

Praktikum Extraction Feature (Local Binary Pattern)

Hero Yudo Martono

Mei 2016

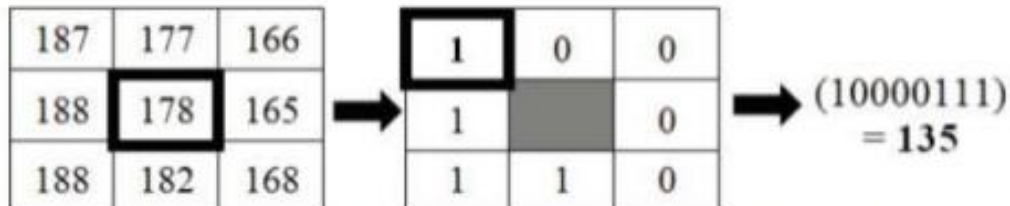
Local Binary Pattern

Local binary patterns

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

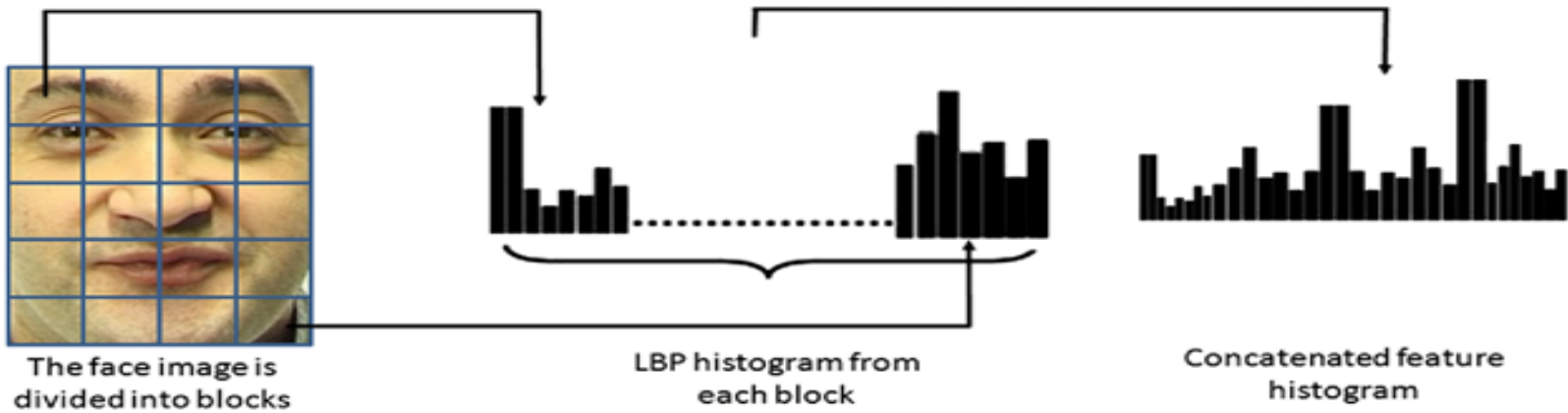
$$s(k) = \begin{cases} 1 & \text{if } k \geq 0 \\ 0 & \text{if } k < 0 \end{cases}$$

l_n : Corresponds to the central pixel value
 l_c : The 8-neighbor 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

4 → 23

1	1	0
1	X	0
0	1	0

1100 0101

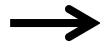
0001 0111

Question

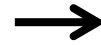
100	110	108
102	105	104
103	107	102

Answer

100	110	108
102	105	104
103	107	102



0	1	1
0	X	0
0	1	0

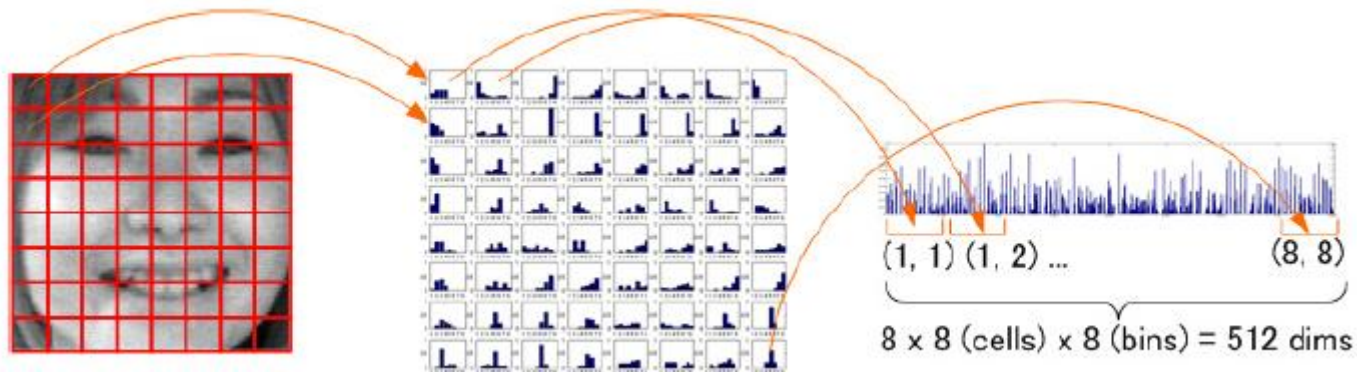
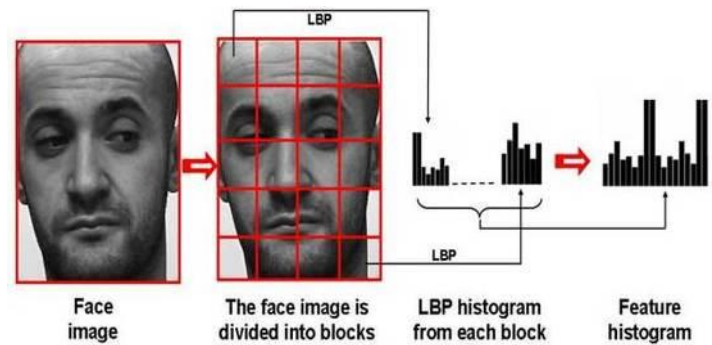
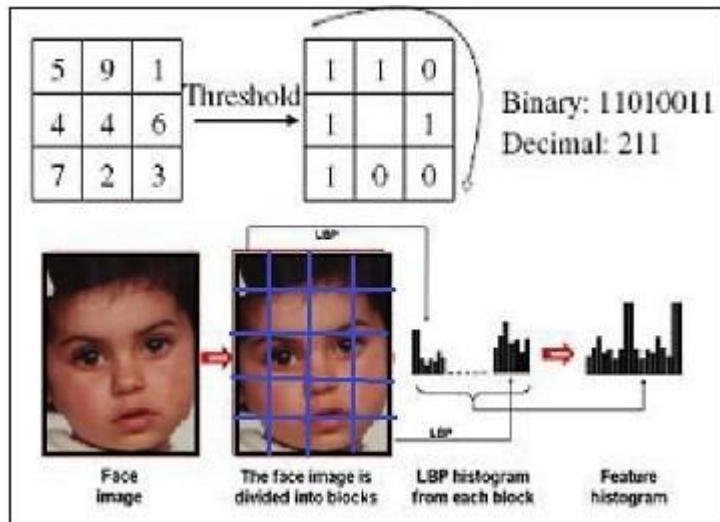


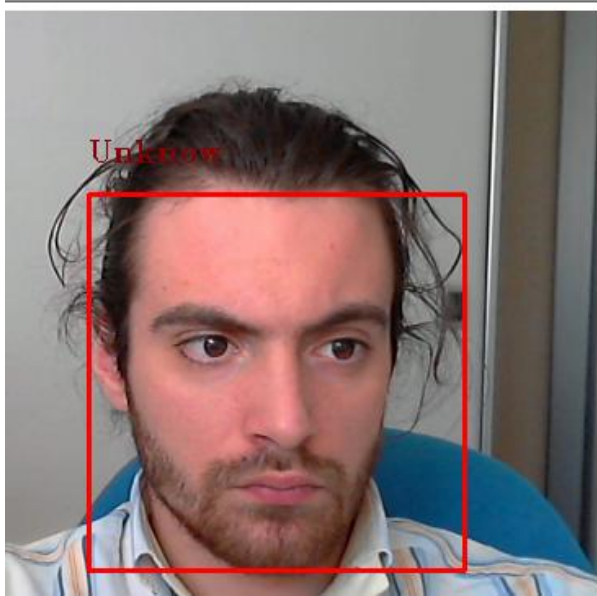
0110 0100



0010 0011

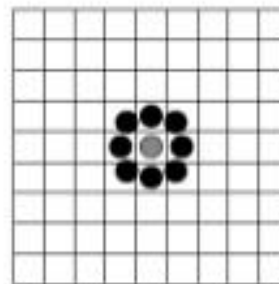
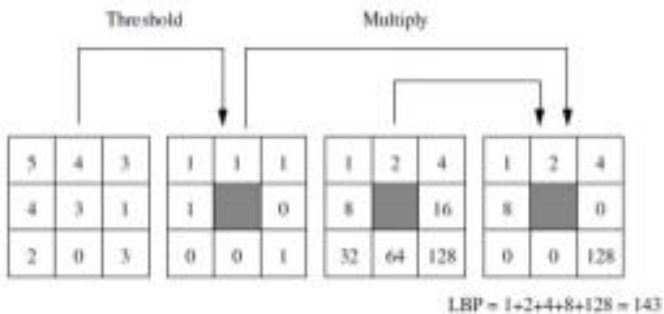
35



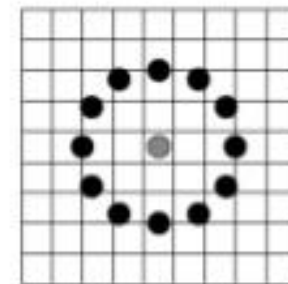


SEGMENTATION I: STRUCTURE-EMPHASIZING LBP FILTER

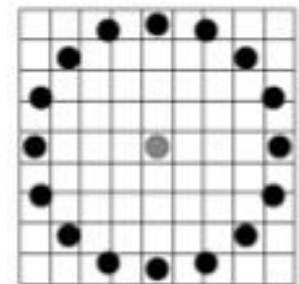
LOCAL BINARY PATTERN:



$P=8, R=1.0$



$P=12, R=2.5$

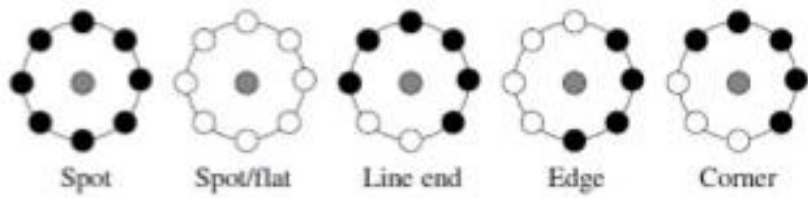


$P=16, R=4.0$

ROTATION INVARIANT UNIFORM LBPs:

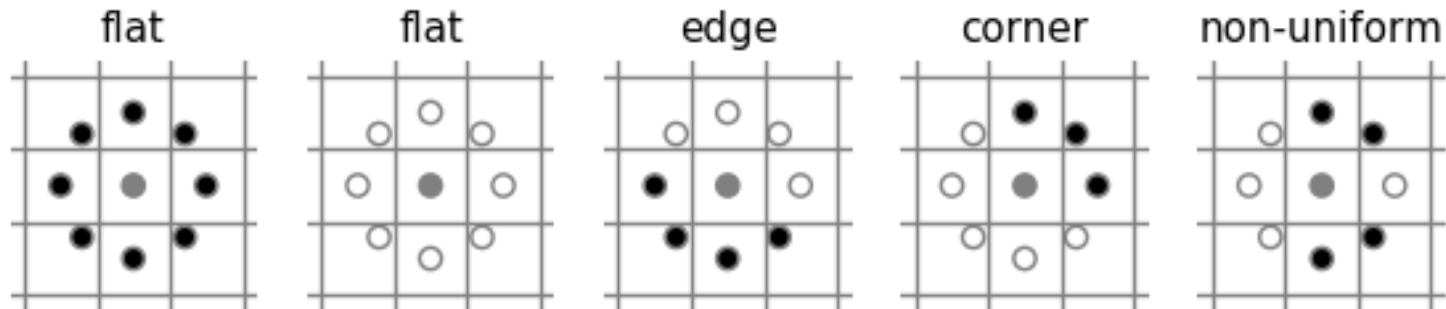


TEXTURE PRIMITIVES:



IM O OJALA ET AL, "MULTIRE SOLUTION 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.

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.

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

0	1	1
1		0
0	0	0

01110000

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

1	1	1
1		0
1	0	1

11110101

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

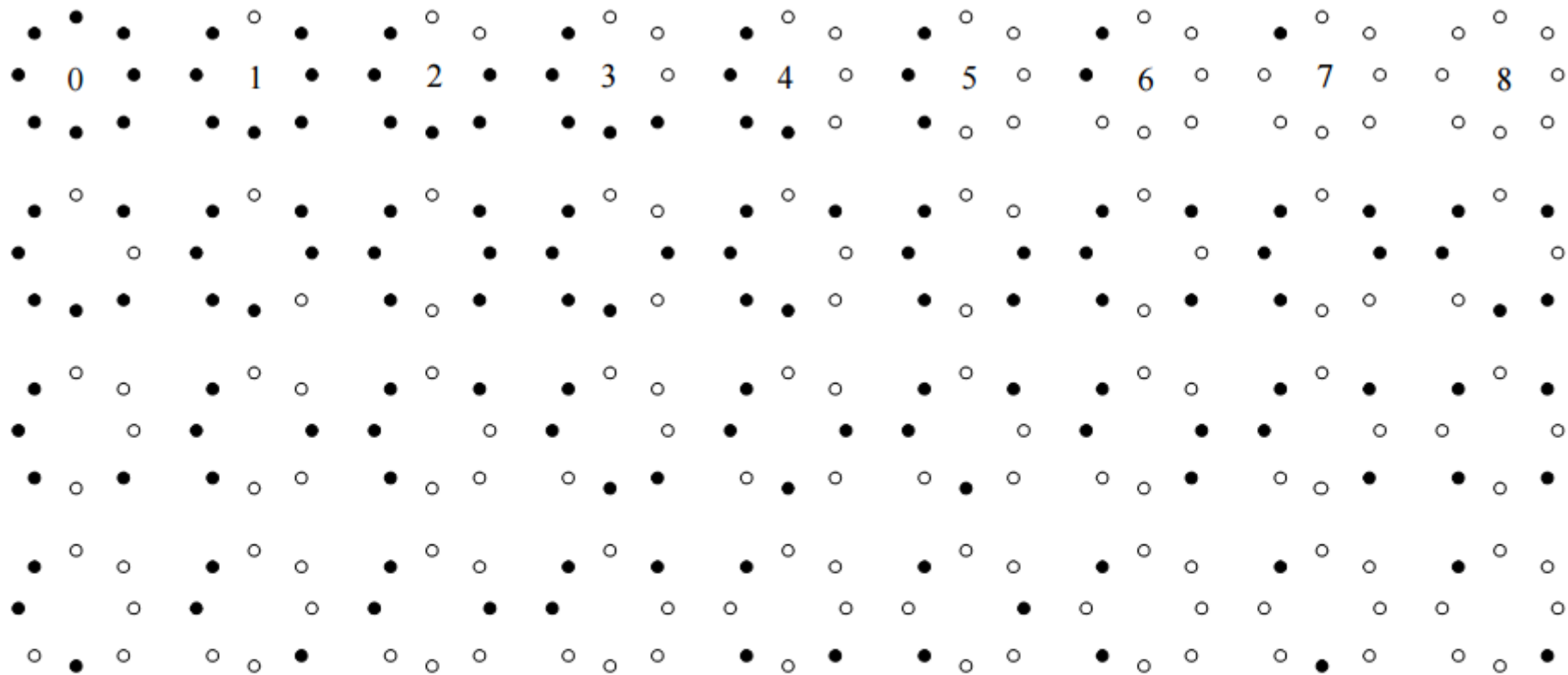


Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine ‘uniform’ patterns, and the numbers inside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

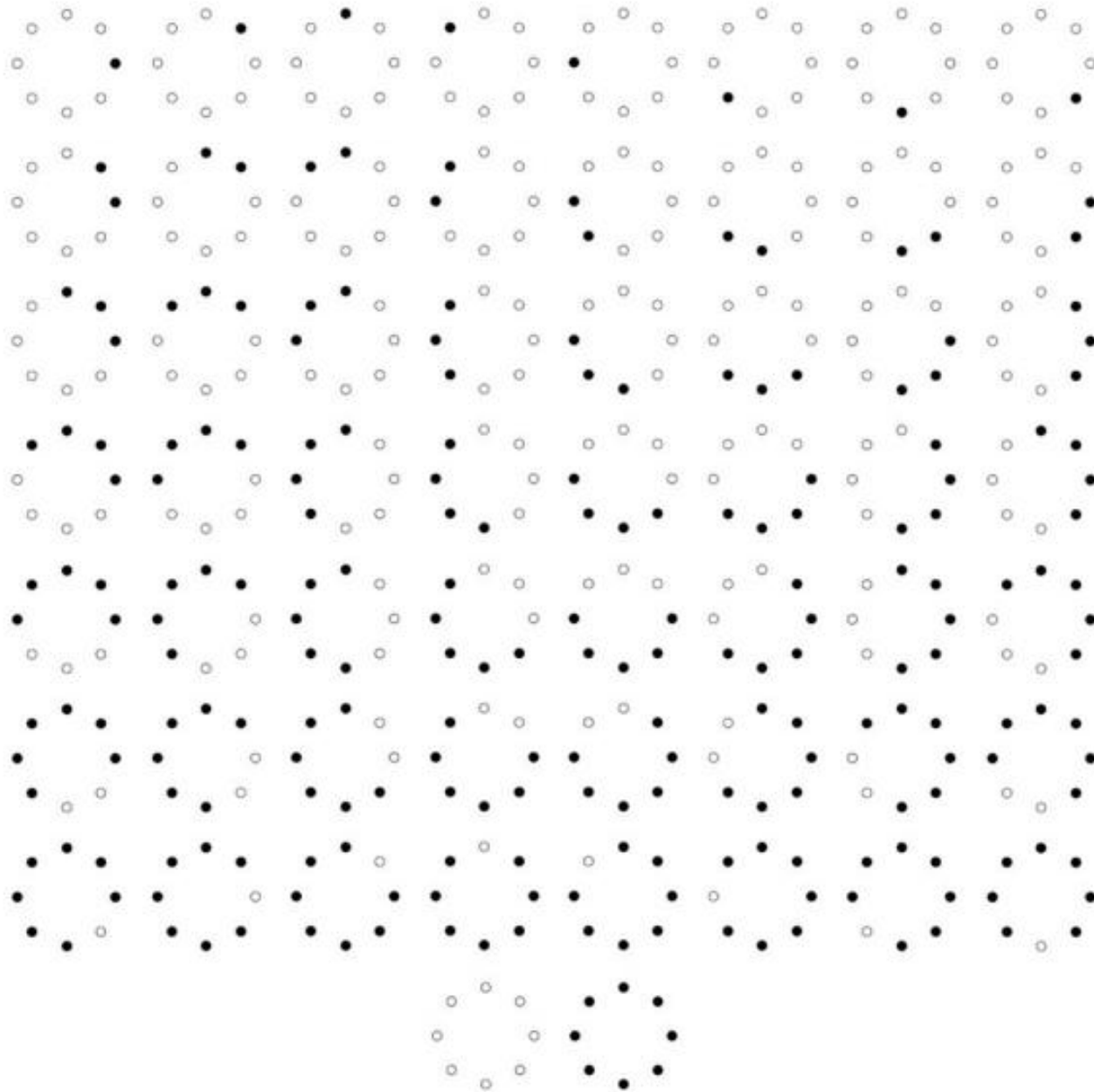


Fig. 3. Uniform LBP patterns when $P=8$. The black and white dots represent the bit values of 1 and 0 in the 8-bit output of the LBP operator.

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, topioli}@ee.oulu.fi
<http://www.ee.oulu.fi/research/imag/texture>

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

where

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

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

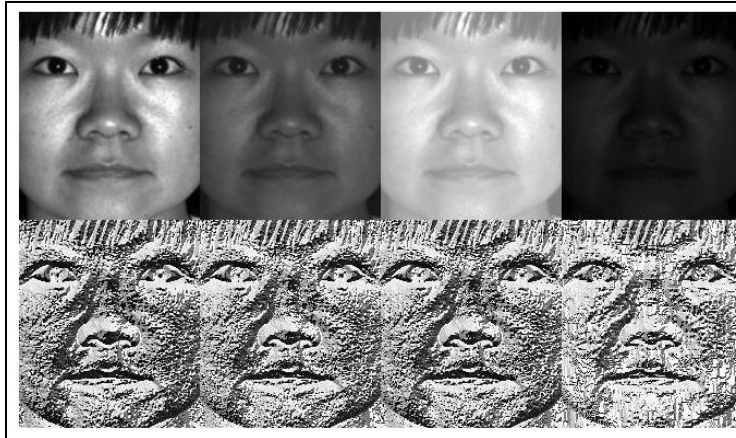
Multiresolution Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns

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, topioli}@ee.oulu.fi
<http://www.ee.oulu.fi/research/imag/texture>

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

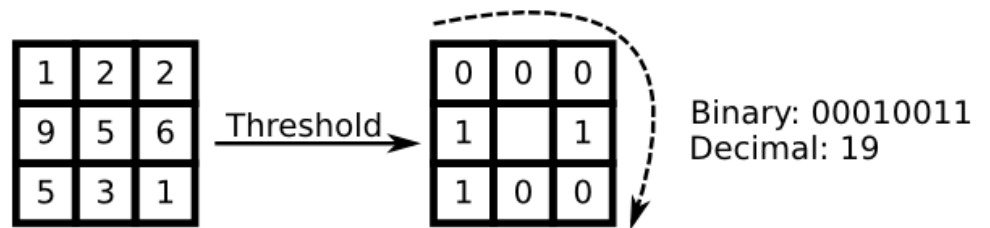
Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine ‘uniform’ patterns, and the numbers inside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

```
1  /*
2  * Copyright (c) 2011. Philipp Wagner <bytefish[at]gmx[dot]de>.
3  * Released to public domain under terms of the BSD Simplified license.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions are met:
7  *   * Redistributions of source code must retain the above copyright
8  *     notice, this list of conditions and the following disclaimer.
9  *   * Redistributions in binary form must reproduce the above copyright
10 *     notice, this list of conditions and the following disclaimer in the
11 *     documentation and/or other materials provided with the distribution.
12 *   * Neither the name of the organization nor the names of its contributors
13 *     may be used to endorse or promote products derived from this software
14 *     without specific prior written permission.
15 *
16 *   See <http://www.opensource.org/licenses/bsd-license>
17 */
18
19 #include "opencv2/core/core.hpp"
20 #include "opencv2/contrib/contrib.hpp"
21 #include "opencv2/highgui/highgui.hpp"
22
```



Algorithmic Description

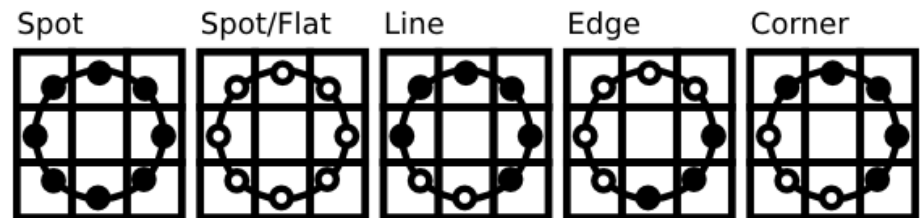
A more formal description of the LBP operator can be given as:

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

, 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:

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

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 classification. However, 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:

$$\begin{aligned} x_p &= x_c + R \cos\left(\frac{2\pi p}{P}\right) \\ y_p &= y_c - R \sin\left(\frac{2\pi p}{P}\right) \end{aligned}$$


Image selection interface for two chairs. The top panel shows a wooden chair on a gray background with a "Browse" button below it. The bottom panel shows a leopard-print chair on a gray background with a "Browse" button below it.

Image selection interface for two chairs. The top panel shows a white chair on a black background with a "Browse" button below it. The bottom panel shows a black and white patterned chair on a black background with a "Browse" button below it.

Image comparison and processing interface. It features two panels, each showing a chair on a grid background with a "Browse" button below. The top panel shows a white chair on a black background, and the bottom panel shows a black chair on a white background. Below the panels are controls for image comparison and processing:

- First Image: ▾
- Comparison
- LBP: ▾
-
- Rotate pic 1
- Rotate pic 2
-

Panel 1 (Left):



Browse



Browse

First Image

Comparison

Color

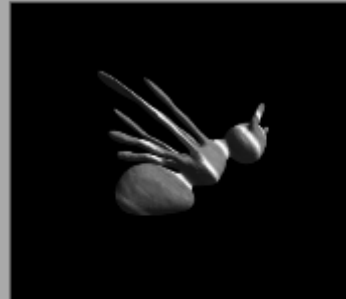
Run Histogram

Rotate pic 1


Rotate pic 2

Go Rotate

Panel 2 (Middle):



Browse



Browse

First Image

Comparison

Gray


Run Histogram

Rotate pic 1

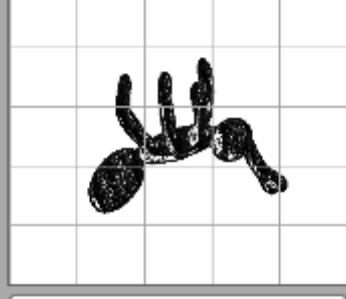
Rotate pic 2

Go Rotate

Panel 3 (Right):



Browse



Browse

First Image

Comparison

LBP

Run Histogram

Rotate pic 1

Rotate pic 2

Go Rotate

Hand image

Browse

Chair image

Browse

First Image

Comparison

LBP

Run Histogram

Rotate pic 1

Rotate pic 2

Go Rotate

Hand image (Grayscale)

Browse

Chair image (Grayscale)

Browse

First Image

Comparison

Gray

Run Histogram

Rotate pic 1

Rotate pic 2

Go Rotate

Hand image (LBP)

Browse

Chair image (LBP)

Browse

First Image

Comparison

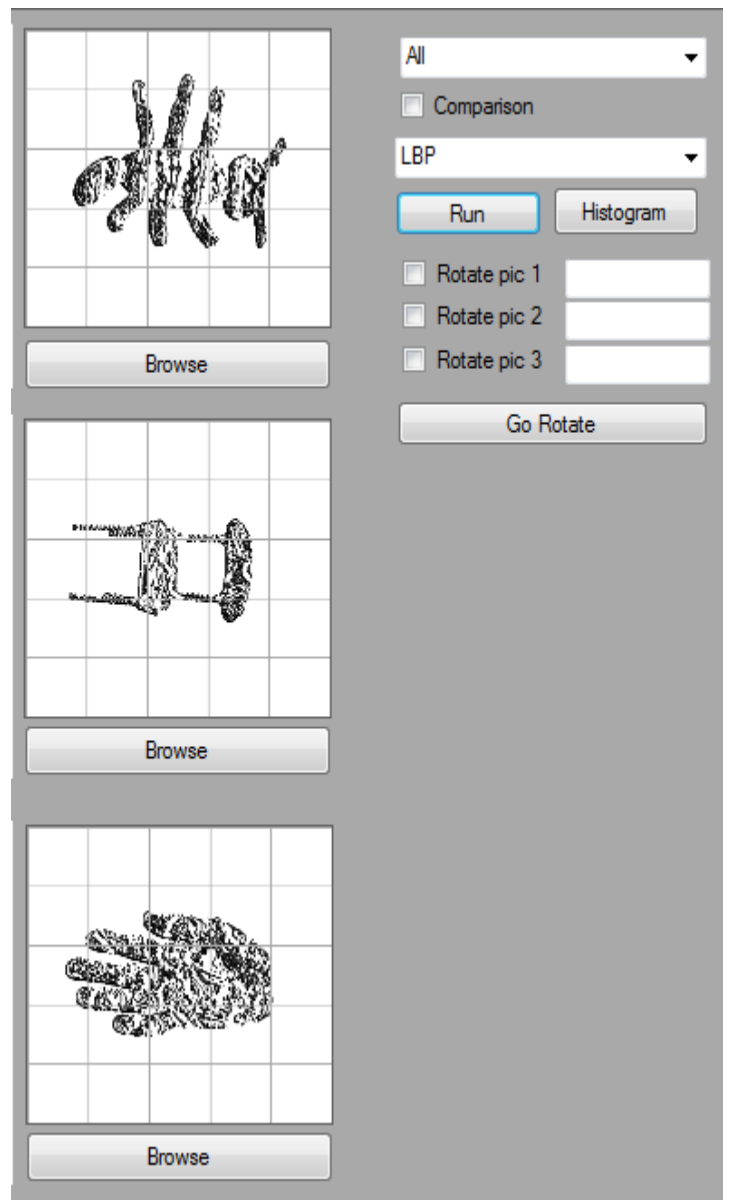
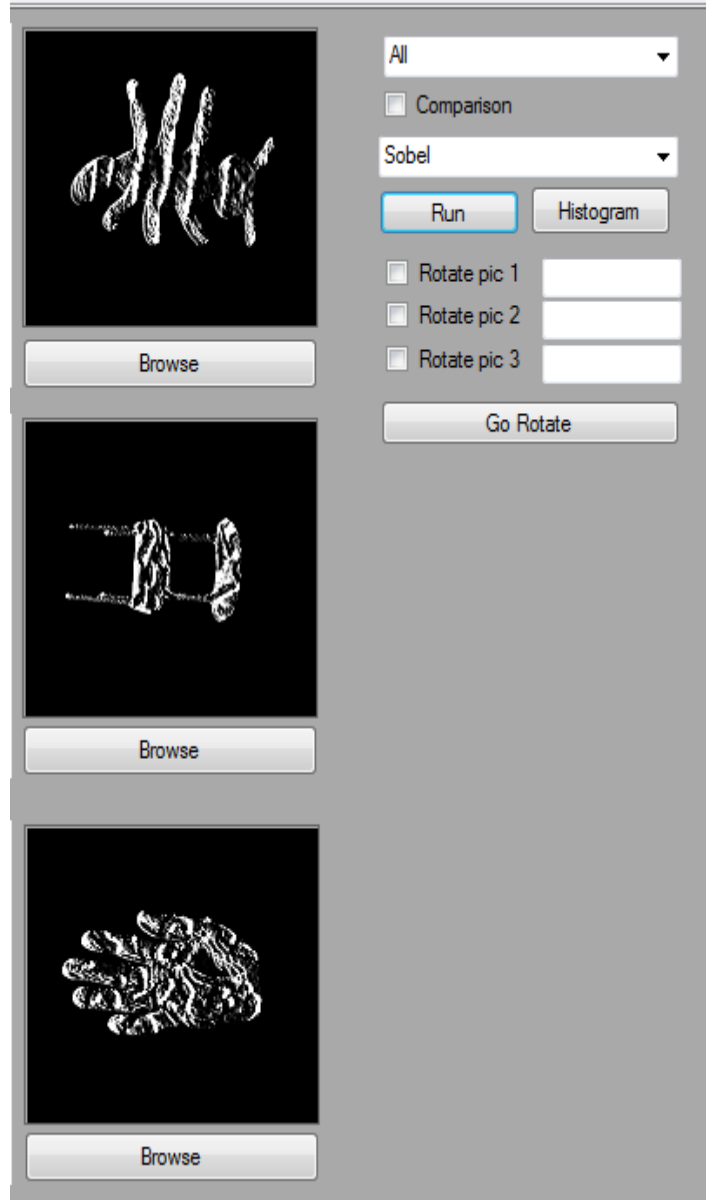
LBP


Run Histogram

Rotate pic 1


Rotate pic 2

Go Rotate







Browse




Browse




Browse



Browse



Browse



Browse

All ▾

Comparison

Sobel ▾

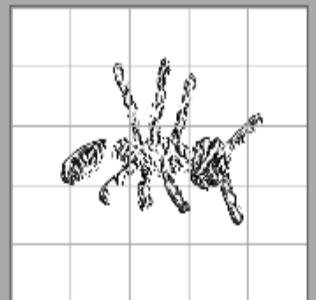
Run Histogram

Rotate pic 1 0

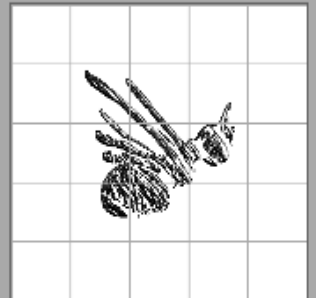
Rotate pic 2 0

Rotate pic 3

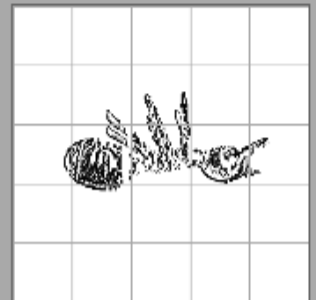
Go Rotate



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;
}
```

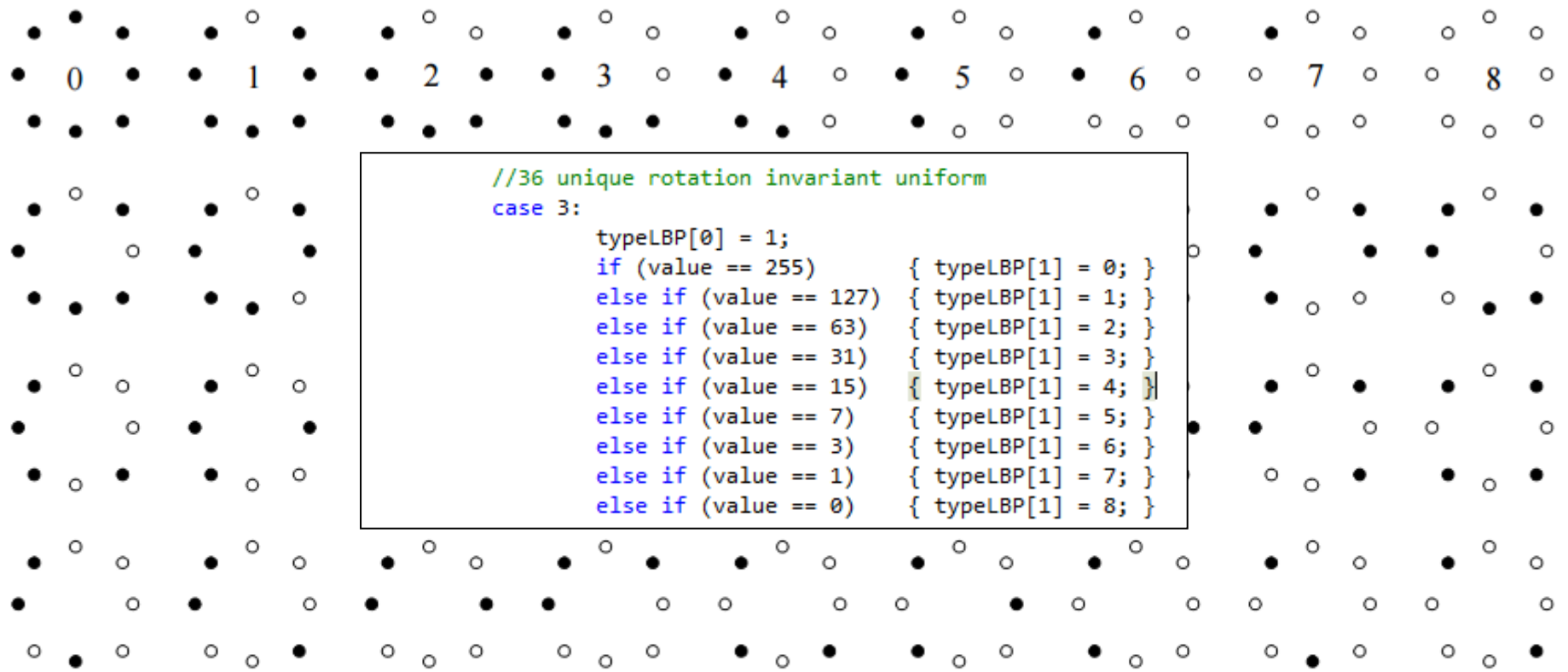


Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine ‘uniform’ patterns, and the numbers inside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

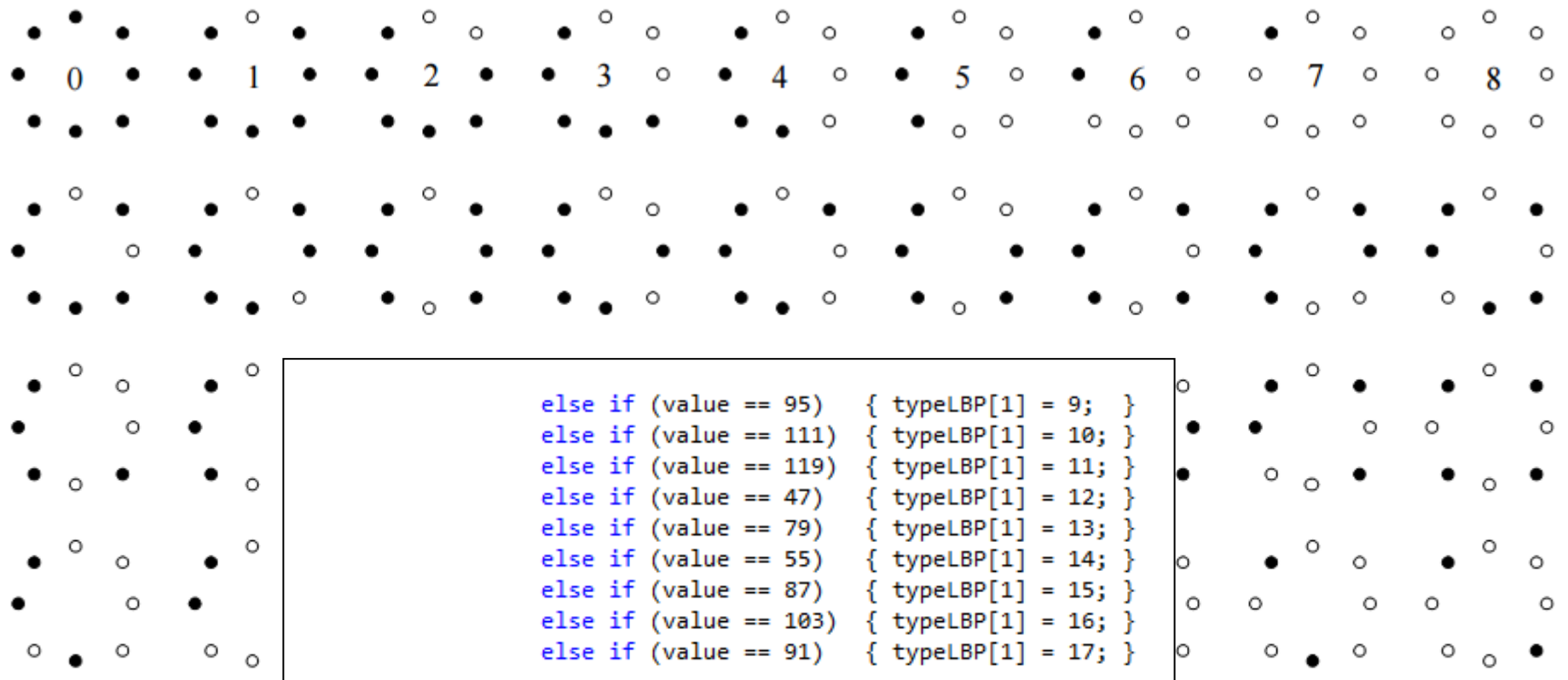


Fig. 2. The 36 unique rotation invariant binary patterns that can occur in the circularly symmetric neighbor set of $LBP_{8,R}^{ri}$. Black and white circles correspond to bit values of 0 and 1 in the 8-bit output of the operator. The first row contains the nine ‘uniform’ patterns, and the numbers inside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

```

```

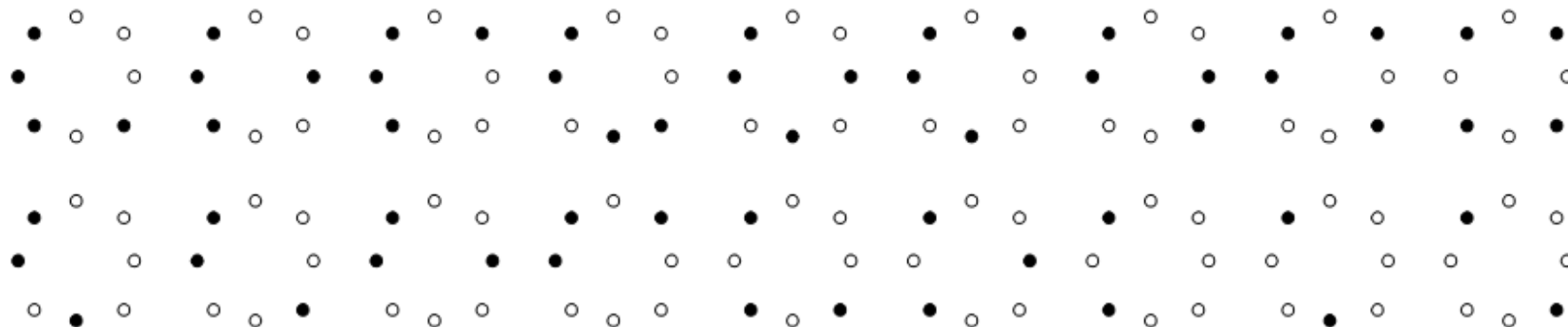
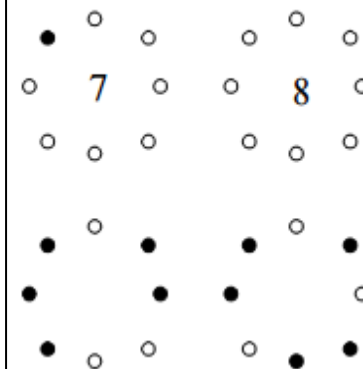
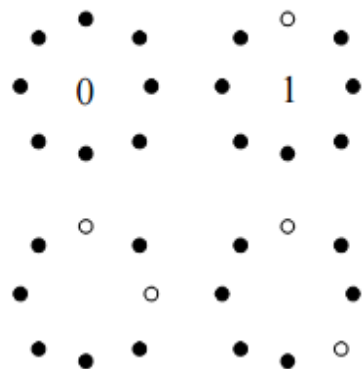
else { typeLBP[0] = 0; typeLBP[1] = 0; }
break;

```

```

}
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;
    }

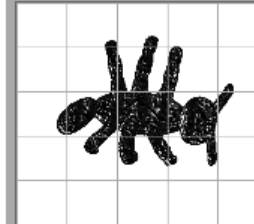
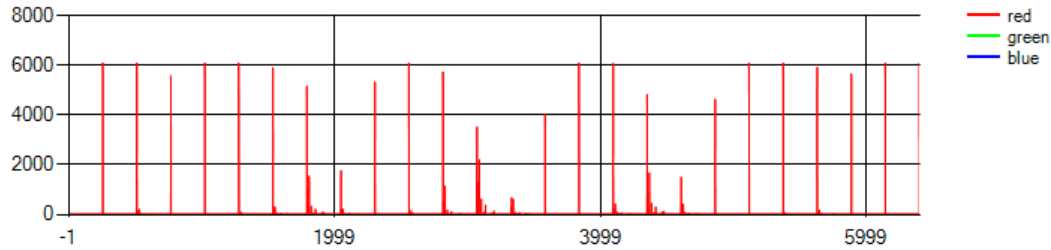
    for (int y = 1; y < numY-1; y++)
    {
        for (int x = 1; x < numX-1; x++)
        {
            for (int i = 0; i < 256; i++)
            {
                switch (num)
                {
                    case 1: this.chart1.Series["red"].Points.AddXY(((y * 5) + x) * 256 + i, histogram[y, x,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



Browse

All

Comparison

LBP

Run

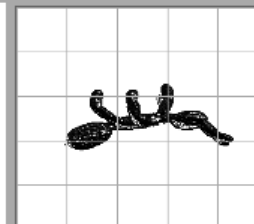
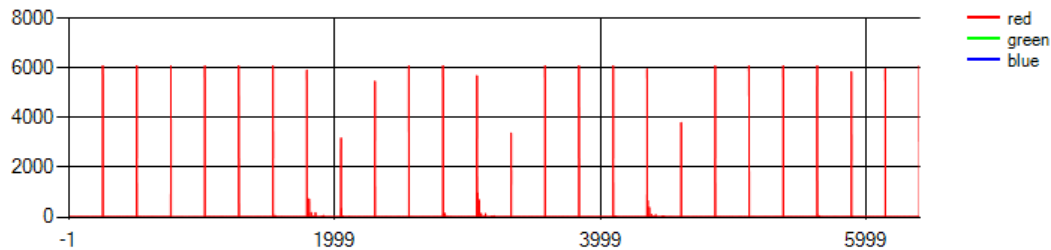
Histogram

Rotate pic 1

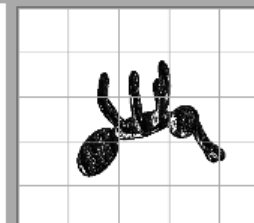
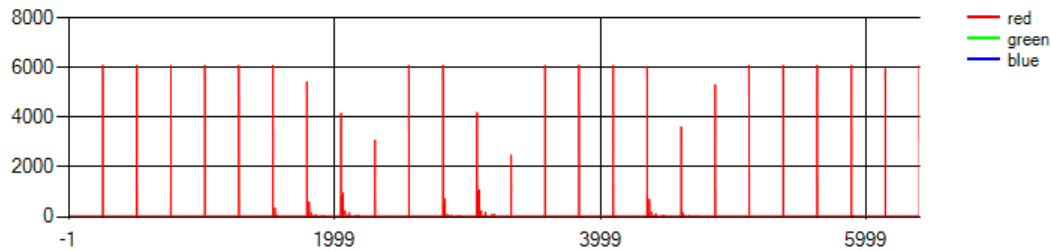
Rotate pic 2

Rotate pic 3

Go Rotate



Browse



Browse

LBP type-1

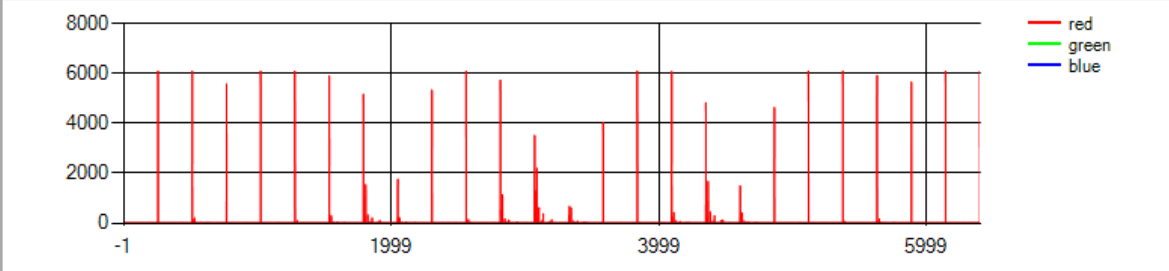
File

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

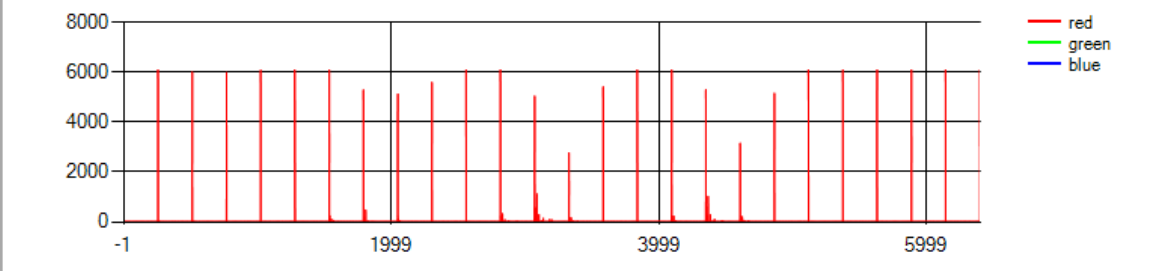


Browse

8000
6000
4000
2000
0

-1 1999 3999 5999

red
green
blue

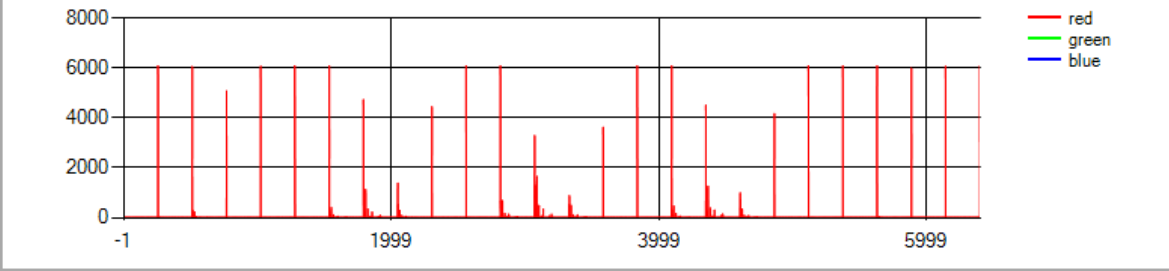


Browse

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

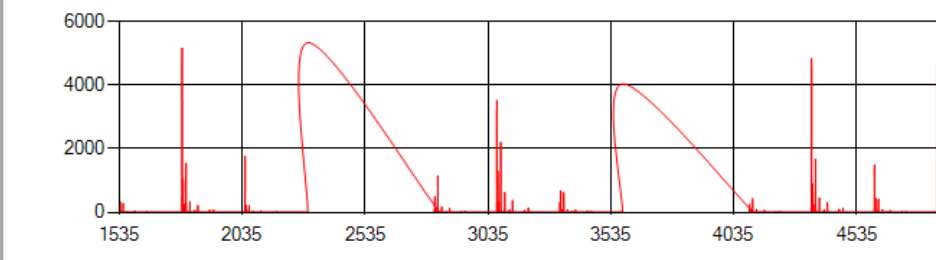
Detailed description: The image shows a software interface for image processing. At the top, there is a menu bar with 'File' and a series of tabs: 'View Image', 'RGB', 'Flip Horizontal', 'Kuantisasi', 'Enhancement', 'Transformasi', 'Histogram', 'Transparan', 'LBP', 'Filter', and 'Color Detection'. The 'Histogram' tab is active. The main area is divided into three rows. Each row contains a histogram on the left and a corresponding image on the right. The histograms show the frequency distribution of pixel values for the red channel (indicated by red bars), with the y-axis ranging from 0 to 8000 and the x-axis from -1 to 5999. The images on the right are binary (black and white) representations of the original images after LBP processing. The first row shows a spider, the second a rotated spider, and the third a hand. To the right of the images is a control panel with a dropdown menu set to 'All', a 'Comparison' checkbox, another dropdown set to 'LBP', 'Run' and 'Histogram' buttons, three 'Rotate pic' checkboxes with input fields, and a 'Go Rotate' button.

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

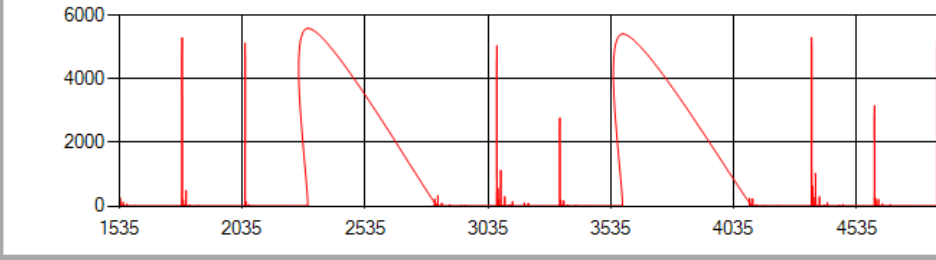


Browse

All
 Comparison
LBP
Run Histogram
 Rotate pic 1
 Rotate pic 2
 Rotate pic 3
Go Rotate

6000
4000
2000
0
1535 2035 2535 3035 3535 4035 4535

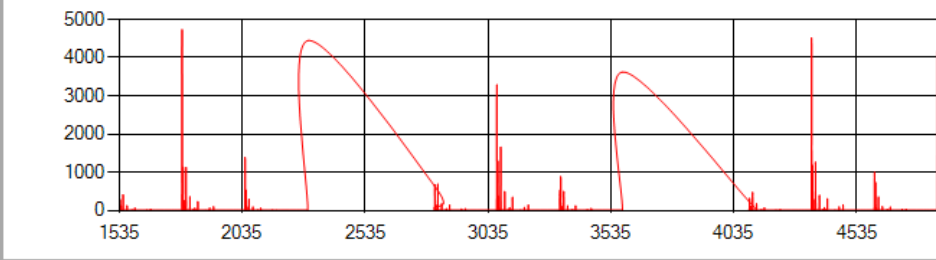
red
green
blue



Browse

5000
4000
3000
2000
1000
0
1535 2035 2535 3035 3535 4035 4535

red
green
blue



Browse

LBP type-2

Form1

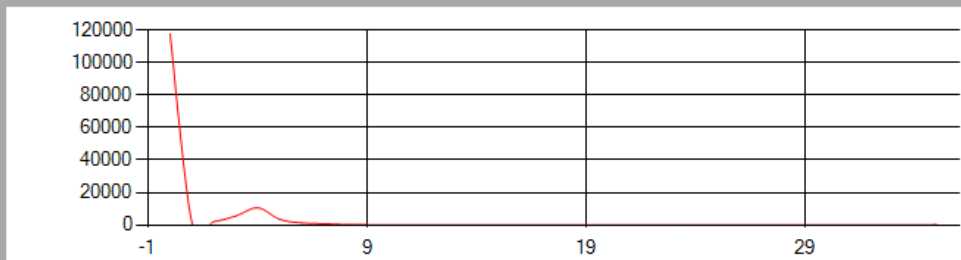
File

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

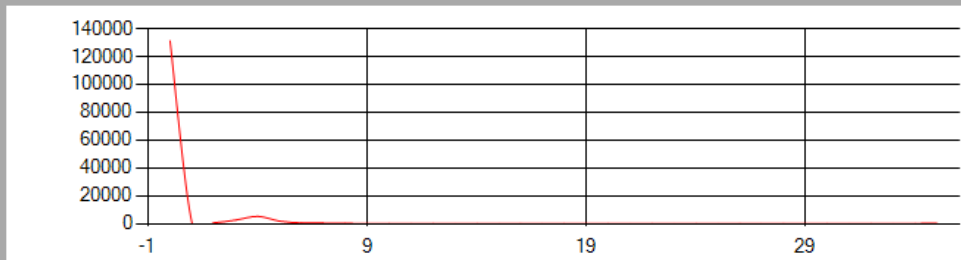


Browse

140000
120000
100000
80000
60000
40000
20000
0

-1 9 19 29

red
green
blue

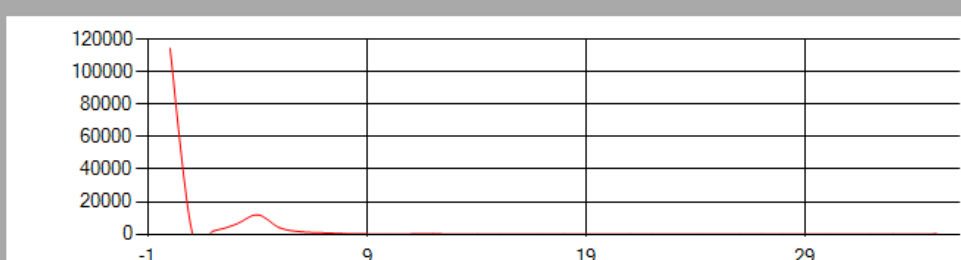


Browse

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

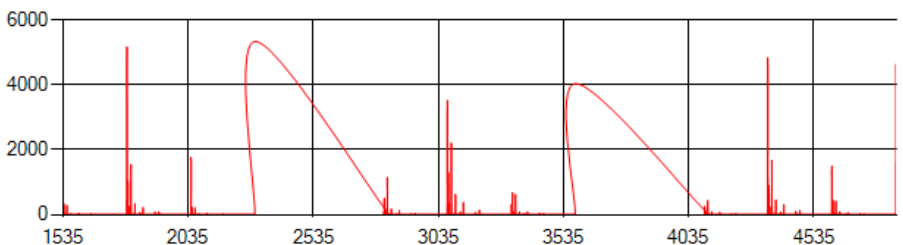
The image shows a software interface for image processing. It features three rows of histograms, each with a corresponding image preview. The histograms show the distribution of pixel values for the red, green, and blue channels. The red channel histograms show a sharp peak at low values (around -1) and a smaller peak around 9. The image previews show a hand with fingers spread, a hand with fingers curled, and a hand with fingers spread. The interface includes a menu bar with options like 'View Object', 'RGB', 'Flip Horizontal', 'Kuantisasi', 'Enhancement', 'Transformasi', 'Histogram', 'Transparan', 'LBP', 'Filter', and 'Color Detection'. On the right side, there are controls for 'All', 'Comparison', 'Color', 'LBP type 1', 'Run', 'Histogram', 'Rotate pic 1', 'Rotate pic 2', 'Rotate pic 3', and 'Go Rotate'.

LBP type-1

View Image RGB Flip Horizontal Kuantisasi Enhancement Transformasi Histogram Transparen LBP Filter Color Detection

6000
4000
2000
0
1535 2035 2535 3035 3535 4035 4535

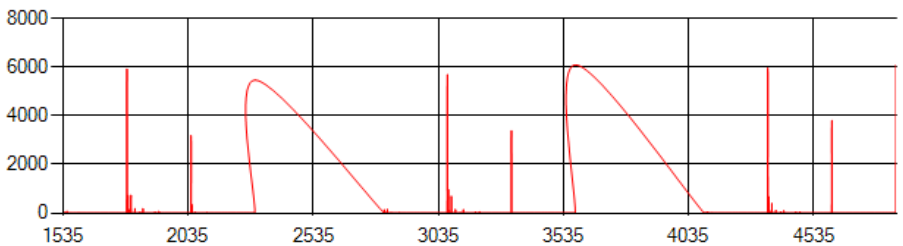
red
green
blue



Browse

8000
6000
4000
2000
0
1535 2035 2535 3035 3535 4035 4535

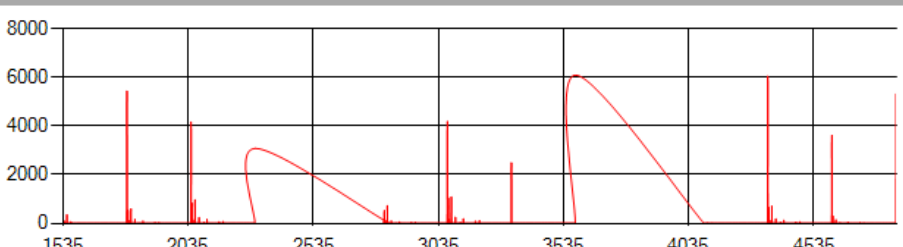
red
green
blue



Browse

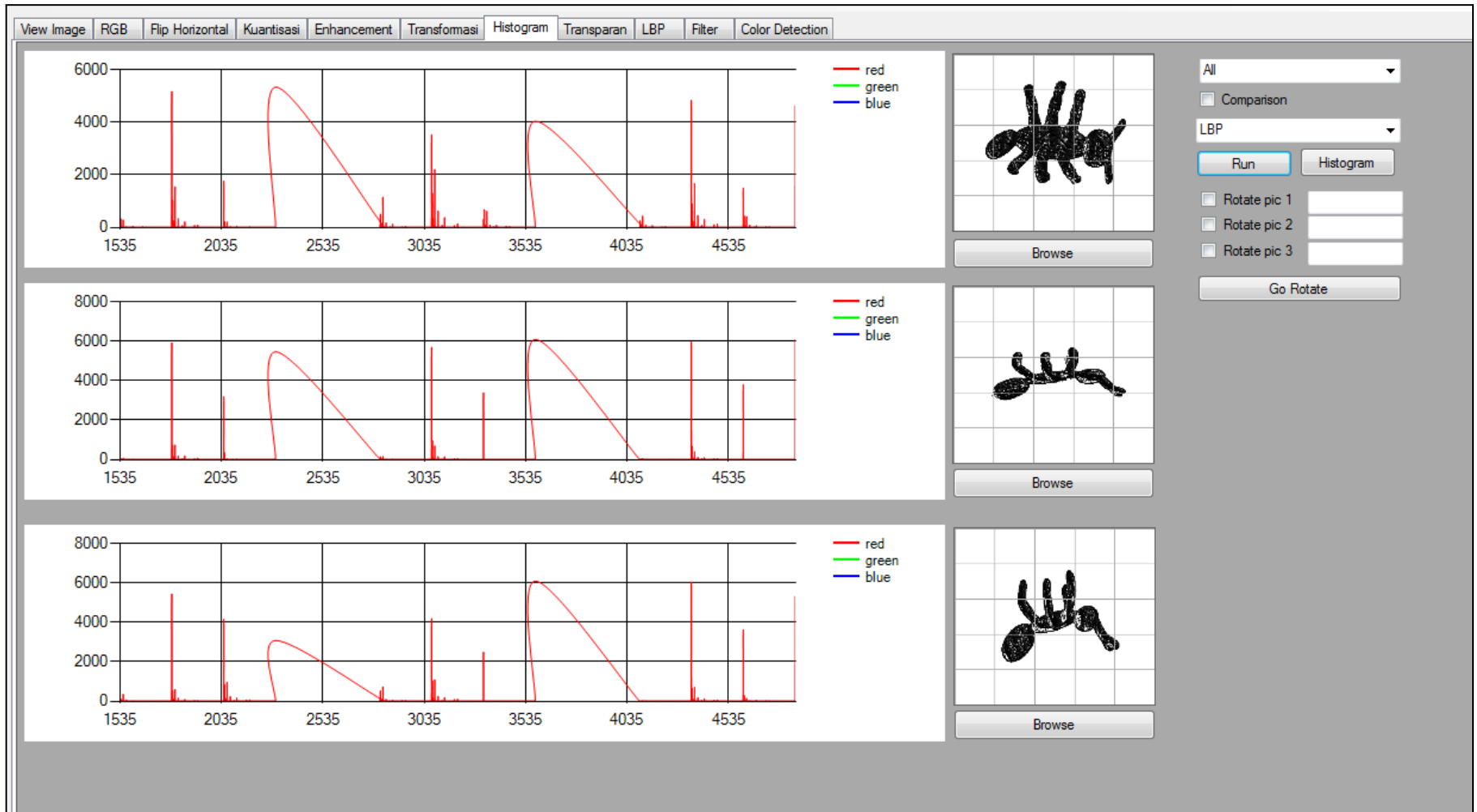
8000
6000
4000
2000
0
1535 2035 2535 3035 3535 4035 4535

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

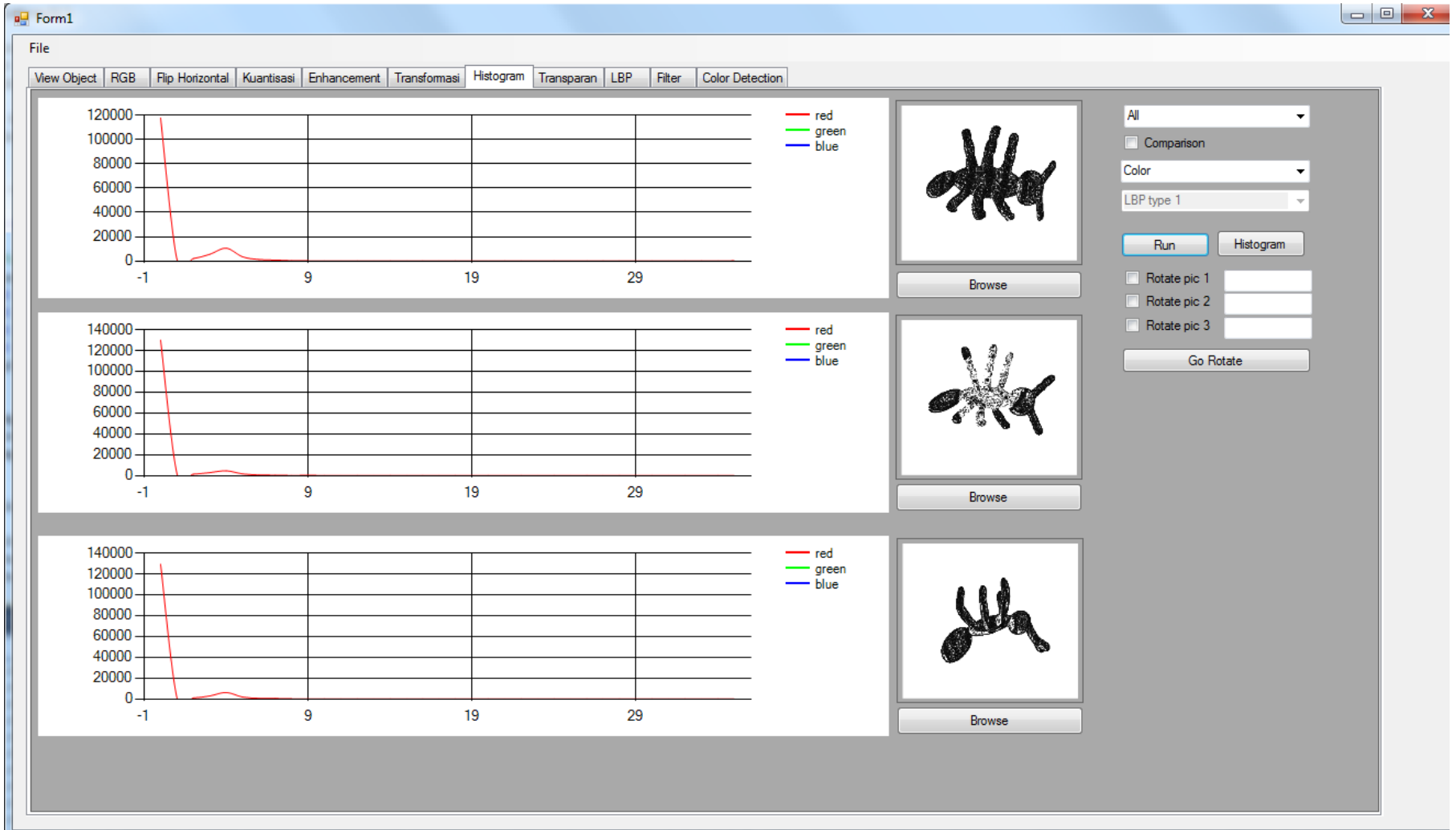
Run Histogram

Rotate pic 1

Rotate pic 2

Rotate pic 3

Go Rotate



UI panel showing three image thumbnails and a control panel.

Image 1: A 3D model of a hand with vertical stripes. Below it is a **Browse** button.

Image 2: A 3D model of a hand with a leopard-like spot pattern. Below it is a **Browse** button.

Image 3: A 3D model of a hand with a leopard-like spot pattern. Below it is a **Browse** button.

Control Panel:

- Dropdown menu: All
- Comparison
- Dropdown menu: Color
- Dropdown menu: LBP type 3
- Buttons: Run, Histogram
- Rotate pic 1 [input field]
- Rotate pic 2 [input field]
- Rotate pic 3 [input field]
- Button: Go Rotate

UI panel showing three image thumbnails and a control panel, with the LBP type dropdown menu open.

Image 1: A 3D model of a hand with vertical stripes. Below it is a **Browse** button.

Image 2: A 3D model of a hand with a leopard-like spot pattern. Below it is a **Browse** button.

Image 3: A 3D model of a hand with a leopard-like spot pattern. Below it is a **Browse** button.

Control Panel:

- Dropdown menu: All
- Comparison
- Dropdown menu: LBP
- Open dropdown menu: LBP type 3 (selected), LBP type 1, LBP type 2, LBP type 4, LBP type 5
- Rotate pic 2 [input field]
- Rotate pic 3 [input field]
- Button: Go Rotate

End