Praktikum Extraction Feature (Local Binary Pattern)

Hero Yudo Martono

Mei 2016

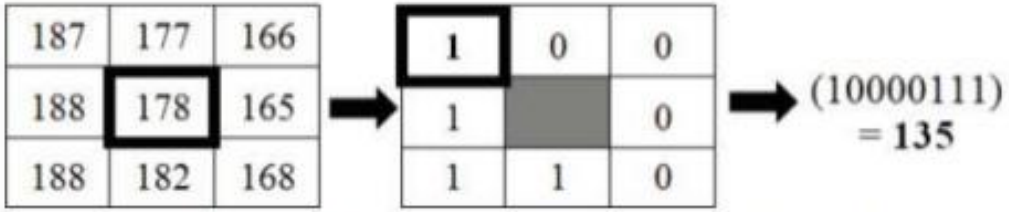
Local Binary Pattern

Local binary patterns

$$LBP(x_c, y_c) = \sum_{n=0}^{7} s(l_n - l_c)2^n$$

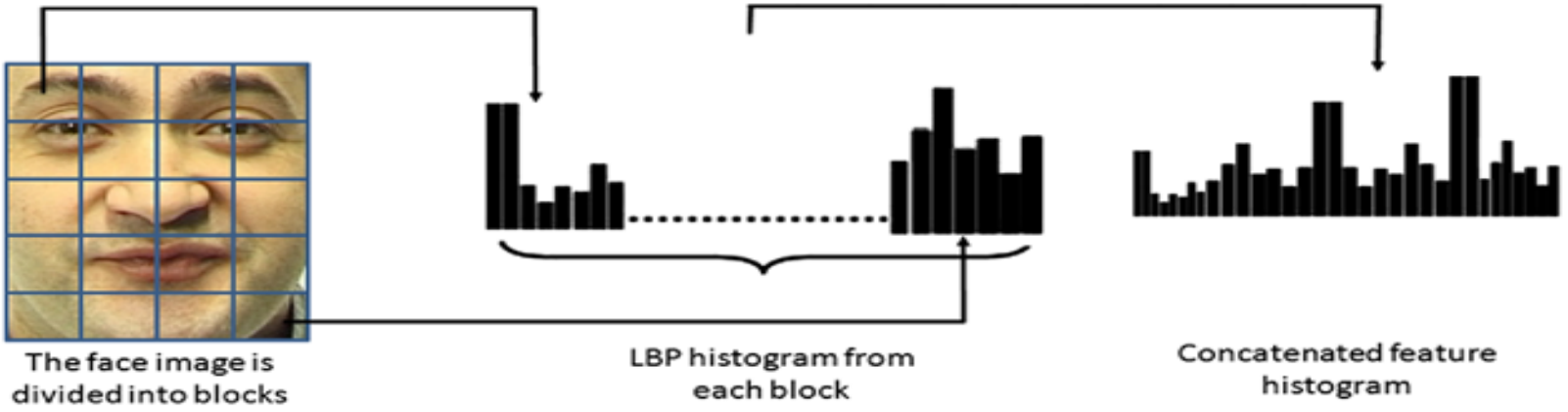
$$s(k) = \begin{cases} 1 & if \quad k \geq 0 \\ 0 & if \quad k < 0 \end{cases}$$

l_n : Corresponds to the central pixel value
l_c : The 8-neigbor pixels values



From a window with nine pixels of the image is defined soon after labeling where binary values larger than the center pixel receives 1 and 0 otherwise then the value in decimal, binary labels.

Local Binary Pattern



Input Image

5	4	2	2	1
3	5	8	1	3
2	5	4	1	2
4	3	7	2	7
1	4	4	2	6

Output LBP Image

		23		

1	1	0
1	X	0
0	1	0

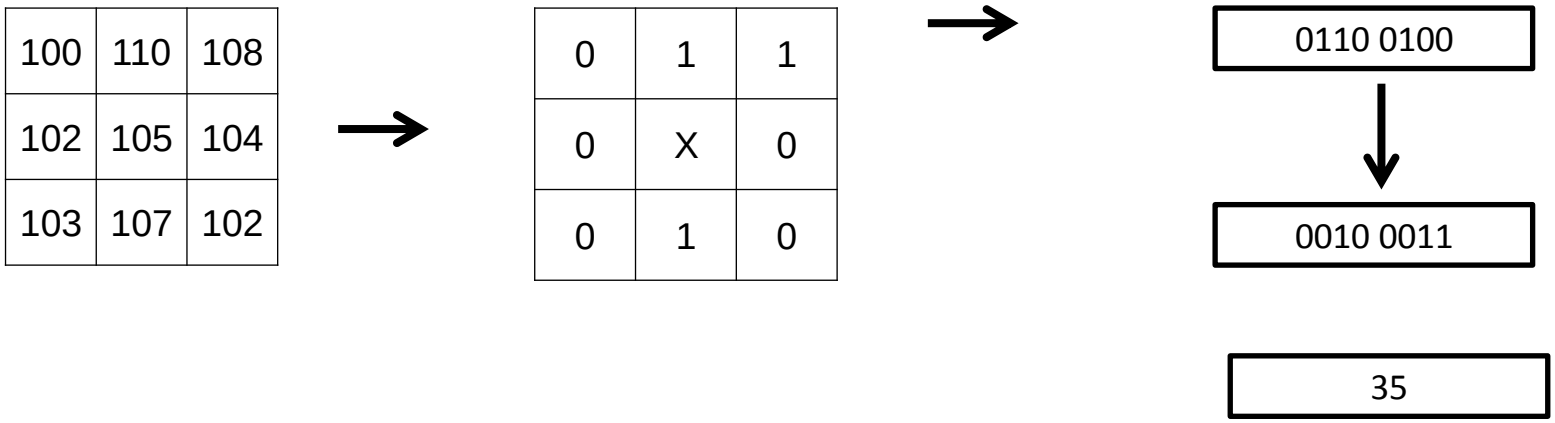
1100 0101

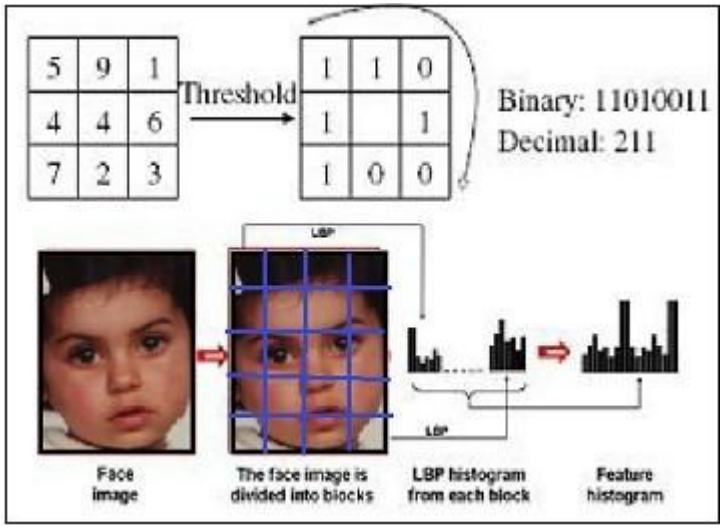
0001 0111

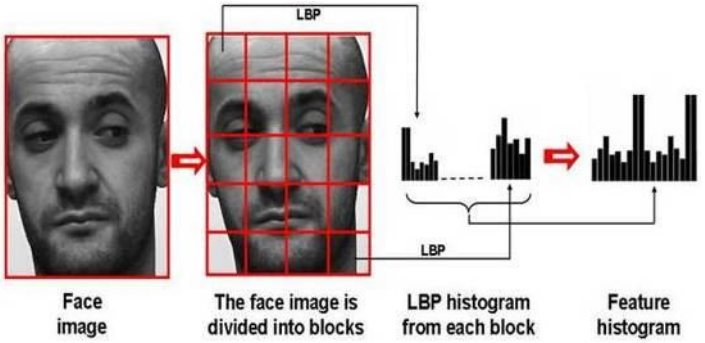
Question

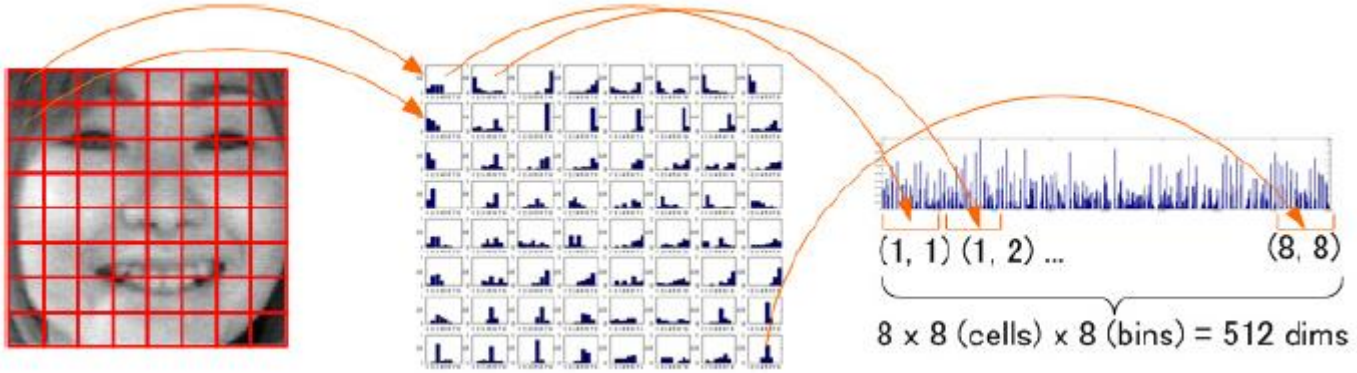
100	110	108
102	105	104
103	107	102

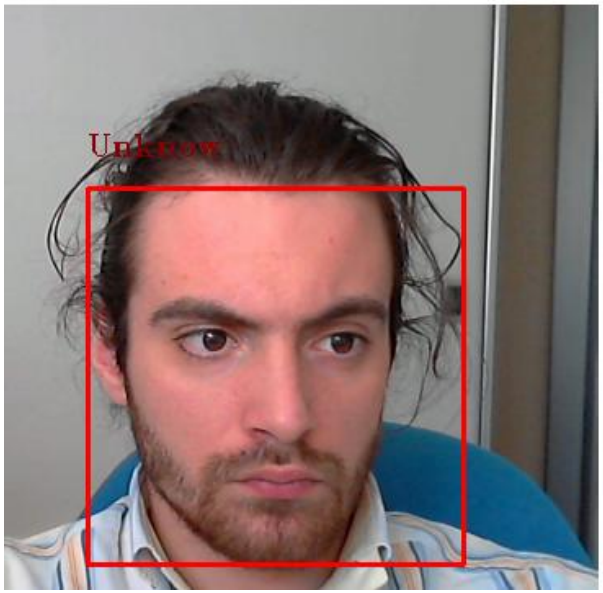
Answer


100 110 108
102 105 104
103 107 102
0 1 1
0 X 0
0 1 0
0110 0100
0010 0011
35


5 9 1
4 4 6
7 2 3
Threshold
1 1 0
1 1
1 0 0
Binary: 11010011
Decimal: 211
LBP
LBP
Face image
The face image is divided into blocks
LBP histogram from each block
Feature histogram

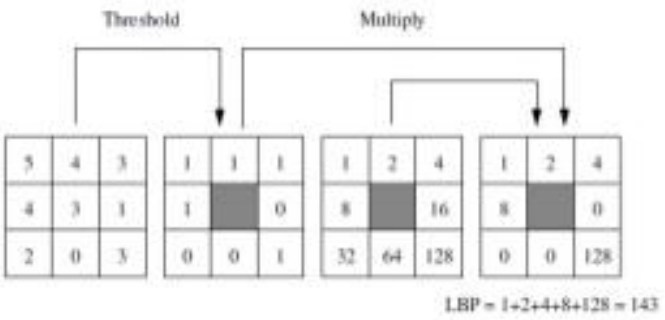

LBP
LBP
Face image
The face image is divided into blocks
LBP histogram from each block
Feature histogram

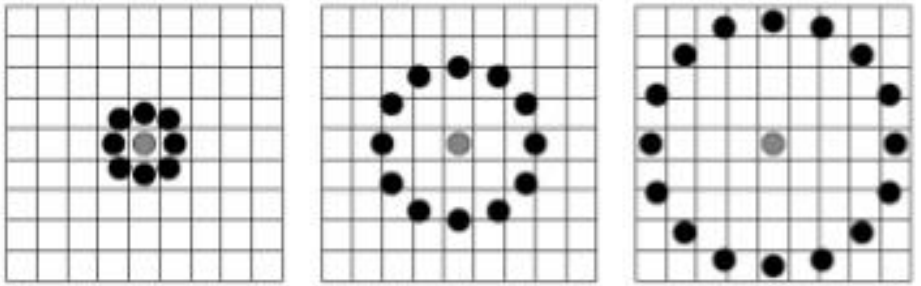

(1, 1) (1, 2) ...
(8, 8)
8 x 8 (cells) x 8 (bins) = 512 dims


Unknow

Segmentation I: Structure-emphasizing LBP filter

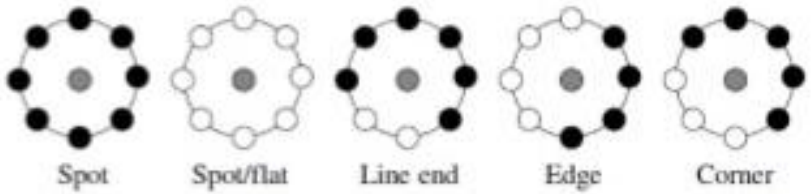
Local Binary Pattern:





Rotation invariant uniform LBPs:

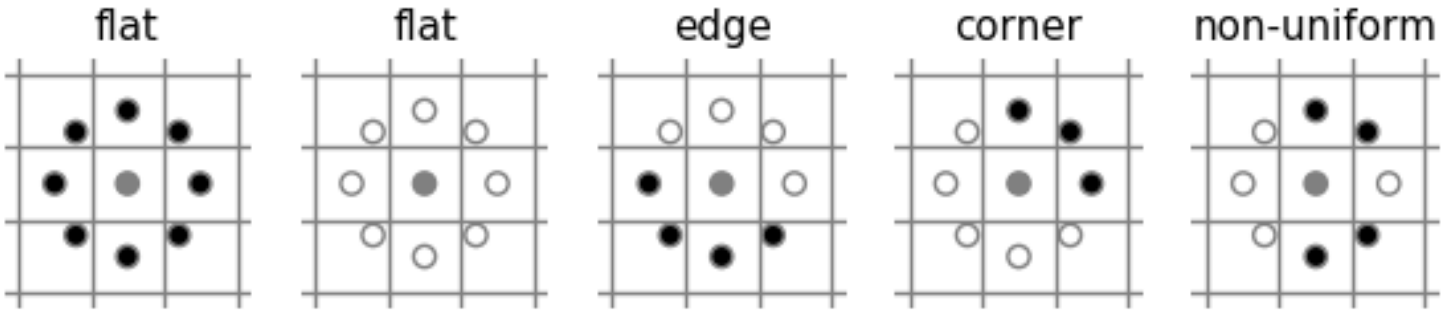
Texture primitives:



Im o Ojala et al., „Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 24, no. 7, July 2002.

Fraunhofer IOSB

Local Binary Pattern



Rotation variance

- We define formula U which value of an LBP pattern is defined as the number of spatial transitions (bitwise 0/1 changes) in that pattern.

$$U(LBP_{P,R}) = \left|s(g_{P-1} - g_c) - s(g_0 - g_c)\right| + \sum_{p=1}^{P-1} \left|s(g_p - g_c) - s(g_{p-1} - g_c)\right|$$

0	1	1
1		0
0	0	0

01110000

$U(LBP_{P,R}) = 2$

1	1	1
1		0
1	0	1

11110101

$U(LBP_{P,R}) = 4$

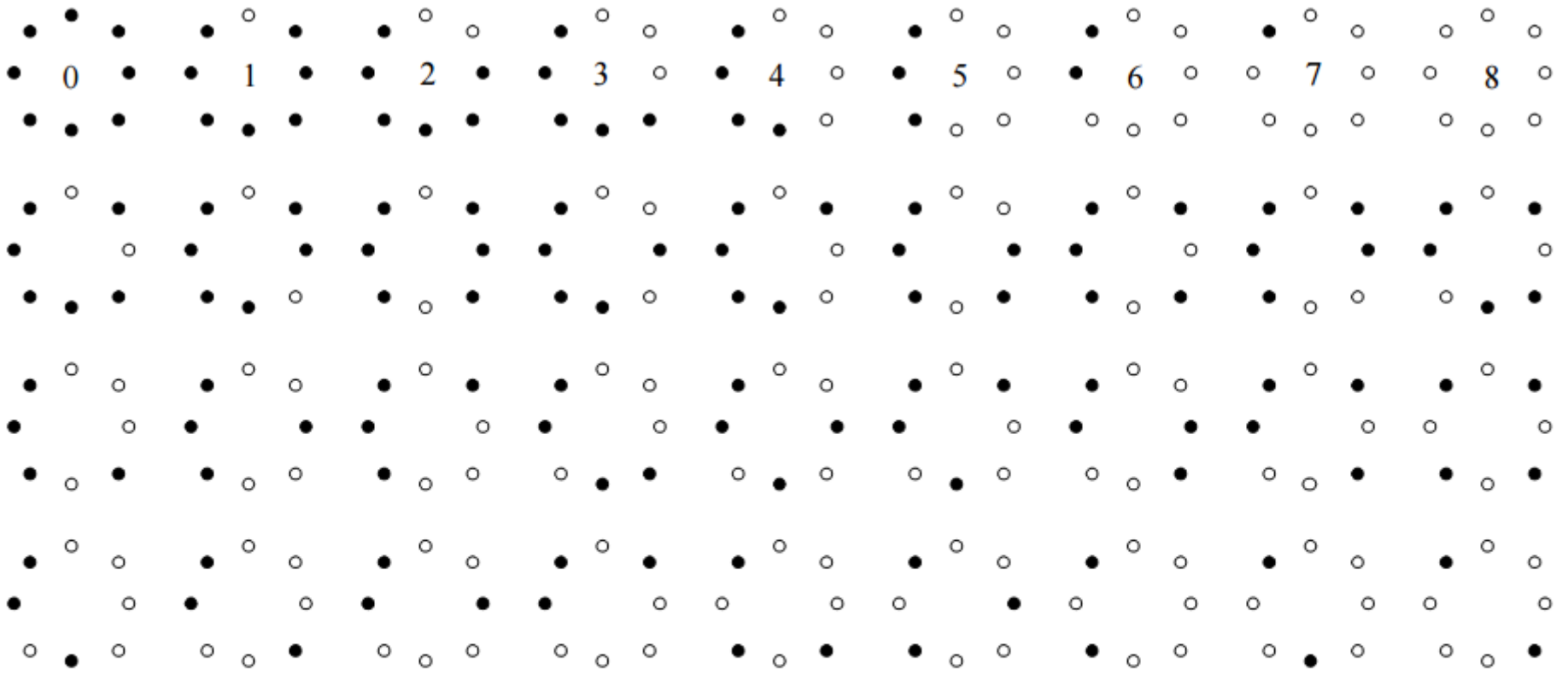


Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine 'uniform' patterns, and the numbers inside them correspond to their unique $LBP_{8,R}^{riu2}$ codes.

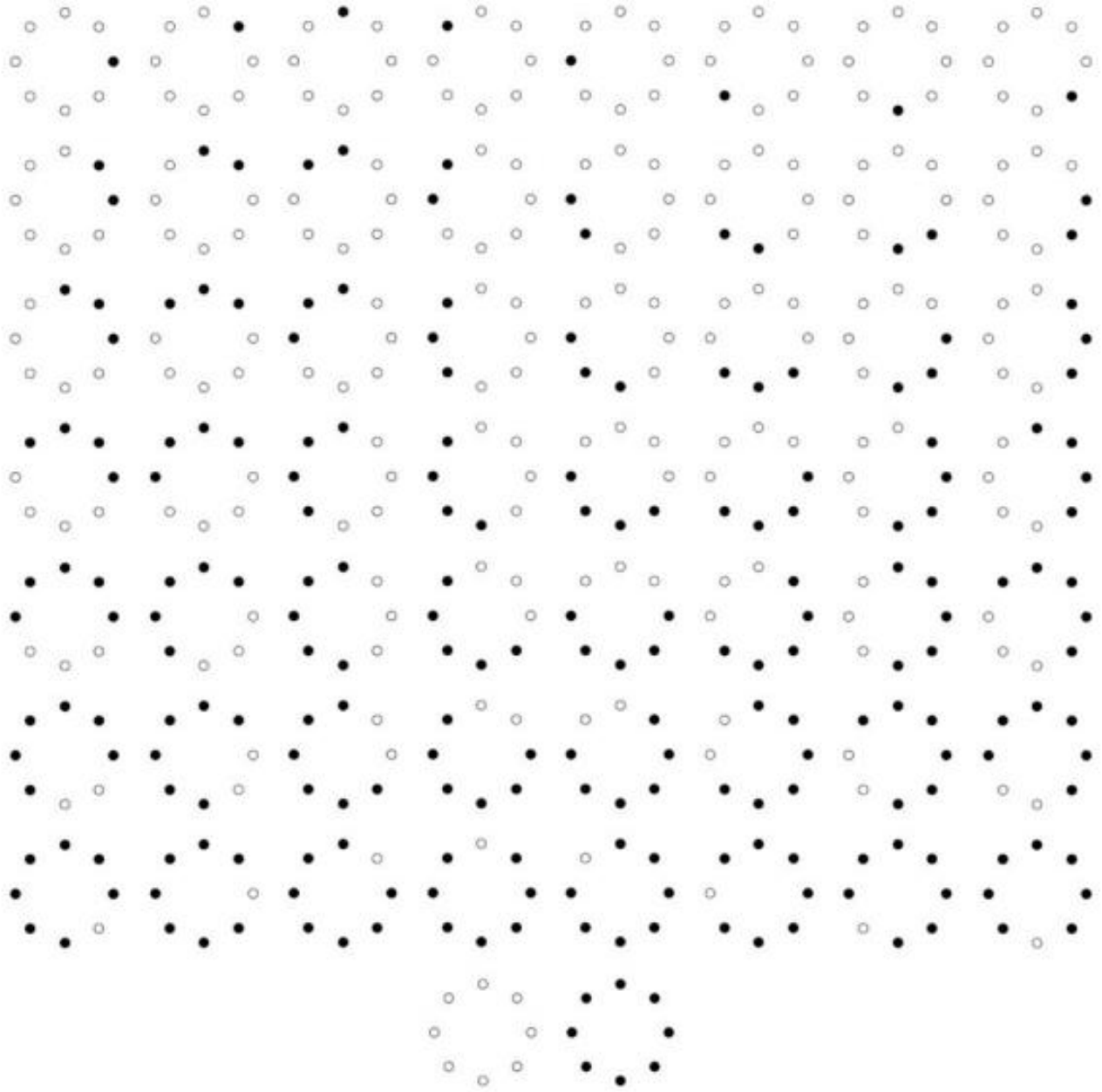


Fig. 3. Uniform LBP patterns when P=8. The black and white dots represent the bit values of 1 and 0 in the 8-bit output of the LBP operator.

Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns

Timo Ojala, Matti Pietikäinen and Topi Mäenpää
Machine Vision and Media Processing Unit
Infotech Oulu, University of Oulu
P.O.Box 4500, FIN - 90014 University of Oulu, Finland
{skidi, mkp, topiolli}@ee.oulu.fi
http://www.ee.oulu.fi/research/imag/texture

$$T \approx t(s(g_0 - g_c), s(g_1 - g_c), \ldots, s(g_{P-1} - g_c)) \tag{5}$$

where

$$s(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \tag{6}$$

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_0) 2^p \tag{7}$$

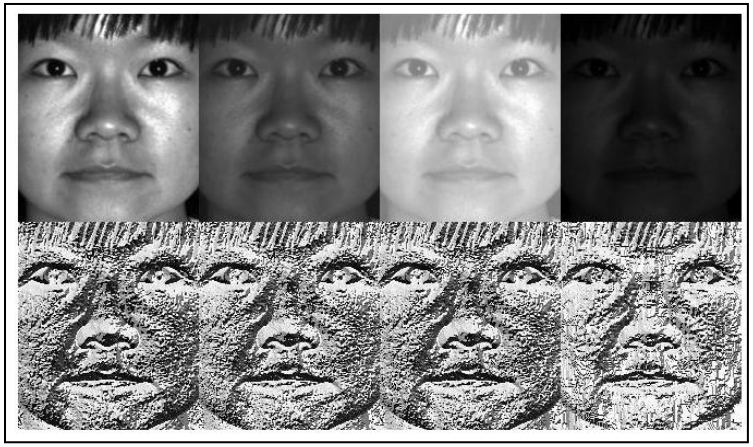
Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns

Timo Ojala, Matti Pietikäinen and Topi Mäenpää
Machine Vision and Media Processing Unit
Infotech Oulu, University of Oulu
P.O.Box 4500, FIN - 90014 University of Oulu, Finland
{skidi, mkp, topiolli}@ee.oulu.fi
http://www.ee.oulu.fi/research/imag/texture

$$LBP^{ri}_{P,R} = min\{ROR(LBP_{P,R}, i) \mid i = 0,1,...,P\text{-}1\} \quad (8)$$

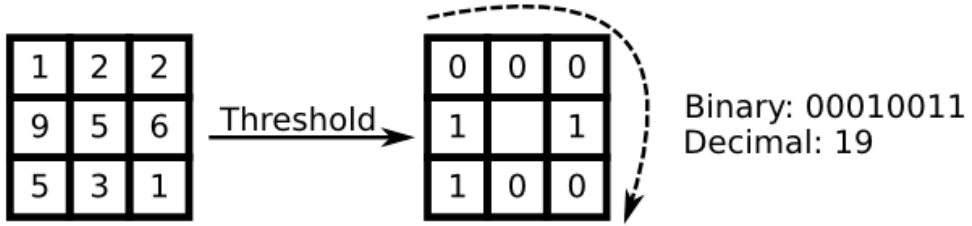
Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}{}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine 'uniform' patterns, and the numbers inside them correspond to their unique $LBP_{8,R}{}^{riu2}$ codes.

$$VAR_{P,R} = \frac{1}{P}\sum_{p=0}^{P-1}(g_p - \mu)^2 \quad \text{, where } \mu = \frac{1}{P}\sum_{p=0}^{P-1} g_p \quad (11)$$



Local Binary Patterns Histograms in OpenCV

```
/*
 * Copyright (c) 2011. Philipp Wagner <bytefish[at]gmx[dot]de>.
 * Released to public domain under terms of the BSD Simplified license.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *   * Neither the name of the organization nor the names of its contributors
 *     may be used to endorse or promote products derived from this software
 *     without specific prior written permission.
 *
 *   See <http://www.opensource.org/licenses/bsd-license>
 */

#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"

```



Algorithmic Description

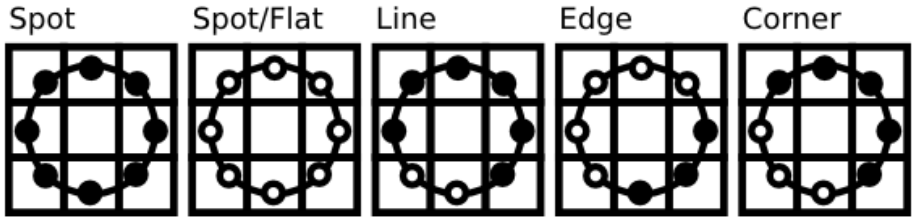
A more formal description of the LBP operator can be given as:

$$LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c)$$

, with (x_c, y_c) as central pixel with intensity i_c; and i_n being the intensity of the the neighbor pixel. s is the sign function defined as:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \qquad (1)$$

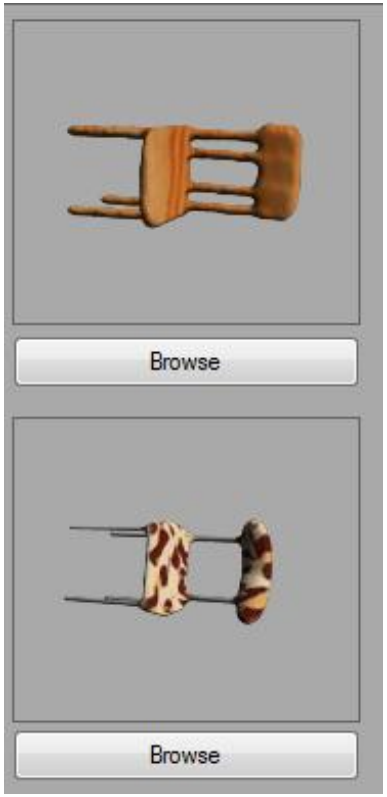
This description enables you to capture very fine grained details in images. In fact the authors were able to compete with state of the art results for texture cla a fixed neighborhood fails to encode details differing in scale. So the operator was extended to use a variable neighborhood in [AHP04]. The idea is to align which enables to capture the following neighborhoods:

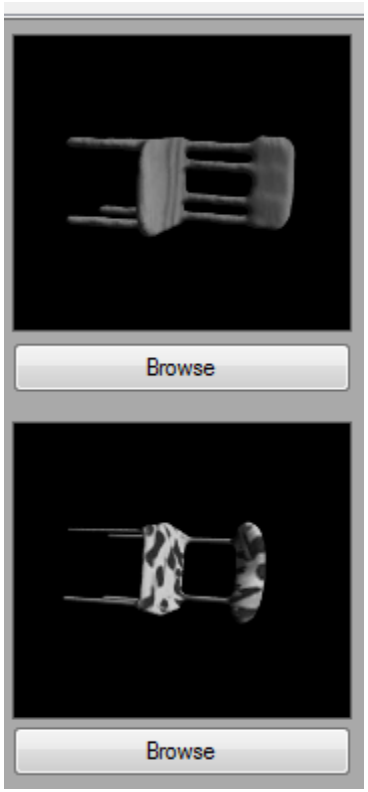


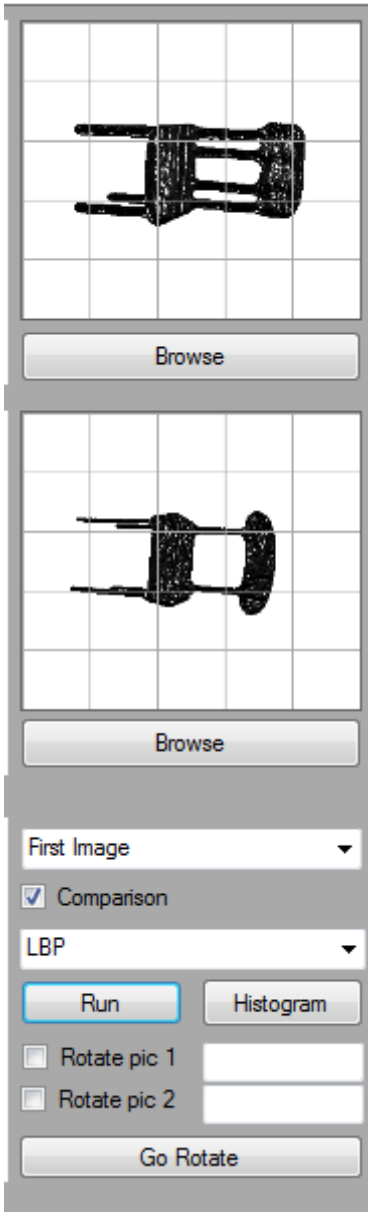
For a given Point (x_c,y_c) the position of the neighbor $(x_p, y_p), p \in P$ can be calculated by:

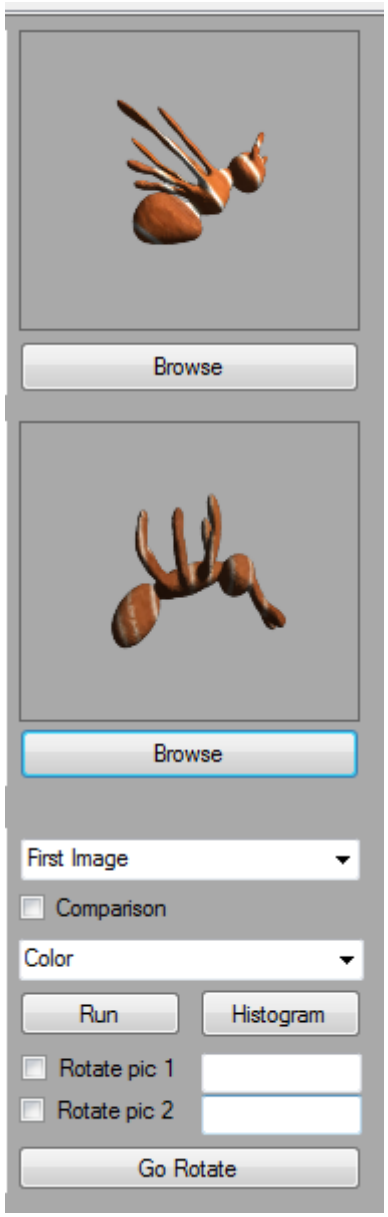
$$x_p = x_c + R\cos(\frac{2\pi p}{P})$$

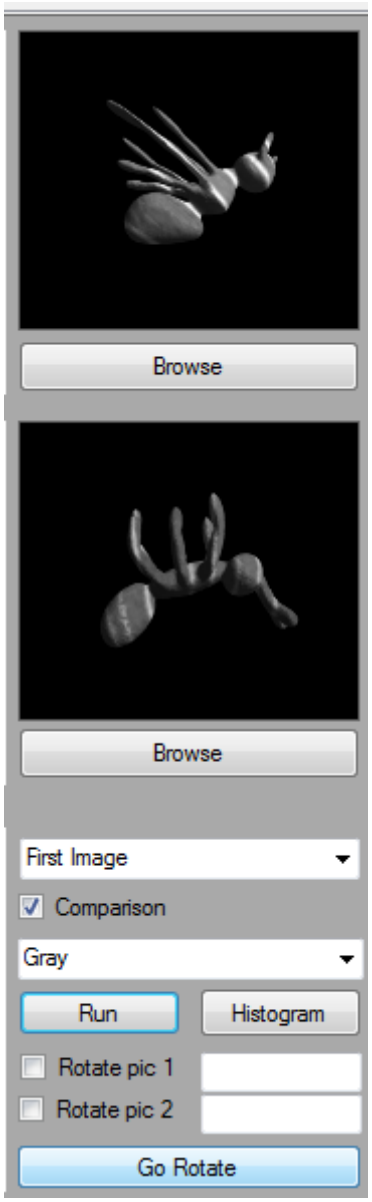
$$y_p = y_c - R\sin(\frac{2\pi p}{P})$$

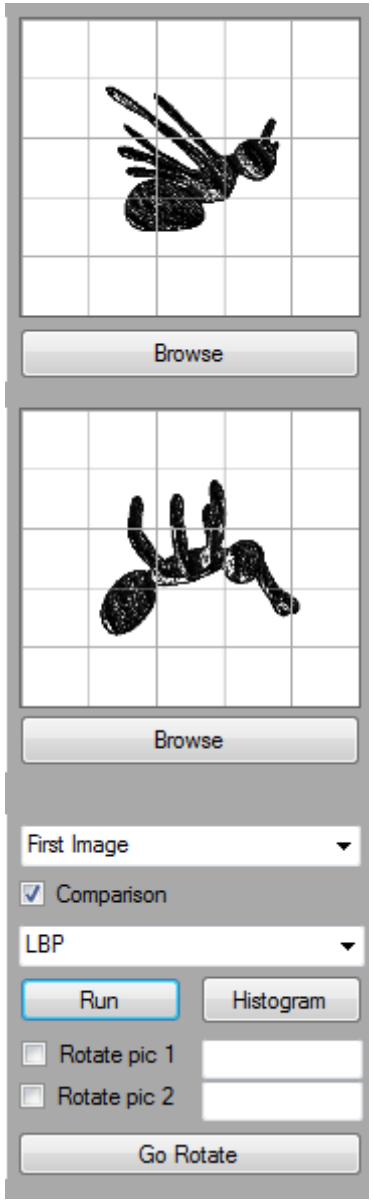

Browse
Browse


Browse
Browse


Browse
Browse
First Image
Comparison
LBP
Run
Histogram
Rotate pic 1
Rotate pic 2
Go Rotate


Browse
Browse
First Image
Comparison
Color
Run
Histogram
Rotate pic 1
Rotate pic 2
Go Rotate


Browse
Browse
First Image
Comparison
Gray
Run
Histogram
Rotate pic 1
Rotate pic 2
Go Rotate


Browse
Browse
First Image
Comparison
LBP
Run
Histogram
Rotate pic 1
Rotate pic 2
Go Rotate

Browse

Browse

First Image

Comparison

LBP

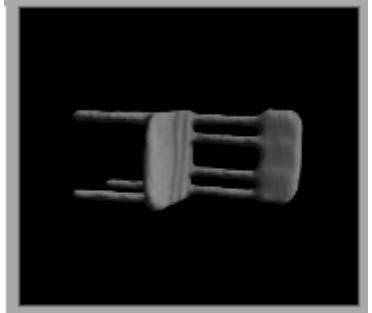
Run | Histogram

Rotate pic 1

Rotate pic 2

Go Rotate

Browse



Browse

First Image

Comparison

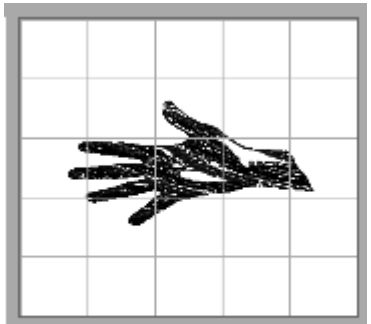
Gray

Run | Histogram

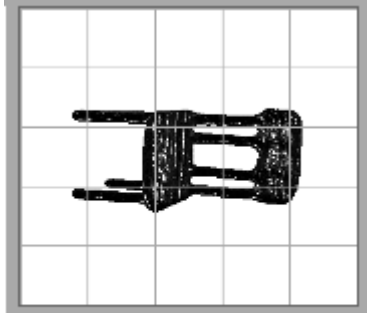
Rotate pic 1

Rotate pic 2

Go Rotate



Browse



Browse

First Image

Comparison

LBP

Run | Histogram

Rotate pic 1

Rotate pic 2

Go Rotate

Browse

Browse

Browse

Browse

Browse

Browse

All

Comparison

Sobel

Run

Histogram

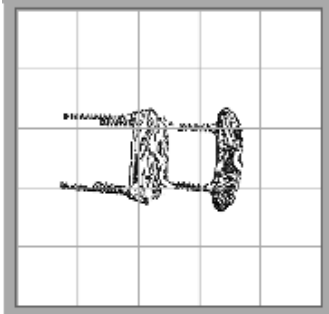
Rotate pic 1

Rotate pic 2

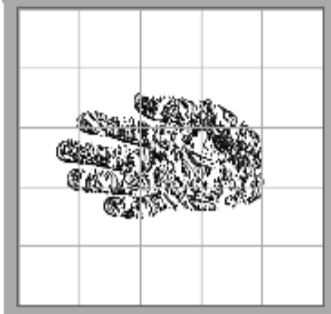
Rotate pic 3

Go Rotate

Browse



Browse



Browse

All

Comparison

LBP

Run

Histogram

Rotate pic 1

Rotate pic 2

Rotate pic 3

Go Rotate

Browse

Browse

Browse

Browse

Browse

Browse

All

Comparison

Sobel

Run

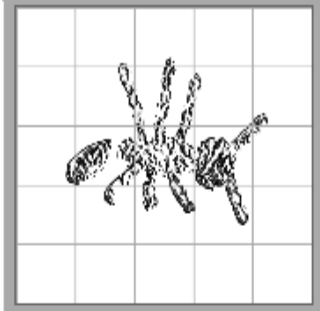
Histogram

Rotate pic 1 0

Rotate pic 2 0

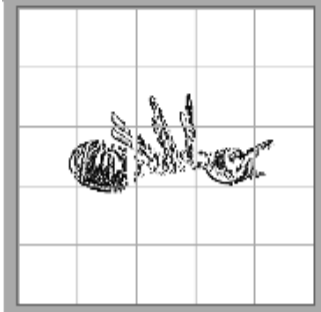
Rotate pic 3

Go Rotate



Browse

Browse



Browse

All

Comparison

LBP

Run

Histogram

Rotate pic 1 0

Rotate pic 2 0

Rotate pic 3

Go Rotate

```
//1=first LBP
//2=rotation invariant uniform , 9 features
//3=riu 36 unique
//4=texture primitives
//5=rotation variance
int num_patchLebar = 5;
int num_patchTinggi = 5;
int[, ,] binLBP1a = new int[num_patchLebar, num_patchTinggi, 256];
int[, ,] binLBP1b = new int[num_patchLebar, num_patchTinggi, 256];
int[, ,] binLBP1c = new int[num_patchLebar, num_patchTinggi, 256];
int[, ,] binLBP2a = new int[num_patchLebar, num_patchTinggi, 9];
int[, ,] binLBP2b = new int[num_patchLebar, num_patchTinggi, 9];
int[, ,] binLBP2c = new int[num_patchLebar, num_patchTinggi, 9];
int[, ,] binLBP3a = new int[num_patchLebar, num_patchTinggi, 36];
int[, ,] binLBP3b = new int[num_patchLebar, num_patchTinggi, 36];
int[, ,] binLBP3c = new int[num_patchLebar, num_patchTinggi, 36];
Class1 action = new Class1();
Bitmap bmp1 = (Bitmap)boxHis1.Image;
Bitmap bmp2 = (Bitmap)boxHis2.Image;
Bitmap bmp3 = (Bitmap)boxHis3.Image;
```

```
//caseSwitch4 = comboBox4.SelectedIndex;
switch (caseSwitch4)
   {
       case 0: Console.WriteLine("All");
               binLBP1a = action.getLBP(bmp1, binLBP1a, 1, num_patchLebar, num_patchTinggi);
               outputBitmap1 = action.getOutputBitmap();
               boxHis1.Image = outputBitmap1;
               binLBP1b = action.getLBP(bmp2, binLBP1b, 1, num_patchLebar, num_patchTinggi);
               outputBitmap2 = action.getOutputBitmap();
               boxHis2.Image = outputBitmap2;
               binLBP1c = action.getLBP(bmp3, binLBP1c, 1, num_patchLebar, num_patchTinggi);
               outputBitmap3 = action.getOutputBitmap();
               boxHis3.Image = outputBitmap3;
               break;
       case 1: Console.WriteLine("First Image");
               binLBP1a = action.getLBP(bmp1, binLBP1a, 1, num_patchLebar, num_patchTinggi);
               outputBitmap1 = action.getOutputBitmap();
               boxHis1.Image = outputBitmap1;
               break;
       case 2: Console.WriteLine("Second Image");
               binLBP1b = action.getLBP(bmp2, binLBP1b, 1, num_patchLebar, num_patchTinggi);
               outputBitmap2 = action.getOutputBitmap();
               boxHis2.Image = outputBitmap2;
               break;
       case 3: Console.WriteLine("Third Image");
               binLBP1c = action.getLBP(bmp3, binLBP1c, 1, num_patchLebar, num_patchTinggi);
               outputBitmap3 = action.getOutputBitmap();
               boxHis3.Image = outputBitmap3;
               break;
   }
```

```
public int[, ,] getLBP(Bitmap bmp, int[, ,] binLBP, int p, int lebar, int tinggi)
{
    Color pixelColor;
    outputBitmap = new Bitmap(bmp.Width, bmp.Height);
    int[] typeLBP = new int[2];
    for (int j = 0; j < tinggi; j++)
    {
        for (int i = 0; i < lebar; i++)
        {
            int batas_x1 = i*(int)bmp.Width / lebar +1;
            int batas_x2 = (i+1) *(int)bmp.Width / lebar -1;
            int batas_y1 = j*(int)bmp.Height / tinggi +1;
            int batas_y2 = (j+1) *(int)bmp.Height / tinggi -1;

            for (int y = batas_y1; y < batas_y2; y++)
            {
                for (int x = batas_x1; x < batas_x2; x++)
                {
                    pixelColor = bmp.GetPixel(x, y);
                    //mendapatkan nilai pola di tiap pixel
                    int value = getValueLBP(outputBitmap, bmp, x, y);

                    //memeriksa jenis type LBP
                    typeLBP = getTypeLBP(typeLBP, value, p);

                    //mendapatkan nilai histogram
                    if (typeLBP[0] == 1) binLBP[i, j, typeLBP[1]]++;

                }//end of x
            }//end of y

        }//end of i
    }//end of j
    Console.WriteLine("LBP Finished");
    return binLBP;
}
```

```
public int getMinShiftLBP(int[] polaLBP)...
public int getValueLBP(Bitmap outputBitmap, Bitmap bmp, int x, int y)
{
    int value = 0;
    int[] polaLBP = new int[8];

    for (int i = 0; i < 8; i++)
    {
        switch (i)
        {
            case 0: if (bmp.GetPixel(x - 1, y - 1).R >= bmp.GetPixel(x, y).R) polaLBP[7] = 1; else polaLBP[7] = 0; break;
            case 1: if (bmp.GetPixel(x , y - 1).R    >= bmp.GetPixel(x, y).R) polaLBP[6] = 1; else polaLBP[6] = 0; break;
            case 2: if (bmp.GetPixel(x + 1, y - 1).R >= bmp.GetPixel(x, y).R) polaLBP[5] = 1; else polaLBP[5] = 0; break;
            case 3: if (bmp.GetPixel(x + 1, y ).R    >= bmp.GetPixel(x, y).R) polaLBP[4] = 1; else polaLBP[4] = 0; break;
            case 4: if (bmp.GetPixel(x +1, y+1 ).R   >= bmp.GetPixel(x, y).R) polaLBP[3] = 1; else polaLBP[3] = 0; break;
            case 5: if (bmp.GetPixel(x , y + 1).R    >= bmp.GetPixel(x, y).R) polaLBP[2] = 1; else polaLBP[2] = 0; break;
            case 6: if (bmp.GetPixel(x - 1, y + 1).R >= bmp.GetPixel(x, y).R) polaLBP[1] = 1; else polaLBP[1] = 0; break;
            case 7: if (bmp.GetPixel(x - 1, y ).R    >= bmp.GetPixel(x, y).R) polaLBP[0] = 1; else polaLBP[0] = 0; break;
        }
    }

    value = getMinShiftLBP(polaLBP);
    outputBitmap.SetPixel(x, y, Color.FromArgb(value,value,value));
    return value;
}
```

```
public Bitmap getOutputBitmap()  {return outputBitmap;}

public int getMinShiftLBP(int[] polaLBP)
{
    int minimal = 255;
    int temp = 0;
    for (int i = 0; i < 8; i++)
    {
        temp = polaLBP[7];
        polaLBP[7] = polaLBP[6];
        polaLBP[6] = polaLBP[5];
        polaLBP[5] = polaLBP[4];
        polaLBP[4] = polaLBP[3];
        polaLBP[3] = polaLBP[2];
        polaLBP[2] = polaLBP[1];
        polaLBP[1] = polaLBP[0];
        polaLBP[0] = temp;

        int value = 0;
        for (int j = 0; j < 8; j++)
        {
            value += ((int)Math.Pow(2, j)) * polaLBP[j];
        }
        if (minimal > value) minimal = value;

    }
    return minimal;
}
```

```
public int[] getTypeLBP(int[] typeLBP, int value, int p){
    typeLBP[0] = 0;
    typeLBP[1] = 10;

    switch (p)
    {

        //LBP jenis first LBP dengan 255 bin
        case 1: typeLBP[0] = 1;
                typeLBP[1] = value;
                break;
        //LBP rotation invariant uniform , 9 feature
        case 2: typeLBP[0] = 1;
                if (value == 255)       { typeLBP[1] = 0; }
                else if (value == 127)  { typeLBP[1] = 1; }
                else if (value == 63)   { typeLBP[1] = 2; }
                else if (value == 31)   { typeLBP[1] = 3; }
                else if (value == 15)   { typeLBP[1] = 4; }
                else if (value == 7)    { typeLBP[1] = 5; }
                else if (value == 3)    { typeLBP[1] = 6; }
                else if (value == 1)    { typeLBP[1] = 7; }
                else if (value == 0)    { typeLBP[1] = 8; }
                else { typeLBP[0] = 0; typeLBP[1] = 0; }
                break;

        //36 unique rotation invariant uniform
        case 3:
                typeLBP[0] = 1;
                if (value == 255)       { typeLBP[1] = 0; }
                else if (value == 127)  { typeLBP[1] = 1; }
                else if (value == 63)   { typeLBP[1] = 2; }
                else if (value == 31)   { typeLBP[1] = 3; }
                else if (value == 15)   { typeLBP[1] = 4; }
                else if (value == 7)    { typeLBP[1] = 5; }
                else if (value == 3)    { typeLBP[1] = 6; }
                else if (value == 1)    { typeLBP[1] = 7; }
                else if (value == 0)    { typeLBP[1] = 8; }
```

```
                else if (value == 95)   { typeLBP[1] = 9;  }
                else if (value == 111)  { typeLBP[1] = 10; }
                else if (value == 119)  { typeLBP[1] = 11; }
                else if (value == 47)   { typeLBP[1] = 12; }
                else if (value == 79)   { typeLBP[1] = 13; }
                else if (value == 55)   { typeLBP[1] = 14; }
                else if (value == 87)   { typeLBP[1] = 15; }
                else if (value == 103)  { typeLBP[1] = 16; }
                else if (value == 91)   { typeLBP[1] = 17; }


                else if (value == 23) { typeLBP[1] = 18; }
                else if (value == 39) { typeLBP[1] = 19; }
                else if (value == 71) { typeLBP[1] = 20; }
                else if (value == 27) { typeLBP[1] = 21; }
                else if (value == 43) { typeLBP[1] = 22; }
                else if (value == 75) { typeLBP[1] = 23; }
                else if (value == 51) { typeLBP[1] = 24; }
                else if (value == 83) { typeLBP[1] = 25; }
                else if (value == 85) { typeLBP[1] = 26; }

                else if (value == 11) { typeLBP[1] = 27; }
                else if (value == 19) { typeLBP[1] = 28; }
                else if (value == 35) { typeLBP[1] = 29; }
                else if (value == 67) { typeLBP[1] = 30; }
                else if (value == 21) { typeLBP[1] = 31; }
                else if (value == 37) { typeLBP[1] = 32; }
                else if (value == 5)  { typeLBP[1] = 33; }
                else if (value == 9)  { typeLBP[1] = 34; }
                else if (value == 17) { typeLBP[1] = 35; }
                    |
                else { typeLBP[0] = 0; typeLBP[1] = 0; }
                break;
        }
    return typeLBP;
}
```

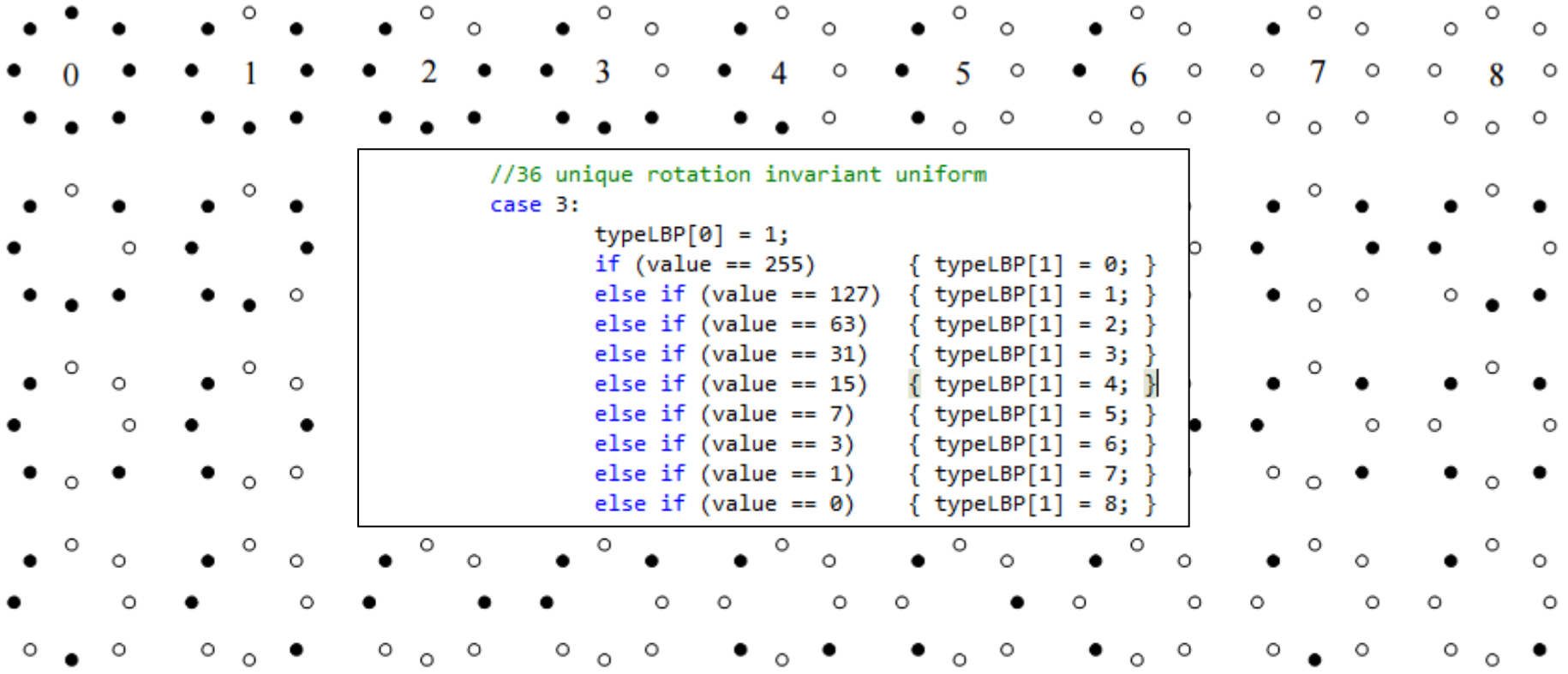
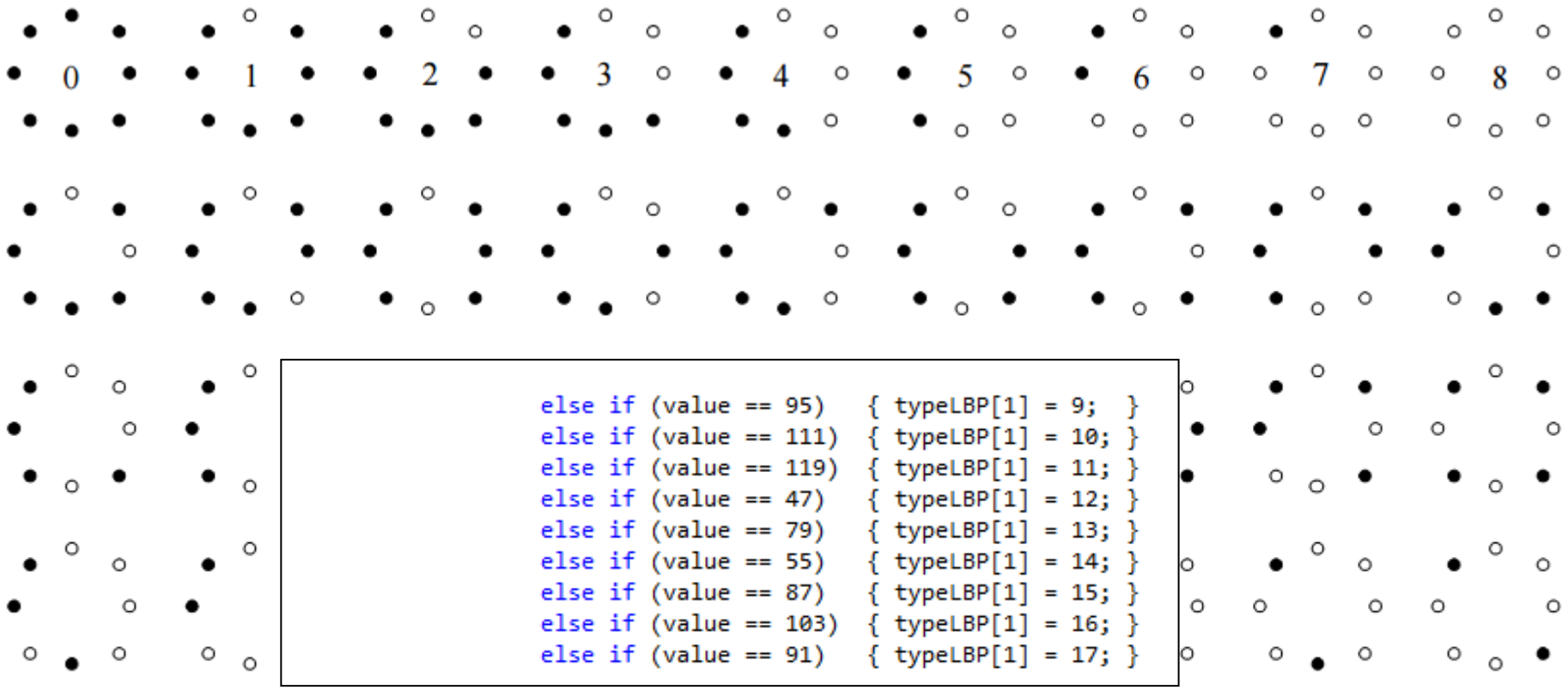
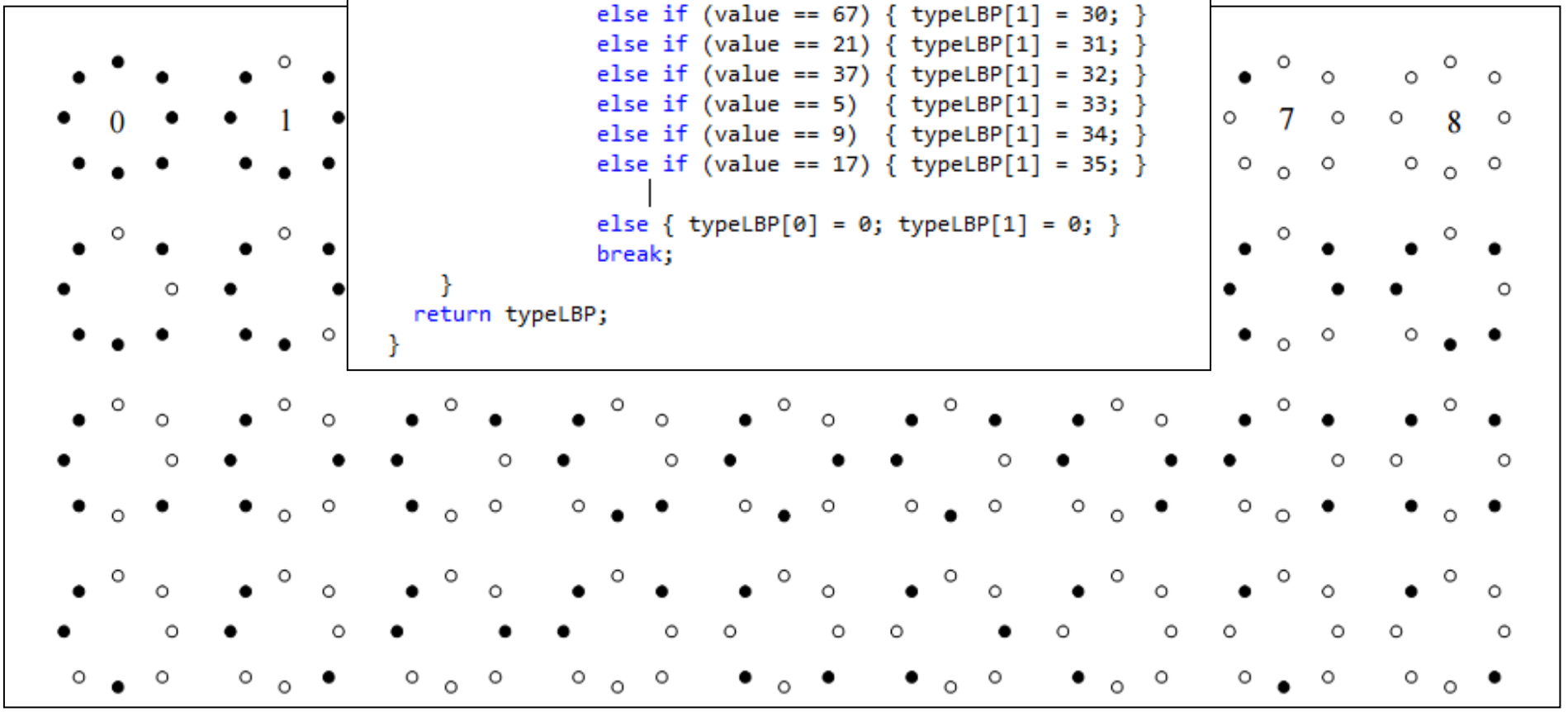


Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine 'uniform' patterns, and the numbers inside them correspond to their unique $LBP_{8,R}^{riu2}$ codes.




```
else if (value == 95)   { typeLBP[1] = 9;  }
else if (value == 111)  { typeLBP[1] = 10; }
else if (value == 119)  { typeLBP[1] = 11; }
else if (value == 47)   { typeLBP[1] = 12; }
else if (value == 79)   { typeLBP[1] = 13; }
else if (value == 55)   { typeLBP[1] = 14; }
else if (value == 87)   { typeLBP[1] = 15; }
else if (value == 103)  { typeLBP[1] = 16; }
else if (value == 91)   { typeLBP[1] = 17; }
```

Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine 'uniform' patterns, and the numbers inside them correspond to their unique $LBP_{8,R}^{riu2}$ codes.

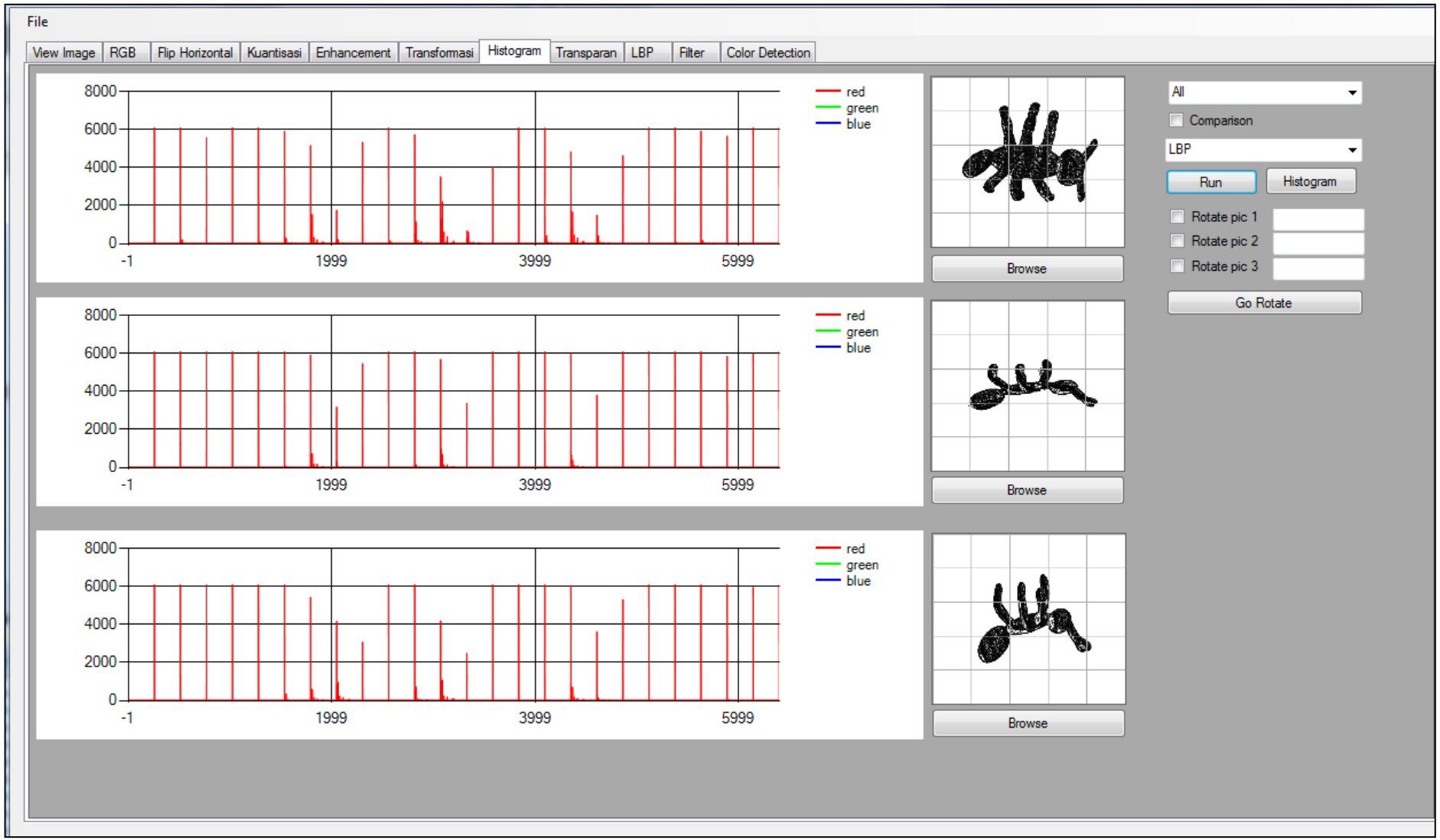



```
                else if (value == 23) { typeLBP[1] = 18; }
                else if (value == 39) { typeLBP[1] = 19; }
                else if (value == 71) { typeLBP[1] = 20; }
                else if (value == 27) { typeLBP[1] = 21; }
                else if (value == 43) { typeLBP[1] = 22; }
                else if (value == 75) { typeLBP[1] = 23; }
                else if (value == 51) { typeLBP[1] = 24; }
                else if (value == 83) { typeLBP[1] = 25; }
                else if (value == 85) { typeLBP[1] = 26; }

                else if (value == 11) { typeLBP[1] = 27; }
                else if (value == 19) { typeLBP[1] = 28; }
                else if (value == 35) { typeLBP[1] = 29; }
                else if (value == 67) { typeLBP[1] = 30; }
                else if (value == 21) { typeLBP[1] = 31; }
                else if (value == 37) { typeLBP[1] = 32; }
                else if (value == 5)  { typeLBP[1] = 33; }
                else if (value == 9)  { typeLBP[1] = 34; }
                else if (value == 17) { typeLBP[1] = 35; }

                else { typeLBP[0] = 0; typeLBP[1] = 0; }
                break;
            }
        return typeLBP;
    }
```

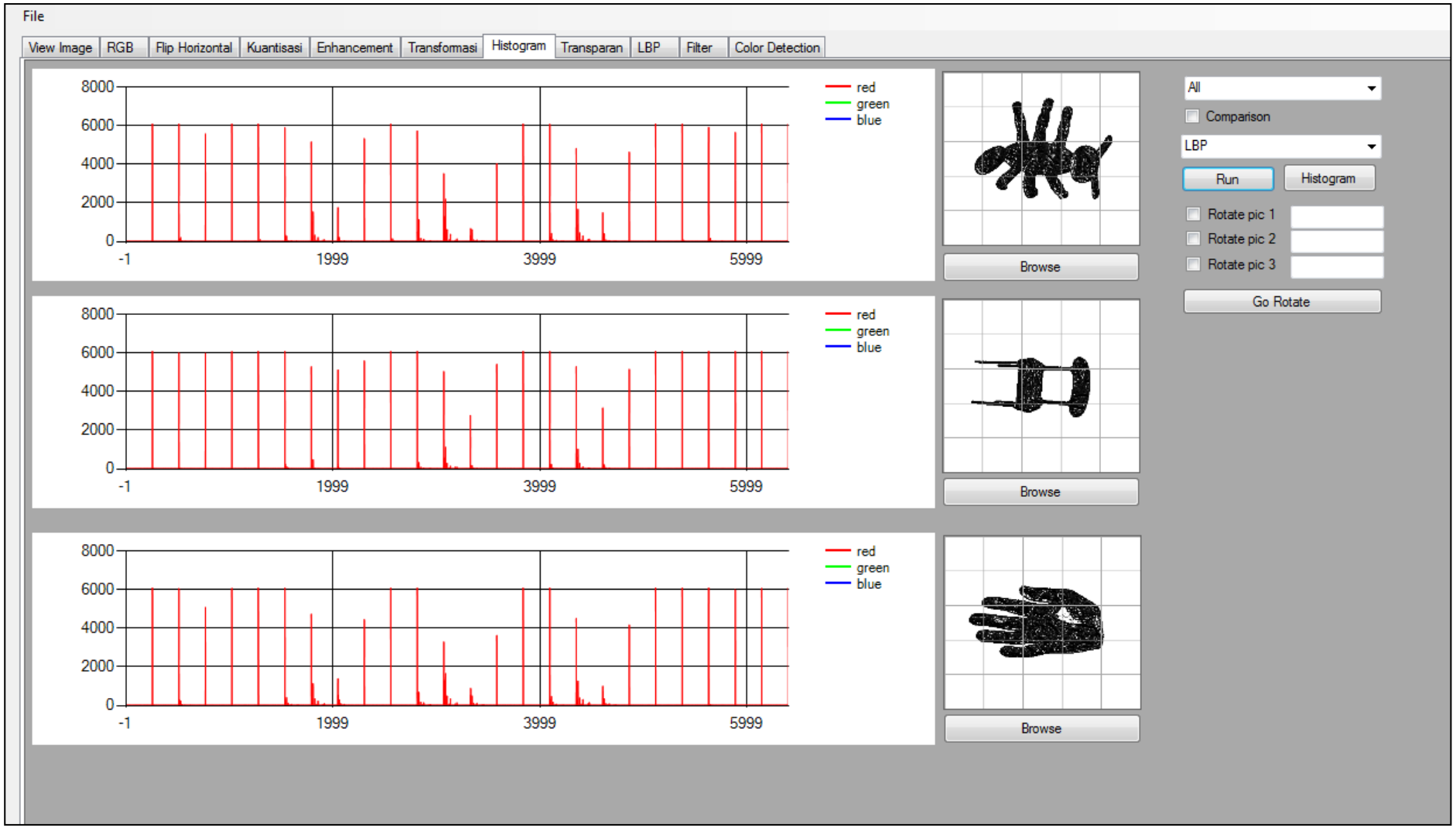
```
//caseSwitch4 = comboBox4.SelectedIndex;
switch (caseSwitch4)
   {
       case 0: Console.WriteLine("All");
               binLBP1a = action.getLBP(bmp1, binLBP1a, 1, num_patchLebar, num_patchTinggi);
               outputBitmap1 = action.getOutputBitmap();
               boxHis1.Image = outputBitmap1;
               binLBP1b = action.getLBP(bmp2, binLBP1b, 1, num_patchLebar, num_patchTinggi);
               outputBitmap2 = action.getOutputBitmap();
               boxHis2.Image = outputBitmap2;
               binLBP1c = action.getLBP(bmp3, binLBP1c, 1, num_patchLebar, num_patchTinggi);
               outputBitmap3 = action.getOutputBitmap();
               boxHis3.Image = outputBitmap3;
               setHistogram(binLBP1a, 1, num_patchTinggi, num_patchLebar);
               setHistogram(binLBP1b, 2, num_patchTinggi, num_patchLebar);
               setHistogram(binLBP1c, 3, num_patchTinggi, num_patchLebar);

               break;
       case 1: Console.WriteLine("First Image");
               binLBP1a = action.getLBP(bmp1, binLBP1a, 1, num_patchLebar, num_patchTinggi);
               outputBitmap1 = action.getOutputBitmap();
               boxHis1.Image = outputBitmap1;
               setHistogram(binLBP1a, 1, num_patchTinggi, num_patchLebar);
               break;
       case 2: Console.WriteLine("Second Image");
```

```
private void setHistogram(int[,,] histogram, int num, int numY, int numX)
{   switch (num)
    {
        case 1: chart1.Series["red"].Points.Clear(); chart2.Series["red"].Points.Clear(); chart3.Series["red"].Points.Clear(); break;
        case 2: chart1.Series["green"].Points.Clear(); chart2.Series["green"].Points.Clear(); chart3.Series["green"].Points.Clear(); break;
        case 3: chart1.Series["blue"].Points.Clear(); chart2.Series["blue"].Points.Clear(); chart3.Series["blue"].Points.Clear(); break;
    }

    for (int y = 1; y < numY-1; y++)
    {
        for (int x = 1; x < numX-1; x++)
        {
            for (int i = 0; i < 256; i++)
            {
                switch (num)
                {
                    case 1: this.chart1.Series["red"].Points.AddXY(((y * 5) + x) * 256 + i, histogram[y, x, i]); break;
                    case 2: this.chart2.Series["red"].Points.AddXY(((y * 5) + x) * 256 + i, histogram[y, x, i]); break;
                    case 3: this.chart3.Series["red"].Points.AddXY(((y * 5) + x) * 256 + i, histogram[y, x, i]); break;
                }
            }
        }
    }

}
```

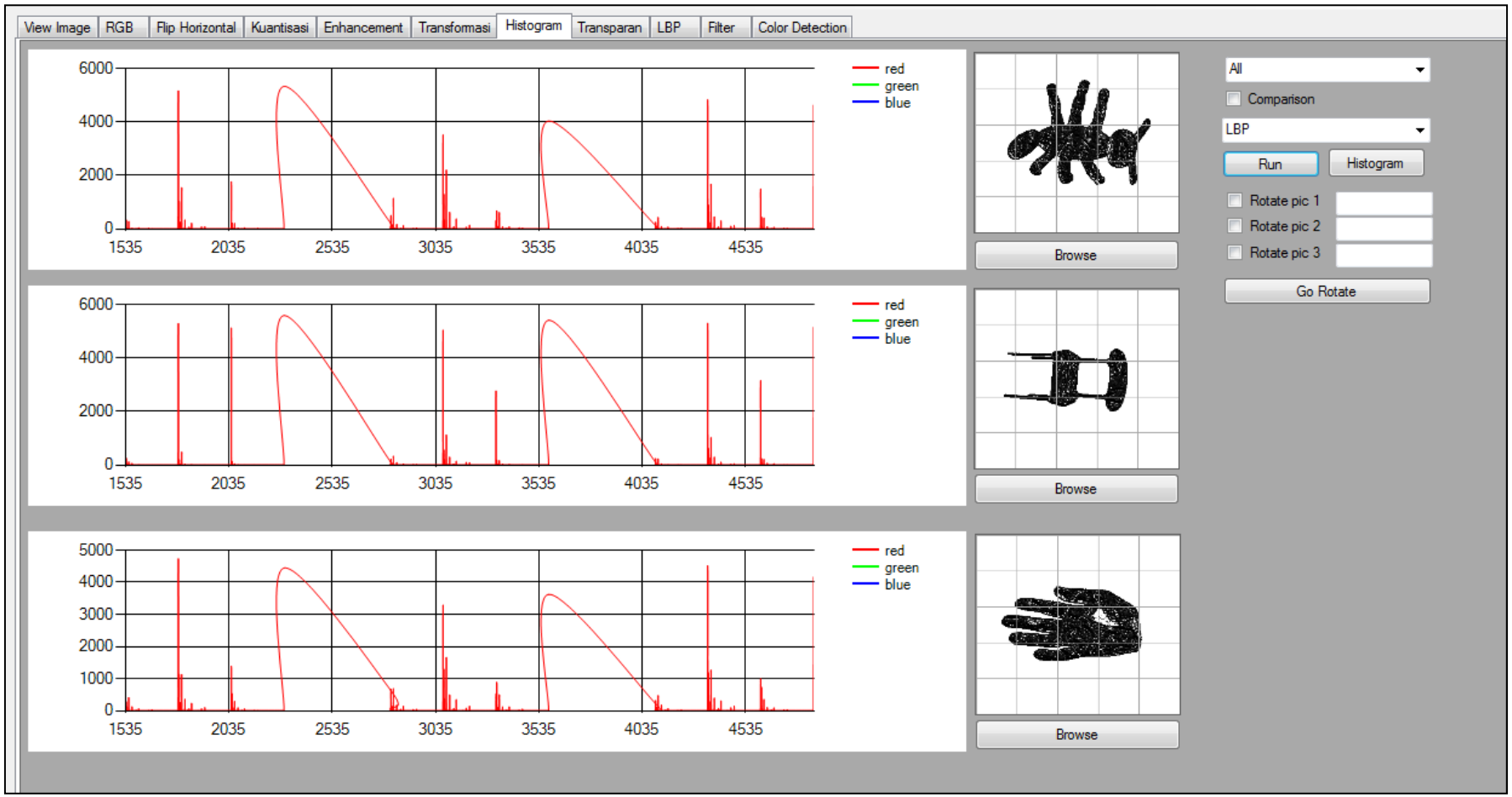
LBP type-1


File
View Image
RGB
Flip Horizontal
Kuantisasi
Enhancement
Transformasi
Histogram
Transparan
LBP
Filter
Color Detection
8000
6000
4000
2000
0
-1
1999
3999
5999
red
green
blue
Browse
All
Comparison
LBP
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate

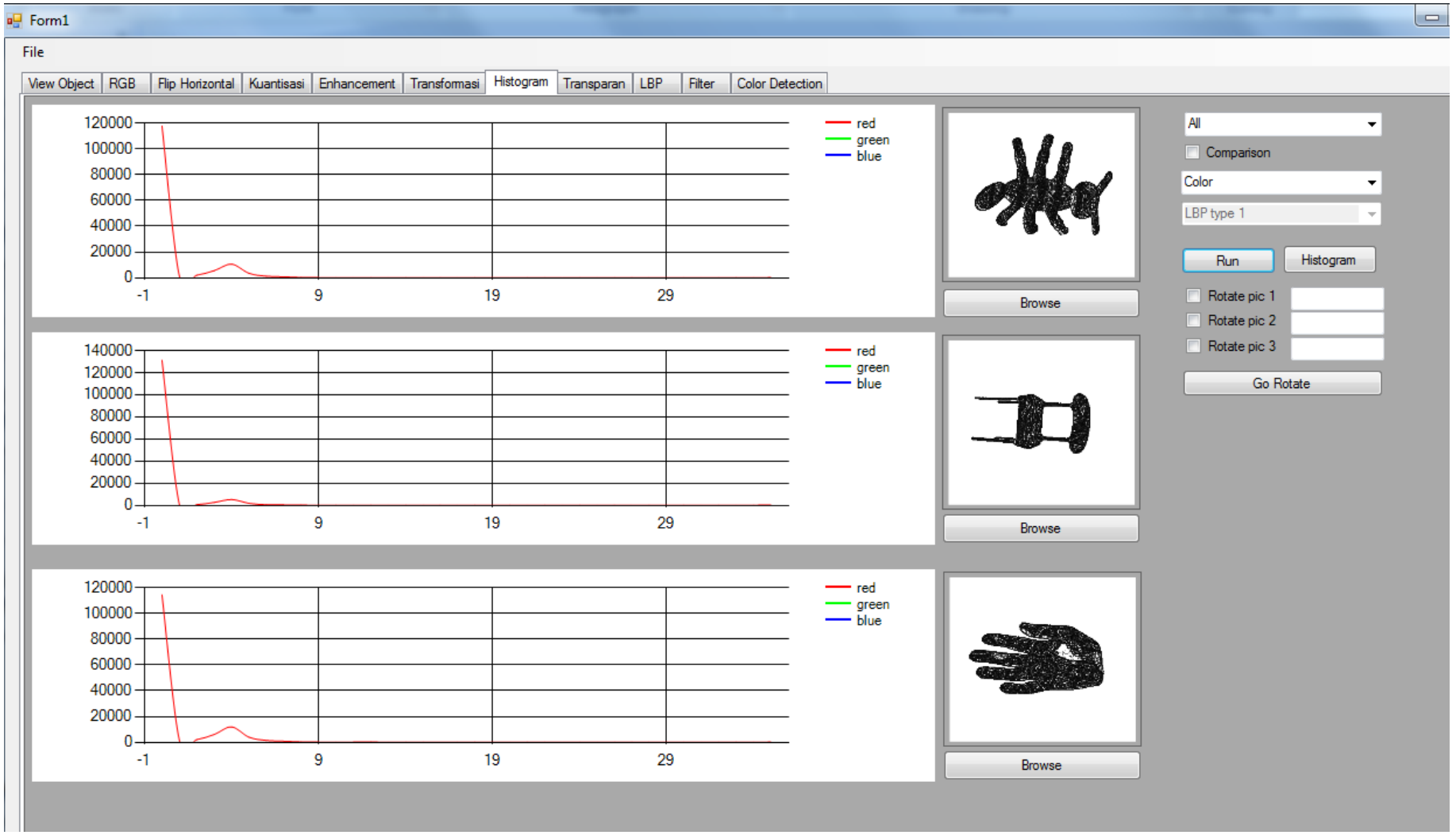
LBP type-1


File
View Image
RGB
Flip Horizontal
Kuantisasi
Enhancement
Transformasi
Histogram
Transparan
LBP
Filter
Color Detection
8000
6000
4000
2000
0
-1
1999
3999
5999
red
green
blue
Browse
All
Comparison
LBP
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate

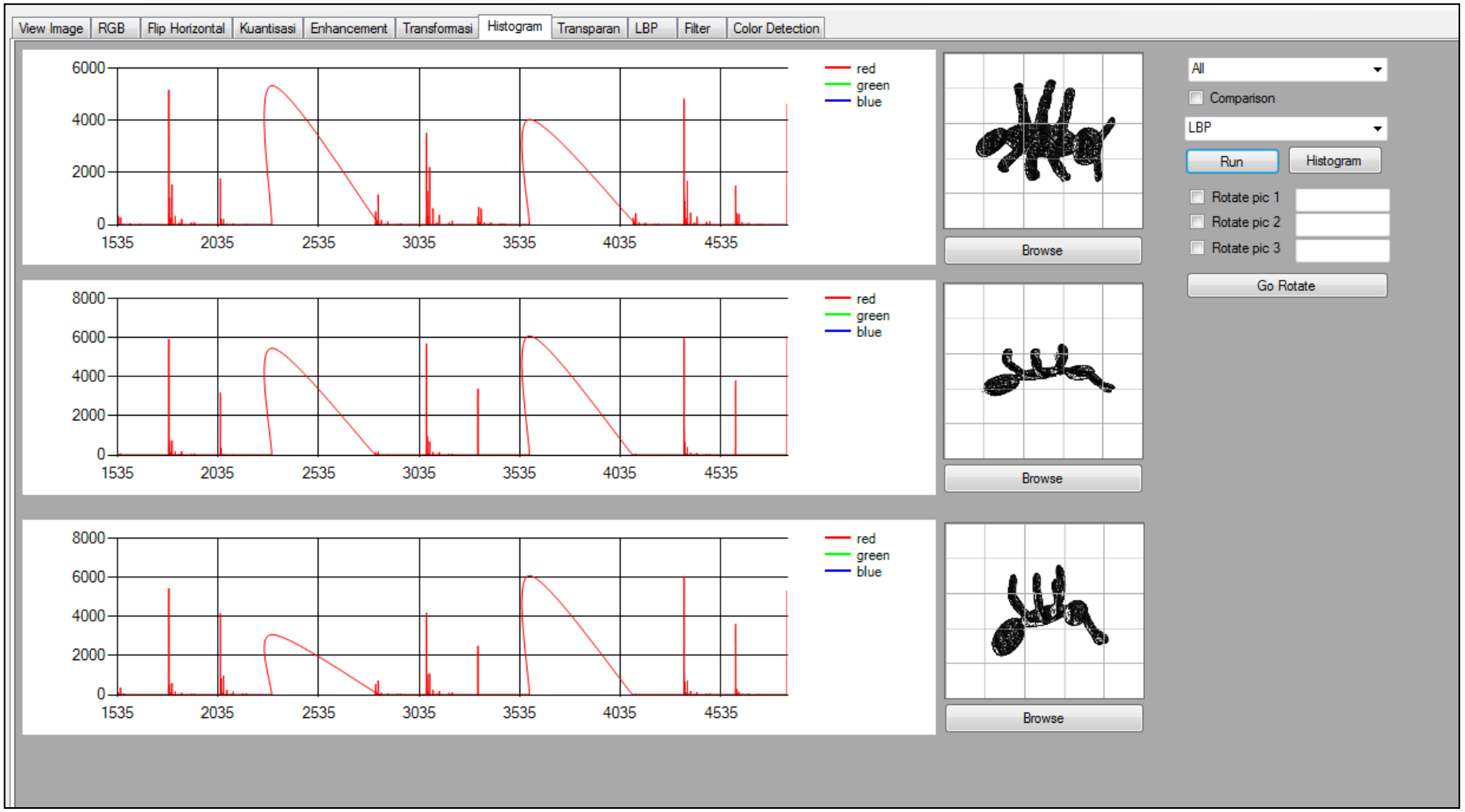
LBP type-1


View Image
RGB
Flip Horizontal
Kuantisasi
Enhancement
Transformasi
Histogram
Transparan
LBP
Filter
Color Detection
6000
4000
2000
0
1535
2035
2535
3035
3535
4035
4535
red
green
blue
Browse
6000
4000
2000
0
1535
2035
2535
3035
3535
4035
4535
red
green
blue
Browse
5000
4000
3000
2000
1000
0
1535
2035
2535
3035
3535
4035
4535
red
green
blue
Browse
All
Comparison
LBP
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate

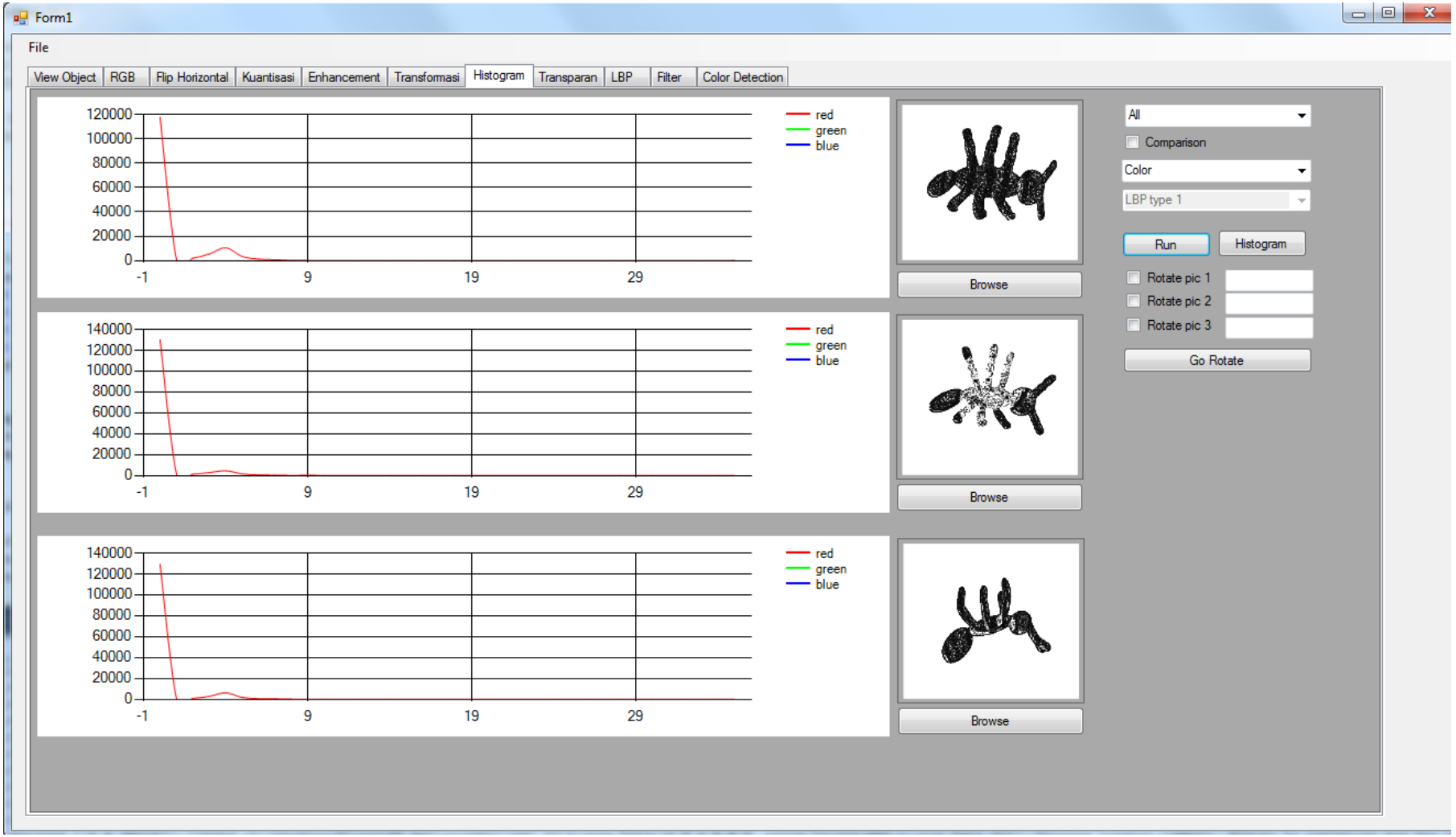
LBP type-2

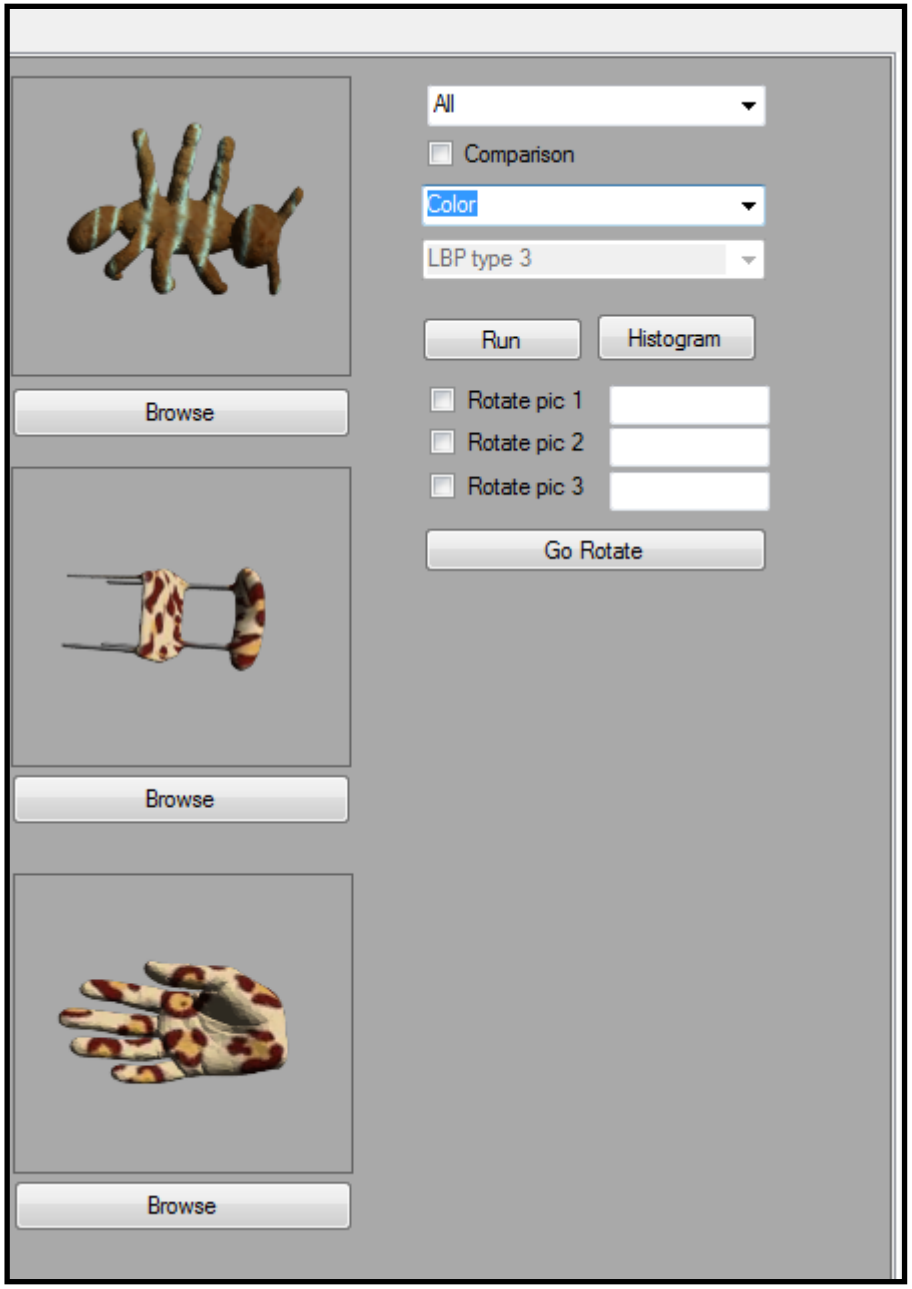

Form1
File
View Object
RGB
Flip Horizontal
Kuantisasi
Enhancement
Transformasi
Histogram
Transparan
LBP
Filter
Color Detection
120000
100000
80000
60000
40000
20000
0
-1
9
19
29
red
green
blue
Browse
140000
120000
100000
80000
60000
40000
20000
0
-1
9
19
29
red
green
blue
Browse
120000
100000
80000
60000
40000
20000
0
-1
9
19
29
red
green
blue
Browse
All
Comparison
Color
LBP type 1
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate

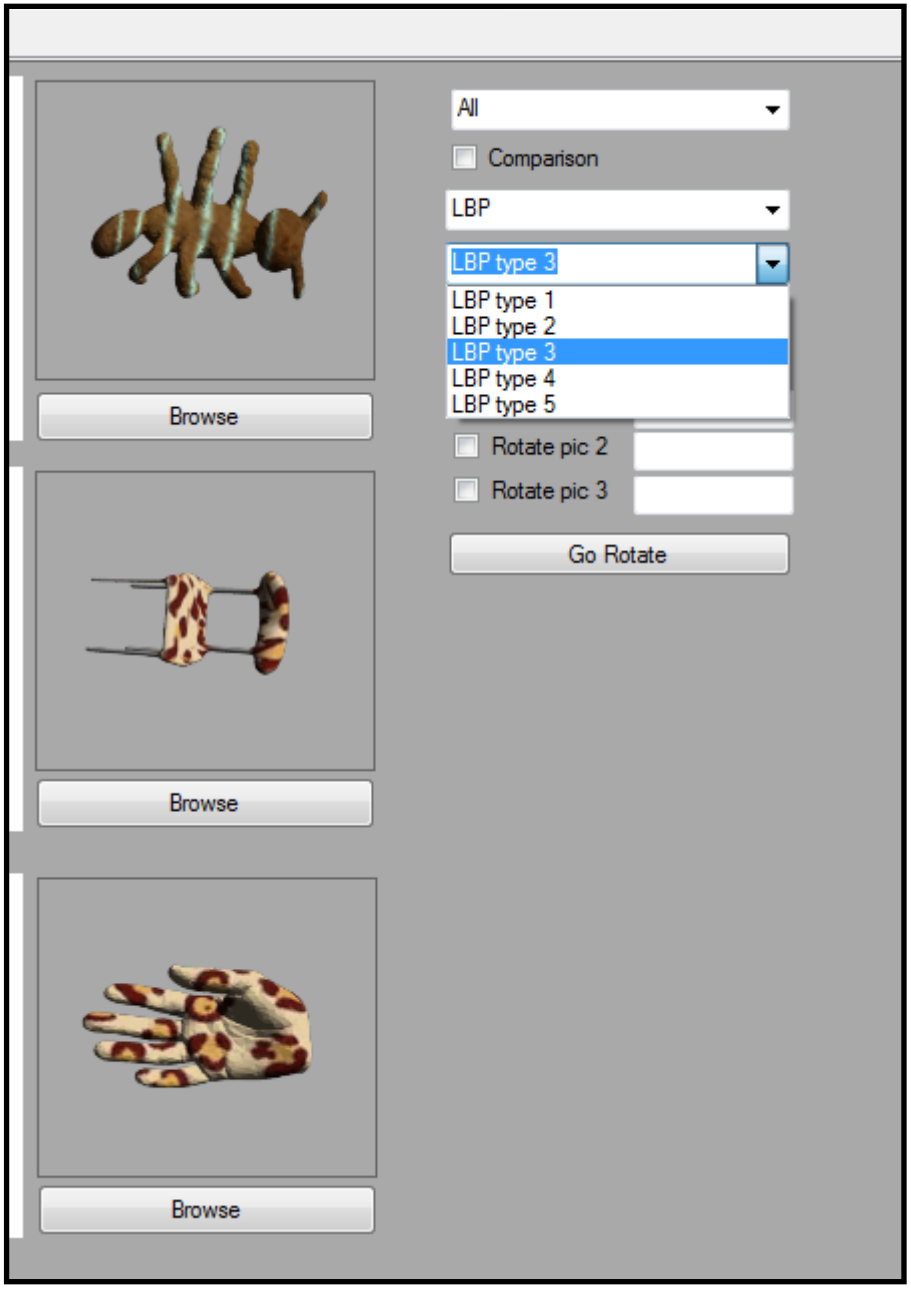
LBP type-1


View Image
RGB
Flip Horizontal
Kuantisasi
Enhancement
Transformasi
Histogram
Transparan
LBP
Filter
Color Detection
6000
4000
2000
0
1535
2035
2535
3035
3535
4035
4535
red
green
blue
Browse
8000
6000
4000
2000
0
1535
2035
2535
3035
3535
4035
4535
red
green
blue
Browse
8000
6000
4000
2000
0
1535
2035
2535
3035
3535
4035
4535
red
green
blue
Browse
All
Comparison
LBP
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate

LBP type-2


Form1
File
View Object
RGB
Flip Horizontal
Kuantisasi
Enhancement
Transformasi
Histogram
Transparan
LBP
Filter
Color Detection
120000
100000
80000
60000
40000
20000
0
-1
9
19
29
red
green
blue
Browse
140000
120000
100000
80000
60000
40000
20000
0
-1
9
19
29
red
green
blue
Browse
140000
120000
100000
80000
60000
40000
20000
0
-1
9
19
29
red
green
blue
Browse
All
Comparison
Color
LBP type 1
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate


All
Comparison
Color
LBP type 3
Run
Histogram
Rotate pic 1
Rotate pic 2
Rotate pic 3
Go Rotate
Browse
Browse
Browse


All
Comparison
LBP
LBP type 3
LBP type 1
LBP type 2
LBP type 3
LBP type 4
LBP type 5
Rotate pic 2
Rotate pic 3
Go Rotate
Browse
Browse
Browse

End