HCR Imaging Python Module User Guide

Containing the Dot Detection 2.0 and Unmix 1.0 Packages

Mark E. Fornace,[1] Samuel J. Schulte,[1] and Niles A. Pierce[1,2]

This user guide describes the installation and use of the `hcr_imaging` Python module, which provides commands for the following two packages (Schulte *et al.*, 2024):

- *Dot Detection 2.0:* detection and colocalization of single-molecule dots in fluorescent HCR images.
- *Unmix 1.0:* linear unmixing of fluorescent multiplex HCR spectral images.

Technical support: `support@moleculartechnologies.org`

1 Dot Detection 2.0

HCR RNA-FISH enables digital RNA absolute quantitation (dHCR imaging mode) with single-molecule resolution in highly autofluorescent samples including whole-mount vertebrate embryos and thick brain slices (Shah *et al.*, 2016; Choi *et al.*, 2018; Schulte *et al.*, 2024). dHCR imaging mode is suitable for low-expression targets imaged at high magnification (e.g., 63× or 100×). Dot Detection 2.0 identifies diffraction-limited single-molecule dots in a fluorescent image obtained either using HCR bandpass imaging (Shah *et al.*, 2016; Choi *et al.*, 2018) or HCR spectral imaging (Schulte *et al.*, 2024). For a given fluorescent channel in a 2D or 3D image, the algorithm identifies a set of dot positions and weights. For a 2-channel redundant detection experiment in which target molecules are detected using two independent probes sets and amplifiers, Dot Detection 2.0 also performs dot colocalization between the two channels. As mRNA false-positive and false-negative rates for each channel go to zero, the colocalization fraction for each channel (fraction of dots in a given channel that are in both channels) will approach one from below (Shah *et al.*, 2016; Choi *et al.*, 2018).

Dot Detection 2.0 is an implementation of dot detection and colocalization via determinant-of-Hessian convolution and Gaussian mixture optimization (see Section S1.4.4 of (Schulte *et al.*, 2024)). This package differs from Dot Detection 1.0 (see Section S1.4.6 of (Choi *et al.*, 2018)) primarily in its programming in Python rather than Mathematica, its use of determinant-of-Hessian (DoH) versus Laplacian-of-Gaussian (LoG) convolution, and its sub-pixel refinement via Gaussian mixture optimization.

1.1 Input file

The package expects one required input file:

- a raw `xy` or `xyz` image file containing one or more channels. A variety of image formats are supported, including `.lif`, `.czi`, and `.tif`.

1.2 Example notebook

Use of Dot Detection 2.0 is illustrated in a Jupyter notebook that contains detailed comments:

- dot detection and colocalization

This notebook may also be viewed or downloaded on GitHub. Data files are available from CaltechDATA. Scripts may be viewed without downloading the data files. Note that due to the limitations of floating-point arithmetic, dot counts may vary slightly depending on computer hardware.

[1]Division of Biology & Biological Engineering, California Institute of Technology, Pasadena, CA 91125, USA. [2]Division of Engineering & Applied Science, California Institute of Technology, Pasadena, CA 91125, USA.

2 Unmix 1.0

HCR spectral imaging enables robust 10-plex, quantitative, high-resolution RNA and protein imaging in highly autofluorescent samples including whole-mount vertebrate embryos and brain sections. The following standardized ingredients are employed (Schulte *et al.*, 2024):

- 10 orthogonal HCR amplifiers (one per target),
- an optimized set of 10 fluorophores (one per amplifier),
- 10 optimized excitation wavelengths (one per fluorophore),
- 10 sets of optimized detection wavelengths (one to three detectors per fluorophore),
- an optimized excitation wavelength and set of four detection wavelengths for autofluorescence (measured by the user for the sample type of interest; treated as an 11th channel),
- spectral imaging hardware that provides the flexibility to use optimal excitation wavelengths for each fluorophore and for autofluorescence (Leica Stellaris 8 confocal microscope),
- linear unmixing software (Leica LAS X or Unmix 1.0) that returns 11 unmixed channels (one per fluorophore and one for autofluorescence).

For users that do not have access to a Leica Stellaris 8 microscope, these robust ingredients and workflow provide a starting point for adaptation to locally available hardware and Unmix 1.0 provides the flexibility to unmix spectral imaging data without use of Leica LAS X software.

Unmix 1.0 provides an efficient implementation of linear unmixing via non-negative least squares regression (see Section S1.4.5 of (Schulte *et al.*, 2024)).

2.1 Input files

The package expects two required input files:

- The sample image, containing the sample `xy` or `xyz` data imaged with each combination of excitation and detector settings. `Unmix 1.0` can handle a variety of common image formats including `.lif`, `.czi`, and `.tif`. Images may be two- or three-dimensional.
- The unmixing matrix, a small matrix with number of rows equal to the number of combinations of excitation and detector settings and number of columns equal to the number of output channels. If using the Leica LAS X software, this matrix can be loaded directly from the `.sdm` file output by that software. If not using the Leica LAS X software, see the example notebook `example-unmixing-matrix.ipynb` for an example of how to construct the unmixing matrix given a set of reference images.

2.2 Example notebooks

Use of Unmix 1.0 is illustrated in two Jupyter notebooks that contain detailed comments:

- Linear unmixing given a presupplied unmixing matrix for zebrafish image
- Linear unmixing matrix construction for zebrafish image

For convenience, two additional notebooks demonstrate identical usage using mouse brain images. These notebooks are equivalent to those above except for the choice of input image.

- Linear unmixing given a presupplied unmixing matrix for mouse brain image
- Linear unmixing matrix construction for mouse brain image

These notebooks may also be viewed or downloaded on GitHub. Data files are available from CaltechDATA. Scripts may be viewed without downloading the data files.

3 Software installation

`hcr_imaging` is distributed as a Python 3 module with a few high-performance routines written in C++. As such, the module may be installed from the provided binaries using `pip` in most cases. If desired, the module may also be installed from source, supposing that a C++ compiler is installed on the host system.

3.1 Installation requirements

We provide pre-built binaries for the `hcr_imaging` package for the following architectures:

- macOS arm64 (M-series processors)
- macOS x86_64 (older Intel processors)
- Linux x86_64
- Linux arm64
- Windows

These binaries are included in the distributed release zipfile here under subdirectory `/binaries`. Please contact our support to request that we provide binaries for a different architecture. Note also that the distributed Windows binaries use slightly slower versions of the necessary linear algebra routines, so in rare cases it may be worth installing from source and specifying your BLAS library directly.

3.2 Installing Python

This image analysis module requires Python 3.8 or greater. For newcomers to Python, an easy way to get a suitable Python 3 environment is by installing Anaconda. For Mac users, make sure to download the arm64-specific Anaconda module if you have an M-series processor. The module will *not* function if you install the provided binaries using an x86_64 installation of Python with an `arm64` Mac architecture.

3.3 Installing the `hcr_imaging` module

A number of Python packages are needed as strict dependencies, and they should be installed automatically (for an up-to-date list, consult the `pyproject.toml` file in the source repository). To install this module, in your terminal enter the following command:

```
python3 -m pip install -U hcr_imaging -f /path/to/binaries
```

(Replace `/path/to/package-binaries` with the path to your local directory containing the distributed `*.whl` files.) It is also recommended that you install `jupyter lab` or `jupyter notebook` to facilitate interactive usage.

If you have a version conflict in one of your pre-installed dependencies, it is recommended to use a fresh conda environment, which you can create via a command like `conda create -n my_environment_name python`. Please contact support if you run into repeated issues with dependency conflicts.

3.4 Source installation

Installation of binaries via `pip` is recommended unless you have an unsupported architecture or specific needs. However, if you have a working C++17 compiler (including `gcc`, `clang`, or Windows Visual Studio), you can also build this module from source via the following command:

```
python3 -m pip install -U -f /path/to/source-code
```

(Replace `/path/to/source-code` with the path to your local directory containing the distributed `setup.py` file.) When installing from source, you may specify the `HCR_IMAGING_BLAS` environmental variable to point to an optimized BLAS library which is installed on your system. This is generally not necessary for macOS, but for Linux and Windows you will obtain higher performance if you specify this library.

3.5 Running example notebooks

For newcomers to the Python ecosystems, we would recommend that you follow the guide here to launch JupyterLab to run Jupyter notebooks like the examples provided. To install JupyterLab, run the following command in your terminal:

```
conda install -c conda-forge jupyterlab # If using Anaconda
pip install jupyterlab # Otherwise
```

(See here for additional instructions for installing JupyterLab, if needed.)

Assuming JupyterLab is installed, launch it while providing the directory example notebook (`*.ipynb`) files:

```
jupyterlab /path/to/examples
```

A browser window should open. Each example notebook may be executed all at once by clicking the "Restart and run all cells" command from the menu. However, the input image file path may have to be changed depending on your downloaded location. Alternatively, each cell may be run in sequence by typing shift-enter from your keyboard for each highlighted cell.

References

Choi, H. M. T., Schwarzkopf, M., Fornace, M. E., Acharya, A., Artavanis, G., Stegmaier, J., Cunha, A., & Pierce, N. A. (2018). Third-generation in situ hybridization chain reaction: Multiplexed, quantitative, sensitive, versatile, robust. *Development*, **145**, dev165753.

Schulte, S. J., Fornace, M. E., Hall, J. K., Shin, G. J., & Pierce, N. A. (2024). HCR spectral imaging: 10-plex, quantitative, high-resolution RNA and protein imaging in highly autofluorescent samples. *Development*, **151**, dev202307.

Shah, S., Lubeck, E., Schwarzkopf, M., He, T.-F., Greenbaum, A., Sohn, C. H., Lignell, A., Choi, H. M. T., Gradinaru, V., Pierce, N. A., & Cai, L. (2016). Single-molecule RNA detection at depth via hybridization chain reaction and tissue hydrogel embedding and clearing. *Development*, **143**, 2862–2867.