基于 BOW 的图片搜索实现

摘要：

本文将 Bag of Words(BOW)模型应用于图像收缩，基于 OpenCV 计算机视觉库提取图像的 SURF 特征，然后用机器学习的方法使用 K-means 算法将特征点聚类成视觉词典，由视觉词典得到训练集中每幅图的词频矢量，以此计算测试图片与训练图片词频矢量之间的欧式距离，升排序得到最佳结果实现图像检索。实验表明，该模型在图像检索方面是准确率极高的。

关键词： K-means；Bag of Words；视觉词典；词频矢量

1.引言

近年来，随着计算机科学的飞速发展，模式识别技术在社会生活中的应用已经非常普遍，特别是计算机图像识别技术，因为其直观，方便，快速，准确的特点，更是受到了人们的青睐。

目前常用的利用计算机自动检索图像的方法是使用颜色、纹理、形状等全局视觉特征来获得图像内容信息，衡量图像之间的相似程度以实现图像检索。然而，这种方法反映的只是图像的客观统计特性，不能被人的视觉所理解。

本文考虑到图像检索系统的用户是根据图像中的目标来判别图像之间的相似性，因而将目标识别中的典型模型-Bag of Words(BOW)模型引入图像的检索识别领域，提取图像的局部特征来实现图像的匹配检索，一定程度上对图像有了语义方面的理解，也屏蔽了复杂背景对检索结果的影响。

自 BOW 模型被引入图像处理领域以来，BOW 模型在图像分类、目标识别等方面发挥着比较重要的作用，本文将基于图像 SURF 特征点和 BOW 模型，并对匹配方法做一定改进，实现一定数量图像数据集的图像匹配搜索。

2.BOW 模型概述

Bag of words 模型最初被用在文本分类中将文档表示成特征矢量。它的基本思想是假定对于一个文本，忽略其词序和语法、句法，仅仅将其看作是一些词汇的集合，而文本中的每个词汇都是独立的。

简单说就是将每篇文档都看成一个袋子(因为里边装的都是词汇，所以称为词袋，Bag of words 即由此来)，然后看这个袋子里装的都是些什么词汇，将其分类。如果文档中猪、马、牛、羊、山谷、土地、拖拉机这样的词汇多些，而银行、大厦、汽车、公园这样的词汇少些，我们就倾向于判断它是一篇描述乡村的

文档，而不是描述城镇的。

与之类似，我们也可以将图像当成一些图像片段（image patch——本文中用 SURF 特征来描述）的集合。譬如构成一幅人脸图像的图像片段必定倾向于描述人的眼睛、鼻子、耳朵、嘴巴这些事物(我们称这些事物为视觉词汇)，如图 1 所示。

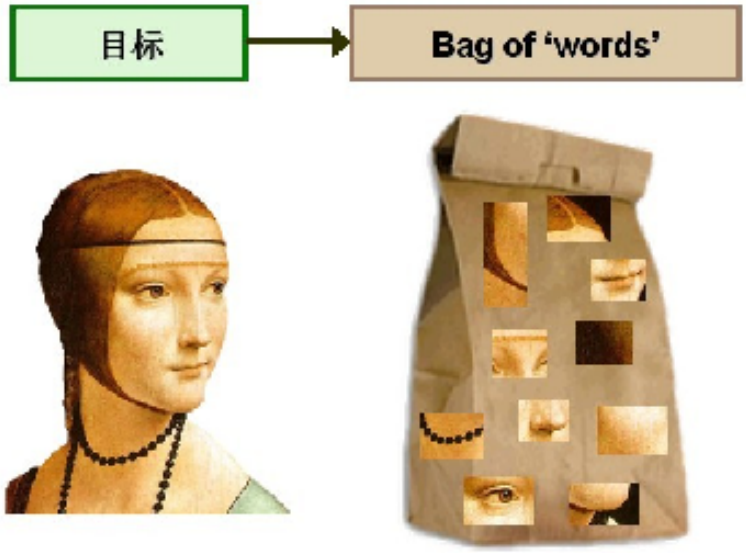


图 1 将 Bag-of-words 模型应用于图像表示

通过观察会发现，同一类目标的不同实例之间虽然存在差异，但我们仍然可以找到它们之间的一些共同的地方，比如说人脸，虽然说不同人的脸差别比较大，但眼睛、嘴、鼻子等一些比较细小的部位，却观察不到太大差别，我们可以把这些不同实例之间共同的部位提取出来，作为识别这一类目标的视觉词汇。

图 2 从图像中提取出相互独立的视觉词汇

而 SURF 算法是提取图像中局部不变特征的应用广泛的算法，但 SURF 在各个方面的性能均接近或超过了 SURF 算法，计算速度却是 SURF 的 3 倍左右，因此我们可以用 SURF 算法从图像中提取不变特征点，作为视觉词汇，并构造单词表，用单词表中的单词表示一幅图像。

BOW 模型算法流程：

首先要对训练图像集进行预处理。包括图像增强、分割、图像同一格式、同一规格的处理等等。

其次提取图像的 SURF 特征。即将每一幅图像划分成很多个图像片段，每一个图像片段都用一个 SURF 描述子矢量来表示。

再次聚类生成视觉词，构建码本。一幅图像的码本是由两部分组成的，视觉

词以及与其对应的 SURF 描述子矢量的数目，即词频。假设码本包含 k 个视觉词序列，将上步中所有的 SURF 描述子矢量用 K-means 聚类生成 k 个簇，最终这 k 个簇中心即为码本的 k 个视觉词，如图 3 所示。然后计算每幅图像中每个 SURF 描述子到这些视觉词的距离，若其距离某个视觉词最近，就将其映射到该视觉词，并将该视觉词对应的词频数增 1。直至将所有的 SURF 描述子都映射到码本中，将每幅图像都用一个与视觉词序列相对应的词频矢量来描述，即图片的视觉词汇直方图，如图 4 所示，这个词频矢量即为相似度检索时所要使用的图像的 Bag of words 特征矢量，另外，由于每幅图像所包含的 SURF 矢量的数目是不确定的，还需要对 Bag of words 特征矢量进行归一化。

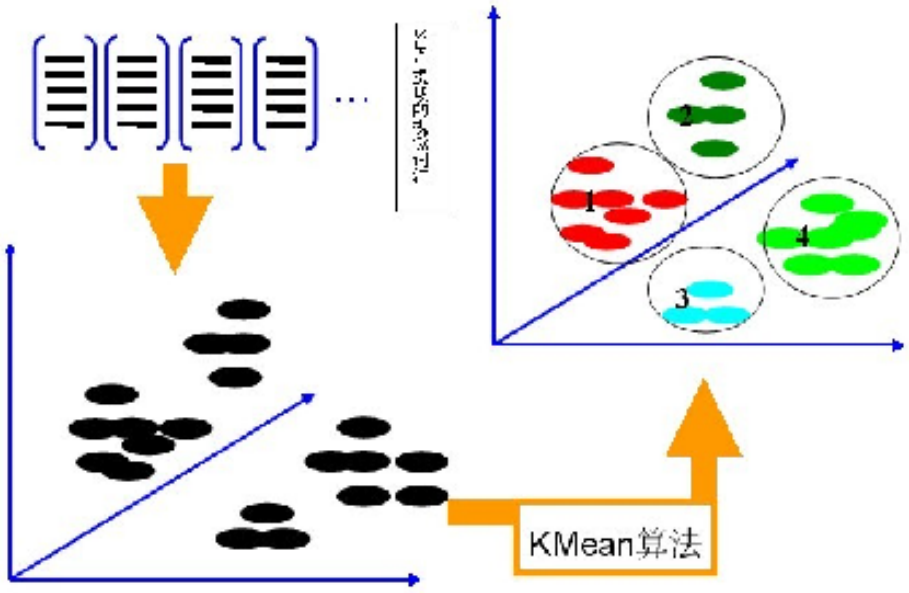


图 3 利用 K-Means 算法构造单词表

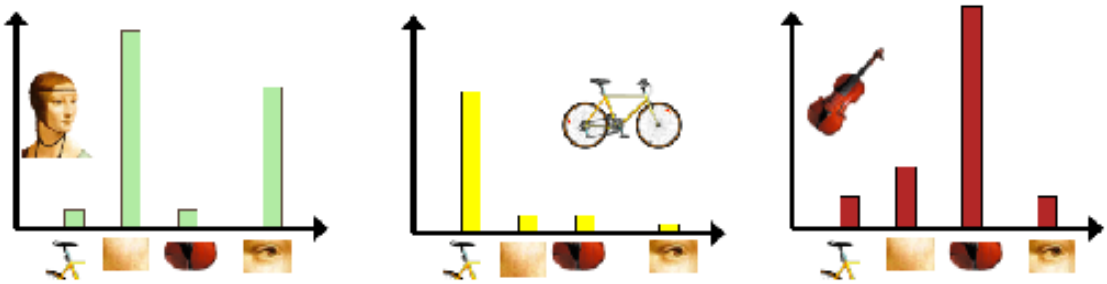


图 4 每幅图像的直方图表示

最后计算图像距离相似度进行检索。测试图像也要经过预处理，提取 SURF 特征，生成码本矢量并进行归一化的过程，然后计算其与训练码本的距离，并将此距离相似度按升序排列，距离最小者即为最相似的图片。

3.系统实现

本图片检索系统是采用 C++语言在 VS2010 平台下基于 OpenCV 开源计算机视觉库提取图片的 SURF 特征，使用 BOW 模型将其表示成特征矢量，最后计算测试图像的码本矢量与训练图像的码本矢量之间的距离实现图像检索。

工程文件如图5所示，训练模块train.cpp在运行前会检测本地是否有训练数据，即训练图像的码本矢量文件，如果没有则运行训练程序。train.cpp要做的工作分为三步：①提取图像库里每幅图片的SURF特征并转化为特征描述子；②对训练集中特征描述子进行K-means聚类，并设置成视觉词典(码本)，将视觉词典保存为dictionary.yml；③计算所有的特征描述子到视觉词典的距离，将每幅图都用视觉词典来表示，将词频矢量保存为bow_descriptor.yml。

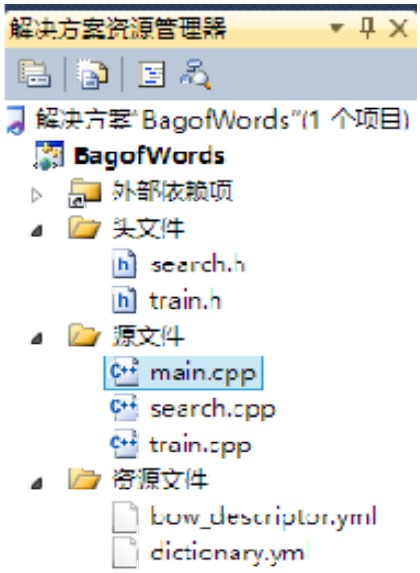


图 5 工程文件列表

search.cpp 则只需做两步工作，一是提取待测图片的 SURF 特征点并转化为描述子，并计算待测图片在视觉词典中的词频矢量，二是计算该特征矢量与训练集词频矢量的欧式距离，并排序。

K-means 聚类的实现：

在 OpenCV 的 BOWKMeansTrainer 类中以 cluster 函数调用 K-means 函数实现 K-means 聚类功能的，其源码如附录 3 所示。K-means 聚类是一种自上而下的聚类方法，即将所有样本视为一类，然后不断从大类中分离出小类，直到不能再分为止。它的优点是简单速度快，但必须提供聚类的数目，而在本项目中聚类的数目即字典的大小我们是知道的，因此 K-means 聚类是比较好的方法。

K-means 聚类是通过迭代的方法得到最优解的，计算各个对象到聚类中心的距离，距离越近相似度越高，最后将满足方差最小的 k 个聚类最为最后的输出。具体步骤：①初始化：从 n 个数据对象任意选择 k 个对象作为初始聚类中心；②根据每个聚类对象的均值(中心对象)，计算每个对象与这些中心对象的距离；并根据最小距离重新对相应对象进行分类；③由新的分类数据，重新计算每个(有变化）聚类的均值（中心对象)；④计算标准测度函数，当满足一定条件，如函数收敛时，则算法终止；如果条件不满足则循环执行。如 OpenCV 中，每次迭代，最大的聚类中心位移 max_center_shift < criteria.epsilon 小于精度要求时，就结束迭代。以及迭代次数超过设定的最大值时++iter == MAX(criteria.maxCount, 2)，也结束。

4.实验结果与分析

本文采用有 70 张图片的数据集，全部都是扫描的课本图片，随机另外拍摄 9 张图片作为测试图片，有 2000 个单词，每张图片有特征点 500 到 5000 不等，由测试输出结果可以看到,对相似度进行排序的 Top1 有 8 张查找到,Top5 有 9 张，即 Top5 的正确率达到 100%。本机 Windows8 操作系统，处理器酷睿 i5,4G 内存，用时 18656.9 毫秒测试完 9 张图片。

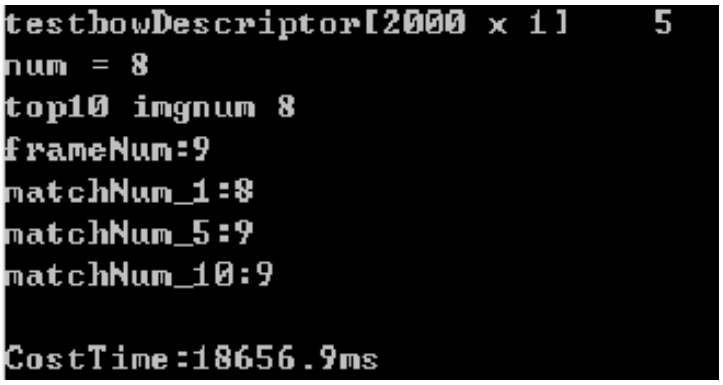


图 5 程序测试结果

实验结果表明：

①实验证明 Bag of Words 模型是可以用来做图像检索的；

②Bag of Words 模型做图像检索准确率较高，匹配准确度与单词表的大小有关；

③特征提取速度和检索速度较快，平均每张图片耗时 2s，图像的尺度变换、亮度、仿射变换和噪声等可变因素不会影响检索结果。

5.总结

本文将 Bag of Words 模型应用于图片检索而不是简单的将图像分类，在图像检索方面是一大创新，训练集都是课本书籍具有较大的相似性，增大了检索的难度，本实验的结果表明方法可行并且效果很好，使用 OpenCV 这样的开源工具使代码编写和功能实现显得更加便捷和准确。本文的缺点在于没有测试大样本情况的检索效果(2000 张的图片库训练时间太长，本机无法支持)，由于聚类的过程比较消耗 CPU 和内存，这需要增加处理器性能和修改算法以进行更加大规模数据的测试也是我们下一步需要进行的工作。

参考文献：

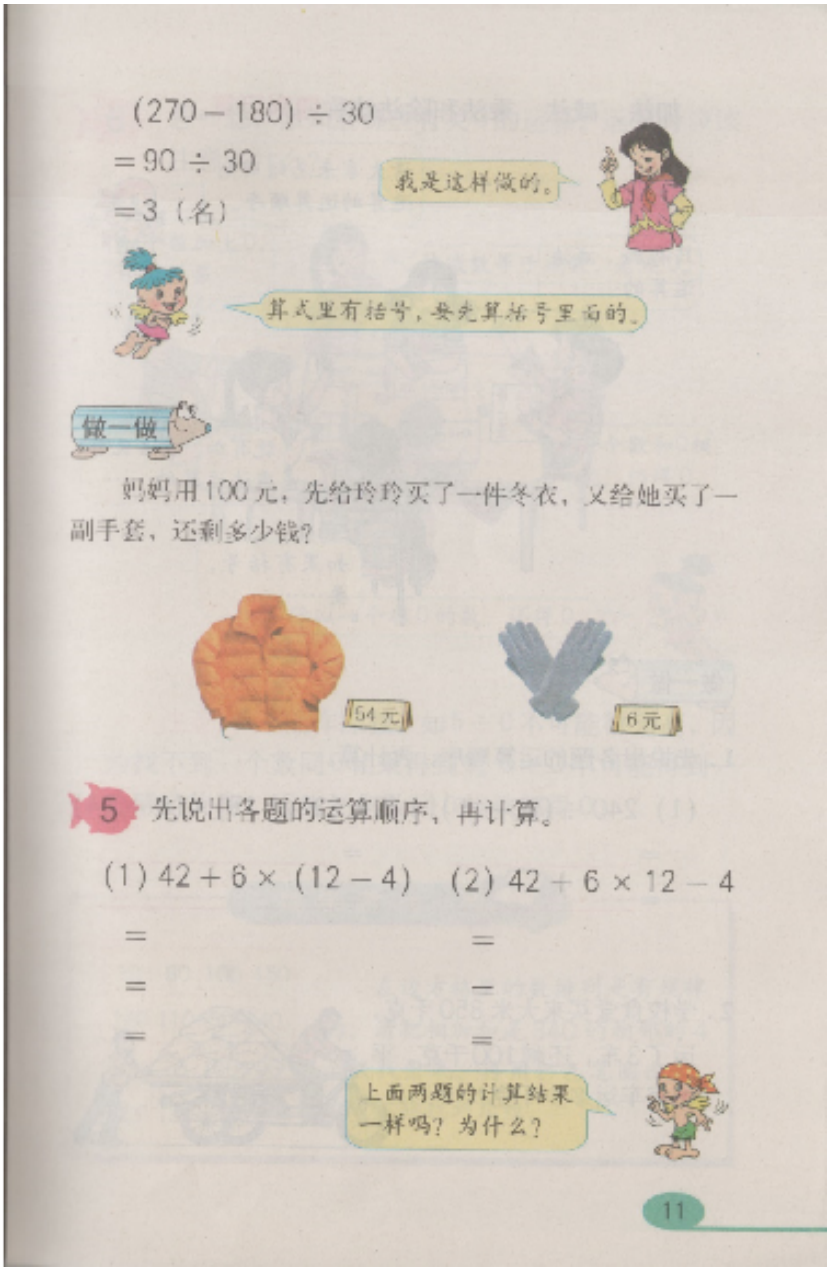
[1] 孙孟柯,张红梅. 基于 Bag of words 模型的图像检索系统的设计与实现.电脑知识与技术，2012.

[2] 刘红光,魏小敏.Bag of Words 算法框架的研究.舰船电子工程.2011 年第 9 期

[3] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, Cédric Bray. Visual Categorization with Bags of Keypoints. ECCV2004
[4] Hui ZHANG, Bangyu LI, Yan WANG, Jinfang ZHANG. Fanjiang XU Bag-of-Words Model Based Image Classification and Evaluation of Image Sample Quality in Remote Sensing

附录 1：

训练集例图



附录 2：BOWkmeansTrainer 的聚类实现代码

```
Mat BOWKMeansTrainer::cluster() const
{
    CV_Assert( !descriptors.empty() );

    int descCount = 0;
    for( size_t i = 0; i < descriptors.size(); i++ )
        descCount += descriptors[i].rows;

    Mat mergedDescriptors( descCount, descriptors[0].cols, descriptors[0].type() );
    for( size_t i = 0, start = 0; i < descriptors.size(); i++ )
    {
        Mat submut = mergedDescriptors.rowRange((int)start, (int)(start +
descriptors[i].rows));
        descriptors[i].copyTo(submut);
        start += descriptors[i].rows;
    }
    return cluster( mergedDescriptors );
}

BOWKMeansTrainer::~BOWK-meansTrainer()
{}

Mat BOWKMeansTrainer::cluster( const Mat& _descriptors ) const
{
    Mat labels, vocabulary;
    kmeans( _descriptors, clusterCount, labels, termcrit, attempts, flags, vocabulary );
    return vocabulary;
}
```

附录 3：OpenCV 中 K-means 的实现源码

```
double cv::ocl::kmeans(const oclMat &_src, int K, oclMat &_bestLabels,
                       TermCriteria criteria, int attempts, int flags, oclMat &_centers)
{
    const int SPP_TRIALS = 3;
    bool isrow = _src.rows == 1 && _src.oclchannels() > 1;
    int N = !isrow ? _src.rows : _src.cols;
    int dims = (!isrow ? _src.cols : 1) * _src.oclchannels();
```

```
int type = _src.depth();

attempts = std::max(attempts, 1);
CV_Assert(type == CV_32F && K > 0 );
CV_Assert( N >= K );

Mat _labels;
if( flags & CV_K-MEANS_USE_INITIAL_LABELS )
{
    CV_Assert( (_bestLabels.cols == 1 || _bestLabels.rows == 1) &&
                _bestLabels.cols * _bestLabels.rows == N &&
                _bestLabels.type() == CV_32S );
    _bestLabels.download(_labels);
}
else
{
    if( !((_bestLabels.cols == 1 || _bestLabels.rows == 1) &&
            _bestLabels.cols * _bestLabels.rows == N &&
            _bestLabels.type() == CV_32S &&
            _bestLabels.isContinuous()))
        _bestLabels.create(N, 1, CV_32S);
    _labels.create(_bestLabels.size(), _bestLabels.type());
}
int* labels = _labels.ptr<int>();

Mat data;
_src.download(data);
Mat centers(K, dims, type), old_centers(K, dims, type), temp(1, dims, type);
vector<int> counters(K);
vector<Vec2f> _box(dims);
Vec2f* box = &_box[0];
double best_compactness = DBL_MAX, compactness = 0;
RNG& rng = theRNG();
int a, iter, i, j, k;

if( criteria.type & TermCriteria::EPS )
    criteria.epsilon = std::max(criteria.epsilon, 0.);
else
    criteria.epsilon = FLT_EPSILON;
criteria.epsilon *= criteria.epsilon;

if( criteria.type & TermCriteria::COUNT )
    criteria.maxCount = std::min(std::max(criteria.maxCount, 2), 100);
else
```

```
        criteria.maxCount = 100;

if( K == 1 )
{
        attempts = 1;
        criteria.maxCount = 2;
}

const float* sample = data.ptr<float>();
for( j = 0; j < dims; j++ )
        box[j] = Vec2f(sample[j], sample[j]);

for( i = 1; i < N; i++ )
{
        sample = data.ptr<float>(i);
        for( j = 0; j < dims; j++ )
        {
                float v = sample[j];
                box[j][0] = std::min(box[j][0], v);
                box[j][1] = std::max(box[j][1], v);
        }
}

for( a = 0; a < attempts; a++ )
{
        double max_center_shift = DBL_MAX;
        for( iter = 0;; )
        {
                swap(centers, old_centers);

                if( iter == 0 && (a > 0 || !(flags & K-MEANS_USE_INITIAL_LABELS)) )
                {
                        if( flags & K-MEANS_PP_CENTERS )
                                generateCentersPP(data, centers, K, rng, SPP_TRIALS);
                        else
                        {
                                for( k = 0; k < K; k++ )
                                        generateRandomCenter(_box, centers.ptr<float>(k), rng);
                        }
                }
                else
                {
                        if( iter == 0 && a == 0 && (flags & K-MEANS_USE_INITIAL_LABELS) )
                        {
```

```
                for( i = 0; i < N; i++ )
                    CV_Assert( (unsigned)labels[i]  < (unsigned)K );
            }

            // compute centers
            centers = Scalar(0);
            for( k = 0; k < K; k++ )
                counters[k] = 0;

            for( i = 0; i < N; i++ )
            {
                sample = data.ptr<float>(i);
                k = labels[i];
                float* center = centers.ptr<float>(k);
                j=0;
#if CV_ENABLE_UNROLLED
                for(; j <= dims - 4; j += 4 )
                {
                    float t0 = center[j] + sample[j];
                    float t1 = center[j+1] + sample[j+1];

                    center[j] = t0;
                    center[j+1] = t1;

                    t0 = center[j+2] + sample[j+2];
                    t1 = center[j+3] + sample[j+3];

                    center[j+2] = t0;
                    center[j+3] = t1;
                }
#endif
                for( ; j < dims; j++ )
                    center[j] += sample[j];
                counters[k]++;
            }

            if( iter > 0 )
                max_center_shift = 0;

            for( k = 0; k < K; k++ )
            {
                if( counters[k] != 0 )
                    continue;
```

```
// if some cluster appeared to be empty then:
//    1. find the biggest cluster
//    2. find the farthest from the center point in the biggest cluster
//    3. exclude the farthest point from the biggest cluster and form a
new 1-point cluster.
int max_k = 0;
for( int k1 = 1; k1 < K; k1++ )
{
    if( counters[max_k] < counters[k1] )
        max_k = k1;
}

double max_dist = 0;
int farthest_i = -1;
float* new_center =    centers.ptr<float>(k);
float* old_center =    centers.ptr<float>(max_k);
float* _old_center = temp.ptr<float>(); // normalized
float scale = 1.f/counters[max_k];
for( j = 0; j < dims; j++ )
    _old_center[j] = old_center[j]*scale;

for( i = 0; i < N; i++ )
{
    if( labels[i] != max_k )
        continue;
    sample = data.ptr<float>(i);
    double dist = normL2Sqr_(sample, _old_center, dims);

    if( max_dist <= dist )
    {
        max_dist = dist;
        farthest_i = i;
    }
}

counters[max_k]--;
counters[k]++;
labels[farthest_i] = k;
sample = data.ptr<float>(farthest_i);

for( j = 0; j < dims; j++ )
{
    old_center[j] -= sample[j];
    new_center[j] += sample[j];
```

```
                }
            }

            for( k = 0; k < K; k++ )
            {
                float* center = centers.ptr<float>(k);
                CV_Assert( counters[k] != 0 );

                float scale = 1.f/counters[k];
                for( j = 0; j < dims; j++ )
                    center[j] *= scale;

                if( iter > 0 )
                {
                    double dist = 0;
                    const float* old_center = old_centers.ptr<float>(k);
                    for( j = 0; j < dims; j++ )
                    {
                        double t = center[j] - old_center[j];
                        dist += t*t;
                    }
                    max_center_shift = std::max(max_center_shift, dist);
                }
            }
        }

        if( ++iter == MAX(criteria.maxCount, 2) || max_center_shift <= criteria.epsilon )
            break;

        // assign labels
        Mat dists(1, N, CV_64F);
        _centers.upload(centers);
        distanceToCenters(_src, _centers, dists, _labels);
        _bestLabels.upload(_labels);

        float* dist = dists.ptr<float>(0);
        compactness = 0;
        for( i = 0; i < N; i++ )
            compactness += (double)dist[i];
    }

    if( compactness < best_compactness )
        best_compactness = compactness;
}
```

```
        return best_compactness;
}
```