Recommended guidelines to make your code accessible

The following guidelines were written to support those participants who want to make their deep learning code available to future users. Their purpose is to provide specific guidelines so that the required files presented in the previous section can be complemented with additional and useful information. This would result in a more accessible package, facilitating its widespread usage. There are three main points targeted in the following lines:

- Reproducibility of the training
- Fine tuning / retraining to process new images
- Submission to other challenges / (bioimage) model zoo / integration with public libraries

Note that the guidelines are written for Python, but the same logic is applicable to other software.

Contents

---

1. Key points for accessibility and usability of the code

To guarantee the usability of a deep learning code, the first two executions should be at least provided. For that we recommend to clearly specify in your code the following items:

1.1 Data

The folder where the data is stored for the training / test / new inference. This folder should have the same structure as the one provided in the CTC. For example:

```
Fluo-C3DL-MDA231
|- 01
|- 01_GT
        |- SEG
        |- TRA
|- 01_ST
        |- SEG
```

Any additional detail for the correct execution of the code should be given (data organization, data subfolders name, images bit depth, filenames or where to download the data for the training and validation).

1.2 Code

Usually called as src / sw / name of the method. The folder should have all the functions / scripts needed for at least (1) data pre-processing, (2) training, (3) data post-processing and (4) inference. The code provided should be self contained, i.e. steps such as data normalization, data augmentation, loss functions, data/image generators, network architecture, callbacks, or thresholding process should all be given in the code.

1.3 Environment set up

List of requirements (requirements.txt), setup.py, a yaml file or a bash code to set up a virtual environment should be provided. This is critical for libraries such as Pytorch, TensorFlow, Keras, PILs, Scipy or OpenCV.

Recommended: Write a bash code `prepare_software.sh` to create a new python virtual environment with all the required dependencies. We recommend to use `requirements.txt` as well so it can be easily used also in Google Colaboratory.

- Anaconda prepare_software.sh. Run it as `bash -i prepare_software.sh` to avoid problems.

```
set -e
# deactivate virtualenv if already active
if command -v conda deactivate > /dev/null; then conda deactivate; fi
# python 3.8 or higher
if test -f anaconda3/envs/your personal env; then
  echo "virtualenv already exists, skipping"
else
  conda create -n your_personal_env python>=3.8
fi
# activate virtual environment
conda activate your personal env
# install python dependencies or use requirements.txt
pip3 install -r requirements.txt
set +e
```

- Python + virtualenv prepare_software.sh. Run it as `bash repare_software.sh`

```
set -e
# deactivate virtualenv if already active
```

```
if command -v deactivate > /dev/null; then deactivate; fi
# python 3.8 or higher
if test -f your_personal_env/bin/activate; then
  echo "virtualenv already exists, skipping"
else
  virtualenv -p python3 your_personal_env
fi
# activate virtual environment
source your_personal_env/bin/activate
# install python dependencies or use requirements.txt
pip3 install -r requirements.txt
set +e
```

1.4 Training and test

Provide an example of how to train the network so the model can be used with different data. This could be done in two different ways; either using a python script (train.py) or a Google Colab notebook (train.ipynb). A **Google Colab notebook**, in particular, would be the ideal choice to enable easy usage by scientists from other expertises not so familiar with programming, such as biologists.

Note: the goal of this file is to show how to train the model using new data, rather than obtaining exactly the same model as the one submitted to CTC.

- Python script (execution example):
  `python train.py --data_dir '../path/to/the/training' --epochs 1000 --learning_rate 0.001 --pretrained-weights '../path/to/submitted/model.h5'`
- Google Colaboratory notebook which uses Google's GPU free service (similar to a Jupyter notebook). Look at StarDist, deepcell-tf or DenoiSeg for ideal examples, or at the following short example:

```
!git clone https://github.com/username/ctc_code.git

# Explain how the data should be organized.

DATAFOLDER = '/content/drive/My Drive/path/to/data/directory'

import os

os.chdir('/content/ctc_code/SW/code')

from data import prepare_training_data

input_data, output_data = prepare_training_data(DATAFOLDER)

from train_networks import train_unet

model = train_unet(input_data, output_data,pretrained_weights = True,…)
```

Some examples of "easy to use" code repositories that fit to the cited key points:

- https://github.com/mpicbg-csbd/stardist
- https://github.com/vanvalenlab/deepcell-tf
- https://github.com/juglab/DenoiSeg
- https://gitlab.fi.muni.cz/xlux/deepwater
- https://github.com/GeorgeSeif/Semantic-Segmentation-Suite
- https://github.com/oist/Usiigaci
- https://github.com/matterport/Mask_RCNN

2. Combine the CTC submission with a public GitHub repository

Public repositories as the ones in GitHub (https://github.com/) contribute to the spread of code among developers and allow the interaction between different users such as the communication of issues, improvement of the code and the development of extensions.

Additionally, the code could be cited even when the developers did not publish their results (https://github.com/AASJournals/Tutorials/blob/master/Repositories/CitingRepositories.md) and there is always a license associated with the code (https://help.github.com/en/enterprise/2.18/user/github/creating-cloning-and-archiving-repositories/licensing-a-repository).

2.1 CTC submission structure

This is the general structure of the CTC submission.

- Data (e.g. Fluo-N2DL-HeLa) **(Required for the CTC submission)**
  01_RES folder
- SW (software) **(Required for the CTC submission)**
  Code
  Trained models
  requirements.txt
  my_script_for_FluoNsDLHela.bat / my_script_for_FluoNsDLHela.sh

Trained models folder (description)

This folder will be used to keep trained models that can be loaded for test and can also serve as the place where newly trained models are saved.

There are two ways of saving a result: (1) save the entire architecture with the trained weights or (2) store an instance of the trained network which only contains the trained weights.

To avoid any problem when loading a trained model, it is recommended that any result is stored as an instance of the weights (e.g. using keras.save_weights or tf.sess.save()). Then, a user could build the network architecture using the code provided and load the desired weights.

2.2 GitHub repository

Aiming for the code to be compatible with libraries such as the bioimage model zoo (https://bioimage.io/), model zoo (https://modelzoo.co/), Kaggle (https://www.kaggle.com/) and CTC (http://celltrackingchallenge.net/), we provide next some guidelines to create an adequate repository.

2.2.1 Creation

Make a **public** GitHub repository that contains **Data** and **code folders.** As it is a code repository, **`/Data/`** and **`/trained models/`** should be empty. It is also recommended to avoid any binary file, as they can be included in a **release** afterwards.

Note: This is an example of how to publish a repository synchronized with the CTC submission. If you already have a repository, check that the accessibility & usability features are present and include the SW folder of the CTC submission.

- GitHub repository (e.g. 'ctc_code')
  Data (empty)
  src (software)
    - code folders
    - CTC submission (SW)
      - Trained models (empty)
      - my_script_for_FluoNsDLHela.bat / my_script_for_FluoNsDLHela.sh

  requirements.txt / setup.py / create_env.yml / create_env.sh
  train.py / train.ipynb (script / notebook)
  test.py / test.ipynb (script / notebook) or infer.py / infer.ipynb (script / notebook)
  README.md (optional)
  LICENSE.md (optional)

README.md: Gathers the information about specific CUDA or Opencl version, links to the training data, citations, or further explanations needed to make the code working.

LICENSE.md file is automatically created by GitHub as soon as there is any specified License.

2.2.2 Release of the CTC submission

Once the code has been submitted to the CTC, **make a release**:

- The .bat/.sh files and all their dependencies for inference should be exactly the same as the ones available at the CTC webpage (http://celltrackingchallenge.net/participants/).
- Use an informative name, e.g. “CTC submission version X.Y.Z date xxx.xx.xx”.
- Provide a short explanation of what is contained inside, link to the datasets that were evaluated with this code, where it was submitted and any other information you find useful.
- Indicate from which branch of the code you want to have the release (it is asked by GitHub).
- Attach the instances of the network that were submitted to the CTC for evaluation, i.e. `/Trained models/` folder.
- Publish the release.

From now on, you can change your code, but this release will remain the same, representing the exact algorithm you submitted to the CTC. You can make as many releases as you want from your code and use them to spread it into different platforms.