# 操作说明

说明：本文写作时，TF2 OD API预训练模型的转换在最新版本OpenVINO 2021.1中还没有被直接支持，主要原因是TF2模型的图结构有了较大变化。本文以TF2 OD API中最简单的 SSD MobileNet v2为例，演示在OpenVINO已经对TF1 OD API支持的基础上，如何经过简单的适配完成TF2 OD API模型的支持。同理，对于其它更复杂的TF2 OD API模型，也可以通过相应的过程尝试适配。

**第一步：**参考最新OpenVINO官方文档，完成并理解TensorFlow Object Detection API SSD模型的转换

从OpenVINO Open Model Zoo或TF1 OD Model Zoo下载ssd_mobilenet_v2_coco_2018_03_29并解压到试验目录（比如：/tmp），然后运行如下MO转换脚本：

```
<INSTALL_DIR>/deployment_tools/model_optimizer/mo_tf.py \
--input_model=/tmp/ssd_mobilenet_v2_coco_2018_03_29/frozen_inference_graph.pb \
--transformations_config \

<INSTALL_DIR>/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_support.json
--tensorflow_object_detection_api_pipeline_config \
/tmp/ssd_mobilenet_v2_coco_2018_03_29/pipeline.config --reverse_input_channels \
--tensorboard_logdir logs
```

**第二步：**初步转换TF2 OD API中对应的SSD模型

从TF2 OD Model Zoo中下载ssd_mobilenet_v2_320x320_coco17_tpu-8并解压到试验目录（比如：/tmp），然后运行如下MO转换脚本：

```
<INSTALL_DIR>/deployment_tools/model_optimizer/mo_tf.py \
--saved_model_dir=/tmp/ssd_mobilenet_v2_320x320_coco17_tpu-8/saved_model \
--input_shape [1,300,300,3] \
--transformations_config \
<INSTALL_DIR>/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_suppo
rt.json
--tensorflow_object_detection_api_pipeline_config \
/tmp/ssd_mobilenet_v2_320x320_coco17_tpu-8/pipeline.config
```

```
--reverse_input_channels \
--tensorboard_logdir logs
```

（pipeline.config中模型在训练前调用了300x300的image_resizer，因此此处转换时给出的shape是300x300）

默认情况下会出现如下错误信息：

```
[ ERROR ] Shape is not defined for output 1 of
"StatefulPartitionedCall/Postprocessor/BatchMultiClassNonMaxSuppression/MultiClassNonMaxSuppression/non_max_suppression_with_scores_78/NonMaxSuppressionV5".
[ ERROR ] Cannot infer shapes or values for node
"StatefulPartitionedCall/Postprocessor/BatchMultiClassNonMaxSuppression/MultiClassNonMaxSuppression/non_max_suppression_with_scores_78/NonMaxSuppressionV5".
[ ERROR ] Not all output shapes were inferred or fully defined for node
"StatefulPartitionedCall/Postprocessor/BatchMultiClassNonMaxSuppression/MultiClassNonMaxSuppression/non_max_suppression_with_scores_78/NonMaxSuppressionV5".
```

错误分析：

联系OpenVINO对TF1 OD API的支持过程，SSD模型的Postprocessor子图是被OpenVINO 自定义的DetectionOutput算子代替了。而ssd_v2_support.json中的子图替换配置信息跟TF2 OD SSD的子图信息不对应，所以转换过程出现了如上错误。

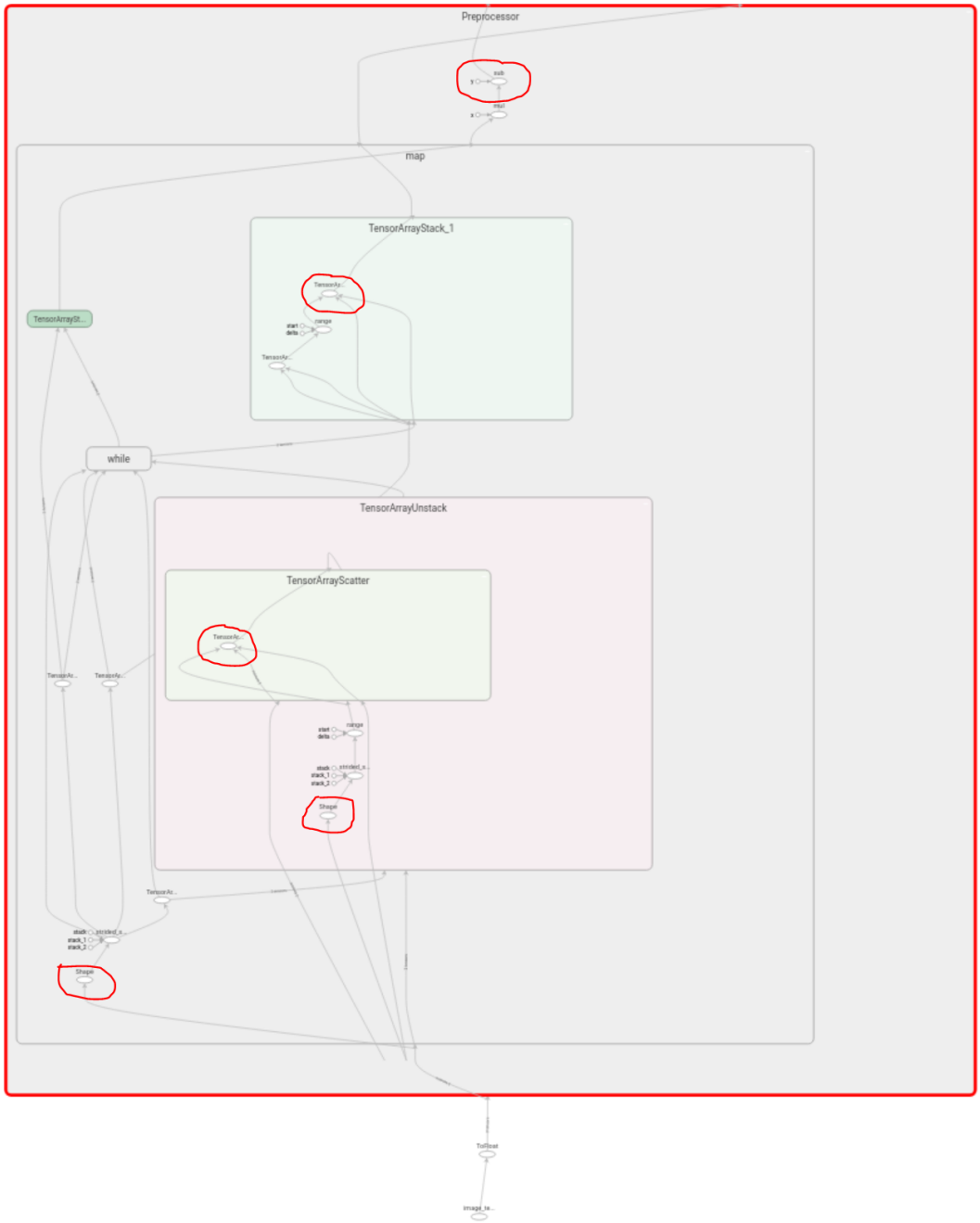
**第三步：**适配SSD子图替换

在TensorBoard中打开第一步生成的事件文件，对照ssd_v2_support.json文件中的ObjectDetectionAPIPreprocessorReplacement子图替换信息，主要是inputs和outputs，定位TensorBoard中Preprocessor子图的相应结点，结合MO源代码<INSTALL_DIR>/deployment_tools/model_optimizer/extensions/front/tf/ObjectDetectionAPI.py理解新的子图的生成过程。

ssd_v2_support.json文件ObjectDetectionAPIPreprocessorReplacement配置：

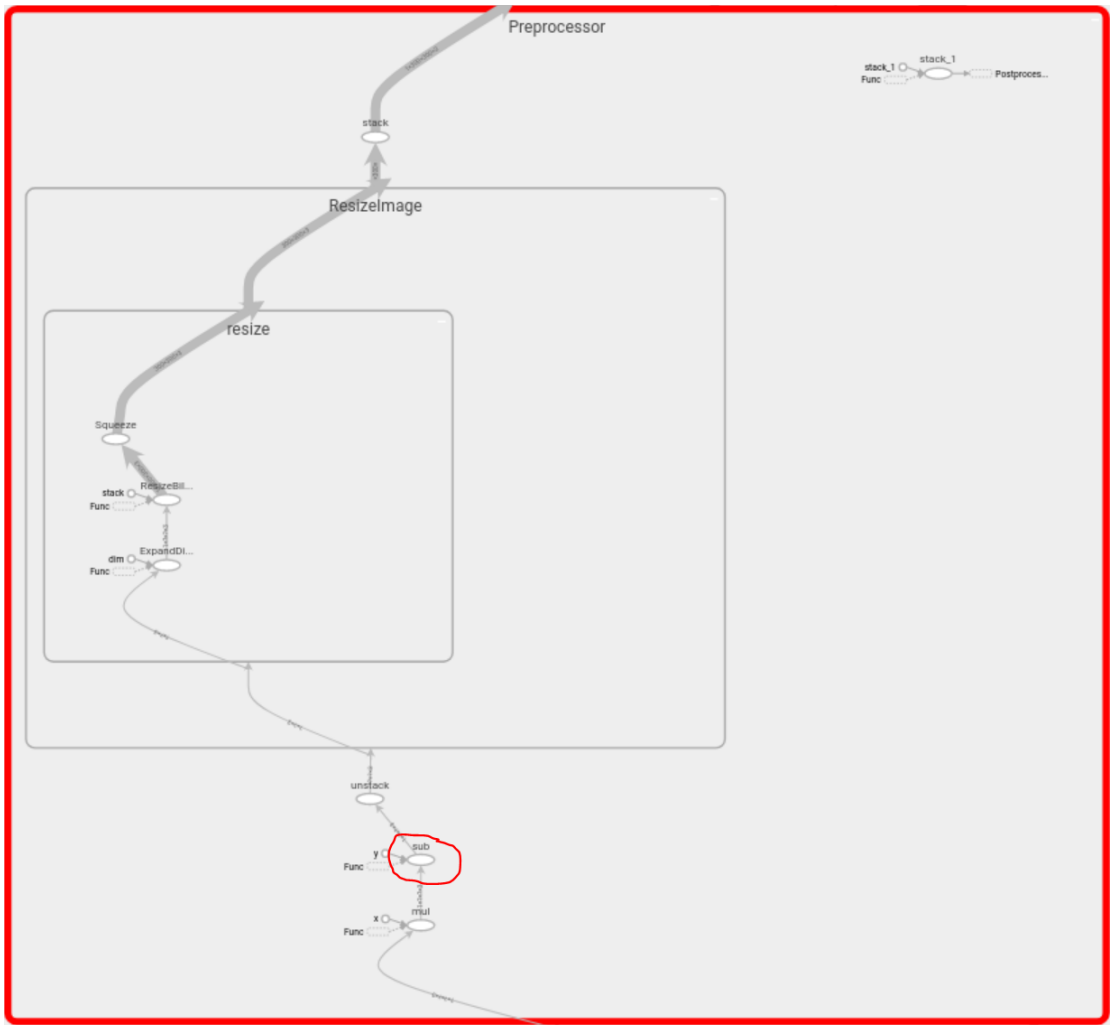
```
{
    "custom_attributes":{
    },
    "id":"ObjectDetectionAPIPreprocessorReplacement",
    "inputs":[
        [
            {
                "node":"map/Shape$",
                "port":0
            },
            {
                "node":"map/TensorArrayUnstack/Shape$",
                "port":0
            },
            {·
                "node":"map/TensorArrayUnstack/TensorArrayScatter/TensorArrayScatterV3$",
                "port":2
            }
        ]
    ],
    "instances":[
        ".*Preprocessor/"
    ],
    "match_kind":"scope",
    "outputs":[
        {
            "node":"sub$",
            "port":0
        },
        {
            "node":"map/TensorArrayStack_1/TensorArrayGatherV3$",
            "port":0
        }
    ]
}
```

TF1 SSD Preprocessor子图的TensorBoard视图：




然后，在 TensorBoard 中打开第二步生成的事件文件，同样定位到 Preprocessor 子图，结合上述生成新子图的过程，复制 ssd_v2_support.json 到 ssd_v2_support_tf2.json 并更新子图替换配置信息，同时根据需要修改 ObjectDetectionAPI.py 中对应的代码（见备注）。

TF2 SSD Preprocessor 子图的 TensorBoard 视图：




sd_v2_support_tf2.json 文件 ObjectDetectionAPIPreprocessorTF2Replacement 配置：

```
{
    "custom_attributes":{
    },
    "id":"ObjectDetectionAPIPreprocessorTF2Replacement",
    "instances":[
        ".*StatefulPartitionedCall/Preprocessor/"
    ],
    "match_kind":"scope",
    "inputs":[
    ],
    "outputs":[
        {
            "node":"sub$",
            "port":0
        }
    ]
}
```

文件 ssd_v2_support_tf2.json 中关于 Preprocessor 子图替换的 inputs 和 outputs 说明：

从 TF2 模型的 TensorBoard 视图可看出，resize 操作被安排在 sub 操作后面了，因此在进行 MO 转换的时候，只需要将 sub 结点的输出作为 Preprocessor 子图的输出即可，而且 inputs 可以保持不变。

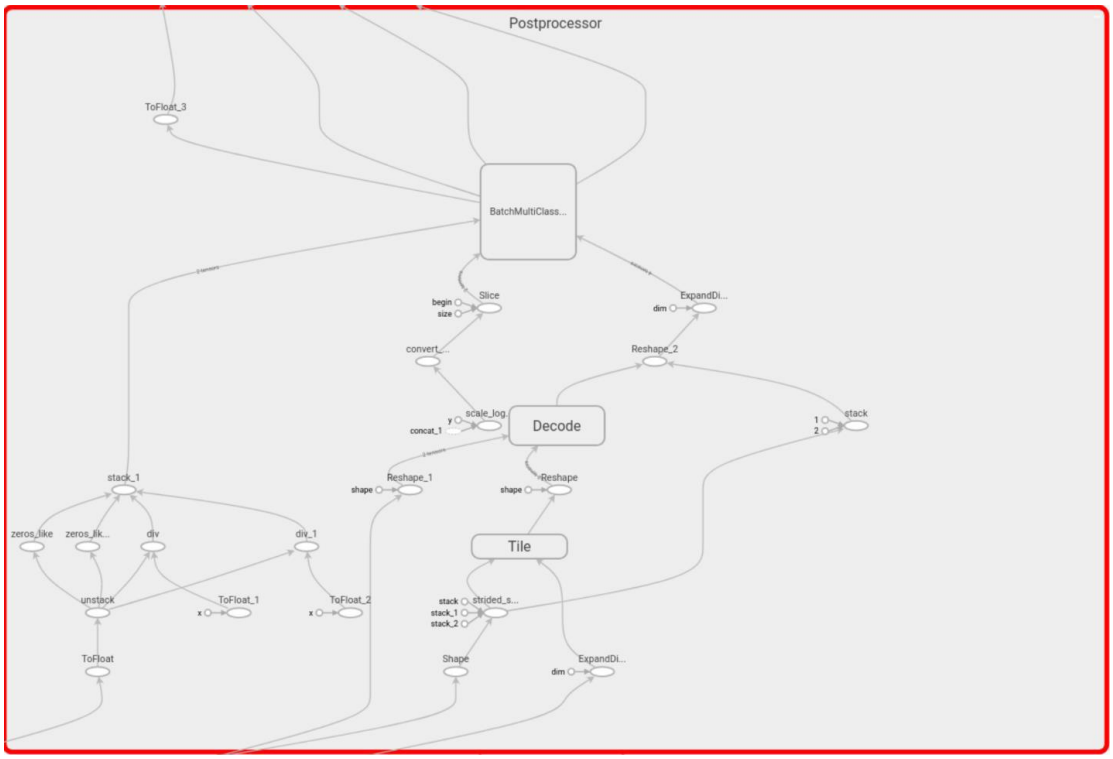
接下来，继续适配 PostProcessor 子图的替换。

对照 ssd_v2_support.json 文件中的 ObjectDetectionAPISSDPostprocessorReplacement 子图替换信息，在 TF1 模型的 TensorBoard 视图中，定位 Postprocessor 子图的相应结点，结合 MO 源代码
<INSTALL_DIR>/deployment_tools/model_optimizer/extensions/front/tf/ObjectDetectionAPI.py 理解新的子图的生成过程。

ssd_v2_support.json 文件 ObjectDetectionAPISSDPostprocessorReplacement 配置：

```
{
    "custom_attributes":{
        "code_type":"caffe.PriorBoxParameter.CENTER_SIZE",
        "pad_mode":"caffe.ResizeParameter.CONSTANT",
        "resize_mode":"caffe.ResizeParameter.WARP",
        "clip_before_nms":false,
        "clip_after_nms":true
    },
    "id":"ObjectDetectionAPISSDPostprocessorReplacement",
    "include_inputs_to_sub_graph":true,
    "include_outputs_to_sub_graph":true,
    "instances":{
        "end_points":[
            "detection_boxes",
            "detection_scores",
            "num_detections"
        ],
        "start_points":[
            "Postprocessor/Shape",
            "Postprocessor/scale_logits",
            "Postprocessor/Tile",
            "Postprocessor/Reshape_1",
            "Postprocessor/ToFloat"
        ]
    },
    "match_kind":"points"
}
```

TF1 SSD Postprocessor 子图的 TensorBoard 视图：




在 ObjectDetectionAPISSDPreprocessorReplacement 的配置中，可以看到原子图的搜索过程使用的是“Points”，分别需要指定 start points 和 end points。再结合 ObjectDetectionAPI.py 中对应的子图生成代码和 SSD 模型的 DetectionOutput 操作过程，不难分析出，Postprocessor 子图的 start points 就是 locations 和 confidences，而 prior boxes 对应结点从代码中可看出是在新子图生成过程中创建的，因此可以不考虑；相应地，end points 就是 Postprocessor 子图的输出。

根据以上分析，在 TF2 模型的 TensorBoard 视图中，定位 Postprocessor 子图的相应结点，并更新 ssd_v2_support_tf2.json。因 Postprocessor 子图替换是整个子图都换成 OpenVINO 提供的定制操作，所以 ObjectDetectionAPI.py 中对应的代码并不需要做修改，除了一些结点的名称变化（见备注）。

TF2 SSD Postprocessor 子图的 TensorBoard 视图：


```
{
    "custom_attributes":{
        "code_type":"caffe.PriorBoxParameter.CENTER_SIZE",
        "pad_mode":"caffe.ResizeParameter.CONSTANT",
        "resize_mode":"caffe.ResizeParameter.WARP",
        "clip_before_nms":false,
        "clip_after_nms":true
    },
    "id":"ObjectDetectionAPISSDPostprocessorTF2Replacement",
    "include_inputs_to_sub_graph":true,
    "include_outputs_to_sub_graph":true,
    "instances":{
        "end_points":[
            "Identity",
            "Identity_1",
            "Identity_2",
            "Identity_3",
            "Identity_4",
            "Identity_5",
            "Identity_6",
            "Identity_7"
        ],
        "start_points":[
            "StatefulPartitionedCall/Postprocessor/raw_box_encodings",
            "StatefulPartitionedCall/Postprocessor/scale_logits"
        ]
    },
    "match_kind":"points"
}
```

**第四步：**成功转换 TF2 OD API 中对应的 SSD 模型

添加 ObjectDetectionAPI.py 代码更新（见备注），并运行以下命令：

```
<INSTALL_DIR>/deployment_tools/model_optimizer/mo_tf.py \
--saved_model_dir=/tmp/ssd_mobilenet_v2_320x320_coco17_tpu-8/saved_model \
--input_shape [1,300,300,3] \
--transformations_config \
<INSTALL_DIR>/deployment_tools/model_optimizer/extensions/front/tf/ssd_v2_support_tf2.json \
--tensorflow_object_detection_api_pipeline_config \
/tmp/ssd_mobilenet_v2_320x320_coco17_tpu-8/pipeline.config --reverse_input_channels
```

备注：针对 ObjectDetectionAPI.py 的代码修改将放在另一个 patch 文件中。