原 LIRe 源代码分析 7：算法类[以颜色布局为例]

2013年11月02日 22:10:31 阅读数：4832

==========

LIRe源代码分析系列文章列表：

LIRe 源代码分析 1：整体结构

LIRe 源代码分析 2：基本接口（DocumentBuilder）

LIRe 源代码分析 3：基本接口（ImageSearcher）

LIRe 源代码分析 4：建立索引（DocumentBuilder）[以颜色布局为例]

LIRe 源代码分析 5：提取特征向量[以颜色布局为例]

LIRe 源代码分析 6：检索（ImageSearcher）[以颜色布局为例]

LIRe 源代码分析 7：算法类[以颜色布局为例]

==========

前面关于LIRe的文章，介绍的都是架构方面的东西，没有细研究具体算法。本文以颜色布局为例，介绍一下算法类的实现。

颜色布局描述符以一种非常紧密的形式有效地表示了图像的颜色空间分布信息。它以非常小的计算代价，带来高的检索效率。因此，颜色布局特征在视频镜头关键帧提取中有很重要的意义。颜色布局提取方法如下:

1 将图像从RGB 空间映射到YCbCr空间, 映射公式为

Y= 0.299* R + 0.587* G + 0.114* B

Cb= - 0.169* R – 0.331* G + 0.500* B

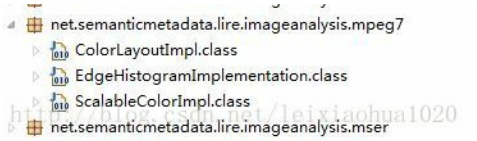
Cr = 0.500* R –0.419* G – 0.081* B

2 将整幅图像分成64块, 每块尺寸为(W /8) *(H/8), 其中W 为整幅图像的宽度, H 为整幅图像的高度, 计算每一块中所有像素的各个颜色分量(Y, Cb, Cr)的平均值, 并以此作为该块的代表颜色(Y, Cb, Cr);

3 对帧图像中各块的颜色分量平均值进行DCT 变换, 得到各分量的一系列DCT 系数;

4 对各分量的DCT 系数, 通过之字形扫描和量化, 取出各自DCT 变换的低频分量, 这三组低频分量共同构成该帧图像的颜色布局描述符。

颜色布局算法的实现位于ColorLayoutImpl类中，该类处于“net.semanticmetadata.lire.imageanalysis.mpeg7”包中，如图所示：



ColorLayoutImpl类的代码量很大，很多地方都还没有研究，在这里仅展示部分已经看过的代码：

```java
/* 雷霄骅
 * 中国传媒大学/数字电视技术
 * leixiaohua1020@126.com
 *
 */
/**
 * Class for extrcating & comparing MPEG-7 based CBIR descriptor ColorLayout
 *
 * @author Mathias Lux, mathias@juggle.at
 */
public class ColorLayoutImpl {
    // static final boolean debug = true;
    protected int[][] shape;
    protected int imgYSize, imgXSize;
    protected BufferedImage img;

    protected static int[] availableCoeffNumbers = {1, 3, 6, 10, 15, 21, 28, 64};
    //特征向量 (Y,Cb,Cr)
    public int[] YCoeff;
```

```
    public int[] CbCoeff;
    public int[] CrCoeff;
    //特征向量的大小
    protected int numCCoeff = 28, numYCoeff = 64;

    protected static int[] arrayZigZag = {
            0, 1, 8, 16, 9, 2, 3, 10, 17, 24, 32, 25, 18, 11, 4, 5,
            12, 19, 26, 33, 40, 48, 41, 34, 27, 20, 13, 6, 7, 14, 21, 28,
            35, 42, 49, 56, 57, 50, 43, 36, 29, 22, 15, 23, 30, 37, 44, 51,
            58, 59, 52, 45, 38, 31, 39, 46, 53, 60, 61, 54, 47, 55, 62, 63
    };

    protected static double[][] arrayCosin = {
    ...
    };
    protected static int[][] weightMatrix = new int[3][64];
    protected BufferedImage colorLayoutImage;


    /**
     * init used by all constructors
     */
    private void init() {
        shape = new int[3][64];
        YCoeff = new int[64];
        CbCoeff = new int[64];
        CrCoeff = new int[64];
        colorLayoutImage = null;
        extract();
    }

    public void extract(BufferedImage bimg) {
        this.img = bimg;
        imgYSize = img.getHeight();
        imgXSize = img.getWidth();
        init();
    }

    private void createShape() {
        int y_axis, x_axis;
        int i, k, x, y, j;
        long[][] sum = new long[3][64];
        int[] cnt = new int[64];
        double yy = 0.0;
        int R, G, B;

        //init of the blocks
        for (i = 0; i < 64; i++) {
            cnt[i] = 0;
            sum[0][i] = 0;
            sum[1][i] = 0;
            sum[2][i] = 0;
            shape[0][i] = 0;
            shape[1][i] = 0;
            shape[2][i] = 0;
        }

        WritableRaster raster = img.getRaster();
        int[] pixel = {0, 0, 0};
        for (y = 0; y < imgYSize; y++) {
            for (x = 0; x < imgXSize; x++) {
                raster.getPixel(x, y, pixel);
                R = pixel[0];
                G = pixel[1];
                B = pixel[2];

                y_axis = (int) (y / (imgYSize / 8.0));
                x_axis = (int) (x / (imgXSize / 8.0));

                k = (y_axis << 3) + x_axis;

                //RGB to YCbCr, partition and average-calculation
                yy = (0.299 * R + 0.587 * G + 0.114 * B) / 256.0;
                sum[0][k] += (int) (219.0 * yy + 16.5); // Y
                sum[1][k] += (int) (224.0 * 0.564 * (B / 256.0 * 1.0 - yy) + 128.5); // Cb
                sum[2][k] += (int) (224.0 * 0.713 * (R / 256.0 * 1.0 - yy) + 128.5); // Cr
                cnt[k]++;
            }
        }

        for (i = 0; i < 8; i++) {
            for (j = 0; j < 8; j++) {
                for (k = 0; k < 3; k++) {
                    if (cnt[(i << 3) + j] != 0)
                        shape[k][(i << 3) + j] = (int) (sum[k][(i << 3) + j] / cnt[(i << 3) + j]);
                    else
                        shape[k][(i << 3) + j] = 0;
                }
            }
        }
    }
```

```
111.
112.     ......（其他代码都已经省略）
113.
114.
115.
116.     private int extract() {
117.
118.         createShape();
119.
120.         Fdct(shape[0]);
121.         Fdct(shape[1]);
122.         Fdct(shape[2]);
123.
124.         YCoeff[0] = quant_ydc(shape[0][0] >> 3) >> 1;
125.         CbCoeff[0] = quant_cdc(shape[1][0] >> 3);
126.         CrCoeff[0] = quant_cdc(shape[2][0] >> 3);
127.
128.         //quantization and zig-zagging
129.         for (int i = 1; i < 64; i++) {
130.             YCoeff[i] = quant_ac((shape[0][(arrayZigZag[i])]) >> 1) >> 3;
131.             CbCoeff[i] = quant_ac(shape[1][(arrayZigZag[i])]) >> 3;
132.             CrCoeff[i] = quant_ac(shape[2][(arrayZigZag[i])]) >> 3;
133.         }
134.
135.         setYCoeff(YCoeff);
136.         setCbCoeff(CbCoeff);
137.         setCrCoeff(CrCoeff);
138.         return 0;
139.     }
140.
141.     /**
142.      * Takes two ColorLayout Coeff sets and calculates similarity.
143.      *
144.      * @return -1.0 if data is not valid.
145.      */
146.     public static double getSimilarity(int[] YCoeff1, int[] CbCoeff1, int[] CrCoeff1, int[] YCoeff2, int[] CbCoeff2, int[] CrCoeff2)
    {
147.         int numYCoeff1, numYCoeff2, CCoeff1, CCoeff2, YCoeff, CCoeff;
148.
149.         //Numbers of the Coefficients of two descriptor values.
150.         numYCoeff1 = YCoeff1.length;
151.         numYCoeff2 = YCoeff2.length;
152.         CCoeff1 = CbCoeff1.length;
153.         CCoeff2 = CbCoeff2.length;
154.
155.         //take the minimal Coeff-number
156.         YCoeff = Math.min(numYCoeff1, numYCoeff2);
157.         CCoeff = Math.min(CCoeff1, CCoeff2);
158.
159.         setWeightingValues();
160.
161.         int j;
162.         int[] sum = new int[3];
163.         int diff;
164.         sum[0] = 0;
165.
166.         for (j = 0; j < YCoeff; j++) {
167.             diff = (YCoeff1[j] - YCoeff2[j]);
168.             sum[0] += (weightMatrix[0][j] * diff * diff);
169.         }
170.
171.         sum[1] = 0;
172.         for (j = 0; j < CCoeff; j++) {
173.             diff = (CbCoeff1[j] - CbCoeff2[j]);
174.             sum[1] += (weightMatrix[1][j] * diff * diff);
175.         }
176.
177.         sum[2] = 0;
178.         for (j = 0; j < CCoeff; j++) {
179.             diff = (CrCoeff1[j] - CrCoeff2[j]);
180.             sum[2] += (weightMatrix[2][j] * diff * diff);
181.         }
182.
183.         //returns the distance between the two desciptor values
184.
185.         return Math.sqrt(sum[0] * 1.0) + Math.sqrt(sum[1] * 1.0) + Math.sqrt(sum[2] * 1.0);
186.     }
187.
188.
189.
190.
191.     public int getNumberOfCCoeff() {
192.         return numCCoeff;
193.     }
194.
195.     public void setNumberOfCCoeff(int numberOfCCoeff) {
196.         this.numCCoeff = numberOfCCoeff;
197.     }
198.
199.     public int getNumberOfYCoeff() {
200.         return numYCoeff;
```

```
    }

    public void setNumberOfYCoeff(int numberOfYCoeff) {
        this.numYCoeff = numberOfYCoeff;
    }


    public String getStringRepresentation() {
        StringBuilder sb = new StringBuilder(256);
        StringBuilder sbtmp = new StringBuilder(256);
        for (int i = 0; i < numYCoeff; i++) {
            sb.append(YCoeff[i]);
            if (i + 1 < numYCoeff) sb.append(' ');
        }
        sb.append("z");
        for (int i = 0; i < numCCoeff; i++) {
            sb.append(CbCoeff[i]);
            if (i + 1 < numCCoeff) sb.append(' ');
            sbtmp.append(CrCoeff[i]);
            if (i + 1 < numCCoeff) sbtmp.append(' ');
        }
        sb.append("z");
        sb.append(sbtmp);
        return sb.toString();
    }

    public void setStringRepresentation(String descriptor) {
        String[] coeffs = descriptor.split("z");
        String[] y = coeffs[0].split(" ");
        String[] cb = coeffs[1].split(" ");
        String[] cr = coeffs[2].split(" ");

        numYCoeff = y.length;
        numCCoeff = Math.min(cb.length, cr.length);

        YCoeff = new int[numYCoeff];
        CbCoeff = new int[numCCoeff];
        CrCoeff = new int[numCCoeff];

        for (int i = 0; i < numYCoeff; i++) {
            YCoeff[i] = Integer.parseInt(y[i]);
        }
        for (int i = 0; i < numCCoeff; i++) {
            CbCoeff[i] = Integer.parseInt(cb[i]);
            CrCoeff[i] = Integer.parseInt(cr[i]);

        }
    }

    public int[] getYCoeff() {
        return YCoeff;
    }

    public int[] getCbCoeff() {
        return CbCoeff;
    }

    public int[] getCrCoeff() {
        return CrCoeff;
    }
}
```

下面介绍几个主要的函数：

提取：

1.extract(BufferedImage bimg)：提取特征向量的函数，里面调用了init()。

2.init()：初始化了 YCoeff，CbCoeff， CrCoeff。调用extract()（注意这个extract()是没有参数的）

3.extract()：完成了提取特征向量的过程，其中调用了createShape()。

4.createShape()：未研究。

获取/设置特征向量（注意：有参数为String和byte[]两种类型的特征向量，按照原代码里的说法，byte[]的效率要高一些）：

1.getStringRepresentation()：获取特征向量

2.setStringRepresentation()：设置特征向量

计算相似度：

getSimilarity(int[] YCoeff1, int[] CbCoeff1, int[] CrCoeff1, int[] YCoeff2, int[] CbCoeff2, int[] CrCoeff2)

主要的变量：

3个存储特征向量（Y,Cb,Cr）的数组：

```java
public int[] YCoeff;
public int[] CbCoeff;
public int[] CrCoeff;
```

特征向量的大小：

```java
protected int numCCoeff = 28, numYCoeff = 64;
```


文章标签： lire 源代码 索引 检索 lucene

个人分类： MPEG7/图像检索 LIRe

所属专栏： 开源多媒体项目源代码分析