

# Convolutional neural networks for classifying chromatin morphology in live cell imaging

Kristina Ulicna<sup>1,2\*</sup>, Laure T.L. Ho<sup>1,3\*</sup>, Christopher J. Soelistyo<sup>1\*</sup>, Nathan J. Day<sup>1</sup> and Alan R. Lowe<sup>1,2,4,5+</sup>

1. Institute of Structural and Molecular Biology, University College London, London, WC1E 6BT, UK
2. London Centre for Nanotechnology, University College London, London, WC1H 0AH, UK
3. MRC Laboratory of Molecular and Cellular Biology, London WC1E 6BT, UK
4. Institute for the Physics of Living Systems, University College London, London WC1E 6BT, UK
5. The Alan Turing Institute, London, London NW1 2DB, UK

\* These authors contributed equally

+ Correspondence to: [a.lowe@ucl.ac.uk](mailto:a.lowe@ucl.ac.uk)

Running head: CNN Annotator of Chromatin Morphology

## Abstract

Chromatin is highly structured, and changes in its organisation are essential in many cellular processes, including cell division. Recently, advances in machine learning have enabled

researchers to automatically classify chromatin morphology in fluorescence microscopy images. In this protocol, we develop user-friendly tools to perform this task. We provide an open-source annotation tool, and a cloud-based computational framework to train and utilise a convolutional neural network to automatically classify chromatin morphology. Using cloud compute enables users without significant resources or computational experience to use a machine learning approach to analyse their own microscopy data.

## Keywords

Machine learning, Live cell imaging, Cell cycle, Image analysis, Computational biology

## 1. Introduction

Chromatin, the higher order structure of nuclear DNA and proteins, constitutes the majority of the nucleus of eukaryotic cells. Changes in chromatin structure lay the basis of many essential nuclear processes, including transcription, meiosis, mitosis, and apoptosis **(1)**.

### 1.1 Chromatin morphology changes throughout cell's life cycle

The cell cycle is the coordination of highly controlled molecular processes which result in the replication of the cell's genetic material and its equal partitioning into two daughter cells upon division (mitosis) **(2)**. The cell cycle stage leading to mitosis is referred to as interphase, the longest phase between subsequent cell divisions defined by cell growth and DNA replication event. The interphase is then followed by mitosis, during which the cell's DNA is compacted, partitioned, and distributed into the newly formed cells.

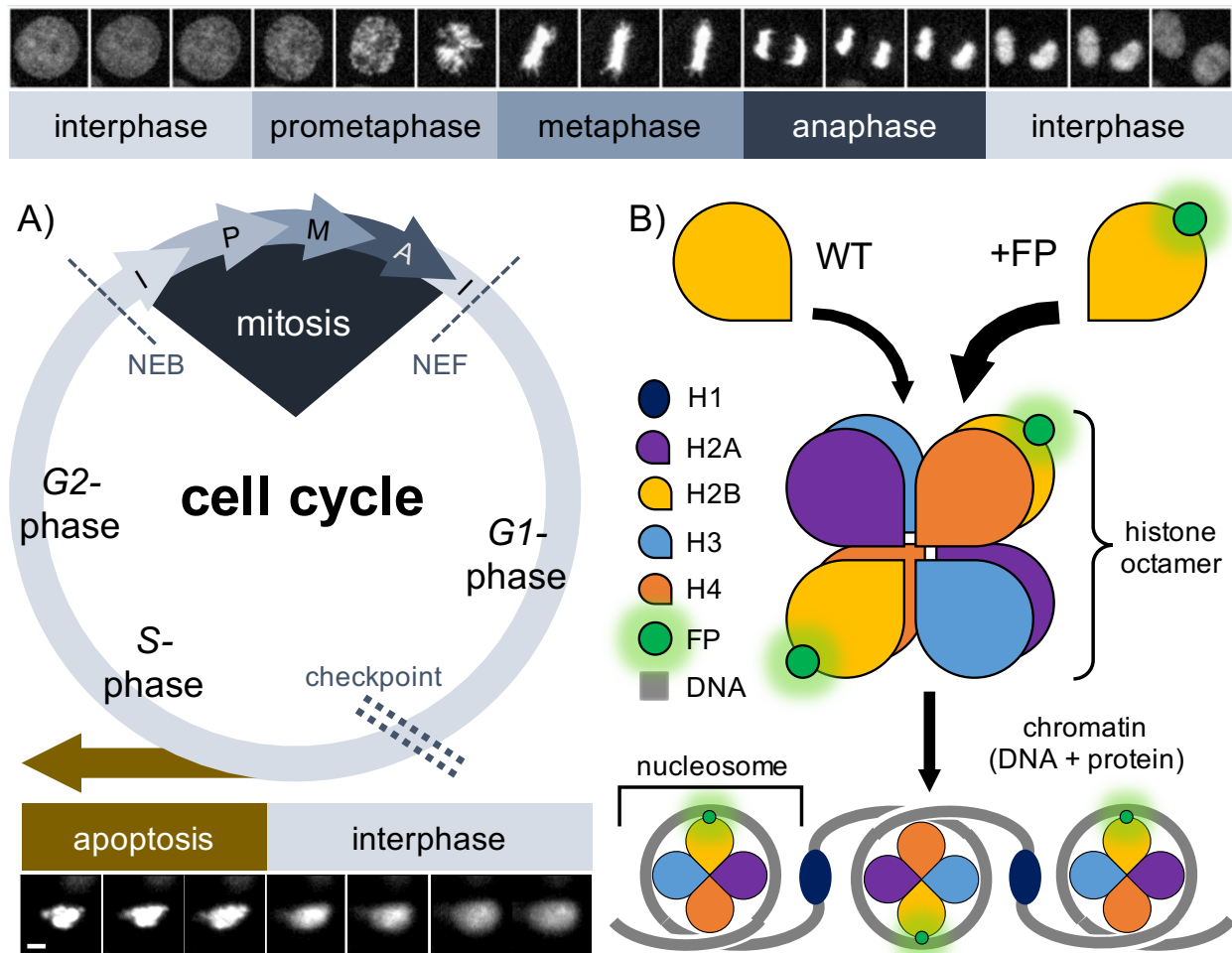
Although mitosis is a continuous process, it can be subdivided into discrete phases such as pro(meta)phase, characterised by chromosome condensation and nuclear envelope breakdown;

metaphase, during which duplicated chromosomes align to the metaphase plate and their kinetochores attach to centrosome microtubules; and finally ana(telo)phase, described by sister chromatids separation into newly formed daughter cells, where new nuclear envelope is formed and cytokinesis begins **(2)**. As a quality control mechanism, the cell can decide to undergo apoptosis, which represents a highly regulated cell signalling process of biochemical and/or mechanical nature leading to programmed cell death.

The progression through all cell cycle stages is an orchestrated series of events with visually distinct morphological changes to the cell nucleus **[Fig 1]**. To visualise the nuclear dynamics, various fluorescent dyes have historically been used, including DAPI **(3)** and Hoechst 33342 **(4)**. These permeate the cell membrane and directly bind to nucleic acids, which limits their application to fixed samples or to relatively short imaging periods (several hours to days), respectively. However, for longer-term live-cell microscopy over several cell divisions across multiple generations, it is preferable to engineer cell lines to stably express fluorescent reporter(s) of nuclear architecture. This can be achieved by hijacking the primary function of core histone proteins, namely H2A, H2B, H3 and H4 **(5)**, which bind to DNA to form nucleosome building blocks of chromatin, a higher order of DNA structure **(6)**. Therefore, overexpression of fusion histone proteins with an attached fluorescent tag, such as GFP or mCherry proteins, offers real-time capturing of the nuclear distribution, condensation level or apoptosis-triggered degradation of chromatin in live eukaryotic cells.

In experiments which require imaging of live-cell samples containing hundreds to thousands of single cells over long durations, manual annotation of each individual cell's stage at any given time point represents a laborious, time-consuming and error-prone task due to the high volume of single-cell observations. Because of the characteristic appearance of the chromatin throughout the cell cycle, automated labelling of microscopy image patches cropped around cell nuclei is

possible with computer vision approaches, which also significantly speeds up the analysis of high microscopy volumes.



**Fig 1 | Chromatin morphology during cell cycle. (A)** Cell cycle progression with phase-specific changes to the H2B-GFP labelled chromatin condensation level. Single-cell time-lapse image patches capturing the process from nuclear envelope breakdown (NEB) to nuclear envelope formation (NEF) are illustrated (top). During interphase, the G1, S and G2 phases are separated by cell cycle checkpoints that regulate progression through the cycle; alternatively, the cell may undergo apoptosis, or programmed cell death (bottom). Images taken every 4 minutes. Scale bar = 5µm. **(B)** Chromatin labelling with a fluorescent histone reporter. A vector with gene encoding for a chosen histone protein (H2B) fused to a visualisation tag, such as green fluorescent protein

*(GFP), can be stably transfected into a cell line. Overexpression of H2B-FP under strong promoter control encourages the incorporation of the tagged histone protein over the wild-type version into the nascent histone octamer, which the nuclear DNA wraps around to form the resulting higher chromatin structure. The GFP tag is associated with the nucleic acid into nucleosome structures, which allows for easy chromatin visualisation in live cells using fluorescence microscopy modality.*

## 1.2 Machine learning strategies for cell state classification

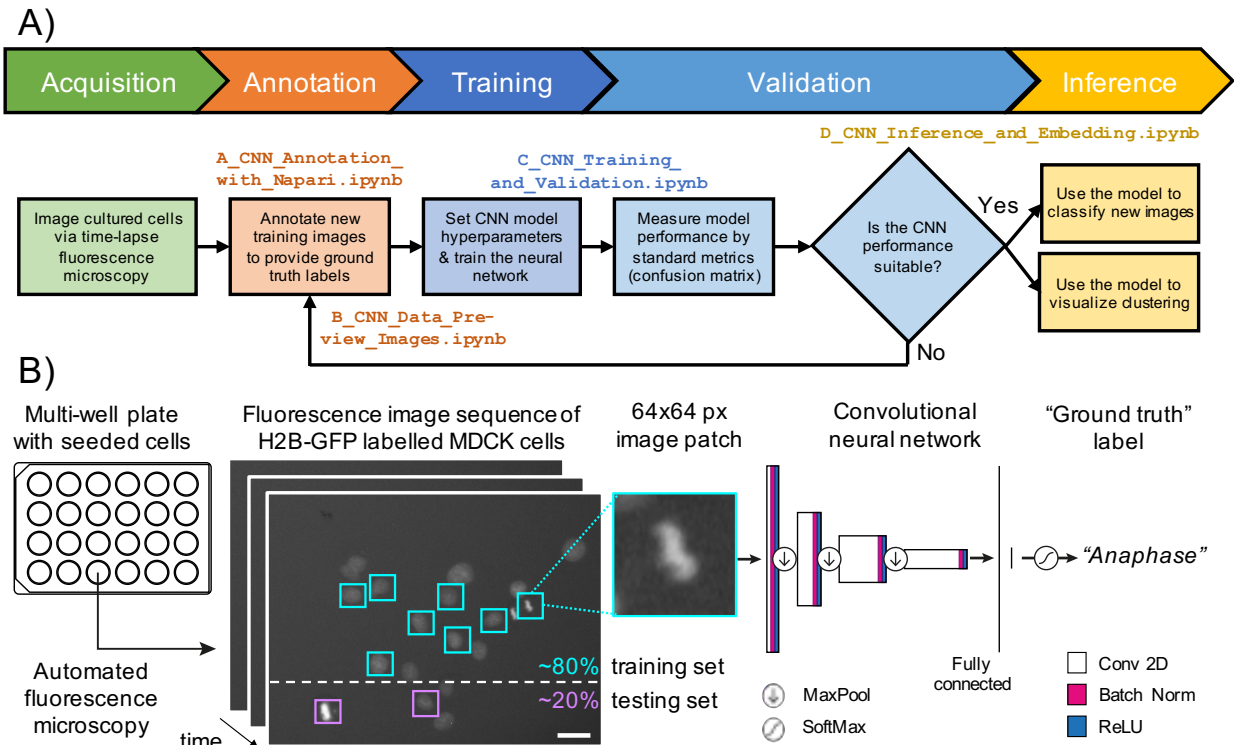
In recent years, several machine learning based methods have been developed to classify cells according to their morphology. For example, a support vector machine (SVM) was implemented to classify fluorescence images of HeLa cells into one of eight states (interphase, prophase, prometaphase, metaphase, early anaphase, late anaphase, telophase or apoptosis) **(7)**. Fluorescence images, exhibiting the cells' chromatin morphology, were embedded in a multi-dimensional feature space. These quantitative features included the circularity and perimeter length of the cell. The SVM then deduced the optimal hyperplane(s) within the feature space separating data points of different categories.

Another algorithm commonly used is a Random Forest classifier, a variant of the Decision Tree algorithm that - like the SVM - classifies data points based on their associated features. *ilastik* is a bioimage analysis tool that uses the Random Forest approach to classify objects or individual pixels according to a set of user-defined categories (e.g. "nucleus", "cytoplasm" etc.) and manual labels **(8)**. For individual pixels, the features obtained are the output of image filters, and they include pixel intensity and pixel edge-ness, amongst others.

Convolutional Neural Networks (CNNs) are deep learning algorithms that have also been used to classify cells based on their morphology. A major advantage of CNNs is that, unlike other approaches, they do not require feature engineering - the network learns to extract image features

that fulfil the task. Over the last decade, this network architecture has demonstrated significant utility in classifying images across many different image domains **(9, 10)**. As an example from the microscopy domain, CNNs have been used to classify actin-labelled fluorescence images of human breast cells as belonging to either “normal” (MCF-10A) or “cancerous” (MCF-7 or MDA-MB-231) cell lines **(11)**. The high performance of the model - more accurate than that of human experts - demonstrates the potential of CNNs to correctly classify cells according to actin filament structure.

As we will demonstrate, this capability can be applied to chromatin structure classification. We have previously used CNNs to classify chromatin morphology in MDCK cell lines **(10, 19)**. In this protocol, we provide the tools and a guide on how the user can create a dataset of single-cell image patches from an example dataset, and train a convolutional neural network to automatically assign labels to single cells throughout the entire cell cycle progression **[Fig 2]**. We also describe how to evaluate the network performance, infer new classifications on previously unseen examples in a fully automated manner, and visualise the encoded image representations in reduced dimensionality embeddings.



**Fig 2 | Overview of the protocol pipeline. (A)** Schematic representation of the protocol pipeline involved in acquiring data, training a convolutional neural network for classification, and its utilisation in inference of chromatin morphology in live-cell microscopy data. Corresponding Jupyter notebooks for each part of this protocol are color-coded accordingly. **(B)** Practical implementation of the protocol. First, time-lapse microscopy data of live proliferating cells in culture are acquired. Next, users annotate a sample of single-cell images to generate training (~80%) and testing (~20%) sets of images, with the associated ground truth labels. Finally, these image-label pairs are used to train a convolutional neural network (CNN) as illustrated. The CNN consists of several convolutional layers that extract image features, followed by densely connected layers that transform the features into a categorical label. Conv 2D, two-dimensional convolution operation; batch norm, batch normalisation; ReLU, rectified linear unit. Scale bar = 50 $\mu$ m.

### 1.3 Exploratory visualisation of CNN feature embedding

Single-cell data is often high-dimensional and requires tools to enable exploratory visualisation. To tackle dimensionality problems encountered when analysing high-dimensional data as we are doing here, dimensionality reduction techniques have been developed and are now applied in various areas **(12, 13)**. These approaches reduce the high number of dimensions by transforming the data into lower-dimensional space while attempting to retain intrinsic features of the original data **(14)**. This low-dimensional embedding is usually 2D or 3D, which greatly facilitates visualisation and interpretation of the high-dimensional data. Techniques broadly use two approaches, either by trying to preserve the overall data distance structure, e.g. in the widely used Principal Component Analysis (PCA) **(15)**, or by prioritising local neighbour distances over global distances, e.g. in t-Distributed Stochastic Neighbor Embedding (t-SNE) **(16)** and Uniform Manifold Approximation and Projection (UMAP) **(17)**.

PCA is a common linear method where high-dimensional data are projected onto the axes that can explain the greatest underlying variance **(15)**. It also allows to quantify the individual contribution of input variables, thus making this method one of the most interpretable and commonly used approaches. However, due to the complex non-linear nature of microscopy imaging data, it is often necessary to use non-linear techniques **(14, 18)** such as t-SNE or UMAP to properly interpret our CNN's latent representations of the training images.

Although t-SNE and UMAP yield similar low-dimensional embeddings, UMAP provides an added feature to reuse a computed transform function on new unseen data in order to project it into the same low-dimensional space **(17)**. UMAP has also been proposed to have quicker performance than t-SNE and other techniques when scaling to considerably larger datasets **(13, 17)**. In this protocol, we use UMAP to visually validate whether the CNN has successfully learned relevant image features to correctly classify the different cell cycle stages. However, these approaches can also be extended to visualise orthogonal reporters. For example, using reporters for both



chromatin morphology and the nuclear envelope, one could cluster cells by chromatin morphology but visualise the structure of the nuclear envelope associated with each mitotic stage.

## 2. Materials

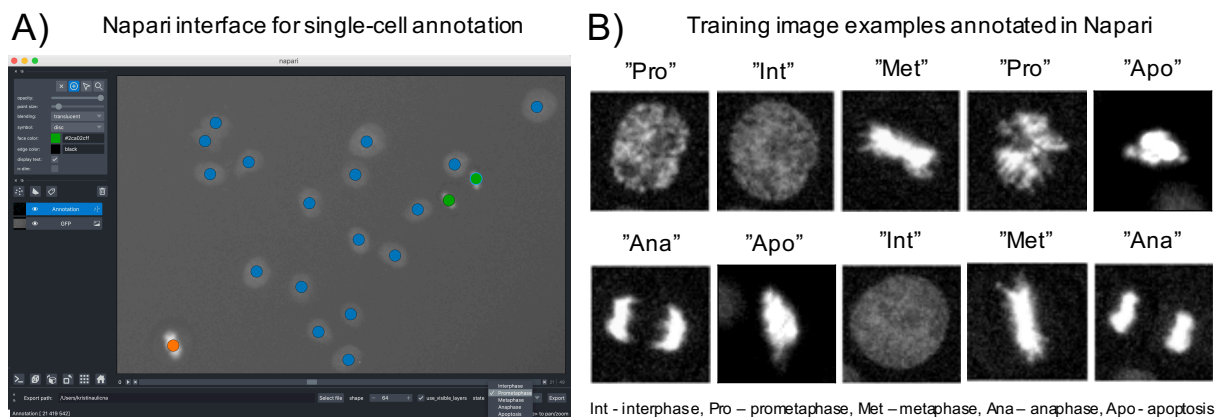
1. This protocol relies on a source of high quality fluorescence microscopy data. These are typically time-lapse image sequences in two or three spatial dimensions where at least one channel is a fluorescent marker for chromatin (see **Note 1**). Each image should be stored as a multidimensional TIF file in the standard format (see **Note 2**).
2. A computer connected to the internet, capable of running a web browser.
3. An installation of Python (see **Note 3**) of version 3.7 or higher on your machine (see **Note 4**).
4. A Google account to train models in the cloud using the provided Google Colab environment (see **Note 5**).
5. Follow the installation instructions on the GitHub repository to create a clean virtual environment with all needed Python libraries and our `cnn-annotator` package:  
<https://github.com/lowe-lab-ucl/cnn-annotator> (see **Note 6**).

## 3. Methods

### 3.1 Annotating training images using Napari

1. At the command line make sure you are in your local `cnn-annotator` directory and type `jupyter notebook`, you will then be redirected to a Jupyter notebook environment in your default browser. Navigate to the `/notebooks/` directory where the repository structure is outlined.

2. Open `A_CNN_Annotation_with_Napari.ipynb` by clicking on the filename. A separate tab will open.
3. In the notebook, modify the file path to your image data (*default:*  
`/data/MDCK_H2B_GFP_movie.tif`).
4. If you are using your own classification labels, create those labels in the notebook using the `CellState` structure (see **Note 7**).
5. Run the notebook and a separate Napari instance with a GUI window will pop up (see **Note 8**).
6. Inside Napari select a label corresponding to a particular chromatin morphology and annotate the image data by placing a small marker at the centre of each cell with the appropriate morphology [**Fig 3**] (see **Note 9**).
7. Aim to annotate several examples of each state that you wish to classify (see **Note 10**).
8. Export the data using the “export” button. This writes a single zip file containing all of the image patches as well as the locations in T(Z)YX and ground truth labels. This zip file can be used as a source of training data for the CNN (see **Note 11**).
9. You can close the Napari instance when done. Repeat steps 5-8 to create additional annotation zip files for training, validation and future inference (see **Note 12**).
10. (Optional) If you wish to assess the class balance and visualise your annotations, return to the Jupyter notebook interface, then open and run  
`B_CNN_Data_Preview_Images.ipynb`.



**Fig 3 | Napari widget for single-cell annotation.** (A) A screenshot of the graphical user interface (GUI) provided by Napari, with the custom-designed annotation widget provided as part of this protocol. The widget allows users to annotate their images with categorical labels representing different chromatin morphologies, and to export these single-cell image patches and annotations for further analysis and training of the CNN using cloud computation. (B) Representative single-cell image patches and annotated label pairs serving as ground truth dataset extracted using the annotation widget.

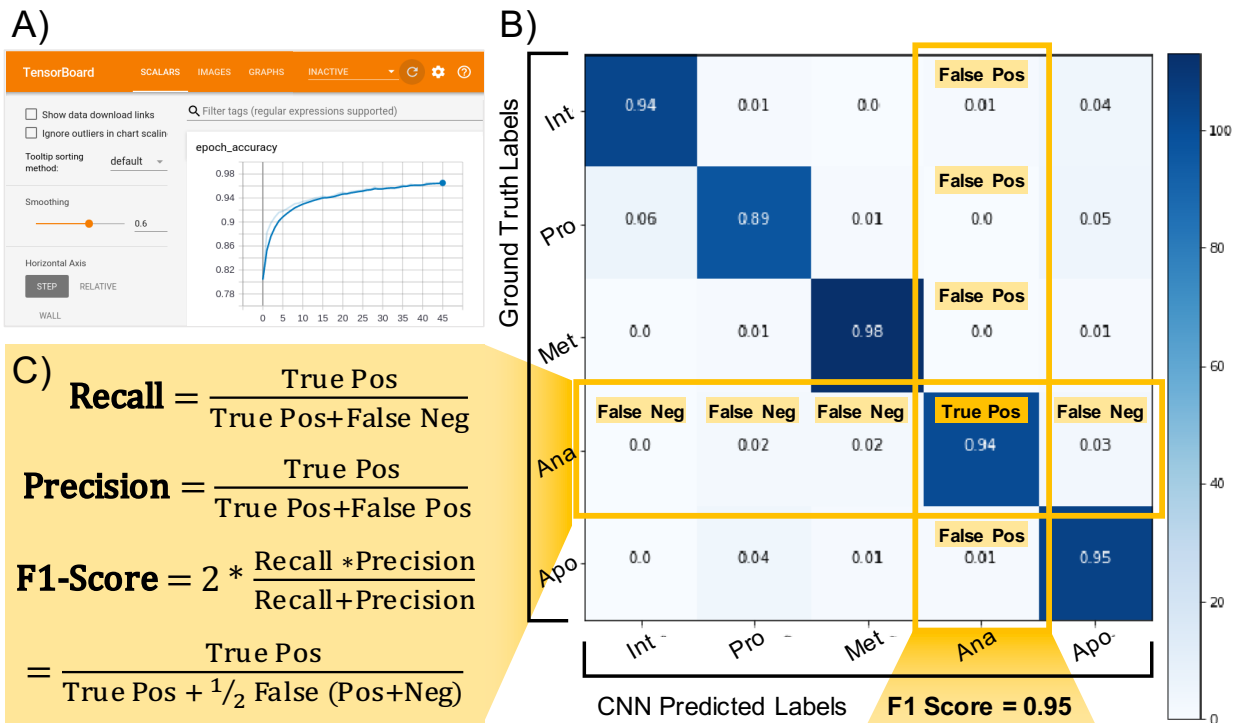
## 3.2 Training a CNN for chromatin morphology classification

### 3.2.1 Training the CNN on a train set

1. From the GitHub repository, click on the “Open in Colab” notebook for Training (see **Note 13**).
2. Follow the instructions in the notebook, and upload the training and validation zip files to the Colab `/content/train/` and `/content/validation/` subfolders.
3. Make sure that you set aside some data for validation (see **Note 14**).
4. Set the hyperparameters required for training (see **Note 15**).
5. Run the Colab notebook to train the model.

### 3.2.2 Evaluating the CNN on a validation set

6. Monitor the training performance using TensorBoard (see **Note 16**), which will include the calculation of a confusion matrix **[Fig 4]** (see **Note 17**).
7. If the network performs satisfactorily on the separate validation data, it means it was sufficiently trained and can now be used to classify examples in new datasets (see **Note 18, Note 19**).
8. Save out and download the trained model (*default name*: `model.h5`) for future inference (see **Note 20**).



**Fig 4 | Evaluation of CNN performance with multi-class categorisation. (A)** Interactive TensorBoard interface for visualising the training of the CNN in real-time. Metrics include the training loss, overall accuracy (shown), and a confusion matrix of training accuracy for each categorical class. **(B)** A confusion matrix for CNN network performance displaying high accuracy on the diagonal of the matrix, indicating agreement between the ground truth labels (rows) and

*the CNN prediction labels (columns). Calculation of “anaphase” class-specific F1-score reflective of network’s recall (sensitivity) and precision (specificity), according to the equations shown in (C).*

### 3.3 Using the CNN for chromatin morphology classification on unseen data

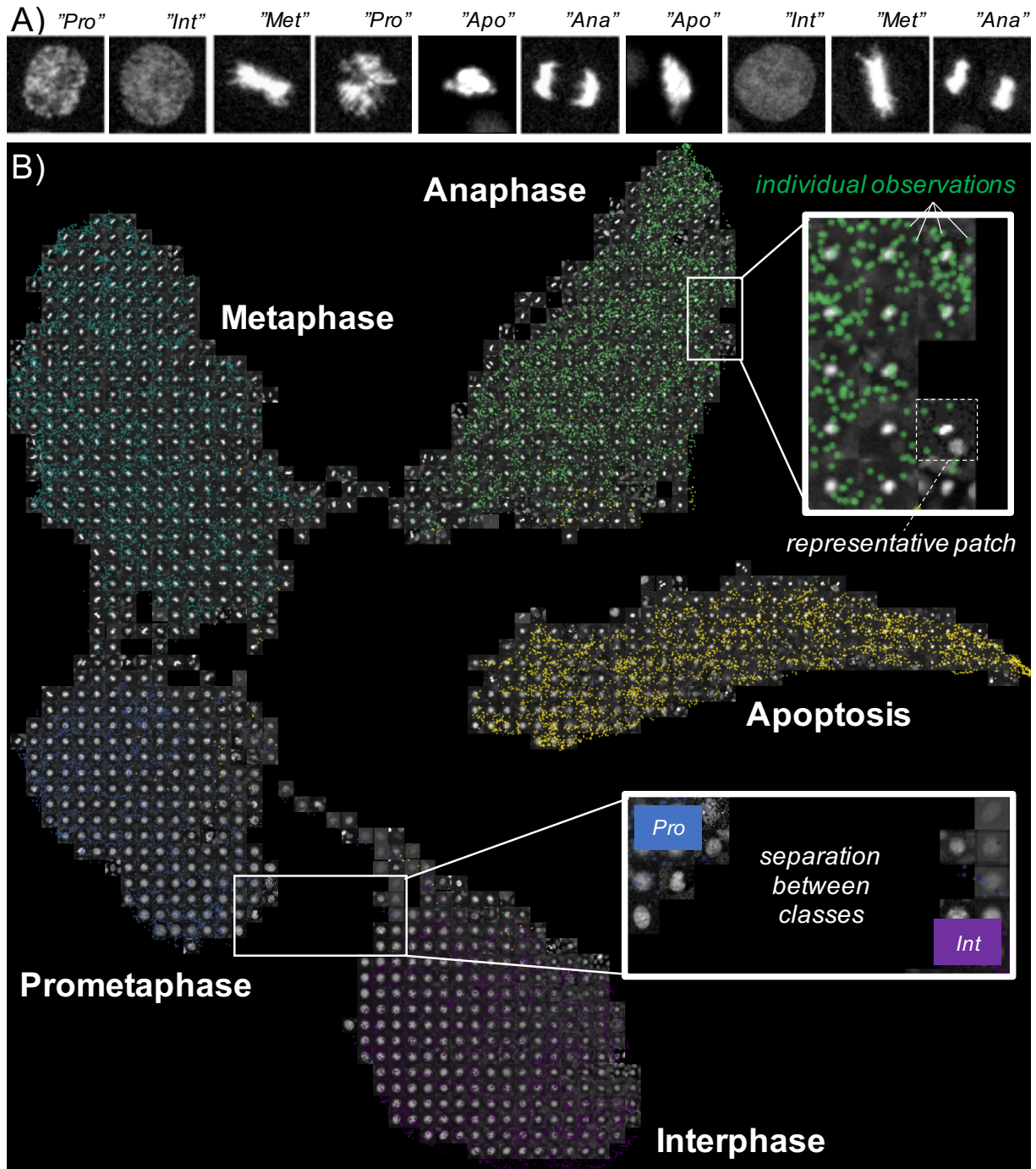
Once the CNN has been successfully trained, the goal is to use it to explore unseen data. The inference network takes an unseen image of a cell nucleus and returns the prediction of its classification label.

#### 3.3.1 Using the trained CNN for inference on a test set

1. From the GitHub repository, click on the “Open in Colab” notebook for Inference (see **Note 13**).
2. Upload your trained model (*default name*: `model.h5`) to the Colab default `/content/` folder.
3. Upload your test zip file(s) to the Colab `/content/test/` subfolder.
4. Run the notebook to perform predictions on real data **[Fig 5]** (see **Note 21**).
5. Calculate the confusion matrix using the notebook (see **Note 17**).
6. Calculate additional performance metrics, including the F1-score, precision and recall (see **Note 22**).

#### 3.3.2 Visualising the CNN feature embedding

7. (Optional) Perform UMAP dimensionality reduction along with a manifold projection of single-cell image patches to visualise the network’s classification in 2D **[Fig 5]** (see **Note 23**).



**Fig 5 | 2D representation of the CNN feature embedding.** (A) Representative predictions from the trained network for the given images. (B) A two-dimensional image-based representation of the CNN output, created using the dimensionality reduction method UMAP. The image represents the manifold generated using 20,000 example images, with individual data points representing

*individual observations overlaid. Clear clustering of image patches with discrete chromatin morphologies (and their categorical labels) can be observed.*

## 4. Notes

1. Although we use H2B-GFP as a fluorescent marker proxy for chromatin organisation, other fluorescent reporters, such as DAPI, can also be used.
2. We use tiff stacks of image data for the annotation stage. These are usually derived from time-lapse microscopy acquisitions and are ordered as T(Z)YXC. Each channel, for example GFP, is a separate stack of images. The brightfield transmission data can also be added as an additional input to improve the accuracy of the classification. Although this protocol is primarily designed for two-dimensional datasets, the same approach can be utilised for volumetric imaging data. We provide some example image data in the GitHub repo.
3. We recommend installing a complete scientific distribution of Python, such as Anaconda. Follow the installation instructions for your platform here:  
<https://www.anaconda.com/products/individual>
4. You can check which version of Python you have installed by typing `python -version` at the command line. We recommend version 3.7 or greater.
5. You will need a Google email address and be signed in to run notebooks in the Google Colab environment. Sign up for an account here: <https://accounts.google.com/signup>
6. Please note that we are constantly adding new features and updating the live repository. For the latest version of the protocol and more detailed instructions, please remember to check <https://github.com/lowe-lab-ucl/cnn-annotator>. Alternatively, you will also find a `requirements.txt` on the repo with which you can install the necessary packages using other means and then install the `cnn-annotator` package from GitHub.

7. We recommend using the following labels for annotating the data: 0 - Interphase, 1 - Prometaphase etc. However, it is possible to generate your own labels depending on the goal of the classification task. In the notebook we have defined five states and given each a unique numerical value. However, it is possible to define your own states (for example, '5 - Telophase') by simply adding another entry to this structure, with a unique numerical value.
8. We use the multidimensional image viewer Napari **(20)** for data visualisation. Read more about Napari here: <https://napari.org/>
9. Annotations that are too close to the edge of the field of view will be exported but will be padded with zero values, and excluded from training by default. However, you are given the option of using or excluding these examples during training of the model. It is useful to provide examples when it is likely that your real dataset will also include examples that are close to the edge of the field of view.
10. You should aim to provide at least 500 examples of each class to properly train the network. This dataset should be free from errors and is referred to as the “ground truth dataset”.
11. Annotation files will be named using a common format and saved to the local drive. Each file is a zip archive containing the raw image patches and the associated metadata. You can find several examples within the GitHub repo (*default save path*:  
`/data/annotation_MM-DD-YYYY--hh-mm-ss.zip`).
12. Each annotation file will be given a unique time and date stamp, allowing multiple datasets to be annotated. We recommend providing data from multiple different experiments to properly sample the real data distribution. This will prevent overfitting during training and present a generalisable model as the result.
13. The “Open in Colab” links can be found on the GitHub repository page:  
<https://github.com/lowe-lab-ucl/cnn-annotator>



14. In addition to the training dataset, it is important to reserve some example images for evaluating the resultant network. These images are “held out” and not used during training. This would typically be 10-20% of your training set, and can be achieved by saving additional annotated images in section 3.2. These annotations should be set aside and not used when training the network. This enables a fair evaluation of the network’s performance.

15. The following hyperparameters can be modified before training your model:

`BATCH_SIZE` - Number of training images the model is trained on in one iteration, after which the model computes its loss before continuing to train.

`BUFFER_SIZE` - Number of elements in a buffer from which each training example will be picked at random for shuffling purposes, it is recommended to have a buffer size equal or larger to the full dataset size for optimal shuffling.

`TRAINING_EPOCHS` - One training epoch is a full pass over the entire training set, i.e. the network will have seen every image once; more specifically one epoch consists of  $(\text{dataset\_size} / \text{BATCH\_SIZE})$  training iterations.

`BOUNDARY_AUGMENTATION` - If set to ‘True’: when building the training dataset, random simulations of cropped images that usually occur at the edge of imaging volumes will be added to the training dataset. This improves the generality of the model.

`INPUT_SHAPE` - Dimensions of an input training image to the network, typically in the format (width, height, channels) for 2D datasets.

16. On TensorBoard, you should see a decrease in the loss and an increase in the accuracy as the network trains. This is an indication that the network’s performance on the validation set is improving. Click on the “SCALARS” tab to see the accuracy and loss during training. Click on the “IMAGES” tab to see the confusion matrix during training.
17. A confusion matrix is an NxN table (where N is the number of classes) that is usually used to evaluate the performance of a classification network on a set of testing or

validation data. The number in row A and column B of the table is equal to the number of testing examples of ground-truth class A that have been classified by the network as belonging to class B. Therefore, the numbers shown in the main diagonal of the table (from top-left to bottom-right) indicate those examples that have been classified correctly by the network.

18. Evaluating the network's performance on a set of previously unseen data is a way of ensuring that the network is able to generalise, and is not simply "remembering" the examples in the training dataset. Hence, it is a way of ensuring that the network is able to conduct the task that it has been trained to do. In general, the acceptable level of accuracy is dependent on the task. For labelling image sequences we routinely achieve accuracies  $>0.95$  and often  $>0.99$ .
19. When using a trained network for inference on unseen data, we assume that the new data are independent and identically distributed (i.i.d.) to the training data.
20. The trained model is stored in the HDF5 file format and can be reused once trained. We recommend keeping a local copy of the trained model once you are satisfied with the performance.
21. The trained network can now be used to assign classification labels to cells in unseen sequences. This is very useful when trying to automatically extract division events from time-lapse sequences or for clustering image sets with orthogonal fluorescence markers.
22. The "recall" of a certain class C is the proportion of ground-truth examples of C that have been correctly classified. Thus, a low recall indicates that there are many ground-truth examples of C that have been incorrectly classified as belonging to another class. The "precision" of class C is the proportion of examples classified by the network as belonging to C that are actually ground-truth examples of C. Thus, a low precision indicates that there are many examples classified by the network as belonging to C, but that are actually ground-truth examples of other classes. The F1 Score of class C is the

harmonic mean of precision and recall, and acts as a metric for evaluating the performance of the network on that specific class. These three class-specific metrics are important because the overall “accuracy” of a network when applied to a testing or validation set describes the general performance of the network across all classes, and does not make clear the difference in performance *between* classes. By calculating these metrics, we may assess how well the network performs for any specific class.

23. By analysing the UMAP manifold projection, we can see how image patches are localised in the feature embedding. Distinct clusters of images should correspond to the various defined classes, implying that the network has appropriately learned underlying image features to properly classify the patches. If images of the same class do not cluster as you would expect, one of the reasons could be that the `n_neighbors` parameter was not properly chosen. For more information about some important UMAP parameters, check out this documentation:

<https://umap-learn.readthedocs.io/en/latest/parameters.html#>

## Acknowledgements

This work was supported by BBSRC grant BB/S009329/1 to ARL. KU, LTLH and CJS were supported by BBSRC London Interdisciplinary Doctoral Programme studentships. NJD was supported by an MRC DTP studentship (MR/N013867/1). We gratefully acknowledge Giulia Vallardi for providing example data and Guillaume Charras for discussions.

## References

1. Kaplan N, Moore IK, Fondufe-Mittendorf Y, et al (2009) The DNA-encoded nucleosome organization of a eukaryotic genome. *Nature* 458:362–366
2. Mitchison TJ and Salmon ED (2001) Mitosis: a history of division. *Nat Cell Biol* 3:E17–E21
3. Kapuscinski J (1995) DAPI: a DNA-Specific Fluorescent Probe. *Biotech Histochem* 70:220–233
4. Durand RE (1982) Use of Hoechst 33342 for cell selection from multicell systems. *J Histochem Cytochem* 30:117–122
5. Kornberg RD and Thomas JO (1974) Chromatin Structure: Oligomers of the Histones. *Science* 184:865–868
6. Kornberg RD and Lorch Y (1999) Twenty-Five Years of the Nucleosome, Fundamental Particle of the Eukaryote Chromosome. *Cell* 98:285–294
7. Held M, Schmitz MHA, Fischer B, et al (2010) CellCognition: time-resolved phenotype annotation in high-throughput live cell imaging. *Nat Methods* 7:747–754
8. Berg S, Kutra D, Kroeger T, et al (2019) ilastik: interactive machine learning for (bio)image analysis. *Nat Methods* 16:1226–1232
9. Krizhevsky A, Sutskever I, and Hinton GE (2017) ImageNet Classification with Deep Convolutional Neural Networks. *Commun ACM* 60:84–901
10. Bove A, Gradeci D, Fujita Y, et al (2017) Local cellular neighborhood controls proliferation in cell competition. *MBoC* 28:3215–3228
11. Oei RW, Hou G, Liu F, et al (2019) Convolutional neural network for cell classification using microscope images of intracellular actin networks. *PLoS ONE* 14:e0213626
12. Koch FC, Sutton GJ, Voineagu I, et al (2020) Supervised Application of Internal Validation Measures to Benchmark Dimensionality Reduction Methods in scRNA-seq Data. *bioRxiv preprint* doi:10.1101/2020.10.29.361451

13. Becht E, McInnes L, Healy J, et al (2019) Dimensionality reduction for visualizing single-cell data using UMAP. *Nat Biotechnol* 37:38–44
14. Van Der Maaten L, Postma E, and Van den Herik J (2009) Dimensionality Reduction: A Comparative Review. *J Mach Learn Res* 10:13
15. Hotelling H (1933) Analysis of a complex of statistical variables into principal components. *J Educ Psychol* 24:417–441
16. Van Der Maaten L and Hinton G (2008) Visualizing Data using t-SNE. *J Mach Learn Res* 9:2579–2605
17. McInnes L, Healy J, and Melville J (2018) UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv preprint arXiv:1802.03426v2*
18. Yin H (2007) Nonlinear dimensionality reduction and data visualization: A review. *Int J Automat Comput* 4:294–303
19. Ulicna K, Vallardi G, Charras G, et al (2020) Automated deep lineage tree analysis using a Bayesian single cell tracking approach. *bioRxiv preprint* doi:10.1101/2020.09.10.276980
20. Sofroniew N, Lambert T, Evans K, et al (2021) napari/napari: 0.4.3rc0, Zenodo. doi:10.5281/zenodo.4435160