Tensorflow 模型转换为 pb

1. 配置环境，安装 ubuntu16.04.3 系统，然后安装 ddk、驱动等；
2. Root 下：

export　LD_LIBRARY_PATH="/home/huawei/ddk/ddk/uihost/lib"

export DDK_HOME="/home/huawei/ddk/ddk"

3. 将训练好的 tensorflow 模型 ckpt 文件转换为字典的形式，即为 key-value 键值对；这里需要注意，转换权重时，atlas 只支持四维度，中间任意一个环节出现 5 维度的张量，后续 omg 时均出错；

```
import argparse
import tensorflow as tf
from core.yolov3 import YOLOV3
from core.config import cfg
parser = argparse.ArgumentParser()
parser.add_argument("--train_from_coco", action='store_true')
flag = parser.parse_args()

org_weights_path = cfg.YOLO.ORIGINAL_WEIGHT
cur_weights_path = cfg.YOLO.DEMO_WEIGHT
preserve_cur_names = ['conv_sbbox', 'conv_mbbox', 'conv_lbbox']
preserve_org_names = ['Conv_6', 'Conv_14', 'Conv_22']

org_weights_mess = []
tf.Graph().as_default()
load = tf.train.import_meta_graph(org_weights_path + '.meta')
with tf.Session() as sess:
    load.restore(sess, org_weights_path)
    for var in tf.global_variables():
        var_name = var.op.name
        var_name_mess = str(var_name).split('/')
        var_shape = var.shape
        if flag.train_from_coco:
            if (var_name_mess[-1] not in ['weights', 'gamma', 'beta', 'moving_mean', 'moving_variance']) or \
                    (var_name_mess[1] == 'yolo-v3' and (var_name_mess[-2] in preserve_org_names)): continue
        org_weights_mess.append([var_name, var_shape])
        #print(var_name,var_shape)
        #print("=> " + str(var_name).ljust(50), var_shape)
print()
tf.reset_default_graph()
```

```
cur_weights_mess = []
tf.Graph().as_default()
with tf.name_scope('input'):
    input_data = tf.placeholder(dtype=tf.float32, shape=(1, 416, 416, 3), name='input_data')
    training = tf.placeholder(dtype=tf.bool, name='trainable')
model = YOLOV3(input_data, training)
for var in tf.global_variables():
    var_name = var.op.name
    var_name_mess = str(var_name).split('/')
    var_shape = var.shape
    #print(var_name_mess[0])
    if flag.train_from_coco:
        if var_name_mess[0] in preserve_cur_names:
            print("==========",var_shape)
            continue
    cur_weights_mess.append([var_name, var_shape])
    #print("=> " + str(var_name).ljust(50), var_shape)

org_weights_num = len(org_weights_mess)
cur_weights_num = len(cur_weights_mess)
if cur_weights_num != org_weights_num:
    raise RuntimeError

print('=> Number of weights that will rename:\t%d' % cur_weights_num)
cur_to_org_dict = {}
for index in range(org_weights_num):
    org_name, org_shape = org_weights_mess[index]
    cur_name, cur_shape = cur_weights_mess[index]
    if cur_shape != org_shape:
        #print(org_weights_mess[index])
        #print(cur_weights_mess[index])
        raise RuntimeError
    cur_to_org_dict[cur_name] = org_name
```

```
with tf.name_scope('load_save'):
    name_to_var_dict = {var.op.name: var for var in tf.global_variables()}
    restore_dict = {cur_to_org_dict[cur_name]: name_to_var_dict[cur_name] for cur_name in cur_to_org_dict}
    load = tf.train.Saver(restore_dict)
    save = tf.train.Saver(tf.global_variables())
    #for var in tf.global_variables():
    #    print(var.op.name,var.shpe)
    #for var in tf.global_variables():
    #    print("=> " + var.op.name)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print('=> Restoring weights from:\t %s' % org_weights_path)
    load.restore(sess, org_weights_path)
    save.save(sess, cur_weights_path)
tf.reset_default_graph()
```

4. 对步骤 3 中转换权重后的 ckpt 文件进行 freeze 转换为 pb 文件；
   对应到模型中每一步图，即为 scope

```
with tf.variable_scope('pred_sbbox'):
    self.pred_sbbox = self.decode(self.conv_sbbox, self.anchors[0], self.strides[0])

with tf.variable_scope('pred_mbbox'):
    self.pred_mbbox = self.decode(self.conv_mbbox, self.anchors[1], self.strides[1])

with tf.variable_scope('pred_lbbox'):
    self.pred_lbbox = self.decode(self.conv_lbbox, self.anchors[2], self.strides[2])
```

   a. 当加上模型后处理的结果，模型文件中 decode 部分，freeze 部分如下所示：

```
def decode(self, conv_output, anchors, stride):
    """
    return tensor of shape [batch_size, output_size, output_size, anchor_per_scale, 5 + num_classes]
           contains (x, y, w, h, score, probability)
    """

    conv_shape       = tf.shape(conv_output)
    batch_size       = conv_shape[0]
    output_size      = conv_shape[1]
    anchor_per_scale = len(anchors)

    conv_output = tf.reshape(conv_output, (batch_size, -1, anchor_per_scale, 5 + self.num_class))
    #return conv_output
    #"""
    conv_raw_dxdy = conv_output[:, :, :, 0:2]
    conv_raw_dwdh = conv_output[:, :, :, 2:4]
    conv_raw_conf = conv_output[:, :, :, 4:5]
    conv_raw_prob = conv_output[:, :, :, 5: ]

    y = tf.tile(tf.range(output_size, dtype=tf.int32)[:, tf.newaxis], [1, output_size])
    x = tf.tile(tf.range(output_size, dtype=tf.int32)[tf.newaxis, :], [output_size, 1])

    xy_grid = tf.concat([x[:, :, tf.newaxis], y[:, :, tf.newaxis]], axis=-1)
    print(xy_grid.shape)
    xy_grid = tf.tile(xy_grid[tf.newaxis,:,  :, tf.newaxis, :], [batch_size,1, 1, anchor_per_scale, 1])
    xy_grid = tf.cast(xy_grid, tf.float32).
    print(xy_grid.shape)
    pred_xy = (tf.sigmoid(conv_raw_dxdy) + xy_grid) * stride
    pred_wh = (tf.exp(conv_raw_dwdh) * anchors) * stride
    pred_xywh = tf.concat([pred_xy, pred_wh], axis=-1)
    print("===shape===",pred_xywh.shape,anchor_per_scale,conv_raw_conf.shape,conv_raw_prob.shape)
    pred_conf = tf.sigmoid(conv_raw_conf)
    pred_prob = tf.sigmoid(conv_raw_prob)

    return tf.concat([pred_xywh, pred_conf, pred_prob], axis=-1)"""
```

   每个子图 scope，如果有多个相同的算子操作，则编号从 0 开始，否则没有编号

```
import tensorflow as tf
from core.yolov3 import YOLOV3

pb_file = "./yolov3_coco.pb"
ckpt_file = "./checkpoint/yolov3_coco_demo.ckpt"
output_node_names = ["input/input_data", "pred_sbbox/Reshape", "pred_mbbox/Reshape", "pred_lbbox/Reshape"]
#output_node_name=["input/input_data", "pred_sbbox/concat_2", "pred_mbbox/concat_2", "pred_lbbox/concat_2"]
with tf.name_scope('input'):
    input_data = tf.placeholder(dtype=tf.float32,shape=(None,416,416,3), name='input_data')

model = YOLOV3(input_data, trainable=False)
print(model.conv_sbbox, model.conv_mbbox, model.conv_lbbox)

sess  = tf.Session(config=tf.ConfigProto(allow_soft_placement=True))
saver = tf.train.Saver()
saver.restore(sess, ckpt_file)

converted_graph_def = tf.graph_util.convert_variables_to_constants(sess,
                          input_graph_def  = sess.graph.as_graph_def(),
                          output_node_names = output_node_names)

with tf.gfile.GFile(pb_file, "wb") as f:
    f.write(converted_graph_def.SerializeToString())
```

b. 当没有加上预测后处理的结果，模型文件中的 decode 部分和 freeze 部分为如下所示：

```
decode(self, conv_output, anchors, stride):
"""
return tensor of shape [batch_size, output_size, output_size, anchor_per_scale, 5 + num_classes]
       contains (x, y, w, h, score, probability)
"""

conv_shape       = tf.shape(conv_output)
batch_size       = conv_shape[0]
output_size      = conv_shape[1]
anchor_per_scale = len(anchors)

conv_output = tf.reshape(conv_output, (batch_size, -1, anchor_per_scale, 5 + self.num_class))
return conv_output
```

Freeze：

```
import tensorflow as tf
from core.yolov3 import YOLOV3

pb_file = "./yolov3_coco.pb"
ckpt_file = "./checkpoint/yolov3_coco_demo.ckpt"
output_node_names = ["input/input_data", "pred_sbbox/Reshape", "pred_mbbox/Reshape", "pred_lbbox/Reshape"]
#output_node_name=["input/input_data", "pred_sbbox/concat_2", "pred_mbbox/concat_2", "pred_lbbox/concat_2"]
with tf.name_scope('input'):
    input_data = tf.placeholder(dtype=tf.float32,shape=(None,416,416,3), name='input_data')

model = YOLOV3(input_data, trainable=False)
print(model.conv_sbbox, model.conv_mbbox, model.conv_lbbox)

sess  = tf.Session(config=tf.ConfigProto(allow_soft_placement=True))
saver = tf.train.Saver()
saver.restore(sess, ckpt_file)

converted_graph_def = tf.graph_util.convert_variables_to_constants(sess,
                          input_graph_def  = sess.graph.as_graph_def(),
                          output_node_names = output_node_names)

with tf.gfile.GFile(pb_file, "wb") as f:
    f.write(converted_graph_def.SerializeToString())
```

5. Omg 指令，将 pb 文件转换为 atlas 加速卡支持的 om 模型文件；

先进入 omg 执行文件夹：/home/huawei/ddk/ddk/uihost/bin

Root 下指令：

./omg --model /home/huawei/yolov3/tensorflow-yolov3/yolov3_coco.pb --framework 3 --output /home/huawei/yolov3/tensorflow_yolov3 --insert_op_conf /home/huawei/samples/Samples/InferObjectDetection/data/models/aipp_yolov3_picture.cfg --input_shape "input/input_data:1,416,416,3"