qlty Documentation

Release 0.2.0

Petrus H. Zwart

Nov 22, 2022

CONTENTS:

CHAPTER

ONE

QLTY

Unstitch and stich back pytorch tensors

- Free software: BSD license
- Documentation: https://qlty.readthedocs.io.

1.1 Features

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install qlty, run this command in your terminal:

```
$ pip install qlty
```

This is the preferred method to install qlty, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for qlty can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/phzwart/qlty
```

Or download the tarball:

```
$ curl -OJL https://github.com/phzwart/qlty/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

qlty provides tools unstitch and stitch pyutorch tensors.

To use qlty in a project import it:

```
import qlty
from qlty import qlty2D
```

Lets make some mock data:

```
import einops
import torch
import numpy as np

x = torch.rand((10,3,128,128))
y = torch.rand((10,1,128,128))
```

Assume that x and y are data wwhose relation you are trying to learn using some network, such that after training, you have something like:

```
y_guess = net(x)
```

with:

```
torch.sum( torch.abs(y_guess - y) ) < a_small_number
```

If the data you have is large and doesn't fit onto your GPU card, or if you need to chop things up into smaller bits for boundary detection qlty can be use. Lets take the above data and chop it into smaller bits:

```
quilt = qlty2D.NCYXQuilt(X=128,
                         Y=128,
                         window=(16,16),
                         step=(4,4),
                         border=(4,4),
                         border_weight=0)
```

This object now allows one to cut any input tensor with shape (N,C,Y,X) into smaller, overlapping patches of size (M,C,Ywindow,Xwindow). The moving window, in this case a 16x16 patch, is moved along the input tensor with steps (4,4). In addition, we define a border region in these patches of 4 pixels wide. Pixels in this area will we assigned weight border_weight (0 in this case) when data is stitched back together (more later).

Lets unstitch the (x,y) training data pair:

```
x_bits, y_bits = quilt.unstitch_data_pair(x,y)
print("x shape: ",x.shape)
print("y shape: ",y.shape)
print("x_bits shape:", x_bits.shape)
print("y_bits shape:", y_bits.shape)
```

Yielding:

```
x shape:  torch.Size([10, 3, 128, 128])
y shape:  torch.Size([10, 128, 128])
x_bits shape: torch.Size([8410, 3, 16, 16])
y_bits shape: torch.Size([8410, 16, 16])
```

If we now make some mock data that a neural network has returned:

    y_mock = torch.rand((8410,17,16,16))

we can sticth it back together into the right shape, averaging overlapping areas, excluding or downweighting border areas:

```
y_stiched, weights = quilt.stitch(y_mock)
```

which gives:

```
print(y_stiched.shape)
torch.Size([10, 17, 128, 128])
```

The 'weights' tensor encodes how many contributors there were for each pixel.

CHAPTER

FOUR

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/phzwart/qlty/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

qlty could always use more documentation, whether as part of the official qlty docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/phzwart/qlty/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *qlty* for local development.

1. Fork the *qlty* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/qlty.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv qlty
$ cd qlty/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 qlty tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/phzwart/qlty/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_qlty
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- Petrus H. Zwart <PHZwart@lbl.gov>

5.2 Contributors

None yet. Why not be the first?

CHAPTER

SIX

HISTORY

6.1 0.1.0 (2021-10-20)

- First release on PyPI.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search