FILE NOT INTENDED FOR PRODUCTION

Deepometry, a framework for applying supervised and weakly supervised deep learning to imaging cytometry

Minh Doan[1,2+], Claire Barnes[3+], Claire McQuin[1], Juan C. Caicedo[1], Allen Goodman[1], Anne E. Carpenter[1*] and Paul Rees[1,3*]

1. Imaging Platform, Broad Institute of Harvard and MIT, Cambridge, Massachusetts MA 02142, United States.
2. Bioimaging Analytics, GlaxoSmithKline, Collegeville, PA 19426, United States.
3. College of Engineering, Swansea University, Bay Campus, Swansea SA1 8EN, United Kingdom.

+, * These authors contributed equally.
Correspondence should be addressed to P.R. (p.rees@swansea.ac.uk +44-1792-295197), A.E.C. (anne@broadinstitute.org +1-617-714-7750) or M.D. (minh.x.doan@gsk.com +1-484-644-5973)

EDITORIAL SUMMARY This protocol describes Deepometry, an open-source application for supervised and weakly supervised deep learning analysis of imaging flow cytometry datasets. The protocol provides runtime scripts for Python, MATLAB and a stand-alone application.

TWEET A new protocol for deep learning analysis of imaging flow cytometry datasets using #Deepometry.

COVER TEASER Deep learning analysis of imaging cytometry data

ABSTRACT

Deep learning offers the potential to extract more than meets the eye from images captured by imaging flow cytometry. This protocol describes the application of deep learning to single-cell images to perform supervised cell classification and weakly supervised learning, using example data from an experiment exploring red blood cell morphology. We describe how to acquire and transform suitable input data as well as the steps required for deep learning training and inference using an open-source web-based application. All steps of the protocol are provided as open-source Python as well as MATLAB runtime scripts, through both command-line and graphic user interfaces. The protocol enables a flexible and friendly environment for morphological phenotyping using supervised and weakly-supervised learning and the subsequent exploration of the deep learning features using multi-dimensional visualization tools. The protocol requires 40 hours when training from scratch, and 1 hour using a pre-trained model.

INTRODUCTION

Imaging flow cytometry (IFC) is a powerful technique that combines the high-throughput nature of flow cytometry together with the ability to capture multi-channel fluorescence images of every cell [1]. Current instruments can measure 12 channel fluorescence images at a magnification of up to 60X from 3 different excitation wavelength sources at rates of thousands of cells per second. This technology has found a host of different applications such

as the investigation of receptor activity [2], nuclear translocation studies [3], the detection of foci in the nucleus of cells with DNA damage [4], asymmetric cell division [5], autophagy [6] and cell cycle analysis [7]. While all of these applications have taken advantage of the image capture capabilities, not possible with traditional flow cytometry, most have only used very basic information from these images. For example, early approaches used simple image-derived morphological features such as cell area or pixel intensities within the cell or its nucleus[3,5,7]. As few parameters were extracted per cell, the typical gating strategies employed in traditional flow cytometry sufficed for these applications but fell far short of the potential in the rich datasets obtained from imaging flow cytometry.

The advent of high throughput fluorescence microscopy has driven the rapid development of techniques to analyse very large numbers of cell images [8,9]. Typically these techniques involve the segmentation of individual cell borders from each field of view, the extraction of the rich information from the multichannel cell images [10], and the subsequent analysis of these high-dimensional datasets using tools such as machine learning [11]. High content analysis, where several hundreds of morphological, intensity and texture features are extracted per fluorescence image, yields a rich dataset that can be used with standard high dimensional analysis techniques to explore cell phenotypes. These high content image analysis tools can be readily applied to images from IFC because the suspension cells do not need to be segmented, which can be challenging for single-cell microscopy analysis [12]. There has also been rapid progress in the development of non-fluorescent high throughput modalities such as quantitative phase [13] and Raman [14] imaging which can provide single-cell images for high content analysis.

Previously we developed open-source software and procedures for extracting image features from the multiple fluorescence, bright field and dark field (side scatter) image channels of imaging flow cytometry data from an ImageStream [15] using the open-source software CellProfiler [16]. CellProfiler measures hundreds of morphological features per channel for individual cells and subcellular structures, such as nuclei. The high-dimensional data can then be analysed using machine learning and clustering algorithms for various tasks including phenotype classification and identifying the impact of drug treatments. In particular, we used supervised machine learning to detect cell cycle phases in cells from imaging flow data and to reconstruct the DNA stain intensity from bright and dark field images [17]. While this approach maximises the usefulness of the information-rich imaging flow cytometry datasets, the process involves several steps requiring different software solutions and is time-consuming. Also, as the researcher needs to predefine the morphological features to be included in the analysis this process can be subjective.

Recent advances in multi-core graphical processing units have provided the computational power required to train deep neural networks. Their accuracy typically surpasses that of traditional machine learning methods for image recognition problems, while also eliminating the hand-tuned feature extraction step[18]. Multichannel images are directly input to a first convolution layer which is connected to several additional convolution layers that down-sample the image size and successively define more complicated image features by training the network with labelled images.

In this protocol, we describe the use of deep learning for the common classification problem of phenotype identification as well as for data exploration. The workflow we describe here uses Deepometry, open-source software we developed to prepare imaging cytometry data in the proper format for deep learning and carry out phenotype classification. We outline the different data pre-processing steps that can affect the usefulness of these algorithms. We

describe how to use the activation values of one of the later fully connected layers of the network to allow data exploration using high dimensional visualisation and mapping techniques. These techniques, such as t-SNE [19], diffusion mapping [20] and UMAP [21], can determine relationships among cells, for example, to map out the timing of each cell within its cell cycle [17] or to visualise the continuous morphology changes of red blood cells during storage for transfusion [22]. We provide scripts in MATLAB and Python including a graphic user interface and a stand-alone application to perform all the steps and techniques outlined here.

Development of the protocol

Our first efforts to exploit the information-rich image data from imaging flow cytometry mirrored the traditional methods applied for microscopy-based high content data analysis. We used IDEAS, the ImageStream analysis software, to export 16-bit (raw unprocessed) TIFF-format images of every cell. Typically, IFC image sizes are small (less than 100X100 pixels) so we developed open-source software to montage the single-cell images to large images with grids of thousands of cells to improve the efficiency of file handling. We used CellProfiler [16] to segment the individual cells and then extract hundreds of features per cell for each fluorescence channel. The features included morphological measurements such as metrics of size and shape, granularity and texture, and fluorescence intensities, from the cells and subcellular organelles. A large number of feature values per cell were then used as input to traditional machine learning algorithms to perform supervised cell classification and unsupervised clustering. This strategy succeeded in demonstrating that the position of a cell in the cell cycle, and the intensity of a DNA stain, could be predicted from just bright and dark field images [17]. We later refined this protocol to generate the montaged cell images directly from the ImageStream proprietary output file (.cif) format [23] eliminating the TIFF image exporting step.

While the ImageStream's single-cell image format is generally inconvenient for input into the majority of cell image analysis software tools, which were developed to analyse wide-field microscopy images, they are ideal for deep learning algorithms, which have been designed to readily accept individual images of each object of interest. We therefore next used the IDEAS software to export images representing each channel, which were then combined into multi-layered TIFF-format images. We found that DeepFlow [18], a neural network based on the "Inception" architecture [24], significantly improved classification accuracy [23] as compared to our previous cell cycle analysis using traditional machine learning, which required feature extraction from the cell images. We also found the activation values of the last fully connected layer of the network to be helpful in exploring the data using tools such as t-SNE, UMAP, and diffusion maps [19, 20, 21].

In addition to refinements to the logistics of handling the incoming image files, our later refinements to the protocol include providing an option to take advantage of a pre-trained deep learning network and use transfer learning to classify cell phenotypes, as opposed to training the DeepFlow network from scratch. This approach can be beneficial when training sets are small, and we successfully used it to classify cells from the bone marrow of patients diagnosed with acute lymphoblastic leukaemia (ALL) [25]. The accuracy of detecting leukemic lymphoblasts was over 93% when the cells were labelled with an ALL-discriminating antibody combination while our strategy of training using just the bright and dark field images (thus reducing substantial labor and reagents) achieved an accuracy of 88% [25]. To achieve this level of accuracy we used a modified version of the ResNet50 architecture, which was developed and trained to perform classification on the million images of the ImageNet database [26]. The original network accepts a 3 (RGB) channel 224 by 224 pixel input so we

modified the network to accept the smaller images from the ImageStream and include an arbitrary number of channels. We took the pre-trained network as a starting point and used transfer learning to speed up the training process. This forms the core of the current Deepometry deep learning protocol described herein.

Clearly, deep learning is a powerful alternative to traditional machine learning on classically extracted features, delivering increased classification accuracy and eliminating the need for feature engineering. Also, deep neural networks can be used to employ novel training strategies such as weakly supervised learning, which we include as part of this protocol. Weakly supervised learning involves training a deep learning network with images annotated in such a way to promote maximum learning and generate the greatest number of generally useful features. This is achieved by annotating the images with a label that may not be directly relevant to the cell morphology but be an easily obtainable label, such as the day the cells were imaged. Obviously, we would not expect the network to classify the images based on such labels and accuracy may be very low. However, setting an auxiliary task for the network can yield interesting benefits. Often the upper convolution layers generate features, which can be used for another task.

We used this weakly supervised learning to classify the morphology changes in red blood cells during storage for transfusion [22]. Instead of annotating thousands of cells into different morphological phenotypes, we trained Deepometry using cells labelled with the number of days the blood had been in storage. The storage time may 'weakly' encode temporal phenotypic variations of the cells and is a far poorer label than the cell phenotype itself. However, these labels are trivial to generate. As expected, the accuracy of predicting the storage time was low however the features generated by the network could be mapped onto a progression of the morphological phenotype of each cell with a high correlation (>0.9) compared with manual labels. The protocol described here includes the option to perform weakly supervised learning by changing the annotation of the cell images used to train the network.

Applications of the method

The protocol and software described herein are flexible; it can perform both supervised and weakly-supervised learning. Researchers can carry out the protocol using our provided easy-to-use scripts, a web browser-based app that has been developed using Flask (Python), or a MATLAB standalone app. The latter options are compiled along with all required dependencies and may be downloaded and used on any machine. The protocol allows visualising the activation values of key layers in the deep learning network using U-map, but activation values can also be exported into any dimensional reduction visualisation tool, for example, to perform cluster analysis or assign a pseudo-time to each cell in the population [23]. The software is not optimized to a particular biological application. Instead, it currently contains a variety of pre-designed architectures and pre-trained classifiers but can also be expanded to accommodate new developments in the rapidly changing field of deep learning.

The protocol might also be expanded to real-time analysis as opposed to the post-acquisition classification (offline analysis) described here. Multiple proof-of-concept studies have demonstrated the possibility to rapidly actualize the categorical prediction in a microfluidic system, allowing on-the-fly cell sorting based entirely on morphology [27,28]. Furthermore, a recent application of a convolutional neural network with 16 layers (VGG-16) enabled an *in silico* enhancement of acquired image data from an imaging flow cytometry system by virtually freezing the motion of flowing cells on the sensor to achieve ~1,000 times longer exposure time for microscopy-grade fluorescence image, allowing high-throughput without

sacrificing sensitivity [29]. The protocol here could be used as a testbed for developing algorithms for on-the-fly classification.

While the protocol was developed for use with imaging flow data, it could be readily adapted to traditional microscopy image data. For use with this protocol, individual cells from microscopy images must be segmented [30,31] and individually cropped cells saved as separate images; these can be input directly to this protocol. The different size cells would be standardised to the correct size for the deep learning network using the padding/cropping steps of the protocol; down-sampling might be required for high-resolution images in order to take advantage of the included networks with modest computing power.

A huge diversity of biological applications could be accomplished with this workflow. Two variants of phenotype classification are possible: first, the system can be trained to identify phenotypes that are annotated by human expert assessment. Second, the system can be trained using an alternate source of ground truth, such as a fluorescent label that identifies the cells of interest, that is not provided to the machine learning system. In this strategy, the system is forced to use available image-based features (such as in the label-free bright field and dark field channels only) to learn to recognize the phenotype. The cues it learns may not even be visible to the human visual system. Label-free cytometry using machine and deep learning on images generated from a range of different measurement techniques has now been used for blood cell classification [32, 33], rare cell [34] and cancer detection [35, 36]. Recently quantitative phase microscopy has been used to measure the progression of cell states including the activation of T cells[37] and progression of B-cell ALL[38] and this protocol could be used on these single-cell images to quantify and visualise progressive morphological changes.

Apart from phenotype classification, this protocol can be used to extract biologically relevant image features that allow the unsupervised arranging of cells in two ways: first, cells can be clustered in space to identify subpopulations, including those that may not be distinguishable to humans, and second, cells can be ordered based on a biological progression in order to identify dynamic processes such as development, metastasis, or degradation.

Comparison with classical bioimaging analysis approaches

A conventional image analytic pipeline typically consists of several steps: i) segmentation of the object-of-interest; ii) feature measurement; iii) feature selection and dimension reduction; iv) classification/regression and phenotyping (upper panel, blue path in **Fig. 1**). This design, however, has a number of limitations. First, each of these steps typically requires different software/platforms, forcing researchers to gather suitable solutions and familiarise themselves with software usage and installations. The input/output formats might be unique for each package and bespoke tools are then required to bridge them. Second, the efficacy of a classical pipeline depends on fine-tuning parameters in each module. Restarting and assessing the pipeline after each adjustment is time-consuming. Finally, the handcrafted features (shape, intensities, texture, etc.) need to be pre-defined. This might not be an issue when the phenotype is known to exhibit certain known characteristics, however, a combination of these features is often required, demanding researchers to carefully conduct feature selection procedures to search for the right set of features amidst the vast number of combinatorial choices.

In contrast, a deep learning-based method naturally supports the interconnection of feature extraction, feature selection and phenotyping in one integrated architecture, and therefore reduces the number of user-tunable parameters for each task (lower panel, orange path in **Fig. 1**). Today state-of-the-art deep neural network architectures allow the information in pixel

patterns to be stored in millions of intrinsic parameters (weights), which are automatically updated each time the training materials pass through the network (an *epoch*). The features that are relevant to the ultimate classification/regression target are extracted and selected directly from the input pixels, which means an accurate segmentation of each object-of-interest might not be necessary. A well-separated, single-framed object, as in the case of IFC, is convenient input for a deep neural network.

Comparison with other deep learning approaches

Given that several deep neural networks are successfully applied in image classification contests in the computer vision domain, such as ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and Common Objects in Context (COCO), it is tempting to apply the winning solutions to bioimaging data; using a network trained in one domain on another is termed *transfer learning*. Limitations exist, however: (a) pre-trained networks are usually based on one (grayscale), three (RGB), or four (CMYK) channel inputs, which are incompatible with the multiple channels contained in biological images (sometimes in the order of 40+ channels); (b) the efficiency of pre-trained networks are often strongly linked to specific input size, such as 224x224 in the original use of ResNet50 on ImageNet; biological images have a wide range of frame sizes that often require preprocessing treatments to fit into pre-trained networks, (c) information per channel in photographic images are often sufficient per se (i.e. a single channel of the red color of a cat still provides sufficient features for the correct classification of the cat); while in bioimaging data, each channel captures unique information of biological components (e.g. fluorescent signals in one channel provides information of proteins, another channel provides information of lipids, exclusively).

Our protocol addresses these issues using our Python library Keras-ResNet [https://github.com/broadinstitute/keras-resnet], which allows unlimited dimensions of the input shape, including widths, heights and number of channels. This framework efficiently enables the implementation of deep residual networks, which contain a stack of convolutional layers that gradually transform the input signal into a feature vector. Each convolutional layer is composed of a set of learned filters that recognize patterns in a small region of the input (size of 3×3) with a stride of 1. The depth of our CNN is appropriate for the typical resolution of IFC; 10 µm-width cells are captured in 20-30 pixels. As layers of a network build a hierarchy of increasingly more complex nonlinear features, deeper networks involving more layers have the ability to detect complex relationships that may be used for challenging classification problems. It has been shown, however, that, as more layers are added to the network, degradation in performance can be observed. Our workflow is based on the ResNet50 architecture which is 50 layers deep and includes skip connections, a strategy ensuring that layers that are not useful do not degrade training. Additionally, our selected CNN architecture has a light footprint and can be efficiently run on a CPU in an acceptable time period for typical classification problems (~30 hours, see **Table 1**).

Recently a general deep learning tool, AIDeveloper (AID)[39] has been developed to allow the application of deep neural networks for image classification. The AID has a user-friendly graphical user interface that allows researchers to choose from several different networks and allows transfer learning from previous training image sets. AID is well suited to the use for the small image datasets from typical cytometry platforms and the authors demonstrate high accuracy on the classification of cell types from whole blood bright field real-time deformability cytometry images. The application of different neural networks to classification problems can run within the graphical user interface while with Deepometry this requires changes to the code. However, currently, AID does not allow the user to visualise the layers

of the network to study cell morphology progression and does not allow the use of weakly supervised learning to eliminate the need for phenotype annotation; this was the major advancement recently demonstrated in the context of red blood cells [21], the subject of this protocol.

Levels of expertise required

The protocol is designed to be accessible to any researcher, regardless of computational expertise. We provide a step-by-step installation guide for the user to install a simple executable app on their local computer (general workflow shown in **Fig. 2**). For those interested in customizing the code, we also provide instructions to install Python and/or MATLAB in the correct configuration to run the protocol web apps and software packages. Once code and accompanying packages have been downloaded onto the computer following the step-by-step instructions, the protocol requires very little prior experience. Knowledge of the dataset and what you hope to achieve is key to derive the most benefit from the workflow. You may wish to explore and interpret some of the features extracted from various layers of the trained model and so some familiarity with the overall network architecture is useful but not crucial. Scientists may need to consult information technologists to assist with setup if the workflow is to be run on a commercial cloud or local server computing system.

Limitations

Deep learning in general is data-hungry and requires training examples on the order of hundreds of thousands to millions, unless pre-trained networks are used. Typically, IFC produces images at this scale, but labelling images according to their phenotype manually is not scalable. It is therefore ideal to use a secondary source of information as ground truth, such as a fluorescent stain in an alternate channel that indicates the phenotype classes, or to physically separate samples containing different phenotype classes, while taking care to minimize technical variation among batches. As well, the accuracy of deep learning depends on the depth and complexity of its architecture, which can require large computational resources, primarily during the training step.

Commercial IFC yields a pixel size between 0.33 and 0.5 µm at 40× magnification, such that 10 µm cells would have a width of 20-30 pixels. Although the input dimensions of 48x48 pixels are sufficient for many phenotypes, it may not be sufficient for others, and high-resolution microscopy may be required. As with all machine and deep learning protocols, the lack of annotated data may be an obstacle in the quest to achieve a trained network capable of accurate classification. Although Deepometry accepts any 3D tensor array inputs (image width x image height x channels) it delivers more meaningful results when the inputs are isolated images of individual objects, i.e. one centered/prominent object in a bounding-box frame. Large images with multiple objects are not ideal, as the object segmentation is not included. One alternative approach is to perform object detection/segmentation, for example using image analysis software such as CellProfiler 4.0 and save the bounding-boxed objects as individual images for input to Deepometry.

Deepometry uses the ResNet50 architecture because of its practical implementation (reasonable model complexity), efficiency and robustness (delivering reasonable results for a wide range of applications) allowing researchers to focus on answering the biological questions at hand rather than finetuning the architecture to achieve the absolute highest accuracy. While advanced users will find it relatively easy to swap alternate networks in the codebase, there is no option to change or alter the deep learning network in the current GUI. Likewise, the use of data augmentation techniques is not facilitated within the application although relatively straightforward to modify in code.

When using image files in a manufacturer's proprietary file format, for example, a .cif file from the ImageStream imaging flow cytometer, the individual image files are extracted and subsequently stored on the host computer hard drive. A typical .cif file can contain 10^5-10^6 single cells with 12 channels and therefore Deepometry needs to store 10^6-10^7 individual .tiff images or 10^5-10^6 .npy/.mat files on the local drive.

Experimental design

Supervised learning design

To perform supervised deep learning requires annotated cell images for each phenotype. A suitable annotation tool can make this often-onerous task easier. For example, the ImageStream analysis software IDEAS has an annotation tool that allows the user to tag multiple phenotypes and save these as separate .cif files which can be input directly into Deepometry. For microscopy data, a tool such as *CellProfiler* [16] can be used to segment the individual cells and *CellProfiler Analyst* can be used to tag individual cells with a phenotype label. For certain applications, for example, label-free classification, a fluorescent biomarker can indicate the phenotypic classes but is excluded from the training process. The strategy of weakly supervised learning described in this protocol can also remove the need to annotate individual cell phenotypes.

If there is no avoiding the use of expert-annotated cell phenotype images, then typically we find that the Deepometry network becomes more accurate when in excess of 100 images per phenotype are used. However, when low numbers of annotated images are used the network becomes very susceptible to over-fitting [40]. ResNet uses drop-out regularization however this alone cannot guarantee over-fitting and the most robust test is using hold-out data which unfortunately also needs to be annotated. As with all machine and deep learning algorithms, an imbalance in the number of annotated images for each phenotype can cause problems in training. Deepometry compensates for this class imbalance in the softmax layer, however, this is no substitute for maximising the number of examples of rarer phenotypes if possible. If lower numbers of training images are unavoidable then the use of cross-validation might be advisable, however, care must be taken to ensure your cross-validation strategy matches the conditions of any future test data [41]. There is a range of cross-validation strategies [41] that can be used when training deep learning algorithms and rather than implicitly incorporating all these techniques we have included the option to randomly partition the image data into training and hold-out sets. However, if there is a class imbalance (more examples of one class versus the others), the user should manually predefine a test and hold-out set.

As with any supervised machine learning algorithm, poorly annotated data causes a significant drop in the accuracy of the network prediction. If there is uncertainty about the phenotype of a significant number of images, then one might introduce an un-scorable category. Optimal results will be achieved when the training and test images are acquired using identical experimental conditions e.g. using the same dyes, same laser excitation, magnification etc. However, it may be helpful to intentionally train a network that is robust to these experimental variations. Often combining data from different experimental conditions as a training set can produce a more generalisable model, although the user must be aware of any batch effects that

might be introduced. We have optimised the training parameters for the deep learning network based on previous studies [21,23,25], however, these may need to be modified to achieve high accuracy, as discussed later in the procedure.

Weakly supervised learning design

The success of a supervised model is largely dependent on the availability of hand-labeled training examples. It is however expensive and time-consuming to create such an annotated database, especially when domain expertise is required, and it is clearly not ideal in high-throughput analysis such as imaging cytometry, where there are generally thousands of objects per class.

When a complete ground-truth annotation is not immediately available, weakly supervised learning is a plausible and rapid alternative for learning feature representations, in which a model can still be trained using easy-to-collect labels that describe the experimental organization of the images. In high-throughput imaging cytometry, for instance, while obtaining a manually assigned phenotypic label for every cell might not be possible, we can still allow a neural network to learn the generic features that represent cells under the same phenotype, based on common metadata such as treatment names, dates of experiments, etc.

When choosing a supervised model, the bias and variance could be addressed by observing the learning curve. When a weakly supervised learning design is used, the error rate for the auxiliary task might not matter as much as how useful the learned features could be, and therefore the observation for bias and variance is not obvious. The best practice, in this case, is to increase the size of the training set, including as many learning examples as possible.

Materials

Equipment

Imaging flow cytometer and companion software:

- In this work, we used image data from the ImageStream X Mark II (Luminex). In general, any high-throughput cytometer that can generate image data of single cells would suffice, although customized code to extract images in a suitable format may be needed. In our case, the INSPIRE acquisition software generates IFC data in the form of a raw image file (.RIF file), which is then loaded into IDEAS (v6.0 or later https://www.luminexcorp.com/imaging-flow-cytometry-support/) for pre-processing. A small dataset for testing Deepometry functionality, containing annotated images of Red Blood Cells is publicly available at https://figshare.com/articles/software/Expert_Annotated_RBC/13053968

Computing system for deep learning workflow:

- Although all steps of the workflow including data preparation, training deep learning classifiers, and visualization can be effectively done by a typical laptop or desktop's CPU (see Table 1), training the classifier on a GPU will significantly reduce training time. We thus suggest operating the workflow on a computer equipped with a modern CUDA-compatible graphics card if possible; a 1x NVIDIA Titan X GPU would suffice. Alternatively, if sufficient expertise is available, a cloud computing resource could also be used, for instance, a high-performance server or a cloud-computing platform such as Amazon AWS.

- The models are trained using single-precision floating point (32-bits), which generally suffices to obtain an accurate classification. Modern CPUs and GPUs can handle double precision (64-bits) but this is not recommended as it has not been shown to improve classification accuracy, and doubles the memory requirements. Required storage space highly depends on the number of objects (cells) collected by IFC, and can range from megabytes (MB) for dozens of objects to gigabytes (GB) for hundreds of thousands of objects. Subsequent steps of the workflow will generate more secondary data and require further storage space, most importantly after data pre-processing. Typically, an IFC experiment of 1 million cells would need ~5 GB disk capacity for initial storage, ~5 GB for pre-processed images produced by IDEAS, and ~10 GB for all downstream analysis steps of the workflow.

Software

CRITICAL The key motivation in developing this workflow was to make it user-friendly so that those with little coding experience could employ deep learning for phenotype classification and unsupervised data exploration. The protocol describes in detail the installation steps for both the Python (Step 1 Option A) and MATLAB versions (Step 1 Options B and C). For the Python version, the instructions include installing all necessary dependencies to accelerate installation and setup. Likewise, MATLAB uses the "Deep learning toolbox". We also provide a stand-alone executable version of Deepometry written in MATLAB for Windows (GPU compatible) which can be installed without any programming language requirements (Step 1 Option C). For those wishing to customize or explore individual modules, the following packages are used.

Essential packages for Python environment (see more details and download sites in **Supplementary note 1**, Python Installation guide):

- Python 3.6
- Tensorflow-gpu 1.9.0
- Keras 2.1.5
- Numpy 1.18.1
- Scipy 1.4.1
- Keras-resnet 0.0.7
- Java Development Kit 8.0/11.0
- Python-bioformats 1.5.2
- Jupyter notebook

Essential packages for MATLAB environment (see more details and download sites in **Supplementary note 2**, MATLAB Installation guide):

- Image processing toolbox
- Deep learning toolbox

For the stand-alone application the details and download sites are given in **Supplementary note 3**, Stand-alone installation guide.

Equipment setup

Image acquisition:

Images should be acquired with the maximum resolution possible, and at least 8-bit depth. Each channel should be captured as an individual grayscale image. No further pre-processing should be performed on the images before beginning this protocol. For common cellular objects (mammalian cells, yeast, bacteria etc.), objective magnification is recommended to be at least 40×, together with high-resolution pixel size (at least 0.5 µm per pixel). A single object should be contained comfortably within a bounded box with dimensions at least 30 x 30 pixels (such that each object has a sufficient number of pixels representing it) but less than 128 x 128 pixels (such that objects can be fed directly into the networks, although larger images can be down-sampled). Usually, high-throughput imaging flow cytometers capture all fluorescence channels simultaneously, however, in other imaging modalities the order in which the channels are imaged may impact the likelihood of photobleaching.

The usual recommendations for any mode of fluorescence microscopy apply in order to obtain high-quality images. The fluorescence dyes should be selected to minimise channel bleed-through and also compensation (using the necessary controls) should be carried out to correct for any bleed-through that might occur. Also, signals generated by different excitation and illumination paths should be calibrated well to avoid misalignment, following the manufacturer's instructions. Likewise, autofocusing should be carefully calibrated, and out-of-focus cells should be excluded using the image acquisition software. For example, using IDEAS software for ImageStream data, there are built-in procedures for gating out cells that are out of focus and also removing debris and multiple cells stuck together. If phenotypes will be manually annotated, IDEAS' annotation tool allows tagging multiple phenotypes and saving them as separate .cif files which can be input directly into Deepometry. All these steps are detailed in the user manual for analysing ImageStream data using IDEAS, which can be downloaded from https://www.luminexcorp.com/download/amnis-ideas-software-user-manual/.

For manually annotating microscopy data, a tool such as *CellProfiler* [16] can be used to segment the individual cells and output a *properties file*. This file allows *CellProfiler Analyst* to be used to view individual cells and tag them with phenotype labels, as described in the manual https://cellprofiler-manual.s3.amazonaws.com/CellProfiler-Analyst-2.2.1/5_classifier.html

For exporting cropped single-cell images from microscopy data, the ***MeasureObjectSizeShape*** module in CellProfiler can be used according to its manual https://cellprofiler-manual.s3.amazonaws.com/CellProfiler-4.0.5/index.html. The module exports each cell's X and Y coordinates together with the bounding box coordinates of the rectangle which encloses that particular cell. This information can then be used to generate single-cell images from the wide-field microscopy files using custom scripts.

Procedure

CRITICAL: Step 1 guides users through the installation of the software and packages required to run Deepometry (Python/MATLAB) or install the stand-alone application (MATLAB). The application of Deepometry to image data analysis starts with step 2 and is

compatible with both the Python and MATLAB versions.

1. For installation of the Python version, follow Option A. For installation of the MATLAB version, follow Option B. For installation of the MATLAB executable version (Windows only), follow Option C.

Option A: Installation – Python version
TIMING: ~1 hour varies according to internet download speed

CRITICAL: Check the GitHub webpage for any updates to the Python version of this Deepometry protocol https://github.com/broadinstitute/deepometry

CRITICAL: The following guide is for **Windows 10 64-bit** users. **UNIX users (Linux or Mac OS)** can skip to step vi (or step vii if **Anaconda** is preinstalled). For a repeat of these steps including screencasts to aid installation and software version recommendations see Supplementary Note 1.

i. Install **Microsoft Visual Studio 2019** (**Community** version is free): https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&rel=16. Choose **Desktop Development with C++.**

ii. Install **Microsoft Visual Studio Build Tools:** https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=16. Choose **C++ build tools.**

iii. Install **Java SE Development Kit 11** https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html.

iv. Once **Java SE Development Kit 11** is installed, set **JAVA_HOME** variables: Open file explorer window, right-click on the '**This PC**' option and select '**Properties**' from the drop-down menu.

The control panel will pop up as a separate window. Select '**Advanced system settings'** from the list appearing at the left of the window.

Select '**Environment Variables**', appearing as a button at the bottom of the window. This will cause a new window to appear. Click the '**New**' button option at the bottom of this window.

A new window allowing the user to specify details for a new System Variable will now appear, select the '**Browse Directory**' option at the bottom of this window followed by '**Java**' then '**jdk-11**' from the list that appears. Select '**OK**' to create this new variable.

v. Select '**Path**' from the second list of '**System variables**' at the bottom half of the window. Click the '**Edit**' button at the bottom of this window. A new window will appear. Type in '**%JAVA_HOME%\bin**' from the list of environment variables provided.

Click '**OK**' to apply changes and close all opened windows.

vi. Install Ana**onda**

- For **WINDOWS,** visit https://repo.anaconda.com/archive/Anaconda3-2019.03-Windows-x86_64.exe. Once installed, right-click the '**Anaconda Navigator**' icon, choose '**Run as administrator**'. It may take a while for Anaconda to fully launch for the first time.

- For **MacOSX**, visit https://docs.anaconda.com/anaconda/install/mac-os/ to install Anaconda. We recommend using the graphical installer, as detailed on the Anaconda website.

  MacOSX users are also advised to install JDK8, using the commands:

```
/usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew tap homebrew/cask-versions
brew update
brew cask install adoptopenjdk8
```

Once installed, run the Anaconda Navigator from the Applications folder.

vii. Create a new environment with **Python 3.6** using the name of your choice in the Anaconda Navigator. For example:

```
conda create --name Deepometry python=3.6
```

Details on creating an Anaconda environment can be found in the Anaconda documentation at this link: https://docs.anaconda.com/anaconda/navigator/tutorials/manage-environments/.

viii. Once the environment is fully created, launch it in a command line terminal window:

```
conda activate Deepometry
```

or use the Anaconda prompt or by using Anaconda Navigator (Click the right arrow next to its name to open a terminal).

ix. Inside the environment terminal, type the following sequence of installation packages. Choose "Yes" when prompted:

```
conda install tensorflow-gpu==1.9.0
```

CRITICAL: if your device does not have a compatible GPU for deep learning, you should only install the CPU-only Tensorflow version by the command:

```
conda install tensorflow==1.9.0
```

Once Tensorflow is installed, continue to install other required libraries for Deepometry:

```
conda install keras==2.1.5 numpy==1.18.1 scipy==1.4.1
pandas==1.0.3 scikit-learn==0.22.1 scikit-image==0.16.2

conda install git jupyter==1.0.0 ipython==7.13.0 seaborn==0.10.1
flask==1.1.2 wtforms==2.3.3 jinja2==2.11.2

conda install -c anaconda openjdk

pip install keras-resnet==0.0.7 opencv-python==4.2.0.34
javabridge==1.0.19 python-bioformats==1.5.2

git clone https://github.com/broadinstitute/deepometry.git

cd deepometry

pip install .
```

x. ***Opening Deepometry GUI using Python (Steps x-xi):*** For a graphical user interface of Deepometry, run the following commands in the environment terminal:

```
git checkout GUI

python Deepometry_GUI.py
```

A HTTP address will be given to run the Deepometry local web app, e.g. http://127.0.0.1:5000 or http://localhost:5000

xi. Open a web browser and type in the address given above. A web app (see **Fig. 3**) will appear for executing deep learning operations (this is a local server running on your own device; no data is sent out to the internet).

Option B: Installation – MATLAB version

TIMING: ~45 minutes, may vary according to internet download speed

CRITICAL: Check the GitHub webpage for any updates to the MATLAB version of this Deepometry protocol https://github.com/ClaireBarnes197/Deepometry_MATLAB_GUI

CRITICAL: Deepometry requires a minimum of MATLAB R2017b, together with the following toolboxes which may be obtained from your MATLAB provider, with the appropriate licenses:

- Statistics and Machine Learning toolbox. https://mathworks.com/products/statistics.html
- Deep learning toolbox. https://mathworks.com/products/deep-learning.html

i. Deepometry makes use of the MATLAB version of Bio-Formats, a standalone Java library for reading and writing life science image file formats[42]. (Note: the minimum MATLAB version recommended for this package is R2017b). Download and unzip the MATLAB toolbox from the link below. https://www.openmicroscopy.org/bio-formats/downloads/

ii. Download the MATLAB UMAP toolbox[43] from File Exchange (link below) and unzip the folder. This toolbox is required for the visualisation step: https://www.mathworks.com/matlabcentral/fileexchange/71902-uniform-manifold-approximation-and-projection-umap

iii. Add the location of the downloaded UMAP file to the MATLAB search path by typing the following at the command line. e.g.

```
>> addpath '/Downloads/Umap'
```

iv. Execute the command:

```
>> run_umap
```

A window will appear which will take you to Google Drive.

v. Access the folder by clicking 'OK' at the window prompt. At the time of this publication, this folder can be found at https://drive.google.com/drive/folders/1VXj6J0D-Z8qE6rkPIx35FIkcOhNWjnrq

A new window with instructions will appear on the screen. You will be directed to download a new updated version of the UMAP distribution, ***'UMAPDistribution.zip'***.

Download and unzip the file before removing the incomplete version downloaded from File Exchange.

vi. This version of UMAP makes use of a new C++ MEX implementation of stochastic gradient descent, which can be downloaded from the above Google Drive location.

Depending on the operating system being Windows or Mac OS, download the appropriate MEX file, along with the file named **lobpcg.m**:

- For **MacOSX: mexStochasticGradientDescent.mexmaci64**
- For **WINDOWS: mexStochasticGradientDescent.mexw64**

Move the MEX and .m files to the UMAP subfolder within your UmapDistribution.

vii. (Optional) Users may wish to improve the understanding of using various UMAP parameters in MATLAB for downstream exploration. For useful examples and comprehensive documentation, type:

```
>> doc run_umap
```

viii. Download the MATLAB PHATE[44] toolbox from GitHub https://github.com/KrishnaswamyLab/PHATE

ix. Add the location of the PHATE module to MATLAB's search path by typing the following at the command line. e.g.

```
>> addpath '/Downloads/PHATE-master/Matlab'
```

x. ***Opening and running Deepometry GUI using (Steps x-xi):*** Download the **GUI_deepometry.mlapp** file, storing it on the appropriate MATLAB path on your machine, for example, 'C:\Users\Documents\MATLAB\Deepometry'.

xi. Run the app by typing:

```
>> GUI_deepometry
```

or by selecting the file and hitting 'run' at the top of the window. This step will bring up the interactive interface (**Fig. 4**).

Option C. Installation – Executable version for Windows (written using MATLAB)

TIMING: ~30 minutes, may vary according to internet download speed

i. The executable version of the app makes use of the MATLAB Runtime installer (Version R2019b), which will be automatically downloaded and installed upon execution of the **'MyAppinstaller.web.exe'** which can be obtained from Figshare (see **Code Availability**). Download the executable file to your machine and right-click, this will initiate the installation process. This application could take some time to install.

ii. A window will appear summarising the app you are about to install, you should click 'Next' to proceed.

iii. Change the destination of the installation as you wish, before clicking 'Next'.

iv. Choose where to save the MATLAB Runtime by selecting a folder, then click 'Next'.

v. Once all folders have been specified, begin the installation by clicking 'Install'. Installation may take some time depending on download speeds, once complete the GUI app will appear as an executable file in your programs folder.

vi. ***Opening and running Deepometry GUI executable version (Step vi):*** Once the necessary executable file from GitHub is installed on your machine, locate the app in your programs folder and click on this to open and begin to use the Deepometry interactive GUI.

CRITICAL STEP: Deepometry accepts images organized in folders and subfolders. It is highly recommended that folder names contain groupable metadata prefixes, such as "Sample A", "sample_B", "sample C", "class-D", "Class-e" as exemplified in the directory structure in **Fig 5**. Note: many separation characters for prefixes are acceptable (including comma ",", hyphen "-", underscore "_" and empty space) and the prefixes are case-insensitive. This directory structure provides the application with important information about your data. After the raw image inputs have been organised into tagged folders, they can then be fed directly to Deepometry to be pre-processed. Images may be in a number of different formats (.cif, .tif or .tiff formats are accepted).

Pre-processing training data

TIMING: typically 5-15 minutes for 50,000-100,000 cells, may vary according to the size of the data and performance of the computer's CPU and memory.

CRITICAL: Steps 2-21 have been validated for a number of datasets, but each step may need adjustment depending on the application. A small dataset for testing Deepometry functionality, containing annotated images of Red Blood Cells is publicly available at https://figshare.com/articles/software/Expert_Annotated_RBC/13053968

2. Image information from original raw files (.cif or .tiff) needs to be converted into Numpy (.npy) or MATLAB (.mat) arrays. Under the section titled **'Preprocessing' (Fig. 3** and **Fig. 4**), input the appropriate parameters as specified in Box 1.

Box 1 | Reshaping and reformatting data inputs for deep learning operations

Input location – The location where the original single-cell images are stored, e.g. C:/Deepometry/Data/raw

Output location – The location where the processed Numpy (.npy) or MATLAB (.mat) arrays should be stored, e.g. C:/Deepometry/Data/parsed

Frame size – The desired size of the resulting image frames (width or height, not both, in pixel unit), e.g. 48. It is highly recommended to use a frame size that comfortably enclosures a single object-of-interest. For instance, a frame size too small could result in the cropping of a cell, while a very large frame size could excessively increase the run time for deep learning operations. The user can load the .cif files into IDEAS and investigate a single cell to determine the pixel size.

Channels – The channels that should be used for training, specified as an index array, e.g. [1,9,12] in Python GUI (minus the brackets, 1,9,12, when using MATLAB). If a single channel is desired, input an integer without brackets. These numbers correspond directly with the channel numbers shown in the IDEAS software.

Montage size (optional) – This option is only used if the user wants to generate per-channel tiled (stitched) montages, which can be efficiently used for image analysis software, such as CellProfiler. Leave blank or input 0 for no stitching otherwise input the number of images to be included in the square montage e.g. 15 (**Fig. 6**).

Train/Hold-out split (optional) – Consider using this option for hold-out–based cross-validation. Use only when parsing the training datasets. Input a ratio to split the parsed data into Training and Hold-out Cross-Validation subsets. The data is randomly shuffled and then split according to the specified percentages, e.g. if users input '80/20', eighty percent (80%) of the data will be defined as training and 20 percent as hold-out. The resulting training set will be then stored in a subfolder named "Train" and can be used for model training; the resulting hold-out set will be stored in a subfolder named "Hold-out" and can be used to evaluate the performance of a trained model. Leave blank if cross-validation does not need to be applied.

CRITICAL STEP: The .cif file from IDEAS contains channel number 1-6 or 1-12 depending on the number of cameras. The channel numbers to be input into Deepometry must correspond to the channel number in the .cif file which should be checked in the IDEAS software. Only the channels containing relevant information should be included.

3. Once all required fields have been populated, click the **'Parse'** button to execute the parsing of raw images. The resulting Numpy (.npy) or MATLAB (.mat) arrays will then be stored in the structure in **Fig. 5**. ?TROUBLESHOOTING

Training a classification model – supervised learning (common to Python and MATLAB)

TIMING: typically 5-20 hours for 50,000-100,000 cells, may vary according to the size of the data and performance of the computer's processors (CPU or GPU) and memory.

CRITICAL When working toward weakly supervised learning, you will need to follow all steps outlined for supervised learning (see Experimental Design). The difference is that the weakly-supervised approach does not require fully annotated images. You should, however, choose a target classification that is fairly useful based on your data structure. For example, when using a similar directory structure to the recommended format in **Fig. 5**, users may choose target classification "Day…" to learn and extract features related to phenotypic biological degradation associated with time.

4. The parsed data from steps 2-3 can now be used to train a ResNet model in the section titled **'Model training' (Fig. 3** and **Fig. 4).** Input the following information:
   Input location – The location where the pre-processed images can be found, e.g. C:/Deepometry/Data/parsed.
   Output location – The location where the resulting deep learning model is to be stored, e.g. C:/Deepometry/Model.
   - For Python, if a folder location is an input here, a fully trained model will be saved as a .h5 file, named after the number of possible classes and the target category prefix, for example, "model_7-class_categories.h5". If no folder location is provided, the input location specified above will be used.
   - For MATLAB, this step will result in a fully trained model, stored at this location. The model will be named according to the time it was saved and the number of classifications it has been trained to recognise.

   Learning iterations – The number of epochs for a deep learning training session. By default, it is set to 512 (40 in MATLAB), which might take several hours (depends on the size of the training materials and available hardware, especially GPUs).
   Target classification – Once the location of training data ("Input location") is specified, click the **'Retrieve'** button to retrieve a list of trainable classification targets. As mentioned in the critical prerequisite for step 2, Deepometry assumes raw input files and parsed data are organized in folders and subfolders with groupable metadata prefixes, such as "Experiment...", "Day...", "Sample...", or "Class...". Once the list of classification targets is populated, choose from the dropdown menu a target as the learning task for the neural network. For example, the provided sample training dataset available in this protocol contains annotated classes of red blood cell phenotypes. For this type of experiment, you would select the "Class..." category
   Temporary folder (MATLAB only) – This option is used to reorganise your data into subfolders named according to the category labels of the target classification. This temporary folder and its content will be removed after the model has been trained. Specify where the temporary folder should be stored. If you are certain about the categorical labelling scheme, it is recommended that you tick the **'Regroup images'** check box. This will store a permanent folder based on the selected labels for later use.

5. Click the **'Train'** button to initiate the deep learning model training. This operation might take several hours (depends on the size of the training materials and available

hardware, especially GPUs). The MATLAB version plots the training accuracy and loss used to optimise the network in real time. ?TROUBLESHOOTING

Evaluation of a trained model

TIMING: typically, 5-15 minutes for 50,000-100,000 cells, may vary according to the size of the data and performance of the computer's processors (CPU or GPU) and memory.

6. Once the classifier has been trained, evaluate the trained model against a held-out dataset to control overfitting. If the user previously used the option ***Train/Hold-out split*** as described in Box 1 (step 2), skip to step 9, since the parsed data in the "Hold-out" subfolder can be readily used here. If not, collect annotated images with appropriate labels for each cell, similar to that of the training materials, and continue to step 7.

7. As described in Box 1, input the following parameters at the **'Preprocessing'** section according to the evaluation data set.
   Input location – The location where the original single-cell images are stored, e.g. C:/Deepometry/Data/raw.
   Output location - The location where the resulting Numpy (.npy) arrays or MATLAB (.mat) arrays should be stored, e.g. C:/Deepometry/Data/parsed.
   Frame size – The desired size of the images (width or height, not both, in pixels) so that the appropriate amount of resizing may be applied, must be the same setting used for the training dataset (step 2) e.g. 48.
   Channels – The channels that were used for training, specified as an index array, must be the same setting as in step 2 e.g [1,9,12] for Python or 1,9,12 for the MATLAB version. If a single channel is desired, input an integer without brackets.
   Montage size (optional) – Use this option to generate per-channel tiled (stitched) montages for feature extract.
   Train/Hold-out split (optional) – Leave blank as this option is not needed for model evaluation.

8. Click the **'Parse'** button to parse the images. The resulting Numpy (.npy) or MATLAB (.mat) arrays will then be processed and stored in the folder structure similar to that of the training dataset.

9. Once the evaluation set is parsed, the accuracy of the trained model can be assessed by comparing predictions made with the known ground truth for each image in a form of a confusion matrix. Input the following fields into the section titled **'Prediction' (Fig. 3** and **Fig. 4**):
   Input location – The location where the processed Numpy/MATLAB arrays are stored, e.g. C:/Deepometry/Data/parsed.
   Output location – The location where evaluation results should be stored e.g. C:/Deepometry/Results. Results include a confusion matrix, displaying counts of all correctly classified and misclassified images belonging to the various categories, and a classification accuracy report.

Target classification – Similar to step 4, first specify the location of the training images in the **'Model training'** section, then hit the **'Retrieve'** button. The directory structure will be examined and all potential classification schemes that may be used as classification targets will appear as part of the dropdown menu **'Target classification'**. Select the categories to evaluate the trained model. It is important to note that in MATLAB, if at step 4, you had chosen to save the re-grouped data, you may skip the actions of choosing a target classification here, instead specify the location of the regrouped folder in ***Input location***.
Temporary folder (MATLAB only) – Input the place where the reorganised test images are to be stored.

CRITICAL STEP: In Python, users should re-define the input location of the training dataset and re-retrieve the proper list of ***Target classification*** from training materials used in the **'Model training'** section. This is critical to ensure the correct reconstruction of the category range, because only the original training data contain images for all categories the network had been exposed to and thus can be effectively used to reconstruct the correct categorization structure. In MATLAB, if you have chosen to save the folder containing re-organised images according to the desired classification, input the location of this folder and skip the retrieving step.

10. Define the location of the trained model (output from step 5) for the field ***Model location***.
    - For Python, the location of the trained model should either be a folder location that contains your trained model or an exact .h5 or .hdf5 file. If the provided folder location contains more than one model, the latest .h5 or .hdf5 will be loaded. If no folder location is provided, the last checkpoint.hdf5 (in deepometry/data/) will be used.

    - For MATLAB, the location of the trained model should be a folder location that contains your trained model. Should this folder contain more than one model, this step will load the most recently saved network with the appropriate number of training classifications, determined by a search of the training dataset (the location for which should be provided for this step).
      CRITICAL STEP Ensure that trained models are stored and located in a separate subfolder from other .mat files that are being created and stored. This can help avoid unrelated files with similar names being loaded in error.

11. Ensure the box **'Unannotated data?'** is not checked, then click the **'Predict'** button to evaluate the accuracy of the trained model. A confusion matrix will be generated (for example for Python see **Figure 7a**). ?TROUBLESHOOTING

Predicting the classification of unlabelled data
TIMING: typically, 5-15 minutes for 50,000-100,000 cells, may vary according to the size of the data and performance of the computer's processors (CPU or GPU) and memory.

12. A trained model can be used to predict the labels of unlabelled images. In contrast to step 6, here users do not need to collect annotated data. As described in Box 1, input the following fields into the section titled **'Preprocessing' (Fig. 3** and **Fig. 4**):
Input location – The location where the original single-cell images are stored, e.g. C:/Deepometry/Data/raw.
Output location - The location where the resulting Numpy (.npy) arrays or MATLAB (.mat) arrays should be stored, e.g. C:/Deepometry/Data/parsed.
Frame size – The desired size of the resulting image frames (in pixels) so that the appropriate amount of resizing may be applied, must be the same setting used for the training dataset (step 2) e.g. 48.
Channels – The channels that were used for training, specified as an index array, must be the same setting as in step 2 e.g [1,9,12] or 1,9,12 for the MATLAB version. If a single channel is desired, input an integer without brackets.
Montage size (optional) – Use this option to generate per-channel tiled (stitched) montages for feature extract.
Train/Hold-out split (optional) – Leave blank as this option is not needed for prediction.

13. After all specified fields are completed, click the **'Parse'** button to action pre-processing of unlabelled data.

14. Once raw image data has been converted to the correct format, the user may feed images to the trained model for categorical prediction. This step will generate a predicted label for each single-cell image. Fill in the input fields in the section titled **'Prediction'**. Inputting the following information similar to steps 4:
Input location – The location where the processed Numpy arrays (.npy) or MATLAB arrays (.mat) are stored, e.g. C:/Deepometry/Data/parsed.
Output location – This step will generate predicted classifications for each image. This information will be stored as a .csv file and a histogram for predicted categories at the location specified here. For example, *C:/Deepometry/Results.*
Target classification – Similar to step 4, first specify the location of the training images in the 'Model training' section, then hit the **'Retrieve'** button. The directory structure will be examined and all potential classification schemes that may be used as classification targets will appear as part of the dropdown menu **'Target classification'**. Select the desired categories to evaluate the trained model.
Temporary folder (MATLAB only) – Input the place where reorganised test images are to be stored.

CRITICAL STEP: In Python, users should re-define the input location of the training dataset and re-retrieve the proper list of ***Target classification*** from the training materials used in the **'Model training'** section. This is critical to ensure the correct reconstruction of the range of categories, because only the original training data contain images for all categories the network had been exposed to and thus can be used to reconstruct the correct categorization structure. In MATLAB, if you have

chosen to save the folder containing the organised images according to the desired classification, input the location of this folder and skip the retrieving step.

15. Define the location of the training module input for the field ***Model location***.
    - For Python the location of the trained model. Input either a folder location or an exact .h5 or .hdf5 file. If the provided folder location contains more than one model, the latest .h5 or .hdf5 will be loaded. If no folder location is provided, the last checkpoint.hdf5 (in deepometry/data/) will be used.

    - For MATLAB the location of the trained model should be the folder location that contains your model. Should this folder contain more than one model, this step will load the most recently saved model with the appropriate number of training classifications, determined by a search of the training dataset, the location for which should be provided for this step.
      CRITICAL STEP: Ensure that trained models are stored and located in a separate subfolder from other .mat files that are being created and stored. This can help avoid unrelated files with similar names being loaded in error.

16. Ensure the box **'Unannotated data?'** is checked, then click the **'Predict'** button to predict the categories of the unknown single cells in testing data.
    ?TROUBLESHOOTING

Extracting and visualizing deep learning feature embeddings
TIMING: typically 5-15 minutes for 50,000-100,000 cells, may vary according to the size of the data and performance of the computer's processors (CPU or GPU) and memory.

CRITICAL The protocol enables the use of Res4a_ReLU, Res5a_ReLU and pool5 (deeper and closer layer toward the ultimate layer, in that order) within the ResNet50 architecture as feature extractors. In general, the deeper the layer is, the more specified the features are captured, with regard to the assigned tasks. For example, if the task and the ultimate goal are both to distinguish different phenotypes of the cells, pool5 layer will more likely capture morphological features that are specific signatures of the phenotypes. In another example, if the task was to classify the date of the experiment, but the actual biological question was to group cells into clusters of morphological similarity, then users would want to use Res4a_ReLU and Res5a_ReLU to preserve the generic features of the cells.
CRITICAL Features may be extracted from several layers of a trained model. Features can be then stored in a tabular data file (.csv/.tsv/.txt) and used in downstream analysis, including unsupervised clustering or dimension reduction techniques. Features output in this format may also be combined with features generated from other analysis platforms and used for in-depth explorations.

17. Input the following information into the **'Feature extraction'** section:
    Input location – The location where processed Numpy arrays (.npy) or MATLAB arrays (.mat) are stored.
    Output location – Specify where the extracted features and metadata should be

stored.

Target classification (MATLAB) – After specifying the input location of the process MATLAB arrays, hit the **'Retrieve'** button. The directory structure will be examined and all potential classification schemes that may be used as classification targets will appear as part of the dropdown menu **'Target classification'**. This information will be used to color-code all plots so that subpopulations may be easily observed. If you do not know labels you may leave this blank

18. Enter the **Model location**.
    - For Python, input either a folder location or an exact .h5 or .hdf5 file. If the provided folder location contains more than one model, the latest .h5 or .hdf5 will be loaded. The model name must contain the numeric identifier of how many categories the model was trained to classify. Please check whether such a file exists after model training, for example, "model_7-class_categories.h5".

    - For MATLAB, input either a folder location or an exact file. If the provided folder location contains more than one model, the latest model trained to recognise the correct number of classifications will be loaded. Errors occurring at this stage may be due to the fact that the model you wish to use does not exist in the specified location and so this should be verified before running this step.

19. Use the radio buttons to input the name of the layer to be used as the feature extractor:
    - For Python choose one of the following layers from which to extract features: *Res40_ReLU*, *Res5a_ReLU* or *pool5.*

    - For MATLAB choose one of the following layers from which to extract features: *activation_40_ReLU*, *activation_25_ReLU* or *Avg_pool_4_4.*

20. Click the **'Extract'** button to execute this step. ?TROUBLESHOOTING
21. (Optional) To further explore the features:
    - For Python, once feature extraction is done, the output files, such as features_extracted_by_pool5.txt, metadata.tsv can be uploaded to http://projector.tensorflow.org for 2D/3D PCA, t-SNE, and UMAP (**Fig. 7**) visualization of deep learning embeddings. A short video demonstrating the generation of a typical visualisation of features using tensorflow can be found here https://www.youtube.com/watch?v=HZZUDobELJM.

    - For MATLAB basic PCA and tSNE plots are generated by default and stored at the specified output location. The application also employs PHATE, a dimensional reduction tool for visualising high-dimensional information such as that provided by the network. A short tutorial outlining all of the parameters and full capability may be found at the following URL: https://dburkhardt.github.io/tutorial/visualizing_phate/ . Moreover, a basic UMAP plot is also saved; for users interested in further exploring the MATLAB version of UMAP and its functionality a short video discussing some of the input variables of interest may be found here (**Supplementary video 1**). This step will also generate a

.mat and .csv file for further exploration and advanced visualisation of your features.

Troubleshooting

If a section is not properly executed, make sure that all required fields are populated to execute a particular step. If you have not given sufficient information, the module will not proceed. It is also important to ensure that only the required fields for a particular section are populated. Providing unnecessary information can also cause errors, for example, filling fields in unrelated sections. Make sure that the raw data have been properly pre-processed/parsed before actioning any of the steps. Model training, evaluation, prediction and feature extraction sections all require that the raw image data be converted into the correct arrays (.npy or .mat). Ensure that data is in the recommended directory structure given in Figure 5.

To aid the user we have optimised the training parameters used by the neural network. When training a ResNet model with a limited number of objects (<2,000), users may need to consider trying different learning rates (parameter lr=0.0001 by default). This can be found at line 42 of the file ***model.py*** within the ***Deepometry*** folder for the Python codebase, or line 110 of MATLAB's train.m file. Also, the number of epochs used to train the network can be modified to improve accuracy or to avoid overfitting. The choice of these parameters is highly dependent on the application (number of classes, number of images, resolution of the images etc.) and we encourage the user to optimise these as they see fit.

All versions of the GUI were intentionally designed to output error messages providing the user with some guidance, mainly warnings about missing input when trying to execute a step. For additional troubleshooting guidance for the specific steps of the procedure, see Table 2.

Table 2| Troubleshooting Table

Step/Section	Problem	Possible reason	Possible solution
Step 3 (Preprocessing)	This step cannot be executed	Missing input information	Hover over the **'Parse'** button to view an interactive tooltip with guidance on which fields are necessary.
Step 3 (Preprocessing)	When .cif files are being used and this step cannot be executed	Images are in an unreadable format	The .cif file must be one exported from Imaging flow cytometry analysis software IDEAS 6.0 or later, which contains a population of cells/objects gated through IDEAS interface (any manual gate will work). It is important

			to ensure that input files are in one of these formats.
Step 3 (Preprocessing)	(Python only) When .cif files are being used, this step is correctly executed once but in the second time, an error page appears.	Javabridge VM is not correctly called the second time; the function 'javabridge.start_vm' can only be called one time.	This error only occurs in Python version of Deepometry. Users should terminate the app by attending to the command line terminal, hitting Ctrl+C twice, and then restart the app Deepometry_GUI.py
Step 3 (Preprocessing)	When .tif/.tiff files are being used, this step is executed but an error page appears.	Missing input information for .tif images.	It is crucial to ensure .tif/.tiff filenames contains a channel identifier, such as 'Ch1' for channel 1, 'Ch2' for channel 2 etc. as in "Class_SmoothDisc_Ch1.tif". User will also need to specify the channel ID(s) to be parsed as described in Box 1, do not leave blank.
Step 5 (Model training)	This step cannot be executed	Missing input information	Hover over the **'Train'** button to view an interactive tooltip with guidance on which fields are necessary.
Step 5 (Model training)	This step cannot be executed	Input images have not been converted	All images should be converted into Numpy/MATLAB arrays (.npy or .mat) to be accepted by the network. Please use the **'Preprocessing'** section of the protocol to convert your images into the correct formats

Step 5 (Model training)	Training accuracy is lower than expected	It is important that once you have used the **Retrieve** function to determine all possible classification targets, check if you have selected the correct classification, **IMPORTANT:** When working toward Weakly supervised learning, an auxiliary classifier may be chosen. In this case, it is normal that training will result in lower accuracies. At the end of this process, a rich feature set will be extracted from the data.	Chosen another classification target from the dropdown menu that appears once the list has been properly populated.
Step 11 (Prediction)	This step cannot be executed	Missing input information	Hover over the **'Predict'** button to view an interactive tooltip with guidance on which fields are necessary.
Step 11 (Prediction)	This step cannot be executed	Ensure that the same categorical system that was used to train the model is correctly re-used for the evaluation data. Your data may include fewer classifications than your training images. However, the training dataset should present every possible class of targets.	Check that the correct classification system has been selected from the dropdown menu.

Step 16 (Prediction)	This step cannot be executed	As this step can also be used to predict the classifications of *unannotated* data, it may be possible that although you wish to use this step for this purpose you have not made this clear (tick the box) and therefore the workflow is looking for ground truth for each image	If you wish to use this step to predict the classifications of unlabelled data you should make sure that you click the **'Unannotated data?'** checkbox
Step 20 (Feature extraction)	This step cannot be executed	Missing input information	Hover over the **'Extract'** button to view an interactive tooltip with guidance on which fields are necessary.
Step 20 (Feature extraction)	This step cannot be executed	Input images have not been converted	All images should be converted into Numpy/MATLAB arrays (.npy or .mat) to be accepted by the network. Please use the **'Preprocessing'** section of the protocol to convert your images into the correct formats
Step 20 (Feature extraction for **MATLAB** version)	This step cannot be executed	You should Retrieve information for the directory structure of images that you wish to extract features for. For the MATLAB version, this step will not execute if this information is missing (Python version is not affected)	Please check to make sure the correct labeling is applied to the images that you are extracting features from.

Anticipated results

For a critical test of the protocol, we suggest using the dataset used to train Deepometry to classify red blood cells into the progressive morphologies associated with damage to the cells [22]. The dataset which includes approximately 67,400 images including 7 different phenotypes (split into a training, unlabelled and holdout set) is available on Figshare (see **Data Availablity**). This dataset was used to generate the confusion matrix, t-SNE and UMAP plots shown in **Fig. 7**. However, Deepometry can take a significant amount of time to train using this large dataset, especially if using the CPU. Therefore, to simply test the installation of the protocol, we suggest using the much smaller sample of the same dataset available on Figshare (see **Data Availablity**). This dataset includes the same 7 classes as the larger datasets, however with 2958 annotated cells for training and 858 cells for testing. The data is also structured in the same format as described in Figure 5 which will allow direct input into the graphic user interface. The results for this dataset may not be as accurate as in the case when training with the larger dataset. The lower number of images might nevertheless serve well to test the protocol. A typical confusion matrix, t-SNE and PHATE results for this dataset are given in **Supplementary Fig. 1**.

During storage, red blood cells deform from smooth/crenated discs to crenated discoids/spheroids and finally irreversibly to crenated/smooth spheres, a process which reduces oxygen delivery and increased viscosity. Current assessments of the suitability of blood for transfusion are done by manually counting these cell morphologies so the automated classification of red blood cells can eliminate this laborious and subjective process. To train the network a significant number of human curated morphology images must be provided, however using weakly supervised learning we demonstrated[22] using this dataset that we could train Deepometry using an auxiliary classification task which requires easily obtainable labels e.g. the age of a blood sample. The network learns useful features which then serve to label the cells with an index which predicts the deterioration of the cells due to morphology changes.

Timing

The time taken to complete the protocol will be dependent on the dataset size (in terms of the number images and number of channels per image) you are using for training. We have estimated the time taken to run each step of the protocol training on the data used to generate the outputs in **Fig. 7** and this is provided in **Table 1** with further detail in **Supplementary Note 4**.

Acknowledgements

This work was supported by the BBSRC (BB/P026818/1 to PR), the National Science Foundation (DBI 1458626 to AEC), and the National Institutes of Health (R35 GM122547 to AEC). Many thanks to Pearl Ryder for testing and confirming the functionality of the protocol.

Competing interests

All authors declare that they have no competing interests as defined by Nature Research, or other interests that might be perceived to influence the interpretation of the article.

Author Contributions Statement

M.D. contributed to software development, design and interpretation of results, manuscript writing. C.B. contributed to the software development, design and interpretation of results, manuscript writing. J.C.C contributed to software development, design and interpretation of results. C.M. and A.G. contributed to software development. A.E.C. contributed to design and interpretation of results, manuscript writing. P.R. contributed to software development, design and interpretation of results, manuscript writing.

Data availability

The full dataset for Annotated images of different phenotypes of red blood cells is publicly available at https://figshare.com/articles/URL7_Annotated_Data/12432506 . Deposited 6 May 2020. The smaller subset for testing Deepometry functionality, containing annotated images of Red Blood Cells is publicly available at https://figshare.com/articles/software/Expert_Annotated_RBC/13053968 . Deposited 9th October 2020.

Code availability

The codebase for Deepometry (Python) is publicly accessible at https://github.com/broadinstitute/deepometry , under BSD 3-Clause License, Broad Institute.
The codebase for Deepometry (Python) is publicly accessible at https://github.com/ClaireBarnes197/Deepometry_MATLAB_GUI , under BSD 3-Clause License, Broad Institute.
The stand-alone MATLAB app is publicly available at https://doi.org/10.6084/m9.figshare.13082231 deposited 18th December, 2020.

Related links

Key references using this protocol:

- Doan M, et al. Proc Natl Acad Sci U S A. 2020;117(35):21381-21390. https://doi.org/10.1073/pnas.2001227117
- Doan M, et al. Cytometry A. 2020;97(4):407-414. https://doi.org/10.1002/cyto.a.23987

References

1. Basiji, D. A., Ortyn, W. E., Liang, L., Venkatachalam, V. & Morrissey, P. Cellular Image Analysis and Imaging by Flow Cytometry. *Clin. Lab. Med*. **27**, 653-670 (2007).
2. Chang, S., Serena, K., Karen, S. & Gyongyi, S. Impaired expression and function of toll-like receptor 7 in hepatitis C virus infection in human hepatoma cells. *Hepatol*. **51**, 35-42 (2009).
3. Maguire, O., Collins, C., O'Loughlin, K., Miecznikowski, H. & H. Minderman. Quantifying nuclear p65 as a parameter for NF-κB activation: Correlation between ImageStream cytometry, microscopy, and Western blot. *Cytometry A*. **79**, 461-469 (2011).
4. Bourton, E. C. *et al.* Multispectral imaging flow cytometry reveals distinct frequencies of γ-H2AX foci induction in DNA double strand break repair defective human cell lines. *Cytometry A*. **81**, 130-137 (2012).
5. Begum, J. *et al.* A method for evaluating the use of fluorescent dyes to track proliferation in cell lines by dye dilution. *Cytometry A*. **83**, 1085-1095 (2013).
6. de la Calle, C., Joubert, P.-E., Law, H.K.W., Hasan, M., & Albert, M.L. Simultaneous assessment of autophagy and apoptosis using multispectral imaging cytometry. *Autophagy* **7**, 1045-1051 (2011).
7. Filby, A. *et al.* An imaging flow cytometric method for measuring cell division history and molecular symmetry during mitosis. *Cytometry A.* **79**, 496-506 (2011).
8. Riordon J. *et. al.* Deep Learning with Microfluidics for Biotechnology. *Trends in Biotechnology* **37**, 310-324 (2019).
9. Isozaki, A. *et al.* AI on a chip. *Lab on a chip* **17**, 3074-3090 (2020).
10. Heath, J.R., Ribas, A. & Mischel, P.S., Single-cell analysis tools for drug discovery and development. *Nat. Rev. Drug Discov.* **15**, 204-216 (2016).
11. Sommer, C. & Gerlich, D.W. Machine learning in cell biology – teaching computers to recognize phenotypes. *J. Cell Sci.* **126**, 1-11 (2013).
12. Caicedo, J. *et al.* Data-analysis strategies for image-based cell profiling. *Nat. Methods.* **14**, 849-863 (2017).
13. Lee, K.C.M. *et. al.* Quantitative Phase Imaging Flow Cytometry for Ultra-Large-Scale Single-Cell Biophysical Phenotyping. *Cytometry A.* **95**, 510-520 (2019).
14. Kumamoto, Y. *et. al.* High-Throughput Cell Imaging and Classification by Narrowband and Low-Spectral-Resolution Raman Microscopy. *J. Phys. Chem. B.* **123**, 12, 2654–2661 (2019).
15. Hennig, H. *et al.* An open-source solution for advanced imaging flow cytometry data analysis using machine learning. *Methods.* **1**, 201-210 (2017).
16. Jones, T. R. *et al.* Cell Profiler Analyst: data exploration and analysis software for complex image-based screens. *BMC Bioinformatics*. **9**, 482 (2008).
17. Blasi, T. *et al.* Label-free cell cycle analysis for high-throughput imaging flow cytometry. *Nat. Commun*. **7**, 10256 (2016).
18. Eulenberg, P. *et al.* Reconstructing cell cycle and disease progression using deep learning. *Nat. Commun.* **8**, 463 (2017).
19. van der Maaten, L. & Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008).
20. Haghverdi, L., Buettner, F. & Theis, F. J. Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics.* **15**, 2989-98 (2015).
21. Becht, E. *et al.* Dimensionality reduction for visualizing single-cell data using UMAP. *Nat. Biotechnol.* **37**, 38-44 (2019).

22. Doan, M. *et al.* Objective assessment of stored blood quality by deep learning. *PNAS.* **117**, 21381-21390 (2020).
23. Hennig, H. *et al.* An open-source solution for advanced imaging flow cytometry data analysis using machine learning. *Methods.* **112**, 201-210 (2017).
24. Szegedy, C. *et al.* Going deeper with convolutions. *Proceedings of the IEEE Conference on CVPR*, 1–9 (2015).
25. Doan, M. *et al.* Label-Free Leukemia Monitoring by Computer Vision. Cytometry A. **97**, 407-414 (2020).
26. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on CVPR*, 770-778 (2016).
27. Nitta, N. *et al.*, Intelligent Image-Activated Cell Sorting, *Cell* **175**, 266-276 (2018).
28. Ota, S. *et al.* Ghost Cytometry, *Science* **360**, 1246-1251 (2018).
29. Mikami, H. *et al.* Virtual-freezing fluorescence imaging flow cytometry. *Nat Commun.* **11**, 1162 (2020).
30. Dang, V.Q. *et al.* Methods for Segmentation and Classification of Digital Microscopy Tissue Images, *Frontiers in Bioengineering and Biotechnology* **7**, (2019)
31. Moen, E. *et al.* Deep learning for cellular image analysis. *Nat Methods* **16**, 1233–1246 (2019).
32. Nassar M., et al. Label-Free Identification of White Blood Cells Using Machine Learning. *Cytometry A* 95, 836-842 (2019).
33. Lippeveld, M., et. al. Classification of Human White Blood Cells Using Machine Learning for Stain-Free Imaging Flow Cytometry. *Cytometry A.* **97**, 308-319. (2020).
34. Dickson M. D. S. et. al., Deep-learning-assisted biophysical imaging cytometry at massive throughput delineates cell population heterogeneity, *Lab Chip* **20**, 3696-3708 (2020).
35. Nissim, N., Dudale, M., Barnea, I and Shaked, N.T. Real-Time Stain-Free Classification of Cancer Cells and Blood Cells Using Interferometric Phase Microscopy and Machine Learning. Cytometry. https://doi.org/10.1002/cyto.a.24227 (2021)
36. Ugele, M., et. al., Label-Free High-Throughput Leukemia Detection by Holographic Microscopy, *Adv. Sci.* **5**, 1800761 (2018).
37. Karandikar *et. al.*, Deep learning approach: Reagent-Free and Rapid Assessment of T Cell Activation State Using Diffraction Phase Microscopy and Deep Learning, *Anal Chem.* **5**, 3405–3411 (2019).
38. Ayyappan, V. *et.al.*, Classical machine learning based on manual feature extraction: Identification and Staging of B-Cell Acute Lymphoblastic Leukemia Using Quantitative Phase Imaging and Machine Learning, *ACS Sens*. **5**, 3281–3289 (2020).
39. M. Kräter *et al.*, AIDeveloper: deep learning image classification in life science and beyond Preprint at bioRxiv, p.2020.03.03.975250 (2020).
40. Teschendorff, A.E. *et al.*, Avoiding common pitfalls in machine learning omic data science. *Nat. Mater.* **18**, 422–427 (2019).
41. Saeb, S., Lonini,L., Jayaraman, A., Mohr, D.C. and Kording, K.P., The need to approximate the use-case in clinical machine learning, *GigaScience* **6**, (2017).
42. Linkert, M. *et.al.*, Metadata matters: access to image data in the real world. *J Cell Biol.* **189**, 777–782 (2010)

43. Meehan, C., Ebrahimian, J., Moore, W. and Meehan, S., Uniform Manifold Approximation and Projection (UMAP), MATLAB Central File Exchange (2021).

44. Moon, K. R., et. al. Visualizing structure and transitions in high-dimensional biological data. *Nature biotechnology* **37** 1482-1492 (2019).

Figure captions

Figure 1 | A comparison of Deepometry protocol with traditional machine learning approaches. The top half of the flowchart represents typical steps executed by traditional analysis, these include the use of commercial software to segment objects-of-interest, generate features, feature reduction and classical machine learning. The bottom half (orange arrows) represents the protocol presented here: the deep learning network accepts inputs in multiple image file formats (.cif, .tiff) and during training, the network simultaneously performs the usual steps from the machine learning protocol. The workflow is significantly simplified and involves minimal user interaction. The side of the representative image displayed here is 48x48 pixel (at 40× magnification, pixel resolution 0.5 μm).

Figure 2 | The overall workflow of Deepometry procedure. Step 1 (not shown here) guides users through the installation of the software and packages required to run Deepometry (Python/MATLAB, Options A/B) or install the stand-alone application (MATLAB, Option C). The application of Deepometry to image data analysis starts with step 2. Steps 2-3, 6-8, 12-13 are preprocessing actions for the training set, validation set, and testing set, respectively, served to transform raw input images to data types and shapes appropriate for deep learning operations. Steps 4-5 are model training actions (highlighted in red). Steps 9-11 and 14-16 are predicting mechanisms for annotated data (highlighted in cyan) and unannotated data (highlighted in purple), respectively. Steps 17-21 are used to extract deep learning feature embeddings for dimension reduction and data exploration.

Figure 3 | A screenshot of Deepometry graphical user interface (Python version).

Figure 4 | A screenshot of Deepometry graphical user interface (MATLAB version).

Figure 5 | Example data structure for use in Deepometry. (a) and **(b)** Recommended data structure. Deepometry is able to process single-cell images in a number of formats (.cif, .tif, and .tiff are acceptable). **(c)** Deepometry takes raw image data and transforms and stores this information into Numpy (.npy) or MATLAB (.mat) arrays, organized into categories mirroring the input folder structure.

Figure 6 | A typical montage image generated by Deepometry. An example of a 15x15 montage output by the software interface, where each tile is 48x48 pixels, generated when selecting "Montage size" as 15 in the **'Preprocessing'** section of the GUI and "Frame size" as "48". This montage image can be readily used for regular image analytic pipelines, for example using CellProfiler, to generate features for input into traditional machine learning techniques for classification and visualisation (upper panel, blue path of **Fig. 1**). The pixel resolution of each single-cell image displayed here is 0.5 μm at 40× magnification.

Figure 7 | Typical outputs of the Deepometry protocol (a) An example of a confusion plot, generated by the Deepometry for the red blood cell data [17]. Correctly classified images are represented by the diagonal of the plot and those images misclassified by the fully trained model sit off-diagonal. Single-cell feature embeddings extracted by the penultimate layer (pool5) of a trained ResNet50 were used as morphological parameters and can be projected on routine t-SNE **(b)** or UMAP **(c)**. Here a 3-D t-SNE allows the visualization of distinct cell clusters, whereas a UMAP allows the reconstruction of a continuum of morphologies, transitioning from disc-shape to spherical phenotypes.

Tables

<table>
<tr><td></td><td>Deepometry
by CPU</td><td>Deepometry
by GPU</td><td>ResNet50
by CPU</td><td>ResNet50
by GPU</td><td>CPCML
by CPU</td></tr>
<tr><td>Data digestion</td><td>18m:00s</td><td>3m:49s</td><td>25m:00s</td><td>3m:49s</td><td>25m:40s</td></tr>
<tr><td>Feature extraction</td><td rowspan="2">30h:20m:18s</td><td rowspan="2">47m:11s</td><td rowspan="2">7d:19h:35m:14s</td><td rowspan="2">04h:26m:25s</td><td>15m:00s</td></tr>
<tr><td>Classification</td><td>04h:00m:00s</td></tr>
<tr><td>User’s attendance</td><td>20 minutes</td><td>20 minutes</td><td>20 minutes</td><td>20 minutes</td><td>8 hours</td></tr>
</table>

Table 1| Deepometry processing times This table demonstrates the Deepometry processing times (CPU: Intel i7-3770 3.4 GHz, GPU: 1x NVIDIA Titan) taken to classify the position of cells within their cell cycle using the data given in reference 13 in comparison with the Keras version of ResNet50 and classical machine learning (CML). The classical machine learning protocol generates montages (e.g. **Fig. 6**) of the individual image files and inputs these into CellProfiler (CP) to measure the cell features which are then used to classify the cells using RUS boosting as described in reference 13. The data consisted of TIFF images of 33,060 single cells, each cell is imaged in 2 channels (Bright field and Dark field), each image is cropped/padded to 48x48 pixels.

SUPPLEMENTARY INFORMATION

- Supplementary Video 1: Video guide to using MATLAB UMAP with Deepometry
- Supplementary Figure 1: Typical outputs of the Deepometry for reduced dataset
- Supplementary Note 1: Python Installation guide
- Supplementary Note 2: MATLAB Installation guide
- Supplementary Note 3: Stand-alone application installation guide.
- Supplementary Note 4: Timings.

Figure 1

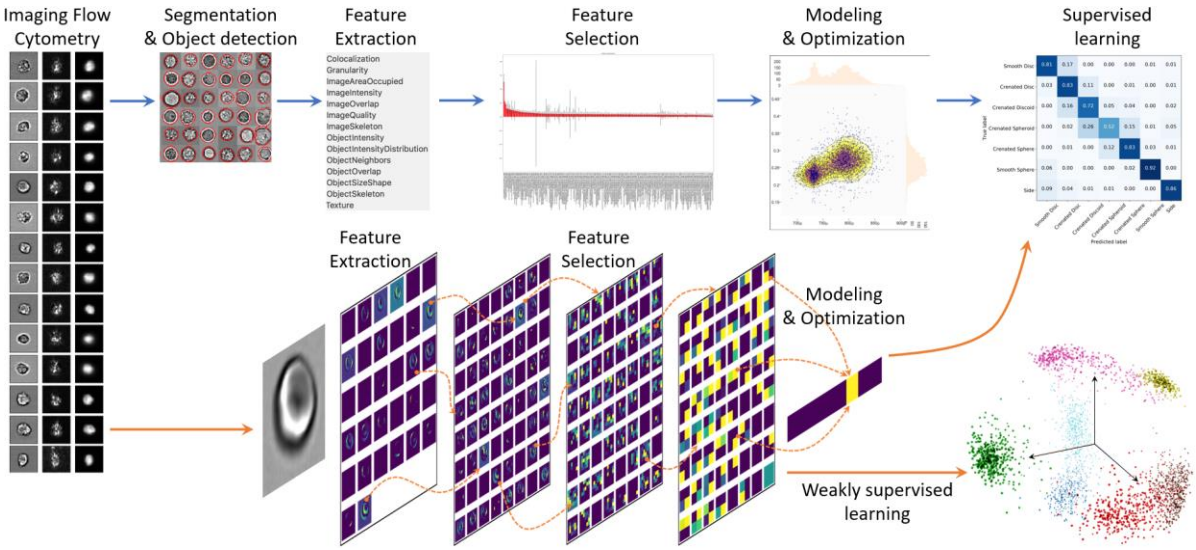


Figure 2

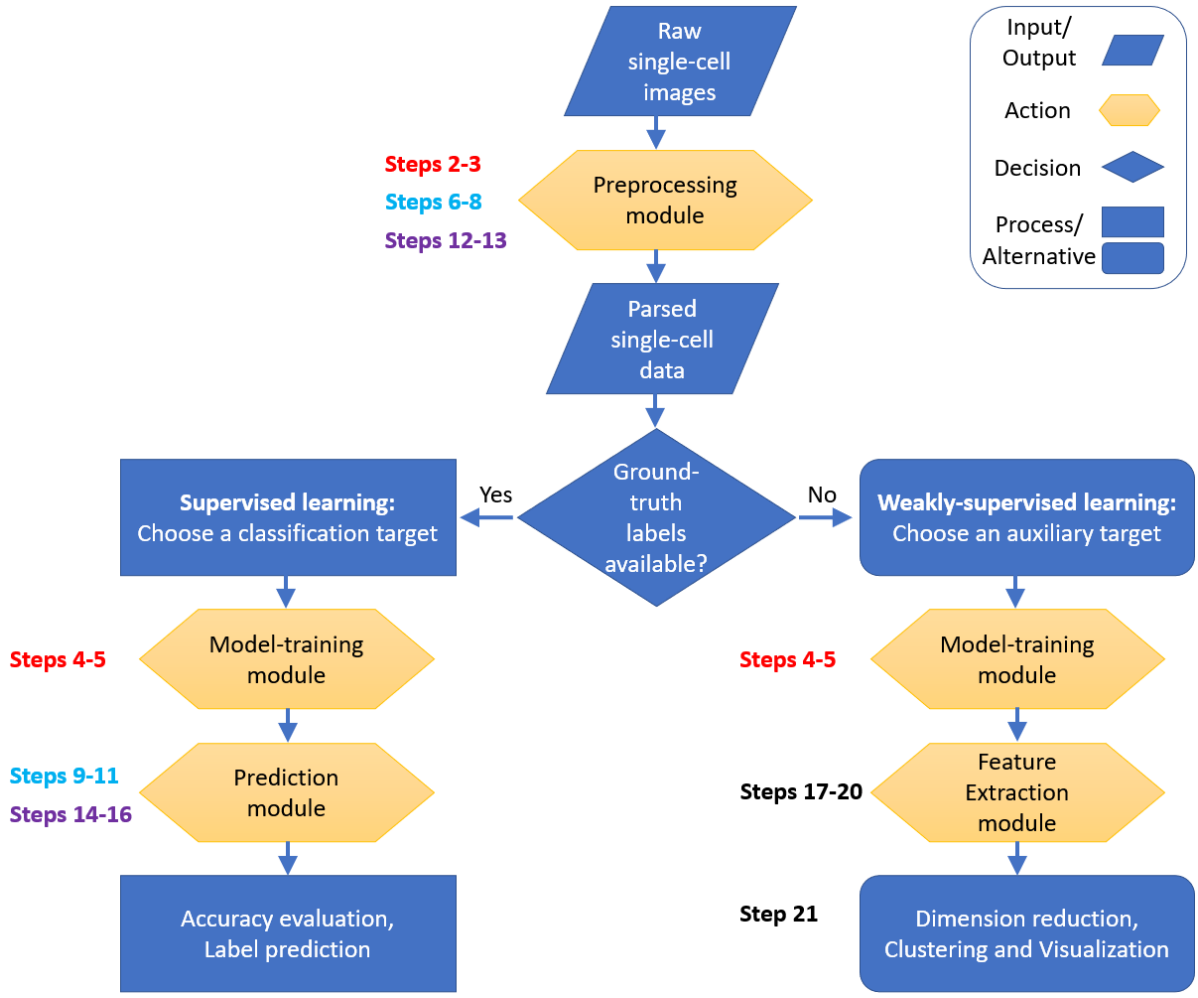


Figure 3

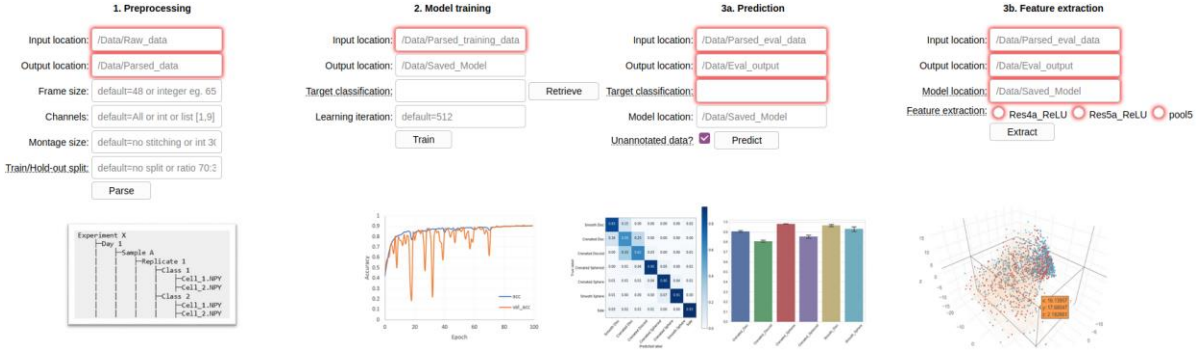


Figure 4

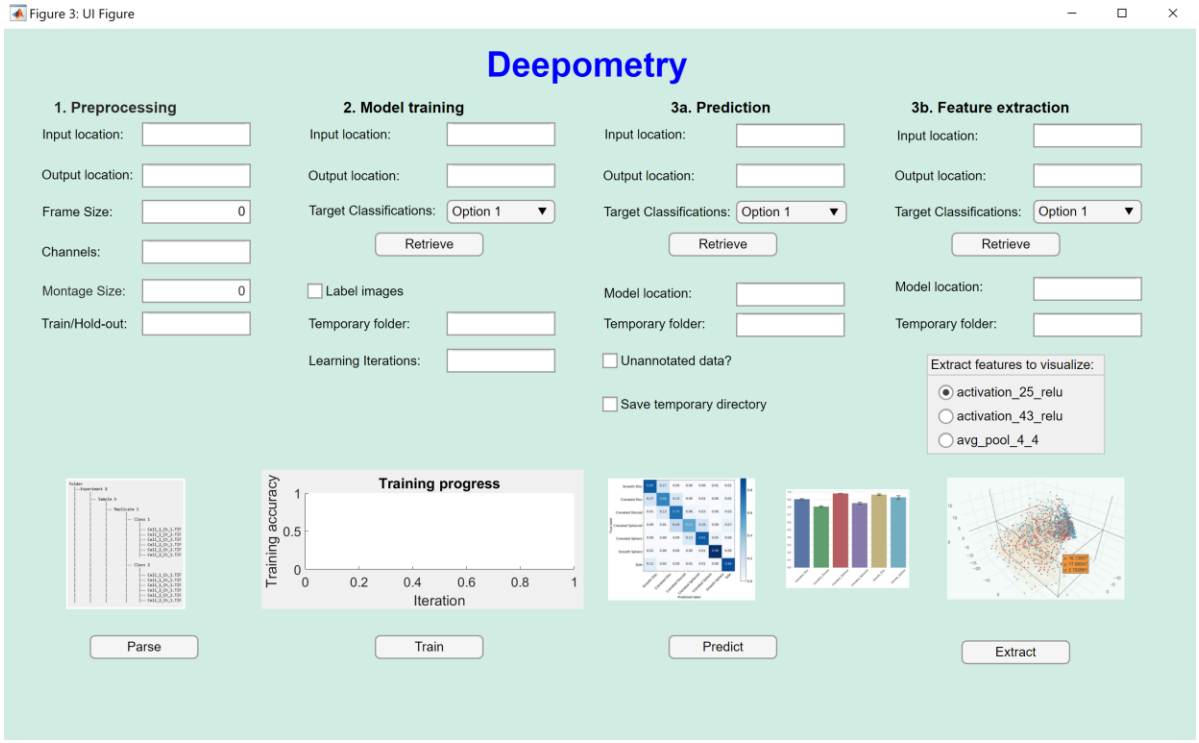


Figure 5

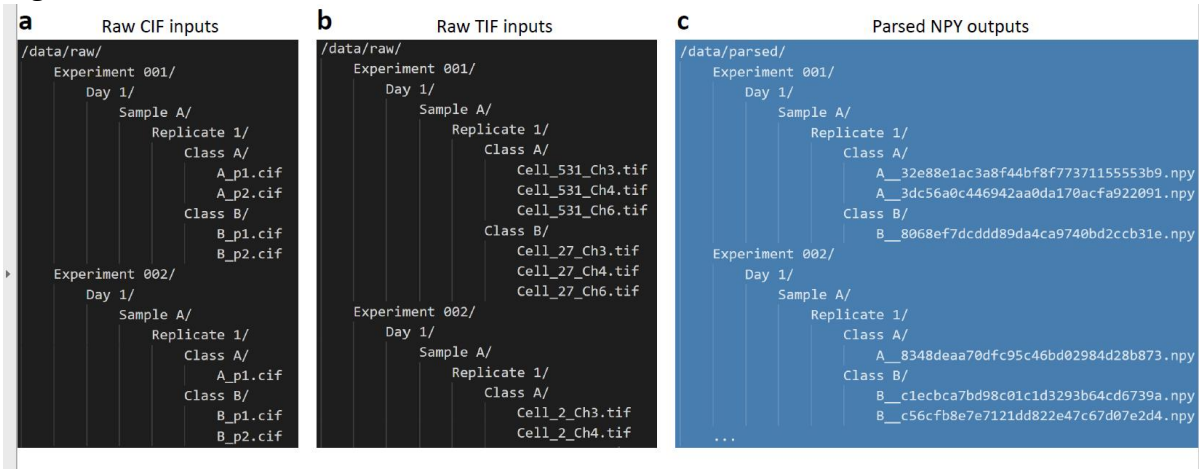


Figure 6

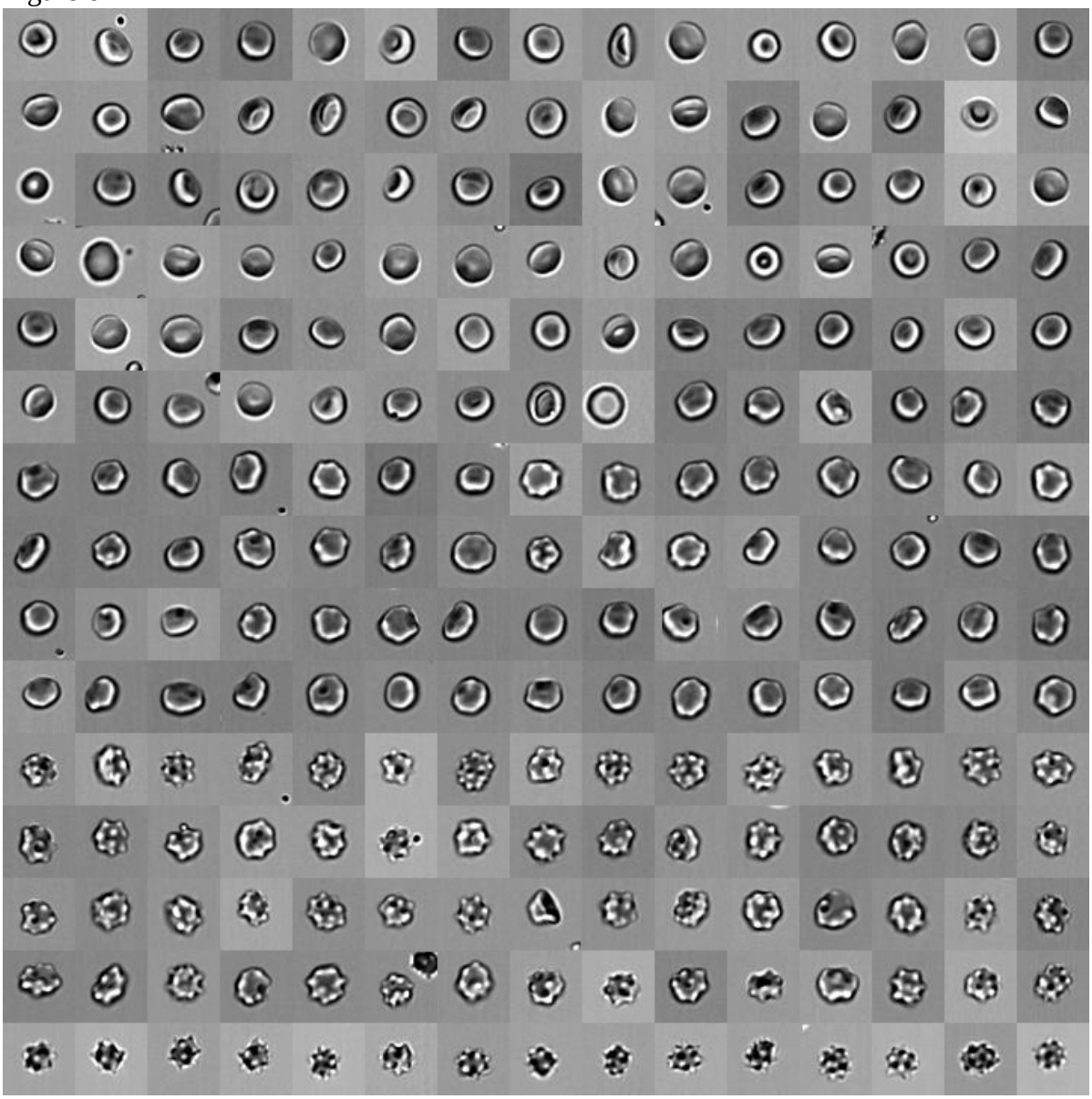
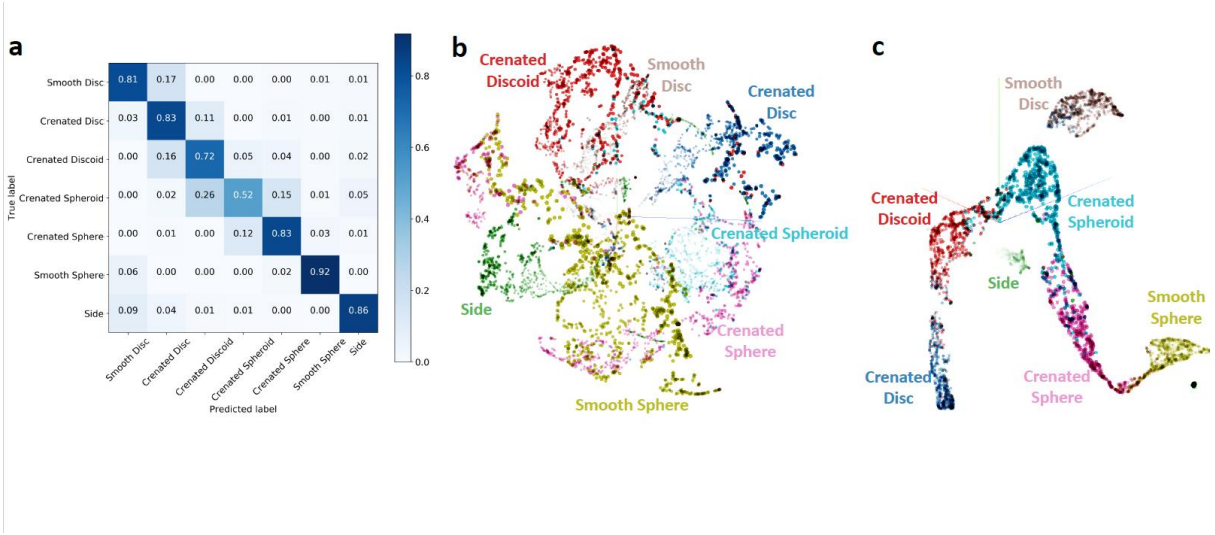


Figure 7


a
Smooth Disc
Crenated Disc
Crenated Discoid
Crenated Spheroid
Crenated Sphere
Smooth Sphere
Side
True label
Predicted label
b
Crenated Discoid
Smooth Disc
Crenated Disc
Crenated Spheroid
Side
Crenated Sphere
Smooth Sphere
c
Smooth Disc
Crenated Discoid
Crenated Spheroid
Side
Smooth Sphere
Crenated Disc
Crenated Sphere