

Object Detection Models on TensorFlow 2

WARNING: This repository will be deprecated and replaced by the solid implementations inside `vision/beta/`.

Prerequisite

To get started, download the code from TensorFlow models GitHub repository or use the pre-installed Google Cloud VM.

```
git clone https://github.com/tensorflow/models.git
```

Next, make sure to use TensorFlow 2.1+ on Google Cloud. Also here are a few package you need to install to get started:

```
sudo apt-get install -y python-tk && \
pip3 install -r ~/models/official/requirements.txt
```

Train RetinaNet on TPU

Train a vanilla ResNet-50 based RetinaNet.

```
TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
RESNET_CHECKPOINT="<path to the pre-trained Resnet-50 checkpoint>"
TRAIN_FILE_PATTERN="<path to the TFRecord training data>"
EVAL_FILE_PATTERN="<path to the TFRecord validation data>"
VAL_JSON_FILE="<path to the validation annotation JSON file>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu="${TPU_NAME?}" \
  --model_dir="${MODEL_DIR?}" \
  --mode=train \
  --params_override="{ type: retinanet, train: { checkpoint: { path:
${RESNET_CHECKPOINT?}, prefix: resnet50/ }, train_file_pattern:
${TRAIN_FILE_PATTERN?} }, eval: { val_json_file: ${VAL_JSON_FILE?},
eval_file_pattern: ${EVAL_FILE_PATTERN?} } }"
```

The pre-trained ResNet-50 checkpoint can be downloaded [here](#).

Note: The ResNet implementation under [detection/](#) is currently different from the one under [classification/](#), so the checkpoints are not compatible. We will unify the implementation soon.

Train a SpineNet-49 based RetinaNet.

```
TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
TRAIN_FILE_PATTERN="<path to the TFRecord training data>"
EVAL_FILE_PATTERN="<path to the TFRecord validation data>"
```

```

VAL_JSON_FILE="<path to the validation annotation JSON file>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu="${TPU_NAME?}" \
  --model_dir="${MODEL_DIR?}" \
  --mode=train \
  --params_override="{ type: retinanet, architecture: {backbone: spinenet,
multilevel_features: identity}, spinenet: {model_id: 49}, train_file_pattern:
${TRAIN_FILE_PATTERN?} }, eval: { val_json_file: ${VAL_JSON_FILE?},
eval_file_pattern: ${EVAL_FILE_PATTERN?} } }"

```

Train a custom RetinaNet using the config file.

First, create a YAML config file, e.g. *my_retinanet.yaml*. This file specifies the parameters to be overridden, which should at least include the following fields.

```

# my_retinanet.yaml
type: 'retinanet'
train:
  train_file_pattern: <path to the TFRecord training data>
eval:
  eval_file_pattern: <path to the TFRecord validation data>
  val_json_file: <path to the validation annotation JSON file>

```

Once the YAML config file is created, you can launch the training using the following command.

```

TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu="${TPU_NAME?}" \
  --model_dir="${MODEL_DIR?}" \
  --mode=train \
  --config_file="my_retinanet.yaml"

```

Train RetinaNet on GPU

Training on GPU is similar to that on TPU. The major change is the strategy type (use "[mirrored](#)" for multiple GPU and "[one device](#)" for single GPU).

Multi-GPUs example (assuming there are 8GPU connected to the host):

```

MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=mirrored \
  --num_gpus=8 \
  --model_dir="${MODEL_DIR?}" \
  --mode=train \
  --config_file="my_retinanet.yaml"

```

```

MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=one_device \
  --num_gpus=1 \
  --model_dir="${MODEL_DIR}" \
  --mode=train \
  --config_file="my_retinanet.yaml"

```

An example with inline configuration (YAML or JSON format):

```

python3 ~/models/official/legacy/detection/main.py \
  --model_dir=<model folder> \
  --strategy_type=one_device \
  --num_gpus=1 \
  --mode=train \
  --params_override="eval:
eval_file_pattern: <Eval TFRecord file pattern>
batch_size: 8
val_json_file: <COCO format groundtruth JSON file>
predict:
  predict_batch_size: 8
architecture:
  use_bfloat16: False
train:
  total_steps: 1
  batch_size: 8
  train_file_pattern: <Eval TFRecord file pattern>
use_tpu: False
"

```

Train Mask R-CNN on TPU

Train a vanilla ResNet-50 based Mask R-CNN.

```

TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
RESNET_CHECKPOINT="<path to the pre-trained Resnet-50 checkpoint>"
TRAIN_FILE_PATTERN="<path to the TFRecord training data>"
EVAL_FILE_PATTERN="<path to the TFRecord validation data>"
VAL_JSON_FILE="<path to the validation annotation JSON file>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu=${TPU_NAME} \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=mask_rcnn \
  --params_override="{train: { checkpoint: { path: ${RESNET_CHECKPOINT}, prefix:
resnet50/ }, train_file_pattern: ${TRAIN_FILE_PATTERN} }, eval: { val_json_file:
${VAL_JSON_FILE}, eval_file_pattern: ${EVAL_FILE_PATTERN} } }"

```

The pre-trained ResNet-50 checkpoint can be downloaded [here](#).

Note: The ResNet implementation under [detection/](#) is currently different from the one under [classification/](#), so the checkpoints are not compatible. We will unify the implementation soon.

Train a SpineNet-49 based Mask R-CNN.

```
TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
TRAIN_FILE_PATTERN="<path to the TFRecord training data>"
EVAL_FILE_PATTERN="<path to the TFRecord validation data>"
VAL_JSON_FILE="<path to the validation annotation JSON file>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu="${TPU_NAME}" \
  --model_dir="${MODEL_DIR}" \
  --mode=train \
  --model=mask_rcnn \
  --params_override="{architecture: {backbone: spinenet, multilevel_features:
identity}, spinenet: {model_id: 49}, train_file_pattern: ${TRAIN_FILE_PATTERN} },
eval: { val_json_file: ${VAL_JSON_FILE}, eval_file_pattern: ${EVAL_FILE_PATTERN} }
}"
```

Train a custom Mask R-CNN using the config file.

First, create a YAML config file, e.g. *my_maskrcnn.yaml*. This file specifies the parameters to be overridden, which should at least include the following fields.

```
# my_maskrcnn.yaml
train:
  train_file_pattern: <path to the TFRecord training data>
eval:
  eval_file_pattern: <path to the TFRecord validation data>
  val_json_file: <path to the validation annotation JSON file>
```

Once the YAML config file is created, you can launch the training using the following command.

```
TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu=${TPU_NAME} \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=mask_rcnn \
  --config_file="my_maskrcnn.yaml"
```

Train Mask R-CNN on GPU

Training on GPU is similar to that on TPU. The major change is the strategy type (use "[mirrored](#)" for multiple GPU and "[one_device](#)" for single GPU).

Multi-GPUs example (assuming there are 8GPU connected to the host):

```
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=mirrored \
  --num_gpus=8 \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=mask_rcnn \
  --config_file="my_maskrcnn.yaml"
```

```
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=one_device \
  --num_gpus=1 \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=mask_rcnn \
  --config_file="my_maskrcnn.yaml"
```

An example with inline configuration (YAML or JSON format):

```
python3 ~/models/official/legacy/detection/main.py \
  --model_dir=<model folder> \
  --strategy_type=one_device \
  --num_gpus=1 \
  --mode=train \
  --model=mask_rcnn \
  --params_override="eval:
eval_file_pattern: <Eval TFRecord file pattern>
batch_size: 8
val_json_file: <COCO format groundtruth JSON file>
predict:
  predict_batch_size: 8
architecture:
  use_bfloat16: False
train:
  total_steps: 1000
  batch_size: 8
  train_file_pattern: <Eval TFRecord file pattern>
use_tpu: False
"
```

Train ShapeMask on TPU

Train a ResNet-50 based ShapeMask.

```

TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
RESNET_CHECKPOINT="<path to the pre-trained Resnet-50 checkpoint>"
TRAIN_FILE_PATTERN="<path to the TFRecord training data>"
EVAL_FILE_PATTERN="<path to the TFRecord validation data>"
VAL_JSON_FILE="<path to the validation annotation JSON file>"
SHAPE_PRIOR_PATH="<path to shape priors>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu=${TPU_NAME} \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=shapemask \
  --params_override="{train: { checkpoint: { path: ${RESNET_CHECKPOINT}, prefix:
resnet50/ }, train_file_pattern: ${TRAIN_FILE_PATTERN} }, eval: { val_json_file:
${VAL_JSON_FILE}, eval_file_pattern: ${EVAL_FILE_PATTERN} } shapemask_head:
{use_category_for_mask: true, shape_prior_path: ${SHAPE_PRIOR_PATH}} }"

```

The pre-trained ResNet-50 checkpoint can be downloaded [here](#).

The shape priors can be downloaded [here] (https://storage.googleapis.com/cloud-tpu-checkpoints/shapemask/kmeans_class_priors_91x20x32x32.npy)

Train a custom ShapeMask using the config file.

First, create a YAML config file, e.g. *my_shapemask.yaml*. This file specifies the parameters to be overridden:

```

# my_shapemask.yaml
train:
  train_file_pattern: <path to the TFRecord training data>
  total_steps: <total steps to train>
  batch_size: <training batch size>
eval:
  eval_file_pattern: <path to the TFRecord validation data>
  val_json_file: <path to the validation annotation JSON file>
  batch_size: <evaluation batch size>
shapemask_head:
  shape_prior_path: <path to shape priors>

```

Once the YAML config file is created, you can launch the training using the following command.

```

TPU_NAME="<your GCP TPU name>"
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
  --tpu=${TPU_NAME} \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=shapemask \
  --config_file="my_shapemask.yaml"

```

Train ShapeMask on GPU

Training on GPU is similar to that on TPU. The major change is the strategy type (use "[mirrored](#)" for multiple GPU and "[one_device](#)" for single GPU).

Multi-GPUs example (assuming there are 8GPU connected to the host):

```
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=mirrored \
  --num_gpus=8 \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=shapemask \
  --config_file="my_shapemask.yaml"
```

A single GPU example

```
MODEL_DIR="<path to the directory to store model files>"
python3 ~/models/official/legacy/detection/main.py \
  --strategy_type=one_device \
  --num_gpus=1 \
  --model_dir=${MODEL_DIR} \
  --mode=train \
  --model=shapemask \
  --config_file="my_shapemask.yaml"
```

An example with inline configuration (YAML or JSON format):

```
python3 ~/models/official/legacy/detection/main.py \
  --model_dir=<model folder> \
  --strategy_type=one_device \
  --num_gpus=1 \
  --mode=train \
  --model=shapemask \
  --params_override="eval:
eval_file_pattern: <Eval TFRecord file pattern>
batch_size: 8
val_json_file: <COCO format groundtruth JSON file>
train:
total_steps: 1000
batch_size: 8
train_file_pattern: <Eval TFRecord file pattern>
use_tpu: False
"
```

Run the evaluation (after training)

```
python3 /usr/share/models/official/legacy/detection/main.py \
  --strategy_type=tpu \
```

```
--tpu=${TPU_NAME} \  
--model_dir=${MODEL_DIR} \  
--mode=eval \  
--model=shapemask \  
--params_override="{eval: { val_json_file: ${VAL_JSON_FILE}, eval_file_pattern:  
${EVAL_FILE_PATTERN}, eval_samples: 5000 } }"
```

`MODEL_DIR` needs to point to the trained path of ShapeMask model. Change `strategy_type=mirrored` and `num_gpus=1` to run on a GPU.

Note: The JSON groundtruth file is useful for [COCO dataset](#) and can be downloaded from the [COCO website](#). For custom dataset, it is unnecessary because the groundtruth can be included in the TFRecord files.

References

1. [Focal Loss for Dense Object Detection](#). Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. IEEE International Conference on Computer Vision (ICCV), 2017.