OMERO Ilastik guide Documentation

Release 0.3.1

Open Microscopy Environment

May 13, 2021

Contents

ilastik is a free open-source interactive learning and segmentation toolkit, with which can be used to leverage machine learning algorithms to easily segment, classify, track and count cells or other experimental data. For more details go to see https://www.ilastik.org/. We will show how to analyze data stored in OMERO, using the ilastik user interface, Fiji and the OMERO ImageJ plugin. We will then also show how to analyze images using the ilastik API and OMERO.py API.

Contents:

CHAPTER 1

Install ilastik and OMERO Python bindings

In this section, we show how to install ilastik in a Conda environment. We will use the ilastik API to analyze data stored in an OMERO server. We will use OMERO.py to interact with the OMERO server.

1.1 Setup

We recommend to install the dependencies using Conda. Conda manages programming environments in a manner similar to virtualenv. You can install the various dependencies following the steps below (Option 1) or build locally a Docker Image using `repo2docker` (Option 2). When the installation is done, you should be ready to use the ilastik API and OMERO, see *Getting started with ilastik API and OMERO*.

The installation below is needed to run the scripts and/or notebooks. If you wish to start your own environment without the scripts/notebooks, copy locally into an `environment.yml` file the content of binder/environment.yml, remove or add the dependencies you need and run the commands below to create a conda environment.

1.1.1 *Option 1*

- Install Miniconda if necessary.
- If you do not have a local copy of the omero-guide-ilastik repository, first clone the repository:

```
$ git clone https://github.com/ome/omero-guide-ilastik.git
```

- Go into the directory:

```
$ cd omero-guide-ilastik
```

- Create a programming environment using Conda:

```
$ conda create -n ilastik python=3.7
```

- Install ilastik, its dependencies and `omero-py` in order to connect to an OMERO server using an installation file:

```
$ conda env update -n ilastik --file binder/environment.yml
```

- Activate the environment:

```
$ conda activate ilastik
```

- Make sure that `ilastik-meta` can be executed:

```
$ chmod -R +x PATH_TO_CONDA/envs/ilastik/ilastik-meta
```

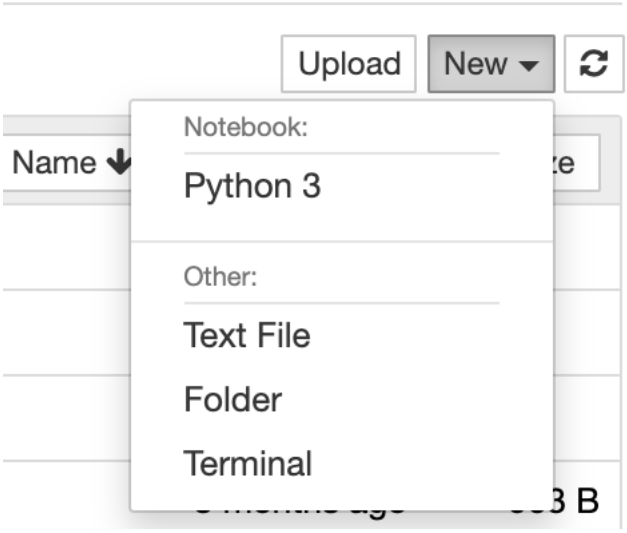
1.1.2 *Option 2*

Alternatively you can create a local Docker Image using `repo2docker`, see `README.md`:

```
$ repo2docker .
```

When the Image is ready:

- Copy the URL displayed in the terminal in your favorite browser
- Click the `New` button on the right-hand side of the window
- Select `Terminal`



- A Terminal will open in a new Tab
- A Conda environment has already been created when the Docker Image was built
- To list all the Conda environment, run:

```
$ conda env list
```

- The environment with ilastik and the OMERO Python bindings is named `notebook`, activate it:

```
$ conda activate notebook
```

CHAPTER 2

Getting started with ilastik API and OMERO

2.1 Description

We will use a Python script showing how to analyze data stored in an OMERO server using the ilastik API. The code snippets are extracted from the Python script `pixel_classification.py`. A notebook is also available, see idr0062_pixel_classification.ipynb.

We will show:

- How to connect to server.
- How load images from a dataset using the OMERO API.
- How to run ilastik using its Python API.
- How to save the generated results as OMERO images. If you are accessing a public resource e.g. idr.openmicroscopy.org, this step will not work.

2.2 Setup

We recommend to use a Conda environment to install ilastik and the OMERO Python bindings. Please read first *Install ilastik and OMERO Python bindings*.

2.3 Resources

We will use an ilastik project created with ilastik version 1.3.3 to analyze 3D images of mouse blastocysts from the Image Data Resource (IDR).

- `ilastik model`.
- Images from IDR idr0062.

For convenience, the IDR data have been imported into the training OMERO.server. This is **only** because we **cannot** save results back to IDR which is a read-only OMERO.server.

2.4 Step-by-Step

In this section, we go over the various steps required to analyse the data. The script used in this document is `pixel_classification.py`.

Connect to the server:

```
def connect(hostname, username, password):
    conn = BlitzGateway(username, password,
                        host=hostname, secure=True)
    conn.connect()
    return conn
```

Load the images:

```
def load_images(conn, dataset_id):
    return conn.getObjects('Image', opts={'dataset': dataset_id})
```

We are now ready to analyze the images:

```
def analyze(conn, images, model, new_dataset):
    # Prepare ilastik
    os.environ["LAZYFLOW_THREADS"] = "2"
    os.environ["LAZYFLOW_TOTAL_RAM_MB"] = "2000"
    args = app.parse_args([])
    args.headless = True
    args.project = model
    shell = app.main(args)
    for image in images:
        input_data = load_numpy_array(image)
        # run ilastik headless
        print('running ilastik using %s and %s' % (model, image.getName()))
        data = [ {"Raw Data": PreloadedArrayDatasetInfo(preloaded_array=input_data,␣
↪axistags=vigra.defaultAxistags("tzyxc"))}]  # noqa
        predictions = shell.workflow.batchProcessingApplet.run_export(data,
                                                                       export_to_
↪array=True)  # noqa
        for d in predictions:
            save_results(conn, image, d, new_dataset)
```

The ilastik project used expects the data array to be in the order **TZYXC**. The order will need to be adjusted depending on the order expected in the ilastik project

```
def load_numpy_array(image):
    pixels = image.getPrimaryPixels()
    size_z = image.getSizeZ()
    size_c = image.getSizeC()
```

(continues on next page)

(continued from previous page)

```
    size_t = image.getSizeT()
    size_y = image.getSizeY()
    size_x = image.getSizeX()
    z, t, c = 0, 0, 0  # first plane of the image
    zct_list = []
    for t in range(size_t):
        for z in range(size_z):  # get the Z-stack
            for c in range(size_c):  # all channels
                zct_list.append((z, c, t))
    values = []
    # Load all the planes as YX numpy array
    planes = pixels.getPlanes(zct_list)
    j = 0
    k = 0
    tmp_c = []
    tmp_z = []
    s = "z:%s t:%s c:%s y:%s x:%s" % (size_z, size_t, size_c, size_y, size_x)
    print(s)
    # axis tzyxc
    print("Downloading image %s" % image.getName())
    for i, p in enumerate(planes):
        if k < size_z:
            if j < size_c:
                tmp_c.append(p)
                j = j + 1
                if j == size_c:
                    # use dstack to have c at the end
                    tmp_z.append(numpy.dstack(tmp_c))
                    tmp_c = []
                    j = 0
                    k = k + 1
        if k == size_z:  # done with the stack
            values.append(numpy.stack(tmp_z))
            tmp_z = []
            k = 0
    return numpy.stack(values)
```

Let's now save the generated data and create a new OMERO image:

```
def save_results(conn, image, data, dataset):
    filename, file_extension = os.path.splitext(image.getName())
    # Save the probabilities file as an image
    print("Saving Probabilities as an Image in OMERO")
    name = filename + "_Probabilities"
    desc = "ilastik probabilities from Image:%s" % image.getId()
    # Re-organise array from tzyxc to zctyx order expected by OMERO
    data = data.swapaxes(0, 1).swapaxes(3, 4).swapaxes(2, 3).swapaxes(1, 2)

    def plane_gen():
        """
        Set up a generator of 2D numpy arrays.
        The createImage method below expects planes in the order specified here
        (for z.. for c.. for t..)
        """
        size_z = data.shape[0]-1
```

(continues on next page)

(continued from previous page)

```
        for z in range(data.shape[0]):  # all Z sections data.shape[0]
            print('z: %s/%s' % (z, size_z))
            for c in range(data.shape[1]):  # all channels
                for t in range(data.shape[2]):  # all time-points
                    yield data[z][c][t]

    conn.createImageFromNumpySeq(plane_gen(), name, data.shape[0],
                                 data.shape[1], data.shape[2],
                                 description=desc, dataset=dataset)
```

When done, close the session:

```
def disconnect(conn):
    conn.close()
```

CHAPTER 3

Run ilastik in parallel using dask

3.1 Description

We will show how to use dask to analyze images in parallel using the ilastik API. Binary data are stored in a public S3 repository in the Zarr format.

3.2 Setup

We recommend to use a Conda environment to install ilastik and the OMERO Python bindings. Please read first *Install ilastik and OMERO Python bindings*.

3.3 Step-by-Step

In this section, we go through the steps required to analyze the data. The script used in this document is `pixel_classification_zarr_parallel.py`.

Connect to the server:

```
def connect(hostname, username, password):
    conn = BlitzGateway(username, password,
                        host=hostname, secure=True)
    conn.connect()
    conn.c.enableKeepAlive(60)
    return conn
```

Load the images:

```
def load_images(conn, dataset_id):
    return conn.getObjects('Image', opts={'dataset': dataset_id})
```

Define the analysis function:

```
def analyze(image_id, model):
    args = app.parse_args([])
    args.headless = True
    args.project = model
    args.readonly = True
    shell = app.main(args)
    input_data = load_from_s3(image_id)
    # run ilastik headless
    data = [ {"Raw Data": PreloadedArrayDatasetInfo(preloaded_array=input_data,␣
↪axistags=vigra.defaultAxistags("tzyxc"))}]  # noqa
    return shell.workflow.batchProcessingApplet.run_export(data, export_to_
↪array=True)  # noqa
```

Helper function load the binary as a numpy array from the Zarr storage format:

```
def load_from_s3(image_id, resolution='0'):
    endpoint_url = 'https://minio-dev.openmicroscopy.org/'
    root = 'idr/outreach/%s.zarr/' % image_id
    # data.shape is (t, c, z, y, x) by convention
    with ProgressBar():
        data = da.from_zarr(endpoint_url + root)
        values = data[:]
        # re-order tczyx -> tzyxc as expected by the ilastik project
        values = values.swapaxes(1, 2).swapaxes(2, 3).swapaxes(3, 4)
        return numpy.asarray(values)
```

Start the Dask client and a local cluster:

```
        cluster = LocalCluster()
        client = Client(cluster)
```

Use the `Dask Future API`. The work starts immediately as we submit work to the cluster:

```
def prepare(client, images, model):
    futures = [client.submit(analyze, i.getId(), model) for i in images]
    return futures
```

We wait until this work is done and gather the results to our local process:

```
def gather_results(client, futures):
    return client.gather(futures)
```

When done, close the session:

```
def disconnect(conn):
    conn.close()
```

In order to use the methods implemented above in a proper standalone script: **Wrap it all up** in `main`:

```
def main():
    # Collect user credentials
    try:
        host = input("Host [wss://outreach.openmicroscopy.org/omero-ws]: ") or 'wss://
↪outreach.openmicroscopy.org/omero-ws'  # noqa
        username = input("Username [trainer-1]: ") or 'trainer-1'
        password = getpass("Password: ")
        dataset_id = input("Dataset ID [6161]: ") or '6161'
        # Connect to the server
        conn = connect(host, username, password)

        # path to the ilastik project
        ilastik_project = "../notebooks/pipelines/pixel-class-133.ilp"

        # Load the images in the dataset
        images = load_images(conn, dataset_id)

        # prepare ilastik
        os.environ["LAZYFLOW_THREADS"] = "2"
        os.environ["LAZYFLOW_TOTAL_RAM_MB"] = "2000"

        # Create-client
        cluster = LocalCluster()
        client = Client(cluster)
        # End-client

        futures = prepare(client, images, ilastik_project)

        start = time.time()
        results = gather_results(client, futures)
        done = time.time()
        elapsed = (done - start) // 60
        print("Compute time (in minutes): %s" % elapsed)
        save_results(results)
    finally:
        disconnect(conn)

    print("done")


if __name__ == "__main__":
    main()
```

CHAPTER 4

Use ilastik as a Fiji plugin and OMERO

4.1 Description

In this section, we will show how to use the ilastik user interface to perform segmentations on multi-z images stored in OMERO. The connection between OMERO and ilastik is facilitated via Fiji, for which both OMERO and ilastik have plugins. The segmentation steps in this part are recorded and saved in the form of an `ilp` file in ilastik (ilastik project). The `ilp` file is used later for the scripting workflow.

We will show:

- How to manually open images from OMERO in ilastik using the Fiji plugin for OMERO and Fiji plugin for ilastik.
- How to segment the multi-z images in ilastik and produce an ilastik Project (`ilp` file) recording the steps.
- How to save the results of the segmentation (ROIs and Probability maps) in OMERO, using the manual workflow and Fiji.
- How to run a script in Fiji, consuming the `ilp` file and running the segmentation of the images coming from an OMERO Dataset, saving the ROIs on the original images in OMERO.
- How to manually classify images.

4.2 Setup

ilastik installation

- ilastik has been installed on the local machine. See https://www.ilastik.org/ for details.

ilastik plugin for Fiji installation instructions

- Start Fiji. Update it (`Help > Update ImageJ`).
- In the `Manage Update Sites` check the checkbox next to the "ilastik" site.
- After the update was successful, restart your Fiji.

- The new ilastik menu item should be under Plugins menu.

Note: The ilastik menu item might be the last in the Plugins dropdown, not necessarily alphabetically ordered.

OMERO plugin for Fiji installation instructions

- For installation instructions, go to Fiji installation.

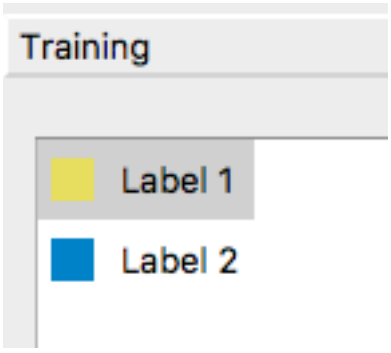
4.3 Resources

- Images from IDR idr0062.

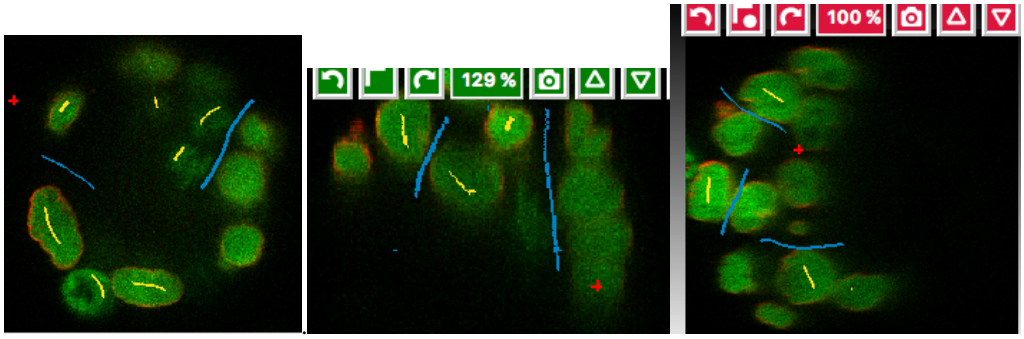
4.4 Step-by-step

4.4.1 Manual training of z-stack segmentation in ilastik

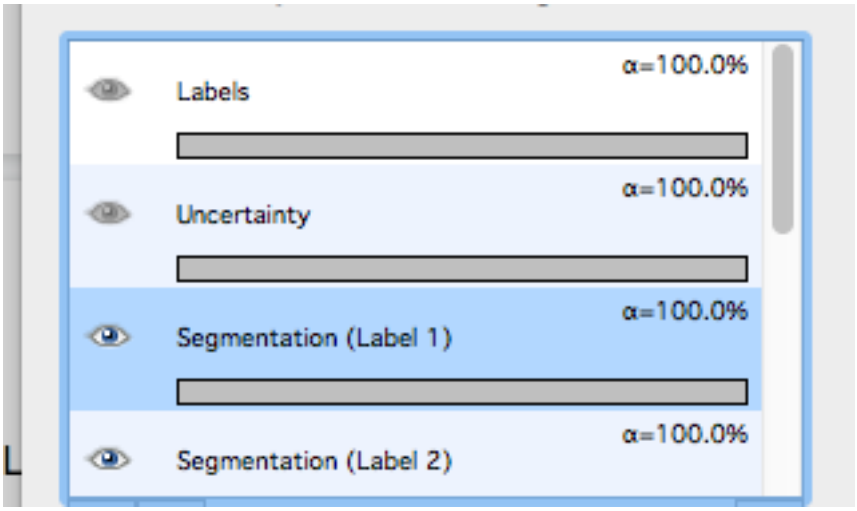
1. Open Fiji, go to `Plugins > OMERO > Connect` to OMERO and connect to OMERO.server provided using the credentials provided.
2. Find the idr0062 Project, the Blastocysts Dataset, open the first image in Fiji.
3. After image has opened in Fiji, go to `Plugins > ilastik > Export HDF5`. The ilastik menu item might be the last in the Plugins dropdown, not necessarily alphabetically ordered.
4. Select a local directory to export to and save the image locally as an `.h5` file.
5. Repeat this step with several images from the Blastocysts Dataset of idr0062.
6. Start ilastik.
7. Click on `Pixel Classification`.
8. Save a new Project in ilastik.
9. Still in ilastik, open the image you saved as `.h5` in previous steps above (central pane, `Add file` button).
10. Three views will open, `xy`, `xz` and `yz`. You can explore the orthogonal views by clicking onto the checkbox in bottom right corner.
11. In Left-hand pane, click `Feature Selection`. Select all available features.
12. You can explore the features at the bottom left corner, but this takes time.
13. Click on the `Training` harmonica in the Left-hand pane.
14. The training UI comes in left-hand pane with two labels already pre-defined by default.



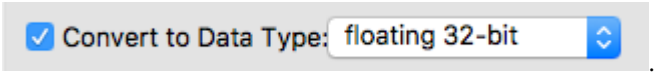
15. Select the first label, and by drawing LINES into the images, select a couple of cells in all three views.



16. Select the second label, and again drawing lines, select some background (also select the narrow `channels` between two almost adjacent cells as `bckgr` (draw a line through them).
17. Click on the `Live Update` button - this will take time, as the image has a large number of planes.
18. Add new lines on cells which are too dim to be selected.
19. Click on `Live Update`.... Repeat.
20. Stop `Live Update`
21. Click on `Suggest Features` button (to the left of `Live Preview` button).
22. A new UI window will open.
23. Click on `Run Feature Selection` in the left-hand pane of this new window. This will take time.
24. Click on `Select Feature Set` button in the bottom middle of the window.
25. The `Suggest Features` window will close on this and you are back in the main ilastik window.
26. Click `Live Update` again.
27. Toggle the images produced visible or not using the `eye` icons and the rendering settings of the particular images in the list in bottom-left corner. Below is an example of viewing the `Segmentation Label 1` and `Segmentation Label 2` layers viewable, the other layers (e.g. `Raw data`) are toggled invisible.



28. Add new lines if some segmentation still does not look right.
29. Click on the `Prediction Export` harmonica tab. In this tab, we will prepare the parameters of the exported images only, and will do the exporting itself later using the `Batch processing` harmonica.
30. In the `Prediction Export` harmonica, select the features to be exported in the `Source` dropdown menu in the left-hand pane. Export sequentially `Probabilities` and `Simple Segmentation` for all three images you opened from OMERO via Fiji, using the `Batch processing` harmonica tab, see below.
31. First, start with selecting `simple Segmentation` in the `Choose Export Image Settings`, select the `Convert to data Type` parameter to be `floating 32 bit`

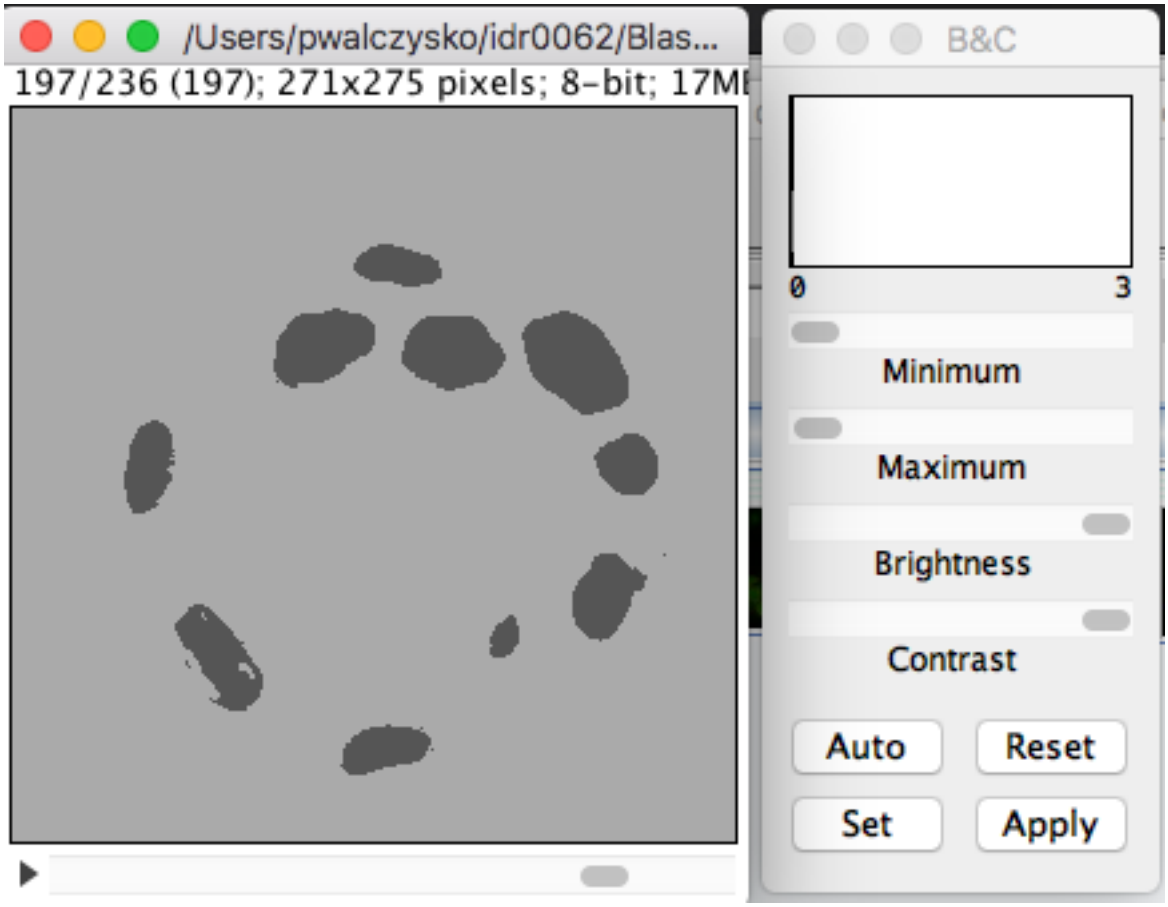


. The files will be exported into the folder where the original images were, unless you choose otherwise.By default, the export format is HDF5 (file extension `.h5`).
32. Now, select in the left-hand pane the harmonica `Batch processing`. In the centre top row of the view, click on `Select Raw Data Files...`. Select all the three raw `.h5` files on your local machine, including the one you have just trained your pixel classification on.
33. Click onto the `Process all data files` button in the left-hand pane.
34. This will create three `.h5` files in the folder you have chosen in the `Choose Export Image Settings` window (by default, these files will be placed in the folder where your raw data exports from OMERO are), the files will be named `...Simple Segmentation.h5`.

35. Return to `Prediction Export` harmonica, select the `Probabilities` parameter in the `Source` dropdown. Go to the `Batch processing` harmonica and click onto the `Process all data files` button in the left-hand pane. This will create another three `.h5` files in the local folder, named `... Probabilities.h5`.

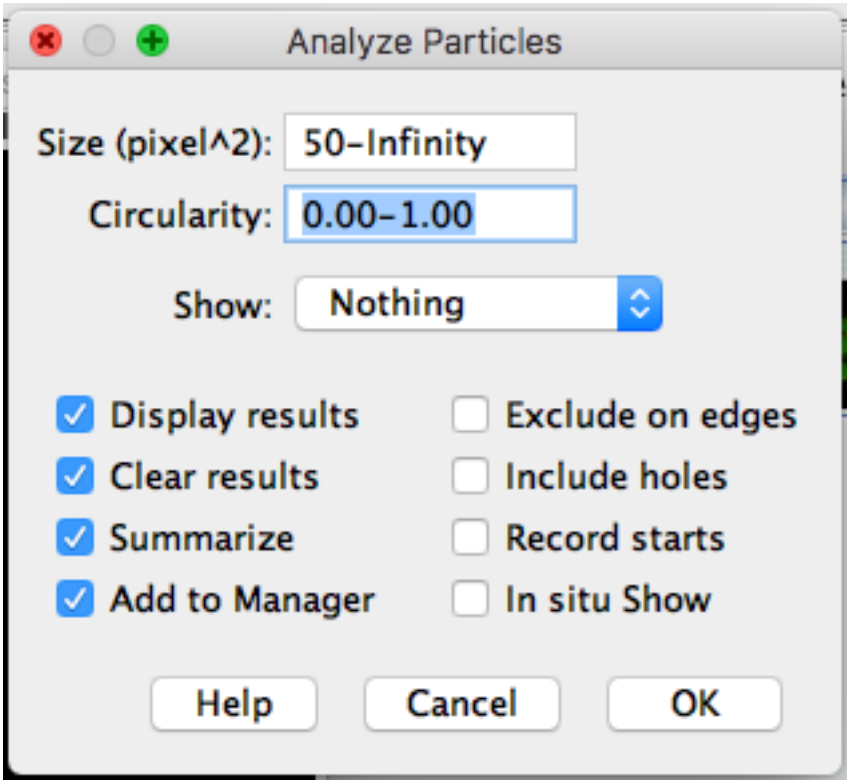
4.4.2 Manual creation of ROIs in Fiji based on segmentations from ilastik and saving the ROIs to OMERO

1. Go to Fiji, `Plugins > Ilastik > Import...`
2. Browse to one of the "..._Simple Segmentation.h5" files which was created in ilastik in previous step and set the "Axis Order" to tzyxc (this might be the default for you). Do not check the checkbox `Apply LUT`. Click `OK`.
3. The 3D image will open in Fiji. Select `Image > Adjust > Brightness and Contrast`. Adjust the `max` slider to the left, until you see the image grow grey (it is probably black just after opening).

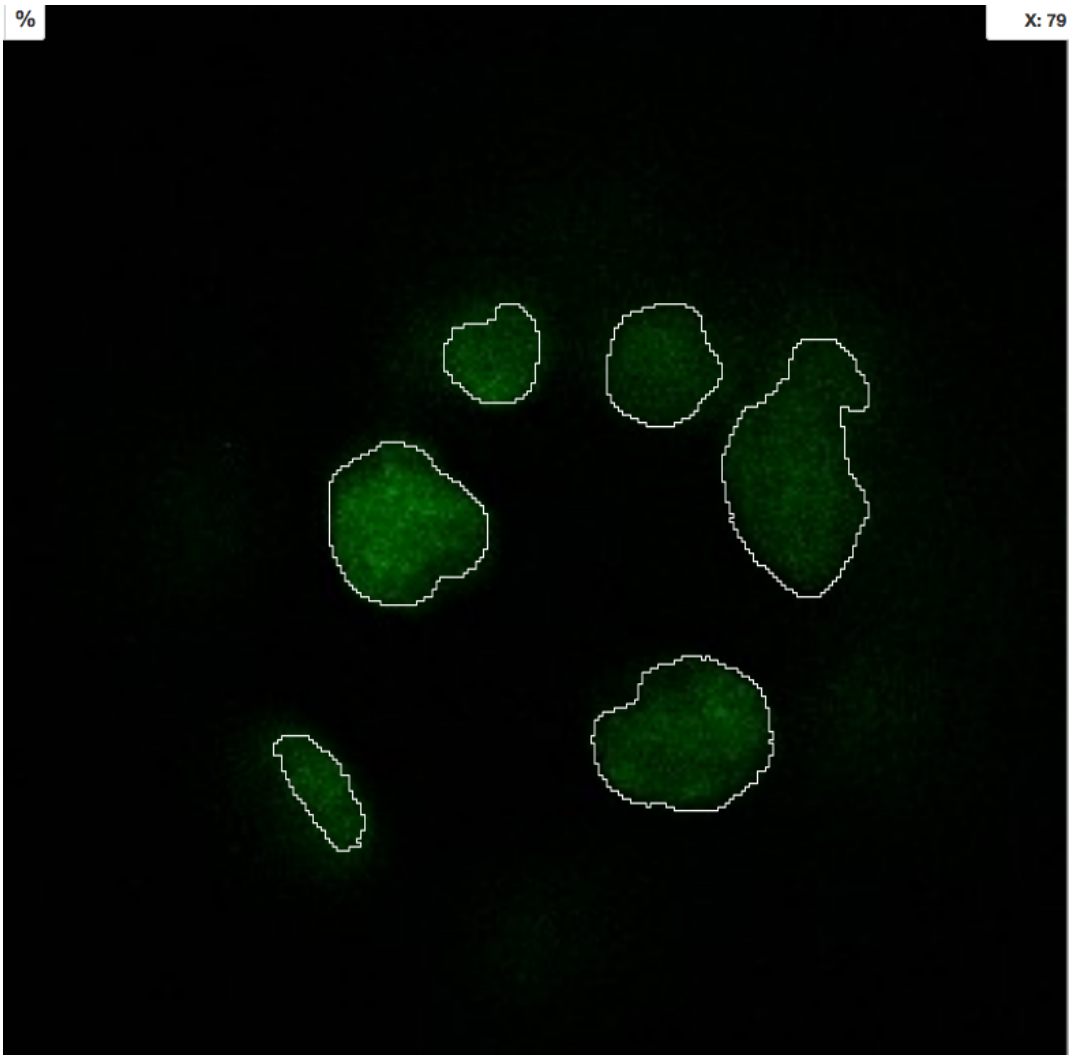


4. Note: Because in ilastik, the `Simple Segmentatio` images have the values of 2 where there is an object and 1 for Background, we need to invert the image for Object Analysis in Fiji. The object analysis (done by the ``Analyze particles` plugin) is done in order to create ROIs which can be saved to OMERO.
5. Select `Image > 8 bit`. This will convert the values in the image into either 0 (cells) or 255 (background).
6. Select `Edit > Invert`. This is needed for the subsequent `Analyze particles` plugin - white objects on black background.

7. Select `Analysis > Analyze Particles`.
8. Change the `Size(pixel^2)` parameter to `50-infinity`



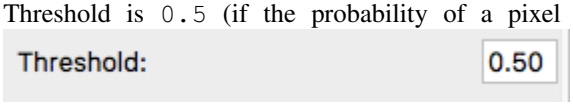
9. Click `OK` and in the next dialog answer `Yes`.
10. Select `Plugins > OMERO > Save image(s)` to OMERO. In the importer dialog, select the target Project and Dataset in OMERO or choose a new one.
11. This will import the `Simple segmentation` image into OMERO with the ROIs from Fiji on it and the contents of the Results table will be attached to this new image.
12. In order to have the ROIs from Fiji also on the original, raw image in OMERO.
13. Do not close the ROI Manager and the Results table.
14. Open the original raw image from OMERO into Fiji.
15. Click on the opened image.
16. Select `Plugins > OMERO > Save ROI(s)` to OMERO (alternatively, you can re-run the analysis in Fiji by clicking on `Measure` in the ROI manager of Fiji to produce a new Results table).
17. In the new dialog, select a name for your results table which will be attached now to the original image.
18. Click `OK`.
19. ROIs and results will be now added to the original, raw image in OMERO



20. Repeat this workflow with the `...Probabilities.h` files. Also, attach the ilastik project itself to the Dataset containing original data in OMERO.

4.4.3 Manual workflow of Object classification on z-stacks in ilastik

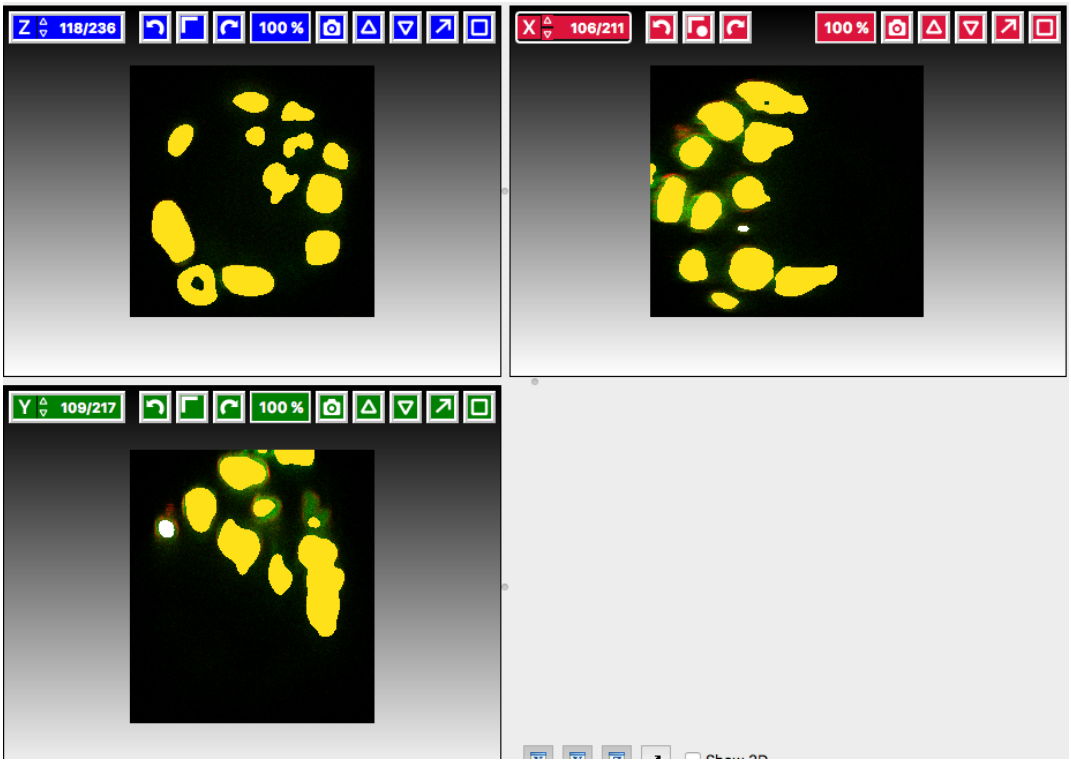
1. Start ilastik, choose the `Object classification with Prediction maps` option and create a new Project and save it.
2. Select in the `Raw data` tab the raw image stored locally and in the `Prediction maps` tab the prediction map which you saved from the `Pixel classification` module for this image previously.
3. Click on `Threshold and Size filter` harmonica in the left-hand pane. This step discerns the objects form background by means of thresholding (note that the "Prediction maps" values are between 0 and 1, where 1 is 100% probability that the pixel is a cell, 0 is a 100% probability that the pixel is backgr.) The other parameter to specify the object except threshold in this tab is size of the object.
4. Threshold is `0.5` (if the probability of a pixel is higher than 0.5, then it is deemed to be a cell)



5. Change `Size` to `minimum 50` 

.
6. Leave the rest of the parameters at default and click `Apply`.
7. A new image will be added to the stack at bottom left called `Final output`. The objects are displayed on it in color coding. Again, you can toggle the images visible and change intensities in bottom left corner.
8. Click on `Object Feature Selection` harmonica and click on the button `Select Features`.
9. In the new window, click on `All excl. Location` button to select almost all features.
10. Click on the `Label classes` harmonica, click on the yellow label (Label 1) 

and select all the cells in all three orthogonal views images.



11. Click on `Object information export` harmonica.
12. Changing the `Source` dropdown menu, export sequentially `Object Predictions` and `Object Probabilities`.
13. Click on `Configure Feature Table Export` button in the left-hand pane and configure the location of the exported Also, changing the export format of the table in the `Format` dropdown menu, export sequentially the table as HDF as well as CSV format. 

14. In the `Features` harmonica, click the `All` button to export all features.
15. Click `OK`.
16. Back in the main ilastik interface, click `Export All` (repeat as necessary to export all formats of the images and the two formats of the export table).

17. Save the Project.
18. Import the CSV to OMERO, as well as the Probabilities.
19. Make an OMERO.table out of the CSV and attach it on the Project in OMERO. This can be done using populate_metadata.py plugin or from scratch using the extended groovy script from Fiji.

CHAPTER 5

Use ilastik using Fiji scripting facility and OMERO

5.1 Description

In this section, the segmentation (using Pixel classification routine of ilastik) of the multi-z images is run in a batch mode. For this we provide scripts which run ilastik in a headless mode. The scripts provide for ilastik a batch of images (coming from an OMERO Dataset) and ilastik is segmenting these images according to the parameters configured and saved in the `ilp` in the manual step above. We offer two scripts covering this workflow, one running in Fiji, and the other using the python frames to export images directly from OMERO to the ilastik running headlessly. Also, we describe in this part how to use ilastik routine Object classification to classify objects on images from OMERO manually.

We will show:

- How to run a script in Fiji, consuming the `ilp` file and running the segmentation of the images coming from an OMERO Dataset,
- How to save the ROIs on the original images in OMERO.

5.2 Setup

ilastik installation

- ilastik has been installed on the local machine. See https://www.ilastik.org/ for details.

ilastik plugin for Fiji installation instructions

- Start Fiji. Update it (`Help > Update ImageJ`).
- in the `Manage Update Sites` check the checkbox next to the "ilastik" site.
- After the update was successful, restart your Fiji.
- The new ilastik menu item should be under Plugins menu.

Note: The ilastik menu item might be the last in the Plugins dropdown, not necessarily alphabetically ordered.

OMERO plugin for Fiji installation instructions

- For installation instructions, go to Fiji installation.

5.3 Resources

- Images from IDR idr0062.
- Groovy script `analyse_dataset_ilastik.groovy`

5.4 Step-by-step

5.4.1 Scripting workflow on z-stacks using ilastik headless, Fiji and OMERO

For this example we will use the Groovy script `analyse_dataset_ilastik.groovy`. The script uses the OMERO JAVA API.

Connect to the server:

```
def connect_to_omero() {
    "Connect to OMERO"

    credentials = new LoginCredentials()
    credentials.getServer().setHostname(HOST)
    credentials.getServer().setPort(PORT)
    credentials.getUser().setUsername(USERNAME.trim())
    credentials.getUser().setPassword(PASSWORD.trim())
    simpleLogger = new SimpleLogger()
    gateway = new Gateway(simpleLogger)
    gateway.connect(credentials)
    return gateway

}
```

Load the images contained in the specified dataset:

```
def get_images(gateway, ctx, dataset_id) {
    "List all images contained in a Dataset"

    browse = gateway.getFacility(BrowseFacility)
    ids = new ArrayList(1)
    ids.add(new Long(dataset_id))
    return browse.getImagesForDatasets(ctx, ids)
}
```

Open the images one-by-one using the Bio-Formats plugin:

```
def open_image_plus(HOST, USERNAME, PASSWORD, PORT, group_id, image_id) {
    "Open the image using the Bio-Formats Importer"
```

(continues on next page)

(continued from previous page)

```
    StringBuilder options = new StringBuilder()
    options.append("location=[OMERO] open=[omero:server=")
    options.append(HOST)
    options.append("\nuser=")
    options.append(USERNAME.trim())
    options.append("\nport=")
    options.append(PORT)
    options.append("\npass=")
    options.append(PASSWORD.trim())
    options.append("\ngroupID=")
    options.append(group_id)
    options.append("\niid=")
    options.append(image_id)
    options.append("] ")
    options.append("windowless=true view=Hyperstack ")
    IJ.runPlugIn("loci.plugins.LociImporter", options.toString())
}
```

Export each image as h5 to a local folder specified interactively by the user during the run of the script. It is assumed that the folder specified by the user contains the ilastik Project prepared beforehand. The export is facilitated by the ilastik plugin for Fiji.

```
        IJ.run("Export HDF5", args);
        imp = IJ.getImage()
        imp.close()
```

Start ilastik headless, using the `Pixel classification` module The script feeds into the `Pixel classification` ilastik module the ilastik Project created during the manual step and also the raw the new created h5 image in the step above.

```
    args = "select=" + output_file + " datasetname=" + inputDataset +  " axisorder="␣
↪+ axisOrder
    println "opening h5 file"
    IJ.run("Import HDF5", args)
    // run pixel classification
    input_image = output_file + "/data";
    args = "projectfilename=" + pixelClassificationProject + " saveonly=false␣
↪inputimage=" + input_image + " chosenoutputtype=" + outputType;
    println "running Pixel Classification Prediction"
    IJ.run("Run Pixel Classification Prediction", args);
```

The headless ilastik `Pixel classification"` module produces `Probabilities` map - this map is immediately opened into Fiji via the ilastik plugin for Fiji.

In Fiji, the Analyze Particles plugin is run on the "Probabilities" map to produce ROIs.

```
    IJ.run("8-bit")
    //white might be required depending on the version of Fiji
    IJ.run(imp, "Auto Threshold", "method=MaxEntropy stack")
    IJ.run(imp, "Analyze Particles...", "size=50-Infinity pixel display clear add␣
↪stack summarize")
```

Once the ROIs are produced, they are saved to OMERO onto the original image which.

```
def save_rois_to_omero(ctx, image_id, imp) {
    " Save ROI's back to OMERO"
    reader = new ROIReader()
    roi_list = reader.readImageJROIFromSources(image_id, imp)
    roi_facility = gateway.getFacility(ROIFacility)
    result = roi_facility.saveROIs(ctx, image_id, exp_id, roi_list)
}
```

Disconnect when done

```
gateway.disconnect()
```

CHAPTER 6

Track mitosis using ilastik

6.1 Description

In this section, a manual tracking workflow is shown on images of cells undergoing mitosis. The lineage of the cells is being followed. The images are timelapses from the Image Data Resource, the "mitocheck" set. As a result of this step, again, an `ilp` file is produced and saved for further use by the follow-up scripting workflow, similarly to the steps one and two described for the multi-z images above.

6.2 Setup

ilastik installation

- ilastik has been installed on the local machine. See https://www.ilastik.org/ for details.

6.3 Step-by-step

1. Open ilastik, create a new Pixel classification project, feeding in the raw data in h5 form. The data come from https://www.ilastik.org/download.html, more concretely the "Mitocheck 2D+t" download https://data.ilastik.org/mitocheck.zip. Download, unzip and feed the h5 file which has not the "export" in its name into this step (Pixel classification).
2. Follow the steps of Pixel classification as described above in the idr0062 workflow - you will have to
    - Adjust the parameters, saving the new project as "mitocheck-pixel-class.ilp"
    - Export "Probabilities", which can be exported as "mitocheck_94570_2D+t_01-53_Probabilities.h5"
    - Close and reopen ilastik. Open the projec "conservationTracking.ilp" from the folder you downloaded from the ilastik site. In the "Raw data", tab of "Input data" make sure the raw data are pointing to where you have your "mitocheck_94570_2D+t_01-53.h5" file locally. Further, in the "Prediction maps" tab of "Input data", exchange the file there by right-clicking on it and selecting the "Replace with file" and

replace this file with the "mitocheck_94570_2D+t_01-53_Probabilities.h5" which you exported from the Pixel classification workflow (see points above)

- Run through the tabs in the LHP, making sure that when Thresholding, you swap the blue and yellow objects (my Pixel class. produced a probabilities map which is swapped in the sense objects vs bckgr coloring). Also, you have to manually select the cells which are dividing and not dividing in the corresponding tabs in LHP in quite a few timeframes, see https://www.ilastik.org/documentation/tracking/tracking#3-division-and-object-count-classifiers for how to do it.
- Further, you have to discern false detections, and 1 object and 2 object blobs manually on quite a few frames, the LHP harmonice is called Object Count classification, as described in https://www.ilastik.org/documentation/tracking/tracking#3-division-and-object-count-classifiers, second part.
- Once done, in the Tracking tab in left-hand paneHP, click on "Track !" button, making sure you did not change any params inadvertently. This will take a while.
- **Select the "Tracking Results Export" tab in LHP and define your export target dir, then export in a row**

  - "mitocheck_94570_2D+t_01-53_Object-Identities.h5",
  - "mitocheck_94570_2D+t_01-53_Tracking-Result.h5",
  - "mitocheck_94570_2D+t_01-53_Merger-Result.h5" and
  - "mitocheck_94570_2D+t_01-53_CSV-Table.h5.csv"

  These are 3 timelapses and one CSV with the tracking results.
- Save the Project as "mitocheck-tracking-serious.ilp". This is the main starting point for the automatic pipeline from OMERO. The pipeline is
  - "mitocheck-pixel-class.ilp" which
    - consumes the "mitocheck_94570_2D+t_01-53.h5"
    - produces the "mitocheck_94570_2D+t_01-53_Probabilities.h5"
  - "Mitocheck-tracking-serious.ilp" which
    - consumes
      - "mitocheck_94570_2D+t_01-53.h5"
      - "mitocheck_94570_2D+t_01-53_Probabilities.h5"
    - produces the outputs
      - "mitocheck_94570_2D+t_01-53_Object-Identities.h5"
      - "mitocheck_94570_2D+t_01-53_Tracking-Result.h5"
      - "mitocheck_94570_2D+t_01-53_Merger-Result.h5"
      - "mitocheck_94570_2D+t_01-53_CSV-Table.h5.csv"

CHAPTER 7

Contribute

Changes to the documentation or the materials should be made directly in the omero-guide-ilastik repository.