SIFT，SURF，ORB，HOG，HARRIS，FAST，RANSAC

图像处理和计算机视觉中用来获取图像局部特征点的算法需要满足几大特性：**可重复性**，同一物体在不同的环境下成像(不同时间、不同角度、不同相机等)，能够检测到同样的特征。**独特性**，特征在某一特定目标上表现为独特性，能够与场景中其他物体相区分，能够达到后续匹配或识别的目的。**局部性**，特征能够刻画图像的局部特性，而且对环境影响因子(光照、噪声等)鲁棒。**紧致性和有效性**，特征能够有效地表达图像信息，而且在实际应用中运算要尽可能地快。

相比于考虑局部邻域范围的**局部特征**，**全局特征**既需要局部特征，也要从整个图像中抽取特征，较多地运用在图像检索领域，例如图像的颜色直方图。

除了以上几点通用的特性外，对于一些图像匹配、检测识别等任务，可能还需进一步考虑图像的局部不变特征。**平移不变性，尺度不变性和旋转不变性**，当图像中的物体或目标发生旋转或者尺度发生变换，依然可以有效地检测或识别。此外，也会考虑局部特征对**光照、阴影的不变性**。

HOG：满足平移不变性，对光照阴影不敏感。不满足旋转不变性和尺度不变性。

Harris：满足平移不变性，旋转不变性，对光照阴影不敏感，但不满足尺度不变性。需要用多尺度 Harris 角点检测。

FAST：满足平移不变性。不满足旋转不变性，不满足尺度不变性。

SIFT：满足平移不变性，旋转不变性，尺度不变性，对光照阴影不敏感。

ORB：满足平移不变性，旋转不变性，尺度不变性。

RANSAC：特征匹配。

SIFT，SURF 和 ORB 既包含特征提取，也包含特征描述。HOG，HARRIS，FAST 只包含特征提取。

HOG：总的来说，是提取出图片的边角信息，勾勒出物体的形象，以此来分类。是一种特征检测和描述算法。

HOG 算法步骤：

1. 预处理输入的图片，灰度化和伽马矫正，可选可不选

2. 计算像素点的梯度值

计算图像$I(x,y)$在X方向和Y方向的梯度

$$I_x = \frac{\partial I}{\partial x} = I(x,y) \otimes \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$$

$$I_y = \frac{\partial I}{\partial y} = I(x,y) \otimes \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}^T$$

$\otimes$ 代表卷积

然后计算总的梯度强度值 g 和梯度方向θ

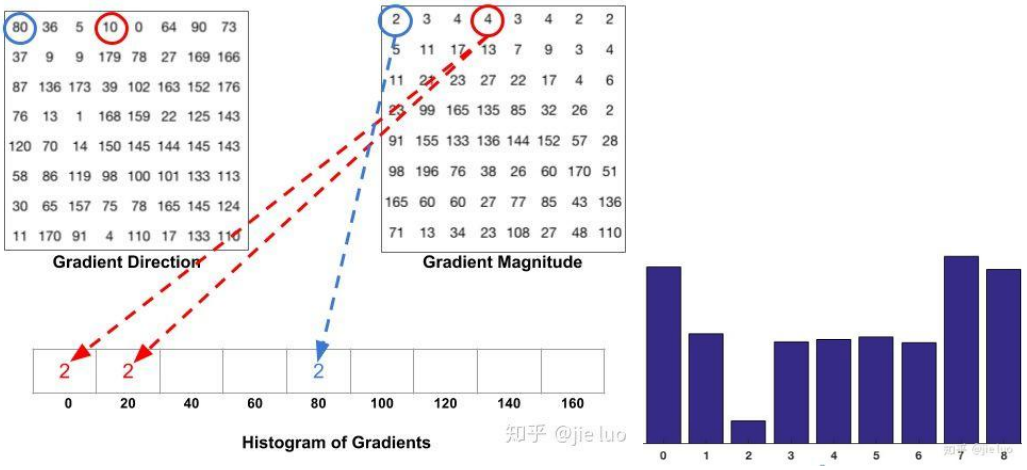
$$g = \sqrt{g_x^2 + g_y^2}$$

$$\theta = arctan\frac{g_x}{g_y}$$

梯度方向取绝对值，因此范围在 0-180

3. 计算梯度直方图

对每一个 8*8 的 cell，将 0-180 分成 9 份，然后统计这个方向区间里梯度强度值的和。



通过计算方向的值到区间的距离来判断应该落入哪个区间。如果遇上可以落入两个区间

的值，就对半分强度值。比如 10 这个方向距离 0 也是 10，距离 20 也是 10，就对半分。统计完成后，观察数据就可以看出这个 cell 是有较多的横边，竖边或角。

4. 对 16*16 大小的 block 的归一化

在上一步，每一个 8*8 的 cell 得到一个直方图 9 维 vector。现在我们对 16*16 的 block 进行归一化，就是将 4 个 cell 的 9 维 vector 叠到一起，成为一个 36 维的 vector，再进行归一化。

对于一个 64*128 大小的图像，每一排是 8 个 cell，每一列是 16 个 cell。在归一化时，block 是 cell by cell 移动，不是 block by block，所以是横 7 个，竖 15 个，得到 7*15 共 105 个 block。每一个 block 得到一个 36 维的 vector，那么 105 个 block 就是 36*105=3780 维的 vector。

5. SVM 分类

至此我们已经得到了图片的 HOG 特征向量，接下来就是用 SVM 来分类了。

Harris：是一种角点检测算法。是一种特征检测算法。角点是在两个方向上灰度变化剧烈的特征点。

我们可以通过计算两个特征值，这两个特征值分别代表两个方向上的变化程度，来判断角点。两个特征值都大，说明是角点，一个大，一个小，说明是边，两个都小，说明是均匀区域。特征值由计算如下矩阵 M 的特征值得来：

$$E(u,v) = \Sigma_x \Sigma_y w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

$$I(x+u,y+v) \approx I(x,y) + I_x(x,y)u + I_y(x,y)v$$

$$E(u,v) \approx \Sigma_{x,y} w(x,y)[I_x(x,y)u + I_y(x,y)v]^2 = [u,v]\, M(x,y) \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M(x,y)=\Sigma_w\begin{bmatrix}I_x^2 & I_xI_y\\ I_xI_y & I_y^2\end{bmatrix}=\begin{bmatrix}A & C\\ C & B\end{bmatrix}A=\Sigma_wI_x^2,B=\Sigma_wI_y^2,C=\Sigma_wI_xI_y$$

判断角点时，无需具体计算矩阵 M 的特征值，使用下式近似计算角点响应值：

$$R=detM-\alpha(traceM)^2$$

$$detM=\lambda_1\lambda_2=AB-C^2\quad traceM=\lambda_1+\lambda_2=A+B$$

式中，detM 为矩阵 M 的行列式，traceM 为矩阵 M 的迹，α为一常数，通常取值为 0.04~0.06。增大α的值，将减小角点响应值 R，减少被检测角点的数量；减小α的值，将增大角点响应值 R，增加被检测角点的数量。

Harris 算法步骤：

1. 计算像素点的梯度值

计算图像$I(x,y)$在X方向和Y方向的梯度

$$I_x=\frac{\partial I}{\partial x}=I(x,y)\otimes\begin{pmatrix}-1 & 0 & 1\end{pmatrix}$$

$$I_y=\frac{\partial I}{\partial y}=I(x,y)\otimes\begin{pmatrix}-1 & 0 & 1\end{pmatrix}^T$$

2. 计算图像两个方向梯度的乘积I_x^2、I_y^2、I_xI_y

3. 使用窗口高斯函数分别对I_x^2、I_y^2、I_xI_y 进行高斯加权，生成矩阵 M

4. 计算每个像素的 Harris 响应值 R，并设定阈值 T，对小于阈值的 R 置零

5. 在一个固定窗口大小的领域内（5*5）进行非极大值抑制，局部极大值点即为图像中的角点。

将 Harris 和高斯尺度空间表示相结合，可以使 Harris 具备尺度不变性。通过定义一个尺度自适应的二阶矩：

$$M=\mu(x,y,\sigma_I,\sigma_D)=\sigma_D^2g(\sigma_I)\otimes\begin{bmatrix}L_x^2(x,y,\sigma_D) & L_xL_y(x,y,\sigma_D)\\ L_xL_y(x,y,\sigma_D) & L_y^2(x,y,\sigma_D)\end{bmatrix}$$

式中，g(σI)表示尺度为σI 的高斯卷积核，Lx(x,y,σD)和 Ly(x,y,σD)表示对图像使用高斯函数 g(σD)进行平滑后取微分的结果。σI 通常称为积分尺度，是决定 Harris 角点当前尺度的变量，σD 为微分尺度，是决定角点附近微分值变化的变量，通常σI 应大于σD。

Harris 尺度算法步骤：

1. 确定尺度空间的一组取值σI=(σ0,σ1,σ2,...,σn)=(σ,kσ,k2σ,...,knσ),σD=sσI

2. 对于给定的尺度空间值σD，进行角点响应值的计算和判断，并做非极大值抑制处理

3. 在位置空间搜索候选角点后，还需在尺度空间上进行搜索，计算候选点的拉普拉斯响应值，并于给定阈值作比较

$$F(x, y, \sigma_n) = \sigma_n^2 |L_{xx}(x, y, \sigma_n) + L_{yy}(x, y, \sigma_n)| \geq threshold$$

4. 将响应值 F 与邻近的两个尺度空间的拉普拉斯响应值进行比较，使其满足

$$F(x, y, \sigma_n) > F(x, y, \sigma_l), \quad l = n - 1, n + 1$$

这样既可确定在位置空间和尺度空间均满足条件的 Harris 角点。

FAST：是一种角点检测算法。比 Harris 速度快，更适合实时应用。是一种特征检测算法。

若某像素点与周围邻域足够多的像素点处于不同区域，则该像素可能为角点。考虑灰度图像，即若某像素点的灰度值比周围邻域足够多的像素点的灰度值大或小，则该点可能为角点。

FAST 算法步骤：

1. 对于图像中的每一个像素点 P，其灰度值为 Ip

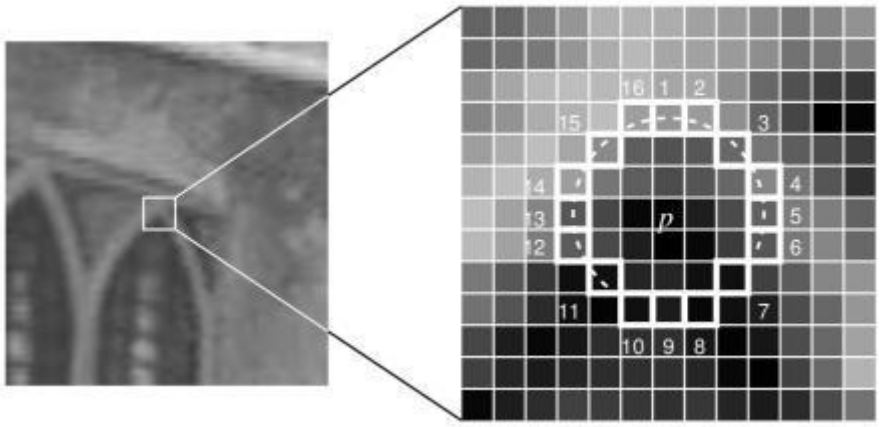
2. 以该像素点为中心考虑一个半径为 3 的离散化的 Bresenham 圆，圆边界上有 16 个

像素

3. 设定一个合适的阈值t，如果圆上有n个连续像素点的灰度值小于Ip-t或者大于Ip+t，那么这个点即可判断为角点（n 的值可取 12 或者 9）。

4. 对于图像中每一个点都算 16 个位置的像素，效率很低。可以使用一种快速排除大部分非角点像素的方法，检查周围 1、5、9、13 四个位置的像素，如果位置 1 和 9 与中心像素 P 点的灰度差小于给定阈值，则 P 点不可能是角点，直接排除；否则进一步判断位置 5 和 13 与中心像素的灰度差，如果四个像素中至少有 3 个像素与 P 点的灰度差超过阈值，则考察邻域圆上 16 个像素点与中心点的灰度差，如果有至少 9 个超过给定阈值则认为是角点。

5. 对于邻近位置存在多个特征点的情况，需要进一步做非极大值抑制。给每个角点一个量化的值 V，假设 P，Q 两个点相邻，分别计算两个点与其周围的 16 个像素点之间的差分和为 V，去除 V 值较小的点，即把非最大的角点抑制掉。



SIFT：目标检测，特征提取。最重要的是解决了尺度不变性的要求。我们对 DOG 序列进行尺度空间主轮廓的提取，并以该主轮廓作为一种特征向量，实现边缘、角点检测和不同分辨率上的特征提取等。是一种特征检测和描述算法。在介绍 SIFT 前，首先解释一下三个概念。

尺度空间表达：人眼视物，近处的物体大而且清晰，远处的物体小而且模糊。我们用高斯核来不同程度的模糊图像，来模拟物体越来越远的状态下的模糊程度。尺度空间，就是不同模糊程度的图像组成的空间。

$$G(x_i, y_i, \sigma) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x-x_i)^2+(y-y_i)^2}{2\sigma^2}\right)$$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

金字塔多分辨率表达：使用低通滤波器平滑图像，对平滑图像进行降采样（通常是水平，竖直方向 1/2），从而得到一系列尺寸缩小的图像。

DoG：前面两者的结合。

SIFT 算法步骤：

1. 构造高斯金字塔。为获取更多的稳定的特征点数目，对原始图像做放大一倍操作，得到尺度参数 σ0=1 （作者这里假定原始图像已有σ=0.5 的高斯模糊）。所以为了得到第一组第一层图片的σ1=1.6，运用高斯模糊的半群性质。要对放大后的图像作 $\sqrt{1.6^2-(0.5\times 2)^2}$ 的高斯平滑。

2. 得到第一组第一层图片的尺度后，运用公式 $\sigma_n = k^{n-1}\cdot\sigma_1$ ， σ1 指的是该组的第一层尺度，K 称为比例系数，作者给出 $k = 2^{\frac{1}{s}}$ 。通过高斯模糊分别计算其它层的结果。S 为每组层数。

3. 我们将第一组倒数第三层的图片作比例因子为 2 的降采样，结果作为第二组的第一层（第二组第一层的尺度是第一组第一层尺度的俩倍）。同理，运用公式 $\sigma_n = k^{n-1}\cdot\sigma_1$ 通过高斯模糊分别计算其它层的结果。

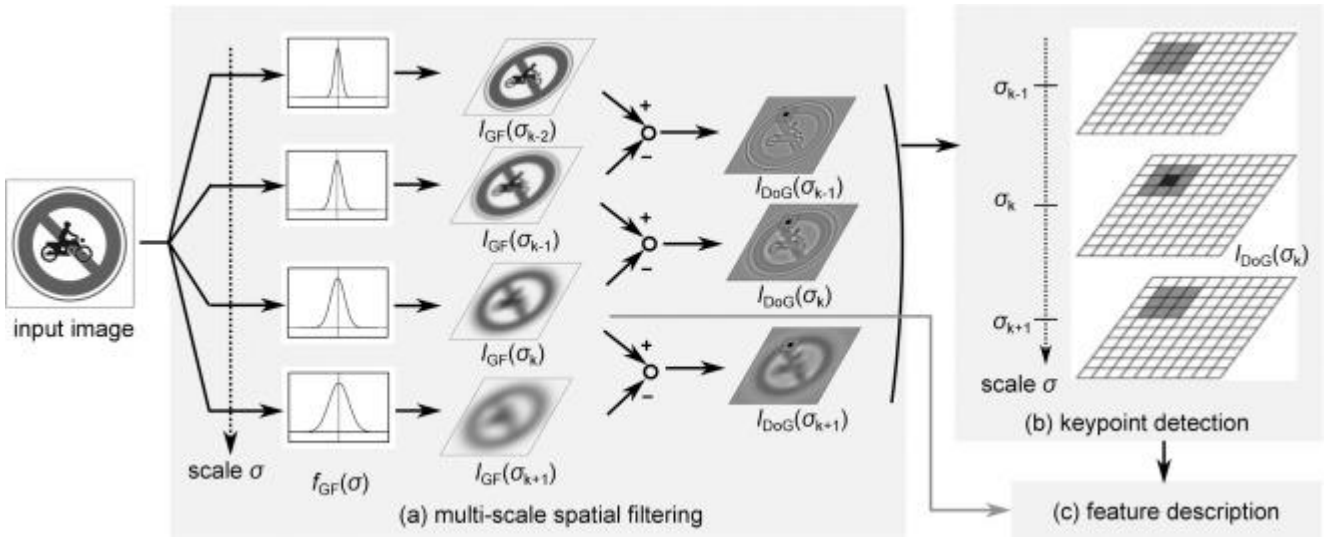
4. 将第二组第三层的图片做比例因子为 2 的降采样，结果作为第三层第一组的结果，同样要注意初始尺度的变化，其它处理跟第二组同。

5. 高斯金字塔的组数：$O = [\log_2(\min(M,N))] - 3$，M,N 分别是图像的行数和列数，t 为塔顶图像的最小维数的对数值，这里取 3。

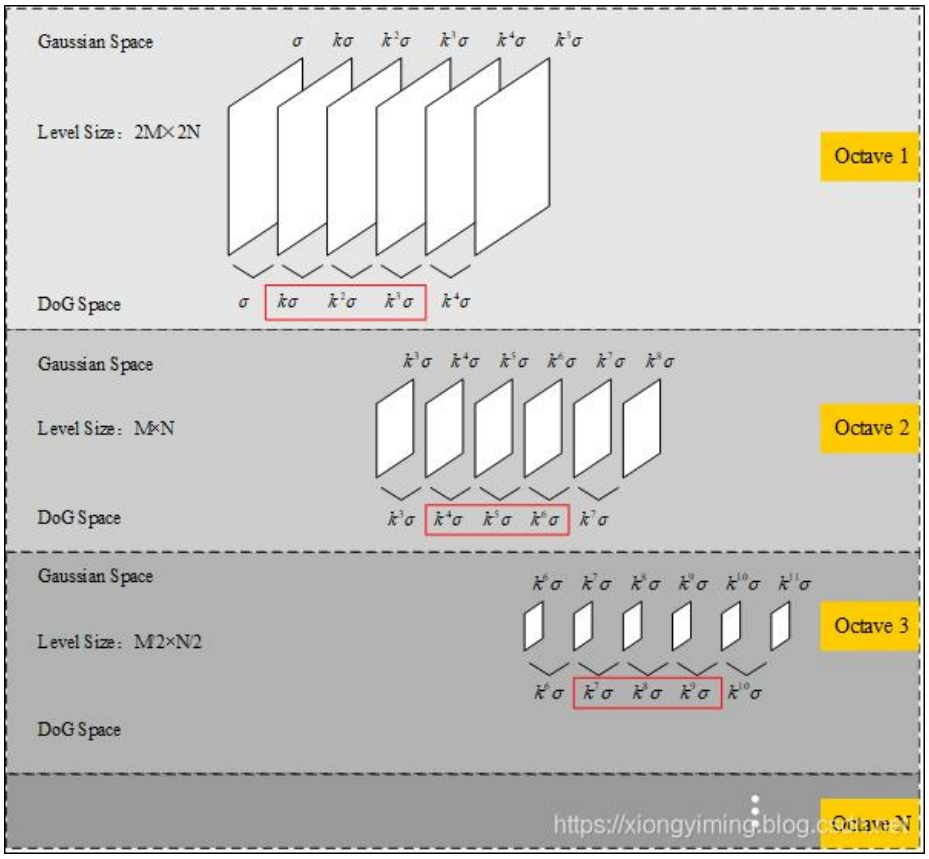
6. 最后可将组内和组间尺度归为：$2^{i-1}(\sigma, k\sigma, k^2\sigma, \cdots k^{n-1}\sigma)$。i 表示金字塔组数，n 表示每一组的层数。

7. 然后用金字塔相邻图像相减构建 DoG 金字塔。

8. 检测 DoG 的局部极值点。用每一个像素点和它所有的相邻点比较，看其是否比它的图像域和尺度域的相邻点大或者小。中间的检测点和它同尺度的 8 个相邻点和上下相邻尺度对应的 9×2 个点共 26 个点比较，以确保在尺度空间和二维图像空间都检测到极值点。一个点如果在 DOG 尺度空间本层以及上下两层的 26 个领域中是最大或最小值时，就认为该点是图像在该尺度下的一个特征点。



在极值比较的过程中，每一组图像的首末两层是无法进行极值比较的，为了满足尺度变化的连续性，我们在每一组图像的顶层继续用高斯模糊生成了 3 幅图像，高斯金字塔每组有 S+3 层图像。DoG 金字塔每组有 S+2 层图像。下图为不同尺度不同层间极值检测示意图。

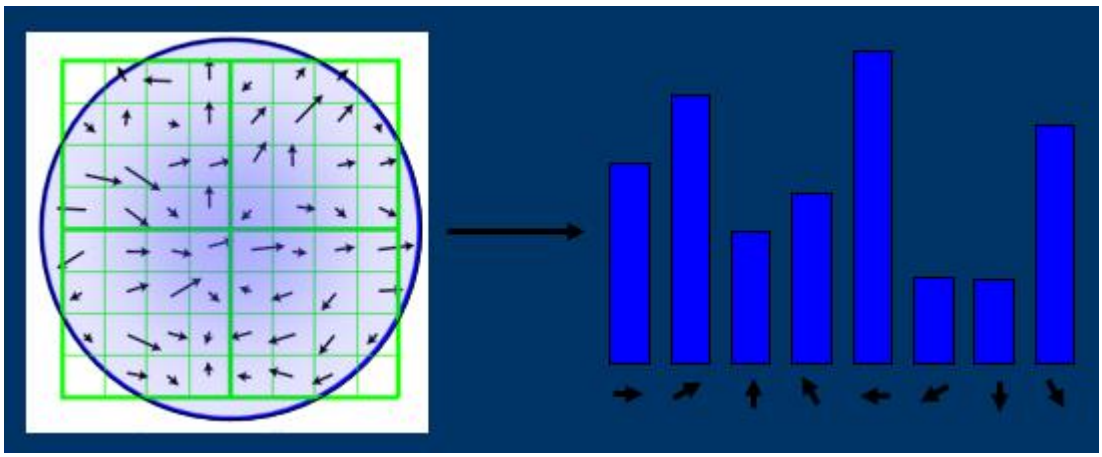


9. 因为得到的极值点往往不是真正的极值点，我们对尺度空间 DoG 函数进行曲线拟合，然后对极值点进行修正。对公式 $D(X) = D + \frac{\partial D}{\partial X}^T X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X$ 求导，得到 $\hat{X} = -\frac{\partial D}{\partial X}^T (\frac{\partial^2 D}{\partial X^2})^{-1}$，其中 X^为极值点偏移量。将修正后的值带入上式，计算出 $D\left(\hat{X}\right) = D + \frac{1}{2}\frac{\partial D}{\partial X}^T X$ 结果小于 0.04 的极值点均抛弃。

10. 去除边缘响应。观察 Hessian 矩阵 $H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$ 的特征值比率。令α为较大特征值，β为较小的特征值。欠佳的峰值点的特点是α和β之间差距太大，大于 10 倍。 $\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r+1)^2}{r}$ 令α=rβ。如果 $\frac{Tr(H)^2}{Det(H)}$ 的比值大于

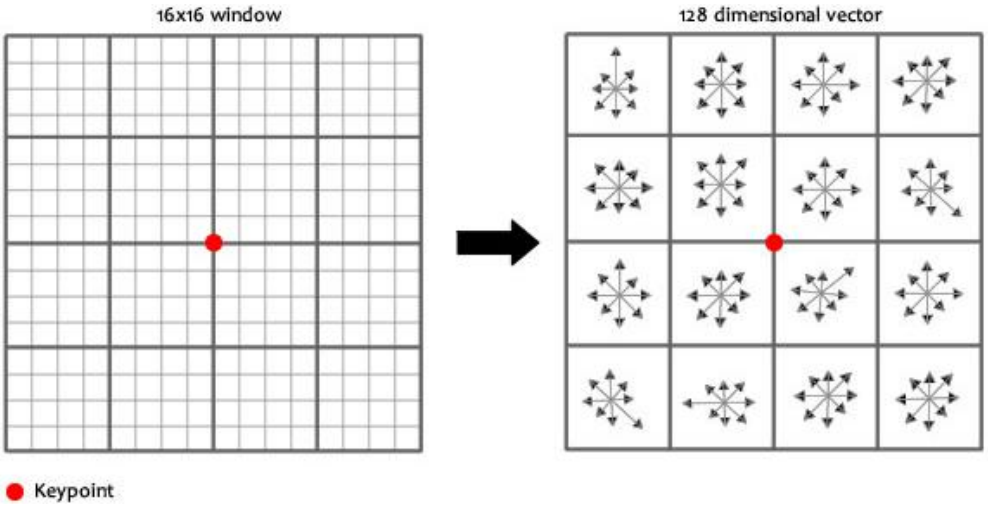
$\frac{(r+1)^2}{r}$ （一般取 r=10），就丢弃这个极值点。

11. 接下来是方向分配。和 HOG 算法一致，首先计算每个像素的梯度幅值和方向。然后对每个极值点，在其 3*1.5*σ半径的圆圈内，统计方向和幅值，以 10 度为一柱。最高的做为主方向，其他的达到最大值 80%的方向可作为辅助方向。



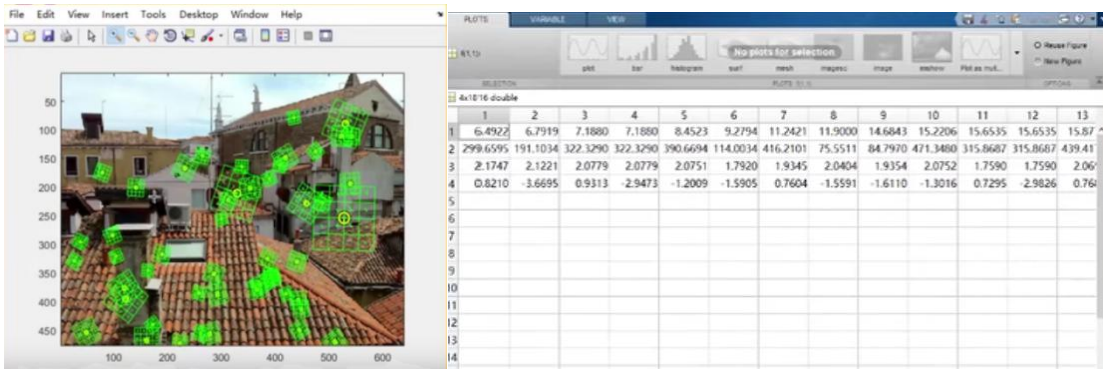
12. 至此，图像的关键点已经检测完毕，每个关键点有三个信息：位置，所处尺度、方向，由此可以确定一个 SIFT 特征区域。

13. 关键点描述。取关键点周围 16*16 个像素点，将其分为 4*4 个格子。，对每个小块，计算 8 个方向上的梯度直方图。这样对每个关键点，形成一个 4*4*8=128 维的描述子，每一维都可以表示 4*4 个格子中一个的 scale/orientation. 将这个向量归一化之后，就进一步去除了光照的影响。

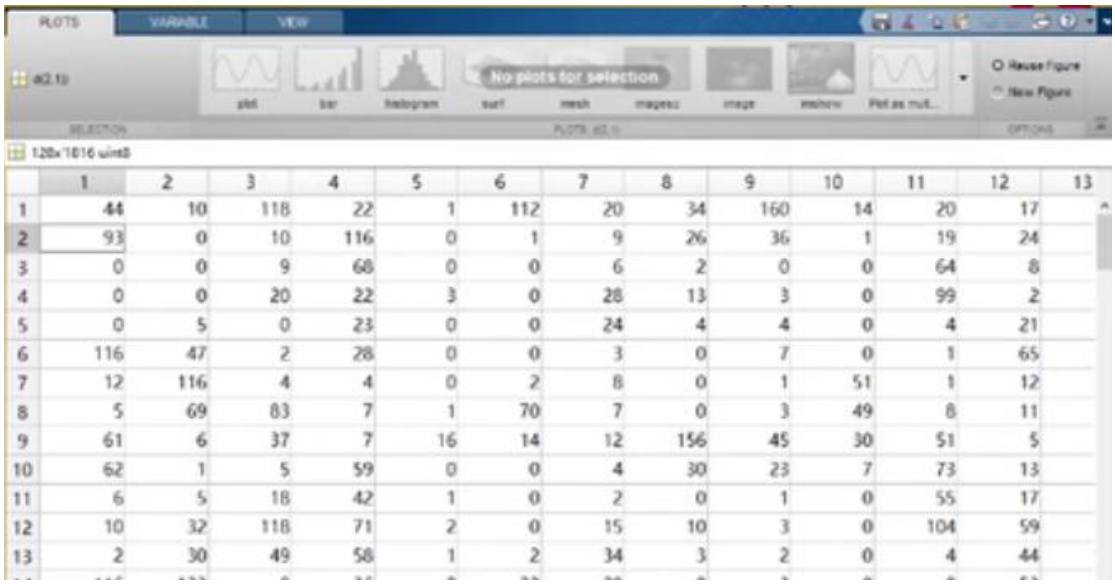


14. 通过计算两个特征点间的欧式距离来确定匹配度，欧氏距离越短，代表两个特征点的匹配度越好。

最后的结果和关键点如下图所示：



Excel 中的一列代表一个关键点，分别是 x, y, scale, rotation.可以看第一个图中框的大小，越大的框代表这个特征点是在尺度比较大的图中找到的。

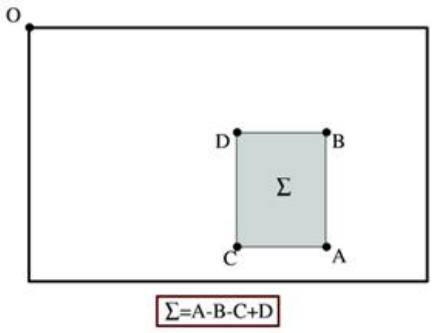


	1	2	3	4	5	6	7	8	9	10	11	12	13
1	44	10	118	22	1	112	20	34	160	14	20	17	
2	93	0	10	116	0	1	9	26	36	1	19	24	
3	0	0	9	68	0	0	6	2	0	0	64	8	
4	0	0	20	22	3	0	28	13	3	0	99	2	
5	0	5	0	23	0	0	24	4	4	0	4	21	
6	116	47	2	28	0	0	3	0	7	0	1	65	
7	12	116	4	4	0	2	8	0	1	51	1	12	
8	5	69	83	7	1	70	7	0	3	49	8	11	
9	61	6	37	7	16	14	12	156	45	30	51	5	
10	62	1	5	59	0	0	4	30	23	7	73	13	
11	6	5	18	42	1	0	2	0	1	0	55	17	
12	10	32	118	71	2	0	15	10	3	0	104	59	
13	2	30	49	58	1	2	34	3	2	0	4	44	

然后可以看每个关键点的 128 维描述。

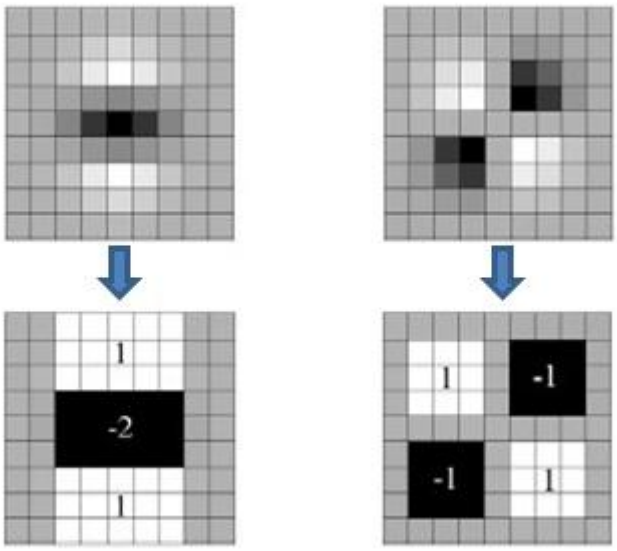
SURF：相当于 SIFT 的加速改进版本。借助于积分图。

积分图：积分图中任意一点(i,j)的值 ii(i,j)，为原图像左上角到任意点(i,j)相应对角线区域灰度值的总和。具体实现时 ii(x,y)可由前面的结果迭代计算得到。得到积分图后，计算任何矩形区域内的像素灰度和只需进行三次加减运算，如下图所示。



SURF 算法步骤：

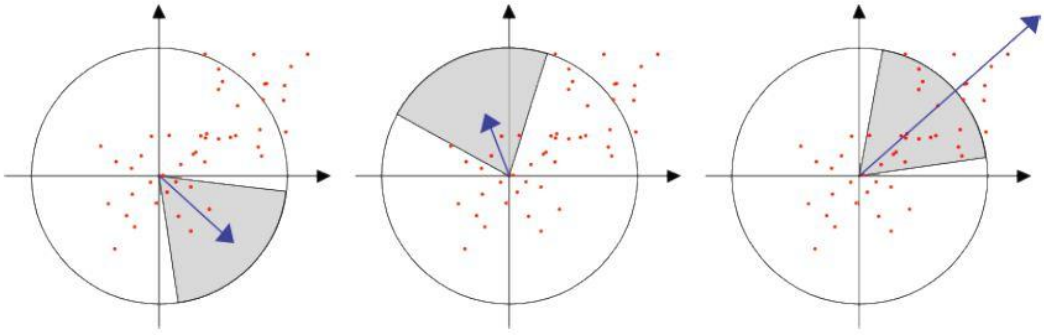
1. 使用盒式滤波器替代高斯滤波器来对图像做卷积操作。为什么盒式滤波器可以提高运算速度呢，这就涉及到积分图的使用。盒式滤波器对图像的滤波转化成计算图像上不同区域间像素和的加减运算问题，这正是积分图的强项，只需要简单几次查找积分图就可以完成，如下图所示。



2. 构造尺度空间。SIFT 用高斯卷积来构成尺度空间。而 SURF 中，不同组间图像的尺寸都是一致的，不同的是不同组间使用的盒式滤波器的模板尺寸逐渐增大，同一组间不同层间使用相同尺寸的滤波器，但是滤波器的模糊系数逐渐增大。比如第一组使用 3*3，第二组使用 5*5；第一组第一层使用（1，-2，1）第二层使用（2，-4，2）这样。

3. 特征点定位，这一步和 SIFT 一致，都是比较上下领域内的 26 个点，初步定位出关键点，再排除错误和比较弱的关键点。筛选出最终的稳定的特征点。

4. 特征点主方向匹配，SIFT 用的是梯度直方图。而在 Surf 中，采用的是统计特征点圆形邻域内的 harr 小波特征。即在特征点的圆形邻域内，统计 60 度扇形内所有点的水平、垂直 harr 小波特征总和，然后扇形以 0.2 弧度大小的间隔进行旋转并再次统计该区域内 harr 小波特征值之后，最后将值最大的那个扇形的方向作为该特征点的主方向。如图：

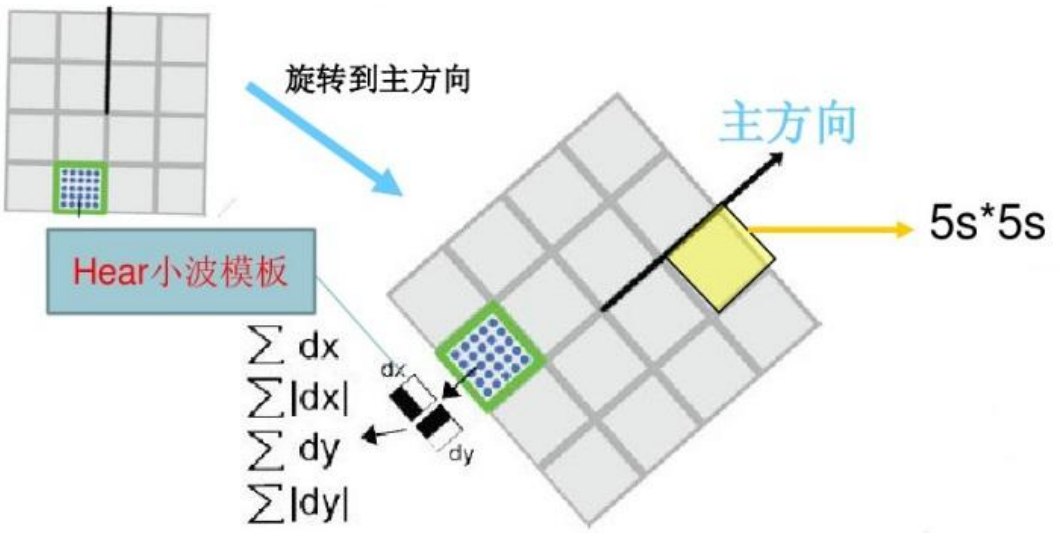


Harr 小波特征：例如我们有一个一维的图像[2,4,6,8,10,12,14,16]。一求均值：我们求相邻像素的均值[3,7,11,15]。这个新的图像分辨率就成了原来的一半(8/2=4)。二求差值。上面的均值我们存储了图像的整体信息。但是很多细节信息我们丢掉了，所以我们同时要记录图像的细节信息，这样在重构时能够恢复图像的全部信息。用 b[m]=(a[2m]−a[2m+1])/2。经过计算我们得到了结果[-1,-1,-1,-1]。这个新的分辨率也成了原来的一半(8/2=4)。然后将整体信息再次分解，直到得到一个值的解。

分辨率	整体信息	细节信息
4	3,7,11,15	-1,-1,-1,-1
2	5，13	-2，-2
1	9	-4

5. 生成特征点描述子。也是在特征点周围取一个 4*4 的矩形区域块，但是所取得矩形区域方向是沿着特征点的主方向。每个子区域统计 25 个像素的水平方向和垂直方向的

haar 小波特征，这里的水平和垂直方向都是相对主方向而言的。该 haar 小波特征为水平方向值之和、垂直方向值之和、水平方向绝对值之和以及垂直方向绝对值之和 4 个方向。把这 4 个值作为每个子块区域的特征向量，所以一共有 4*4*4=64 维向量作为 Surf 特征的描述子，比 Sift 特征的描述子减少了 2 倍。



6. 与 Sift 特征点匹配类似，Surf 也是通过计算两个特征点间的欧式距离来确定匹配度，欧氏距离越短，代表两个特征点的匹配度越好。不同的是 Surf 还加入了 Hessian 矩阵迹的判断，如果两个特征点的矩阵迹正负号相同，代表这两个特征具有相同方向上的对比度变化，如果不同，说明这两个特征点的对比度变化方向是相反的，即使欧氏距离为 0，也直接予以排除。

ORB:

ORB 算法步骤:

1. 用 FAST 得到特征点。

2. 用 rBRIEF 算法来描述特征点。以特征点为圆心，以 d 为半径做圆 o。

3. 在圆 o 内以某一模式选取 N 个点对。假设 N=4，实际应用中 N=512。4 个点为 P1(A, B), P2(A, B), P3(A, B), P4(A, B)。

4. 对这个 4 个点进行 T 操作，T 操作为：

$$T(P(A,B)) = \begin{cases} 1 & I_A > I_B \\ 0 & I_A \le I_B \end{cases}$$

其中 I_A 表示点 A 的灰度

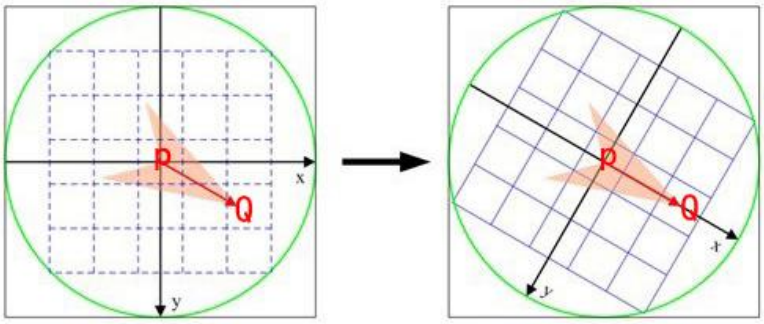
假如 T(P1(A, B)) = 1, T(P2(A, B)) = 0, T(P3(A, B)) = 1, T(P4(A, B)) = 1。那么特征描述向量为 1011。

5. 到现在已经完成了基础步骤，但是还不满足尺度不变性和旋转不变性

6. ORB 本身没有解决尺度不变性，OpenCV 里面的 ORB 用了图像金字塔来改善。

7. 为了解决旋转不变性，我们需要建立坐标系。

8. 在 ORB 中，坐标系的建立是以特征点 P 为圆心，取点区域的质心为另一个点，两点连线为 X 轴建立的 2 维坐标系。



如图，园内为取点区域，每个小格子代表一个像素，现在我们把这块圆心区域看做一块木板，木板上每个点的质量等于其对应的像素值。根据积分学的知识我们可以求出这个密度不均匀木板的质心 Q。质心的计算方法如下：

$$M_{00} = \sum_{X=-R}^{R} \sum_{Y=-R}^{R} I(x,y)$$

$$M_{10} = \sum_{X=-R}^{R} \sum_{Y=-R}^{R} xI(x,y)$$

$$M_{01} = \sum_{X=-R}^{R} \sum_{Y=-R}^{R} yI(x,y)$$

$$Q_X = \frac{M_{10}}{M_{00}}, Q_Y = \frac{M_{01}}{M_{00}}$$

9. 特征点的匹配，例如特征点 A，B 的描述向量为 10101011 和 10101010。我们设定一个阈值，比如 80%。当 A 和 B 的描述子的相似度大于 80%时，我们判断 A,B 是相同的特征点，即这 2 个点匹配成功。在这个例子中 A,B 只有最后一位不同，相似度为 87.5%，大于 80%。则 A 和 B 是匹配的。我们将 A 和 B 进行异或操作就可以轻松计算出 A 和 B 的相似度。而异或操作可以借组硬件完成，具有很高的效率，加快了匹配的速度。

RANSAC：找出最大 inliers 集

RANSAC 步骤：

1. 首先我们先随机假设一小组局内点为初始值。然后用此局内点拟合一个模型，此模型适应于假设的局内点，所有的未知参数都能从假设的局内点计算得出。

2. 用 1 中得到的模型去测试所有的其它数据，如果某个点适用于估计的模型，认为它也是局内点，将局内点扩充。

3. 如果有足够多的点被归类为假设的局内点，那么估计的模型就足够合理。

4. 然后，用所有假设的局内点去重新估计模型，因为此模型仅仅是在初始的假设的局内点估计的，后续有扩充后，需要更新。

5. 最后，通过估计局内点与模型的错误率来评估模型。

6. 整个这个过程为迭代一次，此过程被重复执行固定的次数，每次产生的模型有两个结局，一是因为局内点太少，还不如上一次的模型，而被舍弃。二是比现有的模型更好而被选用。

7. OpenCV 中滤除误匹配对采用 RANSAC 算法寻找一个最佳单应性矩阵 H，矩阵大小为 3×3。RANSAC 目的是找到最优的参数矩阵使得满足该矩阵的数据点个数最多，通常令 h33=1 来归一化矩阵。由于单应性矩阵有 8 个未知参数，至少需要 8 个线性方程

求解，对应到点位置信息上，一组点对可以列出两个方程，则至少包含 4 组匹配点对。

$$s\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

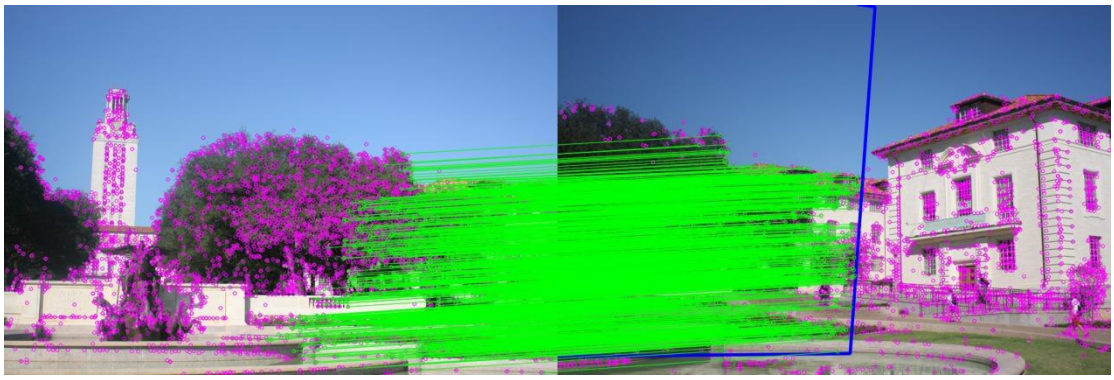
其中(x,y)表示目标图像角点位置，(x′,y′)为场景图像角点位置，s 为尺度参数。

8. RANSAC 算法从匹配数据集中随机抽出 4 个样本并保证这 4 个样本之间不共线，计算出单应性矩阵，然后利用这个模型测试所有数据，并计算满足这个模型数据点的个数与投影误差(即代价函数)，若此模型为最优模型，则对应的代价函数最小。

样例：SIFT 和 RANSAC

SIFT 检测关键点如下

RANSAC 匹配关键点如下



代码如下

```
import cv2 as cv
import numpy as np
src_img1 = cv.imread('C:/Users/Administrator/Desktop/uttower_left.jpg')
src_img2 = cv.imread('C:/Users/Administrator/Desktop/uttower_right.jpg')
sift = cv.SIFT()
kp1, des1 = sift.detectAndCompute(src_img1, None)
kp2, des2 = sift.detectAndCompute(src_img2, None)
img3 = cv.drawKeypoints(src_img1, kp1, src_img1, color=(255, 0, 255))
img4 = cv.drawKeypoints(src_img2, kp2, src_img2, color=(255, 0, 255))
hmerge = np.hstack((img3, img4))
cv.imshow("points", hmerge)
cv.waitKey(0)
cv.imwrite('C:/Users/Administrator/Desktop/key.jpg', hmerge)
FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks = 50)
flann = cv.FlannBasedMatcher(index_params, search_params)
matches = flann.knnMatch(des1,des2,k=2)
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)
if len(good)>10:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)

    M, mask = cv.findHomography(src_pts, dst_pts, cv.RANSAC, 5.0)
    matchesMask = mask.ravel().tolist()

    h,w,_ = src_img1.shape
    pts = np.float32([[0,0],[0,h-1],[w-1,h-1],[w-1,0]]).reshape(-1,1,2)
    dst = cv.perspectiveTransform(pts,M)

    src_img2 = cv.polylines(src_img2,[np.int32(dst)],True,255,3, cv.LINE_AA)
else:
    print("Not enough matches are found - %d/%d" % (len(good),MIN_MATCH_COUNT))
    matchesMask = None
draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)
img3 = cv.drawMatches(src_img1,kp1,src_img2,kp2,good,None,**draw_params)
cv.imshow('matches', img3)
cv.waitKey(0)
cv.imwrite('C:/Users/Administrator/Desktop/match.jpg', img3)
```