

## 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

<b>KATA PENGANTAR</b>	<b>xv</b>
<b>DAFTAR ISI</b>	<b>xvii</b>
<b>DAFTAR GAMBAR</b>	<b>xix</b>
<b>1 PENDAHULUAN</b>	<b>1</b>
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
<b>2 LANDASAN TEORI</b>	<b>5</b>
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 Clustering . . . . .	24
2.7.1 Agglomerative Clustering . . . . .	24
2.7.2 DBSCAN . . . . .	25
<b>3 ANALISIS &amp; EKSPERIMEN</b>	<b>29</b>
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	Analisis Terhadap Sifat-sifat Fitur Lokal pada Gambar POI . . . . .	34
3.3.1	Ide Dasar Analisis . . . . .	34
3.3.2	Tahapan Analisis . . . . .	35
3.3.3	Implementasi . . . . .	37
3.3.4	Hasil Analisis . . . . .	38
3.3.5	Visualisasi Keypoint yang Unik dan Konsisten . . . . .	39
3.4	Analisis Penggunaan Clustering untuk OIR dengan BSIS . . . . .	42
3.4.1	Data Train dan Test . . . . .	42
3.4.2	Metode BSIS . . . . .	44
3.4.3	Tahapan Analisis . . . . .	45
3.4.4	Tahapan Implementasi . . . . .	45
3.4.5	Hasil Analisis . . . . .	46
3.5	Analisis Pengaruh Parameter Agglomerative Clustering terhadap Sebaran Nilai Keunikan dan Konsistensi . . . . .	54
3.5.1	Ide Analisis dan Tahapan . . . . .	55
3.5.2	Hasil Analisis . . . . .	55
3.6	Analisis Pengaruh Batas Nilai Keunikan yang Digunakan terhadap Jumlah Fitur Lokal yang Didapat dan Tingkat Akurasi BSIS . . . . .	56
3.6.1	Ide Analisis dan Tahapan . . . . .	57
3.6.2	Hasil Analisis . . . . .	57
<b>4</b>	<b>PERANCANGAN</b>	<b>59</b>
4.1	Perancangan Implementasi Metode Clustering Pembuatan Model . . . . .	59
4.1.1	Rancangan Struktur Folder . . . . .	59
4.1.2	Rancangan Proses Clustering . . . . .	60
4.2	Rancangan Kelas Data . . . . .	62
4.3	Rancangan Kelas Util . . . . .	63
4.4	Rancangan Kelas-kelas Implementasi BSIS . . . . .	64
<b>DAFTAR REFERENSI</b>		<b>67</b>
<b>A KODE PROGRAM</b>		<b>69</b>
<b>B HASIL EKSPERIMEN</b>		<b>75</b>

## 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 $\sigma$ yang berbeda. . . . .	10
2.7	Efek nilai $\sigma$ 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 <math>I_{\Sigma}</math></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	<i>Flowchart</i> tahapan analisis <i>clustering</i> untuk mencari fitur lokal yang konsisten dan unik. . . . .	35
3.7	Contoh beberapa gambar yang digunakan pada analisis ini. . . . .	36
3.8	Histogram sebaran nilai keunikan. . . . .	39
3.9	Histogram sebaran nilai keunikan. . . . .	39
3.10	Beberapa contoh <i>keypoint</i> dengan nilai konsistensi dan keunikan tinggi. Lingkaran hijau merupakan <i>Keypoint</i> yang nilai konsistensi dan keunikannya tinggi. Sedangkan lingkaran merah merupakan <i>Keypoint</i> yang nilai konsistensi dan keunikannya rendah.	41
3.11	Contoh gambar pada dataset Book Covers. Keempat gambar tersebut berada dalam satu kelas yang sama. . . . .	42
3.12	Contoh gambar referensi dari dataset Book Covers. . . . .	43
3.13	Contoh gambar referensi dari dataset Book Covers. . . . .	44
3.14	Modifikasi pada metode BSIS yang dilakukan pada analisis ini. . . . .	44
3.15	Tahapan yang dilakukan dalam analisis untuk menguji penggunaan <i>clustering</i> pada BSIS. . . . .	45
3.16	Sebaran nilai keunikan ( <i>uniqueness</i> ) untuk Dataset 400. . . . .	46
3.17	Sebaran nilai keunikan ( <i>consistency</i> ) untuk Dataset 400. . . . .	47
3.18	Perbandingan sebaran waktu Dataset 400. . . . .	49
3.19	Sebaran nilai keunikan ( <i>uniqueness</i> ) untuk Dataset 600. . . . .	50
3.20	Sebaran nilai keunikan ( <i>consistency</i> ) untuk Dataset 600. . . . .	51
3.21	Perbandingan sebaran waktu Dataset 600. . . . .	53
3.22	Sebaran nilai keunikan ( <i>uniqueness</i> ) untuk Dataset 400 saat menggunakan parameter <i>Agglomerative Clustering</i> yang berbeda. . . . .	55
3.23	Sebaran nilai keunikan ( <i>consistency</i> ) untuk Dataset 400 saat menggunakan parameter <i>Agglomerative Clustering</i> yang berbeda. . . . .	56
4.1	Rancangan struktur <i>folder</i> untuk pembuatan model. . . . .	59
4.2	Tahapan proses <i>clustering</i> hingga didapat nilai keunikan dan konsistensi. . . . .	60
4.3	Diagram kelas Data. . . . .	62
4.4	Diagram kelas Util . . . . .	63
4.5	Diagram <i>Package bsis</i> . . . . .	64
B.1	Hasil 1 . . . . .	75
B.2	Hasil 2 . . . . .	75
B.3	Hasil 3 . . . . .	75
B.4	Hasil 4 . . . . .	75

# 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 ikut 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**

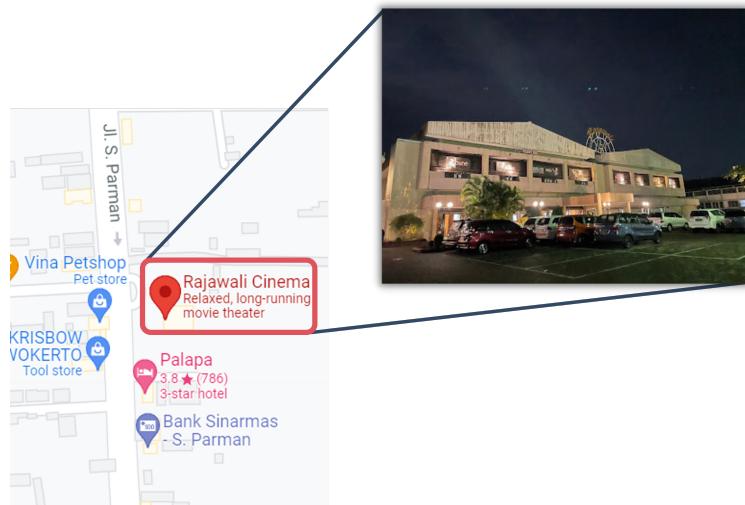
Bab ini berisi kesimpulan yang didapatkan dari hasil analisis serta keseluruhan implementasi dan pengujian yang dilakukan pada penelitian ini.

## 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

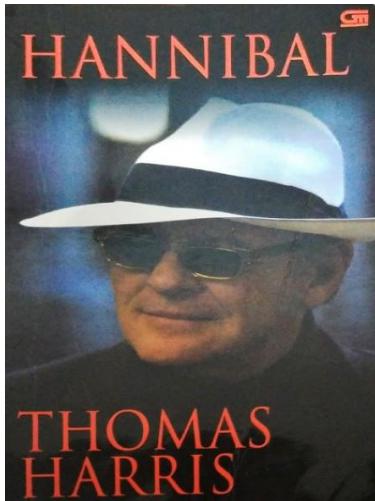
Adanya objek lain yang menutupi sebagian dari objek yang ingin diidentifikasi akan berpotensi menyebabkan beberapa fitur lokal dari objek tidak terdeteksi.

- Sudut Pandang

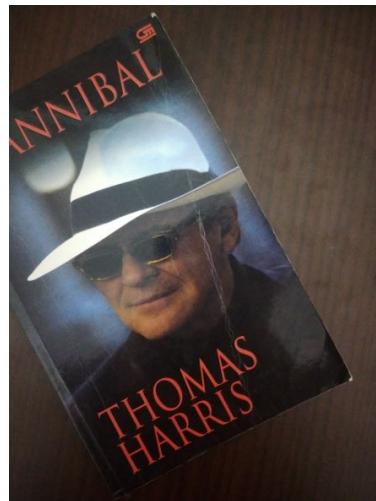
Sudut pengambilan gambar yang berbeda akan memengaruhi pemrosesan. Fitur-fitur lokal dari objek yang ingin diidentifikasi akan menjadi berbeda.

- Translasi

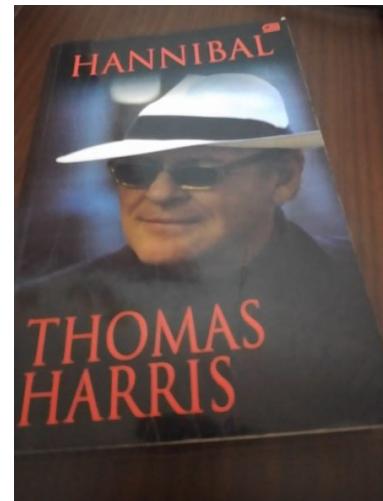
Posisi objek yang ingin diidentifikasi dalam gambar akan memengaruhi pemrosesan. Pencarian pasangan fitur lokal tidak dapat dengan hanya menggunakan posisi di mana fitur lokal tersebut ditemukan pada gambar.



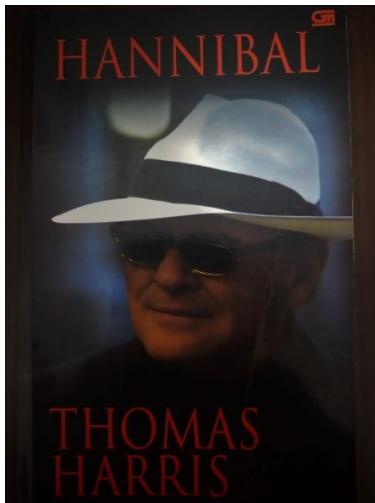
(a) Gambar objek yang ideal, dari sudut tegak lurus



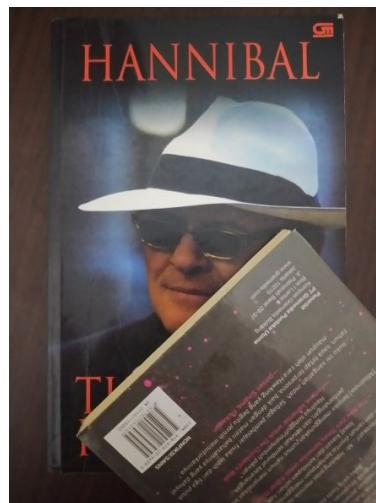
(b) Objek pada gambar mengalami translasi dan rotasi



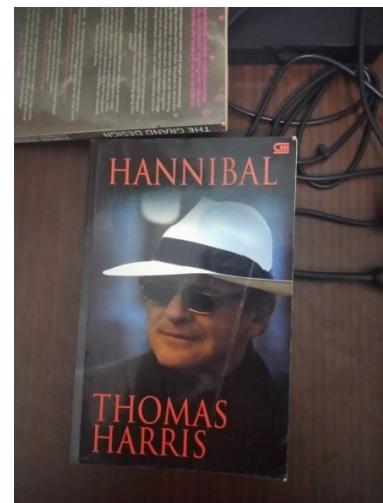
(c) Gambar diambil dari sudut yang berbeda



(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

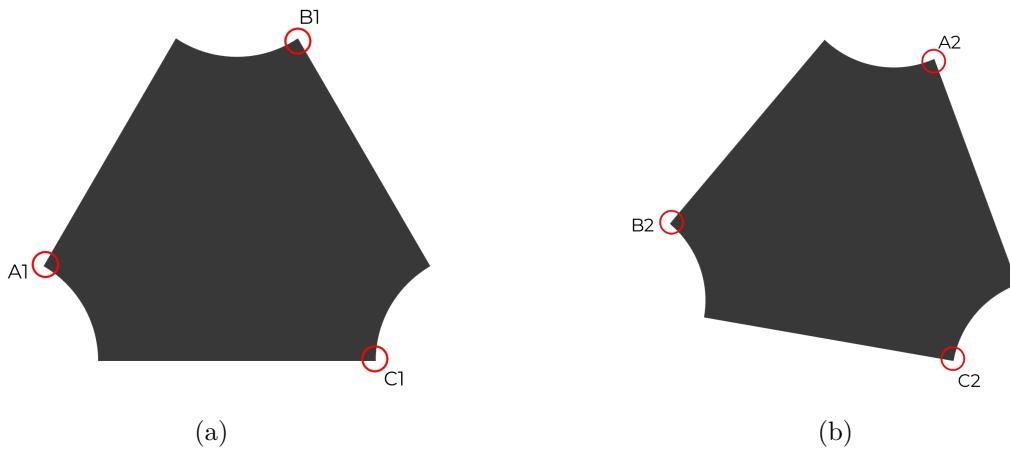
Untuk setiap fitur lokal yang terdeteksi pada gambar masukkan akan dicari beberapa fitur lokal dari gambar pada *dataset* yang paling mirip. Kemiripan fitur lokal ditentukan dengan menghitung jarak (sesuai dengan metrik yang digunakan) antara vektor deskriptor dari kedua fitur lokal tersebut.

#### 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.

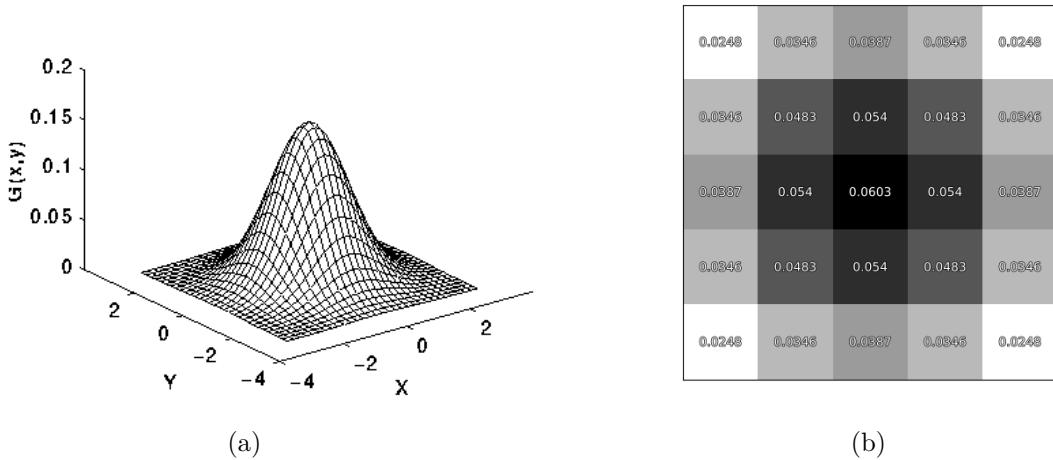
## 2.3 SIFT (Scale Invariant Feature Transform)

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.

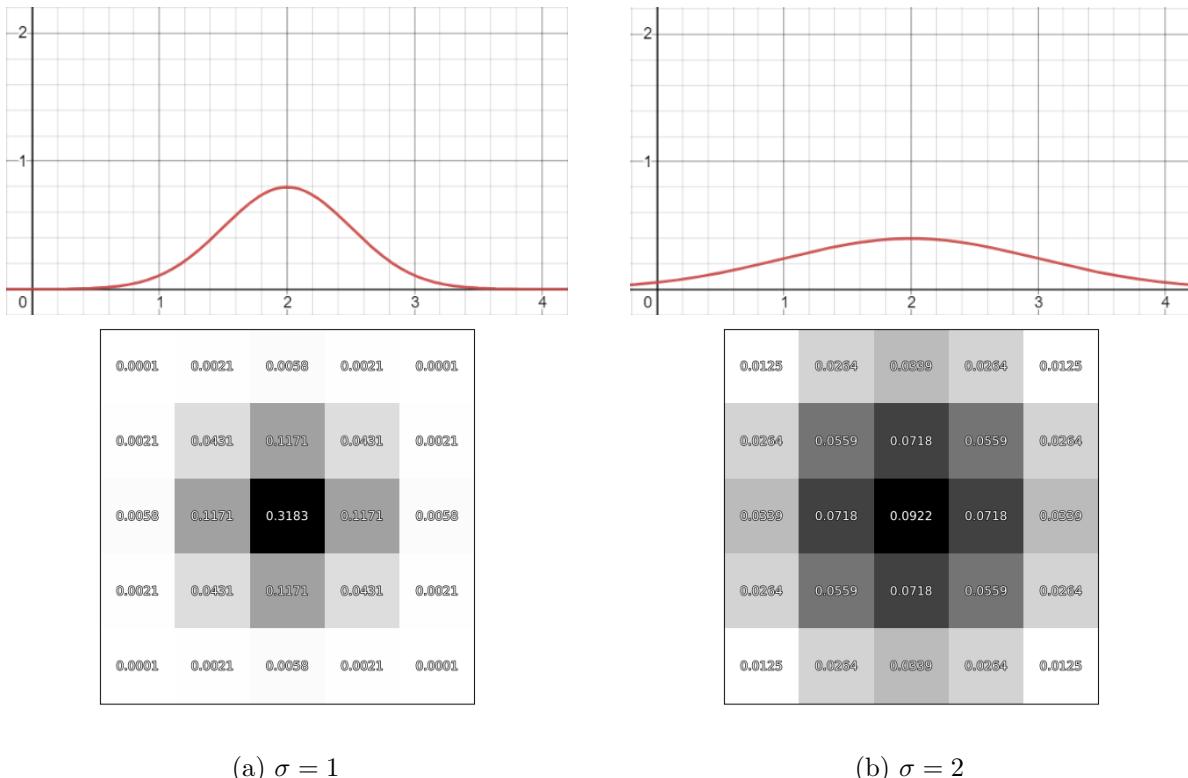
### 2.3.1 Pencarian Extrema

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.

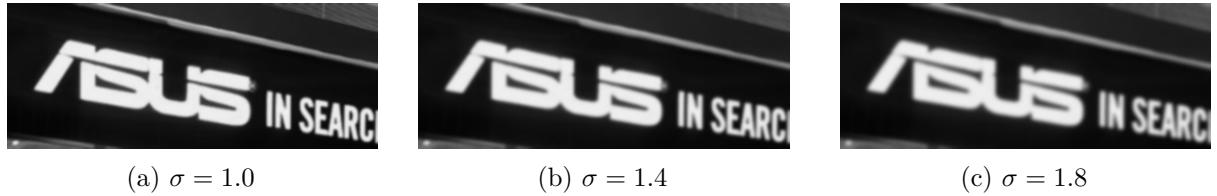
Gambar 2.5: Kurva Gaussian dan bentuk representasi *matrix*-nya.

Pada fungsi Gaussian tingkat penyebaran data dapat diatur dengan mengubah parameter  $\sigma$  yang mengatur nilai deviasi standar. Gambar 2.5a menunjukkan bagaimana nilai  $\sigma$  mengatur bagaimana data tersebar dari *mean*, di mana semakin tinggi nilai  $\sigma$  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  $\sigma$  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  $\sigma$  yang berbeda.

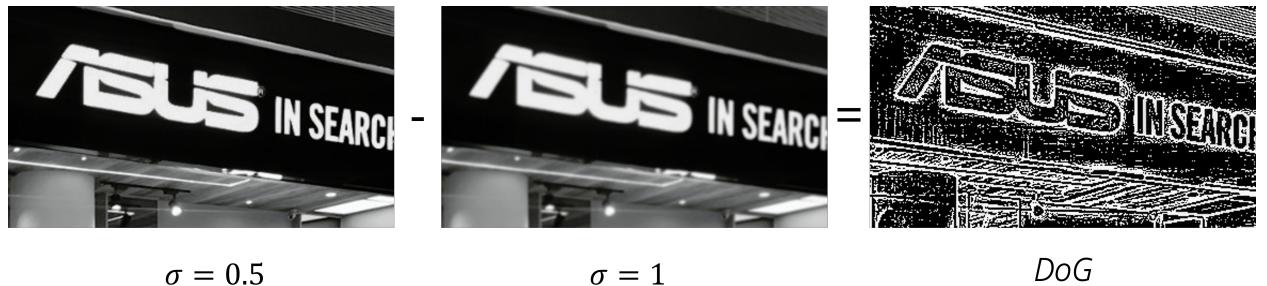
*Matrix* Konvolusi Gaussian ketika diaplikasikan pada gambar akan menyebabkan perubahan nilai tiap *pixel* pada gambar. Nilai dari setiap *pixel* akan menjadi mirip dengan *pixel* tetangga di dekatnya. Perubahan nilai *pixel* akan paling berpengaruh pada daerah dengan perubahan nilai *pixel* yang tinggi. Tingkat perubahan nilai dipengaruhi oleh nilai  $\sigma$  yang digunakan, nilai  $\sigma$  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  $\sigma$  menentukan tingkat *blur* gambar.



Gambar 2.7: Efek nilai  $\sigma$  pada hasil gambar konvolusi.

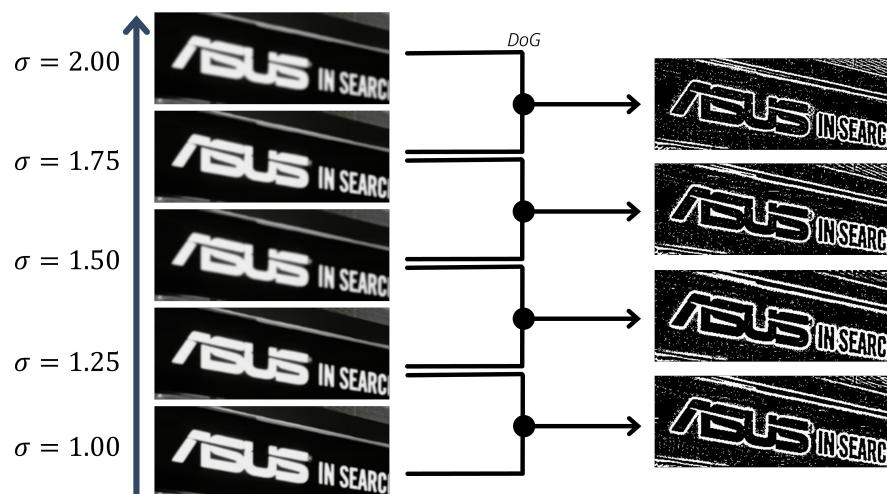
Perubahan nilai  $\sigma$  pada *matrix* konvolusi serta efeknya pada gambar akan dimanfaatkan untuk menghitung *Difference of Gaussian* (DoG). DoG merupakan hasil turunan kedua Gaussian pada gambar. Gambar DoG dapat diperoleh dengan menghitung perbedaan nilai tiap *pixel* dari dua gambar yang telah dikonvolusi oleh *matrix* Gaussian dengan nilai  $\sigma$  yang berbeda. Perbedaan nilai untuk DoG dihitung dengan mengurangi setiap *pixel* pada gambar konvolusi yang memiliki nilai  $\sigma$  yang lebih kecil, dengan setiap *pixel* pada posisi yang sama pada gambar konvolusi yang memiliki nilai  $\sigma$  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  $\sigma$  yang berbeda. Setelah didapatkan beberapa gambar maka akan dihitung DoG untuk setiap gambar yang nilai  $\sigma$ -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.



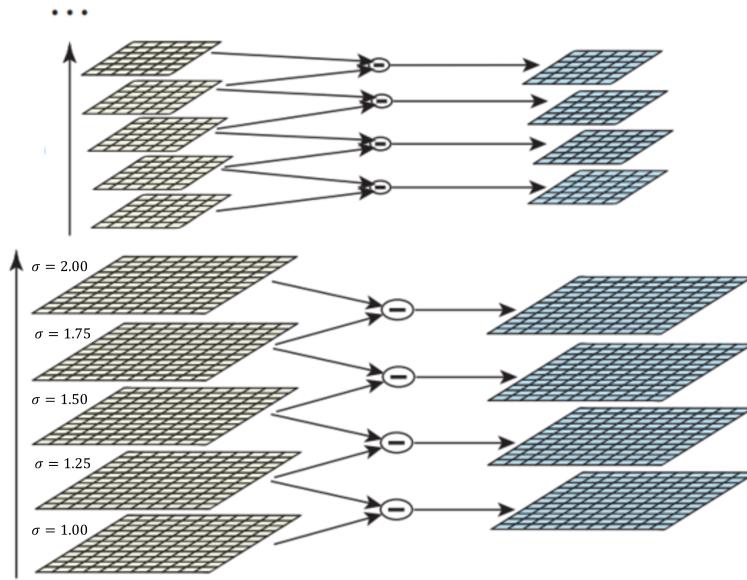
Gambar 2.9: Penggunaan DoG pada SIFT

### 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  $\sigma$  sudah mencapai 2 kali nilai awal
2. Perkecil ukuran gambar (*downsample*) menjadi setengah resolusinya
3. Kembalikan nilai  $\sigma$  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*.

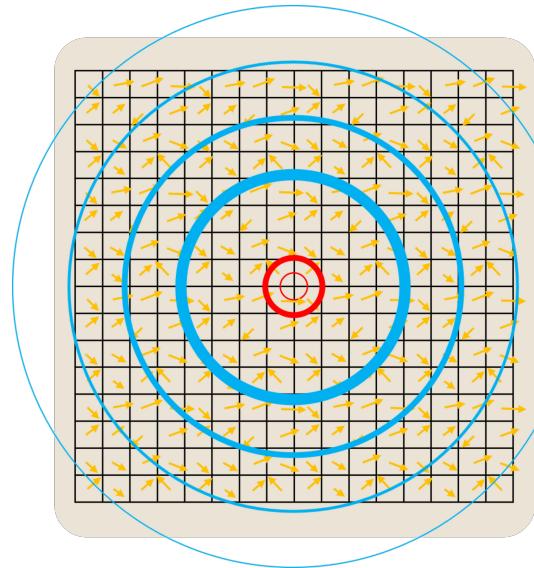
Penentuan orientasi yang dominan dihitung dengan menggunakan *magnitude* ( $m(x, y)$ ) dan orientasi ( $\theta(x, y)$ ) dari *pixel-pixel* dengan menggunakan rumus pada Persamaan 2.2 dan Persamaan 2.3.  $L(x, y)$  pada kedua persamaan tersebut merupakan gambar hasil konvolusi.

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

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

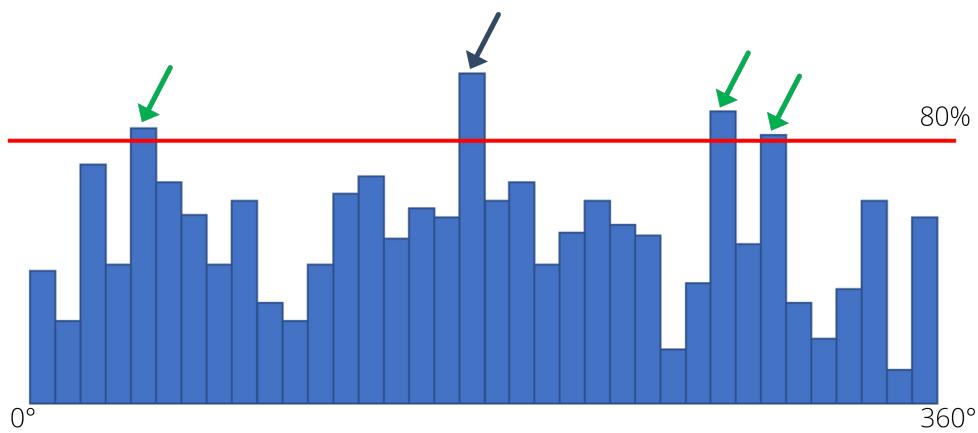
Setiap *pixel* akan dihitung orientasi dan *magnitude*-nya. *Magnitude* akan digunakan sebagai bobot dari *pixel* tersebut. Selain *magnitude*, bobot sebuah *pixel* juga dipengaruhi oleh *Gaussian Weighting*, yang membuat *pixel* yang posisinya dekat dengan titik pusat (titik *keypoint*) akan memiliki 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  $4 \times 4$  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  $\sigma$  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.

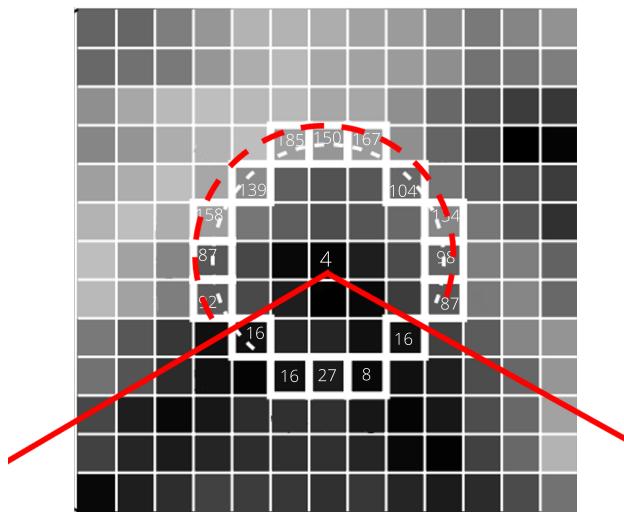
## 2.4 ORB (Oriented FAST and Rotated BRIEF)

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

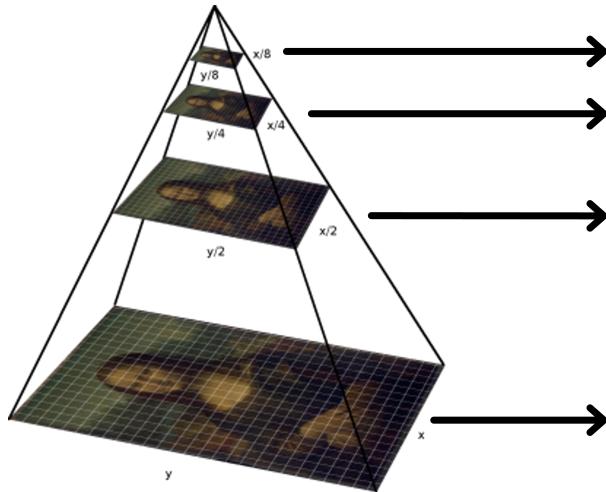
seberapa daerah yang diperiksa merupakan *corner*. ORB akan mengambil  $N$  *keypoint* dengan nilai  $M$  tertinggi untuk ditentukan sebagai *keypoint*.

#### 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.

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

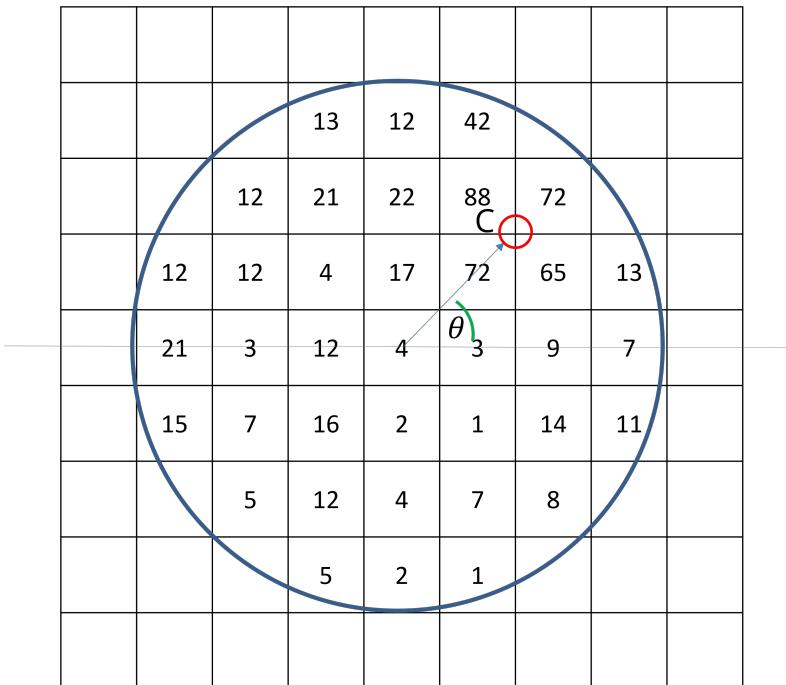
*Centroid* ditentukan dengan menghitung *moment* pada gambar yang didefinisikan pada Persamaan 2.5.

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

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  $\theta$ .



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*.

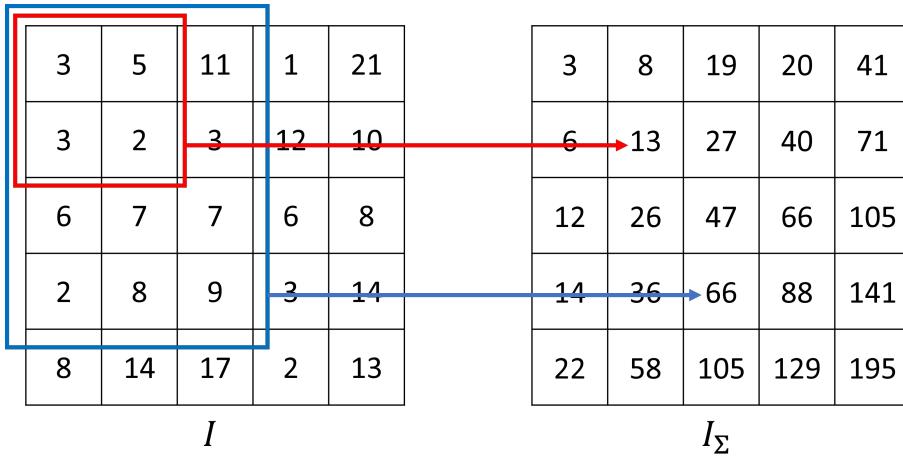
Agar deskriptor bersifat invariant terhadap rotasi gambar terlebih dahulu dirotasi sesuai dengan orientasi yang sudah didapat sebelumnya. Dari gambar yang telah dirotasi diambil daerah berukuran  $31 \times 31$  untuk dilakukan *binary test*. *Binary test* adalah fungsi yang membandingkan nilai intensitas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



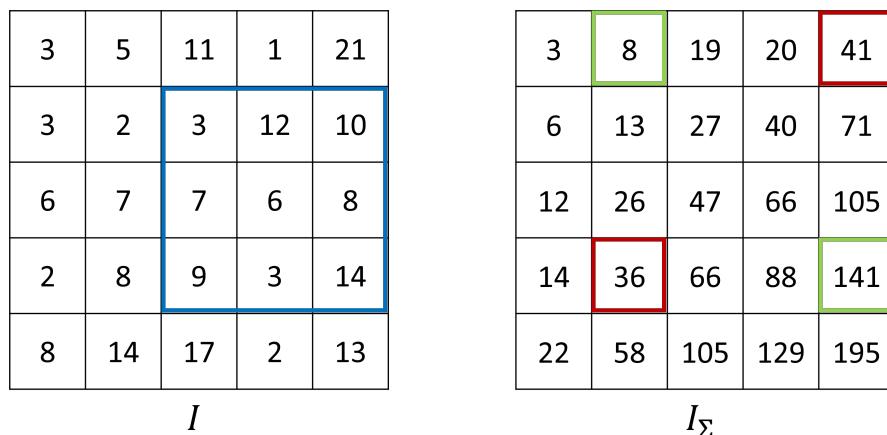
Gambar 2.16: Ilustrasi penghitungan *Integral Image*. *Matrix I* merupakan *matrix* awal dan *Matrix*  $I_{\Sigma}$  merupakan *Integral Image* dari *I*.

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*  $I_{\Sigma}$  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 ( $I_{\Sigma}$ ) 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  $I_{\Sigma}$  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  $5 \times 5$  dalam daerah  $31 \times 31$  yang digunakan untuk menghitung deskriptor. Penghitungan nilai total pada daerah  $5 \times 5$  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 ( $\sigma$ ). *Mean* merupakan rata-rata dari nilai kemiripan dari  $N$  pasangan  $P_Q$  dan  $\sigma$  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

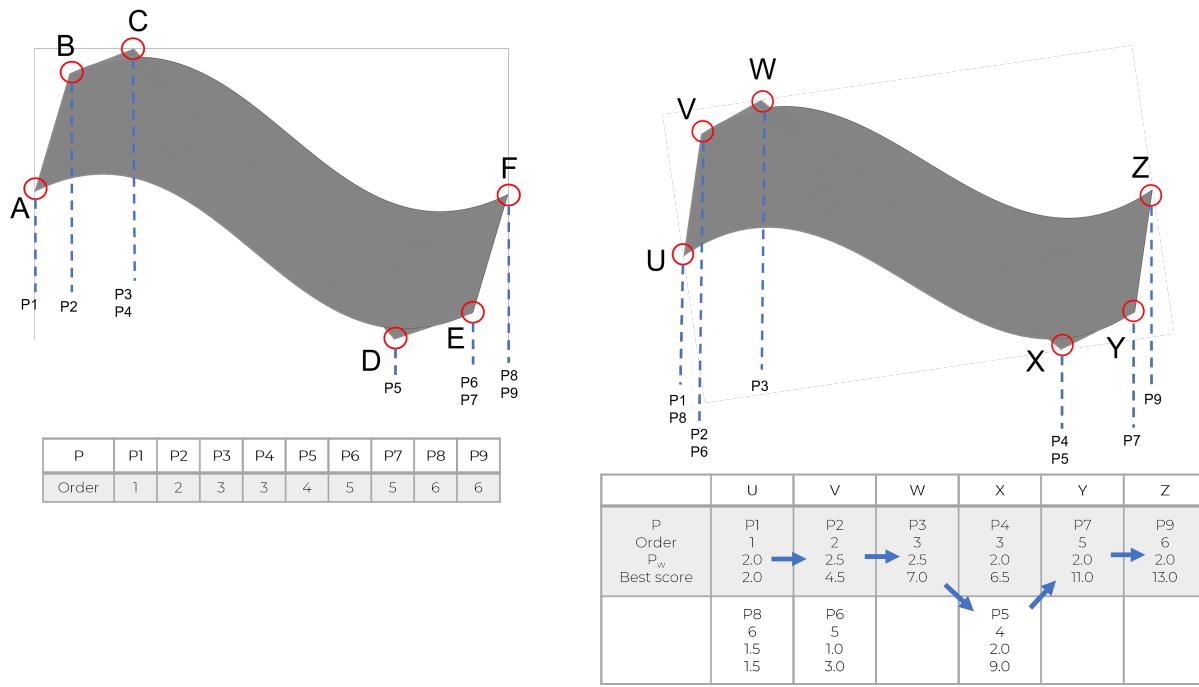
Pada tahap *Pairing* telah didapat beberapa pasangan antara fitur lokal gambar *query* dan gambar *training* yang paling berpotensi merupakan pasangan yang benar. Dari tahap sebelumnya setiap fitur lokal pada gambar *query* dapat dipasangkan dengan lebih dari 1 fitur lokal pada gambar *training*. Pasangan-pasangan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

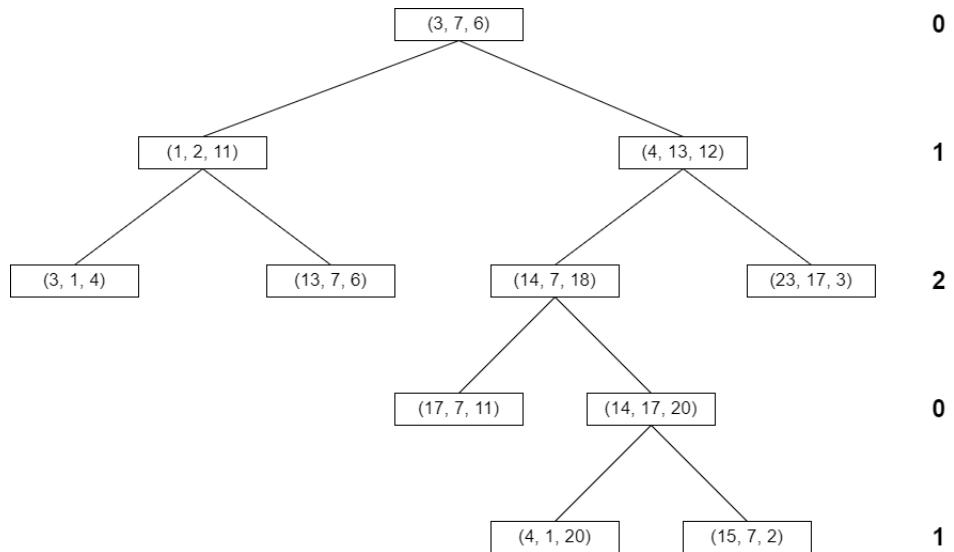
## 2.6 KD-Tree

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 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]

### 2.7.1 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**
- 

### 2.7.2 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  $\epsilon$  yang berpusat pada objek tersebut.  $MinPts$  dan  $\epsilon$  adalah parameter yang ditentukan secara manual. Nilai  $MinPts$  dan  $\epsilon$  akan memengaruhi hasil *cluster* yang dihasilkan.

Setiap objek yang merupakan *core object* beserta anggotanya (objek lain pada radius  $\epsilon$ ) akan menjadi satu *cluster*. Jika dalam radius  $\epsilon$  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  $\epsilon$ -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  $\epsilon$  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
- $\epsilon$ : 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 & EKSPERIMENTASI

### 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 yang dilakukan ini hanya 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.

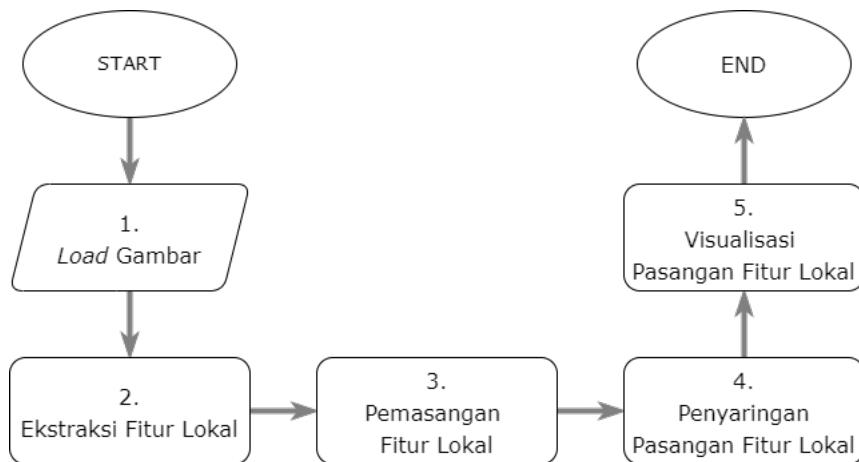
#### 3.1.1 Ide Dasar Analisis

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  $q$ . Cara ini diimplementasikan dengan melakukan *Lowe's Ratio Test* seperti yang telah dijelaskan pada [2.2](#). Setelah didapatkan pasangan-pasangan yang kuat maka 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](#).

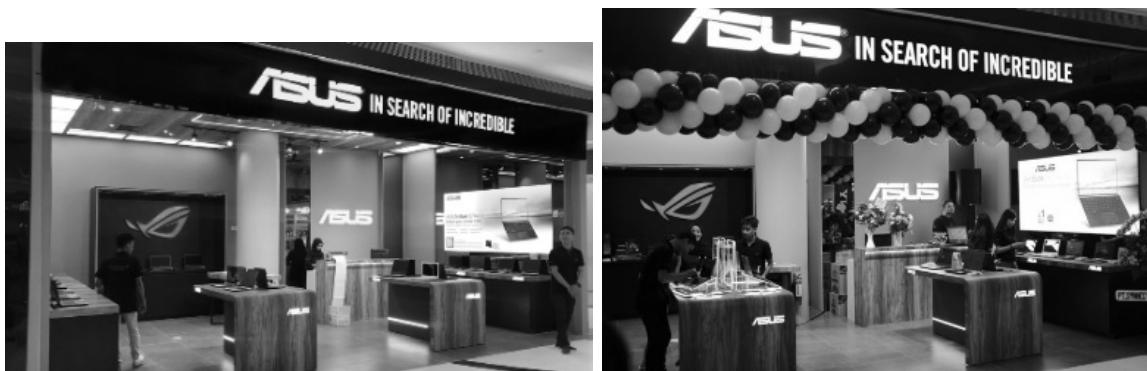


Gambar 3.1: *Flowchart* tahapan analisis pengenalan POI.

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 POI yang sama yang diambil dari sudut pengambilan dan waktu yang berbeda sehingga terdapat objek-objek yang sudut-sudut yang berbeda dari kedua gambar tersebut. Kedua gambar 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. Import library OpenCV (cv2).
2. Load 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 objek dengan nama **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 deskriptor.
6. Membuat pasangan *keypoint* dengan menggunakan fungsi **matcher.knnMatch(descriptors1, descriptors2, 2)**. Parameter **descriptors1** merupakan deskriptor untuk Gambar *Q* dan **descriptors2** merupakan deskriptor 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 deskriptor 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 disaring berdasarkan kekuatan pasangannya.



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

Dilihat dari pasangan-pasangan *keypoint*-nya semua berasal dari bagian POI yang merupakan logo dan bagian konsisten. Baik bagian 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.

Dengan menggunakan ide bahwa logo dan bagian konsisten dari POI memberikan fitur yang kuat, pengenalan POI yang dilakukan dengan algoritma OIR seharusnya dapat ditingkatkan dengan melakukan OIR hanya menggunakan fitur-fitur yang kuat tersebut. OIR dengan hanya menggunakan fitur-fitur yang kuat tersebut dapat diproses dengan waktu yang lebih cepat karena fitur lokal yang diperiksa 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. Di mana 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 dan bagian pilar yang berada di tengah POI tersebut. Sedangkan bagian yang tidak konsisten ditunjukkan oleh objek seperti orang-orang yang lewat. 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 Analisis Terhadap Sifat-sifat Fitur Lokal pada Gambar POI

Pada 3.2 disebutkan bahwa untuk mendapatkan fitur lokal yang berasal dari logo dan bagian konsisten serta unik perlu dicari fitur lokal yang sifatnya konsisten dan unik terhadap suatu POI tersebut. Pencarian fitur lokal konsisten dan unik tersebut dapat dilakukan dengan menggunakan teknik *clustering*.

Pada bagian ini akan dilakukan analisis untuk menguji apakah penggunaan *clustering* dapat menemukan fitur lokal yang sifatnya konsisten dan unik. Dari fitur lokal yang didapatkan dari hasil *clustering* tersebut ingin diperiksa apakah benar berasal dari bagian POI yang merupakan logo atau bagian lain POI yang sifatnya permanen. Bagian yang tidak permanen adalah objek-objek seperti orang-orang yang lewat.

#### 3.3.1 Ide Dasar Analisis

Analisis akan dilakukan dengan menggunakan beberapa gambar yang berbeda. Setiap gambar memiliki kelas yang merupakan POI asal gambar tersebut. Gambar dengan kelas yang sama merupakan gambar dari POI yang sama dengan sudut dan waktu pengambilan yang berbeda. Gambar-gambar tersebut diekstrak fitur lokalnya dan setiap fitur lokal ditandai asal gambarnya dan kelas dari gambar asalnya.

Fitur-fitur lokal dari gambar tersebut lalu dibagi menjadi kelompok-kelompok dengan menggunakan teknik *clustering*. Ada masalah jika ada pola tertentu pada POI yang berulang, pola berulang tersebut akan menghasilkan banyak fitur lokal dengan ciri-ciri yang mirip. Untuk mengatasi itu masing-masing gambar terlebih dahulu dilakukan *clustering* pada fitur lokalnya dan dihitung *centroid* untuk tiap *cluster*-nya. *Centroid-centroid* ini yang nantinya akan dilakukan *clustering* dengan fitur lokal dari gambar lain untuk dihitung konsistensi dan keunikannya.

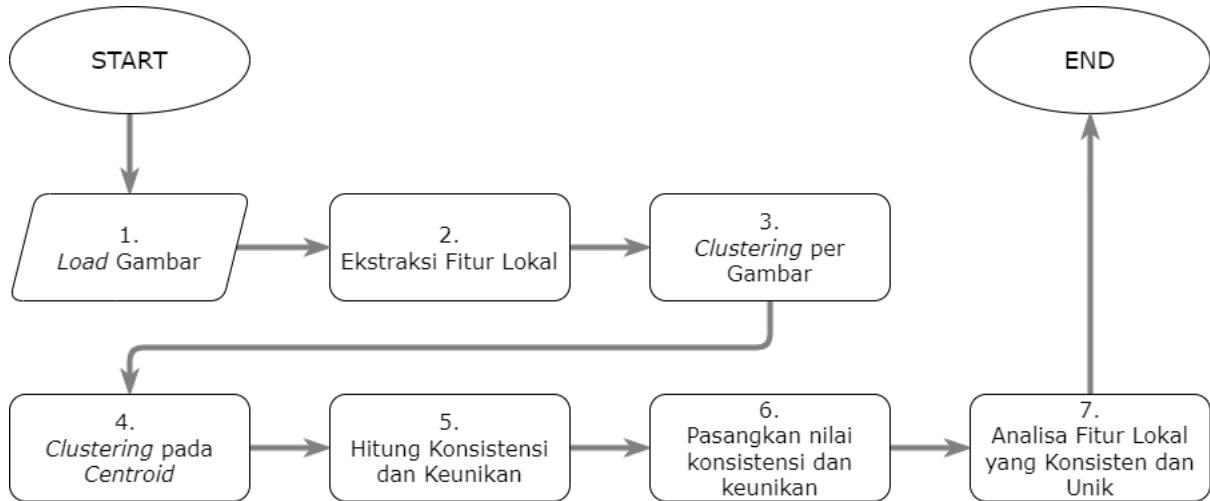
Fitur lokal yang konsisten adalah fitur lokal yang muncul di semua gambar dari sebuah kelas. Kekonsistennan ini dihitung dengan melihat fitur-fitur lokal dalam sebuah *cluster* jika dalam *cluster* tersebut terdapat fitur lokal dari kelas A dan terdapat beberapa fitur lokal lain dari kelas yang sama 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 *clsuter* tersebut (setelah dihapus yang duplikat).

### 3.3.2 Tahapan Analisis

Langkah-langkah analisis yang dilakukan untuk mencari fitur lokal dengan sifat konsisten dan unik dapat dilihat pada *flowchart* di Gambar 3.6



Gambar 3.6: *Flowchart* tahapan analisis *clustering* untuk mencari fitur lokal yang konsisten dan unik.

Langkah-langkah pada *flowchart* di Gambar 3.6 secara rinci adalah sebagai berikut:

#### 1. Load Gambar

Pada analisis ini digunakan sebanyak total 12 gambar yang terbagi menjadi 3 kelas POI. Setiap kelas POI memiliki 4 gambar yang merupakan gambar POI tersebut dengan sudut pengambilan dan waktu pengambilan yang berbeda. Beberapa contoh gambar yang digunakan dapat dilihat pada Gambar 3.7.



Gambar 3.7: 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 3.3.1. 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.

### 3.3.3 Implementasi

Langkah-langkah yang telah dijelaskan pada 3.3.2 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. Load semua gambar yang digunakan dengan fungsi `cv2.imread`. *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 deskriptor dimasukkan ke dalam sebuah *dataframe* dengan nama `descriptors`.
5. Pada *dataframe descriptors* 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 deskriptor dengan mengambil 128 kolom pertama dari *dataframe descriptors*. *Clustering* menggunakan objek *AgglomerativeClustering* dari `sklearn` dengan parameter `distance_threshold` didapat dari menghitung jarak rata-rata dari sampel data sebanyak 50 baris dan dibagi dua. Setelah didapat label untuk tiap *cluster* dilakukan iterasi untuk tiap *cluster*, pada tiap iterasi hitung *centroid* dari *cluster* tersebut. Hasil *centroid* lalu dimasukkan ke dalam *dataframe* bernama `df_centroid`, simpan juga nomor *cluster* (`cluster_label`), nama gambar (`img`), dan kelas gambar (`img_class`).
8. Melakukan *clustering* pada *dataframe df\_centroid*. *Clustering* juga dilakukan dengan menggunakan objek *AgglomerativeClustering* dari `sklearn`. Untuk parameter `distance_threshold` digunakan setengah dari jarak rata-rata sampel data sebanyak 100 baris. Masukkan nomor *cluster* sebagai kolom dengan nama `cluster2_label` di `df_centroid`.
9. Membuat sebuah *dictionary* bernama `cluster2_class_count` yang nantinya digunakan untuk menyimpan nilai keunikan.
10. Mengelompokkan data pada `df_centroid` berdasarkan kolom `cluster2_label`. Untuk setiap kelompok pertama hanya ambil satu *centroid* dari tiap gambar (lakukan `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 (nilai keunikan).
11. 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`.
12. Memasukkan *list uniqueness* sebagai kolom pada `df_centroid`.
13. 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`.

14. Membuat sebuah *list* dengan nama `img_count`. *List* ini akan berisi nilai konsistensi untuk tiap *centroid* di `df_centroid`.
15. 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`.
16. Masukkan `img_count` ke sebagai kolom di `df_centroid` dengan nama `consistency`.
17. 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`.
18. Mengambil *keypoint* dengan nilai `consistency` dan `uniqueness` yang tinggi dan memvisualisasikan *keypoint* tersebut pada gambar aslinya.

### 3.3.4 Hasil Analisis

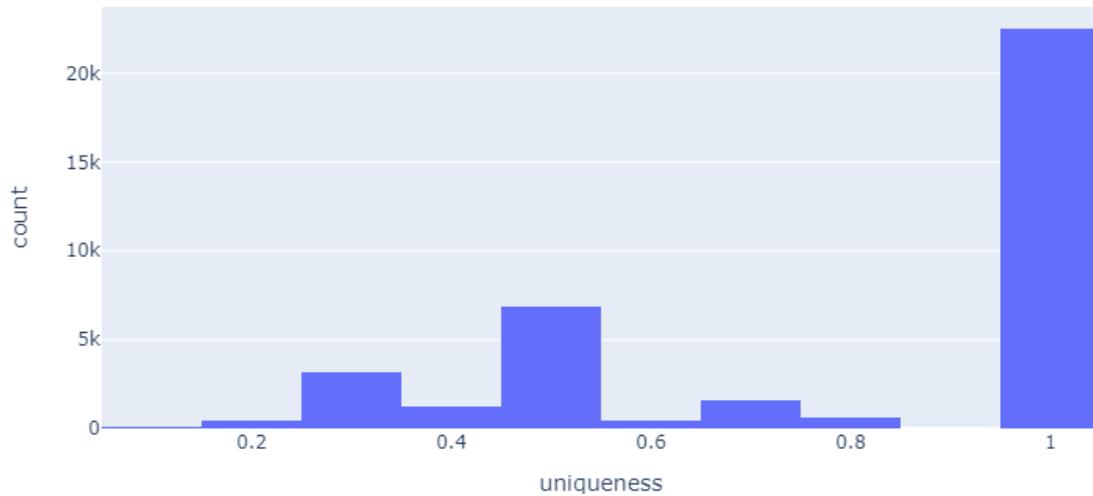
Setelah dilakukan implementasi sesuai dengan tahapan pada 3.3.3 didapatkan tabel berisi deskriptor untuk *keypoint* beserta dengan nilai keunikan (`uniqueness`) dan konsistensinya (`consistency`). Beberapa contoh potongan dari tabel hasil dapat dilihat pada Tabel 3.1

<code>img</code>	<code>img_class</code>	<code>cluster2_label</code>	<code>uniqueness</code>	<code>consistency</code>
hnm_1.jpg	hnm	0	0.5	0.25
hnm_1.jpg	hnm	0	0.5	0.25
asus_3.jpg	asus	0	0.5	0.25
hnm_4.jpg	hnm	1	0.5	0.25
cgv_2.jpg	cgv	1	0.5	0.25
cgv_2.jpg	cgv	1	0.5	0.25
cgv_2.jpg	cgv	2	0.66	0.5
hnm_2.jpg	hnm	2	0.33	0.25
cgv_4.jpg	cgv	2	0.66	0.5
cgv_4.jpg	cgv	2	0.66	0.5
cgv_2.jpg	cgv	2	0.66	0.5
cgv_4.jpg	cgv	3	0.5	0.25
asus_4.jpg	asus	3	0.5	0.25
hnm_2.jpg	hnm	4	1	0.5
hnm_3.jpg	hnm	4	1	0.5
hnm_2.jpg	hnm	4	1	0.5
hnm_2.jpg	hnm	4	1	0.5
asus_3.jpg	asus	5	0.5	0.25
hnm_2.jpg	hnm	5	0.5	0.25
asus_3.jpg	asus	5	0.5	0.25

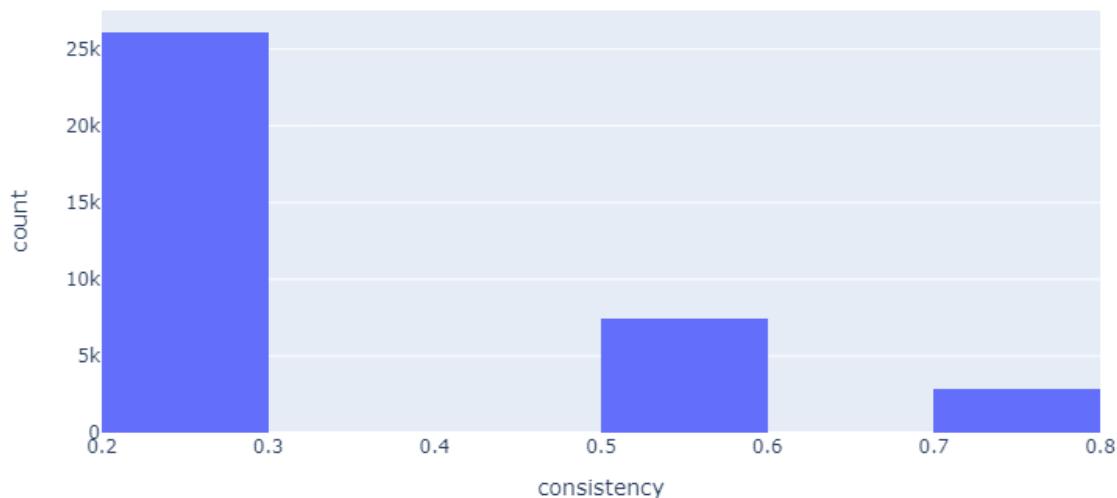
Tabel 3.1: Potongan beberapa contoh dari tabel deskriptor.

Nilai-nilai pada kolom `uniqueness` dan `consistency` didapat dari melakukan penghitungan sebaran anggota tiap *cluster* seperti yang telah dijelaskan di subbab-subbab sebelumnya. Contohnya seperti semua *keypoint* pada `cluster2_label` 4 memiliki nilai keunikan yang sempurna (1) karena semua *keypoint* pada *cluster* tersebut berasal dari kelas (`img_class`) yang sama. Sedangkan nilai konsistensinya hanya 0.5 karena *keypoint-keypoint* dalam *cluster* tersebut walaupun semua berasal dari satu kelas tetapi hanya dari 2 gambar yang berbeda, di mana seharusnya dalam satu kelas terdapat 4 gambar berbeda.

Data pada Tabel 3.1 menunjukkan *keypoint* (fitur lokal) beserta nilai keunikan (`uniqueness`) dan konsistensinya (`consistency`). Dari tabel tersebut terlihat bahwa nilai keduanya cukup beragam. Sebaran untuk nilai keunikan dan konsistensi dapat dilihat pada Gambar 3.8 dan Gambar 3.9.

Histogram Sebaran Nilai "***uniqueness***"

Gambar 3.8: Histogram sebaran nilai keunikan.

Histogram Sebaran Nilai "***consistency***"

Gambar 3.9: Histogram sebaran nilai keunikan.

### 3.3.5 Visualisasi Keypoint yang Unik dan Konsisten

Pada bagian ini akan dilakukan visualisasi *keypoint* dalam gambar untuk melihat apakah *keypoint* yang unik dan konsisten tersebut benar berasal dari bagian yang permanen dalam POI. Untuk itu *keypoint-keypoint* tersebut akan disaring berdasarkan nilai keunikannya dan konsistensinya.

Penyaringan fitur lokal berdasarkan nilai keunikan dan konsistensi dilakukan dengan menetapkan suatu batas bawah untuk masing-masing nilai tersebut. Fitur lokal yang diambil untuk visualisasi adalah fitur lokal yang nilai keunikan dan konsistensinya di atas nilai-nilai batas tersebut. Penentuan batas dilakukan dengan melihat nilai persebaran keduanya dan menetukan nilai batas sehingga jumlah fitur lokal yang didapat optimal untuk melakukan visualisasi.

Sebaran untuk nilai keunikan dapat dilihat pada Gambar 3.8 dari subbab sebelumnya. Dari histogram pada Gambar 3.8 terlihat bahwa sebaran nilai keunikan tidak merata. Mayoritas *keypoint* merupakan *keypoint* yang unik karena memiliki nilai keunikan 1. Untuk itu akan digunakan nilai

0.9 sebagai batas pengambilan untuk visualisasi. *Keypoint* dengan **uniqueness** di atas 0.9 dinilai unik dan akan diambil untuk visualisasi.

Sedangkan untuk nilai konsistensi sebarannya dapat dilihat pada Gambar 3.9. Berbeda dengan nilai keunikan sebaran nilai konsistensi mayoritas berada pada nilai yang kecil yaitu pada rentang 0.2 - 0.3. Untuk mendapatkan *keypoint* dengan jumlah yang optimal akan digunakan nilai 0.5 sebagai batas pengambilan.

Dari hasil penyaringan menggunakan **uniqueness**  $> 0.9$  dan **consistency**  $> 0.7$  didapatkan sebanyak 785 *keypoint* dari total yang berjumlah 36940. *Keypoint* yang didapat tersebut lalu divisualisasikan dalam gambar beserta dengan *keypoint-keypoint* yang sifatnya tidak unik dan konsisten (memiliki nilai keunikan dan konsistensi tinggi). Hasilnya dapat dilihat pada Gambar 3.10.



Gambar 3.10: Beberapa contoh *keypoint* dengan nilai konsistensi dan keunikan tinggi. Lingkaran hijau merupakan *Keypoint* yang nilai konsistensi dan keunikannya tinggi. Sedangkan lingkaran merah merupakan *Keypoint* yang nilai konsistensi dan keunikannya rendah.

Pada gambar pertama (gambar di atas) dari Gambar 3.10 terlihat bahwa *keypoint-keypoint* yang didapat banyak yang berasal dari bagian logo atau objek yang permanen. Sebagian besar berasal dari logo di bagian atas, logo di tembok belakang dan logo di meja. Selain itu untuk *keypoint* yang tidak berasal dari logo semua berasal dari bagian POI yang permanen seperti tembok, meja, atau

poster. Tidak ada *keypoint* yang berasal dari orang.

Sedangkan pada gambar kedua masih ada beberapa *keypoint* yang berasal dari orang yang lewat. Tetapi sebagian besar *keypoint* berasal dari bagian pilar yang berada di tengah gambar yang merupakan bagian permanen dari POI.

Dari hasil analisis ini didapat bahwa dengan metode *clustering* seperti pada langkah-langkah di atas mampu memberikan *keypoint* yang cukup unik dan konsisten. Metode mendapatkan *keypoint* yang unik dan konsisten ini diharapkan dapat memberikan fitur yang kuat jika digunakan untuk melakukan OIR dengan BSIS.

### 3.4 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 kali ini dilakukan dengan menggunakan dataset bernama Book Covers yang berisi gambar cover buku. Dataset tersebut digunakan karena gambar-gambar yang ada dinilai cukup mirip dengan POI, di mana dalam sebuah gambar ada objek inti yang menyerupai logo dan objek latar belakang juga karena pada tahap ini belum dilakukan pengumpulan data.

Analisis dilakukan dengan melakukan identifikasi sejumlah gambar tes terhadap dataset yang menggunakan semua fitur lokal dan dataset yang menggunakan fitur lokal yang telah disaring berdasarkan keunikan dan kekonsistennya. Hasil tes keduanya lalu dibandingkan tingkat akurasi dan waktu pemrosesannya. Proses identifikasi dilakukan dengan menggunakan metode BSIS 2.5.

#### 3.4.1 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 3.11.



Gambar 3.11: Contoh gambar pada dataset Book Covers. Keempat gambar tersebut berada dalam satu kelas yang sama.

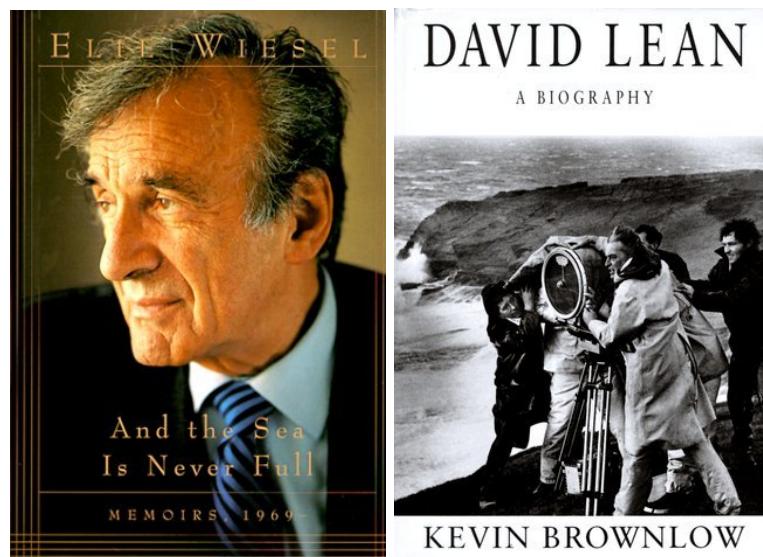
Data *train* yang sama digunakan dua kali dengan ukuran yang berbeda. Ukuran gambar diperkecil sehingga sisi paling panjangnya tidak lebih dari 400 *pixel* untuk set pertama dan tidak lebih dari 600 *pixel* untuk set kedua. Ukuran gambar yang berbeda akan menghasilkan jumlah fitur lokal yang berbeda 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 3.3. Karena keterbatasan *memory* pada perangkat yang digunakan jumlah fitur lokal yang diekstrak dari tiap gambar dibatasi sebanyak 250 fitur lokal.

### Data Test

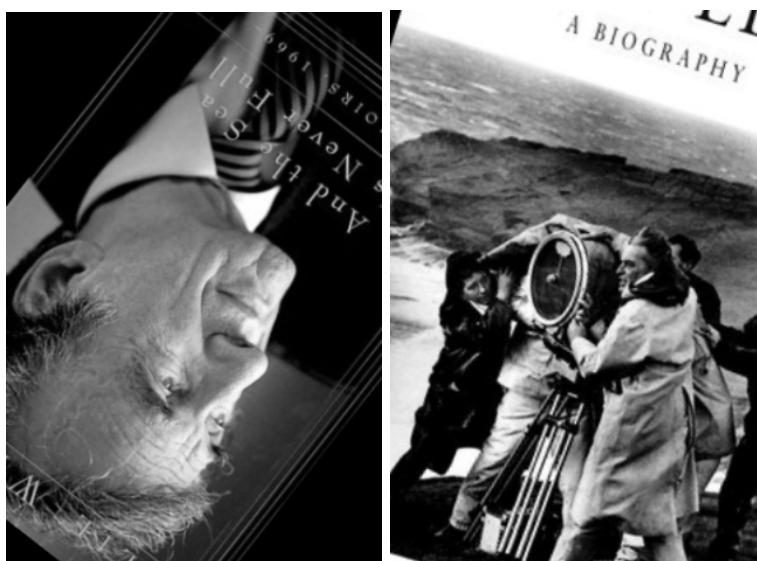
Data *test* yang digunakan diambil dari gambar referensi dataset Book Covers. Gambar referensi merupakan gambar buku yang menjadi objek utama dalam setiap kelas dengan kondisi yang ideal. Contoh gambar referensi dapat dilihat pada Gambar 3.12.



Gambar 3.12: 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 3.13. 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 3.13: Contoh gambar referensi dari dataset Book Covers.

### 3.4.2 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*.

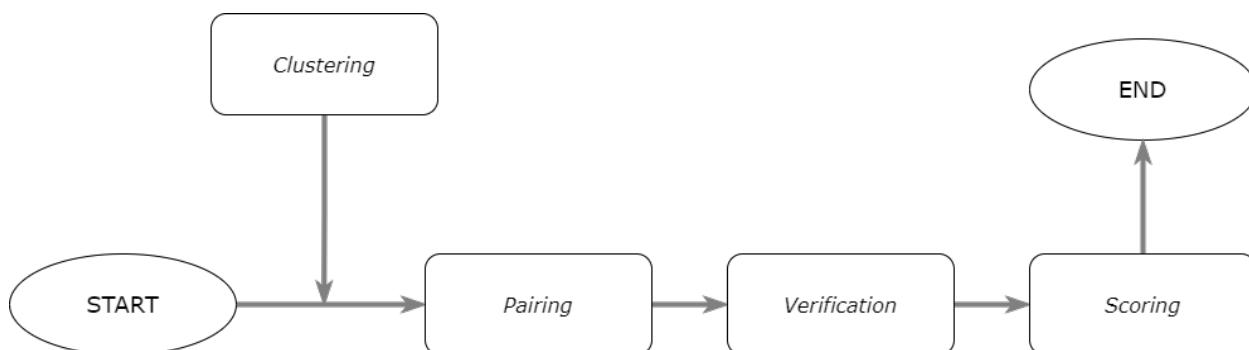
2. *Verification*

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

3. *Scoring*

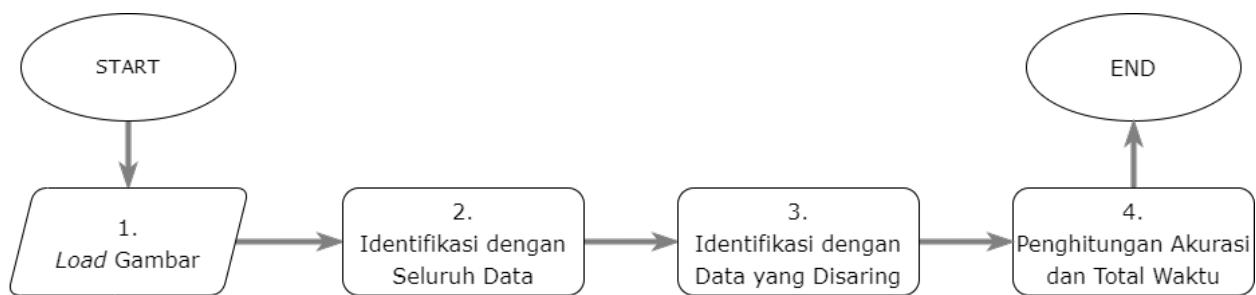
Setelah didapat pasangan-pasangan yang memiliki sifat konsisten secara geometris dihitung total bobot dari pasangan-pasangan tersebut sebagai nilai yang menunjukkan kemiripan dua buah gambar tersebut.

Pada analisis ini dilakukan sedikit modifikasi pada metode BSIS, modifikasi dilakukan seperti yang ditunjukkan pada Gambar 3.14



Gambar 3.14: 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 disaring 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*.



Gambar 3.15: Tahapan yang dilakukan dalam analisis untuk menguji penggunaan *clustering* pada BSIS.

### 3.4.3 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 3.15. 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 Disaring  
Lakukan kembali identifikasi seluruh gambar `test` tetapi dengan menggunakan hanya sebagian dataset yang telah disaring 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.

### 3.4.4 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. Import *library* yang digunakan: OpenCV (`cv2`) dan Pandas (`pandas`).
2. *Load* gambar `test` masukkan 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. *Load* 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. Membuat 5 buah `list` untuk menyimpan hasil deteksi. Dengan rincian sebagai berikut:
  - `q_name`: Untuk menyimpan nama dari gambar `test`.
  - `t_name`: Nama gambar yang paling mirip berdasarkan hasil BSIS.
  - `kdtree_time`: Waktu yang diperlukan implementasi KD-Tree untuk mencari *n* pasangan

paling mirip untuk semua fitur lokal di gambar *test*.

- **total\_time**: Total waktu yang diperlukan oleh BSIS hingga memberikan hasil.
- **total\_weight**: Total bobot pasangan yang dibentuk oleh BSIS.

7. Mengiterasi *test\_dataset* dan masukkan nama gambar ke *q\_name*. Pada setiap iterasi terlebih dahulu melakukan ekstraksi fitur lokal dari gambar *test* dengan fungsi *sift.detectAndCompute*.
8. Menggunakan *keypoint* dan deskriptor dari fitur lokal gambar untuk melakukan BSIS.
9. Mengisi *list t\_name, kdtree\_time, total\_time*, dan *total\_weight* dengan hasil dari BSIS.
10. Menggabungkan *list q\_name, t\_name, kdtree\_time, total\_time*, dan *total\_weight* ke dalam sebuah *dataframe*.
11. Untuk setiap baris di *dataframe* memeriksa apakah hasilnya benar. Hasil dinilai benar jika gambar *train* yang didapat dari hasil BSIS (gambar pada *t\_name*) merupakan salah satu dari 4 gambar yang berada pada kelas yang sama dengan gambar *test*.
12. Melakukan lagi langkah 5-10 dengan dataset yang telah disaring berdasarkan nilai batas *uniqueness* dan *consistency* yang telah ditentukan sebelumnya.

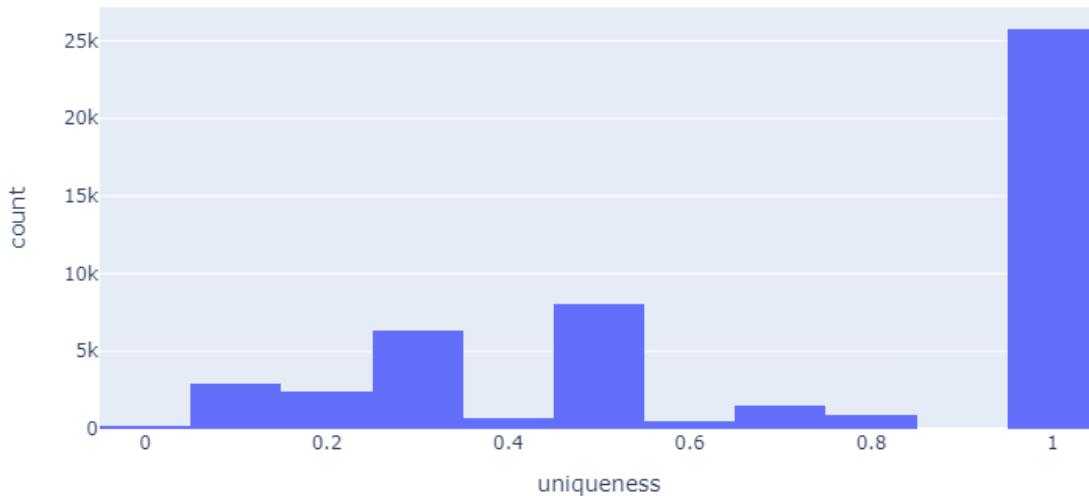
Langkah-langkah tersebut dilakukan untuk Dataset 400 dan Dataset 600. Hasil dari kedua dataset tersebut lalu dibandingkan.

### 3.4.5 Hasil Analisis

#### Dataset 400

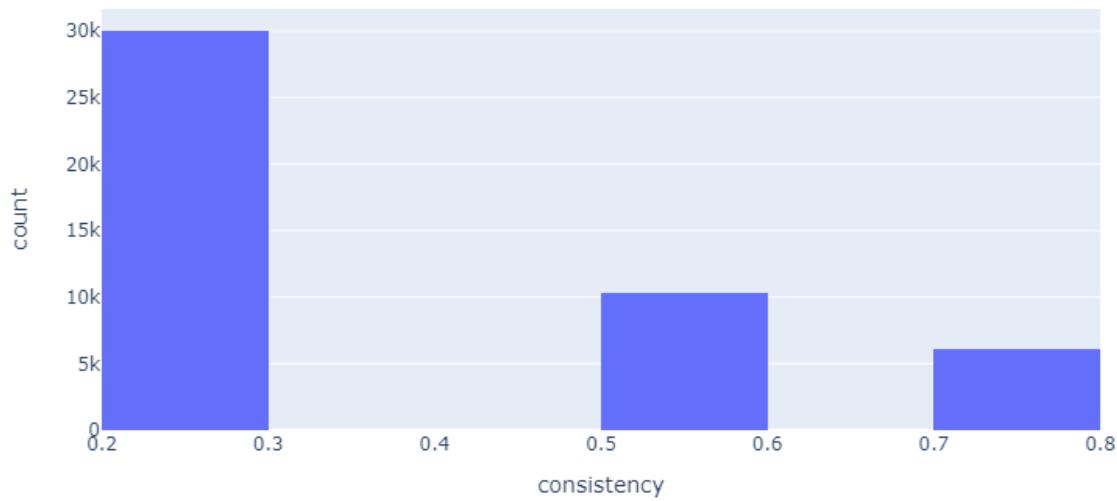
Dataset 400 memiliki total fitur lokal yang dihasilkan dari seluruh gambar sebanyak 49524 fitur lokal. Setelah dilakukan *clustering* menghitung nilai keunikan dan konsistensi didapat hasil sebaran nilai keunikan dan konsistensi seperti pada Gambar 3.16 dan Gambar 3.17.

Histogram Sebaran Nilai "**uniqueness**"



Gambar 3.16: Sebaran nilai keunikan (*uniqueness*) untuk Dataset 400.

Histogram Sebaran Nilai "**consistency**"



Gambar 3.17: Sebaran nilai keunikan (**consistency**) untuk Dataset 400.

Berdasarkan sebaran data tersebut ditentukan batas untuk nilai keunikan (**uniqueness**) adalah  $> 0.99$  dan untuk konsistensi (**consistency**) adalah  $> 0.49$ . Setelah data disaring menggunakan batas-batas tersebut didapatkan total sebanyak 13281 fitur lokal. Data keseluruhan dan data yang telah disaring tersebut lalu digunakan untuk pengenalan gambar *test*.

Hasil pengenalan gambar *test* pada dataset keseluruhan dapat dilihat potongannya pada Tabel 3.2.

	<b>q_name</b>	<b>t_name</b>	<b>kdtree_time</b>	<b>total_time</b>	<b>total_weight</b>	<b>is_true</b>
0	001-test0.jpg	001-iPhone.jpg	1.218	1.331	118.152	1
1	001-test1.jpg	001-Droid.jpg	1.142	1.235	81.080	1
2	002-test0.jpg	002-Droid.jpg	1.038	1.082	75.547	1
3	002-test1.jpg	002-Droid.jpg	1.044	1.094	59.720	1
4	003-test0.jpg	003-iPhone.jpg	1.076	1.164	147.223	1
5	003-test1.jpg	003-iPhone.jpg	1.079	1.188	187.710	1
6	004-test0.jpg	004-iPhone.jpg	1.024	1.061	278.902	1
7	004-test1.jpg	004-iPhone.jpg	1.100	1.218	500.139	1
8	005-test0.jpg	005-Droid.jpg	0.839	0.998	888.100	1
9	005-test1.jpg	005-Droid.jpg	0.747	0.897	737.421	1
...	...	...	...	...	...	...
90	046-test0.jpg	046-iPhone.jpg	0.691	0.736	192.444	1
91	046-test1.jpg	046-iPhone.jpg	0.710	0.770	220.680	1
92	047-test0.jpg	047-iPhone.jpg	0.692	0.737	306.918	1
93	047-test1.jpg	047-iPhone.jpg	0.674	0.690	40.375	1
94	048-test0.jpg	048-iPhone.jpg	0.658	0.690	409.121	1
95	048-test1.jpg	048-iPhone.jpg	0.661	0.683	288.478	1
96	049-test0.jpg	049-iPhone.jpg	0.717	0.825	465.788	1
97	049-test1.jpg	049-iPhone.jpg	0.698	0.789	413.234	1
98	050-test0.jpg	050-Canon.jpg	0.872	1.047	77.518	1
99	050-test1.jpg	044-Droid.jpg	0.796	0.930	33.449	0

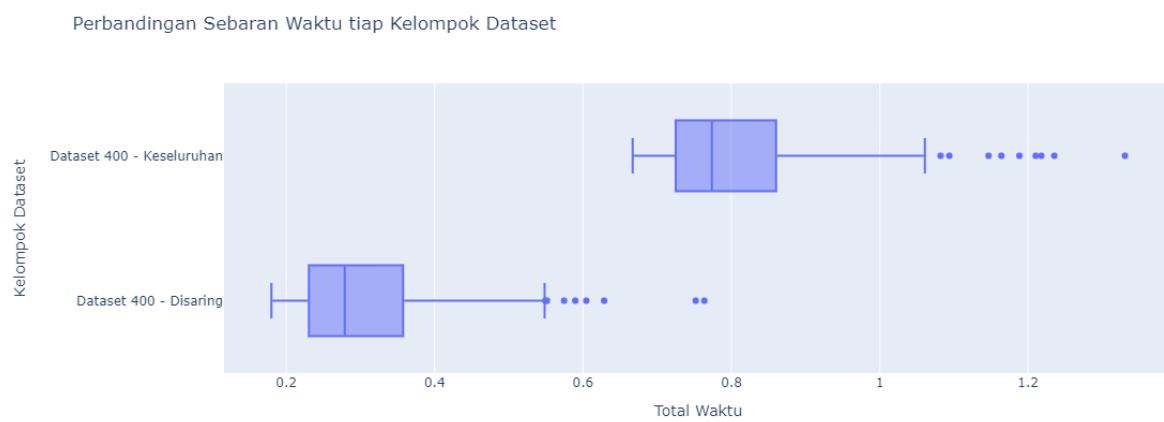
Tabel 3.2: Beberapa contoh hasil BSIS pada keseluruhan Dataset 400.

Sedangkan untuk hasil pada data yang telah disaring beberapa potongannya adalah seperti dapat dilihat pada Tabel 3.3

	<b>q_name</b>	<b>t_name</b>	<b>kdtree_time</b>	<b>total_time</b>	<b>total_weight</b>	<b>is_true</b>
0	001-test0.jpg	001-iPhone.jpg	0.395	0.510	74.987	1
1	001-test1.jpg	001-Droid.jpg	0.368	0.465	58.541	1
2	002-test0.jpg	002-iPhone.jpg	0.313	0.353	53.240	1
3	002-test1.jpg	002-Droid.jpg	0.321	0.360	56.742	1
4	003-test0.jpg	003-Droid.jpg	0.357	0.450	112.820	1
5	003-test1.jpg	003-Droid.jpg	0.337	0.440	147.134	1
6	004-test0.jpg	004-iPhone.jpg	0.283	0.308	58.114	1
7	004-test1.jpg	004-iPhone.jpg	0.365	0.478	59.555	1
8	005-test0.jpg	005-Droid.jpg	0.372	0.548	326.424	1
9	005-test1.jpg	005-Droid.jpg	0.375	0.574	364.295	1
...	...	...	...	...	...	...
90	046-test0.jpg	046-iPhone.jpg	0.200	0.245	150.351	1
91	046-test1.jpg	046-iPhone.jpg	0.211	0.272	229.700	1
92	047-test0.jpg	047-iPhone.jpg	0.202	0.248	133.144	1
93	047-test1.jpg	Nan	0.190	0.204	0.000	0
94	048-test0.jpg	048-iPhone.jpg	0.174	0.197	243.964	1
95	048-test1.jpg	048-iPhone.jpg	0.178	0.190	113.485	1
96	049-test0.jpg	049-iPhone.jpg	0.241	0.336	339.476	1
97	049-test1.jpg	049-iPhone.jpg	0.209	0.276	221.041	1
98	050-test0.jpg	050-Canon.jpg	0.364	0.549	87.451	1
99	050-test1.jpg	002-iPhone.jpg	0.299	0.409	16.458	0

Tabel 3.3: Beberapa contoh hasil BSIS pada Dataset 400 yang telah disaring berdasarkan **uniqueness** dan **consistency**.

Perbedaan sebaran waktu yang digunakan untuk tiap gambar pada saat digunakan data secara keseluruhan dan yang telah disaring dapat dilihat pada Gambar 3.18.



Gambar 3.18: Perbandingan sebaran waktu Dataset 400.

Dari gambar tersebut terlihat bahwa ada perbedaan sebaran nilai yang besar antara dataset yang telah disaring dan keseluruhan. Dataset yang telah disaring memiliki total waktu yang tersebar pada nilai-nilai yang lebih kecil dibandingkan dengan dataset secara keseluruhan. Untuk perbandingan total waktu serta akurasi dari dataset keseluruhan dan yang telah disaring dapat dilihat pada tabel berikut:

<b>Keseluruhan</b>		<b>Disaring</b>	
Total Waktu (detik)	Akurasi (%)	Total Waktu (detik)	Akurasi (%)
82.15	98	31.79	96

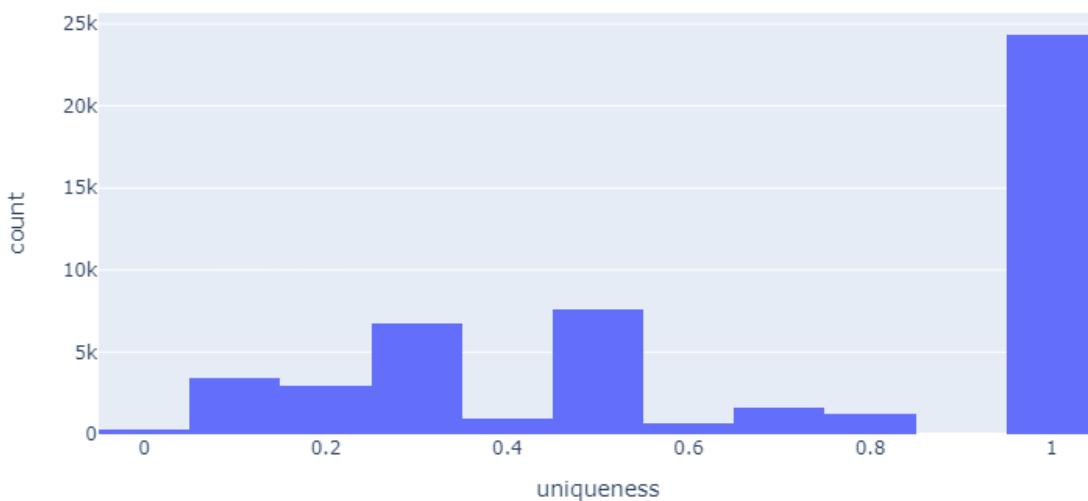
Dari tabel di atas terlihat bahwa penyaringan pada fitur lokal meningkatkan waktu pemrosesan yang dapat dilihat dari penurunan total waktu yang signifikan (61.3%). Penurunan waktu tersebut dikarenakan jumlah fitur lokal yang diproses lebih sedikit. Walaupun begitu penurunan akurasi dari saat menggunakan keseluruhan data ke penggunaan data yang disaring tidak cukup signifikan. Akurasi hanya menurun sebanyak 2%.

Hasil tersebut menunjukkan adanya peningkatan performa BSIS dari segi waktu jika menggunakan fitur lokal yang sudah disaring terlebih dahulu. Walaupun akurasi juga menurun tetapi penurunan akurasi tidak cukup signifikan jika dibandingkan dengan peningkatan kecepatan.

## Dataset 600

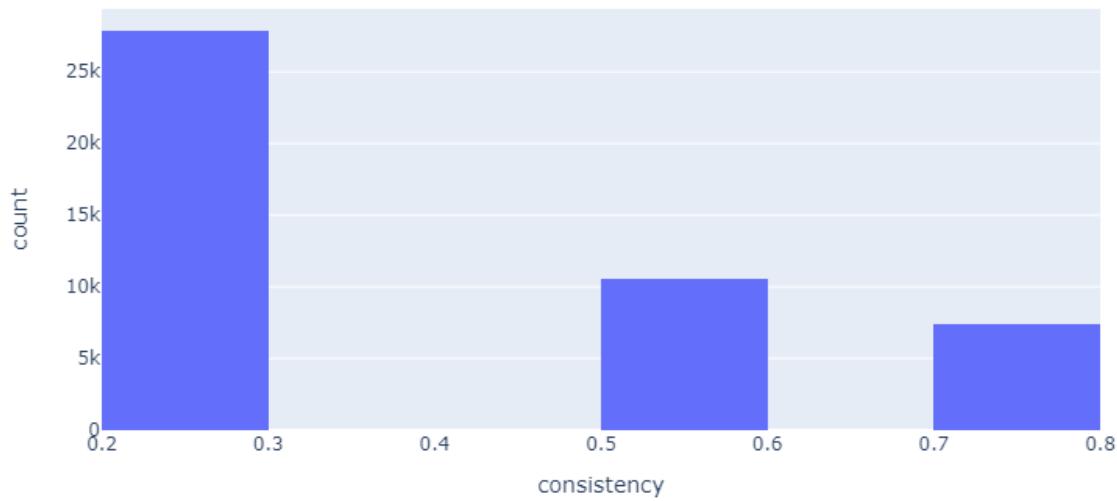
Dataset 600 memiliki total fitur lokal yang dihasilkan dari seluruh gambar sebanyak 49976 fitur lokal. Setelah dilakukan *clustering* menghitung nilai keunikan dan konsistensi didapat hasil sebaran nilai keunikan dan konsistensi seperti pada Gambar 3.19 dan Gambar 3.20.

Histogram Sebaran Nilai "*uniqueness*"



Gambar 3.19: Sebaran nilai keunikan (*uniqueness*) untuk Dataset 600.

Histogram Sebaran Nilai "**consistency**"



Gambar 3.20: Sebaran nilai keunikan (**consistency**) untuk Dataset 600.

Berdasarkan sebaran data tersebut ditentukan batas untuk nilai keunikan (**uniqueness**) adalah  $> 0.99$  dan untuk konsistensi (**consistency**) adalah  $> 0.49$ . Setelah data disaring menggunakan batas-batas tersebut didapatkan total sebanyak 13853 fitur lokal. Data keseluruhan dan data yang telah disaring tersebut lalu digunakan untuk pengenalan gambar *test*. Hasil pengenalan gambar *test* pada dataset keseluruhan dapat dilihat potongannya pada Tabel 3.4.

	<b>q_name</b>	<b>t_name</b>	<b>kdtree_time</b>	<b>total_time</b>	<b>total_weight</b>	<b>is_true</b>
0	001-test0.jpg	001-iPhone.jpg	1.245	1.354	151.124	1
1	001-test1.jpg	001-iPhone.jpg	1.156	1.248	132.160	1
2	002-test0.jpg	002-iPhone.jpg	1.051	1.106	110.707	1
3	002-test1.jpg	002-Droid.jpg	1.050	1.095	69.215	1
4	003-test0.jpg	003-iPhone.jpg	1.082	1.158	154.965	1
5	003-test1.jpg	003-Droid.jpg	1.090	1.181	117.895	1
6	004-test0.jpg	004-iPhone.jpg	1.037	1.077	248.097	1
7	004-test1.jpg	004-iPhone.jpg	1.126	1.259	394.791	1
8	005-test0.jpg	005-Droid.jpg	0.917	1.040	775.478	1
9	005-test1.jpg	005-Droid.jpg	0.743	0.873	578.210	1
...	...	...	...	...	...	...
90	046-test0.jpg	046-Droid.jpg	0.682	0.714	98.835	1
91	046-test1.jpg	046-Droid.jpg	0.699	0.751	146.754	1
92	047-test0.jpg	047-iPhone.jpg	0.679	0.730	303.266	1
93	047-test1.jpg	047-iPhone.jpg	0.668	0.682	18.561	1
94	048-test0.jpg	048-iPhone.jpg	0.652	0.676	270.232	1
95	048-test1.jpg	048-iPhone.jpg	0.650	0.667	166.423	1
96	049-test0.jpg	049-iPhone.jpg	0.710	0.789	297.539	1
97	049-test1.jpg	049-iPhone.jpg	0.696	0.765	311.168	1
98	050-test0.jpg	050-Canon.jpg	0.871	1.037	103.319	1
99	050-test1.jpg	004-iPhone.jpg	0.802	0.915	43.560	0

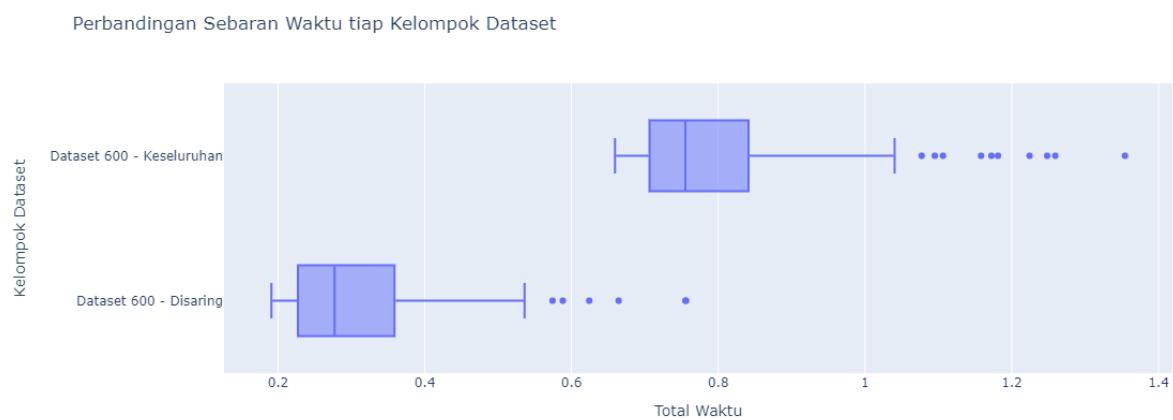
Tabel 3.4: Beberapa contoh hasil BSIS pada keseluruhan Dataset 600.

Sedangkan untuk hasil pada data yang telah disaring beberapa potongannya adalah seperti dapat dilihat pada Tabel 3.5

	q_name	t_name	kdtree_time	total_time	total_weight	is_true
0	001-test0.jpg	001-iPhone.jpg	0.406	0.519	61.144	1
1	001-test1.jpg	001-Droid.jpg	0.383	0.469	58.070	1
2	002-test0.jpg	002-iPhone.jpg	0.333	0.375	87.264	1
3	002-test1.jpg	002-iPhone.jpg	0.335	0.373	37.118	1
4	003-test0.jpg	003-iPhone.jpg	0.353	0.421	109.130	1
5	003-test1.jpg	003-iPhone.jpg	0.340	0.419	79.736	1
6	004-test0.jpg	Nan	0.292	0.312	0.000	0
7	004-test1.jpg	047-iPhone.jpg	0.373	0.475	41.536	0
8	005-test0.jpg	005-Droid.jpg	0.380	0.534	263.913	1
9	005-test1.jpg	005-Droid.jpg	0.398	0.574	378.274	1
...	...	...	...	...	...	...
90	046-test0.jpg	046-Droid.jpg	0.199	0.227	29.213	1
91	046-test1.jpg	046-Droid.jpg	0.213	0.261	82.782	1
92	047-test0.jpg	047-iPhone.jpg	0.203	0.239	66.674	1
93	047-test1.jpg	Nan	0.191	0.203	0.000	0
94	048-test0.jpg	048-iPhone.jpg	0.178	0.199	193.285	1
95	048-test1.jpg	048-iPhone.jpg	0.179	0.191	143.975	1
96	049-test0.jpg	049-iPhone.jpg	0.231	0.313	276.651	1
97	049-test1.jpg	049-iPhone.jpg	0.214	0.276	243.250	1
98	050-test0.jpg	050-Canon.jpg	0.370	0.536	43.878	1
99	050-test1.jpg	027-Droid.jpg	0.303	0.420	20.752	0

Tabel 3.5: Beberapa contoh hasil BSIS pada Dataset 600 yang telah disaring berdasarkan **uniqueness** dan **consistency**.

Perbedaan sebaran waktu yang digunakan untuk tiap gambar pada saat digunakan data secara keseluruhan dan yang telah disaring dapat dilihat pada Gambar 3.21.



Gambar 3.21: Perbandingan sebaran waktu Dataset 600.

Dari gambar tersebut terlihat bahwa ada perbedaan sebaran nilai yang besar antara dataset yang telah disaring dan keseluruhan. Dataset yang telah disaring memiliki total waktu yang tersebar pada nilai-nilai yang lebih kecil dibandingkan dengan dataset secara keseluruhan. Untuk perbandingan total waktu serta akurasi dari dataset keseluruhan dan yang telah disaring dapat dilihat pada tabel berikut:

<b>Keseluruhan</b>		<b>Disaring</b>	
Total Waktu (detik)	Akurasi (%)	Total Waktu (detik)	Akurasi (%)
80.96	96	31.41	89

Dari tabel di atas terlihat bahwa penyaringan pada fitur lokal meningkatkan waktu pemrosesan yang dapat dilihat dari penurunan total waktu yang signifikan (61.2%). Penurunan waktu tersebut dikarenakan jumlah fitur lokal yang diproses lebih sedikit. Tetapi penurunan akurasi dari saat menggunakan keseluruhan data ke penggunaan data yang disaring juga cukup signifikan. Akurasi menurun sebanyak 7%

Hasil tersebut menunjukkan adanya peningkatan performa BSIS dari segi waktu jika menggunakan fitur lokal yang sudah disaring terlebih dahulu. Walaupun akurasi juga menurun tetapi juga terjadi penurunan yang signifikan pada nilai akurasi.

### 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. Perbandingan tersebut dapat dilihat pada tabel berikut:

	<b>Keseluruhan</b>		<b>Disaring</b>	
	Total Waktu (detik)	Akurasi (%)	Total Waktu (detik)	Akurasi (%)
<b>Dataset 400</b>	82.15	98	31.78	96
<b>Dataset 600</b>	80.96	96	31.41	89

Tabel di atas menunjukkan bahwa penggunaan Dataset 400 memberikan hasil yang lebih baik dari Dataset 600. Terutama pada saat penggunaan dataset yang disaring, akurasi dari Dataset 400 mencapai 96% sedangkan untuk Dataset 600 hanya 89% dengan perbedaan waktu yang sangat sedikit.

Dataset 400 memiliki hasil yang lebih baik kemungkinan dikarenakan ukuran gambar pada Dataset 400 lebih mendekati ukuran dari gambar-gambar di data *test*. Karena ukuran yang mirip tersebut maka fitur lokal yang dihasilkan memiliki ciri yang lebih mirip.

## 3.5 Analisis Pengaruh Parameter Agglomerative Clustering terhadap Sebaran Nilai Keunikan dan Konsistensi

Nilai keunikan dan konsistensi dari hasil *clustering* pada tahap sebelumnya tidak tersebar secara merata. Terutama pada nilai keunikan mayoritas data memiliki nilai keunikan 1. Karena sebaran yang terpusat pada nilai 1 ini penyaringan data dengan mengatur nilai keunikan tidak memberikan perbedaan berarti.

Analisis yang dilakukan pada 3.4 menggunakan nilai keunikan dan konsistensi yang didapat dengan menggunakan metode *Agglomerative Clustering* sesuai dengan yang dilakukan pada 3.3. Metode *Agglomerative Clustering* yang dilakukan pada 3.3 membagi *cluster* berdasarkan pada parameter *distance\_threshold* yang di mana pada analisis tersebut didapat dari setengah nilai rata-rata jarak antar data sampel. Pada analisis ini akan diuji apakah dengan mengubah nilai *distance\_threshold* dapat mengubah sebaran nilai keunikan dan konsistensi sehingga jumlah fitur lokal yang digunakan dapat lebih mudah diubah dengan mengatur nilai batas konsistensi dan keunikan.

Dataset yang digunakan pada analisis ini sama seperti analisis sebelumnya yaitu menggunakan dataset Book Covers tetapi hanya menggunakan satu ukuran yaitu Dataset 400. Dataset 400 digunakan pada penelitian ini karena dari hasil sebelumnya didapat bahwa Dataset ini memberikan hasil yang terbaik.

### 3.5.1 Ide Analisis dan Tahapan

Analisis dilakukan dengan mengubah nilai parameter `distance_threshold` di *Agglomerative Clustering* menjadi nilai rata-rata jarak antar data sampel. Nilai `distance_threshold` yang lebih tinggi akan membuat lebih mudah untuk data menjadi tergabung dalam satu *cluster*. *Cluster-cluster* yang dihasilkan akan lebih sedikit dan dengan jumlah data di tiap *cluster*-nya lebih banyak.

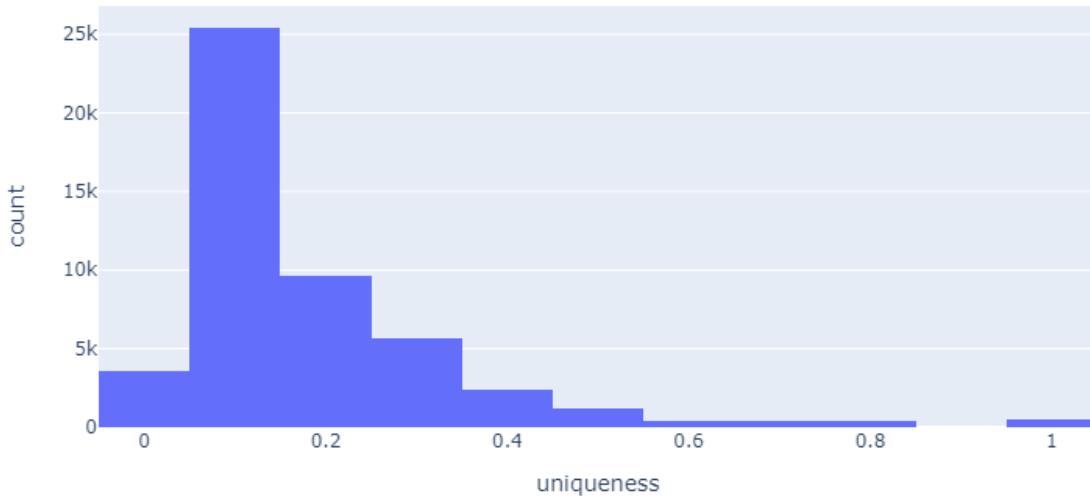
Analisis akan dilakukan pada bagaimana sebaran nilai keunikan dan konsistensi. Bagaimana menentukan batas dari sebaran nilai tersebut dan apakah fitur lokal yang didapat dari hasil penyaringan akan menghasilkan nilai akurasi yang berbeda jika digunakan sebagai dataset dalam identifikasi dengan BSIS.

Tahapan yang dilakukan sama dengan tahapan pada [3.3](#) untuk menghitung nilai keunikan dan konsistensi. Setelah didapatkan nilai tersebut dilakukan identifikasi dengan BSIS dengan tahapan yang sama dengan tahapan pada [3.4](#).

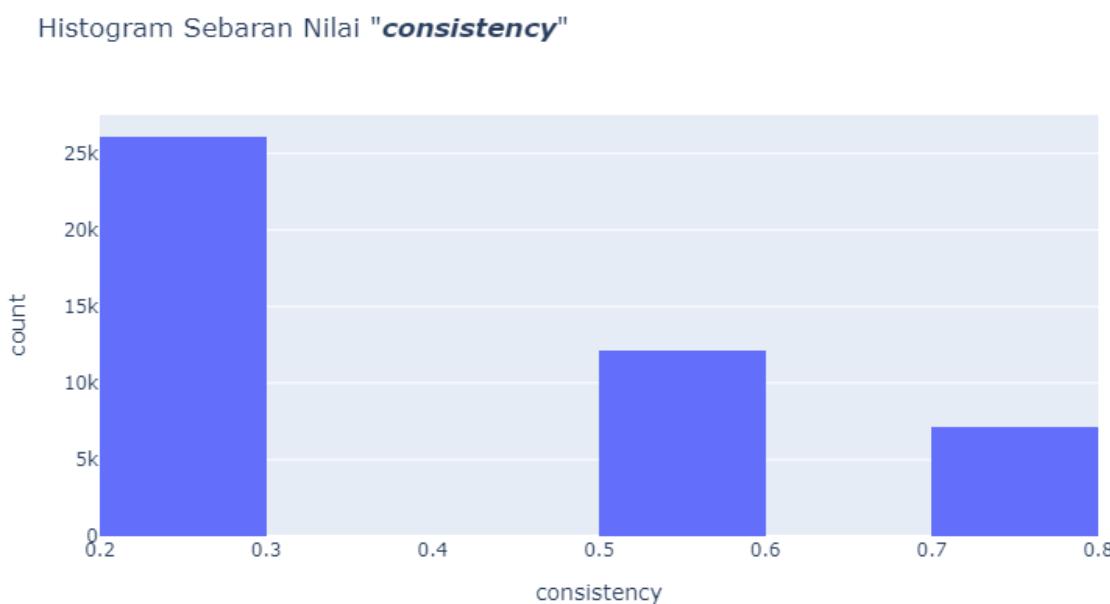
### 3.5.2 Hasil Analisis

Setelah dilakukan *clustering* untuk menghitung nilai keunikan dan konsistensi didapat sebaran nilai seperti pada gambar-gambar berikut ini. Gambar [3.22](#) menunjukkan sebaran nilai keunikan dan Gambar [3.23](#) menunjukkan sebaran nilai konsistensi.

Histogram Sebaran Nilai "**uniqueness**"



Gambar 3.22: Sebaran nilai keunikan (**uniqueness**) untuk Dataset 400 saat menggunakan parameter *Agglomerative Clustering* yang berbeda.



Gambar 3.23: Sebaran nilai keunikan (*consistency*) untuk Dataset 400 saat menggunakan parameter *Agglomerative Clustering* yang berbeda.

Sebaran nilai konsistensi seperti yang terlihat pada Gambar 3.23 memiliki pola yang sama seperti sebaran pada saat menggunakan setengah rata-rata jarak sebagai *distance\_threshold*. Sedangkan untuk nilai keunikan sebaran datanya memiliki pola yang berbeda dari analisis sebelumnya saat menggunakan setengah rata-rata jarak. Pada analisis sebelumnya tingkat fitur lokal dengan tingkat keunikan 1 mendominasi dataset, tetapi pada analisis ini tingkat keunikan fitur lokal justru mayoritas berada di kisaran yang rendah (0.05 - 0.2).

Pada analisis sebelumnya karena mayoritas nilai keunikan ada pada nilai 1 maka perubahan pada batas bawah nilai keunikan saat melakukan penyaringan terhadap fitur lokal tidak akan berdampak pada hasil fitur lokal yang didapat. Tetapi pada penelitian kali ini karena sebaran nilai keunikannya lebih merata dan mayoritas tersebar pada rentang nilai-nilai kecil maka perubahan nilai batas bawah untuk menyaring fitur lokal akan berpengaruh pada jumlah fitur lokal yang didapat.

### 3.6 Analisis Pengaruh Batas Nilai Keunikan yang Digunakan terhadap Jumlah Fitur Lokal yang Didapat dan Tingkat Akurasi BSIS

Pada analisis di 3.5 didapatkan bahwa dengan menggunakan nilai parameter *distance\_threshold* di *Agglomerative Clustering* yang lebih besar menyebabkan *cluster-cluster* yang dihasilkan memiliki ciri berbeda. *Cluster* dengan ciri berbeda tersebut mempengaruhi nilai keunikan dari fitur lokal. Nilai keunikan fitur lokal jadi tersebar secara lebih merata dan terpusat pada nilai pada rentang yang kecil.

Nilai fitur lokal yang lebih tersebar memungkinkan untuk dilakukan penyaringan dengan batas bawah yang berbeda dan didapatkan hasil yang berbeda. Pada bagian ini akan dilakukan analisis untuk mengetahui bagaimana pengaruh penentuan nilai batas bawah terhadap jumlah fitur lokal yang tersaring dan bagaimana pengaruhnya terhadap tingkat akurasi saat data tersebut digunakan untuk identifikasi dengan BSIS.

### 3.6.1 Ide Analisis dan Tahapan

Penelitian akan dilakukan dengan melakukan tahapan yang sama seperti pada penelitian di 3.4. Tetapi pada penelitian ini yang dibandingkan adalah nilai akurasi dan total waktu dari beberapa hasil penyaringan dataset dengan menggunakan batas nilai keunikan yang berbeda. Untuk setiap batas nilai keunikan yang digunakan akan dihitung berapa fitur lokal yang tersisa dan nilai akurasi serta total waktu pemrosesan.

Pada penelitian ini akan digunakan 3 batas nilai keunikan yang berbeda yaitu: 0.2, 0.4, dan 0.6 sedangkan untuk batas nilai konsistensi hanya akan digunakan 0.49 untuk setiap batas nilai keunikan. Dataset untuk data *train* yang digunakan adalah dataset yang sama dengan Dataset 400 pada 3.4. Sedangkan untuk data *test* digunakan data yang sama dengan dataset *test* pada 3.4.

Tahapan analisis yang dilakukan adalah sebagai berikut:

1. *Load* Gambar

*Load* semua gambar *test* yang digunakan.

2. *Load* Dataset

*Load* dataset yang digunakan (Dataset 400) dan saring data tersebut berdasarkan batas nilai keunikan serta konsistensi yang digunakan pada tahap ini. Dihitung juga jumlah data yang tersaring.

3. Identifikasi dengan BSIS

Lakukan identifikasi BSIS pada gambar *test* dengan menggunakan dataset yang telah disaring.

4. Penghitungan Akurasi Serta

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

Langkah-langkah tahapan tersebut dilakukan untuk setiap batas keunikan yang digunakan (0.2, 0.4, dan 0.6). Hasilnya lalu disimpan dan dibandingkan.

### 3.6.2 Hasil Analisis

Setelah dilakukan seluruh tahapan di atas didapatkan hasil sebagai seperti pada Tabel 3.6. Ditulis juga total waktu dan akurasi

Batas Nilai Keunikan	Jumlah Fitur Lokal	Total Waktu (detik)	Akurasi (%)
0.2	11627	30.31	95
0.4	3484	17.12	63
0.6	1232	13.20	37

Tabel 3.6: Perbedaan hasil dari setiap penggunaan batas nilai keunikan yang berbeda.

Dari tabel tersebut terlihat bahwa penggunaan nilai batas yang tinggi memberikan jumlah fitur lokal yang semakin sedikit. Fitur lokal yang sedikit membuat pemrosesan waktu menjadi lebih cepat tetapi juga berdampak pada akurasi. Penggunaan jumlah fitur lokal yang terlalu sedikit menyebabkan nilai akurasi menjadi sangat rendah seperti ditunjukkan pada penggunaan nilai batas 0.4 dan 0.6.

Berdasarkan data Tabel 3.6 dapat disimpulkan bahwa perlu untuk memilih batas nilai yang tepat untuk mendapatkan hasil deteksi yang cepat dan dengan akurasi yang tinggi. Seperti penggunaan nilai keunikan 0.2, fitur lokal yang didapat cukup berbeda jumlahnya dengan jika digunakan semua fitur lokal dan waktu pemrosesan juga lebih singkat secara signifikan (penggunaan semua fitur lokal memiliki waktu pemrosesan selama 82.15), tetapi nilai akurasinya masih termasuk tinggi.



## BAB 4

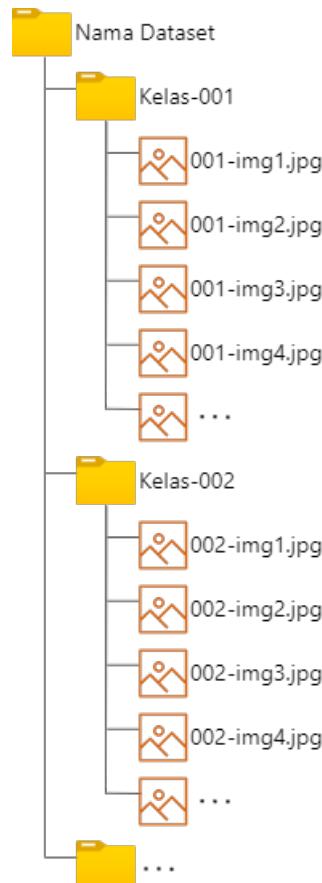
# PERANCANGAN

### 4.1 Perancangan 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 masukkan yang digunakan perlu untuk memiliki sebuah struktur *folder* tertentu. Keterangan struktur *folder* masukkan serta fungsi-fungsi dalam modul hingga struktur *file* keluaran akan dijelaskan pada subbab-subbab berikut ini.

#### 4.1.1 Rancangan Struktur Folder

Rancangan struktur *folder* yang diperlukan untuk dapat digunakan sebagai masukkan pada modul *clustering* adalah sebagai berikut, dapat dilihat pada Gambar 4.1.

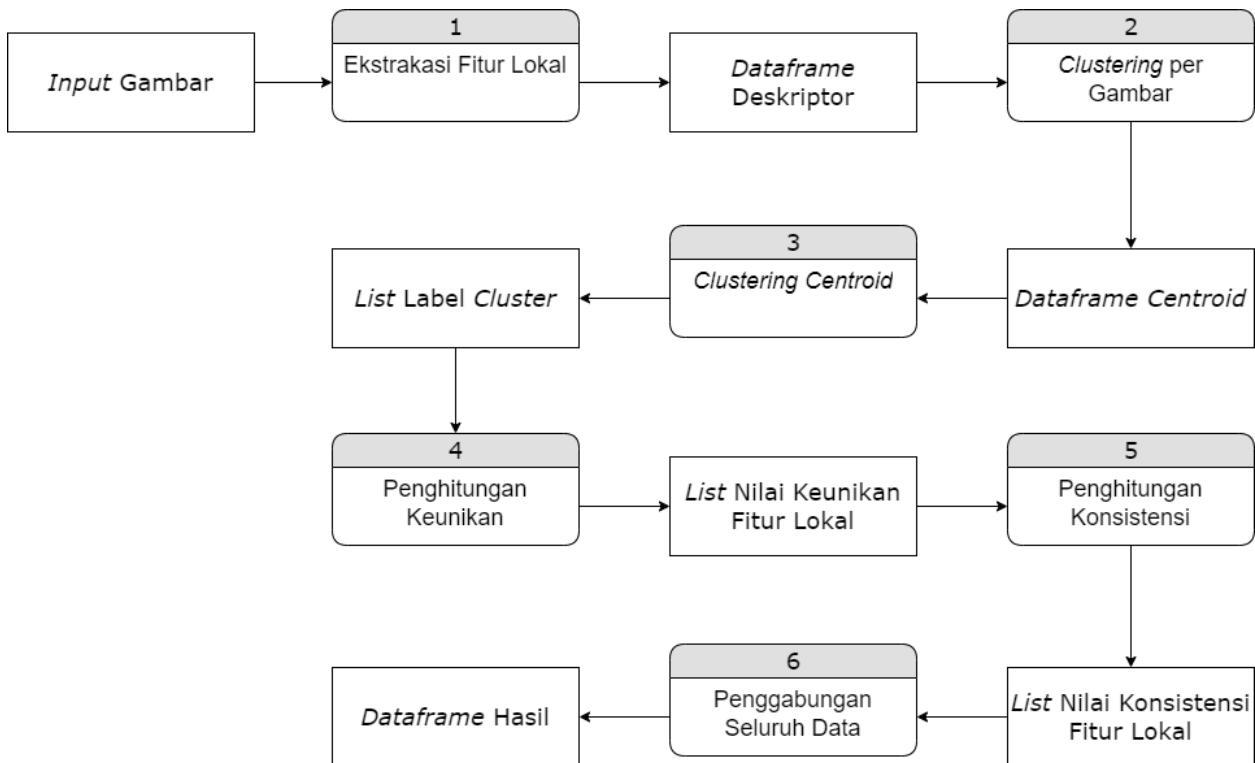


Gambar 4.1: Rancangan struktur *folder* untuk pembuatan model.

Gambar tersebut menunjukkan struktur *folder* masukkan 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*.

#### 4.1.2 Rancangan Proses Clustering

Proses *clustering* menerima dataset gambar sesuai dengan struktur *folder* yang telah dijelaskan sebelumnya pada 4.1.1. 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 4.2.



Gambar 4.2: Tahapan proses *clustering* hingga didapat nilai keunikan dan konsistensi.

Langkah-langkah tersebut merupakan langkah-langkah implementasi yang dilakukan pada tahap pencarian nilai konsistensi dan keunikan yang dilakukan di 3.3. *Clustering* yang dilakukan pada langkah 2 dan 3 dilakukan dengan menggunakan fungsi `agglo_cluster` dari kelas `Util` yang dijelaskan pada 4.2. 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:   masukkan  $v$  ke consistency
8: end for
9: return consistency

```

---

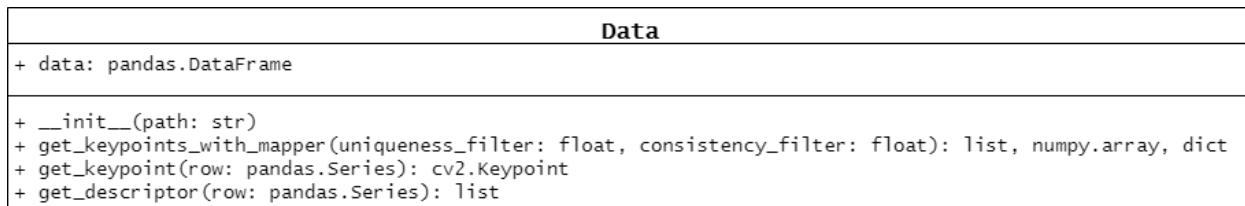
Proses yang dilakukan pada Gambar 4.2 akan menghasilkan sebuah *dataframe* dengan ketentuan kolom seperti pada Tabel 4.1. *Dataframe* tersebut disimpan ke dalam *file pickle* dengan menggunakan *library* Pickle di Python.

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 <b>cluster</b> 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>
kp_point_x	Atribut <i>keypoint</i>	Koordinat x dari <i>keypoint</i>
kp_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*.

## 4.2 Rancangan Kelas Data

Kelas **Data** adalah kelas yang digunakan untuk membaca file **pickle** berisi *dataframe* yang dihasilkan dari proses pada 4.1.2. Kelas ini membaca data dan memiliki beberapa *method* yang digunakan untuk mengubah data dalam *dataframe* menjadi format yang dapat digunakan. Diagram untuk kelas **Data** dapat dilihat pada Gambar 4.3



Gambar 4.3: Diagram kelas **Data**.

Atribut **data** pada kelas tersebut adalah *dataframe* yang dihasilkan dari modul sebelumnya dan dibaca dengan kelas ini. Kegunaan fungsi-fungsi dari kelas **Data** adalah sebagai berikut:

1. **\_\_init\_\_**

Fungsi konstruktor kelas **Data** menerima parameter **path** yang merupakan direktori tempat *file dataframe* yang ingin digunakan. Fungsi akan memasukkan *file* pada direktori **path** ke atribut **data**.

2. **get\_keypoints\_with\_mapper**

Fungsi menyaring **data** dengan menetapkan batas bawah pada kolom **uniqueness** dan

*consistency*. Fungsi ini memberikan tiga keluaran, yaitu sebuah *list* berisi objek *keypoint*, sebuah *array* 2 dimensi berisi deskriptor, dan sebuah *dictionary* yang menunjukkan asal gambar dari setiap *keypoint*.

### 3. get\_keypoint

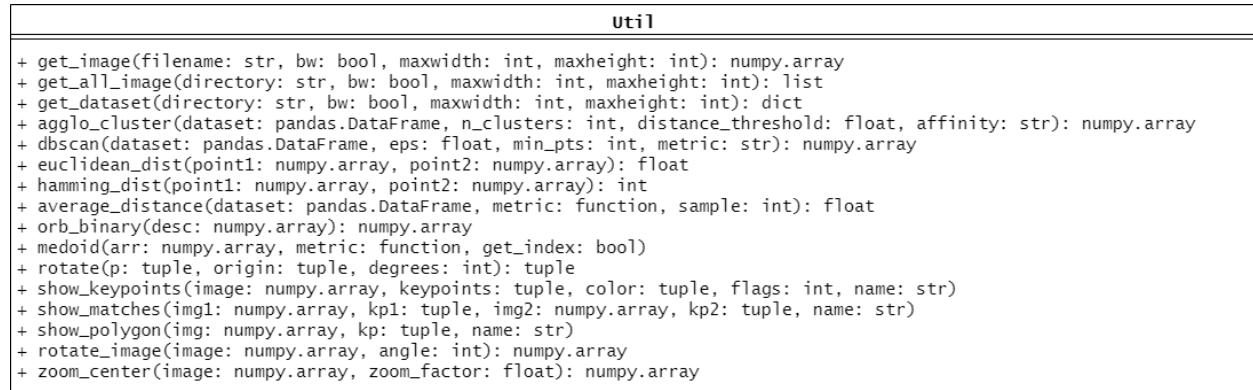
Fungsi berguna untuk membuat objek *keypoint* dari sebuah baris *dataframe*. Fungsi ini mengambil nilai dari kolom `kp_point_x`, `kp_point_y`, `size`, `angle`, `response`, `octave`, `class_id` dan membuat objek *keypoint* dari nilai-nilai tersebut.

### 4. get\_descriptor

Fungsi mengambil 128 kolom pertama dari sebuah baris *dataframe* dan mengembalikan *list* berisi nilai-nilai tersebut.

## 4.3 Rancangan Kelas Util

Kelas *Util* berisi fungsi-fungsi yang banyak 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 4.4.



Gambar 4.4: 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 4.1.1. Fungsi ini digunakan untuk mendapatkan gambar masukkan pada 4.1.2.

### 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 3.3 dengan mengikuti rancangan pada 4.1.2.

### 5. show\_keypoints

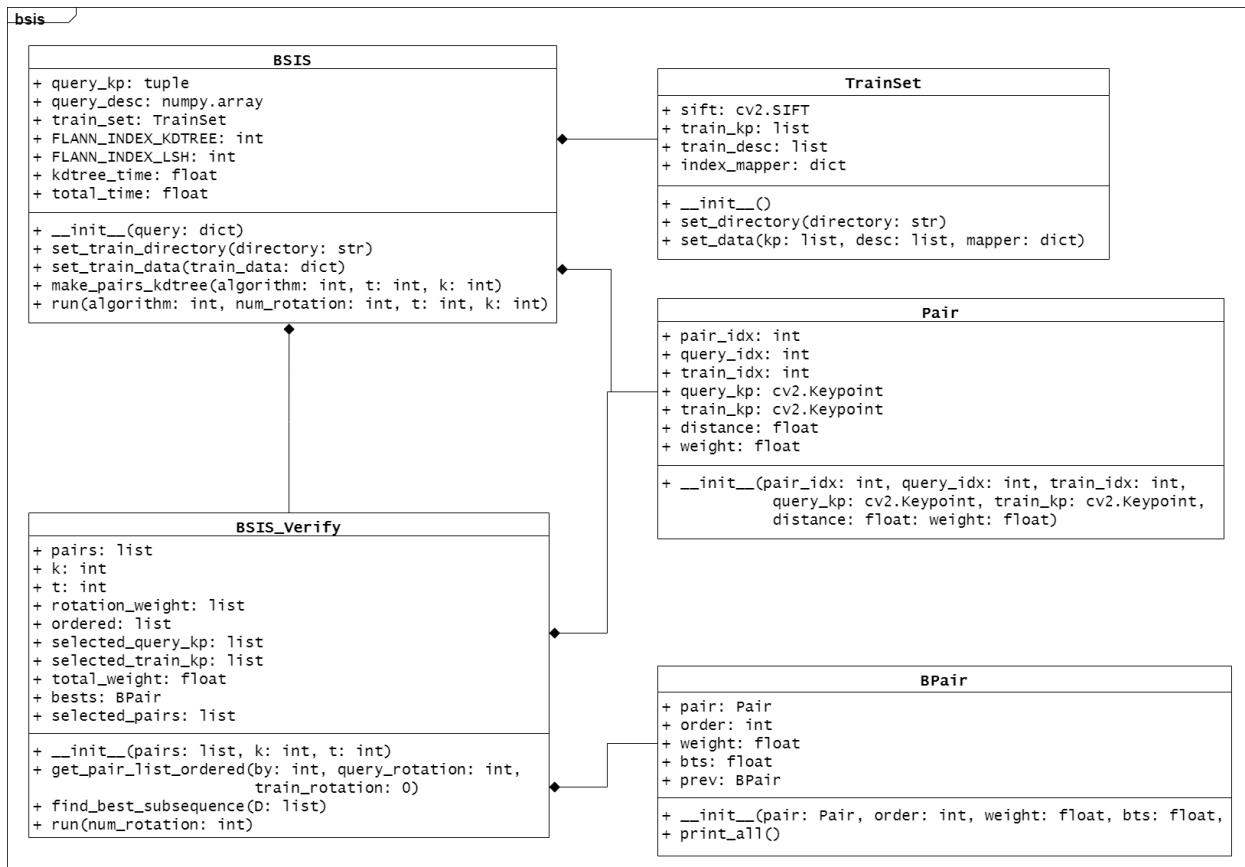
Fungsi untuk menampilkan *keypoint* pada gambar. Fungsi digunakan untuk melakukan visualisasi *keypoint* di gambar yang dilakukan pada 3.3.

### 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](#).

## 4.4 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 [4.5](#). Kelas-kelas pada *package* ini digunakan untuk melakukan BSIS pada [3.4](#) dan [3.6](#) dengan memanggil kelas BSIS.



Gambar 4.5: Diagram *Package* `bsis`.

Untuk melakukan BSIS terhadap sebuah gambar pertama perlu dibuat sebuah objek kelas BSIS. Pembuatan objek BSIS menerima satu masukkan 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 sebuah *dictionary* yang dihasilkan oleh fungsi `get_keypoints_with_mapper` dari kelas Data. Setelah objek BSIS telah mendapatkan gambar masukkan 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
```

---



## 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 operator import itemgetter
8  import time
9  from glob import glob
10 import os
11
12 import util
13
14 #%%
15 class BSIS:
16     def __init__(self, query):
17         self.query_kp = query['kp']
18         self.query_desc = query['desc']
19
20         self.train_set = None
21
22         self.FLANN_INDEX_KDTREE = 1
23         self.FLANN_INDEX_LSH = 6
24
25         self.kdtree_time = 0
26         self.total_time = 0
27
28     def set_train_directory(self, directory):
29         train = TrainSet()
30         train.set_directory(directory)
31         self.train_set = train
32
33     def set_train_data(self, train_data):
34         kp = train_data[0]
35         desc = train_data[1]
36         mapper = train_data[2]
37         train = TrainSet()
38         train.set_data(kp, desc, mapper)
39         self.train_set = train
40
41     def make_pairs_kdtree(self, algorithm, t=4, k=100):
42         index_params = dict(algorithm=algorithm, trees=5)
43         # search_params = dict(checks=50)    # or pass empty dictionary
44         search_params = dict()
45         self.flann = cv2.FlannBasedMatcher(index_params, search_params)
46
47         start_time = time.time()
48         matches = self.flann.knnMatch(self.query_desc, self.train_set.train_desc, k=k)
49         self.kdtree_time = time.time() - start_time
50         print('make_pairs_kdtree---', self.kdtree_time)
51
52         pairs = dict()
53         pair_idx = 0
54         for m in matches:
55             m_dists = list()
56             for i in m:
57                 m_dists.append(i.distance)
58
59             st_dev = np.std(m_dists)
60             mean = np.mean(m_dists)
61             thres = mean - (t * st_dev)
62
63             for i in m:
64                 if i.distance < thres:
65                     weight = ((i.distance - mean) / st_dev) ** 2
66                     if self.train_set.index_mapper[i.trainIdx] not in pairs.keys():
67                         pairs[self.train_set.index_mapper[i.trainIdx]] = list()
68
69                     pairs[self.train_set.index_mapper[i.trainIdx]].append(
70                         BSIS.Verify.Pair(
71                             pair_idx,
72                             i.queryIdx,
73                             i.trainIdx,
74                             self.query_kp[i.queryIdx],
75                             self.train_set.train_kp[i.trainIdx],
```

```

76             i.distance,
77             weight
78         )
79     )
80     pair_idx += 1
81
82     return pairs
83
84 def run(self, algorithm, num_rotation=1, t=4, k=100):
85     start_time = time.time()
86     self.pairs = self.make_pairs_kdtree(algorithm=algorithm, t=t, k=k)
87     self.result = dict()
88
89     maximum = ('', 0)
90     for k, v in self.pairs.items():
91         bsis = BSIS_Verify(v)
92         bsis.run(num_rotation=num_rotation)
93         self.result[k] = {'query_kp': bsis.selected_query_kp, 'train_kp': bsis.selected_train_kp, 'total_weight': bsis.
94                           total_weight}
95         if bsis.total_weight > maximum[1]:
96             maximum = (k, bsis.total_weight)
97
98     self.total_time = time.time() - start_time
99     print('total_time---', self.total_time)
100    return maximum[0]
101
102 #%%
103 class TrainSet:
104     def __init__(self):
105         self.sift = cv2.xfeatures2d.SIFT_create()
106
107         self.train_kp = list()
108         self.train_desc = list()
109         self.index_mapper = dict()
110
111     def set_directory(self, directory):
112         index = 0
113         for filename in glob(os.path.join(directory, '*.jpg')):
114             fname = filename.split('\\')[-1]
115             kp, desc = self.sift.detectAndCompute(util.get_image(filename), None)
116             for k, d in zip(kp, desc):
117                 self.train_kp.append(k)
118                 self.train_desc.append(d)
119                 self.index_mapper[index] = fname
120                 index += 1
121
122         self.train_desc = np.array(self.train_desc)
123
124     def set_data(self, kp, desc, mapper):
125         self.train_kp = kp
126         self.train_desc = desc
127         self.index_mapper = mapper
128
129 #%%
130 class BSIS_Verify:
131     def __init__(self, pairs, k=100, t=4):
132         self.pairs = pairs
133
134         self.k = k
135         self.t = t
136
137 #Class untuk menyimpan pasangan antar keypoint
138 class Pair:
139     def __init__(self, pair_idx, query_idx, train_idx, query_kp, train_kp, distance, weight):
140         self.pair_idx = pair_idx
141         self.query_idx = query_idx
142         self.train_idx = train_idx
143         self.query_kp = query_kp
144         self.train_kp = train_kp
145         self.distance = distance
146         self.weight = weight
147
148 #Class untuk menyimpan informasi pasangan keypoint yang diperlukan pada tahap verifikasi BSIS
149 class BPair:
150     def __init__(self, pair, order, weight, bts, prev):
151         self.pair = pair
152         self.order = order
153         self.weight = weight
154         self.bts = bts
155         self.prev = prev
156
157     def print_all(self):
158         try:
159             print(self.pair.pair_idx, self.order, self.weight, self.bts, self.prev.idx)
160         except:
161             print(self.pair.pair_idx, self.order, self.weight, self.bts, None)
162
163     def get_pair_list_ordered(self, by=0, query_rotation=0, train_rotation=0):
164         paired_query_kp = set([(x.query_idx, x.query_kp.pt[0], x.query_kp.pt[1]) for x in self.pairs])
165         query_origin = (np.mean([i[1] for i in paired_query_kp]), np.mean([i[2] for i in paired_query_kp]))
166         query_order_x_set = [(i[0], util.rotate((i[1], i[2]), query_origin, query_rotation)[by]) for i in paired_query_kp]
167         query_order_x_set = sorted(query_order_x_set, key=itemgetter(1))
168         query_order_x = dict()
169         for i, idx in enumerate(query_order_x_set):
170             query_order_x[idx[0]] = i
171
172         paired_train_kp = set([(x.train_idx, x.train_kp.pt[0], x.train_kp.pt[1]) for x in self.pairs])
173         train_origin = (np.mean([i[1] for i in paired_train_kp]), np.mean([i[2] for i in paired_train_kp]))
174         train_order_x_set = [(i[0], util.rotate((i[1], i[2]), train_origin, train_rotation)[by]) for i in paired_train_kp]

```

```

174     train_order_x_set = sorted(train_order_x_set, key=itemgetter(1))
175     # train_order_x = dict()
176     # for i, idx in enumerate(train_order_x_set):
177     #     train_order_x[idx[0]] = i
178
179     train_idx_dict = dict()
180     for p in self.pairs:
181         if p.train_idx not in train_idx_dict.keys():
182             train_idx_dict[p.train_idx] = list()
183             train_idx_dict[p.train_idx].append(p)
184         else:
185             train_idx_dict[p.train_idx].append(p)
186
187     ordered_list = list()
188     for i, j in train_order_x_set:
189         one_kp_pairs = train_idx_dict[i]
190         col_list = list()
191         for p in one_kp_pairs:
192             col_list.append(self.BPair(p, query_order_x[p.query_idx], p.weight, p.weight, None))
193         ordered_list.append(col_list)
194
195     return ordered_list
196
197 def find_best_subsequence(self, D):
198     # start_time = time.time()
199     best_subsequence = self.BPair(0, 0, 0, 0, None)
200     for i in range(0, len(D)):
201         for j in range(0, len(D[i])):
202             # start_time2 = time.time()
203             dBestPrev = self.BPair(0, 0, 0, 0, None)
204             for k in range(0, i):
205                 for l in range(0, len(D[k])):
206                     if D[k][l].order < D[i][j].order and D[k][l].bts > dBestPrev.bts:
207                         dBestPrev = D[k][l]
208             # print('find dBestPrev {} - {}'.format(i, j), time.time() - start_time2)
209             D[i][j].bts = D[i][j].weight + dBestPrev.bts
210             D[i][j].prev = dBestPrev
211
212             if D[i][j].bts + D[i][j].weight > best_subsequence.bts + best_subsequence.weight:
213                 best_subsequence = D[i][j]
214
215     # print('find_best_subsequence', time.time() - start_time)
216     return best_subsequence
217
218 def run(self, num_rotation=1):
219     max_weight = 0
220     selected_query_kp = list()
221     selected_train_kp = list()
222     self.rotation_weight = list()
223     rotate_by = int(360.0 / num_rotation)
224     for i in range(0, 360, rotate_by):
225         query_rotation = i
226         train_rotation = 0
227         #print('query_rotation: ', query_rotation, '---', 'train_rotation:', train_rotation)
228
229         get_pair_list_start = time.time()
230         self.ordered = self.get_pair_list_ordered(query_rotation=query_rotation, train_rotation=train_rotation)
231
232         if len(self.ordered) < 1:
233             self.selected_query_kp = []
234             self.selected_train_kp = []
235             self.total_weight = 0
236             continue
237         #print('get_pair_list_ordered ---', time.time() - get_pair_list_start)
238
239         find_best_start = time.time()
240         self.bests = self.find_best_subsequence(self.ordered)
241         self.selected_pairs = list()
242         while(self.bests.prev != None):
243             self.selected_pairs.append(self.bests.pair)
244             self.bests = self.bests.prev
245         else:
246             pass
247             #self.selected_pairs.append(self.bests.pair)
248         #print('find_best_subsequence ---', time.time() - find_best_start)
249
250         weight = sum([i.weight for i in self.selected_pairs])
251         self.rotation_weight.append((i, weight))
252         if weight > max_weight:
253             max_weight = weight
254             selected_query_kp = [i.query_kp for i in self.selected_pairs]
255             selected_train_kp = [i.train_kp for i in self.selected_pairs]
256
257         self.selected_query_kp = selected_query_kp
258         self.selected_train_kp = selected_train_kp
259         self.total_weight = max_weight

```

Kode A.2: data\_.py

```

1 import pickle
2 import pandas as pd
3 import numpy as np
4 from util import Image
5 import cv2
6
7 class Data:
8     def __init__(self, path):

```

```

9    self.data = pickle.load(open(path, 'rb'))
10
11    def get_keypoints_list(self, uniqueness_filter=0.0, consistency_filter=0.0):
12        train_image = list()
13        data_group = self.data.groupby('img')
14        for k, v in data_group:
15            selected_feature = v[(v['uniqueness'] > uniqueness_filter) & (v['consistency'] > consistency_filter)]
16            if len(selected_feature.index) > 0:
17                keypoints = list()
18                descriptors = list()
19                group = selected_feature['img_class'].iloc[0]
20                for idx, row in selected_feature.iterrows():
21                    descriptors.append(self.get_descriptor(row))
22                    keypoints.append(self.get_keypoint(row))
23                train_image.append(Image(k, group, tuple(keypoints), np.array(descriptors, dtype='float32')))
24
25        return train_image
26
27    def get_keypoints_dict(self, uniqueness_filter=0.0, consistency_filter=0.0):
28        train_image = dict()
29        data_group = self.data.groupby('img')
30        for k, v in data_group:
31            selected_feature = v[(v['uniqueness'] > uniqueness_filter) & (v['consistency'] > consistency_filter)]
32            if len(selected_feature.index) > 0:
33                keypoints = list()
34                descriptors = list()
35                group = selected_feature['img_class'].iloc[0]
36                for idx, row in selected_feature.iterrows():
37                    descriptors.append(self.get_descriptor(row))
38                    keypoints.append(self.get_keypoint(row))
39                train_image[k] = Image(k, group, tuple(keypoints), np.array(descriptors, dtype='float32'))
40
41        return train_image
42
43    def get_keypoints_with_mapper(self, uniqueness_filter=0.0, consistency_filter=0.0):
44        train_kp = list()
45        train_desc = list()
46        index_mapper = dict()
47        index = 0
48        data_group = self.data.groupby('img')
49        for k, v in data_group:
50            selected_feature = v[(v['uniqueness'] > uniqueness_filter) & (v['consistency'] > consistency_filter)]
51            if len(selected_feature.index) > 0:
52                group = selected_feature['img_class'].iloc[0]
53                for idx, row in selected_feature.iterrows():
54                    train_desc.append(self.get_descriptor(row))
55                    train_kp.append(self.get_keypoint(row))
56                    index_mapper[index] = k
57                index += 1
58
59        train_desc = np.array(train_desc, dtype='float32')
60
61        return train_kp, train_desc, index_mapper
62
63    def get_keypoint(self, row):
64        keypoint = cv2.KeyPoint(
65            x=row['kp_point_x'],
66            y=row['kp_point_y'],
67            size=row['size'],
68            angle=row['angle'],
69            response=row['response'],
70            octave=row['octave'],
71            class_id=row['class_id']
72        )
73
74        return keypoint
75
76    def get_descriptor(self, row):
77        descriptor = row[0:128].tolist()
78        return descriptor

```

Kode A.3: clustering\_sift\_agglo.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 cv2
10 import pandas as pd
11 import numpy as np
12 import util
13 import pickle
14
15 #%%
16 dataset = util.get_dataset('datasets/book_covers_half/', maxheight=400, maxwidth=400)
17
18 #%% Detect keypoint
19 sift = cv2.xfeatures2d.SIFT_create(250)
20
21 keypoints = list()
22 descriptors = pd.DataFrame()
23 labels = list()
24

```

```

25 #buat 128 kolom baru
26 for i in range(128):
27     descriptors.insert(i, i, 0.0)
28
29 #buat kolom untuk menyimpan gambar asal
30 kp_index = 0
31 for group in dataset.keys():
32     for i, img in enumerate(dataset[group]):
33         kp, desc = sift.detectAndCompute(img[1], None)
34         #hasil deteksi deskriptor menjadi dataframe
35         desc_1 = pd.DataFrame(desc)
36         #masukkan ke dataframe dataset
37         descriptors = descriptors.append(desc_1, ignore_index=True)
38         for i, k in enumerate(kp):
39             #masukkan keypoint ke list
40             keypoints.append(k)
41             #label untuk tiap deskriptor, berisi: nama gambar, kelas gambar, dan index keypoint pada list
42             labels.append((group, img[0], kp_index))
43             kp_index += 1
44
45 #masukkan labels ke dalam dataframe
46 descriptors['img'] = [i[1] for i in labels]
47 descriptors['img_class'] = [i[0] for i in labels]
48 descriptors['kp_idx'] = [i[2] for i in labels]
49
50 #%% cluster per image
51 img_group = descriptors.groupby('img')
52
53 #one_image_dict = dict()
54 descs_cluster_label = list()
55 df_centroid = pd.DataFrame()
56
57 for image in img_group.groups.keys():
58     one_image = img_group.get_group(image)
59     one_image_desc_only = one_image.drop(columns=['img', 'img_class', 'kp_idx'])
60     sample_size = 50
61     if len(one_image_desc_only.index) < 50:
62         sample_size = None
63     threshold = util.average_distance(
64         one_image_desc_only,
65         sample=sample_size,
66         metric=util.euclidean_dist
67     )
68     one_image_cluster_label = util.aggro_cluster(
69         one_image_desc_only,
70         distance_threshold=threshold/2,
71         affinity='euclidean'
72     )
73     one_image['cluster_label'] = one_image_cluster_label
74     descs_cluster_label = descs_cluster_label + list(one_image_cluster_label)
75
76 cluster_label_group = one_image.groupby('cluster_label')
77
78 labels_list = list()
79 centroid_arr = np.empty([0, 128])
80 labels_list = list()
81 for label in cluster_label_group.groups.keys():
82     one_cluster = cluster_label_group.get_group(label).drop(columns=['cluster_label', 'img', 'img_class', 'kp_idx'])
83     mean_arr = np.array([np.mean(one_cluster.values, axis=0)])
84     centroid_arr = np.concatenate((centroid_arr, mean_arr))
85     labels_list.append(label)
86
87 centroid_df = pd.DataFrame(centroid_arr)
88 centroid_df['cluster_label'] = labels_list
89 centroid_df['img'] = [image for i in range(len(centroid_df))]
90 centroid_df['img_class'] = one_image['img_class'][0:len(centroid_df.index)].tolist()
91 df_centroid = df_centroid.append(centroid_df)
92
93 descriptors['cluster_label'] = descs_cluster_label
94
95 #%% cluster centroid
96 threshold = util.average_distance(
97     df_centroid.drop(columns=['img', 'img_class', 'cluster_label']),
98     sample=100,
99     metric=util.euclidean_dist
100 )
101 cluster2_labels = util.aggro_cluster(
102     df_centroid.drop(columns=['img', 'img_class', 'cluster_label']),
103     distance_threshold=threshold/2,
104     affinity='euclidean'
105 )
106
107 df_centroid['cluster2_label'] = cluster2_labels
108
109 #%% cluster2 class count
110 cluster2_class_count = dict()
111 cluster2_group = df_centroid.groupby('cluster2_label')
112
113 for c2 in cluster2_group.groups.keys():
114     one_cluster2 = cluster2_group.get_group(c2)
115     one_cluster2 = one_cluster2.drop_duplicates(subset='img')
116     cluster2_class_count[c2] = one_cluster2['img_class'].value_counts(normalize=True)
117
118 #%% put class count in df_centroid
119 img_class_count = [cluster2_class_count[c2lab][imgc] for c2lab, imgc in zip(df_centroid.cluster2_label, df_centroid.img_class)]
120 df_centroid['uniqueness'] = img_class_count
121
122 #%% count number of different images
123 cluster2_img_class_group = df_centroid.groupby(['cluster2_label', 'img_class']).img.unique()

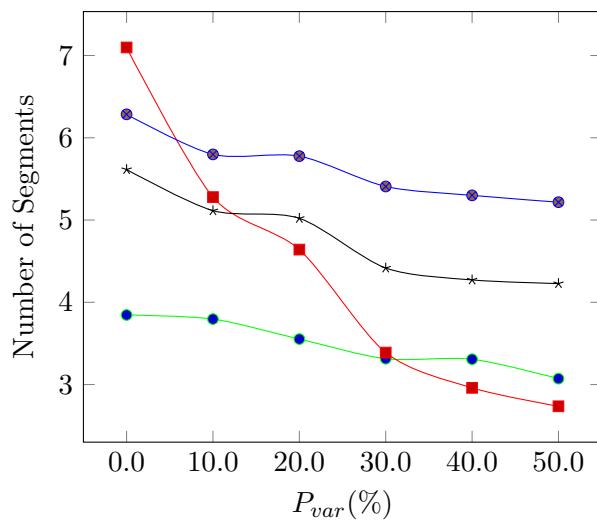
```

```
124| img_count = list()  
125|  
126| for c2l, ic in zip(df_centroid.cluster2_label, df_centroid.img_class):  
127|     img_in_class = cluster2_img_class_group[c2l][ic]  
128|     img_count.append(img_in_class / 4)  
129|  
130| df_centroid['consistency'] = img_count  
131|  
132| #%% join one image with centroid  
133| df_centroid_val = df_centroid[['img', 'cluster_label', 'cluster2_label', 'uniqueness', 'consistency']]  
134| descriptors = pd.merge(descriptors, df_centroid_val, on=['img', 'cluster_label'])  
135|  
136| #%% save keypoints details  
137| point_x = list()  
138| point_y = list()  
139| size = list()  
140| angle = list()  
141| response = list()  
142| octave = list()  
143| class_id = list()  
144| for k in descriptors.kp_idx:  
145|     kp = keypoints[k]  
146|     point_x.append(kp.pt[0])  
147|     point_y.append(kp.pt[1])  
148|     size.append(kp.size)  
149|     angle.append(kp.angle)  
150|     response.append(kp.response)  
151|     octave.append(kp.octave)  
152|     class_id.append(kp.class_id)  
153| descriptors['kp_point_x'] = point_x  
154| descriptors['kp_point_y'] = point_y  
155| descriptors['size'] = size  
156| descriptors['angle'] = angle  
157| descriptors['response'] = response  
158| descriptors['octave'] = octave  
159| descriptors['class_id'] = class_id  
160|  
161| #%%  
162| pickle.dump(descriptors, open('{} .pickle'.format(input()), 'wb'))
```

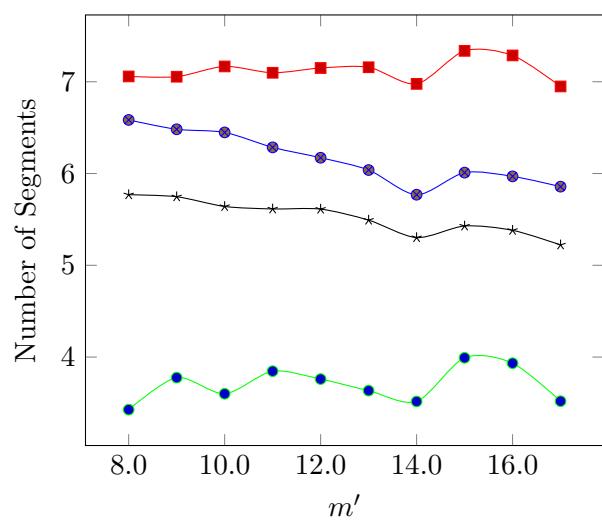
## LAMPIRAN B

### HASIL EKSPERIMENT

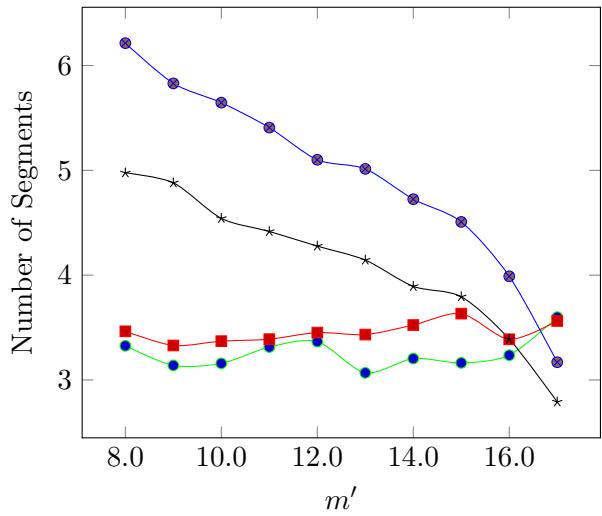
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



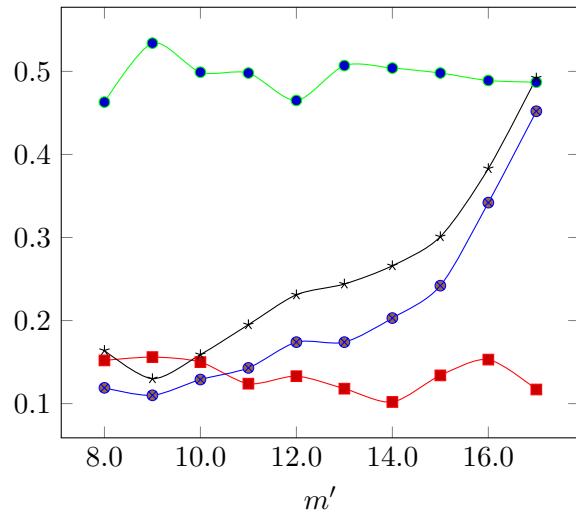
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4