TPU compatible detection pipelines

[TOC]

The TensorFlow Object Detection API supports TPU training for some models. To make models TPU compatible you need to make a few tweaks to the model config as mentioned below. We also provide several sample configs that you can use as a template.

TPU compatibility

Static shaped tensors

TPU training currently requires all tensors in the TensorFlow Graph to have static shapes. However, most of the sample configs in Object Detection API have a few different tensors that are dynamically shaped. Fortunately, we provide simple alternatives in the model configuration that modifies these tensors to have static shape:

- **Image tensors with static shape** - This can be achieved either by using a `fixed_shape_resizer` that resizes images to a fixed spatial shape or by setting `pad_to_max_dimension: true` in `keep_aspect_ratio_resizer` which pads the resized images with zeros to the bottom and right. Padded image tensors are correctly handled internally within the model.

  ```
  image_resizer {
    fixed_shape_resizer {
      height: 640
      width: 640
    }
  }
  ```

  or

  ```
  image_resizer {
    keep_aspect_ratio_resizer {
      min_dimension: 640
      max_dimension: 640
      pad_to_max_dimension: true
    }
  }
  ```

- **Groundtruth tensors with static shape** - Images in a typical detection dataset have variable number of groundtruth boxes and associated classes. Setting `max_number_of_boxes` to a large enough number in `train_config` pads the groundtruth tensors with zeros to a static shape. Padded groundtruth tensors are correctly handled internally within the model.

  ```
  train_config: {
  ```

```
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED/model.ckpt"
  batch_size: 64
  max_number_of_boxes: 200
  unpad_groundtruth_tensors: false
}
```

TPU friendly ops

Although TPU supports a vast number of tensorflow ops, a few used in the TensorFlow Object Detection API are unsupported. We list such ops below and recommend compatible substitutes.

- **Anchor sampling** - Typically we use hard example mining in standard SSD pipeliens to balance positive and negative anchors that contribute to the loss. Hard Example mining uses non max suppression as a subroutine and since non max suppression is not currently supported on TPUs we cannot use hard example mining. Fortunately, we provide an implementation of focal loss that can be used instead of hard example mining. Remove `hard_example_miner` from the config and substitute `weighted_sigmoid` classification loss with `weighted_sigmoid_focal` loss.

  ```
  loss {
    classification_loss {
      weighted_sigmoid_focal {
        alpha: 0.25
        gamma: 2.0
      }
    }
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  ```

- **Target Matching** - Object detection API provides two choices for matcher used in target assignment: `argmax_matcher` and `bipartite_matcher`. Bipartite matcher is not currently supported on TPU, therefore we must modify the configs to use `argmax_matcher`. Additionally, set `use_matmul_gather: true` for efficiency on TPU.

  ```
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
      ignore_thresholds: false
      negatives_lower_than_unmatched: true
  ```

```
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
```

TPU training hyperparameters

Object Detection training on TPU uses synchronous SGD. On a typical cloud TPU with 8 cores we recommend batch sizes that are 8x large when compared to a GPU config that uses asynchronous SGD. We also use fewer training steps (~ 1/100 x) due to the large batch size. This necessitates careful tuning of some other training parameters as listed below.

- **Batch size** - Use the largest batch size that can fit on cloud TPU.

  ```
  train_config {
    batch_size: 1024
  }
  ```

- **Training steps** - Typically only 10s of thousands.

  ```
  train_config {
    num_steps: 25000
  }
  ```

- **Batch norm decay** - Use smaller decay constants (0.97 or 0.997) since we take fewer training steps.

  ```
  batch_norm {
    scale: true,
    decay: 0.97,
    epsilon: 0.001,
  }
  ```

- **Learning rate** - Use large learning rate with warmup. Scale learning rate linearly with batch size. See `cosine_decay_learning_rate` or `manual_step_learning_rate` for examples.

  ```
  learning_rate: {
    cosine_decay_learning_rate {
      learning_rate_base: .04
      total_steps: 25000
      warmup_learning_rate: .013333
      warmup_steps: 2000
    }
  }
  ```

  or

  ```
   learning_rate: {
    manual_step_learning_rate {
  ```

```
      warmup: true
      initial_learning_rate: .01333
      schedule {
        step: 2000
        learning_rate: 0.04
      }
      schedule {
        step: 15000
        learning_rate: 0.004
      }
    }
  }
```

Example TPU compatible configs

We provide example config files that you can use to train your own models on TPU

- ssd_mobilenet_v1_300x300
- ssd_mobilenet_v1_ppn_300x300
- ssd_mobilenet_v1_fpn_640x640 (mobilenet based retinanet)
- ssd_resnet50_v1_fpn_640x640 (retinanet)

Supported Meta architectures

Currently, `SSDMetaArch` models are supported on TPUs. `FasterRCNNMetaArch` is going to be supported soon.