本节将TensorFlow模型导出为RKNN模型：

/\*

为于上一节的同一个目录下：



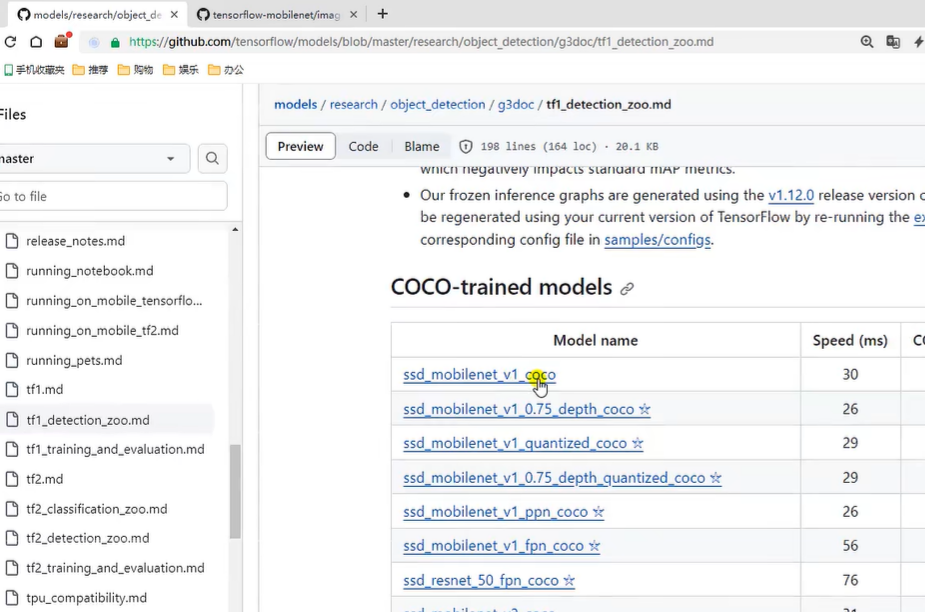
/\*

首先可以查看这个例子的预训练模型的信息：

预训练的模型参考的就是TensorFlow官网训练好的ssd\_mobilenet\_v1的模型，模型的下载地址就是:已经下载下来了：ssd\_mobilenet\_v1\_coco.

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md>

<http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_2018_01_28.tar.gz>



这个模型使用的数据集就是coco数据集。原来的coco数据集的论文版本有90类的，加上一个背景类就是91类。

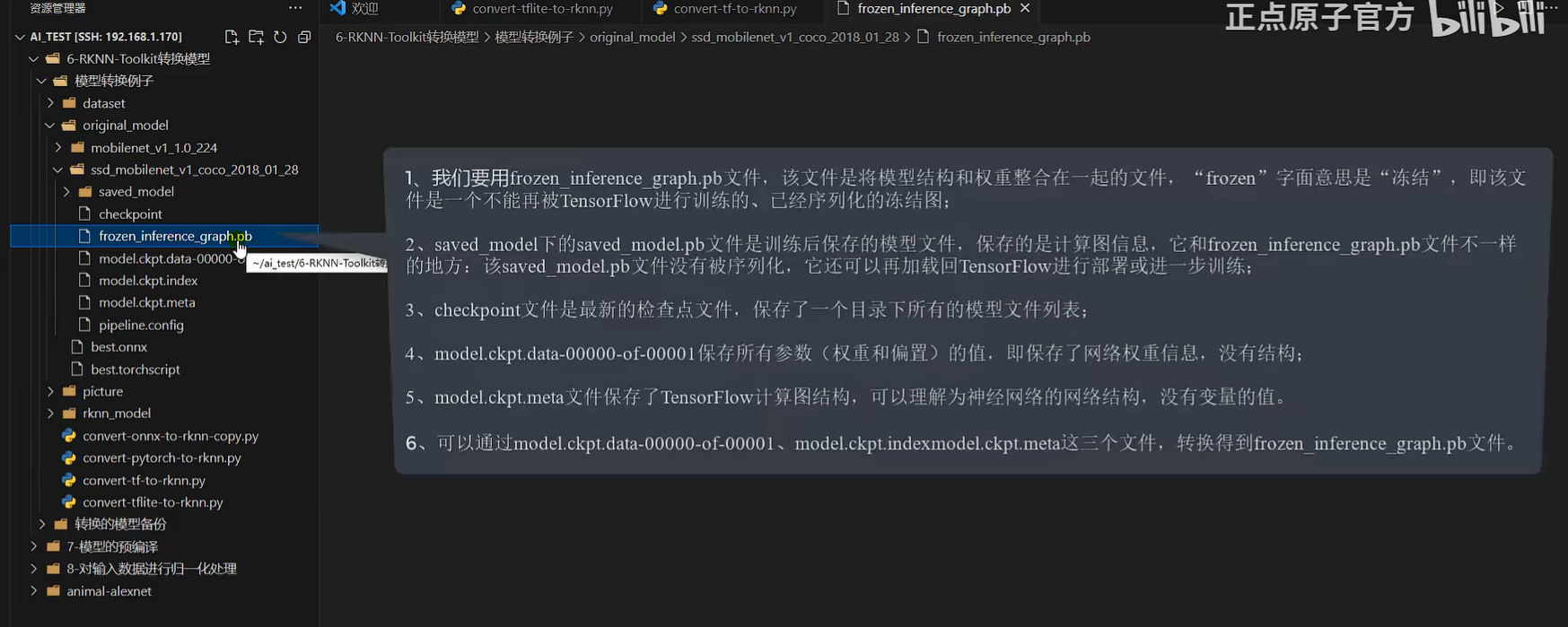
其标签查看：

<https://github.com/tensorflow/models/blob/master/research/object_detection/data/mscoco_label_map.pbtxt>

很多时候，在目标检测这块，只用到原论文版本90类中的80类，具体用到那些可见程序。

\*/

/\*



我们直接使用如上的pb文件，这个文件就是将模型结构和权重整合在一起的文件，这里frozen\_inference\_graph.pb字面意思就是冻结，也就是说这个文件是不能再被TensorFlow训练的已经训练化的冻结图。

saved\_model下的saved\_model.pb文件是训练后保存的文件。包含图形结构，保存的是计算图信息。它和frozen\_inference\_graph.pb是不一样的，saved\_model.pb没有被训练化，还可以再加载回TensorFlow进行部署和进一步的训练。一般我们使用的就是frozen\_inference\_graph.pb文件。

checkpoint是最新的检查点文件，保存了一个目录下所有模型文件的列表。

model.ckpt.data-00000-of-00001保存所有参数（权重和偏置）的值，即保存了网络权重信息，没有结构；

model.ckpt.meta文件保存了TensorFlow计算图结构，可以理解为神经网络的网络结构，没有变量的值。

可以通过model.ckpt.data-0000-of-00001、model.ckpt.index、model.ckpt.meta这三个文件，转换得到最终要使用的frozen\_inference\_graph.pb文件。

\*/

/\*

我们来查看模型转化的代码：

流程同上一节是类似的，不同之处就是加载模型和配置模型，其它地方基本是一样的。



通过手册我们知道，要确定模型输入的节点。然后确定模型的输出节点。以及每个输入节点对应的数据形状。

\*/

/\*

用netron来打开，可以看到输入节点是



/\*

按理说：

模型的输入节点应该为第一个节点：image\_tensor

模型的输出节点应该为最后一个节点：那4个Identity

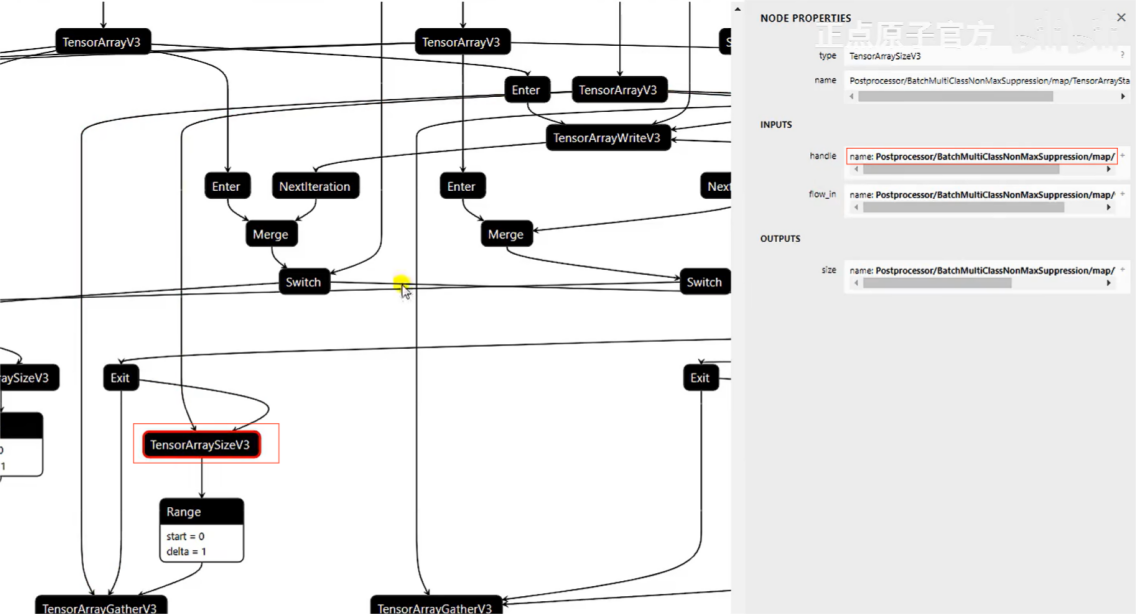
但代码里输入节点写的却是：Preprocessor/sub

输出节点写的是：'concat', 'concat\_1'这两个

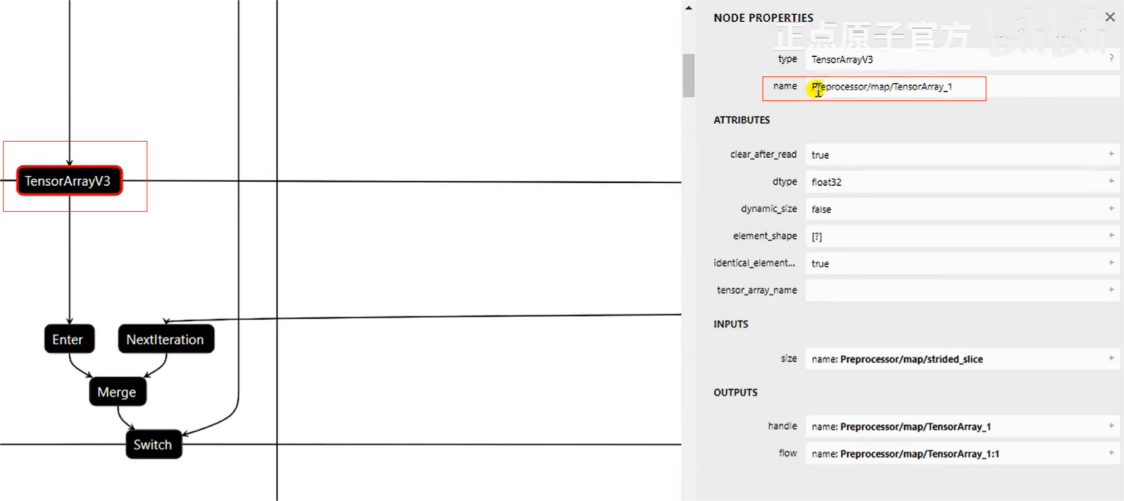
这是为什么呢？

一般情况下，输入节点可以认为是这个模型的第一个节点，输出节点是这个模型的最后一个节点，但是在RKNPU下，有时候不是这样的，因为RKNN框架和tensorFlow的框架是有差别的。

对于RKNPU来说，TensorFlow模型就是我们打开的pb文件，它的前处理和后处理部分不能直接用RKNPU来进行计算，这会影响到模型的推理速度。前处理和后处理部分，大概在RKNPU下就是丢给了CPU进行处理了。而模型中间的部分，才是大量的卷积运算，这部分才交给了RKNPU来进行运算。



图中的Postprocessor就属于后处理部分。



这个preprocessor就属于模型的前处理部分。

前处理部分是给CPU来进行处理的。而不是给RKNPU来进行计算的。

总结：

为什么在转换模型的时候，输入节点不是第一个节点，输出节点不是最后一个节点呢：

1、对于RKNPU来说，该TensorFLow模型（.pb文件）的前处理（preprocessor)和后处理（postprocessor）部分不能直接利用RKNPU来计算，直接利用RKNPU来计算会影响模型推理速度，所以前处理和后处理部分大概是丢给了CPU进行处理了，而模型中间部分大量的卷积运算则使用RKNPU进行运算。

2、此外，该TensorFLow模型有很多的op（operator，算子）在RKNNToolkit下是不支持的，若具有不支持算子的话，要么需要自定义算子来解决，要么，就是需要对模型进行裁剪，若不支持的op太多的话，自定义算子就很麻烦。所以，很多时候需要做裁剪，以上前处理和后处理部分相当于被裁剪掉了，若不进行裁剪，那么多TensorFLow的算子在RKNNToolkit下不支持，岂不是得自定义算子，使用起来更加麻烦。

Conv2D就是进行的卷积运算。

\*/

通过上面的分析，我们知道卷积部分有两个输出，分别是concat和concat\_1节点，输入的话就是中间部分，第一个卷积的节点名称。

那么最终调用这个api来加载pb文件的话，就是代码那样。卷积操作交给RKNPU进行推理。然后模型的前处理和后处理则交给CPU进行处理了。

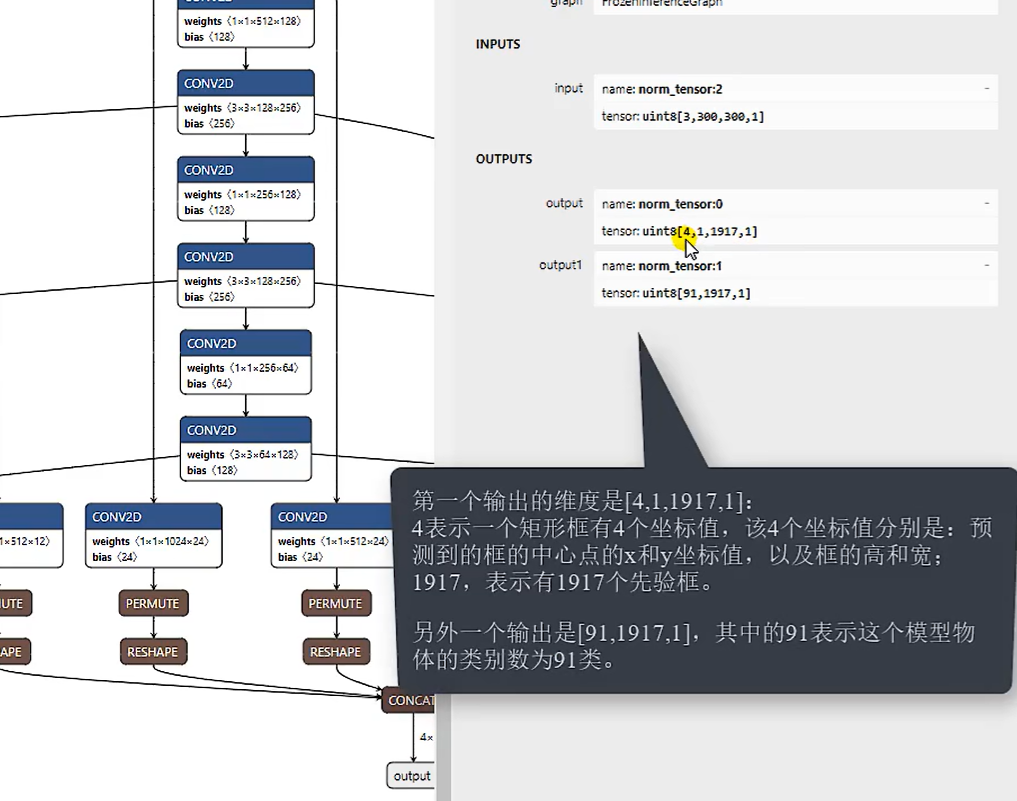
\*/

后续的流程就同上节课一样了。

/\*

转换得到模型文件:ssd\_mobilenet\_v1\_coco.rknn

这个文件在netron下打开这个文件可以看到：这个模型的输入输出信息了



这4个坐标值分别是：

预测得到的框的中心点的x和y坐标

以及框的高和宽

\*/

转换得到的RKNN模型，在后期我们可以通过调用RKNN Toolkit里的python api/rknpu里的c/c++的api来将模型放到开发板上进行推理。

\*/