TomoEncoders Documentation

Release 0.3

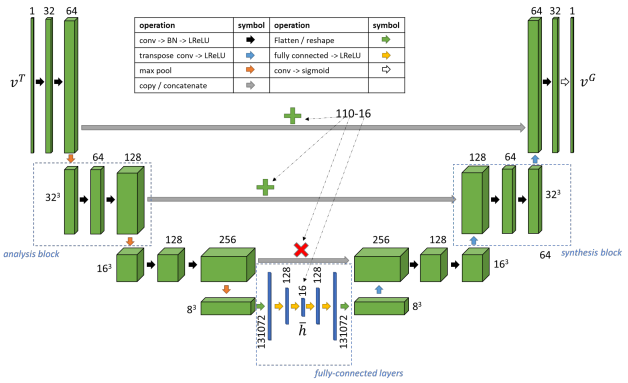
Argonne National Laboratory

Dec 01, 2021

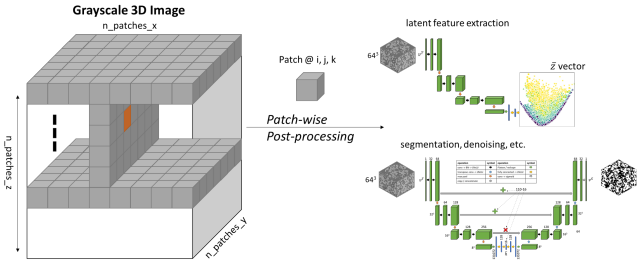
CONTENTS

This code (1) trains a convolutional auto-encoder (CAE) against pairs of grayscale and binarized data (2) extracts the latent space of the CAE and projects it to 2D space. For certain configurations of the architecture it is possible to order the clusters based on porosity metrics. More details are in our upcoming paper at IEEE-ICIP 2021. See the jupyter notebooks section for results. Data will be available shortly.

The architecture is defined with synthesis and analysis blocks inspired by the 3D U-net (note the skip connections).



he encoder-decoder (segmenter or denoiser) is trained by sampling patches of data from the 3D image pairs (grayscale and binarized image) around random coordinates to generate training data. Then, the encoder part is separated, and latent vectors are projected to 2D by PCA. Once trained, patches can be sampled from a given list of coordinates in the grayscale volume to identify morphological similarities.



CHAPTER

ONE

FEATURES

- List here
- the module features

CHAPTER

TWO

CONTRIBUTE

- Documentation: https://github.com/aniketkt/TomoEncoders/tree/master/doc
- Issue Tracker: https://github.com/aniketkt/TomoEncoders/docs/issues
- Source Code: https://github.com/aniketkt/TomoEncoders/project

CHAPTER

THREE

CONTENT

3.1 About

This section describes what the TomoEncoders is about.

3.2 Install

This section covers the basics of how to download and install TomoEncoders. We recommend you to install the Anaconda Python distribution.

Contents:

3.2.1 Installing from source

Clone the TomoEncoders from GitHub repository:

```
git clone https://github.com/aniketkt/TomoEncoders.git TomoEncoders
```

then:

```
cd TomoEncoders
python setup.py install
```

3.3 Train

3.3.1 Train Encoders for Porosity

```
python bin/train_porosity_encoders.py
```

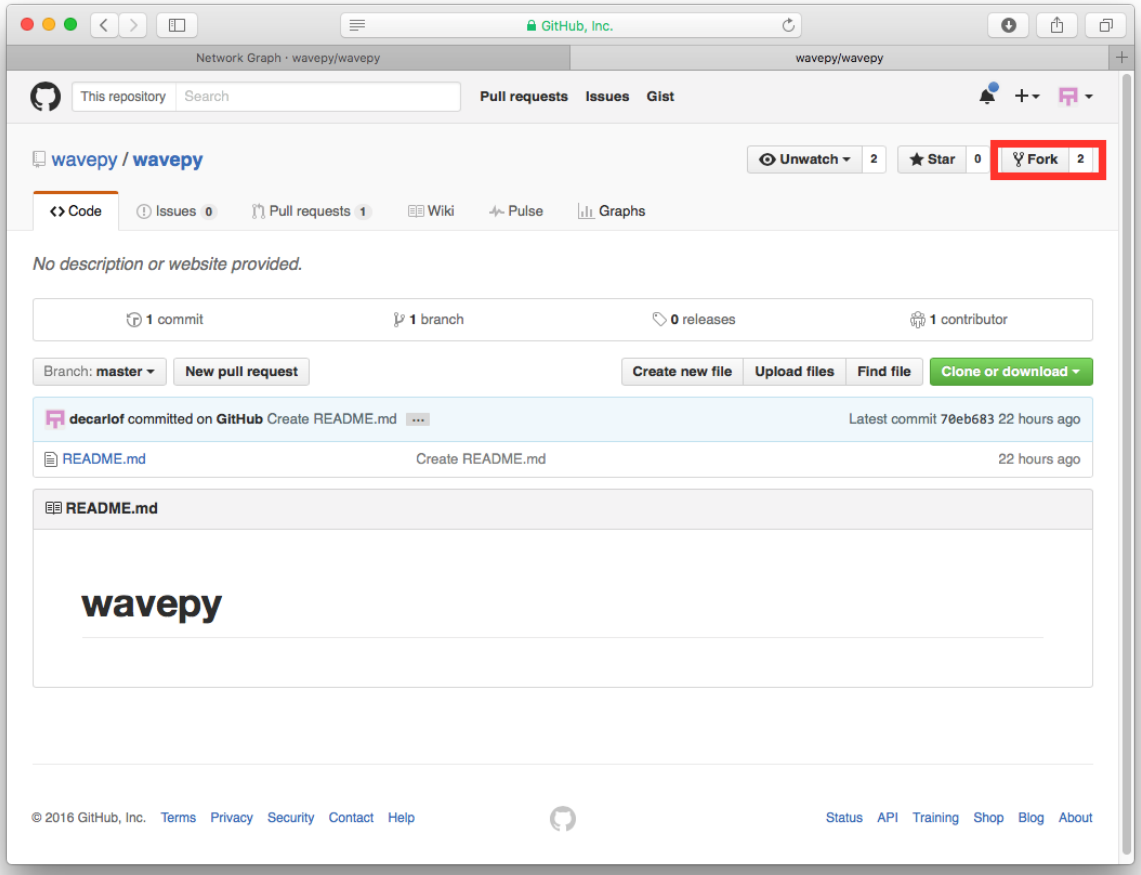
3.4 Development

This section explains the basics for developers who wish to contribute to a project mantained on GitHub and using the fork / pull request mechanism for accepting developer contributions.

Contents:

3.4.1 Fork the repository

The project is maintained on GitHub, which is a version control and a collaboration platform for software developers. To start first register on GitHub and fork the TomoEncoders repository by clicking the **Fork** button in the header of the TomoEncoders repository:

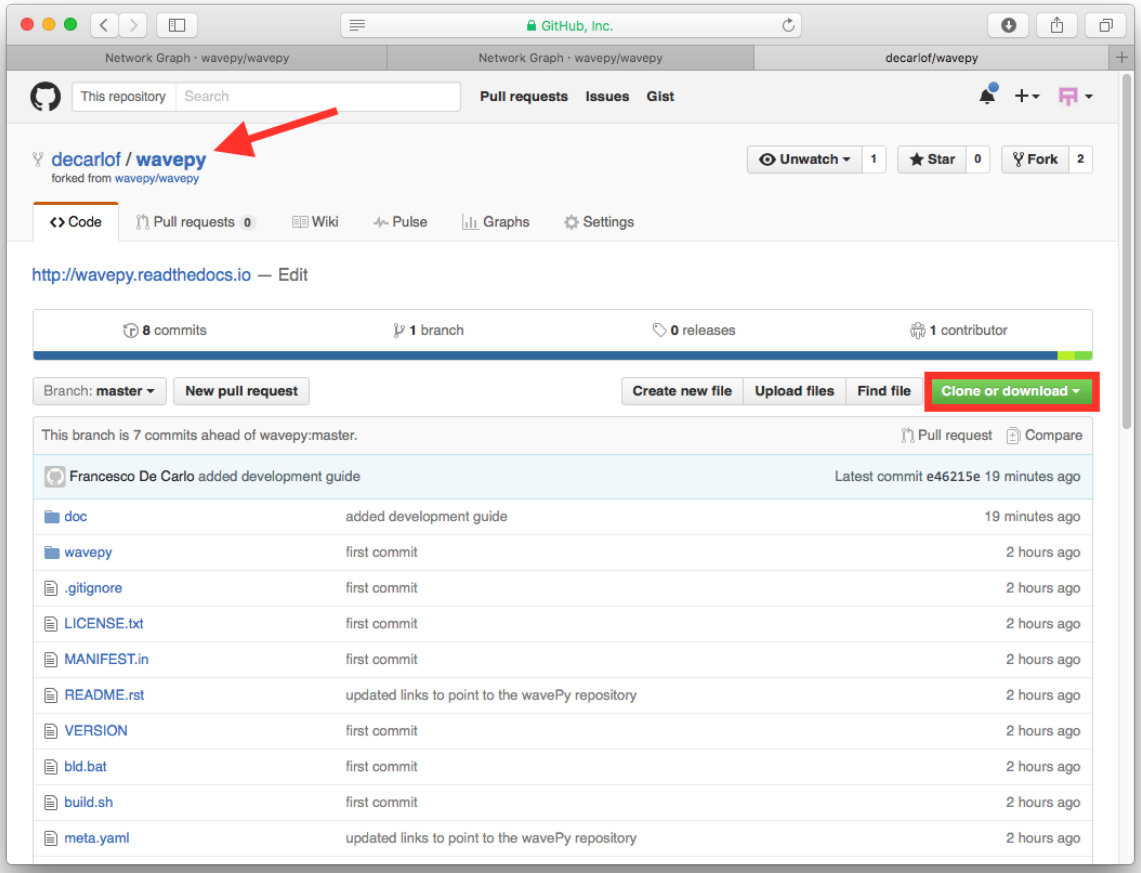


This successfully creates a copy of the project in your personal GitHub space.

3.4.2 Clone the repository

The next thing you want to do is to clone the repository you just created in your personal GitHub space to your local machine.

You can do this by clicking the **Clone in Desktop** button in the bottom of the right hand side bar:



This will launch the GitHub desktop application (available for both Mac and Win) and ask you where you want to save it. Select a location in your computer and feel comfortable with making modifications in the code.

3.4.3 Coding conventions

We try to keep a consistent and readable code. So, please keep in mind the following style and syntax guidance before you start coding.

First of all the code should be well documented, easy to understand, and integrate well into the rest of the project. For example, when you are writing a new function always describe the purpose and the parameters:

```
def my_awesome_func(a, b):
    """
    Adds two numbers.

    Parameters
    ----------
    a : scalar (float)
        First number to add

    b : scalar (float)
        Second number to add
```

(continues on next page)

(continued from previous page)

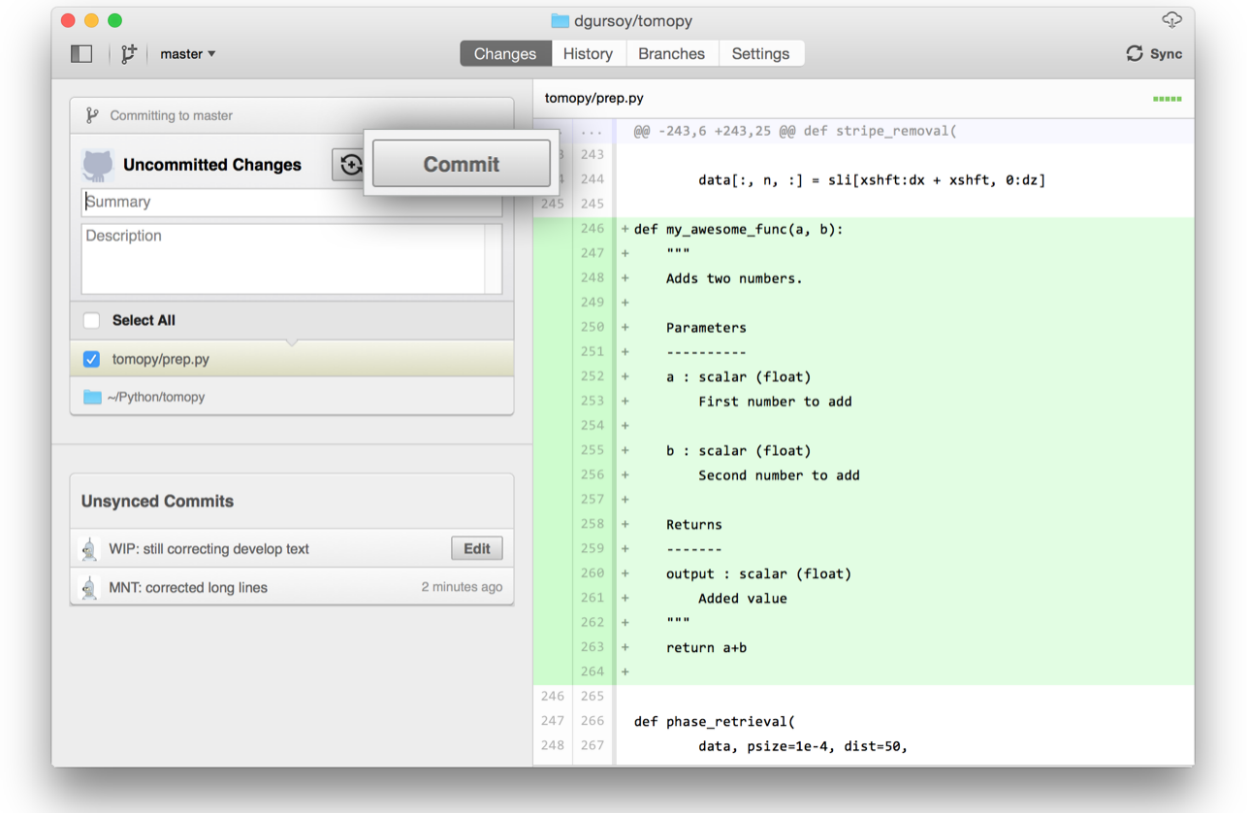
```
    Returns
    -------
    output : scalar (float)
        Added value
    """
    return a+b
```

3.4.4 Package versioning

We follow the X.Y.Z (Major.Minor.Patch) semantic for package versioning. The version should be updated before each pull request accordingly. The patch number is incremented for minor changes and bug fixes which do not change the software's API. The minor version is incremented for releases which add new, but backward-compatible, API features, and the major version is incremented for API changes which are not backward-compatible. For example, software which relies on version 2.1.5 of an API is compatible with version 2.2.3, but not necessarily with 3.2.4.

3.4.5 Commiting changes

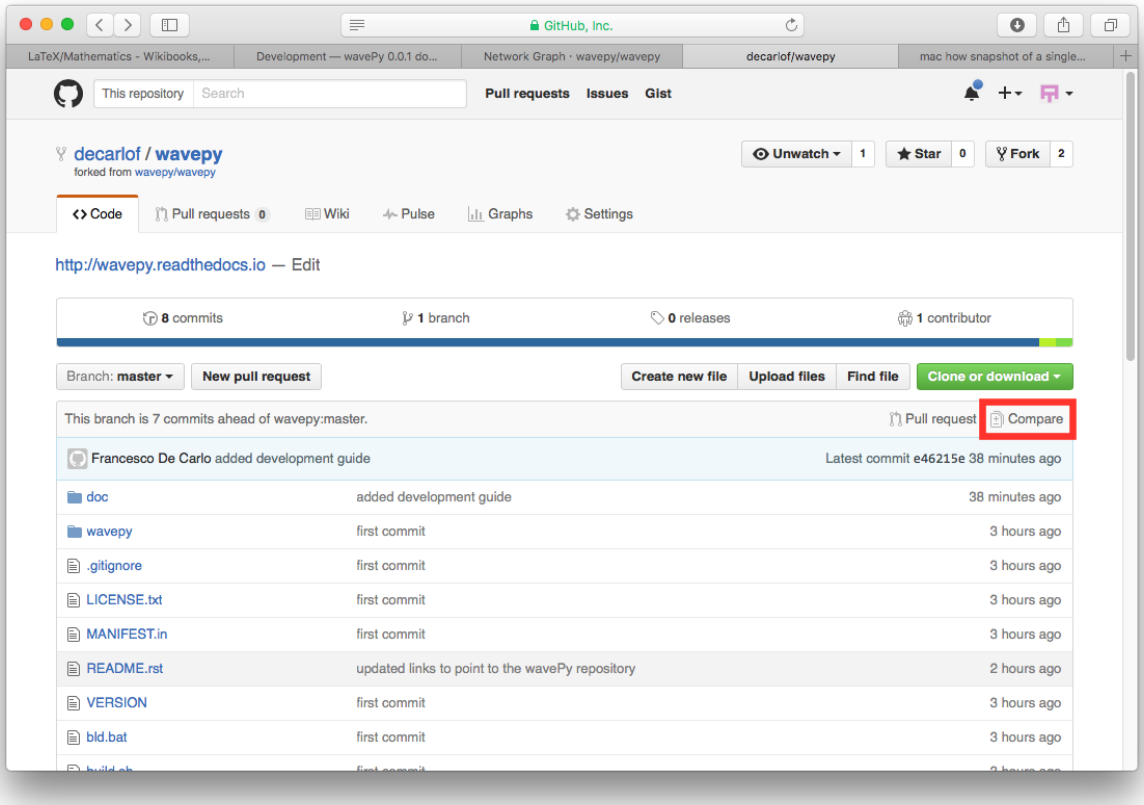
After making some changes in the code, you may want to take a *snapshot* of the edits you made. That's when you make a *commit*. To do this, launch the GitHub desktop application and it should provide you all the changes in your code since your last commit. Write a brief *Summary* and *Description* about the changes you made and click the **Commit** button:



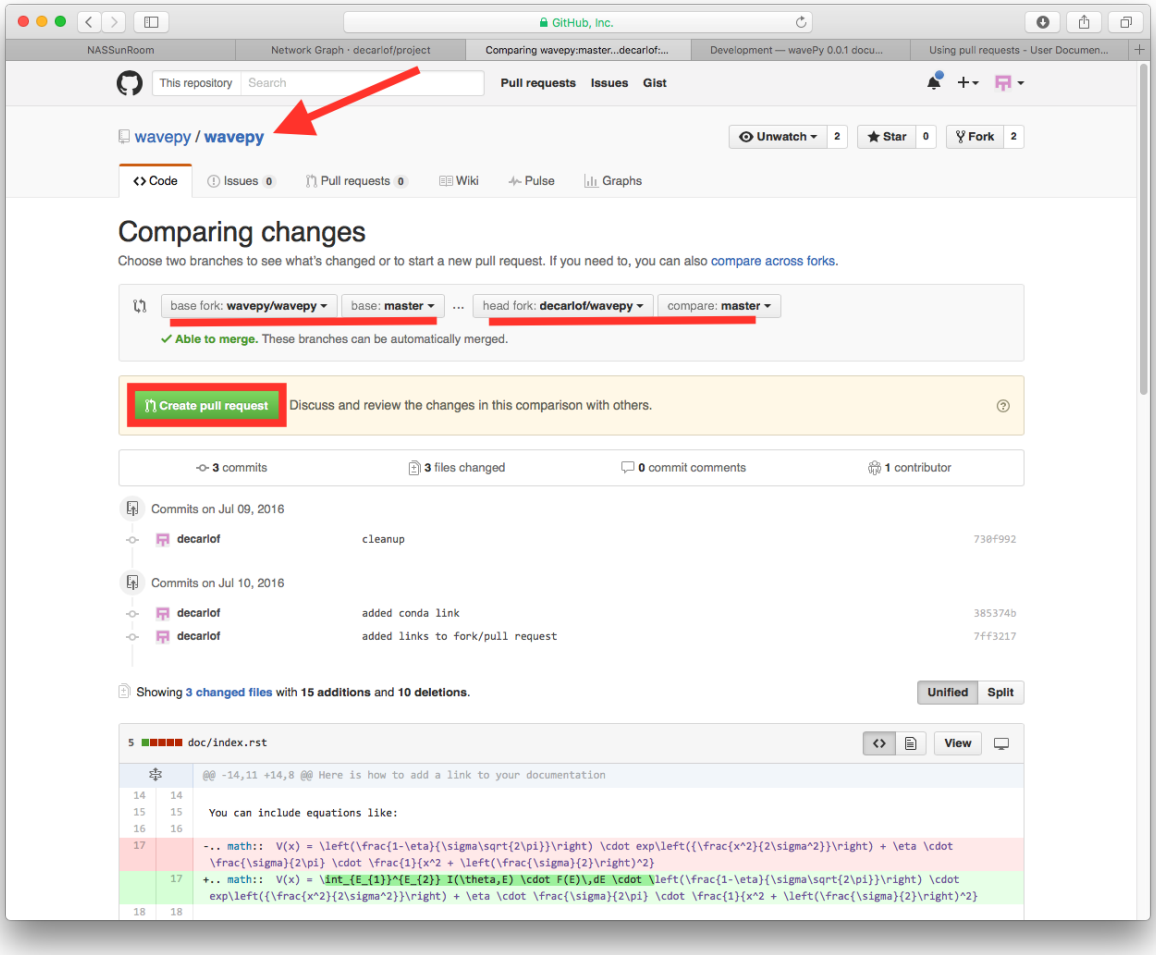
You can continue to make changes, add modules, write your own functions, and take more *Commit snapshots* of your code writing process.

3.4.6 Contributing back

Once you feel that the functionality you added would benefit the community, then you should consider contributing back to the project. For this, go to your online GitHub repository of the project and click on the *compare* button to compare, review and create a pull request.



After clicking on this button, you are presented with a review page where you can get a high-level overview of what exactly has changed between your forked branch and the original project repository. When you're ready to submit your pull request, click **Create pull request**:



Clicking on **Create pull request** sends you to a discussion page, where you can enter a title and optional description. It's important to provide as much useful information and a rationale for why you're making this Pull Request in the first place.

When you're ready typing out your heartfelt argument, click on **Send pull request**.

You're done!

3.5 API reference

tomo_encoders Modules:

3.5.1 tomo_encoders.labeling.detect_voids

Functions:

calc_patch_size(base_size, multiplier)	

continues on next page

Table 1 – continued from previous page

export_voids(vol_seg, n_max_detect[, TIMEIT])	
to_regular_grid(sub_vols, p3d, ...)	(1) select patches where voids exist (2) rescale to upsample_value
upsample_sub_vols(sub_vols, upsampling_fac)	all sub-volumes must have same shape.

tomo_encoders.labeling.detect_voids.**calc_patch_size**(*base_size*, *multiplier*)

tomo_encoders.labeling.detect_voids.**export_voids**(*vol_seg*, *n_max_detect*, *TIMEIT=False*)

tomo_encoders.labeling.detect_voids.**to_regular_grid**(*sub_vols*, *p3d*, *target_patch_size*, *target_vol_shape*, *upsample_fac*)

(1) select patches where voids exist (2) rescale to upsample_value

tomo_encoders.labeling.detect_voids.**upsample_sub_vols**(*sub_vols*, *upsampling_fac*, *TIMEIT=False*, *order=1*)

all sub-volumes must have same shape. upsampling_factor applies to all 3 dimensions equally.

to-do: split into chunks, and try higher-order

3.6 Examples

Here we describe what the examples are doing. You can cite with [].

3.6.1 Example 01

This section contains the example_01 script.

Download file: `example_01.py`

3.7 Credits

3.7.1 Citations

We kindly request that you cite the following article [] if you use project.

3.7.2 References

PYTHON MODULE INDEX

t

INDEX