Segmentasi Citra Biometrik dengan Harmony Search Algorithm dan Improved Harmony Search Algorithm

ERWIN
MUHAMMAD FACHRURROZI

UNSRI PRESS

Segmentasi Citra Biometrik dengan Harmony Search Algorithm dan Improved Harmony Search Algorithm

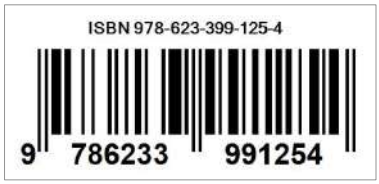
Erwin

Muhammad Fachrurrozi

UPT. Penerbit dan Percetakan
Universitas Sriwijaya 2023
Kampus Unsri Palembang
Jalan Srijaya Negara, Bukit Besar Palembang 30139
Telp. 0711-360969
email : unsri.press@yahoo.com, penerbitunsri@gmail.com
website : www.unsri.unsripress.ac.id

Anggota APPTI No. 005.140.1.6.2021
Anggota IKAPI No. 001/SMS/96

72 halaman : 15.5 x 21 cm



PRAKATA

Puji syukur kehadirat Allah SWT atas limpahan rahmat dan karunianya sehingga Buku referensi dengan judul “Segmentasi Citra Biometrik dengan Harmony Search Algorithm dan Improved Harmony Search Algorithm” dapat diselesaikan. Pada buku ini terdapat pengetahuan dalam pemrosesan citra biometric khususnya citra lidah serta terdapat implementasi program menggunakan Bahasa Java.

Banyak pihak yang telah membantu dalam penyelesaian buku ini. Untuk itu penulis mengucapkan rasa terima kasih kepada RIFDAH NABILAH RAMADHANI dan YENITA selaku mahasiswa yang terlibat dalam penelitian yang kami lakukan dan anggota Forum Riset Grup *Digital Image Processing*.

Kami menyadari masih terdapat kekurangan dalam buku ini. Untuk itu kritik dan saran terhadap penyempurnaan buku ini sangat diharapkan. Semoga buku ini dapat memberi manfaat bagi semua pihak yang membutuhkan.

Wassalam

Palembang, Mei 2023

Penulis

DAFTAR ISI

BAB 1. PENDAHULUAN

1.1. Segmentasi

Segmentasi adalah salah satu bagian terpenting dalam pengolahan citra yang berusaha mengidentifikasi apakah intensitas piksel sesuai dengan kelas yang telah ditetapkan[1]. Segmentasi citra merupakan teknik yang paling dasar. Tujuan segmentasi adalah membagi citra menjadi beberapa area yang berarti sesuai dengan beberapa karakteristik, tingkat abu-abu, warna, dan sebagainya. Karakteristiknya akan sama atau sangat mirip di bagian dalam area tersebut dan berbeda di daerah yang berbeda[2]. Segmentasi citra terdiri dari pembagian kembali ke sejumlah daerah non-pengupasan, homogen, yang masing-masing terhubung secara spasial dan berbeda dari daerah tetangga dalam beberapa properti yang bermakna. Proses segmentasi adalah proses mempartisi citra menjadi beberapa daerah yang tidak berpotongan sehingga masing-masing daerah adalah homogen dan penyatuan tidak ada dua daerah yang berdekatan bersifat homogen[3]. Segmentasi citra memainkan peran penting dalam banyak aplikasi citra medis, sistem keamanan dan pengawasan, citra satelit, aplikasi meteorologi, dan lain-lain. Segmentasi[4][5] adalah metode di mana input adalah citra, namun keluarannya adalah atribut yang diekstraksi. Dari citra itu Segmentasi[6] membagi citra menjadi daerah penyusun atau objeknya. Tingkat dimana pembagian dilakukan tergantung pada masalah yang dipecahkan[7]

1.2. Harmony Search Algorithm(HSA)

Algoritma *harmony search* ditemukan oleh Zong Woo Geem pada tahun 2005. *Harmony Search* merupakan salah satu algoritma mataheuristic yang digunakan untuk melakukan optimalisasi. Sebelumnya penulis akan menjelaskan sedikit mengenai metaheuristic. Metaheuristic merupakan metode optimasi berbasis pendekatan. Optimasi memiliki dua metode yakni metode eksak dan metode

pendekatan. Dikatakan pendekatan karena hasilnya hanya berdasarkan kualitatif tidak seperti metode eksa yang hasilnya bersifat kuantitatif. Metaheuristic merupakan metode pendekatan yang bersifat *independent*, tidak seperti heuristik yang bersifat *dependent*. Bersifat *independent* artinya metode optimasi berbasis pendekatan itu tidak bergantung pada jenis permasalahan, sebaliknya metode heuristik tergantung permasalahan contohnya metode *nearest neighbour*.

Terdapat 5 parameter penting dalam algoritma *harmony search*. Berikut penjelasan kelima parameter tersebut:

1. *Harmony Memory Size* (HMS) merupakan jumlah solusi simultan vector dalam memori harmoni. Nilainya bervariasi dari 1 sampai 2000.
2. *Harmony Memory Considering Rate* (HMCR) merupakan tingkat atau persentase nilai *Harmony Search Algorithm* (HSA) yang dipilih dari memori harmoni. Nilainya bervariasi dari 0,7 sampai 0,99.
3. *Pitch Adjustment Rate* (PAR) merupakan penunjuk tingkat atau persentase pemilihan nilai yang berdekatan. Nilainya bervariasi dari 0,01 sampai 0,99.
4. *Bandwith (BW)* merupakan pengendali pencarian nilai dalam harmony memori. Nilainya bervariasi 0.001 sampai 1.
5. *NI (Number Iteration)* merupakan banyaknya perulangan yang dilakukan selama pemrosesan.

Terdapat penelitian terkait algoritma *harmony search* diantaranya :

Pada penelitian pencarian harmoni lokal terbaik dengan sub populasi dinamis menggunakan algoritma *harmony search* disajikan algoritma pencarian harmoni lokal terbaik dengan subpopulasi dinamis (DLHS) untuk memecahkan masalah optimasi kontinu yang terbatas[8]. Tidak seperti algoritma pencarian harmonis yang ada, algoritma DLHS membagi keseluruhan memori harmonis (HM) ke dalam banyak sub-HM berukuran kecil dan evolusi dilakukan di setiap

sub-HM secara independen. Untuk menjaga keragaman populasi dan untuk meningkatkan keakuratan solusi akhir, pertukaran informasi antar sub-HMTS dicapai dengan menggunakan jadwal regrouping berkala. Selanjutnya, skema improvisasi harmoni baru digunakan untuk memperoleh manfaat dari informasi yang baik yang tertangkap dalam vektor harmoni lokal terbaik. Selain itu, strategi adaptif dikembangkan untuk menyesuaikan parameter yang sesuai dengan masalah tertentu atau fase proses pencarian tertentu. Simulasi dan perbandingan komputasi yang ekstensif dilakukan dengan menggunakan seperangkat 16 masalah patokan dari literatur. Hasil komputasi menunjukkan bahwa keseluruhan algoritma DLHS yang diusulkan lebih efektif atau paling tidak kompetitif dalam menemukan solusi mendekati optimal dibandingkan dengan varian pencarian harmonis mutakhir.

Pada penelitian algoritma *harmony search* menggunakan peta *chaotic* untuk adaptasi parameter yang bertujuan memperbaiki karakteristik konvergensi dan mencegah *harmony search* terjebak pada solusi lokal[9]. Hal ini telah dilakukan dengan menggunakan generator bilangan *chaotic* setiap kali dibutuhkan bilangan acak oleh algoritma *harmony search*. Tujuh *chaotic* baru dari algoritma *harmony search* diusulkan dan peta *chaotic* yang berbeda telah dianalisis dalam fungsi benchmark. Telah terdeteksi bahwa hasil kopling muncul di daerah yang berbeda, seperti *harmony search* dan dinamika kompleks, dapat memperbaiki kualitas daalam beberapa masalah optimasi. Penelitian ini meunjukkan jga bahwa, beberapa metode yang diusulkan sedikit meningkatkan kualitas solusi, yang dalam beberapa kasus memperbaiki kemampuan pencarian global dengan melepaskan solusi lokal.

Penelitian penelitian algoritma *harmony search* untuk Nurse Scheduling Problem (NSP)[10]. Masalah penjadwalan perawat adalah sebuah tugas memberikan shift ke perawat untuk tugas yang harus dilakukan. Kesulitan dalam menangani masalah ini adalah karena tingginya jumlah kendala yang harus

selesaikan. Oleh karena itu, penelitian ini mengusulkan adaptasi dari HSA. Kinerja HSA dievaluasi menggunakan dataset yang ditetapkan oleh International Nurse Rostering Competition 2010 (INRC2010). Parameter HSA yang digunakan adalah: HMS = 100, HMCR = 0,99, PAR = 0,01, dan NI antara 100000 dan 300000, dimana proses pencarian berhenti setelah 5000 iterasi tanpa perbaikan. Hasilnya menunjukkan bahwa HSA dapat digunakan untuk memecahkan NSP.

Pada penelitian algoritma *harmony search* yang diterapkan pada masalah optimasi ekologis (MCSP) dimana jumlah spesies yang diawetkan di suatu daerah akan memaksimalkan sementara jumlah bidang yang dipertimbangkan dibatasi[11]. Harmony search dalam penelitian ini dimodifikasi dari struktur asli agar bisa diaplikasikan ke MCSP: Karena nilai kandidat masing-masing variabel adalah 0 atau 1, harmony search tidak memiliki operasi penyesuaian nada. Karena setiap vektor solusi jarang, kesempatan untuk dipilih terbatas (kurang dari 50%) berdasarkan jumlah spesies di setiap bidang paket. Untuk memulai dengan lingkungan yang baik, vektor solusi pada harmony search awal dipilih dari generasi n. Untuk keragaman vektor solusi di harmony search, vektor yang memiliki spesies baru secara deterministik termasuk dalam harmony search. harmony search diterapkan pada masalah dunia nyata (negara bagian Oregon) yang memiliki 426 spesies dan 441 paket. Dalam 24 kasus dengan jumlah paket yang berbeda, harmony search menemukan solusi optimum global dalam 15 kasus dan solusi optimal mendekati 9 kasus. Bila dibandingkan dengan algoritma meta-heuristik lainnya, algoritma harmony search menemukan solusi yang lebih baik daripada algoritma SA pada 14 kasus sedangkan yang pertama menemukan solusi yang buruk sekali saja. Kelebihan lain dari algoritma harmony search adalah kenyataan bahwa ia menyarankan banyak solusi alternatif karena secara bersamaan menangani beberapa vektor solusi.

Pada penelitian algoritma *harmony search* yang diterapkan ke masalah routing bus sekolah, parameter masalah dan algoritma ditentukan: jumlah alat musik (jumlah node permintaan) = 10. Rentang pitch masing-masing instrumen (rentang nilai dari setiap variabel keputusan) = {bus 1, bus 2, bus 3, bus 4}. HMS = 10 s.d. 100 , HMCR = 0,3 s.d. 0,95, PAR = 0,01 s.d. 0,05. Jumlah iterasi = 1.000 improvisasi. Selanjutnya, harmoni (vektor solusi) secara acak dihasilkan dari kisaran sebanyak HMS. Setelah itu, harmoni baru diimprovisasi berdasarkan tiga aturan (seleksi acak, pertimbangan HM, dan penyesuaian nada). Varian harmoni xbaru kemudian dimasukkan ke fungsi objektif untuk mendapatkan total biaya yang terdiri dari biaya bus tetap, biaya perjalanan bus, dan dua biaya penalti. Tujuan *harmony search* yang diusulkan untuk *routing* bus sekolah adalah untuk meminimalkan total biaya fungsi multi fungsi yang terdiri dari biaya operasi bus, waktu perjalanan bus, dan denda terkait dengan kapasitas bus dan pelanggaran. harmony search dapat menemukan optimamasi global dalam evaluasi fungsi yang jauh lebih sedikit dibandingkan jumlah enumerasi. *Harmony Search* juga menemukan solusi yang lebih baik daripada GA dalam hal jumlah pencapaian optimum global, biaya rata-rata dari beberapa putaran, dan waktu komputasi. Dari hasil ini, algoritma harmony search memiliki potensi untuk diterapkan pada bidang teknik lalu lintas[12].

1.3. Improved Harmony Search Algorithm(IHSA)

IHSA merupakan sebuah metode yang diperkenalkan oleh Mahdavi, Fesanghary, dan Damangir pada tahun 2006, dimana metode tersebut merupakan perkembangan dari metode sebelumnya yakni metode *Harmony Search Algorithm* (HSA). HSA telah berhasil digunakan dalam beberapa referensi penelitian sebelumnya untuk mengatasi masalah segmentasi citra. Alia et al menggunakan *dynamic clustering* berbasis HSA dan hasil eksperimennya menunjukkan kualitas solusi yang lebih baik dibandingkan algoritma *dynamic clustering* sebelumnya[13]. Dalam penelitian lain, HSA digunakan untuk segmentasi citra dan menggabungkannya dengan algorima *Fuzzy*. Penggabungan dua metode ini secara signifikan meningkatkan kualitas dan kecepatan dari konvergensi metode tersebut[14].

1.4. Referensi

[1] D. Oliva, E. Cuevas, G. Pajares, D. Zaldivar, and M. Perez-cisneros, "Multilevel Thresholding Segmentation Based on Harmony Search Optimization," *J. Appl. Math.*, vol. 2013, 2013.

[2] H. Atmaca, M. Bulut, and D. Demir, "HISTOGRAM BASED FUZZY KOHONEN CLUSTERING NETWORK," pp. 1–4, 1996.

[3] N. Jabbar, S. I. Ahson, and M. Mehrotra, "Fuzzy Kohonen Clustering Network for Color Image Segmentation," vol. 3, pp. 254–257, 2011.

[4] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, S. Member, and T. Moriarty, "A Modified Fuzzy C-Means Algorithm for Bias Field Estimation and Segmentation of MRI Data," vol. 21, no. 3, pp. 193–199, 2002.

[5] S. Murugavalli and V. Rajamani, "A HIGH SPEED PARALLEL FUZZY C-MEAN ALGORITHM FOR BRAIN," no. 6, 2006.

[6] R. C. Gonzales and R. E. Wood, *Digital Image Processing*. Prentice Hall, 2005.

[7] D. V. Fernandes, T. Thomas, N. Joseph, and J. Joseph, "Image Segmentation using Fuzzy - Kohonen Algorithm," no. Icmlc, pp. 176–179, 2011.

[8] Q.-K. Pan, P. N. Suganthan, J. J. Liang, and M. F. Tasgetiren, "A local-best harmony search algorithm with dynamic subpopulations," *Eng. Optim.*, vol. 42, no. 2, pp. 101–

117, 2010.

[9] B. Alatas, "Chaotic harmony search algorithms," *Appl. Math. Comput.*, 2010.

[10] M. A. Awadallah, A. T. Khader, M. A. Al-Betar, and A. L. aro Bolaji, "Nurse scheduling using Harmony Search," in *Proceedings - 2011 6th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2011*, 2011.

[11] Z. W. O. O. Geem and J. C. Williams, "Ecological Optimization Using Harmony Search," *Soft Comput.*, 2008.

[12] Z. W. Geem, K. S. Lee, and Y. Park, "Application of Harmony Search to Vehicle Routing," *Am. J. Appl. Sci.*, 2005.

[13] O. M. Alia, R. Mandava, D. Ramachandram, and M. E. Aziz, "Dynamic fuzzy clustering using Harmony Search with application to image segmentation," *2009 IEEE Int. Symp. Signal Process. Inf. Technol.*, pp. 538–543, 2009.

[14] O. M. Alia *et al.*, "Color tongue image segmentation using Fuzzy Kohonen Networks and Genetic Algorithm," *Proc. SPIE---The Int. Soc. Opt. Eng.*, vol. 3962, no. 1, pp. 176–179, 2010.

Bab 2. Data dan Metode

2.1. Data Citra Biometrik

Data citra biometric yang digunakan adalah dataset citra lidah. Data citra lidah yang digunakan didapatkan dari *website* Biometric Reasearch Center, Hongkong. Data citra lidah yang diberikan berupa database sampel citra lidah. Terdapat 12 sampel dengan format data BMP, citra lidah yang diberikan oleh *website* Biometric Reasearch Center. Ukuran citra bervariasi atau dengan kata lain ukuran citra tidak sama satu dengan yang lainnya. Sampel citra lidah yang diberikan hanya menampilkan rongga mulut dengan kondisi lidah bersih. Untuk kepentingan representasi, hanya 5 citra uji yang ditampilkan dan dapat dilihat pada tabel 1.

TABEL 1

Citra Biometric

No	Nama File	Citra
1	Tongue1	
2	Tongue2	

3	Tongue3	
4	Tongue4	
5	Tongue5	

2.2. Harmony Search Algorithm (HSA)

Secara umum algoritma harmony search algorithm terdiri dari tiga fase utama, yaitu inisialisasi harmony search, improvisasi vektor harmoni baru, dan pembaharuan harmony memory dengan kandidat yang lebih baik. Berikut merupakan penjelasan lebih lanjut mengenai ketiga fase tersebut:

1. Inisialisasi Parameter.

   Secara umum, masalah optimasi global dapat disimpulkan pada persamaan (1) dan (2).

$$f(\mathbf{x}), \qquad \mathbf{x} = (x(1), x(2), \ldots, x(n)) \in \mathbf{R}^n, \qquad (1)$$

$x\ (j) \in [l\ (j), u\ (j)] \quad j = 1,2, \ldots, n,$ (2)

Keterangan :

$f\ (\mathbf{x})$ = fungsi tujuan

x = (x (1), x (2) ,. .., $x\ (n)$) adalah kumpulan variabel desain

n = jumlah variabel desain

$l\ (j)$ dan $u\ (j)$ = batas bawah dan atas masing-masing variabel yang dapat dirubah $x\ (j)$

Terdapat lima parameter dalam HSA, yakni *harmony memory* (HMS), yaitu jumlah vektor solusi yang tersimpan dalam *harmony memory* (HM),*harmony-memory consideration rate* (HMCR), *pitch adjustingrate* (PAR), *distancebandwidth* (BW), dan *number of iteration* (NI). Kinerja dari HSA sangat dipengaruhi oleh nilai-nilai yang diberikan pada parameter tersebut.

2. Inisialisasi *Harmony Memory* (HM)

Pada tahap ini, komponen vektor awal dalam HM yaitu vektor HMS di berikan.Misalkan $\mathbf{x}i$ = {xi (1), xi (2), ..., $xi\ (n)$} mewakili vektor harmoni yang dihasilkan secara acak: $xi\ (j) = l\ (j) + (u\ (j) - l\ (j)) \cdot rand$ (0,1) untuk j = 1,2, ..., n dan i = 1,2, ..., HMS, dimana *rand* (0,1) adalah bilangan acak seragam antara 0 dan 1, batas atas dan bawah ruang pencarian ditentukan oleh $l\ (j)$ dan $u\ (j)$. Kemudian, matriks HM diisi dengan vektor harmoni HMS seperti pada persamaan (3) berikut

$$\mathrm{HM}=\begin{bmatrix} x_1 \\ x_2 \\ : \\ x_{HMS} \end{bmatrix} \quad (3)$$

3. Improvisasi Harmoni Memori

Pada tahap ini, sebuah vektor harmoni baru x baru dibangun dengan menerapkan tiga operator berikut: pertimbangan memori, *reinitialization* acak, dan penyesuaian nada. Membangkitkan harmoni baru dikenal sebagai "improvisasi." Dalam tahap pertimbangan memori, nilai variabel keputusan pertama x_{baru} (1) untuk vektor baru dipilih secara acak dari nilai mana pun yang sudah ada pada HM saat ini, yaitu, Dari himpunan {x1 (1), x2 (1), ..., xHMS (1)}. Untuk operasi ini, nomor acak seragam $r1$ dihasilkan dalam kisaran [0,1]. Jika $r1$ kurang dari HMCR, variabel keputusan x_{baru} (1) dihasilkan melalui pertimbangan memori; Jika tidak, x_{new} (1) diperoleh dari reinitialization acak antara batas pencarian [l (1), u (1)]. Nilai dari variabel keputusan lainnya x_{baru} (2), x_{baru} (3), ..., x_{baru} (n) juga dipilih sesuai dengan itu. Oleh karena itu, baik operasi, pertimbangan memori dan reinitialization acak dilakukan dengan persamaan (4)

$$x_{baru}(j) = \begin{cases} x_i(j) \in \{x_1(j), x_2(j), \dots, x_{HMS}(j)\}, dengan\ probabilitas\ HMCR \\ l(j) + \big(u(j) - l(j)\big).rand(0,1), dengan\ probabilitas\ 1 - HMCR \end{cases} \quad (4)$$

Setiap komponen yang diperoleh dengan pertimbangan memori diperiksa lebih lanjut untuk menentukan apakah harus disesuaikan. Untuk operasi ini, tingkat penyesuaian nada (PAR) didefinisikan untuk menetapkan frekuensi penyesuaian dan faktor bandwidth (BW) untuk mengendalikan pencarian lokal di sekitar elemen yang dipilih dari HM. Oleh karena itu, keputusan penyesuaian nada dihitung dengan persamaan (5)

$$x_{baru}(j) = \begin{cases} x_{baru}(j) = x_{baru}(j) \pm rand(0,1).BW, dengan\ probabilitas\ PAR \\ x_{baru}(j), dengan\ probabilitas\ (1 - PAR) \end{cases} \quad (5)$$

4. Pembaharuan Harmoni Memori

Setelah sebuah vektor harmoni baru x_{baru} dihasilkan, memori harmoni diperbarui dan membandingkan x_{baru} dan vektor harmoni terburuk x_w di HM.

Oleh karena itu x_{baru} akan mengganti x_w dan menjadi anggota baru HM jika nilai fitness x_{baru} lebih baik dari nilai fitness x_w.

5. Prosedur Komputasi

Jika tujuan pengujian untuk meminimalisasi maka pembaharuan HM berdasarkan $x_w^c = x_{baru}^c \, jika \, f(x_w^c) < f(x_{baru}^c)$. Namun jika tujuan pengujian untuk memaksimalkan makan pembaharuan HM berdasarkan $x_w^c = x_{baru}^c \, jika \, f(x_w^c) > f(x_{baru}^c)$.

2.3. Metode Improved Harmony Search Algorithm(IHSA)

IHSA merupakan sebuah metode yang diperkenalkan oleh Mahdavi, Fesanghary, dan Damangir pada tahun 2006, dimana metode tersebut merupakan perkembangan dari metode sebelumnya yakni metode *Harmony Search Algorithm* (HSA). Berikut ini akan dijelaskan mengenai metode HSA dan modifikasinya yaitu IHSA.

A. *Harmony Search Algorithm* (HSA)

Harmony Search Algorithm (HSA) merupakan sebuah metode optimasi *metahueristic* baru yang diperkenalkan oleh Z. W. Geem, J. H. Kim, dan G.V. Loganathan pada tahun 2001, dimana metode tersebut telah memperoleh hasil yang sangat baik di bidang optimasi[17].

HSA terinspirasi dari proses improvisasi musisi jazz, yakni dari fenomena musik opera yang terdiri dari berbagai macam alat musik dan menghasilkan melodi yang indah. Prinsip dari algoritma ini ialah meniru proses perbaikan harmoni yang dilakukan oleh kelompok paduan musik. Ketika kelompok paduan musik melakukan perbaikan pada harmoni musik yang dimainkan, maka akan terdapat tiga kemungkinan pilihan, yakni memainkan harmoni yang terkenal berdasarkan ingatan mereka, memainkan harmoni yang mirip dengan harmoni yang terkenal namun ada sedikit penyesuaian, atau membuat harmoni baru. Keunggulan HSA dibandingkan dengan teknik optimasi tradisional, antara lain:

1. HSA merupakan algoritma *metaheuristic*
2. HSA menggunakan pencarian acak *stochastic* (secara acak)
3. HSA tidak memerlukan informasi derivasi
4. Memiliki beberapa parameter
5. HSA dapat dengan mudah diadopsi dalam berbagai jenis masalah optimasi

Berikut ini merupakah parameter dalam HSA[18]:

1. *Harmony Memory Size* (HMS): Jumlah solusi simultan *vector* dalam *Harmony Memory* (HM)
2. *Harmony Memory Considering Rate* (HMCR): Tingkat atau persentase nilai *Harmony Search Algorithm* (HSA) yang dipilih dari memori harmoni
3. *Pitch Adjustment Rate* (PAR) : Penunjuk tingkat atau persentase pemilihan nilai yang berdekatan
4. *Number of Improvisations* (NI): Menunjukkan nomor iterasi dalam algoritma optimasi

Tahapan yang terdapat dalam proses HSA adalah sebagai berikut[19]:

1. Inisialisasi masalah dan parameter algoritma

   Pada tahap ini, masalah optimasi dijelaskan sebagai berikut:

   Minimalkan $f(x) = x_i \in X_i = 1, 2, \ldots, N$

   Dimana:

   $f(x)$ = fungsi tujuan

   x = sekumpulan setiap varibel keputusan x_i

   N = jumlah variabel keputusan

   X_i = sekumpulan kemungkinan rentang nilai untuk setiap variabel keputusan

   Pada langkah ini, parameter HSA di spesifikasikan. Parameter HSA terdiri dari jumlah vektor solusi didalam *harmony memory* (HM) yang disebut dengan *harmony memory size* (HMS), *harmony memory consideration rate*

(HMCR), *pitch adjusting rate* (PAR), dan kriteria pemberhentian yang disebut dengan *number of improvisations* (NI).

2. Inisialisasi *harmony memory* (HM)

   Pada tahap ini, matriks HM diisi dengan HMS yaitu, vektor solusi yang dirandom.

$$\text{HM}=\begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_{N-1}^1 & x_N^1 \\ x_1^2 & x_2^2 & \cdots & x_{N-1}^2 & x_N^2 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ x_1^{HMS-1} & x_2^{HMS-1} & \cdots & x_{N-1}^{HMS-1} & x_N^{HMS-1} \\ x_1^{HMS} & x_2^{HMS} & \cdots & x_{N-1}^{HMS} & x_N^{HMS} \end{bmatrix}$$

3. Improvisasi harmoni baru

   Improvisasi merupakan proses mendapatkan harmoni baru. Vektor harmoni baru dapat diperoleh berdasarkan aturan berikut (Chakraborty et al., 2007):

   - Memilih salah satu nilai dari HSA memory (HMCR)
   - Memilih salah satu nilai yang berdekatan dari HSA memory (pitch adjustmennt)
   - Memilih nilai acak dari kisaran nilai yang mungkin (randomization)

   Didalam memory consideration, nilai variabel keputusan pertama (x_1^1) untuk vektor baru dipilih dari salah satu nilai dalam rentang HM yang ditentukan ($x'^1_1 - x'^{HMS}_1$). Nilai dari variabel keputusan lainnya dipilih dengan cara yang sama. HMCR (0 sampai 1) adalah tingkat pemilihan secara random suatu nilai dari kisaran nilai yang mungkin. Pada langkah ini, HM consideration, pitch adjustment atau random selection diterapkan pada setiap variabel dari vektor harmoni baru secara bergantian.

4. Perbaharui HM

   Pada langkah ini, jika vektor harmoni baru lebih baik dibandingkan harmoni yang terdapat didalam HM, maka berdasarkan nilai fungsi objektif, harmoni baru termasuk dalam HM, dan harmoni terburuk tidak termasuk dalam HM.

5. Periksa kriteria pemberhentian

   Apabila kriteria pemberhentian terpenuhi (maksimum NI), maka komputasi dihentikan. Jika tidak, kembali ke langkah ke-3 dan 4.

B. *Improved Harmony Search Algorithm* (IHSA)

Perbedaan utama antara HSA dan IHSA terletak pada cara penyesuaian PAR dan BW. IHSA memperbaiki kinerja algoritma HSA dan menghilangkan kelemahannya. Metode ini menggunakan PAR dan BW pada langkah ke-3 (improvisasi).

2.4. Rangkuman

Harmony Search Algorithm (HSA) merupakan salah satu algoritma mataheuristik yang digunakan untuk melakukan optimalisasi. Dikatakan metaheuristic karena algoritma ini dapat diaplikasikan dalam berbagai macam masalah. Sedangkan Improved Harmony Search Algorithm (IHSA) merupakan perkembangannya, dimana perbedaan mendasar antara HSA dan IHSA terletak pada cara penyesuaian PAR dan BW. IHSA memperbaiki kinerja algoritma HSA dan menghilangkan kelemahannya. Metode ini menggunakan PAR dan BW pada langkah ke-3 (improvisasi).

2.5. Referensi

[1] D. Oliva, E. Cuevas, G. Pajares, D. Zaldivar, and M. Perez-cisneros, "Multilevel Thresholding Segmentation Based on Harmony Search Optimization," *J. Appl. Math.*, vol. 2013, 2013.

[2] H. Atmaca, M. Bulut, and D. Demir, "HISTOGRAM BASED FUZZY KOHONEN CLUSTERING NETWORK," pp. 1–4, 1996.

[3] N. Jabbar, S. I. Ahson, and M. Mehrotra, "Fuzzy Kohonen Clustering Network for Color Image Segmentation," vol. 3, pp. 254–257, 2011.

[4] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, S. Member, and T. Moriarty, "A Modified Fuzzy C-Means Algorithm for Bias Field Estimation

and Segmentation of MRI Data," vol. 21, no. 3, pp. 193–199, 2002.

[5] S. Murugavalli and V. Rajamani, "A HIGH SPEED PARALLEL FUZZY C-MEAN ALGORITHM FOR BRAIN," no. 6, 2006.

[6] R. C. Gonzales and R. E. Wood, *Digital Image Processing*. Prentice Hall, 2005.

[7] D. V. Fernandes, T. Thomas, N. Joseph, and J. Joseph, "Image Segmentation using Fuzzy - Kohonen Algorithm," no. Icmlc, pp. 176–179, 2011.

[8] Q.-K. Pan, P. N. Suganthan, J. J. Liang, and M. F. Tasgetiren, "A local-best harmony search algorithm with dynamic subpopulations," *Eng. Optim.*, vol. 42, no. 2, pp. 101–117, 2010.

[9] B. Alatas, "Chaotic harmony search algorithms," *Appl. Math. Comput.*, 2010.

[10] M. A. Awadallah, A. T. Khader, M. A. Al-Betar, and A. L. aro Bolaji, "Nurse scheduling using Harmony Search," in *Proceedings - 2011 6th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2011*, 2011.

[11] Z. W. O. O. Geem and J. C. Williams, "Ecological Optimization Using Harmony Search," *Soft Comput.*, 2008.

[12] Z. W. Geem, K. S. Lee, and Y. Park, "Application of Harmony Search to Vehicle Routing," *Am. J. Appl. Sci.*, 2005.

[13] O. M. Alia, R. Mandava, D. Ramachandram, and M. E. Aziz, "Dynamic fuzzy clustering using Harmony Search with application to image segmentation," *2009 IEEE Int. Symp. Signal Process. Inf. Technol.*, pp. 538–543, 2009.

[14] O. M. Alia *et al.*, "Color tongue image segmentation using Fuzzy Kohonen Networks and Genetic Algorithm," *Proc. SPIE---The Int. Soc. Opt. Eng.*, vol. 3962, no. 1, pp. 176–179, 2010.

[15] M. H. J. Vala and A. Baxi, "A review on Otsu image segmentation algorithm," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 2, pp. 387–389, 2013.

[16] W. Kang, Q. Yang, and R. Liang, "The Comparative Research on Image Segmentation Algorithms," pp. 703–707, 2009.

Bab 3. Segmentasi Citra Biometrik dengan HSA

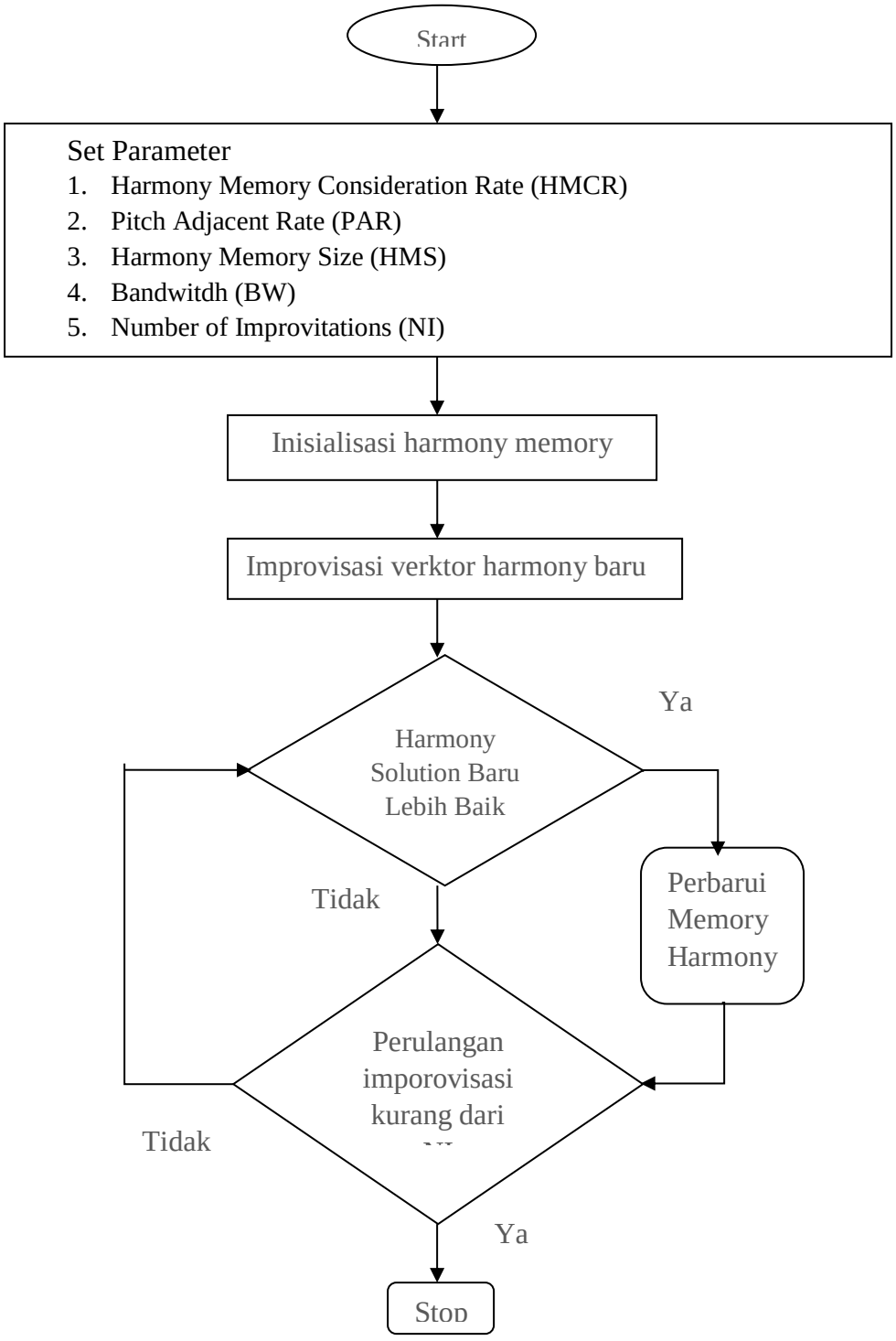
3.1. Segmentasi dengan HSA

Segmentasi citra menggunakan algoritma harmony search tidak bisa berdiri sendiri tanpa adanya teknik yang dilakukan untuk melakukan segmentasi. Segmentasi citra dibagi menjadi dua berdasarkan sifatnya yakni *edge based* (berbasis tepi) dan *region based* (berbasis wilayah). Metode segmentasi yang didasarkan pada sifat diskontinuitas piksel dianggap sebagai teknik berbasis batas atau tepi. Metode segmentasi berbasis tepi mencoba untuk menyelesaikan segmentasi citra dengan mendeteksi tepi atau piksel di antara daerah yang berbeda yang memiliki transisi intensitas yang cepat dan diekstraksi dan dihubungkan dengan bentuk batas objek tertutup[15]. Hasilnya adalah citra biner. Berdasarkan teori terdapat dua metode utama segmentasi berbasis tepi, berbasis histogram keabuan dan berbasis gradien[16]. Partisi segmentasi berbasis wilayah merupakan citra ke dalam wilayah yang serupa menurut satu set kriteria yang telah ditentukan. Segmentasi berbasis wilayah adalah pembagian citra ke area yang sama dengan piksel yang terhubung. Masing-masing piksel di suatu wilayah serupa dengan beberapa karakteristik atau properti yang dihitung seperti warna, intensitas dan / atau tekstur seperti *region growing*, *region splitting*, *merging*, *clustering*, dan *thresholding*.

Harmony search yang mengoptimasi segmentasi citra adalah teknik *clustering* dan teknik *thresholding*. Harmony search pada segmentasi citra teknik clustering melakukan optimasi nilai cluster sehingga dapat mengelompokkan tiap piksel dalam sebuah citra[13]. Sedangkan harmony search pada segmentasi citra teknik thresholding melakukan optimasi nilai ambang sehingga dapat memisahkan bagian foreground dan background dalam sebuah citra[1].

3.2. ALGORITMA HARMONY SEARCH ALGORITHM

Algoritma *harmony search* disajikan dalam bentuk *flowchart* [14] dan dapat dilihat pada gambar 2.



Gambar 2. *Flowchart* algoritma *harmony search*

Penjelasan :

Dalam algoritma harmony search, setiap solusi disebut dengan “harmoni” dan direpresentasikan dengan vektor n-dimensi. Populasi awal vektor harmoni dihasilkan secara acak dan disimpan dalam *harmony memory*. Kandidat harmoni baru didapatkan dari proses pada *harmony memory* menggunakan operasi *memory consideration*, baik dengan *random reinitialization* atau *pitch adjustment*. Selanjutnya, *harmony memory* diperbarui dengan melihat kandidat harmoni baru dan vektor harmoni terburuk di HM. Harmoni terburuk digantikan oleh kandidat baru sehingga bisa menjadi solusi yang lebih baik di *harmony memory*. Proses ini diulang sampai kriteria pemberhentian yang ditentukan terpenuhi.

3.3. Script Prosedur dan Fungsi

HarmonySearch.java

```
package Controller;

import Entity.Constant;
import Entity.HarmonyMemory;
import Entity.ParameterHSA;
import Entity.ResearchData;
import java.awt.image.BufferedImage;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JPanel;
import javax.swing.JTextArea;
```

```
/**
 *
 * @author RazorX
 */
public class HarmonySearch extends Thread{

    private BufferedImage originalImage;
    private int lowerValue;
    private int upperValue;

    private int decisionTotal; //jumlah variable keputusan
    private int hmSize;
    private int cycles;
    private double hmConsiderationRate;

    private HarmonyMemory[] harmonyMemory;
    private double par,bw;

    private double runningTime;

    private int imageWidth, imageHeight;

    Otsu otsu;

    JPanel panelOutput; //untuk simulasi
    JTextArea info;

    //tambahan kriteria
    double lastObjectiveValue;
    int unspoiledObjectiveValue;

    int finalIteration;

    public HarmonySearch(ResearchData data, int k, JPanel
panel, JTextArea info)
```

```
{

   panelOutput = panel;
   this.info = info;
   originalImage = data.getOriginalImage();
   otsu= new Otsu(originalImage);

   imageWidth = originalImage.getWidth();
   imageHeight = originalImage.getHeight();

    Date date = new Date();
    double start = date.getTime();
    initializeProblemAndParameter(k);
    initializeHarmonyMemory();
    improviseNewHarmony();
    date = new Date();
    double end = date.getTime();
    runningTime = (end - start)/1000;

}

public double[] getHistogram()
{
    return otsu.getHistogram();
}

private void initializeProblemAndParameter(int k)
{
    /*problem*/
    decisionTotal = k; //jumlah variable keputusan

    lowerValue = 0;
    upperValue = 255;
    /**parameter*/
```

```
        hmSize = ParameterHSA.getHarmonyMemorySize();
        cycles = ParameterHSA.getCycles();
        hmConsiderationRate =
ParameterHSA.getHMConsiderationRate();
        par = ParameterHSA.getPitchAdjustmentRate();
        bw = ParameterHSA.getBandwith();
    }

    private void initializeHarmonyMemory()
    {
        harmonyMemory = new HarmonyMemory[hmSize];
        int decisionVariable;
        double fitness;

        for (int i = 0; i < hmSize; i++)
        {
            harmonyMemory[i] = new
HarmonyMemory(decisionTotal);

            for(int j=0; j<decisionTotal; j++)
            {
                decisionVariable = getRandom();

                harmonyMemory[i].setSolutionVector(j,
decisionVariable);
            }

            int[] harmony =
harmonyMemory[i].getSolutionVector();
            Arrays.sort(harmony);

            fitness = getFitness(harmony);
            harmonyMemory[i].setSolutionVector(harmony);
            harmonyMemory[i].setFitness(fitness);

        `
```

```
}

    }

    private void improviseNewHarmony(){

        //GuiProgress.logProses("Improvisasi dan perbaharui
Harmony Memory");

        int generation = 0;
        double random;
        int D1; //index pemilihan lokasi pada harmoni memori
secara random
        int D2; //nilai variabel keputusan dari vektor
solusi
        int D3; //nilai variable keputusan berdasarkan PAR
dan BW
        int randomDecisionVariable;

        lastObjectiveValue = 0;
        unspoiledObjectiveValue = 0; //pencatat nilai yang
tak berubah

        do
        {
            HarmonyMemory newHarmonyMemoryVector = new
HarmonyMemory(decisionTotal);

            for(int i=0; i<decisionTotal; i++)
            {
                random = randProbability(0,1); //random dri
0-1;
                //System.out.println("vektor solusi ["+i+"]
| random : "+random);
                if (random < hmConsiderationRate)
                {
                    D1 = (int) random * hmSize; //pemilihan
```

```
index pada HM berdasarkan hmcr
                    //System.out.println("random < 
hmcr("+hmConsiderationRate+")");
                    random = randProbability(0,1);
                    //System.out.println("randomPar : 
"+random);
                    if(random < par){
                        random = randProbability(0,1);
                        D2 = 
newHarmonyMemoryVector.getSolutionVector(i);
                        D3 = adjustedDecisionVariable(D2,i); 
//dapatkan D3 dan sesuaikan agar tidak keluar batas nilai

newHarmonyMemoryVector.setSolutionVector(i, D3);


                        //System.out.println("random < 
par("+par+")");
                        //System.out.println("random lgi : 
"+random+"|D2: "+D2+" |D3: "+D3);
                    }
                    else{
                      D2 = 
harmonyMemory[D1].getSolutionVector(i);

newHarmonyMemoryVector.setSolutionVector(i, D2); //simpan ke 
harmony baru (sementara) utk dibandingkan
                      //System.out.println("random > 
par("+par+")");
                      //System.out.println("D2[D1]: "+D2);
                    }
                }
                else
                {
                    randomDecisionVariable = getRandom(); 
//random sesuai batas atas dan bawah dan sesuai x atau y.

newHarmonyMemoryVector.setSolutionVector(i, 
randomDecisionVariable);
```

```
                    //System.out.println("variable
solusi(D3):"+ randomDecisionVariable);
                }
            }

            int[] harmony =
newHarmonyMemoryVector.getSolutionVector();
            Arrays.sort(harmony);

            double fitness = getFitness(harmony);

newHarmonyMemoryVector.setSolutionVector(harmony);
            newHarmonyMemoryVector.setFitness(fitness);

            int indexOfWorstHarmony =
getIndexOfWorstHarmony(harmonyMemory);

            double newHarmonySolution =
newHarmonyMemoryVector.getFitness();
            double worstHarmonySolution =
harmonyMemory[indexOfWorstHarmony].getFitness();

            if(newHarmonySolution > worstHarmonySolution)
            {
                //harmonyMemory[indexOfWorstHarmony] = new
HarmonyMemory(decisionTotal);
                harmonyMemory[indexOfWorstHarmony] =
newHarmonyMemoryVector;
                //GuiProgress.logProses("HM baru lebih baik,
Gantikan HM Buruk");
            }

            /**Menampilkan Proses yang berjalan dan update
RealTime. Menggunakan Thread**/

            doSimulation(harmony,fitness);
```

```
//GuiProgress.prosesBerjalanProgressBar.setValue(generation*
100/cycles+1);

            generation++;

        }while(generation < cycles &&
!isBestfitnessUnspoiled());

        finalIteration = generation;
    }

    public int getTotalIteration()
    {
        return finalIteration;
    }

    private void doSimulation(int[] harmony, double fitness)
    {

        if(Constant.simulationStatus)
        {
            Controller control = new Controller();
            ThresholdingImage th = new ThresholdingImage();
            BufferedImage tempImage =
th.getSegmentedImage(originalImage, harmony);
            control.updatePanelCitra(tempImage,
panelOutput);

            int[] tempBest =
getSolution().getSolutionVector();
            double tempBestFitness =
getSolution().getFitness();

            info.setText("Threshold terbaik saat ini : \n");
            for(int i=0; i<tempBest.length; i++)
```

```
            {
                info.append("th["+(i+1)+"] =
"+tempBest[i]+"\n");
            }
            info.append("Fitness :
"+tempBestFitness+"\n\n");



            info.append("Threshold Percobaan : \n");
            for(int i=0; i<harmony.length; i++)
            {
                info.append("th["+(i+1)+"] =
"+harmony[i]+"\n");
            }
            info.append("Fitness : "+tempBestFitness+"\n");


            try {
                Thread.sleep(500);
            } catch (InterruptedException ex) {

Logger.getLogger(HarmonySearch.class.getName()).log(Level.SE
VERE, null, ex);
            }
        }
    }


    private boolean isBestfitnessUnspoiled()
    {
        if(Constant.unspoiledStatus)
        {
           //**Kriteria tambahan untuk stop**/
            double currentObjectiveValue =
harmonyMemory[getIndexOfBestHarmony(harmonyMemory)].getFitne
ss();


            if(currentObjectiveValue==lastObjectiveValue)
```

```
            {
                unspoiledObjectiveValue++;
            }
            else
            {
                unspoiledObjectiveValue = 0;
            }

            lastObjectiveValue = currentObjectiveValue;

            if(unspoiledObjectiveValue == (int)cycles*0.1)
//kalo sudah 10%
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }

    private int getIndexOfWorstHarmony(HarmonyMemory[]
harmonyMemory)
    {
        int index;
        double worst,possibleWorst;

        worst = harmonyMemory[0].getFitness();
        index = 0;
```

```
        for(int i=1; i<hmSize; i++)
        {
            possibleWorst = harmonyMemory[i].getFitness();

            if(possibleWorst < worst)
            {
                worst = possibleWorst;
                index = i;
            }
        }

        return index;
    }

    private int getIndexOfBestHarmony(HarmonyMemory[]
harmonyMemory)
    {
        int index;
        double best,possibleBest;

        best = harmonyMemory[0].getFitness();
        index = 0;
        for(int i=1; i<hmSize; i++)
        {
            possibleBest = harmonyMemory[i].getFitness();

            if(possibleBest > best)
            {
                best = possibleBest;
                index = i;
            }
        }
```

```
      return index;
    }

    public HarmonyMemory getSolution()
    {
        return
harmonyMemory[getIndexOfBestHarmony(harmonyMemory)];
    }

    public double getRunningTime()
    {
        return runningTime;
    }

    // kurang metode step 1, step 4 dan metode step 5;
    private int getRandom()
    {
        int randomDecisionVariable;

        randomDecisionVariable =
randDecisionVariable(lowerValue,upperValue);

        return randomDecisionVariable;
    }

    private int randDecisionVariable(int lowerValue, int
upperValue)
    {
        //int randomNum =
ThreadLocalRandom.current().nextInt(lowerValue, upperValue +
1);
        int randomNum = (int)
Math.floor(Math.random()*(upperValue-lowerValue+1)) +
lowerValue;
        return randomNum;
    }
```

```
    private double randProbability(int min, int max)
    {
        Random random = new Random();
        double value = min + (max - min) *
random.nextDouble();

        return value;
    }

    private double getFitness(int[] harmony)
    {
        double fitness;

        otsu.calculateOtsuVariance(harmony);
        fitness = otsu.getOtsuVariance();

        return fitness;
    }



    private int adjustedDecisionVariable(int D2, int i)
    {
        int D3;
        double random = randProbability(0,1);
        if(random < 0.5d){
            D3 = D2 + (int)(random * bw);
        }
        else{
            D3 = D2 - (int)(random * bw);
        }


        if(D3 < lowerValue) D3 = lowerValue;
        else if (D3 > upperValue) D3 = upperValue;
```

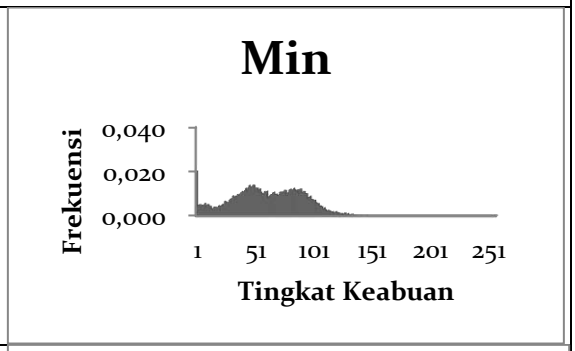
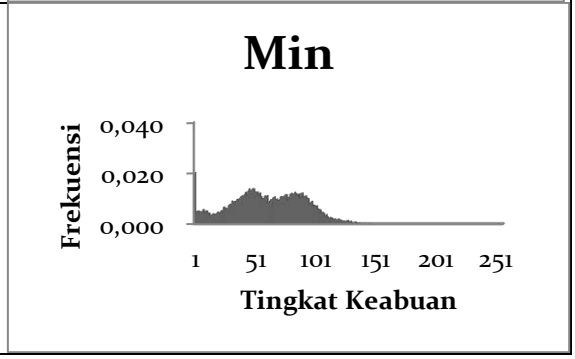
```
            return D3;
        }

}
```

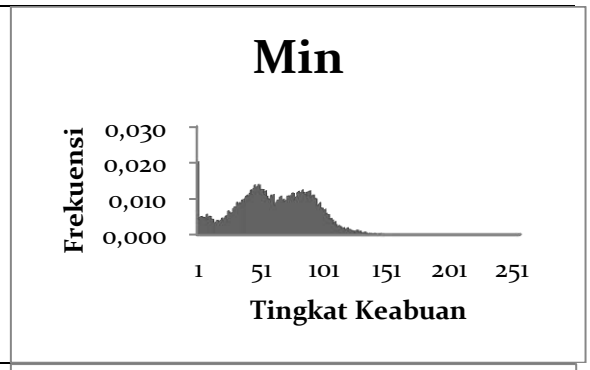
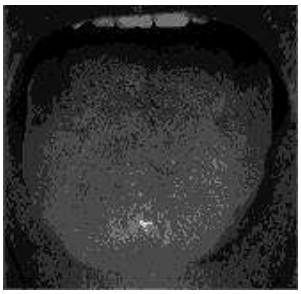
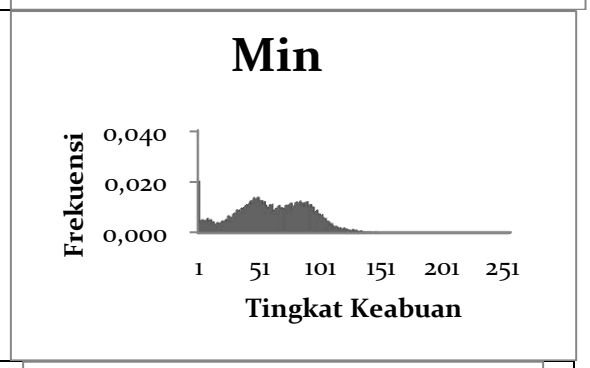
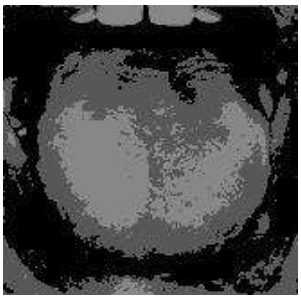
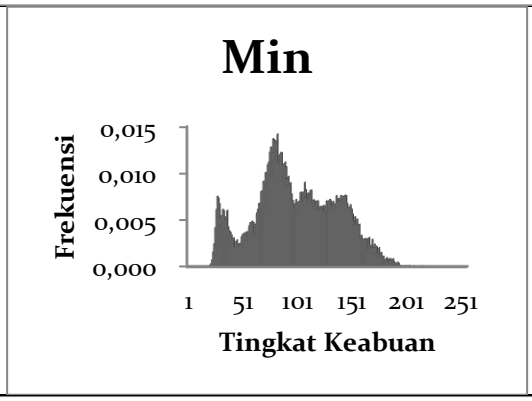
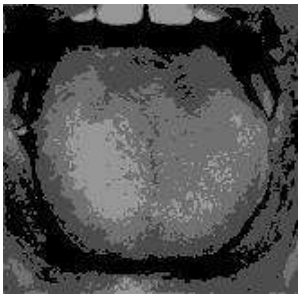
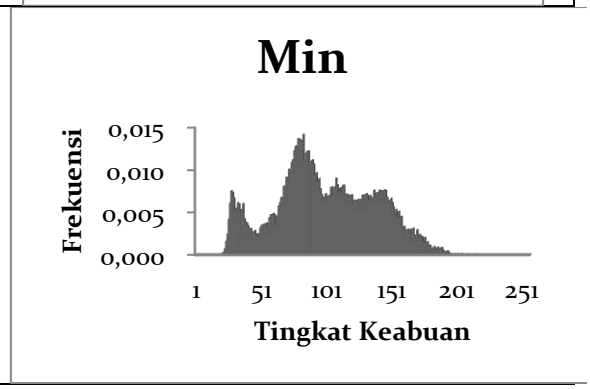
3.4. Hasil Uji coba

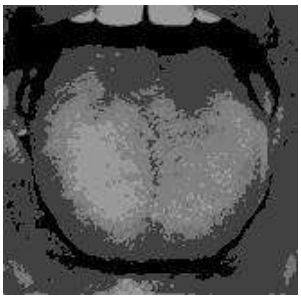
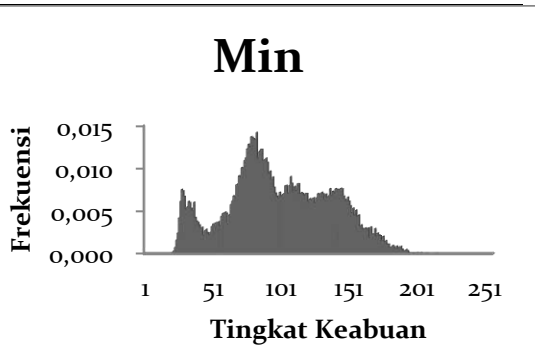
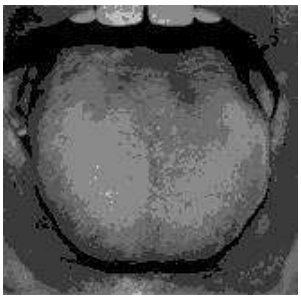
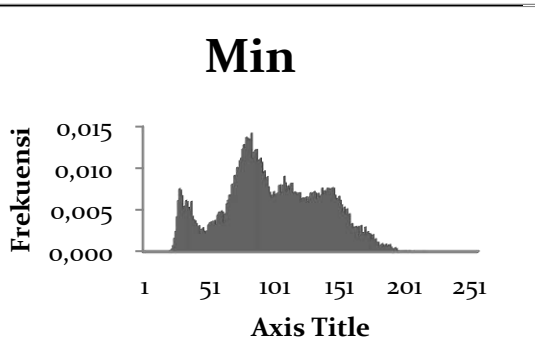
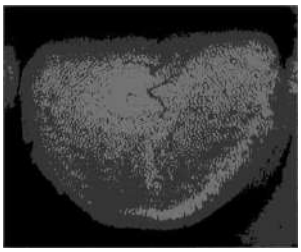
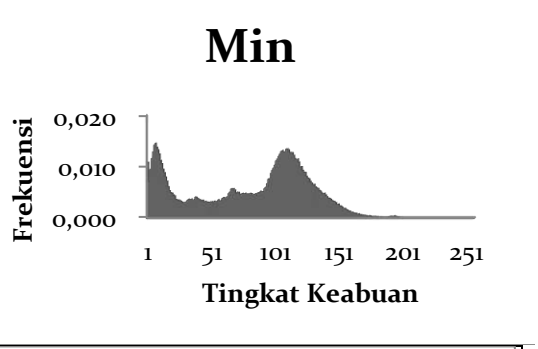
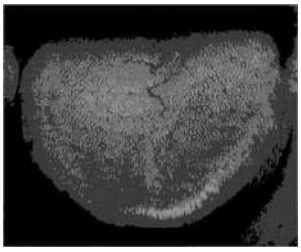
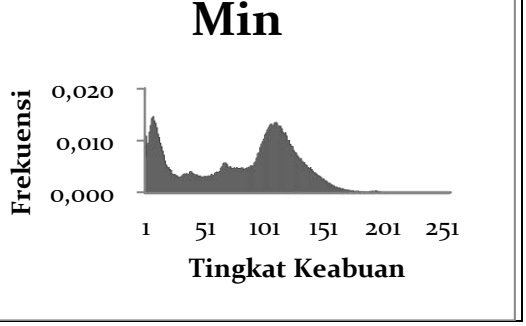
Terdapat 12 dataset citra biometric yang penulis gunakan namun hanya ditampilkan 5 pengujian saja. Hasil pengujian berupa citra segmentasi berdasarkan banyak *threshold threshold* dan disajikan dalam bentuk histogram yang dapat dilihat pada table 2.

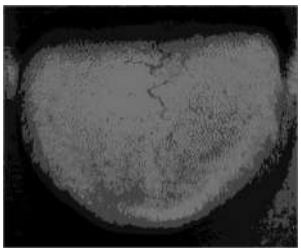
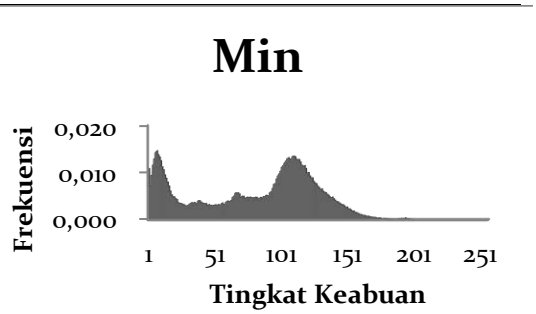
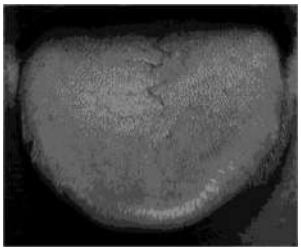
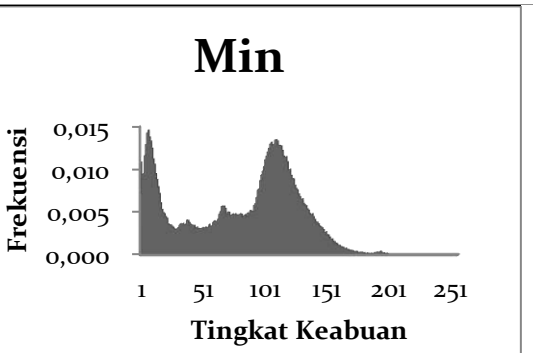
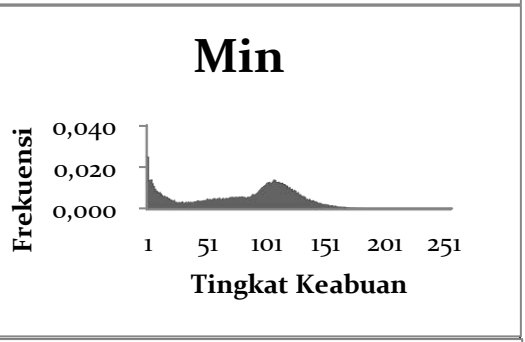
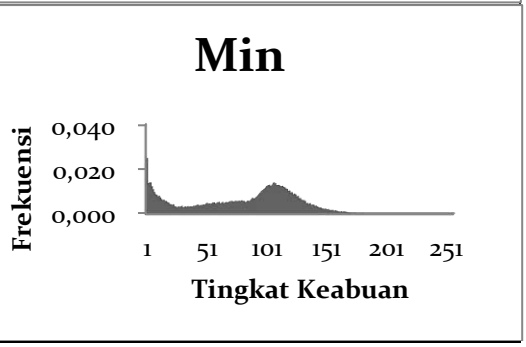
TABEL 2

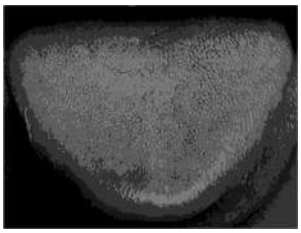
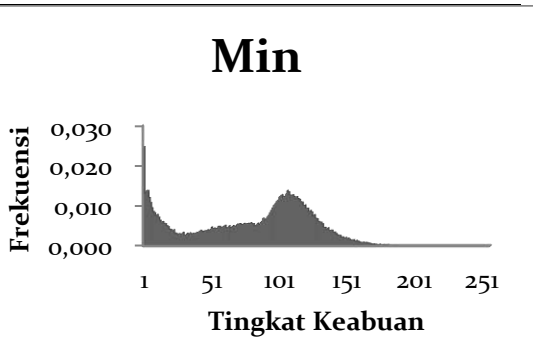
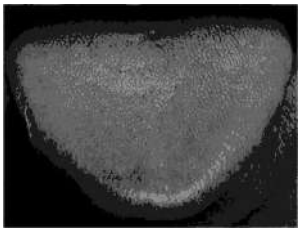
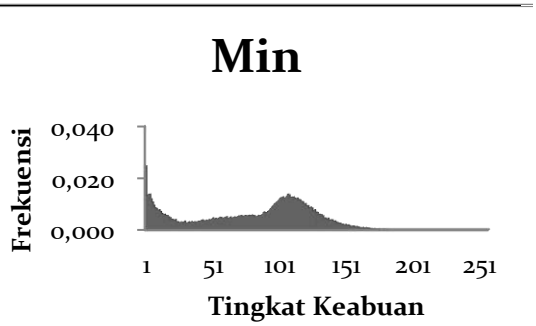
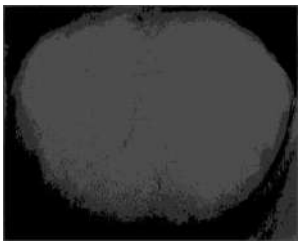
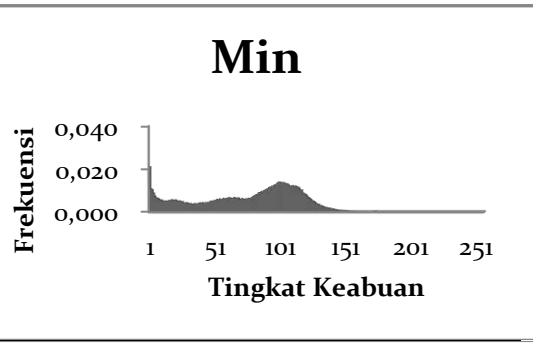
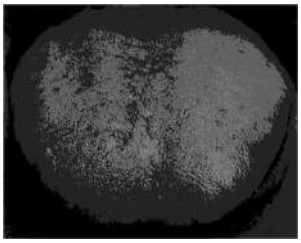
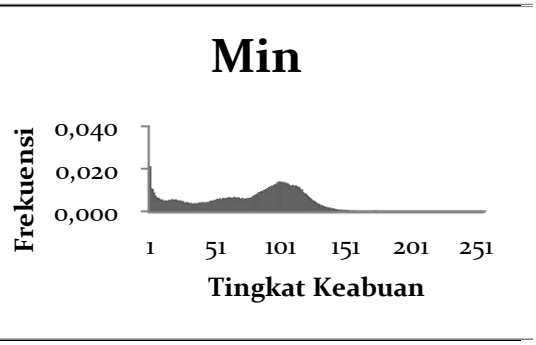
Hasil Pengujian menggunakan Citra Biometric

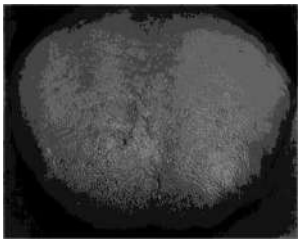
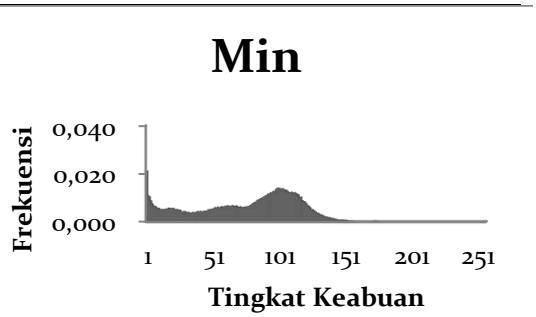
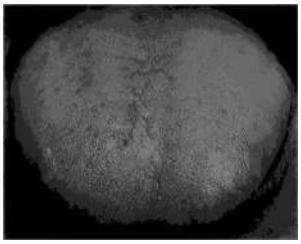
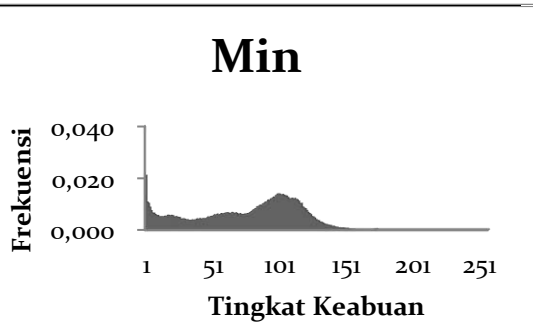
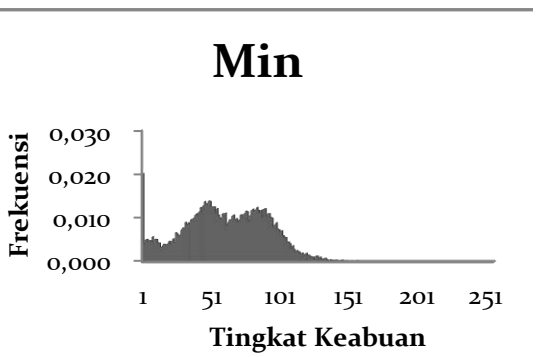
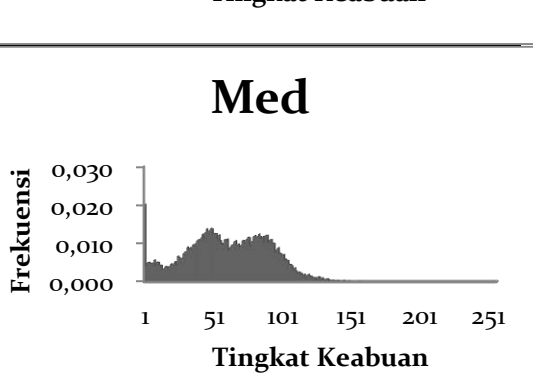
Citra	k	*Threshold* best	Hasil histogram
	2	40,66	
	3	27,68,103	

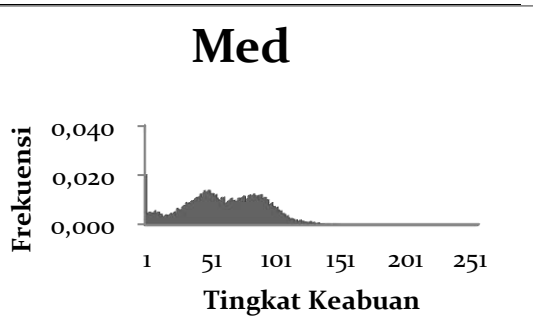
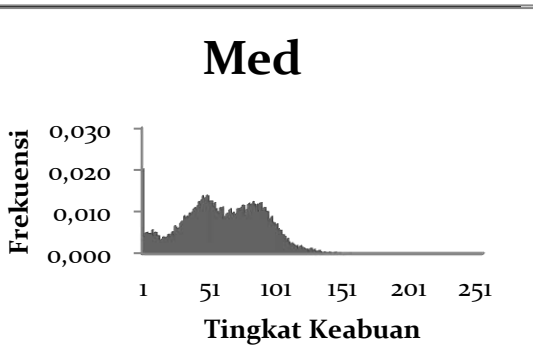
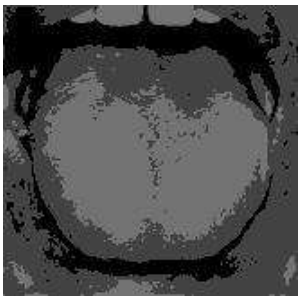
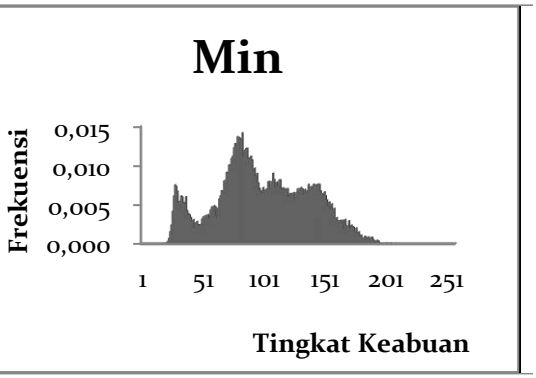
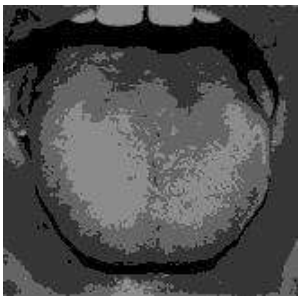
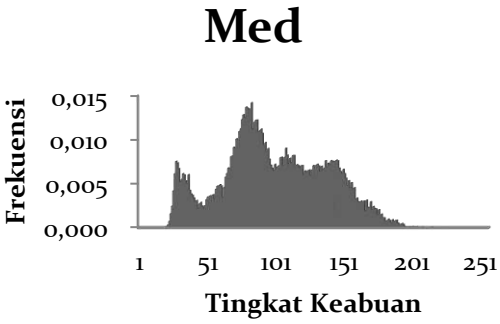
	4	9,59,95,111	
	5	15,47,74,114,209	
	2	88,135	
	3	76,112,150	

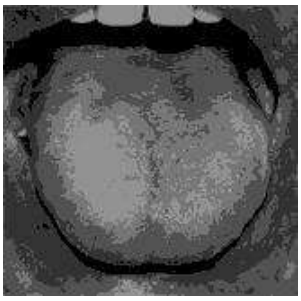
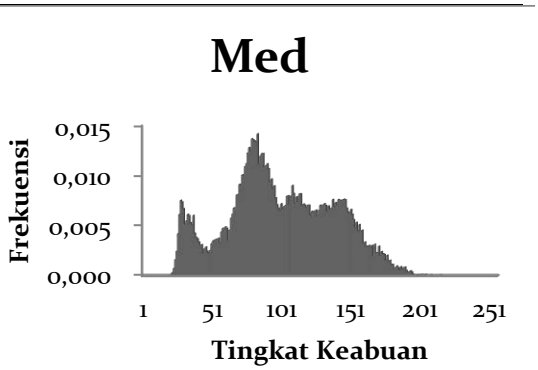
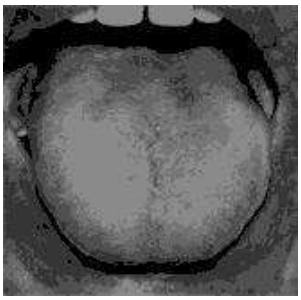
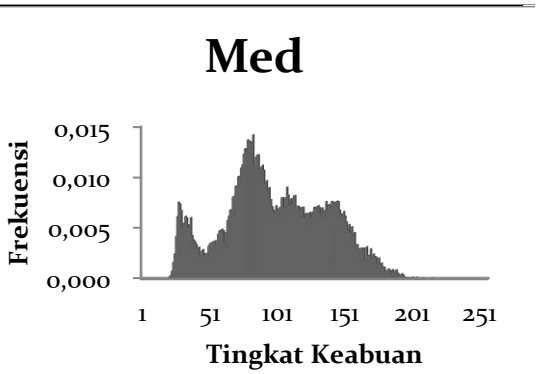
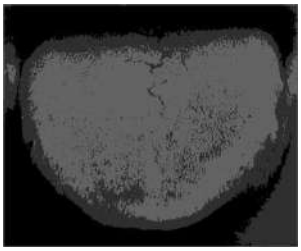
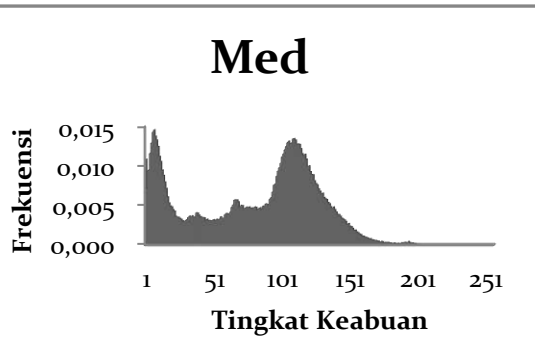
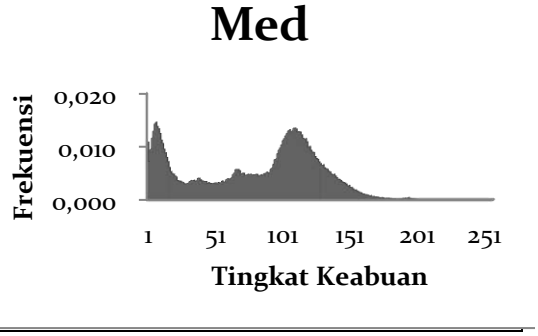
	4	65,114,128 ,153	
	5	58,90,109 ,137,187	
	2	53,115	
	3	59,112,145	

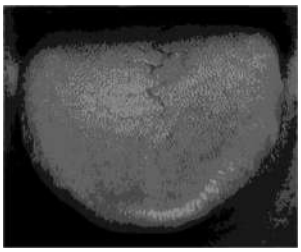
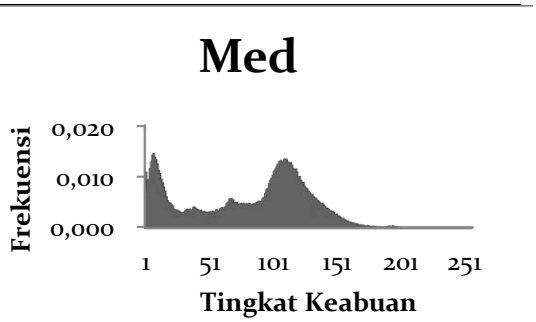
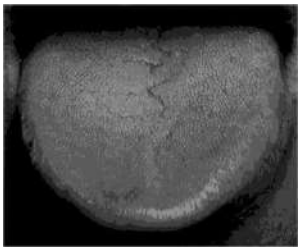
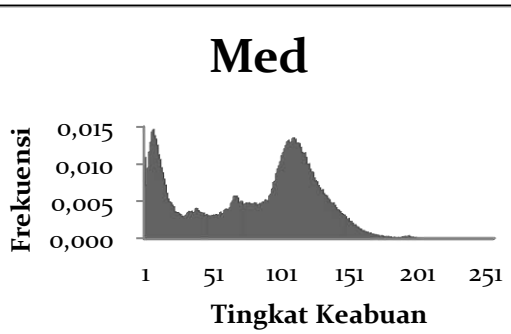
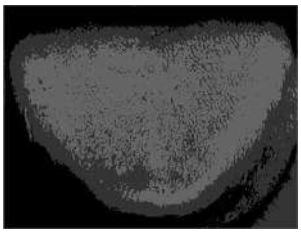
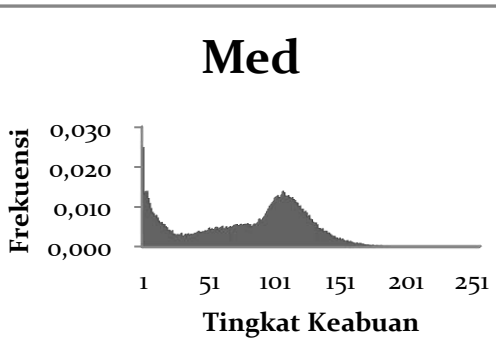
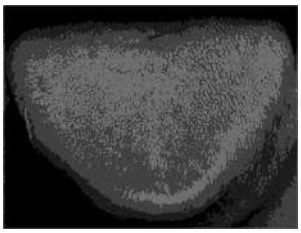
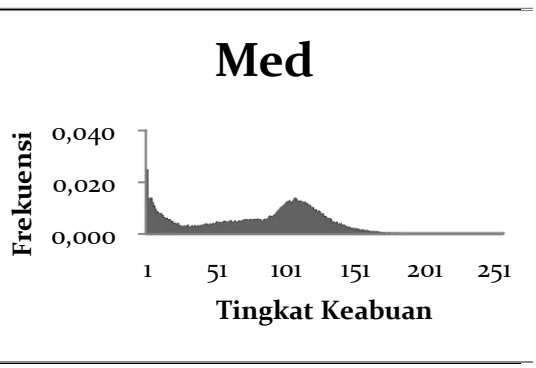
	4	17,68,100, 116	
	5	17,51,78,1 01,136	
	2	18,82	
	3	8,49,90	

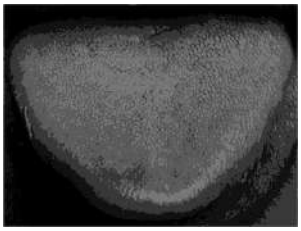
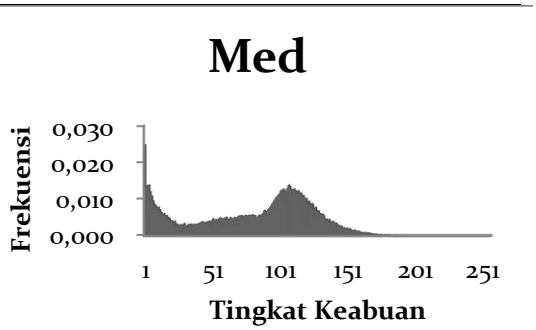
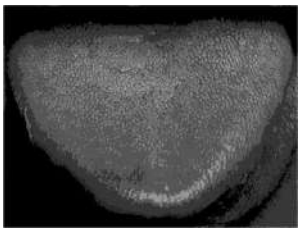
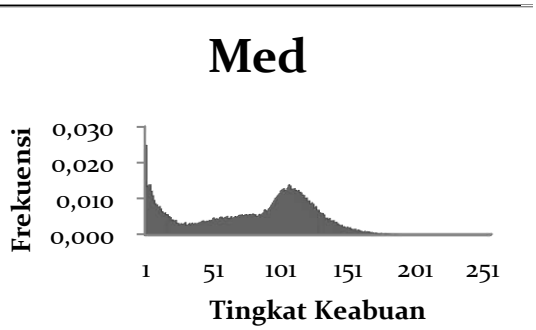
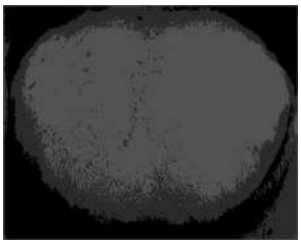
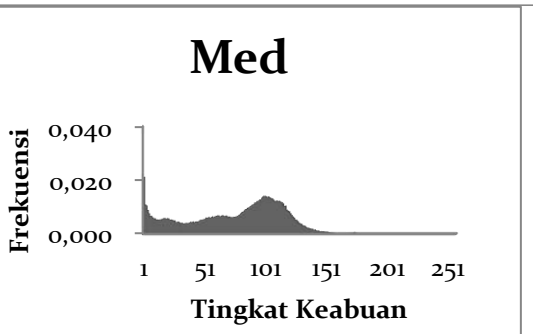
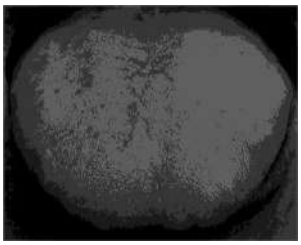
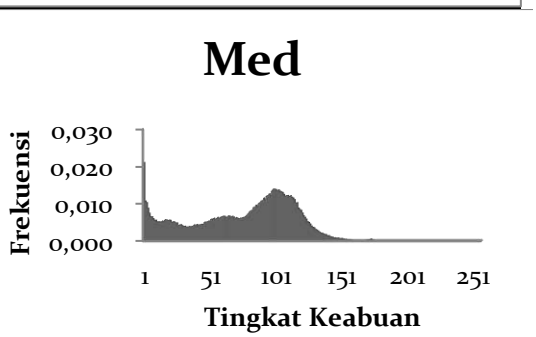
	4	25,52,94,121	
	5	28,83,84,102,139	
	2	49,76	
	3	36,98,124	

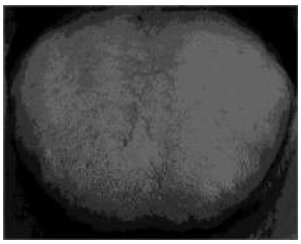
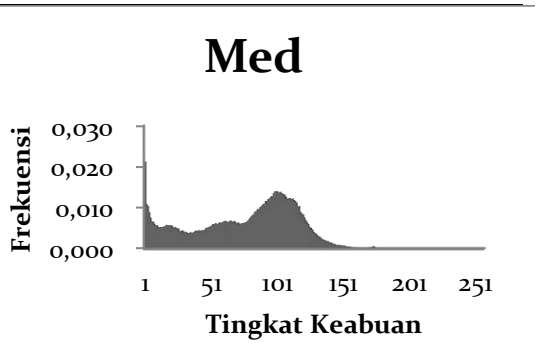
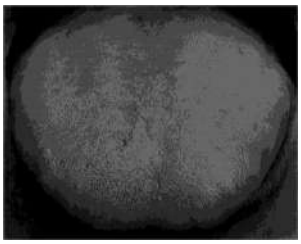
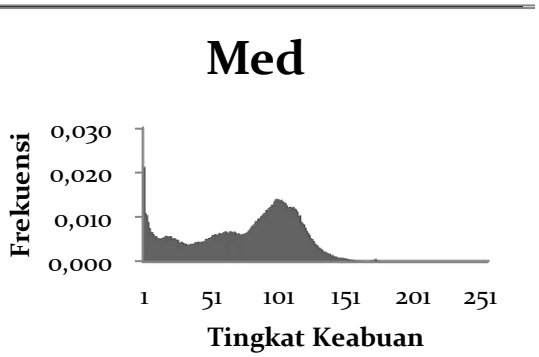
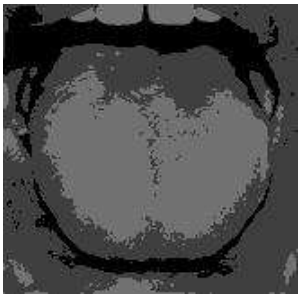
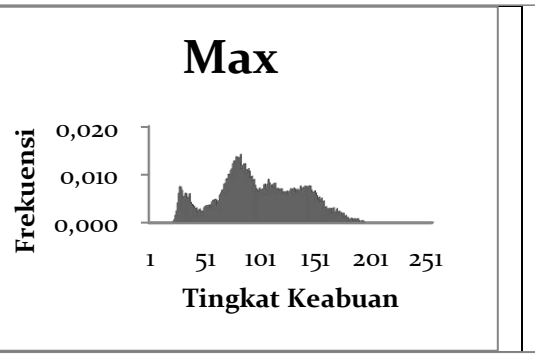
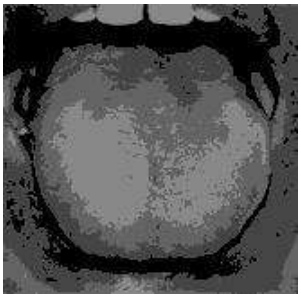
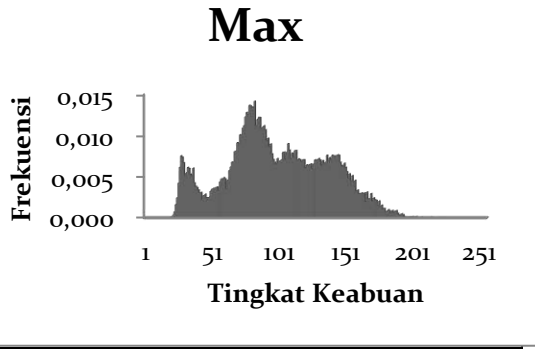
	4	19,68,99,129	
	5	40,68,93,107,128	
	2	45,79	
	3	35,59,88	

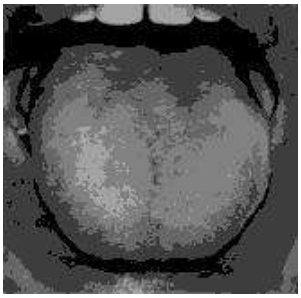
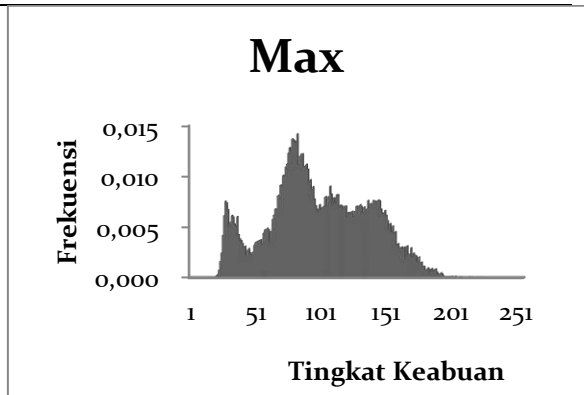
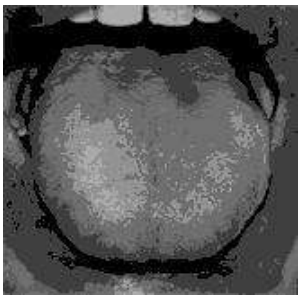
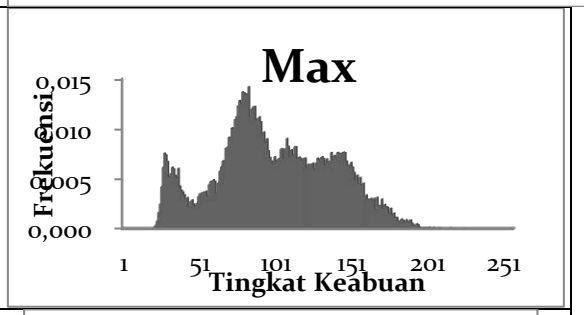
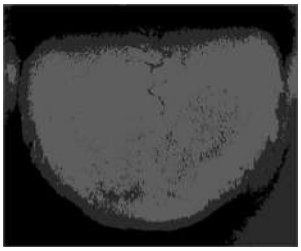
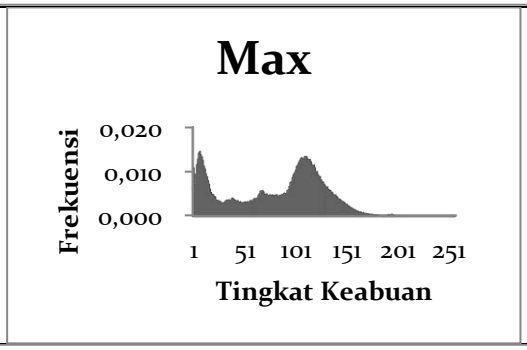
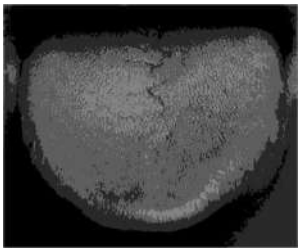
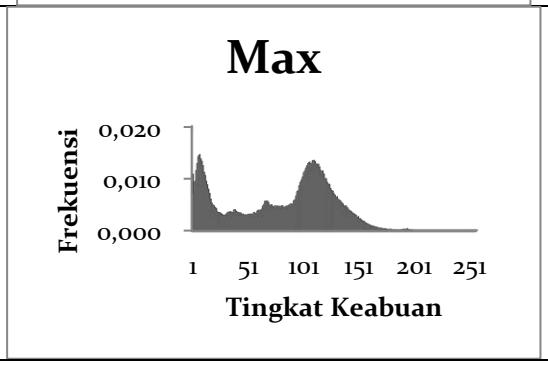
	4	8,45,61,88	
	5	38,64,92,114,242	
	2	67,115	
	3	54,101,139	

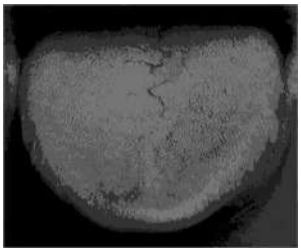
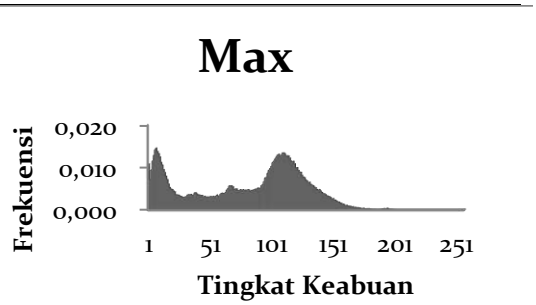
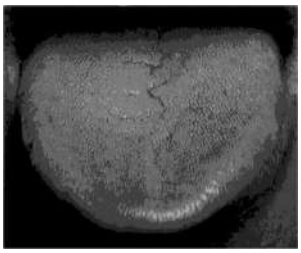
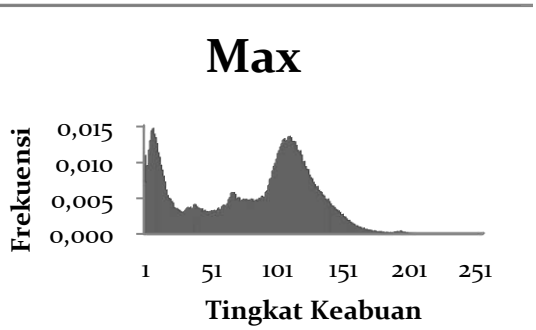
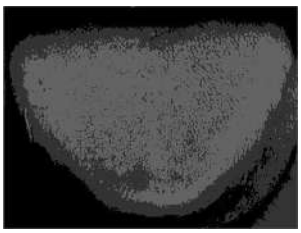
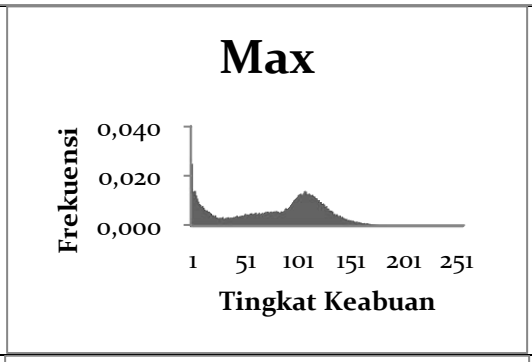
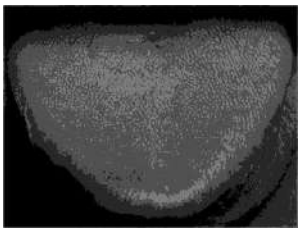
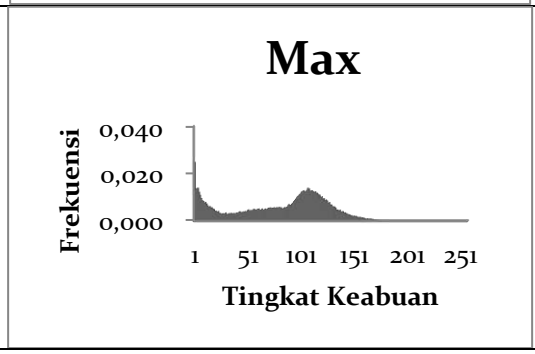
	4	51,82,119,146	
	5	54,80,103,119,138	
	2	47,99	
	3	23,71,112	

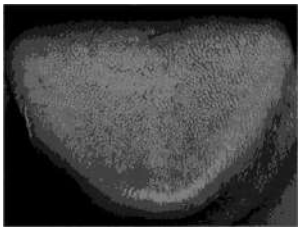
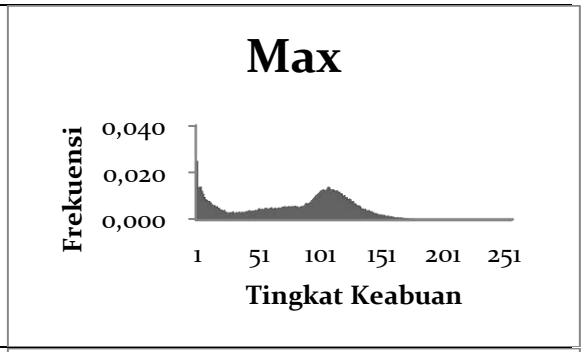
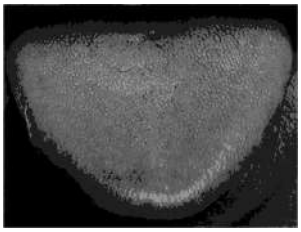
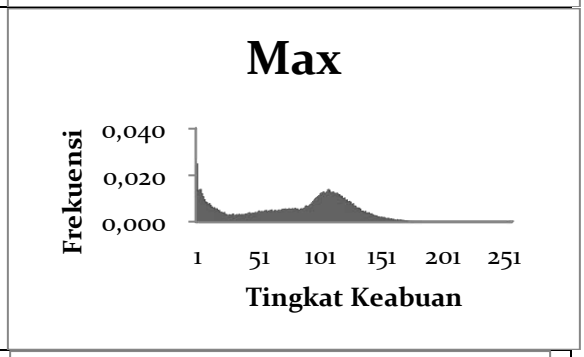
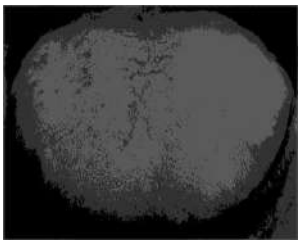
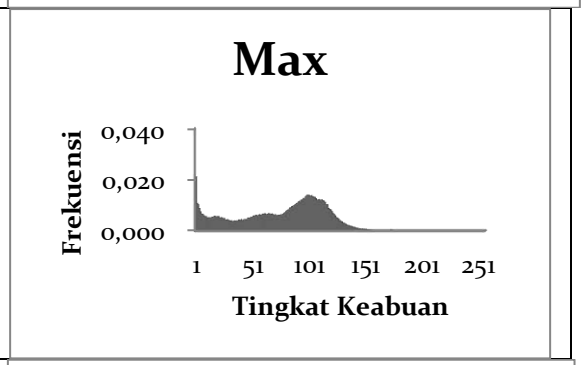
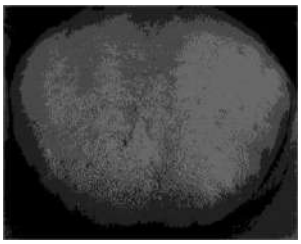
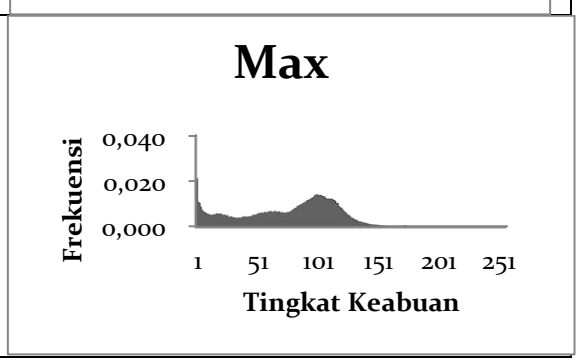
	4	24,73,98,135	
	5	36,65,91,117,142	
	2	51,101	
	3	29,65,11	

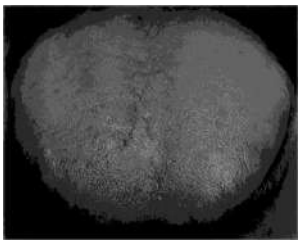
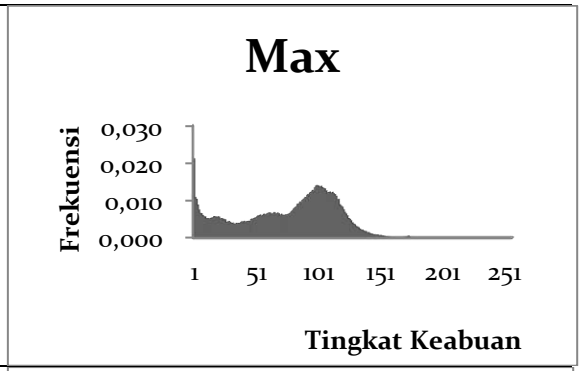
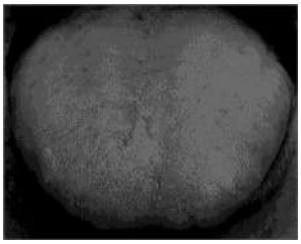
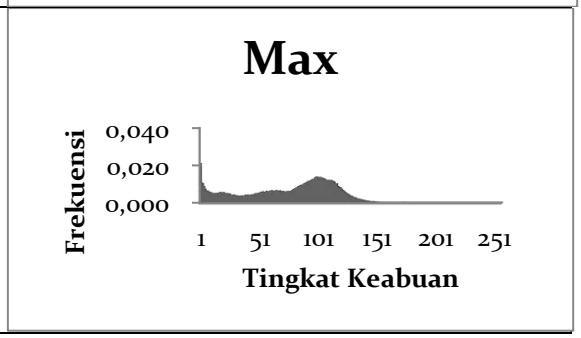
	4	18,56,92,120	
	5	40,55,87,117,151	
	2	43,82	
	3	27,54,95	

	4	17,48,77,101	
	5	21,43,73,103,184	
	2	63,114	
	3	69,100,136	

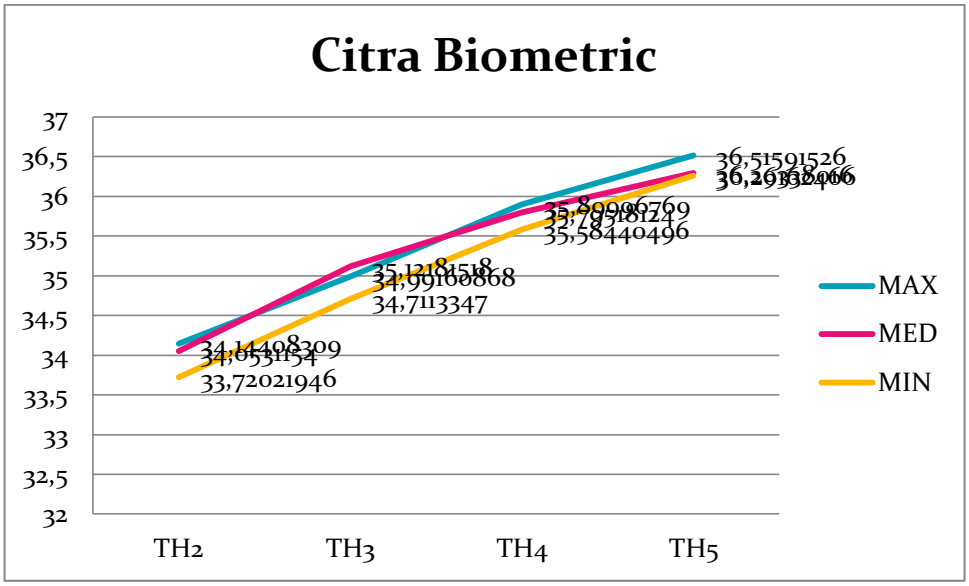
	4	62,103,129,163	
	5	63,95,111,147,171	
	2	2,94	
	3	41,94,131	

	4	31,51,93,117	
	5	37,65,102,124,165	
	2	51,99	
	3	40,81,123	

	4	28,65,101, 125	
	5	31,82,97,11 5,147	
	2	50,90	
	3	33,73,104	

	4	42,77,99,130	
	5	24,48,66,84,107	

Performa algoritma *harmony search* pada citra biometric dilihat menggunakan PSNR yang dapat membandingkan kualitas citra masukan dan citra keluaran. Grafik PSNR berdasarkan *threshold* dan tingkat parameter tiap citra biometric disajikan pada gambar 2.



3.5. Rangkuman

Pengujian citra biometric yang tampak pada grafik histogram menunjukan penyebaran nilai intensitas piksel citra keabuan. Nilai intensitas 0 menunjukan warna hitam murni dan nilai intensitas 255 menunjukan warna putih murni. Sampel yang ditampilkan terdiri dari 5 citra dengan 4 *threshold* dan 3 tingkat parameter HSA. Sehingga totalnya adalah 60 keluaran citra segmentasi. Citra biometric merupakan citra dengan objek lidah, ketika citra lidah memiliki warna lain selain warna lidah didalam objek lidah maka citra tersebut menghasilkan histogram citra yang baik. Citra yang baik berdasarkan grafik histogram adalah histogram mengisi daerah keabuan secara penuh dengan distribusi yang merata pada tiap intensitas keabuan. Perubahan *threshold* dan tingkat nilai parameter algoritma *harmony search* tidak mempengaruhi bentuk grafik secara signifikan. Namun, frekuensi kemunculan intensitas berubah tapi tidak begitu besar. Semakin besar nilai *threshold* yang dimasukan, maka perbedaan intensitas keabuan pada objek lidah semakin terlihat. Noda pada lidah dapat dibedakan menggunakan algoritma ini. Penaikan nilai parameter *harmony search* juga mengakibatkan perbedaan intensitas keabuan objek lidah namun tidak sebaik ketika nilai *threshold* yang dibesarkan. Penggabungan pembesaran *threshold* dan penaikan tingkat menunjukan hasil yang lebih baik.

Pengujian kualitas citra biometric yang menggunakan algoritma *harmony search* disajikan pada grafik PSNR 12 citra biometric. Pengujian melibatkan 12 citra biometric dengan 4 *threshold* dan 3 tingkat nilai parameter algoritma *harmony search*. 12 citra pada setiap *threshold* dicari nilai rata-rata PSNR-nya. Nilai rata-rata PSNR tiap *threshold* disajikan dalam bentuk grafik berdasarkan tingkat parameter HSA dan setelah itu dianalisa. PSNR digunakan untuk membandingkan kualitas citra masukan dan citra keluaran. Nilai PSNR dibawah 30dB menunjukan kualitas yang rendah. Pada citra biometric penggunaan tingkat minimum tidak menunjukan kualitas yang lebih baik dari tingkat lainnya jika dibanding

berdasarkan nilai *threshold* yang sama. Penggunaan nilai medium hampir menghasilkan kualitas yang sama dengan penggunaan nilai parameter maksimum. Penggunaan tingkat parameter medium pada *threshold* 3 menghasilkan kualitas citra yang lebih baik dari parameter lainnya, namun semakin menurun ketika *threshold* diperbesar. Penurunan kualitas ini membuat penggunaan tingkat parameter medium pada *threshold* 5 sama dengan penggunaan parameter minimum. Penggunaan parameter maksimum menghasilkan kualitas yang paling baik diantara tingkat parameter lainnya. Nilai kualitas citra yang menggunakan tingkat parameter maksimum tidak pernah mengalami penurunan.

3.6. Referensi

[1] D. Oliva, E. Cuevas, G. Pajares, D. Zaldivar, and M. Perez-cisneros, "Multilevel Thresholding Segmentation Based on Harmony Search Optimization," *J. Appl. Math.*, vol. 2013, 2013.

[2] H. Atmaca, M. Bulut, and D. Demir, "HISTOGRAM BASED FUZZY KOHONEN CLUSTERING NETWORK," pp. 1–4, 1996.

[3] N. Jabbar, S. I. Ahson, and M. Mehrotra, "Fuzzy Kohonen Clustering Network for Color Image Segmentation," vol. 3, pp. 254–257, 2011.

[4] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, S. Member, and T. Moriarty, "A Modified Fuzzy C-Means Algorithm for Bias Field Estimation and Segmentation of MRI Data," vol. 21, no. 3, pp. 193–199, 2002.

[5] S. Murugavalli and V. Rajamani, "A HIGH SPEED PARALLEL FUZZY C-MEAN ALGORITHM FOR BRAIN," no. 6, 2006.

[6] R. C. Gonzales and R. E. Wood, *Digital Image Processing*. Prentice Hall, 2005.

[7] D. V. Fernandes, T. Thomas, N. Joseph, and J. Joseph, "Image Segmentation using Fuzzy - Kohonen Algorithm," no. Icmlc, pp. 176–179, 2011.

[8] Q.-K. Pan, P. N. Suganthan, J. J. Liang, and M. F. Tasgetiren, "A local-best harmony search algorithm with dynamic subpopulations," *Eng. Optim.*, vol. 42, no. 2, pp. 101–117, 2010.

[9] B. Alatas, "Chaotic harmony search algorithms," *Appl. Math. Comput.*, 2010.

[10] M. A. Awadallah, A. T. Khader, M. A. Al-Betar, and A. L. aro Bolaji, "Nurse scheduling using Harmony Search," in *Proceedings - 2011 6th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2011*, 2011.

[11] Z. W. O. O. Geem and J. C. Williams, "Ecological Optimization Using Harmony Search," *Soft Comput.*, 2008.

[12] Z. W. Geem, K. S. Lee, and Y. Park, "Application of Harmony Search to Vehicle Routing," *Am. J. Appl. Sci.*, 2005.

[13] O. M. Alia, R. Mandava, D. Ramachandram, and M. E. Aziz, "Dynamic fuzzy clustering using Harmony Search with application to image segmentation," *2009 IEEE Int. Symp. Signal Process. Inf. Technol.*, pp. 538–543, 2009.

[14] O. M. Alia *et al.*, "Color tongue image segmentation using Fuzzy Kohonen Networks and Genetic Algorithm," *Proc. SPIE---The Int. Soc. Opt. Eng.*, vol. 3962, no. 1, pp. 176–179, 2010.

[15] M. H. J. Vala and A. Baxi, "A review on Otsu image segmentation algorithm," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 2, pp. 387–389, 2013.

[16] W. Kang, Q. Yang, and R. Liang, "The Comparative Research on Image Segmentation Algorithms," pp. 703–707, 2009.

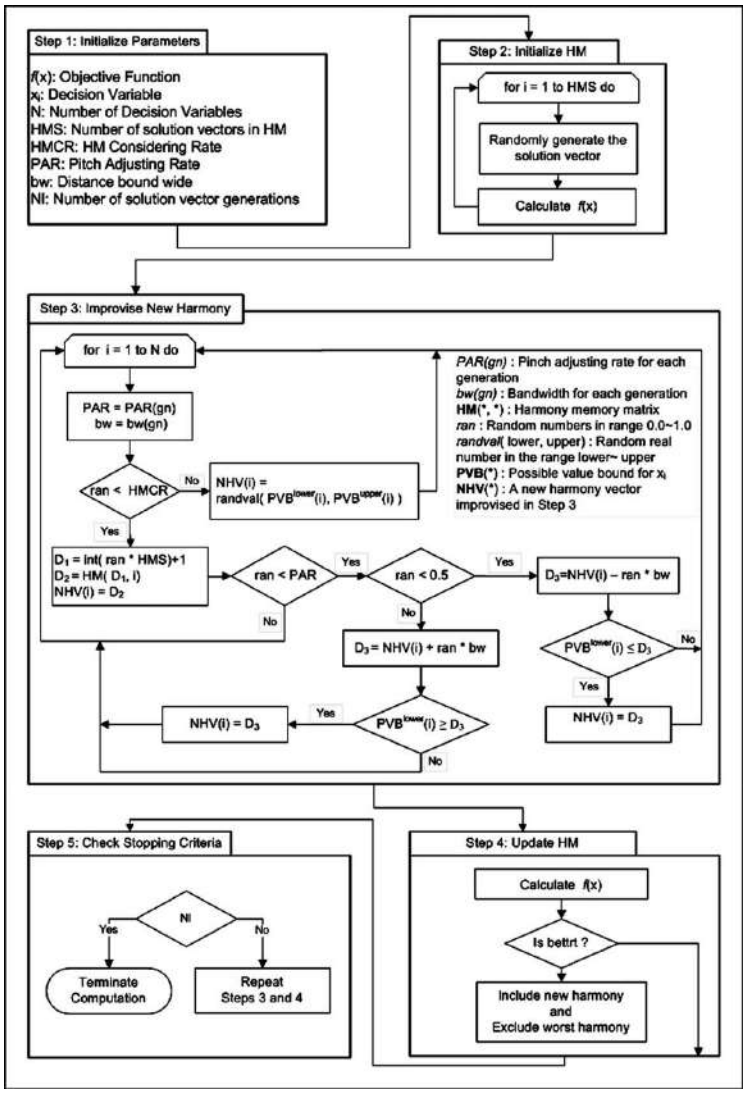
Bab 4. Segmentasi Citra Biometrik dengan IHSA

4.1. Segmentasi dengan IHSA

Segmentasi citra merupakan suatu tahapan dalam pengolahan citra yang membagi citra tersebut ke dalam daerah-daerah *(regions)* yang *non-overlapping* dan homogen, dimana masing-masing daerah saling terhubung dan berbeda dari daerah tetangga (Jabbar, Ahson, dan Mehrotra, 2010). Tujuan utama dilakukannya segmentasi citra ini ialah untuk menyederhanakan gambar sehingga dapat lebih mudah untuk dianalisa selanjutnya. Tingkat ketelitian dari segmentasi citra bergantung pada masalah yang akan diselesaikan sehingga proses segmentasi daripada obyek citra dapat dihentikan saat obyek atau daerahnya telah ditemukan.

Proses segmentasi pada citra sangat menentukan kesuksesan dari prosedur analisis citra. Semakin tinggi tingkat akurasi yang dihasilkan pada tahap segmentasi maka akan semakin bagus pula hasil analisisnya. Segmentasi dibagi berdasarkan 2 sifat dari intensitas piksel, yaitu *discontinuity* dan *similarity*. *Discontinuity* mempartisi citra jika terjadi perubahan intensitas secara tiba-tiba *(edge-based),* sedangkan s*imilarity* mempartisi citra menjadi wilayah atau obyek yang memiliki kesamaan kriteria tertentu *(region-based)* (Gonzales, Woods, dan Masters, 2008).

IHSA merupakan sebuah metode yang diperkenalkan oleh Mahdavi, Fesanghary, dan Damangir pada tahun 2006, dimana metode tersebut merupakan perkembangan dari metode sebelumnya yakni metode *Harmony Search Algorithm* (HSA). Diagram alir mengenai algoritma IHSA dapat dilihat pada Gambar 3 berikut ini:



Gambar 3 Diagram Alir IHSA (Mahdavi et al., 2007)

Tabel 3 dibawah ini merupakan parameter IHSA yang digunakan dalam penelitain ini:

TABEL 3

Parameter Yang Digunakan

NI	HMS	HMCR	PAR Min	PAR Max	BW Min	BW Max
2.000	5	0,9	0,01	0,99	0,001	0,1

4.2. Script Prosedur dan Fungsi

ImprovedHarmonySearch.java

```
1.  package Controller;
2.  import Entity.Constant;
3.  import Entity.HarmonyMemory;
4.  import Entity.ParameterHSA;
5.  import Entity.ParameterIHSA;
6.  import Entity.ResearchData;
7.  import java.awt.image.BufferedImage;
8.  import java.util.Arrays;
9.  import java.util.Collections;
10. import java.util.Comparator;
11. import java.util.Date;
12. import java.util.Random;
13. import java.util.concurrent.ThreadLocalRandom;
14. import java.util.logging.Level;
15. import java.util.logging.Logger;
16. import javax.swing.JPanel;
17. import javax.swing.JTextArea;
18.
19. /**
20.  *
21.  * @author RazorX
22.  */
23. public class ImprovedHarmonySearch extends Thread{
24.
25.     private BufferedImage originalImage;
26.     private int lowerValue;
27.     private int upperValue;
28.
29.     private int decisionTotal; //jumlah variable keputusan
30.     private int hmSize;
31.     private int cycles;
32.     private double hmConsiderationRate;
33.     private double bwMin;
34.     private double bwMaks;
35.     private double parMin;
36.     private double parMaks;
37.
38.     private HarmonyMemory[] harmonyMemory;
39.     private double par,bw;
40.
41.     private double runningTime;
42.
43.     private int imageWidth, imageHeight;
44.
```

```
45.     Otsu otsu;
46.
47.     JPanel panelOutput; //untuk simulasi
48.     JTextArea info;
49.
50.     //tambahan kriteria
51.     double lastObjectiveValue;
52.     int unspoiledObjectiveValue;
53.
54.     int finalIteration;
55.
56.     public ImprovedHarmonySearch(ResearchData data, int k, JPanel panel,
    JTextArea info)
57.     {
58.
59.       panelOutput = panel;
60.       this.info = info;
61.       originalImage = data.getOriginalImage();
62.       otsu= new Otsu(originalImage);
63.
64.       imageWidth = originalImage.getWidth();
65.       imageHeight = originalImage.getHeight();
66.
67.        Date date = new Date();
68.       double start = date.getTime();
69.       initializeProblemAndParameter(k);
70.       initializeHarmonyMemory();
71.       improviseNewHarmony();
72.       date = new Date();
73.       double end = date.getTime();
74.       runningTime = (end - start)/1000;
75.     }
76.
77.     public double[] getHistogram()
78.     {
79.       return otsu.getHistogram();
80.     }
81.
82.     private void initializeProblemAndParameter(int k)
83.     {
84.       /*problem*/
85.       decisionTotal = k; //jumlah variable keputusan
86.
87.       lowerValue = 0;
88.       upperValue = 255;
89.
90.       /**parameter*/
```

```
91.
92.       hmSize = ParameterIHSA.getHarmonyMemorySize();
93.       cycles = ParameterIHSA.getCycles();
94.       hmConsiderationRate = ParameterIHSA.getHMConsiderationRate();
95.       parMin = ParameterIHSA.getPitchAdjustmentRateMin();
96.       parMaks = ParameterIHSA.getPitchAdjustmentRateMax();
97.       bwMin = ParameterIHSA.getBandwithMin();
98.       bwMaks = ParameterIHSA.getBandwithMax();
99.    }
100.
101.   private void initializeHarmonyMemory()
102.   {
103.      harmonyMemory = new HarmonyMemory[hmSize];
104.               int decisionVariable;
105.      double fitness;
106.
107.      for (int i = 0; i < hmSize; i++)
108.               {
109.                  harmonyMemory[i] = new HarmonyMemory(decisionTotal);
110.
111.         for(int j=0; j<decisionTotal; j++)
112.         {
113.            decisionVariable = getRandom();
114.
115.            harmonyMemory[i].setSolutionVector(j, decisionVariable);
116.         }
117.
118.         int[] harmony = harmonyMemory[i].getSolutionVector();
119.         Arrays.sort(harmony);
120.
121.         fitness = getFitness(harmony);
122.         harmonyMemory[i].setSolutionVector(harmony);
123.         harmonyMemory[i].setFitness(fitness);
124.
125.      }
126.
127.   }
128.
129.   private void improviseNewHarmony(){
130.
131.      //GuiProgress.logProses("Improvisasi dan perbaharui Harmony Memory");
132.
133.      int generation = 0;
134.      double random;
135.      int D1; //index pemilihan lokasi pada harmoni memori secara random
136.      int D2; //nilai variabel keputusan dari vektor solusi
137.      int D3; //nilai variable keputusan berdasarkan PAR dan BW
```

```
138.      int randomDecisionVariable;
139.
140.              lastObjectiveValue = 0;
141.      unspoiledObjectiveValue = 0; //pencatat nilai yang tak berubah
142.
143.      do
144.      {
145.         HarmonyMemory newHarmonyMemoryVector = new
    HarmonyMemory(decisionTotal);
146.
147.         par = getPAR(generation);
148.         bw = getBW(generation);
149.
150.         for(int i=0; i<decisionTotal; i++)
151.         {
152.            random = randProbability(0,1); //random dri 0-1;
153.            //System.out.println("vektor solusi ["+i+"] | random : "+random);
154.            if (random < hmConsiderationRate)
155.            {
156.               D1 = (int) random * hmSize; //pemilihan index pada HM berdasarkan
    hmcr
157.               //System.out.println("random < hmcr("+hmConsiderationRate+")");
158.               random = randProbability(0,1);
159.               //System.out.println("randomPar : "+random);
160.                     if(random < par){
161.                  random = randProbability(0,1);
162.                  D2 = newHarmonyMemoryVector.getSolutionVector(i);
163.                  D3 = adjustedDecisionVariable(D2,i); //dapatkan D3 dan sesuaikan
    agar tidak keluar batas nilai
164.                        newHarmonyMemoryVector.setSolutionVector(i, D3);
165.
166.                        //System.out.println("random < par("+par+")");
167.                //System.out.println("random lgi : "+random+"|D2: "+D2+" |D3:
    "+D3);
168.                     }
169.                     else{
170.                D2 = harmonyMemory[D1].getSolutionVector(i);
171.                newHarmonyMemoryVector.setSolutionVector(i, D2); //simpan ke
    harmony baru (sementara) utk dibandingkan
172.                //System.out.println("random > par("+par+")");
173.                //System.out.println("D2[D1]: "+D2);
174.               }
175.            }
176.            else
177.            {
178.               randomDecisionVariable = getRandom(); //random sesuai batas atas
    dan bawah dan sesuai x atau y.
```

```
179.                 newHarmonyMemoryVector.setSolutionVector(i,
    randomDecisionVariable);
180.
181.                 //System.out.println("variable solusi(D3):"+ randomDecisionVariable);
182.             }
183.         }
184.
185.         int[] harmony = newHarmonyMemoryVector.getSolutionVector();
186.
187.         Arrays.sort(harmony);
188.
189.                 double fitness = getFitness(harmony);
190.                 newHarmonyMemoryVector.setSolutionVector(harmony);
191.         newHarmonyMemoryVector.setFitness(fitness);
192.
193.         int indexOfWorstHarmony =
    getIndexOfWorstHarmony(harmonyMemory);
194.
195.         double newHarmonySolution = newHarmonyMemoryVector.getFitness();
196.                 double worstHarmonySolution =
    harmonyMemory[indexOfWorstHarmony].getFitness();
197.
198.                 if(newHarmonySolution > worstHarmonySolution)
199.                 {
200.                     //harmonyMemory[indexOfWorstHarmony] = new
    HarmonyMemory(decisionTotal);
201.             harmonyMemory[indexOfWorstHarmony] =
    newHarmonyMemoryVector;
202.                     //GuiProgress.logProses("HM baru lebih baik, Gantikan HM
    Buruk");
203.                 }
204.
205.                 //System.out.println(generation+"|"+fitness);
206.                 /**Menampilkan Proses yang berjalan dan update RealTime.
    Menggunakan Thread**/
207.
208.                 doSimulation(harmony,fitness);
209.
210.
    //GuiProgress.prosesBerjalanProgressBar.setValue(generation*100/cycles+1);
211.
212.         generation++;
213.
214.     }while(generation < cycles && !isBestfitnessUnspoiled());
215.
216.     finalIteration = generation;
217. }
```

```
218.
219.    public int getTotalIteration()
220.            {
221.        return finalIteration;
222.            }
223.
224.            private void doSimulation(int[] harmony, double fitness)
225.            {
226.                if(Constant.simulationStatus)
227.                  {
228.                    Controller control = new Controller();
229.                    ThresholdingImage th = new ThresholdingImage();
230.                    BufferedImage tempImage =
      th.getSegmentedImage(originalImage, harmony);
231.            control.updatePanelCitra(tempImage, panelOutput);
232.
233.                    int[] tempBest = getSolution().getSolutionVector();
234.                    double tempBestFitness = getSolution().getFitness();
235.
236.                    info.setText("Threshold terbaik saat ini : \n");
237.                    for(int i=0; i<tempBest.length; i++)
238.                    {
239.                       info.append("th["+(i+1)+"] = "+tempBest[i]+"\n");
240.                    }
241.            info.append("Fitness : "+tempBestFitness+"\n\n");
242.
243.
244.                    info.append("Threshold Percobaan : \n");
245.                    for(int i=0; i<harmony.length; i++)
246.                    {
247.                       info.append("th["+(i+1)+"] = "+harmony[i]+"\n");
248.                    }
249.                    info.append("Fitness : "+tempBestFitness+"\n");
250.
251.            try {
252.                       Thread.sleep(500);
253.                    } catch (InterruptedException ex) {
254.
      Logger.getLogger(ImprovedHarmonySearch.class.getName()).log(Level.SEVERE,
      null, ex);
255.                    }
256.                  }
257.            }
258.
259.            private boolean isBestfitnessUnspoiled()
260.            {
261.        if(Constant.unspoiledStatus)
```

```
262.                {
263.                  //**Kriteria tambahan untuk stop**/
264.                   double currentObjectiveValue =
    harmonyMemory[getIndexOfBestHarmony(harmonyMemory)].getFitness();
265.
266.                   if(currentObjectiveValue==lastObjectiveValue)
267.                   {
268.                      unspoiledObjectiveValue++;
269.                   }
270.                   else
271.         {
272.                      unspoiledObjectiveValue = 0;
273.                   }
274.
275.                   lastObjectiveValue = currentObjectiveValue;
276.
277.                   if(unspoiledObjectiveValue == (int)cycles*0.1) //kalo sudah 10%
278.                   {
279.                      return true;
280.                   }
281.         else
282.                   {
283.                      return false;
284.                   }
285.                }
286.                else
287.                {
288.                   return false;
289.                }
290.             }
291.
292.             private int getIndexOfWorstHarmony(HarmonyMemory[]
    harmonyMemory)
293.             {
294.                int index;
295.                double worst,possibleWorst;
296.
297.                worst = harmonyMemory[0].getFitness();
298.                index = 0;
299.                for(int i=1; i<hmSize; i++)
300.                {
301.         possibleWorst = harmonyMemory[i].getFitness();
302.
303.                   if(possibleWorst < worst)
304.                   {
305.                      worst = possibleWorst;
306.                      index = i;
```

```
307.                     }
308.                 }
309.
310.     return index;
311.   }
312.
313.   private int getIndexOfBestHarmony(HarmonyMemory[] harmonyMemory)
314.   {
315.     int index;
316.     double best,possibleBest;
317.
318.     best = harmonyMemory[0].getFitness();
319.     index = 0;
320.             for(int i=1; i<hmSize; i++)
321.     {
322.                 possibleBest = harmonyMemory[i].getFitness();
323.
324.                 if(possibleBest > best)
325.                 {
326.                     best = possibleBest;
327.                     index = i;
328.                 }
329.             }
330.
331.     return index;
332.         }
333.
334.         public HarmonyMemory getSolution()
335.         {
336.             return harmonyMemory[getIndexOfBestHarmony(harmonyMemory)];
337.         }
338.
339.         public double getRunningTime()
340.         {
341.     return runningTime;
342.         }
343.
344.         // kurang metode step 1, step 4 dan metode step 5;
345.         private int getRandom()
346.         {
347.             int randomDecisionVariable;
348.
349.             randomDecisionVariable =
    randDecisionVariable(lowerValue,upperValue);
350.
351.     return randomDecisionVariable;
352.         }
```

```
353.
354.            private int randDecisionVariable(int lowerValue, int upperValue)
355.            {
356.                //int randomNum =
    ThreadLocalRandom.current().nextInt(lowerValue, upperValue + 1);
357.                int randomNum = (int) Math.floor(Math.random()*(upperValue-
    lowerValue+1)) + lowerValue;
358.                return randomNum;
359.            }
360.
361.   private double randProbability(int min, int max)
362.            {
363.                Random random = new Random();
364.                double value = min + (max - min) * random.nextDouble();
365.
366.                return value;
367.            }
368.
369.            private double getFitness(int[] harmony)
370.            {
371.       double fitness;
372.
373.                fitness = otsu.calculateOtsuVariance(harmony);
374.                //fitness = otsu.getOtsuVariance();
375.
376.                return fitness;
377.            }
378.
379.
380.            private int adjustedDecisionVariable(int D2, int i)
381.   {
382.                int D3;
383.                double random = randProbability(0,1);
384.                if(random < 0.5d){
385.                    D3 = D2 + (int)(random * bw);
386.                }
387.                else{
388.                    D3 = D2 - (int)(random * bw);
389.                }
390.
391.
392.                if(D3 < lowerValue) D3 = lowerValue;
393.                else if (D3 > upperValue) D3 = upperValue;
394.
395.                return D3;
396.            }
397.
```

```
398.             private double getPAR(int generation)
399.             {
400.                double PAR = parMin + ((parMaks - parMin)/cycles) * generation;
401.
402.                return PAR;
403.             }
404.
405.             private double getBW(int generation)
406.             {
407.                double c,BW;
408.
409.                c = Math.log(bwMin/bwMaks) / cycles;
410.                BW = bwMaks*Math.exp(c*generation);
411.
412.       return BW;
413.    }
414.}
```

4.3. Hasil Uji Coba

Tabel 4 dibawah ini menunjukkan hasil dari penerapan metode yang diusulkan terhadap 12 citra lidah setelah dilakukan percobaan sebanyak 35 kali.

TABEL 4

Hasil Penerapan *Multilevel Thresholding* Berbasis *Improved Harmony Search Algorithm* (IHSA) Terhadap 12 Citra Lidah

Citra	th	PSNR
Hongkong		
	2	35.558
1	3	36.074
	4	37.198

	5	38.015
	2	33.745
	3	34.761
2		
	4	35.482
	5	36.013
	2	34.674
	3	35.468
3		
	4	36.035
	5	37.138
	2	34.760
	3	35.657
4		
	4	36.377
	5	37.307
	2	35.109
	3	36.012
5		
	4	36.806
	5	37.635

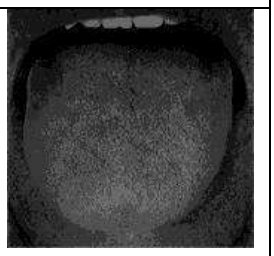
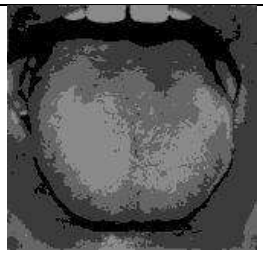
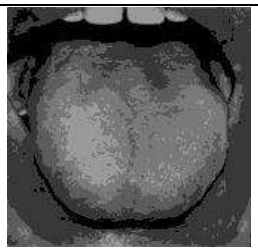
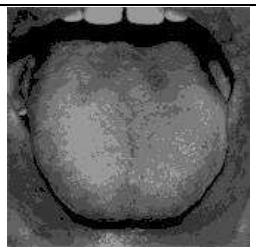
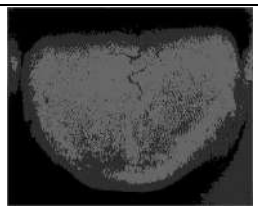
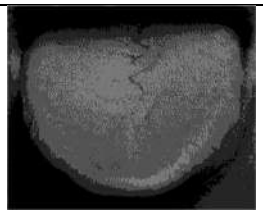
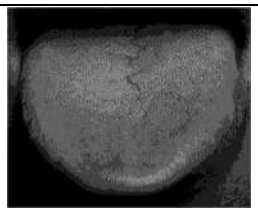
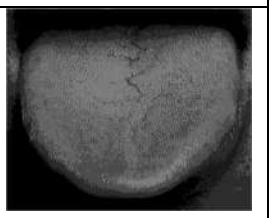
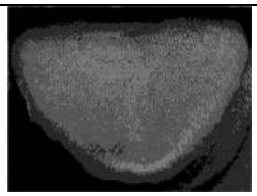
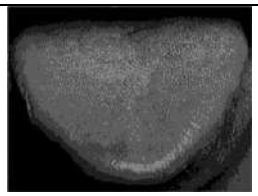
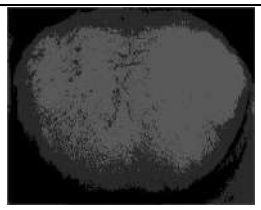
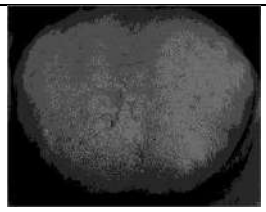
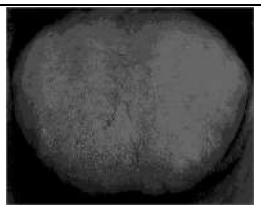
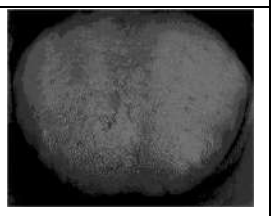
	2	33.673
	3	33.822
6		
	4	34.814
	5	35.560
	2	34.880
	3	35.617
7		
	4	36.061
	5	37.173
	2	31.992
	3	34.857
8		
	4	35.754
	5	36.426
	2	32.037
	3	34.279
9		
	4	35.288
	5	35.936
10	2	31.595

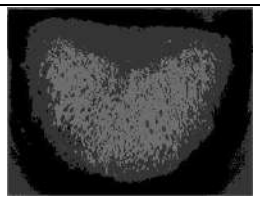
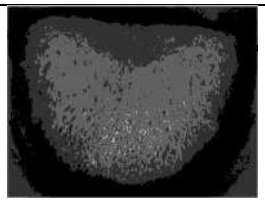
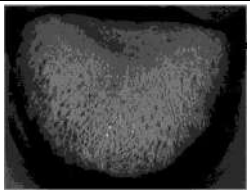
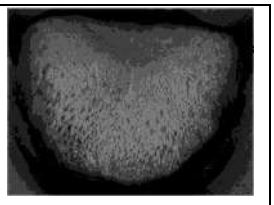
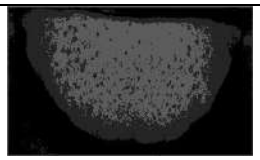
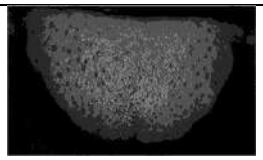
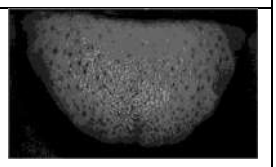
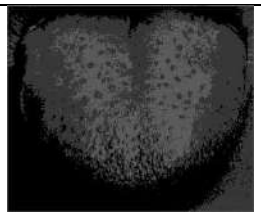
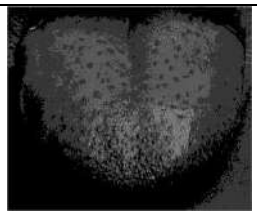
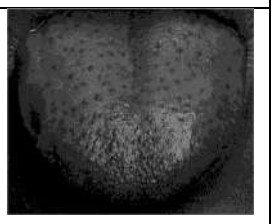
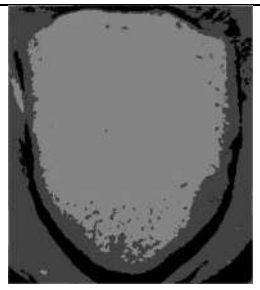
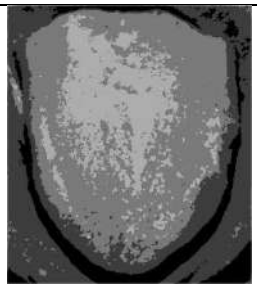
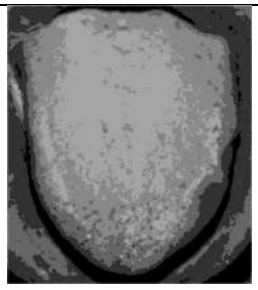
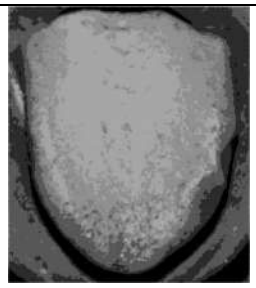
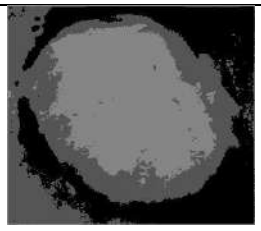
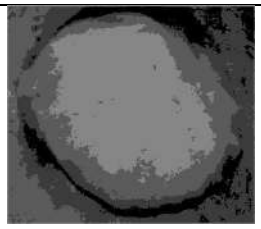
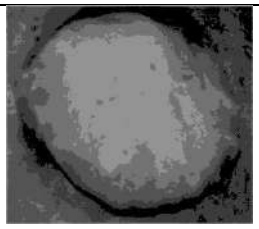
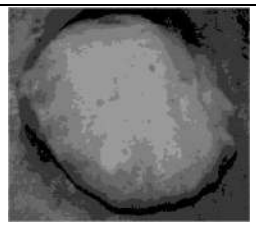
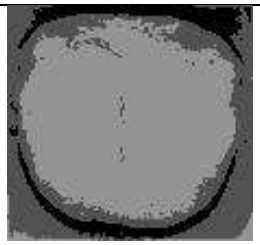
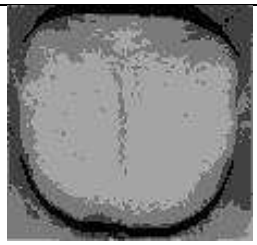
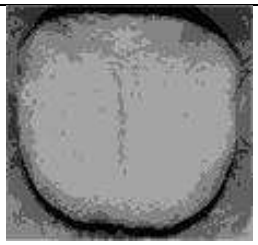
		3	32.908
		4	35.306
		5	35.926
		2	33.115
		3	34.594
11			
		4	35.653
		5	36.082
		2	34.419
		3	35.444
12			
		4	36.032
		5	36.981

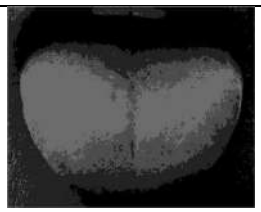
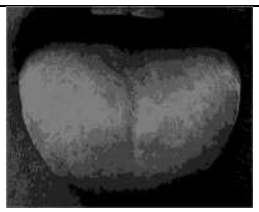
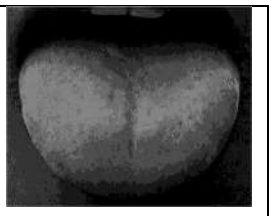
Jika dibandingkan dengan hasil dari penelitian sebelumnya yang telah dipaparkan pada bab tinjauan pustaka maka hasil dari penelitian ini mengalami peningkatan dilihat dari nilai PSNR yang semakin baik. Berdasarkan hasil penerapan *Multilevel Thresholding* Berbasis *Improved Harmony Search Algorithm* (IHSA), maka hasil citra lidah yang telah tersegmentasi dengan menggunakan th=2,3,4,5 dapat dilihat pada Tabel 5 dibawah ini.

TABEL 5

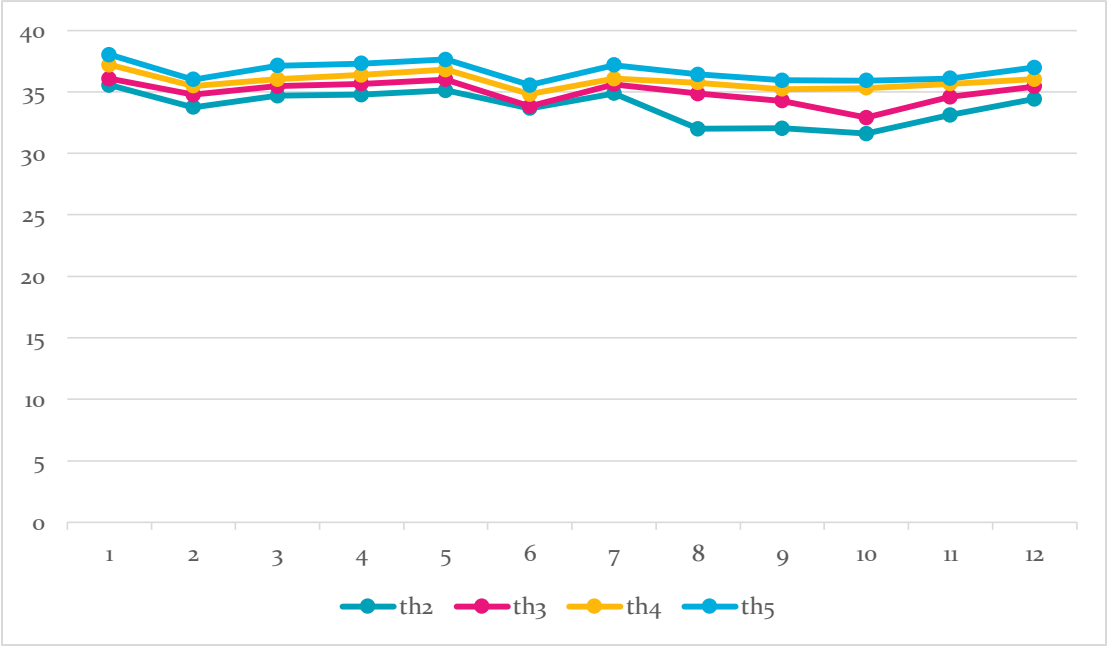
Hasil Citra Lidah Yang Telah Tersegmentasi

Citra Lidah	th=2	th=3	th=4	th=5
Citra 1				
Citra 2				
Citra 3				
Citra 4				
Citra 5				

Citra 6				
Citra 7				
Citra 8				
Citra 9				
Citra 10				
Citra 11				

Citra 12				

Setelah melihat gambar hasil citra lidah yang telah tersegmentasi, maka pada Gambar 1 berikut dapat dilihat bahwa semakin naik threshold, maka nilai PSNR juga akan semakin baik.



Gambar 1 Grafik PSNR 12 Data Citra Lidah Setelah Diterapkan *Multilevel Thresholding* dan *Improved Harmony Search Algorithm* (IHSA)

4.4. Rangkuman

Improved Harmony Search (IHSA) merupakan sebuah metode yang diperkenalkan oleh Mahdavi, Fesanghary, dan Damangir pada tahun 2006, dimana metode tersebut merupakan perkembangan dari metode sebelumnya yakni metode *Harmony Search Algorithm* (HSA). IHSA telah berhasil melakukan segmentasi

lebih baik dibandingkan dengan metode sebelumnya dapat dilihat dari nilai PSNR-nya yang rata-rata lebih dari 30 decibel (dB).

DAFTAR PUSTAKA

[1] D. Oliva, E. Cuevas, G. Pajares, D. Zaldivar, and M. Perez-cisneros, "Multilevel Thresholding Segmentation Based on Harmony Search Optimization," *J. Appl. Math.*, vol. 2013, 2013.

[2] H. Atmaca, M. Bulut, and D. Demir, "HISTOGRAM BASED FUZZY KOHONEN CLUSTERING NETWORK," pp. 1–4, 1996.

[3] N. Jabbar, S. I. Ahson, and M. Mehrotra, "Fuzzy Kohonen Clustering Network for Color Image Segmentation," vol. 3, pp. 254–257, 2011.

[4] M. N. Ahmed, S. M. Yamany, N. Mohamed, A. A. Farag, S. Member, and T. Moriarty, "A Modified Fuzzy C-Means Algorithm for Bias Field Estimation and Segmentation of MRI Data," vol. 21, no. 3, pp. 193–199, 2002.

[5] S. Murugavalli and V. Rajamani, "A HIGH SPEED PARALLEL FUZZY C-MEAN ALGORITHM FOR BRAIN," no. 6, 2006.

[6] R. C. Gonzales and R. E. Wood, *Digital Image Processing*. Prentice Hall, 2005.

[7] D. V. Fernandes, T. Thomas, N. Joseph, and J. Joseph, "Image Segmentation using Fuzzy - Kohonen Algorithm," no. Icmlc, pp. 176–179, 2011.

[8] Q.-K. Pan, P. N. Suganthan, J. J. Liang, and M. F. Tasgetiren, "A local-best harmony search algorithm with dynamic subpopulations," *Eng. Optim.*, vol. 42, no. 2, pp. 101–117, 2010.

[9] B. Alatas, "Chaotic harmony search algorithms," *Appl. Math. Comput.*, 2010.

[10] M. A. Awadallah, A. T. Khader, M. A. Al-Betar, and A. L. aro Bolaji, "Nurse scheduling using Harmony Search," in *Proceedings - 2011 6th International Conference on Bio-Inspired Computing: Theories and Applications, BIC-TA 2011*, 2011.

[11] Z. W. O. O. Geem and J. C. Williams, "Ecological Optimization Using Harmony Search," *Soft Comput.*, 2008.

[12] Z. W. Geem, K. S. Lee, and Y. Park, "Application of Harmony Search to Vehicle Routing," *Am. J. Appl. Sci.*, 2005.

[13] O. M. Alia, R. Mandava, D. Ramachandram, and M. E. Aziz, "Dynamic fuzzy clustering using Harmony Search with application to image segmentation," *2009 IEEE Int. Symp. Signal Process. Inf. Technol.*, pp. 538–543, 2009.

[14] O. M. Alia *et al.*, "Color tongue image segmentation using Fuzzy Kohonen Networks and Genetic Algorithm," *Proc. SPIE---The Int. Soc. Opt. Eng.*, vol. 3962,

no. 1, pp. 176–179, 2010.

[15] M. H. J. Vala and A. Baxi, “A review on Otsu image segmentation algorithm,” *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 2, pp. 387–389, 2013.

[16] W. Kang, Q. Yang, and R. Liang, “The Comparative Research on Image Segmentation Algorithms,” pp. 703–707, 2009.

INDEKS