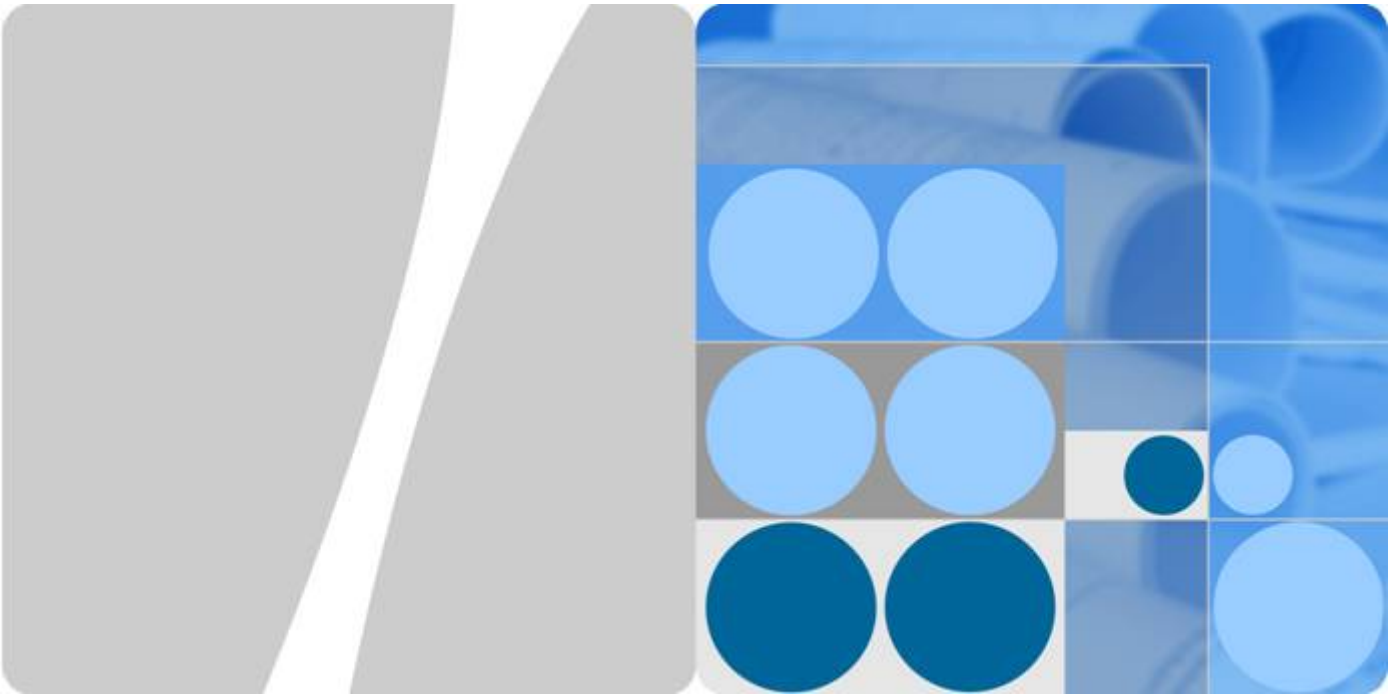


Ascend 310

V100R001

# Mind Studio 快速入门

文档版本　01

发布日期　2019-03-12

华为技术有限公司

HUAWEI

**商标声明**


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

**注意**

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

# 华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

# 目 录



# 1 简介

本文描述了什么是Mind Studio，介绍了Mind Studio的基本功能特性，并以一个开发样例介绍了如何应用Mind Studio的Engine编排功能快速构建自己的AI程序，以及Mind Studio的基本功能离线模型转化。

# 2 MindStudio 工具描述

## 2.1 简介

Mind Studio是一套基于华为NPU（Neural-network Processing Unit）开发的AI全栈开发平台，包括基于芯片的算子开发、以及自定义算子开发，同时还包括网络层的网络移植、优化和分析，另外在业务引擎层提供了一套可视化的AI引擎拖拽式编程服务，极大的降低了AI引擎的开发门槛，全平台通过Web的方式向开发者提供以下4项服务功能。

- 针对算子开发

  Mind Studio提供全套的算子开发、支持真实环境运行，支持针对动态调度的异构程序的可视化调试，支持第三方算子开发，极大的降低了基于华为自研NPU的算子开发门槛，提高算子开发效率，有效提升产品竞争力。

- 针对网络层的开发

  Mind Studio集成了离线模型转换工具（OMG）、模型量化工具、模型运行Profiling分析工具和日志分析工具，极大的提升了网络模型移植和分析优化的效率。

- 针对AI引擎开发

  Mind Studio提供了AI引擎可视化拖拽式编程以及大量的算法代码自动生成技术，极大的降低了开发者的门槛，并且预置了丰富的算法引擎，如：Resnet18等，大大提高了用户AI算法引擎开发及移植效率。

- 针对应用开发

  Mind Studio内部集成了各种工具如Profiler、Compiler等，为用户提供图形化的集成开发环境，通过Mind Studio进行工程管理、编译、调试、仿真、性能分析等全流程开发，从而提高开发效率。

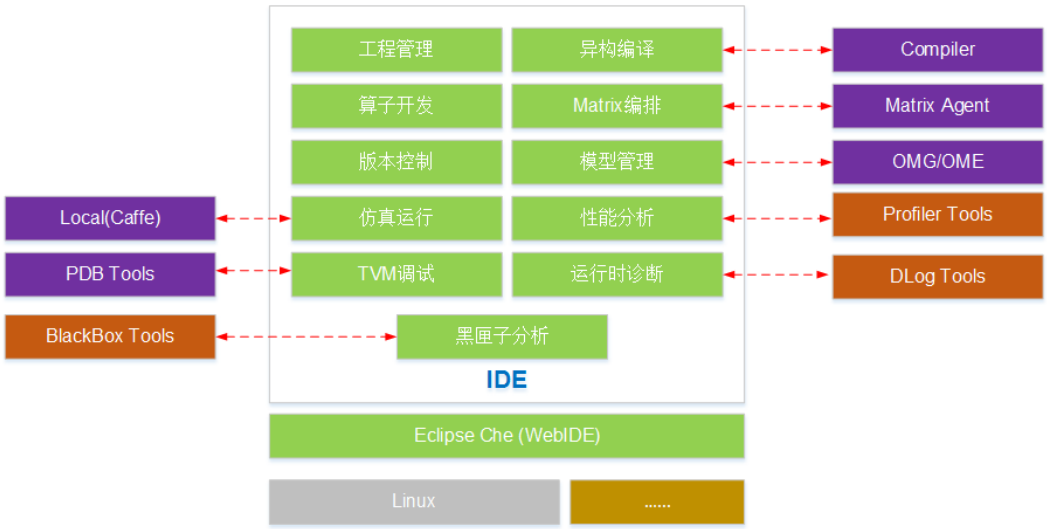
## 2.2 功能说明

### 整体架构

Mind Studio整体框架，如图2-1所示。

**图 2-1** Mind Studio 整体框架




## 功能说明

Mind Studio主要提供以下特性功能：

- **基于神经网络芯片（NPU）的友好编程界面**

  针对算子开发人员，Mind Studio基于CCE编程深度定制适配CCE开发，实现深度融合，扩展CCE语言关键字高亮显示、异构混合代码一键式编译等功能。

- **基于神经网络芯片（NPU）的图形化调试能力**

  针对NPU神经网络芯片上的算子加速库开发，Mind Studio提供图形用户接口，实现加速算子在AI Core/AI CPU上的运行状态实时跟踪。

- **离线模型全自动管理**

  训练好的第三方离线模型（Caffe、Caffe2（暂不支持）、Tensorflow、Mxnet（暂不支持））可以直接通过Mind Studio导入转换成本系统支持的模型，并可一键式自动生成模型接口，方便用户基于模型接口进行编程。

- **业务流程编排“0”编码**

  针对业务流程开发人员，Mind Studio提供基于业务节点的拖拽式编程方式，在Mind Studio上拖拽业务节点并连线，可实现业务编排“0”编码。编排后的编译、运行、结果显示等一站式服务让流程开发更加智能化，整个过程无需编程，完全是通过拖拽和配置完成，非常的简单。能让用户快速上手，从而把更多的精力放在理解业务上，工具本身并不会带来任何额外的学习成本。

- **TE图形化编程能力**

  Mind Studio提供了业界首个基于TE（类TVM）的编程开发的集成开发环境，让不同平台下的算子移植更加迅捷，适配NPU速度更快。

- **日志分析**

  Mind Studio为NPU平台提供覆盖全系统的日志收集与日志分析解决方案，提升运行时算法问题定位效率。统一全系统日志格式，以Web化的形式提供跨平台日志可视化分析能力及运行时诊断能力，提升日志分析系统的易用性。

- **性能分析**

  Mind Studio以图形界面以及命令行两种UI呈现方式，实现针对Host和Device多节点、多模块异构体系的高效、易用、可灵活扩展的系统化性能分析，以及实现针对NPU Device的性能和功耗的同步分析，满足算法优化对系统性能分析的需求。

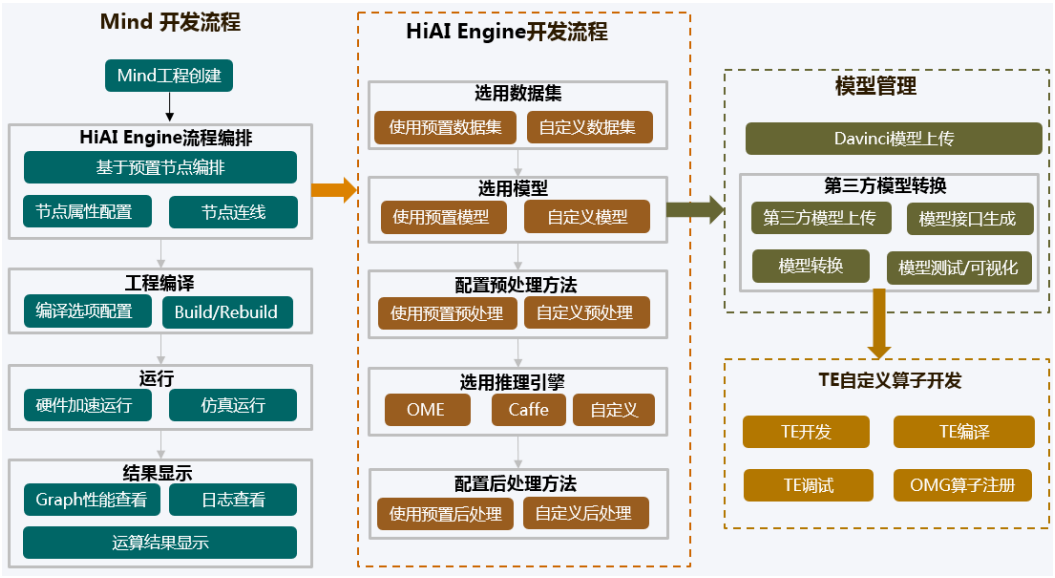
  

- **仿真**
  提供AI核的功能级仿真运行库，通过程序调用仿真运行库模拟执行AI核功能 。

# 2.3 开发流程总览

Mind Studio开发流程如图2-2所示。

**图 2-2** Mind 开发流程总览




# 2.4 应用开发

介绍Mind Studio在进行应用开发时的主要功能。

## 2.4.1 工程管理

Mind Studio支持如下工程类型：

- Python工程
- C/C++工程
- HiAI Engine编排工程（Mind工程）
- 基于离线模型开发的C++工程
- Tensor Engine工程

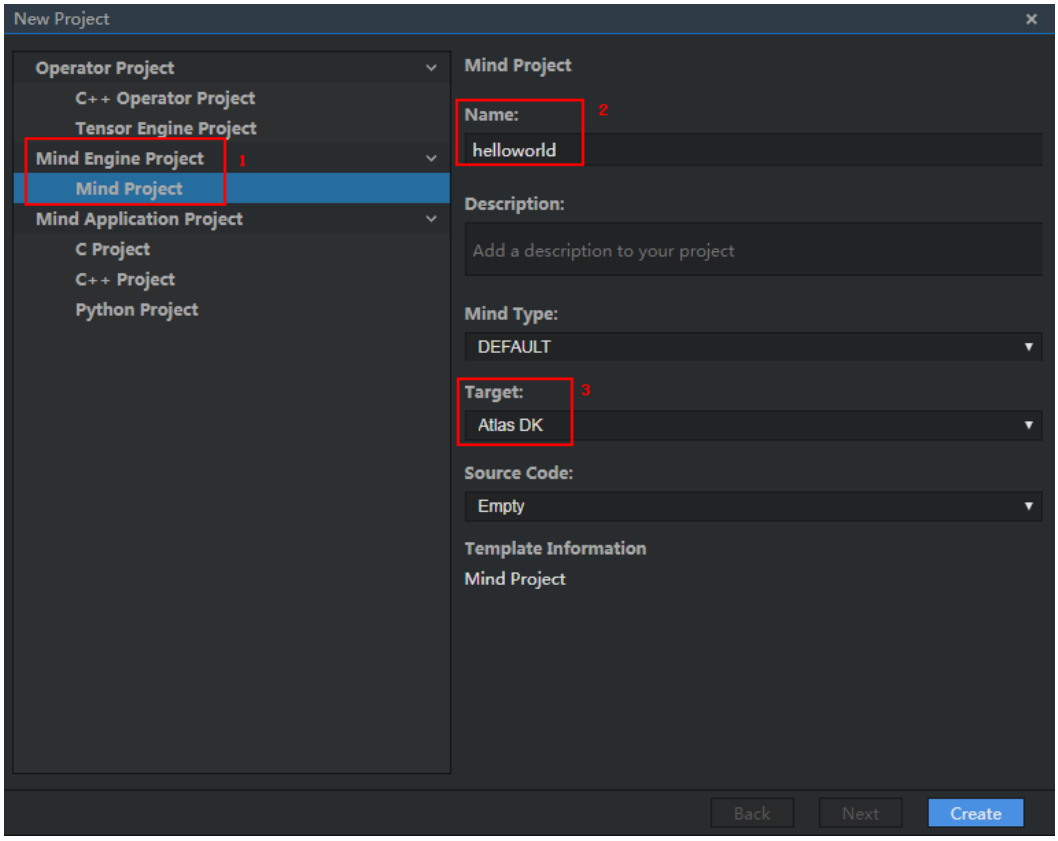
Mind Studio的基础工程管理功能包括：

- 工程创建/删除
- 工程导入/导出
- 工程开启/关闭
- 文件新建/删除
- 文件/文件夹上传

图2-3所示为创建工程示例。

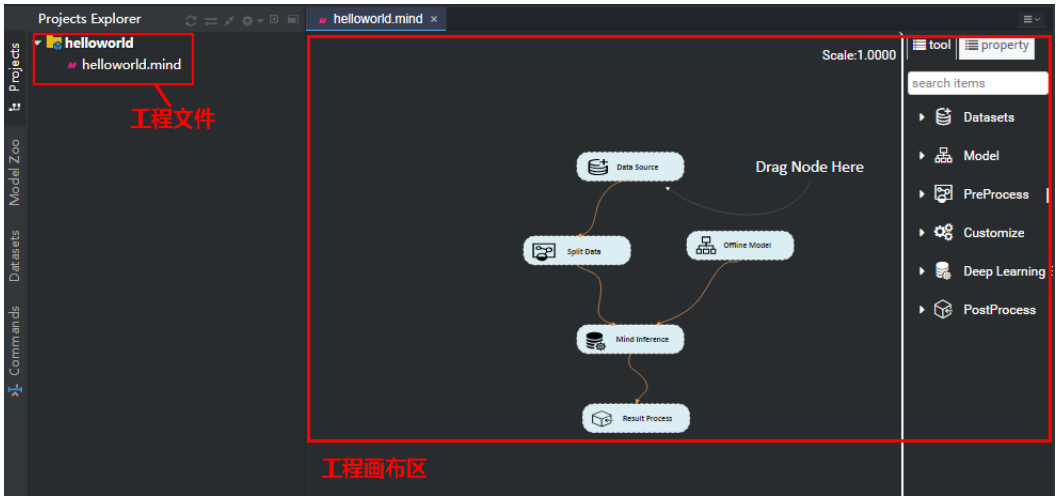
  

**图 2-3** 工程创建




工程创建后，会生成与工程名同名的.mind文件，图2-4为工程创建后的工作区展示。

**图 2-4** 工作区




目前在对画布区域进行操作时，暂不支持如下12类快捷键的使用：

**表 2-1** 画布不支持的快捷键

| 快捷键 | 说明 | 快捷键 | 说明 |
| --- | --- | --- | --- |
| Ctrl+E | 打开最近一次打开的文件 | Alt+F12 | 终端操作窗口 |
| Alt+W | 关闭最近一次打开的文件 | Shift+F10 | 命令行选项 |
| Ctrl+Shift+F | 查找 | Alt+Shift+F9 | Debug配置 |
| Ctrl+Shift+A | 查询可执行的操作 | Alt+← | 上一个文件 |
| Ctrl+Alt+N | 查询文件 | Alt+→ | 下一个文件 |
| Alt+G | 编译配置 | Alt+O | 编辑文件 |

## 2.4.2 图形化业务编排

Mind Studio提供图形化业务编排工具 — HiAI Engine，支持通过拖拽的方式编排工程，流程代码通过DSL自动生成，实现用户“零编码”开发AI应用，从而降低了编码笔误引入Bug的概率。Mind Studio提供了丰富的可视化视图，包括数据流、模型、结果信息、系统分析全部可视化。

Mind Studio支持的节点类型及描述分别如图2-5与表2-2所示。

**图 2-5** Mind Studio 支持的节点类型

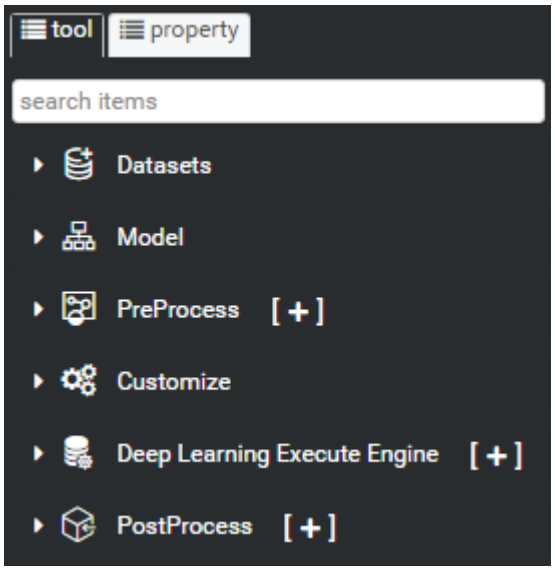



**表 2-2** Mind Studio 支持的节点描述

| 节点类型 | 类型描述 |
| --- | --- |
| Datasets | 数据集节点。 |

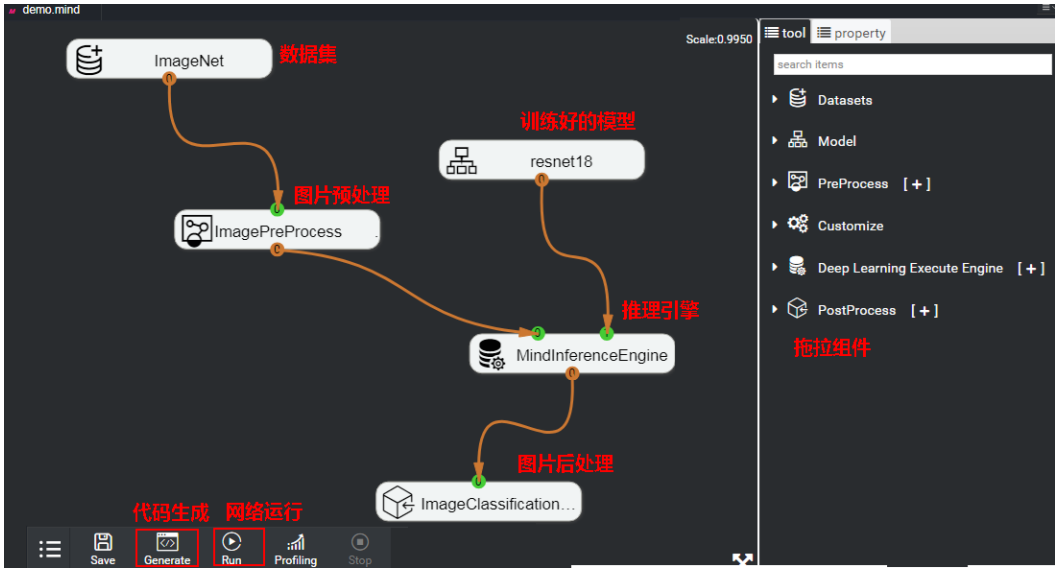
| 节点类型 | 类型描述 |
| --- | --- |
| Model | 模型节点。 |
| PreProcess | 预处理节点。 |
| Customize | 用户自定义节点。 |
| Deep Learning Execute Engine | 深度学习网络执行引擎。 |
| PostProcess | 后处理节点。 |

流程编排的基本操作包括：

- 节点拖拽：将tool栏的业务节点拖拽到画布，在画布中选中节点后，可以在右侧的property页签中对节点设置属性。
- 节点连线：按住鼠标左键进行连线，将流程串接，上一个节点的输出作为下一个节点的输入。

流程编排示例如图2-6所示。

**图 2-6 流程编排示例**




## 2.4.3 模型管理

模型分为内置模型、自定义模型、Caffe模型。

- **内置模型**

  内置模型由Mind Studio预置，在创建工作空间时就已经存在，用户可以在自己创建的mind工程中使用内置模型，但不能对其进行增加，删除和修改。

  当前已经设置的内置模型：Resnet18。

- **自定义模型**

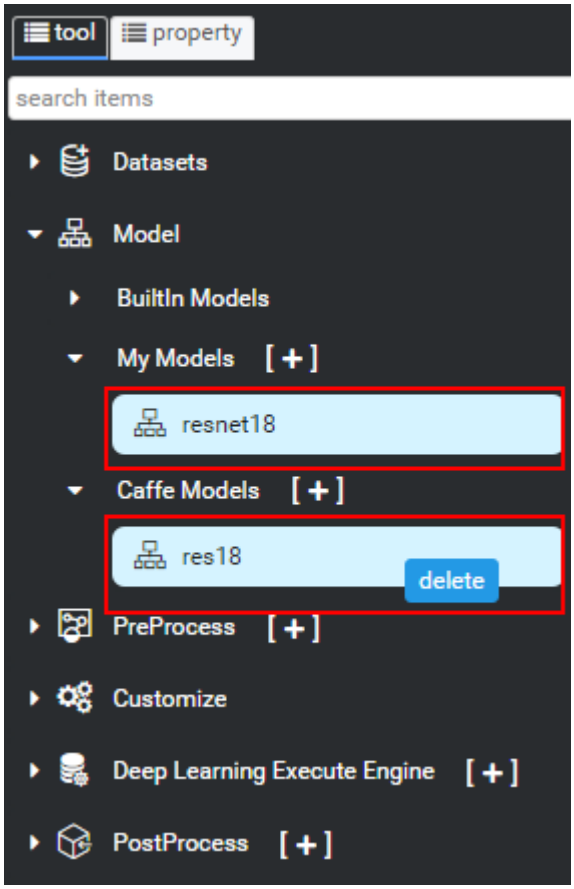
通过模型转化工具，可以将Caffe/Tensorflow等模型转换为自定义离线模型，供用户后续使用。用户可以自行增加自定义模型，刚创建的工作空间自定义模型是空的。自定义模型可以通过模型转换或者新增模型功能增加。

- **Caffe模型**

  通过编排界面的新增Caffe模型，可以将Caffe模型加入到model-zoo中，刚创建的工作空间Caffe模型是空的。

如图2-7所示，右侧的图元节点区域自动显示已有的模型节点，用户进行流程编排时可拖拽使用。

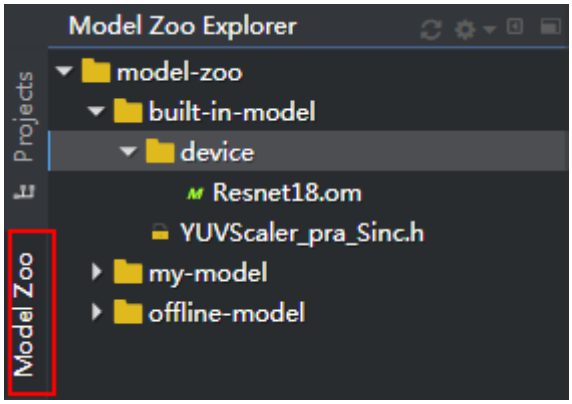
**图 2-7** 图元展示




如图2-8所示，左侧Model Zoo区域可以浏览模型包含的文件结构。

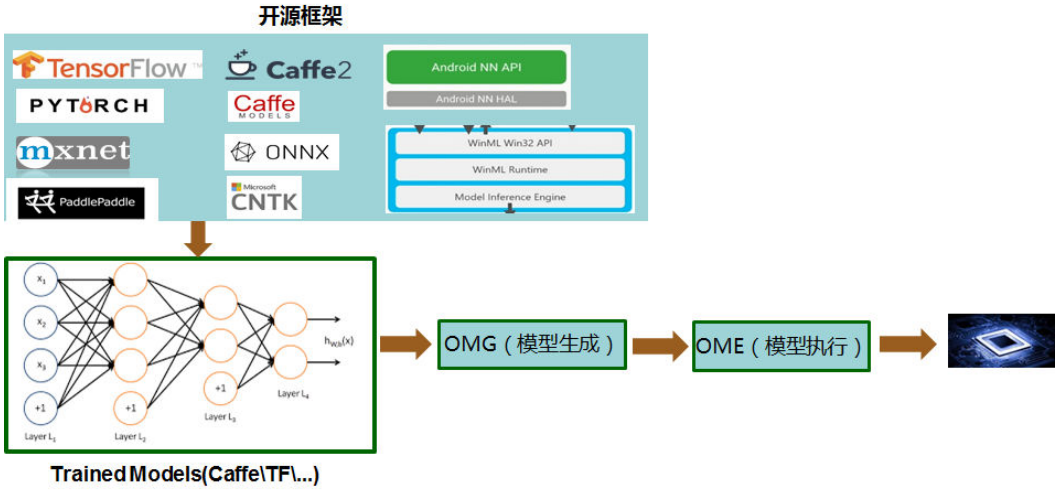
**图 2-8** Model Zoo 区域




## 2.4.4 离线模型转化

离线模型转化可以将Caffe、Tensorflow等业界开源的神经网络模型转化为华为NPU芯片支持的网络模型，华为离线模型转化整体方案如图2-9所示。

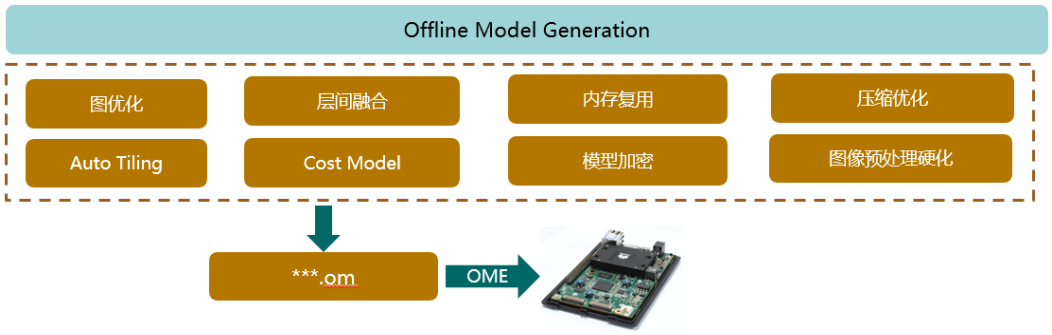
**图 2-9** 离线模型转化整体方案




- OMG（Offline model Generation）：离线模型生成工具，一次性生成可高效在芯片上运行的离线模型。
- OME （Offline model Inference Engine）：离线模型推理执行引擎，全系统最优调度，离线模型高效执行。

离线模型转化的关键技术如图2-10所示。

**图 2-10** 离线模型转化关键技术




在OMG离线模型生成过程中，将在各种调优策略中自动选取合适的优化手段生成离线模型。

模型转化的操作流程如下所示：

**步骤1** 通过右键单击工程名或者菜单栏的Tools选择“Convert Model”，进行离线模型转化。

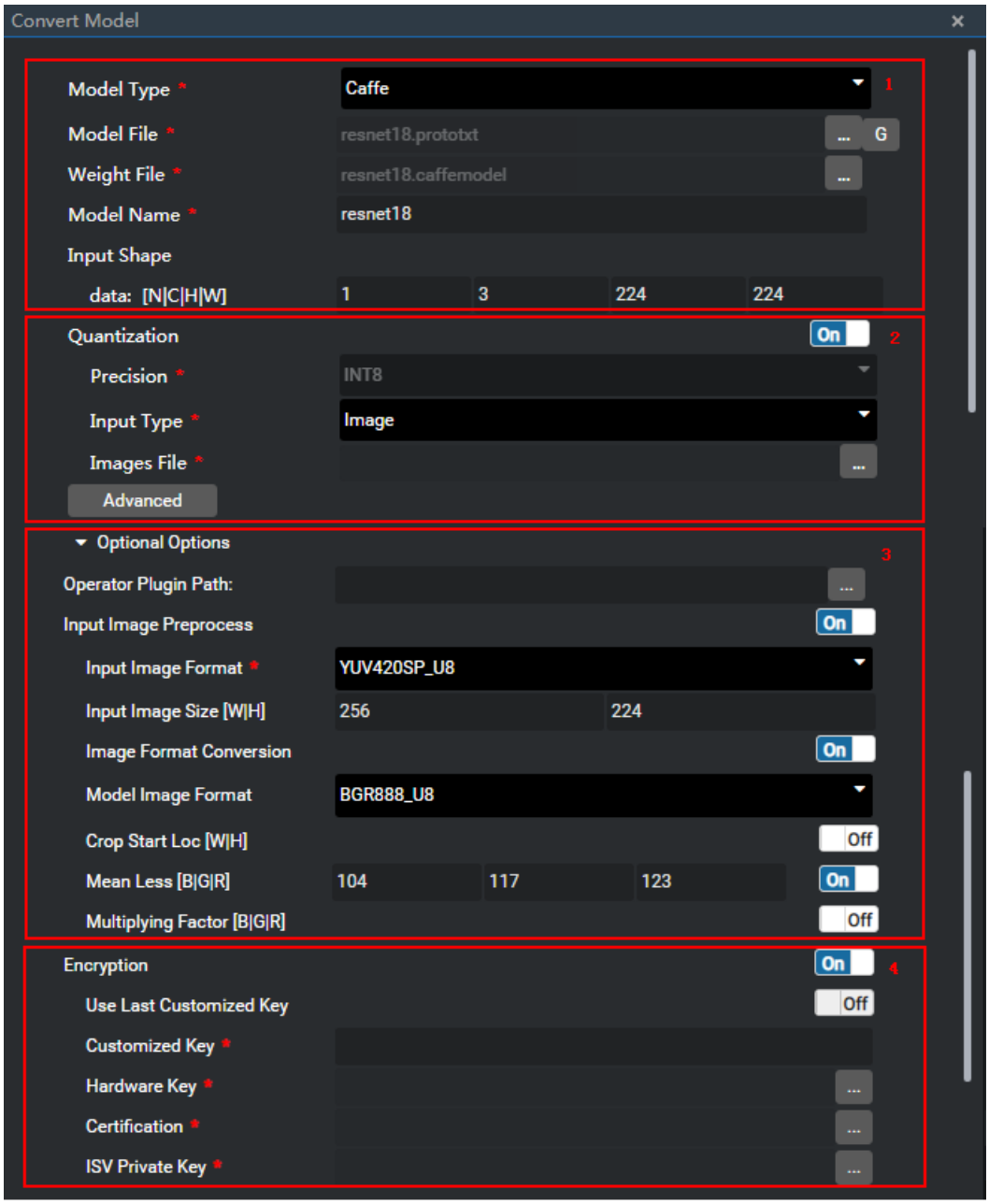
Mind Studio上离线模型转化目前支持从Caffe、Tensorflow模型转换为离线模型。

**步骤2** 支持8bit量化，输入验证集后，推理速度更快，占用内存更少。

如图2-11中的序号2所示。

**图 2-11** 模型转化配置




**步骤3** 图像预处理硬化，和NN网络首层融合，加快运行效率。

如图2-11中的序号3所示。

**步骤4** 支持硬件Key对模型加密，保护模型知识产权。

如图2-11中的序号4所示。

模型转化过程可视化，提示用户转换进度。

转化成功后，提示模型占用存储空间、运行时内存消耗，提前识别资源占用风险。

模型转化失败后，自动算子分析报告生成，友好提示用户。

**----结束**

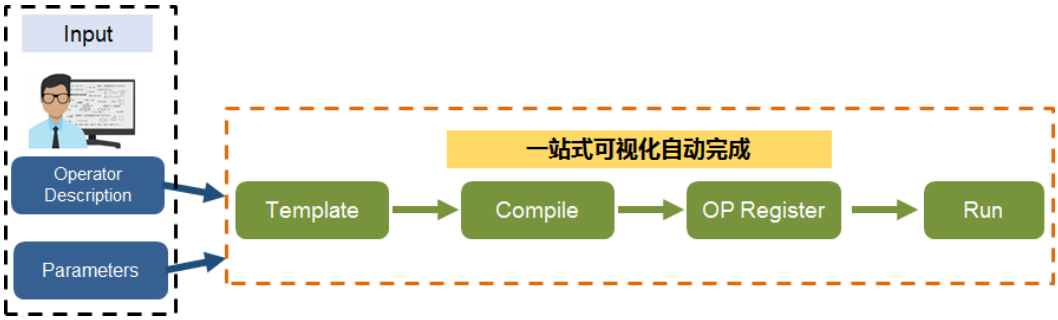
## 2.4.5 自定义算子开发

在Mind Studio中进行模型转化时，如果提示某算子不支持（即在加速算子库如CCE算子库、AI CPU算子库中未实现），则未实现的算子就需要用户进行自定义，自定义的算子可加入到算子库中，使得模型转化过程可以正常进行。

在Mind Studio中提供了TE（Tensor Engine）的自定义算子开发工具，可以开发自定义算子。TE是基于TVM（Tensor Virtual Machine）的自定义算子开发框架，提供了基于Python语法的DSL语言供开发者开发编写自定义算子。

图2-12为自定义算子开发流程图。

**图 2-12 自定义算子开发流程**




在模型转化中使用自定义算子的整体流程如下，详细步骤可以参考《TE自定义算子开发指导》。

**步骤1** 在Mind Studio中使用TE框架开发算子。

1. 创建Mind类型工程。
2. 使用TE框架编写代码实现算子，如果本地有已实现的算子文件，可通过菜单“File > Upload Project”选项上传到用户自定义工程。
3. 构建算子，并测试算子正确性。

**步骤2** 在Mind Studio中开发算子插件，将**步骤1**中开发的算子以插件的形式插入模型转换流程中。

**步骤3** 重新进行离线模型转换。

**----结束**

## 2.4.6 数据集管理

Mind Studio数据集分为内置数据集和自定义数据集，其中内置数据集用户可以直接拖拽使用，自定义数据集需要用户手动导入。

- 内置数据集

  由Mind Studio开发者提供，预置到Mind Studio中，用户可以直接使用。
- 自定义数据集

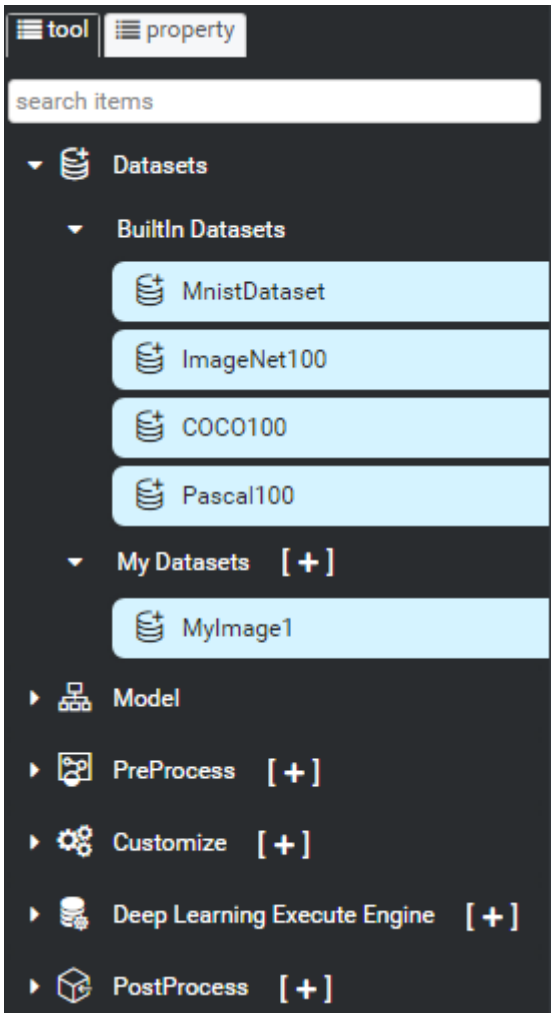
  用户可以自定义数据集功能，可以将图片的集合保存为自定义数据集，供用户后续使用。

  自定义数据集可以通过本地文件或本地文件夹来获取图片来源。

如图2-13所示，右侧的图元节点区域自动显示已有的数据集节点，用户流程编排时可以拖拽使用。

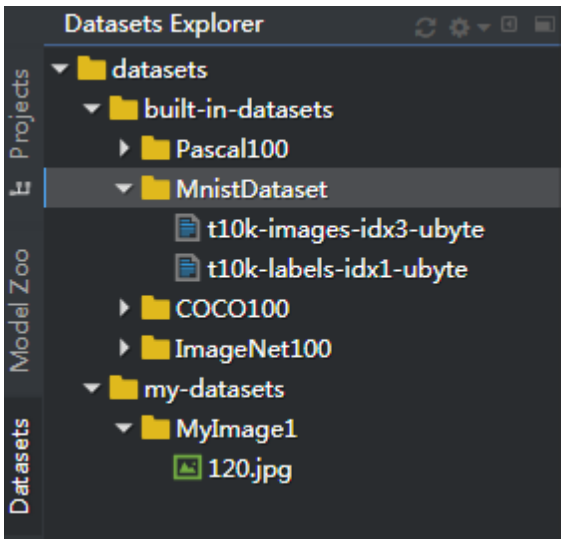
  

**图 2-13** tool 图元展示区




如图2-14所示，Datasets区域可以浏览各数据集包含的文件。

**图 2-14** Datasets 区域



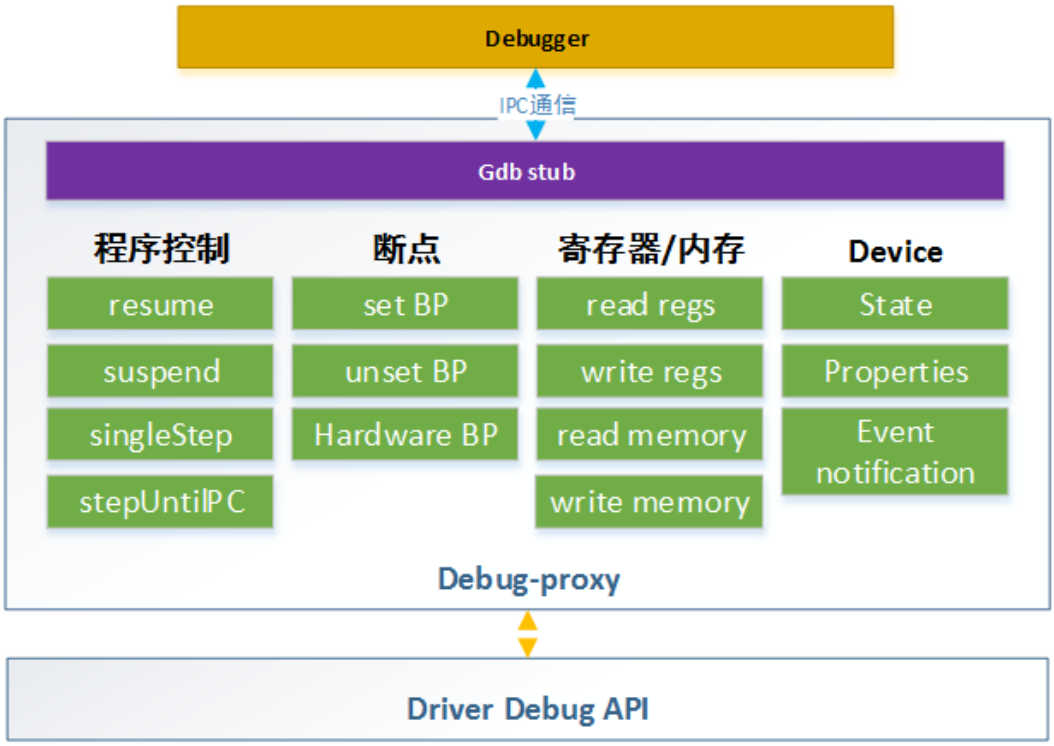

  

# 2.5 调试及性能调优

## 2.5.1 调试工具

Mind Studio调试工具原理如图2-15所示。

**图 2-15** 调试工具运行原理




Mind Studio调试工具的主要功能包括：

- 设置断点，包括软件断点和硬件断点，断点可以是文件+行号，函数名。
- 打印函数调用栈。
- 打印和修改函数内局部变量、参数及全局变量。
- 打印和修改所有寄存器。
- 打印和修改内存。
- 支持attach模式（运行过程中调试）。
- 支持watchpoint功能（监控内存修改）。

Mind Studio调试工具特点如下：

- 支持命令行界面调试。
- 支持图形界面调试。
- 可以调试控制CPU上的代码。
- 可以调试AICore的代码。

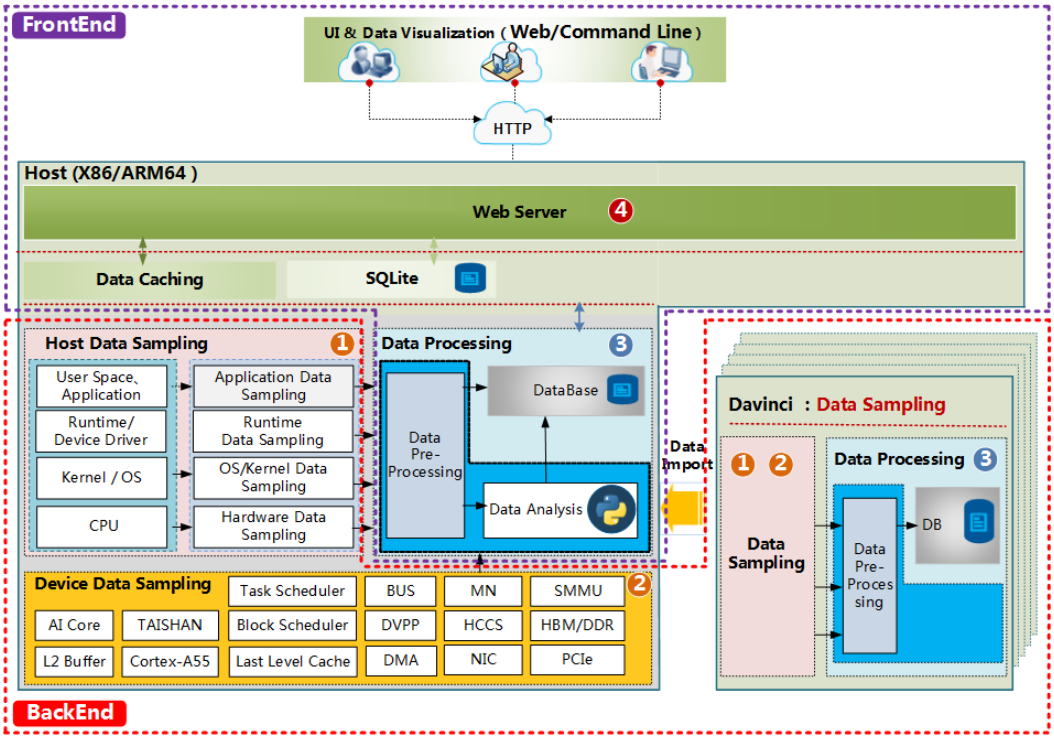
- 可以调试AICPU上的代码。

## 2.5.2 性能 Profiler

Mind Studio以图形界面以及命令行两种UI呈现方式，实现针对Host和Device多节点、多模块异构体系的高效、易用、可灵活扩展的系统化性能分析，以及实现针对NPU Device的性能和功耗的同步分析，满足算法优化对系统性能分析的需求。

性能Profiler原理如图2-16所示。

**图 2-16** 性能 Profiler 原理图




Mind Studio的性能数据分析包含：

- 时间片分析：包含AI Core执行时间片分析，AI CPU执行时间片分析， RUNTIME API执行时间片分析。
- 指令数性能
- 内存性能指标：包含Device内存利用率，内存加载事务数量，内存加载吞吐量，内存加载事务数量等指标。
- HCCS性能指标：包含提供HCCS发送/接收的数据总量，HCCS用户数据的发送/接收总量，HCCS发送/接收的数据开销等。
- FU性能指标：包含FU执行load/store的利用率，控制指令的利用率，特性操作如sin、cos的利用率等。
- Task Scheduler性能指标：包括Task执行顺序、队列状态统计、处理器load统计等。
- 带宽性能指标：包括Host端PCIe接口统计、总线带宽占用统计、DVPP输入/输出接口带宽统计等。
- 系统性能指标：包括系统时钟、内存时钟及温度等。

性能分析结果如图2-17、图2-18和图2-19所示。

**图 2-17** 运行状态分析

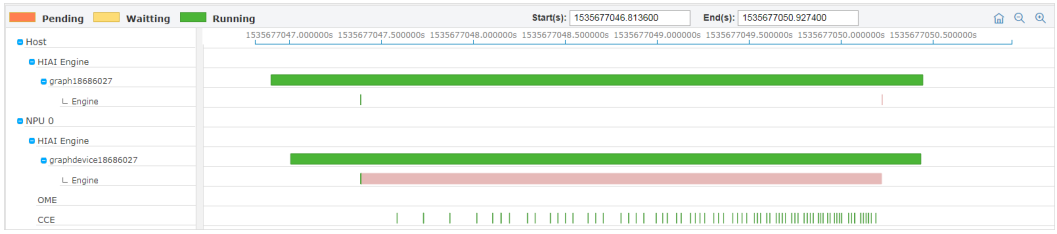



**图 2-18** Thread 分析

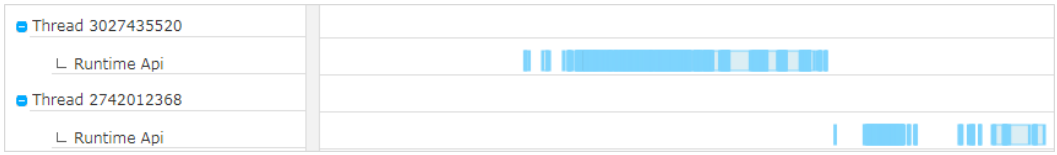
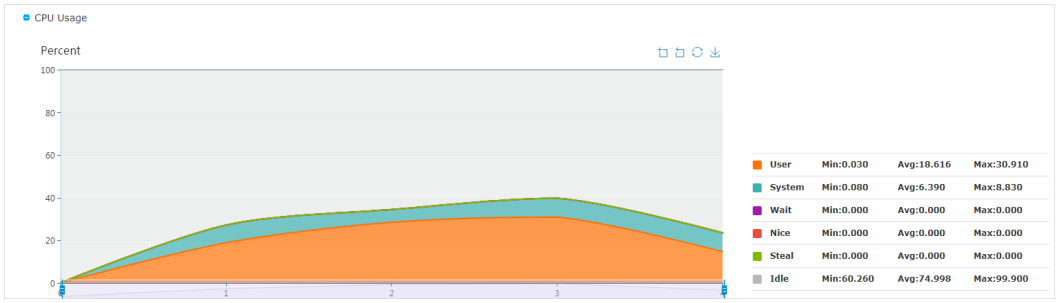



**图 2-19** CPU 利用率分析

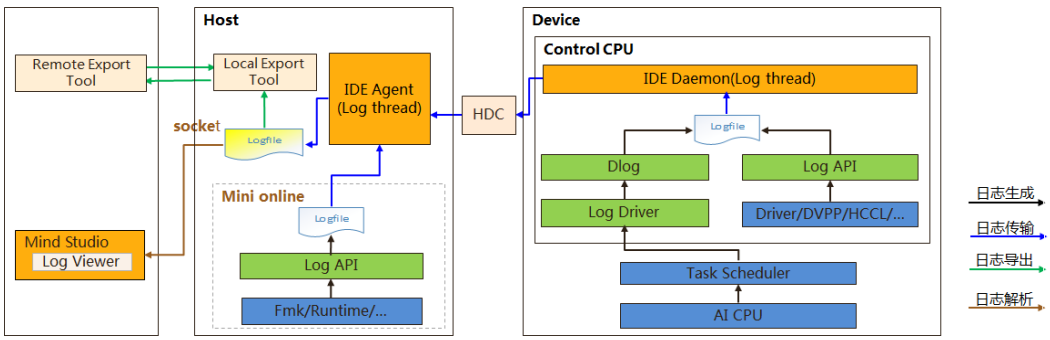



## 2.5.3 日志分析

Mind Studio为NPU平台提供覆盖全系统的日志收集与日志分析解决方案，提升运行时算法问题定位效率。统一全系统日志格式，以Web化的形式提供跨平台日志可视化分析能力及运行时诊断能力，提升日志分析系统的易用性。

Mind Studio日志分析原理如图2-20所示。

**图 2-20** 日志分析原理图




- Device侧负责日志文件生成，通过HDC通道传输日志文件。
- Host侧负责日志文件的转储和压缩。
- Mind Studio负责日志文件的解析和显示。

- 支持界面导出Host侧存储的日志文件。

目前日志支持的模块有：Dlog、Slog、IDE-daemon-host、IDE-daemon-device、Log-agent-host、HCCL、Framework、HiAI Engine、DVPP、Runtime、CCE、HDC、Driver、MDC、DEVMM、Kernel等。

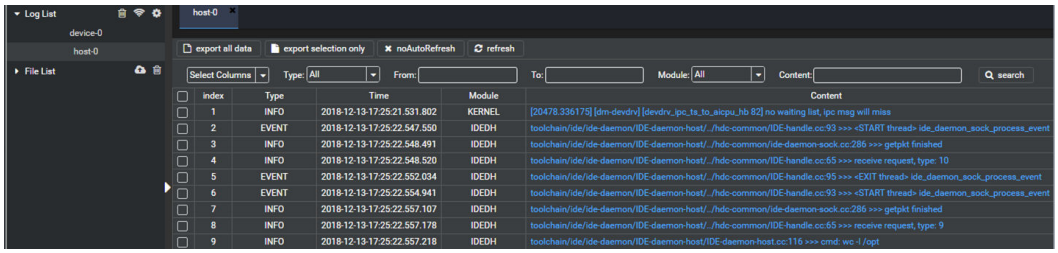
- Slog：系统生成的日志。
- HiAI Engine：模型。
- HCCL：华为集合通信库，提供集合通信的Reduce,Gather等操作的API。
- MDC：自动驾驶，包括规控、空间感知、监控、定位等。
- DEVMM：设备内存管理。
- Kernel：系统内核。

各个模块上报的Log信息在IDE上集中展示。

输出日志支持按模块过滤，按时间点过滤，按Log类型过滤，按关键字查询筛选，离线日志导入分析，并支持Log的全部导出及过滤后Log的导出功能。

日志分析结果如图2-21所示。

**图 2-21 日志分析**




## 2.5.4 黑匣子

黑匣子用来保存死机重启前重要运行信息，为后续的死机定位提供调试信息。

以下场景会触发Mind Studio的黑匣子功能：

- 软件引起的系统停止响应后重启：linux Panic、驱动异常、安全OS异常等。
- 硬件异常引起的系统停止响应后重启：D 芯片超过一定温度、DDR总线停止响应。
- 子系统启动失败，包括控制CPU 系统启动失败、TS 启动失败、LPM3 启动失败。

# 3 构建首个机器学习应用

## 3.1 概述

Mind Studio的Engine流程编排功能提供AI引擎可视化拖拽式编程及算法代码自动生成技术，极大的降低了开发者的门槛。

业务开发人员通过拖拽图形化业务节点、连接业务节点、编辑业务节点属性的方式编排和运行业务流程，实现业务流程编排“0”编码。

HiAI Engine是一个通用的业务流程引擎，主要包含HiAI Engine Agent与HiAI Engine Manager。

- **HiAI Engine Agent**：运行在**Host侧**（**Host侧**指与Device相连接的X86服务器、WindowsPC、3559以及AMD服务器等），会利用Device提供的NN计算能力，完成业务功能。
- **HiAI Engine Manager**：运行在**Device侧**（**Device侧**指安装了NPU芯片的硬件设备，利用Pcie接口与Host侧连接），为Host提供NN计算能力。

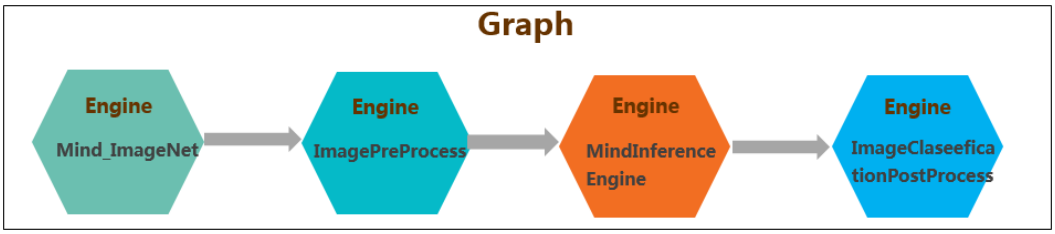
Mind Studio与HiAI Engine的交互流程如图3-1所示。

**图 3-1** 工具与 Engine 交互流程


HiAI Engine中的Engine是一个具体的业务节点，一个业务节点表示一次处理过程，多个Engine组成一个Graph，Graph负责对Engine的管理，如图3-2所示。

**图 3-2** Graph 与 Engine 关系示例

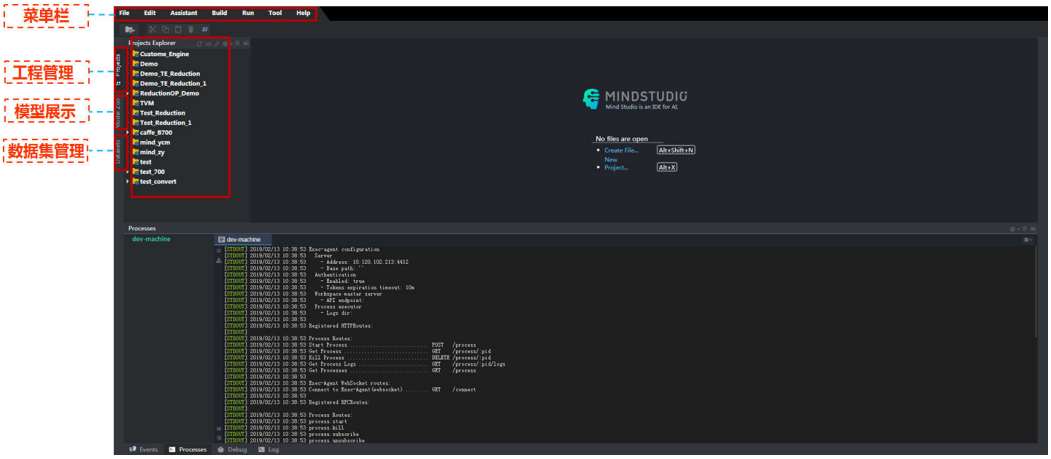



每个Engine节点间的连接在Graph配置文件中配置，节点间数据的实际流向根据具体业务在节点中实现，通过向业务的开始节点灌入数据启动整个Engine计算流程。

# 3.2 工具界面总览

打开Mind Studio工具后进入的工具界面如图3-3所示。

**图 3-3** 工具界面总览



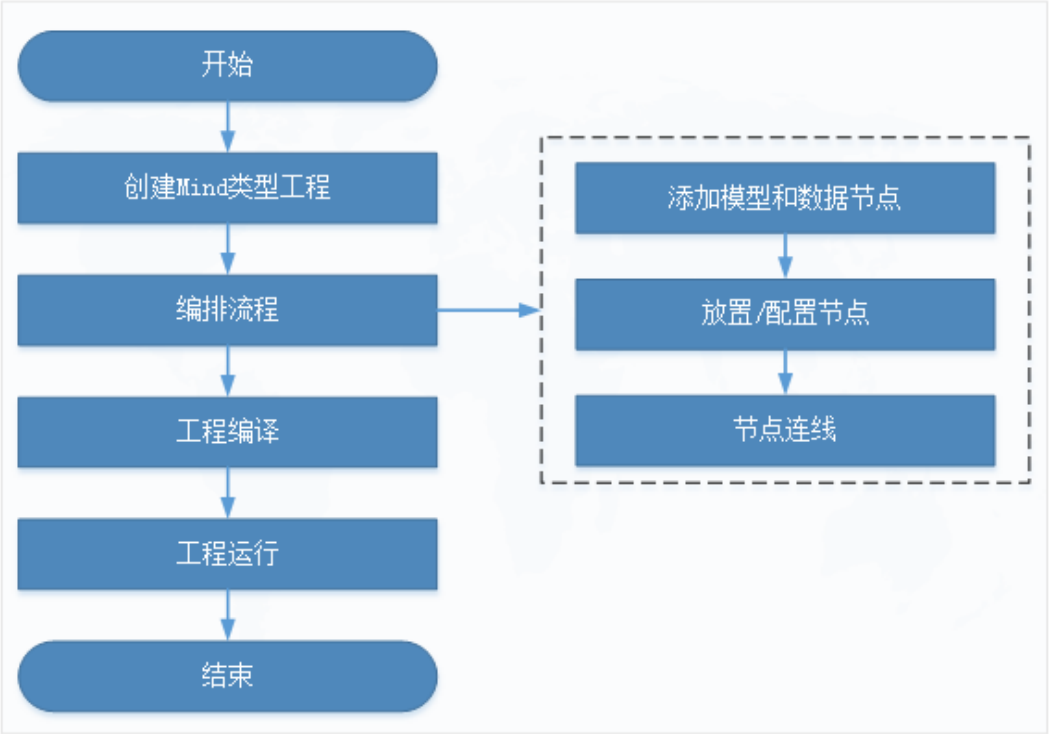

- 顶部为菜单栏，包含工程/文件创建、编辑、删除、工程编译、运行、设置等功能。
- 左侧包含工程管理、模型管理、数据集管理页签，单击对应的页签，可以在页签右侧显示工程、模型或者数据集详情，如上图中的**Projects Explorer**显示工程详情。

# 3.3 操作流程

在Mind Studio中进行业务编排的流程如图3-4所示。

**图 3-4** 业务编排流程




下面以分类网络和检测网络为例，分别介绍工具的编排功能基本使用方法。

## 3.3.1 分类网络操作流程

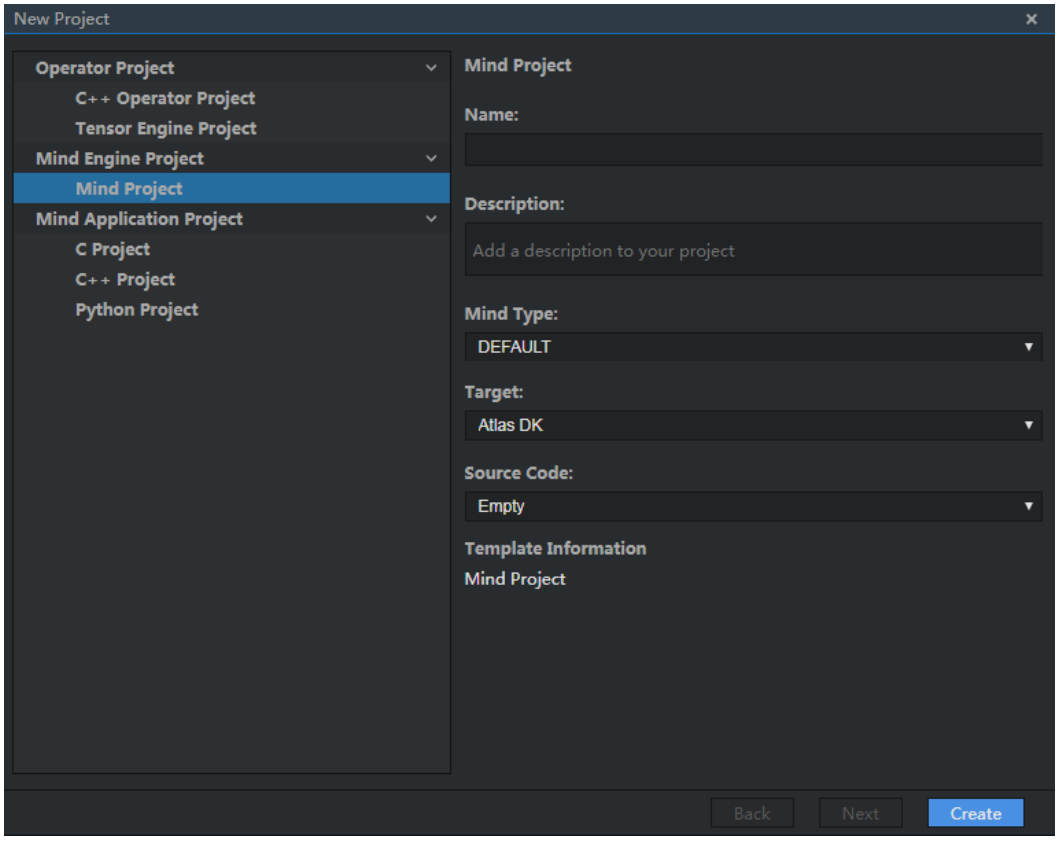
### 3.3.1.1 创建 Mind 工程

**步骤1** 依次选择“File > New > New Project”，弹出“New Project”窗口，创建一个新的工程。

**步骤2** 在“New Project”窗口中选中“Mind Engine Project > Mind Project”工程，显示窗口配置界面，如图3-5所示。

**图 3-5** 创建 Mind Engine 工程




**步骤3** 参考表3-1进行工程配置。

**表 3-1** 参数说明

| 参数 | 参数说明 |
| --- | --- |
| Name | 工程名称，自行配置。<br>名字需要为字符串，不包含空格，空格会自动填充为“-”。 |
| Mind Type | 包含如下两种选项：<br>● DEFAULT：创建的工程会生成画布，用户可以使用拖拽的方式进行工程编排。<br>● CUSTOM：自定义工程，不会生成画布。 |
| Target | 下拉选择运行环境：<br>● Local：仿真环境，用于caffe推理引擎。<br>● ASIC：连接EVB、PCIe单板。<br>● Atlas DK：连接开发者板。<br>**说明**<br>如果新建工程为Local仿真工程，则模型必须使用Model中Caffe Models下的模型组件（“新增Caffe模型组件”章节请参见《Mind Studio基本操作》手册），对应预处理选择ImagePreProcessPillow节点，推理引擎对应必须为CaffeInferenceEngine，详细使用请参见3.6 开源Caffe模型Engine编排。 |

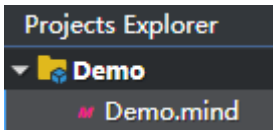
  

| 参数 | 参数说明 |
| --- | --- |
| Source Code | 代码来源：<br>● Empty：选择该项，表示工程非外部导入。<br>● Local(Web Client)：从本地Windows中导入源文件，选择该项，下方出现文件上传输入框。<br>● Local(Web Server)：从后端服务器导入源文件，选择该项，则下方出现输入框填写Mind Studio服务器端代码路径。 |

此工程示例中“Mind Type”选择“DEFAULT”，“Target”选择“Atlas DK”（开发者板），“Source Code”选择“Empty”。

**步骤4** 单击“Create”新建一个**Mind**工程，**DEFAULT**模式下新建的**Mind**工程下会自动生成跟工程名相同的**mind**文件，如图3-6所示，该文件不可复制、删除、重命名。

**图 3-6** 创建工程示例

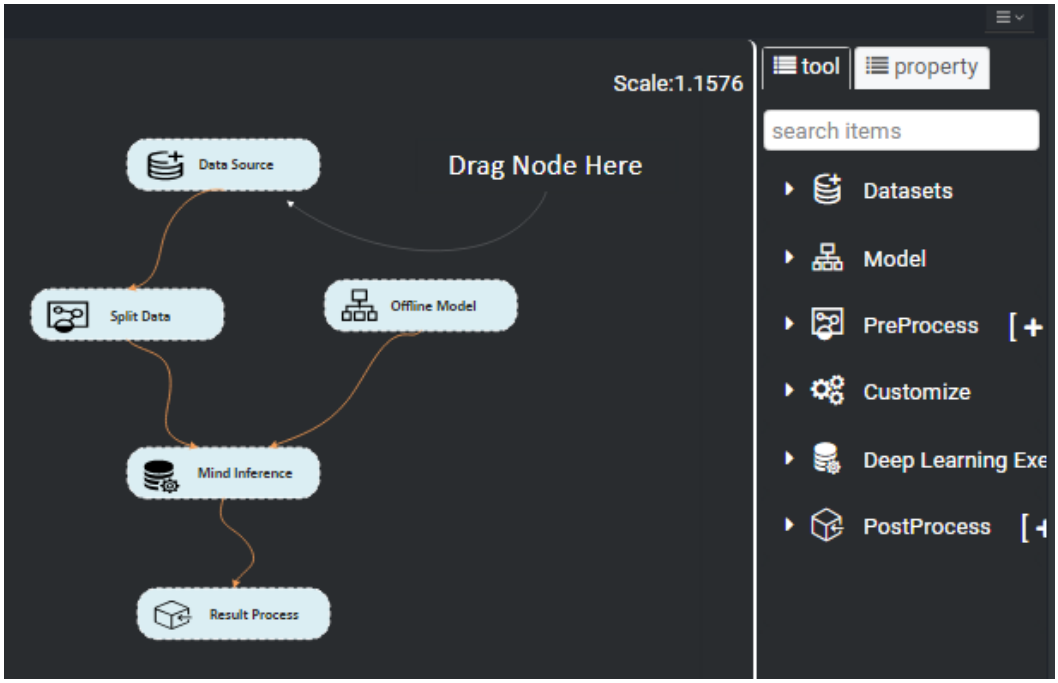



**----结束**

### 3.3.1.2 编排流程

您可以通过双击新创建的后缀名为**.mind**的文件（例如*Demo.mind*）即可打开Engine流程编排窗口，如图3-7所示。

**图 3-7** 流程编排界面

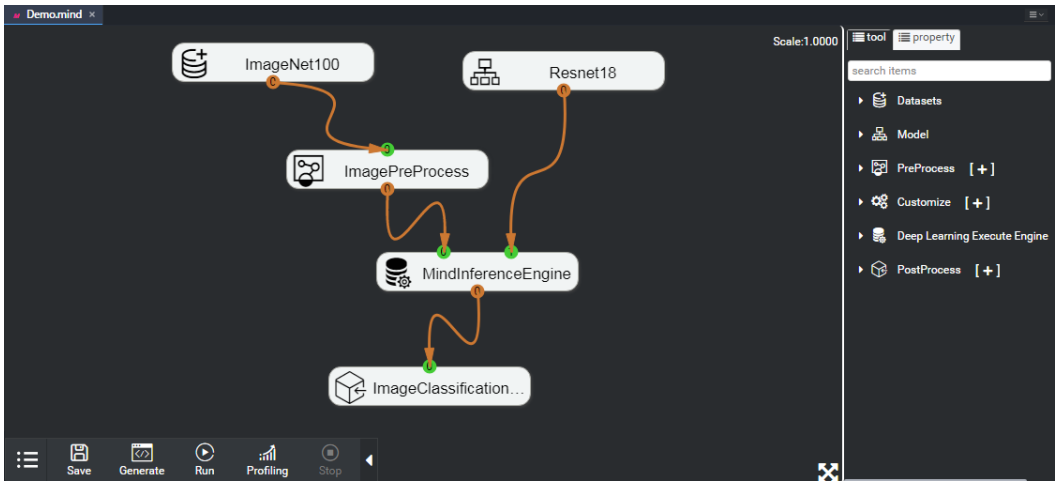


您可以对图中的节点进行如下的操作：放置节点、删除节点、复制节点、设置节点的属性、保存节点、新增节点，详细操作可参考3.7.2 节点基本操作。

**Resnet18**网络如图3-8所示。

**图 3-8** Resnet18 网络示例




**Resnet18**网络主要包含如下节点：一个数据集、一个模型、一个数据预处理、一个执行引擎以及一个图片后处理节点。

## 操作步骤

**步骤1** 添加模型。

本示例使用Mind Studio内置的网络模型 “Model > Built-in Models > Resnet18”。

若用户使用自己的模型，请在右侧工具栏中选择“ Model > My Models”右侧的“[+]”，在弹出“New Model”窗口中导入模型文件即权重文件，配置相关参数，将用户自己的模型转化为支持华为NPU芯片的模型。

详细的模型转化配置参数请参考《Mind Studio基本操作》中的“模型管理 > Mind Engine的模型管理 > 新增自定义模型组件”章节。

**步骤2** 添加数据集节点。

本示例使用Mind Studio内置的数据集“Datasets > BuiltIn Datasets > ImageNet100”。

如果用户想使用自定义数据集，请在右侧工具栏中选择“Datasets > My Datasets”右侧的“[+]”，在弹出的Import Dataset窗口供填写Dataset Name，选择Data Type及Data Source，导入用户自定义数据集。

详细的添加数据集参数请参考《Mind Studio基本操作》中的“数据集管理 > Mind Engine的数据集管理 > 数据集导入”章节。

**步骤3** 将所需节点放置到对应位置。

1. 在右侧**tool**工具栏中，点击“Datasets”项，展开数据集列表，并展开其子项“My Datasets”。
2. 选择**ImageNet**控件，在该控件上长按鼠标左键，将其拖动进入左侧绘制区域，然后松开鼠标左键完成放置，如图3-9所示。

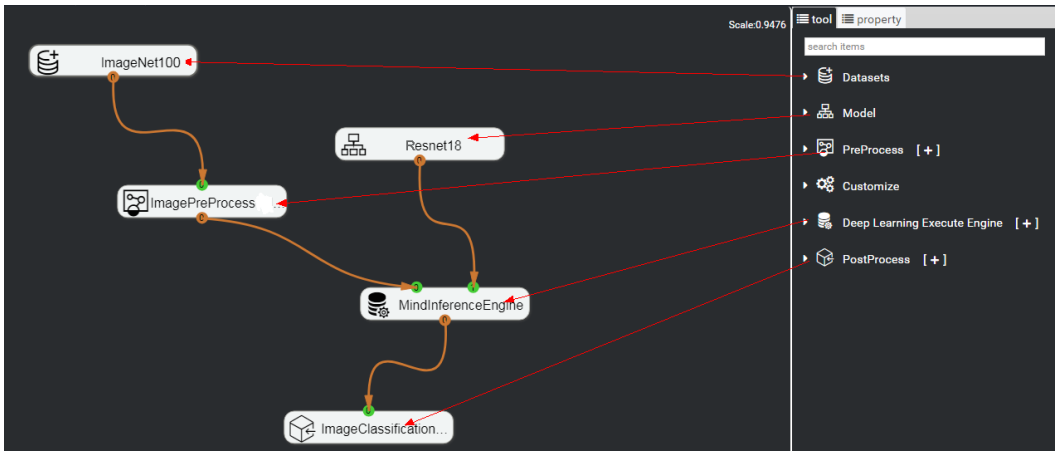
**图 3-9 放置节点操作**


3. 完成**ImageNet**节点的放置后，执行**步骤3.1**和**步骤3.2**放置其他4个节点，具体节点信息如下所示。

- **Model > BuiltIn Models**下的**Resnet18**节点。
- **PreProcess**下的**ImagePreProcess**节点。
- **Deep Learning Execute Engine**下的**MindInferenceEngine**节点。
- **PostProcess**下的**ImageClassificationPostProcess**节点。

完成所有节点放置后，最终节点放置的结果如**图3-10**所示。

**图 3-10 节点放置示例**




**说明**

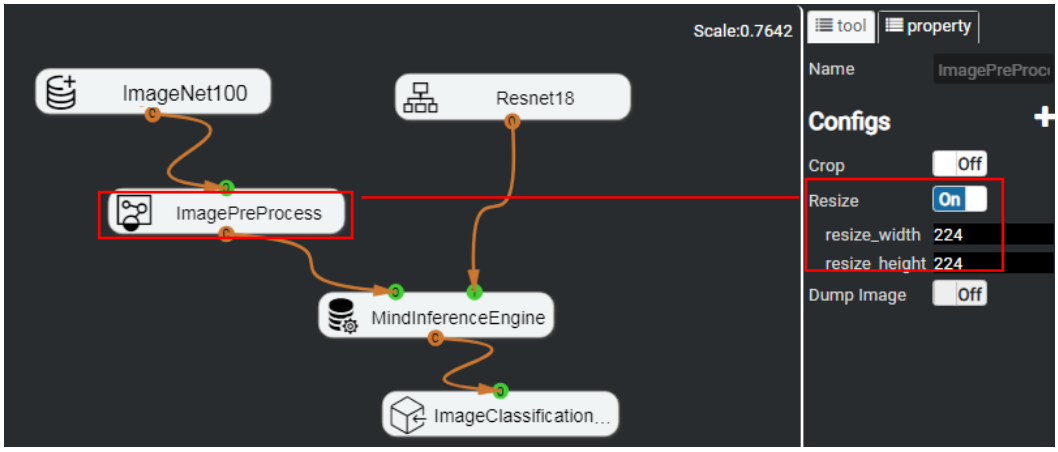
- **Target**为**ASIC**或者**Atlas DK**，**PreProcess**类型需选择**ImagePreProcess**节点。
- **Target**为**Local**，**PreProcess**类型需选择**ImagePreProcessPillow**节点。
- **Deep Learning Execute Engine**下的**CaffeInferenceEngine**与**MindInterfaceEngine**类似，也可以用于分类网络和检测网络的推理，但是**CaffeInferenceEngine**仅支持运行在Local仿真环境。

**步骤4** 配置节点属性。

由于Resnet18网络的要求，需要对**ImagePreProcess**节点的属性进行设置。

1. 点击“ImageProProcess”节点。
2. 在右侧“property”中打开**Resize**开关。
3. 将“resize_width”与“resize_height”都设置为224（默认即为开启，宽与高为224*224），如**图3-11**所示。

**图 3-11** 修改节点属性




**说明**

- Resize的大小设置应与模型的输入要求保持一致，模型要求大小可以通过模型prototxt文件的input_param参数获得，对于Resnet18模型，输入数据格式要求为224*224。
- 仿真环境下预处理节点为ImagePreProcessPillow:

  - Resize属性设置的要求与ImagePreProcess 的属性一致。

  - 该节点还需要设置Mean Value（减均值参数/通道n均值）：

  inception_v4、xcecption、inception_v3这三个模型的B/G/R值均为128。其他模型mean_of_B、mean_of_G、mean_of_R三个值保持默认值即可，默认值分别为104，117，123。

  - 该节点对于densenet、moblienet、moblienet_v2、inception_v4、xcecption、inception_v3这六个模型还需要打开Scale开关，取值为Multiplying Factor(乘系数/通道n方差或（max-min的倒数）)的倒数值；其他网络模型的Scale开关保持默认关闭即可。

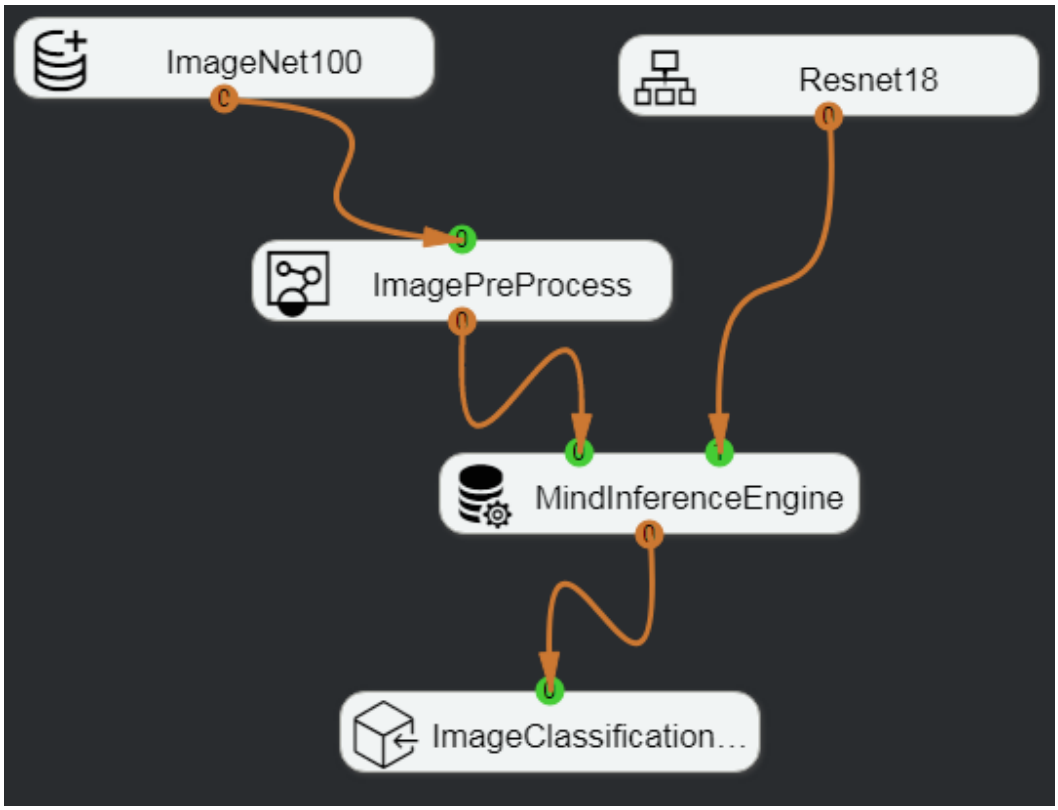
**步骤5** 建立节点连线关系。

在完成所需节点的放置与属性设置后，需要建立其相应的连接关系。

橘黄色的圆形端点为输出端点，可以从该点引出连线，绿色端点为输入端点，可以放置连线。最终节点连线关系示例如图3-12所示。

**图 3-12** 建立节点连线关系




注意

建立连接关系时需注意以下两点:

1. PreProcess类型的节点需连接到Deep Learning Execute Engine类型节点的0号输入端点上。
2. Model类型的节点需连接到Deep Learning Execute Engine类型节点的1号输入端点上。

**步骤6** 单击画布下侧的“Save”。

保存编排的流程。

**----结束**

### 3.3.1.3 编译运行

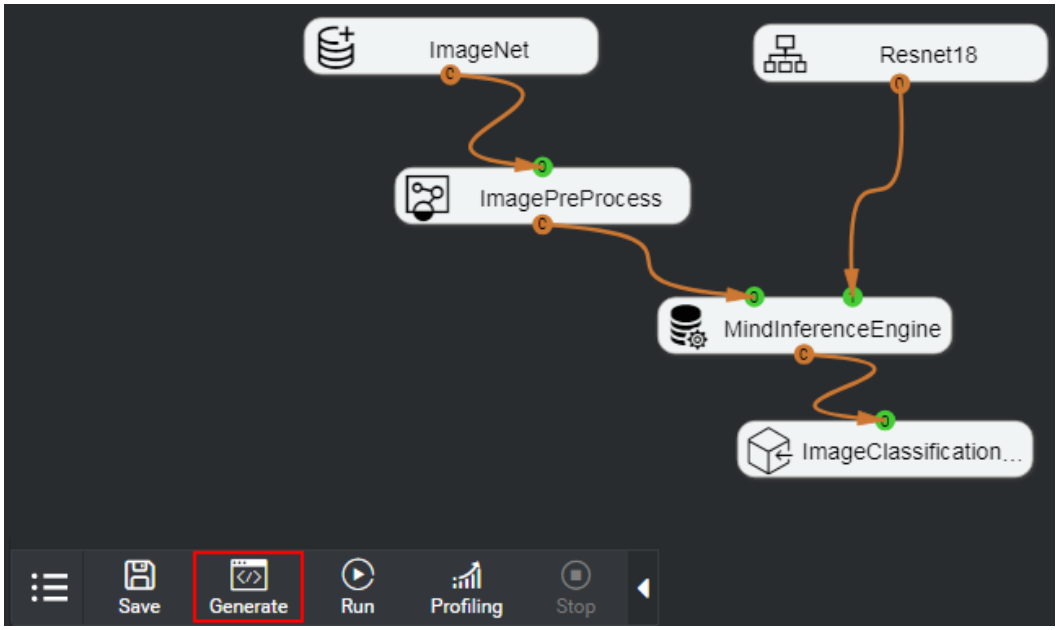
MindEngine支持在设备或者仿真环境中一键式编译运行，根据工程的配置在运行时会执行不同的操作。本示例以在Atlas 200 Developer Kit开发板上为例进行说明。

## 编译

**步骤1** 在完成网络结构的编辑之后，点击左下方的“Generate”来生成对应的源码与执行脚本，如图3-13所示。

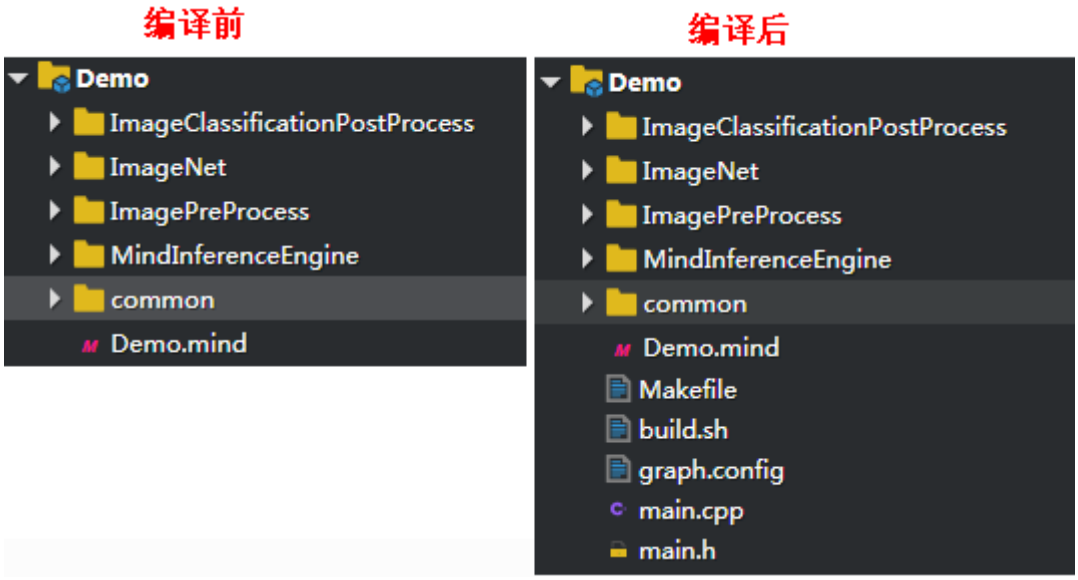
  

**图 3-13** 工程编译




**步骤2** 工程编译后，会自动生成对应的源码与可执行文件，如图3-14所示。

**图 3-14** 编译后生成文件




**步骤3** 解析graph.config文件。

graph文件中配置了Engine相关参数以及Engine之间的连接信息。

- 如图3-15所示，334为ImagePreProcess的ID，“value”为“224”表示resize_width与resize_height的值。

**图 3-15** graph 中配置信息示例

```
engines {
  id: 334
  engine_name: "ImagePreProcess"
  side: DEVICE
  thread_num: 1
  so_name: "./libImagePreProcess.so"
  ai_config {

    items {
      name: "point_x"
      value: "-1"
    }

    items {
      name: "point_y"
      value: "-1"
    }

  items {
    name: "crop_height"
    value: "-1"
  }

  items {
    name: "self_crop"
    value: "1"
  }

  items {
    name: "resize_width"
    value: "224"
  }

  items {
    name: "resize_height"
    value: "224"
  }
```

- 如图3-16所示，graph中配置Engine的连接信息，编号为334的engine的输出数据作为编号为311 eingine的输入数据。

**图 3-16** graph 中连接信息配置

```
connects {
  src_engine_id: 244
  src_port_id: 0
  target_engine_id: 334
  target_port_id: 0
}

connects {
  src_engine_id: 334
  src_port_id: 0
  target_engine_id: 311
  target_port_id: 0
}
```

**----结束**

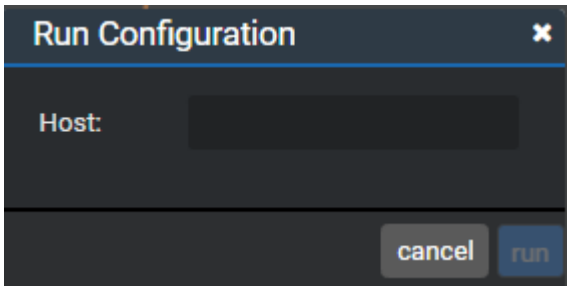
## 运行

**说明**

当工程的target配置选项为ASIC或Atlas DK时，点击Run按钮，会弹出配置host侧IP地址页面，如图3-17所示。若为仿真环境，则不需要配置硬件平台地址，自动启动运行流程。在运行时，工具会自动拷贝工程下模型文件、配置文件、链接库、二进制程序、输入输出数据等到host侧。为提高MindStudio 的运行性能，MindStudio不会自动删除host侧的数据，如用户不需要这些文件，可自行登录host侧，删除/home/HwHiAiUser/HIAI_DATANDMODELSET和/home/HwHiAiUser/HIAI_PROJECTS下的废弃文件夹。

**步骤1** 您可以单击画布下方的“Run”，配置硬件平台的IP地址，如图3-17所示。

**图 3-17** 运行配置




**步骤2** 完成配置后单击“run”，即可启动MindEngine编排流程，此时可以在“dev-machine”窗口中查看对应的运行输出。

**图 3-18** 保存设备信息

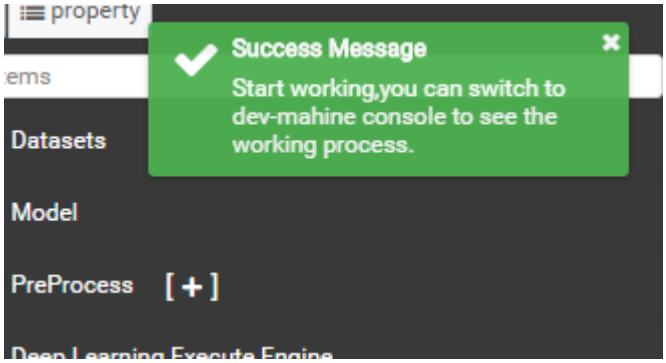
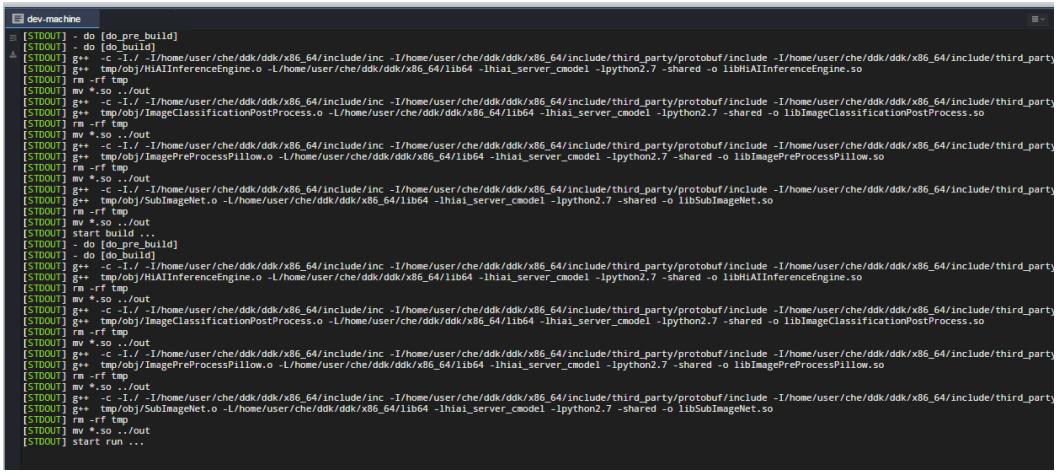


**图 3-19** 运行输出

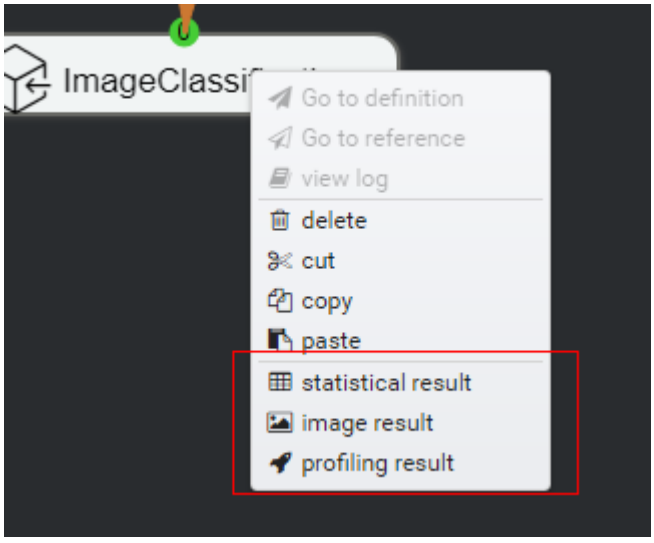



**----结束**

### 3.3.1.4 运行结果查看

运行完成之后，将会在工程根目录下的out文件夹下生成以时间戳命名的结果文件夹，此时即可在**PostProcess**类型节点上右键弹出菜单来查看相应的结果。

**图 3-20** 右键菜单




共有三种类型的结果：image result，statistical result和profiling result。

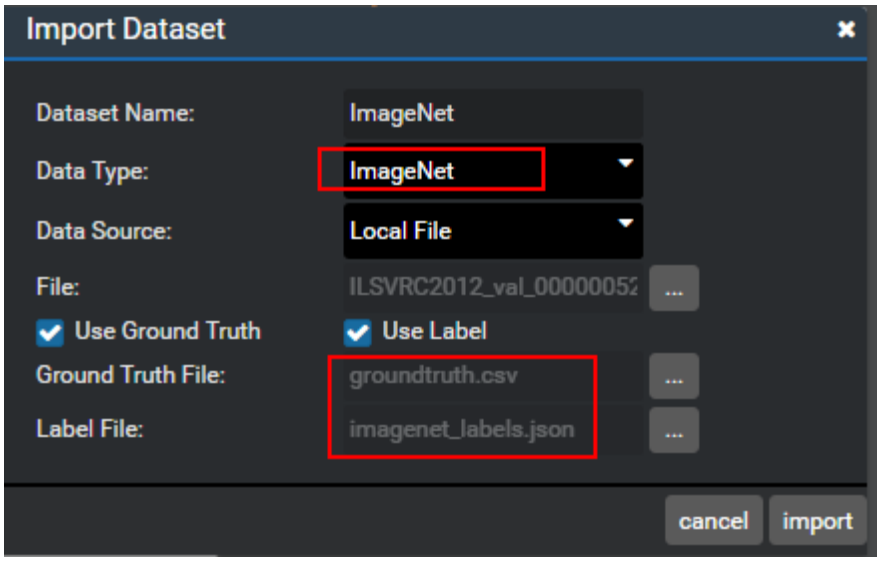
其中，输入数据集是否含有Ground Truth File（标定好的真实数据）和Label File（标签字典文件），对image result 和statistical result的结果显示形态有一定影响。

如下是导入数据集时选择标签字典的样例：

标签字典文件在导入数据时同步导入，Data Type必须选择ImageNet，Ground Truth File和Label File分别选择“标定好的真实数据”和“标签字典文件”，如图3-21所示。

附件是标签字典样例：标签字典.zip

**图 3-21** 标签字典文件导入选择项




不同的选择项会导致不同的结果显示，如表3-2所示。

**表 3-2** 不同选择项对应结果

| 输入数据图片 | Ground Truth File | Label File | 结果显示 |
| --- | --- | --- | --- |
| 导入 | 勾选 | 勾选 | image result 和statistical result均可展示。<br>且image result显示图片推理的类别以及该类别对应的标签文字。 |
| 导入 | 勾选 | - | image result和statistical result均可展示。<br>但是image result显示时只含有图片推理的类别编号，而无法显示该类别对应的标签文字。 |
| 导入 | - | 勾选 | 只显示image result。<br>且image result显示时包含图片推理的类别以及该类别对应的标签文字。 |
| 导入 | - | - | 只显示image result。<br>且image result显示时只含有图片推理的类别编号，而无法显示该类别对应的标签文字。 |

由以上表格总结出：Ground Truth File文件是否导入勾选影响statistical result是否显示；Label File文件是否勾选影响image result结果是否显示标签文字。下面分别说明：

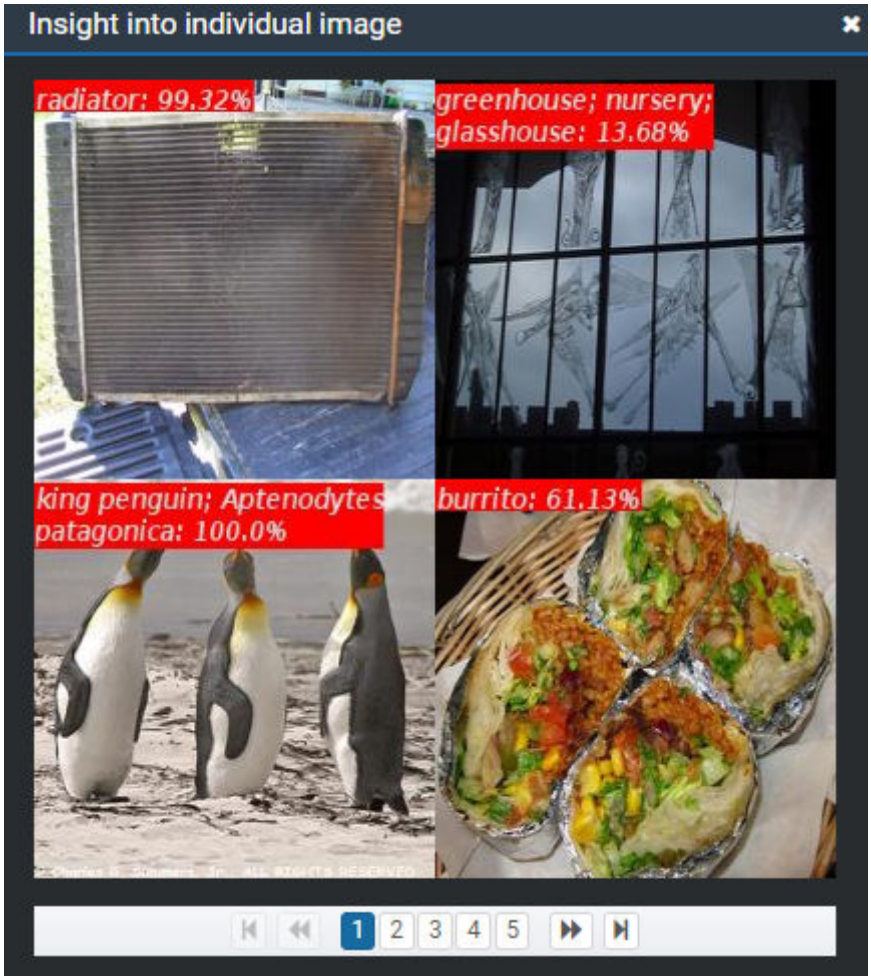
## image result

image result展示推理图片的预测结果，预测结果图片的左上角带有分类和预测概率。

- 如果导入数据集时提供了标签字典文件（Label File）且标签文件确实与所使用的模型配套，则显示时包含标签文字，如图3-22所示。

**图 3-22** 分类网络 image result 示例 1




**说明**

图片上的标签代表该图片的分类标签以及该图片可能为该分类标签中的概率，由于多种标签的含义可能相近，因此同一图片可能有多种分类标签。

- 如果导入数据集时未导入标签字典文件（Label File）或者标签字典文件与模型不配套，则只显示标签编号，如图3-23所示。

**图 3-23** 图片分类示例 2


**说明**

图中No.***（如No.692）表示图片的预测标签编号，百分比（如22.05%）表示预测图片为该分类的一种概率。

## statistical result

如果导入数据集时导入“标定好的真实数据”文件（Ground Truth File），此时可以显示statistical result。

statistical形式的结果为一张表格，列出了所进行推理图片的结果数量，该类数据所含的类别（比如ImageNet默认类别数为1000），以及选取topN hit得到的准确率，如图3-24所示。

**图 3-24** Statistical Result

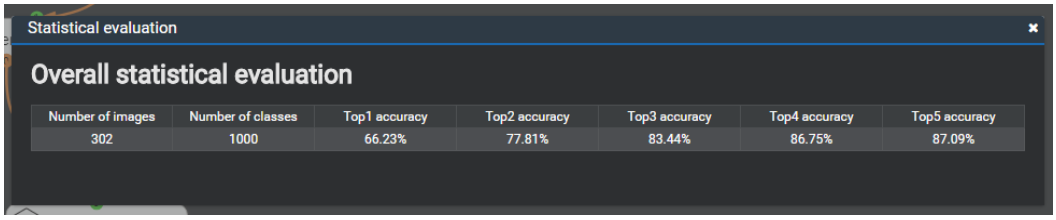


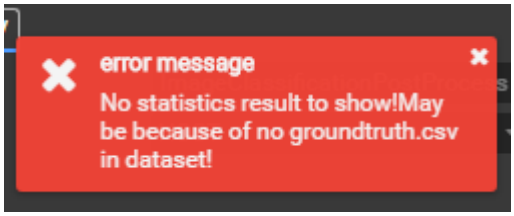
**表 3-3** Statistical 结果说明

| 参数名称 | 参数说明 |
| --- | --- |
| Number of images | 输入图片的总数。 |
| Number of classes | 总的类型数。 |
| Top1 accuracy | 预测概率中最大的分类正确的概率。 |
| Top2 accuracy | 预测概率中前两大的分类中含有正确分类的概率。 |
| Top3 accuracy | 预测概率中前三大的分类中含有正确分类的概率。 |
| Top4 accuracy | 预测概率中前四大的分类中含有正确分类的概率。 |
| Top5 accuracy | 预测概率中前五大的分类中含有正确分类的概率。 |

**说明**

要查看**statistical**，所进行推理的图片数据必须包含在Groundtruth文件中，否则会提示如下错误信息，如图3-25所示。

**图 3-25** 错误信息




## profiling result

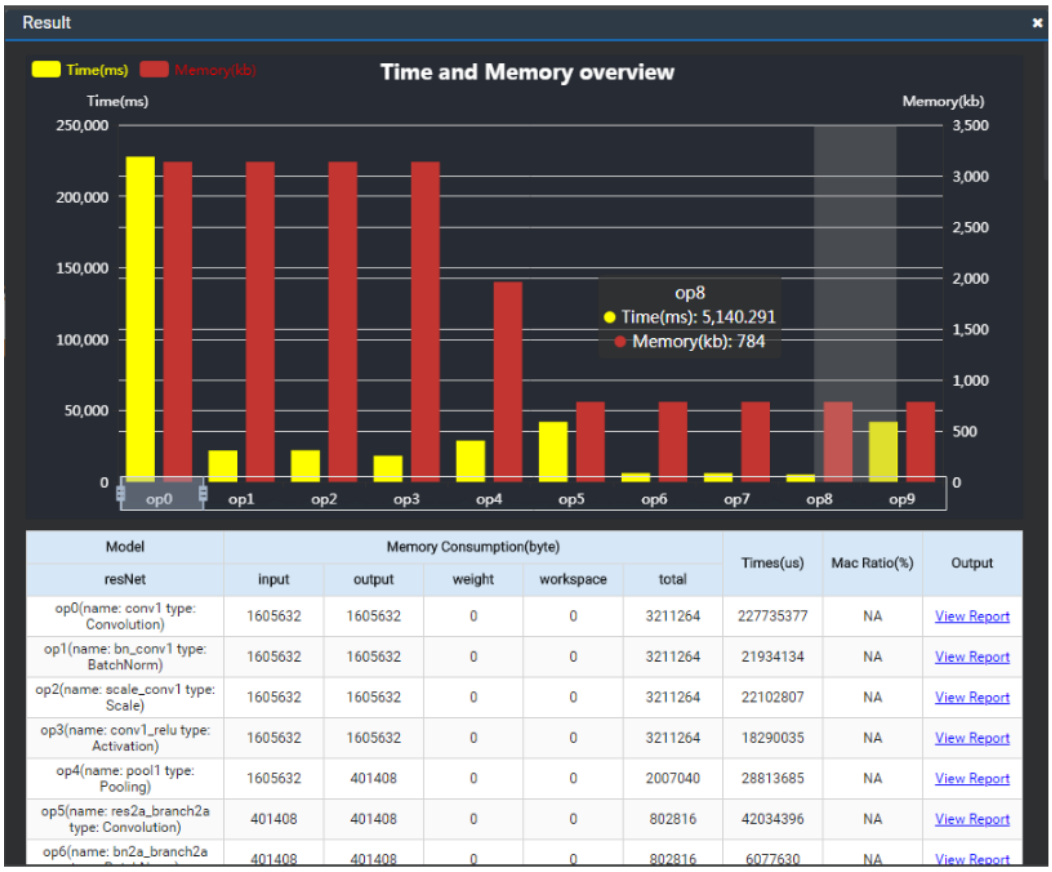
**说明**

该结果仅为仿真环境下编译完成后执行Profiling的结果展示，Atlas DK或者ASIC环境下的Profiling请参见《Mind Studio开发辅助工具》中的Profiling章节。

该结果为网络运行过程中的profiling数据，主要含有以下内容：

**图 3-26** Profiling Result




| Model | Memory Consumption(byte) | | | | | Times(us) | Mac Ratio(%) | Output |
|---|---|---|---|---|---|---|---|---|
| resNet | input | output | weight | workspace | total | | | |
| op0(name: conv1 type: Convolution) | 1605632 | 1605632 | 0 | 0 | 3211264 | 227735377 | NA | View Report |
| op1(name: bn_conv1 type: BatchNorm) | 1605632 | 1605632 | 0 | 0 | 3211264 | 21934134 | NA | View Report |
| op2(name: scale_conv1 type: Scale) | 1605632 | 1605632 | 0 | 0 | 3211264 | 22102807 | NA | View Report |
| op3(name: conv1_relu type: Activation) | 1605632 | 1605632 | 0 | 0 | 3211264 | 18290035 | NA | View Report |
| op4(name: pool1 type: Pooling) | 1605632 | 401408 | 0 | 0 | 2007040 | 28813685 | NA | View Report |
| op5(name: res2a_branch2a type: Convolution) | 401408 | 401408 | 0 | 0 | 802816 | 42034396 | NA | View Report |
| op6(name: bn2a_branch2a | 401408 | 401408 | 0 | 0 | 802816 | 6077630 | NA | View Report |

直方图上主要显示模型每层运行时间和内存占用。直方图下的表中的内容如表3-4所示。

**表 3-4** 直方图表格说明

| resNet | 模型层次名称 |
|---|---|
| input | 输入数据占用内存大小。 |
| output | 输出数据占用内存大小。 |
| weight | 权重。 |
| workspace | 算子计算过程中的临时使用的内存大小。 |
| total | 总占用内存大小。 |
| Time | 运行时间。 |
| Mac Ratio | mac利用率。 |

点击“View Report”即可下载相对应层的**output**数据，如图3-27所示。

**图 3-27** View Report


## 3.3.2 检测网络操作流程

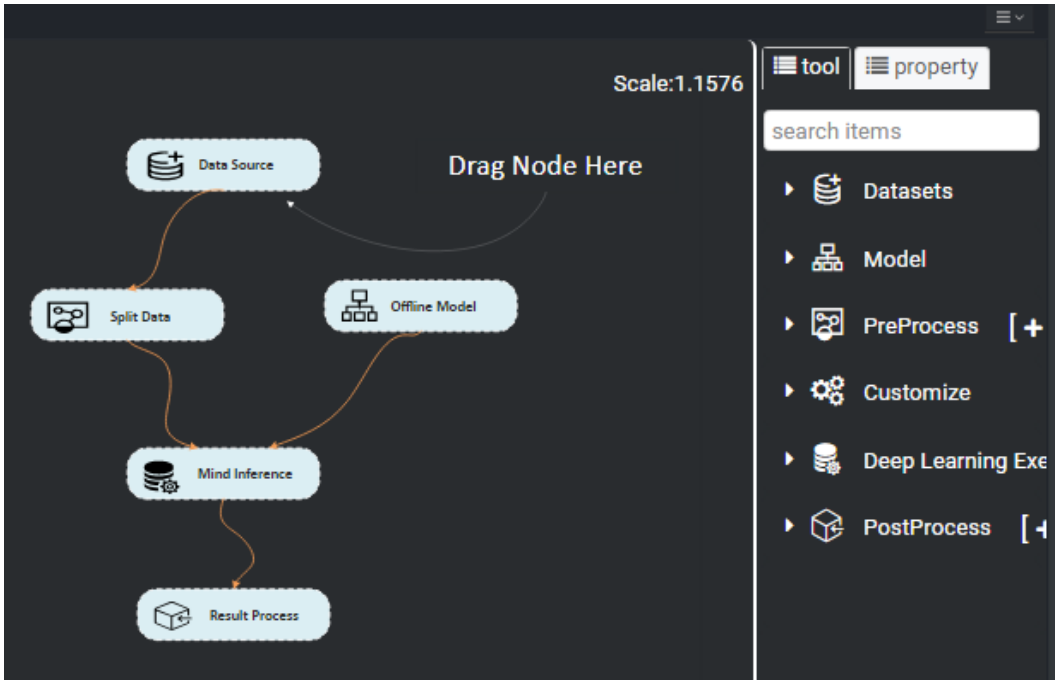
### 3.3.2.1 创建 Mind 工程

请参见3.3.1.1 创建Mind工程。

### 3.3.2.2 编排流程

您可以通过双击新创建的后缀名为**.mind的**文件（例如*Demo.mind*）即可打开Engine流程编排窗口，如图3-28所示。

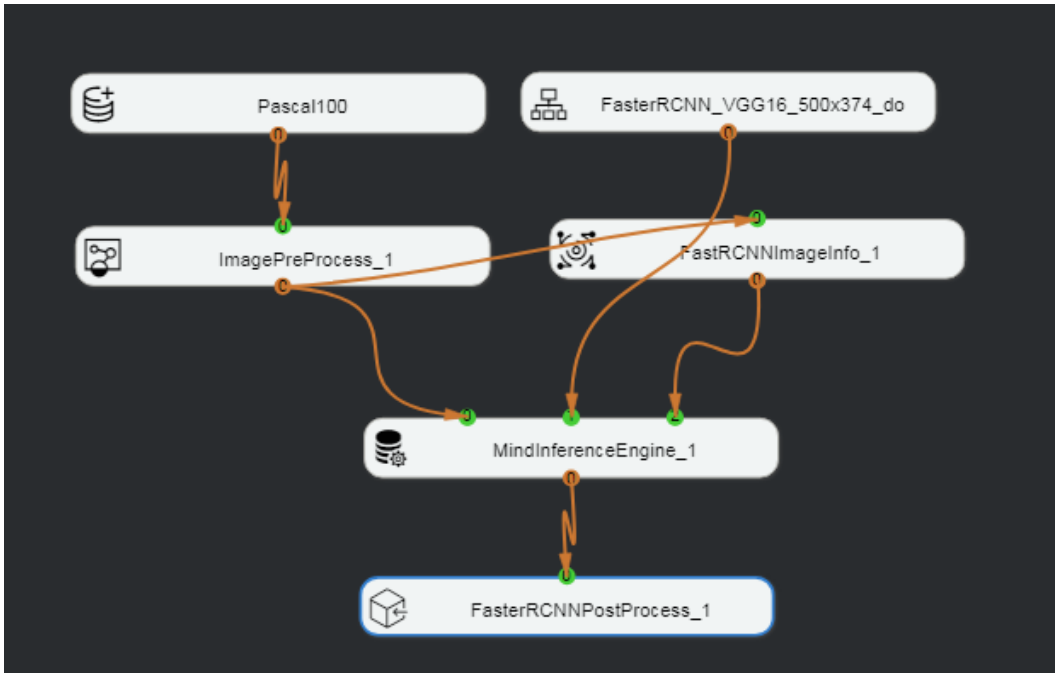
**图 3-28** 流程编排界面




您可以对图中的节点进行如下的操作：放置节点、删除节点、复制节点、设置节点的属性、保存节点、新增节点，详细操作可参考**3.7.2 节点基本操作**。

**FasterRCNN**网络如**图3-29**所示。

**图 3-29** FasterRCNN 网络示例



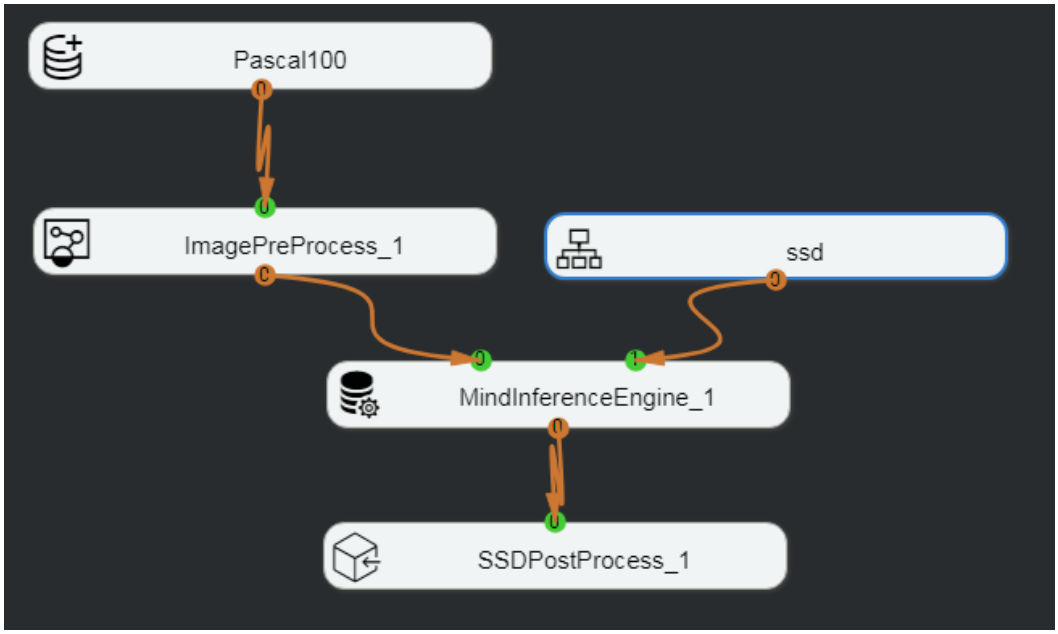

**FasterRCNN**网络主要包含如下节点：一个数据集（Pascal100）、一个模型（FasterRCNN）、一个数据预处理（ImagePreProcess）、一个模型图片信息

（FastRCNNImageInfo）、一个执行引擎（MindInferenceEngine）以及一个图片后处理节点（FasterRCNNPostProcess）。

**SSD**网络如图3-30所示。

**图 3-30** SSD 网络示例




**SSD**网络主要包含如下节点：一个数据集（Pascal100）、一个模型（SSD）、一个数据预处理（ImagePreProcess）、一个执行引擎（MindInferenceEngine）以及一个图片后处理节点（SSDPostProcess）。

## 前提条件

用户使用FasterRCNN和SSD进行流程编排时，如果采用的后处理节点为FasterRCNNPostProcess和SSDPostProcess，在导入caffe模型文件时（如faster-rcnn_resent18.prototxt），需要在该模型文件中的最后一层加入如下算子，然后再模型导入，流程编排才能运行。如果模型文件中已经存在如下算子，则请忽略该部分信息。

FasterRCNN模型文件最后一层补充内容：

```
layer {
 name: "detection_out"         #算子名称
 type: "FSRDetectionOutput"    #算子类型
 bottom: "cls_prob"            #输入分数
 bottom: "bbox_pred"           #预测的修正坐标
 bottom: "rois"                #原始featuremap 上产生的坐标框
 top: "out_box_num"            #输出有效框的个数
 top: "detection_out"          #输出有效框的坐标
 detection_output_param {
 num_classes: 21                #分类类别个数包括背景
 nms_threshold: 0.3             #nms（非最大值抑制） 阈值
 confidence_threshold: 0.8      #过滤框的阈值
 }
 }
```

SSD模型文件最后一层补充内容：

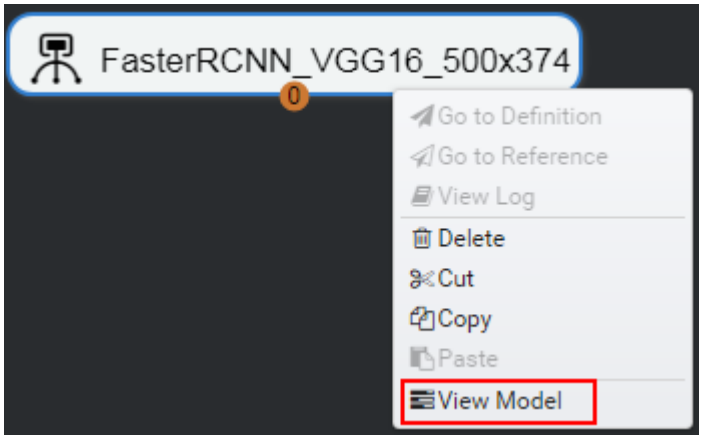
  

```
layer {
 name: "detection_out"       #算子名
 type: "SSDDetectionOutput"  #算子类型
 bottom: "mbox_loc" //loc 坐标输入
 bottom: "mbox_conf_flatten"  #类别分数输入
 bottom: "mbox_priorbox"      #原始featuremap 上产生的坐标框
 top: "detection_out"         #算子输出名称
 include {
 phase: TEST
 }
 detection_output_param {
 num_classes: 21              #分类类别包括背景
 share_location: true         #所有类别共享框
 background_label_id: 0       #背景类别id
 nms_param {
 nms_threshold: 0.45          #nms阈值
 top_k: 400                   #nms后取的框个数
 }
 save_output_param {
 label_map_file: "data/VOC0712/labelmap_voc.prototxt"
 }
 code_type: CENTER_SIZE       #坐标框修正方式
 keep_top_k: 200              #最后输出框个数
 confidence_threshold: 0.3    #过滤框阈值
 }
 }
```

下面以FasterRCNN模型文件为例进行简单说明：

单击My Models右侧的  ，添加FasterRCNN自定义模型组件，导入模型之后，将该模型拖入画布，右击模型选择“View Model”，如图3-31所示。

**图 3-31 查看模型网络结构**




弹出图3-32所示网络结构，原始模型网络结构的最后一层网络分别为预测层（bbox_pred）和类别的预测层（cls_prob），如果不加入detection_out算子，不能直接进行后处理，采用FasterRCNNPostProcess进行带后处理的流程编排时，会执行失败。

**图 3-32** 原始模型网络结构

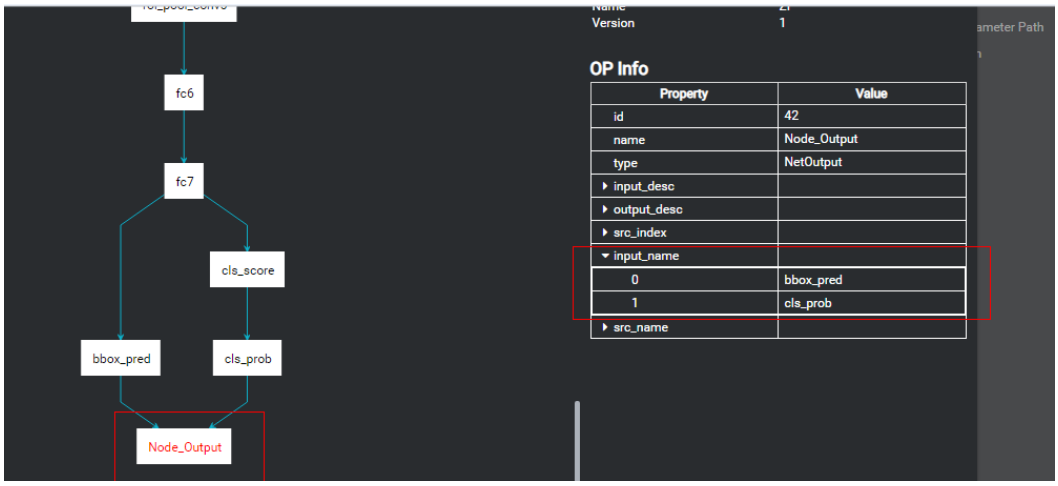



图3-33为加入上述detection_out算子后的模型网络结构，当原始模型网络结构在最后一层增加detection_out算子后，可直接进行带后处理（FasterRCNNPostProcess）的流程编排。

**图 3-33** 加入 detection_out 算子后的模型网络结构


faster-rcnn_prototxt.zip附件为不带detection_out算子和带有detection_out算子的模型文件，仅供参考。

## 操作步骤

请参见3.1.2-流程编排

**步骤1** 请参见**步骤1**。

**步骤2** 请参见**步骤2**。

**步骤3** 将所需节点放置到对应位置。放置节点方法请参见**步骤3**。

如下**表3-5**、**表3-6**为FasterRCNN网络和SSD网络所需节点信息。

**表 3-5** FasterRCNN 网络所需节点信息

| 输入 | 来源 | 备注 |
| --- | --- | --- |
| 数据集 | Datasets > BuiltIn Datasets > Pascal100 | - |
| 模型 | Model > My Models > FasterRCNN | FasterRCNN模型由用户导入，导入方法请参见《Mind Studio基本操作》中的“模型管理 > Mind Engine的模型管理 > 新增自定义模型组件”章节 |
| 数据预处理 | Preprocess > ImagePreProcess | 将该节点属性中“resize_width”与“resize_height”分别设置为500和374。<br>**说明**<br>**Target**为**ASIC**或者**Atlas DK**，**PreProcess**类型需选择**ImagePreProcess**节点。 |
| 模型图片信息 | Customize > FastRCNNImageInfo | - |
| 执行引擎 | Deep Learning Execute Engine > MindInferenceEngine | - |
| 图片后处理节点 | PostProcess > FasterRCNNPostProcess | - |

**表 3-6** SSD 网络所需节点信息

| 输入 | 来源 | 备注 |
| --- | --- | --- |
| 数据集 | Datasets > BuiltIn Datasets > Pascal100 | - |
| 模型 | Model > My Models > SSD | SSD模型由用户导入，导入方法请参见《Mind Studio基本操作》中的“模型管理 > Mind Engine的模型管理 > 新增自定义模型组件”章节 |
| 数据预处理 | Preprocess > ImagePreProcess | 将该节点属性中“resize_width”与“resize_height”分别设置为300和300。<br>**说明**<br>**Target**为**ASIC**或者**Atlas DK**，**PreProcess**类型需选择**ImagePreProcess**节点。 |
| 执行引擎 | Deep Learning Execute Engine > MindInferenceEngine | - |
| 图片后处理节点 | PostProcess > SSDPostProcess | - |

**步骤4** 建立节点连线关系。

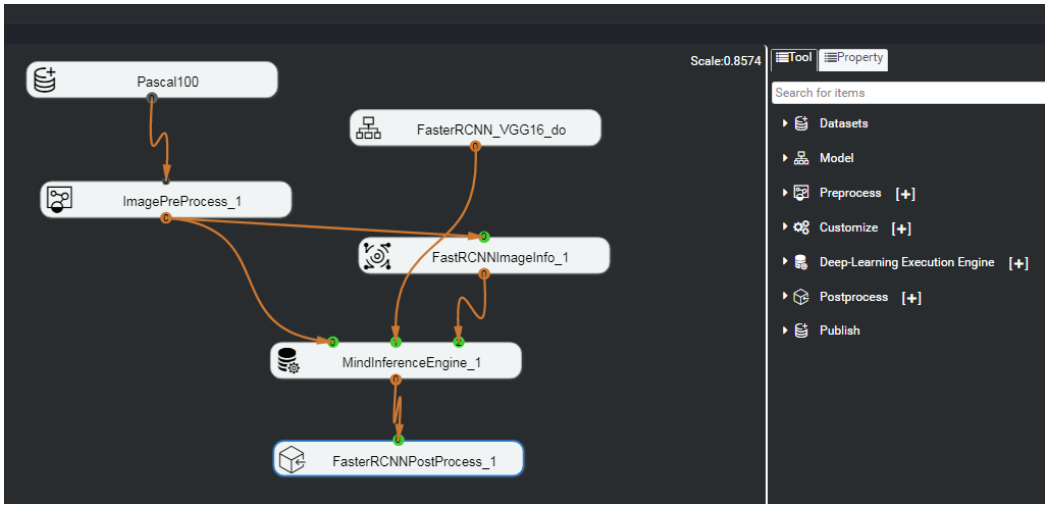
  

在完成所需节点的放置与属性设置后，需要建立其相应的连接关系。

橘黄色的圆形端点为输出端点，可以从该点引出连线，绿色端点为输入端点，可以放置连线。

FasterRCNN最终节点连线关系示例如图3-34所示。

**图 3-34** FasterRCNN 网络建立节点连线关系



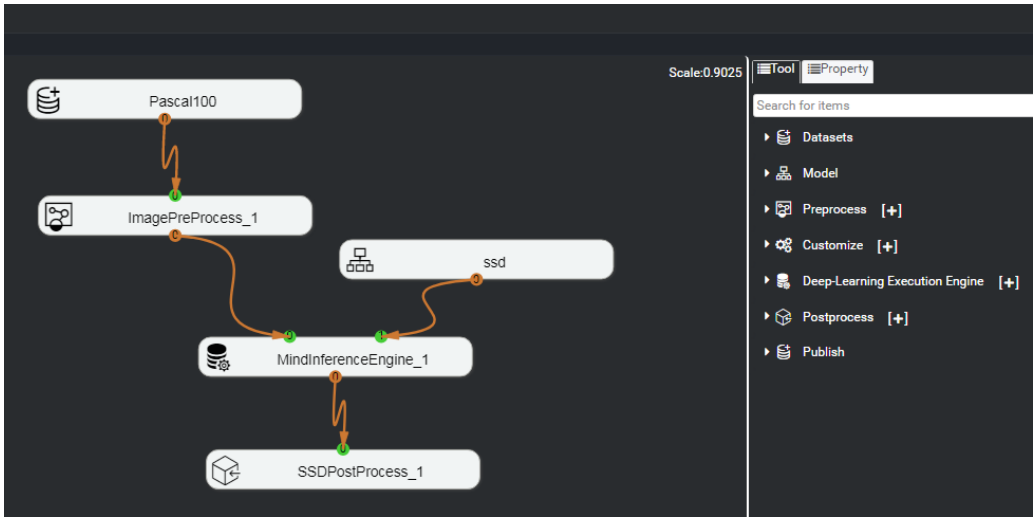

**说明**

建立连接关系时需注意以下两点：

1. Deep Learning Execute Engine类型节点属性设置中，Input Count 设置为3。
2. PreProcess类型的节点需连接到Deep Learning Execute Engine类型节点的0号输入端点上。
3. FasterRCNNImageInfo节点需连接到Deep Learning Execute Engine类型节点的2号输入端点上。
4. Model类型的节点需连接到Deep Learning Execute Engine类型节点的1号输入端点上。

SSD最终节点连线关系示例如图3-35所示。

**图 3-35** SSD 网络建立节点连线关系



注意

建立连接关系时需注意以下两点：

1. PreProcess类型的节点需连接到Deep Learning Execute Engine类型节点的0号输入端点上。
2. Model类型的节点需连接到Deep Learning Execute Engine类型节点的1号输入端点上。

**步骤5** 单击画布下侧的“Save”。

保存编排的流程。

**----结束**

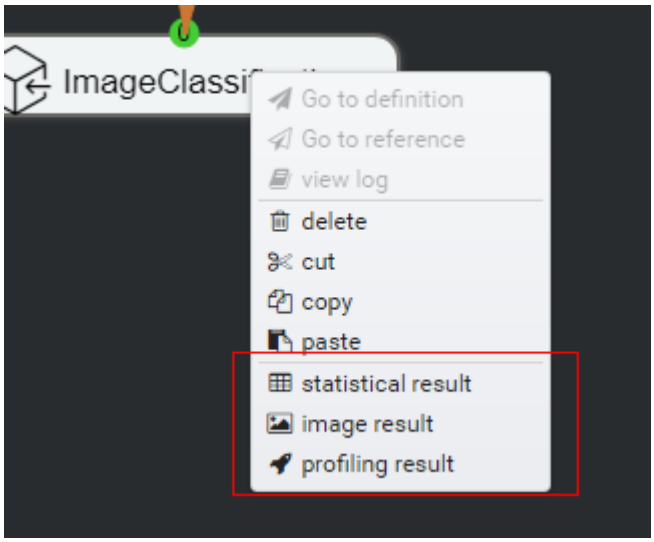
### 3.3.2.3 编译运行

请参见3.3.1.3 编译运行。

### 3.3.2.4 运行结果查看

运行完成之后，将会在工程根目录下的out文件夹下生成以时间戳命名的结果文件夹，此时即可在**PostProcess**类型节点上右键弹出菜单来查看相应的结果。

**图 3-36 右键菜单**




共有如下三种类型的结果：image result，statistical result和profiling result。

## image result

image result展示推理图片的预测结果，对于检测网络的image result结果显示，结果图片带有bbox的图形框和置信度值。如图3-37所示。

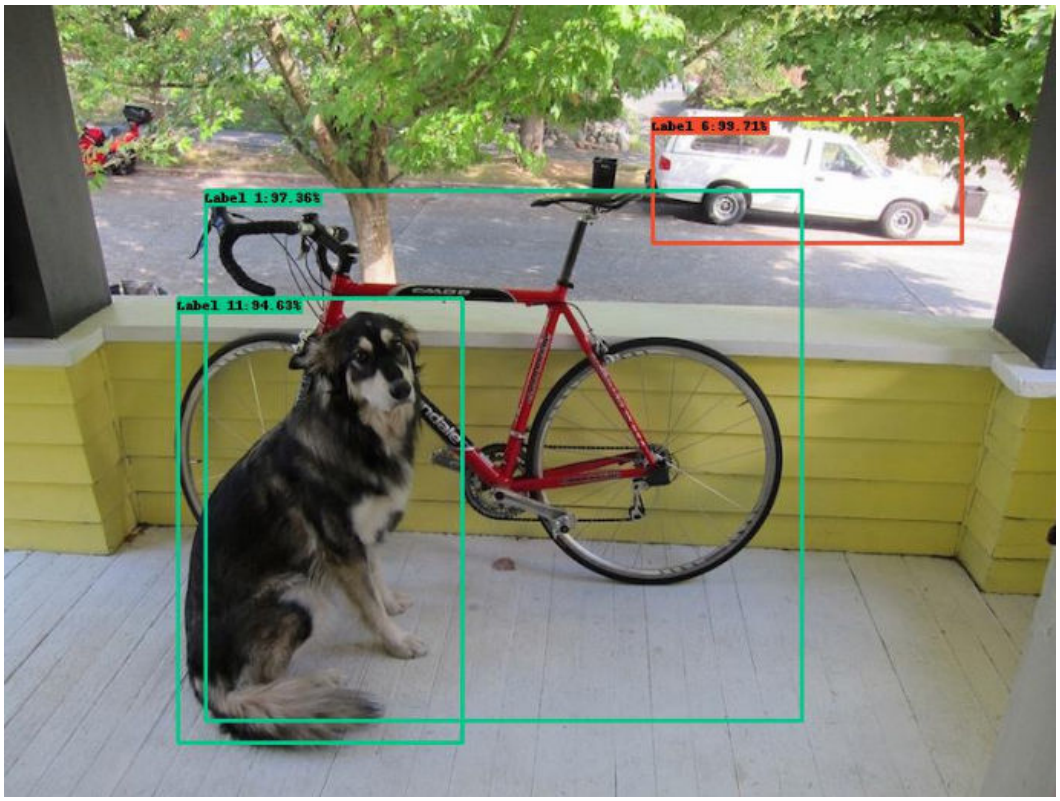
**图 3-37** 检测网络 image result 示例 1


**说明**

1. 检测网络图片上的标签代表该图片的类别以及推理出的置信度值。
2. 检测网络图片上的矩形框为推理出的目标物体所在位置。

- 如果导入数据集时未导入标签字典文件或者标签字典文件与模型不配套，则只显示标签编号，如图3-38所示。

**图 3-38** 图片分类示例 2




**说明**

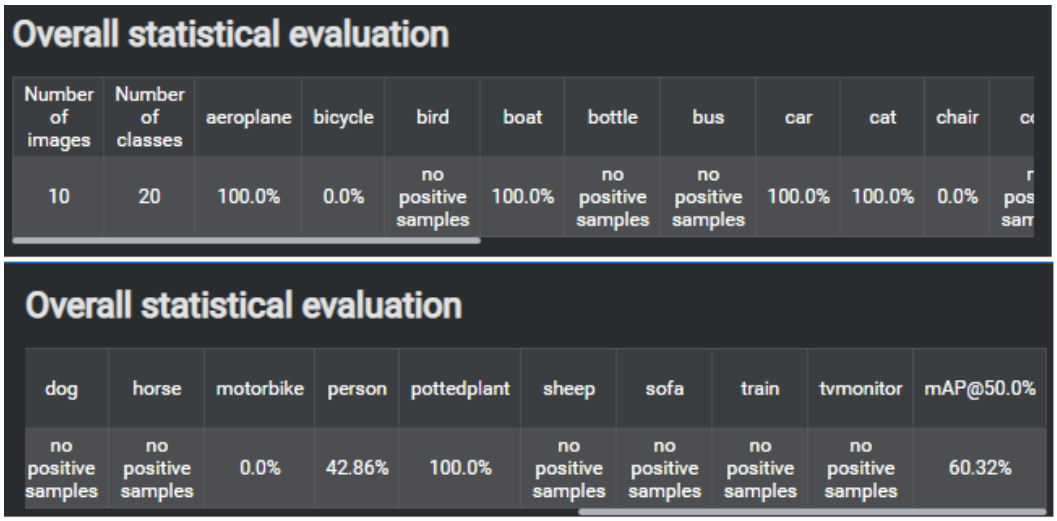
图中Label***（如Label 1）表示图片的预测标签编号，百分比（如37.36%）表示预测图片为该分类的一种概率。

### statistical result

statistical形式的结果为一张表格，列出了所进行推理图片的结果数量，该类数据所含的类别（比如Pascal默认类别数为20），以及每个类别的置信度和总的预测mAP值，如图3-39所示。

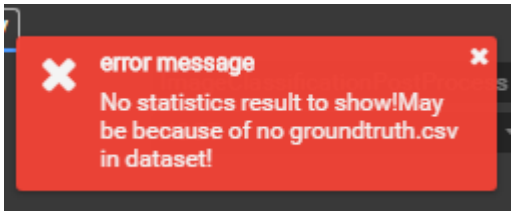
**图 3-39** Statistical Result




**说明**

要查看**statistical**，所进行推理的图片数据必须在标签文件中，否则会提示如图3-40所示错误信息。

**图 3-40** 错误信息




### profiling result

请参见**profiling result**。

# 3.4 （扩展）没有预处理的 Engine 编排

## 3.4.1 简介

为消除预处理对结果的影响，Engine编排时，不加入PreProcess节点，称之为没有预处理的Engine编排。

### 前提条件

本章节是对**3.3 操作流程**的扩展，阅读本章节前请先了解**3.3 操作流程**功能。

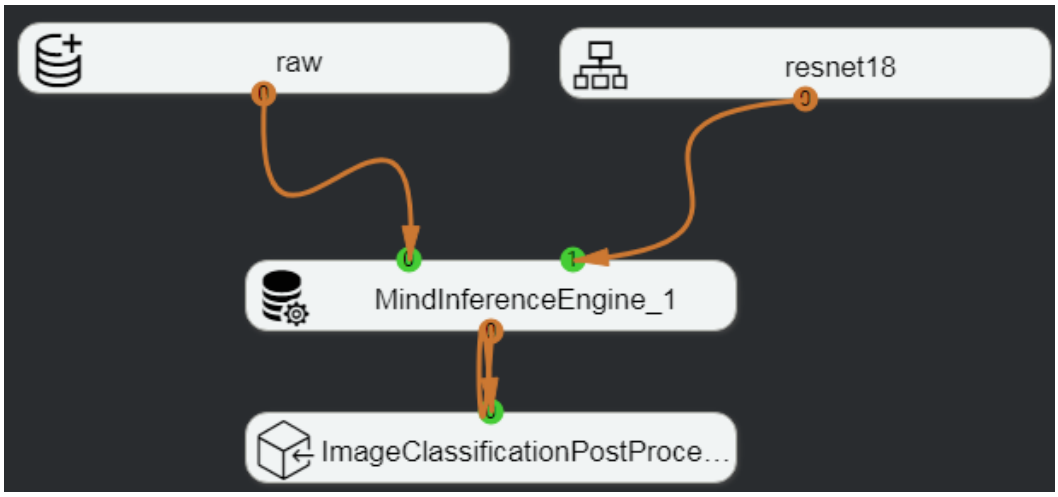
  

## 功能说明

以Resnet18网络为例，没有预处理编排主要包含如下节点：

一个数据集、一个模型、一个执行引擎以及一个图片后处理节点，如图3-41所示。

**图 3-41** 没有预处理的 Engine 编排示例




# 3.4.2 编排流程

## 背景信息

- 本节中所涉及Engine为MindInferenceEngine。
- 网络模型为resnet18，可以使用工具内置Resnet18网络或者自定义导入，自定义导入模型请参见《Mind Studio基本操作》中的新增自定义模型组件。
- 编排的网络结构为没有预处理节点下的编排。

编排流程及注意点如表3-7所示。

**表 3-7** 无预处理节点下的 Engine 编排流程

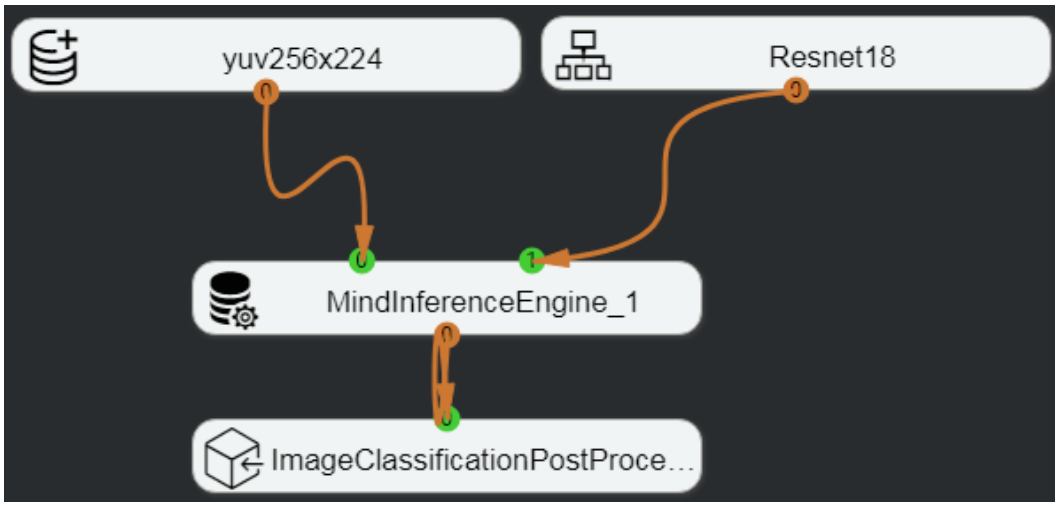
| 步骤 | 硬件环境 |
| --- | --- |
| 创建Mind工程 | Target选择：ASIC/Atlas DK |
| 编辑网络结构 | 支持如下两种数据集：<br>● Raw格式：为BGR float格式的一种，导入数据集时选择Raw类型的数据集。模型必须为自定义的网络模型，且不带aipp文件，即导入网络模型时需要将“Optional Options”，“Input Image Process”设置为“Off”。<br>● nv12格式：nv12为YUV420平面格式的一种，导入数据集时选择Image类型的数据集。<br>**说明**<br>nv12格式的数据集宽和高必须为256x224。 |
| 编译运行 | 请参见3.3.1.3 编译运行。 |

| 步骤 | 硬件环境 |
| --- | --- |
| 运行结果查看 | 请参见3.3.1.4 运行结果查看。 |

编排的网络结构示例如图3-42所示。

**图 3-42** EVB 环境下无预处理的 Engine 编排示例




# 3.5 （扩展）多网络串联 Engine 编排

## 3.5.1 简介

流程编排是一个有向Graph，相关概念如下所示：

- 节点（node）

  为graph的顶点（vertex）。如图3-43中的顶点：0、1、2、3、4、5、6、7、8。
- 枝（branch）

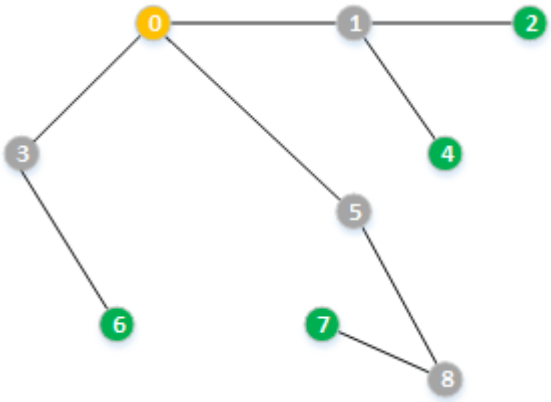
  为graph的边（edge）。如图3-43中的0->1、0->3、0->5、1->2、1->4、3->6、5->8、8->7。
- 根（root）

  一棵树可以想象成从某一个顶点开始进行分枝，则这个顶点就是根。一棵树的每一个节点都可以作为根。如图3-43中可以将节点0作为根。
- 叶（leaf）

  在一棵树上选定根后，如节点0作为根。由根开始不断分枝，途中所有无法再分枝的节点成为叶。如图3-43所示，根为节点0，则节点2、4、6、7为叶。

**图 3-43** Gragh 示意图




## 前提条件

本章节是对**3.3 操作流程**的扩展，阅读本章节前请先了解**3.3 操作流程**功能。

## 功能说明

本文中前面章节介绍的流程编排只是一个网络对图片进行处理，实际应用中，可能存在对于一张图片先通过一个检测网络将图片中的一些关键元素框出来，再针对框出的元素使用其他网络（可以是分类网络，也可以是另外的检测网络）进行更精细的处理。

- 单个网络的情况，网络的终点为后处理节点。
- 多个网络的情况，中间网络的后处理节点的输出数据要作为下个网络的输入数据。

  在目前支持的后处理节点中，只有SSDPostProcess和FasterRCNNPostProcess节点可以直接配置输出端口以及输出条件，将这两个检测网络后处理的结果以及原始数据集发给下面的网络，做进一步处理。

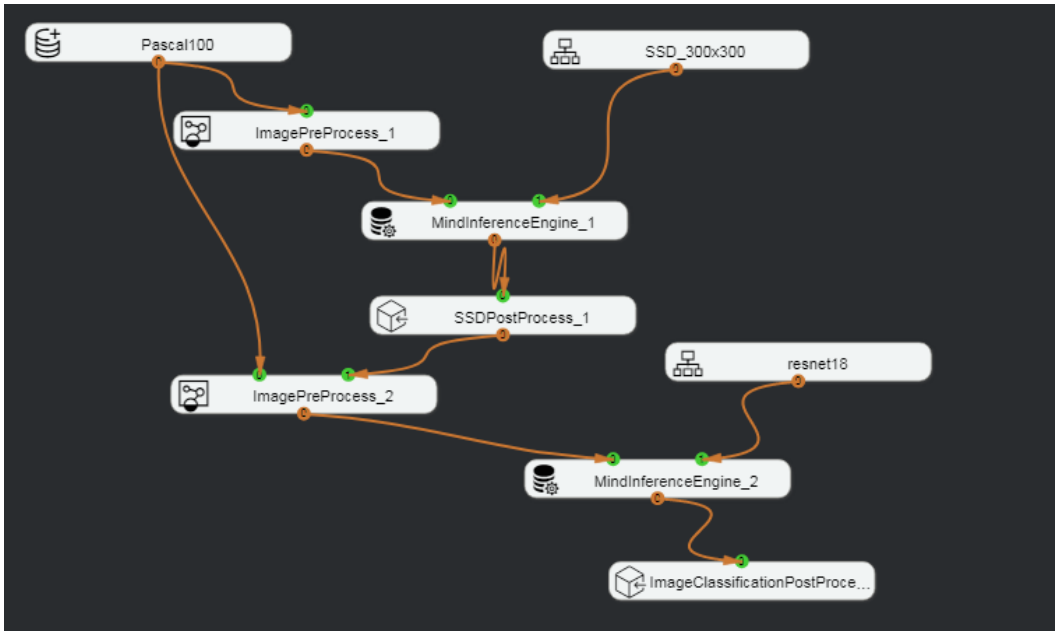
**说明**

目前只支持SSDPostProcess和FasterRCNNPostProcess检测网络配置输出端口，以及输出数据条件。

如图3-44所示，为一个简单的多网络串联示例。

**图 3-44** 多网络串联 Engine 编排示例




## 3.5.2 编排流程

### 前提条件

操作前请先准备好SSD或FasterRCNN检测网络模型，并将模型文件导入My Models中。

**说明**

“新增自定义模型组件”详细操作请参见《Mind Studio基本操作》手册。

编排流程及注意点如表3-8所示。

**表 3-8** 多网络串联下的 Engine 编排流程

| 步骤 | 硬件环境 |
| --- | --- |
| 创建Mind工程 | Target选择：ASIC/Atlas DK |

| 步骤 | 硬件环境 |
| --- | --- |
| 编辑网络结构 | ● Datasets：本示例选择Pascal内置数据集。<br>**说明**<br>多网络串联场景不支持Raw类型的数据集。<br>● Model：第一个网络只支持My Models下的SSD或FasterRCNN检测网络模型，本示例选择SSD检测网络模型。<br>● 预处理节点：只支持ImagePreProcess。<br>**说明**<br>使用SSD网络模型，请将第一个网络预处理节点的“resize_width”和“resize_height”取值设置为“300”。<br>● 推理引擎：只支持MindInferenceEngine。<br>● 第一个网络的后处理节点只能选择SSDPostProcess或FasterRCNNPostProcess。详细配置请参见**3.5.2.1 检测网络后处理输出配置**、**3.5.2.2 检测网络后处理与下个网络前处理的连接**。 |
| 编译运行 | 编译前需要请先参见**3.5.2.3 编译前对graph校验**对graph进行校验，校验通过后再请参见**3.3.1.3 编译运行**。 |
| 运行结果查看 | 请参见**3.3.1.4 运行结果查看**。 |

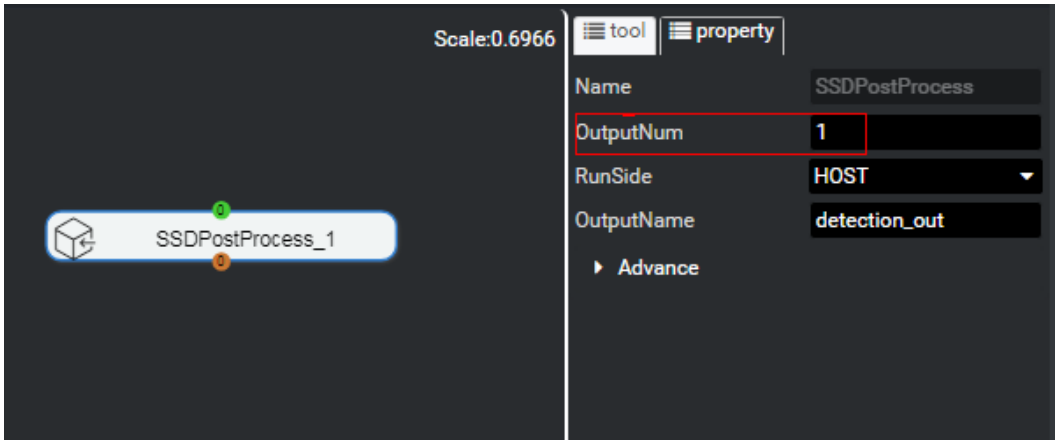
## 3.5.2.1 检测网络后处理输出配置

完成Engine编排后，需要对输出端口以及端口输出条件进行配置。

### 输出端口数量配置

将SSDPostProcess或者FasterRCNNPostProcess后处理节点拖到画布中，在画布中点击该节点，展示该节点的属性，然后将“OutputNum”修改为1（范围0-15），如**图3-45**。如果“OutputNum” ＞0，则检测网络后处理节点下方会展示出对应数量的输出端口，供该节点和下面的节点连接。

**图 3-45** 检测网络后处理 OutputNum 设置

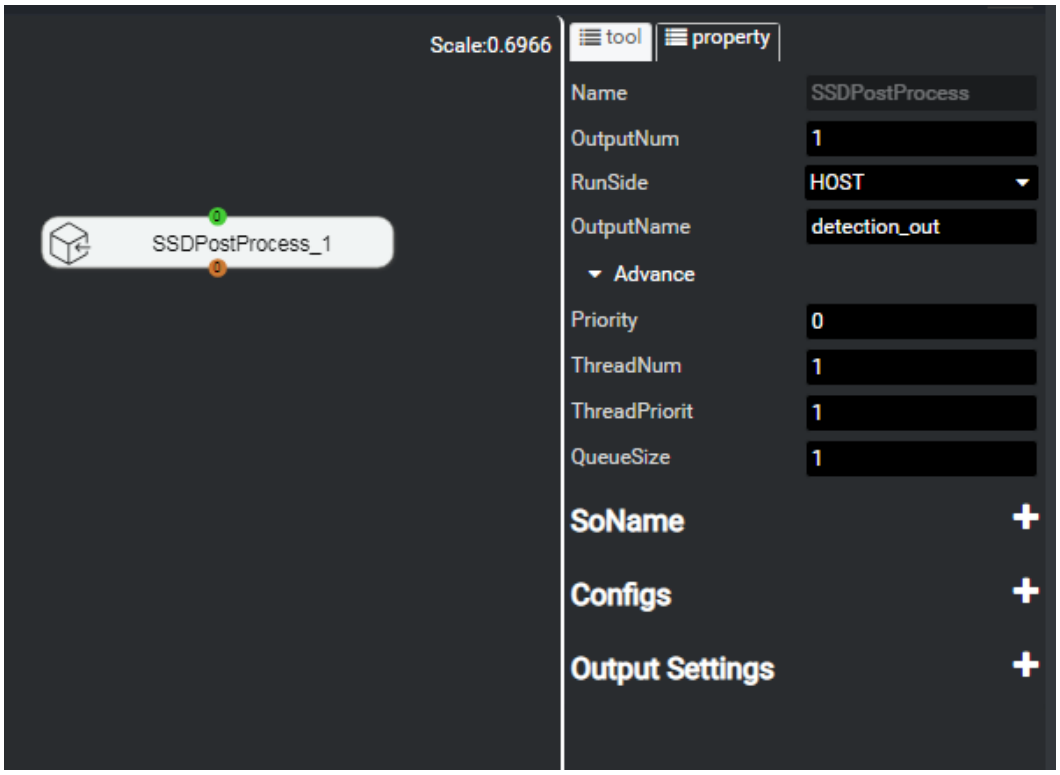


## 输出端口筛选条件配置

- 如果“OutputNum” ＞0，展开“Advance”下拉菜单，可以看到“Output Settings”配置，如图3-46。

**图 3-46 属性设置**




**说明**

多网络串联场景下，SSDPostProcess或者FasterRCNNPostProcess后处理节点中“ThreadNum”参数必须配置为“1”，即不支持多线程。

其他涉及节点：MindInferenceEngine节点中“ThreadNum”参数也必须配置为1。

- 点击“Output Settings”后面的，弹出如图3-47所示配置框。界面参数解释如表3-9所示。

**图 3-47 输出端口筛选条件设置**

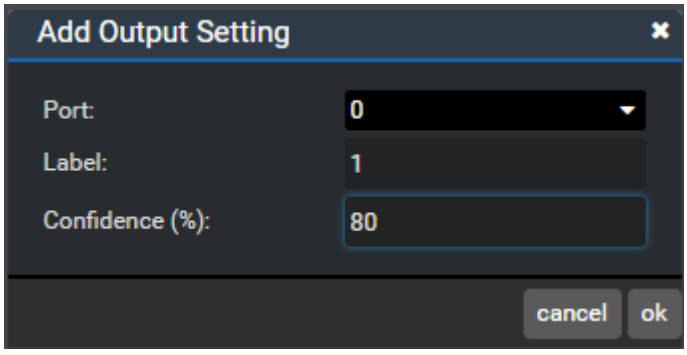


**表 3-9** 输出界面配置信息

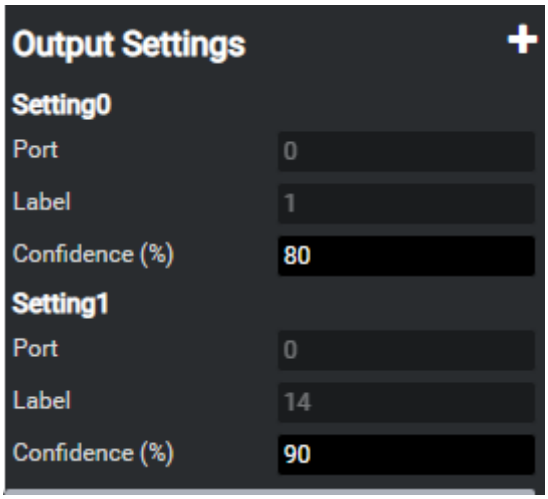
| 参数 | 说明 |
| --- | --- |
| Port | 端口号，因为SSDPostProcess只设置了一个输出端口，所以本例中端口号为0。 |
| Label | 筛选的种类标签，取值为数字。<br>本例中设置为1。 |
| Confidence | 置信度，取值范围为[0,100]，本例设置为80，表示该端口只筛选标签为1的置信度大于80%的图片，输出给下个节点。 |

**说明**

如果某个端口不配置筛选条件，则检测网络处理的所有结果都会输出给下个端口。

- 可以设置多个筛选条件，设置后都展示在Output Settings区域下面，如图3-48，可以在此快捷修改置信度。

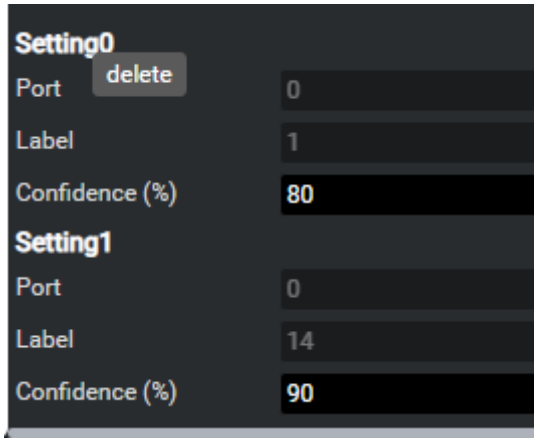
**图 3-48** 筛选条件展示




若您想要删除某条筛选条件，则右键点击“Setting0”，弹出delete按钮，点击“delete”，即可删除，如图3-49所示。

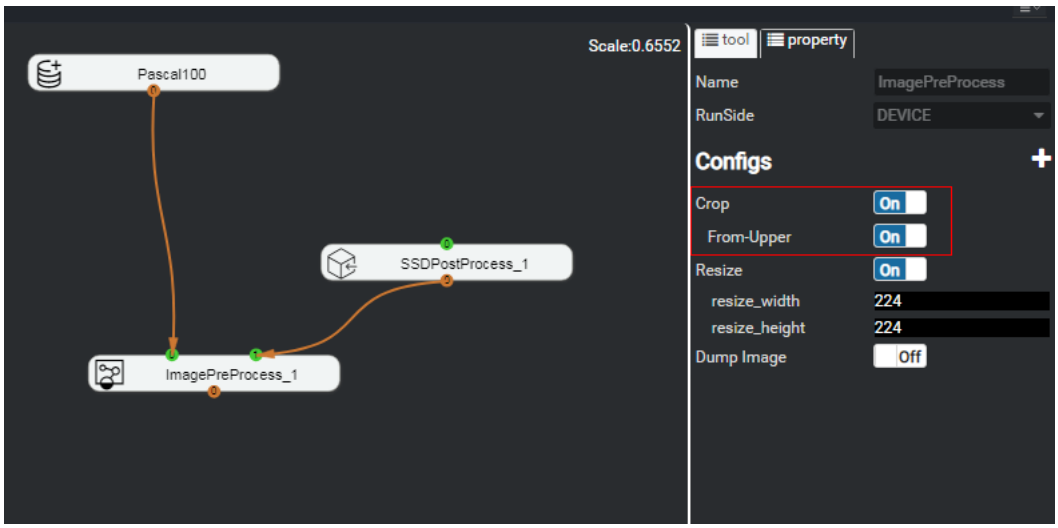
**图 3-49** 删除筛选条件




### 3.5.2.2 检测网络后处理与下个网络前处理的连接

将前处理节点拖入画布，点击该节点展示属性，将“Crop”开关打开，用于增加输入节点，然后再打开“From-Upper”开关，前处理节点的输入端口个数由1个变为2个：数据集连接端口号为0；后处理的输出端口连接前处理节点的端口号为1。如图3-50所示。

**图 3-50** 前处理输入配置




### 3.5.2.3 编译前对 graph 校验

连接好流程编排所需节点，单击“Generate”，会对该流程编排graph做以下校验：

- 如果graph中只有节点没有连线，提示graph中没有连接，不能编译。
- 在同一个graph中如果存在多个推理引擎，不能既有运行在DEVICE侧的，又有运行在HOST侧的。

  **说明**

  推理引擎的归属请参见3.7.1 业务节点介绍中的表3-12。
- 如果有节点没有连接，单独存在于graph之外，则不能编译。
- 叶节点必须运行在HOST侧（开发者板形态不做此校验），否则不能编译。

- 目前暂时不支持并行网络。

# 3.6 开源 Caffe 模型 Engine 编排

## 3.6.1 简介

本节介绍基于开源Caffe模型的Engine编排流程。

## 3.6.2 编排流程

### 前提条件

- 操作本章节前请先准备好Caffe框架的模型文件以及权重文件，并将这两个文件导入Caffe Models中。

  说明

  “新增Caffe模型组件”详细操作步骤请参见《Mind Studio基本操作》手册。
- 操作本章节前请先了解3.3 操作流程功能。

编排流程及注意点如表3-10所示。

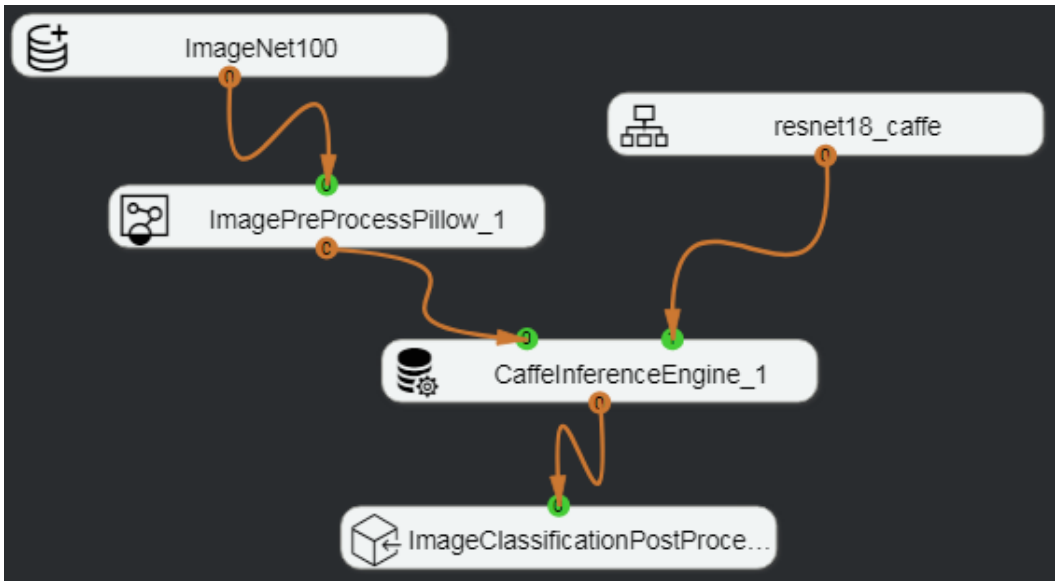
**表 3-10** Engine 编排流程

| 编排流程 | 仿真环境 |
| --- | --- |
| 创建Mind工程 | ● Mind Type：只支持DEFAULT。<br>● Target：只支持Local仿真环境。 |
| 编辑网络结构 | ● Model：只支持Caffe Models下新增的组件，本示例新增组件名称为：*resnet18_caffe*。<br>● 预处理节点：只支持ImagePreProcessPillow。<br>● 推理引擎：只支持CaffeInferenceEngine。 |
| 编译运行 | 请参见3.3.1.3 编译运行。 |
| 运行结果查看 | 请参见3.3.1.4 运行结果查看。 |

开源Caffe模型的Engine编排流程图如图3-51所示。

**图 3-51** 开源 Caffe 模型的 Engine 编排示例




## 3.6.3 （扩展）没有预处理的 Engine 编排

### 前提条件

- 操作本章节前请先准备好Caffe框架的模型文件以及权重文件，并将这两个文件导入Caffe Models中。

  **说明**

  “新增Caffe模型组件”详细操作步骤请参见《Mind Studio基本操作》手册。
- 操作本章节前请先了解**3.3 操作流程**功能。

开源Caffe模型没有预处理的Engine编排流程及注意点如**表3-11**所示。

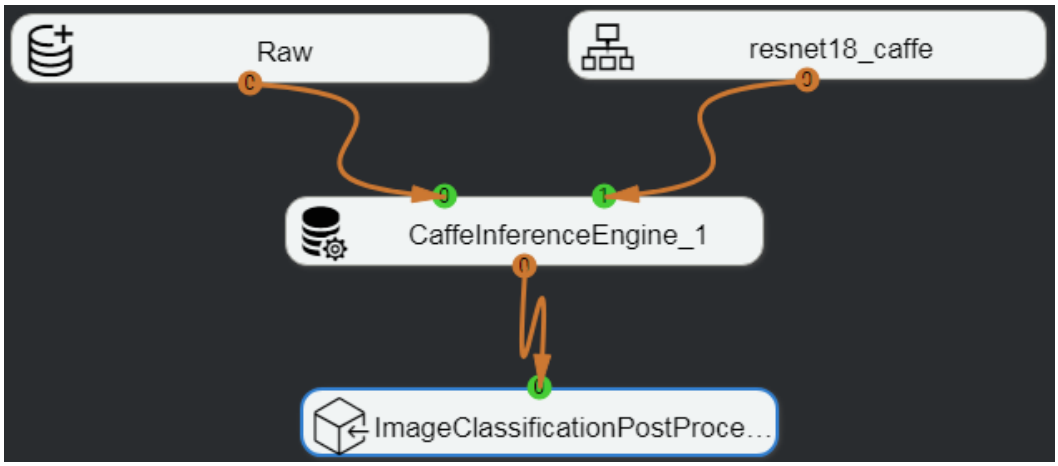
**表 3-11** 开源 Caffe 模型没有预处理的 Engine 编排流程

| 编排流程 | 仿真环境 |
| --- | --- |
| 创建Mind工程 | ● Mind Type：只支持DEFAULT。<br>● Target：只支持Local仿真环境。 |
| 编辑网络结构 | ● Datasets：只支持BGR格式的图片，即Raw类型的数据集。<br>● Model：只支持Caffe Models下新增的组件，本示例新增组件名称为：*resnet18_caffe*。<br>● 推理引擎：只支持CaffeInferenceEngine。 |
| 编译运行 | 请参见**3.3.1.3 编译运行**。 |
| 运行结果查看 | 请参见**3.3.1.4 运行结果查看**。 |

开源Caffe模型的Engine编排流程图如**图3-52**所示。

**图 3-52** 开源 Caffe 模型的 Engine 编排示例




# 3.7 操作参考

## 3.7.1 业务节点介绍

业务开发人员可以在Mind Studio上通过拖拽图形化业务节点、连接业务节点、编辑业务节点属性的方式编排和运行业务流程，实现业务流程编排“0”编码。

一个业务节点表示一次处理过程，例如**ImagePreProcess**节点可以对图片进行重新编辑，设定大小或者缩放。

业务节点间的连接表示节点间数据的流向，连接起点的输出作为连接终点的输入，业务节点的属性配置是运行该节点的所需的参数。

业务节点有以下类型：

- **Dataset**：数据集节点，用于指定网络输入数据。
- **Model**：模型节点，用于指定神经网络模型。
- **PreProcess**：数据预处理节点，用于对数据集中的数据进行预处理。
- **Customize**：数据输入节点。
- **Deep Learning Execute Engine**：神经网络执行节点，用于对网络进行运行。
- **PostProcess**：后处理节点，用于对网络执行结果进行后处理。

当前版本中提供的默认节点如下：

**表 3-12** 内置节点说明

| 节点名 | 功能 | 类型 |
| --- | --- | --- |
| MnistDataset | 预置的手写数字识别的数据集。 | Dataset |
| ImageNet100 | 预置ImageNet数据集，图片个数为100张，用于分类网络输入数据。 | Dataset |
| Pascal100 | 预置PASCAL数据集，图片个数为100张，用于检测网络输入数据。 | Dataset |

| 节点名 | 功能 | 类型 |
| --- | --- | --- |
| COCO100 | 预置COCO数据集，图片个数为100张，用于检测网络输入数据。 | Dataset |
| Resnet18 | 作用：分类网络模型，对图片进行分类处理。 | Model |
| ImagePreProcess | 作用：对图片数据进行预处理，将图片数据转换成nv12格式的yuv数据；<br>运行侧：device侧；<br>数据集：jpg/jpeg/png/nv12格式的图片数据；<br>场景：Target为ASIC或Atlas DK。 | PreProcess |
| ImagePreProcessPillow | 作用：对图片数据进行预处理，将图片数据转换成BGR格式的数据；<br>运行侧：Mind Studio安装服务器侧；<br>数据集：jpg/jpeg/png/nv12格式的图片数据；<br>场景：Target为Local。 | PreProcess |
| MindInferenceEngine | 作用：Mind推理引擎，对分类网络和检测网络进行推理；<br>运行侧：device侧；<br>模型：MindStudio转换后的分类网络或检测网络模型；<br>场景：Target为ASIC、Atlas DK。 | Deep Learning Execute Engine |
| CaffeInferenceEngine | 作用：开源caffe推理引擎；<br>运行侧：Mind Studio安装服务器侧；<br>模型：Caffe模型；<br>场景：Target为Local（前处理节点为ImageProcessPillow）；<br>不适用情况：不支持fasterrcnn网络的多batch（开源代码不支持）。 | Deep Learning Execute Engine |
| ImageClassificationPostProcess | 作用：分类网络后处理节点，分类网络推理结果进行解析，得到图片的推理类别和概率；<br>运行侧：host侧；<br>数据集：分类网络数据集（如Image与ImageNet）；<br>模型：分类网络模型（如Resnet18等）；<br>场景：Target为ASIC、Atlas DK或Local。 | PostProcess |

| 节点名 | 功能 | 类型 |
| --- | --- | --- |
| FasterRCNNPostProcess | 作用：对FasterRCNN网络输出结果进行分析，得到图片的检测结果；<br>运行侧：host侧；<br>数据集：推荐使用检测网络数据集（如PASCAL或COCO），若使用数据集Image与ImageNet中的jpg类型的图片，只支持图形框检测，无法进行标注判断结果。<br>模型：FasterRcnn网络模型；<br>场景：Target为ASIC、Atlas DK或Local。 | PostProcess |
| SSDPostProcess | 作用：对SSD网络输出结果进行分析，得到图片的检测结果；<br>运行侧：host侧；<br>数据集：检测网络数据集（如PASCAL或COCO等）；<br>模型：SSD网络网络模型；<br>场景：Target为ASIC、Atlas DK或Local。 | PostProcess |
| SaveFilePostProcess | 作用：后处理节点，把推理的结果写到文件中，以供用户进行其他处理；<br>运行侧：host侧；<br>数据集：jpg/jpeg/png/nv12/bin格式的图片数据；<br>模型：任意网络模型；<br>场景：Target为ASIC、Atlas DK或Local。 | PostProcess |
| FastRCNNImageInfo | 作用：指定网络输入的图片高、宽、缩放比率等信息（仅用于FasterRcnn网络，作为MindInferenceEngine的第三个输入）；<br>运行侧：device侧；<br>数据集：检测网络数据集（如PASCAL或COCO等）；<br>模型：FasterRcnn网络模型；<br>场景：Target为ASIC、Atlas DK或Local。 | Custmoize |

**说明**

dvpp预处理完成后的结果是yuv420sp，也就是nv12格式的（默认值格式），其中宽128字节对齐，高16字节对齐。

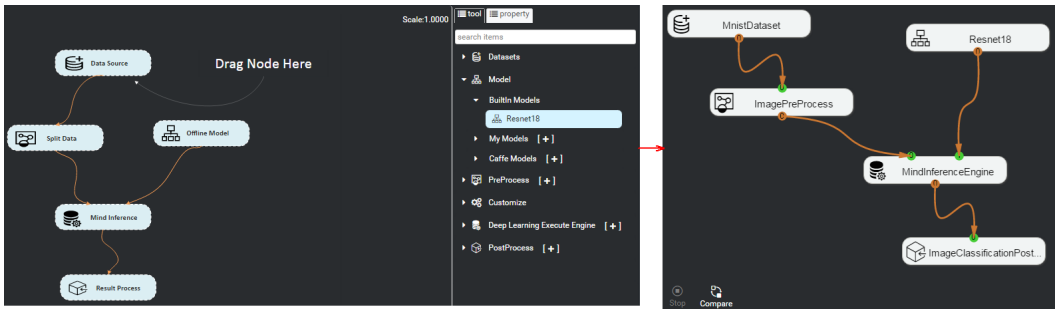
仿真的预处理完成后，默认格式为BGR，按照参数crop、resize、scale对齐。

## 3.7.2 节点基本操作

### 3.7.2.1 放置节点

在Engine流程编排窗口的右边区域框内，**tool**工具栏中显示各节点类型，单击节点类型前面的三角箭头展示各节点，按住鼠标左键即可将其拖拽到左侧绘图区域框内，如图3-53所示，放置后的节点可在绘图区域框内任意拖动。

**图 3-53 放置节点**




**tool**工具栏中节点说明，请参见3.7.1 业务节点介绍。

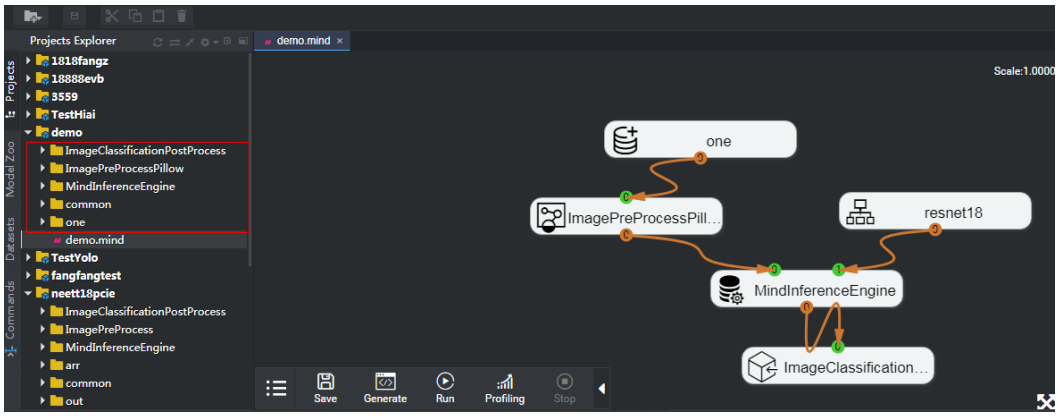
---

**注意**

1. MindEngine流程一般以Datasets节点作为起始节点，以PostProcess节点作为结束节点，连接方向从Datasets至PostProcess。
2. 目前不支持节点同名，同名节点将会代表同一个源文件夹。
3. 每个放置的节点都会在Mind Studio中mind文件所在目录下生成节点同名文件夹，如图3-54所示。

---

**图 3-54 放置节点后生成的文件夹**

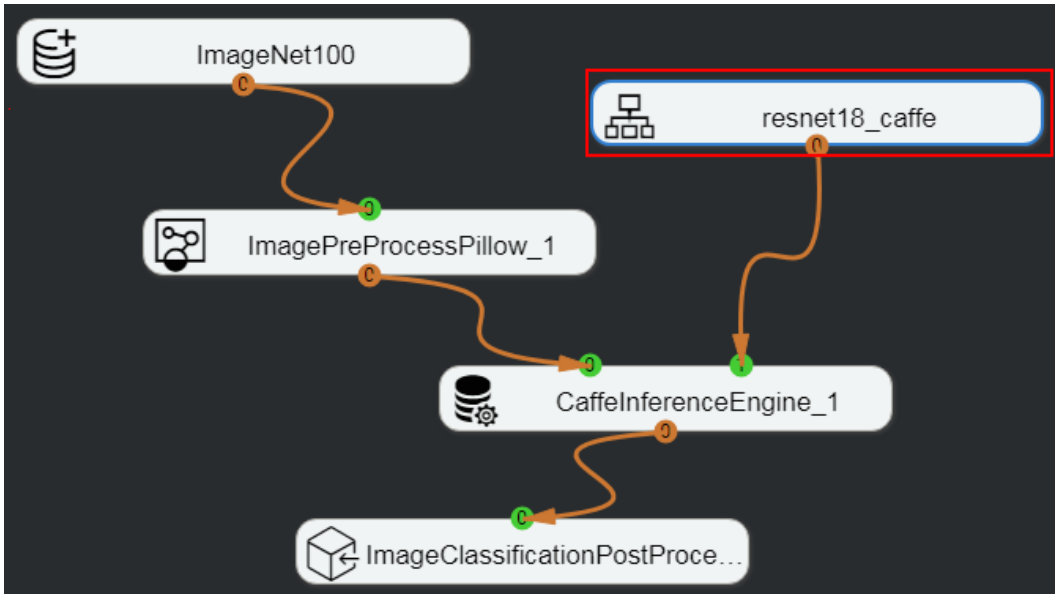



### 3.7.2.2 删除节点

节点的删除有两种方式：通过**delete**按键与右键菜单的方式。

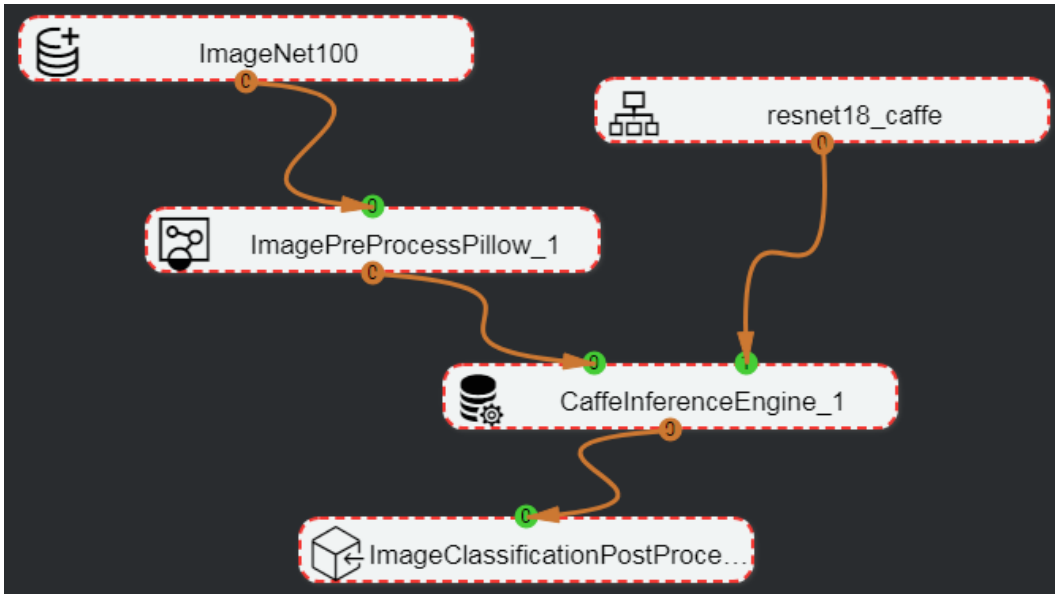
- 使用**delete**按键删除
  a. 删除单个节点：左键单击要删除的节点，被选中的节点会出现蓝色的边框，如图3-55所示，之后按下键盘功能区中的**delete**按键，即可删除该节点，若该节点上存在连线，则连线将被一并删除。

**图 3-55 删除单个节点**




b. 删除多个节点：Engine编排工具支持删除某个区域内的多个节点，具体操作如下：按住**ctrl**键，之后按住鼠标左键选中区域，被选中区域内的节点将会出现红色点状边框，如图3-56所示。

**图 3-56 删除多个节点**



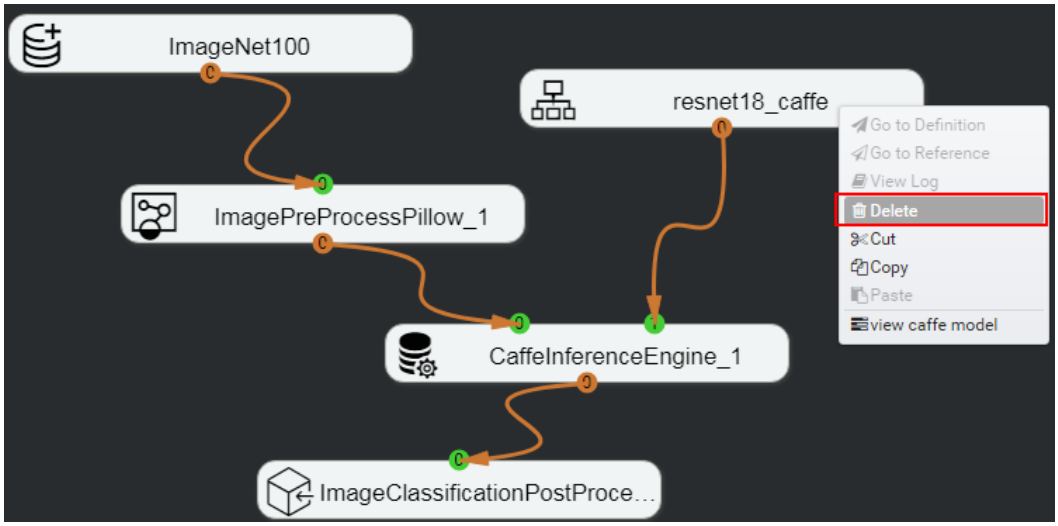

选中区域后松开**ctrl**键，此时使用**delete**按键即可删除该区域内的所有节点。鼠标单击绘制区域的空白部分即可取消选中区域。

- 右键菜单删除

  a. 删除单个节点：左键单击要删除的节点，选中该节点，再右键弹出菜单，选中**delete**选项即可删除该节点，如图3-57所示。

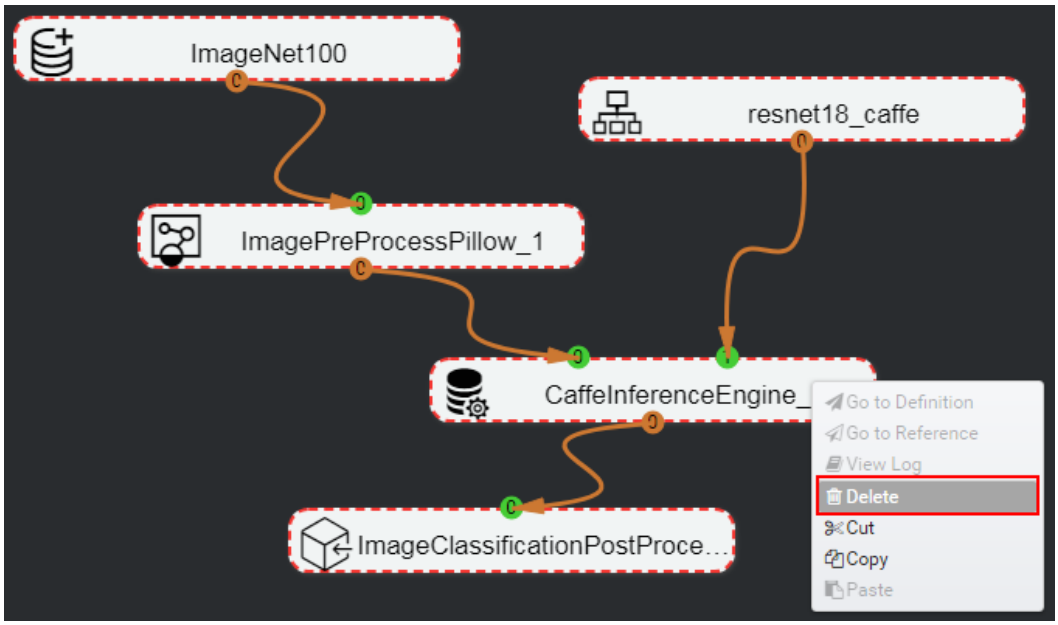
  

**图 3-57 删除单个节点**




b. 删除多个节点：按住ctrl键，同时按住鼠标左键选中区域，在被选中区域节点上右键弹出菜单，选中**delete**选项即可删除选中区域节点，如图3-58所示。

**图 3-58 删除多个节点**




### 3.7.2.3 节点的复制与粘贴

您可以通过对单个节点或多个节点执行复制操作。

选中单个节点或节点区域，在选中的节点上右键弹出菜单选择“copy”，即可完成对节点的复制，在绘图空白区域右键弹出菜单选择“paste”即可黏贴复制的节点，选中多个节点时，若选中的节点之间含有连线，则连线也会被相应的复制，具体操作如图3-59、图3-60、图3-61所示。

**图 3-59** 复制节点

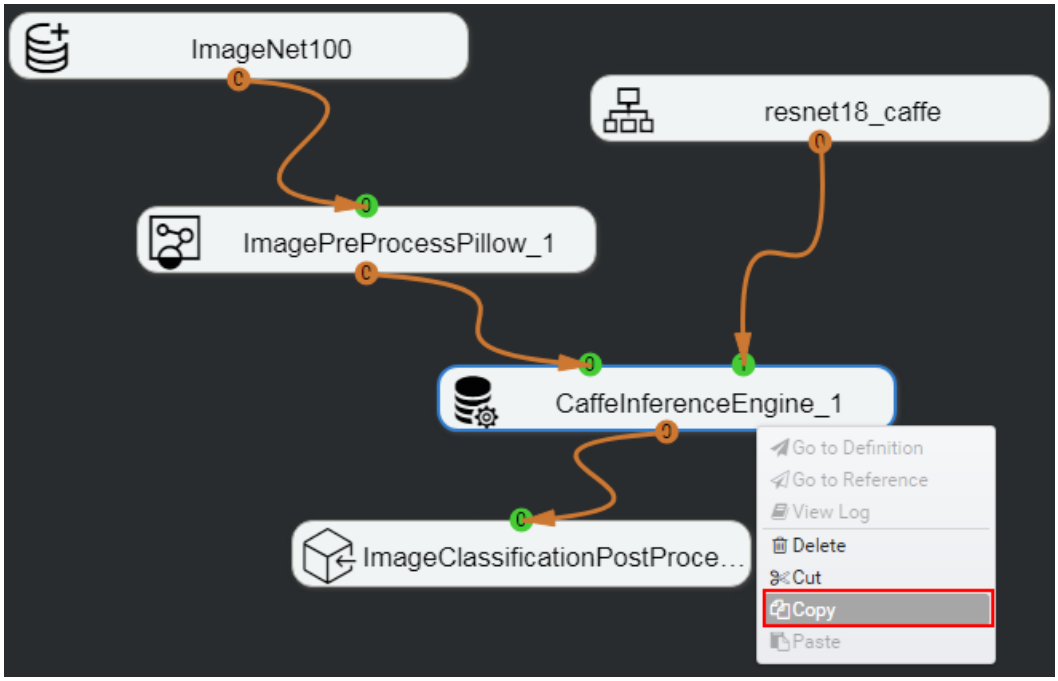



**图 3-60** 粘贴节点

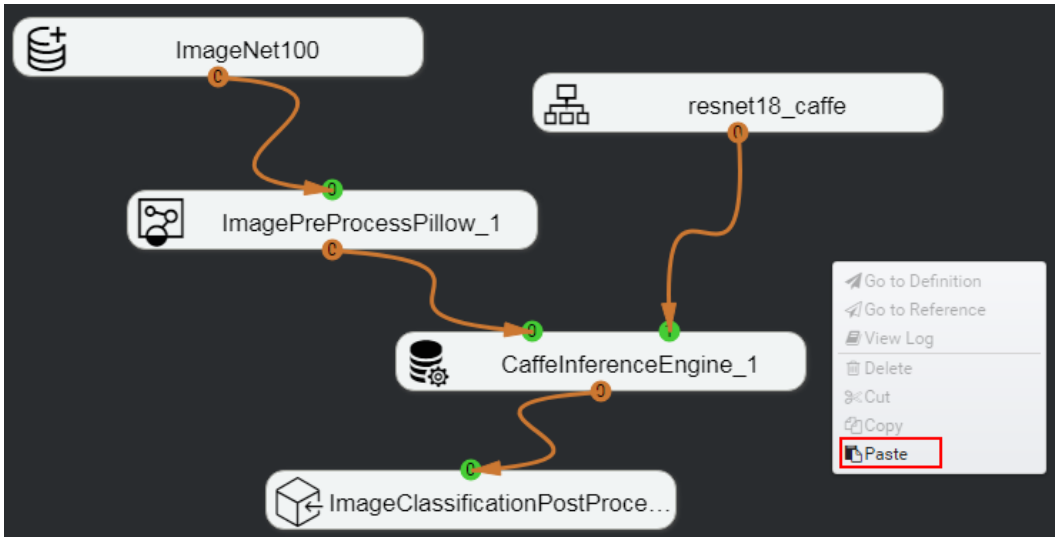
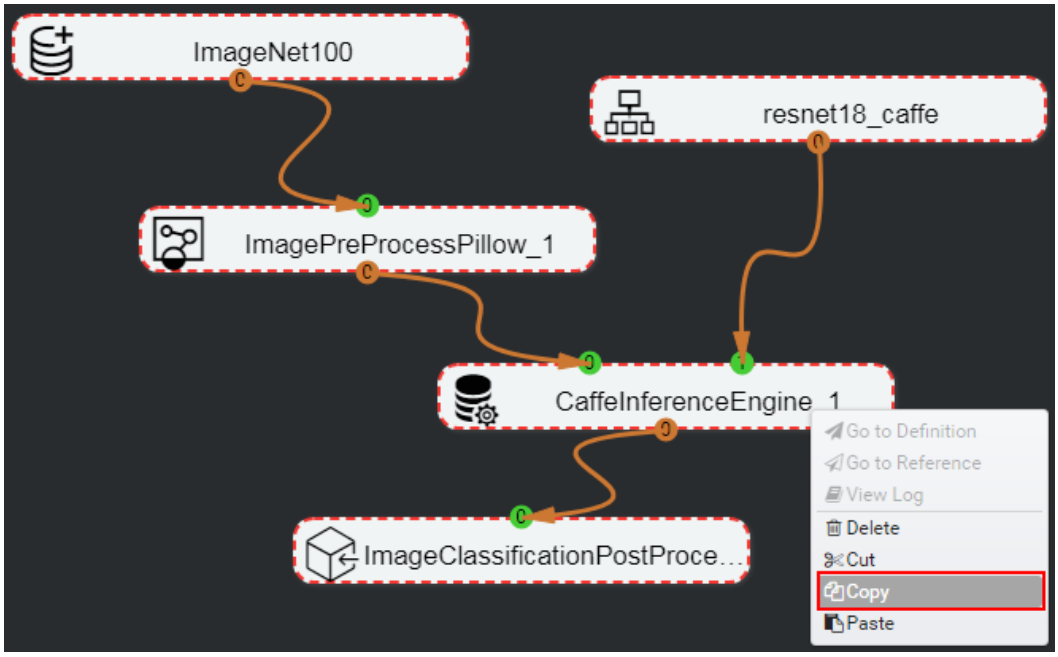


**图 3-61** 复制多个节点

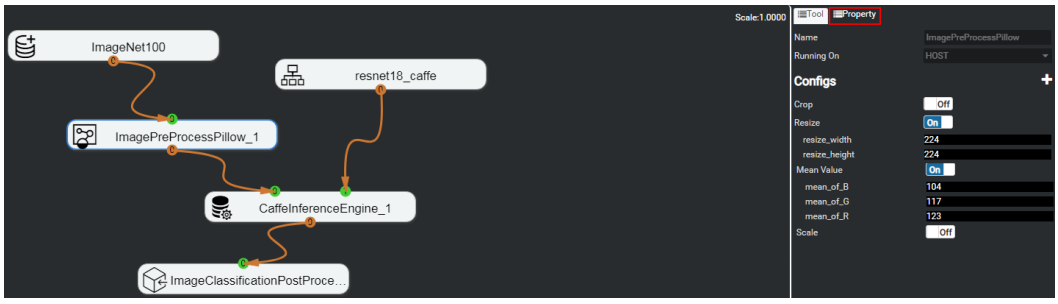



### 3.7.2.4 节点的属性设置

您可以通过运行该节点所需的参数在属性中进行配置。

右边区域框内的**property**栏为属性设置栏，选中节点即可对其属性进行设置，如图3-62所示。

**图 3-62** 设置节点属性




不同类型的节点拥有不同的属性，详见表3-13。

**表 3-13** 节点属性表

| 节点类型 | 属性名 | 描述 |
| --- | --- | --- |
| All | Name | 节点名 |
| Datasets | Path | 数据集存放路径。 |
| | Data Type | 数据类型 |
| | Include YUV420SP | 是否包含YUV420SP |

| 节点类型 | 属性名 | | 描述 |
| --- | --- | --- | --- |
| | Batch | | 一次处理的图片数量。 |
| | Run mode | | 处理图片的方式。 |
| Model | Model Path | | 模型文件存放路径。 |
| | DVPP Parameter Path | | DVPP参数存放路径。 |
| | Decryption | | 是否开启加密。 |
| PreProcess | Running On | | 节点的部署位置，**Host或Device**。 |
| | Crop（是否对图片进行裁剪） | From-Upper | 是否选择从上层Engine传入crop参数。 |
| | | point_x | 裁剪的起始位置横坐标，必须为偶数。 |
| | | point_y | 裁剪的起始位置纵坐标，必须为偶数。 |
| | | crop_width | 裁剪的宽度，必须为偶数，[1/32, 4]。 |
| | | crop_height | 裁剪的高度，必须为偶数，[1/32, 4]。 |
| | Resize（是否对图片进行缩放） | resize_width | 缩放后的宽度，必须为正数，最大支持4096。 |
| | | resizie_height | 缩放后的高度，必须为正数，最大支持4096。 |
| | Mean Value（图像预处理均值） | mean_of_B | 图片预处理B通道，推理过程mean与训练过程mean保持一致，最大支持255。 |
| | | mean_of_R | 图片预处理R通道，推理过程mean与训练过程mean保持一致，最大支持255。 |
| | | mean_of_G | 图片预处理G通道，推理过程mean与训练过程mean保持一致，最大支持255。 |
| | Scale | scale_value | 整比缩放，填写缩放系数，最大支持255。 |
| | Configs | - | 节点配置属性，每个配置含有**Name**和**Value**，可通过Configs后的加号进行添加。 |
| DeepLearningExecuteEngine | Running On | | 节点的部署位置，共有**Host**与**Device**两种。 |

| 节点类型 | 属性名 | | 描述 |
| --- | --- | --- | --- |
| | InputNum | | 表示引擎的输入点数量，默认值为2。 |
| | OutputNum | | 表示引擎的输出点数量，默认值为1。 |
| | Advance | Priority | 节点的优先级。 |
| | | ThreadNum | 节点的线程数 |
| | | ThreadPriority | 节点的线程优先级。 |
| | | QueueSize | 节点的队列长度。 |
| PostProcess | Running On | | 节点的部署位置，共有**Host**与**Device**两种。 |
| | OutputName | | 指定输出层名，分类网络默认prob，检测网络默认detection_out。 |
| | Advance | Priority | 节点的优先级。 |
| | | ThreadNum | 节点的线程数。 |
| | | ThreadPriority | 节点的线程优先级. |
| | | QueueSize | 节点的队列长度。 |
| | | SoName | 节点所需的所有动态库So文件的文件名列表，可通过**SoName**之后的加号进行添加。 |
| | | Configs | 节点配置属性，每个配置含有**Name**和**Value**，可通过Configs后的加号进行添加。 |

### 3.7.2.5 建立连接

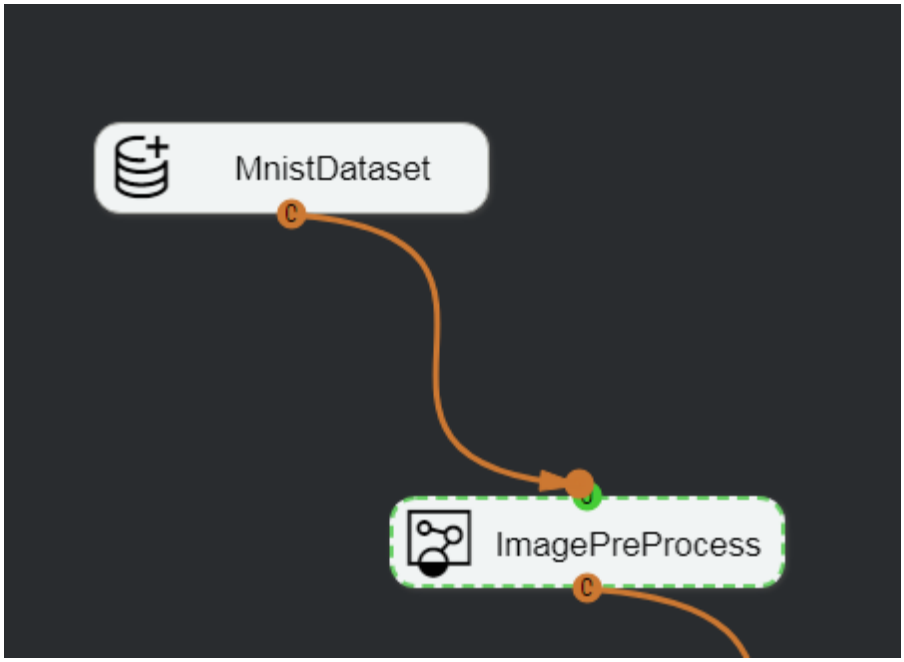
业务节点间的连接表示节点间数据的流向。

在设置好节点的基本属性之后即可开始建立节点的连接关系，每个节点上都带有输入输出端点。

- 橙色的实心端点代表输出端点，绿色端点代表输入端点，一条连线必须从一个节点的输出端点连接到另一个节点的输入端点。
- 鼠标左键按住输出端点，即可拉出连接线，将其放置到另一个节点的输入端点上即可完成一次连接，在拉出连接线时，可连接的节点将出现绿色虚线边框，如图3-63所示。

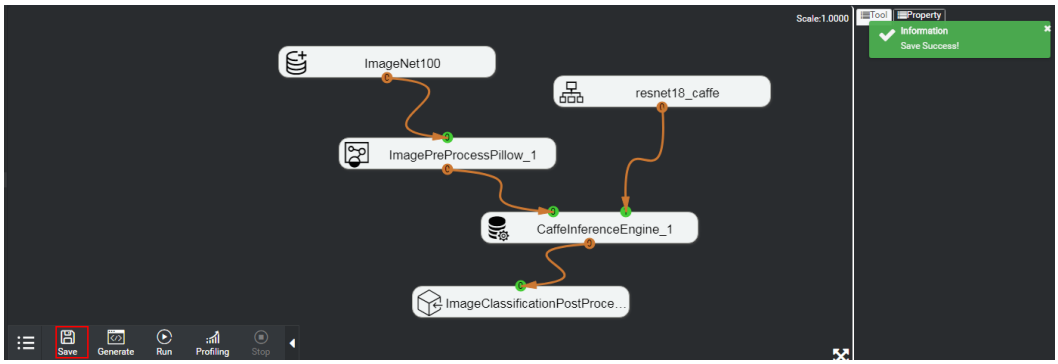
**图 3-63** 节点连接




### 3.7.2.6 保存节点

建立好节点的连线关系之后，点击绘制区域框内左下方的“Save”即可将流程图进行保存，保存成功在右上角会出现保存成功提示框，如图3-64所示。

**图 3-64** 保存连接好的节点

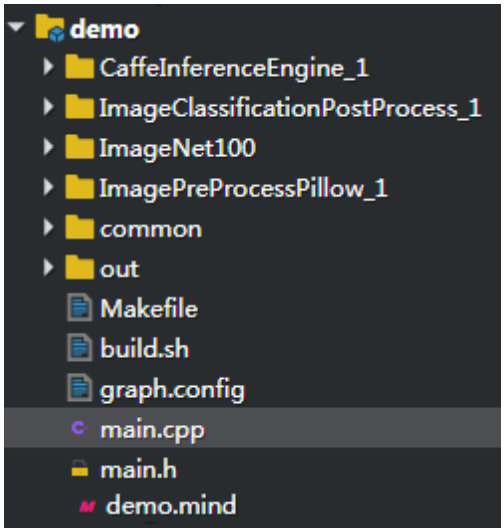



### 3.7.2.7 生成 cpp 文件

MindEngine流程保存之后，点击绘制区域框内左下方的“Generate”，**MindEngine**系统会根据Mind工程的配置生成.cpp文件，该**.cpp**文件将会显示在左边目录下，如图3-65所示。

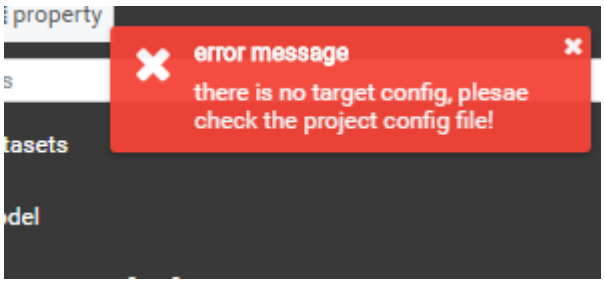
**图 3-65** 生成 cpp 文件




注意

若Mind工程缺少配置，则不会生成对应的cpp文件，且系统会报错，如图3-66所示，此时需要检查工程根路径下的.project文件是否存在或者文件内是否配置有target，target的配置共有三种：Local、ASIC、Atlas DK。

**图 3-66** 错误信息



# 4 离线模型转换

## 4.1 简介

为了让开发人员能够更容易使用训练过的模型，使用很少的代码构建一个机器学习的应用。Mind Studio提供了离线模型系统，实现了以下功能，方便用户快捷、方便地使用离线模型编写AI程序：

- 离线模型转换：支持将Caffe、Tensorflow等业界开源的神经网络模型转化为华为NPU芯片支持的网络模型。

  **说明**

  当前版本只支持Caffe、Tensorflow模型转换，Tensorflow部分功能不支持，暂不支持加密模型的导入。

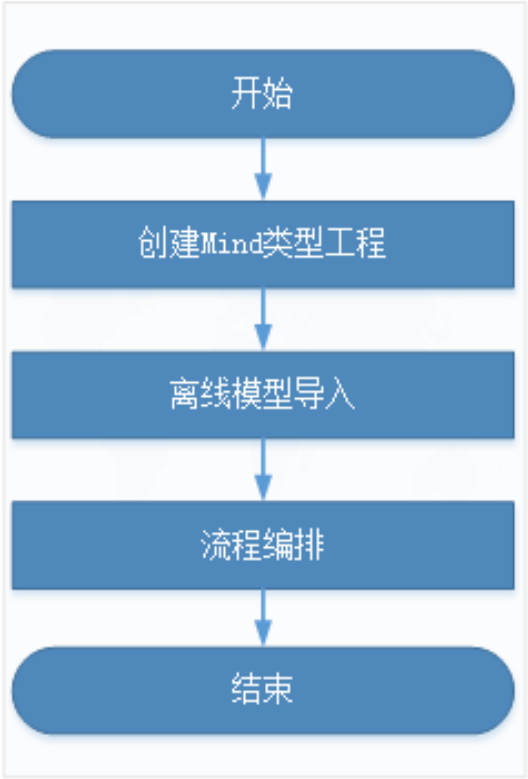
- 离线模型导入：支持将转化好的，未加密模型直接导入到Mind Studio工程中供开发者编码使用。
- 离线模型可视化：支持离线模型的网络结构查看，可以看到每一层网络的详细信息。

## 4.2 开发流程

本示例描述使用用户自行下载的开源网络模型，通过模型转化的方式导入Mind Studio工具，然后通过编排的方式实现AI程序开发的流程。总体开发流程如图4-1所示。

**图 4-1** 使用外部网络进行业务编排



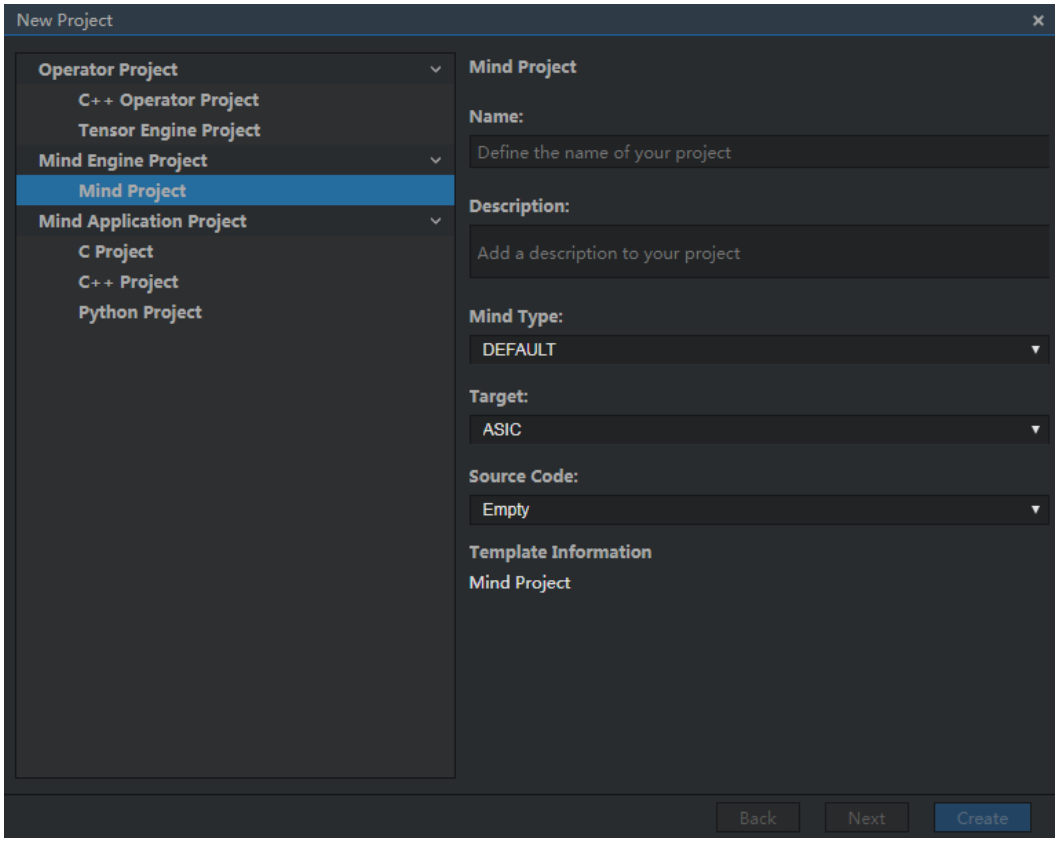

## 4.2.1 创建 Mind 工程

**步骤1** 依次选择“File > New > New Project”，弹出“New Project”窗口，创建一个新的工程。

**步骤2** 在“New Project”窗口中选中“Mind Engine Project > Mind Project”工程，显示窗口配置界面，如图4-2所示。

**图 4-2** 创建 Mind Engine 工程




**步骤3** 单击“Create”新建一个**Mind**工程，生成与工程名同名的“.mind”文件，例如*Demo.mind*。

**----结束**

## 4.2.2 离线模型导入

使用Mind Studio的离线模型导入功能，将外部网络模型转化为华为NPU芯片支持的网络模型。

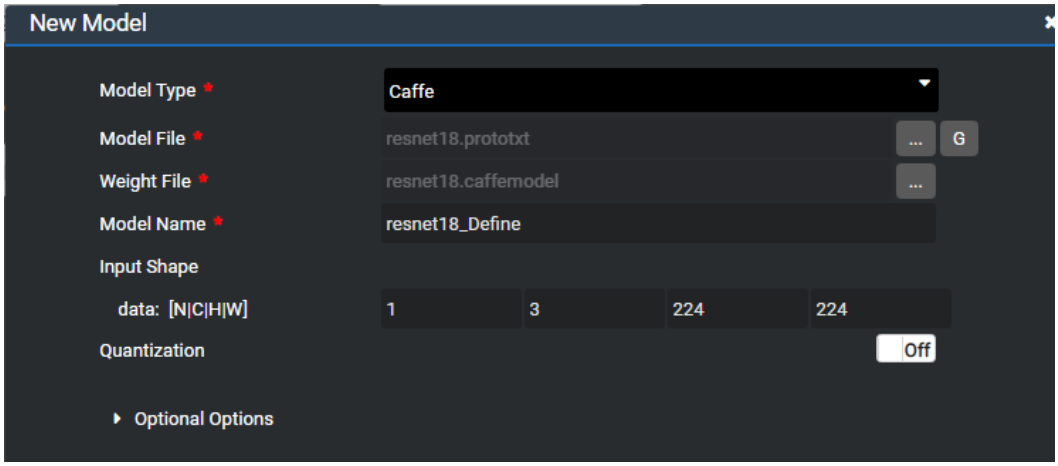
**步骤1** 在“Projects Explorer”窗口中双击*Demo.mind*文件打开编排画布。

**步骤2** 点击My Models右侧的  ，添加自定义模型组件，添加自定义模型组件。

**步骤3** 在弹出的New Model窗口中添加外部网络模型文件，并设置相关参数，如图4-3所示。

**图 4-3** 选择模型文件



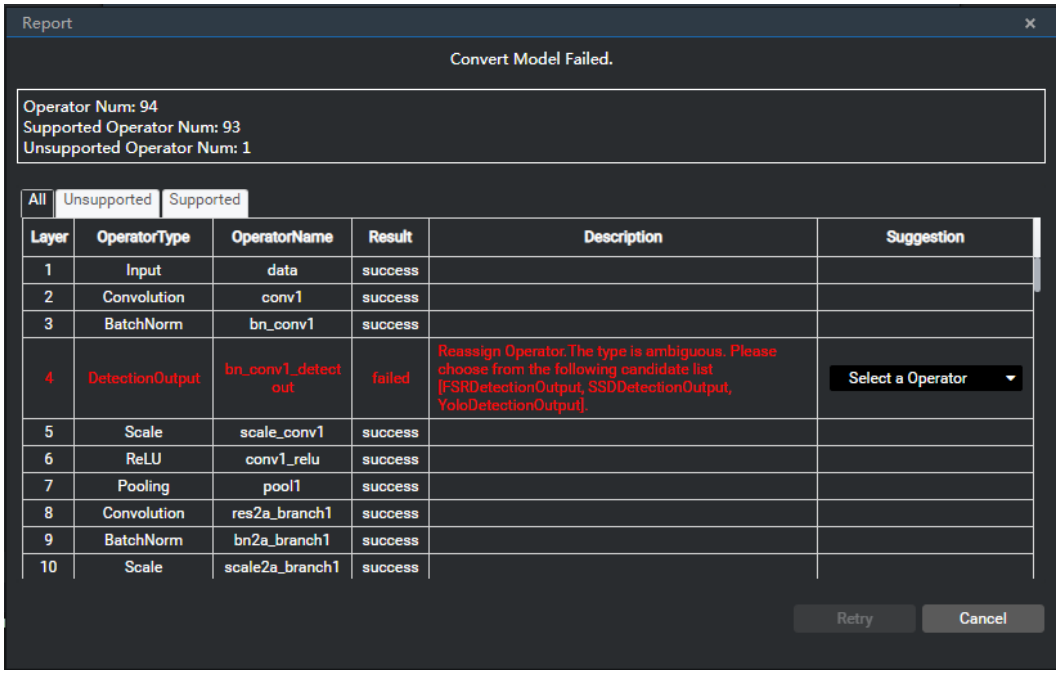

详细的参数描述请参见《Mind Studio基本操作》中的新增自定义模型组件。

**步骤4** 参数配置完成后，“ok”按钮变亮，单击“ok”进行模型转换。

- 如果转化成功，会展示转化成功页面，页面中包含模型在服务器端的路径和模型文件大小。
- 如果转化失败，会显示失败报告。

  如果失败类型仅是重命名算子，需要用户选择该类型对应的算子，如图4-4所示。

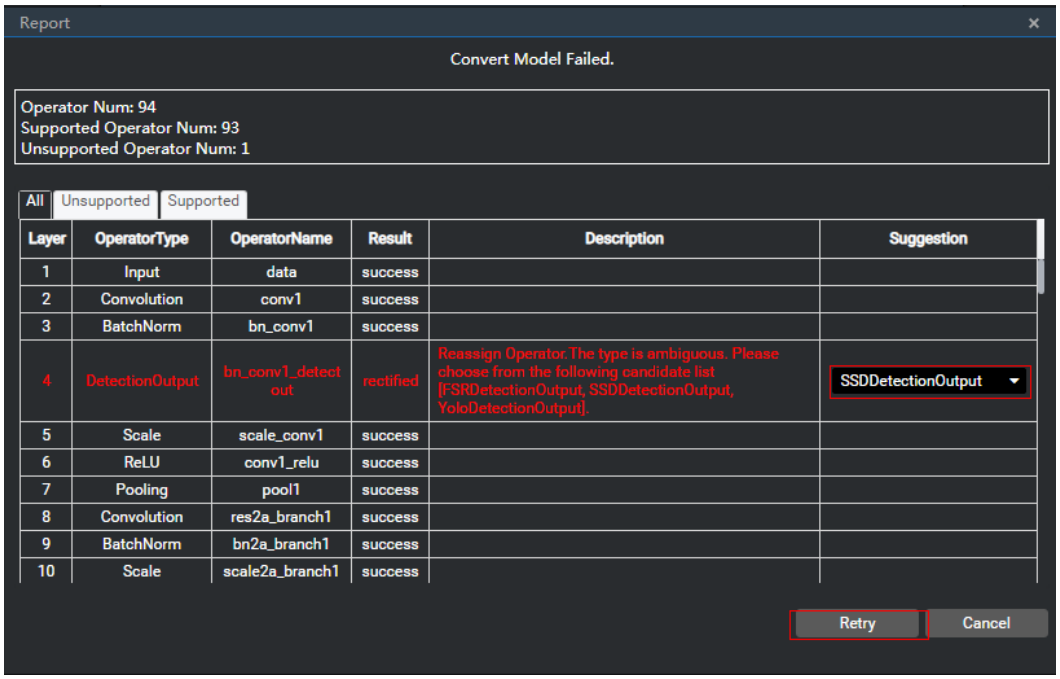
**图 4-4** 转化失败 Report




从下拉框选择一个算子后，单击“Retry”进行重试，如图4-5所示。

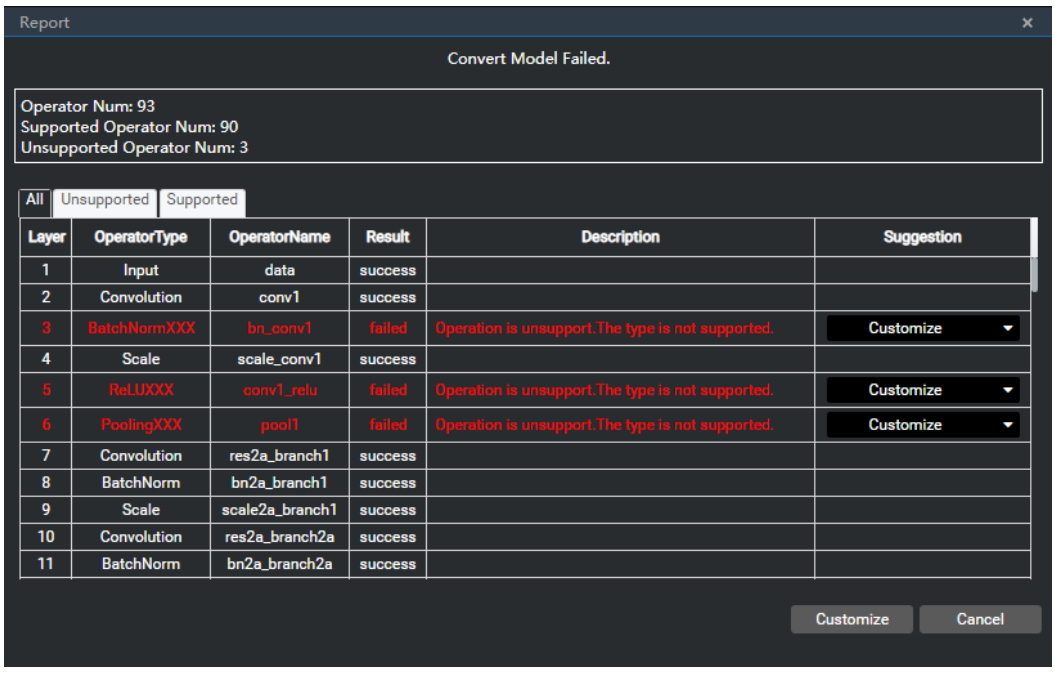
  

**图 4-5 重新选择算子**




如果失败类型包含不支持的算子，如图4-6所示。

**图 4-6 转化失败-包含不支持算子**

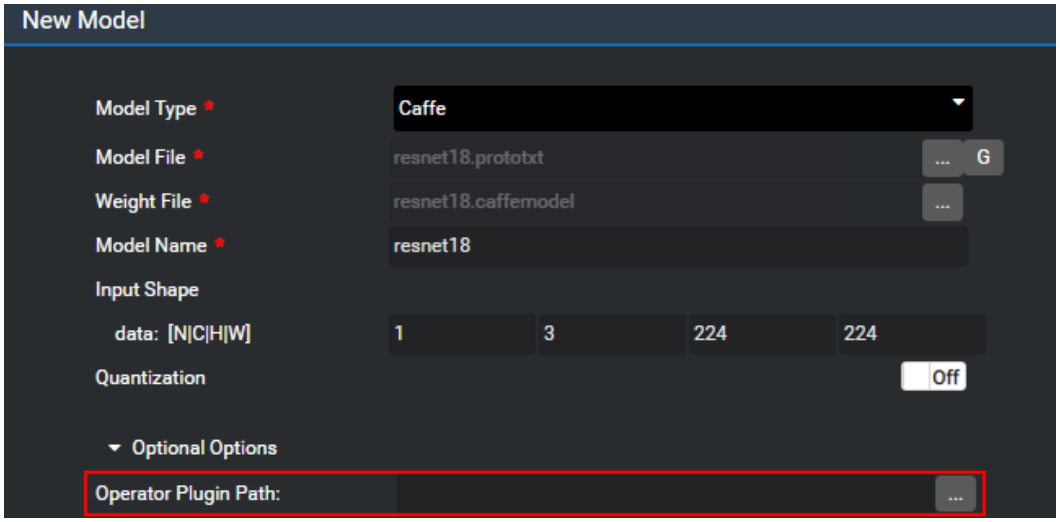



单击“Customize”对该类型的算子创建工程，进行自定义算子插件 开发，详细请参考《TE自定义算子开发指导》或《C++算子开发指导》。

自定义算子插件开发完成后，重新进行离线模型导入，导入时需要选择开发好的自定义算子插件，如图4-7所示。

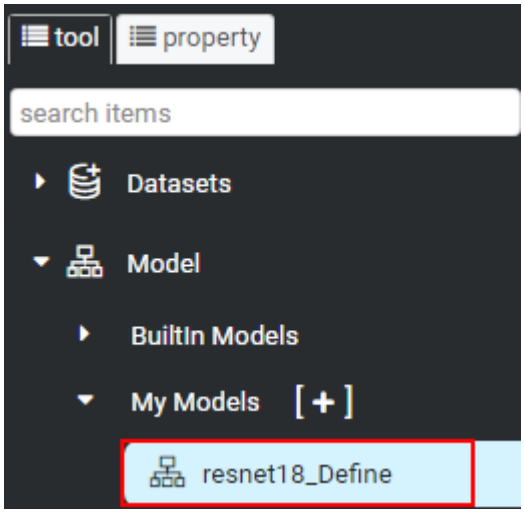
**图 4-7** 导入离线模型--选择自定义开发算子插件




单击“ok”进行转化。

**步骤5** 转化成功后，会在自定义模型组件中展示，供后续编排使用，如图4-8所示。

**图 4-8** 自定义模型组件




**----结束**

## 4.2.3 流程编排

使用自定义网络模型进行Engine编排开发，编排示例如图4-9所示。

**图 4-9** Engine 编排示例