Bring Machine Learning to the Browser With TensorFlow.js — Part I

Applying a web friendly format to a pre-trained model resulting in a web application.

va barbosa Follow
Oct 26, 2018 · 4 min read

"assorted-color leaf hanging decor" by Chris Lawton on Unsplash

*Edited **2019 Mar 11** to include changes introduced in TensorFlow.js 1.0. Additional information about some of these TensorFlow.js 1.0 updates can be found here.*

TensorFlow.js brings machine learning and its possibilities to JavaScript. It is an open source library built to create, train, and run machine learning models in the browser (and Node.js).

Training and building complex models can take a considerable amount of resources and time. Some models require massive amounts of data to provide acceptable accuracy. And, if computationally intensive, may require hours or days of training to complete. Thus, you may not find the browser to be the ideal environment for building such models.

A more appealing use case is importing and running existing models. You train or get models trained in powerful, specialized environments then you import and run the models in the browser for impressive user experiences.

Converting the model

Before you can use a pre-trained model in TensorFlow.js, the model needs to be in a web friendly format. For this, TensorFlow.js provides the `tensorflowjs_converter` tool. The tool converts TensorFlow and Keras models to the required web friendly format. The converter is available after you install the `tensorflowjs` Python package.

```
pip install tensorflowjs
```

install-tensorflowjs hosted with ♡ by GitHub view raw

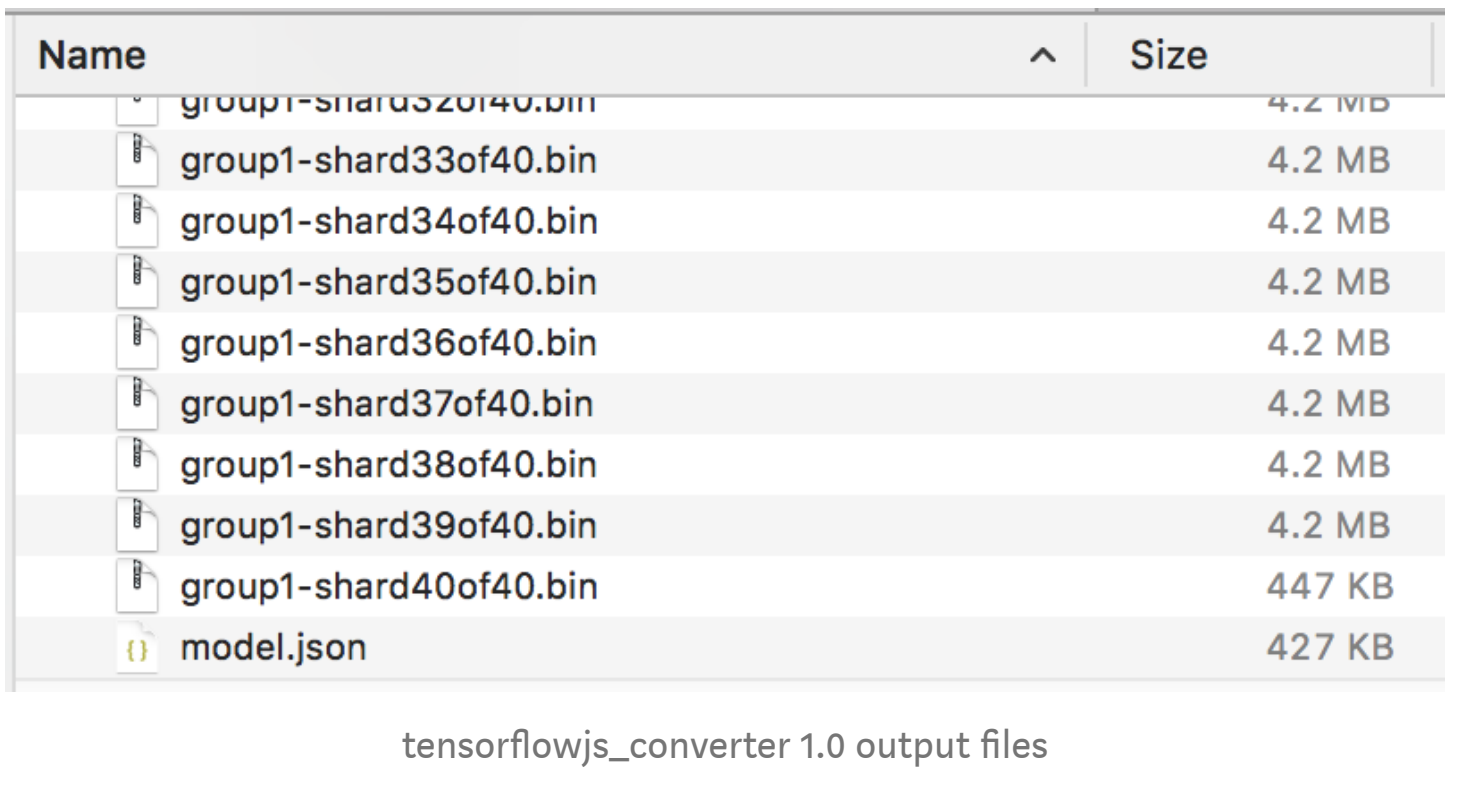
install tensorflowjs using pip

The `tensorflowjs_converter` expects the model and the output directory as inputs. You can also pass optional parameters to further customize the conversion process.

```
tensorflowjs_converter \
    --input_format=tf_frozen_model \
    --output_node_names='SemanticPredictions' \
    /path/to/frozen_inference_graph.pb \
    /path/to/output_dir
```

The output of `tensorflowjs_converter` is a set of files:

- `model.json` — the dataflow graph
- A group of binary weight files called shards. Each shard file is small in size for easier browser caching. And the number of shards depends on the initial model.



Name	Size
group1-shard33of40.bin	4.2 MB
group1-shard34of40.bin	4.2 MB
group1-shard35of40.bin	4.2 MB
group1-shard36of40.bin	4.2 MB
group1-shard37of40.bin	4.2 MB
group1-shard38of40.bin	4.2 MB
group1-shard39of40.bin	4.2 MB
group1-shard40of40.bin	447 KB
model.json	427 KB

tensorflowjs_converter 1.0 output files

***NOTE:** If using `tensorflowjs_converter` version before 1.0, the output produced includes the graph (`tensorflowjs_model.pb`), weights manifest (`weights_manifest.json`), and the binary shards files.*

Run model run

Once converted, the model is ready to load into TensorFlow.js for predictions.

Using Tensorflow.js version 0.x.x:

```
// tensorflow.js 0.x.x
const MODEL_URL = '/model/tensorflowjs_model.pb'
const WEIGHTS_URL = '/model/weights_manifest.json'

// https://js.tensorflow.org/api/0.15.1/#loadFrozenModel
const model = await tf.loadFrozenModel(MODEL_URL, WEIGHTS_URL)
```

Using TensorFlow.js version 1.x.x:

```
// tensorflow.js 1.0.0
const MODEL_URL = '/model/model.json'

// https://js.tensorflow.org/api/1.0.0/#loadGraphModel
const model = await tf.loadGraphModel(MODEL_URL)
```

The imported model is the same as models trained and created with TensorFlow.js.

Convert all models?

You may find it tempting to grab any and all models, convert them to the web friendly format, and run them in the browser. But this is not always possible or recommended. There are several factors for you to keep in mind.

The `tensorflowjs_converter` command can only convert Keras and TensorFlow models. Some supported model formats include SavedModel, Frozen Model, and HDF5.

TensorFlow.js does not support all TensorFlow operations. It currently has a limited set of supported operations. As a result, the converter will fail if the model contains operations not supported.

Thinking and treating the model as a black box is not always enough. Because you can get the model converted and produce a web friendly model does not mean all is well.

Depending on a model's size or architecture, its performance could be less than desirable. Further optimization of the model is often required. In most cases, you will have to pre-process the input(s) to the model, as well as, process the model output(s). So, needing some understanding or inner workings of the model is almost a given.

Getting to know your model

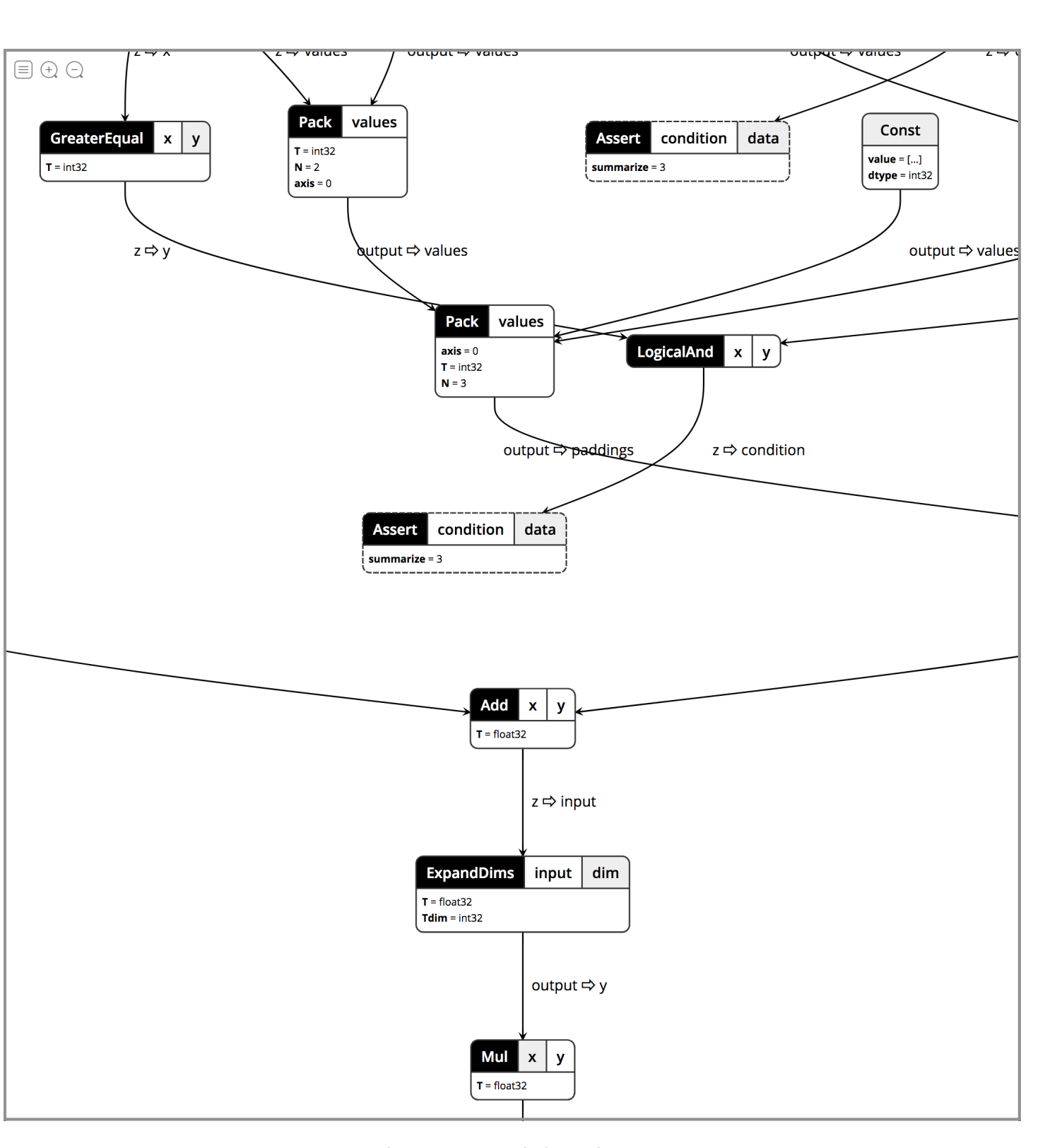
Presumably you have a model available to you. If not, resources exist with an ever growing collection of pre-trained models. A couple of them include:

- TensorFlow Models —a set of official and research models implemented in TensorFlow
- Model Asset Exchange —a set of deep learning models covering different frameworks

These resources provide the model for you to download. They also can include information about the model, useful assets, and links to learn more.

You can review a model with tools such as TensorBoard. It's graph visualization can help you better understand the model.

Another option is Netron, a visualizer for deep learning and machine learning models. It provides an overview of the graph and you can inspect the model's operations.



visualizing a model with Netron

To be continued...

Stay tuned for the follow up to this article to learn how to pull this all together. You will step through this process in greater detail with an actual model and you will take a pre-trained model into web friendly format and end up with a web application.

Thanks to Nick Kasten and Maureen McElaney.

Machine Learning TensorFlow JavaScript Open Source Python

WRITTEN BY
va barbosa
crioulo de cova rodela
Follow