SKRIPSI

APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES PENGENALAN GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN *BACKPROPAGATION*

Disusun Oleh :

ICHWAN SETIAWAN

05.12.509

JURUSAN TEKNIK ELEKTRO S-1

KONSENTRASI TEKNIK KOMPUTER DAN INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

INSTITUT TEKNOLOGI NASIONAL MALANG

2010

LEMBAR PERSETUJUAN

APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES IDENTIFIKASI GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

SKRIPSI

Disusun dan Diajukan sebagai Salah Satu Syarat Untuk Memperoleh
Gelar Sarjana Teknik Komputer Dan Informatika Strata Satu (S–1)

Disusun Oleh :

ICHWAN SETIAWAN

NIM : 05.12.509

Diperiksa dan Disetujui
Dosen Pembimbing

Irmalia Suryani Faradisa, ST, MT
NIP. Y.1030000365

Mengetahui,
Ketua Jurusan Teknik Elektro S-1

Ir. Yusuf Ismail Nakhoda, MT
NIP. Y. 1039500274

JURUSAN TEKNIK ELEKTRO S-1
KONSENTRASI TEKNIK KOMPUTER DAN INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI NASIONAL MALANG
2010

APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES IDENTIFIKASI GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

ICHWAN SETIAWAN
(05.12.509)

Konsentrasi Komputer dan Informatika, Jurusan Teknik Elektro
Fakultas Teknologi Industri
Institut Teknologi Nasional Malang
Jln. Raya Karanglo Km 2 Malang
Email: ichwan.setiawan@rocketmail.com

Abstraksi

Jaringan Syaraf Tiruan adalah paradigma pemrosesan suatu informasi yang terinspirasi oleh sistem sel syaraf biologi, sama seperti otak yang memproses suatu informasi. Pembuatan struktur jaringan syaraf tiruan diilhami oleh jaringan otak manusia. Pada jaringan otak manusia terdapat sel syaraf Neuron yang memiliki tiga komponen penyusun yang saling bekerja sama untuk mengolah sinyal-sinyal informasi. Tiga komponen tersebut adalah dendrit(input), soma atau badan sel dan axon(output). Aplikasi Pengenalan Golongan Darah adalah hasil dari implementasi metode jaringan syaraf tiruan tersebut, aplikasi ini dapat mengenali golongan darah manusia sesuai dengan data pelatihan yang diajarkan padanya melalui pengenalan pola pembekuannya. Pola pembekuan empat golongan darah telah dikenali dan dapat dibedakan dengan baik oleh ahlinya. Jaringan Syaraf Tiruan dengan metode pembelajaran backpropagation digunakan sebagai pembelajaran seorang ahli untuk mengenali golongan darah melalui pola pembekuan masing-masing golongan darah tersebut setelah diberi reagent antigen.

Kata Kunci: Jaringan Syaraf Tiruan, Golongan Darah, Backpropagation.

KATA PENGANTAR

Dengan mengucap syukur kehadirat ALLAH SWT yang telah memberi Rahmat, Hidayah serta Innayah-Nya sehingga penulis dapat menyelesaikan laporan skripsi dengan judul:

"APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES IDENTIFIKASI GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION"

Penulisan skripsi ini disusun guna memenuhi syarat akhir kelulusan pendidikan jenjang Strata–1 di Institut Teknologi Nasional Malang. Dalam penyusunan skripsi ini penulis banyak mendapat bantuan baik moril maupun materiil, saran dan dorongan semangat dari berbagai pihak, untuk itu penulis mengucapkan terima kasih kepada :

1. Bapak Prof. Dr. Ir. Abraham Lomi, MSEE., selaku Rektor ITN Malang.
2. Bapak Ir. Sidik Noertjahyono, MT., selaku Dekan Fakultas Teknologi Industri.
3. Bapak Ir. Yusuf Ismail Nakhoda, MT., selaku Ketua Jurusan Teknik Elektro S–1 ITN Malang
4. Ibu Irmalia Suryani Faradisa, ST, MT., selaku Dosen Pembimbing.
5. Semua pihak yang telah membantu penulis dalam menyelesaikan skripsi ini yang tidak bisa penulis sebutkan satu persatu.

Penulis menyadari bahwa laporan ini masih banyak yang perlu disempurnakan. Oleh sebab itu kritik dan saran yang membangun sangat diharapkan.

Akhir kata, penulis mohon maaf kepada semua pihak bilamana selama penyusunan skripsi ini penyusun membuat kesalahan secara tidak sengaja dan semoga skripsi ini dapat bermanfaat bagi kita semua. Amin.

Malang, Agustus 2010

Penulis

DAFTAR ISI

DAFTAR GAMBAR

BAB II LANDASAN TEORI

BAB III PERANCANGAN SISTEM

BAB IV IMPLEMENTASI DAN PENGUJIAN SISTEM

BAB I
PENDAHULUAN

1.1. Latar Belakang

Perkembangan komputer dewasa ini telah mengalami banyak perubahan yang sangat pesat, seiring dengan kebutuhan manusia yang semakin banyak dan kompleks. Komputer yang pada awalnya hanya digunakan oleh para akademisi dan militer, kini telah digunakan secara luas di berbagai bidang, misalnya: Bisnis, Kesehatan, Pendidikan, Psikologi, Permainan dan sebagainya. Hal ini mendorong para ahli untuk semakin mengembangkan komputer agar dapat membantu kerja manusia atau bahkan melebihi kemampuan kerja manusia.

Dengan kecerdasan buatan, suatu komputer mampu menirukan seorang dokter yang dengan keahliannya dapat membedakan golongan darah manusia antara A, B, AB, dan O. Dengan pendekatan kecerdasan buatan, jaringan syaraf tiruan berusaha menirukan bagaimana pola-pola dibentuk[8].

Tekstur citra golongan darah yang unik dapat dianalisis untuk diidentifikasi. Oleh karena itu, diperlukan sebuah sistem yang mampu menganalisa karakteristik golongan darah sehingga mempermudah dalam mengidentifikasi golongan darah seseorang. Dengan menggunakan jaringan saraf tiruan metode perambatan-balik (*backpropagation*) dapat dibuat sebuah sistem yang mampu menganalisis dan mengidentifikasi golongan darah seseorang[8].

Jaringan syaraf tiruan algoritma *backpropagation* memiliki lapisan lapis jamak (*Multi Layer Net*). Pada umumnya jaringan syaraf tiruan dengan lapisan lapis banyak ini dapat menyelesaikan permasalahan yang lebih sulit daripada jaringan dengan lapisan tunggal[5].

1.2. Rumusan Masalah

Berdasarkan latar belakang maka timbul suatu permasalahan bagaimana membangun dan mengembangkan suatu aplikasi Pengenalan Golongan Darah untuk Proses Identifikasi Golongan Darah.

1.3. Tujuan

Menghasilkan desain aplikasi Pengenalan Golongan Darah untuk Mempercepat Identifikasi Golongan darah Menggunakan Metode Jaringan Syaraf Tiruan *Backpropagation.*

1.4. Batasan Masalah

Agar permasalahan mengarah sesuai dengan tujuan maka pembahasan dibatasi pada hal-hal sebagai berikut :

1. Aplikasi hanya digunakan untuk mengidentifikasi golongan darah manusia.
2. Tidak membahas kesehatan maupun penyakit dalam darah.
3. Sumber pengetahuan jaringan syaraf tiruan backpropagation dan golongan darah diperoleh dari pakar, *e-book* dan buku-buku yang mendukung.
4. Program dibuat menggunakan bahasa pemrograman Delphi 7.0.

1.5. Manfaat Penelitian

Manfaat penelitian ini adalah :

1. Mahasiswa dapat memperoleh pengalaman langsung dalam pembuatan aplikasi, dalam hal ini menggunakan *program builder* Delphi 7
2. Mahasiswa dapat membandingkan ilmu yang telah diperoleh selama perkuliahan dengan penerapan di lapangan kerja dan melakukan suatu pengembangan.

1.6. Metode Penelitian

Metode yang digunakan dalam penyusunan skripsi ini adalah :

1. Studi literatur,

   Yaitu tinjauan pustaka untuk mempelajari teori-teori yang terkait dan berhubungan dengan permasalahan melalui media Referensi Buku, dan Literatur dari Internet.

2. Perancangan

   Merancang dan mengembangkan desain Aplikasi Pengenalan Golongan Darah sesuai dengan literature yang ada.

3. Pembuatan Program

   Pembuatan Aplikasi Pengenalan Golongan Darah Dengan Metode Jaringan Syaraf Tiruan *Backpropagation.*

4. Implementasi & Pengujian

   Merancang dan mengembangkan desain Desain Aplikasi berdasarkan data-data yang diperoleh.

1.7. Sistematika Penulisan

BAB I : PENDAHULUAN

Berisi Latar Belakang, Perumusan Masalah, Tujuan dan Manfaat Penelitian, Pembatasan Permasalahan, Metodelogi Penelitian dan Sistematika Penulisan.

BAB II : LANDASAN TEORI

Berisi tentang landasan teori mengenai permasalahan yang berhubungan dengan pembahasan yang dilakukan, yang di dalamnya memuat teori-teori tentang Pengenalan Golongan Darah, Metode *Backpropagation* dan teori-teori yang mendukung.

BAB III : DESAIN DAN IMPLEMENTASI SISTEM

Bab ini membahas Desain dan analisis dari Aplikasi Pengenalan Golongan Darah pada hardware dan software.

BAB IV : ANALISA DAN PENGUJIAN SISTEM

Berisi tentang Implementasi dari Hasil Desain Aplikasi Pengenalan Golongan Darah.

BAB V : PENUTUP

Berisi tentang kesimpulan yang dapat diambil berdasarkan hasil uraian pada bab–bab sebelumnya dan saran mengenai hasil yang telah diperoleh.

BAB II
LANDASAN TEORI

2.1. Pengolahan Citra

Image Processing atau sering disebut pengolahan citra digital merupakan suatu proses *filter* gambar asli menjadi gambar lain sesuai dengan keinginan kita. Misalnya, kita mendapatkan suatu gambar yang terlalu gelap. Dengan *image Processing*, kita dapat memprosesnya agar mendapatkan gambar yang lebih jelas. [4]

Pada awalnya pengolahan citra ini dilakukan untuk memperbaiki kualitas citra, namun dengan berkembangnya dunia komputasi yang ditandai dengan semakin meningkatnya kapasitas dan kecepatan proses komputer, serta munculnya ilmu-ilmu komputasi yang memungkinkan manusia dapat mengambil informasi dari suatu citra, maka *image processing* tidak dapat dilepaskan dengan bidang *computer vision.*

Sesuai dengan perkembangan *computer vision* itu sendiri, pengolahan citra mempunyai dua tujuan utama yakni sebagai berikut.:

1. Memperbaiki kualitas citra, dimana citra yang dihasilkan dapat menampilkan informasi secara jelas atau dengan kata lain manusia dapat melihat informasi yang diharapkan dengan menginterpretasikan citra

yang ada. Dalam hal ini interpretasi terhadap informasi yang ada tetap dilakukan oleh manusia (*human preception*).

2. Mengekstraksi infomasi ciri yang menonjol pada suatu citra, dimana hasinya adalah informasi citra dimana manusia mendapatkan informasi ciri dari citra secara numerik atau dengan kata lain komputer (mesin) melakukan interpretasi terhadap informasi yang ada pada citra melalui besaran-besaran data yang dapat dibedakan secara jelas (besaran-besaran ini berupa besaran numerik).

Adapun pengertian pengolahan citra yang lain adalah pemrosesan citra, khususnya dengan menggunakan komputer, menjadi citra yang kualitasnya lebih baik. Umumnya, operasi-operasi pada pengolahan citra diterapkan pada citra bila :

1. Perbaikan atau memodifikasi citra perlu dilakukan untuk meningkatkan kualitas penampakan atau untuk menonjolkan beberapa aspek informasi yang terkandung di dalam citra.
2. Elemen di dalam citra perlu dikelompokkan, dicocokkan, atau diukur.
3. Sebagian citra perlu digabung dengan bagian citra yang lain.

Meskipun sebuah citra kaya informasi, namun seringkali citra yang kita miliki mengalami penurunan mutu (degradasi), misalnya mengandung cacat atau derau (*noise*), warnanya terlau kontras, kurang tajam, kabur (*blurring*), dan sebagainya. Citra semacam ini menjadi lebih sulit di

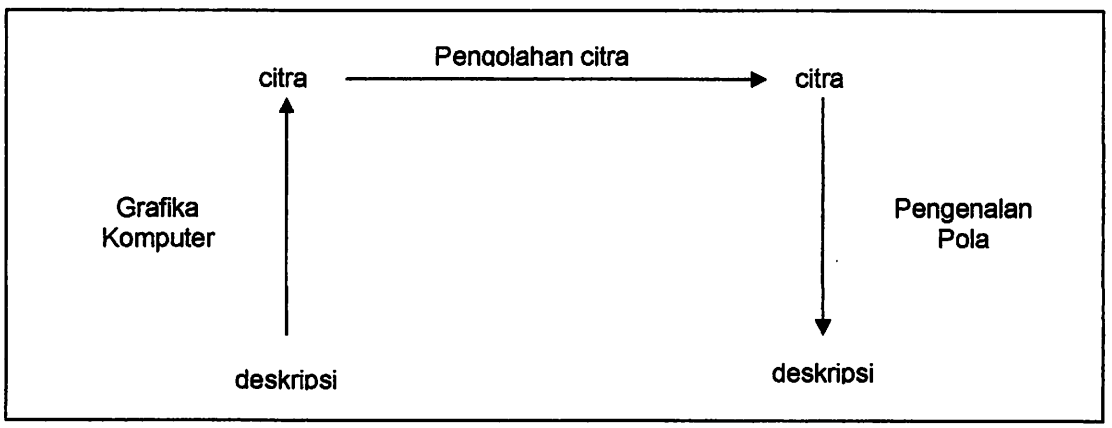
interpretasi karena informasi yang disampaikan oleh citra tersebut menjadi berkurang.

Agar citra yang mengalami gangguan mudah diinterpratasi (baik oleh manusia maupun mesin), maka citra tersebut perlu dimanipulasi menjadi citra lain yang kualitasnya lebih baik.

Dalam bidang komputer, sebenarnya ada 3 bidang *study* yang berkaitan dengan data citra, namun tujuan ketiganya berbeda, yaitu :

1. grafika computer (*computer graphics*)
2. pengolahan citra (*image Processing*)
3. Pengenalan pola (*Pattern Recognition/image interpretation*)

Hubungan antara ketiga bidang grafika computer, pengolahan citra, pengenalan pola dapat kita lihat dalam gambar 2.1 di bawah ini.



Gambar 2.1 Tiga bidang study yang berkaitan dengan citra
Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

Grafika komputer bertujuan menghasilkan citra (lebih tepat disebut grafik atau *picture*) dengan geometri seperti garis, lingkaran, dan sebagainya. Primitif-primitif geometri tersebut memerlukan data deskriptif untuk melukis elemen-elemen gambar. Contoh data deskriprtif adalah koordinat titik, panjang garis, jari-jari lingkaran, tebal garis, warna, dan sebagainya. *Grafika computer* memainkan peranan penting dalam visualisasi dan *virtual reality*.

Gambar 2.2 Alur proses Grafika computer
Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

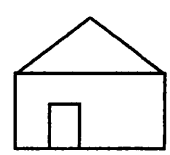
Contoh grafika komputer misalnya menggambar sebuah 'rumah' yang dibentuk oleh garis-garis lurus, dengan data masukan berupa kondinat awal dan kordinat ujung garis.

```
Program :

Line (0, 0, 0, 40)
Line (0, 40, 60, 60)
Line (40, 60, 80, 40)
Line (0, 40, 80, 40)
Line (80, 40, 80, 0)
Line (80, 0, 0, 0)
Line (20, 0, 25, 25)
Line (25, 25, 35, 25)
Line (35, 25, 35, 0)
```

(a)

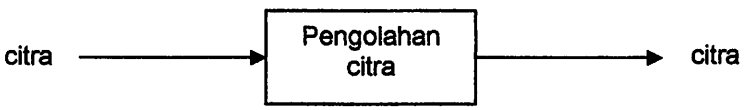
Gambar hasil



(b)

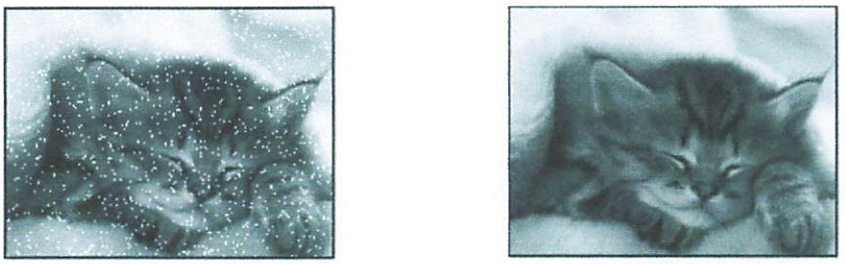
Gambar 2.3 Program grafika komputer untuk membuat gambar rumah
Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

Pengolahan citra bertujuan memperbaiki kualitas citra agar mudah di interpretasi oleh manusia atau mesin (dalam hal ini komputer). Teknik-teknik pengolahan citra mentransformasikan citra menjadi citra lain. Jadi, masukannya adalah citra dan keluarannya adalah juga citra, namun citra keluaran mempunyai kualitas lebih baik daripada citra masukan. Termasuk dalam bidang juga adalah pemampatan citra (*image compression*).



Gambar 2.4 Alur Proses Pengolahan Citra
Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

Pengubahan kontras citra dari gambar gelap menjadi terang dan jelas merupakan contoh operasi pengolahan citra. Contoh operasi pengolahan citra lainnya adalah penghilangan derau (*noise*) pada suatu gambar. Seperti pada gambar 2.5 yang disebelah kiri mengandung derau berupa bintik-bintik putih. Dengan operasi penapisan (*filtering*), derau pada citra masukan dapat dikurangi sehingga dihasilkan citra yang kualitasnya lebih baik.

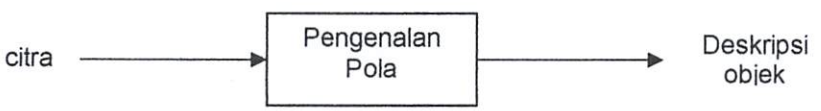


(a) (b)

Gambar 2.5 (a) Citra yang mengandung derau, (b) Hasil dari operasi penipisan derau

Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

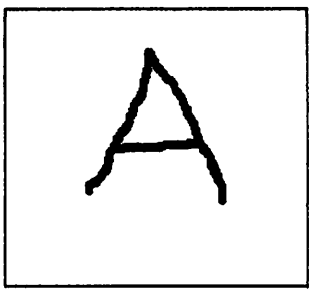
Pengenalan pola mengelompokkan data numerik dan simbolik (termasuk citra) secara otomatis oleh mesin (komputer). Tujuan pengelompokkan adalah untuk mengenali suatu objek di dalam citra. Manusia bisa mengenali objek yang dlihatnya karena otak manusia telah belajar mengklasifikasi objek-objek di alam sehingga mampu membedakan suatu objek dengan objek lainnya. Kemampuan sistem visual manusia ini yang dicoba ditiru oleh komputer. Komputer menerima masukan berupa citra objek yang akan di identifikasi, memproses citra tersebut, dan memberikan keluaran berupa deskripsi objek di dalam citra.



Gambar 2.6 Alur proses pengenalan pola.

Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

Contoh pengenalan pola misalnya citra dalam gambar 2.7 adalah tulisan tangan yang digunakan sebagai data masukan untuk mengenali karakter 'A'. Dengan menggunakan suatu algoritma pengenalan pola, diharapkan komputer dapat mengenali bahwa karakter tersebut adalah 'A'. [3]



Gambar 2.7 Citra karakter 'A' yang digunakan sebagai masukan untuk pengenalan huruf.
Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

2.1.1. Citra

Secara harafiah, citra (*image*) adalah gambar pada bidang dwi matra (dua dimensi) ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*kontinyu*) dari intensitas cahaya pada bidang dwi matra. Sumber cahaya menerangi objek, objek memantulkan kembali sebagian dari berkas cahaya tersebut. Pantulan cahaya ini ditangkap oleh alat-alat *optic*, misalnya pada mata manusia, kamera, pemindai (*scanner*), dan sebagainya, sehingga bayangan objek yang disebut citra tersebut terekam.

Citra sebagai keluaran dari suatu sistem perekaman data dapat bersifat :

1. Optik berupa foto,
2. Analog berupa sinyal video seperti gambar pada monitor televisi,
3. Digital yang dapat langsung disimpan pada suatu pita magnetik.

Citra terbagi menjadi 2 macam diantaranya yaitu :

a. Citra diam (*still image*) adalah citra tunggal yang tidak bergerak, contoh gambar gedung,
b. Citra bergerak (*moving image*) adalah rangkaian citra yang di tampilkan secara beruntun (*sekuensial*) sehingga memberi kesan pada mata kita sebagai gambar yang bergerak. Setiap citra dalam rangkaian itu disebut *frame*. Gambar-gambar yang nampak pada film layer lebar atau *television* pada hakikatnya terdiri atas ratusan sampai ribuan *frame*.

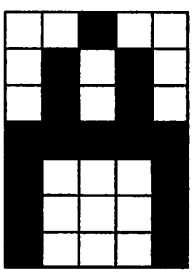
Dalam pengolahan citra, dilakukan operasi terhadap citra asli menjadi citra baru berdasarkan citra asli. Operasi yang dilakukan pada citra dikategorikan sebagai berikut :

1. *Point*, yaitu operasi yang menghasilkan *output* dimana setiap *pixel* hanya dipengaruhi oleh *pixel* pada posisi yang sama dari citra asli.
2. *Local*, yaitu operasi yang menghasilkan *output* dimana *pixel*nya dipengaruhi oleh *pixel* - *pixel* tetangga pada citra asli.
3. *Global*, yaitu operasi yang menghasilkan *output* dimana *pixel* nya dipengaruhi oleh semua *pixel* yang ada dalam citra asli. [7]

2.1.2. Citra Biner

Citra biner (*binary Image*) adalah citra yang hanya mempunyai 2 nilai derajat keabuan : hitam dan putih. Meskipun saat ini citra berwarna lebih disukai karena memberi kesan yang lebih kaya daripada citra biner, namun tidak membuat citra biner mati. Pada beberapa aplikasi citra biner, masih tetap dibutuhkan, misalnya citra logo instansi (yang hanya terdiri atas warna hitam dan putih), citra kode batang (*bar code*) yang tertera pada label barang,citra hasil pemindaian dokumen teks, dan sebagainya.

Pada citra biner *pixel-pixel* objek bernilai 1 dan *pixel-pixel* latar belakang bernilai 0. Pada waktu menampilkan gambar, 0 adalah putih sedangkan objek berwarna hitam adalah 1.



0	0	1	0	0
0	1	0	1	0
0	1	0	1	0
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1

Gambar 2.8 Huruf "A" dan representasi biner dari derajat keabuannya
Sumber : Pengolahan Citra Digital Dengan Pendekatan Algoritmik

Pengkonversian citra hitam putih (*grayscale*) menjadi citra biner dilakukan untuk alasan-alasan sebagai berikut :

1. Untuk mengidentifikasi keberadaan objek, yang direpresentasikan sebagai daerah (*region*) di dalam citra. Misalnya kita ingin memisahkan (*segmentasi*) objek dari latar belakangnya. *Pixel-pixel* objek dinyatakan dengan nilai 1 sedangkan *pixel* lainnya dengan 0. objek ditampilkan seperti gambar siluet. Untuk memperoleh siluet yang bagus, objek harus dapat dipisahkan dengan mudah dari gambar latar belakangnya.

2. Untuk lebih memfokuskan pada analisis bentuk morfologi, yang ada dalam hal ini. Intensitas *pixel* tidak terlalu penting dibandingkan bentuknya.setelah objek dipisahkan dari latar belakangnya, Properti geometri dan morfologi/topologi objek dapat dihitung dari citra biner. Hal ini berguna untuk pengambilan keputusan.

3. Untuk menampilkan citra pada piranti keluaran yang hanya mempunyai resolusi intensitas satu bit, yaitu piranti penampil dua arah atau biner seperti pencetak (*printer*).

4. Mengkonversi citra yang telah ditingkatkan kualitas tepinya (*edge enhancement*) ke penggambaran garis-garis tepi. Ini perlu untuk membedakan tepi yang kuat yang berkoresponden dengan batas-batas objek dengan tepi lemah yang berkoresponden dengan perubahan *illumination*, bayangan dan lain-lain. [3]

2.2. Jaringan Syaraf Tiruan

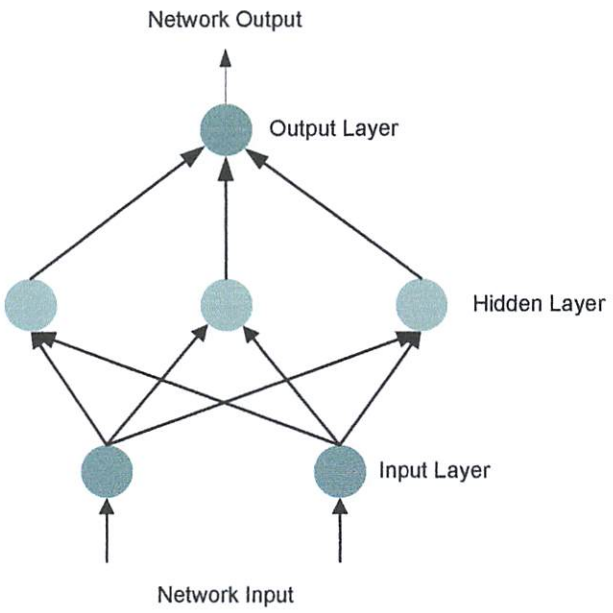
Jaringan Syaraf Tiruan merupakan suatu sistem pemrosesan informasi yang memiliki karakteristik-karakteristik menyerupai jaringan syaraf biologi (Fauset, 1994). Hal yang sama diutarakan oleh Simon Haykin, yang menyatakan bahwa jaringan syaraf tiruan adalah sebuah mesin yang dirancang untuk memodelkan cara otak manusia mengerjakan fungsi atau tugas-tugas tertentu. Mesin ini memiliki kemampuan menyimpan pengetahuan yang dimiliki menjadi bermanfaat. [1]

Secara prinsip jaringan syaraf tiruan dapat melakukan komputasi terhadap semua fungsi yang dapat dihitung (*Computable Function*). Jaringan syarat tiruan dapat melakukan apa yang dapat dilakukan oleh komputer digital normal. [1]

2.2.1. Arsitektur Jaringan Syaraf Tiruan

Secara umum, arsitektur jaringan syaraf tiruan terdiri atas beberapa lapisan, yaitu lapisan masukan (*input layer*), lapisan tersembunyi (*hidden layer*), dan lapisan keluaran (*output layer*). Masing-masing lapisan mempunyai jumlah node atau neuron yang berbeda-beda.

Arsitektur jaringan syaraf tiruan tersebut dapat diilustrasikan seperti dalam Gambar 2.9 berikut ini:



Gambar 2.9 Arsitektur Jaringan Syaraf Tiruan
Sumber : Konsep Kecerdasan Buatan

1. Lapisan masukan (*input layer*)

   Lapisan masukan merupakan lapisan yang terdiri dari beberapa neuron yang akan menerima sinyal dari luar dan kemudian meneruskan ke neuron-neuron lain dalam jaringan. Lapisan ini diilhami berdasarkan ciri-ciri dan cara kerja sel-sel syaraf sensori pada jaringan syaraf biologis.

2. Lapisan tersembunyi (*hidden layer*)

   Lapisan tersembunyi merupakan tiruan dari sel-sel syaraf konektor pada jaringan syaraf biologis. Lapisan tersembunyi berfungsi meningkatkan kemampuan jaringan dalam memecahkan masalah.

Konsekuensi dari adanya lapisan ini adalah pelatihan menjadi semakin sulit atau lama.

3. Lapisan keluaran (*output layer*)

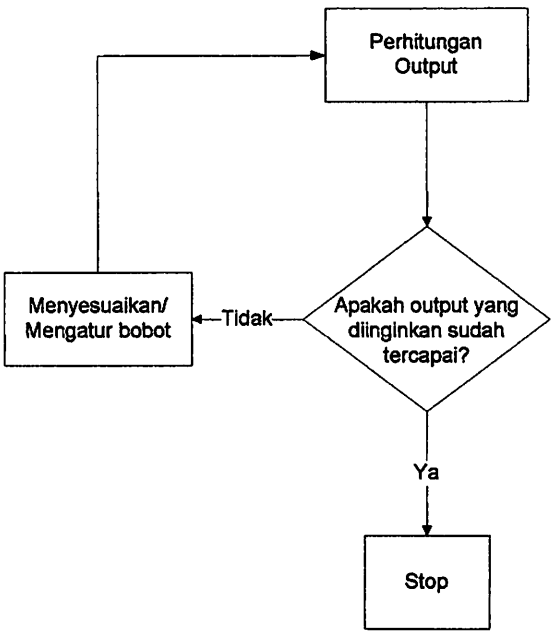
   Lapisan keluaran berfungsi menyalurkan sinyal-sinyal keluaran hasil pemrosesan jaringan. Lapisan ini juga terdiri dari sejumlah neuron. Lapisan keluaran merupakan tiruan sel-sel syaraf motor pada jaringan syaraf biologis. [1]

2.3. Algoritma Pembelajaran Jaringan Syaraf Tiruan

Belajar untuk jaringan syaraf tiruan merupakan suatu proses di mana parameter-parameter bebas jaringan syaraf tiruan diadaptasi melalui suatu proses perangsangan berkelanjutan oleh lingkungan di mana jaringan berada. Suatu jaringan syaraf tiruan belajar dari pengalaman. Proses yang lazim dari pembelajaran meliputi tiga tugas, yaitu:

1. Perhitungan output,
2. Membandingkan output dengan target yang diinginkan,
3. Menyesuaikan bobot dan mengulangi prosesnya.

Proses pembelajaran tersebut dapat dilihat dalam Gambar 2.10 berikut ini:



Gambar 2.10 Proses Pembelajaran dari suatu Jaringan Syaraf Tiruan
Sumber : Konsep Kecerdasan Buatan

Proses pembelajaran atau pelatihan tersebut merupakan proses perubahan bobot antar-neuron sehingga sebuah jaringan dapat menyelesaikan sebuah masalah. Semakin besar bobot keterhubungannya maka akan semakin cepat menyelesaikan suatu masalah. Proses pembelajaran dalam jaringan syaraf tiruan dapat diklasifikasikan menjadi dua bagian, yaitu:

1. *Supervised Learning* (pembelajaran terawasi) yang menggunakan sejumlah pasangan data masukan dan keluaran yang diharapkan. Contoh dari tipe ini adalah metode backpropagation, jaringan hopfield dan perceptron.

2. *Unsupervised Learning* (pembelajaran tidak terawasi) yang hanya menggunakan sejumlah pasangan data masukan tanpa ada contoh keluaran yang diharapkan. [1]

2.3.1. Algoritma Perceptron

Perceptron terdiri dari suatu input dan output. Perceptron merupakan bentuk paling sederhana dari jaringan syaraf tiruan yang biasanya digunakan untuk pengklasifikasian jenis pola khusus yang biasa disebut *linierly separable* (pola-pola yang terletak pada sisi yang berlawanan pada suatu bidang). Output unit akan diasumsikan bernilai 1 jika jumlah bobot inputnya lebih besar daripada threshold. Nilai threshold pada fungsi aktivasi adalah non-negatif. [1]

2.3.2. Metode Backpropagation

Metode *Back Propagation* merupakan metode pembelajaran lanjut yang dikembangkan dari aturan perceptron. Hal yang ditiru dari perceptron adalah tahapan dalam algoritma jaringan.

Metode back propagation ini dikembangkan oleh Rumelhart, Hinton dan Williams sekitar tahun 1986 yang mengakibatkan peningkatan kembali minat terhadap jaringan syaraf tiruan. Metode ini terdiri dari dua tahap, yaitu tahap *feedforward* yang diambil dari perceptron dan tahap *back propagation error.*

Salah satu hal yang membedakan antara back propagation dengan perceptron adalah arsitektur jaringannya. Perceptron memiliki jaringan lapis tunggal sedangkan back propagation memiliki lapisan lapis jamak.

Berikut ini adalah langkah-langkah algoritma pembelajaran untuk back propagation:

1. Inisialisasi bobot (ditentukan oleh bilangan acak yang kecil, antara 0 sampai 1.
2. Selama kondisi berhenti tidak terpenuhi, lakukan langkah 3 sampai langkah 10.
3. Untuk setiap pasang vektor pelatihan, lakukan langkah 3 sampai langkah 8.

Feedforward

4. Setiap neuron pada lapisan masukan (Xi, i=1, 2, ..., n) menerima sinyal masukan xi dan menjalankan sinyal tersebut ke semua neuron pada lapisan selanjutnya (dalam hal ini adalah lapisan tersembunyi).
5. Untuk setiap neuron dalam lapisan tersembunyi (Zj, j=1, 2, ..., p), jumlahkan bobotnya dengan sinyal masukkannya masing-masing :

$$Z_{in_j} = v_{0j} + \sum_{i=1}^{n} x_i v_j$$

Terapkan fungsi aktivasi untuk menghitung nilai sinyal keluaran

$$Z_j = f\left(Z_{in_j}\right)$$

Kemudian kirimkan sinyal ini ke semua neuron pada lapisan selanjutnya (dalam hal ini adalah lapisan keluaran).

6. Untuk setiap neuron pada lapisan keluaran (Yk, k=1, 2, ..., m), jumlahkan bobotnya dengan sinyal masukkannya masing-masing

$$Y_{in_k} = w_{0j} + \sum_{i=1}^{p} z_j w_k$$

Tetapkan fungsi aktivasi untuk menghitung nilai sinyal keluaran

$$Y_k = f(Y_{in_k})$$

Back propagation of error

7. Setiap neuron pada lapisan keluaran (Yk, k=1, 2, ..., m) menerima sebuah pola target yang berhubungan dengan pola masukkan pelatihan kemudian hitung kesalahannya

$$\delta_k = (t_k - y_k).f\left(y_{in_k}\right)$$

Hitung perubahan bobotnya (digunakan nanti untuk mengubah nilai wjk)

$$\Delta w_{j\,k} = \alpha.\delta_k.z_k$$

Hitung perubahan biasnya (digunakan nanti untuk mengubah nilai wok)

$$\Delta w_{0\,k} = \alpha.\delta_k$$

8. Untuk setiap neuron pada lapisan tersembunyi (Zj, j=1, 2, ...,p), jumlahkan nilai delta masukkannya (dari neuron pada lapisan di atasnya).

$$\delta_{in_j} = \sum_{k=1}^{m} \delta_k w_{jk}$$

Kalikan dengan turunan aktivasinya untuk menghitung nilai kesalahannya.

$$\delta_j = \left(\delta_{in_j}\right) . f\left(z_{in_j}\right)$$

Hitung perubahan bobotnya (digunakan nanti untuk mengubah nilai Vij)

$$\Delta v_{ij} = \alpha . \delta_j . x_i$$

Kemudian hitung perubahan biasnya (digunakan nanti untuk mengubah nilai V0j)

$$\Delta v_{0j} = \alpha . \delta_j$$

Update bobot dan bias

9. Untuk setiap neuron pada lapisan keluaran (Yk, k=1, 2, ..., m), ganti nilai bobot dan biasnya (j=0, 1, 2, ..., p)

   $$w_{jk_{(baru)}} = w_{jk_{(lama)}} + \Delta w_{jk}$$

   Untuk setiap neuron pada lapisan tersembunyi (Zj, j=1, 2, ..., p), ganti nilai bobot dan biasnya (i=0, 1, 2, ..., n)

   $$v_{ij_{(baru)}} = v_{ij_{(lama)}} + \Delta v_{ij}$$

10. Uji/periksa kondisi berhenti. [1]

2.4. Delphi 7

Delphi 7 adalah paket bahasa pemrograman yang bekerja dalam sistem operasi Windows, yang mempunyai cakupan kemampuan yang luas dan sangat canggih. Beberapa jenis aplikasi yang dapat kita buat dengan Delphi, termasuk aplikasi untuk mengolah teks, grafik, angka, database dan aplikasi web.

Secara umum kemampuan Delphi adalah menyediakan komponen – komponen dan bahasa pemrograman yang andal, sehingga memungkinkan kita untuk membuat program aplikasi sesuai dengan keinginan, dengan tampilan dan kemampuan yang canggih.

Untuk mempermudah pemrogram dalam membuat program aplikasi, Delphi menyediakan fasilitas pemrograman yang sangat lengkap. Fasilitas pemrograman tersebut dibagi dalam dua kelompok, yaitu obyek dan bahasa pemrograman. Secara ringkas, object adalah suatu komponen yang mempunyai bentuk fisik dan biasanya dapat diliat (visual). Obyek biasanya dipakai untuk melakukan tugas tertentu dan mempunyai batasan–batasan tertentu. Sedangkan bahasa pemrograman secara singkat dapat disebut sebagai sekumpulan teks yang mempunyai arti tertentu dan disusun dengan aturan tertentu serta untuk menjalankan tugas tertentu. Delphi menggunakan struktur bahasa pemrograman Obyek Pascal yang sudah sangat dikenal dikalangan pemrograman professional. Gabungan dari obyek dan bahasa pemrograman ini disebut sebagai bahasa pemrograman berorientasi obyek atau Object Oriented Programming (OOP).

BAB III
PERANCANGAN SISTEM

3.1. Desain Sistem

Pengenalan Golongan Darah adalah proses pengenalan golongan darah melalui pengenalan pola/citra pembekuannya dimana data yang direpresentasikan dapat dikenali sebagai golongan darah yang dimaksud. Pola pembekuan empat golongan darah telah dikenali dan dapat dibedakan oleh ahlinya. Jaringan syaraf tiruan dengan metode pembelajaran Back Propagation digunakan sebagai generalisasi model matematik dari pembelajaran seorang ahli untuk mengenali golongan darah melalui pola pembekuan masing-masing golongan darah tersebut setelah diberi reagent antigen.

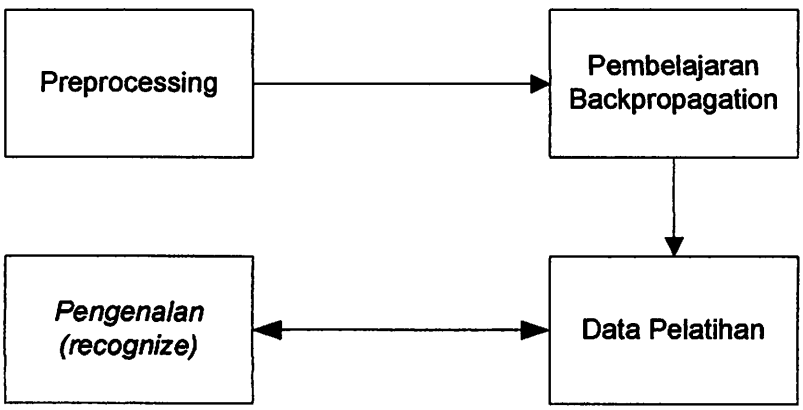
Pengenalan golongan darah dengan metode Jaringan Syaraf Tiruan *Back Propagation* dirancang untuk mendeteksi golongan darah manusia. Pada aplikasi ini, digunakan metode pengolahan citra digital dan selanjutnya akan diteruskan pada pengenalan golongan darah menggunakan metode *backpropagation* sebagai pengambilan keputusan. Pengenalan golongan darah merupakan suatu aplikasi pengenalan pola dimana komputer akan berusaha mengenali golongan darah yang dimasukkan.

Desain aplikasi ini meliputi algoritma yang digunakan dalam sistem yang digambarkan dengan diagram alir proses. Diagram alir proses

menjelaskan tentang proses deteksi tepi, proses training dan proses pengenalannya.

3.1.1. Blok Diagram Sistem

Adapun blok diagram dari aplikasi pengenalan golongan darah seperti dalam Gambar 3.1 sebagai berikut :



Gambar 3.1 Blok Diagram Sistem

Sebelum aplikasi mengenali suatu pola, pola fitur terlebih dahulu melalui preprocessing meliputi deteksi tepi dan segmentasi. Hasil pola yang telah melalui preprocessing kemudian ditraining dengan metode backpropagation. Pengenalan pola dilakukan dengan cara membandingkan citra masukan dengan hasil ekstraksi yang telah tersimpan pada data pelatihan menggunakan metode *Back Propagation*.

3.2. Algoritma dan *FlowChart* Sistem

Implementasi metode pengenalan pola golongan darah manusia ini terdiri dari tiga tahap utama yaitu proses deteksi tepi (*edge detection*), proses pembelajaran (*training*) dan proses pengenalan pola (*recognize*).

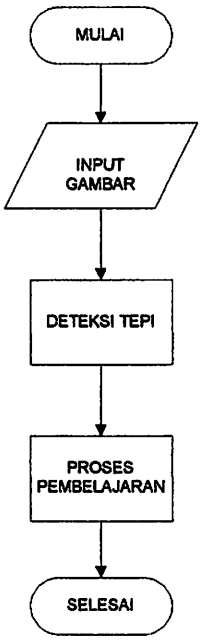
Apabila dikelompokkan berdasarkan penggunaan aplikasi, tahap-tahap di atas dibagi menjadi dua kelompok yaitu

1. Proses pembelajaran sistem terdiri dari proses deteksi tepi (*edge detection*) dan proses pembelajaran backpropagation (*training*).
2. Proses pengenalan yang terdiri dari proses pengenalan pola (*recognize*)

3.2.1. Proses Pembelajaran Sistem

Proses pembelaaran sistem yaitu proses awal untuk mendapatkan pola-pola pelatihan yang akan digunakan untuk proses pada tahap selanjutnya (proses pengenalan pola). Jika semua proses pada tahap pembelajaran sistem telah selesai dilakukan, maka sistem dapat melakukan proses selanjutnya yaitu proses pengenalan pola golongan darah.

Apabila digambarkan dalam diagram, alur kerja proses pembelajaran sistem seperti dalam Gambar 3.2 sebagai berikut :

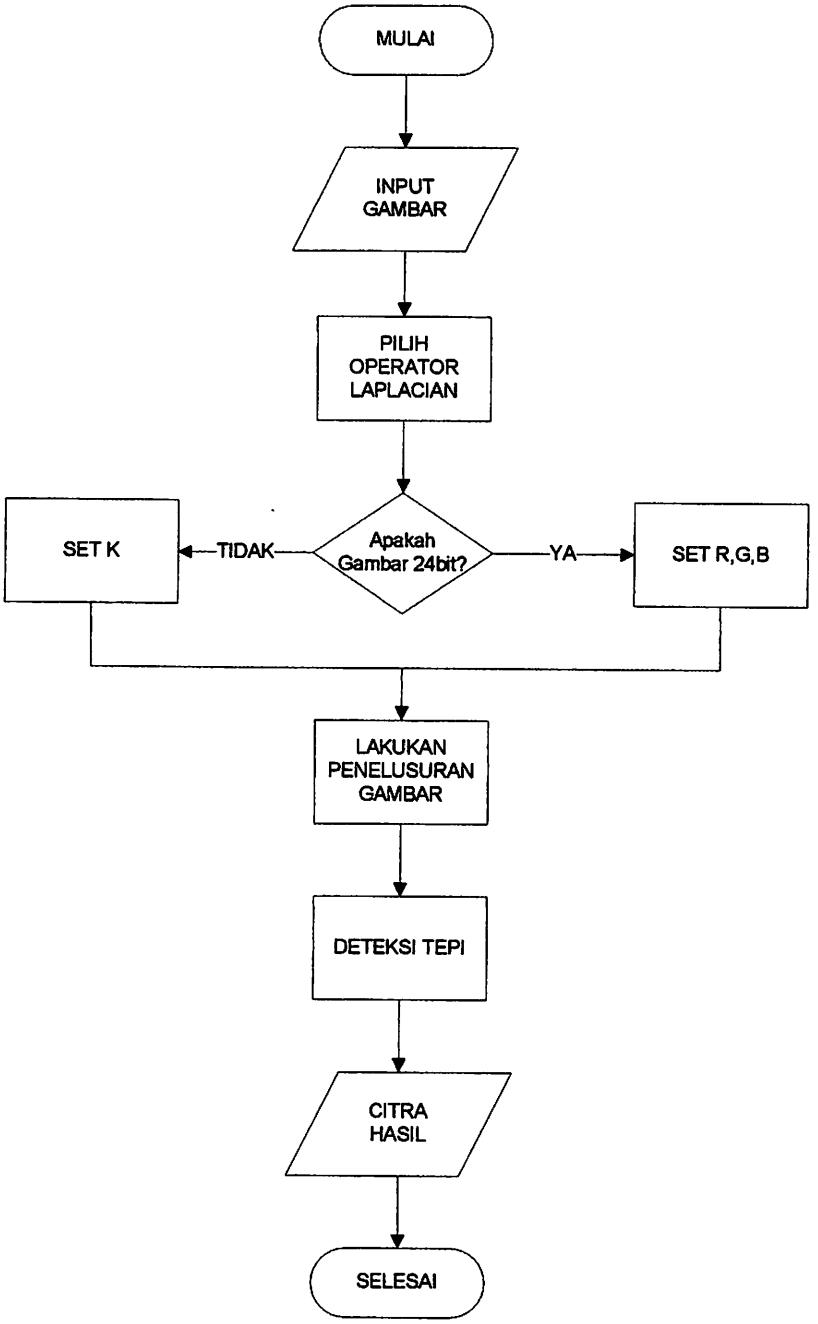


Gambar 3.2 *Flowchart* sistem pembelajaran

Berdasarkan flowchart diatas, *input*an berupa gambar pola darah yang ingin dikenali. Pola yang telah dimasukkan kemudian di deteksi tepi untuk menghasilkan fitu-fitur yang akan digunakan untuk proses pembelajaran backpropagation. Proses deteksi tepi dan proses pembelajaran backpropagation dijelaskan seperti dibawah ini.

1. **Proses Deteksi Tepi (*edge detection*)**

Pada proses deteksi tepi, citra yang dimasukkan dapat berupa citra *grayscale* maupun citra *true color* yang selanjutnya diproses sampai menghasilkan tepi-tepi objek citra. Proses deteksi tepi dapat digambarkan dalam Gambar 3.3 sebagai berikut:



Gambar 3.3 *Flowchart* proses deteksi tepi

Pada flowchart di atas inputan yang dimasukkan merupakan sebuah gambar dimana dapat berupa citra 24bit (*true color*) atau citra 8bit (*grayscale*). Deteksi tepi dengan metode laplacian dimaksudkan untuk

mendapatkan tepi objek yang ideal, idealnya tepi objek yang diinginkan adalah sebuah garis tipis setebal satu piksel agar tidak menimbulkan keraguan bila dilakukan analisis maka deteksi tepi dengan bantuan fungsi turunan kedua (dalam hal ini operator laplacian) akan mempunyai ketebalan satu piksel saja, sesuai dengan garis tepi ideal yang diinginkan. Setelah itu dilakukan tahap penelusuran gambar yang akan dideteksi tepi untuk mendapatkan ciri-ciri dari gambar tersebut.

Algoritma yang diterapkan pada proses deteksi tepi sebagai berikut:

Menentukan operator deteksi tepi turunan kedua (laplacian)

```
MaskLaplacian5: Mask3x3 =
 (( 0, -1,  0),
  ( -1,  4, -1),
  ( 0, -1,  0));
MaskLaplacian91: Mask3x3 =
 (( -1, -1, -1),
  ( -1,  8, -1),
  ( -1, -1, -1));
MaskLaplacian92: Mask3x3 =
 (( 1, -2,  1),
  ( -2,  4, -2),
  ( 1, -2,  1));
```

Inisialisasi tinggi, lebar dan nilai koefisien citra yang akan di deteksi tepi

```
Jika image 8bit
   SetLength(Ki, w, h);
   SetLength(Ko, w, h);

Jika image 24bit
   SetLength(Ri, w, h);    SetLength(Gi, w, h);    SetLength(Bi, w, h);
   SetLength(Ro, w, h);    SetLength(Go, w, h);    SetLength(Bo, w, h);
```

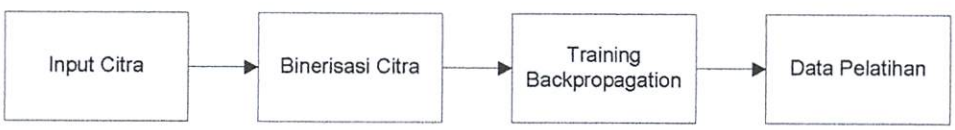
Melakukan penelusuran gambar dan menentukan tepi dari gambar.

```
Jika image 8bit
for y := 0 to h-1 do
  begin
    PC := FormCitra.Image.Picture.Bitmap.ScanLine[y];
    PH := FormHasil.Image.Picture.Bitmap.ScanLine[y];
    for x := 0 to w-1 do
      begin
        Ki[x, y] := PC[x];
        Ko[x, y] := PH[x];
      end;
  end;

Jika image 24bit
for y := 0 to h-1 do
  begin
    PC := FormCitra.Image.Picture.Bitmap.ScanLine[y];
    PH := FormHasil.Image.Picture.Bitmap.ScanLine[y];
    for x := 0 to w-1 do
      begin
        Bi[x, y] := PC[3*x];
        Gi[x, y] := PC[3*x+1];
        Ri[x, y] := PC[3*x+2];
        Bo[x, y] := PH[3*x];
        Go[x, y] := PH[3*x+1];
        Ro[x, y] := PH[3*x+2];
      end;
  end:
```

2. Proses Pembelajaran Backpropagation (*training*)

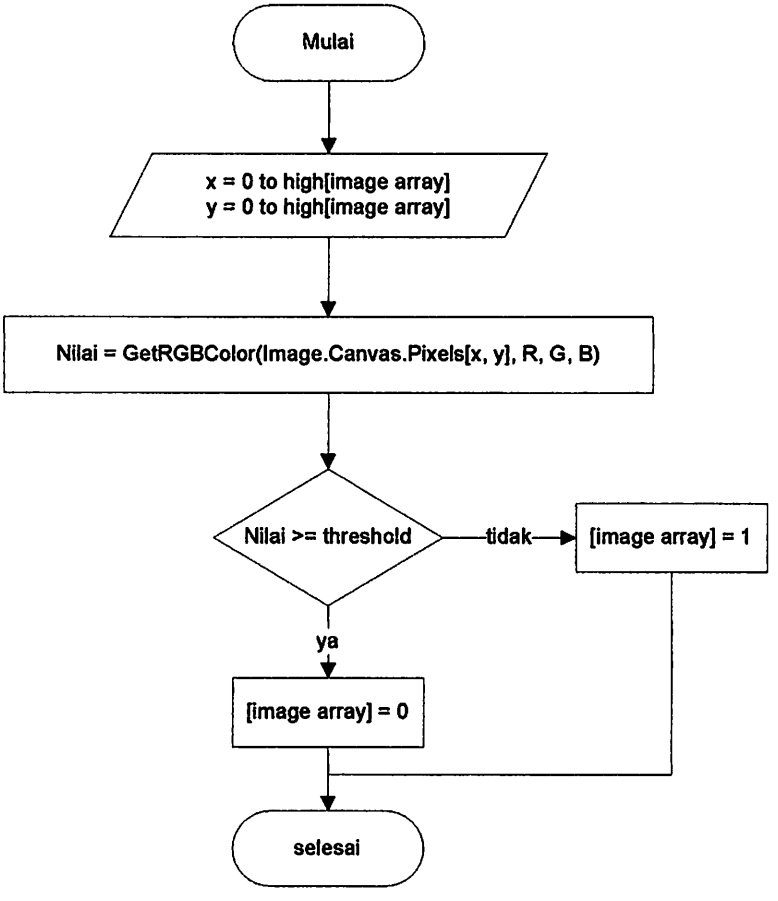
Pada proses pembelajaran backpropagation terdapat beberapa proses yang harus dilakukan yaitu proses binerisasi citra dan proses pembelajaran backpropagation, adapun blok diagram proses pembelajaran *Backpropagation* dapat digambarkan seperti dalam Gambar 3.4 sebagai berikut.



Gambar 3.4 Blok diagram proses pembelajaran backpropagation

Proses binerisasi adalah pemrosesan citra yang telah dideteksi tepi dari citra 24bit (*true color*) atau citra 8bit (*grayscale*) diubah menjadi citra biner 1bit. Operasi yang digunakan untuk proses binerisasi yaitu operasi pengambangan (*threshold*). Operasi *threshold* mengelompokkan nilai derajat keabuan sebuah citra ke dalam dua kelas yaitu hitam dan putih. Dimana hitam didefinisikan menjadi 1 dan putih didefinisikan menjadi 0.

Adapun proses binerisasi image dapat digambarkan dalam Gambar 3.5 sebagai berikut:



Gambar 3.5 *Flowchart* proses binerisasi citra

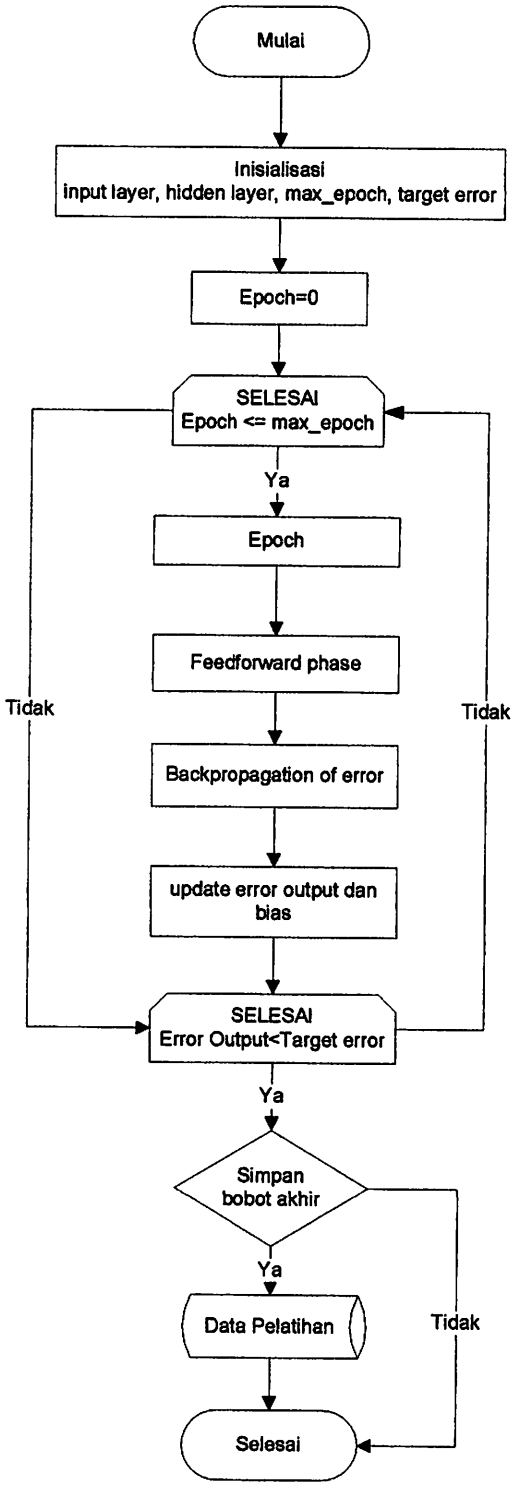
Pemilihan nilai threshold yang digunakan berpengaruh pada ketajaman suatu citra. Rentang nilai threshold yang sering digunakan adalah antara 0-255. Pada flow chart di atas nilai threshold yang digunakan default threshold. Proses ini melakukan pemeriksaan piksel citra, jika piksel citra kurang dari nilai threshold maka piksel citra berubah menjadi 1, begitu juga sebaliknya jika piksel melebihi nilai threshold maka piksel citra berubah menjadi 0.

Pada tahap selanjutnya yaitu proses pembelajaran backpropagation. Untuk proses pembelajaran backpropagation menggunakan error output untuk mengubah nilai-nilai bobotnya dalam arah mundur (*backpropagation of error).* Untuk mendapatkan error tersebut dilakukan tahap perambatan maju (*feedforward*) terlebih dahulu. Pada saat perambatan maju, neuron-neuron diaktifkan dengan menggunakan fungsi aktivasi yang dapat dideferensikan, seperti fungsi sigmoid-bipolar.

$$g(x) = 2f(x) - 1 = \frac{2}{1 + e^{-\sigma x}} - 1 = \frac{1 - e^{-\sigma x}}{1 + e^{-\sigma x}}$$

Fungsi ini tergantung pada steepness parameter (σ). Agar fungsi ini menghasilkan nilai yang dibatasi oleh bilangan biner (0 sampai 1) maka $\sigma = 1$.

Proses Training dapat digambarkan dalam Gambar 3.6 sebagai berikut:



Gambar 3.6 *Flowchart* proses *training backpropagation*

Inisialisasi bobot pada input layer ditentukan oleh bilangan acak yang kecil antara 0 sampai 1. Tahap-tahap feedforward pada backpropagation berfungsi untuk menjalankan sinyal bobot masukan ke semua neuron pada lapisan selanjutnya. Tahapan feedforward menjumlahkan sinyal-sinyal bobot input layer pada hidden layer dan menjumlahkan sinyal-sinyal bobot hidden layer pada output layer. Sinyal-sinyal tersebut diaktivasi dengan fungsi sigmoid bipolar. Tahap selanjutnya yaitu tahap backpropagation error berfungsi untuk melakukan perubahan bobot pada lapisan sebelumnya dan mengupdate bobot yang baru. Tahap ini menghitung informasi error yang didapatkan dari selisih antara nilai target yang ditentukan dengan nilai keluaran dari output layer. Informasi error tersebut digunakan untuk mengoreksi nilai bobot dan bias pada hidden layer dan output layer. Langkah diatas dilakukan berulang-ulang selama kondisi selesai tidak terpenuhi atau kurang dari maksimum epoch.

Algoritma pada proses pembelajaran backpropagation ini dapat diterapkan sebagai berikut:

Inisialisasi Layer

```
FNInputNeuron := NInputNeuron;
FNHiddenNeuron := NHiddenNeuron;
FNOutputNeuron := NOutputNeuron;
InitLayers;
```

FeedForward dari Input Layer ke Hidden Layer

```
//--- Feedforwards from input layer to hidden layer
for j := 1 to FNHiddenNeuron do
begin
 //- Sums its weighted input signals from input layer
 FHiddenLayer[j] := 0;
 for i := 0 to FNInputNeuron do
   FHiddenLayer[j] := FHiddenLayer[j] +
                (FInputLayer[i] * FHiddenLayerWeights[j, i]);
 //- Applies the bipolar sigmoid activation function
 FHiddenLayer[j] := BipolarSigmoid(FHiddenLayer[j]);
end;
```

FeedForward dari Input Layer ke Hidden Layer

```
//--- Feedforwards from hidden layer to output layer
for k := 1 to FNOutputNeuron do
begin
 //- Sums its weighted input signals from hidden layer
 FOutputLayer[k] := 0;
 for j := 0 to FNHiddenNeuron do
  FOutputLayer[k] := FOutputLayer[k] +
              (FHiddenLayer[j] * FOutputLayerWeights[k, j]);
 //- Applies the bipolar sigmoid activation function
 FOutputLayer[k] := BipolarSigmoid(FOutputLayer[k]);
end;
```

Backpropagation Error Phase menghitung perubahan bobot pada Output Layer

```
    //- Computes output layer weights error information
    for k := 1 to FNOutputNeuron do
    begin
     Error := FTrainingSet[l].TargetPattern[k] - FOutputLayer[k];
     if Abs(Error) >= FErrorThreshold then
      FNNeuronError := FNNeuronError + 1;
     FTrainingError := FTrainingError + (Error * Error);
     OutputErrors[k] := Error * BipolarSigmoidDerivation(FOutputLayer[k]);
     for j := 0 to FNHiddenNeuron do
      OutputWeightsCorrection[k, j] :=
G      FLearningRate * OutputErrors[k] * FHiddenLayer[j];
    end;
```

Backpropagation Error Phase menghitung perubahan bobot pada Hidden Layer

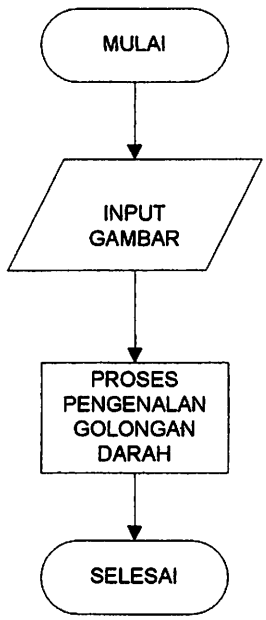
```
//- Computes hidden layer weights error information
for j := 1 to FNHiddenNeuron do
begin
 HiddenErrors[j] := 0;
 for k := 1 to FNOutputNeuron do
  HiddenErrors[j] := HiddenErrors[j] +
              (OutputErrors[k] * FOutputLayerWeights[k, j]);
 HiddenErrors[j] := HiddenErrors[j] *
            BipolarSigmoidDerivation(FHiddenLayer[j]);
 for i := 0 to FNInputNeuron do
  HiddenWeightsCorrection[j, i] :=
   FLearningRate * HiddenErrors[j] * FInputLayer[i]
end;
```

Update bobot dan bias pada Output Layer dan Hidden Layer

```
//- Updates output layer weights and bias
for k := 1 to FNOutputNeuron do
 for j := 0 to FNHiddenNeuron do
  FOutputLayerWeights[k, j] := FOutputLayerWeights[k, j] +
                    OutputWeightsCorrection[k, j];
//- Updates hidden layer weights and bias
for j := 1 to FNHiddenNeuron do
 for i := 0 to FNInputNeuron do
  FHiddenLayerWeights[j, i] := FHiddenLayerWeights[j, i] +
                    HiddenWeightsCorrection[j, i];
```

3.2.2. Algoritma dan *Flowchart* Proses Pengenalan

Pada proses pengenalan golongan darah pengguna hanya memasukkan gambar akan langsung diproses untuk melakukan pengenalan. Diagram alir dapat digambarkan dalam Gambar 3.7 sebagai berikut:



Gambar 3.7 *Flowchart* proses pengenalan golongan darah

Algoritma pada proses pengenalan dapat digambarkan sebagai berikut:

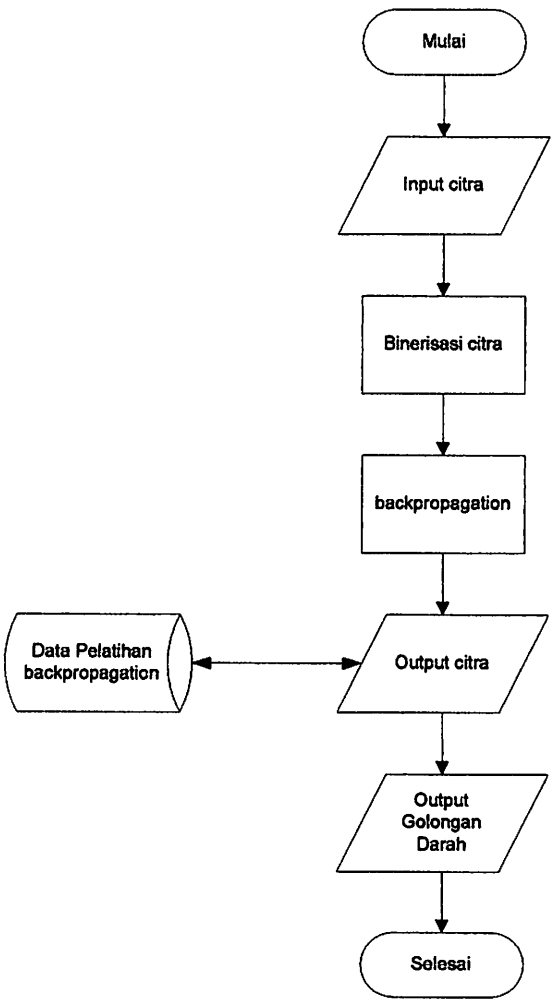
```
DCR.Recognize;
ResultText.SetText(DCR.ResultText);
XX:='';
YY:='';
XX:=Trim(ResultText.RichEdit.Text);
For I:= 1 To Length(XX) do
 Begin
   if XX[i] in ['A','B'] Then
     YY:=YY+XX[i];
 end;

If YY ='AB' Then  Panel1.Caption:='B';
If YY ='BA' Then  Panel1.Caption:='A';
If YY ='AA' Then  Panel1.Caption:='AB';
If YY ='BB' Then  Panel1.Caption:='O';
end;
```

Pada proses pengenalan golongan darah input citra terlebih dahulu mengalami proses binerisasi seperti pada proses pembelajaran backpropagation. Hal ini dimaksudkan untuk mendapatkan hasil yang sesuai

dengan data pelatihan. Setelah dibinerisasi, citra selanjutnya mengalami proses yang sama dengan proses training backpropagation sebelum dibandingkan dengan data pelatihan backpropagation dan menghasilkan output yang diinginkan yaitu berupa golongan darah manusia.

Adapun proses pengenalan dapat digambarkan dalam Gambar 3.8 sebagai berikut:

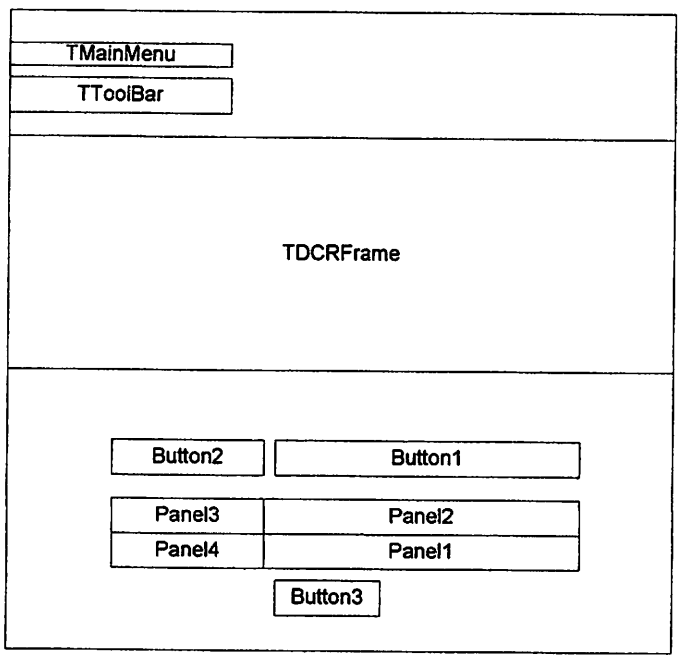


Gambar 3.8 *Flowchart* detail proses pengenalan golongan darah

3.3. Desain Tampilan Aplikasi

Dalam pembuatan aplikasi, *form-form* yang dibutuhkan adalah *Form* Pengenalan Golongan Darah, *Form* Deteksi Tepi dan *Form* Training. Masing-masing fungsi dari *form* tersebut akan dibahas dibawah ini :

3.3.1 *Form* Pengenalan Golongan Darah



Gambar 3.9 *Form* Pengenalan Golongan Darah

Komponen dalam Form Pengenalan Golongan Darah terdiri dari:

1. TMainMenu

   Yaitu menu utama dalam aplikasi ini, dimana sub menu yang terdapat di Main Menu yaitu :

   File | Open Training, Open Picture, Exit.

Process | Training, Deteksi Tepi, Recognize, Options.

Help | About

2. TToolBar

   Menu yang terdapat pada Toolbar ini hampir sama dengan Main Menu, hanya untuk memudahkan user. Menu-menu tersebut yaitu :

   Open Training | Open Picture | Recognize

3. TDCRFrame

   TDCRFrame yaitu frame tempat meletakkan gambar atau image golongan darah yang akan dikenali.

4. Button1

   Button1 merupakan tombol untuk melakukan proses pengenalan | Proses Pengenalan

5. Button2

   Button2 merupakan tombol untuk memasukkan image golongan darah yang akan dikenali | Open Darah Seseorang

6. Button3

   Button3 merupakan tombol untuk melakukan perintah untuk mencetak hasil pengenalan golongan darah | Print

7. Panel1

   Panel1 merupakan tempat untuk mengetahui hasil pengenalan golongan darah apakah golongan darah A, B, AB atau O.

8. Panel2

   Panel2 merupakan panel untuk mengetahui nama orang yang dikenali darahnya, nama tersebut langsung terhubung dengan nama file image golongan darah yang bersangkutan.

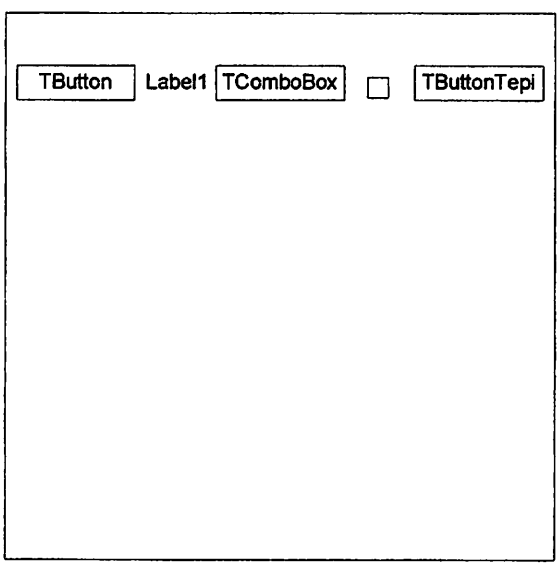
9. Panel3

   Panel3 hanya bersifat label yang bertuliskan | Nama Orang :

10. Panel4

    Panel4 sama seperti panel3 bersifat label yang bertuliskan | Golongan Darah :

3.3.2 *Form* Deteksi Tepi



Gambar 3.10 *Form* Deteksi Tepi

Komponen dalam Form Deteksi Tepi terdiri dari :

1. TButton

TButton berfungsi untuk mengambil citra yang akan di deteksi tepi.

2. TComboBox

   TComboBox yaitu combo box yang berfungsi untuk memilih operator deteksi tepi yang diinginkan.

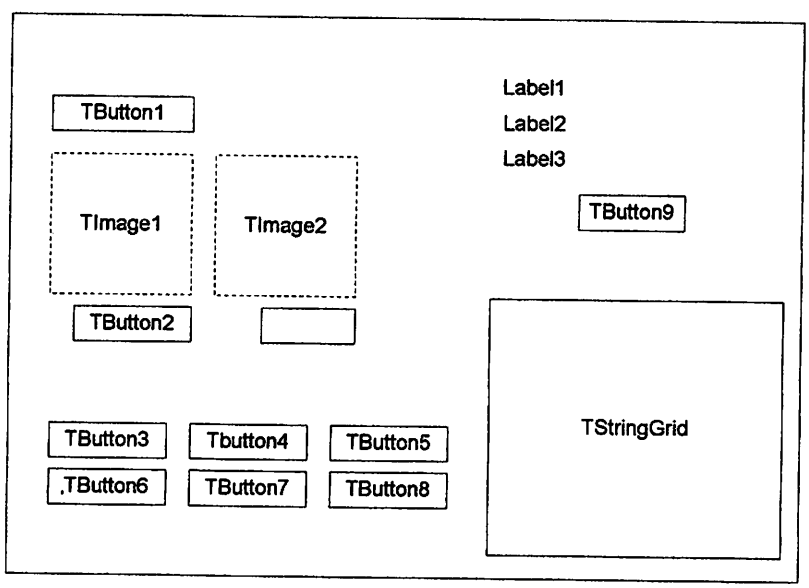
3. TCheckBox

   TCheckBox berfungsi untuk memilih background berwarna putih atau background berwarna hitam.

4. TbuttonTepi

   TbuttonTepi yaitu button yang berfungsi untuk melakukan perintah deteksi tepi.

3.3.3 Form Training



Gambar 3.11 *Form Training*

Komponen dalam Form Training terdiri dari :

1. TButton1

   TButton1 yaitu tombol yang berfungsi untuk mengambil citra yang telah dideteksi tepi untuk di training. | Buka Gambar Training

2. TButton2

   TButton2 yaitu tombol yang berfungsi untuk menghapus gambar atau image pada TImage1 dan TImage2. | Clear

3. TButton3

   TButton3 yaitu tombol yang berfungsi untuk menambahkan gambar pada TImage kedalam data training. | Add Training

4. TButton4

   TButton4 yaitu tombol yang berfungsi untuk melakukan perintah training. | Training

5. TButton5

   TButton5 yaitu tombol yang berfungsi untuk melakukan perintah retrain. | Retrain

6. TButton6

   TButton6 yaitu tombol yang berfungsi untuk melakukan perintah test. | Test

7. TButton7

   TButton7 yaitu tombol yang berfungsi untuk membuka form options | Options

8. TButton8

   TButton8 yaitu tombol yang berfungsi untuk keluar dari form training. | Close

9. TButton9

   TButton9 yaitu tombol yang berfungsi untuk melakukan perintah stop ketika training sedang berjalan. | Stop

10. TImage1

    TImage1 yaitu image frame untuk menempatkan image original yang akan dimasukkan ke dalam data training.

11. TImage2

    TImage2 yaitu image frame untuk meletakkan image hasil yang ditelah ditambahkan kedalam data training.

12. Label1

    Label1 yaitu label yang digunakan untuk mengetahui jumlah epoch.

13. Label2

    Label2 yaitu label yang digunakan untuk mengetahui Training error.

14. Label3

    Label3 yaitu label yang digunakan untuk mengetahui Pixel error.

15. TStringGrid

    TStringGrid yaitu String Grid yang digunakan untuk mengetahui hasil test dari data training yang telah dimasukkan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN SISTEM

Pada bab sebelumnya telah dibahas mengenai analisa desain sistem dari aplikasi pengenalan golongan darah manusia. Pada bagian ini akan dibahas penerapan dari aplikasi pengenalan golongan darah manusia. Data yang akan digunakan dalam pengujian yaitu sample golongan darah manusia yang diperoleh dari penelitian yang dilakukan pada teman-teman. Tujuan dari penerapan aplikasi ini adalah untuk menguji hasil rancangan yang telah dibuat pada bab sebelumnya.

4.1. Pengujian

Tujuan dari pengujian ini adalah untuk mengetahui hasil yang diperoleh pada tiap-tiap langkah. Pengujian ini dilakukan dengan membandingkan hasil yang diperoleh dengan apa yang ingin dicapai. Pengujian program meliputi verifikasi dan validasinya. Verifikasi adalah pengujian terhadap keutuhan program, yakni apakah program mampu berjalan tanpa kesalahan. Sedangkan validasi adalah pengujian terhadap kemampuan program, yakni apakah program mampu berfungsi sesuai dengan yang diharapkan.

Adapun spesifikasi *hardware* dan *software* dari pengujian adalah sebagai berikut :

1. Software

Aplikasi ini dapat berjalan pada semua sistem operasi Windows baik windows XP maupun Windows Vista.

2. Hardware

Perangkat keras yang dapat digunakan untuk pengujian adalah :

1. *Personal computer* Pentium(R) Dual-Core
2. Memory 1 GB
3. Kamera Digital

Untuk menguji program yang telah dibuat, buatlah skenario pengujian sebagai berikut :

1. Memasukkan pola-pola data *training* kedalam program. Data yang dimaksud adalah berbagai variasi sample darah baik yang menggumpal maupun tidak menggumpal.
2. Melaksanakan pengenalan

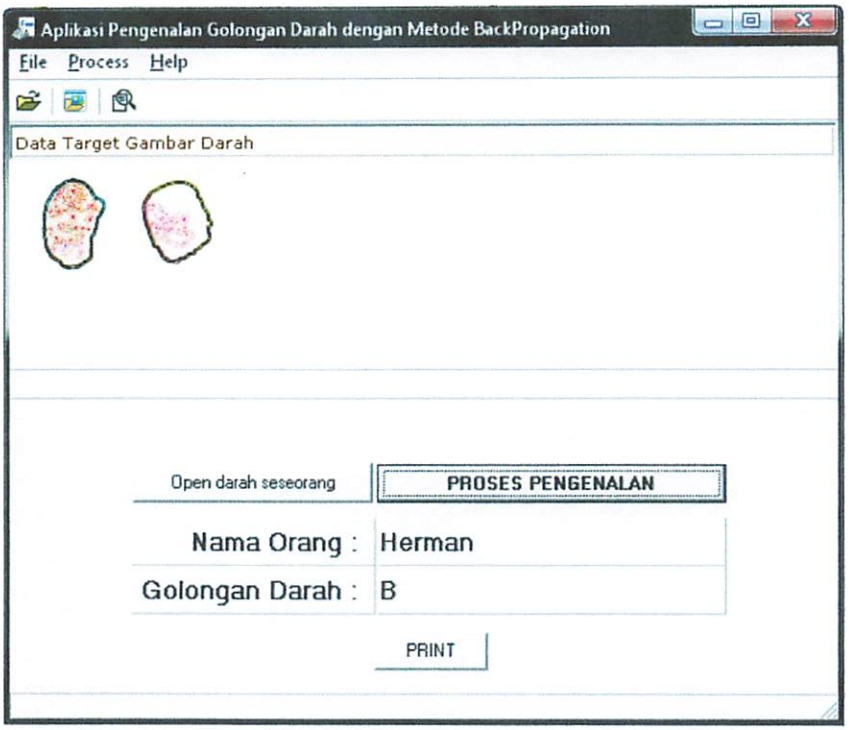
   Pengenalan karakter dilakukan berulang-ulang dengan *sample* yang berbeda-beda.
3. Menguji hasil pengenalan

   Setelah melaksanakan pengenalan, diharapkan minimal sistem mampu mengenali pola *input* yang sama persis dengan yang diajarkan padanya (proses pembelajaran). Jika sistem tidak mampu mengenali, kemungkinan penyebabnya yakni perlu ditambahkan *sample data training* agar sistem mengenal pola yang diajarkan dengan lebih jelas.

4.2. Penggunaan Aplikasi

Pada saat aplikasi di *running* halaman yang pertama kali muncul adalah form pengenalan golongan darah, hal ini untuk memudahkan user langsung

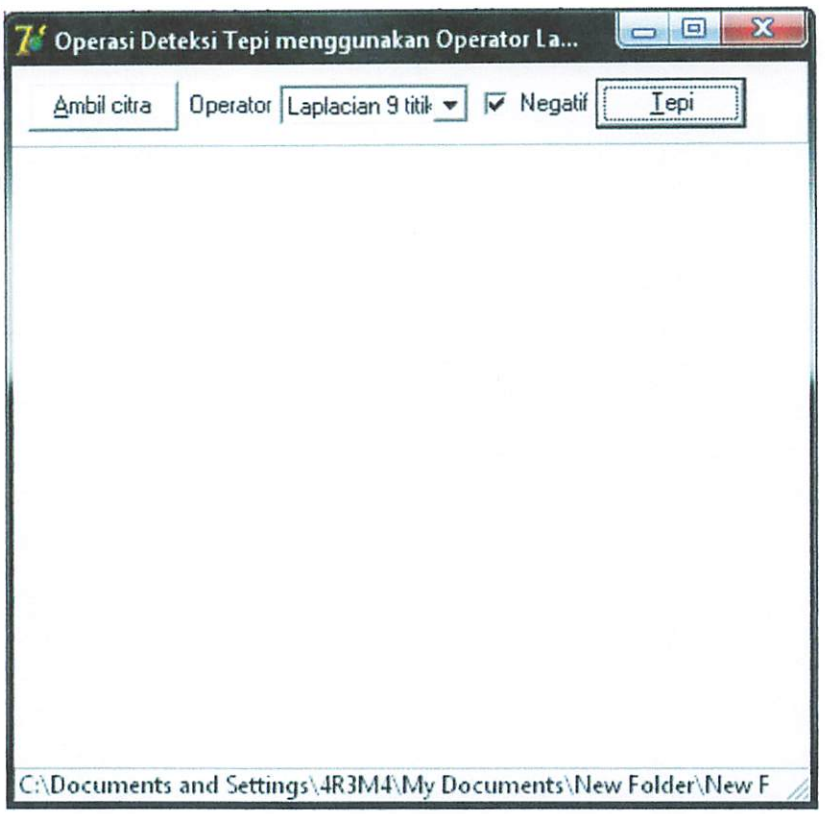
menggunakan aplikasi untuk mengenali golongan darah. Form pengenalan darah dapat digambarkan dalam Gambar 4.1 dibawah ini.



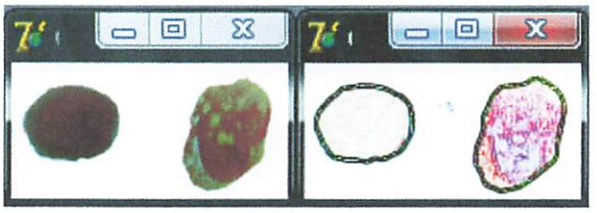
Gambar 4.1. Form Hasil Pengenalan Golongan Darah

Pada saat melakukan proses pengenalan terlebih dahulu harus dimasukkan data pelatihan yang telah dibuat, setelah itu baru memasukkan pola atau gambar golongan darah yang ingin dikenali. Proses pengenalan ini tidak membutuhkan waktu yang banyak, dibutuhkan sekitar 1-2 detik untuk mendapatkan hasilnya.

Untuk proses deteksi tepi dan proses memasukkan data training user hanya menginputkan pola darah yang ingin dideteksi tepi, lalu memilih operator deteksi tepi yang diinginkan. Setelah itu baru melakukan deteksi tepi dengan menekan tombol yang telah disediakan. Form hasil deteksi tepi dapat digambarkan seperti dalam Gambar 4.2 dan Gambar 4.3 dibawah ini:

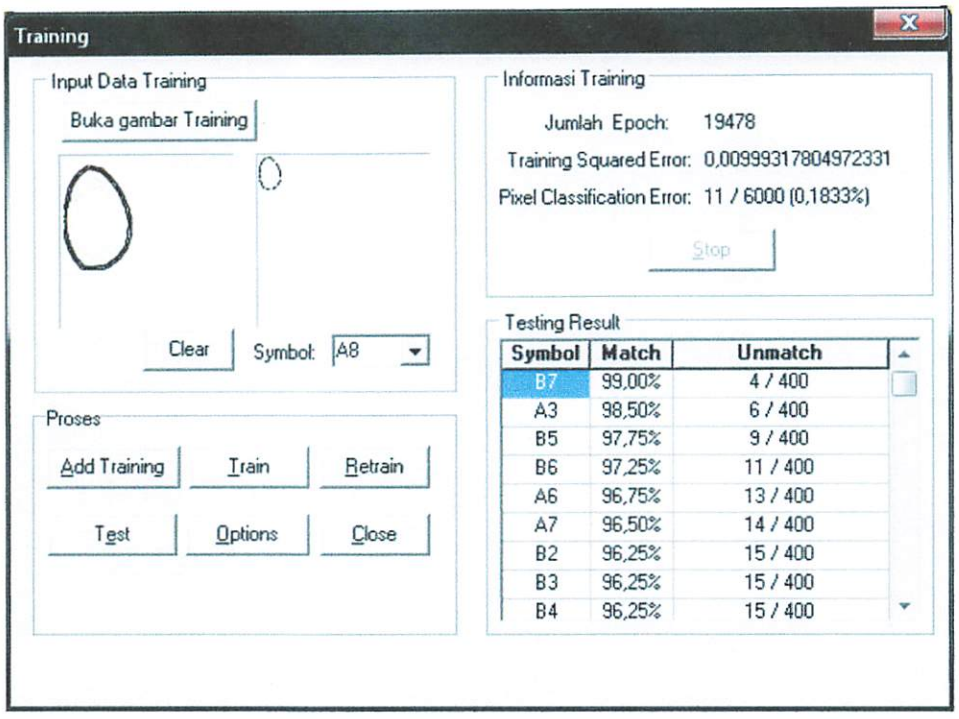


Gambar 4.2. Form Deteksi Tepi

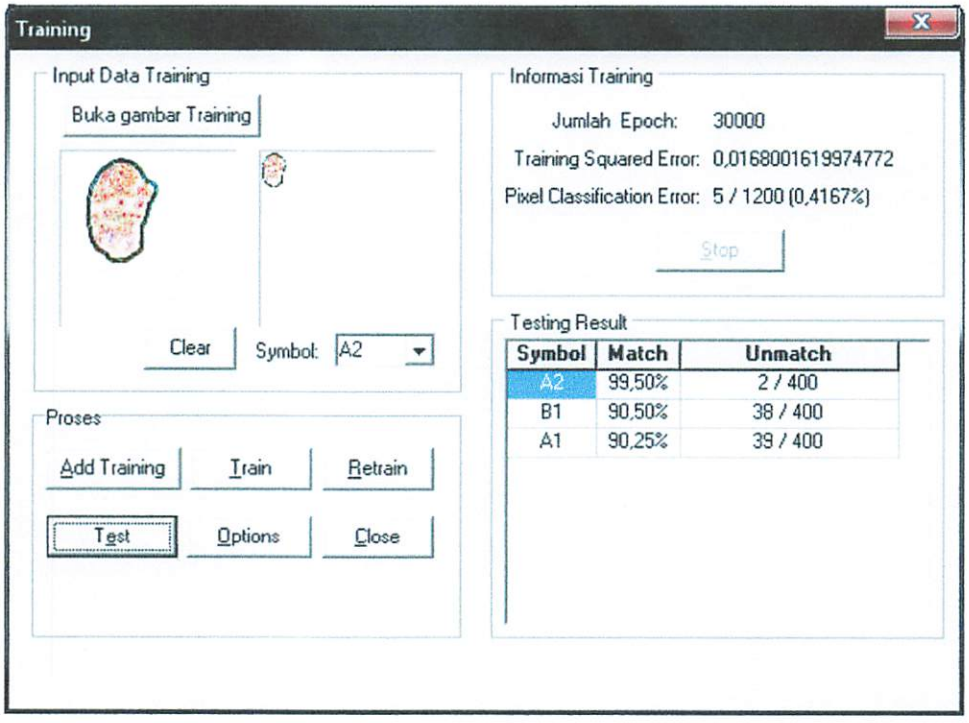


Gambar 4.3. Hasil Deteksi Tepi

Untuk proses memasukkan data training, user harus memasukkan pola golongan darah yang ingin di training, lalu menambahkan pola golongan darah tersebut ke dalam data pelatihan dengan cara menekan tombol *add training*, setelah itu untuk melakukan training user menekan tombol train yang telah disediakan. Form untuk memasukkan data training seperti dalam Gambar 4.4 dan Gambar 4.5 dibawah ini:

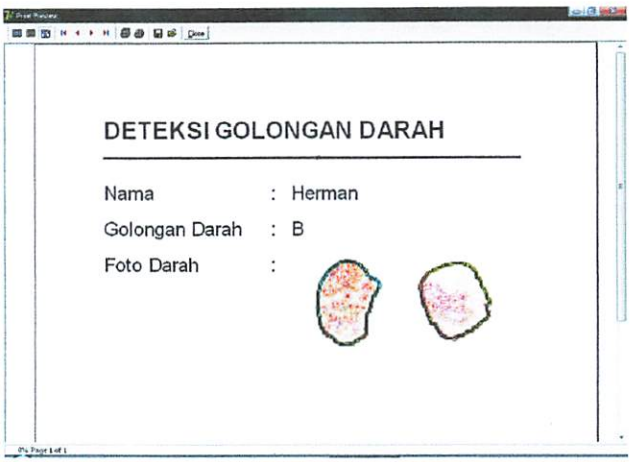


Gambar 4.4. Form Hasil Training dengan Hidden Layer Sebanyak 50



Gambar 4.5. Form Hasil Training dengan Hidden Layer Sebanyak 400

Hasil dari pengenalan golongan darah dapat dicetak. Form untuk mencetak hasil yaitu form Print seperti dalam Gambar 4.6 dibawah ini:



Gambar 4.6. Form Print

4.3 Hasil Pengujian Aplikasi

Uji coba dilakukan untuk mengetahui apakah program dapat berjalan sesuai yang diharapkan dengan lingkungan uji coba yang telah ditentukan dan dilakukan sesuai dengan skenario uji coba. Adapun beberapa hasil uji coba yang telah dilakukan antara lain menguji citra darah yang diperoleh apakah mampu dikenali sebagai golongan darah manusia (A, B, AB dan O)

Pengujian dilakukan menggunakan data yang diperoleh dari penelitian. Gambar yang dimasukkan berekstensi .bmp. Data uji coba yang digunakan sebanyak 8 gambar golongan darah, untuk menghitung persentase dari aplikasi yang dibuat dan dengan data training yang dimasukkan didapatkan rumus:

$$Persentase = \frac{Data\ yang\ berhasil\ dikenali}{jumlah\ data} \times 100\%$$

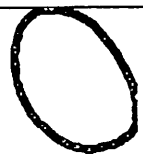
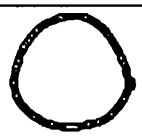
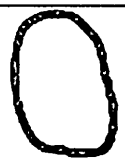
Pengujian pada aplikasi menggunakan skenario percobaan dengan hidden layer default yaitu 50 dan dengan hidden layer 400.

1. Data pelatihan dengan hidden layer 50:

Recognition		**Training**		**Neural Net**	
B&W Threshold	196	Learning Rate	0.001	Input Pattern	
Noise Threshold	10	Weights Initialization Factor	0.5	Height	20
Space Width	22	Error Threshold	1	Width	20
Input Neuron	400	Target Classification Error	-1	Target Pattern	
Hidden Neuron	50	Target Squared Error	0.01	Height	20
Output Neuron	400	Maximum Epoch	10000	Width	20

Data A1	Informasi Training	
	Jumlah Epoch	4477
	Training Squared Error	0.00997
	Pixel Classification error	1/400 (0.25%)
Data A2	Informasi Training	
	Jumlah Epoch	7597
	Training Squared Error	0.00999
	Pixel Classification Error	1/800 (0.08%)
Data A3	Informasi Training	
	Jumlah Epoch	9795
	Training Squared Error	0.00999
	Pixel Classification Error	2/2000 (0.1%)
Data A4	Informasi Training	
	Jumlah Epoch	12071
	Training Squared Error	0.00999
	Pixel Classification Error	4/2800 (0.14%)

Data A5	Informasi Training	
	Jumlah Epoch	15025
	Training Squared Error	0.00999
	Pixel Classification Error	7/3600 (0.19%)
Data A6	Informasi Training	
	Jumlah Epoch	16854
	Training Squared Error	0.00999
	Pixel Classification Error	8/4400 (0.18%)
Data A7	Informasi Training	
	Jumlah Epoch	18334
	Training Squared Error	0.00999
	Pixel Classification Error	9/5200 (0.17%)
Data A8	Informasi Training	
	Jumlah Epoch	19478
	Training Squared Error	0.00999
	Pixel Classification Error	11/6000(0.18%)
Data B1	Informasi Training	
	Jumlah Epoch	6151
	Training Squared Error	0.00999
	Pixel Classification Error	1/800 (0.125%)
Data B2	Informasi Training	
	Jumlah Epoch	8891
	Training Squared Error	0.00999
	Pixel Classification Error	2/1600 (0.125%)
Data B3	Informasi Training	
	Jumlah Epoch	10982
	Training Squared Error	0.00999
	Pixel Classification Error	4/2400 (0.16%)

Data B4	Informasi Training	
	Jumlah Epoch	14000
	Training Squared Error	0.00999
	Pixel Classification Error	7/3200 (0.21%)
Data B5	Informasi Training	
	Jumlah Epoch	16192
	Training Squared Error	0.00999
	Pixel Classification Error	8/4000 (0.2%)
Data B6	Informasi Training	
	Jumlah Epoch	17691
	Training Squared Error	0.00999
	Pixel Classification Error	10/4800 (0.2%)
Data B7	Informasi Training	
	Jumlah Epoch	18708
	Training Squared Error	0.00999
	Pixel Classification Error	11/5600 (0.19%)
Data B8	Informasi Training	
	Jumlah Epoch	20282
	Training Squared Error	0.00999
	Pixel Classification Error	13/6400 (0.2%)

Hasil pelatihan data tersebut diujikan pada gambar golongan darah itu sendiri (*self test*). Dari sample golongan darah tersebut, yang berhasil dikenali sebanyak 3 sample. Jika digunakan rumus perhitungan persentase maka didapatkan :

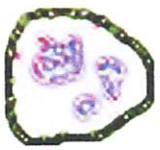
$$Persentase = \frac{3}{8} \times 100\% = 37.5\ \%$$

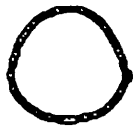
Dari hasil perhitungan didapatkan bahwa validasi dari pengujian dengan hidden layer sebanyak 50 adalah 37.5 %.

2. Data pelatihan dengan hidden layer 400.

Recognition		Training		Neural Net	
B&W Threshold	196	Learning Rate	0.001	Input Pattern	
Noise Threshold	10	Weights Initialization Factor	0.5	Height	20
Space Width	22	Error Threshold	1	Width	20
Input Neuron	400	Target Classification Error	-1	Target Pattern	
Hidden Neuron	400	Target Squared Error	0.01	Height	20
Output Neuron	400	Maximum Epoch	10000	Width	20

Data A1	Informasi Training	
	Jumlah Epoch	10000
	Training Squared Error	0.090
	Pixel Classification error	9/400 (2.25%)
Data A2	Informasi Training	
	Jumlah Epoch	30000
	Training Squared Error	0.033
	Pixel Classification Error	10/1200 (0.83%)
Data A3	Informasi Training	
	Jumlah Epoch	50000
	Training Squared Error	0.011
	Pixel Classification Error	8/2000 (0.4%)
Data A4	Informasi Training	
	Jumlah Epoch	70000
	Training Squared Error	0.018
	Pixel Classification Error	13/2800 (0.46%)
Data A5	Informasi Training	
	Jumlah Epoch	90000
	Training Squared Error	0.020
	Pixel Classification Error	21/3600 (0.58%)
Data A6	Informasi Training	
	Jumlah Epoch	110000
	Training Squared Error	0.010
	Pixel Classification Error	11/4400 (0.25%)

Data A7	Informasi Training	
	Jumlah Epoch	130000
	Training Squared Error	0.009
	Pixel Classification Error	13/5200 (0.25%)
Data A8	Informasi Training	
	Jumlah Epoch	150000
	Training Squared Error	0.010
	Pixel Classification Error	16/6000 (0.26%)
Data B1	Informasi Training	
	Jumlah Epoch	20000
	Training Squared Error	0.030
	Pixel Classification Error	6/800 (0.75%)
Data B2	Informasi Training	
	Jumlah Epoch	40000
	Training Squared Error	0.013
	Pixel Classification Error	7/1600 (0.43%)
Data B3	Informasi Training	
	Jumlah Epoch	60000
	Training Squared Error	0.011
	Pixel Classification Error	7/2400 (0.29%)
Data B4	Informasi Training	
	Jumlah Epoch	80000
	Training Squared Error	0.018
	Pixel Classification Error	15/3200 (0.46%)
Data B5	Informasi Training	
	Jumlah Epoch	100000
	Training Squared Error	0.013
	Pixel Classification Error	13/4000 (0.32%)

Data B6	Informasi Training	
	Jumlah Epoch	120000
	Training Squared Error	0.010
	Pixel Classification Error	13/4800 (0.27%)
Data B7	Informasi Training	
	Jumlah Epoch	140000
	Training Squared Error	0.010
	Pixel Classification Error	14/5600 (0.25%)
Data B8	Informasi Training	
	Jumlah Epoch	160000
	Training Squared Error	0.011
	Pixel Classification Error	19/6400 (0.29%)

Hasil pelatihan data tersebut diujikan pada gambar golongan darah itu sendiri (*self test*). Dari sample golongan darah tersebut, yang berhasil dikenali sebanyak 7 sample. Jika digunakan rumus perhitungan persentase maka didapatkan :

$$Persentase = \frac{7}{8} \times 100\% = 87.5\ \%$$

Dari hasil perhitungan didapatkan bahwa validasi dari pengujian dengan hidden layer sebanyak 400 adalah 87.5 %.

Dari dua buah hasil pengujian tersebut dapat disimpulkan bahwa terdapat beberapa hal yang dapat mempengaruhi hasil yang didapatkan pada aplikasi ini. Hal ini dapat dikarenakan oleh beberapa factor yaitu :

1. Hidden Layer yang diinputkan sangat berpengaruh pada hasil yang didapatkan, semakin banyak hidden layer yang dimasukkan maka hasil yang didapatkan semakin valid.
2. Jumlah data pola golongan darah yang dimasukkan berpengaruh terhadap hasil karna semakin banyak data yang dimasukkan maka semakin rumit training yang dilakukan sehingga perubahan pola menjadi sangat sensitif.
3. Pola golongan darah yang dimasukkan berpengaruh pada hasil, dikarenakan pencahayaan dan hasil dari pengambilan darah berbeda-beda.

BAB V
PENUTUP

5.1. Kesimpulan

Dari perancangan dan pembuatan aplikasi pengenalan golongan darah menggunakan jaringan syaraf tiruan ini dapat diambil kesimpulan sebagai berikut :

1. Dengan hidden layer sebanyak 400 dibutuhkan waktu sekitar 3 sampai 4 menit melakukan pelatihan pembelajaran satu buah data.
2. Semakin banyak data yang dimasukkan ke dalam pelatihan pembelajaran, maka waktu yang dibutuhkan semakin lama.
3. Aplikasi belum dapat secara maksimal mengenali golongan darah sesuai dengan data pelatihan yang diajarkan.
4. Hasil yang didapatkan dipengaruhi oleh beberapa faktor salah satunya yaitu bergantung pada jumlah data dan jumlah hidden layer yang digunakan.
5. Kekurang akuratan hasil juga dipengaruhi oleh data pelatihan yang dimasukkan hanya dua jenis yaitu anti A dan anti B.
6. Data pelatihan baru yang di-*update* ke dalam sistem dapat langsung berpengaruh pada hasil pengenalan golongan darah.

5.1. Saran

Berikut ini adalah saran yang diberikan untuk pengembangan aplikasi selanjutnya :

1. Untuk Pengembangan selanjutnya dapat ditambahkan pendeteksian tepi pada saat pengenalan golongan darah agar secara otomatis pola golongan darah yang dimasukkan dapat langsung dikenali.
2. Untuk data pelatihan pembelajaran backpropagation ditambahkan pengenalan untuk antigen AB agar keakuratan hasil yang didapatkan bisa maksimal.
3. Dari Aplikasi ini perlu ditambahkan form kamera untuk mengambil data secara langsung melalui aplikasi

DAFTAR PUSTAKA

[1] **Anita Desiani dan Muhammad Arhami.** Konsep Kecerdasan Buatan : Penerbit Andi Yogyakarta. 2006

[2] **Mauridhi Hery Purnomo dan Agus Kurniawan.** *Supervised Neural Networks.* Yogyakarta : Gava Media Yogyakarta.2008.

[3] **Munir Rinaldi**. *Pengolahan Citra Digital Dengan Pendekatan Algoritmik.* Bandung: Penerbit Informatika. 2004

[4] **Sigit Riyanto dkk**. *Step By Step* Pengolahan Citra digital. Surabaya: Penerbit Andi. 2005.

[5] **Suyanto, ST, Msc.** *Artificial Intelligence.* Jakarta : Elex Media Komputindo, Jakarta 2008.

[6] **T.Sutoyo, Edy Mulyanto, Vincent Suhartono, Oky Dwi Nurhayati, Wijanarto.** Teori Pengolahan Citra Digital. Yogyakarta : Penerbit Andi Yogyakarta. 2009.

[7] http://id.wikipedia.org/w/index.php?title=Pengolahan_Citra&redirect=no diakses pada tanggal 12 Juli 2010 Pukul 19.00 WIB

[8] **Riyanto**. Deteksi Golongan Darah Manusia : http://lecturer.eepis-its.edu/~riyanto/golda.html diakses pada tanggal 11 Mei 2010 Pukul 20.00 WIB

PERKUMPULAN PENGELOLA PENDIDIKAN UMUM DAM TEKNOLOGI NASIONAL MALANG
INSTITUT TEKNOLOGI NASIONAL MALANG
FAKULTAS TEKNOLOGI INDUSTRI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
PROGRAM PASCASARJANA MAGISTER TEKNIK
BNI (PERSERO) MALANG
BANK NIAGA MALANG
Kampus I : Jl. Bendungan Sigura-gura No. 2 Telp. (0341) 551431 (Hunting), Fax. (0341) 553015 Malang 65145
Kampus II : Jl. Raya Karanglo, Km 2 Telp. (0341) 417636 Fax. (0341) 417634 Malang

BERITA ACARA UJIAN SKRIPSI
FAKULTAS TEKNOLOGI INDUSTRI

NAMA	:	ICHWAN SETIAWAN
NIM	:	05.12.509
JURUSAN	:	TEKNIK ELEKTRO S-1
KONSENTRASI	:	TEKNIK KOMPUTER DAN INFORMATIKA
JUDUL SKRIPSI	:	APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES PENGENALAN GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

Dipertahankan dihadapan Tim Penguji Ujian Skripsi jenjang program Strata Satu (S-1)

Pada Hari	:	Selasa
Tanggal	:	24 Agustus 2010
Dengan nilai	:	B+ (77)

PANITIA UJIAN SKRIPSI

KETUA

Ir. Yusuf Ismail Nakhoda, MT
NIP.Y. 1018800189

ANGGOTA PENGUJI

Dosen Penguji I

M. Ibrahim Ashari, ST, MT.
NIP.P.1030100358

Dosen Penguji II

Sotyohadi, ST
NIP.Y.1039700309

PERKUMPULAN PENGELOLA PENDIDIKAN UMUM DAM TEKNOLOGI NASIONAL MALANG
INSTITUT TEKNOLOGI NASIONAL MALANG
FAKULTAS TEKNOLOGI INDUSTRI
FAKULTAS TEKNIK SIPIL DAN PERENCANAAN
PROGRAM PASCASARJANA MAGISTER TEKNIK
BNI (PERSERO) MALANG
BANK NIAGA MALANG
Kampus I : Jl. Bendungan Sigura-gura No. 2 Telp. (0341) 551431 (Hunting), Fax. (0341) 553015 Malang 65145
Kampus II : Jl. Raya Karanglo, Km 2 Telp. (0341) 417636 Fax. (0341) 417634 Malang

FORMULIR PERBAIKAN SKRIPSI

Nama	:	ICHWAN SETIAWAN
NIM	:	05.12.509
Jurusan	:	Teknik Elektro S-1
Konsentrasi	:	Teknik Komputer & Informatika
Masa Bimbingan	:	30 Mei 2010 s/d 30 Oktober 2010
Judul Skripsi	:	APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES PENGENALAN GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

PENGUJI	TANGGAL	URAIAN REVISI	PARAF
Penguji 1	24 Agustus 2010	1. Kata pengantar pada gambar dan tabel 2. Daftar pustaka urut sesuai abjad	

Disetujui

Penguji I

M. Ibrahim Ashari, ST, MT.
NIP.Y. 1030100358

Mengetahui

Dosen Pembimbing

Irmalia Suryani Faradisa, ST, MT
NIP. Y.1030000365

INSTITUT TEKNOLOGI NASIONAL
Jl. Raya Karanglo 10 KM
MALANG

FORMULIR BIMBINGAN SKRIPSI

Nama : ICHWAN SETIAWAN
NIM : 05.12.509
Masa Bimbingan : 30 Mei 2010 s/d 30 Oktober 2010
Judul Skripsi : APLIKASI PENGENALAN GOLONGAN DARAH UNTUK PROSES IDENTIFIKASI GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

No	Tanggal	Uraian	Paraf Pembimbing
1	16 Juli 2010	Asistensi BAB I, Revisi Latar Belakang	
2	22 Juli 2010	Asistensi BAB III, Revisi *Flowchart* Deteksi Tepi	
3	26 Juli 2010	Asistensi BAB IV, Revisi BAB IV	
4	30 Juli 2010	Asistensi BAB IV, Revisi Pengujian	
5	01 Agustus 2010	ACC BAB I, BAB II, BAB III	
6	03 Agustus 2010	ACC BAB IV, BAB V	
7	07 Agustus 2010	ACC Makalah Seminar Hasil	
8	20 Agustus 2010	ACC Ujian Skripsi	

Malang, 24 Agustus 2010
Dosen Pembimbing

(**Irmalia Suryani Faradisa ST, MT**)
NIP.Y 1030000365

FORM S-4b

Kepada Yth. :

Ketua Jurusan Teknik Elektro
Fakultas Teknologi Industri
Institut Teknologi Nasional Malang

Di Tempat,-

Yang bertanda tangan di bawah ini :

Nama Mahasiswa	: Ichwan Setiawan
NIM	: 05.12.509
Fakultas	: Teknologi Industri
Jurusan	: Teknik Elektro
Konsentrasi	: Teknik Komputer dan Informatika
Angkatan	: 2005

Dengan ini mengajukan permohonan untuk pembatalan judul/bimbingan skripsi, dikarenakan adanya kendala dan hambatan yang tidak memungkinkan untuk melanjutkan pengerjaan skripsi, dengan judul skripsi :

DESAIN APLIKASI MOBILE ONLINE PUBLIC ACCESS CATALOG (OPAC) MENGGUNAKAN TEKNOLOGI .NET

Demikian isi surat permohonan ini kami buat agar dapat dipergunakan seperlunya.

Malang, 29 Maret 2010

Mengetahui
Dosen Pembimbing

Ir. F. Yudi Limpraptono, MT
NIP. Y. 1039 5900274

Yang Menyatakan,

Ichwan Setiawan
NIM. 05.12.509

Institut Teknologi Nasional Malang
Fakultas Teknologi Industri
Jurusan Teknik Elektro S-1

LEMBAR PENGAJUAN JUDUL SKRIPSI
JURUSAN TEKNIK ELEKTRO S-1

Konsentrasi : Teknik Energi Listrik/Teknik Elektronika/Teknik Komputer & Informatika*)

1.	Nama Mahasiswa: ICHWAN SETIAWAN			Nim: 0512509
2.	Waktu Pengajuan	Tanggal:	Bulan:	Tahun:
3.	Spesifikasi Judul (berilah tanda silang)**) a. Sistem Tenaga Elektrik b. Energi & Konversi Energi c. Tegangan Tinggi & Pengukuran d. Sistem Kendali Industri e. Elektronika & Komponen f. Elektronika Digital & Komputer g. Elektronika Komunikasi X lainnya			
4.	Konsultasikan judul sesuai materi bidang ilmu kepada Dosen*) Irmaka F, ST MT Ketua Jurusan Ir. F. Yudi Limpraptono,MT NIP. P. 1039500274			
5.	Judul yang diajukan mahasiswa:	Aplikasi Pengenalan Golongan Darah Untuk Mempercepat Identifikasi Golongan Darah Menggunakan metode Back Propagation Jaringan syaraf Tiruan		
6.	Perubahan judul yang disetujui Dosen sesuai materi bidang ilmu	.....		
7.	Catatan: Penelitian. Metode BP. dibandingkan dg YG lain kelebihan apa dimasukkan. Latar blkg. Blook Diagram pengolahan BP ..... Gbr 2 diperjelas. Persetujuan Judul skripsi yang dikonsultasikan kepada Dosen materi bidang ilmu	Disetujui 24 - 4 2010 Dosen Irmaka ST.		

Perhatian:

1. Formulir pengajuan ini harap dikembalikan kepada jurusan paling lambat satu minggu setelah disetujui kelompok dosen keahlian dengan dilampirkan proposal skripsi beserta persyaratan skripsi sesuai form S-1
2. Keterangan: *) Coret yang tidak perlu
   **) dilingkari a, b, c, ...atau g sesuai bidang keahlian

Form S-2

INSTITUT TEKNOLOGI NASIONAL
Jl. Bendungan Sigura-gura No.2
MALANG

Lampiran : 1 (satu) berkas
Pembimbing Skripsi

Kepada : Yth. Irmalia Suryani Faradisa ST,MT
Dosen Institut Teknologi Nasional
MALANG

Yang bertanda tangan di bawah ini :

Nama	: ICHWAN SETIAWAN
Nim	: 05.12.509
Jurusan	: Teknik Elektro S-1
Konsentrasi	: Teknik Komputer & Informatika

Dengan ini mengajukan permohonan, kiranya Ibu bersedia menjadi Dosen Pembimbing Utama / ~~Pendamping~~ *), untuk penyusunan Skripsi dengan judul (proposal terlampir) :

APLIKASI PENGENALAN GOLONGAN DARAH UNTUK MEMPERCEPAT IDENTIFIKASI GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

Adapun tugas tersebut sebagai salah satu syarat untuk menempuh Ujian Akhir Sarjana Teknik.
Demikian permohonan kami dan atas kesediaan Bapak kami ucapkan terima kasih.

Malang, April 2010

Mengetahui
Ketua Jurusan Teknik Elektro

Ir.F.Yudi Limpraptono, MT
NIP.Y. 10395900274

Hormat kami,

Ichwan Setiawan

Catatan :
*) Coret yang tidak perlu

Form S-3 a

INSTITUT TEKNOLOGI NASIONAL
Jl. Bendungan Sigura-gura No.2
MALANG

PERNYATAAN KESEDIAAN DALAM PEMBIMBING SKRIPSI

Sesuai permohonan dari mahasiswa :

Nama : ICHWAN SETIAWAN

Nim : 05.12.509

Semester : 10 (Sepuluh)

Jurusan : Teknik Elektro S-1

Konsentrasi : Teknik Komputer & Informatika

Dengan ini Menyatakan bersedia / tidak bersedia *) Membimbing Skripsi dari mahasiswa tersebut, dengan judul :

APLIKASI PENGENALAN GOLONGAN DARAH UNTUK MEMPERCEPAT IDENTIFIKASI GOLONGAN DARAH MENGGUNAKAN METODE JARINGAN SYARAF TIRUAN BACKPROPAGATION

Demikian surat pernyataan ini kami buat agar dapat dipergunakan seperlunya.

Malang, April 2010

Kami yang membuat pernyataan,

Irmalia Suryani Faradisa ST,MT

NIP.Y. 1030000365

Catatan :
Setelah disetujui formulir
Diserahkan mahasiswa yang bersangkutan
Kepada Jurusan untuk diproses lebih lanjut
*) Coret yang tidak perlu

Form S-3 b

{**

Class TBackProp

File Name : BackPropCls.pas

Language : Delphi 2005

Author : Theo Zacharias (theo_yz@yahoo.com)

Description : TBackProp is a class that encapsulates a standard
backpropagation neural net object.

Public Members:

* Events : OnTraining, OnTrainingFinish
* Properties: ErrorThreshold (r/w), InputLayer (r/w),
InputPatternHeight (r/o), InputPatternWidth (r/o),
KnownSymbol (r/o), LearningRate (r/w), MaxEpoch (r/w),
Modified (r/o), NHiddenNeuron (r/o), NInputNeuron (r/o),
NNeuronError (r/o), NOutputNeuron (r/o), NTrainingEpoch (r/o),
NTrainingNeuron (r/o), NTrainingPair (r/o), OutputLayer (r/o),
StopTraining (r/w), TargetClassificationError (r/w),
TargetPatternHeight (r/o), TargetPatternWidth (r/0),
TargetSquaredError (r/o), TrainingError (r/o),
WeightsInitFactor (r/w)
* Methods : AddTrainingPairs, Apply, GetResult, NewKnowledge,
OpenKnowledge, Retrain, SaveKnowledge, Train

Last modified on June 25, 2005

**}

```
unit BackPropCls;

interface

uses Classes, Graphics;

const
  //- Network architecture-
  DEFAULT_INPUT_PATTERN_HEIGHT: Integer = 20;
  DEFAULT_INPUT_PATTERN_WIDTH = 20;
  DEFAULT_TARGET_PATTERN_HEIGHT = 20;
  DEFAULT_TARGET_PATTERN_WIDTH = 20;
  DEFAULT_NUMBER_OF_HIDDEN_NEURON = 50;

  //- Training set parameters-
  DEFAULT_LEARNING_RATE = 0.001;
  DEFAULT_MAX_EPOCH = 10000;
  DEFAULT_TARGET_CLASSIFICATION_ERROR = -1;
  DEFAULT_TARGET_SQUARED_ERROR = 0.01;
  DEFAULT_ERROR_THRESHOLD = 1;
  DEFAULT_WEIGHTS_INIT_FACTOR = 0.5;

type
  //- Events -
  TTrainingEvent = procedure(Sender: TObject) of object;
  TTrainingFinishEvent = procedure(Sender: TObject) of object;

  //- Neurons and their weights -
  TNeuron = Single;
  TLayer = array of TNeuron;
  TWeight = Single;
```

```
  TLayerWeights = array of array of TWeight;

  //- Training set -
  TPatternClass = string;
  TTrainingPairs = record
    InputPatterns: array of TLayer;
    TargetPattern: TLayer;
    TargetPatternClass: TPatternClass;
  end;
  TTrainingSet = array of TTrainingPairs;

type
  TBackProp = class(TObject)
  //--- Construction/Destruction ---
  public
    constructor Create(NInputNeuron: Integer;
                       NHiddenNeuron: Integer; NOutputNeuron: Integer);
                       overload; virtual;
    constructor Create(InputPatternHeight: Integer; InputPatternWidth: Integer;
                       TargetPatternHeight: Integer;
                       TargetPatternWidth: Integer; NHiddenNeuron: Integer);
                       overload; virtual;
    constructor Create(FileName: string); overload; virtual;

  //--- Events ---
  private
    FOnTraining: TTrainingEvent;
    FOnTrainingFinish: TTrainingFinishEvent;

  public
    property OnTraining: TTrainingEvent read FOnTraining write FOnTraining;
    property OnTrainingFinish: TTrainingFinishEvent
```

```
        read FOnTrainingFinish write FOnTrainingFinish;

//--- Properties ---
private
 //- Neurons and their weights -
 FNInputNeuron: Integer;
 FNHiddenNeuron: Integer;
 FNOutputNeuron: Integer;
 FInputLayer: TLayer;
 FHiddenLayer: TLayer;
 FOutputLayer: TLayer;
 FHiddenLayerWeights: TLayerWeights;
 FOutputLayerWeights: TLayerWeights;

 //- Training set and its parameters -
 FTrainingSet: TTrainingSet;
 FErrorThreshold: Single;
 FLearningRate: Single;
 FMaxEpoch: Integer;          // maximum epoch for training stop condition
 FStopTraining: Boolean;
 FTargetClassificationError: Single;          // target error for training
 FTargetSquaredError: Single;                 //          stop condition
 FWeightsInitFactor: Single;          // for random weights initialization

 //- Patterns size
 FInputPatternHeight: Integer;
 FInputPatternWidth: Integer;
 FTargetPatternHeight: Integer;
 FTargetPatternWidth: Integer;

 //- Training info -
 FNNeuronError: Integer;
```

```
  FNTrainingEpoch: Integer;
  FNTrainingNeuron: Integer;
  FTrainingError: Single;

  //- Others
  FModified: Boolean;                  // indicates whether the weights or
                                       //  training set have been modified

  //- Properties procedure/functions
  function GetKnownSymbol(Index: Integer): TPatternClass; virtual;
  function GetNTrainingPair: Integer; virtual;
  function GetOutputLayer: TLayer; virtual;
  procedure SetLearningRate(Value: Single); virtual;
  procedure SetMaxEpoch(Value: Integer); virtual;
  procedure SetWeightsInitFactor(Value: Single); virtual;

public
  //- Backpropagation layers -
  property InputLayer: TLayer read FInputLayer write FInputLayer;
  property OutputLayer: TLayer read GetOutputLayer;
  property NInputNeuron: Integer read FNInputNeuron;
  property NHiddenNeuron: Integer read FNHiddenNeuron;
  property NOutputNeuron: Integer read FNOutputNeuron;

  //- Training set parameters -
  property ErrorThreshold: Single read FErrorThreshold write FErrorThreshold;
  property LearningRate: Single read FLearningRate write SetLearningRate;
  property MaxEpoch: Integer read FMaxEpoch write SetMaxEpoch;
  property StopTraining: Boolean read FStopTraining write FStopTraining;
  property TargetClassificationError: Single read FTargetClassificationError
                                      write FTargetClassificationError;
  property TargetSquaredError: Single read FTargetSquaredError
```

```
                                write FTargetSquaredError;
    property WeightsInitFactor: Single read FWeightsInitFactor
                                write SetWeightsInitFactor;

    //- Patterns size -
    property InputPatternHeight: Integer read FInputPatternHeight;
    property InputPatternWidth: Integer read FInputPatternWidth;
    property TargetPatternHeight: Integer read FTargetPatternHeight;
    property TargetPatternWidth: Integer read FTargetPatternWidth;

    //- Training info -
    property KnownSymbol[Index: Integer]: TPatternClass read
GetKnownSymbol;
    property NNeuronError: Integer read FNNeuronError;
    property NTrainingEpoch: Integer read FNTrainingEpoch;
    property NTrainingNeuron: Integer read FNTrainingNeuron;
    property NTrainingPair: Integer read GetNTrainingPair;
    property TrainingError: Single read FTrainingError;

    //- Others -
    property Modified: Boolean read FModified;

  //--- Methods ---
  private
    function BipolarSigmoid(x: Single): Single;
    function BipolarSigmoidDerivation(Fx: Single): Single; overload;
    procedure InitLayers; virtual;
    procedure InitWeights; virtual;
    procedure UpdateNTrainingNeuron; virtual;

  public
    //- Training -
```

```
  procedure AddTrainingPairs(TrainingPairs: TTrainingPairs); virtual;
  procedure Retrain; virtual;
  procedure Train; virtual;

  //- Testing -
  procedure Apply; virtual;
  procedure GetResult(out Symbol: TPatternClass); overload; virtual;
  procedure GetResult(var Symbols: TStringList;
                      var Matchs: TStringList;
                      var Unmatchs: TStringList); overload; virtual;

  //- Knowledge -
  procedure NewKnowledge; overload; virtual;
  procedure NewKnowledge(
            InputPatternHeight: Integer; InputPatternWidth: Integer;
            TargetPatternHeight: Integer; TargetPatternWidth: Integer;
            NHiddenNeuron: Integer); overload; virtual;
  procedure OpenKnowledge(FileName: string); virtual;
  procedure SaveKnowledge(FileName: string); virtual;

 //--- Class Methods ---
 public
  class procedure BitmapToLayer(Bitmap: TBitmap; Layer: TLayer);
 end;

implementation

uses Forms, SysUtils;

//----------------------------
// Construction/Destruction
//----------------------------
```

```
// Purpose    : Creates and initializes a new backpropagation object by
//              defining the size of input layer, hidden layer and output layer
// Inputs     : * NInputNeuron (number of neurons, not including bias, in input
//                       layer; >0)
//              * NHiddenNeuron (number of neurons, not including bias, in
//                       hidden layer; >0)
//              * NOutputNeuron (number of neurons, not including bias, in
//                       output layer; >0)
constructor TBackProp.Create(NInputNeuron: Integer;
                    NHiddenNeuron: Integer; NOutputNeuron: Integer);
begin
  Assert((NInputNeuron > 0) and (NHiddenNeuron > 0) and (NOutputNeuron >
0));

  inherited Create;

  // Initializes layers
  FNInputNeuron := NInputNeuron;
  FNHiddenNeuron := NHiddenNeuron;
  FNOutputNeuron := NOutputNeuron;
  InitLayers;

  // Sets properties default value
  FErrorThreshold := DEFAULT_ERROR_THRESHOLD;
  FLearningRate := DEFAULT_LEARNING_RATE;
  FMaxEpoch := DEFAULT_MAX_EPOCH;
  FTargetClassificationError :=
DEFAULT_TARGET_CLASSIFICATION_ERROR;
  FTargetSquaredError := DEFAULT_TARGET_SQUARED_ERROR;
  FWeightsInitFactor := DEFAULT_WEIGHTS_INIT_FACTOR;
```

```
  // Initializes weights
  InitWeights;

  // Initializes training info
  FNNeuronError := 0;
  FNTrainingEpoch := 0;
  FNTrainingNeuron := 0;
  FTrainingError := 0;

  // Initializes others
  FModified := False;
end;

// Purpose    : Creates and initializes a new backpropagation object taken
//              from external file
// Inputs     : FileName (must exists)
constructor TBackProp.Create(FileName: string);
begin
  Assert(FileExists(FileName));

  inherited Create;
  OpenKnowledge(FileName);
end;

// Purpose    : Creates and initializes a new backpropagation object by
//              defining the size of input pattern, output pattern and hidden
//              layer
// Inputs     : * InputPatternHeight (>0)
//              * InputPatternWidth (>0)
//              * TargetPatternHeight (>0)
//              * TargetPatternWidth (>0)
//              * NHiddenNeuron (number of neurons, not including bias, in
```

```
//                          hidden layer; >0)
constructor TBackProp.Create(
          InputPatternHeight: Integer; InputPatternWidth: Integer;
          TargetPatternHeight: Integer; TargetPatternWidth: Integer;
          NHiddenNeuron: Integer);
begin
  Assert((InputPatternHeight > 0) and (InputPatternWidth > 0) and
       (TargetPatternHeight > 0) and (TargetPatternWidth > 0) and
       (NHiddenNeuron > 0));

  Create(InputPatternHeight * InputPatternWidth, NHiddenNeuron,
       TargetPatternHeight * TargetPatternWidth);
  FInputPatternHeight := InputPatternHeight;
  FInputPatternWidth := InputPatternWidth;
  FTargetPatternHeight := TargetPatternHeight;
  FTargetPatternWidth := TargetPatternWidth;
end;

//--------------
//  Properties
//--------------

// Purpose    : Returns the known symbol with certain index
// Input      : Index (0 <= Index <= number of symbol - 1)
function TBackProp.GetKnownSymbol(Index: Integer): TPatternClass;
begin
  Assert((0 <= Index) and (Index <= High(FTrainingSet)));

  Result := FTrainingSet[Index].TargetPatternClass;
end;

// Purpose    : Returns the number of training pairs (precisely the number of
```

```
//            known symbol)
function TBackProp.GetNTrainingPair: Integer;
begin
  Result := Length(FTrainingSet);
end;

// Purpose    : Returns the output layer without the bias neuron
function TBackProp.GetOutputLayer: TLayer;
var
  k: Integer;                          // looping index for output layer
begin
  SetLength(Result, FNOutputNeuron);
  for k := 1 to FNOutputNeuron do
    Result[k - 1] := FOutputLayer[k];
end;

// Purpose    : Sets the learning rate
// Input      : Value (0 < Value <= 1)
procedure TBackProp.SetLearningRate(Value: Single);
begin
  Assert((0 < Value) and (Value <= 1));

  FLearningRate := Value;
end;

// Purpose    : Sets the maximum number of epoch for the current training (not
//              for overall training)
// Input      : Value (>0)
procedure TBackProp.SetMaxEpoch(Value: Integer);
begin
  Assert(Value > 0);
```

```
  FMaxEpoch := Value;
end;

// Purpose    : Sets the weights initialize factor
// Input      : Value (>0)
procedure TBackProp.SetWeightsInitFactor(Value: Single);
begin
  Assert(Value > 0);

  FWeightsInitFactor := Value;
end;

//-----------
//  Methods
//-----------

// Purpose    : Adds training pairs to the training set
// Input      : TrainingPairs (the training pairs that will be added; must has
//                             the same size with the input and output layer)
// Effect     : Property HasTrainingSet is set to true
procedure TBackProp.AddTrainingPairs(TrainingPairs: TTrainingPairs);
var
  i, k, l, m: Integer;       // looping index for input pattern, target pattern,
                             //       training set and training input patterns
  IxClass: Integer;                    // index of target pattern class
begin
  for i := 0 to High(TrainingPairs.InputPatterns) do
    Assert(Length(TrainingPairs.InputPatterns[i]) = FNInputNeuron + 1);
  Assert(Length(TrainingPairs.TargetPattern) = FNOutputNeuron + 1);

  //--- Initializes a new training pair slot in the training set
  //- Searches for the training pairs' class in the training set
```

```
IxClass := -1;
for l := 0 to High(FTrainingSet) do
 if FTrainingSet[l].TargetPatternClass =
     TrainingPairs.TargetPatternClass then
 begin
  IxClass := l;
  SetLength(FTrainingSet[IxClass].InputPatterns,
         Length(FTrainingSet[IxClass].InputPatterns) +
          Length(TrainingPairs.InputPatterns), FNInputNeuron + 1);
  Break;
 end;
//- If the training pair's class doesn't exist in the training set, create
//  a new class in the training set
if IxClass = -1 then
begin
 SetLength(FTrainingSet, Length(FTrainingSet) + 1);
 IxClass := High(FTrainingSet);
 SetLength(FTrainingSet[IxClass].InputPatterns,
        Length(TrainingPairs.InputPatterns), FNInputNeuron + 1);
 SetLength(FTrainingSet[IxClass].TargetPattern, FNOutputNeuron + 1);
end;

//-- Adds the training pair to the training set
with FTrainingSet[IxClass] do
begin
 for m := 0 to High(TrainingPairs.InputPatterns) do
  for i := 1 to High(TrainingPairs.InputPatterns[m]) do
   InputPatterns[Length(InputPatterns) -
                 Length(TrainingPairs.InputPatterns) + m,
                i] := TrainingPairs.InputPatterns[m, i];
 if Length(FTrainingSet[IxClass].InputPatterns) =
   Length(TrainingPairs.InputPatterns) then
```

```
  begin
    for k := 1 to High(TrainingPairs.TargetPattern) do
      TargetPattern[k] := TrainingPairs.TargetPattern[k];
    TargetPatternClass := TrainingPairs.TargetPatternClass;
  end;
 end;

 //-- Finishing
 UpdateNTrainingNeuron;
 FModified := True;
end;

// Purpose    : Executes the feedforward phase of the backpropagation
// Assumption : Input layer activation value has been set
// Effect     : A continuous activation value for output layer has been computed
procedure TBackProp.Apply;
var
 i, j, k: Integer;        // looping index for input, hidden and output layer
begin
 //--- Feedforwards from input layer to hidden layer
 for j := 1 to FNHiddenNeuron do
 begin
  //- Sums its weighted input signals from input layer
  FHiddenLayer[j] := 0;
  for i := 0 to FNInputNeuron do
   FHiddenLayer[j] := FHiddenLayer[j] +
                (FInputLayer[i] * FHiddenLayerWeights[j, i]);
  //- Applies the bipolar sigmoid activation function
  FHiddenLayer[j] := BipolarSigmoid(FHiddenLayer[j]);
 end;

 //--- Feedforwards from hidden layer to output layer
```

```
 for k := 1 to FNOutputNeuron do
 begin
  //- Sums its weighted input signals from hidden layer
  FOutputLayer[k] := 0;
  for j := 0 to FNHiddenNeuron do
   FOutputLayer[k] := FOutputLayer[k] +
                (FHiddenLayer[j] * FOutputLayerWeights[k, j]);
  //- Applies the bipolar sigmoid activation function
  FOutputLayer[k] := BipolarSigmoid(FOutputLayer[k]);
 end;
end;

// Purpose    : Gets the symbol with higest percentage of match with the pattern
//              in output layer
// Assumption : Output layer activation value has been computed
procedure TBackProp.GetResult(out Symbol: TPatternClass);
var
 Symbols, Matchs, Unmatchs: TStringList;
begin
 Symbols := TStringList.Create;
 Matchs := TStringList.Create;
 Unmatchs := TStringList.Create;
 GetResult(Symbols, Matchs, Unmatchs);
 if Symbols.Count > 0 then
  Symbol := Symbols.Strings[0]
 else
  Symbol := '';
 Unmatchs.Free;
 Matchs.Free;
 Symbols.Free;
end;
```

```
// Purpose    : Gets all symbols with their percentage of match and unmatch
//              information with the pattern in output layer
// Assumption : Output layer activation value has been computed
procedure TBackProp.GetResult(var Symbols: TStringList; var Matchs:
TStringList;
                              var Unmatchs: TStringList);
var
  HammingDistance: Integer;
  k, l: Integer;   // looping index for training input patterns and training set
  MatchPercentage: Single;
  Pos: Integer;                  // position to insert the processed symbol
begin
  for l := 0 to High(FTrainingSet) do
  begin
    //--- Gets the hamming distance
    HammingDistance := 0;
    for k := 1 to FNOutputNeuron do
      if Abs(FTrainingSet[l].TargetPattern[k] - FOutputLayer[k])
        >= FErrorThreshold then
        HammingDistance := HammingDistance + 1;

    //--- Counts the match percentage
    MatchPercentage := (1 - (HammingDistance / FNOutputNeuron)) * 100;

    //--- Inserts to the output variables
    Pos := 0;
    while (Pos < l) do
      if (StrToFloat(Matchs[Pos]) < MatchPercentage) then
        Break
      else
        Pos := Pos + 1;
    Symbols.Insert(Pos, FTrainingSet[l].TargetPatternClass);
```

```
    Matchs.Insert(Pos, FloatToStrF(MatchPercentage, ffFixed, 4, 2));
    Unmatchs.Insert(Pos, IntToStr(HammingDistance) + ' / ' +
                    IntToStr(FNOutputNeuron) + ' ');
  end;
  for Pos := 0 to Matchs.Count - 1 do
    Matchs[Pos] := Matchs[Pos] + '%'
end;

// Purpose    : Creates a new knowledge and maintains the network architecture
// Effects    : * The object has no training set
//              * The training info has been reset
//              * The object is not modified
procedure TBackProp.NewKnowledge;
begin
  //--- New (no) training set
  SetLength(FTrainingSet, 0);

  //--- New (init) the training info
  FNNeuronError := 0;
  FNTrainingEpoch := 0;
  FTrainingError := 0;
  FNTrainingNeuron := 0;

  FModified := False;
end;

// Purpose    : Creates a new knowledge and new network architecture
// Inputs     : * InputPatternHeight (>0)
//              * InputPatternWidth (>0)
//              * TargetPatternHeight (>0)
//              * TargetPatternWidth (>0)
//              * NHiddenNeuron (>0)
```

```
// Effects    : * The object has a new network architecture
//             * The weights have been initialized
//             * The object has no training set
//             * The training info has been reset
//             * The object is not modified
procedure TBackProp.NewKnowledge(
          InputPatternHeight: Integer; InputPatternWidth: Integer;
          TargetPatternHeight: Integer; TargetPatternWidth: Integer;
          NHiddenNeuron: Integer);
begin
  Assert((InputPatternHeight > 0) and (InputPatternWidth > 0) and
       (TargetPatternHeight > 0) and (TargetPatternWidth > 0) and
       (NHiddenNeuron > 0));

  //--- New network architecture
  FInputPatternHeight := InputPatternHeight;
  FInputPatternWidth := InputPatternWidth;
  FTargetPatternHeight := TargetPatternHeight;
  FTargetPatternWidth := TargetPatternWidth;
  FNHiddenNeuron := NHiddenNeuron;
  FNInputNeuron := FInputPatternHeight * FInputPatternWidth;
  FNOutputNeuron := FTargetPatternHeight * FTargetPatternWidth;
  InitLayers;

  //--- New (init) weights
  InitWeights;

  NewKnowledge;
end;

// Purpose    : Opens a new knowledge from external file
// Input      : FileName (must exist)
```

```
procedure TBackProp.OpenKnowledge(FileName: string);
var
 DataSize: Integer;                          // temporary variable
 FileStream: TFileStream;
 i, j, k, l, m, n: Integer;    // looping index for input layer, hidden layer,
                          //              output layer, training set,
                          //          training input patterns and other
begin
 Assert(FileExists(FileName));

 FileStream := TFileStream.Create(FileName, fmOpenRead or
fmShareDenyWrite);
 with FileStream do
 begin
  try
   //--- Opens the network architecture
   Read(FInputPatternHeight, SizeOf(Integer));
   Read(FInputPatternWidth, SizeOf(Integer));
   Read(FTargetPatternHeight, SizeOf(Integer));
   Read(FTargetPatternWidth, SizeOf(Integer));
   Read(FNHiddenNeuron, SizeOf(Integer));
   FNInputNeuron := FInputPatternHeight * FInputPatternWidth;
   FNOutputNeuron := FTargetPatternHeight * FTargetPatternWidth;
   InitLayers;

   //--- Opens the weights
   for j := 1 to FNHiddenNeuron do
    for i := 0 to FNInputNeuron do
     Read(FHiddenLayerWeights[j, i], SizeOf(TWeight));
   for k := 1 to FNOutputNeuron do
    for j := 0 to FNHiddenNeuron do
     Read(FOutputLayerWeights[k, j], SizeOf(TWeight));
```

```
//--- Opens the training set
Read(DataSize, SizeOf(Integer));
SetLength(FTrainingSet, DataSize);
for l := 0 to High(FTrainingSet) do
begin
  Read(DataSize, SizeOf(Integer));
  SetLength(FTrainingSet[l].InputPatterns, DataSize, FNInputNeuron + 1);
  for m := 0 to High(FTrainingSet[l].InputPatterns) do
    for i := 1 to High(FTrainingSet[l].InputPatterns[m]) do
      Read(FTrainingSet[l].InputPatterns[m, i], SizeOf(TNeuron));
  SetLength(FTrainingSet[l].TargetPattern, FNOutputNeuron + 1);
  for k := 1 to High(FTrainingSet[l].TargetPattern) do
    Read(FTrainingSet[l].TargetPattern[k], SizeOf(TNeuron));
  Read(DataSize, SizeOf(Integer));
  SetLength(FTrainingSet[l].TargetPatternClass, DataSize);
  for n := 1 to DataSize do
    Read(FTrainingSet[l].TargetPatternClass[n], SizeOf(Char));
end;

//--- Opens the training parameters
Read(FLearningRate, SizeOf(Single));
Read(FWeightsInitFactor, SizeOf(Single));
Read(FErrorThreshold, SizeOf(Single));
Read(FTargetClassificationError, SizeOf(Single));
Read(FTargetSquaredError, SizeOf(Single));
Read(FMaxEpoch, SizeOf(Integer));

//--- Opens the training info
Read(FNNeuronError, SizeOf(Integer));
Read(FNTrainingEpoch, SizeOf(Integer));
Read(FTrainingError, SizeOf(Single));
```

```
    //--- Update number of training neuron
    UpdateNTrainingNeuron;
   finally
    Free;
   end;
  end;
  FModified := False;
end;

// Purpose    : Trains the backpropagation neural net with weight initialization
//              process
// Assumption : * The training set has been defined
//              * The training parameters have been set
procedure TBackProp.Retrain;
begin
  InitWeights;
  FNTrainingEpoch := 0;
  Train;
end;

// Purpose    : Saves the knowledge to external file
// Input      : FileName (must exists)
procedure TBackProp.SaveKnowledge(FileName: string);
var
  DataSize: Integer;
  FileStream: TFileStream;
  i, j, k, l, m, n: Integer;    // looping index for input layer, hidden layer,
                                //              output layer, training set,
                                //          training input patterns and other
begin
  FileStream := TFileStream.Create(FileName, fmCreate or fmShareExclusive);
```

```
with FileStream do
 try
  //--- Saves the network architecture
  Write(FInputPatternHeight, SizeOf(Integer));
  Write(FInputPatternWidth, SizeOf(Integer));
  Write(FTargetPatternHeight, SizeOf(Integer));
  Write(FTargetPatternWidth, SizeOf(Integer));
  Write(FNHiddenNeuron, SizeOf(Integer));

  //--- Saves the weights
  for j := 1 to FNHiddenNeuron do
   for i := 0 to FNInputNeuron do
    Write(FHiddenLayerWeights[j, i], SizeOf(TWeight));
  for k := 1 to FNOutputNeuron do
   for j := 0 to FNHiddenNeuron do
    Write(FOutputLayerWeights[k, j], SizeOf(TWeight));

  //--- Saves the training set
  DataSize := Length(FTrainingSet);
  Write(DataSize, SizeOf(Integer));
  for l := 0 to High(FTrainingSet) do
  begin
   DataSize := Length(FTrainingSet[l].InputPatterns);
   Write(DataSize, SizeOf(Integer));
   for m := 0 to High(FTrainingSet[l].InputPatterns) do
    for i := 1 to High(FTrainingSet[l].InputPatterns[m]) do
     Write(FTrainingSet[l].InputPatterns[m, i], SizeOf(TNeuron));
   for k := 1 to High(FTrainingSet[l].TargetPattern) do
    Write(FTrainingSet[l].TargetPattern[k], SizeOf(TNeuron));
   DataSize := Length(FTrainingSet[l].TargetPatternClass);
   Write(DataSize, SizeOf(Integer));
   for n := 1 to DataSize do
```

```
        Write(FTrainingSet[l].TargetPatternClass[n], SizeOf(Char));
      end;

      //--- Saves the training parameters
      Write(FLearningRate, SizeOf(Single));
      Write(FWeightsInitFactor, SizeOf(Single));
      Write(FErrorThreshold, SizeOf(Single));
      Write(FTargetClassificationError, SizeOf(Single));
      Write(FTargetSquaredError, SizeOf(Single));
      Write(FMaxEpoch, SizeOf(Integer));

      //--- Saves the training info
      Write(FNNeuronError, SizeOf(Integer));
      Write(FNTrainingEpoch, SizeOf(Integer));
      Write(FTrainingError, SizeOf(Single));
    finally
      Free;
    end;
  FModified := False;
end;

// Purpose    : Trains the backpropagation neural net without weight
//              initialization process
// Assumptions: * There is at least one training pairs in the training set
//              * The training parameters have been set
procedure TBackProp.Train;
label
  Finish;
const
  INFINITE_ERROR = 2;
type
  TErrors = array of single;
```

```
  TWeightCorrection = array of array of Single;
var
  //--- Looping index for input layer, hidden layer, output layer, training set
  //    and training input patterns
  i, j, k, l, m: Integer;

  //--- Weights correction with momentum
  HiddenErrors: TErrors;
  HiddenWeightsCorrection: TWeightCorrection;
  OutputErrors: TErrors;
  OutputWeightsCorrection: TWeightCorrection;

  //--- Training stop condition
  Error: Single;
  StartEpoch: Integer;
begin
  Assert(Length(FTrainingSet) > 0);

  //--- Initializing
  SetLength(OutputErrors, FNOutputNeuron + 1);
  SetLength(HiddenErrors, FNHiddenNeuron + 1);
  SetLength(OutputWeightsCorrection, FNOutputNeuron + 1, FNHiddenNeuron +
1);
  SetLength(HiddenWeightsCorrection, FNHiddenNeuron + 1, FNInputNeuron +
1);

  //--- Repeats the training until number of epoch is greater or equal than
  //    maximum number of epoch defined in property MaxEpcoh or the average
  //    error is less or equal than the target error minimum defined in property
  //    MinError or the user stop the training
  StartEpoch := FNTrainingEpoch;
  FStopTraining := False;
```

```
repeat
 FNTrainingEpoch := FNTrainingEpoch + 1;
 FTrainingError := 0;
 FNNeuronError := 0;

 //--- Repeats for all training pair in the training set
 for l := 0 to High(FTrainingSet) do
  for m := 0 to High(FTrainingSet[l].InputPatterns) do
  begin
   //--- Sets input layer activation value
   for i := 1 to High(FTrainingSet[l].InputPatterns[m]) do
    FInputLayer[i] := FTrainingSet[l].InputPatterns[m, i];

   //--- The feedforward phase
   Apply;

   //--- The backpropagation of error phase
   //- Computes output layer weights error information
   for k := 1 to FNOutputNeuron do
   begin
    Error := FTrainingSet[l].TargetPattern[k] - FOutputLayer[k];
    if Abs(Error) >= FErrorThreshold then
     FNNeuronError := FNNeuronError + 1;
    FTrainingError := FTrainingError + (Error * Error);
    OutputErrors[k] := Error * BipolarSigmoidDerivation(FOutputLayer[k]);
    for j := 0 to FNHiddenNeuron do
     OutputWeightsCorrection[k, j] :=
      FLearningRate * OutputErrors[k] * FHiddenLayer[j];
   end;
   //- Computes hidden layer weights error information
   for j := 1 to FNHiddenNeuron do
   begin
```

```
        HiddenErrors[j] := 0;
        for k := 1 to FNOutputNeuron do
          HiddenErrors[j] := HiddenErrors[j] +
                        (OutputErrors[k] * FOutputLayerWeights[k, j]);
        HiddenErrors[j] := HiddenErrors[j] *
                      BipolarSigmoidDerivation(FHiddenLayer[j]);
        for i := 0 to FNInputNeuron do
          HiddenWeightsCorrection[j, i] :=
            FLearningRate * HiddenErrors[j] * FInputLayer[i]
      end;
      //- Updates output layer weights and bias
      for k := 1 to FNOutputNeuron do
        for j := 0 to FNHiddenNeuron do
          FOutputLayerWeights[k, j] := FOutputLayerWeights[k, j] +
                              OutputWeightsCorrection[k, j];
      //- Updates hidden layer weights and bias
      for j := 1 to FNHiddenNeuron do
        for i := 0 to FNInputNeuron do
          FHiddenLayerWeights[j, i] := FHiddenLayerWeights[j, i] +
                              HiddenWeightsCorrection[j, i];
      Application.ProcessMessages;
    end;
  if FStopTraining then goto Finish;
  if Assigned(OnTraining) then OnTraining(Self);
 until (FNNeuronError <= FTargetClassificationError) or
      (FTrainingError / FNTrainingNeuron <= FTargetSquaredError) or
      (FNTrainingEpoch - StartEpoch >= FMaxEpoch);

Finish:
 FModified := True;
 if Assigned(OnTrainingFinish) then OnTrainingFinish(Self);
end;
```

```
//-----------------
// Class Methods
//-----------------

// Purpose    : Converts pattern in bitmap format to input layer
// Inputs     : * Bitmap
//              * Layer (layer's size = #pixel in Bitmap + 1)
class procedure TBackProp.BitmapToLayer(Bitmap: TBitmap; Layer: TLayer);
var
  i: Integer;                                  // index for input layer
  x, y: Integer;                               // x and y coordinate in the bitmap
begin
  Assert(Length(Layer) = (Bitmap.Width * Bitmap.Height) + 1);

  i := 1;
  for y := 0 to Bitmap.Height - 1 do
    for x := 0 to Bitmap.Width - 1 do
    begin
      if Bitmap.Canvas.Pixels[x, y] = clBlack then
        Layer[i] := 1
      else
        Layer[i] := -1;
      i := i + 1;
    end;
end;

//-----------------------------------
// Private Functions and Procedures
//-----------------------------------

// Purpose    : Returns the bipolar sigmoid function value of a x
```

```
function TBackProp.BipolarSigmoid(x: Single): Single;
begin
 try
  Result := (2 / (1 + Exp(-x))) - 1;
 except
  on EOverflow do Result := 0;
 end;
end;

// Purpose    : Returns the derivation of bipolar sigmoid function value using
//              its function value Fx
function TBackProp.BipolarSigmoidDerivation(Fx: Single): Single;
begin
 Result := ((1 + Fx) * (1 - Fx)) / 2;
end;

procedure TBackProp.InitLayers;
begin
 SetLength(FInputLayer, FNInputNeuron + 1);
 SetLength(FHiddenLayer, FNHiddenNeuron + 1);
 SetLength(FOutputLayer, FNOutputNeuron + 1);
 FInputLayer[0] := 1;                          // input layer's bias
 FHiddenLayer[0] := 1;                         // hidden layer's bias
 SetLength(FHiddenLayerWeights, FNHiddenNeuron + 1, FNInputNeuron + 1);
 SetLength(FOutputLayerWeights, FNOutputNeuron + 1, FNHiddenNeuron + 1);
end;

// Purpose    : Initilizes the backpropagation weights with random value
// Assumption : Parameter for weights random initialization has been set
procedure TBackProp.InitWeights;
var
 i, j, k: Integer;        // looping index for input, hidden and output layer
```

```
begin
 Randomize;
 //--- Initializes the hidden layer weights
 for j := 0 to FNHiddenNeuron do
  for i := 0 to FNInputNeuron do
   FHiddenLayerWeights[j, i] := (Random - 0.5) * (FWeightsInitFactor * 2);

 //--- Initializes the output layer weights
 for k := 1 to FNOutputNeuron do
  for j := 0 to FNHiddenNeuron do
   FOutputLayerWeights[k, j] := (Random - 0.5) * (FWeightsInitFactor * 2);
end;

// Purpose    : Updates the number of training neurons
procedure TBackProp.UpdateNTrainingNeuron;
var
 l: Integer;
begin
 FNTrainingNeuron := 0;
 for l := 0 to High(FTrainingSet) do
  FNTrainingNeuron :=
   FNTrainingNeuron +
   (Length(FTrainingSet[l].InputPatterns) * FNOutputNeuron);
end;

end.
```

```
{*****************************************************************

 Frame TDCRFrame
 ---------------

 File Name    : DCRFra.pas

 Language     : Delphi 2005

 Author       : Theo Zacharias (theo_yz@yahoo.com)

 Description  : TDCRFrame is a class frame that does the preprocess, text
                segmentation and reconstruction step of DCR

 Last modified on June 25, 2005

*****************************************************************}

unit DCRFra;

interface

uses
  Classes, Controls, ExtCtrls, Forms, Windows, Graphics;

const
  DEFAULT_BW_THRESHOLD = 196;
  DEFAULT_NOISE_THRESHOLD = 10;
  DEFAULT_SPACE_WIDTH = 22;

type
  //- Text to character segmentation -
```

```
 TImageArray = array of array of Byte;
 TChar = record
  Pixels: array of TPoint;
  MinPixel: TPoint;
  MaxPixel: TPoint;
  Noise: Boolean;
  CharCode: Char;
 end;
 TWord = record
  Left: Integer;
  Right: Integer;
  Chars: array of TChar;
 end;
 TLine = record
  Top: Integer;
  Bottom: Integer;
  Words: array of TWord;
 end;
 TLines = array of TLine;

 //- Drawing tool -
 TDrawingTool = (dtPencil, dtEraser);

type
 TDCRFrame = class(TFrame)
  Image: TImage;
  pnlTitle: TPanel;
  procedure ImageMouseMove(Sender: TObject; Shift: TShiftState; X,
   Y: Integer);
  procedure ImageMouseUp(Sender: TObject; Button: TMouseButton;
   Shift: TShiftState; X, Y: Integer);
  procedure ImageMouseDown(Sender: TObject; Button: TMouseButton;
```

```
    Shift: TShiftState; X, Y: Integer);

  private
    FDrawing: Boolean;
    FDrawingTool: TDrawingTool;
    FImageArray: TImageArray;
    FLines: TLines;
    FModified: Boolean;
    FResultText: string;

    FBWThreshold: Integer;
    FNoiseThreshold: Integer;
    FSpaceWidth: Integer;

    procedure FilterBnW;
    procedure SegmentLines;
    procedure SegmentWords;
    procedure SegmentChars;
    procedure GetRGBColor(Color: TColor;
                  out R: Integer; out G: Integer; out B: Integer);

  public
    property BWThreshold: Integer read FBWThreshold write FBWThreshold;
    property DrawingTool: TDrawingTool read FDrawingTool write
FDrawingTool;
    property NoiseThreshold: Integer read FNoiseThreshold write
FNoiseThreshold;
    property Modified: Boolean read FModified;
    property SpaceWidth: Integer read FSpaceWidth write FSpaceWidth;
    property ResultText: string read FResultText;

    procedure Clear;
```

```
    procedure Init;
    procedure OpenPicture(FileName: string);
    procedure Recognize;
    procedure SavePicture(FileName: string);
    procedure ShowHideTitle;
  end;

implementation

uses
  SysUtils,
  GlobalMdl, BackPropCls;

{$R *.dfm}

// Purpose    : Deletes all drawing on the text picture area
procedure TDCRFrame.Clear;
begin
  Image.Canvas.FillRect(Rect(0, 0, Image.Width, Image.Height));
  Image.Picture.Bitmap.Width := ClientWidth ;
  Image.Picture.Bitmap.Height := ClientHeight - pnlTitle.Height;
  FModified := False;
end;

// Purpose    : Conducts the preprocessing step, i.e. filter black & whites, of
//              the DCR
procedure TDCRFrame.FilterBnW;
var
  x, y: Integer;
  Grayscale: Real;
  R, G, B: Integer;
begin
```

```
 SetLength(FImageArray, Image.Width, Image.Height);
 for x := 0 to High(FImageArray) do
  for y := 0 to High(FImageArray[0]) do
  begin
    GetRGBColor(Image.Canvas.Pixels[x, y], R, G, B);
    Grayscale := (0.299 * R) + (0.587 * G) + (0.114 * B);
    if GrayScale >= FBWThreshold then
      FImageArray[x, y] := 0
    else
      FImageArray[x, y] := 1;
  end;
end;

// Purpose    : Gets the RGB color of a given color
procedure TDCRFrame.GetRGBColor(Color: TColor;
                        out R: Integer; out G: Integer; out B: Integer);
var
 RGBColor: Integer;
begin
 RGBColor := ColorToRGB(Color);
 R := RGBColor Mod 256;
 G := (RGBColor div 256) Mod 256;
 B := (RGBColor div 256) div 256;
end;

// Purpose    : Prepares to draw on the text picture area
// Event      : DCRFrame.Image.OnMouseDown
procedure TDCRFrame.ImageMouseDown(Sender: TObject; Button:
TMouseButton;
 Shift: TShiftState; X, Y: Integer);
begin
 if (Button = mbLeft) then
```

```
  begin
    FDrawing := True;
    Image.Canvas.MoveTo(X, Y);
    ImageMouseMove(Sender, Shift, X, Y);
  end;
end;

// Purpose    : Draws on the text picture are if in drawing mode
// Event      : DCRFrame.Image.OnMouseMove
procedure TDCRFrame.ImageMouseMove(Sender: TObject; Shift: TShiftState;
X,
  Y: Integer);
var
  dPos: Integer;
begin
  if FDrawing then
    with Image do
      if DrawingTool = dtPencil then
        Canvas.LineTo(X, Y)
      else if DrawingTool = dtEraser then
      begin
        dPos := (Canvas.Pen.Width div 2) + (Canvas.Pen.Width mod 2);
        Canvas.FillRect(Rect(X - dPos, Y - dPos, X + dPos, Y + dPos));
      end;
end;

// Purpose    : Finishes drawing on the text picture area
// Event      : DCRFrame.Image.OnMouseUp
procedure TDCRFrame.ImageMouseUp(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
```

```
  FDrawing := False;
end;

// Purpose    : Opens a picture from external file
// Input      : FileName (must exist)
procedure TDCRFrame.OpenPicture(FileName: string);
begin
  Assert(FileExists(FileName));

  Image.Picture.LoadFromFile(FileName);
  Image.Canvas.Pen.Width := DEFAULT_DRAWING_WIDTH;
  FModified := False;
end;

// Purpose    : Recognizes the picture on the text picture area
procedure TDCRFrame.Recognize;
var
  IxLine, IxWord, IxChar: Integer;
  i: Integer;
  OrgBitmap: TBitmap;
  StretchBitmap: TBitmap;
  ResultSymbol: TPatternClass;
begin
  Screen.Cursor := crHourglass;

  //--- Preprocess
  FilterBnW;

  //--- Text segmentation
  SegmentLines;
  SegmentWords;
  SegmentChars;
```

```
//--- Recognition and reconstruction
FResultText := '';
OrgBitmap := TBitmap.Create;
for IxLine := 0 to High(FLines) do
 for IxWord := 0 to High(FLines[IxLine].Words) do
 begin
  for IxChar := 0 to High(FLines[IxLine].Words[IxWord].Chars) do
   with FLines[IxLine].Words[IxWord].Chars[IxChar] do
    if not Noise then
    begin
     OrgBitmap.Width := MaxPixel.x - MinPixel.x + 1;
     OrgBitmap.Height := MaxPixel.y - MinPixel.y + 1;
     OrgBitmap.Canvas.FillRect(
      Rect(0, 0, OrgBitmap.Width, OrgBitmap.Height));
     for i := 0 to High(Pixels) do
      OrgBitmap.Canvas.Pixels[
       Pixels[i].x - MinPixel.x,
       Pixels[i].y - MinPixel.y] := clBlack;
     StretchBitmap := GetAutoFitBitmap(
                 OrgBitmap, BackProp.InputPatternWidth,
                 BackProp.InputPatternHeight);
     TBackProp.BitmapToLayer(StretchBitmap, BackProp.InputLayer);
     StretchBitmap.Free;
     BackProp.Apply;
     BackProp.GetResult(ResultSymbol);
     FResultText := FResultText + ResultSymbol;
     if (ResultSymbol = '.') and
       (IxWord = High(FLines[IxLine].Words)) and
       (IxChar = High(FLines[IxLine].Words[IxWord].Chars)) then
      FResultText := FResultText + chr(13) + chr(10);
    end;
```

```
    FResultText := FResultText + ' ';
   end;
  OrgBitmap.Free;

  Screen.Cursor := crDefault;
end;

// Purpose    : Saves the picture on the text picture area
procedure TDCRFrame.SavePicture(FileName: string);
begin
  Image.Picture.SaveToFile(FileName);
  FModified := False;
end;

// Purpose    : Conducts the character segmentation (words to characters) step
procedure TDCRFrame.SegmentChars;
const
  UNLABELED = MaxInt;
type
  TPixelLabel = Integer;
  TPixelLabels = array of TPixelLabel;
  TDirection = (drTop, drTopLeft, drLeft, drBottomLeft);
var
  Dir: TDirection;
  IdChars: TPixelLabels;
  IxLine, IxWord, IxChar, IxPixel: Integer;
  LabeledPixels: array of TPixelLabels;
  NewLabel: TPixelLabel;
  PixelLabels: array[TDirection] of TPixelLabel;
  ReplacementIdChars: TPixelLabels;
  ReplacementLabel: TPixelLabel;
  x, y: Integer;
```

```
begin
  for IxLine := 0 to High(FLines) do
    for IxWord := 0 to High(FLines[IxLine].Words) do
    begin
      with FLines[IxLine] do
      begin
        NewLabel := 0;
        SetLength(LabeledPixels, Words[IxWord].Right - Words[IxWord].Left + 2,
                                 Bottom - Top + 3);
        for x := 0 to High(LabeledPixels) do
        begin
          LabeledPixels[x, 0] := UNLABELED;
          LabeledPixels[x, High(LabeledPixels[0])] := UNLABELED;
        end;
        for y := 0 to High(LabeledPixels[0]) do
          LabeledPixels[0, y] := UNLABELED;
        for x := 1 to High(LabeledPixels) do
          for y := 1 to High(LabeledPixels[0]) - 1 do
          begin
            if FImageArray[x + Words[IxWord].Left - 1, y + Top - 1] <> 1 then
              LabeledPixels[x, y] := UNLABELED
            else
            begin
              PixelLabels[drTop] := LabeledPixels[x, y - 1];
              PixelLabels[drTopLeft] := LabeledPixels[x - 1, y - 1];
              PixelLabels[drLeft] := LabeledPixels[x - 1, y];
              PixelLabels[drBottomLeft] := LabeledPixels[x - 1, y + 1];
              if (PixelLabels[drTop] = UNLABELED) and
                (PixelLabels[drTopLeft] = UNLABELED) and
                (PixelLabels[drLeft] = UNLABELED) and
                (PixelLabels[drBottomLeft] = UNLABELED) then
              begin
```

```
        LabeledPixels[x, y] := NewLabel;
        SetLength(IdChars, NewLabel + 1);
        IdChars[NewLabel] := NewLabel;
        NewLabel := NewLabel + 1;
      end
      else
      begin
        ReplacementLabel := UNLABELED;
        for Dir := drTop to drBottomLeft do
          if PixelLabels[Dir] <> UNLABELED then
            if ReplacementLabel = UNLABELED then
            begin
              LabeledPixels[x, y] := IdChars[PixelLabels[Dir]];
              ReplacementLabel := IdChars[PixelLabels[Dir]];
            end
            else
              IdChars[PixelLabels[Dir]] := ReplacementLabel;
      end;
    end;
  end;
end;
for IxChar := 0 to High(IdChars) do
  while IdChars[IdChars[IxChar]] <> IdChars[IxChar] do
    IdChars[IxChar] := IdChars[IdChars[IxChar]];

SetLength(ReplacementIdChars, NewLabel);
for IxChar := 0 to High(ReplacementIdChars) do
  ReplacementIdChars[IxChar] := UNLABELED;
NewLabel := 0;
for IxChar := 0 to High(IdChars) do
begin
  if ReplacementIdChars[IdChars[IxChar]] = UNLABELED then
```

```
 begin
  ReplacementIdChars[IdChars[IxChar]] := NewLabel;
  NewLabel := NewLabel + 1;
 end;
 IdChars[IxChar] := ReplacementIdChars[IdChars[IxChar]];
end;
with FLines[IxLine].Words[IxWord] do
begin
 SetLength(Chars, NewLabel);
 for IxChar := 0 to High(Chars) do
 begin
  Chars[IxChar].MinPixel := Point(MaxInt, MaxInt);
  Chars[IxChar].MaxPixel := Point(0, 0);
  Chars[IxChar].Noise := False;
 end;
 for x := 1 to High(LabeledPixels) do
  for y := 1 to High(LabeledPixels[0]) - 1 do
   if LabeledPixels[x, y] <> UNLABELED then
    with Chars[IdChars[LabeledPixels[x, y]]] do
    begin
     SetLength(Pixels, Length(Pixels) + 1);
     Pixels[High(Pixels)] := Point(x, y);
     if x < MinPixel.x then MinPixel.x := x;
     if y < MinPixel.y then MinPixel.y := y;
     if x > MaxPixel.x then MaxPixel.x := x;
     if y > MaxPixel.Y then MaxPixel.y := y;
    end;
 for IxChar := 0 to High(Chars) - 1 do
  if ((Chars[IxChar].MinPixel.x >= Chars[IxChar + 1].MinPixel.x) and
     (Chars[IxChar].MaxPixel.x <= Chars[IxChar + 1].MaxPixel.x)) or
    ((Chars[IxChar + 1].MinPixel.x >= Chars[IxChar].MinPixel.x) and
     (Chars[IxChar + 1].MaxPixel.x <= Chars[IxChar].MaxPixel.x)) then
```

```
        begin
          SetLength(Chars[IxChar + 1].Pixels,
                Length(Chars[IxChar + 1].Pixels) +
                 Length(Chars[IxChar].Pixels));
          for IxPixel := 0 to High(Chars[IxChar].Pixels) do
            Chars[IxChar + 1].
              Pixels[Length(Chars[IxChar + 1].Pixels) -
                  Length(Chars[IxChar].Pixels) + IxPixel] :=
                Chars[IxChar].Pixels[IxPixel];
          if Chars[IxChar].MinPixel.x < Chars[IxChar + 1].MinPixel.x then
            Chars[IxChar + 1].MinPixel.x := Chars[IxChar].MinPixel.x;
          if Chars[IxChar].MinPixel.y < Chars[IxChar + 1].MinPixel.y then
            Chars[IxChar + 1].MinPixel.y := Chars[IxChar].MinPixel.y;
          if Chars[IxChar].MaxPixel.x > Chars[IxChar + 1].MaxPixel.x then
            Chars[IxChar + 1].MaxPixel.x := Chars[IxChar].MaxPixel.x;
          if Chars[IxChar].MaxPixel.y > Chars[IxChar + 1].MaxPixel.y then
            Chars[IxChar + 1].MaxPixel.y := Chars[IxChar].MaxPixel.y;
          Chars[IxChar].Noise := True;
        end;
      for IxChar := 0 to High(Chars) do
        if not Chars[IxChar].Noise then
          Chars[IxChar].Noise :=
            (Length(Chars[IxChar].Pixels) < FNoiseThreshold);
    end;
  end;
end;

// Purpose    : Conducts the line segmentation (text to lines) step
procedure TDCRFrame.SegmentLines;
var
  BlankRow: Boolean;
  SearchFor: (sfTop, sfBottom);
```

```
  x, y: Integer;
begin
  y := 0;
  SearchFor := sfTop;
  SetLength(FLines, 0);
  repeat
    BlankRow := True;
    for x := 0 to High(FImageArray) do
      if FImageArray[x, y] = 1 then
      begin
        BlankRow := False;
        if SearchFor = sfTop then
        begin
          SetLength(FLines, Length(FLines) + 1);
          FLines[High(FLines)].Top := y;
          SearchFor := sfBottom;
        end;
        Break;
      end;
    if (SearchFor = sfBottom) and BlankRow then
    begin
      FLines[High(FLines)].Bottom := y - 1;
      SearchFor := sfTop;
    end;
    y := y + 1;
  until y = Length(FImageArray[0]);
  if SearchFor = sfBottom then FLines[High(FLines)].Bottom := y - 1;
end;

// Purpose    : Conducts the word segmentation (lines to words) step
procedure TDCRFrame.SegmentWords;
var
```

```
  IxLine: Integer;
  NBlankColumn: Integer;
  SearchFor: (sfLeft, sfRight);
  x, y: Integer;
begin
  for IxLine := 0 to High(FLines) do
    with FLines[IxLine] do
    begin
      x := 0;
      SearchFor := sfLeft;
      NBlankColumn := 0;
      repeat
        NBlankColumn := NBlankColumn + 1;
        for y := Top to Bottom do
          if FImageArray[x, y] = 1 then
          begin
            NBlankColumn := 0;
            if SearchFor = sfLeft then
            begin
              SetLength(Words, Length(Words) + 1);
              Words[High(Words)].Left := x;
              SearchFor := sfRight;
            end;
            Break;
          end;
        if (SearchFor = sfRight) and (NBlankColumn >= FSpaceWidth) then
        begin
          Words[High(Words)].Right := x - NBlankColumn;
          SearchFor := sfLeft;
        end;
        x := x + 1;
      until x = Length(FImageArray);
```

```
    if SearchFor = sfRight then Words[High(Words)].Right := x - 1;
  end;
end;

// Purpose    : Toogles show/hide the title
procedure TDCRFrame.ShowHideTitle;
begin
 pnlTitle.Visible := not pnlTitle.Visible;
 if pnlTitle.Visible then
  Image.Top := pnlTitle.Height + 1
 else
  Image.Top := 0;
end;

// Purpose    : Initializes the frame
procedure TDCRFrame.Init;
begin
 FBWThreshold := DEFAULT_BW_THRESHOLD;
 FNoiseThreshold := DEFAULT_NOISE_THRESHOLD;
 FSpaceWidth := DEFAULT_SPACE_WIDTH;
 Image.Canvas.Pen.Width := DEFAULT_DRAWING_WIDTH;
 Clear;
end;

end.
```