# TensorVision Documentation

***Release 0.1.dev1***

**Marvin Teichmann and Martin Thoma**

February 20, 2017

Contents

TensorVision is a library to build, train and evaluate neural networks in TensorFlow.

TensorVision is a work in progress, input is welcome. The available documentation is limited for now. The project is on GitHub.

CHAPTER 1

# User Guide

The TensorVision user guide explains how to install TensorVision, how to build and train neural networks using TensorVision, and how to contribute to the library as a developer.

## Installation

TensorVision has a couple of prerequisites that need to be installed first, but it is not very picky about versions.

Most of the instructions below assume you are running a Linux or Mac system, but are otherwise very generic.

If you run into any trouble, please check the TensorFlow installation instructions which cover installing the prerequisites for a range of operating systems, or ask for help as a GitHub issue (https://github.com/TensorVision/TensorVision/issues).

### Prerequisites

#### Python + pip

TensorVision currently requires Python 2.7 or 3.4 to run. Please install Python via the package manager of your operating system if it is not included already.

Python includes `pip` for installing additional modules that are not shipped with your operating system, or shipped in an old version, and we will make use of it below. We recommend installing these modules into your home directory via `--user`, or into a virtual environment via `virtualenv`.

#### C compiler

Numpy/scipy require a compiler if you install them via `pip`. On Linux, the default compiler is usually `gcc`, and on Mac OS, it's `clang`. Again, please install them via the package manager of your operating system.

#### numpy/scipy + BLAS

TensorVision requires numpy. Numpy/scipy rely on a BLAS library to provide fast linear algebra routines. They will work fine without one, but a lot slower, so it is worth getting this right (but this is less important if you plan to use a GPU).

If you install numpy and scipy via your operating system's package manager, they should link to the BLAS library installed in your system. If you install numpy and scipy via `pip install numpy` and `pip install scipy`,

make sure to have development headers for your BLAS library installed (e.g., the `libopenblas-dev` package on Debian/Ubuntu) while running the installation command. Please refer to the numpy/scipy build instructions if in doubt.

## Stable TensorVision release

Currently, no stable version is available.

## Bleeding-edge version

To install the latest version of TensorVision, run the following commands:

```
pip install --upgrade https://github.com/TensorVision/TensorVision/archive/master.zip
```

Again, add `--user` if you want to install to your home directory instead.

## Development installation

Alternatively, you can install TensorVision from source, in a way that any changes to your local copy of the source tree take effect without requiring a reinstall. This is often referred to as *editable* or *development* mode. Firstly, you will need to obtain a copy of the source tree:

```
git clone https://github.com/TensorVision/TensorVision.git
```

It will be cloned to a subdirectory called `TensorVision`. Make sure to place it in some permanent location, as for an *editable* installation, Python will import the module directly from this directory and not copy over the files. Enter the directory and install the requirements:

```
cd TensorVision
pip install -r requirements.txt
```

You should also install the additional development requirements which can be found in `requirements-dev.txt`.

To install the TensorVision package itself, in editable mode, run:

```
pip install --editable .
```

As always, add `--user` to install it to your home directory instead.

**Optional**: If you plan to contribute to TensorVision, you will need to fork the TensorVision repository on GitHub. This will create a repository under your user account. Update your local clone to refer to the official repository as `upstream`, and your personal fork as `origin`:

```
git remote rename origin upstream
git remote add origin https://github.com/<your-github-name>/TensorVision.git
```

If you set up an SSH key, use the SSH clone URL instead: `git@github.com:<your-github-name>/TensorVision.git`.

You can now use this installation to develop features and send us pull requests on GitHub, see *Development*!

You can run the tests by

```
python setup.py test
```

### GPU support

Thanks to TensorFlow, TensorVision transparently supports training your networks on a GPU, which may be 10 to 50 times faster than training them on a CPU. Currently, this requires an NVIDIA GPU with CUDA support, and some additional software for TensorFlow to use it.

#### CUDA

Install the latest CUDA Toolkit and possibly the corresponding driver available from NVIDIA: https://developer.nvidia.com/cuda-downloads

Closely follow the *Getting Started Guide* linked underneath the download table to be sure you don't mess up your system by installing conflicting drivers.

After installation, make sure `/usr/local/cuda/bin` is in your `PATH`, so `nvcc --version` works. Also make sure `/usr/local/cuda/lib64` is in your `LD_LIBRARY_PATH`, so the toolkit libraries can be found.

## Configuration

TensorVision comes with reasonable defaults. You only need to read this if you want tweak it to your needs.

TensorVision is configured with environment variables. It is quite easy to set them yourself (see multiple ways). The supported variables are:

- `TV_DIR_DATA`: The default directory where to look for data.
- `TV_DIR_RUNS`: The default directory where to look for the model.
- `TV_IS_DEV`: Either 0 or 1 - set if you want to see debug messages.
- `TV_PLUGIN_DIR`: Directory with Python scripts which will be loaded by utils.py
- `TV_SAVE`: Whether to keep all runs on default. By default runs will be written to `TV_DIR_RUNS/debug` and overwritten by newer runs, unless `tv-train --save` is called.
- `TV_USE_GPUS`: Controll which gpus to use. Default all GPUs are used, GPUs can be specified using `--gpus`. Setting `TV_USE_GPUS='force'` makes the flag `--gpus` compulsory, this is useful in cluster environoments. Use `TV_USE_GPUS='0,3'` to run Tensorflow an the GPUs with ids 0 and 3.
- `TV_STEP_SHOW`: After how many epochs of training should the `TV_STEP_STR` be printed?
- `TV_STEP_EVAL`: After how many epochs of training evaluation is done.
- `TV_STEP_WRITE`: After how many epochs of training checkpoints are written to disk.
- `TV_MAX_KEEP`: How many checkpoints to keep.
- `TV_STEP_STR`: Set what you want to see each 100th step of the training. The default is

```
Step {step}/{total_steps}: loss = {loss_value:.2f}
( {sec_per_batch:.3f} sec (per Batch);
{examples_per_sec:.1f} examples/sec;)
```

## Tutorial

This tutorial introduces the general workflow when using TensorVision. Examples can be found in the Modell Zoo repository.

## Basics

Train a model:

```
tv-train --hypes config.json
```

Evaluate a model:

```
python eval.py
```

Flags:

- `--hypes=myconfig.json`
- `--name=myname`

## Workflow

Each time you get a new task

### Create JSON file

Create a json file (e.g. *cifar10_cnn.json*). It has at least the following content:

```
{
  "model": {
    "input_file": "examples/inputs/cifar10_input.py",
    "architecture_file" : "examples/networks/cifar_net.py",
    "objective_file" : "examples/objectives/softmax_classifier.py",
    "optimizer_file" : "examples/optimizer/exp_decay.py"
  }
}
```

### Adjust input file

The `input_file` contains the path to a Python file. This Python file has to have a function `inputs(hypes, q, phase, data_dir)`.

The parameters of *inputs* are:

- `hypes`: A dictionary which contains everything your `model.json` file had.
- `q`: A queue (e.g. FIFOQueue)
- `phase`: Either `train` or `val`
- `data_dir`: Path to the data. This can be set with `TV_DIR_DATA`.

The expected return value is a tuple (xs, ys), where x is a list of features and y is a list of labels.

### Adjust architecture file

The `architecture_file` contains the architecture of the network. It has to have the function `inference(hypes, images, train=True)`, which takes image tensors creates a computation graph to produce logits

### Adjust objective file

The `objective_file` contains task spezific code od the model. It has to implement the following functions:

- `decoder(hypes, images, train=True)`
- `loss(hypes, decoder, labels)`
- `evaluation(hypes, decoder, labels)`

### Adjust the solver file

The `optimizer_file` contains the path to a Python file. This Python file has to have a function `training(H, loss, global_step, learning_rate)`. It defines how one tries to find a minimum of the loss function. Additionally it should provide a function `get_learning_rate(hype, global_step)`, which returns the current learning rate at each step.

## Scripts

TensorVision brings some scripts which you can use:

- `tv-train`: Trains, evaluates and saves the model network using a queue.
- `tv-continue`: Continues training of a model from logdir.
- `tv-analyze`: Evaluates the model.
- `tv-maskstats`: Get statistics about the distribution of classes in the masks.

## Hypes file

TensorVision makes use of a configuration file for each project. As it was originally intended to have hyperparameters of models, it is commonly called "hypes file" or `hypes.json` throughout this project.

This configuration file allows you to adjust given networks easily to new problem domains.

### data

It is recommended to create one json file for the training data sources as well as one for the testing data sources. Each file is a list of dictionaries, where each dictionary has the keys `raw` and `mask`. For example, your `trainfiles.json` could look like this:

```
[
    {
        "raw": "/home/moose/GitHub/MediSeg/DATA/OP1/img_00.png",
        "mask": "/home/moose/GitHub/MediSeg/DATA/OP1/img_00_GT.png"
    },
    {
        "raw": "/home/moose/GitHub/MediSeg/DATA/OP1/img_01.png",
        "mask": "/home/moose/GitHub/MediSeg/DATA/OP1/img_01_GT.png"
    },
    {
        "raw": "/home/moose/GitHub/MediSeg/DATA/OP1/img_02.png",
        "mask": "/home/moose/GitHub/MediSeg/DATA/OP1/img_02_GT.png"
    }
]
```

You should add the path of those files to your `hypes.json`:

```
"data": {
  "train": "../../DATA/trainfiles.json",
  "test": "../../DATA/testfiles.json"
},
```

While this is not required, it will allow you to use `tv-maskstats` and make your code more readable and easier to adjust.

#### classes

It is recommended to add a description of your labeled data to your hyperparameters file. This makes your code more readable and gives the possibility to use `tv-maskstats` as well as `tensorvision.utils.load_segmentation_mask()`. The `classes` block looks like this:

```
"classes": [
    {"name": "background",
     "colors": ["#000000"],
     "output": "#ff000000"},
    {"name": "instrument",
     "colors": ["#464646", "#a0a0a0", "#ffffff", "default"],
     "output": "#00ff007f"}
 ]
```

# Development

The TensorVision project was started by Marvin Teichmann and Martin Thoma in February 2016.

As an open-source project by researchers for researchers, we highly welcome contributions! Every bit helps and will be credited.

## What to contribute

### Give feedback

To send us general feedback, questions or ideas for improvement.

If you have a very concrete feature proposal, add it to the issue tracker on GitHub:

- Explain how it would work, and link to a scientific paper if applicable.
- Keep the scope as narrow as possible, to make it easier to implement.

### Report bugs

Report bugs at the issue tracker on GitHub. If you are reporting a bug, please include:

- your TensorVision and TensorFlow version.
- steps to reproduce the bug, ideally reduced to a few Python commands.
- the results you obtain, and the results you expected instead.

### Fix bugs

Look through the GitHub issues for bug reports. Anything tagged with "bug" is open to whoever wants to implement it. If you discover a bug in TensorVision you can fix yourself, by all means feel free to just implement a fix and not report it first.

### Implement features

Look through the GitHub issues for feature proposals. Anything tagged with "feature" or "enhancement" is open to whoever wants to implement it. If you have a feature in mind you want to implement yourself, please note that we cannot guarantee upfront that your code will be included. Please do not hesitate to just propose your idea in a GitHub issue first, so we can discuss it and/or guide you through the implementation.

### Write documentation

Whenever you find something not explained well, misleading, glossed over or just wrong, please update it! The *Edit on GitHub* link on the top right of every documentation page and the *[source]* link for every documented entity in the API reference will help you to quickly locate the origin of any text.

## How to contribute

### Edit on GitHub

As a very easy way of just fixing issues in the documentation, use the *Edit on GitHub* link on the top right of a documentation page or the *[source]* link of an entity in the API reference to open the corresponding source file in GitHub, then click the *Edit this file* link to edit the file in your browser and send us a Pull Request. All you need for this is a free GitHub account.

For any more substantial changes, please follow the steps below to setup TensorVision for development.

### Development setup

First, follow the instructions for performing a development installation of TensorVision (including forking on GitHub): *Development installation*

To be able to run the tests and build the documentation locally, install additional requirements with: `pip install -r requirements-dev.txt` (adding `--user` if you want to install to your home directory instead).

If you use the bleeding-edge version of TensorFlow, then instead of running that command, just use `pip install` to manually install all dependencies listed in `requirements-dev.txt` with their correct versions; otherwise it will attempt to downgrade TensorFlow to the known good version in `requirements.txt`.

### Documentation

The documentation is generated with Sphinx. To build it locally, run the following commands:

```
cd docs
make html
```

Afterwards, open `docs/_build/html/index.html` to view the documentation as it would appear on readthedocs. If you changed a lot and seem to get misleading error messages or warnings, run `make clean html` to force Sphinx to recreate all files from scratch.

When writing docstrings, follow existing documentation as much as possible to ensure consistency throughout the library. For additional information on the syntax and conventions used, please refer to the following documents:

- reStructuredText Primer
- Sphinx reST markup constructs
- A Guide to NumPy/SciPy Documentation

### Testing

Tensorvision wants to achieve a code coverage of 100%, which creates some duties:

- Whenever you change any code, you should test whether it breaks existing features by just running the test suite. The test suite will also be run by Travis for any Pull Request to TensorVision.
- Any code you add needs to be accompanied by tests ensuring that nobody else breaks it in future. Coveralls will check whether the code coverage stays at 100% for any Pull Request to TensorVision.
- Every bug you fix indicates a missing test case, so a proposed bug fix should come with a new test that fails without your fix.

To run the full test suite, just do

```
py.test
```

Testing will end with a code coverage report specifying which code lines are not covered by tests, if any. Furthermore, it will list any failed tests, and failed PEP8 checks.

Finally, for a loop-on-failing mode, do `pip install pytest-xdist` and run `py.test -f`. This will pause after the run, wait for any source file to change and run all previously failing tests again.

Before commiting any change, you should run

```
tv-train --hypes examples/cifar10_minimal.json
tv-analyze --hypes examples/cifar10_minimal.json --logdir examples/RUNS/debug/
```

to see if everything still works as expected.

### Sending Pull Requests

When you're satisfied with your addition, the tests pass and the documentation looks good without any markup errors, commit your changes to a new branch, push that branch to your fork and send us a Pull Request via GitHub's web interface.

All these steps are nicely explained on GitHub: https://guides.github.com/introduction/flow/

When filing your Pull Request, please include a description of what it does, to help us reviewing it. If it is fixing an open issue, say, issue #123, add *Fixes #123*, *Resolves #123* or *Closes #123* to the description text, so GitHub will close it when your request is merged.

CHAPTER 2

# API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

## tensorvision.analyze

## tensorvision.core

## tensorvision.eval

## tensorvision.utils

CHAPTER 3

# Indices and tables

- *genindex*
- *modindex*
- *search*