

# PyPlaque: an Open-source Python Package for Phenotypic Analysis of Virus Plaque Assays

Trina De<sup>1,2</sup>, Vardan Andriasyan<sup>3,4,✉</sup>, and Artur Yakimovich<sup>1,2,3,5,✉</sup>

<sup>1</sup>Center for Advanced Systems Understanding (CASUS), Görlitz, Germany

<sup>2</sup>Helmholtz-Zentrum Dresden-Rossendorf e. V. (HZDR), Dresden, Germany

<sup>3</sup>Artificial Intelligence for Life Sciences CIC, London, United Kingdom

<sup>4</sup>Department of Molecular Life Sciences, University of Zürich, Zürich 8057, Switzerland

<sup>5</sup>Institute of Computer Science, University of Wrocław, Wrocław, Poland

Virological plaque assays are the primary method for quantifying infectious particles in a suspension, achieved by incubating a serial dilution of the virus with a monolayer of indicator cells. Existing software tools for quantification of plaque assay images lack modularity, show measurements disagreements or are closed-source - a common hurdle in BioImage analysis. We introduce PyPlaque, an open-source Python package focusing on flexibility and modularity rather than a bulky graphic user interface. Unlike previous methods, an abstracted architecture using object-oriented programming allows accommodation of various experimental containers and specimen carriers as data structures while focusing on phenotype-specific information. Aligned with the logical flow of experimental design and desired quantifications, it delivers insights at multiple granularity levels, facilitating detailed analysis. We demonstrate how this approach allows to focus on alleviating the disagreement in measurements. Furthermore, similar design is generalisable to diverse datasets in various biological contexts that fit our structural paradigm.

plaque assays | viral quantification | morphometry | plaque quantification | data science

Correspondence: [a.yakimovich@hzdr.de](mailto:a.yakimovich@hzdr.de), [vardan.andriasyan@uzh.ch](mailto:vardan.andriasyan@uzh.ch)

## Introduction

The ongoing SARS-CoV-2 pandemic proved that our ability to detect and quantify viruses is vital for stepping up our efforts to both diagnose and research emerging pathogens in a timely fashion. Yet, while a number of novel assays have been developed or improved (1–3), the virological plaque assay (4, 5) remains the only existing way to measure infectious virus isolates and quantify viral spread in tissue culture (6–8). Unlike other assays aiming to quantify a proxy of infection like viral proteins (antigen), gene products (PCR or rtPCR) or disease manifestations (chest X-Ray), the virological plaque assay measures the ability of the isolated or cultured infectious virus to infect cells and produce progeny infectious virions infecting the cells thereafter - a property remaining at the very core of virus virulence.

While slight variations of the plaque assay exist depending on the pathogen, tissue culture, format, or recording method, at its core, every plaque assay requires two main components, a highly diluted (low multiplicity of infection) inoculum and cultured indicator cells (4, 5). An infection assay set up in such a way will result in only a few infected

cells, typically, within a confluent monolayer of indicator cells (4). Upon incubation for the duration of at least one full replication cycle of the pathogen, the initially infected cells will release their infectious progeny. These will, in turn, infect the neighbouring indicator cells leading to the emergence of the clonal patterns of spread. At the right dilution of the inoculum, these patterns will not overlap, allowing the observer to simply count them, thus quantifying the number of infectious particles in the inoculum measured as the number of plaque-forming units (PFU) per unit of volume. As the pathogen spreading patterns may often be intricate, measurements are often performed manually by a trained specialist.

Beyond the quantification of infectious particles, however, virological plaque assay represents a model system allowing to study virus spread phenotypes (9–11). Specifically, the size, shape, and consistency of plaques, as well as the patterns and motion trajectories of the infected cells constituting the virological plaque may reveal a great deal of information about the underlying biological mechanisms leading to the formation of such phenotypes (10–12). Therefore, harnessing this information through computational analysis of biomedical images is an invaluable tool for analysing known and emerging viruses.

To address this, apart from custom project-related or closed-source code, several approaches have been proposed (9, 11, 13–18). However, while useful for a broad range of applications the vast majority of these tools remain highly specialised. For example, Plaque 2.0 (9, 11), Infection Counter (14), and SCFQ (16) are aimed exclusively at fluorescence microscopy. At the same time, ViralPlaque (13) plugin for ImageJ/Fiji (19) software is highly tuned for well-separated crystal-violet-stained plaques. Additionally, Plaque 2.0 (9) (11), and SCFQ (16) specialise in high-content images. Plaque Size Tool (17), in turn, while more versatile from the image source perspective requires only well-separated plaques. Other tools, like Viridot (15) are dedicated to one specific virus only. Finally, all tools existing thus far, are highly reliant on user input via the graphic user interface and don't support modern open-source data analysis languages like Python.

Such a highly scattered landscape of tools indicates that despite the seeming similarity, all these tools were developed for their own purpose. Being a fundamental assay in Infection Biology, plaque assay is performed slightly differently depending on the pathogen, imaging techniques, indicator cell types, throughput requirements, and reagent availability. Furthermore, should the purpose of the assay lay beyond the quantification of PFU, more often than not the plaque assay would be specialised to the final requirements of the study. Evidently, should a unified solution exist in such a case, it would require a great degree of flexibility while taking care of the complexity of plaque assay.

To address this, here we propose PyPlaque (20) - an open-source Python package allowing to simplify plaque assay analysis. Unlike the graphic-user-interface-based tools mentioned above, PyPlaque takes a radically different approach. Instead of a no-code solution, we provide a convenient library that lowers the amount of code required to analyse plaque assay data while maintaining ultimate analysis flexibility. We achieve this by providing Python data structures (akin to Pandas (21) for tabular data). Using these data structures, PyPlaque allows users to account for experiments, dishes or plates and plaque phenotypes. Utilising Python features like IPython/Jupyter (22, 23) interactive notebooks, such scripts or notebooks could be rapidly created and shared. PyPlaque is aimed at programming-savvy biologists and significantly simplifies the development of highly-optimised plaque assay quantification scripts. Leveraging the vast data analysis package ecosystem of Python, notebooks can be adapted by fellow researchers, who may need a set of specialised modifications. We demonstrate the versatility of our approach on several examples, including different variations of plaque staining, and imaging. Furthermore, our examples employ diverse viruses. Lastly, PyPlaque is open-source and requires only open-source software to run.

## Results

**PyPlaque Design Principles.** To develop a highly customisable computational tool for virological plaque analysis we followed several design choices. Firstly, we aimed for the tool to be based upon modern and widespread open-source programming language in order to fit seamlessly into the tool-set of Data Science. This led to Python being the language of choice. Secondly, we aimed to use object-oriented programming (OOP) to create logic and structures allowing us to seamlessly capture the experiment and specimen details. We set out to create structures as seamless as Pandas dataframes (21) to connect important aspects of data acquisition through easy-to-follow abstractions. Finally, we aimed to be minimalistic from the perspective of package dependencies and from the perspective of re-using already existing tools in Python - for example, the said dataframes.

The first key OOP abstractions we have singled out are the *Experiment*, the *Specimen* and the *Phenotype* (fig. 1A). In this sense, the *Experiment* can be viewed as a logical

container for several Petri dishes or multi-titre plates united by a similar experimental purpose. For example, an *Experiment* could be a virus titre determination performed in three biological replicas, which would combine three dishes into one OOP object. A *Specimen* in this example would constitute an individual dish or biological replicate. Within this dish, one may expect to find several plaques - i.e. *Phenotypes* in our abstraction. Combined, such a hierarchical structure is highly versatile and can cover a great variety of biological data. For example, irrespective of whether the experiment aims to perform the plaque assay by the means of an end-point Crystal Violet staining or use fluorescence microscopy, data can make use of the structure described above (fig. 1B).

The final key OOP abstraction we propose is the *View* classes. In our design, the *View* modules aim to contain any kind of OOP class aimed for analysis or visualisation obtained from the three classes mentioned above (see fig. 1B). The choice of the class to be used in order to construct a specific *View* defines the granularity of the results. For example, if the view is constructed for the granularity of the *Phenotype*, it would contain measurements of an individual plaque. Whereas if a *View* is constructed for the entire *Experiment* it would contain average measurements of plaques in this experiment. Once the classes are instantiated, this information is stored as nested dictionaries.

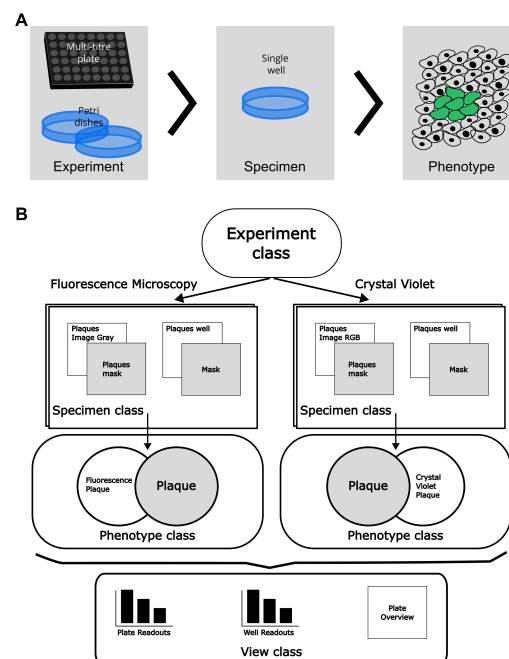


**Fig. 1. Object-oriented programming design of the PyPlaque package.** (A) Schematic depiction of nested classes (grey rectangles with titles) representing the structure of PyPlaque abstractions accompanied by pictorial examples. (B) Flow diagram of the classes available to represent the experimental data. An example demonstrates how the Fluorescence Microscopy experiment or Crystal Violet experiment could make use of the same structures: *Experiment* > *Specimen* > *Phenotype*. A *View* module gives access to classes and methods for analysing different cross-sections of the experimental data for e.g., the whole plate, the well or the plaque.

**Phenotypic Measurements Validation using Synthetic Data.** Once plaque assay images are loaded into the respective data structures measurements can be performed using a respective *View*. Measurements can include count or morphological properties of the plaque phenotypes such as area or perimeter. However, different software platforms may perform such measurements differently, leading to a lack of comparability. To date the most popular platforms for biomedical image analysis include ImageJ/Fiji (19), Matlab (24) and Python (using packages like scikit-image (25)). Thus far, existing specialised packages for plaque quantification have been based on these platforms.

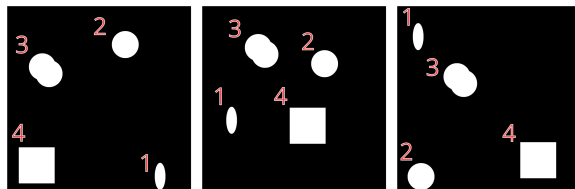


**Fig. 2. Analysis of Synthetic Images.** Synthetic images generated with randomly located shapes of an ellipse, a circle, overlapping circles and a square with IDs, 1, 2, 3 and 4 respectively. 5 such images were generated and used for analysis and comparison between computation software (table 1, table 2) of which 3 are shown above.

To understand how such measurements could be compared between these major platforms we have constructed a set of synthetic binary images depicting simple geometric shapes: an ellipse, a circle, two overlapping circles and a square (fig. 2), assigned to the numbers 1, 2, 3 and 4 respectively. These shapes were chosen to represent both regular and irregular shapes in the simplest way possible for ease of interpretation. To ensure comparability amongst the measurement systems, the shape parameters were identical while their location was chosen at random. fig. 2 shows 3 of 5 images used in the comparison.

Next, we performed two basic measurements of each object in the synthetic images using ImageJ/Fiji (19), Matlab (24) and our Python-based PyPlaque. Specifically, we have quantified the area and perimeter of these objects, as both of these measurements can be used to describe the extent of the virological plaques. In the case of PyPlaque, we used two different methods to estimate the area and perimeter. The first employed a built-in measurement using the scikit-image package (25). Second, was the method based on Pick's theorem (see Page 9, **Pick's Area and Perimeter**), which we developed specifically for PyPlaque.

Area measurement results presented in table 1 suggest that all three platforms performed equally slightly underestimating the area compared to GT. Notably, while the PyPlaque area measurement using scikit-image (25) functionality performed equally to Matlab (24) and ImageJ/Fiji (19), the area measurement using Pick's theorem led to even further underestimation.

Surprisingly, the perimeter measurement results presented in

**Table 1. Object Area in Synthetic Images.** Estimated values are averaged over 5 images where the objects were seeded in random locations. GT stands for ground truth. The performance closest to GT is marked in bold.

| Software                     | Area             |                  | ID                  |                   |
|------------------------------|------------------|------------------|---------------------|-------------------|
|                              | ellipse          | circle           | overlapping circles | square            |
| GT                           | 28274.334        | 70685.835        | 113733.351          | 160000.000        |
| PyPlaque (Pick's)            | 27588.000        | 69812.000        | 100779.000          | 158403.000        |
| Matlab (24)                  | <b>28233.000</b> | <b>70661.000</b> | <b>101806.000</b>   | <b>160000.000</b> |
| ImageJ/Fiji (19)             | <b>28233.000</b> | <b>70661.000</b> | <b>101806.000</b>   | <b>160000.000</b> |
| PyPlaque (scikit-image) (25) | <b>28233.000</b> | <b>70661.000</b> | <b>101806.000</b>   | <b>160000.000</b> |

**Table 2. Object Perimeter in Synthetic Images.** Estimated values are averaged over 5 images where the objects were seeded in random locations. GT stands for ground truth. The performance closest to GT is marked in bold. Second-best is underlined.

| Software                     | Perimeter      |                | ID                  |                 |
|------------------------------|----------------|----------------|---------------------|-----------------|
|                              | ellipse        | circle         | overlapping circles | square          |
| GT                           | 690.384        | 942.478        | 1256.637            | 1600.000        |
| PyPlaque (Pick's)            | 704.000        | 988.000        | <b>1238.828</b>     | <u>1604.000</u> |
| Matlab (24)                  | <b>686.456</b> | <b>939.000</b> | 1155.220            | 1563.716        |
| ImageJ/Fiji (19)             | 722.943        | 991.561        | 1218.923            | <b>1597.657</b> |
| PyPlaque (scikit-image) (25) | 721.872        | 990.489        | <u>1219.024</u>     | <u>1596.000</u> |

table 2 demonstrated an entirely different picture. In the case of *ellipse* and *circle* shapes, Matlab-based measurements were the closest to the GT with PyPlaque measurement based on Pick's theorem showing the second-best results. In the case of the *square* shape ImageJ/Fiji demonstrated the best performance, with PyPlaque measurement based on scikit-image showing the second-best results. Finally, in the case of *overlapping circles*, PyPlaque measurement based on Pick's theorem provided the measurement closest to the GT, with scikit-image performing second best. Crucially, Matlab, ImageJ/Fiji and scikit-image measurements underestimated the perimeter severely on the overlapping circles.

Together our results suggest a surprising incomparability of simple shape measurements across modern bioimage analysis platforms. However, due to the flexibility coming with using a Python package, the user is free to choose the measurement approach depending on the needs of the project. It is worth noting that we have additionally performed the area and perimeter measurements with another popular Python library OpenCV (26). The results of these measurements were largely identical to scikit-image and therefore omitted here.

**Crystal Violet Plaque Assay Measurements at varying Granularity using PyPlaque.** To demonstrate how the structure depicted in fig. 1 can be used to store and analyse data from a conventional plaque assay we performed an analysis of a conventional assay plaque of Vaccinia Virus (VACV) stained with Crystal Violet. The images of the assay plates were taken from an open dataset available online (27). This dataset consists of mobile photographs of the plaque assay plates accompanied by binary masks for wells and individual plaques. Each image of the dataset contains objects of increasing granularity containing quantitative

information about the assay.

Specifically, fig. 3A, shows a 6-well plate (left) and a masked version of the full 6-well plate (right). Such data can be handled by the *PlateImage* class of PyPlaque. fig. 3B goes further in granularity showing a single well from the plate (left) and the corresponding masked version of the well (centre). Such data is handled by our classes *PlaquesWell*. The rightmost column of fig. 3B inset is a single viral plaque from the well. We have the functionality using our class *PlaquesMask* to extract readouts such as the average size of all the plaques in the well, the total count of plaques and the area of an individual plaque as also shown in the figure. The usage of the classes mentioned above are further explained in the [crystal violet plaque](#) notebook found in the [GitHub](#) page of the project. The granularity provided by PyPlaque readouts could be beneficial in comparing plaques across wells treated with virus inoculum at different dilutions. Furthermore, it may allow comparing measurements across plates and different experiments.

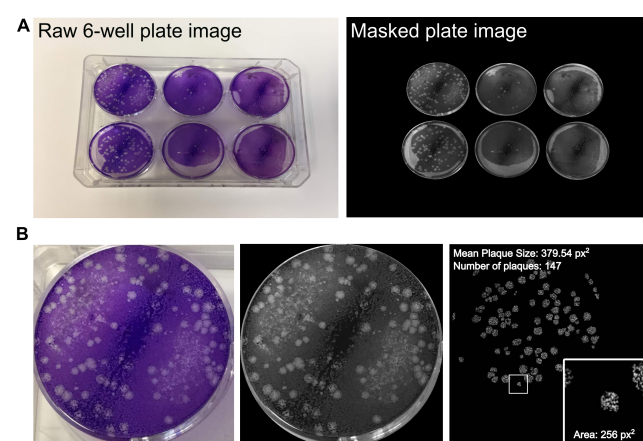


**Fig. 3. Granularity in the Detection of Objects in Crystal Violet Plaque Assay Images of VACV.** Objects with increasing granularity are stored and analysed by PyPlaque. (A) Shows a 6-well plate (left) and a masked version of the full 6-well plate (right). (B) Shows a single well from the plate (left), corresponding masked version of the well (centre) and a single viral plaque from the well (right-inset) with average size of the plaques in the well, the number of plaques (top-left) and area of an individual plaque (inset bottom-right).

**Scalable Plaque Analysis using PyPlaque in High-content Fluorescence Microscopy.** Beyond measuring the titre in the virus inoculum, plaque assay can be employed to measure virus spread under conditions of perturbation. Techniques like high-content fluorescence microscopy allow scaling up such efforts (11, 28). To illustrate how PyPlaque can be employed in quantifying such experiments we have performed an analysis of a control 384-well plate half of which was infected with GFP-transgenic human coronavirus OC43 (CoV-GFP). This plate was obtained from a published dataset (29). The control well plate we used is imaged for two fluorescence channels: CoV-GFP and Nuclei (Hoechst 33342 dye).

Comparably to the PyPlaque workflow constructed for the Crystal Violet plaques, information in the high-content fluorescence microscopy exists at several levels of granularity fig. 4A. In contrast to the Crystal Violet plates, however, individual images in fluorescent microscopy are obtained at the level of wells. That means that the lower granularity plate overview is not available and must be constructed dynamically. This can be achieved using the *View* class. Results in the fig. 4A show that, as expected, the left-hand half of the control plate has visible GFP-positive cells, some of which are clustered into fluorescent CoV plaques. Notably, some wells on the right-hand side contain sample preparation artefacts (e.g. dust and fibres) (30), which may contribute to the background detection due to autofluorescence (fig. 4A, zoomed insets).

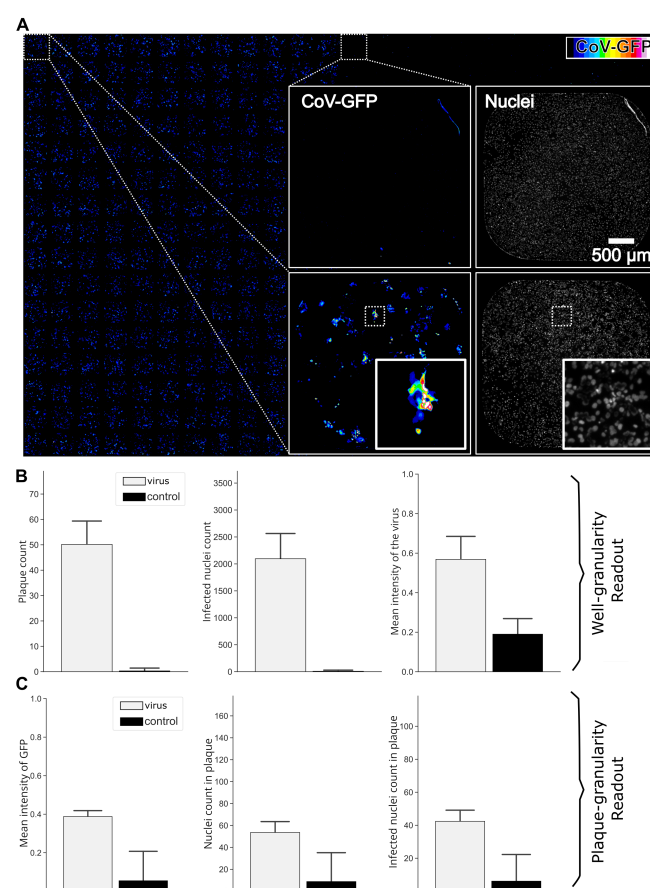


**Fig. 4. Pyplaque measurements at image and object level.** PyPlaque (20) includes methods to provide readouts at an image and object level. (A) Example 384-well Fluorescence microscopy imaged plate. Insets show examples of two wells from two vertical halves of the entire 384-well plate. Left column of the insets show the CoV-GFP channel and right column shows the Nuclei (Hoechst dye) channel. Insets within the insets show a section of a single well for the CoV-GFP and Nuclei channel on the left and right respectively. (B) Plot of readout values at the well level from the software for the plate shown in panel (A). (C) Plot of readout values at the object level for the plate, averaged first over wells. For each plot, we have the infected wells (left) and control wells (right) indicated as virus and control in the legend. Such readouts could be generated by the classes and methods from *View* (see fig. 1)

Noteworthy, fluorescence microscopy does not usually come with segmentation masks. While segmentation masks can be

obtained through advanced deep learning models, in some cases rule-based algorithms like Otsu (31) are sufficient. Both cases are supported by PyPlaque. However, while rule-based thresholding and connectivity handling is built-in via `get_all_plaque_regions` method, the former requires an external model.

Results of quantifications using built-in methods can be seen in fig. 4B and C. Specifically, fig. 4B demonstrates differences between the infected and uninfected halves of the plate (denoted as virus and control respectively) at the granularity of a well. Measurements like plaque count, infected nuclei count and the mean intensity of the virus signal can be obtained using the built-in methods of the PyPlaque package. fig. 4C demonstrate that such analysis is also possible at the level of individual plaque. Measurements such as nuclei count in the plaque, and infected nuclei count in the plaque, as well as mean GFP intensity can be obtained. The latter, in turn, could facilitate the separation of real plaque from a sample preparation artefact by simply filtering out the objects with very low fluorescence during the post-processing of the analysis results. For full list of measurements, please see Page 8, **Classes for well and plaque level readouts**.

Importantly, workflows for the data source as different as mobile photography and high-content fluorescence microscopy can be flexibly constructed using the OOP structure we propose for PyPlaque (20). In the spirit of Python Data Science, these workflows can be implemented as Jupyter (23) notebooks. An example of such an **analysis notebook** can be found in the [GitHub](#) page of the project.

## Discussion

Quantification of phenotypes in the virological assays plays an important role in Virology research and our search for novel antivirals (32, 33). In the case of the virological plaque assay, several quantification tools have been proposed in the past (9, 11, 13–18). However, the overall focus on the graphical user interface (GUI) makes these tools unflexible and hard to maintain. In many cases, the GUI part of the software is as complex as all the other functionalities (11). Furthermore, GUI shifts the focus of the package, forcing a particular workflow design.

The PyPlaque (20) package developed in this work takes a different approach to the quantification of virological assays. Rather than providing a GUI-based solution, we propose a Python package rooted in the Data Science technological stack. Instead of forcing the user to follow a particular workflow, we provide an abstract OOP container for experimental data that can be integrated into Pythonic workflow constructors like Jupyter (22, 23). We argue that by making the data constructs available in the Jupyter environment, our approach is more scalable and flexible due its to compatibility with modern Data Science tools like Pandas (21). In this sense, once the data is ingested, running

the analysis is as simple as putting an action into a for-loop - simplicity accessible to users from various backgrounds.

Incidentally, this approach allowed us to focus on the actual measurements providing users with the choice of methods. In a series of comparisons we demonstrated the lack of agreement in a seemingly trivial task of perimeter measurement in the leading BioImage analysis platforms including Matlab (24), ImageJ/Fiji (19) and Python scikit-image (25). Notably, the go-to Python approach showed the greatest disagreement. To remedy this, we propose an alternative perimeter measurement approach alleviating the disagreement in certain cases.

Furthermore, we argue that the approach proposed here can be generalised to other biological assays yielding a new paradigm. Similar structures of classes at varying granularities from *Experiment* to *Phenotype* can be constructed for many kinds of experiments. In the particular case of experiments conducted with sample carriers shown here, PyPlaque can be used for assays other than plaque assay already. We hope to receive the open-source community contributions to further expand this scope.

## ACKNOWLEDGEMENTS

This work was partially funded by the Center for Advanced Systems Understanding (CASUS) which is financed by Germany's Federal Ministry of Education and Research (BMBF) and by the Saxon Ministry for Science, Culture, and Tourism (SMWK) with tax funds on the basis of the budget approved by the Saxon State Parliament. We thank Urs Greber (University of Zurich, Switzerland) for access to raw data of (29). The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany and by Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus in the programme Center of Excellence for AI-research "Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig", project identification number: ScaDS.AI.

## Methods

**PyPlaque Installation.** The release version of the PyPlaque package can be installed using PyPi package manager by invoking the following command in the command line interface:

- `pip install pyplaque`

The developer version can be installed directly from the *master* and *dev* branches of the [GitHub](#) repository.

**Multi-granularity Level Design.** As mentioned above the design of the PyPlaque classes follows a multi-level granularity approach. This means that the structures of the package allow to group data and metadata at different levels, preserving semantic relationships. These structures are detailed in the subsequent sections.

**Experiment Granularity Level Classes.** At the highest level of granularity, we defined classes *CrystalViolet* and *FluorescenceMicroscopy* which are designed to contain metadata of multiple instances of a multi-titre plate of Crystal Violet Plaque and Fluorescence Plaque respectively. To aid in having a better overview of all of the multi-titre plates in the experiment, both of these classes contain methods *get\_individual\_plates*, *get\_number\_of\_plates* and *read\_from\_path* that identify the individual plates, their total count and read each individual well image respectively.

**Single Plate Granularity Level Classes.** At the next level of granularity are the classes aimed at containing data and metadata for one experimental container, for example corresponding to a single multi-titre plate. The *PlateImage* class accepts an image and corresponding binary masks of individual wells of a single full multi-titre plate. Using class properties such as the number of rows, number of columns and a flag indicating whether or not the plate was imaged inverted or not, given by the user, methods of the class return the following:

- *get\_wells* - returns a list of the images of individual wells of the plate stored as binary Numpy arrays.
- *get\_well\_positions* - returns a dictionary where individual wells of the plate are the units. For each individual well, the image, mask, masked image, bounding box x-y values, row and column number estimated from the well positions are stored with the rows and columns numbered starting from 0.
- *plot\_well\_positions* - returns a plot with bounding boxes drawn around individual wells of the plate (inferred from the plate mask), with rows and columns numbered starting from 0.

**Single Well Granularity Level Classes.** The class *PlaquesWell* is aimed at containing an individual well of a multi-titre plate. It expects information about the row and column numbers, the well image and mask. It has the

method *get\_image* which returns a masked image of the individual well.

The *PlaquesMask* is a feature-enriched class designed with the aim of holding a single binary mask containing multiple plaque instances from an individual well. Using the binary mask of plaques and a name to identify this plaque mask (usually derived from the plate name and the well position that the mask corresponds to), methods of the class return the following:

- *get\_plaques* - returns a list of individual plaques stored as binary Numpy arrays.
- *get\_measure* - returns measures based on the individual plaques in a well as a dictionary. This is more appropriate for cumulative measurements as compared to the *measure* method under class *Plaque* (see Page 6) that gives granular measurements based on each plaque.
- *plot\_centroid* - plots a dotted ring around all the plaques that are found in a well. This ring is centred at the centroid (see Page 8, *get\_centroid*) of all the centres of individual plaques found.

The classes *PlaquesImageRGB* and *PlaquesImageGray* are child classes of *PlaquesMask*. These two classes are designed to hold grayscale image data and RGB image data containing multiple plaque instances respectively with a respective binary mask. At first glance, these classes could be thought of as the same with a flag differentiating the channel in the image they can hold. However they are primarily different because of the methods that they give the user access to while opting for one or the other. Similar to the class *PlaquesMask*, these two classes too use a name (usually derived from the plate name and the well position that the mask corresponds to) to identify the plaque image and mask object it contains. *PlaquesImageGray* additionally contains accepts two parameters *threshold* and *sigma* which are further used to apply Gaussian blur and binarise the grayscale image.

**Single Phenotype Granularity Level Classes.** The class *Plaque* is designed to contain a single virological plaque phenotype as an object. It encapsulates the properties related to a specific plaque, including its mask, centroid coordinates and bounding box, and offers methods for the calculation of the area, eccentricity and roundness of this plaque object. Methods of the class return the following:

- *measure* - returns the bounding box area (in pixels) of a plaque object, as well as an approximation of its actual area based on the proportion of white pixels in the mask.
- *eccentricity* - calculates and returns the eccentricity (34) of an individual plaque object. The eccentricity is determined by fitting an ellipse to the plaque's boundary contour and using the formula,

$$e = \left(1 - \frac{b^2}{a^2}\right)^{\frac{1}{2}}, \quad (1)$$

where  $b$  represents the length of the semi-minor axis, and  $a$  represents the length of the semi-major axis.

- **roundness** - returns the roundness (35) of an individual plaque object. The roundness is calculated using the formula,

$$r = \frac{4\pi \cdot \text{area}}{\text{perimeter}^2}, \quad (2)$$

where  $\text{area}$  is estimated as  $\pi R^2$ , and  $\text{perimeter}$  is calculated as  $2\pi R$ ,  $R$  being the radius estimated from the centre to the top-right corner of the bounding box. This method can alternatively use Pick's (see Page 9, **Pick's Area and Perimeter**) area and perimeter estimates. This value is 1 in the case of a perfect circle.

The classes **CrystalVioletPlaque** and **FluorescencePlaque** are child classes of the class **Plaque**. They allow for the storage of data and metadata of crystal violet or fluorescent dye stained plaques imaged using mobile photography or bright-field microscopy and fluorescence microscopy respectively. Since the masks are binary in both cases, both of these classes are able to use the methods such as **measure**, **eccentricity**, and **roundness** from the parent class.

**Analysis of Nuclei.** The analysis of nuclei begins with the acceptance of a well image corresponding to the nuclei channel stained with fluorescent dye and imaged via fluorescence microscopy. In case a pre-generated mask is available it is also read simultaneously with the images into the plate dictionary structure provided by our method **load\_wells\_for\_plate\_nuclei()** available under the class **FluorescenceMicroscopy**. In case a mask isn't present, it is generated at runtime, using the following steps, some of which are implemented in Python in the package OpenCV (26).

1. **Artefact Removal (based on Intensity)** -  $X_{i,j} = 0$  if  $X_{i,j} > \alpha$  else  $X_{i,j}$ , where  $\alpha$  is the 'artefact\_threshold' for the nuclei channel.

2. **Background Noise Removal (based on Structure)** -

- (a) Dilation

$X' = X \oplus B = \bigcup_{b \in B} X_b$ , where  $X_b$  is the translation of  $X$  by  $b$ .

To be more specific,

$$X'_{i,j} = \begin{cases} 1 & (i,j) \in \{(i,j) | \exists (i',j') \in \mathcal{N}(i,j), X_{i',j'} = 1\} \\ 0 & \text{Otherwise} \end{cases}$$

- (b) Erosion

$X'' = X' \ominus C = \bigcap_{c \in C} X'_{-c}$ , where  $X'_{-c}$  is the translation of  $X'$  by  $-c$ .

To be more specific,

$$X''_{i,j} = \begin{cases} 1 & (i,j) \in \{(i,j) | \forall (i',j') \in \mathcal{N}(i,j), X_{i',j'} = 1\} \\ 0 & \text{Otherwise} \end{cases}$$

when we choose  $B$  as a sub-matrix of  $C$  it leads to the removal of the background noise  $B$  and  $C$  are chosen for the matrix representation of a circle of diameter,  $d = 2 \times \text{'correction\_ball\_radius'}$  for the nuclei channel.

3. **Binarise** -  $X''_{i,j} = 1$  if  $X'_{i,j} > \beta$  else 0, where  $\beta$  is the 'manual\_threshold' for the nuclei channel.

For readouts relating to the nuclei channel please look at the Page 8, **Classes for well and plaque level readouts**.

**Analysis of Virus Signal.** The analysis of the virus signal begins similarly to nuclei with the acceptance of a well image corresponding to the virus channel stained with crystal violet or fluorescent dye and imaged via brightfield or fluorescence microscopy. In case that a pre-generated mask is available it is also read simultaneously with the images into the plate dictionary structure provided by our method **load\_wells\_for\_plate\_nuclei()** available under the class **FluorescenceMicroscopy** or **load\_well\_images\_and\_masks\_for\_plate()** available under the class **CrystalViolet**. The relevant notebooks in the GitHub repository contain practical examples.

In case a mask isn't present, it is generated at runtime just like nuclei. But since there is a difference in how virus channel appears in crystal violet or fluorescent dye stained samples and also under mobile photography, bright-field or fluorescence microscopy, we have two different workflows for mask generation for plaques. Once again, part of these workflows are implemented in Python in the packages OpenCV (26), Scikit-Image (25) and SciPy (36). The workflows are as follows:

#### Virus Mask Generation for Fluorescence Microscopy.

1. **Binarise** -  $X_{i,j} = 1$  if  $X_{i,j} > \beta$  else 0, where  $\beta$  is the 'virus\_threshold' for the virus channel.
2. **Connected Components** -  $X'_{i,j} = 0$  if  $D_{i,j} > \gamma$  else 1, where  $D_{i,j}$  is the distance to the nearest foreground object and  $\gamma$  is the 'plaque\_connectivity' for the virus channel.

#### Virus Mask Generation for Mobile Photography or Brightfield Microscopy.

1. **Gaussian Blurring** -  $X'_{i,j} = X_{i,j} \cdot G_{i,j}$ , where

$$G_{i,j} = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{(i-h/2)^2 + (j-w/2)^2}{2\sigma^2}}$$

where  $(w,h)$  is the shape of  $X$  and  $\sigma$  is the 'sigma' passed to **PlaquesImageGray** Specimen class.

2. **Binarise** -  $X''_{i,j} = 1$  if  $X'_{i,j} > \beta$  else 0, where  $\beta$  is the 'threshold' passed to **PlaquesImageGray** Specimen class.

For readouts relating to the virus channel please look at Page 8, **Classes for Well and Plaque Level Readouts**.

**Classes for Well and Plaque Level Readouts.** PyPlaque returns readouts at the well and plaque level. It is worth mentioning that the modules described in this subsection are intended for and designed to work for fluorescence microscopy images. In this section, we discuss the classes that are used for such analysis and proceed to describe how we compute each of the readouts including those that are showcased in Fig. fig. 4B and C.

The classes *PlateReadout*, *WellImageReadout* and *PlaqueObjectReadout* can be used to deal with the analysis at different granularity levels. They are used to generate readouts pertaining to multiple wells of a single plate of fluorescence plaques, multiple instances of plaques within a single well and finally a single instance of a plaque from a fluorescence plaque well respectively. It is possible to obtain readouts at the well and/or plaque level based on flags passed to *PlateReadout*.

Examples of readouts at the well level obtained by methods under *WellImageReadout* are:

- **get\_max\_plaque\_intensity** - returns  $\max(X_v)$ , where  $X_v$  is the image from the virus channel for the well.
- **get\_mean\_plaque\_intensity** - returns  $\frac{1}{w \cdot h} \sum_{i,j} (X_v)$ , where  $X_v$  is as above with shape  $(w, h)$ .
- **get\_total\_plaque\_intensity** - returns  $\sum_{i,j} (X_v)$ , where  $X_v$  is as above.
- **get\_median\_plaque\_intensity** - returns  $q$  s.t.  $\frac{\sum_{i,j} \mathbb{1}_{X_v < q}}{w \cdot h} \approx 0.5$ , where  $X_v$  is as above and  $\mathbb{1}$  is the indicator function taking value 1 when the condition is satisfied and zero otherwise.
- **get\_nuclei\_count** - returns  $\frac{\sum_{i,j} Y_n}{(nuc_{max} - nuc_{min})}$  where  $Y_n$  is the mask from the nuclei channel for the well and  $(nuc_{min}, nuc_{max})$  are the upper and lower bounds for the area of a nucleus.
- **get\_plaque\_count** - returns the length  $l$  of the list of global maxima coordinates identified from  $X_v$  which is as described above. It is an extension of the steps mentioned in Page 7, *Virus Mask Generation for Fluorescence Microscopy* the further steps being,

3. **Individual Plaques** - getting **regionprops** (from scikit-image (25)) objects based on Connected Components (37) that are greater than  $vir_{min}$ , the lower bound for the area of a plaque from an image like  $X'_{i,j}$  mentioned in Page 7, *textitVirus Mask Generation for Fluorescence Microscopy*.

4. **Counting global maxima in blurred masked plaque image** - For each plaque object,  $X''_p = X'_p(\cdot)Y'_p$  where  $X'_p$  and  $Y'_p$  are the image and mask of a single plaque  $p$  respectively. Then we

use Gaussian blurring with a specified  $\sigma$  as described in Page 7, *Virus Mask Generation for Mobile Photography or Brightfield Microscopy* to get  $X'''_p$ .  $X'''_p$  is then used to identify global maxima (from scikit-image (25)) within it that are identified based on the hyper-parameter  $min_d$  so that the distance between two global maxima found is at least  $(2 \cdot min_d + 1)$  indicating that the spread of the area for which a global maxima is found is at least of radius  $min_d$ . This operation dilates (see Page 7, *Analysis of Nuclei*) and merges neighbouring maxima that become part of the same connected component after dilation. Coordinates  $(x, y)$  of the locations where  $X'''_p$  is equal to the dilated image are returned as global maxima. The length of this list of coordinates can therefore be measured.

- **get\_infected\_nuclei\_count** - returns  $\frac{\sum_{p'} area(p')}{(nuc_{max} - nuc_{min})}$  where  $p' \in \{\text{plaques identified from } Y_v \text{ using step 3 under } get\_plaque\_count \text{ mentioned previously}\}$ ,  $Y_v$  being the mask of the virus channel for the well,  $(nuc_{min}, nuc_{max})$  are as described previously and  $area(p')$  is simply the area of the plaque  $p'$ .
- **get\_lesion\_area** - returns  $\sum_{i,j} (Y_v)$ , where  $Y_v$  is as mentioned above.
- **get\_plaque\_objects** - returns the **regionprops** (from scikit-image (25)) objects identified from  $Y_v$ , where  $Y_v$  is as mentioned above, using step 3 under *get\_plaque\_count* mentioned above.

Other than the methods mentioned above, the class provides the following methods such as *get\_nuclei\_image\_name*, *get\_plaque\_image\_name*, *get\_row*, *get\_column*, *get\_max\_nuclei\_intensity*, *get\_mean\_nuclei\_intensity*, *get\_total\_nuclei\_intensity*, *get\_median\_nuclei\_intensity* as well. They have not been described in detail since they are either similar to the ones described and can be thought of analogously or they are self-explanatory from their names.

The class calls via the *call\_plaque\_object\_readout* method the readouts at the plaque level. An example of readouts at the plaque level obtained by methods under *PlaqueObjectReadout* are:

- **get\_area** - returns **regionprops** (from scikit-image (25)) area of the plaque,  $\sum_{i,j} Y_p$ , where  $Y_p$  is the mask of the plaque or if indicated by a flag the Pick's area (see Page 9, *Pick's Area and Perimeter*) based on  $Y_p$  is returned.
- **get\_perimeter** - returns **regionprops** perimeter of the plaque, or if indicated by a flag the Pick's perimeter (see Page 9, *Pick's Area and Perimeter*) based on  $Y_p$  is returned.

- **get\_centroid** - returns **regionprops** centroid of the plaque,  $(\frac{\sum_{j \in r_p} j}{|r_p|}, \frac{\sum_{i \in c_p} i}{|c_p|})$  where  $|r_p|$  and  $|c_p|$  are the number of pixel rows and columns of the plaque mask  $Y_p$  or the plaque image  $X_p$ .
- **get\_bbox** - returns a tuple  $(x, y, w_p, h_p)$  where  $(x, y)$  are the bounding box coordinates from the top left corner of the plaque and  $(w_p, h_p)$  is the shape of bounding box of the plaque.
- **get\_major\_minor\_axis\_length** - returns a tuple  $(4a, 4b)$  where  $a$  and  $b$  are as described in Eq. (1).
- **get\_eccentricity** - returns the value of eccentricity as described in Eq. (1).
- **get\_roundness** - returns the value of roundness (35) as described in Eq. (2).
- **get\_number\_of\_peaks** - returns the length of a list of coordinates  $(x, y)$  of the global maxima found in the masked image of the plaque (similar to  $X_p''$ ) using the method mentioned in step 4, **Counting global maxima in blurred masked plaque image** of **get\_plaque\_count** under the well level readouts.
- **get\_nuclei\_in\_plaque** - returns  $\frac{area(Nuc)}{(nuc_{max} - nuc_{min})}$  where  $(nuc_{min}, nuc_{max})$  are as described previously and  $area(Nuc)$  is either the sum of non-zero pixels or if indicated by a flag, the Pick's area (see Page 9, **Pick's Area and Perimeter**) of a masked image  $Nuc$  defined as  $\sum_{i,j} Y_p^{ch}(\cdot) Y_n$  where  $Y_p^{ch}$  is the **regionprops** convex-hull (38) of the plaque mask  $Y_p$  and  $Y_n$  is the corresponding nuclei mask from the nuclei channel.
- **get\_infected\_nuclei\_in\_plaque** -  $\frac{\sum_{i,j} Y_p(\cdot) Y_n}{(nuc_{max} - nuc_{min})}$  where  $(nuc_{min}, nuc_{max})$ ,  $Y_p$  and  $Y_n$  are all as described previously.
- **get\_convex\_area** - returns **regionprops** convex area of the plaque which is the sum of non-zero pixels in a binary image of the convex hull (38) of the plaque.
- **max\_GFP\_intensity** -  $max(X_p(i, j(\cdot)) Y_p(i, j))$ , where  $X_p$  and  $Y_p$  are the image and mask of the plaque  $p$ .
- **total\_GFP\_intensity** -  $\sum_{i,j} X_p(i, j)(\cdot) Y_p(i, j)$ , where  $X_p$  and  $Y_p$  is as above.
- **mean\_GFP\_intensity** -  $\frac{\sum_{(i', j')} X_p(i', j')(\cdot) Y_p(i', j')}{|(i', j')|}$ , where  $X_p$  and  $Y_p$  is as above and  $(i', j')$  represents a non-zero pixel in  $X_p(\cdot) Y_p$ .

Other than the methods mentioned above, the class provides the following methods such as **get\_row**, **get\_column** as well. For getting the means, maximums and totals and storing readouts in dataframes, implementations available from

Numpy (39) and Pandas (21) are used. The readouts for at the plaque level are averaged for all plaques within a well before being stored in a dataframe but individual plaque measurements can be accessed as shown in our [fluorescence plaque detailed](#) notebook meant for advanced users of the software.

Further details of the readouts at both levels can be found in the [view](#) module in the [GitHub](#) page of the project.

**Pick's Area and Perimeter.** The Pick's Area and Perimeter used here is the main result of Pick's theorem (40), first showcased by Georg Alexander Pick. It is a geometric result for calculating the area and perimeter of irregular polygons on integer lattices (41). The theorem states that given a simple lattice polygon  $P$ , if  $i$  is the number of integer lattice points inside it and  $p$  is the number of integer lattice points on its perimeter, then the area of the polygon  $area(P)$ , is given by,

$$area(P) = i + \frac{p}{2} - 1 \quad (3)$$

where a point  $(x, y) \in \mathbb{R}^2$  is an integer lattice point, or simply lattice point, **iff** both  $x$  and  $y$  are integers. We define a lattice polygon to be a polygon all of whose vertices are at integer lattice points.

It is important to note that the same formula does not apply for polygons with holes where it changes to,

$$area(P) = i + \frac{p}{2} - q \quad (4)$$

where  $q$  depends on the number of holes. For a proof of the theorem please look at the work here (42).

As for the perimeter and area calculations in our code, we calculate the perimeter in both cases and use it along with the interior measurements to get the total area and simply return the perimeter estimate for the perimeter. We can do so using the following steps:

1. Creating a border image  $X'$  using Dilation or Erosion (see Page 7, **Analysis of Nuclei**) and subtracting or subtracting it from the original respectively. The eroded image forms the interior of the polygon. The neighbourhood definition  $N(i, j)$  defines the result of these operations. For our purposes,

$$N(i, j) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

2. We embolden the resulting border image by convolution (43),  $X'' = X' * K$  where,

$$K = \begin{bmatrix} 10 & 2 & 10 \\ 2 & 1 & 2 \\ 10 & 2 & 10 \end{bmatrix}$$

3. Taking a weighted average of the histogram of the above-mentioned convolved image  $X''$ , giving appropriate weightage to the intensity values in the bins of the histogram. For computational simplicity, we keep the number of bins at 50.

$$p = \sum_i h_i \cdot w_i,$$

$$i \in \{1, \dots, n_b\}, n_b = 50$$

## References

1. Oona Delpuech, Julie A. Douthwaite, Thomas Hill, Dhevahi Niranjani, Nancy T. Malintan, Hannah Duvoisin, Jane Elliott, Ian Goodfellow, Myra Hosmillo, Alexandra L. Orton, Molly A. Taylor, Christopher Brankin, Haidee Pitt, Douglas Ross-Thriepand, Magdalena Siek, Anna Cuthbert, Ian Richards, John R. Ferdinand, Colin Barker, Robert Shaw, Cristina Ariani, Ian Waddell, Steve Rees, Clive Green, Roger Clark, Abhishek Upadhyay, and Rob Howes. Heat inactivation of clinical COVID-19 samples on an industrial scale for low risk and efficient high-throughput qRT-PCR diagnostic testing. *Scientific Reports*, 12(1):2883, 2022. doi: 10.1038/s41598-022-06888-z.
2. Ioanna Smyraki, Martin Ekmann, Antonio Lentini, Nuno Rufino de Sousa, Natali Papanicolaou, Martin Vondracek, Johan Aarum, Hamzah Safari, Shaman Muradrasoli, Antonio Gigliotti Rothfuchs, Jan Albert, Björn Högberg, and Björn Reinius. Massive and rapid COVID-19 testing is feasible by extraction-free SARS-CoV-2 RT-PCR. *Nature Communications*, 11(1):4812, 9 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-18611-5.
3. Olivier Vandenberg, Delphine Martiny, Olivier Rochas, Alex van Belkum, and Zisis Kozlakidis. Considerations for diagnostic COVID-19 tests. *Nature Reviews Microbiology*, 19(3):171–183, 10 2020. ISSN 1740-1526, 1740-1534. doi: 10.1038/s41579-020-00461-z.
4. R Dulbecco. Production of plaques in monolayer tissue cultures by single particles of an animal virus. *Proc. Natl. Acad. Sci. U. S. A.*, 38(8):747–752, 8 1952. doi: 10.1073/pnas.38.8.747.
5. F. d’Herelle and George H. Smith. The bacteriophage and its behaviour. *Nature*, 118(2962): 183–185, 8 1926. doi: 10.1038/118183a0.
6. Bhavesh D. Kevadiya, Jatin Machhi, Jonathan Herskovitz, Maxim D. Oleynikov, Wilson R. Blomberg, Neha Bajwa, Dhruvkumar Soni, Srijaanee Das, Mahmudul Hasan, Milankumar Patel, Ahmed M. Senan, Santhi Gorantla, JoEllyn McMillan, Benson Edagwa, Robert Eisenberg, Channabasavaiah B. Gurumurthy, St Patrick M. Reid, Chamindie Punyadeera, Linda Chang, and Howard E. Gendelman. Diagnostics for SARS-CoV-2 infections. *Nature Materials*, 20(5):593–605, 2 2021. ISSN 1476-1122, 1476-4660. doi: 10.1038/s41563-020-00906-z.
7. Antonio E. Muruato, Camila R. Fontes-Garfias, Ping Ren, Mariano A. Garcia-Blanco, Vireet D. Menachery, Xuping Xie, and Pei-Yong Shi. A high-throughput neutralizing antibody assay for COVID-19 diagnosis and vaccine evaluation. *Nature Communications*, 11(1): 4059, 8 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-17892-0.
8. Alda Natale, Elisa Mazzotta, Nicoletta Mason, Letizia Ceglie, Monica Mion, Annalisa Stefani, Alice Fincato, Francesco Bonfante, Alessio Bortolami, Isabella Monne, Laura Bellinati, Carmine Guadagno, Erika Quaranta, Ambra Pastori, and Calogero Terregino. SARS-cov-2 natural infection in a symptomatic cat: Diagnostic, clinical and medical management in a one health vision. *Animals (Basel)*, 11(6):1640, 6 2021. ISSN 2076-2615. doi: 10.3390/ani11061640.
9. Artur Yakimovich and Jason Mercer. High-content analyses of vaccinia plaque formation. *Methods in Molecular Biology*, 2023:237–253, 2019. doi: 10.1007/978-1-4939-9593-6\_15.
10. Artur Yakimovich, Heidi Gumpert, Christoph J. Burckhardt, Verena A. Lütsch, Andreas Jurget, Ivo F. Sbalzarini, and Urs F. Greber. Cell-free transmission of human adenovirus by passive mass transfer in cell culture simulated in a computer model. *Journal of Virology*, 86(18):10123–10137, 9 2012. ISSN 0022-538X. doi: 10.1128/jvi.01102-12.
11. Artur Yakimovich, Vardan Andriasyan, Robert Witte, I-Hsuan Wang, Vibhu Prasad, Maarit Suomalainen, and Urs F. Greber. Plaque2.0—a high-throughput analysis framework to score virus-cell transmission and clonal cell expansion. *PLOS ONE*, 10(9):e0138760, 9 2015. ISSN 1932-6203. doi: 10.1371/journal.pone.0138760.
12. Corina Beerli, Artur Yakimovich, Samuel Kilcher, Glennys V Reynoso, Gotthold Fläschner, Daniel J Müller, Heather D Hickman, and Jason Mercer. Vaccinia virus hijacks EGFR signalling to enhance virus spread through rapid and directed infected cell motility. *Nature Microbiology*, 4(2):216–225, 2 2019. ISSN 2058-5276. doi: 10.1038/s41564-018-0288-2.
13. Marco Cacciabue, Anabella Currà, and Maria I. Gismondi. ViralPlaque: a Fiji macro for automated assessment of viral plaque statistics. *PeerJ*, 7(e7729):e7729, 9 2019. ISSN 2167-8359. doi: 10.7717/peerj.7729.
14. Siân Culley, Greg Towers, David Selwood, Ricardo Henriques, and Joe Grove. Infection counter: Automated quantification of in vitro virus replication by fluorescence microscopy. *Viruses*, 8(7):201, 7 2016. ISSN 1999-4915. doi: 10.3390/v8070201.
15. Leah K. Katelnick, Ana Coello Escoto, Benjamin D. McElvany, Christian Chávez, Henrik Salje, Wensheng Luo, Isabel Rodríguez-Barraquer, Richard Jarman, Anna P. Durbin, Sean A. Diehl, Derek J. Smith, Stephen S. Whitehead, and Derek A. T. Cummings. Viridot: An automated virus plaque (immunofocus) counter for the measurement of serological neutralizing responses with application to dengue virus. *PLOS Neglected Tropical Diseases*, 12(10):e0006862, 10 2018. ISSN 1935-2727. doi: 10.1371/journal.pntd.0006862.
16. Yannick Kerkhoff, Stefanie Wedepohl, Chuanxiong Nie, Vahid Ahmadi, Rainer Haag, and Stephan Block. A fast open-source Fiji-macro to quantify virus infection and transfection on single-cell level by fluorescence microscopy. *MethodsX*, 9(101834):101834, 9 2022. ISSN 22150161. doi: 10.1016/j.mex.2022.101834.
17. Ellina Trofimova and Paul R. Jäschke. Plaque size tool: An automated plaque analysis tool for simplifying and standardising bacteriophage plaque morphology measurements. *Virology*, 561:1–5, 9 2021. ISSN 00426822. doi: 10.1016/j.virol.2021.05.011.
18. Gridsada Phanomchoeng, Chayatorn Kukiattikoon, Suphanut Plengkharn, Siwaporn Boonyasupayakorn, Saran Salakij, Suvit Poomrittingul, and Luchakorn Wuttisitulkij. Machine-learning-based automated quantification machine for virus plaque assay counting. *PeerJ Computer Science*, 8:e878, 2022.
19. Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel Tomancak, and Albert Cardona. Fiji: an open-source platform for biological-image analysis. *Nature Methods*, 9(7):676–682, 6 2012. ISSN 1548-7091. doi: 10.1038/nmeth.2019.
20. Artur Yakimovich Trina De, Vardan Andriasyan. Pyplaque: v0.2.0, 10 2024.
21. Wes McKinney. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 56–61. SciPy, 2010. doi: 10.25080/majora-92b1922-00a.
22. Fernando Perez and Brian E. Granger. IPython: A system for interactive scientific computing. *Computing in Science & Engineering*, 9(3):21–29, 2007. ISSN 1521-9615. doi: 10.1109/mcse.2007.53.
23. Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.
24. The MathWorks Inc. Matlab version: 9.13.0 (r2022b), 2022.
25. Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Goullart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in python. *PeerJ*, 2:e453, 6 2014. ISSN 2167-8359. doi: 10.7717/peerj.453.
26. G. Bratski. The opencv library. *Dr. Dobbs’s Journal of Software Tools*, 2000.
27. Trina De, Adrian Urbanski, Subasini Thangamani, Maria Wyrzykowska, and Artur Yakimovich. VACVPlaque: mobile photography of Vaccinia virus plaque assay with segmentation masks. 6 2024. doi: 10.14278/rodare.3003.
28. Fanny Georgi, Vardan Andriasyan, Robert Witte, Luca Murer, Silvio Hemmi, Lisa Yu, Melanie Grove, Nicole Meili, Fabien Kuttler, Artur Yakimovich, et al. The fda-approved drug nelfinavir inhibits lytic cell-free but not cell-associated nonlytic transmission of human adenovirus. *Antimicrobial agents and chemotherapy*, 64(9):10–1128, 2020.
29. Luca Murer, Romain Volle, Vardan Andriasyan, Anthony Petkidis, Alfonso Gomez-Gonzalez, Liliane Yang, Nicole Meili, Maarit Suomalainen, Michael Bauer, Daniela Policarpio Sequeira, et al. Identification of broad anti-coronavirus chemical agents for repurposing against sars-cov-2 and variants of concern. *Current research in virological science*, 3:100019, 2022.
30. Vaibhav Sharma and Artur Yakimovich. A deep learning dataset for sample preparation artefacts detection in multispectral high-content microscopy. *Scientific Data*, 11(1):232, 2024.
31. Nobuyuki Otsu et al. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.
32. Robert Witte, Vardan Andriasyan, Fanny Georgi, Artur Yakimovich, and Urs F Greber. Concepts in light microscopy of viruses. *Viruses*, 10(4):202, 2018.
33. Artur Yakimovich. Artificial intelligence in infection biology. *Artificial Intelligence in Medicine*, pages 1–9, 2020.
34. Ayoub B. Ayoub. The eccentricity of a conic section. *The College Mathematics Journal*, 34(2):116–121, 2003. doi: 10.1080/07468342.2003.11921994.
35. Murk J. Bottema. *Circularity of objects in images*, pages 2247–2250. IEEE International Conference on Acoustics, Speech and Signal Processing. Proceedings. Institute of Electrical and Electronics Engineers, 2000. ISBN 0-7803-6293-4. doi: 10.1109/ICASSP.2000.859286.
36. Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.
37. Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, and Yuyan Chao. The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition*, 70:25–43, 2017. doi: 10.1016/j.patcog.2017.04.018.
38. Vignesh Kottayam Viswanathan, Sina Sharif Mansouri, and Christoforos Kanellakis. Chapter 11 - aerial infrastructures inspection. In George Nikolakopoulos, Sina Sharif Mansour, and Christoforos Kanellakis, editors, *Aerial Robotic Workers*, pages 175–211. Butterworth-Heinemann, 2023. ISBN 978-0-12-814909-6. doi: <https://doi.org/10.1016/B978-0-12-814909-6.00017-2>.
39. Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 9 2020. doi: 10.1038/s41586-020-2649-2.
40. G. Pick. Geometrisches zur Zahlenlehre., 1899.
41. JOHN J. SULLIVAN. Polygons on a lattice. *The Arithmetic Teacher*, 20(8):673–675, 1973. ISSN 0004136X.
42. W. W. Funkenbusch. From euler’s formula to pick’s formula using an edge theorem. *The American Mathematical Monthly*, 81(6):647–648, 1974. doi: 10.1080/00029890.1974.11993639.

43. Kunihiro Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4): 193–202, 4 1980. ISSN 1432-0770. doi: 10.1007/BF00344251.