ZeroCostDL4Mic User Manual

Henriques and Jacquemet labs

v1.2 – April 2020

Table of Contents

A. What is ZeroCostDL4Mic?

ZeroCostDL4Mic is a toolbox for the training and implementation of common Deep Learning approaches to microscopy imaging (Figure 1). It exploits the ease-of-use and access to GPU provided by Google Colab. Training data can be uploaded to the Google Drive from where it can be used to train models using the provided Colab notebooks in a web-browser. Predictions on unseen data can then be performed within the same notebook, therefore not requiring any local hardware or software set-up.

This guide describes the workflow of a typical notebook run-through in a step-by-step fashion. You can also refer to this video for a complete walkthrough.

https://www.youtube.com/watch?v=GzD2gamVNHI&feature=youtu.be

A.1. Overview of the workflow

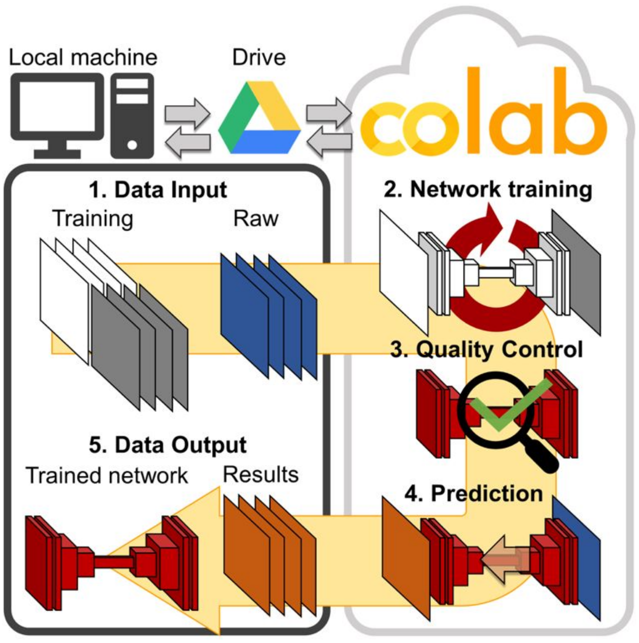


Figure 1: Workflow of ZeroCostDL4Mic. From data upload, execution of training, quality control and prediction on the cloud, to results and trained model download.

ZeroCostDL4Mic exploits online resources made available through Google Colab. **The general workflow comprises the following steps and is showcased in the following sections A-E**:

i) Uploading data to Google Drive
ii) Opening Colab Notebook online in Google Colab
iii) Connecting the Google Drive to the current Colab session
iv) Installation of network components dependencies
v) Set training parameters (including transfer learning and data augmentation)
vi) Run the training
vii) Run Quality control on the trained model to ensure the validity of the trained model
viii) Run predictions on unseen dataset
ix) Download the trained network and the predictions on the unseen datasets

A.2. Networks currently implemented

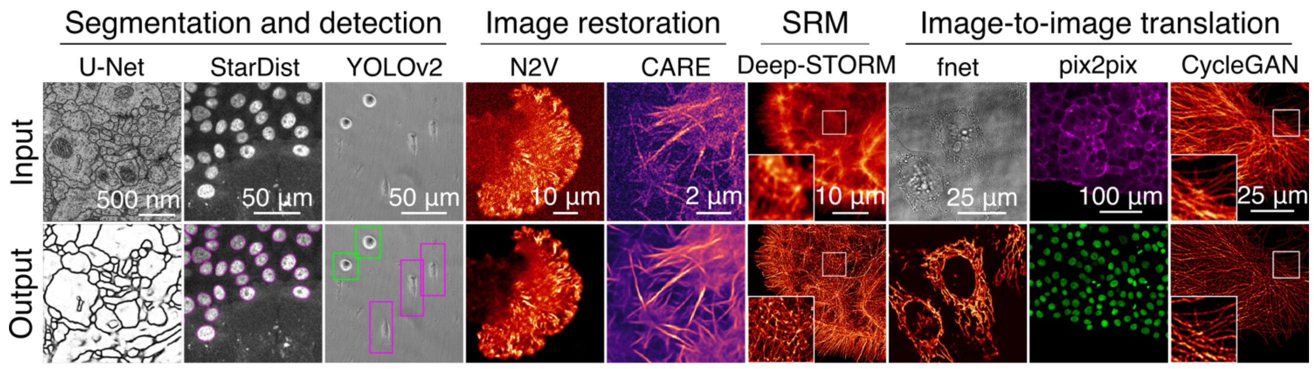


Figure 2: Networks and tasks currently implemented in ZeroCostDL4Mic. Segmentation and object detection, image restoration (including denoising), super-resolution microscopy (SRM) and image-to-image translation (including artificial labelling) are currently available.

- **U-Net (2D and 3D)**: for segmentation of bioimages, typically from electron microscopy data or bright-field imaging using supervised learning[1,2].
- **YOLOv2:** can classify objects and draw bounding boxes around detected objects. The original work is found here[3].
- **Noise2Void (2D and 3D)**: for denoising directly from the raw data, unsupervised so no need for paired training dataset[4].
- **StarDist (2D and 3D)**: for segmentation of cell nuclei in fluorescence images, specifically designed to detect round/oval objects using supervised learning[5,6].
- **CARE (2D and 3D)**: provides image restoration (denoising and resolution enhancement, among other features) using fully supervised learning[7].
- **Label-free prediction (fnet)**: predicts pseudo-fluorescence images from bright-field or EM data using supervised learning. The original work is found here[8].
- **pix2pix**: translates an image belonging to a specific image domain type into another image domain type. The original work is described here[9].
- **CycleGAN**: translates an image belonging to a specific image domain type into another image domain type without requiring the training dataset to be paired (unsupervised learning). The original work is described here[10].
- **Deep-STORM**: processes high-density data from single-molecule localization microscopy (SMLM) acquisitions. Supervised learning is performed on simulated single-molecule data. The original work is described here[11].

A.3. Online resources

Code development is a never-ending endeavour. We keep the framework alive by constantly developing new capabilities and adding new tasks and networks. Therefore, we make available many useful pieces of information via our GitHub page.

Below are a few links to specific online resources:

Google Colab
https://colab.research.google.com/notebooks/intro.ipynb

Link to our bioRxiv paper
https://www.biorxiv.org/content/10.1101/2020.03.20.000133v2

Wiki page on our GitHub repository
https://github.com/HenriquesLab/ZeroCostDL4Mic/wiki

Reporting bugs or issues
https://github.com/HenriquesLab/ZeroCostDL4Mic/issues

Page to the latest releases of the notebooks
https://github.com/HenriquesLab/ZeroCostDL4Mic/releases

Tips, tricks and FAQs
https://github.com/HenriquesLab/ZeroCostDL4Mic/wiki/Tips,-tricks-and-FAQs

Zenodo (dataset repository, use *ZeroCostDL4Mic* in the search bar)
https://zenodo.org

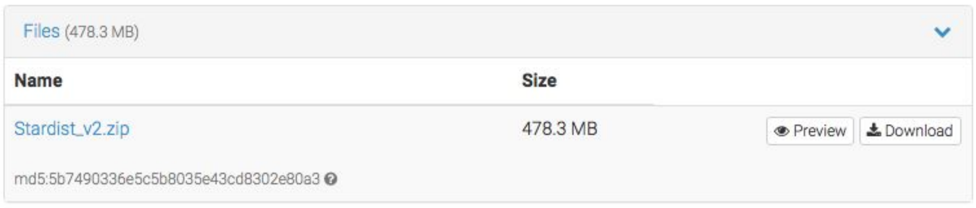
B. Preparing the data for ZeroCostDL4Mic

First, you should decide what task you want to perform using ZeroCostDL4Mic: we currently provide published networks that can perform image segmentation, object detection, denoising, restoration, artificial labelling and image-to-image translation. Please refer to our main Wiki page for info on the currently implemented notebooks. Users should also read our bioRxiv preprint[12] on the general framework to understand whether that is the right tool for the application they have in mind.

B.1. Obtaining the dataset

To start using the provided DL networks, you need some training datasets. You can either test the networks on the example training and test dataset that we provide or generate your own training dataset to be used on your own data. The different pages of our Wiki describe how to acquire the data you need in order to train the different networks.

If you decide to simply test the networks on our example training and test dataset, you can follow the different **Zenodo links that we provide** on our main Wiki page (Figure 3). You can download the whole dataset by clicking on the **Download** button on the Zenodo page. For U-Net, we do not provide a dataset ourselves but point towards an available dataset generated initially for the ISBI 2012 Segmentation challenge[13], see the U-Net page for details.


Files (478.3 MB)

Name	Size
Stardist_v2.zip	478.3 MB

md5:5b7490336e5c5b8035e43cd8302e80a3

Figure 3: Obtaining our example training and test dataset: Zenodo download link, from https://zenodo.org/record/3715492#.Xo8bki2ZNQK

B.2. Uploading the dataset to Google Drive

For Google Colab to have access to these datasets, they need to be uploaded to your Google Drive. You can do that simply by logging into your Google Drive account, clicking on New and then use the *Folder upload* option (Figure 4). **Please ensure that ALL the data is uploaded properly before proceeding further and that the data structure matches the requirements of the notebook to be used. You can find the requirements in section '0. Before getting started'.**

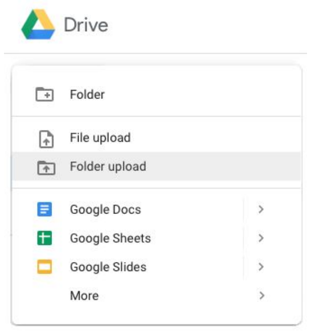
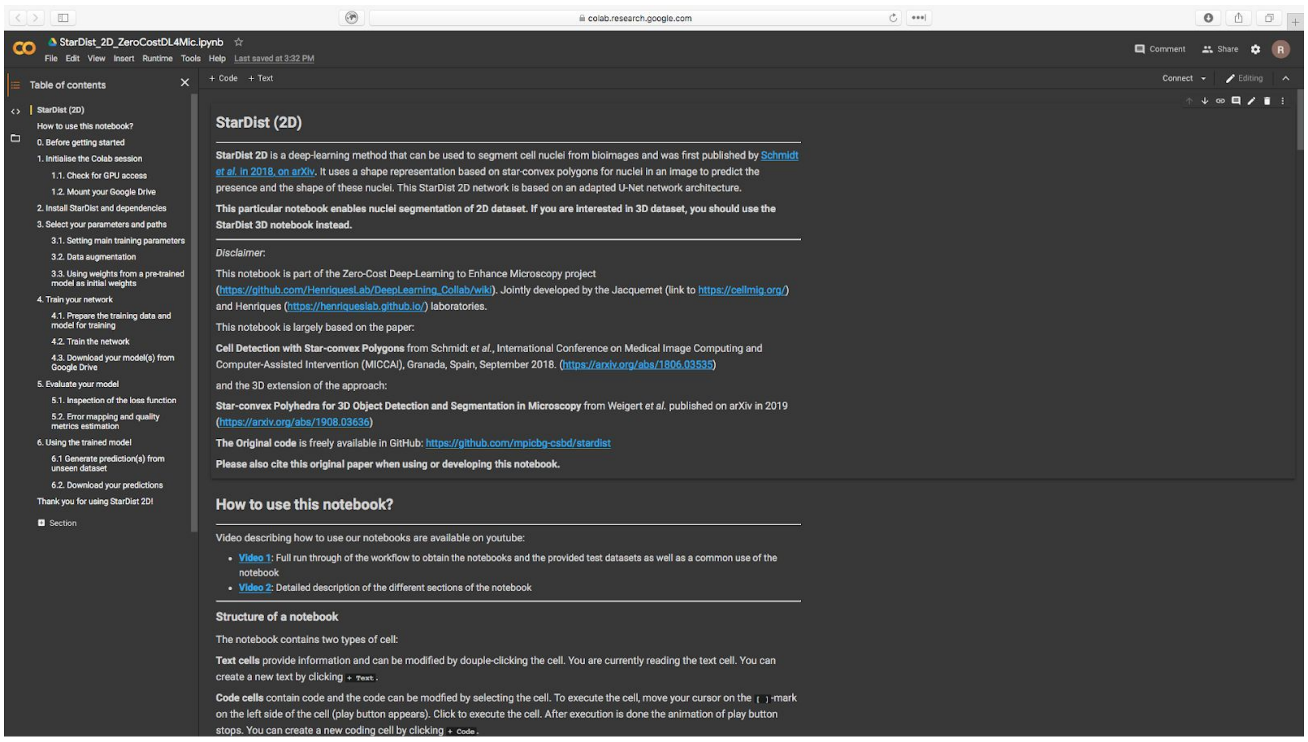


Figure 4: Uploading data to Google Drive.

C. Getting started with the notebook and initialising the Colab session

C.1. Opening the notebook

You can directly open the Colab notebook from our main Wiki page, by clicking on the corresponding **Open In Colab badge** Open in Colab. This will automatically open the notebook in Google Colab (Figure 5). Here, we have chosen to use the Dark mode of the notebook (white text on black background) for improved visibility.


colab.research.google.com
StarDist_2D_ZeroCostDL4Mic.ipynb
File Edit View Insert Runtime Tools Help Last saved at 3:32 PM
Comment Share
+ Code + Text
Connect Editing
Table of contents

StarDist (2D)
How to use this notebook?
0. Before getting started
1. Initialise the Colab session
1.1. Check for GPU access
1.2. Mount your Google Drive
2. Install StarDist and dependencies
3. Select your parameters and paths
3.1. Setting main training parameters
3.2. Data augmentation
3.3. Using weights from a pre-trained model as initial weights
4. Train your network
4.1. Prepare the training data and model for training
4.2. Train the network
4.3. Download your model(s) from Google Drive
5. Evaluate your model
5.1. Inspection of the loss function
5.2. Error mapping and quality metrics estimation
6. Using the trained model
6.1 Generate prediction(s) from unseen dataset
6.2. Download your predictions
Thank you for using StarDist 2D!
Section

StarDist (2D)

StarDist 2D is a deep-learning method that can be used to segment cell nuclei from bioimages and was first published by Schmidt *et al.* in 2018, on arXiv. It uses a shape representation based on star-convex polygons for nuclei in an image to predict the presence and the shape of these nuclei. This StarDist 2D network is based on an adapted U-Net network architecture.

This particular notebook enables nuclei segmentation of 2D dataset. If you are interested in 3D dataset, you should use the StarDist 3D notebook instead.

Disclaimer:

This notebook is part of the Zero-Cost Deep-Learning to Enhance Microscopy project (https://github.com/HenriquesLab/DeepLearning_Collab/wiki). Jointly developed by the Jacquemet (link to https://cellmig.org/) and Henriques (https://henriqueslab.github.io/) laboratories.

This notebook is largely based on the paper:

Cell Detection with Star-convex Polygons from Schmidt *et al.*, International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI), Granada, Spain, September 2018. (https://arxiv.org/abs/1806.03535)

and the 3D extension of the approach:

Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy from Weigert *et al.* published on arXiv in 2019 (https://arxiv.org/abs/1908.03636)

The Original code is freely available in GitHub: https://github.com/mpicbg-csbd/stardist

Please also cite this original paper when using or developing this notebook.

How to use this notebook?

Video describing how to use our notebooks are available on youtube:

- Video 1: Full run through of the workflow to obtain the notebooks and the provided test datasets as well as a common use of the notebook
- Video 2: Detailed description of the different sections of the notebook

Structure of a notebook

The notebook contains two types of cell:

Text cells provide information and can be modified by douple-clicking the cell. You are currently reading the text cell. You can create a new text by clicking `+ Text`.

Code cells contain code and the code can be modfied by selecting the cell. To execute the cell, move your cursor on the `[ ]`-mark on the left side of the cell (play button appears). Click to execute the cell. After execution is done the animation of play button stops. You can create a new coding cell by clicking `+ Code`.

Figure 5: The ZeroCostDL4Mic notebook layout. The notebook shown here is that of StarDist (2D). The table of contents on the left highlights the different sections in the notebook.

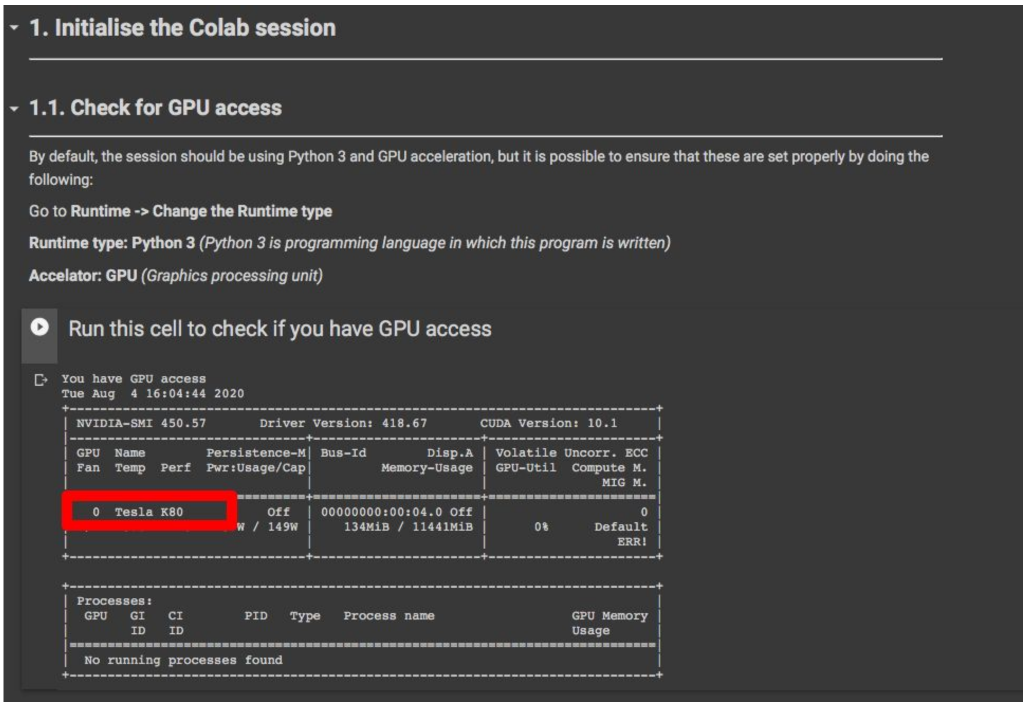
On the left-hand side, you will find a **table of contents** that can be navigated through by clicking on the different items. It is important that you read the **0. Before getting started** section.

At that stage, you will not be able to make changes to the notebook unless you **save a copy of the notebook**. This can be easily done by going into *File* and *Save a copy in Drive...*. If you are signed into a Google Drive account, this will automatically save a copy of the notebook in your Google Drive in the Colab Notebooks folder (Figure 6).

Figure 6: Colab Notebooks folder in Google Drive, where all copies of existing notebooks appear by default. The notebooks can be subsequently moved to other folders.

C.2. Initialising the Colab session

Now you're in the notebook, the first step is to check whether Google Colab will currently allow GPU access and, if so, what type. You can do this by running the **1.1 Check for GPU access** section (Figure 7). The GPU type is described on the last line of the Cell output. Here, we have been allocated a **Tesla K80**.



By default, the session should be using Python 3 and GPU acceleration, but it is possible to ensure that these are set properly by doing the following:

Go to **Runtime -> Change the Runtime type**

Runtime type: Python 3 *(Python 3 is programming language in which this program is written)*

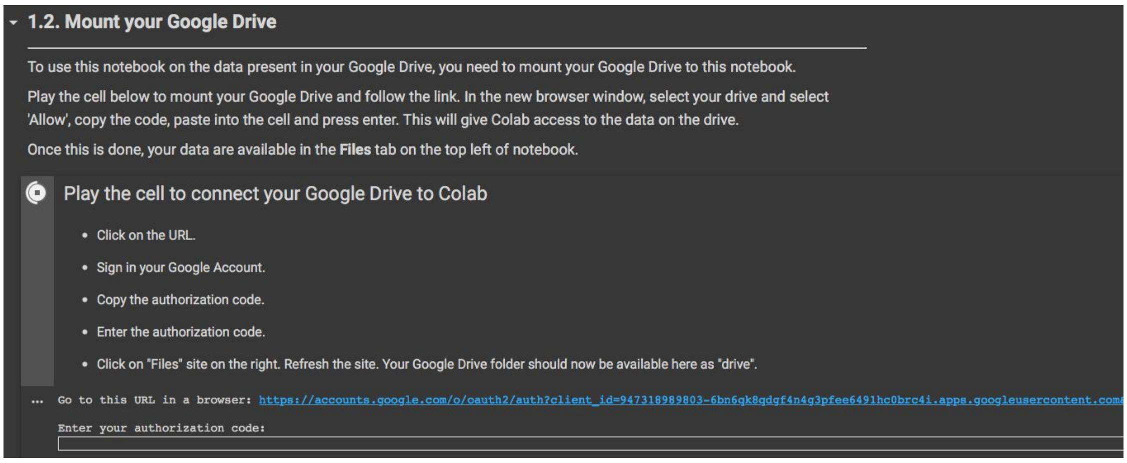
Accelator: GPU *(Graphics processing unit)*

Run this cell to check if you have GPU access

```
You have GPU access
Tue Aug  4 16:04:44 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.57       Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
|                      W / 149W |    134MiB / 11441MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 7: Checking GPU availability and type.

Then, you need to give Google Colab access to the data that you uploaded into your Google Drive. This step is called **Mounting the drive** and is executed in the **1.2 Mount your Google Drive** section (Figure 8). You will need to be logged in and provide an access code accessible via the provided link in the notebook.



1.2. Mount your Google Drive

To use this notebook on the data present in your Google Drive, you need to mount your Google Drive to this notebook.

Play the cell below to mount your Google Drive and follow the link. In the new browser window, select your drive and select 'Allow', copy the code, paste into the cell and press enter. This will give Colab access to the data on the drive.

Once this is done, your data are available in the **Files** tab on the top left of notebook.

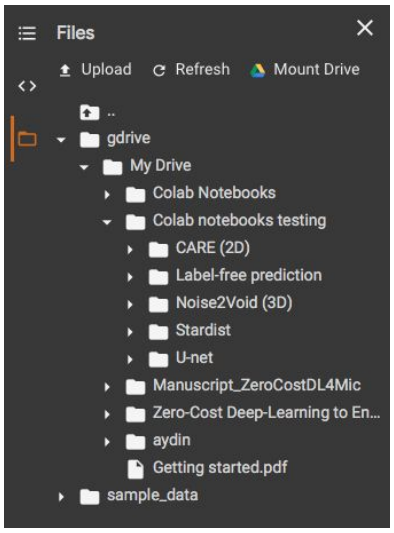
Play the cell to connect your Google Drive to Colab

- Click on the URL.
- Sign in your Google Account.
- Copy the authorization code.
- Enter the authorization code.
- Click on "Files" site on the right. Refresh the site. Your Google Drive folder should now be available here as "drive".

```
... Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com

Enter your authorization code:
```

Figure 8: Mounting Google Drive to the Colab session. The link provided in the cell enables the authentication of the Google Drive and will provide the corresponding authorization code.

Once mounted, the drive and all its content are accessible via the dedicated tab on the left of the page (Figure 9).



```
Files
Upload   Refresh   Mount Drive
..
gdrive
  My Drive
    Colab Notebooks
    Colab notebooks testing
      CARE (2D)
      Label-free prediction
      Noise2Void (3D)
      Stardist
      U-net
    Manuscript_ZeroCostDL4Mic
    Zero-Cost Deep-Learning to En...
    aydin
    Getting started.pdf
sample_data
```

Figure 9: The File tab on the left of the notebook page gives access to the content of the mounted Google Drive.

C.3. Installing network components and related dependencies

Then, we are ready to install the network components and the required dependencies on the virtual machine made available to us by Google Colab. This is done by running the cell corresponding to the **section 2** of the notebook.

Important note: After running section 2, if you already have a trained model and wish to perform quality control on it, you can simply jump to **5. Evaluate your model** section. Similarly, if you only wish to run predictions on a new unseen dataset, you can jump directly to **6. Use the network** section.

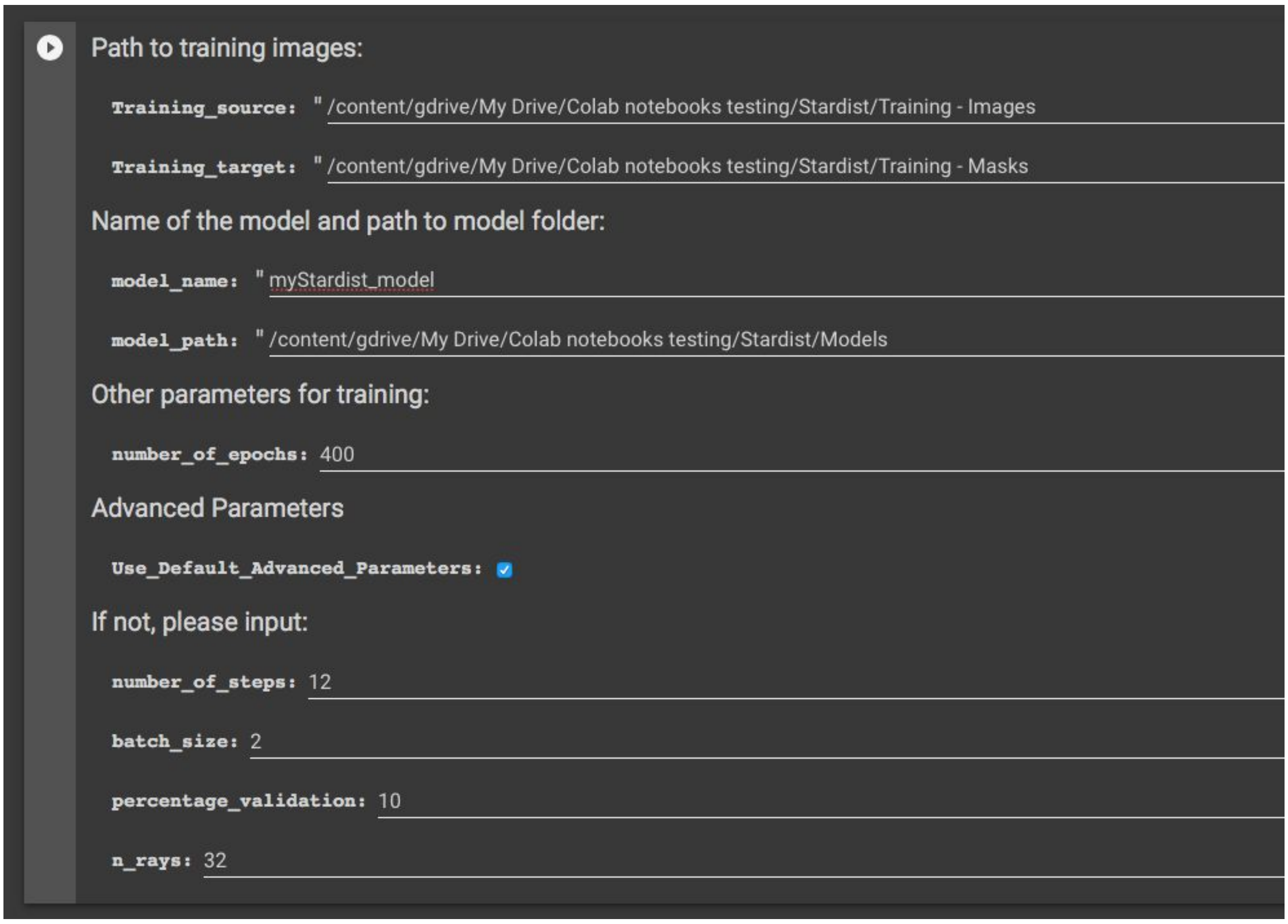
D. Preparing for training

D.1. Setting the main training parameters

Section 3 of the notebook then provides critical information and requires important user input to prepare for the network training. The first part of section 3 highlights and describes the different parameters that the user will need to provide in order to prepare for training the model. This will include **providing the location (path, on the Google Drive) of the different parts of the training dataset** (source and target for fully supervised networks such as U-Net, StarDist, CARE and Label-free prediction, or only the source training dataset for unsupervised networks like Noise2Void). The other parts of the user input will be the training parameters, **e.g. number of epochs, number of steps, batch size.** If you are unsure about the meaning of these parameters, you can also refer to our Glossary page.

Some parameters are currently considered as advanced and can simply be left as their default values if the user wants to get started with a simple training session.

A typical section 3 user input will look like shown in Figure 10. **Do not forget to run this cell for the notebook to take your input into account**.


Path to training images:

Training_source: " /content/gdrive/My Drive/Colab notebooks testing/Stardist/Training - Images

Training_target: " /content/gdrive/My Drive/Colab notebooks testing/Stardist/Training - Masks

Name of the model and path to model folder:

model_name: " myStardist_model

model_path: " /content/gdrive/My Drive/Colab notebooks testing/Stardist/Models

Other parameters for training:

number_of_epochs: 400

Advanced Parameters

Use_Default_Advanced_Parameters: ✓

If not, please input:

number_of_steps: 12

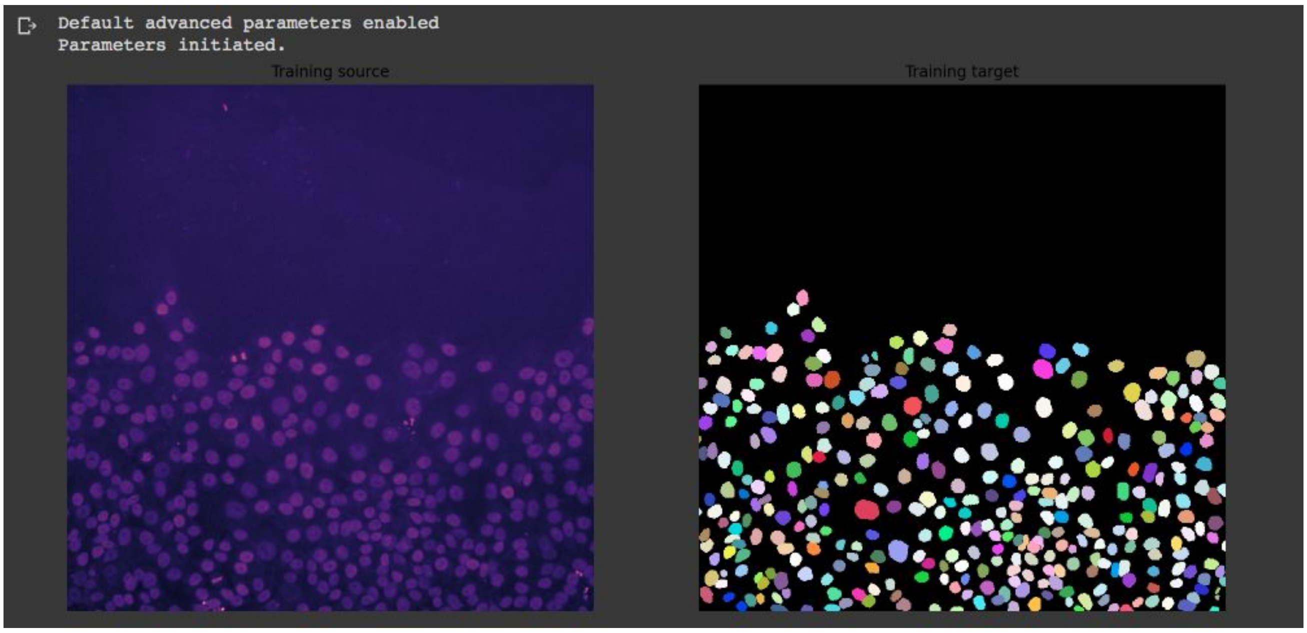
batch_size: 2

percentage_validation: 10

n_rays: 32

Figure 10: Setting the training parameters. Should be provided: paths to training dataset, name for the new model, path to where the new model will be created and saved, network-specific training parameters. Advanced parameters are also accessible.

For some networks, the notebook will highlight a randomly selected pair of source and target images from the training dataset for inspection that the data was uploaded and mounted properly (Figure 11).



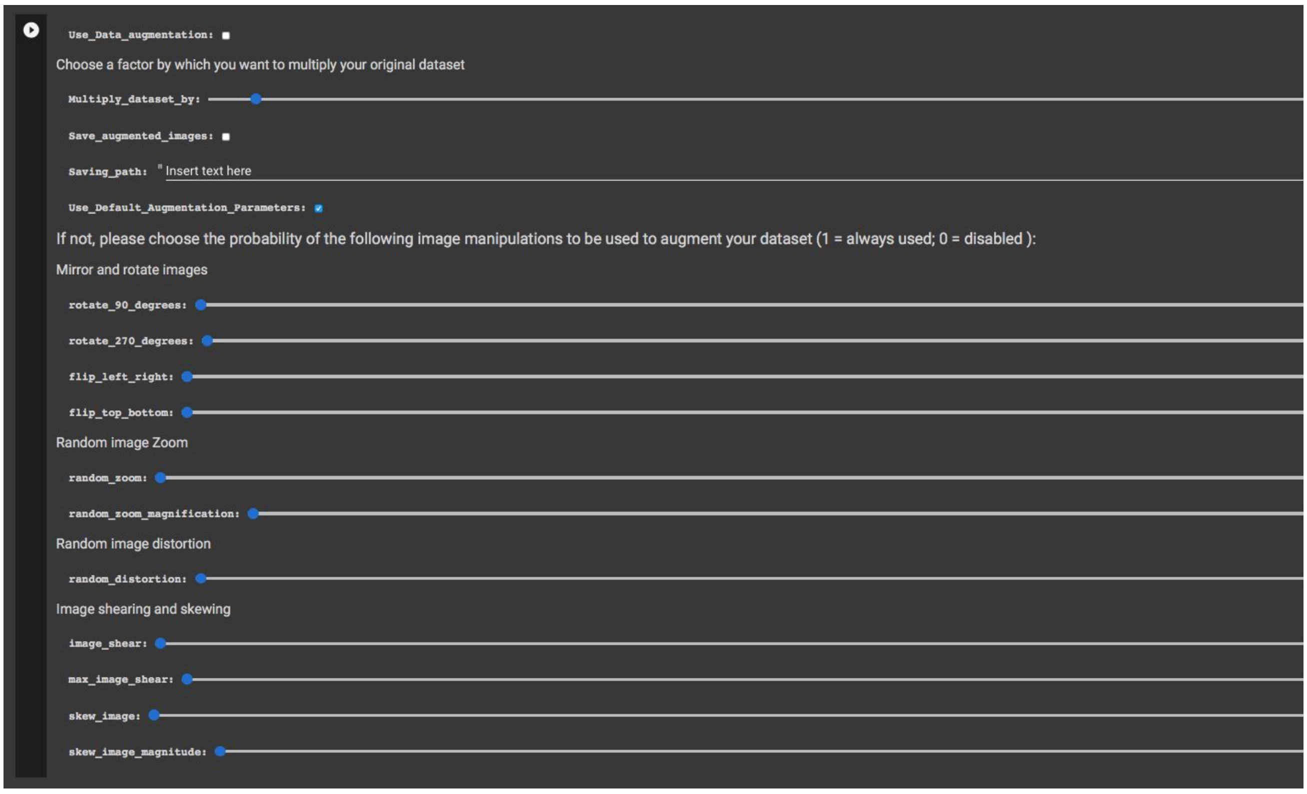
```
Default advanced parameters enabled
Parameters initiated.
```

Figure 11: Display of an example training dataset pair for StarDist 2D. The left image shows the raw data, while the right image represents the nuclei masks. This dataset was taken from the example training dataset that we provide.

D.2. Data augmentation

The following section includes the option of performing **data augmentation** on the currently loaded training dataset. Data augmentation **artificially increases the size of the training dataset** by randomly applying spatial transformations to training image pairs (both source and target), therefore increasing the diversity of the data. The spatial transformations can be flipping with respect to the horizontal or vertical axes, rotations, zoom, shear, skew etc. The random aspect can be included in the choice of which transformations to apply and/or to what extent.

In Figure 12, we show an example of data augmentation notebook section (based on StarDist 2D notebook), highlighting the large choice of transformations that can be used. In particular, for StarDist 2D, **the Augmentor package**[14] **was used** to perform the augmentation.



```
Use_Data_augmentation: ■
Choose a factor by which you want to multiply your original dataset
Multiply_dataset_by:
Save_augmented_images: ■
Saving_path: " Insert text here
Use_Default_Augmentation_Parameters: ☑
If not, please choose the probability of the following image manipulations to be used to augment your dataset (1 = always used; 0 = disabled ):
Mirror and rotate images
rotate_90_degrees:
rotate_270_degrees:
flip_left_right:
flip_top_bottom:
Random image Zoom
random_zoom:
random_zoom_magnification:
Random image distortion
random_distortion:
Image shearing and skewing
image_shear:
max_image_shear:
skew_image:
skew_image_magnitude:
```

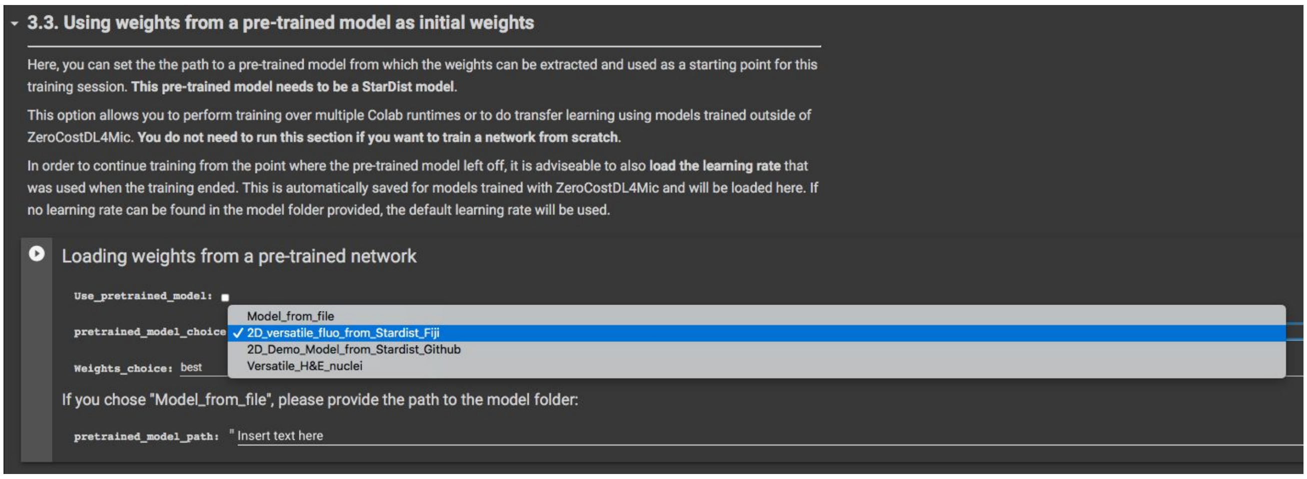
Figure 12: Setting the parameters for Data augmentation used in StarDist 2D. The multiplication factor applied to the training dataset can be selected as well as the probability of each transform to be used.

D.3. Transfer learning

Section 3.3 of the notebook gives access to transfer learning capabilities (see Supplementary Note 5 of our preprint for more details). Briefly, transfer learning allows the user to **load learned weights from a pre-trained model** into the new model prior to training. This allows the new training session to be initialised to an efficient starting point, **benefitting from features learned in the pre-trained model**. Consequently, this can have the advantage of **training more quickly** than with random weight initialisation (requiring fewer epochs) and may **lead to a more stable model**, depending on the pre-trained model used.

Since ZeroCostDL4Mic saves both the “best” (as defined by the set of weights that led to the minimum validation loss any time across the whole range of epochs) and the last set of weights (obtained after the last epoch of the training session), the user can choose to select either of these models to use as pre-trained weights for transfer learning. The corresponding learning rate is automatically loaded too, if possible.

Figure 13 shows an example of the transfer learning section that is available in the StarDist 2D notebook. A number of pre-trained models available online can be directly loaded.



3.3. Using weights from a pre-trained model as initial weights

Here, you can set the the path to a pre-trained model from which the weights can be extracted and used as a starting point for this training session. **This pre-trained model needs to be a StarDist model**.

This option allows you to perform training over multiple Colab runtimes or to do transfer learning using models trained outside of ZeroCostDL4Mic. **You do not need to run this section if you want to train a network from scratch**.

In order to continue training from the point where the pre-trained model left off, it is adviseable to also **load the learning rate** that was used when the training ended. This is automatically saved for models trained with ZeroCostDL4Mic and will be loaded here. If no learning rate can be found in the model folder provided, the default learning rate will be used.

Loading weights from a pre-trained network

Use_pretrained_model:

pretrained_model_choice: ✓ 2D_versatile_fluo_from_Stardist_Fiji

Weights_choice: best

If you chose "Model_from_file", please provide the path to the model folder:

pretrained_model_path: " Insert text here

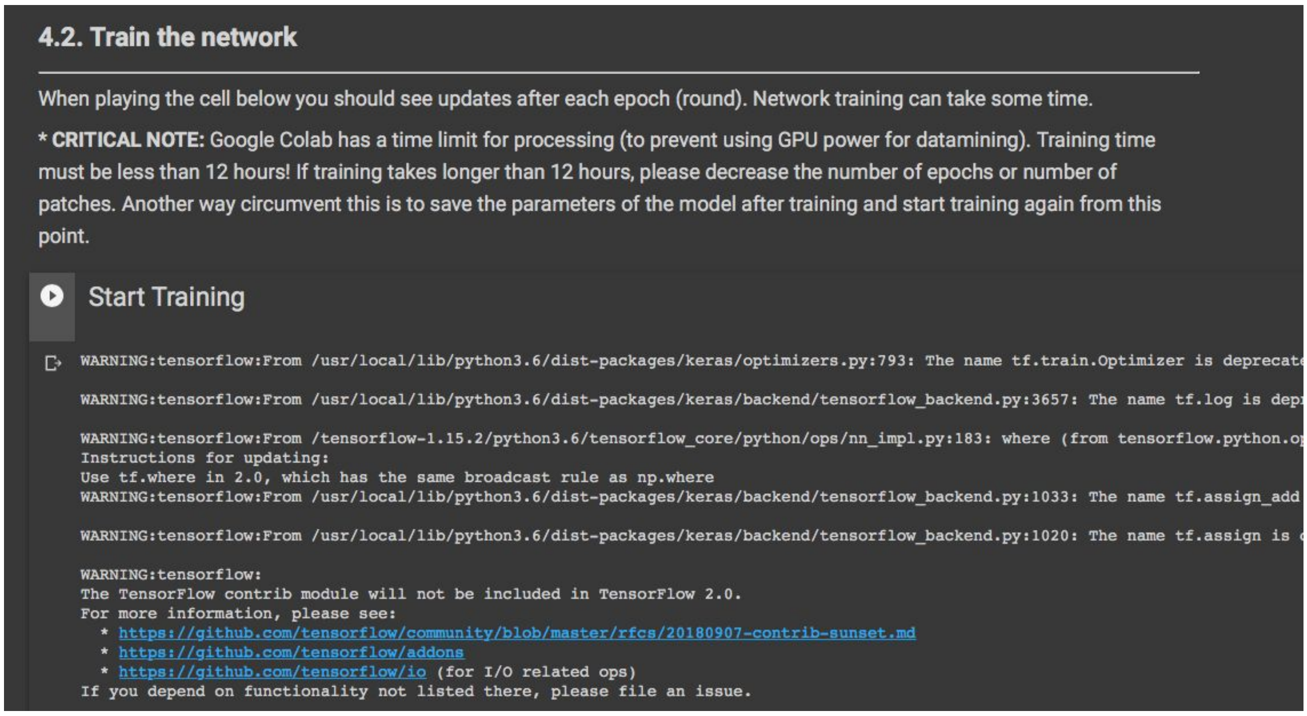
Model_from_file
2D_versatile_fluo_from_Stardist_Fiji
2D_Demo_Model_from_Stardist_Github
Versatile_H&E_nuclei

Figure 13: Transfer learning capabilities in StarDist 2D notebook. A pre-trained model can be used from a locally stored model (Model_from_file) or from StarDist 2D models made available online (e.g. 2D_versatile_fluo_from_Stardist_Fiji)

E. Training the model and performing quality control

E.1. Training

Here we go! Now, you can start the training by running the **4.2 Train the network** section (Figure 14). In some cases, the network may throw some minor warnings about TensorFlow versions, *this is not an issue and can be ignored*. We have enforced specific versions of TensorFlow in order to improve reliability of the notebooks.



4.2. Train the network

When playing the cell below you should see updates after each epoch (round). Network training can take some time.

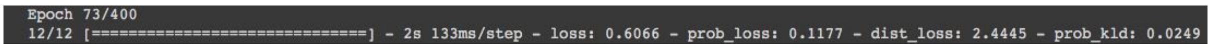
* **CRITICAL NOTE:** Google Colab has a time limit for processing (to prevent using GPU power for datamining). Training time must be less than 12 hours! If training takes longer than 12 hours, please decrease the number of epochs or number of patches. Another way circumvent this is to save the parameters of the model after training and start training again from this point.

Start Training

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:793: The name tf.train.Optimizer is deprecat

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3657: The name tf.log is dep

WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/ops/nn_impl.py:183: where (from tensorflow.python.o
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is 

WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.
```

Figure 14: Training the model with the user-set parameters and training dataset.

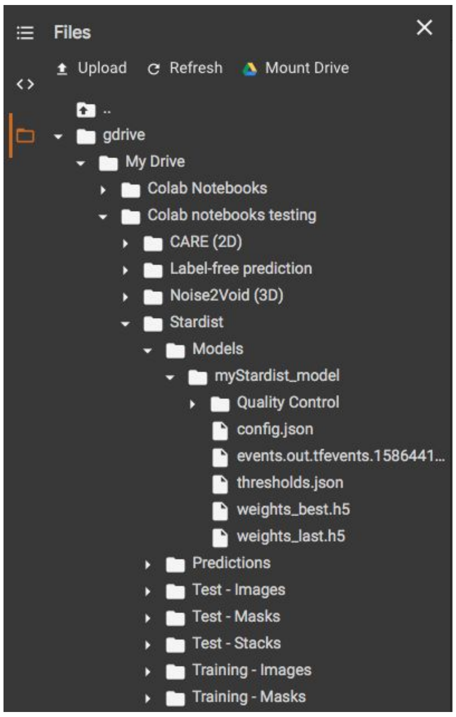
This step can take a few minutes to a few hours depending on the network, training parameters and training dataset size. So be patient! Just ensure that the training time does not exceed 12 h, as this represents the current time limit of a Google Colab session. The time taken by each epoch is displayed for some networks as an output of the network training. This can be used to estimate how long the training will take (in Figure 15 shown as ~2s / epochs and 400 epochs, it should take about ~15min, so yes, you have time for a coffee). The total time taken for the training will be indicated once it is complete. **Pro tip**: **Keep in mind that you can run two Colab sessions in parallel**, so while one model is trained, you can set up another notebook and run it in parallel!



```
Epoch 73/400
12/12 [==============================] - 2s 133ms/step - loss: 0.6066 - prob_loss: 0.1177 - dist_loss: 2.4445 - prob_kld: 0.0249
```

Figure 15: Network training output for each epoch, including the time taken for the epoch. This can help estimating the total time that the training will take.

After training is complete, **the trained model is automatically saved** into the **model_path** folder previously selected (Figure 16). This allows you to set the training running and ignore the risk of running into time-out issues after the training is over. The following sections of the notebooks can be run as a new Colab session if sections 1 and 2 of the notebook are run again.



```
..
gdrive
  My Drive
    Colab Notebooks
    Colab notebooks testing
      CARE (2D)
      Label-free prediction
      Noise2Void (3D)
      Stardist
        Models
          myStardist_model
            Quality Control
            config.json
            events.out.tfevents.1586441...
            thresholds.json
            weights_best.h5
            weights_last.h5
        Predictions
        Test - Images
        Test - Masks
        Test - Stacks
        Training - Images
        Training - Masks
```

Figure 16: The newly trained model is automatically saved in the selected model folder.

You can download the trained model from your Google Drive for safekeeping or for sharing with another user (provided the limitations of using a pre-trained model highlighted in our bioRxiv paper[12]).

E.2. Quality control on the trained model

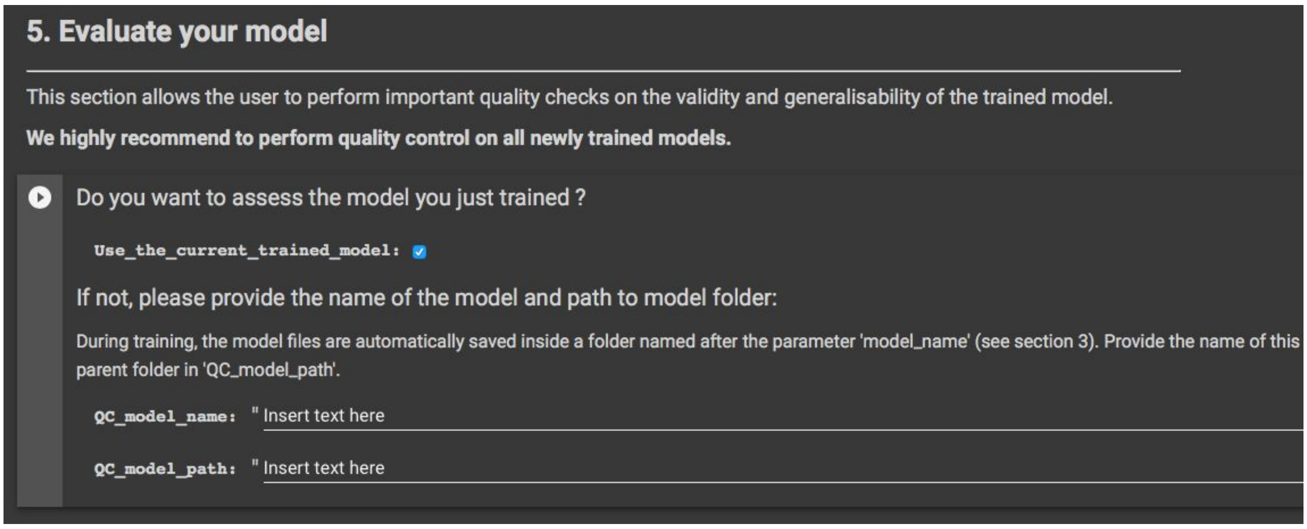
Section 5 of the notebook is very important! It allows you to perform some important quality controls on the trained model. This can be evaluated using two complementary approaches:

(1) Observe the evolution of the loss function as a function of training time. This allows for the inspection of the presence of over- or under-fitting of the model to the training dataset.

(2) Compute quantitative image metrics on the Quality control dataset (also often called "Test dataset" in the machine learning field), where the known ground truth can be compared to the prediction from this trusted dataset.

The quality control datasets should **not** be included in the training dataset during training, otherwise the network performance will be overestimated.

First, you will need to choose whether you are performing Quality control on the current trained model (as obtained in section 4), or whether you will be doing it on a model that was previously trained in a different session (Figure 17). For the latter, you will need to provide a model path and name.



5. Evaluate your model

This section allows the user to perform important quality checks on the validity and generalisability of the trained model.

We highly recommend to perform quality control on all newly trained models.

Do you want to assess the model you just trained ?

Use_the_current_trained_model: ✓

If not, please provide the name of the model and path to model folder:

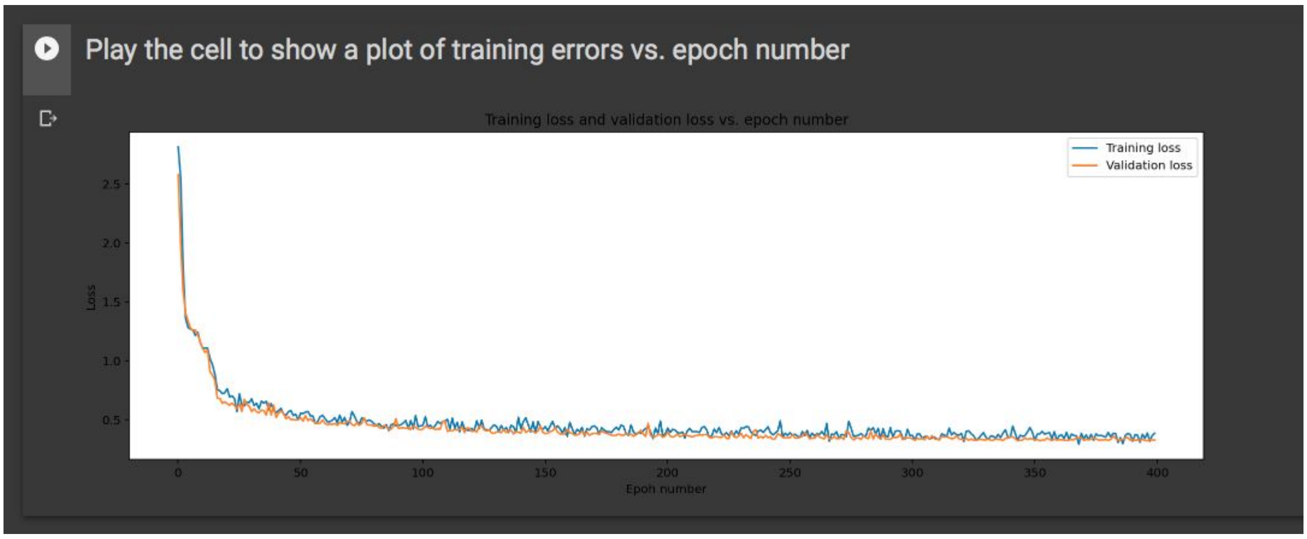
During training, the model files are automatically saved inside a folder named after the parameter 'model_name' (see section 3). Provide the name of this parent folder in 'QC_model_path'.

QC_model_name: " Insert text here

QC_model_path: " Insert text here

Figure 17: Choice of trained model to perform quality control. The path and name of a previously trained model can be given.

Figure 18 shows examples of loss function curves over training time, both from the training and validation dataset. If this doesn't make sense, please see our Glossary page and also this review by Moen *et al.*[15], which explains how to interpret the curves very well.

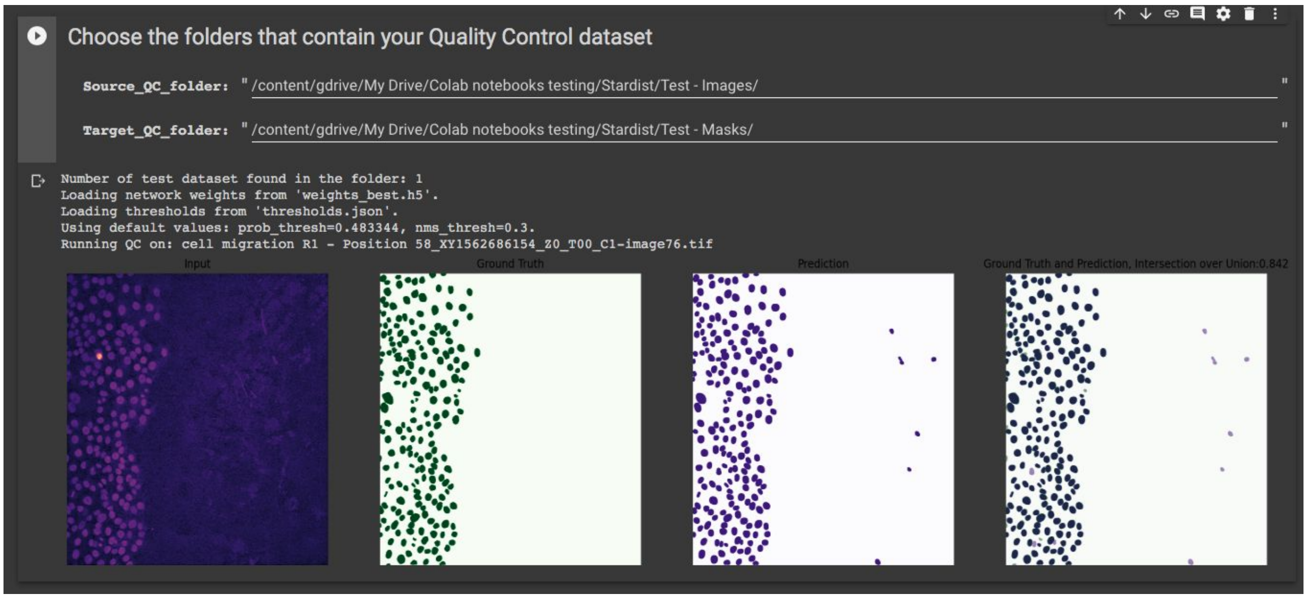


Play the cell to show a plot of training errors vs. epoch number

Figure 18: Evaluation of the model performance by inspection of the training and validation loss curves. A significant deviation over time between the two curves highlights a risk of overfitting of the model to the training dataset.

The second stage of quality control is to see whether the network is able to generalise to unseen dataset, using a quality control dataset. This is performed here by the user providing a dataset with the source images and the equivalent known ground truth targets. Depending on the type of network, we use **Root Squared Error** (RSE) and **Structural SIMilarity** (SSIM) or **Intersection over Union** (IoU) as metrics in order to visually and quantitatively assess whether the model can provide accurate output from unseen data.

Here, in the case of StarDist, because the model predicts a segmentation image, the metric used is Intersection over union (Figure 19). We describe the meaning of these metrics in detail in the Supplementary information of our paper.


Choose the folders that contain your Quality Control dataset

Source_QC_folder: "/content/gdrive/My Drive/Colab notebooks testing/Stardist/Test - Images/"

Target_QC_folder: "/content/gdrive/My Drive/Colab notebooks testing/Stardist/Test - Masks/"

```
Number of test dataset found in the folder: 1
Loading network weights from 'weights_best.h5'.
Loading thresholds from 'thresholds.json'.
Using default values: prob_thresh=0.483344, nms_thresh=0.3.
Running QC on: cell migration R1 - Position 58_XY1562686154_Z0_T00_C1-image76.tif
```

Figure 19: Quantitative assessment of a StarDist model performance using masks overlay and Intersection over Union metric. The image on the left is the source file to be segmented. The green mask represents the manually segmented ground-truth image, the magenta mask is the network prediction, the overlay shows the agreement and disagreement between the two. The Intersection over Union metric is indicated over the overlay image.

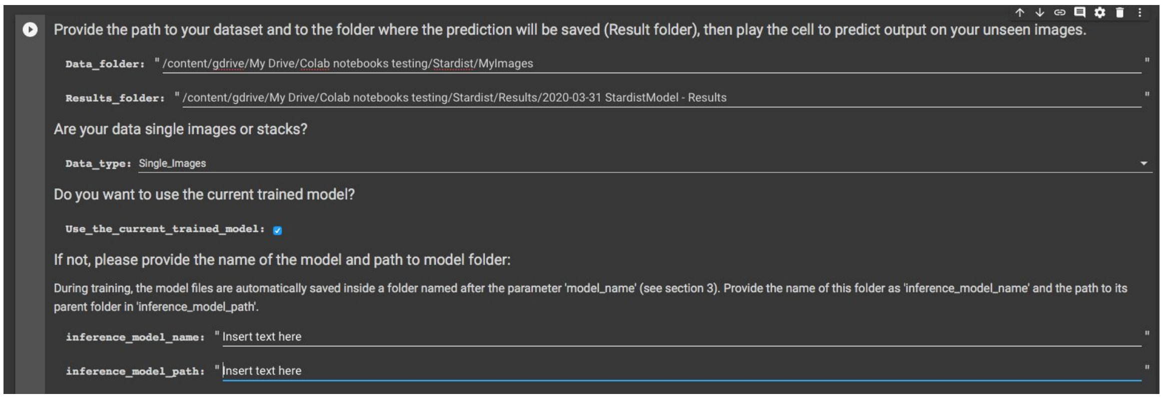
Once the metrics have been calculated the predictions and the metrics maps and indices are all automatically saved in the **model_path** folder in the *Quality Control* folder. You are strongly encouraged to take a look and analyse the data obtained on the test dataset in order to build confidence on the validity of the trained model to analyse the data in the Quality control dataset.

Important note: When wondering if a pre-trained model may be suitable to analyse a new type of data, this section of Quality control can also be used to quantify and estimate if the model will perform well with this new type of data.

F. Using the trained model to obtain predictions on new data

The last step, available in **Section 6**, is what you have been waiting for the whole time: It is the generation of predictions from unseen data using the trained model. This can be performed by giving the path to the Google Drive directory containing the new unseen data that you wish to run the prediction on (Figure 20).

A different model from the currently loaded model can also be used by providing the name and path to the model to be used.



Provide the path to your dataset and to the folder where the prediction will be saved (Result folder), then play the cell to predict output on your unseen images.

Data_folder: " /content/gdrive/My Drive/Colab notebooks testing/Stardist/MyImages "

Results_folder: " /content/gdrive/My Drive/Colab notebooks testing/Stardist/Results/2020-03-31 StardistModel - Results "

Are your data single images or stacks?

Data_type: Single_Images

Do you want to use the current trained model?

Use_the_current_trained_model: ✓

If not, please provide the name of the model and path to model folder:

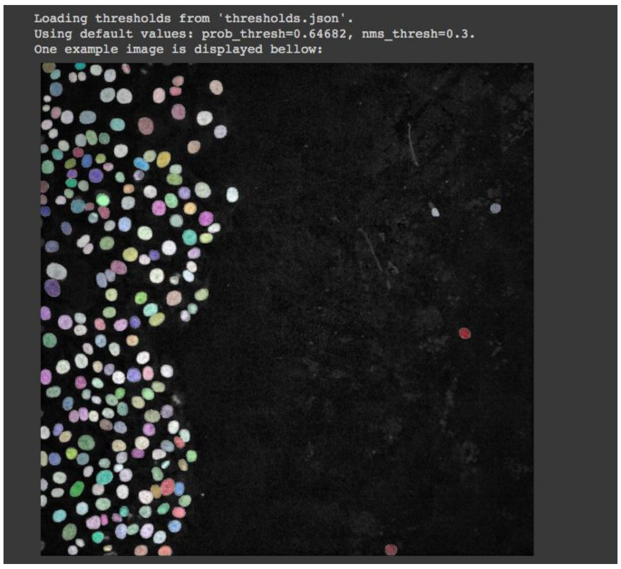
During training, the model files are automatically saved inside a folder named after the parameter 'model_name' (see section 3). Provide the name of this folder as 'inference_model_name' and the path to its parent folder in 'inference_model_path'.

inference_model_name: " Insert text here "

inference_model_path: " Insert text here "

Figure 20: User input to run predictions on unseen data. In the particular case of StarDist, it is possible to run it directly on stacks (or on single images), and therefore there is an option to select this.

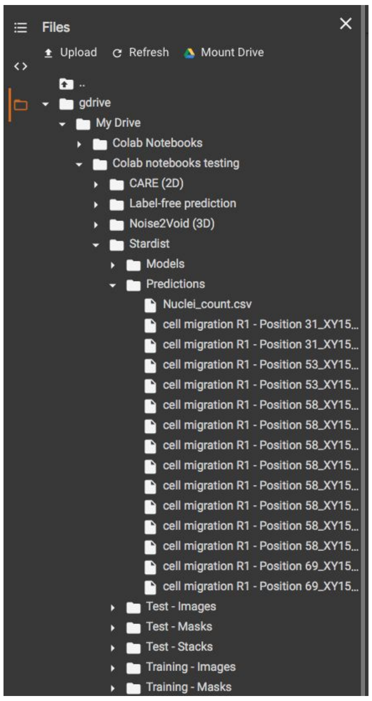
The notebook will show you an example prediction chosen at random from the set of data provided (Figure 21). In this example, we trained a StarDist model and the notebook shows an overlay of the original input image with the mask image obtained from the prediction.



```
Loading thresholds from 'thresholds.json'.
Using default values: prob_thresh=0.64682, nms_thresh=0.3.
One example image is displayed bellow:
```

Figure 21: Prediction output display. The notebook chooses a random dataset to display for the visual assessment of the prediction.

The predictions obtained from the unseen data are now available in the Results folder as chosen earlier and can therefore be downloaded from your Google Drive for further analysis (Figure 22).



```
Files
Upload  Refresh  Mount Drive
..
gdrive
  My Drive
    Colab Notebooks
    Colab notebooks testing
      CARE (2D)
      Label-free prediction
      Noise2Void (3D)
      Stardist
        Models
        Predictions
          Nuclei_count.csv
          cell migration R1 - Position 31_XY15...
          cell migration R1 - Position 31_XY15...
          cell migration R1 - Position 53_XY15...
          cell migration R1 - Position 53_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 58_XY15...
          cell migration R1 - Position 69_XY15...
          cell migration R1 - Position 69_XY15...
        Test - Images
        Test - Masks
        Test - Stacks
        Training - Images
        Training - Masks
```

Figure 22: The predictions are automatically saved in the selected Results folder.

G. Final notes

Many thanks for trying out **ZeroCostDL4Mic**! Whether you find it useful, intuitive or difficult and buggy, we want to hear from you and always welcome constructive feedback. Feel free to report issues in the GitHub issue page or simply Tweet your results using #ZeroCostDL4Mic.

H. References

1. Ronneberger, O., Fischer, P. & Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *ArXiv150504597 Cs* (2015).
2. Çiçek, Ö., Abdulkadir, A., Lienkamp, S. S., Brox, T. & Ronneberger, O. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation. in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016* (eds. Ourselin, S., Joskowicz, L., Sabuncu, M. R., Unal, G. & Wells, W.) vol. 9901 424–432 (Springer International Publishing, 2016).
3. Redmon, J. & Farhadi, A. YOLO9000: Better, Faster, Stronger. in 7263–7271 (2017).
4. Krull, A., Buchholz, T.-O. & Jug, F. Noise2Void - Learning Denoising from Single Noisy Images. *ArXiv181110980 Cs* (2019).
5. Schmidt, U., Weigert, M., Broaddus, C. & Myers, G. Cell Detection with Star-Convex Polygons. in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018* (eds. Frangi, A. F., Schnabel, J. A., Davatzikos, C., Alberola-López, C. & Fichtinger, G.) vol. 11071 265–273 (Springer International Publishing, 2018).
6. Weigert, M., Schmidt, U., Haase, R., Sugawara, K. & Myers, G. Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy. 8 (2020).
7. Weigert, M. *et al.* Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nat. Methods* **15**, 1090–1097 (2018).
8. Ounkomol, C., Seshamani, S., Maleckar, M. M., Collman, F. & Johnson, G. R. Label-free prediction of three-dimensional fluorescence images from transmitted-light microscopy. *Nat. Methods* **15**, 917–920 (2018).
9. Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. Image-to-Image Translation with Conditional Adversarial Networks. *ArXiv161107004 Cs* (2018).

10. Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *ArXiv170310593 Cs* (2018).

11. Nehme, E., Weiss, L. E., Michaeli, T. & Shechtman, Y. Deep-STORM: super-resolution single-molecule microscopy by deep learning. *Optica* **5**, 458 (2018).

12. von Chamier, L. *et al. ZeroCostDL4Mic: an open platform to simplify access and use of Deep-Learning in Microscopy*. http://biorxiv.org/lookup/doi/10.1101/2020.03.20.000133 (2020) doi:10.1101/2020.03.20.000133.

13. Arganda-Carreras, I. *et al.* Crowdsourcing the creation of image segmentation algorithms for connectomics. *Front. Neuroanat.* **9**, (2015).

14. Bloice, M. D., Roth, P. M. & Holzinger, A. Biomedical image augmentation using Augmentor. *Bioinformatics* **35**, 4522–4524 (2019).

15. Moen, E. *et al.* Deep learning for cellular image analysis. *Nat. Methods* **16**, 1233–1246 (2019).