BAB III
PERANCANGAN SISTEM

Dalam pengerjaan perancangan dan pembuatan aplikasi pengenalan karakter alfanumerik JST algoritma *Hopfield* ini menggunakan software Borland Delphi 7.0.

3.1 Alur Proses Sistem

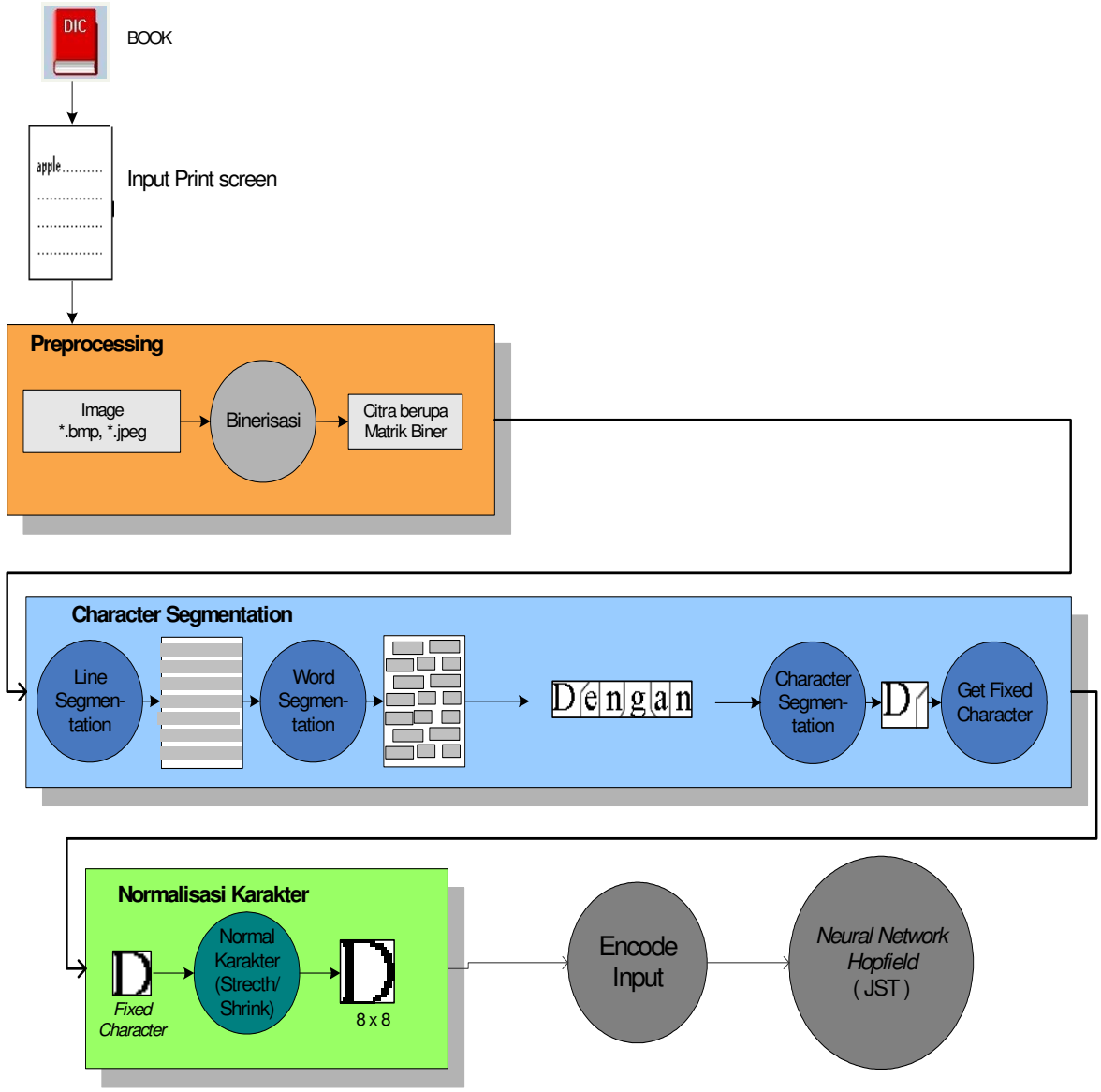
Rangkaian proses yang akan dilakukan oleh software ini adalah baca gambar, *grayscaling, thresholding,* segmentasi karakter, normalisasi hasil segmentasi, ekstrasi ciri *(encode input)* dan *neural network.* Proses *neural network* sendiri membutuhkan proses pelatihan *(training)* agar *output* yang dihasilkan benar dan akurat.

Jadi secara garis besar dapat dikatakan bahwa alur aplikasi ini dimulai dari diinputkannya sebuah gambar dari sebuah gambar teks. Di mana gambar tersebut akan melalui serangkaian manipulasi citra dalam proses *image preprocessing* diantaranya proses *grayscaling, thresholding,* segmentasi karakter, normalisasi dan ekstrasi ciri yang akan mengubah gambar tersebut menjadi serangkaian nilai-nilai yang berguna untuk proses utama berikutnya yaitu : proses pelatihan *(training process)* ataupun proses pengenalan *(recognition process).*

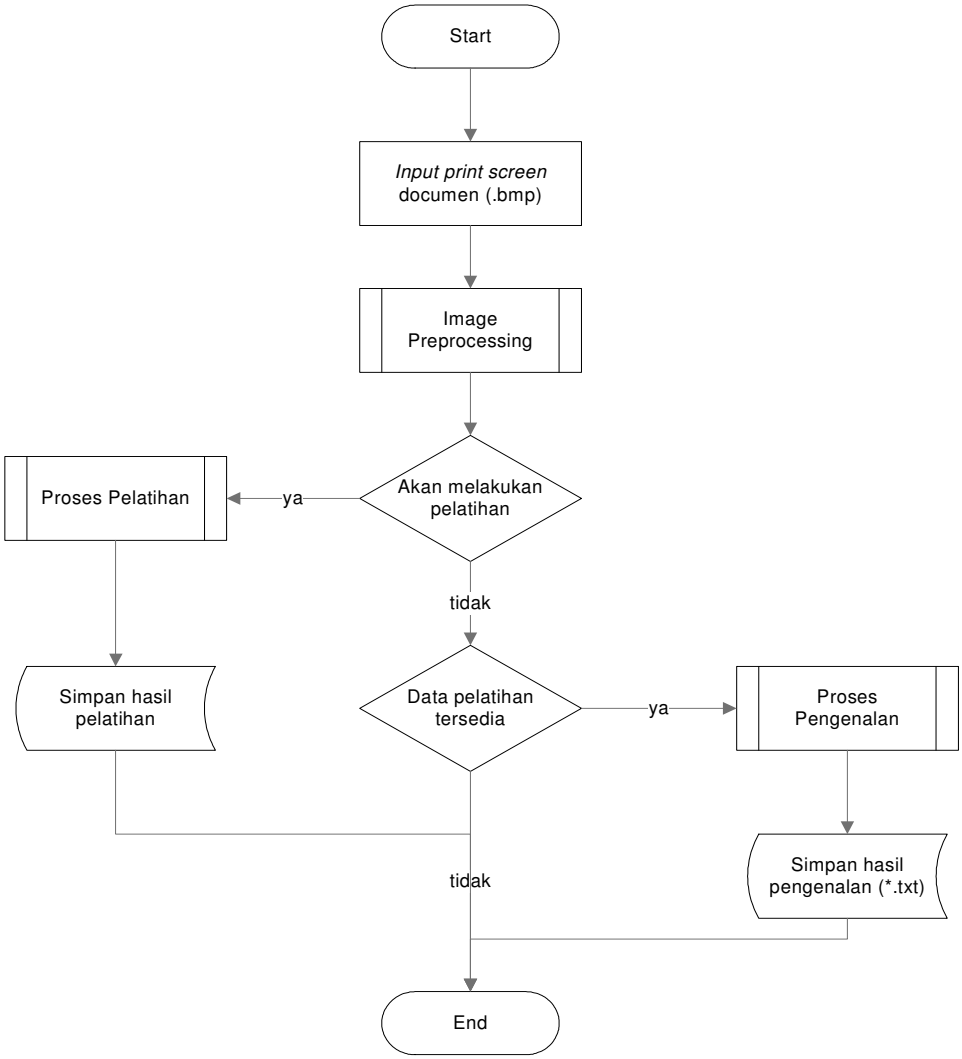
Output dari proses pelatihan adalah sekumpulan nilai bobot jaringan dan vektor pola stabil yang disimpan ke dalam sebuah file pelatihan, file pelatihan tersebut dapat dibuka kembali bila akan di*update* ataupun digunakan dalam proses pengenalan. Sedangkan *output* dari proses pengenalan adalah konversi dari citra yang berisi angka-angka *bipolar* menjadi huruf-huruf atau angka yang dapat disimpan ke dalam *file* teks.

Dalam hal ini perlu diperhatikan bahwa sebelum melakukan proses pengenalan, terlebih dahulu dilakukan pengecekan apakah sudah ada data pelatihan (*training)* yang dapat dijadikan acuan dalam melakukan pengenalan. Bila tidak, maka proses pengenalan tidak bisa dilakukan. Jadi dapat dilihat akan ada tiga modul utama dalam pembuatan aplikasi ini, yaitu: modul *image preprocessing,* modul pelatihan (*training),* dan modul pengenalan (*recognition)*. Modul pelatihan dan modul pengenalan sangat bergantung pada modul *image preprocessing,* dan modul pengenalan selain bergantung pada modul *image*

preprocessing juga bergantung pada kinerja modul pelatihan. Sehingga dapat dilihat bahwa urutan pengerjaan yang seharusnya dilakukan dalam pembuatan aplikasi ini secara berturut-turut membuat modul : *image preprocessing*, pelatihan *(training)*, dan pengenalan (*recognition).*



Gambar 3.1 Ilustrasi Proses Pengenalan *Hopfield*



Gambar 3.2 *Flowchart* Rencana Aplikasi

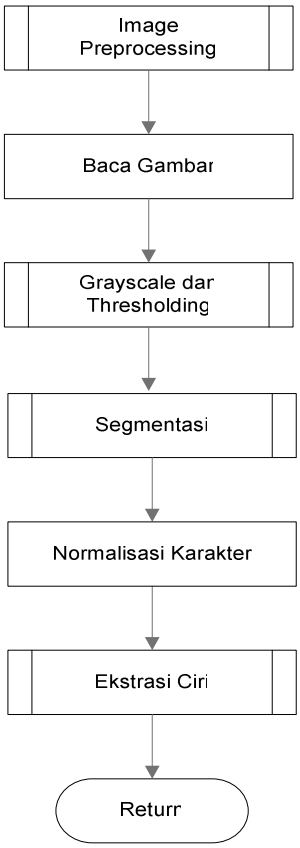
3.2 Proses Pengambilan Gambar

Proses pengambilan gambar yang digunakan sebagai *input* pada sistem yang dibangun adalah rangkaian-rangkaian kata yang berasal dari *file* Microsoft Word yang mewakili huruf dan angka yang sudah dilatihkan kepada sistem, kemudian dilakukan proses *print screen* dan diolah dengan menggunakan aplikasi *paint* sehingga didapat *file* *.*bmp* atau *.*jpg*.

3.3 Modul *Image Preprocessing*

Modul ini merupakan proses yang pertama kali dijalankan bila user hendak melakukan proses pelatihan maupun proses pengenalan. Pertama-tama, gambar yang dibuka akan dibaca dan kemudian setiap huruf dan angka tersebut akan dicari posisi dan ukurannya (segmentasi). Sebelum gambar tersebut disegmentasi terdapat proses *grayscaling, thresholding* hingga menjadi gambar hitam-putih untuk memudahkan dalam pengolahan data selanjutnya.

Proses selanjutnya adalah mencari baris demi baris yang ada pada gambar dan mensegmentasikan tiap kata dan tiap karakter yang ada pada gambar *input*. Proses pengambilan tiap-tiap karakter yang ada pada gambar ini sangatlah penting karena tanpa akurasi *input* yang baik, maka proses pengenalan (*recognition)* tidak akan berfungsi dengan baik. Proses ini ditunjukan oleh *flowchart* pada gambar 3.3:



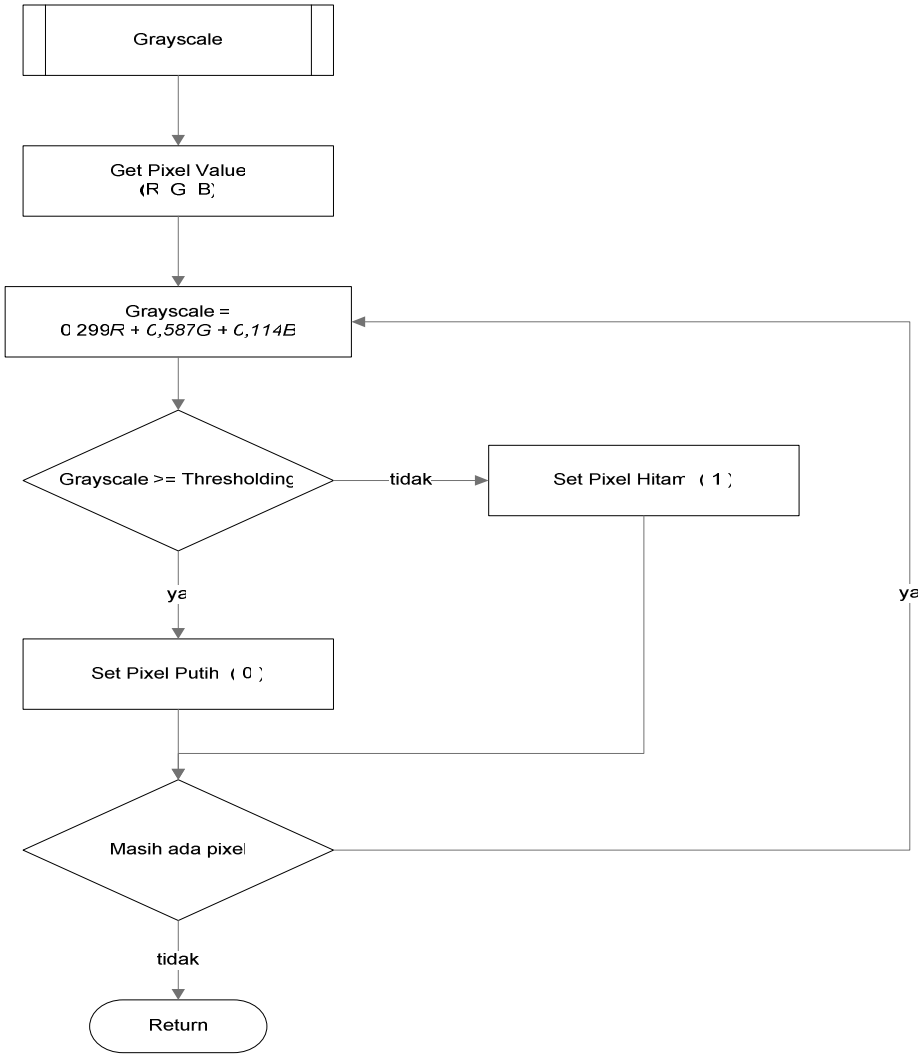
Gambar 3.3 *Flowchart* Modul *Image Preprocessing*

3.3.1 Proses *Grayscaling* dan *Thresholding*

Pada proses *grayscaling* ini gambar *input* yang berwarna dapat diubah menjadi gambar yang tediri dari warna putih dan gradiasi warna hitam dengan menggunakan representasi warna *RGB*. Pengubahan gambar ke dalam bentuk *grayscale* ini dilakukan dengan mengambil nilai *pixel* dari suatu gambar *input* yang kemudian dihitung dengan persamaan yang ada yaitu :

$$Grayscale = 0{,}299R + 0{,}587G + 0{,}114B$$

Persamaan di atas dipilih dari persamaan yang lain dikarenankan 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. 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.



Gambar 3.4 ***Flowchart*** **Proses** ***Grayscaling dan Thresholding***

Ilustrasi proses *grayscaling* dan *thresholding* **:**

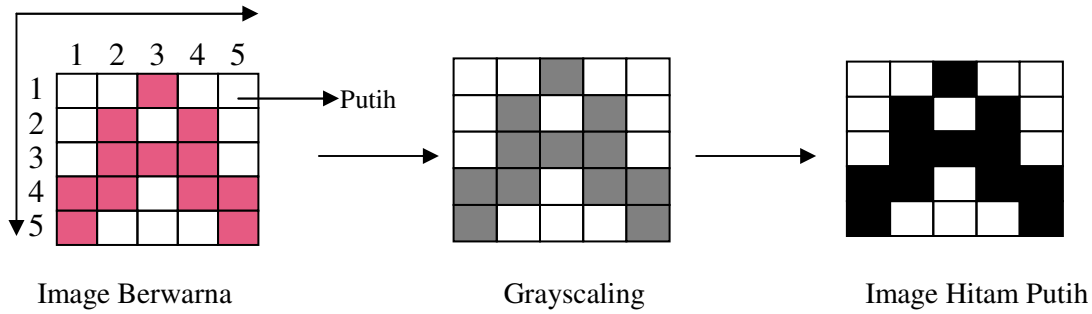


Image Berwarna

Red

255	255	230	255	255
255	230	255	230	255
255	230	230	230	255
230	230	255	230	230
230	255	255	255	230

Green

255	255	90	255	255
255	90	255	90	255
255	90	90	90	255
90	90	255	90	90
90	255	255	255	90

Blue

255	255	130	255	255
255	130	255	130	255
255	130	130	130	255
130	130	255	130	130
130	255	255	255	130

To Grayscale =

$(0.299R + 0.587G + 0.114B)$

Image Grayscale

255	255	136,4	255	255
255	136,4	255	136,4	255
255	136,4	136,4	136,4	255
136,4	136,4	255	136,4	136,4
136,4	255	255	255	136,4

Thresholding =

if P[x,y]>= θ *then*
P[x,y]:=0
else
P[x,y]:=1

Matrik Biner

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

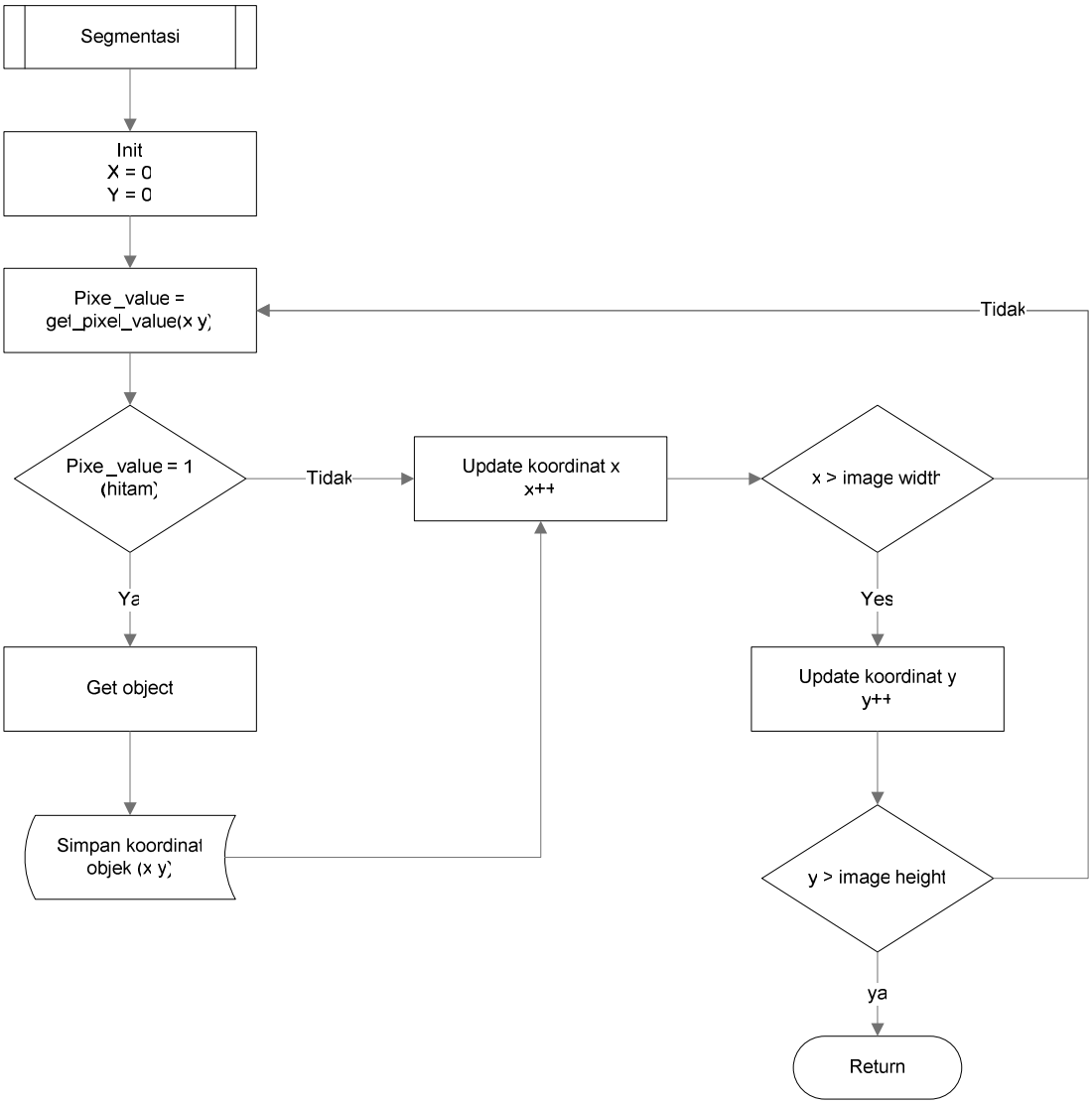
Gambar 3.5 Ilustrasi Proses *Grayscaling dan Thresholding*

Gambar *input* yang berformat *bitmap* akan diubah ke dalam sebuah matrik yang berukuran sesuai dengan ukuran *pixel* dari *input* image tersebut. Selanjutnya akan dilakukan perubahan ke dalam vektor matrik biner yang hanya bernilai 0 atau 1 pada setiap *pixel*-nya. Setelah didapatkan nilai *grayscale* kemudian nilai pada setiap pixel hasil *grayscale* akan dilakukan proses *thresholding* yang akan menyebabkan setiap *pixel* hanya bernilai 0 atau 1.

Setelah proses *grayscale* dan *thresholding* selesai, proses selanjutnya adalah melakukan proses segmentasi (*Segmentation).*.

3.3.2 Proses Segmentasi (*Segmentation)*

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.



Gambar 3.6 *Flowchart* Proses Segmentasi *(Segmentation)*

Dalam hal ini terjadi tiga proses segmentasi, yaitu :

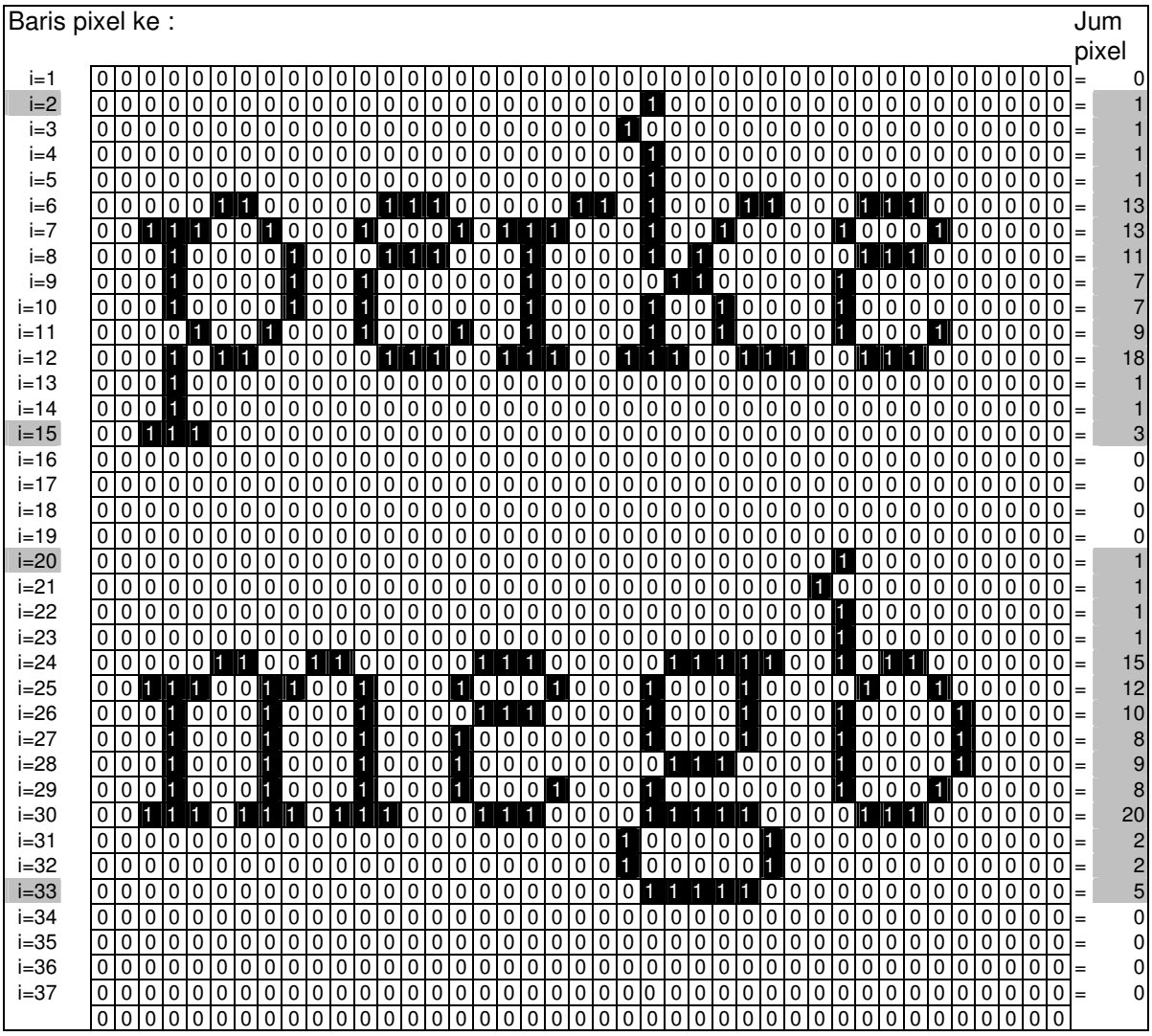
1. Segmentasi Baris (*Line Segmentation)*

   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 *pixel*nya.

Contoh:



Gambar 3.7 Ilustrasi Proses Segmentasi

Misal : (Berdasarkan gambar rangkaian karakter di atas)

Jumlah baris = 2

Baris ke – 1

- Posisi awal : i = 2
- Posisi akhir : i = 15
- Tinggi karakter = (Posisi akhir – Posisi awal +1)

Baris ke – 2

- Posisi awal : i = 20
- Posisi akhir : i = 33
- Tinggi karakter = (Posisi akhir – Posisi awal +1)

2. Segmentasi Kata (*Words Segmentation)*

   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 kata per-baris, posisi awal dan posisi akhir setiap kata terhadap sumbu-*x*. Pada sistem yang akan dibangun ini menggunakan acuan minimal spasi ≥ 5.

   Spasi = Jika ditemukan sederetan pixel putih ≥ acuan di atas

   Besarnya spasi antar kata sangat dipengaruhi oleh ukuran *font* yang digunakan serta kombinasi antara karakter akhir dari kata pertama dan karakter awal dari kata kedua. Dari proses ini maka akan didapatkan :

   - Jumlah kata setiap baris
   - Posisi awal setiap kata terhadap sumbu *x*
   - Posisi akhir setiap kata terhadap sumbu *x*

3. Segmentasi Karakter (*Character Segmentation)*

   Segmentasi karakter penelusurannya hampir sama dengan segmentasi kata, hanya saja pemisahan antar karakter dibatasi minimal 1 *pixel* saja.

Hasil dari proses segmentasi ini adalah posisi, lebar dan tinggi untuk masing-masing karakter.

3.3.3 Normalisasi Karakter

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 8 x 8 *pixel.* Dengan demikian jumlah area yang ada pada setiap sampel akan bersesuaian dengan jumlah *neuron input* jaringan syaraf tiruan yang akan digunakan.

3.3.4 Ekstraksi Ciri / *Encode Input*

Setelah didapatkan informasi matrik hasil proses normalisasi setiap karakter dalam dimensi matrik 8 x 8, 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 64) yang nantinya akan digunakan sebagai masukan JST.

Proses perubahann pola vektor biner (1 dan 0) ke vektor bipolar bisa juga menggunakan persamaan :

$$y = (2u_1 - 1, 2u_2 - 1, ..., 2u_n - 1) \qquad (21)$$

Keterangan :

y : *output* berupa nilai bipolar 1 atau -1

u_n : elemen ke *n* dari vektor *U*, di mana *U* adalah vektor pola biner 1 atau 0

Contoh :

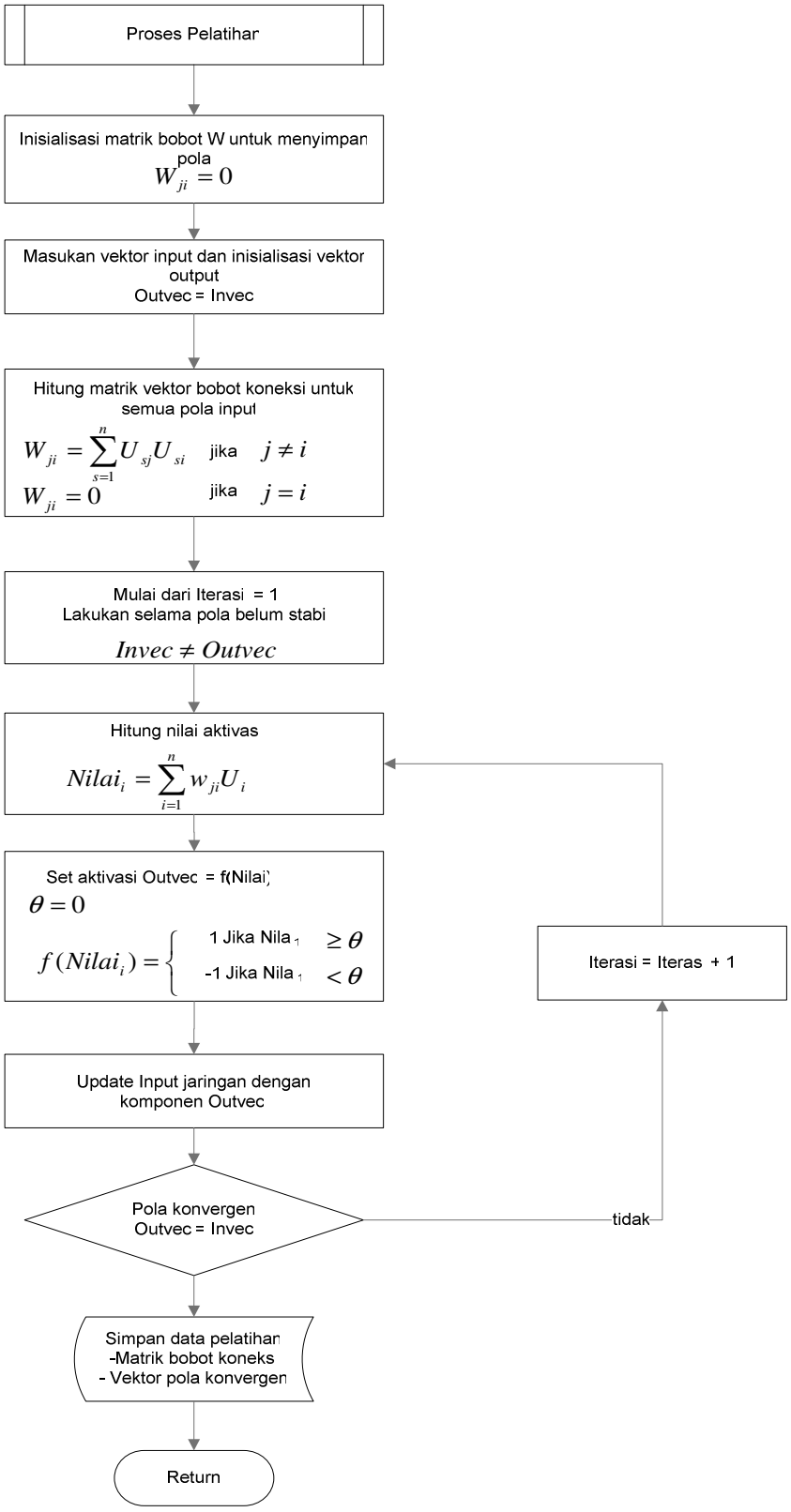
Misal untuk pola A mempunyai vektor biner 10011, maka vektor bipolar untuk pola A adalah :

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

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

3.4 Modul Pelatihan (*Training Process)*

Pada proses pelatihan ini, jaringan akan dilatih berdasarkan vektor *input* gambar huruf dan angka yang sudah disiapkan sebelumya untuk proses pelatihan. Di mana gambar *input* pelatihan sudah mengalami proses dari *image preprocessing* hingga *encode input* seperti yang sudah dijelaskan sebelumnya kecuali proses segmentasi. Dalam proses ini huruf dan angka dilatihkan satu per satu sesuai dengan *input* vektornya. Adapun *output* dari modul ini adalah file data informasi jaringan saraf tiruan yang berisi bobot koneksi antar *neuron* dari semua pola vektor yang terbentuk dan vektor pola yang dipanggil secara stabil (konvergen)
.



Gambar 3.8 *Flowchart* Proses *Neural Network* Pada Modul Pelatihan

Pertama vektor pola biner dinormalisasi dengan ukuran 8x8, kemudian vektor biner pola dirubah ke dalam bentuk *bipolar* yang mempunyai nilai 1 atau -1 kemudian dilakukan proses *encode input* yang menghasilkan matrik vektor 1x64, vektor inilah yang digunakan sebagai *input* pelatihan maupun *input* pengenalan. Sebelum proses pelatihan berjalan hal pertama yang perlu dilakukan adalah membuat sebuah matrik vektor bobot koneksi untuk menyimpan pola dengan kondisi awal $W_{ji} = 0$. Setelah itu tiap pola *input* vektor yang ada dihitung dengan persamaan $W_{ji} = \sum_{s=1}^{n} U_{sj} U_{si}$ jika $j \neq i$ dan $W_{ij} = 0$ jika j = i, untuk mendapatkan matrik bobotnya, persamaan tersebut menunjukan bahwa bobot – bobot pada matrik vektor *W* yang terletak pada diagonal utamanya adalah nol (0) yang menunjukan bahwa *neuron-neuron* pada jaringan *Hopfield* tidak memiliki hubungan dengan dirinya sendiri ($W_{ji} = 0$; 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. Untuk pola baru yang akan dilatihkan langkahnya sama yaitu membuat matrik bobot koneksi W_{ji} kemudian matrik bobot yang lama dijumlahkan dengan matrik bobot yang baru untuk mendapatkan bobot matrik koneksi yang bisa merepresentasikan tiap pola $W_{ji}^{(baru)} = W_{ji}^{(lama)} + U_j^{(baru)} U_i^{(baru)}$.

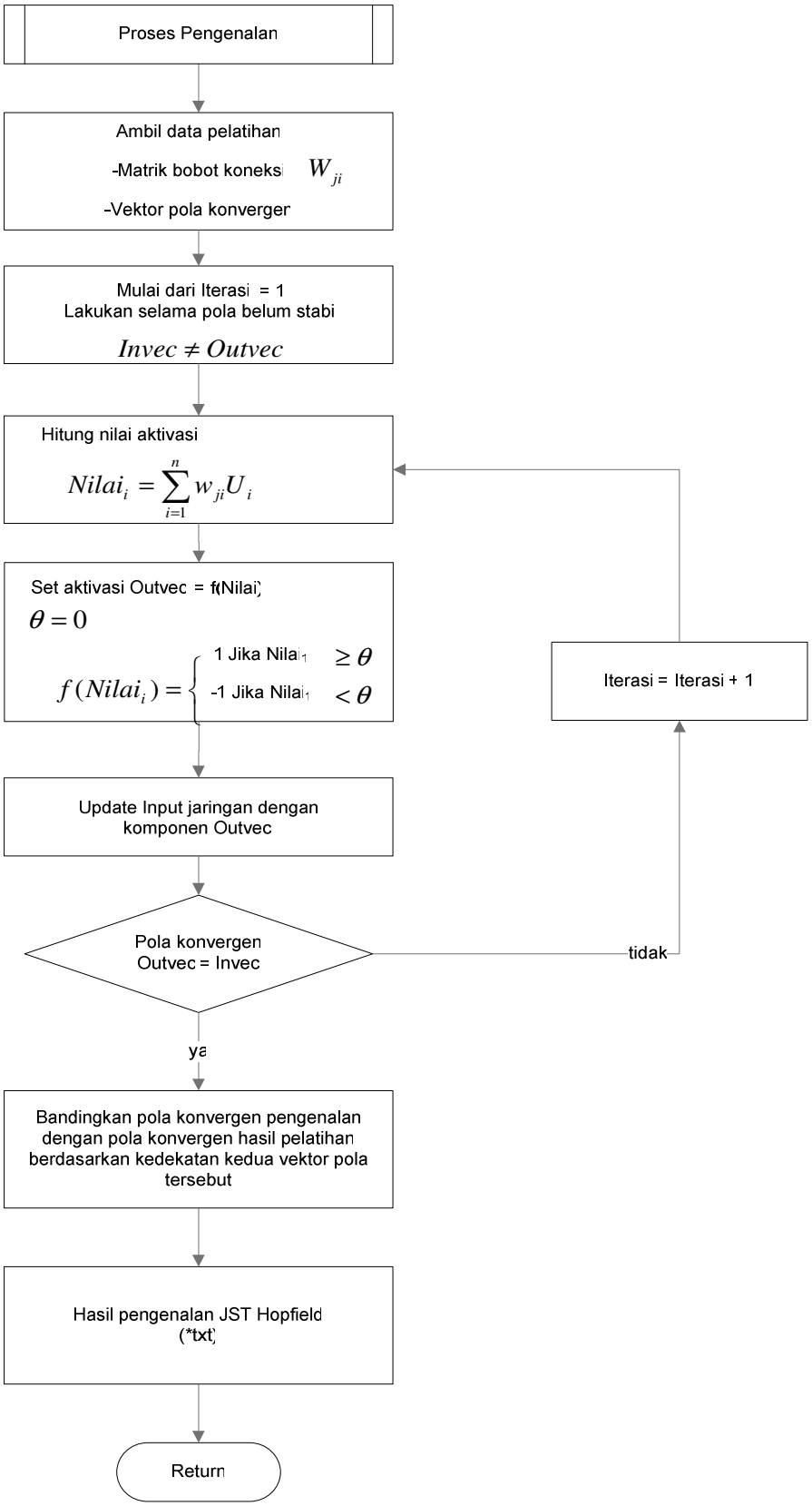
Setelah didapatkan matrik bobot koneksi langkah selanjutnya adalah menghitung nilai aktivasi tiap pola dengan persamaan $Nilai_i = \sum_{i=1}^{n} U_i w_{ji}$ untuk mendapatkan sinyal keluaran di mana ketika diaktivasi dengan fungsi *threshold* akan menghasilkan pola yang dimaksud. Proses ini akan dilakukan berulang-ulang sampai batas iterasi sambil terus memperbarui *node*-nya dengan menggunakan komponen $Outvec_i$ pada *neural network* yang digunakan sampai jaringan kovergen, konvergen disini dalam arti vektor pola tersebut dipanggil secara stabil. Pembaharuan nilai ini akan terus disimpan dan menjadi patokan untuk iterasi berikutnya sehingga menyebabkan jaringan syaraf tiruan yang ada akan menjadi makin akurat dari waktu ke waktu. Pembaharuan dalam JST *Hopfield* dilakukan secara asinkron (*asynchronous update)*, artinya pembaharuan tidak dilakukan secara bersamaan kesemua *output* yang diumpankan kembali sebagai *input*, melainkan hanya dilakukan pada satu

komponen vektor pada satu waktu yang kemudian diumpankan kembali sebagai *input* jaringan untuk iterasi selanjutnya.

Yang harus diajarkan secara manual kepada sistem *Hopfield neural network* ini adalah pemberian informasi terhadap setiap kelompok *output* yang terjadi. Karena bagaimanapun juga peran manusia sebagai pihak yang lebih dulu mengetahui informasi mengenai huruf cetak mutlak diperlukan.

3.5 Modul Pengenalan (*Recognition Process)*

Input dari modul pengenalan ini sebenarnya sama dengan *input* dari modul pelatihan yaitu berupa file gambar dokumen, hanya saja gambar *input* dalam proses pengenalan ini berupa gambar yang tersusun dari rangkaian kata dan tentunya proses segmentasi perlu dilakukan. *Input* vektor untuk proses pengenalan ini berupa matrik vektor pola 1x64. Selain *input* berupa vektor pola gambar, modul ini juga memerlukan data tambahan yaitu berupa nilai-nilai bobot koneksi jaringan dan vektor pola konvergen dari tiap karakter yang telah dilatihkan sebelumnya melalui modul pelatihan. Pada dasarnya pada modul pengenalan ini hanya membandingkan dan memanggil *output* suatu pola yang sudah dilatihkan sebelumnya.



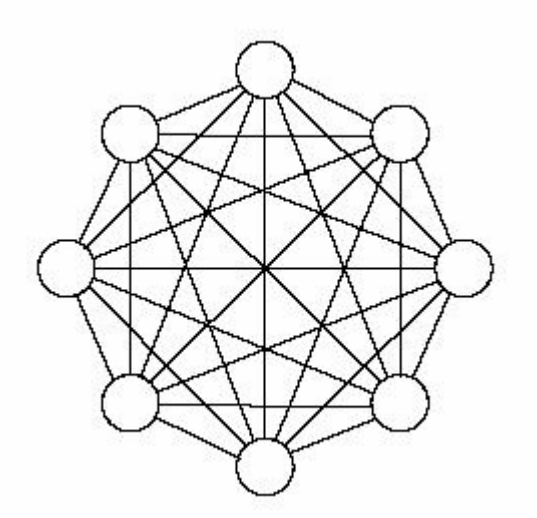
Gambar 3.9 *Flowchart* Proses *Neural Network* Pada Modul Pengenalan

Sesuai dengan sifat JST *Hopfield* yaitu *auto-associative* ini berarti JST *Hopfield* bisa berperilaku seperti memori yaitu bisa menyimpan informasi yang pernah diberikan dan menampilkan kembali informasi tersebut apabila diminta. Dikarenakan pada proses karakter ini sangat tergantung pada proses segmentasi yang baik, apabila terdapat proses segmentasi yang tidak *valid* sehingga pola yang diberikan tidak persis seperti salah satu pola dalam pembelajaran maka JST *Hopfield* mampu memilih salah satu pola dalam memorinya yang paling mirip ini dikarenakan JST *Hopfield* juga besifat *high-fault tolerance* dengan asumsi pemilihan *output* pengenalan ini adalah *output* yang sesuai dengan kedekatan vektor pola *input* terhadap pola target. Proses segmentasi yang kurang baik juga bisa menyebabkan JST memanggil pola palsu (*spurious stable state)* di mana jaringan berkonvergensi terhadap pola lain.

3.6 Desain Neural Network

Jaringan *Hopfield* merupakan jaringan *associative memory* di mana jaringan tersebut bisa mengingat pola yang telah tersimpan. Topologi jaringan *Hopfield* merupakan jaringan di mana setiap *node* saling terhubungan penuh. JST ini terdiri atas satu lapis (*layer*) dan terdiri dari beberapa *neuron*. Lebih lanjut, masing-masing *neuron* terhubung satu sama lain. Artinya, seluruh *neuron* itu berfungsi sebagai masukan (*input*), tetapi sekaligus sebagai keluaran (*output*).Setiap *node* dalam JST *Hopfield* memiliki bobot koneksi yang simetris, akan tetapi pada JST *Hopfield* setiap *node*nya tidak memiliki hubungan dengan dirinya sendiri, hal ini memenuhi $W_{ij} = W_{ji}$ (di mana i = baris dan j = kolom) untuk $i \neq j$ dan $W_{ij} = 0$ untuk $i = j$.

Secara umum, nilai keluaran JST ditentukan oleh nilai masukan, bobot koneksi (*weight*), dan fungsi aktivasi. Bentuk fungsi aktivasi dipilih berdasarkan masalah yang akan diselesaikan. Dalam program ini fungsi aktivasi yang digunakan yaitu fungsi *bipolar* dengan *threshold (threshold function)* yang mempunyai *range* -1 atau 1.



Gambar 3.10 Topologi Jaringan *Hopfield*

Jumlah *neuron* dalam sistem JST adalah *nx* kali *ny pixel*. Dengan melihat gambar dari topologi jaringan *Hopfield* maka jumlah *neuron input* dan *neuron output* akan sama, pada program ini memakai 64 buah *neuron* hal ini dilihat dari jumlah karakter yang akan dikenali oleh program ini yaitu sebanyak 62 karakter. Pemilihan jumlah neuron yang dipakai yaitu 64 buah, dikarenakan matriks vektor untuk setiap pola haruslah sama baris dan kolomnya (*symetric).* Dengan melihat jumlah variasi ouput yang akan dikenali yaitu sebanyak 62 objek maka digunakan sebuah *array* dua dimensi 8 x 8 dengan anggapan ukuran tersebutlah yang paling mendekati. Dikarenakan pada JST *Hopfield* tiap *node*-nya mempunyai bobot koneksi (tidak dengan dirinya sendiri) sehingga jumlah bobot koneksi yang terbentuk adalah 64x64 = 4096 bobot, matrik bobot koneksi yang terbentuk juga simetris antara baris dan kolomnya.

Adapun kumpulan *node output* dari jaringan saraf tiruan yang ada dapat digambarkan sebagai berikut :

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 0

Gambar 3.11 Karakter-Karakter *Arial* Yang Menjadi *Input dan Output* Dalam JST

Setiap sampel yang diamati dibagi ke dalam sebuah *array* dua dimensi 8 x 8 untuk mewakili karakter-karakter yang akan dikenali kemudian di-*encode* sehingga menghasilkan vektor *input* 1x64 yang digunakan sebagai masukan baik proses pelatihan maupun pengenalan. Fungsi aktivasi yang digunakan dalam jaringan ini adalah fungsi aktivasi *bipolar* dengan *threshold*

Input pola yang disimpan oleh jaringan adalah dimensi vektor *n*, di mana *n* adalah jumlah *neuron* dalam jaringan. Setelah inisialisasi aktivasi ditentukan pada *neuron*, jaringan akan mulai menghitung pada setiap iterasi sampai kondisi stabil tercapai yang artinya jaringan dianggap bisa mengenali dan mengingat pola yang dilatihkan. *Output* dari jaringan adalah vektor dari nilai aktivasi *neuron* pada kondisi stabil.

Jadi ketika jaringan yang sudah dilatih diberikan *input* karakter, JST *Hopfield* akan secara otomatis melakukan pengelompokan (*clustering)* sehingga nilai-nilai *input* yang serupa akan mengelompok pada *node output* yang berdekatan atau sama.

Sebagai ilustrasi, JST dengan struktur empat *neuron* diberikan pembelajaran untuk mengingat pola biner 1 0 1 0 dan 0 1 0 1. Dari pola masukan ini bisa dibentuk nilai-nilai bobot yang akan memberikan keluaran yang dikehendaki. Penentuan nilai-nilai bobot ini merupakan bagian dari proses pembelajaran JST.

Pada saat pengetesan, bila JST itu diberi masukan pola 1 0 1 0, akan dikeluarkan jawaban pola 1 0 1 0. Jika pola masukan adalah 0 1 0 1, maka dikeluarkan pola yang sama, yaitu 0 1 0 1. Ini mencerminkan sifat *auto-associative*. Apabila pola yang diberikan tidak persis seperti salah satu pola dalam pembelajaran maka JST *Hopfield* mampu memilih salah satu pola dalam memorinya yang paling mirip. Misalnya, diberikan masukan pola 0 1 0 0, maka keluarannya adalah 0 1 0 1, yaitu salah satu pola yang pernah dilatihkan. Sifat seperti ini disebut sebagai *high-fault tolerance* berdasarkan kedekatan antara vektor pola *input* dengan vektor pola target.

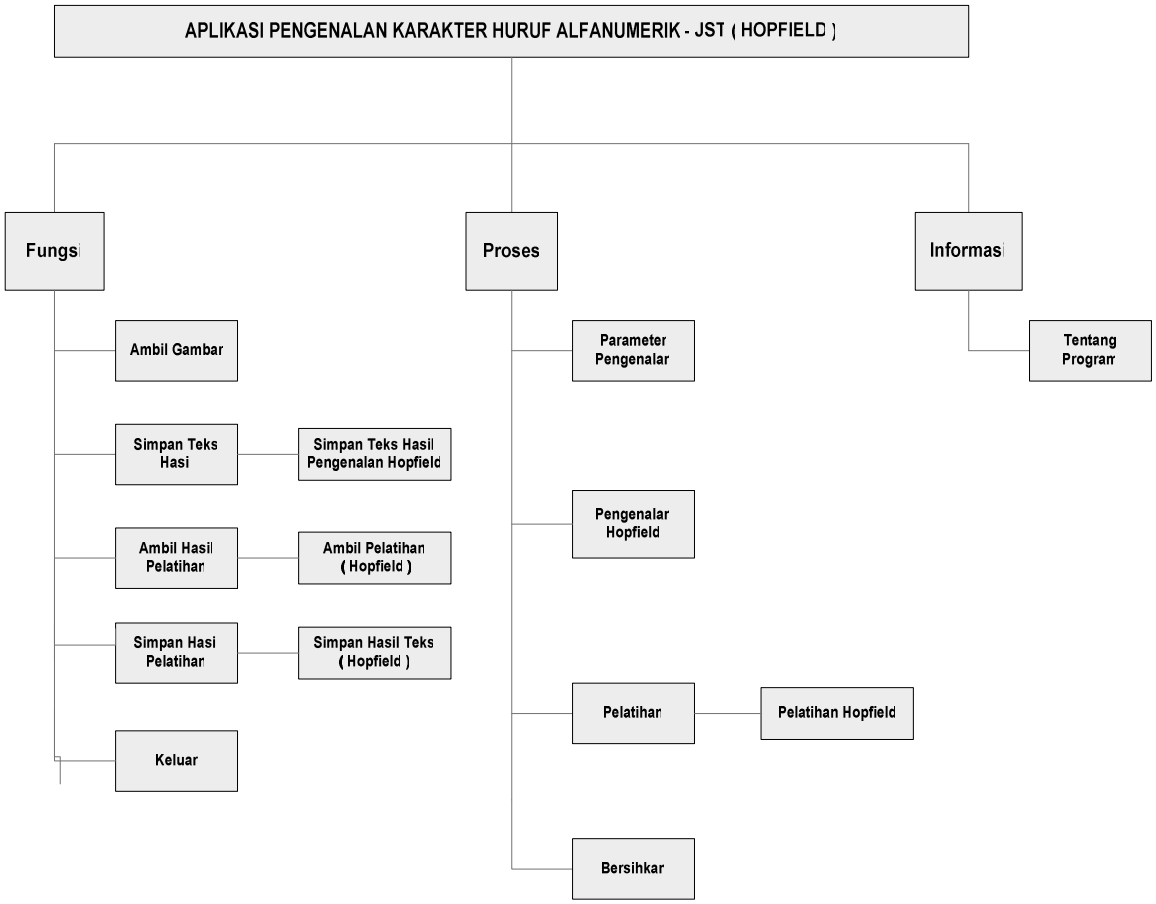
3.7 Desain Menu

Sudah dijelaskan sebelumnya, secara umum alur proses keseluruhan sistem dibagi menjadi tiga yaitu : modul *image preprocessing,* modul pelatihan dan modul pengenalan. Tetapi dalam desain menu program yang dibuat hanya terdapat menu untuk modul pelatihan (*training),* modul pelatihan (*recognition)* dan menu *option* untuk mengubah-ubah

parameter dalam jaringan. Dalam perancangan ini proses pada modul *image preprocessing* tidak dihilangkan, melainkan digabung dengan modul pelatihan dan pengenalan.

Tahapan perancangan dalam pembuatan suatu program merupakan tahapan yang sangat penting karena tahapan ini akan menentukan apakah suatu program akan memiliki kontruksi yang baik. Desain program yang dibuat harus mudah untuk digunakan, dapat dipahami dan dimengerti oleh pemakai ataupun orang lain, serta informasi yang dihasilkan dari program tersebut dapat dimengerti.

Menu pada perangkat lunak digunakan untuk mempersingkat suatu perintah yang terdapat dalam bahasa pemrograman. Dengan adanya menu maka akan memudahkan dalam menampilkan suatu jendela (*windows*) pada perangkat lunak. Adapun struktur menu untuk perangkat lunak terlihat pada diagram berikut:



Gambar 3.12 Perancangan Menu Utama Program

Menu yang dirancang pada perangkat lunak ini terdiri dari 3 menu utama, yaitu :

1. Fungsi, terdiri dari submenu ambil gambar teks, simpan teks hasil pengenalan, ambil hasil pelatihan, simpan hasil pelatihan dan keluar.
2. Proses, terdiri dari submenu parameter pengenalan, pengenalan Hopfield, pelatihan, bersihkan.
3. Informasi merupakan keterangan singkat aplikasi serta identitas dari pembuat program.