**Histogram of Oriented Gradients算法详解**

目录

[1. HOG算法简介 2](#_Toc375773451)

[2. HOG算法步骤 6](#_Toc375773452)

[2.1 色彩和伽马归一化（color and gamma normalization） 6](#_Toc375773453)

[2.2 梯度的计算（Gradient computation） 6](#_Toc375773454)

[2.3 构建方向的直方图(creating the orientation histograms) 7](#_Toc375773455)

[2.4 把细胞单元组合成大的区间(grouping the cells together into larger blocks) 12](#_Toc375773456)

[2.5 区间归一化（Block normalization） 13](#_Toc375773457)

[2.6 SVM分类器（SVM classifier）进行分类 14](#_Toc375773458)

[3. HOG算法简单汇总及实现 17](#_Toc375773459)

[3.1 HOG流程简单汇总： 17](#_Toc375773460)

[3.2 例图lena图： 17](#_Toc375773461)

[3.3 HOG原理matlab代码： 18](#_Toc375773462)

[4. HOG算法详细分析及实现 20](#_Toc375773463)

[4.1 HOG Descriptor in MATLAB 20](#_Toc375773464)

[4.2 HOG Person Detector Tutorial 21](#_Toc375773465)

[4.3 Gradient Vectors 26](#_Toc375773466)

[4.4 Image Derivative 30](#_Toc375773467)

[4.5 matlab code 34](#_Toc375773468)

[5. Digit Classification Using HOG Features 40](#_Toc375773469)

[5.1 Digit Data Set 41](#_Toc375773470)

[5.2 Using HOG Features 42](#_Toc375773471)

[5.3 Train the Classifier 43](#_Toc375773472)

[5.4 Test the Classifier 44](#_Toc375773473)

[5.5 Results 44](#_Toc375773474)

[5.6 Summary 45](#_Toc375773475)

[5.7 References 45](#_Toc375773476)

[5.8 Appendix - Helper functions 45](#_Toc375773477)

[参考文献 46](#_Toc375773478)

[附录A: HOG特征提取matlab代码 46](#_Toc375773479)

[Code 1 46](#_Toc375773480)

本文包含HOG算法(方向梯度直方图)的详细解释及特征描述子的计算matlab代码；

## HOG算法简介

方向梯度直方图（Histogram of Oriented Gradient, HOG）特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。此方法使用了图像的本身的梯度方向特征，类似于边缘方向直方图方法，SIFT描述子，和上下文形状方法，但其特征在于其在一个网格密集的大小统一的方格单元上计算，而且为了提高精确度使用了重叠的局部对比度归一化的方法。

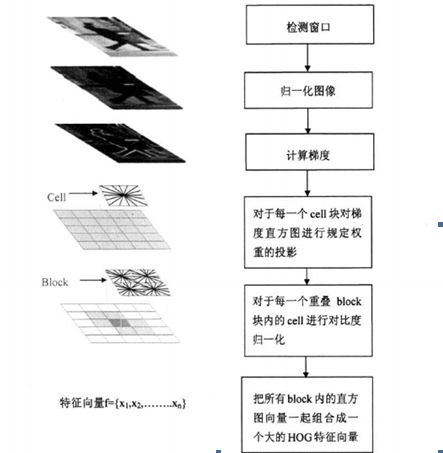
HOG descriptors 是应用在计算机视觉和图像处理领域，用于目标检测的特征描述器。这项技术是用来计算局部图像梯度的方向信息的统计值。这种方法跟边缘方向直方图（edge orientation histograms）、尺度不变特征变换（scale-invariant feature transform descriptors）以及形状上下文方法（ shape contexts）有很多相似之处，但与它们的不同点是：HOG描述器是在一个网格密集的大小统一的细胞单元（dense grid of uniformly spaced cells）上计算，而且为了提高性能，还采用了重叠的局部对比度归一化（overlapping local contrast normalization）技术。

这篇文章的作者Navneet Dalal和Bill Triggs是法国国家计算机技术和控制研究所French National Institute for Research in Computer Science and Control (INRIA)的研究员。他们在这篇文章中首次提出了HOG方法。这篇文章被发表在2005年的CVPR上。他们主要是将这种方法应用在静态图像中的行人检测上，但在后来，他们也将其应用在电影和视频中的行人检测，以及静态图像中的车辆和常见动物的检测。

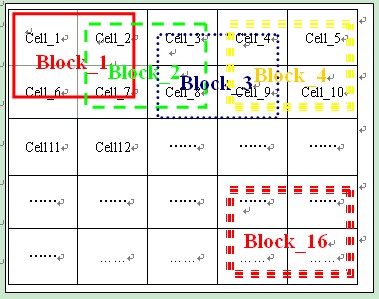
HOG描述器最重要的思想是：在一副图像中，局部目标的表象和形状（appearance and shape）能够被梯度或边缘的方向密度分布很好地描述。具体的实现方法是：首先将图像分成小的连通区域，我们把它叫细胞单元。然后采集细胞单元中各像素点的梯度的或边缘的方向直方图。最后把这些直方图组合起来就可以构成特征描述器。为了提高性能，我们还可以把这些局部直方图在图像的更大的范围内（我们把它叫区间或block）进行对比度归一化（contrast-normalized），所采用的方法是：先计算各直方图在这个区间（block）中的密度，然后根据这个密度对区间中的各个细胞单元做归一化。通过这个归一化后，能对光照变化和阴影获得更好的效果。

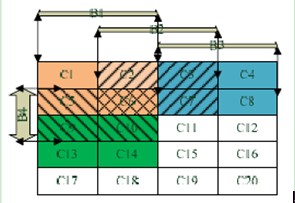
与其他的特征描述方法相比，HOG描述器后很多优点。首先，由于HOG方法是在图像的局部细胞单元上操作，所以它对图像几何的（geometric）和光学的（photometric）形变都能保持很好的不变性，这两种形变只会出现在更大的空间领域上。其次，作者通过实验发现，在粗的空域抽样（coarse spatial sampling）、精细的方向抽样（fine orientation sampling）以及较强的局部光学归一化（strong local photometric normalization）等条件下，只要行人大体上能够保持直立的姿势，就容许行人有一些细微的肢体动作，这些细微的动作可以被忽略而不影响检测效果。综上所述，HOG方法是特别适合于做图像中的行人检测的。

接下来将具体介绍HOG特征的提取过程，下图1给出了完整的HOG特征提取算法和过程。图1如下：



而具体的区域分块图，cell和Block的关系图为：





上面两图充分说明了cell和Block的关系。如图一，假设图像为40\*40，假定每个Block有2\*2个cell，每个cell为8\*8。由于相邻的两个Block之间有50%(水平或垂直相邻 水平和垂直步进是1个cell /8个像素 )或25%(对角相邻)的重叠部分，因此这幅图有4\*4个Block存在。

HOG提取过程中的核心步骤可以简单归纳如下：

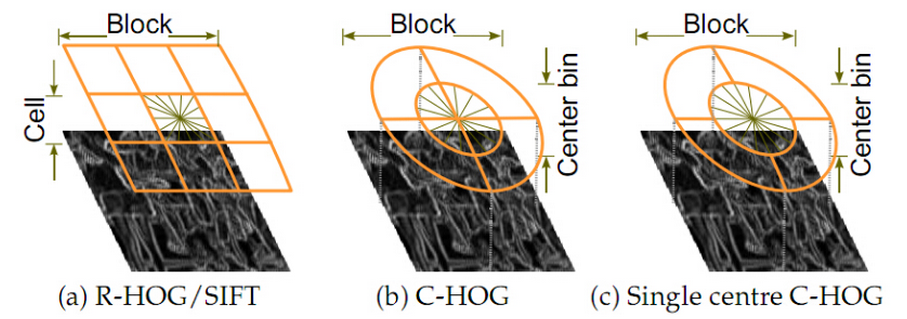
(1) 计算图像横坐标和纵坐标方向的梯度，并据此计算每个像素位置的梯度方向值；

(2) 把图像按照空间位置均匀划分小块，即图2中的cell，在cell内按照既定的量化标准统计梯度方向的直方图，得到该cell对应的特征向量，然后再把图2中一个block内所有cell的特征向量串联起来便得到该block的HOG特征。

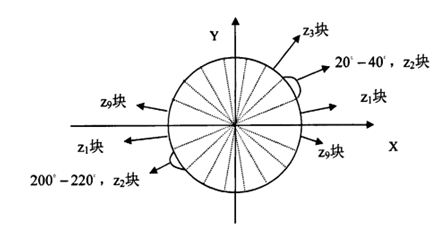
图2如下：



HOG特征是由一个矩形block中的多个矩形cell的方向梯度直方图串联得到，其实block和cell都可以不是矩形，只不过矩形是较常用的一种，这种HOG特征可称之为矩形HOG，实际上还存在圆形HOG和中心环绕HOG，当然我们考虑的都是矩形的。



矩形HOG使用重复遍历的矩形cell网格，Block块是由密集重复遍历的cell组成的，每一个块内独立进行特征向量归一化，以减少光照的影响。一般每个block由m×m个cell网格组成，而cell由n×n个像素组成。并且每个cell的梯度方向分成z个方向块，使用cell中的梯度方向和幅度对z个方向进行加权投影，最后每个cell产生z维的特征向量。Dalal等用于人体检测的HOG选取z=9，即将360度分成9个方向块，继而用于对方向梯度进行投影，如下图：



 每个cell在按上图划分进行投影后，便得到一个z维的特征向量，block块中的所有cell对应的特征向量串联起来便构成了HOG特征。

结合40\*40像素的图像有16个block，每个block包含4个cell，每个cell包含9个方向块的特征向量，则此幅图像共有16\*4\*9=576个特征向量；

获得图像的HOG特征向量之后，就可将

上图是作者做的行人检测试验，其中（a）表示所有训练图像集的平均梯度（average gradient across their training images）；（b）和（c）分别表示：图像中每一个区间（block）上的最大最大正、负SVM权值；（d）表示一副测试图像；（e）计算完R-HOG后的测试图像；（f）和（g）分别表示被正、负SVM权值加权后的R-HOG图像。

算法的实现：

## HOG算法步骤

### 2.1 色彩和伽马归一化（color and gamma normalization）

作者分别在灰度空间、RGB色彩空间和LAB色彩空间上对图像进行色彩和伽马归一化，但实验结果显示，这个归一化的预处理工作对最后的结果没有影响，原因可能是：在后续步骤中也有归一化的过程，那些过程可以取代这个预处理的归一化。所以，在实际应用中，这一步可以省略。根据原论文中的说明，可以采用“Square root gamma compression of each colour channel”，即对每个颜色通道进行平方根的伽马压缩，实现归一化。

### 2.2 梯度的计算（Gradient computation）

最常用的方法是：简单地使用一个一维的离散微分模板（1-D centered, point discrete derivative mask）在一个方向上或者同时在水平和垂直两个方向上对图像进行处理，更确切地说，这个方法需要使用下面的滤波器核滤除图像中的色彩或变化剧烈的数据（color or intensity data）。 具体方法及步骤如下：

1. 方法：

首先用[-1,0,1]梯度算子对原图像做卷积运算，得到x方向（水平方向，以向右为正方向）的梯度分量gradscalx，然后用[1,0,-1]’梯度算子对原图像做卷积运算，得到y方向（竖直方向，以向上为正方向）的梯度分量gradscaly。然后当gradscalx>=0, gradscaly>=0时，说明梯度方向是朝向第一象限的，当gradscalx>=0, gradscaly<0时，说明梯度方向是朝向第二象限的，诸如此类，结合象限信息，就可以利用反正切函数atan求出在signed和unsigned各自情况下正确的梯度角度。

1. 实施步骤：
   1. 假设检测窗为64(列)\*128(行)大小，block为16\*16大小，每个block划分为4个cell，block每次滑动8个像素(也就是一个cell的宽)，以及梯度方向划分为9个区间，在0~180度范围内统计，以下的说明都以上述假设为例.
   2. btly与btlx分别表示block所在位置左上角点处的坐标。对于前述假设，一个检测窗内会有105个block存在，因此第一个block左上角的坐标是(1,1),第二个是(9,1) (列，行)…,此行最后一个是block的左上角坐标是(49,1),然后下一个block就需要向下滑动8个像素，并回到最左边,此时的block左上角坐标为(1,9),接着block重新开始新的横向滑动…如此这般,在检测窗内最后一个block的坐标就是(49,113).(像素)
   3. block每滑动到一个新的位置，就需要停下来计算它内部的那四个cell中的梯度方向直方图.(bj,bi)就是来存储cell左上角的坐标的（cell的坐标以block左上角为原点）.
   4. (j,i)就表示cell中的像素在整个检测窗（64\*128的图像）中的坐标.另外，我在程序里有个jorbj与iorbi，这在Localinterpolate的情况下（也就是标准的原始HOG情况），就是bj与bi.

作者也尝试了其他一些更复杂的模板，如3×3 Sobel 模板，或对角线模板（diagonal masks），但是在这个行人检测的实验中，这些复杂模板的表现都较差，所以作者的结论是：模板越简单，效果反而越好。作者也尝试了在使用微分模板前加入一个高斯平滑滤波，但是这个高斯平滑滤波的加入使得检测效果更差，原因是：许多有用的图像信息是来自变化剧烈的边缘，而在计算梯度之前加入高斯滤波会把这些边缘滤除掉。

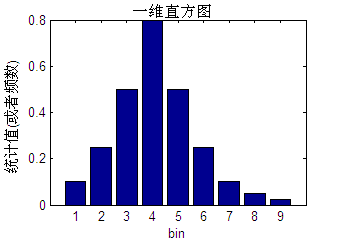
### 2.3 构建方向的直方图(creating the orientation histograms)

第三步就是为图像的每个细胞单元构建梯度方向直方图。细胞单元中的每一个像素点都为某个基于方向的直方图通道（orientation-based histogram channel）投票。投票是采取加权投票（weighted voting）的方式，即每一票都是带权值的，这个权值是根据该像素点的梯度幅度计算出来。可以采用幅值本身或者它的函数来表示这个权值，实际测试表明：使用幅值来表示权值能获得最佳的效果，当然，也可以选择幅值的函数来表示，比如幅值的平方根（square root）、幅值的平方（square of the gradient magnitude）、幅值的截断形式（clipped version of the magnitude）等。细胞单元可以是矩形的（rectangular），也可以是星形的（radial）。直方图通道是平均分布在0-1800（无向）或0-3600（有向）范围内。作者发现，采用无向的梯度和9个直方图通道，能在行人检测试验中取得最佳的效果。

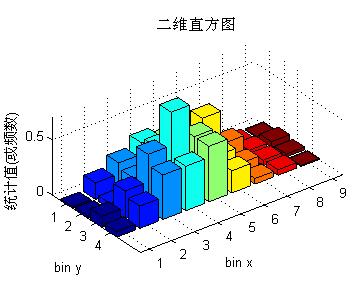
//////////////////////////////////////////////////////////////////////////////////////////////////////////

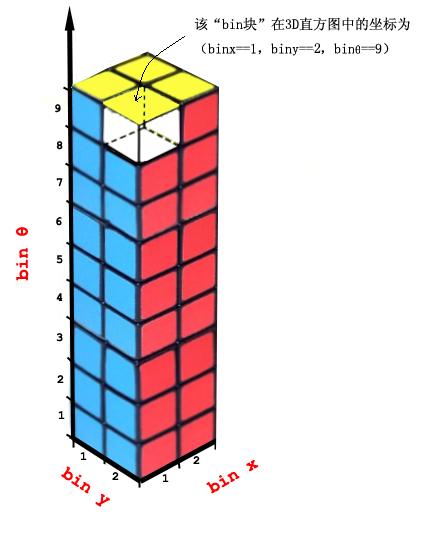
关于hist3dbig：

这是一个三维的矩阵，用来存储三维直方图。最常见的一维的直方图是这个样子，

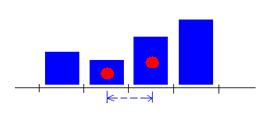


二维直方图呢？是这个样子，一个一个的柱子是一个统计bin，柱子的高低代表统计值的大小

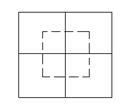
  
三维直方图呢？是这个样子，立体的一个一个的小格子，每个小格子是一个统计bin, 小格子用来装统计值。以上面的例子，那么对一个block来说，它的直方图是下面这样的：



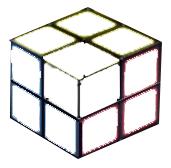
再来说线性插值，线性插值时，一个统计值需被“按一定比例分配”到这个统计点最邻近的区间中去，下面的图显示了一维直方图时，落在虚线标记范围内的统计点，它最近邻的区间就是标有红色圆点的两个区间

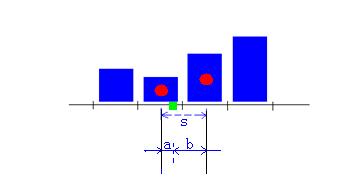


若是二维直方图，那落在如下虚线矩形中的统计点，周围的这四个统计区间就是它最近邻的区间。这个虚线矩形由四个统计区间各自的1/4组成。

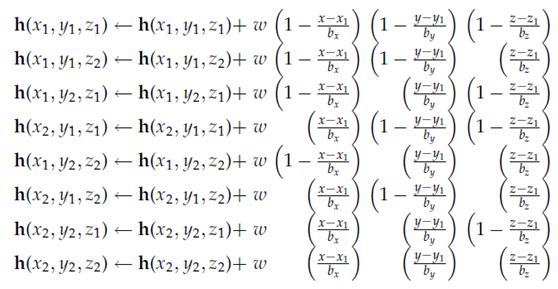


三维直方图，对一个统计点来说，它的最近邻的区间有八个，如下图，可以想象一下，只有当这个统计点落在由如下八个统计区间各自的1/8组成的一个立方体内内时，这八个区间才是对统计点最近邻的。



统计时如何分配权重呢？以一维直方图简单说一下线性插值的意思，对于下面绿色小方点(x)的统计值来说，假设标红点的两个bin的中心位置分别为x1，x2，那么对于x，它的分配权重为左边bin: 1-(x-x1)/s, 即 1-a/s = b/s, 右边bin: 1-(x2-x)/s, 即1-b/s = a/s.  


类似，那么对三维直方图来说，统计时的累积式（从Dalal的论文里截来的）就是：



上面，w 就是准备被分配的统计值。(x1,y1,z1)…共八个点表示八个统计区间的中心位置坐标，上式用h(x1,y1,z1)这样的标记来表示所要累积的统计区间。我在编程时就使用的这个式子，只不过我用bin的下标号来表示bin块，就像前面三维直方图示意中(binx=1,biny=2,binθ=9)，不过在程序中θ轴是用z轴表示了。

                binx1 = floor((jorbj-1+cellpw/2)/cellpw) + 1;

                biny1 = floor((iorbi-1+cellph/2)/cellph) + 1;

                binz1 = floor((go+(or\*pi/nthet)/2)/(or\*pi/nthet)) + 1;

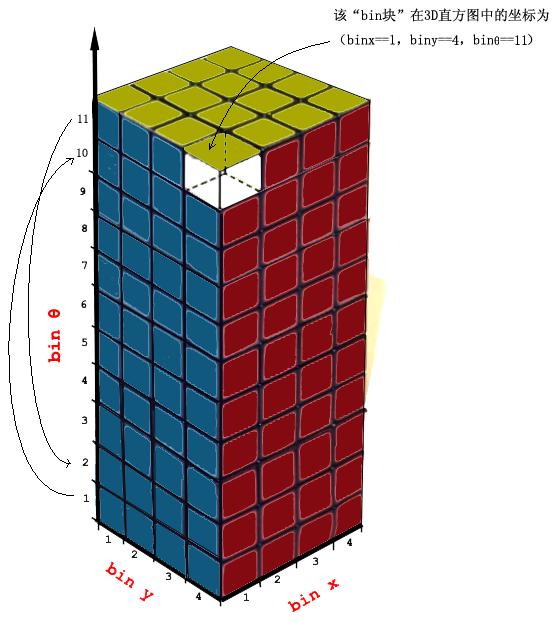
                binx2 = binx1 + 1;

                biny2 = biny1 + 1;

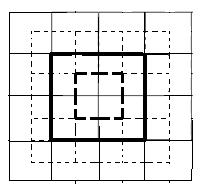
                binz2 = binz1 + 1;

这几句，就是用来计算八个统计区间中心点的坐标的。

在计算前面所讲的统计区间的中心坐标，分配权值之前，我为了处理边缘时程序简洁点，就给那个2\*2\*9的立体直方图外边又包了一层，形成了一个4\*4\*11的三维直方图(示意图如下)，原来的2\*2\*9直方图就是被包在中间的部分。这样，在原来直方图里坐标为(binx=1,biny=2,binz=9)的bin，在新的直方图里坐标为(binx=2,biny=3,binz=10)。

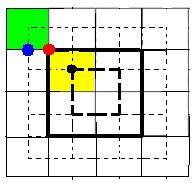


对上面的4\*4\*11的直方图来个与xoy平面平行的剖面图：



粗实线框就是原三维直方图的剖面，也就是一个block，对于像落在粗实线框与粗虚线框之间的点，其最近邻区间是不够8个的，我为了写程序时省点脑力。。。，就用外扩了的这一圈bin，这样落在粗实线框与粗虚线框之间的统计点有了8个区间，用matlab编程时，那个四层for循环中的部分就只用把那八个累积公式写上，也不用判断是不是在落在像上面粗实线框与粗虚线框之间的那种区域。在程序中2\*2\*9的直方图为hist3d，4\*4\*11的直方图为hist3dbig.当在这个hist3dbig中计算都结束后，我把外层这一圈剥去，就是hist3d了。

有了这些准备，我就可以计算出当前像素点的梯度方向幅值应该往hist3dbig中的哪八个bin块累积了。binx1，biny1，binz1 在这里就是那个八个bin块之中离  当前要统计的像素点在直方图中对应的位置  最接近的bin块的下标。binx2，biny2，binz2对应就是最远的bin块的下标了。x1,y1,z1就是bin块(binx1，biny1，binz1)中心点对应的实际像素所在的位置（x1,y1）与梯度方向的角度(z1). 我仍然以原block(即没扩前的block)左上角处作为x1，y1的原点，因为matlab以1作为图像像素索引的开始，我把原点就认为是(1,1)，那(1,1)左边外扩出来的部分，就给以0，-1，-2，-3…这样的坐标,向上也类似，如下图所示，(1,1)位置为红点所示，蓝点处坐标就是(-3,1).



扩展出来的绿块的下标是(binx=1,biny=1,binz1=1),由于像素坐标在红点处为(1,1)，而黄块才是block的第一个cell，对应bin块的下标(2,2).因为下标设计的原因，我在求x1,y1,z1时减了1.5而非0.5.

                x1 = (binx1-1.5)\*cellpw + 0.5;

                y1 = (biny1-1.5)\*cellph + 0.5;

                z1 = (binz1-1.5)\*(or\*pi/nthet);

上面的式子中x1，y1还加了0.5，因为像素坐标是离散的，而第一个坐标总是从1开始，这样对如图中第一个cell的中心（黑点）处应该是4.5. z1没加0.5,是因为角度值是从0开始的，并且是连续的。

在signed（即梯度方向从0度到360度）情况下，因为实际上角度的投票区间是首尾相接环形的，若统计间隔是40度，那么0-40度和320-360度就是相邻区间，那么在4\*4\*11的直方图中，投给binz==11区间(相当于360-380度)的值应该返给binz==2(0-40度)，投给binz==1区间的值应该返给binz==10区间，如4\*4\*11直方图中所示，对应在程序中就是

    if or == 2

        hist3dbig(:,:,2) = hist3dbig(:,:,2) + hist3dbig(:,:,nthet+2);

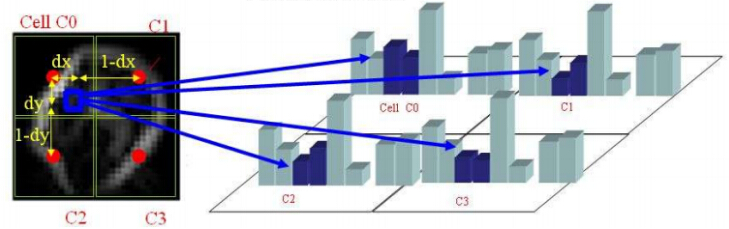
        hist3dbig(:,:,(nthet+1)) = hist3dbig(:,:,(nthet+1)) + hist3dbig(:,:,1);

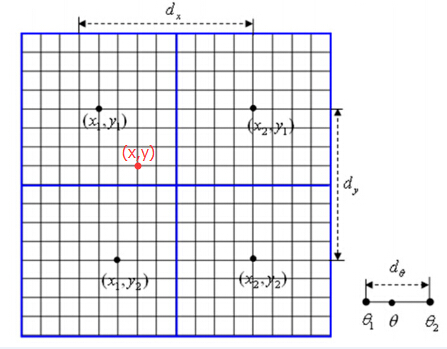
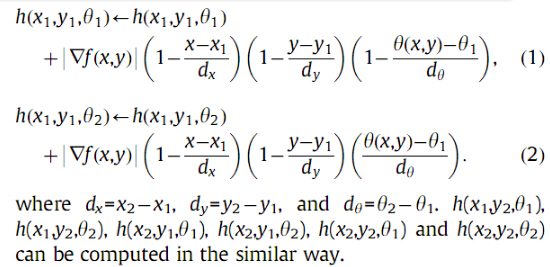
    end

网上的另外一种理解：http://www.cnblogs.com/wyuzl/p/6792216.html

http://blog.csdn.net/u011448029/article/details/11709443 ([HOG特征中的三线插值法](http://blog.csdn.net/u011448029/article/details/11709443))

## Tri-linear interpolation

在计算每个cell的梯度直方图时，可以用三线性插值来提高计算速率。对于每个cell里的点，我们认为都是一个三维向量(x,y,θ), 如下图所示某一待处理像素点它位于block中的C0单元中， 利用该点与四个cell中的中心像素点 （图中4个圆点，(x1,y1),...,(x4,y4)） 的距离计算权值， 将待处理像素点的梯度幅值分别加权累加到C0、C1、 C2、 C3中相应的直方图上，与θ相邻的两个bin上（θ1，θ2）。   
  


  
参见上图，以(x,y,θ)对cell0的梯度直方图即h(x1,y1,θ1)的加权投影为例，三线性插值公式如下：   
  
这样不断累加更新得到最终的h(x1,y1,θ1),...,h(x4,y4,θ4)，就是我们所要4个cell的梯度直方图。

最后这四个cell的直方图hist 就是由

h( x1 , y1, [ θ1 θ2 θ3 ... θ(din个数)] ),

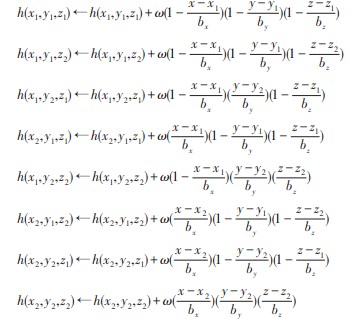
h( x1 , y2, [ θ1 θ2 θ3 ... θ(din个数)] ),

h( x2 , y1, [ θ1 θ2 θ3 ... θ(din个数)] ),

h( x2 , y2, [ θ1 θ2 θ3 ... θ(din个数)] ),

在两个位置坐标(x,y) 和一个方向坐标( θ )上进行三线性插值，关键要解决的问题是应该在哪些 bin上进行加权累加，累加时权值又是多少。将一个像素点处的梯度幅值加权分配到4 个cell 中与该点梯度方向相邻的2 个bin上。按照下面的公式修正直方图向量：

(每个像素 对周围cell中的bin的影响)



### 2.4 把细胞单元组合成大的区间(grouping the cells together into larger blocks)

由于局部光照的变化（variations of illumination）以及前景-背景对比度（foreground-background contrast）的变化，使得梯度强度（gradient strengths）的变化范围非常大。这就需要对梯度强度做归一化，作者采取的办法是：把各个细胞单元组合成大的、空间上连通的区间（blocks）。这样以来，HOG描述器就变成了由各区间所有细胞单元的直方图成分所组成的一个向量。这些区间是互有重叠的，这就意味着：每一个细胞单元的输出都多次作用于最终的描述器。区间有两个主要的几何形状——矩形区间（R-HOG）和环形区间（C-HOG）。R-HOG区间大体上是一些方形的格子，它可以有三个参数来表征：每个区间中细胞单元的数目、每个细胞单元中像素点的数目、每个细胞的直方图通道数目。作者通过实验表明，行人检测的最佳参数设置是：3×3细胞/区间、6×6像素/细胞、9个直方图通道。作者还发现，在对直方图做处理之前，给每个区间（block）加一个高斯空域窗口（Gaussian spatial window）是非常必要的，因为这样可以降低边缘的周围像素点（pixels around the edge）的权重。

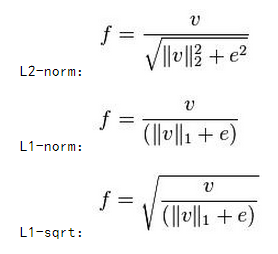
R-HOG跟SIFT描述器看起来很相似，但他们的不同之处是：R-HOG是在单一尺度下、密集的网格内、没有对方向排序的情况下被计算出来（are computed in dense grids at some single scale without orientation alignment）；而SIFT描述器是在多尺度下、稀疏的图像关键点上、对方向排序的情况下被计算出来（are computed at sparse, scale-invariant key image points and are rotated to align orientation）。补充一点，R-HOG是各区间被组合起来用于对空域信息进行编码（are used in conjunction to encode spatial form information），而SIFT的各描述器是单独使用的（are used singly）。

C-HOG区间（blocks）有两种不同的形式，它们的区别在于：一个的中心细胞是完整的，一个的中心细胞是被分割的。如右图所示：

作者发现C-HOG的这两种形式都能取得相同的效果。C-HOG区间（blocks）可以用四个参数来表征：角度盒子的个数（number of angular bins）、半径盒子个数（number of radial bins）、中心盒子的半径（radius of the center bin）、半径的伸展因子（expansion factor for the radius）。通过实验，对于行人检测，最佳的参数设置为：4个角度盒子、2个半径盒子、中心盒子半径为4个像素、伸展因子为2。前面提到过，对于R-HOG，中间加一个高斯空域窗口是非常有必要的，但对于C-HOG，这显得没有必要。C-HOG看起来很像基于形状上下文（Shape Contexts）的方法，但不同之处是：C-HOG的区间中包含的细胞单元有多个方向通道（orientation channels），而基于形状上下文的方法仅仅只用到了一个单一的边缘存在数（edge presence count）。

### 2.5 区间归一化（Block normalization）

作者采用了四中不同的方法对区间进行归一化，并对结果进行了比较。引入v表示一个还没有被归一化的向量，它包含了给定区间（block）的所有直方图信息。| | vk | |表示v的k阶范数，这里的k去1、2。用e表示一个很小的常数。这时，归一化因子可以表示如下：



(上面的e就是)

L1范数是指向量中各个元素绝对值之和 ||W||1

 L2范数是指向量各元素的平方和然后求平方根 ||W||2

* **向量范数**

1-范数：

||x||_1 = \sum_{i=1}^N|x_i|，即向量元素绝对值之和，matlab调用函数norm(x, 1) 。

2-范数：

||\textbf{x}||_2 =\sqrt{\sum_{i=1}^Nx_i^2}，Euclid范数（欧几里得范数，常用计算向量长度），即向量元素绝对值的平方和再开方，matlab调用函数norm(x, 2)。

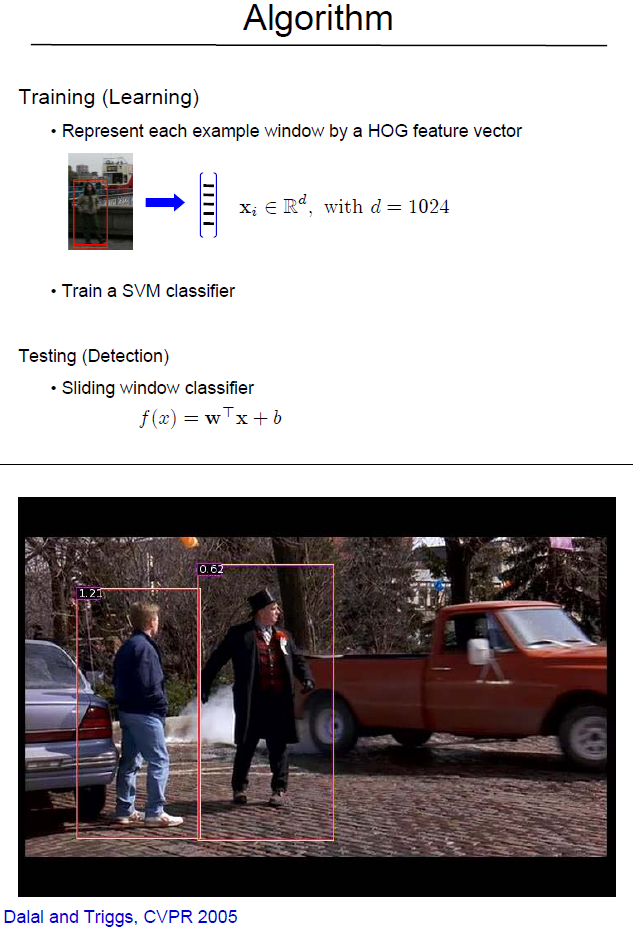
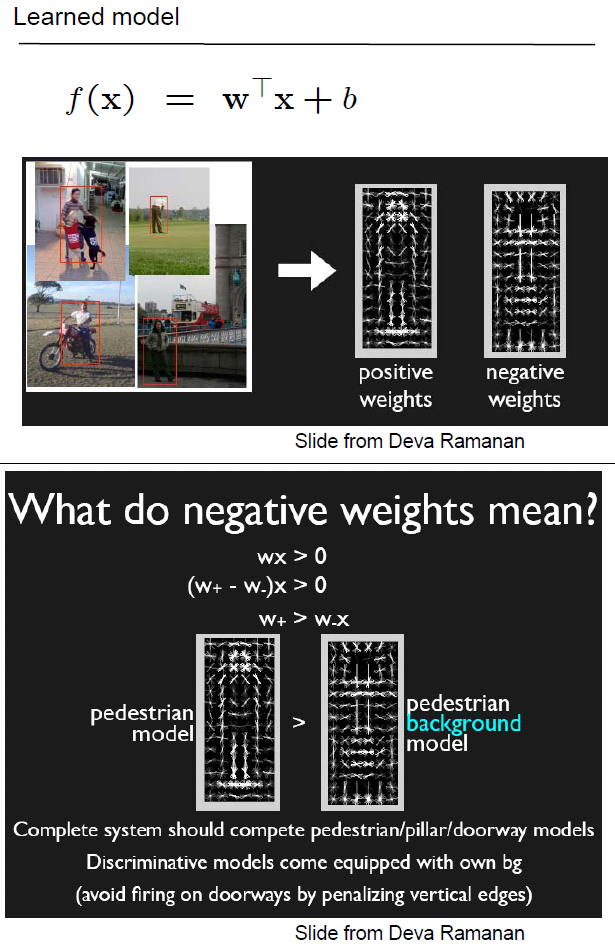
还有第四种归一化方式：L2-Hys，它可以通过先进行L2-norm，对结果进行截短（clipping），然后再重新归一化得到。作者发现：采用L2-Hys, L2-norm, 和 L1-sqrt方式所取得的效果是一样的，L1-norm稍微表现出一点点不可靠性。但是对于没有被归一化的数据来说，这四种方法都表现出来显著的改进。

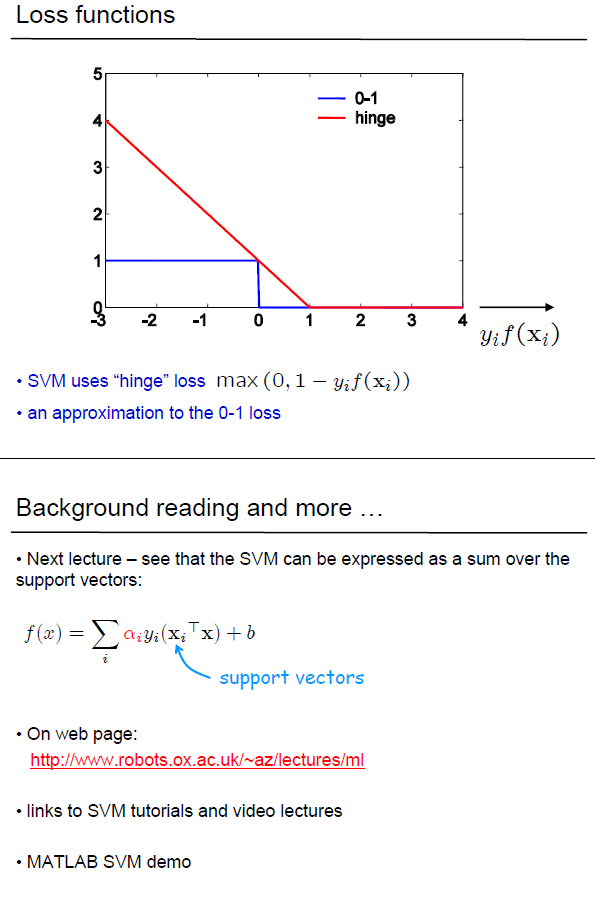
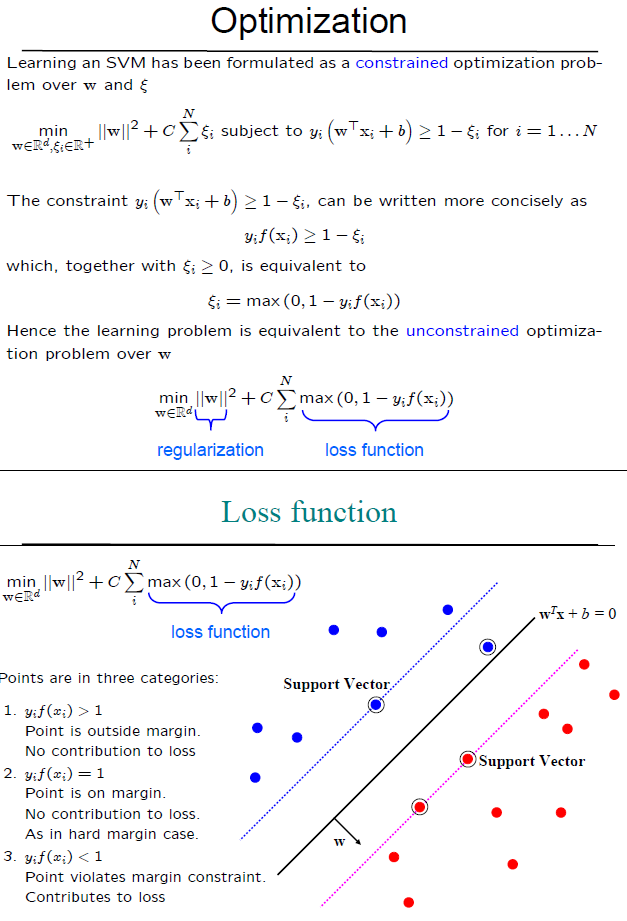
过拟合。至于过拟合是什么，上面也解释了，就是模型训练时候的误差很小，但在测试的时候误差很大，也就是我们的模型复杂到可以拟合到我们的所有训练样本了，但在实际预测新的样本的时候，糟糕的一塌糊涂。

### 2.6 SVM分类器（SVM classifier）进行分类

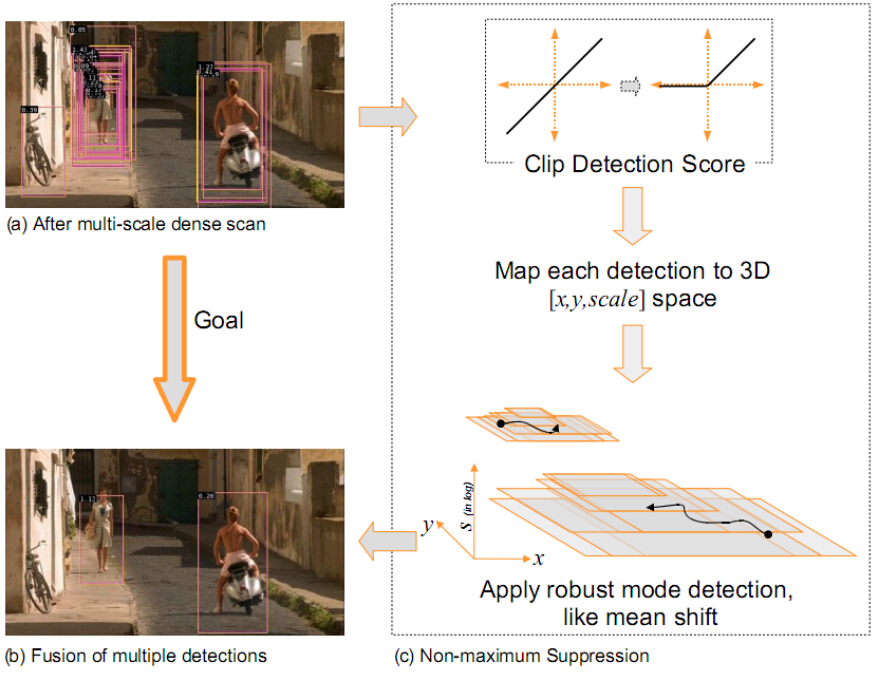
最后一步就是把提取的HOG特征输入到SVM分类器中，寻找一个最优超平面作为决策函数。作者采用的方法是：使用免费的SVMLight软件包加上HOG分类器来寻找测试图像中的行人。

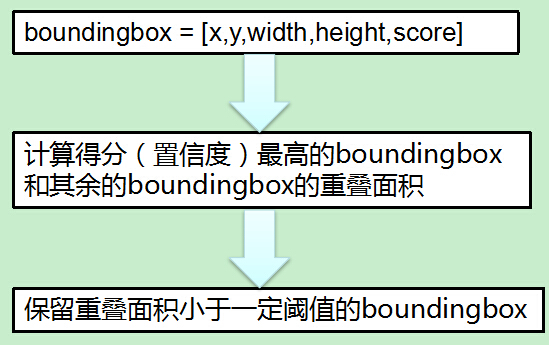
此步的实施可参见SVM的具体操作步骤；



# Multi-Scale Object Localisation

  
上图右上角是对SVM分类结果的置信度做个映射 得到检测评分。检测过程就是用固定大小的窗口 对 多个尺度的图像进行滑窗检测，将多个尺度计算得到的矩形框都还原成原图尺寸，再进行非极大值抑制(NMS,Non-maximum Suppression)处理。在物体检测非极大值抑制应用十分广泛，主要目的是为了消除多余的框，找到最佳的检测框的位置，大致思想如下图所示。 



# Conclusions

HOG的优点：   
- 核心思想是所检测的局部物体外形能够被梯度或边缘方向的分布所描述，HOG能较好地捕捉局部形状信息，对几何和光学变化都有很好的不变性；   
- HOG是在密集采样的图像块中求取的，在计算得到的HOG特征向量中隐含了该块与检测窗口之间的空间位置关系。

矩形HOG和SIFT有些相似的地方，关于SIFT具体看这篇博文[SIFT特征提取分析](http://blog.csdn.net/abcjennifer/article/details/7639681)

HOG的缺陷：   
- 很难处理遮挡问题，人体姿势动作幅度过大或物体方向改变也不易检测（这个问题后来在[DPM](http://blog.csdn.net/masibuaa/article/details/17924671)中采用可变形部件模型的方法得到了改善）；   
- 跟SIFT相比，HOG没有选取主方向，也没有旋转梯度方向直方图，因而本身不具有旋转不变性（较大的方向变化），其旋转不变性是通过采用不同旋转方向的训练样本来实现的；   
- 跟SIFT相比，HOG本身不具有尺度不变性，其尺度不变性是通过缩放检测窗口图像的大小来实现的；   
- 此外，由于梯度的性质，HOG对噪点相当敏感，在实际应用中，在Block和Cell划分之后，对于得到各个像区域中，有时候还会做一次高斯平滑去除噪点。

## HOG算法简单汇总及实现

HOG（Histogram of Oriented Gradient）方向梯度直方图，主要用来提取图像特征，最常用的是结合svm进行行人检测。

算法流程图如下（[这篇论文上的](http://tel.archives-ouvertes.fr/docs/00/39/03/03/PDF/NavneetDalalThesis.pdf)）：

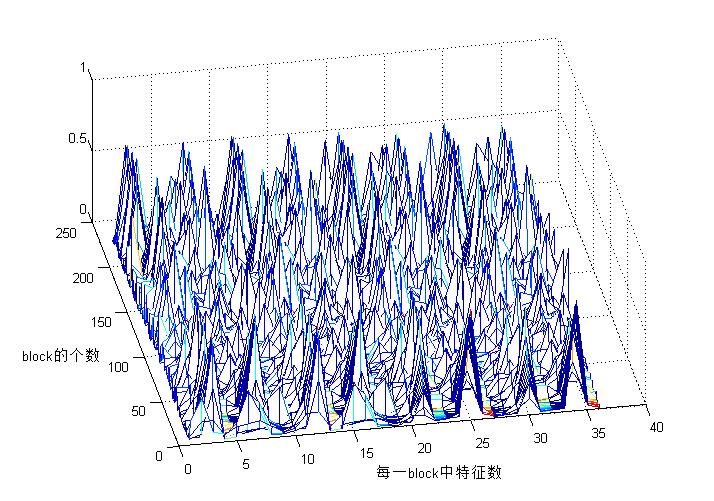
### 3.1 HOG流程简单汇总：

1. 对原图像gamma校正，img=sqrt(img);
2. 求图像竖直边缘，水平边缘，边缘强度，边缘斜率。
3. 将图像每16\*16（取其他也可以）个像素分到一个cell中。对于256\*256的lena来说，就分成了16\*16个cell了。
4. 对于每个cell求其梯度方向直方图。通常取9（取其他也可以）个方向（特征），也就是每360/9=40度分到一个方向，方向大小按像素边缘强度加权。最后归一化直方图。
5. 每2\*2（取其他也可以）个cell合成一个block，所以这里就有（16-1）\*（16-1）=225个block。
6. 所以每个block中都有2\*2\*9个特征，一共有225个block，所以总的特征有225\*36个。
7. 当然一般HOG特征都不是对整幅图像取的，而是对图像中的一个滑动窗口取的。

### 3.2 例图lena图：



求得的225\*36个特征：



### 3.3 HOG原理matlab代码：

clear all; close all; clc;

img=double(imread('lena.jpg'));

imshow(img,[]);

[m n]=size(img);

img=sqrt(img); %伽马校正

%下面是求边缘

fy=[-1 0 1]; %定义竖直模板

fx=fy'; %定义水平模板

Iy=imfilter(img,fy,'replicate'); %竖直边缘

Ix=imfilter(img,fx,'replicate'); %水平边缘

Ied=sqrt(Ix.^2+Iy.^2); %边缘强度

Iphase=Iy./Ix; %边缘斜率，有些为inf,-inf,nan，其中nan需要再处理一下

%下面是求cell

step=16; %step\*step个像素作为一个单元

orient=9; %方向直方图的方向个数

jiao=360/orient; %每个方向包含的角度数

Cell=cell(1,1); %所有的角度直方图,cell是可以动态增加的，所以先设了一个

ii=1;

jj=1;

for i=1:step:m %如果处理的m/step不是整数，最好是i=1:step:m-step

ii=1;

for j=1:step:n %注释同上

tmpx=Ix(i:i+step-1,j:j+step-1);

tmped=Ied(i:i+step-1,j:j+step-1);

tmped=tmped/sum(sum(tmped)); %局部边缘强度归一化

tmpphase=Iphase(i:i+step-1,j:j+step-1);

Hist=zeros(1,orient); %当前step\*step像素块统计角度直方图,就是cell

for p=1:step

for q=1:step

if isnan(tmpphase(p,q))==1 %0/0会得到nan，如果像素是nan，重设为0

tmpphase(p,q)=0;

end

ang=atan(tmpphase(p,q)); %atan求的是[-90 90]度之间

ang=mod(ang\*180/pi,360); %全部变正，-90变270

if tmpx(p,q)<0 %根据x方向确定真正的角度

if ang<90 %如果是第一象限

ang=ang+180; %移到第三象限

end

if ang>270 %如果是第四象限

ang=ang-180; %移到第二象限

end

end

ang=ang+0.0000001; %防止ang为0

Hist(ceil(ang/jiao))=Hist(ceil(ang/jiao))+tmped(p,q); %ceil向上取整，使用边缘强度加权

end

end

Hist=Hist/sum(Hist); %方向直方图归一化

Cell{ii,jj}=Hist; %放入Cell中

ii=ii+1; %针对Cell的y坐标循环变量

end

jj=jj+1; %针对Cell的x坐标循环变量

end

%下面是求feature,2\*2个cell合成一个block,没有显式的求block

[m n]=size(Cell);

feature=cell(1,(m-1)\*(n-1));

for i=1:m-1

for j=1:n-1

f=[];

f=[f Cell{i,j}(:)' Cell{i,j+1}(:)' Cell{i+1,j}(:)' Cell{i+1,j+1}(:)'];

feature{(i-1)\*(n-1)+j}=f;

end

end

%到此结束，feature即为所求

%下面是为了显示而写的

l=length(feature);

f=[];

for i=1:l

f=[f;feature{i}(:)'];

end

figure

mesh(f)

## HOG算法详细分析及实现

http://chrisjmccormick.wordpress.com/2013/05/09/hog-descriptor-in-matlab/

### 4.1 HOG Descriptor in MATLAB

To help in my understanding of the HOG descriptor, as well as to allow me to easily test out modifications to the descriptor, I wrote functions in Octave for computing the HOG descriptor for a detection window.

**HOG Tutorial**

For a tutorial on the HOG descriptor, check out my [HOG tutorial post](http://chrisjmccormick.wordpress.com/2013/05/09/hog-person-detector-tutorial/).

**Source files**

[getHOGDescriptor.m](https://dl.dropboxusercontent.com/u/94180423/getHOGDescriptor.m) - Computes the HOG descriptor for a 66×130 pixel image / detection window. The detection window is actually 64×128 pixels, but an extra pixel is required on all sides for computing the gradients.

[getHistogram.m](https://dl.dropboxusercontent.com/u/94180423/getHistogram.m) – Computes a single 9-bin histogram for a cell. Used by ‘getHOGDescriptor’.

Octave code is compatible with MATLAB, so you should also be able to run these functions in MATLAB.

**Differences with OpenCV implementation**

* OpenCV uses L2 hysteresis for the block normalization.
* OpenCV weights each pixel in a block with a gaussian distribution before normalizing the block.
* The sequence of values produced by OpenCV does not match the order of the values produced by this code.

**Order of values**

You may not need to understand the order of bytes in the final vector in order to work with it, but if you’re curious, here’s a description. The values in the final vector are grouped according to their block. A block consists of 36 values: 1 block  \*  4 cells / block  \* 1 histogram / cell \* 9 values / histogram = 36 values / block. The first 36 values in the vector come from the block in the top left corner of the detection window, and the last 36 values in the vector come from the block in the bottom right. Before unwinding the values to a vector, each block is represented as a 3D dimensional matrix, 2x2x9, corresponding to the four cells in a block with their histogram values in the third dimension. To unwind this matrix into a vector, I use the colon operator ‘:’, e.g., A(:).  You can reshape the values into a 3D matrix using the ‘reshape’ command. For example:

% Get the top left block from the descriptor.

block1 = H(1:36);

% Reshape the values into a 2x2x9 matrix B1.

B1 = reshape(block1, 2, 2, 9);

### 4.2 HOG Person Detector Tutorial

[May 9, 2013](http://chrisjmccormick.wordpress.com/2013/05/09/hog-person-detector-tutorial/) · by [Chris McCormick](http://chrisjmccormick.wordpress.com/author/chrisjmccormick/) · in [Tutorials](http://chrisjmccormick.wordpress.com/category/tutorials/). ·

One of the most popular and successful “person detectors” out there right now is the HOG with SVM approach. When I attended the Embedded Vision Summit in April 2013, it was the most common algorithm I heard associated with person detection.

HOG stands for Histograms of Oriented Gradients. HOG is a type of “feature descriptor”. The intent of a feature descriptor is to generalize the object in such a way that the same object (in this case a person) produces as close as possible to the same feature descriptor when viewed under different conditions. This makes the classification task easier.

The creators of this approach trained a Support Vector Machine (a type of machine learning algorithm for classification), or “SVM”, to recognize HOG descriptors of people.

The HOG person detector is fairly simple to understand (compared to SIFT object recognition, for example). One of the main reasons for this is that it uses a “global” feature to describe a person rather than a collection of “local” features. Put simply, this means that the entire person is represented by a single feature vector, as opposed to many feature vectors representing smaller parts of the person.

The HOG person detector uses a sliding detection window which is moved around the image. At each position of the detector window, a HOG descriptor is computed for the detection window. This descriptor is then shown to the trained SVM, which classifies it as either “person” or “not a person”.

To recognize persons at different scales, the image is subsampled to multiple sizes. Each of these subsampled images is searched.

**Original Work**

The HOG person detector was introduced by Dalal and Triggs at the CVPR conference in 2005. The original paper is available [here](http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf).

The original training data set is available [here](http://pascal.inrialpes.fr/data/human/).

**Gradient Histograms**

The HOG person detector uses a detection window that is 64 pixels wide by 128 pixels tall.

Below are some of the original images used to train the detector, cropped in to the 64×128 window.

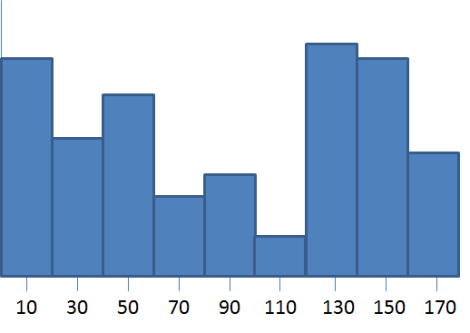
[](http://chrisjmccormick.files.wordpress.com/2013/05/trainingimages.png)

To compute the HOG descriptor, we operate on 8×8 pixel cells within the detection window. These cells will be organized into overlapping blocks, but we’ll come back to that.

Here’s a zoomed-in version of one of the images, with an 8×8 cell drawn in red, to give you an idea of the cell size and image resolution we’re working with.

[](http://chrisjmccormick.files.wordpress.com/2013/05/crop001025a.png)

Within a cell, we compute the gradient vector at each pixel (see my post on [gradient vectors](http://chrisjmccormick.wordpress.com/2013/05/07/gradient-vectors/) if you’re unfamiliar with this concept). We take the 64 gradient vectors (in our 8×8 pixel cell) and put them into a 9-bin histogram. The Histogram ranges from 0 to 180 degrees, so there are 20 degrees per bin.

[](http://chrisjmccormick.files.wordpress.com/2013/05/histogram.png)

Note: Dalal and Triggs used “unsigned gradients” such that the orientations only ranged from 0 to 180 degrees instead of 0 to 360.

For each gradient vector, it’s contribution to the histogram is given by the magnitude of the vector (so stronger gradients have a bigger impact on the histogram). We split the contribution between the two closest bins. So, for example, if a gradient vector has an angle of 85 degrees, then we add 1/4th of its magnitude to the bin centered at 70 degrees, and 3/4ths of its magnitude to the bin centered at 90.

I believe the intent of splitting the contribution is to minimize the problem of gradients which are right on the boundary between two bins. Otherwise, if a strong gradient was right on the edge of a bin, a slight change in the gradient angle (which nudges the gradient into the next bin) could have a strong impact on the histogram.

Why put the gradients into this histogram, rather than simply using the gradient values directly? The gradient histogram is a form of “quantization”, where in this case we are reducing 64 vectors with 2 components each down to a string of just 9 values (the magnitudes of each bin). Compressing the feature descriptor may be important for the performance of the classifier, but I believe the main intent here is actually to generalize the contents of the 8×8 cell.

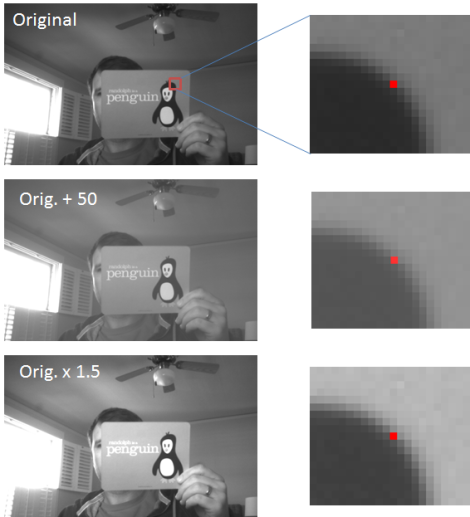
Imagine if you deformed the contents of the 8×8 cell slightly. You might still have roughly the same vectors, but they might be in slightly different positions within the cell and with slightly different angles. The histogram bins allow for some play in the angles of the gradients, and certainly in their positions (the histogram doesn’t encode where each gradient is within the cell, it only encodes the “distribution” of gradients within the cell).

**Normalizing Gradient Vectors**

The next step in computing the descriptors is to normalize the histograms. Let’s take a moment to first look at the effect of normalizing  gradient vectors in general.

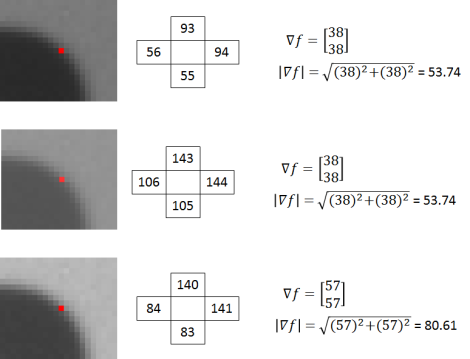
In my post on [gradient vectors](http://chrisjmccormick.wordpress.com/2013/05/07/gradient-vectors/), I show how you can add or subtract a fixed amount of brightness to every pixel in the image, and you’ll still get the same the same gradient vectors at every pixel.

It turns out that by normalizing your gradient vectors, you can also make them invariant to multiplications of the pixel values.  Take a look at the below examples. The first image shows a pixel, highlighted in red, in the original image. In the second image, all pixel values have been increased by 50. In the third image, all pixel values in the original image have been multiplied by 1.5.

[](http://chrisjmccormick.files.wordpress.com/2013/05/multiplication.png)

Notice how the third image displays an increase in contrast. The effect of the multiplication is that bright pixels became much brighter while dark pixels only became a little brighter, thereby increasing the contrast between the light and dark parts of the image.

Let’s look at the actual pixel values and how the gradient vector changes in these three images. The numbers in the boxes below represent the values of the pixels surrounding the pixel marked in red.

[](http://chrisjmccormick.files.wordpress.com/2013/05/magnitudes.png)

The gradient vectors are equivalent in the first and second images, but in the third, the gradient vector magnitude has increased by a factor of 1.5.

If you divide all three vectors by their respective magnitudes, you get the same result for all three vectors: [ 0.71  0.71]‘.

So in the above example we see that by dividing the gradient vectors by their magnitude we can make them invariant (or at least more robust) to changes in contrast.

Dividing a vector by its magnitude is referred to as normalizing the vector to unit length, because the resulting vector has a magnitude of 1. Normalizing a vector does not affect its orientation, only the magnitude.

**Histogram Normalization**

Recall that the value in each of the nine bins in the histogram is based on the magnitudes of the gradients in the 8×8 pixel cell over which it was computed. If every pixel in a cell is multiplied by 1.5, for example, then we saw above that the magnitude of all of the gradients in the cell will be increased by a factor of 1.5 as well. In turn, this means that the value for each bin of the histogram will also be increased by 1.5x. By normalizing the histogram, we can make it invariant to this type of illumination change.

**Block Normalization**

Rather than normalize each histogram individually, the cells are first grouped into blocks and normalized based on all histograms in the block.

The blocks used by Dalal and Triggs consisted of 2 cells by 2 cells. The blocks have “50% overlap”, which is best described through the illustration below.

[](http://chrisjmccormick.files.wordpress.com/2013/05/blocks.png)

This block normalization is performed by concatenating the histograms of the four cells within the block into a vector with 36 components (4 histograms x 9 bins per histogram). Divide this vector by its magnitude to normalize it.

The effect of the block overlap is that each cell will appear multiple times in the final descriptor, but normalized by a different set of neighboring cells. (Specifically, the corner cells appear once, the other edge cells appear twice each, and the interior cells appear four times each).

Honestly, my understanding of the rationale behind the block normalization is still a little shaky. In my earlier normalization example with the penguin flash card, I multiplied every pixel in the image by 1.5, effectively increasing the contrast by the same amount over the whole image. I imagine the rationale in the block normalization approach is that changes in contrast are more likely to occur over smaller regions within the image. So rather than normalizing over the entire image, we normalize within a small region around the cell.

**Final Descriptor Size**

The 64 x 128 pixel detection window will be divided into 7 blocks across and 15 blocks vertically, for a total of 105 blocks. Each block contains 4 cells with a 9-bin histogram for each cell, for a total of 36 values per block. This brings the final vector size to 7 blocks across x 15 blocks vertically x 4 cells per block x 9-bins per histogram = 3,780 values.

**HOG Detector in OpenCV**

OpenCV includes a class for running the HOG person detector on an image.

Check out [this post](http://www.magicandlove.com/blog/2011/08/26/people-detection-in-opencv-again/) for some example code that should get you up and running quickly with the HOG person detector, using a webcam as the video source.

**HOG Descriptor in Octave / MATLAB**

To help in my understanding of the HOG descriptor, as well as to allow me to easily test out modifications to the descriptor, I’ve written a function in Octave for computing the HOG descriptor for a 64×128 image.

As a starting point, I began with the MATLAB code provided by another researcher [here](http://www.mathworks.com/matlabcentral/fileexchange/28689-hog-descriptor-for-matlab). That code doesn’t implement all of the features of the original HOG person detector, though, and didn’t make very effective use of vectorization.

### 4.3 Gradient Vectors

[May 7, 2013](http://chrisjmccormick.wordpress.com/2013/05/07/gradient-vectors/) · by [Chris McCormick](http://chrisjmccormick.wordpress.com/author/chrisjmccormick/) · in [Tutorials](http://chrisjmccormick.wordpress.com/category/tutorials/). ·

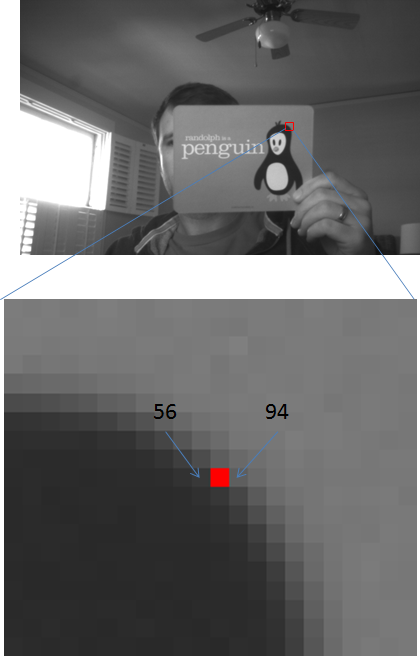
Gradient vectors (or “image gradients”) are one of the most fundamental concepts in computer vision; many vision algorithms involve computing gradient vectors for each pixel in an image.

After a quick introduction to how gradient vectors are computed, I’ll discuss some of its properties which make it so useful.

**Computing The Gradient Image**

A gradient vector can be computed for every pixel an image. It’s simply a measure of the change in pixel values along the x-direction and the y-direction around each pixel.

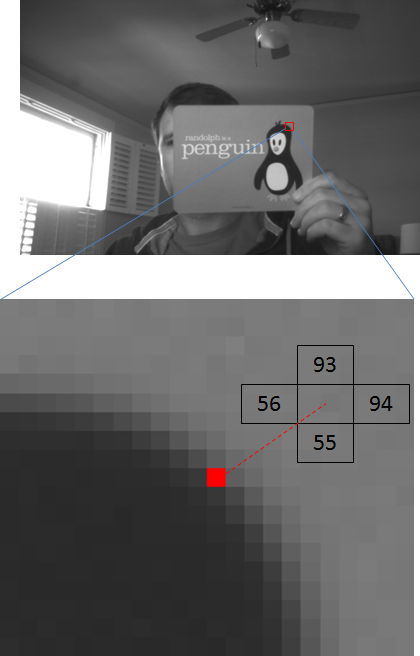
Let’s look at a simple example; let’s say we want to compute the gradient vector at the pixel highlighted in red below.

[](http://chrisjmccormick.files.wordpress.com/2013/05/dxexample.png)

This is a grayscale image, so the pixel values just range from 0 – 255 (0 is black, 255 is white). The pixel values to the left and right of our pixel are marked in the image: 56 and 94. We just take the right value minus the left value and say that the rate of change in the x direction is 38 (94 – 56 = 38).

Note: At this pixel, the pixels from dark to light as we move left to right. If we looked at the same spot on the left side of the penguin’s head where the pixels instead change from light to dark, we’d get a negative value for the change. You can compute the gradient by subtracting left from right or right from left, you just have to be consistent across the image.

We can do the same for the pixels above and below to get the change in the y-direction:

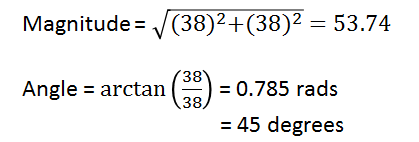
[](http://chrisjmccormick.files.wordpress.com/2013/05/dxdyexample.png)

93 – 55 = 38 in the y-direction.

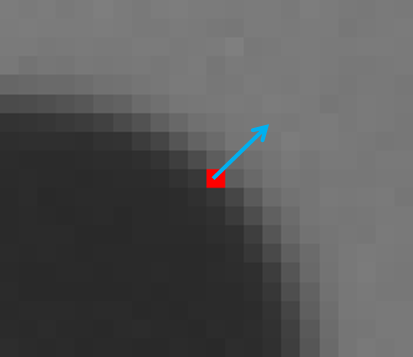
Putting these two values together, we now have our gradient vector.

[vector](http://chrisjmccormick.files.wordpress.com/2013/05/vector.png)

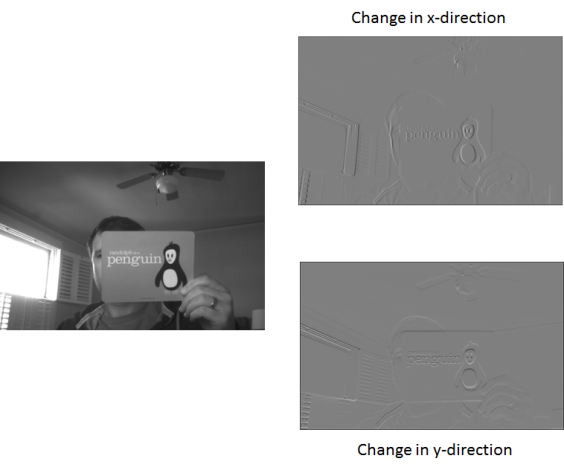
We can also use the equations for the magnitude and angle of a vector to compute those values.

[](http://chrisjmccormick.files.wordpress.com/2013/05/polarcoordinates.png)

We can now draw the gradient vector as an arrow on the image. Notice how the direction of the gradient vector is perpendicular to the edge of the penguin’s head–this is an important property of gradient vectors.

[](http://chrisjmccormick.files.wordpress.com/2013/05/vectorarrow.png)

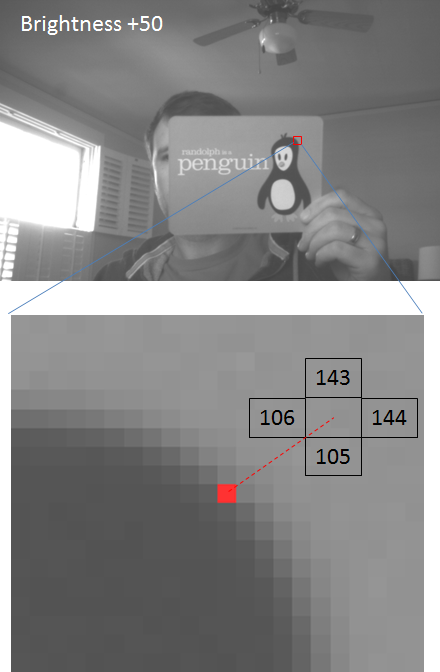
Let’s see what it looks like to compute the change in the x and y direction at every pixel for the image. Note that the difference in pixel values can range from -255 to 255. This is too many values to store in a byte, so we have to map the values to the range 0 – 255. After performing this mapping, pixels with a large negative change will be black, pixels with a large positive change will be white, and pixels with little or no change will be gray.

[](http://chrisjmccormick.files.wordpress.com/2013/05/gradientimage.png)

**Gradient Vector Applications**

The first and most obvious application of gradient vectors is to edge detection. You can see in the gradient images how large gradient values correspond to strong edges in the image.

The other less obvious application is to feature extraction. Look at what happens to the gradient vector when I increase the brightness of the image by adding 50 to all of the pixel values.

[](http://chrisjmccormick.files.wordpress.com/2013/05/brightness50.png)

In this brighter image, the rate of change in the x-direction is still 144 – 106 = 38, and the rate of change in the y-direction is still 143 – 105 = 38, the same as in our original image. So even though the pixel values are all completely different, we still get the same gradient vector at this pixel!

When we base our feature descriptors on gradient vectors instead of just the raw pixel values, we gain some “lighting invariance”. We’ll  compute the same descriptor (or at least closer to the same descriptor) for an object under different lighting conditions, making it easier to recognize the object despite changes in lighting.

**Mathematics**

This is a brief introduction to gradient vectors without much use of the mathematical terms or expressions for what we’re doing. In another post, titled [Image Derivatives](http://chrisjmccormick.wordpress.com/2013/02/26/image-derivative/), I approach the same topic from a more mathematical perspective. The Image Derivatives post is actually my notes on a computer vision lecture given by Professor Shah which is freely available online.

### 4.4 Image Derivative

[February 26, 2013](http://chrisjmccormick.wordpress.com/2013/02/26/image-derivative/) · by [Chris McCormick](http://chrisjmccormick.wordpress.com/author/chrisjmccormick/) · in [Uncategorized](http://chrisjmccormick.wordpress.com/category/uncategorized/). ·

Taking the derivative of an image is a concept that I’ve seen come up both in edge detection and in computing optical flow. It’s confused the heck out of me because I would normally think of derivatives in terms of taking the derivative of a continuous function. However, with an image, you have a 2D matrix of seemingly random values, so what could it mean to take the derivative?

When taking the derivative of an image, you’re actually taking what’s called a discrete derivative, and it’s more of an approximation of the derivative. One simple example is that you can take the derivative in the x-direction at pixel x1 by taking the difference between the pixel values to the left and right of your pixel (x0 and x2).

I think it’s easiest to see how the image derivative is useful in locating edges. The derivative of a function tells you the rate of change, and an edge constitutes a high rate of change.

**Sources**

There are a number of sources I’ve used for learning about this topic.

**University of Central Florida (UCF) Lecture on YouTube**

* [**Lecture 02 – Filtering**](http://www.youtube.com/watch?v=1THuCOKNn6U)
* The discussion of image derivatives starts at about 6:00 and runs till 17:45 in the video.

**University of  Nevada, Reno (UNR) Lecture Slides**

* [Image Processing Fundamentals](http://www.cse.unr.edu/~bebis/CS474/Lectures/SpatialFiltering.ppt)
* The discussion of derivatives goes from slide 35 to the end.

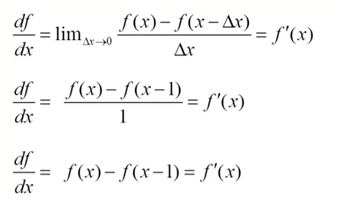
**Wikipedia**

* Prewitt Operator: <http://en.wikipedia.org/wiki/Prewitt_operator>
  + This is the operator Dr. Shah discusses in the YouTube lecture.

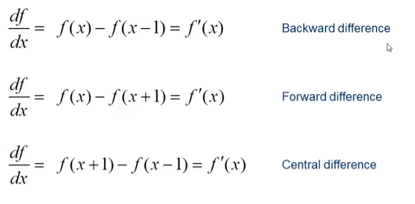
**Notes**

Dr. Shah’s video lecture begins with a quick refresher on the definition of a derivative in Calculus. In calculus we have continuous valued functions, but with images we have discrete data.

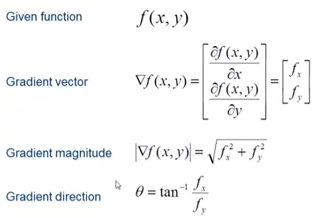
The first equation below shows the calculus definition of a derivative. With image data, the smallest possible delta x is 1, so we use the second and third equations to approximate the derivative.

****

Taking the difference between x and x-1 is just one possibility, and is called the “backward difference”. The other options are:

****

An image actually has three variables, an x-coordinate, a y-coordinate, and an intensity. So the derivative of an image has two dimensions. We can take the derivative in the x direction and in the y direction, and together these make up the “gradient vector”:

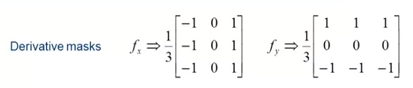
****

[at 13:31 in the video]

Going back to edge detection, the gradiant direction gives you the normal to the edge (it’s perpindicular to the edge), and the gradient magnitude gives you the strength of the edge.

To compute the derivative with respect to x at a given pixel, it sounds like in practice, to reduce the affects of noise, we:

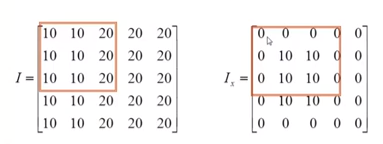
* Use the “central difference”
* Average the derivative of the pixel with that of the row above and row below:

****

[15:07]

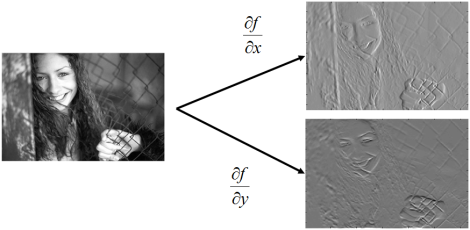
Apply the mask by multiplying each component of the matrix with the corresponding pixel value (the pixel of interest is at the center of the matrix) and sum them all up.

Again, we could just use the middle row for fx and the middle column for fy, but we include the surrounding pixels to help reduce the affect of noise.

We can’t apply the mask to the border pixels so we don’t, the derivative at those pixels is set to 0.  
  
[16:42]

He doesn’t mention this until he talks about edge detectors later on, but the derivative mask he’s using here is called the Prewitt operator, and you can read more about it on Wikipedia: http://en.wikipedia.org/wiki/Prewitt\_operator. The Prewitt operator omitts the 1/3, I’m guessing for the sake of computational efficiency.

Both lectures provide the following image as an example of the derivative with respect to x and y.

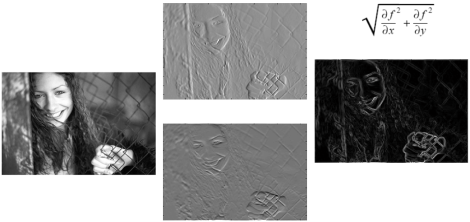
[](http://chrisjmccormick.files.wordpress.com/2013/02/imagederivative.png)

Notice how, in the df/dx image, the vertical boundary between her face and hair is more apparent, or similarly the vertical boundary between her hair and the wall. The df/dy image, on the other hand, accentuates horizontal edges like the side/top of her hand, the edge of her eyes, and her eyebrows.

The significance of the greyscale values in these images had me confused for a while. The reason the images are mostly grey is that the value of the derivative actually ranges from -255 to 255, but to visualize it we must scale this to the range 0 to 255. So anywhere the derivative is zero (no edge), it’s given the value 128 (a neutral grey).

Also notice how, in the df/dx image, a horizontal transition from light to dark (like the side of her face) is colored white, while a dark to light transition (like on her hand) is colored black.

We measure the strength of an edge by combining the df/dx and df/dy values, as shown below.

[](http://chrisjmccormick.files.wordpress.com/2013/02/imagederivative2.png)

Again, the grayscale values in the final image caused me some confusion. If most of the pixel values are about 128, how does the final image end up mostly black? The pixel values in the final image (the gradient magnitude values) are computed using the original derivative values ranging from -255 to 255. So in the final image, areas with no edges are black, and areas with edges (light to dark transitions or dark to light transitions) are colored white.

### 4.5 matlab code

(1) getHOGDescriptor

function H = getHOGDescriptor(img)

% GETHOGDESCRIPTOR computes a HOG descriptor vector for the supplied image.

% H = getHOGDescriptor(img)

%

% This function takes a 130 x 66 pixel gray scale image (128 x 64 with a

% 1-pixel border for computing the gradients at the edges) and computes a

% HOG descriptor for the image, returning a 3,780 value column vector.

%

% Parameters:

% img - A grayscale image matrix with 130 rows and 66 columns.

% Returns:

% A column vector of length 3,780 containing the HOG descriptor.

%

% The intent of this function is to implement the same design choices as

% the original HOG descriptor for human detection by Dalal and Triggs.

% Specifically, I'm using the following parameter choices:

% - 8x8 pixel cells

% - Block size of 2x2 cells

% - 50% overlap between blocks

% - 9-bin histogram

%

% The above parameters give a final descriptor size of

% 7 blocks across x 15 blocks high x 4 cells per block x 9 bins per hist

% = 3,780 values in the final vector.

%

% A couple other important design decisions:

% - Each gradient vector splits its contribution proportionally between the

% two nearest bins

% - For the block normalization, I'm using L2 normalization.

%

% Differences with OpenCV implementation:

% - OpenCV uses L2 hysteresis for the block normalization.

% - OpenCV weights each pixel in a block with a gaussian distribution

% before normalizing the block.

% - The sequence of values produced by OpenCV does not match the order

% of the values produced by this code.

% $Author: ChrisMcCormick $ $Date: 2013/12/04 22:00:00 $ $Revision: 1.2 $

% Revision Notes:

% v1.2

% - Bug fix: Changed call from 'getUnsHistogram' to 'getHistogram'.

% - Replaced 'rows' and 'columns' calls with 'size' for Matlab users.

% v1.1

% - Changed the 'hy' filter mask to match the OpenCV filter mask (it's now

% [-1; 0; 1] instead of [1; 0; -1]).

% - Added a required one-pixel border around the image for computing

% gradients for the edge pixels.

% The number of bins to use in the histograms.

numBins = 9;

% The cells are 8 x 8 pixels.

cellSize = 8;

% Empty vector to store computed descriptor.

H = [];

% Verify the image size is 66 x 130.

[height, width] = size(img);

if ((width ~= 66) || (height ~= 130))

disp("Image size must be 130 x 66 pixels (128x64 with 1px border).\n");

return;

end

% Compute the number cells horizontally and vertically (should be 8 x 16).

numHorizCells = 8;

numVertCells = 16;

% ===============================

% Compute Gradient Vectors

% ===============================

% Compute the gradient vector at every pixel in the image.

% Create the operators for computing image derivative at every pixel.

hx = [-1,0,1];

hy = hx';

% Compute the derivative in the x and y direction for every pixel.

dx = imfilter(double(img), hx);

dy = imfilter(double(img), hy);

% Remove the 1 pixel border.

dx = dx(2 : (size(dx, 1) - 1), 2 : (size(dx, 2) - 1));

dy = dy(2 : (size(dy, 1) - 1), 2 : (size(dy, 2) - 1));

% Convert the gradient vectors to polar coordinates (angle and magnitude).

angles = atan2(dy, dx);

magnit = ((dy.^2) + (dx.^2)).^.5;

% =================================

% Compute Cell Histograms

% =================================

% Compute the histogram for every cell in the image. We'll combine the cells

% into blocks and normalize them later.

% Create a three dimensional matrix to hold the histogram for each cell.

histograms = zeros(numVertCells, numHorizCells, numBins);

% Cast the image to floating point values.

img = double(img);

% For each cell in the y-direction...

for row = 0:(numVertCells - 1)

% Compute the row number in the 'img' matrix corresponding to the top

% of the cells in this row. Add 1 since the matrices are indexed from 1.

rowOffset = (row \* cellSize) + 1;

% For each cell in the x-direction...

for col = 0:(numHorizCells - 1)

% Select the pixels for this cell.

% Compute column number in the 'img' matrix corresponding to the left

% of the current cell. Add 1 since the matrices are indexed from 1.

colOffset = (col \* cellSize) + 1;

% Compute the indices of the pixels within this cell.

rowIndeces = rowOffset : (rowOffset + cellSize - 1);

colIndeces = colOffset : (colOffset + cellSize - 1);

% Select the angles and magnitudes for the pixels in this cell.

cellAngles = angles(rowIndeces, colIndeces);

cellMagnitudes = magnit(rowIndeces, colIndeces);

% Compute the histogram for this cell.

% Convert the cells to column vectors before passing them in.

histograms(row + 1, col + 1, :) = getHistogram(cellMagnitudes(:), cellAngles(:), numBins);

end

end

% ===================================

% Block Normalization

% ===================================

% Take 2 x 2 blocks of cells and normalize the histograms within the block.

% Normalization provides some invariance to changes in contrast, which can

% be thought of as multiplying every pixel in the block by some coefficient.

% For each cell in the y-direction...

for row = 1:(numVertCells - 1)

% For each cell in the x-direction...

for col = 1:(numHorizCells - 1)

% Get the histograms for the cells in this block.

blockHists = histograms(row : row + 1, col : col + 1, :);

% Put all the histogram values into a single vector (nevermind the

% order), and compute the magnitude.

% Add a small amount to the magnitude to ensure that it's never 0.

magnitude = norm(blockHists(:)) + 0.01;

% Divide all of the histogram values by the magnitude to normalize

% them.

normalized = blockHists / magnitude;

% Append the normalized histograms to our descriptor vector.

H = [H; normalized(:)];

end

end

end

//////////////////////////////////////////////////////////////////////////////////

1. getHistogram

function H = getHistogram(magnitudes, angles, numBins)

% GETHISTOGRAM Computes a histogram for the supplied gradient vectors.

% H = getHistogram(magnitudes, angles, numBins)

%

% This function takes the supplied gradient vectors and places them into a

% histogram with 'numBins' based on their unsigned orientation.

%

% "Unsigned" orientation means that, for example, a vector with angle

% -3/4 \* pi will be treated the same as a vector with angle 1/4 \* pi.

%

% Each gradient vector's contribution is split between the two nearest bins,

% in proportion to the distance between the two nearest bin centers.

%

% A gradient's contribution to the histogram is equal to its magnitude;

% the magnitude is divided between the two nearest bin centers.

%

% Parameters:

% magnitudes - A column vector storing the magnitudes of the gradient

% vectors.

% angles - A column vector storing the angles in radians of the

% gradient vectors (ranging from -pi to pi)

% numBins - The number of bins to place the gradients into.

% Returns:

% A row vector of length 'numBins' containing the histogram.

% $Author: ChrisMcCormick $ $Date: 2013/12/04 22:00:00 $ $Revision: 1.2 $

% Revision Notes:

% v1.2

% - Expanded '+=' since this gave Matlab users trouble.

% v1.1

% - The function was actually hardcoded to 9 bins; it now properly supports

% specifying 'numBins'.

% - It now returns an unsigned histogram. This has been shown to improve

% accuracy for person detection.

% Compute the bin size in radians. 180 degress = pi.

binSize = pi / numBins;

% The angle values will range from 0 to pi.

minAngle = 0;

% Make the angles unsigned by adding pi (180 degrees) to all negative angles.

angles(angles < 0) = angles(angles < 0) + pi;

% The gradient angle for each pixel will fall between two bin centers.

% For each pixel, we split the bin contributions between the bin to the left

% and the bin to the right based on how far the angle is from the bin centers.

% For each pixel's gradient vector, determine the indeces of the bins to the

% left and right of the vector's angle.

%

% The histogram needs to wrap around at the edges--vectors on the far edges of

% the histogram (i.e., close to -pi or pi) will contribute partly to the bin

% at that edge, and partly to the bin on the other end of the histogram.

% For vectors with an orientation close to 0 radians, leftBinIndex will be 0.

% Likewise, for vectors with an orientation close to pi radians, rightBinIndex

% will be numBins + 1. We will fix these indeces after we calculate the bin

% contribution amounts.

leftBinIndex = round((angles - minAngle) / binSize);

rightBinIndex = leftBinIndex + 1;

% For each pixel, compute the center of the bin to the left.

leftBinCenter = ((leftBinIndex - 0.5) \* binSize) - minAngle;

% For each pixel, compute the fraction of the magnitude

% to contribute to each bin.

rightPortions = angles - leftBinCenter;

leftPortions = binSize - rightPortions;

rightPortions = rightPortions / binSize;

leftPortions = leftPortions / binSize;

% Before using the bin indeces, we need to fix the '0' and '10' values.

% Recall that the histogram needs to wrap around at the edges--bin "0"

% contributions, for example, really belong in bin 9.

% Replace index 0 with 9 and index 10 with 1.

leftBinIndex(leftBinIndex == 0) = numBins;

rightBinIndex(rightBinIndex == (numBins + 1)) = 1;

% Create an empty row vector for the histogram.

H = zeros(1, numBins);

% For each bin index...

for (i = 1:numBins)

% Find the pixels with left bin == i

pixels = (leftBinIndex == i);

% For each of the selected pixels, add the gradient magnitude to bin 'i',

% weighted by the 'leftPortion' for that pixel.

H(1, i) = H(1, i) + sum(leftPortions(pixels)' \* magnitudes(pixels));

% Find the pixels with right bin == i

pixels = (rightBinIndex == i);

% For each of the selected pixels, add the gradient magnitude to bin 'i',

% weighted by the 'rightPortion' for that pixel.

H(1, i) = H(1, i) + sum(rightPortions(pixels)' \* magnitudes(pixels));

end

end

## Digit Classification Using HOG Features

<http://www.mathworks.com/help/vision/examples/digit-classification-using-hog-features.html>

注: 适用于matlab2013b

[Open this Example](matlab:edit%20HOGDigitClassificationExample)[Open this Example](http://www.mathworks.com/help/vision/examples/rmvd_matlablink__f99044f19efbe76b7b7e54603eff7228.html)

This example shows how to classify digits using HOG features and an SVM classifier.

Object classification is an important task in many computer vision applications, including surveillance, automotive safety, and image retrieval. For example, in an automotive safety application, you may need to classify nearby objects as pedestrians or vehicles. Regardless of the type of object being classified, the basic procedure for creating an object classifier is:

* Acquire a labeled data set with images of the desired object.
* Partition the data set into a training set and a test set.
* Train the classifier using features extracted from the training set.
* Test the classifier using features extracted from the test set.

To illustrate, this example shows how to classify numerical digits using HOG (Histogram of Oriented Gradient) features [1] and an SVM (Support Vector Machine) classifier. This type of classification is often used in many Optical Character Recognition (OCR) applications.

The example uses the svmtrain and svmclassify functions from the Statistics Toolbox™ and the extractHOGFeatures function from the Computer Vision System Toolbox™.

function HOGDigitClassificationExample

### 5.1 Digit Data Set

For training, synthetic images are created using the insertText function from the Computer Vision System Toolbox™. The training images each contain a digit surrounded by other digits, which mimics how digits are normally seen together. Using synthetic images is convenient and it enables the creation of a variety of training samples without having to manually collect them. For testing, scans of handwritten digits are used to validate how well the classifier performs on data that is different than the synthetic training data. Although this is not the most representative data set, there is enough data to train and test a classifier, and show the feasibility of the approach.

% Load training and test data

load('digitDataSet.mat', 'trainingImages', 'trainingLabels', 'testImages');

% Update file name relative to matlabroot

dataSetDir = fullfile(matlabroot,'toolbox','vision','visiondemos');

trainingImages = fullfile(dataSetDir, trainingImages);

testImages = fullfile(dataSetDir, testImages);

trainingImages is a 200-by-10 cell array of training image file names; each column contains both the positive and negative training images for a digit. trainingLabels is a 200-by-10 matrix containing a label for each image in the trainingImage cell array. The labels are logical values indicating whether or not the image is a positive instance or a negative instance for a digit. testImages is a 12-by-10 cell array containing the image file names of the handwritten digit images. There are 12 examples per digit.

% Show training and test samples

figure;

subplot(2,3,1); imshow(trainingImages{3,2});

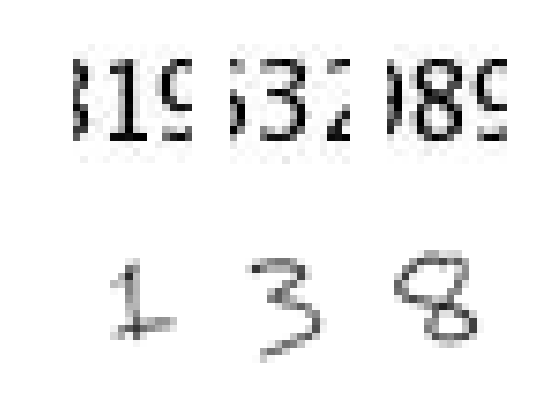
subplot(2,3,2); imshow(trainingImages{23,4});

subplot(2,3,3); imshow(trainingImages{4,9});

subplot(2,3,4); imshow(testImages{2,2});

subplot(2,3,5); imshow(testImages{5,4});

subplot(2,3,6); imshow(testImages{8,9});



Note that prior to training and testing a classifier the following pre-processing step is applied to images from this dataset:

function J = preProcess(I)

lvl = graythresh(I);

J = im2bw(I,lvl);

end

This pre-processing step removes noise artifacts introduced while collecting the image samples and helps provide better feature vectors for training the classifier. For example, the output of this pre-processing step on a couple of training and test images is shown next:

exTestImage = imread(testImages{5,4});

exTrainImage = imread(trainingImages{23,4});

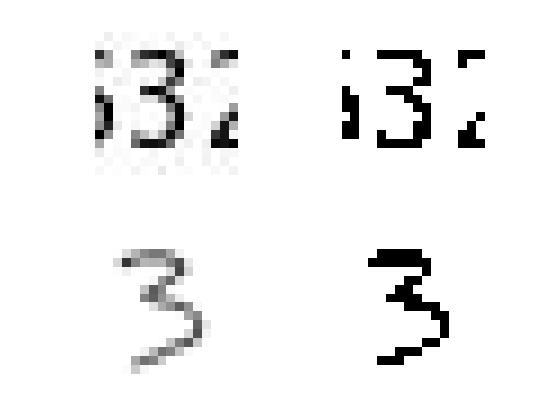
figure;

subplot(2,2,1); imshow(exTrainImage);

subplot(2,2,2); imshow(preProcess(exTrainImage));

subplot(2,2,3); imshow(exTestImage);

subplot(2,2,4); imshow(preProcess(exTestImage));



### 5.2 Using HOG Features

The data used to train the SVM classifier are HOG feature vectors extracted from the training images. Therefore, it is important to make sure the HOG feature vector encodes the right amount of information about the object. The extractHOGFeatures function returns a visualization output that can help form some intuition about just what the "right amount of information" means. By varying the HOG cell size parameter and visualizing the result, you can see the effect the cell size parameter has on the amount of shape information encoded in the feature vector:

img = imread(trainingImages{4,3});

% Extract HOG features and HOG visualization

[hog\_2x2, vis2x2] = extractHOGFeatures(img,'CellSize',[2 2]);

[hog\_4x4, vis4x4] = extractHOGFeatures(img,'CellSize',[4 4]);

[hog\_8x8, vis8x8] = extractHOGFeatures(img,'CellSize',[8 8]);

% Show the original image

figure;

subplot(2,3,1:3); imshow(img);

% Visualize the HOG features

subplot(2,3,4);

plot(vis2x2);

title({'CellSize = [2 2]'; ['Feature length = ' num2str(length(hog\_2x2))]});

subplot(2,3,5);

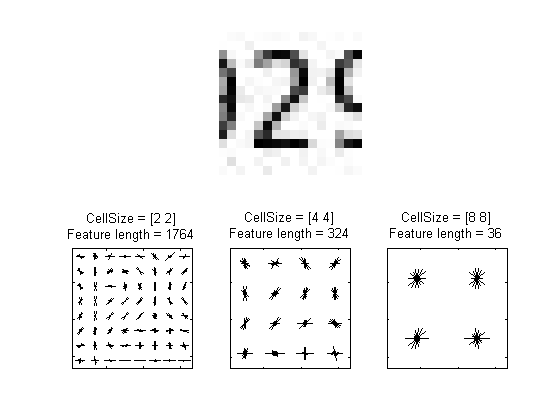
plot(vis4x4);

title({'CellSize = [4 4]'; ['Feature length = ' num2str(length(hog\_4x4))]});

subplot(2,3,6);

plot(vis8x8);

title({'CellSize = [8 8]'; ['Feature length = ' num2str(length(hog\_8x8))]});



The visualization shows that a cell size of [8 8] does not encode much shape information, while a cell size of [2 2] encodes a lot of shape information but increases the dimensionality of the HOG feature vector significantly. A good compromise is a 4-by-4 cell size. This size setting encodes enough spatial information to visually identify a digit shape while limiting the number of dimensions in the HOG feature vector, which helps speed up training. In practice, the HOG parameters should be varied with repeated classifier training and testing to identify the optimal parameter settings.

cellSize = [4 4];

hogFeatureSize = length(hog\_4x4);

### 5.3 Train the Classifier

Digit classification is a multi-class classification problem, where you have to classify an object into one out of the ten possible digit classes. The SVM algorithm in the Statistics Toolbox™, however, produces a binary classifier, which means that it is able to classify an object into one of two classes. In order to use a binary SVM for digit classification, 10 such classifiers are required; each one trained for a specific digit. This is a common technique used to solve multi-class classification problems with binary classifiers and is known as "one-versus-all" or "one-versus-rest" classification.

% Train an SVM classifier for each digit

digits = char('0'):char('9');

for d = 1:numel(digits)

% Pre-allocate trainingFeatures array

numTrainingImages = size(trainingImages,1);

trainingFeatures = zeros(numTrainingImages,hogFeatureSize,'single');

% Extract HOG features from each training image. trainingImages

% contains both positive and negative image samples.

for i = 1:numTrainingImages

img = imread(trainingImages{i,d});

img = preProcess(img);

trainingFeatures(i,:) = extractHOGFeatures(img,'CellSize',cellSize);

end

% Train a classifier for a digit. Each row of trainingFeatures contains

% the HOG features extracted for a single training image. The

% trainingLabels indicate if the features are extracted from positive

% (true) or negative (false) training images.

svm(d) = svmtrain(trainingFeatures, trainingLabels(:,d));

end

### 5.4 Test the Classifier

Now the SVM classifiers can be tested using the handwritten digit images shown earlier.

% Run each SVM classifier on the test images

for d = 1:numel(digits)

% Pre-allocate testFeatures array

numImages = size(testImages,1);

testFeatures = zeros(numImages, hogFeatureSize, 'single');

% Extract features from each test image

for i = 1:numImages

img = imread(testImages{i,d});

img = preProcess(img);

testFeatures(i,:) = extractHOGFeatures(img,'CellSize',cellSize);

end

% Run all the SVM classifiers

for digit = 1:numel(svm)

predictedLabels(:,digit,d) = svmclassify(svm(digit), testFeatures);

end

end

### 5.5 Results

Tabulate the classification results for each SVM classifier.

displayTable(predictedLabels)

digit | svm(0) svm(1) svm(2) svm(3) svm(4) svm(5) svm(6) svm(7) svm(8) svm(9)

---------------------------------------------------------------------------------------------------

0 | 6 0 0 0 0 0 6 0 2 0

1 | 3 10 0 0 0 0 0 2 0 0

2 | 0 2 8 0 0 0 1 1 0 0

3 | 0 0 0 7 0 0 4 0 0 0

4 | 0 0 0 0 9 0 0 0 0 1

5 | 0 0 0 0 0 4 7 0 1 0

6 | 0 0 0 0 2 0 6 0 3 0

7 | 0 0 0 1 0 0 0 5 0 1

8 | 0 0 0 1 0 0 0 1 5 2

9 | 0 1 0 1 1 1 0 0 0 2

The columns of the table contain the classification results for each SVM classifier. Ideally, the table would be a diagonal matrix, where each diagonal element equals the number of images per digit (12 in this example). Based on this data set, digit 1, 2, 3, and 4 are easier to recognize compared to digit 6, where there are many false positives. Using more representative data sets like MNIST [2] or SVHN [3], which contain thousands of handwritten characters, is likely to produce a better classifier compared with the one created using this example data set.

### 5.6 Summary

This example illustrated the basic procedure for creating an object classifier using the extractHOGfeatures function from the Computer Vision System Toolbox and the svmclassify and svmtrain functions from the Statistics Toolbox™. Although HOG features and SVM classifiers were used here, other features and machine learning algorithms can be used in the same way. For instance, you can explore using different feature types for training the classifier; or you can see the effect of using other machine learning algorithms available in the Statistics Toolbox™ such as k-nearest neighbors.

### 5.7 References

[1] Dr. Edgar Seemann. Computer Vision:Histograms of Oriented Gradients[ppt]

[2] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection", Proc. IEEE Conf. Computer Vision and Pattern Recognition, vol. 1, pp. 886-893, 2005.

[3] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86, 2278-2324.

[4] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.

### 5.8 Appendix - Helper functions

function displayTable(labels)

colHeadings = arrayfun(@(x)sprintf('svm(%d)',x),0:9,'UniformOutput',false);

format = repmat('%-9s',1,11);

header = sprintf(format,'digit |',colHeadings{:});

fprintf('\n%s\n%s\n',header,repmat('-',size(header)));

for idx = 1:numel(digits)

fprintf('%-9s', [digits(idx) ' |']);

fprintf('%-9d', sum(labels(:,:,idx)));

fprintf('\n')

end

end

end

## 6. 参考文献

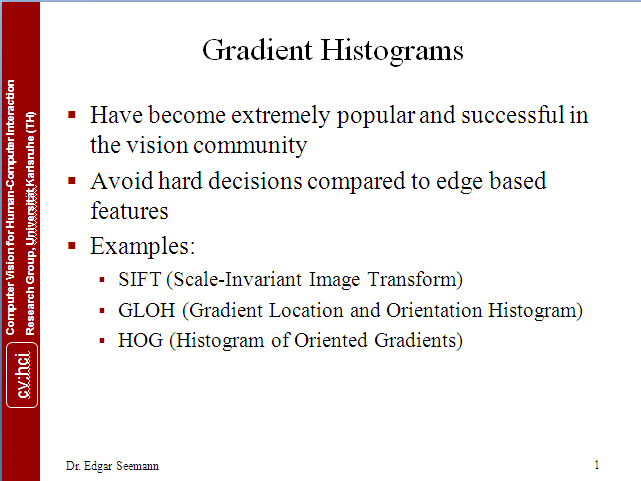
[1] <http://blog.csdn.net/pp5576155/article/details/7023709>

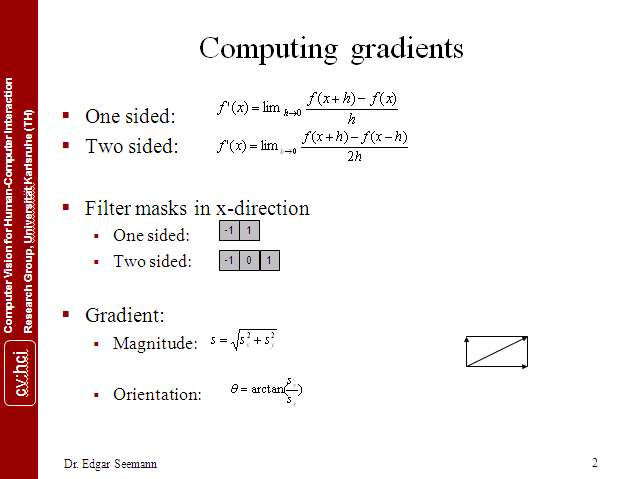
[2] Navneet Dalal and Bill Triggs.Histograms of Oriented Gradients for Human Detection, 2005

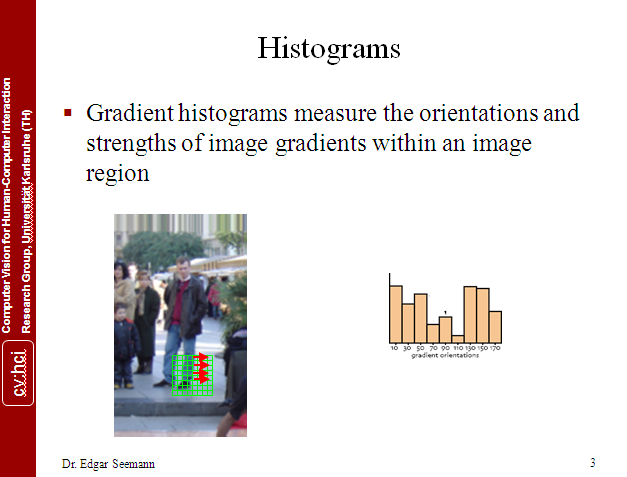
[3] http://wenku.baidu.com/view/4902089b51e79b8968022673.html

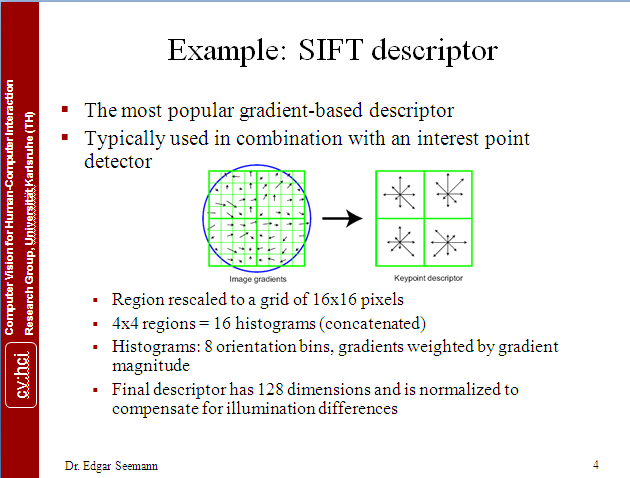
[4]http://www.robots.ox.ac.uk/~az/lectures/ml/lect2.pdf

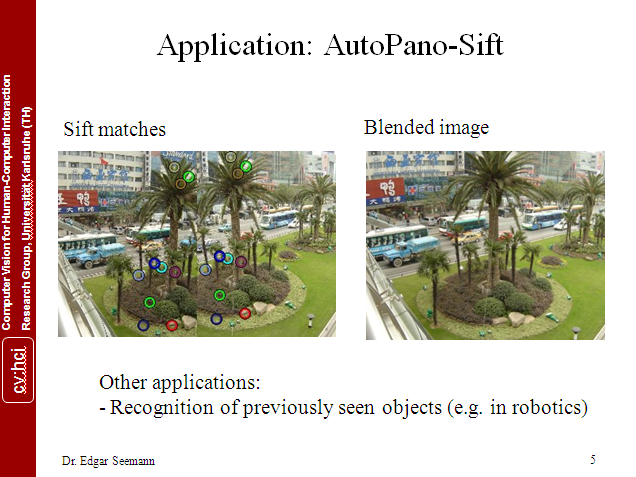
## 附录A: HOG特征提取matlab代码

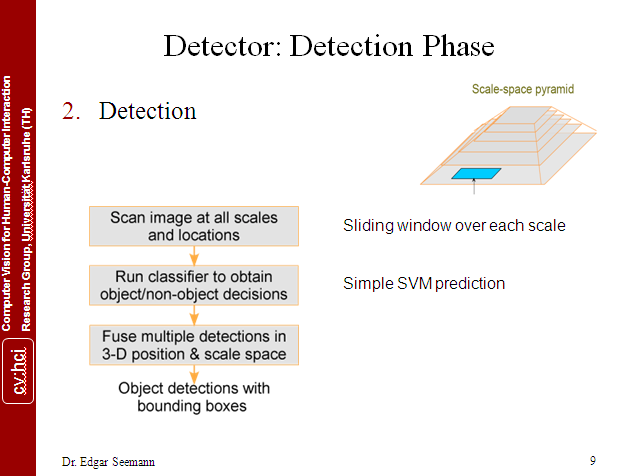
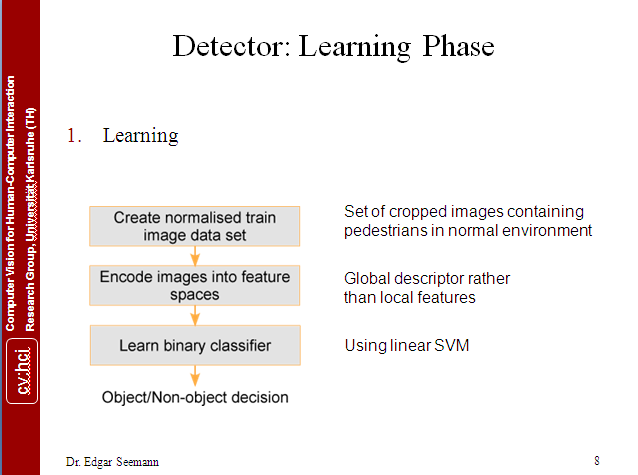
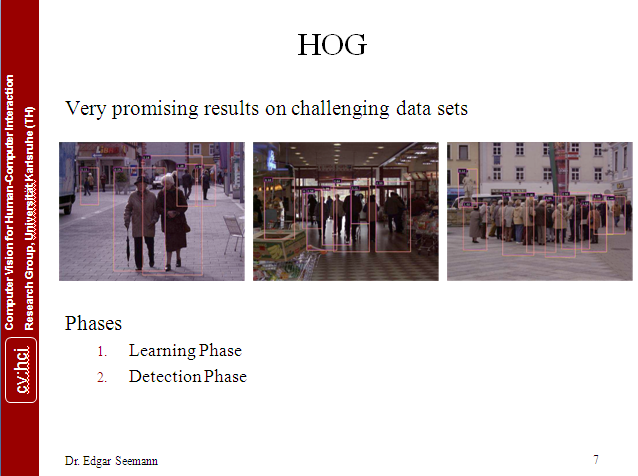
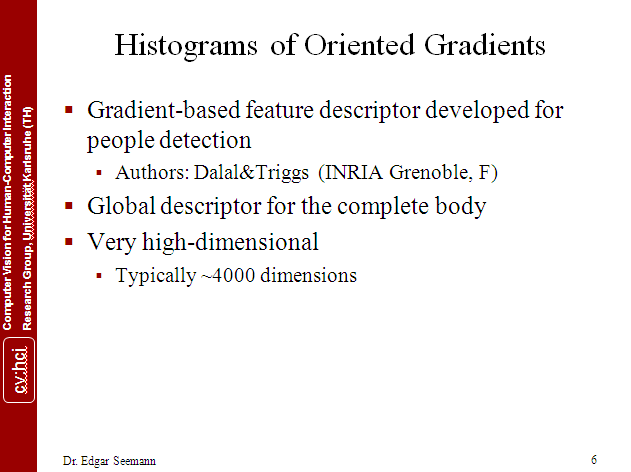




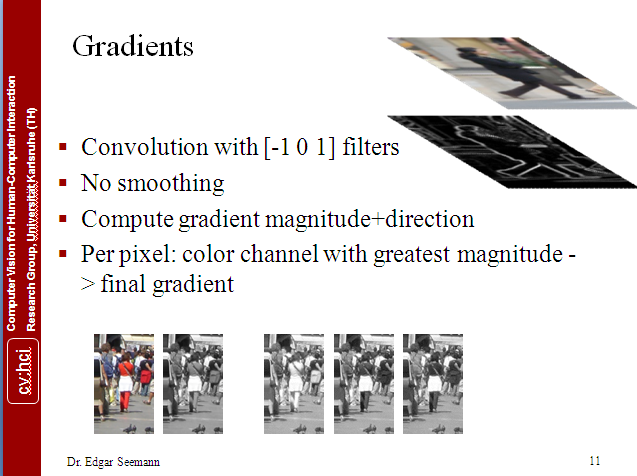


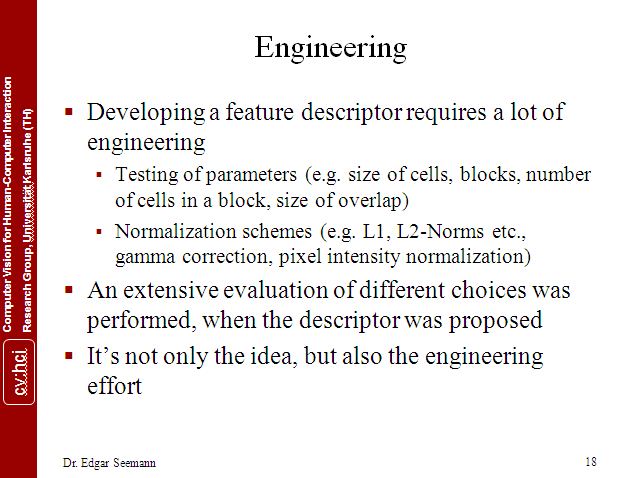
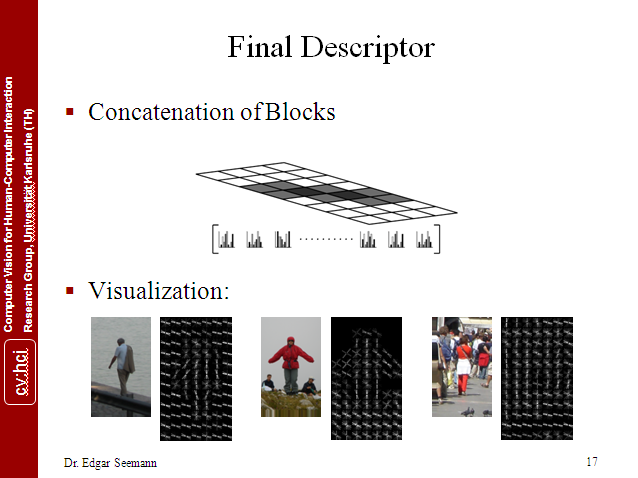
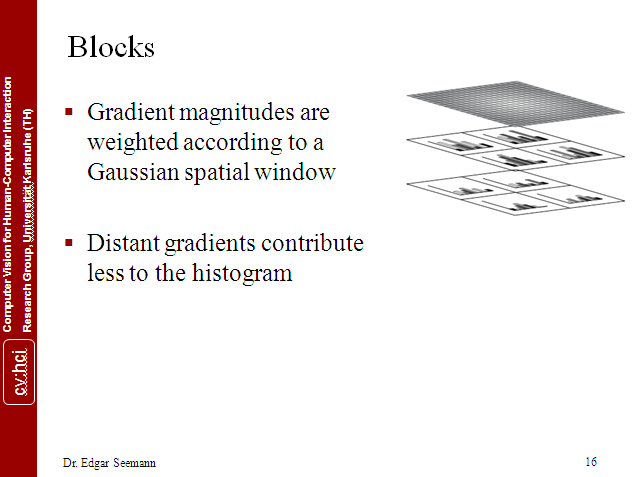
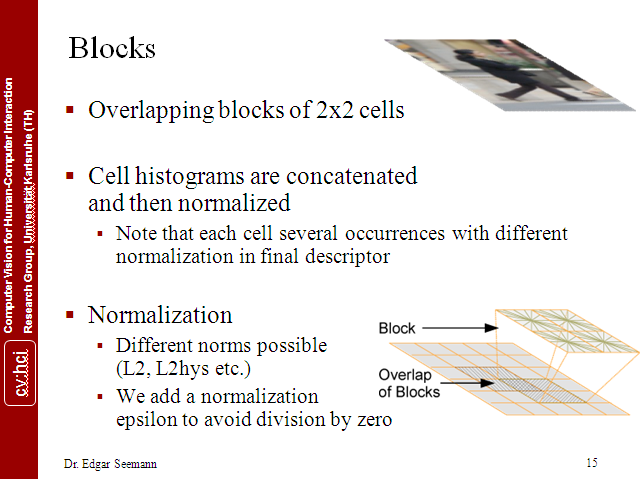
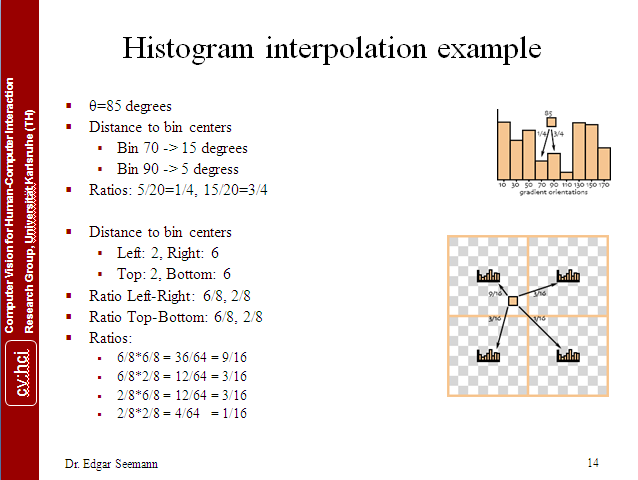
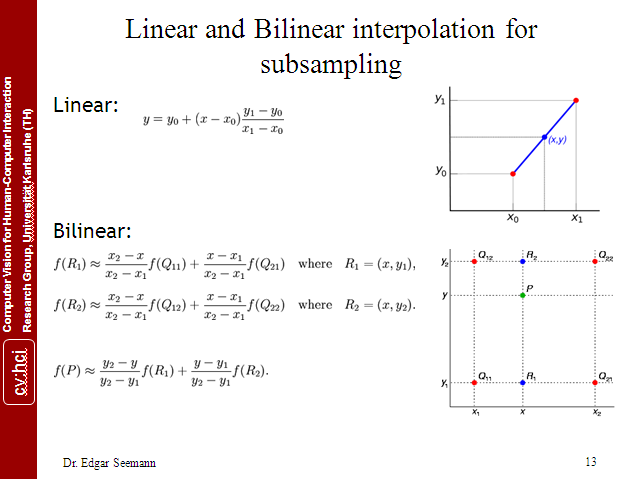
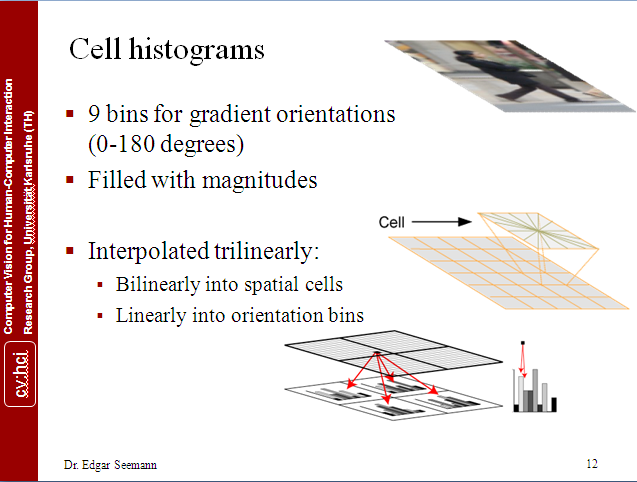


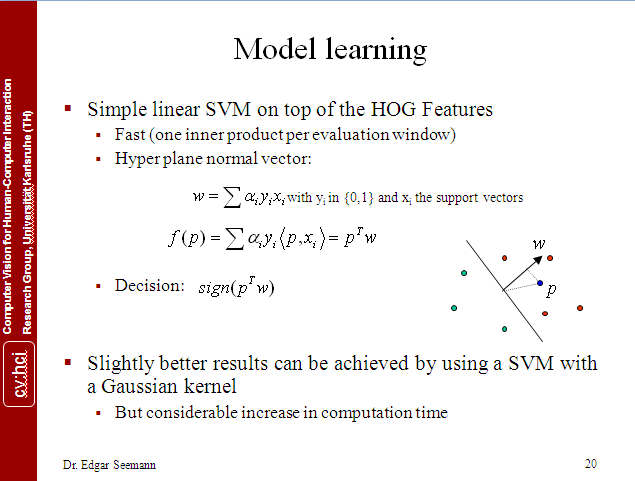












## 附录B: HOG特征提取matlab代码

|  |  |  |
| --- | --- | --- |
| 001 | **function** F = hogcalculator(img, cellpw, cellph, nblockw, nblockh,... | |
| 002 | nthet, overlap, isglobalinterpolate, issigned, normmethod) |

|  |  |  |
| --- | --- | --- |
| 003 | % HOGCALCULATOR calculate R-HOG feature vector of an input image using the | |
| 004 | % procedure presented in Dalal and Triggs's paper in CVPR 2005. |

|  |  |
| --- | --- |
| 005 | % |
| 006 |  | |

|  |  |
| --- | --- |
| 007 | % Author:   timeHandle |
| 008 | % Time:     March 24, 2010 | |

|  |  |  |
| --- | --- | --- |
| 009 | %           May 12，2010 update. | |
| 010 | % |

|  |  |
| --- | --- |
| 011 | %       this copy of code is written for my personal interest, which is an |
| 012 | %       original and inornate realization of [Dalal CVPR2005]'s algorithm |

|  |  |
| --- | --- |
| 013 | %       without any optimization. I just want to check whether I understand |
| 014 | %       the algorithm really or not, and also do some practices for knowing |

|  |  |
| --- | --- |
| 015 | %       matlab programming more well because I could be called as 'novice'. |
| 016 | %       OpenCV 2.0 has realized Dalal's HOG algorithm which runs faster |

|  |  |
| --- | --- |
| 017 | %       than mine without any doubt, ╮(╯▽╰)╭ . Ronan pointed a error in |
| 018 | %       the code，thanks for his correction. Note that at the end of this |

|  |  |  |
| --- | --- | --- |
| 019 | %       code, there are some demonstration code，please remove in your work. | |
| 020 |  |

|  |  |
| --- | --- |
| 021 | % |
| 022 | % F = hogcalculator(img, cellpw, cellph, nblockw, nblockh, | |

|  |  |  |
| --- | --- | --- |
| 023 | %    nthet, overlap, isglobalinterpolate, issigned, normmethod) | |
| 024 | % |

|  |  |
| --- | --- |
| 025 | % IMG: |
| 026 | %       IMG is the input image. | |

|  |  |
| --- | --- |
| 027 | % |
| 028 | % CELLPW, CELLPH: | |

|  |  |  |
| --- | --- | --- |
| 029 | %       CELLPW and CELLPH are cell's pixel width and height respectively. | |
| 030 | % |

|  |  |
| --- | --- |
| 031 | % NBLOCKW, NBLCOKH: |
| 032 | %       NBLOCKW and NBLCOKH are block size counted by cells number in x and | |

|  |  |  |
| --- | --- | --- |
| 033 | %       y directions respectively. | |
| 034 | % |

|  |  |
| --- | --- |
| 035 | % NTHET, ISSIGNED: |
| 036 | %       NTHET is the number of the bins of the histogram of oriented | |

|  |  |
| --- | --- |
| 037 | %       gradient. The histogram of oriented gradient ranges from 0 to pi in |
| 038 | %       'unsigned' condition while to 2\*pi in 'signed' condition, which can |

|  |  |  |
| --- | --- | --- |
| 039 | %       be specified through setting the value of the variable ISSIGNED by | |
| 040 | %       the string 'unsigned' or 'signed'. |

|  |  |
| --- | --- |
| 041 | % |
| 042 | % OVERLAP: | |

|  |  |  |
| --- | --- | --- |
| 043 | %       OVERLAP is the overlap proportion of two neighboring block. | |
| 044 | % |

|  |  |
| --- | --- |
| 045 | % ISGLOBALINTERPOLATE: |
| 046 | %       ISGLOBALINTERPOLATE specifies whether the trilinear interpolation | |

|  |  |  |
| --- | --- | --- |
| 047 | %       is done in a single global 3d histogram of the whole detecting | |
| 048 | %       window by the string 'globalinterpolate' or in each local 3d |

|  |  |
| --- | --- |
| 049 | %       histogram corresponding to respective blocks by the string |
| 050 | %       'localinterpolate' which is in strict accordance with the procedure | |

|  |  |
| --- | --- |
| 051 | %       proposed in Dalal's paper. Interpolating in the whole detecting |
| 052 | %       window requires the block's sliding step to be an integral multiple |

|  |  |
| --- | --- |
| 053 | %       of cell's width and height because the histogram is fixing before |
| 054 | %       interpolate. In fact here the so called 'global interpolation' is |

|  |  |
| --- | --- |
| 055 | %       a notation given by myself. at first the spatial interpolation is |
| 056 | %       done without any relevant to block's slide position, but when I was |

|  |  |
| --- | --- |
| 057 | %       doing calculation while OVERLAP is 0.75, something occurred and |
| 058 | %       confused me o\_\_O"… . This let me find that the operation I firstly |

|  |  |
| --- | --- |
| 059 | %       did is different from which mentioned in Dalal's paper. But this |
| 060 | %       does not mean it is incorrect ^◎^, so I reserve this. As for name, |

|  |  |  |
| --- | --- | --- |
| 061 | %       besides 'global interpolate', any others would be all ok, like 'Lady GaGa' | |
| 062 | %       or what else, :-). |

|  |  |
| --- | --- |
| 063 | % |
| 064 | % NORMMETHOD： | |

|  |  |  |
| --- | --- | --- |
| 065 | %       NORMMETHOD is the block histogram normalized method which can be | |
| 066 | %       set as one of the following strings: |

|  |  |
| --- | --- |
| 067 | %               'none', which means non-normalization; |
| 068 | %               'l1', which means L1-norm normalization; | |

|  |  |
| --- | --- |
| 069 | %               'l2', which means L2-norm normalization; |
| 070 | %               'l1sqrt', which means L1-sqrt-norm normalization; | |

|  |  |  |
| --- | --- | --- |
| 071 | %               'l2hys', which means L2-hys-norm normalization. | |
| 072 | % F： |

|  |  |
| --- | --- |
| 073 | %       F is a row vector storing the final histogram of all of the blocks |
| 074 | %       one by one in a top-left to bottom-right image scan manner, the |

|  |  |  |
| --- | --- | --- |
| 075 | %       cells histogram are stored in the same manner in each block's | |
| 076 | %       section of F. |

|  |  |
| --- | --- |
| 077 | % |
| 078 | % Note that CELLPW\*NBLOCKW and CELLPH\*NBLOCKH should be equal to IMG's | |

|  |  |  |
| --- | --- | --- |
| 079 | % width and height respectively. | |
| 080 | % |

|  |  |
| --- | --- |
| 081 | % Here is a demonstration, which all of parameters are set as the |
| 082 | % best value mentioned in Dalal's paper when the window detected is 128\*64 | |

|  |  |
| --- | --- |
| 083 | % size(128 rows, 64 columns): |
| 084 | %       F = hogcalculator(window, 8, 8, 2, 2, 9, 0.5, | |

|  |  |  |
| --- | --- | --- |
| 085 | %                               'localinterpolate', 'unsigned', 'l2hys'); | |
| 086 | % Also the function can be called like: |

|  |  |
| --- | --- |
| 087 | %       F = hogcalculator(window); |
| 088 | % the other parameters are all set by using the above-mentioned "dalal's | |

|  |  |  |
| --- | --- | --- |
| 089 | % best value" as default. | |
| 090 | % |

|  |  |
| --- | --- |
| 091 |  |
| 092 | **if** nargin < 2 | |

|  |  |  |
| --- | --- | --- |
| 093 | % set default parameters value. | |
| 094 | cellpw = 8; |

|  |  |
| --- | --- |
| 095 | cellph = 8; |
| 096 | nblockw = 2; | |

|  |  |  |
| --- | --- | --- |
| 097 | nblockh = 2; | |
| 098 | nthet = 9; |

|  |  |
| --- | --- |
| 099 | overlap = 0.5; |
| 100 | isglobalinterpolate = 'localinterpolate'; | |

|  |  |  |
| --- | --- | --- |
| 101 | issigned = 'unsigned'; | |
| 102 | normmethod = 'l2hys'; |

|  |  |
| --- | --- |
| 103 | **else** |
| 104 | **if** nargin < 10 | |

|  |  |  |
| --- | --- | --- |
| 105 | error('Input parameters are not enough.'); | |
| 106 | **end** |

|  |  |  |
| --- | --- | --- |
| 107 | **end** | |
| 108 |  |

|  |  |  |
| --- | --- | --- |
| 109 | % check parameters's validity. | |
| 110 | [M, N, K] = size(img); |

|  |  |
| --- | --- |
| 111 | **if** mod(M,cellph\*nblockh) ~= 0 |
| 112 | error('IMG''s height should be an integral multiple of CELLPH\*NBLOCKH.'); | |

|  |  |
| --- | --- |
| 113 | **end** |
| 114 | **if** mod(N,cellpw\*nblockw) ~= 0 | |

|  |  |  |
| --- | --- | --- |
| 115 | error('IMG''s width should be an integral multiple of CELLPW\*NBLOCKW.'); | |
| 116 | **end** |

|  |  |  |
| --- | --- | --- |
| 117 | **if** mod((1-overlap)\*cellpw\*nblockw, cellpw) ~= 0 ||... | |
| 118 | mod((1-overlap)\*cellph\*nblockh, cellph) ~= 0 |

|  |  |
| --- | --- |
| 119 | str1 = 'Incorrect OVERLAP or ISGLOBALINTERPOLATE parameter'; |
| 120 | str2 = ', slide step should be an intergral multiple of cell size'; | |

|  |  |  |
| --- | --- | --- |
| 121 | error([str1, str2]); | |
| 122 | **end** |

|  |  |
| --- | --- |
| 123 |  |
| 124 | % set the standard deviation of gaussian spatial weight window. | |

|  |  |  |
| --- | --- | --- |
| 125 | delta = cellpw\*nblockw \* 0.5; | |
| 126 |  |

|  |  |  |
| --- | --- | --- |
| 127 | % calculate gradient scale matrix. | |
| 128 | hx = [-1,0,1]; |

|  |  |
| --- | --- |
| 129 | hy = -hx'; |
| 130 | gradscalx = imfilter(double(img),hx); | |

|  |  |  |
| --- | --- | --- |
| 131 | gradscaly = imfilter(double(img),hy); | |
| 132 | **if** K > 1 |

|  |  |
| --- | --- |
| 133 | gradscalx = max(max(gradscalx(:,:,1),gradscalx(:,:,2)), gradscalx(:,:,3)); |
| 134 | gradscaly = max(max(gradscaly(:,:,1),gradscaly(:,:,2)), gradscaly(:,:,3)); |

|  |  |
| --- | --- |
| 135 | **end** |
| 136 | gradscal = sqrt(double(gradscalx.\*gradscalx + gradscaly.\*gradscaly)); | |

|  |  |
| --- | --- |
| 137 |  |
| 138 | % calculate gradient orientation matrix. | |

|  |  |
| --- | --- |
| 139 | % plus small number for avoiding dividing zero. |
| 140 | gradscalxplus = gradscalx+ones(size(gradscalx))\*0.0001; | |

|  |  |
| --- | --- |
| 141 | gradorient = zeros(M,N); |
| 142 | % unsigned situation: orientation region is 0 to pi. | |

|  |  |  |
| --- | --- | --- |
| 143 | **if** strcmp(issigned, 'unsigned') == 1 | |
| 144 | gradorient =... |

|  |  |  |
| --- | --- | --- |
| 145 | atan(gradscaly./gradscalxplus) + pi/2; | |
| 146 | or = 1; |

|  |  |
| --- | --- |
| 147 | **else** |
| 148 | % signed situation: orientation region is 0 to 2\*pi. | |

|  |  |
| --- | --- |
| 149 | **if** strcmp(issigned, 'signed') == 1 |
| 150 | idx = find(gradscalx >= 0 & gradscaly >= 0); | |

|  |  |  |
| --- | --- | --- |
| 151 | gradorient(idx) = atan(gradscaly(idx)./gradscalxplus(idx)); | |
| 152 | idx = find(gradscalx < 0); |

|  |  |  |
| --- | --- | --- |
| 153 | gradorient(idx) = atan(gradscaly(idx)./gradscalxplus(idx)) + pi; | |
| 154 | idx = find(gradscalx >= 0 & gradscaly < 0); |

|  |  |  |
| --- | --- | --- |
| 155 | gradorient(idx) = atan(gradscaly(idx)./gradscalxplus(idx)) + 2\*pi; | |
| 156 | or = 2; |

|  |  |
| --- | --- |
| 157 | **else** |
| 158 | error('Incorrect ISSIGNED parameter.'); | |

|  |  |  |
| --- | --- | --- |
| 159 | **end** | |
| 160 | **end** |

|  |  |
| --- | --- |
| 161 |  |
| 162 | % calculate block slide step. | |

|  |  |
| --- | --- |
| 163 | xbstride = cellpw\*nblockw\*(1-overlap); |
| 164 | ybstride = cellph\*nblockh\*(1-overlap); |

|  |  |
| --- | --- |
| 165 | xbstridend = N - cellpw\*nblockw + 1; |
| 166 | ybstridend = M - cellph\*nblockh + 1; |

|  |  |
| --- | --- |
| 167 |  |
| 168 | % calculate the total blocks number in the window detected, which is | |

|  |  |
| --- | --- |
| 169 | % ntotalbh\*ntotalbw. |
| 170 | ntotalbh = ((M-cellph\*nblockh)/ybstride)+1; | |

|  |  |  |
| --- | --- | --- |
| 171 | ntotalbw = ((N-cellpw\*nblockw)/xbstride)+1; | |
| 172 |  |

|  |  |
| --- | --- |
| 173 | % generate the matrix hist3dbig for storing the 3-dimensions histogram. the |
| 174 | % matrix covers the whole image in the 'globalinterpolate' condition or |

|  |  |  |
| --- | --- | --- |
| 175 | % covers the local block in the 'localinterpolate' condition. The matrix is | |
| 176 | % bigger than the area where it covers by adding additional elements |

|  |  |  |
| --- | --- | --- |
| 177 | % (corresponding to the cells) to the surround for calculation convenience. | |
| 178 | **if** strcmp(isglobalinterpolate, 'globalinterpolate') == 1 |

|  |  |
| --- | --- |
| 179 | ncellx = N / cellpw; |
| 180 | ncelly = M / cellph; |

|  |  |
| --- | --- |
| 181 | hist3dbig = zeros(ncelly+2, ncellx+2, nthet+2); |
| 182 | F = zeros(1, (M/cellph-1)\*(N/cellpw-1)\*nblockw\*nblockh\*nthet); | |

|  |  |  |
| --- | --- | --- |
| 183 | glbalinter = 1; | |
| 184 | **else** |

|  |  |  |
| --- | --- | --- |
| 185 | **if** strcmp(isglobalinterpolate, 'localinterpolate') == 1 | |
| 186 | hist3dbig = zeros(nblockh+2, nblockw+2, nthet+2); |

|  |  |  |
| --- | --- | --- |
| 187 | F = zeros(1, ntotalbh\*ntotalbw\*nblockw\*nblockh\*nthet); | |
| 188 | glbalinter = 0; |

|  |  |
| --- | --- |
| 189 | **else** |
| 190 | error('Incorrect ISGLOBALINTERPOLATE parameter.') | |

|  |  |  |
| --- | --- | --- |
| 191 | **end** | |
| 192 | **end** |

|  |  |
| --- | --- |
| 193 |  |
| 194 | % generate the matrix for storing histogram of one block; | |

|  |  |  |
| --- | --- | --- |
| 195 | sF = zeros(1, nblockw\*nblockh\*nthet); | |
| 196 |  |

|  |  |
| --- | --- |
| 197 | % generate gaussian spatial weight. |
| 198 | [gaussx, gaussy] = meshgrid(0:(cellpw\*nblockw-1), 0:(cellph\*nblockh-1)); | |

|  |  |
| --- | --- |
| 199 | weight = exp(-((gaussx-(cellpw\*nblockw-1)/2)... |
| 200 | .\*(gaussx-(cellpw\*nblockw-1)/2)+(gaussy-(cellph\*nblockh-1)/2)... | |

|  |  |  |
| --- | --- | --- |
| 201 | .\*(gaussy-(cellph\*nblockh-1)/2))/(delta\*delta)); | |
| 202 |  |

|  |  |
| --- | --- |
| 203 | % vote for histogram. there are two situations according to the interpolate |
| 204 | % condition('global' interpolate or local interpolate). The hist3d which is |

|  |  |  |
| --- | --- | --- |
| 205 | % generated from the 'bigger' matrix hist3dbig is the final histogram. | |
| 206 | **if** glbalinter == 1 |

|  |  |
| --- | --- |
| 207 | xbstep = nblockw\*cellpw; |
| 208 | ybstep = nblockh\*cellph; |

|  |  |
| --- | --- |
| 209 | **else** |
| 210 | xbstep = xbstride; | |

|  |  |  |
| --- | --- | --- |
| 211 | ybstep = ybstride; | |
| 212 | **end** |

|  |  |
| --- | --- |
| 213 | % block slide loop |
| 214 | **for** btly = 1:ybstep:ybstridend | |

|  |  |
| --- | --- |
| 215 | **for** btlx = 1:xbstep:xbstridend |
| 216 | **for** bi = 1:(cellph\*nblockh) | |

|  |  |  |
| --- | --- | --- |
| 217 | **for** bj = 1:(cellpw\*nblockw) | |
| 218 |  |

|  |  |
| --- | --- |
| 219 | i = btly + bi - 1; |
| 220 | j = btlx + bj - 1; |

|  |  |  |
| --- | --- | --- |
| 221 | gaussweight = weight(bi,bj); | |
| 222 |  |

|  |  |
| --- | --- |
| 223 | gs = gradscal(i,j); |
| 224 | go = gradorient(i,j); | |

|  |  |
| --- | --- |
| 225 |  |
| 226 | **if** glbalinter == 1 | |

|  |  |
| --- | --- |
| 227 | jorbj = j; |
| 228 | iorbi = i; |

|  |  |
| --- | --- |
| 229 | **else** |
| 230 | jorbj = bj; | |

|  |  |  |
| --- | --- | --- |
| 231 | iorbi = bi; | |
| 232 | **end** |

|  |  |
| --- | --- |
| 233 |  |
| 234 | % calculate bin index of hist3dbig | |

|  |  |
| --- | --- |
| 235 | binx1 = floor((jorbj-1+cellpw/2)/cellpw) + 1; |
| 236 | biny1 = floor((iorbi-1+cellph/2)/cellph) + 1; |

|  |  |  |
| --- | --- | --- |
| 237 | binz1 = floor((go+(or\*pi/nthet)/2)/(or\*pi/nthet)) + 1; | |
| 238 |  |

|  |  |
| --- | --- |
| 239 | **if** gs == 0 |
| 240 | **continue**; | |

|  |  |  |
| --- | --- | --- |
| 241 | **end** | |
| 242 |  |

|  |  |
| --- | --- |
| 243 | binx2 = binx1 + 1; |
| 244 | biny2 = biny1 + 1; |

|  |  |  |
| --- | --- | --- |
| 245 | binz2 = binz1 + 1; | |
| 246 |  |

|  |  |
| --- | --- |
| 247 | x1 = (binx1-1.5)\*cellpw + 0.5; |
| 248 | y1 = (biny1-1.5)\*cellph + 0.5; |

|  |  |  |
| --- | --- | --- |
| 249 | z1 = (binz1-1.5)\*(or\*pi/nthet); | |
| 250 |  |

|  |  |
| --- | --- |
| 251 | % trilinear interpolation. |
| 252 | hist3dbig(biny1,binx1,binz1) =... | |

|  |  |
| --- | --- |
| 253 | hist3dbig(biny1,binx1,binz1) + gs\*gaussweight... |
| 254 | \* (1-(jorbj-x1)/cellpw)\*(1-(iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 255 | \*(1-(go-z1)/(or\*pi/nthet)); |
| 256 | hist3dbig(biny1,binx1,binz2) =... | |

|  |  |
| --- | --- |
| 257 | hist3dbig(biny1,binx1,binz2) + gs\*gaussweight... |
| 258 | \* (1-(jorbj-x1)/cellpw)\*(1-(iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 259 | \*((go-z1)/(or\*pi/nthet)); |
| 260 | hist3dbig(biny2,binx1,binz1) =... | |

|  |  |  |
| --- | --- | --- |
| 261 | hist3dbig(biny2,binx1,binz1) + gs\*gaussweight... | |
| 262 | \* (1-(jorbj-x1)/cellpw)\*((iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 263 | \*(1-(go-z1)/(or\*pi/nthet)); |
| 264 | hist3dbig(biny2,binx1,binz2) =... | |

|  |  |  |
| --- | --- | --- |
| 265 | hist3dbig(biny2,binx1,binz2) + gs\*gaussweight... | |
| 266 | \* (1-(jorbj-x1)/cellpw)\*((iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 267 | \*((go-z1)/(or\*pi/nthet)); |
| 268 | hist3dbig(biny1,binx2,binz1) =... | |

|  |  |  |
| --- | --- | --- |
| 269 | hist3dbig(biny1,binx2,binz1) + gs\*gaussweight... | |
| 270 | \* ((jorbj-x1)/cellpw)\*(1-(iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 271 | \*(1-(go-z1)/(or\*pi/nthet)); |
| 272 | hist3dbig(biny1,binx2,binz2) =... | |

|  |  |  |
| --- | --- | --- |
| 273 | hist3dbig(biny1,binx2,binz2) + gs\*gaussweight... | |
| 274 | \* ((jorbj-x1)/cellpw)\*(1-(iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 275 | \*((go-z1)/(or\*pi/nthet)); |
| 276 | hist3dbig(biny2,binx2,binz1) =... | |

|  |  |  |
| --- | --- | --- |
| 277 | hist3dbig(biny2,binx2,binz1) + gs\*gaussweight... | |
| 278 | \* ((jorbj-x1)/cellpw)\*((iorbi-y1)/cellph)... |

|  |  |
| --- | --- |
| 279 | \*(1-(go-z1)/(or\*pi/nthet)); |
| 280 | hist3dbig(biny2,binx2,binz2) =... | |

|  |  |  |
| --- | --- | --- |
| 281 | hist3dbig(biny2,binx2,binz2) + gs\*gaussweight... | |
| 282 | \* ((jorbj-x1)/cellpw)\*((iorbi-y1)/cellph)... |

|  |  |  |
| --- | --- | --- |
| 283 | \*((go-z1)/(or\*pi/nthet)); | |
| 284 | **end** |

|  |  |  |
| --- | --- | --- |
| 285 | **end** | |
| 286 |  |

|  |  |  |
| --- | --- | --- |
| 287 | % In the local interpolate condition. F is generated in this block | |
| 288 | % slide loop. hist3dbig should be cleared in each loop. |

|  |  |  |
| --- | --- | --- |
| 289 | **if** glbalinter == 0 | |
| 290 | **if** or == 2 |

|  |  |  |
| --- | --- | --- |
| 291 | hist3dbig(:,:,2) = hist3dbig(:,:,2)... | |
| 292 | + hist3dbig(:,:,nthet+2); |

|  |  |
| --- | --- |
| 293 | hist3dbig(:,:,(nthet+1)) =... |
| 294 | hist3dbig(:,:,(nthet+1)) + hist3dbig(:,:,1); | |

|  |  |
| --- | --- |
| 295 | **end** |
| 296 | hist3d = hist3dbig(2:(nblockh+1), 2:(nblockw+1), 2:(nthet+1)); | |

|  |  |
| --- | --- |
| 297 | **for** ibin = 1:nblockh |
| 298 | **for** jbin = 1:nblockw | |

|  |  |  |
| --- | --- | --- |
| 299 | idsF = nthet\*((ibin-1)\*nblockw+jbin-1)+1; | |
| 300 | idsF = idsF:(idsF+nthet-1); |

|  |  |  |
| --- | --- | --- |
| 301 | sF(idsF) = hist3d(ibin,jbin,:); | |
| 302 | **end** |

|  |  |
| --- | --- |
| 303 | **end** |
| 304 | iblock = ((btly-1)/ybstride)\*ntotalbw +... | |

|  |  |
| --- | --- |
| 305 | ((btlx-1)/xbstride) + 1; |
| 306 | idF = (iblock-1)\*nblockw\*nblockh\*nthet+1; | |

|  |  |  |
| --- | --- | --- |
| 307 | idF = idF:(idF+nblockw\*nblockh\*nthet-1); | |
| 308 | F(idF) = sF; |

|  |  |  |
| --- | --- | --- |
| 309 | hist3dbig(:,:,:) = 0; | |
| 310 | **end** |

|  |  |  |
| --- | --- | --- |
| 311 | **end** | |
| 312 | **end** |

|  |  |
| --- | --- |
| 313 |  |
| 314 | % In the global interpolate condition. F is generated here outside the | |

|  |  |  |
| --- | --- | --- |
| 315 | % block slide loop | |
| 316 | **if** glbalinter == 1 |

|  |  |
| --- | --- |
| 317 | ncellx = N / cellpw; |
| 318 | ncelly = M / cellph; |

|  |  |
| --- | --- |
| 319 | **if** or == 2 |
| 320 | hist3dbig(:,:,2) = hist3dbig(:,:,2) + hist3dbig(:,:,nthet+2); | |

|  |  |  |
| --- | --- | --- |
| 321 | hist3dbig(:,:,(nthet+1)) = hist3dbig(:,:,(nthet+1)) + hist3dbig(:,:,1); | |
| 322 | **end** |

|  |  |  |
| --- | --- | --- |
| 323 | hist3d = hist3dbig(2:(ncelly+1), 2:(ncellx+1), 2:(nthet+1)); | |
| 324 |  |

|  |  |
| --- | --- |
| 325 | iblock = 1; |
| 326 | **for** btly = 1:ybstride:ybstridend | |

|  |  |
| --- | --- |
| 327 | **for** btlx = 1:xbstride:xbstridend |
| 328 | binidx = floor((btlx-1)/cellpw)+1; | |

|  |  |
| --- | --- |
| 329 | binidy = floor((btly-1)/cellph)+1; |
| 330 | idF = (iblock-1)\*nblockw\*nblockh\*nthet+1; | |

|  |  |  |
| --- | --- | --- |
| 331 | idF = idF:(idF+nblockw\*nblockh\*nthet-1); | |
| 332 | **for** ibin = 1:nblockh |

|  |  |
| --- | --- |
| 333 | **for** jbin = 1:nblockw |
| 334 | idsF = nthet\*((ibin-1)\*nblockw+jbin-1)+1; | |

|  |  |
| --- | --- |
| 335 | idsF = idsF:(idsF+nthet-1); |
| 336 | sF(idsF) = hist3d(binidy+ibin-1, binidx+jbin-1,: ); | |

|  |  |  |
| --- | --- | --- |
| 337 | **end** | |
| 338 | **end** |

|  |  |
| --- | --- |
| 339 | F(idF) = sF; |
| 340 | iblock = iblock + 1; | |

|  |  |  |
| --- | --- | --- |
| 341 | **end** | |
| 342 | **end** |

|  |  |  |
| --- | --- | --- |
| 343 | **end** | |
| 344 |  |

|  |  |
| --- | --- |
| 345 | % adjust the negative value caused by accuracy of floating-point |
| 346 | % operations.these value's scale is very small, usually at E-03 magnitude | |

|  |  |  |
| --- | --- | --- |
| 347 | % while others will be E+02 or E+03 before normalization. | |
| 348 | F(F<0) = 0; |

|  |  |
| --- | --- |
| 349 |  |
| 350 | % block normalization. | |

|  |  |
| --- | --- |
| 351 | e = 0.001; |
| 352 | l2hysthreshold = 0.2; | |

|  |  |  |
| --- | --- | --- |
| 353 | fslidestep = nblockw\*nblockh\*nthet; | |
| 354 | **switch** normmethod |

|  |  |  |
| --- | --- | --- |
| 355 | **case** 'none' | |
| 356 | **case** 'l1' |

|  |  |
| --- | --- |
| 357 | **for** fi = 1:fslidestep:size(F,2) |
| 358 | div = sum(F(fi:(fi+fslidestep-1))); | |

|  |  |  |
| --- | --- | --- |
| 359 | F(fi:(fi+fslidestep-1)) = F(fi:(fi+fslidestep-1))/(div+e); | |
| 360 | **end** |

|  |  |
| --- | --- |
| 361 | **case** 'l1sqrt' |
| 362 | **for** fi = 1:fslidestep:size(F,2) | |

|  |  |
| --- | --- |
| 363 | div = sum(F(fi:(fi+fslidestep-1))); |
| 364 | F(fi:(fi+fslidestep-1)) = sqrt(F(fi:(fi+fslidestep-1))/(div+e)); | |

|  |  |
| --- | --- |
| 365 | **end** |
| 366 | **case** 'l2' | |

|  |  |
| --- | --- |
| 367 | **for** fi = 1:fslidestep:size(F,2) |
| 368 | sF = F(fi:(fi+fslidestep-1)).\*F(fi:(fi+fslidestep-1)); | |

|  |  |
| --- | --- |
| 369 | div = sqrt(sum(sF)+e\*e); |
| 370 | F(fi:(fi+fslidestep-1)) = F(fi:(fi+fslidestep-1))/div; | |

|  |  |
| --- | --- |
| 371 | **end** |
| 372 | **case** 'l2hys' | |

|  |  |
| --- | --- |
| 373 | **for** fi = 1:fslidestep:size(F,2) |
| 374 | sF = F(fi:(fi+fslidestep-1)).\*F(fi:(fi+fslidestep-1)); | |

|  |  |
| --- | --- |
| 375 | div = sqrt(sum(sF)+e\*e); |
| 376 | sF = F(fi:(fi+fslidestep-1))/div; | |

|  |  |  |
| --- | --- | --- |
| 377 | sF(sF>l2hysthreshold) = l2hysthreshold; | |
| 378 | div = sqrt(sum(sF.\*sF)+e\*e); |

|  |  |  |
| --- | --- | --- |
| 379 | F(fi:(fi+fslidestep-1)) = sF/div; | |
| 380 | **end** |

|  |  |
| --- | --- |
| 381 | **otherwise** |
| 382 | error('Incorrect NORMMETHOD parameter.'); | |

|  |  |  |
| --- | --- | --- |
| 383 | **end** | |
| 384 |  |

|  |  |
| --- | --- |
| 385 | % the following code, which can be removed because of having no |
| 386 | % contributions to HOG feature calculation, are just for result |

|  |  |
| --- | --- |
| 387 | % demonstration when the trilinear interpolation is 'global' for this |
| 388 | % condition is easier to give a simple and intuitive illustration. so in |

|  |  |  |
| --- | --- | --- |
| 389 | % 'local' condition it will produce error. | |
| 390 | figure; |

|  |  |
| --- | --- |
| 391 | hold on; |
| 392 | axis equal; | |

|  |  |
| --- | --- |
| 393 | xlim([0, N]); |
| 394 | ylim([0, M]); |

|  |  |
| --- | --- |
| 395 | **for** u = 1:(M/cellph) |
| 396 | **for** v = 1:(N/cellpw) | |

|  |  |
| --- | --- |
| 397 | cx = (v-1)\*cellpw + cellpw/2 + 0.5; |
| 398 | cy = (u-1)\*cellph + cellph/2 + 0.5; |

|  |  |  |  |
| --- | --- | --- | --- |
| 399 | hist3d(u,v,:)=0.9\*min(cellpw,cellph)\*hist3d(u,v,:)/max(hist3d(u,v,:)); | | |
| 400 | | **for** t = 1:nthet |

|  |  |
| --- | --- |
| 401 | s = hist3d(u,v,t); |
| 402 | thet = (t-1)\*pi/nthet + pi\*0.5/nthet; | |

|  |  |
| --- | --- |
| 403 | x1 = cx - s\*0.5\*cos(thet); |
| 404 | x2 = cx + s\*0.5\*cos(thet); |

|  |  |
| --- | --- |
| 405 | y1 = cy - s\*0.5\*sin(thet); |
| 406 | y2 = cy + s\*0.5\*sin(thet); |

|  |  |  |
| --- | --- | --- |
| 407 | plot([x1,x2],[M-y1+1,M-y2+1]); | |
| 408 | **end** |

|  |  |  |
| --- | --- | --- |
| 409 | **end** | |
| 410 | **end** |