1. **题目**

基于内容的图像检索

1. **摘要**

本文实现了基于内容的图像检索，具体使用的数据集prid2011数据集，这个数据集包括2个监控摄像头下大量的行人图片

A摄像头：      

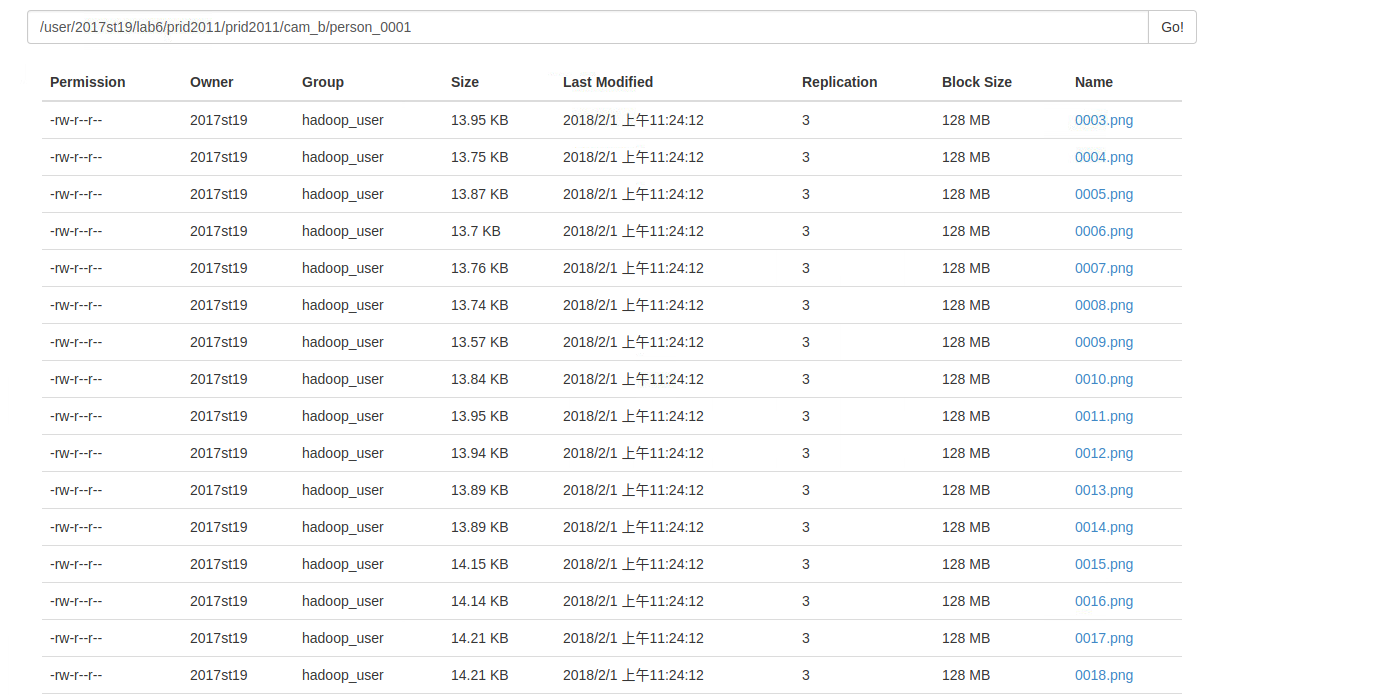
B摄像头：      

任务是给定A摄像头的某个行人，找到B摄像头里的同一个人，即最佳匹配问题。由上图可见，不同摄像头下同一个行人的差别是很大的，目前用的较多的深度学习方法最好的Rank1准确率大概在80%左右。

因此寻找一个合适的特征至关重要，本文选取的是CVPR2015的一篇文章提出的LOMO特征[3]，由于图像数量巨大，使用单机需要非常久的时间，因此本文通过mapreduce来并行化地对一个图像提取LOMO特征。检索算法选取的是K近邻。

1. **技术难点**
   1. **图片存储**

HDFS是以数据块的形式分布式存储数据，每个数据块默认为 64M，NameNode 节点负责将数据块存储到 DataNode 节点，DataNode 节点负责管理数据在本节点的存储。本文研究的图像数据集中的图像数据，单个图像的大小都为几十 KB 或者几百 KB，图像数据的数量却比较大。如果按照传统的存储方式，即每幅图为一个数据块，虽然简单易实现，但是由于每幅图像不超过几百KB，必将造成 HDFS 处理小文件的问题，如下图所示：



问题如下：

(1) HDFS 由 NameNode 以对象的形式将文件保存到内存中，每个文件都有对象与之对应，众多的小文件必将产生大量的对象，大量的对象必将对 NameNode 的内存带来更大的压力，降低 HDFS 的可扩张性；

(2) MapReduce 模型是以文件的分片为基础，每个分片都有 Map 任务与之对应，每个小文件都会分成若干个分片，过多的小文件将会产生数量巨大的分片，随之将会有数量巨大的 Map 任务与之对应，最终导致 MapReduce 的任务分配机制频繁的调动，使得整个系统的性能降低。

* 1. **图片****特征提取**

在基于内容的图像检索中，是计算图像特征之间的相似度来完成检索的，所有首先必须提取后台大量的图像数据的视觉内容特征。

1. **解决方法和设计思路**
   1. **图片存储**

针对数据集的文件小和数量多的特性，结合 Hadoop 自身处理大文件的优势，根据 HDFS 解决小文本文件时的提出的顺序文件思想，将图像集中的小文件合并成为一个大的逻辑文件，该文件包含图像名称信息和图像内容信息，作为 HDFS的一个输入。

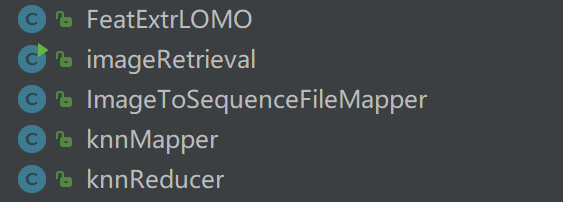
* 1. **图片特征提取**

本文实现了CVPR2015的一篇文章中提出的LOMO特征[3]，结合了HSV和SILTP特征[2]，对光照变化、视角变化等具有一定的鲁棒性。在第7部分详细阐述。

1. **设计说明**

本文主要包括5个程序：

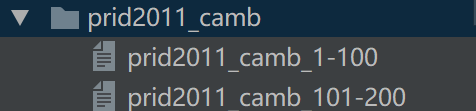
imageRetrieval是驱动程序，ImageToSequenceFileMapper负责图像存储，FeatExtrLOMO负责图像特征提取，knnMapper和knnReducer负责图像检索。



* 1. **图片存储(****ImageToSequenceFileMapper.java)**

顺序文件提供了二进制 key/value 对的存储模式，可以作为小文件的容器。Key为图像名，value为图像像素值。

|  |  |
| --- | --- |
| 输入 | 包含图像数据目录信息的多个文件 |
| 输出 | Key：文件名，Value：图像像素值 |
| Mapper | For 该文件中所有的图像路径  读取图像内容  输出<key,value>对，key为文件名(Text类型)，value为图像内容(BytesWritable类型) |
| Reducer | 无 |

例如，输入参数为”prid2011\_camb”，此文件夹的构成：  


其包含两个文件：prid2011\_camb\_1-100、prid2011\_camb\_101-200。

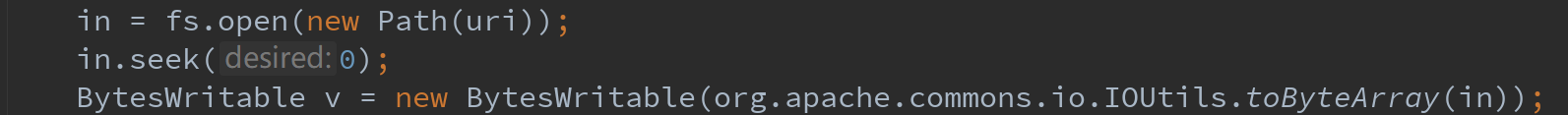
prid2011\_camb\_1-100文件的部分内容如下：

  
每一行都是图像文件所在路径的信息。

prid2011\_camb\_101-200类似

。

读取图像文件的核心代码如下：



其中，v是即将发射给reducer的value。

最终ImageToSequenceFileMapper.java输出的结果是一个顺序文件：

|  |  |
| --- | --- |
| Key(Text类型) | Value(BytesWritable类型) |
| C:/prid2011/cam\_b/person\_0001/0003.png | 图像像素值 |
| C:/prid2011/cam\_b/person\_0002/0014.png | 图像像素值 |
| C:/prid2011/cam\_b/person\_0003/0017.png | 图像像素值 |
| …… | …… |

* 1. **图片特征提取(FeatExtrLOMO.java)**

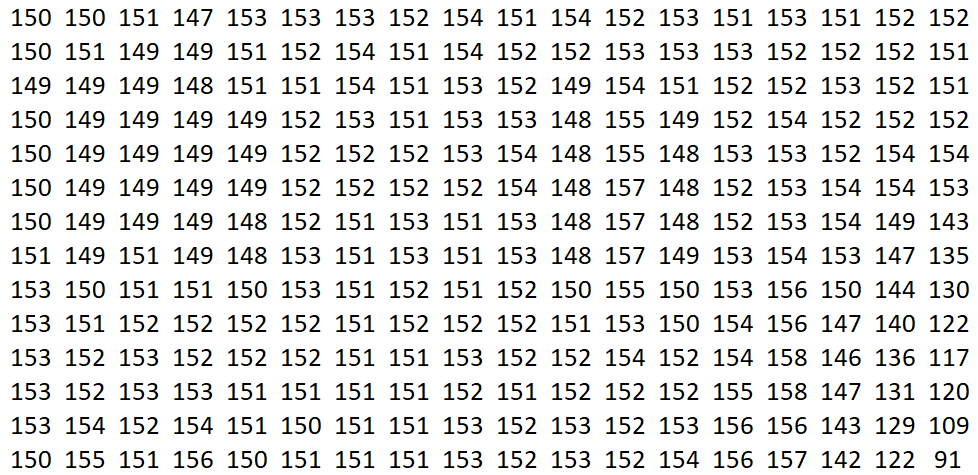
图像特征表述了图像的内容，作为检索的匹配对象，特征的质量至关重要，高质量的特征无疑会提高检索的准确率，所以图像特征提取模块的重要性显而易见。由于

本文中使用的是监控摄像头下的行人图像，因此使用了目前行人检索中比较流行的一种特征：LOMO(Local Maximal Occurrence)特征，这是2015年CVPR上的一篇文章。文章中提取了HSV颜色直方图和SILTP描述子，并且分别基于这两种特征提取LOMO特征。

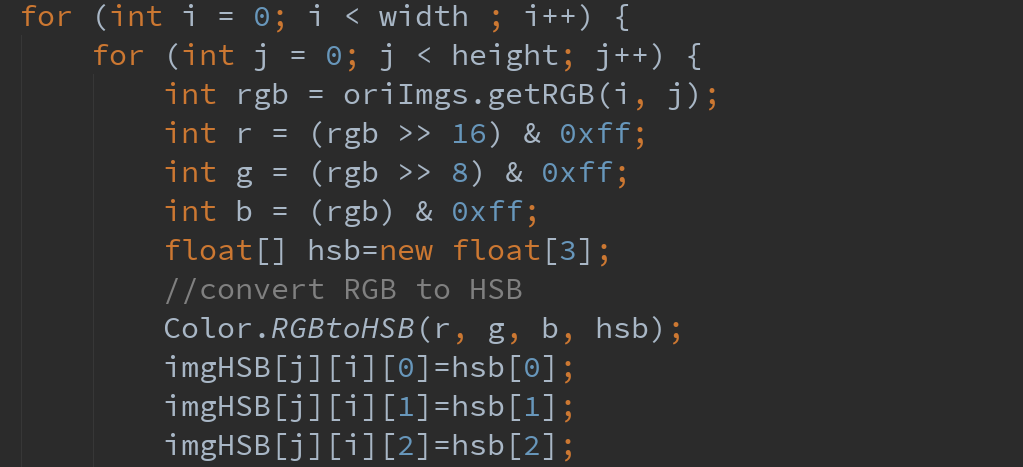
FeatExtrLOMO.java实现了LOMO特征提取，代码量约600行，是此实验的关键点。

**8.2.1基于联合HSV的LOMO特征(**FeatExtrLOMO.java/ PyramidMaxJointHist**)**

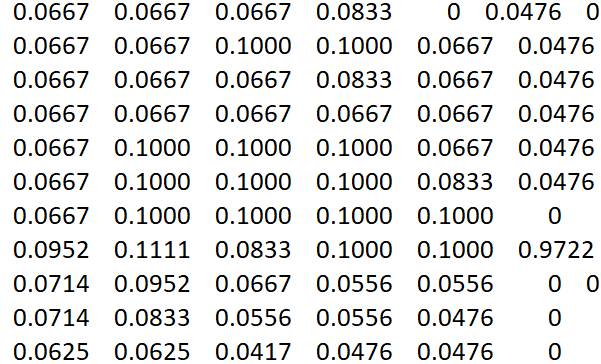
部分原始图像RGB值如下：



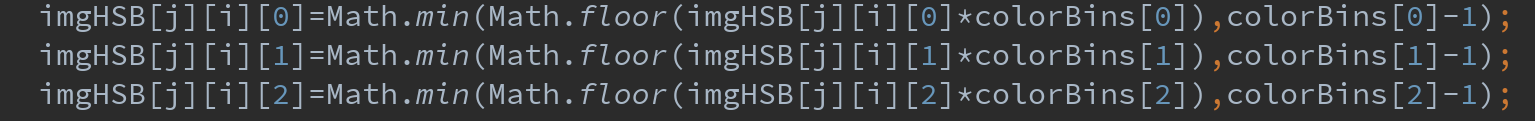
利用JAVA API中的RGBtoHSB函数把RGB空间转化到HSV空间：



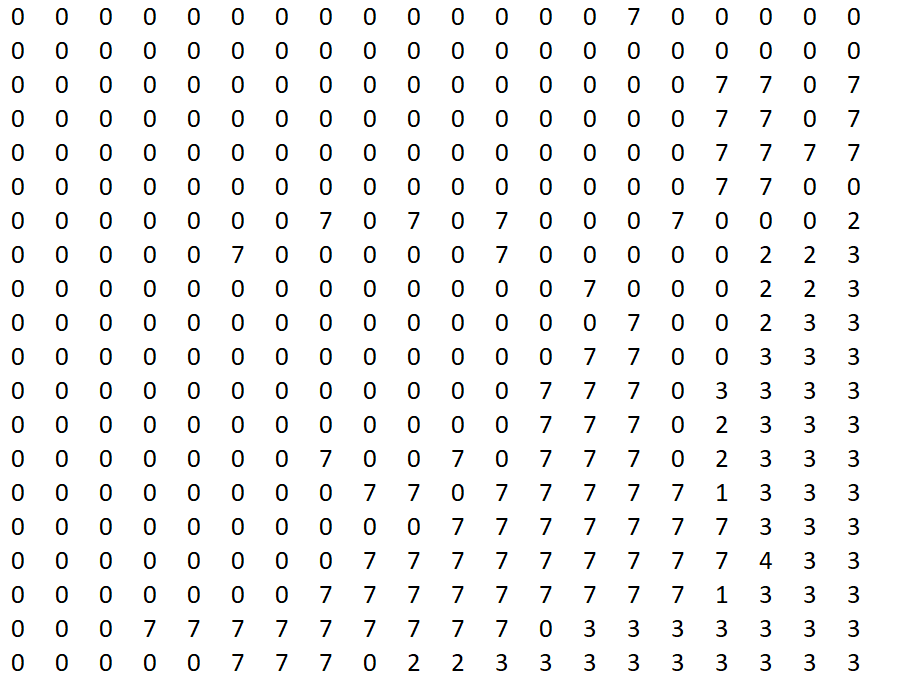
部分HSV值如下：



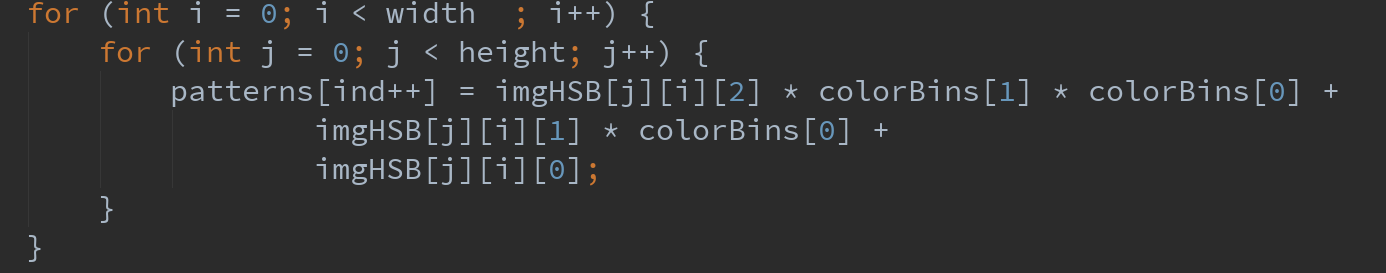
计算颜色直方图需要将颜色空间划分成若干个小的颜色区间，每个小区间成为直方图的一个bin。这个过程称为颜色量化。本实验中直方图的bins设置为[8,8,8]。以下是量化过程：



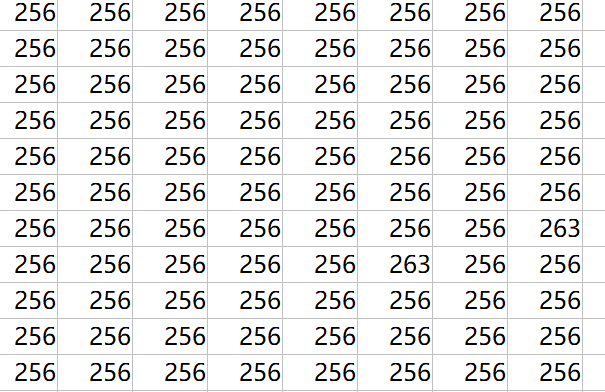
产生的部分结果如下：



对HSV三通道进行编码，计算联合值，计算公式是V\*8\*8+S\*8+H：



部分结果如下所示：



然后，使用一个10\*10的窗口来扫描图片（128\*64），如下图所示，其步长为5。

10\*1064

128

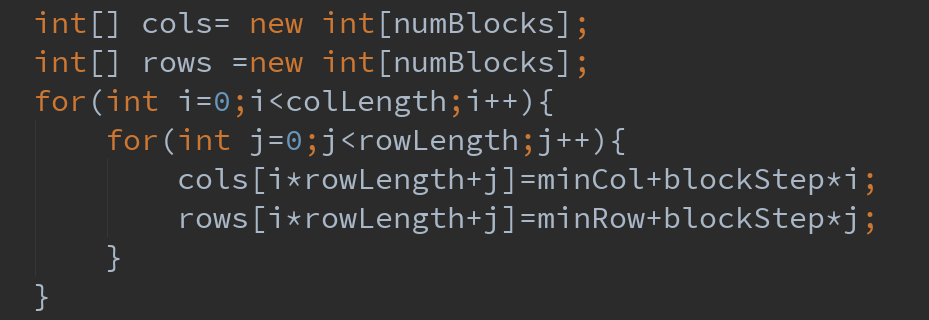
列数为64，步长为5，则扫描的列起始位置共11个（假设从1开始）：

[ 1 6 11 16 21 26 31 36 41 46 51]

行数为128，步长为5，则扫描的行起始位置共24个：

[ 1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116]

接下来，将11个列编号复制24次，将24个行编号复制11次。



24\*11=264，因此各得到一个264维的向量：

列cols：

[ 1 1 … 1 6 6 ... 6 ……. 51 51 … 51 ]

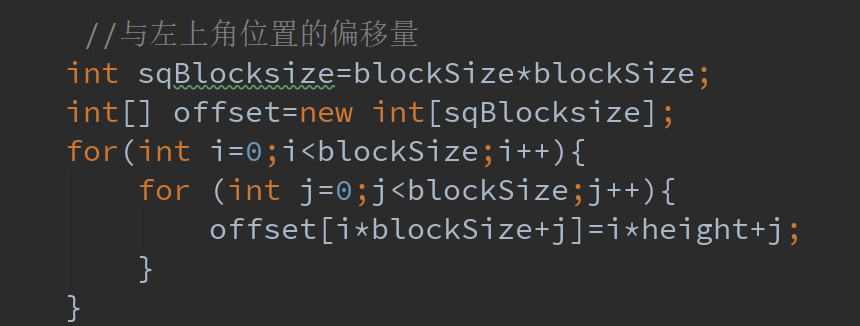
24个

行rows：

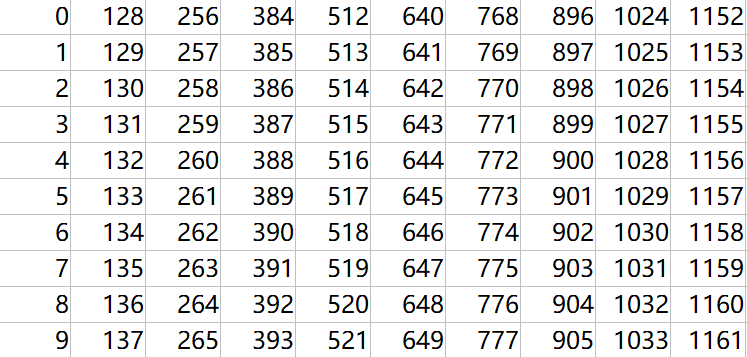
[ 1 6 11 … 116 1 6 11 … 116 1 6 11 … 116 ]

11个

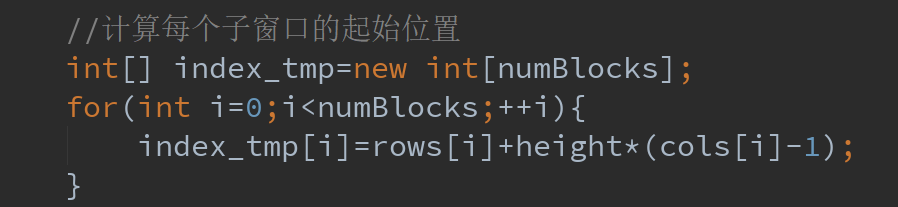
窗口大小为10，计算与左上角位置的偏移量offset(100维)：



offset的值如下：

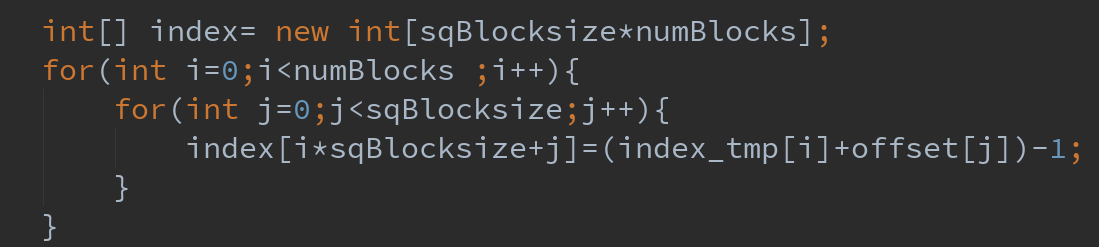


把原始128\*64维度按列展开，再计算每个子窗口的起始位置()：



部分结果如下：

[ 1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 641 646 651 656 661 666 … … ]

然后计算展开之后的每个子窗口的所有点的坐标值：  


得到一个26400的向量，每100个值都代表一个子窗口(10\*10)的100个坐标值,部分结果如下：

[ 1 2 3 4 5 6 7 8 9 10 129 130 131 132 133 134 135 136 137 138 257 … …. ]

坐标计算顺序如下，先扫描①方向，再扫描②方向：

②

2

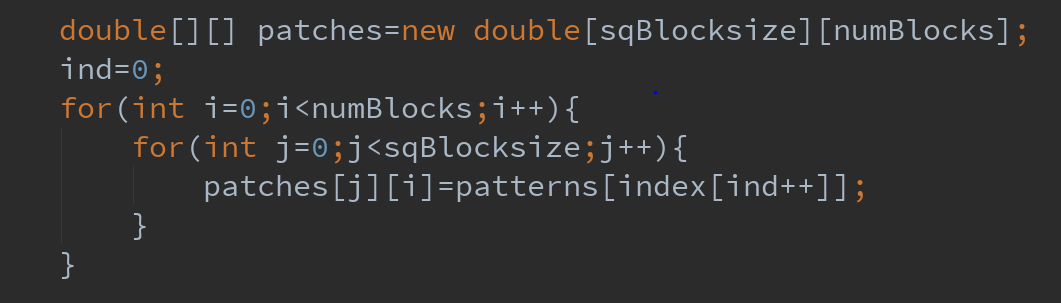
1

3

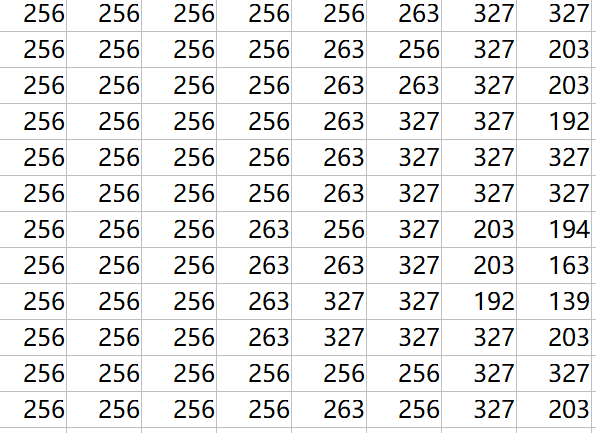
4

①

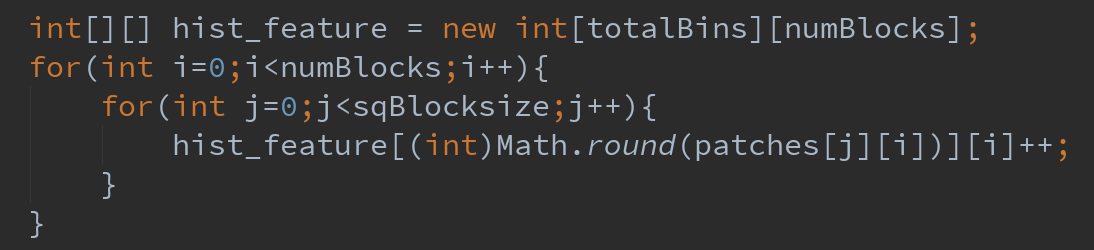
根据以上计算出来的坐标，索引之前计算的联合HSV值，得到每一个子窗口的联合HSV值，得到维度为100\*264的矩阵：



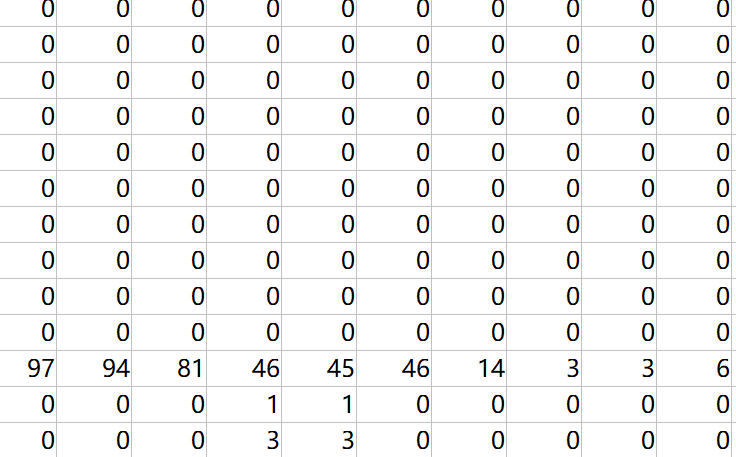
部分结果如下：



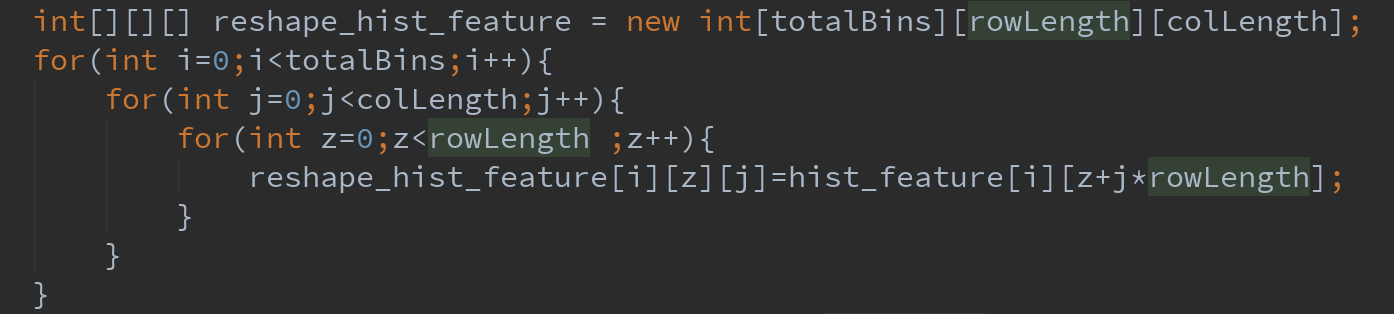
计算每个子窗口的直方图，比特位共计8\*8\*8=512，



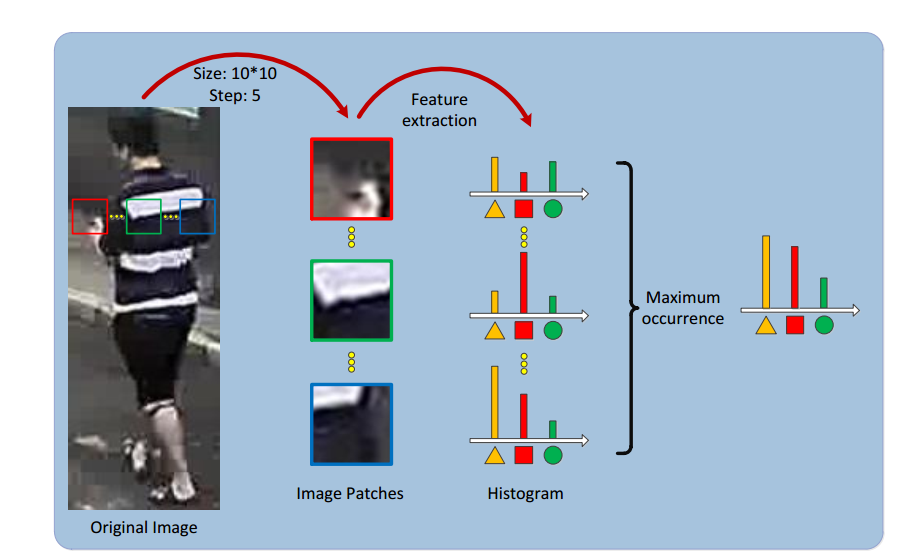
得到的hist\_feature的维度是512\*264,512表示比特位，264表示264个子窗口，部分值如下图所示。其中大部分比特位都是0值。



重新排列512\*264维度的hist\_feature，使其维度变为512\*24\*11：



下面是这篇文章的重点部分：LOMO（Local Maximal Occurrence），其思想一图以蔽之：



计算每一行(即水平方向)的最大hist值，用此值取代这一行的值：

第1行

Max(row1)

第2行

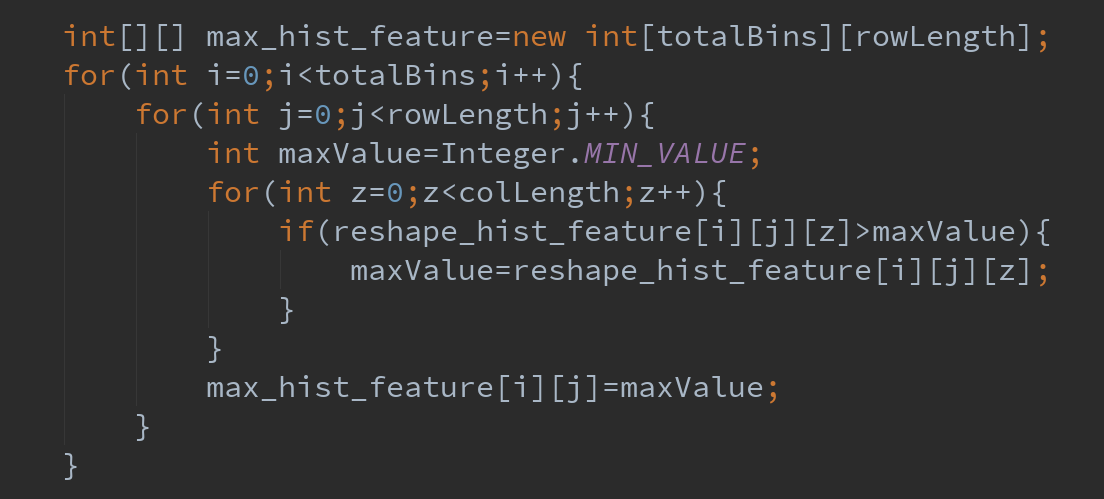
Max(row2)

Max(row3)

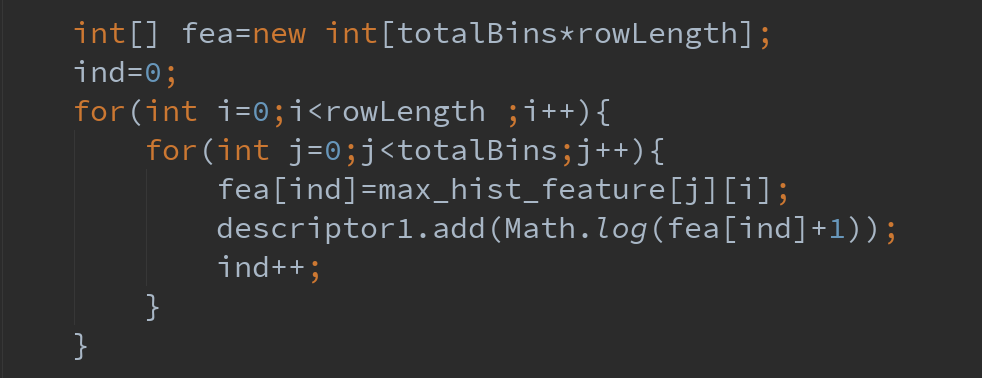
第3行

……

产生一个512\*24维度的矩阵，矩阵的每一列是图像每一行最大的hist值，代码如下：



然后将其展开成一个512\*24=12288的向量，作为HSV直方图特征的一部分：



进一步考虑多尺度信息，建立一个三层金字塔，使用2\*2的平均池化操作对原始128\*64的图像进行下采样，再重复2次上述所述的所有操作。此部分几乎全是张量的变换，不再赘述，详见FeatExtrLOMO.java/ PyramidMaxJointHist结尾部分。

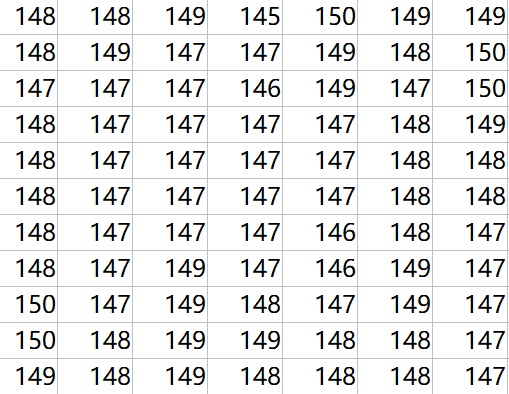
**8.2.2基于SILTP的LOMO特征(**FeatExtrLOMO.java/ PyramidMaxSILTPHist**)**

SILTP(Scale Invariant Local Patterns)是CVPR2010的一篇文章，下面会详细说明。

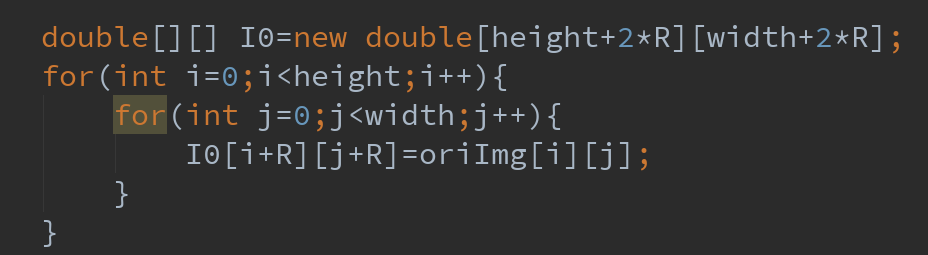
首先，不再将RGB空间转化到HSV空间，而是转化为灰度值：

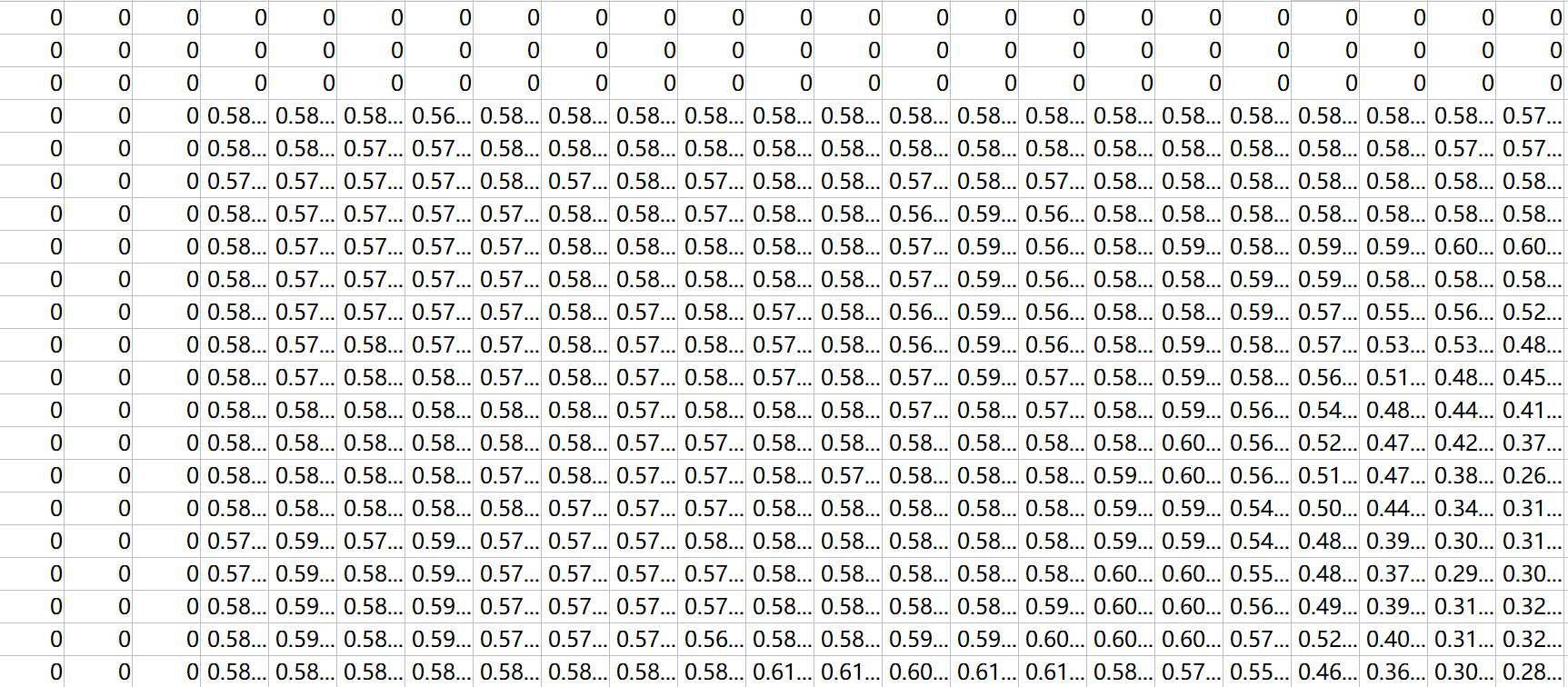


部分结果如下：

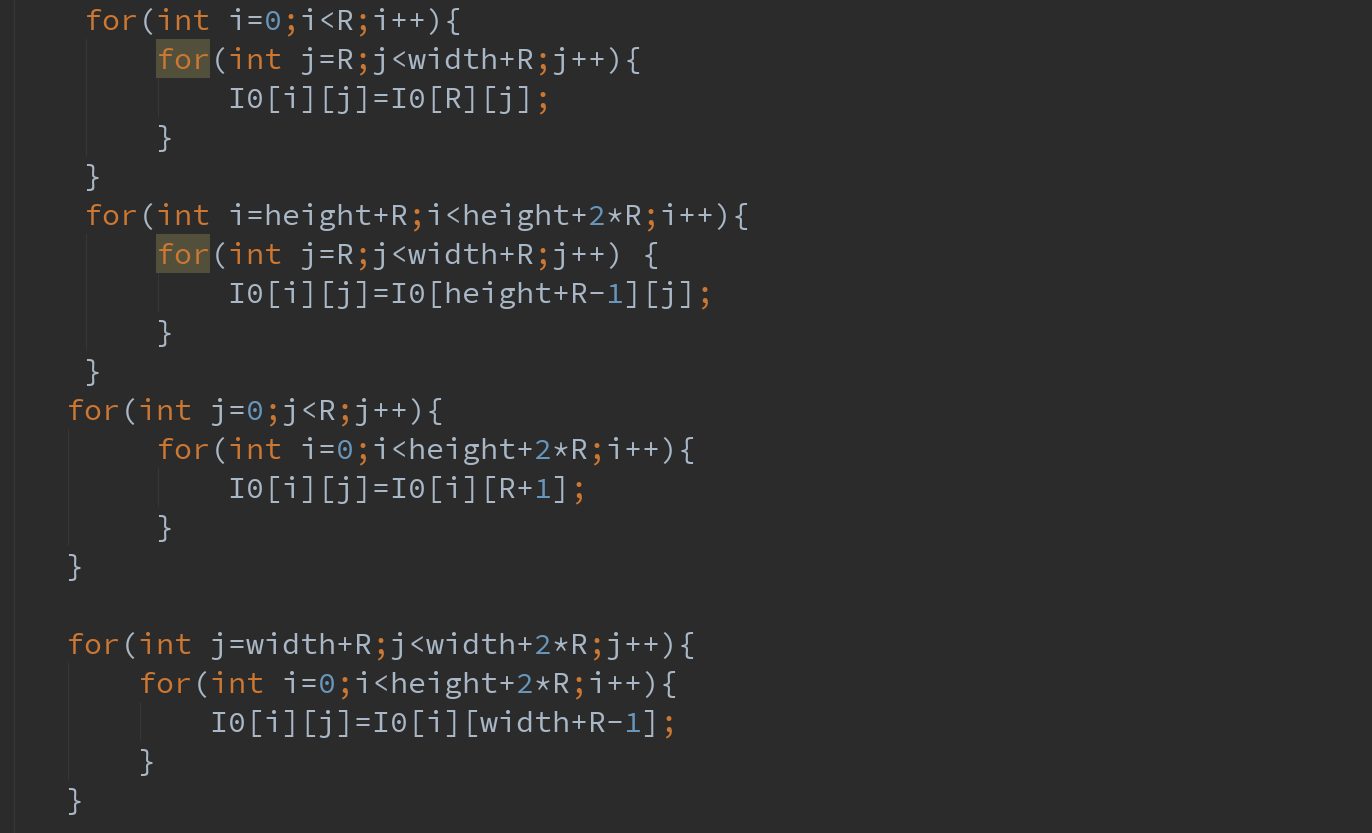


下面计算SILTP特征。首先把原始的128\*64维度的图像放在一个更大的容器里，也就是向左、向右、向上、向下扩展R个单位，值为0，这里R设为3：

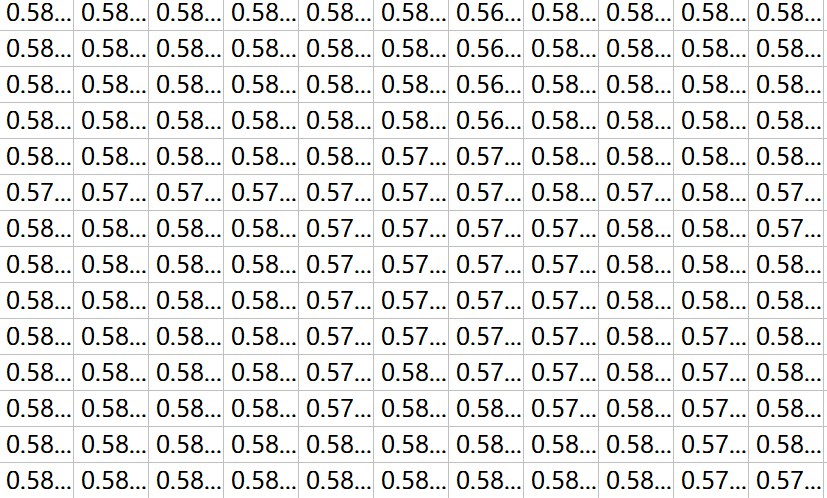


部分值如下：  


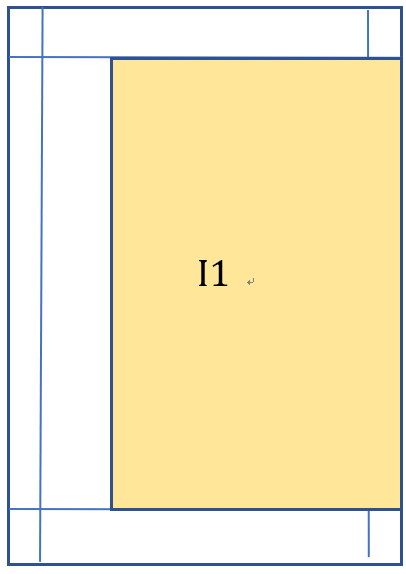
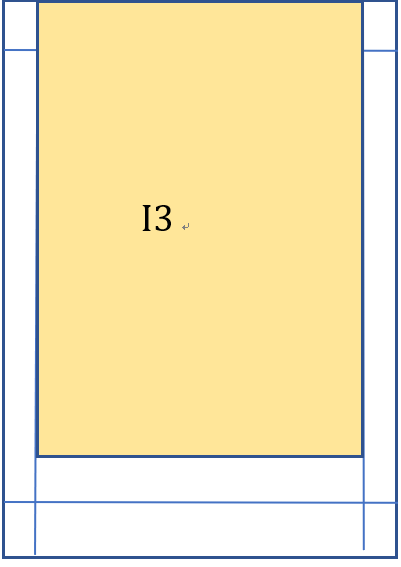
然后将边框图像像素复制到外部区域：

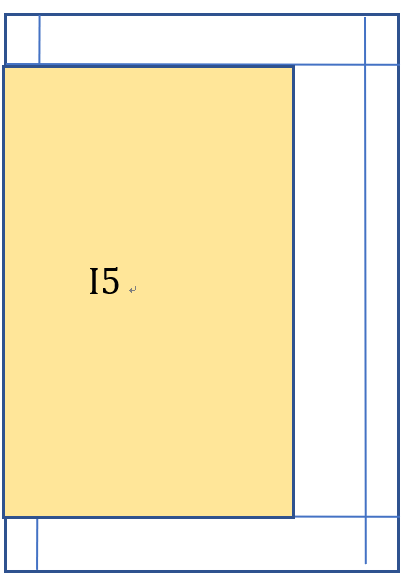
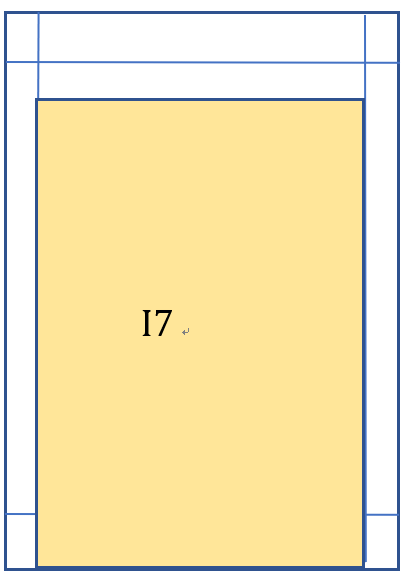


部分值如下：

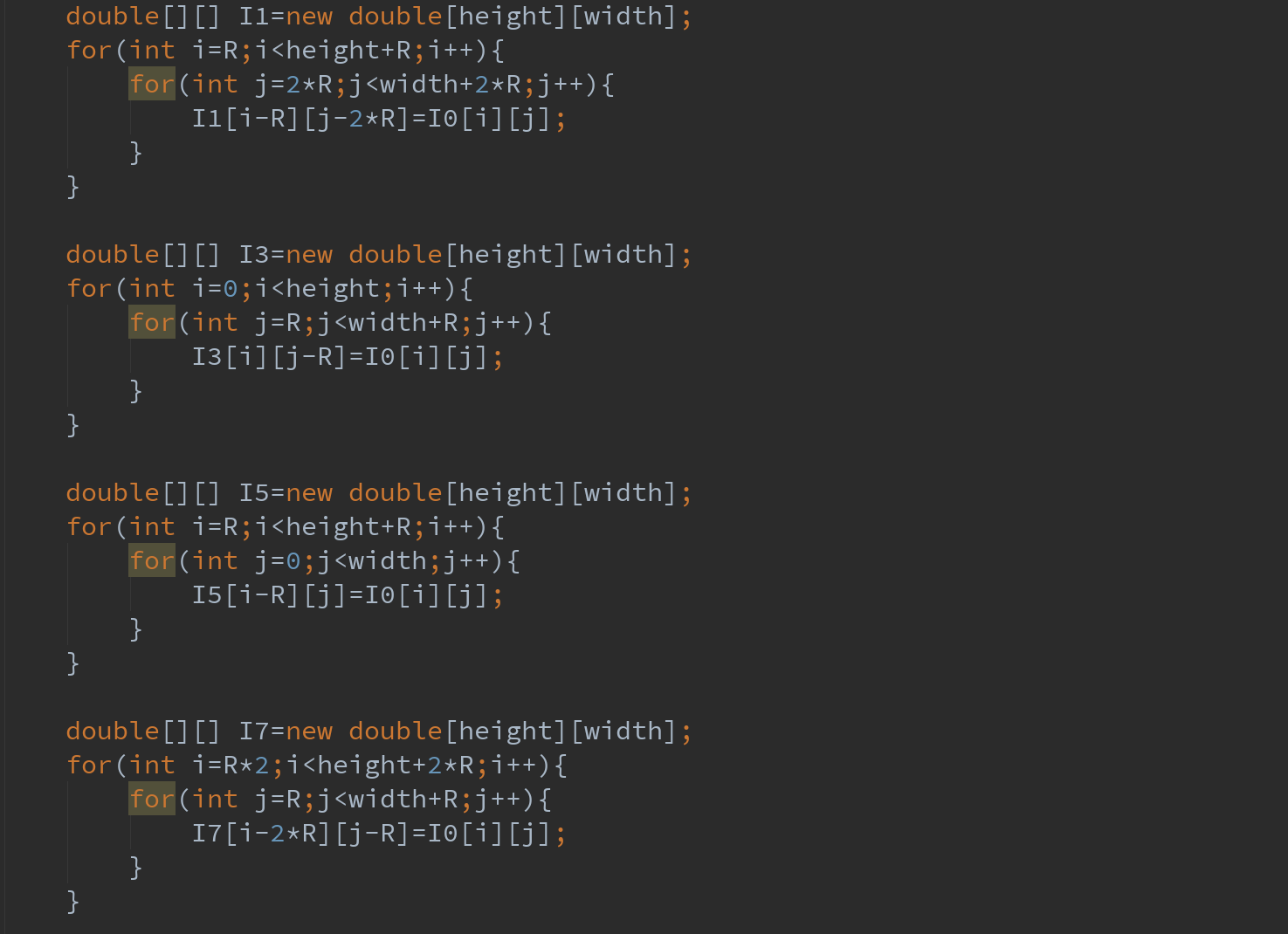


复制指定方向的图像I1、I3、I5、I7，其示意图如下所示：

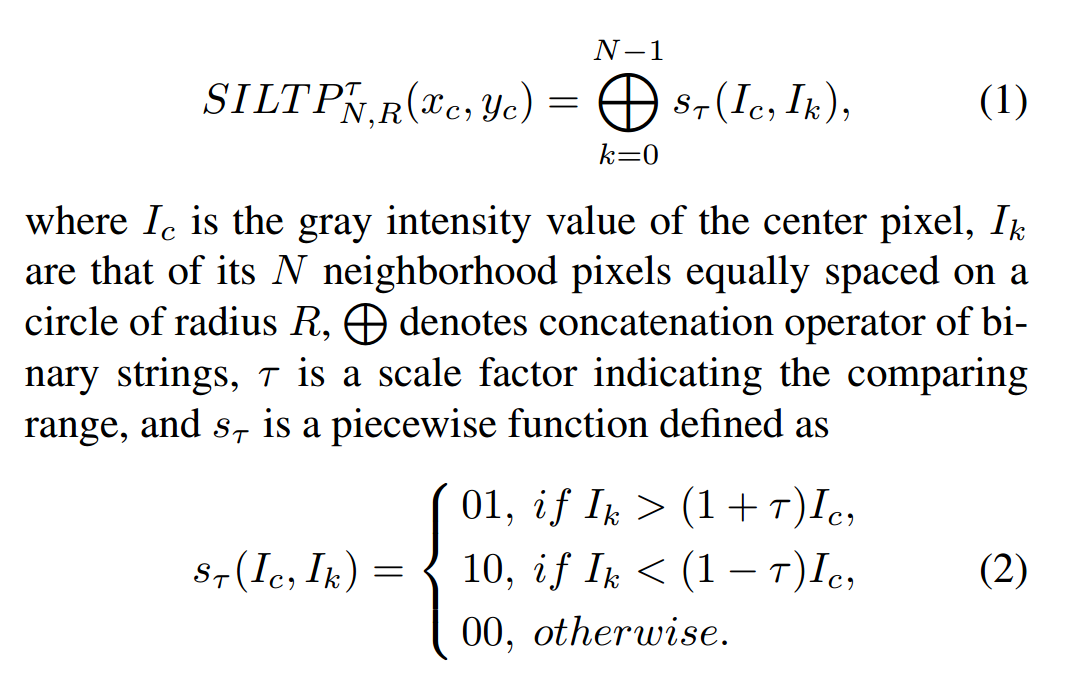
 

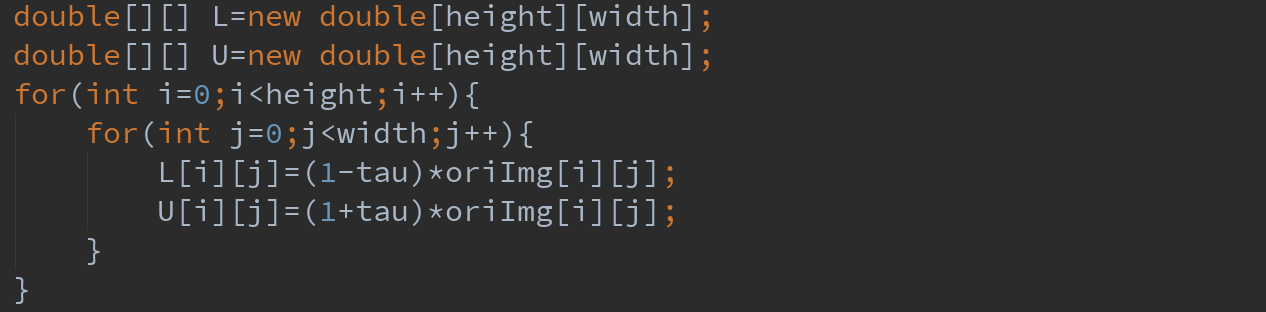
代码如下：

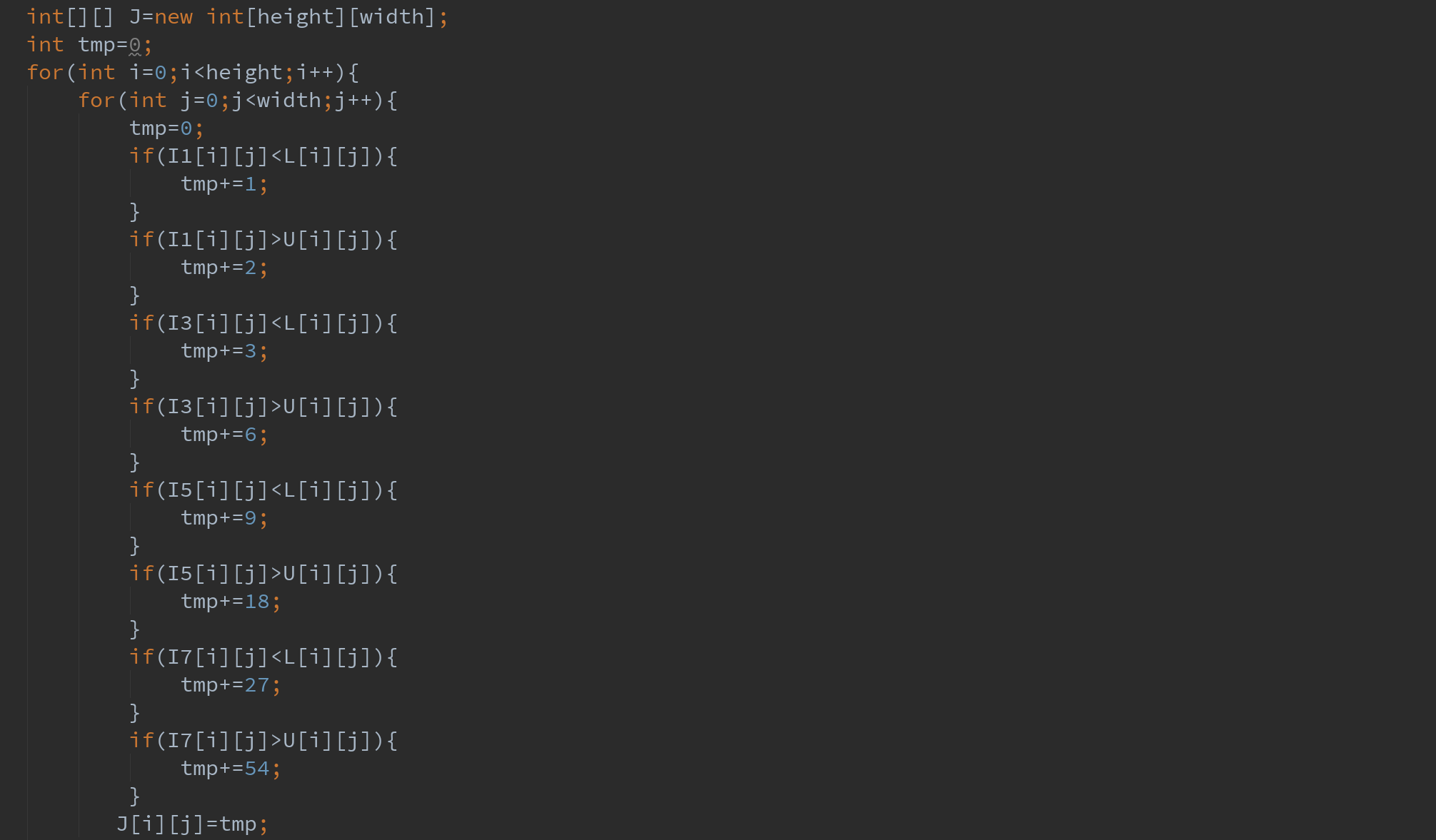


根据论文中的计算方法：

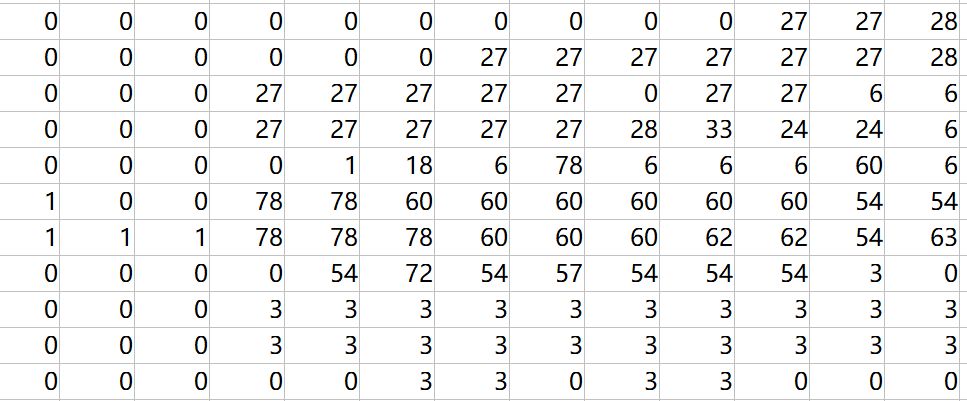


首先计算上界(1+τ)Ic和下界(1-τ)Ic，



然后计算SILTP值：  


得到一个128\*64维的矩阵， 部分结果如下：



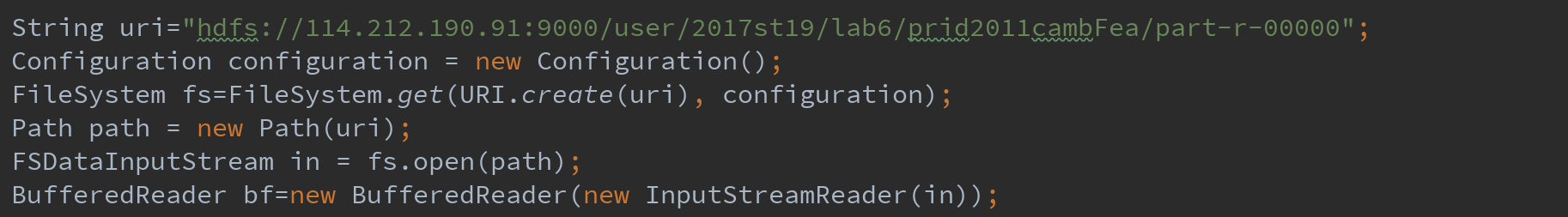
后面的步骤与基于HSV直方图的LOMO特征一样，不再赘述。

这一部分的算法如下：

|  |  |
| --- | --- |
| 输入 | Key:图像名(Text) Value: 图像内容 (BytesWritable) |
| 输出 | Key:图像名(Text) Value:图像LOMO特征(Text) |
| Mapper | 计算LOMO特征  输出<key,value>对，key为图像名(Text类型)，value为图像特征(Text类型) |
| Reducer | 无 |

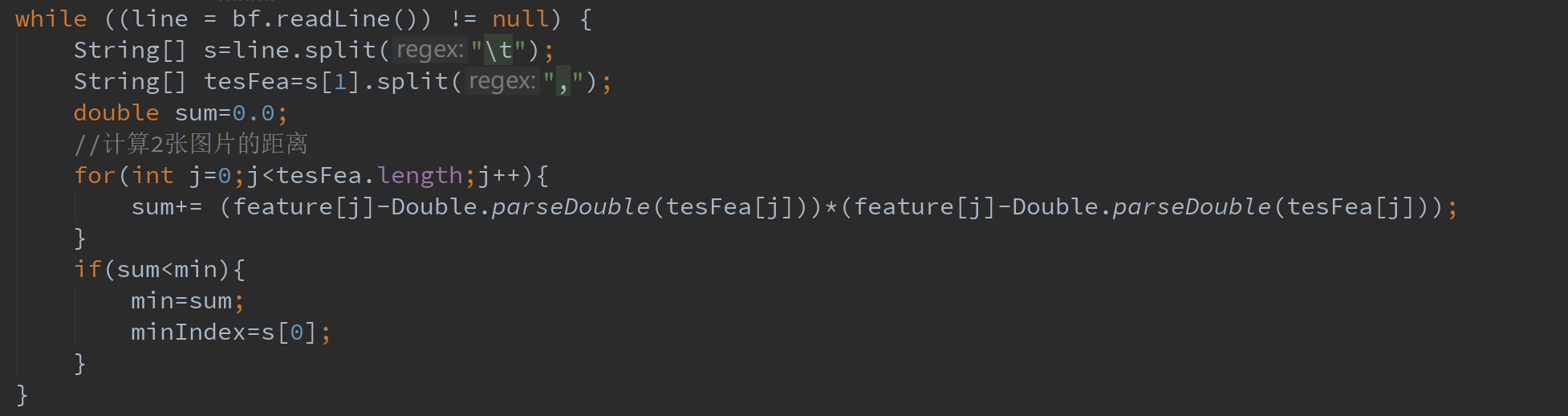
* 1. **图片检索(****knnMapper.java和knnReducer.java)**

本文中K设置为1，将上一步中产生的图片匹配数据库放在HDFS中供每个节点共享访问。

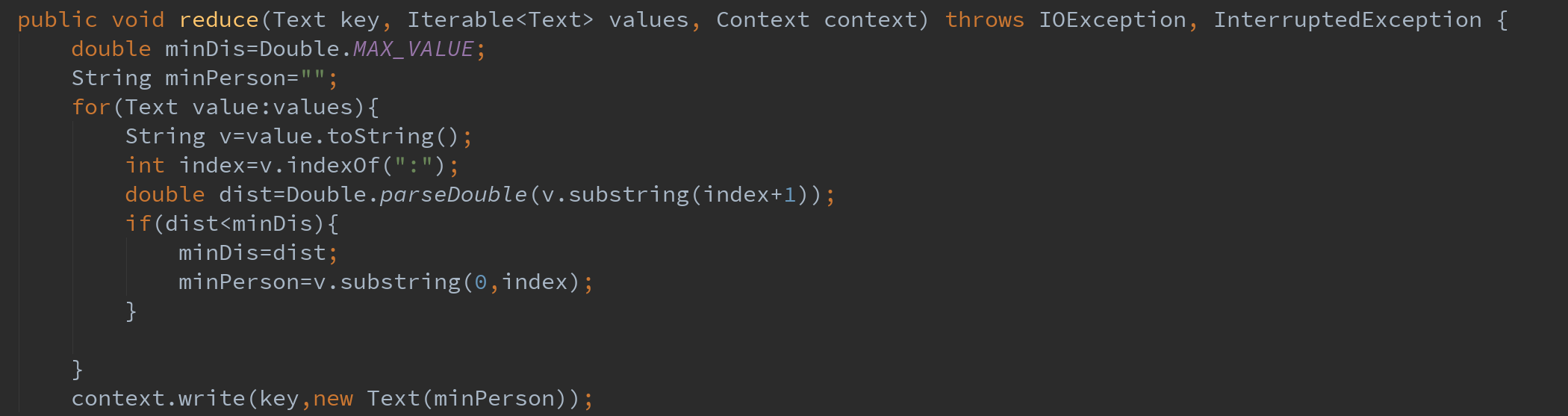
在knnMapper.java中，首先读取HDFS文件prid2011cambFea：  


本文使用的是viedo to viedo的检索形式。即给定某人的多张图片：person1/1.png 、person1/2.png、 person1/3.png，分别计算person1/1.png、person1/2.png、 person1/3.png与数据库中所有训练集的最佳匹配，得到：person1/10.png、person4/3.png、person10/2.png。然后在这几个最佳匹配中再选择距离最小的一张图片，如person1/10.png。由此得到最终的最佳匹配图片。

在Map函数中，计算测试样本与每一个训练样本的距离，输出最相似的图片：



在Reducer中，收到的value是key的最佳匹配的几个候选，Reducer再次计算最后的最佳匹配图像，发射出去。



这部分的算法如下:

|  |  |
| --- | --- |
| 输入 | Key:图像名(Text) Value: 图像LOMO特征(Text) |
| 输出 | Key:图像名(Text) Value:最佳匹配图像名(Text) |
| Mapper | For line in file:  Dist=calcDist(line, key)  If Dist<bsfDist:  bsfDist=Dist  输出<key,value>对，key为图像名(Text类型)，value为目前最相似图像名字(Text类型) |
| Reducer | For val:values:  If val<bsfDist:  bsfDist=val  输出<key,value>对，key为图像名(Text类型)，value为最佳匹配图像名字(Text类型) |

1. **实验**

本实验分为2个部分。

第一个部分是建立图像数据库，预先计算好数据库中的图像的特征，存入HDFS，供后期图像检索使用。数据库中包括16000多张图片，这一部分包括2个Job。放在集群上跑的时候，集群上当时任务不多，因此比较快地完成了，共耗时约1.5小时，相较于单机版极大地节省了运行时间。

第二个部分是真正的图像检索部分，用户输入图片，可以是单帧，也可以是多帧，输出是最佳匹配图片的索引。