Running DeepLab on ADE20K Semantic Segmentation Dataset

This page walks through the steps required to run DeepLab on ADE20K dataset on a local machine.

Download dataset and convert to TFRecord

We have prepared the script (under the folder `datasets`) to download and convert ADE20K semantic segmentation dataset to TFRecord.

```
# From the tensorflow/models/research/deeplab/datasets directory.
bash download_and_convert_ade20k.sh
```

The converted dataset will be saved at ./deeplab/datasets/ADE20K/tfrecord

Recommended Directory Structure for Training and Evaluation

```
+ datasets
   - build_data.py
   - build_ade20k_data.py
   - download_and_convert_ade20k.sh
   + ADE20K
     + tfrecord
    + exp
      + train_on_train_set
        + train
        + eval
        + vis
     + ADEChallengeData2016
       + annotations
         + training
         + validation
       + images
         + training
         + validation
```

where the folder `train_on_train_set` stores the train/eval/vis events and results (when training DeepLab on the ADE20K train set).

Running the train/eval/vis jobs

A local training job using `xception_65` can be run with the following command:

```
# From tensorflow/models/research/
python deeplab/train.py \
    --logtostderr \
    --training_number_of_steps=150000 \
    --train_split="train" \
```

```
    --model_variant="xception_65" \
    --atrous_rates=6 \
    --atrous_rates=12 \
    --atrous_rates=18 \
    --output_stride=16 \
    --decoder_output_stride=4 \
    --train_crop_size="513,513" \
    --train_batch_size=4 \
    --min_resize_value=513 \
    --max_resize_value=513 \
    --resize_factor=16 \
    --dataset="ade20k" \
    --tf_initial_checkpoint=${PATH_TO_INITIAL_CHECKPOINT} \
    --train_logdir=${PATH_TO_TRAIN_DIR}\
    --dataset_dir=${PATH_TO_DATASET}
```

where ${PATH_TO_INITIAL_CHECKPOINT} is the path to the initial checkpoint. ${PATH_TO_TRAIN_DIR} is the directory in which training checkpoints and events will be written to (it is recommended to set it to the `train_on_train_set/train` above), and ${PATH_TO_DATASET} is the directory in which the ADE20K dataset resides (the `tfrecord` above)

Note that for train.py:

1. In order to fine tune the BN layers, one needs to use large batch size (> 12), and set fine_tune_batch_norm = True. Here, we simply use small batch size during training for the purpose of demonstration. If the users have limited GPU memory at hand, please fine-tune from our provided checkpoints whose batch norm parameters have been trained, and use smaller learning rate with fine_tune_batch_norm = False.
2. User should fine tune the `min_resize_value` and `max_resize_value` to get better result. Note that `resize_factor` has to be equal to `output_stride`.
3. The users should change atrous_rates from [6, 12, 18] to [12, 24, 36] if setting output_stride=8.
4. The users could skip the flag, `decoder_output_stride`, if you do not want to use the decoder structure.

Running Tensorboard

Progress for training and evaluation jobs can be inspected using Tensorboard. If using the recommended directory structure, Tensorboard can be run using the following command:

```
tensorboard --logdir=${PATH_TO_LOG_DIRECTORY}
```

where `${PATH_TO_LOG_DIRECTORY}` points to the directory that contains the train directorie (e.g., the folder `train_on_train_set` in the above example). Please note it may take Tensorboard a couple minutes to populate with data.