TUGAS AKHIR – KI141502

SEGMENTASI CITRA PADA ROBOT SEPAK BOLA BERODA MENGGUNAKAN *MULTILAYER NEURAL NETWORK* DAN FITUR WARNA HSV

ALAM AR RAAD STONE
NRP 05111440007001

Dosen Pembimbing I
Dr. Eng. Nanik Suciati, S.Kom., M.Kom.

Dosen Pembimbing II
Dini Adni Navastara, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

TUGAS AKHIR – KI141502

SEGMENTASI CITRA PADA ROBOT SEPAK BOLA BERODA MENGGUNAKAN *MULTILAYER NEURAL NETWORK* DAN FITUR WARNA HSV

ALAM AR RAAD STONE
NRP 05111440007001

Dosen Pembimbing I
Dr. Eng. Nanik Suciati, S.Kom., M.Kom.

Dosen Pembimbing II
Dini Adni Navastara, S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

UNDERGRADUATE THESES – KI141502

IMAGE SEGMENTATION ON WHEELED SOCCER ROBOT USING MULTILAYER NEURAL NETWORK AND HSV COLOR FEATURE

ALAM AR RAAD STONE
NRP 05111440007001

Supervisor I
Dr. Eng. Nanik Suciati, S.Kom., M.Kom.

Supervisor II
Dini Adni Navastara, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

SEGMENTASI CITRA PADA ROBOT SEPAK BOLA BERODA MENGGUNAKAN *MULTILAYER NEURAL NETWORK* DAN FITUR WARNA HSV

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Cerdas dan Visi
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh
ALAM AR RAAD STONE
NRP: 05111440007001

Disetujui oleh Dosen Pembimbing Tugas Akhir

1. Dr. Eng. Nanik Suciati, S.Kom., M.Kom. .....
   NIP: 19710428 199412 2 001 (Pembimbing 1)

2. Dini Adni Navastara, S.Kom., M.Sc. .....
   NIP: 19851017 201504 2 001 (Pembimbing 2)

SURABAYA
JUNI, 2018

[Halaman ini sengaja dikosongkan]

SEGMENTASI CITRA PADA ROBOT SEPAK BOLA BERODA MENGGUNAKAN *MULTILAYER NEURAL NETWORK* DAN FITUR WARNA HSV

Nama Mahasiswa	**: ALAM AR RAAD STONE**
NRP	**: 05111440007001**
Jurusan	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr. Eng. Nanik Suciati, S.Kom., M.Kom.**
Dosen Pembimbing 2	**: Dini Adni Navastara, S.Kom., M.Sc.**

ABSTRAK

Robot sepak bola beroda merupakan robot beroda yang dikembangkan untuk bermain sepak bola secara full autonomous. Robot bertanding secara tim melawan tim lain pada lapangan indoor yang telah disesuaikan ukurannya. Selama pertandingan, tidak diperbolehkan adanya campur tangan manusia. Pada robot, telah di pasang sebuah kamera dengan tujuan untuk menemukan di mana objek penting berada.

Salah satu tahapan sebelum mendeteksi objek adalah segmentasi. Segmentasi citra bertujuan untuk memisahkan objek dengan latar belakang atau membagi citra ke dalam beberapa daerah dengan setiap daerah memiliki kemiripan atribut. Salah satu cara untuk melakukan segmentasi citra adalah dengan mengklasifikasikan tiap piksel pada citra sebagai objek tertentu maupun latar belakang. Pada tugas akhir ini, dilakukan klasifikasi tiap piksel pada ruang warna HSV menjadi 6 kelas. Yaitu: kawan (cyan), lawan (magenta), lapangan (hijau), garis lapangan (putih), bola (orange), dan objek lain (hitam). Proses klasifikasi dilakukan dengan menerapkan model Multilayer Neural Network. Kemudian hasil klasifikasi tersebut digunakan untuk membangun lookup table yang akan digunakan untuk klasifikasi tiap piksel warna secara cepat pada komputer robot.

Dari hasil uji coba dan fine tuning terhadap hyperparameter dan arsitektur pada multilayer neural network, didapatkan nilai

error rata-rata terkecil yaitu 0,16%. Kemudian dari evaluasi hasil segmentasi, diperoleh error rata-rata sebesar 19,37%.

Kata kunci: Segmentasi Citra, Neural Network, Lookup Table

IMAGE SEGMENTATION ON WHEELED SOCCER ROBOT USING MULTILAYER NEURAL NETWORK AND HSV COLOR FEATURE

Student's Name	**: ALAM AR RAAD STONE**
Student's ID	**: 05111440007001**
Department	**: Informatika FTIK-ITS**
First Advisor	**: Dr. Eng. Nanik Suciati, S.Kom., M.Kom.**
Second Advisor	**: Dini Adni Navastara, S.Kom., M.Sc.**

ABSTRACT

The wheeled soccer robot is a wheeled robot developed to play football in full autonomous. Robots compete in teams against other teams on an adjusted indoor field. During the game, no human intervention is allowed. In robots, a camera has been installed in order to find where the important object is.

One of the stages before detecting the object is segmentation. Image segmentation aims to separate objects against a background or divide the image into several areas with each region having similar attributes. One way to segment an image is to classify each pixel in the image as a particular object or background. In this final project, segmentation is done by classifying each pixel on the HSV color space into 6 classes. Namely: friend (cyan), opponent (magenta), field (green), field line (white), ball (orange), and other objects (black). The process of classification is done by applying Multilayer Neural Network model. Then the classification results are used to build a lookup table that will be used for the classification of each color pixel quickly on the robot computer.

From test result and fine tuning to hyperparameter and architecture on multilayer neural network, got the smallest average error value that is 0.16%. Then from the evaluation of the results of segmentation, obtained an average error of 19.37%.

Keywords: Image Segmentation, Neural Network, Lookup Table

KATA PENGANTAR

بسم الله الرحمن الرحيم

Segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul *"***SEGMENTASI CITRA PADA ROBOT SEPAK BOLA BERODA MENGGUNAKAN *MULTILAYER NEURAL NETWORK* DAN FITUR WARNA HSV***"***.

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis dapat belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Departemen Informatika ITS. Dengan Tugas Akhir ini, penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari.

Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT dan Nabi Muhammad SAW.
2. Keluarga penulis yang selalu memberi semangat dan motivasi kepada penulis dalam menjalankan perkuliahan meskipun terpisah oleh jarak.
3. Ibu Dr. Eng. Nanik Suciati, S.Kom., M.Kom., dan ibu Dini Adni Navastara, S.Kom., M.Sc. selaku pembimbing yang telah membantu, membimbing, dan memotivasi penulis dalam menyelesaikan tugas akhir ini.
4. Bapak Radityo Anggoro, S.Kom., M.Sc. selaku koordinator Tugas Akhir, dan segenap dosen Informatika yang telah memberikan ilmunya.
5. Teman-teman mahasiswa Informatika angkatan 2014 yang yang telah membantu, berbagi ilmu, menjaga kebersamaan, dan memberi motivasi kepada penulis.
6. Teman-teman CSSMoRA ITS yang telah banyak membantu dan menemani penulis sejak sebelum masuk masa perkuliahan sampai sekarang.

7. Teman-teman Tim Robot Sepak Bola Beroda IRIS ITS yang telah membantu penulis dalam menyelesaikan tugas akhir.
8. Serta semua pihak yang yang telah turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga, dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2018

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Robot merupakan alat yang dapat membantu manusia baik dengan kontrol oleh manusia maupun secara otomatis dengan program yang telah ditanamkan dahulu pada robot. Untuk memacu perkembangan keilmuan di bidang robotika, maka banyak sekali kontes yang diadakan di seluruh dunia setiap tahunnya dengan tema yang beraneka ragam. Salah satunya adalah Kontes Robot Sepak Bola Indonesia Beroda (KRSBI beroda). Kontes Robot Sepak Bola Indonesia Beroda ini merupakan salah satu kegiatan yang merupakan bagian dari Kontes Robot Indonesia (KRI) sebagai ajang kompetisi rancang bangun dan rekayasa dalam bidang robotika. Kontes Robot Sepakbola Indonesia Beroda diselenggarakan berdasarkan aturan pada lomba internasional *RoboCup Middle Size League* (MSL).

Di antara banyak bidang riset pada Robot Sepak Bola Beroda, visi komputer merupakan salah satu yang paling menantang. Hal ini dikarenakan robot menangkap citra yang memiliki karakteristik perpindahan lingkungan yang cepat. Pemain kawan, pemain lawan, dan bola berpindah dengan cepat, dan sering kali tidak dapat diprediksi. Robot diharuskan menangkap citra lingkungan tersebut menggunakan kamera, dan menemukan di mana objek penting berada. Tidak ada waktu untuk menjalankan algoritma yang kompleks. Semuanya harus diperhitungkan dalam waktu singkat, atau hanya akan menjadi sia-sia. [1]

Salah satu tahapan sebelum mendeteksi objek adalah segmentasi. Segmentasi citra bertujuan untuk memisahkan objek dengan latar belakang atau membagi citra ke dalam beberapa daerah dengan setiap daerah memiliki kemiripan atribut. Salah satu metode segmentasi citra yang sederhana dan

sering digunakan adalah *thresholding*. Metode *thresholding* dilakukan dengan cara mengganti nilai piksel dengan nilai baru apabila nilai piksel lama berada di antara nilai konstan tertentu. Metode ini berlaku pada citra dengan skala keabuan. Pada citra berwarna, segmentasi dilakukan dengan cara memisahkan tiap komponen warna pada citra, kemudian dilakukan *thresholding* secara terpisah, lalu digabungkan kembali menggunakan operator *and*. Model ruang warna yang sering digunakan di dalam proses *thresholding* pada citra berwarna adalah ruang warna HSV. Bagaimanapun, terkadang sulit untuk menentukan nilai konstan yang akan digunakan pada proses *thresholding*.

Cara lain untuk melakukan segmentasi citra adalah dengan mengklasifikasikan tiap piksel pada citra sebagai objek tertentu maupun latar belakang. Dalam tugas akhir ini, akan dilakukan klasifikasi tiap piksel pada ruang warna HSV menjadi 6 kelas. Yaitu: kawan, lawan, lapangan, garis lapangan, bola, dan objek lain. Proses klasifikasi dilakukan dengan menerapkan model *Multilayer Neural Network* yang dilatih pada komputer lain menggunakan data yang diambil dari Robot Sepak Bola Beroda pada waktu yang beragam. Kemudian hasil klasifikasi tersebut digunakan untuk membangun *Lookup Table* yang akan digunakan untuk klasifikasi tiap piksel warna secara cepat pada komputer robot.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara melakukan segmentasi pada citra robot sepak bola beroda?
2. Bagaimana cara membangun model *Multilayer Neural Network* untuk mengklasifikasikan tiap piksel pada ruang warna HSV ke dalam 6 kelas yang telah ditentukan?
3. Bagaimana cara mengevaluasi kinerja model *Multilayer Neural Network* yang telah dibuat?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Data yang digunakan adalah dataset milik Tim Robot Sepak Bola Institut Teknologi Sepuluh Nopember. Dataset berupa video RGB berukuran 480 x 640 piksel yang ditangkap menggunakan kamera Logitech C922.
2. Implementasi program dilakukan pada lingkungan komputer desktop menggunakan bahasa *python* untuk pelatihan *Multilayer Neural Network*, C++ untuk implementasi pada robot, serta pustaka pendukung *OpenCV*.
3. Tiap piksel pada ruang warna HSV diklasifikasikan ke dalam 6 kelas. Yaitu: kawan (cyan), lawan (magenta), lapangan (hijau), garis lapangan (putih), bola (orange), dan objek lain (hitam).
4. Keluaran akhir dari program adalah citra yang telah dilakukan segmentasi berdasarkan *lookup table* yang dibangun dari hasil klasifikasi tiap piksel pada ruang warna HSV menggunakan *multilayer neural network*.

1.4 Tujuan

Tujuan dari tugas akhir ini adalah untuk membangun model *Multilayer Neural Network* yang dapat mengklasifikasikan tiap piksel pada ruang warna HSV ke dalam 6 kelas yang telah ditentukan, serta membangun program untuk melakukan segmentasi citra menggunakan *Lookup Table* yang dibangun dari hasil klasifikasi seluruh piksel pada ruang warna HSV.

1.5 Manfaat

Tugas akhir ini diharapkan dapat bermanfaat bagi perkembangan keilmuan pada bidang robot sepak bola, serta dapat dikembangkan lebih lanjut sehingga hasil segmentasi dapat diolah menjadi data yang berguna bagi robot sepak bola beroda.

1.6 Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.
   Tahap awal untuk memulai pengerjaan Tugas Akhir adalah penyusunan proposal Tugas Akhir. Proposal Tugas Akhir yang diajukan memiliki gagasan yang sama dengan Tugas Akhir ini, yaitu segmentasi citra pada robot sepak bola beroda menggunakan *multilayer neural network* dan ruang warna HSV.

2. Studi literatur
   Pada tahap ini dilakukan pencarian, pembelajaran dan pemahaman informasi dan literatur yang diperlukan untuk implementasi *multilayer neural network.* Informasi dan literatur didapatkan dari literatur jurnal ilmiah dan sumber-sumber informasi lain yang berhubungan.

3. Analisis dan desain perangkat lunak
   Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini.

4. Implementasi perangkat lunak

   Implementasi merupakan tahap membangun rancangan program yang telah dibuat. Pada tahapan ini direalisasikan apa yang terdapat pada tahapan sebelumnya, sehingga menjadi sebuah program yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi

   Pada tahapan ini dilakukan uji coba pada data yang telah dikumpulkan. Pengujian dan evaluasi akan dilakukan dengan menggunakan bahasa *Python*. Tahapan ini dimaksudkan untuk mengevaluasi kesesuaian data dan program serta mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.

   Pada tahapan ini disusun buku yang memuat dokumentasi mengenai pembuatan serta hasil dari implementasi perangkat lunak yang telah dibuat.

1.7 Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

Bab I Pendahuluan

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

Bab II Dasar Teori

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan untuk mendukung pembuatan Tugas Akhir ini.

Bab III Perancangan Perangkat Lunak

Bab ini berisi tentang desain sistem yang disajikan dalam bentuk diagram ataupun *pseudocode*.

Bab IV Implementasi

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *code* yang digunakan untuk proses implementasi.

Bab V Uji Coba Dan Evaluasi

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

Bab VI Kesimpulan Dan Saran

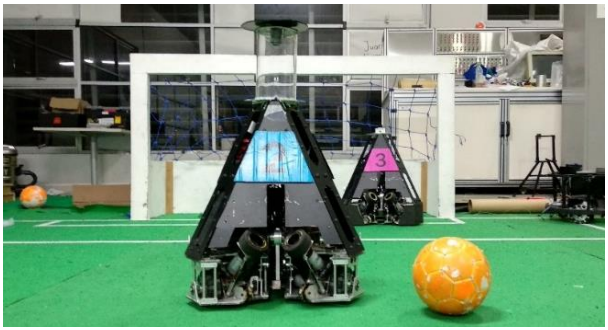
Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi penjelasan teori dan materi yang berkaitan dengan algoritma yang diajukan. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan perangkat lunak.

2.1 Robot Sepak Bola Beroda

Robot sepak bola beroda merupakan robot beroda yang dikembangkan untuk bermain sepak bola secara *full autonomous*. Robot bertanding secara tim melawan tim lain pada lapangan *indoor* yang telah disesuaikan ukurannya. Selama pertandingan, tidak diperbolehkan adanya campur tangan manusia.



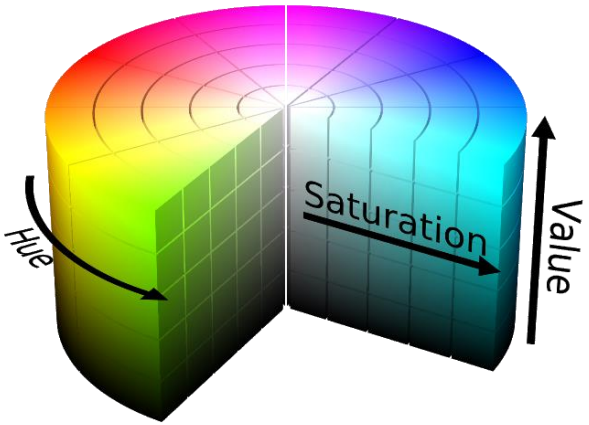
Gambar 2.1 Robot Sepak Bola Beroda ITS tahun 2017

Kontes Robot Sepak Bola Indonesia Beroda merupakan salah satu kegiatan yang merupakan bagian dari Kontes Robot Indonesia (KRI) sebagai ajang kompetisi rancang bangun dan rekayasa dalam bidang robotika. Kontes Robot Sepakbola Indonesia Beroda diselenggarakan berdasarkan aturan pada lomba internasional *RoboCup Middle Size League* (MSL).

Dalam Kontes Robot Sepak Bola Indonesia Beroda, mahasiswa dituntut untuk bisa mengembangkan kemampuan dalam bidang mekanika, manufaktur, elektronika, pemrograman, kecerdasan buatan, pengolahan citra, komunikasi digital, strategi, kemampuan meneliti dan menulis artikel, sekaligus diperlukan pengembangan ke arah disiplin, toleransi, sportifitas, kerjasama, saling menghargai, kontrol emosi dan kemampuan *softskill* lainnya. [2]

2.2 Ruang Warna HSV

HSV merupakan singkatan dari *Hue*, *Saturation*, dan *Value*. Ruang warna HSV sering digunakan pada visi komputer karena memisahkan antara *luma* atau intensitas citra dari *chroma* atau informasi warna [3]. Dengan menggunakan ruang warna ini, dimungkinkan untuk melakukan seleksi warna tertentu tanpa melihat intensitas cahaya yang jatuh pada objek tersebut. Hal ini sangat berguna terutama pada seleksi warna objek yang berada di luar ruangan dengan intensitas cahaya yang beraneka ragam.



Gambar 2.2 Geometri tabung ruang warna HSV [3]

Citra warna HSV direpresentasikan dalam geometri silinder seperti ditunjukkan pada Gambar 2.2. *Hue* dinyatakan sebagai sudut tabung, *saturation* dinyatakan sebagai jari-jari tabung dan *value* dinyatakan oleh tinggi tabung.

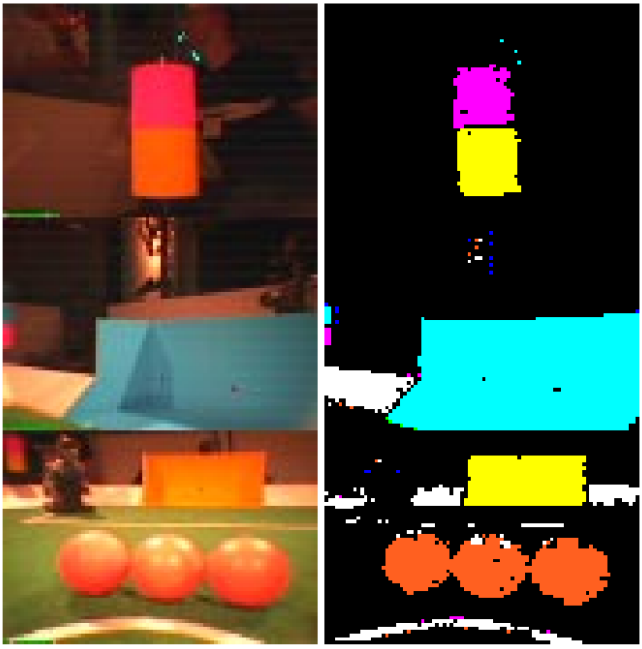
Proses konversi citra dari ruang warna RGB ke HSV dilakukan dengan cara [4]:

1. $V \leftarrow \max(R, G, B)$
2. $S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{v} & \text{if } V \neq 0 \\ 0 & otherwise \end{cases}$
3. $H \leftarrow \begin{cases} 60(G-B)/(V - min(R,G,B)) & if\ V = R \\ 120 + 60(B-R)/(V - min(R,G,B)) & if\ V = G \\ 240 + 60(R-G)/(V - min(R,G,B)) & if\ V = B \end{cases}$
4. Jika $H < 0$, maka $H \leftarrow H + 360$
5. $V \leftarrow 255V$
6. $S \leftarrow 255S$
7. $H \leftarrow H/2$

2.3 Segmentasi

Segmentasi citra merupakan proses yang ditujukan untuk mendapatkan objek-objek yang terkandung di dalam citra atau membagi citra ke dalam beberapa daerah dengan setiap objek atau daerah memiliki kemiripan atribut. Citra dengan satu objek saja dapat dibedakan dari latar belakangnya, sedangkan citra yang mengandung sejumlah objek, segmentasi menjadi lebih kompleks [5]. Segmentasi dilakukan sebagai langkah awal untuk klasifikasi objek pada citra.

Contoh segmentasi citra dapat dilihat pada Gambar 2.3. Citra sebelah kiri merupakan citra yang diambil oleh robot sepak bola pada saat RoboCup-99. Sedangkan citra sebelah kanan merupakan citra hasil segmentasi.



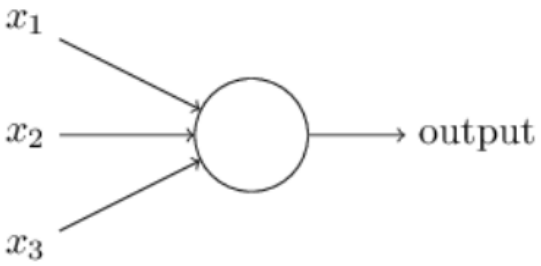
Gambar 2.3 Contoh Segmentasi Citra [6]

2.4 *Multilayer Neural Network*

Multilayer Neural Network atau *Multilayer Perceptron* merupakan salah satu dari jenis *Feedforward Artificial Neural Network*. *Multilayer Neural Network* terdiri dari setidaknya tiga *layer*. *Layer* pertama disebut dengan *input layer*, *layer* terakhir disebut dengan *output layer*, dan *layer* di antara *input layer* dan *output layer* disebut dengan *hidden layer*. Setiap *neuron* selain pada *input layer* memiliki *transfer function* atau *activation function*. *Activation function* yang biasa digunakan pada *Multilayer Neural Network* adalah *sigmoid*.

2.4.1 *Perceptron*

Perceptron dikembangkan pada tahun 1950 – 1960 oleh Frank Rosenblatt, yang terinspirasi oleh Warren McCulloch dan Walter Pitts. Perceptron mengambil input x1, x2, ..., dan mengeluarkan sebuah output biner. [7]



Gambar 2.4 Perceptron [7]

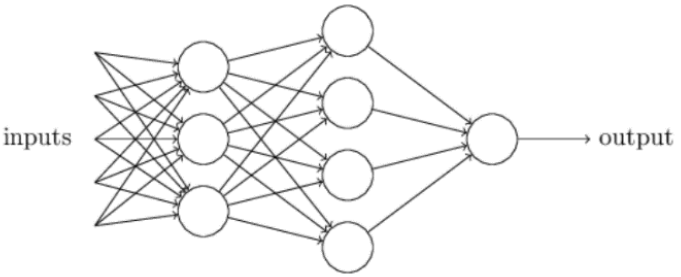
Pada Gambar 2.4, *perceptron* memiliki tiga input, *x1*, *x2*, *x3*. Secara umum, *perceptron* bisa memiliki input lebih banyak maupun lebih sedikit. Rosenblatt mengusulkan aturan sederhana untuk menghitung output. Rosenblatt memperkenalkan *weight*, *w1*, *w2*, ..., bilangan real yang menunjukkan tingkat pengaruh input terhadap output. Output, ditentukan dengan Persamaan 2.1.

$$\text{output} = \begin{cases} 0 \; if \; \sum_j w_j x_j \leq threshold \\ 1 \; if \; \sum_j w_j x_j > threshold \end{cases} \quad \textbf{(2.1)}$$

Pada Persamaan 2.1, x_j merupakan input ke *j*, sedangkan w_j merupakan *weight* ke *j*. Seperti *weight*, *threshold* merupakan bilangan real yang merupakan parameter dari *perceptron*. Dengan berbagai variasi *weight* dan *threshold*,

pembaca dapat memperoleh model pengambilan keputusan yang berbeda.

Tentunya, *perceptron* bukan merupakan model yang sempurna dari pengambilan keputusan pada manusia. Tetapi Gambar 2.5 dapat mengilustrasikan bagaimana *perceptron* dapat mempertimbangkan berbagai macam kemungkinan untuk membuat keputusan.



Gambar 2.5 Multilayer Perceptron [7]

Pada Gambar 2.5. Layer (kolom) pertama, membuat 3 keputusan sederhana. Kemudian, setiap *perceptron* dari layer ke dua membuat keputusan dengan mempertimbangkan hasil dari layer pertama. Dengan cara ini, *perceptron* dari layer ke dua dapat membuat keputusan yang lebih kompleks dari layer pertama. Dan kemudian, keputusan yang lebih kompleks lagi dapat dibuat oleh *perceptron* pada layer ke tiga. Dengan cara ini, *perceptron* pada banyak layer dapat membentuk keputusan yang canggih.

Kondisi $\sum_j w_j x_j > threshold$ itu tidak praktis untuk digunakan. Maka, dibuatlah dua perubahan notasi agar lebih sederhana. Yang pertama, adalah dengan menuliskan $\sum_j w_j x_j$ sebagai dot product. $w \cdot x \equiv \sum_j w_j x_j$, w merupakan vektor *weight* dan x merupakan vektor input. Perubahan ke dua, dengan memindahkan threshold ke sisi lain dari pertidaksamaan, dan menggantinya dengan yang kita kenal dengan *bias*, $b \equiv -threshold$. Aturan perceptron dapat ditulis

ulang dengan Persamaan 2.2, dimana x merupakan input, w merupakan *weight*, dan b merupakan *bias*.

$$output = \begin{cases} 0 \; if \; \sum_j w \cdot x + b \leq 0 \\ 1 \; if \; \sum_j w \cdot x + b > 0 \end{cases} \quad \textbf{(2.2)}$$

2.4.2 *Sigmoid Neuron*

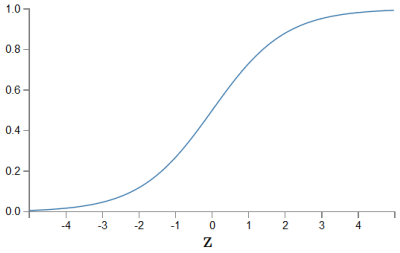
Sama seperti perceptron, *sigmoid neuron* memiliki input, *x1*, *x2*, ..., *weight*, *w1*, *w2*, ..., dan *bias*, *b*. Tetapi output dari *sigmoid neuron* bukan 0 atau 1, melainkan $\sigma(w \cdot x + b)$, dimana σ disebut dengan fungsi sigmoid, dan didefinisikan dengan Persamaan 2.3. Pada Persamaan 2.3, σ merupakan fungsi sigmoid, dan z merupakan perkalian antara *w* dan *x*, ditambah *b*.

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \quad \textbf{(2.3)}$$

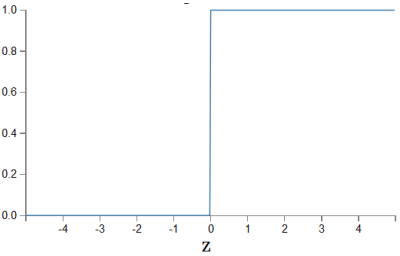
Untuk memahami persamaan antara *perceptron* dan *sigmoid neuron*, misal $z \equiv w \cdot x + b$ merupakan bilangan positif yang besar, maka $e^{-z} \approx 0$, dan $\sigma(z) \approx 1$. Dengan kata lain, ketika $z \equiv w \cdot x + b$ merupakan bilangan positif yang besar, maka output dari sigmod neuron mendekati 1, seperti *perceptron*. Misal $z \equiv w \cdot x + b$ merupakan bilangan negatif yang besar, maka $e^{-z} \to \infty$, dan $\sigma(z) \approx 0$. Dengan kata lain, ketika $z \equiv w \cdot x + b$ merupakan bilangan negatif yang besar, maka output dari sigmod neuron mendekati 0, juga seperti *perceptron*. Bentuk dari fungsi *sigmoid* merupakan versi lebih halus dari *step function*.

Perbedaan besar antara *perceptron* dan *sigmoid neuron* yaitu *sigmoid neuron* tidak hanya menghasilkan output 0 atau 1. *Sigmoid neuron* dapat menghasilkan output bilangan real

antara 0 dan 1, sehingga angka yang dihasilkan dapat bervariasi seperti 0,093, dan 0,997.



Gambar 2.6 Grafik Sigmoid Function [7]



Gambar 2.7 Grafik Step Function [7]

2.4.3 *Gradient Descent*

Tujuan pelatihan *neural network* adalah untuk menemukan *weight* dan *bias* yang sesuai sehingga output dari *neural network* sesuai dengan yang diharapkan dengan cara meminimalkan *cost*. Untuk menghitung seberapa dekat hasil *neural network* dengan data sebenarnya, maka didefinisikanlah *cost function*.

$$C(w,b) \equiv \frac{1}{2n}\sum_{x}\|y(x)-a\|^2 \qquad \textbf{(2.4)}$$

Pada Persamaan 2.4, *w* merupakan *weight pada neural network*, *b* untuk *bias*, *n* merupakan jumlah *data training*, *y(x)*

merupakan vektor output dari input x pada data sebenarnya, dan *a* merupakan vektor output dari input x pada *neural network*. C merupakan *quadratic cost function*, atau terkadang disebut dengan *mean squared error* (MSE). Dari *bentuk quadratic cost function*, dapat diketahui bahwa *C(w, b)* adalah non-negatif, karena setiap penjumlahan selalu non-negatif. Algoritma pelatihan dikatakan baik apabila dapat menemukan *weight* dan *bias* yang menghasilkan $C(w,b) \approx 0$. Sebaliknya, apabila *C(w, b)* menghasilkan nilai yang besar maka berarti *y(x)* tidak dekat dengan output *a* pada banyak data input.

Gradient descent merupakan algoritma optimasi yang digunakan untuk menemukan nilai dari parameter (koefisien) dari fungsi *(f)* dengan meminimalisasi *cost*. *Gradient descent* digunakan ketika parameter tidak dapat dikalkulasi secara analitis dan harus dicari dengan algoritma optimasi. Penerapan rumus gradient descent untuk melatih neural network dapat dilihat pada Persamaan 2.5-2.7.

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial wk} \tag{2.5}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial bl} \tag{2.6}$$

$$C = \frac{1}{\mathrm{n}} \sum_x C_x \tag{2.7}$$

Dimana η merupakan *learning rate*, C_x merupakan *cost* dari satu contoh data training, *C* merupakan rata-rata dari C_x, *n* merupakan jumlah data, w_k merupakan *weight* lama, w'_k merupakan *weight* baru, b_l merupakan *bias* lama, dan b'_l merupakan *bias* baru. Dengan secara berulang-ulang memperbarui *weight* dan *bias*, diharapkan dapat meminimalisasi *cost*. Pada prakteknya, ketika jumlah data

training sangat besar, perhitungan ini dapat memakan waktu yang lama, sehingga proses pelatihan berjalan lambat.

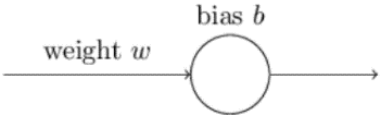
Kemudian, ditemukanlah ide yang bernama *stochastic gradient descent* yang digunakan untuk mempercepat pelatihan. Idenya ialah dengan memperkirakan gradient dengan cara menghitung berdasarkan sampel kecil data training yang dipilih secara random. Dengan merata-rata sampel kecil ini, proses pelatihan dapat berjalan lebih cepat. *Stochastic gradient descent* bekerja dengan memilih secara random sejumlah *m data training*. *Data training* terpilih *x1*, *x2*, ..., *xm*, disebut dengan *mini-batch*. Kemudian lakukan perhitungan sampai *data training* terpilih habis, atau yang biasa disebut dengan menyelesaikan satu *epoch* pelatihan. Lalu, mulai lagi dengan *epoch* selanjutnya. Persamaannya dapat dituliskan seperti pada Persamaan 2.8 dan Persamaan 2.9. Pada kedua persamaan tersebut, terdapat *m* yang merupakan jumlah data pada *mini-batch*.

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial wk} \tag{2.8}$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{x_j}}{\partial bl} \tag{2.9}$$

2.4.4 *Cross-Entropy Cost Function*

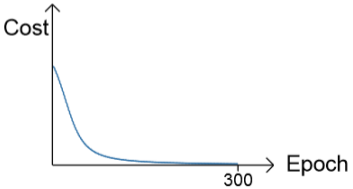
Secara ideal, *neural network* diharapkan dapat belajar secara cepat dari *error*. Namun pada praktiknya, inisialisasi *weight* dan *bias* yang random berpengaruh terhadap waktu pelatihan *neural network*. Sebagai perbandingan, Gambar 2.8 merupakan sebuah *neuron* yang akan dilatih untuk mengeluarkan output 0 apabila input 1.



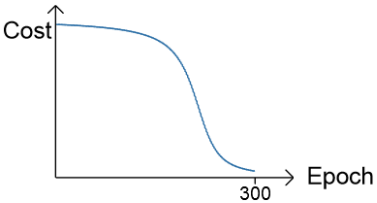
Gambar 2.8 Neuron [7]

Misalkan *weight* awal (*w*) = 0,6, *bias* awal (*b*) = 0,9, *learning rate* (*η*) = 0,15, dan fungsi *cost* yang digunakan adalah *quadratic cost function*. Maka grafik *cost* terhadap *epoch* yang dihasilkan dapat dilihat pada Gambar 2.9.

Dapat dilihat bahwa *neuron* dapat memperbaiki *weight*, dan *bias* secara cepat. Namun, berbeda apabila *weight* awal (*w*) = 2,0, dan *bias* awal (*b*) = 2,0. Dengan *learning rate* dan *cost function* yang sama, grafik *cost* terhadap *epoch* yang dihasilkan dapat dilihat pada Gambar 2.10.



Gambar 2.9 Grafik Cost Terhadap Epoch 1 [7]



Gambar 2.10 Grafik Cost Terhadap Epoch 2 [7]

Pada Gambar 2.10 terlihat bahwa proses pelatihan berjalan lebih lambat. Hal tersebut dapat diatasi dengan mengganti *quadratic cost function* dengan *cost function* lain, yang dikenal dengan *cross-entropy cost function*. *Cross-*

entropy cost pada satu *neuron* dapat dihitung dengan menggunakan Persamaan 2.10.

$$C = -\frac{1}{n}\sum_{x}[y\ ln\ a + (1-y)\ ln(1-a)] \qquad (2.10)$$

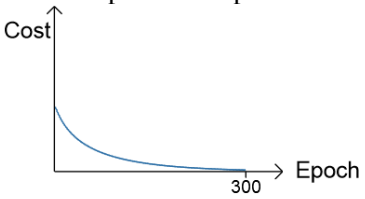
Dimana *n* adalah jumlah *data training*, *x* adalah input, *y* adalah output yang diharapkan, dan *a* adalah output sekarang. Sedangkan *cross-entropy cost* pada banyak *neuron* pada banyak *layer* dapat dihitung dengan Persamaan 2.11.

$$C = -\frac{1}{n}\sum_{x}\sum_{j}[y_j\ ln\ a_j^L + (1-y_j)\ ln(1-a_j^L)] \qquad (2.11)$$

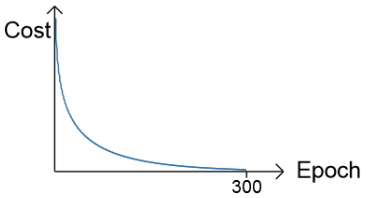
Dimana y_j adalah output yang diharapkan pada neuron *j* dari input *x*, a_j^L adalah output sekarang pada neuron *j* dari input *x*, *n* merupakan jumlah *data training*, *x* merupakan jumlah input, dan *j* adalah jumlah *neuron*.

Dari contoh neuron pada Gambar 2.8, misalkan *weight* awal (*w*) = 0,6, *bias* awal (*b*) = 0,9, dan fungsi *cost* yang digunakan adalah *cross-entropy cost function*. Maka grafik *cost* terhadap *epoch* yang dihasilkan dapat dilihat pada Gambar 2.11.

Sedangkan apabila *weight* awal (*w*) = 2,0, dan *bias* awal (*b*) = 2,0. Dengan *cost function* yang sama, grafik *cost* terhadap *epoch* yang dihasilkan dapat dilihat pada Gambar 2.12.



Gambar 2.11 Grafik Cost Terhadap Epoch 3 [7]



Gambar 2.12 Grafik Cost Terhadap Epoch 4 [7]

Untuk melihat apa yang terjadi pada grafik, maka *learning rate* yang digunakan pada *quadratic cost function*, dan *cross-entropy cost function* tidaklah sama. Pada Gambar 2.11, dan Gambar 2.12, *learning rate* yang digunakan adalah 0,005.

Dengan berbedanya *learning rate* yang digunakan, bukan berarti Gambar 2.11 dan Gambar 2.12 menjadi tidak berarti. Poin yang didapat dari grafik pada Gambar 2.11 dan 2.12 bukan kecepatan absolut pelatihan, akan tetapi bagaimana kecepatan pelatihan berubah. Yang mana, hal tersebut tidak bergantung dengan bagaimana *learning rate* diatur.

2.4.5 *L2 Regularization*

Meningkatkan jumlah *data training* merupakan salah satu cara untuk mengurangi *overfitting*. Ada cara lain yang dapat dilakukan untuk mengurangi *overfitting* meskipun data yang dimiliki tetap, cara ini dikenal dengan nama *regularization*. Salah satu yang sering digunakan ialah *weight decay* atau *L2 regularization*. Ide dari *L2 regularization* ialah dengan menambahkan *extra term* pada *cost function*, yaitu *regularization term*. Persamaan *regularized cross-entropy* dapat dilihat pada Persamaan 2.12.

$$C = -\frac{1}{n}\sum_{x}\sum_{j}\left[y_j \, ln \, a_j^L + \left(1 - y_j\right) ln\left(1 - a_j^L\right)\right] + \frac{\lambda}{2n}\sum_{w} w^2 \tag{2.12}$$

Term pertama merupakan persamaan umum untuk *cross-entropy*. Sedangkan untuk *term* kedua, dimana $\lambda > 0$ merupakan *regularization parameter*, w adalah *weight*, dan *n* adalah jumlah *data training*. Persamaan 2.13 dapat ditulis ulang menjadi Persamaan 2.13. Pada Persamaan 2.13, *C* merupakan *cost function*, C_0 merupakan *unregularized cost function*, *n* merupakan jumlah *data training*, λ merupakan *regularization parameter* (*lambda*), dan *w* merupakan *weight*.

$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2 \tag{2.13}$$

Jenis *regularization* lain yang telah dikembangkan ialah *L1 regularization*. Persamaan *L1 regularization* dapat dilihap pada Persamaan 2.14. Pada Persamaan 2.14, *C* merupakan *cost function*, C_0 merupakan *unregularized cost function*, *n* merupakan jumlah *data training*, λ merupakan *regularization parameter* (*lambda*), dan *w* merupakan *weight*.

$$C = C_0 + \frac{\lambda}{n}\sum_w |w| \tag{2.14}$$

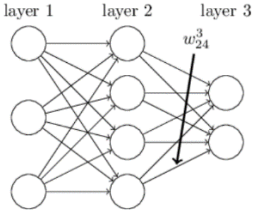
Persamaan 2.14 mirip dengan *L2 regularization*. Namun, tentu saja sifat yang dihasilkan dari *L1 regularization* berbeda dengan *L2 regularization*.

2.4.6 *Backpropagation*

Algoritma *backpropagation* mulanya diperkenalkan pada tahun 1970-an. Namun algoritma ini belum terlalu diapresiasi, sampai dengan munculnya paper oleh David Rumelhart, Geoffrey Hinton, dan Ronald Williams pada tahun 1986. Paper tersebut menunjukkan beberapa contoh *neural network* dimana *backpropagation* bekerja jauh lebih cepat

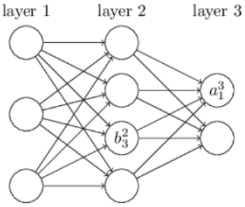
daripada pendekatan lain yang telah diperkenalkan terlebih dahulu. [6]

Untuk menyeragamkan penulisan, maka disepakati beberapa aturan notasi. *Weight* w_{jk}^{l} merupakan *weight* koneksi dari *neuron* k pada *layer* $(l-1)$ ke *neuron j* pada *layer l*. Gambar 2.13 menunjukkan *weight* koneksi dari *neuron* 4 pada *layer* 2 ke *neuron* 2 pada *layer* 3.



Gambar 2.13 Weight [7]

Kemudian b_j^l merupakan *bias neuron j* pada *layer l*, dan a_j^l merupakan aktivasi *neuron j* pada *layer l*. Gambar 2.14 menunjukkan contoh penggunaan notasi *bias* dan aktivasi.



Gambar 2.14 Bias dan Aktivasi [7]

Algoritma *backpropagation* secara singkat dapat ditulis sebagai berikut:

1. **Input** *x*: Sesuaikan aktivasi a^1 pada *input layer* dengan input *x*.
2. **Feedforward**: Untuk tiap *layer l* = 2, 3, ..., L, hitung $z^l = w^l a^{l-1} + b^l$, dan $a^l = \sigma(z^l)$.
3. **Output error** δ^L: Hitung vektor $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error**: Untuk tiap layer l = L – 1, L – 2, ..., 2, hitung $\delta^{\boldsymbol{l}} = \left(\left(w^{l+1}\right)^{\boldsymbol{T}}\delta^{\boldsymbol{l+1}}\right) \odot \sigma'(z^l)$.
5. **Output**: *Gradient* dari *cost function* didapatkan dari persamaan $\frac{\partial C}{\partial \mathrm{w}^{\mathrm{l}}_{\mathrm{jk}}} = \mathrm{a}^{\mathrm{l}-1}_{\mathrm{k}}\delta^{\mathrm{l}}_{\mathrm{j}}$ dan $\frac{\partial C}{\partial \mathrm{b}^{\mathrm{l}}_{\mathrm{j}}} = \delta^{\mathrm{l}}_{\mathrm{j}}$.

Dapat dilihat mengapa algoritma tersebut disebut dengan *backpropagation*. *Error* vektor δ^{l} dihitung mundur, dimulai dari *layer* terakhir. Seperti yang telah dideskripsikan sebelumnya, algoritma *backpropagation* menghitung *gradient* dari *cost function* pada 1 data training, $C = C_x$. Pada praktiknya, algoritma *backpropagation* dikombinasikan dengan algoritma *learning* seperti *stochastic gradient descent*, yang menghitung *gradient* untuk banyak data training.

Misal, diberikan *mini-batch* sejumlah m data training, algoritma berikut merupakan penerapan *gradient descent learning* berdasarkan *mini-batch*:

1. **Input**: kumpulan data training.
2. **Untuk setiap data training** *x*:
   2.1. Sesuaikan aktivasi a^1 pada *input layer* dengan input *x*.
   2.2. **Feedforward**: Untuk tiap *layer* l = 2, 3, ..., L, hitung $z^{x,l} = w^l a^{x,l-1} + b^l$, dan $a^{x,l} = \sigma\left(z^{x,l}\right)$.
   2.3. **Output error** $\delta^{\boldsymbol{x,L}}$: Hitung vektor $\delta^{\boldsymbol{x,L}} = \nabla_{\boldsymbol{a}} C_x$.
   2.4. **Backpropagate the error**: Untuk tiap *layer* l = L – 1, L – 2, ..., 2, hitung $\delta^{\boldsymbol{x,l}} = \left(\left(w^{l+1}\right)^{\boldsymbol{T}}\delta^{\boldsymbol{x,l+1}}\right) \odot \sigma'(z^{x,l})$.
3. **Gradient descent**: untuk tiap l = L, L – 1, ..., 2 perbarui *weight* dengan aturan $w^l \rightarrow w^l - \frac{\eta}{m}\sum_x \delta^{\boldsymbol{x,l}}\left(a^{x,l-1}\right)^{\boldsymbol{T}}$, dan *bias* dengan aturan $b^l \rightarrow b^l - \frac{\eta}{m}\sum_x \delta^{\boldsymbol{x,l}}$ (tanpa parameter regularisasi). Sedangkan, apabila menggunakan regularisasi L2, maka rumus untuk memperbarui *weight* ialah $w^l \rightarrow \left(1 -\right.$

$\frac{\eta\lambda}{n}\Big) w^l - \frac{\eta}{m}\sum_x \delta^{x,l}\left(a^{x,l-1}\right)^T$, dan rumus untuk memperbarui *bias* tetap sama seperti tanpa regularisasi.

2.5 *Lookup Table*

Lookup Table merupakan *array* yang menggantikan operasi perhitungan dengan operasi indeks. Dengan menggunakan *Lookup Table* waktu untuk komputasi secara signifikan dapat lebih cepat karena mendapatkan data dari memori memakan waktu yang relatif singkat dibandingkan dengan melakukan operasi perhitungan. Tabel dihitung pada komputer lain kemudian hasilnya disimpan untuk digunakan pada komputer robot. Pada tugas akhir ini, rumus untuk melakukan indeks nilai kelas dari masing-masing ruang warna HSV pada tabel dapat dilihat pada Persamaan 2.14.

$$\text{LUT}[\text{H} << 16 \mid \text{S} << 8 \mid \text{V}] \rightarrow \text{kelasHSV} \qquad (2.14)$$

Pada Persamaan 2.14, dilakukan *shift left* nilai *hue* (8 bit) sebanyak 16, dan *shift left* nilai *saturation* (8 bit) sebanyak 8. Kemudian dilakukan operasi *or* pada ketiganya. Nilai hasil *or* kemudian digunakan sebagai indeks array yang diberi nama LUT.

[Halaman ini sengaja dikosongkan]

BAB III
DESAIN PERANGKAT LUNAK

Pada bab ini akan dijelaskan gambaran umum program melalui *flowchart* dan dilanjutkan dengan penjabaran terkait lingkungan pengembangan perangkat lunak.

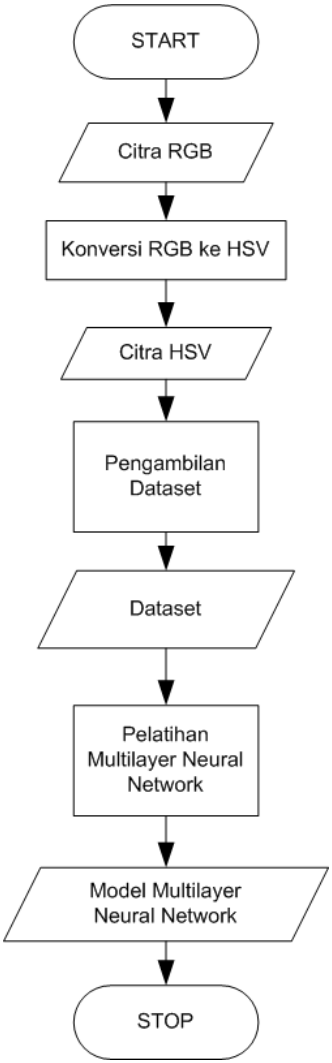
3.1 Desain Program Secara Umum

Secara umum, desain perangkat lunak yang digunakan pada tugas akhir ini terdiri dari tiga bagian utama. Yaitu pembentukan model *Multilayer Neural Network*, pembentukan *Lookup Table*, dan pengujian hasil segmentasi.

3.1.1 Pembentukan Model *Multilayer Neural Network*

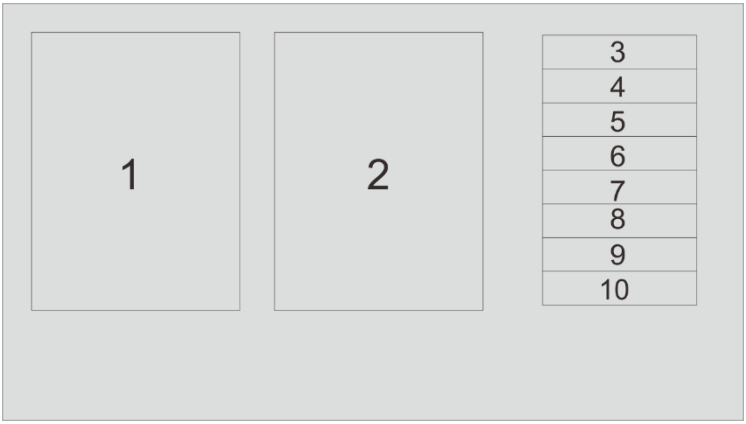
Sistem dimulai dengan menerima citra RGB berukuran 480x640 dari kamera *Logitech C922*. Kemudian, citra dikonversi ke ruang warna HSV. Citra yang telah dikonversi, piksel-pikselnya diklasifikasi secara manual menggunakan bantuan perangkat lunak ke dalam 6 kelas yang telah ditentukan, yaitu: kawan (cyan), lawan (magenta), lapangan (hijau), garis lapangan (putih), bola (orange), dan objek lain. Setelah mendapatkan data yang cukup, berupa *hue*, *saturation*, *value*, dan kelasnya, data kemudian digunakan untuk melatih model *Multilayer Neural Network.*

Karena keterbatasan perangkat keras, tidak dimungkinkan menggunakan seluruh data untuk melatih model *Multilayer Neural Network*. Sehingga, dipilih 500000 data secara random dari data yang didapatkan. Dari data terpilih, 80 persennya digunakan sebagai data training, kemudian sisanya sebagai data testing. Secara umum, *flowchart* program pembentukan model Multilayer Neural Network dapat dilihat pada Gambar 3.1.



Gambar 3.1 Flowchart Program Pembentukan Model Multilayer Neural Network

Pada tugas akhir ini dibuat program khusus dengan antar muka grafis untuk pengambilan dataset. Hal ini dilakukan untuk mempermudah proses pengambilan dataset. Desain antarmuka program pengambilan dataset dapat dilihat pada Gambar 3.2. Sedangkan spesifikasi elemen pada antarmuka program pengambilan dataset dapat dilihat pada Tabel 3.1.



Gambar 3.2 Desain Antarmuka Program Pengambilan Dataset

Dataset yang dihasilkan dari tahap pengambilan dataset ialah piksel-piksel HSV yang telah terklasifikasi secara manual kedalam 6 kelas. Dataset kemudian digunakan pada tahap pelatihan *multilayer neural network*. *Multilayer neural network* yang akan dilatih terdiri dari *input layer* dengan 3 *neuron*, 1 *hidden layer*, dan *output layer* dengan 6 neuron, serta menggunakan fungsi aktivasi *sigmoid*.

3.1.2 Pembentukan *Lookup Table*

Implementasi sebenarnya pada robot, *Lookup Table* digunakan untuk menggantikan operasi perhitungan *Multilayer Neural Network* dengan operasi indeks. Dengan menggunakan *Lookup Table* waktu untuk komputasi secara signifikan dapat

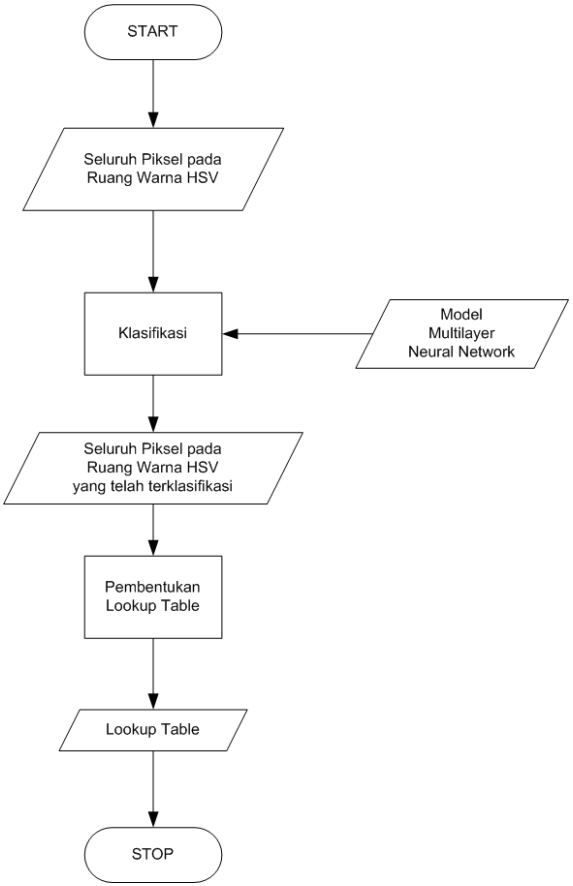
lebih cepat karena mendapatkan data dari memori memakan waktu yang relatif singkat dibandingkan dengan melakukan operasi perhitungan.

Pada tahap ini, model *Multilayer Neural Network* yang didapatkan dari sub bab 3.1.1 digunakan untuk mengklasifikasikan seluruh piksel pada ruang warna HSV. Kemudian hasil klasifikasi disimpan pada *Lookup Table* yang indeksnya dihitung menggunakan Persamaan 2.14. *Flowchart* program pembentukan *Lookup Table* dapat dilihat pada Gambar 3.3.

Tabel 3.1 Spesifikasi Elemen Pada Antarmuka Program Pengambilan Dataset

No.	Elemen	Tipe/Label	Kegunaan
1	Video	*Image*	Menampilkan Data Video
2	Video 2	*Image*	Tempat Pengambilan Data (*mousePressed, mouseDragged, mouseReleased*)
3	Start/Stop Video	*Button*	*Start*, dan *Stop* Data Video
4	Save Dataset	*Button*	Menyimpan Dataset ke File
5	Add Data Field	*Button*	Menambahkan Data Kelas *field/green*
6	Add Data Line	*Button*	Menambahkan Data Kelas *line/white*
7	Add Data Ball	*Button*	Menambahkan Data Kelas *ball/orange*
8	Add Data Cyan	*Button*	Menambahkan Data Kelas *cyan*
9	Add Data Magenta	*Button*	Menambahkan Data Kelas *magenta*
10	Add Data Black	*Button*	Menambahkan Data Kelas *black*

Indeks *lookup table* merupakan hasil operasi *or* antara nilai *hue* (8 bit) yang telah dilakukan *shift left* sebanyak 16, nilai *saturation* (8 bit) yang telah dilakukan *shift left* sebanyak 8, dan nilai *value* (8 bit). Ilustrasi *lookup table* pada tugas akhir ini dapat dilihat pada Tabel 3.2.



Gambar 3.3 Flowchart Program Pembentukan Lookup Table

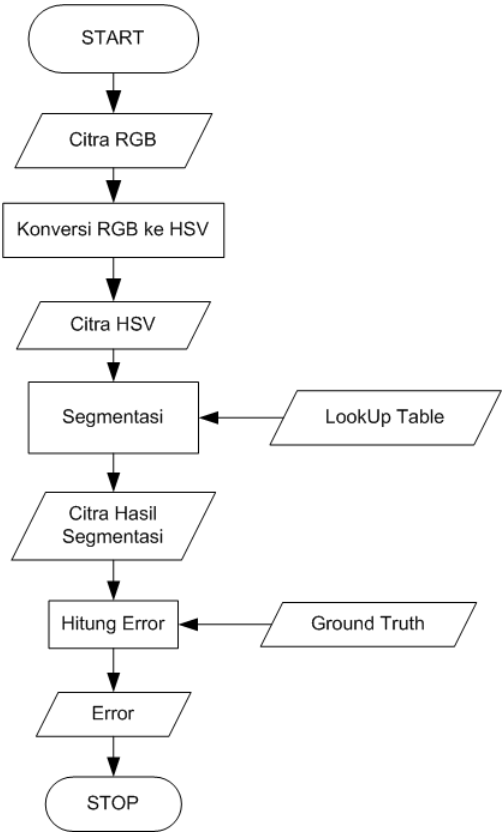
Tabel 3.2 Ilustrasi Lookup Table

H	S	V	Indeks (bin)	Indeks (int)	Kelas
0	0	0	000000000000 000000000000	0	3
0	0	1	000000000000 000000000001	1	3
. . .	. . .	. . .	. . .	. . .	. . .
0	1	0	000000000000 000100000000	256	3
0	1	1	000000000000 000100000001	257	3
. . .	. . .	. . .	. . .	. . .	. . .
255	255	255	111111111111 111111111111	16777215	5

3.1.3 Pengujian Hasil Segmentasi

Pengujian dilakukan untuk mengetahui performa segmentasi citra berdasarkan *Lookup Table* yang dihasilkan dari sub bab 3.1.2. Pada tugas akhir ini, performa yang diperhatikan adalah *error* dari model dikarenakan tujuan utama model adalah mendapatkan tingkat kesalahan yang sekecil mungkin. *Error* didefinisikan dengan Persamaan 3.1.

$$error = \frac{prediksi\ salah}{total\ data} \times 100\% \qquad (3.1)$$



Gambar 3.4 Flowchart Program Pengujian Hasil Segmentasi

Program dimulai dengan melakukan konversi citra RGB yang digunakan sebagai citra pengujian, menjadi citra HSV. Kemudian, dilakukan segmentasi citra dengan cara memperbarui piksel citra dengan kelas yang sesuai berdasarkan *Lookup Table* yang dihasilkan dari sub bab 3.1.2.

Pada tahap perhitungan *error*, tiap piksel citra hasil segmentasi akan dibandingkan dengan *ground truth*. *Error* dihitung dengan menggunakan Persamaan 3.1. Dengan total

data merupakan panjang citra dikalikan dengan lebar citra, dan prediksi salah merupakan jumlah piksel yang tidak sesuai dengan g*round truth*.

3.2 Lingkungan Pengembangan Perangkat Lunak

Pada sub bab 3.1.1, dijelaskan gambaran umum tahapan-tahapan pembentukan model *Multilayer Neural Network*. Pada tahap pengambilan citra, konversi citra dari RGB ke HSV, sampai dengan tahap pengambilan data, perangkat lunak ditulis dengan bahasa *C++* dan dengan menggunakan bantuan pustaka *OpenCV*.

OpenCV adalah pustaka fungsi-fungsi pemrograman untuk computer vision. Pustaka ini menggunakan lisensi BSD. Oleh karena itu, pustaka ini bebas digunakan untuk bidang akademik maupun komersial. Pustaka ini ada dalam bahasa *C++*, *Python* dan *Java* (*Android*) dan memiliki dukungan untuk sistem operasi *Windows*, *Linux*, *Android*, *iOS* dan *Mac OS*. [5]

Gambar 3.5 Logo OpenCV [5]

Sedangkan tahap pelatihan *Multilayer Neural Network* pada sub bab 3.1.1, dan seluruh tahapan proses pembentukan *Lookup Table* (sub bab 3.1.2) ditulis dengan bahasa *Python 3.6*. Hal ini dikarenakan operasi matematis cenderung lebih mudah diimplementasikan dengan bahasa *Python* daripada *C++*. Seluruh tahapan proses pengujian hasil segmentasi ditulis menggunakan bahasa *C++* dan dengan menggunakan bantuan

pustaka *OpenCV*. *Ground Truth* pada sub bab 3.1.3 merupakan citra yang telah dilakukan segmentasi secara manual oleh manusia yang didapatkan dengan bantuan perangkat lunak CorelDRAW X7.

[Halaman ini sengaja dikosongkan]

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.

4.1 Lingkungan Implementasi

Lingkungan implementasi yang akan digunakan pada tugas akhir ini adalah sebagai berikut:

4.1.1 Perangkat Keras

Lingkungan implementasi pada tugas akhir ini adalah sebuah *personal computer* (PC). PC yang digunakan pada tugas akhir ini memiliki prosesor Intel Core i5-7200U dengan kecepatan 2,6 GHz, *Random Access Memory* (RAM) sebesar 4,00 GB, dan kartu grafis NVIDIA GeForce 920MX.

4.1.2 Perangkat Lunak

Program dibangun menggunakan *Python 3.6* pada sistem operasi *Windows* dengan menggunakan IDE PyCharm, serta *C++* menggunakan IDE Visual Studio 2015.

4.2 Implementasi Pembentukan Model *Multilayer Neural Network*

Pada sub bab ini, akan dijabarkan implementasi fungsi-fungsi yang digunakan pada tahap pembentukan model *Multilayer Neural Network*, beserta kode sumber.

4.2.1 Implementasi Fungsi Konversi Citra RGB ke HSV

Pada tahap ini, citra RGB yang diperoleh, dikonversi menjadi citra HSV, untuk kemudian diproses lebih lanjut pada tahap selanjutnya.

```
cvtColor(frame, frameHSV, CV_BGR2HSV);
```

Kode 4.1 Fungsi Konversi Citra RGB ke HSV

4.2.2 Implementasi Tahap Pengambilan Data Training

Pada tahap ini, akan dijabarkan beberapa fungsi yang akan digunakan pada tahap pengambilan data training.

4.2.2.1 Implementasi Fungsi onButtonEvent()

Fungsi **onButtonEvent()** digunakan sebagai *event handler* tombol-tombol yang digunakan pada tahap pengambilan data. Beberapa tombol yang digunakan antara lain: tombol untuk memainkan dan stop video, tombol untuk mengambil data tiap kelas, dan tombol untuk menyimpan hasil pengambilan data.

```
void ofApp::onButtonEvent(ofxDatGuiButtonEvent e)
{
        // stop video button
        if (e.target->is("STOP"))
        {
                play = false;
                e.target->setName("PLAY");
                e.target->setLabel("PLAY");
        }
        // play video button
        else if (e.target->is("PLAY"))
        {
                play = true;
                e.target->setName("STOP");
                e.target->setLabel("STOP");
        }
        // show HSV image
        else if (e.target->is("HSV"))
        {
                frame2show2state = HSV;
        }
```

```
// show segmented image
else if (e.target->is("SEGMENTED"))
{
            frame2show2state = SEGMENTED;
}
// reset dataset
else if (e.target->is("RESET DATA"))
{
            save.lock();
            save.reset_index();
            save.unlock();
}
else if (e.target->is("ADD DATA FIELD"))
{
            frame2show2state = DATA;
            working = frame.clone();
            add_data = true;
            data_colors = GREEN;
}
else if (e.target->is("ADD DATA LINE"))
{
            frame2show2state = DATA;
            working = frame.clone();
            add_data = true;
            data_colors = WHITE;
}
else if (e.target->is("ADD DATA BALL"))
{
            frame2show2state = DATA;
            working = frame.clone();
            add_data = true;
            data_colors = ORANGE;
}
else if (e.target->is("ADD DATA CYAN"))
{
            frame2show2state = DATA;
            working = frame.clone();
            add_data = true;
            data_colors = CYAN;
}
else if (e.target->is("ADD DATA MAGENTA"))
{
            frame2show2state = DATA;
            working = frame.clone();
            add_data = true;
            data_colors = MAGENTA;
}
else if (e.target->is("ADD DATA BLACK"))
{
            frame2show2state = DATA;
            working = frame.clone();
            add_data = true;
            data_colors = BLACK;
}\
```

```
            // save dataset
            else if (e.target->is("SAVE SETTING"))
            {
                        save.lock();
                        save.indexs = indexs;
                        save.saving = true;
                        save.unlock();
            }
    }
```

Kode 4.2 Fungsi onButtonEvent()

4.2.2.2 Implementasi Fungsi mousePressed()

Fungsi **mousePressed()** digunakan untuk mendapatkan posisi kursor *mouse*, ketika *mouse* ditekan. Pada tahap ini, posisi kursor akan disimpan pada objek *base* dari kelas *cv::Point*, dan objek *r* dari kelas *cv::Rect*. Objek *r* memiliki attribut *x*, *y*, *width*, dan *height*.

```
void mousePressed(int x, int y, int button)
{
         base.x = x;
         base.y = y;
         r.x = x;
         r.y = y;
         r.width = 0;
         r.height = 0;
}
```

Kode 4.3 Fungsi mousePressed()

4.2.2.3 Implementasi Fungsi mouseDragged()

Fungsi **mouseDragged()** digunakan untuk mendapatkan posisi kursor *mouse*, ketika *mouse* sedang diseret. Pada fungsi ini, attribut *x*, *y*, *width*, dan *height* pada objek *r* akan diperbarui.

```
void mouseDragged(int x, int y, int button)
{
         if (add_data == false) return;
```

```
          int dx = abs(r.x - x);
          int dy = abs(r.y - y);

          if (x < base.x)
          {
                    r.x = x;
                    r.width = abs(x - base.x);
          }
          else
          {
                    r.width = dx;
          }
          if (y < base.y)
          {
                    r.y = y;
                    r.height = abs(y - base.y);
          }
          else
          {
                    r.height = dy;
          }
}
```

Kode 4.4 Fungsi mouseDragged()

4.2.2.4 Implementasi Fungsi mouseReleased()

Fungsi **mouseReleased()** digunakan untuk mendapatkan posisi kursor *mouse*, ketika *mouse* dilepas. Pada fungsi ini, piksel-piksel yang berada didalam persegi panjang yang posisi x, y, panjang, dan lebarnya berasal dari objek **r**, akan diambil nilai *hue*, *saturation*, dan *value*nya. Kemudian kelas dari piksel tersebut akan disimpan berdasarkan tombol yang sedang dipilih pada fungsi **onButtonEvent()**.

```
void mouseReleased(int x, int y, int button)
{
          if (add_data == true)
          {
                    uint8_t* pixelPtr = (uint8_t*) frameHSV.data;
                    int cn = frameHSV.channels();
                    for (int i = r.y; i < r.y + r.height; i++)
                    {
                              for (int j = r.x; j < r.x + r.width; j++)
```

```
                            {
                                int h = pixelPtr[i *
frameHSV.cols * cn + j * cn + 0];
                                int s = pixelPtr[i *
frameHSV.cols * cn + j * cn + 1];
                                int v = pixelPtr[i *
frameHSV.cols * cn + j * cn + 2];
                                indexs[h << 16 | s << 8 | v] =
data_colors;
                            }
                    }
          }
          r = Rect();
          add_data = false;
}
```

Kode 4.5 Fungsi mouseReleased()

4.2.3 Implementasi Tahap Pelatihan *Multilayer Neural Network*

Pada tahap ini, akan dijabarkan beberapa fungsi yang akan digunakan pada tahap pelatihan *multilayer neural network*.

4.2.3.1 Implementasi Fungsi SGD()

Fungsi **SGD()** merupakan implementasi dari *stochastic gradient descent* pada *multilayer neural network*. Pada fungsi ini, untuk tiap *epoch*, akan diambil data secara random sebanyak *mini_batch_size*, kemudian memanggil fungsi **update_mini_batch()**. *Eta* merupakan *learning rate* dari *multilayer neural network*, dan *lambda* merupakan parameter regularisasi.

```
def SGD(self, training_data, epochs, mini_batch_size, eta,
            lmbda = 0.0,
            evaluation_data=None,
            monitor_evaluation_cost=False,
            monitor_evaluation_accuracy=False,
            monitor_training_cost=False,
            monitor_training_accuracy=False,
            early_stopping_n = 0):
```

```
        best_accuracy=1
        training_data = list(training_data)
        n = len(training_data)
        if evaluation_data:
            evaluation_data = list(evaluation_data)
            n_data = len(evaluation_data)

        best_accuracy=0
        no_accuracy_change=0
        evaluation_cost, evaluation_accuracy = [], []
        training_cost, training_accuracy = [], []
        for j in range(epochs):
            random.shuffle(training_data)
            mini_batches = [
                training_data[k:k+mini_batch_size]
                for k in range(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(
                    mini_batch, eta, lmbda, len(training_data))

            print("Epoch %s training complete" % j)

            if monitor_training_cost:
                cost = self.total_cost(training_data, lmbda)
                training_cost.append(cost)
                print("Cost on training data: {}".format(cost))
            if monitor_training_accuracy:
                accuracy = self.accuracy(training_data, convert=True)
                training_accuracy.append(accuracy)
                print("Accuracy on training data: {} /
{}".format(accuracy, n))
            if monitor_evaluation_cost:
                cost = self.total_cost(evaluation_data, lmbda,
convert=True)
                evaluation_cost.append(cost)
                print("Cost on evaluation data: {}".format(cost))
            if monitor_evaluation_accuracy:
                accuracy = self.accuracy(evaluation_data)
                evaluation_accuracy.append(accuracy)
                print("Accuracy on evaluation data: {} /
{}".format(self.accuracy(evaluation_data), n_data))

            if early_stopping_n > 0:
                if accuracy > best_accuracy:
                    best_accuracy = accuracy
                    no_accuracy_change = 0
                    #print("Early-stopping: Best so far
{}".format(best_accuracy))
                else:
                    no_accuracy_change += 1

                if (no_accuracy_change == early_stopping_n):
                    #print("Early-stopping: No accuracy change in
last epochs: {}".format(early_stopping_n))
```

```
                    return evaluation_cost, evaluation_accuracy,
training_cost, training_accuracy

        return evaluation_cost, evaluation_accuracy, \
            training_cost, training_accuracy
```

Kode 4.6 Fungsi SGD()

4.2.3.2 Implementasi Fungsi update_mini_batch()

Fungsi **update_mini_batch()** akan memperbarui *weight* dan *bias* pada *multilayer neural network* dengan menerapkan *gradient descent* menggunakan *backpropagation* ke satu *mini batch*. *Eta* merupakan *learning rate* dari *multilayer neural network*, *lambda* merupakan parameter regularisasi, dan *n* merupakan jumlah dataset.

```
def update_mini_batch(self, mini_batch, eta, lmbda, n):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb + dnb for nb, dnb in zip(nabla_b,
delta_nabla_b)]
            nabla_w = [nw + dnw for nw, dnw in zip(nabla_w,
delta_nabla_w)]
        self.weights = [(1 – eta * (lmbda / n)) * w - (eta /
len(mini_batch)) * nw
                        for w, nw in zip(self.weights, nabla_w)]
        self.biases = [b - (eta / len(mini_batch)) * nb
                       for b, nb in zip(self.biases, nabla_b)]
```

Kode 4.7 Fungsi update_mini_batch()

4.2.3.3 Implementasi Fungsi backprop()

Fungsi **backprop()** merupakan implementasi dari *backpropagation*. Fungsi ini menghasilkan *tuple* yang berisi *nabla_b*, dan *nabla_w*. Keduanya merepresentasikan *gradient* untuk *cost function*.

```
def backprop(self, x, y):
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
```

```
        # feedforward
        activation = x
        activations = [x] # list to store all the activations, layer
by layer
        zs = [] # list to store all the z vectors, layer by layer
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation) + b
            zs.append(z)
            activation = sigmoid(z)
            activations.append(activation)
        # backward pass
        delta = (self.cost).delta(zs[-1], activations[-1], y)
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())

        for l in range(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) *
sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-
1].transpose())
        return (nabla_b, nabla_w)
```

Kode 4.8 Fungsi backprop()

4.2.3.4 Implementasi Fungsi sigmoid()

Fungsi **sigmoid()** merupakan implementasi dari fungsi aktivasi *sigmoid*.

```
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
```

Kode 4.9 Fungsi sigmoid()

4.2.3.5 Implementasi Fungsi sigmoid_prime()

Fungsi **sigmoid_prime()** merupakan *derivative* dari fungsi sigmoid pada sub bab 4.2.3.4.

```
def sigmoid_prime(z):
    return sigmoid(z) * (1 - sigmoid(z))
```

Kode 4.10 Fungsi sigmoid_prime()

4.2.3.6 Implementasi Kelas CrossEntropyCost

Kelas **CrossEntropyCost** memiliki dua fungsi. Yang pertama, fungsi *fn()*, digunakan untuk menghitung *cost*

menggunakan *cross-entropy cost function* yang telah dijelaskan pada sub bab 2.2.4. Sedangkan yang kedua, fungsi *delta*(), digunakan pada proses *backpropagation*.

Untuk meminimalkan *cross entropy error* pada saat pelatihan menggunakan *backpropagation*, tidak diperlukan untuk benar-benar menghitung *cross entropy*, cukup menghitung perbedaan antara nilai target dan nilai sekarang.

```
class CrossEntropyCost(object):

    @staticmethod
    def fn(a, y):
        return np.sum(np.nan_to_num(-y*np.log(a)-(1-y)*np.log(1-a)))

    @staticmethod
    def delta(z, a, y):
        return (a-y)
```

Kode 4.11 Kelas CrossEntropyCost

4.2.3.7 Implementasi Fungsi accuracy()

Fungsi **accuracy()** digunakan untuk menghitung jumlah data training yang berhasil diklasifikasikan dengan benar oleh *multilayer neural netrwork*.

```
def accuracy(self, data):
        results = [(np.argmax(self.feedforward(x)), y)
                for (x, y) in data]

        result_accuracy = sum(int(x == y) for (x, y) in results)
        return result_accuracy
```

Kode 4.12 Fungsi accuracy()

4.2.3.8 Implementasi Fungsi feedforward()

Fungsi **feedforward()** digunakan untuk menghitung *output* dari *multilayer neural network* apabila menerima *input a*.

```
def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a) + b)
        return a
```

Kode 4.13 Fungsi feedforward()

4.2.3.9 Implementasi Fungsi total_cost()

Fungsi **total_cost()** digunakan untuk menghitung *total cost* dari dataset *data*.

```
def total_cost(self, data, lmbda):
        cost = 0.0
        for x, y in data:
            a = self.feedforward(x)
            cost += self.cost.fn(a, y)/len(data)
            cost += 0.5*(lmbda/len(data))*sum(np.linalg.norm(w)**2
for w in self.weights)
        return cost
```

Kode 4.14 Fungsi total_cost()

4.2.3.10 Implementasi Fungsi save()

Fungsi **save()** digunakan untuk menyimpan arsitektur, *weight*, *bias*, dan *cost* dari *multilayer neural network*.

```
def save(self, filename):
        data = {"sizes": self.sizes,
                "weights": [w.tolist() for w in self.weights],
                "biases": [b.tolist() for b in self.biases],
                "cost": str(self.cost.__name__)}
        f = open(filename, "w")
        json.dump(data, f)
        f.close()
```

Kode 4.15 Fungsi save()

4.2.3.11 Implementasi Fungsi load()

Fungsi **load()** digunakan untuk memuat *multilayer neural network* dari file yang telah disimpan menggunakan fungsi **save()**.

```
def load(filename):
    f = open(filename, "r")
    data = json.load(f)
    f.close()
    cost = getattr(sys.modules[__name__], data["cost"])
    net = Network(data["sizes"], cost=cost)
    net.weights = [np.array(w) for w in data["weights"]]
    net.biases = [np.array(b) for b in data["biases"]]
    return net
```

Kode 4.16 Fungsi load()

4.2.3.12 Implementasi Fungsi classify()

Fungsi **classify()** digunakan untuk menentukan kelas dari input *data*.

```
def classify(self, data):
        return np.argmax(self.feedforward(data))
```

Kode 4.17 Fungsi classify()

4.3 Implementasi Pembentukan *Lookup Table*

Pembentukan *Lookup Table* dimulai dengan mengklasifikasikan seluruh piksel pada ruang warna HSV menggunakan *multilayer neural network*. Kemudian, hasil klasfikasi disimpan dalam *lookup table*. Indeks tabel yang digunakan untuk menyimpan kelas dari nilai *hue h*, *saturation s*, dan *value v*, didapatkan dari Persamaan 2.14.

```
LUT = array('B')
for i in range(256 ** 3):
    LUT.append(0)

for h in range(0, 256):
    for s in range(0, 256):
        for v in range(0, 256):
            data = np.array([[h], [s], [v]]) / 255
            LUT[h << 16 | s << 8 | v] = nn.classify(data)

with open('..\Deployment\Deployment\indexs.bin', 'wb') as f:
    LUT.tofile(f)
```

Kode 4.18 Pembentukan Lookup Table

4.4 Implementasi Pengujian Hasil Segmentasi

Pengujian hasil segmentasi, dilakukan dengan cara membandingkan data *ground truth* dengan hasil segmentasi citra menggunakan *lookup table* yang telah didapatkan dari proses pembentukan *lookup table*. *Ground Truth* merupakan citra yang telah dilakukan segmentasi secara manual oleh manusia yang didapatkan dengan bantuan perangkat lunak CorelDRAW X7.

```
#include <opencv2\opencv.hpp>

#define NUM_IMAGE 4
#define INFO_MEMORY_SIZE (256 * 256 * 256)

enum COLORS
{
	ORANGE = 0,
	GREEN,
	WHITE,
	BLACK,
	CYAN,
	MAGENTA
};

using namespace cv;
using namespace std;

unsigned char *indexs  = new unsigned char[INFO_MEMORY_SIZE]();

int main()
{
	Mat image[NUM_IMAGE];
	Mat image_hsv[NUM_IMAGE];
	Mat image_index[NUM_IMAGE];
	Mat segmented[NUM_IMAGE];
	Mat ground_truth[NUM_IMAGE];

	string cacheFilename =
"../../Deployment/Deployment/indexs.bin";
	open_index(cacheFilename);

	for (int i = 0; i < NUM_IMAGE; i++)
	{
		image[i] = imread("../../image-test/img-" +
to_string(i) + ".png");
		image_index[i] = Mat(image[i].size(), CV_8UC1,
Scalar(WHITE));
		segmented[i] = Mat(image[i].size(), CV_8UC3,
Scalar(0, 0, 0));
		cvtColor(image[i], image_hsv[i], CV_BGR2HSV);
		convertImageToIndex(image_hsv[i], image_index[i]);
		PaintIndexImage(image_index[i], segmented[i]);
		ground_truth[i] = imread("../../image-test/img-
seg-" + to_string(i) + ".png");

		int error = 0;

		for (int j = 0; j < image[i].rows; j++)
		{
			for (int k = 0; k < image[i].cols; k++)
			{
				Vec3b colour_segmented =
segmented[i].at<Vec3b>(Point(k, j));
```

```
                                        Vec3b colour_ground_truth =
ground_truth[i].at<Vec3b>(Point(k, j));
                                        if (colour_segmented[0] !=
colour_ground_truth[0] || colour_segmented[1] !=
colour_ground_truth[1] || colour_segmented[2] !=
colour_ground_truth[2])
                                        {
                                                error++;
                                        }
                                }
                        }

                        float err, acc;
                        err = (float)error * 100 / (image[i].rows *
image[i].cols);
                        cout << "error " << i << ": " << err << endl;
                        acc = 100 - err;
                        cout << "acc " << i << ": " << acc << endl;
                }

                system("pause");
                delete[] indexs;
                return 0;
}
```

Kode 4.19 Pengujian Hasil Segmentasi

4.4.1 Implementasi Fungsi open_index()

Fungsi **open_index()** digunakan untuk membuka file *lookup table* yang dihasilkan dari proses pembentukan *lookup table* (sub bab 4.3).

```
void open_index(string cacheFilename)
{
        std::ifstream cache(cacheFilename.c_str(), std::ios::in |
std::ios::binary);

        if (!cache.is_open())
        {
                std::cout << "Error Opening" << std::endl;
                return;
        }

        cache.seekg(0, std::ios::beg);
        cache.read((char *)indexs, INFO_MEMORY_SIZE);
        cache.close();
}
```

Kode 4.20 Fungsi open_index()

4.4.2 Implementasi Fungsi convertImageToIndex()

Fungsi **convertImageToIndex()** digunakan untuk mengubah citra HSV menjadi citra satu layer. Citra satu layer (CV_8UC1). Tiap piksel citra ini merupakan kelas dari tiap piksel pada citra HSV.

```
void convertImageToIndex(cv::Mat &original, cv::Mat &index)
{
	unsigned char h, s, v;
	uint8_t* pixelPtr = (uint8_t*)original.data;
	int cn = original.channels();
	for (int i = 0; i < original.rows; i++)
	{
		for (int j = 0; j < original.cols; j++)
		{
			h = pixelPtr[i*original.cols*cn + j*cn +
0];
			s = pixelPtr[i*original.cols*cn + j*cn +
1];
			v = pixelPtr[i*original.cols*cn + j*cn +
2];
			index.ptr(i)[j] = indexs[h << 16 | s << 8
| v];
		}
	}
}
```

Kode 4.21 Fungsi convertImageToIndex()

4.4.3 Implementasi Fungsi PaintIndexImage()

Fungsi **PaintIndexImage()** akan mewarnai citra *dst*, sesuai dengan angka indeks pada citra *src*, dengan cara memanggil fungsi **PaintPixelWithColor()**.

```
void PaintIndexImage(cv::Mat &src, cv::Mat &dst)
{
	for (unsigned p = 0; p < src.cols * src.rows; p++)
	{
		switch ((char)src.ptr()[p])
		{
		case ORANGE:
			PaintPixelWithColor(dst, p * 3, ORANGE);
			break;
```

```
                    case GREEN:
                              PaintPixelWithColor(dst, p * 3, GREEN);
                              break;
                    case WHITE:
                              PaintPixelWithColor(dst, p * 3, WHITE);
                              break;
                    case BLACK:
                              PaintPixelWithColor(dst, p * 3, BLACK);
                              break;
                    case CYAN:
                              PaintPixelWithColor(dst, p * 3, CYAN);
                              break;
                    case MAGENTA:
                              PaintPixelWithColor(dst, p * 3, MAGENTA);
                              break;
                    default:
                              PaintPixelWithColor(dst, p * 3, 999);
                              break;
                    }
          }
}
```

Kode 4.22 Fungsi PaintIndexImage()

4.4.4 Implementasi Fungsi PaintPixelWithColor()

Fungsi **PaintPixelWithColor()** digunakan untuk mewarnai piksel citra *img* pada posisi *pos*, menggunakan warna sesuai kelasnya.

```
void PaintPixelWithColor(cv::Mat &img, int pos, int color)
{
          switch (color)
          {
          case ORANGE:
                    img.ptr()[pos] = 255;
                    img.ptr()[pos + 1] = 165;
                    img.ptr()[pos + 2] = 0;
                    break;
          case GREEN:
                    img.ptr()[pos] = 0;
                    img.ptr()[pos + 1] = 255;
                    img.ptr()[pos + 2] = 0;
                    break;
          case WHITE:
                    img.ptr()[pos] = 255;
                    img.ptr()[pos + 1] = 255;
                    img.ptr()[pos + 2] = 255;
                    break;
```

```
            case BLACK:
                    img.ptr()[pos] = 0;
                    img.ptr()[pos + 1] = 0;
                    img.ptr()[pos + 2] = 0;
                    break;
            case CYAN:
                    img.ptr()[pos] = 20;
                    img.ptr()[pos + 1] = 0;
                    img.ptr()[pos + 2] = 200;
                    break;
            case MAGENTA:
                    img.ptr()[pos] = 255;
                    img.ptr()[pos + 1] = 0;
                    img.ptr()[pos + 2] = 255;
                    break;
            default:
                    img.ptr()[pos] = 127;
                    img.ptr()[pos + 1] = 127;
                    img.ptr()[pos + 2] = 127;
                    break;
            }
}
```

Kode 4.23 Fungsi PaintPixelWithColor()

[Halaman ini sengaja dikosongkan]

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi beberapa tahap, serta analisa setiap pengujian.

5.1 Lingkungan Uji Coba

Lingkungan uji coba meliputi perangkat lunak dan perangkat keras. Pada tugas akhir ini, digunakan lingkungan uji coba pada sebuah komputer *desktop*. Spesifikasi lingkungan uji coba pada tugas akhir ini dapat dilihat pada Tabel 5.1.

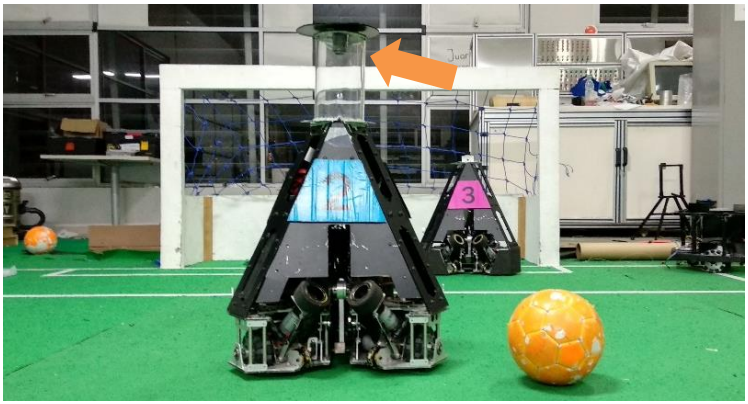
Tabel 5.1 Spesifikasi Lingkungan Uji Coba

Perangkat Lunak	
OS	Windows 8.1 - 64 Bit
IDE	Visual Studio 2015 PyCharm Edu 2018.1
Perangkat Keras	
CPU	Intel(R) Core(TM) i5 – 7200U 2.6 GHz
RAM	4 GB
GPU	Nvidia Ge Force 920MX
GPU Memory	2 GB

Spesifikasi tersebut cukup memadai untuk melatih model *multilayer neural network* dengan kecepatan komputasi yang baik, namun RAM masih cenderung kurang, sehingga tidak semua *dataset* pada tugas akhir ini dapat digunakan, melainkan hanya sebagian kecil saja.

5.2 Dataset

Dataset citra yang digunakan adalah adalah dataset milik Tim Robot Sepak Bola Institut Teknologi Sepuluh Nopember. Dataset merupakan video yang ditangkap menggunakan kamera Logitech C922. Pada bagian depan kamera ini, dipasang lensa FishEye 235°. Karakter lensa FishEye yang cembung menghasilkan citra yang terdistorsi melengkung. Lensa FishEye digunakan untuk meningkatkan *angles of view*. Kamera dipasang pada bagian atas robot dengan arah menghadap ke bawah. Pemasangan kamera dapat dilihat pada Gambar 5.1.

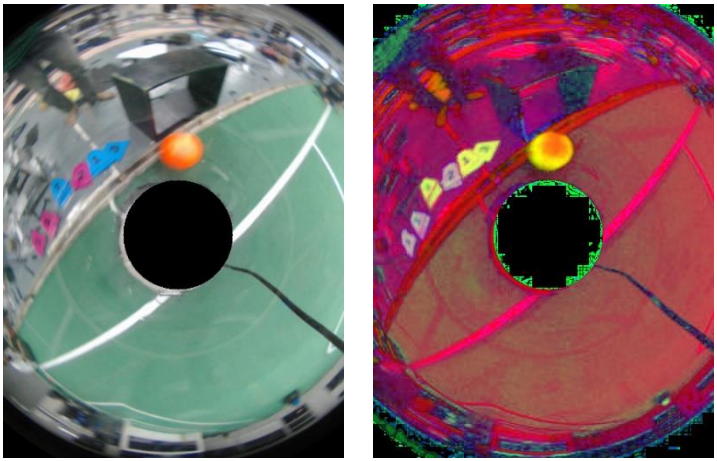


Gambar 5.1 Letak Pemasangan Kamera

Dari dataset video, kemudian diambil sampel piksel yang diklasifikasi secara manual sesuai kelas yang ditentukan. Data inilah yang akan digunakan untuk melatih model *Multilayer Neural Network*. Jumlah sampel piksel yang telah terklasifikasi secara manual adalah sebanyak 8.234.496.

5.3 Uji Coba Proses

Sebelum pengujian utama, akan dilakukan uji coba terhadap proses yang terjadi pada program. Uji coba tersebut yaitu uji coba terhadap proses konversi citra RGB ke HSV. Hasil konversi dapat dilihat pada Gambar 5.2.



Gambar 5.2 Citra RGB (kiri), dan Citra HSV (kanan)

5.4 Skenario dan Evaluasi Pengujian

Pengujian pada tugas akhir ini bertujuan untuk mendapatkan model dengan nilai *error* terkecil. Hal tersebut dilakukan dengan melakukan optimasi *hyperparameter* pada model. *Hyperparameter* merupakan variabel yang menentukan struktur jaringan, seperti jumlah *neuron*, maupun variabel yang menentukan bagaimana jaringan dilatih, seperti *learning rate*. *Hyperparameter* harus ditentukan sebelum proses pelatihan. Proses pengujian akan dilakukan melalui beberapa tahap.

1. *Fine Tuning Hyperparameter*: Untuk mengoptimasi *hyperparameter* jaringan agar mendapat model yang berfungsi dengan lebih optimal.

2. Perbandingan Arsitektur: Untuk mencari arsitektur model *multilayer neural network* dengan nilai *error* terkecil.
3. Uji Coba Segmentasi: Untuk memastikan apakah *lookup table* yang telah dibuat berdasarkan hasil klasifikasi menggunakan *multilayer neural network* telah layak diimplementasikan pada robot, serta untuk membandingkan kecepatan hasil segmentasi menggunakan *lookup table* dengan segmentasi menggunakan sistem lama yang telah diimplementasikan pada robot sebelumnya (menggunakan *thresholding*).

Pada tahap 1 dan 2, akan diambil secara random sebanyak 500.000 dataset (karena keterbatasan sumber daya). Kemudian, 80% dari data tersebut digunakan sebagai *data training*, dan 20% sisanya digunakan sebagai *data testing*. Sedangkan pada tahap 3, evaluasi hasil segmentasi dilakukan dengan membandingkan tiap piksel citra hasil segmentasi dengan *ground truth*. *Ground truth* merupakan citra yang telah dilakukan segmentasi secara manual dengan menggunakan bantuan perangkat lunak pihak ke 3.

5.5 Hasil Uji Coba

Pada sub bab ini akan ditampilkan hasil uji coba berupa visualisasi dalam bentuk tabel hasil pengujian berdasarkan skenario yang terdapat pada sub bab 5.3.

5.5.1 *Fine Tuning Hyperparameter*

Fine tuning hyperparameter bertujuan untuk mencari konfigurasi *hyperparameter* yang paling baik pada *multilayer neural network* pada dataset yang telah disediakan. Pada tugas akhir ini akan dilakukan *fine tuning* pada *learning rate* dan *lambda*. Untuk mengurangi *fluktuasi* pada kualitas model, akan diambil nilai rata-rata dari 2 pengujian pada masing-masing konfigurasi.

5.5.1.1 *Learning Rate*

Hyperparameter yang dilakukan *fine tuning* pertama kali ialah *learning rate*. Pada proses *fine tuning learning rate*, jumlah *epoch* yang digunakan berjumlah 50, dengan *batch size* sebanyak 10. *Fine tuning* pada proses ini tidak menggunakan L2 *regularization*, sehingga *lambda* diset dengan nilai 0,0. Konfigurasi yang digunakan dapat dilihat pada Tabel 5.2.

Tabel 5.2 Hyperparameter dan Arsitektur Tetap Pengujian Learning Rate

Nama	**Nilai**
***Hyperparameter* Jaringan**	
Epoch	50
Batch Size	10
Lambda	0,0
Arsitektur Jaringan	
Neuron tiap *Layer*	3 – 6 – 6

Dari hasil pengujian menggunakan berbagai nilai *learning rate* yang berbeda, didapatkan rata-rata *error* terkecil pada nilai *learning rate* 0,3. Sehingga, selanjutnya proses *fine tuning* akan dilakukan dengan menggunakan nilai *learning rate* 0,3. Hasil dari pengujian *learning rate* dapat dilihat pada Tabel 5.3.

Tabel 5.3 Hasil Pengujian Learning Rate

Learning Rate	**Rata-rata *Error Training***	**Rata-rata *Error Testing***
0,01	0,79%	0,77%
0,1	0,32%	0,33%
0,2	0,40%	0,42%
0,3	**0,26%**	**0,25%**
0,4	0,29%	0,28%

5.5.1.2 *Lambda*

Setelah dilakukan proses *fine tuning learning rate*, proses selanjutnya yaitu *fine tuning lambda*. *Lambda* merupakan *hyperparameter* pada L2 *regularization*. Pada proses *fine tuning lambda*, jumlah *epoch* yang digunakan berjumlah 50, dengan *batch size* sebanyak 10, serta *learning rate* 0,3 yang merupakan nilai terbaik yang didapatkan dari proses *fine tuning learning rate*. Konfigurasi yang digunakan dapat dilihat pada Tabel 5.4.

Tabel 5.4 Hyperparameter dan Arsitektur Tetap Pengujian Lambda

Nama	**Nilai**
***Hyperparameter* Jaringan**	
Epoch	50
Batch Size	10
Learning Rate	0,3
Arsitektur Jaringan	
Neuron tiap *Layer*	3 – 6 – 6

Dari hasil pengujian menggunakan beberapa angka *lambda* yang berbeda, didapatkan rata-rata *error* terkecil pada nilai *lambda* 0,1. Sehingga, selanjutnya pengujian akan dilakukan dengan menggunakan nilai *lambda* 0,1. Hasil dari pengujian *lambda* dapat dilihat pada Tabel 5.5.

Tabel 5.5 Hasil Pengujian Lambda

Lambda	**Rata-rata *Error Training***	**Rata-rata *Error Testing***
0	0,26%	0,25%
0,1	**0,21%**	**0,21%**
0,01	0,27%	0,26%
0,001	0,43%	0,43%

5.5.2 Perbandingan Arsitektur

Pada pengujian ini akan dilakukan pengujian terhadap model *multilayer neural network* dengan tujuan melakukan komparasi arsitektur jaringan untuk mendapatkan model terbaik. Arsitektur jaringan terbaik kemudian akan digunakan untuk mengklasifikasi seluruh piksel pada ruang warna HSV untuk selanjutnya digunakan untuk membangun *lookup table*.

Secara umum, *multilayer neural network* dapat memiliki *hidden layer* dan *neuron* pada tiap *layer* dengan jumlah berapapun. Namun, pada tugas akhir ini, arsitektur jaringan yang digunakan dibatasi dengan menggunakan 1 *hidden layer* saja, dengan jumlah *neuron* pada *input layer* berjumlah 3 (nilai *hue*, nilai *saturation*, dan nilai *value*), dan jumlah *neuron* pada *output layer* berjumlah 6 (kawan, lawan, lapangan, garis lapangan, bola, dan objek lain), serta menggunakan fungsi aktivasi *sigmoid*. Sehingga perbandingan arsitektur pada subbab ini dilakukan dengan membandingkan jumlah *neuron* pada *hidden layer*.

Konfigurasi yang digunakan pada pengujian arsitektur merupakan hasil *fine tuning* yang telah dilakukan pada subbab sebelumnya, yaitu menggunakan *learning rate* dengan nilai 0,3 dan *lambda* dengan nilai 0,1. Konfigurasi yang digunakan dapat dilihat pada Tabel 5.6.

Tabel 5.6 Hyperparameter Pengujian Arsitektur Jaringan

Nama	**Nilai**
***Hyperparameter* Jaringan**	
Epoch	50
Batch Size	10
Learning Rate	0,3
Lambda	0,1
Arsitektur Jaringan	
Hidden Layer	1

Dari hasil pengujian, didapatkan rata-rata *error* terkecil sebesar 0,16% pada arsitektur jaringan dengan 15 *neuron* pada *hidden layer*. Berdasarkan hasil pengujian, terlihat bahwa jumlah *neuron* pada *hidden layer* berbanding lurus dengan *run time*, yang berarti bahwa proses komputasi akan semakin lama apabila jumlah *neuron* semakin banyak. Namun demikian, semakin banyak *neuron* tidak selalu menghasilkan *error* yang lebih kecil. Hasil dari pengujian arsitektur *multilayer neural network* dapat dilihat pada Tabel 5.7. Dari hasil perbandingan arsitektur ini, arsitektur jaringan *multilayer neural network* dengan rata-rata error terkecil akan digunakan untuk klasifikasi tiap piksel pada ruang warna HSV yang akan digunakan pada proses selanjutnya.

Tabel 5.7 Hasil Pengujian Arsitektur Jaringan

Neuron* tiap *Layer	**Rata-rata *Run Time***	**Rata-rata *Error Training***	**Rata-rata *Error Testing***
3 – 6 – 6	**51,72 menit**	0,21%	0,21%
3 – 10 – 6	56,12 menit	0,19%	0,20%
3 – 13 – 6	55,87 menit	0,17%	0,17%
3 – 15 – 6	57,52 menit	**0,16%**	**0,16%**
3 – 17 – 6	61,41 menit	0,18%	0,19%
3 – 20 – 6	55,36 menit	0,22%	0,20%
3 – 50 – 6	53,25 menit	0,23%	0,22%
3 – 100 – 6	65,08 menit	0,21%	0,21%
3 – 200 – 6	68,00 menit	0,19%	0,18%
3 – 300 – 6	73,53 menit	0,22%	0,23%
3 – 400 – 6	77,73 menit	0,18%	0,17%
3 – 500 – 6	90,02 menit	0,27%	0,25%

5.5.3 Uji Coba Segmentasi

Uji coba segmentasi dilakukan untuk memastikan apakah *lookup table* yang telah dibuat berdasarkan hasil klasifikasi menggunakan *multilayer neural network* telah layak

diimplementasikan pada robot. Pada subbab ini, uji coba dilakukan dengan membandingkan tingkat *error* segmentasi dan waktu pemrosesan antara sistem lama (menggunakan metode *thresholding*) dan sistem yang digunakan pada tugas akhir ini.

Tiap piksel citra hasil segmentasi dari kedua sistem akan dibandingkan dengan *ground truth*. *Error* dihitung dengan menggunakan Persamaan 3.1. Dengan total data merupakan panjang citra dikalikan dengan lebar citra, dan prediksi salah merupakan jumlah piksel yang tidak sesuai dengan *ground truth*. Jumlah citra yang digunakan untuk uji coba adalah 4 citra, dengan *ground truth* merupakan citra uji coba yang telah dilakukan segmentasi secara manual oleh manusia yang didapatkan dengan bantuan perangkat lunak CorelDRAW X7.

Metode *thresholding* dilakukan dengan cara mengganti nilai piksel dengan nilai baru apabila nilai piksel lama berada di antara nilai konstan tertentu. Parameter tetap yang digunakan pada sistem lama dengan metode *thresholding* dapat dilihat pada Tabel 5.8.

Tabel 5.8 Parameter Tetap Metode Thresholding

Kelas	***Hue***		***Saturation***		***Value***	
	min	**max**	**min**	**max**	**min**	**max**
Hijau	37	64	37	121	125	255
Putih	0	255	0	50	177	255
Orange	60	125	52	255	202	255
Cyan	16	64	196	255	144	255
Magenta	127	148	66	215	121	240
Hitam	0	255	0	255	0	83

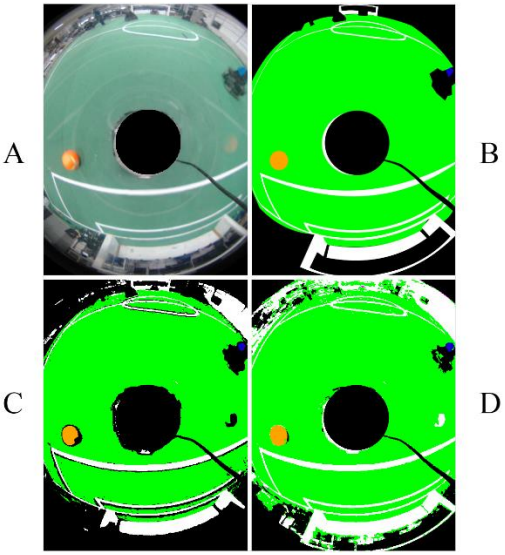
Berdasarkan hasil uji coba segmentasi, error rata-rata yang didapatkan pada sistem baru lebih besar daripada sistem lama. Hal ini dikarenakan banyaknya *noise* yang terdapat pada citra, seperti warna tembok yang mirip dengan garis lapangan. Namun, secara visual, terlihat bahwa hasil segmentasi

menggunakan sistem baru memberikan hasil yang lebih baik di dalam lapangan dibandingkan dengan sistem lama. Sedangkan kecepatan segmentasi menggunakan sistem baru lebih cepat daripada sistem lama. Hal ini dikarenakan operasi baca memori memakan waktu yang relatif singkat dibandingkan dengan melakukan operasi perhitungan dan operasi logika. Hasil dari uji coba segmentasi dapat dilihat pada Tabel 5.9. Sedangkan contoh citra hasil segmentasi dapat dilihat pada Gambar 5.3.

Tabel 5.9 Hasil Uji Coba Segmentasi

Keterangan	**Citra ke-**	***Run Time***	***Error***
Sistem Lama	1	0,1700s	9,55%
	2	0,1660s	12,75%
	3	0,1770s	8,87%
	4	0,1630s	13,18%
	Rata-rata	0,1690s	11,09%
Sistem Baru	1	0,0620s	14,92%
	2	0,0710s	16,05%
	3	0,0560s	12,96%
	4	0,0550s	33,57%
	Rata-rata	0,0610s	19,38%

Sistem lama yang dimaksud pada tugas akhir ini merupakan metode segmentasi yang telah diimplementasikan pada robot sebelumnya, yaitu menggunakan *thresholding*. Metode *thresholding* dilakukan dengan cara mengganti nilai piksel dengan nilai baru apabila nilai piksel lama berada di antara nilai konstan tertentu. Sedangkan sistem baru yang dimaksud pada tugas akhir ini ialah metode segmentasi yang akan diimplementasikan pada robot, yaitu menggunakan *lookup table* yang dibangun dari hasil klasifikasi tiap piksel pada ruang warna HSV menggunakan *multilayer neural network*.



Gambar 5.3 A. Citra Asli, B. Ground Truth, C. Hasil Segmentasi Menggunakan Sistem Lama, D. Hasil Segmentasi Menggunakan Sistem Baru

5.6 Analisis Hasil Uji Coba

Pada sub bab ini akan dilakukan analisis dari apa yang didapat dari uji coba yang dilakukan pada sub bab sebelumnya. Dari hasil uji coba yang telah dilakukan, dapat dilakukan beberapa analisis sebagai berikut:

1. Berdasarkan hasil *fine tuning hyperparameter*, didapatkan *error* pengujian terkecil pada nilai *learning rate* sebesar 0,3, dan nilai *lambda* sebesar 0.1.
2. Kemudian, dari serangkaian hasil uji coba, didapatkan model akhir dengan nilai *error* pada data testing sebesar 0,16%, atau mendapat nilai akurasi sebesar 99,84%. Pelatihan dari model tersebut berjalan selama 57,52 menit.
3. Berdasarkan hasil uji coba, selisih nilai *error training* dan *error testing* bernilai sangat kecil. Sehingga dapat

disimpulkan bahwa tidak terjadi *overfitting* pada model yang telah dibangun.

4. Berdasarkan uji coba segmentasi, didapatkan *error* rata-rata sebesar 19,38% pada sistem baru, lebih besar daripada sistem lama. Hal ini dikarenakan banyaknya *noise* yang terdapat pada citra, seperti warna tembok yang mirip dengan garis lapangan.
5. Kecepatan segmentasi menggunakan *lookup table* sedikit lebih cepat daripada *thresholding*. Hal ini dikarenakan operasi baca memori memakan waktu yang relatif singkat dibandingkan dengan melakukan operasi perhitungan dan operasi logika.

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, juga terdapat saran yang ditujukan untuk pengembangan penelitian lebih lanjut.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Telah berhasil dibangun model *multilayer neural network* untuk kasus klasifikasi tiap piksel pada ruang warna HSV ke dalam 6 kelas yang telah ditentukan.
2. Hasil segmentasi menggunakan *lookup table* yang telah dibuat berdasarkan hasil klasifikasi menggunakan *multilayer neural network* menunjukkan hasil yang cukup baik dengan rata-rata error sebesar 19,38%.
3. Kecepatan segmentasi menggunakan *lookup table* lebih cepat daripada *thresholding*. Sehingga layak diimplementasikan pada robot yang memiliki visi dengan karakteristik perpindahan lingkungan yang cepat.

6.2 Saran

Saran yang diberikan untuk pengembangan aplikasi ini adalah:

1. Penggunaan pustaka pihak ketiga yang mendukung komputasi menggunakan GPU pada tahap pelatihan *multilayer neural network*, sehingga waktu pelatihan dapat dilakukan dengan lebih cepat.

2. Perlu adanya proses dan pengembangan lebih lanjut sehingga hasil segmentasi dapat diolah menjadi data yang berguna bagi robot.

DAFTAR PUSTAKA

[1] A. J. R. Neves, A. J. Pinho, D. A. Martins and B. Cunha, "An Efficient Omnidirectional Vision System For Soccer Robots: From Calibration To Object Detection," *Mechatronics*, vol. 21, pp. 399-410, 2011.

[2] D. J. P. d. K. Direktorat Kemahasiswaan, "Buku Panduan Kontes Robot Sepakbola Indonesia Divisi Beroda," Kementerian Riset, Teknologi dan Pendidikan Tinggi Republik Indonesia, 2018.

[3] Y. Prakoso, "Desain Dan Implementasi Pengukuran Posisi Bola Menggunakan Kamera 360 Derajat Pada Robot Sepak Bola," Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya, 2017.

[4] "OpenCV: Color conversions," [Online]. Available: https://docs.opencv.org/3.4.1/de/d25/imgproc_color_conversions.html#color_convert_rgb_hsv. [Accessed 27 Juni 2018].

[5] A. Rachmawan, "Penentuan Posisi Robot Sepak Bola Beroda Menggunakan Rotary Encoder dan Kamera," Fakultas Teknologi Elektro, Institut Teknologi Sepuluh Nopember, Surabaya, 2017.

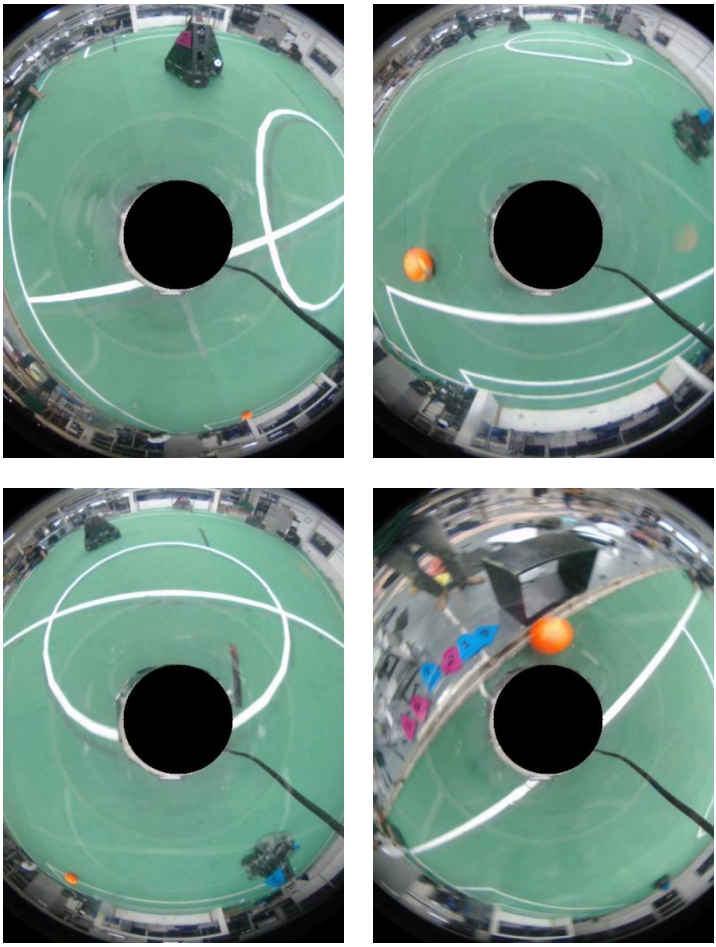
[6] J. Bruce, T. Balch and M. Veloso, "Fast and Inexpensive Color Image Segmentation for Interactive Robots," in *Intelligent Robots and Systems, 2000. (IROS 2000).*, Takamatsu, Japan, 2000.

[7] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

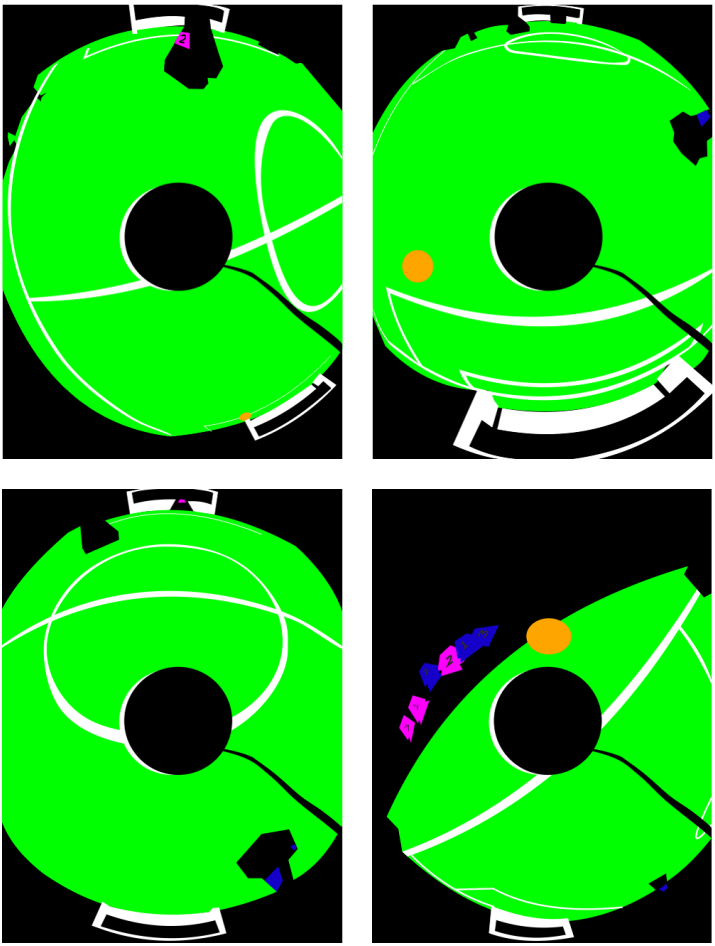
[8] M. A. Morshidi, M. H. Marhaban and A. Jantan, "Color segmentation using multi layer neural network and the HSV color space," in *International Conference on Computer and Communication Engineering, 2008. ICCCE 2008*, Kuala Lumpur, Malaysia, 2008.

[Halaman ini sengaja dikosongkan]

LAMPIRAN

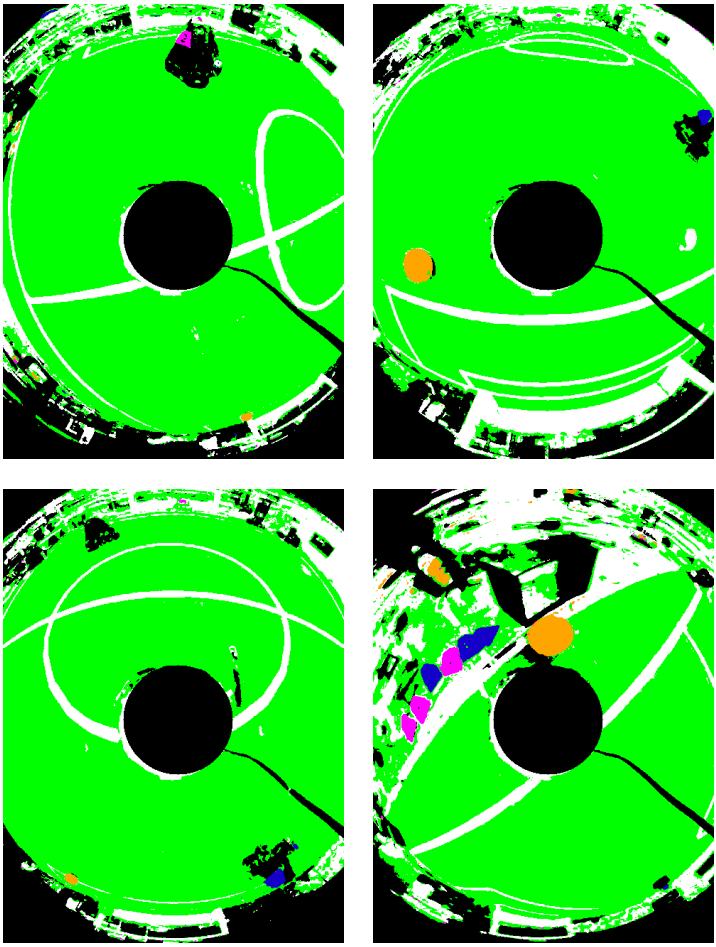


Gambar A. 1 Contoh Dataset



Gambar A. 2 Ground Truth

Gambar A. 3 Hasil Segmentasi Menggunakan Sistem Lama



Gambar A. 4 Hasil Segmentasi Menggunakan Sistem Baru

BIODATA PENULIS

Alam Ar Raad Stone, lahir di Pacitan, pada tanggal 9 Maret 1997. Penulis menyelesaikan pendidikan dasar di SDN Bungur 1 (2001 – 2008), dilanjutkan dengan pendidikan menengah di MTs Darul Huda (2008 – 2011) dan MA Darul Huda (2011 – 2014). Penulis memulai pendidikan S1 Informatika di Institut Teknologi Sepuluh Nopember pada tahun 2014.

Selama menempuh pendidikan di kampus, penulis aktif mengikuti kegiatan Robotika ITS sebagai programmer tim Robot Sepak Bola Beroda IRIS ITS (sebelumnya bernama Aljazari). Penulis mengambil bidang minat Komputasi Cerdas dan Visi (KCV). Komunikasi dengan penulis dapat dilakukan melalui email: **alamstone@outlook.com**.