Intracellular Receptors

Materials and Methods: Analysis and Processing

Srdjan

February 6, 2021

1 General pipeline

A typical experiment involving imaging of pancreatic slices in our lab concerns a single field of view showing 10s–100s of cells, in a recording of at least several, often dozens, of gigabytes. Current tools (ImageJ, ...) rely on loading the recording, or its part, into memory, for viewing, analysis, and processing. It also requires laborious and long human engagement. We have developed a set of interdependent tools to automatize as much as possible the pipeline. The crucial elements of our pipeline are the following:

- (Semi-)automatic detection of regions of interest (ROIs);
- Transformation of ROI time traces into standard score ("z-score");
- Quantification of the phenotype for each ROI in terms of the distribution of events of different durations.

Our toolset is inspired and relies on CaImAn [1], a similar package developed for the purposes in neuroscience research.

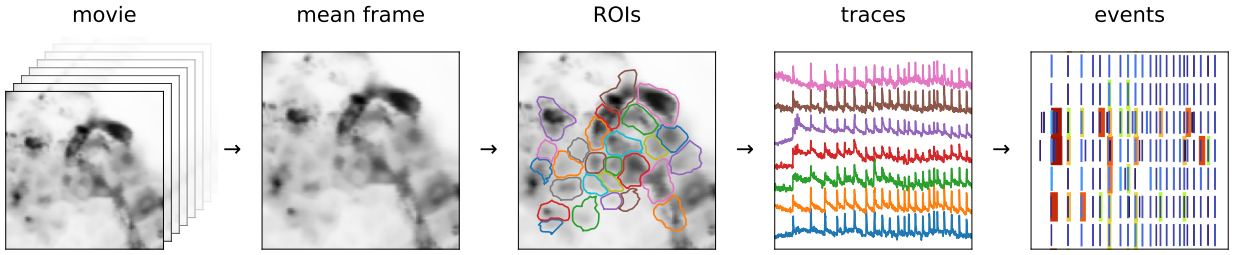

Figure 1: An illustration of our processing and analysis pipeline: (*i*) From a full movie, we calculate the mean (or other statistic) across all frames. (*ii*) We pass the mean image through a band-pass filter and define ROIs by detecting local peaks. (*iii*) We save ROIs with all the important information (time traces, ROI coordinates, movie statistics, recording frequency, pixel size, etc.). (*iv*) Traces contain features at very different time scales—with different timescales presumably important for different cell types. We collect them into separable events for analysis.

1.1 (Semi-)Automatic Detection of Regions of Interest

Once imported, a recording is stored as a 3-dimensional ($T \times x \times y$) numpy array [2]. When the recording is stable, obtaining a mean image, or any other statistic over frame, is rather trivial. In case there is horizontal movement, it can be corrected for by aligning the frames to a template. For this we use the functionality present in CaImAn [1], except that high frequency recordings

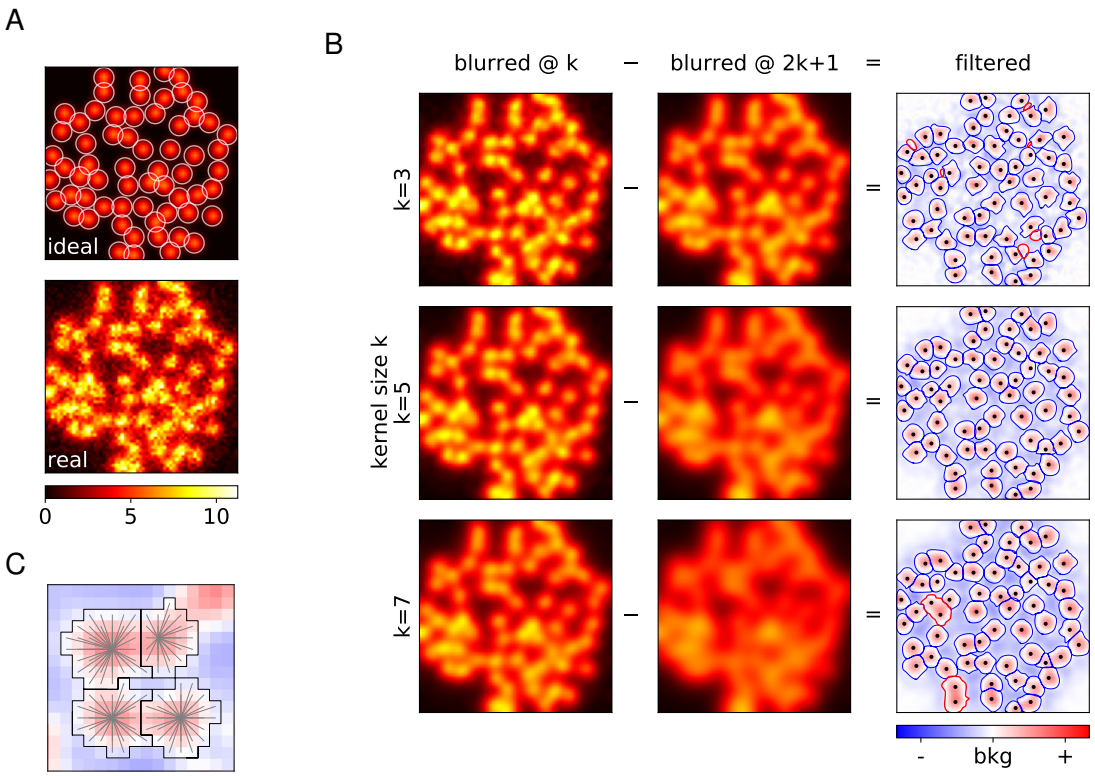

Figure 2: (A) Computationally created template 64×64 image with uniform spherical cells, and with Poisson noise added. (B) Band-pass filtering of the "realistic" image from A, with different kernel sizes. The size of the kernel determines the approximate size of the ROIs obtained. In thick red contours we emphasize misidentified ROIs; the dots indicate the real locations of the cells. (C) Each ROI is constructed by explicitly searching for a closest peak in intensity. A pixel can only be part of a single ROI.

need to be rebinned to some moderate frequency (a few Hz), before correcting, in order to reduce the noise level. Once the translation offsets are obtained, we use them to correct the recording in original frequency.

To define regions of interest, we blur the representative image by a kernel of the size we expect cells to be, and at the scale double of that. The difference between these two images represents a band-pass filter of the original, where the local intensity variation are emphasized (fig.2). We then pass through all pixels where the value of the filtered image is positive (or larger than a small positive threshold), and for each pixel we search for a local peak in its vicinity. All the pixels that lead to the same local peak are then grouped into a single ROI.

Representative image can be a mean over all frames or any other statistic. In addition our code supports standard deviation, mean and standard deviation of the first derivative of the movie, and a "robust maximum" of the movie. As "robust maximum" we define a very high percentile of the set absolute values of a set, essentially a value close to its maximum, by default it is 10th largest. We avoid the maximum as a means to make analysis robust to outliers. This statistic is sensitive to cells which fire extremely rarely during a recording, so that the mean of those pixels is negligible. By default, we choose an average of the mean and high percentile as a representative image for band-pass filtering and ROI extraction.

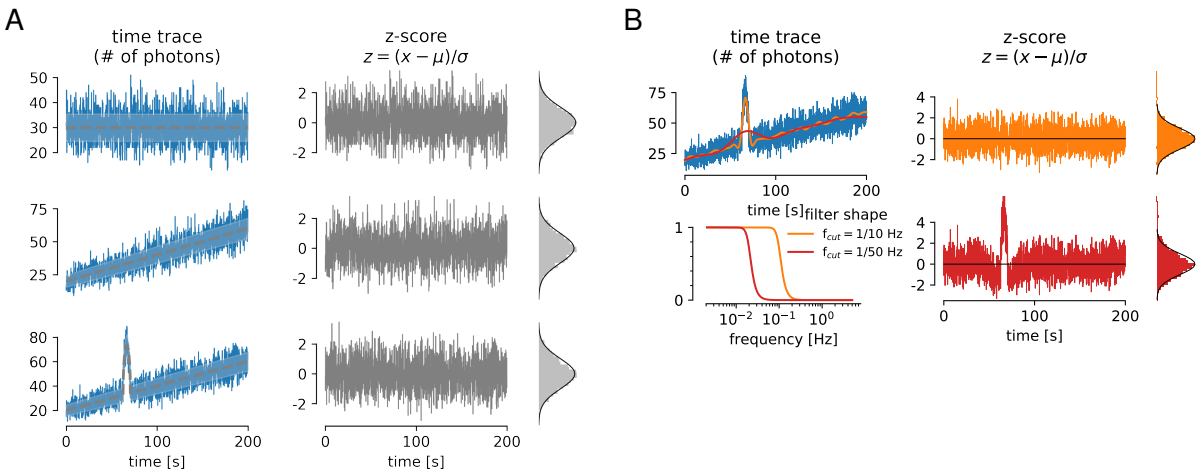

Figure 3: (A) *Left*: Simulated traces with Poisson noise (blue) around different means (grey) with different temporal features. The shaded area represents one standard deviation interval around mean. *Right*: Irrespective of the shape of the mean, z-score transformation behaves exactly as a Gaussian variable with zero mean and unit standard deviation.
(B) The values of sz-score depends on the reference. In subfigure (A), the reference curves are known, but in general they need to be inferred, typically by low-pass filtering. The cut-off frequency f_{cut} of a filter determines the timescale of the events that can be detected in z-score. If filtered at with too high cut-off (orange), the resulting smooth curve follows too closely the trace and the event is not visible in z. With $f_{cut} = 1/50$Hz, the resulting z-score has obvious outliers ($z \geq 3$), visible both in trace and in the histogram.

1.2 Trace processing

1.2.1 z-score and filtering

In an ideal detector, recording a time trace of a single pixel in absence of any signal would consist of independent values of the number of photons detected during the dwell time on that pixel. The values (x) are then distributed according to the Poisson distribution, with standard deviation (σ_1) being equal to the square root of the mean μ_1, $\sigma_1 = \sqrt{\mu_1}$.

A transformation to a new variable

$$z = \frac{x - \mu}{\sigma}$$

is called *standard transformation*, and the new variable standard score or z-score (fig.3). Essentially, it recasts the initial quantity x in the units of standard deviation from the expected mean. So, z spends 95% of the time between -2 and 2, and 99.7% between -3 and 3. Probability of $z > 3$ is very small $p < 0.0013$, which is why it is often considered a threshold value for pointing out the outliers.

If we were to rebin a trace by a factor of N, the relationship between standard deviation and mean remains, but for the summed quantities $\sigma_N = \mu_N = N\mu_1$. Normalization to the same mean as before $\tilde{\mu} = \mu_1$, results in the standard deviation $\sqrt{N}$ times lower $\tilde{\sigma} = \sigma_1/\sqrt{N}$.

Similarly, if we were to sum independent pixels ($a, b, c \ldots$), the resulting trace would have a mean which is a simple sum of the means of individual trace, and standard deviation $\sigma_{a+b+\ldots} = \sqrt{\mu_a + \mu_b + \ldots}$. We use this when estimating the traces for ROIs, which by definition consist of multiple pixels.

Correcting for the filtering distortion Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis

egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1.2.2 Realistic detectors

means vs variance plot for photon counting and for some nikon experiment

1.3 Identification of Events

2 Analyses

2.1 El Toro

2.2 Isradipine

2.3 Low Ca

2.4 etc

References

[1] Andrea Giovannucci, Johannes Friedrich, Pat Gunn, Jeremie Kalfon, Brandon L Brown, Sue Ann Koay, Jiannis Taxidis, Farzaneh Najafi, Jeffrey L Gauthier, Pengcheng Zhou, Baljit S Khakh, David W Tank, Dmitri B Chklovskii, and Eftychios A Pnevmatikakis. Caiman: An open source tool for scalable calcium imaging data analysis. *eLife*, 8:e38173, 2019.

[2] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.