Aplikasi Algoritma *Divide and Conquer* pada Segmentasi Citra

Penggunaan algoritma *divide and conquer* pada segmentasi citra menggunakan metode *split and merge*

Firizky Ardiansyah - 13520095
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (13520095@std.stei.itb.ac.id)

***Abstract*—Segmentasi citra merupakan sebuah proses yang bertujuan untuk mempartisi citra ke dalam objek-objek atau wilayah-wilayah citra tertentu. Dalam pemrosesan citra digital, segmentasi citra digunakan dalam berbagai hal di antaranya dalam rekognisi teks, rekognisi objek, pengolahan citra medis, atau pencarian berbasis citra. *Split and merge* adalah salah satu metode dalam segmentasi citra yang implementasinya berbasis pemisahan wilayah. Metode ini pada implementasinya menggunakan algoritma *divide and conquer* dalam memisahkan wilayah-wilayah dalam citra berdasarkan parameter tertentu. Pada makalah ini akan dibahas pendekatan *split and merge* dalam segmentasi citra, keunggulan metode ini, serta kekurangan dan limitasinya.**

Keywords—segmentasi; divide and conquer; algoritma; citra; split and merge

I. PENDAHULUAN

Pengolahan citra merupakan implementasi komputasi yang krusial dalam meningkatnya kebutuhan terkait visi komputer. Data visual dapat diolah menjadi informasi yang sangat berharga dan saat ini menjadi salah satu perspektif komputasi yang memiliki daya saing tinggi di dunia komputasi. Hal ini berkaitan dengan berkembangnya disiplin ilmu kecerdasan buatan yang salah satu indranya berasal dari data yang dilihat atau data visual.

Perspektif visual dalam dunia komputasi tentunya akan berbeda dengan perspektif data lainnya yang menggunakan objek konkret berupa angka maupun teks. Faktanya, masukan berupa data visual merupakan abstraksi yang paling dekat dengan aktivitas manusia. Manusia menilai impresi sebuah objek salah satunya menggunakan mata, impresi ini kemudian diterima sebagai informasi visual dan diolah sebagai informasi yang lebih deskriptif. Konsep inilah yang digunakan dalam pengolahan objek visual di dunia komputasi.

Citra merupakan objek visual statis yang saat ini menjadi disiplin ilmu yang menarik dalam informatika. Salah satu informasi yang bisa diperoleh dari citra adalah terkait objek-objek di dalamnya. Kemampuan untuk mendeteksi objek berdasarkan data visual yang dimiliki, tentunya akan meningkatkan kualitas respon komputasi salah satunya dalam memberikan data yang lebih deskriptif terkait data visual tersebut.

Segmentasi citra merupakan salah satu proses komputasi dalam pengolahan citra yang bertujuan untuk memisahkan objek-objek atau wilayah-wilayah tertentu di dalam citra. Dalam dunia medis, segmentasi citra misalnya digunakan sebagai pengenalan organ-organ pada hasil *rontgen*. Selain itu, dalam rekognisi objek, segmentasi citra merupakan langkah awal yang mengubah data visual mentah ke dalam wilayah-wilayah warna yang lebih mudah diidentifikasi.

Salah satu metode yang paling sederhana dalam segmentasi citra adalah metode *split and merge*. Metode ini pada dasarnya menggabungkan wilayah-wilayah pada citra yang dianggap homogen berdasarkan parameter tertentu dan memisahkan wilayah-wilayah yang tidak homogen. Metode ini menggunakan algoritma *divide and conquer* dalam pemisahan dan penggabungan wilayah berdasarkan parameter tersebut.

Makalah ini akan membahas penggunaan metode *split and merge* dalam segmentasi citra dengan parameter homogenitas wilayah diperoleh dari kesamaan warna. Performa segmentasi akan dinilai secara kualitatif. Makalah ini juga akan membahas pengaruh ambang toleransi parameter homogenitas terhadap hasil yang diperoleh.

II. LANDASAN TEORI

A. Citra Digital

Citra atau gambar merupakan sinyal dua dimensi yang bersifat *continue* dan dapat diamati secara visual. Sebuah citra dapat didefinisikan dalam sebuah fungsi dua dimensi, $f(x, y)$, menyatakan intensitas citra pada koordinat spasial citra x dan y. Nilai x, y dan f yang berupa kuantitas diskrit disebut sebagai citra digital. Citra digital terbentuk dari elemen-elemen terkecil citra dengan lokasi spasial tertentu dan dengan nilai tertentu. Elemen ini disebut sebagai *pixel* atau elemen citra [3].

Citra digital dapat direpresentasikan sebagai matriks berukuran $M \; x \; N$.

$$f(x,y) = \begin{bmatrix} f(0,0) & \cdots & f(0,N-1) \\ \vdots & \ddots & \vdots \\ f(M-1,0) & \cdots & f(M-1,N-1) \end{bmatrix} \quad (1)$$

Ukuran matriks $M\ x\ N$ disebut sebagai resolusi citra dan setiap elemen dari matriks inilah yang disebut sebagai *pixel* [6]. Operasi terhadap citra digital dilakukan terhadap representasi ini. Nilai *pixel* dari matriks representasi citra dapat berupa intensitas dari elemen citra pada posisi yang bersangkutan atau representasi numerik dari warna.

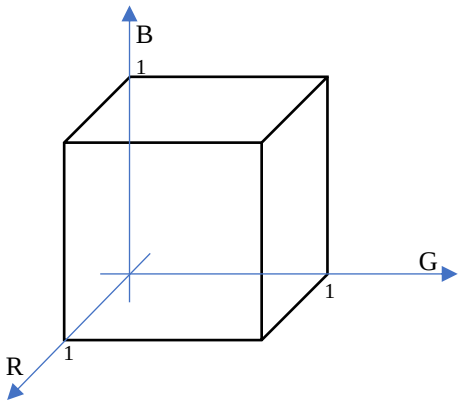
B. *Warna*

Warna adalah *powerful descriptor* dari sebuah citra. Warna dapat digunakan sebagai identifikasi objek dan ekstraksinya. Warna yang diterima manusia dipersepsikan berdasarkan properti dari cahaya yang dipantulkan dari sebuah objek [6].

Penelitian menunjukkan bahwa mata manusia memiliki sensitivitas yang tinggi pada tiga buah warna, yaitu *red* (R), *green* (G), dan *blue* (B). Warna lain, diterima oleh manusia sebagai kombinasi dari ketiga warna tersebut. Karenanya, warna-warna tersebut disebut sebagai warna pokok atau *primary color*. Commission Internationale de l'Eclairage – the International Commission of Illumination atau CIE mendefinisikan gelombang warna pokok *red*, *green*, dan *blue* masing-masing merupakan cahaya dengan panjang gelombang 700 nm, 546.1 nm, dan 435.8 nm [3].

Ketiga warna pokok, *red*, *green*, dan *blue*, dalam citra digital, membentuk menjadi salah satu model warna yang disebut sebagai model warna RGB. Model ini didasarkan pada sistem koordinat tiga dimensi yang sumbu-sumbunya merupakan komponen dari warna pokok. Sebuah titik pada sistem koordinat ini berkorespondensi dengan warna dari kombinasi warna pokok dengan proporsi yang sesuai dengan jaraknya terhadap sumbu warna pokok yang bersangkutan.

Secara formal, model warna RGB direpresentasikan menjadi koordinat tiga dimensi, yaitu tupel dengan nilai (R, G, B). Model ini membentuk subruang berbentuk kubus satuan pada sistem koordinat tiga dimensi. Rentang nilai RGB dengan demikian berada pada interval $[0, 1]$.



Gambar 1. Subruang Model Warna RGB.

Komputer merepresentasikan model RGB menggunakan bit sehingga pada implementasinya, subruang yang tadinya berbentuk kubus satuan, direpresentasikan dalam komputer menjadi subruang kubus dengan panjang sisi sebanyak nilai maksimal dari nilai-nilai yang dapat direpresentasikan oleh bit tersebut. Banyaknya bit yang digunakan dalam representasi model ini disebut sebagai *pixel depth*. Misalnya dalam citra dengan 24 bit *pixel depth*, masing-masing kanal pada model RGB, akibatnya memiliki panjang 8 bit. Degnan demikian, subruang yang dibentuk oleh model ini adalah kubus dengan panjang sisi $2^8 - 1 = 255$ karena nilai maksimal yang dapat direpresentasikan oleh biner dengan panjang 8 bit adalah 255.

Representasi bit dari RGB mengakibatkan koordinat vektor-vektor warna yang bisa direpresentasikan oleh model ini hanyalah koordinat yang latis atau diskrit. Dampak dari ketidakkontinuan representasi komputer salah satunya adalah persepsi dua warna yang berbeda, meskipun tidak signifikan perbedaannya, dapat memiliki representasi model RGB yang sama persis. Selain itu, pengolahan citra secara aljabar biasanya merupakan operasi non-diskrit sehingga untuk diterapkan pada representasi yang diskrit memungkinkan adanya ketidaktelitian atau eror tertentu dalam perhitungan. Beberapa model lain biasanya digunakan sebagai alternatif model RGB, misalnya model HSV yang menggunakan komponen *hue*, *saturation*, dan *value* (*brightness*) dari sebuah cahaya warna tampak. Namun demikian, RGB bukan berarti tidak dapat digunakan dalam pengolahan citra. Model RGB mudah untuk diproses secara komputasi sehingga masih kerap dijadikan alternatif dalam pengolahan citra digital.

C. *Homogenitas Warna*

Persepsi manusia terhadap warna dari satu individu dengan individu yang lain cukup beragam. Penglihatan manusia, sebagai akibatnya, secara tidak langsung bersifat subjektif. Dua buah warna berbeda bisa dianggap warna yang sama dalam perspektif tertentu. Karena itu, diperlukan sebuah metode yang lebih kuantitatif dalam mengurangi nilai subjektivitas tersebut. Metode kuantitatif dalam mengukur keseragaman warna biasanya disebut sebagai metrik warna.

Metrik warna dari model RGB yang paling umum adalah *euclidean color metric*. Metrik ini mengukur jarak antar dua warna berdasarkan jarak euklidean antara dua warna tersebut di dalam sistem koordinat RGB. Misalnya, untuk menghitung jarak antara (R_1, G_1, B_1) dengan (R_2, G_2, B_2) bisa menggunakan rumus berikut.

$$\Delta = \sqrt{|R_1 - R_2|^2 + |G_1 - G_2|^2 + |B_1 - B_2|^2} \quad (3)$$

Penelitian menunjukkan bahwa metrik euklidean pada model RGB di atas kurang baik untuk digunakan dalam membedakan warna. Terdapat aproksimasi yang lebih baik yang dapat digunakan untuk mencari jarak antara dua buah warna dalam model warna RGB. Pendekatan ini disebut sebagai *low-cost approximation* [10].

Secara matematis, jarak antara dua warna, (R_1, G_1, B_1) dengan (R_2, G_2, B_2), diukur menggunakan formula sebagai berikut.

$$\bar{R} = \frac{R_1+R_2}{2}$$
$$\Delta R = R_2 - R_1$$
$$\Delta G = G_2 - G_1$$
$$\Delta B = B_2 - B_1$$

$$\Delta = \sqrt{\left(2+\frac{\bar{R}}{256}\right)\Delta R^2 + 4\Delta G^2 + \left(2+\frac{256-\bar{R}}{256}\right)\Delta B^2} \quad (4)$$

Hasil dari pendekatan ini hampir serupa dengan model warna LUV yang dianggap memiliki tingkat persepsi keseragaman yang cukup baik [10].

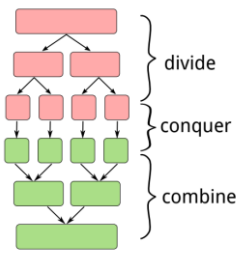
Predikat homogenitas warna dalam sekumpulan warna salah satunya bisa ditentukan menggunakan metrik warna. Misalnya, dimiliki sekumpulan warna dalam model warna RGB, untuk memutuskan sekumpulan warna tersebut seragam atau homogen dapat menggunakan metode pengambangan. Nilai yang diperiksa dalam metode ini bisa beragam. Salah satu nilai yang bisa diperiksa misalnya dengan meninjau jarak setiap warna terhadap warna dominannya dari kumpulan warna tersebut. Jika setiap warna pada suatu kumpulan memiliki jarak yang masih berada pada nilai ambang yang ditentukan, kumpulan tersebut dapat diputuskan seragam. Secara matematis, misalnya akan diputuskan sebuah kumpulan $\mathcal{G}$ dengan n warna dan warna dominan $\bar{C}$, penentuan homogenitas kumpulan warna ini bisa dituliskan sebagai berikut.

$$\boldsymbol{P}(\mathcal{G}): \forall i \in [0,n) \Rightarrow T_{min} \leq \Delta_{C_i,\bar{C}} \leq T_{max} \quad (5)$$

Warna dominan dapat dicari berdasarkan histogram warna dari kumpulan warna yang dimiliki. Namun, warna dominan juga dapat didekati dengan mencari rata-rata seluruh kanal pada kumpulan warna. Adapun pencarian nilai ambang T_{min} dan T_{max}, dapat diperoleh dengan estimasi atau dicari secara kualitatif.

D. Algoritma Divide and Conquer

Algoritma *divide and conquer* merupakan suatu strategi komputasi yang membagi persoalan menjadi beberapa upa-persoalan dan menyelesaikan masing-masing upa-persoalan tersebut sehingga membentuk solusi persoalan semula. Properti yang perlu dipenuhi dari upa-persoalan yang dibangun adalah kemiripannya dengan persoalan awal sehingga kompleksitas permasalahan upa-persoalan akan lebih rendah dari kompleksitas persoalan awal meskipun dengan karakteristik yang sama [9].



Gambar 2. Ilustrasi Algoritma Divide and Conquer [1].

Implementasi algoritma ini biasanya menggunakan pendekatan rekursif karena sifatnya yang mempertahankan karakteristik persoalan awal. Upa-persoalan akan terus dibagi menjadi upa persoalan yang lebih kecil hingga suatu batas tertentu (biasanya berdasarkan ukuran upa-persoalan yang ditinjau).

E. Segmentasi Citra

Berdasarkan [3], misalnya $\mathcal{G}$ adalah wilayah keseluruhan citra digital yang akan disegmentasi. Tinjau partisi $\mathcal{G}$ ke dalam n buah subwilayah baru, $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n$. Segmentasi citra salah satunya bisa diperoleh jika memenuhi properti sebagai berikut.

1) *Partisi mencakup keseluruhan pixel di dalam citra.*

$$\cup_{i=1}^{n} \mathcal{G}_i = \mathcal{G} \quad (6)$$

2) *Setiap elemen dalam partisi merupakan elemen-elemen yang saling bertetangga.*
3) *Semua partisi bersifat disjoint.*

$$\forall i,j \wedge i \neq j \Rightarrow \mathcal{G}_i \cap \mathcal{G}_j = \emptyset \quad (7)$$

4) *Semua partisi memenuhi predikat homogenitas.*

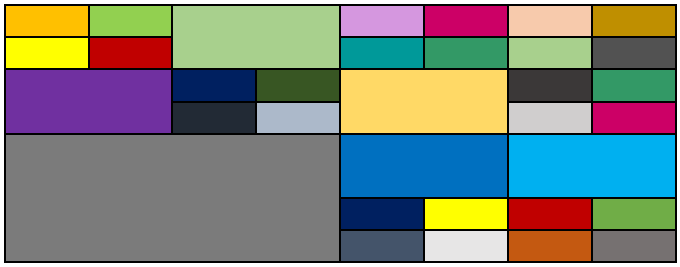
$$\forall i \Rightarrow \mathbf{P}(\mathcal{G}_i) = \text{TRUE} \quad (8)$$

5) *Partisi yang bertetangga gabungannya tidak memenuhi predikat homogenitas.*

$$\forall i,j \wedge \mathbf{Adj}(\mathcal{G}_i, \mathcal{G}_j) \Rightarrow \mathbf{P}(\mathcal{G}_i \cup \mathcal{G}_j) = \text{FALSE} \quad (9)$$

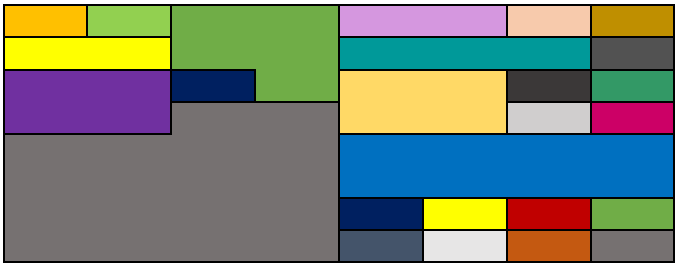
Metode segmentasi ini sering disebut juga sebagai segmentasi berbasis wilayah. Terdapat berbagai macam implementasi segmentasi wilayah, masing-masing dibedakan dengan teknik partisinya atau predikat homogenitas yang didefinisikan. Definisi wilayah yang saling bertetangga juga beragam tetapi biasanya didefinisikan sebagai *pixel-pixel* yang saling bertetangga secara spasial.

Salah satu metode segmentasi wilayah adalah *split and merge*. Basis implementasi metode ini adalah algoritma *divide and conquer*. Citra dipartisi menjadi empat kuadran wilayah sama luas. Masing-masing kuadran menjadi upa-persoalan baru untuk dipartisi atau diselesaikan lebih lanjut. Sebuah wilayah tidak akan dipartisi lebih lanjut jika setiap *pixel* di dalamnya sudah homogen. Proses partisi ini akan membagi citra ke dalam wilayah-wilayah persegi panjang yang masing-masing sudah homogen.



Gambar 3. Ilustrasi *Split* pada Metode *Split and Merge.*

Langkah selanjutnya, setelah citra dipartisi, adalah menggabungkan partisi-partisi bertetangga yang gabungan *pixel-pixel* dari partisi tersebut memenuhi predikat homogen. Citra akan selesai disegmentasi jika sudah tidak ada partisi yang bertetangga yang gabungan partisinya memenuhi predikat homogen.



Gambar 4. Ilustrasi *Merge* pada Metode *Split and Merge*.

III. Implementasi dan Eksperimen

Pada bagian ini akan dibahas implementasi dalam bahasa Phyton metode *split and merge* dalam segmentasi citra. Beberapa pengujian juga akan dianalisis dan dinilai performanya.

A. Desain Implementasi

Segmentasi citra merupakan pemisahan objek dalam citra berdasarkan homogenitas warna. Citra direpresentasikan menjadi matriks yang terdiri atas *pixel-pixel*. Masing-masing pixel memiliki nilai representasi model warna RGB dengan 24 bit *pixel depth*. Definisikan warna dominan $\bar{C}$ dari sembarang partisi $\mathcal{G}_i$ dengan n elemen adalah sebagai berikut.

$$\bar{C} = \left(\frac{\sum_{R \in \text{red}(\mathcal{G}_i)} R}{n}, \frac{\sum_{G \in \text{green}(\mathcal{G}_i)} G}{n}, \frac{\sum_{B \in \text{blue}(\mathcal{G}_i)} B}{n} \right) \quad (10)$$

Dengan kata lain, warna dominan adalah warna yang kanal-kanalnya merupakan rata-rata dari seluruh elemen pada partisi $\mathcal{G}_i$. Adapun metode pencarian jarak dua buah warna mengikuti formula (4) yaitu menggunakan *low-cost approximation*.

Statistik yang ditinjau untuk predikat homogenitas pada implementasi ini didasarkan pada jangkauan dari jarak elemen-elemen pada partisi $\mathcal{G}_i$ terhadap warna dominan. Secara matematis, definisi predikat homogenitas dapat dituliskan sebagai berikut.

$$\mathbf{P}(\mathcal{G}_i) : \max_{C_j \in \mathcal{G}} \Delta_{C_j,\bar{C}} - \min_{C_j \in \mathcal{G}} \Delta_{C_j,\bar{C}} \leq \text{tolerance} \quad (11)$$

Rentang toleransi berada pada interval $[0, 765]$. Batas atas diperoleh dari nilai maksimal yang mungkin dicapai oleh fungsi jarak. Rentang ini dipilih secara kualitatif dengan pengujian berulang untuk mengetahui nilai mana yang paling baik. Jika nilai toleransi terlalu kecil, warna yang seharusnya tergabung akan terpisah. Sebaliknya, jika nilai toleransi terlalu besar, warna yang seharusnya terpisah akan tergabung.

Paradigma pemrograman yang diterapkan pada implementasi ini adalah paradigma pemrograman berorientasi objek.

B. Implementasi Split and Merge

Didefinisikan struktur data *Node* merupakan kelas untuk objek partisi hasil algoritma *Split*. Kelas ini menyimpan informasi elemen-elemen *pixel* yang dimiliki wilayah tersebut serta sebuah predikat untuk menentukan homogenitas wilayah tersebut. Selain itu, kelas ini juga memiliki metode *setter* untuk mempartisi *node* menjadi empat kuadran (sesuai langkah *split*).

```
class Node:
    def __init__(self, img, x, y, topLeft = None, topRight = None, bottomLeft = None, bottomRight =
None):
        self.img = img
        self.x = x
        self.y = y
        if(topLeft == None):
            self.isLeaf = True
        else:
            self.isLeaf = False

        # children element
        self.topLeft = topLeft
        self.topRight = topRight
        self.bottomLeft = bottomLeft
        self.bottomRight = bottomRight

    # predicate indicating this node is homogenous or not
    def isHomogenous(self, tolerance):
        if(self.img.shape[0] == 0 or self.img.shape[1] == 0):
            return True

        rgb = np.array([[[np.floor(np.mean(self.img[:,:,0])),
        np.floor(np.mean(self.img[:,:,1])),
        np.floor(np.mean(self.img[:,:,2]))]]],
        dtype = np.uint8)

        redMean = (self.img[:, :, 0] + rgb[:, :, 0])//2
        rgbDiff = np.square(self.img - rgb)

        distances = rgbDiff[:, :, 0] * (2 + redMean / 256)
        + rgbDiff[:, :, 1] * (4)
        + rgbDiff[:, :, 2] * (2 + (255 - redMean) / 256)

        distances = np.sqrt(distances)
        mx = np.max(distances)
        mn = np.min(distances)
        if(mx - mn < tolerance):
            self.img[:, :] = rgb[0, 0]
            return True
        else:
            return False

    # predicate indicating whether this node adjacent with node `other`
    def isAdj(self, other):
        otherH = other.img.shape[0]
        otherW = other.img.shape[1]
        h = self.img.shape[0]
        w = self.img.shape[1]
        if other.x == self.x + h or other.x + otherH == self.x:
            return self.y - otherW <= other.y <= self.y + w
        if other.y == self.y + w or other.y + otherW == self.y:
            return self.x - otherH <= other.x <= self.x + h
        return False

    # set child (partition into 4 quadrants)
    def setChildren(self, topLeft, topRight, bottomLeft, bottomRight):
        self.topLeft = topLeft
        self.topRight = topRight
        self.bottomLeft = bottomLeft
        self.bottomRight = bottomRight
```

Gambar 5. Implementasi Kelas *Node*.

Kelas *Cluster*, didefinisikan sebagai kelas dari objek yang terdiri dari elemen-elemen partisi yang bertetangga dan memenuhi predikat homogenitas. Sebuah *Cluster* akan diwarnai dengan warna dominannya sesuai dengan formula (10). Untuk meningkatkan performa waktu eksekusi program, masing-masing elemen diakses melalui struktur data *dictionary*, sehingga dalam penambahan elemen *Cluster*, yang perlu dicek hanyalah elemen dengan nilai spasial absis dan ordinatnya memungkinkan untuk bertetangga dengan elemen baru.

Perhatikan bahwa kelas *Cluster* menerapkan predikat homogenitas pada warna dominan saat ini dengan warna calon elemen yang akan menjadi anggota *Cluster*. Hal tersebut berakibat adanya kemungkinan berubahnya jarak elemen yang telah ada di dalam *Cluster* dengan warna dominan baru akibat penambahan elemen. Atas alasan tersebut, toleransi *split* dan

merge tidak perlu seragam (masing-masing dicoba dalam pengujian).

```python
class Cluster:
    def __init__(self, first: Node):
        # dominant color
        self.rgb = np.array([[first.img[0, 0]]], dtype=np.uint8)
        self.list = [first]

        # dictionary by its spacial boundary coordinates
        self.dictbyX = {first.x: [0]}
        self.dictbyY = {first.y: [0]}
        self.dictbyXH = {first.x + first.img.shape[0]: [0]}
        self.dictbyYW = {first.y + first.img.shape[1]: [0]}

    # Check whether there exists element in this cluster that adjacent with the new elm
    def isMember(self, elm: Node):
        found = False
        if elm.x + elm.img.shape[0] in self.dictbyX.keys():
            for member in [self.list[i] for i in self.dictbyX[elm.x+elm.img.shape[0]]]:
                if(elm.isAdj(member)):
                    found = True
                    break

        if not(found) and elm.y + elm.img.shape[1] in self.dictbyY.keys():
            for member in [self.list[i] for i in self.dictbyY[elm.y+elm.img.shape[1]]]:
                if(elm.isAdj(member)):
                    found = True
                    break

        if not(found) and elm.y in self.dictbyYW.keys():
            for member in [self.list[i] for i in self.dictbyYW[elm.y]]:
                if(elm.isAdj(member)):
                    found = True
                    break

        if not(found) and elm.x in self.dictbyXH.keys():
            for member in [self.list[i] for i in self.dictbyXH[elm.x]]:
                if(elm.isAdj(member)):
                    found = True
                    break

        if not(found):
            return np.finfo(float).max

        redMean = (int(elm.img[0, 0, 0]) + int(self.rgb[0, 0, 0]))//2
        diffr = int(elm.img[0, 0, 0]) - int(self.rgb[0, 0, 0])
        diffg = int(elm.img[0, 0, 1]) - int(self.rgb[0, 0, 1])
        diffb = int(elm.img[0, 0, 2]) - int(self.rgb[0, 0, 2])

        return np.sqrt((((512+redMean)*diffr*diffr)>>8)
                       + 4*diffg*diffg + (((767-redMean)*diffb*diffb)>>8))


    # if new elm is "worthy" to be in the cluster, i.e. the element and the cluster accept
    # the homogeneity predicate
    def appendElm(self, elm: Node):
        if elm.x in self.dictbyX.keys():
            self.dictbyX[elm.x].append(len(self.list))
        else:
            self.dictbyX[elm.x] = [len(self.list)]

        if elm.y in self.dictbyY.keys():
            self.dictbyY[elm.y].append(len(self.list))
        else:
            self.dictbyY[elm.y] = [len(self.list)]

        if elm.x + elm.img.shape[0] in self.dictbyXH.keys():
            self.dictbyXH[elm.x+elm.img.shape[0]].append(len(self.list))
        else:
            self.dictbyXH[elm.x+elm.img.shape[0]] = [len(self.list)]

        if elm.y + elm.img.shape[1] in self.dictbyYW.keys():
            self.dictbyYW[elm.y+elm.img.shape[1]].append(len(self.list))
        else:
            self.dictbyYW[elm.y+elm.img.shape[1]] = [len(self.list)]

        for i in range(3):
            self.rgb[0, 0, i] = (int(self.rgb[0, 0, i])*len(self.list)
                                 + int(elm.img[0, 0, i]))//(len(self.list)+1)
        self.list.append(elm)

    # colorize all element in the cluster with its dominant color
    def colorizeAll(self):
        for elm in self.list:
            elm.img[:, :] = self.rgb[0, 0]
```

Gambar 6. Implementasi Kelas *Cluster*.

Kontroler dari kedua kelas di atas diimplementasikan di dalam kelas *Tree*. Kelas ini melakukan proses *split and merge* lalu menampilkan hasilnya.

Perhatikan bahwa citra dipartisi ke dalam kuadran yang tidak harus selalu sama luasnya tetapi diusahakan untuk sama luas. Partisi akan berhenti jika jumlah *pixel* wilayah yang bersangkutan sudah berukuran satu atau wilayah partisi sudah memiliki properti homogenitas.

Proses *merge* diimplementasikan dengan mencari *Cluster* yang memiliki elemen yang bertetangga dengan elemen baru serta memiliki properti homogenitas. Setiap *Cluster* pada akhir *merge* diwarnai sesuai dengan warna dominannya. Keluaran yang dihasilkan adalah citra hasil proses *split* dan *citra* hasil proses *merge*.

```python
class Tree:
    def __init__(self, img, splitTolerance, mergeTolerance):
        tmp = copy.deepcopy(img)

        self.root = Node(tmp, 0, 0)
        self.nodeList = []
        self.clusterList = []

        self.constructTree(self.root, splitTolerance)
        self.splittedImg = copy.deepcopy(tmp)

        self.mergeTree(mergeTolerance)
        self.mergedImg = copy.deepcopy(tmp)

    # split the image
    def constructTree(self, parent: Node, tolerance):
        if(not(parent.isHomogenous(tolerance))):
            halfHeight = parent.img.shape[0]//2
            halfWidth = parent.img.shape[1]//2

            x = parent.x
            y = parent.y


            topLeft = Node(parent.img[:halfHeight, :halfWidth], x, y)
            topRight = Node(parent.img[:halfHeight, halfWidth:], x, y+halfWidth)
            bottomLeft = Node(parent.img[halfHeight:, :halfWidth], x+halfHeight, y)
            bottomRight = Node(parent.img[halfHeight:, halfWidth:], x+halfHeight, y+halfWidth)

            parent.setChildren(topLeft, topRight, bottomLeft, bottomRight)

            self.constructTree(topLeft, tolerance)
            self.constructTree(topRight, tolerance)
            self.constructTree(bottomLeft, tolerance)
            self.constructTree(bottomRight, tolerance)
        elif(parent.img.shape[0] !=0 and parent.img.shape[1] != 0):
            self.nodeList.append(parent)

    # merge the image
    def mergeTree(self, tolerance):
        for elm in self.nodeList:
            if(len(self.clusterList)==0):
                self.clusterList.append(Cluster(elm))
            else:
                found = False
                for group in self.clusterList:
                    if group.isMember(elm) < tolerance:
                        group.appendElm(elm)
                        found = True
                if not(found):
                    self.clusterList.append(Cluster(elm))
        self.colorize()

    def colorize(self):
        for cluster in self.clusterList:
            cluster.colorizeAll()

    # show
    def showSplittedImg(self):
        plt.imshow(self.splittedImg)

    # show merged image result
    def showMergedImg(self):
        plt.imshow(self.mergedImg)
```

Gambar 7. Implementasi Kelas Tree.

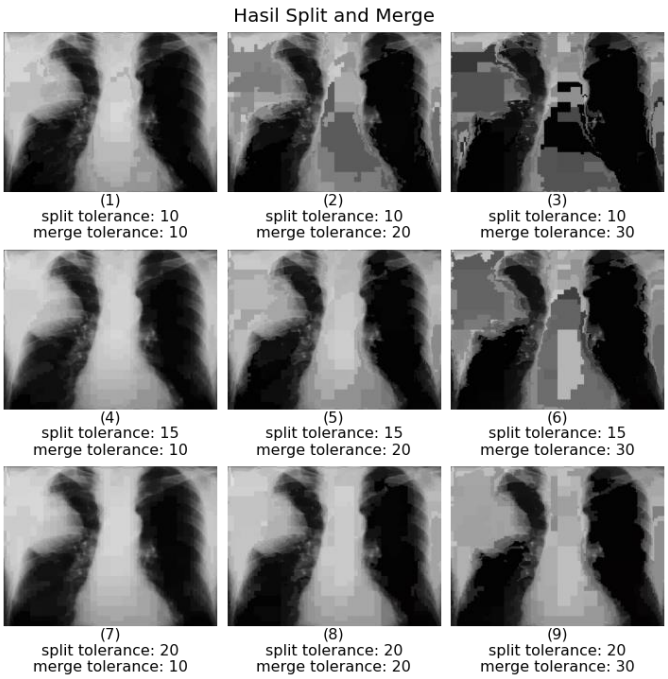
C. *Pengujian*

Pada setiap pengujian di bawah ini, toleransi *split* yang akan diuji adalah 10, 15, dan 20, sedangkan toleransi *merge* yang akan diuji adalah 10, 20, dan 30.

Pengujian pertama akan dilakukan pada citra dengan resolusi $636 \, x \, 476$, berikut adalah tampilan gambar awal sebelum diproses.

Gambar 8. Kasus Uji 1 Hasil *Rontgen* Thorax Manusia [2].

Gambar tersebut merupakan hasil *rontgen* bagian dada manusia yang memiliki sel kanker (di sebelah kiri gambar). Berikut adalah hasil *split and merge* citra uji di atas.



Gambar 9. Hasil *Split and Merge* Pengujian 1.

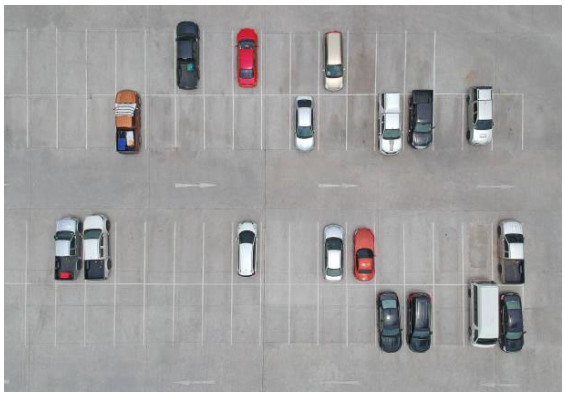
Adapun pengurangan warna citra dari berbagai variasi toleransi setelah dilakukan *merge* adalah sebagai berikut.

```
Persentase pengurangan warna:
Gambar 1: 73.64802248064333%
Gambar 2: 89.08253863860573%
Gambar 3: 95.22884218707961%
Gambar 4: 63.76842901117892%
Gambar 5: 84.00388831884214%
Gambar 6: 93.41145973969866%
Gambar 7: 54.0572408618287%
Gambar 8: 76.51409583020688%
Gambar 9: 89.69878872333584%
```

Gambar 10. Persentase Pengurangan Warna Hasil *Merge* Pengujian 1.

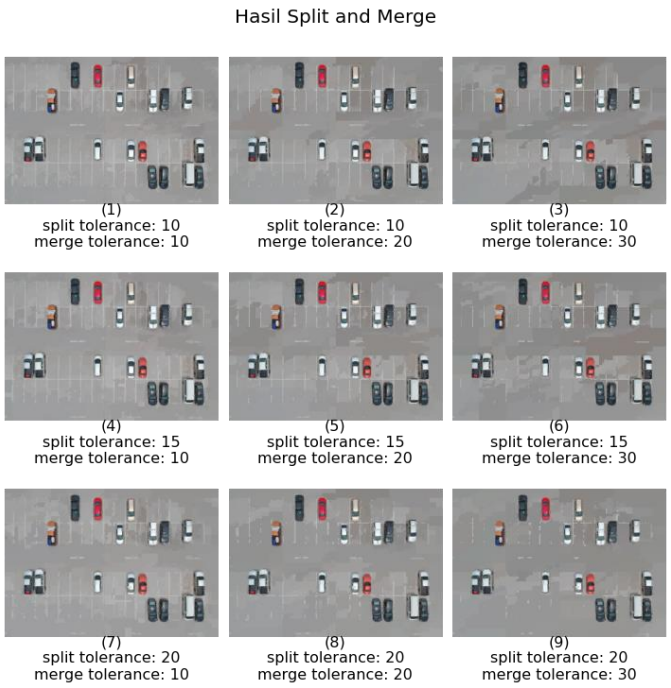
Perhatikan bahwa semakin besar toleransi *merge*, warna yang tergabung juga semakin banyak (pengurangan warnanya menjadi lebih banyak). Secara kualitatif, jika dinilai berdasarkan persentase pengurangan warna dan kehalusan gambar, gambar yang lebih baik segmentasinya adalah gambar 1, 5, dan 9. Hal ini terlihat bukan hanya dari persentase pengurangan warna yang tidak terlalu kecil tetapi juga gambar hasilnya masih mempertahankan struktur gambar awal (bentuk aslinya masih terlihat).

Pengujian kedua dilakukan menggunakan citra dengan resolusi $494 \; x \; 349$, berikut adalah tampilan awal citra.



Gambar 11. Kasus Uji 2 Tampilan Atas Sebuah Parkir Mobil [11].

Citra di atas merupakan tampilan atas sebuah tempat parkir mobil dengan beberapa mobil yang sedang parkir. Hasil segmentasi dari citra di atas diperoleh sebagai berikut.



Gambar 12. Hasil *Split and Merge* Pengujian 2.

Hasil di atas masing-masing memiliki persentase pengurangan warna sebagai berikut.

```
Persentase pengurangan warna:
Gambar 1: 61.43541262436527%
Gambar 2: 77.86401354084961%
Gambar 3: 85.92247037939148%
Gambar 4: 54.1401731508424%
Gambar 5: 72.19887200488559%
Gambar 6: 81.66469087904588%
Gambar 7: 51.760139204925714%
Gambar 8: 70.11778878329541%
Gambar 9: 79.9156739392317%
```

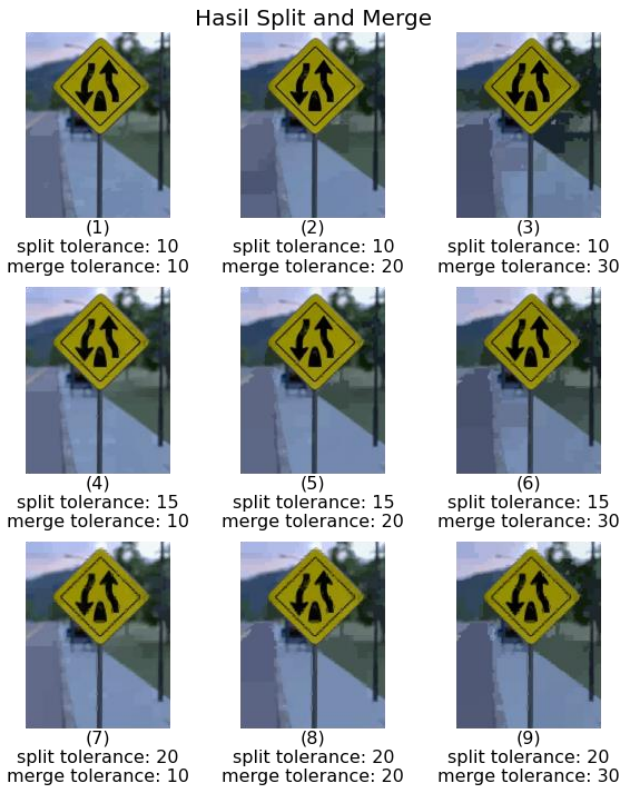
Gambar 13. Persentase Pengurangan Warna Hasil *Merge* Pengujian 2.

Tampilan dari gambar 1 hingga gambar 9 menunjukkan hasil yang cukup baik karena masing-masing mempertahankan bentuk gambar awalnya. Pengurangan warna paling besar terjadi pada gambar ketiga. Meskipun angkanya mencapai 85%, bentuk objek-objek masih terlihat dengan jelas, artinya, pemisahan objek pada gambar 3 memiliki performanya relatif baik. Meskipun demikian, waktu yang diperlukan untuk segmentasi kesembilan variasi toleransi di atas mencapai 55.2 menit, cukup jauh dibandingkan citra sebelumnya yang hanya membutuhkan 17.4 menit.

Pada pengujian ketiga, citra yang digunakan memiliki resolusi $232\ x\ 300$. Citra ini merupakan sebuah potret rambu jalanan. Berikut adalah tampilan awal citra.

Gambar 14. Kasus Uji 3 Potret Rambu Jalan [5].

Menggunakan variasi toleransi yang sama dengan kasus uji sebelumnya, diperoleh hasil segmentasi citra di atas adalah sebagai berikut.



Gambar 15. Hasil *Split and Merge* Pengujian 3.

Pengurangan warna yang terjadi pada masing-masing variasi toleransi adalah sebagai berikut.

```
Persentase pengurangan warna:
Gambar 1: 43.522363673497175%
Gambar 2: 69.62352780992579%
Gambar 3: 81.5167812648921%
Gambar 4: 38.02786478021385%
Gambar 5: 63.527378766605466%
Gambar 6: 76.44454044713252%
Gambar 7: 34.92307692307692%
Gambar 8: 59.69230769230769%
Gambar 9: 73.07692307692308%
```

Gambar 16. Persentase Pengurangan Warna Hasil *Merge* Pengujian 3.

Dibandingkan kasus uji sebelumnya, terlihat bahwa pengurangan warna hasil *merge* tidak terlalu signifikan. Hal ini juga didukung dengan hasil segmentasi yang masih mempertahankan gambar aslinya (beberapa hasil segmentasi seharusnya masih memiliki bagian yang masih bisa digabung). Gambar ketiga secara persentase merupakan citra dengan kualitas segmentasi yang cukup baik, meskipun bagian jalan terlihat ada fragmentasi yang tidak diperlukan.

IV. Kesimpulan dan Saran

Berdasarkan implementasi dan pengujian di bagian sebelumnya, dapat disimpulkan bahwa *split and merge* merupakan algoritma yang cukup sederhana, tetapi memiliki daya guna yang relatif baik untuk melakukan segmentasi citra. Toleransi dan kasus uji yang dipilih penulis jelas tidak merepresentasikan populasi statistik sebenarnya bagaimana mayoritas citra dalam kehidupan nyata. Setiap citra memiliki rentang dan variansi warnanya masing-masing sehingga pemilihan toleransi untuk satu citra dengan citra lainnya akan bervariasi dampaknya.

Semakin besar toleransi, segmentasi akan lebih tidak sensitif terhadap kontras antar dua warna sehingga citra akan cenderung membentuk wilayah-wilayah besar yang tergabung. Sebaliknya, semakin kecil toleransi, segmentasi akan lebih sensitif terhadap perbedaan antara dua warna sehingga terjadi pemisahan objek yang lebih banyak. Namun, ukuran besar-kecilnya toleransi sangat bergantung dengan kondisi citra yang diuji. Semakin kecil rentang warna dari sebuah citra (warna global citra bersifat *uniform*), semakin besar kemungkinan objek untuk bergabung dengan objek lain ketika toleransinya bertambah.

Performa secara kebutuhan waktu untuk *split and merge* tidak terlalu baik. Proses *merge* memiliki kompleksitas terburuk $O(\text{jumlah node}^2)$. Resolusi citra dengan warna yang cukup beragam akan mengakibatkan waktu algoritma bertambah secara signifikan.

Segmentasi citra ini masih bisa ditingkatkan performanya dalam banyak hal. Pemilihan predikat homogenitas dan penentuan warna dominan bisa meningkatkan performa segmentasi. Warna dominan bisa diperoleh dengan menggunakan *clustering* dengan meninjau histogram warna. Predikat homogenitas dengan toleransi yang diestimasi menggunakan prinsip-prinsip statistik memungkinkan performa yang lebih baik. Adapun untuk optimasi waktu eksekusi, proses *merge* bisa menggunakan algoritma DFS atau BFS dalam mencari *connected component* dari simpul-simpul yang terbentuk pada proses *split*.

Link Repositori Github

Repositori *source code* implementasi dapat diakses melalui firizky29/image-segmentation-dnc-python (github.com).

Link Menuju Video Youtube

Video penjelasan makalah ini dapat diakses melalui (2807) Aplikasi Algoritma Divide and Conquer pada Segmentasi Citra - YouTube.

Ucapan Terima Kasih

Rasa syukur dan ucapan terima kasih penulis sampaikan kepada Tuhan Yang Maha Esa sebab berkat rahmat-Nya, penulis dapat menyelesaikan makalah ini dengan baik. Terima kasih juga penulis sampaikan kepada Ibu Dr. Nur Ulfa Maulidevi S.T., M.Sc. dan Ibu Masayu Leylia Khodra, S.T., M.T. selaku dosen pengampu mata kuliah Strategi Algoritma tahun ajaran 2022/2023 atas bimbingan dan ilmu yang telah diajarkan selama satu semester ini. Tak lupa, penulis mengucapkan terima kasih kepada orang tua, keluarga, dan teman-teman penulis atas dukungan dan do'anya.

Referensi

[1] Aji, I. F. & Gozali, W., Pemrograman Kompetitif Dasar, CV. Nulisbuku Jendela Dunia. Versi 1.9.

[2] Barbetakis, Nikolaos & Samanidis, Georgios & Paliouras, Dimitrios & Mavroudi, Eleni & Boukovinas, Ioannis & Tsilikas, Christodoulos. (2008). Distant forearm muscle metastasis from squamous cell lung carcinoma. Tüberküloz ve toraks. 56. 109-12.

[3] Gonzalez, R. C. & Woods, R. E. (2008), Digital image processing, Prentice Hall, Upper Saddle River, N.J.

[4] https://cp-algorithms.com/data_structures/disjoint_set_union.html diakses pada 20 Mei 2022 pukul 13.00 WIB.

[5] https://driving-tests.org/road-signs/ diakses pada 21 Mei pukul 13.15 WIB.

[6] https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2020-2021/01-Pengantar-Pengolahan-Citra-Bag1-2021.pdf diakses pada 8 Mei 2022 pukul 21.00 WIB.

[7] https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2021-2022/16-Warna-bagian1-2022.pdf diakses pada 19 Mei 2022 pukul 22.00 WIB.

[8] https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2019-2020/17-Segmentasi-Citra.pdf diakses pada 8 Mei 2022 pukul 20.00 WIB.

[9] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf diakses pada 19 Mei 2022 pukul 22.00 WIB.

[10] https://www.compuphase.com/cmetric.htm diakses pada 8 Mei 2022 pukul 20.00 WIB.

[11] https://www.istockphoto.com/id/foto-foto/car-park-from-above diakses pada 20 Mei 2022 pukul 18.00 WIB.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2022

Firizky Ardiansyah
13520095