KATA PENGANTAR

Puji dan Syukur penulis panjatkan ke hadirat Allah SWT, yang telah melimpahkan segala petunjuk dan rahmat-Nya sehingga penulis dapat menyelesaikan kewajiban menyusun tugas akhir ini. Penyusunan tugas akhir ini merupakan salah satu syarat untuk menyelesaikan program pendidikan Sarjana Strata-1 (S1) Jurusan Teknik *Elektro Universitas Brawijaya Malang*. Adapun judul dari laporan tugas akhir ini yaitu **"PENGENALAN KARAKTER TEKS MENGGUNAKAN METODE *NEURAL NETWORK BACKPROPAGATION*".**

Penulis menyadari bahwa dalam penyusunan laporan tugas akhir ini masih terdapat banyak kekurangan, baik dalam materi maupun teknisnya. Hal ini disebabkan pengetahuan dan kemampuan penulis yang terbatas. Karenanya, dengan rendah hati penulis menerima kritik dan saran dari pembaca yang sifatnya membangun dan memperbaiki sehingga dapat mengarah kepada penyusunan yang lebih baik.

Penulis juga menyadari bahwa penulisan laporan tugas akhir ini tidak dapat diselesaikan dengan baik tanpa adanya bantuan dan kerja sama dari berbagai pihak. Pada kesempatan ini penulis mengucapkan banyak terima kasih kepada Ayah, Ibu serta kakak yang selalu mendo'akan dan senantiasa memberikan dorongan, mengingatkan dan mendukung baik moril maupun materiil dalam berbagai aktifitas selama ini, serta kasih sayang dan pengertian yang telah mereka berikan kepada penulis, khususnya dalam menyelesaikan skripsi ini. Serta tidak lupa penulis mengucapkan terima kasih kepada yang terhormat :

1. Bapak Adharul Muttaqin, ST., MT. Dan Bapak Mochammad Rif'an, ST., MT. selaku dosen pembimbing yang telah memberikan bimbingan dan pengarahan-pengarahan dalam proses penyusunan tugas akhir ini, sehingga dapat menambah ilmu bagi penulis.
2. Untuk teman-teman seperjuangan Rachmania Nur D, Rizal Arif Z, Ita Dwi P, Suci Imani P, Ach. Fajaruddin A., serta seluruh warga besar CORE terima kasih atas bantuan dan kerjasamanya selama ini.
3. Serta semua pihak yang tidak bisa penulis sebutkan satu persatu, yang telah membantu terselesaikannya laporan tugas akhir ini.

Akhir kata, semoga Allah SWT melimpahkan semua rahmat dan karunia kepada semua pihak yang telah membantu terselesaikannya Tugas Akhir ini. Dan semoga Tugas Akhir ini dapat bermanfaat bagi para pembacanya.

Malang, 21 Agustus 2013

Penulis

ABSTRAK

Titis Hayuning W.P, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, Januari 2014, *Pengenalan Karakter Teks Menggunakan Metode Neural Network Backpropagation,* Dosen Pembimbing : Adharul Muttaqin, ST., MT. dan Mochammad Rif'an, ST., MT.

Pengenalan karakter (*Optical Character Recognition*) merupakan salah satu cabang dari pengenalan pola.Salah satu metode pengenalan pola yang saat ini banyak digunakan adalah *Neural Network Backpropagation* yang sistem kerjanya seperti otak manuasia yang dapat mengenali pola setelah mendapat training atau pelatihan terhadap beberapa contoh yang diberikan terlebih dahulu.

Dalam pengenalan pola perlu didapatkan beberapa fitur dari karakter itu sendiri. Fitur yang diambil pada sebuah karakter meliputi jumlah segmen pembentuk karakter dan bentuk segmennya yang secara garis besar dibedakan menjadi garis, kurva, atau loop. Pengambilan fitur menggunakan metode fuzzy untuk dapat membedakan bentuk-bentuk segmennya sesuai dengan parameter yang diberikan. Sistem pengenalan karakter ini memiliki prosentase keberhasilan dalam pengenalan karakter sekitar 80.12% dengan kemampuannya mengenali 647 data dari 800 data yang terdiri atas data training dan data uji.

Kata kunci: *Backpropagation, Neural Network,* OCR, pengenalan pola.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di era yang serba terkomputerisasi seperti saat ini, mendorong perkembangan informasi dan pengetahuan dalam bentuk digital. Namun, adakalanya suatu data yang ada hanya terdapat dalam bentuk fisik dan dikehendaki untuk didigitalisasi. Untuk mengatasi permasalahan tersebut, maka diperlukan suatu sistem komputer yang dapat mengenali/membaca huruf, di mana masukannya dapat berupa teks cetak maupun tulisan tangan. Sistem komputer seperti ini umum dikenal dengan nama OCR (Optical Character Recognition).

OCR termasuk pada pengenalan pola (*pattern recognition*) di mana sebuah sistem pengenal pola akan mencoba membaca / mengenali apakah citra masukan yang diterima cocok dengan salah satu citra yang telah ditentukan. Pengenalan pola banyak diaplikasikan untuk mendeteksi sidik jari, tulisan, tanda tangan, bahkan wajah seseorang. Namun, sistem aplikasi OCR yang akan dibangun ini adalah khusus untuk mengenali tulisan teks cetak yang diketik dan dicetak menggunakan printer. Dokumen fisik tersebut adakalanya dikehendaki menjadi dokumen digital, saat softcopy dari data tersebut hilang atau terhapus, OCR dapat dengan mudah membantu mengatasi problem tersebut.

Dalam sistem pengenalan pola, perlu suatu metode dan algoritma yang "pintar", dalam artian metode komputasi tersebut dapat berdiri sendiri dalam mengambil keputusan untuk jangka waktu yang panjang, sehingga output yang diinginkan sesuai dengan harapan *developer*. *Neural Networks* atau Jaringan Saraf Tiruan merupakan salah satu metode komputasi yang banyak digunakan dalam membangun sistem pengenalan pola. Karena prinsip kerjanya yang meniru jaringan saraf biologis, di mana proses pengenalan pola didapat dari proses *learning*, sehingga sistem dapat digunakan untuk input data yang dinamis dengan output mendekati hasil yang diharapkan.

Algoritma *Backpropagation* digunakan untuk meminimalisasi error yang mungkin terjadi saat proses pengenalan pola, karena algoritma ini menghitung setiap perubahan *error weight* (EW) yang terjadi antara output hitungan dengan output yang sebenarnya. Berdasarkan latar belakang yang telah dipaparkan di atas, maka penulis mengangkat judul "Pengenalan Karakter Teks Menggunakan Metode *Neural Network Backpropagation*".

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, maka ditemukan beberapa rumusan masalah sebagai berikut :

1. Bagaimana mengenali teks cetak dengan menggunakan metode *neural network backpropagation.*
2. Bagaimana pengambilan fitur untuk membedakan karakter yang mirip.
3. Bagaimana perancangan, implementasi dan pengujian aplikasi sistem pengenalan karakter teks cetak menggunakan metode *neural networks backpropagation.*

1.3 Batasan Masalah

Beberapa batasan masalah yang ada pada tugas akhir (skripsi) ini adalah :

1. Sistem ini menangani pengenalan berdasarkan karakter yang terdiri atas huruf latin besar dan huruf latin kecil (A-Z, a-z), angka arab (0-9), dan 10 simbol / tanda baca.
2. Masukan sistem adalah berupa citra dokumen hasil *scanning* kaarakter cetak yang jelas dan tegak, tidak miring ataupun buram.
3. Karakter berupa karakter cetak dengan tipografi jenis Times New Roman, Arial, dan Tahoma.

1.4 Tujuan

Tujuan penyusunan tugas akhir (skripsi) ini adalah membangun suatu aplikasi yang mampu mengenali karakter teks cetak dengan tingkat akurasi

optimal menggunakan *neural networks backpropagation* dan menggunakan ekstraksi fitur bentuk geometri.

1.5 Sistematika Penulisan

Sistematika penulisan laporan skripsi ini adalah sebagai berikut :

BAB I Pendahuluan

Menjelaskan latar belakang, rumusan masalah, batasan masalah, tujuan dan manfaat.

BAB II Dasar Teori

Menjelaskan kajian pustaka dan dasar teori yang digunakan pada skripsi ini.

BAB III Metodologi

Menjelaskan metode yang digunakan dalam pengerjaan skripsi berikut langkah – langkah yang akan diambil.

BAB IV Perancangan dan Implementasi

Menjelaskan langkah langkah perancangan dan hasil implementasi aplikasi pengenalan karakter teks cetak berikut penjelasan algoritma yang digunakan dan penjelasan dari setiap langkah – langkahnya.

BAB V Pengujian dan Analisis

Membahas pengujian dari sistem yang telah dibuat berikut analisa dari hasilnya sehingga dapat diketahui kelebihan dan kekurangan sistem.

BAB VII Kesimpulan dan Saran

Berisi kesimpulan hasil penulisan dan saran untuk pengembangan sistem selanjutnya.

BAB II
DASAR TEORI

Bab ini menjelaskan tentang kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi dan melandasi pelaksanan penelitian aplikasi pengenalan karakter teks cetak.

2.1. Tipografi

Tipografi merupakan seni memilih dan menata huruf dengan pengaturan penyebarannya pada ruang-ruang yang tersedia, untuk menciptakan kesan khusus, sehingga akan menolong pembaca untuk mendapatkan kenyamanan membaca semaksimal mungkin.

Saat ini tippografi mengalami perkembangan dari fase penciptaan dengan tangan hingga mengalami komputerisasi. Fase komputerisasi membuat penggunaan tipografi menjadi lebih mudah dan dalam waktu yang cepat jenis pilihan huruf yang ratusan jumlahnya.

2.1.1. Jenis-Jenis Huruf

Bila diperhatikan dari bentuk struktur gambar huruf dapat dikelompokkan menjadi 3 golongan yaitu kelompok Serif, Sans Serif, dan Fantasi.

a. Jenis Huruf Serif

Serif adalah garis tipis yang ada pada ujung kaki atau lengan huruf. Jenis kelompok huruf Serif berarti kelompok huruf yang memiliki kaki atau lengan huruf. Bila diperhatikan kelompok jenis ini terdiri dari tiga gaya yaitu huruf Roman, huruf Bodoni, dan huruf Egyption. Jenis huruf yang termasuk huruf Serif antara lain: Time Roman, Book Antiqua, Courier New, Leter Gothic, dll.

b. Jenis Huruf Sans Serif

Huruf Sans Serif adalah huruf yang tidak mempunyai kaki dan lengan huruf. Perbedaan antar tebal dan tipisnya boleh dikatakan tidak ada. Kesan jenis ini sangat sesuai dengan pekerjaan halus yang memberi kesan sederhana, tidak ramai namun tetap manis. Pada komunikasi visual, jenis huruf tanpa kait ini berhasil menarik

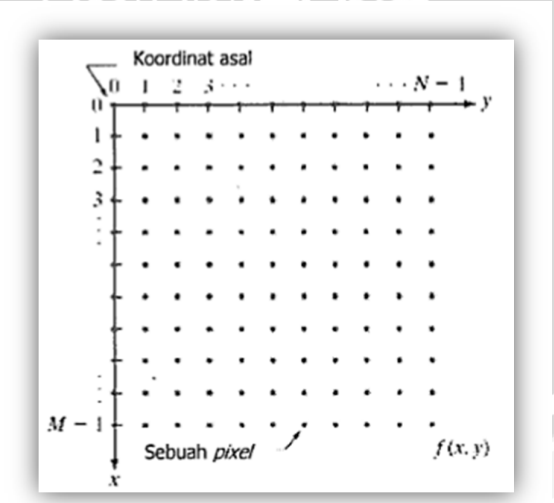
perhatian banyak orang terutama tampilan pada wajah judul. Jenis huruf yang termasuk Sans Serif antara lain: Mercator, Gill Sans, Univers, Futura, Helvetica, Antique, Eurostile, Lucida Sans, dll.

c. Jenis Fantasi

Huruf fantasi merupakan huruf yang penuh lekuk-lekuk seperti tunas menjalar. Jenis huruf ini sering dipergunakan untuk hiasan pada kata atau kalimat yang berfungsi sebagai penarik perhatian pembaca. Dalam penampilannya, jenis huruf ini tidak huruf kapital semua, karena bila tampil dengan huruf kapital semua akan mengganggu penglihatan atau kelihatan kaku dan sulit dibaca. Jenis huruf ini antara lain: Script, Astral, Inline, Lincoin, Neon, dll.

2.2 Konsep Dasar Citra Digital

Citra digital merupakan sebuah larik (array) yang berisi nilai-nilai real maupun kompleks yang direpresentasikan dengan deretan bit tertentu. Suatu citra dapat didefinisikan sebagai fungsi *f(x,y)* berukuran M baris dan N kolom, dengan x dan y adalah koordinat spasial, amplitudo f di titik koordinat *(x,y)* dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x, y, dan nilai amplitudo f secara keseluruhan berhingga (*finite*) dan bernilai diskrit maka dapat dikatakan bahwa citra tersebut adalah citra digital.



Gambar 2.1 Koordinat citra digital

Dari koordinat di atas, citra digital dapat ditulis dalam bentuk matrik sebagai berikut:

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

Gambar 2.2 Bentuk matrik dari citra digital

Nilai pada suatu irisan antara baris dan kolom (pada posisi x,y) disebut dengan *picture elements*, *image elements, pels,* atau pixels. Istilah terakhir (pixel) paling sering digunakan pada citra digital.

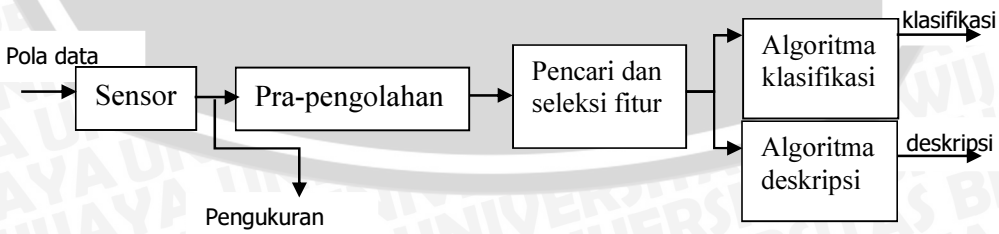
2.3 Pengenalan Pola (*Pattern Recognition*)

Teknik pengenalan pola merupakan salah satu komponen penting dari mesin atau sistem cerdas (*machine intelligence*) yang digunakan baik untuk mengolah data maupun dalam pengambilan keputusan.

Secara umum pengenalan pola (*pattern recognition*) adalah suatu ilmu untuk mengklasifikasikan atau menggambarkan sesuatu berdasarkan pengukuran kuantitatif fitur (ciri) atau sifat utama dari suatu objek. Pola sendiri adalah suatu entitas yang terdefinisi dan dapat diidentifikasikan dan diberi nama. Sidik jari adalah suatu contoh pola. Pola bisa dinyatakan dalam notasi vektor atau matriks.

2.3.1 Struktur Sistem Pengenalan Pola

Secara umum, struktur sistem pengenalan pola dapat digambarkan dengan diagram di bawah ini:



Gambar 2.3 Struktur sistem pengenalan pola

a. *Sensor* berfungsi untuk menangkap objek dari dunia nyata dan selanjutnya diubah menjadi sinyal digital melalui proses digitalisasi.

b. *Pra-pengolahan* berfungsi mempersiapkan citra atau sinyal agar dapat menghasilkan ciri yang lebih baik pada tahap berikutnya. Pada tahap ini sinyal informasi ditonjolkan dan sinyal pengganggu (derau) diminimalisasi.

c. *Pencari dan seleksi* fitur berfungsi menemukan karakteristik pembeda yang mewakili sifat utama sinyal sekaligus mengurangi dimensi sinyal menjadi sekumpulan bilangan yang lebih sedikit tetapi representatif.

d. *Algoritma klarifikasi* berfungsi untuk mengelompokkan fitur ke dalam kelas yang sesuai.

e. *Algoritma deskripsi* berfungsi memberikan deskripsi pada sinyal.

2.3.2 Pendekatan Pengenalan Pola

Terdapat 3 macam pendekatan pengenalan pola, yaitu:

a. Pendekatan pengenalan pola statistikal (StatPR)

Pengenalan pola statistikal memiliki asumsi suatu basis statistik untuk algoritma klasifikasi, sehingga pola dipilah berdasarkan model statistik dari fitur.

b. Pendekatan pengenalan pola sintatik (SyntPR)

Pada pendekatan ini, pola dipilah berdasarkan keserupaan ukuran struktural.

c. Pendekatan pengenalan pola neural (NeuroPR)

Pendekatan ini menggunakan metode jaringan saraf tiruan (JST) untuk mengidentifikasi pola, sehingga pemilahan pola dilakukan berdasarkan tanggapan suatu neuron jaringan pengolah sinyal (neuron) terhadap stimulus masukan (pola).

2.4 Logika Fuzzy

Logika fuzzy adalah suatu cara yang tepat untuk memetakan ruang input kedalam suatu ruang output. Konsep ini diperkenalkan dan dipublikasikan pertama kali oleh Lotfi A.Zadeh, seorang professor dari University of California di Berkeley pada tahun 1965. Logika fuzzy menggunakan ungkapan bahasa untuk menggambarkan nilai variable. Logika fuzzy bekerja dengan menggunakan derajat keanggotaan dari sebuah nilai yang kemudian digunakan

untuk menentukan hasil yang ingin dihasilkan berdasarkan atas spesifikasi yang telah ditentukan.

Konsep logika fuzzy ini dikembangkan karena beberapa alasan, yang diantaranya adalah :

1. Keanggotaan suatu elemen terhadap suatu himpunan menjadi "fuzzy" atau "samar". Dalam hal ini, ada elemen yang posisinya sebagai anggota atau bukan anggota dari suatu himpunan tidak diketahui dengan pasti.
2. Kaeanggotaan elemen dalam suatu himpunan pasti sangat curam dan mempunyai batasan yang kaku.

Penilaian yang dilakukan dengan logika klasik hanya mungkinkan untuk mengolah informasi yang benar atau salah. Logika ini tidak mampu menangani informasi yang tidak pasti, walaupun di dalam informasi tersebut ada data yang tersimpan. Dalam logika klasik, keanggotaan suatu elemen dalam himpunan dipresentasikan dengan 0 bila bukan anggota dan 1 bila merupakan anggota. Jadi himpunannya adalah {0,1}. Sedangkan pada logika fuzzy nilai keanggotaan ada pada interval [0,1]. Jadi pada dasarnya system fuzzy merupakan gagasan aproksimasi yang didasarkan pada gagasan eksak.

2.5 *Neural Network*

Neural Network (NN) adalah suatu metode komputasi yang meniru sistem jaringan saraf biologis. Metode ini menggunakan elemen perhitungan nonlinier dasar yang disebut neuron yang diorganisasikan sebagai jaringan yang saling berhubungan, sehingga mirip dengan jaringan saraf manusia. NN dibentuk untuk memecahkan suatu masalah tertentu seperti pengenalan pola atau klasifikasi karena proses pembelajaran.

Seperti halnya dalam neuron biologi, NN juga merupakan sistem yang bersifat menoleransi kesalahan dalam (*fault tolerant*) 2 hal. Pertama, dapat mengenali sinyal input yang agak berbeda dari yang pernah diterima sebelumnya. Kedua, tetap mampu bekerja meskipun beberapa neuronnya tidak mampu bekerja dengan baik. Jika sebuah neuron rusak, neuron lain dapat dilatih untuk menggantikan fungsi neuron yang rusak tersebut.

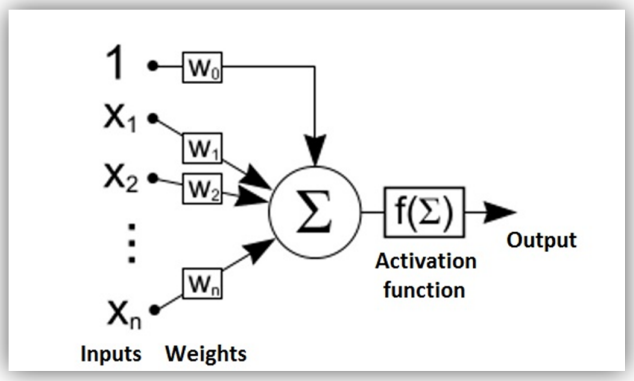
Neural Networks, seperti halnya dengan manusia, belajar dari suatu contoh

karena mempunyai karakteristik yang adaptif, yaitu dapat belajar dari data-data sebelumnya dan mengenal pola data yang selalu berubah. Selain itu, NN merupakan sistem yang tak terprogram, maksudnya semua keluaran atau kesimpulan yang ditarik oleh jaringan didasarkan pada pengalamannya selama mengikuti proses pembelajaran/pelatihan.

Hal yang ingin dicapai dengan melatih NN adalah untuk mencapai keseimbangan antara kemampuan *memorisasi* dan *generalisasi.* Kemampuan *memorisasi* adalah kemampuan NN untuk mengambil kembali secara sempurna sebuah pola yang telah dipelajari. Sedangkan kemampuan *generalisasi* adalah kemampuan NN untuk menghasilkan respons yang bisa diterima terhadap pola-pola input yang serupa (namun tidak identik) dengan pola-pola yang sebelumnya telah dipelajari. Hal ini sangat bermanfaat bila suatu saat diinputkan informasi baru yang belum pernah dipelajari ke dalam NN, maka NN masih akan tetap dapat memberikan tanggapan yang baik, memberikan keluaran yang paling mendekati.

NN menyerupai otak manusia dalam dua hal, yaitu:

a. Pengetahuan diperoleh jaringan melalui proses belajar.
b. Kekuatan hubungan antar sel saraf (neuron) yang dikenal sebagai bobot-bobot sinaptik digunakan untuk menyimpan pengetahuan.



Gambar 2.4 Model neuron

Dalam prosesnya, keberhasilan *Neural Network* ditentukan oleh 3 hal, yaitu:

a. Pola hubungan antar neuron (arsitektur jaringan).
b. Metode untuk menentukan bobot penghubung (metode *training/learning*).

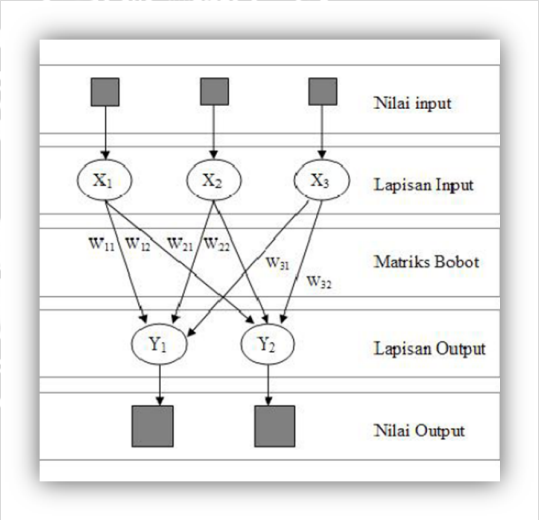
c. Fungsi aktivasi, yaitu fungsi yang digunakan untuk menentukan keluaran suatu neuron.

2.5.1 Arsitektur Jaringan NN

Berdasarkan banyaknya lapisan (*layer*) pada jaringan NN, maka terdapat 3 macam arsitektur jaringan yang dikenal, yaitu:

a. Single Layer

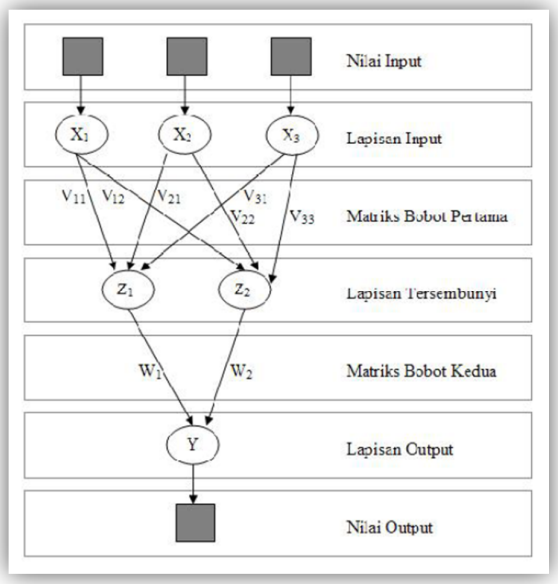
Jaringan dengan lapisan tunggal terdiri dari 1 layer input dan 1 laryer output. Setiap neuron/unit yang terdapat di dalam layer input selalu terhubung dengan setiap neuron yang terdapat pada layer output. Jaringan ini hanya menerima input kemudian secara langsung akan mengolahnya menjadi output tanpa harus melalui lapisan tersembunyi. Contoh algoritma yang menggunakan metode ini adalah Adaline, Hopfield, dan Perceptron.



Gambar 2.5 *Single Layer*

b. Multilayer

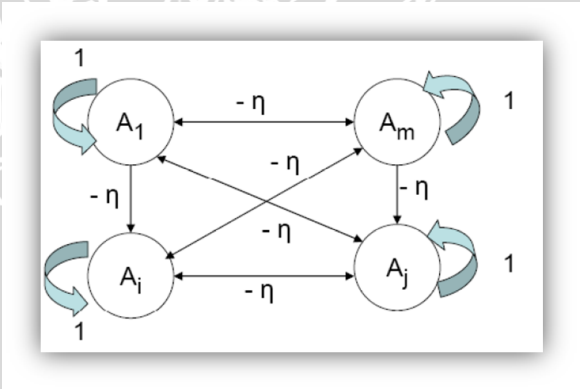
Jaringan dengan lapisan jamak memiliki ciri khas tertentu yaitu memiliki 3 jenis layer yakni layer input, layer tersembunyi, dan layer output. Jaringan dengan lapisan banyak ini dapat menyelesaikan permasalahan yang lebih kompleks dibandingkan jaringan dengan lapisan tunggal. Namun, proses pelatihan sering membutuhkan waktu yang cenderung lama. Contoh algoritma yang menggunakan metode ini adalah Madaline, *Backpropagation, Neocognitron.*



Gambar 2.6 *Multi Layer*

c. Competitive Layer

Pada jaringan ini sekumpulan neuron bersaing untuk mendapatkan hak menjadi aktif. Contoh algoritma yang menggunakan metode ini adalah LVQ.



Gambar 2.7 *Competitive Layer*

2.5.2 Proses Pembelajaran

Semua metode pembelajaran yang digunakan untuk Jaringan Saraf adaptif dapat diklasifikasikan ke dalam dua kategori utama, yaitu:

a. Supervised Learning

Pembelajaran ini menggabungkan pengajar eksternal, sehingga setiap unit output diberitahu respon apa yang seharusnya diinginkan oleh

sinyal input. Paradigma pembelajaran terawasi termasuk pada koreksi *error learning*, penguatan pembelajaran, dan pembelajaran stokastik.

Hal penting dari supervised learning ini adalah masalah konvergensi kesalahan, yaitu minimalisasi *error* antara nilai satuan yang diinginkan dengan hasil hitungan. Tujuannya adalah untuk menentukan set bobot yang dapat meminimalisasi *error*.

b. Unsupervised Learning

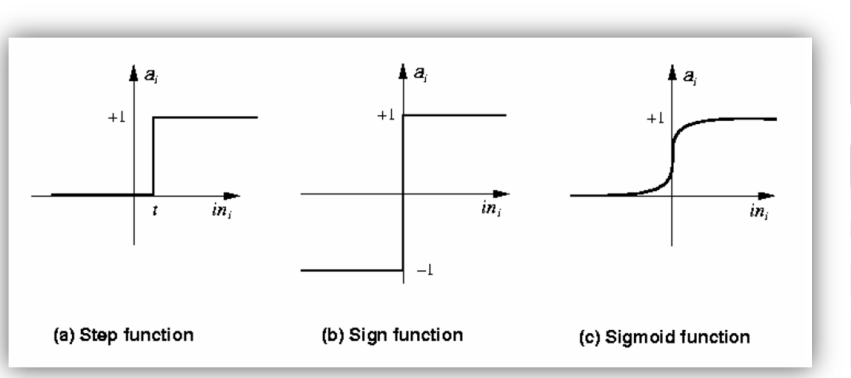
Pembelajaran ini tidak menggunakan pengajar eksternal dan hanya berdasarkan informasi lokal, hal ini disebut juga *self-organisation*, di mana mereka mengatur sendiri data yang disajikan ke jaringan.

c. Hybrid Learning

Merupakan kombinasi dari dua pembelajaran di atas. Sebagian dari bobot-bobotnya ditentukan melalui *supervised learning* dan sebagian lainnya dari *unsupervised learning*.

2.5.3 Fungsi Aktivasi

Mengaktifkan jaringan saraf tiruan berarti mengaktifkan setiap neuron yang dipakai pada jaringan tersebut. Banyak fungsi yang dapat dipakai sebagai pengaktif, seperti fungsi-fungsi goniometri dan hiperboliknya, fungsi unit step, impulse, sigmoid, dan lain sebagainya seperti pada Gambar 2.8, tetapi yang umum dipakai adalah fungsi sigmoid, karena dianggap lebih mendekati kinerja sinyal otak.



Gambar 2.8 Macam-macam fungsi aktivasi

Keterangan gambar:

a. Step(x) $\begin{cases} 1, \text{untuk } (x >= t) \\ 0, \text{untuk } (x < t) \end{cases}$

b. Sign(x) $\begin{cases} 1, \text{untuk } (x >= 0) \\ -1, \text{untuk } (x < 0) \end{cases}$

c. Sigmoid(x)

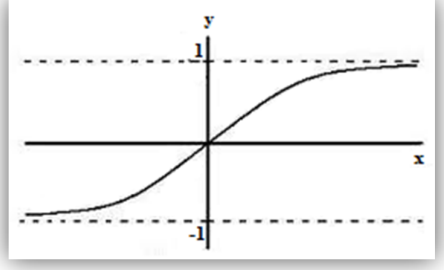
Ada dua jenis fungsi sigmoid, unipolar dan bipolar. Fungsi sigmoid unipolar grafiknya seperti yang ditunjukkan pada Gambar 2.8 (c), dengan persamaan fungsinya seperti di bawah ini

$$y = \frac{1}{1 + e^{-x}}$$

Untuk fungsi sigmoid bipolar disebut juga sebagai persamaan tangen hiperbolik. Persamaan fungsinya seperti di bawah ini:

$$y = \frac{1 - e^{-x}}{1 + e^{-x}}$$

grafik fungsinya ditunjukkan seperti pada gambar 2.9.



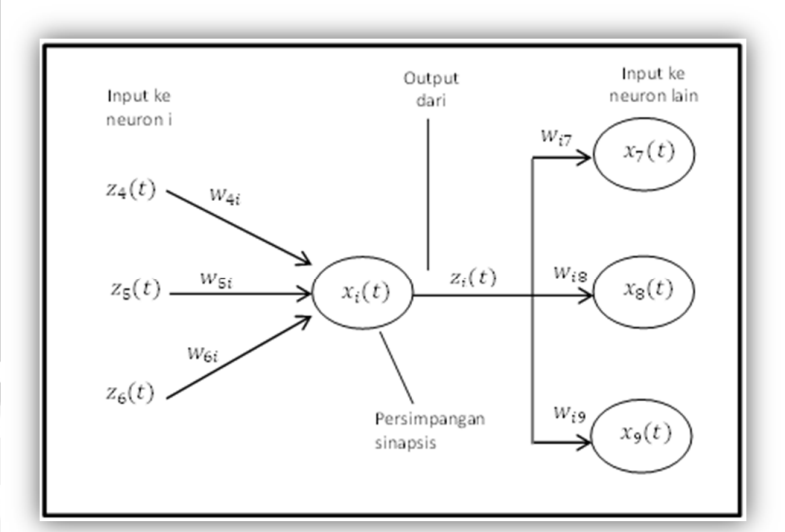
Gambar 2.9 Grafik fungsi sigmoid bipolar

2.5.4 Algoritma *Backpropagation*

Dalam rangka untuk melatih jaringan saraf melakukan beberapa tugas, maka bobot setiap unit harus disesuaikan sedemikian rupa sehingga error antara output yang diinginkan dan output yang sebenarnya berkurang. Proses ini mensyaratkan bahwa jaringan saraf menghitung turunan kesalahan dari bobot / *error weigths* **(EW).** Dengan kata lain, ia harus menghitung bagaimana perubahan *error weigths* masing-masing meningkat atau menurun sedikit. Algoritma *backpropagation* adalah metode yang paling banyak digunakan untuk menentukan **EW** tersebut.

Algoritma *backpropagation* paling mudah untuk dipahami jika semua unit dalam jaringan linear. Algoritma menghitung masing-masing **EW** dengan menghitung EA pertama**,** tingkat di mana perubahan kesalahan sebagai tingkat aktivitas unit berubah. Untuk unit output, **EA** adalah perbedaan antara aktual dan output yang diinginkan. Untuk menghitung **EA** pada unit tersembunyi di lapisan sebelum lapisan output, pertama-tama kita mengidentifikasi semua bobot antara unit tersembunyi dan unit output yang terhubung. Kemudian dikalikan oleh bobot **EA** dari unit-unit output dan menambahkan hasil. Jumlah ini sama dengan **EA** untuk unit tersembunyi yang dipilih. Setelah menghitung semua **EA** dalam lapisan tersembunyi sebelum lapisan output, kita dapat menghitung **EA** untuk lapisan lainnya, bergerak dari lapisan ke lapisan dalam arah berlawanan dengan cara menyebarkan melalui jaringan. Inilah apa yang disebut dengan *backpropagation*. Setelah **EA** dihitung untuk sebuah unit, maka selanjutnya menghitung **EW** untuk setiap koneksi yang masuk unit. **EW** adalah hasil dari EA dan aktivitasnya melalui koneksi yang masuk.

Untuk unit non-linear, algoritma *backpropagation* mencakup langkah tambahan. Sebelum menyebarkan kembali, **EA** harus dikonversi ke dalam **EI,** tingkat di mana perubahan kesalahan sebagai masukan total yang diterima oleh unit yang berubah.



Gambar 2.10 Pola konektivitas ciri khas pada arsitektur jaringan.

Sebuah unit pada lapisan output menentukan aktivitas dengan mengikuti dua langkah prosedur, yaitu:

1. Pertama, menghitung total bobot masukan X_j, dengan menggunakan rumus:

$$X_j = \sum_i (y_i W_{ij})$$

dimana y_i adalah tingkat aktivitas unit *j* pada lapisan sebelumnya dan W_{ij} adalah berat dari koneksi antara unit *i* dan unit *j*.

2. Selanjutnya, unit menghitung aktivitas y_j menggunakan beberapa fungsi dari bobot total input dengan menggunakan fungsi sigmoid unipolar:

$$y_j = \frac{1}{1 + e^{-x_j}}$$

Setelah kegiatan semua unit output telah ditentukan, jaringan menghitung erorr (E) , yang didefinisikan oleh persamaan:

$$\mathrm{E} = \frac{1}{2} \sum_i (y_i - d_i)^2$$

di mana y_i adalah tingkat aktivitas unit *j* pada lapisan atas dan d_i adalah output yang diinginkan oleh unit *j*.

Algoritma *backpropagation* terdiri dari empat langkah, yaitu:

1. Hitung seberapa cepat perubahan *error* sebagai aktivitas unit output yang berubah. Derivatif *error* ini (EA) adalah perbedaan antara aktual dan aktivitas yang diinginkan.

$$EA_j = \frac{\partial E}{\partial y_j} = y_j - d_j$$

2. Hitung seberapa cepat perubahan *error* sebagai masukan total yang diterima oleh unit output yang diubah. Kuantitas (EI) ini adalah jawaban dari langkah 1 dikalikan dengan tingkat di mana output dari perubahan unit sebagai masukan total berubah.

$$EI_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{dy_j}{dx_j} = EA_j y_j (1 - y_j)$$

2. Hitung seberapa cepat perubahan *error* sebagai bobot pada koneksi ke unit output yang diubah. Kuantitas (EW) ini adalah jawaban dari

langkah 2 dikalikan oleh tingkat aktivitas dari unit yang mana merupakan asal koneksi.

$$EW_{ij} = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{dx_j}{dW_{ij}} = EI_j y_i$$

4. Hitung seberapa cepat perubahan *error* sebagai aktivitas unit di lapisan sebelumnya berubah. Langkah penting memungkinkan propagasi kembali diterapkan untuk jaringan *multilayer*. Ketika aktivitas unit dalam perubahan lapisan sebelumnya, hal tersebut mempengaruhi aktivitas dari semua unit output yang terhubung. Jadi untuk menghitung efek keseluruhan pada *error*, ditambahkan secara bersama-sama semua efek terpisah pada unit output. Tapi masing-masing efek ini mudah untuk dihitung. Ini adalah jawaban pada langkah 2 dikalikan dengan bobot koneksi ke unit output.

$$EA_i = \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{dx_j}{dy_i} = \sum_j EI_j W_{ij}$$

2.6 MATLAB

Nama Matlab merupakan akronim dari kata *Matrix Laboratory*. Versi pertama Matlab ditulis pada tahun 1970. Saat itu, Matlab digunakan untuk pelatihan dalam teori matrik, aljabar linier, dan analisis numerik. Pada tahun sebelumnya, Matlab telah direvisi. Fungsi-fungsi matlab ini digunakan untuk menyelesaikan masalah bagian khusus, yang disebut *toolboxes.* Toolboxes dapat digunakan untuk bidang pengolahan sinyal, sistem pengaturan, *fuzzy logic*, *numeral networks,* optimasi, pengolahan citra, dan simulasi lainnya.

Matlab merupakan sistem interaktif dari sebuah program bahasa. Elemen data dasar merupakan sebuah matrik yang tidak membutuhkan deklarasi ukuran atau jenis data. Oleh karena itu, banyak masalah perhitungan dapat diselesaikan pada waktu singkat dan perhitungan diambil untuk dituliskan ke dalam bahasa Fortran atau C.

BAB III
METODOLOGI

Bab ini menjelaskan mengenai langkah-langkah yang dilakukan untuk membuat aplikasi pengenalan karakter teks cetak menggunakan metode *Neural Network Backpropagation*. Berikut ini menjabarkan metode penelitian yang digunakan dalam penyusunan skripsi ini.

3.1 Studi Literatur

Studi literatur berguna untuk memperoleh data dan menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

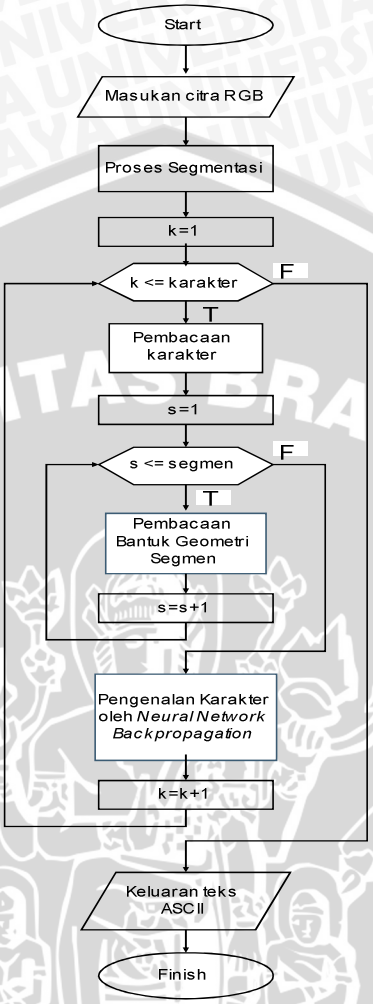
1. Mempelajari *image processing*.
2. Mempelajari ekstraksi fitur menggunakan metode fuzzy.
3. Mempelajari *Neural Network Backpropagation*.
4. Mempelajari bahasa pemrograman *MatLab*.

3.2 Diagram Sistem

Gambar 3.1 Diagram Sistem Pengenalan Karakter

3.3 Cara Kerja Sistem

Gambar 3.2 merupakan diagram alir cara kerja sistem.



Gambar 3.2 Diagram alir sistem pengenalan karakter

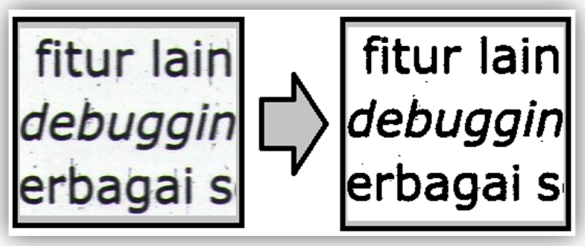
3.3.1 Segmentasi

Proses pertama dalam sistem yang akan dibuat adalah segmentasi. Pada tahap ini, sebuah dokumen masukan yang merupakaan file citra RGB akan ditelusuri keberadaan karakter-karakternya. Beberapa proses yang terdapat pada tahap segmentasi dipaparkan pada subbab-subbab berikut ini.

3.3.1.1 Gambar Biner

Pada tahap pertama ini, input yang diterima oleh sistem berupa citra dengan level pixel beragam dari 0 (hitam) hingga 255 (putih). Oleh karena itu, citra tersebut perlu diubah menjadi lebih sederhana yaitu menjadi citra biner, di mana hanya terdapat dua komponen di

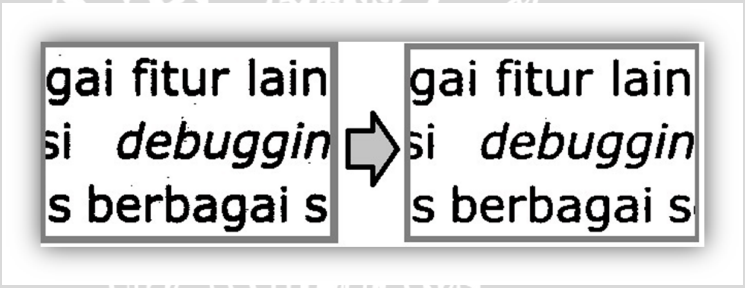
dalamnya yaitu 0 (merepresentasikan warna hitam) dan 1 (merepresentasikan warna putih).



Gambar 3.3. Pengubahan menjadi gambar biner

3.3.1.2 Filtering

Citra masukan tidak selalu bersih, adakalanya terdapat beberapa *noise* yang dapat mempersulit proses pengambilan fiturnya. Untuk itu, diperlukan proses filtering sehingga data dapat dibaca dengan benar. Karena pada citra karakter lebih banyak ditemui noise berupa titik-titik kecil atau biasa disebut dengan *salt and pepper noise* maka proses filtering yang dipilih adalah median filter. Median filter ini cukup banyak digunakan untuk mengatasi *noise* jenis tersebut.



Gambar 3.4 Proses filtering

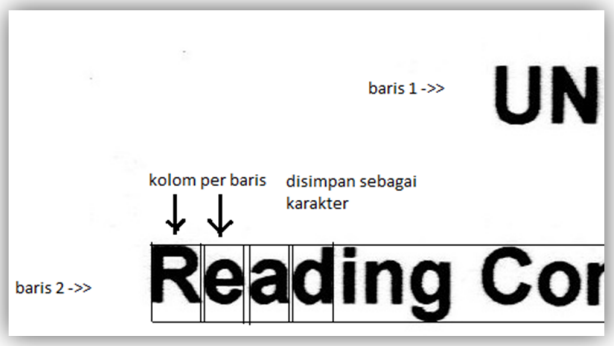
3.3.1.3 Pencarian Baris

Pada dokumen masukan, selanjutnya akan dilakukan pencarian baris yang di dalamnya terdapat pixel hidup. Di mana pada baris-baris yang didapat ini terdapat karakter-karakter yang akan dikenali.

3.3.1.4 Pencarian Kolom

Pada setiap baris yang telah didapatkan sebelumnya, dilakukan penelusuran secara horizontal untuk mendapatkan koordinat kolom-kolom yang merupakan sebuah karakter. Proses ini berlangsung dari

baris pertama hingga baris terakhir yang terdapat pada dokumen seperti yang ditunjukkan pada gambar 3.5.



Gambar 3.5 Proses segmentasi

Keluaran dari keseluruhan proses segmentasi merupakan kumpulan karakter yang disimpan dalam bentuk per baris. Gambar 3.6 merupakan *sampel* karakter keluaran proses segmentasi.

Gambar 3.6 Keluaran segmentasi

3.3.2 Pembacaan Karakter

Tahap kedua setelah semua karakter di dalam dokumen didapatkan, dilakukan pembacaan fitur pada setiap karakter. Pembacaan karakter meliputi beberapa proses di bawah ini.

3.3.2.1 Menentukan *Template*

Fitur pertama yang diambil pada sebuah karakter adalah template dari karakter. *Template* di sini merupakan posisi karakter pada hasil pemotongan dari segmentasi. Terdapat 4 klasifikasi *template* yang akan dipergunakan sebagai fitur, yaitu:

1. *Template* 1, untuk *template* dengan bagian atas saja yang kosong.

2. *Template* 2, untuk *template* dengan bagian atas dan bawah kosong.
3. *Template* 3, untuk *template* dengan bagian bawah saja yang kosong.
4. *Template* 4, untuk *template* yang penuh terisi oleh karakter.

Sebagai contoh pada gambar 3.6 akan diklasifikasikan sebagai *template* 3.

3.3.2.2 Mengecek Bagian Karakter yang Terpisah

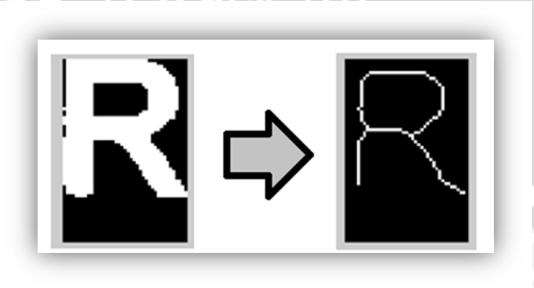
Pada beberapa karakter, satu karakter dapat memiliki bagian yang terpisah seperti pada karakter i, j, ?, !, dll. Untuk itu, mengecek bagian karakter yang terpisah dapat menjadi fitur dari sebuah karakter.

Karakter yang terpisah akan dicek secara horizontal maupun vertikal. Bila karakter tersebut terpisah secara vertikal, maka fitur selanjutnya yang akan diambil adalah posisi titik pada karakter tersebut. Ada dua pilihan yang dapat diambil pada posisi titik ini, yaitu posisi titik di bawah dan posisi titik di atas.

3.3.2.3 Thinning

Setelah mendapatkan 3 fitur di atas, karakter akan dipecah menjadi beberapa segmen. Untuk memudahkan pencarian segmen, maka dilakukan proses *thinning.*

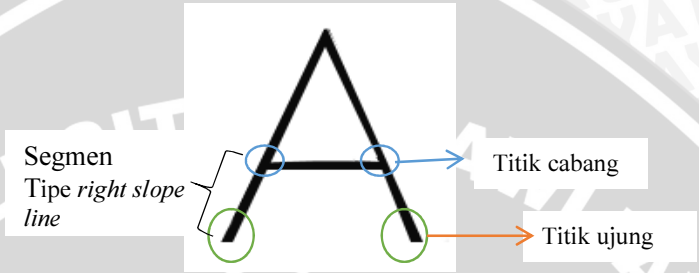
Thinning merupakan proses penipisan karakter sehingga didapatkan bentuk *skeleton* yang memudahkan pengambilan segmen pada karakter. Proses *thinning* seperti yang terlihat pada gambar 3.7.



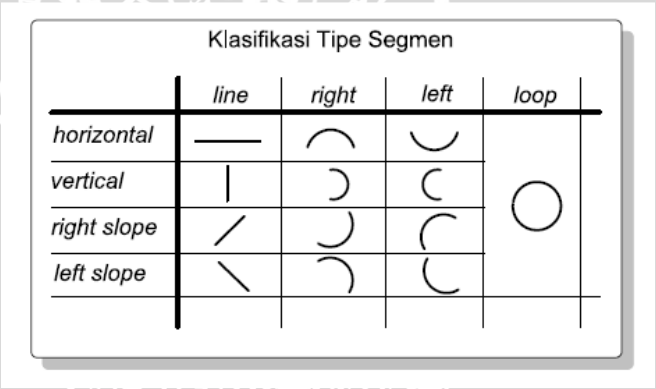
Gambar 3.7. Proses *thinning*

3.3.2.4 Pencarian Titik Penting

Titik penting yang dimaksud di sini adalah titik ujung dan titik cabang seperti pada gambar 3.8. Kedua titik ini nantinya akan digunakan sebagai titik awal dan titik akhir dalam memulai pencarian tetangga terdekat untuk mengetahui segmen penyusun karakter. Setelah itu segmen tersebut akan dicek bentuknya sesuai label yang diberikan gambar 3.9.



Gambar 3.8. Jenis-jenis titik penting



Gambar 3.9 Klasifikasi tipe segmen

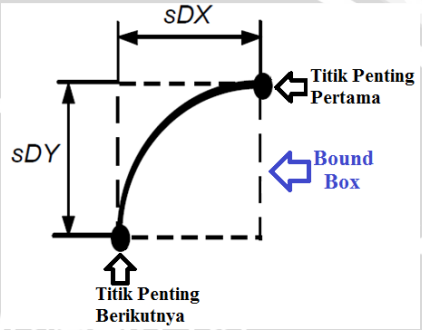
3.3.2.5 Pencarian Segmen

Sebelum memulai pencarian segmen, terlebih dahulu dibuat sebuah matrik temporer yang ukurannya sama dengan matrik karakter, tetapi nilai pixelnya semua diisi 0.

Dari setiap titik penting, ditelusur setiap titik tetangga terdekat sampai bertemu pada titik penting berikutnya. Semua koordinat titik tersebut disimpan dalam sebuah matrik *px2*, di mana *p* merupakan banyak pixel pembentuk segmen, dan 2 adalah kolom matrik yang menunjukkan posisi pixel (kolom 1 untuk *x* dan kolom 2 untuk *y*).

Untuk setiap titik yang dilalui, pada matrik temporer koordinat pixel diubah menjadi 1 yang mengindikasikan bahwa pixel tersebut telah dilalui dan dapat diabaikan.

Setiap segmen dihitung *bound box*-nya untuk pengklasifikasian ukuran segmen. Gambar 3.10 menunjukkan area *bound box* dari sebuah segmen berbentuk kurva dengan sDX merupakan panjang *bound box* dan sDY merupakan tinggi *bound box*.



Gambar 3.10. *Bound box* segmen

3.3.2.6 Pencarian *Hole*

Hole atau lubang yang terdapat pada karakter juga merupakan fitur yang dapat diambil. Sebelum memulai pembacaan bentuk geometri segmen, akan dicek terlebih dahulu apakah karakter memiliki *hole* dan jumlahnya.

Hole bisa merupakan sebuah *loop* atau dua segmen yang titik awal dan titik akhirnya saling bertautan. Karena itu, pencarian *hole* dapat dimulai dengan melihat keterhubungan antara titik ujung pada setiap segmen.

3.3.2.7 Pembacaan Bentuk Geometri Tiap Segmen

Setiap segmen yang telah didapatkan selanjutnya akan ditentukan bentuk geometrinya berdasarkan klasifikasi bentuk yang akan dijabarkan pada subbab berikutnya. Bentuk geometri tiap segmen ini akan menjadi fitur penting karakter yang nantinya akan direpresentasikan berdasarkan posisi segmen pada karakter tersebut.

Tetapi, untuk menghindari terlalu banyaknya segmen yang mungkin terdapat pada karakter, seperti pada karakter keluarga *Serif*, maka yang diambil menjadi fitur hanya segmen yang berukuran

sedang dan besar. Sedangkan untuk yang berukuran kecil akan diabaikan.

3.3.2.8 Fitur Karakter

Fitur karakter merupakan keluaran dari proses pembacaan karakter. Karena fitur ini yang akan digunakan oleh jaringan Neural Network Backpropagation dalam proses pengenalan karakter nantinya, maka fitur karakter harus diseragamkan formatnya. Hal ini bertujuan untuk generalisasi bagi jaringan.

Fitur karakter merupakan sebuah matrik berukuran 118x1. Di mana setiap bit yang berisi nilai 1 dalam barisnya menyatakan informasi tertentu berdasarkan format yang telah ditetapkan sebelumnya. Bila dalam bit tersebut berisi nilai 0, maka artinya karakter tersebut tidak memiliki fitur yang dimaksudkan. Format fitur karakter ditunjukkan pada tabel 3.1.

Tabel 3.1 Tabel format fitur karakter

No.	Fitur Karakter
1.	Template 1
2.	Template 2
3.	Template 3
4.	Template 4
5.	1 Hole
6.	2 Hole
7.	Terpisah vertikal
8.	Terpisah horizontal
9.	Posisi titik di atas
10.	Posisi titik di bawah
11-22 Posisi atas kiri	
11.	Bentuk Geometri 1
12.	Bentuk Geometri 2
13.	Bentuk Geometri 3
14.	Bentuk Geometri 4
15.	Bentuk Geometri 5

16.	Bentuk Geometri 6
17.	Bentuk Geometri 7
18.	Bentuk Geometri 8
19.	Bentuk Geometri 9
20.	Bentuk Geometri 10
21.	Bentuk Geometri 11
22.	Bentuk Geometri 12
23-34	Posisi atas tengah
35-46	Posisi atas kanan
47-58	Posisi tengah kiri
59-70	Posisi tengah tengah
71-82	Posisi tengah kanan
83-94	Posisi bawah kiri
95-106	Posisi bawah tengah
107-118	Posisi bawah kanan

3.3.3 Pembacaan Bentuk Geometri

Bentuk geometri merupakan fitur utama karena dalam skripsi ini lebih menitikberatkan pada ekstraksi fiturnya. Pembacaan bentuk geometri merupakan bagian dari pembacaan karakter. Pada tahap ini metode yang dipakai adalah metode fuzzy. Proses-proses yang dilakukan untuk mendapatkan bentuk geometri dipaparkan pada subbab-subbab berikut ini.

3.3.3.1 Pengecekan Sebuah *Loop*

Secara umum, terdapat 3 bentuk dasar dari geometri yang akan diambil, yaitu *loop*, garis, dan kurva. Namun, bentuk *loop* akan menjadi bentuk pertama yang akan diidentifikasi pada sebuah segmen. Karena kedua bentuk yang lain harus melewati beberapa tahapan yang lebih rumit dari *loop*. Suatu segmen akan diklasifikasikan sebagai *loop* bila titik awal dan titik akhir segmen adalah sama.

3.3.3.2 Menghitung Persamaan Garis Pokok

Segmen selain *loop* selanjutnya akan ditentukan masuk pada kategori garis atau kurva. Untuk dapat menentukan bentuk segmen tersebut, maka diperlukan sebuah garis bantu yang akan menjadi garis

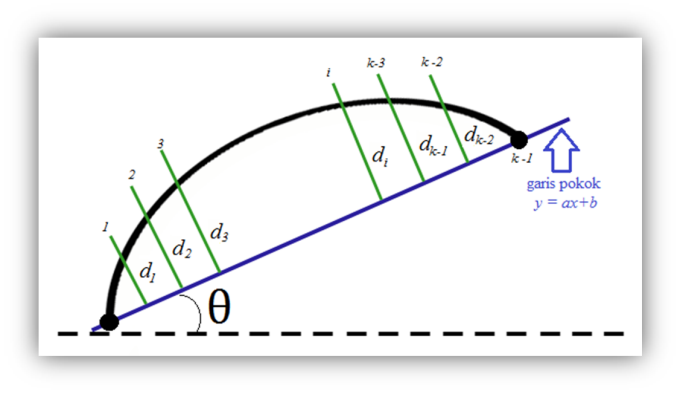
pokok segmen tersebut. Garis pokok merupakan garis linier yang menghubungkan titik awal dengan titik akhir segmen. Perhitungan garis pokok memenuhi persamaan garis linier:

$$y = ax + b$$

Setelah itu dihitung jarak rata-rata tiap pixel segmen ke garis pokok (deviasi). Untuk menghitung deviasi berlaku persamaan:

$$\mathrm{D} = \frac{\sum_{i=1}^{k-2} d_i}{n}$$

Pada persamaan di atas, D melambangkan deviasi, *d* merupakan jarak tiap pixel segmen ke garis pokok, dan *n* adalah banyak pixel yang dihitung. Gambar 3.11 berikut ini akan memperjelas ilustrasi di atas.



Gambar 3.11 Penghitungan garis pokok dan deviasi

3.3.3.3 Menentukan Bentuk Dasar Segmen

Bentuk dasar segmen merupakan dua pilihan antara bentuk garis atau bentuk kurva. Dengan melihat nilai deviasi yang didapatkan pada proses sebelumnya, maka akan didapatkan bentuk dasar dari segmen tersebut.

Suatu segmen diklasifikasikan ke dalam bentuk dasar garis apabila memenuhi pertidaksamaan berikut:

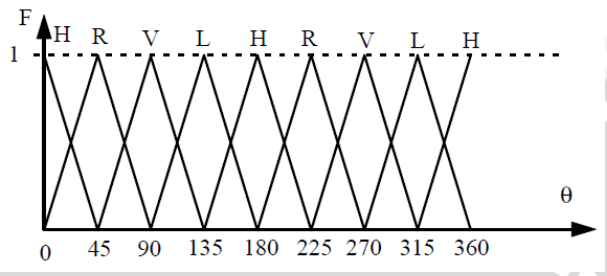
$$D \leq \beta$$

dengan D merupakan deviasi, sedangkan β adalah ambang batas kurva. Bila nilai β lebiha besar daripada nilai D, maka bentuk dasar segmen tersebut adalah sebuah kurva.

3.3.3.4 Menentukan Orientasi Segmen

Selanjutnya segmen akan ditentukan orientasi atau kemiringannya. Terdapat 4 klasifikasi orientasi yaitu Horizontal (H), Vertikal (V), Serong Kiri (L), dan Serong Kanan (R).

Penghitungan orientasi segmen mengikuti fungsi keanggotaan fuzzy pada gambar 3.12.

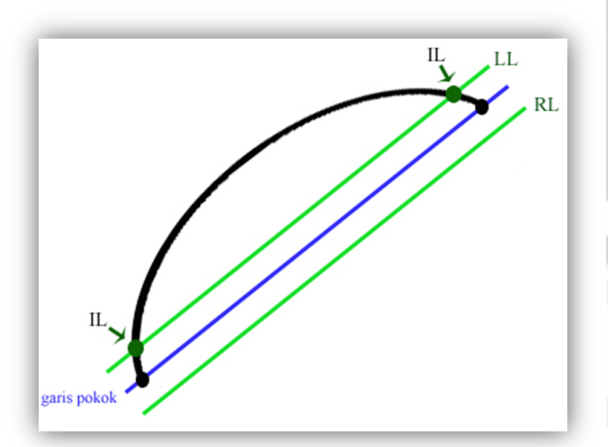


Gambar 3.12 Fungsi keanggotaan orientasi segmen

3.3.3.5 Menentukan Jenis Kurva

Pada segmen yang memiliki bentuk dasar kurva, masih harus ditentukan jenis kurvanya. Karena sebuah kurva memiliki sebuah sisi terbuka sehingga jenis kurva digolongkan ke dalam dua jenis, yaitu kurva hadap kanan dan kurva hadap kiri.

Untuk memastikan jenis kurva, maka diperlukan 2 garis bantu yang sejajar dengan garis pokok. Masing-masing garis bantu tersebut berada di kiri (LL) dan di kanan (RL) garis pokok. Selanjutnya dilihat berapa titik perpotongan (IL / IR) antara garis bantu tersebut dengan kurva.



Gambar 3.13 Ilustrasi penentuan jenis kurva

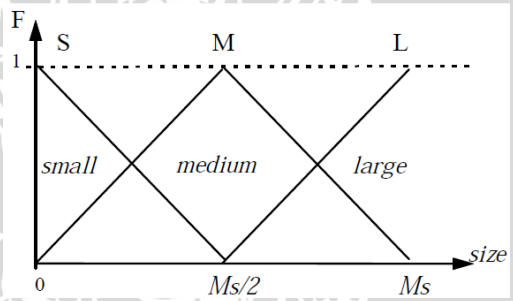
3.3.3.6 Menghitung Ukuran Segmen

Pengklasifikasian ukuran segmen dibagi menjadi tiga macam, yaitu kecil (S), sedang (M), dan besar (L). Fungsi anggota fuzzy pada gambar 3.14 diberlakukan untuk menentukan ukuran segmen.

$$\text{FS}\ (size)\ = \begin{cases} 0, & size > 0.5\ \text{Ms} \\ \dfrac{-2}{\text{Ms}\ \times\ \text{size}} + 1, & \text{size} \leq 0.5\ \text{Ms} \end{cases}$$

$$\text{FM}(size) = -\left|\frac{-2}{\text{Ms}\ \times\ \text{size}} - 1\right| + 1$$

$$\text{FL}\ (size)\ = \begin{cases} 0, & size < 0.5\ \text{Ms} \\ \left|\dfrac{-2}{\text{Ms}\ \times\ \text{size}} + 1\right|, & \text{size} > 0.5\ \text{Ms} \end{cases}$$



Gambar 3.14 Fungsi keanggotaan ukuran segmen

- Untuk garis horizontal:

  Size = banyak pixel anggota segmen.

  Ms = lebar karakter.
- Untuk garis vertikal:

  Size = banyak pixel anggota segmen.

  Ms = tinggi karakter.
- Untuk kurva :

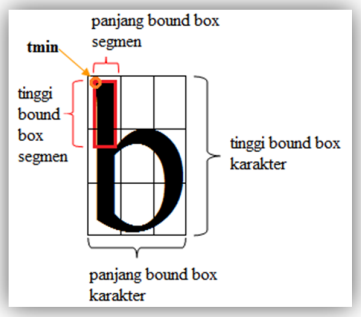
  Size = sDx× sDY, (ukuran *bound box* kurva)

  Ms = $m \times n$, (ukuran karakter)

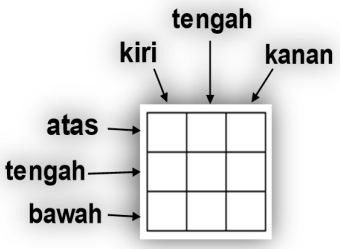
3.3.3.7 Menentukan Posisi Segmen

Setelah bentuk dan ukuran segmen diketahui, selanjutnya perlu diketahui posisi segmen pembentuk karakter tersebut. Posisi segmen

akan dibagi menjadi 6 bagian yang diinterpretasikan ke dalam 4 bit biner. Pada gambar 3.15 (a) menunjukkan pengambilan sebuah segmen dari karakter yang posisinya akan dipetakan ke dalam salah satu bagian pada gambar 3.15 (b).



(a)



(b)

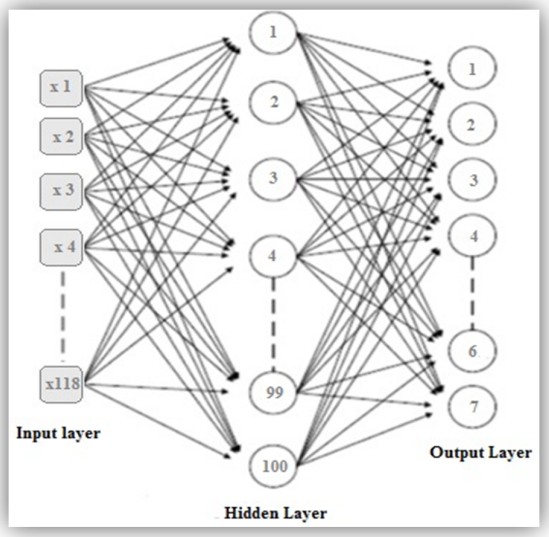
Gambar 3.15 Posisi Segmen

3.3.4 Pengenalan

Pengenalan karakter yang telah diambil fiturnya dilakukan oleh jaringan *Neural Network Backpropagation*. Namun sebelum digunakan untuk mengenali karakter, jaringan *Neural* tersebut perlu mendapatkan pelatihan terlebih dahulu. Jaringan yang telah dilatih inilah yang selanjutnya disimpan untuk digunakan dalam mengenali karakter berdasarkan fitur karakter tersebut.

3.3.4.1 Pembuatan Jaringan

Jaringan *Neural* yang dibuat terdiri atas 3 layer, yaitu layer masukan yang merupakan fitur dari karakter sejumlah 118 fitur, kemudian *hidden layer* dengan jumlah *node* 100, selanjutnya layer terakhir adalah layer keluaran dengan 7 *node* yang merupakan nilai biner dari karakter. Arsitektur jaringan *Neural* yang akan dibuat seperti ditunjukkan pada gambar 3.16.



Gambar 3.16. Jaringan *Neural Backpropagation*

3.3.4.2 Proses Pelatihan / *Training*

Sebelum melakukan proses training, diperlukan data masukan berupa fitur karakter yang akan ditraining. Data training yang digunakan berjumlah total 400 data, dengan dengan rincian 150 data masing-masing untuk huruf kecil dan huruf besar, serta data angka dan tanda baca masing-masing 50 data.

Untuk itu sebagai masukan pada layer masukan berupa matrik berukuran 118x400, di mana 118 merupakan fitur karakter. Sedangkan layer keluaran berisi data target yang merupakan bentuk biner dari tiap data karakter dan dikemas dalam matrik berukuran 7x400.

Training jaringan NNB dilakukan secara serentak dengan memasukkan matrik berukuran 118x400 untuk layer masukan, dan matrik berukuran 7x400 untuk layer keluaran.

Selanjutnya *Toolbox Neural Network* Matlab akan memilah data masukan tersebut ke dalam 3 kategori, yaitu:

1. 60% pertama dari data akan menjadi data yang dilatih.
2. 20% data selanjutnya akan digunakan sebagai validasi jaringan.
3. 20% data terakhir akan digunakan untuk menguji generalisasi jaringan.

Berikutnya fungsi aktivasi yang digunakan pada *hidden layer* menggunakan fungsi transfer sigmoid yang hanya memiliki keluaran antara 0 dan 1. Sedangkan fungsi pembelajaran yang dipilih adalah *Scalled Conjugate Gradient* yang sangat membantu dalam mempercepat proses pelatihan untuk data yang cukup banyak.

Setelah terbentuk jaringan yang telah dilatih, selanjutnya jaringan tersebut disimpan untuk digunakan dalam pengenalan karakter.

3.3.5 Pengubahan Menjadi ASCII Teks

Dari hasil pengenalan yang dilakukan oleh jaringan *Neural* masih berupa nilai biner dari karakter. Untuk memudahkan dalam bahasa yang mudah dimengerti oleh manusia, maka nilai biner ini harus diubah menjadi bentuk ASCII karakter tersebut, yaitu bentuk teks karakter.

3.4 Pengujian Sistem

Pengujian sistem merupakan tahap untuk menguji coba keberhasilan sistem dan untuk mengetahui kekuranga-kekurangan yang mungkin perlu mendapat perbaikan.

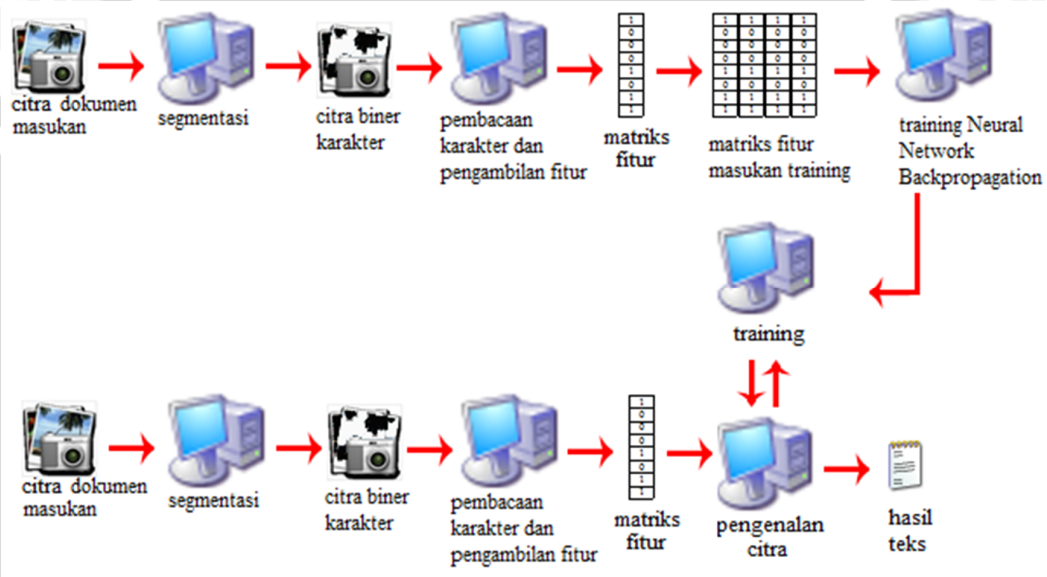
3.5 Kesimpulan dan Saran

Penarikan kesimpulan dilakukan berdasarkan pengujian dan analisis yang telah dilakukan terhadap sistem. Selanjutnya saran diberikan untuk perbaikan yang mungkin diperlukan oleh sistem untuk penelitian selanjutnya .

BAB IV
PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan perancangan dan implementasi perangkat lunak dari aplikasi pengenalan karakter teks menggunakan *Neural Networks Backpropagation.* Implementasi menggunakan bahasa pemrograman Matlab versi 7.7 tahun 2008.

4.1 Blok Diagram Sistem



Gambar 4.1 Diagram Sistem Pengenalan Karakter

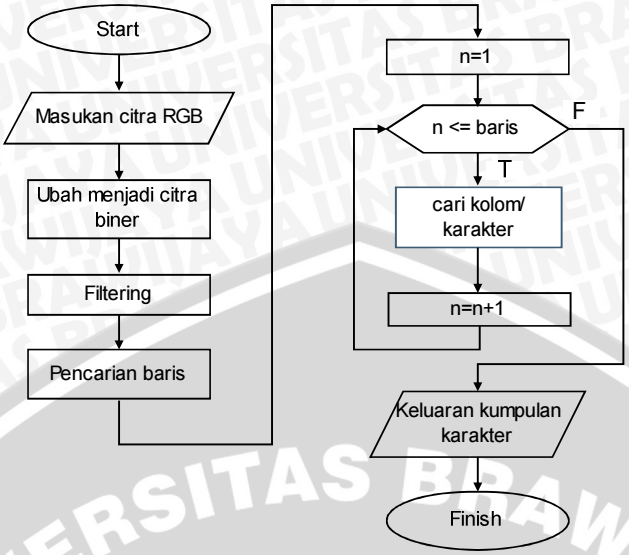
Pada blok diagram sistem yang ditunjukkan oleh gambar 4.1 menggambarkan 2 proses dari sistem yaitu pengenalan sebelum training dan setelah training. Penjelasan berikut akan menjabarkan fungsi dari beberapa komponen penyusun yang sama dalam 2 proses tersebut maupun yang berbeda dari keduanya.

1. Citra dokumen masukan, merupakan gambar karakter RGB 8 bit dengan format file *bitmap, *jpg, *png, ataupun *tif.
2. Segmentasi, merupakan proses pemisahan tiap karakter yang terdapat pada citra. Pertama kali citra yang merupakan citra RGB akan diubah menjadi bentuk binernya, di mana hanya terdapat warna hitam (0) dan putih (1) saja. Setelah itu akan dilakukan proses *filtering* untuk menghilangkan *noise* yang mungkin ada. Proses selanjutnya adalah pencarian baris dan pencarian kolom untuk mendapatkan posisi tiap-tiap karakter.

3. Citra biner karakter, merupakan keluaran dari proses segmentasi. Sebagai keluaran segmentasi, sangatlah mungkin terdapat beberapa karakter dari sebuah citra masukan.
4. Pembacaan karakter dan pengambilan fitur, merupakan proses pemecahan karakter menjadi beberapa segmen yang akan diambil fiturnya. Oleh karena itu, pada proses kedua ini melibatkan sebuah rangkaian proses pembacaan bentuk geometri untuk mendapatkan fitur dari karakter.
5. Matriks fitur, merupakan keluaran proses pembacaan karakter. Bentuk matriks ini telah memuat informasi berupa fitur karakter yang direpresentasikan sebagai matriks berukuran *118x1* yang memiliki nilai biner 0 dan 1 saja.
6. Matriks fitur masukan training, merupakan sejumlah matriks fitur dari data atau sampel citra yang akan ditraining sebelum sistem pengenalan karakter ini dapat digunakan. Matriks berukuran *118 x n* ini akan menjadi masukan bagi jaringan *Neural Network*, di mana *n* merupakan jumlah data yang akan training. Masukan ini hanya terdapat pada proses pengenalan sebelum training.
7. *Training*, proses pembelajaran karakter bagi jaringan Neural Network untuk dapat mengenali karakter. Pada proses ini diberikan beberapa data contoh tiap karakter dan dilakukan sebelum sistem dapat difungsikan sebagai pengenal karakter teks.
8. Pengenalan citra, merupakan proses pengenalan yang dilakukan setelah system mendapat training, sehingga pengenalan karakter didasarkan pada pengalaman sistem setelah dilatih dengan data training.
9. Hasil teks, keluaran sistem berupa karakter teks yang dapat disimpan dalam format .txt.

4.2 Perancangan dan Implementasi Segmentasi

Proses segmentasi berfungsi untuk memotong citra dokumen berdasarkan karakter-karakter yang ditemukan di dalamnya. Diagram sistem proses segmentasi terlihat seperti pada gambar 4.2.



Gambar 4.2 Diagram alir proses segmentasi

4.2.1 Masukan Segmentasi

Masukan segmentasi berupa citra dokumen dengan ekstensi bmp, jpg, gif, png atau tif. Citra akan diubah dalam bentuk matriks citra dengan format RGB dengan ukuran 8 bit. Kemudian, citra RGB diubah menjadi bentuk citra biner yang hanya memiliki nilai 0 (putih) dan 1(hitam). Pada citra biner tersebut kemudian dilakukan proses *filtering* untuk menghilangkan *noise* yang mungkin terdapat dalam citra. Gambar 4.3 adalah implementasi dalam bentuk program.

```
L = imread(img);
level = graythresh(L);
cbiner= im2bw(L, level);
medfil = medfilt2(cbiner, [2 1]);
medfil = ~medfil;
```

Gambar 4.3 *Listing* program proses citra biner dan median filter

Variabel **L** merupakan hasil pembacaan matriks citra dengan format RGB yang dilakukan oleh fungsi `imread`, di mana parameter masukan `img` merupakan citra dokumen masukan yang dibaca oleh fungsi `imread`.

Pengubahan menjadi bentuk citra biner yang dilakukan oleh fungsi `im2bw` memerlukan 2 parameter, yaitu matriks citra RGB dan sebuah nilai ambang batas intensitas citra untuk menentukan nilai pixel tersebut akan diubah ke dalam warna hitam atau putih. Variabel **level** pada baris kedua merupakan nilai ambang batas yang diperlukan. Hasil nilai **level** didapatkan dari fungsi `graythresh` yang akan menghitung nilai intensitas citra yang

berada pada rentang 0 sampai 1. Karena itulah, parameter masukan yang diperlukan oleh fungsi `graythresh` adalah matriks citra yang akan diubah ke dalam bentuk citra biner yaitu variabel **L**. Selanjutnya fungsi `im2bw` akan mengubah citra RGB menjadi citra biner yang hasilnya disimpan dalam variabel **cbiner**.

Proses filtering menggunakan fungsi `medfilt2` yang berguna untuk menghilangkan *noise* yang mungkin terdapat pada citra. Fungsi `medfilt2` merupakan fungsi *median filtering* dari matlab yang prosesnya memerlukan masukan berupa matriks citra biner yaitu variabel **cbiner** dan dan sebuah matriks yang menunjukkan bahwa proses median filtering akan dilakukan pada tiap jangkauan tetangga *m x n* dari citra. Pada baris ketiga di atas, masukan kedua adalah matriks *2x1*, yang berarti bahwa proses *median filtering* akan dilakukan tiap jangkauan tetangga *2x1* pixel. Hasil proses yang dilakukan oleh fungsi pada baris ketiga akan disimpan pada variabel **medfil**.

Variabel **medfil** menyimpan matriks citra biner dengan warna background putih. Karena untuk proses selanjutnya lebih mudah menggunakan citra dengan background hitam dan karakter putih, maka pada baris keempat dilakukan pembalikan pada variabel **medfil** dengan menggunakan perintah negasi.

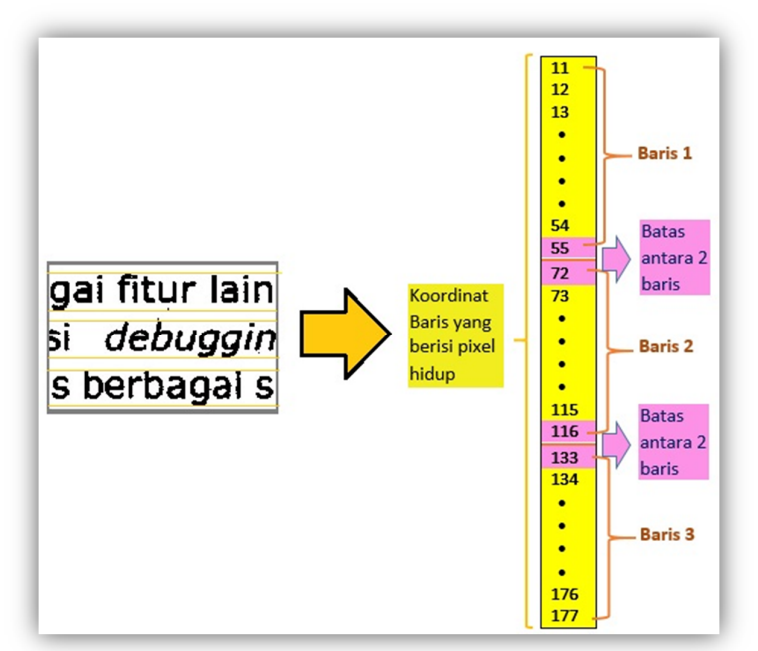
4.2.2 Pemotongan Karakter pada Citra Masukan

Setelah mendapatkan bentuk matriks biner dari citra, selanjutnya adalah proses pencarian karakter pada citra masukan. Proses pertama adalah dengan mendeteksi baris yang terdapat pixel hidup pada citra. Sedangkan untuk proses kedua adalah mendeteksi karakter dengan mencari kolom yang terdapat pixel hidup pada setiap baris yang telah ditemukan tadi. Implementasi proses pencarian baris dalam bentuk listing program ditunjukkan pada gambar 4.4. Sedangkan pada gambar 4.5 merupakan ilustrasi proses pencarian baris.

```
topBottom = find(sum(medfil,2));
%get Line
j=1;
line = [];l =[];
for i = 1:length(topBottom)-1
if (topBottom(i)+1 == topBottom(i+1))&...
(i+1 ~= length(topBottom))
l = [l; topBottom(i)];
elseif (topBottom(i)+1 == topBottom(i+1)) &...
(i+1 == length(topBottom))
line{j}=[l; topBottom(i); topBottom(i+1)];
else line{j}=[l; topBottom(i)];
l=[];
j = j+1;
end
end
```

Gambar 4.4 *Listing* program pencarian baris

Pencarian baris merupakan proses *scanning* matriks citra dari atas ke bawah. Karena itu pada baris pertama proses *scanning* dilakukan dengan melakukan penjumlahan kolom variabel **medfil** menggunakan fungsi `sum`, masukan fungsi ini yaitu variabel **medfil** yang akan dijumlah dan nilai **2** mengacu pada kolom matriks **medfil.** Selanjutnya fungsi `find` pada matlab berfungsi untuk mencari semua nilai 1 pada matriks masukan, yaitu matriks hasil penjumlahan variabel **medfil** secara kolom. Hasil dari proses ini disimpan pada variabel **topBottom** yang berupa matriks berukuran *i x 1*.



Gambar 4.5 Proses pencarian baris

Selanjutnya proses pencarian baris dimulai dengan menelusuri matriks **topBottom**. Variabel **j** menyimpan jumlah baris yang akan didapat dari proses pencarian baris. Sedangkan variabel **line** merupakan array yang akan menyimpan baris yang ditemukan. Selanjutnya variabel **l** akan menyimpan posisi-posisi koordinat baris dari awal hingga akhir baris yang ditemukan.

Proses *scanning* baris dilakukan dengan perulangan sejumlah **i**, di mana variabel **i** menunjukkan banyaknya anggota **topBottom**. Satu baris yang berisi karakter ditunjukkan dari anggota **topBottom** yang berurutan nilainya. Untuk itu bila anggota **topBottom** yang dicek nilainya memiliki selisih nilai tepat 1 lebih kecil dari anggota **topBottom** berikutnya, matriks **l** akan terus ditambah anggotanya hingga persyaratan di atas tidak terpenuhi, yaitu ketika nilai anggota **topBottom** memiliki selisih lebih dari 1 dari nilai anggota **topBottom** berikutnya. Matriks **l** yang berisi koordinat-koordinat posisi baris yang ditemukan selanjutnya akan menyerahkan hasilnya untuk disimpan pada variabel **line** di mana pada variabel **line** hasil tersebut akan ditandai atau disimpan pada anggota ke-**j**, dan kemudian matriks **l** direset ulang menjadi matriks kosong untuk proses selanjutnya. Proses tersebut terus berulang hingga anggota terakhir dari **topBottom**, selain itu penyimpanan anggota oleh variabel **line** terus ditambahkan 1 atau pada listing program diatas ditunjukkan dengan nilai `j=j+1`.

Implementasi proses pencarian kolom tiap baris dalam bentuk listing program ditunjukkan pada gambar 4.6.

```
chars =[];
for l = 1:length(line) %number of lines
thisLine = medfil(line{l},:);
gap = find(sum(thisLine,1));
j=1;
charsLine=[];
c =[];
%find char each line
for i = 1:length(gap)-1
if (gap(i)+1 == gap(i+1)) &...
(i ~= length(gap)-1)
c = [c gap(i)];
elseif (gap(i)+1 == gap(i+1)) &...
(i == length(gap)-1)
charsLine{j} = [c gap(i) gap(i+1)];
elseif (gap(i+1) - gap(i)) > 20
charsLine{j} = [c gap(i)];
charsLine{j+1}=[]; %space
c = [];
j = j+2;
else charsLine{j}=[c gap(i)];
c=[];
j = j+1;
end
end
chars{l}={charsLine};
end
```

Gambar 4.6 *Listing* program pencarian kolom

Langkah-langkah proses pencarian kolom pada tiap baris secara garis besar sama dengan proses pencarian baris. Hanya saja pada pencarian kolom, proses *scanning* dilakukan dari kiri ke kanan, sehingga yang dilakukan adalah penjumlahan baris.

Variabel **chars** merupakan array yang menyimpan koordinat-koordinat kolom pada tiap baris karakter. Sehingga jumlah kolom anggota **chars** akan sama dengan jumlah anggota **line**. Perbedaannya anggota **chars** pada setiap kolom masih berupa array yang menyimpan koordinat-koordinat kolom posisi karakter yang ditemukan.

Proses pencarian dilakukan pada sejumlah baris yang ditemukan atau jumlah anggota pada variabel **line**. Pada setiap anggota **line** ke-**i**, terlebih dahulu pemotongan citra **medfil** dilakukan dari posisi baris pertama hingga baris terakhir yang ditunjukkan oleh variabel **line** ke-**i**, dan untuk sementara disimpan pada variabel **thisLine**.

Selanjutnya dilakukan penjumlahan baris pada matriks **thisLine** dengan perintah `sum(thisLine,1)` dan kembali menggunakan fungsi `find` untuk mencari pixel hidup dan disimpan pada variabel **gap**.

Proses berikutnya kembali mengeset nilai **j** menjadi 1 untuk pertama kali, yang nantinya akan digunakan sebagai penanda anggota **charsLine** dan akan terus bertambah 1 hingga ditemukan anggota terakhir dari **gap**. Variabel **charsLine** digunakan untuk menyimpan sementara kolom-kolom koordinat karakter yang ditemukan pada tiap baris **line**. Sedangkan variabel **c** digunakan sebagai penyimpan sementara koordinat kolom sebuah karakter. Prinsip yang sama berlaku untuk pencarian kolom ini, yaitu nilai **gap** yang berurutan diasumsikan sebagai koordinat kolom bagi karakter.

Perulangan kembali dilakukan pada setiap anggota **gap** yang bila memiliki nilai selisih 1 lebih kecil dari nilai anggota **gap** berikutnya akan disimpan sementara pada matriks **c**. Anggota matriks **c** akan terus bertambah hingga nilai selisih nilai anggota **gap** ke-**i** dengan anggota **gap** berikutnya adalah lebih dari 1. Bila hal ini terjadi, maka hasil matriks **c** akan diserahkan dan disimpan pada **charsLine** dan ditandai sebagai anggota ke-**j**, selanjutnya matriks **c** akan direset menjadi matriks kosong untuk proses berikutnya dan nilai **j** akan ditambahkan 1. Ada suatu kondisi khusus di mana selisih anggota **gap** ke-**i** dengan anggota **gap** berikutnya lebih dari 20, maka **charsLine** ke-**j** akan disimpan sebagai matriks kosong, dan variabel **c** akan direset menjadi matriks kosong, sedangkan nilai **j** akan ditambahkan 2. Selanjutnya perulangan akan terus terjadi sampai anggota terakhir dari **gap**.

Bila anggota **gap** telah selesai dicari kolom-kolomnya, hasil proses tersebut yaitu array **charsLine** akan memberikan hasilnya untuk disimpan pada variabel **chars** pada anggota ke-**l**. Selanjutnya kembali dilakukan perulangan untuk anggota **line** ke-**l+1** hingga anggota **line** terakhir.

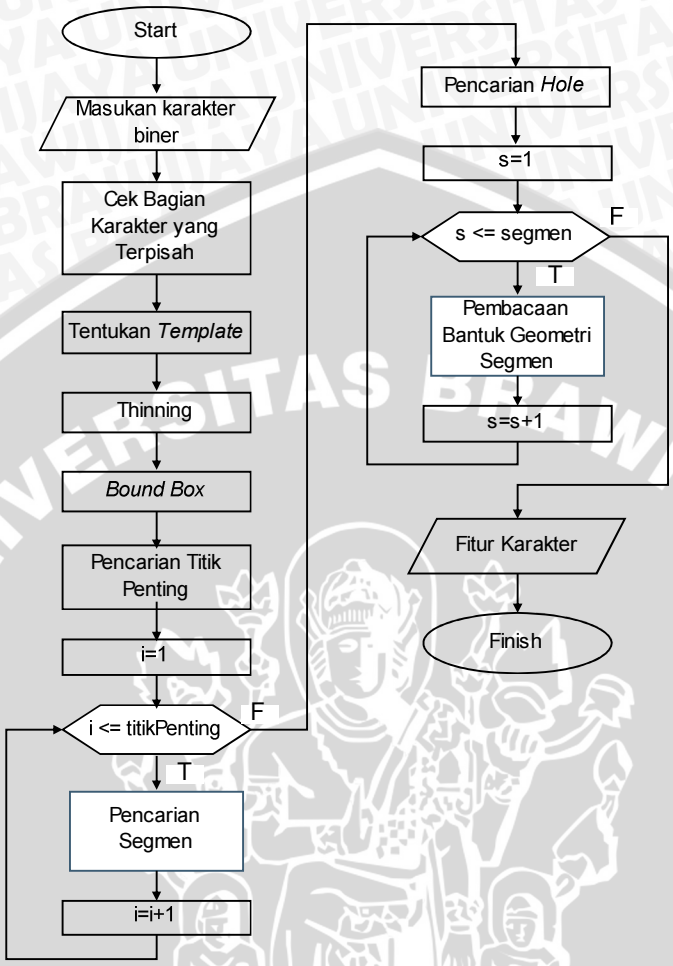
4.2.3 Keluaran Segmentasi

Keluaran dari proses segmentasi merupakan koordinat-koordinat pixel citra-citra karakter yang terdapat dalam citra masukan. Setiap citra karakter selanjutnya akan diambil fiturnya pada proses pembacaan karakter.

4.3 Perancangan dan Implementasi Pembacaan Karakter

Pembacaan karakter adalah proses pencarian fitur karakter dengan cara memecah sebuah karakter menjadi beberapa segmen pembentuk karakter dan mengklasifikasikan bentuk geometrinya. Selain itu juga diambil beberapa fitur umum untuk membantu

mengenali karakter yang mempunyai kemiripan bentuk. Diagram alir proses pembacaan karakter ditunjukkan pada gambar 4.7.



Gambar 4.7 Diagram alir proses pembacaan karakter

4.3.1 Masukan Proses Pembacaan Karakter

Masukan pada tahap ini adalah matrik citra biner karakter yang didapatkan dari proses segmentasi. Gambar 4.8 adalah listing program untuk mendapatkan masukan.

```
function obj = Karakter(inp)
             obj.inpLetter = inp;
```

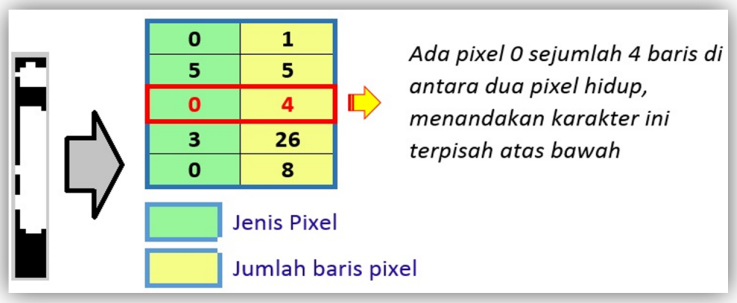
Gambar 4.8 *Listing* program masukan pembacaan karakter

Variabel **inp** merupakan matrik citra biner karakter yang akan diambil fiturnya. Karena telah berupa matrik, maka fungsi imread tidak diperlukan di sini. Baris pertama menunjukkan bahwa **inp** menjadi masukan bagi *class* Karakter. Keluaran *class* ini nantinya akan disimpan pada variabel **obj**. Di dalam variabel **obj** terdapat beberapa variabel yang menjadi data atau *properties* dari *class* Karakter. Untuk itu pada baris kedua, *properties*

inpLetter akan menyimpan matrik citra biner karakter masukan untuk digunakan pada *method* di dalam *class*.

4.3.2 Pendeteksian Bagian Karakter yang Terpisah

Pendeteksian bagian karakter yang terpisah dilakukan secara vertikal maupun horizontal. Untuk mendeteksi pemisahan bagian karakter secara vertikal, dilakukan *scanning* pixel hidup dari atas ke bawah. Tetapi, perlu pula dilihat adanya jarak antara pixel hidup yang berurutan satu dengan lainnya. Karena itu diperlukan sebuah matrik yang dapat menunjukkan jumlah pixel hidup dan pixel kosong (yang bernilai 0) secara berurutan dari atas ke bawah. Proses ilustrasi di atas dapat diperjelas dengan gambar 4.9.



Gambar 4.9 Proses pencarian bagian karakter terpisah vertikal

Proses *scanning* dari atas ke bawah ditunjukkan oleh listing program pada gambar 4.10 berikut.

```
%cek terpisah atas bawah
atasBawah=[];
sumKol = sum(obj.inpLetter,2);
s=1; hitung = 0;
while(s<length(sumKol))
    if (s+1)~= length(sumKol)
       if (sumKol(s)==0 & sumKol(s+1)==0) |...
              (sumKol(s)~=0 & sumKol(s+1)~=0)
           hitung=hitung+1;
       elseif (sumKol(s)==0 & sumKol(s+1)~=0) |...
              (sumKol(s)~=0 & sumKol(s+1)==0)
           hitung = hitung+1;
           atasBawah=[atasBawah; sumKol(s) hitung];
           hitung=0;
       end
    else hitung=hitung+1;
       atasBawah=[atasBawah; sumKol(s) hitung];
    end
    s=s+1;
end
```

Gambar 4.10 *Listing* program *scanning* dari atas ke bawah

Pada baris kedua merupakan pendefinisian variabel **atasBawah** sebagai bentuk matrik kosong. Variabel **atasBawah** ini nantinya akan berisi matrik

jumlah pixel hidup dan pixel kosong. Kemudian dilakukan penjumlahan secara kolom pada citra biner karakter yaitu matrik yang disimpan **obj.inpuLetter**, karena yang scanning dilakukan dari atas ke bawah. Variabel **sumKol** merupakan hasil dari penjumlahan kolom per baris. Variabel **s** diberi nilai awal 1 karena akan menjadi parameter dalam perulangan yang akan dilakukan sejumlaha anggota **sumKol**. Untuk menghitung banyaknya pixel yang berurutan, maka digunakanlah variabel **hitung** yang nilai awalnya 0.

Perulangan dilakukan untuk setiap anggota **sumKol**. Pixel hidup ditunjukkan dengan nilai anggota **sumKol** lebih dari 0, sedangkan nilai anggota **sumKol** yang 0 merupakan pixel kosong. Pada tiap perulangan, diambil nilai anggota **sumKol** sesuai urutan perulangan. Bila anggota **sumKol** berikutnya merupakan jenis pixel yang sama dengan jenis pixel yang masuk saat perulangan, maka **hitung** akan ditambahkan 1, bila sebaliknya maka **hitung** tetap akan ditambahkan 1 namun hasilnya akan diserahkan pada matrik **atasBawah**. Jadi saat matrik **atasBawah** ditambah anggotanya, kolom pertama yang ditambahkan merupakan nilai pixelnya, yaitu 0 atau 1, dan kolom kedua merupakan jumlah pixel tersebut yang diambil dari nilai **hitung**. Selanjutnya **hitung** kembali direset menjadi nilai 0.

```
%cek terpisah samping kiri kanan
kiriKanan= [];
sumBar = sum(obj.inpLetter,1);
s=1;
hitung = 0;
   while(s<length(sumBar))
      if (s+1)~= length(sumBar)
        if (sumBar(s)==0 & sumBar(s+1)==0) |...
              (sumBar(s)~=0 & sumBar(s+1)~=0)
            hitung=hitung+1;
        elseif (sumBar(s)==0 & sumBar(s+1)~=0) |...
              (sumBar(s)~=0 & sumBar(s+1)==0)
            hitung = hitung+1;
            kiriKanan=[kiriKanan;sumBar(s) hitung];
            hitung=0;
        end
      else hitung=hitung+1;
           kiriKanan=[kiriKanan; sumBar(s) hitung];
      end
   s=s+1;
   end
```

Gambar 4.11 *Listing* program *scanning* dari kiri ke kanan

Prinsip yang sama berlaku pada pencarian bagian karakter terpisah secara horizontal. Hanya saja variabel **atasBawah** diganti dengan **kiriKanan**. Karena *scanning* dilakukan secara kolom, maka variabel **sumBar** digunakan

untuk menyimpan hasil penjumlahan **obj.inputLetter** secara baris. Proses selanjutnya yang terjadi pada **sumBar** seperti yang dilakukan sebelumnya pada **sumKol**. Listing program pada gambar 4.11 menunjukkan proses pencarian secara horizontal.

Selanjutnya tinggal melihat hasil yang dibawa oleh variabel **atasBawah** dan **kiriKanan**. Bila jumlah baris **atasBawah** lebih dari 1, maka kemungkinan besar karakter tersebut memiliki bagian karakter yang terpisah. Variabel **baris** dengan nilai lebih dari 0 menunjukkan bahwa ada bagian terpisah. Untuk terpisah samping, maka digunakan variabel **kolom**. Gambar 4.12 merupakan listing program dari ilustrasi di atas.

```
if length(atasBawah)>1
   ab_copy = atasBawah (2:end-1,:);
   baris = find(ab_copy==0);
else baris=0;
end

kolom = find(kiriKanan==0);

if any(baris)
   terpisah = 7; %terpisah atas bawah
   atas = atasBawah(baris,2);
   bawah = atasBawah(baris+2,2);
   if (atas<bawah)
      obj.posisiTitik = 9; %titik di atas
   else obj.posisiTitik = 10; %titik di bawah
   end
elseif any(kolom)
   terpisah = 8; %terpisah kiri kanan
else terpisah = 0; %tidak ada bagian terpisah
end
```

Gambar 4.12 *Listing* program penentuan terpisah dan posisi titik

Pada baris ke-15, bila karakter tersebut terdeteksi terpisah secara vertikal, selanjutnya akan dideteksi posisi titik yang disimpan pada variabel **posisiTitik**.

4.3.3 Pengambilan Template Karakter

Pengambilan template karakter mengambil hasil yang disimpan oleh variabel **atasBawah**. Listing program pada gambar 4.13 menunjukkan proses penentuan *template* karakter berdasarkan matrik **atasBawah**.

```
%cek_template
if (atasBawah(1,1)==0 && atasBawah(1,2)==1)
   atasBawah (1,:)=[];
elseif (atasBawah(end,1)==0 && atasBawah(end,2)==1)
   atasBawah (end,:)=[];
else atasBawah = [atasBawah];
end

if (atasBawah(1,1)==0 && atasBawah(end,1)~=0)
   obj.template = 1; % atas saja yg kosong
elseif (atasBawah(1,1)==0 && atasBawah(end,1)==0)
   obj.template = 2; % atas bawah yg kosong
elseif (atasBawah(1,1)~=0 && atasBawah(end,1)==0)
   obj.template = 3; % bawah saja yg kosong
else obj.template = 4; % full
end
```

Gambar 4.13 *Listing* program penentuan *template*

4.3.4 Proses *Thinning*

Thinning dilakukan untuk mengambil bentuk *skeleton* karakter sehingga memudahkan system dalam mengambil fitur. Proses *thinning* menggunakan fungsi `bwmorph` dari matlab. Di mana parameter yang diperlukan oleh fungsi `bwmorph` adalah matrik biner dari karakter yaitu **obj.inpLetter,** sebuah operasi yang akan dilakukan yaitu *thinning*, yang dilambangkan dengan string `'thin'` dan jumlah perulangan yang akan dilakukan yaitu parameter `'Inf'` yang artinya proses *thinning* akan dilakukan hingga pada matrik karakter tidak ada bagian yang perlu mendapat proses *thinning*. Gambar 4.14 menunjukkan listing program untuk proses *thinning* yang kemudian hasilnya akan disimpan pada variabel **thinn**.

```
function thinn = get.thinn(obj)
     thinn = bwmorph(obj.inpLetter,'thin',Inf);
end
```

Gambar 4.14 *Listing* program proses *thinning*

4.3.5 Pengambilan *Bound Box* Karakter

Pengambilan bound box karakter berguna sebagai pembanding dalam menentukan ukuran segmen-segmen pembentuk karakter. Bound box sendiri merupakan panjang dan lebar karakter. Pengambilan bound box ditunjukkan oleh listing program pada gambar 4.15.

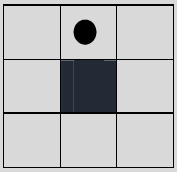
```
function boundBox = bound_box(obj)
      [x, y] = find(obj.thinn);
      m=[x, y];
      xmax = max([m(:,1)]);
      xmin = min([m(:,1)]);
      ymax = max([m(:,2)]);
      ymin = min([m(:,2)]);
      boundBox=[xmin ymin; xmax ymax];
end
```

Gambar 4.15 *Listing* program pengambilan *bound box* karakter

Pengambilan bound box dilakukan dengan mencari semua koordinat pixel hidup pada matrik citra biner **obj.thinn** yang akan disimpan pada variabel **x** dan **y** yang masing-masing menunjukkan baris dan kolom. Selanjutnya diambil koordinat minimum dan maximum masing-masing baris dan kolom dengan menggunakan fungsi `min` dan `max`. Pada akhirnya variabel **boundBox** akan menyimpan koordinat titik terendah dan tertinggi.

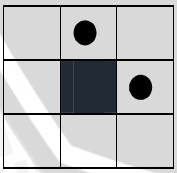
4.3.6 Pencarian Titik Penting

Titik penting digunakan untuk mencari segmen-segmen pembentuk karakter. Titik penting di sini adalah titik ujung dan titik cabang. Pencarian titik penting dilakukan dengan mengecek ke-8 tetangga pada tiap koordinat. Bila ternyata hanya memiliki satu tetangga saja, maka titik tersebut akan disimpan sebagai titik ujung. Namun, bila tetangga yang ditemukan hanya dua, titik tersebut akan diabaikan karena bukan merupakan titik penting seperti yang terlihat pada gambar 4.16.



N(P)= 1
Karena N(P) = 1, maka titik ini adalah titik ujung

(a)



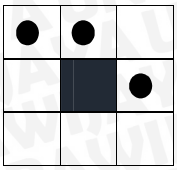
N(P)= 2
Karena N(P) = 2, maka titik ini bukan titik penting

(b)

Gambar 4.16 Ilustrasi pengambilan titik ujung

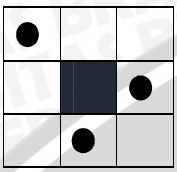
Lain halnya jika pada suatu titik ditemukan tetangga lebih dari dua, titik tersebut memiliki kompetensi untuk menjadi sebuah titik cabang bila memenuhi kondisi yang telah ditentukan. Syarat/kondisi yang akan membuat

sebuah titik dikategorikan sebagai titik cabang dijabarkan pada ilustrasi pada gambar 4.17.



(a)

N(P)= 3
S(P) = 1
Karena S(P) = 1, maka titik ini bukan titik cabang



(b)

N(P)= 3
S(P) = 0
Karena S(P) = 0, maka titik ini adalah titik cabang

Gambar 4.17 Ilustrasi pengambilan titik cabang

Gambar 4.18 merupakan listing program pencarian titik penting.

```
function titikPenting = cari_titikPenting(obj)
    titikPenting=[];
    [row, col]=find(obj.thinn);
    pixel1 = [row, col];
    for n = 1:size(pixel1,1)
        x = pixel1(n,1);
        y = pixel1(n,2);
 [tetangga,jum,jrkPixel]=...
            cari_tetangga(obj.thinn, x, y);%except xy
       if jum==1
          titikPenting=[titikPenting; x y jum];
       elseif ((jum==3)||(jum==4))&&(jrkPixel==0)
          titikPenting=[titikPenting; x y jum];
       else titikPenting=[titikPenting];
       end
   end
end
```

Gambar 4.18 *Listing* program pencarian titik penting

Pada baris kedua merupakan inisialisasi matrik kosong sebagai tempat untuk menyimpan **titikPenting**. Selanjutnya dicari pixel hidup pada **obj.thinn** yang koordinatnya disimpan pada **row** sebagai baris dan **col** sebagai kolom. Pada tahap berikutnya dilakukan perulangan sejumlah anggota **pixel1**.

Pada tiap perulangan diambil koordinat pixel hidup dan dicek jumlah serta jarak tetangga dengan menggunakan fungsi `cari_tetangga` yang parameter masukannya berupa matrik citra biner *thinning*, serta koordinat titik yang akan dicek. Keluaran dari fungsi ini adalah **tetangga** yang menyimpan koordinat masing-masing tetangga, **jum** yang menyimpan jumlah tetangga, serta **jrkPixel** yang menyimpan jarak antar tetangga.

Dari hasil yang didapatkan saat mencari tetangga, lalu dilihat bila nilai **jum** adalah 1, yang merupakan tanda titik ujung, maka matrik **titikPenting** akan ditambahkan anggotanya yaitu koordinat baris dan kolom dari titik tersebut dan jumlah tetangganya. Sedangkan bila nilai **jum** sama dengan 3 atau 4 dengan **jrkPixel** bernilai 0, maka titik tersebut termasuk titik cabang sehingga akan ditambahkan pada matrik **titikPenting**.

4.3.7 Pencarian Segmen

Selanjutnya dicari segmen-segmen pembentuk karakter yang dimulai dari satu titik penting ke titik penting lainnya. Gambar 4.19 merupakan listing program pencarian segmen pembentuk karakter.

```
function segmen= track_segmen(obj)
   if any(obj.titikPenting) %if 1
     thinnCopy = obj.thinn;
     segmen =[];
     for tp=1:size(obj.titikPenting,1)
       xtp = obj.titikPenting(tp,1);
       ytp = obj.titikPenting(tp,2);
       cabang = obj.titikPenting(tp,3);
       complete =[];
       i = 1;
       while(i<cabang+1) %cari segmen antara TP
         t=cari_tetangga(thinnCopy,xtp,ytp);
         if ~isempty(t) %ada tetangga,mulai cari
            xNext = t(1,1);
            yNext = t(1,2);
            notComplete=[xtp,ytp;xNext,yNext];
            thinnCopy(xNext,yNext)= 0;
            notTp = 0;

            while (notTp < 1)
      [notComplete,xNext,yNext,thinnCopy,notTp]=...
      find_nextPixel(notComplete, xNext, yNext, ...
            thinnCopy, obj.titikPenting);
            end
            complete{i} = [notComplete];
        else complete{i} =[];
        end %end if
        i = i+1;
      end %end while

      %rowPrev = complete(:,1);
      segmen{tp}=[complete];
     end %end for
  else segmen =[];
  end %end if 1
end %end function
```

Gambar 4.19 *Listing* program pencarian segmen

Pencarian segmen dimulai dengan melihat apakah ada anggota pada **obj.titikPenting**. Bila ada maka dilakukan pengkopian matrik citra biner

obj.thinn oleh **thinnCopy**, karena **thinnCopy** nanti akan digunakan untuk mengecek titik yang telah dilewati. Selanjutnya inisialisasi matrik **segmen** sebagai matrik kosong yang akan dipakai sebagai tempat untuk menyimpan segmen-segmen karakter.

Kemudian dilakukan perulangan sejumlah anggota titik penting. Pada tiap perulangan akan diambil koordinat titik penting dan cabang dari titik penting tersebut. Kemudian dilakukan inisisalisasi matrik kosong **complete** sebagai tempat untuk menyimpan koordinat-koordinat segmen sementara. Variabel **i** dimulai dari 1 merupakan perhitungan untuk perulangan sejumlah cabang dari titik penting.

Perulangan pencarian segmen dimulai dengan mencari tetangga dari titik penting pada baris ke 14, bila terdapat tetangga maka ada kemungkinan sebuah segmen. **xNext** dan **yNext** merupaka baris dan kolom dari tetangga titik penting. Selanjutnya matrik **notComplete** digunakan untuk menyimpan koordinat segmen sementara yang pada awalnya berisi koordinat titik penting dan tetangganya. Selanjutnya koordinat **(xNext, yNext)** pada **thinnCopy** diset nilainya menjadi 0 sebagai tanda bahwa titik tersebut telah dilewati. Baris berikutnya menunjukkan variabel **notTp** yang diset 0 untuk memulai pencarian anggota segmen lainnya hingga bertemu dengan titik penting lainnya dan pencerian segmen tersebut akan berakhir.

Pada tiap perulangan menggunakan fungsi find_nextPixel yang membutuhkan parameter masukan **notComplete** sebagai matrik segmen yang belum lengkap, **xNext** dan **yNext** sebagai koordinat titik terakhir segmen untuk mencari anggota berikutnya, **thinnCopy** untuk mengecek dan mengeset titik koordinat yang telah dilewati, serta **obj.titikPenting** untuk melihat apakah anggota berikutnya merupakan titik penting atau bukan. Keluaran dari fungsi tersebut adalah **notComplete** yang telah terupdate berisi anggota baru, **xNext** dan **yNext** yang merupakan koordinat anggota segmen terbaru, **thinnCopy** yang terbaru yang telah berubah oleh pixel yang telah dilewati, serta **notTp** yang bernilai 0 bila anggota terbaru segmen bukan titik penting dan bernilai 1 bila anggota segmen merupakan titik penting. Perulangan berakhir dengan melihat nilai **notTp**.

Karena pada setiap titik penting terdapat kemungkinan memiliki jumlah cabang lebih dari 1, selanjutnya saat sebuah segmen telah didapatkan, hasil

dari **notComplete** disimpan pada **complete** dan ditandai sebagai anggota ke-**i**, yaitu urutan sebuah cabang dari titik penting.

Pada baris ke-31 anggota ke-**i** pada matrik **complete** akan berupa matrik kosong bila tetangga dari titik penting tidak ada atau telah dilewati. Selanjutnya nilai **i** akan ditambahkan 1 untuk memulai pencarian pada cabang berikutnya. Bila semua cabang telah selesai dilakukan pencarian segmen, hasil yang disimpan oleh matrik **complete** selanjutnya disimpan pada **segmen** pada urutan ke-**tp**. Matrik **segmen** akan berupa matrik kosong bila tidak anggota pada matrik titik penting.

4.3.8 Pengecekan Keberadaan *Hole*

Hole atau lubang pada karakter diambil sebagai fitur umum dari karakter. Keberadaan suatu *hole* teerdeteksi bila suatu segmen memiliki koordinat titik awal dan titik akhir yang sama. Selain itu, bila dua buah segmen saling bertautan, maka hal tersebut juga dimasukkan sebagai *hole*. Pada gambar 4.20 menunjukkan listing program pencarian *hole*.

Pengecekan sebuah *hole* terlebih dahulu dilakukan dengan melihat adanya titik penting atau tidak. Karena saat sebuah karakter tidak memiliki titik penting, kemungkinan besar karakter tersebut adalah karakter O, o, dan 0. Sehingga nilai **lubang** akan langsung diset 1.

Bila terdapat titik penting, maka yang dilakukan pertama kali adalah mengecek titik awal dan titik akhir segmen. Inisialisasi matrik **awalAkhir** sebagai matrik kosong dilakukan sebagai tempat menyimpan titik awal dan titik akhir segmen. Karena segmen berupa array yang di dalamnya masih terdapat segmen tiap titik penting, maka terjadi 2 perulangan yaitu yang ditandai dengan variabel **i** sejumlah banyaknya anggota **obj.segmen**, kemudian yang kedua ditandai dengan variabel **j** sejumlah anggota di dalam **obj.segmen** ke-**i**.

Fungsi matlab `any` digunakan untuk mengecek apakah **obj.segmen{i}{j}** tidak kosong. Setelah itu, diambil titik awal dan titik akhir dari segmen. Kemudian dilakukan pengecekan untuk memastikan bahwa titik awal dan titik akhir bukanlah sebuah titik ujung, selain itu dilakukan penomoran bahwa titik awal dan titik ujung tersebut merupakan anggota titik penting ke sekian.

```
function lubang = get.hole(obj)
   if isempty(obj.titikPenting)
      lubang = 1;
   else
      awalAkhir = [];
      for i = 1:length(obj.segmen)
        lcell = length(obj.segmen{i});
        for j = 1:lcell
          if any(obj.segmen{i}{j})
            awal = obj.segmen{i}{j}(1,:);
            akhir = obj.segmen{i}{j}(end,:);
            aw=0;ak=0;
            for k = 1: size(obj.titikPenting,1)
              this = obj.titikPenting(k,1:2);
              if obj.titikPenting(k,3)~= 1
                 if awal == this & akhir ~= this
                    aw = k;
                 elseif(awal~=this) & (akhir==this)
                    ak = k;
                 elseif(awal==this) & (akhir==this)
                    aw = k; ak=k;
                 end
              end
            end
            awalAkhir = [awalAkhir; aw ak];
          else awalAkhir = [awalAkhir];
          end
        end
      end

      if isempty(awalAkhir)
        lubang =0;
      else
        awalAkhir2=[awalAkhir(:,2),awalAkhir(:,1)];
        lubang=0;
        for a = 1: size(awalAkhir,1)
          if awalAkhir(a,1)==awalAkhir(a,2)
             lubang= lubang+1;
          else
            for b = a+1:size(awalAkhir,1)
              if awalAkhir(a,:)==awalAkhir(b,:)|...
                    awalAkhir(a,:)==awalAkhir2(b,:)
                lubang= lubang+1;
              else lubang = lubang;
              end
            end
          end
        end
      end
   end
end
```

Gambar 4.20 *Listing* program pencarian *hole*

Bila syarat awal terpenuhi, selanjutnya penomoran pada titik awal dan titik ujung ditambahkan pada matrik **awalAkhir** dengan format kolom pertama menyimpan titik awal dan kolom kedua menyimpan titik akhir.

Setelah perulangan selesai, selanjutnya tinggal mengecek matrik **awalAkhir** kosong atau tidak. Jika kosong maka **lubang** akan bernilai 0 yang berarti tidak ada *hole* pada karakter. Jika **awalAkhir** memiliki anggota, akan dilihat jika ada kesamaan titik awal dengan titik akhir, maka **lubang** akan ditambahkan 1. Bila titik awal tidak sama dengan titik akhir, maka akan dilihat anggota titik ujung lainnya yang memiliki kesamaan dengan titik awal dan titik akhir segmen tersebut, yang berarti bahwa kedua segmen saling bertautan, dan **lubang** kembali ditambahkan 1 bila hal tersebut ditemukan. Selain dari kondisi di atas, jumlah **lubang** akan tetap sama seperti semula.

4.3.9 Pembacaan Bentuk Geometri tiap Segmen

Pembacaan bentuk geometri tiap segmen melibatkan sebuah *class* **BagianKarakter** yang berfungsi untuk mengklasifikasikan bentuk segmen-segmen dari karakter sesuai dengan metode fuzzy yang diberikan. Listing program dari pembacaan bentuk geometri terlihat seperti pada gambar 4.21.

```
function bl = get.bagianLain(obj)
   bl =[];
   for i = 1:length(obj.segmen)
     lcell = length(obj.segmen{i});
     for j = 1:lcell
       if any(obj.segmen{i}{j})
         s = obj.segmen{i}{j};
         if (length(s)>4)
         sb = BentukGeometri(obj.boundBox, s);
           if sb.geometri == 13
              bl =[bl];
           elseif (sb.geometri < 13) & ...
                    (sb.size_bagian > 1)
              bl=[bl; sb.geometri,sb.posisi(1), ...
                   sb.posisi(2)];
           end
         end
       else bl =[bl];
       end
     end
   end
end
```

Gambar 4.21 *Listing* program pembacaan bentuk geometri tiap segmen

Matrik **bl** nantinya akan menyimpan bentuk-bentuk geometri dari semua segmen karakter selain bentuk *loop*, karena diasumsikan *hole* telah didapatkan sebelumnya.

Selanjutnya diambil tiap anggota **obj.segmen{i}{j}** yang kemudian akan disimpan sementara oleh variabel **s**. Pembacaan bentuk geometri segmen hanya dilakukan pada segmen dengan jumlah anggota lebih dari 4. Setelah itu

dilakukan pengklasifikasian segmen oleh *class* BentukGeometri, yang mana parameter masukannya adalah **obj.boundBox** sebagai *bound box* dari karakter, dan **s** sebagai segmen yang berisi koordinat-koordinat dari sebuah segmen.

Variabel **sb** akan menyimpan bentuk geometri yang direpresentasikan sebagai angka sesuai urutan bentuk geometri beserta ukuran segmen. Pada percabangan selanjutnya dilakukan untuk mengecek bentuk geometri yang diakses dengan **sb.geometri**. Bila **sb.geometri** bernilai 13 yang merupakan sebuah loop, maka **bl** tidak akan ditambahkan anggotanya. Kemudian bila **sb.geometri** memiliki nilai di bawah 13 dan **sb.size_bagian** yang merupakan ukuran segmen memiliki nilai lebih dari 1 atau dapat dikatakan bahwa ukuran segmen tidak kecil, maka **bl** akan ditambahkan bentuk geometrinya **sb.geometri** pada kolom pertama, kemudian **sb.posisi(1)** pada kolom kedua yang menunjukkan posisi vertikal segmen, dan kolom ketiga ditempati oleh **sb.posisi(2)** sebagai penunjuk posisi segmen secara horizontal.

4.3.10 Keluaran Pembacaan Karakter

Keluaran pembacaan karakter adalah berupa fitur karakter yang terformat sebagai matrik berukuran 118x1. Di mana 118 merupakan fitur-fitur bentuk segmen karakter yang telah diambil sebelumnya. Keluaran pembacaan karakter ini yang nantinya digunakan sebagai data masukan pada jaringan *Neural Network Backpropagation.* Gambar 4.22 menunjukkan implementasi keluaran pembacaan karakter dalam bentuk listing program.

Pada baris kedua dilakukan inisialisasi matrik berukuran 118x1 yang berisi nilai 0 semua dan disimpan oleh variabel **this**. Kemudian secara berurutan dimasukkan fitur-fitur umum terlebih dahulu yaitu template untuk pertama kali. Karena nilai **obj.template** telah disesuaikan memiliki nilai dari 1 sampai 4, maka pengubahan **this** langsung dilakukan pada urutan **obj.template** dengan memberi nilai 1 yang menunjukkan template karakter. Selanjutnya pengubahan dilakukan pada fitur *hole* yang berada pada urutan ke-5 dan ke-6. Karena nilai *hole* adalah 1 dan 2, maka pengaksesan fitur dilakukan dengan menambahkan 4 pada nilai **obj.hole** yaitu dengan menuliskan `this(obj.hole+4)` yang diisi dengan nilai 1. Begitu pula untuk pengubahan fitur terpisah dan posisi titik mengikuti nilai yang dibawa oleh **obj.terpisah** dan **obj.posisiTitik**.

Bila fitur umum telah dimasukkan, selanjutnya adalah tahap memasukkan fitur khusus yaitu bentuk geometri segmen. Perulangan akan dilakukan sejumlah anggota **obj.bagianLain**. Karena penempatan fitur berdasarkan posisi, maka pengecekan dilakukan pada posisi anggota **obj.bagianLain** ke-**i**.

```
function this = get_fitur(obj)
   this=zeros(118,1);
   this(obj.template)=1;
   if any(obj.hole)
     this(obj.hole + 4) = 1;
   elseif any(obj.terpisah)
     this(obj.terpisah) = 1;
   elseif any(obj.posisiTitik)
     this(obj.posisiTitik)=1;
   else this= [this];
   end

   for i = 1:size(obj.bagianLain,1)
     if (obj.bagianLain(i,3:4)== [1 1])
       this(obj.bagianLain(i,1) +10)=1;
     elseif (obj.bagianLain(i,3:4)== [1 2])
       this(obj.bagianLain(i,1) +22)=1;
     elseif (obj.bagianLain(i,3:4)== [1 3])
       this(obj.bagianLain(i,1) +34)=1;
     elseif (obj.bagianLain(i,3:4)== [2 1])
       this(obj.bagianLain(i,1) +46)=1;
     elseif (obj.bagianLain(i,3:4)== [2 2])
       this(obj.bagianLain(i,1) +58)=1;
     elseif (obj.bagianLain(i,3:4)== [2 3])
       this(obj.bagianLain(i,1) +70)=1;
     elseif (obj.bagianLain(i,3:4)== [3 1])
       this(obj.bagianLain(i,1) +82)=1;
     elseif (obj.bagianLain(i,3:4)== [3 2])
       this(obj.bagianLain(i,1) +94)=1;
     elseif (obj.bagianLain(i,3:4)== [3 3])
       this(obj.bagianLain(i,1) +106)=1;
     else this = [this];
     end
   end
end
```

Gambar 4.22 *Listing* program penentuan fitur

Bila posisi anggota **obj.bagianLain** ke-**i** berada pada atas-kiri, yang ditunjukkan pada kolom ke-3 dan ke-4 bernilai [1 1], maka pengubahan fitur **this** menjadi nilai 1 dilakukan pada *range* 11 hingga 22, yaitu dengan menambahkan 10 pada nilai bentuk geometri yang berada pada kolom pertama **obj.bagianLain**.

Selanjutnya jika posisi berada pada atas-tengah, atau nilai kolom ke-3 dan ke-4 adalah [1 2], maka pengubahan fitur **this** dilakukan pada urutan ke-

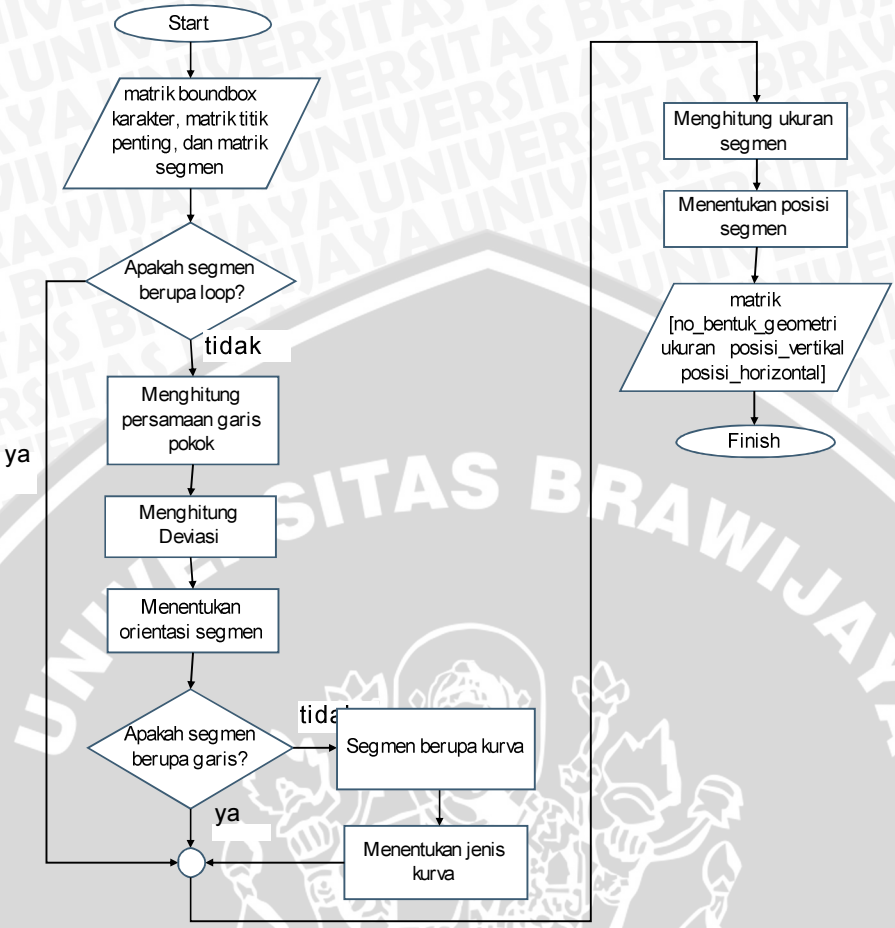
23 hingga ke-34. Percabangan ketiga adalah untuk posisi atas-kanan yang ditunjukkan oleh nilai [1 3], pengubahan dilakukan pada *range* 35 hingga 46.

Baris ke-20 pada listing di atas menunjukkan bila posisi segmen berada pada tengah-kiri atau [2 1], pengubahan **this** dilakukan pada urutan baris ke-47 sampai 58. Kemudian bila segmen berada pada posisi tengah-tengah yang nilainya [2 2], maka fitur pada posisi antara 59 hingga 70 akan mengalami perubahan. Sedangkan untuk posisi tengah-kanan atau [2 3], maka fitur akan dimasukkan pada urutan ke-71 hingga 82.

Masih terdapat 3 posisi terakhir yaitu bawah-kiri yang nilainya [3 1] rentang fiturnya berada pada baris **this** ke-83 hingga 94. Selanjutnya pada posisi bawah-tengah yaitu [3 2], **this** akan berubah menjadi 1 untuk *range* 95 sampai 106. Posisi terakhir yaitu bawah-kanan atau [3 3], menempati baris ke-107 hingga 118.

4.4 Perancangan dan Implementasi Pembacaan Bentuk Geometri

Pembacaan bentuk geometri merupakan bagian dari pengambilan fitur karakter. Untuk memudahkan dalam pemilahan proses, maka pembacaan bentuk geometri dijadikan sebuah class tersendiri yang di dalamnya menggunakan metode fuzzy. Segmen karakter akan diklasifikasikan ke dalam 13 bentuk geometri yang telah ditetapkan sebelumnya. Gambar 4.23 menunjukkan diagram alir dalam pembacaan bentuk geometri.



Gambar 4.23 Diagram alir proses pembacaan bentuk geometri

4.4.1 Masukan Proses Pembacaan Bentuk Geometri

Sebagai masukan dalam pembacaan bentuk geometri adalah *bound box* karakter dan matrik segmen yang berisi koordinat-koordinat pixel yang membentuk sebuah segmen. Implementasi dalam program dapat dilihat pada gambar 4.24 berikut ini.

```
function obj = BentukGeometri(bbKarakter, segmen)
            obj.bbKarakter = bbKarakter;
            obj.koord_elemen = segmen;
end
```

Gambar 4.24 *Listing* program masukan *class* BentukGeometri

Baris pertama menunjukkan bahwa *class* `BentukGeometri` memerlukan dua masukan yaitu **bbKarakter** sebagai bound box karakter dan **segmen** yang berisi koordinat-koordinat pixel dari sebuah segmen. Keluaran dari *class* ini adalah sebuah objek **obj** yang memiliki atribut **bbKarakter** yang menyimpan bound box karakter dan **koord_elemen** yang menyimpan **segmen**.

4.4.2 Pengecekan Loop

Sebelum memulai pengklasifikasian bentuk geometri, hal yang perlu diklasifikasi terlebih dahulu adalah bentuk *loop*. Sebuah *loop* terdeteksi bila dalam satu segmen tersebut memiliki titik awal dan titik akhir yang sama. Listing program untuk memastikan sebuah *loop* ditunjukkan pada gambar 4.25.

```
start=obj.koord_elemen(1,:);
ending=obj.koord_elemen(end,:);
  if start == ending
     this.size = size(obj.bbKarakter, 'loop', ...
                      obj.koord_elemen, 'x');
     this = [13 this.size];
```

Gambar 4.25 *Listing* program pengecekan *loop*

Variabel **start** merupakan titik awal segmen, dan **ending** merupakan titik akhir segmen. Bila **start** sama dengan **ending**, maka proses berikutnya adalah langsung menuju penghitungan ukuran.

4.4.3 Penghitungan Garis Pokok

Garis pokok merupakan garis linier yang menghubunhkan antara titik awal segmen dengan titik akhir segmen. Persamaan garis pokok harus memenuhi persamaan umum $y=ax+c$. Gambar berikut menunjukkan penghitungan garis pokok.

```
function this = baseLine(segmen)
    start = segmen(1,:);
    ending = segmen(end,:);
    x = [start(1,1), ending(1,1)];
    y = [start(1,2), ending(1,2)];
    this = polyfit(x,y,1);
end
```

Gambar 4.26 *Listing* program penghitungan garis pokok

Variabel **start** merupakan koordinat titik awal segmen. Sedangkan **ending** merupaka titik akhir segmen. Selanjutnya variabel **x** menyimpan koordinat baris dari **start** dan **ending**. Kemudian variabel **y** menyimpan koordinat kolom dari **start** dan **ending**. Penghitungan garis linier menggunakan fungsi matlab `polyfit` yang masukannya berupa titik **x** dan **y**, dan parameter ketiga menunjukkan orde dari persamaan yang dicari. Karena garis linier memiliki orde 1, maka parameter ketiga dari fungsi `polyfit` ini adalah 1.

4.4.4 Pendeteksian Bentuk Dasar Segmen

Bentuk dasar segmen disimpulkan berdasarkan rata-rata deviasi yang diperoleh. Deviasi sendiri merupakan jarak tiap elemen segmen ke garis pokok. Listing program berikut merupakan implementasi penghitungan deviasi.

```
function this = deviasi(baseLine, segmen)
    %gradien tegak lurus dg base line
    gradientBL = baseLine(1);
    cBL = baseLine(2);
    gradient2 = -1/gradientBL;
    devSegmen=[];
    for s = 2:length(segmen)-1
       x1 = segmen(s,1);
       y1 = segmen(s,2);

       c2 =(y1-(gradient2*x1)); %b untuk pers garis
yg tegak lurus dg base line
       %cari titik potong k-2 garis
       xp=(c2-cBL)/(gradientBL-gradient2);
       yp=(gradient2*xp)+c2;

       %jarak (x1,y1) ke (xp,yp)
       devSegmen(s)= sqrt(((y1-yp)^2)+((x1-xp)^2));
    end;

    n = length(devSegmen);
    this =(sum(devSegmen)/n);
end
```

Gambar 4.27 *Listing* program penghitungan deviasi

Karena penghitungan deviasi membutuhkan persamaan garis pokok dan segmen, maka masukan dari fungsi adalah **baseLine** yang menyimpan persamaan garis pokok dan **segmen** yang menyimpan koordinat semua elemen segmen.

Berikutnya adalah mencari gradient yang tegak lurus dengan gradient garis pokok. Variabel **gradientBL** menyimpan gradient garis pokok, sedangkan **cBL** menyimpan nilai *c* dari garis pokok. Selanjutnya **gradient2** merupakan gradient tegak lurus dengan garis pokok yang didapatkan dengan membagi -1 dengan gradient garis pokok.

Baris ke-7 merupakan inisisalisasi **devSegmen** sebagai matrik kosong yang akan menympan nilai deviasi tiap elemen segmen. Selanjutnya dilakukan perulangan dari elemen kedua segmen hingga elemen sebelum titik akhir segmen.

Pada tiap perulangan diambil titik **x1** dan **y1** dari elemen **segmen** yang merupakan baris dan kolom elemen. Selanjutnya dicari nilai **c2** yang

merupakan persamaan garis baru antara garis pokok ke elemen yang saling tegak lurus. Penghitungan garis baru tersebut menggunakan **gradient2** yang sebelumnya telah didapatkan.

Setelah didapatkan persamaan garis baru, kemudian dicari titik potong kedua garis pada garis pokok. Variabel **xp** dan **yp** merupakan titik potong kedua garis. Baru kemudian **devSegmen** ditambahkan anggota ke-**s** berupa nilai jarak antara garis pokok ke elemen yang dicari dengan menggunakan persamaan pitagoras.

Setelah semua elemen didapatkan jarak deviasinya, maka berikutnya dicari rata-rata dari deviasi ini dengan menjumlah semua deviasi dan membagi dengan banyaknya anggota **devSegmen**.

Bila nilai deviasi telah didapatkan, baru kemudian bentuk dasar segmen bisa disimpulkan. Suatu segmen akan berupa garis bila nilai deviasi kurang dari 5 dan berupa kurva bila sebaliknya. Untuk bentuk segmen garis disimbolkan dengan *char* **g** dan untuk segmen kurva disimbolkan dengan *char* **k**. implementasi dalam bentuk listing program untuk pengamblan bentuk dasar terlihat pada gambar 4.28 berikut.

```
function this = bentukDasar(deviasi)
    %obj.deviasi = get.deviasi(obj.element);
    if deviasi < 5
       this  = 'g';
    else this  = 'k';
    end
end
```

Gambar 4.28 *Listing* program penentuan bentuk dasar segmen

4.4.5 Pendeteksian Orientasi Segmen

Orientasi segmen merupakan arah kemiringan segmen. Terdapat 4 klasifikasi orientasi yaitu vertikal, horizontal, serong kiri, dan serong kanan. Untuk penghitungannya pertama kali diperlukan sudut theta yang merupakan sudut yang dibentuk garis pokok dengan garis horizontal. Selanjutnya pencarian orientasi dilakukan berdasarkan fungsi keanggotaan fuzzy berikut ini :

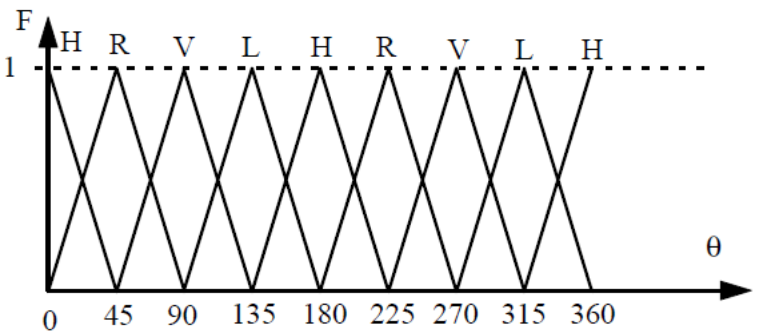
$$FH(\theta) = 1 - \min\{\min[|\theta|, |180 - \theta|, |360 - \theta|]/45, 1\}$$

$$FV(\theta) = 1 - \min\{\min[|90 - \theta|, |270 - \theta|]/45, 1\}$$

$$FR(\theta) = 1 - \min\{\min[|45 - \theta|, |225 - \theta|]/45, 1\}$$

$$FL(\theta) = 1 - \min\{\min[|135 - \theta|, |315 - \theta|]/45, 1\}$$

dengan F merupakan nilai dari fungsi keanggotaan di atas yang dapat digambarkan dengan diagram pada gambar 4.29.



Gambar 4.29 Fungsi keanggotaan orientasi segmen

Sedangkan gambar 4.30 menunjukkan implementasi dalam bentuk listing program.

```
function this = orientasi(baseLine)
    theta = atand(baseLine(1));
    if theta <= 0
        theta = theta+180;
    end;

    fh = 1 - (min([abs(theta)/45, abs(180-...
            theta)/45, abs(360-theta)/45, 1]));
    fv = 1 - (min([abs(90-theta)/45, abs(270-...
            theta)/45, 1]));
    fr = 1 - (min([abs(45-theta)/45, abs(225-...
            theta)/45, 1]));
    fl = 1 - (min([abs(135-theta)/45, abs(315-...
            theta)/45, 1]));

    nilai_segmen = max([fh fv fr fl]);
    if nilai_segmen == fh
        this = 'h';
    elseif nilai_segmen == fv
        this = 'v';
    elseif nilai_segmen == fr;
        this = 'rs';
    else this = 'ls';
    end
end
```

Gambar 4.30 *Listing* program penentuan orientasi segmen

Masukan pada fungsi `orientasi` adalah **baseLine**, karena perhitungan **theta** memerlukan gradient garis pokok. Hal ini terlihat pada baris kedua, di mana variabel **theta** merupakan hasil dari arc tangensial dari kolom pertama **baseLine** yang merupakan gradient dari **baseLine**. Namun, bila **theta** memiliki nilai negatif, maka **theta** perlu dijadikan positif yang kesetaraan sudutnya senilai dengan menambahkan 180 pada **theta**.

Berikutnya adalah pengimplementasian fungsi fuzzy, di mana **fh** merupakan nilai horizontal, **fv** akan menyimpan nilai vertikal, **fr** adalah nilai serong kanan dan **fl** adalah nilai untuk serong kiri.

Dari keempat nilai yang didapatkan selanjutnya diambil nilai tertinggi untuk mendapatkan orientasi segmen. Orientasi segmen akan berupa data *char* yang mana **h** merupakan horizontal, **v** adalah vertikal, **rs** merupakan serong kanan, dan **ls** untuk serong kiri.

4.4.6 Pencarian Jenis Kurva

Pencarian jenis kurva hanya berlaku pada segmen dalam kategori kurva atau yang memiliki nilai *char* **k**. Sedangkan untuk segmen berupa garis tidak akan melewati tahap ini.

Jenis kurva diperlukan karena kurva memiliki 2 sisi yang berbeda, yaitu sisi terbuka dan tertutup. Untuk jenis kurva hanya akan dicari 2 parameter yang membedakan yaitu arah hadap sisi terbuka ke kanan atau ke kiri. Parameter ini nantinya bergabung dengan orientasi untuk dipakai dalam klasifikasi 8 bentuk kurva.

Gambar 4.31 merupakan implementasi pencarian kurva dalam bentuk program.

```
function this = jenisKurva(bentukDasar, segmen, ...
                    baseLine)
   if strcmp(bentukDasar, 'k')
     x1 = segmen(2,1);
     y1 = segmen(2,2);
     x2 = segmen(end-1,1);
     y2 = segmen(end-1,2);
     funcY = polyfit([x1, x2],[y1, y2], 1);
     cf = funcY(2);
     if (baseLine(2)> 0) & (baseLine(2) <= cf)...
         | ((baseLine(2)< 0) & (baseLine(2) <= cf))
       this = 'r';
     elseif (baseLine(2)> 0) & (baseLine(2)>cf)...
         |((baseLine(2)< 0) & (baseLine(2) > cf))
       this = 'l';
     end
   end
end
```

Gambar 4.31 *Listing* program penentuan jenis kurva

Fungsi `jenisKurva` memerlukan 3 masukan yaitu **bentukDasar**, **segmen**, dan **baseLine**. Pada baris ketiga adalah memastikan bahwa bentuk dasar segmen berupa kurva. Selanjutnya adalah menghitung persamaan garis linier yang diambil dari titik kedua segmen ke titik sebelum titik akhir.

Persamaan ini disimpan oleh variabel **funcY**. Variabel **cf** adalah nilai *c* dari **funcY**.

Nilai **cf** kemudian dibandingkan dengan nilai *c* pada garis pokok yang diakses dari variabel **baseLine** kolom kedua. Jenis kurva akan merupakan kurva hadap kanan bila memenuhi salah satu persyaratan berikut. Syarat pertama nilai **baseLine(2)** adalah lebih dari 0 dan kurang dari atau sama dengan nilai **cf**. Sedangkan syarat kedua adalah jika nilai **baseLine(2)** kurang dari 0 dan kurang dari atau sama dengan **cf**. Pilihan kedua yaitu untuk jenis kurva hadap kiri. Persyaratan untuk kurva hadap kiri adalah jika nilai **baseLine(2)** lebih dari 0 maka harus lebih dari **cf**. Bila **baseLine(2)** kurang dari 0, maka nilainya harus lebih besar dari **cf**.

4.4.7 Penghitungan Ukuran

Ukuran segmen dicari dengan membandingkan segmen dengan keseluruhan karakter. Bila segmen berupa garis horizontal atau vertikal, maka ukuran pembanding yang diambil adalah ukuran karakter sesuai orientasi garis. Namun, bila segmen berupa kurva atau garis serong kiri dan serong kanan, maka ukuran pembanding yang diambil adalah luas *bound box* karakter. Listing program pencarian ukuran ditunjukkan pada gambar 4.32.

```
function this = size(bb, bentukDasar, segmen, ...
                     orientasi)
    pjgLbr=[abs(bb(2,1)-bb(1,1)), ...
            abs(bb(2,2)-bb(1,2))];
    if (strcmp(bentukDasar,'g'))& ...
            (strcmp(orientasi,'h'))
      jumtitik = length(segmen);
      boundBox = pjgLbr(1,2);
    elseif (strcmp(bentukDasar,'g'))& ...
            (strcmp(orientasi,'v'))
      jumtitik = length(segmen);
      boundBox = pjgLbr(1,1);
    else
      lrow = (max(segmen(:,1))-min(segmen(:,1)))+1;
      lcol = (max(segmen(:,2))-min(segmen(:,2)))+1;
      jumtitik = lrow*lcol;
      boundBox = pjgLbr(1,1)*pjgLbr(1,2);
    end

    if jumtitik > 0.5*boundBox
      fs = 0;
      fm = -(abs(2/boundBox*jumtitik-1))+1;
      fl = (-2/boundBox*jumtitik)+3;
    elseif jumtitik == 0.5*boundBox
      fs = (-2/boundBox*jumtitik);
      fm = -(abs(2/boundBox*jumtitik-1))+1;
      fl = 0;
    elseif jumtitik < 0.5*boundBox
      fl = 0;
      fm = -(abs(2/boundBox*jumtitik-1))+1;
      fs = (-2/boundBox*jumtitik)+1;
    end;

    nilai_size = max ([fs fm fl])
    if nilai_size == fs
      this = 1; %kecil
    elseif nilai_size == fm
      this = 2; %sedang/medium
    elseif nilai_size == fl
      this = 3; %besar
    end;
end
```

Gambar 4.32 *Listing* program penentuan ukuran segmen

Baris pertama menunjukkan bahwa masukan pada fungsi adalah **bb** sebagai bound box karakter, **bentukDasar**, **segmen,** dan **orientasi**. Berikutnya variabel **pjgLbr** adalah ukuran vertikal karakter dan ukuran horizontal karakter.

Baris ketiga adalah untuk mengeset **jumtitik** dan **boundBox**. Jika bentuk dasar segmen berupa garis dan orientasi horizontal, maka **jumtitik** adalah panjang segmen dan **boundBox** merupakan ukuran horizontal karakter yang diambil dari **pjgLbr** kolom kedua.

Percabangan ketiga terjadi untuk segmen yang memiliki bentuk dasar garis dan berorientasi vertikal. Variabel **jumtitik** di sini akan diset sebagai panjang segmen, sedangkan **boundBox** diberi nilai ukuran vertikal karakter yang diambil dari **pjgLbr** kolom pertama.

Lain halnya bila segmen tersebut memiliki bentuk dasar kurva atau garis dengan orientasi serong, maka **jumtitik** merupakan luas dari bound box segmen itu sendiri. Sedangkan **boundBox** merupakan luas karakter.

Selanjutnya adalah mengikuti metode fuzzy untuk pencarian ukuran karakter. Di mana dari ketiga nilai **fs**, **fm**, dan **fl** akan diambil nilai terbesar dan akan menjadi ukuran dari segmen.

4.4.8 Pencarian Posisi

Posisi segmen akan ditempatkan ke dalam 9 posisi yang mungkin. Ke-9 posisi tersebut meliputi 3 posisi secara vertikal dan 3 posisi secara horizontal yang kemudian digabung menjadi 1 dan menjadi posisi segmen di dalam karakter. *Listing* program pencarian posisi ditunjukkan pada gambar 4.33.

```
function this = cari_posisi(segmen, bb)
  xmin = bb(1,1); ymin= bb(1,2);
  xmax = bb(2,1); ymax= bb(2,2);
  kxbb = (xmax-xmin); kybb = (ymax-ymin);

  sxmin = min (segmen(:,1));
  symin = min(segmen(:,2));
  sxmax = max(segmen(:,1));
  symax = max(segmen(:,2));
  sxbb = sxmax-sxmin;
  sybb = symax-symin;

  pAtasBawah = ini_posisi(xmin, kxbb, sxmin, sxbb);
  pKiriKanan = ini_posisi(ymin, kybb, symin, sybb);
  this = [pAtasBawah; pKiriKanan];
```

Gambar 4.33 *Listing* program pencarian posisi

Masukan untuk pencarian posisi adalah segmen dan bound box karakter. Pencarian posisi ini melibatkan sebuah fungsi ini_posisi untuk menentukan 3 posisi secara vertikal atau horizontal sesuai dengan masukan yang diberikan. Masukan fungsi ini_posisi sendiri adalah titik minimal segmen, panjang segmen, titik minimal karakter, dan panjang karakter.

Pada baris ke-13 adalah untuk pencarian posisi secara vertikal yang akan disimpan oleh variabel **pAtasBawah**. Karena pencarian secara vertikal, maka titik minimal segmen yang dibutuhkan adalah baris minimal dari segmen yang disimpan oleh **xmin**. Sedangkan panjang segmen yaitu **kxbb** sama dengan

tinggi segmen yang merupaka selisih dari baris tertinggi dan terendah segmen. Berikutnya masukan ketiga adalah **sxmin** yang merupakan baris terendah dari karakter, dan **sxbb** merupakan tinggi karakter.

Begitu pula dengan pencarian posisi secara horizontal yang hasilnya disimpan pada **pKiriKanan**. Masukan fungsi ini_posisi yang pertama adalah kolom minimal dari segmen yang dibawa oleh **ymin**. Masukan berikutnya adalah panjang segmen yang nilainya disimpan oleh varibel **kybb**. Sedangkan **symin** adalah kolom minimal dari karakter, dan **sybb** adalah panjang karakter. Keluaran dari fungsi cari_posisi adalah matrik 2x1, yang mana baris pertama menunjukkan posisi atas-bawah dan baris kedua menunjukkan posisi dari kiri ke kanan.

4.4.9 Keluaran Pembacaan Bentuk Geometri

Keluaran pembacaan geometri meliputi bentuk geometri yang memiliki rentang nilai antara 1 sampai 13, serta ukuran dan posisi segmen. Gambar 4.34 menunjukkan listing program keluaran bentuk geometri.

Fungsi ubahAngka merupakan fungsi untuk mengubah bentuk geometri yang semula berupa string menjadi angka yang memudahkan dalam pemetaan fitur karakter nantinya. Parameter masukannya adalah berupa bentuk dasar, orientasi, jenis kurva, dan ukurannya.

Sebelum memulai pengubahan, hal pertama yang perlu dilakukan adalah menginisialisasi sebuah matrik berukuran 1x2 yang beranggotakan 0 sebagai tempat untuk menyimpan keluaran. Kolom pertama dari matrik ini adalah nomor dari bentuk geometri yang akan berisi nilai 1 sampai 13, dan kolom kedua merupakan nomor dari ukuran yang memiliki rentang nilai 1 sampai 3.

```
function this = ubahAngka(bentukDasar,orientasi,...
                  jenisKurva, size)

this =zeros(1,2);

if strcmp(bentukDasar,'g')
   if strcmp(orientasi,'h')
      this(:)=[1 size];
   elseif strcmp(orientasi,'v')
      this=[2 size];
   elseif strcmp(orientasi,'rs')
      this(:)=[3 size];
   else this(:)=[4 size];
   end
else
   if strcmp(jenisKurva,'r')
        if strcmp(orientasi,'h')
            this(:)=[5 size];
        elseif strcmp(orientasi,'v')
            this(:)=[6 size];
        elseif strcmp(orientasi,'rs')
            this(:)=[7 size];
        else this(:)=[8 size];
        end
   else
        if strcmp(orientasi,'h')
            this(:)=[9 size];
        elseif strcmp(orientasi,'v')
            this(:)=[10 size];
        elseif strcmp(orientasi,'rs')
            this(:)=[11 size];
        else this(:)=[12 size];
        end
    end
end
```

Gambar 4.34 *Listing* program keluaran pembacaan geometri

Selanjutnya untuk mendapatkan keluaran berupa bentuk geometri yang berupa angka beserta ukurannya adalah dengan memanfaatkan fungsi percabangan yang akan mengecek masukan untuk memberikan hasil yang sesuai.

4.5 Perancangan dan Implementasi Training Jaringan NNB

Sebelum dapat mengenali karakter, maka jaringan *Neural* perlu mendapatkan pelatihan. Pelatihan di sini dilakukan dengan menyediakan contoh data fitur-fitur karakter beserta nilai biner dari karakter tersebut sebagai target keluaran yang diharapkan.

4.5.1 Masukan Training Jaringan NNB

Masukan sistem berupa fitur-fitur dari karakter yang direpresentasikan dengan matrik berukuran 118 x 400. Di mana 118 merupakan fitur biner yang didapatkan dari tiap karakter dan 400 merupakan jumlah total karakter yang

akan ditraining. Perincian 400 data ini adalah huruf besar dan huruf kecil masing-masing 150 karakter, sedangkan angka dan tanda baca masing-masing 50 karakter.

Begitu pula dengan target yang disediakan yaitu matrik berukuran 7x400. Di mana 7 merupakan nilai biner dari karakter dan 400 adalah jumlah datanya.

4.5.2 Pembuatan Jaringan Neural dan Training

Pembuatan jaringan *Neural* adalah dengan memanfaatkan *Neural Network Toolbox* dari Matlab. Gambar 4.35 berikut merupakan listing program jaringan *Neural.*

```
function tr = jaring_neural()


clc;
clear all;
close all;

load input10;
masukann = input10;

load targetB10;
keluarann = targetB10;

jaring_neural= newff(masukann,keluarann, ...
[100],{'logsig', 'tansig'},'trainscg','learngdm');
jaring_neural = init(jaring_neural);
jaring_neural.trainParam.epochs = 50000;
jaring_neural.trainParam.goal = 0.01;
jaring_neural.trainParam.lr = 0.05;
jaring_neural.trainParam.mc = 0.95;

[jaringanTrainingB, tr] = train (jaring_neural,...
              masukann, keluarann);
save jaringanTrainingB;
end
```

Gambar 4.35 *Listing* program pembuatan dan *training* jaringan *Neural*

Fungsi `jaring_neural` adalah fungsi yang menjalankan jaringan *Neural* dan melakukan training secara langsung. Variabel **masukann** mengambil nilai dari matrik **input10** yang berukuran 118x400. Fungsi `load` adalah untuk memanggil matrik **input10** yang telah disimpan sebelumnya. Sedangkan variabel **keluarann** mengambil nilai dari **targetB10** yang berupa matrik berukuran 7x400.

Selanjutnya pembuatan jaringan Neural dengan menggunakan fungsi matlab newff yang membutuhkan parameter **masukann** sebagai data

masukan, **keluaraann** sebagai target, angka **100** merupakan jumlah *hidden layer* (hanya menggunakan 1 *hidden layer*), **'logsig'** merupakan fungsi transfer yang digunakan antara layer masukan dan *hidden layer*, **'tansig'** merupakan fungsi transfer antara *hidden layer* dan layer keluaran, **'trainscg'** merupakan fungsi training yang digunakan untuk pelatihan jaringan, dan **'learngdm'** adalah fungsi pembelajaran bobot/bias dari *Backpropagation*. Jaringan yang telah dibuat selanjutnya disimpan dalam **jaring_neural**.

Selanjutnya fungsi `init` digunakan untuk memperbarui bobot dan bias dari **jaring_neural**. Berikutnya terdapat 4 baris pelengkap jaringan yang dapat digunakan untuk menambah performa jaringan dalam melakukan training. Pelengkap pertama merupakan jumlah *epoch* maksimal yang dapat dilakukan jaringan, kemudian nilai *goal* minimal yang berusaha dicapai jaringan, selanjutnya *learning rate* digunakan untuk menjaga training jaringan agar tetap stabil dan tidak berosilasi terlalu tinggi sehingga error yang didapatkan juga dapat diminimalisir, terakhir adalah momentum konstan yang diberikan selama pelatihan.

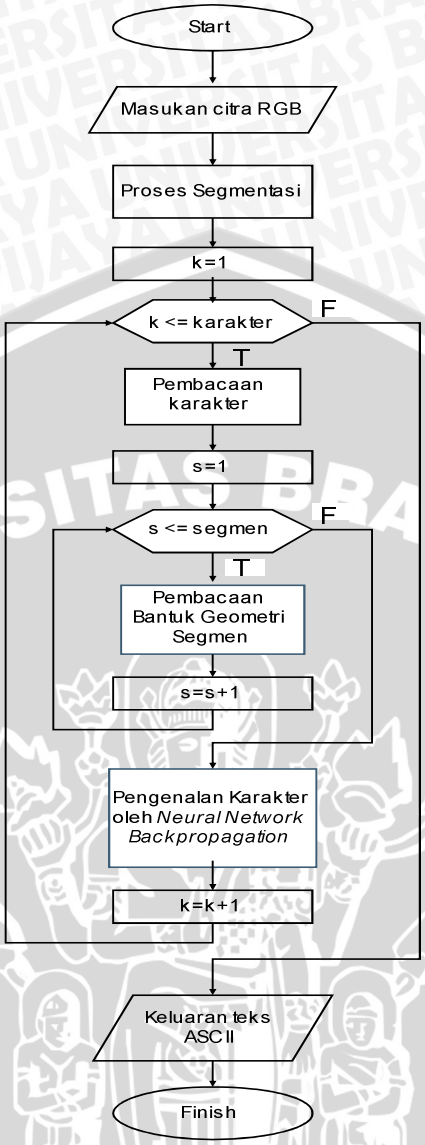
Setelah jaringan *Neural* selesai dibuat, berikutnya adalah melakukan training dengan fungsi matlab train yang parameter masukannya adalah jaringan *Neural* yang telah dibuat, kemudian data masukan yang akan dtraining, serta target yang diharapkan. Jaringan yang telah ditaining selanjutnya disimpan oleh **JaringanTrainingB** dan selanjutnya disimpan dalam bentuk file matlab dengan menggunakan fungsi `save` sehingga dapat dipanggil dan digunakan sewaktu-waktu dibutuhkan.

4.5.3 Keluaran Proses Training Jaringan Neural

Keluaran dari proses training adalah sebuah jaringan yang telah mendapat training dan siap untuk digunakan dalam pengenalan karakter. Jaringan keluaran tersebut disimpan dalam variabel **JaringanTrainingB**.

4.6 Perancangan dan Implementasi Sistem Pengenalan Karakter

Sistem pengenalan karakter meliputi keseluruhan proses di atas dan memanfaatkan jaringan *Neural* yang telah mendapat training. Gambar 4.36 menunjukkan diagram alir dari system pengenalan karakter.



Gambar 4.36 Diagram alir sistem pengenalan karakter

4.6.1 Masukan Sistem Pengenalan Karakter

Masukan sistem adalah citra dokumen yang berisi beberapa karakter dengan ekstensi bmp, jpg, gif, png atau tif. Selanjutnya citra ini akan melewati tahap segmentasi sebagai proses pemisahan karakter-karakter dalam citra masukan menjadi karakter-karakter tunggal.

4.6.2 Proses Pengenalan Karakter

Karakter-karakter tunggal yang didapatkan dari proses segmentasi selanjutnya melewati tahap pemisahan segmen-segmen pembentuk karakter untuk diambil fitur bentuk geometrinya. Dari fitur yang didapatkan ini

selanjutnya proses pengenalan dilakukan oleh jaringan *Neural* yang telah dilatih sebelumnya.

4.6.3 Keluaran Sistem Pengenalan Karakter

Keluaran dari sistem pengenalan karakter adalah file teks dari citra masukan yang berisi karakter yang telah dikenali. Selanjutnya file ini dapat disimpan dengan ekstensi *txt.*

4.6.4 *User Interface* Sistem Pengenalan Karakter

Fungsi yang terdapat dalam *Interface* Sistem Pengenalan Karakter meliputi:

1. Judul Program, merupakan nama program.
2. Tombol Buka, berfungsi untuk membuka file citra dokumen.
3. Panel Gambar, berfungsi untuk meletakkan citra dokumen masukan yang diambil dari komputer.
4. Tombol Ubah, berfungsi untuk mengubah citra masukan menjadi teks karakter.
5. Panel Teks, berfungsi untuk menampilkan hasil pengubahan citra masukan yang telah dikenali.
6. Tombol Simpan, berfungsi untuk menyimpan teks karakter kedalam komputer dengan ekstensi txt.
7. Tombol Keluar, berfungsi untuk keluar dari program

Bentuk rancangan *interface* sistem pengenalan karakter seperti pada gambar 4.37.

Gambar 4.37 Rancangan *interface* sistem pengenalan karakter

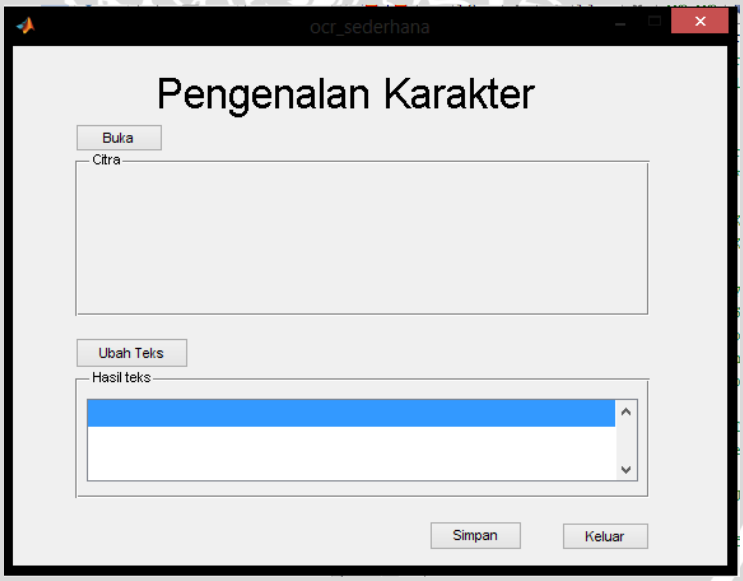
Beberapa hal yang perlu diperhatikan dalam pembuatan *user interface*

Sistem Pengenalan Karakter menggunakan bahasa pemrograman matlab dapat dilihat dalam tabel 4.1.

Tabel 4.1 Tabel Implementasi Sistem Pengenalan Karakter

No.	Style	Name/String	Tag	Visible
1.	text	Pengenalan Karakter	text1	on
2.	pushbutton	Buka	Buka	on
3.	axes	-	axes1	off
4.	pushbutton	Ubah Teks	ubah	on
5.	listbox	-	hasilTeks	on
6.	pushbutton	Simpan	simpan	on
7.	pushbutton	Keluar	keluar	on

Dari tabel 4.1 di atas diperoleh hasil implementasi dari perancangan *User Interface* Sistem Pengenalan Karakter seperti pada gambar 4.38 berikut.



Gambar 4.38 Implementasi sistem pengenalan karakter

BAB V
PENGUJIAN DAN ANALISIS

Pada bab ini dibahas mengenai pengujian dan analisis sistem dimana akan terlihat kekurangan – kekurangan pada perangkat lunak untuk selanjutnya diadakan pengembangan sistem.

Pengujian dan Analisis Terhadap Pengenalan Karakter Masukan

Pengujian sistem dalam mengenali citra karakter masukan dilakukan dengan beberapa metode berikut:

1. Pengujian dilakukan pada dua data berbeda, yaitu data training dan data yang belum pernah mendapat training sama sekali (selanjutnya disebut sebagai data non training).
2. Pengujian dilakukan secara berkelompok dan terbagi menjadi 4 kelompok, yaitu kelompok huruf besar, huruf kecil, angka, dan tanda baca.

4.6.5 Pengujian Data Training dan Non Training

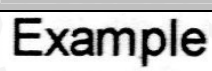
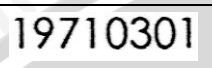
Pengujian untuk melihat seberapa baik pengenalan sistem dilakukan pada dua data berbeda, yaitu data yang telah ditraining serta data baru yang belum pernah mendapat training.

Pada pengujian dilakukan dengan memasukkan citra masukan yang merupakan sebuah kata. Tabel 5.1 dan tabel 5.2 merupakan tabel perbandingan antara dua data tersebut.

Tabel 5.1 Tabel Pengujian Data Training

No	Citra masukan	Hasil	Benar
1	paragraphs?	poragrapb/?	8
2	English	Eng1ish	6
3	University	Univecsi1y	8
4	0610630002	061O630002	9

Tabel 5.2 Tabel Pengujian Data Non Training

No	Citra masukan	Hasil	Benar
1		Exomp1e	5
2		qvicklg	4
3		Hngincering	9
4		1971O30l	6

Pada pengujian setiap kata terlihat bahwa kesalahan banyak terjadi pada huruf yang agak mirip. Adanya *template* membantu sistem untuk membedakan antara huruf besar dengan huruf kecil. Kesalahan pembacaan terjadi bila terdapat huruf yang memiliki kemiripan bentuk seperti huruf g dan y yang memiliki ekor di bawah, serta huruf I, l, dan angka 1 yang mirip sehingga hasil penentuan sistem adakalanya tertukar satu sama lain. Kesalahan pembacaan berikutnya terjadi pada angka 0, huruf o, dan O yang dikarenakan oleh pembacaan template yang terkadang berbeda.

Pada huruf yang agak mirip lainnya seperti b, d, k, h, p, dan q masih terbilang berhasil karena posisi *ascender* yaitu garis vertikal atas maupun *descender* yang merupakan garis vertikal bawah masih dapat dibedakan berdasarkan posisi.

Sedangkan pada tanda baca yang memiliki bentuk amat kecil seperti titik dan koma, kesalahan pembacaan terjadi karena pada proses *thinning* keduanya dapat memiliki bentuk yang sama, di mana sebuah titik yang tebal adakalanya pada proses thinning menjadi sebuah garis kecil seperti pada tanda baca koma.

Selain itu terlihat perbedaan antara dua tabel di atas, dimana data yang telah mendapat training dapat dikenali lebih baik daripada data non training. Tetapi hasil pengenalan untuk data baru masih terbilang cukup bagus.

4.6.6 Pengujian Secara Berkelompok

Pengujian sistem secara berkelompok adalah untuk menguji keberhasilan sistem secara umum. Pengujian ini juga dilakukan pada dua data

yang berbeda yaitu data training dan data non training tetapi dengan jumlah karakter yang sama yaitu 400 karakter.

Perincian 400 data karakter tersebut yaitu 50 data untuk masing-masing angka dan tanda baca, serta 150 data untuk masing-masing huruf besar dan huruf kecil dengan rasio setiap karakter diwakili dengan 5 *sample* sedangkan karakter vokal (a, i, u, e, o) diwakili dengan 9 *sample*, hal ini dikarenakan karakter vokal lebih banyak muncul pada suatu dokumen.

Selanjutnya dari hasil yang berhasil dikenali oleh sistem akan diambil prosentase kesuksesan rata-rata dari sistem. Hasil dari sistem secara umum ditampilkan pada tabel 5.3.

Tabel 5.3 Prosentase Rata-Rata Sistem

No	Kelompok Pengujian	Data Training yang Dikenali	Data Non Training yang Dikenali	(%)
1	Huruf Kecil (150 data)	125	120	81.6%
2	Huruf Besar (150 data)	122	121	81%
3	Tanda Baca (50 data)	40	38	78%
4	Angka (50 data)	43	38	81%
Prosentase Rata-Rata				**80.12%**

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, pengujian dan analisis yang dilakukan, maka diambil kesimpulan sebagai berikut:

1. Berdasarkan hasil pengujian sistem terhadap pengenalan citra karakter, citra yang telah ditraining lebih mudah dikenali daripada citra yang tidak melalui proses training.
2. Adanya template membantu dalam memisahkan antara karakter yang mirip maupun antara huruf besar dan huruf kecil, namun ketergantungan pada template juga berdampak kesalahan pembacaan bila satu baris berisi karakter-karakter yang tidak memiliki variasi template.
3. Sistem dapat mengenali dengan lebih baik untuk karakter yang memiliki data training cukup banyak, sedangkan banyaknya data yang dimasukkan dalam training mempengaruhi kinerja jaringan sehingga untuk dapat melatih data yang cukup banyak dan beragam diperlukan *resource* yang lebih besar pula.
4. Sistem memiliki tingkat keberhasilan pengenalan rata-rata sebesar 80.12% dengan kemampuannya mengenali 647 data dari 800 data yang terdiri atas data training dan data uji.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan sistem pengenalan karakter teks cetak ini antara lain :

1. Perlu pengambilan fitur yang lebih *general*, mungkin dengan metode lain atau fitur lain, untuk memperbaiki kinerja sistem, sehingga tidak bergantung pada 1 fitur.
2. Ditambahkan untuk penanganan untuk citra masukan yang berbentuk *italic* atau citra dari hasil *scanner* yang miring.

DAFTAR PUSTAKA

J. Gilewski, et al., 1997. "Education Aspects: Handwriting Recognition Neural Networks Fuzzy Logic". Makalah disampaikan pada *IAPR International Conference on Pattern Recognition and Information Processing*. 1997.

Nurwanto, Tri Budi. 2007. "Pengenalan Huruf Tulisan Tangan Menggunakan Logika Fuzzy Dengan Pendekatan Neural Networks Back Propagation". *Skripsi* tidak diterbitkan.Bandung : Jurusan Teknik Informatika STT Telkom, 2007.

Putra, Darma. 2010. *Pengolahan Citra Digital*. Yogyakarta : Penerbit Andi.

Pramuditya, Yoga. *Sejarah dan Kajian Tipografi.*

http://share.pdfonline.com/a674d46519cc477db51e87c3db86dd80/899EEd01.pdf.

(diakses 27 Juli 2011)

Ramza, M.T., Harry. 2007. *Teknik Pemrograman Menggunakan Matlab.* Jakarta: Grasindo.

Haralambous, Yannis. 2007. *Fonts & Encodings.* United States of America : O'Reilly Media.

LAMPIRAN

1. Daftar karakter data training yang tidak dapat dikenali sistem.

No	Kelompok Pengujian	Karakter asli	Keluaran sistem
1.	Huruf Kecil	a	q
		c	C
		c	3
		d	t
		e	a
		e	c
		g	f
		i	)
		i	:
		l	(
		l	(
		l	(
		o	,
		o	,
		o	d
		o	l
		r	l
		s	#
		s	#
		s	1
		u	U
		v	V
		v	Y
		x	X
		z	2
2.	Huruf Besar	B	R
		B	a
		B	A
		D	R
		D	G
		G	C
		G	E
		I	(
		J	j
		J	)
		K	k
		L	l
		M	E
		M	E
		O	G
		O	U
		O	G
		Q	q
		R	P
		R	S
		S	C
		S	C
		S	1
		S	3
		V	U
		V	U
		Z	2
3.	Tanda Baca	&	g
		(	)
		(	)
		?	7
		?	;
		!	)

		,	.
		,	)
		,	.
		:	;
4.	Angka	0	,
		0	L
		1	q
		2	r
		5	7
		7	c
		9	1

2. Daftar karakter data non training yang tidak dapat dikenali sistem.

No	Kelompok Pengujian	Karakter asli	Keluaran sistem
1.	Huruf Kecil	c	C
		c	3
		c	C
		d	t
		d	q
		e	c
		e	&
		e	a
		g	q
		i	9
		i	:
		i	b
		i	9
		j	J
		j	(
		l	x
		l	(
		l	(
		o	G
		o	d
		o	f
		o	d
		o	,
		s	1
		s	C
		s	C
		u	r
		u	U
		v	Y
		v	V
2.	Huruf Besar	B	P
		B	S
		B	R
		D	G
		D	R
		G	c
		G	3
		G	L
		H	h
		I	Y
		I	(
		J	j
		J	j
		J	(
		J	(
		J	j
		L	l
		L	P
		M	E
		O	G
		O	G
		O	u
		Q	q
		S	3
		S	1
		S	C
		V	y
		V	U

3.	Tanda Baca	&	U
		&	g
		(	)
		(	)
		(	)
		“	4
		?	3
		?	;
		!	3
		,	)
		,	.
		,	.
4.	Angka	0	E
		0	L
		0	O
		1	(
		1	)
		5	y
		7	C
		7	c
		8	0
		9	q
		9	1
		9	1