Part of this code is from : https://github.com/BPHO-Salk/PSSR

Results obtained with this code : https://github.com/thibaut1998e/centriole_internship

1)General overview of the code :

train.py : train a neural network with fastai and save a model (most of this code is from PSSR)

inference.py : test the model and save the results

transformations.py : various transformations to apply on images (crop, padd, convolution, resizing …)

apply_transformation.py : contains a method which applies a list of transformations to all the images in a folder and save the outputs.

paths_definitions.py : paths constants

main.py : apply degradation model, train a neural network and test it

plot_graphs.py : plot several curves

topaz_commands : methods which execute topaz command lines

split_source_data_train_valid.py

files to clean data : invert_channels.py , delete_list.py, delete2D_images.py

convert_lif.py, convertles .lif in. tiff

dossier utils (mostly from PSSR) : definition of several architectures, metrics and losses (feature loss, l1 loss)

2. Preparation of data

The first step is to get .tiff images of centrioles using .lif files. To do that use the python file *convert_lif.py* by specifying at the top of it the source and destination folders. Set also the variable *nb_channels*. The destination folder is then split in 2 separate folders deconv and raw each of them are split in 2 channels c1 and c2.

Then you can clean the data by removing unwanted data, to do this use the python file *delete_images* by specifying the folder and the path of the txt file containing names of undesired images. You can also delete 2D images, which are often undesired with the code *delete2Dimages*. For some images channels are inverted by mistake, use similarly the code *invert_channels* to fix it.

The next step is to split the data into 2 subfolders train and valid, use the code *split_source_data_train_valid.py*. You can specify the proportion of validation.

3. Create virtual environements

a) Virtual environment to train and test a fastai model :

```
conda create -n fastai_env python==3.7.3
```

```
conda activate fastai_env
pip install fastai==1.0.55
pip install scikit-image
pip install torch==1.1.0
pip install torchvision==0.3.0
pip install libtiff
pip install czifile
```

b) Virtual environment to train a topaz model

Another environment is needed to use codes which executes topaz commands lines, to install topaz and create the environment follow their guide at : https://github.com/tbepler/topaz.

Then you are ready to use the code ! 😊

4. Train and test a fastai model for super resolution

Before strarting using it, modify the constant path variables in the file *paths_definition* if needed.

A method which is useful for many purposes in the code is the method *transforms()* in the python file *apply_transformation.py*. It takes as input a source folder *folder_in*, a destination folder *folder_out* and a list of transformations. A transformation is a function which takes as input a 2D or 3D array (representing an image) and returns a 2-3D array or a list of 2-3D array. The method applies to all images in *folder_in* the list of transformation and stores the outputs in *folder_out*. The method is recursively called on each subfolders so that folder_out keeps the architecture of folder_in.

Source images, got from the code *convert_lif,* are 3D images but we want 2D images to train our model. Moreover, to train a model with fastai , it is required for all images to have the same shape, and the shape is slightly different from one image to another. We then need to take one (or several) section(s) for each image and crop them. To do that we call *transforms()* with the method *cross_section*(), in the file *apply_transformations.py*. You can call this function in the __main__ of this file.

Then to add spots to HR images use the method *add_spots()*. It is the only transformation which takes times so its better to do it only once.

Once it's done you can run the code *main.py*. It does 3 things : apply the degradation model to HR images by convolving, resizing and normalizing. Then train and save a model with fastai by calling the code *train.py*. Finnaly it tests the model with the code i*nference.py*. LR images are generated from the folder HR_spots, using also the method *transforms()*, called by *conv_resize_norm()*. You can choose training and testing parameters in this file. Results are saved in the folder *test_results*.

Important parameters :

HR_folder : path of the folder containing HR images. It should be split in 2 subfolders *train* and *valid*.

spot_fold : LR images are generated from this folder if *spots* is set to true. It should also be split in train valid, the resulting LR folder is then also split in train and valid. The LR folder is saved in the folder *training_sets*., but if you don't want to save it here you can still change its path location with the variable *LR_folder*.

test_images_path : path of 3D images on which the prediction is made

size : fastai applies rescaling so that HR images have a shape of (size, size), this parameter is recommanded to be set to the size of images in HR_folder (rescaling will have no effect)

Model : architecture used, currently only wnresnet works well.

model_name : it is defined accordingly to the parameters set but you can still choose it by setting this parameter.

skip_train : if you only want to test a model you can set this boolean to True, the model at the location {models}/{model_name}.pkl should already exists.

Raw : if you train on raw data, it is recommanded to set it to true so that it would be taken into account in the definition of the name of the model.

Loss parameters :

alpha : power in L_α losss which is defined as :

$L_\propto = \frac{1}{n} * \|y - f(x)\|_\propto$

y : real HR image x : LR image n : number of pixel in the image

f : neural network f(x) : predicted image

feature_loss : if True a feature loss with vgg feature extractor is used, the feature loss (or content loss) is defined as :

$CL_{l,\beta} = \frac{1}{n} \left\|\varphi^l(y) - \varphi^l(f(x)\right\|_\beta$

φ : vgg neural network used to extract feature

l = parameter *nb_layer_vgg* : number of layer used in vgg feature extractor, the state of the neural network φ at layer *l* represents the features.

β = parameter *betha* in the code.

You can combine linearly these 2 losses with the parameter *prop* :

$M_{\alpha,\beta,l,p}$ = *prop*$*CL_{\beta,l}$ + (1-*prop*)* L_α

Use topaz detection in inference.py :

topaz : if true prediction is made only on patches centered on centrioles other pixels are set to 0, size of the patches cropped is *size*.

topaz2 : if true makes 3D detection of centriole in LR images, and returns an output image for each centriole.

If *topaz* or *topaz2* is set to true you shoul provide the path of the txt file containing the centers for all the slices of each images in the folder *test_images_path*. You will see how to compute this txt file in the next section.

5. Predict position of particles with topaz classifier

Methods which executes topaz commands are located in the file *topaz_commands.py*

Topaz data are located in the folder *data_centriole_detection,* which contains 2 folders train and valid and hand-labelled positions of centriole in 2 .txt files, *valid_labels.txt* and *train_labels.txt*.

The method *train_topaz_model* uses these data to train a model which detects position of centrioles by executing a topaz command line. It first rescales if needed the images so that the centriole size in pixel matches the receptieve field of their neural network, then trains a model. You can visualize training curves with the method *plot_metrics* in *plot_loss.py*.

Then to predict the postition of centers and associated scores, you can either use a model that you have trained or use the model *centriole_detection_epoch100.sav*, already trained on this data set. To do that call the method *compute_center_particles()* in the same file. Results are saved in a txt.

After that, you can use this txt file to crop images on these predicted centers. There are 2 methods for that one for 2D images and one for 3D images.

For 2D images call the method *crop_topaz()* in *apply_transformations.py*. This method first calls the method *get_center_dict_from_txt()* which returns a dictionnary from the txt file computed by topaz. Keys are names of images and values a list of *Center* object, a center is defined by 2 attributes its score and its position (a couple). Then images are cropped with this dictionnary.

For 3D images call *crop_threeD_topaz()*. It uses the transformation *crop_threeD_with_center_dict(),* which has several options.

This transformation detects the position in 3D of centrioles given positions in 2D for all the slices. You need to specify a radius : center closer than radius, in two different slices or in the same slice, are considered to represent the same particule.

If the option *average* = True is given the center in 3D is computed as the average of all centers (closer than raidus) in 2D. This is a weighted average by the scores. So if you set a threshold lower than 0 in *get_center_dict_from_txt()* make sure that you have normalized the scores (with option *normalize* in *get_center_dict_from_txt())*.

You can save graphs of intensity through slices with the option *save_figure*. In these figures we can visualize 2 things : slices of local minima of intensity correspond to end slices of centrioles. Besides local maxima of intensity are slices where the centriole is the most visible. We can use this fact to cropp images in a smarter way.

First by setting the option cut = True, patches are cropped between 2 local minima of intensity.

Secondly with average = False, the 3D center is defined as the 2D center of the slice of highest intensity.