pyimagetest

Release 0.3.0

Philip Meier

Aug 13, 2020

CONTENTS

If you have ever worked with multiple image backends at the same time you know that it can cumbersome to check images from different backends for equality. `pyimagetest` is a Python library that provides utilities for unit testing with images. It provides `ImageTestCase` that enables convenient image loading and comparison.

As of now the following image backends are builtin:

- imageio
- Pillow
- torchvision

`pyimagetest` requires Python 3.6 or later and is based on numpy. The code lives on GitHub and is licensed under the 3-Clause BSD License.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

`pyimagetest` is a proper Python package and listed on PyPI. To install the latest stable version run

```
pip install pyimagetest
```

To install the latest unreleased version from source run

```
git clone https://github.com/pmeier/pyimagetest
cd pyimagetest
pip install .
```

1.1.1 Installation with builtin backends

Although `pyimagetest` has support for some *image backends built in*, by default none are installed. To install the requirements for all builtin backends, run the pip command with the `[builtin_backends]` extra.

```
pip install pyimagetest[backends]
```

1.2 Usage examples

The following examples showcase the functionality of `pyimagetest`. This requires some backends to be installed. You can either install them for each example individually or simply *install all builtin backends*.

1.2.1 General usage

- Requirements: `pip install imageio Pillow`

`ImageTestCase` provides two convenience methods to ease unit testing with images:

1. `load_image()` loads and image from a `file` with a given `backend`.
2. `assertImagesAlmostEqual()` compares two images of possible different backends on equality.

A simple I/O test that compares `imageio` and `Pillow` could look like this:

```
import pyimagetest
from os import path

class ImageTester(pyimagetest.ImageTestCase):
    def test_io(self):
        file = path.join("path", "to", "test", "image")

        imageio_image = self.load_image(file, backend="imageio")
        pil_image = self.load_image(file, backend="PIL")

        self.assertImagesAlmostEqual(imageio_image, pil_image)
```

1.2.2 Working with a single backend and / or file

- Requirements: `pip install imageio`

If you mainly work with a single image backend and / or a file, you can ease up your workflow by overwriting `default_image_backend()` and / or `default_image_file()`. The return values are then used in `load_image()` if no `backend` and / or `file` is given:

```
import pyimagetest
from os import path

class ImageTester(pyimagetest.ImageTestCase):
    def default_image_backend(self):
        return "imageio"

    def default_image_file(self):
        return path.join("path", "to", "test", "image")

    def test_io(self):
        file = path.join("path", "to", "test", "image")
        backend = "imageio"

        specific_image = self.load_image(file, backend)
        default_image = self.load_image()

        self.assertImagesAlmostEqual(specific_image, default_image)
```

1.2.3 Creating a custom backend

- Requirements: `pip install imageio`

If you want to work with an backend not included in `pyimagetest` you can create your own by subclassing `ImageBackend`:

```
from pyimagetest.backends import ImageBackend
import imageio


class MyImage:
    @staticmethod
    def from_numpy(data):
        ...
```

(continues on next page)

(continued from previous page)

```
    def to_numpy(self):
        ...


class MyBackend(ImageBackend):
    def native_image_type(self):
        return MyImage

    def import_image(self, file):
        return MyImage.from_numpy(imageio.imread(file))

    def export_image(self, image):
        return image.to_numpy()
```

To able to access `MyBackend` at runtime you can add it within the constructor of the test case:

```
import pyimagetest
from os import path


class ImageTester(pyimagetest.ImageTestCase):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.add_image_backend("MyBackend", MyBackend())

    def test_my_backend(self):
        file = path.join("path", "to", "test", "image")

        my_image = self.load_image(file, backend="MyBackend")
```

Note: If you add a custom backend with the same `native_image_type()` as a builtin backend, you can remove the builtin one with `remove_image_backend()`. Otherwise the automatic backend inference of `assertImagesAlmostEqual()` might not work as intended.

Note: If you create a custom backend based on an open-source Python package, consider contributing it to `pyimagetest`.

CHAPTER

TWO

CONTRIBUTING GUIDE LINES

We appreciate all contributions. If you are planning to contribute bug-fixes or documentation improvements, please open a pull request (PR) without further discussion. If you planning to contribute new features, please open an issue and discuss the feature with us first.

To start working on `pyimagetest-new` clone from the latest version and install the development requirements:

```
PYIMAGETEST-NEW_ROOT = pyimagetest-new
git clone https://github.com/pmeier/pyimagetest-new $PYIMAGETEST-NEW_ROOT
cd $PYIMAGETEST-NEW_ROOT
pip install -r requirements-dev.txt
pre-commit install
```

Every PR is subjected to multiple checks that it has to pass before it can be merged. The checks are performed by tox . Below you can find details and instructions how to run the checks locally.

2.1 Code format and linting

`pyimagetest-new` uses isort to sort the imports, black to format the code, and flake8 to enforce PEP8 compliance.

Furthermore, `pyimagetest-new` is PEP561 compliant and checks the type annotations with mypy .

To format your code run

```
cd $PYIMAGETEST-NEW_ROOT
tox -e format
```

Note: Amongst others, `isort` and `black` are run by pre-commit before every commit.

To run the full lint check locally run

```
cd $PYIMAGETEST-NEW_ROOT
tox -e lint
```

2.2 Tests

`pyimagetest-new` uses pytest to run the test suite. You can run it locally with

```
cd $PYIMAGETEST-NEW_ROOT
tox
```

Note: `pyimagetest-new` adds the following custom options with the corresponding `@pytest.mark.*` decorators: - `--skip-large-download`: `@pytest.mark.large_download` - `--skip-slow`: `@pytest.mark.slow` - `--run-flaky`: `@pytest.mark.flaky`

Options prefixed with `--skip` are run by default and skipped if the option is given. Options prefixed with `--run` are skipped by default and run if the option is given.

These options are passed through `tox` if given after a `--` flag. For example, the CI invocation command is equivalent to:

```
cd $PYIMAGETEST-NEW_ROOT
tox -- --skip-large-download
```

2.3 Documentation

To build the html and latex documentation locally, run

```
cd $PYIMAGETEST-NEW_ROOT
tox -e docs
```

CHAPTER

THREE

PACKAGE REFERENCE

3.1 pyimagetest

class pyimagetest.**ImageBackend**
ABC for image backends.

Each subclass has to implement the *native_image_type* as well as the basic I/O methods *import_image()* and *export_image()*.

abstract export_image(*image: Any*) → numpy.ndarray
Exports an image to `numpy.ndarray`.

The output is of `shape == (height, width, channels)` and of `dtype == numpy.float32`.

Parameters **image** – Image to be exported.

abstract import_image(*file: str*) → Any
Imports an image from `file`.

Parameters **file** – Path to the file that should be imported.

abstract property native_image_type
Native image type of the backend.

This is used to infer the `ImageBackend` from a given image.

pyimagetest.**add_image_backend**(*name: str*, *backend: pyimagetest.backends.ImageBackend*, *allow_duplicate_type: bool = False*) → None
Adds custom backend to the available backends.

Parameters

- **name** – Name of the backend
- **backend** – Backend
- **allow_duplicate_type** – If `True`, no check for duplicate *native_image_type* s is performed. Defaults to `False`.

Raises **RuntimeError** – If another *ImageBackend* with the same *native_image_type* already present and `allow_duplicate_type` is `False`.

Note: If you add an *ImageBackend* with a duplicate *native_image_type*, the automatic backend inference with *infer_image_backend()* might not work correctly.

---

pyimagetest.**remove_image_backend**(*name: str*) → None
Removes a backend from the known backends.

Parameters **name** – Name of the backend to be removed

pyimagetest.**infer_image_backend**(*image: Any*) → pyimagetest.backends.ImageBackend
Infers the corresponding backend from the `image`.

Parameters **image** – Image with type of any known backend

Raises **RuntimeError** – If type of image does not correspond to any known image backend

pyimagetest.**assert_images_almost_equal**(*image1: Any*, *image2: Any*, *mae: float = 0.01*, *backend1: Optional[Union[pyimagetest.backends.ImageBackend, str]] = None*, *backend2: Optional[Union[pyimagetest.backends.ImageBackend, str]] = None*) → None
Image equality assertion.

Parameters

- **image1** – Image 1
- **image2** – Image 2
- **mae** – Maximum acceptable mean absolute error (MAE). Defaults to `1e-2`.
- **backend1** – *ImageBackend* or its name for `image1`. If omitted, the backend is inferred from `image1` with *infer_image_backend()*.
- **backend2** – *ImageBackend* or its name for `image2`. If omitted, the backend is inferred from `imag2` with *infer_image_backend()*.

Raises **AssertionError** – If *image1* and *image2* are not equal up to the acceptable MAE.

PYTHON MODULE INDEX

p

INDEX