BAB 4. IMPLEMENTASI SISTEM

Pada bab ini akan dibahas tentang implementasi sistem sesuai dengan teori dan desain sistem pada bab-bab sebelumnya. Implementasi sistem meliputi implementasi aplikasi, implementasi *class* GLCM dan implementasi *class* FuzzySet.

4.1. Implementasi Aplikasi

Aplikasi pengenalan pola batik ini dibuat menggunakan bahasa pemrograman C# dengan *Microsoft Visual Studio 2010* sebagai *Integrated Development Environment (IDE)*. Alasan penggunaan C# adalah kemudahan dalam pembuatan *Graphical User Interface (GUI)* menggunakan komponen *Windows Form* dan banyaknya fasilitas yang disediakan *framework .Net*. Dalam pembuatan aplikasi ini digunakan *library Emgu.CV* untuk melakukan *histogram equalization* dan untuk kemudahan dalam mengubah *color space* gambar. Pembuatan aplikasi ini menggunakan paradigma pemrograman berbasis obyek dan pemrograman prosedural.

4.2. Implementasi Static *Class* GLCM dan Sistem Ekstraksi Fitur

Static Class GLCM adalah implementasi dari desain class GLCM pada subbab 3.2.1. Semua *data member* dan *function* dalam *class* ini bersifat *static*, sehingga *class* ini dapat selalu digunakan sewaktu-waktu. Data hasil sementara dari masing-masing proses akan dikembalikan ke program utama dan menjadi parameter pada proses berikutnya. Daftar fungsi dan prosedur yang digunakan dalam sistem ekstraksi fitur dapat dilihat pada Tabel 4.1.

Tabel 4.1 Daftar fungsi dan prosedur untuk sistem ekstraksi fitur

Segmen Program	Nama Fungsi / Prosedur	Keterangan	*Flowchart*
4.1	GLCM (*constructor*)	Menginisialisasi nilai *level* dan *offset* pada *class* GLCM	-
4.2	Create GLCM	Menghitung GLCM dari gambar	*Flowchart* 3.4

Tabel 4.1 Daftar fungsi dan prosedur untuk sistem ekstraksi fitur (lanjutan)

4.3	Hitung *Correlation*	Menghitung nilai *Correlation* dari GLCM yang diinputkan	*Flowchart* 3.6
4.4	Hitung *Contrast*	Menghitung nilai *Contrast* dari GLCM yang diinputkan	Flowchart 3.7
4.5	Hitung *Energy*	Menghitung nilai *Energy* dari GLCM yang diinputkan	*Flowchart* 3.8
4.6	Hitung *Homogeneity*	Menghitung nilai *Homogeneity* dari GLCM yang diinputkan	*Flowchart* 3.9
4.7	Hitung RowMean	Menghitung *weighted mean* dari GLCM dengan *weight* berupa indeks baris	*Flowchart* 3.10
4.8	Hitung ColMean	Menghitung *weighted mean* dari GLCM dengan *weight* berupa indeks kolom	*Flowchart* 3.11
4.9	Hitung RowStdDev	Menghitung standar deviasi dari GLCM dengan *weight* berupa indeks baris	*Flowchart* 3.12
4.10	Hitung ColStdDev	Menghitung standar deviasi dari GLCM dengan *weight* berupa indeks kolom	*Flowchart* 3.13
4.11	Scale	Mengubah rentang nilai piksel gambar	*Flowchart* 3.14
4.12	Hitung Fitur	Melakukan input gambar, penghitungan fitur, hingga memasukkan fitur ke dalam tabel	*Flowchart* 3.3

4.2.1. ***GLCM***

GLCM adalah *constructor* untuk melakukan inisialisasi nilai *offset* dan *level* dalam *static class* GLCM. Prosedur ini adalah *static constructor* yang berarti prosedur ini tidak dipanggil secara manual dalam program utama dan terpanggil secara otomatis. Karena merupakan *static constructor*, prosedur ini tidak mempunyai *parameter*. Segmen program untuk prosedur ini dapat dilihat pada Segmen 4.1.

Segmen 4.1 *Source code constructor GLCM*

```
static GLCM() {
    _level = 3; _offsetSize = 4;
    _offsets = new Tuple<int, int>[_offsetSize];
    _offsets[0] = new Tuple<int, int>(0, 1);
    _offsets[1] = new Tuple<int, int>(1, 1);
    _offsets[2] = new Tuple<int, int>(1, 0);
    _offsets[3] = new Tuple<int, int>(1, -1);
}
```

4.2.2. ***Create GLCM***

Create GLCM adalah fungsi yang digunakan untuk membuat GLCM dari gambar batik. Fungsi ini memiliki 2 buah *parameter* yaitu gambar *grayscale* yang akan diolah dan sebuah variabel *boolean* untuk penanda apakah GLCM yang dihasilkan dinormalisasi atau tidak. Untuk dapat diproses lebih lanjut, GLCM harus dinormalisasi terlebih dahulu, sehingga parameter *boolean* itu hanya berguna saat GLCM yang dihasilkan perlu ditampilkan. Hasil dari fungsi ini adalah GLCM sejumlah *offset* yang digunakan. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.2.

Segmen 4.2 *Source code* fungsi *Create GLCM*

```
public static double[,,] CreateGLCM(Image<Gray, Byte> img,
    bool normalize = true)
{
    int size = (1 << level);
    double increment;
    double[, ,] matrix = new double[size, size, _offsetSize];
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            for (int k = 0; k < _offsetSize; k++)
                matrix[i, j, k] = 0.0;
    for (int k = 0; k < _offsetSize; k++)
    {
        increment = (double)(1.0 / (2 * (img.Width –
            Math.Abs(_offsets[k].Item2)) * (img.Height –
            Math.Abs(_offsets[k].Item1))));
        for (int i = _offsets[k].Item1; i < img.Height; i++)
            for (int j = _offsets[k].Item2 > 0 ? _offsets[k].Item2 : 0;
                j < img.Width + (_offsets[k].Item2 > 0 ? 0 :
                _offsets[k].Item2); j++)
            {
                int newX = i - _offsets[k].Item1, newY = j –
                    _offsets[k].Item2;
                matrix[(img.Data[i, j, 0]), (img.Data[newX, newY, 0]), k]
                    += normalize ? increment : 1.0;
                matrix[(img.Data[newX, newY, 0]), (img.Data[i, j, 0]), k]
                    += normalize ? increment : 1.0;
            }
    }
    return matrix;
}
```

4.2.3. Hitung *Correlation*

Hitung *Correlation* adalah fungsi untuk menghitung nilai *correlation* dari GLCM yang dihasilkan fungsi *Create GLCM*. Fungsi ini memiliki 1 parameter yaitu *array* GLCM. Untuk mempersingkat perhitungan, fungsi ini dibantu dengan 4 fungsi lain. Fungsi ini menghasilkan nilai *correlation* sejumlah *offset* yang digunakan. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.3.

Segmen 4.3 *Source code* fungsi hitung *Correlation*

```
public static double[] Correlation(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    double[] mr = RowMean(glcm);
    double[] mc = ColMean(glcm);
    double[] sr = RowStdDev(glcm);
    double[] sc = ColStdDev(glcm);
    for (int k = 0; k < _offsetSize; k++)
    {
        double tot = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                tot += (i + 1 - mr[k]) * (j + 1 - mc[k]) * glcm[i, j, k];
        ans[k] = tot / (sr[k] * sc[k]);
    }
    return ans;
}
```

4.2.4. Hitung *Contrast*

Hitung *Contrast* adalah fungsi untuk menghitung nilai *contrast* dari GLCM yang dihasilkan fungsi *Create GLCM*. Fungsi ini memiliki 1 parameter yaitu *array* GLCM. Fungsi ini menghasilkan nilai *contrast* sebanyak *offset* yang digunakan. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.4.

Segmen 4.4 *Source code* fungsi hitung *Contrast*

```
public static double[] Contrast(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    for (int k = 0; k < _offsetSize; k++)
    {
        double tot = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                tot += (i - j) * (i - j) * glcm[i, j, k];
        ans[k] = tot;
    }
    return ans;
}
```

4.2.5. Hitung *Energy*

Hitung *Energy* adalah fungsi untuk menghitung nilai *energy* dari GLCM yang dihasilkan fungsi *Create GLCM*. Fungsi ini memiliki 1 parameter yaitu *array* GLCM. Fungsi ini menghasilkan nilai *energy* sejumlah *offset* yang digunakan. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.5.

Segmen 4.5 *Source code* fungsi hitung *Energy*

```
public static double[] Energy(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    for (int k = 0; k < _offsetSize; k++)
    {
        double tot = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                tot += glcm[i, j, k] * glcm[i, j, k];
        ans[k] = tot;
    }
    return ans;
}
```

4.2.6. Hitung *Homogeneity*

Hitung *Homogeneity* adalah fungsi untuk menghitung nilai *homogeneity* dari GLCM yang dihasilkan fungsi *Create GLCM*. Fungsi ini memiliki 1 parameter yaitu *array* GLCM. Fungsi ini menghasilkan nilai *homogeneity* sejumlah *offset* yang digunakan. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.6.

Segmen 4.6 *Source code* fungsi hitung *Homogeneity*

```
public static double[] Homogeneity(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    for (int k = 0; k < _offsetSize; k++)
    {
        double tot = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                tot += glcm[i, j, k] / (1 + Math.Abs(i - j));
        ans[k] = tot;
    }
    return ans;
}
```

4.2.7. Hitung *RowMean*

Hitung *RowMean* adalah fungsi untuk menghitung *weighted mean* dengan menggunakan indeks baris sebagai *weight*. Fungsi ini dibuat untuk mempersingkat perhitungan yang harus dilakukan pada fungsi Hitung

Correlation. Indeks yang digunakan sebagai *weight* dimulai dari 1, bukan dimulai dari 0. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.7.

Segmen 4.7 *Source code* fungsi hitung *RowMean*

```
private static double[] RowMean(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    for (int k = 0; k < _offsetSize; k++)
    {
        ans[k] = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                ans[k] += (i + 1) * glcm[i, j, k];
    }
    return ans;
}
```

4.2.8. Hitung *ColMean*

Hitung *ColMean* adalah fungsi untuk menghitung *weighted mean* dengan menggunakan indeks kolom sebagai *weight*. Secara keseluruhan fungsi ini hampir sama dengan fungsi Hitung *RowMean*, perbedaannya hanya pada *weight* yang digunakan. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.8.

Segmen 4.8 *Source code* fungsi hitung *ColMean*

```
private static double[] ColMean(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    for (int k = 0; k < _offsetSize; k++)
    {
        ans[k] = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                ans[k] += (j + 1) * glcm[i, j, k];
    }
    return ans;
}
```

4.2.9. Hitung *RowStdDev*

Hitung RowStdDev adalah fungsi untuk menghitung standar deviasi dengan menggunakan indeks baris sebagai *weight*. Sama dengan hitung *RowMean* dan hitung *ColMean*, fungsi ini hanya berguna untuk membantu fungsi Hitung *Correlation*, sehingga hak aksesnya adalah *private*. Hasil dari fungsi ini adalah standar deviasi untuk masing-masing GLCM. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.9.

Segmen 4.9 *Source code* fungsi hitung *RowStdDev*

```
private static double[] RowStdDev(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    double[] mean = RowMean(glcm);
    for (int k = 0; k < _offsetSize; k++)
    {
        ans[k] = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                ans[k] += (i + 1 - mean[k]) *
                          (i + 1 - mean[k]) *
                          glcm[i, j, k];
        ans[k] = Math.Sqrt(ans[k]);
    }
    return ans;
}
```

4.2.10. Hitung *ColStdDev*

Hitung ColStdDev adalah fungsi untuk menghitung standar deviasi dengan menggunakan indeks kolom sebagai *weight*. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.10.

Segmen 4.10 *Source code* fungsi hitung *ColStdDev*

```
private static double[] ColStdDev(double[, ,] glcm)
{
    double[] ans = new double[_offsetSize];
    double[] mean = ColMean(glcm);
    for (int k = 0; k < _offsetSize; k++)
    {
        ans[k] = 0.0;
        for (int i = 0; i < glcm.GetLength(0); i++)
            for (int j = 0; j < glcm.GetLength(1); j++)
                ans[k] += (j + 1 - mean[k]) *
                          (j + 1 - mean[k]) *
                          glcm[i, j, k];
        ans[k] = Math.Sqrt(ans[k]);
    }
    return ans;
}
```

4.2.11. *Scaling*

Scaling adalah fungsi untuk mengubah rentang nilai dalam gambar. Fungsi ini perlu dilakukan sebagai tahap *preprocessing* sebelum gambar diolah menjadi GLCM. Fungsi ini memiliki 3 parameter, yaitu gambar yang mau diubah rentang nilainya dan batas atas & bawah yang baru. Hasil yang dikembalikan dari fungsi ini adalah gambar yang telah diubah rentang nilainya. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.11.

Segmen 4.11 *Source code* fungsi *Scaling*

```
public static Image<Gray, Byte> Scale(Image<Gray, Byte> img,
    byte newMin = 1, byte newMax = 8)
```

```
{
    Image<Gray, Byte> result = img.CopyBlank();
    double slope = (double)(newMax - newMin + 1.0) / 255.0;
    for (int i = 0; i < img.Height; i++)
        for (int j = 0; j < img.Width; j++)
        {
            result.Data[i, j, 0] =
                (Byte)Math.Floor(img.Data[i, j, 0] * slope);
            if (result.Data[i, j, 0] >= newMax)
                result.Data[i, j, 0]--;
        }
    return result;
}
```

4.2.12. **Hitung Fitur**

Hitung Fitur adalah prosedur yang dipanggil oleh program utama untuk mengolah data gambar dengan cara memanggil fungsi-fungsi pada *static class* GLCM. Input gambar tidak dilakukan pada prosedur ini, tapi dilakukan oleh prosedur lain yang minor dalam program. Prosedur ini mengolah data hingga memasukkan fitur hasilnya ke dalam tabel. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.12.

Segmen 4.12 *Source code* fungsi Hitung Fitur

```
private void buttonLearn_Click(object sender, EventArgs e)
{
    if (currImgName != "" && label5.Text != "")
    {
        // Pre-Process
        images.Add(currImgName);
        img1 = new Image<Gray, Byte>(currImg.ToBitmap());
        img1._EqualizeHist();
        img = GLCM.Scale(img1);
        // Hitung Fitur
        double[, ,] mat = GLCM.CreateGLCM(img);
        double[] cont = GLCM.Contrast(mat);
        double[] corr = GLCM.Correlation(mat);
        double[] ener = GLCM.Energy(mat);
        double[] homo = GLCM.Homogeneity(mat);
        // Masukkan data ke DataGridView
        string append = "";
        string[] row = new string[4 * cont.Length + 3];
        row[0] = images.Count.ToString();
        for (int i = 0; i < cont.Length; i++)
        {
            row[4 * i + 1] = cont[i].ToString("F4");
            append += cont[i].ToString("F4") + ";";
```

Segmen 4.12 *Source code* fungsi Hitung Fitur (lanjutan)

```
            append += cont[i].ToString("F4") + ";";
            row[4 * i + 2] = corr[i].ToString("F4");
            append += corr[i].ToString("F4") + ";";
            row[4 * i + 3] = ener[i].ToString("F4");
            append += ener[i].ToString("F4") + ";";
            row[4 * i + 4] = homo[i].ToString("F4");
```

```
            append += homo[i].ToString("F4") + ";";
        }
        row[4 * cont.Length + 1] = textBox1.Text;
        row[4 * cont.Length + 2] = textBox2.Text;
        append += textBox1.Text + ";" + textBox2.Text;
        dt.Rows.Add(row);
        dataGridView1.DataSource = dt;
        // Simpan ke file
        using (StreamWriter sw = File.AppendText("data.txt"))
        {
            sw.WriteLine(append);
        }
        currImg.Save("images/" + (imageNow + 1) + ".jpg");
        // Refresh form untuk input selanjutnya
        currImgName = "";
        textBox1.Text = "";
        textBox2.Text = "";
        pictureBox1.Image = null;
        pictureBox2.Image = currImg.ToBitmap();
        imageNow = images.Count - 1;
        label4.Text = "Saved Image " + (imageNow + 1).ToString();
        label5.Text = "";
    }
    else
        MessageBox.Show("Masukkan gambar dahulu menggunakan tombol
'Browse..'\nTuliskan jenis dari gambar batik yang dimasukkan");
}
```

4.3. Implementasi *Class FuzzySet* dan Sistem Pengenalan

Tabel 4.2 Daftar fungsi dan prosedur untuk sistem pengenalan

Segmen Program	Nama Fungsi / Prosedur	Keterangan	*Flowchart*
4.13	*Get Fuzzy Value*	Menghitung fuzzy value untuk suatu nilai	Flowchart 3.20
4.14	*ID3 Decision Tree*	Menentukan jenis dari sebuah gambar menggunakan data dari *learning dataset*	Flowchart 3.15
4.15	Pengenalan	Menjalankan proses pengenalan dan mencatat respon pengguna	-

Class FuzzySet adalah implementasi dari desain *class FuzzySet* pada subbab 3.2.2. Pada subbab ini dibahas mengenai fungsi-fungsi yang major pada class FuzzySet dan sistem pengenalan. Sistem pengenalan menggunakan data dari *learning dataset* untuk dapat memperkirakan jenis isen dari suatu data baru dari *testing dataset*. Daftar fungsi dan prosedur yang digunakan dalam sistem pengenalan dapat dilihat pada Tabel 4.2.

4.3.1. **Tentukan *Fuzzy Value***

Fungsi ini berguna untuk menentukan *fuzzy value* dari sebuah *fuzzy set*. Fungsi ini memiliki satu *parameter*, yaitu nilai yang dihitung *fuzzy value*-nya. Hasil yang dikembalikan dari fungsi ini adalah *fuzzy value* dari *parameter*. Cara kerja fungsi ini menggunakan *binary search* untuk menentukan titik pada *bezier curve* dengan nilai x sama dengan *parameter*. *Fuzzy value* dari nilai tersebut adalah nilai y dari titik tersebut. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.13.

Segmen 4.13 *Source code* fungsi Tentukan *Fuzzy Value*

```
public double GetFuzzyValue(double value)
{
    double ans = 0.0;
    for (int i = 1; i < _vertices.Count - 1; i++)
    {
        double x1 = (_vertices[i - 1].Item1 + _vertices[i].Item1) / 2,
            y1 = (_vertices[i - 1].Item2 + _vertices[i].Item2) / 2,
            x2 = (_vertices[i + 1].Item1 + _vertices[i].Item1) / 2,
            y2 = (_vertices[i + 1].Item2 + _vertices[i].Item2) / 2;
        if (value >= x1 && value < x2)
        {
            double lo = x1, hi = x2, mid = (lo + hi) / 2.0;
            while (hi - lo > 1e-4)
            {
                double x = (1.0 - mid) * (1.0 - mid) * x1 +
                           2 * (1.0 - mid) * mid * _vertices[i].Item1 +
                           mid * mid * x2;
                if (x > value)
                    hi = mid;
                else if (x < value)
                    lo = mid;
                mid = (lo + hi) / 2;
            }
            ans = (1.0 - mid) * (1.0 - mid) * y1 +
                  2 * (1.0 - mid) * mid * _vertices[i].Item2 +
                  mid * mid * y2;
        }
    }
    double x0 = (_vertices[1].Item1 + _vertices[0].Item1) / 2.0,
           y0 = (_vertices[1].Item2 + _vertices[0].Item2) / 2.0;
    if (value < x0 && value >= _vertices[0].Item1)
    {
        if (Math.Abs(x0 - _vertices[0].Item1) > 1e-4)
            ans = _vertices[0].Item2 +
                  (value - _vertices[0].Item1) *
                  (y0 - _vertices[0].Item2) /
                  (x0 - _vertices[0].Item1);
```

Segmen 4.13 *Source code* fungsi Tentukan *Fuzzy Value* (lanjutan)

```
        else
            ans = Math.Max(y0, _vertices[0].Item2);
    }
    double xn = (_vertices[_vertices.Count - 1].Item1 +
```

```
                _vertices[_vertices.Count - 2].Item1) / 2.0,
            yn = (_vertices[_vertices.Count - 1].Item2 +
                _vertices[_vertices.Count - 2].Item2) / 2.0;
    if (value >= xn && value <= _vertices[_vertices.Count - 1].Item1)
    {
        if (Math.Abs(_vertices[_vertices.Count - 1].Item1 - xn) > 1e-4)
            ans = yn +
                (value - xn) *
                (_vertices[_vertices.Count - 1].Item2 - yn) /
                (_vertices[_vertices.Count - 1].Item1 - xn);
        else
            ans = Math.Max(_vertices[_vertices.Count - 1].Item2, yn);
    }
    if (ans < 0.0)
        ans = 0.0;
    return ans;
}
```

4.3.2. *ID3 Decision Tree*

Fungsi ini berguna untuk menentukan perkiraan jenis dari data fitur yang dihasilkan GLCM. Fungsi ini bersifat rekursif; tiap langkah rekursi akan mengurangi jumlah data yang digunakan. Rekursi berakhir saat kedalaman tree sudah melebihi batas tertentu atau saat seluruh data pada tingkat ini memiliki jenis yang sama. Jika sudah melebihi batas tertentu fungsi akan mengembalikan jawaban yang menandakan bahwa pencarian dihentikan. Jika seluruh data pada tingkat ini memiliki jenis yang sama fungsi akan mengembalikan jenis tersebut. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.14.

Segmen 4.14 *Source code* fungsi *ID3 Decision Tree*

```
private string ID3(List<DataRow> rows, double[] values, int depth)
{
    double[] informationGain = new double[16];
    double[, ,] fuzzyValues = null;
    double informationNeed = 0.0;
    Tuple<string, int>[] listOfJenis;
    FuzzySet[][] antecedentFuzzySet = new FuzzySet[16][];
    List<Tuple<double, int>> listOfColumns = new List<Tuple<double,int>>();
    int numOfJenis = 1;
    rows = rows.OrderBy(row => row[17]).ToList();
    if (depth == maxDepth)
        return "Pruned";
    for (int i = 1; i < rows.Count; i++)
        if (rows[i]["Jenis"].ToString() != rows[i - 1]["Jenis"].ToString())
            numOfJenis++;
    listOfJenis = new Tuple<string, int>[numOfJenis];
    fuzzyValues = new double[16, numOfJenis, rows.Count];
```

Segmen 4.14 *Source code* fungsi *ID3 Decision Tree* (lanjutan)

```
for (int i = 1, count = 1, now = 0; i <= rows.Count; i++)
    {
        rows[i - 1]["JenisNum"] = now.ToString();
        if (i == rows.Count ||
```

```
            rows[i]["Jenis"].ToString() != rows[i - 1]["Jenis"].ToString())
        {
            listOfJenis[now++] = new Tuple<string, int>
                (rows[i - 1]["Jenis"].ToString(), count);
            count = 1;
        }
        else
            count++;
    }
    if (numOfJenis == 1)
        return listOfJenis[0].Item1;
    for (int i = 0; i < numOfJenis; i++)
        informationNeed += -((double)listOfJenis[i].Item2 / (rows.Count) *
                             Math.Log((double)listOfJenis[i].Item2 /
                             (rows.Count), 2.0));
    for (int kolom = 1; kolom < 17; kolom++)
    {
        rows = rows.OrderBy(row => row[kolom]).ToList();
        antecedentFuzzySet[kolom - 1] = new FuzzySet[numOfJenis];
        for (int i = 0; i < numOfJenis; i++)
            antecedentFuzzySet[kolom - 1][i] = new FuzzySet(kolom);
        for (int baris = 1, lo = 0, hi = 0; baris <= rows.Count; baris++)
            if (baris == rows.Count ||
                rows[baris]["Jenis"].ToString() !=
                rows[baris - 1]["Jenis"].ToString())
            {
                int jenisIndex = int.Parse(rows[baris - 1]["JenisNum"]
                    .ToString());
                if (antecedentFuzzySet[kolom - 1][jenisIndex]
                    .Vertices.Count > 0 &&
                    Math.Abs(antecedentFuzzySet[kolom - 1][jenisIndex]
                    .Vertices[antecedentFuzzySet[kolom - 1][jenisIndex]
                    .Vertices.Count - 1].Item1 -
                    double.Parse(rows[lo - 1][kolom].ToString())) > 1e-6)
                    antecedentFuzzySet[kolom - 1][jenisIndex]
                        .Vertices.Add(new Tuple<double, double>(
                        lo == 0 ? 0.0 :
                        double.Parse(rows[lo - 1][kolom].ToString()), 0));
                antecedentFuzzySet[kolom - 1][jenisIndex]
                    .Vertices.Add(new Tuple<double, double>(
                    lo == 0 ? 0.0 :
                    double.Parse(rows[lo][kolom].ToString()), 1));
                if (lo != hi)
                    antecedentFuzzySet[kolom - 1][jenisIndex]
                        .Vertices.Add(new Tuple<double, double>(
                        hi == rows.Count - 1 ? 1.0 :
                        double.Parse(rows[hi][kolom].ToString()), 1));
                antecedentFuzzySet[kolom - 1][jenisIndex]
                    .Vertices.Add(new Tuple<double, double>(
                    hi == rows.Count - 1 ? 1.0 :
                    double.Parse(rows[hi + 1][kolom].ToString()), 0));
                lo = baris;
                hi = baris;
            }
```

Segmen 4.14 *Source code* fungsi *ID3 Decision Tree* (lanjutan)

```
            else
                hi++;
        for (int i = 0; i < rows.Count; i++)
        {
```

```
            double sum = 0.0;
            for (int j = 0; j < numOfJenis; j++)
            {
                fuzzyValues[kolom - 1, j, i] =
                   antecedentFuzzySet[kolom - 1][j]
                   .GetFuzzyValue(double.Parse(rows[i][kolom].ToString()));
                sum += fuzzyValues[kolom - 1, j, i];
            }
            for (int j = 0; j < numOfJenis; j++)
                fuzzyValues[kolom - 1, j, i] /= sum;
        }
        double entropy = 0.0;
        for (int i = 0; i < numOfJenis; i++)
        {
            double I = 0.0, alpha = 0.0;
            for (int j = 0; j < rows.Count; j++)
                alpha += fuzzyValues[kolom - 1, i, j];
            for (int j = 0; j < numOfJenis; j++)
            {
                double alpha_jk = 0.0;
                for (int k = 0; k < rows.Count; k++)
                    alpha_jk += Math.Min(
                        fuzzyValues[kolom - 1, i, k],
                        int.Parse(rows[k]["JenisNum"].ToString()) == j ?
                            1.0 : 0.0);
                if (alpha_jk > 0.0)
                    I += -(alpha_jk / alpha *
                        Math.Log(alpha_jk / alpha, 2.0));
            }
            entropy += alpha / (rows.Count) * I;
        }
        informationGain[kolom - 1] = informationNeed - entropy;
        listOfColumns.Add(new Tuple<double, int>(
            informationGain[kolom - 1], kolom));
    }
    listOfColumns.Sort();
    listOfColumns.Reverse();
    rows = rows.OrderBy(row => row[listOfColumns[0].Item2]).ToList();
    double minEntropy = -1.0;
    int cuttingIndex = -1;
    for (int baris = 1; baris < rows.Count; baris++)
        if (rows[baris]["Jenis"].ToString() !=
            rows[baris - 1]["Jenis"].ToString())
        {
            double entropy = 0.0;
            for (int i = 0; i < numOfJenis; i++)
            {
                int intLeft = 0, intRight = 0;
                for (int j = 0; j < rows.Count; j++)
                    if (rows[j]["JenisNum"].ToString() == i.ToString() &&
                        j < baris)
                        intLeft++;
                    else if (rows[j]["JenisNum"].ToString() ==
                        i.ToString())
```

Segmen 4.14 *Source code* fungsi *ID3 Decision Tree* (lanjutan)

```
                        intRight++;
                entropy -= (double)intLeft * (intLeft == 0 ? 0.0 :
                    Math.Log((double)intLeft / baris, 2.0));
                entropy -= (double)intRight * (intRight == 0 ? 0.0 :
```

```
                    Math.Log((double)intRight / (rows.Count - baris), 2.0));
            }
            entropy /= rows.Count;
            if (Math.Abs(minEntropy + 1.0) < 1e-4 || minEntropy > entropy)
            {
                minEntropy = entropy;
                cuttingIndex = baris;
            }
        }
    List<DataRow> nextRows = new List<DataRow>();
    double cuttingValue = double.Parse(
        rows[cuttingIndex][listOfColumns[0].Item2].ToString());
    if (values[listOfColumns[0].Item2 - 1] < cuttingValue)
        for (int i = 0; i < cuttingIndex; i++)
            nextRows.Add(rows[i]);
    else
        for (int i = 0; i < rows.Count - cuttingIndex; i++)
            nextRows.Add(rows[i + cuttingIndex]);
    return ID3(nextRows, values, depth + 1);
}
```

4.3.3. **Pengenalan**

Prosedur ini berguna untuk menerima input gambar dan batas kedalaman *decision tree* dari pengguna, lalu memanggil fungsi ID3 Decision Tree untuk mendapatkan perkiraan jenis. Perkiraan jenis ini dikembalikan ke pengguna dan pengguna dapat memberi respon apakah hasil proses pengenalan tersebut tepat atau tidak tepat. Jika ternyata tidak tepat, pengguna diminta untuk memberikan respon berupa jenis yang benar. Respon pengguna ini akan dimasukkan ke dalam data yang akan digunakan untuk proses pengenalan selanjutnya. Segmen program untuk prosedur ini dapat dilihat pada Segmen 4.15.

Segmen 4.15 *Source code* prosedur Pengenalan

```
private void buttonTest_Click(object sender, EventArgs e)
{
    if (recogImg == null)
    {
        MessageBox.Show("Input gambar batik terlebih dahulu");
        return;
    }
    if (!int.TryParse(textBox3.Text, out maxDepth))
    {
        MessageBox.Show("Parsing input gagal");
        return;
    }
    if (!dt.Columns.Contains("JenisNum"))
    {
```

Segmen 4.15 *Source code* prosedur Pengenalan (lanjutan)

```
        dt.Columns.Add("JenisNum");
        dataGridView1.Sort(this.dataGridView1.Columns["Jenis"],
            ListSortDirection.Ascending);
        for (int i = 1, now = 0; i <= dataGridView1.RowCount - 1; i++)
```

```
        {
            dataGridView1.Rows[i - 1].Cells["JenisNum"].Value =
                now.ToString();
            if (i == dataGridView1.RowCount - 1 ||
                dataGridView1.Rows[i].Cells["Jenis"].Value.ToString() !=
                dataGridView1.Rows[i - 1].Cells["Jenis"].Value.ToString())
                now++;
        }
    }
    img1 = new Image<Gray, Byte>(recogImg.ToBitmap());
    img1._EqualizeHist();
    img = GLCM.Scale(img1);
    double[, ,] mat = GLCM.CreateGLCM(img);
    double[] cont = GLCM.Contrast(mat);
    double[] corr = GLCM.Correlation(mat);
    double[] ener = GLCM.Energy(mat);
    double[] homo = GLCM.Homogeneity(mat);
    double[] values = new double[16];
    for (int i = 0; i < 4; i++)
    {
        values[4 * i] = cont[i];
        values[4 * i + 1] = corr[i];
        values[4 * i + 2] = ener[i];
        values[4 * i + 3] = homo[i];
    }
    List<DataRow> rows = new List<DataRow>();
    foreach (DataRow row in dt.Rows)
        if (row[0].ToString() != "")
            rows.Add(row);
    string result = ID3(rows, values, 0);
    DialogResult dialogResult = MessageBox.Show("Hasil pengenalan: " +
        result + "\nApakah hasil ini sesuai?", "Hasil pengenalan",
        MessageBoxButtons.YesNo);
    if (dialogResult == DialogResult.No)
    {
        string correctResult = Prompt.ShowDialog("Apakah jenis sebenarnya
            dari gambar\nini?", "Hasil pengenalan");
        if (correctResult != "")
        {
            string append = "";
            string[] row = new string[4 * cont.Length + 3];
            row[0] = images.Count.ToString();
            for (int i = 0; i < cont.Length; i++)
            {
                row[4 * i + 1] = cont[i].ToString("F4");
                append += cont[i].ToString("F4") + ";";
                row[4 * i + 2] = corr[i].ToString("F4");
                append += corr[i].ToString("F4") + ";";
                row[4 * i + 3] = ener[i].ToString("F4");
                append += ener[i].ToString("F4") + ";";
                row[4 * i + 4] = homo[i].ToString("F4");
                append += homo[i].ToString("F4") + ";";
            }
            row[4 * cont.Length + 1] = textBox1.Text;
```

Segmen 4.15 *Source code* prosedur Pengenalan (lanjutan)

```
            row[4 * cont.Length + 2] = textBox2.Text;
            append += textBox1.Text + ";" + textBox2.Text;
            dt.Rows.Add(row);
            dataGridView1.DataSource = dt;
```

```
            using (StreamWriter sw = File.AppendText("data.txt"))
                sw.WriteLine(append);
            currImg.Save("images/" + (imageNow + 1) + ".jpg");
            MessageBox.Show("Data tersimpan.");
        }
    }
    label7.Text = "";
    pictureBox3.Image = null;
    recogImg = null;
}
```

4.4. Implementasi *Static Class Prompt*

Static Class Prompt adalah *static class* yang digunakan untuk membuat kotak dialog (*dialog box*) yang dapat menerima inputan berupa teks. Kotak dialog yang disediakan C# hanya dapat menerima respon berupa tombol saja, sehingga dirasa perlu untuk membuat kotak dialog yang dapat menerima respon berupa teks. Daftar fungsi dan prosedur yang digunakan dalam sistem ekstraksi fitur dapat dilihat pada Tabel 4.3.

Tabel 4.3 Daftar fungsi dan prosedur untuk implementasi *Prompt*

Segmen Program	Nama Fungsi / Prosedur	Keterangan	*Flowchart*
4.16	*Show Dialog*	Menampilkan kotak dialog yang menerima respon berupa teks	-

4.4.1. *Show Dialog*

Show Dialog adalah sebuah fungsi untuk menampilkan kotak dialog dengan tulisan dan judul tertentu & menerima respon berupa teks. Dalam aplikasi ini *Prompt* digunakan untuk menerima respon dari pengguna apabila hasil pengenalan yang diberikan berbeda dengan data dari pengguna. Fungsi ini memiliki 2 *parameter*, yaitu *string* yang ditampilkan di bagian tengah dan di bagian atas (judul) dari kotak dialog. Hasil dari fungsi ini adalah *string* yang diinputkan pengguna. Segmen program untuk fungsi ini dapat dilihat pada Segmen 4.16.

Segmen 4.16 *Source code* fungsi *Show Dialog*

```
public static string ShowDialog(string text, string caption)
{
    Form prompt = new Form()
    {
```

```
        Width = 300,
        Height = 150,
        FormBorderStyle = FormBorderStyle.FixedDialog,
        Text = caption,
        StartPosition = FormStartPosition.CenterScreen
    };
    Label textLabel = new Label() { Left = 50, Top = 20, Text = text };
    TextBox textBox = new TextBox() { Left = 50, Top = 50, Width = 200 };
    Button confirmation = new Button() { Text = "Ok", Left = 100,
        Width = 100, Top = 80, DialogResult = DialogResult.OK };
    confirmation.Click += (sender, e) => { prompt.Close(); };
    prompt.Controls.Add(textBox);
    prompt.Controls.Add(confirmation);
    prompt.Controls.Add(textLabel);
    prompt.AcceptButton = confirmation;

    return prompt.ShowDialog() == DialogResult.OK ? textBox.Text : "";
}
```