# TensorLayer Contributor Guideline

## Welcome to contribute!

You are more than welcome to contribute to TensorLayer! If you have any improvement, please send us your [pull requests](https://help.github.com/en/articles/about-pull-requests). You may implement your improvement on your [fork](https://help.github.com/en/articles/working-with-forks).

## Checklist

\* Continuous integration

\* Build from sources

\* Unittest

\* Documentation

\* General intro to TensorLayer2

\* How to contribute a new `Layer`

\* How to contribute a new `Model`

\* How to contribute a new example/tutorial

## Continuous integration

We appreciate contributions

either by adding / improving examples or extending / fixing the core library.

To make your contributions, you would need to follow the [pep8](https://www.python.org/dev/peps/pep-0008/) coding style and [numpydoc](https://numpydoc.readthedocs.io/en/latest/) document style.

We rely on Continuous Integration (CI) for checking push commits.

The following tools are used to ensure that your commits can pass through the CI test:

\* [yapf](https://github.com/google/yapf) (format code), compulsory

\* [isort](https://github.com/timothycrosley/isort) (sort imports), optional

\* [autoflake](https://github.com/myint/autoflake) (remove unused imports), optional

You can simply run

```bash

make format

```

to apply those tools before submitting your PR.

## Build from sources

```bash

# First clone the repository and change the current directory to the newly cloned repository

git clone https://github.com/zsdonghao/tensorlayer2.git

cd tensorlayer2

# Install virtualenv if necessary

pip install virtualenv

# Then create a virtualenv called `venv`

virtualenv venv

# Activate the virtualenv

## Linux:

source venv/bin/activate

## Windows:

venv\Scripts\activate.bat

# ============= IF TENSORFLOW IS NOT ALREADY INSTALLED ============= #

# basic installation

pip install .

# advanced: for a machine \*\*without\*\* an NVIDIA GPU

pip install -e ".[all\_cpu\_dev]"

# advanced: for a machine \*\*with\*\* an NVIDIA GPU

pip install -e ".[all\_gpu\_dev]"

```

## Unittest

Launching the unittest for the whole repo:

```bash

# install pytest

pip install pytest

# run pytest

pytest

```

Running your unittest code on your implemented module only:

```bash

# install coverage

pip install coverage

cd /path/to/your/unittest/code

# For example: cd tests/layers/

# run unittest

coverage run --source myproject.module -m unittest discover

# For example: coverage run --source tensorlayer.layers -m unittest discover

# generate html report

coverage html

```

## Documentation

Even though you follow [numpydoc](https://numpydoc.readthedocs.io/en/latest/) document style when writing your code,

this does not ensure those lines appear on TensorLayer online documentation.

You need further modify corresponding RST files in `docs/modules`.

For example, to add your implemented new pooling layer into documentation, modify `docs/modules/layer.rst`. First, insert layer name under Layer list

```rst

Layer list

----------

.. autosummary::

NewPoolingLayer

```

Second, find pooling layer part and add:

```rst

.. -----------------------------------------------------------

.. Pooling Layers

.. -----------------------------------------------------------

Pooling Layers

------------------------

New Pooling Layer

^^^^^^^^^^^^^^^^^^^^^^^^^^

.. autoclass:: NewPoolingLayer

```

Finally, test with local documentation:

```bash

cd ./docs

make clean

make html

# then view generated local documentation by ./html/index.html

```

## General intro to TensorLayer2

\* TensorLayer2 is built on [TensorFlow2](https://www.tensorflow.org/alpha), so TensorLayer2 is purely eager, no sessions, no globals.

\* TensorLayer2 supports APIs to build static models and dynamic models. Therefore, all `Layers` should be compatible with the two modes.

```python

# An example of a static model

# A static model has inputs and outputs with fixed shape.

inputs = tl.layers.Input([32, 784])

dense1 = tl.layers.Dense(n\_units=800, act=tf.nn.relu, in\_channels=784, name='dense1')(inputs)

dense2 = tl.layers.Dense(n\_units=10, act=tf.nn.relu, in\_channels=800, name='dense2')(dense1)

model = tl.models.Model(inputs=inputs, outputs=dense2)

# An example of a dynamic model

# A dynamic model has more flexibility. The inputs and outputs may be different in different runs.

class CustomizeModel(tl.models.Model):

def \_\_init\_\_(self):

super(CustomizeModel, self).\_\_init\_\_()

self.dense1 = tl.layers.Dense(n\_units=800, act=tf.nn.relu, in\_channels=784, name='dense1')

self.dense2 = tl.layers.Dense(n\_units=10, act=tf.nn.relu, in\_channels=800, name='dense2')

# a dynamic model allows more flexibility by customising forwarding.

def forward(self, x, bar=None):

d1 = self.dense1(x)

if bar:

return d1

else:

d2 = self.dense2(d1)

return d1, d2

model = CustomizeModel()

```

\* More examples can be found in [examples](examples/) and [tests/layers](tests/layers/). Note that not all of them are completed.

## How to contribute a new `Layer`

\* A `NewLayer` should be a derived from the base class [`Layer`](tensorlayer/layers/core.py).

\* Member methods to be overrided:

- `\_\_init\_\_(self, args1, args2, inputs\_shape=None, name=None)`: The constructor of the `NewLayer`, which should

- Call `super(NewLayer, self).\_\_init\_\_(name)` to construct the base.

- Define member variables based on the args1, args2 (or even more).

- If the `inputs\_shape` is provided, call `self.build(inputs\_shape)` and set `self.\_built=True`. Note that sometimes only `in\_channels` should be enough to build the layer like [`Dense`](tensorlayer/layers/dense/base\_dense.py).

- Logging by `logging.info(...)`.

- `\_\_repr\_\_(self)`: Return a printable representation of the `NewLayer`.

- `build(self, inputs\_shape)`: Build the `NewLayer` by defining weights.

- `forward(self, inputs, \*\*kwargs)`: Forward feeding the `NewLayer`. Note that the forward feeding of some `Layers` may be different during training and testing like [`Dropout`](tensorlayer/layers/dropout.py).

\* Unittest:

- Unittest should be done before a pull request. Unittest code can be written in [tests/](tests/)

\* Documents:

- Please write a description for each class and method in RST format. The description may include the functionality, arguments, references, examples of the `NewLayer`.

\* Examples: [`Dense`](tensorlayer/layers/dense/base\_dense.py), [`Dropout`](tensorlayer/layers/dropout.py), [`Conv`](tensorlayer/layers/convolution/simplified\_conv.py).

## How to contribute a new `Model`

\* A `NewModel` should be derived from the base class [`Model`](tensorlayer/models/core.py) (if dynamic) or an instance of [`Model`](tensorlayer/models/core.py) (if static).

\* A static `NewModel` should have fixed inputs and outputs. Please check the example [`VGG\_Static`](tensorlayer/models/vgg.py)

\* A dynamic `NewModel` has more flexiblility. Please check the example [`VGG16`](tensorlayer/models/vgg16.py)

## How to contribute a new example/tutorial

\* A new example/tutorial should implement a complete workflow of deep learning which includes (but not limited)

- `Models` construction based on `Layers`.

- Data processing and loading.

- Training and testing.

- Forward feeding by calling the models.

- Loss function.

- Back propagation by `tf.GradientTape()`.

- Model saving and restoring.

\* Examples: [MNIST](examples/basic\_tutorials/tutorial\_mnist\_mlp\_static.py), [CIFAR10](examples/basic\_tutorials/tutorial\_cifar10\_cnn\_static.py), [FastText](examples/text\_classification/tutorial\_imdb\_fasttext.py)