TOR Tutorial for Model Training and Inference

Platform: Linux (tested on CentOS 7 and Ubuntu 18.04)

NOTE: You do not need to have root permissions to do anything, as long as you are able to get Anaconda installed on the machine you're working on (that is, as long as you get the conda setup script on your machine somehow)

Setup

Assuming Anaconda 3 is already installed, first create a new conda environment called `tor` to hold all dependencies of this project.

```
conda create --name tor python=3.6
```

Everything you do will be on this environment, so to activate it you will need to run:

```
conda activate tor
```

Using conda, install the dependencies by running the following commands:

```
conda install protobuf
conda install lxml
conda install Cython
conda install jupyter
conda install matplotlib
conda install pandas
pip install opencv-python
conda install tensorflow=1.15
# conda install tensorflow-gpu=1.15 if you want to use a gpu installation
instead
```

Download the Tensorflow Object Detection API from TOR's fork pranaymethuku/models of tensorflow/models.

Although we need the entire models folder, our main focus is the `models/research/object_detection` directory. The next step is to compile the protobuf files within the `models/research` directory:

```
cd <path_to_your_tensorflow_installation>/models/research/
protoc object_detection/protos/*.proto --python_out=.
```

While still in the models/research folder, set the PYTHONPATH variable.

```
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

You can test your setup using the following command:

```
python object_detection/builders/model_builder_test.py
```

Which should show something like the following output:

```
...............
----------------------------------------------------------------------
Ran 15 tests in 0.123sOK
```

Now you have everything set up!

Project Structure

The rest of the steps will be executed from within the `models/research/object_detection` directory. In this directory, we have augmented the framework by creating some new directories, namely:

- `pre_trained_models` - where all pre-trained models from the model zoo will be stored.
- `tor_models` - which will hold all newly trained models within the following sub-directories:
  - `tier_1/` - will hold models for Tier 1
  - `tier_2/` - will hold models for Tier 2

  ... and so on.
- `pre_training_scripts` - where all scripts needed before running the trining script will be stored.
- `inference_scripts` - where all scripts needed to run inference will be stored.

Essentially, the directory structure under `object_detection/` would look something like this:

```
├── tor_models
│   ├── tier_1
│   │   ├── tier_1_faster_rcnn_inception_v2_coco_2018_01_28
│   │   │   ├── training
│   │   │   ├── faster_rcnn_inception_v2_coro.config
│   │   │   └── ...
│   │   ├── tier_1_ssd_inception_v2_coco_2018_01_28
│   │   │   └── ...
│   │   └── ...
│   ├── tier_2
│   │   └── ...
│   └── ...
├── pre_trained_models
│   ├── faster_rcnn_inception_v2_coco_2018_01_28
│   │   ├── model.ckpt.meta
│   │   ├── model.ckpt.index
│   │   ├── saved_model
│   │   │   └── ...
│   │   └── ...
│   └── ...
├── pre_training_scripts
│   ├── xml_to_csv.py
│   ├── generate_tfrecord.py
│   └── ...
├── inference_scripts
│   ├── detection.py
│   ├── gui.py
│   └── ...
└── ...
```

Pre-training

For the purposes of this tutorial, we will train a **Tier 2** model. Furthermore, we will skip the data-collection/post-processing/labeling step and assume you have `train_data` and `test_data` directories already created with correspondingly labeled .xml files for your .jpgs, as well as an appropriately configured `labelmap.pbtxt` file, all in the `tor_models/tier_2` directory.

In our case, the `labelmap.pbtxt` file looks something like this:

```
item {
  id: 1
  name: 'Sedan'
}

item {
  id: 2
  name: 'Hatchback'
}

item {
  id: 3
  name: 'SUV'
}

item {
  id: 4
  name: 'Truck'
}
```

Run the `xml_to_csv.py` script from the `pre_training_scripts` directory to generate labeled .csv files for `train_data` and `test_data`

```
python xml_to_csv.py --source=<path_to_train> --csv-file=
<path_to_train_labels.csv>
```

In this case,

```
python pre_training_scripts/xml_to_csv.py --
source=tor_models/tier_2/train_data --csv-
file=tor_models/tier_2/train_labels.csv
python pre_training_scripts/xml_to_csv.py --
source=tor_models/tier_2/test_data --csv-
file=tor_models/tier_2/test_labels.csv
```

Next, run the `generate_tfrecord.py` script from the `pre_training_scripts` directory to generate .tfrecord files from the generated .csv files for `train_data/` and `test_data/`

```
python generate_tfrecord.py -c=<path_to_train_labels.csv> -r=
<path_to_train.tfrecord> -i=<path_to_train> -l=<path_to_labelmap.pbtxt>
```

In this case,

```
python pre_training_scripts/generate_tfrecord.py -
c=tor_models/tier_2/train_labels.csv -r=tor_models/tier_2/train.tfrecord -
i=tor_models/tier_2/train_data -l=tor_models/tier_2/labelmap.pbtxt
python pre_training_scripts/generate_tfrecord.py -
c=tor_models/tier_2/test_labels.csv -r=tor_models/tier_2/test.tfrecord -
i=tor_models/tier_2/test_data -l=tor_models/tier_2/labelmap.pbtxt
```

Download a pre-trained model file (from the model zoo). For the purposes of this tutorial we'll download faster_rcnn_inception_v2_coco and store the extracted `faster_rcnn_inception_v2_coco_2018_01_28` folder in `pre_trained_models/`.

```
wget
http://download.tensorflow.org/models/object_detection/faster_rcnn_inceptio
n_v2_coco_2018_01_28.tar.gz
tar -xvzf faster_rcnn_inception_v2_coco_2018_01_28.tar.gz
```

Next, we will create a pre-trained model specific directory called `tier_2_faster_rcnn_inception_v2_coco_2018_01_28` under the `tier_2` directory. This way we can train different Tier 2 models with different pre-trained models. Copy the corresponding .config file (`faster_rcnn_inception_v2_coco.config`) from `object_detection/samples/config/` into the `tier_2_faster_rcnn_inception_v2_coco_2018_01_28/` directory.

Next, we will modify the .config file.

NOTE: The paths must be entered with single forward slashes (*NOT* backslashes), or TensorFlow will give a file path error when trying to train the model. Also, the paths must be **absolute** and in double quotation marks ("), not single quotation marks (').

1. Modify the `num_classes` to the number of classes you're using. In this case we will go with 4.

2. Modify the absolute path to the `fine_tune_checkpoint` to be the checkpoint from the downloaded model folder. In our case, "/udrive/student/ksmith012007/models/research/object_detection/pre_trained_models/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt". It is important to note that `model.ckpt` file does not actually exist, only `model.ckpt.index` and `model.ckpt.meta` exist, but do not worry about that.

3. Modify the absolute paths to the `input_path` and `label_map_path` attributes file under the `train_input_reader`, `eval_input_reader`, and `label_map_path` attributes, to point to the corresponding .tfrecord and .pbtxt files. In our case,

```
train_input_reader: {
  tf_record_input_reader {
    input_path:
"/udrive/student/ksmith012007/models/research/object_detection/tor_models/tier_2/train.tfrecord"
  }
  label_map_path:
"/udrive/student/ksmith012007/models/research/object_detection/tor_models/tier_2/labelmap.pbtxt"
}
```

and

```
eval_input_reader: {
  tf_record_input_reader {
    input_path:
"/udrive/student/ksmith012007/models/research/object_detection/tor_models/tier_2/test.tfrecord"
  }
  label_map_path:
"/udrive/student/ksmith012007/models/research/object_detection/tor_models/tier_2/labelmap.pbtxt"
  shuffle: false
  num_readers: 1
}
```

Lastly, we will be creating a directory called `training` under `tier_2_faster_rcnn_inception_v2_coco_2018_01_28` where we will store the checkpoint data.

Training

Now that we have everything we need to run the training script, `model_main.py` under the `object_detection` directory. Before we run this file, we need to modify it a bit to add some more configurations. Specifically, we need to modify the default `save_checkpoints_steps` and `keep_checkpoint_max` parameters of the `tf.estimator.RunConfig` function to save checkpoints every 1000 steps and keep the best 200 checkpoints.

```
def main(unused_argv):
  flags.mark_flag_as_required('model_dir')
  flags.mark_flag_as_required('pipeline_config_path')
  # update the next line
  config = tf.estimator.RunConfig(model_dir=FLAGS.model_dir,
save_checkpoints_steps=1000, keep_checkpoint_max=200)
  # rest of the code ...
```

Once that is done, we can run the training script from the `object_detection` directory using the earlier `faster_rcnn_inception_v2_coco.config` file and the `tier_2_faster_rcnn_inception_v2_coco_2018_01_28/training` directory we created as follows:

```
python model_main.py --
model_dir=tor_models/tier_2/tier_2_faster_rcnn_inception_v2_coco_2018_01_28
/training --
pipeline_config_path=tor_models/tier_2/tier_2_faster_rcnn_inception_v2_coco
_2018_01_28/faster_rcnn_inception_v2_coco.config --num_train_steps=200000
```

Based on the --num_train_steps parameter, the training will run for 200000 steps.

With this step, the training has started, and your output will look something like this. Of course, the below output is just an intermediate snippet while training.

```
INFO:tensorflow:loss = 0.10141424, step = 6000
I0407 22:51:33.054176 139827692275520 basic_session_run_hooks.py:262] loss
= 0.10141424, step = 6000
INFO:tensorflow:global_step/sec: 1.30346
I0407 22:52:49.771748 139827692275520 basic_session_run_hooks.py:692]
global_step/sec: 1.30346
INFO:tensorflow:loss = 0.16542536, step = 6100 (76.720 sec)
I0407 22:52:49.773873 139827692275520 basic_session_run_hooks.py:260] loss
= 0.16542536, step = 6100 (76.720 sec)
INFO:tensorflow:global_step/sec: 2.17699
I0407 22:53:35.706618 139827692275520 basic_session_run_hooks.py:692]
global_step/sec: 2.17699
INFO:tensorflow:loss = 0.07526648, step = 6200 (45.938 sec)
```

```
I0407 22:53:35.711387 139827692275520 basic_session_run_hooks.py:260] loss
= 0.07526648, step = 6200 (45.938 sec)
INFO:tensorflow:global_step/sec: 1.43452
I0407 22:54:45.416280 139827692275520 basic_session_run_hooks.py:692]
global_step/sec: 1.43452
INFO:tensorflow:loss = 0.026271442, step = 6300 (69.707 sec)
I0407 22:54:45.418284 139827692275520 basic_session_run_hooks.py:260] loss
= 0.026271442, step = 6300 (69.707 sec)
INFO:tensorflow:global_step/sec: 1.90723
I0407 22:55:37.848028 139827692275520 basic_session_run_hooks.py:692]
global_step/sec: 1.90723
INFO:tensorflow:loss = 0.085402355, step = 6400 (52.433 sec)
I0407 22:55:37.851746 139827692275520 basic_session_run_hooks.py:260] loss
= 0.085402355, step = 6400 (52.433 sec)
INFO:tensorflow:global_step/sec: 2.32246
```

You can view the progress of the training job by using TensorBoard. To do this, open a terminal and navigate to the `object_detection` directory, and run the following command:

```
tensorboard --
logdir=tor_models/tier_2/tier_2_faster_rcnn_inception_v2_coco_2018_01_28/tr
aining
```

Model Export

Now that we have a trained model we need to generate an inference graph, which can be used to run the model. For doing so we need to first of find out the highest saved step number. For this, we need to navigate to the `tier_2_faster_rcnn_inception_v2_coco_2018_01_28/training` directory and look for the model.ckpt file with the biggest index. Let's call that index XXXX. Next, we need to run the `export_inference_graph.py` as follows:

```
python export_inference_graph.py --input_type=image_tensor --
pipeline_config_path=tor_models/tier_2/tier_2_faster_rcnn_inception_v2_coco
_2018_01_28/faster_rcnn_inception_v2_coco.config --
trained_checkpoint_prefix=tor_models/tier_2/tier_2_faster_rcnn_inception_v2
_coco_2018_01_28/training/model.ckpt-XXXX --
output_directory=tor_models/tier_2/tier_2_faster_rcnn_inception_v2_coco_201
8_01_28/inference_graph
```

This will generate the `inference_graph/` in the `tier_2_faster_rcnn_inception_v2_coco_2018_01_28/training` directory, which contains a `frozen_inference_graph.pb`.

Inference

---TODO---

Resources

- Custom Object Detection using TensorFlow from Scratch
- ML | Training Image Classifier using Tensorflow Object Detection API
- TensorFlow Object Detection API tutorial