BAB 5 IMPLEMENTASI

Bab ini menjelaskan mengenai implementasi dari hasil perancangan yang telah dibuat sebelumnya. Pembahasan terdiri dari penjelasan tentang spesifikasi sistem, implementasi algoritma, dan implementasi antarmuka.

5.1 Spesifikasi Sistem

Spesifikasi sistem digunakan untuk menjelaskan secara detail mengenai kesesuaian implementasi. Spesifikasi sistem menjelaskan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam sistem.

5.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan dalam implementasi metode *backpropagation* dengan insialisasi bobot *nguyen-widrow* untuk klasifikasi rumah layak huni dijelaskan pada Tabel 5.1:

Tabel 5. 1 Spesifikasi Perangkat Keras

Nama	Spesifikasi
Prosesor	Intel® Pentium® P6100 2,00 GHz
Memori (RAM)	1,00 GB
Hardisk	320 GB

5.1.2 Spesifikasi Perangkat Lunak

Spesifikasi perangkat lunak yang digunakan dalam implementasi metode *backpropagation* dengan insialisasi bobot *nguyen-widrow* untuk klasifikasi rumah layak huni dijelaskan pada Tabel 5.2:

Tabel 5. 2 Spesifikasi Perangkat Lunak

Nama	Spesifikasi
Sistem Operasi	Sistem Operasi Windows 7 32-bit
Bahasa Pemrograman	Java
Tools data	Microsoft Office 2013

5.2 Batasan Implementasi

Beberapa batasan dalam klasifikasi rumah layak huni menggunakan metode *Backpropagation* antara lain sebagai berikut:

1. Data masukan yang diterima pada sistem berupa:
    - Data rumah layak huni dan rumah tidak layak huni di Desa Kidal Kecamatan Tumpang Kabupaten Malang.
    - Nilai *learning rate,* iterasi maksimum, error toleransi.

2. Keluaran yang dihasilkan oleh sistem berupa hasil klasifikasi rumah layak huni berdasarkan hasil inisialisasi bobot *nguyen-widrow* pada learning *backpropagation*.
3. Kriteria yang digunakan dalam penelitian ini sebanyak 15 kriteria yaitu jumlah penghasilan per bulan, pekerjaan, jumlah anggota keluarga, jumlah KK dalam rumah, status kepemilikan rumah, kondisi rumah, luas bangunan, luas tanah, jenis lantai rumah, bahan dinding rumah, kondisi dinding rumah, bahan atap rumah, kamar MCK, air bersih, dan listrik.
4. Pengelolaan data dan perhitungan dapat dilakukan oleh pengguna dengan mengubah nilai *hidden layer* dan jumlah perbandingan data

5.3 Implementasi Algoritma

Implementasi algoritma menjelaskan tentang algoritma dari klasifikasi rumah layak huni yang terdiri dari implementasi *Nguyen-widrow* dan implementasi algoritma *Backpropagation*.

5.3.1 Implementasi Normalisasi Data

Proses normalisasi data digunakan untuk menghasilkan rentng nilai antara 0,1 – 0,9. Data *input* dan target dilakukan penskalaan dengan menggunakan metode *min-max*. Implementasi proses normalisasi data ditunjukkan pada Tabel 5.3.

Tabel 5. 3 Kode Program Implementasi Normalisasi Data

```
1  public static void Normalisasi() {
2     for (int i = 0; i < data.length; i++) {
3        for (int j = 0; j < JumlahNormalisasi[0].length; j++){
4              normalisasi_upah[i] = (0.8 * (konversi_upah[i] –
5              min) / (maxupah - min)) + 0.1;
6              normalisasi_pekerjaan[i] = (0.8 *
7              (konversi_pekerjaan[i] - min) / (maxpekerjaan –
8              min)) + 0.1;
9              normalisasi_jml[i] = (0.8 * (konversi_jml[i] –
10             min) / (maxjml - min)) + 0.1;
11             normalisasi_kk[i] = (0.8 * (konversi_kk[i] - min)
12             / (maxkk - min)) + 0.1;
13             normalisasi_status[i] = (0.8 *(konversi_status[i]
14             - min) / (maxstatus - min)) + 0.1;
15             normalisasi_kondisi[i] = (0.8 *
16             (konversi_kondisi[i] - min) / (maxkondisi - min))
17             + 0.1;
18             normalisasi_lb[i] = (0.8 * (konversi_lb[i] - min)
19             / (maxlb - min)) + 0.1;
```

```
20              normalisasi_lt[i] = (0.8 * (konversi_lt[i] - min)
21              / (maxlt - min)) + 0.1;
22              normalisasi_jenis[i] = (0.8 * (konversi_jenis[i]
23              - min) / (maxjenis - min)) + 0.1;
24              normalisasi_dinding[i] = (0.8 *
25              (konversi_dinding[i] - min) / (maxdinding - min))
26              + 0.1;
27              normalisasi_kondisiDinding[i] = (0.8
28              (konversi_kondisiDinding[i] - min) /
29              (maxkondisiDinding - min)) + 0.1;
30              normalisasi_atap[i] = (0.8 * (konversi_atap[i] -
31              min) / (maxatap - min)) + 0.1;
32              normalisasi_mck[i] = (0.8 * (konversi_mck[i] -
33              min) / (maxmck - min)) + 0.1;
34              normalisasi_air[i] = (0.8 * (konversi_air[i] -
35              min) / (maxair - min)) + 0.1;
36              normalisasi_listrik[i] = (0.8 *
37              (konversi_listrik[i] - min) / (maxlistrik - min))
38              + 0.1;
39          }
40      }
```

Pada Tabel 5.3 baris 1-40 merupakan fungsi mencari perhitungan normalisasi. Perhitungan tersebut menggunakan nilai maksimum dan nilai minimum serta batas maksimum(ω) dan batas minimum(α).

5.3.2 Implementasi *Nguyen-widrow*

Proses inisialisasi nonot menggunakan metode *Nguyen-widrow.* Pertama, dilakukan inisialisasi bobot secara acak dengan range (-0,5) – 0,5. Kemudian akan dilakukan perhitungan *Nguyen-widrow* dengan menggunakan bobot awal tersebut untuk mendapatkan bobot baru. Berikut implementasi metode *Nguyen-widrow* untuk inisialisasi bobot yang ditunjukkan pada Tabel 5.4.

Tabel 5. 4 Kode Program Implementasi *Nguyen-widrow*

```
1   public static void Nguyen(int iterasi) {
2      double min = -0.5, max = 0.5;
3      Sistem.out.println("Nguyen");
4      for (int a = 0; a < hiddenLayer; a++) {
5         for (int b = 0; b < JumlahNormalisasi[0].length; b++){
6             Vij[a][b] = rand.nextDouble()+ min;
7             pangkat += (Math.pow(Vij[a][b], 2));
```

```
8             Sistem.out.print(Vij[a][b] + "   ");
9         }
10            VijMutlak[a] = Math.sqrt(pangkat);
11            Sistem.out.print("VijMutlak=" + VijMutlak[a]);
12            Sistem.out.println();
13    }
14    for (int m = 0; m < iterasi; m++) {
15       Sistem.out.println("-------------------------------");
16       Sistem.out.println("Iterasi Ke - " + (m + 1));
17       Sistem.out.println("Vij Baru");
18            for (int a = 0; a < hiddenLayer; a++) {
19               for (int b = 0; b < 16; b++) {
20                  if (b == 0) {
21                    VijBaru[a][0] = rand.nextDouble() +
22                    min;
23                  } else {
24                    VijBaru[a][b] = (beta * Vij[a][b]) /
25                    VijMutlak[a];
26                  }
27                  Sistem.out.print("  " + VijBaru[a][b]);
28               }
29            }
30            Sistem.out.println("Wkj");
31            for (int a = 0; a < 4; a++) {
32               W[a] = (rand.nextDouble()+beta) - (0.5*beta);
33               Sistem.out.println("  " + W[a]);
34            }
35    }
```

Pada Tabel 5.4 baris 4-8 merupakan fungsi inisialisasi bobot V dimana pada baris 6 merupakan fungsi inisialisasi bobot secara acak dengan range [-0,5 – 0,5]. Untuk baris 10-12 merupakan fungsi untuk mengitung Vmutlak. Baris 14-29 merupakan fungsi untuk menghitung bobot baru dari V. Pada baris 30-35merupakan fungsi inisialisasi bobot W dimana pada baris 32 merupakan fungsi inisialisasi bobot secara acak dengan range [-β – β].

5.3.3 Implementasi *Feed Forward*

Proses *feed forward* memiliki proses perhitungan nilai masukan dari inisialisasi bobot hingga perhitungan nilai keluaran pada *output layer*.

5.3.3.1 Implementasi Masukan pada *Hidden Layer*

Perhitungan masukan ke *hidden layer* dari *input layer* merupakan hasil perkalian dari data yang telah dinormalisasi dengan metode *min-max* dengan inisialisasi bobot yang yang didapat dari *Nguyen-widrow.* Implementasi proses perhitungan ditunjukkan pada Tabel 5.5.

Tabel 5. 5 Kode Program Implementasi Masukan pada *Hidden Layer*

```
1  public static void Feedforward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("------------------------------");
4        Sistem.out.println("Iterasi Ke - " + (m + 1));
5        Sistem.out.println("Feedforward");
6        Sistem.out.println("Z_net");
7        for (int a = 0; a < dataLatih.length; a++) {
8           for (int b = 0; b < hiddenLayer; b++) {
9              for (int i = 0; i < JumlahNormalisasi[0].length;
10                  i++) {
11                total[a][b] += (JumlahNormalisasi[a][i] *
12                VijBaru[b][i + 1]);
13                z_net[a][b] = VijBaru[b][0] + total[a][b];
14             }
15             Sistem.out.print("  " + z_net[a][b]);
16          }
17       }
```

Pada Tabel 5.5 merupakan bagian fungsi dari *feed forward.* Pada baris 1-17 merupakan fungsi menghitung Z_net (masukan ke *hidden* layer) dimana dilakukan perulangan sebanyak jumlah data, *hidden,* dan *input* untuk menjumlahkan hasil perkalian data yang telah dinormalisasi dengan bobot yang didapat dari inisialisasi bobot *Nguyen-widrow*.

5.3.3.2 Implementasi Keluaran pada *Hidden Layer*

Perhitungan keluaran dari *hidden layer* menuju *output layer* merupakan hasil perhitungan fungsi aktivasi *sigmoid* biner. Implementasi proses perhitungan ditunjukkan pada Tabel 5.6.

Tabel 5. 6 Kode Program Implementasi Keluaran pada *Hidden Layer*

```
1  public static void Feedforward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("Zi");
4        for (int a = 0; a < dataLatih.length; a++) {
5           for (int b = 0; b < hiddenLayer; b++) {
```

```
6                 Zi[a][b] = 1 / (1 + (Math.pow(eksponen,
7                 z_net[a][b])));
8                 Sistem.out.print("  " + Zi[a][b]);
9             }
10    }
```

Pada Tabel 5.6 merupakan fungsi menghitung Zi (keluaran pada *hidden layer*) yang ditunjukkan pada baris 1-10. Pada fungsi ini dilakukan perulangan sebanyak jumlah data dan *hidden* untuk dilakukan perhitungan dengan menggunakan nilai Z_net yang telah didapat.

5.3.3.3 Implementasi Masukan pada *Output Layer*

Perhitungan masukan ke *output layer* dari *hidden layer* merupakan hasil perkalian antara keluaran pada *hidden layer* dengan bobot yang diinisialisasi menggunakan metode *Nguyen-widrow.* Implementasi proses perhitungan ditunjukkan pada Tabel 5.7.

Tabel 5. 7 Kode Program Implementasi Masukan pada *Output Layer*

```
1  public static void Feedforward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        double x[] = new double[dataLatih.length];
4        Sistem.out.println("Y_net");
5        for (int b = 0; b < dataLatih.length; b++) {
6           for (int i = 0; i < 3; i++) {
7               x[b] += ((W[i + 1] * Zi[b][i]));
8               y_net[b] = W[0] + x[b];
9           }
10            Sistem.out.println("  " + y_net[b]);
11     }
12 }
```

Pada Tabel 5.7 merupakan fungsi menghitung Y_net (masukan pada *output layer*) yang ditunjukkan pada baris 1-12 . Pada fungsi ini dilakukan perulangan sebanyak jumlah data dan *hidden* untuk dilakukan perhitungan dengan menjumlahkan hasil dari perkalian antara Zi dengan bobot W.

5.3.3.4 Implementasi Keluaran pada *Output Layer*

Perhitungan keluara pada *output layer* menggunakan fungsi aktivasi *sigmoid* biner. Implementasi proses perhitungan ditunjukkan pada Tabel 5.8.

Tabel 5. 8 Kode Program Implementasi Keluaran pada *Output Layer*

```
1  public static void Feedforward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("Yk");
```

```
4         for (int b = 0; b < dataLatih.length; b++) {
5             Yk[b] = 1 / (1 + (Math.pow(eksponen, -y_net[b])));
6             Sistem.out.println("  " + Yk[b]);
7         }
8     }
9 }
```

Pada Tabel 5.8 Merupakan bagian fungsi menghitung Yk (keluaran pada *output layer*) yang ditunjukkan pada baris 1-9. Pada fungsi ini dilakukan perulangan sebanyak jumlah data untuk dilakukan perhitungan dengan menggunakan nilai Y_net yang telah didapat.

5.3.4 Implementasi *Backpropagation*

Proses perhitungan *Backpropagation* dimulai dengan menghitung galat hingga mendapatkan bobot baru dari proses perbaikan bobot.

5.3.4.1 Implementasi Galat pada *Output Layer*

Perhitungan galat *output layer* merupakan perkalian dari masukan pada *output layer* dengan hasil pengurangan antara target dengan keluaran pada *output layer.* Implementasi proses perhitungan ditunjukkan pada Tabel 5.9.

Tabel 5. 9 Kode Program Implementasi Galat pada Output Layer

```
1  public static void Backward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("------------------------------");
4        Sistem.out.println("Iterasi Ke - " + (m + 1));
5        Sistem.out.println("Backward");
6        Sistem.out.println("GAlat");
7        for (int a = 0; a < dataLatih.length; a++) {
8           galatK[a] = (konversi_kelas[a] - Yk[a]) * Yk[a] *
9           (1 - Yk[a]);
10           Sistem.out.print("  " + galatK[a]);
11       }
12    }
13 }
```

Pada Tabel 5.9 merupakan fungsi menghitung galat pada *output layer* yang ditunjukkan pada baris 1-13 .

5.3.4.2 Implementasi Perbaikan Bobot W

Perhitungan perbaikan bobot dengan melakukan perkalian antara nilai *alpha* dengan nilai *error output* dan masukan pada *hidden layer*. Implementasi proses perhitungan ditunjukkan pada Tabel 5.10.

Tabel 5. 10 Kode Program Implementasi Perbaikan Bobot W

```
1  public static void Backward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("DeltaW");
4        for (int i = 0; i < dataLatih.length; i++) {
5           deltaWk[i] = learningRate * galatK[i];
6           for (int j = 0; j < 3; j++) {
7              deltaW[i][j] = learningRate * galatK[i] *
8               Zi[i][j];
9              Sistem.out.print("  " + deltaW[i][j]);
10          }
11       }
12    }
13 }
```

Pada Tabel 5.10 merupakan fungsi menghitung perbaikan bobot yang ditunjukkan pada baris 1-13. Pada baris 6-9 merupakan fungsi menghitung deltaW. Pada baris 5 merupakan fungsi menghitung deltaW0k.

5.3.4.3 Implementasi Galat pada *Hidden Layer*

Perhitungan galat *hidden layer* merupakan hasil perkalian dari galat_net dengan keluaran pada *output layer.* Galat_net sendiri merupakan hasil penjumlahan dari perkalian inisialisasi bobot pada *Nguyen-widrow* dengan galat pada *output layer*. Implementasi proses perhitungan ditunjukkan pada Tabel 5.11.

Tabel 5. 11 Kode Program Implementasi Galat pada Hidden Layer

```
1  public static void Backward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("Galatnet");
4        for (int i = 0; i < dataLatih.length; i++) {
5           for (int j = 0; j < 3; j++) {
6              galatnet[i][j] = galatK[i] * deltaW[i][j];
7              galatJ[i][j] = galatnet[i][j] * Zi[i][j] * (1 –
8               Zi[i][j]);
9              Sistem.out.print("  " + galatJ[i][j]);
10          }
12       }
13    }
14 }
```

Pada Tabel 5.11 merupakan fungsi menghitung galat pada *hidden layer* yang ditunjukkan pada baris 1-14. Pada baris 6 merupakan perhitungan galat_net. Untuk baris 7 merupakan perhitungan galat.

5.3.4.4 Implementasi Perbaikan Bobot V

Perhitungan perbaikan bobot dengan melakukan perkalian antara nilai *alpha* dengan nilai *error hidden* dan masukan pada *hidden layer*. Implementasi proses perhitungan ditunjukkan pada Tabel 5.12.

Tabel 5. 12 Kode Program Implementasi Perbaikan Bobot V

```
1  public static void Backward(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        Sistem.out.println("Galatnet");
4        for (int i = 0; i < dataLatih.length; i++) {
5           for (int j = 0; j < 3; j++) {
6              deltaV[i][j] = learningRate * galatJ[i][j] *
7               Zi[i][j];
8              Sistem.out.print("  " + deltaV[i][j]);
9           }
10         }
12      }
13 }
```

Pada Tabel 5.12 merupakan fungsi menghitung perbaikan bobot yang ditunjukkan pada baris 1-13. Pada baris 6 merupakan fungsi menghitung deltaV.

5.3.5 Implementasi Perubahan Bobot

Perhitungan bobot terbaru merupakan hasil dari penjumlahan bobot inisialisasi dengan metode *Nguyen-widrow* dengan hasil koreksi bobot yang dilakukan setelah perbaikan bobot. Implementasi proses perhitungan ditunjukkan pada Tabel 5.13.

Tabel 5. 13 Kode Program Implementasi Perubahan Bobot

```
1  public static void PerubahanBobot(int iterasi) {
2     for (int m = 0; m < iterasi; m++) {
3        if (m == 0) {
4           Sistem.out.println("Wbaru");
5           for (int i = 0; i < dataLatih.length; i++) {
6              for (int j = 0; j < 4; j++) {
7                  Wbaru[i][j] = W[j] + deltaW[i][0];
8                  Sistem.out.print("  " + Wbaru[i][j]);
9              }
```

```
10          }
11          Sistem.out.println("VBaru");
12          for (int i = 0, x = 0; i < deltaV[0].length; i++) {
13             Sistem.out.println("i" + i);
14             x++;
15             for (int j = 0; j < deltaV.length; j++) {
16                Sistem.out.println("j" + j);
17                for (int k = 0; k < VijBaru[0].length; k++) {
18                   Vbaru[i][x] = deltaV[j][i] +
19                   VijBaru[i][k];
20                   Sistem.out.print("  " + Vbaru[i][x]);
21                }
22             }
23          }
24          for (int i = 0; i < dataLatih.length; i++) {
25             eror[i] = konversi_kelas[i] - Yk[i];
26             pangkattes += Math.pow(eror[i], 2);
27          }
28           mse = 1 / dataLatih.length * pangkattes;
29           if (mse < epsilon) {
30              iterasi = 1;
31           } else {
32              iterasi = iterasi;
33           }
34      } else {
35           for (int a = 0; a < hiddenLayer; a++) {
36              for (int b = 0; b < 16; b++) {
37                  VijBaru[a][b] = Vbaru[a][7];
38                  Sistem.out.print("  " + VijBaru[a][b]);
39              }
40      }
41           for (int i = 0; i < 4; i++) {
42               W[i] = Wbaru[7][i];
43               Sistem.out.println("  " + W[i]);
44           }
45      }
46  }
```

Pada tabel 5.13 merupakan proses perhitungan bobot baru yang ditunjukkan pada baris 1-46 . Pada baris 11-20 merupakan fungsi *update* bobot V terbaru dan dilanjutkan untuk melakukan *update* bobot W pada baris 4-8.

5.3.6 Implementasi Denormalisasi Data

Data hasil klasifikasi atau nilai keluaran pada *output layer* dilakukan denormalisasi untuk mengembalikan nilai ke semula. Implementasi proses denormalisasi data ditunjukkan pada Tabel 5.14.

Tabel 5. 14 Kode Program Implementasi Denormalisasi Data

```
1  public static void Denormalisasi() {
2     Sistem.out.println("Denormalisasi");
3     for (int i = 0; i < dataTesting.length; i++) {
4        denormalisasi[i] = Math.round((((YkUji[i] - 0.1) * (2
5        - 1)) / 0.8) + 1.0);
6        Sistem.out.println(" " + denormalisasi[i]);
7     }
8  }
```

Pada tabel 5.14 merupakan fungsi denormalisasi yang ditunjukkan pada baris 1-8 . Pada fungsi ini dilakukan perulangan sebanyak jumlah data untuk menghitung denormalisasi dengan menggunakan Yk, maksimum data, dan minimum data.

5.3.7 Implementasi Perhitungan RMSE

Perhitungan RMSE menggunakan data yang telah dinormalisasi dan nilai keluaran pada *output layer*. Implementasi proses perhitungan ditunjukkan pada Tabel 5.15.

Tabel 5. 15 Kode Program Implementasi Perhitungan RMSE

```
1  public static void Error() {
2     Sistem.out.println("DenormalisasiUji");
3     for (int i = 0; i < dataTesting.length; i++) {
4        denor[i] = (((YkUji[i] - 0.1) * (2 - 1)) / 0.8) + 1.0;
5        Sistem.out.println(" " + denormalisasi[i]);
6     }
7     for (int i = 0; i < dataTesting.length; i++) {
8        erorUji[i] = konversi_kelas[i] - denor[i];
9        pangkatUji += Math.pow(erorUji[i], 2);
10       Sistem.out.println("ErorUji = " + erorUji[i]);
11    }
12    rmse = Math.sqrt(pangkatUji / dataTesting.length);
13    akurasiUji = 100 - rmse;
```

```
14      Sistem.out.println("Rmse" + rmse);
15      Sistem.out.println("Akurasi Uji = " + akurasiUji);
16  }
```

Pada tabel 5.15 merupakan proses perhitungan untuk menghitung *error* dengan rumus RMSE. Pada baris 7-10 merupakan perhitungan untuk mendapatkan nilai *error* yang didapat dari selisih hasil klasifikasi dari data asli. Baris 12 merupakan rumus RMSE dimana dilakukan penjumlahan dari *error* yang dikuadratkan kemudian dibagi dengan jumlah data. Selanjutnya di akar untuk menghasilkan nilai *error* RMSE.

5.4 Implementasi Antarmuka

Implementasi antarmuka memberikan penjelasan mengenai antarmuka dari implementasi *Nguyen-widrow* pada *Backpropagation* untuk klasifikasi rumah layak huni. Implementasi antarmuka terdiri dari:

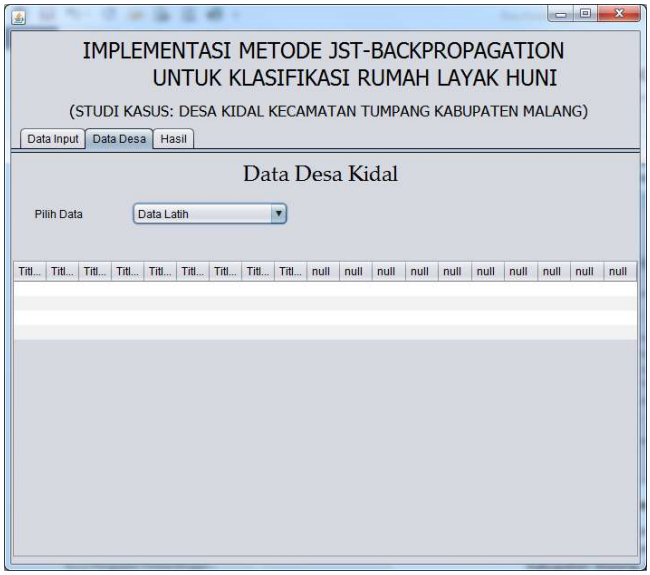
5.4.1 Implementasi Halaman Utama

Halaman utama merupakan halaman yang menyediakan form untuk memasukkan data parameter yang akan digunakan dalam perhitungan. Gambar 5.1 merupakan *screenshot* halaman utama.

Gambar 5. 1 Implementasi Halaman Utama

5.4.2 Implementasi Halaman Data Desa

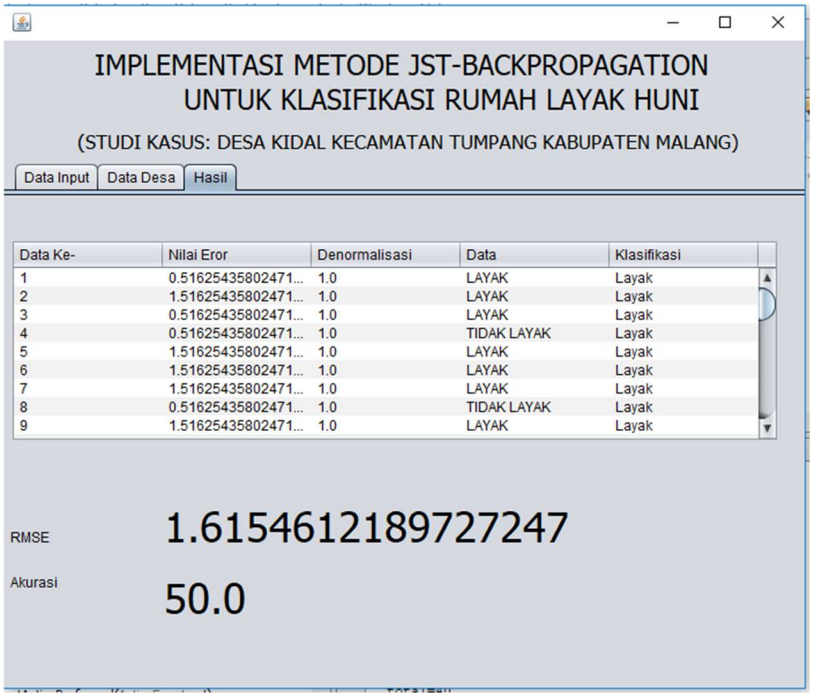
Halaman data desa merupakan halaman yang menampilkan data rumah layak huni dan rumah tidak layak huni di Desa Kidal Kecamatan Tumpang Kabupaten Malang. Pada halaman data desa terdapat dua macam data yaitu data latih dan data uji. Gambar 5.2 merupakan *screenshot* halaman data desa.



Gambar 5. 2 Implementasi Halaman Data Desa

5.4.3 Implementasi Halaman Hasil

Halaman hasil merupakan halaman yang menampilkan hasil dari perhitungan menggunakan *Backpropagation* dengan inisialisasi bobot *Nguyen-widrow* yang telah dilakukan. Halaman hasil berisi nilai denormalisasi, nilai *error*, nilai RMSE, dan nilai akurasi. Gambar 5.3 merupakan *screenshot* halaman hasil.



Gambar 5. 3 Implementasi Halaman Hasil