LBP 推导

Soul Walker

2020 年 7 月 22 日

0.1 原始 LBP

对于原始 LBP 算子定义在 3×3 的窗口内，窗口中心元素作为阈值，与其相邻的 8 个像素的灰度值进行比较，若周围的像素值大于中心的像素值，则该位置标记为 1，否则标记为 0（这里有体现 LBP 特征提取的一个优点，即 LBP 记录的是中心像素点与其周围像素点的差值标记，所以光照引起的所有像素灰度值同增同减，并不会导致 LBP 产生很大的变换）。那么这样的话，以左上角元素为其实位置，顺时针旋转记录这 8 个位置的被标记的值，可以得到 8 位的二进制数，那么这个 8 位二进制数有 2^8 种，这个数值就是 LBP 码（一般会转换为十进制），这个数值作为这个 3×3 窗口的中心像素点的 LBP 值。我们通过使用直方图对这 256 种 LBP 码可以做出统计。那么就可以形成一个有 256 个块的直方图，也就是说 256 维的直方图可以直接作为特征向量。

0.1.1 公式

$$LBP(x_c, y_c) = \sum_{p=1}^{P-1} sign(I(p) - I(c)) * 2^p \tag{0.1}$$

$$s(x) = \begin{cases} 1, x \geq 0 \\ 0, otherwise \end{cases} \tag{0.2}$$

其中，p 表示 3×3 窗口中除了中心像素点外的第 p 个像素点，$I(c)$ 表示中心像素点的灰度值，$I(p)$ 表示领域内第 p 个像素点的灰度值。

0.2 圆形 LBP 算子

对于上述原始 LBP 算子的扩展与补充（主要是适应不同尺度，以及旋转不变性要求），那么改进之后的 LBP 算法可以允许在半径为 R 的圆形区域内有任意多的像素点，那么我们就可以得到以半径为 R 的圆形区域内有 P 个采样点的 LBP 算子

0.2.1 公式

$$LBP_{P,R}(x_c, y_c) = \sum_{p=1}^{P} sign(I(p) - I(c)) * 2^P \tag{0.3}$$

$$\begin{cases} x_p = x_c + R * cos(\frac{2\pi p}{P}) \\ y_p = y_c - R * sin(\frac{2\pi p}{P}) \end{cases} \tag{0.4}$$

$$sign(x) = \begin{cases} 1, x \geq 0 \\ 0, otherwise \end{cases} \tag{0.5}$$

$$sign(x) = \begin{cases} 1, x \geq 0 \\ 0, otherwise \end{cases} \tag{0.6}$$

其中，p 表示圆形区域中总计 P 个采样点中的第 p 个采样点，$I(c)$ 表示中心像素的灰度值，$I(p)$ 表示圆形边界像素点中第 p 个点的灰度值。
在实现中还有一点需要解释的是圆形 LBP 特征中非整数位置灰度值计算的双线性插值方法。双线性插值其实就是非整数点的像素等于周围像素点的加权平均，而我们的目的就是求出权值及周围像素点的位置

以 $P=8, R=1$ 为例有如下步骤:
首先，假设中心点位置为 (i,j)，那么，左上角位置:

$$\begin{aligned} x &= i + R \times cos(\frac{2\pi p}{P}) \\ y &= j - R \times sin(\frac{2\pi p}{P}) \end{aligned} \tag{0.7}$$

然后,我们计算需要进行加权平均的四个点的位置 $(x_1,y_1),(x_2,y_2),(x_1,y_2),(x_2,y_1)$

$$\begin{aligned} x_1 &= floor(x) \\ y_1 &= floor(y) \\ x_2 &= ceil(x) \\ y_2 &= ceil(y) \end{aligned} \tag{0.8}$$

其中，$floor$ 为向下取整，$ceil$ 为向上取整。
接下来，我们将坐标映射到 0 1 之间:

$$\begin{aligned} cx &= x - x_1 \\ cy &= y - y_1 \end{aligned} \tag{0.9}$$

然后，计算权重:

$$\begin{aligned} w_1 &= (1-cx) \times (1-cy) \\ w_2 &= cx \times (1-cy) \\ w_3 &= (1-cx) \times cy \\ w_4 &= cx \times cy \end{aligned} \tag{0.10}$$

最后，求非整数位置的灰度值:

$$res = img(x_1,y_1) \times w_1 + img(x_1,y_2) \times w_2 + img(x_2,y_1) \times w_3 + img(x_2,y_2) \times w_4 \tag{0.11}$$

0.3 代码实现

```
import numpy as np
import math


def CircularLocalBinaryPattern(img, P, R):
    """
    This is a implement of Curcular Local Binary Pattern
    Args:
        img: The image we want to process
        P: The number of sample points
        R: The radius of the sampled circular area

    Returns:
        The image after being processed.
    """
    height, width = img.shape
    img.astype(dtype=np.float32)
    res = img.copy()

    # To contain the nearest point of the center point
    neighbours = np.zeros((1, P), dtype=np.uint8)
    # To contain the value of LBP formula
    LBPValue = np.zeros((1, P), dtype=np.uint8)
    for x in range(R, width - R - 1):
        for y in range(R, height - R - 1):
            LBP = 0.
            # First calculate the pixel values corresponding to a
                total of P points, using bilinear interpolation
            for n in range(P):
                theta = float(2 * np.pi * n) / P
                xn = x + R * np.cos(theta)
                yn = y - R * np.sin(theta)

                # Round down
```

```
            x1 = int(math.floor(xn))
            y1 = int(math.floor(yn))

            # Round up
            x2 = int(math.ceil(xn))
            y2 = int(math.ceil(yn))

            # Map coordinates between 0 and 1
            cx = np.abs(x - x1)
            cy = np.abs(y - y1)

            # The weights of Bilinear interpolation
            w1 = (1 - cx) * (1 - cy)
            w2 = cx * (1 - cy)
            w3 = (1 - cx) * cy
            w4 = cx * cy

            # Calculate the gray value of the n-th sampling point
                according to the bilinear interpolation formula
            neighbour = img[y1, x1] * w1 + img[y2, x1] * w2 +
                img[y1, x2] * w3 + img[y2, x2] * w4
            neighbours[0, n] = neighbour

        center = img[y, x]

        for n in range(P):
            if neighbours[0, n] > center:
                LBPValue[0, n] = 1
            else:
                LBPValue[0, n] = 0

        for n in range(P):
            LBP += LBPValue[0, n] * 2 ** n
```

```
            # Map gray space to 0-255
            res[y, x] = int(LBP / (2 ** P - 1) * 255)

    return res
```