Configuring the Object Detection Training Pipeline

Overview

The Tensorflow Object Detection API uses protobuf files to configure the training and evaluation process. The schema for the training pipeline can be found in object_detection/protos/pipeline.proto. At a high level, the config file is split into 5 parts:

1. The `model` configuration. This defines what type of model will be trained (ie. meta-architecture, feature extractor).
2. The `train_config`, which decides what parameters should be used to train model parameters (ie. SGD parameters, input preprocessing and feature extractor initialization values).
3. The `eval_config`, which determines what set of metrics will be reported for evaluation (currently we only support the PASCAL VOC metrics).
4. The `train_input_config`, which defines what dataset the model should be trained on.
5. The `eval_input_config`, which defines what dataset the model will be evaluated on. Typically this should be different than the training input dataset.

A skeleton configuration file is shown below:

```
model {
(... Add model config here...)
}

train_config : {
(... Add train_config here...)
}

train_input_reader: {
(... Add train_input configuration here...)
}

eval_config: {
}

eval_input_reader: {
(... Add eval_input configuration here...)
}
```

Picking Model Parameters

There are a large number of model parameters to configure. The best settings will depend on your given application. Faster R-CNN models are better suited to cases where high accuracy is desired and latency is of lower priority. Conversely, if processing time is the most important factor, SSD models are recommended. Read our paper for a more detailed discussion on the speed vs accuracy tradeoff.

To help new users get started, sample model configurations have been provided in the object_detection/samples/model_configs folder. The contents of these configuration files can be pasted into `model` field of the skeleton configuration. Users should note that the `num_classes` field should be changed to a value suited for the dataset the user is training on.

Defining Inputs

The Tensorflow Object Detection API accepts inputs in the TFRecord file format. Users must specify the locations of both the training and evaluation files. Additionally, users should also specify a label map, which define the mapping between a class id and class name. The label map should be identical between training and evaluation datasets.

An example input configuration looks as follows:

```
tf_record_input_reader {
  input_path: "/usr/home/username/data/train.record"
}
label_map_path: "/usr/home/username/data/label_map.pbtxt"
```

Users should substitute the `input_path` and `label_map_path` arguments and insert the input configuration into the `train_input_reader` and `eval_input_reader` fields in the skeleton configuration. Note that the paths can also point to Google Cloud Storage buckets (ie. "gs://project_bucket/train.record") for use on Google Cloud.

Configuring the Trainer

The `train_config` defines parts of the training process:

1. Model parameter initialization.
2. Input preprocessing.
3. SGD parameters.

A sample `train_config` is below:

```
batch_size: 1
optimizer {
  momentum_optimizer: {
    learning_rate: {
      manual_step_learning_rate {
        initial_learning_rate: 0.0002
        schedule {
          step: 0
          learning_rate: .0002
        }
        schedule {
          step: 900000
          learning_rate: .00002
        }
        schedule {
          step: 1200000
          learning_rate: .000002
        }
      }
    }
    momentum_optimizer_value: 0.9
  }
  use_moving_average: false
}
fine_tune_checkpoint: "/usr/home/username/tmp/model.ckpt-#####"
from_detection_checkpoint: true
gradient_clipping_by_norm: 10.0
data_augmentation_options {
  random_horizontal_flip {
  }
}
```

Model Parameter Initialization

While optional, it is highly recommended that users utilize other object detection checkpoints. Training an object detector from scratch can take days. To speed up the training process, it is recommended that users re-use the feature extractor parameters from a pre-existing object classification or detection checkpoint. `train_config` provides two fields to specify pre-existing checkpoints: `fine_tune_checkpoint` and `from_detection_checkpoint`. `fine_tune_checkpoint` should provide a path to the pre-existing checkpoint (ie:"/usr/home/username/checkpoint/model.ckpt-#####"). `from_detection_checkpoint` is a boolean value. If false, it assumes the checkpoint was from an object classification checkpoint. Note that starting from a detection checkpoint will usually result in a faster training job than a classification

checkpoint.

The list of provided checkpoints can be found here.

Input Preprocessing

The `data_augmentation_options` in `train_config` can be used to specify how training data can be modified. This field is optional.

SGD Parameters

The remainings parameters in `train_config` are hyperparameters for gradient descent. Please note that the optimal learning rates provided in these configuration files may depend on the specifics of the training setup (e.g. number of workers, gpu type).

Configuring the Evaluator

Currently evaluation is fixed to generating metrics as defined by the PASCAL VOC challenge. The parameters for `eval_config` are set to reasonable defaults and typically do not need to be configured.