CellOrganizer Tutorial for MMBIOS 2015

http://tinyurl.com/nzg2qst

Gregory R. Johnson gj@andrew.cmu.edu
Devin P. Sullivan devin.sullivan@scilifelab.se
April, 2015

Introduction

CellOrganizer is a software package that learns generative models of cell organization from fluorescence micrographs. These models are useful for modeling the dependency between compartments of the cell, allowing for a compact representation of cell geometries present in cell images and generating image of geometries useful for spatially realistic biochemical simulations. There are two main functions which this tutorial will cover: `img2slml`, the top-level function to train a generative model of cell morphology, and `slml2iml`, the top-level function to generate an instance from a trained model.

Who is this tutorial for?

This tutorial was written for people who have experience with fluorescence microscopy, no experience with CellOrganizer and possibly some experience with Matlab, generative models, or cell modeling. Users should be interested in learning how to use the automated modeling tools provided by CellOrganizer to explore their image data.

Something of a disclaimer

CellOrganizer is research code, and as such it is under constant development. Although we do our best to ensure our code is reliable, we distribute this code under the GNU public license without any type of warranty. For this reason, we hope not but, a feature may not work as expected. Please do not hesitate to contact us at cellorganizer@compbio.cmu.edu with any questions or issues you have.

Resources

Publications:
http://cellorganizer.org/Publications/

Software:
cellorganizer.org - Current CellOrganizer distribution

Other Software:
ImageJ - This is great software for viewing your images and those synthesized from CellOrganizer. This tutorial uses ImageJ in some spots.

Image Databases:

- http://www.openmicroscopy.org/
- www.cellimagelibrary.org
- http://www.proteinatlas.org/subcellular
- http://murphylab.web.cmu.edu/data/

Prerequisites

An OS X, Linux or Unix operating system

Matlab installation (Matlab 2013a or newer) with the following toolboxes: Image Processing Toolbox, Statistics Toolbox, Curve Fitting Toolbox and the SymBiology Toolbox.

Some basic familiarity with programming (preferably Matlab).

Some provided images OR images that satisfy the following requirements:

- Single cell images (These may be segmented from multi-cell fields)
- Nuclear shape (typically DAPI, Hoechst, or another DNA marker) channels
- Cell shape marker (cytoplasmic, plasma membrane, or ubiquitous markers work best)
- Images saved in a BioFormats compatible format.

Other useful image features:

- Cell shape segmentations
- Nuclear shape segmentations
- vesicular or cytoskeletal protein channels of interest

Setup

Step 0: Download CellOrganizer from cellorganizer.org and unzip it into a working directory

Alternatively, If you are at the MMBIOS 2015 Tutorial Room, you can open a terminal and type:

```
bash cellorganizer
```

Step 1: Add the CellOrganizer directory to the Path.

This can be done by navigating to the top level “cellorganizer” directory in the Current Folder panel within MATLAB and typing `setup` in the MATLAB command line.

Step 2: Get your images on the same computer.

If you don't have your own images, you can download some samples here: http://goo.gl/P2ryQ2. These are 3D HeLa images with a nuclear stain (channel 0), cell stain (channel 1) and protein stain (channel 2). The tagged protein is lamp2, a lysosomal protein. Unzip the .tgz file to some directory of your choice. Optionally, to decrease training time, set aside 10 to 15 images that will not be used.

Demos

Provided with the CellOrganizer software bundle are many demos that illustrate how the software’s functionality. Use these demos to familiarize yourself with further aspects of this program. Type `demoinfo` in the Command Window to list the provided demos and what their objective. These demos are made to be mainly autonomous of user input.

Training

`img2slml.m` is the main function used for training. It takes 5 inputs: a flag describing the dimensionality of the data (i.e. 2D or 3D, but this tutorial describes only 3D functionality), images for the nuclear channel, images for the cell shape channel, images for the protein channel (optional) and parameters used to change various model settings. The training portion of this tutorial covers the very basic setup required to get `img2slml` up and running.

Step 0: Create a "scratch" script.

This will keep track of what you have done so far and provide a resource for later use of CellOrganizer. Create a file named `mmbios_tutorial_train.m` somewhere outside of the CellOrganizer directory. Instead of inputting the following commands into the Command Window, type (or copy and paste) them into your scratch script.

Step 1: Create variables containing your images.

There are three ways to do this depending on how you have stored your images, each of which has its own strengths: string wildcards, a cell array of file paths, and a cell array of function handles.

To make life easier in the future, start by defining a variable that contains the path to the images' directory. Something like this:

```
img_dir = '/Users/greg/Desktop/LAM';
```

However, the directory path will depend on where the images were unzipped.

Option 1: String wild-cards:

If your files are named in some basic pattern, wildcards are the easiest way to get your file information into CellOrganizer. If you downloaded the images in Step 2 of Setup, your wildcard setup would be something like this:

```
nuc_img_paths = [img_dir '/cell*_ch0_t1.tif'];
cell_img_paths = [img_dir '/cell*_ch1_t1.tif'];
prot_img_paths = [img_dir '/cell*_ch2_t1.tif'];
```

Option 2: Cell-array of string paths:

Alternatively, you can store the images as individual paths in a cell array. Since there are 50 images, we will loop through the directory and store each name in an element of a cell array. There are more "programmatically correct" ways to do this, but this is the most direct. For the sake of training time, we'll iterate over only the first 15 images.

```
for i = 1:15
      nuc_img_paths{i} = [img_dir '/cell' num2str(i) '_ch0_t1.tif'];
      cell_img_paths{i} = [img_dir '/cell' num2str(i) '_ch1_t1.tif'];
      prot_img_paths{i} = [img_dir '/cell' num2str(i) '_ch2_t1.tif'];
end
```

Option 3: Function handles:
If you're very comfortable with Matlab, you can pass a cell-array of anonymous function handles as your images into CellOrganizer. If the previous sentence doesn't make any sense to you, it's probably best that you skip this part of the tutorial. An example of using function handles would be:

```
for i = 1:15
      nuc_img_paths{i} = @() ml_readimage([img_dir '/cell' num2str(i)...
            '_ch0_t1.tif']);
      cell_img_paths{i} = @() ml_readimage([img_dir '/cell' num2str(i)...
            '_ch1_t1.tif']);
      prot_img_paths{i} = @() ml_readimage([img_dir '/cell' num2str(i)...
            '_ch2_t1.tif']);
end
```

Here we're using the CellOrganizer provided function `ml_readimage` to read in and return the actual image matrix, but any function that returns the actual image matrix of data will work.

Step 2: Setup the parameter structure.
The parameter structure tells CellOrganizer how you want to build a model, and allows for option input. Most of the options have default values so we don't have to set them manually for this tutorial. However, we do need to know the pixel resolution of the images and a filename to save. To define the appropriate parameters, we create a struct variable and set fields accordingly:

```
%this is the filename to be saved
train_param.model.filename = 'model.mat';

%this is the pixel resolution in um of the images
train_param.model.resolution = [0.049, 0.049, 0.2000];

%this tells CellOrganizer what images we are inputting
train_param.train.flag = 'all';
```

The parameter `train_param.train.filename` defines where the .mat file containing the resulting model will be saved. By setting `train_param.train.flag` to `'all'` we specify training a model that trains a nuclear shape, cell shape and protein distribution model. We can also specify the train flag as `'framework'` to train just a nuclear shape and cell shape model (and we would therefore no longer need to provide protein images), or set the flag to `'nuc'` and just train a nuclear shape model (and not have to provide cell shape images).

So far we have the *bare minimum* requirements for setting up a model. We will set one more parameter to speed up the tutorial.

```
train_param.model.downsampling = [6,6,1.5];
```

This downsamples our input images by 4 in the X and Y dimensions, decreasing the memory used for the tutorial.

Step 2.5: Add a model type so CellOrganizer knows what kind of model to train.

```
train_param.nucleus.type = 'cylindrical_surface';
train_param.cell.type = 'ratio';
train_param.protein.type = 'vesicle';
train_param.debug = true;
```

Now that we have everything together, we can finally train the model:

```
img2slml('3D', nuc_img_paths, cell_img_paths, prot_img_paths, train_param);
```

If you don't have any protein images and/or cell images, you can just use empty brackets as input:

```
img2slml('3D', nuc_img_paths, cell_img_paths, [], train_param);
```

(make sure to adjust your `train_param.train.flag` to reflect the inputs to `img2slml)`

Step 3: Run your script
Press the play button on the top of the Matlab window or type the name of your script into the Command Window. If you used a lot of images or did not aggressively downsample your images this may take some time. In the meantime go get a coffee or lunch.

Step 4: Your model
Now that your run of CellOrganizer has completed without error, you should have a .mat file named model.mat in the directory in which you ran the code. Congratulations, you made it! If you load that file into your Workspace you'll see that this is another struct with fields. This is the model of your cell images. You'll notice that it's a lot smaller in file size than the collection of source images you used to train it. Take some time to explore these fields.

Synthesis

Here we will go over how to synthesize a synthetic cell shape in CellOrganizer. The main function here is `slml2img.m`. This function takes two inputs: a cell array of paths to the models from which we want to synthesize an image and a list of parameters.

Step 0: Create a “scratch” script.
Here we create a new script and call it `mmbios_tutorial_synthesis.m`.

Step 1: Setup the model and parameter inputs
Start by defining two variables: a cell-array containing the path to the model you created in the **Training** section, and a different parameter training structure. If you followed the instructions in the **Training** section, then you should be able to create a variable using the same path as in **Step 2** of **Training**.

```
model_path = {'model.mat'};
```

Alternatively you can generate images from the models provided by the CellOrganizer distribution:

```
model_path = {'./cellorganizer-v2.2/models/3D/lamp2.mat'};
```

Use the model of the Lamp2 pattern in HeLa cells.

The parameter structure is set up similarly to that in **Training**. We will define where we want the images to be saved, a prefix for the saved files and the number of images desired

```
%save into the current directory
synth_param.targetDirectory = './';

synth_param.prefix = 'mmbios_synthesis_tutorial';

%generate two images
synth_param.numberOfSynthesizedImages = 2;
```

Now that we have everything set up, we can generate an image or two!

Step 2: Synthesize
As a last input, we must call `slml2img.m`:

```
slml2img(model_path, synth_param);
```

Save your file and run it. This may take a little bit, especially if you have decided to generate many images.

Step 3: Check out your images
Now that the image generation is completed, you can check them out. In the current directory you should see a folder named “mmbios_synthesis_tutorial”, and in that should be two directories, “cell1” and “cell2”, each of which contain images corresponding to the nuclear

shape, cell shape and protein images drawn from the model you trained in the **Training** section. While these images can be opened in ImageJ, we are going to demonstrate two useful tools in CellOrganizer that we frequently use to explore our synthesized images.

First we're going to create an *indexed* by combining output images.

```
%read in each image to a variable
im_cell = ml_readimage('<path to cell image>');
im_dna = ml_readimage('<path to nucleus image>');
im_prot = ml_readimage('<path to protein image>');

%create an empty image
im_indexed = zeros(size(im_cell));

%Set the cell shape, nuclear shape and protein shape values to 1,2,3
respectively
im_indexed(im_cell>0) = 1;
im_indexed(im_dna>0) = 2;
im_imdexed(im_prot>0) = 3;
```

Now that we have an image, we can view it with the function img2vol:

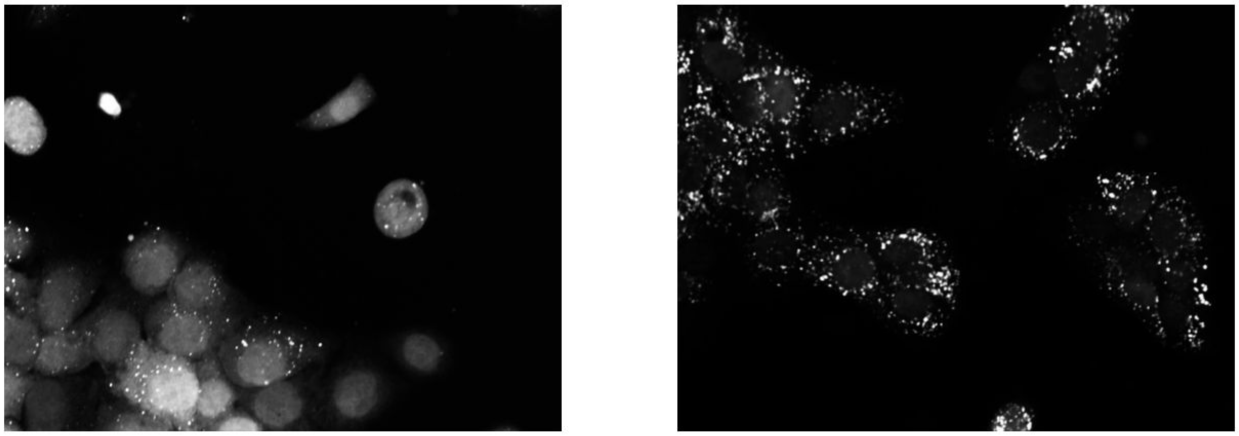
```
figure, img2vol(ml_downsize(im_indexed, [2,2,2]));
```

Congratulations you have created a synthetic cell geometry!

Visualizing Model Results

Although generating synthetic cell shapes is fun and all, the real power of CellOrganizer lies in it's ability to describe distributions of cell geometries and the organization of components within them. Here we will demonstrate how to plot some interesting results.

Background:
Upon exposure to Bafilomycin A1, microtubule associated protein light chain 3 (LC3) localizes into autophagosomes for degradation and forms punctate structures.


Images of eGFP-LC3 tagged RT112 cells at 40x under normal conditions (left) and in a 50uM Bafilomycin condition (right).

Let's say we are curious as to how the number of autophagosomes changes with Bafilomycin concentration. Given a collection of images under different concentrations we can segment out the cell shapes and train a model for the cells contained in each image. It just so happens that we have already done that, and the models and associated drug concentrations, can be found here: goo.gl/oitZaa

That .mat file has two variables saved in it. One is a list of drug micromolar concentrations, and the other is a list of models trained with images of cells at those concentrations (like the above two images). For each model, we're going to plot the number of autophagosomes versus the Bafilomycin concentration.

Step 0: Create another "scratch" script
Lets call this one `plotObjsByModel.m`

Step 1: Load the model data into the workspace.
Load the .mat file you downloaded into the Workspace by double clicking on it. You should see two variables, `conc`, and `models`. These are the variables that contain the drug concentrations and trained CellOrganizer models of cells exposed to Bafilomycin at those concentrations. You can access the first model by typing `models{1}`, the second model by `models{2}` and so on.

You will see that there are a lot of components to these models, but we're just interested in the number of objects under each conditions.

Step 2: Plot the average number of autophagosomes for each model
We must access the component of the model that contains the distribution for the number of objects. We can access that in the first model with:

```
models{1}.proteinModel.objectModel.numStatModel
```

The output should be:

```
ans =

     name: 'gamma'
    alpha: 2.7464
     beta: 19.291
```

This means that the distribution over the number of objects contained in the cells that were used to train this model is modeled as a gamma distribution with two parameters, alpha, and beta. It just so happens that the average of a gamma distribution is the product of these two parameters. Let's write a loop to get the average number of autophagosomes from each model:

```
for i = 1:length(models)
      numObjsModel = models{i}.proteinModel.objectModel.numStatModel;
      avgObjects(i) = numObjsModel.alpha*numObjsModel.beta;
end
```

Now we simply plot the number of objects versus drug concentration. Here we will use a semilog plot to make visualization easier:

```
semilogx(conc, avgObjects, 'linestyle', 'none', 'marker', '.', 'markersize',
10)
ylabel('Mean Number of Objects Per Model')
xlabel('[Bafilomycin A1] (uM)')
```