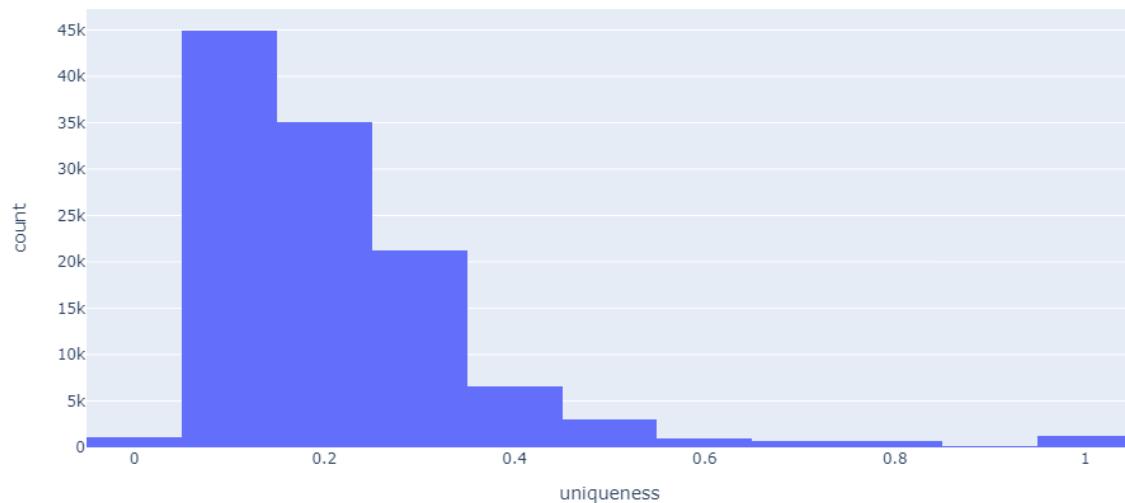
12 **5.2.2 Penentuan Threshold dan Hasil Analisis Metode SIFT**

13 **GSV 400**

14 Sebaran nilai konsistensi dan keunikan untuk *dataset* GSV 400 dapat dilihat pada Gambar [5.4](#) dan
15 Gambar [5.5](#).



Gambar 5.4: Histogram sebaran nilai keunikan pada *dataset* GSV 400.

Histogram Sebaran Nilai ***uniqueness*** Dataset GSV 400Gambar 5.5: Histogram sebaran nilai keunikan *dataset* GSV 400.

- Dengan melihat sebaran nilai konsistensi dan keunikan pada Gambar 5.4 dan Gambar 5.5 maka akan digunakan nilai *threshold* senilai 0.3 untuk konsistensi dan 0.3 untuk keunikan. Kedua nilai tersebut didapat dengan cara yang sama dengan yang dilakukan pada 3.8.
Pengujian pada *dataset* GSV 400 akan dilakukan dengan menggunakan 3 set *threshold* yang berbeda. Ketiga set *threshold* yang digunakan dapat dilihat pada Tabel 5.1.

	Konsistensi	Keunikan
Keseluruhan	0.0	0.0
Threshold 1	0.3	0.3
Threshold 2	0.6	0.7

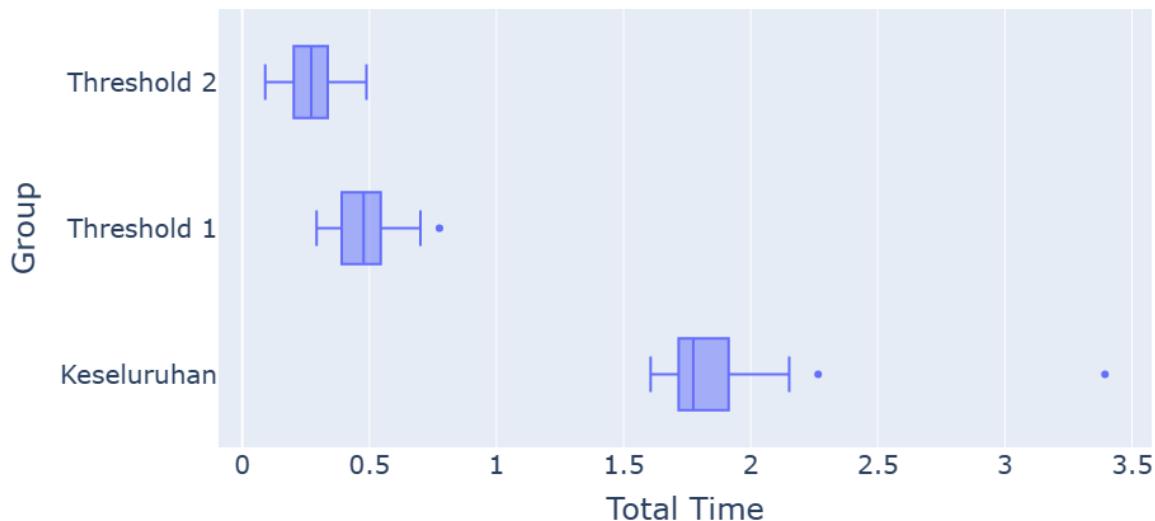
Tabel 5.1: Tiga set *threshold* yang digunakan untuk metode SIFT *dataset* GSV 400.

- Hasil dari pengujian dapat dilihat pada Tabel 5.2. Untuk sebaran waktu bagi tiap nilai *threshold* yang digunakan dapat dilihat pada *boxplot* di Gambar 5.6.

	Total Waktu Ekstraksi (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	1.29	92.55	94
Threshold 1	1.23	24.09	84
Threshold 2	1.23	13.97	12

Tabel 5.2: Hasil pengujian pada *dataset* GSV 400.

Perbandingan Sebaran Waktu tiap Kelompok Dataset



Gambar 5.6: Sebaran waktu pengujian metode SIFT *dataset GSV 400*.

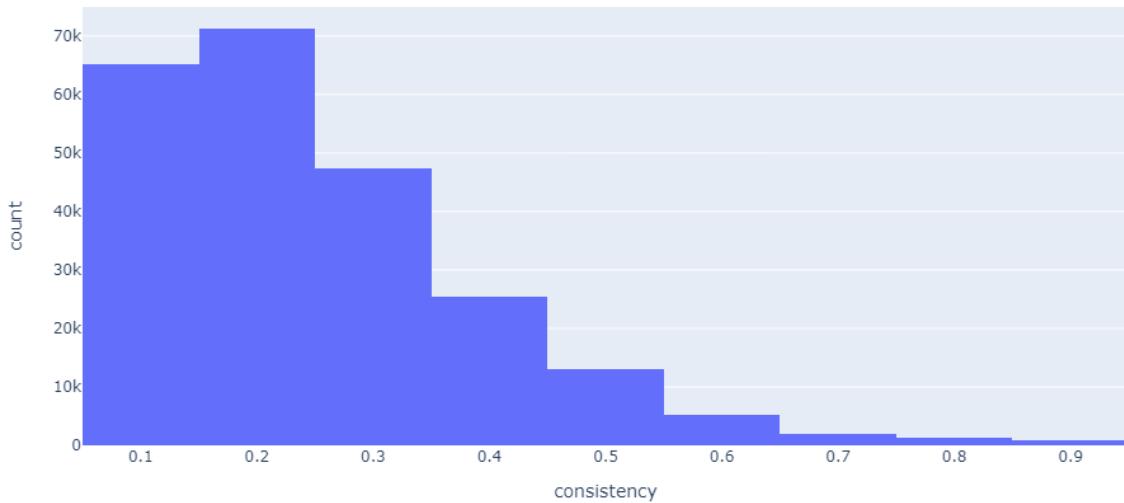
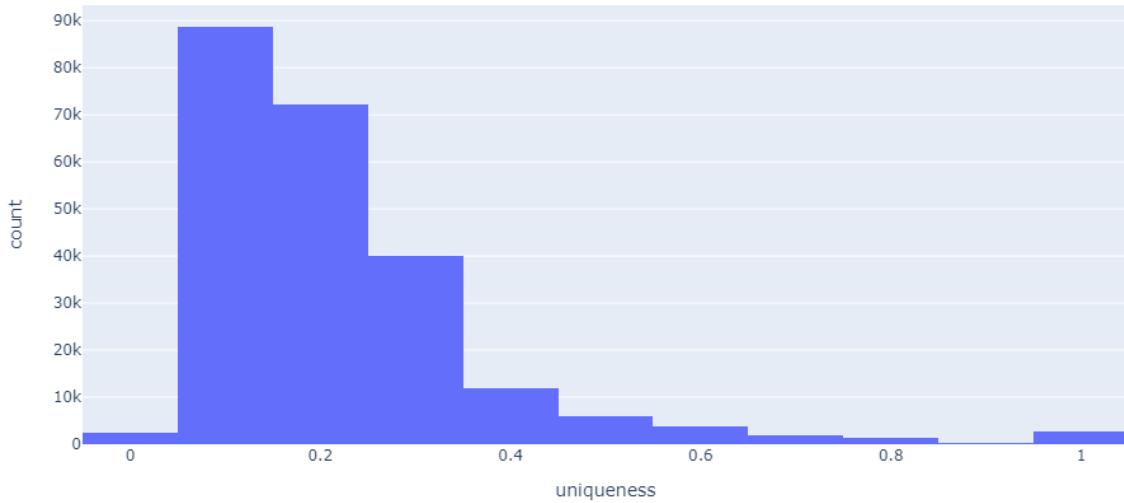
Dari *boxplot* pada Gambar 5.6 terlihat bahwa terdapat perbedaan sebaran waktu yang cukup signifikan antara *dataset* keseluruhan dan *dataset* yang tersaring. Perbedaan cukup tinggi terlihat di antara *dataset* Keseluruhan dan hasil penyaringan dengan Threshold 1, sedangkan untuk perbedaan antara penyaringan dengan Threshold 1 dan Threshold 2 perbedaannya tidak terlalu signifikan. Rincian nilai untuk ketiga *boxplot* pada Gambar 5.9 dapat dilihat pada Tabel 5.3 berikut.

	Keseluruhan	Threshold 1	Threshold 2
<i>min</i>	1.6063	0.2920	0.0902
<i>lower fence</i>	1.6063	0.2920	0.0902
q1	1.1716	0.3915	0.2020
q2 (median)	1.7741	0.4778	0.2714
q3	1.9132	0.5442	0.3365
<i>upper fence</i>	2.1514	0.7007	0.4890
<i>max</i>	3.3937	0.7756	0.4890

Tabel 5.3: Nilai-nilai perbandingan waktu metode SIFT *dataset GSV 400*.

6 GSV 600

Sebaran nilai konsistensi dan keunikan untuk *dataset* GSV 600 dapat dilihat pada Gambar 5.7 dan Gambar 5.8.

Histogram Sebaran Nilai **consistency** Dataset GSV 600Gambar 5.7: Histogram sebaran nilai keunikan pada *dataset* GSV 600.Histogram Sebaran Nilai **uniqueness** Dataset GSV 600Gambar 5.8: Histogram sebaran nilai keunikan *dataset* GSV 600.

Dengan menggunakan cara yang sama seperti sebelumnya, berdasarkan sebaran yang dapat dilihat pada Gambar 5.7 dan Gambar 5.8 maka akan digunakan nilai *threshold* senilai 0.3 untuk konsistensi dan 0.3 untuk keunikan. Tiga set *threshold* yang digunakan pada pengujian *dataset* ini dapat dilihat pada Tabel 5.4.

	Konsistensi	Keunikan
Keseluruhan	0.0	0.0
Threshold 1	0.3	0.3
Threshold 2	0.6	0.7

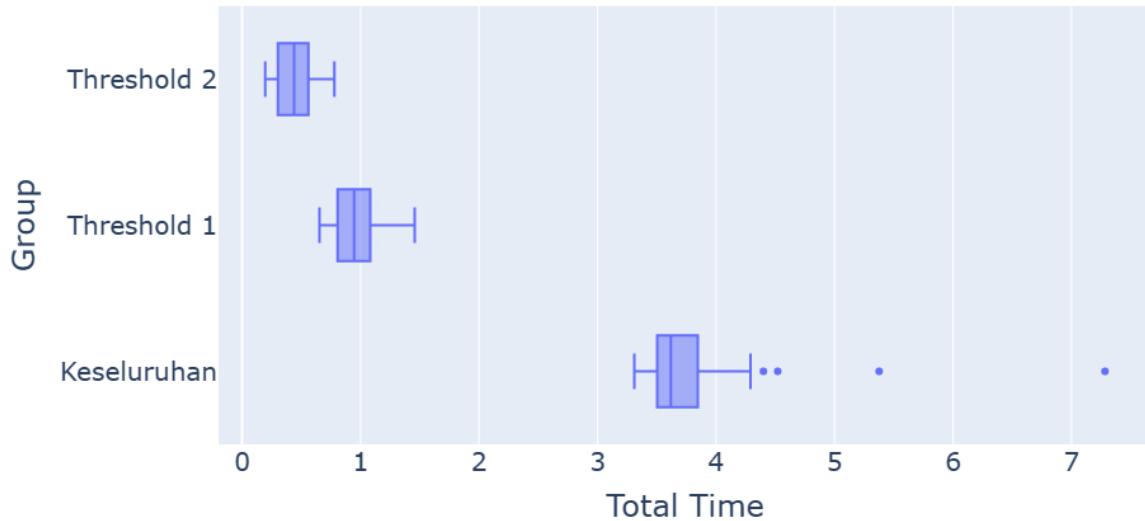
Tabel 5.4: Tiga set *threshold* yang digunakan untuk metode SIFT *dataset* GSV 600.

- 1 Setelah dilakukan pengujian terhadap *dataset* GSV 600 dengan menggunakan tiga set *threshold*
 2 tersebut didapat hasil yang dapat dilihat pada Tabel 5.5. Sebaran waktu untuk tiap *threshold* yang
 3 digunakan dapat dilihat secara rinci pada *boxplot* di Gambar 5.9.

	Total Waktu Ekstraksi (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	2.50	189.54	100
Threshold 1	2.65	47.95	84
Threshold 2	2.39	22.04	10

Tabel 5.5: Hasil pengujian pada *dataset* GSV 600.

Perbandingan Sebaran Waktu tiap Kelompok Dataset



Gambar 5.9: Sebaran waktu pengujian metode SIFT *dataset* GSV 600.

- 4 Tiga buah *boxplot* pada Gambar 5.9 menunjukkan perbedaan sebaran waktu antar tiga jenis
 5 *dataset*. Sebaran waktu di ketiga jenis *dataset* tersebut relatif sama dengan sebaran pada saat
 6 digunakan *dataset* GSV 400 (Gambar 5.6). Terdapat perbedaan yang cukup signifikan antara *dataset*
 7 keseluruhan dan yang telah tersaring dengan Threshold 1, tetapi untuk penyaringan Threshold 1
 8 dan Threshold 2 perbedaannya tidak signifikan. Rincian nilai untuk ketiga *boxplot* pada Gambar 5.9
 9 dapat dilihat pada Tabel 5.6 berikut.

	Keseluruhan	Threshold 1	Threshold 2
<i>min</i>	3.3089	0.6535	0.1954
<i>lower fence</i>	3.3089	0.6535	0.1954
q1	3.5025	0.8086	0.3019
q2 (median)	3.6173	0.9444	0.4393
q3	3.8465	1.0810	0.5591
<i>upper fence</i>	4.2913	1.4579	0.7784
<i>max</i>	7.2820	1.4579	0.7784

Tabel 5.6: Nilai-nilai perbandingan waktu metode SIFT *dataset* GSV 600.

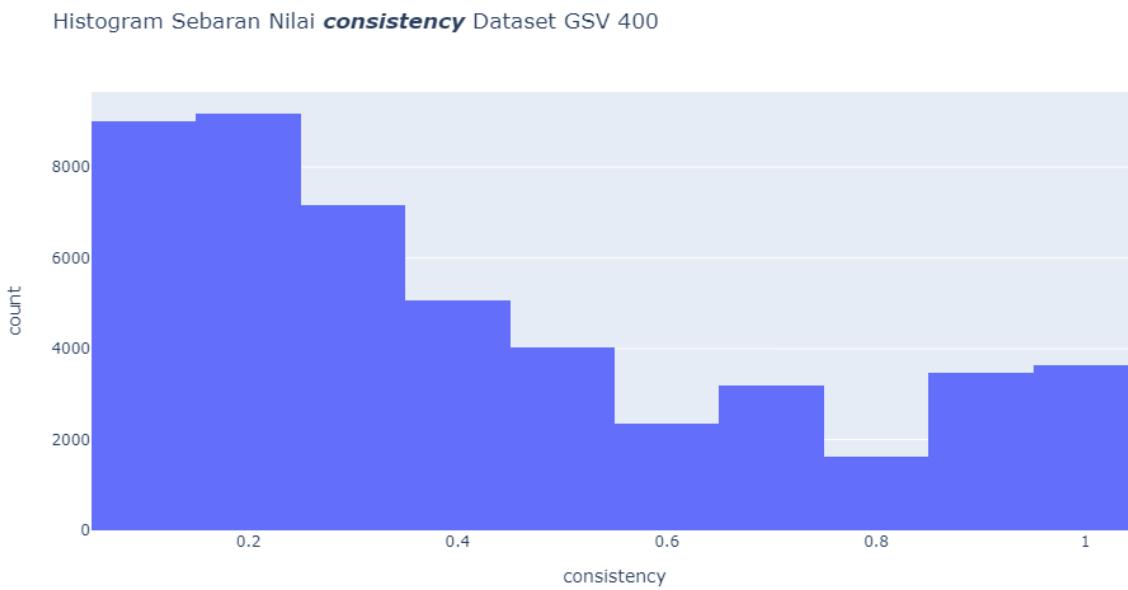
Hasil pada Tabel 5.2 dan Tabel 5.5 menunjukkan nilai *threshold* yang didapat dari hasil analisis pada *threshold* (3.7) memberikan hasil BSIS dengan akurasi yang sangat rendah. Cara penentuan *threshold* dengan melihat hasil pada gambar tersebut ternyata tidak efektif jika digunakan untuk melakukan identifikasi. Hal ini mungkin dikarenakan dalam melakukan identifikasi gambar banyak fitur lokal yang berasal bukan dari bagian yang merupakan logo atau objek penting, tetapi sebenarnya berguna dalam identifikasi.

5.2.3 Penentuan Threshold dan Hasil Analisis Metode ORB

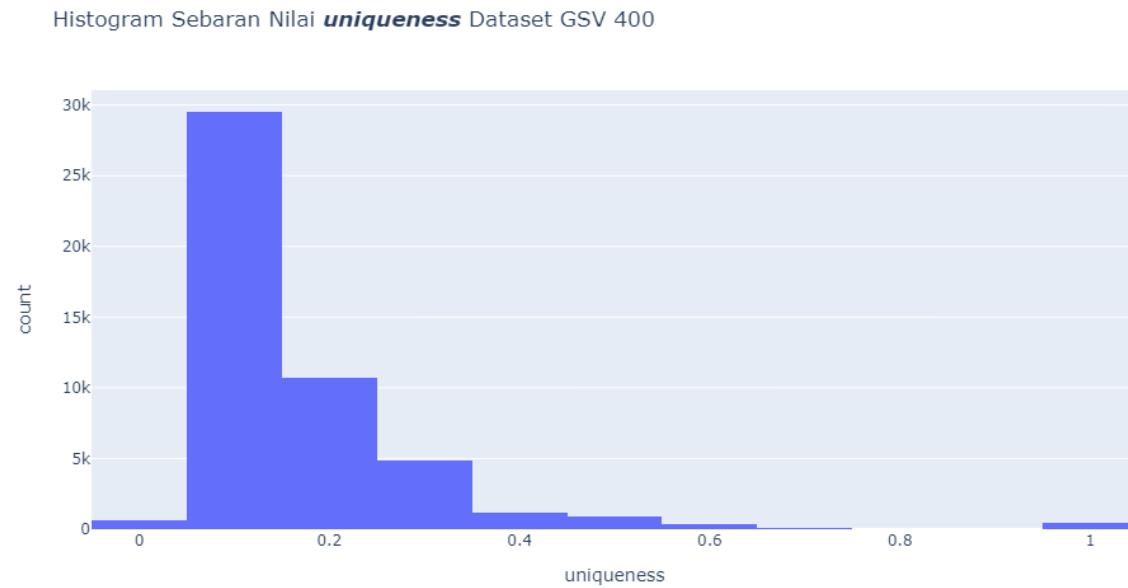
Pada pengujian dengan metode ORB ini hanya akan digunakan satu set *threshold*. *Threshold* yang digunakan adalah *threshold* yang didapat dari melihat hasil sebaran nilai konsistensi dan keunikan. Nilai *threshold* yang didapat dengan melihat hasilnya pada gambar seperti pada 3.7 tidak digunakan pada pengujian ini, karena pada penelitian ini belum dilakukan analisis *threshold* dengan menggunakan metode ORB.

GSV 400

Sebaran nilai konsistensi dan keunikan untuk *dataset* GSV 400 dapat dilihat pada Gambar 5.10 dan Gambar 5.11.



Gambar 5.10: Histogram sebaran nilai keunikan pada *dataset GSV 400*.



Gambar 5.11: Histogram sebaran nilai keunikan *dataset GSV 400*.

- ¹ Dari kedua histogram pada Gambar 5.10 dan Gambar 5.11 ditetapkan nilai *threshold* untuk
² konsistensi senilai 0.3 dan untuk keunikan senilai 0.3. Dua set *threshold* yang digunakan pada
³ pengujian ini dapat dilihat pada Tabel 5.7.

	Konsistensi	Keunikan
Keseluruhan	0.0	0.0
Threshold 1	0.3	0.3

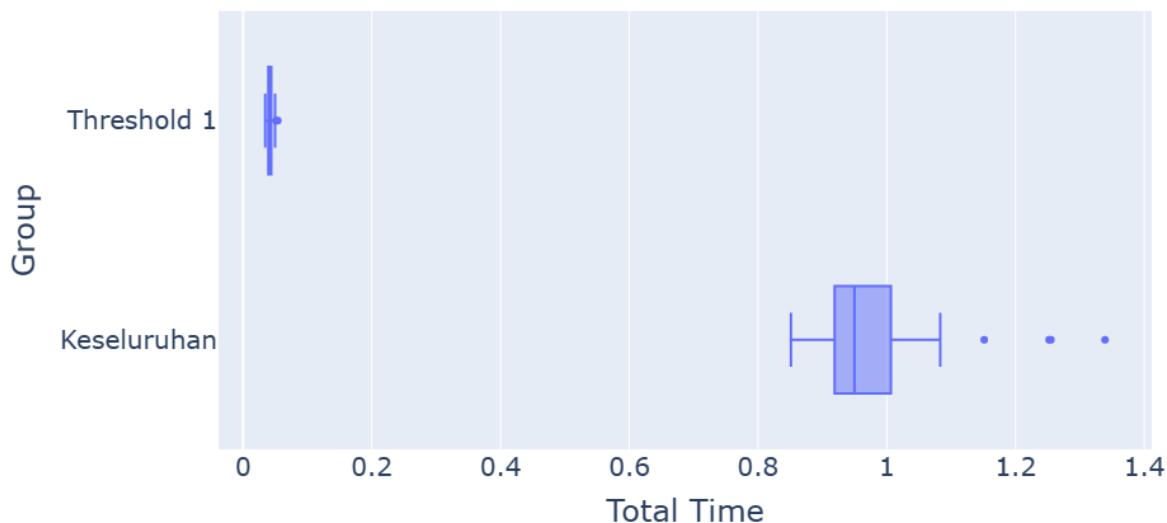
Tabel 5.7: Tiga set *threshold* yang digunakan untuk metode ORB *dataset* GSV 400.

- 1 Setelah dilakukan pengujian terhadap *dataset* didapat hasil seperti yang dapat dilihat pada
 2 Tabel 5.8. Untuk sebaran waktu secara rinci dapat dilihat pada *boxplot* di Gambar 5.12.

	Total Waktu Ekstraksi (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	0.17	49.04	78
Threshold 1	0.17	2.08	14

Tabel 5.8: Hasil pengujian pada *dataset* GSV 400.

Perbandingan Sebaran Waktu tiap Kelompok Dataset

Gambar 5.12: Sebaran waktu pengujian metode ORB *dataset* GSV 400.

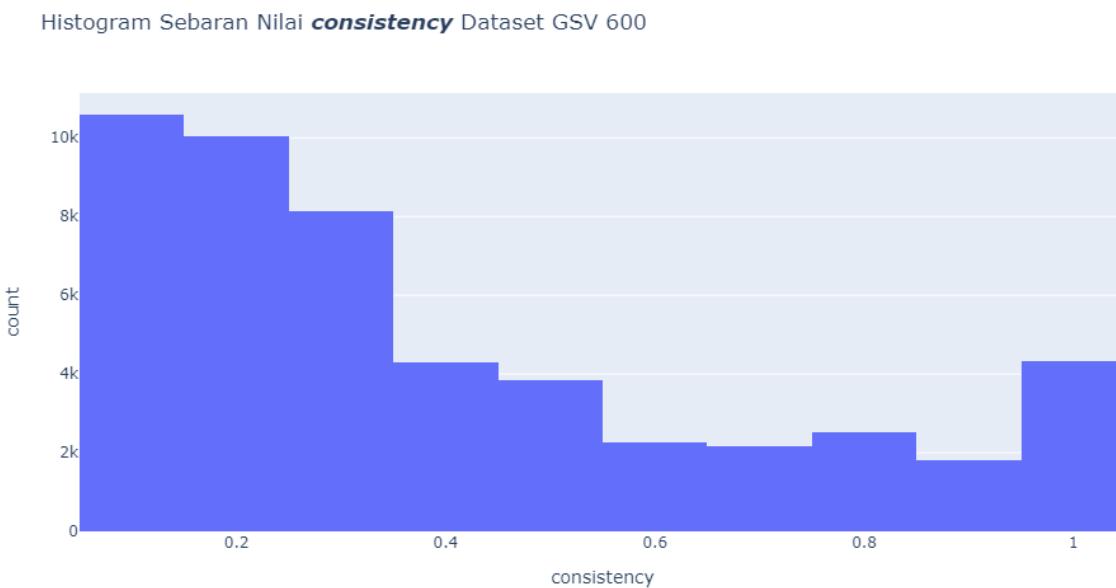
- 3 *Boxplot* pada Gambar 5.12 menunjukkan perbedaan sebaran waktu antara *dataset* GSV 400
 4 secara keseluruhan dan yang telah tersaring dengan menggunakan Threshold 1. Jika dilihat terdapat
 5 perbedaan yang signifikan antar keduanya, nilai-nilai pada *dataset* keseluruhan tersebar di kisaran
 6 0.8 sampai 1.1, sedangkan untuk *dataset* yang telah tersaring nilai-nilainya tersebar di kisaran 0.03
 7 sampai 0.05. Rincian nilai untuk kedua *boxplot* pada Gambar 5.12 dapat dilihat pada Tabel 5.9
 8 berikut.

	Keseluruhan	Threshold 1
<i>min</i>	0.8510	0.0344
<i>lower fence</i>	0.8510	0.0344
q1	0.9190	0.0391
q2 (median)	0.9498	0.0410
q3	1.0063	0.0435
<i>upper fence</i>	1.0829	0.0495
<i>max</i>	1.3390	0.0540

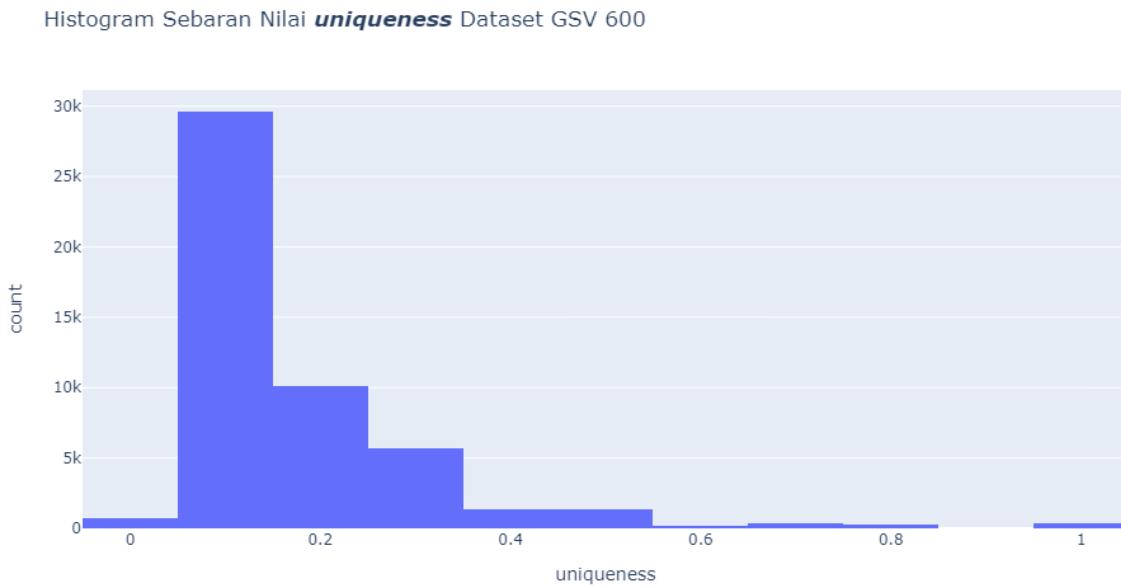
Tabel 5.9: Nilai-nilai perbandingan waktu metode ORB *dataset GSV 400*.

1 GSV 600

- 2 Sebaran nilai konsistensi dan keunikan untuk *dataset GSV 600* dapat dilihat pada Gambar 5.13 dan
 3 Gambar 5.14.



Gambar 5.13: Histogram sebaran nilai keunikan pada *dataset GSV 600*.



Gambar 5.14: Histogram sebaran nilai keunikan *dataset* GSV 600.

Dapat dilihat dari kedua histogram di Gambar 5.13 dan Gambar 5.14 bahwa sebaran nilai konsistensi dan keunikan untuk GSV 600 sama dengan sebaran pada GSV 400. Dari sebaran nilai yang sama antara GSV 400 dan GSV 600 ini maka untuk GSV 600 dapat digunakan set *threshold* yang sama dengan GSV 400. Dua set *threshold* yang digunakan pada pengujian GSV 600 ini dapat dilihat pada Tabel 5.10.

	Konsistensi	Keunikan
Keseluruhan	0.0	0.0
Threshold 1	0.3	0.3

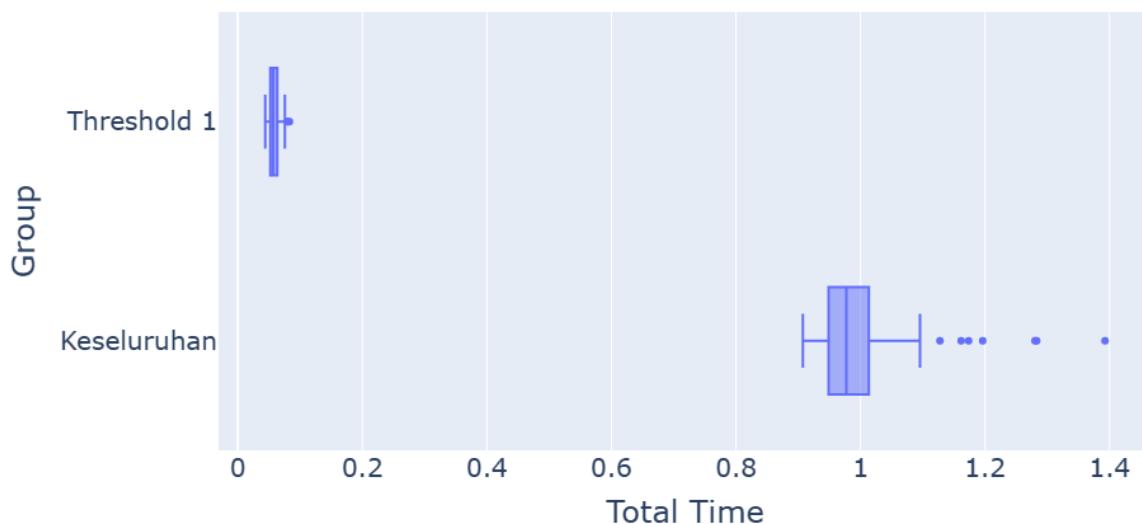
Tabel 5.10: Tiga set *threshold* yang digunakan untuk metode ORB *dataset* GSV 600.

Setelah dilakukan pengujian terhadap *dataset* didapat hasil seperti yang dapat dilihat pada Tabel 5.11. Waktu pengujian secara rinci dapat dilihat pada *boxplot* di Gambar 5.15.

	Total Waktu Ekstraksi (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	0.29	50.54	80
Threshold 1	0.29	2.92	20

Tabel 5.11: Hasil pengujian pada *dataset* GSV 600.

Perbandingan Sebaran Waktu tiap Kelompok Dataset



Gambar 5.15: Sebaran waktu pengujian metode ORB *dataset GSV 600*.

Boxplot pada Gambar 5.15 menunjukkan perbedaan sebaran waktu antara *dataset GSV 600* secara keseluruhan dan yang sudah tersaring dengan *Threshold 1*. Secara garis besar perbedaan antar keduanya cukup signifikan, relatif sama dengan sebaran pada *dataset GSV 400* (Gambar 5.6), hanya saja nilai-nilai pada *dataset GSV 600* berada pada kisaran nilai yang lebih tinggi. Pada *dataset GSV 600* ini secara keseluruhan nilai-nilainya berkisar di 0.9 sampai 1.1, sedangkan untuk yang telah tersaring nilai-nilainya berkisar di 0.04 sampai 0.07. Rincian nilai untuk kedua *boxplot* pada Gambar 5.15 tersebut dapat dilihat pada Tabel 5.12 berikut.

	Keseluruhan	Threshold 1
<i>min</i>	0.9073	0.0440
<i>lower fence</i>	0.9073	0.0440
q1	0.9487	0.0522
q2 (median)	0.9771	0.0567
q3	1.0013	0.0631
<i>upper fence</i>	1.0957	0.0758
<i>max</i>	1.3927	0.0830

Tabel 5.12: Nilai-nilai perbandingan waktu metode ORB *dataset GSV 600*.

5.2.4 Analisis Hasil Pengujian

Pada bagian analisis ini akan dibandingkan hasil dari pengujian antara SIFT dan ORB yang telah dilakukan. Pengujian yang dilakukan dengan menggunakan *threshold* dari hasil analisis *threshold* tidak akan ikut dibandingkan, karena nilai *threshold* tersebut hanya digunakan pada pengujian dengan SIFT. Perbandingan yang dilakukan akan dibagi berdasarkan ukuran gambar yang digunakan. Hasilnya dapat dilihat pada subbab-subbab berikut ini.

¹ GSV 400

		Jumlah Fitur Lokal	Total Waktu Ekstraksi (s)	Total Waktu BSIS (s)	Akurasi (%)
SIFT	Keseluruhan	115685	1.25	90.99	94
	Threshold 1	15365	1.29	24.30	84
ORB	Keseluruhan	48707	0.17	51.12	76
	Threshold 1	1958	0.33	1.81	28

Tabel 5.13: Hasil perbandingan pengujian SIFT dan ORB pada *dataset* GSV 400.

- ² Dari hasil pada Tabel 5.13 terlihat bahwa pada terdapat perbedaan hasil yang cukup signifikan
³ antara SIFT dan ORB. Terlihat bahwa dari SIFT dan ORB terdapat penurunan waktu ekstraksi fitur
⁴ yang cukup signifikan. Total waktu BSIS pada ORB juga menurun secara signifikan dibandingkan
⁵ dengan SIFT, tetapi penurunan ini lebih dikarenakan jumlah fitur lokal yang dihasilkan ORB
⁶ tidak sebanyak yang dihasilkan oleh SIFT. Nilai akurasi dari SIFT dan ORB juga mengalami
⁷ penurunan yang signifikan. Akurasi dari ORB ada pada angka yang termasuk rendah terutama
⁸ pada hasil yang telah disaring, hasil ini berbeda dari hasil yang didapat dari pengujian pada analisis
⁹ di 3.9. Pada pengujian di analisis sebelumnya identifikasi dengan menggunakan metode ORB masih
¹⁰ menghasilkan nilai akurasi yang cukup tinggi walaupun nilainya masih lebih kecil dari SIFT.

¹¹ GSV 600

		Jumlah Fitur Lokal	Total Waktu Ekstraksi (s)	Total Waktu BSIS (s)	Akurasi (%)
SIFT	Keseluruhan	231463	2.69	187.64	100
	Threshold 1	33735	2.57	47.58	84
ORB	Keseluruhan	49968	0.31	51.10	78
	Threshold 1	2145	0.29	2.99	20

Tabel 5.14: Hasil perbandingan pengujian SIFT dan ORB pada *dataset* GSV 600.

- ¹² Hasil yang dapat dilihat pada Tabel 5.14 menunjukkan pola yang sama dari SIFT dan ORB, di
¹³ mana SIFT memerlukan waktu yang lebih lama baik dari waktu ekstrak maupun waktu untuk
¹⁴ BSIS, tetapi juga menghasilkan akurasi yang lebih tinggi. Pada *dataset* ini waktu yang diperlukan
¹⁵ baik untuk ekstrak maupun BSIS lebih tinggi dari waktu yang diperlukan *dataset* GSV 400. Waktu
¹⁶ yang lebih lama tersebut dikarenakan ukuran gambar yang lebih besar sehingga lebih banyak fitur
¹⁷ lokal yang dapat dihasilkan dari tahap ekstraksi.

1

BAB 6

2

KESIMPULAN & SARAN

3 6.1 Kesimpulan

4 Kesimpulan yang diperoleh setelah mengerjakan skripsi ini sebagai berikut:

- 5 1. Model pengenalan POI dapat dibuat dengan membuat model berupa vektor *descriptor* dari
6 fitur lokal serta nilai konsistensi dan nilai keunikan dari fitur lokal tersebut. Nilai konsistensi
7 dan nilai keunikan tersebut dapat digunakan untuk menyaring fitur lokal sehingga diperoleh
8 fitur lokal yang konsisten atau unik saja. Berdasarkan pada analisis dan pengujian yang
9 dilakukan, penyaringan fitur lokal dengan menggunakan nilai konsistensi dan nilai keunikan
10 dapat membuang sebagian besar fitur lokal pada saat diuji dengan menggunakan *dataset* GSV.
11 Pada pengujian dengan menggunakan metode SIFT dan menggunakan nilai *threshold* 0.3
12 untuk nilai konsistensi dan 0.3 untuk nilai keunikan, didapat fitur lokal sebanyak 15365 dari
13 keseluruhan yang sebanyak 115685 fitur lokal untuk *dataset* GSV 400. Untuk *dataset* GSV 600
14 jumlah fitur lokal yang didapat setelah penyaringan dengan menggunakan *threshold* konsistensi
15 dan keunikan yang sama dengan GSV 400 sebanyak 33735 dari keseluruhan sebanyak 231463
16 fitur lokal.
- 17 2. Telah dibuat perangkat lunak yang dapat melakukan identifikasi POI dari sebuah gambar
18 dengan memanfaatkan model yang telah dihasilkan. Perangkat lunak melakukan pengenalan
19 POI dengan menggunakan metode BSIS. Model yang telah dihasilkan sebelumnya dapat
20 digunakan untuk memilih fitur lokal yang penting saja dan dapat mempercepat waktu proses
21 BSIS. Pada pengujian menggunakan metode SIFT *dataset* GSV, penyaringan fitur lokal
22 berdasarkan sebaran nilai konsistensi dan nilai keunikan dapat mempercepat total waktu
23 proses sebanyak 66.69 detik (73.2%) dengan penurunan nilai akurasi sebanyak 10% saat diuji
24 pada *dataset* berukuran maksimum 400 *pixel*. Pada saat pengujian dengan ukuran gambar
25 yang lebih besar, yaitu ukuran gambar maksimum 600 *pixel* terjadi penurunan total waktu
26 proses sebanyak 140.06 detik (74.6%) dengan penurunan akurasi sebanyak 10%.
- 27 3. Penggunaan metode ekstraksi fitur lokal ORB dibandingkan dengan SIFT cukup mempercepat
28 proses ekstraksi fitur lokal, walaupun dengan akurasi yang menurun juga. Pada pengujian di
29 *dataset* GSV, metode ORB secara rata-rata lebih cepat 80% dari metode SIFT pada ukuran
30 gambar maksimum 400 *pixel*, dengan penurunan akurasi sebanyak 18% tanpa dilakukan penya-
31 ringan. Pada *dataset* berukuran 600 *pixel* rata-rata penurunan waktu ekstraksi fitur adalah
32 88%, dengan penurunan nilai akurasi sebanyak 22% pada saat tidak dilakukan penyaringan.

6.2 Saran

Berdasarkan hasil yang telah diperoleh dan kesimpulan yang telah ditarik, terdapat beberapa saran yang mungkin dapat digunakan untuk penelitian lebih lanjut sebagai berikut:

1. Metode *clustering* dapat dilakukan dengan menggunakan metode selain Agglomerative Clustering. Contoh metode *clustering* lain yang dapat digunakan seperti DBSCAN.
2. Nilai akurasi dan nilai keunikan selain digunakan untuk menyaring fitur lokal dapat dicoba untuk digunakan sebagai bobot tambahan pada fitur lokal saat dilakukan BSIS. Cara ini akan menyebabkan fitur lokal yang sifatnya konsisten atau unik menjadi lebih berpengaruh terhadap penghitungan bobot antar pasangan gambar.

DAFTAR REFERENSI

- [1] Lowe, G. (2004) Sift-the scale invariant feature transform. *Int. J.*, **2**, 2.
- [2] Rublee, E., Rabaud, V., Konolige, K., dan Bradski, G. (2011) Orb: An efficient alternative to sift or surf. *2011 International conference on computer vision*, pp. 2564–2571. Ieee.
- [3] Kusuma, G. P., Harjono, K. D., dan Putra, M. T. D. (2019) Geometric verification method of best score increasing subsequence. *IEEE* , ?
- [4] Kusuma, G. P., Szabo, A., Yiqun, L., dan Lee, J. A. (2012) Appearance-based object recognition using weighted longest increasing subsequence. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 3668–3671. IEEE.
- [5] Bentley, J. L. (1975) Multidimensional binary search trees used for associative searching. *Communications of the ACM*, **18**, 509–517.

LAMPIRAN A

KODE PROGRAM

Kode A.1: bsis.py

```
1 """
2 Class untuk melakukan BSIS
3 """
4 #%%
5 import cv2
6 import numpy as np
7 from typing import Union
8 from operator import itemgetter
9 import time
10 import os
11 from glob import glob
12 from . import util
13 from scipy.spatial import KDTree
14 from enum import Enum
15
16 #%%
17 class BSIS:
18     class FLANN_INDEX():
19         LINEAR = 0
20         KDTREE = 1
21         KMEANS = 2
22         COMPOSITE = 3
23         KDTREE_SINGLE = 4
24         HIERARCHICAL = 5
25         LSH = 6
26         SAVED = 254
27         AUTOTUNED = 255
28
29     def __init__(self, query: Union[dict, tuple]):
30         if type(query) == dict:
31             self.query_kp = query['kp']
32             self.query_desc = query['desc']
33         elif type(query) == tuple:
34             self.query_kp = query[0]
35             self.query_desc = query[1]
36
37         self.train_set = None
38
39         self.pairing_time = 0
40         self.total_time = 0
41
42     def set_train_directory(self, directory: str):
43         train = TrainSet()
44         train.set_directory(directory)
45         self.train_set = train
46
47     def set_train_data(self, train_data):
48         if isinstance(train_data, tuple):
49             if len(train_data) == 3:
50                 kp = train_data[0]
51                 desc = train_data[1]
52                 mapper = train_data[2]
53                 train = TrainSet()
54                 train.set_data(kp, desc, mapper)
55                 self.train_set = train
56         elif isinstance(train_data, np.ndarray):
57             kp = list()
58             desc = list()
59             mapper = dict()
60             for i, lf in enumerate(train_data):
61                 kp.append(lf.keypoint)
62                 desc.append(lf.descriptor)
63                 mapper[i] = lf.img
64             kp = tuple(kp)
65             desc = np.array(desc)
66             train = TrainSet()
67             train.set_data(kp, desc, mapper)
68             self.train_set = train
69
70     def make_pairs(self, algorithm, t=4, k=100):
71         index_params = dict(algorithm=algorithm, trees=5)
72         # search_params = dict(checks=50)    # or pass empty dictionary
73         search_params = dict()
74         flann = cv2.FlannBasedMatcher(index_params, search_params)
```

```

76
77     start_time = time.time()
78     cv2.setRNGSeed(1311)
79     matches = flann.knnMatch(self.query_desc, self.train_set.train_desc, k=k)
80     self.pairing_time = time.time() - start_time
81     print('make_pairs---', self.pairing_time)
82
83     pairs = dict()
84     pair_idx = 0
85     for m in matches:
86         m_dists = list()
87         for i in m:
88             m_dists.append(i.distance)
89
90         st_dev = np.std(m_dists)
91         mean = np.mean(m_dists)
92         thres = mean - (t * st_dev)
93         self.thres_ = thres
94
95         for i in m:
96             if i.distance < thres:
97                 weight = ((i.distance - mean) / st_dev) ** 2
98                 if self.train_set.index_mapper[i.trainIdx] not in pairs.keys():
99                     pairs[self.train_set.index_mapper[i.trainIdx]] = list()
100
101            pairs[self.train_set.index_mapper[i.trainIdx]].append(
102                Pair(
103                    pair_idx,
104                    i.queryIdx,
105                    i.trainIdx,
106                    self.query_kp[i.queryIdx],
107                    self.train_set.train_kp[i.trainIdx],
108                    i.distance,
109                    weight
110                )
111            )
112            pair_idx += 1
113
114    return pairs
115
116 def run(self, algorithm: int, num_rotation=1, t=4, k=100):
117     start_time = time.time()
118     self.pairs = self.make_pairs(algorithm=algorithm, t=t, k=k)
119     self.result = dict()
120
121     maximum = ('', 0)
122     for k, v in self.pairs.items():
123         bsis = BSIS.Verify(v)
124         bsis.run(num_rotation=num_rotation)
125         self.result[k] = {'query_kp': bsis.selected_query_kp, 'train_kp': bsis.selected_train_kp, 'total_weight': bsis.
126                         total_weight}
127         if bsis.total_weight > maximum[1]:
128             maximum = (k, bsis.total_weight)
129
130     self.total_time = time.time() - start_time
131     print('total_time---', self.total_time)
132     return maximum[0]
133
134 #%%
135 class TrainSet:
136     def __init__(self):
137         self.sift = cv2.xfeatures2d.SIFT_create()
138
139         self.train_kp = list()
140         self.train_desc = list()
141         self.index_mapper = dict()
142
143     def set_directory(self, directory):
144         index = 0
145         for filename in glob(os.path.join(directory, '*.jpg')):
146             fname = filename.split('\\')[-1]
147             kp, desc = self.sift.detectAndCompute(util.get_image(filename), None)
148             for k, d in zip(kp, desc):
149                 self.train_kp.append(k)
150                 self.train_desc.append(d)
151                 self.index_mapper[index] = fname
152                 index += 1
153
154     self.train_desc = np.array(self.train_desc)
155
156     def set_data(self, kp, desc, mapper):
157         self.train_kp = kp
158         self.train_desc = desc
159         self.index_mapper = mapper
160
161 #Class untuk menyimpan pasangan antar keypoint
162 class Pair:
163     def __init__(self, pair_idx, query_idx, train_idx, query_kp, train_kp, distance, weight):
164         self.pair_idx = pair_idx
165         self.query_idx = query_idx
166         self.train_idx = train_idx
167         self.query_kp = query_kp
168         self.train_kp = train_kp
169         self.distance = distance
170         self.weight = weight
171
172     def print_all(self):
173         string = f'pair_idx:{self.pair_idx}\nquery_idx:{self.query_idx}\ntrain_idx:{self.train_idx}\ndistance:{self.
174             distance}\nweight:{self.weight}\n'

```

```

173     print(string)
174
175 #Class untuk menyimpan informasi pasangan keypoint yang diperlukan pada tahap verifikasi BSIS
176 class BPair:
177     def __init__(self, pair, order, weight, bts, prev):
178         self.pair = pair
179         self.order = order
180         self.weight = weight
181         self.bts = bts
182         self.prev = prev
183
184     def print_all(self):
185         try:
186             print(self.pair.pair_idx, self.order, self.weight, self.bts, self.prev.idx)
187         except:
188             print(self.pair.pair_idx, self.order, self.weight, self.bts, None)
189
190 #%%
191 class BSIS_Verify:
192     def __init__(self, pairs, k=100, t=4):
193         self.pairs = pairs
194
195         self.k = k
196         self.t = t
197
198     def get_pair_list_ordered(self, by=0, query_rotation=0, train_rotation=0):
199         paired_query_kp = set([(x.query_idx, x.query_kp.pt[0], x.query_kp.pt[1]) for x in self.pairs])
200         query_origin = (np.mean([i[1] for i in paired_query_kp]), np.mean([i[2] for i in paired_query_kp]))
201         query_order_x_set = [(i[0], util.rotate((i[1], i[2]), query_origin, query_rotation)[by]) for i in paired_query_kp]
202         query_order_x_set = sorted(query_order_x_set, key=itemgetter(1))
203         query_order_x = dict()
204         for i, idx in enumerate(query_order_x_set):
205             query_order_x[idx[0]] = i
206
207         paired_train_kp = set([(x.train_idx, x.train_kp.pt[0], x.train_kp.pt[1]) for x in self.pairs])
208         train_origin = (np.mean([i[1] for i in paired_train_kp]), np.mean([i[2] for i in paired_train_kp]))
209         train_order_x_set = [(i[0], util.rotate((i[1], i[2]), train_origin, train_rotation)[by]) for i in paired_train_kp]
210         train_order_x_set = sorted(train_order_x_set, key=itemgetter(1))
211         # train_order_x = dict()
212         # for i, idx in enumerate(train_order_x_set):
213         #     train_order_x[idx[0]] = i
214
215         train_idx_dict = dict()
216         for p in self.pairs:
217             if p.train_idx not in train_idx_dict.keys():
218                 train_idx_dict[p.train_idx] = list()
219                 train_idx_dict[p.train_idx].append(p)
220             else:
221                 train_idx_dict[p.train_idx].append(p)
222
223         ordered_list = list()
224         for i, j in train_order_x_set:
225             one_kp_pairs = train_idx_dict[i]
226             col_list = list()
227             for p in one_kp_pairs:
228                 col_list.append(BPair(p, query_order_x[p.query_idx], p.weight, p.weight, None))
229             ordered_list.append(col_list)
230
231     return ordered_list
232
233     def find_best_subsequence(self, D):
234         best_subsequence = BPair(0, 0, 0, 0, None)
235         for i in range(1, len(D)):
236             for j in range(0, len(D[i])):
237                 # start_time2 = time.time()
238                 dBestPrev = BPair(0, 0, 0, 0, None)
239                 for k in range(0, i):
240                     for l in range(0, len(D[k])):
241                         if D[k][l].order < D[i][j].order and D[k][l].bts > dBestPrev.bts:
242                             dBestPrev = D[k][l]
243                         # print('find dBestPrev {} - {}'.format(i, j), time.time() - start_time2)
244                         D[i][j].bts = D[i][j].weight + dBestPrev.bts
245                         D[i][j].prev = dBestPrev
246
247                         if D[i][j].bts + D[i][j].weight > best_subsequence.bts + best_subsequence.weight:
248                             best_subsequence = D[i][j]
249
250     return best_subsequence
251
252     def find_best_subsequence_(self, D):
253         best_subsequence = BPair(0, 0, 0, 0, None)
254         #dictionary menyimpan BPair dengan bts terbaik untuk tiap order
255         best_per_order = dict()
256
257         #iterasi kolom
258         for i in range(1, len(D)):
259             #iterasi baris pada tiap kolom
260             for j in range(0, len(D[i])):
261
262                 if D[i][j].order not in best_per_order.keys():
263                     best_per_order[D[i][j].order] = D[i][j]
264                 else:
265                     if D[i][j].bts > best_per_order[D[i][j].order].bts:
266                         best_per_order[D[i][j].order] = D[i][j]
267
268         dBestPrev = BPair(0, 0, 0, 0, None)
269         for ord in range(0, D[i][j].order):
270             if ord in best_per_order.keys():
271                 if best_per_order[ord].bts > dBestPrev.bts:

```

```

272                     dBestPrev = best_per_order[ord]
273
274             D[i][j].bts = D[i][j].weight + dBestPrev.bts
275             D[i][j].prev = dBestPrev
276
277             if D[i][j].bts + D[i][j].weight > best_subsequence.bts + best_subsequence.weight:
278                 best_subsequence = D[i][j]
279
280     return best_subsequence
281
282 def run(self, num_rotation=1):
283     max_weight = 0
284     selected_query_kp = []
285     selected_train_kp = []
286     self.rotation_weight = []
287     rotate_by = int(360.0 / num_rotation)
288     for i in range(0, 360, rotate_by):
289         query_rotation = i
290         train_rotation = 0
291         #print('query_rotation: ', query_rotation, '---', 'train_rotation:', train_rotation)
292
293         get_pair_list_start = time.time()
294         self.ordered = self.get_pair_list_ordered(query_rotation=query_rotation, train_rotation=train_rotation)
295
296         if len(self.ordered) < 1:
297             self.selected_query_kp = []
298             self.selected_train_kp = []
299             self.total_weight = 0
300             continue
301         #print('get_pair_list_ordered ---', time.time() - get_pair_list_start)
302
303         find_best_start = time.time()
304         self.bests = self.find_best_subsequence(self.ordered)
305         self.selected_pairs = []
306         while self.bests.prev != None:
307             self.selected_pairs.append(self.bests.pair)
308             self.bests = self.bests.prev
309         else:
310             pass
311             #self.selected_pairs.append(self.bests.pair)
312         #print('find_best_subsequence ---', time.time() - find_best_start)
313
314         weight = sum([i.weight for i in self.selected_pairs])
315         self.rotation_weight.append((i, weight))
316         if weight > max_weight:
317             max_weight = weight
318             selected_query_kp = [i.query_kp for i in self.selected_pairs]
319             selected_train_kp = [i.train_kp for i in self.selected_pairs]
320
321         self.selected_query_kp = selected_query_kp
322         self.selected_train_kp = selected_train_kp
323         self.total_weight = max_weight

```

Kode A.2: cluster_model.py

```

1 import pickle
2 import pandas as pd
3 import numpy as np
4 import cv2
5
6 from enum import Enum
7
8 class ClusterModel:
9     class Meta:
10         def __init__(self, n: int, maxsize, desc_type, class_names, images_per_class):
11             self.n = n
12             self.maxsize = maxsize
13             self.desc_type = desc_type
14             self.class_names = class_names
15             self.images_per_class = images_per_class
16
17     class Descriptor(Enum):
18         ORB = 'ORB'
19         SIFT = 'SIFT'
20
21     class LocalFeature:
22         def __init__(self, descriptor, keypoint, img, img_class, consistency, uniqueness):
23             self.descriptor = descriptor
24             self.keypoint = keypoint
25             self.img = img
26             self.img_class = img_class
27             self.consistency = consistency
28             self.uniqueness = uniqueness
29
30     def __init__(self, **kwargs):
31         self.meta = self.Meta(
32             n=kwargs.get('n'),
33             maxsize=kwargs.get('maxsize'),
34             desc_type=kwargs.get('desc_type'),
35             class_names=kwargs.get('class_names'),
36             images_per_class=kwargs.get('images_per_class')
37         )
38
39     def set_data(self, **kwargs):
40         if 'dataframe' in kwargs:
41             self.from_dataframe(kwargs.get('dataframe'))
42

```

```

43|     def from_dataframe(self, dataset):
44|         #get descriptor array
45|         if self.meta.desc_type == self.Descriptor.SIFT:
46|             self.descriptors = dataset.iloc[:, :128].values
47|         elif self.meta.desc_type == self.Descriptor.ORB:
48|             self.descriptors = dataset.iloc[:, :32].values
49|
50|         #get cluster labels
51|         self.cluster_label = dataset.cluster_label.values
52|         self.cluster2_label = dataset.cluster2_label.values
53|
54|         #get keypoints array
55|         keypoint_columns = ['point_x', 'point_y', 'size', 'angle', 'response', 'octave', 'class_id']
56|         self.keypoints = dataset.loc[:, keypoint_columns].values
57|
58|         #get image details
59|         self.img = dataset.img.values
60|         self.img_class = dataset.img_class.values
61|
62|         #get consistency and uniqueness
63|         self.consistency = dataset.consistency.values
64|         self.uniqueness = dataset.uniqueness.values
65|
66|     def get_local_features(self, min_uniqueness=0.0, min_consistency=0.0, img=None, img_class=None):
67|         cons_mask = self.consistency >= min_consistency
68|         uniq_mask = self.uniqueness >= min_uniqueness
69|
70|         if img:
71|             img_mask = self.img == img
72|         else:
73|             img_mask = np.full(self.img.shape[0], True)
74|         if img_class:
75|             img_class_mask = self.img_class == img_class
76|         else:
77|             img_class_mask = np.full(self.img_class.shape[0], True)
78|
79|         mask = cons_mask & uniq_mask & img_mask & img_class_mask
80|
81|         filtered_descriptors = self.descriptors[mask]
82|         filtered_keypoints = np.apply_along_axis(self.get_keypoint, axis=1, arr=self.keypoints[mask])
83|         filtered_img = self.img[mask]
84|         filtered_img_class = self.img_class[mask]
85|         filtered_consistency = self.consistency[mask]
86|         filtered_uniqueness = self.uniqueness[mask]
87|
88|         filtered_columns = zip(filtered_descriptors,
89|                                filtered_keypoints,
90|                                filtered_img,
91|                                filtered_img_class,
92|                                filtered_consistency,
93|                                filtered_uniqueness,
94|                                )
95|
96|         local_features = list()
97|         for desc, kp, img, img_class, cons, uniq in filtered_columns:
98|             local_features.append(self.LocalFeature(desc, kp, img, img_class, cons, uniq))
99|
100|        return np.array(local_features, dtype=self.LocalFeature)
101|
102|    def get_dataframe(self):
103|        df = pd.DataFrame(self.descriptors)
104|        df['img'] = self.img
105|        df['img_class'] = self.img_class
106|        df['cluster_label'] = self.cluster_label
107|        df['cluster2_label'] = self.cluster2_label
108|        df['uniqueness'] = self.uniqueness
109|        df['consistency'] = self.consistency
110|
111|        return df
112|
113|    def save(self, filename):
114|        with open(filename, 'wb') as file:
115|            pickle.dump(self.__dict__, file, 2)
116|
117|    def load(self, filename):
118|        with open(filename, 'rb') as file:
119|            class_dict = pickle.load(file)
120|            self.__dict__.update(class_dict)
121|
122|    def get_keypoint(self, row):
123|        keypoint = cv2.KeyPoint(
124|            x=float(row[0]),
125|            y=float(row[1]),
126|            size=float(row[2]),
127|            angle=float(row[3]),
128|            response=float(row[4]),
129|            octave=int(row[5]),
130|            class_id=int(row[5])
131|        )
132|
133|        return keypoint

```

Kode A.3: clustering_sift.py

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Mar 13 12:41:29 2022

```

```

4
5 @author: gianm
6 """
7
8 #%%
9 #Import libraries
10 import warnings
11 warnings.simplefilter(action='ignore', category=FutureWarning)
12
13 import cv2
14 import pandas as pd
15 import numpy as np
16
17 import util
18 from clustermodel import ClusterModel
19
20 import argparse
21 import time
22
23 from threading import Thread
24
25 #%%
26 #Load Dataset
27 def load_dataset(dataset_dir, maxheight=600, maxwidth=600):
28     dataset = util.get_dataset(dataset_dir, maxheight=maxheight, maxwidth=maxwidth)
29     return dataset
30
31 #%%
32 #Detect keypoint
33 def detect_keypoint(dataset):
34     sift = cv2.SIFT_create()
35
36     keypoints = list()
37     labels = list()
38
39     #buat kolom untuk menyimpan gambar asal
40     kp_index = 0
41     desc_arrays = list()
42     for group in dataset.keys():
43         for i, img in enumerate(dataset[group]):
44             kp, desc = sift.detectAndCompute(img[1], None)
45             #masukkan ke dataframe dataset
46             desc_arrays.append(desc)
47             for i, k in enumerate(kp):
48                 #masukkan keypoint ke list
49                 keypoints.append(k)
50                 #label untuk tiap deskriptor, berisi: nama gambar, kelas gambar, dan index keypoint pada list
51                 labels.append((group, img[0], kp_index))
52                 kp_index += 1
53
54     descs = np.concatenate(desc_arrays, axis=0)
55     descriptors = pd.DataFrame(descs)
56
57     #masukkan labels ke dalam dataframe
58     descriptors.loc[:, 'img'] = [i[1] for i in labels]
59     descriptors.loc[:, 'img_class'] = [i[0] for i in labels]
60     descriptors.loc[:, 'kp_idx'] = [i[2] for i in labels]
61
62     return keypoints, descriptors
63
64 #%%
65 #Cluster per image
66 def cluster_per_image(descriptors):
67     descriptors_ = descriptors
68     img_group = descriptors_.groupby('img')
69
70     one_image_dict = dict()
71     descs_cluster_label = list()
72     df_centroids = list()
73
74     for image, one_image in img_group:
75         one_image_desc_only = one_image.drop(columns=['img', 'img_class', 'kp_idx'])
76         sample_size = 100
77         if len(one_image_desc_only.index) < sample_size:
78             sample_size = None
79         threshold = util.combination_distance(
80             one_image_desc_only,
81             sample=sample_size,
82             metric=util.euclidean_dist
83         )
84         one_image_cluster_label = util.aggro_cluster(
85             one_image_desc_only,
86             distance_threshold=threshold,
87             affinity='euclidean'
88         )
89         one_image.loc[:, 'cluster_label'] = one_image_cluster_label
90         descs_cluster_label = descs_cluster_label + list(one_image_cluster_label)
91
92         cluster_label_group = one_image.groupby('cluster_label')
93
94         centroid_arrays = list()
95         labels_list = list()
96         for label, one_cluster in cluster_label_group:
97             one_cluster = one_cluster.drop(columns=[ 'cluster_label', 'img', 'img_class', 'kp_idx'])
98             mean_arr = np.array([np.mean(one_cluster.values, axis=0)])
99             centroid_arrays.append(mean_arr)
100            labels_list.append(label)
101        centroid_arr = np.concatenate(centroid_arrays, axis=0)
102

```

```

103     centroid_df = pd.DataFrame(centroid_arr)
104     centroid_df.loc[:, 'cluster_label'] = labels_list
105     centroid_df.loc[:, 'img'] = [image] * len(centroid_df.index)
106     centroid_df.loc[:, 'img_class'] = one_image.img_class.head(len(centroid_df.index)).tolist()
107     df_centroids.append(centroid_df)
108
109 df_centroid = pd.concat(df_centroids, axis=0)
110 df_centroid = df_centroid.reset_index()
111
112 descriptors_['cluster_label'] = descs_cluster_label
113
114 return df_centroid, descriptors_
115
116 #%%
117 #Cluster centroid
118 def cluster_centroid(df_centroid):
119     df_centroid_ = df_centroid
120     sample_size = 100
121     threshold = util.combination_distance(
122         df_centroid_.drop(columns=['img', 'img_class', 'cluster_label']),
123         sample=sample_size,
124         metric=util.euclidean_dist
125     )
126
127     cluster2_labels = util.aggro_cluster(
128         df_centroid_.drop(columns=['img', 'img_class', 'cluster_label']),
129         distance_threshold=threshold,
130         affinity='euclidean'
131     )
132
133 df_centroid_['cluster2_label'] = cluster2_labels
134 return df_centroid_
135
136 #%%
137 #Calculate uniqueness
138 def calculate_uniqueness(df_centroid):
139     df_centroid_ = df_centroid
140     cluster2_class_count = dict()
141     cluster2_group = df_centroid_.groupby('cluster2_label')
142
143     for c2 in cluster2_group.groups.keys():
144         one_cluster2 = cluster2_group.get_group(c2)
145         one_cluster2 = one_cluster2.drop_duplicates(subset='img')
146         cluster2_class_count[c2] = one_cluster2['img_class'].value_counts(normalize=True)
147
148     img_class_count = [cluster2_class_count[c2lab][imgc] for c2lab, imgc in zip(df_centroid_.cluster2_label, df_centroid_.img_class)]
149     df_centroid_['uniqueness'] = img_class_count
150
151 return df_centroid_
152
153 #%%
154 #Calculate consistency
155 def calculate_consistency(df_centroid, num_of_images):
156     df_centroid_ = df_centroid
157     cluster2_img_class_group = df_centroid_.groupby(['cluster2_label', 'img_class']).img.nunique()
158     img_count = list()
159
160     for c2l, ic in zip(df_centroid_.cluster2_label, df_centroid_.img_class):
161         img_in_class = cluster2_img_class_group[c2l][ic]
162         img_count.append(img_in_class / num_of_images)
163
164     df_centroid_['consistency'] = img_count
165
166 return df_centroid_
167
168 #%%
169 #Join one image with centroid
170 def join_with_centroid(df_centroid, descriptors):
171     df_centroid_val = df_centroid[['img', 'cluster_label', 'cluster2_label', 'uniqueness', 'consistency']]
172     descriptors_ = pd.merge(descriptors, df_centroid_val, on=['img', 'cluster_label'])
173
174 return descriptors_
175
176 #%%
177 #Save keypoints details
178 def save_keypoint_details(keypoints, descriptors):
179     descriptors_ = descriptors
180     point_x = list()
181     point_y = list()
182     size = list()
183     angle = list()
184     response = list()
185     octave = list()
186     class_id = list()
187
188     for k in descriptors_.kp_idx:
189         kp = keypoints[k]
190         point_x.append(kp.pt[0])
191         point_y.append(kp.pt[1])
192         size.append(kp.size)
193         angle.append(kp.angle)
194         response.append(kp.response)
195         octave.append(kp.octave)
196         class_id.append(kp.class_id)
197
198     descriptors_['point_x'] = point_x
199     descriptors_['point_y'] = point_y
200     descriptors_['size'] = size
201     descriptors_['angle'] = angle
202     descriptors_['response'] = response

```

```

201     descriptors_['octave'] = octave
202     descriptors_['class_id'] = class_id
203
204     return descriptors_
205
206 def dataset_details(dataset):
207     class_names = list(dataset.keys())
208     num_of_images = set([len(i) for i in dataset.values()])
209     if len(num_of_images) == 1:
210         return class_names, list(num_of_images)[0]
211     else:
212         raise ValueError('Dataset contains uneven number of images!')
213
214 def loading_animation(process_name):
215     anim = '|/-\\'
216     idx = 0
217     while True:
218         print(f'{process_name}---{anim[idx%len(anim)]}', end='\r')
219         idx += 1
220         time.sleep(0.1)
221         global stop_threads
222         if stop_threads:
223             break
224
225 def main(args):
226     dir = args.dir
227     maxsize = int(args.maxsize)
228
229     start_time = time.time()
230     global stop_threads
231
232     #Load Dataset
233     stop_threads = False
234     thread = Thread(target=loading_animation, args=(('Load_Dataset', )))
235     thread.start()
236     dataset = load_dataset(dir, maxheight=maxsize, maxwidth=maxsize)
237     class_names, image_per_class = dataset_details(dataset)
238     load_dataset_time = time.time()
239     stop_threads = True
240     thread.join()
241     print(f'Load_Dataset---{round(load_dataset_time - start_time, 2)}s')
242
243     #Detect Keypoints
244     stop_threads = False
245     thread = Thread(target=loading_animation, args=(('Detect_Keypoints', )))
246     thread.start()
247     keypoints, descriptors = detect_keypoint(dataset)
248     detect_keypoint_time = time.time()
249     stop_threads = True
250     thread.join()
251     print(f'Detect_Keypoints---{round(detect_keypoint_time - load_dataset_time, 2)}s')
252
253     #Cluster per Image
254     stop_threads = False
255     thread = Thread(target=loading_animation, args=(('Cluster_per_Image', )))
256     thread.start()
257     df_centroid, descriptors = cluster_per_image(descriptors)
258     cluster_per_image_time = time.time()
259     stop_threads = True
260     thread.join()
261     print(f'Cluster_per_Image---{round(cluster_per_image_time - detect_keypoint_time, 2)}s')
262
263     #Cluster Centroid
264     stop_threads = False
265     thread = Thread(target=loading_animation, args=(('Cluster_Centroid', )))
266     thread.start()
267     df_centroid = cluster_centroid(df_centroid)
268     cluster_centroid_time = time.time()
269     stop_threads = True
270     thread.join()
271     print(f'Cluster_Centroid---{round(cluster_centroid_time - cluster_per_image_time, 2)}s')
272
273     #Calculate Uniqueness
274     stop_threads = False
275     thread = Thread(target=loading_animation, args=(('Calculate_Uniqueness', )))
276     thread.start()
277     df_centroid = calculate_uniqueness(df_centroid)
278     calculate_uniqueness_time = time.time()
279     stop_threads = True
280     thread.join()
281     print(f'Calculate_Uniqueness---{round(calculate_uniqueness_time - cluster_centroid_time, 2)}s')
282
283     #Calculate Consistency
284     stop_threads = False
285     thread = Thread(target=loading_animation, args=(('Calculate_Consistency', )))
286     thread.start()
287     df_centroid = calculate_consistency(df_centroid, image_per_class)
288     calculate_consistency_time = time.time()
289     stop_threads = True
290     thread.join()
291     print(f'Calculate_Consistency---{round(calculate_consistency_time - calculate_uniqueness_time, 2)}s')
292
293     #Join with Centroid
294     stop_threads = False
295     thread = Thread(target=loading_animation, args=(('Join_with_Centroid', )))
296     thread.start()
297     descriptors = join_with_centroid(df_centroid, descriptors)
298     join_with_centroid_time = time.time()
299     stop_threads = True

```

```

300    thread.join()
301    print(f'Join_with_Centroid---{round(join_with_centroid_time,-_calculate_consistency_time,_2)}s')
302
303    #Save Keypoint Details
304    stop_threads = False
305    thread = Thread(target=loading_animation, args=( 'Save_Keypoint_Details', ))
306    thread.start()
307    descriptors = save_keypoint_details(keypoints, descriptors)
308    save_keypoint_details.time = time.time()
309    stop_threads = True
310    thread.join()
311    print(f'Save_Keypoint_Details---{round(save_keypoint_details.time-_join_with_centroid_time,_2)}s')
312
313    #create ClusterModel
314    stop_threads = False
315    thread = Thread(target=loading_animation, args=( 'Create_ClusterModel', ))
316    thread.start()
317    n = len(descriptors.index)
318    desc_type = ClusterModel.Descriptor.SIFT
319
320    cm = ClusterModel(
321        n=n,
322        desc_type=desc_type,
323        class_names=class.names,
324        image_per_class=image_per_class
325    )
326
327    cm.set_data(dataframe=descriptors)
328
329    dataset_name = dir.split('/')[1]
330    cm_name = f'{dataset_name}_sift_agglo_{maxsize}.cm'
331    cm.save(cm_name)
332    create_clustermodel_time = time.time()
333    stop_threads = True
334    thread.join()
335    print(f'Create_ClusterModel---{round(create_clustermodel_time-_save_keypoint_details_time,_2)}s')
336
337    total_time = time.time()
338    print(f'__Model_saved_to_{cm_name}')
339    print(f'\nTotal_time---{round(total_time-_start_time,_2)}s')
340
341
342 if __name__ == '__main__':
343     parser = argparse.ArgumentParser(description='clustering')
344     parser.add_argument('--dir', help='file_dir', default=None)
345     parser.add_argument('--maxsize', help='maximum_image_size', default=600)
346
347     args = parser.parse_args()
348
349     main(args)

```

Kode A.4: clustering_orb.py

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Mar 13 12:41:29 2022
4
5  @author: gianm
6  """
7
8  #%%
9  #Import libraries
10 import warnings
11 warnings.simplefilter(action='ignore', category=FutureWarning)
12
13 import cv2
14 import pandas as pd
15 import numpy as np
16
17 from modules import util
18 from modules.clustermodel import ClusterModel
19
20 import argparse
21 import time
22
23 from threading import Thread
24
25 #%%
26 #Load Dataset
27 def load_dataset(dataset_dir, maxheight=600, maxwidth=600):
28     dataset = util.get_dataset(dataset_dir, maxheight=maxheight, maxwidth=maxwidth)
29     return dataset
30
31 #%%
32 #Detect keypoint
33 def detect_keypoint(dataset):
34     orb = cv2.ORB_create()
35
36     keypoints = list()
37     labels = list()
38
39     #buat kolom untuk menyimpan gambar asal
40     kp_index = 0
41     desc_arrays = list()
42     for group in dataset.keys():
43         for i, img in enumerate(dataset[group]):
44             kp, desc = orb.detectAndCompute(img[1], None)

```

```

45     desc = util.int_to_binary(desc)
46     #masukkan ke dataframe dataset
47     desc_arrays.append(desc)
48     for i, k in enumerate(kp):
49         #masukkan keypoint ke list
50         keypoints.append(k)
51         #label untuk tiap deskriptor, berisi: nama gambar, kelas gambar, dan index keypoint pada list
52         labels.append((group, img[0], kp_index))
53         kp_index += 1
54
55     descs = np.concatenate(desc_arrays, axis=0)
56     descriptors = pd.DataFrame(descs)
57
58     #masukkan labels ke dalam dataframe
59     descriptors.loc[:, 'img'] = [i[1] for i in labels]
60     descriptors.loc[:, 'img_class'] = [i[0] for i in labels]
61     descriptors.loc[:, 'kp_idx'] = [i[2] for i in labels]
62
63     return keypoints, descriptors
64
65 #%%
66 #Cluster per image
67 def cluster_per_image(descriptors):
68     descriptors_ = descriptors
69     img_group = descriptors_.groupby('img')
70
71     #one_image_dict = dict()
72     descs_cluster_label = list()
73     df_centroids = list()
74
75     for image, one_image in img_group:
76         one_image_desc_only = one_image.drop(columns=['img', 'img_class', 'kp_idx'])
77         sample_size = 100
78         if len(one_image_desc_only.index) < sample_size:
79             sample_size = None
80         threshold = util.combination_distance(
81             one_image_desc_only,
82             sample=sample_size,
83             metric=util.hamming_bin
84         )
85         one_image_cluster_label = util.agglo_cluster(
86             one_image_desc_only,
87             distance_threshold=threshold,
88             affinity='hamming'
89         )
90         one_image.loc[:, 'cluster_label'] = one_image_cluster_label
91         descs_cluster_label = descs_cluster_label + list(one_image_cluster_label)
92
93     cluster_label_group = one_image.groupby('cluster_label')
94
95     centroid_arrays = list()
96     labels_list = list()
97     for label, one_cluster in cluster_label_group:
98         one_cluster = one_cluster.drop(columns=['cluster_label', 'img', 'img_class', 'kp_idx'])
99         mean_arr = np.array([np.mean(one_cluster.values, axis=0)])
100        centroid_arrays.append(mean_arr)
101        labels_list.append(label)
102    centroid_arr = np.concatenate(centroid_arrays, axis=0)
103
104    centroid_df = pd.DataFrame(centroid_arr)
105    centroid_df.loc[:, 'cluster_label'] = labels_list
106    centroid_df.loc[:, 'img'] = [image] * len(centroid_df.index)
107    centroid_df.loc[:, 'img_class'] = one_image.img_class.head(len(centroid_df.index)).tolist()
108    df_centroids.append(centroid_df)
109
110    df_centroid = pd.concat(df_centroids, axis=0)
111    df_centroid = df_centroid.reset_index()
112
113    descriptors_[['cluster_label']] = descs_cluster_label
114
115    return df_centroid, descriptors_
116
117 #%%
118 #Cluster centroid
119 def cluster_centroid(df_centroid):
120     df_centroid_ = df_centroid
121     sample_size = 100
122     threshold = util.combination_distance(
123         df_centroid_.drop(columns=['img', 'img_class', 'cluster_label']),
124         sample=sample_size,
125         metric=util.hamming_bin
126     )
127
128     cluster2_labels = util.agglo_cluster(
129         df_centroid_.drop(columns=['img', 'img_class', 'cluster_label']),
130         distance_threshold=threshold,
131         affinity='hamming'
132     )
133
134     df_centroid_[['cluster2_label']] = cluster2_labels
135     return df_centroid_
136
137 #%%
138 #Calculate uniqueness
139 def calculate_uniqueness(df_centroid):
140     df_centroid_ = df_centroid
141     cluster2_class_count = dict()
142     cluster2_group = df_centroid_.groupby('cluster2_label')
143

```

```

144     for c2 in cluster2_group.groups.keys():
145         one_cluster2 = cluster2_group.get_group(c2)
146         one_cluster2 = one_cluster2.drop_duplicates(subset='img')
147         cluster2_class_count[c2] = one_cluster2['img_class'].value_counts(normalize=True)
148
149     img_class_count = [cluster2_class_count[c2lab][imgc] for c2lab, imgc in zip(df_centroid_.cluster2_label, df_centroid_.
150         img_class)]
151     df_centroid_['uniqueness'] = img_class_count
152
153     return df_centroid_
154
155 #%%%
156 #Calculate consistency
157 def calculate_consistency(df_centroid, num_of_images):
158     df_centroid_ = df_centroid
159     cluster2_img_class_group = df_centroid_.groupby(['cluster2_label', 'img_class']).img.nunique()
160     img_count = list()
161
162     for c2l, ic in zip(df_centroid_.cluster2_label, df_centroid_.img_class):
163         img_in_class = cluster2_img_class_group[c2l][ic]
164         img_count.append(img_in_class / num_of_images)
165
166     df_centroid_[‘consistency’] = img_count
167
168     return df_centroid_
169
170 #Join one image with centroid
171 def join_with_centroid(df_centroid, descriptors):
172     df_centroid_val = df_centroid[['img', 'cluster_label', 'cluster2_label', 'uniqueness', 'consistency']]
173     descriptors_ = pd.merge(descriptors, df_centroid_val, on=['img', 'cluster_label'])
174
175     descriptors_only = np.array(descriptors_.iloc[:, range(256)], dtype='uint8')
176     data_only = descriptors_.iloc[:, range(256, 263)]
177
178     descriptors_only = pd.DataFrame(util.binary_to_int(descriptors_only))
179     descriptors_ = pd.concat([descriptors_only, data_only], axis=1)
180
181     return descriptors_
182
183 #%%
184 #Save keypoints details
185 def save_keypoint_details(keypoints, descriptors):
186     descriptors_ = descriptors
187     point_x = list()
188     point_y = list()
189     size = list()
190     angle = list()
191     response = list()
192     octave = list()
193     class_id = list()
194
195     for k in descriptors_.kp_idx:
196         kp = keypoints[k]
197         point_x.append(kp.pt[0])
198         point_y.append(kp.pt[1])
199         size.append(kp.size)
200         angle.append(kp.angle)
201         response.append(kp.response)
202         octave.append(kp.octave)
203         class_id.append(kp.class_id)
204
205     descriptors_[‘point_x’] = point_x
206     descriptors_[‘point_y’] = point_y
207     descriptors_[‘size’] = size
208     descriptors_[‘angle’] = angle
209     descriptors_[‘response’] = response
210     descriptors_[‘octave’] = octave
211     descriptors_[‘class_id’] = class_id
212
213     return descriptors_
214
215 def dataset_details(dataset):
216     class_names = list(dataset.keys())
217     num_of_images = set([len(i) for i in dataset.values()])
218     if len(num_of_images) == 1:
219         return class_names, list(num_of_images)[0]
220     else:
221         raise ValueError('Dataset contains uneven number of images!')
222
223 def loading_animation(process_name):
224     anim = '|/-\\'
225     idx = 0
226     while True:
227         print(f'{process_name} {anim[idx % len(anim)]}', end='\r')
228         idx += 1
229         time.sleep(0.1)
230         global stop_threads
231         if stop_threads:
232             break
233
234 def main(args):
235     dir = args.dir
236     maxsize = int(args.maxsize)
237
238     start_time = time.time()
239     global stop_threads
240
241     #Load Dataset
242     stop_threads = False
243     thread = Thread(target=loading_animation, args=(‘Load_Dataset’, ))

```

```

242     thread.start()
243     dataset = load_dataset(dir, maxheight=maxsize, maxwidth=maxsize)
244     class_names, image_per_class = dataset_details(dataset)
245     load_dataset_time = time.time()
246     stop_threads = True
247     thread.join()
248     print(f'Load_Dataset---{round(load_dataset_time-_start_time,_2)}s')
249
250 #Detect Keypoints
251 stop_threads = False
252 thread = Thread(target=loading_animation, args=('Detect_Keypoints', ))
253 thread.start()
254 keypoints, descriptors = detect_keypoint(dataset)
255 detect_keypoint_time = time.time()
256 stop_threads = True
257 thread.join()
258 print(f'Detect_Keypoints---{round(detect_keypoint_time-_load_dataset_time,_2)}s')
259
260 #Cluster per Image
261 stop_threads = False
262 thread = Thread(target=loading_animation, args=('Cluster_per_Image', ))
263 thread.start()
264 df_centroid, descriptors = cluster_per_image(descriptors)
265 cluster_per_image_time = time.time()
266 stop_threads = True
267 thread.join()
268 print(f'Cluster_per_Image---{round(cluster_per_image_time-_detect_keypoint_time,_2)}s')
269
270 #Cluster Centroid
271 stop_threads = False
272 thread = Thread(target=loading_animation, args=('Cluster_Centroid', ))
273 thread.start()
274 df_centroid = cluster_centroid(df_centroid)
275 cluster_centroid_time = time.time()
276 stop_threads = True
277 thread.join()
278 print(f'Cluster_Centroid---{round(cluster_centroid_time-_cluster_per_image_time,_2)}s')
279
280 #Calculate Uniqueness
281 stop_threads = False
282 thread = Thread(target=loading_animation, args=('Calculate_Uniqueness', ))
283 thread.start()
284 df_centroid = calculate_uniqueness(df_centroid)
285 calculate_uniqueness_time = time.time()
286 stop_threads = True
287 thread.join()
288 print(f'Calculate_Uniqueness---{round(calculate_uniqueness_time-_cluster_centroid_time,_2)}s')
289
290 #Calculate Consistency
291 stop_threads = False
292 thread = Thread(target=loading_animation, args=('Calculate_Consistency', ))
293 thread.start()
294 df_centroid = calculate_consistency(df_centroid, image_per_class)
295 calculate_consistency_time = time.time()
296 stop_threads = True
297 thread.join()
298 print(f'Calculate_Consistency---{round(calculate_consistency_time-_calculate_uniqueness_time,_2)}s')
299
300 #Join with Centroid
301 stop_threads = False
302 thread = Thread(target=loading_animation, args=('Join_with_Centroid', ))
303 thread.start()
304 descriptors = join_with_centroid(df_centroid, descriptors)
305 join_with_centroid_time = time.time()
306 stop_threads = True
307 thread.join()
308 print(f'Join_with_Centroid---{round(join_with_centroid_time-_calculate_consistency_time,_2)}s')
309
310 #Save Keypoint Details
311 stop_threads = False
312 thread = Thread(target=loading_animation, args=('Save_Keypoint_Details', ))
313 thread.start()
314 descriptors = save_keypoint_details(keypoints, descriptors)
315 save_keypoint_details_time = time.time()
316 stop_threads = True
317 thread.join()
318 print(f'Save_Keypoint_Details---{round(save_keypoint_details_time-_join_with_centroid_time,_2)}s')
319
320 #create ClusterModel
321 stop_threads = False
322 thread = Thread(target=loading_animation, args=('Create_ClusterModel', ))
323 thread.start()
324 n = len(descriptors.index)
325 desc_type = ClusterModel.Descriptor.ORB
326
327 cm = ClusterModel(
328     n=n,
329     maxsize=maxsize,
330     desc_type=desc_type,
331     class_names=class_names,
332     image_per_class=image_per_class
333 )
334
335 cm.set_data(dataframe=descriptors)
336
337 dataset_name = dir.split('/')[-1]
338 cm_name = f'{dataset_name}_orb_agglo_{maxsize}.cm'
339 cm.save(cm_name)
340 create_clustermodel_time = time.time()

```

```

341     stop_threads = True
342     thread.join()
343     print(f'Create_ClusterModel_{round(create_clustermodel_time_ - save_keypoint_details_time_,_2)}s')
344
345     total_time = time.time()
346     print(f'__Model_saved_to_{cm_name}')
347     print(f'\nTotal_time_{round(total_time_ - start_time_,_2)}s')
348
349
350 if __name__ == '__main__':
351     parser = argparse.ArgumentParser(description='clustering')
352     parser.add_argument('--dir', help='file_dir', default=None)
353     parser.add_argument('--maxsize', help='maximum_image_size', default=600)
354
355     args = parser.parse_args()
356
357     main(args)

```

Kode A.5: util.py

```

1 """
2 Modul berisi fungsi-fungsi yang akan sering digunakan
3 """
4 #%%
5 """
6 LIBRARIES
7 """
8 import re
9 import matplotlib.pyplot as plt
10 import cv2
11
12 #untuk memuat file (gambar)
13 import os
14 from glob import glob
15
16 #untuk pemrosesan data
17 import numpy as np
18 from sklearn.cluster import AgglomerativeClustering
19 from scipy.spatial.distance import hamming
20 from sklearn.metrics import pairwise_distances
21 from sklearn.cluster import DBSCAN
22
23 #lain-lain
24 import time
25 from itertools import combinations
26
27 #%%
28 def get_image(filename, bw=True, maxheight=None, maxwidth=None):
29     """
30     Fungsi untuk memuat satu gambar dari file
31
32     Parameter:
33         filename : path dari file gambar
34         bw : apakah gambar hitam putih (grayscale)
35         maxheight : tinggi maksimal dari gambar (default = 400px)
36         maxwidth : lebar maksimal gambar (default = 400px)
37
38     #load gambar
39     img = cv2.imread(filename)
40
41     #convert warna gambar
42     if bw:
43         img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
44
45     #ubah ukuran gambar
46
47     if maxheight != None:
48         #ukuran awal
49         height = img.shape[0]
50         width = img.shape[1]
51
52         #cek apakah height > maxheight
53         if height > maxheight:
54             divider_height = maxheight / height
55             new_height = int(height * divider_height)
56             new_width = int(width * divider_height)
57             img = cv2.resize(img, (new_width, new_height))
58
59     if maxwidth != None:
60         #ukuran setelah disesuaikan tingginya
61         height = img.shape[0]
62         width = img.shape[1]
63
64         #cek apakah width > maxwidth
65         if width > maxwidth:
66             divider_width = maxwidth / width
67             new_height = int(height * divider_width)
68             new_width = int(width * divider_width)
69             img = cv2.resize(img, (new_width, new_height))
70
71     return img
72
73 def get_all_images(directory, bw=True, maxwidth=None, maxheight=None):
74     images = list()
75     for filename in glob(os.path.join(directory, '*.jpg')):
76         fname = re.split(r'\\|/', filename)[-1]
77         images.append((fname, get_image(filename, bw=bw, maxwidth=maxwidth, maxheight=maxheight)))

```

```

78
79     return images
80
81
82 def get_dataset(directory, bw=True, maxwidth=None, maxheight=None):
83     """
84     Fungsi untuk mengambil semua gambar dari dataset
85
86     Parameter:
87         directory : path berisi dataset
88         bw : apakah gambar hitam putih (grayscale)
89         maxheight : tinggi maksimal dari gambar (default = 400px)
90         maxwidth : lebar maksimal gambar (default = 400px)
91     """
92
93     imgs = dict()
94     directory_ = directory + '/*'
95     for dirname in glob(directory_):
96         group = re.split(r'\\|/', dirname)[-1]
97         for filename in glob(os.path.join(dirname, '*.jpg')):
98             fname = re.split(r'\\|/', filename)[-1]
99             if group not in imgs.keys():
100                 imgs[group] = list()
101             imgs[group].append((fname, get_image(filename, bw=bw, maxwidth=maxwidth, maxheight=maxheight)))
102
103     return imgs
104
105 def rename_dataset(directory):
106     """
107     Fungsi untuk mengubah nama pada file yang tersusun dengan format dataset.
108     format nama akan menjadi 'test_{group}_{index}.jpg'
109
110     Parameter:
111         directory : path berisi dataset
112     """
113     directory_ = directory + '/*'
114     for dirname in glob(directory_):
115         group = re.split(r'\\|/', dirname)[-1]
116         i = 0
117         for filename in glob(os.path.join(dirname, '*.jpg')):
118             new_fname = 'test_{g}_{i}.jpg'.format(g=group, i=i)
119             filepath = filename.split('/')
120             filepath[-1] = new_fname
121             new_filename = '/'.join(filepath)
122             os.rename(filename, new_filename)
123             i += 1
124
125
126
127 def aggro_cluster(dataset, n_clusters=None, distance_threshold=None, affinity='euclidean'):
128     """
129     Fungsi untuk melakukan clustering Agglomerative
130
131     Parameter:
132         dataset : data yang akan di clustering
133         n_clusters : jumlah cluster yang diinginkan
134         distance_threshold : batas untuk memisah cluster
135     """
136     array_dataset = np.array(dataset.values)
137
138     if bool(n_clusters) != bool(distance_threshold):
139         if affinity == 'hamming':
140             aggro_model = AgglomerativeClustering(
141                 n_clusters=n_clusters,
142                 distance_threshold=distance_threshold,
143                 affinity=hamming_bin_affinity,
144                 linkage='complete'
145             ).fit(array_dataset)
146         elif affinity == 'hamming_int':
147             aggro_model = AgglomerativeClustering(
148                 n_clusters=n_clusters,
149                 distance_threshold=distance_threshold,
150                 affinity=hamming_int_affinity,
151                 linkage='complete'
152             ).fit(array_dataset)
153         else:
154             aggro_model = AgglomerativeClustering(
155                 n_clusters=n_clusters,
156                 distance_threshold=distance_threshold,
157                 affinity=affinity
158             ).fit(array_dataset)
159             cluster_object = aggro_model.labels_
160             return cluster_object
161     else:
162         raise Exception('n_clusters XOR distance_threshold should return True!')
163
164 def dbscan(dataset, eps=0.5, min_pts=5, metric='euclidean'):
165     """
166     Fungsi untuk melakukan clustering DBSCAN
167
168     Parameter:
169         dataset : data yang akan di clustering
170         eps : radius minimum untuk membuat cluster
171         min_pts : minimum jumlah elemen pada radius untuk membuat cluster
172     """
173     array_dataset = np.array(dataset.values)
174     if metric == 'hamming':
175         dbscan_model = DBSCAN(
176             eps=eps,

```

```

177         min_samples=min_pts,
178         metric=hamming_int
179     ).fit(array_dataset)
180 else:
181     dbscan_model = DBSCAN(
182         eps=eps,
183         min_samples=min_pts,
184         metric=metric
185     ).fit(array_dataset)
186
187 cluster_object = dbscan_model.labels_
188 return cluster_object
189
190 def euclidean_dist(point1, point2):
191     return np.linalg.norm(point1 - point2)
192
193 def countSetBits(n):
194     count = 0
195     while (n):
196         count += n & 1
197         n >= 1
198     return count
199
200 def hamming_dist(i):
201     return countSetBits(np.bitwise_xor(int(i[0]), int(i[1])))
202
203 def hamming_int(point1, point2):
204     arr = np.array([point1, point2])
205     return np.sum(np.apply_along_axis(hamming_dist, 0, arr))
206
207 def hamming_bin(point1, point2):
208     return hamming(point1, point2) * len(point1)
209
210 def hamming_bin_affinity(X):
211     return pairwise_distances(X, metric=hamming_bin)
212
213 def hamming_int_affinity(X):
214     return pairwise_distances(X, metric=hamming_int)
215
216 def combination_distance(dataset, metric, sample=None):
217     """
218     Fungsi untuk menghitung jarak rata-rata dari tiap elemen di dataset
219
220     Parameter:
221         dataset : data yang akan dihitung rata-rata jaraknya
222         sample : jumlah sampel yang akan digunakan untuk menghitung rata-rata. Jika 'None' maka semua data digunakan
223     """
224     sample_df = dataset
225     if sample != None:
226         sample_df = dataset.sample(sample).values
227
228     dists = np.array([metric(p1, p2) for p1, p2 in combinations(sample_df, 2)])
229     return np.sum(dists) / dists.shape[0]
230
231 def chunks(lst, n):
232     """
233     Yield successive n-sized chunks from lst.
234     """
235     for i in range(0, len(lst), n):
236         yield lst[i:i + n]
237
238 def int_to_binary(desc):
239     """
240     Fungsi untuk mengubah descriptor ORB yang dihasilkan opencv (32bit integer) menjadi
241     format 256 bit binary
242
243     Parameter:
244         desc : array berukuran i x 32 berisi integer
245     """
246     binary_desc = []
247     for row in desc:
248         binary_row = []
249         for num in row:
250             binary = '{0:08b}'.format(num)
251             for i in binary:
252                 binary_row.append(int(i))
253         binary_desc.append(binary_row)
254
255     binary_desc = np.array(binary_desc, dtype='uint8')
256
257     return binary_desc
258
259 def binary_to_int(desc):
260     """
261     Fungsi untuk mengubah descriptor ORB vektor 256 bit binary menjadi 32 bit integer
262
263     Parameter:
264         desc : array berukuran i x 256 berisi data binary
265     """
266     int_desc = []
267     for row in desc:
268         int_row = []
269         chunked_row = chunks(row, 8)
270         for c in chunked_row:
271             res = 0
272             for ele in c:
273                 res = (res << 1) | ele
274             int_row.append(res)
275         int_desc.append(int_row)
276
277     int_desc = np.array(int_desc, dtype='uint8')

```

```

276     return int_desc
277
278 def medoid(arr, metric, get_index=False):
279     """
280     Fungsi untuk mencari medoid dari array berukuran n x n
281
282     Parameter:
283         desc : array 2 dimensi
284         metric : fungsi jarak yang digunakan
285     """
286     dists = pairwise_distances(arr, metric=metric)
287
288     max = np.zeros(dists.shape[0])
289     for i, v in enumerate(dists):
290         max[i] = np.sum(v)
291
292     medoid = arr[np.argmin(max)]
293
294     if get_index:
295         return np.argmin(max)
296     else:
297         return np.array([medoid])
298
299 def rotate(p, origin=(0, 0), degrees=0):
300     """
301     Fungsi untuk merotasi suatu koordinat berdasarkan suatu titik pusat
302
303     Parameter:
304         p : koordinat yang akan dirotasi
305         origin : titik pusat untuk melakukan rotasi
306         degress : jumlah seberapa besar rotasi
307     """
308
309     angle = np.deg2rad(degrees)
310     R = np.array([[np.cos(angle), -np.sin(angle)],
311                  [np.sin(angle), np.cos(angle)]])
312     o = np.atleast_2d(origin)
313     p = np.atleast_2d(p)
314
315     return np.squeeze((R @ (p.T-o.T) + o.T).T)
316
317 def rgb_to_bgr(r, g, b):
318     return (b, g, r)
319
320 def show_keypoints(image, keypoints, color=(0,255,0), flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT, name=False, return_img=False):
321     """
322     Fungsi untuk melakukan visualisasi keypoint pada gambar
323
324     Parameter:
325         image : gambar yang digunakan untuk menampilkan
326         keypoints : tuple berisi keypoint
327         color : warna keypoint yang ditampilkan
328         name : nama file jika gambar disimpan
329     """
330     keypoints_image = cv2.drawKeypoints(image, keypoints, image, color=color, flags=flags)
331
332     if return_img:
333         return keypoints_image
334
335     if name:
336         cv2.imwrite(name, keypoints_image)
337         cv2.imshow('Keypoint', keypoints_image)
338         cv2.waitKey(0)
339
340 def show_matches(img1, kp1, img2, kp2, name=False, return_img=False):
341     """
342     Fungsi untuk menampilkan pasangan keypoint pada dua gambar
343
344     Parameter:
345         img1 : gambar pertama
346         kp1 : keypoint untuk gambar pertama
347         img2 : gambar kedua
348         kp2 : keypoint untuk gambar kedua
349         name : nama file jika gambar disimpan
350     """
351     matches = list()
352     for i in range(len(kp1)):
353         matches.append(cv2.DMatch(i, i, 0))
354
355     img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, flags=2)
356
357     if return_img:
358         return img_matches
359
360     if name:
361         cv2.imwrite(name, img_matches)
362         cv2.imshow('Keypoint_Matches', img_matches)
363         cv2.waitKey()
364
365 def put_text(img,
366             text,
367             org,
368             font=cv2.FONT_HERSHEY_SIMPLEX,
369             fontScale=1,
370             color=(0, 255, 0),
371             thickness=1):
372     """
373     Fungsi untuk menambahkan text pada gambar
374     """

```

```

375     img = cv2.putText(img, text, org, font, fontScale, color, thickness, cv2.LINE_AA)
376     return img
377
378 def show_polygon(img, kp, name=False, return_img=False):
379     """
380     Fungsi untuk menampilkan polygon yang menyambungkan keypoint-keypoint pada gambar
381
382     Parameter:
383         img      : gambar yang digunakan untuk menampilkan
384         kp       : tuple berisi keypoint
385         name    : nama file jika gambar disimpan
386
387     polygon = np.array([[i.pt[0], i.pt[1]] for i in kp], np.int32)
388     img_mod = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
389     img_mod = cv2.polylines(img_mod, polygon, True, (255, 255, 255), 2)
390     for i, p in enumerate(kp):
391         img_mod = put_text(img_mod, str(i), (int(p.pt[0]), int(p.pt[1])), fontScale=0.7)
392
393     if return_img:
394         return img_mod
395
396     if name:
397         cv2.imwrite(name, img_mod)
398         cv2.imshow('Shapes', img_mod)
399         cv2.waitKey()
400
401 def display_image(img, cmap='viridis'):
402     plt.axis('off')
403     plt.imshow(img, cmap=cmap)
404
405 """
406 Fungsi-fungsi untuk melakukan modifikasi pada gambar, digunakan pada pembuatan data untuk test
407 """
408 def rotate_image(image, angle):
409     image_center = tuple(np.array(image.shape[1::-1]) / 2)
410     rot_mat = cv2.getRotationMatrix2D(image_center, -angle, 1.0)
411     result = cv2.warpAffine(image, rot_mat, image.shape[1::-1], flags=cv2.INTER_LINEAR)
412     return result
413
414 def zoom_center(image, zoom_factor=1.5):
415     y_size = image.shape[0]
416     x_size = image.shape[1]
417
418     # define new boundaries
419     x1 = int(0.5 * x_size * (1 - 1 / zoom_factor))
420     x2 = int(x_size - 0.5 * x_size * (1 - 1 / zoom_factor))
421     y1 = int(0.5 * y_size * (1 - 1 / zoom_factor))
422     y2 = int(y_size - 0.5 * y_size * (1 - 1 / zoom_factor))
423
424     # first crop image then scale
425     img_cropped = image[y1:y2, x1:x2]
426     return cv2.resize(img_cropped, None, fx=zoom_factor, fy=zoom_factor)

```

Kode A.6: bsis_test.py

```

1 # %%
2 #load library
3 import pandas as pd
4 import cv2
5 import time
6 import pickle
7
8 from modules.bsis import BSIS
9 from modules import util
10 from modules.clustermodel import ClusterModel
11
12 from dplyr import *
13
14 # %% [markdown]
15 # #### Parameters
16
17 # %%
18 dataset_name = ''
19 clustermodel_dir = ''
20 testdata_dir = ''
21
22 # %%
23 #load train cluster model
24 cluster_model = ClusterModel()
25 cluster_model.load(clustermodel_dir)
26 maxsize = cluster_model.meta.maxsize
27
28 if cluster_model.meta.desc_type == ClusterModel.Descriptor.SIFT:
29     extract_method = cv2.SIFT_create()
30     algorithm = BSIS.FLANN_INDEX_KDTREE
31 elif cluster_model.meta.desc_type == ClusterModel.Descriptor.ORB:
32     extract_method = cv2.ORB_create()
33     algorithm = BSIS.FLANN_INDEX_LSH
34
35 bsis_param = dict(
36     num_rotation=20,
37     algorithm=algorithm,
38     k=100,
39     t=3
40 )
41
42 #load test data

```

```

43| query_dataset = util.get_dataset(testdata_dir, maxwidth=maxsize, maxheight=maxsize)
44|
45| img_class = cluster_model.get_dataframe().drop_duplicates(subset=['img'])[['img', 'img_class']]
46| img_class = img_class.set_index('img')
47|
48| # %% [markdown]
49| # ### Non-filtered
50|
51| # %%
52| #get train data
53| train_data = cluster_model.get_local_features(min_consistency=0.0, min_uniqueness=0.0)
54|
55| q_name = list()
56| q_class = list()
57| most_similar = list()
58| most_similar_class = list()
59| total_weight = list()
60| same_class_idx = list()
61| same_class_weight = list()
62| extract_time = list()
63| pairing_time = list()
64| total_bsisc_time = list()
65| bsisc_list = list()
66|
67| # %%
68| for k, v in query_dataset.items():
69|     for i in v:
70|         q_name.append(i[0])
71|         start_time = time.time()
72|         kp, desc = extract_method.detectAndCompute(i[1], None)
73|         ext_time = time.time() - start_time
74|         extract_time.append(ext_time)
75|
76|         print(i[0])
77|         print('feature_extract---', ext_time)
78|
79|         query = {'kp': kp, 'desc': desc}
80|
81|         bsisc = BSIS(query)
82|         bsisc.set_train_data(train_data)
83|
84|         maximum = bsisc.run(**bsisc_param)
85|         most_similar.append(maximum)
86|         if maximum != '':
87|             most_similar_class.append(img_class.loc[maximum, 'img_class'])
88|             total_weight.append(bsisc.result[maximum]['total_weight'])
89|         else:
90|             most_similar_class.append('')
91|             total_weight.append(0)
92|
93|         sorted_result = [(idx, k, v['total_weight']) for idx, (k, v) in enumerate(sorted(bsisc.result.items(), key=lambda item:
94|             item[1]['total_weight'], reverse=True))]
95|         for idx, n, w in sorted_result:
96|             if n != '':
97|                 if img_class.loc[n, 'img_class'] == k:
98|                     same_class_idx.append(idx)
99|                     same_class_weight.append(w)
100|                     break
101|             else:
102|                 same_class_idx.append('')
103|                 same_class_weight.append('')
104|
105|         q_class.append(k)
106|         pairing_time.append(bsisc.pairing_time)
107|         total_bsisc_time.append(bsisc.total_time)
108|         bsisc_list.append(bsisc)
109|         print('')
110|
111| # save bsisc_list
112| bsisc_list_tupled = list()
113| for b in bsisc_list:
114|     bsisc_list_tupled.append([(idx, k, v['total_weight']) for idx, (k, v) in enumerate(sorted(b.result.items(), key=lambda item:
115|             item[1]['total_weight'], reverse=True))])
116| pickle.dump(bsisc_list_tupled, open(f'{dataset_name}_{cluster_model.meta.desc_type.value}_{maxsize}_full.pkl', 'wb'))
117|
118| # %%
119| df_result = pd.DataFrame()
120| df_result['q_name'] = q_name
121| df_result['q_class'] = q_class
122| df_result['most_similar'] = most_similar
123| df_result['most_similar_class'] = most_similar_class
124| df_result['total_weight'] = total_weight
125| df_result['same_class_idx'] = same_class_idx
126| df_result['same_class_weight'] = same_class_weight
127| df_result['extract_time'] = extract_time
128| df_result['pairing_time'] = pairing_time
129| df_result['total_bsisc_time'] = total_bsisc_time
130|
131| is_true = list()
132| total = 0
133| for q, t in zip(q_class, most_similar):
134|     if t != '':
135|         if q == img_class.loc[t, 'img_class']:
136|             total += 1
137|             is_true.append(1)
138|         else:
139|             is_true.append(0)

```

```

140     else:
141         is_true.append(0)
142 print(total)
143 df_result['is_true'] = is_true
144 df_result.to_csv(f'{dataset_name}_{cluster_model.meta.desc_type.value}_full.csv', index=False)
145
146 # %% [markdown]
147 # ### Filtered
148
149 # %%
150 train_data_filtered = cluster_model.get_local_features(min_consistency=0.3, min_uniqueness=0.3)
151
152 q_name_filtered = list()
153 q_class_filtered = list()
154 most_similar_filtered = list()
155 most_similar_class_filtered = list()
156 total_weight_filtered = list()
157 same_class_idx_filtered = list()
158 same_class_weight_filtered = list()
159 extract_time_filtered = list()
160 pairing_time_filtered = list()
161 total_bsisc_time_filtered = list()
162 bsisc_list_filtered = list()
163
164 # %%
165 for k, v in query_dataset.items():
166     for i in v:
167         q_name_filtered.append(i[0])
168         start_time = time.time()
169         kp, desc = extract_method.detectAndCompute(i[1], None)
170         ext_time = time.time() - start_time
171         extract_time_filtered.append(ext_time)
172
173     print(i[0])
174     print('feature_extract---', ext_time)
175
176     query = {'kp': kp, 'desc': desc}
177
178     bsisc = BSIS(query)
179     bsisc.set_train_data(train_data_filtered)
180
181     maximum = bsisc.run(**bsisc_param)
182     most_similar_filtered.append(maximum)
183     if maximum != '':
184         most_similar_class_filtered.append(img_class.loc[maximum, 'img_class'])
185         total_weight_filtered.append(bsisc.result[maximum]['total_weight'])
186     else:
187         most_similar_class_filtered.append('')
188         total_weight_filtered.append(0)
189
190     sorted_result = [(idx, k, v['total_weight']) for idx, (k, v) in enumerate(sorted(bsisc.result.items(), key=lambda item: item[1]['total_weight'], reverse=True))]
191     for idx, n, w in sorted_result:
192         if n != '':
193             if img_class.loc[n, 'img_class'] == k:
194                 same_class_idx_filtered.append(idx)
195                 same_class_weight_filtered.append(w)
196                 break
197             else:
198                 same_class_idx_filtered.append('')
199                 same_class_weight_filtered.append('')
200
201     q_class_filtered.append(k)
202     pairing_time_filtered.append(bsisc.pairing_time)
203     total_bsisc_time_filtered.append(bsisc.total_time)
204     bsisc_list_filtered.append(bsisc)
205     print('')
206
207 # %%
208 #save bsisc_list
209 bsisc_list_tupled_filtered = list()
210 for b in bsisc_list_filtered:
211     bsisc_list_tupled_filtered.append([(idx, k, v['total_weight']) for idx, (k, v) in enumerate(sorted(b.result.items(), key=lambda item: item[1]['total_weight'], reverse=True))])
212
213 pickle.dump(bsisc_list_tupled_filtered, open(f'{dataset_name}_{cluster_model.meta.desc_type.value}_filtered.pkl', 'wb'))
214
215 # %%
216 df_result_filtered = pd.DataFrame()
217 df_result_filtered['q_name'] = q_name_filtered
218 df_result_filtered['q_class'] = q_class_filtered
219 df_result_filtered['most_similar'] = most_similar_filtered
220 df_result_filtered['most_similar_class'] = most_similar_class_filtered
221 df_result_filtered['total_weight'] = total_weight_filtered
222 df_result_filtered['same_class_idx'] = same_class_idx_filtered
223 df_result_filtered['same_class_weight'] = same_class_weight_filtered
224 df_result_filtered['extract_time'] = extract_time_filtered
225 df_result_filtered['pairing_time'] = pairing_time_filtered
226 df_result_filtered['total_bsisc_time'] = total_bsisc_time_filtered
227
228 is_true = list()
229 total = 0
230 for q, t in zip(q_class_filtered, most_similar_filtered):
231     if t != '':
232         if q == img_class.loc[t, 'img_class']:
233             total += 1
234             is_true.append(1)
235         else:
236             is_true.append(0)

```

```

237     else:
238         is_true.append(0)
239     print(total)
240
241 df_result_filtered['is_true'] = is_true
242 df_result_filtered.to_csv(f'{dataset_name}_{cluster_model.meta.desc_type.value}_filtered.csv', index=False)

```

Kode A.7: threshold_scoring.py

```

1 #%%
2 from modules.clustermodel import ClusterModel
3 from modules import util
4 from bs4 import BeautifulSoup
5 import pandas as pd
6
7 #%%
8 #Fungsi untuk membuat dictionary berisi batas-batas objek target bagi tiap gambar
9 def create_target(directory, name):
10     target = []
11     xml_name = name.split('.')[0] + '.xml'
12     with open(f'{directory}/{xml_name}', 'rb') as file:
13         ann = BeautifulSoup(file.read(), 'xml')
14         object = ann.findAll('object')
15         for o in object:
16             bndbox = o.find('bndbox')
17             target.append({
18                 'xmin': int(bndbox.find('xmin').string),
19                 'xmax': int(bndbox.find('xmax').string),
20                 'ymin': int(bndbox.find('ymin').string),
21                 'ymax': int(bndbox.find('ymax').string)
22             })
23
24     return target
25
26 #Fungsi untuk menentukan apakah sebuah titik masuk dalam target
27 def determine(point_x, point_y, targets):
28     for t in targets:
29         if (point_x >= t['xmin'] and
30             point_x <= t['xmax'] and
31             point_y >= t['ymin'] and
32             point_y <= t['ymax']):
33             return True
34     else:
35         return False
36
37 def divide(x, y):
38     if y == 0:
39         return 0
40     else:
41         return (x / y)
42
43 #Fungsi menghitung score, dengan cara membagi jumlah titik di dalam target dengan total jumlah titik
44 def calculate_score(row):
45     return divide(row['inside_target'], (row['inside_target'] + row['outside_target']))
46
47 #%%
48 directory = 'datasets/annotated_gsv'
49 images = util.get_dataset(directory)
50
51 targets = {}
52 for k, v in images.items():
53     for img in v:
54         img_dir = f'{directory}/{k}'
55         targets[img[0]] = create_target(img_dir, img[0])
56
57 #targets = {i[0]: create_target(directory, i[0]) for i in images}
58
59 #%%
60 poi_cm = ClusterModel()
61 poi_cm.load('result/gsv_sift_agglo_400.cm')
62
63 #%%
64 results = {i: [] for i in images.keys()}
65
66 for i in range(6, 11):
67     for k, v in images.items():
68         names = []
69         inside_target = []
70         outside_target = []
71
72         for name, img in v:
73             try:
74                 lf = poi_cm.get_local_features(min_uniqueness=i/10, img=name)
75                 it = 0
76                 ot = 0
77                 for l in lf:
78                     point_x = l.keypoint.pt[0]
79                     point_y = l.keypoint.pt[1]
80                     if (determine(point_x, point_y, targets[name])):
81                         it += 1
82                     else:
83                         ot += 1
84
85                     names.append(name)
86                     inside_target.append(it)
87                     outside_target.append(ot)
88             except:

```

```

89         names.append(name)
90         inside_target.append(0)
91         outside_target.append(0)
92
93     df_score = dict(
94         {
95             'img': names,
96             'inside_target': inside_target,
97             'outside_target': outside_target,
98         }
99     )
100    results[k].append(df_score)
101
102
103 # %%
104 img = list()
105 img_class = list()
106 threshold = list()
107 inside_target = list()
108 outside_target = list()
109
110 for k, v in results.items():
111     for t, i in enumerate(v):
112         img = img + i['img']
113         img_class = img_class + ([k.split('_')[1]] * 10)
114         threshold = threshold + [(0.5 + ((t + 1) / 10)) * 10]
115         inside_target = inside_target + i['inside_target']
116         outside_target = outside_target + i['outside_target']
117
118 # %%
119 df_results = pd.DataFrame({
120     'img': img,
121     'img_class': img_class,
122     'threshold': threshold,
123     'inside_target': inside_target,
124     'outside_target': outside_target
125 })
126
127 df_results['score'] = df_results.apply(calculate_score, axis=1)

```

Kode A.8: server.py

```

1  #- Libraries
2  from flask import Flask, request, jsonify, render_template
3  from flask_cors import CORS
4  import os
5
6  #-- Routing Files --
7  from routes.app import app_routes
8
9  #-- Flask init --
10 app = Flask(__name__)
11 app.config['UPLOAD_FOLDER'] = './static/uploads'
12 cors = CORS(app)
13
14 #-- Routes --
15 @app.route('/app<url>', methods=['GET', 'POST'])
16 def apps(url):
17     return app_routes(url, app, request)
18
19 if __name__ == '__main__':
20     app.run(host='localhost', port=1113)

```

Kode A.9: app.py

```

1  from os import listdir
2  from os.path import join
3  from werkzeug.utils import secure_filename
4  from flask import jsonify
5  from routes.detect_image import detect_image
6  import json
7
8  #-- Methods --
9  def listmodels():
10     models_list = [m for m in listdir('./static/models')]
11     response = jsonify(models_list)
12     return response
13
14 def save_image(file):
15     fname = file.filename
16     save_dir = join(app.config['UPLOAD_FOLDER'], secure_filename(fname))
17     file.save(save_dir)
18     return save_dir
19
20 def detect():
21     global _req
22     global _app
23     img_dir = save_image(_req.files['file'])
24     param = json.loads(_req.form['param'])
25
26     response = jsonify(detect_image(param['model'], float(param['consistency']), float(param['uniqueness']), img_dir))
27
28     return response
29

```

```

30| #--Routes--
31 routes = {
32   'models': listmodels,
33   'detect': detect
34 }
35
36 #--app & request object-
37 _app = None
38 _req = None
39
40 def app_routes(url, app, req):
41   global routes
42   global _req
43   global _app
44
45   _req = req
46   _app = app
47
48   return routes[url]()

```

Kode A.10: detect_image.py

```

1 from modules.bsis import BSIS
2 from modules.clustermodel import ClusterModel
3 from modules import util
4 import cv2
5 import time
6 import os
7
8 models_dir = './static/models/'
9
10 def detect_image(model, consistency, uniqueness, img_dir):
11   global models_dir
12
13   #load ClusterModel
14   cm = ClusterModel()
15   cm.load(f'{models_dir}{model}')
16
17   #get train_data
18   train_data = cm.get_local_features(min_consistency=consistency, min_uniqueness=uniqueness)
19
20   #load image
21   img_name = img_dir.split('/')[-1]
22   img = util.get_image(img_dir, maxheight=600, maxwidth=600)
23
24   #detect keypoints
25   sift = cv2.SIFT_create()
26   kp, desc = sift.detectAndCompute(img, None)
27   query = {
28     'kp': kp,
29     'desc': desc
30   }
31
32   #BSIS
33   bsis = BSIS(query)
34   bsis.set_train_data(train_data)
35   most_similar = bsis.run(num_rotation=20, algorithm=BSIS.FLANN_INDEX.KDTREE, k=100, t=3)
36
37   #Result
38   res = list()
39   for k, v in sorted(bsis.result.items(), key=lambda i: i[1]['total_weight'], reverse=True)[:10]:
40     train_img_class = k.split('.')[0]
41     train_img_dir = f'./static/dataset/poi/{train_img_class}/{k}'
42     train_img = util.get_image(train_img_dir, maxheight=600, maxwidth=600)
43
44     img_matches = util.show_matches(img, v['query_kp'], train_img, v['train_kp'], return_img=True)
45     img_matches_name = f'{time.time()}.jpg'
46     img_matches_dir = f'./static/detection/{img_matches_name}'
47     cv2.imwrite(img_matches_dir, img_matches)
48
49     if v['total_weight'] > 0:
50       res.append((k, round(v['total_weight'], 3), img_matches_name))
51     else:
52       break
53
54   return res

```

Kode A.11: App.js

```

1 import './App.css'
2 import { useRef, useState } from 'react'
3
4 import ModelPicker from './components/model_picker'
5 import ImagePicker from './components/image_picker'
6 import ImageDisplay from './components/image_display'
7 import ResultList from './components/result_list'
8
9 import url from './config/urls'
10
11 import axios from 'axios'
12
13 function App() {
14   const uniqueness = useRef(0)
15   const consistency = useRef(0)

```

```

16| const model = useRef(null)
17| const image_upload = useRef(null)
18|
19| const [imageSrc, setImageSrc] = useState('./img_placeholder.png')
20| const [imageName, setImageName] = useState('img_placeholder.png')
21| const [totalWeight, setTotalWeight] = useState(0.0)
22|
23|
24| const data = ['img_placeholder.png', 0.0, 'img_placeholder.png']
25| const [resultList, setResultList] = useState([data])
26|
27| const changeUniqueness = (x) => {
28|   uniqueness.current = x
29| }
30|
31| const changeConsistency = (x) => {
32|   consistency.current = x
33| }
34|
35| const changeModel = (x) => {
36|   model.current = x
37| }
38|
39| const changeImage = (x) => {
40|   image_upload.current = x
41| }
42|
43| const _detect = () => {
44|   if (model.current != null & image_upload.current != null) {
45|     let detect_param = {
46|       uniqueness: uniqueness.current,
47|       consistency: consistency.current,
48|       model: model.current
49|     }
50|     let formData = new FormData()
51|     formData.append('param', JSON.stringify(detect_param))
52|     formData.append('file', image_upload.current)
53|     axios.post(url.app.detect, formData)
54|       .then((res) => {
55|         setResultList(res.data)
56|       })
57|     } else {
58|       alert('Image not selected!')
59|     }
60|   }
61|
62| const changeImageDisplay = (data) => {
63|   setImageName(data[0])
64|   setTotalWeight(data[1])
65|   setImageSrc(`${url.BASE_URL}/static/detection/${data[2]}`)
66| }
67|
68| return (
69|   <div className="container-fluid_p-3">
70|     <div className='row'>
71|       <div className="col-3">
72|         <div className="container-fluid">
73|           <ImagePicker changeImage={changeImage} />
74|         </div>
75|         <div className="container-fluid_mt-3">
76|           <ModelPicker changeConsistency={changeConsistency} changeUniqueness={changeUniqueness} changeModel={changeModel} />
77|         </div>
78|         <div className="container-fluid_mt-3">
79|           <button className="btn btn-primary" onClick={_detect}>Detect</button>
80|         </div>
81|       </div>
82|       <div className="col-9">
83|         <div className="container-fluid">
84|           <ImageDisplay imageName={imageName} image_src={imageSrc} total_weight={totalWeight} />
85|         </div>
86|         <ResultList result_list={resultList} changeImageDisplay={changeImageDisplay}/>
87|       </div>
88|     </div>
89|   )
90|
91| }
92|
93| export default App

```

Kode A.12: image_picker.js

```

1 import React, { useEffect, useMemo, useState } from "react"
2
3
4 const ImagePicker = ({ changeImage }) => {
5   const [image, setImage] = useState(null)
6
7   const onImageChange = (event) => {
8     setImage(event.target.files[0])
9   }
10
11   const image_src = useMemo(() => {
12     if(image != null) {
13       return URL.createObjectURL(image)
14     } else {
15       return '/img_placeholder.png'
16     }

```

```

17 }, [image])
18
19 useEffect(() => {
20     changeImage(image)
21 }, [image])
22
23
24 return (
25     <div className="container_p-0">
26         <div className="container_border_rounded">
27             <img src={image_src} className="img-fluid" />
28         </div>
29         <div className="container_mt-1">
30             <div className="border_border-2_border-primary_rounded_p-1">
31                 <input type="file" accept="image/*" onChange={onImageChange} />
32             </div>
33         </div>
34     </div>
35 )
36
37 export default ImagePicker

```

Kode A.13: model_picker.js

```

1 import React, { useState, useEffect } from "react"
2 import axios from "axios"
3 import RangeSlider from "./range_slider"
4 import url from '../config/urls'
5
6 const ModelPicker = ({ changeUniqueness, changeConsistency, changeModel }) => {
7     const [models, setModels] = useState([])
8
9     const _retrieveModels = () => {
10         axios.get(url.app.models)
11             .then((res) => {
12                 setModels(res.data)
13                 changeModel(res.data[0])
14             })
15     }
16
17     useEffect(() => {
18         _retrieveModels()
19     }, [])
20
21     const pickModel = (event) => {
22         changeModel(event.target.value)
23     }
24
25     return(
26         <div className="container">
27             <div className="row">
28                 <select className="form-select" onChange={pickModel}>
29                     {
30                         models.map((m) => {
31                             return <option value={m} key={m}>{m}</option>
32                         })
33                     }
34                 </select>
35             </div>
36             <div className="row">
37                 <div>
38                     <span className="fw-lighter">Min. Uniqueness</span>
39                     <RangeSlider changeFunction={changeUniqueness}/>
40                 </div>
41                 <div>
42                     <span className="fw-lighter">Min. Consistency</span>
43                     <RangeSlider changeFunction={changeConsistency}/>
44                 </div>
45             </div>
46         </div>
47     )
48 }
49
50 export default ModelPicker

```

Kode A.14: range_slider.js

```

1 import React, { useState, useEffect } from "react"
2
3 const RangeSlider = ({ changeFunction }) => {
4     const [sliderValue, setSliderValue] = useState(0.0)
5
6     const updateSlider = (event) => {
7         setSliderValue(event.target.value)
8     }
9
10    useEffect(() => {
11        changeFunction(sliderValue)
12    }, [sliderValue])
13
14    return (
15        <div className="container">
16            <div className="row">

```

```

17         <div className="col-9">
18             <input className="form-range" type="range" value={sliderValue} min={0} max={1} step={0.1} onChange={
19                 updateSlider}/>
20         </div>
21         <div className="col-auto">
22             {sliderValue}
23         </div>
24     </div>
25   )
26 }
27
28 export default RangeSlider;

```

Kode A.15: result_list.js

```

1 import React, { useEffect, useMemo, useState } from "react"
2 import ImageItem from "./image_item"
3
4 const ResultList = ({ result_list, changeImageDisplay }) => {
5     const [selection, setSelection] = useState(Array(10).fill(false))
6
7     const changeSelection = (i) => {
8         const arr = Array(10).fill(false)
9         arr[i] = true
10        changeImageDisplay(result_list[i])
11        setSelection(arr)
12    }
13
14    useEffect(() => {
15        changeSelection(0)
16    }, [result_list])
17
18    const imageItems = useMemo(() => {
19        let items = []
20        for(let i = 0; i < result_list.length; i++) {
21            let data = {
22                image_name: result_list[i][0],
23                total_weight: result_list[i][1]
24            }
25            items.push(<ImageItem key={i} data={data} is_selected={selection[i]} clickFunction={() => {changeSelection(i)}}/>)
26        }
27        return items
28    }, [selection, result_list])
29
30    return (
31        <div>
32            {imageItems}
33        </div>
34    )
35 }
36
37 export default ResultList

```

Kode A.16: image_item.js

```

1 import React from "react"
2
3 const ImageItem = ({ data, is_selected, clickFunction }) => {
4     const style = {
5         minWidth: '12vw'
6     }
7     return(
8         <button className={"btn btn-light float-start p-3 m-1 border rounded text-start"} + (is_selected ? "border-primary": "")>
9             <div style={style} onClick={clickFunction}>
10                 <div className="h5">
11                     {data.image_name}
12                 </div>
13                 <div className="fw-light h5">
14                     {data.total_weight}
15                 </div>
16             </button>
17     )
18 }
19
20 export default ImageItem

```

Kode A.17: image_display.js

```

1 import React from "react"
2
3 const ImageDisplay = ({ image_src, image_name, total_weight }) => {
4     return (
5         <div className="row">
6             <div className="col">
7                 <div className="container-fluid p-0">
8                     <div className="row_border rounded">
9                         <img src={image_src} className="img-fluid" alt="detection_result" style={{maxHeight: '600px'}} />
10                     </div>
11                 <div>
12                     <div className="float-start pe-4">
13

```

```
14         <div className="h3_fw-light">{image_name}</div>
15     </div>
16     <div className="float-start">
17         <div className="h3_fw-light">{total_weight}</div>
18     </div>
19     </div>
20   </div>
21 </div>
22 }
23 }
24
25 export default ImageDisplay
```

LAMPIRAN B
HASIL EKSPERIMEN

Tabel B.1: Hasil pengujian menggunakan metode SIFT pada keseluruhan *dataset* Book Covers 400.

q name	q class	most simi-lar	most simi-lar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
001-test0.jpg	1	001-Droid.jpg	1	629.1482423	0	629.1482423	0.020999908	1.66706109	1.869061708	1
001-test1.jpg	1	001-Droid.jpg	1	462.2304691	0	462.2304691	0.018012285	1.62799859	1.779994249	1
002-test0.jpg	2	002-iPhone.jpg	2	235.1784678	0	235.1784678	0.01901722	1.600989819	1.702990055	1
002-test1.jpg	2	002-iPhone.jpg	2	316.3429108	0	316.3429108	0.017002821	1.630995035	1.749991417	1
003-test0.jpg	3	003-Droid.jpg	3	514.3578947	0	514.3578947	0.017990112	1.638990402	1.825973511	1
003-test1.jpg	3	003-iPhone.jpg	3	499.8033857	0	499.8033857	0.020001173	1.62799263	1.808002949	1
004-test0.jpg	4	004-iPhone.jpg	4	412.1667971	0	412.1667971	0.018993378	1.592974424	1.644951344	1
004-test1.jpg	4	004-iPhone.jpg	4	669.9447951	0	669.9447951	0.018012285	1.655979395	1.810938835	1
005-test0.jpg	5	005-Canon.jpg	5	1513.946756	0	1513.946756	0.019009352	1.725572348	2.38758111	1
005-test1.jpg	5	005-Canon.jpg	5	1128.99708	0	1128.99708	0.018998861	1.666008711	2.348004341	1
006-test0.jpg	6	006-iPhone.jpg	6	1933.492973	0	1933.492973	0.021981478	1.702003717	3.221021414	1
006-test1.jpg	6	006-iPhone.jpg	6	1785.255445	0	1785.255445	0.021981478	1.751653671	3.152661562	1
007-test0.jpg	7	007-Droid.jpg	7	170.8073706	0	170.8073706	0.016974211	1.604986906	1.655986309	1
007-test1.jpg	7	007-iPhone.jpg	7	375.0744195	0	375.0744195	0.017005682	1.639015675	1.748992205	1

009-test0.jpg	9	009-iPhone.jpg	9	1128.24997	0	1128.24997	0.020996332	1.70400548	2.024987459	1
009-test1.jpg	9	009-iPhone.jpg	9	1050.010436	0	1050.010436	0.019004822	1.67299366	1.956977844	1
010-test0.jpg	10	010-Canon.jpg	10	1793.274797	0	1793.274797	0.019004107	1.676999807	2.552014589	1
010-test1.jpg	10	010-iPhone.jpg	10	1626.399453	0	1626.399453	0.016977787	1.638997555	2.267994881	1
011-test0.jpg	11	011-iPhone.jpg	11	1079.717575	0	1079.717575	0.019015074	1.606989622	1.843010426	1
011-test1.jpg	11	011-iPhone.jpg	11	1038.987015	0	1038.987015	0.016997337	1.621496201	1.945498466	1
012-test0.jpg	12	012-iPhone.jpg	12	357.2669118	0	357.2669118	0.016001701	1.574014664	1.64601469	1
012-test1.jpg	12	012-iPhone.jpg	12	313.6730546	0	313.6730546	0.015980482	1.627986908	1.697963476	1
013-test0.jpg	13	013-iPhone.jpg	13	1349.212479	0	1349.212479	0.0210042	1.800995827	2.427969217	1
013-test1.jpg	13	013-iPhone.jpg	13	1263.281877	0	1263.281877	0.025012255	1.711386919	2.209384918	1
014-test0.jpg	14	014-iPhone.jpg	14	321.6551445	0	321.6551445	0.01699543	1.61383605	1.72683692	1
014-test1.jpg	14	014-iPhone.jpg	14	247.6194718	0	247.6194718	0.016001225	1.589011669	1.6539886	1
015-test0.jpg	15	015-Droid.jpg	15	2114.121739	0	2114.121739	0.027004957	1.840943575	3.654950142	1
015-test1.jpg	15	015-Droid.jpg	15	1794.243877	0	1794.243877	0.023991346	1.829413652	3.005905867	1
016-test0.jpg	16	016-iPhone.jpg	16	170.7826473	0	170.7826473	0.017974854	1.664693594	1.75869298	1
016-test1.jpg	16			0			0.01700592	1.572638273	1.575620413	0

017-test0.jpg	17	017-Droid.jpg	17	490.4708703	0	490.4708703	0.019995928	1.602246761	1.767015219	1
017-test1.jpg	17	017-Droid.jpg	17	291.698331	0	291.698331	0.017989874	1.679250002	1.768274069	1
018-test0.jpg	18	018-iPhone.jpg	18	301.8914014	0	301.8914014	0.018356085	1.628308535	1.763267994	1
018-test1.jpg	18	018-iPhone.jpg	18	426.1995341	0	426.1995341	0.020209789	1.61888504	1.80045104	1
019-test0.jpg	19	019-Canon.jpg	19	1315.748052	0	1315.748052	0.020997524	1.653012276	2.138672829	1
019-test1.jpg	19	019-iPhone.jpg	19	1231.244953	0	1231.244953	0.02099824	1.653544426	2.016552925	1
020-test0.jpg	20	020-iPhone.jpg	20	1087.484949	0	1087.484949	0.018995762	1.605420351	2.023594379	1
020-test1.jpg	20	020-Canon.jpg	20	472.9404691	0	472.9404691	0.016001225	1.576728582	1.711958408	1
021-test0.jpg	21	021-iPhone.jpg	21	248.4978424	0	248.4978424	0.016739845	1.561736822	1.624257326	1
021-test1.jpg	21	021-Droid.jpg	21	361.5828857	0	361.5828857	0.016980648	1.596224785	1.660448551	1
022-test0.jpg	22	022-Droid.jpg	22	1022.270699	0	1022.270699	0.013317108	1.596787214	1.923198223	1
022-test1.jpg	22	022-iPhone.jpg	22	552.073247	0	552.073247	0.016011238	1.57073164	1.741191387	1
023-test0.jpg	23	023-Canon.jpg	23	264.5396352	0	264.5396352	0.016614437	1.566115141	1.616274357	1
023-test1.jpg	23	023-Canon.jpg	23	124.5112244	0	124.5112244	0.01395154	1.562011242	1.602916002	1
024-test0.jpg	24	024-5800.jpg	24	518.0491998	0	518.0491998	0.017981052	1.5962286	1.772688627	1
024-test1.jpg	24	024-5800.jpg	24	506.5280904	0	506.5280904	0.021013021	1.6267941	1.881604671	1

025-test0.jpg	25	025-iPhone.jpg	25	462.2310678	0	462.2310678	0.017819881	1.574960709	1.702238798	1
025-test1.jpg	25	025-Droid.jpg	25	464.3225153	0	464.3225153	0.021068811	1.595582008	1.770852566	1
026-test0.jpg	26	026-Droid.jpg	26	629.1707388	0	629.1707388	0.017988443	1.593266249	1.770131588	1
026-test1.jpg	26	026-Droid.jpg	26	477.9364708	0	477.9364708	0.017001867	1.559127092	1.706781149	1
027-test0.jpg	27	027-iPhone.jpg	27	476.0214833	0	476.0214833	0.015984535	1.578643799	1.658835411	1
027-test1.jpg	27	027-iPhone.jpg	27	626.6190111	0	626.6190111	0.016990662	1.567123175	1.645207644	1
028-test0.jpg	28	028-Droid.jpg	28	948.5672495	0	948.5672495	0.016980886	1.589864969	1.826461315	1
028-test1.jpg	28	028-Droid.jpg	28	768.7422833	0	768.7422833	0.016977549	1.599676847	1.811889172	1
029-test0.jpg	29	029-iPhone.jpg	29	1052.687518	0	1052.687518	0.017638922	1.622413397	1.999554873	1
029-test1.jpg	29	029-iPhone.jpg	29	766.2375529	0	766.2375529	0.018983603	1.642974854	1.903253794	1
030-test0.jpg	30	030-iPhone.jpg	30	555.9032437	0	555.9032437	0.017045259	1.611602306	1.740703106	1
030-test1.jpg	30	030-iPhone.jpg	30	768.0234726	0	768.0234726	0.017007113	1.590816259	1.750802994	1
031-test0.jpg	31	031-iPhone.jpg	31	141.5908071	0	141.5908071	0.01600194	1.567952633	1.609950066	1
031-test1.jpg	31	031-iPhone.jpg	31	198.6532835	0	198.6532835	0.015676498	1.567567825	1.622554064	1
032-test0.jpg	32	032-Droid.jpg	32	935.782129	0	935.782129	0.015993357	1.675002098	1.930001736	1
032-test1.jpg	32	032-Droid.jpg	32	1139.945645	0	1139.945645	0.015993118	1.608032703	1.910010576	1

033-test0.jpg	33	033-iPhone.jpg	33	413.9265172	0	413.9265172	0.017014265	1.632004738	1.775011539	1
033-test1.jpg	33	033-iPhone.jpg	33	421.4921973	0	421.4921973	0.016979694	1.608967066	1.779976368	1
034-test0.jpg	34	034-iPhone.jpg	34	594.4908681	0	594.4908681	0.015978575	1.571964979	1.727971554	1
034-test1.jpg	34	034-iPhone.jpg	34	106.0453352	0	106.0453352	0.013991356	1.552006721	1.583005667	1
035-test0.jpg	35	035-iPhone.jpg	35	1052.901605	0	1052.901605	0.016990423	1.633987665	1.923996925	1
035-test1.jpg	35	035-iPhone.jpg	35	957.6471877	0	957.6471877	0.016996145	1.609004736	1.861004353	1
036-test0.jpg	36	036-Canon.jpg	36	122.2464755	0	122.2464755	0.013986826	1.549962759	1.589977503	1
036-test1.jpg	36	036-Droid.jpg	36	115.659195	0	115.659195	0.017977953	1.58100009	1.619000435	1
037-test0.jpg	37	037-iPhone.jpg	37	578.7615901	0	578.7615901	0.017990828	1.625978231	1.841956139	1
037-test1.jpg	37	037-iPhone.jpg	37	1345.60445	0	1345.60445	0.019005299	1.62196207	1.934978008	1
038-test0.jpg	38	038-Canon.jpg	38	935.6859586	0	935.6859586	0.017980814	1.629044771	2.048003435	1
038-test1.jpg	38	038-Canon.jpg	38	376.0736741	0	376.0736741	0.021980047	1.651995659	1.860974789	1
039-test0.jpg	39	039-iPhone.jpg	39	411.4752564	0	411.4752564	0.015984297	1.592015982	1.686995029	1
039-test1.jpg	39	039-iPhone.jpg	39	558.011455	0	558.011455	0.016988516	1.600021362	1.743998051	1
040-test0.jpg	40	040-iPhone.jpg	40	799.360344	0	799.360344	0.01900506	1.618003845	1.870996952	1
040-test1.jpg	40	040-iPhone.jpg	40	788.3093195	0	788.3093195	0.020000935	1.661987066	2.028906822	1

041-test0.jpg	41	041-Droid.jpg	41	1485.358294	0	1485.358294	0.018995047	1.637999773	2.039008856	1
041-test1.jpg	41	041-Droid.jpg	41	1509.766318	0	1509.766318	0.018002033	1.602015495	2.077000856	1
042-test0.jpg	42	042-Canon.jpg	42	1168.736298	0	1168.736298	0.015978336	1.587997437	2.062023401	1
042-test1.jpg	42	042-iPhone.jpg	42	601.5708298	0	601.5708298	0.01697135	1.586987019	1.890987635	1
043-test0.jpg	43	043-iPhone.jpg	43	614.1568468	0	614.1568468	0.016989946	1.581024647	1.706024885	1
043-test1.jpg	43	043-iPhone.jpg	43	356.8270365	0	356.8270365	0.015972853	1.555027723	1.617029428	1
044-test0.jpg	44	044-iPhone.jpg	44	935.3572401	0	935.3572401	0.020972252	1.592017174	1.934005737	1
044-test1.jpg	44	044-iPhone.jpg	44	579.5403746	0	579.5403746	0.019979954	1.569980383	1.757974625	1
045-test0.jpg	45	045-Droid.jpg	45	1484.637932	0	1484.637932	0.016991138	1.603990316	2.442987204	1
045-test1.jpg	45	045-iPhone.jpg	45	689.6182002	0	689.6182002	0.01699543	1.589994669	1.777032137	1
046-test0.jpg	46	046-Droid.jpg	46	1528.538984	0	1528.538984	0.004967213	1.583981514	1.944018126	1
046-test1.jpg	46	046-Droid.jpg	46	982.1736602	0	982.1736602	0.007971048	1.599972963	2.145986319	1
047-test0.jpg	47	047-iPhone.jpg	47	951.0266603	0	951.0266603	0.01854372	1.584995031	1.747021437	1
047-test1.jpg	47	047-iPhone.jpg	47	393.7409637	0	393.7409637	0.016978979	1.563986301	1.628012657	1
048-test0.jpg	48	048-iPhone.jpg	48	418.5264285	0	418.5264285	0.015977859	1.564958811	1.610996008	1
048-test1.jpg	48	048-iPhone.jpg	48	279.405548	0	279.405548	0.015983105	1.570970058	1.605006933	1

049-test0.jpg	49	049-iPhone.jpg	49	1895.894374	0	1895.894374	0.017974138	1.625997543	2.313031912	1
049-test1.jpg	49	049-iPhone.jpg	49	1267.748451	0	1267.748451	0.016971827	1.583997965	2.034013748	1
050-test0.jpg	50	050-Canon.jpg	50	412.7992409	0	412.7992409	0.030996084	1.746997356	2.141008139	1
050-test1.jpg	50	050-Canon.jpg	50	151.0068071	0	151.0068071	0.026039124	1.692551613	1.945575953	1
051-test0.jpg	51	051-Canon.jpg	51	944.5893061	0	944.5893061	0.009985924	1.605973244	1.832997084	1
051-test1.jpg	51	051-iPhone.jpg	51	747.3832234	0	747.3832234	0.006985426	1.604995012	1.901009321	1

Tabel B.2: Hasil pengujian menggunakan metode SIFT pada *dataset* Book Covers 400 yang telah tersaring.

q name	q class	most simi-lar	most simi-lar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
001-test0.jpg	1	001-Droid.jpg	1	232.3877	0	232.3877	0.019025	0.385011	0.555011	1
001-test1.jpg	1	001-Droid.jpg	1	130.4338	0	130.4338	0.019991	0.369988	0.487	1
002-test0.jpg	2	002-iPhone.jpg	2	127.1147	0	127.1147	0.014978	0.34398	0.418987	1
002-test1.jpg	2	002-Canon.jpg	2	88.68993	0	88.68993	0.019004	0.347	0.415976	1
003-test0.jpg	3	003-iPhone.jpg	3	162.5516	0	162.5516	0.018996	0.363967	0.494963	1
003-test1.jpg	3	003-iPhone.jpg	3	146.0346	0	146.0346	0.020005	0.365014	0.500014	1
004-test0.jpg	4	046-Droid.jpg	46	10.58084	13	0	0.017989	0.335976	0.370977	0
004-test1.jpg	4	004-iPhone.jpg	4	22.65011	0	22.65011	0.021015	0.376999	0.487024	1
005-test0.jpg	5	005-Droid.jpg	5	751.0404	0	751.0404	0.021985	0.385988	0.676985	1
005-test1.jpg	5	005-Droid.jpg	5	549.178	0	549.178	0.020007	0.394985	0.689516	1
006-test0.jpg	6	006-iPhone.jpg	6	1195.401	0	1195.401	0.020992	0.42501	1.00001	1
006-test1.jpg	6	006-iPhone.jpg	6	1133.922	0	1133.922	0.020992	0.417012	0.931988	1
007-test0.jpg	7	007-iPhone.jpg	7	16.24197	0	16.24197	0.018989	0.345015	0.406013	1
007-test1.jpg	7	007-iPhone.jpg	7	149.0801	0	149.0801	0.017989	0.353024	0.431023	1

009-test0.jpg	9	009-iPhone.jpg	9	420.6978	0	420.6978	0.01897	0.386991	0.580024	1
009-test1.jpg	9	009-iPhone.jpg	9	362.384	0	362.384	0.019977	0.380028	0.558028	1
010-test0.jpg	10	010-Canon.jpg	10	1082.255	0	1082.255	0.018978	0.387024	0.806998	1
010-test1.jpg	10	010-iPhone.jpg	10	1072.623	0	1072.623	0.018001	0.381996	0.744542	1
011-test0.jpg	11	011-iPhone.jpg	11	365.5422	0	365.5422	0.018995	0.346988	0.463974	1
011-test1.jpg	11	011-iPhone.jpg	11	468.5829	0	468.5829	0.019973	0.35003	0.49103	1
012-test0.jpg	12	012-iPhone.jpg	12	137.4502	0	137.4502	0.015972	0.327997	0.365974	1
012-test1.jpg	12	012-iPhone.jpg	12	150.0378	0	150.0378	0.017989	0.336017	0.378017	1
013-test0.jpg	13	013-iPhone.jpg	13	580.046	0	580.046	0.021977	0.395708	0.681525	1
013-test1.jpg	13	013-iPhone.jpg	13	472.205	0	472.205	0.021991	0.418012	0.665038	1
014-test0.jpg	14	014-Droid.jpg	14	212.172	0	212.172	0.017977	0.335749	0.431738	1
014-test1.jpg	14	014-iPhone.jpg	14	134.5958	0	134.5958	0.016001	0.330976	0.386976	1
015-test0.jpg	15	015-Droid.jpg	15	1720.524	0	1720.524	0.025993	0.51497	1.665979	1
015-test1.jpg	15	015-Droid.jpg	15	1391.828	0	1391.828	0.026002	0.492011	1.331013	1
016-test0.jpg	16	016-iPhone.jpg	16	131.4724	0	131.4724	0.01797	0.35003	0.454998	1
016-test1.jpg	16			0	1	0	0.017001	0.328997	0.336023	0

017-test0.jpg	17	017-Droid.jpg	17	248.1868	0	248.1868	0.017977	0.344975	0.478971	1
017-test1.jpg	17	017-iPhone.jpg	17	230.5367	0	230.5367	0.015996	0.345029	0.411001	1
018-test0.jpg	18	018-iPhone.jpg	18	273.34	0	273.34	0.019004	0.356019	0.528031	1
018-test1.jpg	18	018-iPhone.jpg	18	322.3829	0	322.3829	0.017967	0.358999	0.533984	1
019-test0.jpg	19	019-Canon.jpg	19	733.1827	0	733.1827	0.019996	0.387006	0.655998	1
019-test1.jpg	19	019-Canon.jpg	19	628.4362	0	628.4362	0.020998	0.383834	0.604806	1
020-test0.jpg	20	020-Canon.jpg	20	521.8628	0	521.8628	0.018997	0.360992	0.572003	1
020-test1.jpg	20	020-Canon.jpg	20	316.3547	0	316.3547	0.018986	0.335976	0.414974	1
021-test0.jpg	21	021-5800.jpg	21	24.56904	0	24.56904	0.017993	0.335984	0.385014	1
021-test1.jpg	21	021-5800.jpg	21	24.16446	0	24.16446	0.01599	0.337003	0.380015	1
022-test0.jpg	22	022-iPhone.jpg	22	256.8743	0	256.8743	0.019989	0.360963	0.519975	1
022-test1.jpg	22	022-iPhone.jpg	22	168.8548	0	168.8548	0.016977	0.340999	0.421009	1
023-test0.jpg	23	043-Droid.jpg	43	12.31625	18	0	0.016968	0.341971	0.395988	0
023-test1.jpg	23	023-Canon.jpg	23	13.42624	0	13.42624	0.017978	0.337999	0.381011	1
024-test0.jpg	24	024-5800.jpg	24	184.7422	0	184.7422	0.01997	0.360989	0.490996	1
024-test1.jpg	24	024-5800.jpg	24	260.325	0	260.325	0.019992	0.38	0.585013	1

025-test0.jpg	25	025-Droid.jpg	25	126.3387	0	126.3387	0.018989	0.34603	0.421988	1
025-test1.jpg	25	025-Canon.jpg	25	171.4103	0	171.4103	0.018003	0.362998	0.488996	1
026-test0.jpg	26	026-Droid.jpg	26	149.0424	0	149.0424	0.019995	0.363012	0.50498	1
026-test1.jpg	26	026-Droid.jpg	26	204.6774	0	204.6774	0.017006	0.340037	0.418025	1
027-test0.jpg	27	027-iPhone.jpg	27	96.23894	0	96.23894	0.017989	0.341012	0.392012	1
027-test1.jpg	27	027-iPhone.jpg	27	167.2905	0	167.2905	0.017977	0.343976	0.404977	1
028-test0.jpg	28	028-Droid.jpg	28	217.8351	0	217.8351	0.018	0.367029	0.486017	1
028-test1.jpg	28	028-Droid.jpg	28	190.0879	0	190.0879	0.018989	0.365963	0.461972	1
029-test0.jpg	29	029-Canon.jpg	29	322.6078	0	322.6078	0.020004	0.373023	0.572023	1
029-test1.jpg	29	029-Droid.jpg	29	258.4214	0	258.4214	0.020967	0.368987	0.547012	1
030-test0.jpg	30	030-iPhone.jpg	30	345.2881	0	345.2881	0.016969	0.337013	0.416988	1
030-test1.jpg	30	030-iPhone.jpg	30	436.0825	0	436.0825	0.019005	0.343179	0.430976	1
031-test0.jpg	31	048-5800.jpg	48	12.00357	14	0	0.018001	0.333988	0.377	0
031-test1.jpg	31	051-iPhone.jpg	51	21.33729	15	0	0.016967	0.33998	0.387978	0
032-test0.jpg	32	032-iPhone.jpg	32	458.9125	0	458.9125	0.018002	0.347013	0.468013	1
032-test1.jpg	32	032-Droid.jpg	32	592.8894	0	592.8894	0.015968	0.345998	0.469963	1

033-test0.jpg	33	044-iPhone.jpg	44	23.10951	1	20.41346	0.019013	0.380008	0.51901	0
033-test1.jpg	33	006-iPhone.jpg	6	35.56878	1	35.47428	0.017984	0.381965	0.514997	0
034-test0.jpg	34	034-iPhone.jpg	34	306.06	0	306.06	0.016969	0.34299	0.445001	1
034-test1.jpg	34			0	1	0	0.016969	0.329997	0.342992	0
035-test0.jpg	35	035-iPhone.jpg	35	440.0771	0	440.0771	0.019002	0.366013	0.511	1
035-test1.jpg	35	035-Canon.jpg	35	375.0055	0	375.0055	0.01699	0.355996	0.466998	1
036-test0.jpg	36	036-iPhone.jpg	36	86.00871	0	86.00871	0.016006	0.330035	0.358	1
036-test1.jpg	36	036-Droid.jpg	36	80.0039	0	80.0039	0.016011	0.324025	0.357	1
037-test0.jpg	37	037-iPhone.jpg	37	363.4992	0	363.4992	0.02	0.376	0.589971	1
037-test1.jpg	37	037-iPhone.jpg	37	633.8103	0	633.8103	0.019004	0.381006	0.615	1
038-test0.jpg	38	038-iPhone.jpg	38	369.0493	0	369.0493	0.019992	0.381998	0.623011	1
038-test1.jpg	38	038-iPhone.jpg	38	174.4071	0	174.4071	0.02199	0.386973	0.60401	1
039-test0.jpg	39	039-iPhone.jpg	39	214.941	0	214.941	0.016966	0.334985	0.413996	1
039-test1.jpg	39	039-iPhone.jpg	39	259.1395	0	259.1395	0.017968	0.343991	0.456012	1
040-test0.jpg	40	040-iPhone.jpg	40	303.0223	0	303.0223	0.017979	0.350503	0.469517	1
040-test1.jpg	40	040-iPhone.jpg	40	540.5188	0	540.5188	0.017979	0.364989	0.530988	1

041-test0.jpg	41	041-Droid.jpg	41	414.0009	0	414.0009	0.017992	0.350991	0.476967	1
041-test1.jpg	41	041-Droid.jpg	41	457.3757	0	457.3757	0.019001	0.359987	0.497012	1
042-test0.jpg	42	042-Canon.jpg	42	825.7304	0	825.7304	0.017977	0.351975	0.649965	1
042-test1.jpg	42	042-Droid.jpg	42	381.5028	0	381.5028	0.019013	0.349	0.550012	1
043-test0.jpg	43	043-Droid.jpg	43	193.0744	0	193.0744	0.018977	0.339993	0.410991	1
043-test1.jpg	43	043-iPhone.jpg	43	176.8952	0	176.8952	0.016999	0.329183	0.373008	1
044-test0.jpg	44	044-Droid.jpg	44	571.1126	0	571.1126	0.021006	0.355	0.520966	1
044-test1.jpg	44	044-Droid.jpg	44	235.2054	0	235.2054	0.020007	0.351023	0.441035	1
045-test0.jpg	45	045-Droid.jpg	45	833.1329	0	833.1329	0.019977	0.374027	0.735027	1
045-test1.jpg	45	045-Droid.jpg	45	310.5167	0	310.5167	0.017973	0.339	0.427975	1
046-test0.jpg	46	046-Droid.jpg	46	1137.204	0	1137.204	0.005996	0.35097	0.556002	1
046-test1.jpg	46	046-Droid.jpg	46	864.7995	0	864.7995	0.006966	0.359023	0.690011	1
047-test0.jpg	47	047-iPhone.jpg	47	284.1589	0	284.1589	0.019892	0.349993	0.456018	1
047-test1.jpg	47	047-iPhone.jpg	47	55.52329	0	55.52329	0.018966	0.336024	0.371997	1
048-test0.jpg	48	048-iPhone.jpg	48	64.97981	0	64.97981	0.017993	0.324977	0.352982	1
048-test1.jpg	48	048-iPhone.jpg	48	52.85404	0	52.85404	0.014991	0.323975	0.343975	1

049-test0.jpg	49	049-Droid.jpg	49	922.1406	0	922.1406	0.018996	0.373987	0.693988	1
049-test1.jpg	49	049-Droid.jpg	49	677.4326	0	677.4326	0.019002	0.356	0.598433	1
050-test0.jpg	50	050-Canon.jpg	50	228.614	0	228.614	0.031994	0.484009	0.870003	1
050-test1.jpg	50	050-Canon.jpg	50	91.65449	0	91.65449	0.029	0.443979	0.732	1
051-test0.jpg	51	051-Canon.jpg	51	249.9822	0	249.9822	0.008968	0.352987	0.423987	1
051-test1.jpg	51	051-Canon.jpg	51	335.0506	0	335.0506	0.008992	0.356034	0.457032	1

Tabel B.3: Hasil pengujian menggunakan metode SIFT pada keseluruhan *dataset* Book Covers 600.

q name	q class	most simi-lar	most simi-lar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
001-test0.jpg	1	001-Droid.jpg	1	1008.878	0	1008.878	0.029005	2.646774	2.917914	1
001-test1.jpg	1	001-Droid.jpg	1	657.8173	0	657.8173	0.028586	2.644469	2.843122	1
002-test0.jpg	2	002-iPhone.jpg	2	303.2822	0	303.2822	0.025229	2.623021	2.716293	1
002-test1.jpg	2	002-iPhone.jpg	2	224.5355	0	224.5355	0.024546	2.596861	2.691894	1
003-test0.jpg	3	003-Droid.jpg	3	563.4376	0	563.4376	0.02553	2.588651	2.777034	1
003-test1.jpg	3	003-Canon.jpg	3	607.5375	0	607.5375	0.025571	2.610729	2.816157	1
004-test0.jpg	4	004-iPhone.jpg	4	392.0435	0	392.0435	0.022155	2.541632	2.601216	1
004-test1.jpg	4	004-iPhone.jpg	4	812.4728	0	812.4728	0.024479	2.63147	2.840397	1
005-test0.jpg	5	005-Droid.jpg	5	2331.732	0	2331.732	0.024984	2.632551	3.856704	1
005-test1.jpg	5	005-Canon.jpg	5	1682.929	0	1682.929	0.02496	2.63268	3.843215	1
006-test0.jpg	6	006-Canon.jpg	6	2283.783	0	2283.783	0.027978	2.675496	5.354849	1
006-test1.jpg	6	006-Canon.jpg	6	2169.879	0	2169.879	0.027992	2.704978	5.219052	1
007-test0.jpg	7	007-iPhone.jpg	7	279.2355	0	279.2355	0.023976	2.562552	2.65554	1
007-test1.jpg	7	007-Canon.jpg	7	557.5786	0	557.5786	0.024989	2.609213	2.780117	1

009-test0.jpg	9	009-iPhone.jpg	9	1952.955	0	1952.955	0.027839	2.67584	3.498353	1
009-test1.jpg	9	009-iPhone.jpg	9	1975.833	0	1975.833	0.024118	2.613074	3.241032	1
010-test0.jpg	10	010-Canon.jpg	10	2591.133	0	2591.133	0.027304	2.648475	4.030515	1
010-test1.jpg	10	010-Canon.jpg	10	1792.886	0	1792.886	0.029807	2.596758	3.582265	1
011-test0.jpg	11	011-iPhone.jpg	11	1486.418	0	1486.418	0.025323	2.549638	2.893324	1
011-test1.jpg	11	011-iPhone.jpg	11	1145.626	0	1145.626	0.02534	2.539768	2.919146	1
012-test0.jpg	12	012-iPhone.jpg	12	409.5207	0	409.5207	0.024869	2.52433	2.61332	1
012-test1.jpg	12	012-iPhone.jpg	12	389.5285	0	389.5285	0.023004	2.540711	2.629566	1
013-test0.jpg	13	013-iPhone.jpg	13	1804.781	0	1804.781	0.027988	2.656383	3.823611	1
013-test1.jpg	13	013-iPhone.jpg	13	1707.552	0	1707.552	0.028	2.661267	3.61034	1
014-test0.jpg	14	014-Canon.jpg	14	328.9795	0	328.9795	0.022998	2.527616	2.626124	1
014-test1.jpg	14	014-iPhone.jpg	14	174.5504	0	174.5504	0.022567	2.520232	2.567178	1
015-test0.jpg	15	015-Droid.jpg	15	3531.26	0	3531.26	0.032974	2.803507	9.269831	1
015-test1.jpg	15	015-Droid.jpg	15	4705.806	0	4705.806	0.031156	2.80298	7.452904	1
016-test0.jpg	16	016-iPhone.jpg	16	158.3398	0	158.3398	0.024619	2.574901	2.667125	1
016-test1.jpg	16			0			0.022	2.514632	2.518618	0

017-test0.jpg	17	017-Droid.jpg	17	561.0308	0	561.0308	0.025988	2.567749	2.775338	1
017-test1.jpg	17	017-Canon.jpg	17	253.9426	0	253.9426	0.02299	2.547386	2.64239	1
018-test0.jpg	18	018-iPhone.jpg	18	406.4891	0	406.4891	0.025983	2.572263	2.739299	1
018-test1.jpg	18	018-iPhone.jpg	18	567.9574	0	567.9574	0.023401	2.575772	2.816226	1
019-test0.jpg	19	019-Canon.jpg	19	1539.85	0	1539.85	0.027993	2.582576	3.18923	1
019-test1.jpg	19	019-Canon.jpg	19	1483.546	0	1483.546	0.023981	2.602741	3.019543	1
020-test0.jpg	20	020-iPhone.jpg	20	1104.762	0	1104.762	0.026695	2.577266	3.086588	1
020-test1.jpg	20	020-Canon.jpg	20	521.5358	0	521.5358	0.023064	2.53827	2.687811	1
021-test0.jpg	21	021-Droid.jpg	21	427.9708	0	427.9708	0.022983	2.542229	2.613231	1
021-test1.jpg	21	021-Droid.jpg	21	519.9392	0	519.9392	0.023	2.529131	2.619389	1
022-test0.jpg	22	022-Droid.jpg	22	1320.937	0	1320.937	0.022002	2.544266	3.098594	1
022-test1.jpg	22	022-Canon.jpg	22	596.9799	0	596.9799	0.022011	2.543648	2.816836	1
023-test0.jpg	23	023-Canon.jpg	23	205.394	0	205.394	0.027984	2.587474	2.656496	1
023-test1.jpg	23	023-iPhone.jpg	23	80.93094	0	80.93094	0.020981	2.521769	2.566191	1
024-test0.jpg	24	024-5800.jpg	24	438.2392	0	438.2392	0.033975	2.571781	2.769104	1
024-test1.jpg	24	024-5800.jpg	24	622.8986	0	622.8986	0.027995	2.606506	2.934941	1

025-test0.jpg	25	025-Droid.jpg	25	566.8126	0	566.8126	0.027986	2.554564	2.756027	1
025-test1.jpg	25	025-Droid.jpg	25	764.811	0	764.811	0.030832	2.562635	2.807552	1
026-test0.jpg	26	026-Droid.jpg	26	979.3026	0	979.3026	0.026988	2.555748	2.781731	1
026-test1.jpg	26	026-Droid.jpg	26	721.5943	0	721.5943	0.025257	2.51737	2.685901	1
027-test0.jpg	27	027-iPhone.jpg	27	562.6839	0	562.6839	0.021982	2.686381	2.773623	1
027-test1.jpg	27	027-iPhone.jpg	27	826.7414	0	826.7414	0.022572	2.576941	2.68145	1
028-test0.jpg	28	028-Droid.jpg	28	1198.815	0	1198.815	0.024208	2.6058	3.00844	1
028-test1.jpg	28	028-5800.jpg	28	981.2772	0	981.2772	0.031069	2.596828	2.98157	1
029-test0.jpg	29	029-iPhone.jpg	29	1545.489	0	1545.489	0.026011	2.584537	3.151733	1
029-test1.jpg	29	029-iPhone.jpg	29	1216.066	0	1216.066	0.023991	2.627527	2.918506	1
030-test0.jpg	30	030-Droid.jpg	30	505.3344	0	505.3344	0.023997	2.524821	2.664552	1
030-test1.jpg	30	030-Droid.jpg	30	673.3566	0	673.3566	0.023998	2.529975	2.719852	1
031-test0.jpg	31	031-iPhone.jpg	31	154.4256	0	154.4256	0.022993	2.532489	2.586972	1
031-test1.jpg	31	031-iPhone.jpg	31	121.947	0	121.947	0.02198	2.519937	2.57271	1
032-test0.jpg	32	032-Droid.jpg	32	912.6652	0	912.6652	0.022985	2.564428	2.953651	1
032-test1.jpg	32	032-Canon.jpg	32	1295.207	0	1295.207	0.018931	2.536609	2.966717	1

033-test0.jpg	33	033-iPhone.jpg	33	529.1241	0	529.1241	0.02246	2.584203	2.820216	1
033-test1.jpg	33	033-5800.jpg	33	693.7353	0	693.7353	0.023986	2.640167	2.954489	1
034-test0.jpg	34	034-iPhone.jpg	34	607.2756	0	607.2756	0.021987	2.536062	2.72941	1
034-test1.jpg	34	034-Canon.jpg	34	141.1449	0	141.1449	0.018992	2.548935	2.579073	1
035-test0.jpg	35	035-iPhone.jpg	35	1443.325	0	1443.325	0.027198	2.620984	3.073404	1
035-test1.jpg	35	035-iPhone.jpg	35	1199.712	0	1199.712	0.024998	2.58335	2.966578	1
036-test0.jpg	36	036-Canon.jpg	36	193.7126	0	193.7126	0.022983	2.534262	2.614494	1
036-test1.jpg	36	036-Canon.jpg	36	129.1033	0	129.1033	0.020002	2.522072	2.574223	1
037-test0.jpg	37	037-Canon.jpg	37	670.3124	0	670.3124	0.022992	2.627594	2.896826	1
037-test1.jpg	37	037-iPhone.jpg	37	1696.19	0	1696.19	0.029452	2.682593	3.094529	1
038-test0.jpg	38	038-Canon.jpg	38	1670.437	0	1670.437	0.025991	2.701118	3.483377	1
038-test1.jpg	38	038-Canon.jpg	38	730.7935	0	730.7935	0.022257	2.670805	2.964128	1
039-test0.jpg	39	039-iPhone.jpg	39	292.9869	0	292.9869	0.024003	2.591558	2.676521	1
039-test1.jpg	39	039-iPhone.jpg	39	502.3632	0	502.3632	0.022989	2.595233	2.75382	1
040-test0.jpg	40	040-iPhone.jpg	40	787.1199	0	787.1199	0.022677	2.584494	2.886027	1
040-test1.jpg	40	040-Droid.jpg	40	786.8032	0	786.8032	0.023716	2.61996	3.116122	1

041-test0.jpg	41	041-Droid.jpg	41	1521.703	0	1521.703	0.022001	2.562227	3.003076	1
041-test1.jpg	41	041-Droid.jpg	41	1828.803	0	1828.803	0.023994	2.606921	3.244892	1
042-test0.jpg	42	042-Canon.jpg	42	1007.735	0	1007.735	0.020987	2.541367	2.98444	1
042-test1.jpg	42	042-iPhone.jpg	42	745.2634	0	745.2634	0.022001	2.600764	2.976365	1
043-test0.jpg	43	043-iPhone.jpg	43	694.4509	0	694.4509	0.021994	2.613083	2.78077	1
043-test1.jpg	43	043-iPhone.jpg	43	454.326	0	454.326	0.021994	2.560803	2.652247	1
044-test0.jpg	44	044-iPhone.jpg	44	1119.157	0	1119.157	0.026519	2.626619	3.046797	1
044-test1.jpg	44	044-iPhone.jpg	44	633.4043	0	633.4043	0.030666	2.618272	2.861459	1
045-test0.jpg	45	045-iPhone.jpg	45	1575.195	0	1575.195	0.02699	2.657723	3.728981	1
045-test1.jpg	45	045-iPhone.jpg	45	691.3251	0	691.3251	0.023994	2.629022	2.867119	1
046-test0.jpg	46	046-Droid.jpg	46	1333.548	0	1333.548	0.003074	2.614168	2.994488	1
046-test1.jpg	46	046-Canon.jpg	46	974.6754	0	974.6754	0.006995	2.67864	3.255859	1
047-test0.jpg	47	047-iPhone.jpg	47	1146.802	0	1146.802	0.024001	2.55679	2.759603	1
047-test1.jpg	47	047-iPhone.jpg	47	794.3396	0	794.3396	0.022522	2.51593	2.595945	1
048-test0.jpg	48	048-iPhone.jpg	48	228.726	0	228.726	0.019951	2.550658	2.593652	1
048-test1.jpg	48	048-iPhone.jpg	48	237.4743	0	237.4743	0.019983	2.567601	2.600608	1

049-test0.jpg	49	049-iPhone.jpg	49	1705.661	0	1705.661	0.024071	2.628047	3.297538	1
049-test1.jpg	49	049-iPhone.jpg	49	1367.834	0	1367.834	0.026009	2.655339	3.168519	1
050-test0.jpg	50	050-Canon.jpg	50	663.7977	0	663.7977	0.048262	2.761298	3.225827	1
050-test1.jpg	50	050-Canon.jpg	50	193.5959	0	193.5959	0.042975	2.736582	3.029025	1
051-test0.jpg	51	051-Canon.jpg	51	879.0476	0	879.0476	0.007004	2.582263	2.798209	1
051-test1.jpg	51	051-Canon.jpg	51	880.9015	0	880.9015	0.007002	2.594021	2.887813	1

Tabel B.4: Hasil pengujian menggunakan metode SIFT pada *dataset* Book Covers 600 yang telah tersaring.

q name	q class	most simi-lar	most simi-lar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
001-test0.jpg	1	001-Droid.jpg	1	283.5246	0	283.5246	0.032006	0.648237	0.817751	1
001-test1.jpg	1	001-Droid.jpg	1	207.6495	0	207.6495	0.029573	0.629159	0.75982	1
002-test0.jpg	2	002-Droid.jpg	2	166.2228	0	166.2228	0.027187	0.58234	0.65944	1
002-test1.jpg	2	002-iPhone.jpg	2	126.2681	0	126.2681	0.025623	0.598063	0.662577	1
003-test0.jpg	3	003-Canon.jpg	3	175.9875	0	175.9875	0.026443	0.616944	0.734532	1
003-test1.jpg	3	003-Droid.jpg	3	209.9821	0	209.9821	0.024183	0.618942	0.764685	1
004-test0.jpg	4	015-iPhone.jpg	15	9.307289	10	0	0.024769	0.59062	0.641178	0
004-test1.jpg	4	004-Droid.jpg	4	37.34017	0	37.34017	0.028239	0.635201	0.808134	1
005-test0.jpg	5	005-Droid.jpg	5	1013.724	0	1013.724	0.025407	0.643848	0.977824	1
005-test1.jpg	5	005-Droid.jpg	5	891.7088	0	891.7088	0.024377	0.643618	0.996853	1
006-test0.jpg	6	006-Canon.jpg	6	1424.279	0	1424.279	0.031845	0.713064	1.486875	1
006-test1.jpg	6	006-Canon.jpg	6	1196.832	0	1196.832	0.028388	0.703556	1.421654	1
007-test0.jpg	7	007-iPhone.jpg	7	85.49716	0	85.49716	0.025118	0.582585	0.634959	1
007-test1.jpg	7	007-Canon.jpg	7	119.9637	0	119.9637	0.027652	0.609286	0.703154	1

009-test0.jpg	9	009-Droid.jpg	9	1092.078	0	1092.078	0.025171	0.661521	1.05228	1
009-test1.jpg	9	009-iPhone.jpg	9	983.6772	0	983.6772	0.025609	0.632349	0.940179	1
010-test0.jpg	10	010-Canon.jpg	10	1914.767	0	1914.767	0.023705	0.653555	1.270543	1
010-test1.jpg	10	010-Canon.jpg	10	1465.357	0	1465.357	0.022003	0.623719	1.048598	1
011-test0.jpg	11	011-iPhone.jpg	11	625.8513	0	625.8513	0.024978	0.582463	0.720618	1
011-test1.jpg	11	011-iPhone.jpg	11	642.2525	0	642.2525	0.025963	0.587401	0.733443	1
012-test0.jpg	12	012-iPhone.jpg	12	140.2773	0	140.2773	0.022989	0.566506	0.612294	1
012-test1.jpg	12	012-5800.jpg	12	152.2017	0	152.2017	0.022002	0.56595	0.608939	1
013-test0.jpg	13	013-iPhone.jpg	13	810.3532	0	810.3532	0.026378	0.665172	1.063264	1
013-test1.jpg	13	013-iPhone.jpg	13	610.3872	0	610.3872	0.02597	0.685894	1.042355	1
014-test0.jpg	14	014-iPhone.jpg	14	146.5219	0	146.5219	0.021993	0.570698	0.645634	1
014-test1.jpg	14	014-iPhone.jpg	14	137.7128	0	137.7128	0.021985	0.570147	0.622147	1
015-test0.jpg	15	015-Droid.jpg	15	3282.338	0	3282.338	0.031698	0.822073	6.030129	1
015-test1.jpg	15	015-Droid.jpg	15	4162.791	0	4162.791	0.03274	0.80066	4.378031	1
016-test0.jpg	16	016-iPhone.jpg	16	87.85676	0	87.85676	0.021979	0.596512	0.708459	1
016-test1.jpg	16			0			0.022977	0.552787	0.552787	0

017-test0.jpg	17	017-Droid.jpg	17	283.2579	0	283.2579	0.023015	0.581706	0.713265	1
017-test1.jpg	17	017-Droid.jpg	17	143.6688	0	143.6688	0.025342	0.569414	0.636776	1
018-test0.jpg	18	018-iPhone.jpg	18	375.4892	0	375.4892	0.023973	0.601723	0.777379	1
018-test1.jpg	18	018-iPhone.jpg	18	390.3717	0	390.3717	0.023619	0.608521	0.804947	1
019-test0.jpg	19	019-Canon.jpg	19	897.3853	0	897.3853	0.023989	0.635363	0.984532	1
019-test1.jpg	19	019-Canon.jpg	19	761.6047	0	761.6047	0.023753	0.634884	0.912606	1
020-test0.jpg	20	020-iPhone.jpg	20	492.9788	0	492.9788	0.023988	0.606782	0.844722	1
020-test1.jpg	20	020-Canon.jpg	20	266.9555	0	266.9555	0.021989	0.572425	0.660241	1
021-test0.jpg	21	021-5800.jpg	21	48.92633	0	48.92633	0.020982	0.572505	0.617712	1
021-test1.jpg	21	021-Droid.jpg	21	80.62909	0	80.62909	0.02399	0.576615	0.635612	1
022-test0.jpg	22	022-Droid.jpg	22	412.1163	0	412.1163	0.022978	0.620684	0.838129	1
022-test1.jpg	22	022-Canon.jpg	22	155.3836	0	155.3836	0.021003	0.576929	0.643165	1
023-test0.jpg	23	023-Canon.jpg	23	52.89357	0	52.89357	0.022193	0.575121	0.640122	1
023-test1.jpg	23	023-Canon.jpg	23	24.5838	0	24.5838	0.021991	0.575344	0.621042	1
024-test0.jpg	24	024-5800.jpg	24	229.8717	0	229.8717	0.02941	0.61511	0.777046	1
024-test1.jpg	24	024-5800.jpg	24	432.2148	0	432.2148	0.026384	0.629686	0.865678	1

025-test0.jpg	25	025-Droid.jpg	25	204.6619	0	204.6619	0.026003	0.594755	0.665735	1
025-test1.jpg	25	025-Droid.jpg	25	190.8702	0	190.8702	0.02738	0.595278	0.706275	1
026-test0.jpg	26	026-Droid.jpg	26	315.2763	0	315.2763	0.025774	0.607547	0.750535	1
026-test1.jpg	26	026-Droid.jpg	26	320.8266	0	320.8266	0.022953	0.579074	0.656672	1
027-test0.jpg	27	047-iPhone.jpg	47	21.09001	11	0	0.021924	0.574218	0.633596	0
027-test1.jpg	27	041-iPhone.jpg	41	15.99274	6	0	0.022559	0.573249	0.609562	0
028-test0.jpg	28	028-Canon.jpg	28	346.9484	0	346.9484	0.033726	0.635642	0.806265	1
028-test1.jpg	28	028-Canon.jpg	28	480.7047	0	480.7047	0.024979	0.625572	0.803148	1
029-test0.jpg	29	029-iPhone.jpg	29	466.6782	0	466.6782	0.027521	0.622956	0.889076	1
029-test1.jpg	29	029-Canon.jpg	29	265.7829	0	265.7829	0.025009	0.629369	0.811147	1
030-test0.jpg	30	030-Droid.jpg	30	267.2531	0	267.2531	0.017214	0.584158	0.665139	1
030-test1.jpg	30	030-Droid.jpg	30	347.6749	0	347.6749	0.023999	0.59381	0.692452	1
031-test0.jpg	31	046-Droid.jpg	46	20.20224	1	16.15326	0.021989	0.571866	0.629059	0
031-test1.jpg	31	046-Canon.jpg	46	11.06073	7	0	0.030433	0.58016	0.625379	0
032-test0.jpg	32	032-Droid.jpg	32	565.37	0	565.37	0.023988	0.589816	0.73391	1
032-test1.jpg	32	032-Droid.jpg	32	725.27	0	725.27	0.022972	0.591944	0.742535	1

033-test0.jpg	33	033-5800.jpg	33	108.4689	0	108.4689	0.022004	0.638911	0.774845	1
033-test1.jpg	33	033-iPhone.jpg	33	104.952	0	104.952	0.022758	0.648988	0.811613	1
034-test0.jpg	34	034-iPhone.jpg	34	252.4133	0	252.4133	0.022976	0.581913	0.708897	1
034-test1.jpg	34	034-5800.jpg	34	42.698	0	42.698	0.016422	0.571393	0.5964	1
035-test0.jpg	35	035-iPhone.jpg	35	454.1452	0	454.1452	0.026003	0.618027	0.785376	1
035-test1.jpg	35	035-iPhone.jpg	35	441.8001	0	441.8001	0.02443	0.587948	0.731249	1
036-test0.jpg	36	036-Droid.jpg	36	103.7253	0	103.7253	0.020758	0.562863	0.60389	1
036-test1.jpg	36	036-iPhone.jpg	36	90.64003	0	90.64003	0.019964	0.567101	0.599108	1
037-test0.jpg	37	037-Canon.jpg	37	261.0928	0	261.0928	0.024995	0.628816	0.834798	1
037-test1.jpg	37	037-iPhone.jpg	37	734.4222	0	734.4222	0.024958	0.637949	0.900555	1
038-test0.jpg	38	038-iPhone.jpg	38	732.7776	0	732.7776	0.024974	0.6649	1.017797	1
038-test1.jpg	38	038-iPhone.jpg	38	355.6225	0	355.6225	0.026031	0.636518	0.83892	1
039-test0.jpg	39	039-Droid.jpg	39	159.4156	0	159.4156	0.025869	0.567907	0.633942	1
039-test1.jpg	39	039-iPhone.jpg	39	277.0095	0	277.0095	0.023845	0.590006	0.705922	1
040-test0.jpg	40	040-Droid.jpg	40	382.9575	0	382.9575	0.023715	0.596471	0.740149	1
040-test1.jpg	40	040-Droid.jpg	40	417.4699	0	417.4699	0.022002	0.604263	0.794663	1

041-test0.jpg	41	041-Droid.jpg	41	832.6718	0	832.6718	0.024742	0.601907	0.806645	1
041-test1.jpg	41	041-Droid.jpg	41	944.507	0	944.507	0.022982	0.608203	0.849985	1
042-test0.jpg	42	042-Canon.jpg	42	584.0026	0	584.0026	0.021408	0.600052	0.822481	1
042-test1.jpg	42	042-Droid.jpg	42	367.1519	0	367.1519	0.022989	0.595179	0.766176	1
043-test0.jpg	43	043-iPhone.jpg	43	395.4825	0	395.4825	0.023	0.582317	0.673339	1
043-test1.jpg	43	043-iPhone.jpg	43	148.2937	0	148.2937	0.022978	0.567817	0.60583	1
044-test0.jpg	44	044-iPhone.jpg	44	513.8183	0	513.8183	0.02766	0.615936	0.834047	1
044-test1.jpg	44	044-iPhone.jpg	44	367.8986	0	367.8986	0.027	0.596513	0.740163	1
045-test0.jpg	45	045-Droid.jpg	45	1132.652	0	1132.652	0.028651	0.629332	1.157041	1
045-test1.jpg	45	045-Droid.jpg	45	472.1491	0	472.1491	0.022005	0.584568	0.692629	1
046-test0.jpg	46	046-iPhone.jpg	46	1062.521	0	1062.521	0.006003	0.605546	0.869586	1
046-test1.jpg	46	046-Canon.jpg	46	922.2082	0	922.2082	0.005991	0.613112	0.999905	1
047-test0.jpg	47	047-iPhone.jpg	47	261.425	0	261.425	0.027623	0.602052	0.709358	1
047-test1.jpg	47	047-iPhone.jpg	47	203.4441	0	203.4441	0.025321	0.585466	0.631479	1
048-test0.jpg	48	048-Canon.jpg	48	26.36597	0	26.36597	0.009511	0.56732	0.589745	1
048-test1.jpg	48	048-Canon.jpg	48	19.9229	0	19.9229	0.019973	0.590513	0.608502	1

049-test0.jpg	49	049-iPhone.jpg	49	1004.902	0	1004.902	0.026591	0.626616	0.95807	1
049-test1.jpg	49	049-iPhone.jpg	49	693.9433	0	693.9433	0.026593	0.60918	0.852923	1
050-test0.jpg	50	050-Canon.jpg	50	252.8481	0	252.8481	0.043347	0.76581	1.213429	1
050-test1.jpg	50	006-Canon.jpg	6	54.08301	1	46.28163	0.037603	0.700851	0.989943	0
051-test0.jpg	51	051-Canon.jpg	51	273.3912	0	273.3912	0.007981	0.5931	0.703387	1
051-test1.jpg	51	051-Canon.jpg	51	316.4998	0	316.4998	0.008325	0.591227	0.72814	1

Tabel B.5: Hasil pengujian menggunakan metode ORB pada keseluruhan *dataset* Book Covers 400.

q name	q class	most simi-lar	most simi-lar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
001-test0.jpg	1	001-iPhone.jpg	1	1135.494	0	1135.494	0.001993	0.938006	1.549221	1
001-test1.jpg	1	001-iPhone.jpg	1	620.6132	0	620.6132	0.00205	0.926299	1.477034	1
002-test0.jpg	2	002-iPhone.jpg	2	257.3484	0	257.3484	0.00302	0.870448	1.417418	1
002-test1.jpg	2	002-iPhone.jpg	2	320.9403	0	320.9403	0.001987	0.86997	1.397741	1
003-test0.jpg	3	003-Droid.jpg	3	570.5712	0	570.5712	0.0037	0.909367	1.58367	1
003-test1.jpg	3	003-iPhone.jpg	3	686.8686	0	686.8686	0.003103	0.875539	1.489177	1
004-test0.jpg	4	004-iPhone.jpg	4	435.4783	0	435.4783	0.001999	0.668233	0.965942	1
004-test1.jpg	4	004-iPhone.jpg	4	1379.885	0	1379.885	0.002699	0.939817	1.590159	1
005-test0.jpg	5	005-Droid.jpg	5	824.2659	0	824.2659	0.002468	0.915383	1.608746	1
005-test1.jpg	5	005-iPhone.jpg	5	397.1632	0	397.1632	0.004344	0.859156	1.499157	1
006-test0.jpg	6	006-iPhone.jpg	6	1364.519	0	1364.519	0.00515	0.936898	1.676573	1
006-test1.jpg	6	006-iPhone.jpg	6	1726.401	0	1726.401	0.002998	0.944518	1.689246	1
007-test0.jpg	7	007-iPhone.jpg	7	301.4036	0	301.4036	0.002256	0.801807	1.267158	1
007-test1.jpg	7	007-iPhone.jpg	7	1010.217	0	1010.217	0.002971	0.740983	1.294955	1

009-test0.jpg	9	009-iPhone.jpg	9	879.8179	0	879.8179	0.002987	0.805846	1.467852	1
009-test1.jpg	9	009-iPhone.jpg	9	939.824	0	939.824	0.003001	0.874467	1.467645	1
010-test0.jpg	10	010-iPhone.jpg	10	896.9966	0	896.9966	0.002989	0.80108	1.469387	1
010-test1.jpg	10	010-iPhone.jpg	10	664.5872	0	664.5872	0.002314	0.792443	1.412473	1
011-test0.jpg	11	011-iPhone.jpg	11	141.9015	0	141.9015	0.002012	0.758003	1.095431	1
011-test1.jpg	11	011-iPhone.jpg	11	444.7705	0	444.7705	0.001998	0.844352	1.356786	1
012-test0.jpg	12	012-iPhone.jpg	12	60.48585	0	60.48585	0.000997	0.743298	1.069689	1
012-test1.jpg	12	012-5800.jpg	12	47.763	0	47.763	0.002001	0.697134	0.932349	1
013-test0.jpg	13	013-iPhone.jpg	13	970.5403	0	970.5403	0.003997	0.883979	1.551395	1
013-test1.jpg	13	013-iPhone.jpg	13	818.9619	0	818.9619	0.003012	0.78292	1.427867	1
014-test0.jpg	14	014-iPhone.jpg	14	586.2259	0	586.2259	0.003	0.724615	1.237714	1
014-test1.jpg	14	014-iPhone.jpg	14	302.3968	0	302.3968	0.002003	0.746835	1.117483	1
015-test0.jpg	15	015-Droid.jpg	15	128.1243	0	128.1243	0.002986	0.753389	1.308187	1
015-test1.jpg	15	019-Canon.jpg	19	82.80885	26	41.79882	0.003997	0.742251	1.32752	0
016-test0.jpg	16	016-iPhone.jpg	16	530.5085	0	530.5085	0.003	0.763129	1.240589	1
016-test1.jpg	16			0			0.001591	0	0	0

017-test0.jpg	17	017-Droid.jpg	17	416.8133	0	416.8133	0.00204	0.727055	1.19797	1
017-test1.jpg	17	017-Droid.jpg	17	150.257	0	150.257	0.001976	0.609849	0.874092	1
018-test0.jpg	18	018-iPhone.jpg	18	162.0116	0	162.0116	0.003296	0.882666	1.423491	1
018-test1.jpg	18	018-iPhone.jpg	18	321.9313	0	321.9313	0.001905	0.971759	1.567811	1
019-test0.jpg	19	019-iPhone.jpg	19	219.2575	0	219.2575	0.003232	0.902894	1.526055	1
019-test1.jpg	19	019-iPhone.jpg	19	242.4292	0	242.4292	0.003	0.813545	1.399522	1
020-test0.jpg	20	020-iPhone.jpg	20	1540.565	0	1540.565	0.001987	0.761446	1.625134	1
020-test1.jpg	20	020-iPhone.jpg	20	297.3525	0	297.3525	0	0.847243	1.384662	1
021-test0.jpg	21	004-iPhone.jpg	4	57.28095	3	33.03471	0.001002	0.667302	0.858254	0
021-test1.jpg	21	004-iPhone.jpg	4	89.1852	30	22.35583	0.002022	0.731055	1.092077	0
022-test0.jpg	22	022-Droid.jpg	22	1070.341	0	1070.341	0.003001	0.918327	1.519535	1
022-test1.jpg	22	022-Droid.jpg	22	370.1457	0	370.1457	0.001995	0.800678	1.233384	1
023-test0.jpg	23	023-iPhone.jpg	23	722.5788	0	722.5788	0.004307	1.006799	1.637591	1
023-test1.jpg	23	023-iPhone.jpg	23	371.1066	0	371.1066	0.002288	0.849184	1.358228	1
024-test0.jpg	24	004-iPhone.jpg	4	85.55175	45	30.88542	0.002997	0.888786	1.474206	0
024-test1.jpg	24	024-iPhone.jpg	24	263.5127	0	263.5127	0.002979	0.783083	1.396073	1

025-test0.jpg	25	025-iPhone.jpg	25	442.7088	0	442.7088	0.002	0.806468	1.368504	1
025-test1.jpg	25	025-iPhone.jpg	25	655.1504	0	655.1504	0.001995	0.893732	1.452426	1
026-test0.jpg	26	026-iPhone.jpg	26	896.6603	0	896.6603	0.001618	0.766273	1.411345	1
026-test1.jpg	26	026-iPhone.jpg	26	482.422	0	482.422	0.003011	0.822938	1.411027	1
027-test0.jpg	27	027-iPhone.jpg	27	1288.529	0	1288.529	0.001989	0.790998	1.408509	1
027-test1.jpg	27	027-iPhone.jpg	27	905.4552	0	905.4552	0.002253	0.903529	1.469681	1
028-test0.jpg	28	028-Droid.jpg	28	863.8348	0	863.8348	0.00194	0.832343	1.48528	1
028-test1.jpg	28	028-iPhone.jpg	28	1658.603	0	1658.603	0.001997	0.773254	1.481694	1
029-test0.jpg	29	029-Canon.jpg	29	620.9206	0	620.9206	0.002008	0.930833	1.564932	1
029-test1.jpg	29	029-Canon.jpg	29	112.2789	0	112.2789	0.002012	0.788359	1.345493	1
030-test0.jpg	30	030-Droid.jpg	30	1619.585	0	1619.585	0.002019	0.896327	1.629256	1
030-test1.jpg	30	030-Droid.jpg	30	2023.491	0	2023.491	0.001982	0.8778	1.757521	1
031-test0.jpg	31	031-iPhone.jpg	31	300.1581	0	300.1581	0.001991	0.746953	1.202801	1
031-test1.jpg	31	031-iPhone.jpg	31	379.8874	0	379.8874	0.00199	0.81145	1.273494	1
032-test0.jpg	32	032-iPhone.jpg	32	99.64463	0	99.64463	0.001987	0.88139	1.494259	1
032-test1.jpg	32	032-iPhone.jpg	32	240.5112	0	240.5112	0.002999	0.925131	1.506208	1

033-test0.jpg	33	033-iPhone.jpg	33	345.7131	0	345.7131	0.007264	0.890712	1.484094	1
033-test1.jpg	33	033-iPhone.jpg	33	534.7501	0	534.7501	0.002977	0.807033	1.43827	1
034-test0.jpg	34	034-iPhone.jpg	34	105.8592	0	105.8592	0.001998	0.836732	1.296538	1
034-test1.jpg	34	049-Droid.jpg	49	26.80765			0.002011	0.588138	0.667871	0
035-test0.jpg	35	035-iPhone.jpg	35	777.403	0	777.403	0.001991	0.834684	1.485601	1
035-test1.jpg	35	035-iPhone.jpg	35	603.8242	0	603.8242	0.002996	0.889595	1.476726	1
036-test0.jpg	36	036-iPhone.jpg	36	306.9033	0	306.9033	0	0.733853	1.024453	1
036-test1.jpg	36	036-iPhone.jpg	36	352.1592	0	352.1592	0.001001	0.615787	0.841595	1
037-test0.jpg	37	016-iPhone.jpg	16	98.44903	50	35.22287	0.002997	0.852374	1.45866	0
037-test1.jpg	37	022-5800.jpg	22	105.5419	1	100.59	0.001578	0.748267	1.316303	0
038-test0.jpg	38	038-iPhone.jpg	38	711.3614	0	711.3614	0.007216	0.838756	1.48613	1
038-test1.jpg	38	038-iPhone.jpg	38	176.8558	0	176.8558	0.00301	0.838654	1.414118	1
039-test0.jpg	39	039-iPhone.jpg	39	207.8798	0	207.8798	0	0.712938	1.173355	1
039-test1.jpg	39	039-iPhone.jpg	39	628.968	0	628.968	0.002988	0.803526	1.378599	1
040-test0.jpg	40	011-Canon.jpg	11	96.52838	46	30.57061	0.002104	0.757834	1.319873	0
040-test1.jpg	40	040-iPhone.jpg	40	400.0018	0	400.0018	0.002011	0.831966	1.378823	1

041-test0.jpg	41	041-iPhone.jpg	41	1697.886	0	1697.886	0.002	0.802164	1.522826	1
041-test1.jpg	41	041-iPhone.jpg	41	1805.208	0	1805.208	0.001999	0.805924	1.532494	1
042-test0.jpg	42	042-iPhone.jpg	42	1364.031	0	1364.031	0.003003	0.828084	1.492691	1
042-test1.jpg	42	042-iPhone.jpg	42	906.5798	0	906.5798	0.002013	0.79907	1.418433	1
043-test0.jpg	43	028-Canon.jpg	28	67.80983	2	55.59579	0.002012	0.767422	1.185895	0
043-test1.jpg	43	007-5800.jpg	7	28.83514	26	0	0.001998	0.598591	0.723924	0
044-test0.jpg	44	044-iPhone.jpg	44	1573.858	0	1573.858	0.003014	0.830949	1.691473	1
044-test1.jpg	44	044-Canon.jpg	44	1096.059	0	1096.059	0.002012	0.851653	1.538981	1
045-test0.jpg	45	045-iPhone.jpg	45	2294.785	0	2294.785	0.003011	0.732904	1.84683	1
045-test1.jpg	45	045-iPhone.jpg	45	436.676	0	436.676	0.001991	0.900452	1.47931	1
046-test0.jpg	46	035-Canon.jpg	35	55.26045	84	0	0.001109	0.721423	1.099063	0
046-test1.jpg	46	046-iPhone.jpg	46	90.38015	0	90.38015	0.001973	0.724192	1.135882	1
047-test0.jpg	47	047-iPhone.jpg	47	545.1434	0	545.1434	0.001988	0.888169	1.449388	1
047-test1.jpg	47	047-iPhone.jpg	47	175.4989	0	175.4989	0.001011	0.824237	1.321791	1
048-test0.jpg	48	048-5800.jpg	48	474.7422	0	474.7422	0.001	0.819301	1.26781	1
048-test1.jpg	48	048-5800.jpg	48	296.5828	0	296.5828	0.001	0.69077	0.963898	1

049-test0.jpg	49	049-iPhone.jpg	49	1034.678	0	1034.678	0.002012	0.850751	1.599734	1
049-test1.jpg	49	049-iPhone.jpg	49	768.4667	0	768.4667	0.003	0.879489	1.554672	1
050-test0.jpg	50	004-iPhone.jpg	4	72.66529	16	45.54792	0.004012	0.814311	1.409534	0
050-test1.jpg	50	002-iPhone.jpg	2	73.51343	20	46.11453	0.002981	0.821396	1.40793	0
051-test0.jpg	51	051-iPhone.jpg	51	480.803	0	480.803	0.002002	0.784526	1.259905	1
051-test1.jpg	51	051-Canon.jpg	51	460.1334	0	460.1334	0.001971	0.781836	1.272069	1

Tabel B.6: Hasil pengujian menggunakan metode ORB pada keseluruhan *dataset* Book Covers 600.

q name	q class	most simi-lar	most simi-lar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
001-test0.jpg	1	001-iPhone.jpg	1	1021.409	0	1021.409	0.002988	0.947627	1.617839	1
001-test1.jpg	1	001-iPhone.jpg	1	554.5899	0	554.5899	0.003825	0.899658	1.523541	1
002-test0.jpg	2	002-iPhone.jpg	2	530.2862	0	530.2862	0.003108	0.893548	1.488138	1
002-test1.jpg	2	002-iPhone.jpg	2	557.1502	0	557.1502	0.003174	0.876019	1.468091	1
003-test0.jpg	3	003-iPhone.jpg	3	926.9529	0	926.9529	0.00415	0.93227	1.563217	1
003-test1.jpg	3	003-iPhone.jpg	3	1037.82	0	1037.82	0.004909	0.907956	1.571083	1
004-test0.jpg	4	004-iPhone.jpg	4	348.1715	0	348.1715	0.002	0.754647	1.106301	1
004-test1.jpg	4	004-iPhone.jpg	4	1225.425	0	1225.425	0.00308	0.912192	1.585728	1
005-test0.jpg	5	005-Droid.jpg	5	1879.901	0	1879.901	0.00456	0.892998	1.627837	1
005-test1.jpg	5	005-Droid.jpg	5	1003.108	0	1003.108	0.002177	0.909882	1.55915	1
006-test0.jpg	6	006-iPhone.jpg	6	1510.414	0	1510.414	0.005112	0.928026	1.722374	1
006-test1.jpg	6	006-iPhone.jpg	6	1935.466	0	1935.466	0.003758	0.926928	1.812523	1
007-test0.jpg	7	007-iPhone.jpg	7	540.3802	0	540.3802	0.00297	0.835968	1.436054	1
007-test1.jpg	7	007-iPhone.jpg	7	1089.931	0	1089.931	0.002975	0.817918	1.448335	1

009-test0.jpg	9	009-iPhone.jpg	9	1045.491	0	1045.491	0	0.845621	1.56031	1
009-test1.jpg	9	009-iPhone.jpg	9	1202.326	0	1202.326	0.002974	0.860657	1.515516	1
010-test0.jpg	10	010-Canon.jpg	10	1424.247	0	1424.247	0.003	0.84066	1.589484	1
010-test1.jpg	10	010-Canon.jpg	10	1385.46	0	1385.46	0.004	0.864319	1.598789	1
011-test0.jpg	11	011-Droid.jpg	11	301.6231	0	301.6231	0.003974	0.813772	1.31605	1
011-test1.jpg	11	011-iPhone.jpg	11	872.7564	0	872.7564	0.002987	0.894348	1.543006	1
012-test0.jpg	12	012-Canon.jpg	12	71.05124	0	71.05124	0.001976	0.721986	1.047304	1
012-test1.jpg	12	018-Droid.jpg	18	48.34664	1	46.97396	0.002	0.681816	0.933388	0
013-test0.jpg	13	013-iPhone.jpg	13	1133.806	0	1133.806	0.003999	0.85868	1.57323	1
013-test1.jpg	13	013-iPhone.jpg	13	760.8469	0	760.8469	0.003998	0.852671	1.601844	1
014-test0.jpg	14	014-iPhone.jpg	14	1163.314	0	1163.314	0.002999	0.846838	1.544633	1
014-test1.jpg	14	014-iPhone.jpg	14	579.7319	0	579.7319	0.001987	0.773392	1.24676	1
015-test0.jpg	15	015-iPhone.jpg	15	450.5319	0	450.5319	0.008678	0.726359	1.326416	1
015-test1.jpg	15	015-iPhone.jpg	15	206.9443	0	206.9443	0.001193	0.726603	1.309945	1
016-test0.jpg	16	016-iPhone.jpg	16	496.2113	0	496.2113	0.002987	0.769823	1.320478	1
016-test1.jpg	16			0			0.006541	0.538338	0.539351	0

017-test0.jpg	17	017-Droid.jpg	17	680.3718	0	680.3718	0.003695	0.823591	1.396466	1
017-test1.jpg	17	017-iPhone.jpg	17	424.8694	0	424.8694	0.001999	0.73589	1.083966	1
018-test0.jpg	18	018-iPhone.jpg	18	331.5772	0	331.5772	0.002998	0.90769	1.518487	1
018-test1.jpg	18	018-iPhone.jpg	18	504.3123	0	504.3123	0	0.931675	1.588359	1
019-test0.jpg	19	019-iPhone.jpg	19	470.6346	0	470.6346	0.003999	0.868399	1.492277	1
019-test1.jpg	19	019-iPhone.jpg	19	230.4184	0	230.4184	0.004001	0.887246	1.485472	1
020-test0.jpg	20	020-iPhone.jpg	20	2220.376	0	2220.376	0.003013	0.857304	2.038892	1
020-test1.jpg	20	020-iPhone.jpg	20	497.1765	0	497.1765	0.002987	0.869569	1.484565	1
021-test0.jpg	21	021-Droid.jpg	21	60.35216	0	60.35216	0.002987	0.654359	0.915265	1
021-test1.jpg	21	001-iPhone.jpg	1	71.01002	39	22.48166	0.002001	0.724524	1.158373	0
022-test0.jpg	22	022-Droid.jpg	22	768.8462	0	768.8462	0.003	0.859174	1.49321	1
022-test1.jpg	22	022-Droid.jpg	22	153.8379	0	153.8379	0.002588	0.902837	1.427156	1
023-test0.jpg	23	023-iPhone.jpg	23	930.2219	0	930.2219	0.002013	0.878712	1.494001	1
023-test1.jpg	23	023-iPhone.jpg	23	605.7348	0	605.7348	0.002016	0.899576	1.479867	1
024-test0.jpg	24	024-Droid.jpg	24	92.84222	0	92.84222	0.004013	0.876643	1.467411	1
024-test1.jpg	24	024-iPhone.jpg	24	180.6839	0	180.6839	0.002989	0.940818	1.540215	1

025-test0.jpg	25	025-iPhone.jpg	25	756.1208	0	756.1208	0.003	0.937792	1.553174	1
025-test1.jpg	25	025-iPhone.jpg	25	877.067	0	877.067	0	0.900475	1.539202	1
026-test0.jpg	26	026-iPhone.jpg	26	738.1857	0	738.1857	0.003	0.843288	1.497323	1
026-test1.jpg	26	026-iPhone.jpg	26	489.7546	0	489.7546	0.002988	0.848304	1.41097	1
027-test0.jpg	27	027-iPhone.jpg	27	685.3003	0	685.3003	0.002987	0.896113	1.501998	1
027-test1.jpg	27	027-iPhone.jpg	27	304.1874	0	304.1874	0.003102	0.897653	1.460305	1
028-test0.jpg	28	028-iPhone.jpg	28	1523.554	0	1523.554	0.002999	0.840236	1.65226	1
028-test1.jpg	28	028-iPhone.jpg	28	1953.314	0	1953.314	0.007261	0.873121	1.682481	1
029-test0.jpg	29	029-Canon.jpg	29	805.7343	0	805.7343	0.003987	0.866242	1.514124	1
029-test1.jpg	29	029-Canon.jpg	29	324.4385	0	324.4385	0.002908	0.848796	1.462907	1
030-test0.jpg	30	030-Droid.jpg	30	2094.859	0	2094.859	0.002992	0.8609	1.75399	1
030-test1.jpg	30	030-Droid.jpg	30	2918.161	0	2918.161	0.003009	0.854944	1.926728	1
031-test0.jpg	31	031-iPhone.jpg	31	449.6052	0	449.6052	0.001976	0.829006	1.418635	1
031-test1.jpg	31	031-iPhone.jpg	31	424.529	0	424.529	0.003	0.813432	1.394123	1
032-test0.jpg	32	032-iPhone.jpg	32	154.6026	0	154.6026	0.003013	0.967102	1.581327	1
032-test1.jpg	32	032-iPhone.jpg	32	226.8784	0	226.8784	0.002352	0.874694	1.480962	1

033-test0.jpg	33	033-iPhone.jpg	33	178.0604	0	178.0604	0.003568	0.86296	1.470306	1
033-test1.jpg	33	033-iPhone.jpg	33	366.437	0	366.437	0.003003	0.855719	1.477266	1
034-test0.jpg	34	034-iPhone.jpg	34	163.6094	0	163.6094	0.001545	0.826824	1.360194	1
034-test1.jpg	34	020-Canon.jpg	20	57.33279			0.002022	0.564557	0.690928	0
035-test0.jpg	35	035-Canon.jpg	35	718.5032	0	718.5032	0.002987	0.798214	1.400592	1
035-test1.jpg	35	035-Canon.jpg	35	736.7636	0	736.7636	0.002987	0.867955	1.511675	1
036-test0.jpg	36	036-iPhone.jpg	36	478.3148	0	478.3148	0.001999	0.763795	1.229135	1
036-test1.jpg	36	036-iPhone.jpg	36	431.1966	0	431.1966	0.002024	0.677897	0.989995	1
037-test0.jpg	37	030-iPhone.jpg	30	122.0664	3	78.28572	0.002992	0.819531	1.403636	0
037-test1.jpg	37	022-iPhone.jpg	22	147.306	1	93.85325	0.009382	0.795858	1.400423	0
038-test0.jpg	38	038-Droid.jpg	38	852.5563	0	852.5563	0.003029	0.81019	1.542343	1
038-test1.jpg	38	038-Droid.jpg	38	403.6865	0	403.6865	0.002999	0.81976	1.438048	1
039-test0.jpg	39	039-iPhone.jpg	39	469.8602	0	469.8602	0.002145	0.79824	1.362736	1
039-test1.jpg	39	039-iPhone.jpg	39	995.8834	0	995.8834	0.002979	0.788509	1.50894	1
040-test0.jpg	40	040-Droid.jpg	40	314.0586	0	314.0586	0.003	0.82783	1.393617	1
040-test1.jpg	40	040-Droid.jpg	40	1024.816	0	1024.816	0.003008	0.796302	1.407589	1

041-test0.jpg	41	041-iPhone.jpg	41	2145.156	0	2145.156	0.003	0.805438	1.707262	1
041-test1.jpg	41	041-iPhone.jpg	41	2222.854	0	2222.854	0.003989	0.787574	1.755825	1
042-test0.jpg	42	042-iPhone.jpg	42	1342.332	0	1342.332	0.002989	0.824256	1.563006	1
042-test1.jpg	42	042-iPhone.jpg	42	870.4408	0	870.4408	0.003024	0.831753	1.514236	1
043-test0.jpg	43	023-iPhone.jpg	23	91.35481	4	66.32069	0.001976	0.769939	1.246119	0
043-test1.jpg	43	017-iPhone.jpg	17	31.60104	31	0	0	0.559932	0.693274	0
044-test0.jpg	44	044-Canon.jpg	44	1760.381	0	1760.381	0.004	0.89234	1.996668	1
044-test1.jpg	44	044-Canon.jpg	44	1290.363	0	1290.363	0.003011	0.83854	1.63423	1
045-test0.jpg	45	045-Droid.jpg	45	1562.018	0	1562.018	0.002976	0.813274	1.878415	1
045-test1.jpg	45	045-iPhone.jpg	45	592.6875	0	592.6875	0.003009	0.884382	1.493355	1
046-test0.jpg	46	023-Droid.jpg	23	64.74094	79	9.577897	0.002024	0.690399	1.106293	0
046-test1.jpg	46	010-Droid.jpg	10	59.18194	5	44.97207	0.001	0.696059	1.135798	0
047-test0.jpg	47	047-iPhone.jpg	47	605.1373	0	605.1373	0.003002	0.857918	1.487223	1
047-test1.jpg	47	047-iPhone.jpg	47	332.345	0	332.345	0.002016	0.804342	1.277528	1
048-test0.jpg	48	048-iPhone.jpg	48	591.1165	0	591.1165	0.003	0.836656	1.364422	1
048-test1.jpg	48	048-iPhone.jpg	48	499.1092	0	499.1092	0.001987	0.71949	1.042289	1

049-test0.jpg	49	049-iPhone.jpg	49	1908.616	0	1908.616	0.004	0.86238	1.688444	1
049-test1.jpg	49	049-iPhone.jpg	49	1365.77	0	1365.77	0.001923	0.909344	1.635938	1
050-test0.jpg	50	051-Canon.jpg	51	100.0674	5	68.23035	0.006047	0.818877	1.434896	0
050-test1.jpg	50	016-Droid.jpg	16	104.1857	11	55.03606	0.004994	0.821686	1.418965	0
051-test0.jpg	51	051-Canon.jpg	51	290.1125	0	290.1125	0.002025	0.768367	1.216331	1
051-test1.jpg	51	051-Canon.jpg	51	344.5323	0	344.5323	0.001991	0.80208	1.300799	1

Tabel B.7: Hasil pengujian menggunakan metode SIFT pada keseluruhan dataset GSV 400.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	alfamart 10.jpg	alfamart	67.37269	0	67.37269	0.034268	1.649762	1.852541	1
test al-famart 2.jpg	alfamart	alfamart 7.jpg	alfamart	82.908	0	82.908	0.020284	1.589816	1.771248	1
test al-famart 3.jpg	alfamart	alfamart 5.jpg	alfamart	80.59929	0	80.59929	0.021675	1.555369	1.67468	1
test al-famart 4.jpg	alfamart	alfamart 9.jpg	alfamart	38.36149	0	38.36149	0.022183	1.586816	1.730361	1
test al-famart 5.jpg	alfamart	starbucks 5.jpg	starbucks	40.11285	2	33.03739	0.032174	1.67056	1.885181	0
test binus 1.jpg	binus	binus 1.jpg	binus	1266.102	0	1266.102	0.027469	1.54416	1.780372	1
test binus 2.jpg	binus	binus 9.jpg	binus	449.9437	0	449.9437	0.026992	1.523604	1.681201	1
test binus 3.jpg	binus	binus 5.jpg	binus	354.1618	0	354.1618	0.0266	1.524337	1.703564	1
test binus 4.jpg	binus	binus 2.jpg	binus	583.1021	0	583.1021	0.023586	1.504713	1.606394	1
test binus 5.jpg	binus	binus 1.jpg	binus	194.3614	0	194.3614	0.027551	1.542766	1.685981	1
test gembul 1.jpg	gembul	gembul 1.jpg	gembul	1080.499	0	1080.499	0.027752	1.541152	1.726412	1
test gembul 2.jpg	gembul	gembul 1.jpg	gembul	312.9171	0	312.9171	0.026403	1.545139	1.665886	1

test gembul 3.jpg	gembul	gembul 1.jpg	gembul	272.6699	0	272.6699	0.028779	1.595237	1.846609	1
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	1233.326	0	1233.326	0.020994	1.576777	1.924598	1
test gembul 5.jpg	gembul	gembul 1.jpg	gembul	406.7817	0	406.7817	0.024002	1.527668	1.652732	1
test harris 1.jpg	harris	harris 2.jpg	harris	258.8197	0	258.8197	0.017988	1.550992	1.695888	1
test harris 2.jpg	harris	harris 7.jpg	harris	211.493	0	211.493	0.024757	1.598701	1.784069	1
test harris 3.jpg	harris	harris 7.jpg	harris	265.3326	0	265.3326	0.026795	1.583568	1.767219	1
test harris 4.jpg	harris	harris 7.jpg	harris	299.51	0	299.51	0.026229	1.569875	1.745261	1
test harris 5.jpg	harris	harris 2.jpg	harris	309.9828	0	309.9828	0.025983	1.575749	1.778547	1
test oxy 1.jpg	oxy	oxy 10.jpg	oxy	408.3313	0	408.3313	0.028875	1.62693	1.905654	1
test oxy 2.jpg	oxy	oxy 5.jpg	oxy	1316.373	0	1316.373	0.027994	1.554917	1.769516	1
test oxy 3.jpg	oxy	oxy 3.jpg	oxy	1153.379	0	1153.379	0.02524	1.517286	1.710245	1
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	382.0388	0	382.0388	0.026989	1.539311	1.701292	1
test oxy 5.jpg	oxy	oxy 10.jpg	oxy	1705.691	0	1705.691	0.031971	1.632222	2.067142	1
test ping 1.jpg	ping	ping 1.jpg	ping	163.4006	0	163.4006	0.026976	1.561988	1.738852	1
test ping 2.jpg	ping	ping 1.jpg	ping	254.4602	0	254.4602	0.027	1.536962	1.71657	1
test ping 3.jpg	ping	ping 6.jpg	ping	2521.88	0	2521.88	0.026991	1.603671	3.393789	1

test ping 4.jpg	ping	ping	ping 2.jpg	ping	256.5624	0	256.5624	0.028804	1.576417	1.77048	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	2214.143	0	2214.143	0.029049	1.587431	2.265225	1
test porcafe 1.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	1396.895	0	1396.895	0.025385	1.610965	2.146364	1
test porcafe 2.jpg	porcafe	porcafe	porcafe 7.jpg	porcafe	1615.791	0	1615.791	0.024977	1.577893	2.022534	1
test porcafe 3.jpg	porcafe	porcafe	porcafe 9.jpg	porcafe	1654.261	0	1654.261	0.016175	1.548292	1.812493	1
test porcafe 4.jpg	porcafe	porcafe	porcafe 5.jpg	porcafe	488.2118	0	488.2118	0.026405	1.556229	1.701984	1
test porcafe 5.jpg	porcafe	harris	7.jpg	harris	34.95018	2	25.75951	0.020934	1.570218	1.73089	0
test starbucks 1.jpg	starbucks	starbucks	starbucks 2.jpg	starbucks	106.9772	0	106.9772	0.026977	1.57192	1.749758	1
test starbucks 2.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	1077.506	0	1077.506	0.025149	1.60178	2.033559	1
test starbucks 3.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	554.8613	0	554.8613	0.030016	1.650135	2.151448	1
test starbucks 4.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	864.2347	0	864.2347	0.027488	1.565202	1.98131	1
test starbucks 5.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	962.6447	0	962.6447	0.024942	1.614625	2.105474	1
test toyota 1.jpg	toyota	toyota	toyota 10.jpg	toyota	217.3184	0	217.3184	0.023584	1.595982	1.814487	1
test toyota 2.jpg	toyota	toyota	porcafe 4.jpg	porcafe	40.20416	1	29.72628	0.027982	1.650695	1.913203	0
test toyota 3.jpg	toyota	toyota	toyota 9.jpg	toyota	44.07315	0	44.07315	0.025016	1.61772	1.836881	1
test toyota 4.jpg	toyota	toyota	toyota 9.jpg	toyota	226.9311	0	226.9311	0.032018	1.637922	1.921248	1

test toyota 5.jpg	toyota	toyota 1.jpg	toyota	221.6307	0	221.6307	0.024554	1.574683	1.757211	1
test yogya 1.jpg	yogya	yogya 6.jpg	yogya	542.2241	0	542.2241	0.02403	1.534065	1.734461	1
test yogya 2.jpg	yogya	yogya 5.jpg	yogya	309.8666	0	309.8666	0.024683	1.536174	1.655829	1
test yogya 3.jpg	yogya	yogya 10.jpg	yogya	473.546	0	473.546	0.020968	1.559084	1.777141	1
test yogya 4.jpg	yogya	yogya 2.jpg	yogya	469.424	0	469.424	0.021264	1.565062	1.79292	1
test yogya 5.jpg	yogya	yogya 2.jpg	yogya	1052.773	0	1052.773	0.028185	1.598817	1.91476	1

Tabel B.8: Hasil pengujian menggunakan metode SIFT pada *dataset* GSV 400 yang telah tersaring dengan Threshold 1

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	starbucks 3.jpg	starbucks	53.95838	8	24.07874	0.026009	0.300073	0.539224	0
test al-famart 2.jpg	alfamart	alfamart 9.jpg	alfamart	33.34672	0	33.34672	0.02256	0.288693	0.477204	1
test al-famart 3.jpg	alfamart	alfamart 5.jpg	alfamart	54.25892	0	54.25892	0.018619	0.239165	0.378287	1
test al-famart 4.jpg	alfamart	starbucks 3.jpg	starbucks	41.46633	26	0	0.019817	0.271016	0.442601	0
test al-famart 5.jpg	alfamart	starbucks 3.jpg	starbucks	61.85729	19	11.30521	0.027582	0.302586	0.557746	0
test binus 1.jpg	binus	binus 9.jpg	binus	173.2239	0	173.2239	0.023312	0.227573	0.391015	1
test binus 2.jpg	binus	binus 9.jpg	binus	182.1656	0	182.1656	0.02203	0.223663	0.374551	1
test binus 3.jpg	binus	binus 5.jpg	binus	236.4116	0	236.4116	0.024842	0.221189	0.373864	1
test binus 4.jpg	binus	binus 3.jpg	binus	79.63454	0	79.63454	0.022998	0.210279	0.292067	1
test binus 5.jpg	binus	binus 1.jpg	binus	56.31012	0	56.31012	0.025281	0.245637	0.397536	1
test gembul 1.jpg	gembul	gembul 1.jpg	gembul	185.5076	0	185.5076	0.025999	0.243411	0.362565	1
test gembul 2.jpg	gembul	gembul 1.jpg	gembul	94.78813	0	94.78813	0.024287	0.237072	0.354912	1

test gembul 3.jpg	gembul	gembul 10.jpg	gembul	102.3463	0	102.3463	0.026898	0.285545	0.515877	1
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	363.1202	0	363.1202	0.020714	0.275776	0.497695	1
test gembul 5.jpg	gembul	gembul 1.jpg	gembul	63.17592	0	63.17592	0.024832	0.244094	0.35308	1
test harris 1.jpg	harris	starbucks 3.jpg	starbucks	46.40022	3	19.33645	0.018096	0.25083	0.391587	0
test harris 2.jpg	harris	starbucks 6.jpg	starbucks	50.0247	3	24.17192	0.024688	0.274355	0.475703	0
test harris 3.jpg	harris	ping 10.jpg	ping	47.82192	2	41.19869	0.0222	0.300391	0.521867	0
test harris 4.jpg	harris	harris 7.jpg	harris	57.81005	0	57.81005	0.027568	0.258089	0.438499	1
test harris 5.jpg	harris	harris 2.jpg	harris	54.45104	0	54.45104	0.025535	0.282117	0.48138	1
test oxy 1.jpg	oxy	oxy 6.jpg	oxy	76.35348	0	76.35348	0.028642	0.305213	0.590876	1
test oxy 2.jpg	oxy	oxy 5.jpg	oxy	283.2824	0	283.2824	0.025581	0.238482	0.364765	1
test oxy 3.jpg	oxy	oxy 3.jpg	oxy	327.6274	0	327.6274	0.025414	0.22484	0.345767	1
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	149.4616	0	149.4616	0.026452	0.248455	0.42166	1
test oxy 5.jpg	oxy	oxy 10.jpg	oxy	239.6162	0	239.6162	0.031619	0.316681	0.638361	1
test ping 1.jpg	ping	ping 4.jpg	ping	70.07965	0	70.07965	0.024787	0.266515	0.447726	1
test ping 2.jpg	ping	ping 1.jpg	ping	127.7816	0	127.7816	0.028022	0.260492	0.402286	1
test ping 3.jpg	ping	ping 6.jpg	ping	556.2249	0	556.2249	0.026052	0.311231	0.673754	1

test ping 4.jpg	ping	ping	ping 2.jpg	ping	80.47826	0	80.47826	0.025258	0.258067	0.449754	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	611.1583	0	611.1583	0.02802	0.307019	0.58913	1
test porcafe 1.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	525.5699	0	525.5699	0.024262	0.29185	0.544229	1
test porcafe 2.jpg	porcafe	porcafe	porcafe 7.jpg	porcafe	450.0601	0	450.0601	0.024993	0.279535	0.488207	1
test porcafe 3.jpg	porcafe	porcafe	porcafe 9.jpg	porcafe	306.8478	0	306.8478	0.017469	0.26263	0.413768	1
test porcafe 4.jpg	porcafe	porcafe	porcafe 5.jpg	porcafe	56.05049	0	56.05049	0.025969	0.257893	0.425192	1
test porcafe 5.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	56.83876	0	56.83876	0.021168	0.280298	0.483777	1
test starbucks 1.jpg	starbucks	starbucks	starbucks 6.jpg	starbucks	72.55669	0	72.55669	0.025018	0.280888	0.478437	1
test starbucks 2.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	659.2316	0	659.2316	0.024969	0.303241	0.655414	1
test starbucks 3.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	375.0962	0	375.0962	0.029998	0.334227	0.775646	1
test starbucks 4.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	554.033	0	554.033	0.022852	0.255075	0.559296	1
test starbucks 5.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	728.7059	0	728.7059	0.026034	0.30427	0.700796	1
test toyota 1.jpg	toyota	toyota	toyota 5.jpg	toyota	84.67934	0	84.67934	0.023005	0.29964	0.551009	1
test toyota 2.jpg	toyota	toyota	starbucks 7.jpg	starbucks	54.06151	3	31.57439	0.029696	0.337128	0.613129	0
test toyota 3.jpg	toyota	toyota	starbucks 3.jpg	starbucks	72.98084	5	25.80107	0.026551	0.29046	0.532985	0
test toyota 4.jpg	toyota	toyota	toyota 9.jpg	toyota	100.0272	0	100.0272	0.028151	0.3167	0.643098	1

test toyota	toyota	toyota 1.jpg	toyota	55.51829	0	55.51829	0.021045	0.277493	0.48399	1
5.jpg										
test yogya	yogya	yogya 6.jpg	yogya	149.9976	0	149.9976	0.022977	0.23787	0.387659	1
1.jpg										
test yogya	yogya	yogya 4.jpg	yogya	74.53375	0	74.53375	0.024179	0.234309	0.381116	1
2.jpg										
test yogya	yogya	yogya 10.jpg	yogya	64.76149	0	64.76149	0.023073	0.258888	0.436357	1
3.jpg										
test yogya	yogya	yogya 1.jpg	yogya	115.1646	0	115.1646	0.02247	0.259901	0.479221	1
4.jpg										
test yogya	yogya	yogya 2.jpg	yogya	198.0291	0	198.0291	0.028579	0.294598	0.517269	1
5.jpg										

Tabel B.9: Hasil pengujian menggunakan metode SIFT pada *dataset* GSV 400 yang telah tersaring dengan Threshold 2

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	ping 4.jpg	ping	113.9927			0.026994	0.062935	0.326197	0
test al-famart 2.jpg	alfamart	ping 4.jpg	ping	88.66449			0.020995	0.057011	0.287862	0
test al-famart 3.jpg	alfamart	ping 4.jpg	ping	90.25828			0.017989	0.043977	0.210033	0
test al-famart 4.jpg	alfamart	ping 4.jpg	ping	110.3566			0.019774	0.05915	0.265621	0
test al-famart 5.jpg	alfamart	ping 4.jpg	ping	120.4487			0.027955	0.078191	0.406936	0
test binus 1.jpg	binus	ping 4.jpg	ping	81.30008			0.025009	0.031031	0.156021	0
test binus 2.jpg	binus	ping 4.jpg	ping	53.92333			0.023988	0.030012	0.125625	0
test binus 3.jpg	binus	ping 1.jpg	ping	46.51112			0.023992	0.027006	0.122157	0
test binus 4.jpg	binus	starbucks 6.jpg	starbucks	40.47819			0.023999	0.018006	0.090246	0
test binus 5.jpg	binus	ping 4.jpg	ping	84.9625			0.026005	0.037001	0.174205	0
test gembul 1.jpg	gembul	ping 4.jpg	ping	92.30347			0.025983	0.042009	0.192997	0
test gembul 2.jpg	gembul	ping 4.jpg	ping	75.32821			0.024508	0.042002	0.195494	0

test gembul 3.jpg	gembul	ping 4.jpg	ping	107.7352			0.025993	0.064018	0.336507	0
test gembul 4.jpg	gembul	ping 4.jpg	ping	91.41037			0.021009	0.065019	0.273644	0
test gembul 5.jpg	gembul	ping 4.jpg	ping	80.16116			0.025995	0.035996	0.183006	0
test harris 1.jpg	harris	ping 4.jpg	ping	92.35919			0.016993	0.046012	0.212475	0
test harris 2.jpg	harris	ping 4.jpg	ping	99.20123			0.024	0.07559	0.294175	0
test harris 3.jpg	harris	ping 4.jpg	ping	100.0878			0.024976	0.063215	0.300846	0
test harris 4.jpg	harris	ping 4.jpg	ping	99.09031			0.02599	0.05101	0.24342	0
test harris 5.jpg	harris	ping 4.jpg	ping	78.03238			0.02601	0.067899	0.280118	0
test oxy 1.jpg	oxy	ping 4.jpg	ping	123.3804			0.028989	0.093271	0.433243	0
test oxy 2.jpg	oxy	ping 4.jpg	ping	63.20928			0.024992	0.039233	0.167855	0
test oxy 3.jpg	oxy	ping 4.jpg	ping	86.45664			0.023557	0.031476	0.144135	0
test oxy 4.jpg	oxy	ping 4.jpg	ping	86.12379			0.025234	0.040033	0.202002	0
test oxy 5.jpg	oxy	ping 4.jpg	ping	118.4525			0.029999	0.113029	0.489012	0
test ping 1.jpg	ping	ping 4.jpg	ping	85.88064	0	85.88064	0.027699	0.060997	0.253832	1
test ping 2.jpg	ping	ping 4.jpg	ping	80.27622	0	80.27622	0.024999	0.041011	0.219992	1
test ping 3.jpg	ping	ping 4.jpg	ping	99.29993	0	99.29993	0.027991	0.080347	0.415099	1

test ping 4.jpg	ping	ping	ping 4.jpg	ping	97.29793	0	97.29793	0.025002	0.059691	0.269264	1
test ping 5.jpg	ping	ping	ping 4.jpg	ping	97.30555	0	97.30555	0.027997	0.085915	0.392014	1
test porcafe 1.jpg	porcafe	porcafe	ping 4.jpg	ping	105.4496			0.024001	0.088776	0.355797	0
test porcafe 2.jpg	porcafe	starbucks 6.jpg	starbucks	starbucks	93.31948			0.023977	0.061342	0.26779	0
test porcafe 3.jpg	porcafe	ping 4.jpg	ping	82.41403			0.015648	0.03997	0.202652	0	
test porcafe 4.jpg	porcafe	ping 4.jpg	ping	82.40263			0.02498	0.059302	0.244781	0	
test porcafe 5.jpg	porcafe	starbucks 6.jpg	starbucks	starbucks	99.56068			0.019818	0.058623	0.266564	0
test starbucks 1.jpg	starbucks	starbucks 6.jpg	starbucks	starbucks	76.44157	0	76.44157	0.024839	0.055024	0.286909	1
test starbucks 2.jpg	starbucks	ping 4.jpg	ping	108.3339	2	77.69	0.024111	0.100628	0.426194	0	
test starbucks 3.jpg	starbucks	ping 4.jpg	ping	107.8229	1	71.74954	0.029745	0.119199	0.463614	0	
test starbucks 4.jpg	starbucks	ping 4.jpg	ping	76.49786	1	71.90027	0.022136	0.044085	0.238706	0	
test starbucks 5.jpg	starbucks	ping 4.jpg	ping	79.38198	1	69.9007	0.025021	0.0783	0.37194	0	
test toyota 1.jpg	toyota	ping 4.jpg	ping	112.2303			0.022383	0.082721	0.331542	0	
test toyota 2.jpg	toyota	ping 4.jpg	ping	115.3542			0.030001	0.106296	0.431296	0	
test toyota 3.jpg	toyota	ping 4.jpg	ping	124.9036			0.025979	0.08799	0.369776	0	
test toyota 4.jpg	toyota	ping 4.jpg	ping	117.2509			0.03047	0.089	0.465718	0	

test toyota 5.jpg	toyota	ping 4.jpg	ping	103.3882			0.023003	0.074192	0.322186	0
test yogya 1.jpg	yogya	ping 4.jpg	ping	94.25256			0.02199	0.036998	0.192021	0
test yogya 2.jpg	yogya	ping 4.jpg	ping	71.65961			0.023977	0.040266	0.174545	0
test yogya 3.jpg	yogya	ping 4.jpg	ping	94.88474			0.022975	0.058669	0.281146	0
test yogya 4.jpg	yogya	ping 4.jpg	ping	101.6426			0.025975	0.063924	0.282218	0
test yogya 5.jpg	yogya	ping 4.jpg	ping	103.2069			0.027006	0.083855	0.335692	0

Tabel B.10: Hasil pengujian menggunakan metode SIFT pada keseluruhan dataset GSV 600.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	alfamart 9.jpg	alfamart	43.88283891	0	43.88283891	0.048996449	3.262537003	3.555122375	1
test al-famart 2.jpg	alfamart	alfamart 6.jpg	alfamart	134.9010307	0	134.9010307	0.044626713	3.284591913	3.612622499	1
test al-famart 3.jpg	alfamart	alfamart 5.jpg	alfamart	342.9367564	0	342.9367564	0.038006544	3.160750151	3.419305325	1
test al-famart 4.jpg	alfamart	alfamart 9.jpg	alfamart	85.04772329	0	85.04772329	0.042081118	3.294008732	3.566497564	1
test al-famart 5.jpg	alfamart	alfamart 4.jpg	alfamart	80.43082757	0	80.43082757	0.057245731	3.311172962	3.678896904	1
test binus 1.jpg	binus	binus 1.jpg	binus	2477.985088	0	2477.985088	0.04843998	3.103140831	3.622117043	1
test binus 2.jpg	binus	binus 10.jpg	binus	526.8094294	0	526.8094294	0.046061754	3.12247014	3.322226286	1
test binus 3.jpg	binus	binus 5.jpg	binus	486.2411278	0	486.2411278	0.050706387	3.113040209	3.492045879	1
test binus 4.jpg	binus	binus 2.jpg	binus	1257.962803	0	1257.962803	0.050467253	3.081514359	3.38315773	1
test binus 5.jpg	binus	binus 1.jpg	binus	416.5965367	0	416.5965367	0.050168276	3.219418764	3.480409622	1
test gembul 1.jpg	gembul	gembul 1.jpg	gembul	3153.062754	0	3153.062754	0.053000212	3.159463882	3.881484032	1
test gembul 2.jpg	gembul	gembul 1.jpg	gembul	840.0560626	0	840.0560626	0.056995869	3.234839439	3.481710434	1

test gembul 3.jpg	gembul	gembul 10.jpg	gembul	836.2810269	0	836.2810269	0.064382315	3.270531654	3.8396523	1
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	2888.069283	0	2888.069283	0.04312396	3.215966225	4.291336536	1
test gembul 5.jpg	gembul	gembul 1.jpg	gembul	769.4739181	0	769.4739181	0.049312353	3.138177395	3.376404524	1
test harris 1.jpg	harris	harris 2.jpg	harris	257.1018821	0	257.1018821	0.036771774	3.18898797	3.440978289	1
test harris 2.jpg	harris	harris 7.jpg	harris	621.7807488	0	621.7807488	0.049028873	3.254989147	3.611975193	1
test harris 3.jpg	harris	harris 7.jpg	harris	875.9768975	0	875.9768975	0.050001621	3.260987043	3.67698741	1
test harris 4.jpg	harris	harris 7.jpg	harris	369.5350071	0	369.5350071	0.052025557	3.197551012	3.502576113	1
test harris 5.jpg	harris	harris 6.jpg	harris	425.5212674	0	425.5212674	0.051980734	3.172958136	3.514937162	1
test oxy 1.jpg	oxy	oxy 10.jpg	oxy	1021.027193	0	1021.027193	0.05300498	3.305589437	3.846589565	1
test oxy 2.jpg	oxy	oxy 5.jpg	oxy	1311.874301	0	1311.874301	0.04903388	3.057975531	3.308977365	1
test oxy 3.jpg	oxy	oxy 3.jpg	oxy	2273.60263	0	2273.60263	0.050998449	3.059999943	3.576984406	1
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	465.5670098	0	465.5670098	0.049013615	3.115020037	3.324031591	1
test oxy 5.jpg	oxy	oxy 10.jpg	oxy	2074.335265	0	2074.335265	0.057973385	3.220016479	3.795028448	1
test ping 1.jpg	ping	ping 7.jpg	ping	216.8496391	0	216.8496391	0.054989338	3.20954299	3.540549994	1
test ping 2.jpg	ping	ping 7.jpg	ping	437.7218848	0	437.7218848	0.054975033	3.206999302	3.569973469	1
test ping 3.jpg	ping	ping 6.jpg	ping	3984.883211	0	3984.883211	0.049005985	3.273021936	7.28203392	1

test ping 4.jpg	ping	ping	ping 2.jpg	ping	419.4038328	0	419.4038328	0.054006338	3.273556471	3.924536467	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	3709.919246	0	3709.919246	0.05300951	3.283968687	5.375547886	1
test porcafe 1.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	2556.694529	0	2556.694529	0.042013168	3.203532457	4.399528742	1
test porcafe 2.jpg	porcafe	porcafe	porcafe 7.jpg	porcafe	3261.271562	0	3261.271562	0.048004627	3.23502326	4.520386934	1
test porcafe 3.jpg	porcafe	porcafe	porcafe 9.jpg	porcafe	2439.275357	0	2439.275357	0.03498888	3.164992571	3.761000395	1
test porcafe 4.jpg	porcafe	porcafe	porcafe 5.jpg	porcafe	838.0716427	0	838.0716427	0.052002192	3.221975565	3.531979084	1
test porcafe 5.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	40.15895734	0	40.15895734	0.05099678	3.322008848	3.663025379	1
test starbucks 1.jpg	starbucks	starbucks	starbucks 2.jpg	starbucks	100.6193337	0	100.6193337	0.054009438	3.192579746	3.448567152	1
test starbucks 2.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	1424.419272	0	1424.419272	0.049978495	3.377026796	4.134011984	1
test starbucks 3.jpg	starbucks	starbucks	starbucks 4.jpg	starbucks	1089.484019	0	1089.484019	0.060991287	3.348990202	4.235001802	1
test starbucks 4.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	822.2335314	0	822.2335314	0.046975613	3.204601526	3.726614475	1
test starbucks 5.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	989.9975126	0	989.9975126	0.052977562	3.432982206	4.275995493	1
test toyota 1.jpg	toyota	toyota	toyota 3.jpg	toyota	234.4881034	0	234.4881034	0.052979708	3.286000729	3.667031527	1
test toyota 2.jpg	toyota	toyota	toyota 10.jpg	toyota	56.90332566	0	56.90332566	0.059991837	3.422974348	3.923998833	1
test toyota 3.jpg	toyota	toyota	toyota 9.jpg	toyota	122.1180438	0	122.1180438	0.049977064	3.289608002	3.642620087	1
test toyota 4.jpg	toyota	toyota	toyota 9.jpg	toyota	236.3179442	0	236.3179442	0.057963848	3.28099227	3.690994263	1

test toyota 5.jpg	toyota	toyota 1.jpg	toyota	198.292157	0	198.292157	0.045001507	3.233022213	3.572004318	1
test yogya 1.jpg	yogya	yogya 6.jpg	yogya	1074.789651	0	1074.789651	0.045002222	3.143194199	3.567206383	1
test yogya 2.jpg	yogya	yogya 5.jpg	yogya	427.0954882	0	427.0954882	0.046999454	3.148017406	3.387029648	1
test yogya 3.jpg	yogya	yogya 10.jpg	yogya	486.471704	0	486.471704	0.042986393	3.215976477	3.512999058	1
test yogya 4.jpg	yogya	yogya 2.jpg	yogya	534.6569187	0	534.6569187	0.047965288	3.209018707	3.627022982	1
test yogya 5.jpg	yogya	yogya 2.jpg	yogya	1443.322157	0	1443.322157	0.056023598	3.325987577	3.968012333	1

Tabel B.11: Hasil pengujian menggunakan metode SIFT pada *dataset* GSV 600 yang telah tersaring dengan Threshold 1.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	starbucks 3.jpg	starbucks	68.54411686	2	57.19119493	0.04758811	0.632351398	0.993697882	0
test al-famart 2.jpg	alfamart	alfamart 1.jpg	alfamart	81.63951877	0	81.63951877	0.045012951	0.630872726	0.953374386	1
test al-famart 3.jpg	alfamart	alfamart 5.jpg	alfamart	180.1258911	0	180.1258911	0.042999744	0.588610411	0.80656004	1
test al-famart 4.jpg	alfamart	starbucks 8.jpg	starbucks	62.92588488	6	31.12953963	0.04499054	0.600402117	0.895377159	0
test al-famart 5.jpg	alfamart	starbucks 3.jpg	starbucks	75.98798898	8	31.7959663	0.054032564	0.685589075	1.099560976	0
test binus 1.jpg	binus	binus 1.jpg	binus	381.8953941	0	381.8953941	0.050028086	0.515964031	0.725239277	1
test binus 2.jpg	binus	binus 9.jpg	binus	115.5429075	0	115.5429075	0.052990675	0.494901657	0.653594971	1
test binus 3.jpg	binus	binus 5.jpg	binus	351.5259846	0	351.5259846	0.056459665	0.534397602	0.812419891	1
test binus 4.jpg	binus	binus 5.jpg	binus	192.4717382	0	192.4717382	0.058989286	0.497010231	0.683586121	1
test binus 5.jpg	binus	binus 6.jpg	binus	97.77537987	0	97.77537987	0.063994884	0.512980938	0.72801137	1
test gembul 1.jpg	gembul	gembul 1.jpg	gembul	793.0721061	0	793.0721061	0.056621552	0.523601055	0.753749609	1
test gembul 2.jpg	gembul	gembul 1.jpg	gembul	177.3351391	0	177.3351391	0.055176497	0.531260252	0.751717329	1

test gembul 3.jpg	gembul	gembul 10.jpg	gembul	464.4915166	0	464.4915166	0.062764406	0.699153423	1.17323947	1	
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	965.3573788	0	965.3573788	0.043227911	0.604784966	1.020493269	1	
test gembul 5.jpg	gembul	gembul 1.jpg	gembul	91.244535	0	91.244535	0.05629468	0.529677391	0.727311611	1	
test harris 1.jpg	harris	harris 2.jpg	harris	59.00699066	0	59.00699066	0.038342237	0.530070066	0.755784988	1	
test harris 2.jpg	harris	harris 1.jpg	harris	61.79510511	0	61.79510511	0.053803205	0.642782688	1.018257856	1	
test harris 3.jpg	harris	harris 7.jpg	harris	139.0113861	0	139.0113861	0.052625179	0.685531139	1.081039667	1	
test harris 4.jpg	harris	harris 7.jpg	harris	93.65341515	0	93.65341515	0.058035135	0.588820219	0.896430016	1	
test harris 5.jpg	harris	starbucks 7.jpg	starbucks	76.79964722	1	45.68879866	0.055999517	0.623890638	0.944160461	0	
test oxy 1.jpg	oxy	oxy 10.jpg	oxy	151.8738453	0	151.8738453	0.054596901	0.705008507	1.138389111	1	
test oxy 2.jpg	oxy	oxy 5.jpg	oxy	450.2183813	0	450.2183813	0.05582881	0.513012886	0.677093744	1	
test oxy 3.jpg	oxy	oxy 3.jpg	oxy	432.6305958	0	432.6305958	0.053319454	0.521878481	0.722848177	1	
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	195.1432868	0	195.1432868	0.05569315	0.521733761	0.72922945	1	
test oxy 5.jpg	oxy	oxy 10.jpg	oxy	89.51553109	0	89.51553109	0.060203791	0.64299798	1.030009508	1	
test ping 1.jpg	ping	ping	ping 7.jpg	ping	210.2243689	0	210.2243689	0.061021805	0.608015537	0.944672585	1
test ping 2.jpg	ping	ping	ping 1.jpg	ping	160.8484484	0	160.8484484	0.059754848	0.596907377	0.881237268	1
test ping 3.jpg	ping	ping	ping 6.jpg	ping	1403.467895	0	1403.467895	0.050317287	0.676408291	1.349802971	1

test ping 4.jpg	ping	ping	ping 2.jpg	ping	176.5114224	0	176.5114224	0.062478065	0.648492813	1.111064434	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	966.4306481	0	966.4306481	0.059245586	0.662539482	1.193323374	1
test porcafe 1.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	550.2511412	0	550.2511412	0.043705225	0.58941555	0.916722536	1
test porcafe 2.jpg	porcafe	porcafe	porcafe 7.jpg	porcafe	790.6267436	0	790.6267436	0.052260876	0.632974625	1.008014679	1
test porcafe 3.jpg	porcafe	porcafe	porcafe 9.jpg	porcafe	804.2630387	0	804.2630387	0.03555131	0.570047617	0.862792492	1
test porcafe 4.jpg	porcafe	porcafe	porcafe 5.jpg	porcafe	232.1503432	0	232.1503432	0.055901527	0.590989351	0.90805769	1
test porcafe 5.jpg	porcafe	starbucks	starbucks 3.jpg	starbucks	83.9004451	11	28.550996	0.048387289	0.656124592	1.043027401	0
test starbucks 1.jpg	starbucks	starbucks	starbucks 6.jpg	starbucks	89.04741318	0	89.04741318	0.051872492	0.590447187	0.867738724	1
test starbucks 2.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	725.0894553	0	725.0894553	0.053201437	0.715760469	1.362653017	1
test starbucks 3.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	576.7838868	0	576.7838868	0.062714815	0.735003471	1.457994938	1
test starbucks 4.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	602.9007083	0	602.9007083	0.051727533	0.546473265	0.924649477	1
test starbucks 5.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	688.3576366	0	688.3576366	0.054157972	0.704324961	1.372716665	1
test toyota 1.jpg	toyota	toyota	toyota 3.jpg	toyota	121.9034425	0	121.9034425	0.051965952	0.664485693	1.06567812	1
test toyota 2.jpg	toyota	toyota	starbucks 5.jpg	starbucks	75.00369249	11	33.02513484	0.063321829	0.782204628	1.308560133	0
test toyota 3.jpg	toyota	toyota	starbucks 6.jpg	starbucks	70.22309301	4	48.12609934	0.05434227	0.642469406	1.010777473	0
test toyota 4.jpg	toyota	toyota	starbucks 6.jpg	starbucks	74.60594997	2	52.78150847	0.059197187	0.69075799	1.099770546	0

test toyota 5.jpg	toyota	toyota 1.jpg	toyota	103.5738187	0	103.5738187	0.048144102	0.618160963	0.984055042	1
test yogya 1.jpg	yogya	yogya 6.jpg	yogya	465.2300768	0	465.2300768	0.04822731	0.550886154	0.8422544	1
test yogya 2.jpg	yogya	yogya 5.jpg	yogya	118.9258683	0	118.9258683	0.051003218	0.527394295	0.740731955	1
test yogya 3.jpg	yogya	yogya 10.jpg	yogya	116.7483333	0	116.7483333	0.047771692	0.573090553	0.868910074	1
test yogya 4.jpg	yogya	yogya 6.jpg	yogya	168.599995	0	168.599995	0.044658661	0.61245966	0.963150978	1
test yogya 5.jpg	yogya	yogya 2.jpg	yogya	306.2397692	0	306.2397692	0.058321476	0.650067091	1.090495825	1

Tabel B.12: Hasil pengujian menggunakan metode SIFT pada *dataset* GSV 600 yang telah tersaring dengan Threshold 2.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	gembul 10.jpg	gembul	104.4045347			0.058634281	0.148262262	0.49857688	0
test al-famart 2.jpg	alfamart	gembul 10.jpg	gembul	121.8132841			0.043330908	0.145573139	0.499787092	0
test al-famart 3.jpg	alfamart	gembul 6.jpg	gembul	85.80205463			0.036437511	0.097212791	0.31697607	0
test al-famart 4.jpg	alfamart	gembul 10.jpg	gembul	99.48777681			0.041983366	0.124009132	0.398210764	0
test al-famart 5.jpg	alfamart	gembul 10.jpg	gembul	138.4488737			0.053149462	0.187748909	0.587728024	0
test binus 1.jpg	binus	gembul 10.jpg	gembul	74.90250884	7	19.4093494	0.048416615	0.061199903	0.209976912	0
test binus 2.jpg	binus	gembul 10.jpg	gembul	78.81689342	5	27.37246984	0.045009851	0.05024147	0.195425034	0
test binus 3.jpg	binus	gembul 2.jpg	gembul	56.2362558	7	26.19918327	0.050123453	0.088178158	0.329859734	0
test binus 4.jpg	binus	gembul 10.jpg	gembul	50.61479313	7	11.93696502	0.046086073	0.056823254	0.229154348	0
test binus 5.jpg	binus	gembul 6.jpg	gembul	73.10682129	5	24.99033506	0.050095797	0.061731815	0.225030661	0
test gembul 1.jpg	gembul	gembul 10.jpg	gembul	88.73166716	0	88.73166716	0.048573256	0.075368643	0.274341822	1
test gembul 2.jpg	gembul	gembul 2.jpg	gembul	81.92359004	0	81.92359004	0.052199125	0.082169294	0.259227514	1

test gembul 3.jpg	gembul	gembul 3.jpg	gembul	115.8722111	0	115.8722111	0.05501318	0.16800189	0.572993755	1
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	156.838454	0	156.838454	0.04099822	0.137629271	0.444914103	1
test gembul 5.jpg	gembul	gembul 6.jpg	gembul	71.17337411	0	71.17337411	0.048987627	0.073999882	0.233363152	1
test harris 1.jpg	harris	gembul 10.jpg	gembul	76.72886326			0.034989119	0.089002371	0.297153711	0
test harris 2.jpg	harris	gembul 10.jpg	gembul	107.6790212			0.046980619	0.16618824	0.496013641	0
test harris 3.jpg	harris	gembul 2.jpg	gembul	91.7241994			0.046998262	0.174597979	0.553909779	0
test harris 4.jpg	harris	gembul 10.jpg	gembul	93.86548828			0.048988581	0.127175808	0.399330854	0
test harris 5.jpg	harris	gembul 3.jpg	gembul	93.84409299			0.048172474	0.13899684	0.433791161	0
test oxy 1.jpg	oxy	gembul 10.jpg	gembul	113.3036672			0.051999569	0.168874502	0.543315887	0
test oxy 2.jpg	oxy	gembul 10.jpg	gembul	66.51482306			0.056229353	0.059994698	0.197999001	0
test oxy 3.jpg	oxy	gembul 2.jpg	gembul	66.70573169			0.048007011	0.067012548	0.210409403	0
test oxy 4.jpg	oxy	gembul 10.jpg	gembul	60.72609503			0.048989773	0.078011513	0.237395525	0
test oxy 5.jpg	oxy	gembul 10.jpg	gembul	115.4690946			0.051998615	0.151997805	0.530278683	0
test ping 1.jpg	ping	gembul 6.jpg	gembul	75.82631782	11	9.008480319	0.052641392	0.14131999	0.608866215	0
test ping 2.jpg	ping	gembul 10.jpg	gembul	79.40780525	11	0	0.051967382	0.140021086	0.556052685	0
test ping 3.jpg	ping	gembul 6.jpg	gembul	106.9148665	7	36.77579523	0.0449121	0.178496838	0.581241131	0

test ping 4.jpg	ping	ping	gembul 10.jpg	gembul	116.7854397	10	10.35144556	0.052008152	0.149537563	0.529401541	0
test ping 5.jpg	ping	ping	gembul 10.jpg	gembul	118.620442	10	31.32272185	0.051983595	0.181717873	0.632534266	0
test porcafe 1.jpg	porcafe	porcafe	gembul 10.jpg	gembul	94.55820539			0.040988684	0.117250204	0.387277603	0
test porcafe 2.jpg	porcafe	porcafe	gembul 6.jpg	gembul	97.59599828			0.048000574	0.145608902	0.480243683	0
test porcafe 3.jpg	porcafe	porcafe	gembul 10.jpg	gembul	104.9601554			0.03187418	0.103128672	0.338964462	0
test porcafe 4.jpg	porcafe	porcafe	gembul 10.jpg	gembul	117.2342694			0.04984045	0.121260166	0.418880463	0
test porcafe 5.jpg	porcafe	porcafe	gembul 10.jpg	gembul	124.3900102			0.044667959	0.163007975	0.57552743	0
test starbucks 1.jpg	starbucks	starbucks	gembul 10.jpg	gembul	83.75775481			0.047048092	0.118672848	0.38473773	0
test starbucks 2.jpg	starbucks	starbucks	gembul 3.jpg	gembul	116.1420384			0.049247742	0.191362858	0.668661118	0
test starbucks 3.jpg	starbucks	starbucks	gembul 3.jpg	gembul	109.546913			0.055585861	0.238063335	0.773079157	0
test starbucks 4.jpg	starbucks	starbucks	gembul 3.jpg	gembul	82.32261283			0.044500113	0.094118357	0.327900887	0
test starbucks 5.jpg	starbucks	starbucks	gembul 10.jpg	gembul	128.9702257			0.048027277	0.20250535	0.689183235	0
test toyota 1.jpg	toyota	toyota	gembul 6.jpg	gembul	115.8521563			0.048976183	0.215321064	0.601758003	0
test toyota 2.jpg	toyota	toyota	gembul 10.jpg	gembul	123.4366701			0.053981304	0.232186556	0.778404713	0
test toyota 3.jpg	toyota	toyota	gembul 10.jpg	gembul	95.9946753			0.053717375	0.152093172	0.49998641	0
test toyota 4.jpg	toyota	toyota	gembul 6.jpg	gembul	109.6872352			0.049134016	0.197367907	0.609149456	0

test toyota 5.jpg	toyota	gembul 10.jpg	gembul	101.1012217			0.04497838	0.143386602	0.466284752	0
test yogya 1.jpg	yogya	gembul 10.jpg	gembul	74.95336078			0.041022301	0.094233513	0.301909685	0
test yogya 2.jpg	yogya	gembul 10.jpg	gembul	87.35912461			0.047002316	0.081689358	0.282901764	0
test yogya 3.jpg	yogya	gembul 2.jpg	gembul	81.30135172			0.040315866	0.108581543	0.385013103	0
test yogya 4.jpg	yogya	gembul 10.jpg	gembul	91.83315285			0.042456865	0.133542061	0.428768396	0
test yogya 5.jpg	yogya	gembul 10.jpg	gembul	117.3530744			0.054987192	0.166488647	0.559195042	0

Tabel B.13: Hasil pengujian menggunakan metode ORB pada keseluruhan dataset GSV 400.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	harris 7.jpg	harris	115.3629325	8	67.38330413	0.003996372	0.509679317	1.012064695	0
test al-famart 2.jpg	alfamart	yogya 10.jpg	yogya	78.90221678	11	66.98417272	0.002978086	0.497779369	0.968752384	0
test al-famart 3.jpg	alfamart	alfamart 10.jpg	alfamart	112.2922681	0	112.2922681	0.002011061	0.450865269	0.921067715	1
test al-famart 4.jpg	alfamart	starbucks 1.jpg	starbucks	103.7184505	2	71.38236644	0.002999544	0.442101479	0.928516626	0
test al-famart 5.jpg	alfamart	yogya 8.jpg	yogya	98.23974521	1	76.51745805	0.004854202	0.430342913	0.912137508	0
test binus 1.jpg	binus	binus 1.jpg	binus	1833.981479	0	1833.981479	0.00403285	0.45724225	1.082997322	1
test binus 2.jpg	binus	binus 5.jpg	binus	568.1456341	0	568.1456341	0.00289607	0.440426826	0.944591284	1
test binus 3.jpg	binus	binus 5.jpg	binus	485.170858	0	485.170858	0.003010035	0.421118975	0.886862278	1
test binus 4.jpg	binus	binus 2.jpg	binus	1282.764581	0	1282.764581	0.003200054	0.46640873	1.048002958	1
test binus 5.jpg	binus	binus 1.jpg	binus	417.4537462	0	417.4537462	0.002995729	0.508383751	1.04293108	1
test gembul 1.jpg	gembul	gembul 1.jpg	gembul	1524.906154	0	1524.906154	0.003988266	0.486575842	1.051190615	1
test gembul 2.jpg	gembul	gembul 1.jpg	gembul	242.6257916	0	242.6257916	0.002995729	0.449454308	0.962724686	1

test gembul 3.jpg	gembul	gembul 1.jpg	gembul	263.7575793	0	263.7575793	0.004963398	0.40110445	0.851081133	1
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	681.5082871	0	681.5082871	0.003000259	0.427312851	0.93131876	1
test gembul 5.jpg	gembul	gembul 1.jpg	gembul	85.51267169	0	85.51267169	0.003976345	0.447009802	0.952071667	1
test harris 1.jpg	harris	harris 6.jpg	harris	242.448513	0	242.448513	0.001997709	0.422000647	0.901009083	1
test harris 2.jpg	harris	oxy 4.jpg	oxy	104.4213422	9	63.19509595	0.001999378	0.427999973	0.939972639	0
test harris 3.jpg	harris	oxy 5.jpg	oxy	82.5608035	3	69.70184707	0.003997564	0.405013561	0.890013695	0
test harris 4.jpg	harris	harris 7.jpg	harris	132.3242643	0	132.3242643	0.003014326	0.439997911	0.948996067	1
test harris 5.jpg	harris	harris 6.jpg	harris	178.8936352	0	178.8936352	0.004012346	0.432003021	0.942978859	1
test oxy 1.jpg	oxy	oxy 10.jpg	oxy	293.6464461	0	293.6464461	0.003998995	0.427005768	0.908008337	1
test oxy 2.jpg	oxy	oxy 5.jpg	oxy	667.9930644	0	667.9930644	0.002993584	0.412022114	0.909995556	1
test oxy 3.jpg	oxy	oxy 3.jpg	oxy	2747.415983	0	2747.415983	0.002997637	0.437006474	1.254989147	1
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	539.5381663	0	539.5381663	0.002002954	0.433030844	0.948010206	1
test oxy 5.jpg	oxy	oxy 10.jpg	oxy	265.4100819	0	265.4100819	0.004019737	0.474969864	0.994949579	1
test ping 1.jpg	ping	ping	ping 1.jpg	ping	424.5135288	0	424.5135288	0.004003525	0.425005913	1.00599885
test ping 2.jpg	ping	ping	ping 1.jpg	ping	595.8912916	0	595.8912916	0.003999472	0.4500494	1.0610044
test ping 3.jpg	ping	ping	ping 6.jpg	ping	2213.758522	0	2213.758522	0.003999472	0.426048756	1.339015484

test ping 4.jpg	ping	ping	ping 2.jpg	ping	516.6702198	0	516.6702198	0.004000187	0.470989466	1.006373882	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	1698.041652	0	1698.041652	0.002523661	0.460249662	1.054122925	1
test porcafe 1.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	1286.014486	0	1286.014486	0.003995895	0.474918842	1.15133357	1
test porcafe 2.jpg	porcafe	porcafe	porcafe 7.jpg	porcafe	2113.628946	0	2113.628946	0.0045259	0.426879883	1.252082109	1
test porcafe 3.jpg	porcafe	porcafe	porcafe 9.jpg	porcafe	1777.425835	0	1777.425835	0.002994299	0.484643221	1.055940628	1
test porcafe 4.jpg	porcafe	porcafe	porcafe 5.jpg	porcafe	106.8276829	0	106.8276829	0.00399518	0.462921143	0.963189363	1
test porcafe 5.jpg	porcafe	binus	binus 6.jpg	binus	92.28462992	1	89.50430045	0.003032684	0.424873352	0.912909746	0
test starbucks 1.jpg	starbucks	oxy	oxy 2.jpg	oxy	90.34187912	2	80.04758998	0.002998352	0.433583736	0.950669527	0
test starbucks 2.jpg	starbucks	starbucks	starbucks 6.jpg	starbucks	444.3883132	0	444.3883132	0.003584385	0.427221775	0.919077396	1
test starbucks 3.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	206.7380635	0	206.7380635	0.004002094	0.426140308	0.967045546	1
test starbucks 4.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	332.6596789	0	332.6596789	0.002945185	0.461506605	0.958499432	1
test starbucks 5.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	786.6935911	0	786.6935911	0.004013062	0.430009127	0.954998255	1
test toyota 1.jpg	toyota	porcafe	porcafe 3.jpg	porcafe	101.8854018	10	61.95720902	0.004009962	0.433990955	0.912994385	0
test toyota 2.jpg	toyota	binus	binus 10.jpg	binus	97.4207851	3	83.98756083	0.00399971	0.42899847	0.910991907	0
test toyota 3.jpg	toyota	toyota	toyota 9.jpg	toyota	106.0103281	0	106.0103281	0.004000187	0.419024706	0.919023514	1
test toyota 4.jpg	toyota	toyota	toyota 1.jpg	toyota	146.1385514	0	146.1385514	0.003999949	0.434020758	0.925981283	1

test toyota 5.jpg	toyota	ping 1.jpg	ping	122.8381269	6	73.67768026	0.003012896	0.438017607	0.938979149	0
test yogya 1.jpg	yogya	yogya 6.jpg	yogya	697.6708744	0	697.6708744	0.001997709	0.425995111	0.983022451	1
test yogya 2.jpg	yogya	yogya 5.jpg	yogya	268.9906618	0	268.9906618	0.001998425	0.432976007	0.917968273	1
test yogya 3.jpg	yogya	yogya 10.jpg	yogya	717.3109685	0	717.3109685	0.003013372	0.437981367	0.989986897	1
test yogya 4.jpg	yogya	yogya 3.jpg	yogya	124.9211068	0	124.9211068	0.003985643	0.423966408	0.941985607	1
test yogya 5.jpg	yogya	yogya 2.jpg	yogya	210.5393094	0	210.5393094	0.003010988	0.432996511	0.916000128	1

Tabel B.14: Hasil pengujian menggunakan metode ORB pada *dataset* GSV 400 yang telah tersaring dengan Threshold 1.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract ti-me	pairing ti-me	total bsis time	is true
test al-famart 1.jpg	alfamart			0			0.004016399	0.026979446	0.045979738	0
test al-famart 2.jpg	alfamart			0			0.002999544	0.023694992	0.041701078	0
test al-famart 3.jpg	alfamart	yogya 10.jpg	yogya	12.08961866			0.002975225	0.02198863	0.045997381	0
test al-famart 4.jpg	alfamart			0			0.002990723	0.022996664	0.045002937	0
test al-famart 5.jpg	alfamart			0	1	0	0.00400281	0.022994518	0.046007633	0
test binus 1.jpg	binus			0	0	0	0.003991842	0.024016619	0.047994614	0
test binus 2.jpg	binus	binus 5.jpg	binus	9.371208184	0	9.371208184	0.002997398	0.021996737	0.054020643	1
test binus 3.jpg	binus			0			0.00300312	0.017010212	0.034998417	0
test binus 4.jpg	binus			0			0.00301218	0.018996477	0.040995598	0
test binus 5.jpg	binus			0			0.003556013	0.017990828	0.038972616	0
test gembul 1.jpg	gembul			0	0	0	0.002997398	0.02400732	0.044509649	0
test gembul 2.jpg	gembul			0			0.003000021	0.022580862	0.040448904	0

test gembul 3.jpg	gembul			0			0.003952503	0.018748283	0.037472486	0
test gembul 4.jpg	gembul			0			0.004296064	0.020074368	0.036376238	0
test gembul 5.jpg	gembul			0			0.005033731	0.018042564	0.038248062	0
test harris 1.jpg	harris			0	1	0	0.003000021	0.017612696	0.037378073	0
test harris 2.jpg	harris			0	1	0	0.004011393	0.018667221	0.040077925	0
test harris 3.jpg	harris			0			0.003988504	0.019818544	0.042375326	0
test harris 4.jpg	harris			0			0.002999544	0.020922899	0.043132782	0
test harris 5.jpg	harris			0			0.003000736	0.021686077	0.04032135	0
test oxy 1.jpg	oxy			0	0	0	0.003764868	0.019115448	0.03995347	0
test oxy 2.jpg	oxy			0	1	0	0.003012896	0.018719196	0.040518761	0
test oxy 3.jpg	oxy			0	2	0	0.003480673	0.019899368	0.043133974	0
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	15.29542898	0	15.29542898	0.002120495	0.024163485	0.051350832	1
test oxy 5.jpg	oxy			0			0.003000498	0.020997524	0.039524794	0
test ping 1.jpg	ping			0	0	0	0.004030704	0.017943621	0.040619373	0
test ping 2.jpg	ping			0			0.003926754	0.017842531	0.034915686	0
test ping 3.jpg	ping	ping 6.jpg	ping	25.08807828	0	25.08807828	0.003195524	0.016947746	0.04248023	1

test ping 4.jpg	ping	ping			0	0	0	0.003012657	0.017796993	0.041083813	0
test ping 5.jpg	ping	ping			0	1	0	0.003997803	0.019356251	0.041255474	0
test porcafe 1.jpg	porcafe	porcafe 1.jpg	porcafe	porcafe	15.23745577	0	15.23745577	0.002147913	0.020633936	0.045102119	1
test porcafe 2.jpg	porcafe	porcafe 2.jpg	porcafe	porcafe	60.03217563	0	60.03217563	0.003542423	0.017890453	0.049573421	1
test porcafe 3.jpg	porcafe	porcafe 9.jpg	porcafe	porcafe	12.89258941	0	12.89258941	0.00201273	0.01935339	0.041781187	1
test porcafe 4.jpg	porcafe				0	0	0	0.004282475	0.021280766	0.045904636	0
test porcafe 5.jpg	porcafe				0			0.003129721	0.019881725	0.043269873	0
test starbucks 1.jpg	starbucks				0			0.002986431	0.019183159	0.039845943	0
test starbucks 2.jpg	starbucks				0			0.004060984	0.017832994	0.041587114	0
test starbucks 3.jpg	starbucks				0	1	0	0.004209995	0.020775557	0.043409109	0
test starbucks 4.jpg	starbucks				0	2	0	0.003023386	0.021269798	0.048790216	0
test starbucks 5.jpg	starbucks				0	1	0	0.003277063	0.017935753	0.038324356	0
test toyota 1.jpg	toyota				0			0.003020525	0.017514706	0.035768032	0
test toyota 2.jpg	toyota				0			0.005959034	0.017309189	0.034406662	0
test toyota 3.jpg	toyota				0			0.005010128	0.016882181	0.03584528	0
test toyota 4.jpg	toyota				0			0.004785299	0.020056963	0.040662527	0

test toyota 5.jpg	toyota			0			0.00201273	0.018071413	0.038351059	0
test yogya 1.jpg	yogya			0			0.002193689	0.021857023	0.041414499	0
test yogya 2.jpg	yogya			0			0.003007412	0.019634247	0.043548584	0
test yogya 3.jpg	yogya	yogya 10.jpg	yogya	10.72827041	0	10.72827041	0.001999617	0.017096281	0.040552378	1
test yogya 4.jpg	yogya			0			0.003247976	0.019891739	0.040234566	0
test yogya 5.jpg	yogya			0			0.003032446	0.016096592	0.039158583	0

Tabel B.15: Hasil pengujian menggunakan metode ORB pada keseluruhan dataset GSV 600.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	oxy 6.jpg	oxy	98.02000954	4	70.96307549	0.006000042	0.515535355	1.018949986	0
test al-famart 2.jpg	alfamart	gembul 5.jpg	gembul	88.82308594	3	83.56963469	0.00541091	0.493768692	0.990065336	0
test al-famart 3.jpg	alfamart	alfamart 8.jpg	alfamart	120.3330479	0	120.3330479	0.003983021	0.486436844	1.004821062	1
test al-famart 4.jpg	alfamart	binus 9.jpg	binus	93.35607694	4	74.26867471	0.004867792	0.466839314	0.946701765	0
test al-famart 5.jpg	alfamart	porcafe 5.jpg	porcafe	94.82801882	1	84.64500036	0.007693768	0.473519087	0.983357191	0
test binus 1.jpg	binus	binus 1.jpg	binus	1100.298427	0	1100.298427	0.004099607	0.475098372	1.011099577	1
test binus 2.jpg	binus	binus 5.jpg	binus	162.18595	0	162.18595	0.00689888	0.493988037	1.013700247	1
test binus 3.jpg	binus	binus 5.jpg	binus	448.1506329	0	448.1506329	0.005346775	0.458014965	0.950614691	1
test binus 4.jpg	binus	binus 2.jpg	binus	1507.588176	0	1507.588176	0.005612612	0.45100832	1.048827648	1
test binus 5.jpg	binus	binus 2.jpg	binus	361.4753554	0	361.4753554	0.006944418	0.467936277	0.971553802	1
test gembul 1.jpg	gembul	gembul 1.jpg	gembul	1965.153859	0	1965.153859	0.004076242	0.477442741	1.095779896	1
test gembul 2.jpg	gembul	gembul 1.jpg	gembul	282.7131286	0	282.7131286	0.004071474	0.458084345	0.983215332	1

test gembul 3.jpg	gembul	gembul 1.jpg	gembul	175.8913229	0	175.8913229	0.006011009	0.417621136	0.907322884	1
test gembul 4.jpg	gembul	gembul 10.jpg	gembul	1231.816995	0	1231.816995	0.007021189	0.4550879	0.998720646	1
test gembul 5.jpg	gembul	gembul 1.jpg	gembul	429.4797157	0	429.4797157	0.007152081	0.457659721	0.978624105	1
test harris 1.jpg	harris	harris 6.jpg	harris	255.6591396	0	255.6591396	0.005017996	0.458313465	0.963541508	1
test harris 2.jpg	harris	toyota 9.jpg	toyota	93.93968348	13	55.09322125	0.005034685	0.438382387	0.919789076	0
test harris 3.jpg	harris	yogya 4.jpg	yogya	113.5911977	7	63.15045974	0.005997658	0.431476355	0.913661242	0
test harris 4.jpg	harris	harris 7.jpg	harris	108.1636295	0	108.1636295	0.005999804	0.470335484	0.943287134	1
test harris 5.jpg	harris	harris 6.jpg	harris	285.0220291	0	285.0220291	0.005975485	0.456088543	0.939261436	1
test oxy 1.jpg	oxy	oxy 10.jpg	oxy	270.6399602	0	270.6399602	0.006000757	0.44946003	0.967087746	1
test oxy 2.jpg	oxy	oxy 5.jpg	oxy	365.6984235	0	365.6984235	0.003988743	0.446508408	0.953468084	1
test oxy 3.jpg	oxy	oxy 3.jpg	oxy	2564.347178	0	2564.347178	0.006576061	0.455507994	1.196181774	1
test oxy 4.jpg	oxy	oxy 5.jpg	oxy	215.7661073	0	215.7661073	0.00402832	0.480512142	1.001801729	1
test oxy 5.jpg	oxy	oxy 10.jpg	oxy	261.1097037	0	261.1097037	0.006973505	0.448095798	0.962239027	1
test ping 1.jpg	ping	ping	ping 1.jpg	ping	448.3387826	0	448.3387826	0.00694561	0.431566238	1.161633968
test ping 2.jpg	ping	ping	ping 1.jpg	ping	810.5726662	0	810.5726662	0.006000042	0.432027102	1.173571348
test ping 3.jpg	ping	ping	ping 6.jpg	ping	2803.98064	0	2803.98064	0.009654522	0.428393364	1.392712116

test ping 4.jpg	ping	ping	ping 2.jpg	ping	755.6585043	0	755.6585043	0.006955624	0.435856581	1.048986435	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	1699.777089	0	1699.777089	0.001912355	0.455510378	1.035294533	1
test porcafe 1.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	2050.698446	0	2050.698446	0.00600028	0.446958065	1.280272484	1
test porcafe 2.jpg	porcafe	porcafe	porcafe 7.jpg	porcafe	2298.822089	0	2298.822089	0.006995201	0.449655294	1.282984257	1
test porcafe 3.jpg	porcafe	porcafe	porcafe 9.jpg	porcafe	2552.173103	0	2552.173103	0.00500083	0.438646078	1.127622128	1
test porcafe 4.jpg	porcafe	porcafe	porcafe 1.jpg	porcafe	94.46805111	0	94.46805111	0.004996538	0.467915773	0.964490652	1
test porcafe 5.jpg	porcafe	harris	harris 4.jpg	harris	89.02910231	5	72.96399677	0.005012274	0.438464403	0.927491665	0
test starbucks 1.jpg	starbucks	binus	binus 9.jpg	binus	102.0617054	4	83.70536049	0.006028891	0.465997458	0.991386414	0
test starbucks 2.jpg	starbucks	starbucks	starbucks 6.jpg	starbucks	377.4237915	0	377.4237915	0.005995989	0.461549997	0.975587606	1
test starbucks 3.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	403.2373869	0	403.2373869	0.006999969	0.453007698	0.97942996	1
test starbucks 4.jpg	starbucks	starbucks	starbucks 5.jpg	starbucks	245.9459008	0	245.9459008	0.004994392	0.486495018	1.00554204	1
test starbucks 5.jpg	starbucks	starbucks	starbucks 3.jpg	starbucks	1135.853638	0	1135.853638	0.006013155	0.431210518	1.002939463	1
test toyota 1.jpg	toyota	toyota	toyota 4.jpg	toyota	89.33011786	0	89.33011786	0.006006241	0.452978611	0.947360754	1
test toyota 2.jpg	toyota	toyota	yogya 2.jpg	yogya	80.664355	11	61.72073886	0.009000063	0.460381746	0.947689295	0
test toyota 3.jpg	toyota	toyota	toyota 9.jpg	toyota	96.4757343	0	96.4757343	0.00700736	0.449977398	0.947862387	1
test toyota 4.jpg	toyota	toyota	toyota 9.jpg	toyota	86.39838255	0	86.39838255	0.008000612	0.466035128	0.96476078	1

test toyota 5.jpg	toyota	yogya 10.jpg	yogya	78.03486993	4	67.33056181	0.004975319	0.44860673	0.948763609	0
test yogya 1.jpg	yogya	yogya 5.jpg	yogya	561.7900707	0	561.7900707	0.004987001	0.438018799	0.954960823	1
test yogya 2.jpg	yogya	yogya 5.jpg	yogya	218.9978714	0	218.9978714	0.005024672	0.452224731	0.947485209	1
test yogya 3.jpg	yogya	yogya 10.jpg	yogya	593.9163864	0	593.9163864	0.004978657	0.444640636	0.963627338	1
test yogya 4.jpg	yogya	yogya 3.jpg	yogya	203.648762	0	203.648762	0.003175259	0.45994544	0.968565226	1
test yogya 5.jpg	yogya	yogya 2.jpg	yogya	330.940293	0	330.940293	0.006999969	0.441601515	0.937046766	1

Tabel B.16: Hasil pengujian menggunakan metode ORB pada *dataset* GSV 600 yang telah tersaring dengan Threshold 1.

q name	q class	most simi-lar	most similar class	total weight	same class idx	same class weight	extract time	pairing time	total bsis time	is true
test al-famart 1.jpg	alfamart	ping 7.jpg	ping	20.8604701	4	0	0.006008148	0.022979736	0.066141367	0
test al-famart 2.jpg	alfamart	binus 5.jpg	binus	9.19251838			0.00500226	0.025639057	0.054410219	0
test al-famart 3.jpg	alfamart	binus 6.jpg	binus	15.89060277	10	0	0.005006075	0.03198576	0.083001614	0
test al-famart 4.jpg	alfamart	starbucks 8.jpg	starbucks	10.1375411			0.005009174	0.025991917	0.056002855	0
test al-famart 5.jpg	alfamart	starbucks 8.jpg	starbucks	23.14023404	9	0	0.007982731	0.026988029	0.07100153	0
test binus 1.jpg	binus	binus 2.jpg	binus	55.95640791	0	55.95640791	0.005998135	0.024987936	0.064000368	1
test binus 2.jpg	binus	ping 7.jpg	ping	21.57897371	3	0	0.003974915	0.030212641	0.080129147	0
test binus 3.jpg	binus	binus 2.jpg	binus	44.86453444	0	44.86453444	0.00517416	0.021971464	0.055959463	1
test binus 4.jpg	binus	binus 2.jpg	binus	170.4176966	0	170.4176966	0.004495859	0.021724701	0.060692787	1
test binus 5.jpg	binus	binus 2.jpg	binus	57.42729101	0	57.42729101	0.006189108	0.023540974	0.060808659	1
test gembul 1.jpg	gembul	binus 2.jpg	binus	34.27982067	3	0	0.005182981	0.025093079	0.059687376	0
test gembul 2.jpg	gembul			0			0.005276442	0.022382975	0.052275896	0

test gembul 3.jpg	gembul	gembul 7.jpg	gembul	9.461450319	0	9.461450319	0.005555391	0.020147562	0.048331022	1	
test gembul 4.jpg	gembul			0			0.005937576	0.023094177	0.05683589	0	
test gembul 5.jpg	gembul			0			0.005687237	0.020432949	0.048498869	0	
test harris 1.jpg	harris	ping 4.jpg	ping	11.8192021	6	0	0.004793167	0.022020102	0.056742907	0	
test harris 2.jpg	harris	binus 5.jpg	binus	27.32960701	5	0	0.006089211	0.024019957	0.062149525	0	
test harris 3.jpg	harris	binus 5.jpg	binus	21.90174943			0.00699544	0.020582914	0.05191803	0	
test harris 4.jpg	harris	starbucks 8.jpg	starbucks	19.62089729			0.005176306	0.025845766	0.064424515	0	
test harris 5.jpg	harris	binus 5.jpg	binus	10.55273398			0.005521774	0.025081396	0.064037085	0	
test oxy 1.jpg	oxy	oxy	starbucks 4.jpg	starbucks	31.80001758	1	11.17425626	0.006445408	0.021164656	0.058575153	0
test oxy 2.jpg	oxy	oxy	binus 2.jpg	binus	26.12903537	1	14.60967904	0.00399828	0.022943258	0.075842142	0
test oxy 3.jpg	oxy	oxy	binus 9.jpg	binus	14.94233619	1	12.86697633	0.005514145	0.021085501	0.05410409	0
test oxy 4.jpg	oxy	oxy	starbucks 8.jpg	starbucks	29.46745044	3	0	0.004256964	0.024983406	0.071964264	0
test oxy 5.jpg	oxy			0	2	0	0.006210566	0.021220207	0.044752598	0	
test ping 1.jpg	ping	ping	ping 2.jpg	ping	62.63267314	0	62.63267314	0.006851912	0.019486189	0.054696083	1
test ping 2.jpg	ping	ping	ping 2.jpg	ping	52.67439492	0	52.67439492	0.005617619	0.019665003	0.059036016	1
test ping 3.jpg	ping	ping	ping 9.jpg	ping	93.0782574	0	93.0782574	0.005985022	0.018827438	0.061584949	1

test ping 4.jpg	ping	ping	ping 2.jpg	ping	27.69417096	0	27.69417096	0.007473946	0.019203663	0.051379204	1
test ping 5.jpg	ping	ping	ping 10.jpg	ping	99.12106282	0	99.12106282	0.00517869	0.023225546	0.063196898	1
test porcafe 1.jpg	porcafe	starbucks	starbucks 4.jpg	starbucks	11.23128443			0.005095959	0.020941257	0.0499053	0
test porcafe 2.jpg	porcafe	binus	binus 10.jpg	binus	10.61794292	7	0	0.005057335	0.018846035	0.053706169	0
test porcafe 3.jpg	porcafe	binus	binus 7.jpg	binus	10.76971201			0.004274607	0.018435717	0.045514584	0
test porcafe 4.jpg	porcafe	binus	binus 2.jpg	binus	24.22112039			0.005754948	0.022321463	0.062687874	0
test porcafe 5.jpg	porcafe	starbucks	starbucks 8.jpg	starbucks	19.30520256	1	0	0.005438089	0.021178961	0.056523085	0
test starbucks 1.jpg	starbucks	binus	binus 2.jpg	binus	22.91802141	1	0	0.00600934	0.023189306	0.059782505	0
test starbucks 2.jpg	starbucks	binus	binus 5.jpg	binus	32.43788327	6	0	0.00636816	0.022590876	0.064227343	0
test starbucks 3.jpg	starbucks	binus	binus 2.jpg	binus	32.62954643	1	9.173835682	0.006525993	0.023925066	0.057706356	0
test starbucks 4.jpg	starbucks	oxy	oxy 10.jpg	oxy	36.45448468	3	9.428410969	0.003667355	0.024710417	0.080087185	0
test starbucks 5.jpg	starbucks				0	0	0	0.006216764	0.018322706	0.044000626	0
test toyota 1.jpg	toyota				0	0	0	0.00427103	0.020687819	0.054525137	0
test toyota 2.jpg	toyota	starbucks	starbucks 5.jpg	starbucks	10.31690914			0.008476257	0.021845102	0.05160737	0
test toyota 3.jpg	toyota	binus	binus 5.jpg	binus	9.203051159			0.006837845	0.020359516	0.046619654	0
test toyota 4.jpg	toyota	oxy	oxy 10.jpg	oxy	38.03266147	5	0	0.008941889	0.023803949	0.070285797	0

test toyota 5.jpg	toyota	binus 10.jpg	binus	18.92313145	1	17.47626786	0.006083965	0.021314621	0.054041624	0
test yogya 1.jpg	yogya	toyota 3.jpg	toyota	11.23012161			0.004188299	0.02139473	0.052533388	0
test yogya 2.jpg	yogya	binus 5.jpg	binus	9.168996886			0.004994392	0.020671844	0.053801775	0
test yogya 3.jpg	yogya			0			0.004188538	0.020849943	0.050862312	0
test yogya 4.jpg	yogya	binus 5.jpg	binus	10.34038445	5	0	0.004488945	0.022826195	0.06020689	0
test yogya 5.jpg	yogya			0			0.007066727	0.019810677	0.045055151	0