torchwebio Documentation

Release 0.0.1.post1.dev2+g86f554d

Prass, The Nomadic Coder

Sep 28, 2022

CONTENTS

CHAPTER

ONE

TORCHWEBIO

Yet another model to web app generator

torchwebio is a simple package that turns your pytorch model into a web application.

1.1 Why?

Researchers often train models or fine tune models using pytorch and use dashboards like tensorboard to track the progress of the training routine.

These dashboards often give you metrics like loss and accuracies and visualizing some sample images. But they don't help you visualize the actual real-world performance of your model in-training.

Imagine you want to test your model against some other sample images that you didn't plan before starting the training routine. Imagine you want to share your "latest-greatest-and-bestest" model with your manager. Imagine you wanted to continuously train your model and always share the latest-and-greatest with the world.

That's where this package comes in.

Read the relevant blog post here

Find the documentation here

1.2 How?

The code that powers this package is actually **very, very** simple. It is built upon the shoulders of giants: pywebio for app generation, and timm for Image classification models.

I am to be very opinionated, and support interfaces set out by the popular neural network libraries. This allows the `webalyzer` interface to be extremely simple, without needing to support all the "chaos" that exists out there.

Problem type	Library	Status
Image Classification	timm	Implemented
Object detection, Instance segmentation	detectron2	Coming Soon
NLP models	transformers	Coming soon

Credits to pywebio for a super simple framework for python web UI generation.

1.3 Install

```
pip install torchwebio
```

1.4 Usage

1.4.1 Simple visualization

```
import timm
from torchwebio.webalyzer import webalyzer

# Load a TIMM-like model or a regular pytorch model
model = timm.create_model('tf_efficientnet_b0_apss', pretrained=True)

# ....
# Fine tune the model
# ....


# Launch the web UI
webalyzer(model)
```

1.4.2 Auto-updating application (coming soon!)

```
import timm
from torchwebio.webalyzer import webalyzer, updater

# Load a TIMM-like model or a regular pytorch model
model = timm.create_model('tf_efficientnet_b0_apss', pretrained=True)

webalyzer(model)

for idx, (data, labels) in enumerate(dataloader):
     # do some finetuning
     outputs = model(data)
     loss = criterion(outputs, labels)
     loss.backward()
     # ....

     # update the app every 1000 iterations
     if idx % 1000 == 0:
          updater(model)
```

For more examples, look into the demo folder

1.5 Contribute

Check out CONTRIBUTING.md. More to come!

1.6 Help out

I don't do this a day job, but if you do want to help out, buy me a coffee. I promise to donate 100% of the proceeds on the account :)

More than the money, this give me a data point on people "actually" interested in this project

1.7 Contents

1.7.1 torchwebio

Yet another model to web app generator

torchwebio is a simple package that turns your pytorch model into a web application.

Why?

Researchers often train models or fine tune models using pytorch and use dashboards like tensorboard to track the progress of the training routine.

These dashboards often give you metrics like loss and accuracies and visualizing some sample images. But they don't help you visualize the actual real-world performance of your model in-training.

Imagine you want to test your model against some other sample images that you didn't plan before starting the training routine. Imagine you want to share your "latest-greatest-and-bestest" model with your manager. Imagine you wanted to continuously train your model and always share the latest-and-greatest with the world.

That's where this package comes in.

Read the relevant blog post here

Find the documentation here

How?

The code that powers this package is actually **very, very** simple. It is built upon the shoulders of giants: pywebio for app generation, and timm for Image classification models.

I am to be very opinionated, and support interfaces set out by the popular neural network libraries. This allows the `webalyzer` interface to be extremely simple, without needing to support all the "chaos" that exists out there.

Problem type	Library	Status
Image Classification	timm	Implemented
Object detection, Instance segmentation	detectron2	Coming Soon
NLP models	transformers	Coming soon

Credits to pywebio for a super simple framework for python web UI generation.

Install

```
pip install torchwebio
```

Usage

Simple visualization

```
import timm
from torchwebio.webalyzer import webalyzer

# Load a TIMM-like model or a regular pytorch model
model = timm.create_model('tf_efficientnet_b0_apss', pretrained=True)

# ....
# Fine tune the model
# ....


# Launch the web UI
webalyzer(model)
```

Auto-updating application (coming soon!)

```
import timm
from torchwebio.webalyzer import webalyzer, updater

# Load a TIMM-like model or a regular pytorch model
model = timm.create_model('tf_efficientnet_b0_apss', pretrained=True)

webalyzer(model)

for idx, (data, labels) in enumerate(dataloader):
```

(continues on next page)

(continued from previous page)

```
    # do some finetuning
    outputs = model(data)
    loss = criterion(outputs, labels)
    loss.backward()
    # ....

    # update the app every 1000 iterations
    if idx % 1000 == 0:
         updater(model)
```

For more examples, look into the demo folder

Contribute

Check out CONTRIBUTING.md. More to come!

Help out

I don't do this a day job, but if you do want to help out, buy me a coffee. I promise to donate 100% of the proceeds on the account :)

More than the money, this give me a data point on people "actually" interested in this project

1.7.2 Contributing

Welcome to `torchwebio` contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but other kinds of contributions are also appreciated.

If you are new to using git or have never collaborated in a project previously, please have a look at contribution-guide.org. Other resources are also listed in the excellent guide created by FreeCodeCamp[1].

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, Python Software Foundation's Code of Conduct is a good reference in terms of behavior guidelines.

Issue Reports

If you experience bugs or general issues with `torchwebio`, please have a look on the issue tracker. If you don't see anything useful there, please feel free to fire an issue report.

Tip: Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

---

[1] Even though, these resources focus on open source projects and communities, the general ideas behind collaborating with other developers to collectively create software are general and can be applied to all sorts of environments, including private companies and proprietary code bases.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

Documentation Improvements

You can help improve `torchwebio` docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

`torchwebio` documentation uses Sphinx as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way was a code contribution.

When working on documentation changes in your local machine, you can compile them using tox :

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (`http://localhost:8000`):

```
python3 -m http.server --directory 'docs/_build/html'
```

Code Contributions

Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the issue tracker to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated virtual environment to avoid any problems with your installed Python packages. This can easily be done via either virtualenv:

```
virtualenv <PATH TO VENV>
source <PATH TO VENV>/bin/activate
```

or Miniconda:

```
conda create -n torchwebio python=3 six virtualenv pytest pytest-cov
conda activate torchwebio
```

Clone the repository

1. Create an user account on GitHub if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

   ```
   git clone git@github.com:YourLogin/torchwebio.git
   cd torchwebio
   ```

4. You should run:

```
pip install -U pip setuptools -e .
```

   to be able to import the package under development in the Python REPL.

5. Install pre-commit:

```
pip install pre-commit
pre-commit install
```

   `torchwebio` comes with a lot of hooks configured to automatically help the developer to check the code being written.

Implement your changes

1. Create a branch to hold your changes:

```
git checkout -b my-feature
```

   and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add docstrings to new functions, modules and classes, especially if they are part of public APIs.

3. Add yourself to the list of contributors in `AUTHORS.rst`.

4. When you're done editing, do:

```
git add <MODIFIED FILES>
git commit
```

   to record your changes in git.

   Please make sure to see the validation messages from pre-commit and fix any eventual issues. This should automatically use flake8/black to check/fix the code style in a way that is compatible with the project.

   Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

   Moreover, writing a descriptive commit message is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

   to look for recurring communication patterns.

   ---

5. Please check that your changes don't break any unit tests with:

```
tox
```

   (after having installed tox with `pip install tox` or `pipx`).

   You can also use tox to run several other pre-configured tasks in the repository. Try `tox -av` to see a list of the available checks.

Submit your contribution

1. If everything works fine, push your local branch to the remote server with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and click "Create pull request" to send your changes for review.

Troubleshooting

The following tips can be used when facing problems to build or test the package:

1. Make sure to fetch all the tags from the upstream repository. The command `git describe --abbrev=0 --tags` should return the version you are expecting. If you are trying to run CI scripts in a fork repository, make sure to push all the tags. You can also try to remove all the egg files or the complete egg folder, i.e., `.eggs`, as well as the `*.egg-info` folders in the `src` folder or potentially in the root of your project.

2. Sometimes tox misses out when new dependencies are added, especially to `setup.cfg` and `docs/requirements.txt`. If you find any problems with missing dependencies when running a command with tox, try to recreate the `tox` environment using the `-r` flag. For example, instead of:

```
tox -e docs
```

   Try running:

```
tox -r -e docs
```

3. Make sure to have a reliable tox installation that uses the correct Python version (e.g., 3.7+). When in doubt you can run:

```
tox --version
# OR
which tox
```

   If you have trouble and are seeing weird errors upon running tox, you can also try to create a dedicated virtual environment with a tox binary freshly installed. For example:

```
virtualenv .venv
source .venv/bin/activate
.venv/bin/pip install tox
.venv/bin/tox -e all
```

4. Pytest can drop you in an interactive session in the case an error occurs. In order to do that you need to pass a `--pdb` option (for example by running `tox -- -k <NAME OF THE FALLING TEST> --pdb`). You can also setup breakpoints manually instead of using the `--pdb` option.

Maintainer tasks

Releases

If you are part of the group of maintainers and have correct user permissions on PyPI, the following steps can be used to release a new version for `torchwebio`:

1. Make sure all unit tests are successful.
2. Tag the current commit on the main branch with a release tag, e.g., `v1.2.3`.
3. Push the new tag to the upstream repository, e.g., `git push upstream v1.2.3`
4. Clean up the `dist` and `build` folders with `tox -e clean` (or `rm -rf dist build`) to avoid confusion with old builds and Sphinx docs.
5. Run `tox -e build` and check that the files in `dist` have the correct version (no `.dirty` or git hash) according to the git tag. Also check the sizes of the distributions, if they are too big (e.g., > 500KB), unwanted clutter may have been accidentally included.
6. Run `tox -e publish -- --repository pypi` and check that everything was uploaded to PyPI correctly.

1.7.3 License

1.7.4 Changelog

Version 0.1 (development)

- Feature A added
- FIX: nasty bug #1729 fixed
- add your changes here!

1.7.5 torchwebio

torchwebio package

Subpackages

torchwebio.models package

Subpackages

torchwebio.models.image package

Submodules

torchwebio.models.image.imageclassificationmodel module

class torchwebio.models.image.imageclassificationmodel.**ImageClassificationModel**(*model_name: Optional[str] = ''*, *number_of_results: Optional[int] = 5*, *model: Optional[Module] = None*, *category_list: Optional[str] = None*)

Bases: object

An adapter class to expose all Image classification functionality for models that follow the interface of pytorch-image-models (timm).

In a nutshell, this class exposes the following functionality : pre-process, forward and post-process. The process method does all three at a go, and is pretty much the only method that needs to be called externally.

Additionally, there are some helper static methods that are grouped here only for encapsulation purposes

forward(*img_tensor: Tensor*) → Tensor

Forward pass of the pytorch model

Parameters
img_tensor (*Tensor*) – input image after pre-processing.

Returns
Output probabilities from the network

Return type
Tensor

post_process(*probabilities: Tensor*) → List[Tuple[str, float]]

Post process the soft-max output of the model. Extracts top-K results from the last layer,

and maps them to the corresponding categories.

Parameters
probabilities (*Tensor*) – Tensor of probabilities (output of the network)

Returns
List of pairs of Category label and category score

Return type
List[Tuple[str, float]]

pre_process(*img: <module 'PIL.Image' from '/home/docs/checkouts/readthedocs.org/user_builds/torchwebio/envs/latest/lib/python3.8/site-packages/PIL/Image.py'>*) →
Tensor

Preprocess the image. Basically processes the image through the transformations. Assumes batch size = 1

Parameters
img (*Image*) – Image to pre-process.

Returns
pre-processed image as a tensor

Return type
Tensor

process_img(*img*) → List[Tuple[str, float]]

Processes a single image. Calls pre_process, forward and post_process in succession

Parameters
img (*PIL.Image*) – Input image as a PIL object

Returns
Output scores and categories

Return type
List[Tuple[str, float]]

Module contents

Module contents

Submodules

torchwebio.exceptions module

exception torchwebio.exceptions.**BaseTorchWebioException**

Bases: Exception

Base exception of the library

exception torchwebio.exceptions.**ComingSoonException**(*feature*, *error*)

Bases: *BaseTorchWebioException*

A special exception for features that are not yet implemented, but are planned

torchwebio.webalyzer module

class torchwebio.webalyzer.**Model_Types**(*value*)

Bases: IntEnum

An enumeration.

IMAGE_MODEL = 1

NLP_MODEL = 2

torchwebio.webalyzer.**image_class_webio**(*model: Module*, *title*, *subtitle*, *category_list=None*)

Implementation of a PyWebIO application for image classification models.

Accepts an image and outputs a table of classification scores.

Parameters

- **model** (*nn.Module*) – Pytorch model that needs to do the prediction
- **title** (*_type_*) – Title for the application
- **subtitle** (*_type_*) – Subtitle for the application
- **category_list** (*_type_, optional*) – URL to the category list to map categories indices to labels, by default None

torchwebio.webalyzer.**webalyzer**(*model: Module*, *title: str = 'Image Classification'*, *subtitle: str = 'Calculates classsification labels on ImageNet'*, *model_type:* Model_Types *= Model_Types.IMAGE_MODEL*, *class_category_list: Optional[str] = None*)

Function to generate an application based on the model. Currently supports timm-like Image classification models Coming soon: NLP and more Computer vision model suppport

Parameters

- **model** (*nn.Module*) – Pytorch model to be turned into an application
- **title** (*str, optional*) – Title for the application, by default “Image Classification”
- **subtitle** (*str, optional*) –

Subtitle or body of the application, by default
"Calculates classsification labels on ImageNet"

- **model_type** (*Model_Types, optional*) – Type of the model, by default Model_Types.IMAGE_MODEL
- **class_category_list** (*str, optional*) – Mapping output vectors to class categories., by default None

Raises

ComingSoonException – If a non ImageModel Model_Types is passed, an exception is raised

Module contents

1.8 Indices and tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX