

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**

LEMBAR PENGESAHAN

PENERAPAN DATA MINING PADA MASALAH PENGENALAN POINT OF INTEREST

Gian Martin Dwibudi

NPM: 6181801015

Bandung, 31 Desember 2022

Menyetujui,

Pembimbing Utama

Pembimbing Pendamping

Kristopher David Harjono, M.T.

«pembimbing pendamping/2»

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENERAPAN DATA MINING PADA MASALAH PENGENALAN POINT OF INTEREST

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 31 Desember 2022



Gian Martin Dwibudi
NPM: 6181801015

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

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

«kepada siapa anda mempersembahkan skripsi ini. . . ?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini . . . »

Bandung, Desember 2022

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xxi
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)	9
2.3.1 Pencarian Extrema	9
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)	15
2.4.1 Pencarian Keypoint	16
2.4.2 Penentuan Skala	17
2.4.3 Penentuan Orientasi	17
2.4.4 Pembuatan Deskriptor	18
2.4.5 ORB di OpenCV	20
2.5 BSIS (Best Score Increasing Subsequence)	20
2.5.1 Pairing	21
2.5.2 Verification	21
2.5.3 Scoring	23
2.6 KD-Tree	23
2.7 Locality Sensitive Hashing	24
2.8 Clustering	24
2.8.1 Agglomerative Clustering	24
2.8.2 DBSCAN	25
3 ANALISIS & EKSPERIMEN	29
3.1 Analisis Pengenalan POI	29
3.1.1 Ide Dasar Analisis	29
3.1.2 Tahapan Analisis	29

3.1.3	Implementasi	31
3.1.4	Hasil Analisis	31
3.2	Analisis Fitur Lokal dari Logo dan Bagian yang Konsisten	32
3.3	Dataset yang Digunakan	34
3.3.1	Dataset SMVS	34
3.3.2	Dataset GSV	35
3.4	Analisis Sifat Konsisten pada Fitur Lokal dari Gambar POI	39
3.4.1	Ide Dasar dan Tahapan Analisis	39
3.4.2	Implementasi	40
3.4.3	Hasil	41
3.5	Analisis Sifat Unik pada Fitur Lokal dari Gambar POI	43
3.5.1	Ide Dasar dan Tahapan Analisis	43
3.5.2	Implementasi	44
3.5.3	Hasil	45
3.6	Analisis Tingkat Keunikan dan Kekonsistennan Fitur Lokal pada Gambar POI	47
3.6.1	Ide Dasar Analisis	47
3.6.2	Tahapan Analisis	48
3.6.3	Implementasi	50
3.6.4	Hasil Analisis	51
3.6.5	Visualisasi Nilai Konsistensi dan Keunikan Fitur Lokal	52
3.7	Analisis Penentuan Nilai Threshold untuk Konsistensi dan Keunikan	55
3.7.1	Ide Dasar dan Tahapan Analisis	58
3.7.2	Metode Scoring	58
3.7.3	Hasil Analisis	60
3.8	Analisis Penggunaan Clustering untuk OIR dengan BSIS	62
3.8.1	Data Train dan Test	63
3.8.2	Metode BSIS	64
3.8.3	Tahapan Analisis	65
3.8.4	Tahapan Implementasi	66
3.8.5	Hasil Analisis	67
3.9	Analisis Metode Ekstraksi Fitur Lokal ORB	72
3.9.1	Hasil Pengujian	72
4	PERANCANGAN	75
4.1	Rancangan Alur Program	75
4.2	Rancangan Implementasi Metode Clustering Pembuatan Model	76
4.2.1	Rancangan Struktur Folder	76
4.2.2	Rancangan Proses Clustering	76
4.3	Rancangan Kelas ClusterModel	79
4.4	Rancangan Kelas Util	81
4.5	Rancangan Kelas-kelas Implementasi BSIS	81
4.6	Rancangan Perangkat Lunak BSIS	83
5	PENGUJIAN	85
5.1	Ide Analisis	85
5.2	Penentuan Threshold dan Hasil Analisis	85
5.2.1	Metode SIFT	85
5.2.2	Metode ORB	89
5.3	Analisis Hasil Pengujian	92
5.3.1	GSV 400	92
5.3.2	GSV 600	92

6 KESIMPULAN	95
DAFTAR REFERENSI	97
A KODE PROGRAM	99
B HASIL EKSPERIMEN	113

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.	5
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.	8
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.	10
2.7	Efek nilai σ pada hasil gambar konvolusi.	11
2.8	Operasi DoG pada gambar	11
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>	14
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>	16
2.14	<i>Image Pyramid</i> pada ORB	17
2.15	Ilustrasi penentuan orientasi pada ORB.	18
2.16	Ilustrasi penghitungan <i>Integral Image</i> . <i>Matrix I</i> merupakan <i>matrix</i> awal dan <i>Matrix I_{Σ}</i> merupakan <i>Integral Image</i> dari <i>I</i>	19
2.17	Penghitungan nilai total sebuah daerah dengan menggunakan <i>Integral Image</i>	20
2.18	Tahapan verifikasi pada BSIS.	22
2.19	Contoh pohon struktur KD-Tree. Angka di sebelah kanan menunjukkan elemen vektor yang diperiksa pada level tersebut.	23
2.20	Contoh <i>cluster</i> dengan bentuk non- <i>circular</i> . Objek-objek yang tersebar seperti ini dapat tergabung menjadi <i>cluster</i> dengan menggunakan DBSCAN.	26

3.1	<i>Flowchart</i> tahapan analisis pengenalan POI	30
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>	30
3.3	Pasangan <i>keypoint</i> dari Gambar <i>Q</i> dan Gambar <i>T</i> yang kuat.	32
3.4	Beberapa sudut pengambilan dan waktu pengambilan berbeda dari suatu POI yang sama.	33
3.5	POI yang memiliki logo dengan sudut yang mirip.	34
3.6	Beberapa contoh gambar yang digunakan pada analisis untuk melihat sifat konsisten pada fitur lokal.	40
3.7	Histogram sebaran jumlah gambar unik tiap <i>cluster</i>	42
3.8	Contoh <i>keypoint</i> yang konsisten menurut analisis ini.	43
3.9	Beberapa contoh gambar yang digunakan pada analisis untuk melihat sifat unik pada gambar POI.	44
3.10	Histogram sebaran jumlah kelas gambar yang unik tiap <i>cluster</i>	46
3.11	Contoh <i>keypoint</i> yang unik menurut analisis ini.	47
3.12	<i>Flowchart</i> tahapan analisis <i>clustering</i> untuk mencari fitur lokal yang konsisten dan unik.	48
3.13	Contoh beberapa gambar yang digunakan pada analisis ini.	49
3.14	Histogram sebaran nilai keunikan.	52
3.15	Histogram sebaran nilai keunikan.	52
3.16	Beberapa contoh gambar POI beserta <i>keypoint</i> yang fitur lokalnya sudah diberi warna sesuai dengan nilai konsistensi dan keunikannya.	54
3.17	Contoh objek yang sifatnya konsisten dan unik dalam POI.	56
3.18	Contoh objek yang sifatnya konsisten dan unik dalam POI.	57
3.19	Contoh tampilan aplikasi LabelImg.	59
3.20	Skor ketepatan untuk tiap <i>threshold</i> pada nilai konsistensi.	61
3.21	Skor ketepatan untuk tiap <i>threshold</i> pada nilai keunikan.	62
3.22	Contoh gambar pada <i>dataset</i> Book Covers. Keempat gambar tersebut berada dalam satu kelas yang sama.	63
3.23	Contoh gambar referensi dari <i>dataset</i> Book Covers.	64
3.24	Contoh gambar referensi dari <i>dataset</i> Book Covers.	64
3.25	Modifikasi pada metode BSIS yang dilakukan pada analisis ini.	65
3.26	Tahapan yang dilakukan dalam analisis untuk menguji penggunaan <i>clustering</i> pada BSIS.	65
3.27	Sebaran nilai keunikan (<i>consistency</i>) untuk Dataset 400.	67
3.28	Sebaran nilai keunikan (<i>uniqueness</i>) untuk Dataset 400.	67
3.29	Perbandingan sebaran waktu Dataset 400.	68
3.30	Sebaran nilai keunikan (<i>uniqueness</i>) untuk Dataset 600.	69
3.31	Sebaran nilai keunikan (<i>consistency</i>) untuk Dataset 600.	70
3.32	Perbandingan sebaran waktu Dataset 600.	70
4.1	Rancangan alur program pada penelitian ini.	75
4.2	Rancangan struktur <i>folder</i> untuk pembuatan model.	76
4.3	Tahapan proses <i>clustering</i> hingga didapat nilai keunikan dan konsistensi.	77
4.4	Diagram kelas <i>ClusterModel</i>	80
4.5	Diagram kelas <i>Util</i>	81
4.6	Diagram <i>Package</i> <i>bsis</i>	82
4.7	Rancangan tampilan perangkat lunak untuk BSIS.	84
5.1	Histogram sebaran nilai keunikan pada <i>Dataset</i> GSV 400.	86
5.2	Histogram sebaran nilai keunikan <i>Dataset</i> GSV 400.	86

5.3	Histogram sebaran nilai keunikan pada <i>Dataset GSV 600</i>	87
5.4	Histogram sebaran nilai keunikan <i>Dataset GSV 600</i>	88
5.5	Histogram sebaran nilai keunikan pada <i>Dataset GSV 400</i>	89
5.6	Histogram sebaran nilai keunikan <i>Dataset GSV 400</i>	90
5.7	Histogram sebaran nilai keunikan pada <i>Dataset GSV 600</i>	91
5.8	Histogram sebaran nilai keunikan <i>Dataset GSV 600</i>	91
B.1	Hasil 1	113
B.2	Hasil 2	113
B.3	Hasil 3	113
B.4	Hasil 4	113

BAB 1

PENDAHULUAN

1.1 Latar Belakang

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

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



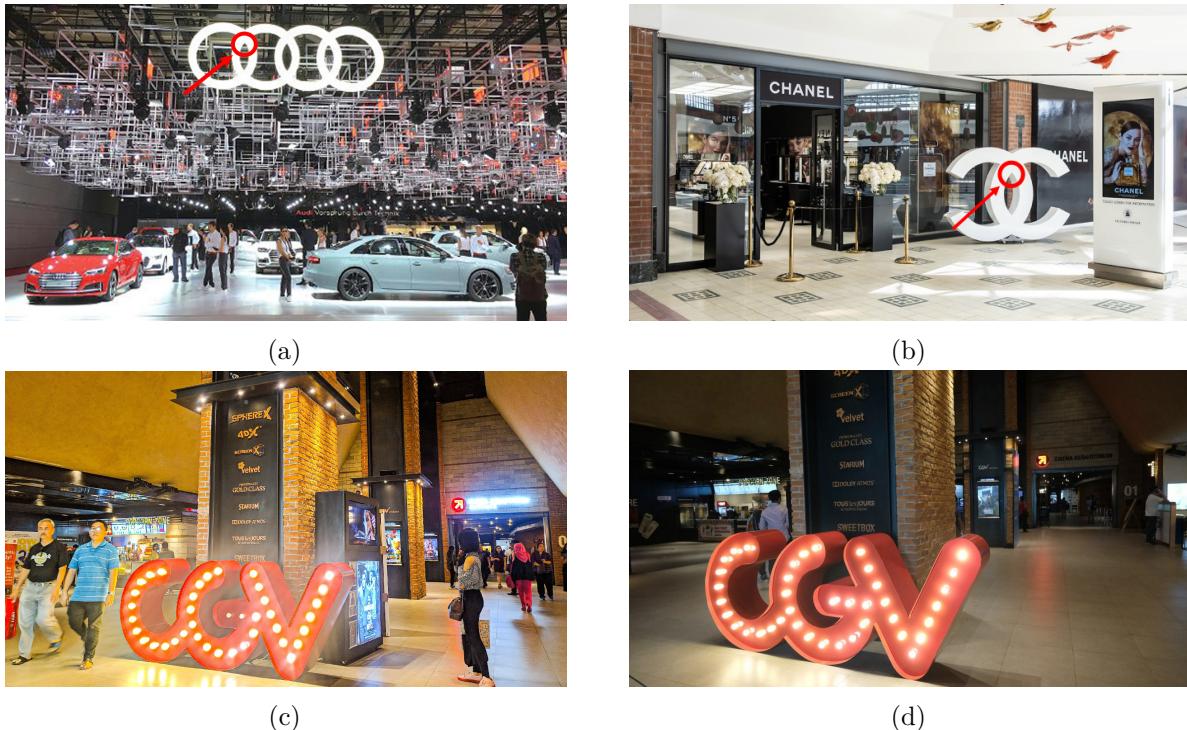
Gambar 1.1: Contoh POI yang memiliki logo unik. POI seperti ini dapat dikenali dengan melakukan identifikasi pada logo tersebut.

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

Fitur lokal merupakan fitur yang mendeskripsikan sebuah daerah penting (*keypoint*) pada gambar. Salah satu cara mendapatkan *Keypoint* adalah dengan mencari sudut-sudut atau perpotongan garis (*corner*) yang terdapat pada gambar. *Keypoint-keypoint* yang terdeteksi pada gambar akan memiliki sebuah vektor untuk mendeskripsikan daerah di sekitar *keypoint* tersebut yang disebut sebagai vektor deskriptor. Proses pencarian fitur lokal pada penelitian ini akan dilakukan dengan menggunakan metode *Scale Invariant Feature Transform* (SIFT) dan *Oriented FAST and Rotated BRIEF* (ORB). Metode SIFT dan ORB akan menghasilkan vektor deskriptor untuk tiap fitur lokal yang terdeteksi, vektor deskriptor ini dapat digunakan untuk mengidentifikasi fitur lokal.

Salah satu masalah yang ada pada pengenalan POI adalah pada sebuah gambar POI tidak semua fitur lokal yang dideteksi bersifat unik terhadap POI tersebut. Ada fitur lokal yang juga

dimiliki oleh POI lain atau fitur lokal yang berasal dari objek di latar belakang yang sifatnya tidak konsisten. Masalah ini akan mempersulit pada proses OIR untuk mengidentifikasi POI yang tepat. Gambar 1.2 menunjukkan masalah-masalah ini, gambar 1.2a dan 1.2b menunjukkan fitur lokal yang mirip dari dua POI yang berbeda, sedangkan gambar 1.2c dan 1.2d menunjukkan objek-objek latar belakang yang tidak konsisten pada POI. Penelitian ini akan melakukan analisis untuk menemukan dan memisahkan fitur-fitur lokal tersebut agar tidak diproses dalam pembuatan model POI.



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.

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

1.2 Rumusan Masalah

Skripsi ini memiliki rumusan masalah sebagai berikut:

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

1.3 Tujuan

Skripsi ini memiliki tujuan sebagai berikut:

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

1.4 Batasan Masalah

Berikut batasan-batasan masalah dari skripsi ini:

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

1.5 Metodologi

Skripsi ini akan memiliki metodologi sebagai berikut:

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

1.6 Sistematika Pembahasan

Sistematika pembahasan yang digunakan pada penelitian ini adalah sebagai berikut:

1. Bab 1 Pendahuluan
Bab ini berisi tentang hal-hal yang menggambarkan skripsi ini secara garis besar. Hal yang dibahas merupakan latar belakang masalah, rumusan masalah, tujuan penelitian, batasan masalah, dan metodologi penelitian.
2. Bab 2 Landasan Teori
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* dan DBSCAN, 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

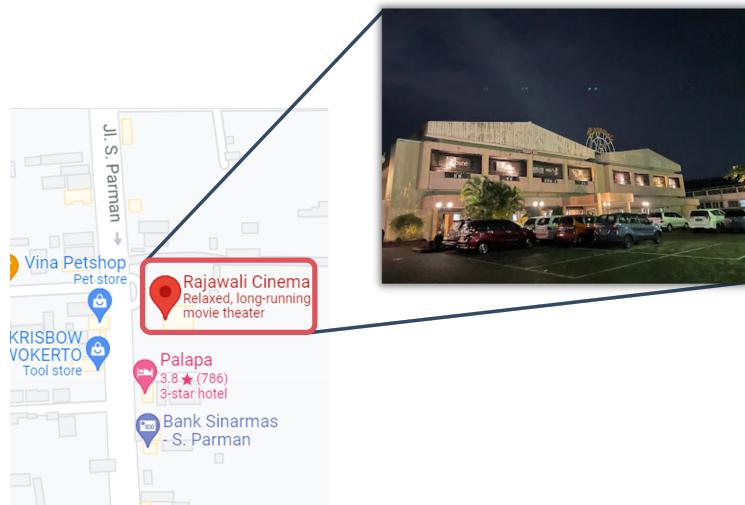
Bab ini berisi kesimpulan yang didapatkan dari hasil analisis serta keseluruhan implementasi dan pengujian yang dilakukan pada penelitian ini.

BAB 2

LANDASAN TEORI

2.1 Point of Interest

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



Gambar 2.1: Salah satu contoh POI.

Beberapa POI tertentu dapat memiliki logo yang sifatnya unik. POI tersebut dapat dikenali dengan hanya melihat logonya saja. Contoh bisa dilihat pada Gambar 2.2. POI dengan logo unik ini dapat dikenali oleh komputer salah satunya dengan menggunakan teknik *Object Instance Recognition* (OIR).



Gambar 2.2: Contoh POI dengan logo unik.

2.2 Object Instance Recognition

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

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

- Cahaya

Perubahan pada tingkat pencahayaan pada gambar yang mengakibatkan perubahan nilai *pixel-pixel* pada gambar.

- Skala

Jarak diambilnya gambar yang berisi objek. Perbedaan ukuran objek pada gambar akan menyebabkan sudut-sudut pada objek menjadi berbeda.

- Rotasi

Orientasi atau arah pengambilan gambar yang berbeda akan mengakibatkan objek pada gambar jadi terlihat berbeda. Sudut-sudut akan menjadi berbeda karena arah hadapnya berbeda.

- Latar Belakang

Objek-objek lain di sekitar objek yang ingin diidentifikasi akan berpotensi mempersulit pemrosesan. Objek-objek tersebut dapat menghasilkan fitur-fitur lokal yang tidak relevan terhadap objek yang ingin diidentifikasi.

- Bagian Objek Tertutup

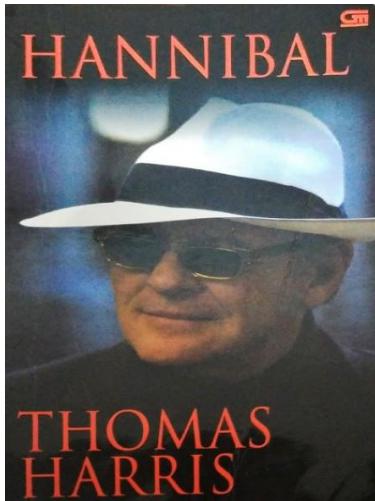
Adanya objek lain yang menutupi sebagian dari objek yang ingin diidentifikasi akan berpotensi menyebabkan beberapa fitur lokal dari objek tidak terdeteksi.

- Sudut Pandang

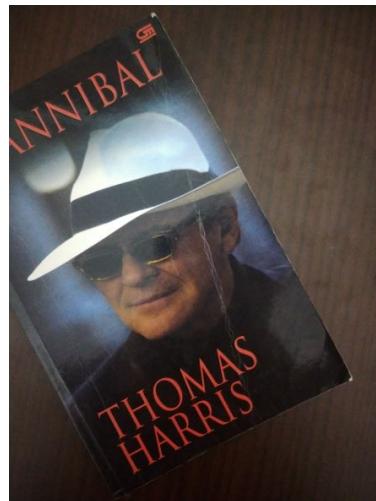
Sudut pengambilan gambar yang berbeda akan memengaruhi pemrosesan. Fitur-fitur lokal dari objek yang ingin diidentifikasi akan menjadi berbeda.

- Translasi

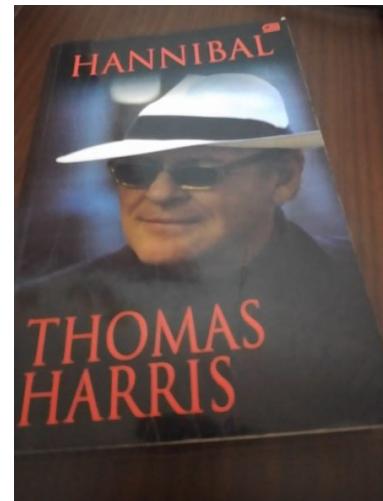
Posisi objek yang ingin diidentifikasi dalam gambar akan memengaruhi pemrosesan. Pencarian pasangan fitur lokal tidak dapat dengan hanya menggunakan posisi di mana fitur lokal tersebut ditemukan pada gambar.



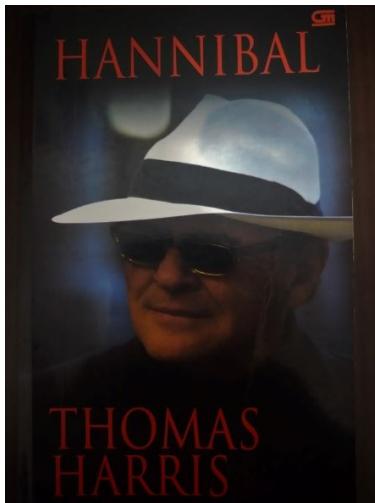
(a) Gambar objek yang ideal, dari sudut tegak lurus



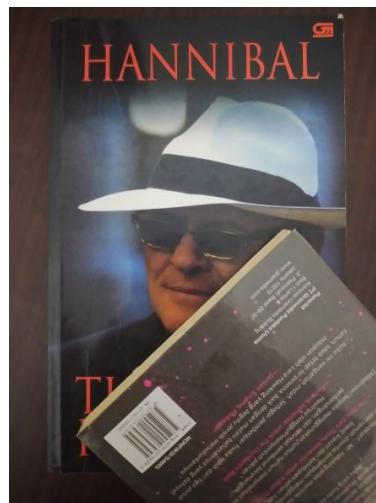
(b) Objek pada gambar mengalami translasi dan rotasi



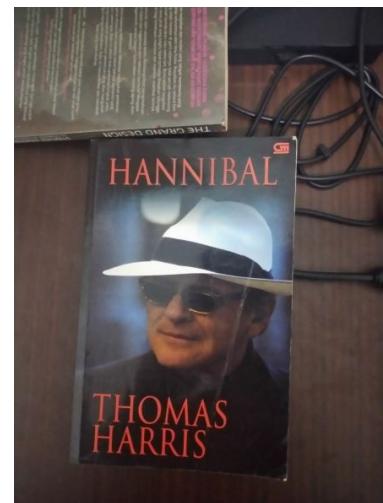
(c) Gambar diambil dari sudut yang berbeda



(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

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

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

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

Proses pencarian pasangan fitur lokal dari gambar masukkan dan *dataset* dilakukan dengan menggunakan vektor deskriptor dari fitur lokal. Perubahan-perubahan pada gambar walaupun sedikit akan menyebabkan perubahan nilai pada vektor deskriptor. Vektor deskriptor dari fitur

lokal pada gambar masukkan hampir tidak akan persis sama dengan vektor deskriptor dari gambar yang ada di *dataset*. Oleh karena itu pencarian pasangan fitur lokal dilakukan dengan mencari pasangan fitur lokal yang memiliki nilai kemiripan paling tinggi.

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

1. Ekstraksi Fitur

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

2. Pairing

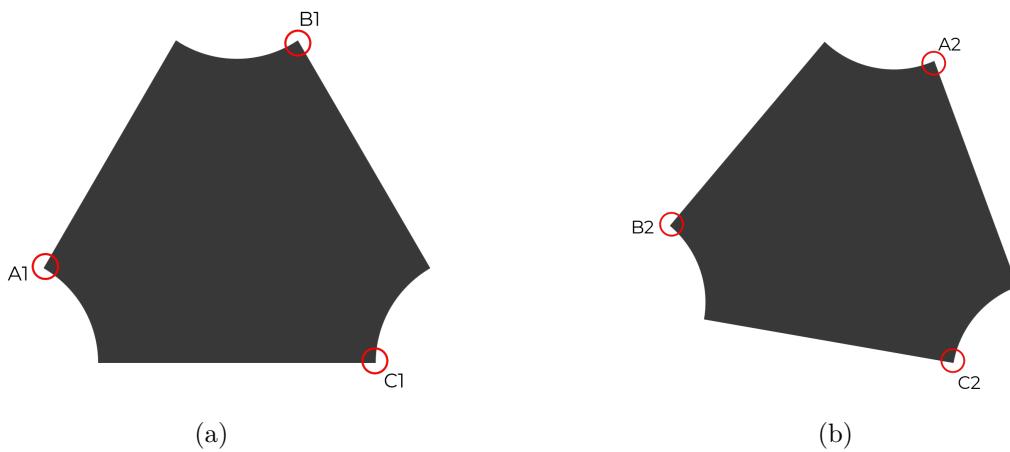
Untuk setiap fitur lokal yang terdeteksi pada gambar masukkan akan dicari beberapa fitur lokal dari gambar pada *dataset* yang paling mirip. Kemiripan fitur lokal ditentukan dengan menghitung jarak (sesuai dengan metrik yang digunakan) antara vektor deskriptor dari kedua fitur lokal tersebut.

3. Verification

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

4. Scoring

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



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

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

Pada orientasi seperti di gambar, pasangan A1 dengan A2 dan B1 dengan B2 merupakan pasangan yang tidak konsisten. Hal tersebut dikarenakan posisi A1 yang berada di sebelah kiri B1 tetapi pasangan dari A1, yaitu A2 berada di sebelah kiri pasangan dari B2, pasangan dari B1. Sedangkan pasangan C1 dengan C2 merupakan pasangan yang konsisten, karena baik C1 maupun

C2 memiliki posisi relatif yang sama terhadap pasangan fitur lokal lain.

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

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

Cara lain yang dapat dilakukan pada tahap verifikasi adalah dengan melakukan *Lowe's Ratio Test*. Cara ini tidak mencari pasangan yang konsisten secara geometris melainkan hanya menyaring pasangan yang kuat. *Lowe's Ratio Test* mencari pasangan yang kuat dengan membandingkan 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:

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

Variabel n pada persamaan di atas merupakan sebuah nilai konstan yang nilainya lebih dari 1. Persamaan tersebut bertujuan untuk mencari pasangan yang benar-benar mirip dan nilai kemiripannya berbeda jauh dengan pasangan lainnya. Nilai n dapat diatur untuk menyatakan seberapa perlu seberapa jauh jarak pasangan termirip dengan kedua termirip.

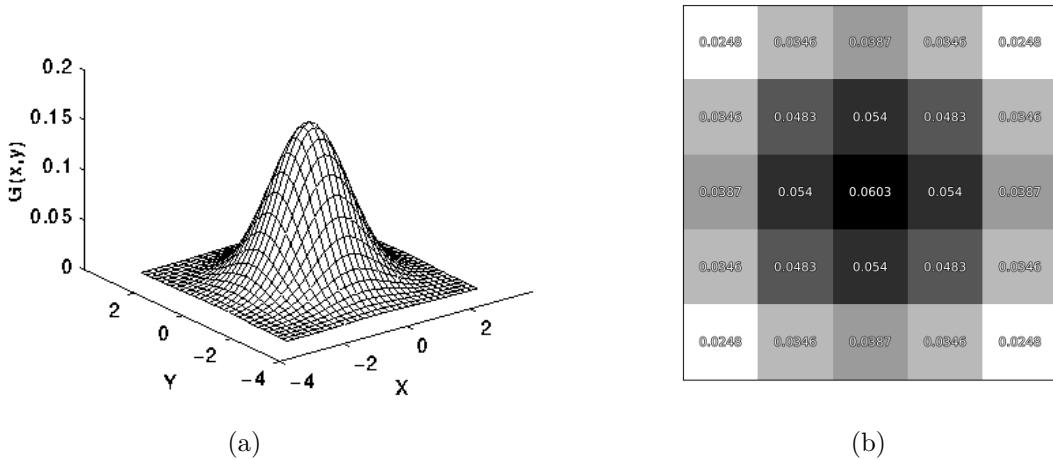
2.3 SIFT (Scale Invariant Feature Transform)

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

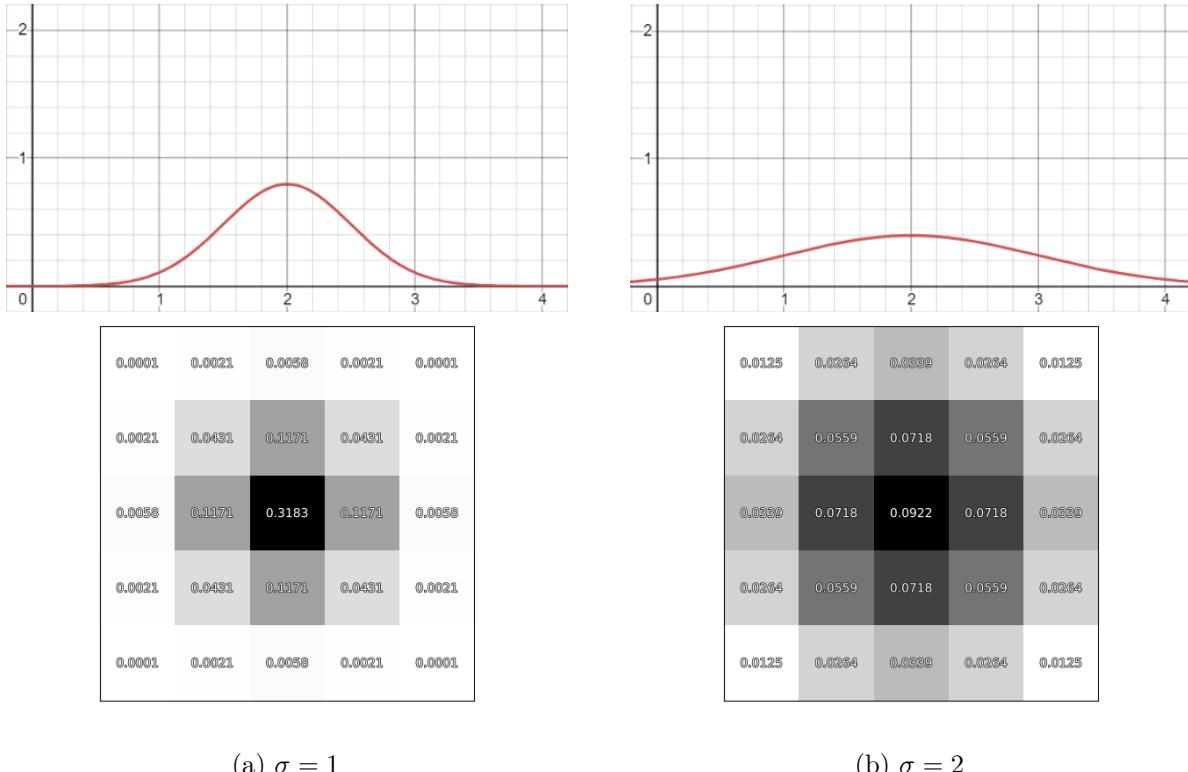
2.3.1 Pencarian Extrema

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

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

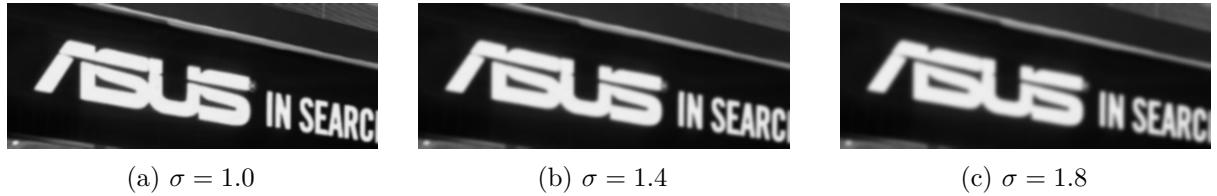
Gambar 2.5: Kurva Gaussian dan bentuk representasi *matrix*-nya.

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

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

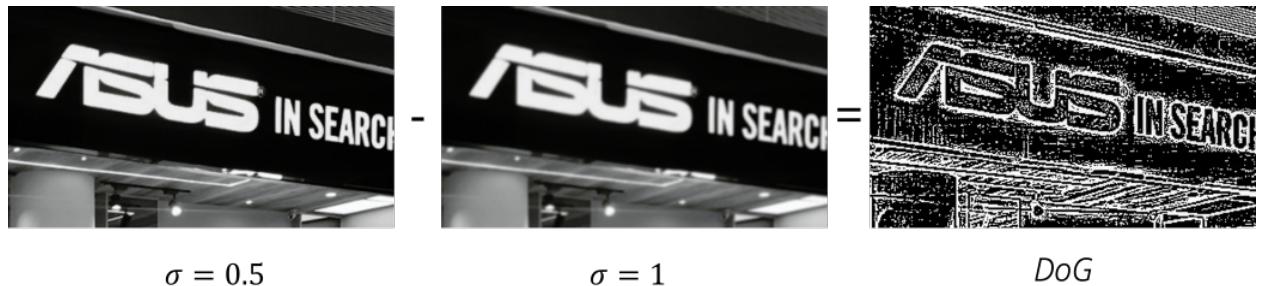
Matrix Konvolusi Gaussian ketika diaplikasikan pada gambar akan menyebabkan perubahan nilai tiap *pixel* pada gambar. Nilai dari setiap *pixel* akan menjadi mirip dengan *pixel* tetangga di dekatnya. Perubahan nilai *pixel* akan paling berpengaruh pada daerah dengan perubahan nilai *pixel* yang tinggi. 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.

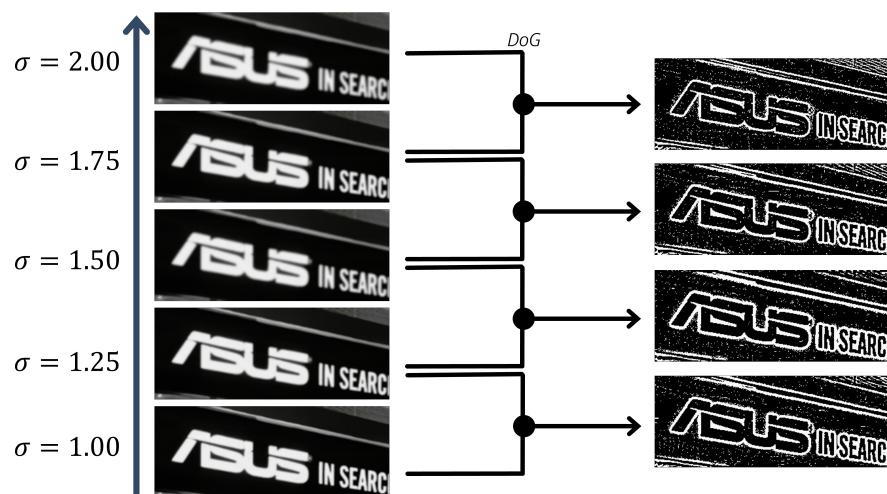
Perubahan nilai σ pada *matrix* konvolusi serta efeknya pada gambar akan dimanfaatkan untuk menghitung *Difference of Gaussian* (DoG). DoG merupakan hasil turunan kedua Gaussian pada gambar. Gambar DoG dapat diperoleh dengan menghitung perbedaan nilai tiap *pixel* dari dua gambar yang telah dikonvolusi 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.



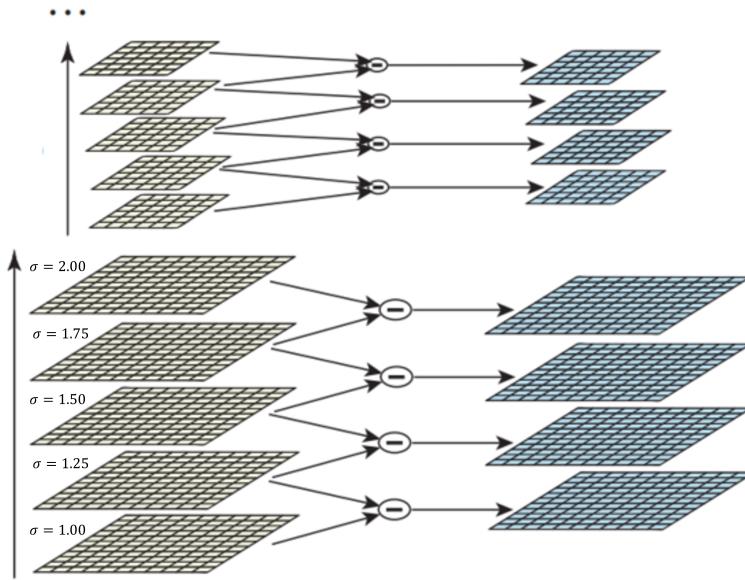
Gambar 2.9: Penggunaan DoG pada SIFT

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*.

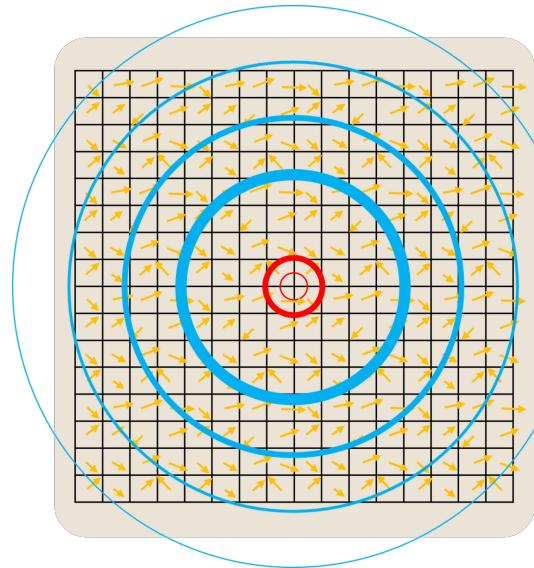
Penentuan orientasi yang dominan dihitung dengan menggunakan *magnitude* ($m(x, y)$) dan orientasi ($\theta(x, y)$) dari *pixel-pixel* dengan menggunakan rumus pada Persamaan 2.2 dan Persamaan 2.3. $L(x, y)$ pada kedua persamaan tersebut merupakan gambar hasil konvolusi.

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

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

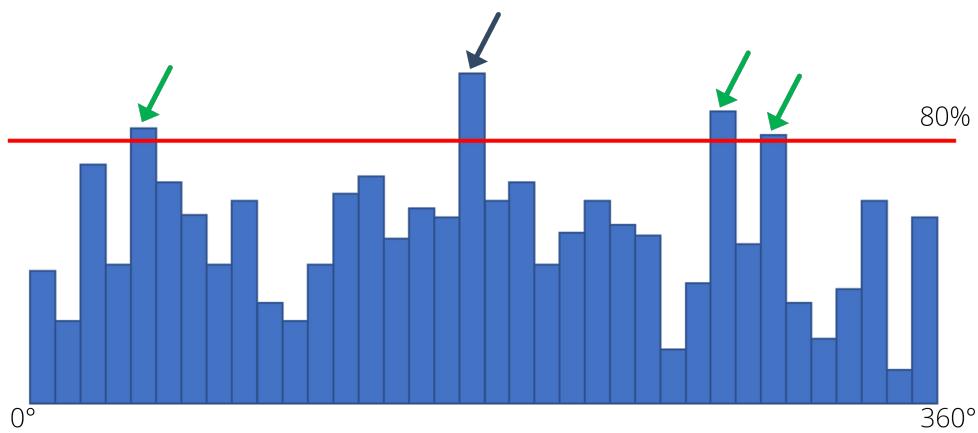
Setiap *pixel* akan dihitung orientasi dan *magnitude*-nya. *Magnitude* akan digunakan sebagai bobot dari *pixel* tersebut. Selain *magnitude*, bobot sebuah *pixel* juga dipengaruhi oleh *Gaussian Weighting*, yang membuat *pixel* yang posisinya dekat dengan titik pusat (titik *keypoint*) akan memiliki bobot

yang lebih tinggi dibanding yang lokasinya jauh dari titik pusat. Ilustrasi pada Gambar 2.11 menunjukkan bagaimana pembobotan dihitung, *pixel-pixel* yang berada dekat dengan *keypoint* (titik tengah) akan diberi bobot yang lebih besar—ditandai dengan lingkaran yang tebal. Sedangkan 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

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

Dari histogram tersebut puncak nilai bin tertinggi akan digunakan sebagai orientasi dari *keypoint*. Untuk puncak-puncak lain yang berada dalam rentang 80% dari puncak tertinggi akan digunakan

untuk membuat *keypoint* baru pada lokasi yang sama dengan orientasi yang berbeda sesuai dengan nilai orientasi pada bin tersebut.

2.3.4 Pembuatan Deskriptor

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

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

2.3.5 SIFT di OpenCV

Library OpenCV di Python memiliki modul implementasi SIFT. Modul dapat digunakan untuk melakukan ekstraksi fitur lokal dari gambar dan menghasilkan vektor deskriptor untuk tiap fitur lokal. Untuk melakukan deteksi fitur lokal, pertama perlu untuk dibuat objek SIFT dengan menggunakan fungsi `SIFT_create`. Beberapa parameter dari fungsi `SIFT_create` yang relevan 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.

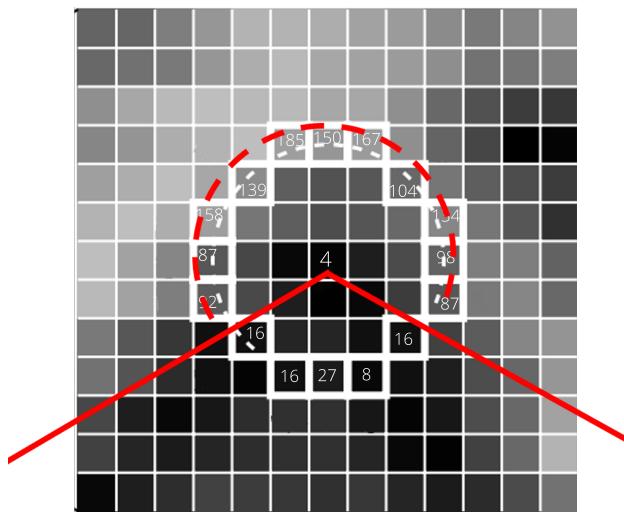
2.4 ORB (Oriented FAST and Rotated BRIEF)

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 invarian terhadap rotasi, translasi dan invarian 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*.

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

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

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

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

Teknik *Harris Corner Measure* akan menghasilkan sebuah skor M yang menunjukkan nilai

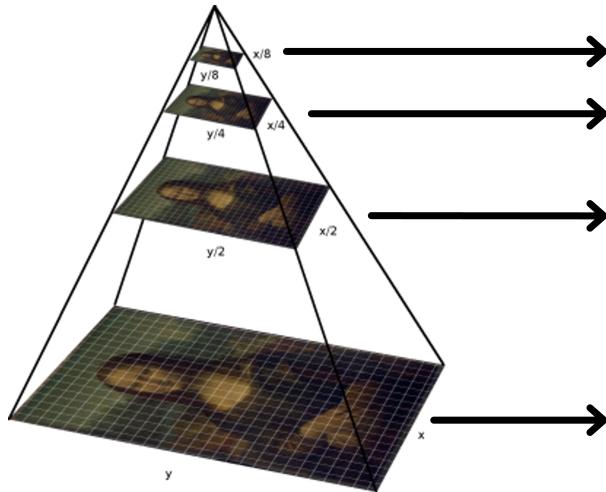
seberapa daerah yang diperiksa merupakan *corner*. ORB akan mengambil N *keypoint* dengan nilai M tertinggi untuk ditentukan sebagai *keypoint*.

2.4.2 Penentuan Skala

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

Untuk mencapai sifat ini ORB menggunakan metode *Image Pyramid*. ORB menggunakan *Image Pyramid* dengan cara memperkecil ukuran gambar beberapa kali dan untuk setiap ukuran gambar dilakukan deteksi untuk *keypoint*. Dengan menggunakan cara ini *keypoint* tidak akan benar-benar invariant terhadap perubahan skala, *keypoint* hanya invariant terhadap beberapa skala gambar yang digunakan pada *pyramid*.

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



Gambar 2.14: *Image Pyramid* pada ORB

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

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.

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

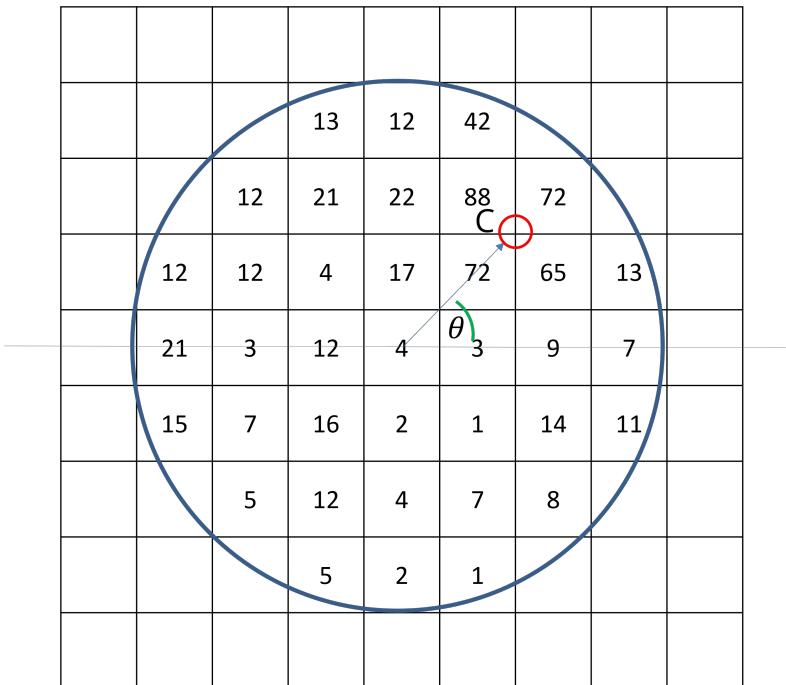
Centroid ditentukan dengan menghitung *moment* pada gambar yang didefinisikan pada Persamaan 2.5.

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

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

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

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



Gambar 2.15: Ilustrasi penentuan orientasi pada ORB.

2.4.4 Pembuatan Deskriptor

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

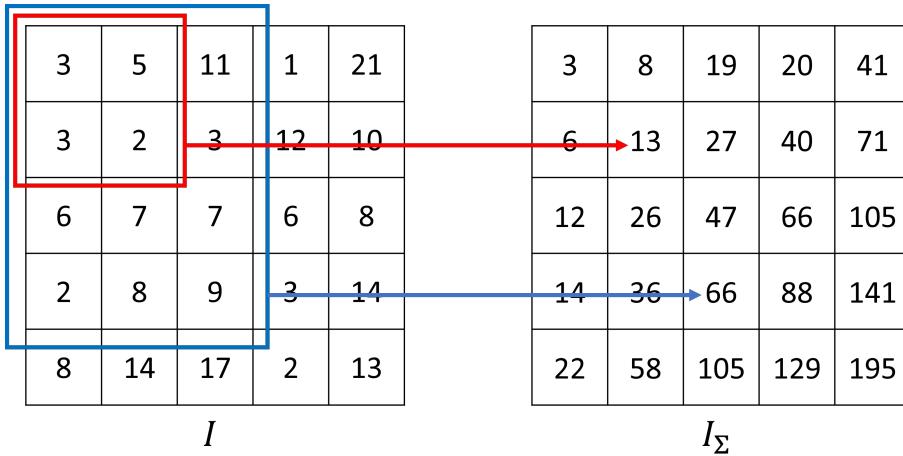
Agar deskriptor bersifat invariant terhadap rotasi gambar terlebih dahulu dirotasi sesuai dengan orientasi yang sudah didapat sebelumnya. Dari gambar yang telah dirotasi diambil daerah berukuran 31×31 untuk dilakukan *binary test*. *Binary test* adalah fungsi yang membandingkan nilai intensitas antar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

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

Fungsi tersebut akan menghasilkan nilai 1 atau 0 bergantung pada nilai intensitas dari dua titik 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.



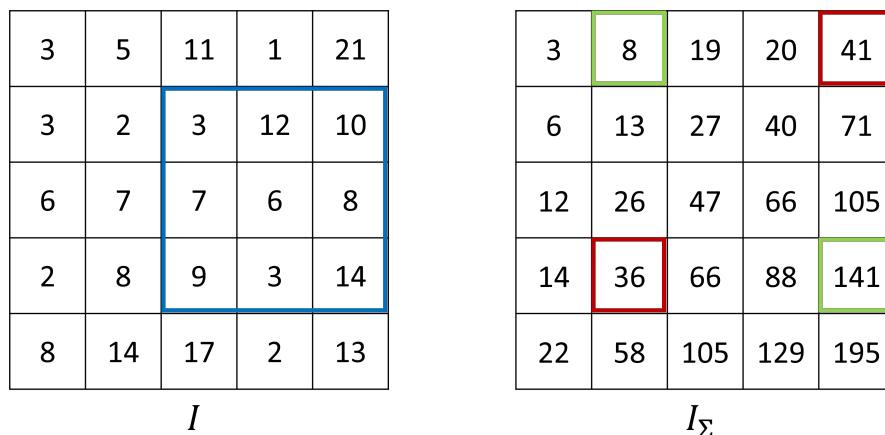
Gambar 2.16: Ilustrasi penghitungan *Integral Image*. *Matrix I* merupakan *matrix* awal dan *Matrix* I_{Σ} merupakan *Integral Image* dari *I*.

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 $141 - 41 - 36 + 8 = 72$.



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

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

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

2.4.5 ORB di OpenCV

Library OpenCV di Python memiliki implementasi untuk metode ORB. Implementasi ORB ini memiliki kegunaan dan cara penggunaan yang mirip dengan implementasi SIFT. Untuk melakukan deteksi fitur lokal perlu untuk terlebih dahulu dibuat objek ORB dengan fungsi `ORB_create`. Beberapa parameter fungsi `ORB_create` yang relevan terhadap skripsi ini adalah:

- `nfeatures`: jumlah maksimum fitur lokal yang dihasilkan.
- `scaleFactor`: rasio pengurangan resolusi pada setiap level *pyramid*.
- `nlevels`: jumlah tingkatan (*level*) pada *pyramid*.
- `firstLevel`: tingkat dalam *pyramid* sebagai tempat gambar asli.
- `patchSize`: ukuran daerah yang digunakan untuk membuat deskriptor.

Objek ORB tersebut lalu dapat digunakan untuk mendeteksi fitur lokal dalam gambar 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 tersebut menghasilkan *tuple* berisi objek *keypoint* dan array vektor deskriptor untuk tiap objek *keypoint*. Setiap vektor merupakan vektor berisi bilangan biner berjumlah 256 elemen. Objek *keypoint* 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`: lapisan *pyramid* di mana *keypoint* tersebut didapatkan.

2.5 BSIS (Best Score Increasing Subsequence)

Pada tahapan OIR (lihat 2.2) sebuah pasangan fitur lokal perlu untuk memiliki sifat yang mirip (dilihat dari vektor deskriptornya) dan juga konsisten secara geometris. Pasangan yang konsisten

secara geometris adalah pasangan yang memiliki posisi spasial yang konsisten terhadap pasangan lain di sekitarnya (contoh dapat dilihat pada Gambar 2.4 di 2.2). BSIS [3] adalah salah satu metode yang dapat digunakan untuk menentukan apakah pasangan fitur lokal bersifat konsisten secara geometris.

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

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

2.5.1 Pairing

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

Untuk setiap fitur lokal, BSIS hanya akan mengambil beberapa pasangan yang benar-benar mirip atau paling berpeluang menjadi pasangan yang benar. Pasangan yang benar-benar mirip pada BSIS adalah pasangan yang nilai kemiripannya tinggi dan memiliki perbedaan yang cukup jauh dengan nilai kemiripan pasangan lain.

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

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

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

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

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

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

2.5.2 Verification

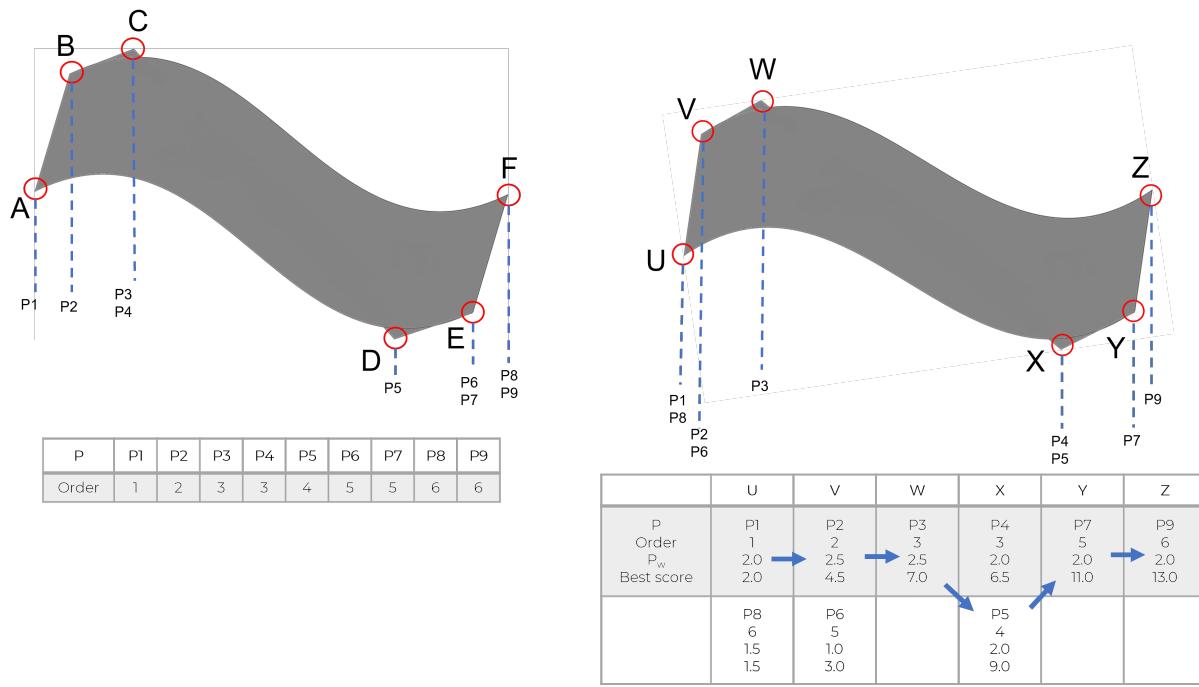
Pada tahap *Pairing* telah didapat beberapa pasangan antara fitur lokal gambar *query* dan gambar *training* yang paling berpotensi merupakan pasangan yang benar. Dari tahap sebelumnya setiap fitur lokal pada gambar *query* dapat dipasangkan dengan lebih dari 1 fitur lokal pada gambar *training*. Pasangan-pasangan tersebut akan diperiksa pada tahap ini untuk mencari pasangan mana

saja yang konsisten secara geometris. Setiap fitur lokal baik dari gambar *query* maupun *training* hanya akan berada dalam satu pasangan setelah dilakukan verifikasi.

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

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

1. Ambil semua fitur lokal dari gambar *query* dan proyeksikan pada sumbu x . Setiap fitur lokal akan diberi sebuah *order* sesuai dengan urutan kemunculannya.
2. Ambil semua fitur lokal dari gambar *training* dan proyeksikan pada sumbu x . Urutkan fitur lokal tersebut sesuai dengan urutan kemunculannya.
3. Buat tabel untuk fitur lokal pada gambar *query* dengan setiap kolom merupakan fitur lokal yang diurutkan berdasarkan kemunculannya pada sumbu x ;
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 2.18: Tahapan verifikasi pada BSIS.

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.

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 .

Tahapan di atas dilakukan beberapa kali pada orientasi fitur lokal gambar *training* yang berbeda untuk mencari orientasi gambar yang memberikan total skor pasangan paling tinggi.

2.5.3 Scoring

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

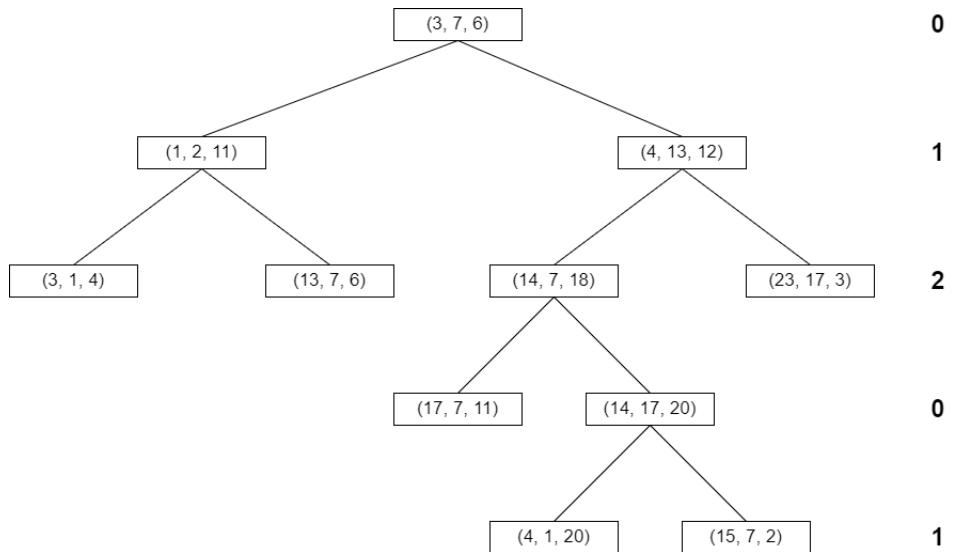
2.6 KD-Tree

KD-Tree [5] merupakan sebuah struktur data yang dapat digunakan untuk mencari vektor yang paling mirip dengan waktu proses yang relatif lebih singkat. Struktur KD-Tree digunakan untuk 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.

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

Gambar 2.19 menunjukkan contoh pohon KD-Tree dengan jumlah 3 elemen.



Gambar 2.19: Contoh pohon struktur KD-Tree. Angka di sebelah kanan menunjukkan elemen vektor yang diperiksa pada level tersebut.

KD-Tree membuat pohon dari sekumpulan vektor di *dataset*. Vektor pertama yang masuk akan menjadi *root* dari pohon. Vektor selanjutnya akan dibandingkan nilai elemen pertamanya dengan

elemen pertama dari *root*, jika nilainya lebih kecil maka vektor tersebut akan menjadi *child* sebelah kiri dan menjadi *child* sebelah kanan jika sebaliknya. Proses berlanjut untuk vektor selanjutnya, elemen di vektor yang masuk akan dibandingkan dengan vektor pada *node* pohon sesuai dengan *level* pohon saat itu.

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

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

2.7 Locality Sensitive Hashing

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

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

2.8 Clustering

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

Beberapa contoh metode *clustering* yang ada adalah *Agglomerative Clustering* dan DBSCAN. Kedua teknik tersebut menggunakan metode dasar yang berbeda dan akan menghasilkan *cluster* dengan ciri yang berbeda juga. Kedua metode tersebut dijelaskan pada dua subbab berikut. Penjelasan berdasarkan pada [6]

2.8.1 Agglomerative Clustering

Teknik *Agglomerative Clustering* adalah salah satu teknik *clustering* yang berbasis hierarki (*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

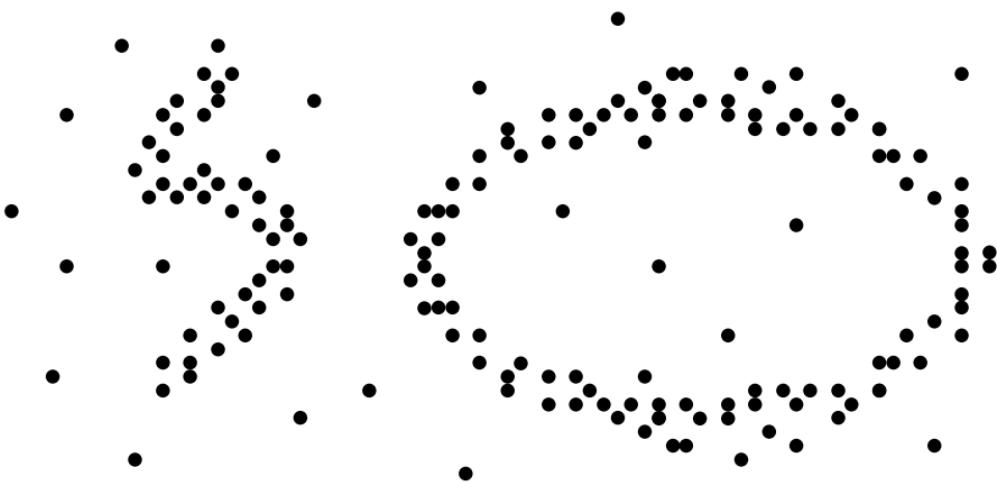
Output: set berisi *cluster-cluster*

- 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**
-

2.8.2 DBSCAN

DBSCAN atau *Density-Based Spatial Clustering of Applications with Noise* adalah salah satu metode *clustering* yang berbasis kepadatan (*density based*). Pada DBSCAN sebuah *cluster* merupakan sebuah daerah *dense* yang dipisahkan oleh daerah *sparse*. Daerah *dense* adalah sebuah kumpulan yang terdiri dari beberapa objek, di mana objek-objek tersebut memiliki ciri yang mirip. Sedangkan daerah *sparse* adalah daerah di mana hanya terdapat sedikit objek dengan sifat yang saling mirip.

Cluster yang dihasilkan oleh DBSCAN tidak selalu berbentuk *circular* (contoh seperti pada Gambar 2.20) dan objek-objek yang berada dalam satu *cluster* tidak selalu merupakan data yang mirip. Selama sebuah daerah *dense* memiliki daerah *dense* lain di dekatnya, kedua daerah tersebut akan terus bergabung menjadi satu daerah *dense* yang lebih besar. Untuk sebuah daerah *dense* D dapat bergabung ke daerah lain yang terdiri dari lebih dari satu daerah *dense* C , daerah *dense* D hanya perlu untuk dekat dengan salah satu daerah *dense* dari C . Karena bentuknya yang tidak *circular* dan tidak tentu miripnya objek dalam satu *cluster*, maka *cluster* tidak dapat direpresentasikan dengan sebuah titik tengah atau *centroid*.



Gambar 2.20: Contoh *cluster* dengan bentuk non-*circular*. Objek-objek yang tersebar seperti ini dapat tergabung menjadi *cluster* dengan menggunakan DBSCAN.

Penyusunan *cluster* pada DBSCAN dimulai dengan mencari objek yang merupakan *core object*. *Core object* merupakan objek yang memiliki setidaknya $MinPts$ objek lain pada radius ϵ yang berpusat pada objek tersebut. $MinPts$ dan ϵ adalah parameter yang ditentukan secara manual. Nilai $MinPts$ dan ϵ akan memengaruhi hasil *cluster* yang dihasilkan.

Setiap objek yang merupakan *core object* beserta anggotanya (objek lain pada radius ϵ) akan menjadi satu *cluster*. Jika dalam radius ϵ objek tersebut terdapat objek lain yang juga merupakan *core object*, maka akan digabungkan menjadi satu *cluster*. *Core object* yang saling berdekatan ini akan terus digabungkan menjadi *cluster* yang besar.

Proses pencarian *cluster* pada DBSCAN dimulai dengan pertama menandai semua objek pada *dataset*, D dengan *unvisited*. Dari D diambil sebuah objek, p secara acak dan ditandai sebagai *visited*. Jika p bukan merupakan *core object* maka objek tersebut merupakan *noise*. Sebaliknya jika p adalah *core object* maka p akan dimasukkan ke *cluster C* dan semua objek pada daerah ϵ -nya dimasukkan ke dalam set N . Objek-objek pada N akan diiterasi dan dimasukkan ke *cluster C*. Jika ditemukan objek yang merupakan *core object* maka semua objek lain pada daerah ϵ objek tersebut akan dimasukkan juga ke N . Proses berlanjut sampai tidak ada lagi objek di N . Objek-objek yang sudah diperiksa tersebut akan ditandai sebagai *visited*. Setelah itu akan diambil lagi sebuah objek secara acak yang masih bertanda *unvisited* dan proses diulang untuk membentuk *cluster* baru.

Proses didefinisikan pada *Pseudocode 2* berikut:

Pseudocode 2: DBSCAN**Input:**

- D : dataset berisi n buah objek
- ϵ : parameter radius untuk menghitung daerah
- $MinPts$: batas kepadatan suatu daerah.

Output: set berisi *cluster-cluster*

```
1: tandai semua objek sebagai unvisited
2: while masih terdapat objek yang unvisited do
3:   pilih objek  $p$  secara acak
4:   tandai  $p$  sebagai visited
5:   if dalam radius  $\epsilon$  dari  $p$  terdapat setidaknya  $MinPts$  objek then
6:     buat cluster baru  $C$ , masukkan  $p$  ke dalam  $C$ 
7:     buat variabel  $N$ , berisi semua objek dalam radius  $\epsilon$  dari  $p$ 
8:     for semua objek  $p'$  di  $N$  do
9:       if objek  $p'$  unvisited then
10:        tandai  $p'$  sebagai visited
11:        if dalam radius  $\epsilon$  dari  $p'$  terdapat setidaknya  $MinPts$  objek then
12:          masukkan semua objek tersebut ke dalam  $N$ 
13:        end if
14:        if  $p'$  bukan anggota dari cluster manapun then
15:          masukkan  $p'$  ke dalam  $C$ 
16:        end if
17:      end if
18:    end for
19:    output  $C$ 
20:  else
21:    tandai  $p$  sebagai noise
22:  end if
23: end while
```

BAB 3

ANALISIS & EKSPERIMENT

3.1 Analisis Pengenalan POI

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

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

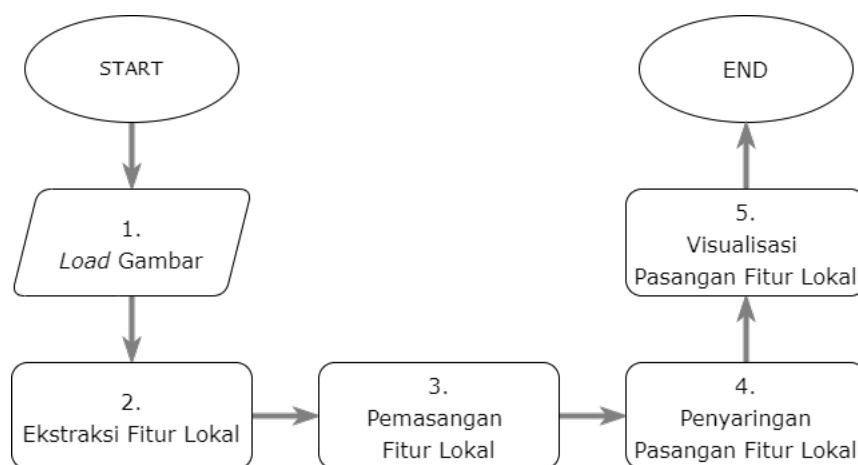
3.1.1 Ide Dasar Analisis

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

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

3.1.2 Tahapan Analisis

Untuk melakukan implementasi dari ide di atas dilakukan langkah-langkah analisis seperti yang 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 kemiripannya lebih besar dari n kali nilai kemiripan pasangan kedua termirip.

5. *Visualisasi Pasangan Fitur Lokal*

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

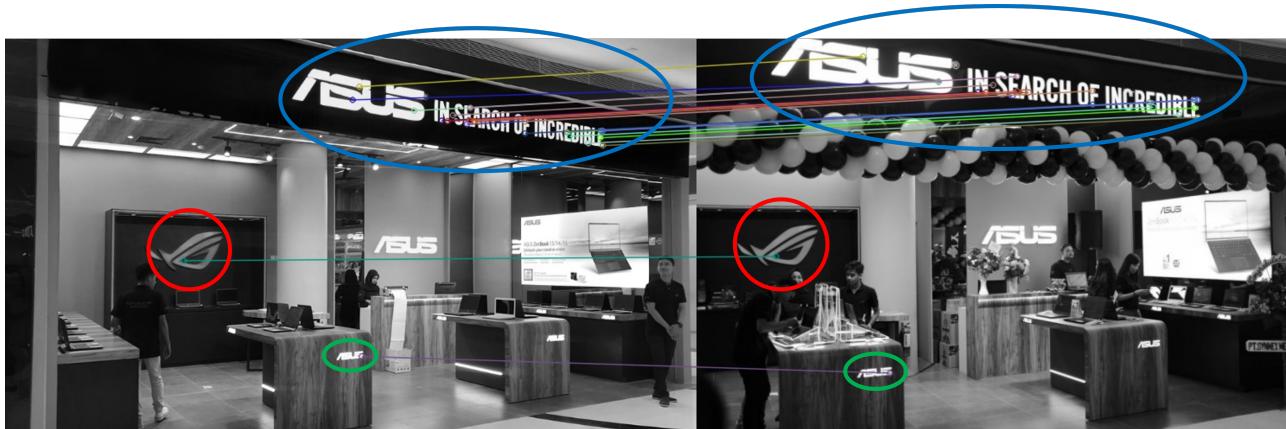
3.1.3 Implementasi

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

1. Melakukan *import library* OpenCV (cv2).
2. Memuat kedua gambar dengan menggunakan fungsi cv2.imread Masing-masing gambar akan disimpan dalam bentuk *array* 2 dimensi.
3. Membuat objek SIFT dengan menggunakan fungsi cv2.SIFT_create masukkan ke sebuah variabel bernama **sift**.
4. Menggunakan objek **sift** untuk melakukan ekstraksi fitur lokal dengan menggunakan fungsi **sift.detectAndCompute** dan memasukkan *array* gambar sebagai parameter. Fungsi akan mengembalikan sebuah *tuple* yang berisi objek **keypoint** dan sebuah *array* dua dimensi berukuran $n \times 128$ dengan n merupakan jumlah **keypoint**. Lakukan ini untuk kedua gambar.
5. Membuat objek *descriptor matcher* berbasis FLANN dengan menggunakan fungsi cv2.DescriptorMatcher_create(cv2.DescriptorMatcher_FLANNBASED) masukkan ke dalam objek bernama **matcher**. Objek ini berguna untuk membuat pasangan **keypoint** dengan menggunakan vektor *descriptor*.
6. Membuat pasangan **keypoint** dengan menggunakan fungsi **matcher.knnMatch(descriptors1, descriptors2, 2)**. Parameter **descriptors1** merupakan *descriptor* untuk Gambar *Q* dan **descriptors2** merupakan *descriptor* untuk Gambar *T* sedangkan angka 2 menunjukkan bahwa dicari dua pasangan yang paling mirip. Fungsi mengembalikan sebuah *list* dengan setiap elemennya berisi 2 pasangan, *list* ini disimpan dengan nama **knn_matches**. Setiap pasangan memiliki atribut **distance** yang menunjukkan jarak *descriptor* dari kedua **keypoint** di pasangan tersebut.
7. Menyaring pasangan **keypoint** yang cukup kuat dengan melakukan iterasi pada *list* **knn_matches** dan mengambil pasangan yang jarak pasangan terdekatnya kurang dari 0.3 kali jarak pasangan kedua terdekat. Masukkan pasangan yang terpilih ke dalam *list* bernama **good_matches**.
8. Melakukan visualisasi dari pasangan-pasangan **keypoint** di **good_matches** dengan terlebih dahulu menggambarkan pasangan **keypoint** pada gambar asal (Gambar *Q* dan Gambar *T*) menggunakan fungsi cv2.drawMatches. Gambar asal yang sudah digambarkan pasangan **keypoint**-nya tersebut lalu ditampilkan dengan menggunakan fungsi cv2.imshow.

3.1.4 Hasil Analisis

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



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

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

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

3.2 Analisis Fitur Lokal dari Logo dan Bagian yang Konsisten

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



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

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

Bagian POI yang merupakan logo dan objek yang konsisten tersebut tidak selalu dapat memberikan fitur lokal yang baik untuk dilakukan pengenalan. Terdapat beberapa bagian yang walaupun konsisten tetapi sifatnya tidak unik terhadap POI tersebut. Seperti dapat dilihat pada Gambar 3.5, bagian yang diberi lingkaran merah merupakan bagian yang berasal dari logo kedua POI tersebut. Walaupun berasal dari logo POI yang berbeda tetapi bentuk sudut yang ditunjukkan lingkaran merah tersebut mirip. Sudut yang mirip tersebut akan menghasilkan fitur lokal yang mirip juga. Fitur lokal yang berasal dari POI berbeda tetapi cirinya mirip dapat mempersulit proses pengenalan

POI. Untuk itu selain dicari fitur lokal yang konsisten perlu juga dicari fitur lokal yang sifatnya unik terhadap sebuah POI.



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

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

3.3 Dataset yang Digunakan

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

3.3.1 Dataset SMVS

Dataset SMVS atau Stanford Mobile Visual Search merupakan *dataset* yang dapat diakses melalui website Stanford Libraries (<https://exhibits.stanford.edu/data/catalog/rb470rw0983>). *Dataset* memiliki beberapa kumpulan data yang masing-masing merupakan kumpulan gambar dari benda-benda dengan jenis yang sama. Daftar kelompok data yang ada pada SMVS adalah sebagai berikut:

- **book_covers**: berisi gambar-gambar *cover* buku.
- **bussines_cards**: berisi gambar-gambar kartu nama.
- **cd_covers**: berisi gambar-gambar sampul kaset CD.
- **dvd_covers**: berisi gambar-gambar sampul kaset DVD.
- **landmarks**: berisi gambar-gambar dari bangunan-bangunan seperti gedung, rumah, dan lain-lain.
- **museum_paintings**: berisi gambar-gambar lukisan di museum.
- **print**: berisi gambar-gambar hasil foto dari dokumen hasil *print*.
- **video_frames**: berisi gambar-gambar yang merupakan foto dari layar yang sedang memutar video.

Pada penelitian ini *dataset* dari SMVS yang digunakan adalah *dataset book_covers*. *Dataset* tersebut digunakan karena gambar-gambar pada *dataset* ini cenderung mudah untuk diproses. Data dari kelompok *book_covers* juga terbagi dengan format sesuai dengan yang dibutuhkan pada penelitian, di mana terdapat beberapa kelas gambar dan untuk setiap kelas terdapat beberapa foto dari benda yang sama diambil dari sudut pengambilan yang berbeda.

Pada *dataset* SMVS ini terdapat kelompok gambar **landmark** yang berisi gambar bangunan-bangunan di sebuah lokasi tertentu yang menyerupai POI. Objek yang digunakan pada kelompok data tersebut sesuai dengan objek yang diteliti pada penelitian ini. Kelompok gambar **landmark** tersebut walaupun berisi objek yang sesuai, tetapi format data tidak sesuai dengan yang diperlukan

pada penelitian ini. Gambar-gambar dalam kelompok gambar `landmark` tidak terbagi menjadi kelas-kelas sesuai lokasi atau POI pada gambarnya.

3.3.2 Dataset GSV

Dataset GSV ini merupakan dataset berisi gambar-gambar POI. *Dataset* ini dikumpulkan dengan tujuan mengumpulkan data yang sesuai dengan topik penelitian. Pada penelitian ini objek yang diteliti adalah gambar dari sebuah POI, sedangkan pada *dataset* SMVS tidak terdapat data berisi POI yang sesuai dengan kebutuhan penelitian.

Dalam *dataset* GSV terdapat beberapa kelas, di mana setiap kelas merupakan sebuah POI tertentu. Setiap kelas terdiri dari gambar-gambar sebuah POI yang diambil dari sudut waktu pengambilan yang berbeda. Pengambilan dari sudut dan waktu yang berbeda ditujukan agar dalam gambar-gambar di sebuah kelas ada bagian dari POI yang selalu muncul dan ada yang hanya muncul di satu atau beberapa gambar saja.

Data-data gambar dalam *dataset* GSV ini diperoleh dari dua sumber yang berbeda. Sumber yang pertama merupakan Google Street View (<https://www.google.com/streetview/>). Google Street View merupakan sebuah aplikasi yang dapat digunakan untuk melihat tampilan 3D dari titik-titik di jalan raya. Dengan menggunakan Google Street View diambil gambar-gambar POI yang dapat dilihat dari titik di jalan raya sebagai data. Sumber yang kedua adalah dengan mengambil gambar secara langsung dengan menggunakan kamera. Gambar-gambar yang dihasilkan baik dari Google Street View dan dengan menggunakan kamera memiliki ciri yang sama sehingga keduanya dapat digabungkan ke dalam satu *dataset*.

Keseluruhan *dataset* terdiri dari sebanyak 150 gambar. Gambar-gambar tersebut terbagi menjadi 10 kelas, di mana setiap kelas memiliki 15 gambar berbeda. Contoh gambar untuk tiap 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	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)	
oxy	Jl. Ciumbuleuit No. 163, Ciumbuleuit, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87389, 107.60448)	
ping	Jl. Bukit Indah I No. 6, Ciumbuleuit, Cidadap, Kota Bandung, Jawa Barat 40141 (-6.87406, 107.60375)	

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)	
-------	---	--

3.4 Analisis Sifat Konsisten pada Fitur Lokal dari Gambar POI

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

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

3.4.1 Ide Dasar dan Tahapan Analisis

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

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

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

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

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

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

3.4.2 Implementasi

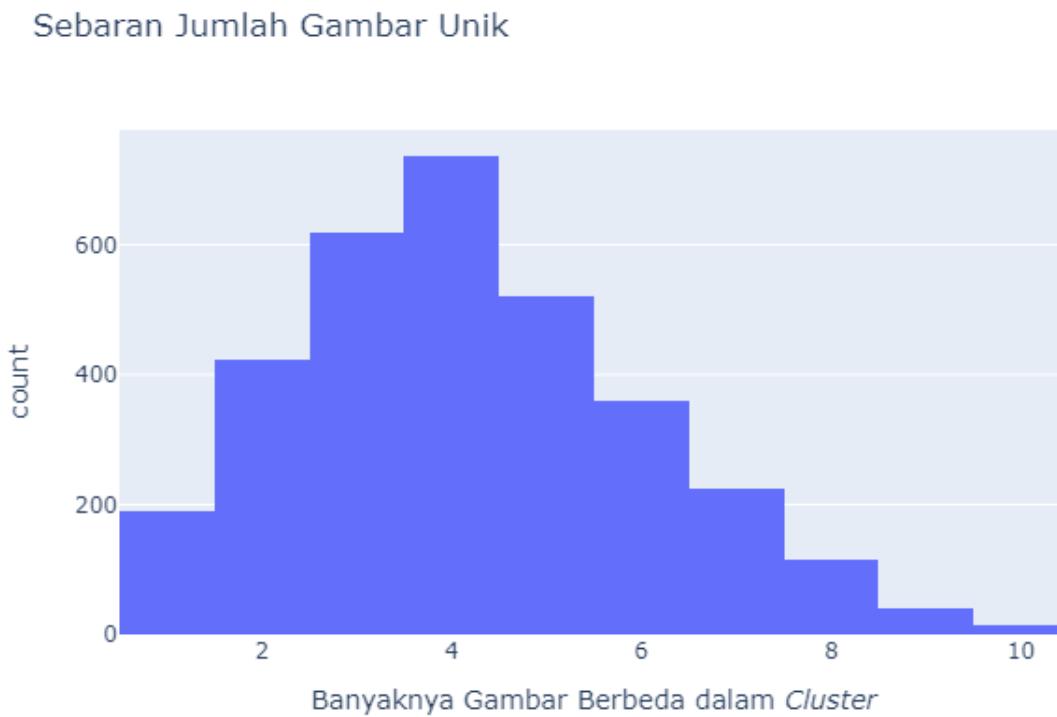
Analisis ini dilakukan dengan menggunakan Python versi 3.7.5 dari distribusi Anaconda. Seluruh tahap pemrosesan gambar dan ekstraksi fitur lokal dilakukan dengan menggunakan fungsi-fungsi

dari *library* OpenCV versi 4.5.5.64. Metode *clustering* yang digunakan adalah Metode *Agglomerative Clustering* dengan menggunakan implementasi dari *library* Scikit-learn versi 1.0.2. Tahap pemrosesan dan tabulasi data dilakukan dengan menggunakan *library* Pandas versi 1.3.5, serta visualisasi data dihasilkan dengan menggunakan *library* Plotly. Tahapan implementasi yang dilakukan adalah sebagai berikut:

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

3.4.3 Hasil

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



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

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

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

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



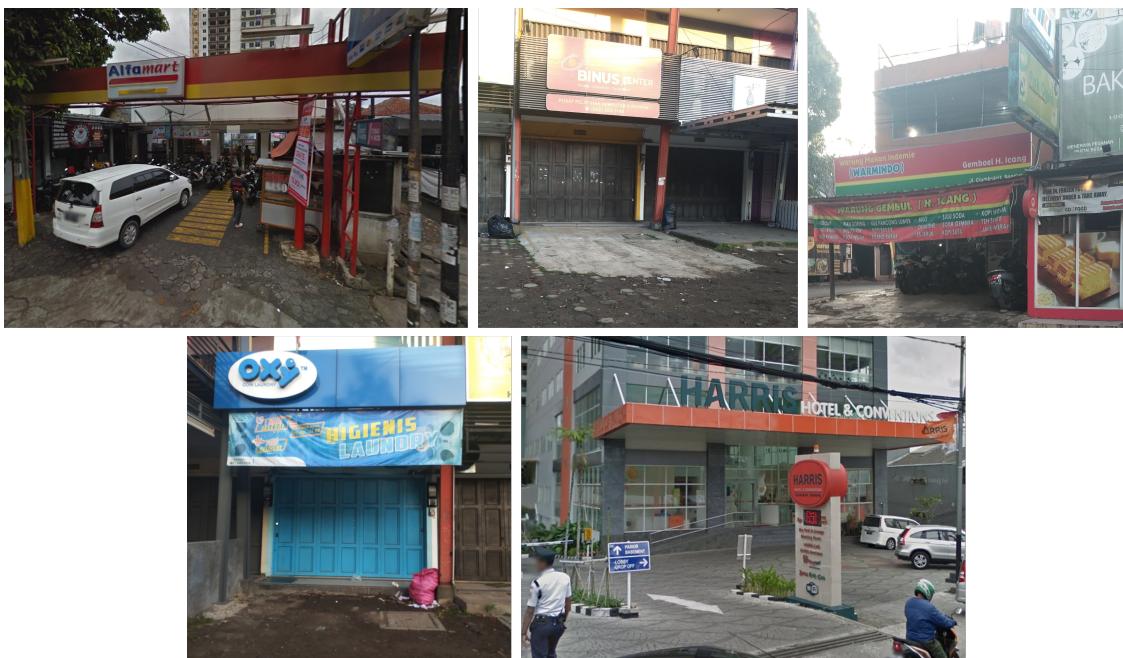
Gambar 3.8: Contoh *keypoint* yang konsisten menurut analisis ini.

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.

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

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

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

1. Memuat file gambar yang digunakan
2. Melakukan ekstraksi fitur pada semua gambar
3. Menggabungkan vektor-vektor *descriptor* dari semua gambar ke dalam sebuah *dataframe*. Setiap vektor *descriptor* akan dicatat gambar asalnya serta kelas gambarnya (nama POI)
4. Melakukan *clustering* pada seluruh vektor *descriptor* dari *dataframe*. *Clustering* dilakukan menggunakan cara yang sama dengan *clustering* pada
5. Menghitung banyak kelas gambar berbeda yang muncul dalam setiap *cluster*. **3.4.**

Setelah melakukan semua tahapan di atas akan didapat *cluster-cluster* fitur lokal beserta dengan jumlah kelas gambar yang berbeda untuk tiap *cluster*-nya. Jumlah kelas gambar berbeda tersebut akan menentukan apakah fitur lokal dalam *cluster* memiliki sifat yang unik atau tidak.

3.5.2 Implementasi

Analisis ini dilakukan dengan menggunakan Python versi 3.7.5 dari distribusi Anaconda. Seluruh tahap pemrosesan gambar dan ekstraksi fitur lokal dilakukan dengan menggunakan fungsi-fungsi dari *library* OpenCV versi 4.5.5.64. Metode *clustering* yang digunakan adalah Metode *Agglomerative Clustering* dengan menggunakan implementasi dari *library* Scikit-learn versi 1.0.2. Tahap pemrosesan dan tabulasi data dilakukan dengan menggunakan *library* Pandas versi 1.3.5, serta untuk visualisasi

data dihasilkan dengan menggunakan *library* Plotly. Tahapan implementasi yang dilakukan adalah sebagai berikut:

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

3.5.3 Hasil

Setelah selesai dilakukan semua tahapan implementasi akan didapat kelompok-kelompok fitur lokal serta jumlah kelas gambar yang unik untuk setiap kelompok. Banyaknya kelas gambar yang unik tersebut menentukan apakah sifat fitur-fitur lokal dalam kelompok tersebut termasuk unik atau tidak. Jumlah kelas gambar unik yang tinggi menunjukkan bahwa sebuah kelompok memiliki sifat fitur lokal yang tidak unik, karena jumlah kelas gambar unik yang tinggi berarti sifat fitur lokal pada gambar tersebut muncul di banyak jenis POI yang berbeda dan tidak unik terhadap satu POI saja. Sebaran nilai jumlah kelas gambar yang unik dapat dilihat pada histogram di Gambar 3.10.



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

Terlihat dari histogram pada Gambar 3.10 bahwa ada *cluster* dengan jumlah kelas gambar unik yang sedikit seperti 1 dan 2, di mana merupakan *cluster* tersebut merupakan *cluster* dengan sifat yang unik. Jumlah *cluster* dengan sifat yang unik termasuk sedikit dibandingkan dengan jumlah *cluster* yang sifatnya tidak unik. Sedikitnya jumlah *cluster* dengan sifat yang unik menunjukkan bahwa dalam sebuah gambar POI sebagian besar fitur lokal merupakan fitur lokal yang sifatnya tidak unik.

Beberapa contoh *keypoint* yang fitur lokalnya memiliki sifat unik berdasarkan analisis ini dapat dilihat pada Gambar 3.11. *Keypoint* yang ditampilkan adalah yang berasal dari *cluster* di mana 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 Kekonsistenan 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 serta unik dan ada yang tidak. Tetapi dari kedua analisis tersebut belum didapat didapat sebuah nilai yang dapat menentukan tingkat kekonsistenan atau keunikan suatu fitur lokal. Analisis pada bagian ini bertujuan untuk membuat sebuah metode yang dapat memberikan nilai konsistensi dan keunikan pada suatu fitur lokal. Nilai konsistensi dan keunikan ini nantinya akan berguna untuk menyaring fitur lokal, sehingga dapat dengan mudah diambil fitur lokal yang sifatnya unik dan konsisten saja.

Analisis yang dilakukan merupakan penggabungan dari analisis di 3.4 dan 3.5, serta dengan beberapa modifikasi untuk mengatasi masalah yang ditemui. Nilai konsistensi dan 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 dan 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-ciri yang mirip. Banyaknya fitur lokal dengan ciri yang sama tersebut diatasi dengan terlebih dahulu melakukan *clustering* pada masing-masing gambar. *Clustering* yang dilakukan pada masing-masing gambar akan menghasilkan *centroid* dari masing-masing *cluster*. *Centroid-centroid* ini nantinya digabungkan dengan *centroid-centroid* dari gambar lain untuk dilakukan *clustering* tahap kedua. *Cluster-cluster* yang terbentuk dari *clustering* tahap kedua ini digunakan untuk menentukan nilai konsistensi dan keunikan fitur lokal.

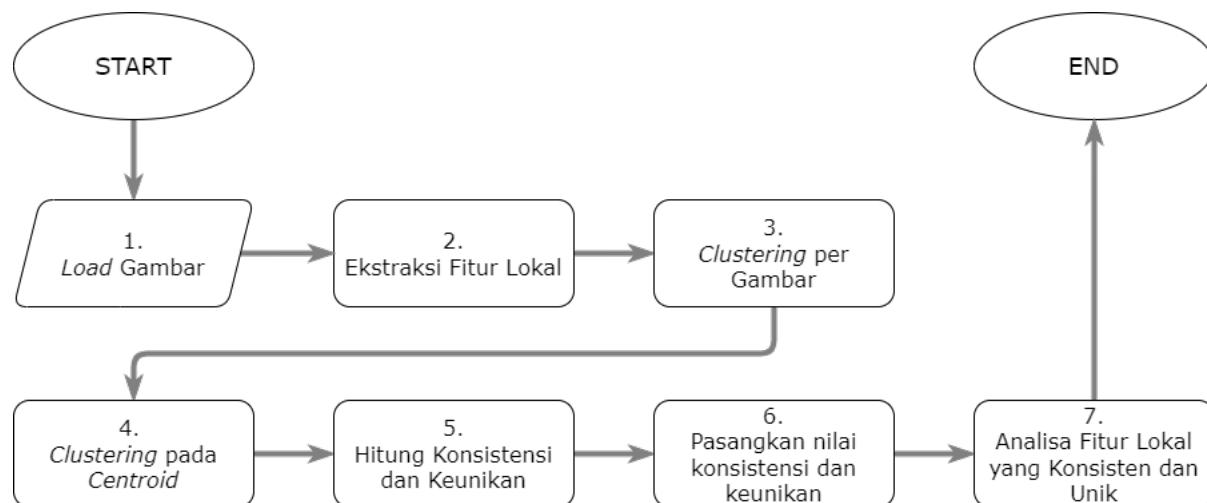
Fitur lokal yang konsisten adalah fitur lokal yang muncul di semua gambar dari sebuah kelas. Kekonsistenan ini dihitung dengan melihat fitur-fitur lokal dalam sebuah *cluster* jika dalam *cluster*

tersebut terdapat fitur lokal dari kelas A dan terdapat beberapa fitur lokal lain dari kelas yang sama tetapi dari gambar yang berbeda maka fitur lokal dengan kelas A di *cluster* tersebut merupakan fitur lokal yang konsisten. Kekonsistennan ini akan ditunjukkan oleh sebuah nilai yang didapat dari menghitung jumlah gambar yang berbeda dari kelas POI yang sama dalam sebuah *cluster* dan membaginya dengan jumlah gambar POI tersebut dalam *dataset*.

Fitur lokal unik adalah fitur lokal yang hanya muncul di satu kelas POI dan tidak muncul di POI lain. Dapat dihitung dengan fitur-fitur lokal dalam sebuah *cluster*. Jika sebagian besar fitur lokal berasal dari kelas gambar A maka fitur lokal dengan kelas gambar A di *cluster* tersebut merupakan fitur lokal yang unik. Nilai keunikan didapat dari terlebih dahulu menghapus fitur lokal yang gambarnya memiliki duplikat dan lalu untuk tiap kelas gambar dihitung jumlahnya dan dibagi dengan jumlah anggota *cluster* tersebut (setelah dihapus yang duplikat).

3.6.2 Tahapan Analisis

Langkah-langkah analisis yang dilakukan untuk mencari fitur lokal dengan sifat konsisten dan unik 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* pada *centroid-centroid* tersebut. Dari *cluster-cluster* yang terbentuk dianalisis pada tahapan berikutnya.

5. Hitung Konsistensi dan Keunikan

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

6. Pasangkan Nilai Konsistensi dan Keunikan

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

7. Analisa Fitur Lokal yang Konsisten dan Unik

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

3.6.3 Implementasi

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

1. Import *library* OpenCV (`cv2`), Scikit-learn (`sklearn`), dan Pandas (`pandas`).
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*.
3. Membuat objek SIFT dengan fungsi `cv2.SIFT_create` masukkan ke dalam objek dengan nama `sift`.
4. Menggunakan objek `sift` untuk melakukan ekstraksi fitur lokal dari masing-masing gambar. Ekstraksi dilakukan dengan fungsi `sift.detectAndCompute`. *Keypoint* yang dihasilkan dimasukkan ke dalam sebuah *list* bernama `keypoints` untuk semua gambar. Untuk `descriptor` dimasukkan ke dalam sebuah *dataframe* dengan nama `descriptors`.
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`.
6. Mengelompokkan *dataframe* `descriptors` berdasarkan kolom `img` dengan menggunakan fungsi `groupby`.
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](#).
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`), dan kelas gambar (`img_class`).
9. Melakukan *clustering* pada *dataframe* `df_centroid`. *Clustering* dilakukan dengan cara yang sama seperti sebelumnya. Nomor *cluster* dimasukkan ke dalam `df_centroid` sebagai kolom dengan nama `cluster2_label`.
10. Membuat sebuah *dictionary* bernama `cluster2_class_count` yang nantinya digunakan untuk menyimpan nilai keunikan.
11. Mengelompokkan data pada `df_centroid` berdasarkan kolom `cluster2_label`. Untuk setiap kelompok pertama hanya ambil satu baris dari tiap gambar (`drop_duplicates(subset='img')`). Setelah itu hitung persentase banyaknya tiap kelas gambar dengan fungsi `value_counts(normalize=True)` pada kolom `img_class`. Masukkan hasilnya ke dalam `cluster2_class_count`. *Dictionary* `cluster2_class_count` akan memiliki nomor dari `cluster2_label` sebagai *key* dan *value*-nya akan berisi sebuah *series* dengan *key* merupakan kelas gambar dan isinya persentase kemunculan kelas gambar tersebut.
12. Membuat sebuah *list* dengan nama `uniqueness`. Isi *list* tersebut dengan melakukan iterasi pada kolom `cluster2_label` dan `img_class` dan ambil nilai persentase keunikan yang bersesuaian dari `cluster2_class_count`.
13. Memasukkan *list* `uniqueness` sebagai kolom pada `df_centroid`.
14. Mengelompokkan `df_centroid` berdasarkan `cluster2_label` dan `img_class` dengan menggunakan fungsi `groupby`. Untuk setiap kelompok dihitung jumlah gambar yang berbeda. Hasil

- pengelompokan dimasukkan ke dalam variabel bernama `cluster2_img_class_group`.
15. Membuat sebuah *list* dengan nama `img_count`. *List* ini akan berisi nilai konsistensi untuk tiap *centroid* di `df_centroid`.
 16. Melakukan iterasi pada kolom `cluster2_label` dan `img_class` dari `df_centroid`. Untuk setiap iterasi ambil nilai yang bersesuaian dari `cluster2_img_class_group`. Nilai tersebut dibagi dengan 4 (jumlah gambar tiap kelas) dan dimasukkan ke dalam *list* `img_count`.
 17. Masukkan `img_count` ke sebagai kolom di `df_centroid` dengan nama `consistency`.
 18. Mengambil kolom `cluster_label`, `cluster2_label`, `uniqueness`, dan `consistency` dari `df_centroid`. Lakukan `merge` pada kolom-kolom tersebut dengan *dataframe* `descriptors` dengan bertumpu pada kolom `cluster_label`.

3.6.4 Hasil Analisis

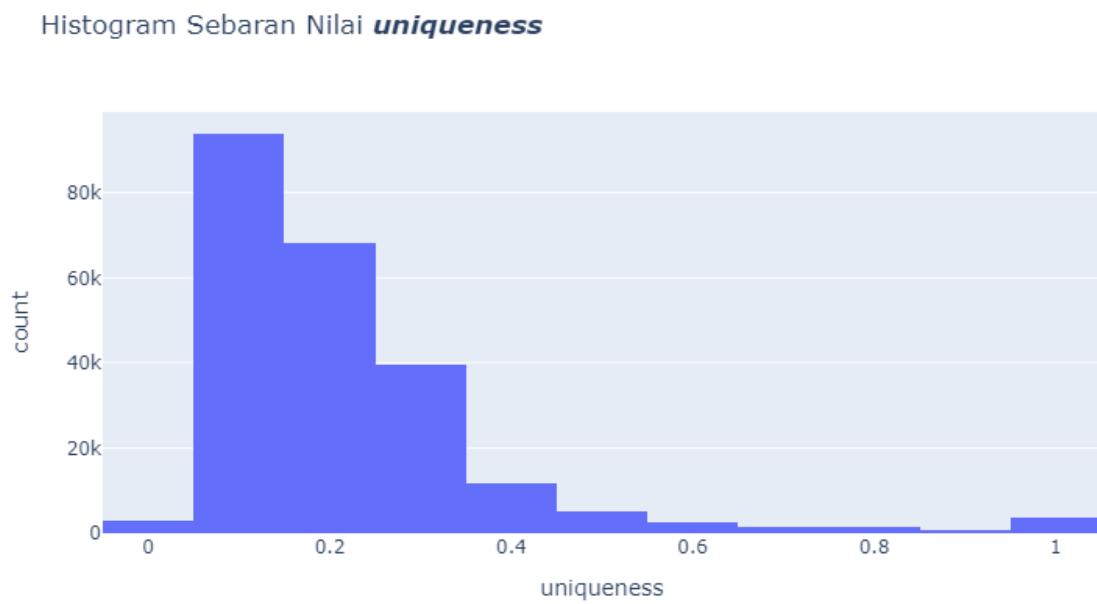
Setelah dilakukan implementasi sesuai dengan tahapan pada 3.6.3 didapatkan tabel berisi *descriptor* untuk *keypoint* beserta dengan nilai keunikan (*uniqueness*) dan konsistensinya (*consistency*). Beberapa contoh potongan dari tabel hasil dapat dilihat pada Tabel 3.2

<code>img</code>	<code>img_class</code>	<code>cluster_label</code>	<code>cluster2_label</code>	<code>uniqueness</code>	<code>consistency</code>
...
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

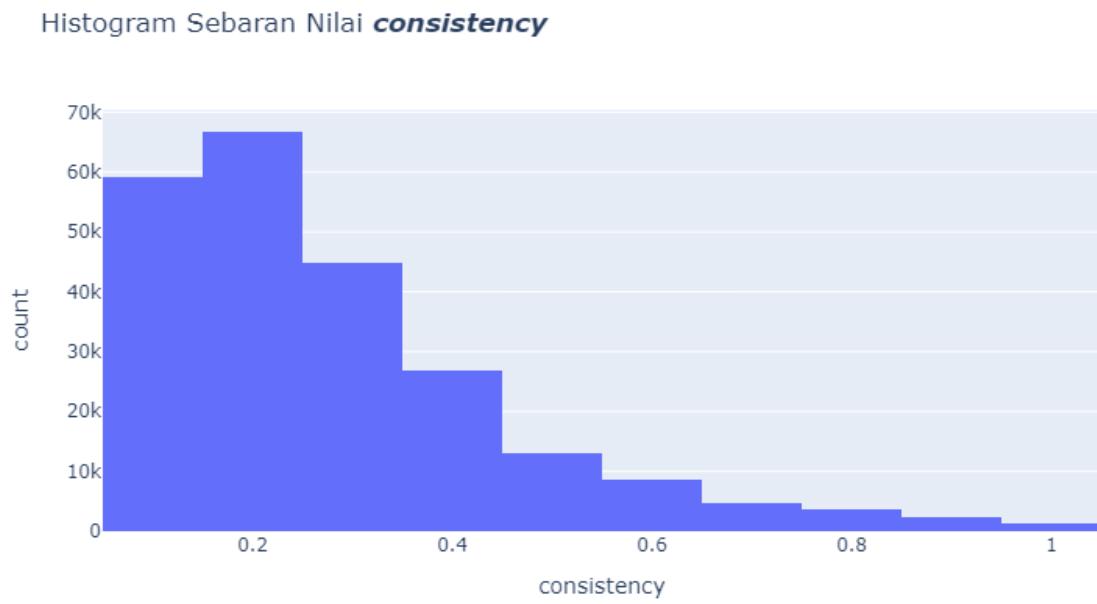
Tabel 3.2: Potongan beberapa contoh dari tabel *descriptor*.

Nilai-nilai pada kolom `uniqueness` dan `consistency` didapat dari melakukan penghitungan sebaran anggota tiap *cluster* seperti yang telah dijelaskan di subbab-subbab sebelumnya.

Data pada Tabel 3.2 menunjukkan fitur lokal dari *keypoint* beserta nilai konsistensi (`consistency`) dan keunikannya (`uniqueness`). Dari tabel tersebut terlihat bahwa nilai keduanya cukup beragam. Sebaran untuk nilai keunikan dan konsistensi dapat dilihat pada Gambar 3.14 dan Gambar 3.15.



Gambar 3.14: Histogram sebaran nilai keunikan.



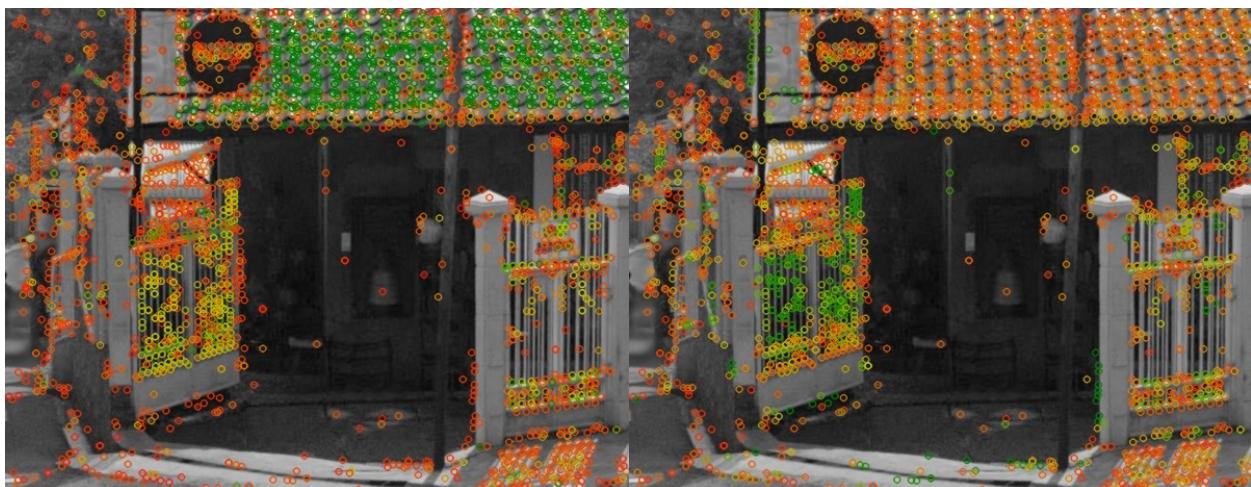
Gambar 3.15: Histogram sebaran nilai keunikan.

3.6.5 Visualisasi Nilai Konsistensi dan Keunikan Fitur Lokal

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

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

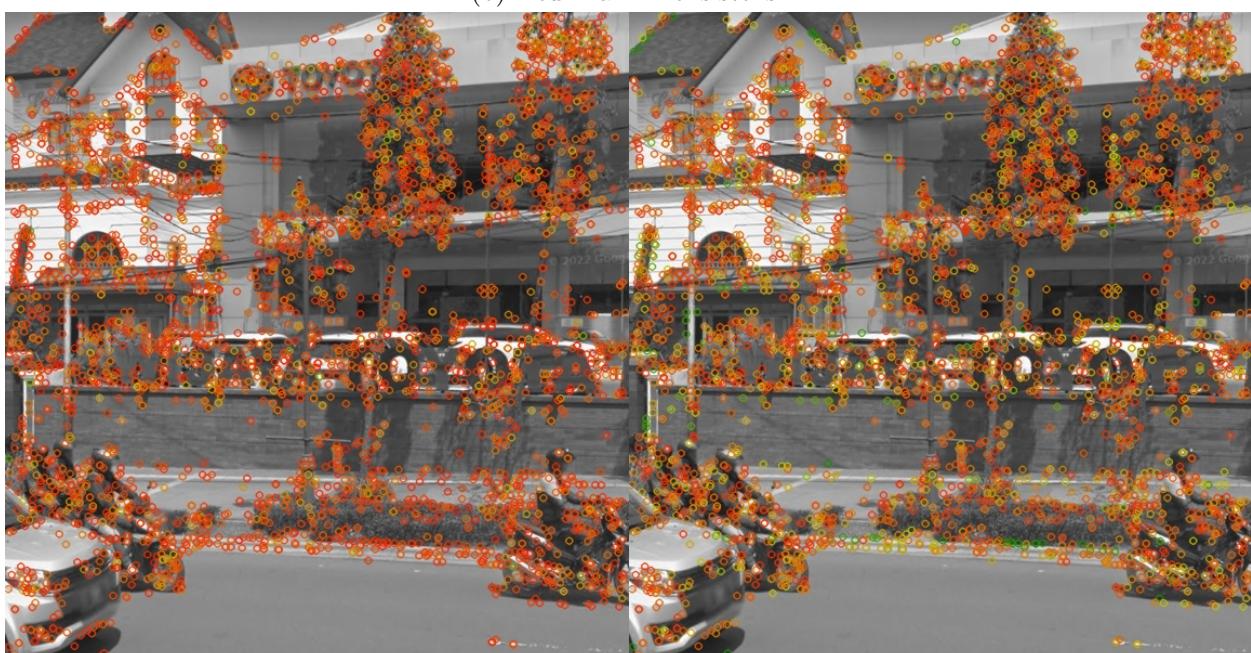
Untuk melihat nilai konsistensi dan keunikan dari fitur lokal pada bagian-bagian POI maka perlu untuk menampilkan *keypoint* dari fitur lokal pada gambar asalnya. *Keypoint* yang ditampilkan pada gambar asal tersebut perlu untuk memiliki sebuah indikasi untuk menunjukkan nilai konsistensi dan keunikan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

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

Gambar 3.16a

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

Gambar 3.16b

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

Gambar 3.16c

- Gambar ini merupakan contoh kelas yang hasil penghitungan nilai konsistensi dan keunikan memberikan hasil yang kurang baik.
- Baik dari nilai konsistensi maupun keunikan, sangat sedikit *keypoint* yang memiliki nilai tinggi (warna hijau).
- Terlihat beberapa *keypoint* berwarna hijau pada gambar konsistensi, tetapi letaknya tersebar 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

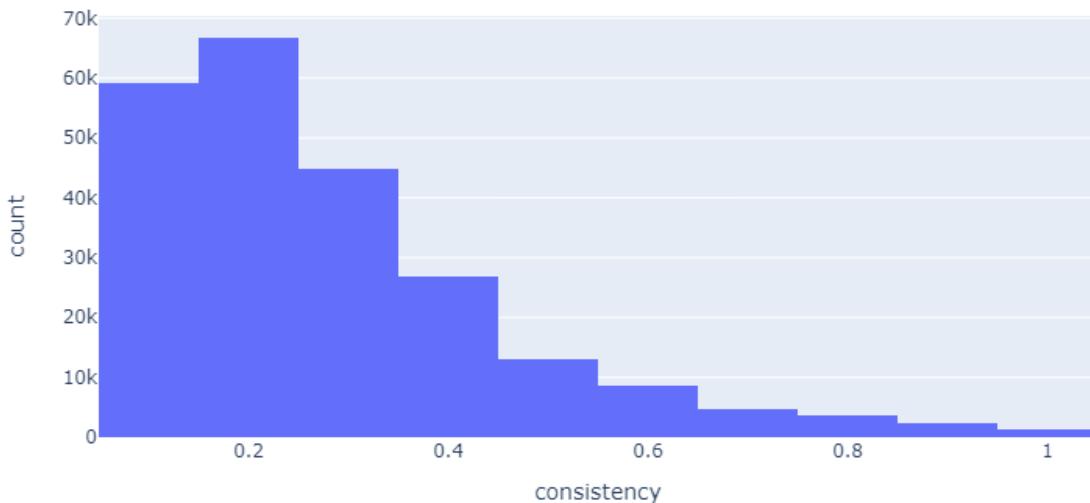
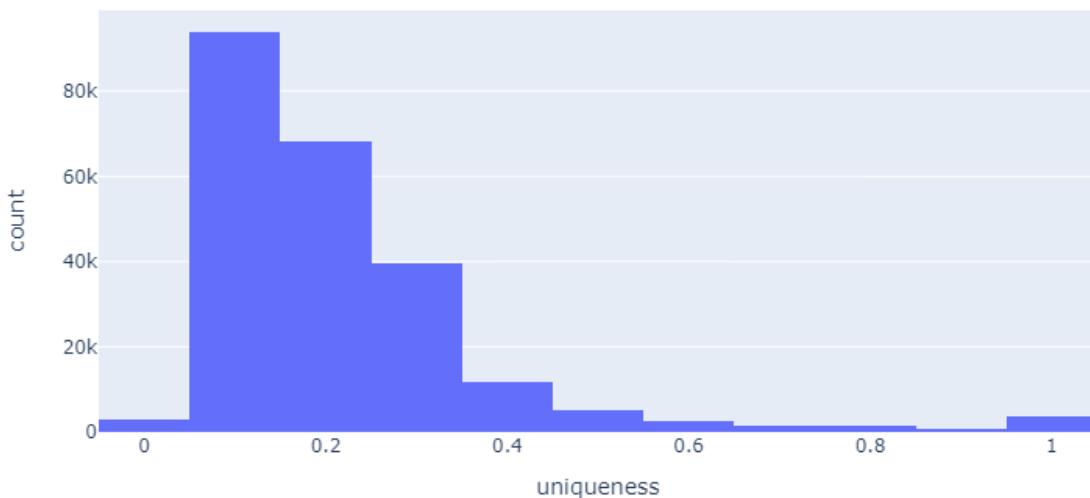
dan unik dalam penelitian ini diartikan sebagai objek yang selalu terlihat dalam gambar-gambar POI dan memiliki bentuk atau pola yang tidak terlihat di POI lain. Contoh yang paling mudah dari objek-objek dengan sifat yang konsisten dan unik ini adalah logo dari POI. Gambar 3.17 merupakan beberapa contoh 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 ditunjukan oleh dua histogram pada Gambar 3.18. Jika dilihat dari kedua histogram pada Gambar 3.18 tersebut baik nilai konsistensi dan keunikan tersebut 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.

Jika dilihat dari contoh pada Gambar 3.17 bagian yang konsisten dan unik dari dari sebuah gambar POI hanya merupakan bagian kecil jika dibandingkan dengan keseluruhan gambar. Karena ingin diambil fitur lokal dari bagian POI yang merupakan bagian kecil dari gambar maka hanya perlu diambil sebagian kecil dari keseluruhan fitur lokal. Oleh karena itu—dengan melihat sebaran dari kedua histogram pada Gambar 3.18—fitur lokal dengan nilai konsistensi dan keunikan < 0.6 akan dibuang terlebih dahulu. Pembuangan fitur lokal ini karena fitur lokal dengan nilai konsistensi dan keunikan tersebut diasumsikan berasal dari bagian POI dengan sifat tidak konsisten dan tidak unik yang merupakan sebagian besar dari sebuah gambar POI.

3.7.1 Ide Dasar dan Tahapan Analisis

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

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

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

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

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

3.7.2 Metode Scoring

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

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

```

1 <annotation>
2   <folder>annotated_alfamart</folder>
3   <filename>alfamart_1.jpg</filename>
4   <path>D:\ Skripsi - Program3\ datasets\ annotated_poi\ alfamart\ alfamart_1.
      jpg</path>
5   <source>
6     <database>Unknown</database>
7   </source>
8   <size>
9     <width>600</width>
10    <height>474</height>
11    <depth>1</depth>
12  </size>
13 <segmented>0</segmented>
14 <object>
15   <name>target </name>
16   <pose>Unspecified </pose>
17   <truncated>0</truncated>
18   <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>
```

Potongan kode tersebut menunjukkan sebuah *file xml* yang dihasilkan dari LabelImg. Pada *file* tersebut terlihat bahwa ada sebuah anotasi dengan nama **target** yang ditunjukkan oleh *tag object*.

Area yang ditandai sebagai target dapat dilihat pada *tag bndbox* yang berada di dalam *tag object*.

Fungsi penandaan daerah tersebut akan digunakan pada setiap gambar untuk membuat *file xml* yang menunjukkan anotasi pada gambar tersebut. Sebuah gambar dapat memiliki lebih dari satu anotasi, di mana beberapa anotasi berbeda itu akan disimpan pada sebuah *file xml* yang sama.

Pada analisis ini setiap gambar yang digunakan pada *dataset* akan terlebih dahulu diberi anotasi dengan label *target*. Setiap gambar dapat memiliki lebih dari satu anotasi (ada beberapa objek konsisten dan unik pada POI yang saling terpisah). Setiap *keypoint* yang didapat setelah tersaring dengan *threshold* akan ditentukan apakah *keypoint* tersebut berada di dalam atau di luar daerah *target*. Nilai *score* akan ditentukan dari persentase *keypoint* yang berada di dalam daerah *target*.

3.7.3 Hasil Analisis

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

Nilai Konsistensi

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

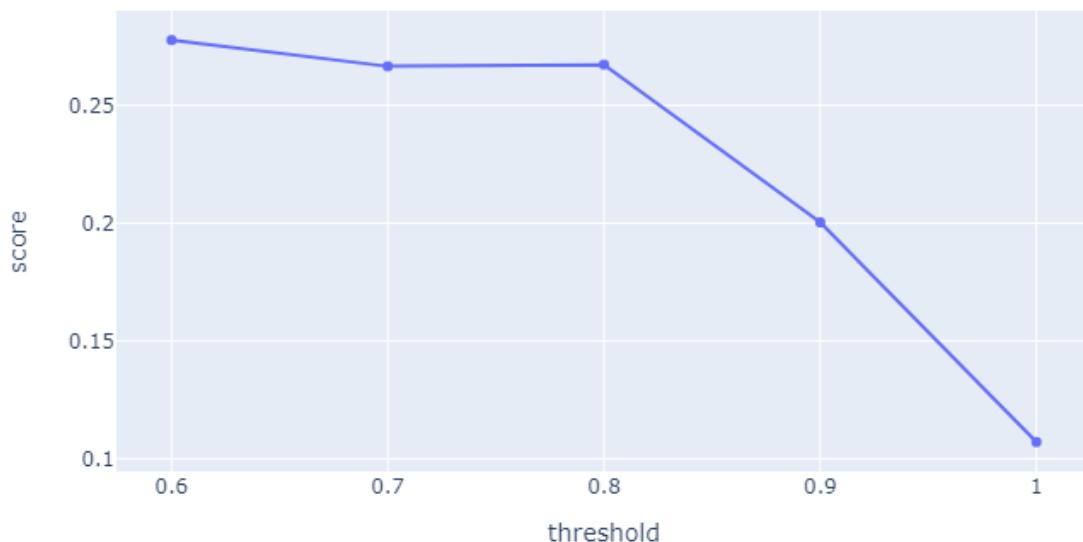
threshold	0.6	0.7	0.8	0.9	1.0
img_class					
alfamart	0.149268	0.144771	0.158488	0.089263	0.112664
binus	0.507728	0.476035	0.294314	0.010000	0.000000
gembul	0.287119	0.291524	0.311475	0.264435	0.186858
harris	0.186551	0.182550	0.185563	0.233336	0.000000
oxy	0.083837	0.071717	0.061667	0.061667	0.000000
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.

Jika dilihat dari Tabel 3.3 skor ketepatan cenderung berada pada angka yang kecil. Skor ketepatan juga nilainya sangat berbeda tiap kelas gambar. Contohnya seperti kelas *ping* nilainya bisa mencapai ~0.7 pada *threshold* 0.6, dibandingkan dengan skor untuk kelas *oxy* yang nilai tertingginya hanya pada ~0.08 saja. Nilai skor yang sangat berbeda tersebut menunjukkan bahwa ada kelas-kelas POI yang cenderung mudah dikenali dan ada yang lebih sulit untuk dikenali. POI yang mudah dikenali mungkin adalah POI yang memiliki objek dengan sebuah pola yang kuat dan mudah terdeteksi oleh algoritma pencarian fitur lokal.

Untuk memilih *threshold* mana yang paling baik digunakan untuk pemotongan pada nilai konsistensi maka akan dihitung rata-rata nilai skor seluruh kelas untuk tiap *threshold*. Hasilnya 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.

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

Nilai Keunikan

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

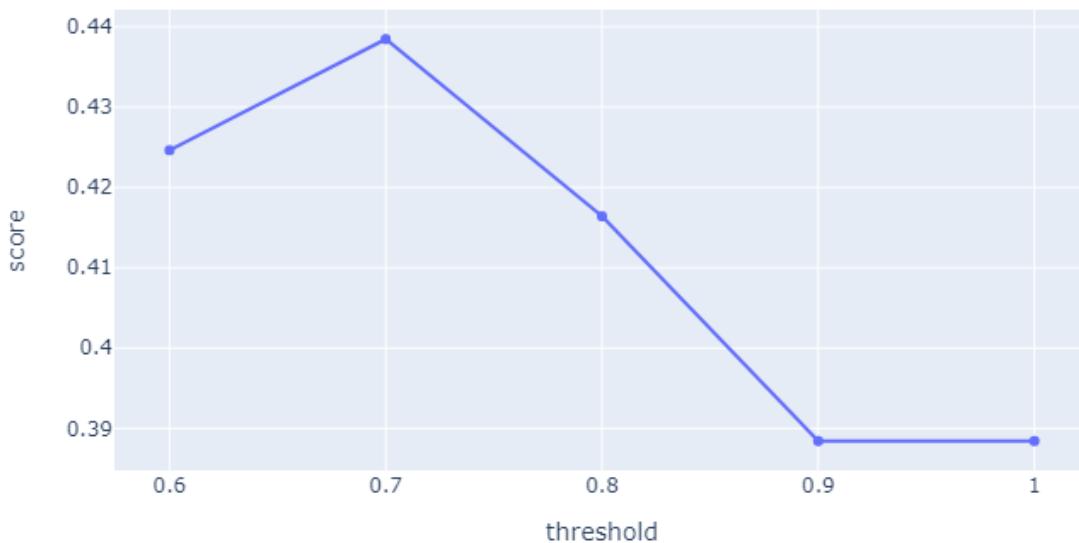
threshold	0.6	0.7	0.8	0.9	1.0
img_class					
alfamart	0.515710	0.366667	0.383333	0.100000	0.100000
binus	0.677219	0.680215	0.676765	0.591431	0.591431
gembul	0.635185	0.660688	0.679726	0.680655	0.680655
harris	0.208597	0.158760	0.059408	0.154762	0.154762
oxy	0.161507	0.053333	0.055098	0.000000	0.000000
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 Clustering 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. Hasil tes keduanya lalu dibandingkan tingkat akurasi dan waktu pemrosesannya. Proses identifikasi dilakukan dengan menggunakan metode BSIS 2.5.

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.



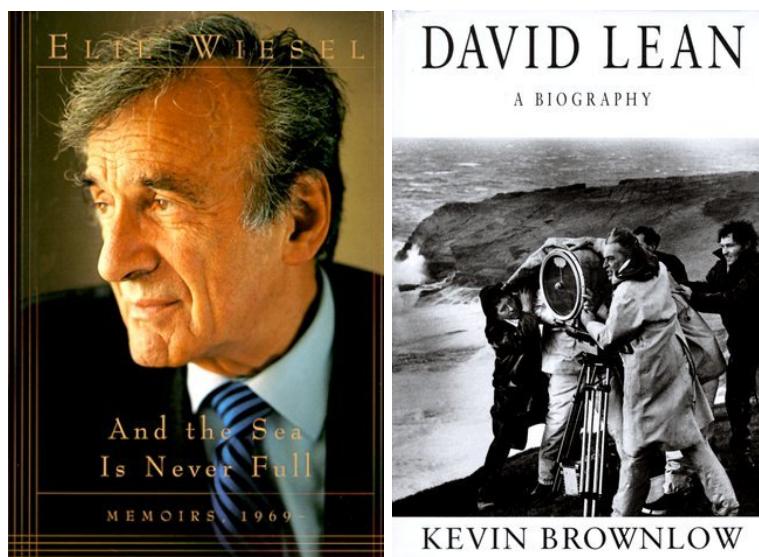
Gambar 3.22: Contoh gambar pada *dataset* Book Covers. Keempat gambar tersebut berada dalam satu kelas yang sama.

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.

Gambar-gambar tersebut diekstrak fitur lokalnya dan untuk setiap fitur lokal dihitung nilai konsistensi dan keunikan menggunakan metode seperti pada 3.6. Fitur-fitur lokal dari gambar ini lah yang akan digunakan untuk melakukan identifikasi gambar tersebut.

Data Test

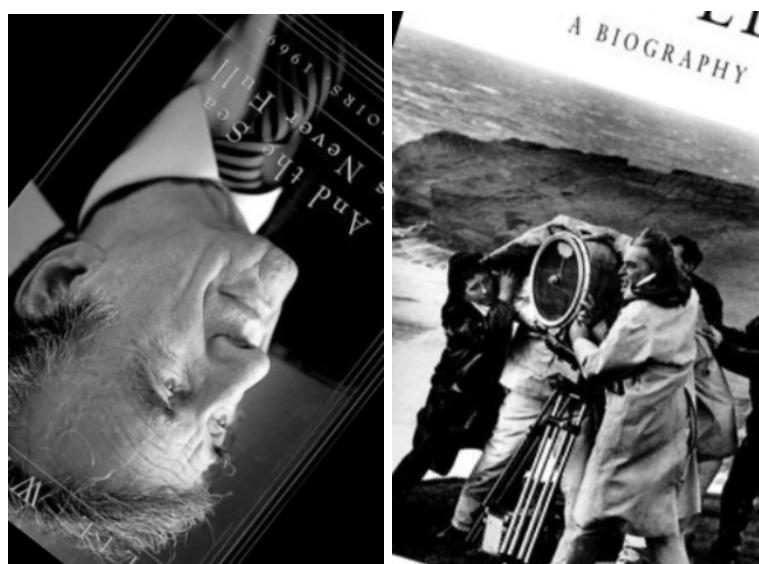
Data *test* yang digunakan diambil dari gambar referensi *dataset* Book Covers. Gambar referensi merupakan gambar buku yang menjadi objek utama dalam setiap kelas dengan kondisi yang ideal. Contoh gambar referensi dapat dilihat pada Gambar 3.23.



Gambar 3.23: Contoh gambar referensi dari *dataset Book Covers*.

Gambar-gambar referensi tersebut berjumlah total 50 gambar masing-masing untuk tiap kelas. Dari 50 gambar tersebut lalu ditransformasi dengan mengubah tingkat perbesaran dan rotasi secara acak. Transformasi tersebut dilakukan sebanyak dua kali untuk tiap gambar. Contoh gambar referensi yang telah ditransformasi dapat dilihat pada Gambar 3.24. Masing-masing hasil transformasi disimpan sebagai satu gambar sehingga terdapat total sebanyak 100 gambar untuk data *test*.

Data *test* yang berjumlah 100 tersebut memiliki ukuran lebar yang berkisar antara 133 – 441 *pixel*, sedangkan untuk tingginya berkisar antara 198 – 502 *pixel*.



Gambar 3.24: Contoh gambar referensi dari *dataset Book Covers*.

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*.

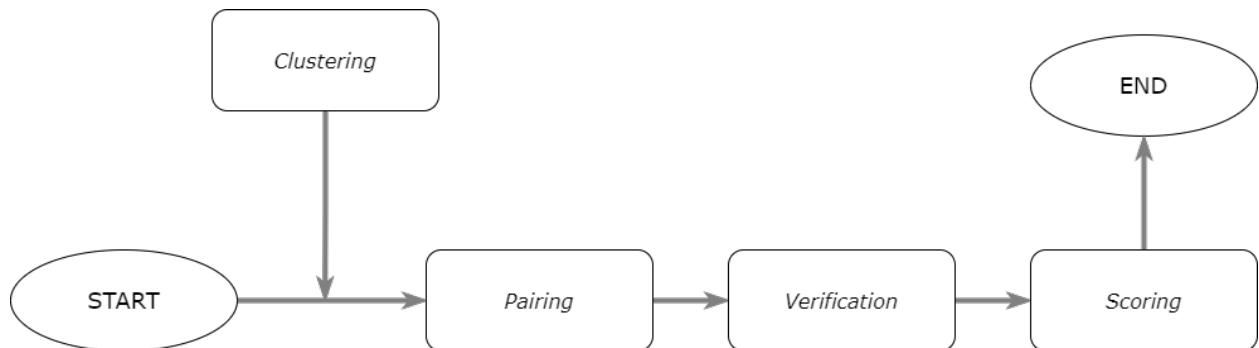
2. Verification

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

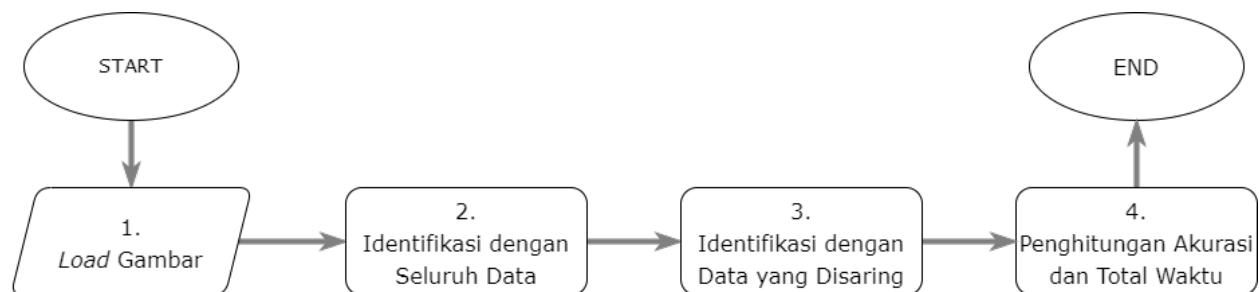


Gambar 3.25: Modifikasi pada metode BSIS yang dilakukan pada analisis ini.

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

3.8.3 Tahapan Analisis

Analisis dilakukan dengan tahapan pada dua *dataset* yang berbeda dengan tahapan yang sama. Dataset pertama akan disebut Dataset 400 yang menggunakan gambar dengan ukuran sisi terbesarnya tidak lebih dari 400 *pixel*. Dataset kedua disebut Dataset 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.

Secara rinci tahapan yang dilakukan adalah sebagai berikut:

1. Load Gambar

Load semua gambar *test* yang digunakan.

2. Identifikasi dengan BSIS pada seluruh Dataset

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*. Untuk setiap gambar hitung waktu yang digunakan untuk melakukan BSIS hingga didapat hasilnya. Total waktu tidak termasuk waktu yang digunakan untuk ekstraksi fitur.

3. Identifikasi dengan BSIS pada Dataset yang Telah tersaring

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

4. Penghitungan Akurasi dan Total Waktu

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

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

3.8.4 Tahapan Implementasi

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 data. Langkah-langkah implementasi yang dilakukan adalah sebagai berikut:

1. Melakukan *import* untuk *library* yang digunakan: OpenCV (*cv2*) dan Pandas (*pandas*).
2. Memuat semua gambar *test* masukan ke dalam variabel *list* bernama *test_dataset*. Setiap elemen dalam *test_dataset* berupa *tuple* dengan dua elemen. Elemen pertama merupakan nama gambar dan elemen kedua merupakan *array* gambar tersebut.
3. Memuat *dataset* yang digunakan untuk data *train*.
4. Menentukan parameter batas nilai **uniqueness** dan **consistency** untuk menyaring *dataset*.
5. Membuat objek SIFT dengan fungsi *cv2.SIFT_create*.
6. Melakukan identifikasi gambar dengan BSIS. Tahap *pairing* dalam BSIS dilakukan dengan menggunakan metode KD-Tree.
7. Menyimpan 10 nilai berikut dari hasil BSIS. Nilai-nilai ini disimpan dalam sebuah *list* untuk tiap nilai:
 - **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 melakukan 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.
8. Menggabungkan 10 *list* tersebut ke dalam sebuah *dataframe*. Masing-masing *list* menjadi satu kolom dalam *dataframe*.
9. Untuk setiap baris di *dataframe* memeriksa apakah hasilnya benar. Hasil adalah benar jika nilai pada **q_class** sama dengan **most_similar_class**.
10. Melakukan lagi langkah 6-8 dengan *dataset* yang telah tersaring berdasarkan nilai batas **consistency** dan **uniqueness** yang telah ditentukan sebelumnya.

Langkah-langkah tersebut dilakukan untuk Dataset 400 dan Dataset 600. Hasil dari kedua *dataset*

tersebut lalu dibandingkan.

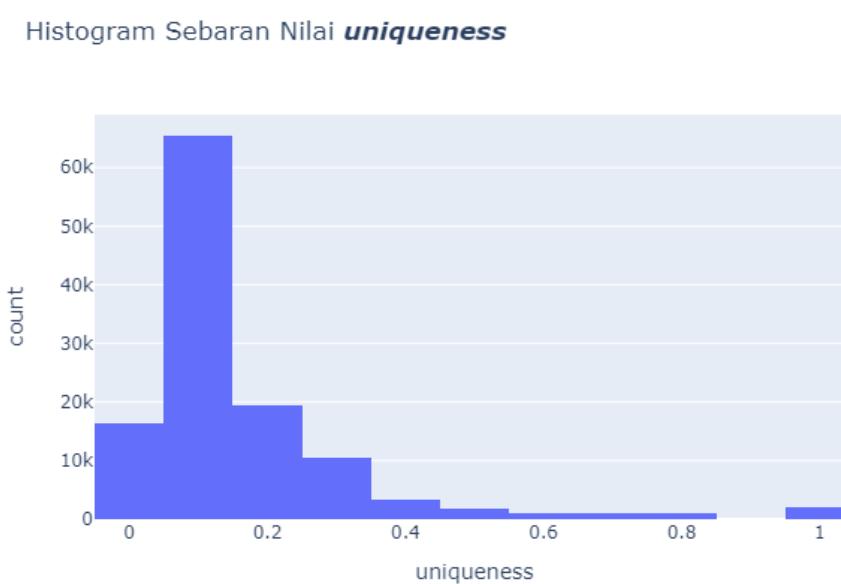
3.8.5 Hasil Analisis

Dataset 400

Dataset 400 memiliki total fitur lokal yang dihasilkan dari seluruh gambar sebanyak 121909 fitur lokal. Setelah dilakukan *clustering* menghitung nilai keunikan dan konsistensi didapat hasil sebaran nilai keunikan dan konsistensi seperti pada Gambar 3.27 dan Gambar 3.28.



Gambar 3.27: Sebaran nilai keunikan (**consistency**) untuk Dataset 400.



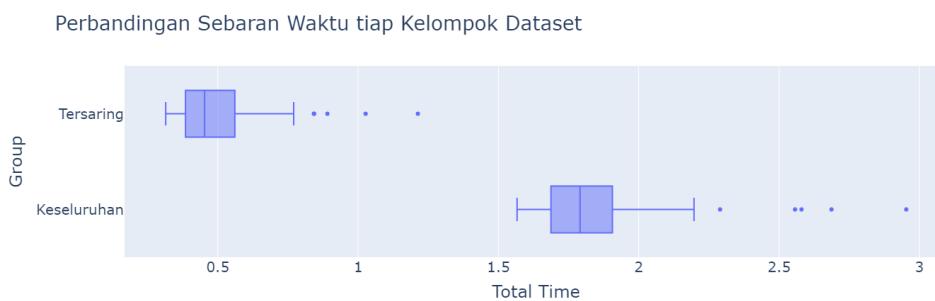
Gambar 3.28: Sebaran nilai keunikan (**uniqueness**) untuk Dataset 400.

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:

- Memiliki nilai **consistency** ≥ 0.5
- Memiliki nilai **uniqueness** ≥ 0.2

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 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.



Gambar 3.29: Perbandingan sebaran waktu Dataset 400.

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

	Tersaring	Keseluruhan
<i>min</i>	0.31442	1.56648
<i>lower fence</i>	0.31442	1.56648
q1	0.38489	1.68716
q2 (median)	0.45294	1.79097
q3	0.56089	1.90613
<i>upper fence</i>	0.77044	2.19699
<i>max</i>	1.21295	2.95327

Tabel 3.5: Nilai-nilai perbandingan waktu Dataset 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. Perbandingan nilai akurasi serta total waktu BSIS antara data yang tersaring dan data keseluruhan dapat dilihat pada Tabel 3.6.

	Total Waktu (s)	Akurasi (%)
Keseluruhan	183.79	99
Tersaring	49.25	98

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

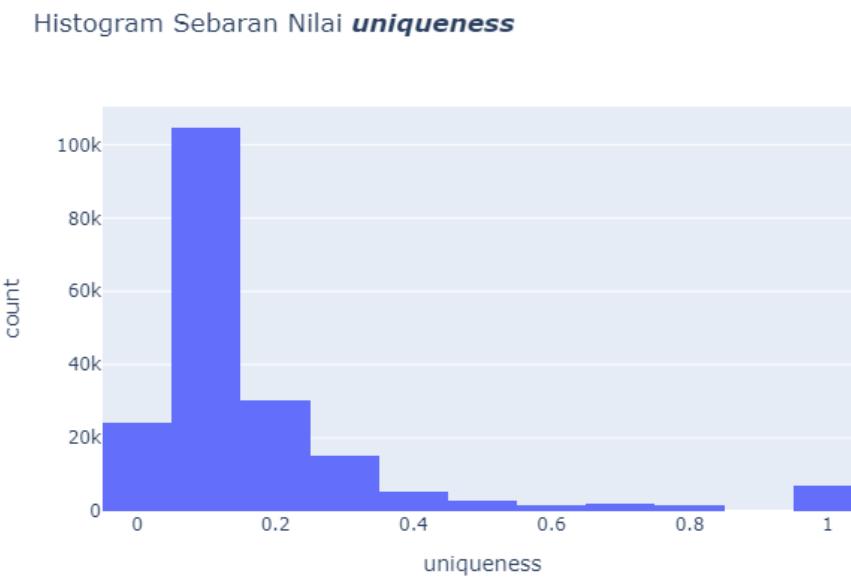
Dari data pada Tabel 3.6 didapat beberapa poin berikut:

- Ada penurunan sebanyak 73.15% pada total waktu dari *dataset* keseluruhan dan tersaring.
- Akurasi menurun sebanyak 1% dari *dataset* keseluruhan dan tersaring.

Terlihat bahwa pada pengujian dengan *dataset* ini penyaringan fitur lokal dengan menggunakan nilai konsistensi dan keunikan memberikan hasil yang cukup baik. Dengan menggunakan hanya sebagian dari *dataset* dapat meningkatkan waktu proses secara signifikan dengan tetap mendapat hasil akurasi yang tinggi.

Dataset 600

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



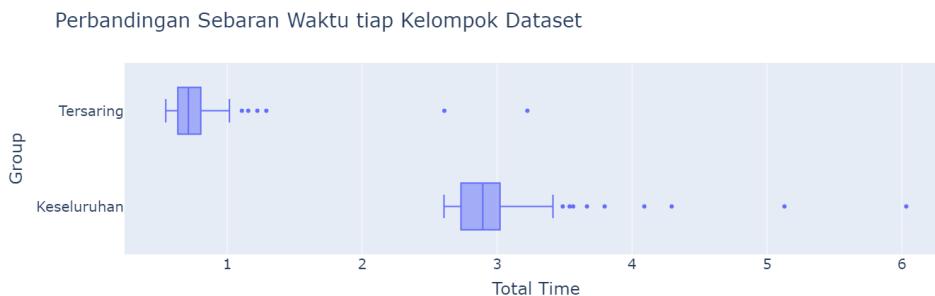
Gambar 3.30: Sebaran nilai keunikan (*uniqueness*) untuk Dataset 600.



Gambar 3.31: Sebaran nilai keunikan (*consistency*) untuk Dataset 600.

Sebaran nilai konsistensi dan keunikan untuk Dataset 600 memiliki ciri yang sama dengan sebaran untuk Dataset 400. Pada Dataset 600 ini akan digunakan nilai batas yang sama dengan Dataset 400 untuk menyaring nilai konsistensi dan keunikan. Nilai-nilai batas yang digunakan adalah 0.5 untuk konsistensi dan 0.2 untuk keunikan. Setelah dilakukan penyaringan dengan menggunakan batas-batas tersebut didapat sebanyak 43056 data. *Dataset* keseluruhan dan yang sudah tersaring tadi digunakan untuk melakukan identifikasi gambar-gambar di *dataset test*.

Perbedaan sebaran waktu yang digunakan untuk tiap gambar pada saat digunakan *dataset* secara keseluruhan dan yang telah tersaring dapat dilihat pada Gambar 3.32.



Gambar 3.32: Perbandingan sebaran waktu Dataset 600.

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

	Tersaring	Keseluruhan
<i>min</i>	0.54459	2.60656
<i>lower fence</i>	0.54459	2.60656
q1	0.63408	2.73240
q2 (median)	0.71225	2.89405
q3	0.80423	3.02188
<i>upper fence</i>	1.01644	3.41406
<i>max</i>	3.22431	6.03207

Tabel 3.7: Nilai-nilai perbandingan waktu Dataset 600.

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

	Total Waktu (s)	Akurasi (%)
Keseluruhan	299.23	99
Tersaring	78.64	93

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

Dari data pada Tabel 3.8 didapat beberapa poin berikut:

- Ada penurunan sebanyak 73.71% pada total waktu dari *dataset* keseluruhan dan tersaring.
- Akurasi menurun sebanyak 6% dari *dataset* keseluruhan dan tersaring.

Sama seperti pengujian pada Dataset 400, pada pengujian ini penyaringan *dataset* meningkatkan kecepatan waktu proses dengan cukup signifikan. Sedikit berbeda dengan Dataset 400, pada penggunaan Dataset 600 ini penurunan nilai akurasi lebih banyak. Nilai akurasi menurun sebanyak 6% dibandingkan pada Dataset 400 yang hanya menurun 1%.

Perbandingan Dataset 400 dan Dataset 600

Percobaan yang dilakukan dengan menggunakan Dataset 400 dan Dataset 600 memiliki perbedaan yang cukup terlihat dari sisi akurasi yang dihasilkan dan waktu proses yang diperlukan. Perbandingan tersebut dapat dilihat pada tabel berikut:

	Keseluruhan		Tersaring	
	Total Waktu (s)	Akurasi (%)	Total Waktu (s)	Akurasi (%)
Dataset 400	183.79	99	49.25	98
Dataset 600	299.23	99	78.64	93

Tabel di atas menunjukkan bahwa penggunaan Dataset 400 memerlukan waktu proses yang lebih cepat dibanding dengan Dataset 600. Perbedaan waktu proses yang diperlukan ini disebabkan karena Dataset 600 memiliki jumlah total data (fitur lokal) yang lebih banyak dibanding Dataset 400. Tetapi jika dilihat dari akurasi, Dataset 400 memiliki hasil yang lebih baik dibandingkan dengan Dataset 600 pada *dataset* yang telah tersaring. Padahal seharusnya ukuran gambar yang lebih besar memberikan hasil yang lebih baik karena terdapat lebih banyak titik yang dapat menjadi *keypoint*.

Pada saat ini belum diketahui kenapa Dataset 400 memiliki hasil akurasi yang lebih baik dibanding Dataset 600. Perkiraan sementara adalah karena gambar-gambar yang terdapat di *dataset test* memiliki ukuran yang kecil. Gambar-gambar *test* tersebut memiliki ukuran yang lebih

mirip ke Dataset 400 sehingga fitur-fitur lokal yang dimiliki oleh data *test* lebih mendekati fitur-fitur lokal dari Dataset 400.

3.9 Analisis Metode Ekstraksi Fitur Lokal ORB

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

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

- Tidak membuat objek SIFT melainkan membuat objek ORB dengan fungsi (`cv2.ORB_create()`).
- Tahap *pairing* pada BSIS dilakukan dengan menggunakan struktur data LSH.
- Pengujian dilakukan hanya pada *dataset* keseluruhan, tidak pada *dataset* yang telah tersaring seperti pada 3.8.

Dataset train yang digunakan pada analisis ini juga sama dengan analisis di 3.8. Ada dua variasi *dataset train* yang digunakan Dataset 400 dan Dataset 600. *Dataset test* yang digunakan juga sama seperti pada analisis sebelumnya.

3.9.1 Hasil Pengujian

Setelah dilakukan identifikasi dengan BSIS menggunakan fitur lokal ORB dihitung total waktu ekstraksi fitur dari seluruh gambar *test* dan akurasinya. Nilai total waktu dan akurasi tersebut kemudian dibandingkan dengan pada saat digunakan fitur lokal SIFT, yang merupakan hasil dari analisis pada 3.8. Hasilnya untuk masing-masing Dataset 400 dan Dataset 600 dapat dilihat pada Tabel 3.9 dan Tabel 3.10.

	Total Waktu Ekstrak (s)	Total Waktu BSIS (s)	Akurasi (%)
ORB	0.26	151.69	86
SIFT	1.75	183.79	99

Tabel 3.9: Perbandingan Waktu Proses dengan Akurasi Metode ORB dan SIFT pada Dataset 400

	Total Waktu Ekstrak (s)	Total Waktu BSIS (s)	Akurasi (%)
ORB	0.33	169.25	90
SIFT	2.54	299.23	99

Tabel 3.10: Perbandingan Waktu Proses dengan Akurasi Metode ORB dan SIFT pada Dataset 600

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

Dataset 400

- Dari penggunaan metode SIFT ke ORB terdapat penurunan total waktu ekstrak sebesar 85.14%, dan penurunan total waktu BSIS sebesar 17.46%.

- Metode ORB memiliki nilai akurasi yang lebih rendah sebesar 13% dari metode SIFT.

Dataset 600

- Dari penggunaan metode SIFT ke ORB terdapat penurunan total waktu ekstrak sebesar 87%, dan penurunan total waktu BSIS sebesar 43.43%.
- Metode ORB memiliki nilai akurasi yang lebih rendah sebesar 9% dari metode SIFT.

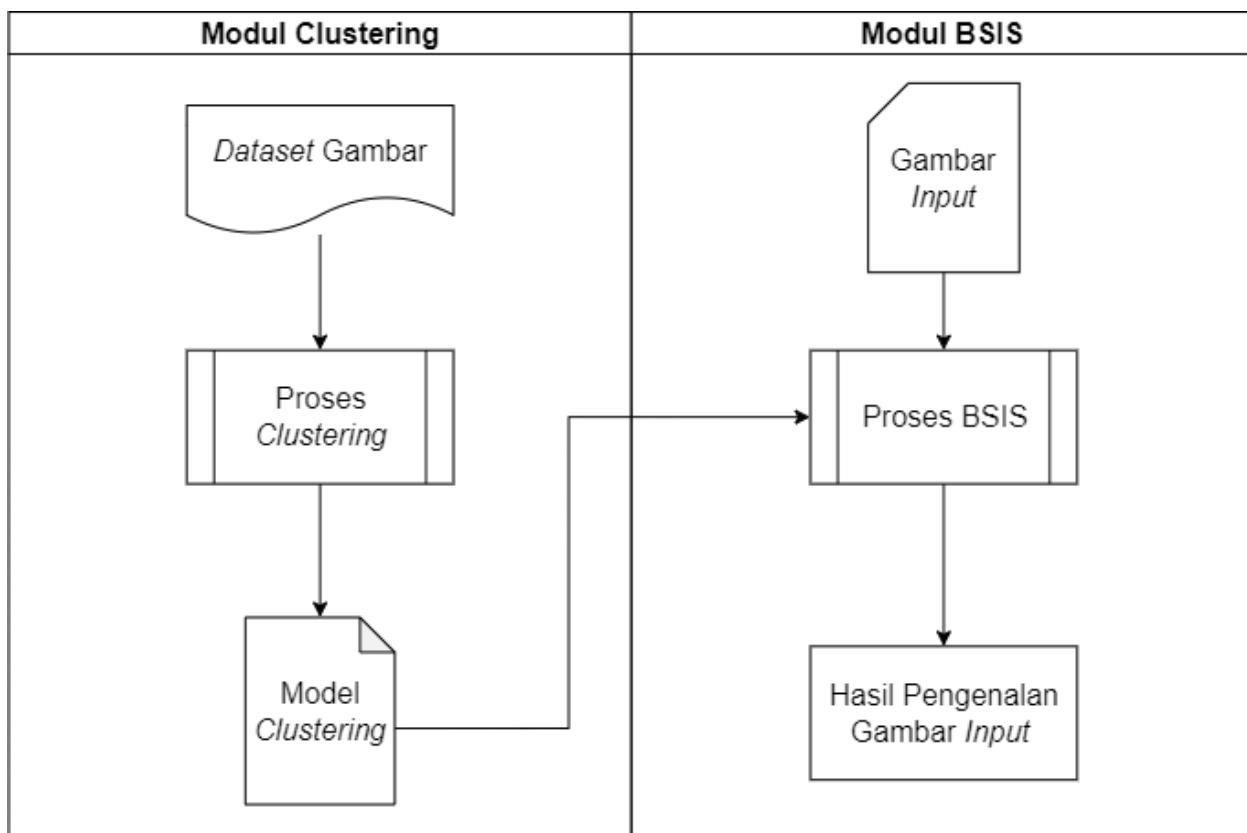
Penurunan yang lebih signifikan terjadi pada waktu yang diperlukan untuk melakukan ekstraksi fitur. Setelah telah didapat fitur lokal waktu proses yang diperlukan untuk BSIS dengan metode ORB lebih cepat dari SIFT tetapi perbedaannya tidak sebanyak perbedaan waktu ekstraksi.

BAB 4

PERANCANGAN

4.1 Rancangan Alur Program

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



Gambar 4.1: Rancangan alur program pada penelitian ini.

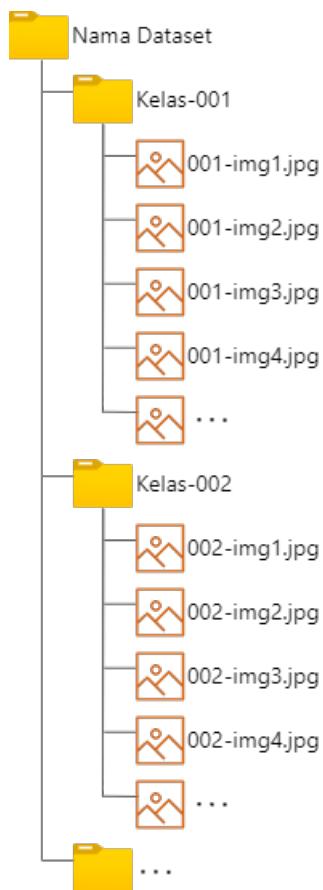
Berdasarkan dari diagram pada Gambar 4.1 tersebut program akan terbagi menjadi 2 modul. Modul pertama merupakan modul *clustering* yang implementasinya dijelaskan pada 4.2. Modul yang kedua adalah modul BSIS yang implementasinya dijelaskan pada 4.5. Selain itu untuk model yang dihasilkan oleh proses *clustering* dan digunakan untuk proses BSIS akan dijelaskan pada 4.3.

4.2 Rancangan Implementasi Metode Clustering Pembuatan Model

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

4.2.1 Rancangan Struktur Folder

Rancangan struktur *folder* yang diperlukan untuk dapat digunakan sebagai masukan pada modul *clustering* adalah sebagai berikut, dapat dilihat pada Gambar 4.2.

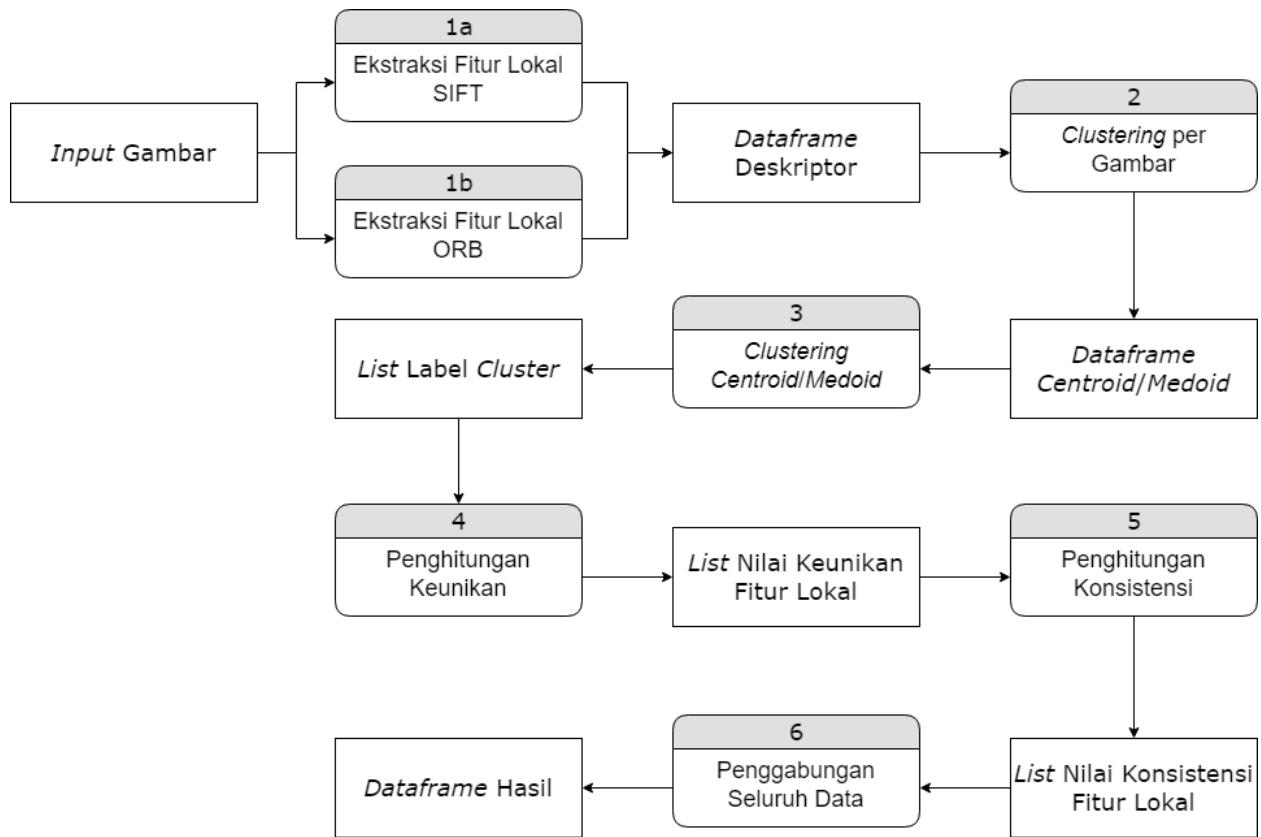


Gambar 4.2: Rancangan struktur *folder* untuk pembuatan model.

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

4.2.2 Rancangan Proses Clustering

Proses *clustering* menerima *dataset* gambar sesuai dengan struktur *folder* yang telah dijelaskan 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 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.

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

- Di langkah 2 dan 3 *clustering* untuk SIFT dilakukan dengan menggunakan metrik jarak Euclidean, sedangkan untuk ORB metrik jarak yang digunakan adalah Hamming.
- Di langkah 2 saat menggunakan SIFT dihitung *centroid* untuk setiap hasil *clustering* tiap gambar, sedangkan pada ORB yang dihitung adalah *medoid*.

Selain dari perbedaan-perbedaan tersebut langkah implementasi yang dilakukan sama, baik untuk metode ekstraksi fitur lokal SIFT maupun ORB. Proses penghitungan nilai keunikan dapat dilihat pada Pseudocode 3.

Pseudocode 3: 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

```

Sedangkan untuk penghitungan nilai konsistensi mengikuti proses pada Pseudocode 4.

Pseudocode 4: 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

```

Proses yang dilakukan pada Gambar 4.3 akan menghasilkan sebuah *dataframe* dengan ketentuan kolom seperti pada Tabel 4.1. *Dataframe* tersebut disimpan ke dalam *file pickle* dengan menggunakan *library* Pickle di Python.

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

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

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

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

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

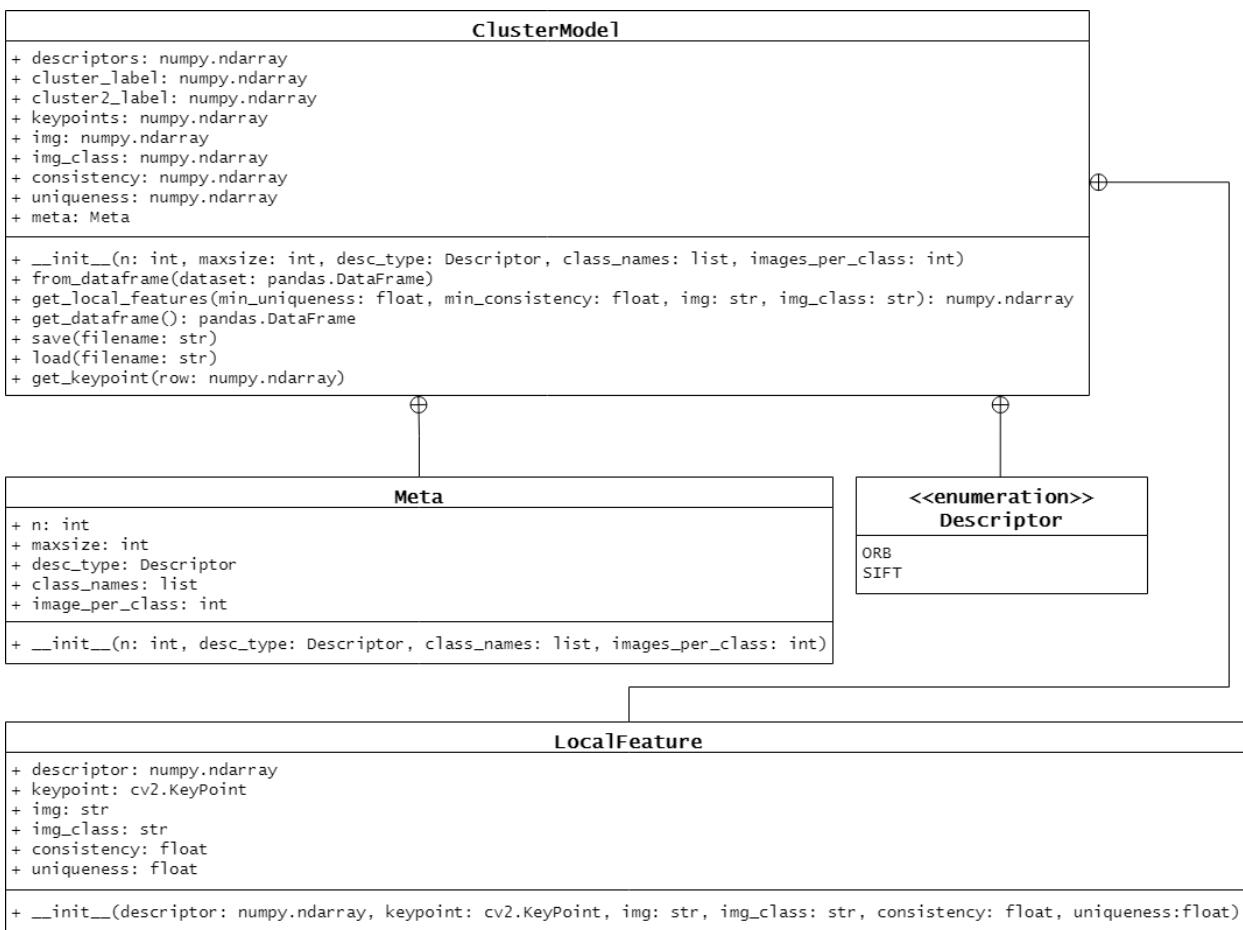
```
python clustering_sift.py --dir datasets/gsv --maxsize 600
```

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

Kelas `ClusterModel` sendiri memiliki beberapa fungsi penting sebagai berikut:

- `from_dataframe`: fungsi yang digunakan untuk membuat objek `ClusterModel` dari *dataframe* yang berisi kolom-kolom seperti pada Tabel 4.1.
- `get_local_features`: fungsi untuk mengembalikan sebuah *list* berisi objek `LocalFeature`. Fungsi ini digunakan untuk menghasilkan masukkan yang dapat digunakan oleh kelas BSIS.
- `get_dataframe`: fungsi untuk mengembalikan data yang disimpan dalam objek `ClusterModel` dalam bentuk *dataframe*. Fungsi ini berguna untuk melakukan analisis pada hasil proses *clustering* yang telah tersimpan sebagai objek `ClusterModel`.

4.4 Rancangan Kelas Util

Kelas *Util* berisi fungsi-fungsi pembantu yang digunakan pada kelas-kelas lainnya. Fungsi-fungsi ini berguna untuk mempermudah proses komputasi saat menggunakan kelas lainnya dan saat 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 + average_distance(dataset: pandas.DataFrame, metric: function, sample: int): float + orb_binary(desc: numpy.array): numpy.array + 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

Beberapa fungsi penting dari kelas *Util* adalah sebagai berikut:

1. **get_image**

Fungsi untuk memuat gambar dari *file*. Parameter **bw** menyatakan apakah gambar yang dimuat akan diubah menjadi format *grayscale*. Sedangkan untuk **maxwidth** dan **maxheight** digunakan untuk mengatur ukuran dari gambar.

2. **get_all_image**

Fungsi untuk memuat semua gambar dalam suatu *folder*. Fungsi ini memanggil fungsi *get_image* untuk semua *file* gambar dalam *folder* di *directory*.

3. **get_dataset**

Fungsi untuk memuat semua *file* gambar dengan fungsi *get_image* dari *file* yang tersusun sesuai struktur *folder* seperti yang dijelaskan pada 4.2.1. Fungsi ini digunakan untuk mendapatkan gambar masukkan pada 4.2.2.

4. **agglo_cluster**

Fungsi untuk melakukan *Agglomerative Clustering* pada *dataset*. Fungsi akan mengembalikan 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. **show_keypoints**

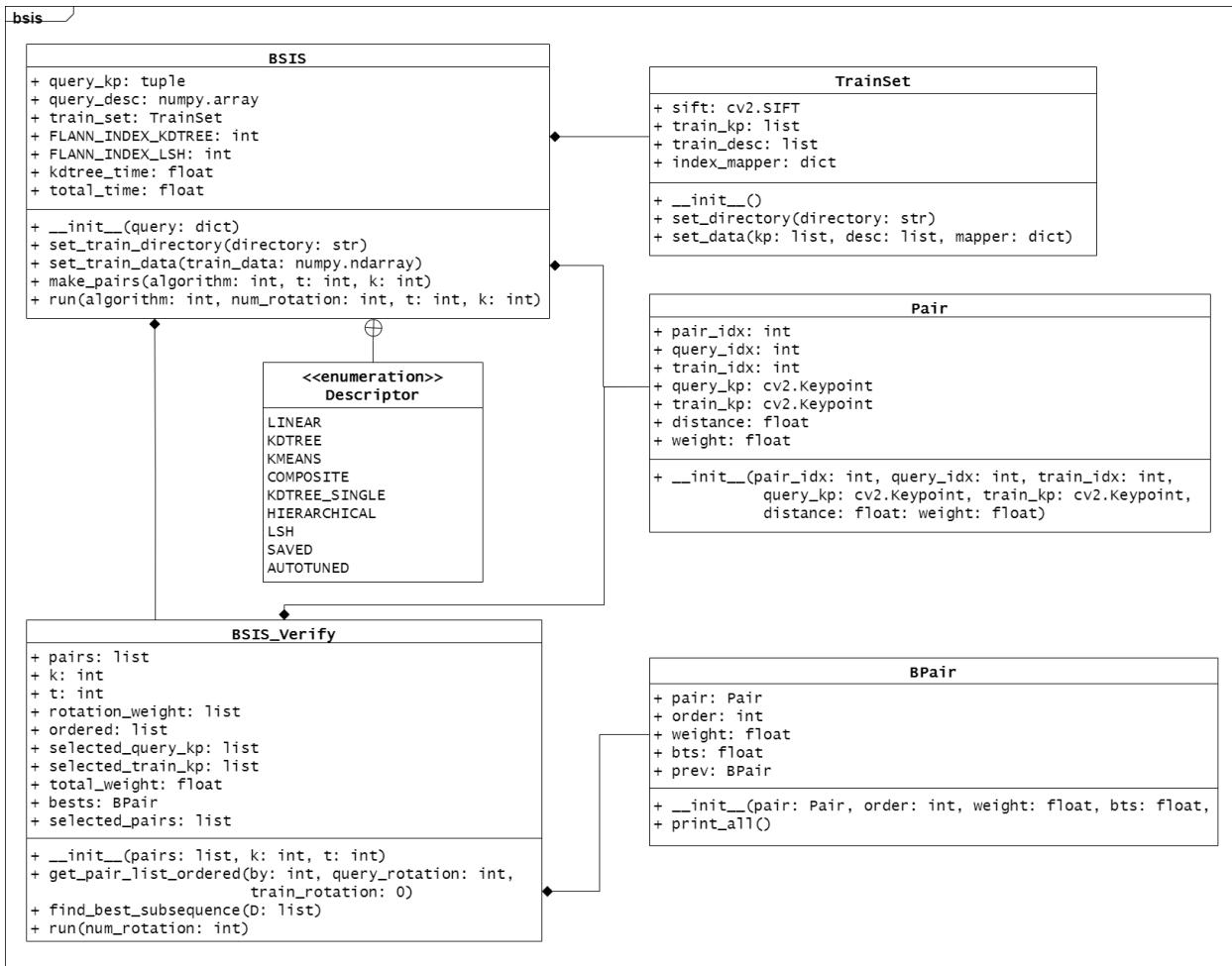
Fungsi untuk menampilkan *keypoint* pada gambar. Fungsi digunakan untuk melakukan visualisasi *keypoint* di gambar yang dilakukan pada 3.6.

6. **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 dengan memanggil kelas BSIS.



Gambar 4.6: Diagram Package `bsis`.

Untuk melakukan BSIS terhadap sebuah gambar pertama perlu dibuat sebuah objek kelas BSIS. Pembuatan objek BSIS menerima satu masukan yaitu sebuah *dictionary* yang berisi *list keypoint* dan *array* deskriptor. Setelah itu isi `train_set` dengan memanggil fungsi `set_train_data`. Fungsi `set_train_data` menerima *list* yang dihasilkan oleh fungsi `get_local_features` dari kelas `ClusterModel`. Setelah objek BSIS telah mendapatkan gambar masukan dan *dataset train* panggil fungsi `run` untuk menjalankan proses identifikasi. Setelah proses identifikasi selesai objek BSIS tersebut akan menyimpan hasil dari identifikasi.

Fungsi-fungsi lain dalam *package* tersebut dipanggil oleh kelas BSIS saat pemrosesan, kegunaan masing-masing kelas adalah sebagai berikut:

1. TrainSet

Kelas untuk mengatur *dataset* yang digunakan pada kelas BSIS.

2. Pair

Kelas untuk menyimpan pasangan fitur lokal. Fungsi `make_pairs_kdtree` dari BSIS mengembalikan sebuah *list* yang berisi objek `Pair`.

3. BPair

Kelas untuk menyimpan sebuah *sequence* `Pair`. Digunakan pada tahap verifikasi (`find_best_subsequence`) di `BSIS_Verify`. *Sequence* `Pair` diimplementasikan dalam bentuk objek `BPair` dalam objek `BPair`. Atribut `prev` dalam `BPair` akan diisi dengan objek `BPair` lain yang juga dapat memiliki objek `BPair` di atribut `prev`-nya.

4. BSIS_Verify

Kelas untuk melakukan verifikasi geometris dari pasangan fitur lokal yang dihasilkan oleh fungsi `make_pairs_kdtree` dari BSIS. Pada kelas ini pertama dilakukan pembuatan *list* 2

dimensi yang mengurutkan pasangan berdasarkan *order x* (urutan kemunculan *keypoint* di sumbu x) dari *keypoint* di gambar *train* yang dilakukan oleh fungsi `get_pair_list_ordered`. *List* yang dihasilkan oleh fungsi `get_pair_list_ordered` memiliki elemen yang merupakan *list* juga dengan panjang yang dapat berbeda-beda dan isinya merupakan objek *BPair* yang menyimpan informasi *Pair* dan atribut *prev*-nya kosong (berisi `None`). Setelah itu dilakukan verifikasi pada *list* tersebut dengan menggunakan fungsi `find_best_subsequence`. Proses verifikasi pada `find_best_subsequence` didefinisikan pada Pseudocode 5.

Pseudocode 5: FIND_BEST_SUBSEQUENCE

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: buat dictionary best_per_order.
3: for i=1 hingga LENGTH(D) do
4:   for j=1 hingga LENGTH(D[i]) do
5:     if ORDER(D[i][j]) tidak ada di best_per_order then
6:       isi best_per_order[ORDER(D[i][j])] dengan D[i][j]
7:     else if BTS(D[i][j]) lebih besar dari BTS(best_per_order[ORDER(D[i][j])]) then
8:       isi best_per_order[ORDER(D[i][j])] dengan D[i][j]
9:     end if
10:    objek BPair dBestPrev
11:    for ord=1 hingga ORDER(D[i][j]) do
12:      if ord ada di best_per_order then
13:        if BTS(best_per_order) lebih besar dari BTS(dBestPrev) then
14:          isi dBestPrev dengan best_per_order[ord]
15:        end if
16:      end if
17:    end for
18:    isi BTS(D[i][j]) dengan WEIGHT(D[i][j]) + BTS(dBestPrev)
19:    isi PREV(D[i][j]) dengan dBestPrev
20:    if BTS(D[i][j]) + WEIGHT(D[i][j]) > BTS(best_subsequence) +
WEIGHT(best_subsequence) then
21:      isi best_subsequence dengan D[i][j]
22:    end if
23:  end for
24: end for
25: return best_subsequence

```

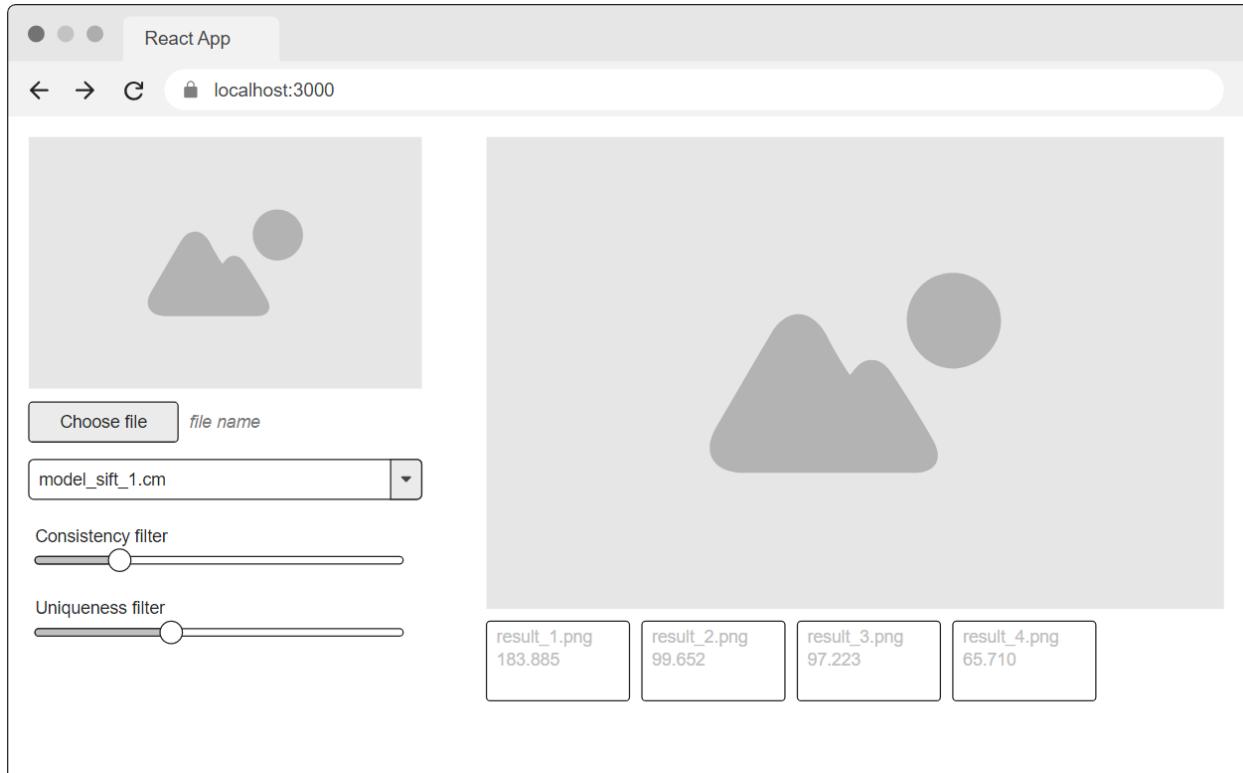
4.6 Rancangan Perangkat Lunak BSIS

Untuk lebih mudah melakukan BSIS pada sebuah gambar masukkan, dibuat sebuah perangkat lunak dengan GUI yang dapat melakukan identifikasi gambar masukkan dengan BSIS. Perangkat lunak yang dibuat memiliki fitur sebagai berikut:

- Memilih model yang ingin digunakan untuk identifikasi gambar masukkan.
- Memilih gambar masukkan yang ingin diidentifikasi.
- Menentukan nilai *threshold* untuk konsistensi dan keunikan.
- Melakukan identifikasi pada gambar yang telah dipilih, dengan menggunakan model serta

threshold yang telah ditentukan.

Saat dilakukan identifikasi pada gambar, perangkat lunak akan melakukan BSIS pada gambar masukkan dengan menggunakan *dataset* berdasarkan model dan nilai *threshold* yang telah dipilih. 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.

BAB 5

PENGUJIAN

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

5.1 Ide Analisis

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

Pada pengujian ini penyaringan fitur lokal akan dilakukan menggunakan dua cara. Penyaringan pertama dilakukan dengan menggunakan nilai *threshold* yang didapat dari hasil analisis di [3.7](#). Berdasarkan pada hasil analisis di [3.7](#), penyaringan pertama akan dilakukan dengan menggunakan nilai *threshold* 0.6 untuk konsistensi dan 0.7 untuk keunikan. Penyaringan kedua ditujukan untuk menggunakan nilai *threshold* yang lebih rendah sehingga dapat digunakan sebagai perbandingan. Untuk penyaringan kedua akan ditentukan *threshold* dengan cara yang sama dengan penentuan *threshold* pada analisis di [3.7](#).

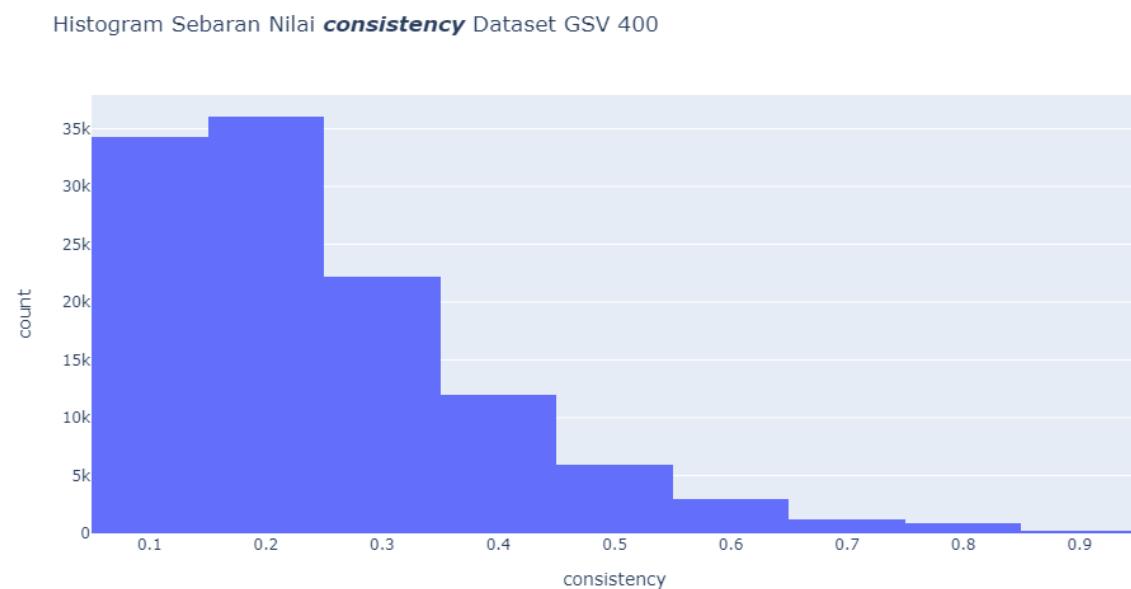
Pengujian ini akan dilakukan menggunakan *dataset* GSV dengan dua ukuran gambar yang berbeda sama seperti pada analisis di [3.8](#). Dua ukuran gambar yang digunakan akan dinamakan GSV 400 dan GSV 600, sesuai ukuran sisi terpanjang pada gambar-gambar yang digunakan. Berbeda dengan analisis pada [3.8](#),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

5.2 Penentuan Threshold dan Hasil Analisis

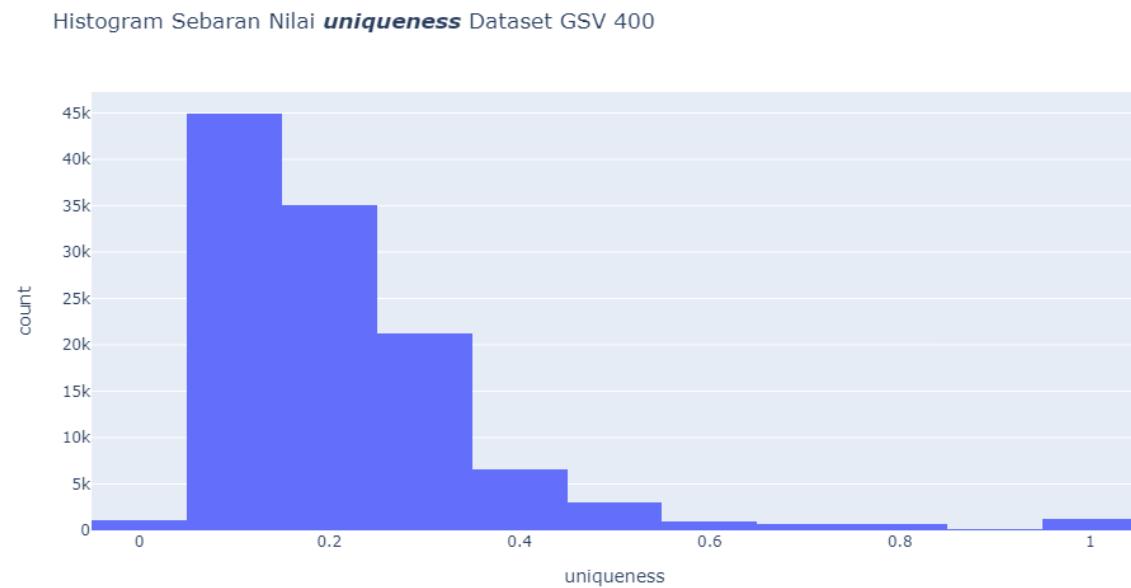
5.2.1 Metode SIFT

GSV 400

Sebaran nilai konsistensi dan keunikan untuk *dataset* GSV 400 dapat dilihat pada Gambar [5.1](#) dan Gambar [5.2](#).



Gambar 5.1: Histogram sebaran nilai keunikan pada *Dataset GSV 400*.



Gambar 5.2: Histogram sebaran nilai keunikan *Dataset GSV 400*.

Dengan melihat sebaran nilai konsistensi dan keunikan pada Gambar 5.1 dan Gambar 5.2 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 adalah sebagai berikut:

- Keseluruhan
 - konsistensi: 0.0
 - keunikan: 0.0
- Threshold 1

- konsistensi: 0.3
- keunikan: 0.3
- Threshold 2
 - konsistensi: 0.6
 - keunikan: 0.7

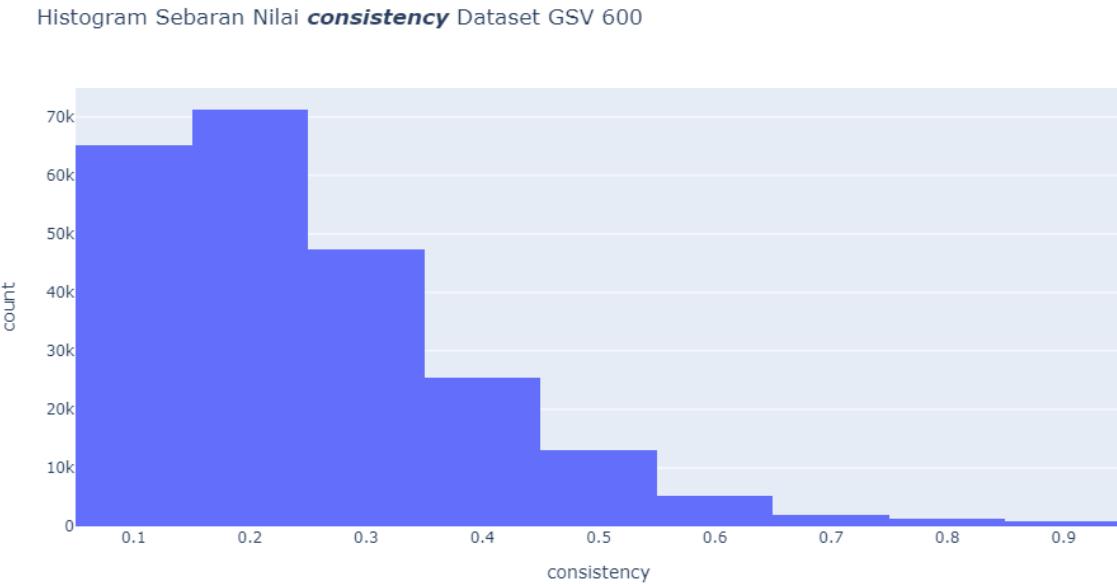
Telah dilakukan pengujian terhadap data *test* dengan menggunakan ketiga set nilai *threshold* tersebut. Hasil dari pengujian dapat dilihat pada Tabel 5.1

	Total Waktu Ekstrak (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	1.25	90.99	94
Threshold 1	1.29	24.30	84
Threshold 2	1.21	13.67	10

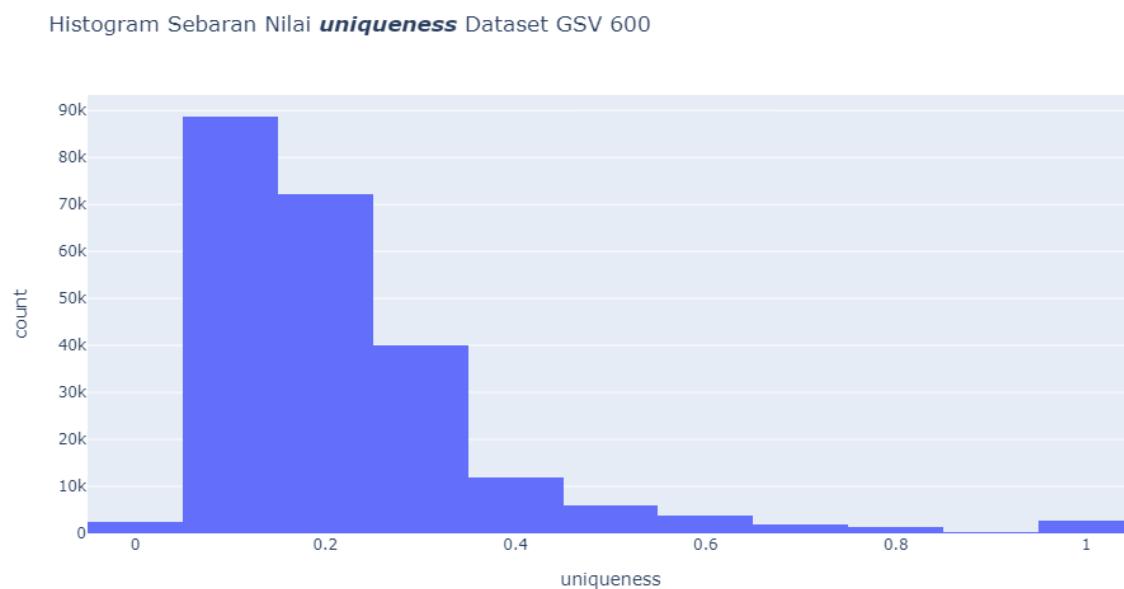
Tabel 5.1: Hasil pengujian pada *dataset GSV 400*.

GSV 600

Sebaran nilai konsistensi dan keunikan untuk *dataset GSV 600* dapat dilihat pada Gambar 5.3 dan Gambar 5.4.



Gambar 5.3: Histogram sebaran nilai keunikan pada *Dataset GSV 600*.



Gambar 5.4: Histogram sebaran nilai keunikan *Dataset GSV 600*.

Dengan menggunakan cara yang sama seperti sebelumnya, berdasarkan sebaran yang dapat dilihat pada Gambar 5.3 dan Gambar 5.4 maka akan digunakan nilai *threshold* senilai 0.3 untuk konsistensi dan 0.3 untuk keunikan. Tiga set *threshold* yang digunakan pada pengujian *dataset* ini adalah sebagai berikut:

- Keseluruhan
 - konsistensi: 0.0
 - keunikan: 0.0
- Threshold 1
 - konsistensi: 0.3
 - keunikan: 0.3
- Threshold 2
 - konsistensi: 0.6
 - keunikan: 0.7

Setelah dilakukan pengujian terhadap *dataset* GSV 600 dengan menggunakan tiga set *threshold* tersebut didapat hasil yang dapat dilihat pada Tabel 5.2.

	Total Waktu Ekstrak (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	2.69	187.64	100
Threshold 1	2.57	47.58	84
Threshold 2	2.75	20.55	16

Tabel 5.2: Hasil pengujian pada *dataset* GSV 600.

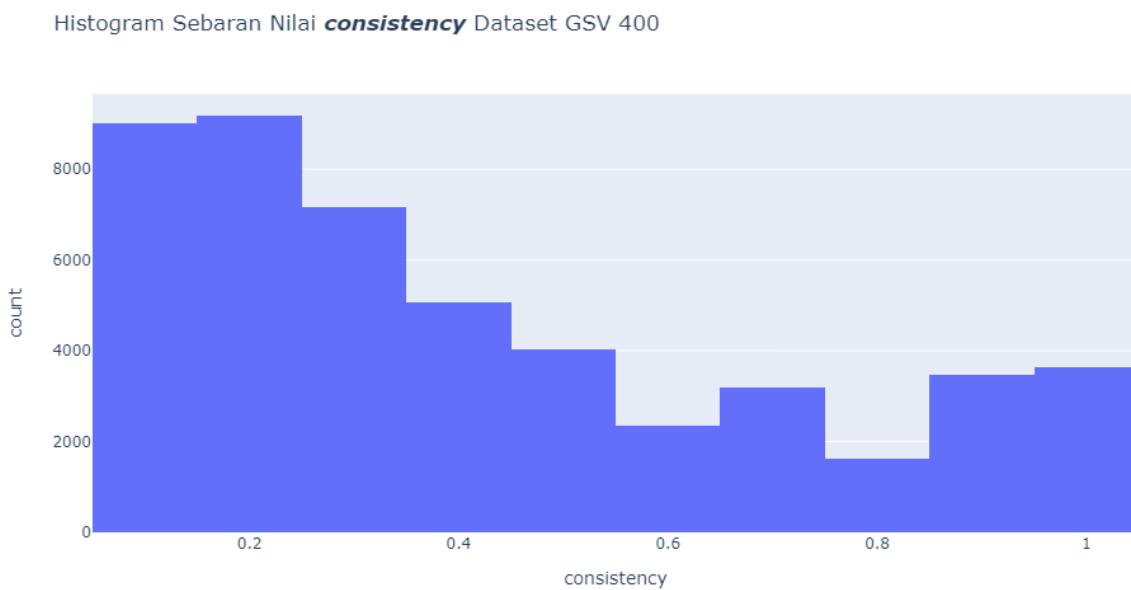
Hasil pada Tabel 5.1 dan Tabel 5.2 menunjukkan nilai *threshold* yang didapat dari hasil analisis pada *threshold* (3.7) memberikan hasil BSIS dengan akurasi yang sangat rendah. Cara penentuan *threshold* dengan melihat hasil pada gambar tersebut ternyata tidak efektif jika digunakan untuk melakukan 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.

5.2.2 Metode ORB

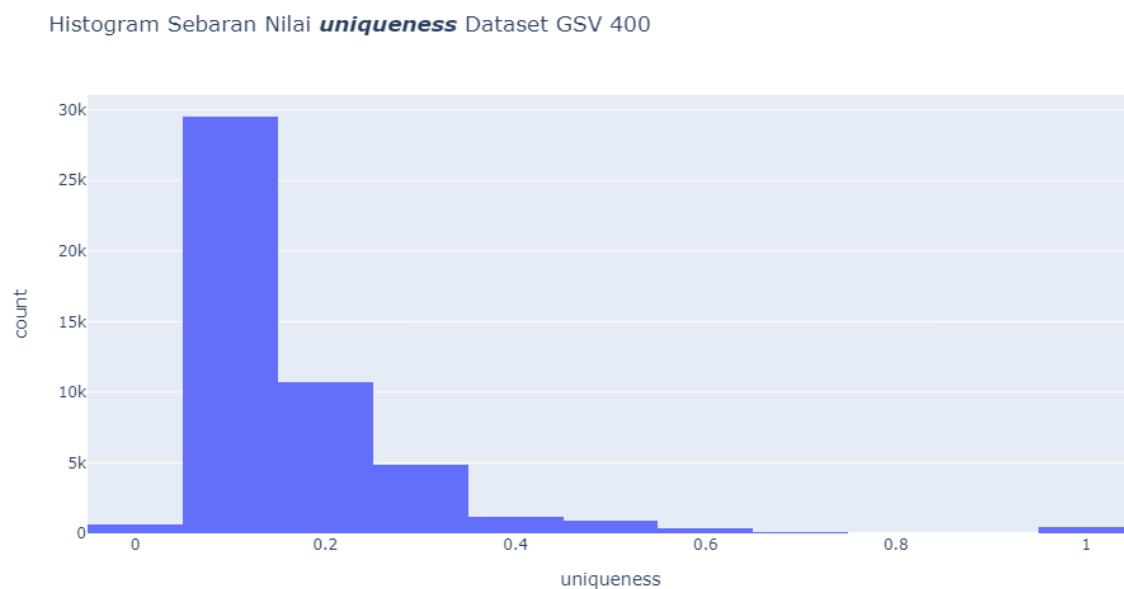
Pada pengujian dengan metode ORB ini hanya akan digunakan satu set *threshold*. *Threshold* yang digunakan adalah *threshold* yang didapat dari melihat hasil sebaran nilai konsistensi dan keunikan. Nilai *threshold* yang didapat dengan melihat hasilnya pada gambar seperti pada [3.7](#) tidak digunakan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

GSV 400

Sebaran nilai konsistensi dan keunikan untuk *dataset GSV 400* dapat dilihat pada Gambar [5.5](#) dan Gambar [5.6](#).



Gambar 5.5: Histogram sebaran nilai keunikan pada *Dataset GSV 400*.



Gambar 5.6: Histogram sebaran nilai keunikan *Dataset GSV 400*.

Dari kedua histogram pada Gambar 5.5 dan Gambar 5.6 ditetapkan nilai *threshold* untuk konsistensi senilai 0.3 dan untuk keunikan senilai 0.3. Dua set *threshold* yang digunakan pada pengujian ini adalah sebagai berikut:

- Keseluruhan
 - konsistensi: 0.0
 - keunikan: 0.0
- Threshold 1
 - konsistensi: 0.3
 - keunikan: 0.3

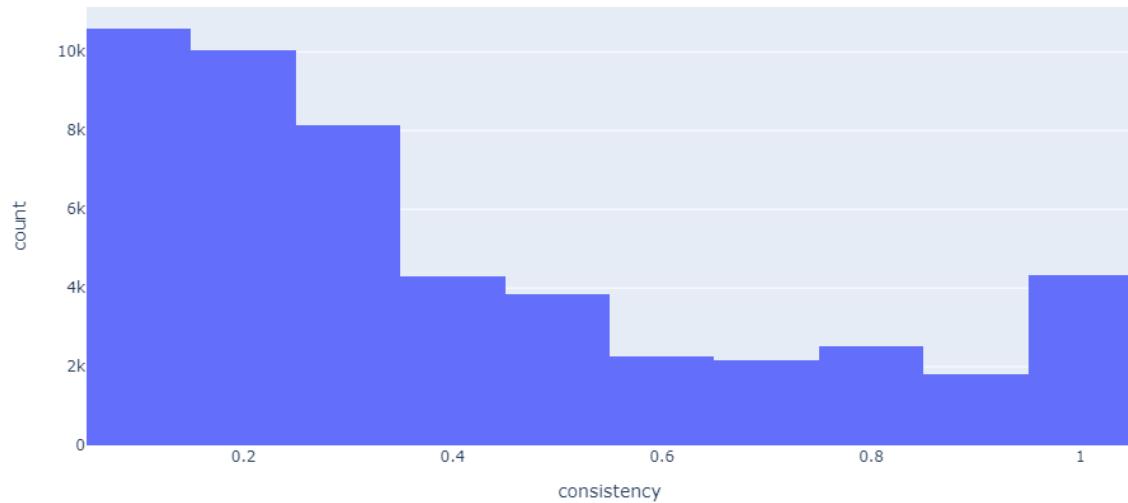
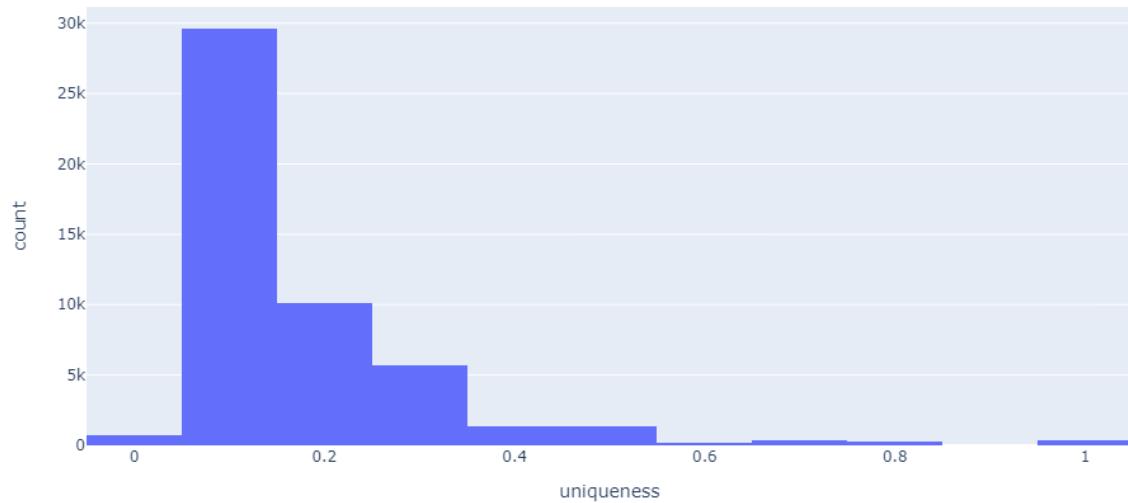
Setelah dilakukan pengujian terhadap *dataset* didapat hasil seperti yang dapat dilihat pada Tabel 5.3

	Total Waktu Ekstrak (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	0.17	51.12	76
Threshold 1	0.33	1.81	28

Tabel 5.3: Hasil pengujian pada *dataset* GSV 400.

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*.

Dapat dilihat dari kedua histogram di Gambar 5.7 dan Gambar 5.8 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 adalah sebagai berikut:

- Keseluruhan
 - konsistensi: 0.0
 - keunikan: 0.0
- Threshold 1

- konsistensi: 0.3
- keunikan: 0.3

Setelah dilakukan pengujian terhadap *dataset* didapat hasil seperti yang dapat dilihat pada Tabel 5.4

	Total Waktu Ekstrak (s)	Total Waktu BSIS(s)	Akurasi (%)
Keseluruhan	0.31	51.10	78
Threshold 1	0.29	2.99	20

Tabel 5.4: Hasil pengujian pada *dataset* GSV 600.

5.3 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.

5.3.1 GSV 400

		Total Waktu Ekstrak (s)	Total Waktu BSIS (s)	Akurasi (%)
SIFT	Keseluruhan	1.25	90.99	94
	Threshold 1	1.29	24.30	84
ORB	Keseluruhan	0.17	51.12	76
	Threshold 1	0.33	1.81	28

Tabel 5.5: Hasil perbandingan pengujian SIFT dan ORB pada *dataset* GSV 400.

Dari hasil pada Tabel 5.5 terlihat bahwa pada terdapat perbedaan hasil yang cukup signifikan antara SIFT dan ORB. Terlihat bahwa dari SIFT dan ORB terdapat penurunan waktu ekstraksi fitur yang cukup signifikan. Total waktu BSIS pada ORB juga menurun secara signifikan dibandingkan dengan SIFT, tetapi penurunan ini lebih dikarenakan jumlah fitur lokal yang dihasilkan ORB tidak sebanyak yang dihasilkan oleh SIFT. Nilai akurasi dari SIFT dan ORB juga mengalami penurunan yang signifikan. Akurasi dari ORB ada pada angka yang termasuk rendah terutama pada hasil yang telah disaring, hasil ini berbeda dari hasil yang didapat dari pengujian pada analisis di 3.9. Pada pengujian di analisis sebelumnya identifikasi dengan menggunakan metode ORB masih menghasilkan nilai akurasi yang cukup tinggi walaupun nilainya masih lebih kecil dari SIFT.

5.3.2 GSV 600

		Total Waktu Ekstrak (s)	Total Waktu BSIS (s)	Akurasi (%)
SIFT	Keseluruhan	2.69	187.64	100
	Threshold 1	2.57	47.58	84
ORB	Keseluruhan	0.31	51.10	78
	Threshold 1	0.29	2.99	20

Tabel 5.6: Hasil perbandingan pengujian SIFT dan ORB pada *dataset* GSV 600.

Hasil yang dapat dilihat pada Tabel 5.6 menunjukkan pola yang sama dari SIFT dan ORB, di mana SIFT memerlukan waktu yang lebih lama baik dari waktu ekstrak maupun waktu untuk

BSIS, tetapi juga menghasilkan akurasi yang lebih tinggi. Pada *dataset* ini waktu yang diperlukan baik untuk ekstrak maupun BSIS lebih tinggi dari waktu yang diperlukan *dataset* GSV 400. Waktu yang lebih lama tersebut dikarenakan ukuran gambar yang lebih besar sehingga lebih banyak fitur lokal yang dapat dihasilkan dari tahap ekstraksi.

BAB 6

KESIMPULAN

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.
- [6] Han, J., Pei, J., dan Kamber, M. (2011) *Data mining: concepts and techniques*. Elsevier.

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

```

76|     start_time = time.time()
77|     cv2.setRNGSeed(1311)
78|     matches = flann.knnMatch(self.query_desc, self.train_set.train_desc, k=k)
79|     self.pairing_time = time.time() - start_time
80|     print('make_pairs---', self.pairing_time)
81|
82|     pairs = dict()
83|     pair_idx = 0
84|     for m in matches:
85|         m_dists = list()
86|         for i in m:
87|             m_dists.append(i.distance)
88|
89|         st_dev = np.std(m_dists)
90|         mean = np.mean(m_dists)
91|         thres = mean - (t * st_dev)
92|         self.thres_ = thres
93|
94|         for i in m:
95|             if i.distance < thres:
96|                 weight = ((i.distance - mean) / st_dev) ** 2
97|                 if self.train_set.index_mapper[i.trainIdx] not in pairs.keys():
98|                     pairs[self.train_set.index_mapper[i.trainIdx]] = list()
99|
100|                pairs[self.train_set.index_mapper[i.trainIdx]].append(
101|                    Pair(
102|                        pair_idx,
103|                        i.queryIdx,
104|                        i.trainIdx,
105|                        self.query_kp[i.queryIdx],
106|                        self.train_set.train_kp[i.trainIdx],
107|                        i.distance,
108|                        weight
109|                    )
110|                )
111|                pair_idx += 1
112|
113|     return pairs
114|
115| def make_pairs_scipy(self, t=4, k=100):
116|     start_time = time.time()
117|     tree = KDTree(self.train_set.train_desc)
118|     distance, index = tree.query(self.query_desc, k=k, workers=-1)
119|     self.pairing_time = time.time() - start_time
120|     print('make_pairs---', self.pairing_time)
121|
122|     pairs = dict()
123|     pair_idx = 0
124|
125|     for queryIdx, (dist, idx) in enumerate(zip(distance, index)):
126|         st_dev = np.std(dist)
127|         mean = np.mean(dist)
128|         thres = mean - (t * st_dev)
129|         self.thres_ = thres
130|
131|         for d, trainIdx in zip(dist, idx):
132|             if d < thres:
133|                 weight = ((d - mean) / st_dev) ** 2
134|                 if self.train_set.index_mapper[trainIdx] not in pairs.keys():
135|                     pairs[self.train_set.index_mapper[trainIdx]] = list()
136|
137|                 pairs[self.train_set.index_mapper[trainIdx]].append(
138|                     Pair(
139|                         pair_idx,
140|                         queryIdx,
141|                         trainIdx,
142|                         self.query_kp[queryIdx],
143|                         self.train_set.train_kp[trainIdx],
144|                         d,
145|                         weight
146|                     )
147|                 )
148|                 pair_idx += 1
149|
150|     return pairs
151|
152| def run_scipy(self, num_rotation=1, t=4, k=100):
153|     start_time = time.time()
154|     self.pairs = self.make_pairs_scipy(t=t, k=k)
155|     self.result = dict()
156|
157|     maximum = ('', 0)
158|     for k, v in self.pairs.items():
159|         bsis = BSIS_Verify(v)
160|         bsis.run(num_rotation=num_rotation)
161|         self.result[k] = {'query_kp': bsis.selected_query_kp, 'train_kp': bsis.selected_train_kp, 'total_weight': bsis.
162|                         total_weight}
163|         if bsis.total_weight > maximum[1]:
164|             maximum = (k, bsis.total_weight)
165|
166|     self.total_time = time.time() - start_time
167|     print('total_time---', self.total_time)
168|     return maximum[0]
169|
170| def run(self, algorithm: int, num_rotation=1, t=4, k=100):
171|     start_time = time.time()
172|     self.pairs = self.make_pairs(algorithm=algorithm, t=t, k=k)
173|     self.result = dict()

```

```

174     maximum = ('', 0)
175     for k, v in self.pairs.items():
176         bsis = BSIS_Verify(v)
177         bsis.run(num_rotation=num_rotation)
178         self.result[k] = {'query_kp': bsis.selected_query_kp, 'train_kp': bsis.selected_train_kp, 'total_weight': bsis.
179                         total_weight}
180         if bsis.total_weight > maximum[1]:
181             maximum = (k, bsis.total_weight)
182
183     self.total_time = time.time() - start_time
184     print('total_time---', self.total_time)
185     return maximum[0]
186
187 #%%
188 class TrainSet:
189     def __init__(self):
190         self.sift = cv2.xfeatures2d.SIFT_create()
191
192         self.train_kp = list()
193         self.train_desc = list()
194         self.index_mapper = dict()
195
196     def set_directory(self, directory):
197         index = 0
198         for filename in glob(os.path.join(directory, '*.jpg')):
199             fname = filename.split('\\')[-1]
200             kp, desc = self.sift.detectAndCompute(util.get_image(filename), None)
201             for k, d in zip(kp, desc):
202                 self.train_kp.append(k)
203                 self.train_desc.append(d)
204                 self.index_mapper[index] = fname
205                 index += 1
206
207         self.train_desc = np.array(self.train_desc)
208
209     def set_data(self, kp, desc, mapper):
210         self.train_kp = kp
211         self.train_desc = desc
212         self.index_mapper = mapper
213
214 #Class untuk menyimpan pasangan antar keypoint
215 class Pair:
216     def __init__(self, pair_idx, query_idx, train_idx, query_kp, train_kp, distance, weight):
217         self.pair_idx = pair_idx
218         self.query_idx = query_idx
219         self.train_idx = train_idx
220         self.query_kp = query_kp
221         self.train_kp = train_kp
222         self.distance = distance
223         self.weight = weight
224
225     def print_all(self):
226         string = f'pair_idx:{self.pair_idx}\nquery_idx:{self.query_idx}\ntrain_idx:{self.train_idx}\ndistance:{self.
227             distance}\nweight:{self.weight}\n'
228         print(string)
229
230 #Class untuk menyimpan informasi pasangan keypoint yang diperlukan pada tahap verifikasi BSIS
231 class BPair:
232     def __init__(self, pair, order, weight, bts, prev):
233         self.pair = pair
234         self.order = order
235         self.weight = weight
236         self.bts = bts
237         self.prev = prev
238
239     def print_all(self):
240         try:
241             print(self.pair.pair_idx, self.order, self.weight, self.bts, self.prev.idx)
242         except:
243             print(self.pair.pair_idx, self.order, self.weight, self.bts, None)
244
245 #%%
246 class BSIS_Verify:
247     def __init__(self, pairs, k=100, t=4):
248         self.pairs = pairs
249
250         self.k = k
251         self.t = t
252
253     def get_pair_list_ordered(self, by=0, query_rotation=0, train_rotation=0):
254         paired_query_kp = set([(x.query_idx, x.query_kp.pt[0], x.query_kp.pt[1]) for x in self.pairs])
255         query_origin = (np.mean([i[1] for i in paired_query_kp]), np.mean([i[2] for i in paired_query_kp]))
256         query_order_x_set = [(i[0], util.rotate((i[1], i[2]), query_origin, query_rotation)[by]) for i in paired_query_kp]
257         query_order_x_set = sorted(query_order_x_set, key=itemgetter(1))
258         query_order_x = dict()
259         for i, idx in enumerate(query_order_x_set):
260             query_order_x[idx[0]] = i
261
262         paired_train_kp = set([(x.train_idx, x.train_kp.pt[0], x.train_kp.pt[1]) for x in self.pairs])
263         train_origin = (np.mean([i[1] for i in paired_train_kp]), np.mean([i[2] for i in paired.train_kp]))
264         train_order_x_set = [(i[0], util.rotate((i[1], i[2]), train_origin, train_rotation)[by]) for i in paired_train_kp]
265         train_order_x_set = sorted(train_order_x_set, key=itemgetter(1))
266         # train_order_x = dict()
267         # for i, idx in enumerate(train_order_x_set):
268         #     train_order_x[idx[0]] = i
269
270         train_idx_dict = dict()
271         for p in self.pairs:
272             if p.train_idx not in train_idx_dict.keys():

```

```

271         train_idx_dict[p.train_idx] = list()
272         train_idx_dict[p.train_idx].append(p)
273     else:
274         train_idx_dict[p.train_idx].append(p)
275
276     ordered_list = list()
277     for i, j in train_order_x_set:
278         one_kp_pairs = train_idx_dict[i]
279         col_list = list()
280         for p in one_kp_pairs:
281             col_list.append(BPair(p, query_order_x[p.query_idx], p.weight, p.weight, None))
282         ordered_list.append(col_list)
283
284     return ordered_list
285
286 def find_best_subsequence(self, D):
287     best_subsequence = BPair(0, 0, 0, 0, None)
288     #dictionary menyimpan BPair dengan bts terbaik untuk tiap order
289     best_per_order = dict()
290
291     #iterasi kolom
292     for i in range(1, len(D)):
293         #iterasi baris pada tiap kolom
294         for j in range(0, len(D[i])):
295
296             if D[i][j].order not in best_per_order.keys():
297                 best_per_order[D[i][j].order] = D[i][j]
298             else:
299                 if D[i][j].bts > best_per_order[D[i][j].order].bts:
300                     best_per_order[D[i][j].order] = D[i][j]
301
302             dBestPrev = BPair(0, 0, 0, 0, None)
303             for ord in range(0, D[i][j].order):
304                 if ord in best_per_order.keys():
305                     if best_per_order[ord].bts > dBestPrev.bts:
306                         dBestPrev = best_per_order[ord]
307
308             D[i][j].bts = D[i][j].weight + dBestPrev.bts
309             D[i][j].prev = dBestPrev
310
311             if D[i][j].bts + D[i][j].weight > best_subsequence.bts + best_subsequence.weight:
312                 best_subsequence = D[i][j]
313
314     return best_subsequence
315
316 def run(self, num_rotation=1):
317     max_weight = 0
318     selected_query_kp = list()
319     selected_train_kp = list()
320     self.rotation_weight = list()
321     rotate_by = int(360.0 / num_rotation)
322     for i in range(0, 360, rotate_by):
323         query_rotation = i
324         train_rotation = 0
325         #print('query_rotation: ', query_rotation, '---', 'train_rotation:', train_rotation)
326
327         get_pair_list_start = time.time()
328         self.ordered = self.get_pair_list_ordered(query_rotation=query_rotation, train_rotation=train_rotation)
329
330         if len(self.ordered) < 1:
331             self.selected_query_kp = []
332             self.selected_train_kp = []
333             self.total_weight = 0
334             continue
335         #print('get_pair_list_ordered ---', time.time() - get_pair_list_start)
336
337         find_best_start = time.time()
338         self.bests = self.find_best_subsequence(self.ordered)
339         self.selected_pairs = list()
340         while(self.bests.prev != None):
341             self.selected_pairs.append(self.bests.pair)
342             self.bests = self.bests.prev
343         else:
344             pass
345             #self.selected_pairs.append(self.bests.pair)
346         #print('find_best_subsequence ---', time.time() - find_best_start)
347
348         weight = sum([i.weight for i in self.selected_pairs])
349         self.rotation_weight.append((i, weight))
350         if weight > max_weight:
351             max_weight = weight
352             selected_query_kp = [i.query_kp for i in self.selected_pairs]
353             selected_train_kp = [i.train_kp for i in self.selected_pairs]
354
355         self.selected_query_kp = selected_query_kp
356         self.selected_train_kp = selected_train_kp
357         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, desc_type, class_names, image_per_class):
11             self.n = n
12             self.desc_type = desc_type
13             self.class_names = class_names
14             self.image_per_class = image_per_class
15
16     class Descriptor(Enum):
17         ORB = 'ORB'
18         SIFT = 'SIFT'
19
20     class LocalFeature:
21         def __init__(self, descriptor, keypoint, img, img_class, consistency, uniqueness):
22             self.descriptor = descriptor
23             self.keypoint = keypoint
24             self.img = img
25             self.img_class = img_class
26             self.consistency = consistency
27             self.uniqueness = uniqueness
28
29     def __init__(self, n, desc_type, class_names, images_per_class):
30         self.meta = self.Meta(
31             n=n,
32             desc_type=desc_type,
33             class_names=class_names,
34             image_per_class=images_per_class
35         )
36
37     def set_data(self, **kwargs):
38         if 'dataframe' in kwargs:
39             self.from_dataframe(kwargs.get('dataframe'))
40
41
42     def from_dataframe(self, dataset):
43         #get descriptor array
44         if self.meta.desc_type == self.Descriptor.SIFT:
45             self.descriptors = dataset.iloc[:, :128].values
46         elif self.meta.desc_type == self.Descriptor.ORB:
47             self.descriptors = dataset.iloc[:, :32].values
48
49         #get cluster labels
50         self.cluster_label = dataset.cluster_label.values
51         self.cluster2_label = dataset.cluster2_label.values
52
53         #get keypoints array
54         keypoint_columns = ['point_x', 'point_y', 'size', 'angle', 'response', 'octave', 'class_id']
55         self.keypoints = dataset.loc[:, keypoint_columns].values
56
57         #get image details
58         self.img = dataset.img.values
59         self.img_class = dataset.img_class.values
60
61         #get consistency and uniqueness
62         self.consistency = dataset.consistency.values
63         self.uniqueness = dataset.uniqueness.values
64
65     def get_local_features(self, min_uniqueness=0.0, min_consistency=0.0, img=None, img_class=None):
66         cons_mask = self.consistency >= min_consistency
67         uniq_mask = self.uniqueness >= min_uniqueness
68
69         if img:
70             img_mask = self.img == img
71         else:
72             img_mask = np.full(self.img.shape[0], True)
73         if img_class:
74             img_class_mask = self.img_class == img_class
75         else:
76             img_class_mask = np.full(self.img_class.shape[0], True)
77
78         mask = cons_mask & uniq_mask & img_mask & img_class_mask
79
80         filtered_descriptors = self.descriptors[mask]
81         filtered_keypoints = np.apply_along_axis(self.get_keypoint, axis=1, arr=self.keypoints[mask])
82         filtered_img = self.img[mask]
83         filtered_img_class = self.img_class[mask]
84         filtered_consistency = self.consistency[mask]
85         filtered_uniqueness = self.uniqueness[mask]
86
87         filtered_columns = zip(filtered_descriptors,
88                                filtered_keypoints,
89                                filtered_img,
90                                filtered_img_class,
91                                filtered_consistency,
92                                filtered_uniqueness)
93
94         local_features = []
95         for desc, kp, img, img_class, cons, uniq in filtered_columns:
96             local_features.append(self.LocalFeature(desc, kp, img, img_class, cons, uniq))
97
98         return np.array(local_features, dtype=self.LocalFeature)
99
100    def get_dataframe(self):
101        df = pd.DataFrame(self.descriptors)
102        df['img'] = self.img
103        df['img_class'] = self.img_class
104        df['cluster_label'] = self.cluster_label
105        df['cluster2_label'] = self.cluster2_label
106        df['uniqueness'] = self.uniqueness

```

```

107     df['consistency'] = self.consistency
108
109     return df
110
111 def save(self, filename):
112     with open(filename, 'wb') as file:
113         pickle.dump(self.__dict__, file, 2)
114
115 def load(self, filename):
116     with open(filename, 'rb') as file:
117         class_dict = pickle.load(file)
118         self.__dict__.update(class_dict)
119
120 def get_keypoint(self, row):
121     keypoint = cv2.KeyPoint(
122         x=float(row[0]),
123         y=float(row[1]),
124         size=float(row[2]),
125         angle=float(row[3]),
126         response=float(row[4]),
127         octave=int(row[5]),
128         class_id=int(row[5])
129     )
130
131     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')

```

```

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.agglo_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.agglo_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 keypoint 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     for k in descriptors_.kp_idx:
188         kp = keypoints[k]
189         point_x.append(kp.pt[0])
190         point_y.append(kp.pt[1])
191         size.append(kp.size)
192         angle.append(kp.angle)
193         response.append(kp.response)
194         octave.append(kp.octave)
195         class_id.append(kp.class_id)
196     descriptors_[‘point_x’] = point_x
197     descriptors_[‘point_y’] = point_y
198     descriptors_[‘size’] = size
199     descriptors_[‘angle’] = angle
200     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: 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 = []
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 = {}
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] = []
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 dinginkan
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_affinity,
144                 linkage='complete'
145             ).fit(array_dataset)
146         elif affinity == 'hamming2':
147             aggro_model = AgglomerativeClustering(
148                 n_clusters=n_clusters,
149                 distance_threshold=distance_threshold,
150                 affinity=hamming_affinity2,
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_int2(point1, point2):
204     total = 0
205     for i, j in zip(point1, point2):
206         total += countSetBits(np.bitwise_xor(int(i), int(j)))
207
208     return total
209

```

```

210| def hamming_int(point1, point2):
211|     arr = np.array([point1, point2])
212|     return np.sum(np.apply_along_axis(hamming_dist, 0, arr))
213|
214| def hamming_bin(point1, point2):
215|     return hamming(point1, point2) * len(point1)
216|
217| def hamming_affinity(X):
218|     return pairwise_distances(X, metric=hamming_bin)
219|
220| def hamming_affinity2(X):
221|     return pairwise_distances(X, metric=hamming_int2)
222|
223| def average_distance(dataset, metric, sample=None):
224|     """
225|         Fungsi untuk menghitung jarak euclidean rata-rata dari tiap elemen di dataset
226|
227|     Parameter:
228|         dataset : data yang akan dihitung rata-rata jaraknya
229|         sample : jumlah sampel yang akan digunakan untuk menghitung rata-rata. Jika 'None' maka semua data digunakan
230|     """
231|     sample_df = dataset
232|     if sample != None:
233|         sample_df = dataset.sample(sample)
234|
235|     dist = 0
236|     m = 0
237|     while m < len(sample_df.index) - 1:
238|         n = m + 1
239|         while n < len(sample_df.index):
240|             d = metric(sample_df.iloc[m], sample_df.iloc[n])
241|             dist = dist + d
242|             n = n + 1
243|         m = m + 1
244|     combination = len(list(combinations(sample_df.index, 2)))
245|     return dist / combination
246|
247| def combination_distance(dataset, metric, sample=None):
248|     sample_df = dataset
249|     if sample != None:
250|         sample_df = dataset.sample(sample).values
251|
252|     dists = np.array([metric(p1, p2) for p1, p2 in combinations(sample_df, 2)])
253|     return np.sum(dists) / dists.shape[0]
254|
255| def chunks(lst, n):
256|     """Yield successive n-sized chunks from lst."""
257|     for i in range(0, len(lst), n):
258|         yield lst[i:i + n]
259|
260| def int_to_binary(desc):
261|     """
262|         Fungsi untuk mengubah descriptor orb yang dihasilkan opencv (32bit integer) menjadi
263|         format 256 bit binary
264|
265|     Parameter:
266|         desc : array berukuran i x 32 berisi integer
267|     """
268|     binary_desc = []
269|     for row in desc:
270|         binary_row = []
271|         for num in row:
272|             binary = '{0:08b}'.format(num)
273|             for i in binary:
274|                 binary_row.append(int(i))
275|         binary_desc.append(binary_row)
276|
277|     binary_desc = np.array(binary_desc, dtype='uint8')
278|
279|     return binary_desc
280|
281| def binary_to_int(desc):
282|     int_desc = []
283|     for row in desc:
284|         int_row = []
285|         chunked_row = chunks(row, 8)
286|         for c in chunked_row:
287|             res = 0
288|             for ele in c:
289|                 res = (res << 1) | ele
290|             int_row.append(res)
291|         int_desc.append(int_row)
292|
293|     int_desc = np.array(int_desc, dtype='uint8')
294|
295|     return int_desc
296|
297| def medoid(arr, metric, get_index=False):
298|     """
299|         Fungsi untuk mencari medoid dari array berukuran n x n
300|
301|     Parameter:
302|         desc : array 2 dimensi
303|         metric : fungsi jarak yang digunakan
304|     """
305|     dists = pairwise_distances(arr, metric=metric)
306|
307|     max = np.zeros(dists.shape[0])
308|     for i, v in enumerate(dists):

```

```

309     max[i] = np.sum(v)
310
311 medoid = arr[np.argmin(max)]
312
313 if get_index:
314     return np.argmin(max)
315 else:
316     return np.array([medoid])
317
318 def rotate(p, origin=(0, 0), degrees=0):
319     """
320     Fungsi untuk merotasi suatu koordinat berdasarkan suatu titik pusat
321
322     Parameter:
323         p      : koordinat yang akan dirotasi
324         origin : titik pusat untuk melakukan rotasi
325         degrees : jumlah seberapa besar rotasi
326     """
327 angle = np.deg2rad(degrees)
328 R = np.array([[np.cos(angle), -np.sin(angle)],
329               [np.sin(angle), np.cos(angle)]])
330 o = np.atleast_2d(origin)
331 p = np.atleast_2d(p)
332
333 return np.squeeze((R @ (p.T-o.T) + o.T).T)
334
335 def rgb_to_bgr(r, g, b):
336     return (b, g, r)
337
338 def show_keypoints(image, keypoints, color=(0,255,0), flags=cv2.DRAW_MATCHES_FLAGS_DEFAULT, name=False, return_img=False):
339     """
340     Fungsi untuk melakukan visualisasi keypoint pada gambar
341
342     Parameter:
343         image      : gambar yang digunakan untuk menampilkan
344         keypoints : tuple berisi keypoint
345         color     : warna keypoint yang ditampilkan
346         name      : nama file jika gambar disimpan
347     """
348 keypoints_image = cv2.drawKeypoints(image, keypoints, image, color=color, flags=flags)
349
350 if return_img:
351     return keypoints_image
352
353 if name:
354     cv2.imwrite(name, keypoints_image)
355 cv2.imshow('Keypoint', keypoints_image)
356 cv2.waitKey(0)
357
358 def show_matches(img1, kp1, img2, kp2, name=False, return_img=False):
359     """
360     Fungsi untuk menampilkan pasangan keypoint pada dua gambar
361
362     Parameter:
363         img1      : gambar pertama
364         kp1       : keypoint untuk gambar pertama
365         img2      : gambar kedua
366         kp2       : keypoint untuk gambar kedua
367         name      : nama file jika gambar disimpan
368     """
369 matches = list()
370 for i in range(len(kp1)):
371     matches.append(cv2.DMatch(i, i, 0))
372
373 img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches, None, flags=2)
374
375 if return_img:
376     return img_matches
377
378 if name:
379     cv2.imwrite(name, img_matches)
380 cv2.imshow('Keypoint_Matches', img_matches)
381 cv2.waitKey()
382
383 def put_text(img,
384             text,
385             org,
386             font=cv2.FONT_HERSHEY_SIMPLEX,
387             fontScale=1,
388             color=(0, 255, 0),
389             thickness=1):
390     """
391     Fungsi untuk menambahkan text pada gambar
392     """
393 img = cv2.putText(img, text, org, font, fontScale, color, thickness, cv2.LINE_AA)
394 return img
395
396 def show_polygon(img, kp, name=False, return_img=False):
397     """
398     Fungsi untuk menampilkan polygon yang menyambungkan keypoint-keypoint pada gambar
399
400     Parameter:
401         img      : gambar yang digunakan untuk menampilkan
402         kp       : tuple berisi keypoint
403         name    : nama file jika gambar disimpan
404     """
405 polygon = np.array([[i.pt[0], i.pt[1]] for i in kp], np.int32)
406 img_mod = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
407 img_mod = cv2.polylines(img_mod, polygon, True, (255, 255, 255), 2)

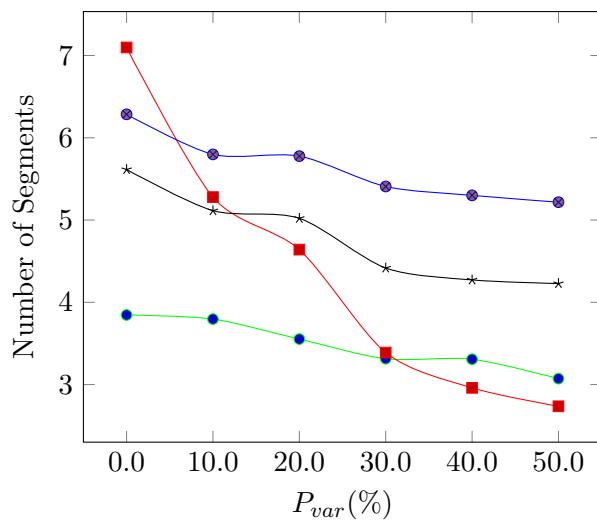
```

```
408     for i, p in enumerate(kp):
409         img_mod = put_text(img_mod, str(i), (int(p.pt[0]), int(p.pt[1])), fontScale=0.7)
410
411     if return_img:
412         return img_mod
413
414     if name:
415         cv2.imwrite(name, img_mod)
416         cv2.imshow('Shapes', img_mod)
417         cv2.waitKey()
418
419 def display_image(img, cmap='viridis'):
420     plt.axis('off')
421     plt.imshow(img, cmap=cmap)
422
423 """
424 Fungsi-fungsi untuk melakukan modifikasi pada gambar, digunakan pada pembuatan data untuk test
425 """
426 def rotate_image(image, angle):
427     image_center = tuple(np.array(image.shape[1::-1]) / 2)
428     rot_mat = cv2.getRotationMatrix2D(image_center, -angle, 1.0)
429     result = cv2.warpAffine(image, rot_mat, image.shape[1::-1], flags=cv2.INTER_LINEAR)
430     return result
431
432 def zoom_center(image, zoom_factor=1.5):
433     y_size = image.shape[0]
434     x_size = image.shape[1]
435
436     # define new boundaries
437     x1 = int(0.5 * x_size * (1 - 1 / zoom_factor))
438     x2 = int(x_size - 0.5 * x_size * (1 - 1 / zoom_factor))
439     y1 = int(0.5 * y_size * (1 - 1 / zoom_factor))
440     y2 = int(y_size - 0.5 * y_size * (1 - 1 / zoom_factor))
441
442     # first crop image then scale
443     img_cropped = image[y1:y2, x1:x2]
444     return cv2.resize(img_cropped, None, fx=zoom_factor, fy=zoom_factor)
```

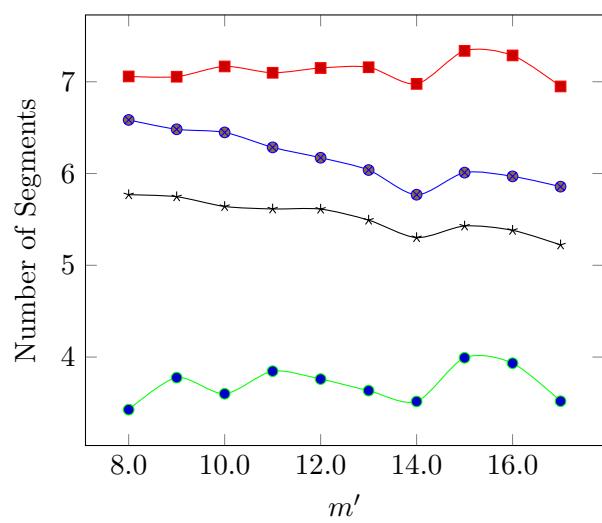
LAMPIRAN B

HASIL EKSPERIMENT

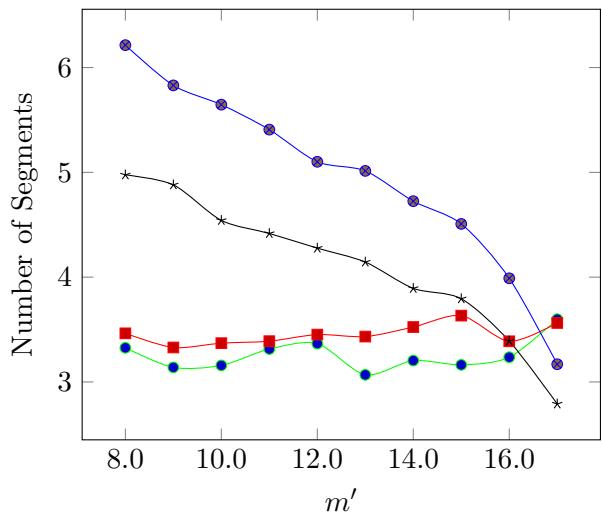
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



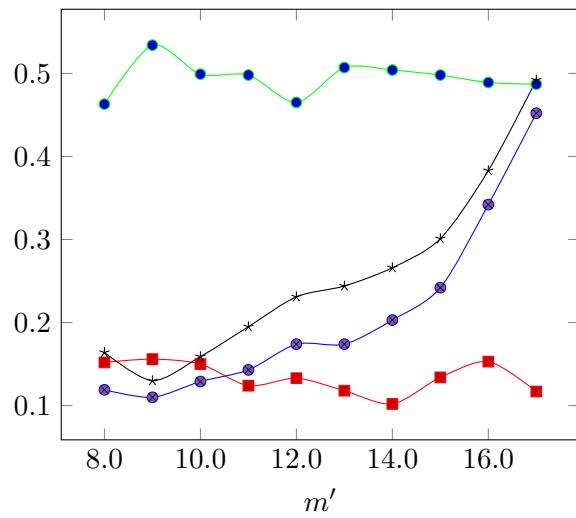
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4