Opleiding Informatica

Comparing and improving different analysis workflows for High compound screening data

Robbert Versluis

Supervisors:
Rohola Hosseini
Lu Cao

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

30/07/2021

Abstract

This bachelor thesis investigates the performance of different High Content Screening (HCS) analysis tools based on Cellprofiler by using a test screen and measuring how long it takes to fully analyze it. The analysis tools are also tested on their ease of use. The CellProfiler app itself produced results the fastest, being three times faster than Jupyter Notebook and around fifteen times faster than Galaxy. For Jupyter and Galaxy, methods to improve their performance have been attempted. These methods would add multi-processing to the analysis tools which increased performance. However, it didn't had much success, as introducing multi-processing was either impossible in the case of Galaxy or brought with it more complications than could be forseen for Jupyter. After considering all factors, Jupyter Notebook has the most promise as HCS analysis tool as it is a very flexible tool where new scripts can easily be added to.

Content

1. Introduction

1.1 High-Content Screening

Imaging many cell phenotypes under different conditions

HCS, or High Content-Screening, is an image analysis technique originating from the 90’s. This technique has become more and more popular over time because technology has improved so much as well, and automated microscopes can now acquire images faster than the human eye can analyze them (Li et al., 2019) [1]. HCS is a phenotypic screening technique, which means analyzing differences in the appearance (or phenotype) of cells after they have been subjected to compounds, such as candidate drugs or small interfering RNAs (siRNA) to modify gene expression in cells. The simplest example of a phenotypic change is the death of a cell. Other changes would be examining the localization of cell organelles after being exposed to different compounds.

How to use HCS

HCS are performed on multiwell plates, which are either 96-, 384- or 1536-well plates, with the 384-well plate being used most frequently (Boutros et al., 2015) [2]. To perform an HCS, cells first need to be grown in the multiwell plate. In each well, around 10 cells are grown to a size of over 100 cells. Secondly, the cells in each well are treated with the compound of your choice of interest. As an example, a siRNA library containing thousands of siRNA molecules can be screened using a well of cells for each molecule can be added to investigate the function of onco-related genes upon inhibition of these genes. Thirdly, to visualize the phenotypes of the cells, fluorescent probes are used that stain the cell. Ideally, these probes are specific, only stains the desired organelles or target molecules of choice, and bright, meaning it must be bright enough to have short exposure times. It is possible to combine multiple probes within the same sample and images are taken simultaneously using different channels for each probe. The signal recorded for each channel is filtered using bandpass filters for emitted light. In case of using multiple probes within the same sample, the bleed-through must be minimized between channels and these effects must be taken into account during image analysis (Boutros et al., 2015). Fourthly, images are acquired for each well and plate in study and analyzed.

Types of HCS

HCS can be subdivided in two types that could be performed. The first type of HCS is fixed cells with fluorescently labeled targets. The second type of HCS is live cells with multicolor fluorescent indicators and “biosensors”. Fixed cell assays are the simplest, because the initially living cells can be subjected to a multitude of different compounds before fixing them. After the cells are fixated, there is no need to deal with any environmental variables. The downside is that the information acquired from the cells consists solely of the single time point of fixing them. Fixating and labeling can be automated to process the assays efficiently. Live cell assays yield more information because they are measured both over space and time. The environment of the cells needs to be controlled in this case, as a different environment can change their behavior. (Giuliano et al., 1997) [3].

Image acquisition, more cells is more time or lower resolution
Within the image acquisition, the option exists to only take images of part of each well, rather than the entire well. This can be done to increase the resolution of the images, such as when analyzing subcellular structures (Boutros et al., 2015). This does increase the time it takes to analyze the images.

Advantage of imaging cell dynamics
The strength of HCS is that images are taken over time as well as over space. It's almost like analyzing a video recording of the cell. The effects of compounds are measured over time and more accurate conclusions can be drawn from the images this way. The entire screening procedure can be automated, which includes cell seeding and pipetting during the fixation- or staining-steps of the process (Laufer et al., 2014).

Image Analysis in HCS
The analysis steps in HCS are performed by using different software's and tools. The entire analysis can be divided in two parts, upstream analysis and downstream analysis. Upstream analysis part resolves and interprets the image data to tabular information. The downstream analysis part refers to extracting data, visualizing data, plotting data, and performing statistics on tubular data retrieved from the previous part to draw conclusions and test hypotheses. This is done because automated microscopes can now acquire images faster than the human eye can analyze them (Li et al., 2019).

1.2 FAIR

FAIR data is an acronym for findability, accessibility, interoperability and reusability. In 2016, a publication by a number of scientists [4] introduced FAIR as a set of principles that "may act as a guideline for those wishing to enhance the reusability of their data holdings". The principles improve the findability, accessibility, interoperability and reusability of data. Recently, HCS-data itself has become more in line with the FAIR principles, but the software tools that analyze HCS aren't necessarily following FAIR as well. There aren't any guidelines for FAIR software.

1.3 Research Question

This research project investigates a FAIR-workflow for the analysis of High Content Screening results. The open-source software tools CellProfiler app, Jupyter Notebook and Galaxy will be tested on their performance in analyzing high content screens. In addition, we will examine/review the usability and FAIRness of these tools. If applicable, the workflows of these tools will be improved using multiprocessing, resulting in better performance.

2. Materials & Methods

2.1 CellProfiler

Features of CellProfiler

CellProfiler[5] is a tool for quantifying images. This means as much as counting and measuring the images given to it. A process called a pipeline is used to get the desired output of the images. A pipeline consists of a sequence of modules with each module advancing the process. Examples include processing an image by identifying objects on the image, resizing the image, measuring overlap between objects, creating a histogram based on measurements taken earlier, or exporting findings to a spreadsheet. Each module also comes with extensive documentation, telling the user what it does and what it can be used for. All the variables within a module have similar documentation.

Modularity of CellProfiler's pipeline

CellProfiler’s pipeline is modular. Every module in that pipeline can be changed without any trouble, if the user wants to. The modules can be mixed and matched until the desired output is reached. This modular nature also means that modules can be put in an already existing pipeline. After editing a few variables of other modules in the pipeline, the process works without any trouble. This is in part thanks to CellProfiler’s graphical user interface.

Interface of CellProfiler

CellProfiler’s interface is a Graphical User Interface (GUI). This means that users will have an easier time getting used to the program, and will have less trouble finding their way around CellProfiler. It also works intuitively. For example, a pipeline made in CellProfiler can be saved. If that pipeline is needed when using CellProfiler at a later time, it can be dragged and dropped into the pipeline-area straight from the documents folder, and the pipeline is directly imported.

Documentation of CellProfiler

If something is unclear to the user, the help-tab of CellProfiler offers as much documentation as the modules themselves have. It is also open-source, so edits can be made to a personal version to improve upon it.

CellProfiler can be seen as a deterministic tool. This means that the outcome is not random. If the same images are sent as input and use the same pipeline, the result will always be the same. The version used in this project is 4.0.7.

2.2 Jupyter Notebook

Just like CellProfiler, Jupyter[6] is a tool that is being used in HCS image analysis. In this project, CP pipelines are imported into Jupyter, which allows it to run CellProfiler in “headless mode”, or using CellProfiler pipelines without the CellProfiler application.

Starting Jupyter

Instead of using an application to launch Jupyter, it is started from the command line. In this project, a Windows computer with the ubuntu 18.04 LTS-application from the microsoft store is used to have a linux environment in which Jupyter is called from. Anaconda is installed as well. Anaconda is used to make an environment in which Jupyter Notebook is installed and launched from. When Jupyter is started, a link will be given to the user, which will be copy-pasted into an internet browser from which Jupyter is used.

Features of Jupyter

When Jupyter is used in a browser, the user will first be able to select which file they want to edit and run. When a file is selected, Jupyter will open the interactive environment in which this file can be run and edited. This works through multiple cells, where each cell can either be code or markdown that can be created. Cells with markdown will be ignored when running the cells and are solely explanatory. Examples of this would be markdown cells which tell what program does or what a code-cell below requires as input. The code cells contain programming code. Each cell can be run individually. This can be compared to having a program where parts of the code can be run separately from the rest of the program. For example, the first part of the code that will be executed will be the part where the dependencies and imports are. The next part of code can then be the definitions of the program. Jupyter will return the output, if any, when a cell has finished running. Jupyter will also output errors. Jupyter notebook is compatible with several programming languages, like Python, R or MatLab. The Jupyter Notebook version used is 6.1.4.

2.3 Galaxy

Galaxy[7] is a website which can be used to perform HCS analysis. It has sequential, modular modules, just like the CellProfiler app. Galaxy allows the user to select modules from the search bar to put in their history, which is their workflow, and edit parameters of those modules. Calculations for Galaxy are done on the galaxy server, rather than on a local computer. If modules are inserted in the history, Galaxy will immediately execute them as a job.

In a CellProfiler setting, Galaxy offers CellProfiler-like modules which are the same as the ones used in the application, or existing CellProfiler pipelines can be imported, and Galaxy allows data to be imported from remote databases. It also offers tutorials created by the community on how to create certain workflows. [8]

2.4 Computer

The hardware used in testing consist of a quad-core Intel i5-7500 CPU (3.4 GHZ) and 16 GB of RAM. The computer used Windows 10, and to simulate Ubuntu, used the Ubuntu 18.04 LTS shell.

3. Implementation

3.1 CellProfiler app

To use CellProfiler, images and a workflow need to be imported into the application. There are a multitude of modules available to add to a workflow. An example of a CellProfiler workflow that's used in this research can be seen in figure 1. An explanation for the modules is taken from the documentation of the CellProfiler app itself.

Images
The Images module allows a location to be specified of files to be analyzed by the pipeline; setting this module correctly is the first step in creating a new project in CellProfiler. These files can be located on a local hard drive, on a networked computer elsewhere, or accessible with a URL. Rules can also be provided to specify only those files that need to be analyzed out of a larger collection (for example, from a folder containing both images for analysis and non-image files that should be disregarded).

Metadata
The Metadata module allows metadata to be extracted and associated with the images. The metadata can be extracted from the image file itself, from a part of the file name or location, and/or from a text file.

NamesAndTypes
The NamesAndTypes module gives images and/or channels a meaningful name to a particular image or channel, as well as defining the relationships between images to create an image set. Once the relevant images have been identified using the Images module (and/or has had metadata associated with the images using the Metadata module), the NamesAndTypes module gives each image a meaningful name by which modules in the analysis pipeline will refer to it.

Groups
The Groups module organizes sets of images into groups.

EnchanceOrSurpressFeatures
EnhanceOrSuppressFeatures enhances or suppresses certain image features (such as speckles, ring shapes, and neurites), which can improve subsequent identification of objects.

This module enhances or suppresses the intensity of certain pixels relative to the rest of the image, by applying image processing filters to the image. It produces a grayscale image in which objects can be identified using an Identify module.

IdentifyPrimaryObjects
IdentifyPrimaryObjects identifies biological objects of interest. It requires grayscale images containing bright objects on a dark background. Incoming images must be 2D (including 2D slices of 3D images)

MaskObjects
MaskObjects removes objects outside of a specified region or regions.

This module allows objects or portions of objects that are outside of a specified region (mask) to be deleted. For example, after identifying nuclei and tissue regions in previous Identify modules, all nuclei that are outside of a tissue region can be excluded.

If using a masking image, the mask is composed of the foreground (white portions); if using a masking object, the mask is composed of the area within the object. The user can choose to remove only the portion of each object that is outside of the region, remove the whole object if it is partially or fully outside of the region, or retain the whole object unless it is fully outside of the region.

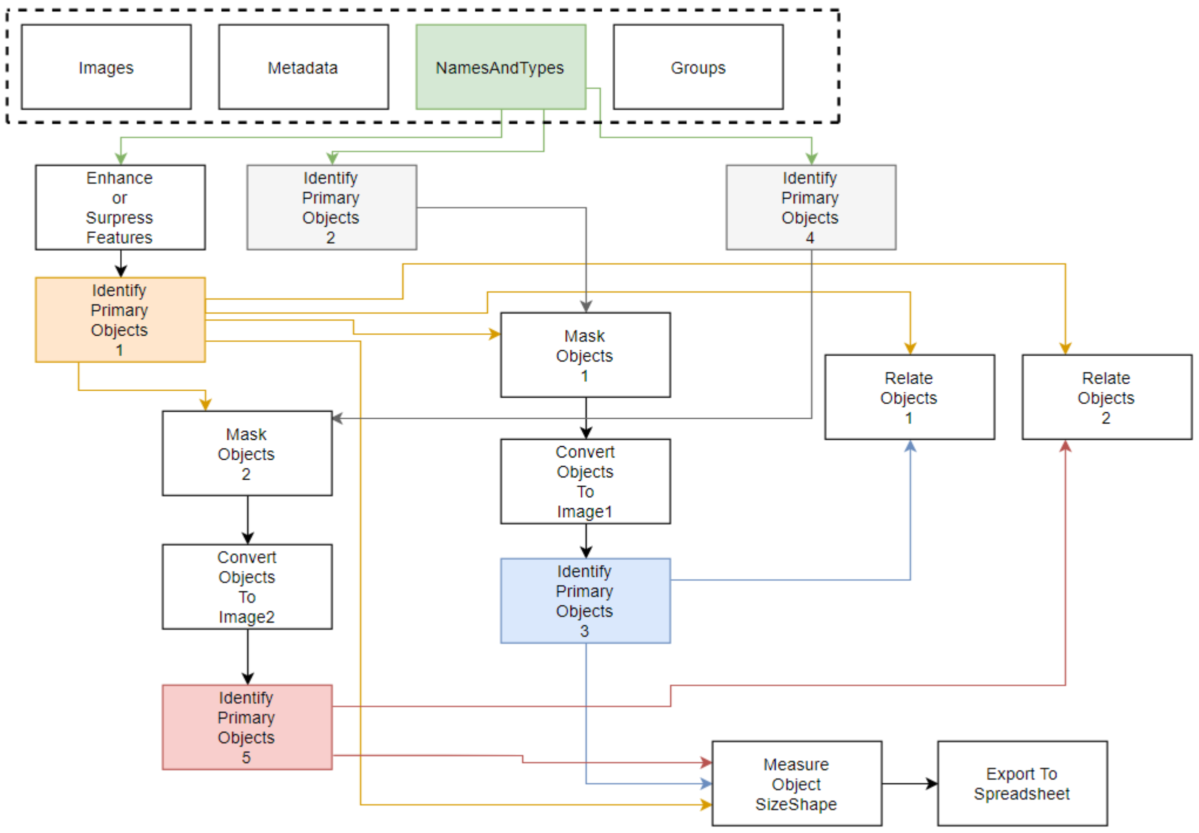
ConvertObjectsToImage
ConvertObjectsToImage converts identified objects into an image. This module allows previously identified objects to be converted into an image according to a selected colormap, which can then be saved with the SaveImages module.

RelateObjects
RelateObjects assigns relationships; all objects (e.g., speckles) within a parent object (e.g., nucleus) become its children.

This module allows child objects to be associated with parent objects. This is useful for counting the number of children associated with each parent, and for calculating mean measurement values for all children that are associated with each parent.

MeasureObjectSizeShape
MeasureObjectSizeShape measures several area and shape features of identified objects. Given an image with identified objects (e.g., nuclei or cells), this module extracts area and shape features of each one.

ExportToSpreadsheet
ExportToSpreadsheet exports measurements into one or more files that can be opened in Excel or other spreadsheet programs. This module will convert the measurements to a comma-, tab-, or other character-delimited text format and save them to the hard drive in one or several files, as requested.



Images
Metadata
NamesAndTypes
Groups
Enhance or Surpress Features
Identify Primary Objects 2
Identify Primary Objects 4
Identify Primary Objects 1
Mask Objects 1
Relate Objects 1
Relate Objects 2
Mask Objects 2
Convert Objects To Image1
Convert Objects To Image2
Identify Primary Objects 3
Identify Primary Objects 5
Measure Object SizeShape
Export To Spreadsheet

Figure 1: Schematic of a CP workflow, where inputs come from and where outputs go to

3.2 Jupyter Notebook

As mentioned earlier, Jupyter Notebook works with two different types of cells: coding cells and markup cells. The markup cells can be ignored as their only purpose is to inform the user. What will follow is a brief summary of all the coding cells and what they do.

Setting up

The first coding cells (Import Packages, Definitions and helper functions and Set Cell Output Directory) are used to set up the rest of the script. Python packages are imported and functions are defined. The most important functions are the following, which are the analyzation steps

- Analyze Plate: This function creates variables in which the information is stored. On top of that, it creates a list of all the wells on the plate.
- Analyze Well: Iterates over all the wells and extracts their location within the plate
- Analyze Field: Loads all the information of what image is linked to the location found in Analyze Well. It loads the image into the CellProfiler pipeline, runs the pipeline and stores the results as a csv, inserting the information into a dataframe and giving the files unique names.

In the third coding cell, a temp-directory is created in which the results will be stored.

This allows the user to run the notebook several times without the storage space on the computer or server running out.

Selecting which plate to analyze
The next block of coding cells all refer to selecting which plate to analyze. In the first coding cell (Connect to Omero@UL), the notebook attempts to establish a connection with the Omero web server. The input consists of a username and password. If that is successful, the connection remains. The next cell (User and group info) gives the user the choice to use data from other users in your Omero group. The following cells (Your screens and plates and Fetch the Plate data) have an interactive widget in which you can select which plate to analyze. The first cell shows you a list of all the screens and plates there are, and the second cell allows the user to pick which plate will be analyzed. A coding cell directly below the previous one can be used to hardcode the plate ID which needs to be selected, if the user wants to ignore the previous steps.

Getting additional data
The next block of coding cells (Fetch Annotations, Format to DF, Load CP Pipeline) will add the final touches to the script before the pipeline is run. The first coding cell (Fetch Annotations) allows the user to select the metadata-file that is associated with the plate from a dropdown widget. In the next cell (Format to DF), this metadata-file, which contains information that could be used as input variables for both the script and the downstream analysis, is converted to a dataframe. Finally, the third coding cell (Load CP Pipeline) has a path to the pipeline file, which will then be loaded into the notebook. The first four modules (Images, Metadata, NamesAndTypes, Groups), which are used in the CellProfiler app to access and group images, are removed. In Jupyter Notebook, the data for these four modules are retrieved from the OMERO database.

Running CP
In the cell “Run CP Pipeline” the plate is analyzed using the pipeline, using the functions explained earlier. After the analysis is complete, the results are concatenated into new CSV files. This coding cell has a few hardcoded lines where the channel names are initialized.

Calculating and Uploading results
After the CellProfiler cell has been executed, the results can be visualized in Notebook. The concatenated CSV files are saved to the Omero server as attachments to the plate which has been analyzed. Finally, the connection to Omero is severed.

3.3 Galaxy
As said before, Galaxy works based on workflows, just like CellProfiler. Making workflows in Galaxy to use for CellProfiler analysis can be achieved with two

different options. Either by importing a CellProfiler pipeline into the history (figure 2), or by putting in individual CellProfiler modules (figure 3):

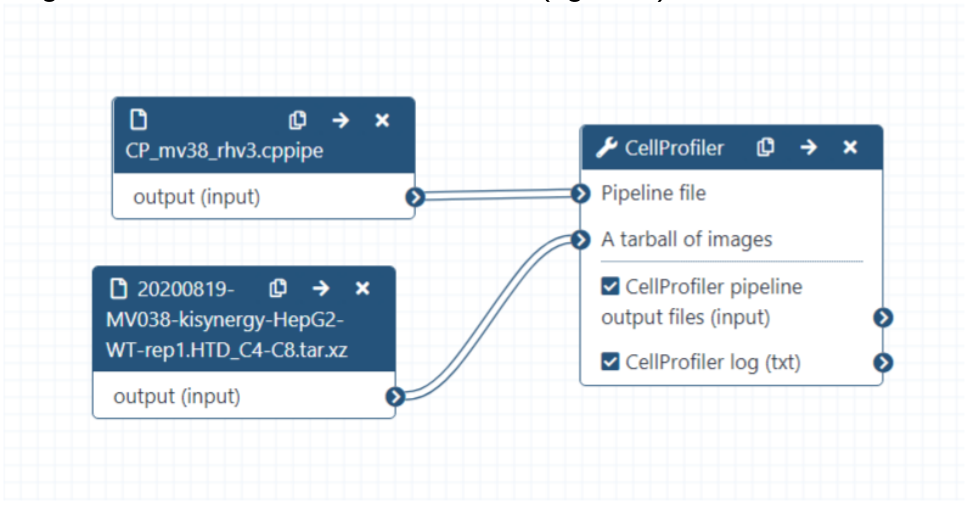


Figure 2: Picture of a Galaxy Workflow with a CellProfiler pipeline imported. When a CellProfiler pipeline is imported this way, the images that are to be analyzed can be uploaded as a tar.xz file. Then, the pipeline-file will be imported, and a Cellprofiler module will be added which takes the image file and the pipeline file as inputs. This job is submitted to the galaxy server and once done, displays the results.

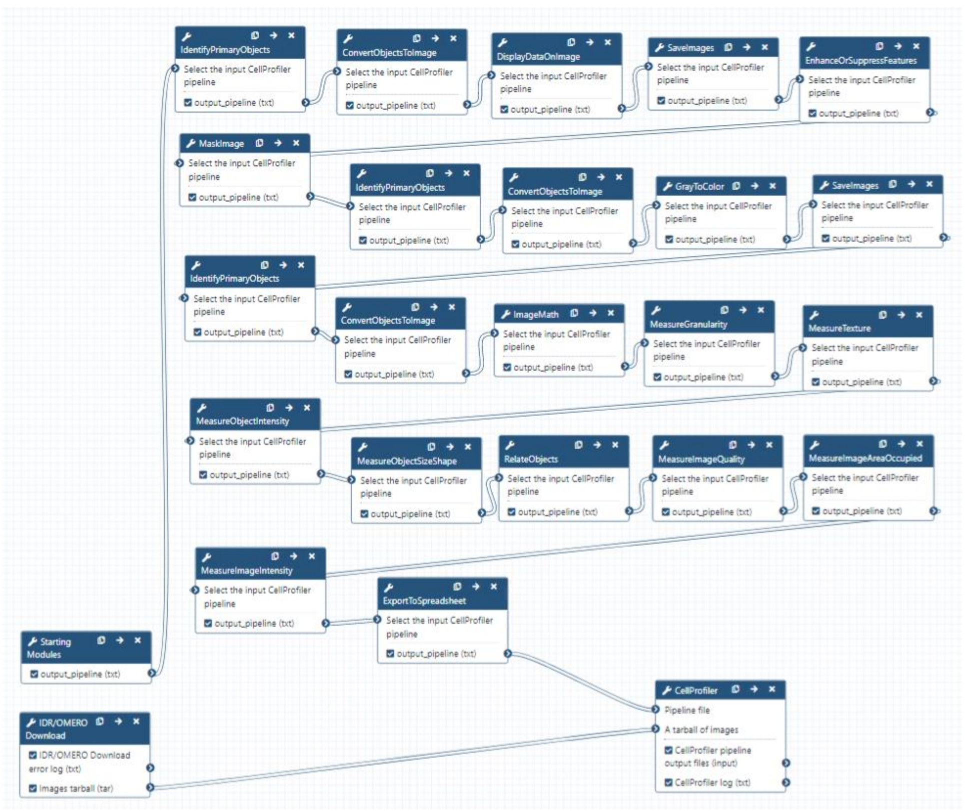


Figure 3: Picture of a Galaxy Workflow where all CellProfiler modules are added individually. This is similar to the way workflows are created in the CellProfiler app. A CellProfiler tutorial on galaxy [9] has been used as template for this.

4. Experiments

To measure the performance of the analysis tools, a test dataset was used. This dataset consists of 216 wells with three different channels and two images taken of each well, for a total of 1296 images. This data has been analyzed using the CellProfiler app, Jupyter Notebook and the Galaxy server. The same workflow has been used for all applications. The results can be seen in Table 1. Additionally, Table 1 features the amount of processor cores available to the application for use, how many cores they used and how much memory was available.

Application	Time	Cores Available	Cores Used	Memory Available
CP app	40m 23s	4	4	16GB
Jupyter Notebook	2h 7m 21s (Analysis only)	4	1	16GB
Galaxy	6h 10m (+4h 57m)	1	1	4GB

Table 1: Performance results of running the test dataset with different analysis tools based on CellProfiler

4.1 CellProfiler

Performance

When running the test dataset in the CellProfiler app, the process took roughly 40 minutes from start to finish. CellProfiler app used all four processor cores available. There was 16GB of RAM available.

Usability

Regarding using CellProfiler app, it is open-source and can be downloaded for Windows and mac. It only accepts local 2D images as input for the pipeline, so it can't run multi-dimensional data. Pipelines and projects are also stored locally. Every pipeline has a specific format and channels are named before the pipeline is executed. The pipeline is reusable, easy to adjust and there is a large amount of documentation present as well.

4.2 Jupyter Notebook

Performance

As can be seen in Table 1, Jupyter Notebook a bit over two hours to run the analysis cell of its own notebook. The time taken to run the entire script couldn’t be measured as some cells required user-input. However, those cells did not add a significant amount of time in comparison to the analysis cell. Jupyter Notebook used only 1 of the available 4 cores, and had 16GB or RAM available for use.

Usability

Similar to CellProfiler, Jupyter Notebook is also downloadable as open-source software as part of Anaconda. It might additionally need a conda environment set up for it in order to make proper use of it. The notebook itself is easy to edit, and using it for other datasets can be as simple as only swapping out the pipeline. Results can be stored online, but this isn't done by default. Additionally, data can be retrieved from sources like OMERO which makes multi-dimensional data analysis possible.

4.3 Galaxy

Performance

Running a Galaxy workflow with the CellProfiler pipeline and the images imported took over 11 hours to give results back. The 6 hours 10 minutes were how long Galaxy said executing the workflow took, but it took an additional 5 hours for the results to show up. Galaxy Servers had one core available, and 4GB of memory.

Usability

Galaxy as a software tool is very easy and accessible to use. The site is easily findable and everything can be public. It is simple to edit workflows and there is a good documentation for every module and their variables.

5. Discussion

5.1 Observations

Interpretation of results

In regards to performance, in our setup the CellProfiler app is significantly better than the other tools tested. The analysis of the test dataset using CellProfiler app was roughly three times faster than using Jupyter Notebook. CellProfiler app has multiprocessing built-in, where the other software tools do not have native multiprocessing.

While it's difficult to make concrete claims about FAIRness, Galaxy seems to be the best option with FAIRness in mind. As long as you have access to FAIR data, it is easy to find workflows in Galaxy. If the workflows are public, they are also accessible to the public. For interoperability, the workflow can be combined with downstream analysis workflows in Galaxy. For reusability, comparable datasets can be analyzed using the same workflow with minor adjustments. The CellProfiler app and Jupyter Notebook neither have an option to share workflows online.

Comparison of Jupyter and CellProfiler

Advantages that Jupyter has over CellProfiler lie mainly in the flexibility that Jupyter offers. Jupyter isn’t only compatible with CellProfiler pipelines. Programs can be used before the CellProfiler pipeline is inserted which alters the input of the pipeline. Jupyter also allows downstream analysis of the images, which CellProfiler does not. It is possible to include parallel processing with Jupyter. Finally, Jupyter allows data retrieval from remote databases, where CellProfiler can only use images that are locally stored on the system.

CellProfiler does have a few advantages compared to Jupyter. The interface of CellProfiler is easier and more intuitive than Jupyter’s interface, and it is easier to give a set of images as input. The pipeline can also be edited rather quickly in CellProfiler, if the intermediate output of a step of the pipeline is not to the user’s liking.

Comparison of Galaxy and CellProfiler

While Galaxy offers the same modules that CellProfiler does, not everything is the same between the two.

To start, it does not offer just CellProfiler modules. Any amount and kind of computation can be performed before the modules relating to the CellProfiler pipeline are used. As said before, calculations for Galaxy are done on the galaxy server, rather than on a local computer. This has advantages and disadvantages. Advantages would be that a pipeline can be executed overnight without the need to have the computer being turned on, and that the duration to execute a history is the same regardless of what kind of hardware the user has. Galaxy directly executing modules that are added to the history rather than waiting for the user to say that they want to run the workflow saves time in the long run. Especially when considering that the jobs being run on the server means there are no hardware strains on the user side while they add more modules to the history.

Disadvantages would be that the server hardware is usually inferior to the hardware used on university computers, and jobs may take several hours. On top of that, the speed with which jobs are executed can depend on how many jobs are running on the server already. A spike in the amount of running jobs will slow down the process. We assume this also happened when we were doing the experiments, where galaxy took almost 15x longer than CellProfiler to produce results.

On top of this, some of the CellProfiler modules that are used in this project are missing variables in Galaxy. What variables are missing are listed below:

- Mask misses variables → No handling of objects that are partially masked. No numbering of resulting objects
- IdentifyPrimaryObjects missing variables based on the adaptive/otsu parameters → Automatically calculate size of smoothing filter for declumping, Automatically calculate minimum allowed distance between local maxima, Speed up by using lower-resolution image to find local maxima, Display accepted local maxima, Fill holes in identified objects or Size of smoothing filter
- MeasureObjectSizeShape missing variable → Can't calculate advanced features
- ExporttoSpreadsheet missing variables → Can't Add image file and folder names to an object data file, Select the measurements to export or export all measurement types

Output from different tools
After testing on the dataset, the output consists of nine CSV files. While all three programs had the exact same pipeline and the exact same set of images to work on, the output between them differed slightly. Some applications gave more variables in the output. In Jupyter Notebook, images are read and analyzed one-by-one, and the results are saved as CSV files. However, to make the files unique before they get concatenated, additional columns are added (Image, Well, WellName, Field) to identify the files before they are concatenated. This automatically happens in CellProfiler app and Galaxy

In Galaxy, the RelateObjects module didn't work as intended, and the two CSV files related to that module (AnV_masked_primaryID and PI_masked_primaryID) don't have any content.

5.2 Improvements
After the initial results, there was a large disparity between the three software tools, while they used the same test dataset and the same workflow. It should theoretically be possible to have the tools run as fast as CellProfiler did, so we've looked into

ways of improving the performance of the three software tools. The CellProfiler application itself was unable to be improved, as that would imply editing the source code of the program itself, while the focus was on adjusting the scripts. For Galaxy, we've tried uploading channels independently and getting uploads from a different OMERO server than was listed in the Omero-upload module. However, both didn't amount to any improvements.

Jupyter
Jupyter Notebook ran on the same hardware as the CellProfiler app did, but took three times longer to complete. Considering it only used one core of the four that were available, combined with the fact that it is the easiest to improve as it consists of python scripts, it was the most suited to make improvement to of all software tools. What follows will be an overview of methods that have been used to try and improve the performance of Jupyter.

- **Using the multiprocessing package**:

As can be seen in Table 1, Jupyter only used one of the four cores available to it. With the multiprocessing package, workers can be defined which will each use an individual core. This way, the analysis step of Jupyter (Analyze Plate/Well/Field) could be parallelized. In Analyze Well, the program iterates over all well locations. The parallel workers would each have a range defined from within they could iterate over the wells, so they would not overlap. However, when doing this, the code threw an error for a function within CellProfiler, warning about calling the function multiple times while iterating over the same variable. This could not be fixed, and running the code afterwards without the parallel workers also kept that error.

- **Using Dask Bag**:

The Dask package allows datasets to be split into multiple bags. Functions can then be performed on the bags in parallel. Dask can only Bag certain kinds of data, and images are not one of them.

- **Changing from Notebook to Python script**:

Rather than working within the boundaries set by Jupyter Notebook, making the notebook a Python Script and running the script multiple times with arguments separate from each other in parallel can work. It will bypass the problem caused by the multiprocessing package too. What would need to be kept in mind is that the output files all have different names before being concatenated, as it would cause problems if similar-named files are stored in the same folder. This possibility has not been fully worked out yet.

5.3 Conclusion

In summary, our findings pointed out that each software tool has its own advantages and disadvantages. CellProfiler is the best analysis tool performance-wise, but it can't be improved, unlike Jupyter Notebook. Although we haven't been able to finish adding parallelism to Jupyter Notebook, we expect parallelized Jupyter Notebook scripts to be at least as fast as CellProfiler app. In regards of usability and FAIR, Galaxy is the best analysis tool ad it's designed with sharing in mind. However, Jupyter Notebook has the most promise of the three software tools tested. The fact that it can be combined with any other script gives it a potential that can’t be reached with the CellProfiler app. While CellProfiler app is open-source, it can't be adjusted as easy as a Jupyter Notebook cell can be altered. Galaxy has a similar problem to the CellProfiler app, where new modules are dependent on Galaxy Toolsheds, which are maintained by a different party. Notebook scripts can be added to automatically save results online as well.

For future work we therefore recommend improving Jupyter Notebook and adding parallelism to the notebook, or changing it to a Python Script and working from there to improve the performance.

References

1. Li, S., & Xia, M. (2019). Review of high-content screening applications in toxicology. Archives of toxicology, 93(12), 3387-3396. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7011178/
2. Boutros, M., Heigwer, F., & Laufer, C. (2015). Microscopy-based high-content screening. Cell, 163(6), 1314-1325. https://www.sciencedirect.com/science/article/pii/S0092867415014877
3. Giuliano, K. A., DeBiasio, R. L., Dunlay, R. T., Gough, A., Volosky, J. M., Zock, J., ... & Taylor, D. L. (1997). High-content screening: a new approach to easing key bottlenecks in the drug discovery process. Journal of Biomolecular Screening, 2(4), 249-259. https://journals.sagepub.com/doi/pdf/10.1177/108705719700200410
4. Wilkinson, M., Dumontier, M., Aalbersberg, I. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* **3,** 160018 (2016). https://doi.org/10.1038/sdata.2016.18
5. McQuin C, Goodman A, Chernyshev V, Kamentsky L, Cimini BA, Karhohs KW, Doan M, Ding L, Rafelski SM, Thirstrup D, Wiegraebe W, Singh S, Becker T, Caicedo JC, Carpenter AE (2018). CellProfiler 3.0: Next-generation image processing for biology. PLoS Biol. 16(7):e2005970 / doi. PMID: 29969450
6. Kluyver, Thomas, Ragan-Kelley, Benjamin, Pérez, Fernando, Granger, Brian, Bussonnier, Matthias, Frederic, Jonathan, Kelley, Kyle, Hamrick, Jessica, Grout, Jason, Corlay, Sylvain, Ivanov, Paul, Avila, Damián, Abdalla, Safia, Willing, Carol and Jupyter development team, (2016) Jupyter Notebooks – a publishing format for reproducible computational workflows. Loizides, Fernando and Scmidt, Birgit (eds.) In *Positioning and Power in Academic Publishing: Players, Agents and Agendas.* IOS Press. pp. 87-90 . (doi:10.3233/978-1-61499-649-1-87).
7. Enis Afgan, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update, *Nucleic Acids Research*, Volume 46, Issue W1, 2 July 2018, Pages W537–W544, doi:10.1093/nar/gky379
8. Batut et al., 2018 Community-Driven Data Analysis Training for Biology Cell Systems 10.1016/j.cels.2018.05.012
9. Beatriz Serrano-Solano, Jean-Karim Hériché, 2021 Nucleoli segmentation and feature extraction using CellProfiler (Galaxy Training Materials). https://training.galaxyproject.org/training-material/topics/imaging/tutorials/tutorial-CP/tutorial.html Online; accessed Mon Jul 19 2021