BAB IV
IMPLEMENTASI

4.1. Lingkungan Implementasi

Dalam proses tahap implementasi, lingkungan sistem merupakan sebuah factor yang harus diperhatikan. Faktor lingkungan sistem ini meliputi lingkungan perangkat keras dan lingkungan perangkat lunak. Lingkungan perangkat lunak dank eras ini harus diperhatikan untuk dapat memenuhi kebutuhan sistem.

4.1.1. Lingkungan Perangkat Keras

Perangkat keras yang digunakan untuk melaksanakan penelitian tentang pengklasifikasian genre musik berdasarkan fitur audio dengan menggunakan *Fuzzy Support Vector Machine* adalah:

1. Prosesor Intel® Core™ i3 CPU @2.67GHz
2. Memori RAM 2GB
3. *Harddisk* 500GB
4. Monitor
5. *Keyboard*
6. *Mouse*

4.1.2. Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan untuk melakukan penelitian Implementasi *Fuzzy Support Vector Machine* untuk pengklasifikasian genre musik berdasarkan fitur audio adalah:

1. Sistem Operasi Windows 8 Profesional.
2. Aplikasi dibangun dengan bahasa pemrograman Java menggunakan *Java Development Kit* (JDK) 1.7.0_21 dan ditulis melalui Netbeans 7.3. dengan menggunakan Maven *build automation tool*
3. Microsoft Word 2010,dan Microsoft Excel 2010.

4.2. Implementasi Program

Dalam tahap implementasi, program dibuat dalam bentuk prosedur dan kelas yang mencakup tahapan-tahapan yang dipaparkan dalam metodologi penelitian. Dalam tahapan implementasi proses yang dilakukan adalah proses proses ekstraksi fitur, proses pelatihan,serta proses klasifikasi dan perhitugan akurasi.

4.1.1. Proses Ekstraksi Fitur

Proses Ekstraksi Fitur dilakukan dengan melakukan beberapa proses yang terdapat dalam beberapa kelas. Proses ekstraksi fitur dilakukan melalui beberapa tahapan meliputi load audio sample, normalisasi, *framing*, *windowing*, FFT, Mel-*Filterbank, Discreete Cosine Transform.*

4.1.1.1.Proses load audio sample

Dalam proses ini, audio sample yang berformat wav akan diambil data samplenya dan dimasukkan kedalam sebuah array int. Proses ini dilakukan untuk masing-masing data. Proses ini dilakukan dengan menggunakan kelas AudioFormat. Gambar 4.1 menunjukkan kode program proses load audio sample.

```
public int[] extractAmplitudo() {
          bis = new ByteArrayInputStream(arrFile);
          try {
              audioInputStream = AudioSystem.getAudioInputStream(bis);
              format = audioInputStream.getFormat();
              audioBytes = new byte[(int)
(audioInputStream.getFrameLength() * format.getFrameSize())];
              durationMSec = (long)
((audioInputStream.getFrameLength() * 1000) /
audioInputStream.getFormat().getFrameRate());
              durationSec = durationMSec / 1000.0;
              audioInputStream.read(audioBytes);
              audioData=null;
              if (format.getSampleSizeInBits() == 16) {
                  int nlengthInSamples = audioBytes.length / 2;
                  audioData = new int[nlengthInSamples];
                  if (format.isBigEndian()) {
                      for (int i = 0; i < nlengthInSamples; i++) {
                          int MSB = audioBytes[2 * i];
```

```
                        int LSB = audioBytes[2 * i + 1];
                    }
                } else {
                    for (int i = 0; i < nlengthInSamples; i++) {
                        int LSB = audioBytes[2 * i];
                        int MSB = audioBytes[2 * i + 1];
                        audioData[i] = MSB << 8 | (255 & LSB);
                    }
                }
            }
        } catch (UnsupportedAudioFileException e) {
            System.out.println("unsupported file type, during
extract amplitude");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("IOException during extracting
amplitude");
            e.printStackTrace();
        }
        return audioData;
    }
```

Gambar 4.1 *Source Code* Proses Load Data

4.1.1.2.Proses Normalisasi

Proses normalisasi merupakan proses yang dilakukan untuk mengubah nilai data sample yang berupa integer menjadi sebuah nilai double dengan rentang 1-100. Dikarenakan data audio merupakan data 16 bit. Nilai pembagi yang digunakan adalah 32767 yang merupakan nilai maksimal pada data 16 bit. Proses normalisasi ditunjukkan oleh Gambar 4.2

```
private double[] Normalisasi(int sample[]){
        double x[]=new double[sample.length];
        for (int i=0;i<x.length;i++){
        if
(sample[i]>0)x[i]=((double)sample[i]/(double)32767)*((do
uble)100);
        else
x[i]=((double)sample[i]/(double)32768)*((double)100);
        }
        return x;
}
```

Gambar 4.2 *Source Code* Proses Normalisasi

4.1.1.3.Proses Framing

Pada proses ini masing-masing sample yang ada akan dibagi kedalam sejumlah frame yang berdimensi M serta memiliki jarak N pada data sample original.Proses Framing ini akan dilakukan untuk masing-masing data audio baik data testing maupun data training. Gambar 4.3 menunjukkan proses framing pada sample ternormalisasi.

```
public Framing(double S[],int M,int N_Ms,int FrameRate){
       this.S=S;
       this.N_Ms=N_Ms;
       M_Sample=M;
       N_Sample=(int)(N_Ms*FrameRate/1000);
       jumFrame=(int)(S.length/(M_Sample+N_Sample));
       Frame=new double[jumFrame][M_Sample];
       int i=0,k=0;
        while (i<S.length && k<jumFrame){
            int j=0;
            while (j<M_Sample){
                //Frame[k][j]=S[i];
                Frame[k][j]=Windowing(S[i],j);
                j++;i++;
            }
            k++;
            i=i+N_Sample;
        }
    }
```

Gambar 4.3 *Source Code* Proses Framing

4.1.1.4.Proses Windowing

Pada proses *windowing* digunakan metode *hamming window* yang dijelaskan pada persamaan 2.1.Proses windowing dilakukan untuk masing-masing data sample audio dalam masing-masing fame. Proses *windowing* akan ditunjukkan pada Gambar 4.4.

```
private double Windowing (double data,int x){
        return data*HammingWindow(x);
    }
private double HammingWindow(int x){
        return ((double)0.54-((double)0.46 *Math.cos
                (((double)2.0*Math.PI*x)/M_Sample)));
}
```

Gambar 4.4 *Source Code* Proses Windowing

4.1.1.5.Proses *Fast Fourier Transform*

Dengan proses Fast Fourier Transform data sample yang berdomain waktu dirubah menjadi berdomain frequency. Proses FFT dilakukan secara rekursif dan akan menghasilkan sejumlah bilangan kompleks. Untuk melakukan operasi bilangan kompleks dibutuhkan sebuah object bilangan kompleks. Dengan menggunakan object bilangan kompleks dilakukan secara rekursif. Gambar 4.5 menunjukkan proses FFT dengan object bilangan kompleks.

```
private BilanganKompleks []fft(BilanganKompleks x[]){
        int N=x.length;
        if (N==1)return x;
        else {
            BilanganKompleks []ganjil=new BilanganKompleks[N/2];
            BilanganKompleks []genap=new BilanganKompleks[N/2];
            for (int k=0;k<N/2;k++){
                genap[k]=x[2*k];ganjil[k]=x[(2*k)+1];
            }
            BilanganKompleks O[]=fft(ganjil);
            BilanganKompleks E[]=fft(genap);
            BilanganKompleks X[]=new BilanganKompleks[N];
            for (int k=0;k<N/2;k++){
                double teta=(double)(-2)*(double)k*Math.PI/(double)N;
                BilanganKompleks W=new
BilanganKompleks(Math.cos(teta),Math.sin(teta));
                X[k]=E[k].jumlah(W.kali(O[k]));
                X[k+(N/2)]=E[k].kurangi(W.kali(O[k]));
```

```
            }
            return X;
        }
}
```

Gambar 4.5 *Source Code* Proses *Fast Fourier Transform*

Hasil transformasi fourier berupa bilangan kompleks. Bilangan kompleks ini akan dikomputasikan sehingga diperoleh nilai magnitude dari masing-masing data pada sebuah frame. Proses komputasi nilai magnitude pada masing-masing data bilangan kompleks ditunjukkan olleh gambar 4.6.

```
public double magnitude(){
        return Math.sqrt(Math.pow(real,2)+Math.pow(imaginer,2));
}
```

Gambar 4.6 *Source Code* Proses *Fast Fourier Transform*

4.1.1.6.Proses Filtrasi *Mel-Filterbank*

Dalam proses filtrasi *mel-filterbank* dilakukan filtrasi data frekuensi pada masing-masing frame. Filtrasi dilakukan terhadap sejumlah n *mel-filter* dimana n akan menentukan jumlah fitur yang digunakan pada proses klasifikasi. Proses filtrasi Mel-Filterbank ditunjukkan dengan Gambar 4.7 .

```
public MelFilterBank(double Fatas, double Fbawah, int
JumlahFilterbank,double F[]){
        this.Fatas=Fatas;
        this.Fbawah=Fbawah;
        Matas=FtoM(Fatas);
        Mbawah=FtoM(Fbawah);
        NFilter=JumlahFilterbank;
        double interval=(Matas-Mbawah)/(NFilter+(double)1);
        MFilterPoint=new double[NFilter+2];
        FFilterPoint=new double[NFilter+2];
        FilterBank=new double[NFilter][F.length];
        for (int i=0;i<NFilter+2;i++){
            MFilterPoint[i]=Mbawah+(interval*i);
            FFilterPoint[i]=MtoF(MFilterPoint[i]);
        }
        for (int i=1;i<NFilter+1;i++){
            for (int j=0;j<F.length;j++){
                if (FFilterPoint[i-
```

```
                    1]<=F[j]&&F[j]<=FFilterPoint[i]){
                                    FilterBank[i-1][j]=(F[j]-
                    FFilterPoint[i-1])/(FFilterPoint[i]-
                    FFilterPoint[i-1]);
                }
                else if
                    (FFilterPoint[i]<=F[j]&&F[j]<=FFilterPoint[
                    i+1]){
                                    FilterBank[i-
                    1][j]=(FFilterPoint[i+1]-
                    F[j])/(FFilterPoint[i+1]-FFilterPoint[i]);
                }
                else FilterBank[i-1][j]=0;
            }
        }
    }
```

Gambar 4.7 *Source Code* Proses Filtrasi *Mel-Filterbank*

4.1.1.7.Proses *Discreete Cosine Transform*

Hasil dari filtrasi terhadap *mel-filterbank* akan ditransformasikan kembali dengan menggunakan discreete cosine transform. Hasil transformasi ini akan dirata-rata untuk tiap frame pada sebuah file audio. Rata-rata nilai transformasi ini merupakan fitur yang digunakan untuk pelatihan dan pengklasifikasian. Proses Discreete Cosine Transform ditunjukkan pada Gambar 4.8

```
private double DCT(double []Sk,int NFilter,int n){
        double c=0;
        for (int i=1;i<=NFilter;i++){
         if (Sk[i-1]!=0) c=c+((Math.log10(Sk[i-1])
          *Math.cos(n*((double)i-(double)0.5)*(Math.PI/NFilter))));
         else c=c+0;
        }
        return c;
}
```

Gambar 4.8 *Source Code* Proses *Discreete Cosine Transform*

4.1.2. Proses Pelatihan

Dalam proses pelatihan, data yang digunakan dapat berasal langsung dari data audio atau menggunakan dataset hasil ekstraksi fitur yang telah disimpan sebelumnya. Dataset yang disimpan sebelumnya disimpan dalam format data .csv. Untuk proses pelatihan dilakukan dalam beberapa tahapan meliputi proses load

data, proses komputasi hessian matrix, proses optimasi nilai alfa, dan proses komputasi w dan b optimum.

4.1.1.1. Proses Load Data *Training*

Proses Load Data bertujuan untuk mengambil data hasil ekstraksi fitur yang telah disimpan sebelumnya dalam sebuah file berekstensi .csv. Dalam proses ini data yang di *load* merupakan data yang digunakan untuk proses pelatihan. Gambar 4.9

```
public void loadTraining(String path){
   BufferedReader br;
            try {
                Fitur=
Integer.parseInt(FiturTextField.getText().trim());
                File x=new
File(path+"\\"+datasetFileName.getText().trim()+"-
"+Fitur+".csv");
                br = new BufferedReader( new FileReader(x));
                String strLine = "";
                StringTokenizer st ;
                String genre="";
                for (int i=0;i<5;i++){
                    String nama[]=new String[50];
                    double C[][]=new double[50][Fitur];
                    for (int j=0;j<50;j++){
                       strLine=br.readLine();
                       st = new StringTokenizer(strLine, ",");
                       int index=0;
                       while(st.hasMoreTokens())
                       {
                            if
(index==0)nama[j]=st.nextToken();
                            else if (index==1)genre="";
                            else C[j][index-
2]=Double.valueOf(st.nextToken());
                            index++;
                       }
                    }
                    dataTesting[i]=new LoadData(genre,nama,C);
                }
        } catch (FileNotFoundException ex) {
            Logger.getLogger(AntarmukaSistem.class.getName()).log(Level.SEVERE, null,
ex);
```

```
        } catch (IOException ex) {
        Logger.getLogger(AntarmukaSistem.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Gambar 4.9 *Source Code* Proses *Load* Data *Training*

4.1.1.2. Proses Komputasi Hessian Matrix

Untuk dapat melakukan optimasi nilai alfa, data yang didapat dari ekstraksi fitur maupun dari file csv harus disusun kedalam sebuah matrix hessian. Dalam permasalahan ini nilai dot dari data yang ada digantikan dengan fungsi *Gaussian RBF* yang merupakan fungsi kernel yang digunakan. Nilai alfa akan memberikan gambaran, data mana saja yang merupakan support vector. Proses komputasi hessian matrix dapat dilihat pada Gambar 4.10

```
private double[][] hessian(double X[][],double Y[],int
sigma){
    double H[][]=new double[Y.length][Y.length];
    for (int i=0;i<Y.length;i++){
        for (int j=0;j<Y.length;j++){
            H[i][j]=Hij(X,i,j,sigma);
        }
    }
    return H;
    }
private double Hij(double X[][],int i,int j,int sigma){
    return GaussianKernel(X[i],X[j],sigma);
}
private double GaussianKernel(double xi[],double xj[],int
sigma){
    double p=0;
    for (int i=0;i<xi.length;i++){
        p=p+Math.pow(xi[i]-xj[i],2);
    }
    return Math.exp(((-1)*p)/(2*Math.pow(sigma,2)));
}
```

Gambar 4.10 *Source Code* Proses Komputasi Hessian Matrix

4.1.1.3. Proses Optimasi Nilai Alfa

Dalam proses pelatihan proses optimasi nilai alfa dilakukan dengan menggunakan library svm yang bernama libsvm. Dalam proses ini harus diinputkan sebuah nilai variable C yang merupakan variable slack. Proses Optimasi Nilai Alfa dapat dilihat pada Gambar 4.11

```
private void optimasiAlfa(svm_problem masalah,
svm_parameter parameter){
    hasil = svm.svm_train(masalah, parameter);
    svm_node[][] support=hasil.SV;
    svIndex=new int [hasil.SV.length];
    for (int i=0;i<support.length;i++){
        svIndex[i]=(int) support[i][0].value;
    }
    alfa=new double[y.length];
    for (int i=0;i<y.length;i++){
        boolean isSV=false;
        for (int j=0;j<support.length;j++){
            if (svIndex[j]==i){
            isSV=true;
            alfa[i]=hasil.sv_coef[0][j];
            break;}
    }
    if (!isSV)alfa[i]=0;
    }
}
```

Gambar 4.11 *Source Code* Proses Optimasi Nilai Alfa

4.1.1.4. Proses Komputasi W dan B optimum

Proses komputasi w dan b optimum dilakukan dengan menggunakan persamaan 2.24 dan 2.25. Hasil nilai w dan b untuk masing-masing pasangan kelas ini akan digunakan untuk klasifikasi FSVM pada tahap berikutnya. Proses Komputasi w dan b optimum ditunjukkan oleh Gambar 4.12

```
private void WBoptimum (){
  w=new double[dataI.get(0).getCAvg().length];
  for (int i=0;i<w.length;i++){
     w[i]=0;
     for (int j=0;j<y.length;j++){
         w[i]=w[i]+(alfa[j]*y[j]*x[j][i]);
     }
 }
     double SV[][]=new
double[hasil.nSV[0]][dataI.get(0).getCAvg().length];
     for (int i=0;i<SV.length;i++){
     if (svIndex[i]>0){
     double temp[]=x[svIndex[i]];
     SV[i]=temp ;}
  }
  B=hasil.rho[0];
}
```

Gambar 4.12 *Source Code* Proses Komputasi W dan B optimum

4.1.3. Proses Klasifikasi dan Perhitungan Akurasi

Proses Klasifikasi dilakukan dengan menggunakan nilai W dan B yang diperoleh dari data latih untuk menentukan nilai keanggotaan dari sebuah data terhadap sebuah kelas. Untuk melakukan klasifikasi harus dilakukan proses ekstraksi fitur terhadap data audio yang akan diujikan. Data uji yang telah diekstraksi fiturnya dapat disimpan untuk digunakan kembali. Proses Klasifikasi dan perhitungan akurasi dengan FSVM dapat dilihat pada Gambar 4.13

```
for (int i=0;i<dataTesting[l].getData().size();i++){
     double []membership=new double[pelatihan.length];
     String []kelas=new String[pelatihan.length];
     for (int j=0;j<pelatihan.length;j++){
         if (j==0)kelas[j]=pelatihan[j][1].GenreI;
         else kelas[j]=pelatihan[j][0].GenreI;
         membership[j]=0;
         for (int k=0;k<pelatihan[j].length;k++){
             if (j!=k){
             double W[]=pelatihan[j][k].getW();
             double m=0;
```

```
            for (int
h=0;h<dataTesting[l].getData().get(i).getCAvg().length;h++)
{

m=m+(W[h]*dataTesting[l].getData().get(i).getCAvg()[h]);
            }
            m=m+pelatihan[j][k].getB();
            if (membership[j]>m)membership[j]=m;
            }
        }
    }
    int indexMax=0;
    for (int j=0;j<membership.length;j++){
        if (membership [j]>1){indexMax=j;break;}
        if (membership[j]>membership[indexMax])indexMax=j;
    }

dataTesting[l].getData().get(i).setGenre(kelas[indexMax]);
    dataUji.addRow(new
String[]{dataTesting[l].getData().get(i).getNama(),dataTest
ing[l].getData().get(i).getGenre()});
    String Nama=dataTesting[l].getData().get(i).getNama();
    String GenreLabel=Nama.substring(0,
Nama.indexOf("."));

    if
(GenreLabel.equalsIgnoreCase(dataTesting[l].getData().get(i
).getGenre()))benar++;
    jumData++;
}}}
```

Gambar 4.13 *Source Code* Proses Klasifikasi

4.3. Implementasi Antarmuka

Dalam aplikasi ini, implementasi antarmuka terbagai dalam 2 bagian utama, yaitu:

1. Form Pelatihan

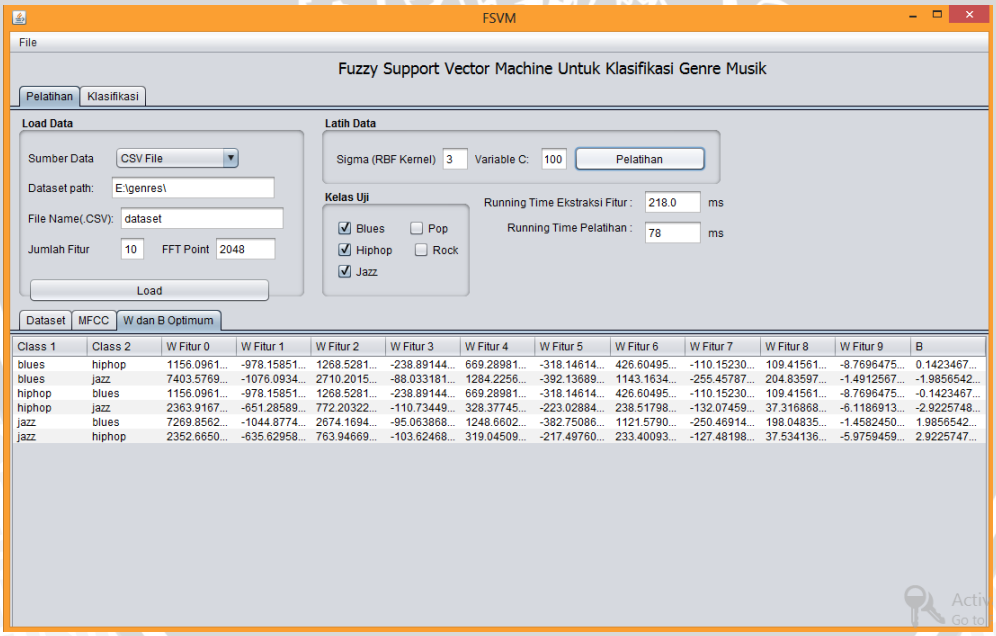
   Form Pelatihan digunakan untuk melakukan *load* data dari data audio maupun data csv untuk pelatihan serta melakukan pelatihan terhadap FSVM sesuai kelas yang diinginkan.

2. Form Klasifikasi

   Form Klasifikkasi digunakan untuk melakukan *load* data audio maupun data csv yang akan diklasifikasikan, melakukan klasifikasi terhadap data yang telah dimuat, serta menghitung tingkat akurasi dari data yang ada.

4.1.1. Form Pelatihan

Form Pelatihan merupakan form yang digunakan untuk mengekstraksi fitur atau memuat data latih sekaligus untuk melakukan klasifikasi sesuai dengan parameter yang ada. Gambar 4.14 menunjukkan implementasi form ekstraksi fitur. Pada Form Pelatihan pengguna dapat menginputkan jenis sumber data(file audio atau file csv) serta lokasi data yang akan digunakan. Selain itu terdapat pula text field untuk melakukan inputan jumlah fitur serta FFT point yang merupakan sebagian parameter uji. Saat pengguna menekan tombol Load proses ekstraksi fitur akan dilakukan jika pengguna menggunakan sumber data File Audio,



Gambar 4.14 Form Pelatihan

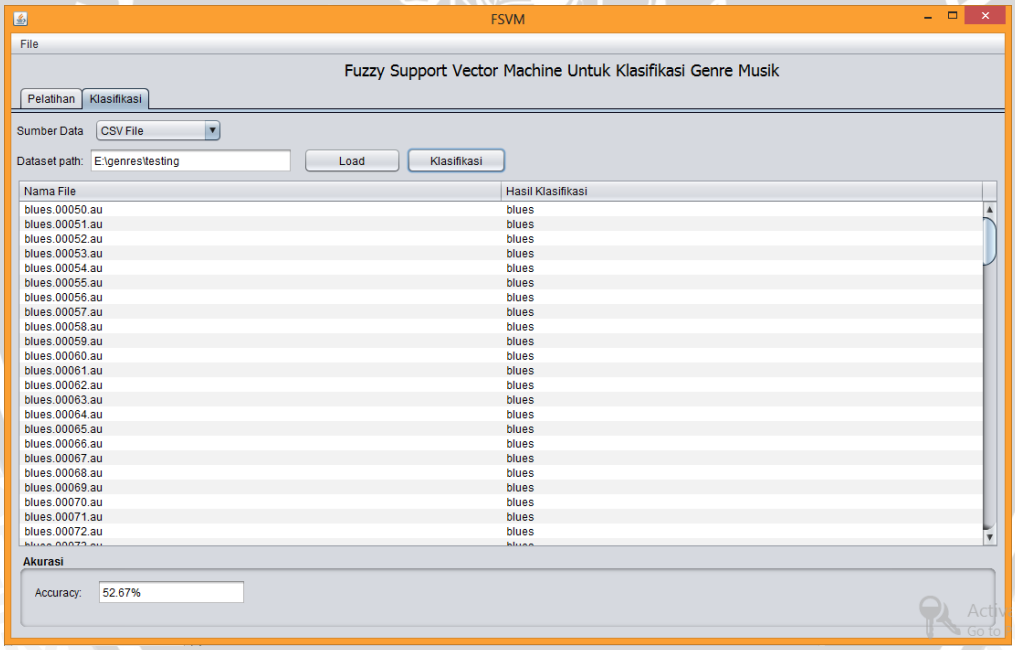
Pada bagian latih data terdapat dua parameter yang akan digunakan untuk pelatihan yaitu Sigma serta C. Tombol Pelatihan hanya dapat digunakan jika pengguna telah melakkan mekanisme load data. Pada bagian kelas uji terdapat

beberapa pilihan kelas sesuai batasan masalah. Kelas yang terpilih merupakan kelas yang digunakan untuk melakukan klasifikasi.

Pada Form ini terdapat pula *texfield running time* ekstraksi fitur serta *running time* pelatihan. Kedua *textfield* ini berfungsi untuk mengetahui waktu eksekusi pelatihan dan ekstraksi fitur. Pengguna baru dapat memasuki form klasifikasi pada saat pengguna telah melakukan pelatihan FSVM.

4.1.2. Form Klasifikasi

Form Klasifikasi merupakan form yang berfungsi untuk melakukan klasifikasi pada data uji serta melakukan perhitungan akurasi pada hasil klasifikasi. Gambar 4.15 menunjukkan hasil implementasi form klasifikasi.



Gambar 4.15 Implementasi Form Klasifikasi

Pada Form Pengujian ini user dapat memilih sumber data uji serta lokasinya. Tombol Load berfungsi untuk memuat data kedalam sistem dan tombol klasifikasi berfungsi untuk melakukan klasifikasi pada data yang telah termuat.