BAB IV
IMPLEMENTASI

4.1 Lingkungan Implementasi

Implementasi perangkat lunak ini berupa sistem yang menerapkan metode *Al-Alaoui Backpropagation* untuk peramalan cuaca. Parameter yang digunakan dalam sistem ini antara lain yaitu suhu, tekanan udara, kelembaban udara, kecepatan angin. Pada lingkungan implementasi terdapat dua faktor yang mempengaruhi yaitu faktor perangkat keras dan faktor perangkat lunak.

4.1.1 Lingkungan Perangkat Keras

Perangkat keras yang digunakan dalam implementasi sistem ini antara lain yaitu :

1. Processor Intel® Core ™ 2 Duo CPU T5870 @ 2.00GHz
2. Memori 2 GB
3. Harddisk 256 GB
4. Monitor 14’
5. Keyboard

4.1.2 Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam implementasi sistem ini antara lain yaitu :

1. Sistem Operasi yang digunakan Windows 7
2. Aplikasi dibuat dengan Netbeans 6.1
3. JDK yang digunakan adalah jdk 1.6
4. Library Java untuk memproses data dari Excel menggunakan jxl.jar

4.2 Implementasi Program

4.2.1 Implemntasi Proses Prediksi

Terdapat beberapa proses dalam proses prediksi ini, antara lain yaitu : proses baca file, mencari nilai maksimum dan minimum data, normalisasi data, perhitungan *feedforward*, perhitungan *backpropagation*, perhitungan *weight update*, perhingan MSE, dan terakhir perhitungan tingkat akurasi data.

4.2.2 Proses Baca File

Proses baca file merupakan proses yang berfungsi untuk memanggil data dari excel yang akan digunakan dalam prediksi. Proses implementasi baca file ditunjukkan pada *sourcecode* 4.1.

```
public class loadDataLatih
{
     // inisialisasi awal parameter
    String fileName;
    Atribut[]dataSet=null;
    double [] a=null;
    double [] b=null;
    double [] c=null;
    double [] d=null;
    double [] e=null;
    String [] f=null;
    // constructor
    public loadDataLatih(String fileName)
    {
         this.fileName=fileName;
    }
    // Proses baca excel
    public Atribut[] baca()
    {
       File file = new File(fileName);
       String cell = "";
       try
       {
           Workbook workbook =
           jxl.Workbook.getWorkbook(file);
           Sheet[] sheets = workbook.getSheets();
           int row = sheets[0].getRows();
```

```
            int column = sheets[0].getColumns();
            dataset = new Atribut[row-1];
            a = new double[row-1];
            b = new double[row-1];
            c = new double[row-1];
            d = new double[row-1];
            e = new double [row-1];
            f = new String[row-1];

           for(int i=0;i<dataSet.length;i++)
           {
             dataSet[i]= new Atribut(0,"",0,0,0, 0, 0,"");
           }
           for(int i=1 ; i< row ; i++)
           {
              String tgl = sheets[0].getCell
                           (1,i).getContents();
              double x1 = Double.parseDouble(sheets[0].getCell
                          (2,i).getContents());
              double x2 = Double.parseDouble(sheets[0].getCell
                          (3,i).getContents());
              double x3 = Double.parseDouble(sheets[0].getCell
                          (4,i).getContents());
              double x4 = Double.parseDouble(sheets[0].getCell
                          (5,i).getContents());
              double x5 = Double.parseDouble(sheets[0].getCell
                          (6,i).getContents());
              String k = sheets[0].getCell(7,i).getContents();
              dataSet[i-1]= newAtribut(i,tgl,x1,x2,x3,x4,x5,k);
              a[i-1] = dataSet[i-1].suhu;
              b[i-1] = dataSet[i-1].tekanan;
              c[i-1] = dataSet[i-1].kelembaban;
              d[i-1] = dataSet[i-1].kecepatan;
              e[i-1] = dataSet[i-1].target;
              f[i-1] = dataSet[i-1].kategori;
            }
          }
        catch (Exception ex)
        {
             System.out.println("Gagal membaca data"
                                +ex.getMessage());
        }
         return dataSet;
    }
}
```

Sourcecode 4.1 Proses Baca File

4.2.3 Proses Mencari Nilai Maksimum dan Minimum Data

Proses mencari nilai maksimum dan minimum merupakan proses yang berfungsi mencari nilai maksimum dan minimum suatu data yang akan digunakan dalam proses normalisasi data. Proses mencari nilai maksimum dan minimum ditunjukkan pada *sourcecode* 4.2.

```
public void temp(double [] suhu, double [] tekanan, double []
kelembaban, double [] kecepatan)
    {
        this.suhu=suhu;
        this.tekanan=tekanan;
        this.kelembaban=kelembaban;
        this.kecepatan=kecepatan;
    }
    // Proses menghitung nilai minimum maksimum
    public void MinMax()
    {
        for (int i=1;i<suhu.length;i++)
        {
            if (maxS<suhu[i])
            {
                if(minS >suhu[i])
                {
                    minS=suhu[i];
                }
                maxS=suhu[i];
            }
            else if (minS>suhu[i])
            {
                minS=suhu[i];
            }
        }
        for (int i=1;i<tekanan.length;i++)
        {
            if (maxT < tekanan[i])
            {
                if(minT >tekanan[i])
                {
                    minT =tekanan[i];
                }
                maxT =tekanan[i];
            }
            else if (minT > tekanan[i])
```

```
            {
                minT =tekanan[i];
            }
        }
        for (int i=1;i<kelembaban.length;i++)
        {
            if (maxK <kelembaban[i])
            {
                if(minK >kelembaban[i])
                {
                    minK =kelembaban[i];
                }
                maxK =kelembaban[i];
            }
            else if (minK>kelembaban[i])
            {
                minK=kelembaban[i];
            }
        }
        for (int i=1;i<kecepatan.length;i++)
        {
            if (maxKc<kecepatan[i])
            {
                if(minKc >kecepatan[i])
                {
                    minKc=kecepatan[i];
                }
                maxKc=kecepatan[i];
            }
            else if (minKc>kecepatan[i])
            {
                minKc=kecepatan[i];
            }
        }
    }
```

Sourcecode 4.2 Proses Mencari Nilai Maksimum dan Minimum Data

4.2.4 Proses Normalisasi Data

Proses normalisasi data merupakan proses yang berfungsi untuk mengubah data menjadi nilai antara 0-1. Proses normalisasi data ditunjukkan pada *Sourcecode* 4.3.

```
public void Normalisasi()
    {
        normSuhu = new double[suhu.length];
        normTekanan = new double[tekanan.length];
        normKelembaban = new double[kelembaban.length];
        normKecepatan = new double[kecepatan.length];

        for (int i=0;i<suhu.length;i++)
        {
            normSuhu[i]=((0.8*(suhu[i]-minS)/(maxS-
                        minS))+0.1);
        }
        for (int i=0;i<tekanan.length;i++)
        {
            normTekanan[i]=((0.8*(tekanan[i]-minT)/(maxT-
                           minT))+0.1);
        }
        for (int i=0;i<kelembaban.length;i++)
        {
            normKelembaban[i]=((0.8*(kelembaban[i]-
                               minK)/(maxK-minK))+0.1);
        }
        for (int i=0;i<kecepatan.length;i++)
        {
            normKecepatan[i]=((0.8*(kecepatan[i]-
                              minKc)/(maxKc-minKc))+0.1);
         }
    }
}
```

Sourcecode 4.3 Proses Normalisasi Data

4.2.5 Proses Pelatihan

4.2.5.1 Proses *Feedforward*

Proses *feedforward* merupakan proses yang digunakan untuk mengaktifkan neuron-neuron pada *hidden layer*. Dalam proses *feedforward* terdapat beberapa proses perhitungan antara lain yaitu : proses perhitungan nilai znet, *hidden layer*, ynet, dan *output layer*. Proses *feedforward* ditunjukkan pada *sourcecode* 4.4.

```
public void feedforward(int a)
    {
          Znet = new double [neuroHidden];
          Z= new double [neuroHidden];

          //Menghitung nilai Hidden Layer
          for (int j=0;j<neuroHidden;j++)
          {
               // Hitung Nilai Znetj
              Znet[j]= bias+((suhutemp.get(a)*VijUpdate[0][j])+
                       (tekanantemp.get(a)*VijUpdate[1][j])+
                       (kelembabantemp.get(a)*VijUpdate[2][j])+
                       (kecepatantemp.get(a)*VijUpdate[3][j]));

              // Hitung nilai Zj (Hidden Layer)
              Z[j] = 1/(1+Math.exp(-Znet[j]));
          }
          // Menghitung Nilai Ynet
          Ynet.set(a,(bias+((Z[0]*WjkUpdate[0])+(Z[1]*
                   WjkUpdate[1])+(Z[2]*WjkUpdate[2])+(Z[3]*
                   WjkUpdate[3]))));
           // Menghitung Nilai Output Layer
           Y.set(a,(1/(1+Math.exp(-Ynet.get(a)))));

    }
```

Sourcecode 4.4 Proses Feedforward

4.2.5.2 Proses *Backpropagation*

Dalam proses *backpropagation* terdapat beberapa proses perhitungan antara lain yaitu : proses perhitungan nilai galat *output layer*, koreksi bobot antara *hidden layer* dengan *output layer*, galat *hidden layer*, koreksi bobot antara *hidden layer* dengan *input layer*. Proses *backpropagation* ditunjukkan pada *sourcecode* 4.5.

```
    public void backpropagation(double alpha,int a)
    {
       this.alpha=alpha;
       DeltaWjk = new double [neuroHidden];
       Deltanet = new double [neuroHidden];
       GalatHidden = new double [neuroHidden];
       DeltaVij = new double [neuroInput][neuroHidden];
```

```
        // Menghitung nilai galat output layer
        GalatOut.set (a,((targettemp.get(a)-Y.get(a))*Y.get(a)*
                    (1-Y.get(a))));

        for (int j =0;j<neuroHidden;j++)
        {
            // Menghitung koreksi bobot antara hidden layer
               dengan output layer
            DeltaWjk[j]= alpha*GalatOut.get(a)*Z[j];
            Deltanet[j]= GalatOut.get(a)*WjkUpdate[j];

            // Menghitung galat hidden layer
            GalatHidden[j]= Deltanet[j]*Z[j]*(1-Z[j]);
         }

        // Menghitung koreksi bobot antara input layer dengan
           hidden layer
        for (int j =0;j<neuroHidden;j++)
        {
            DeltaVij[0][j]=alpha*GalatHidden[j]*suhutemp.get(a);

            DeltaVij[1][j]=alpha*GalatHidden[j]*tekanantemp.get(a);
            DeltaVij[2][j]=alpha*GalatHidden[j]*
                           kelembabantemp.get(a);
            DeltaVij[3][j]=alpha*GalatHidden[j]*
                           kecepatantemp.get(a);
        }
    }
```

Sourcecode 4.5 Proses *Backpropagation*

4.2.5.3 Proses *Weight Update*

Proses *weight update* merupakan proses yang digunakan untuk me. Dalam proses *weight update* terdapat beberapa proses perhitungan antara lain yaitu : proses perhitungan nilai bobot baru antara *hidden layer* dengan *output layer* dan bobot baru antara *hidden layer* dengan *input layer*. Proses *weight update* ditunjukkan pada *sourcecode* 4.6.

```
    // Proses weight update
     public void weightupdate(double momen,int a)
     {
         DeltaWjkMomen = new double [neuroHidden];
         WjkNew = new double [neuroHidden];
         DeltaVijMomen = new double[neuroInput][neuroHidden];
         VijNew = new double [neuroInput][neuroHidden];
```

```
        for (int j=0;j<neuroHidden;j++)
        {
            for (int k =0;k<neuroOutput;k++)
            {
                // Menghitung nilai bobot baru antara hidden layer
                  dengan output layer
                DeltaWjkMomen[j] = (DeltaWjk[j])+
                                   (momen*DeltaWjk[j]);
                WjkNew[j] = WjkUpdate[j]+DeltaWjkMomen[j];
            }
        }
        for (int i =0;i<neuroInput;i++)
        {
            for (int j = 0; j<neuroHidden;j++)
            {
                // Menghitung nilai bobot baru antara input layer
                  dengan hidden layer
                DeltaVijMomen[i][j] = (DeltaVij[i][j])+
                                      (momen*DeltaVij[i][j]);
                VijNew[i][j]= VijUpdate[i][j]+DeltaVijMomen[i][j];
            }
        }
        VijUpdate = VijNew;
        WjkUpdate = WjkNew;
    }
```

Sourcecode 4.6 Proses *Weight Update*

4.2.6 Proses Menghitung *Error*

Proses menghitung *error* merupakan proses yang digunakan untuk menghitung nilai kesalahan antara *output* data dengan *output layer*. Proses menghitung *error* ditunjukkan pada *sourcecode* 4.7.

```
// Menghitung nilai error ( MSE )
    public void Error()
    {
       for (int a =0;a<suhutemp.size();a++)
       {
            MSE = MSE+(Math.pow(targettemp.get(a)-Y.get(a),2));
       }
       MSE = MSE/suhutemp.size();
    }
```

Sourcecode 4.7 Proses Menghitung Error

4.2.7 Proses Duplikasi Data

Proses duplikasi data merupakan proses yang digunakan untuk menduplikasikan data yang salah pada saat melakukan klasifikasi. Proses ini ditunjukkan pada *sourcecode* 4.8.

```
// Proses Duplikasi data
    public void duplikasi()
    {
        // Klasifikasi berdasarkan kategori data
        for(int a=0;a<suhutemp.size();a++)
        {
            if(targettemp.get(a)<=0.3)
            {
                kelas.set(a,1);
            }
            else if(targettemp.get(a)>0.3&&targettemp.get(a)<=0.6)
            {
                kelas.set(a, 2);
            }
            else if (targettemp.get(a)>0.6)
            kelas.set(a, 3);
        }
        // Klasifikasi nilai Y ke dalam kelas-kelas yang ada
        for (int a=0;a<suhutemp.size();a++)
        {
            if (Y.get(a)<=0.3)
            {
              kategoridata.set(a,1);
            }
            else if (Y.get(a)>0.3&&Y.get(a)<=0.6)
            {
              kategoridata.set(a,2);
            }
            else if (Y.get(a)>0.6)
            {
              kategoridata.set(a,3);
            }

            if(kategoridata.get(a)==3)
            {
               Klasifikasi.set(a,"Hujan");
            }
            else if(kategoridata.get(a)==2)
```

```
            {
                Klasifikasi.set(a,"Berawan");
            }
            else if(kategoridata.get(a)==1)
            {
                Klasifikasi.set(a,"Cerah");
            }
        }
        // Proses duplikasi data yang salah
        for (int a=0;a<suhutemp.size();a++)
        {
            if (kelas.get(a)!=kategoridata.get(a))
            {
                data.add(a);
            }
        }
        // Menambahkan data yang salah
        for (int a =0;a<data.size();a++)
        {
            suhutemp.add(suhutemp.get(data.get(a)));
            tekanantemp.add(tekanantemp.get(data.get(a)));
            kelembabantemp.add(kelembabantemp.get
                                (data.get(a)));
            kecepatantemp.add(kecepatantemp.get
                              (data.get(a)));
            targettemp.add(targettemp.get(data.get(a)));
            kategoritemp.add(kategoritemp.get(data.get(a)));
            kelas.add(kelas.get(data.get(a)));
            kategoridata.add(kategoridata.get(data.get(a)));
            GalatOut.add(GalatOut.get(data.get(a)));
            Ynet.add(Ynet.get(data.get(a)));
            Y.add(Y.get(data.get(a)));
            Klasifikasi.add(Klasifikasi.get(data.get(a)));

        }
    }
}
```

Sourcecode 4.8 Proses Menghitung Error

4.2.8 Proses Pengujian

4.2.8.1 Proses *Feedforward*

Proses *feedforward* merupakan proses yang digunakan untuk mengaktifkan neuron-neuron pada *hidden layer*. Dalam proses *feedforward* terdapat beberapa proses perhitungan antara lain yaitu : proses perhitungan nilai znet, *hidden layer*, ynet, dan *output layer*. Proses *feedforward* ditunjukkan pada *sourcecode* 4.9.

```
public void feedforward (double [] suhu, double [] tekanan,
double [] kelembaban, double [] kecepatan,double [][] VijUpdate,
double []WjkUpdate)
    {
        this.suhu = suhu;
        this.tekanan = tekanan;
        this.kelembaban = kelembaban;
        this.kecepatan = kecepatan;
        this.VijUpdate = VijUpdate;
        this.WjkUpdate = WjkUpdate;
        Znet = new double [neuroHidden];
        Z= new double [neuroHidden];
        Ynet= new double [suhu.length];
        Y= new double [suhu.length];
        for (int a=0;a<suhu.length;a++)
        {
            //Menghitung nilai Hidden Layer
            for (int j=0;j<neuroHidden;j++)
            {
               // Hitung Nilai Znetj
               Znet[j] = bias+((suhu[a]*VijUpdate[0][j])+
                         (tekanan[a]*VijUpdate[1][j])+
                         (kelembaban[a]*VijUpdate[2][j])+
                         (kecepatan[a]*VijUpdate[3][j]));
               // Hitung nilai Zj (Hidden Layer)
               Z[j] = 1/(1+Math.exp(-Znet[j]));
            }
            // Menghitung Nilai Ynet
            Ynet[a] = bias+((Z[0]*WjkUpdate[0])+
                      (Z[1]*WjkUpdate[1])+(Z[2]*WjkUpdate[2])+
                      (Z[3]*WjkUpdate[3]));
            // Menghitung Nilai Output Layer
            Y[a]=1/(1+Math.exp(-Ynet[a]));
        }
    }
```

Sourcecode 4.9 Proses Feedforward

4.2.9 Proses Menghitung Tingkat Akurasi

Proses menghitung tingat akurasi merupakan proses yang digunakan untuk menghitung tingkat akurasi dari prediksi yang dilakukan. Proses ini ditunjukkan pada *sourcecode* 4.10.

```
public void akurasi (double target [])
    {
        this.target=target;
        kelas = new int [suhu.length];
        kategoridata = new int [suhu.length];
        Klasifikasi = new String [suhu.length];

         // Klasifikasi berdasarkan kategori data
        for(int a=0;a<suhu.length;a++)
        {
            if(target[a]<=0.3)
            {
                kelas[a]=1;
            }
            else if (target[a]>0.3&&target[a]<=0.6)
            {
                kelas[a]=2;
            }
            else if (target[a]>0.6)
            kelas[a]=3;
        }

        // Klasifikasi nilai Y ke dalam kelas-kelas yang ada
        for (int a=0;a<suhu.length;a++)
        {

            if (Y[a]>0.6)
            {
              kategoridata[a] = 3;
            }
            else if (Y[a]<=0.3)
            {
              kategoridata[a] = 1;
            }
            else if (Y[a]>0.3&&Y[a]<=0.6)
            {
```

```
                kategoridata[a] = 2;
            }
            if(kategoridata[a]==3)
            {
                Klasifikasi[a] ="Hujan";
            }
            else if(kategoridata[a]==2)
            {
                Klasifikasi[a]= "Berawan";
            }
            else if(kategoridata[a]==1)
            {
                Klasifikasi[a] = "Cerah";
            }
        }
        // Proses duplikasi data yang salah
        for (int a=0;a<suhu.length;a++)
        {
            if (kelas[a]==kategoridata[a])
            {
                benar++;
            }
            else if (kelas[a]!=kategoridata[a])
            {
                salah++;
            }
        }
        akurasi = (benar*100)/suhu.length;
    }
}
```

Sourcecode 4.10 Proses Menghitung Tingkat Akurasi

4.3 Implementasi Antarmuka

Implementasi antarmuka terdiri dari dua bagian utama, yaitu :

1. *Form* Pelatihan

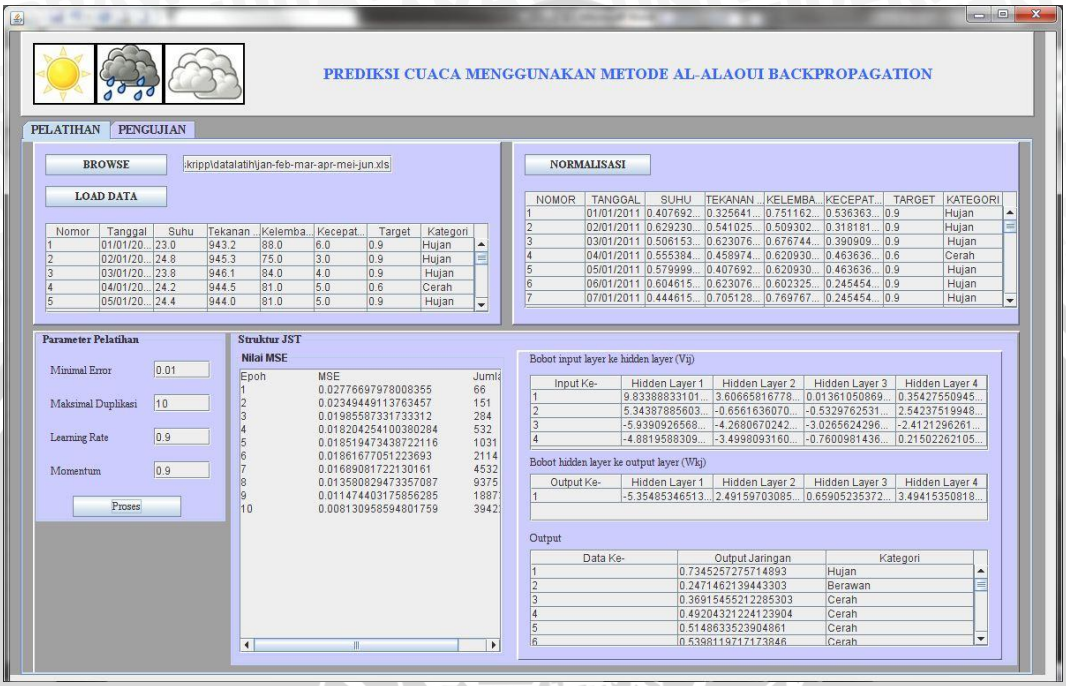
   Form pelatihan digunakan sebagai antarmuka untuk mendapatkan struktur jaringan syaraf tiruan yang terbaik.

2. *Form* Pengujian

   Form pengujian digunakan sebagai antarmuka untuk melakukan pengujian kondisi cuaca terhadap data sebenarnya dengan data hasil prediksi.

4.3.1 *Form* Pelatihan

Form pelatihan digunakan untuk mendapatkan struktur jaringan syaraf tiruan yang terbaik. Gambar *form* pelatihan ditunjukkan pada gambar 4.1.



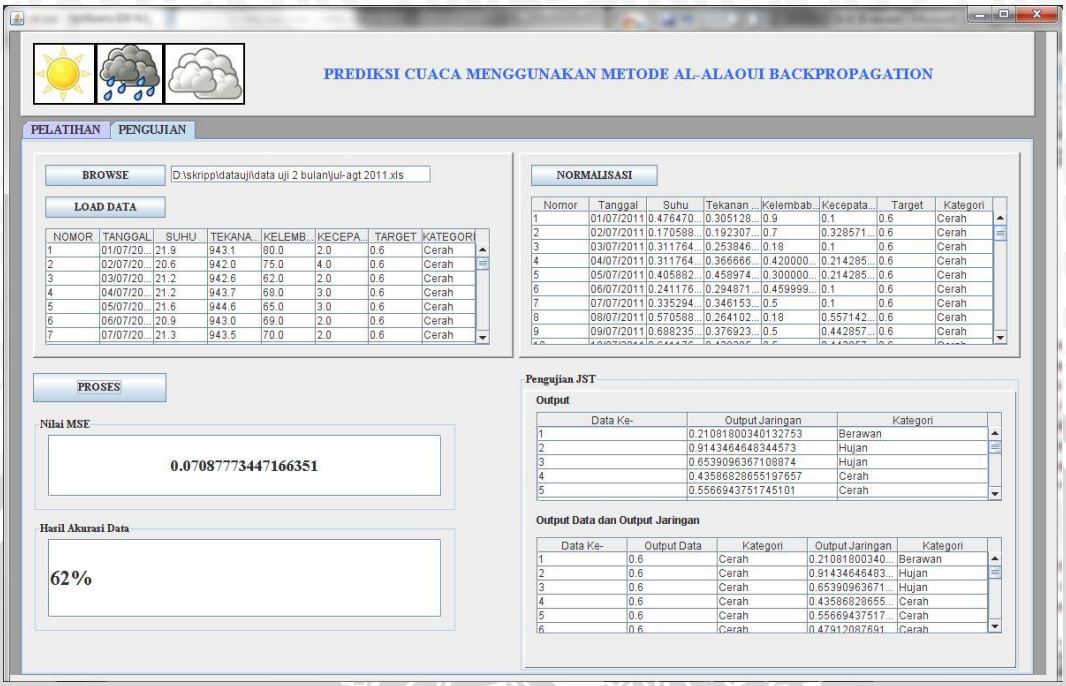
Gambar 4.1 *Form* Pelatihan

Pada *form* pelatihan terdapat 4 sub menu yaitu sub menu *browse* berfungsi untuk memilih data yang akan digunakan dalam pelatihan, sub menu *load* data berfungsi untuk menampilkan data yang akan digunakan dalam pelatihan, sub menu normalisasi berfungsi untuk mengubah data yang digunakan ke dalam bentuk *range* 0 sampai 1, sub menu proses yang berfungsi untuk melakukan proses jaringan syaraf tiruan. Dari pelatihan ini akan ditampilkan data yang digunakan, data yang telah dinormalisasi, bobot akhir dari pelatihan, nilai *output layer*, dan hasil MSE setiap *epoch*.

4.3.2 *Form* Pengujian

Form pengujian merupakan antarmuka untuk melakukan prediksi serta melakukan pengujian antara *output* hasil pengujian dengan *output* data dan

menghitung tingkat akurasi dari prediksi tersebut. Gambar dari *form* pengujian ditunjukkan pada gambar 4.2.



Gambar 4.2 *Form* Pengujian

Pada *form* pengujian terdapat 4 sub menu yaitu sub menu *browse* berfungsi untuk memilih data yang akan digunakan dalam pelatihan, sub menu *load* data berfungsi untuk menampilkan data yang akan digunakan dalam pelatihan, sub menu normalisasi berfungsi untuk mengubah data yang digunakan ke dalam bentuk *range* 0 sampai 1, sub menu proses yang berfungsi untuk melakukan proses prediksi. Dari pengujian ini akan ditampilkan data yang digunakan, data yang telah dinormalisasi, nilai *output layer*, perbandingan nilai *output* data dengan *output layer*, hasil MSE, dan tingkat akurasi prediksi.