

***Content Based Image Retrieval* pada Citra Motif Batik
dengan Ekstraksi Fitur *Region Adjacency Graph* dan
Pengukuran Kemiripan *Graph Matching***

SKRIPSI

Diajukan untuk menempuh ujian sarjana
pada Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Padjadjaran

MUHAMAD WIJAYA ADISAPUTRA

NPM 140810140034



UNIVERSITAS PADJADJARAN

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

PROGRAM STUDI S-1 TEKNIK INFORMATIKA

JATINANGOR

2018

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan ke hadirat Allah SWT yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan penyusunan skripsi yang berjudul “***Content Based Image Retrieval* pada Citra Motif Batik dengan Ekstraksi Fitur *Region Adjacency Graph* dan Pengukuran Kemiripan *Graph Matching***” sebagai salah satu syarat dalam menempuh ujian sarjana pada Program Studi S-1 Teknik Informatika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Padjadjaran.

Dalam proses penyusunan dan penulisan skripsi ini tidak terlepas dari bantuan, bimbingan, serta dukungan dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis mengucapkan terima kasih kepada Bapak Dr. Juli Rejito, M. Kom., selaku dosen pembimbing utama, dosen wali, sekaligus Kepala Program Studi S-1 Teknik Informatika, dan Bapak Rudi Rosadi, S.Si., M.Kom., selaku dosen pembimbing pendamping, yang dengan bijaksana memberikan bimbingan, masukan, kesabaran, serta waktu selama penelitian dan penulisan skripsi ini.

Tidak lupa penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Prof. Dr. Sudrajat, MS., selaku Dekan FMIPA Universitas Padjadjaran.
2. Dr. Setiawan Hadi, M. Sc. CS, selaku Kepala Departemen Ilmu Komputer Universitas Padjadjaran.
3., selaku dosen penguji sidang sarjana penulis.

4. Seluruh staff pengajar dan tata usaha Departemen Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alama Universitas Padjadjaran yang juga telah membantu penulis dalam menyelesaikan proses penulisan skripsi.
5. Bapak Akmal M.T., selaku dosen Teknik Informatika Universitas Padjadjaran yang membantu mengarahkan penulis selama penelitian ini.
6. Orang tua dan keluarga penulis yang menjadi sumber semangat dan motivasi terbesar yang selalu mendukung baik secara moral maupun materi.
7. Hidayaturrehman S.Kom., selaku rekan penulis yang telah membantu selama proses pembangunan program.
8. Teman-teman Assembly 2014 yang telah memberi motivasi, serta mengisi kekosongan hati penulis selama menyelesaikan skripsi ini.
9. Lukman Permadi S.Stat dan Naufal Jabroh Akmal, selaku sahabat penulis yang memberikan dorongan kepada penulis untuk segera tamat sarjana.
10. Teman-teman alumni SMAN 2 Kuningan yang selalu memberi motivasi di saat penulis mengalami masa-masa sulit selama penyusunan skripsi ini.
11. Seluruh pihak lain yang tidak dapat penulis sebutkan satu persatu, yang telah memberikan motivasi dan bantuan kepada penulis.

Akhir kata, penulis berharap semoga skripsi ini dapat bermanfaat bagi pengembangan ilmu di bidang informatika

Jatinangor, Mei 2018

Penulis

ABSTRAK

Indonesia merupakan negara yang memiliki kekayaan budaya melimpah. Salah satu ciri khas budaya Indonesia yaitu Batik. Batik memiliki banyak sekali varian motif yang berbeda, bahkan hampir setiap suku bangsa di Indonesia memiliki jenis-jenis motif dan warna batiknya sendiri. Dengan banyaknya jenis motif batik di Indonesia, tidak semua orang dapat mengenali sebuah motif batik bahkan warga negara Indonesia itupun sendiri. Melalui bantuan komputer, pencarian informasi mengenai sebuah motif batik ataupun untuk menemukan motif batik serupa dapat dilakukan lebih mudah. Bidang yang dapat berperan untuk memecahkan permasalahan ini yaitu *Content Based Image Retrieval*. Pada penelitian ini metode yang digunakan adalah segmentasi *felzenszwalb*, ekstraksi *region adjacency graph*, dan *graph matching* melalui algoritma *VF2 isomorphism* dan *graph edit distance*. Jumlah citra motif inti batik yang digunakan sebagai data adalah sebanyak 180 data berupa citra berwarna berukuran 64 x 64. Berdasarkan hasil penelitian, ditemukan bahwa melalui keseluruhan metode didapatkan nilai *precision* hingga 91.47 %, namun dengan nilai *f1 score* sebesar 10.51 %. Untuk nilai *f1 score* terbesar didapat hingga 53.92 %, namun nilai *precision* turun hingga 51.09 %.

Kata Kunci : Pengenalan Pola, Citra Digital, Visi Komputer, *Content Based Image Retrieval*, Segmentasi Citra, *Region Adjacency Graph*, *Graph Matching*.

ABSTRACT

Indonesia is a country with a plenty of cultural wealth. One of Indonesian culture characteristic is Batik. Batik has many variants of different patterns, even almost every tribe in Indonesia has their own kind of batik patterns and colors. Because of the number of batik pattern, not everyone can recognize a batik pattern even they are an Indonesian. With the help of computers, searching information or finding some batik patterns can be easier. Content Based Image Retrieval is one of the fields that can handle this problem. In this research, the used methods are Felzenszwalb segmentation, region adjacency graph extraction, and graph matching using VF2 isomorphism algorithm and graph edit distance. The number of colored batik pattern images as the research data is 180 images in size 64 x 64. Based on the result, it is found the precision value up to 91.47 %, but at the expense of f_1 score down to 10.51 %. The highest f_1 score result is obtained up to 53.92 %, but the precision is down to 51.09 %.

Keywords : Pattern Recognition, Digital Image, Computer Vision, Content Based Image Retrieval, Image Segmentation, Region Adjacency Graph, Graph Matching.

DAFTAR ISI

PENGESAHAN	ii
KATA PENGANTAR	iii
ABSTRAK	v
<i>ABSTRACT</i>	vi
DAFTAR ISI	vii
DAFTAR TABEL	x
DAFTAR GAMBAR	xi
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Identifikasi Masalah	4
1.3 Batasan Masalah	4
1.4 Maksud dan Tujuan Penelitian	5
1.5 Manfaat Penelitian	5
1.6 Metodologi Penelitian	6
1.7 Sistematika Penulisan	6
BAB II TINJAUAN PUSTAKA	8
2.1 Landasan Teori	8
2.1.1 Citra Digital	8
2.1.2 Content Based Image Retrieval (CBIR)	9
2.1.3 Teori Dasar <i>Graph</i>	10
2.1.3 Segmentasi Berbasis <i>Graph</i>	11
2.1.4 <i>Region Adjacency Graph</i> Mean Color	12

2.1.5	<i>Graph Matching</i>	13
2.1.6	<i>VF2 Algorithm</i>	16
2.1.7	<i>Graph Edit Distance</i>	17
2.1.8	<i>F₁ Score</i>	18
2.1.9	<i>Python Programming Language</i>	19
2.2	Penelitian Sebelumnya.....	21
BAB III METODOLOGI PENELITIAN.....		26
3.1	Desain Sistematis Penelitian	26
3.1.1	Studi Literatur	27
3.1.2	Persiapan Data.....	27
3.1.3	Segmentasi Citra	29
3.1.4	Ekstraksi Fitur melalui Pembangunan RAG	30
3.1.5	<i>Graph Matching</i>	31
3.2	Alat dan Bahan Penelitian	32
3.3	Perancangan Program Aplikasi Antarmuka	33
3.3.1	<i>Flowchart</i> Program	33
3.3.2	Perancangan Desain Antarmuka Program	37
BAB IV HASIL DAN ANALISIS		41
4.1	Uji Parameter dan Hasil Segmentasi Citra	41
4.2	Ekstraksi RAG	44
4.3	<i>Graph Matching</i>	46
4.3.1	Algoritma VF2	48
4.3.2	<i>Graph Edit Distance</i>	50

4.3.3	Hasil Evaluasi <i>Graph Matching</i>	51
4.4	Uji Coba Halaman Antarmuka	54
4.4.1	Halaman Masukan.....	54
4.4.2	Halaman Segmentasi	55
4.4.3	Halaman Hasil <i>Matching</i>	56
BAB V SIMPULAN DAN SARAN		58
5.1	Simpulan.....	58
5.2	Saran	59
DAFTAR PUSTAKA		60
LAMPIRAN		62

DAFTAR TABEL

Tabel 2.1 Tabel precision untuk metode berbasis teori graph	22
Tabel 2.2 Evaluasi Uji Coba Pengaruh Jumlah Data Training	23
Tabel 4.1 Hasil Pengujian Parameter Segmentasi.....	43
Tabel 4.2 Sampel Hasil Segmentasi Citra untuk Setiap Kelas.....	44
Tabel 4.3 Tabel Hasil Pembangunan RAG	46
Tabel 4.4 Sampel Hasil Graph Matching melalui Algoritma VF2	49
Tabel 4.5 Hasil Graph Matching untuk $GED \leq 8$	51
Tabel 4.6 Hasil Evaluasi Graph Matching	53

DAFTAR GAMBAR

Gambar 2.1 Representasi Matriks untuk Citra Digital.....	9
Gambar 2.2 Diagram Alur Sistem CBIR	9
Gambar 2.3 Contoh <i>Attributed Relational Graph</i>	11
Gambar 2.4 Penggambaran citra dengan pendekatan graph	12
Gambar 2.5 Sampel Penggambaran RAG.....	13
Gambar 2.6 Graph Isomorphism.....	14
Gambar 2.7 Subgraph Isomorphism	14
Gambar 2.8 Monomorphism	15
Gambar 2.9 Maximum Common Subgraph	15
Gambar 2.10 Contoh sederhana penerapan algoritma VF2	17
Gambar 2.11 Rata-rata precision pada percobaan setiap kategori citra	25
Gambar 3.1 Metode Penelitian.....	26
Gambar 3.2 Sampel Dataset Citra Sintetis	27
Gambar 3.3 Contoh hasil pengambilan motif inti batik	28
Gambar 3.4 Representasi Satu Piksel dalam Python	29
Gambar 3.5 Representasi hasil segmentasi.....	30
Gambar 3.6 Penggambaran proses ekstraksi RAG	30
Gambar 3.7 Flowchart Segmentasi Citra	34
Gambar 3.8 Flowchart Pencocokkan Citra	35
Gambar 3.9 Halaman Masukan Citra	37
Gambar 3.10 Halaman Segmentasi Citra	39
Gambar 3.11 Halaman Graph Matching	40

Gambar 4.1 Operasi Penyetelan Parameter Segmentasi	42
Gambar 4.2 Operasi Pembangunan RAG	45
Gambar 4.3 Potongan Program Penyimpanan *.json	48
Gambar 4.4 Operasi Graph Matching dengan Algoritma VF2	49
Gambar 4.5 Operasi Graph matching menggunakan Graph Edit Distance	50
Gambar 4.6 Potongan Program Evaluasi Graph Matching	52
Gambar 4.7 Potongan halaman input.html.....	54
Gambar 4.8 Halaman Masukan Citra.....	55
Gambar 4.9 Potongan Program Segmentation.html.....	55
Gambar 4.10 Beberapa Sampel Halaman Hasil Segmentasi	56
Gambar 4.11 Program Halaman Output.html	57
Gambar 4.12 Halaman Hasil Graph Matching untuk Citra Motif Inti Batik	57

BAB I

PENDAHULUAN

Bab ini menguraikan tentang latar belakang penelitian, identifikasi masalah, batasan masalah, maksud dan tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan.

1.1 Latar Belakang

Pengenalan pola atau *pattern recognition* merupakan kemampuan mengenali objek berdasarkan ciri-ciri dan pengetahuan yang diamati dari objek-objek tersebut. Implementasi pengenalan pola salah satunya yaitu terhadap citra digital, yang biasa disebut visi komputer (*computer vision*). Visi komputer telah diimplementasikan di berbagai area kehidupan, seperti pada pengenalan wajah, huruf, sidik jari, dan berbagai objek lainnya. Saat ini, bidang tersebut memiliki berbagai macam permasalahan yang masih sukar dipecahkan, sehingga masih gencar dikembangkan oleh para peneliti. Pada dasarnya, proses ini bekerja dengan mencari kemiripan fitur-fitur suatu citra digital dengan citra digital lainnya.

Untuk memperoleh fitur-fitur pada setiap citra digital perlu dilakukan proses ekstraksi fitur atau *feature extraction*. Fitur-fitur ini akan digunakan sebagai parameter khusus untuk membedakan suatu objek citra dengan citra lainnya. Sebelumnya, ekstraksi fitur yang biasa digunakan pada pengenalan citra yaitu

dengan menggunakan ekstraksi ciri berupa bentuk, warna, ukuran, geometri, tekstur, dan lain-lain. Teknik analisis citra berbasis piksel yang tradisional tidak menghasilkan ekstraksi yang efisien karena hanya merepresentasikan konten-konten dari setiap piksel. Pendekatan yang sangat menjanjikan adalah dengan mengekstraksi *graph* dari citra (Akmal, Suwardi, & Munir, 2017). Hal ini dikarenakan *graph* diyakini memiliki struktur penjas yang lebih efektif berkat kemampuannya untuk merepresentasikan informasi relasional. Selain itu, *graph* juga dapat digunakan untuk menyediakan deskripsi citra yang lebih efisien melalui kumpulan simpul dengan atribut ditentukan menurut komponen citra, serta *edges* dengan pendekatan nilai bobot yang sesuai dengan kebutuhan citra (Sharma, et al., 2012). Pada beberapa penelitian telah dilakukan uji coba menggunakan ekstraksi fitur berbasis *graph* salah satunya dengan menghasilkan *Region Adjacency Graph* (RAG) dari citra tersegmentasi, di mana penelitian-penelitian tersebut menghasilkan fitur yang lebih fleksibel dan kuat dari ekstraksi fitur piksel tradisional.

Graph matching merupakan sebuah teknik yang *powerful* dan *robust* untuk digunakan di bidang-bidang sains. Teknik ini juga sudah banyak digunakan di permasalahan visi komputer, seperti *feature correspondence*, *object recognition*, dan *video analysis* (Cho & Lee, 2012). Tidak seperti teknik lainnya, *graph matching* memberikan fleksibilitas yang lebih terhadap model objek serta menjanjikan hasil dari proses pengenalan dan pencocokan yang baik.

Content-Based Image Retrieval (CBIR) merupakan salah satu penerapan dari visi komputer. Digunakan pada permasalahan pencarian citra digital dalam data

berskala besar. Perbedaannya dengan metode tradisional yaitu CBIR menggunakan fitur-fitur piksel yang terdapat di dalam sebuah citra, sedangkan metode tradisional menggunakan indeksasi citra berdasarkan deskripsi atau teks untuk setiap citra di dalam *database*.

Indonesia merupakan negara yang memiliki banyak kebudayaan. Hampir dari setiap kebudayaan memiliki corak dan keunikan tersendiri, termasuk di bidang seni. Indonesia memiliki sebuah ciri khas seni motif yang biasa diterapkan pada pakaian yaitu batik. Hampir setiap suku di Indonesia memiliki ciri khas batiknya masing-masing. Bahkan batik telah ditetapkan oleh UNESCO sebagai Warisan Kemanusiaan untuk Budaya Lisan dan Nonbendawi (*Masterpieces of the Oral and Intangible Heritage of Humanity*).

Sebelumnya terdapat penelitian mengenai pengenalan motif batik dengan deteksi tepi *Canny* dan *K-Nearest Neighbor*. Pada penelitian tersebut didapatkan akurasi hingga 66,67 % dengan menggunakan nilai *lower threshold* = 0,010, *upper threshold* = 0,115, dan $k=1$ (Yodha & Kurniawan, 2014). Pada penelitian ini, penulis mencoba melakukan proses *retrieval* berdasarkan konten citra pada motif inti batik dengan menggunakan pengolahan berbasis *graph*. Adapun metode ataupun algoritma yang akan digunakan dalam proses ini adalah segmentasi *Felzenszwalb*, ekstraksi RAG, dan *graph matching* menggunakan *VF2 Isomorphism* dan *Graph Edit Distance*.

1.2 Identifikasi Masalah

Mengacu pada latar belakang tersebut, didapatkan identifikasi masalah yang menjadi pokok penelitian ini, yaitu:

1. Bagaimana spesifikasi dataset yang dapat digunakan dalam penelitian ini.
2. Bagaimana *pre-processing* citra yang optimal sehingga didapatkan fitur-fitur RAG untuk proses pencocokan objek citra.
3. Bagaimana cara mengimplementasikan *graph matching* pada RAG agar memberikan hasil yang relevan pada pengambilan citra digital yang digunakan dalam penelitian ini.

1.3 Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, akan dilakukan pembatasan masalah penelitian sebagai berikut:

1. Dataset yang digunakan adalah kumpulan citra digital berukuran 64 x 64 yang disediakan oleh peneliti baik berupa citra *artificial* maupun citra motif inti batik.
2. Citra *artificial* adalah citra yang dibuat penulis sedemikian rupa sehingga relevan untuk menguji metode pada penelitian ini.
3. Citra motif inti batik adalah citra motif batik yang diambil dari sumber-sumber lain (tidak dibuat oleh penulis).
4. Citra digital yang akan diujikan hanya berasal dari *dataset* peneliti.
5. Program dibangun menggunakan bahasa pemrograman Python.
6. Hasil dari penelitian ini akan menunjukkan kelompok citra yang diprediksi memiliki kemiripan dengan citra masukan.

1.4 Maksud dan Tujuan Penelitian

Maksud dari penelitian ini adalah membangun sebuah metode yang mampu melakukan pengambilan citra baik *artificial* maupun motif inti batik dari bentuk *region adjacency graph* menggunakan teknik *graph matching*.

Tujuan yang ingin dicapai oleh penulis dari penelitian ini adalah:

1. Menentukan karakteristik citra digital yang relevan digunakan untuk pengambilan citra menggunakan teknik *graph matching* pada hasil ekstraksi RAG.
2. Menghasilkan fitur-fitur RAG yang optimal untuk menunjang pada tahap *graph matching*.
3. Menghasilkan pengambilan kelompok citra yang diprediksi memiliki kemiripan berdasarkan parameter dan algoritma tertentu pada setiap citra digital yang diujikan pada penelitian ini.

1.5 Manfaat Penelitian

Manfaat yang diharapkan penulis dari penelitian ini adalah:

1. Diharapkan hasil penelitian ini dapat digunakan untuk pengembangan sistem CBIR melalui teknik deskripsi *graph*.
2. Dapat digunakan sebagai referensi untuk penelitian selanjutnya baik melalui perubahan *dataset* maupun pengembangan metode.

1.6 Metodologi Penelitian

Penelitian yang akan dilakukan menggunakan metode sebagai berikut:

1. Studi literatur mengenai penelitian terkait dan teori pengenalan pola terhadap citra digital khususnya teknik *graph matching* dari berbagai sumber seperti buku elektronik, prosiding konferensi, artikel jurnal, dan skripsi sebelumnya.
2. Pembuatan (untuk citra *artificial*) dan pengumpulan dataset yang akan digunakan dalam penelitian.
3. Mengimplementasikan algoritma untuk proses-proses ekstraksi ciri dan pengenalan pola untuk menentukan kemiripan citra digital.
4. Melakukan evaluasi hasil dari uji coba pengenalan citra menggunakan teknik berbasis *graph*.

1.7 Sistematika Penulisan

Sistematika penulisan dalam skripsi ini adalah sebagai berikut:

BAB I PENDAHULUAN

Bab ini menguraikan latar belakang penelitian, identifikasi masalah, batasan masalah, tujuan penelitian, kegunaan penelitian, metodologi penelitian, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini berisi uraian penelitian terkait yang pernah dilakukan sebelumnya mengenai CBIR pada citra menggunakan teknik berbasis deskripsi *graph*. Pada bagian ini juga dipaparkan mengenai beberapa teori serta tulisan yang berkaitan dengan penelitian yang dilakukan.

BAB III METODOLOGI PENELITIAN

Pada bab ini diuraikan mengenai proses penelitian yang dilakukan. Bab ini juga menjelaskan alat serta bahan yang digunakan selama proses penelitian. Bab ini menguraikan setiap langkah yang dilakukan selama penelitian.

BAB IV HASIL DAN PEMBAHASAN

Bab ini membahas mengenai hasil dari proses CBIR yang dilakukan menggunakan ekstraksi fitur RAG dan pengukuran kemiripan *graph matching*. Pada bab ini akan dikaji tingkat akurasi dari fitur dan algoritma yang digunakan.

BAB V SIMPULAN DAN SARAN

Bab ini merupakan penutup dari skripsi yang berisi simpulan dan saran yang diambil dari pembahasan pada skripsi ini.

BAB II

TINJAUAN PUSTAKA

Bab ini berisi uraian mengenai penelitian terkait yang pernah dilakukan sebelumnya mengenai *Content Based Image Retrieval* menggunakan ekstraksi fitur *RAG* dan pengukuran kemiripan *graph matching*. Pada bagian ini juga dipaparkan mengenai teori serta tulisan yang berkaitan dengan penelitian yang dilakukan.

2.1 Landasan Teori

Berikut adalah beberapa landasan teori yang digunakan sebagai dasar-dasar pemikiran untuk metode yang digunakan untuk proses pengambilan citra, mulai dari *pre-processing* citra, ekstraksi fitur *RAG*, hingga *Graph Matching*.

2.1.1 Citra Digital

Citra digital diartikan sebagai suatu fungsi intensitas cahaya dua dimensi, yang dinyatakan oleh $f(x,y)$, di mana nilai pada koordinat spasial (x,y) menyatakan intensitas citra pada titik tersebut. Citra sebagai salah satu komponen multimedia memegang peranan sangat penting sebagai bentuk informasi visual.

Meskipun sebuah citra kaya informasi, namun seringkali citra yang kita miliki mengalami penurunan mutu sehingga menjadi lebih sulit diinterpretasi karena informasi yang disampaikan oleh citra tersebut menjadi berkurang (Munir, 2004). Dalam hal ini bidang pengolahan citra memiliki kewajiban untuk

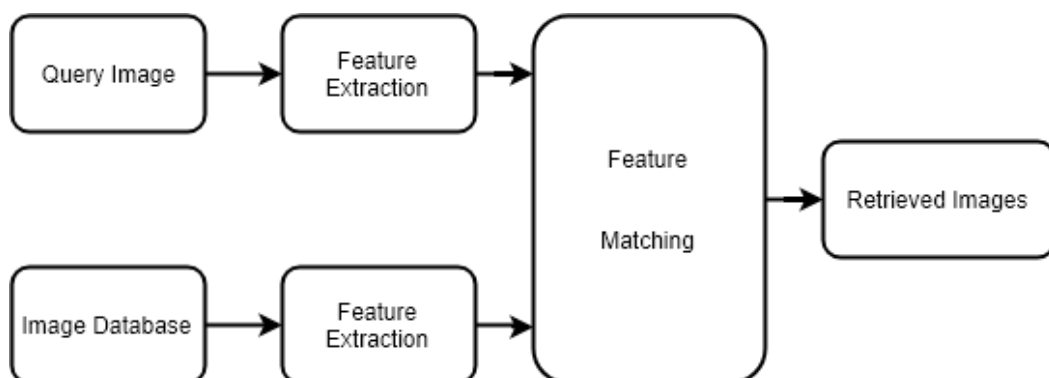
memperbaiki kekurangan suatu citra tersebut. Gambar 2.1 menunjukkan representasi matriks untuk citra digital.

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix}$$

Gambar 2.1 Representasi Matriks untuk Citra Digital

2.1.2 Content Based Image Retrieval (CBIR)

CBIR merupakan implementasi teknik visi komputer pada permasalahan pengambilan citra. Yaitu permasalahan pada pencarian citra digital di dalam data yang besar. CBIR merupakan pembaruan dari metode tradisional yang menggunakan indeksasi dalam bentuk teks. Gambar 2.2 menunjukkan alur pada sistem CBIR.



Gambar 2.2 Diagram Alur Sistem CBIR

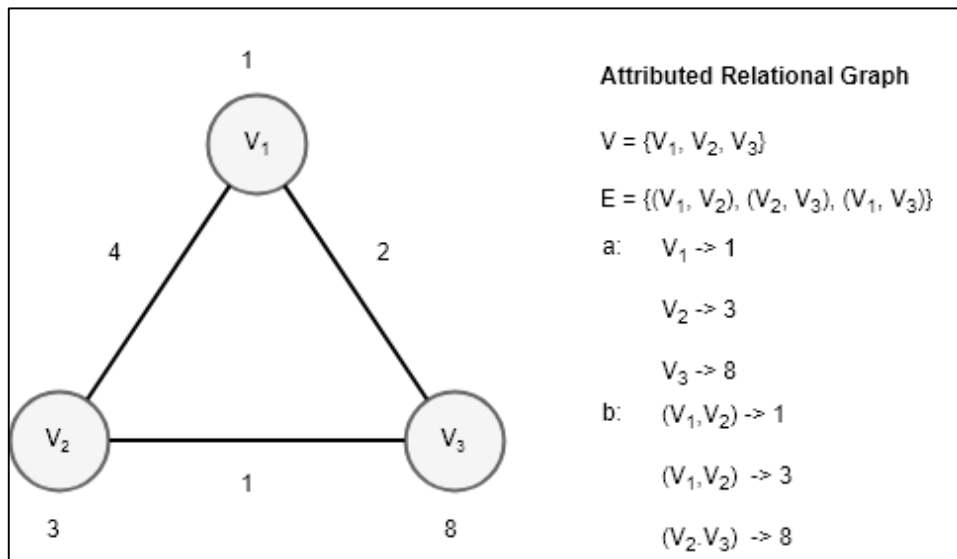
Dalam CBIR, citra diambil berdasarkan perbandingan warna, bentuk, tekstur, atau informasi lain yang berkaitan dengan citra itu sendiri. CBIR dibuat karena diyakini bahwa melakukan masukan kata berupa kata kunci atau *metadata* secara manual oleh manusia memakan waktu dan belum tentu menuju kata kunci yang tepat untuk mendeskripsikan citra tersebut.

2.1.3 Teori Dasar *Graph*

Graph adalah sebuah himpunan yang memiliki sejumlah titik-titik, yang disebut simpul (*vertex* atau *node*), yang saling dihubungkan melalui garis-garis yang disebut sisi (*edge*) (Sharma, et al., 2012). Suatu sisi dapat menghubungkan suatu simpul dengan simpul yang sama, sisi tersebut dinamakan *loop*.

Secara umum *graph* G didefinisikan sebagai pasangan himpunan (V, E) , ditulis dengan $G = (V, \varepsilon)$ yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul (*nodes*) dan ε adalah himpunan sisi (*edges*) yang menghubungkan sepasang simpul (Angelia, 2011).

Sebuah *graph* G dikatakan *Attributed Relational Graph* (ARG) ketika simpul-simpul dan sisi-sisinya direpresentasikan dalam atribut-atribut tertentu. Atribut simpul untuk simpul n_i dinotasikan sebagai sebuah vektor $a_i = [a_i^{(k)}]$, ($k = 1, 2, 3, \dots, K$), di mana K adalah jumlah dari atribut-atribut simpul dalam vektor a_i , dan atribut-atribut sisi (*weights*) untuk sisi e_i dinotasikan sebagai $b_j = [b_j^{(m)}]$, ($m = 1, 2, 3, \dots, M$), di mana M adalah jumlah atribut-atribut sisi dalam vektor b_j (Sharma, et al., 2012). Gambar 2.3 menunjukkan contoh ARG.



Gambar 2.3 Contoh *Attributed Relational Graph*

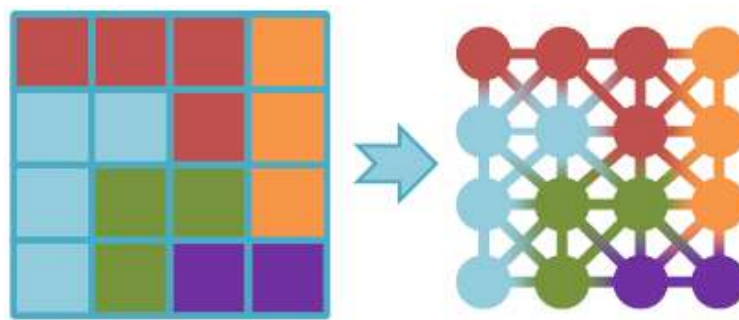
2.1.3 Segmentasi Berbasis *Graph*

Sebelum dibangun sebuah RAG diperlukan pengelompokkan piksel-piksel ke dalam kelompok-kelompok (*regions / clusters*) tertentu, proses ini dinamakan segmentasi. Salah satu metode segmentasi citra berbasis *graph* yaitu metode yang diusulkan oleh Felzenszwalb.

Metode segmentasi ini diusulkan pada *International Journal of Computer Vision* (2004) ke-59 dengan judul “*Efficient Graph-Based Image Segmentation*” yang ditulis oleh Pedro F. Felzenszwalb. Metode ini menggunakan *graph* dalam bentuk *Minimum Spanning Tree* (MST) dan pendekatan algoritma *Kruskal* sebagai representasi untuk proses segmentasi. Algoritma ini telah diujikan untuk citra *real* maupun *artificial* (buatan). *Running time* untuk algoritma ini berjalan secara linier dengan jumlah *edges* (Felzenszwalb, 2004). Metode ini dapat mengukur batas-batas regional dengan membandingkan dua buah kuantitas: berdasarkan perbedaan

intensitas batas regional yang bersebrangan, dan berdasarkan intensitas antar piksel-piksel yang bertetangga di setiap regional.

Melalui pendekatan *graph*. Dapat didefinisikan $G = (V, E)$ sebagai *graph* tidak berarah dengan simpul $v_i \in V$, dan sisi $(v_i, v_j) \in E$ menunjukkan hubungan sepasang piksel yang bertetangga. Dengan kata lain V merupakan piksel-piksel dalam sebuah citra dan ω bobot dari sebuah sisi yang diukur berdasar intensitas perbedaan sepasang piksel (v_i, v_j) . Contoh penggambaran *graph* untuk citra digital ditunjukkan oleh Gambar 2.4.



Gambar 2.4 Penggambaran citra dengan pendekatan graph

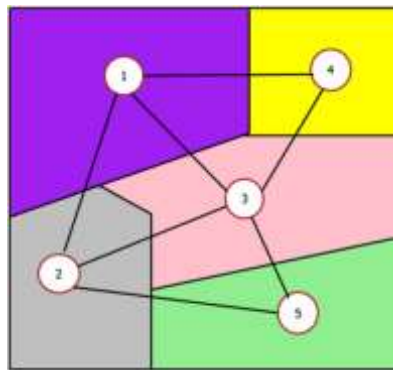
Algoritma pada metode ini berisi masukan berupa *graph* $G = (V, E)$, dengan n simpul dan m sisi. Keluaran yang akan dihasilkan adalah segmentasi V ke dalam komponen $S = (C_1, \dots, C_r)$ yang merupakan gabungan komponen-komponen regional yang telah dikelompokkan.

2.1.4 Region Adjacency Graph Mean Color

RAG merupakan sebuah ARG yang memiliki simpul-simpul yang dapat merepresentasikan kumpulan daerah (*regions*) dan sisi-sisi yang dapat merepresentasikan hubungan antar simpul yang berdekatan. RAG memberikan

keefektifan dalam aplikasi untuk representasi informasi dari suatu citra. RAG telah banyak digunakan dalam bidang segmentasi citra berwarna (Tremeau A, 2000).

RAG menghubungkan bagian-bagian citra yang telah terpartisi melalui proses sebelumnya. Manfaat utama dari RAG yaitu dapat menunjukkan “*spatial view*” dari citra. Salah satu cara untuk merepresentasikan RAG yang berisi kumpulan simpul dari setiap partisi. Penggambaran contoh RAG ditunjukkan Gambar 2.5.



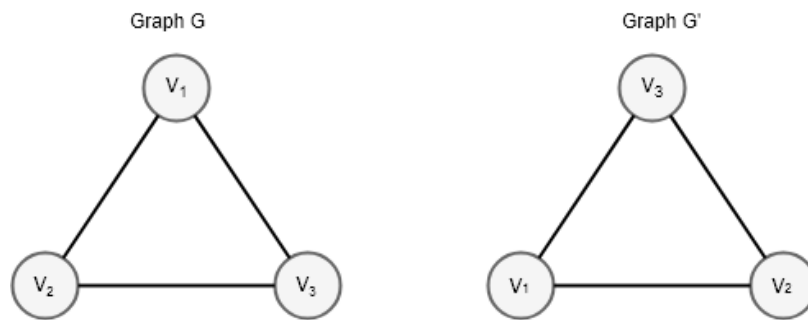
Gambar 2.5 Sampel Penggambaran RAG

2.1.5 *Graph Matching*

Graph Matching adalah proses perbandingan dua buah *graph* untuk mengukur sebuah hubungan kemiripan maupun ketidakmiripan antar simpul dan sisi dari kedua *graph* tersebut. Hal ini mengacu pada proses pemetaan F dari simpul-simpul suatu *graph* G ke simpul-simpul dari *graph* lain G' yang memenuhi batas-batas atau kriteria optimal. Berikut merupakan konsep dasar dari proses metode *graph matching*:

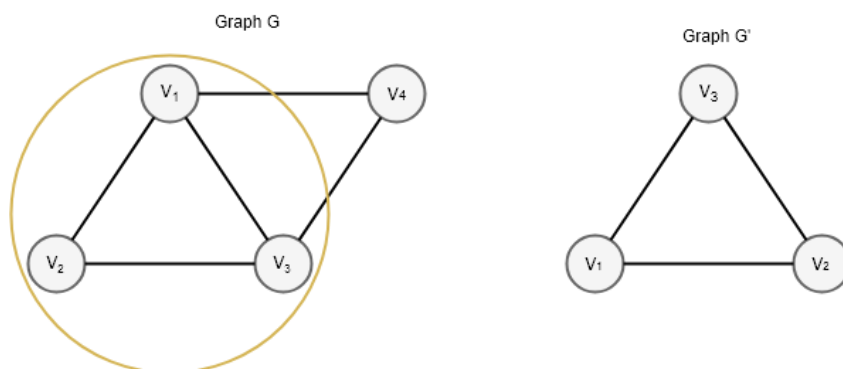
1. *Graph Isomorphism*. Proses ini menemukan struktur yang tepat antar dua buah objek *graphs*. Di dalamnya terjadi korespondensi satu-satu antar

simpul dan sisi dari kedua *graph*. Gambar 2.6 menunjukkan contoh *Graph Isomorphism*.



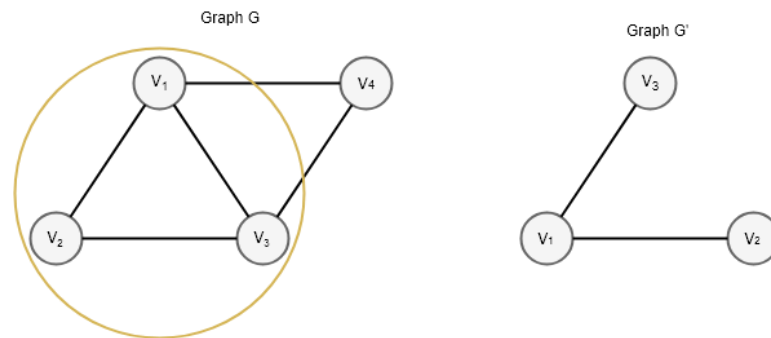
Gambar 2.6 *Graph Isomorphism*

2. *Subgraph Isomorphism*. Proses ini menemukan sebuah *subgraph* G' dalam sebuah *graph* G . Jika simpul-simpul yang berhubungan di dalam *graph* G dihapus, akan ditemukan sebuah *subgraph* G' . Gambar 2.7 menunjukkan contoh *Subgraph Isomorphism*.

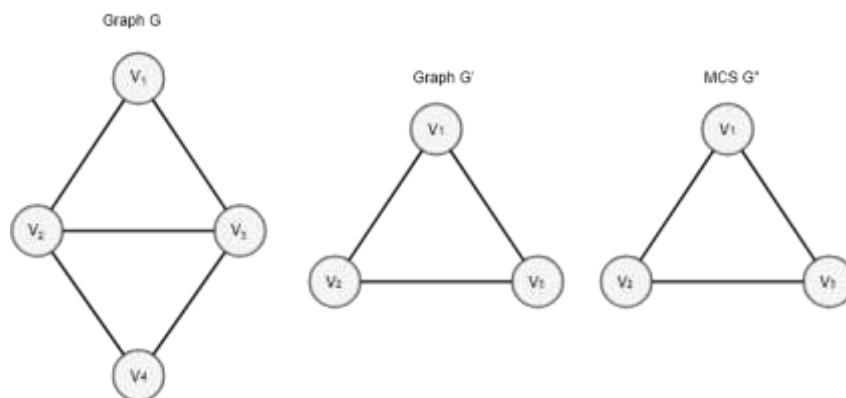


Gambar 2.7 *Subgraph Isomorphism*

3. *Monomorphism*. Proses ini lebih fleksibel dari *subgraph isomorphism* karena disini diperbolehkan ada sisi lain pada simpul di dalam *graph* yang lebih besar. Gambar 2.8 menunjukkan contoh *monomorphism*.

Gambar 2.8 *Monomorphism*

4. *Maximum Common Subgraph* (MCS). Sebuah MCS dari dua *graphs*, G dan G' , adalah sebuah *graph* G'' yang merupakan bagian dari G dan G' . Sehingga MCS memiliki jumlah simpul maksimal yang mungkin pada kedua *graphs*. Gambar 2.9 menunjukkan contoh MCS.

Gambar 2.9 *Maximum Common Subgraph*

Graph Matching telah menjadi topik penelitian bidang *Computer Science* selama beberapa dekade (Riesen, 2015). Dua kategori utama dalam *graph matching*

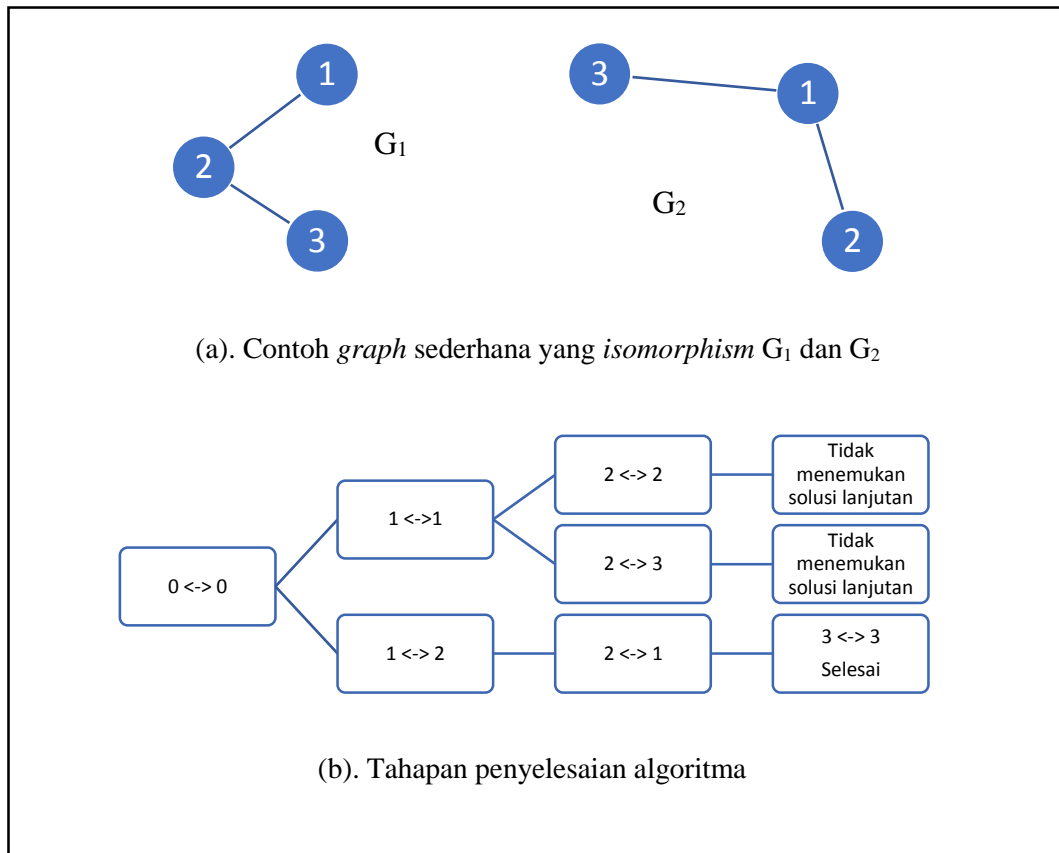
yaitu *exact graph matching* dan *inexact graph matching* (atau *error-tolerant graph matching*).

1. *Exact Graph Matching*. Kategori ini bertujuan untuk menemukan hubungan yang identik antara dua objek *graph* tanpa mengubah simpul atau sisi dari sebuah *graph*. Yang artinya terdapat hubungan *bijective* antar simpul dari g_1 dan g_2 .
2. *Inexact Graph Matching* atau *Error-Tolerant Graph Matching*. Di samping itu, kategori ini bekerja untuk mengukur seberapa mirip atau seberapa berbeda sebuah *graph* dengan *graph* lainnya. Hal ini dilakukan ketika *exact graph matching* tidak dapat dilakukan, karena adanya perbedaan jumlah simpul dan sisi pada bagian *graph* yang lain.

2.1.6 VF2 Algorithm

Algoritma ini diperkenalkan oleh Cordella pada tahun 2001. Algoritma ini dapat digunakan untuk masalah *graph isomorphism* dan *graph-subgraph isomorphism* yang mana merupakan bagian dari *exact graph matching* (Cordella, Foggia, & Vento, 2001). VF2 merupakan perkembangan dari algoritma Ullman yang sebelumnya hanya bisa digunakan untuk masalah *graph isomorphism*.

Secara sederhana, algoritma ini bekerja dengan melakukan pemetaan dari simpul pada G_1 ke setiap simpul-simpul di G_2 . Gambar 2.10 menunjukkan contoh sederhana penerapan algoritma ini.



Gambar 2.10 Contoh sederhana penerapan algoritma VF2

2.1.7 Graph Edit Distance

Graph edit distance (GED) adalah sebuah teknik untuk mengukur kemiripan antar dua buah *graph*. Konsep awal teknik ini pertama dikenalkan oleh Alberto Sanfeliu dan King-Sun Fu pada tahun 1983 (Sanfeliu & Fu, 1983). Penerapan utamanya yaitu pada *inexact graph matching* atau *error-tolerant pattern recognition*.

GED antar dua buah *graph* g_1 dan g_2 secara matematika dituliskan sebagai $GED(g_1, g_2)$, dan dapat didefinisikan sebagai berikut:

$$GED(g_1, g_2) = \min_{(e_1, \dots, e_k) \in P(g_1, g_2)} \sum_{i=1}^k c(e_i) \quad (1)$$

Dimana $P(g_1, g_2)$ dinotasikan sebagai sebuah set *edit paths* yang mentransformasikan g_1 ke bentuk isomorfis dari g_2 dan $c(e) \geq 0$ merupakan nilai biaya dari setiap operasi perubahan *graph*.

Beberapa operasi dasar pada *graph edit distance* meliputi:

- *Vertex Insertion*, menambahkan sebuah simpul beratribut pada sebuah *graph*.
- *Vertex Deletion*, menghapus sebuah simpul dari sebuah *graph*.
- *Vertex Subtitution*, mengganti nilai atribut dari sebuah simpul.
- *Edge Insertion*, memberikan sisi baru yang telah diberi nilai bobot pada sepasang simpul.
- *Edge Deletion*, menghapus sisi antar sepasang simpul.
- *Edge Substitution*, mengganti nilai bobot pada sebuah sisi.

2.1.8 F_1 Score

Dalam analisis statistika pada *binary classification*, f_1 score merupakan salah satu tekntik untuk mengukur besarnya akurasi pada suatu uji coba. Untuk menghitungnya diperlukan nilai *precision* dan *recall*.

$$F_1score = \frac{2}{\frac{1}{recall} + \frac{1}{Precision}} \quad (2)$$

Pada konteks *information retrieval*, *precision* dan *recall* didefinisikan dalam kondisi *retrieved documents* (dokumen yang dihasilkan dari sebuah proses

pengambilan) dan *relevant documents* (dokumen yang didefinisikan sesuai dengan topik bahasan). Pengukuran untuk *precision* dan *recall* ini didefinisikan dalam (Perry, Kent, & Berry, 1955). Fungsi berikut merupakan definisi secara matematis pengukuran *precision* dan *recall* pada konteks ini.

$$Precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|} \quad (3)$$

$$Recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|} \quad (4)$$

2.1.9 Python Programming Language

Python merupakan bahasa pemrograman tingkat tinggi dengan *interpreter* yang dapat digunakan untuk pemrograman secara umum. Pertama kali ditemukan pada 1991 oleh Guido van Rossum.

Python memiliki fitur tipe sistem yang dinamis dan pengolahan memori otomatis. Bahasa ini juga mendukung beberapa paradigma pemrograman seperti *object-oriented*, *imperative*, *functional* dan *procedural*. *Python* juga memiliki *library* standar yang banyak . Berikut beberapa contoh *library* tambahan pada bahasa pemrograman *python*.

1. NumPy

NumPy merupakan sebuah paket dasar untuk komputasi sains dengan bahasa pemrograman *Python*. Dengan menggunakan *library* ini perhitungan matematis dapat diselesaikan lebih mudah, karena memiliki fitur objek larik n-dimensi, fungsi yang banyak, dan lain-lain.

2. *Matplotlib*

Matplotlib adalah sebuah *library* penyusunan 2D pada bahasa pemrograman *python* yang menghasilkan gambar berkualitas dalam berbagai format dan lingkungan interaktif di seluruh *platform*. *Matplotlib* dapat digunakan dalam skrip *python*, *python* dan *IPython shell*, *jupyter notebook*, *web application server*, dan *graphical user interface* (Hunter, 2007).

3. *Virtualenv*

Virtualenv adalah sebuah *tools* untuk membuat *python environment* terisolasi. Melalui *tools* ini kita dapat memasang aplikasi tambahan *python* dengan berbagai versi berbeda di luar folder *global site-packages* pada *python* yang terpasang di sistem operasi. Dan kita dapat menghapusnya lagi ketika sudah tidak diperlukan tanpa mengganggu direktori *python*.

4. *Jupyter Notebook*

Jupyter Notebook merupakan sebuah perangkat lunak *open-source* untuk komputasi interaktif pada berbagai bahasa pemrograman. Perangkat ini memudahkan pengguna secara interaktif, dan juga dapat menyimpan hasil eksekusi program.

5. *Scikit-Image*

Scikit-Image adalah sebuah *library* pengolahan citra yang mengimplementasikan algoritma dan kegunaannya dalam riset, edukasi, dan industri (Walt, et al., 2014). *Library* ini dirilis di bawah lisensi BSD (*Berkeley Software Distribution*) yang menyediakan dokumentasi API yang baik dalam

bahasa pemrograman *python*, dan dikembangkan oleh kolaborator internasional.

6. *NetworkX*

NetworkX merupakan sebuah paket *python* untuk pembuatan, manipulasi, dan pembelajaran pada struktur, dinamika, dan fungsi dari jaringan yang kompleks (Hagberg, Schult, & Swart, 2008). Paket ini memiliki data struktur untuk *graphs*, *digraphs*, dan *multigraphs*. Serta memiliki banyak fungsi untuk membantu algoritma pengolahan dan analisisnya. Paket ini pun berada di bawah lisensi BSD.

7. *Flask*

Flask merupakan sebuah *microframework* untuk bahasa pemrograman *python* yang berbasis pada *Werkzeug* WSGI dan *Jinja 2*. *Framework* ini juga berada di bawah lisensi BSD. Melalui *framework* ini *python* dapat diintegrasikan lebih mudah dengan halaman *web*.

2.2 Penelitian Sebelumnya

Berikut adalah beberapa penelitian sebelumnya yang berkaitan dengan penelitian ini.

1. *Determining Similiarity in Histological Images using Graph-Theoretic Description and Matching Methods for Content-Based Image Retrieval in Medical Diagnostics* (Sharma, et al., 2012)

Penelitian ini mendeskripsikan sebuah metode untuk menentukan kemiripan antar citra-citra jaringan makhluk hidup berbasis teori deskripsi *graph*, dengan tujuan *Content Based Retrieval*. Pencapaian tertinggi dari metode ini telah

didapatkan pada representasi *graph* pada gambar biopsi payudara dan pencarian *tree* berbasis teknik *inexact graph matching* telah digunakan dalam memfasilitasi proses *automatic retrieval* dari citra yang secara struktural mirip dengan gambar yang diberikan dari *database* besar.

Hasil dari evaluasi yang dilakukan menunjukkan efektifitas dan superioritas *Graph-Based Image Retrieval* melebihi teknik berbasis histogram. Kompleksitas *graph matching* yang digunakan telah dikurangi berdasarkan *state of the art* pada metode *inexact matching* yang optimal dengan menerapkan pra-syarat untuk pencocokan simpul dan desain yang canggih dari fungsi estimasi, terutama fungsi prognosis. Tabel 2.1 menunjukkan hasil evaluasi *precision* untuk metode berbasis teori *graph* dalam penelitian ini.

Tabel 2.1 Tabel *precision* untuk metode berbasis teori *graph*

P_s / Window Size	64 x 64	128 x 128	256 x 256	512 x 512	Average P_s
P_{10}	80	55	63	70	67
P_{20}	63	44	53	40	50
P_{30}	58	39	50	36	46
P_{40}	53	33	38	33	39
P_{50}	46	29	35	29	35

Pada tabel tersebut dapat dilihat bahwa nilai *precision* tertinggi untuk metode ini berada pada citra 64 x 64 yaitu senilai **80 %**. Adapun pada setiap *window size* nilai tertinggi selalu terjadi pada citra 64 x 64. Dan rata-rata setiap ukuran piksel terbesar terdapat pada *windows size* ukuran 10, yaitu sebesar 67 %.

2. Pengenalan Motif Batik Menggunakan Deteksi Tepi *Canny* dan *K-Nearest Neighbor* (Yodha & Kurniawan, 2014)

Salah satu budaya ciri khas Indonesia yang telah dikenal duni adalah batik. Penelitian ini bertujuan untuk mengenali 6 jenis motif batik pada buku karangan H.Santosa Doellah yang berjudul “Batik: Pengaruh Zaman dan Lingkungan”. Proses klasifikasi akan melalui 3 tahap yaitu preprosesing, feature extraction dan klasifikasi.

Preproses mengubah citra warna batik menjadi citra *grayscale*. Pada tahap *feature extraction* citra *grayscale* ditingkatkan kontrasnya dengan *histogram equalization* dan kemudian menggunakan deteksi tepi *Canny* untuk memisahkan moti batik dengan citra latar belakangnya dan untuk mendapatkan pola dari motif batik tersebut. Hasil ekstraksi kemudian dikelompokkan dan diberi label sesuai motfinya masing-masin dan kemudian diklasifikasikan menggunakan *k-Nearest Neighbor* menggunakan pencarian jarak *Manhattan*. Tabel 2.2 menunjukkan hasil evaluasi uji coba dari pengaruh data *training* terhadap akurasi.

Tabel 2.2 Evaluasi Uji Coba Pengaruh Jumlah Data Training

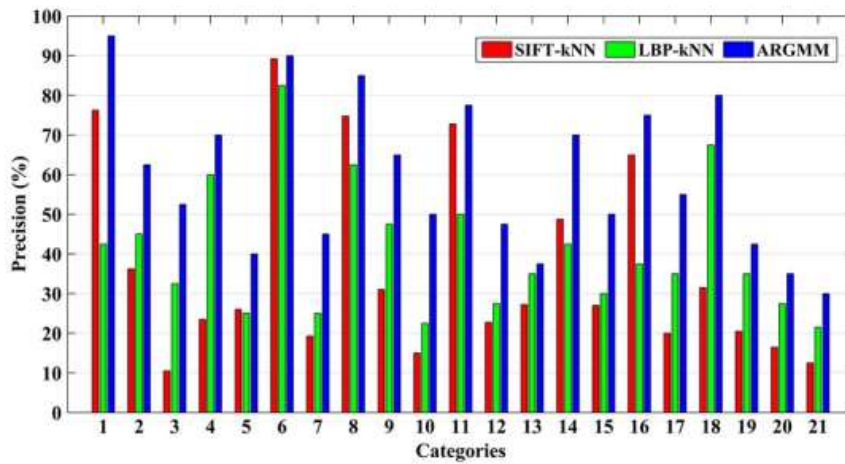
Jumlah Data Training	Jumlah Data Testing	Akurasi	Waktu (Detik)
210	30	56,57 %	11,63
240	30	66,67 %	16,56
270	30	66,6 %	19

Hasil uji coba diperoleh akurasi tertinggi mencapai 100% pada penggunaan data testing sama dengan data training (dataset sebanyak 300 citra). Pada penggunaan data training yang berbeda dengan data testing diperoleh akurasi tertinggi 66,67 %. Kedua akurasi tersebut diperoleh dengan menggunakan *lower threshold* = 0,010 dan *upper threshold* = 0,115 dan menggunakan $k=1$. Tabel 2.2 menunjukkan evaluasi uji coba pengaruh jumlah data *training* terhadap akurasi pengenalan citra.

3. *Region-Based Retrieval of Remote Sensing Images Using an Unsupervised Graph-Theoretic Approach* (Chaudhuri, Demir, & Bruzzone, 2016)

Jurnal ini memperkenalkan sebuah pendekatan *unsupervised* baru dalam *region-based retrieval* pada citra penginderaan jauh (*remote sensing images*). Pendekatan yang diusulkan yaitu melalui 2 tahap: 1) memodelkan setiap citra dengan sebuah *graph*, yang menyediakan representasi citra berbasis *region* yang mengombinasikan informasi lokal dan spasial, dan 2) mengambil citra dalam arsip yang memiliki kemiripan terbesar dengan citra masukan dengan mengevaluasi kemiripan berdasar *graph*.

Penelitian yang dilakukan pada arsip citra udara menunjukkan bahwa pendekatan yang diusulkan secara signifikan meningkatkan kinerja proses *retrieval* dibanding *state-of-the-art* metode *unsupervised RS image retrieval*. Gambar 2.11 menunjukkan hasil *precision* dari rata-rata percobaan (dalam 20 percobaan per kategori) untuk seluruh kategori yang diujikan dalam penelitian ini menggunakan SIFT-kNN, LBP-kNN, dan ARGMM yang diusulkan.



Gambar 2.11 Rata-rata *precision* pada percobaan setiap kategori citra

Dari gambar di atas dapat dilihat, bahwa nilai rata-rata *precision* hasil klasifikasi menggunakan metode yang diusulkan selalu lebih baik dibanding metode lainnya. Dimana pada citra kategori ke-1 metode ini menghasilkan nilai tertinggi yaitu *precision* > 90 %.

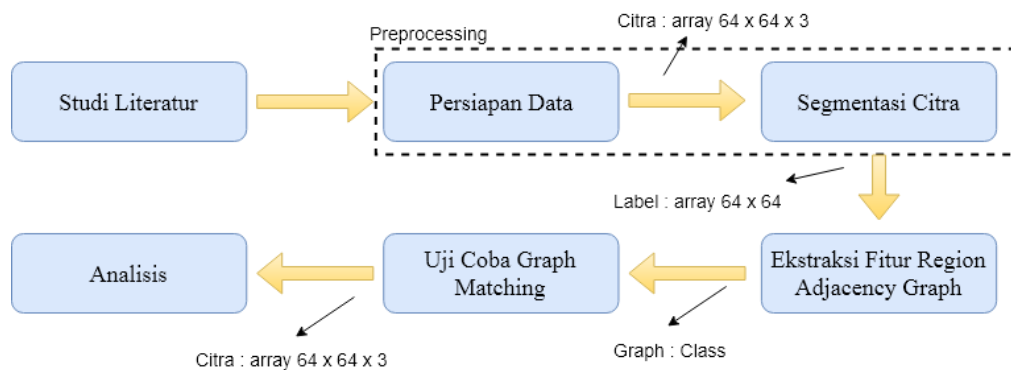
BAB III

METODOLOGI PENELITIAN

Bab ini berisi uraian mengenai desain sistematika proses penelitian mulai dari *pre-processing* citra, ekstraksi fitur *RAG*, hingga *Graph Matching*. Pada bab ini juga akan diuraikan mengenai alat dan bahan yang digunakan selama proses penelitian.

3.1 Desain Sistematika Penelitian

Penelitian ini dilakukan untuk membangun sebuah program analisis berbasis *graph* yang dapat digunakan untuk mengenali citra digital. Seluruh proses analisis mulai dari segmentasi, ekstraksi fitur, dan pengklasifikasian akan dilakukan dengan metode berbasis *graph*. Data yang digunakan berupa citra digital sederhana yang merepresentasikan sebuah objek. Adapun untuk proses pengujian, akan dilakukan dengan menguji setiap data terhadap keseluruhan dataset. Gambar 3.1 menunjukkan metode penelitian yang akan dilakukan.



Gambar 3.1 Metode Penelitian

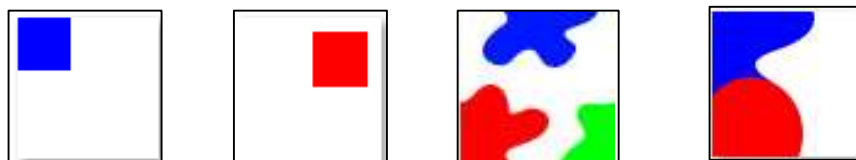
3.1.1 Studi Literatur

Pada tahap ini dilakukan studi literatur terhadap penelitian-penelitian yang terkait dengan operasi pengenalan citra berbasis *graph*, mulai dari segmentasi, ekstraksi fitur, dan pengklasifikasian. Pada tahap ini dipelajari teknik-teknik yang telah menjadi *state-of-the-art* dari jurnal (Sharma, et al., 2012) pada studi kasus *Histological Images*, sehingga dapat menjadi rujukan dalam pembangunan metode penelitian.

Pada tahap ini juga dipelajari bagaimana penentuan spesifikasi citra digital agar memenuhi dalam penelitian ini, sehingga diharapkan dapat bekerja optimal dalam proses pengenalan citra pada metode yang dilakukan.

3.1.2 Persiapan Data

Pada tahap ini akan dilakukan persiapan data sesuai dengan batasan masalah yang telah ditentukan sebelumnya. Data sintetis yang dibuat dalam proses penelitian ini sebanyak 200 buah citra digital dengan ukuran 64 x 64 piksel. Data terdiri dari 4 kelas berbeda sehingga masing-masingnya berjumlah 50 buah. Sampel data citra ditunjukkan pada Gambar 3.2 Sampel Dataset Citra Sintetis.



Gambar 3.2 Sampel Dataset Citra Sintetis

Data tersebut dibuat menggunakan Adobe Photoshop CS6 64 bit. Citra digambar secara manual pada halaman 64 x 64 dengan menambahkan sejumlah

shape dengan warna yang telah ditentukan. Kemudian masing-masing citra disimpan dalam ekstensi berkas *.jpg di dalam media penyimpanan.

Untuk dataset *real* yang digunakan adalah motif inti batik. Motif inti batik yang digunakan pada penelitian ini terdiri dari 3 varian: 1) Batik Grompol, 2) Batik Parang, dan 3) Batik Kawung. Setiap jenis motif batik terdiri masing-masing 60 citra untuk dimasukkan ke dalam data set. Citra dipersiapkan melalui *adobe photoshop* dengan teknik *cropping* dari citra batik utuh. Gambar 3.3 menunjukkan contoh hasil pengambilan motif inti batik dari citra batik utuh.



Gambar 3.3 Contoh hasil pengambilan motif inti batik

Dalam bahasa pemrograman *python* penginputan citra dibantu dengan modul *io* dari *library scikit-image*, kemudian setiap citra direpresentasikan dalam matriks 3 dimensi dengan nilai $64 \times 64 \times 3$. Dimana nilai 64 pertama menunjukkan baris piksel, nilai 64 kedua kolom piksel, dan nilai 3 menunjukkan jumlah *channel* pada citra. Visualisasi dari representasi tiap piksel pada bahasa pemrograman *python* ditunjukkan oleh Gambar 3.4 Representasi Satu Piksel dalam Python



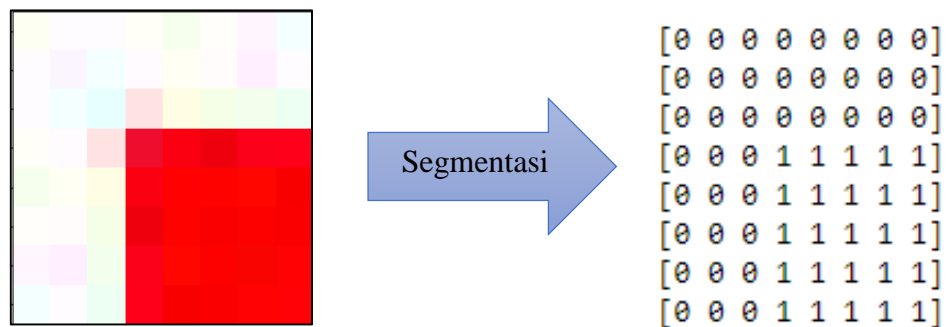
Gambar 3.4 Representasi Satu Piksel dalam *Python*

3.1.3 Segmentasi Citra

Pada tahap ini akan dilakukan proses segmentasi citra menjadi bagian-bagian yang lebih sederhana. Metode yang digunakan pada penelitian ini merupakan segmentasi melalui pendekatan berbasis *graph* dengan metode yang diusulkan oleh Felzenswalb.

Proses segmentasi ini dibantu dengan *library* dari *scikit-image* melalui sebuah fungsi *segmentation.felzenswalb*. Kemudian dilakukan penyetelan terhadap parameter dari fungsi tersebut, untuk mendapatkan hasil segmentasi sesuai dengan persepsi jumlah *region* yang seharusnya.

Input pada tahap ini adalah citra digital $64 \times 64 \times 3$. Setelah diproses, *output* dihasilkan berupa label dengan tipe data *array* n -dimensi ukuran 64×64 yang menunjukkan label *region* untuk setiap piksel. Gambar 3.5 menunjukkan contoh sederhana segmentasi pada citra $8 \times 8 \times 3$, sehingga didapatkan label berupa array 8×8 . Selanjutnya label akan disimpan dalam format **.csv* di dalam media penyimpanan.

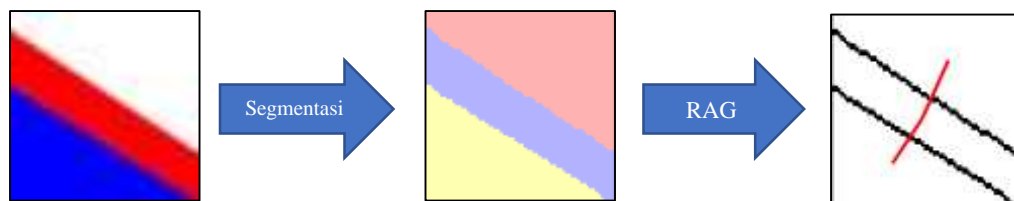


Gambar 3.5 Representai hasil segmentasi

3.1.4 Ekstraksi Fitur melalui Pembangunan RAG

Di tahap ini akan dilakukan ekstraksi fitur pada setiap citra yang telah tersegmentasi melalui pembangunan RAG. RAG dibangun dengan menghitung nilai *centroid* dari setiap label segmentasi, menghitung nilai rata-rata RGB pada setiap *centroid*, kemudian menghubungkan antar *centroid* menggunakan *euclidean distance*. Gambar 3.6 merupakan penggambaran proses ekstraksi RAG.

Output dari tahap ini akan menghasilkan sebuah *graph* untuk masing-masing



Gambar 3.6 Penggambaran proses ekstraksi RAG

citra yang berperan sebagai hasil ekstraksi fitur dalam penelitian ini. Fitur-fitur yang dihasilkan dari setiap *graph* berupa:

1. *Nodes*, simpul-simpul yang terbentuk
2. *Total Color*, total nilai piksel RGB dari masing-masing simpul

3. *Centroid*, nilai titik tengah dari masing-masing simpul
4. *Mean Color*, nilai rata-rata piksel dalam RGB dari masing-masing simpul
5. *Edges*, sisi-sisi yang terbentuk dari setiap simpul
6. *Weight*, nilai *euclidean distance* dari masing-masing *edge*

Melalui bantuan *library Networkx* pada bahasa pemrograman *python*, *graph* yang dihasilkan akan memiliki tipe data berupa *class* dari *library* tersebut. Kemudian *graph* dari RAG masing-masing citra akan disimpan pada media penyimpanan dalam format berkas *.pickle.

3.1.5 Graph Matching

Pada langkah ini akan diberikan citra sebagai masukan untuk memperoleh data yang memiliki kemiripan dengan citra tersebut berdasarkan algoritma *graph matching* yang ditetapkan. Adapun algoritma yang digunakan yaitu algoritma *VF2* dan *Graph Edit Distance*. *Output* dari langkah ini berupa citra digital yang memiliki kemiripan dengan citra uji yang diberikan dalam setiap percobaan.

Citra yang akan diujikan diambil dari dataset, dan akan diproses terhadap setiap dataset. Citra tersebut akan diberikan perlakuan yang sama seperti pada dataset yaitu segmentasi, ekstraksi fitur RAG, kemudian dilakukan pencocokkan terhadap keseluruhan dataset.

Algoritma *VF2* bekerja berdasarkan prinsip *graph isomorphism*, sehingga pada algoritma ini tidak dibutuhkan parameter tertentu sebagai nilai masukan dalam penelitian ini. Sedangkan untuk *Graph Edit Distance* dapat dilakukan penyetelan

parameter berupa batasan nilai *cost* untuk setiap hasil pencocokkan, sehingga hasilnya dapat dibuat lebih fleksibel.

3.2 Alat dan Bahan Penelitian

Dalam penelitian untuk mengenali citra melalui pendekatan berbasis *graph* ini dibutuhkan perangkat keras dan perangkat lunak sebagai berikut.

1. Perangkat Keras

Perangkat keras yang digunakan adalah sebuah komputer yang memiliki spesifikasi sebagai berikut:

- *Processor* : Intel® Celeron® CPU B820 @ 1.70GHz
- *RAM* : 4.00 GB
- *Disk Space* : HDD 500 GB 5400 rpm
- *Video Graphics* : Intel HD Graphic

2. Perangkat Lunak

Perangkat lunak yang digunakan adalah sebagai berikut:

- Sistem Operasi : Windows 8.1 Enterprise 64-bit
- Bahasa Pemrograman : Python 2.7
- Aplikasi : Notepad ++, Sublime Text 3, Adobe Photoshop CS6, Paint
- Library Tambahan : Numpy, Scikit-Image, Matplotlib, NetworkX 1.9, Virtualenv, Flask, Jupyter notebook

3.3 Perancangan Program Aplikasi Antarmuka

Perancangan program aplikasi antarmuka sederhana ini dilakukan guna memudahkan *end-user* dalam menggunakan program hasil analisis dari penelitian ini. Dalam program antarmuka yang dibangun ini akan memuat program yang dapat mempermudah dalam menampilkan hasil segmentasi dan hasil pencocokkan citra-citra yang dianggap memiliki kemiripan untuk satu masukan citra.

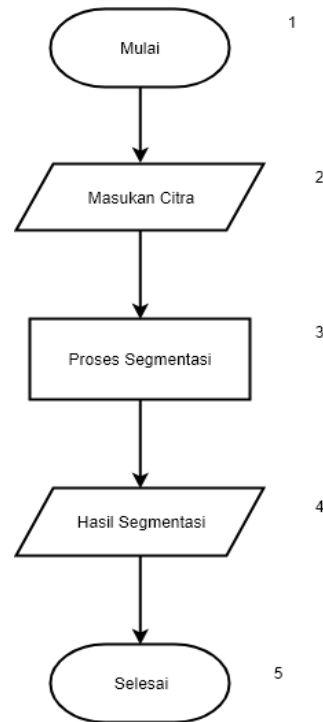
Citra yang dimasukan akan diproses dengan parameter statis, baik pada tahap segmentasi maupun *graph matching*. Parameter tersebut merupakan hasil analisis dari program konsep dalam *jupyter notebook*. Sehingga parameter yang dapat diubah hanya citra masukannya saja.

3.3.1 Flowchart Program

Pada program aplikasi antarmuka ini akan dibangun dua program utama, yaitu program untuk menampilkan segmentasi pada citra masukan dan program untuk pemrosesan *graph matching* serta menampilkan hasilnya. Program-program berikut akan dibangun menggunakan bahasa pemrograman python, dan menggunakan html sebagai media antarmuka.

a. *Flowchart* Segmentasi Citra

Berikut cara kerja program untuk menampilkan segmentasi pada citra:



Gambar 3.7 *Flowchart* Segmentasi Citra

Seperti yang ditunjukkan pada Gambar 3.7 dijelaskan *flowchart* program untuk menampilkan segmentasi citra serta mencetak nilai-nilai label citra hasil segmentasi. Berikut adalah penjelasan *flowchart* segmentasi citra:

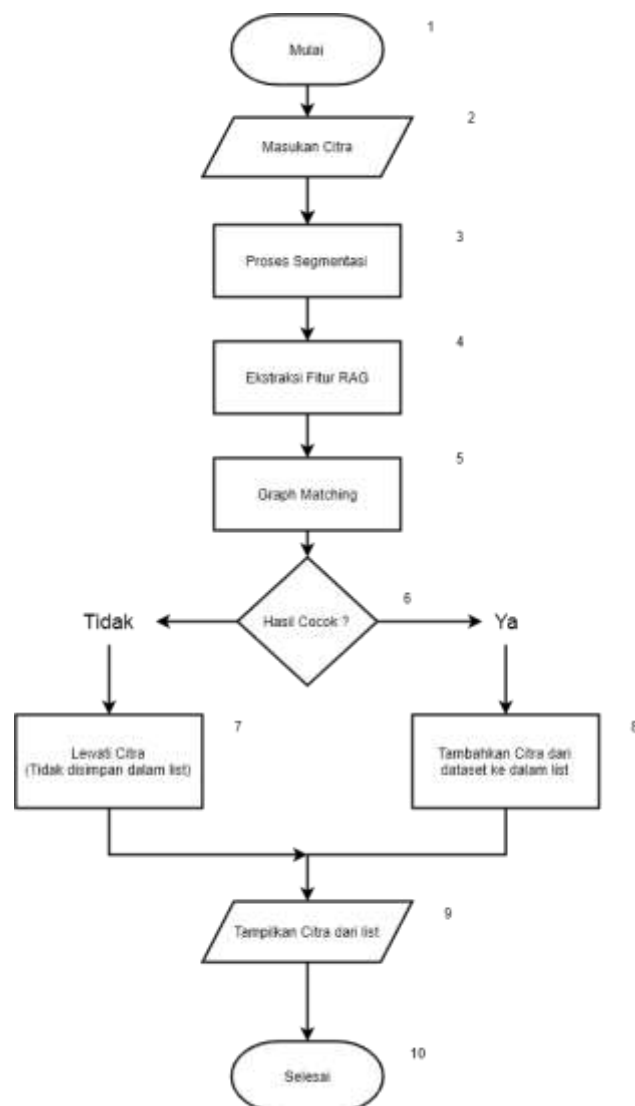
1. Merupakan *state* awal dari aplikasi. Pada *state* ini dilakukan proses pengaktifan halaman *web* melalui *localhost*
2. Pengguna memilih citra yang akan di proses dari folder yang telah ditentukan
3. Segmentasi di dalam program akan menghasilkan nilai-nilai label berukuran *array* 64 x 64 dari citra masukan

4. Hasil segmentasi beserta nilai-nilai label akan ditampilkan pada halaman antarmuka

5. *State* akhir aplikasi

b. *Flowchart* Pencocokan Citra

Berikut cara kerja program untuk pencocokkan citra serta menampilkan hasilnya dalam halaman antarmuka:



Gambar 3.8 *Flowchart* Pencocokkan Citra

Pada Gambar 3.8 dijelaskan *flowchart* pencocokkan citra masukan dengan seluruh citra dalam dataset.

Berikut adalah penjelasan *flowchart* pencocokkan citra:

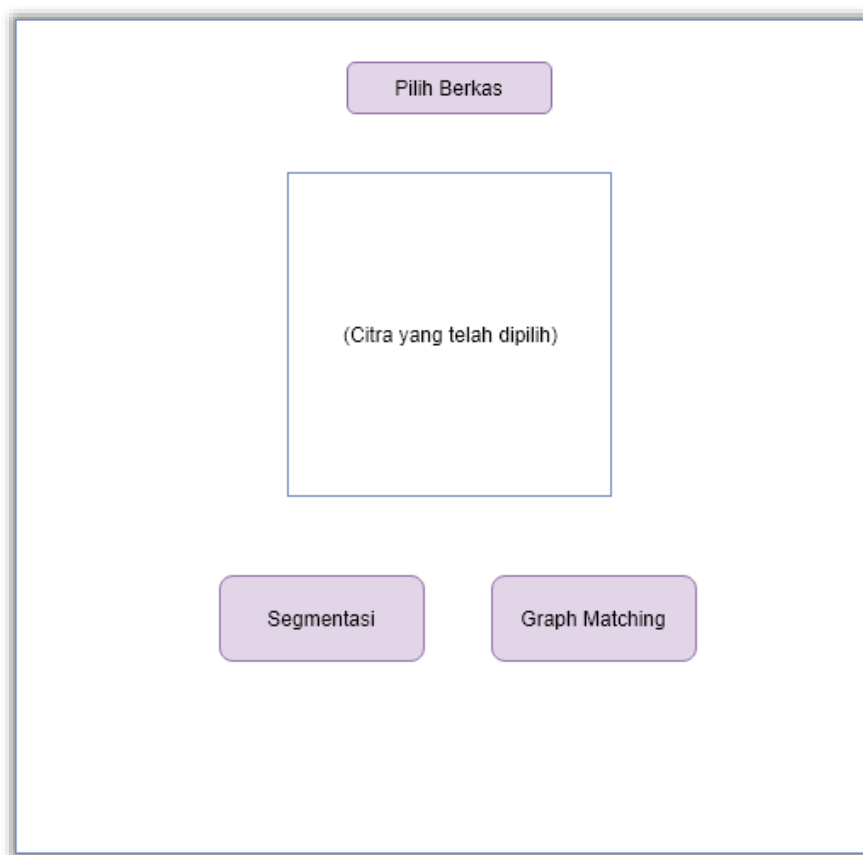
1. Merupakan *state* awal dari aplikasi. Pada *state* ini dilakukan proses pengaktifan halaman *web* melalui *localhost*
2. Pengguna memilih citra yang akan di proses dari folder yang telah ditentukan
3. Segmentasi di dalam program akan menghasilkan nilai-nilai label berukuran *array* 64 x 64 dari citra masukan
4. Ekstraksi fitur akan membangun sebuah RAG dari citra dan label yang telah dihasilkan dari proses segmentasi sebelumnya. RAG yang dihasilkan akan disimpan dalam bentuk *class* yang sesuai untuk *library* *networkx* pada bahasa pemrograman python
5. Proses ini akan melakukan pencocokkan RAG dari citra masukan dengan setiap RAG dalam dataset.
6. Program melakukan pengecekan hasil pencocokkan. Jika citra dianggap cocok, program akan dilanjutkan ke *state* 8. Jika tidak cocok, program akan dilanjutkan ke *state* 7.
7. Citra yang dicocokkan akan dilewati dan tidak disimpan dalam *list*
8. Citra yang dicocokkan akan disimpan dalam *list*
9. Menampilkan citra-citra dari *list* ke dalam halaman *web*.
10. Merupakan *state akhir* dari aplikasi.

3.3.2 Perancangan Desain Antarmuka Program

Desain antarmuka pada program ini akan dikembangkan dalam bentuk halaman *web*. *Server* yang digunakan merupakan *localhost* yang disiapkan dari modul *flask* dari bahasa pemrograman python. Berikut adalah rancangan desain antarmuka dari program aplikasi ini:

a. Halaman Masukan Citra

Halaman masukan adalah halaman awal pada saat program aplikasi dijalankan. Halaman ini ditujukan untuk memilih berkas citra dari folder yang telah ditentukan.



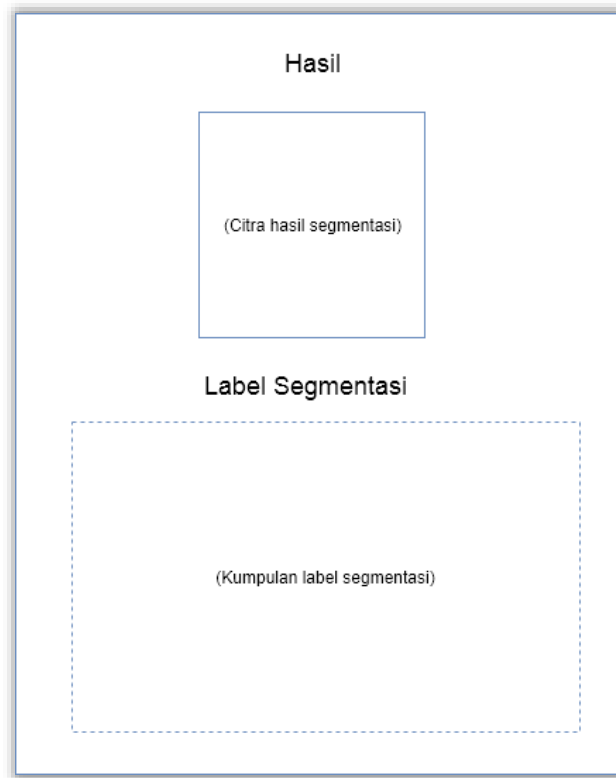
Gambar 3.9 Halaman Masukan Citra

Desain antarmuka halaman masukan pada Gambar 3.9 yang dirancang memiliki:

1. *Button* “pilih berkas” berfungsi untuk memilih berkas citra melalui *file explorer* dari sistem operasi
2. *Image* “citra yang dipilih” berfungsi untuk menampilkan citra yang telah dipilih dari *button* “pilih berkas
3. *Button* “segmentasi” berfungsi untuk memproses serta menampilkan segmentasi dari citra terpilih
4. *Button* “*graph matching*” berfungsi untuk melakukan dan menampilkan hasil dari proses pencocokkan secara keseluruhan pada citra terpilih dengan setiap dataset yang telah disediakan.

b. Halaman Segmentasi Citra

Halaman segmentasi adalah halaman untuk menampilkan hasil segmentasi pada citra secara visual, dan juga mencetak nilai-nilai label dari hasil segmentasi tersebut. Melalui *button* pada halaman inputan, *html* akan mengirimkan *path* berkas citra. Yang selanjutnya akan diproses menggunakan bahasa pemrograman *python* seperti pada program analisis. Kemudian hasil dari program tersebut akan dikirimkan di-*render* kepada halaman *html* segmentasi citra ini.



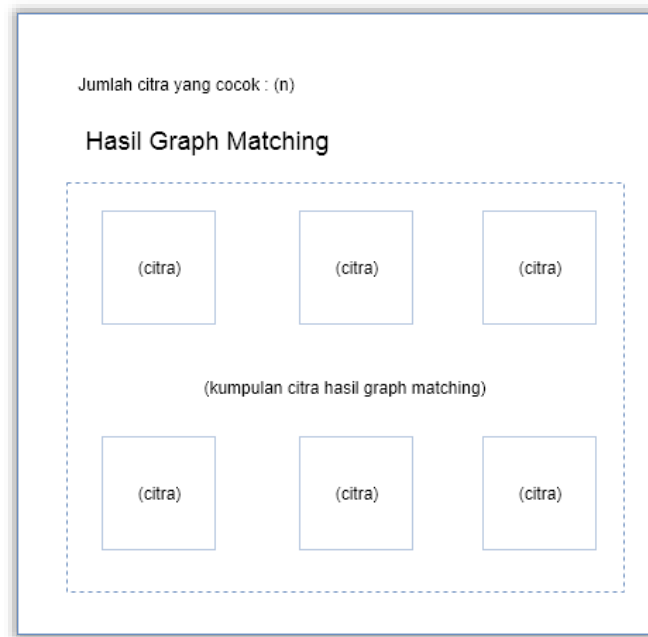
Gambar 3.10 Halaman Segmentasi Citra

Desain antarmuka halaman hasil segmentasi pada Gambar 3.10 yang dirancang memiliki:

1. *Text* “hasil” berfungsi sebagai judul untuk konten citra hasil segmentasi di bawahnya
2. *Image* “citra hasil segmentasi” berfungsi untuk menampilkan citra hasil segmentasi
3. *Text* “label segmentasi” berfungsi sebagai judul untuk konten label citra hasil segmentasi yang berada di bawahnya
4. *Text* “kumpulan label segmentasi” berfungsi untuk mencetak nilai-nilai label hasil segmentasi citra sebanyak 64 x 64

c. Halaman *Graph Matching*

Halaman *graph matching* adalah halaman untuk menampilkan citra hasil proses pencocokan citra masukan dengan setiap pada dataset secara visual, serta mencetak jumlahnya.



Gambar 3.11 Halaman *Graph Matching*

Desain antarmuka halaman hasil *graph matching* pada Gambar 3.10 yang dirancang memiliki:

1. *Text* “jumlah citra cocok: n” berfungsi untuk menunjukkan jumlah citra yang tampil atau dianggap cocok dengan citra masukan
2. *Text* “hasil *graph matching*” berfungsi sebagai judul untuk citra hasil *graph matching* yang berada di bawahnya
3. *Image* “citra” merupakan citra-citra yang ditampilkan sebagai hasil yang dianggap cocok dengan citra masukan dari proses *graph matching*

BAB IV

HASIL DAN ANALISIS

Bab ini membahas mengenai hasil dari *Content Based Image Retrieval* menggunakan ekstraksi RAG dan pengukuran kemiripan *graph matching*. Pada bab ini akan dikaji mengenai penentuan parameter segmentasi dan tingkat akurasi dari metode *Graph Matching* yang digunakan dalam penelitian ini.

4.1 Uji Parameter dan Hasil Segmentasi Citra

Pada segmentasi *felzenszwalb* dari *scikit-image* terdapat 3 buah parameter bebas utama yaitu *scale*, *sigma*, dan *min_size*. *Scale* merupakan parameter bebas yang dapat diatur untuk mengubah besaran setiap *cluster* atau *region*, semakin besar nilai *scale* semakin besar ukuran setiap *cluster*-nya. *Sigma* merupakan parameter yang berfungsi untuk mengatur ukuran *kernel* pada *gaussian smoothing* dalam segmentasi ini. Dan *min_size* merupakan nilai ukuran minimum komponen untuk setiap *cluster*.

Berdasar hasil uji pada referensi pada citra 320 x 240 dengan nilai $scale = 1$, $\sigma = 0.8$, dan $min_size = 300$ (Felzenszwalb, 2004), nilai terbaik untuk parameter *scale* dan *sigma* telah ditetapkan sebagai nilai *default* pada modul di *scikit-image* ($scale = 1$, $\sigma = 0.8$). Maka dari itu, dalam penelitian ini parameter yang disetel hanyalah *min_size* yang nilainya tentu akan berbeda karena bergantung pada ukuran piksel setiap citra. Gambar 4.1 menunjukkan kode dalam bahasa pemrograman python untuk operasi penyetelan parameter segmentasi.

```

# pengulangan uji parameter min_size
for k in range(200,400,10):
    count = np.array([0, 0, 0, 0])
    print ('min_size ->> {}'.format(k))
# uji segmentasi untuk 200 data artificial
for i in range(0,200):
    if(i % 50 == 0):
        print(count)
        print('-----')
    # segmentasi citra
    img = io.imread(aFile[i])
    labels = segmentation.felzenszwalb(img, min_size=k )
    labels = labels.astype(int)
    # cek keunikan label untuk 4 kelas citra
    u = len(np.unique(labels))
    if(i<50 and u != 2):
        count[0] = count[0] + 1
    elif(i>=50 and i<100 and u != 3):
        count[1] = count[1] + 1
    elif(i >= 100 and i<150 and u != 4):
        count[2] = count[2] + 1
    elif(i >= 150 and i<200 and u !=3):
        count[3] = count[3] + 1
    else:
        pass
    # print hasil uji untuk setiap parameter
print(count)
print('jumlah segmen unik : {}'.format(np.sum(count)))

```

Gambar 4.1 Operasi Penyetelan Parameter Segmentasi

Penyetelan parameter segmentasi dilakukan terhadap citra *artificial* sebanyak 200 buah yang telah disiapkan. Pengujian ini tidak menggunakan *ground truth* sebagai pembandingnya. Karena citra *artificial* yang digunakan sudah sederhana, uji parameter akan dilakukan untuk mencari keunikan jumlah setiap kelompok citra

dengan melakukan *tracing* di sekitar nilai terbaik pada referensi ($min_size = 300$). Semakin kecil nilai keunikan *cluster* pada setiap kelas citra, maka semakin baik segmentasi yang dilakukan. Berikut merupakan hasil observasi penentuan parameter untuk tahap segmentasi ditunjukkan oleh Tabel 4.1.

Tabel 4.1 Hasil Pengujian Parameter Segmentasi

<i>min_size</i>	Kelompok Citra				Total Cluster
	I	II	III	IV	Unik
200	2	0	27	3	32
210	2	0	25	3	30
220	2	0	23	3	28
230	2	0	23	2	27
240	2	0	22	2	26
250	2	0	22	2	26
260	2	0	23	0	25
270	0	0	21	0	21
280	0	0	21	0	21
290	0	0	20	0	20
300	0	0	22	0	22
310	0	0	22	0	22
320	0	2	24	0	26
330	0	2	24	0	26
340	0	2	25	0	27

Berdasar hasil observasi di atas, nilai keunikan *cluster* terendah didapat pada $min_size = 290$. Adapun untuk parameter setelahnya nilai total keunikan akan semakin tinggi. Sehingga nilai di atas yang akan digunakan untuk penelitian ini.

Setelah menentukan parameter untuk proses segmentasi. Selanjutnya melalui parameter tersebut dilakukan segmentasi terhadap keseluruhan citra *artificial* dan motif batik. Kemudian label segmentasi disimpan dalam format *.csv di dalam sistem operasi. Tabel 4.2 menunjukkan sampel hasil segmentasi citra yang mewakili setiap kelas citra.

Tabel 4.2 Sampel Hasil Segmentasi Citra untuk Setiap Kelas

Nama File	Citra Asli		Citra Segmentasi	Jumlah Segmen
img030.jpg				2
img094.jpg				3
img101.jpg				4
img156.jpg				3
batik020.jpg				5
batik114.jpg				8
batik173.jpg				3

4.2 Ekstraksi RAG

Pada tahap ini tidak ada parameter yang menentukan keakuratan pembangunan RAG. Sehingga seluruh citra akan langsung diproses dengan fungsi yang sama dan *class* hasil pembangunan RAG akan langsung disimpan dalam


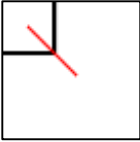

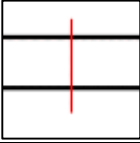

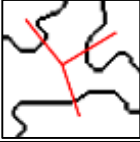
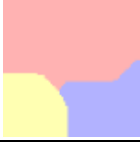


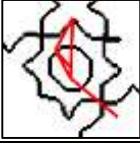

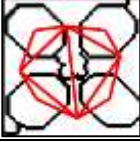
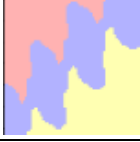

format *.gpickle dalam sistem operasi. Gambar 4.2 menunjukkan operasi pembangunan RAG, serta penyimpanannya dalam sistem operasi.

```
# Pengulangan untuk setiap citra
for i in range(len(imgFile)):
    img = io.imread('blank.jpg')
    # membaca label dari *.csv
    label = genfromtxt(labelFile[i], delimiter=',')
    label = label.astype(int)
    # fungsi utama pembentukan RAG
    g=graph.rag_mean_color(img,label,connectivity=2,mode='distance')
    gshow = graph.show_rag(label,g,img,edge_width=4)
    # operasi penyimpanan graph dalam *.gpickle
    if(i<9):
        name = 'img00{}.gpickle'.format(i+1)
    elif (i<99):
        name = 'img0{}.gpickle'.format(i+1)
    else:
        name = 'img{}.gpickle'.format(i+1)
    path3 = 'graph/'+name
    nx.write_gpickle(g, path3)
```

Gambar 4.2 Operasi Pembangunan RAG

Class RAG yang telah disimpan, tidak dapat dibaca secara manual dalam *text editor*. Namun dapat dibaca kembali dengan mudah melalui fungsi dari modul *networkx.read_gpickle()*. Tabel 4.3 berisi sampel hasil pembangunan RAG yang mewakili setiap kelas citra.

Tabel 4.3 Tabel Hasil Pembangunan RAG

Nama File	Citra Segmentasi	Jumlah Segmen	RAG	Hasil Ekstraksi <i>Graph</i>
img030.jpg		2		Jumlah <i>node</i> : 2 Jumlah <i>edge</i> : 1
img094.jpg		3		Jumlah <i>node</i> : 3 Jumlah <i>edge</i> : 2
img101.jpg		4		Jumlah <i>node</i> : 4 Jumlah <i>edge</i> : 3
img156.jpg		3		Jumlah <i>node</i> : 3 Jumlah <i>edge</i> : 3
batik020.jpg		5		Jumlah <i>node</i> : 5 Jumlah <i>edge</i> : 5
batik114.jpg		8		Jumlah <i>node</i> : 8 Jumlah <i>edge</i> : 13
batik173.jpg		3		Jumlah <i>node</i> : 3 Jumlah <i>edge</i> : 2

4.3 Graph Matching

Pada tahap ini akan diuji tingkat akurasi dari kedua metode *graph matching* yang telah disebutkan sebelumnya yaitu *graph isomorphism* dengan algoritma VF2 dan *error-tolerant matching* menggunakan *graph edit distance*. Kedua algoritma ini akan diujikan terhadap seluruh kelompok citra baik *artificial* maupun motif batik.

Sebelum dilakukan proses *graph matching*, data pemanggilan disimpan dalam bentuk *.json untuk mempermudah dalam pemanggilan berkas-berkas yang dibutuhkan. Adapun struktur penyimpanan data pada *.json yang digunakan untuk setiap *list* data adalah sebagai berikut.

1. *Key*, berfungsi sebagai penanda identitas kelompok citra. Tipe data yang digunakan ‘*string*’
2. *Img*, berfungsi menyimpan *path* untuk berkas citra. Tipe data yang digunakan ‘*string*’
3. *Label*, berfungsi menyimpan *path* untuk berkas label hasil segmentasi. Tipe data yang digunakan ‘*string*’
4. *Graph*, berfungsi menyimpan *path* untuk berkas *graph* hasil ekstraksi RAG. Tipe data yang digunakan ‘*string*’.

List dari berkas *.json tersebut akan digunakan pada tahap analisis evaluasi dari metode *graph matching*. *Key* akan digunakan sebagai pembanding nilai kebenaran pada setiap operasi *graph matching*, sedangkan ketiga variabel lainnya berfungsi sebagai *path* pada pemanggilan masing-masing berkas. Gambar 4.3 menunjukkan potongan program penyimpanan berkas dalam format *.json pada sistem operasi.

```
# inisiasi list
path1 = 'img2/'
path2 = 'labels2'
path3 = 'graph2/'
imgFile = []
labelFile = []
graphFile = []
dataKu = []
# ambil file dari sistem operasi
for filename1 in glob.glob(os.path.join(path1, '*.jpg')):
```

```

        imgFile.append(filename1)
    for filename2 in glob.glob(os.path.join(path2, '*.csv')):
        labelFile.append(filename2)
    for filename3 in glob.glob(os.path.join(path3, '*.gpickle')):
        graphFile.append(filename3)

# penyimpanan dalam dictionary untuk setiap citra
for i in range(len(imgFile)):
    data = {'key': '0', 'img': '0', 'label': '0', 'graph': '0'}
    data['img'] = imgFile[i]
    data['label'] = labelFile[i]
    data['graph'] = graphFile[i]
    if (i < 60):
        data['key'] = 'grompol'
    elif ( i<120 ):
        data['key'] = 'kawung'
    else:
        data['key'] = 'parang'
    # tambah ke list dataKu
    dataKu.append(data)

# simpan dalam *.json dalam sistem operasi
with open('dataKu2.json', 'w') as outfile:
    for hostDict in dataKu:
        json.dump(hostDict, outfile)
        outfile.write('\n')

```

Gambar 4.3 Potongan Program Penyimpanan *.json

4.3.1 Algoritma VF2

Pada algoritma ini, proses *matching* langsung dapat dilakukan tanpa menentukan parameter bebas terlebih dulu. Algoritma ini akan langsung membandingkan kedua *graph* memiliki sifat *isomorphism* bernilai benar atau salah.

Dalam program ini operasi *isomorphism* telah disimpan dalam modul *networkx*. Pemanggilan fungsi dapat dilakukan dengan baris kode *gm.is_isomorphic()*, yang akan menghasilkan output *boolean*. Gambar 4.4 menunjukkan potongan program dari operasi *graph matching* menggunakan algoritma ini.

```

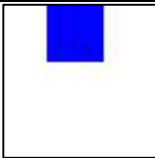
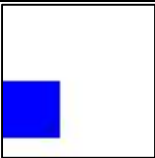






# Pengulangan untuk setiap citra
for i in range(len(aList)):
    # inisiasi ulang list yang cocok
    matchedList = []
    img = io.imread(aList[i]['img'])
    # segmentasi dan ekstrak RAG citra input
    label = segmentation.felzenszwalb(img, min_size=290 )
    g1=graph.rag_mean_color(img,label,connectivity=2,mode='distance')
    # perbandingan graph ke seluruh graph tersimpan
    for j in range(0,len(aList)):
        g2 = nx.read_gpickle(aList[j]['graph'])
        gm = isomorphism.GraphMatcher(g1,g2)
        if(gm.is_isomorphic()):
            matchedList.append(aList[j]['img'])

```

Gambar 4.4 Operasi *Graph Matching* dengan Algoritma VF2

Jika *gm.is_isomorphic()* bernilai *True*, maka data akan ditambahkan ke dalam *list* baru. Sedangkan jika bernilai *False*, data tidak akan diproses. Beberapa hasil dari proses *graph matching* dengan algoritma ini ditunjukkan pada Tabel 4.4.

Tabel 4.4 Sampel Hasil *Graph Matching* melalui Algoritma VF2

Nama Berkas	Citra 1	Citra 2	Hasil Seharusnya	Hasil Prediksi Program
img001.jpg dan img010.jpg			True	True
img061.jpg dan img095.jpg			True	True
batik001.jpg dan batik031.jpg			True	False
batik147.jpg dan batik148.jpg			True	True

4.3.2 Graph Edit Distance

Pada *Graph Edit Distance* (GED), untuk proses *matching* dilakukan dengan menggunakan sebuah parameter yang berfungsi sebagai pembatas nilai *cost* dari transformasi g_1 menjadi g_2 .

Pada program ini algoritma disimpan dalam berkas “ged4py” yang berisi kelas, abstrak, dan algoritma untuk pencarian GED. Hasil dari setiap operasi *matching* melalui GED berupa nilai *cost* dalam tipe data *integer*. Kemudian pembatasan diberikan melalui teknik *tracing* untuk menemukan hasil yang dianggap optimal. Gambar 4.5 menunjukkan potongan operasi utama *graph matching* menggunakan GED.




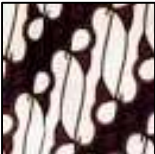



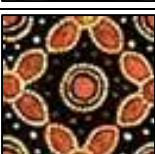
```
# Pengulangan untuk setiap Citra
for i in range(len(aList)):
    # inisiasi list cocok
    matchedList = []
    img = io.imread(aList[i]['img'])
    label = segmentation.felzenszwalb(img, min_size=290 )
    g1 = graph.rag_mean_color(img,label,connectivity=2,mode='distance')
    # matching terhadap seluruh citra
    for j in range(0,len(aList)):
        g2 = nx.read_gpickle(aList[j]['graph'])
        # hitung cost GED dan berikan nilai pembatas pada fungsi if
        if(ged.compare(g1,g2)) <= 8:
            matchedList.append(aList[j]['img'])
```

Gambar 4.5 Operasi *Graph matching* menggunakan *Graph Edit Distance*

Pada fungsi *ged.compare*(g_1, g_2), *output* yang dihasilkan berupa nilai *cost* untuk mentransformasikan g_1 menjadi g_2 . Jika *ged.compare*(g_1, g_2) ≤ 8 atau *cost* ≤ 8 bernilai *True*, maka data akan ditambahkan ke dalam *list* baru. Sedangkan

jika bernilai *False*, data tidak akan diproses. Beberapa hasil dari proses *graph matching* dengan algoritma ini ditunjukkan pada Tabel 4.5.

Tabel 4.5 Hasil *Graph Matching* untuk $GED \leq 8$

Nama Berkas	Citra 1	Citra 2	Hasil Seharusnya	Hasil Prediksi Program
batik001.jpg dan batik031.jpg			<i>True</i>	<i>True</i>
batik147.jpg dan batik148.jpg			<i>True</i>	<i>True</i>
batik001.jpg dan batik090.jpg			<i>False</i>	<i>True</i>
Batik151.jpg dan batik058.jpg			<i>False</i>	<i>True</i>

4.3.3 Hasil Evaluasi *Graph Matching*

Pada penelitian ini evaluasi dilakukan kedua jenis data terhadap data sesamanya, *artificial* terhadap *artificial*, dan motif batik terhadap motif batik. Setiap data dibandingkan dengan seluruh data sehingga pada citra *artificial* didapat masing-masing 200 buah nilai *precision*, *recall*, dan, *f₁ score*, sedangkan pada citra motif batik didapat 180 buah nilai *precision*, *recall*, dan, *f-score*. Kemudian untuk setiap jenis data, diambil sebuah nilai rata-rata untuk setiap algoritma/parameter yang diberikan. Gambar 4.6 merupakan bagian dari program evaluasi hasil *graph matching*.

```

aList = []
with open('dataKu1.json') as f:
    for line in f:
        j_content = json.loads(line)
        aList.append(j_content)

precisionList = []
recallList = []
nList = []

for i in range(len(aList)):
    matchedList = []
    relevant = 0
    img = io.imread(aList[i]['img'])
    label = segmentation.felzenszwalb(img, min_size=290 )
    g1=graph.rag_mean_color(img,label,connectivity=2,mode='distance')
    for j in range(0,len(aList)):
        g2 = nx.read_gpickle(aList[j]['graph'])
        gm = isomorphism.GraphMatcher(g1,g2)
        if(gm.is_isomorphic()):
            matchedList.append(aList[j]['img'])
            if (aList[i]['key'] == aList[j]['key']):
                relevant += 1
    n = len(matchedList)
    precision = (relevant/n)
    recall = (relevant/50)
    precisionList.append(precision)
    recallList.append(recall)
    nList.append(n)

p = np.mean(np.array(precisionList))
r = np.mean(np.array(recallList))
n = np.mean(np.array(nList))
fm = (2*p*r)/(p+r)
print ('nilai precision : %.2f %%' % (p*100))
print ('nilai recall : %.2f %%' % (r*100))
print ('nilai f-measure : %.2f %%' % (fm*100))

```

Gambar 4.6 Potongan Program Evaluasi *Graph Matching*

Program di atas dijalankan berulang secara *trial & error* untuk kedua jenis citra. Hal ini dilakukan agar mendapatkan hasil evaluasi yang baik dari kedua algoritma dan parameternya. Tabel 4.6 menunjukkan hasil evaluasi dari proses *graph matching* yang telah diujikan dengan berbagai parameter.

Tabel 4.6 Hasil Evaluasi *Graph Matching*

Data	<i>precision</i> (%)	<i>recall</i> (%)	<i>f₁ score</i> (%)
<i>Artificial (VF2)</i>	96.24	84.52	90.00
<i>Artificial (GED ≤ 1)</i>	97.08	78.94	87.07
<i>Artificial (GED ≤ 2)</i>	97.08	78.94	87.07
<i>Artificial (GED ≤ 3)</i>	46.72	81.12	59.29
<i>Artificial (GED ≤ 4)</i>	38.65	82.40	52.61
<i>Artificial (GED ≤ 5)</i>	31.49	87.56	45.65
<i>Batik (VF2)</i>	89.68	21.94	35.26
<i>Batik (GED ≤ 1)</i>	91.47	5.57	10.51
<i>Batik (GED ≤ 5)</i>	77.68	24.91	37.72
<i>Batik (GED ≤ 7)</i>	68.30	41.54	51.66
<i>Batik (GED ≤ 8)</i>	51.09	57.07	53.92
<i>Batik (GED ≤ 9)</i>	41.54	64.37	50.49

Berdasarkan tabel di atas, hasil evaluasi terbaik pada data citra *artificial* diperoleh melalui algoritma VF2 dengan nilai *f₁ score* 90.00 %. Baik nilai *precision* maupun *recall* untuk algoritma ini memiliki nilai yang baik yaitu 96.24 % dan 84.52 %. Namun nilai *precision* tertinggi dicapai pada $GED \leq 1$, yaitu 97.08 %. Kemudian pada data citra motif inti batik, hasil akurasi tertinggi dicatat pada *Graph Edit Distance* dengan parameter $GED \leq 8$. Nilai *f₁ score* yang diperoleh sebesar 53.92 %, serta *precision* dan *recall* yaitu 51.09 % dan 57.07 %. Namun nilai *precision* tertinggi didapat pada $GED \leq 1$, yaitu 91.47 %.

4.4 Uji Coba Halaman Antarmuka

Pada tahap ini akan diujikan program dengan parameter yang telah dianalisa sebelumnya ke dalam bentuk program antarmuka berbasis *web*. Halaman yang akan diujikan meliputi halaman masukan, halaman segmentasi, dan halaman hasil pengambilan citra.

4.4.1 Halaman Masukan

Halaman ini diberi nama halaman `input.html`, dibangun menggunakan *html* dan *javascript*. *Html* berperan dalam pembuatan tampilan *web*, sedangkan *javascript* digunakan dalam proses pengambilan citra untuk membantu menampilkan citra dalam halaman *web*. Halaman ini ditampilkan dengan mengetikkan `localhost:4555` di *address bar* pada *browser*. Gambar 4.7 merupakan potongan program yang digunakan dalam pembangunan halaman ini.

```
<body>
<div class="container" align="center">
  <div class="jumbotron">
    <h1>Image Retrieval</h1>
  </div>
  <hr>
  <form id="upload-form" action="{{ url_for('process') }}" method="POST"
  enctype="multipart/form-data" class="form-horizontal">
    <input id="file-picker" class="custom-file-input" class="btn btn-default"
    type="file" name="file" accept="image/jpg" multiple>
    <span class="custom-file-control"></span>
    <br><br>
    <div id="msg"></div>
    
    <br><br>
    <input type="submit" name="submit" value="Segment" class="btn btn-
    default">
    <input type="submit" name="submit" value="Search" class="btn btn-
    primary">
  </div>
</body>
```

Gambar 4.7 Potongan halaman `input.html`

Dapat dilihat pada Gambar 4.8 bahwa keseluruhan fungsi pada halaman ini sudah dapat beroperasi. Mulai dari fungsi pilih berkas, hingga menampilkannya dalam kotak citra yang tersedia dan menyimpan alamat berkasnya dalam program. Begitupun fungsi dari kedua *button*, yang akan dijelaskan di sub bab berikutnya.



Gambar 4.8 Halaman Masukan Citra

4.4.2 Halaman Segmentasi

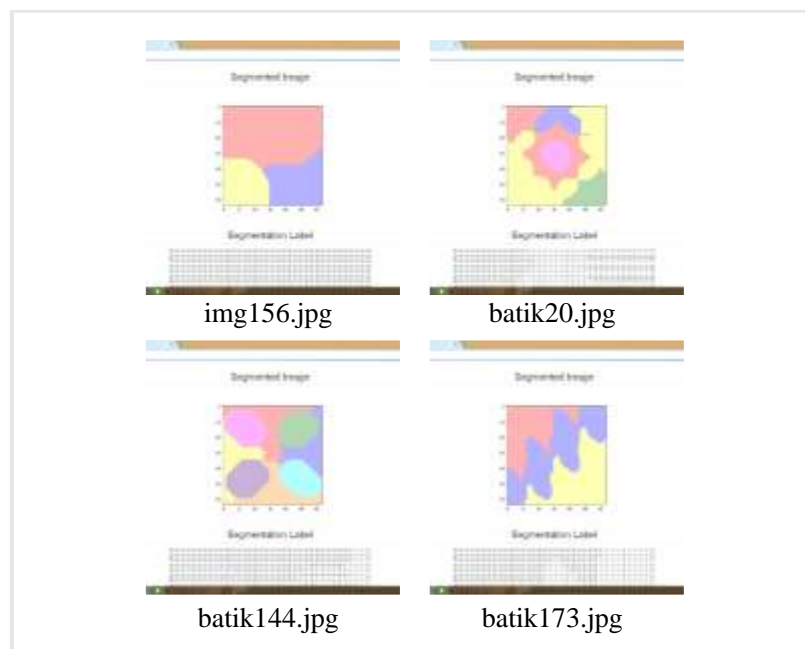
Halaman ini diberi nama *segmentation.html*, terdiri dari *html* dan *python*. *Html* berperan dalam pembuatan tampilan *web*, sedangkan *python* digunakan untuk program segmentasi. Halaman ini diakses dengan menekan *button* “*segment*” pada halaman *input.html*. Gambar 4.9 menunjukkan potongan program untuk *segmentation.html*.

```
<body>
<div class="container" align="center">

    <div class="row">
        <h1 class="page-header">Segmented Image</h1>
        <hr>
        
        <h1 class="page-header">Segmentation Label</h1>
        <hr>
        {% for i in labels %}
            <div >{{ i }}</div>
        {% endfor %}
        <hr>
    </div>
</div>
</body>
```

Gambar 4.9 Potongan Program *Segmentation.html*

Hasil proses dari operasi segmentasi pada *python* dapat di-*render* dan dicetak kembali pada *segmentation.html*. Pada Gambar 4.10 ditunjukkan halaman hasil segmentasi dari beberapa sampel citra masukan.



Gambar 4.10 Beberapa Sampel Halaman Hasil Segmentasi

4.4.3 Halaman Hasil *Matching*

Halaman ini diberi nama *halaman output.html*. Terdiri dari *html* dan *python* yang berperan sama seperti pada *segmentation.html*. Halaman ini dapat diakses dengan *button* “*search*” dengan keterangan algoritmanya di *halaman input.html*.

Berkas citra yang telah disimpan dari halaman masukan, kemudian diproses dalam *python* melalui algoritma *graph matching* seperti pada program analisis. Adapun algoritma dan parameter yang digunakan merupakan hasil terbaik yang diambil dari hasil evaluasi penelitian. Skrip kode untuk halaman ini ditunjukkan oleh Gambar 4.x.

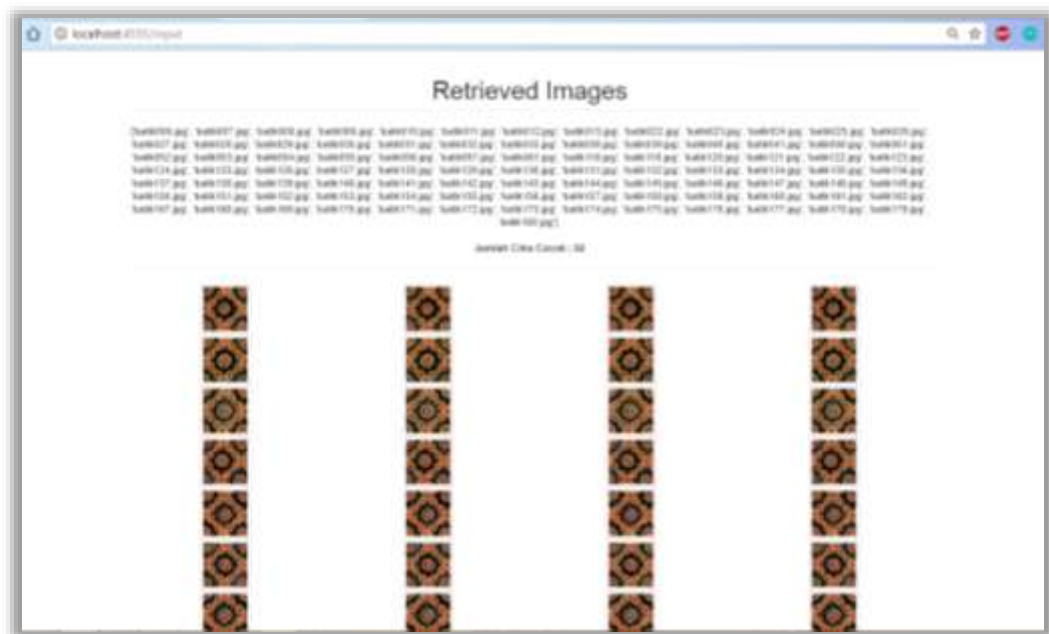
```

<body>
<div class="container" align="center">
  <div class="row">
    <div class="col-lg-12">
      <h1 class="page-header">Retrieved Images</h1>
    </div>
    {{image_names}}
    <br><br>
    <b>Jumlah Citra Cocok : {{jumlah}} </b>
    <hr>
    {% for image_name in image_names %}
    <div class="col-lg-3 col-md-5 col-xs-6 thumb">
      
    </div>
    {% endfor %}
  </div>
</div>
</body>

```

Gambar 4.11 Program *Halaman Output.html*

Untuk data citra *artificial* algoritma *graph matching* yang terbaik adalah algoritma VF2. Sedangkan untuk data citra motif inti batik yaitu *Graph Edit Distance* dengan parameter $GED \leq 8$. Gambar 4.9 menunjukkan halaman hasil *Graph Matching* citra masukan dengan data citra motif inti batik.



Gambar 4.12 Halaman Hasil *Graph Matching* untuk Citra Motif Inti Batik

BAB V

SIMPULAN DAN SARAN

Bab ini merupakan penutup dari skripsi yang berisi kesimpulan dan saran yang diambil dari pembahasan pada skripsi ini.

5.1 Simpulan

1. Dataset yang digunakan merupakan citra *artificial* dan motif inti batik, yang kontennya berupa kumpulan klaster-klaster.
2. Proses *pre-processing* dilakukan dengan *cropping* dan segmentasi Felzenszwalb terhadap citra berwarna (RGB), dengan parameter segmentasi $min_size = 290$.
3. Jumlah klaster hasil segmentasi mempengaruhi jumlah simpul dan sisi yang terbentuk pada RAG.
4. Algoritma dan parameter *graph matching* yang optimal pada penelitian ini adalah algoritma VF2 untuk citra *artificial* dengan nilai f_1score sebesar 90.00 %, dan *Graph Edit Distance* dengan parameter $GED \leq 8$ untuk citra motif inti batik dengan f_1score sebesar 53.92 %.
5. Pada penelitian ini menunjukkan bahwa penerapan algoritma *graph matching* pada RAG citra motif inti batik mampu memberikan nilai *precision* terbesar hingga 91.47 %.

5.2 Saran

1. Pengembangan lebih lanjut dapat menggunakan data yang memiliki klaster-klaster lebih kompleks dan perbedaan warna yang kontras. Serta kembangkan metode dengan menggunakan atribut RAG lainnya seperti *mean color*, *total color*, dan *weight* untuk mendapatkan nilai-nilai ciri yang lebih spesifik.
2. Evaluasi lebih lanjut pada tahap segmentasi untuk menghasilkan citra tersegmentasi yang baik, sehingga RAG yang dihasilkan akan relatif lebih bagus.
3. Untuk penelitian terkait bidang lain, ganti metode *graph matching* menggunakan metode berbasis *machine learning* atau *deep learning* untuk mendapatkan hasil yang lebih optimal.

DAFTAR PUSTAKA

- Akmal, Suwardi, I., & Munir, R. (2017). *Ekstraksi Graf dalam Analisis Citra Berbasis Graf*. Institut Teknologi Bandung, Sekolah Tinggi Elektro dan Informatika, Bandung.
- Angelia, Y. (2011). *Algoritma Pencocokan Objek Geometri Citra Berbasis Graph untuk Pemilihan Kembali (Retrieval)*. Institut Sepuluh Nopember (ITS), Jurusan Teknik Elektro-FTI, Surabaya.
- Chaudhuri, B., Demir, B., & Bruzzone, L. (2016). Region-Based Retrieval of Remote Sensing Images Using an Unsupervised Graph-Theoretic Approach. *IEEE Geoscience and Remote Sensing Letters*, 987-991.
- Cho, M., & Lee, K. M. (2012). Progressive Graph Matching: Making a Move of Graphs via Probabilistic Voting. *Computer Vision and Pattern Recognition (CVPR)*. Providence, Rhode Island: IEEE.
- Cordella, L. P., Foggia, P., & Vento, M. (2001). *An Improved Algorithm for Matching Large Graphs*. Semantic Scholar.
- Felzenszwalb, P. &. (2004). Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*.
doi:<https://doi.org/10.1023/B:VISI.0000022288.19776.77>
- Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference (SciPy2008)*, (pp. 11-15). Pasadena.

- Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9, 90-95. doi:10.1109/MCSE.2007.55
- Munir, R. (2004). *Pengolahan Citra Digital*. Bandung: Informatika.
- Perry, J. W., Kent, A., & Berry, M. M. (1955). Machine Literature Searching X. Machine language; Factors Underlying Its Design and Development. *Journal of the Association for Information Science and Technology*. doi:10.1002/asi.5090060411
- Riesen, K. (2015). *Structural Pattern Recognition with Graph Edit Distance*. Olten: Springer. doi:10.1007/978-3-319-27252-8
- Sanfeliu, A., & Fu, K.-S. (1983). A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics* (pp. 353 - 362). IEEE. doi:10.1109/TSMC.1983.6313167
- Sharma, H., Alekseychuk, A., Leskovsky, P., Hellwich, O., Anand, R., Zerber, N., & Hufnagl, P. (2012). Determining similiarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics. *Diagnostic Pathology*.
- Tremeau A, C. P. (2000). Regions adjacency graph applied to color image segmentation. *IEEE Trans Image Process*, 9.
- Walt, S. v., Schonberger, J. L., Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., . . . the scikit-image contributors. (2014). Scikit-ImageL Image Processing in Python. *Bioinformatics and Genomics*.
- Yodha, J., & Kurniawan, A. (2014). Pengenalan Motif Batik Menggunakan Deteksi Tepi Canny dan K-Nearest Neighbor. *Jurnal Teknologi Informasi*.

LAMPIRAN

Lampiran 1 Kode untuk Pembuatan *Localhost* dan proses *Graph Matching*

```
import glob
import os
import numpy as np
import StringIO
import base64
import networkx as nx
import matplotlib.pyplot as plt
from numpy import genfromtxt
from skimage import io, segmentation, color
from skimage.future import graph
from networkx.algorithms import isomorphism
from flask import Flask, request, render_template, send_from_directory
from ged4py.algorithm import graph_edit_dist as ged

__author__ = 'wijaya'

app = Flask(__name__)

APP_ROOT = os.path.dirname(os.path.abspath(__file__))

@app.route("/")
def index():
    return render_template("halaman input.html")

@app.route("/input", methods=["POST"])
def process():
    if request.method == 'POST':
        if request.form['submit'] == 'Segment':
            for upload in request.files.getlist("file"):
                print(upload)
                print("{} is the file name".format(upload.filename))
                filename = upload.filename

                filename = filename.encode('utf-8')
                filename = 'static/{}'.format(filename)
                print(filename)
                print(type(filename))
                img = io.imread(filename)
                label = segmentation.felzenszwalb(img, min_size=290 )
                bg = io.imread('blank.jpg')
                out = color.label2rgb(label, bg, kind='overlay')
                pelot = StringIO.StringIO()
                plt.imshow(out)
                plt.savefig(pelot, format='png')
                pelot.seek(0)
                plot_url = base64.b64encode(pelot.getvalue())
                return render_template("segmentation.html", labels = label,
plot_url = plot_url)
            elif request.form['submit'] == 'Search (VF2)':
                for upload in request.files.getlist("file"):
                    print(upload)
```

```

        print("{} is the file name".format(upload.filename))
        filename = upload.filename

        print(filename)
        filename = filename.encode('utf-8')
        filename = 'static/{}'.format(filename)
        print(filename)
        print(type(filename))
        path1 = 'images/'
        path2 = 'graph/'
        imgFile = []
        graphFile = []

        for filename1 in glob.glob(os.path.join(path1, '*.jpg')):
            imgFile.append(filename1)

        for filename2 in glob.glob(os.path.join(path2, '*.gpickle')):
            graphFile.append(filename2)

        matchedList = []
        img = io.imread(filename)
        label = segmentation.felzenszwalb(img, min_size=290)
        print(label)
        g1 = graph.rag_mean_color(img, label, connectivity=2,
mode='distance')
        for i in range(len(graphFile)):
            g2 = nx.read_gpickle(graphFile[i])
            gm = isomorphism.GraphMatcher(g1,g2)
            if(gm.is_isomorphic()): # isomorphism
                matchedList.append(imgFile[i])

        newList = []
        for i in range(len(matchedList)):
            f1 = matchedList[i].replace("images\\", "")
            newList.append(f1)

        n = len(newList)

        return render_template("halaman output.html",
image_names=newList, jumlah=n)
    else:
        for upload in request.files.getlist("file"):
            print(upload)
            print("{} is the file name".format(upload.filename))
            filename = upload.filename

            filename = filename.encode('utf-8')
            filename = 'static/{}'.format(filename)
            print(filename)
            print(type(filename))
            path1 = 'images/'
            path2 = 'graph/'
            imgFile = []
            graphFile = []

            for filename1 in glob.glob(os.path.join(path1, '*.jpg')):
                imgFile.append(filename1)

            for filename2 in glob.glob(os.path.join(path2, '*.gpickle')):
                graphFile.append(filename2)

```

```

        matchedList = []
        img = io.imread(filename)
        label = segmentation.felzenszwalb(img, min_size=290 )
        print(label)
        g1 = graph.rag_mean_color(img, label, connectivity=2,
mode='distance')
        for i in range(len(graphFile)):
            g2 = nx.read_gpickle(graphFile[i])
            gm = isomorphism.GraphMatcher(g1,g2)
            if(ged.compare(g1,g2,False)) <= 8:
                matchedList.append(imgFile[i])

        newList = []
        for i in range(len(matchedList)):
            f1 = matchedList[i].replace("images\\", "")
            newList.append(f1)

        n = len(newList)

        return render_template("halaman output.html",
image_names=newList, jumlah=n)

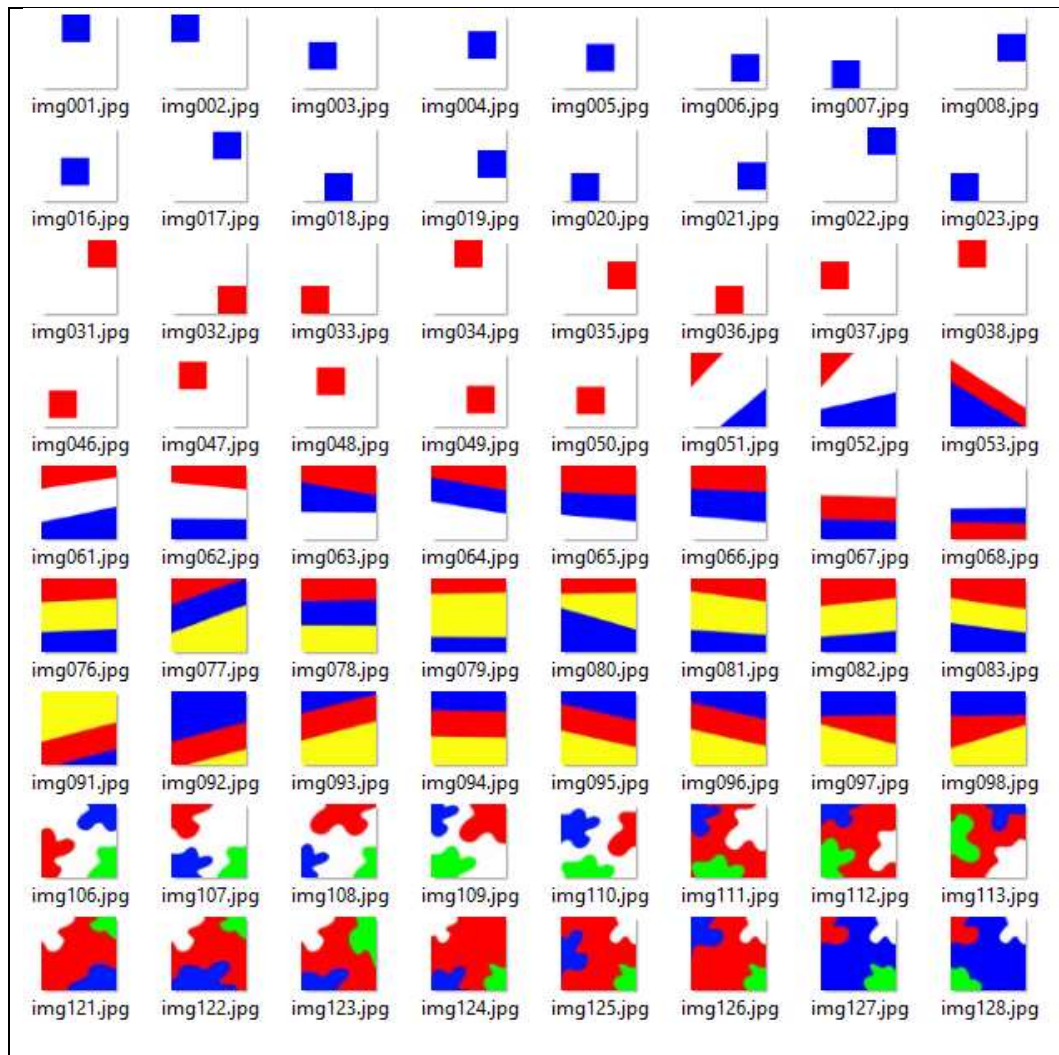
@app.route('/upload/<filename>')
def send_image(filename):
    return send_from_directory("images", filename)

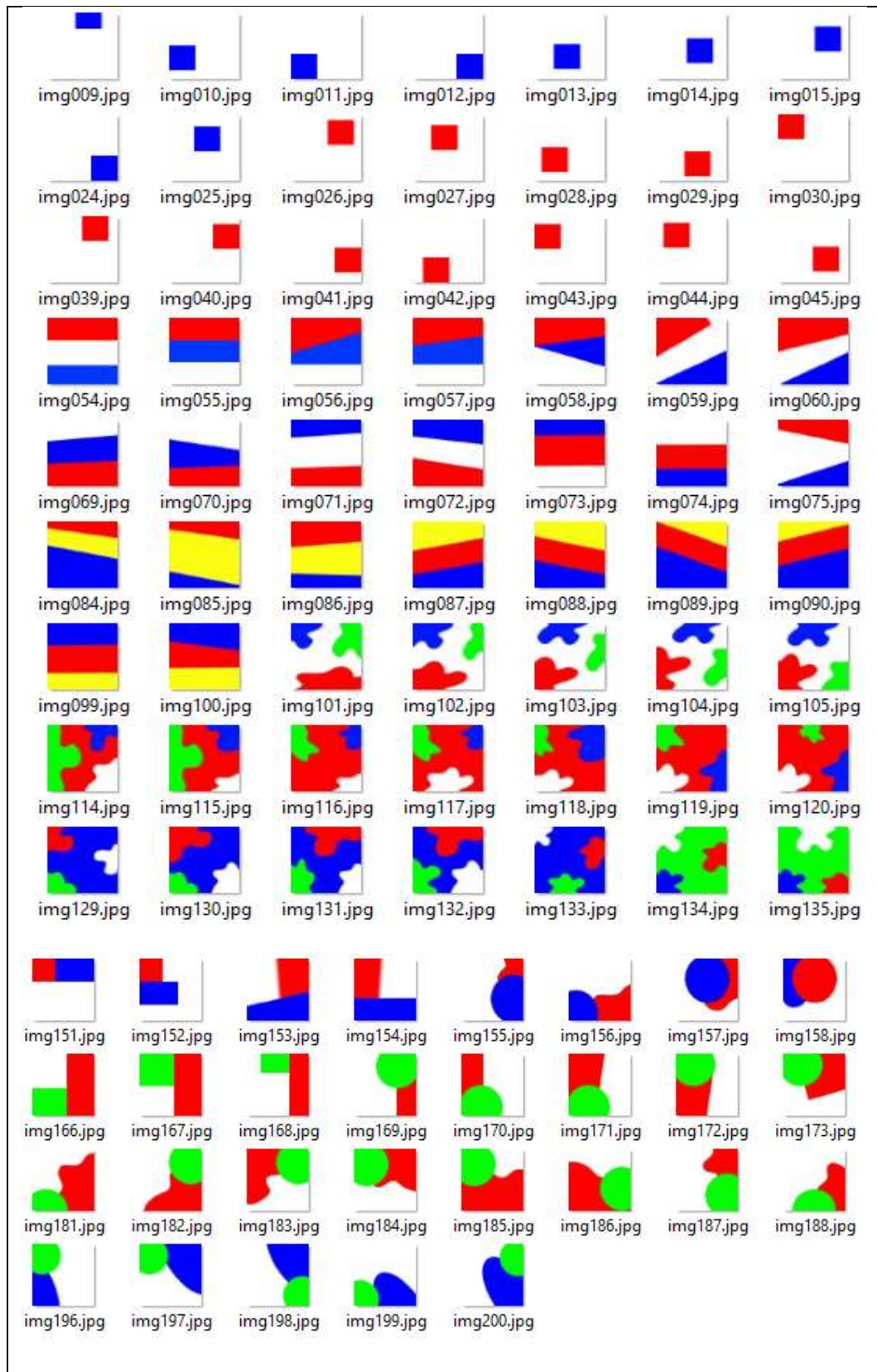
@app.route('/output')
def get_gallery():
    image_names = ['img001.jpg', 'img050.jpg']
    n = len(image_names)
    print(image_names)
    return render_template("halaman output.html", image_names=image_names,
jumlah=n)

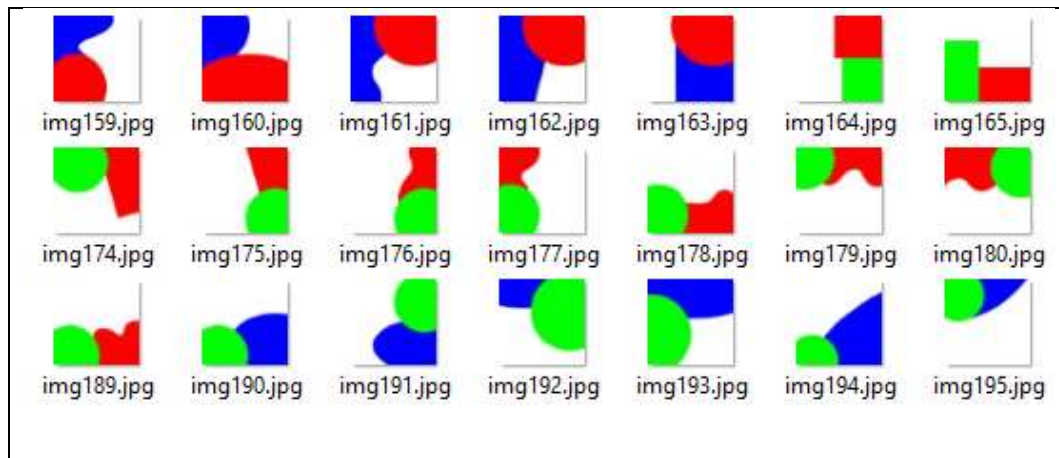
if __name__ == "__main__":
    app.run(port=4555, debug=True)

```

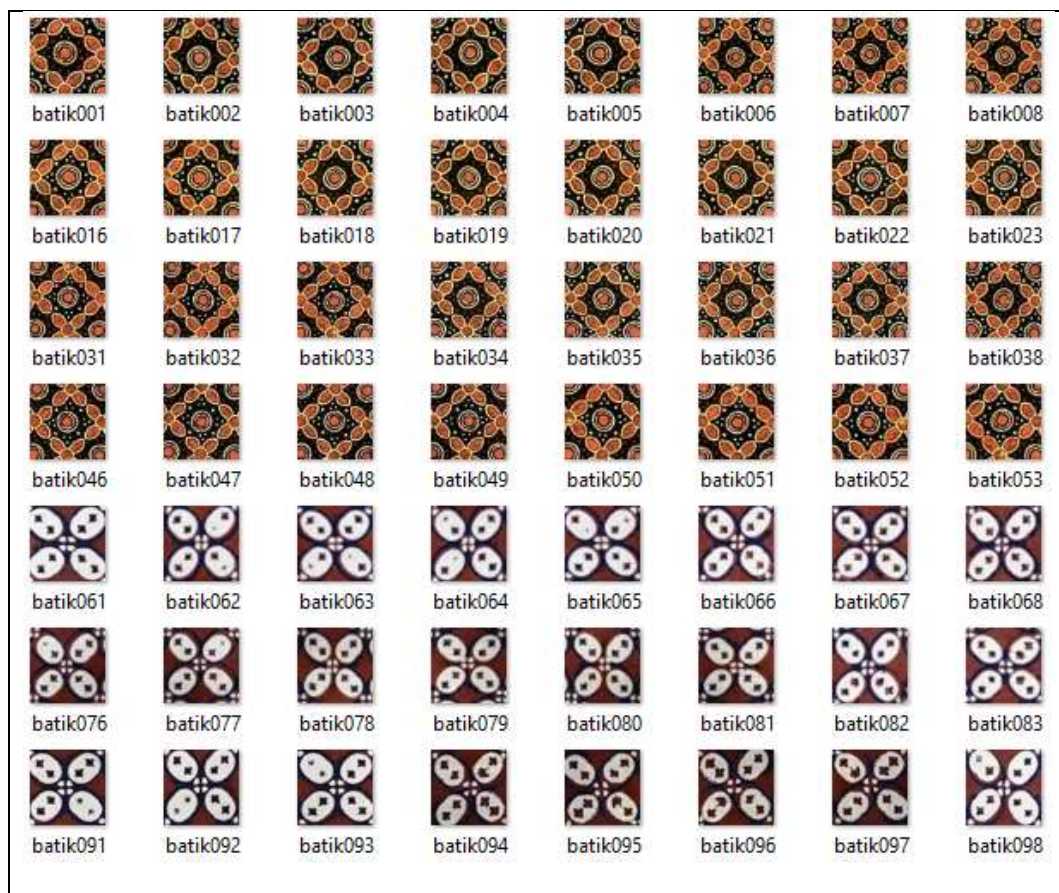
Lampiran 2 Citra *Artificial*

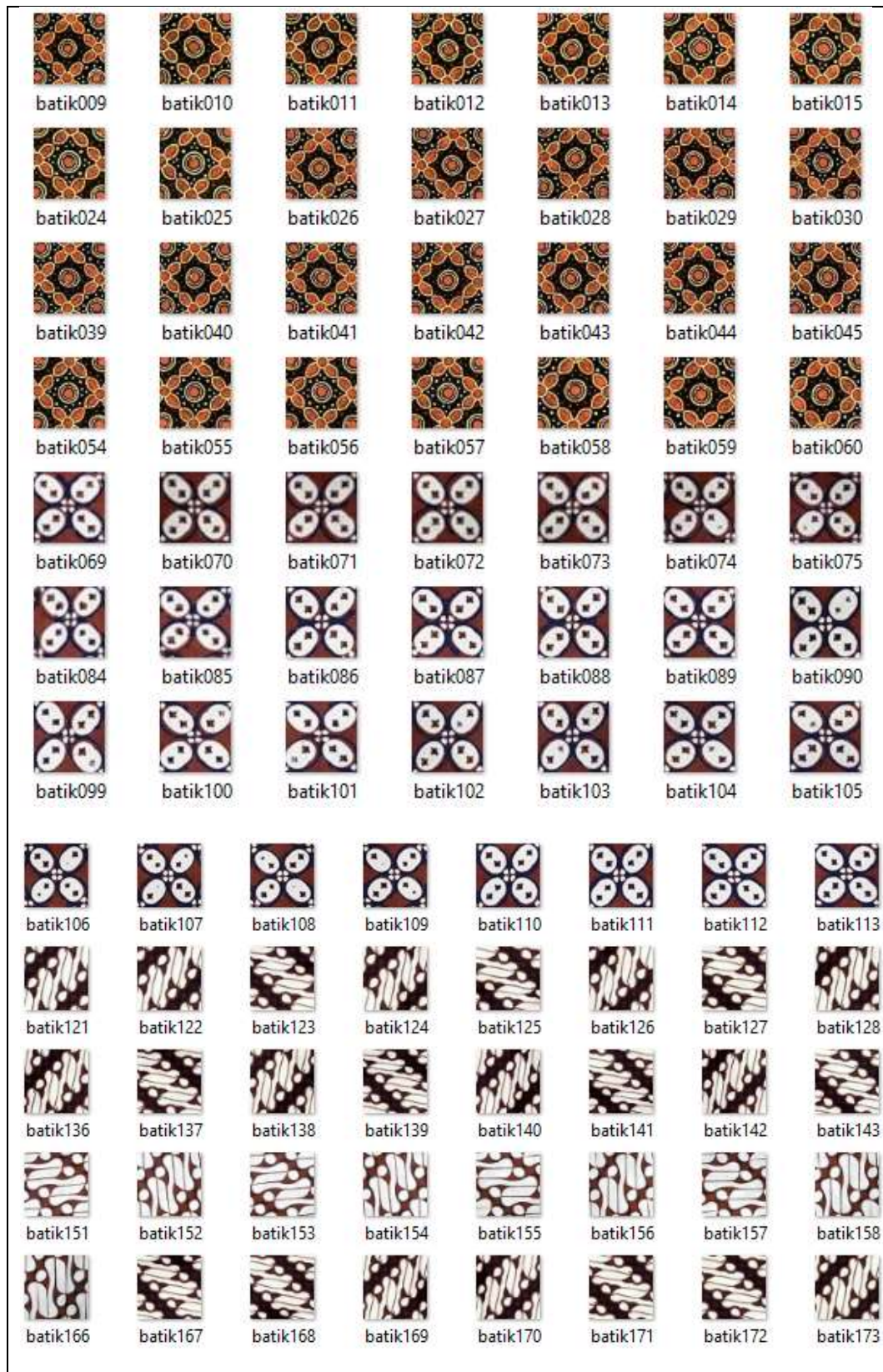


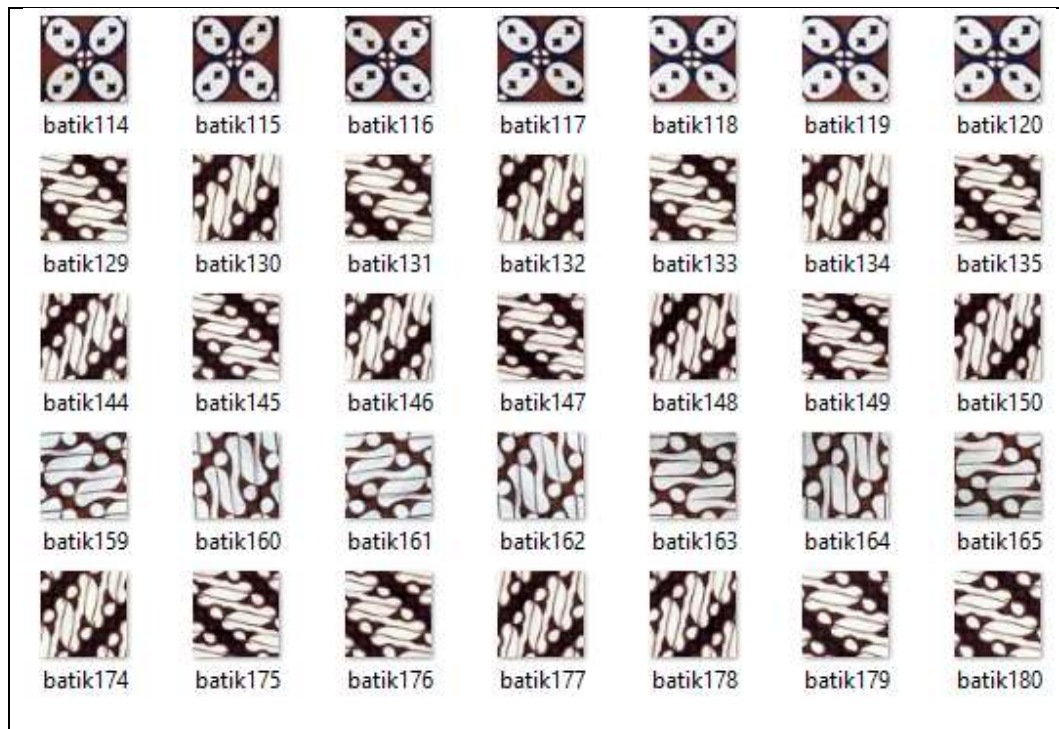




Lampiran 3 Citra Motif Inti Batik







Lampiran 4 Alamat Github Kumpulan Program Skripsi Lengkap.

<https://github.com/mwijayaa/Skripsi-Analisis-Citra-berbasis-Graph/tree/master/Program>