unetseg Documentation

Release unknown

Dymaxion Labs

May 19, 2022

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 unetseg

U-Net semantic segmentation for satellite imagery

*This digital tool is part of the catalog of tools of the **Inter-American Development Bank**. You can learn more about the IDB initiative at code.iadb.org*

1.1.1 Description

A set of classes and CLI tools for training a semantic segmentation model based on the U-Net architecture, using Tensorflow and Keras.

This implementation is tuned specifically for satellite imagery and other geospatial raster data.

1.1.2 Documentation

- Stable
- Latest

1.1.3 Bugs / Questions

- Report bugs/feature requests
- Ask questions

1.1.4 Contributing

Bug reports and pull requests are welcome on GitHub at the issues page. This project is intended to be a safe, welcoming space for collaboration, and contributors are expected to adhere to the Contributor Covenant code of conduct.

Made with contrib.rocks.

1.1.5 License

This project is licensed under Apache 2.0. Refer to LICENSE.txt.

1.2 Installation

1.2.1 Stable release

Use pip to install from PyPI:

Install from pip:

```
pip install unetseg
```

1.2.2 From source

The source for satproc can be installed from the GitHub repo.

```
python -m pip install git+https://github.com/dymaxionlabs/unetseg.git
```

To install for local development, you can clone the repository:

```
git clone https://github.com/dymaxionlabs/unetseg.git
```

If you don't have Poetry installed, follow these instructions first.

Then, install all dependencies. Poetry will automatically create a virtual environment for you:

```
cd unetseg
poetry install
```

Whenever you want to bring the latest changes, just run `git pull` from the cloned repository.

1.3 unetseg

1.3.1 unetseg package

Subpackages

unetseg.console namespace

Submodules

unetseg.console.predict module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the [options.entry_points] section in setup.cfg:

console_scripts = fibonacci = unetseg.skeleton:run

Then run *python setup.py install* which will install the command *fibonacci* inside your current environment. Besides console scripts, the header (i.e. until _logger...) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

unetseg.console.predict.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

unetseg.console.predict.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

unetseg.console.predict.**run**()
Entry point for console_scripts

unetseg.console.predict.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

unetseg.console.train module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the [options.entry_points] section in setup.cfg:

console_scripts = fibonacci = unetseg.skeleton:run

Then run *python setup.py install* which will install the command *fibonacci* inside your current environment. Besides console scripts, the header (i.e. until _logger...) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

unetseg.console.train.**main**(*args*)
Main entry point allowing external calls

Parameters **args** ([*str*]) – command line parameter list

unetseg.console.train.**parse_args**(*args*)
Parse command line parameters

Parameters **args** ([*str*]) – command line parameters as list of strings

Returns command line parameters namespace

Return type argparse.Namespace

unetseg.console.train.**run**()
Entry point for console_scripts

unetseg.console.train.**setup_logging**(*loglevel*)
Setup basic logging

Parameters **loglevel** (*int*) – minimum loglevel for emitting messages

Submodules

unetseg.evaluate module

unetseg.evaluate.**plot_data_generator**(*num_samples: int = 3*, *fig_size=(20, 10)*, ***, *train_config: unetseg.train.TrainConfig*, *img_ch: int = 3*)

Plots some samples from a data generator.

Parameters

- **num_samples** (*int*) – Number of samples to plot.
- **fig_size** (*tuple*) – Figure size.
- **img_ch** (*int*) – Number of channels.
- **train_config** (TrainConfig) – Training configuration object.

unetseg.evaluate.**plot_data_results**(*num_samples: int = 3*, *fig_size=(20, 10)*, ***, *predict_config: unetseg.predict.PredictConfig*, *img_ch: int = 3*, *n_bands: int = 3*)

Plots some samples from the results directory. :param num_samples: Number of samples to plot. :type num_samples: int :param fig_size: Figure size. :type fig_size: tuple :param img_ch: Number of channels. :type img_ch: int :param predict_config: Prediction onfiguration object. :type predict_config: PredictConfig

unetseg.postprocess module

unetseg.postprocess.**crop_image**(*img: numpy.ndarray*, *margin_ratio: float*) → numpy.ndarray

Center crop an image, with a margin of `margin_ratio`

unetseg.postprocess.**get_bounds_from_image_files**(*image_files: List[str]*) → Tuple[float]

Get bounds from all images, and transform to pixels based on the affine transform

unetseg.postprocess.**merge**(*images_dir: str*, *output_path: str*, *crop_margin_ratio: float = 0.125*)

Merge all images in `images_dir` into a single image

unetseg.predict module

class unetseg.predict.**PredictConfig**(*images_path=''*, *results_path=''*, *batch_size=32*, *model_architecture='unet'*, *model_path='unet.h5'*, *height=320*, *width=320*, *n_channels=3*, *n_classes=1*, *class_weights=0*)

Bases: object

unetseg.predict.**predict**(*cfg:* unetseg.predict.PredictConfig)

Performs inference based on a configuration object

Parameters **cfg** (PredictConfig) – Configuration object

unetseg.train module

class unetseg.train.**TrainConfig**(*images_path*, *masks_path=None*, *width=200*, *height=200*, *n_channels=3*, *n_classes=1*, *apply_image_augmentation=True*, *model_path='unet.h5'*, *model_architecture='unet'*, *validation_split=0.1*, *test_split=0.1*, *epochs=15*, *steps_per_epoch=2000*, *early_stopping_patience=3*, *batch_size=32*, *seed=None*, *evaluate=True*, *class_weights=0*)

Bases: object

unetseg.train.**build_data_generator**(*image_files: List[str]*, ***, *config:* unetseg.train.TrainConfig, *mask_dir: str*)

Build data generator based on a list of images and directory of binary masks.

Parameters

- **image_files** (*List[str]*) – List of paths to images.
- **config** (TrainConfig) – Configuration object.
- **mask_dir** (*str*) – Path to directory with binary masks.

Yields *tuple* – Tuple of image and mask batch.

unetseg.train.**build_model_unet**(*cfg:* unetseg.train.TrainConfig) → keras.engine.training.Model

Build U-Net model class.

Parameters **cfg** (TrainConfig) – Configuration for training.

Returns U-Net model class.

Return type Model

unetseg.train.**build_model_unetplusplus**(*cfg:* unetseg.train.TrainConfig) → keras.engine.training.Model

Builds a U-Net++ model.

Parameters **cfg** (TrainConfig) – Training configuration.

Returns The U-Net++ model.

Return type Model

unetseg.train.**get_mask_raster**(*image_path: str*, *n_channels: Optional[int] = None*, ***, *mask_dir: str*) → numpy.ndarray

Get respective mask raster from image path.

Parameters

- **image_path** (*str*) – Path to image.
- **n_channels** (*int, optional*) – Number of channels in image. The default is None.
- **mask_dir** (*str, optional*) – Path to mask directory. The default is None.

Returns Mask image.

Return type np.ndarray

unetseg.train.**get_raster**(*image_path: str*, *n_channels: Optional[int] = None*) → numpy.ndarray

Loads a raster image from a file.

Parameters

- **image_path** (*str*) – Path to the image file.
- **n_channels** (*int, optional*) – Number of channels in the image. If not specified, the number of channels is inferred from the image file.

Returns The loaded image.

Return type np.ndarray

unetseg.train.**preprocess_input**(*image: numpy.ndarray*, *mask: numpy.ndarray*, ***, *config:* unetseg.train.TrainConfig) → Tuple[numpy.ndarray, numpy.ndarray]

Preprocess input image and masks.

Parameters

- **image** (*np.ndarray*) – Input image.
- **mask** (*np.ndarray*) – Input mask.
- **config** (TrainConfig) – Training configuration.

Returns Preprocessed image and mask.

Return type Tuple[np.ndarray, np.ndarray]

unetseg.train.**train**(*cfg:* unetseg.train.TrainConfig)

Performs training and evaluation of the model based on a configuration object.

Parameters **cfg** (TrainConfig) – Configuration object containing all the necessary parameters for training.

unetseg.utils module

unetseg.utils.**grouper**(*iterable*, *n*, *fillvalue=None*)

Collect data into fixed-length chunks or blocks

unetseg.utils.**load_model**(*model_path: str*) → keras.engine.training.Model

Load model from `model_path`

unetseg.utils.**resize**(*image*, *size*)

Resize multiband image to an image of size (h, w)

Module contents

1.4 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.4.1 [0.2.1] - 2022-02-02

Changed

- fix(evaluate): Change `masks` to `extent`

1.4.2 [0.2.0] - 2022-01-13

Added

- New `masks_path` training config attribute for specifying a custom directory for masks

Changed

- Upgrade to Tensorflow 2+
- Support for satproc >= 0.1.8
- Depend on Python >= 3.7

1.4.3 [0.1.10] - 2022-01-11

Changed

- Force requirement to Python < 3.8 (caused by dependency to TF 1.15)
- Add missing dependencies
- Update docstrings

1.4.4 [0.1.5] - 2021-08-18

Added

- New model Unet++ and setting `model_architecture` on training config object

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX