PENERAPAN *HOPFIELD NEURAL NETWORK* UNTUK PENGENALAN TULISAN TANGAN PADA SEBUAH CITRA DIGITAL

SKRIPSI

Oleh:
AGUNG HERDIYANTO
0710960039-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2012

UNIVERSITAS BRAWIJAYA

PENERAPAN *HOPFIELD NEURAL NETWORK* UNTUK PENGENALAN TULISAN TANGAN PADA SEBUAH CITRA DIGITAL

Skripsi

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Oleh:

AGUNG HERDIYANTO

0710960039-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2012


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENERAPAN *HOPFIELD NEURAL NETWORK* UNTUK PENGENALAN TULISAN TANGAN PADA SEBUAH CITRA DIGITAL

oleh:
Agung Herdiyanto
0710960039-96

Setelah dipertahankan di depan Majelis Penguji
Pada tanggal
Dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing I,	**Pembimbing II,**
Drs. Achmad Ridok, M.Kom	**Dany Primanita, S.T**
NIP. 196808251994031002	**NIP. 197711162005012003**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Abdul Rouf Alghofari, M.Sc
NIP. 19670907 1992031 001


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN
Saya yang bertanda tangan di bawah ini :

Nama	**:** Agung Herdiyanto
NIM	**:** 0710960039-96
Jurusan	**:** Matematika
Program Studi	**:** Ilmu Komputer
Penulis skripsi berjudul	**:** Penerapan Hopfield Neural Network untuk Pengenalan Tulisan Tangan Pada Sebuah Citra Digital

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, Juli 2012
Yang menyatakan,

Agung Herdiyanto
NIM. 0710960039-96


UNIVERSITAS BRAWIJAYA

PENERAPAN *HOPFIELD NEURAL NETWORK* UNTUK PENGENALAN TULISAN TANGAN PADA SEBUAH CITRA DIGITAL

ABSTRAK

Membaca sebuah tulisan tangan pada citra digital merupakan masalah yang sangat kompleks dan rumit dimana inti dari masalah adalah bagaimana sistem dapat mengenali tulisan tangan tersebut mewakili sebuah karakter tertentu . Tujuan dari pembacaan tulisan tangan ini adalah agar data yang terkandung dalam citra tersebut dapat diolah lebih lanjut dalam sebuah data digital. Metode *Hopfield* merupakan salah satu metode dari Jaringan Saraf Tiruan yang dapat digunakan untuk menyelesaikan permasalahan pembacaan tulisan tangan pada citra digital. Berdasarkan hasil penelitian ini tingkat akurasi untuk pembacaan karakter tunggal mencapai 97,2%, sedangkan tingkat akurasi untuk pembacaan rangkaian kata mencapai 82,6%

Kata Kunci : JST, Hopfield, Citra Digital, Tulisan Tangan


UNIVERSITAS BRAWIJAYA

HOPFIELD NEURAL NETWORK IMPLEMENTATION ON HANDWRITTEN CHARACTER RECOGNITION ON A DIGITAL IMAGE

ABSTRACT

Reading a handwritten character from a digital image could be complex and complicated. The main problem is how a system able to recognize that the handwritten character represent specific character. The purpose is that the data contained in the image can be further processed in a digital data. *Hopfield* method is one of the methods of Neural Network which can be used to solve problems reading the handwriting on a digital image. Based on the result of this study, the level of accuracy for reading single character reached 97,2%, while the level of accuracy for the reading of a phrase reaches 82,6%.

Key word : Neural Network, Hopfield, Digital Image, Handwritten

KATA PENGANTAR

Puji syukur kehadirat Allah SWT, hanya dengan rahmat dan karunia yang telah diberikan kepada penulis, sehingga penulis dapat menyelesaikan skripsi yang berjudul "Penerapan Hopfield Neural Network untuk Pengenalan Tulisan Tangan Pada Sebuah Citra Digital".

Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer Universitas Brawijaya.

Dalam menyelesaikan skripsi ini, penulis telah mendapat bantuan dan dukungan dari banyak pihak. Untuk itu, penulis ingin menyampaikan penghargaan dan ucapan terima kasih kepada:

1. Drs. Achmad Ridok, M.Kom., dan Dany Primanita, S.T., selaku dosen pembimbing yang telah dengan bijaksana dan sabar dalam membimbing penyusunan skripsi ini.
2. Dr. Abdul Rouf Al-Ghofari, M.Sc., selaku Ketua Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Brawijaya.
3. Drs. Marji, M.T., selaku Ketua Program Studi Ilmu Komputer, Jurusan Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Brawijaya.
4. Segenap Bapak dan Ibu dosen yang telah mendidik dan mengajarkan ilmunya kepada penulis selama menempuh pendidikan di Program Studi Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Brawijaya.
5. Secara khusus penulis ingin mengucapkan terima kasih kepada Ayah yang penulis banggakan dan Ibu tercinta serta adik-adik penulis yang telah banyak memberikan dukungan dan pengorbanan baik secara moril maupun materiil sehingga penulis dapat menyelesaikan studi dengan baik.
6. Eka Riana Sari, S.Kom., yang selalu menemani setiap hari dan memberikan semangat dan doa di saat suka maupun duka.
7. Semua sahabat di Program Studi Ilmu Komputer FMIPA Universitas Brawijaya yang telah banyak memberikan

bantuan, dorongan serta motivasi sehingga skripsi ini dapat terselesaikan.

8. Semua pihak lain yang membantu untuk terselesaikannya skripsi ini yang tidak dapat disebutkan satu-persatu.

Semoga penulisan laporan skripsi ini bermanfaat bagi pembaca sekalian. Penulis menyadari bahwa skripsi ini masih jauh dari kesempurnaan sehingga penulis mengharapkan kritik dan saran yang membangun dari pembaca.

Malang, Juli 2012

Penulis

DAFTAR ISI

DAFTAR GAMBAR


UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

DAFTAR SOURCECODE

BAB I
PENDAHULUAN

1.1. Latar Belakang Masalah

Pengenalan tulisan tangan (*handwriting recognition*) adalah kemampuan komputer untuk menerima *input* tulisan tangan dari sumber seperti dokumen kertas, foto, layar sentuh dan perangkat lainnya. Data yang diperoleh ini dianggap sebagai representasi statis dari tulisan tangan. Pengenalan tulisan tangan relatif sulit, dikarenakan orang yang berbeda memiliki gaya tulisan tangan yang berbeda pula.

Teknologi digital memungkinkan untuk menyimpan data atau informasi dalam bentuk elektronik. Buku dan dokumen yang semula tersimpan dalam bentuk cetakan kertas dapat diubah dalam format penyimpanan digital. Diperlukan proses *scanning* untuk mengubah data tersebut ke dalam bentuk digital. Untuk mengolah data yang diperoleh dalam bentuk teks, dibutuhkan perangkat lunak untuk mengkonversi citra ke dalam bentuk teks. Untuk itu dikembangkan suatu perangkat lunak pemrosesan citra untuk mengenali teks dalam sebuah citra dengan menggunakan *Neural Network*.

Neural Network merupakan salah satu representasi buatan dari otak manusia yang selalu mencoba untuk mensimulasikan proses pembelajaran pada otak manusia. Seperti halnya otak manusia, jaringan syaraf juga terdiri dari beberapa *neuron* yang mentransformasikan informasi yang diterima melalui sambungan keluarnya menuju ke neuron-neuron yang lain dengan nama bobot. *Neural Network* digunakan sebagai formula dasar atau rumus-rumus perhitungan dalam proses tersebut. *Neural Network* lazim digunakan sebagai sebuah metode dalam pengenalan pola dikarenakan tingkat akurasi pengenalannya sangat baik, mencapai 90% (Pratomo, 2011). Sebuah penelitian mengenai pengenalan tulisan tangan sebelumnya telah dilakukan oleh Anita Pal dan Dayashankar Singh pada tahun 2010 dalam pengenalan pola tulisan Inggris. Penelitian berhasil membuktikan bahwa metode *Neural Network* yang digunakan utuk pengenalan tulisan tangan menghasilkan tingkat akurasi yang tinggi dan waktu pelatihan yang singkat (Pal, 2010). Beberapa penelitian lain juga telah dilakukan untuk berbagai macam kasus menggunakan

Neural Network, antara lain pengenalan huruf dan angka tulisan tangandengan menggunakan jaringan syaraf buatan propagasi balik oleh AP Nugraha dan AB mutiara pada tahun 2003, serta aplikasi *Neural Network* untuk pengenalan huruf jawa oleh Nazla Nurmila dan kawan-kawan pada tahun 2007.

Neural Network mempunyai beberapa metode yang masing-masing berguna dalam bidang tertentu. Pada bidang pengenalan pola digunakan metode *Hopfield* yang dikembangkan oleh J.J. Hopfield pada tahun 1982. Pada jaringan *Hopfield*, semua neuron saling berhubungan penuh. Neuron yang satu mengeluarkan *output* dan kemudian menjadi *input* bagi semua neuron yang lain. Proses pengiriman dan penerimaan sinyal antar neuron ini secara *feedback* tertutup terus menerus sampai dicapai kondisi stabil. Metode ini memiliki kecepatan yang sepadan dengan tingkat keakurasian pengenalan polanya. Jaringan *Hopfield* mampu menyimpan berbagai pola yang diberikan kepadanya dan kemudian digunakan untuk mengenali pola baru yang akan diberikan (Matecki, 1995).

Metode *Hopfield* untuk pengenalan pola ini telah dilakukan pada berbagai riset, salah satunya pengenalan pola citra dua dimensi yang menghasilkan tingkat akurasi 98,34% (Triwijoyo, 2007). Riset mengenai perbandingan antara metode *backpropagation* dan *Hopfield* dalam pengenalan huruf pada citra digital menunjukkan bahwa *Hopfield* mempunyai tingkat keberhasilan yang lebih bagus dilihat dari parameternya (Lejap, 2007).

Berdasarkan latar belakang yang telah dipaparkan, maka judul yang diambil dalam skripsi ini adalah "**Penerapan *Hopfield Neural Network* Untuk Pengenalan Tulisan Tangan Pada Sebuah Citra Digital**".

1.2 Rumusan Masalah

Berdasarkan latar belakang pemilihan judul, maka dapat dirumuskan permasalahan sebagai berikut :

1. Bagaimana menerapkan *Hopfield Neural Network* untuk mengenali citra digital tulisan tangan sebagai sebuah karakter.

2. Berapa prosentase akurasi penerapan *Hopfield Neural Network* untuk mengenali citra digital tulisan tangan sebagai sebuah karakter .

1.3 Batasan Masalah

Karena keterbatasan waktu, pengetahuan serta sumberdaya penulis, maka ruang lingkup permasalahan dalam merancang perangkat lunak ini antara lain :

1. Data masukan berupa citra digital (*.bmp, *.jpg) dengan resolusi maksimal 500x500 *pixel*.
2. Karakter yang dilatihkan berupa alfabet dan angka (latin).
3. Tanda baca seperti titik(.) atau koma(,) dihiraukan.
4. Posisi teks pada citra tidak bertumpuk/bersambung satu sama lain.
5. Warna latar belakang citra putih.

1.4 Tujuan Penelitian

Tujuan penyusunan tugas akhir ini adalah untuk menerapkan serta membangun perangkat lunak berbasis *Hopfield Neural Network* untuk mengenali citra digital tulisan tangan sebagai sebuah karakter, serta mengevaluasi hasil keluaran perangkat lunak.

1.5 Manfaat Penelitian

Manfaat dari penyusunan tugas akhir ini yaitu perangkat lunak dapat memudahkan pengguna untuk mengubah teks dalam citra ke dalam bentuk *file* teks sehingga data dapat diolah lebih lanjut.

1.6 Metodologi Penelitian

Langkah-langkah pembuatan perangkat lunak ini antara lain :

1. Mempelajari buku dan artikel yang terkait dengan materi pada penelitian, seperti pemrosesan citra, *Hopfield Neural Network*, dan bahasa pemrograman Delphi.
2. Merancang perangkat lunak menggunakan bahasa pemrograman Delphi.
3. Merancang antar-muka perangkat lunak.
4. Menguji-coba perangkat lunak untuk mengetahui bahwa sistem berjalan dengan baik.

1.7 Sistematika Penulisan

Skripsi ini disusun berdasarkan sistematika penulisan sebagai berikut:

1. BAB I PENDAHULUAN
   Bab ini berisi latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian serta sistematika penulisan.
2. BAB II TINJAUAN PUSTAKA
   Bab ini menguraikan teori-teori yang berhubungan dengan pemrosesan citra, jaringan saraf tiruan, metode *Hopfield*.
3. BAB III METODOLOGI DAN PERANCANGAN SISTEM
   Bab ini membahas tentang perancangan sistem dan perancangan *interface* sistem.
4. BAB IV IMPLEMENTASI DAN PEMBAHASAN
   Bab ini berisi implementasi sistem dan pengujian perangkat lunak yang dibangun.
5. BAB V PENUTUP
   Bab ini berisi kesimpulan dari seluruh rangkaian penelitian serta saran untuk kemungkinan pengembangan lebih lanjut.

BAB II
TINJAUAN PUSTAKA

2.1 Jaringan Saraf Tiruan (Neural Network)

Jaringan saraf tiruan (JST) atau *neural network* adalah suatu metode komputasi yang meniru sistem jaringan biologis. Metode ini menggunakan elemen perhitungan non-linier dasar yang disebut neuron yang diorganisasikan sebagai jaringan yang saling berhubungan, sehingga mirip dengan jaringan saraf manusia. Jaringan saraf tiruan dibentuk untuk memecahkan suatu masalah tertentu seperti pengenalan pola atau klasifikasi karena proses pembelajaran (Yani, 2005).

Layaknya neuron biologi, JST juga merupakan sistem yang bersifat "*fault tolerant*" dalam 2 hal. Pertama, dapat mengenali sinyal input yang agak berbeda dari yang pernah diterima sebelumnya. Sebagai contoh, manusia sering dapat mengenali seseorang yang wajahnya pernah dilihat dari foto atau dapat mengenali seseorang yang wajahnya agak berbeda karena sudah lama tidak menjumpainya. Kedua, tetap mampu bekerja meskipun beberapa neuronnya tidak mampu bekerja dengan baik (Prasetia, 2009). Jika sebuah neuron rusak, neuron lain dapat dilatih untuk menggantikan fungsi neuron yang rusak tersebut.

Jaringan saraf tiruan belajar dari suatu contoh karena mempunyai karakteristik yang adaptif, yaitu dapat belajar dari data-data sebelumnya dan mengenal pola data yang selalu berubah. Seain itu, JST merupakan sistem yang tak terprogram, artinya semua keluaran atau kesimpulan yang ditarik oleh jaringan didasarkan pada pengalamannya selama mengikuti proses pembelajaran/pelatihan (Prasetia, 2009).

Hal yang ingin dicapai dengan melatih JST adalah untuk mencapai keseimbangan antara keseimbangan *memorisasi* dan *generalisasi*. Yang dimaksud kemampuan *memorisasi* adalah kemampuan JST untuk mengambil kembali secara sempurna sebuah pola yang telah dipelajari. Kemampuan *generalisasi* adalah kemampuan JST untuk menghasilkan respons yang bisa diterima terhadap pola-pola input yang serupa (namun tidak identik) dengan pola-pola sebelumnya yang telah dipelajari (Puspitaningrum, 2006). Hal ini sangat bermanfaat bila pada suatu hari ke dalam JST itu

diinputkan informasi baru yang belum pernah dipelajari, maka JST itu masih akan tetap dapat memberikan tanggapan yang baik, memberikan keluaran yang paling mendekati.

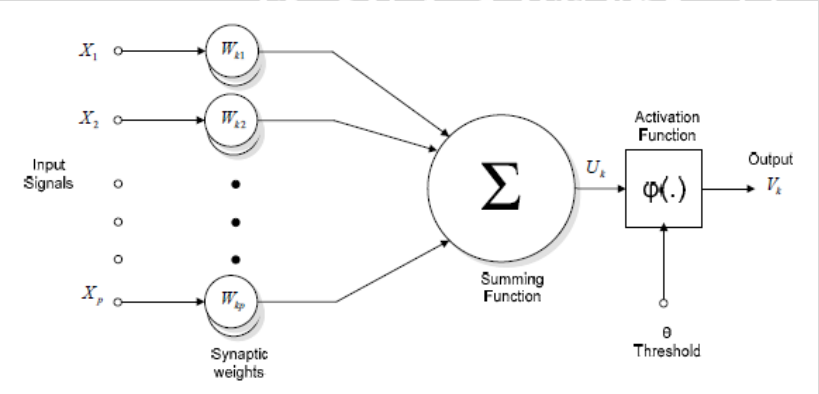
Jaringan saraf tiruan berkembang secara pesat pada beberapa tahun terakhir. Jaringan saraf tiruan telah dikembangkan sebelum adanya suatu komputer konvensional yang canggih dan terus berkembang walaupun pernah mengalami masa vakum selama beberapa tahun.

Jaringan Saraf Tiruan ditentukan oleh tiga hal, yaitu:

1. Pola hubungan antar neuron(disebut arsitektur jaringan).
2. Metode untuk menentukan bobot penghubung (disebut metode *training/learning*).
3. Fungsi aktivasi, yaitu fungsi yang digunakan untuk menentukan keluaran suatu neuron(Jek Siang, 2004).

2.1.1 Model Neuron

Satu sel syaraf terdiri dari tiga bagian, yaitu: fungsi penjumlahan (*summing function*), fungsi aktivasi (*activation function*) dan keluaran (*output*) (Prasetia, 2009). Gambar 2.1 mengilustrasikan model neuron.



Gambar 2.1 Model Neuron

Neuron buatan diatas mirip dengan sel neuron biologis. Informasi akan dikirim ke neuron dengan bobot tertentu. Input ini akan diproses oleh suatu fungsi yang akan menjumlahkan nilai-nilai bobot yang ada. Hasil penjumlahan kemudian akan dibandingkan dengan

suatu nilai ambang (*threshold*) tertentu melalui fungsi aktivasi setiap neuron. Apabila input tersebut melewati suatu nilai ambang tertentu, maka neuron tersebut akan diaktifkan, jika tidak maka neuron tidak akan diaktifkan. Apabila neuron tersebut diaktifkan, maka neuron tersebut akan mengirimkan output melalui bobot-bobot outputnya ke semua neuron yang berhubungan dengannya.

Sehingga dapat disimpulkan bahwa neuron terdiri dari 3 elemen pembentuk, yaitu (Prasetia, 2009):

1. Himpunan unit-unit yang dihubungkan dengan jalur koneksi. Jalur-jalur tersebut memiliki bobot yang berbeda-beda. Bobot yang bernilai positif akan memperkuat sinyal dan yang bernilai negatif akan memperlemah sinyal yang dibawa. Jumlah, struktur dan pola hubungan antar unit-unit tersebut akan menentukan arsitektur jaringan.
2. Suatu unit penjumlah yang akan menjumlahkan input-input sinyal yang sudah dikalikan dengan bobotnya.
3. Fungsi aktivasi yang akan menentukan apakah sinyal dari input neuron akan diteruskan ke neuron lain atau tidak.

2.1.2 Konsep Dasar Jaringan Saraf Tiruan

Setiap pola-pola informasi input dan output yang diberikan ke dalam JST diproses dalam neuron. Neuron-neuron tersebut terkumpul di dalam lapisan-lapisan yang disebut *neuron layers*. Lapisan-lapisan penyusun JST tersebut dapat dibagi menjadi 3, yaitu (Prasetia, 2009):

1. Lapisan input

   Unit-unit di dalam lapisan input disebut unit-unit input. Unit-unit input tersebut menerima pola inputan data dari luar yang menggambarkan suatu permasalahan.
2. Lapisan tersembunyi

   Unit-unit di dalam lapisan tersembunyi disebut unit-unit tersembunyi. Dimana outputnya tidak dapat langsung diamati.
3. Lapisan output

   Unit-unit di dalam lapisan output disebut unit-unit output. Output dari lapisan ini merupakan solusi JST terhadap suatu permasalahan.

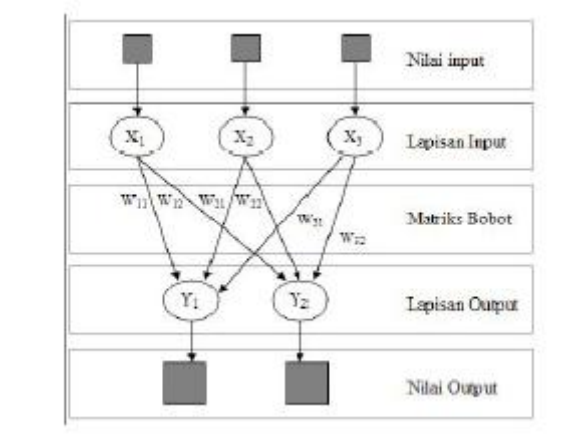
2.1.3 Arsitektur Jaringan

JST memiliki beberapa arsitektur jaringan yang sering digunakan dalam berbagai aplikasi. Arsitektur JST tersebut, antara lain (Kusumadewi, 2003):

1. Jaringan layer tunggal (*Single layer network*)

   Jaringan dengan lapisan tunggal terdiri dari 1 *layer* input dan 1 *layer* output.setiap neuron/unit yang terdapat di dalam lapisan input selalu terhubung dengan setiap neuron yang terdapat pada lapisan output. Jaringan ini hanya menerima input kemudian secara langsung akan mengolahnya menjadi output tanpa harus melalui lapisan tersembunyi. Ilustrasi jaringan layer tunggal seperti digambarkan pada gambar 2.2.

   Contoh algoritma JST yang menggunakan metode ini, yaitu *ADALINE, Hopfield* dan *Perceptron.*

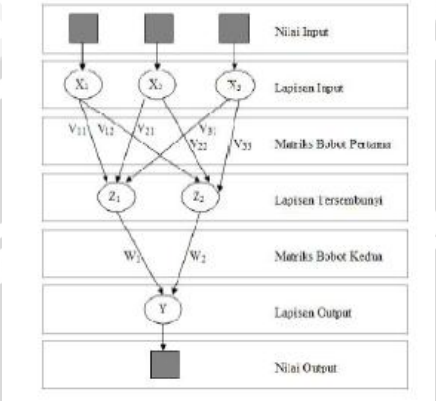


Gambar 2.2 Arsitektur Layer Tunggal

2. Jaringan layer jamak (*multi layer network*)

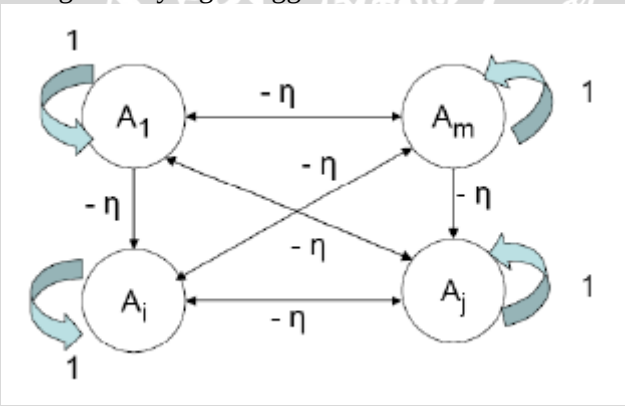
   Jaringan dengan lapisan jamak memiliki ciri khas tertentu yaitu memiliki 3 jenis lapisan, yaitu lapisan input, lapisan output dan juga lapisan tersembunyi. Jaringan dengan banyak lapisan ini dapat menyelesaikan permasalahan yang lebih kompleks dibandingkan jaringan dengan lapisan tunggal. Namun, proses pelatihan sering membutuhkan waktu yang cenderung lama. Ilustrasi Jaringan layer jamak seperti digambarkan pada gambar 2.3.

Contoh algoritma JST yang menggunakan metode ini, yaitu *MADALINE*, *Backpropagation* dan *Neocognitron*.



Gambar 2.3 Arsitektur Layer Jamak

3. Jaringan dengan lapisan kompetitif (*competitive layer network*)
   Pada jaringan ini sekumpulan neuron bersaing untuk mendapatkan hak menjadi aktif. Ilustrasi jaringan dengan lapisan kompetitif seperti digambarkan pada gambar 2.4.
   Contoh algoritma yang menggunakan metode ini adalah *LVQ*.



Gambar 2.4 Arsitektur Layer Kompetitif

2.1.4 Metode Pelatihan/Pembelajaran

Cara berlangsungnya pembelajaran atau pelatihan JST dikelompokkan menjadi 3, yaitu (Puspitaningrum, 2006):

1. *Supervised learning* (Pembelajaran terawasi)

   Pada metode ini, setiap pola yang diberikan ke dalam JST telah diketahui outputnya. Selisih antara pola output aktual (output yang dihasilkan) dengan pola output yang dikehendaki (output target) yang disebut *error* digunakan untuk mengoreksi bobot JST sehingga JST mampu menghasilkan output sedekat mungkin dengan pola target yang telah diketahui oleh JST.

   Contoh algoritma JST yang menggunakan metode ini adalah *Hebbian, Perceptron, ADALINE, Boltzman, Hopfield* dan *Backpropagation.*

2. *Unsupervised learning* (Pembelajaran tak terawasi)

   Pada metode ini, tidak memerlukan target output. Pada metode ini tidak dapat ditentukan hasil seperti apakah yang diharapkan selama proses pembelajaran. Selama proses pembelajaran, nilai bobot disusun dalam suatu range tertentu tergantung pada nilai input yang diberikan. Tujuan pembelajaran ini adalah mengelompokkan unit-unit yang hampir sama dalam suatu area tertentu. Pembelajaran ini biasanya sangat cocok untuk klasifikasi pola.

   Contoh algoritma JST yang menggunakan metode ini adalah *Competitive, Hebbian, Kohonen, LVQ* dan *Neocognitron.*

3. *Hybrid learning* (Pembelajaran hibrida)

   Merupakan kombinasi dari metode pembelajaran *supervised learning* dan *unsupervised learning*. Sebagian dari bobot-bobotnya ditentukan melalui pembelajaran terawasi dan sebagian lainnya melalui pembelajaran tak terawasi.

   Contoh algoritma JST yang menggunakan metode ini adalah algoritma *RBF*.

2.1.5 Fungsi Aktivasi

Dalam JST, fungsi aktivasi digunakan untuk menentukan keluaran suatu neuron. Argumen suatu fungsi aktivasi adalah net masukan (kombinasi linier masukan dan bobotnya) (Jek Siang, 2004). Fungsi aktivasi pada *Hopfield* akan mengubah N input (x_1, x_2, x_3, ... , x_n) yang masing-masing memiliki bobot *W* dengan

$$\boldsymbol{y\_in_i} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{W_{ji} U_i} \qquad (2.1)$$

Keterangan:
y_in_i : sinyal output untuk aktivasi

Fungsi aktivasi yang digunakan pada penelitian ini adalah (Jek Siang, 2004) fungsi *threshold* (batas ambang)

$$f(x) = \begin{cases} \mathbf{1\ untuk}\ x > a \\ \mathbf{0\ untuk}\ x \leq a \end{cases} \quad (2.2)$$

Keterangan :
a : nilai ambang (*threshold*)

Fungsi *threshold* (2.1) merupakan fungsi *threshold biner*. Untuk kasus bilangan bipolar, maka angka 0 diganti dengan angka -1, sehingga persamaan (2.1) diubah menjadi:

$$f(x) = \begin{cases} \mathbf{1\ untuk}\ x > a \\ \mathbf{-1\ untuk}\ x \leq a \end{cases} \quad (2.3)$$

Adakalanya dalam JST ditambahkan suatu unit masukan yang nilainya selalu 1. Unit tersebut dikenal dengan bias. Bias dapat dipandang sebagai sebuah input yang nilainya selalu 1. Bias berfungsi untuk mengubah *threshold* menjadi 0.

2.1.6 Fungsi Transfer

Karakter dari Jaringan Syaraf Tiruan tergantung atas bobot dan fungsi input-output(fungsi transfer) yang mempunyai ciri tertentu untuk setiap unit. Fungsi ini terdiri dari 3 kategori yaitu (Yani,2005):

1. *Linear units.*
   Aktifitas output sebanding dengan jumlah bobot output.
2. *Threshold units.*
   Output diatur oleh satu dari dua tingkatan tergantung dari apakah jumlah input lebih besar atau lebih kecil dari nilai ambang.
3. *Sigmoid units.*
   Output terus menerus berubah tetapi tidak berbentuk linear. Unit ini mengandung kesamaan yang lebih besar dari sel syaraf sebenarnya dibandingkan dengan *linear* dan *threshold* unit, namun ketiganya harus dipertimbangkan dengan perkiraan kasar.

Untuk membuat Jaringan Syaraf Tiruan diperlukan beberapa kerja khusus. Harus dipilih bagaimana unit-unit dihubungkan antara satu dengan yang lain dan harus mengatur bobot dari hubungan tersebut

secara tepat. Hubungan tersebut menentukan apakah mungkin suatu unit mempengaruhi unit yang lain. Bobot menentukan kekuatan dari pengaruh tersebut.

Dapat dilakukan pembelajaran terhadap 3 lapisan pada JST untuk melakukan kerja khusus dengan menggunakan prosedur dibawah ini (Yani, 2005):

1. Memperkenalkan JST dengan contoh pembelajaran yangterdiri dari sebuah pola dari aktifitas untuk unit-unit input bersama dengan pola yang diharapkan dari aktifitas untuk unit-unit output.
2. Menentukan seberapa dekat output sebenarnya dari JST sesuai dengan output yang diharapkan.
3. Mengubah bobot setiap hubungan agar JST menghasilkan suatu perkiraan yang lebih baik dari output yang diharapkan.

Cara lain yang sering digunakan untuk menghitung EW adalah dengan menggunakan algoritma *Backpropagation*. Saat ini merupakan metode yang penting untuk pembelajaran JST. Metode ini dikembangkan secara mandiri oleh 2 tim yaitu Fogelman-Soulie, Gallinari dan Le Cun dari Prancis dan Rumelhart,Hinton dan Williams dari Amerika (Yani, 2005).

2.2 Metode *Hopfield*

Metode ini dikembangkan oleh John Hopfield pada tahun 1980. John Hopfield menggambarkan suatu *associative memory* yang dapat diterapkan dan kemudian mendemonstrasikan masalah optimasi dan klasifikasi yang dapat diselesaikan oleh jaringannya. Jaringan *Hopfield* biner mempunyai suatu lapisan pengolah. Setiap unit pengolah mempunyai sebuah nilai aktifitas atau kondisi *(state)* yang bersifat biner (Hermawan, 2006). Jaringan *Hopfield* bisa bekerja dengan kondisi 0 dan 1. Jaringan juga dapat bekerja jika digunakan nilai 1 dan -1, hanya saja diperlukan sedikit perubahan dalam persamaannya.

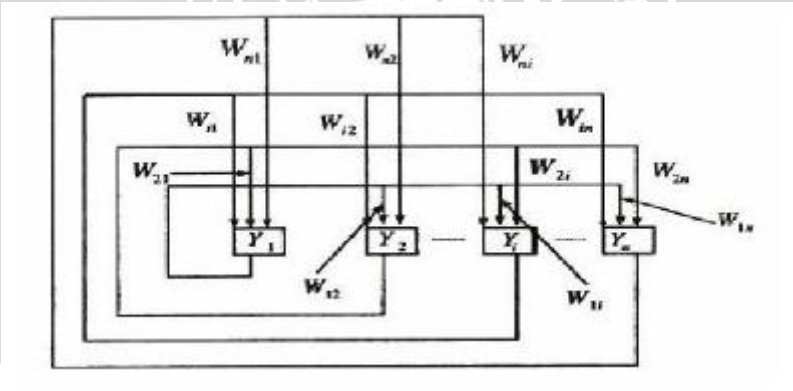
Unit-unit pengolah dalam jaringan *Hopfield* adalah terhubung penuh *(fully-connected*) yaitu setiap unit terhubung ke tiap-tiap unit yang lain. Hubungan-hubungan tersebut adalah hubungan langsung dan setiap pasang unit pengolah mempunyai hubungan dalam dua arah. Topologi hubungan ini mempunyai jaringan bersifat *recursive* karena keluaran dari setiap unit memberi masukan ke unit yang lain. Jaringan ini memiliki bobot-bobot yang simetris. Pada jaringan

Hopfield setiap unit tidak memiliki hubungan dengan dirinya sendiri, secara matematik hal ini memenuhi$W_{ij}=W_{ji}$(di mana i = baris dan j = kolom) untuk $i \neq j$ dan $W_{ij}= 0$ untuk $i = j$ (Puspitaningrum, 2006). Gambar 2.5 menunjukkan contoh matrik bobot jaringan *Hopfield.*

$$W_{ji} = \begin{bmatrix} 0 & w_{12} & w_{13} & w_{14} & w_{15} & w_{116} \\ w_{21} & 0 & w_{23} & w_{24} & w_{25} & w_{26} \\ w_{31} & w_{32} & 0 & w_{34} & w_{35} & w_{36} \\ w_{41} & w_{42} & w_{43} & 0 & w_{44} & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & 0 & w_{55} \\ w_{61} & w_{62} & w_{63} & w_{64} & w_{65} & 0 \end{bmatrix}$$

Gambar 2.5 Matrik Bobot Jaringan *Hopfield* sebagai vektor W

Perhatikan bahwa bobot–bobot yang terletak pada diagonal utamanya adalah nol (0) yang menunjukkan bahwa neuron-neuron pada jaringan *Hopfield* tidak memiliki hubungan dengan dirinya sendiri ($W_{ij}= 0$; i=j). Sementara itu kesimetrisan vektor bobot berarti berlakunya $W_{ij}=W_{ji}$di mana $i \neq j$, sehingga$W_{12}= W_{21}$, $W_{13}= W_{31}$, $W_{23}= W_{32}$..., dan seterusnya.



Gambar 2.6 Arsitektur Jaringan *Hopfield*

Dari gambar 2.6 di atas, dapat dijelaskan bahwa output setiap simpul diumpanbalikkan ke input dari simpul–simpul lainnya

melalui bobot koneksi W_{ij} yang tetap. Nilai W_{ij} mula–mula diinisialisasi menggunakan algoritma *Hopfield* untuk sebuah pola yang masuk. Pola yang tidak dikenal dimasukkan ke dalam jaringan pada waktu kondisi nol. Kemudian jaringan akan beriterasi sampai output yang dihasilkan konvergen. Konvergen adalah pola output yang tidak berubah sampai iterasi selesai. Pola yang dinyatakan oleh simpul–simpul output setelah jaringan konvergen adalah output jaringan syaraf tiruan.

2.2.1 Algoritma Pembelajaran JST *Hopfield*

Ada beberapa versi dari algoritma *Hopfield* . Pada penelitian ini menggunakan input vektor yang berupa angka bipolar. Untuk menyimpan sekumpulan angka bipolar digunakan notasi:

U_i ,i = (1,2,3,...,n) (2.4)

Keterangan:

n : jumlah pola yang dilatihkan

U_i : elemen i dari vektor U.

Sedangkan untuk menyimpan bobot matriks W={W_{ji}}, dengan notasi:

$$\boldsymbol{W_{ji} = \sum_{s=1}^{n} U_{sj} U_{si}} \quad (2.5)$$

Jika $i \neq j$ dan W_{ji}= 0 jika $i = j$

Keterangan :

W_{ij} : matrik bobot koneksi dari unit j ke unit i

n : jumlah pola yang dilatihkan

U_{si} : elemen i dari vektor U yang memiliki nilai +1 dan -1

U_{sj} : elemen j dari vektor U yang memiliki nilai +1 dan -1

Adapun algoritma pembelajaran *Hopfield* dari kedua penjelasan di atas dan apabila diterapkan pada pola angka yang berjumlah sepuluh buah yang berupa angka nol (0) sampai sembilan (9) adalah sebagai berikut (Puspitaningrum, 2006):

1. Menciptakan pola vektor U, yang merupakan himpunan n simpul untuk pola yang ke–i, seperti pada persamaan 2.3.
2. Menciptakan matriks bobot koneksi sebagai tempat pola contoh yaitu angka 0 sampai pola angka 9 yang disimpan dalam suatu konstanta, menggunakan persamaan 2.5.

3. Mengerjakan langkah 4-6 selama pola belum konvergen input≠output.
4. Untuk tiap vektor U_i, i = (1,2,3,...,n) , hitung nilai y_in_i dengan persamaan 2.1.
5. Hitung $Outvec_1 = f(y\_in_i)$ di mana f adalah fungsi ambang (*threshold function)*

$$f(t) = \begin{cases} \mathbf{1 \text{ jika } } \boldsymbol{y\_in > \theta} \\ \mathbf{0 \text{ jika } } \boldsymbol{y\_in \leq \theta} \end{cases}$$

(2.6)

Di mana *threshold* θ biasanya sama dengan 0.
6. Update *neuron input* jaringan dengan komponen $Outvec_i$
7. Test konvergensi.

2.3 Citra Digital

Citra adalah gambar pada bidang dua dimensi. Dalam tinjauan matematis, citra merupakan fungsi kontinu dari intensitas cahaya pada bidang dua dimensi (Sada, 2004). Ketika sumber cahaya menerangi objek, objek memantulkan kembali sebagian cahaya tersebut. Pantulan ini ditangkap oleh alat-alat pengindera optik, misalnya mata manusia, kamera, scanner dan sebagainya. Bayangan objek tersebut akan terekam sesuai intensitas pantulan cahaya. Ketika alat optik yang merekam pantulan cahaya itu merupakan mesin digital, misalnya kamera digital, maka citra yang dihasilkan merupakan citra digital. Pada citra digital, kontinuitas intensitas cahaya dikuantisasi sesuai resolusi alat perekam.

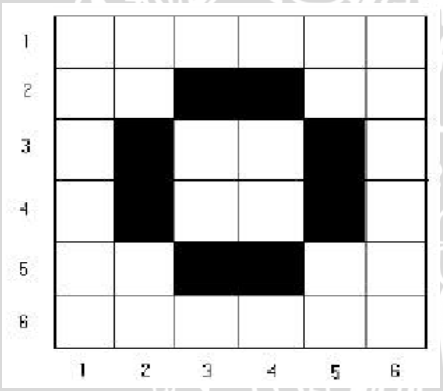
2.3.1 Citra Dua Dimensi

Citra dua dimensi yang berwarna terbentuk dari suatu segi atau bentuk dari *array* dengan pola berbentuk titik-titik yang dikenal dengan *picture element* atau *pixel*. Masing-masing citra mempunyai tingkat ke dalaman warna yang berbeda (Pratomo, 2011). Apabila *pixel* ini mempunyai warna maka disebut citra berwarna. Dengan menggunakan komposisi warna merah(*Red*), hijau(*Green*) dan biru(*Blue*) yang disebut RGB.

Pada umumnya suatu citra berbentuk persegi meskipun terlihat bulat, karena bagian lainnya digantikan dengan warna yang lain. Citra persegi tersebut tersusun dari *array pixel* sebanyak M baris dan

N kolom, M dan N ini merupakan besar resolusi dari suatu citra, misal M=256 dan N=256 maka resolusi citra tersebut adalah 256x256 dengan total *pixel* adalah 256x256=65.535. Untuk citra *grayscale* mempunyai tingkat abu-abu atau *gray* mulai 0 (intensitas cahaya nol) sampai 255 (intensitas penuh), yang merupakan batas minimal dari warna hitam dan putih untuk gambar fotografi (Pratomo, 2011). Pada komputer tiap-tiap *pixel* dihitung sebesar 1 *byte* sehingga untuk 65.535 *pixel* membutuhkan ruang penyimpanan sebesar 65.535 *byte.*

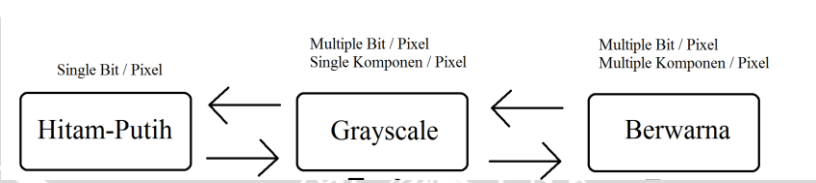
Citra berwarna memerlukan ruang penyimpanan lebih banyak daripada citra *grayscale* (Pratomo, 2011). Citra berwarna disusun dari warna RGB untuk tiap *pixel*nya. Sehingga tiap *pixel* memerlukan 3 *byte.* Pada resolusi 256x256, ruang penyimpanan yang diperlukan adalah 256x256x3=196.608 *byte.* Sehingga dapat disimpulkan bahwa lebih banyak warna dan resolusi akan menambah ruang penyimpanan dan waktu yang diperlukan untuk pemrosesan. Gambar 2.7 merupakan contoh bagaimana *pixel* terdapat dalam suatu gambar.



Gambar 2.7 Citra dengan resolusi 6x6

Citra yang berwarna dapat diturunkan menjadi *grayscale*, dan *grayscale* dapat diturunkan menjadi hitam dan putih. Hal ini dinamakan dengan konversi warna dari konsepsual data (Pratomo, 2011). Pada saat kompresi warna akan terjadi perbedaan akurasi warna yaitu pada nilai *bit* per *pixel*nya. Suatu *pixel* bisa terdapat satu

atau lebih komponen warna dan suatu komponen mempunyai *bit* yang berbeda untuk tiap nilainya. Selama konversi sejumlah *bit* pada *pixel* dikurangi sehingga ada data yang hilang. Hal ini mengakibatkan sulitnya mendapatkan citra yang bagus saat *bit*nya dikurangi. Untuk konversi yang berkebalikan, hanya menduplikasikan (*copy*) datanya ke dalam format data yang ditentukan, tetapi tidak dapat menambah keakurasian data citra. Pada gambar 2.8 memperlihatkan skema dari konversi mulai dari citra berwarna sampai citra hitam putih.



Gambar 2.8 Konversi citra berwarna, *grayscale* dan hitam-putih

2.3.2 Grayscaling

Grayscale merupakan suatu format citra pada sistem *digital image.* Pada suatu citra, format RGB (*Red, Green, Blue*) mewakili citra pada sistem digital untuk setiap *pixel,* dimana masing-masing komponen warna diwakili dengan satu *byte.* Karena nilai variasi yang ada pada masing-masing komponen RGB adalah 0-255, maka total nilai variasi-variasi yang dihasilkan untuk sistem warna ini adalah 256x256x256 atau 16.777.216 (Kristanto, 2002).

Kalkulasi pemrosesan gambar dengan sistem RGB ini sangatlah memboroskan memori dan juga waktu. Untuk itu diperlukan reduksi warna pada pemrosesan citra, terutama pada *object classification.* Format RGB sendiri tidaklah memberikan respon yang baik, sehingga digunakan sistem *grayscale.*

Sistem citra *grayscale* memerlukan satu *byte* (delapan *bit*) untuk penyimpanan data, mempunyai kemungkinan warna 0 (hitam) sampai 255 (putih). Dalam hal ini nilai variasi RGB untuk sistem citra *grayscale* adalah sama. Konversi dari sistem warna RGB menjadi *grayscale* ini ada beberapa macam, yang pertama adalah merata-rata setiap komponen warna RGB.

$$\boldsymbol{Grayscale} = \frac{\boldsymbol{Red+Green+Blue}}{\boldsymbol{3}} \qquad (2.7)$$

Yang kedua adalah menggunakan sistem YUV (sistem warna pada NTSC) yaitu dengan cara mengambil komponen Y (iluminasi). Komponen Y sendiri dapat diperoleh dari sistem warna RGB dengan konversi. Cara inilah yang paling sering digunakan untuk konversi sistem warna ke sistem *grayscale* (Kristanto, 2002). Sedangkan yang digunakan pada penelitian kali ini menggunakan persamaan

*Grayscale = (0.299*R + 0.587*G + 0.114*B)* (2.8)

2.3.3 Thresholding

Threshold adalah operasi citra *bi-level*, yaitu operasi yang mengubah *grayscale* menjadi 0 (putih) atau 1 (hitam) (Kristanto, 2002). *Threshold* akan bekerja dengan sempurna pada citra yang telah di*grayscale* terlebih dahulu. Akan ada informasi yang hilang dengan *threshold* ini, tetapi *threshold* ini juga memberi keuntungan pemrosesan gambar yang paling mudah.

Threshold didefinisikan sebagai

$$\boldsymbol{g(x,y)} = \begin{cases} \mathbf{1\ untuk}\ \boldsymbol{f(x,y) > T} \\ \mathbf{0\ untuk}\ \boldsymbol{f(x,y) \leq T} \end{cases}$$

(2.9)

dimana T adalah batas *threshold*nya. T sendiri bisa merupakan konstanta apapun ataupun variabel dimana nilainya diperoleh dari nilai *grayscale.*(G.W Awcock & R. Thomas, 1996)

Operasi *threshold* lebih baik dilakukan dengan melakukan segmentasi pada citra. Tiap-tiap citra dibagi menjadi beberapa bagian, dan kemudian setiap bagian tersebut dilakukan operasi *threshold* dengan nilai T yang berbeda-beda. Proses *threshold* ada dua macam, yaitu:

1. *Single threshold*

Single threshold adalah proses mengganti suatu warna berdasarkan ketentuan nilai variabel T (Kristanto, 2002). Nilai variabel disini mempunyai range 0-255. Nilai variabel akan menentukan batas warna yang akan diubah, yakni nilai diatas dan dibawah variabel. Nilai diatas maupun dibawahnya akan berubah sesuai dengan warna yang diinginkan. Contoh aplikasi *single threshold* seperti gambar 2.9 dengan nilai variabel T adalah 200:

Gambar 2.9 *Single threshold*

2. *Dual threshold*

Dual threshold mempunyai dua buah variabel, yaitu nilai minimum dan maksimum. Warna akan berubah jika nilai variabel ada antara nilai minimum dan maksimum. Diluar nilai itu juga akan berubah sesuai dengan warna yang diinginkan (Kristanto, 2002).

2.3.4 Segmentationing

Segmentation adalah sebuah proses untuk memindahkan obyek dari *background*, sehingga obyek tersebut dapat digunakan untuk keperluan yang lain (Wijaya, 2006). Seiring dengan berkembangnya teknologi pada aplikasi yang memproses sebuah obyek seperti rekonstruksi obyek tiga dimensi, pengenalan benda, pengenalan tulisan, deteksi wajah, pengkodean obyek dan lain-lain maka proses *segmentation* menjadi semakin diperlukan. Hasil dari *segmentation* juga harus semakin akurat karena ketidak akuratan hasil segmentasi akan mempengaruhi pula hasil proses selanjutnya. Banyak metode yang dapat digunakan pada proses *segmentation* seperti dengan menggunakan *threshold* baik *adaptive threshold* ataupun tidak, pendeteksian tepi obyek menggunakan *filter* Sobel, Prewitt ataupun yang lain. Secara umum proses *segmentation* tersebut terbagi menjadi tiga bagian yaitu berdasar klasifikasi, berdasar tepi dan berdasar daerah (Wijaya, 2006).

1. *Classification-based. Segmentation* berdasarkan kesamaan suatu ukuran dari nilai *pixel*. Salah satu cara paling mudah adalah *thresholding.* Pada *thresholding* global, *segmentation* berdasarkan pada sejenis histogram, sedangkan pada *thresholding* lokal, *segmentation* dilakukan berdasarkan posisi pada citra. Citra dibagi menjadi bagian-bagian yang saling melengkapi, jadi sifatnya dinamis.

2. *Edge-based.* Proses *segmentation* untuk mendapatkan garis yang ada pada citra dengan anggapan bahwa garis tersebut merupakan tepi

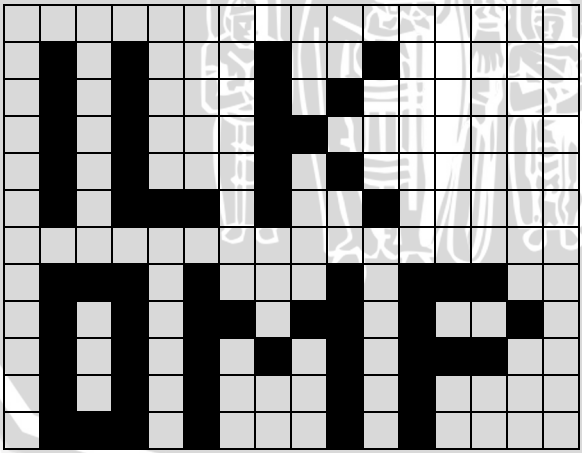
dari obyek yang memisahkan obyek yang satu dengan obyek yang lain atau antara obyek dengan *background.*

3. *Region-based. Segmentation* dilakukan berdasarkan kumpulan *pixel* yang memiliki kesamaan (tekstur, warna atau tingkat warna abu-abu) dimulai dari suatu titik ke titik-titik lain yang ada disekitarnya.

Proses segmentasi yang dilakukan pada penelitian kali ini terbagi dalam dua proses segmentasi, yaitu:

1. Segmentasi Baris

Segmentasi baris merupakan proses awal dari modul proses segmentasi karakter. Proses yang dilakukan pada tahap segmentasi baris ini adalah dengan memetakan keseluruhan gambar yang berisi rangkaian karakter pada sumbu-*y* untuk mendapatkan posisi dan tinggi dari tiap-tiap baris (gambar tersebut diasumsikan gambar yang hanya berisi rangkaian karakter). Hal ini dilakukan dengan menghitung jumlah *pixel* yang bernilai 1 (pixel hitam) pada sumbu-*y*. Selanjutnya dari hasil perhitungan *pixel* terhadap sumbu-*y* tersebut akan diperoleh posisi baris (posisi awal dan posisi akhirnya) dan tinggi setiap baris rangkaian karakter yang ditentukan berdasarkan jumlah *pixel* yang muncul pada setiap baris *pixelnya*. Berdasarkan contoh pada gambar 2.10, terdapat 2 baris, dengan baris ke-1 posisi awal pada *pixel* ke-2, dan posisi akhir pada *pixel* ke-6. Sedangkan baris ke-2 posisi awalnya pada *pixel* ke-8 dengan posisi akhir pada *pixel* ke-12.



Gambar 2.10 Ilustrasi Proses Segmentasi

2. Segmentasi Karakter

Setiap baris hasil dari tahap segmentasi baris sebelumnya dipetakan pada sumbu- *x* lalu dihitung jumlah *pixel* bernilai satu (1)(*pixel* hitam). Seperti pada tahap segmentasi baris, hasil pemetaan pada sumbu-*x* ini akan mendapatkan jumlah karakter per-baris, posisi awal dan posisi akhir setiap karakter terhadap sumbu-*x*. Pada sistem yang akan dibangun ini menggunakan acuan minimal spasi ≥1. Dari proses ini maka akan didapatkan jumlah karakter tiap baris, posisi awal dan akhir tiap-tiap karakter. Berdasarkan contoh pada gambar 2.10, posisi awal pada karakter pertama di baris pertama adalah pada *pixel* ke-2, sedangkan posisi akhirnya adalah pada *pixel* ke-2. Posisi awal karakter pertama di baris kedua adalah pada *pixel* ke-2, dengan posisi akhirnya pada *pixel* ke-4.

2.3.5 Normalisasi Citra

Dalam pemrosesan citra, normalisasi adalah proses yang mengubah nilai batas-batas *pixel*. Tujuan dari perubahan nilai batas ini adalah untuk menyamakan jumlah neuron input tiap-tiap karakter yang dilatih/dikenali. Normalisasi akan mengubah citra dalam kisaran (Min, Max) ke dalam kisaran (MinBaru, MaxBaru) (Gonzales, 2007). Misalnya, jika rentang batas citra adalah 50 sampai 180 *pixel* dan kisaran yang diinginkan adalah 0 sampai 255 maka proses akan mengurangi sebanyak 50 *pixel* dari setiap *pixel*, membuat kisaran 0 hingga 130. Kemudian setiap intensitas *pixel* dikalikan dengan 255/130, sehingga membuat rentang 0 sampai 255.

2.4 Akurasi

Akurasi merupakan seberapa dekat suatu angka hasil pengukuran terhadap angka sebenarnya (*true value atau reference value*). Dalam penelitian ini akurasi diagnosis dihitung dari jumlah diagnosis yang tepat dibagi dengan jumlah data. Tingkat akurasi diperoleh dengan perhitungan sesuai dengan persamaan 2.10 (Pratomo, 2011).

$$\boldsymbol{Tingkat\ akurasi} = \frac{\sum \boldsymbol{data\ uji\ benar}}{\sum \boldsymbol{total\ data\ uji}} \qquad (2.10)$$

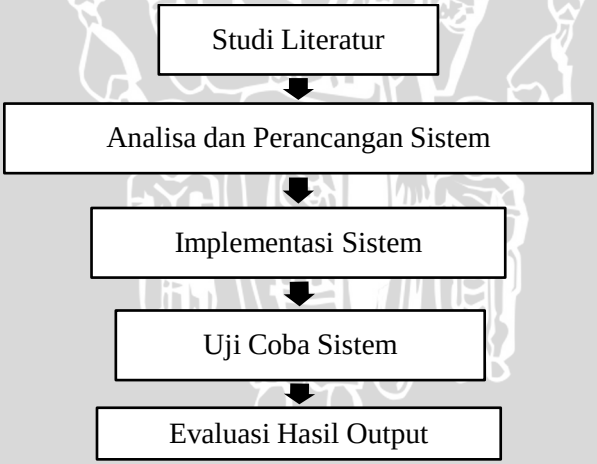
$$\boldsymbol{Akurasi}\ (\%) = \frac{\sum \boldsymbol{data\ uji\ benar}}{\sum \boldsymbol{total\ data\ uji}} \boldsymbol{x}\ \mathbf{100}\% \qquad (2.11)$$

BAB III
METODOLOGI DAN PERANCANGAN

Pada bab metodologi dan perancangan ini akan dibahas rancangan yang digunakan dan langkah pengujian yang dilakukan dalam pembuatan sistem pengenalan tulisan tangan menggunakan jaringan saraf tiruan. Langkah-langkah yang dijalankan dalam penelitian ini, yaitu:

1. Mempelajari literatur yang berhubungan dengan pemrosesan citra dan jaringan saraf tiruan.
2. Menganalisa dan melakukan perancangan system dengan metode jaringan saraf tiruan *Hopfield*.
3. Membangun perangkat lunak berdasarkan analisis dan perancangan yang telah dilakukan (implementasi).
4. Melakukan uji coba terhadap perangkat lunak.
5. Mengevaluasi output yang dihasilkan oleh hasil ujicoba sistem.

Adapun langkah-langkah penelitian dapat digambarkan dalam bentuk diagram alir yang ditunjukkan pada gambar 3.1.



Gambar 3.1 Diagram Alir Penelitian

3.1 Studi Literatur

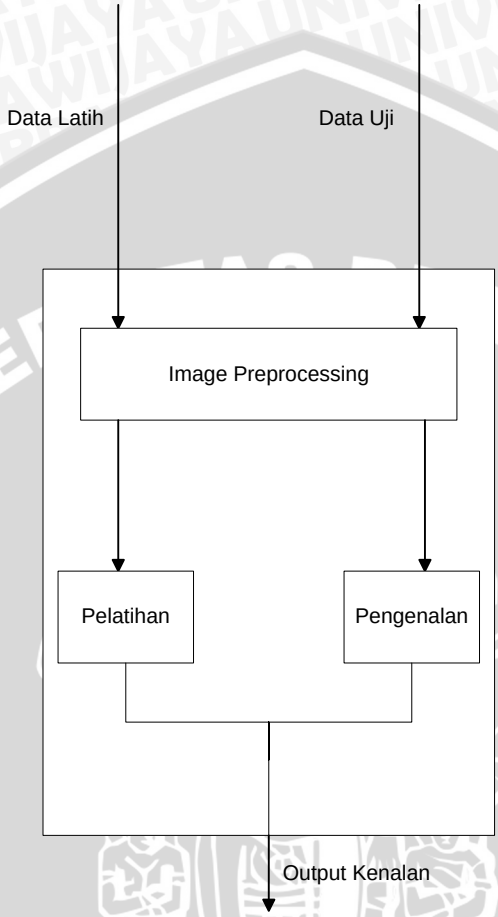
Dalam penelitian ini dibutuhkan studi literature untuk merealisasikan tujuan dan penyelesaian masalah. Teori-teori mengenai pemrosesan citra, jaringan saraf tiruan serta metode *Hopfield Neural Network* digunakan sebagai dasar penelitian yang diperoleh dari buku, jurnal dan browsing dari internet. Data dan teori yang diperoleh kemudian dijadikan acuan untuk diimplementasikan ke dalam program.

3.2 Deskripsi Umum Sistem

Secara umum sistem yang dibangun adalah suatu perangkat lunak untuk mengenali tulisan tangan dalam sebuah inputan berupa citra menggunakan jaringan saraf tiruan. Sistem ini bertujuan untuk mempermudah pengolahan data berupa tulisan tangan dalam sebuah citra.

3.3 Perancangan Sistem

Pada perancangan sistem ini, diperlukan beberapa tahapan yang perlu dilalui agar dapat membentuk system pengenalan tulisan tangan menggunakan jaringan saraf tiruan.Dalam proses klasifikasi ini memerlukan input berupa citra yang berisi tulisan tangan. Citra inputan tersebut akan mengalami serangkaian manipulasi citra sehingga mengubah citra tersebut menjadi serangkaian nilai-nilai yang berguna untuk proses utama berikutnya, yaitu proses pelatihan dan proses pengenalan. *Output* dari proses pelatihan adalah sekumpulan nilai bobot jaringan dan vektor pola stabil yang disimpan ke dalam sebuah file pelatihan. File pelatihan tersebut dapat dibuka kembali bila akan di*update* ataupun digunakan dalam proses pengenalan. Sedangkan *output* dari proses pengenalan adalah konversi dari citra yang berisi angka-angka *bipolar* menjadi huruf-huruf atau angka yang dapat disimpan ke dalam *file* teks. Alur system ini digambarkan dalam bentuk *block diagram* seperti pada gambar 3.2

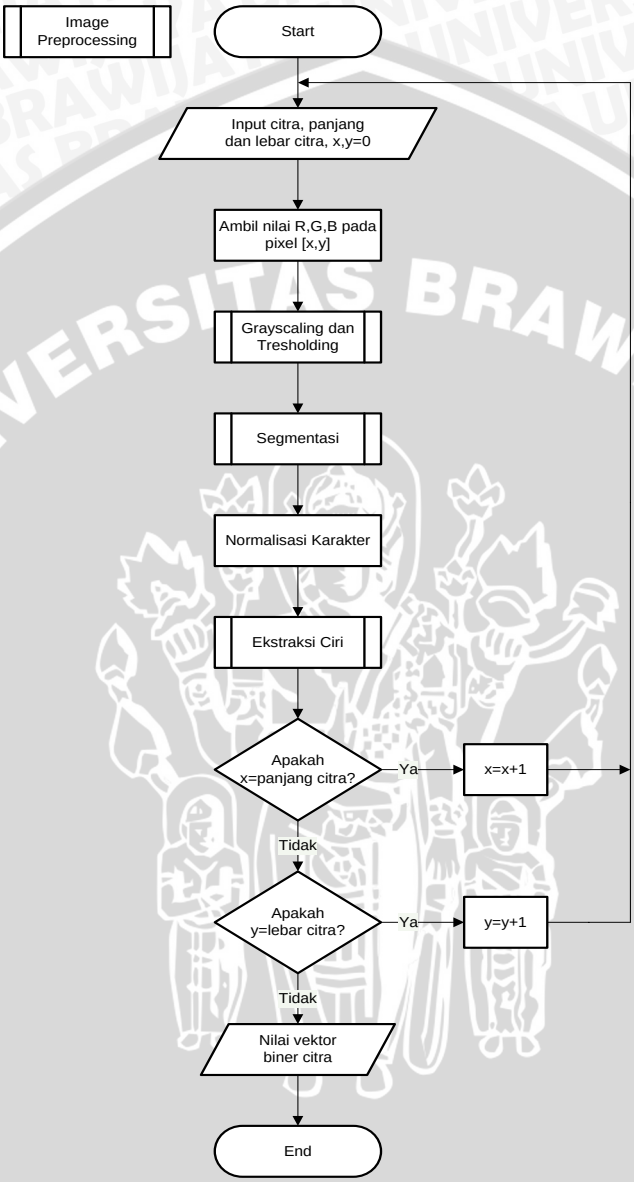


Gambar 3.2 Alur Proses

3.3.1 Image Preprocessing

Setelah citra diinputkan, selanjutnya citra akan mengalami *preprocessing* atau pemrosesan citra sebelum mengalami pelatihan atau pengenalan. Citra akan melalui beberapa proses, yaitu *grayscaling* dan *thresholding* sehingga citra berubah menjadi citra hitam putih yang memudahkan dalam proses berikutnya, segmentasi

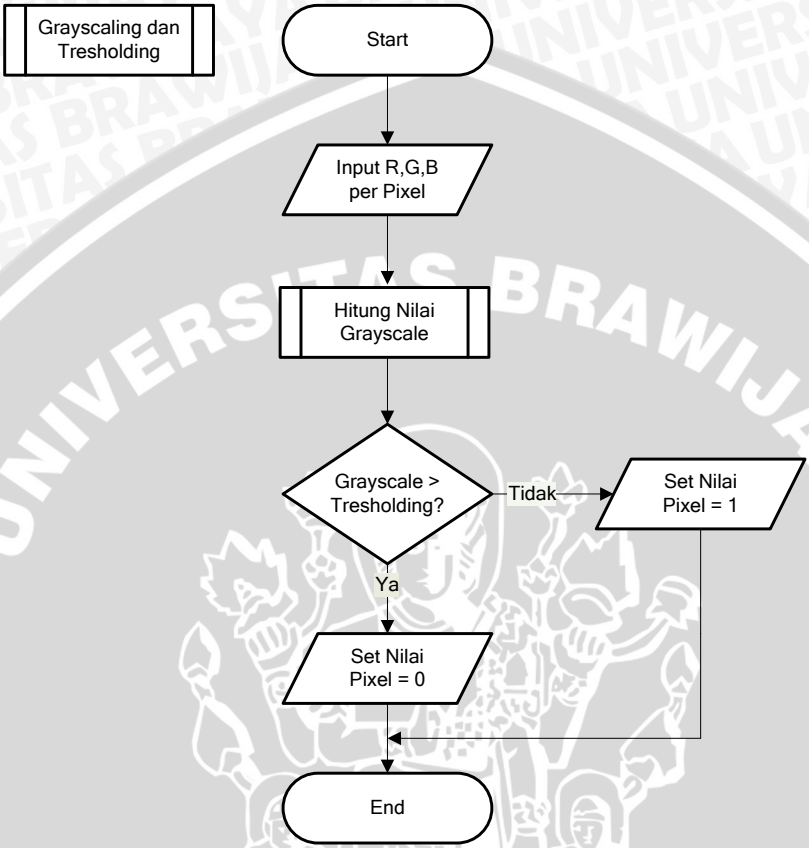
citra. Segmentasi citra menyeleksi tiap baris, kata, hingga karakter yang ada pada citra inputan. Setelah menjalani proses segmentasi maka akan didapatkan posisi, lebar dan tinggi masing-masing karakter, sehingga didapatkan data berupa matriks per karakter. Selanjutnya dilakukan normalisasi data per karakter agar mendapatkan data yang berukuran seragam. Proses terakhir yaitu ekstraksi setiap *pixel* dari citra ke dalam sebuah vektor agar dapat dihasilkan kumpulan data yang seragam pada setiap sampel yang akan diamati. Proses *image preprocessing* ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.3.



Gambar 3.3 Alur *Image Preprocessing*

3.3.1.1 Grayscaling dan Thresholding

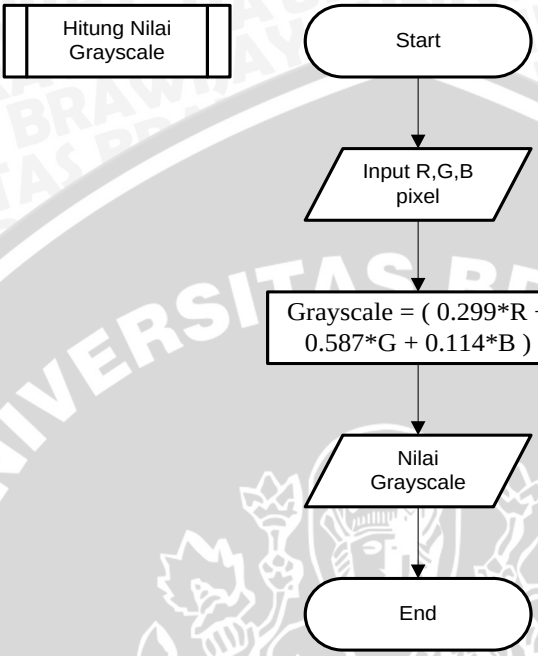
Pada proses *grayscaling* ini citra inputan yang berwarna dapat diubah menjadi citra yang tediri dari warna putih dan gradiasi warna hitam dengan menggunakan representasi warna *RGB.* Pengubahan citra ke dalam bentuk *grayscale* ini dilakukan dengan mengambil nilai *pixel* dari suatu citra inputan yang kemudian dihitung dengan persamaan yang dipilih (persamaan 2.22) dari persamaan yang lain dikarenakan persamaan tersebut menghasilkan *output* yang lebih baik dari sistem yang dibuat berdasarkan *trial* dan *error* dalam beberapa percobaan. Proses *thresholding* dilakukan dengan cara memeriksa apakah nilai intensitas dari sebuah *pixel* berada di bawah atau di atas sebuah nilai *intensity threshold* yang telah ditentukan. Nilai *threshold* yang digunakan dalam penelitian ini adalah 150. Apabila nilai *pixel* tersebut berada di atas batas nilai yang telah ditentukan, maka *pixel* tersebut akan diubah menjadi putih yang berarti bahwa *pixel* tersebut merupakan *background*, dan sebaliknya bila *pixel* tersebut berada di bawah batas nilai yang ditentukan maka *pixel* tersebut akan diubah menjadi berwarna hitam yang berarti dianggap sebuah karakter. Proses *grayscaling* dan *thresholding* ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.4.



Gambar 3.4 Alur *Grayscale* dan *Thresholding*

3.3.1.1.1 Hitung Nilai Grayscale

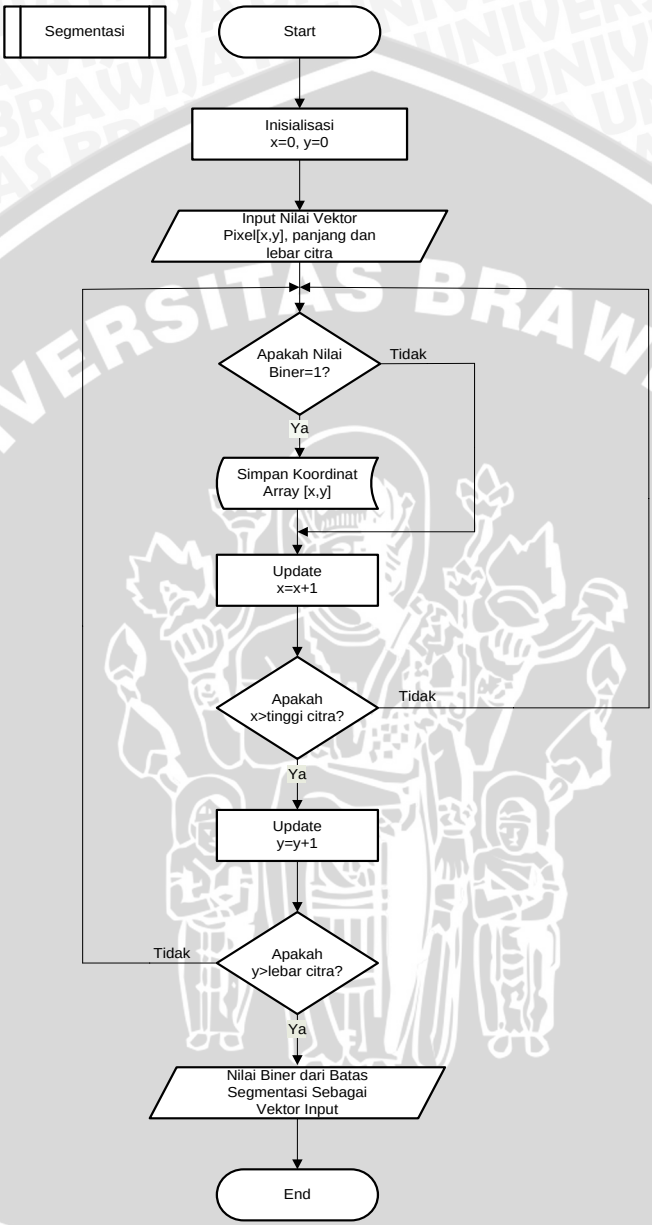
Pada proses ini, sistem menerima inputan berupa nilai RGB tiap-tiap *pixel*. Kemudian dilanjutkan dengan menghitung nilai *grayscale*, dengan menggunakan persamaan 2.18. Setelah didapatkan nilai *grayscale* tiap *pixel*, sistem melanjutkan ke proses selanjutnya. Proses menghitung nilai *grayscale* ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.5.



Gambar 3.5 Alur Hitung Nilai *Grayscale*

3.3.1.2 Proses Segmentasi

Pada proses segmentasi dilakukan dengan memetakan jumlah titik hitam setiap baris pada gambar ke sumbu *y* (*Y-Mapping)* dan setiap baris karakter hasil pemetaan tersebut dipetakan lagi ke sumbu *x* (*X-Mapping).* Setiap koordinat karakter hasil segmentasi disimpan dalam variabel *array* yang sudah disiapkan. Proses segmentasi ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.6.



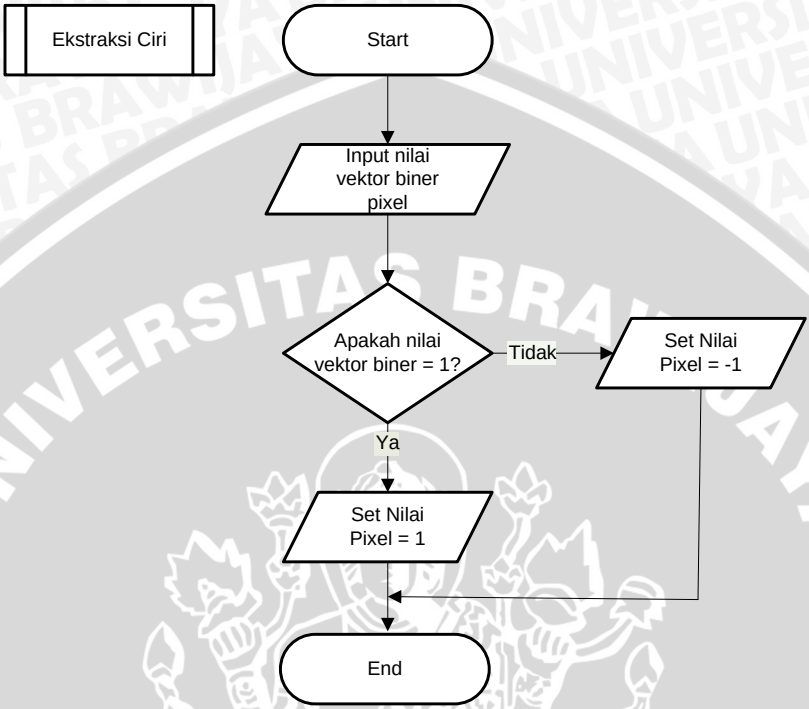
Gambar 3.6 Alur Proses Segmentasi

3.3.1.3 Proses Normalisasi

Karena data yang dihasilkan dari proses segmentasi dapat berbeda-beda dimensinya. Oleh karena itu untuk mendapatkan data yang seragam, akurat dan konsisten dari setiap sampel, data gambar hasil segmentasi tersebut akan dinormalisasikan (*stretch/shrink)* menjadi gambar dengan ukuran 10x10 *pixel*. Dengan demikian jumlah area yang ada pada setiap sampel akan bersesuaian dengan jumlah *neuron input* jaringan syaraf tiruan yang akan digunakan.

3.3.1.4 Ekstraksi Ciri

Setelah didapatkan informasi matrik hasil proses normalisasi setiap karakter dalam dimensi matrik 10x10, selanjutnya adalah mengektraksi setiap *pixel* dari citra ke dalam sebuah vektor hal ini dilakukan agar dapat dihasilkan kumpulan data yang seragam pada setiap sampel yang akan diamati. Vektor yang terbentuk bernilai 1 dan -1. Ciri-ciri citra adalah pixel-pixel yang memiliki nilai 1 dan 0. Pixel bernilai 1 adalah pixel berwarna hitam atau objek, sebaliknya pixel bernilai 0 adalah pixel berwarna putih atau *background*. Pada proses ini pixel-pixel yang yang bernilai 1 dan 0 dicek kembali, apabila pixel tersebut berwarna hitam (1) maka diset tetap dengan nilai 1, sebaliknya jika pixel berwarna putih (0) akan diset dengan nilai -1. Alasannya karena JST *Hopfield* berkerja dengan nilai 1 atau -1 sehingga fungsi aktifasi yang digunakan adalah *bipolar threshold (threshold function)* yang mempunyai range -1 sampai 1. Nilai matrik dari hasil proses ini selanjutnya akan diubah menjadi matrik (1 x 256) yang nantinya akan digunakan sebagai masukan JST. Proses segmentasi ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.7.



Gambar 3.7 Alur Proses Ekstraksi Ciri

3.3.2 Proses Pelatihan

Aplikasi dari proses pelatihan seperti ditunjukkan pada gambar 3.8 dapat dijelaskan sebagai berikut.

1. Menciptakan pola vektor *U*, yang merupakan himpunan *n* simpul untuk pola yang ke–i
2. Menciptakan matriks bobot koneksi

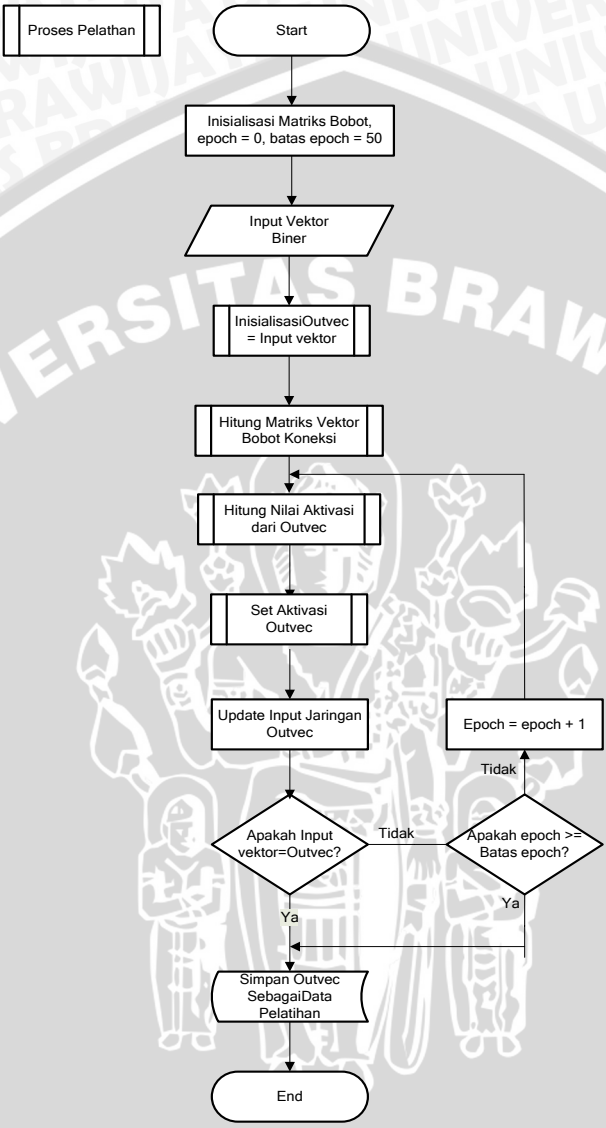
Siapkan U_i untuk i = 1...10.

Inisialisasi matriks *W* (orde nxn) untuk menyimpan pola menggunakan persamaan 2.12.

3. Mengerjakan langkah 4-6 selama pola belum konvergen input≠output.

4. Untuk tiap vektor U_i, i = (1,2,3,...,n) , hitung nilai y_in_i dengan persamaan 2.13
5. Hitung $Outvec_1$= $f(y\_in_i)$ di mana f adalah fungsi ambang (*threshold function)* menggunakan persamaan 2.14
6. Update *neuron input* jaringan dengan komponen $Outvec_i$
7. Test konvergensi, yaitu membandingkan tiap-tiap unit dari input vector dengan tiap-tiap unit dari output vector. Jika hasilnya sesuai/sama, maka didapatkan nilai output vector sebagai data pelatihan.

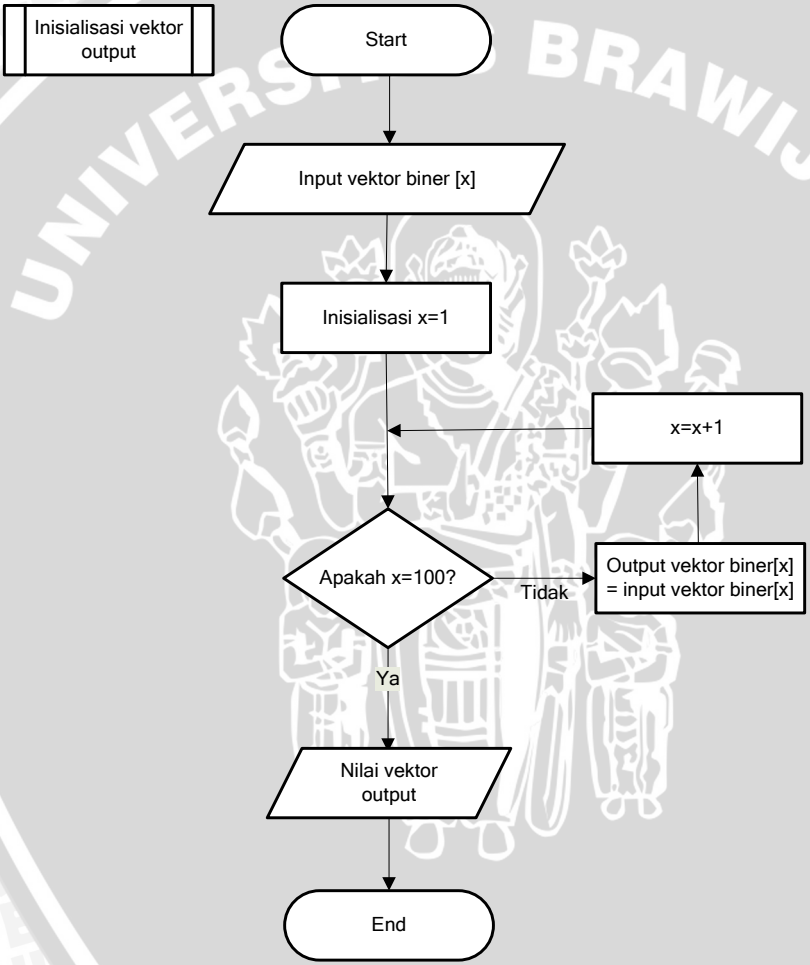
Proses pelatihan ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.8.



Gambar 3.8 Alur Proses Pelatihan

3.3.2.1 Inisialisasi Vektor Output

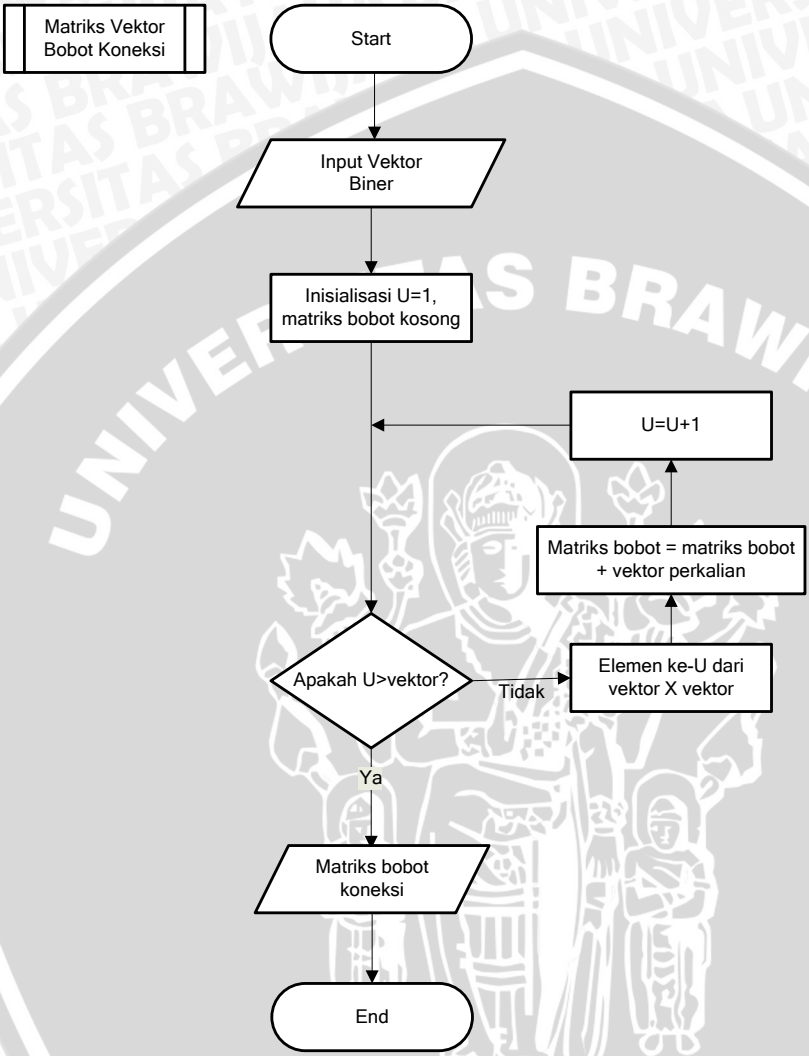
Pada proses inisialisasi vektor output ini, tiap-tiap vektor output pada array ke-x akan diisikan nilai dari vektor input pada array ke-x, sehingga akan ada 100 nilai vektor output yang bersesuaian dengan 100 nilai vektor input (berasal dari matriks neuron 10x10). Proses inisialisasi vektor output ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.9.



Gambar 3.9 Alur Inisialisasi Vektor Output

3.3.2.2 Matriks Vektor Bobot Koneksi

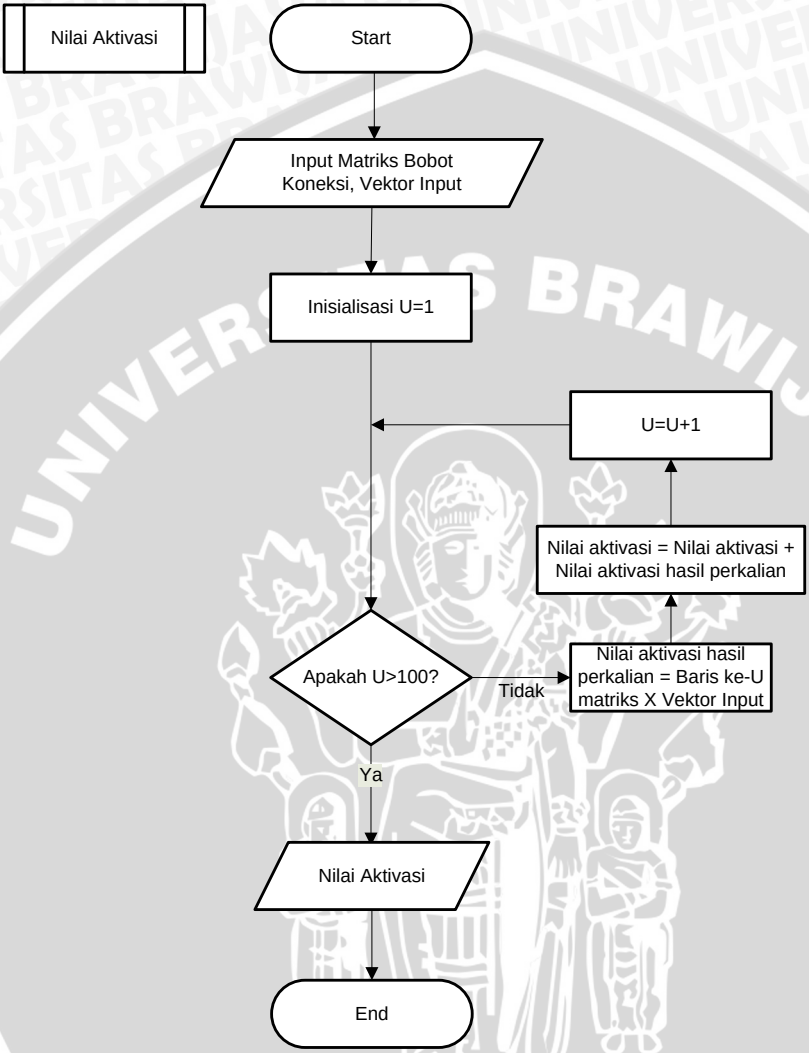
Pada proses perhitungan matriks vektor bobot koneksi ini, mula-mula diinisialisasikan sebuah matriks koneksi bobot kosong. Selanjutnya matriks diisi dengan hasil perkalian antar tiap elemen ke-U pada vektor dengan vektor itu sendiri, sehingga didapat matriks vektor bobot koneksi. Proses perhitungan matriks vektor bobot koneski ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.10.



Gambar 3.10 Alur Perhitungan Matriks Vektor Bobot Koneksi

3.3.2.3 Nilai Aktivasi

Pada proses perhitungan nilai aktivasi ini, tiap-tiap baris pada matriks bobot koneksi akan dikalikan dengan vektor inputan, sehingga didapatkan nilai aktivasi tiap baris. Proses perhitungan nilai aktivasi ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.11.

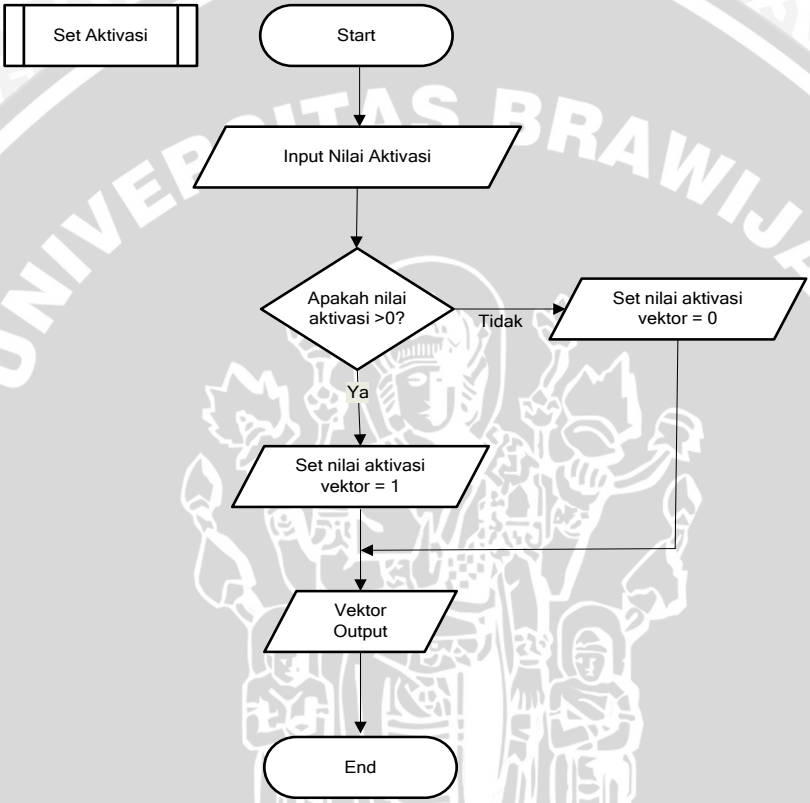


Gambar 3.11 Alur Perhitungan Nilai Aktivasi

3.3.2.4 Set Nilai Aktivasi Output

Pada proses penentuan nilai aktivasi output ini, inputan nilai aktivasi dari proses sebelumnya akan diseleksi, apabila nilai tersebut

lebih dari 0, maka nilai aktivasi vektor outputnya adalah 1. Kebalikannya, apabila nilai tersebut kurang dari 0, maka nilai aktivasi outputnya 0. Proses penentuan nilai aktivasi output ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.12.



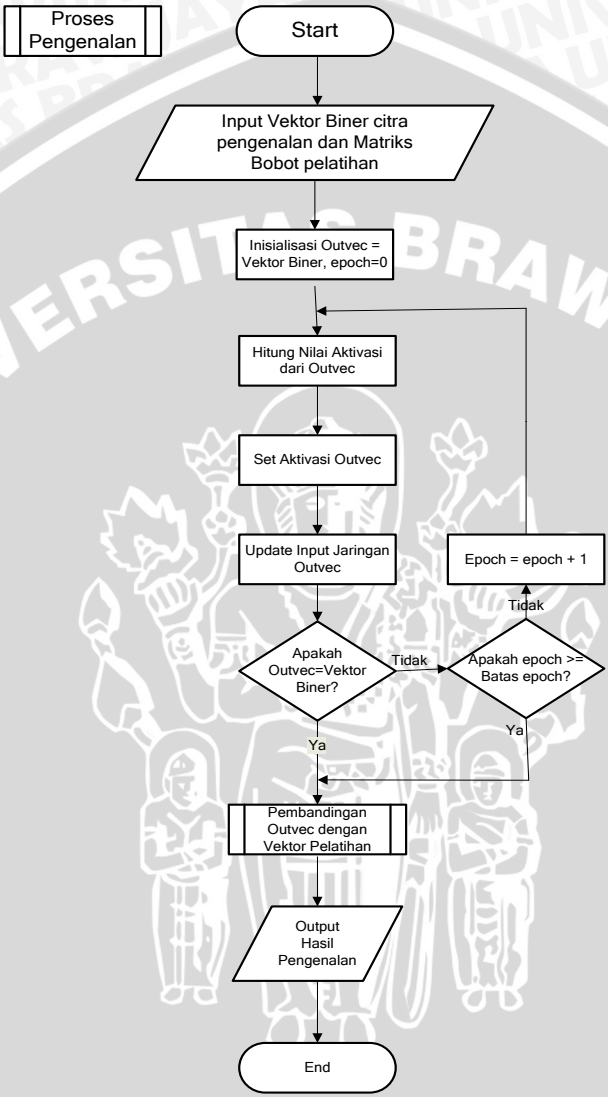
Gambar 3.12 Alur Set Aktivasi Output

3.3.3 Proses Pengenalan

Aplikasi dari proses pengenalan seperti ditunjukkan pada gambar 3.13 dapat dijelaskan sebagai berikut.

1. Mendapatkan input berupa vektor biner dan matriks bobot dari pelatihan sebelumnya
2. Mengerjakan langkah 3-5 selama pola belum konvergen input≠output.
3. Untuk tiap vektor U_i, i = (1,2,3,...,n) , hitung nilai y_in_i dengan persamaan 2.13
4. Hitung $Outvec_1 = f(y\_in_i)$ di mana f adalah fungsi ambang (*threshold function)* menggunakan persamaan 2.14
5. Update *neuron input* jaringan dengan komponen $Outvec_i$
6. Test konvergensi.
7. Pembandingan antara vektor output dengan data dari pelatihan, dibandingkan tiap vektor output dengan vektor yang ada dalam database, kemudian diseleksi yang mempunyai nilai kemiripan paling tinggi.
8. Output hasil pengenalan.

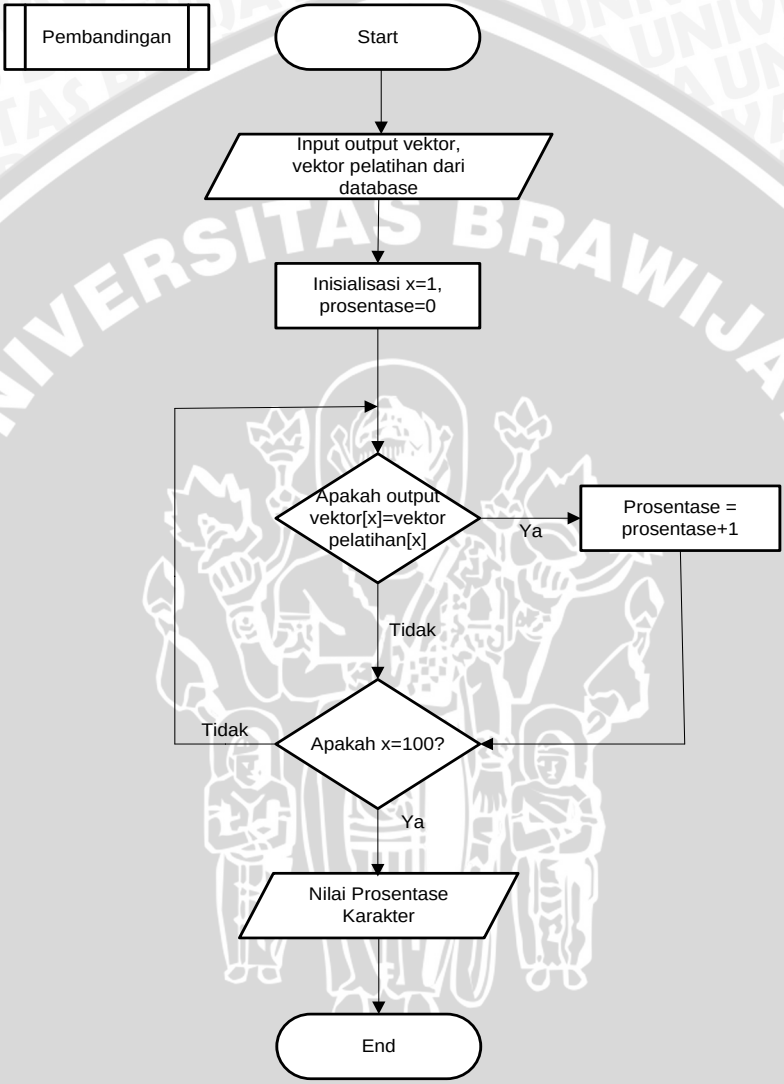
Proses pengenalan ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.13.



Gambar 3.13 Alur Proses Pengenalan

3.3.3.1 Pembandingan Outvec dengan Vektor Pelatihan

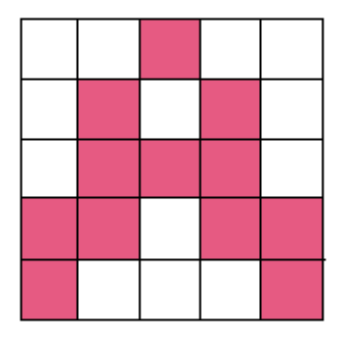
Pada proses ini, vektor pelatihan yang ada dalam database akan di-*load* dan disimpan dalam sebuah tabel. Selanjutnya untuk tiap karakter, vektor pelatihannya akan dibandingkan dengan vektor uotput untuk mngetahui prosentase kemiripannya. Setelah menghitung semua prosentase kemiripan tiap-tiap karakter, maka diambil karakter dengan tingkat prosentase terbesar sebagai output hasil pengenalan. Proses pembandingan outvec dengan vektor pelatihan ini digambarkan dalam bentuk *flowchart* seperti pada gambar 3.14.



Gambar 3.14 Alur Pembandingan Outvec dengan Vektor Pelatihan

3.4 Contoh Perhitungan Manual

Berikut ini merupakan contoh perhitungan manual untuk sebuah karakter (A) dengan ukuran 5x5 pixel (contoh pada gambar 3.15) seperti ditunjukkan pada tabel 3.1 menggunakan persamaan 2.22.



Gambar 3.15 Contoh Data

Tabel 3.1 Data Nilai RGB per Pixel

Pixel ke	Nilai Red	Nilai Green	Nilai Blue
1	255	255	255
2	255	255	255
3	230	90	130
4	255	255	255
5	255	255	255
6	255	255	255
7	230	90	130
8	255	255	255
9	230	90	130
10	255	255	255
11	255	255	255
12	230	90	130
13	230	90	130

14	230	90	130
15	255	255	255
16	230	90	130
17	230	90	130
18	255	255	255
19	230	90	130
20	230	90	130
21	230	90	130
22	255	255	255
23	255	255	255
24	255	255	255
25	230	90	130

Setelah mendapatkan nilai R,G, dan B tiap pixel, selanjutnya dihitung nilai *grayscale*nya berdasarkan persamaan (3.1). Didapatkan data seperti ditunjukkan pada tabel 3.2.

Tabel 3.2 Data Nilai *Grayscale* per Pixel

Pixel ke	0.299 x R	0.587 x G	0.114 x B	Nilai *Grayscale*
1	255	255	255	255
2	255	255	255	255
3	68.77	52.83	14.82	136.4
4	255	255	255	255
5	255	255	255	255
6	255	255	255	255
7	68.77	52.83	14.82	136.4
8	255	255	255	255
9	68.77	52.83	14.82	136.4
10	255	255	255	255
11	255	255	255	255
12	68.77	52.83	14.82	136.4

13	68.77	52.83	14.82	136.4
14	68.77	52.83	14.82	136.4
15	255	255	255	255
16	68.77	52.83	14.82	136.4
17	68.77	52.83	14.82	136.4
18	255	255	255	255
19	68.77	52.83	14.82	136.4
20	68.77	52.83	14.82	136.4
21	68.77	52.83	14.82	136.4
22	255	255	255	255
23	255	255	255	255
24	255	255	255	255
25	68.77	52.83	14.82	136.4

Data nilai *grayscale* per pixel kemudian dibandingkan dengan nilai *intensity threshold* yang telah ditentukan. Nilai *intensity threshold* yang digunakan pada penelitian ini adalah 150. Didapatkan data matriks biner seperti ditunjukkan pada tabel 3.3

Tabel 3.3 Data Nilai Biner per Pixel

Pixel ke	Nilai *Grayscale*	Nilai Biner	**Pixel ke**	Nilai *Grayscale*	Nilai Biner
1	255	0	**14**	136.4	1
2	255	0	**15**	255	0
3	136.4	1	**16**	136.4	1
4	255	0	**17**	136.4	1
5	255	0	**18**	255	0
6	255	0	**19**	136.4	1
7	136.4	1	**20**	136.4	1
8	255	0	**21**	136.4	1
9	136.4	1	**22**	255	0

10	255	0	**23**	255	0
11	255	0	**24**	255	0
12	136.4	1	**25**	136.4	1
13	136.4	1			

Berdasarkan tabel 3.3, karakter A mempunyai matrik biner 001000010100111011011110001, maka vektor bipolar untuk karakter A adalah:

y = (2.0-1, 2.0-1,2.1-1,2.0-1,2.0-1,2.0-1,2.1-1,2.0-1,2.1-1,2.0-1,2.0-1,2.1-1,2.1-1,2.1-1,2.0-1,2.1-1,2.1-1,2.0-1,2.1-1,2.1-1,2.1-1,2.0-1,2.0-1,2.0-1,2.1-1)

= -1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1 $^{\text{bipolar}}$

3.4.1 Contoh Perhitungan Proses Pelatihan

1. Berdasarkan tabel 3.3, vektor biner yang diujikan yaitu 001000010100111011011110001

Dengan vektor bipolar yang didapat adalah

-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 1 -1 1 1 1 -1 -1 -1 1 $^{\text{bipolar}}$

2. Membentuk matriks bobot dimulai dengan matriks koneksi bobot kosong.

$$\overbrace{\begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \end{pmatrix}}^{25} \Bigg\} 25$$

3. Inisialisasi matriks bobot untuk menyimpan pola

$$W_{ji} = \sum_{s=1}^{25} U_{sj} U_{si}$$

W_{ji} = (-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)

+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(-1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)
+(1)(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1)

Dengan diagonal utama diubah dengan nilai 0, didapat matriks bobot koneksi seperti pada gambar 3.16 berikut

0	1	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
1	0	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
-1	-1	0	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	1	1	-1	1	1	1	-1	-1	-1	1
1	1	-1	0	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
1	1	-1	1	0	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
1	1	-1	1	1	0	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
-1	-1	1	-1	-1	-1	0	-1	1	-1	-1	1	1	1	-1	1	1	-1	1	1	1	-1	-1	-1	1
1	1	-1	1	1	1	-1	0	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
-1	-1	1	-1	-1	-1	1	-1	0	-1	-1	1	1	1	-1	1	1	-1	1	1	1	-1	-1	-1	1
1	1	-1	1	1	1	-1	1	-1	0	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
1	1	-1	1	1	1	-1	1	-1	1	0	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	1	-1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	0	1	1	-1	1	1	-1	1	1	1	-1	-1	-1	1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	0	1	-1	1	1	-1	1	1	1	-1	-1	-1	1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	0	-1	1	1	-1	1	1	1	-1	-1	-1	1
1	1	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	0	-1	-1	1	-1	-1	-1	1	1	1	-1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	0	1	-1	1	1	1	-1	-1	-1	1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	1	0	-1	1	1	1	-1	-1	-1	1
1	1	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	0	-1	-1	-1	1	1	1	-1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	1	1	-1	0	1	1	-1	-1	-1	1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	1	1	-1	1	0	1	-1	-1	-1	1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	1	1	-1	1	1	0	-1	-1	-1	1
1	1	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	0	1	1	-1
1	1	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	0	1	-1
1	1	-1	1	1	1	-1	1	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	1	1	0	-1
-1	-1	1	-1	-1	-1	1	-1	1	-1	-1	1	1	1	-1	1	1	-1	1	1	1	-1	-1	-1	0

Gambar 3.16 Matriks Bobot Koneksi

4. Dianggap bahwa output vektor ≠ input vektor, selanjutnya hitung nilai aktivasi

$$y\_in_i = \sum_{i=1}^{25} W_{ji} U_i$$

y_in_i = (0 0 1 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 1 0 0 0 1) * (0 1 -1 1 1 1 -1 1 -1 1 1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 -1),(1 0 -1 1 1 1 -1 1 -1 1 1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 -1),(-1 -1 0 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1),...,(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 0)

y_in_i = -12 -12 11 -12 -12 -12 11 -12 11 -12 -12 11 11 11 -12 11 11 -12 11 11 11 -12 -12 -12 11

5. Set aktivasi vektor output, dengan fungsi

$$f(t) = \begin{cases} 1 \text{ jika } y\_in \geq 0 \\ 0 \text{ jika } y\_in \leq 0 \end{cases}$$

Didapat vektor output : 001000101001110110110001

6. Uji konvergen, didapat output vektor = input vektor.

3.4.2 Contoh Perhitungan Proses Pengenalan

1. Karakter yang diujikan seperti pada gambar 3.15, didapatkan vektor biner sebagai berikut

001000101001110110110001

Dengan vektor bipolar yang didapat adalah

-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 -1 -1 -1 1 [bipolar]

2. Matriks bobot yang digunakan adalah matriks bobot yang digunakan pada proses pelatihan, seperti tercantum pada gambar 3.16

3. Hitung nilai aktivasi

$$y\_in_i = \sum_{i=1}^{25} W_{ji} U_i$$

y_in_i = (0 0 1 0 0 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 1 0 0 0 1) * (0 1 -1 1 1 1 -1 1 -1 1 1 1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 -1),(1 0 -1 1 1 1 -1 1 -1 1 1 -1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1 1 -1),(-1 -1 0 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 1 -1 1 1 1 -1 -1 -1 1),...,(-1 -1 1 -1 -1 -1 1 -1 1 -1 -1 1 1 1 -1 1 1 -1 1 1 1 1 -1 -1 -1 0)

y_in_i = -12 -12 11 -12 -12 -12 11 -12 11 -12 -12 11 11 11 -12 11 11 -12 11 11 11 -12 -12 -12 11

4. Set aktivasi vektor output, dengan fungsi

$$f(t) = \begin{cases} 1 \text{ jika } y\_in \geq 0 \\ 0 \text{ jika } y\_in \leq 0 \end{cases}$$

Didapat vektor output : 001000101001110110110001

5. Uji konvergen, didapat output vektor = input vektor.

6. Perbandingan antara vektor output pengenalan dengan vektor biner pelatihan

Vektor output pengenalan = 001000101001110110110001

Vektor biner pelatihan = 001000101001110110110001

7. Output hasil pengenalan “huruf A”

3.5 Perancangan Uji Coba

Pada bagian perancangan uji coba akan dijelaskan mengenai pengujian data. Pengujian pertama dilakukan dengan citra yang berisi satu buah karakter yang merepresentasikan setiap karakter. Output sistem kemudian disimpan dalam tabel 3.4.

Tabel 3.4 Tabel Pengujian Tiap Karakter

Karakter	Output Pengenalan	
	Output	Validitas

Selanjutnya dihitung prosentase akurasi berdasarkan output karakter, disimpan daam tabel 3.5

Tabel 3.5 Tabel Prosentase Hasil Pengujian Tiap Karakter

Jumlah Karakter	Keterangan		
	Jumlah Valid	Error Rate (%)	Akurasi (%)

Pengujian selanjutnya dilakukan dengan citra yang berisi rangkaian kata. Output sistem disimpan dalam tabel 3.6.

Tabel 3.6 Tabel Pengujian Rangkaian Kata

Keterangan	Hasil
Ukuran file	
Jumlah kata	
Jumlah baris	
Jumlah karakter (tanpa spasi)	
Jumlah pengenalan valid	
Jumlah pengenalan error	
Akurasi	

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi

Lingkungan implementasi yang dijelaskan dalam sub bab ini meliputi lingkungan implementasi perangkat keras dan perangkat lunak.

4.1.1 Lingkungan Implementasi Perangkat Keras

Perangkat keras yang digunakan dalam pembuatan aplikasi Penerapan *Hopfield Neural Network* Untuk Pengenalan Tulisan Tangan Pada Sebuah Citra Digital ini menggunakan Notebook dengan spesifikasi sebagai berikut :

1. Processor Intel(R) Core(TM)2 Duo CPU T5800 @ 2.00GHz (2 CPUs), ~2.0GHz.
2. Memory 4096MB RAM.
3. Harddisk 250GB.
4. VGA NVIDIA GeForce 9300M GS 1777MB.

4.1.2 Lingkungan Implementasi Perangkat Lunak

Perangkat lunak yang digunakan untuk mendukung pembuatan aplikasi pengenalan tulisan tangan ini meliputi :

1. Sistem Operasi Windows 7 32-bit.
2. Delphi 7.
3. Adobe Photoshop CS5.

4.2 Implementasi Program

Berdasarkan analisis dan perancangan yang telah dijelaskan pada bab 3, maka pada sub bab ini akan dijelaskan proses pengimplementasiannya.

4.2.1 Implementasi *Load* Data/Citra

Untuk memuat gambar ke dalam Timage yang telah dipesan, diperlukan sebuah TopenPictureDialog yang akan memanggil nama file yang akah diproses. Nama file kemudian di*load* ke dalam parameter *picture* dalam Timage.

Proses *load* citra ditunjukkan pada *source code* 4.1.

```
procedure TForm2.openimage(var image:string);
begin
if OpenPictureDialog1.Execute then
   image:=OpenPictureDialog1.Filename;
   Image1.Picture.LoadFromFile(image);
end;
```

Source Code 4.1 Memuat Citra

4.2.2 Implementasi Grayscaling Dan Thresholding

Setelah komponen Timage terisi dengan citra yang akan diproses, alngkah selanjutnya mengubah nilai RGB tiap *pixel* pada citra tersebut menjadi nilai biner, yaitu 0 dan 1, dengan cara mengkonversi citra pada Timage ke dalam skala keabuan menggunakan persamaan 2.22. Selanjutnya akan diseleksi berdasarkan nilai ambang, dalam penelitian kali ini nilai ambang yang digunakan adalah 150. Jika nilai skala keabuan melebihi nilai ambang, maka nilai biner pada *pixel* tersebut adalah 1, sedangkan jika skala keabuannya dibawah nilai ambang, maka nilai binernya adalah 0.

Proses *grayscaling* dan *thresholding* citra ditunjukkan pada *source code* 4.2.

```
procedure TForm2.grayscale(var lokasi:string);
begin
Image1.Picture.LoadFromFile(lokasi);
image3.Visible:=true;
with Image1.Picture.Bitmap do
begin
  for w:=0 to image1.Picture.Width-1 do
  begin
    for h:=0 to image1.picture.Height-1 do
    Begin
      warna:=ColorToRgb(canvas.Pixels[w,h]);
      R:=GetRValue(warna);
      G:=GetGValue(warna);
      B:=GetBValue(warna);
      if (round( 0.299*R + 0.587*G + 0.114*B
))>=150 then
      vRgbtemp:=255 else vRgbtemp:=0 ;
      if vRgbtemp=255 then
      segmn:=0 else segmn:=1;
      canvas.Pixels[w, h]:=Rgb(vRgbTemp, vRgbTemp,
vRgbTemp);
      pix[h,w]:=segmn;
    end;
  end;
end;
end;
```

Source Code 4.2 Proses *Grayscaling* dan *Thresholding*

Setelah mendapatkan citra biner, proses selanjutnya adalah menyeleksi citra inputan untuk menghasilkan inputan berupa karakter tunggal. Cara yang dilakukan adalah:

1. Menghitung jumlah nilai biner 1 pada tiap baris *pixel.*
2. Menentukan batas atas dan batas bawah baris kalimat, dengan cara menyeleksi apakah pada baris *pixel* tersebut jumlah nilai biner 1 nya lebih dari 0. Jika ya, maka baris *pixel* tersebut merupakan awal baris ke-x. Selanjutnya dilakukan pengecekan untuk baris dibawahnya, apakah jumlah nilai biner 1 nya lebih dari 0. Jika tidak, maka baris sebelumnya adalah akhir baris dari baris ke-x.

3. Menghitung jumlah nilai biner 1 pada tiap kolom *pixel* dalam batas awal baris dan akhir baris ke-x.
4. Menentukan batas awal dan akhir huruf, dengan cara menyeleksi apakah pada kolom *pixel* tersebut jumlah nilai biner 1 nya lebih dari 0. Jika ya, maka kolom *pixel* tersebut merupakan awal huruf ke-y. Selanjutnya dilakukan pengecekan untuk kolom di samping kanannya, apakah jumlah nilai biner 1 nya lebih dari 0. Jika tidak, maka kolom di samping kirinya adalah akhir huruf dari huruf ke-y.
5. Menghitung jumlah nilai biner 1 pada tiap baris *pixel* dalam batas awal huruf dan akhir huruf ke-y.
6. Menentukan batas atas dan batas bawah huruf, dengan cara menyeleksi apakah pada baris *pixel* tersebut jumlah nilai biner 1 nya lebih dari 0. Jika ya, maka baris *pixel* tersebut merupakan batas atas huruf ke-y. Selanjutnya dilakukan pengecekan untuk baris dibawahnya, apakah jumlah nilai biner 1 nya lebih dari 0. Jika tidak, maka baris sebelumnya adalah batas bawah dari huruf ke-y.

Hasil akhir yang didapat dari proses segmentasi ini adalah sebuah nilai yang disimpan dalam 4 array yakni awalhuruf[x,y], akhirhuruf[x,y], atashuruf[x,y] dan bawahhuruf[x,y].

Proses segmentasi citra ditunjukkan pada *source code* 4.3.

```
procedure TForm2.segmentation(var gambar:TImage);
begin
for i:=0 to gambar.picture.Height-1 do
begin
  for j:=0 to gambar.Picture.Width-1 do
  begin
    x:=x+pix[i,j];
  end;
  sum[a]:=x;
  a:=a+1;
  x:=0;
end;

for k:=0 to gambar.Picture.Height-1 do
begin
  if sum[k]>0 then if sum[k-1]=0 then m:=m+1;
  if sum[k]>0 then
  for l:=k downto 0 do
  begin
    if sum[l]>0 then awalbaris[m]:=l else break;
    akhirbaris[m]:=k;
  end;
end;

for z:=1 to m do
begin
  u:=0;
  n:=0;
  for i:=0 to gambar.picture.Width-1 do
  begin
    for j:=awalbaris[z] to akhirbaris[z] do
    begin
      x:=x+pix[j,i];
    end;
    sum2[u]:=x;
    u:=u+1;
    x:=0;
  end;

  //tentukan batas awal dan akhir huruf
  for k:=0 to gambar.Picture.Width-1 do
  begin
    if sum2[k]>0 then if sum2[k-1]=0 then n:=n+1;
    if sum2[k]>0 then
```

```
  for k:=0 to gambar.Picture.Width-1 do
  begin
    if sum2[k]>0 then if sum2[k-1]=0 then n:=n+1;
    if sum2[k]>0 then
    for l:=k downto 0 do
    begin
      if sum2[l]>0 then awalkolom[z,n]:=l else break;
      akhirkolom[z,n]:=k;
    end;
  end;
end;
```

Source Code 4.3 Proses Segmentasi

Setelah menentukan batas-batas inputan untuk tiap karakter, proses selanjutnya adalah mengambil nilai biner di dalam batas-batas tersebut. Nilai-nilai ini kemudian dimuat ulang ke dalam sebuah komponen Timage lain yang telah ditentukan ukurannya, yaitu 10x10 *pixel*. Sehingga didapatkan sejumlah 100 nilai biner sebagai vektor input pada proses selanjutnya.

Proses normalisasi ditunjukkan pada *source code* 4.4.

```
procedure TForm2.normalization(var s,t:integer);
begin
image2.Width := 10;
image2.Height:= 10;
image2.Canvas.StretchDraw(image2.Canvas.Cliprect,
image1.Picture.Graphic);
image2.Canvas.CopyRect(image2.Canvas.ClipRect,
                       image1.Canvas,
                       Rect(awalkolom[s,t],
                            atashuruf[s,t],
                            akhirkolom[s,t]+1,
                            bawahhuruf[s,t]+1));
end;
```

Source Code 4.4 Proses Normalisasi

4.2.5 Implementasi Ekstraksi Ciri

100 nilai biner yang didapat melalui proses normalisasi akan diubah menjadi nilai bipolar. Nilai biner 0 akan diubah menjadi -1, sedangkan nilai biner 1 tidak berubah.

Proses ekstraksi ciri ditunjukkan pada *source code* 4.5.

```
procedure TForm2.ekstrak(var vrgbtemp:integer);
begin
if vRgbtemp=255 then segmn:=-1 else segmn:=1;
end;
```

Source Code 4.5 Proses Ekstraksi Ciri

4.2.6 Implementasi Set Vektor Input

Set vektor input yang akan digunakan dalam perhitungan. Vektor input didapat dari citra dalam Timage hasil normalisasi.

Proses set vektor input ditunjukkan pada *source code* 4.6.

```
procedure TForm2.inoutvec(var i,j,s,t,aa:integer);
begin
input[aa]:=temp[j,i];
if input[aa]=-1 then input[aa]:=0 else input[aa]:=1;
neuron[aa]:=temp[j,i];
aa:=aa+1;
end;
```

Source Code 4.6 Proses Set Vektor Input

4.2.7 Implementasi Hitung Nilai Bobot dan Set Nilai Aktivasi

Proses ini hanya dilakukan dalam pelatihan. Pertama-tama menentukan nilai *epoch* atau batas perulangan, yaitu 50. Kemudian dilakukan proses penghitungan bobot, yakni bobot lama ditambahkan dengan hasil perkalian matriks neuron input. Akan dilakukan penyeleksian juga, untuk diagonal utama pada matriks bobot, nilainya adalah 0. Sementara nilai aktivasinya diperoleh dari penjumlahan dari perkalian baris ke-u pada matriks bobot dikalikan dengan vektor input. Dalam proses ini juga dilakukan uji konvergensi, yaitu apabila untuk setiap vektor input sama dengan vektor output, maka nilai konvergensi adalah *true*. Jika proses

mencapai *epoch* atau mencapai nilai konvergensi *true*, maka proses akan dihentikan.

Proses hitung nilai bobot dan set nilai aktivasi ditunjukkan pada *source code* 4.7.

```
procedure TForm2.mvbk(var i,j:integer);
begin
if i=j then bobot[i,j]:=0 else
bobot[i,j]:=bobot[i,j]+(neuron[i]*neuron[j]);
end;

procedure TForm2.aktiv(var i,j:integer);
begin
aktivasi[i]:=aktivasi[i]+(bobot[i,j]*input[j]);
end;

procedure TForm2.setaktiv(var i:integer);
begin
if aktivasi[i]>0 then output[i]:=1 else output[i]:=0;
end;
```

Source Code 4.7 Proses Hitung Nilai Bobot dan Set Nilai Aktivasi

4.2.8 Implementasi Pembandingan Data Latih

Data latih yang disimpan dalam database mula-mula di*load* ke dalam sebuah tabel, berisi inisialisasi karakter dan vektor output. Vektor output data latih inilah yang akan dibandingkan nilainya dengan vektor output hasil set nilai aktivasi pengenalan, sehingga untuk tiap karakter data latih akan diketahui prosentase kemiripian dengan karakter yang akan dikenali. Selanjutnya dipilih berdasarkan nilai prosentase terbesar, dan diambil data inisialisasi karakternya sebagai output karakter hasil pengenalan.

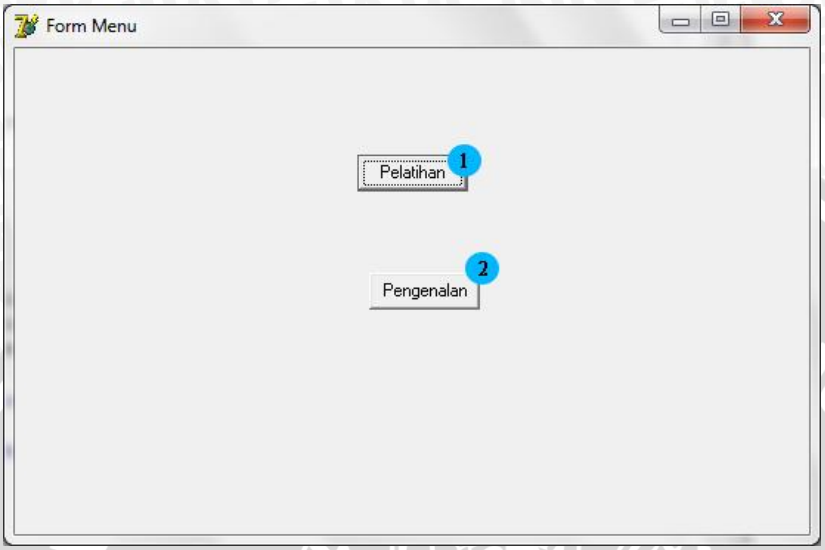
Proses pembandingan data latih ditunjukkan pada *source code* 4.8

```
procedure TForm3.persen(var s,t:integer);
begin
prosentase:=0;
for i:=1 to ADOQuery1.RecordCount do
begin
  prostemp:=0;
  for j:=2 to 101 do
  begin
    if neuron[j-1]=strtoint(SG.Cells[j,i]) then
prostemp:=prostemp+1;
  end;
  if prostemp>prosentase then prosentase:=prostemp;
  if prostemp>=prosentase then
kar[s,t]:=SG.Cells[1,i];
end;
end;
```

Source Code 4.8 Proses Pembandingan Data Latih

4.3 Implementasi Antarmuka

Sistem yang dirancang pada penelitian ini dibagi dalam 3 form utama, yaitu form menu yang berisi menu pilihan pelatihan / pengenalan, form pelatihan untuk memasukkan data latih dan form pengenalan. Gambar 4.1 menunjukkan antarmuka form menu.

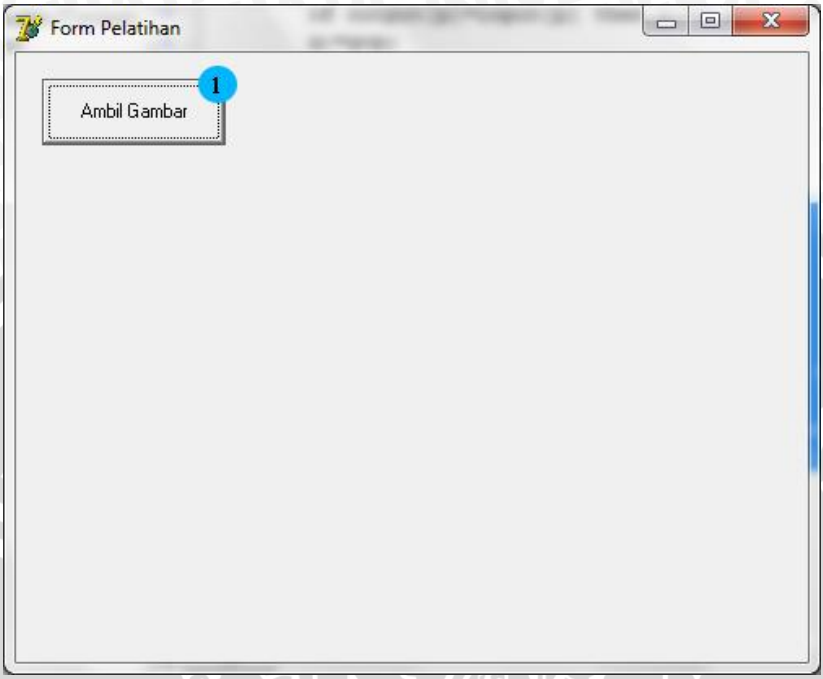


Gambar 4.1 Antarmuka Form Menu

Form menu adalah form utama, sebagai navigasi pengguna untuk langkah selanjutnya. Terdapat 2 tombol dalam form menu, yaitu:

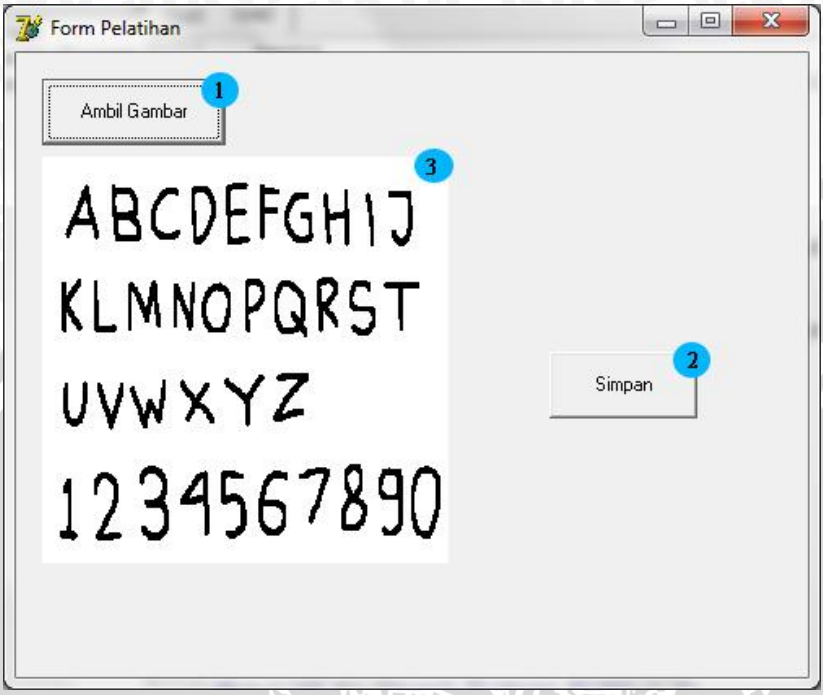
1. Tombol pelatihan, untuk membuka form pelatihan
2. Tombol pengenalan, untuk membuka form pengenalan.

Gambar 4.2 menunjukkan antarmuka awal form pelatihan.



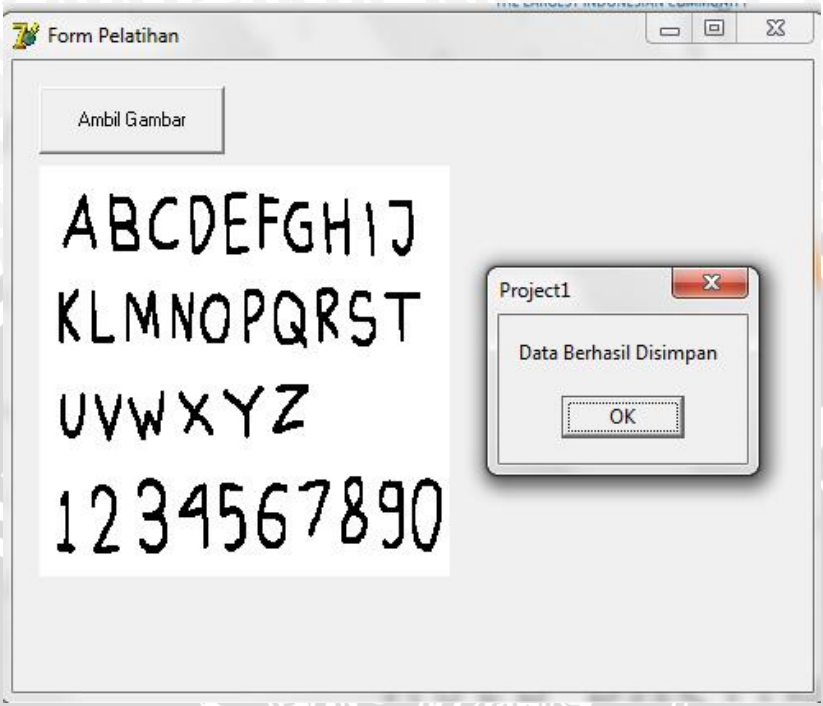
Gambar 4.2 Antarmuka Awal Form Pelatihan

Form pelatihan mula-mula hanya mengandung satu tombol, yaitu tombol ambil gambar, yang berfungsi untuk me*load* citra *training set* yang akan dilatihkan. Setelah citra dipilih, maka antarmuka akan berubah seperti pada gambar 4.3.



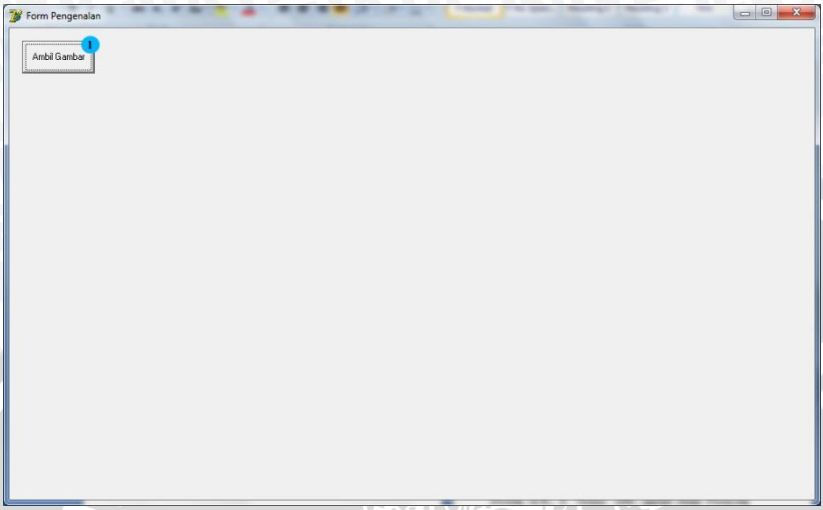
Gambar 4.3 Antarmuka Form Pelatihan

Setelah memilih citra *training set* yang akan dilatih, maka tampilan form akan berubah seperti pada gambar 4.3. Akan muncul sebuah panel gambar(3) yang merepresentasikan citra *training set* yang telah diubah menjadi citra hitam putih. Tombol(2) berfungsi untuk memproses citra *training set* menjadi data latih. Setelah tombol ditekan, akan muncul sebuah dialog *box* seperti gambar 4.4.



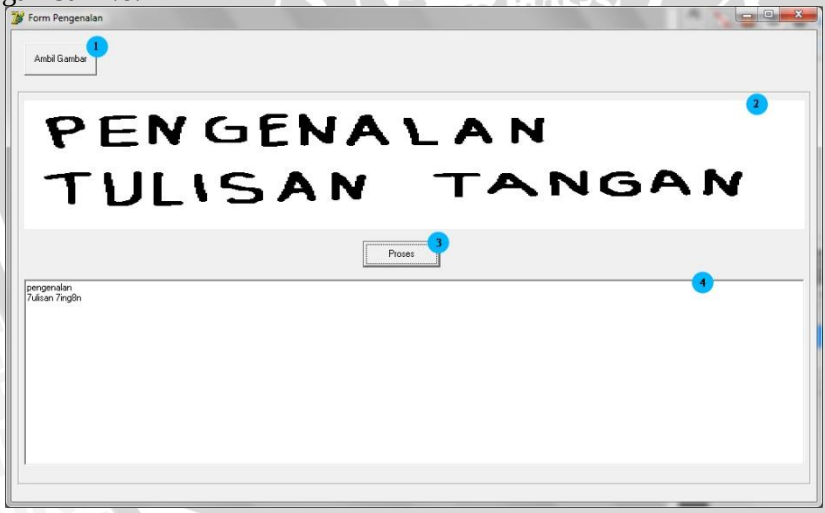
Gambar 4.4 Dialog *Box*

Gambar 4.5 menunjukkan antarmuka form pengenalan.



Gambar 4.5 Antarmuka Awal Form Pengenalan

Form pengenalan mempunyai 1 tombol, berfungsi untuk memuat citra yang akan dikenali. Setelah citra dipilih, maka form akan menampilkan komponen-komponen yang tersembunyi, seperti pada gambar 4.6.



Gambar 4.6 Form Pengenalan

Komponen-komponen tersembunyi pada form pengenalan akan muncul begitu citra pengenalan dipilih. Terdapat komponen citra pengenalan(2), yaitu komponen yang menampilkan citra yang akan dikenali dalam format hitam-putih. Selanjutnya ada tombol proses(3), yaitu tombol untuk memproses citra yang akan dikenali sehingga didapatkan output hasil pengenalan seperti pada komponen(4).

4.4 Implementasi Uji Coba

Pada sub bab berikut dilakukan pembahasan mengenai implementasi dari pengujian pada sistem serta ringkasan hasil pengujiannya.

4.4.1 Hasil Pengujian Tiap Karakter

Untuk pengujian pertama dilakukan dengan gambar yang berisi karakter yang merepresentasikan setiap huruf dan angka. Hasil pengujian karakter ditunjukkan pada tabel 4.1.

Tabel 4.1 Tabel Uji Coba Tiap Karakter

Karakter	Output Pengenalan	
	Output	Validitas
A	A	✓
B	B	✓
C	C	✓
D	D	✓
E	E	✓
F	F	✓
G	G	✓
H	H	✓
I	I	✓
J	J	✓
K	K	✓
L	L	✓
M	M	✓
N	N	✓
O	O	✓
P	P	✓

Q	O	X
R	R	✓
S	S	✓
T	T	✓
U	U	✓
V	V	✓
W	W	✓
X	X	✓
Y	Y	✓
Z	Z	✓
0	0	✓
1	1	✓
2	2	✓
3	3	✓
4	4	✓
5	5	✓
6	6	✓
7	7	✓
8	8	✓
9	9	✓

Prosentase akurasi berdasarkan ouput karakter seperti ditunjukkan tabel 4.2.

Tabel 4.2 Tabel Prosentase Hasil Pengujian Karakter

Jumlah Karakter	Keterangan		
	Jumlah Valid	Error Rate (%)	Akurasi (%)
36	35	2.8	97.2

Dari tabel 4.1 kesalahan pengujian pengenalan terdapat pada karakter yang memiliki kemiripan bentuk, yaitu Q dan O. Dalam pengujian ini dilakukan dengan tujuan apakah sistem bisa mengenali dengan baik tiap pola karakter dengan membandingkan terhadap ukuran karakter yang sama dengan yang dilatihkan.Kesalahan kemungkinan disebabkan oleh proses normalisasi yang dilakukan oleh sistem, karena pada proses normalisasi *(stretch)* semua karakter diseragamkan ukurannya menjadi 10x10 *pixel*, hal inilah yang

kemudian menjadikan sistem salah dalam mengenali pola yang dimaksud dikarenakan terjadinya perubahan nilai vektor *input* suatu karakter akibat proses normalisasi tersebut sehingga jaringan stabil pada pola lain.

Pada sistem yang dibuat didesain dengan asumsi gambar yang digunakan sebagai input memiliki *noise* yang sekecil mungkin. Dalam pengambilan suatu *input* dari hasil *scanning* terkadang membuat gambar menjadi kurang tajam dari objek aslinya yang bisa dianggap *noise* karena bisa menurunkan akurasi JST dalam melakukan pengenalan, walaupun dalam perspektif penglihatan mata manusia masih bisa terbaca sebagai karakter itu sendiri tetapi tidak dengan sistem. Akibat adanya *noise* tersebut yang memungkinkan sistem tidak bisa mengenali karakter yang dimaksud dengan benar sehingga sistem akan mengambil dan menampilkan suatu pola yang memiliki kedekatan antara vektor input dengan pola target.

Akibat adanya pola input yang kurang sempurna seperti ketajaman gambar, objek karakter yang terlalu besar maupun terlalu kecil ukurannya. Untuk ketajaman gambar akan mempengaruhi dalam proses pemisahan *background* dengan objek yaitu proses *grayscaling dan thresholding* di mana terkadang nilai *Threshold* pada objek melebihi nilai *default Threshold* (150) sehingga objek tersebut yang seharusnya merepresentsikan suatu karakter tertentu akan terbaca sebagai *background.* Pada dasarnya walaupun *input* pengenalan berupa *image* hitam–putih tetapi *image* tersebut bukanlah *image* yang hanya tersusun atas warna hitam dan putih saja melainkan kombinasi dari representasi warna merah, hijau dan biru atau RGB (*Red, Green, Blue).* Sedangkan ukuran suatu karakter nantinya sangat berpengaruh dalam proses normalisasinya.

4.4.2 Hasil Pengujian Rangkaian Kata

Untuk pengujian kedua dilakukan dengan gambar yang berisi rangkaian kata dalam paragraf. Hasil pengujian seperti ditunjukkan pada tabel 4.3.

Tabel 4.3 Tabel Uji Coba Rangkaian Kata

Keterangan	Hasil
Ukuran File	500x172 *pixel*
Jumlah Kata	3
Jumlah Baris	2
Jumlah Karakter (Tanpa Spasi)	23
Jumlah Pengenalan Valid	19
Jumlah Pengenalan Error	4
Akurasi	82.6 %

Selain disebabkan adanya kemiripan bentuk, kesalahan pengenalan juga disebabkan karena proses segmentasi yang kurang baik pada dua karakter yang tersambung sangat tipis sehingga seharusnya terdapat dua karakter akan tetapi sistem menganggap itu sebagai satu karakter. Tersambung di sini dalam arti pada suatu koordinat *pixel* (x,y) ada bagian dari kedua karakter tersebut yang berhimpitan koordinatnya oleh karena itu sistem tidak bisa memisahkan kedua karakter tersebut sebagai dua karakter dikarenakan tidak memenuhi parameter yang ada sehingga sistem menganggapnya sebagai satu karakter. Semakin besar ukuran dari tulisan maka semakin kecil kemungkinan terjadinya kesalahan pada proses segmentasi. Kemungkinan terbesar kesalahan pengenalan pada ukuran tulisan yang lebih besar adalah kesalahan pada saat proses normalisasi yang menyebabkan JST stabil pada pola lain. Untuk ukuran tulisan yang lebih kecil kemungkinan kesalahan terjadi pada proses segmentasi. Pada percobaan ini terlihat sistem cukup baik dalam mengenali suatu rangkaian kata yaitu 82,6%.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Setelah melakukan pengujian pengenalan karakter dari suatu kata dalam paragraf menggunakan JST *Hopfield* baik secara ujicoba maupun berdasarkan teori, serta menganalisa hasil pengujian, maka dapat diambil kesimpulan-kesimpulan sebagai berikut:

1. Penerapan JST *Hopfield* pada penelitian ini yaitu dengan mengimplementasikan algoritma *Hopfield* kedalam proses pelatihan dan pengenalan. Algoritma *Hopfield* memproses 100 buah input neuron yang didapat dari hasil pemrosesan citra sehingga menghasilkan 100 buah output neuron, dimana output tersebut digunakan sebagai bahan pembanding output neuron lain pada saat pengenalan citra.
2. Persentase akurasi hasil pengenalan tiap karakter untuk ukuran yang sama dengan yang dilatihkan adalah 97,2%. Persentase akurasi hasil pengenalan rangkaian kata dalam satu paragraf adalah 82,6%. Pada pengujian pengenalan citra paragraf berwarna berdasarkan pengaruh parameter *threshold*, secara *default threshold* yang digunakan pada penelitian ini adalah 150. Rata-rata hasil pengenalan untuk setiap jenis ukuran karakter yang tidak dilatihkan bergantung pada ukuran karakter yang telah dilatihkan, apabila jenis karakter tersebut mirip dengan salah satu model karakter yang dilatihkan maka rata-rata hasil pengenalan akan tinggi.

5.2 Saran

1. Proses segmentasi karakter dapat disempurnakan dengan menggunakan beberapa metode pemrosesan citra yang lain.
2. Untuk pengembangan lebih lanjut diharapkan sistem bisa mengenali input gambar yang terkena *noise* dengan menambahkan metode untuk *filtering noise.*
3. Sistem diharapkan bisa mengenali simbol dan mengenali karakter dari kata yang tidak tegak lurus atau tercetak miring dan/atau bersambung.


UNIVERSITAS BRAWIJAYA

DAFTAR PUSTAKA

Awcock, G.W., R. Thomas. 1996. *Applied Image Processing.* McGraw - Hill Intemational Edition, Singapore.

Gonzales, Rafael C., Woods, Richard Eugene. 2007. *Digital Image Processing.* Prentice Hall.

Hermawan, Arif. 2006. *Jaringan Syaraf Tiruan Teori dan Aplikasi.* Penerbit Andi, Jogjakarta.

Kristanto, Triono. 2002. *Validasi Uang Kertas Rupiah dengan Teknologi Image Processing.* UKpetra, Surabaya.

Kusumadewi, Sri. 2003. *Artificial Intelligence (Teknik dan Aplikasinya).* Penerbit Graha Ilmu, Jogjakarta.

Lejap, Fatima Gelu. 2007. *Aplikasi Jaringan Saraf Tiruan Perbandingan Backpropagation dan Hopfield Untuk Pengenalan Huruf Pada Citra Digital.* Universitas Komputer Indonesia, Jakarta.

Matecki, U. Seehusen, S. 1995. *Hopfield Nets and Binarization Techniques of Grayscale Images.* EUFIT, Germany.

Munir, R. 2004. *Pengolahan Citra Digital Dengan Pendekatan Algoritmik.* Informatika, Bandung.

Pal, Anita. Singh, Dayashankar. 2010. *Handwritten English Character Recognition Using Neural Network.* U.P Technical University, Lucknow.

Prasetia, Dita Nurul. 2009. *Analisis Pengenalan Teks Cetak Menggunakan Neural Network Backpropagation.* Institut Teknologi Harapan Bangsa.

Pratomo, Deddy. 2011. *Rancang Bangun Perangkat Lunak Pengenalan Wajah dengan Menggunakan Hopfield Network.* STIKOM, Surabaya.

Puspitaningrum, Diah. 2006. *Pengantar Jaringan Saraf Tiruan.* Penerbit Andi, Jogjakarta.

Sada, Ira Herawati. 2004. *Pemodelan Wahah 3D Melalui Pendeteksian Fitur Wajah 2D Menggunakan Teknik Morphing.* IT Telkom, Bandung.

Siang, Jong Jek. 2004. *Jaringan Saraf Tiruan & Pemrogramannya Menggunakan Matlab.* Penerbit Andi, Jogjakarta.

Triwijoyo. 2007. *Penggunaan Jaringan Saraf Tiruan Hopfield untuk Pengenalan Pola Citra Dua Dimensi.* ITS. Surabaya.

Wijaya, William Nugroho. 2006. *Perancangan dan pembuatan aplikasi image segmentation menggunakan active contour (SNAKE)*. UK Petra, Surabaya.

Yani, Eli. 2005. *Pengantar Jaringan Saraf Tiruan.* MateriKuliah.com.