Algorithms for Extracting various Local Texture Features

B Ashwath Rao, N Gopalakrishna Kini
Department of Computer Science & Engg., Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka State, India-576 104
E-mail: ashwath.rao.b@gmail.com

Abstract. In the machine learning and computer vision domain, images are represented using their features. Color, shape, and texture are some of the prominent types of features. Over time, the local features of an image have gained importance over the global features due to their high discerning ability in localized regions. The texture features are widely used in image indexing and content-based image retrieval. In the last two decades, various local texture features have been formulated. For a complete description of images, effective and efficient features are necessary. In this paper, we provide algorithms for 10 local texture feature extraction. These texture descriptors have been formulated since the year 2015. We have designed algorithms so that they are time efficient and memory space-efficient. We have implemented these algorithms and verified their output correctness.

1 Introduction
Image indexing and Content-based Image retrieval are solutions to efficient searching of images from large image databases. For efficient indexing and retrieval of images first, the images are described by using their features. Local Binary Pattern introduced by Ojala et al.[1] is one of the earliest and efficient texture feature descriptors. Since then, various types of local texture features have been introduced by various researchers. In this article, we provide 10 algorithms for various recent texture feature descriptors.

This work is part of a research project on content-based medical image retrieval. We plan to use these algorithms in the task of extracting features from medical images. Algorithms will provide detailed steps for tasks. In this paper, the task is extracting various textural features from images. As of date, more than about 20 texture feature descriptors being formulated by various researchers. Out of these, we have chosen recent 10 texture descriptors. We have provided efficient and easy-to-read algorithms.

The outline of this paper is as follows. In Section 2, we cover the literature review. In Section 3, we cover algorithms for various local texture feature extraction. In Section 4, we provide the future scope and conclude.

2 Literature Review
One of the earliest local texture descriptors is Local Binary Pattern(LBP)[1]. In this feature computation for each non-border pixel, the neighbors at a radius are compared with the center pixel. Based on the sign of the difference formed from comparison, the pattern is
generated. The LBP is simple and has been the basis for devising various other local texture descriptors. Co-occurrence of pixel pairs is a statistical approach towards describing images. In the Center Symmetric Local Binary Co-occurrence pattern (CSLBCoP) [2], first, the center symmetric pattern is computed and then the obtained feature map’s co-occurrence pattern is computed in different directions.

A texture feature descriptor that gives importance to diagonal extremas is proposed and is named Local Diagonal Extrema Pattern (LDEP) [3]. In this feature computation, the first-order local diagonal derivates are computed and then these are compared with the center pixel to form the feature vector. In Local Wavelet Pattern (LWP) [4], the relationship among the neighboring pixels is first wavelet decomposed. Next, the center pixel is transformed to match the range of decomposed neighbors. By establishing the relation between the two, the LWP is derived. The researchers propose this method for CT images retrieval based on their content. Local Tri-directional Pattern (LTriDP) [5] uses the relationship between adjacent neighbors with neighboring pixels and the relationship of the neighboring pixel with the center pixel. A three-directional relationship is encoded in this pattern.

Local Bit-plane Decoded Pattern (LBDP) [6] is similar to LBP. But instead of comparing pixel intensities of neighbors with center pixel, a transformed neighbors with center pixel is used. The vector formed out of each bit from among neighbors is used instead of the direct neighbor intensity. Local Neighborhood Difference Pattern (LNDP) [7], is a texture feature descriptor that considers the relation between neighbors with its adjacent pixels. The sign of difference of neighbor with its adjacent pixel is used in computing the pattern. A novel texture feature descriptor called Local Neighborhood Intensity difference Pattern (LNIP) is proposed [8]. This descriptor encodes the difference of neighboring pixels with its adjacent neighbors. Along with the difference(sign) pattern, the magnitude pattern is also encoded in this pattern. Two new feature descriptors named Threshold Local Binary AND Pattern (TLBAP) and Local Adjacent Neighborhood Average Difference Pattern (LANADP) are proposed [9]. TLBAP uses a threshold calculated from among neighbors and performs bit-wise AND with LBP feature to compute the feature at a center pixel. This is repeated for all non-border pixels to compute the TLBAP feature for the entire image. The LANADP feature [9] is computed based on the difference of neighboring pixels with its neighbors in vertical, horizontal, and diagonal directions.

3 Algorithms for various Local Texture Features Extraction

3.1 Local Binary Pattern

Algorithm 1 shows detailed steps for Local Binary Pattern feature extraction from an image. The image intensity matrix is \( arr \), which is input to the findLBP function as an argument. The arrays \( dx \) and \( dy \) contain the relative offset in the X and the Y direction required for determining the immediate 8 neighbors around a center pixel. The array \( \text{multiple2} \) contains powers of two ranging from \( 2^0 \) to \( 2^7 \). If the sign of differences of the neighbor pixel with the center pixel is positive or zero, the corresponding powers of two are added. The for loop in line 12 in the algorithm is used to traverse across the 8 neighbors of a center pixel.

Algorithm 1 Algorithm for Local Binary Pattern Feature Extraction

```plaintext
function findLBP(arr)
  h, w ← height and width of array arr
  dx ← \{1, 1, 0, -1, -1, -1, 0, 1\}
  dy ← \{0, -1, -1, -1, 0, 1, 1, 1\}
  multiple2[1] ← 1
  for l = 2, ..., 8 do
    multiple2[l] ← bitshift(multiple2[l - 1], 1))
  for y = 2, ..., h - 1 do
```
for $x = 2, \ldots, w - 1$ do
    $btot \leftarrow 0$
    $centerPx \leftarrow arr[y][x]$
    for $k = 1, \ldots, 8$ do
        $neigh \leftarrow arr[y + dy[k]][x + dx[k]]$
        if $neigh >= centerPx$ then
            $btot \leftarrow btot + \text{multiple}2[k]$
        end if
    end for
    $lbpFeatures[y - 1][x - 1] \leftarrow btot$
end for

3.2 Center Symmetric Local Binary Co-occurrence Pattern

Algorithm 2 shows steps for computation of Center Symmetric Local Binary Co-occurrence Pattern. In line 10 in the algorithm, the center symmetric pattern is computed. The difference of a neighbor with its symmetric neighbor is used in the center symmetric pattern. Then in line 12, using the $\text{GRAYCOMATRIX}$ function, the co-occurrence feature of the center symmetric pattern is generated by taking offset in 4 angles ($0^\circ, 45^\circ, 90^\circ, \text{and} 135^\circ$).

Algorithm 2 Algorithm for Center Symmetric Local Binary Co-occurrence Pattern Feature Extraction

1 function findCSCoLBPFeatures(arr)
2 \(h, w \leftarrow \text{height and width of array } arr\)
3 \(dx \leftarrow \{1, 1, 0, -1, -1, 0, 1\}\)
4 \(dy \leftarrow \{0, -1, -1, 0, 1, 1\}\)
5 let $csFeatures$ be an unsigned byte array of size $h - 2 \times w - 2$
6 for $y = 2, \ldots, h - 1$ do $\triangleright$ navigate across row
7     for $x = 2, \ldots, w - 1$ do $\triangleright$ navigate across column
8         for $k = 1, \ldots, 4$ do
9             if $arr[y + dy[k]][x + dx[k]] - arr[y + dy[k + 4]][x + dx[k + 4]] >= 0$ then
10                 $csFeatures[y - 1][x - 1] \leftarrow \text{BITSET}(csFeatures[y - 1][x - 1], k)$
11             end if
12         end for
13         $offsets \leftarrow \{0, 1, -1, 1, 0, -1\}\)
14         $cscolbpFeatures \leftarrow \text{GRAYCOMATRIX}(csFeatures, 'Offset', offsets, 'NumLevels', 16, 'GrayLimits', [015])$
15         $cscolbpFeatures \leftarrow cscolbpFeatures[:,0] + cscolbpFeatures[:,1] + cscolbpFeatures[:,2] + cscolbpFeatures[:,3]$
16         $cscolbpFeatures \leftarrow \text{RESHAPE}(cscolbpFeatures, 1, []$
17     end for
18 return cscolbpFeatures

3.3 Local Diagonal Extrema Pattern

Algorithm 3 shows detailed steps in computing the Local Diagonal Extrema Pattern for an image. The for loops in lines 10 and 11 iterates across rows and columns of the image. The loop starts with 2 as the first row or column is a border pixel. Lines 21 to 27 determine extrema values. Finally, the LDEP feature is generated based on the difference of extrema values with the center pixel.

Algorithm 3 Algorithm for Local Diagonal Extrema Pattern Feature Extraction

1 function findLDEP(arr)
2 \(h, w \leftarrow \text{width and height of array } arr\)
3 \(dx \leftarrow \{1, 1, 0, -1, -1, 0, 1\}\)
4 \(dy \leftarrow \{0, -1, -1, 0, 1, 1\}\)
5 $maxNegDiff \leftarrow 32767$
6 \text{minPosDiff} \leftarrow 32767 \\
7 \text{multiple}2[1] \leftarrow 1 \\
8 \text{for } l = 2, \ldots, 16 \text{ do} \\
9 \quad \text{multiple}2[l] \leftarrow \text{BITSHIFT}(\text{multiple}2[l - 1], 1) \\
10 \text{for } y = 2, \ldots, h - 1 \text{ do} \\
11 \quad \text{for } x = 2, \ldots, w - 1 \text{ do} \\
12 \quad \quad \text{centerPx} \leftarrow \text{arr}[y][x] \\
13 \quad \quad \text{diagNeigh}[1] \leftarrow \text{arr}[y - 1][x + 1] \\
14 \quad \quad \text{diagNeigh}[2] \leftarrow \text{arr}[y - 1][x - 1] \\
15 \quad \quad \text{diagNeigh}[3] \leftarrow \text{arr}[y + 1][x - 1] \\
16 \quad \quad \text{diagNeigh}[4] \leftarrow \text{arr}[y + 1][x + 1] \\
17 \quad \text{tauMax} \leftarrow 1 \\
18 \quad \text{tauMin} \leftarrow 1 \\
19 \quad \text{for } g = 0, \ldots, 2 \text{ do} \\
20 \quad \quad \text{for } t = 0, \ldots, 3 \text{ do} \\
21 \quad \quad \quad \text{diff} \leftarrow \text{diagNeigh}[t + g + 1\%4] + 1) - \text{diagNeigh}[t + 1] \\
22 \quad \quad \quad \text{if } \text{diff} < 0\&\& \text{diff} < \text{maxNegDiff} \text{ then} \\
23 \quad \quad \quad \quad \text{maxNegDiff} \leftarrow \text{diff} \\
24 \quad \quad \quad \quad \text{tauMax} \leftarrow t + 1 \\
25 \quad \quad \quad \text{if } \text{diff} >= 0\&\& \text{diff} < \text{minPosDiff} \text{ then} \\
26 \quad \quad \quad \quad \text{minPosDiff} \leftarrow \text{diff} \\
27 \quad \quad \quad \quad \text{tauMin} \leftarrow t + 1 \\
28 \quad \quad \quad \quad \text{itaumax} \leftarrow \text{diagNeigh}[\text{tauMax}] \\
29 \quad \quad \quad \quad \text{itamin} \leftarrow \text{diagNeigh}[\text{tauMin}] \\
30 \quad \quad \quad \text{if } \text{itaumax} - \text{centerPx} >= 0\&\& \text{itamin} - \text{centerPx} >= 0 \text{ then} \\
31 \quad \quad \quad \quad \delta \leftarrow 1 \\
32 \quad \quad \quad \text{else if } \text{itaumax} - \text{centerPx} < 0\&\& \text{itamin} - \text{centerPx} < 0 \text{ then} \\
33 \quad \quad \quad \quad \delta \leftarrow 0 \\
34 \quad \quad \quad \text{else} \\
35 \quad \quad \quad \quad \delta \leftarrow 2 \\
36 \quad \quad \quad \ldepFeatures[y - 1][x - 1] \leftarrow \text{BITSHIFT}(1, \text{tauMax} + 8 \ast \delta) + \text{BITSHIFT}(1, \text{tauMin} + 4 + 8 \ast \delta) \\
37 \quad \text{return } \ldepFeatures

3.4 Local Wavelet Pattern
Algorithm 4 shows detailed steps for computing Local Wavelet Patter (LWP) for an image. The pre-computed Haar matrix coefficients are saved in the \text{haarMat} variable in line 18. The neighbors of a center pixel are wavelet decomposed in line 24. Then, the center pixel is transformed in line 25. Based on the sign of difference between wavelet decomposed neighbors and transformed center pixel, the feature is calculated in line 29.

Algorithm 4 Algorithm for Local Wavelet Pattern Feature Extraction

1 \textbf{function} \text{lwpcenterPixelTransform}(c, neighbourCount, N, l, g) \\
2 \quad \textbf{if} \ \text{neighbourCount} <= N/(\text{BITSHIFT}(1, l)) \textbf{then} \\
3 \quad \quad \text{val} \leftarrow \text{BITSHIFT}(1, (l/2)) \ast c \\
4 \quad \textbf{else if} \ \text{neighbourCount} <= N/(\text{BITSHIFT}(1, l - 1)) \textbf{then} \\
5 \quad \quad \textbf{if} \ \ l > 1 \textbf{then} \\
6 \quad \quad \quad \text{val} \leftarrow 2^{(l/2)} \ast c - 2^{(l/2 - 1)} \ast (g - 1) \\
7 \quad \quad \textbf{else} \\
8 \quad \quad \quad \text{val} \leftarrow 2^{(l/2)} \ast c - (g - 1)/2
Algorithm 5 shows detailed steps for computing Local Tri-Directional Pattern Feature Extraction for an image. For each of the neighbors, the difference in three directions is calculated in lines 22 to 24. Based on these differences the positive and negative pattern is generated. By using norm(Euclidean distance), the magnitude pattern is determined in line 32.

**Algorithm 5** Algorithm for Local Tri-Directional Pattern Feature Extraction

```plaintext
function LTriDP_F(val)
    if val < 0 then
        out ← 1
    else
        out ← 0
    return out

function FINDLTriDP(arr)
    h, w ← height and width of array arr
    dx ← {1, 1, 0, −1, −1, −1, 0, 1}
    dy ← {0, 1, 1, 0, −1, −1, −1}
```

3.5 Local Tri-directional Pattern
let multiple2 be an unsigned byte array of size 8

\[ \text{multiple2}[1] \leftarrow 128 \]

\[
\text{for } l = 2, \ldots, 8 \text{ do}
\]

\[ \text{multiple2}[l] \leftarrow \text{BITSHIFT}(\text{multiple2}[l-1],-1) \]

let \( \text{ltridpp, ltridpn, ltridpm} \) be 3D unsigned byte arrays of size \( 3 \times h - 2 \times w - 2 \)

\[
\text{for } y = 2, \ldots, h - 1 \text{ do} \quad \triangleright \text{navigate across row}
\]

\[
\text{for } x = 2, \ldots, w - 1 \text{ do} \quad \triangleright \text{navigate across column}
\]

\[
\text{bitot1} \leftarrow 0
\]

\[
\text{bitot2} \leftarrow 0
\]

\[
\text{bmag} \leftarrow 0
\]

\[
\text{for } k = 1, \ldots, 8 \text{ do}
\]

\[
d_1 \leftarrow \text{LTriDP}_f(\text{arr}[y + dy[k]][x + dx[k]] - \text{arr}[y + dy[\text{mod}(k-2,8)+1]][x+dx[\text{mod}(k-2,8)+1]])
\]

\[
d_2 \leftarrow \text{LTriDP}_f(\text{arr}[y + dy[k]][x + dx[k]] - \text{arr}[y + dy[\text{mod}(k,8)+1]][x+dx[\text{mod}(k,8)+1]])
\]

\[
d_3 \leftarrow \text{LTriDP}_f(\text{arr}[y + dy[k]][x + dx[k]] - \text{arr}[y][x])
\]

\[
\text{if } (d_1 + d_2 + d_3) \text{mod} 3 == 1 \text{ then}
\]

\[
\text{bitot1} \leftarrow \text{bitot1} + \text{multiple2}[k]
\]

\[
\text{else if } (d_1 + d_2 + d_3) \text{mod} 3 == 2 \text{ then}
\]

\[
\text{bitot2} \leftarrow \text{bitot2} + \text{multiple2}[k]
\]

\[
\text{m1} \leftarrow \text{NORM}([\text{arr}[y + dy[\text{mod}(k-2,8)+1]][x+dx[\text{mod}(k-2,8)+1]] - \text{arr}[y][x]][\text{arr}[y + dy[\text{mod}(k,8)+1]][x+dx[\text{mod}(k,8)+1]] - \text{arr}[y][x]])
\]

\[
\text{m2} \leftarrow \text{NORM}([\text{arr}[y + dy[\text{mod}(k-2,8)+1]][x+dx[\text{mod}(k-2,8)+1]] - \text{arr}[y + dy[k]][x+dx[k]] - \text{arr}[y + dy[\text{mod}(k,8)+1]][x+dx[\text{mod}(k,8)+1]] - \text{arr}[y + dy[k]][x+dx[k]])]
\]

\[
\text{if } m_1 >= m_2 \text{ then}
\]

\[
\text{bmag} \leftarrow \text{bmag} + \text{multiple2}[k]
\]

\[
\text{ltridpp}[y-1][x-1] \leftarrow \text{bitot1}
\]

\[
\text{ltridpm}[y-1][x-1] \leftarrow \text{bitot2}
\]

\[
\text{ltridpm}[y-1][x-1] \leftarrow \text{bmag}
\]

\[
\text{return } \text{ltridpp, ltridpm, ltridpm}
\]

### 3.6 Local Bit-plane Decoded Pattern

Algorithm 6 shows steps for computing Local Bit-plane Decoded Pattern (LBDP). In line 18, the bit-plane is generated from the neighbor. The loop variable \( k \) represents the index of the neighbor and the loop variable \( l \) represents the 8-bit position. Then the bit-plane value is compared with the center pixel to generate the LBDP feature in line 21.

#### Algorithm 6 Algorithm for Local Bit-plane Decoded Pattern Feature Extraction

\[
\textbf{function} \quad \text{FINDLBDP}(\text{arr, BitDepth})
\]

\[
h, w \leftarrow \text{height and width of array arr}
\]

\[
dx \leftarrow \{1, 1, 0, -1, 1, -1, 0, 1\}
\]

\[
dy \leftarrow \{0, -1, -1, 1, -1, 0, 1\}
\]

\[
\text{multiple2}[1] \leftarrow 1
\]

\[
\text{for } l = 2, \ldots, 8 \text{ do} \quad \triangleright \text{navigate across row}
\]

\[
\text{multiple2}[l] \leftarrow \text{BITSHIFT}(\text{multiple2}[l-1], 1)
\]

\[
\text{for } y = 2, \ldots, h - 1 \text{ do} \quad \triangleright \text{navigate across row}
\]

\[
\text{for } x = 2, \ldots, w - 1 \text{ do} \quad \triangleright \text{navigate across column}
\]

\[
\text{for } l = 1, \ldots, 8 \text{ do}
\]
\[ \text{bitp}[l] \leftarrow 0 \]
\[ \text{btot} \leftarrow 0 \]
\[ \text{for } k = 1, \ldots, 8 \text{ do} \]
\[ \text{cur} \leftarrow \text{arr}[y + dy[k]][x + dx[k]] \]
\[ \text{pow} \leftarrow 1 \]
\[ \text{for } l = 1, \ldots, 8 \text{ do} \]
\[ \text{if } \text{bitget}(\text{cur}, l + \text{BitDepth} - 8) == 1 \text{ then} \]
\[ \text{bitp}[l] \leftarrow \text{bitp}[l] + \text{pow} \]
\[ \text{pow} \leftarrow \text{pow} \times 2 \]
\[ \text{end for} \]
\[ \text{end for} \]
\[ \text{for } l = 1, \ldots, 8 \text{ do} \]
\[ \text{if } \text{bitp}[l] > \text{cur} \text{ then} \]
\[ \text{btot} \leftarrow \text{btot} + \text{multiple2}[l] \]
\[ \text{end if} \]
\[ \text{lbdpFeatures}[y - 1][x - 1] \leftarrow \text{btot} \]
\[ \text{end for} \]
\[ \text{return } \text{lbdpFeatures} \]

### 3.7 Local Neighborhood Difference Pattern

Algorithm 7 shows steps for computing Local Neighborhood Difference Pattern (LNDP). The sign of difference of neighbor with its two adjacent neighbors is calculated in line 20. Based on this sign, the LNDP feature is generated in line 21.

#### Algorithm 7 Algorithm for Local Neighborhood Difference Pattern Feature Extraction

1. \textbf{function} \( f_1(I_p, I_c) \)
2. \textbf{if} \( I_p - I_c \geq 0 \) \textbf{then}
3. \quad \text{out} \leftarrow 1
4. \textbf{else}
5. \quad \text{out} \leftarrow 0
6. \textbf{return} out
7. \textbf{function} \( \text{FindLNDP(arr)} \)
8. \( h, w \leftarrow \text{height and width of array arr} \)
9. \( dx \leftarrow \{1, 1, 0, -1, -1, -1, 0, 1\} \)
10. \( dy \leftarrow \{0, -1, -1, -1, 0, 1, 1, 1\} \)
11. \textit{let multiple2 be an array of size 8 of unsigned byte}
12. \textit{multiple2[1] \leftarrow 1}
13. \textit{for } l = 2, \ldots, 8 \textbf{do}
14. \quad \textit{multiple2}[l] \leftarrow \text{bitshift}(\text{multiple2}[l - 1], 1)
15. \textit{let lndpFeatures be a 2D unsigned byte array of size } h - 2 \times w - 2
16. \textit{for } y = 2, \ldots, h - 1 \textbf{do}
17. \quad \textit{for } x = 2, \ldots, w - 1 \textbf{do}
18. \quad \quad \text{btot} \leftarrow 0
19. \quad \textit{for } k = 1, \ldots, 8 \textbf{do}
20. \quad \quad \textit{if } f_1(\text{arr}[y + dy[mod(k, 8) + 1]][x + dx[mod(k, 8) + 1]], \text{arr}[y + dy[k]][x + dx[k]]) == f_1(\text{arr}[y + dy[mod(k - 2, 8) + 1]][x + dx[mod(k - 2, 8) + 1]], \text{arr}[y + dy[k]][x + dx[k]]) \textbf{ then}
21. \quad \quad \quad \text{btot} \leftarrow \text{btot} + \text{multiple2}[k]
22. \quad \quad \textit{lndpFeatures}[y - 1][x - 1] \leftarrow \text{btot}
23. \quad \textbf{end for}
24. \textbf{return} lndpFeatures

### 3.8 Local Neighborhood Intensity Pattern

Algorithm 8 shows steps for computing Local Neighborhood Intensity Pattern (LNIP). In this feature descriptor, first, the relationship among neighbors is established. Next, XOR of the
obtained relation is done to calculate the feature.

Algorithm 8 Algorithm for Local Neighborhood Intensity Pattern Feature Extraction

```plaintext
1 function FINDLNIP(arr)
2     h, w ← height and width of array arr
3     let lnips be array of size $h - 2 \times w - 2$
4     let lnipm be array of size $h - 2 \times w - 2$
5     a ← 0
6     b ← 0
7     let $b_1, b_2, d, M$ be array of size 8
8     s ← 0
9     for $i = 2, \ldots, h - 1$ do
10        for $j = 2, \ldots, w - 1$ do
11           $t ← 0$
12           $I[1] ← arr[i][j + 1]$
13           $I[2] ← arr[i + 1][j + 1]$
14           $I[3] ← arr[i + 1][j]$
15           $I[4] ← arr[i + 1][j - 1]$
16           $I[5] ← arr[i][j - 1]$
17           $I[6] ← arr[i - 1][j - 1]$
18           $I[7] ← arr[i - 1][j]$
19           $I[8] ← arr[i - 1][j + 1]$
20           $Ic ← arr[i][j]$
21     for $k = 1, \ldots, 8$ do
22        if $k \mod 2 == 0$ then
23           $b_1[k] ← (I[k - 1] \geq I[k]) \ast 2 + (I[\mod(k + 1, 8)] \geq I[k])$
24           $b_2[k] ← (I[k - 1] \geq Ic) \ast 2 + (I[\mod(k + 1, 8)] \geq Ic)$
25           $M[k] ← \text{abs}(I[k - 1] - I[k]) + \text{abs}(I[\mod(k + 1, 8)]) - I[k]$
26        else
27           $in1 ← 1 + (\mod(k + 5, 7))$
28           $in2 ← 1 + (\mod(k + 6, 9))$
29           $in3 ← (\mod(k + 2, 8))$
30           $b_1[k] ← (I[in1] \geq I[k]) \ast 8 + (I[in2] \geq I[k]) \ast 4 + (I[k + 1]) \geq$
31            $I[k]) \ast 2 + (I[in3] \geq I[k])$
32           $b_2[k] ← (I[in1] \geq Ic) \ast 8 + (I[in2] \geq Ic) \ast 4 + (I[k + 1]) \geq$
33            $Ic) \ast 2 + (I[in3] \geq Ic)$
34           $M[k] ← \text{abs}(I[in1] - I[k]) + \text{abs}(I[in2] - I[k]) + \text{abs}(I[k + 1] - I[k]) +$
35            $\text{abs}(I[in3] - I[k])$
36           $d[k] ← \text{xor}(b_1[k], b_2[k])$
37           $count ← 0$
38     while $d[k] > 0$ do
39        if $d[k] \mod 2 == 1$ then
40           $count ← count + 1$
41           $d[k] ← i\text{divide}(d[k], \text{int16}(2))$
42        $x ← 1$
43     if $k \mod 2 == 1$ then
44        $x ← 2$
45     if $count \geq x$ then
46        $\text{lnips}[i - 1][j - 1] ← \text{lnips}[i - 1][j - 1] + \text{pow}(8 - k)$
47        $M[k] ← M[k] / (2 \ast x)$
```

8
\[ t \leftarrow t + \text{abs}(I[k] - Ic) \]
\[ t \leftarrow t/8 \]
\[ \text{for } k = 1, \ldots, 8 \text{ do} \]
\[ \text{lnipm}[i-1][j-1] \leftarrow \text{lnipm}[i-1][j-1] + (M[k] \geq t) \times \text{pow}(2(8 - k)) \]
\[ \text{return lnips, lnipm} \]

3.9 Threshold Local Binary AND Pattern

Algorithm 9 shows steps for determining Threshold Local Binary AND Pattern (TLBAP). This descriptor uses both Threshold Local Pattern and Local Binary Pattern. An input threshold (a value between 0 and 1) is used in this descriptor. First, the maximum among neighbors is determined and then the maximum is multiplied by the threshold. Based on the sign of difference of this with the neighbors, Threshold Local Binary Pattern is generated. Then, the Local Binary Pattern is computed for the center pixel. These two patterns are ANDeD to produce TLBAP.

Algorithm 9 Algorithm for Threshold Local Binary AND Pattern Feature Extraction

1. function FINDTLBAP(arr, th)
2.   \( h, w \leftarrow \text{height and width of array } arr \)
3.   \( dx \leftarrow \{1, 1, 0, -1, -1, 0, 1\} \)
4.   \( dy \leftarrow \{0, -1, -1, 0, 1, 1, 1\} \)
5.   multiple2[1] \leftarrow 1
6.   for \( l = 2, \ldots, 8 \) do
7.     multiple2[\( l \)] \leftarrow \text{bitshift}(\text{multiple2}[\( l - 1 \)], 1)
8.   for \( y = 2, \ldots, h - 1 \) do  \( \triangleright \) navigate across row
9.     for \( x = 2, \ldots, w - 1 \) do  \( \triangleright \) navigate across column
10.    btot \leftarrow 0
11.   centerPx \leftarrow arr[y][x]
12.   maxneigh \leftarrow arr[y + dy[1]][x + dx[1]]
13.   for \( k = 2, \ldots, 8 \) do
14.       neigh \leftarrow arr[y + dy[k]][x + dx[k]]
15.       if \( \text{neigh} > \text{maxneigh} \) then
16.           maxneigh \leftarrow \text{neigh}
17.       thmaxneigh \leftarrow th * \text{maxneigh}
18.   for \( k = 1, \ldots, 8 \) do
19.       neigh \leftarrow arr[y + dy[k]][x + dx[k]]
20.       if \( \text{neigh} > \text{thmaxneigh} \) then
21.           btot \leftarrow btot + multiple2[k]
22.   tlbpFeatures[y-1][x-1] \leftarrow btot
23. end
24. tlbpFeatures \leftarrow \text{FINDLBP}(arr)
25. return tlbpFeatures

3.10 Local Adjacent Neighborhood Average Difference Pattern

Algorithm 10 shows detailed steps in computing the LANADP feature from an image. The array \( arr \) represents the image intensity matrix. The arrays \( dx \) and \( dy \) represents relative offset for accessing each of the 8 neighbors in X and Y direction from a center pixel. Lines 14 and 15 in the algorithm show the computation of average from the adjacent neighbors of a center pixel. In lines 16 and 17, the LANADP feature is computed based on the sign of average adjacent neighbor differences.
Algorithm 10 Algorithm for Local Adjacent Neighborhood Average Difference Pattern Feature Extraction

1  function findLANADP(arr)
2     h, w ← height and width of array arr
3     dx ← {1, 1, 0, −1, −1, −1, 0, 1}
4     dy ← {0, −1, −1, −1, 0, 1, 1, 1}
5     multiple2[1] ← 1
6     for l = 2, . . . , 8 do
7         multiple2(l) ← bitshift(multiple2(l − 1), 1) ⊥ bitshift is a function for shifting bits
8     for y = 2, . . . , h − 1 do
9         for x = 2, . . . , w − 1 do
10            btot ← 0
11            centerPx ← arr[y][x]
12            for k = 1, . . . , 8 do
13                neigh ← arr[y + dy[k]][x + dx[k]] ⊥ fetch the kth neighbor
14                x1k = (arr[y + dy[1 + mod(k, 8)]][x + dx[1 + mod(k, 8)]] + arr[y + dy[1 + mod(k +
15                  1, 8)][x + dx[1 + mod(k + 1, 8)]]]) / 2 − neigh
16                x2k = (arr[y + dy[8 + mod(k − 1, −8)]][x + dx[8 + mod(k − 1, −8)]] + arr[y +
17                  dy[8 + mod(k − 2, −8)][x + dx[8 + mod(k − 2, −8)]]]) / 2 − neigh
18                if (x1k < 0&&x2k < 0)||(x1k >= 0&&x2k >= 0) then
19                    btot ← btot + multiple2[k]
20            end if
21            lanadpFeatures[y − 1][x − 1] ← btot
22        end for
23     end for
24     return lanadpFeatures
25 
4 Results
We have implemented each of the algorithms using MATLAB R2020b. We have tested the output from each of the algorithms for correctness. The obtained output and expected output are shown in the following Table1. The input image matrix of size 5 × 5 is given as

\[
arr = \begin{bmatrix}
4 & 8 & 3 & 11 & 2 \\
9 & 5 & 6 & 0 & 3 \\
1 & 2 & 16 & 5 & 10 \\
3 & 8 & 9 & 2 & 1 \\
6 & 10 & 2 & 3 & 5 \\
\end{bmatrix}
\]

Table 1: Algorithm’s output for the sample input arr

| Algorithm | Expected Result | Obtained Result |
|-----------|-----------------|-----------------|
| LBP       | [149 74 255]    | [149 74 255]    |
|           | [239 0 57]      | [239 0 57]      |
|           | [67 36 254]     | [67 36 254]     |
| CSCoLBP   | A vector of 1024 zeros with the following indexed elements 1. 68,75, 138,156,167,389,404,428,585,655,661, 682,692,874,927,949. | A vector of 1024 zeros with the following indexed elements 1. 68,75, 138,156,167,389,404,428,585,655,661, 682,692,874,927,949. |
| | \[
\begin{bmatrix}
3145728 & 8704 & 8704 \\
16896 & 34 & 34 \\
8704 & 34 & 8704
\end{bmatrix}
\] | \[
\begin{bmatrix}
3145728 & 8704 & 8704 \\
16896 & 34 & 34 \\
8704 & 34 & 8704
\end{bmatrix}
\] |
|---|---|
| LDP | \[
\begin{bmatrix}
506 & 506 & 506 \\
510 & 504 & 506 \\
506 & 504 & 510
\end{bmatrix}
\] | \[
\begin{bmatrix}
506 & 506 & 506 \\
510 & 504 & 506 \\
506 & 504 & 510
\end{bmatrix}
\] |
| LTriDP | \[
LTriDP_p = \begin{bmatrix}
128 & 0 & 136 \\
113 & 162 & 20 \\
128 & 64 & 40
\end{bmatrix}
\] | \[
LTriDP_p = \begin{bmatrix}
128 & 0 & 136 \\
113 & 162 & 20 \\
128 & 64 & 40
\end{bmatrix}
\] |
| | \[
LTriDP_n = \begin{bmatrix}
32 & 72 & 37 \\
2 & 20 & 33 \\
26 & 9 & 2
\end{bmatrix}
\] | \[
LTriDP_n = \begin{bmatrix}
32 & 72 & 37 \\
2 & 20 & 33 \\
26 & 9 & 2
\end{bmatrix}
\] |
| | \[
LTriDm = \begin{bmatrix}
128 & 8 & 173 \\
115 & 255 & 20 \\
156 & 200 & 58
\end{bmatrix}
\] | \[
LTriDm = \begin{bmatrix}
128 & 8 & 173 \\
115 & 255 & 20 \\
156 & 200 & 58
\end{bmatrix}
\] |
| LBDP | \[
\begin{bmatrix}
56 & 71 & 78 \\
162 & 50 & 114 \\
156 & 122 & 124
\end{bmatrix}
\] | \[
\begin{bmatrix}
56 & 71 & 78 \\
162 & 50 & 114 \\
156 & 122 & 124
\end{bmatrix}
\] |
| LNDP | \[
\begin{bmatrix}
190 & 111 & 238 \\
29 & 215 & 149 \\
202 & 237 & 175
\end{bmatrix}
\] | \[
\begin{bmatrix}
190 & 111 & 238 \\
29 & 215 & 149 \\
202 & 237 & 175
\end{bmatrix}
\] |
| LNIP | \[
\begin{bmatrix}
64 & 0 & 80 \\
4 & 85 & 0 \\
5 & 0 & 69
\end{bmatrix}
\] | \[
\begin{bmatrix}
64 & 0 & 80 \\
4 & 85 & 0 \\
5 & 0 & 69
\end{bmatrix}
\] |
| TLBAP | \[
\begin{bmatrix}
128 & 64 & 32 \\
1 & 0 & 16 \\
2 & 4 & 8
\end{bmatrix}
\] | \[
\begin{bmatrix}
128 & 64 & 32 \\
1 & 0 & 16 \\
2 & 4 & 8
\end{bmatrix}
\] |
| LANADP | \[
\begin{bmatrix}
254 & 255 & 127 \\
63 & 246 & 215 \\
206 & 237 & 109
\end{bmatrix}
\] | \[
\begin{bmatrix}
254 & 255 & 127 \\
63 & 246 & 215 \\
206 & 237 & 109
\end{bmatrix}
\] |
5 Conclusion and Future Scope
In the last section, we have presented algorithms for 10 local texture descriptors. We have tested the correctness of these algorithms by implementing them in MATLAB and then running them on a sample image matrix. The same is shown in Table 1.

These different texture descriptors have been formulated over time and no descriptor is good in all cases. Hence we have considered various texture descriptors.

The algorithms can be enhanced and adapted further to run in parallel computing environments like multicore systems, FPGAs, or GPUs. Also, optimization of these algorithms further under the computing environment is an open challenge.

References
[1] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.
[2] Manisha Verma and Balasubramanian Raman. Center symmetric local binary co-occurrence pattern for texture, face and bio-medical image retrieval. *Journal of Visual Communication and Image Representation*, 32:224–236, 2015.
[3] Shiv Ram Dubey, Satish Kumar Singh, and Rajat Kumar Singh. Local diagonal extrema pattern: a new and efficient feature descriptor for CT image retrieval. *IEEE Signal Processing Letters*, 22(9):1215–1219, 2015.
[4] Shiv Ram Dubey, Satish Kumar Singh, and Rajat Kumar Singh. Local wavelet pattern: a new feature descriptor for image retrieval in medical CT databases. *IEEE Transactions on Image Processing*, 24(12):5892–5903, 2015.
[5] Manisha Verma and Balasubramanian Raman. Local tri-directional patterns: A new texture feature descriptor for image retrieval. *Digital Signal Processing*, 51:62–72, 2016.
[6] Shiv Ram Dubey, Satish Kumar Singh, and Rajat Kumar Singh. Local bit-plane decoded pattern: a novel feature descriptor for biomedical image retrieval. *IEEE Journal of Biomedical and Health Informatics*, 20(4):1139–1147, 2015.
[7] Manisha Verma and Balasubramanian Raman. Local neighborhood difference pattern: A new feature descriptor for natural and texture image retrieval. *Multimedia Tools and Applications*, 77(10):11843–11866, 2018.
[8] Prithaj Banerjee, Ayan Kumar Bhunia, Avirup Bhattacharyya, Partha Pratim Roy, and Subrahmanyam Murala. Local neighborhood intensity pattern—a new texture feature descriptor for image retrieval. *Expert Systems with Applications*, 113:100–115, 2018.
[9] Ranjit Biswas, Sudipta Roy, and Debraj Purkayastha. An efficient content-based medical image indexing and retrieval using local texture feature descriptors. *International Journal of Multimedia Information Retrieval*, 8(4):217–231, 2019.