



登临 HammingTM V2

Relay IR 子图划分操作指导

DL-DG/SW-056A-01

2024-05-30

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

更新历史

版本	更新描述
01	第一次发布

章节目录

章节目录

1 简介

2 操作指导

2.1 使用子图标注API加入子图划分信息

2.1.1 定义

2.1.2 接口描述

2.1.3 标注方法

2.2 配置子图划分文件

2.2.1 说明

2.2.2 配置方法

1 简介

本手册主要介绍在Relay IR中加入子图划分信息的操作方法。

其中涉及的概念简介如下：

- **Relay IR**：是Apache TVM项目开发的中间表示，用于表示模型、优化模型和高效部署模型。
- **子图划分**：在Relay IR上插入表示子图边界的表达式，用来确定子图的区域信息，划分出子图。

2 操作指导

有两种方式可以在Relay IR中加入子图划分信息：

1. 使用Relay API构造计算图时，可以直接使用子图标注API加入子图划分信息。
2. 从原始模型（ONNX）导入计算图时，可以指定一个子图标注配置文件。

2.1 使用子图标注API加入子图划分信息

2.1.1 定义

“用子图标注API加入子图划分信息”是指：

找到子图的输入或者输出expr，并通过提供的 `region_begin` 和 `region_end` 接口对输入和输出进行标注。由于显式标注了图的输入和输出，于是就确定了子图的区域信息。

2.1.2 接口描述

接口 `region_begin` 和 `region_end`，都位于 `dl/relay/op/transform.py` 中，如下图所示：

```
def region_begin(data, name):
    r"""region begin

    This operator indicate the begin of the Relay ir region of original frontend(ONNX, Caffe, TensorFlow ...) node
    (or call the subgraph in relay ir)

    Parameters
    -----
    data: relay.Expr
    name : String , the name of the subgraph

    Returns
    -----
    result : relay.Expr
    """
    return _make.region_begin(data, name)

def region_end(data, name):
    r"""region end

    This operator indicate the end of the Relay ir region of original frontend(ONNX, Caffe, TensorFlow ...) node
    (or call the subgraph in relay ir)

    Parameters
    -----
    data: relay.Expr
    name : String , the name of the subgraph

    Returns
    -----
    result : relay.Expr
    """
    return _make.region_end(data, name)
```

- 接口 `region_begin` 要求参数 `data` 是 `relay.Expr`，参数 `data` 表示子图的输入；要求参数 `name` 是 `string`，参数 `name` 表示这个子图的名字。
- 接口 `region_end` 要求参数 `data` 是 `relay.Expr`，参数 `data` 表示子图的输出；要求参数 `name` 是 `string`，参数 `name` 表示这个子图的名字。

说明：

1. 当子图的输入或者输出是 `relay.const` 的时候，一般不使用 `region_begin` 或者 `region_end` 来进行标注，详见 [2.1.3 标注方法](#)。
2. `region_begin` 和 `region_end`，要求参数 `data` 的 `checked_type` 是 `TensorType`。如果 `data` 是个 `relay.expr.Tuple`，或者 `data` 的 `checked_type` 是 `TupleType`，不应该直接使用 `data` 作为 `region_begin` 和 `region_end` 的参数，而是需要对 `data` 做特殊处理，详见 [2.1.3 标注方法](#)。

2.1.3 标注方法

假如当前Relay接口构建的网络如下图，并需要根据提供的 `region_begin` 和 `region_end` 接口，标注出subgraph "0"、subgraph "1"、subgraph "2"的region信息，操作方法如下：

```

data = relay.var("data", shape=(2,20,8,8),dtype="float32")
kernel_np = np.random.uniform(-1, 1, size=(6,20,3,3)).astype("float32")
kernel = relay.const(kernel_np, dtype="float32")
bias = relay.var("bias", shape=(6,),dtype="float32")

# begin subgraph "0"
conv2d = relay.nn.conv2d(data, kernel)
bias_add = relay.nn.bias_add(conv2d,bias)
# end subgraph "0"

# begin subgraph "1"
split = relay.split(bias_add, 2, axis=0)
# end subgraph "1"

part_0 = split[0]
part_1 = split[1]

# begin subgraph "2"
add = relay.op.add(part_0, part_1)
# end subgraph "2"

mod = tvm.IRModule.from_expr(add)

```

1. 首先标注subgraph "0"。

找到subgraph "0"的所有输入和输出。输入有data和kernel，输出有bias_add。于是应该使用两个 `region_begin` 来标注subgraph "0"的输入，使用一个 `region_end` 来标注subgraph "0"的输出。

但由于kernel是个 `relay.const`，没有必要标注 `relay.const` 的输入。所以使用一个 `region_begin` 来标注subgraph "0"的输入，使用一个 `region_end` 来标注subgraph "0"的输出，如下图所示：

```

data = relay.var("data", shape=(2,20,8,8),dtype="float32")
kernel_np = np.random.uniform(-1, 1, size=(6,20,3,3)).astype("float32")
kernel = relay.const(kernel_np, dtype="float32")
bias = relay.var("bias", shape=(6,),dtype="float32")

# begin subgraph "0"
sub_0_begin_0 = dl.relay.op.transform.region_begin(data, "0")
conv2d = relay.nn.conv2d(sub_0_begin_0, kernel)
bias_add = relay.nn.bias_add(conv2d,bias)
sub_0_end_0 = dl.relay.op.transform.region_end(bias_add, "0")
# end subgraph "0"

```

2. 然后标注subgraph "1"。

找到subgraph "1"的所有输入和输出，输入有sub_0_end_0，输出有split。于是应该使用一个 `region_begin` 来标注subgraph "1"的输入，使用一个 `region_end` 来标注subgraph "1"的输出。

由于拆分的 `checked_type` 是个 `TupleType`，而不是 `TensorType`，所以需要进行特殊处理。需要将拆分后的每个域 (field) 拿出来，每个field标注为 `region_end`，然后将这些 `region_end` 构建成一个 `Tuple`，如下图所示：


```

data = relay.var("data", shape=(2,20,8,8),dtype="float32")
kernel_np = np.random.uniform(-1, 1, size=(6,20,3,3)).astype("float32")
kernel = relay.const(kernel_np, dtype="float32")
bias = relay.var("bias", shape=(6,),dtype="float32")

# begin subgraph "0"
sub_0_begin_0 = dl.relay.op.transform.region_begin(data, "0")
conv2d = relay.nn.conv2d(sub_0_begin_0, kernel)
bias_add = relay.nn.bias_add(conv2d,bias)
sub_0_end_0 = dl.relay.op.transform.region_end(bias_add, "0")
# end subgraph "0"

# begin subgraph "1"
sub_1_begin_0 = dl.relay.op.transform.region_begin(sub_0_end_0, "1")
split = relay.split(sub_1_begin_0, 2, axis=0)
part_0 = split[0]
part_1 = split[1]
sub_1_end_0 = dl.relay.op.transform.region_end(part_0, "1")
sub_1_end_1 = dl.relay.op.transform.region_end(part_1, "1")
split_tuple = relay.expr.Tuple([sub_1_end_0, sub_1_end_1])
# end subgraph "1"

part_0 = split_tuple[0]
part_1 = split_tuple[1]

```

3. 最后标注subgraph "2"。

找到subgraph "2"的所有输入和输出，输入现在有split_tuple[0]和split_tuple[1]，输出有add。于是使用两个region_begin来标注subgraph "2"的输入，一个region_end来标注subgraph "2"的输出，如下图所示：


```

data = relay.var("data", shape=(2,20,8,8),dtype="float32")
kernel_np = np.random.uniform(-1, 1, size=(6,20,3,3)).astype("float32")
kernel = relay.const(kernel_np, dtype="float32")
bias = relay.var("bias", shape=(6,),dtype="float32")

# begin subgraph "0"
sub_0_begin_0 = dl.relay.op.transform.region_begin(data, "0")
conv2d = relay.nn.conv2d(sub_0_begin_0, kernel)
bias_add = relay.nn.bias_add(conv2d,bias)
sub_0_end_0 = dl.relay.op.transform.region_end(bias_add, "0")
# end subgraph "0"

# begin subgraph "1"
sub_1_begin_0 = dl.relay.op.transform.region_begin(sub_0_end_0, "1")
split = relay.split(sub_1_begin_0, 2, axis=0)
part_0 = split[0]
part_1 = split[1]
sub_1_end_0 = dl.relay.op.transform.region_end(part_0, "1")
sub_1_end_1 = dl.relay.op.transform.region_end(part_1, "1")
split_tuple = relay.expr.Tuple([sub_1_end_0, sub_1_end_1])
# end subgraph "1"

part_0 = split_tuple[0]
part_1 = split_tuple[1]

# begin subgraph "2"
sub_2_begin_0 = dl.relay.op.transform.region_begin(part_0, "2")
sub_2_begin_1 = dl.relay.op.transform.region_begin(part_1, "2")
add = relay.op.add(sub_2_begin_0, sub_2_begin_1)
sub_2_end_0 = dl.relay.op.transform.region_end(add, "2")
# end subgraph "2"

mod = tvm.IRModule.from_expr(sub_2_end_0)

```

4. 标注后的Relay IR 如下图所示：

```

dl/tests/python/test_mark_subgraph_region.py::test_create_subgraph_after def @main(%data: Tensor[(2, 20, 8, 8), float32] /* ty=Tensor[(2, 20, 8, 8), float32] */, %bias: Tensor[(6), float32] /* ty=Tensor[(6), float32] */) -> Tensor[(1, 6, 6), float32] {
  %0 = dl.region_begin(%data, subgraph_name="0") /* ty=Tensor[(2, 20, 8, 8), float32] */;
  %1 = nn.conv2d(%0, meta[relay.Constant][0] /* ty=Tensor[(6, 20, 3, 3), float32] */, padding=[0, 0, 0, 0]) /* ty=Tensor[(2, 6, 6, 6), float32] */;
  %2 = nn.bias_add(%1, %bias) /* ty=Tensor[(2, 6, 6, 6), float32] */;
  %3 = dl.region_end(%2, subgraph_name="0") /* ty=Tensor[(2, 6, 6, 6), float32] */;
  %4 = dl.region_begin(%3, subgraph_name="1") /* ty=Tensor[(2, 6, 6, 6), float32] */;
  %5 = split(%4, indices_or_sections=2) /* ty=Tensor[(1, 6, 6, 6), float32], Tensor[(1, 6, 6, 6), float32] */;
  %6 = %5.0 /* ty=Tensor[(1, 6, 6, 6), float32] */;
  %7 = dl.region_end(%6, subgraph_name="1") /* ty=Tensor[(1, 6, 6, 6), float32] */;
  %8 = %5.1 /* ty=Tensor[(1, 6, 6, 6), float32] */;
  %9 = dl.region_end(%8, subgraph_name="1") /* ty=Tensor[(1, 6, 6, 6), float32] */;
  %10 = dl.region_begin(%7, subgraph_name="2") /* ty=Tensor[(1, 6, 6, 6), float32] */;
  %11 = dl.region_begin(%9, subgraph_name="2") /* ty=Tensor[(1, 6, 6, 6), float32] */;
  %12 = add(%10, %11) /* ty=Tensor[(1, 6, 6, 6), float32] */;
  dl.region_end(%12, subgraph_name="2") /* ty=Tensor[(1, 6, 6, 6), float32] */
}

```

2.2 配置子图划分文件

2.2.1 说明

“配置子图划分文件”适用于TVM中添加的MarkSubgraphRegion优化。

该优化可直接在原始输入模型上，通过指定所划子图所包含的所有算子的相关信息（一般是原始模型中算子的name），完成原始模型上子图的划分。

配置文件的主要任务，就是收集原始模型中子图所包含的所有算子的相关信息（一般是原始模型中算子的name），同时还需要将这些收集的算子信息，对应到子图名上。

MarkSubgraphRegion根据该配置文件的内容，在Relay IR中找出对应子图所包含的全部算子，将这些算子的集合当做该子图的region。

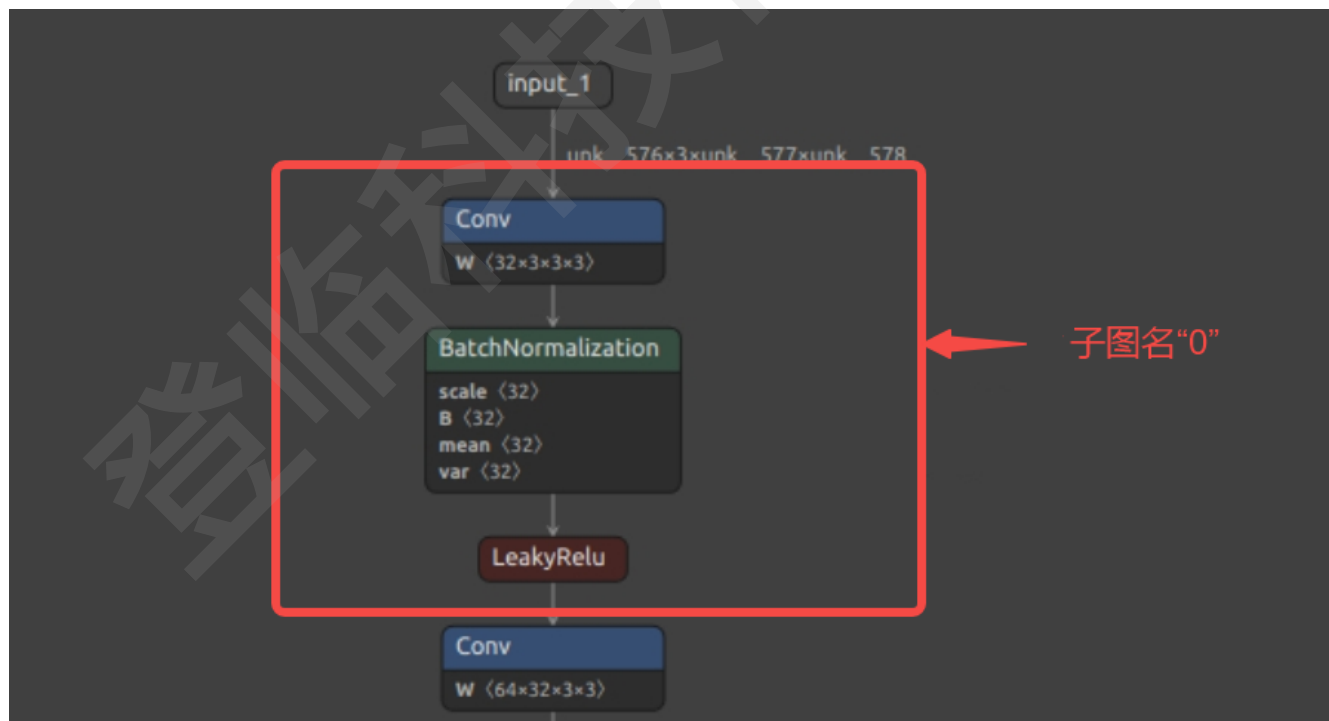
同时进行以下操作：

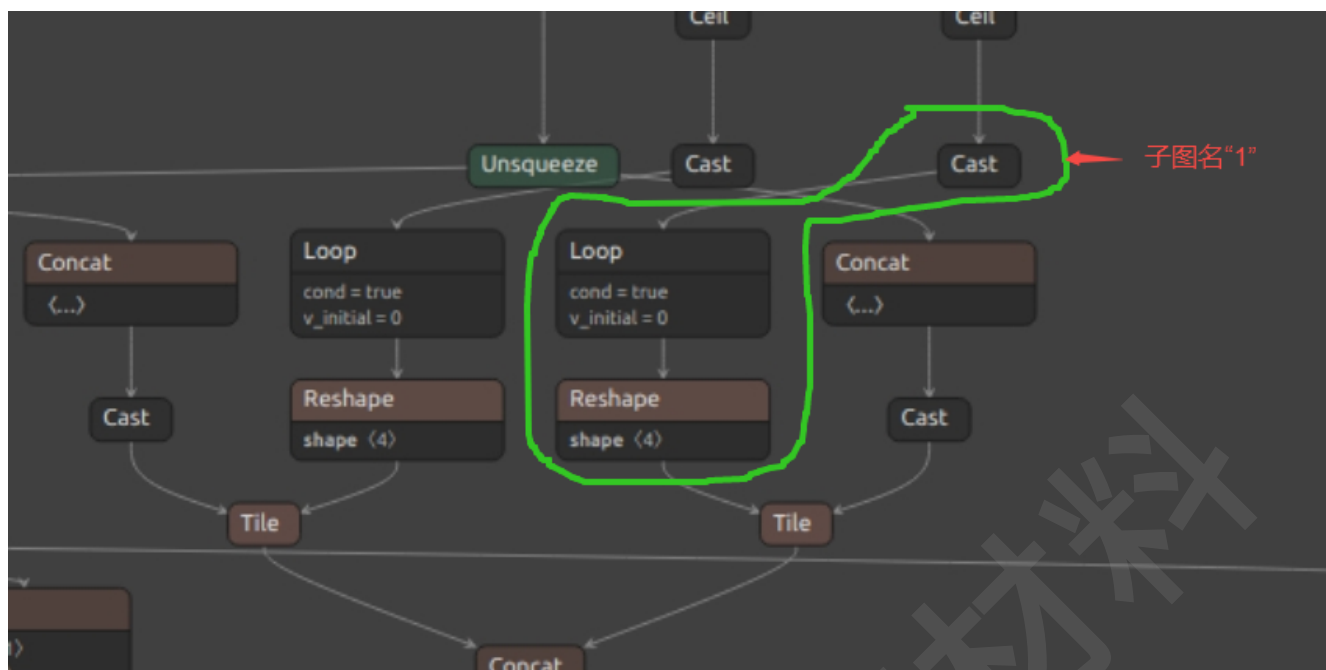
- 在region的每个输入（subgraph_input）后，插入 `d1.region_begin(subgraph_input, subgraph_name)` 来标志子图的输入边界。
- 在region的每个输出（subgraph_output）后，插入 `d1.region_end(subgraph_output, subgraph_name)` 来标志子图的输出边界。

2.2.2 配置方法

假如模型

是 `/mercury/share/DLI/onnx/models/vision/object_detection_segmentation/yolov3/model/yolov3-10.onnx`，使用Netron打开模型后，将该模型中的红圈部分作为子图“0”，将绿圈部分作为子图“1”，开始从原始模型中划分出这两个子图，如下图所示：

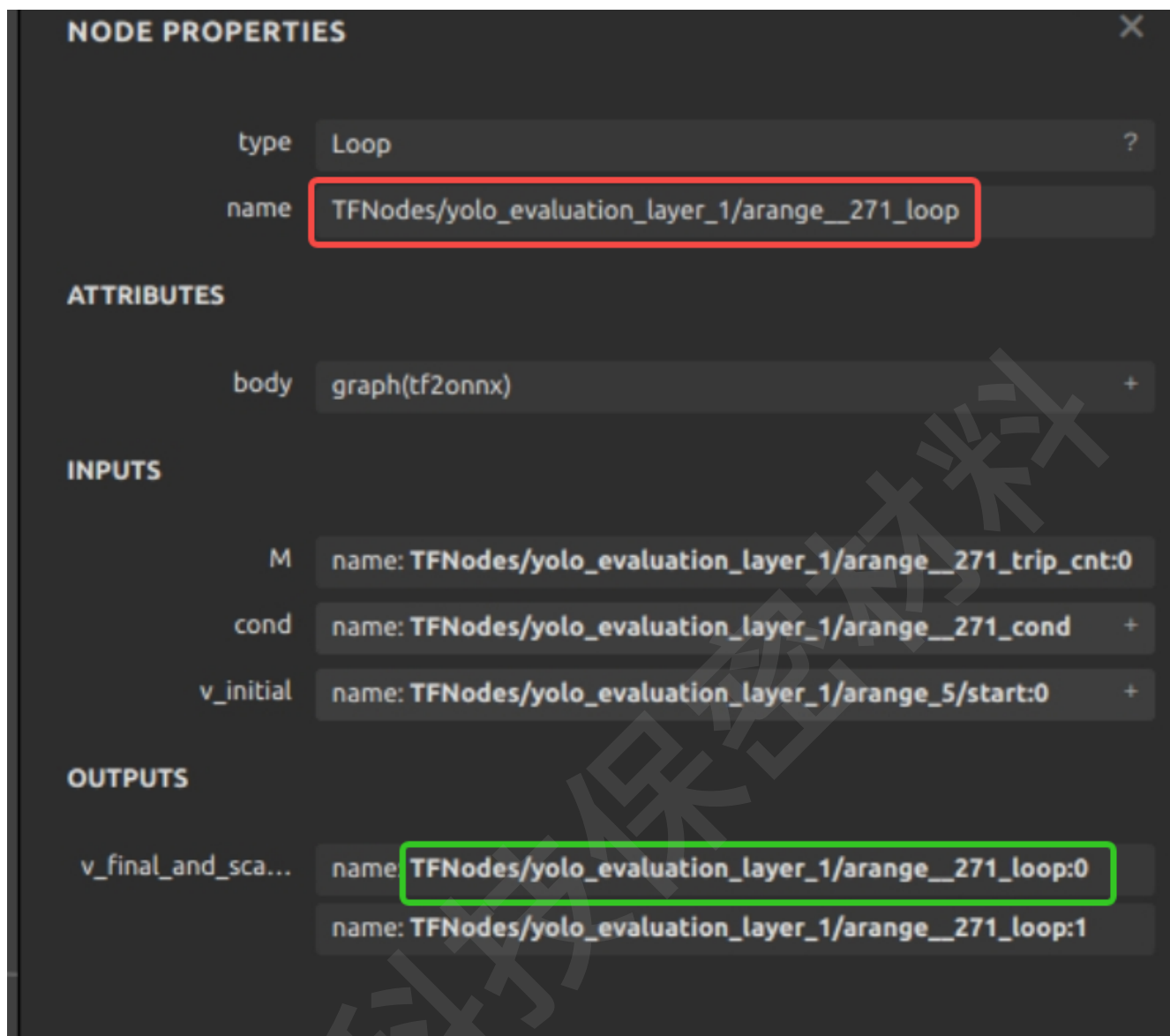




以子图“1”举例：

收集绿圈中每个算子的name（红色标志）作为该算子的唯一标识。如果这个算子没有name（红色标志），就取该算子OUTPUTS中的第一个name（绿色标志）作为该算子的唯一标识。

将收集的所有算子信息存入列表中。这里用子图“1”中的Loop算子来解释该过程，如下图：



所以子图“1”收集的信息是：

```
["TFNodes/yolo_evaluation_layer_1/arange__271_trip_cnt", "TFNodes/yolo_evaluation_layer_1/arange__271_loop", "TFNodes/yolo_evaluation_layer_1/Reshape_1"]
```

子图“0”收集的信息是：

```
["conv2d_1", "batch_normalization_1", "leaky_re_lu_1"]
```

将上述子图收集的信息，对应到子图的名字并存入一个.json文件中，该文件的内容如下：

```
{
  "0":["conv2d_1","batch_normalization_1","leaky_re_lu_1"],
  "1":
  ["TFNodes/yolo_evaluation_layer_1/arange__271_trip_cnt","TFNodes/yolo_evaluation_layer_1/arange__271_loop", "TFNodes/yolo_evaluation_layer_1/Reshape_1"]
}
```

配置完子图划分的.json文件后，可以通过以下命令实现子图的划分：

```
python3 -m dl convert yolov3-10.onnx --optimize --subgraph-config-file=config.json
```

说明：

添加 `--optimize` 参数才能开启MarkSubgraphRegion优化。