Blob detector

Jason Klebes

June 1, 2023

Package `blob-detector` uses a Laplacian of Gaussian filter to detect bright/dark features of a given diameter in 2D and 3D images. It is available on matlab file exchange and on github. Detailed instruction on calling the function and its parameters are in the README.

1 Motivation

Detection of bright circular disks/spheres, and returning their coordinates as a matlab list, is needed as the first step in nuclei tracking. The Matlab Image Processing Toolbox apparently lacks an equivalent functionality, as far as I can find.

2 Theory

see also

inf.ed.ac

wikipedia

Java implementation used in trackmate

python scikit-image implementation

The Laplacian of Gaussian filter can be thought of as first applying a Gaussian filter to smooth the image, then taking the Laplacian (spatial second derivative). With the correct choice of Gaussian width σ as $\sigma = r/\sqrt{n}$ in n dimensions ($\sigma = r\sqrt{2}$ in two dimensions), the series of operations transforms the centers of spheres of radius r into distinct intensity maxima/minima. We then identify regional maxima on the filtered image as the locations of blobs in the original image.

The image must be convolved first with the Gaussian kernel, then with the Laplacian operator. Equivalently, the two kernels are first combined to the LoG kernel

$$LG^{2D} = \frac{1}{\pi\sigma^4}\left(-1 + \frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

in two dimensions,

$$LG^{3D} = \frac{1}{\sqrt{2\pi}^3\sigma^5}\left(-3 + \frac{x^2+y^2+z^2}{\sigma^2}\right)e^{-\frac{x^2+y^2+z^2}{2\sigma^2}} \tag{2}$$

in three dimensions, and generally

$$LG = \frac{1}{\sqrt{2\pi}^n \sigma^{n+2}} \left(-n + \frac{\sum x_i x^i}{\sigma^2} \right) e^{-\frac{\sum x^i x_i}{2\sigma^2}} \tag{3}$$

in n dimensions, which is then convolved with the image in one step.

To implement the LoG filter in matlab, we construct the kernel as a small 2D or 3D matrix, deriving σ from the user-input estimated diameter/radius of the blobs and filling a matrix with numerical values of LG^{2D} or LG^{3D} at distance (x, y) or (x, y, z) from the center of the kernel, and convolve the kernel with the image using matlab builtin `conv2` or `convn`.

3 Detailed implementation

First a median filter is applied to the image; this can be disabled by passing name-value argument `MedianFilter=false`.

The LoG kernel is constructed as described above in function `LoG_kernel` or `Log_kernel_3D`. It's an array of dimensions `kernelSize` $\times$ `kernelSize` or `kernelSize` $\times$ `kernelSize` $\times$ `kernelSize` . Following the trackmate implementation, kernelsize is chosen as $3 + 2\max(2, \lceil 3\sigma + .05 \rceil)$, scaling roughly as 6σ. This kernel size includes crucial features of the kernel, the central minimum and the circular maximum; smaller kernels do not effectively apply a LoG filter and lead to worse results.

The image is convolved with the kernel using matlab's builtin `conv2` / `convn` function or another implementation, see below.

Next the coordinates of prominent maxima in the convolved image must be identified. Following scikit-image, I find regional maxima by applying a maximum filter (`imdilate`) and identifying pixels where the original image is equal to the dilated image.

Particles within one radius of the image edges are deleted, unless a different distance is set as `BorderWidth`. Set `BorderWidth=0` to disable.

Intensity of the central coordinate in the original image of the putative particle is used as a "quality" score. By default particles with 10% of the brightest particle or less are removed. In our cases with grainy images, a cutoff of 30% seems more appropriate. Set as `QualityFilter=0.3`.

Overlapping particle pairs are found and the lower-intensity particle is removed when `OverlapFilter=true` is given. Warning: $O(N^2)$ runtime in number of particles. Can be slow in large or 3D images if the quality filter is not used and large numbers of spurious particles are found.

The result is returned as a $3 \times N$ or $4 \times N$ array with columns (x, y, quality) or (x,y,z,quality).

4 Handling large images

The convolution is the most costly step for large and/or 3D images. The matlab default convolution algorithm seems to use something like "overlap-and-save" method, appropriate for cases where the kernel is much smaller than the image. An alternative convolution implementation by fourier transforming both image and kernel and multiplying pointwise in Fourier space (convolution theorem)

Method	Time
original conv2	.21s
GPUArray conv2	.33s
CUDAconvolution	.82s
imageJ trackmate	2.8s

Table 1: Time to run `blobdetect` on a 5201×2560 2D image, particle diameter 16.

Method	Time
original convn	20.9s
GPUArray convn	28.6s
CUDAconvolution3D	1.5s
imageJ trackmate	10.1s

Table 2: Time to run `blobdetect3D` on a $450 \times 450 \times 283$ 2D image, diameter 16 (image + kernel dimension < 512).

was found to be slower. GPU-parallelized Fourier-space convolution was found to be much faster.

I wrote a separate package, matlabCUDAconvolution, to call C++ CUDA convolutions from matlab. Install this package (place the files somewhere on matlabpath) and call blobdetect with option `GPU=true` to make use of the GPU-parallel convolution.

The CUDA GPU method is limited by fitting all arrays onto GPU memory simultaneously. It can be used straightforwardly on 2D images up to ?? elements and 3D images up to ?? elements on my GPU with 12GB GPU memory. For even larger 3D images, GPU memory is insufficient. It's simplest and fastest to handle these by applying `blobdetect/blobdetect3D` to tiled subimages separately and merge the resulting tables of coordinates.

Additionally, a Decimation In Frequency approach to the convolution was tried, following Karas and Svoboda 2011. While theoretical more accurate and general than tiled application of blobdetect, the decomposition and recomposition steps were too costly.

Method	Time
convn	2534s (42min)
tiled GPUArray convn	1969s (32min)
tiled CUDAconvolution3D	153s (3min)
DIF CUDAconvolution3D	650s (11min)
imageJ trackmate	674s (11min)

Table 3: Time to run `blobdetect3D` on a $5201 \times 2560 \times 283$ 3D image (bigger than GPU memory), `diameter=16`, kernel size $5? \times ? \times ?$.

5 Further

Write a wrapper unifying `blobdetect` and `blobdetect3D`, detecting whether the input is 2D or 3D.

Overlap detection could be sped up using k-dimensional tree or information from adjacent time points.

Scale-free LoG filter could be implemented to identify spheres of heterogeneous / unknown diameter and their diameters.